Centro Universitario de la Defensa en la Escuela Naval Militar

TRABAJO FIN DE GRADO

Predicción de tipo de buque utilizando representaciones de trayectorias y técnicas de Inteligencia Artificial

Grado en Ingeniería Mecánica

ALUMNO: Jorge Roldán Calvo

DIRECTORES: Belén Barragáns Martínez

Pablo Sendín Raña

CURSO ACADÉMICO: 2023-2024

Universidade Vigo

Centro Universitario de la Defensa
en la Escuela Naval Militar

TRABAJO FIN DE GRADO

Predicción de tipo de buque utilizando representaciones de trayectorias y técnicas de Inteligencia Artificial

Grado en Ingeniería Mecánica

Intensificación en Tecnología Naval

Cuerpo General

Universidade Vigo

RESUMEN

Al analizar los flujos AIS que caracterizan a cada embarcación, se observa que un tercio de los mensajes presentan el campo “tipo de buque” sin cubrir. Este campo es de vital importancia para la detección de anomalías en el comportamiento de buques. Para abordar este inconveniente, se propuso en anteriores Trabajos Fin de Grado el uso de un algoritmo de clasificación alimentado por diferentes campos de los mensajes AIS tratando de encontrar el conjunto de características que mejor predice el tipo de embarcación (datos estáticos, dinámicos o de su zona de actividad).

En este TFG se plantea el empleo de la información dinámica del buque para generar trayectorias, a partir de las cuales se extraen hasta 67 nuevas características que alimentan el algoritmo de clasificación. Se entrena el modelo con un *dataset* de tráfico mundial de dos meses para clasificar los cinco tipos de embarcaciones con mayor presencia. En la fase de evaluación, se obtiene la predicción a partir de características exclusivamente basadas en trayectorias, con un grado de precisión que mejora el de algunas propuestas anteriores. Al combinar las nuevas características con parámetros estáticos del buque se mejora significativamente la calidad de las predicciones obtenidas empleando únicamente datos estáticos.

PALABRAS CLAVE

Inteligencia Artificial, Aprendizaje Automático, AIS, Tipo de buque, Trayectorias

AGRADECIMIENTOS

Me gustaría agradecer a todo el personal que ha contribuido en mi formación técnica, profesional y en valores; tanto a oficiales y mandos de la armada y los demás ejércitos, como a todos los profesores pertenecientes al Centro Universitario de la Defensa. Gracias por hacer de mí el Oficial que la Armada precisa.

Especial mención merecen mis dos tutores para este TFG, Belén y Pablo, que gracias a su experiencia y amor hacia oficio de docencia y hacia esta institución, han logrado motivarme día a día para realizar este apasionante a la par que complejo trabajo. Gracias por su dedicación.

A mis compañeros y amigos de la 424-154, con los que he estado conviviendo estos últimos cinco años que no cambiaría por nada del mundo. Gracias por convertiros en mis hermanos.

A mis amigos, que sin entender el mundo en el que me metí hace cinco años, han seguido manteniendo ese vínculo incondicional que espero que dure para siempre. Gracias por vuestra paciencia.

A mi novia, Leti, que ha sido mi compañera de aventuras en este último tramo de mi formación académica y que me motiva a ser mejor persona cada día, dándome su amor y su apoyo aún desde la distancia. Gracias por estar a mi lado.

A mi familia, que me sigue acompañando por el largo camino que llevo recorrido con orgullo y plena confianza en mis decisiones, siendo el faro que me guía en mis noches más oscuras. Sin vosotros, no sería la persona que soy hoy, gracias por dedicarme vuestra vida.

CONTENIDO

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

1 INTRODUCCIÓN Y OBJETIVOS

1.1 Contexto y motivación

"El mar es la principal vía de transporte de mercancías a nivel global, y es responsable de más del 90% de las cargas transportadas anualmente en el mundo" [1]. En enero de 2023, la flota mercante mundial estaba compuesta por 105.500 millones de embarcaciones de al menos 100 toneladas brutas, de las cuales 56.500 millones de buques tenían más de 1000 toneladas brutas [2]. Con el acelerado crecimiento del número de embarcaciones, el entorno del tráfico marítimo se ha vuelto más complejo y el control, la seguridad y legalidad en la mar se ven amenazados.

Desde 2004, la Organización Marítima Internacional (OMI) ha ordenado la instalación de equipos del sistema de identificación automática (*Automatic Identification System,* AIS) en barcos de pasajeros internacionales y barcos de 300 toneladas o más, para fortalecer la seguridad y la protección marítimas. Sin embargo, con el uso generalizado de AIS, la fiabilidad y la precisión de la información AIS se ha cuestionado en los últimos años, dado que parte de la información estática proporcionada por AIS es introducida directamente por la tripulación del barco y puede proporcionarse información incorrecta o falseada [3]. En [4], se revela que más del 80% de los accidentes en la mar son debidos a errores humanos y se hace un análisis de los conocimientos de los que operan los equipos AIS y de su manejo del sistema para un correcto y óptimo uso del sistema. El estudio extrae como principal conclusión que es necesario algún tipo de control ante estos errores humanos.

La voluminosa cantidad de datos históricos generados desde la implementación del sistema AIS y el crecimiento exponencial de las tecnologías de Inteligencia Artificial (IA) y del aprendizaje automático o *Machine Learning* (ML) hacen plantear al Ministerio de Defensa y, en particular, a la Armada la razón de ser de este marco de investigación. En la *"Estrategia de desarrollo, implantación y uso de la Inteligencia Artificial en el Ministerio de Defensa"* [5], se detalla explícitamente la necesidad del desarrollo de técnicas de IA para el "conocimiento y vigilancia del entorno en los ámbitos terrestre, marítimo, aeroespacial, ciberespacial y cognitivo. Análisis masivo de datos, información y desinformación". Ya en 2022 quedaba reflejado en las líneas generales de la Armada para 2022 [6] el interés por la explotación de esta nueva tecnología.

La línea de investigación a la que pertenece este TFG que surge para dar respuesta ante la necesidad planteada por la Armada al Centro Universitario de la Defensa en la Escuela Naval Militar es precisamente la consecuencia de los nuevos esfuerzos del Centro de Operaciones de Vigilancia de Acción Marítima (COVAM) en desarrollar procedimientos operativos que hagan uso de técnicas de IA para el conocimiento del entorno marítimo (CEM).

El COVAM tiene entre sus objetivos detectar anomalías en el entorno marítimo. Una anomalía en los patrones de comportamiento de los buques ocurre cuando las acciones de un barco en una zona no son coherentes con el tráfico general de ésta o con datos históricos del propio buque. La Armada, a través del COVAM, ha identificado un conjunto de anomalías de interés para la vigilancia marítima y ha creado un listado para su detección mediante Inteligencia Artificial, basándose en factores como el tipo de buque, la zona y la cinemática. Sin embargo, los datos AIS, que proporcionan información clave, a menudo son incompletos o erróneos debido a la falta de transmisión de campos obligatorios. El campo del tipo de buque cobra especial relevancia dándose la circunstancia de que aproximadamente un tercio de las tramas AIS carecen de dicha información [7]. La ausencia de este dato dificulta la identificación de anomalías, generándose así una nueva línea de trabajo que trata de predecir el valor de dicho campo a partir de otra información sí disponible, contribuyendo así a las necesidades del COVAM en el uso de Inteligencia Artificial para el CEM.

El empleo de imágenes satelitales para la clasificación de embarcaciones es uno de los enfoques para resolver este problema y conlleva ventajas sustanciales. Éstas permiten la identificación de buques que carecen de AIS o que han pausado su transmisión, estrategia habitual para evadir la detección [8]. El principal desafío de este enfoque es la resolución limitada de las imágenes satelitales, que puede dificultar la identificación de las embarcaciones de menor tamaño.

A la vista de estas circunstancias, es esencial desarrollar un reconocedor eficaz de tipos de barcos para abordar la falta de datos en el Sistema de Identificación Automática. Aunque existen diferentes propuestas como el uso de Redes Neuronales Convolucionales (CNN) o Redes Neuronales de Grafo (GNN) para la clasificación basada en datos de AIS, un modelo de aprendizaje profundo presenta desafíos de verificación y demanda una considerable cantidad de datos de entrenamiento. En este marco de trabajo, se propone la extracción de características significativas a través de diversas propiedades intrínsecas al sistema AIS. Este enfoque, aunque requiere altos esfuerzos de preprocesamiento de datos, ha demostrado ser más eficiente en cuanto a tiempos de trabajo y calidad de las predicciones ante un modelo de aprendizaje profundo [3].

La línea de investigación se inicia con [7], demostrando que se puede predecir el tipo de barco a partir de otros campos de los mensajes AIS, utilizando el algoritmo *Random Forest*. En [9], se profundizó en esta propuesta, validando los resultados obtenidos anteriormente. En ambos se concluyó que los datos estáticos adquieren gran relevancia para la predicción. Como medida innovadora, en [10] se exploró cómo el empleo de información relacionada con el área de actividad del buque podría mejorar la predicción del tipo de barco, concluyendo que esta información contribuye positivamente a la mejora de la predicción, beneficiando especialmente a ciertos tipos de barcos.

Enlazando con los trabajos previos, en este TFG se propone el análisis del comportamiento de los buques estudiando sus trayectorias, pues parece que puede desvelar parámetros valiosos para mejorar la calidad de la predicción del algoritmo.

1.2 Objetivos

Para este TFG, se han planteado distintos objetivos para poder contribuir a la línea de investigación. El principal propósito de este trabajo consiste en desarrollar un modelo que utilice técnicas de inteligencia artificial con el fin de predecir el tipo de buque a partir de los datos AIS dinámicos que permiten generar trayectorias.

Para lograr esto, se propone el estudio y análisis de los datos estáticos que sienten una base robusta a la que se le pueda agregar un nuevo conjunto de características basadas en trayectorias. Se procederá a construir el modelo buscando optimizar su rendimiento mediante ajustes en los parámetros del algoritmo y utilizando diversas técnicas de preprocesamiento de datos manteniendo predicciones similares a las previas realizadas con los datos estáticos.

Además, es un objetivo principal de este trabajo de fin de grado investigar la validez de los datos dinámicos a través de la generación de trayectorias y la extracción de características diferenciadoras para contribuir a la mejora de la clasificación de los diferentes tipos de buques. Para ello, será necesario estudiarlos por sí solos y en conjunción con los estáticos, de manera que habrá que hacer un pretratamiento de estos últimos para cohesionarlos con los datos de trayectorias. Por último, se pretende definir diferentes experimentos para validar el modelo.

1.3 Estructura de la memoria

Después de establecer el marco y los objetivos de este trabajo, se procede a explicar la estructura de la memoria en el presente capítulo. El documento se divide en cinco capítulos, seguidos por la bibliografía y los anexos. Posteriormente la memoria se divide en:

- Estado del Arte: Se realiza una revisión del marco teórico que abarca este trabajo, incluyendo:
  - Introducción al concepto de inteligencia artificial y sus aplicaciones para un contexto militar.
  - Explicación del aprendizaje automático y sus tipos, así como una revisión de los algoritmos de inteligencia artificial populares y sus funcionamientos.
  - Introducción al Conocimiento del Entorno Marítimo (CEM), el papel del COVAM y la importancia de las tecnologías de inteligencia artificial para su desarrollo. Además, se presenta el sistema AIS como fuente de datos para el trabajo y se revisan investigaciones previas relevantes.
  - Revisión bibliográfica. Se exponen los estudios previos que guardan mayor relación con este proyecto y han sentado las bases para la elaboración dl mismo.
- Desarrollo del TFG: Se describen las herramientas y datos utilizados para cumplir los objetivos, justificando el uso de *Random Forest* como algoritmo para el modelo. Se detallan los pasos desde el preprocesamiento de datos éstaticos, hasta la generación de un conjunto de parámetros robusto, al que se le pueda añadir más información. Posteriormente se inicia el estudio de los comportamientos cinemáticos de los buques utilizando librerías para representar sus trayectorias, discretizarlas y extraer características de las mismas. Finalmente se explica cómo se preparan todos los parámetros para realizar distintos experimentos con los mismos.
- Resultados del TFG: Se exponen los resultados de los experimentos que surgen de alimentar con los parámetros al algoritmo y se analizan para determinar los parámetros más característicos extraídos de las trayectorias.
- Conclusiones y Líneas Futuras: Se discuten los resultados obtenidos y se proponen posibles líneas futuras para mejorar o complementar el trabajo, avanzando en la consecución de los objetivos de esta línea de investigación.

Finalmente, se incluye la bibliografía con las fuentes utilizadas a lo largo del trabajo y una serie de anexos que apoyan el desarrollo del trabajo para una mejor comprensión del mismo.

2 ESTADO DEL ARTE

En este capítulo se abordarán el concepto de Inteligencia Artificial (IA), analizando sus definiciones y sus principales aplicaciones, especialmente, las enfocadas al ámbito militar, haciendo hincapié en las líneas de acción que se proponen nacionalmente para su explotación efectiva. Posteriormente, se explicará una de las ramas de la IA, el aprendizaje automático, que será la herramienta central de este trabajo. Se hará una descripción de todas las subdisciplinas de esta rama de la IA, finalizando con una breve reseña del aprendizaje profundo y de otras ramas que comprenden la IA, que, aunque se encuentran fuera del ámbito de estudio, merecen su mención dada la estrecha relación que guardan con el aprendizaje automático. A continuación, se hará un estudio de los diferentes algoritmos de aprendizaje automático que se utilizarán para esta investigación.

En línea con el uso de la IA en la Armada, se procederá con la explicación del concepto de conciencia situacional marítima (*Maritime Situational Awareness*, MSA) y la labor del COVAM para lograr los objetivos que propone la Armada para el conocimiento del entorno marítimo (CEM). Finalizando con esta materia, se explicarán los datos AIS con los que se trabajará para esta investigación (procedentes del propio COVAM).

Por último, se realizará una revisión bibliográfica de aquellos estudios que guarden cierta relación con la materia a investigar, haciendo hincapié en los TFG que preceden a éste y que forman parte de la misma línea de investigación.

2.1 Inteligencia Artificial

2.1.1 Definición

Cuando se abordan los aparentemente modernos e innovadores temas como la Inteligencia Artificial, suelen invadirnos pensamientos distópicos en los que las máquinas se convierten en seres humanos perfectos con una capacidad de procesamiento y almacenamiento de datos ilimitada. Nada más lejos de la realidad, la Inteligencia Artificial está presente en la mayor parte de los procesamientos tecnológicos y es imprescindible abordar sus múltiples definiciones para comprender el potencial que esconde esta disciplina y el reto que supone crear un proceso que se asemeje a la inteligencia humana.

La Inteligencia Artificial se puede definir, a grandes rasgos, como el desarrollo de sistemas informáticos capaces de realizar tareas que normalmente requieren de cualidades humanas, como el aprendizaje, el razonamiento, la percepción y la toma de decisiones. Las definiciones pueden variar según organismos, pero suelen incluir la capacidad de aprender de los datos y de adaptarse a nuevas situaciones. Según la RAE, por ejemplo, la Inteligencia Artificial es la *"disciplina científica que se ocupa de crear programas informáticos que ejecutan operaciones comparables a las que realiza la*

mente humana, como el aprendizaje o el razonamiento lógico" [11]. El término fue acuñado por primera vez por John McCarthy, miembro del Departamento de Ciencias Informáticas de la Universidad de Stanford, durante la conferencia de Dartmouth. McCarthy se refirió a este concepto como la *"ciencia e ingenio de crear máquinas inteligentes, en particular, programas de cómputo inteligentes"* [12] .

En estas definiciones se asume el concepto de "inteligencia" como trivialmente definido y parcelado. Sin embargo, las definiciones que esta palabra ha ido adquiriendo a lo largo de la historia se han diferenciado en dos ramas: el enfoque racional y el humano. Debe aclararse que contrastar estos dos enfoques no implica que los humanos sean "irracionales", simplemente que no actúan perfectamente. De hecho, existen numerosos estudios de psicología que catalogan los sesgos cognitivos y los errores sistemáticos que los seres humanos cometemos como consecuencia de nuestra naturaleza [13]. Es por ello por lo que el término de Inteligencia Artificial sigue siendo interpretable y, dependiendo de su objetivo, habrá que adaptar el diseño de la IA a la definición más conveniente, pudiendo crear una IA que se asemeje a un humano o una más racional que ejecute operaciones de manera sistemática y precisa.

La Inteligencia Artificial se ha integrado en numerosos aspectos de la vida cotidiana, mostrando su impacto en diversas áreas. En el ámbito comercial la IA optimiza las compras en línea, personaliza la publicidad y facilita la gestión de inventarios. En la web mejora los motores de búsqueda para ofrecer resultados más relevantes, mientras que, en dispositivos móviles, asistentes virtuales basados en IA proporcionan respuestas personalizadas. Además, la IA desempeña un papel crucial en traducciones automáticas, asegura la eficiencia energética en hogares y ciudades inteligentes, impulsa funciones de seguridad en vehículos y contribuye a la ciberseguridad. En contextos de salud, la IA facilita análisis médicos y diagnósticos, y en la agricultura, optimiza la producción alimentaria. Asimismo, juega un papel clave en la administración pública al prever desastres naturales y reducir sus impactos. En resumen, la IA ha permeado en diversas esferas, transformando la forma en que vivimos e interactuamos con la tecnología [14].

El Ministerio de Defensa (MDEF) aprueba en junio de 2023 la *"Estrategia de desarrollo, implantación y uso de la Inteligencia Artificial en el Ministerio de Defensa"* [5] que marcará la línea a seguir para "aprovechar el potencial" de esta tecnología y explotar su "uso militar potencial" para no quedarse atrás [15]. En dicha estrategia, el Ministerio de Defensa ya define la IA en base a sus necesidades:

> *"Sistemas de software, y posiblemente también de hardware, diseñados por humanos que, ante un objetivo complejo, actúan en la dimensión física o digital percibiendo su entorno, a través de la adquisición e interpretación de datos estructurados o no estructurados, razonando sobre el conocimiento, procesando la información derivada de estos datos y decidiendo las mejores acciones para lograr el objetivo dado. Los sistemas de IA pueden usar reglas simbólicas o aprender un modelo numérico, y también pueden adaptar su comportamiento al analizar cómo el medio ambiente se ve afectado por sus acciones previas".*

Por tanto, en materia de Defensa, la IA es una herramienta que debe ser diseñada con un determinado objetivo a través de la adquisición de datos obtenidos por el mismo sistema o por fuentes propias o externas. En general, a efectos del Ministerio de Defensa y específicamente de esta línea de investigación, se precisará de una IA racional, sistemática y capaz de analizar grandes volúmenes de datos.

2.2 Aplicaciones militares de la Inteligencia Artificial

En el ámbito de las aplicaciones militares, la Inteligencia Artificial emerge como un componente estratégico de vital importancia, ofreciendo una amplia gama de capacidades que redefinen la naturaleza misma de la guerra contemporánea. El Ministerio de Defensa plasma su preocupación en [5], advirtiendo que *"la IA está modificando el entorno global de Seguridad y Defensa, ofreciendo una oportunidad sin precedentes para fortalecer la ventaja tecnológica del MDEF, pero también aumenta los riesgos y amenazas a los que se debe hacer frente."* El MDEF considera que *"esta tecnología afecta de modo transversal a todas las actividades militares desarrolladas por las Fuerzas Armadas en apoyo de sus tareas principales, defensa colectiva, gestión de crisis y seguridad cooperativa."*

Su implementación abarca desde la optimización de sistemas logísticos y la gestión eficiente de recursos hasta el desarrollo de sistemas autónomos capaces de realizar operaciones tácticas con una gran precisión y velocidad. La IA permite la interpretación instantánea de vastos conjuntos de datos, facilitando así el mando y control en entornos complejos y dinámicos. Este paradigma transformador no solo amplifica la capacidad de respuesta militar, sino que también plantea cuestionamientos éticos sobre el uso prudente de dichas tecnologías en el contexto de conflictos armados.

2.2.1 IA para el Ministerio de Defensa

La visión estratégica de la IA en el MDEF proyecta, a través de [5], una integración exhaustiva de esta tecnología en las Fuerzas Armadas (FAS). Su implementación pretende ser identificable, orientada a potenciar la eficacia en las misiones y con intención de extenderse a todos los niveles de operación. A la vez que se pretende explotar el potencial militar de la IA, se trata de adoptar medidas para defenderse del uso de ésta por parte de posibles adversarios. Paralelamente, se promueve el desarrollo del talento especializado en IA, respaldados por una infraestructura de gestión y explotación de datos obtenidos a partir de colaboraciones con organismos privados y públicos. El MDEF pretende actuar de acuerdo con la legislación nacional e internacional en materia de IA y seguir los principios éticos que rigen el uso de la IA, enfocándose en áreas de aplicabilidad predefinidas.

Los pilares en los que se sustentará la aplicación estratégica de la Inteligencia Artificial en el Ministerio de Defensa serán: la gestión del dato, demandando la digitalización y acceso a grandes volúmenes de datos provenientes de fuentes que aseguren su calidad, seguridad y trazabilidad; la infraestructura para la IA, integrada en la Infraestructura Integral de Información para la Defensa (I3D); la investigación, desarrollo e innovación, abordando los desafíos éticos y tecnológicos asociados con la aplicación de la IA en la Defensa; y la formación, captación y retención del talento.

Para parcelar el extenso campo de estudio de la IA y de acuerdo con las líneas de acción basadas en los pilares mencionados anteriormente, el MDEF ha establecido casos de uso inspirados en las directrices de Organizaciones Internacionales de Seguridad y Defensa encuadrados en el marco del Proceso de Planeamiento de la Defensa, bajo la supervisión del Jefe de Estado Mayor de la Defensa (JEMAD). Incluyen áreas como movilidad militar, inteligencia, guerra electrónica, autonomía en sistemas no tripulados, apoyo logístico, conocimiento y vigilancia del entorno, ciberdefensa, apoyo a la toma de decisiones, análisis geoespacial, meteorológico y oceanográfico, gestión de la información y la infraestructura, así como gestión del talento y formación. Sin embargo, el MDEF no excluye la posibilidad de incorporar otras áreas de interés en el futuro. La línea de investigación de este TFG se encuadra en el área de conocimiento y vigilancia del entorno marítimo.

Por último, para garantizar la fiabilidad de la Inteligencia Artificial en el cumplimiento de las misiones del MDEF, el Plan Estratégico contempla los principios fundamentales en los que se debe basar su desarrollo, implementación y uso. Estos principios incluyen, entre otros, la conformidad con el derecho nacional e internacional, la supervisión humana para la atribución de responsabilidades, la transparencia en las aplicaciones de IA, la realización continua de pruebas de seguridad, la posibilidad de desconexión ante comportamientos no deseados y el respeto a la privacidad.

En resumen, el MDEF es consciente del nuevo cambio de paradigma en cuanto al uso de la IA en las FAS y demuestra su preparación e iniciativa para la incorporación de estas tecnologías promulgando la estrategia que se ha desglosado a lo largo de este subapartado.

2.2.2 Gemelo Digital y mantenimiento predictivo en la Armada

Tras la elaboración del nuevo Plan de Acción del EMA que incluía un nuevo Concepto de Apoyo Logístico (CAL), aprobado en julio de 2017, se inició una fase de transformación del apoyo logístico en la Armada. En abril de 2018, Navantia propuso tres niveles de alcance para el proyecto de la fragata F-110: la Maqueta Digital Avanzada (MDA), el Gemelo Digital Básico (GDB) y el Gemelo Digital Avanzado (GDA). En términos generales, la MDA es la base que incluye la estructura y modelización detallada del buque, el GDB agrega funcionalidades para mejorar el sostenimiento y aspectos operativos, incluyendo algunas técnicas de IA, y, finalmente, el GDA se enfoca en la simulación avanzada, análisis predictivo y modelos de Inteligencia Artificial para la toma de decisiones.

Estas propuestas, asociadas a una estimación presupuestaria, buscaban implementar tecnologías de la Industria 4.0 en el diseño y operación de los buques. La selección del alcance del Gemelo Digital Básico para la fragata F-110 se basó en la priorización de aspectos prácticos de interés para la Armada, con ciertas funciones de sostenimiento, estableciendo las bases para futuros desarrollos tecnológicos.

El Gemelo Digital que está implementando la Armada en sus unidades representa de manera virtual un buque real, enriqueciendo la MDA con datos en tiempo real de su estado de mantenimiento y con predicciones basadas en modelos de Inteligencia Artificial. Este sistema avanzado recopila información de los sensores de los equipos a bordo, proporcionando una visión dinámica y detallada de su funcionamiento, estado y disponibilidad en el tiempo. El Gemelo Digital permite un análisis en tiempo real desde el Centro de Supervisión y Análisis de Datos de la Armada (CESADAR) a través de la plataforma ATAVIA, que incluye un módulo de sostenimiento predictivo basado en Redes Neuronales [16].

Además, el Gemelo Digital incorporará un software de Mantenimiento Predictivo (PdM) para predecir fallos en equipos cuyo funcionamiento se centra en técnicas de IA. Primero, los datos de los sensores se limpian y procesan para eliminar fallos y errores, y luego se almacenan en una base de datos. Después, los datos se normalizan y se introducen en un módulo de predicción para prever el estado futuro del equipo. El “mantenimiento inteligente” [17] busca, a través de estos programas, la detección de anomalías y la anticipación de fallos para mejorar la seguridad operativa, y disminuir los costes de mantenimiento. Es por ello por lo que la integración del Gemelo Digital en la Armada mejorará la eficiencia operativa, la planificación estratégica y la toma de decisiones en un contexto en el que la información y los datos juegan un papel determinante.

2.2.3 Guerra de Minas y Vehículos Submarinos Autónomos

Los vehículos submarinos autónomos (*Autonomous Underwater Vehicle*, AUV) son vehículos propulsados por baterías eléctricas o células de combustible, y pueden llevar una variedad de sensores para diferentes misiones submarinas. En contraste con los Vehículos Operados Remotamente (*Remotely Operated Vehicle*, ROV), los AUV son controlados por un ordenador interno, operan de forma autónoma sin necesidad de una conexión física con una plataforma de control, mientras que los ROV son controlados por operadores humanos a través de un cable umbilical. Los AUV ya presentan por sí solos técnicas de Inteligencia Artificial en el control de los mismos, permitiéndoles tomar decisiones al instante, interpretar datos ambientales y ajustar su comportamiento según las condiciones del entorno submarino. Por ello, la IA representa un importante avance para el uso de los AUV, mejorando la eficiencia y precisión de las operaciones submarinas y permitiéndoles recopilar datos relevantes para su posterior análisis [18].

En cuanto a las aplicaciones de los AUV, estos vehículos tienen un amplio espectro de usos. Se emplean en la evaluación ambiental en operaciones navieras, la vigilancia de puertos para detectar amenazas y la exploración detallada de fondos marinos. Además, se utilizan en la industria del

petróleo y gas, en operaciones de salvamento marítimo, en la caza de minas, en la hidrografía y en la arqueología submarina.

La segunda aplicación de la IA viene relacionada con una de las aplicaciones de los AUV: la caza de minas. Las minas submarinas representan una amenaza significativa para las embarcaciones marítimas, utilizándose para dirigir el movimiento o denegar el libre tránsito. Las medidas contra minas (MCM) se centran en localizar y neutralizar las minas para garantizar la libertad de movimiento. Como se ha detallado anteriormente, la búsqueda de minas se realiza cada vez más con vehículos submarinos autónomos equipados con sonar de apertura sintética (SAS), que proporciona imágenes acústicas de alta resolución del lecho marino. Dado que los AUV recopilan grandes cantidades de imágenes SAS, la clasificación automática de objetivos es de gran utilidad para la detección de minas [19].

El alto rendimiento de las redes neuronales profundas (*Deep Neural Network*, DNN) para la clasificación de imágenes y su rápido avance en los últimos años ha suscitado interés en muchas áreas de defensa, siendo la detección, identificación y clasificación de minas una de ellas. Por ejemplo, en [20] se describe cómo se colocaron diferentes objetos (incluyendo minas simuladas, *mine like objects*, rocas y otros objetos elaborados por el hombre) en el lecho marino en varias ubicaciones geográficas y se utilizó un AUV para inspeccionar el lecho marino con un SAS. Los resultados tras entrenar a diversos algoritmos mostraron que la DNN tiene un rendimiento significativamente superior, con una mayor probabilidad de detección de formas de minas y tasas de falsa alarma más bajas en comparación con un clasificador de objetivos tradicional. De manera similar, [21] describe cómo generar imágenes sintéticas de SAS de objetos con forma de cilindro y varios paisajes marinos que pueden ser utilizados para entrenar una DNN y creando así un ciclo de entrenamiento [19].

La combinación entre detección, clasificación y guiado de los AUV todavía sigue investigándose, aunque los acelerados avances apuntan hacia un modelo en el que el AUV es capaz de, a través de distintas técnicas de IA, navegar de manera autónoma, detectar y clasificar minas u otros objetos potencialmente peligrosos y, en última instancia, destruirlas o neutralizarlas.

2.2.4 Ciberseguridad

En el contexto de la ciberseguridad, la detección de intrusiones emerge como un componente crítico para preservar la integridad, confidencialidad y disponibilidad de la información frente a actividades maliciosas en el dominio del ciberespacio. La funcionalidad esencial de los sistemas de detección de intrusiones (*Intrusion Detection System*, IDS) radica en la capacidad para clasificar el tráfico de red, discerniendo entre patrones normales e intrusivos, y generando alertas ante posibles amenazas. Existen IDS basados en firmas, que, a pesar de su eficacia para detectar patrones de ataques conocidos, presentan dificultades en ataques no identificados previamente, limitando su capacidad de adaptación a una guerra que está en constante evolución [19].

La IA surge como solución al problema de la detección de intrusiones, especialmente, el uso de técnicas de aprendizaje automático que buscan mejorar la precisión en la clasificación de ataques conocidos y detectar patrones anómalos en el tráfico de red. Aunque prometedores, estos enfoques encuentran desafíos relacionados con la disponibilidad limitada de datos de entrenamiento, la alta variabilidad en el tráfico de red y los costes asociados a errores en la detección. En [22], se abordan los problemas de las pruebas de penetración, necesarias para la identificación de vulnerabilidades de seguridad, explorándose desde una perspectiva de IA, utilizando modelos lógicos de la red, como grafos de ataque o árboles de decisión. La intersección entre la ciberseguridad, la detección de intrusiones y la aplicación de la IA aún plantea desafíos y por ello, se continúan volcando esfuerzos desde el MDEF para el desarrollo de estas nuevas herramientas.

2.2.5 Uso de sensores en el campo de batalla

En la recopilación de estudios del Centro Conjunto de Desarrollo de Conceptos (CCDC) del Ministerio de Defensa [23], se habla de cómo el uso de sensores en el campo de batalla maximiza el potencial de la IA. La integración de múltiples sensores a los soldados, vehículos y sistemas de captación de información del entorno, como estaciones meteorológicas portátiles, supone una ventaja a la hora de transmitir datos de interés. Como se ha venido diciendo, la IA requiere de grandes cantidades de datos, por lo que es necesario el desarrollo de sensores integrados.

La integración de sensores al soldado es una de las iniciativas que se proponen en el primer capítulo del estudio citado anteriormente: *"El cuerpo humano emite información de manera constante, consciente e inconsciente todo el tiempo"* [23]. Actualmente, se utilizan varios tipos de sensores no invasivos en los soldados para recopilar datos biométricos, como la actividad cardíaca, cerebral y muscular, las respuestas emocionales y de estrés de la piel, y la temperatura corporal. La voz también se emplea para identificación y control de dispositivos. Gracias a la combinación de múltiples señales fisiológicas, como la respiración, la sudoración y la presión sanguínea, se proporciona una imagen más completa del estado del individuo, incluyendo niveles de estrés y lesiones. Estas tecnologías mejorarán el rendimiento y la seguridad de los soldados en el campo y son la viva expresión de la industria 4.0 que interconecta elementos como la robótica y la realidad aumentada, siempre bajo el motor de procesamiento de la Inteligencia Artificial. Aunque existen prototipos de cascos que permiten la integración de sensores para el registro de estos datos en el campo de batalla, hoy en día, solo se utilizan estos datos en centros alejados del combate.

2.2.6 Vehículos aéreos no tripulados

Los drones son una evolución natural de las aeronaves y seguirán apareciendo como componentes necesarios para el desarrollo de múltiples operaciones. Actualmente, los drones equipados con Inteligencia Artificial podrían brindar nuevas oportunidades para apoyar a las tropas militares al proporcionar apoyo o información de manera más rápida.

Existen varios tipos de UAV militares, que se pueden clasificar en tres categorías según su altitud y autonomía: UAV tácticos, UAV de altitud media y larga duración (*Medium Altitude, Long Endurance*, MALE) y UAV de alto techo de operación y larga duración (*High Altitude, Long Endurance*, HALE) [24]. A efectos de IA, los más prometedores son los drones tácticos, ya que las decisiones autónomas de los mismos afectan únicamente al nivel táctico, sin comprometer el nivel operacional o estratégico.

Los drones tácticos son de pequeño tamaño y son fácilmente transportables por los soldados en el campo. Su principal misión es ayudar directamente a las tropas en el teatro de operaciones, proporcionándoles inteligencia situada en una zona geográfica restringida. El Ejército de los Estados Unidos ha estado utilizando los drones tácticos *Nova* de reconocimiento en entornos urbanos desde 2018, ya que los soldados se enfrentan a lo que llaman "*the fatal funnel"* [25], uno de los mayores riesgos que supone operar en zonas cerradas. El ejército francés también se ha equipado con drones *Parrot* con fines de observación diurna y nocturna y desde 2020, ha incorporado mini drones de reconocimiento Spy'Ranger (SMDR). Sus funciones abarcan desde localizar a un enemigo o una trampa durante la progresión de un equipo en tierra, hasta ajustar el fuego de artillería. Todos ellos cuentan con una integración de IA para la detección temprana de objetos y el vuelo semiautónomo, que permite al operador utilizarlo con seguridad en ambientes de combate sin requerimientos extraordinarios en el manejo de los drones.

Los UAV MALE se utilizan para misiones de inteligencia (detección de enemigos) o misiones de ataque en un radio mucho más amplio que los drones tácticos. Durante su intervención en Afganistán, el Ejército de los Estados Unidos equipó sus drones MALE, como el dron *Reapers*, con sistemas de armas. Los sistemas de adquisición de blancos y de estabilización en vuelo se logran gracias a la integración de IA en el propio dron. Por otro lado, los drones HALE que vuelan en el límite

estratosférico y carecen de misiles, ofrecen las mismas funciones que un satélite en términos de capacidad de análisis de imágenes, pero, a diferencia de éstos, pueden ser posicionados rápidamente sobre un área dada para obtener vigilancia permanente.

La operación de estos UAV de clase MALE y HALE no es completamente autónoma y solamente está automatizada para ciertas funciones. Actualmente, ningún piloto de drones (operador) tiene la libertad de tomar una decisión de disparo sin la autorización previa obtenida a través de la cadena de mando, al igual que un piloto convencional. El concepto de autonomía dentro de los drones militares aún se encuentra en fase de desarrollo, existiendo, además, incertidumbres con respecto a su uso, ya que su autonomía iría en contra de la estructura de la cadena de mando que garantiza un uso proporcional y legítimo de la fuerza. Por ello, el desarrollo de estas tecnologías está centrado en los drones tácticos, que prometen ofrecer grandes ventajas en operaciones urbanas gracias a la implementación de funciones inteligentes como discriminación entre objetivos y rehenes, análisis de daños, neutralización de objetivos y despeje autónomo de edificios.

2.3 Aprendizaje Automático (*Machine Learning*)

El aprendizaje automático es una subdisciplina de la Inteligencia Artificial que se enfoca en el desarrollo de algoritmos y modelos que permiten a las máquinas aprender patrones a partir de datos. En lugar de programar explícitamente reglas de lógica, el aprendizaje automático permite a las máquinas aprender de ejemplos y experiencias para mejorar su rendimiento en tareas específicas.

En la Figura 2-1 podemos observar la diferencia entre la programación tradicional, en la que el ordenador procesa unos datos a través de un programa para obtener una solución; y el aprendizaje automático, en el que los datos y la solución son variables ya conocidas y el ordenador busca patrones para generar un programa capaz de distinguir dichos patrones.

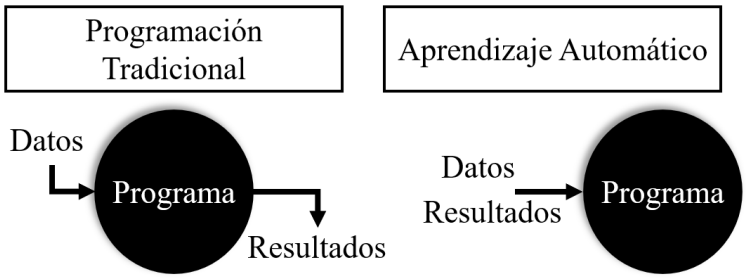


Figura 2-1: Esquemas de programación tradicional y de aprendizaje automático (elaboración propia)

El uso de técnicas de *Machine Learning* requiere una sólida base de algoritmos matemáticos y estadísticos para analizar una enorme cantidad de datos (*Big Data*). Para entender la clasificación de los algoritmos de *Machine Learning* es necesario entender los tipos de problemas a los que se puede enfrentar el algoritmo: clasificación, regresión o agrupamiento. A raíz de estos problemas surgen distintos tipos de algoritmos que se pueden clasificar fácilmente según su método de aprendizaje (aprendizaje supervisado, no supervisado, semisupervisado o por refuerzo). En los siguientes apartados se explicarán los tipos de problemas que se pretenden resolver con el aprendizaje automático. Posteriormente se procederá con una explicación de cada uno de los tipos de algoritmos que podemos encontrar (aprendizaje automático según su método de aprendizaje) y de los problemas que están destinados a resolver y, finalmente, se aclararán los conceptos de *Deep Learning* (DL), redes neuronales (*Neural Networks*, NN) y otros subcampos de la IA.

2.3.1 Tipos de problemas

Los problemas de clasificación buscan identificar a qué conjunto de categorías pertenece una nueva observación en función del conjunto de datos de entrenamiento que contiene las categorías observadas. Los problemas de clasificación se dividen en problemas de clasificación de dos clases y

problemas de clasificación multiclase. Predecir si un tumor es maligno o no [26] (Figura 2-2) sería un ejemplo del primer tipo y clasificar los diferentes tipos de buques sería un ejemplo de la clasificación multiclase.

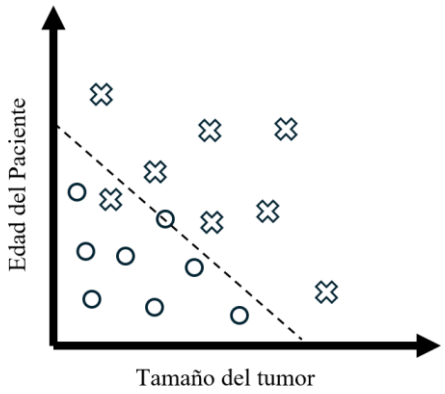


Figura 2-2: Ejemplo gráfico de algoritmo de clasificación de dos clases (elaboración propia)

El resultado de un problema de clasificación es un valor discreto que indica la clase predicha en la que se encuentra una observación, aunque en ocasiones también puede ser un valor continuo, que indica la probabilidad de que una observación pertenezca a una clase particular. Por ejemplo, se predice que el buque sea un pesquero una probabilidad del 0,85. Aquí, 0,85 es el valor continuo que indica la confianza de la predicción, y se puede convertir en un valor discreto seleccionando la predicción con la probabilidad más alta.

La técnica de regresión realiza una predicción estimando la relación entre variables. A diferencia de la clasificación, que prevé la clase a la que pertenece una observación, la regresión produce una variable de salida continua. Aplicaciones prácticas de problemas de regresión abarcan desde pronosticar el precio de una casa a partir de su tamaño (Figura 2-3) hasta predecir las temperaturas futuras en una determinada zona o calcular los tiempos a los que se deben realizar mantenimientos a ciertas piezas. La regresión modela y prevé valores numéricos en diversos contextos, siendo así una herramienta muy valiosa para la toma de decisiones basada en datos históricos [26].

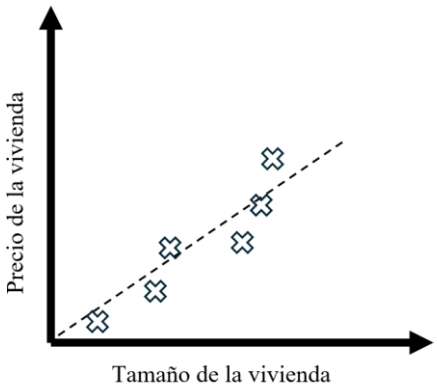


Figura 2-3: Ejemplo gráfico de algoritmo de regresión (elaboración propia)

El agrupamiento, a diferencia de la clasificación, identifica similitudes entre objetos, que agrupa según esas características en común y que les diferencian de los otros grupos de objetos. En otras palabras, la clasificación es un proceso supervisado que asigna una etiqueta a cada objeto en función de su similitud con los objetos de entrenamiento, mientras que el agrupamiento es un proceso no supervisado que agrupa los objetos en función de su similitud [27]. Ejemplos de estos problemas incluyen determinar qué votantes comparten preferencias por el mismo partido o identificar lotes de munición que presentan fallas de manera similar. Esta técnica se utiliza cuando se busca descubrir un

patrón específico en los datos. Este tipo de algoritmos se emplean en [28], donde se utiliza HDBSCAN para la detección de zonas de fondeo en las que se concentran los mensajes AIS. En la Figura 2-4 se ilustra un esquema de dicho estudio. Como se puede observar, los datos no están etiquetados.

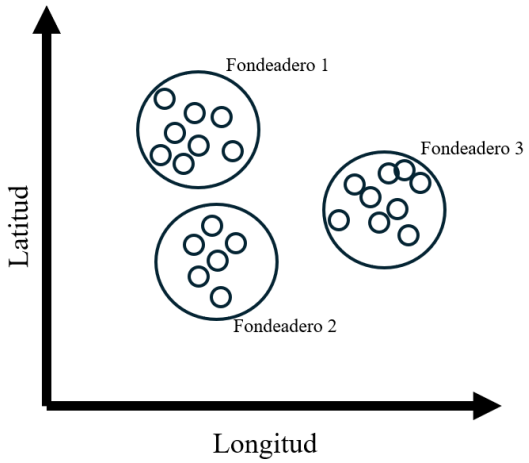


Figura 2-4: Ejemplo gráfico de algoritmo basado en agrupamiento (elaboración propia)

Tras la explicación de estos tres tipos de problemas que la IA puede solucionar, podemos inferir con total certeza que el problema que se tratará en este TFG será de clasificación, en el que las clases a predecir serán el tipo de buque, como se ha ejemplificado en la explicación de los problemas de clasificación.

2.3.2 Aprendizaje supervisado

El aprendizaje supervisado se sitúa como una subcategoría dentro del ámbito del ML y, a su vez, de la IA. Estos algoritmos toman conjuntos de datos etiquetados para entrenarse y, posteriormente, clasifican datos de naturaleza similar o predicen resultados con alta precisión. A medida que se incorporan datos al modelo, éste ajusta sus ponderaciones hasta lograr una adaptación efectiva, lo cual forma parte del proceso de validación cruzada [29].

Su modo de funcionamiento se basa en la utilización de un conjunto de datos de entrenamiento para enseñar a los modelos a generar la salida deseada. Dicho conjunto de datos debe contener entradas y salidas correctas que permiten al modelo aprender con el tiempo. El algoritmo evalúa su precisión mediante la función de pérdida, ajustándose hasta minimizar el error de manera suficiente.

El aprendizaje supervisado se desglosa en dos tipos de problemas en la minería de datos: clasificación y regresión. Como se ha visto en el subapartado anterior, la clasificación emplea un algoritmo para asignar de manera precisa datos de prueba a categorías específicas. Reconoce entidades dentro del conjunto de datos e intenta extraer conclusiones sobre cómo deben etiquetarse o definirse estas entidades. La regresión, por otro lado, se utiliza para comprender la relación entre variables dependientes e independientes. Ambos problemas se pueden abordar con aprendizaje automático supervisado. Sabiendo que este TFG aborda un problema de clasificación, se utilizará (debidamente justificado en el capítulo de desarrollo del TFG) un algoritmo de aprendizaje supervisado para la resolución de éste.

Existen distintos algoritmos de aprendizaje supervisado, generalmente implementados con lenguajes de programación como R o Python. Entre ellos destacan: las redes neuronales, que imitan la interconectividad del cerebro humano mediante capas de nodos y aprenden a través del proceso del gradiente descendente, se explicarán en mayor profundidad en el último subapartado; algoritmos de *Naive Bayes*, basados en el Teorema de Bayes que asume independencia condicional entre las clases; algoritmos de regresión lineal y logística, utilizados para identificar relaciones entre variables y

realizar predicciones; máquinas de vectores de soporte (SVM), que se emplean tanto para clasificación como para regresión, creando un hiperplano que maximiza la distancia entre clases; K Vecino Más Cercano (KNN), un algoritmo para clasificación basado en la proximidad de puntos de datos; y *Random Forest*, un algoritmo flexible que utiliza múltiples árboles de decisión para reducir la varianza y mejorar la precisión de las predicciones, entre otros. Todos éstos se detallarán más adelante en el apartado de algoritmos de aprendizaje automático.

Por último, a pesar de sus beneficios evidentes, el aprendizaje supervisado enfrenta grandes desafíos al crear modelos seguros y precisos. La estructuración de modelos de aprendizaje supervisado puede demandar niveles significativos de experiencia y el entrenamiento de estos modelos puede ser un proceso largo y demandante de recursos. Los conjuntos de datos pueden tener, además, una mayor probabilidad de contener errores humanos, ya que normalmente los datos deben ser preprocesados de manera que el algoritmo reciba unos datos fiables y libres de incoherencias, de lo contrario podría resultar en un aprendizaje incorrecto de los algoritmos. Finalmente, existen limitaciones en agrupaciones y clasificaciones automáticas, ya que, a diferencia de los modelos de aprendizaje no supervisado, el aprendizaje supervisado no puede realizar agrupaciones en clúster ni clasificar datos por sí solo [29].

2.3.3 Aprendizaje no supervisado

El aprendizaje no supervisado hace uso de algoritmos de ML para analizar y agrupar conjuntos de datos que carecen de etiquetas en clústeres. Estos algoritmos, a diferencia de los de aprendizaje supervisado, identifican agrupaciones de datos y patrones ocultos sin requerir intervención humana, por lo que se muestran sumamente útiles para tareas de determinación de patrones [30].

Existen diferentes enfoques para la resolución de problemas de agrupación. La agrupación en clústeres, como ya se ha visto, es una técnica de minería de datos que organiza datos no etiquetados en función de sus similitudes o diferencias, agrupándolos en estructuras o patrones. Estos algoritmos pueden ser exclusivos, superpuestos, jerárquicos o probabilísticos.

Mientras que la agrupación exclusiva estipula que un punto de datos pertenece a un solo clúster, como el algoritmo de agrupación en clústeres k-medias, la agrupación superpuesta permite que un punto de datos pertenezca a varios clústeres con diferentes grados de pertenencia. Por otro lado, la agrupación jerárquica, también conocida como análisis de clústeres jerárquicos, se clasifica en aglomerativa o divisiva. Utiliza métodos como enlace de Ward, enlace promedio, enlace completo y enlace único para medir la similitud entre clústeres. Por último, la agrupación probabilística se basa en la probabilidad de que un punto de datos pertenezca a una distribución específica, siendo el modelo de mezcla gaussiana un ejemplo común [30].

Para el cumplimiento de su objetivo, los algoritmos de ML no supervisado utilizan mecanismos como reglas de asociación o reducciones de dimensionalidad. Las reglas de asociación detectan relaciones entre variables en un conjunto de datos, siendo los algoritmos como “Apriori” los utilizados para generar estas reglas. Cuando el número de características en un conjunto de datos es alto, la reducción de dimensionalidad se utiliza para gestionar el rendimiento de los algoritmos. Métodos como Análisis de Componentes Principales (PCA), Descomposición en Valores Singulares (SVD) y codificadores automáticos son muy comunes para este tipo de depuración de datos [31].

El aprendizaje no supervisado presenta numerosos inconvenientes, como la complejidad computacional, tiempos de entrenamiento más extensos y el riesgo de resultados inexactos. La intervención humana a veces es necesaria para validar las variables de salida, y la base sobre la cual se agrupan los datos puede carecer de transparencia. Es por ello por lo que, a efectos de esta línea de investigación, se utilizará un demostrador que utilice técnicas de aprendizaje supervisado.

2.3.4 Aprendizaje semisupervisado

El aprendizaje semisupervisado ofrece un equilibrio entre el aprendizaje supervisado y el no supervisado. Durante el entrenamiento, utiliza un conjunto de datos etiquetado más pequeño para guiar la clasificación y la extracción de características de un conjunto de datos más extenso y no etiquetado. Se trata de una forma de resolver el problema cuando se carece de suficientes datos etiquetados para un algoritmo de aprendizaje supervisado o etiquetar datos es demasiado costoso.

En definitiva, la utilización de datos tanto etiquetados como no etiquetados permite que los modelos de aprendizaje semisupervisado mejoren su comprensión de patrones y relaciones en los datos. Al combinar las ventajas de la guía supervisada con las amplias cantidades de datos no etiquetados, estos modelos buscan lograr una mayor precisión y generalización en sus predicciones.

2.3.5 Aprendizaje por refuerzo

El aprendizaje por refuerzo constituye un campo dentro del aprendizaje automático que encuentra su inspiración en la psicología conductista. Su objetivo es determinar las acciones que debe llevar a cabo un agente de software en un entorno específico con el fin de maximizar alguna medida de "recompensa" o premio acumulado. En términos simples, el agente adquiere la capacidad de tomar decisiones mediante la retroalimentación que obtiene del entorno en forma de recompensas o castigos. Este enfoque se aplica en diversas áreas, como robótica, juegos, publicidad en línea, control de procesos y más. Resumiendo, a diferencia del aprendizaje supervisado y no supervisado, el aprendizaje por refuerzo no depende de un conjunto de datos etiquetados o no etiquetados. En su lugar, el algoritmo aprende a través de la interacción con el entorno y la retroalimentación que recibe.

2.3.6 Deep Learning (DL)

La Inteligencia Artificial es el término más amplio que abarca ML y DL. Existen distintas categorías de la IA: Inteligencia Artificial Estrecha (IAE), considerada una IA "débil" que se enfoca en realizar tareas específicas, como ganar un juego de ajedrez o identificar a una persona en fotos; la Inteligencia Artificial General (IAG), clasificada como una IA "fuerte" que tendría la capacidad de desempeñarse al nivel de otro ser humano; y la Inteligencia Artificial Superinteligente (IASI), que superaría la inteligencia y habilidades humanas. Actualmente, no existen formas explícitas de la IAG ni la IASI, aunque la investigación en este campo se desarrolla de manera muy acelerada [32].

El aprendizaje automático, como ya se ha visto, es un subconjunto de la Inteligencia Artificial. Cuando se configura correctamente, ayuda a realizar predicciones que minimizan los errores derivados de simples conjeturas. El aprendizaje automático clásico o "no profundo" depende de la intervención humana para permitir que un sistema informático identifique patrones, aprenda, realice tareas específicas y proporcione resultados precisos. Los expertos humanos determinan la jerarquía de características para entender las diferencias entre las entradas de datos, generalmente requiriendo datos más estructurados para aprender. Por ejemplo, si se le mostrara una serie de buques, como "remolcador", "pesquero" y "carguero", un experto humano determinaría las características distintivas de cada tipo de buque. La distancia de sus trayectorias, su eslora o su calado podrían ser características distintivas. Además, se podrían utilizar etiquetas para simplificar el proceso de aprendizaje a través del aprendizaje supervisado.

Por otro lado, el aprendizaje automático profundo, aunque puede aprovecharlos, no necesariamente requiere un conjunto de datos etiquetado, permitiéndole procesar datos no estructurados en su forma cruda (como texto o imágenes) y determinar automáticamente el conjunto de características que distinguen "remolcador", "pesquero" y "carguero" entre sí. A medida que generamos más datos masivos, los científicos de datos utilizarán más la técnica de *Deep Learning*. La principal diferencia, por tanto, entre el aprendizaje automático y el aprendizaje profundo radica en cómo aprende cada algoritmo y en cuántos datos utiliza cada tipo de algoritmo.

El aprendizaje profundo automatiza gran parte del proceso de extracción de características, eliminando parte de la intervención humana requerida. Permite, a su vez, el uso de grandes conjuntos de datos no necesariamente estructurados, ganándose el título de aprendizaje automático escalable. Esta capacidad es indispensable ya que se estima que más del 80% de los datos disponibles para el uso de IA son no estructurados [32]. Observar patrones en los datos permite que un modelo de aprendizaje profundo agrupe adecuadamente las entradas. Tomando el ejemplo anterior, podríamos agrupar imágenes de remolcadores, pesqueros y cargueros en sus respectivas categorías según las similitudes o diferencias identificadas en las imágenes [8].

En esta línea, las redes neuronales, también llamadas redes neuronales artificiales (RNA) o redes neuronales simuladas (RNS), son un subconjunto del aprendizaje automático y son la columna vertebral de los algoritmos de aprendizaje profundo. Se llaman "neuronales" porque imitan cómo las neuronas en el cerebro se comunican entre sí [33], tal y como se muestra en la Figura 2-5.

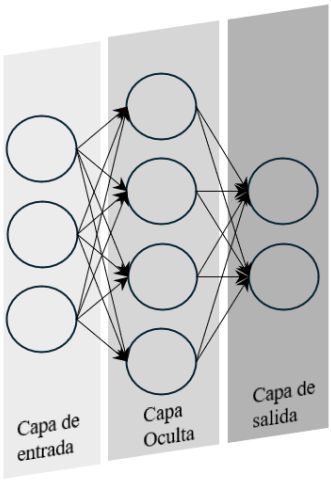


Figura 2-5: Arquitectura de una red neuronal (elaboración propia)

Las redes neuronales están compuestas por capas de nodos: una capa de entrada, una o más capas ocultas y una capa de salida. Cada nodo es una neurona artificial que se conecta con la siguiente, y cada uno tiene un peso y un valor umbral. Cuando la salida de un nodo está por encima del valor umbral especificado, ese nodo se activa y envía sus datos a la siguiente capa de la red. De lo contrario, no se envían datos a la siguiente capa a través de ese nodo. Esta propiedad de las redes neuronales permite que las tareas de reconocimiento de voz e imágenes pueden llevarse a cabo en minutos en lugar de las horas que tomarían manualmente. El algoritmo de búsqueda de Google es uno de los ejemplos más conocidos de una red neuronal.

La diferencia entre el aprendizaje profundo y las redes neuronales radica, precisamente en la palabra "profundo" que se refiere al número de capas en una red neuronal (véase Figura 2-6). Una red neuronal de más de tres capas, incluyendo las de entrada y salida, puede considerarse un algoritmo de aprendizaje profundo. La mayoría de las redes neuronales profundas son de avance, lo que significa que fluyen en una dirección, desde la entrada hasta la salida. Sin embargo, también se puede entrenar un modelo mediante retropropagación, es decir, moviéndose en la dirección opuesta, desde la salida hasta la entrada. La retropropagación nos permite calcular y atribuir el error asociado con cada neurona, lo que nos permite reajustar el algoritmo de manera óptima [32].

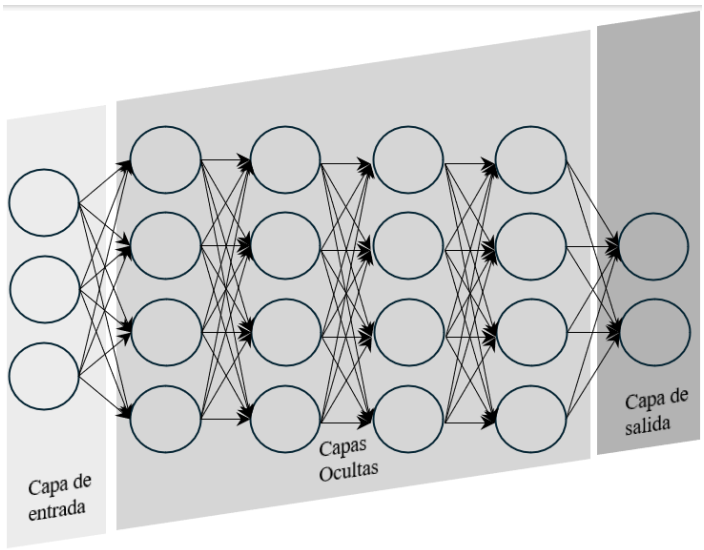


Figura 2-6: Esquema de una red neuronal profunda (elaboración propia)

2.4 Algoritmos de aprendizaje automático supervisado

2.4.1 Árboles de decisión

Los árboles de decisión son una técnica de aprendizaje supervisado que se utiliza para resolver problemas de clasificación y regresión. En lugar de fórmulas matemáticas complejas, estos algoritmos imitan el razonamiento humano a través de un diagrama de flujo similar a un árbol. El árbol comienza con un nodo raíz que representa el conjunto de datos completo. A partir de ahí, se divide en dos o más subnodos en función de una condición específica. Este proceso de división se repite en cada subnodo hasta que se llega a los nodos hoja, que representan las predicciones finales.

Los árboles de decisión tienen la ventaja de ser fáciles de comprender e interpretar, ya que simulan el proceso de toma de decisiones que un humano podría realizar. Sin embargo, uno de los desafíos principales es elegir las características correctas para las divisiones. Existen dos métodos para esto: en primer lugar, la ganancia de información busca dividir el conjunto de datos en subconjuntos más homogéneos; en segundo lugar, el índice de Gini minimiza la probabilidad de que dos elementos de un subconjunto sean de la misma clase.

Las ventajas de los árboles de decisión se manifiestan en varios parámetros de comparación. En primer lugar, suelen ofrecer una precisión superior a otros algoritmos de clasificación como *K-Nearest Neighbors* (KNN) o *Naive Bayes*. Además, son robustos frente a la sobrecarga y el ruido en los datos, lo que los hace adecuados para conjuntos de datos complejos y variables. Otra ventaja es su capacidad de interpretación, ya que permite visualizar la importancia de las variables en la clasificación final, lo que facilita la comprensión del modelo; esta herramienta es de gran utilidad para el procesado de datos y el cálculo de parámetros. También es eficiente en cuanto a tiempo de entrenamiento y uso de memoria, lo que lo hace adecuado para conjuntos de datos relativamente grandes.

Sin embargo, los árboles de decisión presentan ciertas desventajas que han de ser consideradas. Por ejemplo, el proceso de entrenamiento puede ser más complejo que el de otros algoritmos debido a la necesidad de ajustar múltiples árboles de decisión. Además, aunque es posible interpretar los árboles individuales del bosque, la interpretación global del modelo puede resultar más complicada, generándose una caja negra sobre el funcionamiento general del algoritmo. Por otro lado, existe el riesgo de sobreajuste si el modelo se ajusta demasiado a los datos de entrenamiento, lo que puede afectar su capacidad para generalizar bien a nuevos datos. Esta última consideración del sobreajuste de datos ha de ser tenida muy en cuenta ya que puede contaminar las predicciones al estar utilizando un *dataset* muy específico.

2.4.2 Random Forest

Random Forest es un algoritmo supervisado diseñado principalmente para la clasificación de datos. Su funcionamiento se basa en la creación de un conjunto de árboles de decisión, conformando lo que se conoce como un "bosque", donde cada árbol se entrena con una muestra aleatoria del conjunto de datos original. Se trata de un algoritmo ampliamente reconocido en el ámbito del aprendizaje automático y se presenta como una herramienta disponible en la librería de Python, *scikit-learn* [34]. Esta librería proporciona un conjunto completo de herramientas destinadas a tareas de aprendizaje automático y minería de datos, facilitando así el desarrollo de modelos predictivos y de clasificación.

El proceso de entrenamiento de *Random Forest* comienza mediante la selección aleatoria de muestras del conjunto de datos original con reemplazo, lo que significa que un mismo punto de datos puede aparecer en múltiples subconjuntos. Luego, para cada subconjunto, se construye un árbol de decisión utilizando un algoritmo estándar como CART (*Classification and Regression Trees*). Esta aleatoriedad en la selección de muestras y variables durante la construcción de los árboles ayuda a reducir la correlación entre los árboles individuales y a mejorar la diversidad del conjunto.

Durante la etapa de predicción, cada árbol del bosque realiza una clasificación o regresión individual basada en las características de entrada y produce un resultado. La predicción final se determina mediante la combinación de las predicciones de todos los árboles en el bosque. En el caso de clasificación, se utiliza un enfoque de votación mayoritaria para determinar la clase final, mientras que en la regresión se puede utilizar el promedio de las predicciones de todos los árboles.

Esta técnica de agrupación de árboles de decisión ofrece varias ventajas sobre un solo árbol de decisión. En primer lugar, al combinar múltiples árboles, *Random Forest* tiende a reducir el sobreajuste y a mejorar la precisión de las predicciones. Además, al permitir la selección aleatoria de muestras y variables en cada árbol, el modelo es más robusto frente a la variabilidad en los datos y menos susceptible a valores atípicos o ruido [35].

2.4.3 Otros algoritmos de aprendizaje supervisado

Tras haber analizado *Random Forest* como algoritmo supervisado en el apartado anterior, es importante explorar otros algoritmos relevantes dentro del campo del aprendizaje automático. Estos algoritmos presentan diversas características y aplicaciones que los hacen útiles en diferentes contextos.

Uno de los algoritmos más conocidos es el de *K-Nearest Neighbors* (KNN). Este método se basa en la idea de que los puntos de datos con características similares tienden a pertenecer a la misma clase o tener valores de salida similares. Durante la etapa de predicción, el algoritmo calcula la distancia entre el punto de datos a clasificar y los puntos de datos de entrenamiento más cercanos, y asigna la etiqueta más común entre sus vecinos más cercanos como la etiqueta de clasificación. Una diferencia fundamental con *Random Forest* es que KNN no construye un modelo explícito durante la fase de entrenamiento, sino que almacena los datos de entrenamiento y calcula las predicciones en función de la proximidad a estos datos. Esto puede hacer que KNN sea más susceptible al ruido en los datos y menos eficiente en términos de tiempo de predicción, especialmente en conjuntos de datos grandes [36].

Otro algoritmo comúnmente utilizado es el de las máquinas de vectores de soporte (*Support Vector Machines* o SVM). SVM es un algoritmo de clasificación que busca encontrar el hiperplano óptimo que mejor separa las clases en un espacio de características de alta dimensión. Durante el entrenamiento, SVM maximiza el margen entre las clases y asigna los puntos de datos a una clase en función de en qué lado del hiperplano caen. Mientras que SVM se enfoca en encontrar el mejor hiperplano de separación lineal o no lineal, *Random Forest* puede capturar relaciones no lineales entre las características gracias a sus árboles. SVM puede ser más adecuado para conjuntos de datos con alta dimensionalidad y es menos propenso al sobreajuste en comparación con *Random Forest* en ciertos escenarios [37].

Gradient Boosting, por otro lado, es una técnica de aprendizaje automático que construye un conjunto de modelos predictivos de manera secuencial, donde cada modelo se enfoca en corregir los errores del modelo anterior. Durante el entrenamiento, *Gradient Boosting* busca minimizar la función de pérdida al ajustar gradualmente los pesos de los datos en los árboles de decisión. Aunque puede ser más lento que *Random Forest*, este método de autocorrección cíclica puede lograr una alta precisión en la predicción, especialmente en conjuntos de datos complejos y desbalanceados [38].

La regresión lineal y polinomial son métodos más simples pero efectivos para problemas de regresión. La regresión lineal busca establecer una relación lineal entre una variable independiente y una variable dependiente, mientras que la regresión polinomial permite modelar relaciones no lineales al agregar términos de potencia a la ecuación. En la misma línea, los métodos bayesianos utilizan el teorema probabilístico de Bayes para estimar la probabilidad de una hipótesis dado un conjunto de datos. Son útiles cuando se necesita tener en cuenta la incertidumbre en las predicciones y pueden adaptarse bien a diferentes contextos [39]. Son particularmente útiles cuando se necesita tener en cuenta la incertidumbre en las predicciones y pueden adaptarse bien a una variedad de problemas, desde la clasificación hasta la regresión y el agrupamiento [40].

Para finalizar, aunque existen varios algoritmos de aprendizaje automático supervisado, el analizado en el apartado anterior, *Random Forest*, destaca por su balance entre eficiencia temporal y calidad en la predicción. Su capacidad para generar predicciones rápidas y precisas lo hace especialmente adecuado para proyectos donde se requiera una alta precisión y eficiencia en el tiempo de predicción. Por lo tanto, para el trabajo en cuestión, *Random Forest* se adecúa como mejor modelo, proporcionando un equilibrio óptimo entre precisión, interpretabilidad y eficiencia.

2.5 Conocimiento del Entorno Marítimo (CEM)

El Conocimiento del Entorno Marítimo (CEM) implica la fusión y análisis de información de múltiples fuentes para obtener una visión detallada y en tiempo real de los espacios marítimos de interés nacional. Esta labor continua de síntesis y análisis genera lo que se conoce como *"Recognized Maritime Picture"* (RMP), una representación precisa de las actividades en curso en las aguas territoriales. A diferencia del pasado, el entorno marítimo ahora se extiende más allá de las fronteras tradicionales, adaptándose a un concepto más amplio que considera tanto la ubicación geográfica como el factor temporal. Este enfoque dinámico y amplio refleja la naturaleza cambiante de las amenazas y los intereses en el mar.

Con más de 200,000 buques transitando anualmente por las rutas marítimas bajo soberanía española [7], la protección de estas áreas es esencial no solo para la economía nacional, sino también para la seguridad internacional. El mar se ha convertido en un medio utilizado por diversas organizaciones para llevar a cabo actividades ilícitas, que abarcan desde el terrorismo hasta el tráfico de armas y la inmigración ilegal. En este contexto, la Armada Española tiene la responsabilidad de salvaguardar estos espacios críticos.

Surge así la necesidad imperante de ejercer vigilancia y control efectivos sobre estos espacios marítimos. En este sentido, el conocimiento detallado del entorno marítimo se ha vuelto indispensable para planificar y ejecutar acciones de seguridad marítima de manera efectiva. En la actualidad, el Centro de Operaciones y Vigilancia de Acción Marítima (COVAM) desempeña un papel crucial al recopilar y analizar información de diversas fuentes para proporcionar un panorama completo del entorno marítimo. Esto permite a las Fuerzas Armadas, a través de la Fuerza de Acción Marítima (FAM), llevar a cabo operaciones con pleno conocimiento de la situación y una comprensión precisa de los desafíos y amenazas presentes en el mar.

2.5.1 COVAM

El Centro de Operaciones y Vigilancia de Acción Marítima (COVAM) [41], ubicado en Cartagena y parte del Estado Mayor del Almirante de Acción Marítima (ALMART), no se limita a ser un espacio físico con medios de Mando y Control y personal cualificado, como tradicionalmente se concebían los Centros de Operaciones Navales. Su función va más allá y se diferencia en varios aspectos.

A diferencia de los Centros de Operaciones Navales, cuyo enfoque está principalmente orientado hacia la vigilancia y operaciones militares en el mar, el COVAM se establece como un elemento que aglutina la información que la Armada ofrece a la Comunidad Marítima. Esto significa que, aunque la Armada conserva su capacidad de vigilancia y operaciones navales, el COVAM se concibe como un punto de cooperación con otros actores, especialmente en el contexto de la Acción Marítima.

Como se ha mencionado previamente, el concepto de "entorno marítimo" ha evolucionado considerablemente, y ahora abarca no solo áreas geográficas específicas, sino también intereses cambiantes en el tiempo y en función de nuestras contribuciones a organizaciones internacionales. Esta evolución ha sido impulsada por la globalización y las nuevas tecnologías, que han ampliado las capacidades de vigilancia y cooperación en áreas extensas y distantes.

El COVAM está estructurado en tres capas de seguridad para manejar diferentes tipos de información. La primera capa es abierta y se encarga de la información de conocimiento general del entorno marítimo. La segunda capa maneja información sensible, mientras que la tercera capa está reservada para información clasificada y de acceso exclusivamente militar. Esta estructura permite analizar y distribuir la información de manera efectiva y segura, adaptándose a las necesidades de conocimiento y compartición en cada situación.

Para apoyar a la Comunidad Marítima, el COVAM utiliza principalmente las dos primeras capas, asegurando que la información analizada se contraste con la disponible en la capa clasificada, garantizando la protección adecuada de la información sensible. Además, la arquitectura abierta del COVAM permite la participación física de personal de otros organismos de la Administración Marítima en el análisis y fusión de información, facilitando el acceso a los sistemas necesarios sin complicaciones técnicas [41].

El Almirante de Acción Marítima (ALMART) desempeña, como una de sus funciones, la de Comandante del Mando Operativo Marítimo (CMOM). Su responsabilidad principal radica en el planeamiento, dirección y supervisión de las operaciones de vigilancia y seguridad en los espacios marítimos de responsabilidad y soberanía nacional, así como en aquellos de interés estratégico. Para llevar a cabo esta misión, el CMOM colabora estrechamente con diversas autoridades y organismos, tanto civiles como militares, a nivel nacional e internacional. Es aquí donde se representa una de las funciones del COVAM: la de coordinación con múltiples organismos para un fin común. Además, el ALMART también tiene la tarea de mantener actualizado el Conocimiento del Entorno Marítimo en los espacios de interés nacional, siendo responsable de activar, coordinar y gestionar el Sistema de Cooperación y Orientación al Tráfico Marítimo (NCAGS) a nivel nacional.

El COVAM, en colaboración con las unidades de la Fuerza de Acción Marítima (FAM), se encarga de velar por la seguridad de las aguas de interés nacional, que incluyen las aguas territoriales españolas, el Golfo de Guinea, el Golfo Pérsico y el Océano Índico [42]. Su principal objetivo es proporcionar la *Recognized Maritime Picture* (RMP) a todas las unidades de la Armada, estaciones en tierra y agencias civiles que lo soliciten, garantizando una vigilancia constante para cumplir con diversas responsabilidades operativas y tácticas.

El personal del COVAM se distribuye de manera que se cubran todos los roles necesarios para garantizar su funcionamiento continuo. Además, cuenta con especialistas encargados del planeamiento y preparación de actividades, análisis del Conocimiento del Entorno Marítimo, y mantenimiento de bases de datos y sistemas. Estos recursos se alimentan de diversas fuentes de información, siendo los datos AIS una de las más relevantes.

2.5.2 Datos AIS

El Sistema de Identificación Automática (AIS) fue desarrollado originalmente para evitar colisiones y, hoy en día, se ha convertido en una herramienta fundamental para el control del tráfico marítimo y la vigilancia de la seguridad en la mar. En el año 2002, el Convenio internacional para la seguridad de la vida humana en el mar (SOLAS, por sus siglas en inglés) de la Organización Marítima Internacional (OMI) hizo obligatorio el uso del AIS para buques de pasajeros y de más de 300 toneladas brutas en viajes internacionales.

Existen dos tipos de transpondedores AIS. Por un lado, los de clase A son obligatorios para buques de más de 300 toneladas brutas que realizan viajes intercontinentales. Ofrecen información más completa sobre el buque, incluyendo su posición, velocidad, rumbo, calado y tipo de carga. Los de clase B, por otro lado, se diseñaron para embarcaciones de recreo y buques de menor tamaño. El transpondedor AIS clase B, a pesar de ofrecer menos información que los de clase A, es más ligero y económico. En el COVAM, por ejemplo, la información codificada en las tramas AIS se decodifica, procesa y se integra con datos de otras fuentes para ser visualizada en consolas de presentación. Los datos se organizan en cuatro grupos: estáticos, dinámicos, relacionados con el viaje y mensajes de seguridad [43].

Los datos estáticos son transmitidos por AIS cada 6 minutos o bajo petición y presentan los siguientes campos:

- Número MMSI: identificador único de 9 dígitos para cada estación transmisora.
- Número IMO: código de 7 dígitos asignado a cada buque para facilitar su identificación y mejorar la seguridad marítima.
- Nombre y distintivo de llamada del buque.
- Tipo de buque.
- Ubicación de la antena de fijación de posición, que permiten conocer la eslora y manga del buque.

La información dinámica se transmite con más frecuencia que la estática. La latencia de transmisión varía según los cambios de velocidad y rumbo. Los campos dinámicos son los siguientes:

- Posición GPS del barco con indicación de precisión.
- Marca de tiempo de la posición en hora UTC.
- Rumbo sobre fondo: Rumbo efectivo del buque.
- Velocidad sobre fondo: Velocidad efectiva del buque.

La información relacionada con el viaje se transmite con la misma cadencia que los datos estáticos y los mensajes breves relacionados con la seguridad se transmiten bajo demanda del transmisor. Los campos de estos dos grupos se muestran a continuación:

- Calado del buque.
- Tipo de cargamento.
- Plan de ruta: información sobre el último puerto de salida, el siguiente puerto de recalada, y la fecha y hora de salida y recalada.
- Mensajes de texto libre: sirven para transmitir información breve a unidades y estaciones cercanas con el objetivo de mejorar la seguridad marítima.

El AIS ha demostrado ser una herramienta indispensable para mejorar la seguridad marítima al reducir el riesgo de colisiones entre buques y facilitar la respuesta ante emergencias en el mar. Además, permite un mejor control del tráfico marítimo al permitir a las autoridades costeras monitorizar el movimiento de los buques en tiempo real. Sin embargo, el AIS presenta desventajas importantes, como la posibilidad de falseamiento de datos y errores humanos al introducir información, lo que puede comprometer su fiabilidad. Además, la falta de estandarización de campos,

como los campos de texto libre, dificulta la interpretación precisa de la información transmitida. A pesar de estas limitaciones, el AIS sigue siendo un sistema fundamental para mejorar la seguridad y la eficiencia del tráfico marítimo, al proporcionar una mayor transparencia en el entorno marítimo y facilitar la comunicación entre buques y autoridades costeras.

2.6 Estudios Relacionados

En este apartado, se abordarán estudios que han servido como guía para este TFG, permitiendo enfocar ciertos aspectos adecuadamente gracias al trabajo previo. En primer lugar, se hará una breve revisión bibliográfica de los estudios relacionados con vigilancia marítima y detección de anomalías. Posteriormente, se explicarán los TFG anteriores a éste que contribuyen a una misma línea de investigación desarrollada en el CUD-ENM. Por último, se procederá con el resumen de cinco estudios que han servido de guía para la elaboración del modelo de este proyecto.

2.6.1 Vigilancia Marítima y detección de anomalías

La vigilancia marítima se lleva a cabo mediante la combinación de múltiples sensores y fuentes: estaciones de radar, aeronaves de patrulla, aprovechamiento de los datos AIS, etc. Estas fuentes generan una gran cantidad de datos de las embarcaciones y la detección manual de comportamientos anómalos es difícil. En lugar de ello, se emplean enfoques de aprendizaje automático para crear modelos de normalidad a partir de patrones de movimiento de las embarcaciones. Cualquier desviación de estos modelos se considera anómala y se debe presentar a los operadores para su inspección manual [19].

Un enfoque inicial para la detección de anomalías marítimas utiliza la arquitectura de red neuronal *Fuzzy ARTMAP* para modelar la velocidad normal de las embarcaciones basándose en la ubicación del puerto [44]. Otro enfoque utiliza el aprendizaje asociativo de patrones de movimiento para predecir el movimiento de una embarcación utilizando como datos de entrada su posición GPS y su rumbo [45]. Otros emplean agrupamiento no supervisado basado en modelos de mezclas gaussianas (GMM) [46] y estimación de densidad de kernel (KDE) [47]. Estos modelos permiten la detección de embarcaciones que cambian de dirección, cruzan carriles marítimos, se desplazan en dirección opuesta o viajan a alta velocidad. Enfoques más recientes utilizan redes bayesianas para detectar si se ha falseado el valor tipo de buque, así como movimientos discontinuos, imposibles y deambulantes [48]. El desarrollo futuro de la detección de anomalías marítimas también debería considerar las embarcaciones circundantes y la interacción entre múltiples embarcaciones [19].

2.6.2 Líneas predecesoras

Este TFG es la continuación de tres trabajos anteriores llevados a cabo en el Centro Universitario de la Defensa en la Escuela Naval Militar (CUD-ENM).

En [7], se propuso utilizar aprendizaje automático supervisado para predecir el tipo de barco a partir de otros campos de los mensajes AIS, utilizando el algoritmo *Random Forest*. En [9], se profundizó en esta propuesta, mejorando el preprocesado y tratamiento de los datos, así como optimizando los tiempos de ejecución y la calidad de la predicción del tipo de barco. En ambos se concluyó que los datos estáticos adquieren gran relevancia para la predicción.

En [10], se exploró cómo el empleo de información relacionada con el área de actividad del buque podría mejorar la predicción del tipo de barco, concluyendo que esta información contribuye positivamente a la mejora de la predicción, beneficiando especialmente a ciertos tipos de barcos. De esta manera, se concluyó que el análisis del comportamiento de los buques puede mejorar sensiblemente la predicción. Es así como nace la propuesta de este TFG, el análisis del comportamiento de los buques estudiando sus trayectorias puede desvelar parámetros valiosos para mejorar la calidad de la predicción del algoritmo.

2.6.3 Estudios de análisis de trayectorias y modelos de predicción

En [49], se utilizaron datos AIS para clasificar barcos en aguas de Indonesia. Se emplearon características como eslora, manga y calado para clasificar los barcos en 9 tipos. En el estudio, se usó un 80% de los datos para entrenamiento y el resto para pruebas. Los resultados mostraron una precisión del 99.8%. Además, introduce el estudio de trayectorias, algo novedoso para nuestra línea de investigación y el uso del algoritmo *XG Boost*, un algoritmo que como hemos visto anteriormente ofrece mayor precisión que *Random Forest* a cambio de mayores tiempos de procesado. Sin embargo, este estudio utiliza campos no presentes en las tramas AIS proporcionadas para este trabajo, y la extracción de las características para trayectorias muestra cierta opacidad que impide su reproducibilidad. A pesar de ello, propone el uso de medias para la reducción de datos y el uso de una zona geográfica de datos reducida optimiza los tiempos, convirtiéndolo en un eficaz demostrador de las capacidades del análisis de trayectorias.

El modelo realizado en [50] alcanzó una precisión del 97.51% en la clasificación de tipos de barcos tomando como datos los mensajes Tipo B transmitidos por buques en la Bahía Alemana. En este estudio, se alcanzan grandes precisiones gracias a la calidad de los datos a pesar del reducido tamaño del *dataset* (unos 7000 buques distintos que en dos meses generan más de 700000 trayectorias), ya que se obtienen mensajes AIS con una cadencia de entre 2 y 10 segundos. La extracción de parámetros sigue una metodología en la que cada trayectoria conforma un nuevo dato, aumentando la cantidad de datos para entrenamiento, utilizando la misma cantidad de datos en realidad, lo que podría generar el indeseado sobreajuste de datos. Sin embargo, la compresión de trayectorias y el filtrado de las mismas son buenas aproximaciones a la mejora de calidad de datos. Las limitaciones en la calidad de los datos proporcionados para la realización de este TFG impedirán el uso de estas técnicas de preprocesado.

En [51], se busca extraer parámetros de trayectorias y almacenarlos en base a un solo buque. Este enfoque se asemeja más al objetivo de este TFG ya que permite aunar los datos estáticos y dinámicos. Por lo que respecta a la extracción de parámetros, éstos son muy parecidos a los mencionados en el anterior estudio ([50]), sin embargo, la combinación de parámetros, tal y como se muestra en la sección de "*Trajectory Feature Extraction"* de la Figura 2-7, resulta interesante para su inclusión en este TFG. Finalmente, el autor propone dos métodos para convertir el modelo de entrenamiento en clasificador en línea que pueda analizar datos en tiempo real.

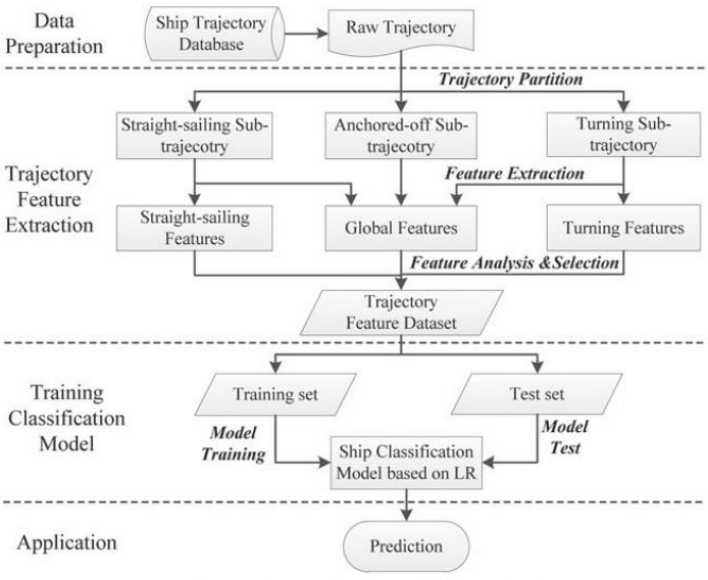


Figure 1. Overall process of building a ship classification model.

Figura 2-7: Metodología de extracción de parámetros [51]

En [52], se utilizan datos históricos de AIS proporcionados por la Autoridad Marítima Danesa para crear trayectorias de barcos y extraer 39 características, incluyendo propiedades conductuales, geográficas y medidas de las características de apariencia de los barcos. Se entrenan varios algoritmos de clasificación para identificar cuatro tipos de barcos: pesqueros, pasajeros, petroleros y cargueros. El estudio concluye que las características diseñadas para el reconocimiento de los buques pueden expresar elementos característicos de las trayectorias de los barcos y ser utilizadas para su clasificación. Analizando una amplia variedad de algoritmos, se concluye que con *Random Forest* se obtienen los mejores resultados, alcanzando una precisión del 84.05% con un vector de características de 39 dimensiones.

En [3], se extrae un conjunto de características de trazados digitales de [53] junto con otras características obtenidas de los estudios previos mencionados (subapartados *2.6.3-2.6.5)* y entrena varios modelos de clasificación, obteniendo la mejor predicción (81,7%) con *Random Forest*. Se utilizaron datos AIS de la Autoridad Marítima Danesa, con una distribución temporal de 5 días en noviembre de 2022.

3 DESARROLLO DEL TFG

En el presente capítulo, se pretende detallar cada paso tomado, desde la adquisición de datos hasta su empleo en los distintos algoritmos de aprendizaje supervisado. Tras haber explicado todas las técnicas de ML en el capítulo anterior, se justificará debidamente el uso de cada herramienta empleada para la realización de todas las pruebas.

La metodología que se seguirá estará dividida en dos ramas que se irán desarrollando paralelamente. Se comenzará con un análisis exhaustivo de los datos tanto estáticos como dinámicos, de manera que las conclusiones extraídas de ambos análisis sirvan de justificación para las funciones de preprocesado que les serán aplicadas para su reducción de tamaño. Este paso de preprocesado resulta esencial ya que reducirá tiempos de manera drástica a la hora de procesar los parámetros. En líneas anteriores, la limpieza de datos se realizaba de manera paralela al procesado de los mismos, aumentando los tiempos de ejecución enormemente. Se pretende, por tanto, definir bien la línea entre el preprocesado (o limpieza) de datos y su posterior procesado, siendo la terminación de la primera parte, requisito esencial para el comienzo de la segunda. Por último, solo quedaría alimentar a los algoritmos y describir los experimentos que se han realizado. En el Anexo III se muestra un diagrama de flujo de la metodología general.

Es necesario recalcar que el objetivo principal del TFG no es otro que la extracción de parámetros en base a trayectorias de los datos dinámicos. Sin embargo, gran parte del proceso se nutre de los TFG ya realizados, y, en este proceso se han encontrado formas de posible mejoría en cuanto a precisión del algoritmo y se han implementado en la medida de lo posible, dejando aquellas cuestiones más complejas propuestas en el capítulo de líneas futuras.

3.1 Entorno de trabajo

En este apartado se describirán las herramientas que se han utilizado para el desarrollo de este TFG. En primer lugar, se explicará el lenguaje de programación que se usará, continuando con el entorno de gestión de bibliotecas. Después se describirán las características de la plataforma de trabajo que se utiliza para el análisis de datos y el entrenamiento del algoritmo. Por último, se presentarán ciertas bibliotecas de interés especial para este TFG, dejando aquellas secundarias brevemente explicadas en el Anexo IV, comentadas en el código.

3.1.1 Python

Python [54] es un lenguaje de programación de alto nivel y de propósito general que ha ganado una gran popularidad debido a su sintaxis clara y legible. Esta legibilidad lo hace atractivo tanto para principiantes como para programadores experimentados, ya que facilita la comprensión y el

mantenimiento del código a lo largo del tiempo. Además de su legibilidad, *Python* es conocido por ser un lenguaje versátil que se puede utilizar en numerosas aplicaciones, desde el desarrollo web y la automatización de tareas hasta el análisis de datos e inteligencia artificial.

Una de las ventajas clave de *Python* es que cuenta con una comunidad grande y activa de desarrolladores que contribuyen con bibliotecas y marcos de trabajo que amplían su funcionalidad y lo hacen aún más atractivo a la hora de usarlo para labores de investigación.

3.1.2 Anaconda

Anaconda [55] es una distribución de *Python* que destaca en el ámbito de la ciencia de datos y la computación científica. Una de las características más importantes de Anaconda es su gestor de paquetes llamado *conda*, que simplifica enormemente la instalación, actualización y gestión de paquetes y sus dependencias. Esto es especialmente útil en entornos científicos donde los proyectos pueden depender de numerosas bibliotecas y versiones específicas de estas bibliotecas.

Además del gestor de paquetes, Anaconda ofrece la capacidad de crear entornos virtuales. Estos entornos virtuales permiten a los usuarios aislar las dependencias de un proyecto específico, lo que ayuda a evitar conflictos entre versiones de paquetes y garantiza la reproducibilidad del entorno de desarrollo. Anaconda también proporciona una amplia colección de paquetes preinstalados diseñados específicamente para la ciencia de datos, el aprendizaje automático y otras áreas relacionadas, lo que facilita la configuración inicial de un entorno de trabajo para proyectos complejos. Para este proyecto, se ha creado un entorno en *conda* de manera que permanezcan intactas las bibliotecas utilizadas y sus versiones. A efectos individuales del TFG, puede no ser relevante el uso de un entorno específico, sin embargo, no deja de ser una buena práctica para facilitar el estudio de futuros investigadores.

3.1.3 Jupyter Lab

Jupyter Lab [56] es una interfaz de usuario interactiva diseñada para trabajar con documentos de *Jupyter*, que combinan código, texto explicativo y resultados en la misma interfaz. La capacidad de integrar código y resultados en un único documento facilita la creación de informes y análisis de datos reproducibles y compartibles.

Una de las características distintivas de *Jupyter Lab* es su capacidad para admitir múltiples lenguajes de programación, no solo *Python*. Esto lo hace útil en una variedad de contextos más allá de la ciencia de datos, como la educación y la investigación académica. *Jupyter Lab* es altamente personalizable, lo que permite a los usuarios adaptar la interfaz de usuario a sus necesidades específicas y preferencias de trabajo. Además, su arquitectura extensible permite a los usuarios agregar nuevas funcionalidades y características a través de extensiones, lo que lo convierte en una herramienta flexible y poderosa para una variedad de tareas de desarrollo y análisis de datos.

3.1.4 Bibliotecas Python

A continuación, se explicarán brevemente las bibliotecas utilizadas para este trabajo. Se han escogido aquellas más reseñables y que han sido de especial relevancia para el TFG, dejando, para aquellas menos importantes (aunque necesarias), una breve explicación de la librería en el código proporcionado en el Anexo IV.

- *csv*: Esta librería proporciona funciones para leer y escribir archivos CSV (Valores Separados por Comas), un formato común para intercambiar datos tabulares.
- *os*: Permite la manipulación de archivos y directorios en el sistema operativo. Proporciona funciones para trabajar con rutas de archivos, crear, renombrar y eliminar archivos, y otras operaciones relacionadas con el sistema de archivos.
- *pandas*: Es una de las bibliotecas más utilizadas para la manipulación y análisis de datos en *Python*. Proporciona estructuras de datos flexibles y eficientes, como *DataFrame* y *Series*,

así como funciones para realizar operaciones de limpieza, transformación, filtrado y análisis de datos.

- *geopandas*: Extiende las funcionalidades de *pandas* para trabajar con datos geoespaciales. Permite la manipulación y análisis de datos geográficos, como puntos, líneas y polígonos, y proporciona herramientas para realizar operaciones espaciales y visualización de datos geográficos.
- *movingpandas*: Es una librería especializada en el análisis de datos de movimiento, como trayectorias y conjuntos de puntos GPS. Proporciona funciones para realizar análisis temporales y espaciales de datos de movimiento, como cálculo de velocidad, aceleración y detección de paradas.
- *numpy*: Es una librería fundamental para la computación científica en *Python*. Proporciona estructuras de datos eficientes para trabajar con matrices multidimensionales, así como funciones para realizar operaciones matemáticas y estadísticas.
- *matplotlib*: Es una librería utilizada para la visualización de datos en *Python*. Proporciona una amplia variedad de funciones para crear gráficos estáticos y dinámicos, incluyendo gráficos de líneas, barras, dispersión, histogramas y más.
- *seaborn*: Es una librería de visualización de datos estadísticos basada en *matplotlib*. Proporciona una interfaz de alto nivel para crear gráficos estadísticos atractivos e informativos.
- *scikit-learn*: Es una librería de aprendizaje automático en *Python* que proporciona herramientas simples y eficientes para la minería y análisis de datos. Incluye una amplia variedad de algoritmos de aprendizaje supervisado y no supervisado, así como herramientas para evaluación de modelos, selección de características y preprocesamiento de datos.
- *tqdm*: Proporciona una barra de progreso visual para mostrar el progreso de bucles iterativos y otras tareas en tiempo real. Es útil para monitorizar el progreso de tareas largas e iterativas.

Las demás bibliotecas siguen la misma línea, proporcionando funcionalidades específicas para tareas como visualización geoespacial, generación de informes de perfil de datos, agrupación de datos, cálculo de distancias geoespaciales, visualización interactiva de datos, entre otros. Cada una de estas bibliotecas amplía las capacidades de *Python* en diferentes áreas de análisis y visualización de datos.

3.1.5 Datos empleados

Los datos utilizados en el demostrador para el entrenamiento del algoritmo de predicción vienen dados por el mismo COVAM, que recoge los datos de tráfico mundial transmitidos por todos los buques desde su equipo AIS. El formato de los datos depende de si son estáticos (que se transmiten con menor frecuencia) o dinámicos (cuya latencia es mayor). Los datos proporcionados por el COVAM se procesan con una librería específica de tratamiento y transformación de datos AIS: *pyais* [57]. Esta librería no será objeto de estudio, ya que en este trabajo se parte del formato transformado que se explica a continuación. Una vez tratados, todos ellos vienen en un formato CSV clasificados en carpetas que los dividen por meses. Los meses proporcionados son los meses de enero y febrero de 2023. Para mayor claridad, se realizarán todas las explicaciones con los datos del mes de enero y se replicarán algunos experimentos con el resto de los meses para corroborar resultados.

3.2 Preparación de datos estáticos

El procesado de los datos estáticos ha sido objeto de estudio en [7]. En este TFG nos limitaremos a preparar los datos estáticos siguiendo las mismas directrices, pero implementando ciertas mejoras en cuanto a eficiencia de procesado, incoherencias no resueltas anteriormente y estructuración de los datos de manera que se puedan concatenar con aquellos que se extraigan de los dinámicos a partir de

las trayectorias. Todo el pretratamiento de los datos estáticos se irá documentando con imágenes explicativas del código, dejándose en el Anexo IV el código *Prep_Statics*.

3.2.1 Análisis de datos

Los datos estáticos se localizan en 31 ficheros CSV, uno por día del mes de enero de 2023 y 28 ficheros CSV para el mes de febrero, de entre 100 y 200 Mb aproximadamente, sumando por directorio un total de unos 5,5 y 5 Gb de datos, respectivamente. El primer paso será leer los ficheros de un solo mes y almacenarlos en un *Data Frame* (DF), que no es más que una estructura de datos tabulados bidimensionalmente que se utiliza en el análisis de datos y la programación en *Python* con la librería de *pandas*. En la Figura 3-1, se muestra el código utilizado para leer los ficheros y almacenarlos en un DF.

```
1  %%time
2  # Almacena los nombres solo los archivos que contienen "static" en el nombre
3  lista_archivos_static = [archivo for archivo in os.listdir('2023-24') if "static" in archivo]
4  # Para ver los nombres de los archivos: print(lista_archivos_static)
5  # Crea un DataFrame vacío para ir almacenando los datos de todos los csv estáticos
6  df_raw_static = pd.DataFrame()
7  # Itera sobre los archivos "static" y los carga en el DataFrame
8  for archivo_static in lista_archivos_static:
9      df_temporary = pd.read_csv(os.path.join('2023-24', archivo_static))
10     df_raw_static = pd.concat([df_raw_static,df_temporary], ignore_index=True)
11 # Ahora df_raw_static contiene todos los datos de los archivos estáticos
```

Figura 3-1: Lectura y almacenamiento de los datos estáticos

Tras esto, quedan almacenadas en el DF, llamado *df_raw_static*, todas las filas de datos extraídos de los 31 ficheros. La función mágica *"%%time"* de la línea 1 permite mostrar por pantalla los tiempos de procesado, que servirán más adelante para justificar el uso de ciertas funciones, estructuras y bibliotecas en lugar de otras que se venían usando en los TFG que preceden a éste. En la Figura 3-2 se muestra la estructura de este DF, visualizando las primeras y las últimas filas. Como podemos observar en la Figura 3-2, existen casi más de 9 millones de entradas en el DF que contienen los 9 campos. La Tabla 3-1 muestra una breve descripción de cada campo que contiene el DF.

```
1  df_raw_static.shape
```

```
(93850348, 9)
```

```
1  df_raw_static.head(3)
```

	timestamp	MMSI	IMO	to_bow	to_stern	to_port	to_starboard	draught	shiptype
0	1672527350	352489000	9683661	276	57	26	34	21.2	ShipType.Tanker
1	1672527350	273810100	8907125	37	68	17	3	8.8	ShipType.Fishing
2	1672527350	273448240	8509155	29	65	12	3	6.0	ShipType.Fishing

```
1  df_raw_static.tail(3)
```

	timestamp	MMSI	IMO	to_bow	to_stern	to_port	to_starboard	draught	shiptype
93850345	1675205546	229863000	1012476	37	19	5	5	3.1	ShipType.PleasureCraft
93850346	1675205546	314537000	9573921	146	23	19	8	5.6	ShipType.Cargo
93850347	1675205547	538071332	1013004	60	47	7	7	4.3	ShipType.PleasureCraft

Figura 3-2: Estructura de *df_raw_static*

Campos	Descripción
Timestamp	Hora de recepción del mensaje AIS
MMSI	*Maritime Mobile Service Identity*
IMO	Número de Identificación de la Organización Marítima Internacional
To_bow	Distancia de la antena a la proa
To_stern	Distancia de la antena a la popa
To_port	Distancia de la antena al costado de babor
To_starboard	Distancia de la antena al costado de estribor
Draught	Calado del buque
Shiptype	Tipo de buque

Tabla 3-1: Descripción de los campos estáticos AIS

El análisis de estos datos es imprescindible para entender cómo se comportan, de manera que sepamos como preprocesarlos. No solo es necesario definir los campos, sino que también hay que comprender el comportamiento de los datos en función de los mismos. Este proceso se denomina análisis exploratorio de datos (EDA, por sus siglas en inglés). En este subapartado, se mostrará, además de la estructura inicial del DF, cómo se distribuyen los datos, con el fin de extraer una serie de conclusiones. Se debe recordar que los datos, aunque muestran similitudes con los datos tratados en los TFG anteriores, no son los mismos ni presentan el mismo formato, ya que son *datasets* distintos. Por tanto, es posible que algunos procesos de filtrado utilizados anteriormente ya no sean necesarios o viceversa.

El primer enfoque que se le debe dar al análisis de los datos es la comprobación de los duplicados. Las filas duplicadas no solo aumentan el tamaño de los datos, lo que da lugar a mayores tiempos de procesamiento, sino que también pueden falsear la calidad de las predicciones del algoritmo, ya que los datos de prueba del algoritmo pueden estar duplicados en los datos de entrenamiento.

Como podemos observar en la lista que recoge el MMSI y las veces que se repite en el DF (Figura 3-3), existen MMSI que se llegan a repetir hasta más de 5000 veces. En total, existen 85834 MMSI distintos en las más de 9 millones de filas del DF. A continuación, (Figura 3-4) se representará gráficamente la dispersión de los MMSI frente a las veces que éste se repite.

```
1  print(df_raw_static['MMSI'].value_counts())
```

```
MMSI
210231000    5185
230938210    5164
264900021    5100
257088680    5070
258020030    5042
             ...
525125012       1
273336280       1
657211400       1
244060589       1
355401000       1
Name: count, Length: 85834, dtype: int64
```

Figura 3-3: Número de mensajes que comparten el mismo MMSI

En el gráfico de dispersión de la Figura 3-4, la mitad de los MMSI se repiten al menos casi 400 veces. Este comportamiento resulta, contra intuitivamente, de lo más normal, puesto que la tasa de transmisión de los mensajes estáticos es de uno cada 6 minutos, por lo que en una navegación de 40 horas ya se transmitirían 400 mensajes AIS de tipo estático. Para asegurarnos de que se cumplen los valores de la gráfica, se imprimen las primeras 42917 entradas de la lista (Figura 3-5), que coincide con la mitad de la longitud total de la lista inicial. Comprobando el valor de la entrada número 42917 (resaltada en amarillo) se repite un total de 393 veces, lo cual coincide con el valor de la mediana marcada en el gráfico.

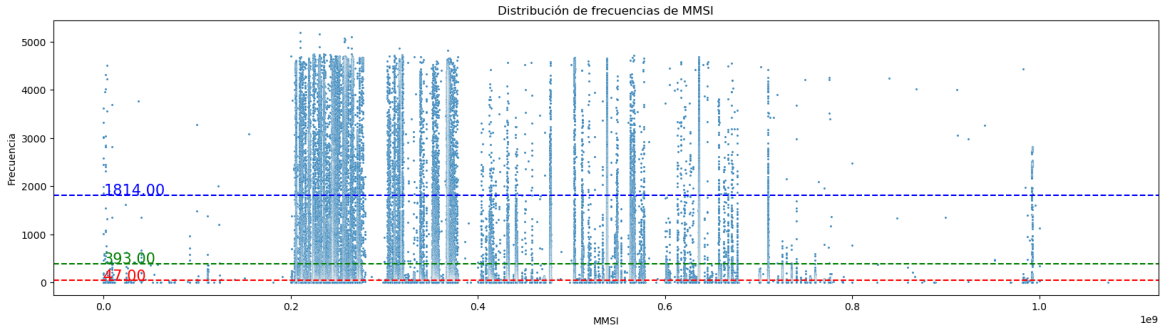


Figura 3-4: Gráfico de dispersión de MMSI frente al número mensajes emitidos en el mes

```
1  df_raw_static['MMSI'].value_counts().head(42917)

MMSI
210231000    5185
230938210    5164
264900021    5100
257088680    5070
258020030    5042
             ...
377489000     393
357952000     393
228794000     393
374298000     393
538002510     393
Name: count, Length: 42917, dtype: int64
```

Figura 3-5: Lista de frecuencias de MMSI hasta llegar a la mediana (resaltada en amarillo)

En la Figura 3-6, se observa un histograma en el que se representa, en saltos de 100, la cantidad de MMSI frente a las veces que se repiten. En dicho histograma, vemos que la mayoría de MMSI (unos 35000) se repiten entre 1 y 200 veces (escalón del histograma) y a medida que avanzamos en la cantidad de repeticiones, existe menor cantidad de buques. Por ejemplo, unos 5000 MMSI se repiten entre 200 y 300 veces. Esto quiere decir que el grueso de los buques transmite en AIS durante poco tiempo. Esta información será de gran ayuda a la hora de tratar los datos dinámicos. Como conclusiones para el preprocesamiento, sería interesante considerar quedarnos solo con una muestra de cada MMSI que se puede llegar a repetir hasta 5000 veces, sin embargo, veremos cómo los datos, a pesar de ser estáticos varían con el tiempo para el mismo buque, especialmente valores como el calado, que cambian en función de la carga. Los parámetros *to_bow*, *to_stern*, *to_port* y *to_starboard* también cambian con el tiempo, debido a un posible reposicionamiento de la antena. Por ello, se calculará una media de dichos parámetros para obtener un único valor para cada MMSI.

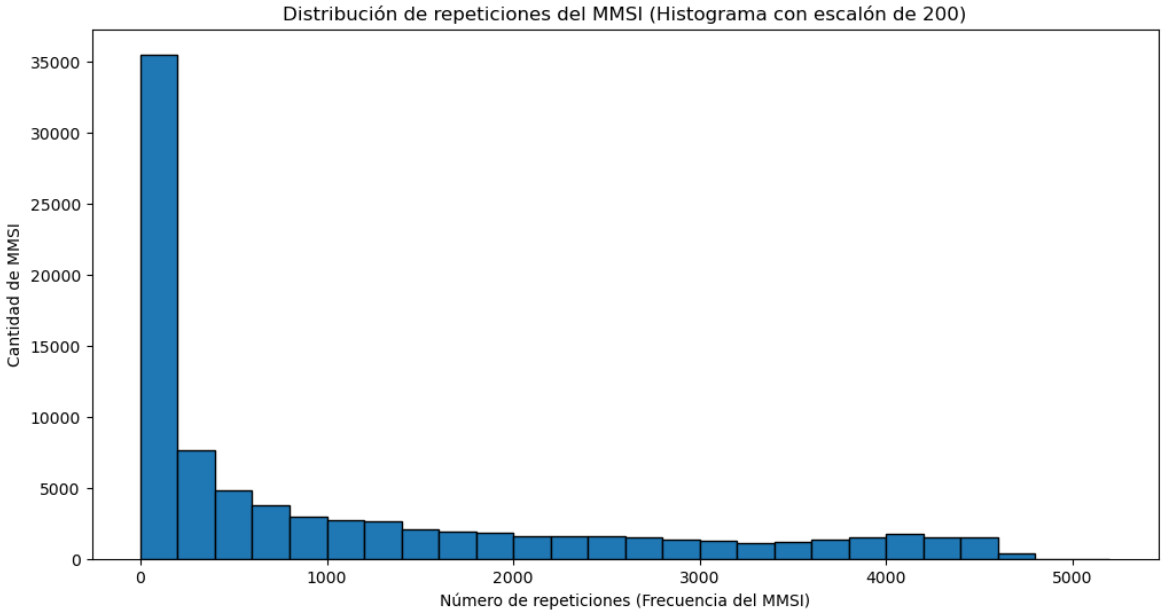


Figura 3-6: Histograma de número de mensajes repetidos

En la Figura 3-7, se observa una lista de todos los campos *shiptype* y su porcentaje presente en los mensajes de todo el mes. La Figura 3-8 muestra un diagrama de sectores en el que aparezcan las 10 categorías del campo *shiptype* más frecuentes. A efectos de este trabajo, nos centraremos en las categorías más representativas. Observamos también que muchas categorías son un subtipo de un tipo de buque con mayor presencia. En anteriores TFGs se agrupan las categorías para tener mayor cantidad de datos válidos, ya que parten de un formato en el que el campo *shiptype* viene codificado con un número. Esta práctica también se realizará en este TFG, utilizando otros métodos basados en diccionarios, ya que el campo no viene cubierto en un formato numérico, sino en un formato *string*.

Ship Type	Percentage		
ShipType.Cargo	19.583932	ShipType.Cargo_Reserved	0.276160
ShipType.Tug	10.262081	ShipType.AntiPollutionEquipment	0.268554
ShipType.Fishing	10.166100	ShipType.DivingOps	0.233824
ShipType.Tanker	7.192190	ShipType.Passenger_Reserved	0.230591
ShipType.Passenger	5.504481	ShipType.OtherType_Reserved	0.202404
ShipType.NotAvailable	5.111624	ShipType.HSC_NoAdditionalInformation	0.200848
ShipType.Cargo_NoAdditionalInformation	5.062283	ShipType.Cargo_HazardousCategory_B	0.179649
ShipType.OtherType	4.062052	ShipType.Tanker_HazardousCategory_C	0.169166
ShipType.Towing	3.853443	ShipType.Cargo_HazardousCategory_C	0.133699
ShipType.PleasureCraft	3.641912	ShipType.NonCombatShip	0.112623
ShipType.SPARE	3.614028	ShipType.Passenger_HazardousCategory_A	0.104387
ShipType.OtherType_NoAdditionalInformation	2.294338	ShipType.Tanker_Reserved	0.091303
ShipType.Tanker_NoAdditionalInformation	2.163035	ShipType.MedicalTransport	0.087074
ShipType.Cargo_HazardousCategory_A	1.825507	ShipType.WIG_HazardousCategory_B	0.056106
ShipType.PilotVessel	1.729630	ShipType.Passenger_HazardousCategory_B	0.049088
ShipType.SearchAndRescueVessel	1.704077	ShipType.HSC_Reserved	0.044408
ShipType.DredgingOrUnderwaterOps	1.652333	ShipType.OtherType_HazardousCategory_A	0.038532
ShipType.Passenger_NoAdditionalInformation	1.596450	ShipType.OtherType_HazardousCategory_B	0.036569
ShipType.HSC	1.185486	ShipType.Passenger_HazardousCategory_D	0.032259
ShipType.Sailing	0.902428	ShipType.OtherType_HazardousCategory_C	0.028760
ShipType.LawEnforcement	0.694332	ShipType.WIG_NoAdditionalInformation	0.028743
ShipType.MilitaryOps	0.663279	ShipType.WIG_Reserved	0.026818
ShipType.Tanker_HazardousCategory_B	0.530997	ShipType.Passenger_HazardousCategory_C	0.024184
ShipType.Towing_LengthOver200	0.465419	ShipType.WIG_HazardousCategory_C	0.017185
ShipType.Tanker_HazardousCategory_A	0.461671	ShipType.OtherType_HazardousCategory_D	0.016851
ShipType.Cargo_HazardousCategory_D	0.397480	ShipType.HSC_HazardousCategory_D	0.015686
ShipType.Tanker_HazardousCategory_D	0.354696	ShipType.HSC_HazardousCategory_A	0.012341
ShipType.WIG	0.295565	ShipType.WIG_HazardousCategory_A	0.007162
ShipType.PortTender	0.290491	ShipType.HSC_HazardousCategory_B	0.007003
		ShipType.WIG_HazardousCategory_D	0.005593
		ShipType.HSC_HazardousCategory_C	0.001090

Figura 3-7: Listado de tipos de buques y su porcentaje frente al número total de mensajes

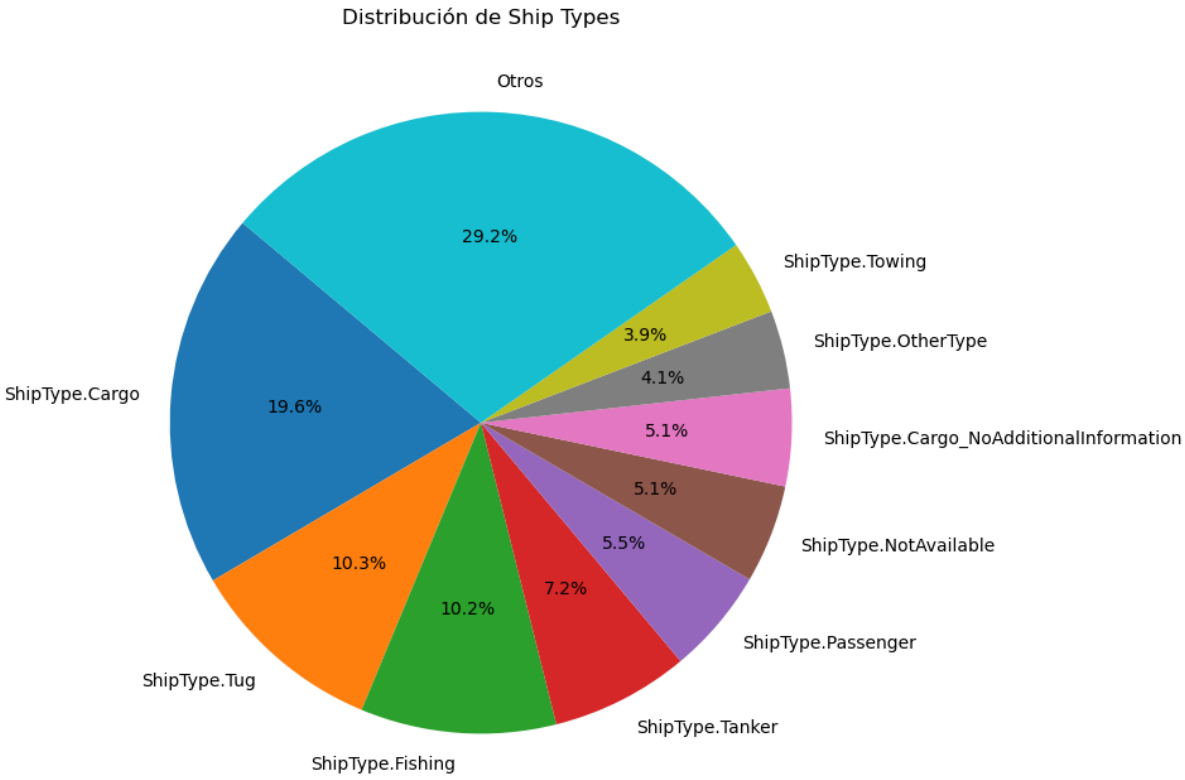


Figura 3-8: Diagrama de sectores de la Figura 3-7

Se presenta en la Figura 3-9 un histograma con los valores de calado de cada buque. Como se puede observar, existe un gran número de entradas con valor de calado 0. Esto puede deberse a que

este campo no ha sido cubierto o que, al tratarse de una embarcación de pequeñas dimensiones, el usuario aproxima el valor del calado a 0. Los valores de *draught* igual a cero podrían generar problemas en el cálculo de parámetros. Como ya veremos en el filtro, las filas con calados igual a cero se desecharán, así como las filas cuyos futuros valores de eslora y manga vayan a ser 0. A pesar de eliminar datos posiblemente útiles, este filtro nos facilitará las operaciones, tal y como se venía haciendo en anteriores ocasiones.

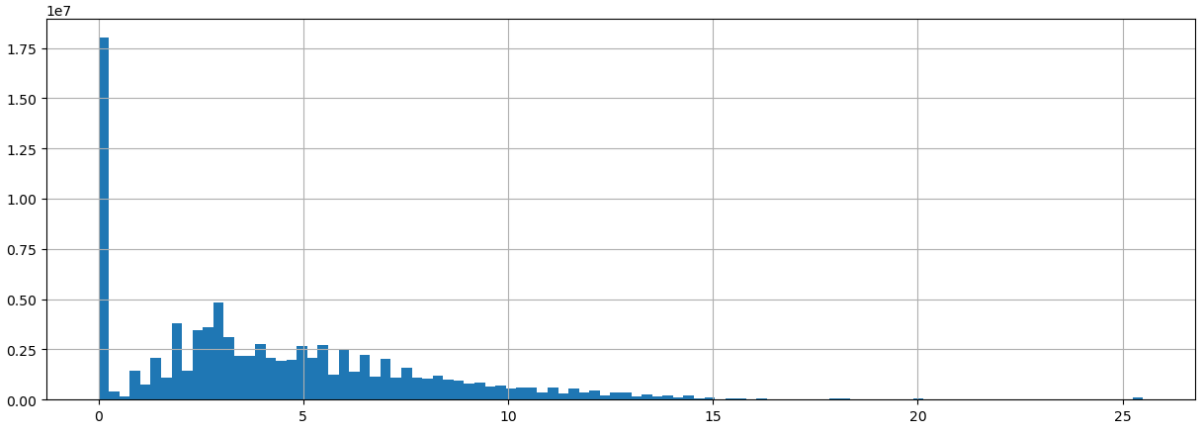


Figura 3-9: Histograma de calados

Finalmente, se realiza una comprobación de infinitos y valores nulos ya que en anteriores TFG se trata de eliminar este tipo de valores. En la Figura 3-10 se observa que el DF no contiene ni infinitos ni valores nulos, por lo que sus posteriores procesamientos tampoco los tendrán. Como veremos en el cálculo de parámetros, existen algunas divisiones que pueden ocasionar infinitos, sin embargo, se explicará en el correspondiente apartado cómo se actúa frente a dichos problemas.

```
#Comprobación de infinitos con pandas sustituye inf por nan
has_infinity = (df_raw_static.replace([np.inf, -np.inf], np.nan)).isna().any().any()

if has_infinity:
    print("El DataFrame contiene infinitos y/o valores nulos")
else:
    print("El DataFrame no contiene ni infinitos ni valores nulos")
#Acabamos de comprobar que no tiene infinitos ni valores nulos.
```

```
El DataFrame no contiene ni infinitos ni valores nulos
```

Figura 3-10: Comprobación de valores infinitos y nulos

3.2.2 Preprocesado de datos

El primer aspecto que se debe tratar es la fusión de subgrupos. El objetivo es realizar el estudio con los tipos de buques que más representación tienen en el conjunto, sin embargo, antes de eliminar todas las categorías que no pertenezcan al conjunto deseado, es necesario combinar los subgrupos dentro de un tipo de buque genérico. Por ejemplo, en la Figura 3-7, observamos que existen diferentes subclases para *shiptype.tanker* (*shiptype.tanker_hazardous_category_A, B, C…*) que, aunque tengan una representación muy baja por separado, agrupándolas se aumenta significativamente el porcentaje de representación de la categoría. En la Figura 3-11 observamos los nuevos porcentajes al agrupar las categorías. Debe recordarse que este conjunto de datos todavía contiene duplicados.

```
# Agrupa las subcategorías de buques en las categorías principales.
# Renombra los campos eliminando lo que viene después del guion bajo
df_clean_static_1['shiptype'] = df_clean_static_1['shiptype'].apply(lambda x: re.sub(r'_.*$', '', x))
```

Ship Type	Percentage
ShipType.Cargo	27.458709
ShipType.Tanker	10.963058
ShipType.Tug	10.262081
ShipType.Fishing	10.166100
ShipType.Passenger	7.541440
ShipType.OtherType	6.679505
ShipType.NotAvailable	5.111624
ShipType.Towing	4.318862
ShipType.PleasureCraft	3.641912
ShipType.SPARE	3.614028
ShipType.PilotVessel	1.729630
ShipType.SearchAndRescueVessel	1.704077
ShipType.DredgingOrUnderwaterOps	1.652333
ShipType.HSC	1.466862
ShipType.Sailing	0.902428
ShipType.LawEnforcement	0.694332
ShipType.MilitaryOps	0.663279
ShipType.WIG	0.437173
ShipType.PortTender	0.290491
ShipType.AntiPollutionEquipment	0.268554
ShipType.DivingOps	0.233824
ShipType.NonCombatShip	0.112623
ShipType.MedicalTransport	0.087074

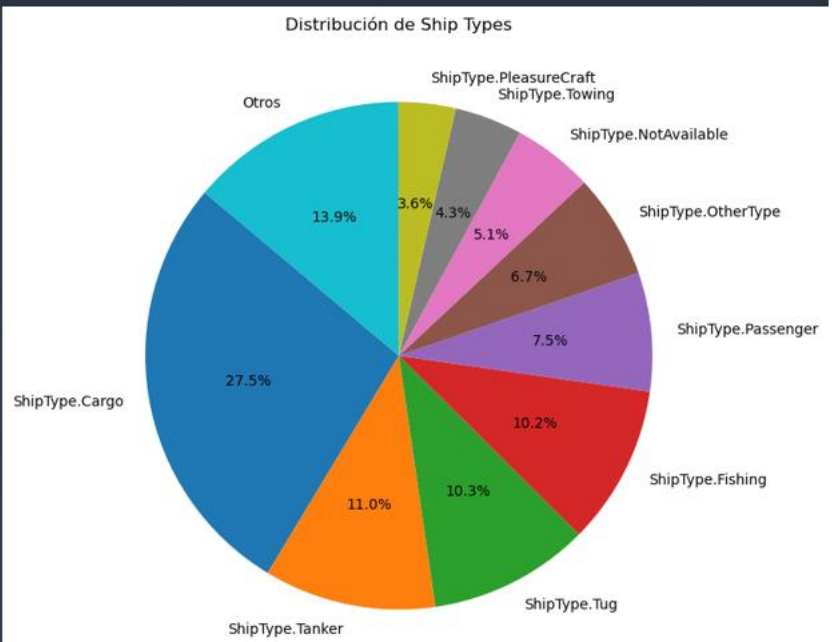


Figura 3-11: Porcentaje de tipos de buque tras la agrupación de subgrupos

La Tabla 3-2 muestra el aumento de representación de cada clase. Los buques *cargo*, *tanker* y *passenger* han aumentado en gran medida su porcentaje, especialmente el *cargo*, que ha aumentado casi un 8%, ofreciendo más datos para este trabajo.

Shiptype	**Representación original**	**Representación añadiendo subgrupos**
Cargo	19.58%	27.46%
Tanker	7.19%	10.96%
Fishing	10.17%	10.17%
Passenger	5.50%	7.54%
Tug	10.26%	10.26%

Tabla 3-2: Aumento de representación en las clases

Tras esta operación, se procede a eliminar las categorías que no se podrán predecir dada su escasa representación. En la Figura 3-12, se muestra cómo se reduce el número de filas, pasando de un DF con 93850348 entradas a uno con 49467377 (un 53% del DF inicial).

```
# Eliminación de las filas con shiptype no deseado
df_reduced_static = df_reduced_static.loc[df_raw_static['shiptype'].isin(['ShipType.Cargo',
                                                                           'ShipType.Tug',
                                                                           'ShipType.Fishing',
                                                                           'ShipType.Tanker',
                                                                           'ShipType.Passenger'])]
```

```
print(df_reduced_static.shape)
print(len(df_reduced_static)/len(df_raw_static))
```

```
(49467377, 9)
0.5270878377563395
```

Figura 3-12: Estructura de *df_reduced_static* y porcentaje de reducción

El próximo paso consiste en eliminar los campos que no son necesarios. En primer lugar, el campo *timestamp* podría ser de utilidad para analizar el flujo temporal de mensajes estáticos, pero, tras su análisis, se extraen resultados con tiempos de transmisión irregulares que carecen de relevancia, especialmente, al tratarse de datos estáticos. En segundo lugar, el campo *IMO* es otro identificador que tiene la misma función que el MMSI. Como veremos en los datos dinámicos, el campo *IMO* no existe y utilizaremos el MMSI para correlacionar datos estáticos y dinámicos. Por tanto, nos quedaremos con el identificador MMSI y suprimiremos el campo *IMO*, que no aporta ninguna información adicional. En la Figura 3-13 se muestra cómo se suprimen los campos y el DF resultante.

```
# Eliminación de columnas no deseadas
df_reduced_static = df_reduced_static.drop(columns=['IMO', 'timestamp'])
df_reduced_static.head()
```

	MMSI	to_bow	to_stern	to_port	to_starboard	draught	shiptype
0	352489000	276	57	26	34	21.2	ShipType.Tanker
1	273810100	37	68	17	3	8.8	ShipType.Fishing
2	273448240	29	65	12	3	6.0	ShipType.Fishing
6	412602000	161	29	14	18	7.1	ShipType.Cargo
7	512009984	13	12	6	7	0.0	ShipType.Tug

Figura 3-13: Estructura de *df_reduced_static* tras la eliminación de los campos *timestamp* e *IMO*

Como se ha inferido en una de las conclusiones de análisis de datos, la mayor reducción de datos viene dada por la gestión de duplicados. Para ello, es preciso extraer los valores medios de cada campo por MMSI como valor determinante del calado y asociado a un único MMSI. Ya que la media puede no representar fielmente el conjunto, sería interesante extraer parámetros estadísticos de los datos de ese campo antes de reducirlos, aunque, para mantener las predicciones fieles a los anteriores TFG y poder comparar resultados, se dejará el valor medio únicamente. Antes de realizar las medias es necesario asegurarse de que un MMSI corresponde a un mismo buque. La forma más eficiente de comprobar esto es asegurándose de que las agrupaciones de cada fila con mismo MMSI tengan cubierto el mismo *shiptype*. De esta manera, eliminamos la posibilidad de tener dos buques de distinto tipo con un mismo MMSI. Aunque podrían quedar aún buques diferentes con un mismo MMSI, al ser del mismo *shiptype*, no debería afectar al algoritmo tan gravemente.

En las siguientes celdas de código (Figura 3-13) se realizan múltiples operaciones. En primer lugar, se obtiene un nuevo DF que contiene las filas sin las incoherencias explicadas anteriormente. Después se almacena otro DF con las filas que muestran incoherencias para analizarlas, en total, se han

encontrado casi medio millón de filas incoherentes sobre los 49 millones de filas anteriores (casi un 1% del DF). Tomando como ejemplo el primer MMSI del DF, observamos que su distribución de *shiptype* no es uniforme (Figura 3-14) ya que hay otros buques que han transmitido con ese mismo MMSI. En este caso, lo ideal sería eliminar las muestras menos representativas, sin embargo, al tratarse de menos de un 1% de los datos y para optimizar los tiempos de procesamiento, se ha optado por eliminar la totalidad de los datos incoherentes.

```
df_clean_static = (
    df_reduced_static
    .groupby('MMSI')
    # Filtra valores del mismo grupo (MMSI) con shiptype diferente
    .filter(lambda x: len(set(x['shiptype'])) == 1)
)
```

```
# Almacena las filas filtradas para comprobar que son incoherentes
df_trash = df_reduced_static[~df_reduced_static.index.isin(df_clean_static.index)]
df_trash = df_trash.sort_values(by='MMSI')
df_trash.shape
```

```
(461172, 7)
```

```
print(len(df_trash)/len(df_reduced_static))
```

```
0.009322750223849548
```

```
df_trash.head(1)
```

	MMSI	to_bow	to_stern	to_port	to_starboard	draught	shiptype
49563853	1	44	44	9	9	4.0	ShipType.Passenger

```
df_trash_example = df_trash[df_trash['MMSI'] == 1]
```

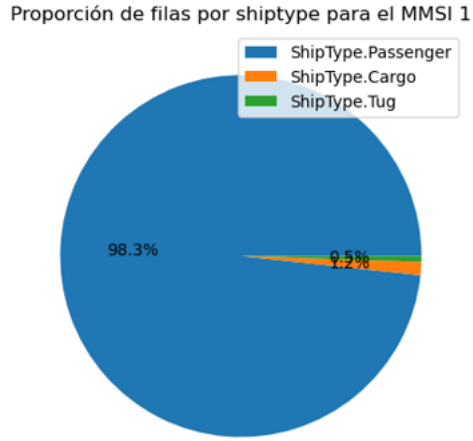


Figura 3-14: Filtrado, listado y ejemplo de MMSI incoherente

Si analizamos la muestra filtrada (ver Figura 3-15), vemos como el 100% de los mensajes transmitidos por un mismo MMSI presentan el mismo valor para el campo *shiptype*, por lo que la muestra está libre de incoherencias. De esta forma, nos quedan 49 millones de entradas.

```
df_clean_static = df_clean_static.sort_values(by='MMSI')
```

```
df_clean_static.shape
```

```
(49006205, 7)
```

```
df_clean_static.head(1)
```

	MMSI	to_bow	to_stern	to_port	to_starboard	draught	shiptype
13563579	5	10	27	5	5	3.4	ShipType.Tug

```
df_clean_example = df_clean_static[df_clean_static['MMSI'] == 5]
```

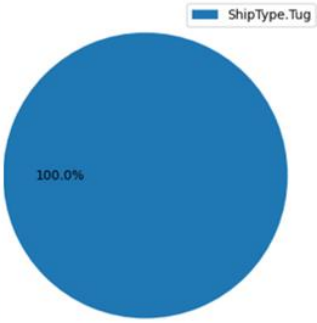


Figura 3-15: Proporción de *shiptype* para el MMSI 5 contenido en *df_clean_static*

Otro de los requisitos que el buque con único MMSI debe cumplir es que los valores del resto de campos sean coherentes. Esto quiere decir que los valores de calado, eslora y manga no pueden ser 0. La eslora y la manga se comprueban fácilmente estableciendo que los valores de *to_bow* y *to_stern* no pueden ser 0 simultáneamente, así como los de *to_port* y *to_starboard*. El valor de calado 0, sin

embargo, presenta problemáticas no tan obvias como los de eslora y manga. Un calado 0 puede significar que se trata de un buque con tan poco calado que el usuario lo ha aproximado a 0, pero también puede provenir de un buque cuyo calado es mayor que 0 y simplemente se ha decidido no cubrir ese campo. Como veremos en la Figura 3-16 los calados 0 provienen principalmente de petroleros, cuyo calado no debería ser aproximado a 0. Por ello, consideraremos todos los buques con el campo de calado 0 como no fiables, lo que, aunque suponga una reducción considerable del DF (Figura 3-16), asegura la fiabilidad de los datos.

```
# Elimina los valores de calado, eslora y manga = 0
df_clean_static = (
    df_clean_static
    .query('draught != 0\
        and not (to_bow == 0 and to_stern == 0)\
        and not (to_port == 0 and to_starboard == 0)')
)
```

```
df_clean_static.shape
```

```
(40576316, 7)
```

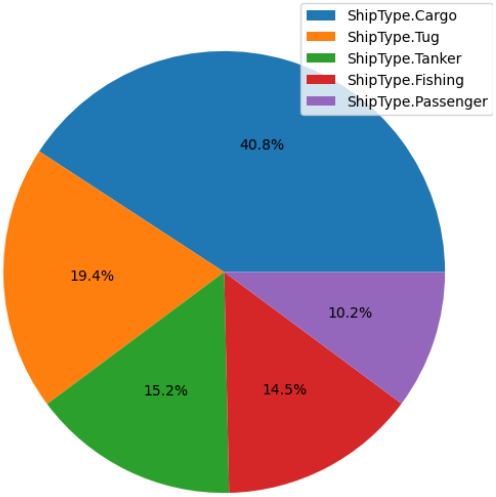


Figura 3-16: Eliminación de incoherencias de calado, eslora y manga. Proporción resultante tras el filtrado.

Existen, además, ciertos valores en el DF que, por su anomalía en el conjunto, deben ser eliminados. Estos valores, denominados *outliers*, pueden influir negativamente al calcular las medias. Para un mejor entendimiento de lo que suponen los *outliers*, será necesario calcular los parámetros de eslora y manga, por lo que se tendrá que procesar antes de seguir con el proceso de limpieza del DF.

3.2.3 Extracción de parámetros

La extracción de valores determinantes de parámetros estáticos ya fue estudiada y, al no ser el objetivo principal de este TFG, se mantendrán los mismos. La función de cálculo de parámetros (Figura 3-17) toma *df_clean_static* (con 7 columnas) y le agrega 11 columnas nuevas.

```
#Función para el cálculo de atributos estáticos
def statics_processing(df):
 df['len'] = df.to_bow+df.to_stern #Eslora
 df['wid'] = df.to_port+df.to_starboard #Manga
 df['ldivw'] = df['len'] / df['wid']#Cociente entre eslora y manga
 df['ldivd'] = df['len'] / df['draught']#Cociente entre eslora y calado
 df['wdivd'] = df['wid'] / df['draught'] #Cociente entre manga y calado
 df['area'] = df.len*df.wid #Area de la cubierta aproximada= eslora x manga
 df['grith'] = df.len+df.wid #Suma de eslora y manga
 df['aml'] =df.len*df.draught #area longitudinal sumergida
 df['amt'] =df.wid*df.draught #area transversal sumergida
 df['vs'] =df.len*df.draught*df.wid # volumen sumergido
 df['aol'] = df['to_bow'] / df['len'] #proporción que supone a sobre la eslora total
 return df
```

Figura 3-17: Función de cálculo de parámetros estáticos

Es ahora cuando podemos estudiar la distribución de mangas, esloras y calados según el tipo de buque y ver los valores anómalos. En las siguientes figuras, se representan las distribuciones de esloras (Figura 3-18), mangas (Figura 3-19) y calados (Figura 3-20) con una longitud de las mechas (en los gráficos de velas, las mechas representan el primer y tercer cuartil del conjunto, mostrando en sus extremos inferior y superior el valor mínimo y máximo respectivamente) muy por encima de los valores medios, ocasionadas por los *outliers*.

La eliminación de *outliers* (Figura 3-21) se replantea, a diferencia de los anteriores TFG, con operaciones vectorizadas de *pandas*, lo que supone una mejora significativa ya que evita el uso de bucles explícitos y aprovecha las capacidades de dicha librería para realizar operaciones a nivel de grupo. Además, se elimina únicamente el primer y el último percentil de datos, reduciendo el DF en solo un 2%, que se muestra en la segunda celda de la Figura 3-21.

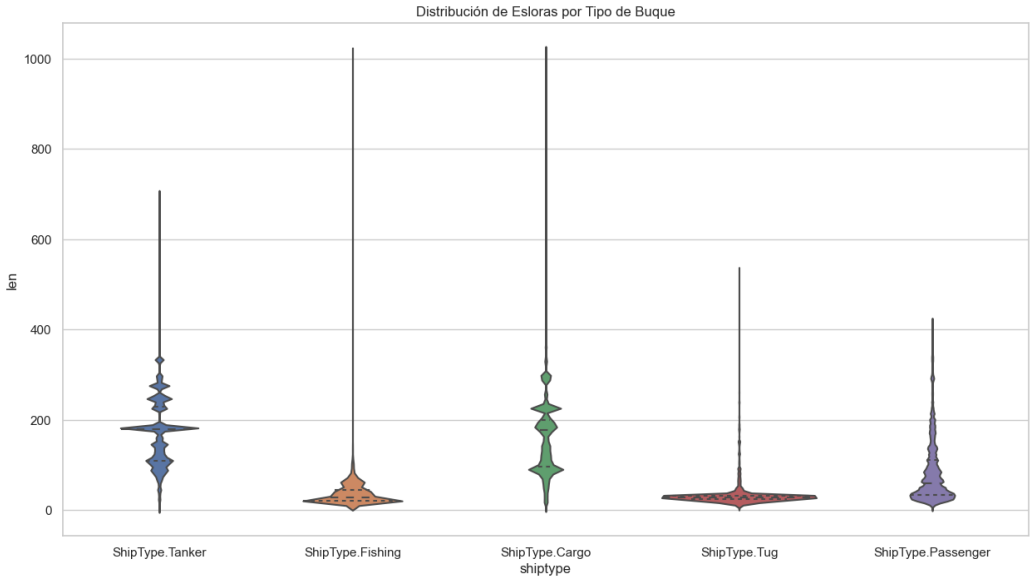


Figura 3-18: Distribución de esloras por tipo de buque

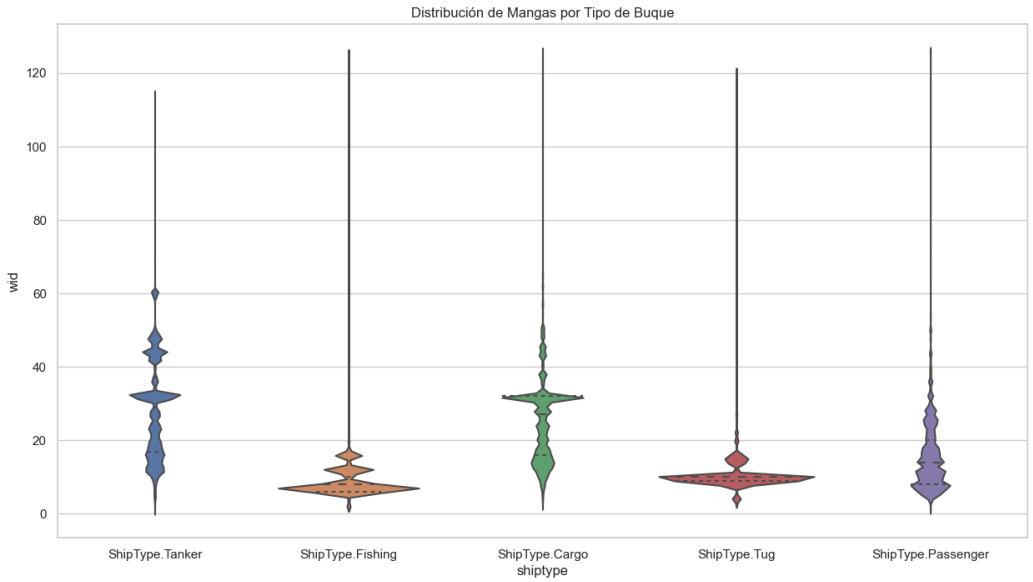


Figura 3-19: Distribución de mangas por tipo de buque

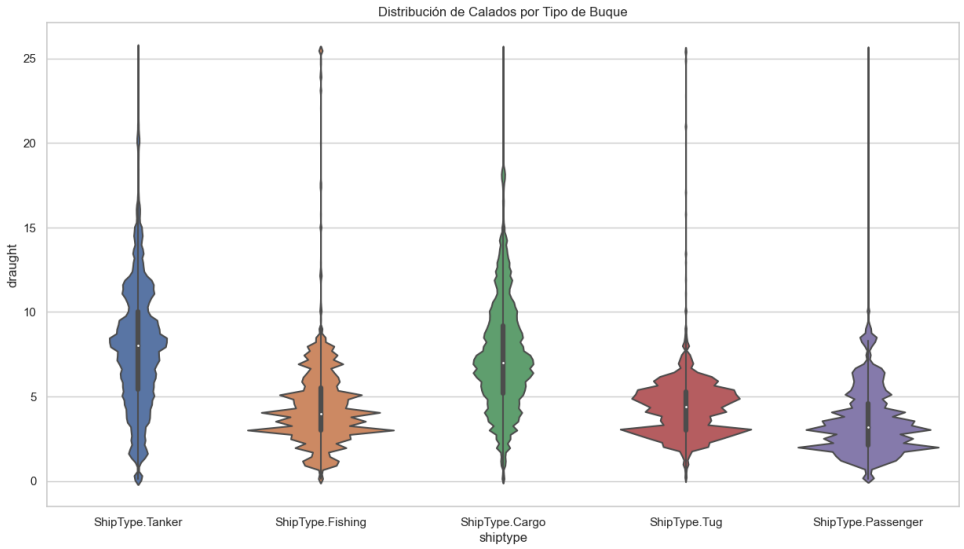


Figura 3-20: Distribución de calados por tipo de buque

```python
atributes = ['len', 'wid', 'draught']

# Calcular cuantiles para los atributos definidos para cada tipo de barco
q_low = df_processed_static.groupby('shiptype')[atributes].transform(lambda x: x.quantile(0.01))
q_high = df_processed_static.groupby('shiptype')[atributes].transform(lambda x: x.quantile(0.99))

# Establecer a NaN los valores por debajo del cuantil del 1% o por encima del cuantil del 99%
df_processed_static[atributes] = df_processed_static[atributes].where(
    (df_processed_static[atributes] >= q_low) & (df_processed_static[atributes] <= q_high)
)
# Eliminar filas con al menos un valor NaN en 'len', 'wid' o 'draught'
df_processed_static = df_processed_static.dropna(subset=atributes, how='any')
```

```python
df_processed_static.shape
```

```
(38947774, 18)
```

Figura 3-21: Eliminación de *outliers* y estructura del DF resultante

Tras la eliminación de *outliers* podemos observar la reducción de las mechas, ya que, eliminando únicamente el primer percentil tanto por encima como por debajo, obtenemos una distribución mucho más centrada. En los gráficos de violines de las Figuras 3-22, 3-23 y 3-24 se observa este cambio en eslora, manga y calado, respectivamente. En el interior del violín también se puede observar una vela que representa valores medios, primer y tercer cuartil y valores máximos y mínimos de la distribución. Las mechas de las velas, esta vez, presentan menor longitud, por lo que los valores anómalos se han descartado satisfactoriamente.

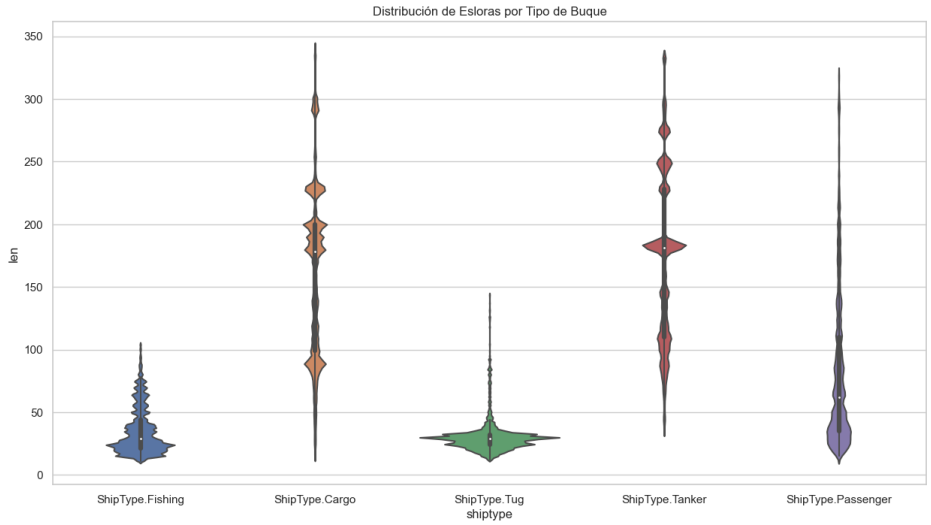


Figura 3-22: Distribución de esloras por tipo de buque tras la eliminación de *outliers*

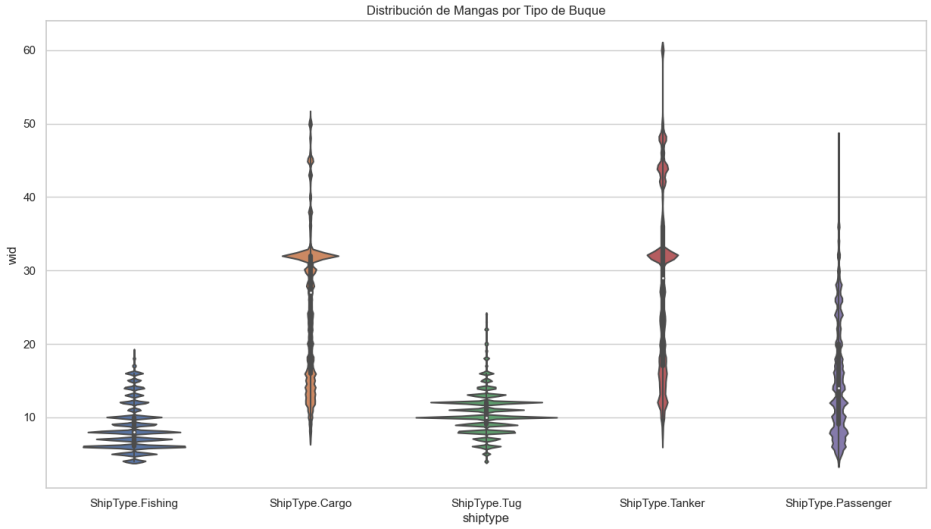


Figura 3-23: Distribución de mangas por tipo de buque tras la eliminación de *outliers*

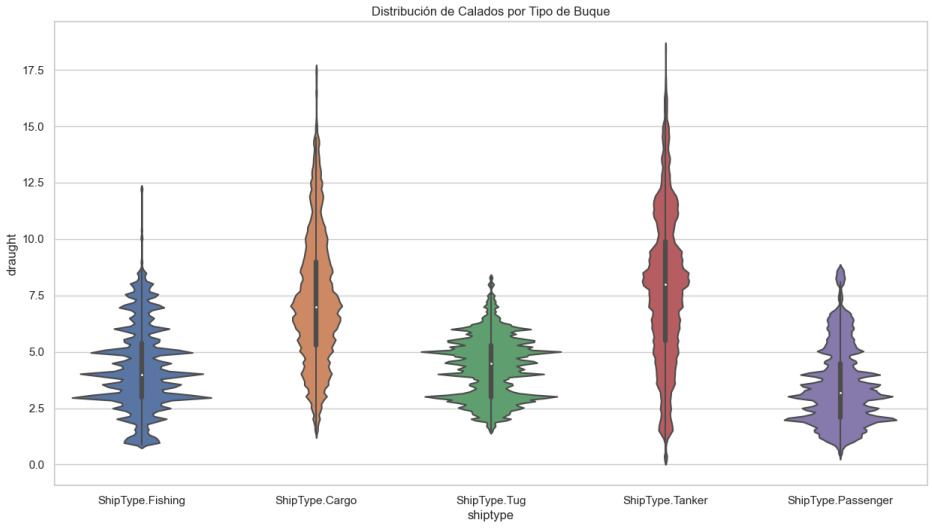


Figura 3-24: Distribución de calados por tipo de buque tras la eliminación de *outliers*

Una vez que se ha comprobado que el DF no posee datos incoherentes ni *outliers* que alteren los valores medios, se puede pasar a calcular las medias de los valores de los campos estáticos. La mayoría de los campos deberían tener esos los valores iguales, sin embargo, por cambio de posición de la antena transmisora o por variaciones en el calado debido a cargas, pueden diferir mínimamente entre ellos, debiéndose mantener ciertas relaciones entre valores. En la Figura 3-25 observamos cómo se reduce *df_processed_static* con casi 39 millones de entradas a *df_final_static* con algo más de 40 mil entradas. Será este último DF sobre el que se iterará para concatenar los datos dinámicos.

```
# Eliminamos el campo shiptype para calcular las medias por MMSI
df_means = df_processed_static.groupby('MMSI')[['draught', 'to_bow', 'to_stern', 'to_port',
                                                'to_starboard', 'len', 'wid', 'ldivw',
                                                'ldivd', 'wdivd', 'area', 'grith',
                                                'aml', 'amt', 'vs', 'aol']].mean().reset_index()
```

```
# Merge en base a MMSI y eliminamos las filas duplicadas.
df_final_static = pd.merge(df_means, df_processed_static[['MMSI', 'shiptype']], on='MMSI', how='left')
df_final_static = df_final_static.drop_duplicates(subset='MMSI').reset_index(drop=True)
```

```
df_final_static.shape
```

```
(40859, 18)
```

Figura 3-25: Cálculo de medias por MMSI y estructura de *df_final_static*

La distribución final de los tipos de buques se muestra en la Figura 3-26. En las Figuras 3-27, 3-28 y 3-29 se muestran, respectivamente, las distribuciones de esloras, mangas y calados con valores medios y únicos para cada MMSI. Como podemos observar, la distribución no varía en forma aun habiéndose reducido significativamente el conjunto de datos, por lo que los valores medios representan fielmente los conjuntos de millones de datos duplicados.

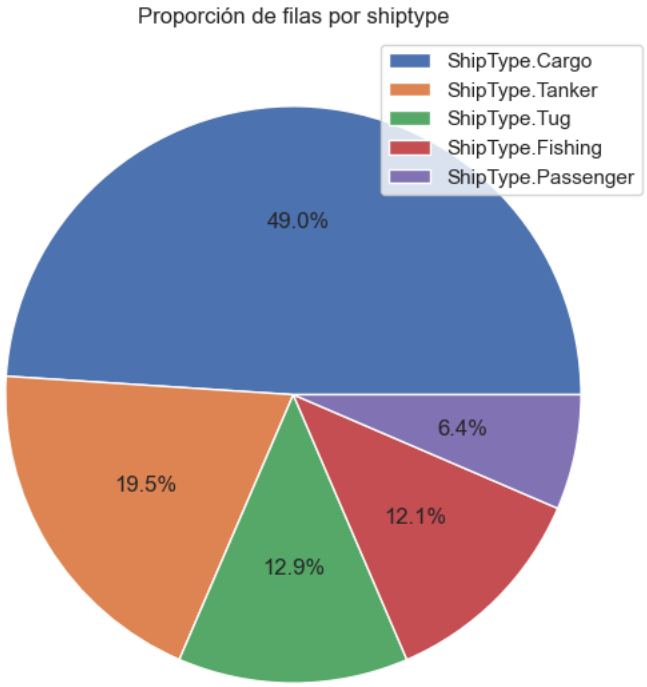


Figura 3-26: Representación de cada *shiptype* en *df_final_static*

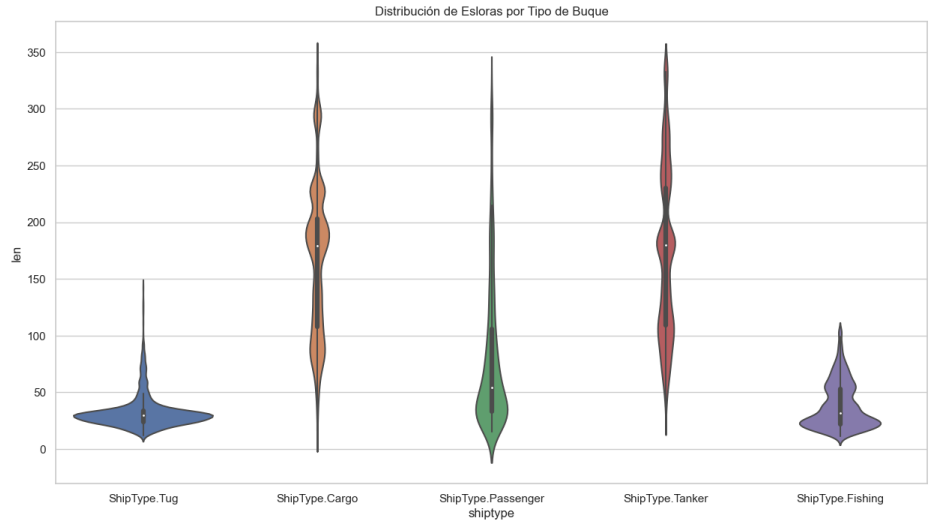


Figura 3-27: Distribución de esloras por tipo de buque tras cálculo de valores medios por MMSI

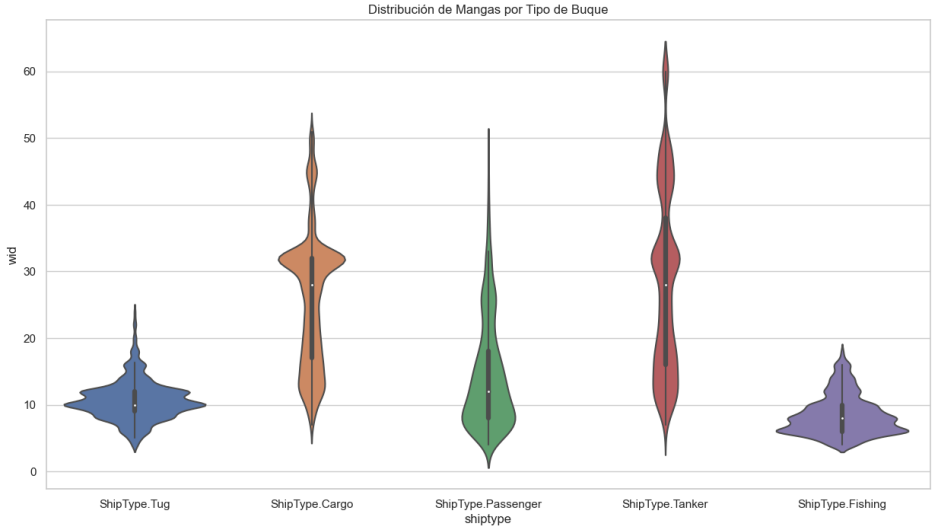


Figura 3-28: Distribución de mangas por tipo de buque tras cálculo de valores medios por MMSI

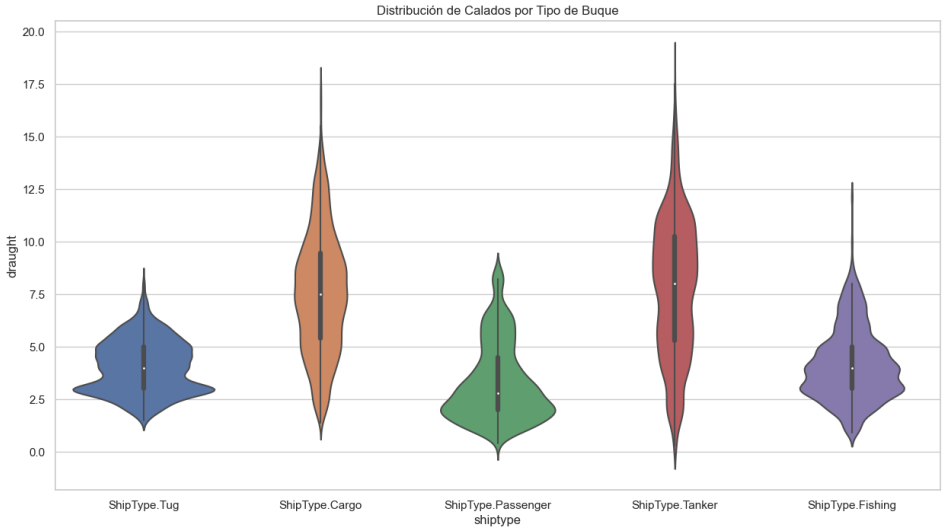


Figura 3-29: Distribución de calados por tipo de buque tras cálculo de valores medios por MMSI

3.3 Preparación de datos dinámicos

El procesado de los datos dinámicos también fue razón de estudio en anteriores trabajos de esta línea de investigación, sin embargo, si bien en [10] se propone el uso de celdas *H3* para mejorar las predicciones, en esta aproximación se propone hacer un uso específico de las posiciones GPS de cada mensaje dinámico, para generar una trayectoria. Aunque se definirá más adelante el concepto de trayectoria, el conjunto de datos de un mes para un solo buque (un único MMSI) genera una colección de trayectorias que deben dividirse para extraer parámetros individuales de cada una de ellas.

3.3.1 Análisis de datos

El análisis de los datos dinámicos presenta más complicaciones. Los datos se han facilitado, además, de una forma distinta, de manera que cada MMSI está aislado en un único fichero que contiene toda la información dinámica de ese buque en ese mes. El análisis de todos los datos dinámicos en su conjunto sería problemático en cuanto a procesamiento ya que el DF recursivo no podría procesarse por falta de capacidad en los equipos empleados. Además, como veremos más adelante, los campos no aportan de manera intuitiva información relevante, y deben ser previamente tratados antes de ser analizados. Los campos de estos mensajes se presentan en la Tabla 3-3.

Campos	Descripción
Timestamp	Hora de recepción del mensaje AIS
MMSI	*Maritime Mobile Service Identity*
lat	Latitud GPS en el instante de transmisión
lon	Longitud GPS en el instante de transmisión
speed	Velocidad en el instante de transmisión
course	Rumbo en el instante de transmisión
heading	Dirección de la proa en el instante de transmisión

Tabla 3-3: Descripción de los campos dinámicos AIS

Como se puede apreciar, los mensajes proporcionados carecen del campo *shiptype*, por lo que el primer paso es asociar cada MMSI a su respectivo *shiptype*. Para ello, utilizaremos el DF final de los datos estáticos, que almacenamos previamente en un fichero CSV y nos quedaremos únicamente con los dos campos de interés: MMSI y *shiptype* (Figura 3-30). A este DF le llamaremos *df_MMSI_shiptype*.

```
df_MMSI_shiptype = (
    pd.read_csv('final_static_data.csv')
    .filter(items=['MMSI', 'shiptype'])
    .sort_values(by='MMSI')
    .reset_index(drop=True)
)
df_MMSI_shiptype.to_csv('MMSI_shiptype_data.csv', index = False)
```

Figura 3-30: Creación de *df_MMSI_shiptype*

Ahora, *df_MMSI_shiptype* contiene una lista de todos los MMSI del mes que tienen asociado un campo *shiptype*. La siguiente operación consiste en leer uno a uno cada fichero dinámico (Figura 3-31), almacenarlo en *df_raw_dynamic*, comprobar que el MMSI existe en *df_MMSI_shiptype* y

combinar ambos datos para asociar el fichero de datos dinámicos a un *shiptype*. Esta operación se incluirá en un futuro bucle, aunque, para analizar trayectorias de manera puntual, utilizaremos este método de lectura y combinación.

```
In [ ]:
# Obtiene la lista de archivos CSV en el directorio
lista_archivos_dynamic = (
    [archivo for archivo in os.listdir('Dynamic_by_MMSI') if archivo.endswith('.csv')])
# Lee el primer archivo CSV y lo almacena en un DataFrame
df_raw_dynamic = pd.read_csv(os.path.join('Dynamic_by_MMSI', lista_archivos_dynamic[1]))
```

Figura 3-31: Lectura del primer archivo CSV de la carpeta de datos dinámicos por MMSI

```
In [ ]:
# Realiza el merge basado en las columnas 'mmsi' y 'MMSI'
df_merged_dynamic = pd.merge(df_raw_dynamic, df_MMSI_shiptype, left_on='mmsi', right_on='MMSI',
                             how='inner')
# Elimina la columna 'MMSI' duplicada
df_merged_dynamic.drop(columns=['MMSI'], inplace=True)
# Elimina filas con valores NaN en la columna 'shiptype'
df_merged_dynamic.dropna(subset=['shiptype'], inplace=True)
#Ordena por timestamp
df_merged_dynamic = df_merged_dynamic.sort_values(by='timestamp')
```

Figura 3-32: Asociación de *shiptype* a *df_raw_dynamic*

Estas dos operaciones irán realizándose para una pequeña cantidad de MMSI distintos con el fin de realizar un análisis específico de cada uno de ellos, simplemente cambiando el número que representa el archivo CSV (fila 5 en la celda de la Figura 3-31). La lectura de datos en el bucle se realiza de manera similar, aunque en lugar de leer la lista de nombres en los CSV de la carpeta, lee el *df_MMSI_shiptype* y busca el archivo asociado al MMSI. Este proceso se detallará en el subapartado *3.3.7*.

Antes de comenzar con la generación de trayectorias para su análisis individual, analicemos parámetros genéricos de los valores dinámicos del mes de enero. El único parámetro de extracción inmediata es la longitud de cada archivo CSV, que se traduce al número de mensajes que cada buque ha transmitido a lo largo del mes. En la Figura 3-33 se muestra el código utilizado para almacenar en un *DataFrame* (*df_longitudes*) la información dimensional de cada fichero y el tipo de buque que ha transmitido ese conjunto de mensajes. En la Figura 3-34, se muestra un diagrama de violines del número de mensajes de cada buque divididos en los 5 tipos de buque de estudio. Como se puede apreciar, el primer cuartil (Q1) de los datos ronda entre los valores 350 y 550, esto quiere decir que alrededor del 20% de los buques han transmitido menos de 300 mensajes. Además, en el percentil 10 de los datos, las muestras rondan una extensión de 100 mensajes.

En todos los experimentos se filtrará solo un 10% de los datos estableciendo un número mínimo de 100 mensajes, ya que, tras haber analizado las cadencias de transmisión, se ha observado que son mucho mayores de lo esperado, por lo que una trayectoria de 100 mensajes podría definir perfectamente una ruta de 100 horas, dado que la cadencia de transmisión sería de 1 mensaje por hora. Esta baja cadencia de transmisión resultará problemática para distinguir entre trayectorias fiables y no fiables, ya que una cadencia de mensajes mayor a 1 hora disminuye sensiblemente la precisión de trazado de una ruta.

```python
# Inicializa un DataFrame para almacenar las longitudes según el shiptype
data = {'shiptype': [], 'longitud': []}

for MMSI, row in df_MMSI_shiptype.iterrows():
    # Busca el archivo CSV correspondiente al MMSI actual
    csv_file = os.path.join('Dynamic_by_MMSI', f"dynamic_{row['MMSI']}.csv")

    # Verifica si el archivo CSV existe
    if os.path.exists(csv_file):
        df_raw_dynamic = pd.read_csv(csv_file)

        # Calcula la longitud del DataFrame actual y añádela al DataFrame de datos
        shiptype = row['shiptype']
        longitud_df = len(df_raw_dynamic)
        data['shiptype'].append(shiptype)
        data['longitud'].append(longitud_df)

# Crea un DataFrame a partir de los datos recopilados
df_longitudes = pd.DataFrame(data)
```

Figura 3-33: Creación de *df_longitudes*

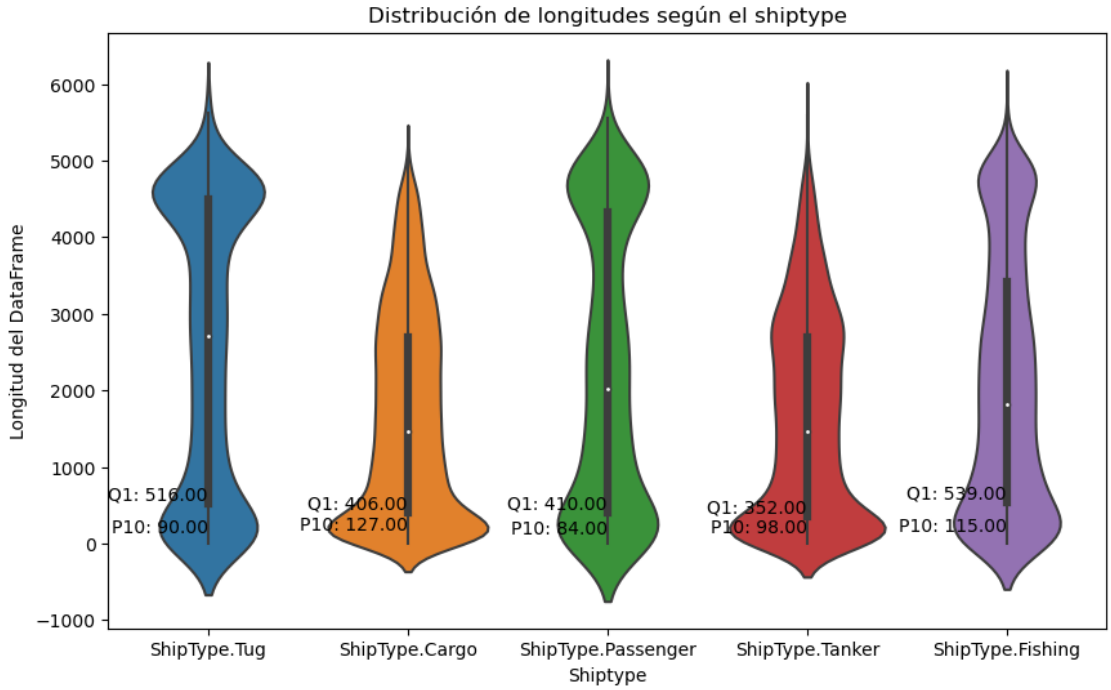


Figura 3-34: Distribución de extensiones mensuales según el campo *shiptype*

Para la transformación del *df_merged_dynamic* en un conjunto de trayectorias, se utilizarán las bibliotecas *geopandas* (para crear un *geodataframe* opcionalmente) y *movingpandas* (para crear trayectorias a partir de puntos GPS). En la Figura 3-35 podemos observar cómo se crea la trayectoria del mes a partir de la librería. Nótese que el parámetro *timestamp* se ha transformado de formato UNIX a un formato *datetime* y que la columna de *geometry* se ha generado a partir de copias de latitud y longitud para generar un punto geométrico con datum WGS84 (datum universal de GPS con crs 4326). Al mostrarlo por pantalla (línea 5 de Figura 3-35) obtenemos un objeto *mpd.Trajectory* que no es más que la ruta completa del buque. Como podemos observar, la ruta comprende los datos del 31 de diciembre de 2022 al 31 de enero de 2023 y tiene una longitud de unos 700 km. Con el comando *.df*

podemos observar el objeto en formato tabla. Este comando será de gran utilidad para la extracción de ciertos parámetros.

```
%%time
# Genera las trayectorias sin pasar por un gdf, le pasamos los parámetros directamente
month_trajectories = mpd.Trajectory(df_merged_dynamic,'Remolcador 1',
                                    t='timestamp', x='lon_copy', y='lat_copy', crs=4326)
print(month_trajectories)
month_trajectories.df
```

```
Trajectory Remolcador 1 (2022-12-31 22:57:51 to 2023-01-31 22:51:24) | Size: 4446 | Length: 725152.3m
Bounds: (123.6318, 8.575555, 124.759902, 9.617302)
LINESTRING (124.75866 8.57808, 124.75867 8.578085, 124.758668 8.578097, 124.75869 8.578082, 124.7586
CPU times: total: 359 ms
Wall time: 408 ms
```

timestamp	mmsi	lat	lon	speed	course	heading	shiptype	geometry	traj_id
2022-12-31 22:57:51	100003661	8.578080	124.758660	0.0	37.9	511	ShipType.Tug	POINT (124.75866 8.57808)	Remolcador 1
2022-12-31 23:04:17	100003661	8.578085	124.758670	0.0	37.9	511	ShipType.Tug	POINT (124.75867 8.57808)	Remolcador 1
2022-12-31 23:09:18	100003661	8.578097	124.758668	0.0	37.9	511	ShipType.Tug	POINT (124.75867 8.57810)	Remolcador 1
2022-12-31 23:14:20	100003661	8.578082	124.758690	0.0	37.9	511	ShipType.Tug	POINT (124.75869 8.57808)	Remolcador 1
2022-12-31 23:19:30	100003661	8.578080	124.758643	0.0	37.9	511	ShipType.Tug	POINT (124.75864 8.57808)	Remolcador 1

Figura 3-35: Creación de *month_trajectories* del Remolcador 1 y muestra de *df_month_trajectories*

Veamos ahora la trayectoria de este buque sobre un mapa. En la Figura 3-36 (imagen de la izquierda) se muestra el recorrido del buque a lo largo del mes con una leyenda colorimétrica de velocidades y en la imagen de la derecha de la Figura 3-36 se muestra ampliada la zona con mayor actividad. Este buque remolcador (*Remolcador 1*) que opera en las islas Filipinas nos ofrece una información de gran valor acerca de su clase, pues, por norma general, los remolcadores realizan numerosas trayectorias a lo largo de todo un mes a baja velocidad y cubriendo distancias relativamente cortas. En el apartado de discretización de trayectorias, veremos cómo podemos discernir entre los momentos en los que inicia una nueva ruta utilizando un detector de paradas ya que en este momento, el objeto *mpd.Trajectory* contiene una única trayectoria que comprende tanto estados de parada como estados de navegación durante todo el mes.

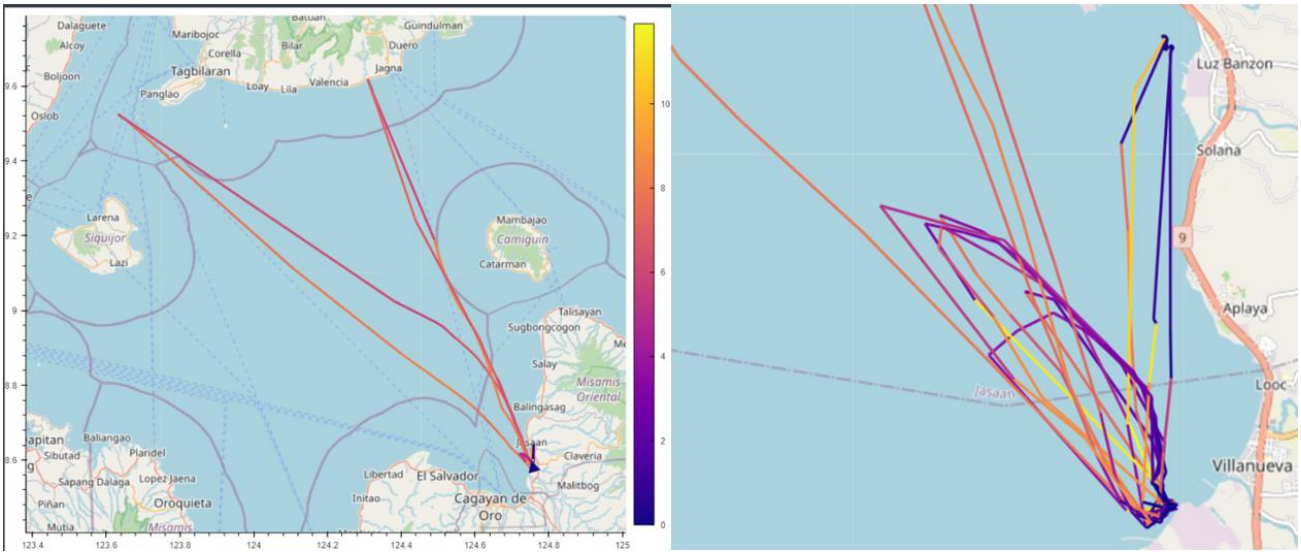


Figura 3-36: Trayectoria de enero de 2023 del *Remolcador 1*

Realizando las mismas operaciones de lectura y fusión, podemos representar gráficamente un pequeño conjunto de trayectorias para analizar las características representativas de cada buque. En las imágenes de la Figura 3-37 se muestra la trayectoria de un petrolero (*Petrolero 1*), con velocidades elevadas, que bordea la costa oeste de Centroamérica finalizando el mes con un hipódromo de espera para, posiblemente, cruzar el Estrecho de Panamá. En la Figura 3-38 se muestra la trayectoria de un carguero (*Carguero 1*) que realiza un tránsito entre Noruega, Finlandia, Agadir (Marruecos) y Málaga a una velocidad constante de entre 10 y 12 nudos.

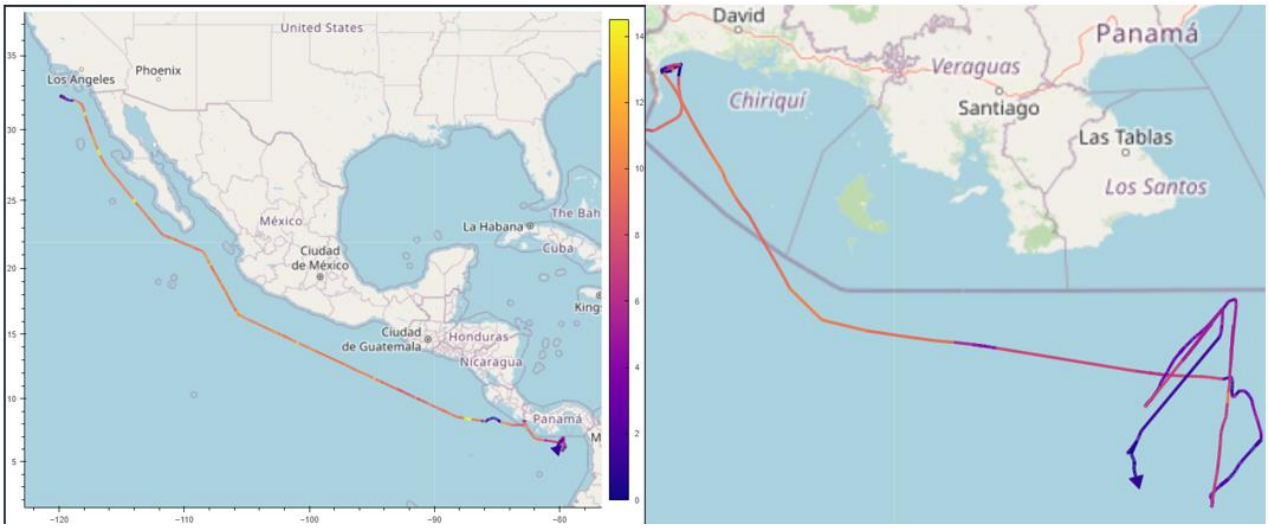


Figura 3-37: Trayectoria de enero de 2023 de *Petrolero 1*

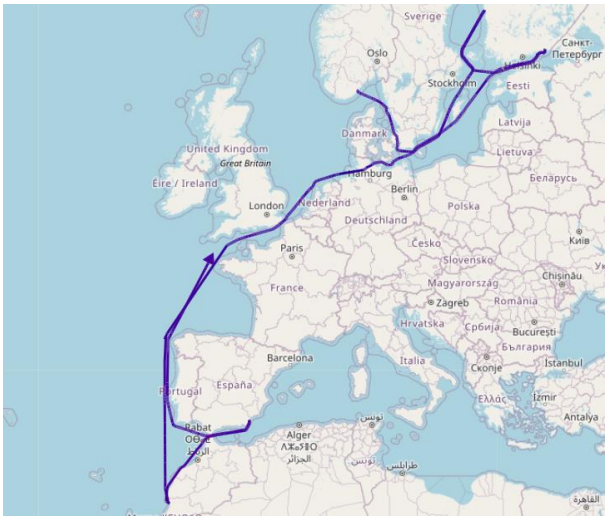


Figura 3-38: Trayectoria de enero de 2023 de *Carguero 1*

3.3.2 Discretización de trayectorias

Hasta ahora, la metodología que se ha seguido ha consistido en la limpieza de datos y la extracción de parámetros a partir de los mensajes AIS. Los parámetros extraídos a partir de los datos dinámicos en anteriores TFG consideraban valores estadísticos como velocidades o aceleraciones medias y sus varianzas, entre otros. Estos datos pueden llegar a ser determinantes para el algoritmo, pero deben ser tratados de una manera muy específica. En [10] se presenta un enfoque utilizando celdas *H3*, tal y como se vio en el último apartado del estado del arte. Este enfoque presentó mejoras sustanciales en las predicciones de ciertos tipos de buques. En este TFG el enfoque planteado es la división en trayectorias, discretizando los momentos en tránsito del buque de los momentos de parada del mismo y extrayendo parámetros de ambas situaciones por separado.

En este apartado, se explicará una de las aportaciones principales de este TFG. El uso de los datos dinámicos para la generación de trayectorias permite extraer parámetros característicos desde otro punto de vista. La discretización de las trayectorias consiste en aislar trayectorias independientes a partir de la trayectoria mensual almacenada como una sola. Para ello, es necesario definir ciertos conceptos.

En primer lugar, definiremos trayectoria como el conjunto ordenado en el tiempo de mensajes que un mismo buque transmite entre dos estados de parada. Las paradas se considerarán como tal cuando un flujo continuo de mensajes presenta durante un tiempo definido posiciones GPS dentro de un área de diámetro determinado. Estas paradas serán la transición entre el final de una trayectoria y el comienzo de la siguiente. Por último, la trayectoria mensual (*month_trajectories*) viene definida como la unión de los puntos GPS contenidos en el DF con información dinámica de un mes de un solo MMSI, esta información podrá contener estados de navegación o estados de parada.

En la Figura 3-39 podemos observar un ejemplo gráfico que ayuda a comprender las anteriores definiciones. Como se puede observar, los puntos muestran la posición GPS en el momento de la transmisión del mensaje, y el color de los mismos representa el paso del tiempo. Cuando existe una gran diferencia de tiempo entre dos mensajes con posiciones GPS muy cercanas, el algoritmo establece una parada. Este método de discretización es independiente del flujo de mensajes ya que solo tiene en cuenta el tiempo entre mensajes y la distancia entre los puntos desde los que fueron transmitidos. Gracias a eso podemos discretizar las trayectorias de buques cuya latencia de transmisión sea muy baja.

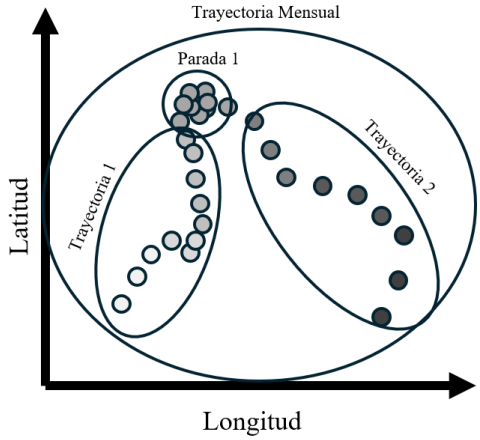


Figura 3-39: Ejemplo gráfico de la discretización de trayectorias

Comenzando con el ejemplo del *Carguero 1*, en la Figura 3-40 se muestran los estados de parada que se han detectado utilizando como parámetros umbrales un diámetro de 100 metros y 1 hora. Esto quiere decir que si existe un conjunto de mensajes consecutivos encerrados en un área circular cuyo diámetro es de 100 metros durante más de una hora, se considerará una parada. Estos parámetros se han establecido de forma empírica, en base al estudio de rutas individuales de distintos tipos de buques, observando cuáles son los que mejor detectan paradas reales en puerto o fondeados. El uso de un diámetro de 100 m se estableció inicialmente para asegurar la detección de paradas aun cuando el AIS transmite una posición imprecisa. Sin embargo, este umbral puede detectar paradas cuando el buque transita a una velocidad muy baja durante mucho tiempo, por lo que en futuros experimentos (en particular después de la segunda iteración) se utilizará un umbral de 50 metros, manteniendo un tiempo mínimo de 1 hora.

Figura 3-40: Puntos de parada del *Carguero 1*

En la Figura 3-41, se muestra el código que genera los vectores de paradas y los gráficos resultantes de los mismos. Estos vectores consisten en la "trayectoria" que ha seguido el buque estando parado, que, aunque en un caso ideal debería ser un punto, no lo es ya que el buque se ve afectado por el efecto del viento y la mar estando fondeado y por los errores de transmisión del AIS. Como se puede observar en la Figura 3-41, existen 3 vectores de parada que se han detectado de manera errónea, ya que el buque está en plena navegación entre puertos, pero ha reducido su velocidad y ha comenzado a hacer diversas maniobras que han hecho que el detector lo identifique como parada. Para solucionar este problema, cambiamos el anillo de parada a 50 metros, una distancia considerable, aunque no tan permisiva, para evitar la partición de trayectorias en paradas inexistentes. En la Figura 3-42 observamos los gráficos de las paradas detectadas con el nuevo umbral que aparece en el código de la misma figura.

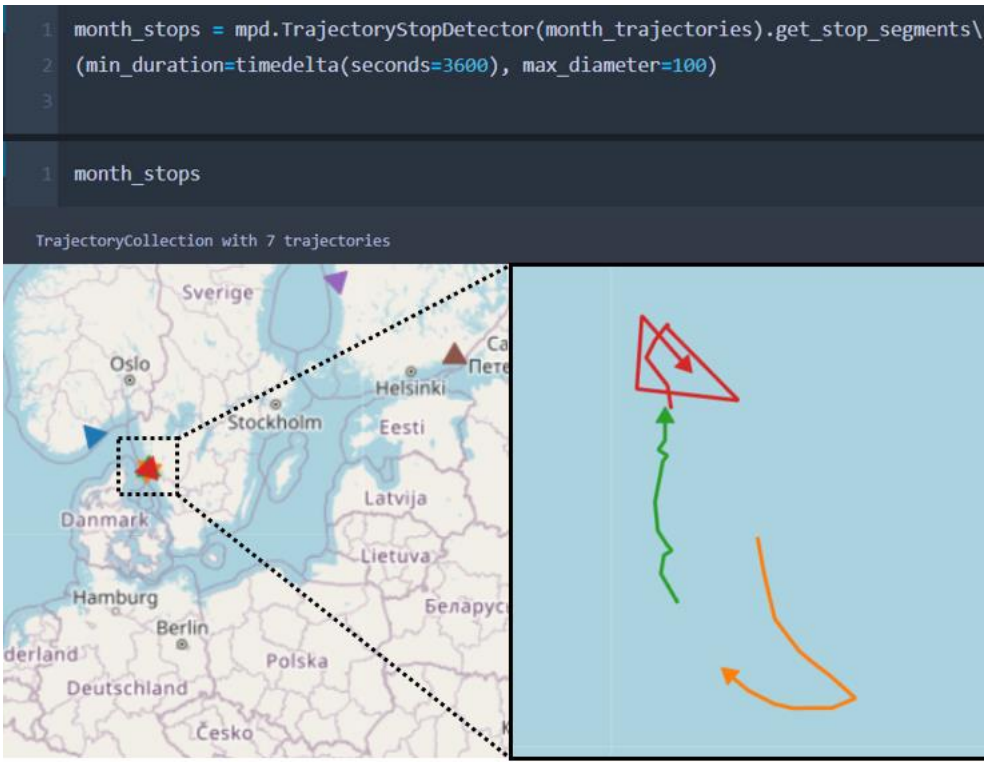


Figura 3-41: Extracción de vectores de parada con umbrales 3600 s y 100 m

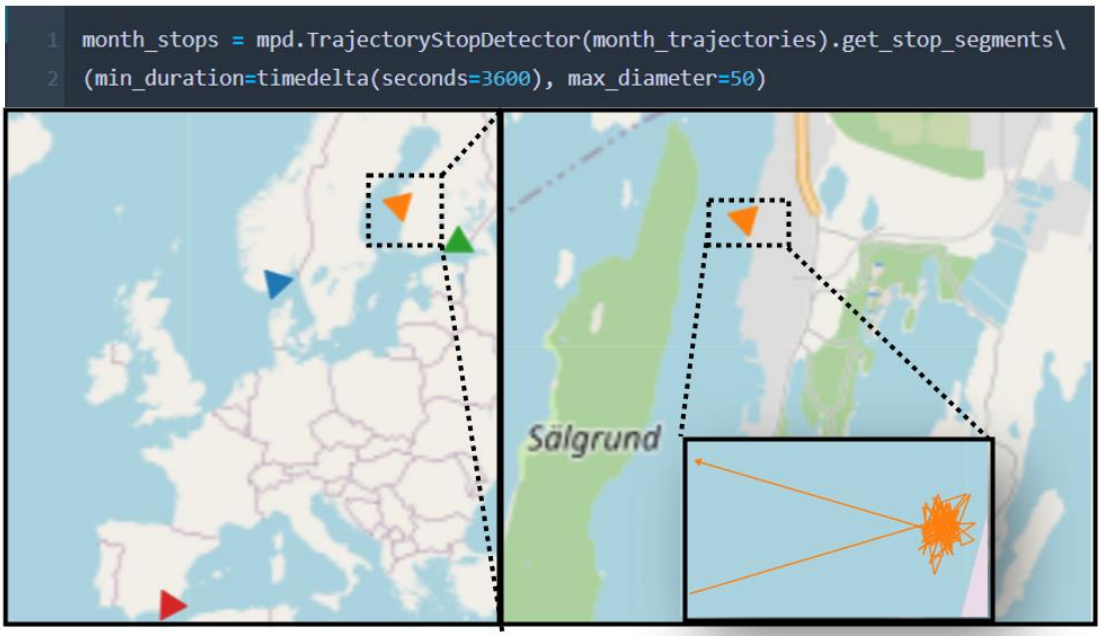


Figura 3-42: Vectores de parada con diámetro umbral de 50 m. Ampliación de la parada en Kaskinen, Finlandia

En la Figura 3-43 se muestra el mismo proceso de extracción de vectores de paradas para el ejemplo del *Remolcador 1*. Como podemos observar, el número de paradas es francamente mayor al del *Carguero 1*, presentando un número total de 39 trayectorias. En la Figura 3-43, además, se detecta una parada no real en medio del mar. Este comportamiento resulta normal en los remolcadores, ya que este tipo de buque suele permanecer parado durante largos períodos de tiempo a la espera de recibir a otro buque para remolcarle.

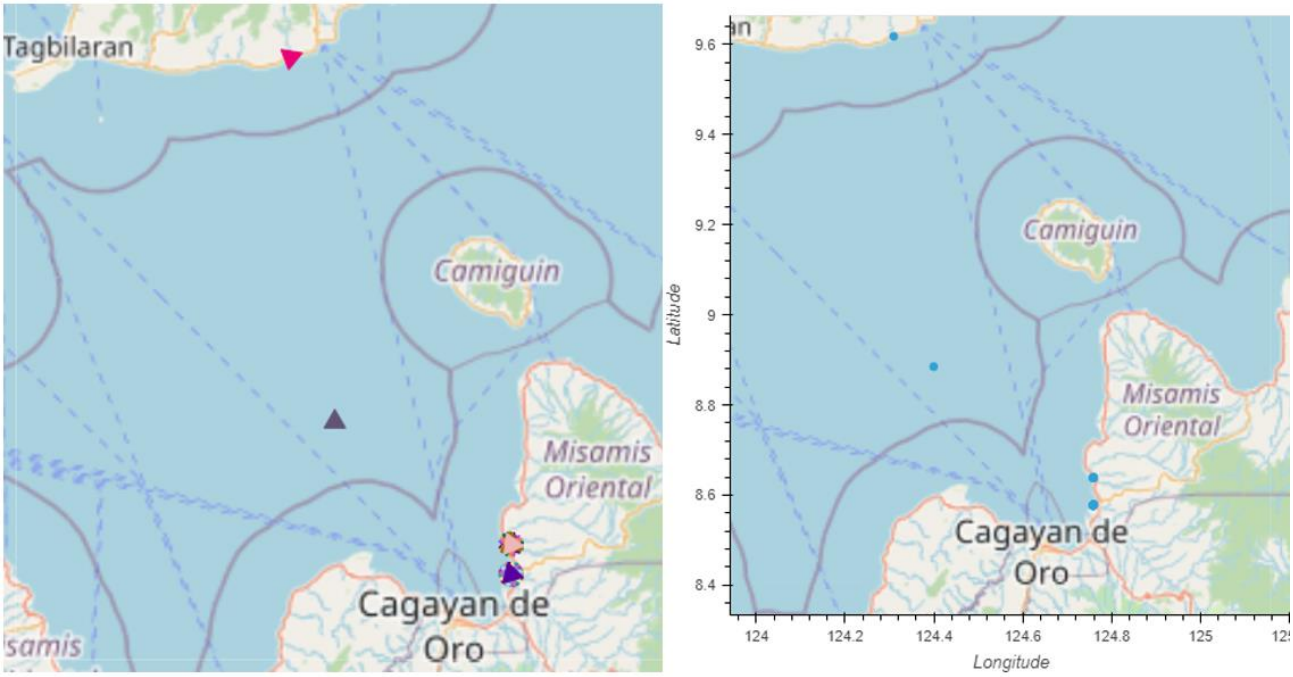


Figura 3-43: Paradas del *Remolcador 1*

Tras la detección de las paradas, se realiza una partición de la trayectoria mensual que consiste en restarle las paradas a la propia trayectoria mensual de manera que quedan discretizadas las trayectorias individuales. En la Figura 3-44 observamos las trayectorias realizadas durante el mes de enero por el *Carguero 1*. Como se puede observar, la información mostrada presenta únicamente los datos en navegación. Asimismo, se muestran en la Figura 3-45 las rutas generadas para el *Remolcador 1*.

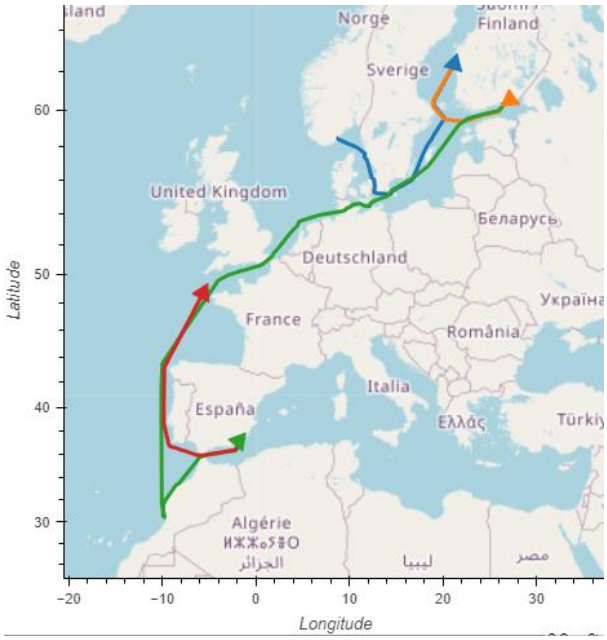


Figura 3-44: Trayectorias del *Carguero 1*

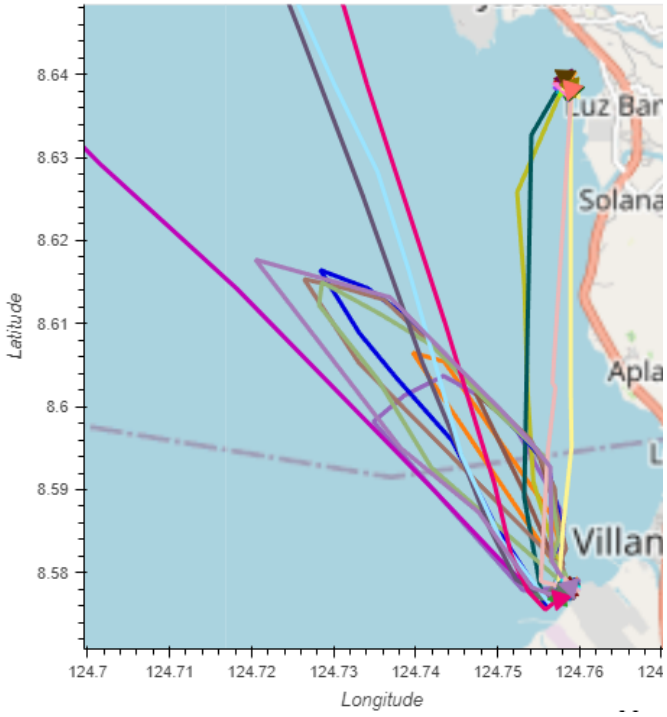


Figura 3-45: Trayectorias del *Remolcador 1* (umbrales de 50 m y 1 hora)

En definitiva, en este apartado, tras la discretización de la trayectoria mensual, nos encontramos, para cada MMSI, con tres objetos: el primer objeto *mpd.Trajectory* contiene la ruta completa del buque a lo largo de todo el mes. El segundo objeto del tipo *mpd.TrajectoryCollection* contiene un conjunto de trayectorias que representan los estados de parada del buque mostrados en las Figuras 3-41, 3-42 y 3-43. El tercer y último objeto se trata de otra colección de trayectorias, esta vez aquellas que contienen los estados en tránsito del buque. El primer objeto se llamará *month_trajectories*, el segundo *month_stops* y el tercero *month_separated_trajectories*. Del primer objeto se obtendrá información genérica del mes, del segundo objeto se obtendrá información de cada parada realizada en ese mes (*df_stop_parameters*) y del tercer objeto se obtendrá información de cada trayectoria individual de ese mes (*df_traj_parameters*). La combinación de los DF habiendo previamente reducido

la dimensionalidad de los dos últimos DF mencionados, resultará en una única fila de datos (para cada MMSI) contenida por *df_cinematic_parameters.* En la Figura 3-46 se muestra un esquema de la combinación de los DF, con ejemplos de parámetros extraídos de cada DF para mejor comprensión. En los subapartados 3.3.4 y 3.3.5 se explicarán los parámetros que se han escogido de cada objeto para alimentar al algoritmo.

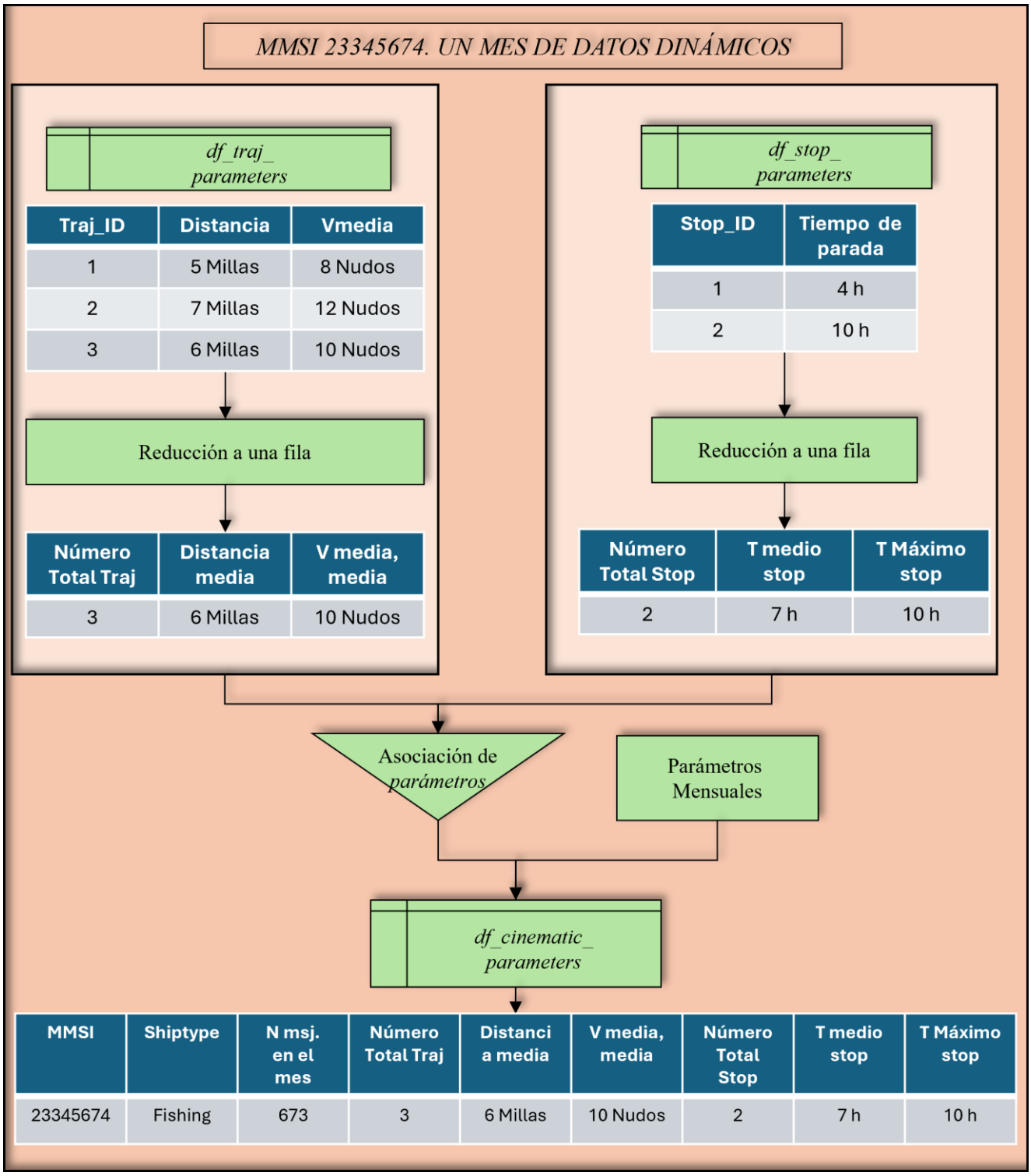


Figura 3-46: Esquema de obtención de parámetros cinemáticos por MMSI

3.3.3 Preprocesado de datos

Como en el apartado de la preparación de estáticos, el preprocesado de datos viene de la mano del análisis de los mismos, sin embargo, este paso se va realizando a lo largo de todo el bucle. Inicialmente, se realiza un proceso de limpieza y preparación de *df_merged_dynamic*, con la función *def clean*, mostrada en la Figura 3-47. Con esta limpieza inicial, se elimina la columna *mmsi* (en minúscula) duplicada a causa de la combinación de los datos estáticos y dinámicos, se ordenan los mensajes temporalmente y se filtran los valores incoherentes de latitud y de longitud. Además, se realiza una copia de estos dos campos para alimentárselo, como se ha visto en el subapartado anterior, al procesador de trayectorias sin perder del DF los valores en crudo de latitud y longitud, ya que servirán para la extracción de ciertos parámetros.

```python
def clean(df):
    # Eliminar columna 'MMSI'
    df.drop(columns=['mmsi'], inplace=True)
    # Ordenar por timestamp
    df.sort_values(by='timestamp', inplace=True)
    # Convertir timestamp a datetime
    df['timestamp'] = pd.to_datetime(df['timestamp'], unit='s')
    # Reemplazar valores de latitud fuera de rango con NaN
    df['lat'] = df['lat'].where((df['lat'] >= -90) & (df['lat'] <= 90), np.nan)
    # Reemplazar valores de longitud fuera de rango con NaN
    df['lon'] = df['lon'].where((df['lon'] >= -180) & (df['lon'] <= 180), np.nan)
    # Hacer copias de lat y lon
    df = df.assign(lat_copy=df['lat'], lon_copy=df['lon'])
    return df
```

Figura 3-47: Limpieza del DF inicial

En el apartado de análisis de datos, se muestra un diagrama de velas de las longitudes de los DF en la Figura 3-34. El primer paso del preprocesado del bucle será comprobar que el DF contiene información suficiente. Como se detalló anteriormente, se establece un mínimo de 100 mensajes por buque para considerar una ruta correctamente definida. De esta manera, se eliminan alrededor del 10% del total de los buques, los cuales, carecen de información suficiente para definir una ruta.

Tras el análisis de los primeros parámetros, se observa que la cadencia de transmisión media de los mensajes es de al menos 6 minutos, alcanzando tasas de transmisión de días. Esta baja calidad del conjunto de datos impedirá realizar filtrados más exhaustos, ya que se reduce el conjunto de datos en gran medida. Sin embargo, se proponen distintos filtrados de la información dinámica para, contando con un *dataset* de calidad y con alta cadencia de transmisión, generar rutas sin incoherencias que evite la generación de *outliers* durante la extracción de parámetros. Estos procesos de limpieza se han dejado como base teórica para futuros trabajos, sin embargo, no se han implementado en este trabajo ya que reducían significativamente el conjunto de datos. En el Anexo IV, en el código *prep_dynamics*, queda reflejado el apartado de compresión de rutas, un generalizador de tiempos que igualaría todas las tasas de transmisión al tiempo deseado, reduciendo significativamente el tamaño de los datos. Existen, además otras bibliotecas de compresión de rutas que podrían realizar un mejor trabajo. De nuevo, todas estas iniciativas son viables para tasas de transmisión de pocos segundos.

Por tanto, el problema de tasa máxima de transmisión queda resuelto, ya que en nuestro *dataset* posee tasas de transmisión muy bajas, siendo la máxima de 6 minutos entre mensajes, y la mínima varios días. El problema de la tasa mínima deriva en la calidad de la trayectoria y se considera que es imposible definir una trayectoria cuyos mensajes están separados en el tiempo más de 10 horas. Este problema se soluciona, por tanto, con la longitud del DF, ya que un buque que transmite durante 30

días cada 10 horas generaría un DF de 72 filas, siendo descartado por el filtro de longitud de DF mínimo de 100 mensajes. En un caso en el que el buque transmite cada 6 minutos durante menos de 10 horas, a pesar de tener una cadencia alta de transmisión, no cumpliría con el filtro de mensajes ya que habría transmitido menos de 100 mensajes. Esto nos asegura que las rutas de todo un mes deben definirse, no sólo con alta cadencia de transmisión, sino durante un tiempo prolongado.

Además, existen otro tipo de filtrados que se pueden aplicar a rutas de mayor calidad, como la eliminación de *outliers* de velocidad real, calculando la velocidad que existe entre dos puntos y eliminando los picos de velocidad que se deben a errores de transmisión. En la Figura 3-48 se aprecia como se esclarece la trayectoria de este petrolero. Este filtrado puede resultar útil para mensajes duplicados en los que, como vemos en el mapa de la izquierda de dicha figura, la estación repetidora transmite su posición GPS, creándose una ruta no real entre el buque y la estación repetidora.

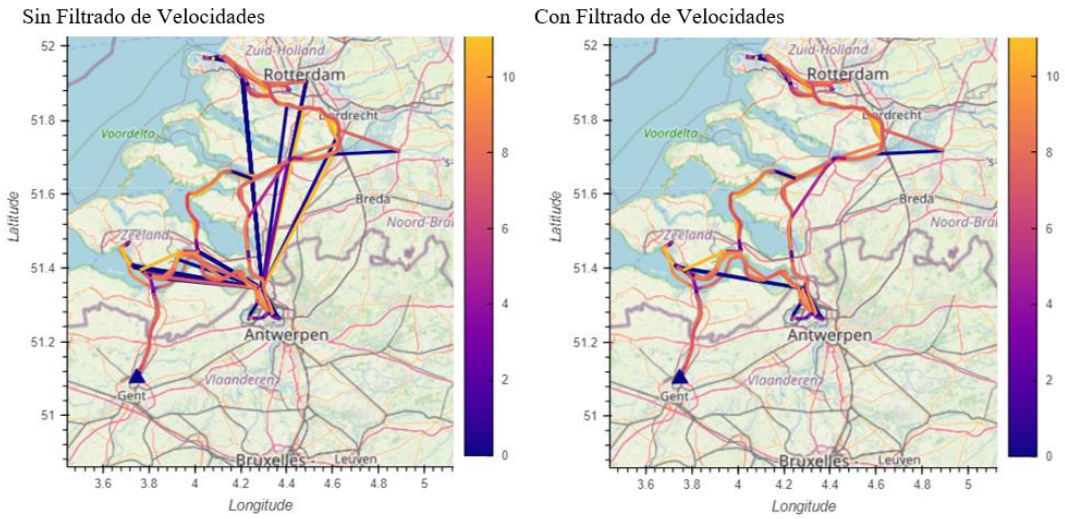


Figura 3-48: Filtrado de *outliers* de velocidad real

3.3.4 Extracción de parámetros de trayectorias

La extracción de parámetros de trayectorias presentó numerosas dificultades que se han tenido que resolver en los apartados anteriores. En primer lugar, se aisló cada conjunto de datos en base a su MMSI de manera que se obtuvo una matriz con posiciones y parámetros de dirección y rumbo para un mismo buque. En segundo lugar, se aisló cada una de las trayectorias del mismo buque, utilizando las bibliotecas mencionadas en el subapartado de discretización de trayectorias y explicadas en el apartado de entorno de trabajo. Por último, una vez extraídos los parámetros de cada trayectoria individualmente, se agruparon estos parámetros en valores estadísticos. Por ejemplo, para el cálculo de la velocidad media de un MMSI, se propone el cálculo de la velocidad media de cada trayectoria y un posterior cálculo de la media de todas las velocidades medias. Este método parece no aportar información adicional inicialmente, sin embargo, ya solo la modelización de las trayectorias eliminaría los mensajes en los que el buque está en un estado de parada (con velocidad 0 o muy cercana a ella): amarrado o fondeado, que pueden contaminar las medias. Además, se propone el mismo análisis para los estados de parada, que será detallado en el siguiente apartado.

La extracción de los parámetros de trayectorias vendrá dada por la función *calcular_parametros_trayectoria*, que toma como único objeto la colección de trayectorias *month_separated_trayectories*. En este subapartado (y en los dos posteriores) se explicarán inicialmente los parámetros escogidos en el primer experimento y después, se irán explicando las modificaciones que se realizaron para disminuir el número de parámetros y añadir o modificar otros, según el avance de los experimentos. Debe aclararse que la mayoría de los cambios se han realizado en base a los resultados del experimento anterior, pero, para mantener una clara estructura en la memoria,

se dejarán para la discusión de resultados las justificaciones que supusieron un cambio en el experimento siguiente; tanto en extracción de parámetros, como en el preprocesado de los mismos.

Para comprender las fases en las que se ha desarrollado este trabajo, los experimentos se dividirán en iteraciones. Estas iteraciones representan un cambio en la extracción de parámetros o en el preprocesado de los datos. Por ejemplo, en la primera iteración, se realizan varios experimentos: primera iteración de enero, primera iteración de febrero, primera iteración global… Todas ellas, presentan el mismo método de extracción de parámetros, pero utilizan distintos datos de entrada.

En la primera iteración, se proponen los parámetros que se muestran en la Figura 3-49.

```
def calcular_parametros_trayectoria(month_separated_trajectories):
# DataFrame vacío con las columnas correspondientes
    traj_parameters = ['traj_N',
                       'traj_max_lat', 'traj_min_lat', 'traj_mean_lat',
                       'traj_max_lon', 'traj_min_lon', 'traj_mean_lon',
                       'traj_max_dif_lat', 'traj_max_dif_lon',
                       'traj_max_sep', 'traj_dist',
                       'traj_dist_0-0.5', 'traj_dist_0.5-1', 'traj_dist_1-1.5',
                       'traj_dist_1.5-2', 'traj_dist_2-5', 'traj_dist_5-10',
                       'traj_dist_10-20', 'traj_dist_>20',
                       'traj_time', 'traj_msg_rate',
                       'traj_time_at_speed_0', 'traj_time_over_speed_0',
                       'traj_mean_time_between_msg', 'traj_max_time_between_msg',
                       'traj_min_time_between_msg',
                       'traj_mean_speed', 'traj_max_speed', 'traj_min_speed',
                       'traj_mean_speed_var', 'traj_max_speed_var', 'traj_min_speed_var',
                       'traj_max_acc', 'traj_min_acc', 'traj_mean_acc',
                       'traj_max_dec', 'traj_min_dec', 'traj_mean_dec',
                       'traj_max_c_h_error', 'traj_min_c_h_error', 'traj_mean_c_h_error',
                       'traj_max_course_rate', 'traj_min_course_rate', 'traj_mean_course_rate',
                       'traj_max_head_rate', 'traj_min_head_rate', 'traj_mean_head_rate']

    df_traj_parameters = pd.DataFrame(columns=traj_parameters)
```

Figura 3-49: Parámetros en la primera iteración

Los parámetros calculados en este código se dividen en varios grupos, cada uno proporcionando información específica sobre las trayectorias geoespaciales analizadas. Estos grupos incluyen parámetros geográficos, aspectos temporales, velocidades, aceleraciones y deceleraciones, errores de rumbo/dirección y tasas de cambio de rumbo y dirección. A continuación, se detallará cada grupo y sus parámetros asociados.

Parámetros geográficos:

Estos parámetros proporcionan una descripción detallada de la extensión geográfica y la distancia recorrida en la trayectoria, lo que permite una comprensión completa de su cobertura espacial.

- *traj_max_lat*: latitud máxima alcanzada en la trayectoria.
- *traj_min_lat*: latitud mínima alcanzada en la trayectoria.
- *traj_mean_lat*: latitud media de la trayectoria.
- *traj_max_lon*: longitud máxima alcanzada en la trayectoria.
- *traj_min_lon*: longitud mínima alcanzada en la trayectoria.
- *traj_mean_lon*: longitud media de la trayectoria.

- *traj_max_dif_lat*: diferencia entre la latitud máxima y la latitud mínima de la trayectoria, es decir, diferencia de latitudes máxima de la trayectoria.
- *traj_max_dif_lon*: diferencia entre la longitud máxima y la longitud mínima de la trayectoria, es decir, diferencia de longitudes máxima de la trayectoria.
- *traj_max_sep*: máxima separación entre puntos no necesariamente consecutivos de la trayectoria. Este parámetro está relacionado con la diferencia de latitudes y longitudes máximas, pero teniendo en cuenta las dos a la vez. Este parámetro tuvo que desecharse en la primera iteración ya que generaba elevados tiempos de procesamiento durante el estudio de rendimiento temporal. Sin embargo, los valores de diferencia de latitud y longitud ya ofrecen una visión de la dimensión espacial de la ruta.
- *traj_dist*: distancia total recorrida en la trayectoria.

Parámetros temporales:

Estos parámetros se centran en el tiempo transcurrido durante la trayectoria y la frecuencia de los mensajes registrados. Estos parámetros ofrecen información crucial sobre la frecuencia y la duración de los mensajes registrados, así como la distribución temporal de la velocidad en la trayectoria.

- *traj_time*: tiempo total transcurrido de inicio a fin de la trayectoria.
- *traj_msg_rate*: tasa de mensajes en la trayectoria obtenida dividiendo el número total de mensajes en la trayectoria por el tiempo total de la trayectoria.
- *traj_time_at_speed_0*: tiempo transcurrido en el que el buque marcaba velocidad 0 en plena trayectoria. Este parámetro nos puede dar una idea sobre el número de veces que el buque ha parado motores sin que haya realizado una parada como tal durante una hora completa. Como veremos en los resultados, es uno de los parámetros menos representativos, lo que quiere decir que las trayectorias se han discretizado correctamente.
- *traj_time_over_speed_0*: tiempo transcurrido en el que el buque marcaba una velocidad mayor que cero en navegación. Este parámetro debería ser muy parecido a *traj_time* aunque dará ligeramente mejores resultados debido a que se eliminan los tiempos a velocidad 0. La mejora será tan insignificante en comparación con el tiempo de cálculo que requiere, que se desecharán estos dos últimos parámetros en futuras iteraciones.
- *traj_mean_time_between_msg*: tiempo promedio entre mensajes durante la trayectoria.
- *traj_max_time_between_msg*: máximo tiempo entre mensajes durante la trayectoria.
- *traj_min_time_between_msg*: mínimo tiempo entre mensajes durante la trayectoria.

Estos últimos tres parámetros pueden ofrecer una visión sobre cómo de fiable es la trayectoria en cuanto a calidad de trazado, dejando para futuras iteraciones el filtrado de mensajes para que todas las trayectorias cuenten con un tiempo mínimo entre mensajes (para reducir el tamaño del DF que contiene la trayectoria) y máximo entre mensajes (para desechar los buques que no transmiten lo suficientemente a menudo como para generar una trayectoria fiable).

Parámetros de velocidad:

Estos parámetros proporcionan información detallada sobre la velocidad a lo largo de la trayectoria, incluyendo la velocidad media, máxima y mínima, así como las variaciones en la velocidad.

- *traj_mean_speed*: La velocidad media durante la trayectoria.
- *traj_max_speed*: La velocidad máxima alcanzada en la trayectoria.
- *traj_min_speed*: La velocidad mínima alcanzada en la trayectoria.
- *traj_mean_speed_var*: La variación media de velocidad entre dos mensajes consecutivos. Este parámetro calcula aceleración y deceleración sin diferenciar entre ellas, por lo que si la trayectoria es completa (de parada a parada) el valor debería siempre ser cercano a cero.

- *traj_max_speed_var*: La variación máxima de velocidad entre mensajes (debería coincidir con el valor de aceleración máxima).
- *traj_min_speed_var*: La variación mínima de velocidad entre mensajes (debería coincidir con el valor de deceleración mínima, es decir, deceleración más negativa).

Mensajes transmitidos por rangos de velocidad:

En una primera instancia, para estos parámetros se pretendía calcular la distancia recorrida cuando el buque se encontraba en ciertos rangos de velocidades, ya que, analizando las representaciones de las trayectorias de las Figuras 3-36, 3-37 y 3-38 (entre otros no recogidos en la memoria), se observaba un claro patrón en el rango de velocidades en los que transita cada buque. Sin embargo, el cálculo de la distancia diferenciando rangos de velocidades suponía un esfuerzo computacional muy elevado, por lo que se recurrió a contar los mensajes que se encontraban en dichos rangos. Por ello, en esta primera función permaneció el nombre *traj_dist_* aunque en el posterior cálculo de parámetros cinemáticos, se renombrará como *traj_msg_*. En futuras iteraciones se propondrá la normalización de ciertos valores, ya que es más interesante el porcentaje de mensajes que se encuentran en un rango de velocidad que el número de ellos, ya que el segundo, depende de la extensión de la trayectoria y de la tasa de transmisión de mensajes.

- *traj_dist_0-0.5*: Número de mensajes que contienen velocidades entre 0 y 0.5 nudos.
- *traj_dist_0.5-1*: Número de mensajes que contienen velocidades entre 0.5 y 1 nudos.
- *traj_dist_1-1.5*: Número de mensajes que contienen velocidades entre 1 y 1.5 nudos.
- *traj_dist_1.5-2*: Número de mensajes que contienen velocidades entre 1.5 y 2 nudos.
- *traj_dist_2-5*: Número de mensajes que contienen velocidades entre 2 y 5 nudos.
- *traj_dist_5-10*: Número de mensajes que contienen velocidades entre 5 y 10 nudos.
- *traj_dist_10-20*: Número de mensajes que contienen velocidades entre 10 y 20 nudos.
- *traj_dist_>20*: Número de mensajes que contienen velocidades mayores a 20 nudos.

Como veremos, estos parámetros son de gran interés para el algoritmo, especialmente en velocidades bajas, por lo que en futuras iteraciones se hará un estudio de diferentes rangos de velocidad, para determinar cuáles son los más representativos de cada tipo de buque.

Parámetros de aceleraciones y deceleraciones:

Estos parámetros describen las aceleraciones y deceleraciones experimentadas durante la trayectoria, calculadas con el campo de velocidad y el tiempo entre mensajes. La información sobre cómo la velocidad cambia a lo largo del tiempo permite la evaluación de la dinámica de movimiento y las fuerzas aplicadas en la trayectoria.

- *traj_max_acc*: máxima aceleración alcanzada en la trayectoria.
- *traj_min_acc*: mínima aceleración alcanzada en la trayectoria.
- *traj_mean_acc*: aceleración media durante la trayectoria.
- *traj_max_dec*: máxima deceleración alcanzada en la trayectoria.
- *traj_min_dec*: mínima deceleración alcanzada en la trayectoria.
- *traj_mean_dec*: deceleración media durante la trayectoria.

En muchos parámetros, los valores medios suelen eliminar la característica diferenciable de cada buque, siendo los valores máximos preferidos por el algoritmo. Sin embargo, en el caso de las aceleraciones y deceleraciones, el algoritmo da mayor peso a los valores medios que a los máximos y los mínimos. Esto puede deberse a que existan mensajes con irregularidades pronunciadas en todos los tipos de buque, pero, al calcular medias, se suavizan los valores, generando parámetros característicos de cada tipo de buque.

Parámetros de rumbo y dirección:

Estos parámetros describen los errores entre el rumbo y la dirección, así como las tasas de cambio en estas variables, proporcionando información sobre la estabilidad en el rumbo y la precisión de la navegación. Toman como valores los campos de *course* (el rumbo que sigue sobre fondo) y *heading* (la dirección a la que está apuntando su proa), además de las diferencias de tiempo entre mensajes consecutivos para calcular las tasas de cambio de rumbo y dirección.

- *traj_max_c_h_error*: máximo error entre rumbo y dirección.
- *traj_min_c_h_error*: mínimo error entre rumbo y dirección.
- *traj_mean_c_h_error*: error medio entre rumbo y dirección.
- *traj_max_course_rate*: tasa máxima de cambio de rumbo entre dos mensajes consecutivos.
- *traj_min_course_rate*: tasa mínima de cambio de rumbo entre dos mensajes consecutivos.
- *traj_mean_course_rate*: tasa media de cambio de rumbo entre dos mensajes consecutivos.
- *traj_max_head_rate*: tasa máxima de cambio de dirección entre dos mensajes consecutivos.
- *traj_min_head_rate*: tasa mínima de cambio de dirección entre dos mensajes consecutivos.
- *traj_mean_head_rate*: tasa media de cambio de dirección entre dos mensajes consecutivos.

Estos parámetros se consideraron fundamentales para evaluar la precisión y estabilidad del rumbo y la dirección a lo largo de la trayectoria ya que un buque de grandes proporciones debería tener un rumbo más estable a lo largo de una navegación ya que se ve menos afectado por deriva y abatimiento que un buque pequeño (al menos en instantes de tiempo cercanos). A pesar de las suposiciones, el algoritmo no los consideró como parámetros relevantes y, tras analizar el motivo de estos resultados, se descubrió que el campo *heading* rara vez viene cubierto correctamente por lo que las incoherencias entre rumbos y direcciones son independientes del tipo de buque. Por otro lado, las tasas de cambio de rumbo tampoco presentan gran relevancia para el algoritmo. Este parámetro se esperaba también como característico ya que las tasas de cambio de rumbo o dirección dependen en gran medida de la maniobrabilidad de los buques, sin embargo, dada la baja cadencia de transmisión de los datos, no es posible extraer valores precisos de cambio de rumbo. Por estas razones, se desecharán todos los parámetros de rumbo y dirección además de eliminarse los campos de *course* y *heading* en preprocesados de futuras iteraciones para reducir el tamaño del DF. Otros parámetros relativos al rumbo también se consideraron, como el rumbo principal de la ruta (calculando la moda de los rumbos del DF) y la presencia de rumbo principal (calculando el cociente entre la moda y el total de mensajes de la trayectoria). Sin embargo, dada la baja fiabilidad de estos campos y la baja cadencia de los datos, se optó por descartar los parámetros relativos a estos campos.

3.3.5 Extracción de parámetros de paradas

La extracción de los parámetros de paradas vendrá dada por la función *calcular_parametros_paradas* que toma como único objeto la colección de trayectorias *month_stops*. Como en el apartado anterior, la mayoría de los cambios se han realizado en base a los resultados del experimento anterior. En el caso de la extracción de parámetros de paradas, no fue hasta la cuarta iteración cuando se comenzaron a incluir para alimentar al algoritmo. A continuación, se explicarán los parámetros de paradas.

- *stop_N*: representa el número de la parada. El objetivo de este parámetro, al igual que el de *traj_N*, es contar el número de paradas (trayectorias en caso de *traj_N*) realizadas en ese mes.
- *stop_dist*: representa la distancia que se ha recorrido en la parada. Este parámetro da una idea sobre si la parada es estable (en puerto) o inestable (fondeado). Se prevé que los buques pesqueros sean más inestables. Sin embargo, como la discretización está basada en un radio de distancia, veremos cómo la distancia recorrida durante la parada no es un parámetro determinante.

- *stop_time*: representa el tiempo total en el que el buque ha estado parado. Se prevé que los tiempos de parada sean muy representativos del tipo de buque. La diferenciación entre carguero y petrolero puede marcarla este parámetro ya que la estiba de contenedores es un proceso más largo que la carga de un petrolero. Por ello, de este parámetro se extraerán tres parámetros que se guardarán en el DF de parámetros mensuales. El tiempo máximo de parada en ese mes, el tiempo medio de todas las paradas realizadas en ese mes, y el tiempo mínimo (ver Figura 3-46 para entender este concepto). Todos ellos se explicarán en el subapartado de extracción de parámetros del mes.

Estos parámetros acabaron posicionados en la primera mitad de la clasificación, siendo el tiempo de parada medio el más relevante. Se espera, que con mejores *dataset*, puedan realizarse discretizaciones de rutas más precisas, elevando la relevancia de estos parámetros, especialmente, el número de paradas y la distancia recorrida durante la parada.

3.3.6 Extracción de parámetros del mes

La extracción de parámetros del mes se divide en dos procesos y vendrá dada por la función *calcular_parametros_cinemáticos,* que toma como objeto la trayectoria *month_trajectories, df_traj_parameters* y *df_stop_parameters* (a partir de la tercera iteración). Por un lado, se obtienen los parámetros de la trayectoria mensual, todos ellos precedidos por *month_* a partir de la trayectoria mensual. Por otro lado, calcula valores medios, máximos y mínimos de los DF obtenidos anteriormente para almacenar en una sola fila de un solo MMSI información de varias trayectorias y paradas. Siguiendo la misma dinámica que en los anteriores, se explicarán los parámetros extraídos en la primera iteración y, posteriormente, los nuevos parámetros con el paso de las iteraciones.

Parámetros generales:

- *N_Trajectories*: Número total de trayectorias realizadas en todo el mes. Se calcula extrayendo el valor máximo de la columna *traj_N* del *df_traj_parameters.*

Parámetros geográficos:

- *traj_max_lat_mean, traj_min_lat_mean, traj_mean_lat_mean*: Media de las latitudes máximas, mínimas y medias de las trayectorias, tomadas de *df_traj_parameters.*
- *traj_max_lon_mean, traj_min_lon_mean, traj_mean_lon_mean*: Media de las longitudes máximas, mínimas y medias de las trayectorias.
- *month_max_lat, month_min_lat, month_mean_lat*: Latitud máxima, mínima y media del mes, tomadas de *month_trajectories.*
- *month_max_lon, month_min_lon, month_mean_lon*: Longitud máxima, mínima y media del mes.
- *traj_max_dif_lat_mean, traj_max_dif_lon_mean*: Media de las diferencias máximas de latitud y longitud entre puntos (no necesariamente consecutivos) de las trayectorias.
- *month_max_dif_lat, month_max_dif_lon*: Diferencia máxima de latitud y longitud de todo el mes.

Distancias:

- *traj_max_sep_max*: Distancia máxima entre puntos de la trayectoria, al depender *traj_max_sep* de *df_traj_parameters*, este parámetro se igualó a cero ya que no se pudo calcular porque suponía elevadas cargas computacionales.
- *traj_dist_max, traj_dist_min, traj_dist_mean*: Distancia máxima, mínima y media recorrida en una trayectoria.
- *month_dist*: Distancia total recorrida durante todo el mes.

Aspectos temporales:

- *traj_time_mean*: Tiempo medio entre todas las trayectorias.
- *traj_msg_rate_mean, traj_msg_rate_max, traj_msg_rate_min*: Tasa media, máxima y mínima de mensajes por unidad de tiempo en las trayectorias.
- *traj_mean_time_between_msg_mean, traj_max_time_between_msg_max, traj_min_time_between_msg_min*: Tiempo medio, máximo y mínimo entre mensajes en las trayectorias.

Velocidades:

- *traj_mean_speed_mean, month_mean_speed*: Media de la velocidad media de las trayectorias y velocidad media del mes. Estos valores, aparentemente iguales, diferirán ya que el primero no tendrá en cuenta la velocidad de las paradas, mientras que el segundo sí.
- *traj_max_speed_max, month_max_speed*: Velocidad máxima de las trayectorias y velocidad máxima del mes. En este caso, ambos parámetros deberían coincidir ya que las altas velocidades están presentes en los tramos de navegación.
- *traj_min_speed_min, month_min_speed*: Velocidad mínima de las trayectorias y velocidad mínima del mes. Aunque no tienen por qué coincidir, veremos cómo en la mayor parte de los casos sí lo hacen, ya que tanto en navegación como en paradas casi siempre hay un valor de velocidad 0.
- *traj_mean_speed_var_mean*: Media de la variación de velocidad media de las trayectorias, al igual el parámetro del que procede (*traj_mean_speed_var*) debería ser un valor muy cercano a 0.

Aceleraciones y deceleraciones:

- *traj_max_acc_max, traj_min_acc_min, traj_mean_acc_mean*: Aceleración máxima, media y mínima de las trayectorias.
- *traj_max_dec_max, traj_min_dec_min, traj_mean_dec_mean*: Deceleración máxima, media y mínima de las trayectorias.

Parámetros de rumbo y dirección:

- *traj_max_c_h_error_max, traj_min_c_h_error_min, traj_mean_c_h_error_mean*: Error máximo, mínimo y medio de rumbo/dirección de las trayectorias.
- *traj_max_course_rate_max, traj_min_course_rate_min, traj_mean_course_rate_mean*: Tasa máxima, mínima y media de cambio de rumbo de las trayectorias.
- *traj_max_head_rate_max, traj_min_head_rate_min, traj_mean_head_rate_mean*: Tasa máxima, mínima y media de cambio de dirección de las trayectorias.

Como se observa, en esta primera iteración no se aplican parámetros de paradas ya que no es hasta la cuarta iteración cuando se incluyen dichos parámetros.

Durante la segunda iteración se realizan dos cambios. El primero es la eliminación de los parámetros menos relevantes para reducir tiempos de procesamiento y tamaño del DF de parámetros. El segundo, la relativización de los mensajes por rango de velocidad. Tal y como se dijo en el subapartado de cálculo de parámetros de trayectorias, el número de mensajes que contienen un rango de velocidades puede depender de muchos factores como la tasa de transmisión de los mensajes. Para paliar estas influencias externas, se dividen estos valores entre el número total de mensajes, quedando así un porcentaje de la ruta en el que el buque ha transcurrido a un cierto rango de velocidad.

En la Figura 3-50 se muestra una tabla de los cambios realizados de la segunda iteración. En la primera columna se encuentra el parámetro de la trayectoria en concreto y en la segunda, el valor medio, máximo y/o mínimo de todas las trayectorias realizadas durante ese mes. Como podemos observar en dicha figura, cada parámetro tiene asociado una letra. La letra A significa que es un parámetro aceptado por el algoritmo, genera buenas predicciones y se mantendrá para posteriores iteraciones. La letra U significa que es un parámetro útil, esto quiere decir que es necesario para calcular otros parámetros de clase A y que también puede estar bien posicionado o bien que ha sido

utilizado para analizar el conjunto de datos (tiempo entre mensajes o tasa de transmisión). La letra C quiere decir que es un parámetro característico del estudio de trayectorias. Esta clase se presume que debería mejorar con un conjunto de datos más preciso y con unas trayectorias mejor definidas. La letra N indica que es un parámetro no utilizado para la siguiente iteración debido a que:

- requiere demasiado procesamiento a pesar de considerarse característico (*traj_max_sep*);
- a pesar de ser característico, el algoritmo lo considera poco relevantes por la baja precisión de las trayectorias (*traj_max_dif_lat/lon*)
- existen otros parámetros que engloban al mismo y están posicionados con igual nivel de relevancia (*traj_time_over_speed_0* debería ser muy parecido a *traj_time*, requiriendo el segundo menos tiempo de procesamiento);
- pertenece a un conjunto que se ha descartado por baja fiabilidad del campo (parámetros de rumbo y dirección).

Por último, la letra P significa prescindible, ya que no es especialmente relevante para el algoritmo, pero tampoco requiere mucho procesamiento por lo que puede dejarse para la siguiente iteración. Los colores, por otro lado, muestran si los parámetros se han descartado (salmón) o se han conservado (verde).

traj_parameters	cinematic_parameters	CLASE
Traj_N	N_Trajectories	C
traj_max_lat	traj_max_lat_mean	N
traj_min_lat	traj_min_lat_mean	N
traj_mean_lat	traj_mean_lat_mean	N
traj_max_lon	traj_max_lon_mean	N
traj_min_lon	traj_min_lon_mean	N
traj_mean_lon	traj_mean_lon_mean	N
	month_max_lat	U
	month_min_lat	U
	month_mean_lat	P
	month_max_lon	U
	month_min_lon	U
	month_mean_lon	P
traj_max_dif_lat	traj_max_dif_lat_mean	C
traj_max_dif_lon	traj_max_dif_lon_mean	C
	month_max_dif_lat	A
	month_max_dif_lon	A
traj_max_sep	traj_max_sep_mean	N
traj_dist	traj_dist_max	A
	traj_dist_min	N
	traj_dist_mean	A
	month_dist	A
traj_time	traj_time_mean	A
traj_msg_rate	traj_msg_rate_mean	U
	traj_msg_rate_max	U
	traj_msg_rate_min	U
traj_dist_0-0.5	traj_msg_0_0dec5_mean	A
traj_dist_0.5-1	traj_msg_0dec5_1_mean	A
traj_dist_1-1.5	traj_msg_1_1dec5_mean	A
traj_dist_1.5-2	traj_msg_1dec5_2_mean	A
traj_dist_2-5	traj_msg_2_5_mean	A
traj_dist_5-10	traj_msg_5_10_mean	A
traj_dist_10-20	traj_msg_10_20_mean	C
traj_dist_>20	traj_msg_over20_mean	C

traj_time_at_speed_0	traj_time_at_speed_0_mean	N
	traj_time_at_speed_0_max	N
	traj_time_at_speed_0_min	N
traj_time_over_speed_0	traj_time_over_speed_0_mean	N
	traj_time_over_speed_0_max	N
	traj_time_over_speed_0_min	N
traj_mean_time_between_msg	traj_mean_time_between_msg_mean	U
traj_max_time_between_msg	traj_max_time_between_msg_max	U
traj_min_time_between_msg	traj_min_time_between_msg_min	U
traj_mean_speed	traj_mean_speed_mean	C
	month_mean_speed	A
traj_max_speed	traj_max_speed_max	C
	month_max_speed	A
traj_min_speed	traj_min_speed_min	N
	month_min_speed	N
traj_mean_speed_var	traj_mean_speed_var_mean	N
traj_max_speed_var	traj_max_speed_var_max	N
traj_min_speed_var	traj_min_speed_var_min	N
traj_max_acc	traj_max_acc_max	A
traj_min_acc	traj_min_acc_min	N
traj_mean_acc	traj_mean_acc_mean	A
traj_max_dec	traj_max_dec_max	N
traj_min_dec	traj_min_dec_min	N
traj_mean_dec	traj_mean_dec_mean	A
traj_max_c_h_error	traj_max_c_h_error_max	N
traj_min_c_h_error	traj_min_c_h_error_min	N
traj_mean_c_h_error	traj_mean_c_h_error_mean	N
traj_max_course_rate	traj_max_course_rate_max	N
traj_min_course_rate	traj_min_course_rate_min	N
traj_mean_course_rate	traj_mean_course_rate_mean	N
traj_max_head_rate	traj_max_head_rate_max	N
traj_min_head_rate	traj_min_head_rate_min	N
traj_mean_head_rate	traj_mean_head_rate_mean	N

Figura 3-50: Clasificación de parámetros

En la tercera iteración se mantienen los mismos parámetros que en la segunda iteración, sin embargo, cambia el valor de los umbrales, tal y como se explicó en el apartado de discretización de trayectorias. En la cuarta iteración, se busca definir distintos rangos de velocidad para encontrar los que más caracterizan la ruta de un buque, encontrando los rangos que más caracterizan a cada buque y dejando en la quinta iteración aquellos que hayan aportado mejores resultados.

3.3.7 Bucle de extracción de parámetros

El bucle completo que extrae los parámetros permanece casi invariable a lo largo de todas las iteraciones, mostrándose el de cada una en el código *prep_dynamics*, en el Anexo IV. En grandes rasgos, el bucle itera sobre cada MMSI generando trayectorias, las discretiza y posteriormente extrae los parámetros de trayectorias, paradas y mensuales, combinándose finalmente todos ellos para añadirlo a una única fila de parámetros cinemáticos. Cabe destacar que se utiliza la palabra "cinemático" ya que los parámetros comprenden aspectos de comportamiento del buque, no solo limitándose a las características dinámicas de las que se parte con el *dataset* inicial. En la Figura 3-51, se muestra el diagrama de flujo del bucle de extracción de parámetros.

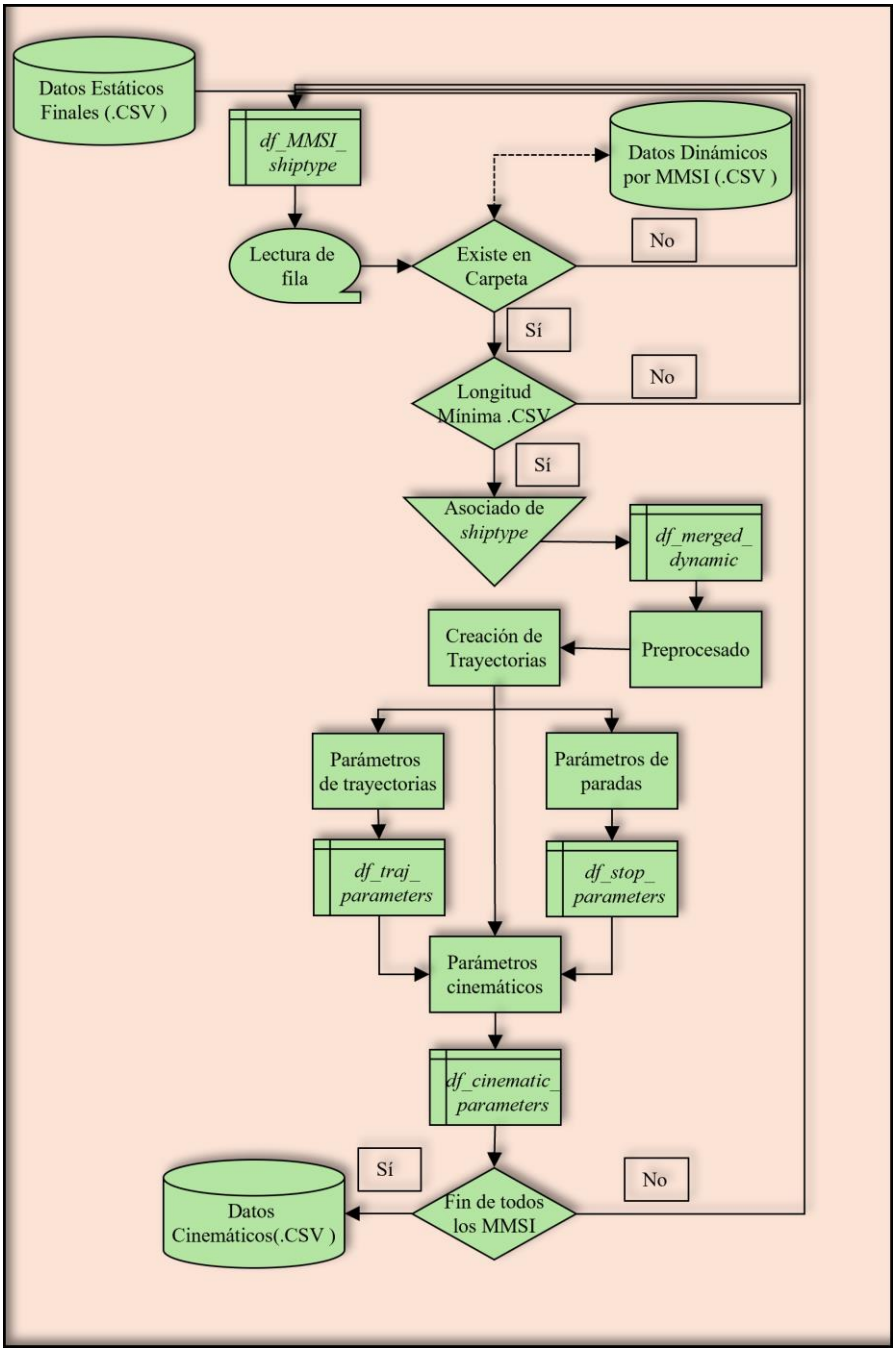


Figura 3-51: Diagrama de flujo del bucle de extracción de parámetros cinemáticos

El estudio de tiempos de ejecución se realizó a lo largo de ejecuciones puntuales y durante todo el bucle de algunas iteraciones. Gracias a este estudio, se descartaron numerosos parámetros y se redujeron tiempos de ejecución, sin embargo, estos tiempos siguen siendo muy altos, realizando una

extracción completa de parámetros en un tiempo de entre 72 y 96 horas, siendo el mayor para las primeras iteraciones. Por ello, más allá de la eliminación de parámetros que requieren altos tiempos de procesamiento, no procede el análisis en profundidad de los tiempos de ejecución, ya que estos vienen condicionados principalmente por la generación de la trayectoria, y no por la extracción de parámetros. No obstante, se ha dejado en el código del Anexo IV, todos los cálculos de tiempos de procesamiento, de modo que se pueda profundizar en la optimización del mismo en futuros trabajos.

Los elevados tiempos de ejecución, se deben al proceso de creación de trayectorias, que comprende el uso de la biblioteca *moovingpandas* para generar la trayectoria mensual, separarla en trayectorias individuales y extraer las trayectorias de paradas. El tiempo de creación de trayectoria oscila entre los 4 y 6 segundos dependiendo de la complejidad de la misma, que, al multiplicarlo por los más de 40 mil buques, se obtiene un tiempo medio de 55 horas. Estos tiempos se calcularon en diversos equipos personales con capacidades computacionales similares, existiendo variaciones despreciables según el equipo utilizado. El utilizado mayormente y con el que se ha generado el código y los tiempos mostrados en las figuras es un portátil que cuenta con un procesador *AMD Ryzen 7 5825U* de 2.00 GHz con integración de *Radeon Graphics* y una RAM de 16 GB. Estos elevados tiempos se pueden paliar haciendo uso de una base de datos que almacene los objetos de trayectorias ya generados. De esta forma, solo haría falta generarlas en la primera iteración, dejando la extracción de parámetros como única carga computacional, la cual, ha sido pulida a lo largo de este trabajo.

4 RESULTADOS DEL TFG

En el presente capítulo se presentarán los resultados derivados de los experimentos llevados a cabo, destacando solo aquellos cuyos resultados permitieron extraer conclusiones significativas. Se procederá inicialmente a explicar las métricas empleadas para evaluar los resultados del algoritmo, seguido por un análisis de los experimentos organizados por iteraciones, con una iteración final que busca un balance entre número de parámetros y calidad de los mismos. Por último, se realizará una evaluación global de los experimentos teniendo en cuenta tiempos de ejecución y mejoras en predicción con respecto a anteriores proyectos.

Cabe destacar que el preprocesado y procesado de datos ha sufrido numerosas modificaciones a lo largo del proyecto y ambos procesos son susceptibles al cambio o modificación para mejorar el proceso. Para favorecer posibles líneas en esta dirección se ha procurado generar un código claro y estructurado, detallado el Anexo IV y dividido en preparación de estáticos, preparación de dinámicos y resultados. La experimentación realizada se ha visto gravemente afectada por los tiempos de procesamiento de los datos, lo que ha restringido en gran medida el número y la variedad de experimentos.

4.1 Métricas empleadas

Como se ha mencionado anteriormente, la librería *scikit-learn* nos ofrece la función *classification_report* que proporciona diversas métricas comúnmente empleadas para la evaluación de prestaciones de algoritmos de clasificación en la literatura, que serán utilizadas para evaluar los resultados de cada experimento. A continuación, se describen y se contextualizan las métricas presentes en los informes derivados de cada experimento:

- **Exactitud (*Accuracy*):** Esta métrica cuantifica el porcentaje de casos que el modelo ha predicho correctamente. Se calcula dividiendo el número de predicciones correctas entre el total de predicciones realizadas. Aunque es ampliamente utilizada debido a su simplicidad, su interpretación debe ser cautelosa, especialmente en conjuntos de datos desbalanceados. Por ejemplo, en un escenario de mil buques donde existe una predominancia de buques de tipo mercante (pongamos 800), si el modelo evalúa todos los buques como mercantes la exactitud sería del 80%, aunque el modelo no logre distinguir adecuadamente los pesqueros de los remolcadores, lo cual indica una limitación en su capacidad de generalización.
- **Precisión (*Precision*):** Esta métrica mide el porcentaje de predicciones correctas realizadas para cada clase en particular. Se calcula como la proporción entre las predicciones correctas de una clase y el total de predicciones realizadas para esa clase. Por ejemplo, si el modelo clasifica cien elementos como pesqueros, pero solo 50 de ellos lo son realmente, la

precisión para pesqueros sería del 50%. Por el contrario, si clasifica 40 elementos como pesqueros y todos ellos lo son realmente, aunque falten 10 por clasificar como pesquero, la precisión sería del 100%. Para compensar esto último, tenemos el parámetro que se explica a continuación.

- **Exhaustividad (*Recall*):** Esta métrica evalúa el porcentaje de elementos de una clase específica que el modelo ha logrado clasificar correctamente. Se calcula como la proporción entre los datos correctamente clasificados de una clase y el total de datos pertenecientes a esa clase. Tomando el ejemplo anterior, si existen 50 pesqueros en total en los datos, pero el modelo solo clasifica correctamente 40 de ellos, entonces la exhaustividad para esa clase sería del 80%.
- **Valor-F1 (*F1-Score*):** Esta medida combina la exhaustividad y la precisión en un solo valor. Es la medida que ofrece una visión más completa de la calidad de la predicción de clases al considerar ambas métricas, ya que enfocarse únicamente en una puede conducir a interpretaciones erróneas. El Valor-F1 se calcula como la media armónica entre la exhaustividad y la precisión.

Además, los informes proporcionan valores de medias aritméticas de los parámetros explicados anteriormente para todas las clases disponibles en el algoritmo de clasificación, así como una media ponderada, que tiene en cuenta la cantidad de datos de cada clase. Sin embargo, al igual que en los anteriores TFG y dado que en este estudio se trabaja con un conjunto de datos desbalanceado, se prestará mayor atención a la precisión, la exhaustividad y, especialmente, el F1-Score, dejando la exactitud y la media ponderada de las métricas en un segundo plano.

4.2 Diseño y estructuración de los experimentos

Los experimentos realizados se dividirán en dos grupos principales. En el primer grupo, compuesto por 2 experimentos, se comprueba que la optimización del código en el procesado y preprocesado de los datos estáticos no ha perjudicado la predicción del algoritmo. En ambos experimentos se analiza el DF resultante de la preparación de estáticos (*df_final_static*). Cada uno de los experimentos será el resultado de la preparación de datos, por un lado, del mes de enero, y por otro, del mes de febrero de 2023.

El segundo grupo de experimentos conforma el estudio de resultados del análisis de trayectorias. Estará compuesto por un total de 5 iteraciones en las que se comprobarán los resultados, por un lado, de los parámetros de trayectorias (*df_cinematic_parameters*) y, por otro lado, de la combinación de los parámetros estáticos (ya analizados individualmente en el primer grupo de experimentos) y de trayectorias (*df_global_parameters*). De esta manera, quedan en total un total de 12 experimentos, siendo dos de ellos la validación con los datos del mes de febrero.

Como se puede observar en la Figura 4-1, la iteración 0 conforma el primer grupo de experimentos que, como se ha explicado anteriormente, su objetivo es asegurar que los MMSI extraídos para el tratamiento de los datos dinámicos sean fiables y las predicciones coincidan con anteriores realizadas; la iteración 1, en la que se proponen un gran número de parámetros de los cuales muchos deben ser descartados, cuenta con un total de 4 experimentos donde los dos últimos validan las conclusiones que se extraen de los dos primeros; y las posteriores iteraciones cuentan con dos experimentos cada una utilizando únicamente los datos del mes de enero. En la misma figura, además, se resume la evolución de los experimentos con la columna que muestra la exactitud, marcándose en negrita los resultados más relevantes.

Para la consulta de parámetros específicos de los resultados, quedará disponible el código *Results.ipynb* en el Anexo IV.

Iteración	Experimento	Datos	Nomenclatura	Exactitud	Mes
0	1	Estáticos	*final_static_data_jan.CSV*	**0,86**	Enero
	2	Estáticos	*final_static_data_feb.CSV (0-2)*	0,85	Febrero
1	3	Trayectorias	*Cinematic_parameters_jan.CSV*	0,78	Enero
	4	Globales	Combinación de experimentos 1 y 3	**0,91**	
	5	Trayectorias	*Cinematic_parameters_feb.CSV*	0,79	Febrero
	6	Globales	Combinación de experimentos 2 y 5	**0,91**	
2	7	Trayectorias	*Cinematic_parameters_2.CSV*	0,78	Enero
	8	Globales	Combinación de experimentos 1 y 7	0,90	
3	9	Trayectorias	*Cinematic_parameters_3.CSV*	0,77	Enero
	10	Globales	Combinación de experimentos 1 y 9	0,90	
4	11	Trayectorias	*Cinematic_parameters_4.CSV*	0,77	Enero
	12	Globales	Combinación de experimentos 1 y 11	0,90	
5	13	Trayectorias	*Cinematic_parameters_5.CSV*	0,79	Enero
	14	Globales	Combinación de experimentos 1 y 13	**0,91**	

Figura 4-1: Estructura de los experimentos

4.3 Análisis de resultados

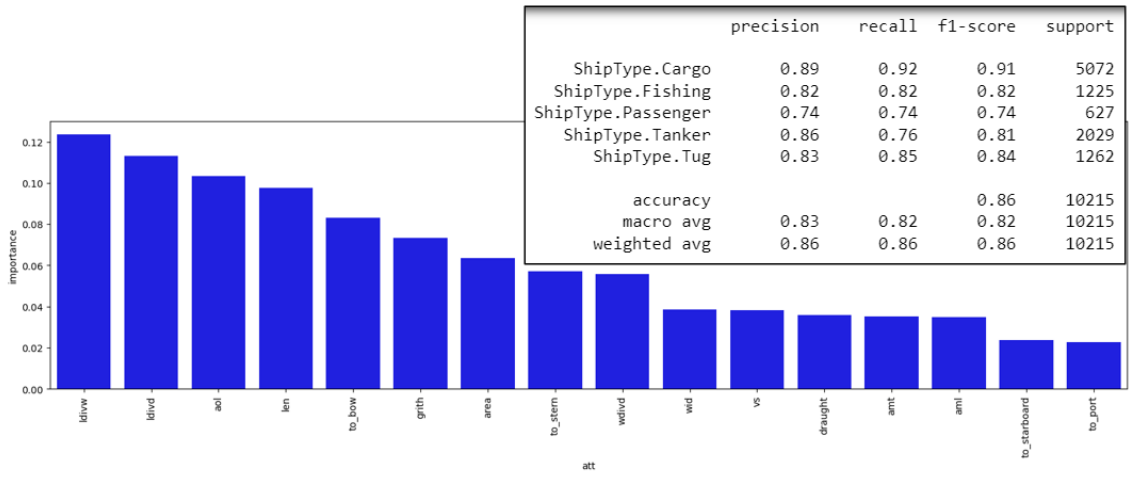
4.3.1 Resultados de la preparación de estáticos: Iteración 0

La preparación de estáticos tenía como propósito principal obtener un conjunto de buques que son capaces de transmitir información estática fiable, por ello, además de los filtrados que se realizaron en anteriores TFG, se llevaron a cabo otra serie de operaciones que permitían obtener un único MMSI con valores estáticos medios y sin presencia de *outliers*. Para comprobar que las operaciones adicionales y los cambios para mejorar la eficiencia y legibilidad del código no supusieran una disminución de la calidad de la predicción, se realizaron los experimentos 1 y 2, de enero y febrero respectivamente.

Las figuras que se observarán durante el resto de este capítulo mostrarán los resultados de los experimentos y mantendrán la misma estructura. En la parte superior derecha se mostrará una leyenda de las métricas explicadas anteriormente con sus correspondientes porcentajes, siendo la última columna (*support*) la cantidad de datos de la que ha dispuesto el algoritmo para evaluarse. En la parte inferior, se mostrará un gráfico de barras en el que indica qué atributos (eje x: *att*) han sido más útiles para el algoritmo (eje y: *importance*).

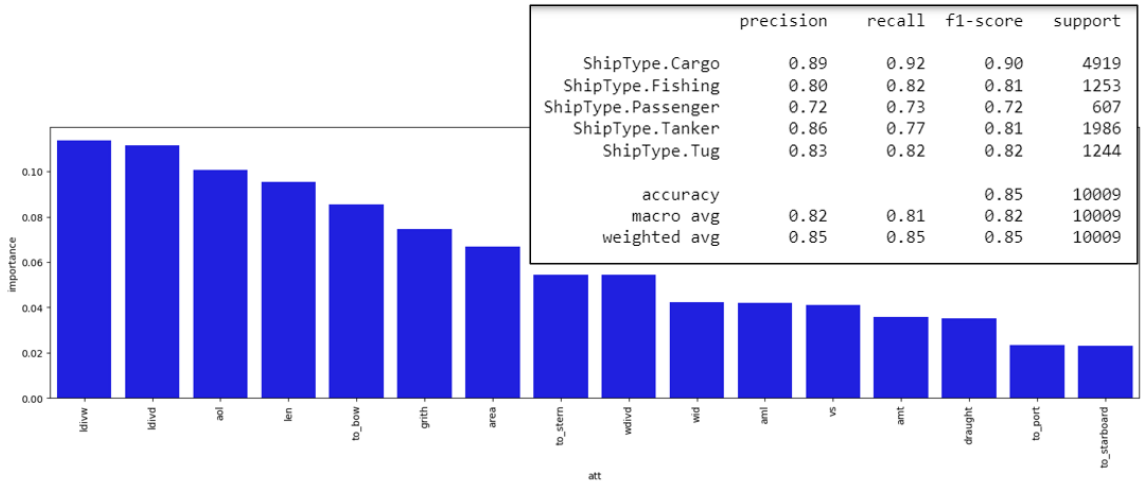
En las Figura 4-2 y 4-3 se muestran los resultados de los experimentos 1 y 2 respectivamente. Como se puede apreciar, los valores se mantienen en la misma línea que las predicciones de anteriores proyectos, presentando una disminución en todas las métricas de entre un 2 y un 5%. Esto se debe muy posiblemente a la densa reducción que ha sufrido el conjunto de datos, teniendo únicamente, como se explicó en el apartado de preparación de estáticos del anterior capítulo, una muestra reducida de unas 40 mil entradas a partir de un conjunto de datos de 9 millones de entradas. Además, a diferencia de anteriores trabajos, se ha optado por no usar técnicas de *oversampling* ni *undersampling* las cuales suelen mejorar las predicciones, ya que así permite la realización de comparaciones realistas, con una misma estructura en las predicciones. En cualquier caso, las predicciones de los experimentos se mantienen coherentes entre ambos, siendo ligeramente mayor la de enero, ya que existen más datos para ese mes. En cuanto a la importancia de los atributos, éstos presentan el mismo orden de

importancia que en los anteriores trabajos, manteniéndose, por tanto, la coherencia con los resultados de los trabajos previos y la continuidad de la línea de investigación.



	precision	recall	f1-score	support
ShipType.Cargo	0.89	0.92	0.91	5072
ShipType.Fishing	0.82	0.82	0.82	1225
ShipType.Passenger	0.74	0.74	0.74	627
ShipType.Tanker	0.86	0.76	0.81	2029
ShipType.Tug	0.83	0.85	0.84	1262
accuracy			0.86	10215
macro avg	0.83	0.82	0.82	10215
weighted avg	0.86	0.86	0.86	10215

Figura 4-2: Resultados del experimento 1



	precision	recall	f1-score	support
ShipType.Cargo	0.89	0.92	0.90	4919
ShipType.Fishing	0.80	0.82	0.81	1253
ShipType.Passenger	0.72	0.73	0.72	607
ShipType.Tanker	0.86	0.77	0.81	1986
ShipType.Tug	0.83	0.82	0.82	1244
accuracy			0.85	10009
macro avg	0.82	0.81	0.82	10009
weighted avg	0.85	0.85	0.85	10009

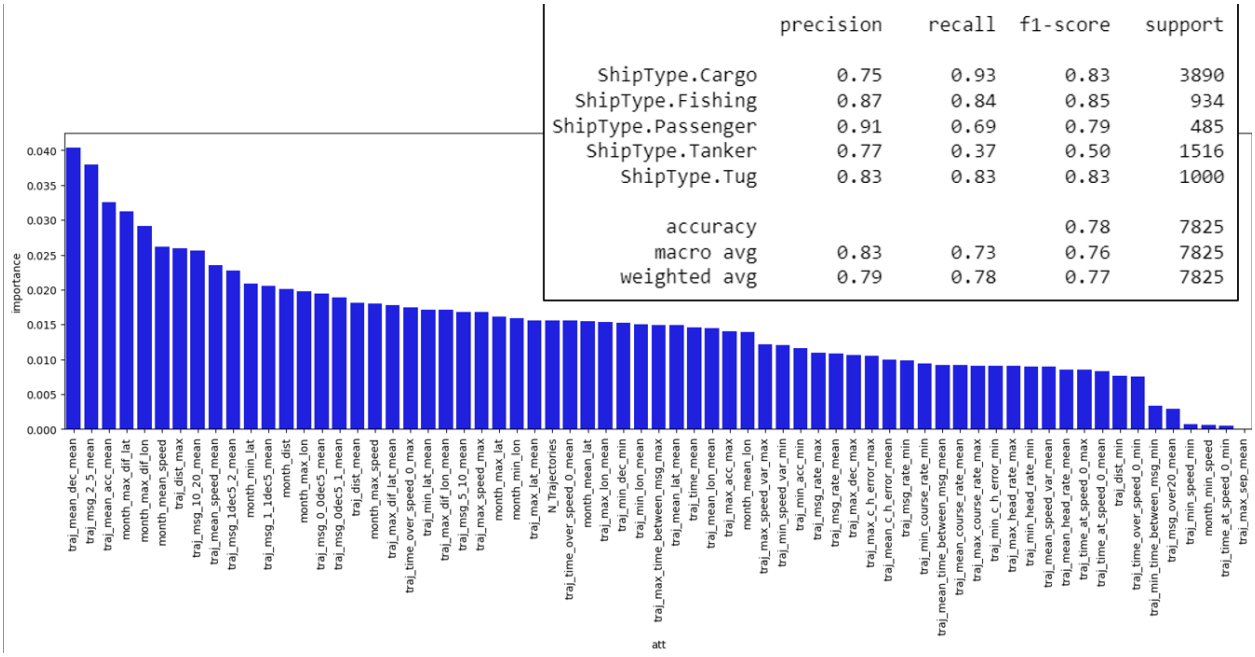
Figura 4-3: Resultados del experimento 2

4.3.2 Resultados de la preparación de trayectorias: Iteración 1

Una vez adaptados los datos estáticos a esta metodología de estudio de trayectorias, se procede al estudio inicial de parámetros cinemáticos. En esta primera iteración, se realizan dos experimentos para el mes de enero (uno con los parámetros cinemáticos y otro con la combinación de cinemáticos y estáticos) y se validan los resultados repitiendo los mismos dos experimentos para el mes de febrero. De nuevo, no se utilizan técnicas de *oversampling* ni *undersampling* para hacer comparaciones estrictas con las iteraciones anteriores.

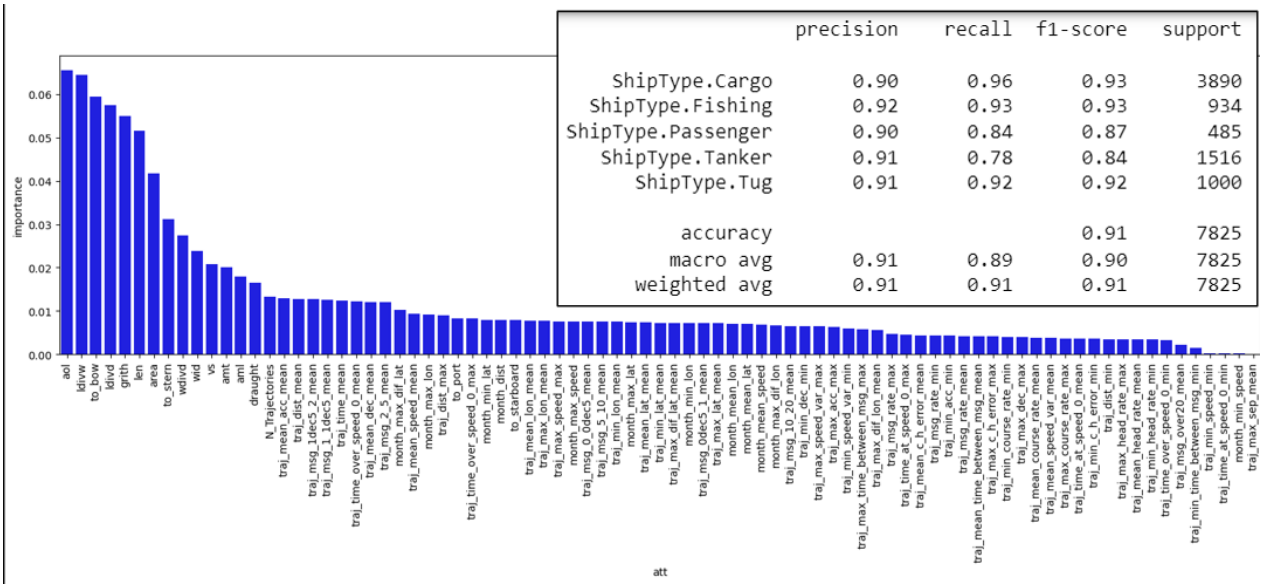
Como se puede observar en las Figuras 4-4 y 4-6 la exactitud del análisis inicial de trayectorias de enero y febrero, respectivamente, es del 78 y 79%. Esta predicción resulta ya desde la primera iteración de lo más prometedora, ya que de los parámetros dinámicos que se emplearon [9], se obtenían predicciones de alrededor del 70% sin técnicas de *oversampling* y los parámetros derivados del estudio de celdas *H3* entre un 55% y un 65% de exactitud. En cuanto a la combinación de ambas (Figuras 4-5 y 4-7), se observa como aumentan a un 91% de exactitud para ambos meses, siendo los parámetros estáticos más influyentes en las predicciones.

Por lo que respecta a las predicciones por clases, vemos cómo en los experimentos con datos de trayectorias (Figuras 4-4 y 4-6), el algoritmo presenta dificultades para predecir los petroleros (*Shiptype.Tanker*) ya que, al igual que en los datos estáticos, muestran grandes similitudes con los cargueros tanto en forma como en comportamiento cinemático. Sin embargo, este problema se ve aliviado parcialmente por algunos parámetros estáticos, permitiendo una mejor diferenciación entre ambas clases, lo que se ve reflejado en los experimentos 4 y 6 (Figuras 4-5 y 4-7), en la columna de *f1-score* para el petrolero.



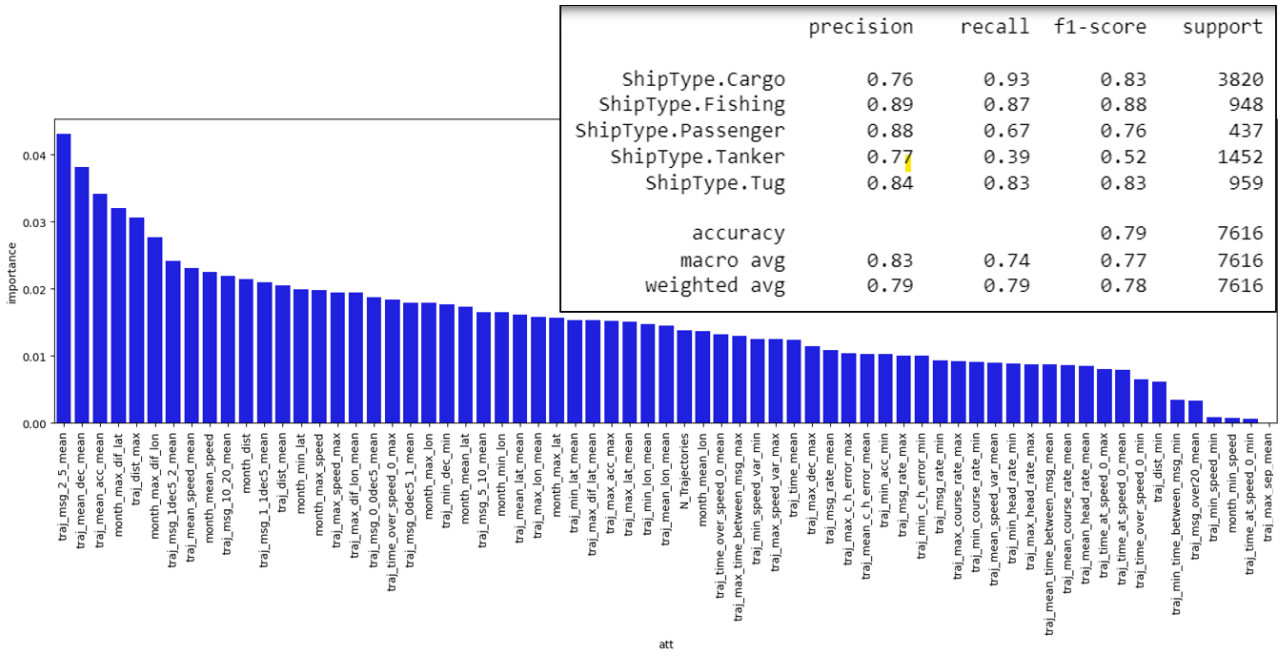
	precision	recall	f1-score	support
ShipType.Cargo	0.75	0.93	0.83	3890
ShipType.Fishing	0.87	0.84	0.85	934
ShipType.Passenger	0.91	0.69	0.79	485
ShipType.Tanker	0.77	0.37	0.50	1516
ShipType.Tug	0.83	0.83	0.83	1000
accuracy			0.78	7825
macro avg	0.83	0.73	0.76	7825
weighted avg	0.79	0.78	0.77	7825

Figura 4-4: Resultados del experimento 3



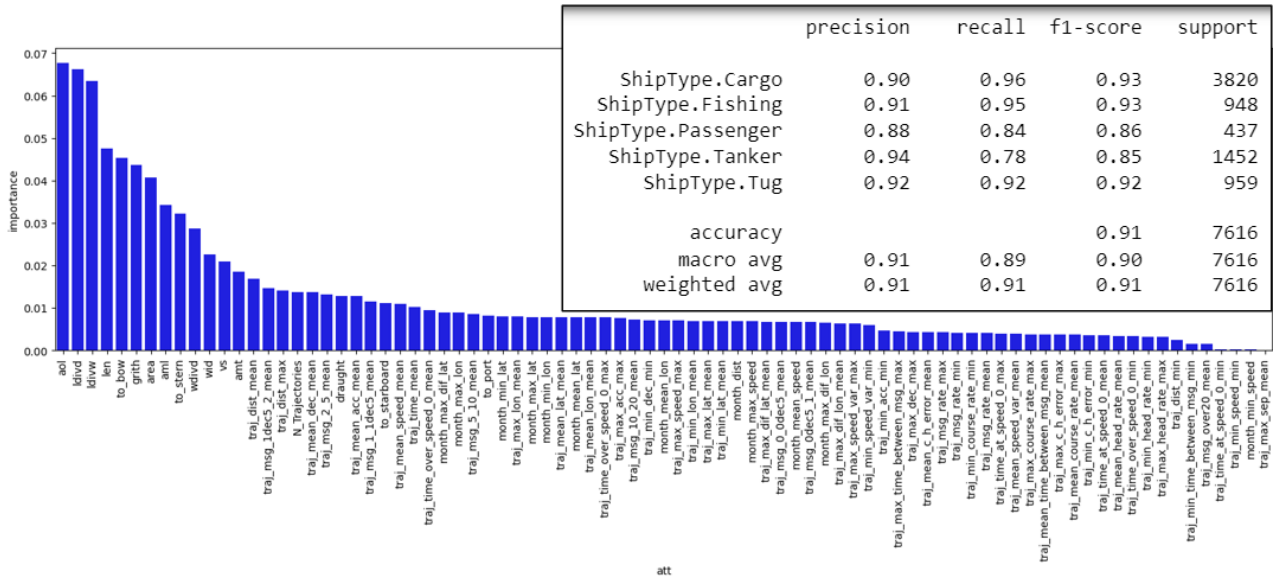
	precision	recall	f1-score	support
ShipType.Cargo	0.90	0.96	0.93	3890
ShipType.Fishing	0.92	0.93	0.93	934
ShipType.Passenger	0.90	0.84	0.87	485
ShipType.Tanker	0.91	0.78	0.84	1516
ShipType.Tug	0.91	0.92	0.92	1000
accuracy			0.91	7825
macro avg	0.91	0.89	0.90	7825
weighted avg	0.91	0.91	0.91	7825

Figura 4-5: Resultados del experimento 4



	precision	recall	f1-score	support
ShipType.Cargo	0.76	0.93	0.83	3820
ShipType.Fishing	0.89	0.87	0.88	948
ShipType.Passenger	0.88	0.67	0.76	437
ShipType.Tanker	0.77	0.39	0.52	1452
ShipType.Tug	0.84	0.83	0.83	959
accuracy			0.79	7616
macro avg	0.83	0.74	0.77	7616
weighted avg	0.79	0.79	0.78	7616

Figura 4-6: Resultados del experimento 5



	precision	recall	f1-score	support
ShipType.Cargo	0.90	0.96	0.93	3820
ShipType.Fishing	0.91	0.95	0.93	948
ShipType.Passenger	0.88	0.84	0.86	437
ShipType.Tanker	0.94	0.78	0.85	1452
ShipType.Tug	0.92	0.92	0.92	959
accuracy			0.91	7616
macro avg	0.91	0.89	0.90	7616
weighted avg	0.91	0.91	0.91	7616

Figura 4-7: Resultados del experimento 6

4.3.3 Resultados de la preparación de trayectorias: Iteración 2

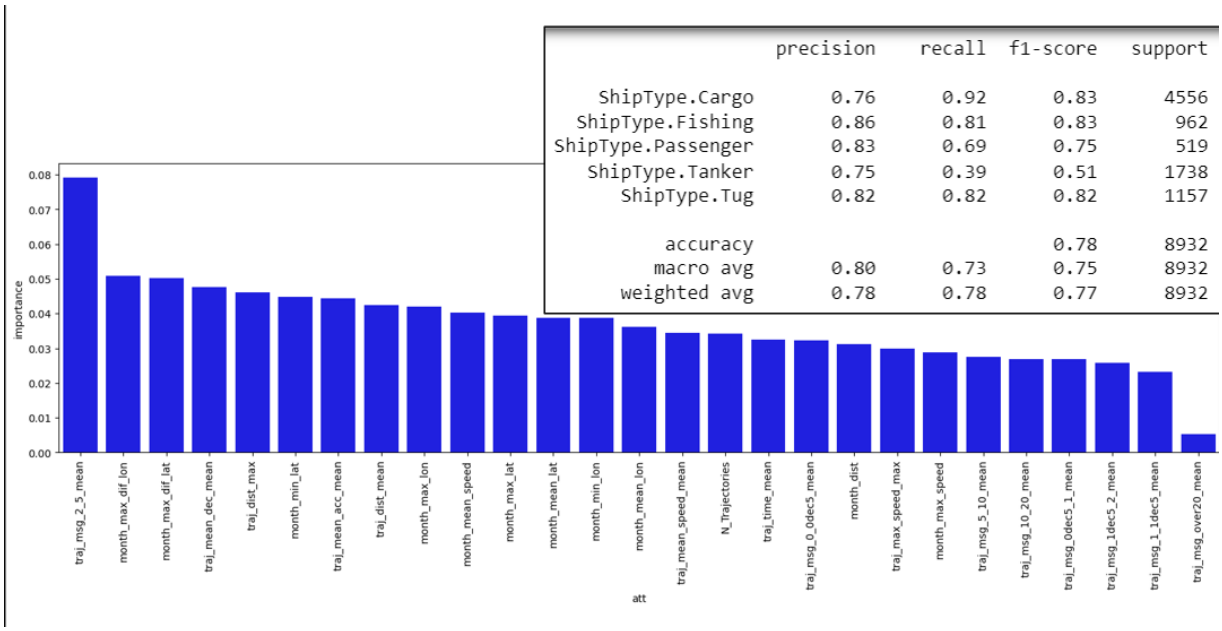
En la segunda iteración, se espera una ligera reducción en la calidad de las predicciones ya que se eliminan un gran número de parámetros para aliviar el procesamiento de las trayectorias. Tal y como viene reflejado en el capítulo anterior, en el apartado de extracción de parámetros del mes, se eliminan ciertos parámetros y se modifican únicamente los parámetros que corresponden al número de mensajes absoluto por rangos velocidad, los cuales se sustituyen por el porcentaje del total de mensajes de la trayectoria en dicho rango de velocidad. Se mantienen, además los umbrales de discretización correspondientes a la anterior iteración (diámetro de 100 metros durante 1 hora). En la Figura 4-8 se muestran los resultados de los parámetros de trayectorias aislados. Tal y como se había previsto, la exactitud disminuye algo menos de un 1% a cambio de la eliminación de 41 parámetros de las 67 iniciales, quedándose únicamente con 26 de ellos.

En cuanto a los parámetros más característicos, cabe mencionar el porcentaje de mensajes a velocidades bajas (entre 2 y 5 nudos) en una trayectoria. Las diferencias máximas de latitud y de

longitud, que podríamos denominar apartamientos máximos de la ruta, también son parámetros característicos para el algoritmo, aunque no presentan particularidades sobre la trayectoria, sino sobre el conjunto de datos mensual. Otros parámetros como aceleraciones y deceleraciones medias de la trayectoria y la distancia media y máxima de cada trayectoria también se muestran relevantes en el algoritmo.

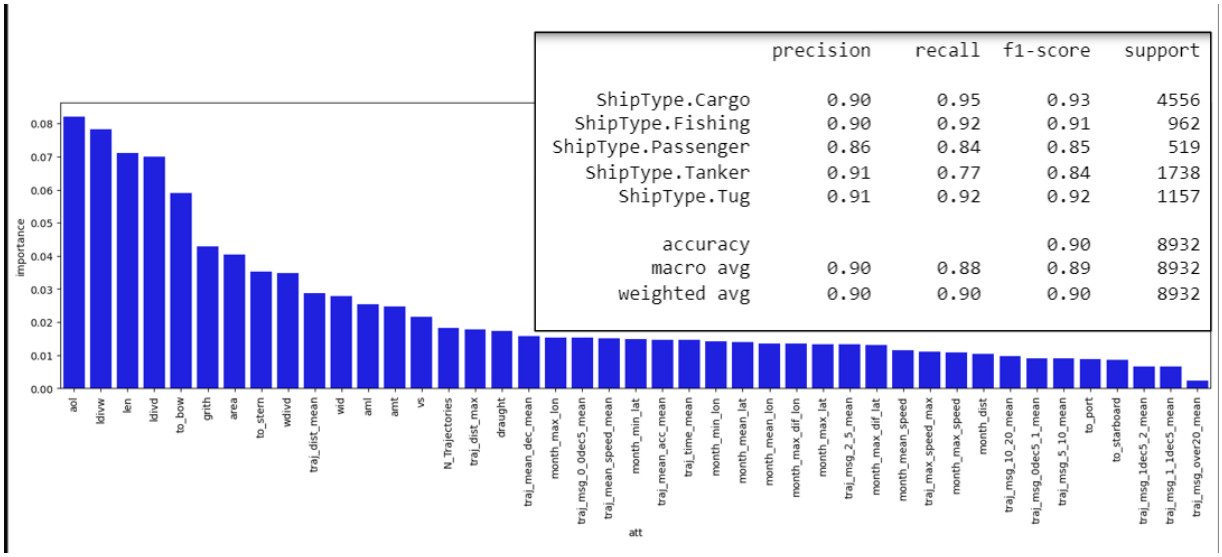
Observando la Figura 4-9, se aprecia una reducción del 1% debida también a la reducción de parámetros. Los parámetros estáticos pasan al primer plano destacándose entre ellos la distancia media de las trayectorias, el número de trayectorias y la distancia máxima de todas las trayectorias realizadas en ese mes. Los parámetros mensuales pasan a segundo plano debido, probablemente, a que los estáticos aportan información más fiable para clasificar ciertos tipos de buques.

Con esta iteración, se demuestra que los parámetros cinemáticos, presentan gran potencial por sí solos, incluyéndose entre ellos algunos no necesariamente relacionados con la generación de trayectorias. Sin embargo, combinándose con los estáticos, adquieren mayor relevancia aquellos que sí provienen de la generación de trayectorias, siempre por detrás de los estáticos.



	precision	recall	f1-score	support
ShipType.Cargo	0.76	0.92	0.83	4556
ShipType.Fishing	0.86	0.81	0.83	962
ShipType.Passenger	0.83	0.69	0.75	519
ShipType.Tanker	0.75	0.39	0.51	1738
ShipType.Tug	0.82	0.82	0.82	1157
accuracy			0.78	8932
macro avg	0.80	0.73	0.75	8932
weighted avg	0.78	0.78	0.77	8932

Figura 4-8: Resultados del experimento 7



	precision	recall	f1-score	support
ShipType.Cargo	0.90	0.95	0.93	4556
ShipType.Fishing	0.90	0.92	0.91	962
ShipType.Passenger	0.86	0.84	0.85	519
ShipType.Tanker	0.91	0.77	0.84	1738
ShipType.Tug	0.91	0.92	0.92	1157
accuracy			0.90	8932
macro avg	0.90	0.88	0.89	8932
weighted avg	0.90	0.90	0.90	8932

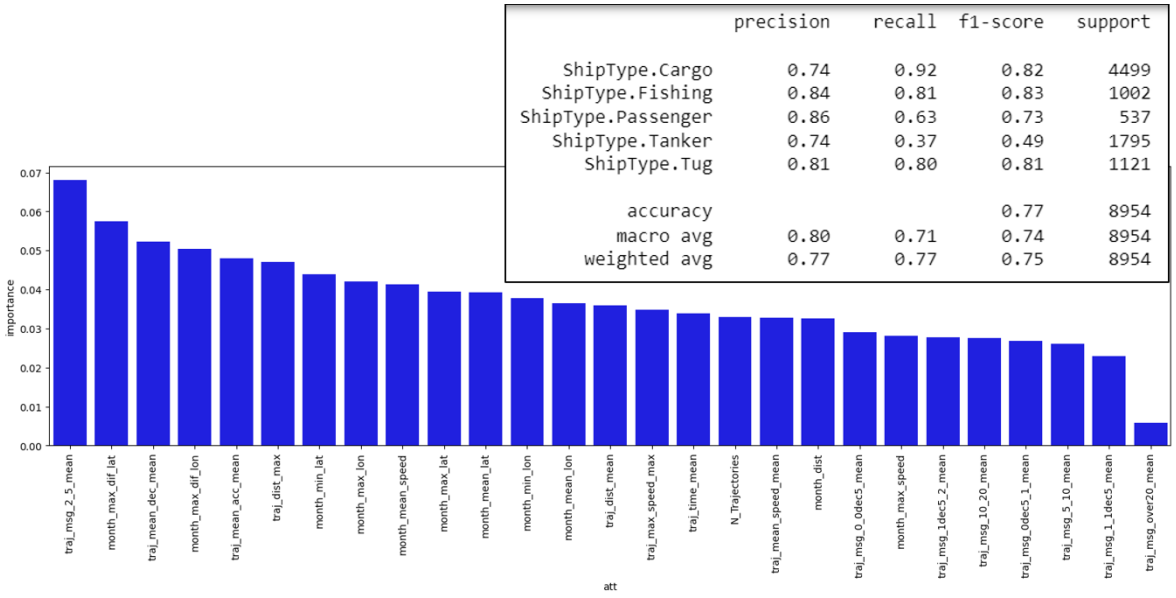
Figura 4-9: Resultados del experimento 8

4.3.4 Resultados de la preparación de trayectorias: Iteración 3

En esta iteración se mantienen los parámetros, aunque se modifican los umbrales de discretización de paradas. A partir de esta iteración se considera una parada cuando el buque transmite durante más de una hora desde un círculo de no más de 50 metros de diámetro. Esto permite, como se observó en el capítulo anterior, generar paradas con menor probabilidad cuando el buque transita a velocidades bajas.

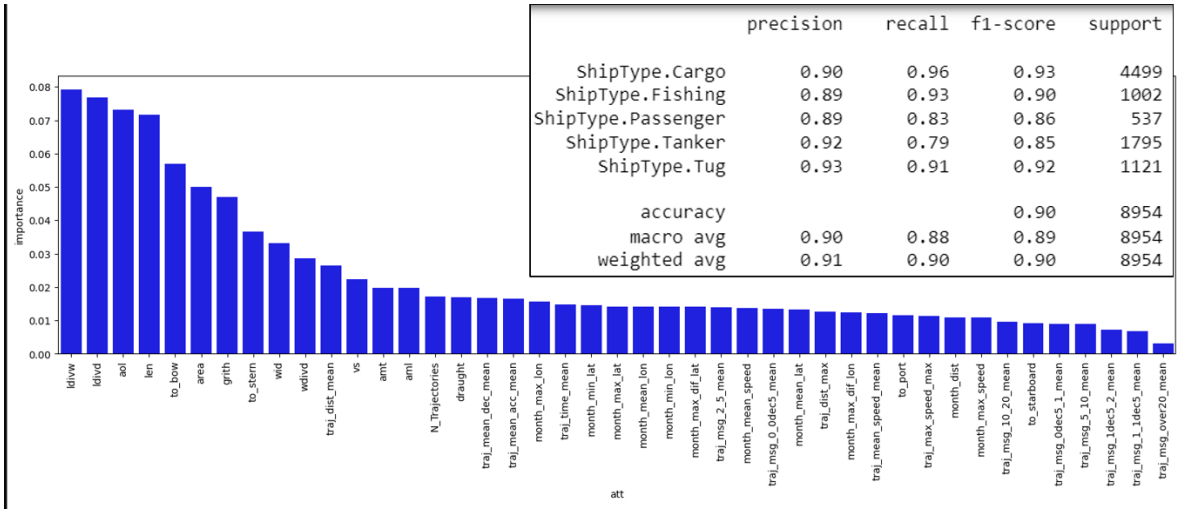
Los resultados utilizando únicamente los parámetros cinemáticos, como se puede observar en la Figura 4-10, muestran una exactitud del 77%. Esto es debido, probablemente a que, a pesar de que la discretización es más precisa, los datos presentan rutas con muy baja cadencia, resultando en un filtro que para rutas bien definidas (las mostradas en los ejemplos del anterior capítulo) discretizan mejor las trayectorias, aunque para el resto de las rutas puede suponer un filtro demasiado restrictivo.

La predicción de la combinación de estos parámetros con los estáticos (Figura 4-11) se mantiene casi constante disminuyendo la columna *f1-score* en un 1% para los pesqueros, curiosamente los que presentan más complicaciones a la hora de generar sus trayectorias debido a su cinemática irregular, y aumentando en un 1% en los buques de pasajeros, que poseen trayectorias y tiempos de paradas bien definidos y que muestran una gran cadencia de transmisión.



	precision	recall	f1-score	support
ShipType.Cargo	0.74	0.92	0.82	4499
ShipType.Fishing	0.84	0.81	0.83	1002
ShipType.Passenger	0.86	0.63	0.73	537
ShipType.Tanker	0.74	0.37	0.49	1795
ShipType.Tug	0.81	0.80	0.81	1121
accuracy			0.77	8954
macro avg	0.80	0.71	0.74	8954
weighted avg	0.77	0.77	0.75	8954

Figura 4-10: Resultados del experimento 9



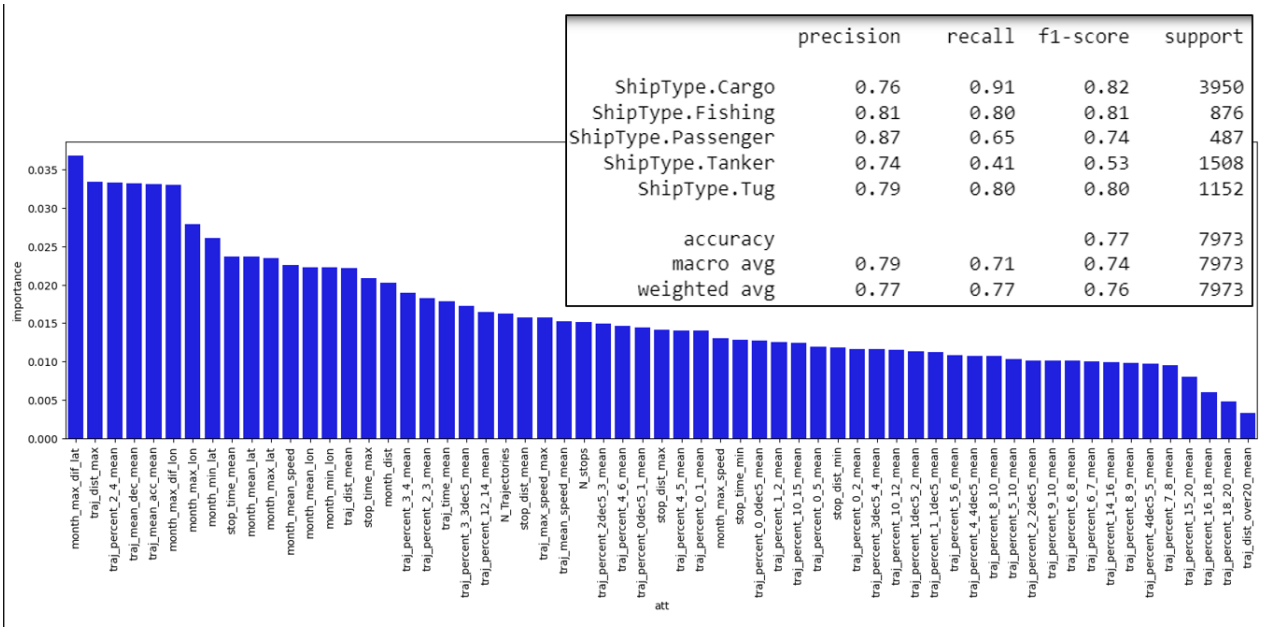
	precision	recall	f1-score	support
ShipType.Cargo	0.90	0.96	0.93	4499
ShipType.Fishing	0.89	0.93	0.90	1002
ShipType.Passenger	0.89	0.83	0.86	537
ShipType.Tanker	0.92	0.79	0.85	1795
ShipType.Tug	0.93	0.91	0.92	1121
accuracy			0.90	8954
macro avg	0.90	0.88	0.89	8954
weighted avg	0.91	0.90	0.90	8954

Figura 4-11: Resultados del experimento 10

4.3.5 Resultados de la preparación de trayectorias: Iteración 4

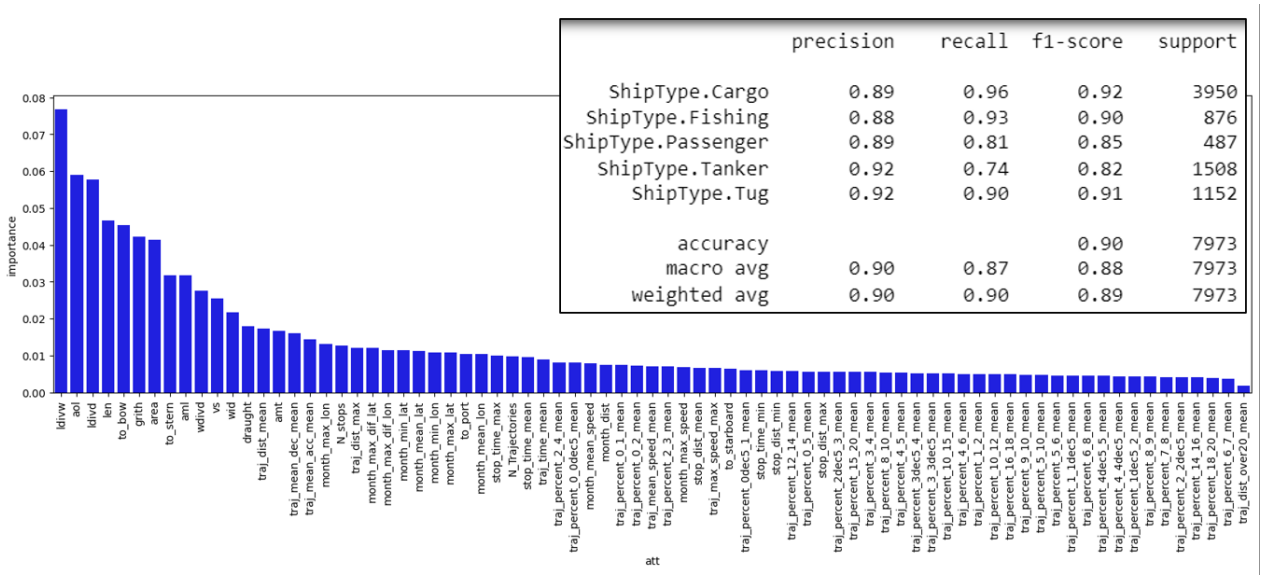
Para esta iteración, se añade una variedad de rangos de velocidad que permita determinar cuáles son los más característicos para cada tipo de buque. Los rangos de velocidad para el parámetro de porcentaje de mensajes por rango de velocidad son los siguientes: velocidades de 0 a 20 nudos en intervalos de 5 nudos, de 0 a 20 en intervalos de 2 de 0 a 10 en intervalos de 1 y de 0 a 5 en intervalos de 0,5. Las predicciones presentan una ligera mejora en ambos experimentos debido al aumento de parámetros, aunque, al ser dependientes unos de otros, se requiere eliminar los menos representativos para no ponderar en exceso los resultados en base a estos parámetros.

Como se observa en ambos resultados (Figuras 4-12 y 4-13), los rangos más característicos son los de incrementos de 2 nudos para velocidades bajas (de 0 a 6 nudos). Por ello, todos los demás rangos se descartan para la siguiente iteración, así como los parámetros de tiempo de parada mínimo y distancia de parada mínima, que suelen ser 0 cuando los datos no son abundantes a la hora de generar las trayectorias, por lo que pierde importancia para el algoritmo.



	precision	recall	f1-score	support
ShipType.Cargo	0.76	0.91	0.82	3950
ShipType.Fishing	0.81	0.80	0.81	876
ShipType.Passenger	0.87	0.65	0.74	487
ShipType.Tanker	0.74	0.41	0.53	1508
ShipType.Tug	0.79	0.80	0.80	1152
accuracy			0.77	7973
macro avg	0.79	0.71	0.74	7973
weighted avg	0.77	0.77	0.76	7973

Figura 4-12: Resultados del experimento 11



	precision	recall	f1-score	support
ShipType.Cargo	0.89	0.96	0.92	3950
ShipType.Fishing	0.88	0.93	0.90	876
ShipType.Passenger	0.89	0.81	0.85	487
ShipType.Tanker	0.92	0.74	0.82	1508
ShipType.Tug	0.92	0.90	0.91	1152
accuracy			0.90	7973
macro avg	0.90	0.87	0.88	7973
weighted avg	0.90	0.90	0.89	7973

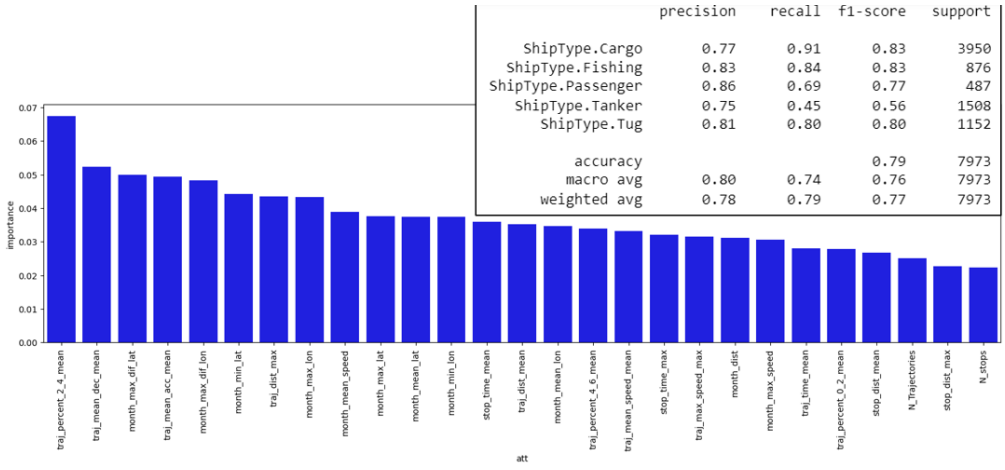
Figura 4-13: Resultados del experimento 12

4.3.6 Resultados de la preparación de trayectorias: Iteración 5

En esta última iteración, se tiene como objetivo dejar los parámetros más característicos para obtener una relación óptima entre parámetros, calidad de la predicción y observaciones empíricas de la generación de trayectorias. Los resultados obtenidos que utilizan únicamente parámetros de trayectorias superan todas las anteriores iteraciones, mejorando la exactitud de la predicción en un 1%, como se aprecia en la Figura 4-14. Los buques con mejores resultados son los cargueros y los pesqueros, en el caso de los cargueros, gracias a su elevada presencia en el conjunto de datos y en el caso de los pesqueros, por su característica cinemática que la diferencia de las demás clases. Para los petroleros aún existen dificultades para su detección. De nuevo, esto se debe a que el petrolero posee características cinemáticas muy similares a las del carguero, pero tiene mucha menor presencia en comparación con él, por ello, aunque la precisión para el petrolero sea del 75% (muy parecida a la del carguero) su exhaustividad disminuye a un 45%, debido probablemente a que muchos de los petroleros fueron clasificados como cargueros sin afectar en exceso la precisión de los cargueros.

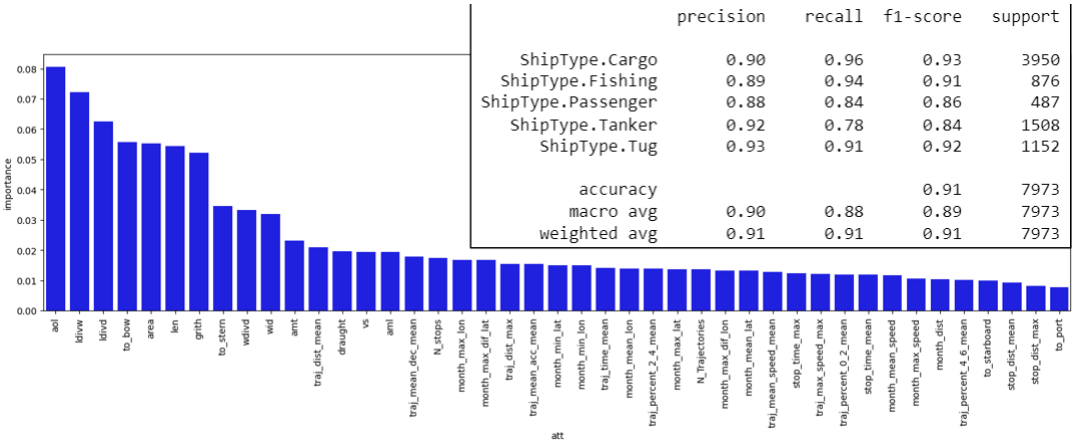
En la Figura 4-15, se observan los resultados de la combinación de parámetros de trayectorias y estáticos. De nuevo, los estáticos se mantienen en el foco de atención, aunque aparecen parámetros como la distancia media de la trayectoria que se posicionan en mejor lugar que algunos estáticos. Otros parámetros como el número de paradas y la deceleración media de la trayectoria también se posicionan en los primeros puestos, estando aún por detrás de los estáticos.

Refiriendo a la Figura 4-1, los resultados de esta última iteración se presentan como los que aportan una mejor calidad en la predicción en relación con los parámetros utilizados, observándose una mejora de un 5% ante el uso exclusivo de los parámetros estáticos y obteniendo una exactitud del 79% haciendo uso únicamente de los datos cinemáticos.



	precision	recall	f1-score	support
ShipType.Cargo	0.77	0.91	0.83	3950
ShipType.Fishing	0.83	0.84	0.83	876
ShipType.Passenger	0.86	0.69	0.77	487
ShipType.Tanker	0.75	0.45	0.56	1508
ShipType.Tug	0.81	0.80	0.80	1152
accuracy			0.79	7973
macro avg	0.80	0.74	0.76	7973
weighted avg	0.78	0.79	0.77	7973

Figura 4-14: Resultados del experimento 13



	precision	recall	f1-score	support
ShipType.Cargo	0.90	0.96	0.93	3950
ShipType.Fishing	0.89	0.94	0.91	876
ShipType.Passenger	0.88	0.84	0.86	487
ShipType.Tanker	0.92	0.78	0.84	1508
ShipType.Tug	0.93	0.91	0.92	1152
accuracy			0.91	7973
macro avg	0.90	0.88	0.89	7973
weighted avg	0.91	0.91	0.91	7973

Figura 4-15: Resultados del experimento 14

5 CONCLUSIONES Y LÍNEAS FUTURAS

En este capítulo se expondrán las conclusiones principales extraídas a la finalización del presente TFG. Cada una de estas conclusiones estará respaldada por una serie de recomendaciones y líneas de acción que se explicarán en el segundo apartado de este capítulo. Estas acciones futuras se han diseñado con el objetivo de perfeccionar la capacidad predictiva del algoritmo empleado.

5.1 Conclusiones

Las conclusiones obtenidas se estructuran en tres pilares esenciales que reflejan el proceso de investigación llevado a cabo en este estudio. Estas conclusiones proporcionan un fundamento sólido para el desarrollo de futuras investigaciones y la mejora continua de las metodologías aplicadas en el análisis de datos para la Armada.

5.1.1 Datos estáticos

Tal y como se apuntó en el capítulo 3, la extracción de parámetros estáticos no era materia de estudio de este TFG, sin embargo, la preparación de los mismos para poder combinarlos con los datos dinámicos es un reto que se completa exitosamente en este trabajo. La solución, que radica en extraer los valores medios para cada buque, ha demostrado ser adecuada ya que ha mantenido las predicciones que se venían sacando en anteriores trabajos, teniendo un conjunto de datos mucho menor, aunque más denso.

Esta reducción de datos tiene como consecuencia una ligera disminución en la exactitud de la predicción, por ello se propone para futuros estudios la extracción de parámetros estadísticos que aporten más información que una sola media, estudiando todos ellos y descartando aquellos menos relevantes para el algoritmo. Además, se propone un filtrado de datos estáticos que contemple los calados con valor 0, a través del cálculo de medias o de asignación de valores aleatorios según el tipo de buque. De esta forma, se perderían menos datos estáticos y consecuentemente, se partiría de un mayor conjunto de datos para la extracción de sus respectivos parámetros dinámicos.

5.1.2 Datos dinámicos

Debido a la gran cantidad de datos dinámicos, el análisis de los mismos se enfocó de una manera distinta que los estáticos, suponiendo inicialmente valores de velocidad, rumbo y dirección fiables, y tasas de transmisión adecuadas para la generación de trayectorias. Tras la extracción de los primeros parámetros, se analizan las distribuciones de rumbos, direcciones, velocidades y tasas de transmisión.

Se observa que los campos de rumbo y dirección no son del todo fiables y no aportan la información que se esperaba de la trayectoria, descartándose para reacción de tiempos de procesamiento. De las velocidades, se obtienen valores no coincidentes con la velocidad obtenida por

cálculo de posiciones consecutivas, principalmente debido a repeticiones de mensajes transmitidos por otra estación. Por ello, se propone la limpieza de valores anómalos de velocidad derivados de las repeticiones de mensajes en dos puntos geográficos muy separados en un intervalo de tiempo muy corto. Esta propuesta queda reflejada para una única trayectoria en el código anexo de preparación de dinámicos, aunque, al no poderse llegar a implementar para el bucle de extracción de parámetros, se propone como mejora para futuros trabajos debiendo ser validada con los resultados que se obtengan. El análisis de las tasas de transmisión aportó una visión nueva que cuestionaba la calidad de los datos, ya que generaban trayectorias poco definibles debido a las bajas tasas de transmisión. Una de las propuestas que se considera imprescindible para la consecución de esta línea de investigación es la obtención de datos por otras vías, como fuentes abiertas que proporcionen flujos continuos AIS.

5.1.3 Generación de trayectorias y extracción de parámetros

En línea con los objetivos de este TFG se logra mejorar sensiblemente la calidad de la predicción de anteriores trabajos con un coste computacional, sin embargo, muy alto. El estudio de trayectorias ha demostrado que puede proveer características muy determinantes para la mejora del algoritmo, aportando, por sí solas, una predicción más precisa que en los anteriores estudios de datos dinámicos y abriendo un nuevo reto al estudio exhaustivo de trayectorias y su validación con mayor cantidad y variedad de experimentos.

Según la literatura relacionada con este trabajo, aún se puede aumentar la calidad de la predicción en gran medida, a través de un gran número de características. Sin embargo, las restricciones temporales debido a los altos tiempos de computación plantean nuevos métodos de generación de trayectorias que sean computacionalmente más eficientes, proponiendo el uso de otras librerías o generando los parámetros de trayectorias de manera directa. Con estos nuevos métodos, se podría plantear incluso el estudio exclusivo de los datos dinámicos, abriendo otros caminos para la obtención de datos por medio de otros sensores como radares, los cuales únicamente proporcionan información de posición y velocidad para un instante determinado.

De la discretización de trayectorias, se extrae que existen distintos métodos para lograr esto, siendo algunos, mejores alternativas para un conjunto de datos con tasas de transmisión muy irregulares a cambio de una mayor carga computacional. El análisis de los resultados que puedan ofrecer estos métodos alternativos se propone para trabajos que partan de un conjunto de datos regular y con altas tasas de transmisión.

5.2 Líneas Futuras

Considerando las conclusiones detalladas en el capítulo anterior, se proponen varias líneas futuras de investigación.

5.2.1 Mejora en la extracción de datos estáticos

Se sugiere considerar aquellos MMSI descartados en la preparación de los estáticos, por ejemplo, aquellos con calado 0, de manera que, aunque los datos estáticos permanezcan vacíos, los parámetros dinámicos intervendrían en la predicción. De esta forma, aunque no se aumentaría el número de datos estáticos, se partiría de un mayor conjunto de datos para la extracción de sus respectivos parámetros dinámicos.

5.2.2 Refinamiento del análisis de datos dinámicos

Se propone la investigación de métodos de limpieza de datos anómalos y la exploración de fuentes adicionales de datos AIS para abordar problemas de baja tasa de transmisión. Además, se sugiere la exploración de técnicas de filtrado y compresión más avanzadas para mejorar la calidad de las trayectorias y optimizar tiempos de cálculo.

5.2.3 Optimización en la generación de trayectorias y extracción de parámetros

Se propone investigar métodos alternativos más eficientes y recopilar datos de otros sensores, como radares, para obtener información adicional. Además, se sugiere la investigación de diferentes métodos de discretización de trayectorias para identificar las mejores alternativas según las características específicas del conjunto de datos disponible. Estas líneas futuras de investigación tienen el potencial de mejorar significativamente la calidad y la precisión de los análisis de datos marítimos para aplicaciones en la Armada.

6 BIBLIOGRAFÍA

[1] A. P. Galguera, «El mar Rojo: un nuevo reto para el comercio mundial», Cinco Días. Accedido: 16 de enero de 2024. [En línea]. Disponible en: https://cincodias.elpais.com/opinion/2024-01-16/el-mar-rojo-un-nuevo-reto-para-el-comercio-mundial.html

[2] «*Merchant fleet – UNCTAD Handbook of Statistics 2023*». Accedido: 16 de enero de 2024. [En línea]. Disponible en: https://hbs.unctad.org/merchant-fleet/

[3] S. Baeg y T. Hammond, «*Ship Type Classification Based on the Ship Navigating Trajectory and Machine Learning*», presentado en Joint Proceedings of the ACM IUI Workshops 2023, March 2023, Sydney, Australia.

[4] A. Harati-Mokhtari, A. Wall, P. Brooks, y J. Wang, «*Automatic Identification System (AIS): Data Reliability and Human Error Implications*», *J. Navig.*, vol. 60, n.º 3, pp. 373-389, sep. 2007, doi: 10.1017/S0373463307004298.

[5] «Resolución 11197/2023, de 29 de junio, de la Secretaria de Estado de Defensa, por la que se aprueba la Estrategia de desarrollo, implantación y uso de la Inteligencia Artificial en el Ministerio de Defensa.» [En línea]. Disponible en: https://publicaciones.defensa.gob.es/media/downloadable/files/links/2/0/20230706.pdf

[6] Almirante Jefe del Estado Mayor de la Armada (AJEMA) y Ministerio de Defensa, «Líneas generales de la Armada 2022». 2022.

[7] G. Rodríguez Casajús, B. Barragáns Martínez (advisor), y P. Sendín Raña (advisor), «Predicción de tipo de buque utilizando datos AIS y técnicas de inteligencia artificial», abr. 2022, Accedido: 15 de marzo de 2024. [En línea]. Disponible en: http://calderon.cud.uvigo.es/handle/123456789/646

[8] A. González Rodríguez, «Clasificación de Tipos de Buque mediante Imágenes en Abierto Aplicando Metodología de Inteligencia Artificial». Trabajo de Fin de Formación.

[9] R. Fernández Martín, B. (advisor) Barragáns Martínez, y P. (advisor) Sendín Raña, «Validación de una serie de parámetros estáticos y dinámicos para la predicción de tipo de buque utliizando técnicas de inteligencia artificial», abr. 2023, Accedido: 15 de marzo de 2024. [En línea]. Disponible en: http://calderon.cud.uvigo.es/handle/123456789/677

[10] I. Gandarillas Carrara, B. (advisor) Barragáns Martínez, y P. (advisor) Sendín Raña, «Predicción de tipo de buque utilizando información de áreas de actividad y técnicas de inteligencia artificial», abr. 2023, Accedido: 15 de marzo de 2024. [En línea]. Disponible en: http://calderon.cud.uvigo.es/handle/123456789/680

[11] R.- ASALE y RAE, «inteligencia | Diccionario de la lengua española», «Diccionario de la lengua española» - Edición del Tricentenario. Accedido: 9 de enero de 2024. [En línea]. Disponible en: https://dle.rae.es/inteligencia

[12] «*A proposal for the Dartmouth Summer Research Project on artificial inteligence*». Accedido: 9 de enero de 2024. [En línea]. Disponible en: https://www-formal.stanford.edu/jmc/history/dartmouth/dartmouth.html
[13] *Judgment under Uncertainty: Heuristics and Biases.* Cambridge University Press.
[14] «¿Qué es la inteligencia artificial y cómo se usa? | Noticias | Parlamento Europeo». Accedido: 24 de enero de 2024. [En línea]. Disponible en: https://www.europarl.europa.eu/news/es/headlines/society/20200827STO85804/que-es-la-inteligencia-artificial-y-como-se-usa
[15] «Inteligencia Artificial para mejorar la eficacia de las Fuerzas Armadas». Accedido: 22 de enero de 2024. [En línea]. Disponible en: https://www.larazon.es/espana/defensa/inteligencia-artificial-mejorar-eficacia-fuerzas-armadas_20240110659ebd0e67d53e0001df511d.html
[16] J. Díaz del Río Durán, «Futuro Gemelo Digital (GD) de la F-110», *Revista General de la Marina*, junio de 2020. Accedido: 26 de febrero de 2024. [En línea]. Disponible en: https://armada.defensa.gob.es/archivo/rgm/2020/06/rgmjunio20cap7.pdf
[17] J. Díaz del Río Durán, «Mantenimiento Inteligente en la Armada: En Vanguardia y Trazando un Nuevo Futuro», *Revista General de la Marina*, noviembre de 2021. Accedido: 26 de febrero de 2024. [En línea]. Disponible en: https://armada.defensa.gob.es/archivo/rgm/2021/11/RGMNoviembre2021cap07.pdf
[18] J. R. Conforto Sesto, «Vehículos Autónomos Submarinos: Nuevos Actores en las Operaciones Navales», *Revista General de la Marina*, junio de 2009. [En línea]. Disponible en: https://armada.defensa.gob.es/archivo/rgm/2009/06/cap07.pdf
[19] Dr Peter Svenmarck, Dr Linus Luotsinen, Dr Mattias Nilsson, y Dr Johan Schuber, «*Possibilities and Challenges for Artificial Intelligence in Military Applications*», *NATO STO-MP-IST-160*.
[20] D. P. Williams, «*Underwater target classification in synthetic aperture sonar imagery using deep convolutional neural networks*», en *2016 23rd International Conference on Pattern Recognition (ICPR)*, Cancun: IEEE, dic. 2016, pp. 2497-2502. doi: 10.1109/ICPR.2016.7900011.
[21] K. Denos, M. Ravaut, A. Fagette, y H.-S. Lim, «*Deep learning applied to underwater mine warfare*», en *OCEANS 2017 - Aberdeen*, jun. 2017, pp. 1-7. doi: 10.1109/OCEANSE.2017.8084910.
[22] G. Kumar, K. Kumar, y M. Sachdeva, «*The use of artificial intelligence based techniques for intrusion detection: a review*», *Artif. Intell. Rev.*, vol. 34, n.º 4, pp. 369-387, dic. 2010, doi: 10.1007/s10462-010-9179-5.
[23] Á. G. de Ágreda y I. M. Herranz, «Usos militares de la inteligencia artificial, la automatización y la robótica», Publicaciones de Defensa. 2020. Accedido: 20 de febrero de 2024. [En línea]. Disponible en: https://publicaciones.defensa.gob.es/
[24] E.-M. Sébastien (www.exolab.fr), «*Numalis - Autonomous military drones soon equipped with AI, reality or fiction?*» Accedido: 12 de febrero de 2024. [En línea]. Disponible en: https://numalis.com/publications-37-autonomous_military_drones_soon_equipped_with_ai_reality_or_fiction.php
[25] E. Ackerman, «*A Navy SEAL, a Quadcopter, and a Quest to Save Lives in Combat*», *Wired*. Accedido: 12 de febrero de 2024. [En línea]. Disponible en: https://www.wired.com/story/shield-ai-quadcopter-military-drone/
[26] W.-M. Lee, *Python Machine Learning*, 1er ed. Wiley, 2019.
[27] «Clasificación vs. clusterización: una explicación práctica». Accedido: 29 de enero de 2024. [En línea]. Disponible en: https://blog.bismart.com/la-clasificación-y-la-clusterización-una-explicación-práctica
[28] M. Rodelgo Lacruz, B. Barragáns Martínez, N. Fernández García, P. Sendín Raña, y A. Suárez García, «Análisis de datos AIS en tiempo real para la detección de anomalías en el entorno marítimo», *Libro Resúmenes DESEID 2022*, Accedido: 20 de febrero de 2024. [En línea]. Disponible en: https://www.tecnologiaeinnovacion.defensa.gob.es/es-

es/Presentacion/deseid_2022/Documents/Libro%20de%20res%C3%BAmenes%20DESEID%202022.pdf

[29] «¿Qué es el aprendizaje supervisado? | IBM». Accedido: 1 de febrero de 2024. [En línea]. Disponible en: https://www.ibm.com/es-es/topics/supervised-learning

[30] «¿Qué es el aprendizaje no supervisado? | IBM». Accedido: 1 de febrero de 2024. [En línea]. Disponible en: https://www.ibm.com/es-es/topics/unsupervised-learning

[31] F. P. Montero y N. P. Mariné, «Descomposición en valores singulares: introducción y aplicaciones. Análisis de componentes principales (PCA) y descomposición en valores singulares (SVD)», *Univ. Oberta Catalunya*, PID_00262387.

[32] I. D. and A. Team, «*AI vs. Machine Learning vs. Deep Learning vs. Neural Networks: What's the difference?*», IBM Blog. Accedido: 1 de febrero de 2024. [En línea]. Disponible en: https://www.ibm.com/blog/ai-vs-machine-learning-vs-deep-learning-vs-neural-networks/www.ibm.com/blog/ai-vs-machine-learning-vs-deep-learning-vs-neural-networks

[33] J. Hurwitz, «*Machine Learning For Dummies®, IBM Limited Edition*», 2018.

[34] «sklearn.ensemble.RandomForestClassifier», scikit-learn. Accedido: 13 de marzo de 2024. [En línea]. Disponible en: https://scikit-learn/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html

[35] «*Random Forests for Complete Beginners - victorzhou.com*». Accedido: 13 de marzo de 2024. [En línea]. Disponible en: https://victorzhou.com/blog/intro-to-random-forests/

[36] «*What is the k-nearest neighbors algorithm? | IBM*». Accedido: 14 de marzo de 2024. [En línea]. Disponible en: https://www.ibm.com/topics/knn

[37] C. Cortes y V. Vapnik, «*Support-vector networks*», *Mach. Learn.*, vol. 20, n.º 3, pp. 273-297, sep. 1995, doi: 10.1007/BF00994018.

[38] A. Saini, «*Gradient Boosting Algorithm: A Complete Guide for Beginners*», Analytics Vidhya. Accedido: 14 de marzo de 2024. [En línea]. Disponible en: https://www.analyticsvidhya.com/blog/2021/09/gradient-boosting-algorithm-a-complete-guide-for-beginners/

[39] R. Agrawal, «*Master Polynomial Regression With Easy-to-Follow Tutorials*», Analytics Vidhya. Accedido: 14 de marzo de 2024. [En línea]. Disponible en: https://www.analyticsvidhya.com/blog/2021/07/all-you-need-to-know-about-polynomial-regression/

[40] A. Gelman, J. B. Carlin, H. S. Stern, D. B. Dunson, A. Vehtari, y D. B. Rubin, «*Bayesian Data Analysis*», 3.ª ed.

[41] M. García de la Gándara, «El COVAM de la Armada al servicio de la Comunidad Marítima», *Inst. Esp. Estud. Estratég.*, n.º 55/2011.

[42] «ENCOMAR - Entorno Colaborativo Marítimo de la Armada». Accedido: 12 de febrero de 2024. [En línea]. Disponible en: https://encomar.covam.es/en

[43] «Convenio internacional para la seguridad de la vida humana en el mar, 1974 (Convenio SOLAS)». Accedido: 13 de marzo de 2024. [En línea]. Disponible en: https://www.imo.org/es/About/Conventions/Pages/International-Convention-for-the-Safety-of-Life-at-Sea-(SOLAS)%2C-1974.aspx

[44] B. J. Rhodes, N. A. Bomberger, M. Seibert, y A. M. Waxman, «*Maritime Situation Monitoring and Awareness Using Learning Mechanisms*», en *MILCOM 2005 - 2005 IEEE Military Communications Conference*, Atlantic City, NJ, USA: IEEE, 2005, pp. 1-7. doi: 10.1109/MILCOM.2005.1605756.

[45] B. J. Rhodes, N. A. Bomberger, y M. Zandipour, «*Probabilistic associative learning of vessel motion patterns at multiple spatial scales for maritime situation awareness*», en *2007 10th International Conference on Information Fusion*, Quebec City, QC, Canada: IEEE, jul. 2007, pp. 1-8. doi: 10.1109/ICIF.2007.4408127.

[46] R. Laxhammar, «*Anomaly detection for sea surveillance*», 11th International Conference on Information Fusion 2008.

[47] I. Mohíno Hernández, Ed., «De las Células a los Bits», presentado en 12th International Conference on Information Fusion, Seattle, Washington, USA: IEEE, jul. 2009.

[48] S. Mascaro, A. E. Nicholso, y K. B. Korb, «*Anomaly detection in vessel tracks using Bayesian networks*», *Int. J. Approx. Reason.*, vol. 55, n.º 1, pp. 84-98, ene. 2014, doi: 10.1016/j.ijar.2013.03.012.

[49] N. Damastuti, A. S. Aisjah, y A. Masroeri, «*Vessel Classifying and Trajectory Based on Automatic Identification System Data*», *IOP Conf. Ser. Earth Environ. Sci.*, vol. 830, n.º 1, p. 012049, sep. 2021, doi: 10.1088/1755-1315/830/1/012049.

[50] P. Kraus, C. Mohrdieck, y F. Schwenker, «*Ship classification based on trajectory data with machine-learning methods*», en *2018 19th International Radar Symposium (IRS)*, jun. 2018, pp. 1-10. doi: 10.23919/IRS.2018.8448028.

[51] K. Sheng, Z. Liu, D. Zhou, A. He, y C. Feng, «*Research on Ship Classification Based on Trajectory Features*», *J. Navig.*, vol. 71, n.º 1, pp. 100-116, ene. 2018, doi: 10.1017/S0373463317000546.

[52] D. Luo, P. Chen, J. Yang, X. Li, y Y. Zhao, «*A New Classification Method for Ship Trajectories Based on AIS Data*», *J. Mar. Sci. Eng.*, vol. 11, n.º 9, Art. n.º 9, sep. 2023, doi: 10.3390/jmse11091646.

[53] R. Blagojevic, S. H.-H. Chang, y B. Plimmer, «*The Power of Automatic Feature Selection: Rubine on Steroids*» The Eurographics Association, 2010. doi: 10.2312/SBM/SBM10/079-086.

[54] «*Welcome to Python.org*». Accedido: 20 de marzo de 2024. [En línea]. Disponible en: https://www.python.org/

[55] «*Anaconda | The World's Most Popular Data Science Platform*», Anaconda. Accedido: 6 de enero de 2024. [En línea]. Disponible en: https://www.anaconda.com/

[56] «*JupyterLab Documentation — JupyterLab 4.2.0a1 documentation*». Accedido: 20 de marzo de 2024. [En línea]. Disponible en: https://jupyterlab.readthedocs.io/en/latest/

[57] «*Welcome to Pyais's documentation! — Pyais 2.1.1 documentation*». Accedido: 28 de marzo de 2024. [En línea]. Disponible en: https://pyais.readthedocs.io/en/latest/

ANEXO I: IMPLICACIONES SOCIALES, Y/O ECONÓMICAS, Y/O AMBIENTALES

En cumplimiento del Sello Internacional de Calidad EUR-ACE, se ha realizado un análisis de las implicaciones sociales, económicas y ambientales del presente TFG. Durante el desarrollo de la memoria del TFG, se han abordado diversas secciones que recogen la conciencia relativa a los aspectos sociales económicos y ambientales, que se resumirán a continuación para dejar constancia de dichas implicaciones de este trabajo.

En primer lugar, debe destacarse la relevancia de la industria naviera en la economía global y su impacto en la creación y mantenimiento de empleos. El estudio de técnicas que permitan su gestión eficiente aporta gran capacidad para el desarrollo sostenible en este sector. Asimismo, se considera que las implicaciones sociales de la implementación de tecnologías de inteligencia artificial en la industria marítima influirán directamente en la formación y capacitación continua de los trabajadores para adaptarse a los avances.

Por otro lado, se propone que la predicción de tipos de buques mediante técnicas de inteligencia artificial puede contribuir en dos líneas de acción que mejoren la sostenibilidad del sector. La primera y principal línea de investigación, es contribuir a la detección de anomalías en el entorno marítimo, que permitirá a las autoridades, a través del COVAM a impedir actividades ilícitas que puedan suponer un riesgo para el entorno marítimo los para buques que transitan libremente por dicho entorno. La segunda línea contribuye al desarrollo de técnicas de análisis de trayectorias para posibles estudios de optimización de las mismas, reduciendo así el consumo de combustible y las emisiones contaminantes.

ANEXO II: REFLEXIONES ÉTICAS Y SOCIALES

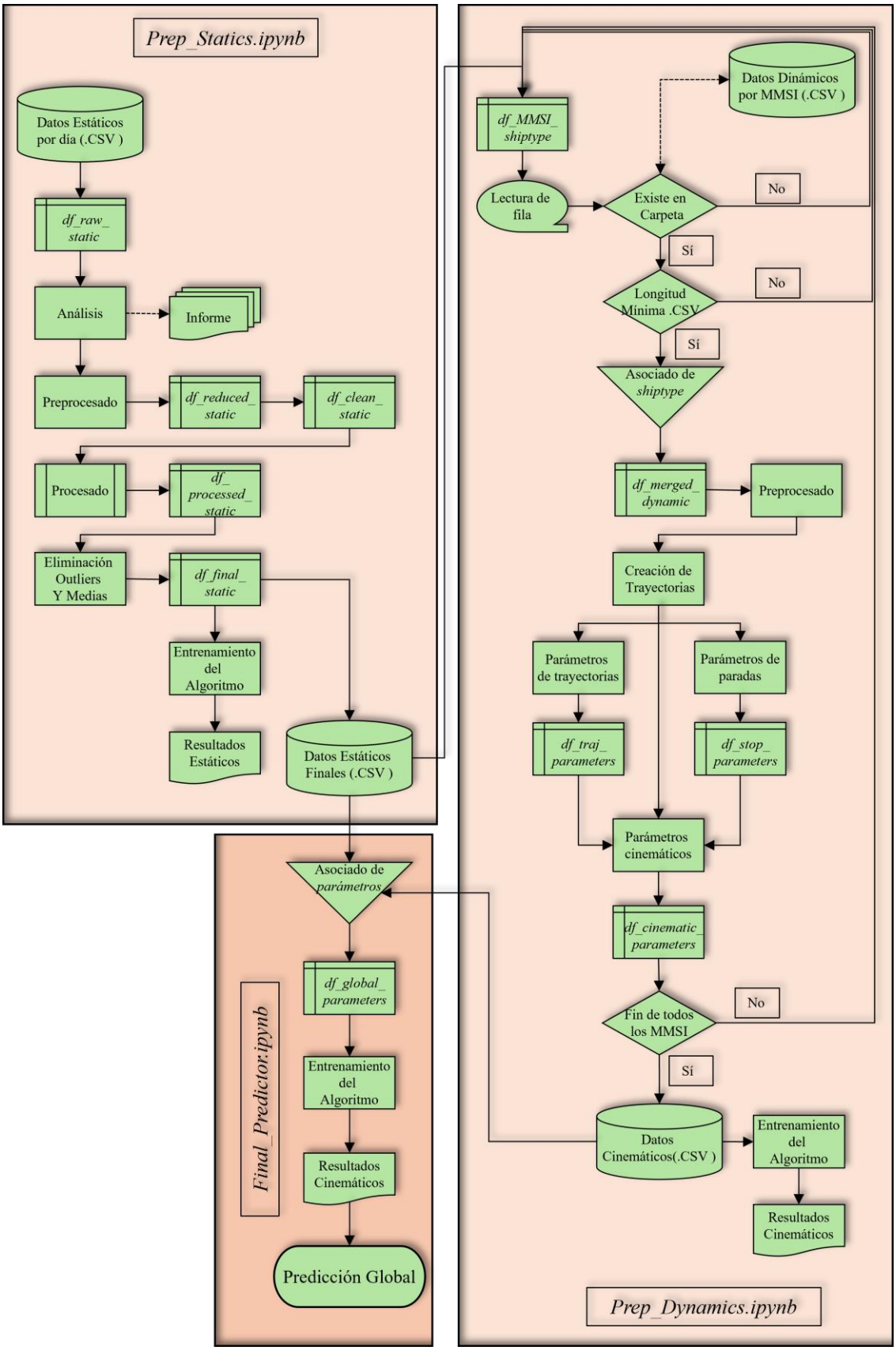
El avance tecnológico en la predicción de tipos de buques mediante inteligencia artificial representa un hito significativo en la modernización y optimización de la vigilancia marítima. Con estas tecnologías, se pretende ofrecer la capacidad de analizar grandes volúmenes de datos y generar predicciones precisas sobre el comportamiento y la clasificación de los buques, lo que conduce a mejoras sustanciales en términos de eficiencia operativa y seguridad en la navegación.

Sin embargo, este progreso tecnológico también plantea una serie de desafíos éticos y sociales que no deben pasarse por alto. Uno de los aspectos más críticos es la protección de la privacidad de los datos, ya que el uso de algoritmos de inteligencia artificial implica la recopilación y análisis de información sensible sobre las operaciones marítimas y los patrones de navegación. Existe el riesgo de que esta información sea utilizada de manera indebida o compartida sin el consentimiento adecuado, lo que podría comprometer la seguridad y la competitividad de las empresas navieras, así como la privacidad de los individuos involucrados.

Además, la implementación de sistemas de inteligencia artificial en la Armada puede plantear preocupaciones en términos de equidad y justicia laboral. Si bien estas tecnologías tienen el potencial de aumentar la eficiencia y reducir los costes operativos, también podrían conducir a la automatización de ciertas tareas y la disminución de la demanda de mano de obra humana. Esto podría tener un impacto negativo en los trabajadores del sector.

Por lo tanto, es esencial que tanto los desarrolladores como los usuarios de estas herramientas sean plenamente conscientes de su impacto ético y social, y trabajen en la implementación de medidas que mitiguen posibles efectos negativos. Esto incluye la adopción de políticas de protección de datos sólidas, la promoción de la transparencia en el uso de algoritmos de inteligencia artificial y la inversión en programas de formación y reciclaje laboral para garantizar una transición justa hacia un entorno laboral digitalizado. Además, es fundamental que se fomente un diálogo abierto y colaborativo entre todas las partes interesadas, incluidos los reguladores, las empresas, los trabajadores y la sociedad en general, para abordar los desafíos éticos y sociales asociados con el avance tecnológico en la Armada.

ANEXO III: DIAGRAMA DE FLUJO DEL TRABAJO



ANEXO IV: DESARROLLO DEL CÓDIGO

PREPARACIÓN DE ESTÁTICOS (Prep_Statics.ipynb)

LIBRERÍAS

In [1]:
```
# Importación de librerías
import csv
import os
# Lectura de archivos en directorios
import pandas as pd
# Análisis de datos que proporciona estructuras de datos
import numpy as np
# Computación científica en Python
from tqdm import tqdm
# Barras de progreso en bucles iterativos
import seaborn as sns
# Visualización de datos basada en Matplotlib
import matplotlib.pyplot as plt
# Visualización de datos en 2D.
from sklearn.model_selection import train_test_split, GridSearchCV, GroupShuffleSplit
# Proporciona funciones relacionadas con la selección y evaluación de modelos
from sklearn.preprocessing import MinMaxScaler
# Normalización de datos
from sklearn.metrics import classification_report
# Muestra los resultados del algoritmo
from sklearn.ensemble import RandomForestClassifier
# Clasificadores de bosque aleatorio implementado en scikit-learn
from sklearn.pipeline import Pipeline
# Simplificar el flujo de trabajo del modelo
from ydata_profiling import ProfileReport
# Útil para análisis exploratorio de datos (EDA)
import re
# Maneja strings de datos
```

DEFINICIÓN DE FUNCIÓN DE PROCESADO DE ESTÁTICOS

In [2]:
```
#Función para el cálculo de atributos estáticos
def statics_processing(df):
 df['len'] = df.to_bow+df.to_stern # Eslora
 df['wid'] = df.to_port+df.to_starboard # Manga
 df['ldivw'] = df['len'] / df['wid']# Cociente entre eslora y manga
 df['ldivd'] = df['len'] / df['draught']# Cociente entre eslora y calado
 df['wdivd'] = df['wid'] / df['draught'] # Cociente entre manga y calado
 df['area'] = df.len*df.wid # Area de la cubierta aproximada= eslora x manga
 df['grith'] = df.len+df.wid # Suma de eslora y manga
 df['aml'] =df.len*df.draught # area longitudinal sumergida
 df['amt'] =df.wid*df.draught # area transversal sumergida
 df['vs'] =df.len*df.draught*df.wid # volumen sumergido
 df['aol'] = df['to_bow'] / df['len'] # proporción que supone a sobre la eslora total
 return df
```

LECTURA DE ARCHIVOS Y ALMACENAMIENTO EN DF

In [3]:
```
%%time
# Explicación de como se genera raw_static_data.csv
# Con pocos datos es viable trabajar con todos a la vez (útil para EDA)
# Almacena los nombres solo los archivos que contienen "static" en el nombre
lista_archivos_static = [archivo for archivo in os.listdir('2023-24') if "static" in archivo]
# Para ver los nombres de los archivos: print(lista_archivos_static)
# Crea un DataFrame vacío para ir almacenando los datos de todos los csv estáticos
df_raw_static = pd.DataFrame()
# Itera sobre los archivos "static" y los carga en el DataFrame
for archivo_static in lista_archivos_static:
    df_temporary = pd.read_csv(os.path.join('2023-24', archivo_static))
    df_raw_static = pd.concat([df_raw_static,df_temporary], ignore_index=True)
# Ahora df_raw_static contiene todos los datos de los archivos estáticos
```

```
CPU times: total: 1min 42s
Wall time: 2min 9s
```

In [ ]:
```
# Guarda todo el df en un solo csv
df_raw_static.to_csv('raw_static_data.csv', index = False)
```

In [3]:
```
# Lee el csv
df_raw_static = pd.read_csv('raw_static_data.csv')
```

ANÁLISIS EN BRUTO DEL DF (EDA)

In [ ]:
```
# Análisis automático predefinido por librerías para ver correlaciones y parámetros genéricos
ProfileReport(df_raw_static, title="eda")
```

Forma del DF

In [4]:
```
df_raw_static.shape
```

Out[4]:
```
(93850348, 9)
```

In [5]:
```
df_raw_static.head(3)
```

Out[5]:

	timestamp	MMSI	IMO	to_bow	to_stern	to_port	to_starboard	draught	shiptype
0	1672527350	352489000	9683661	276	57	26	34	21.2	ShipType.Tanker
1	1672527350	273810100	8907125	37	68	17	3	8.8	ShipType.Fishing
2	1672527350	273448240	8509155	29	65	12	3	6.0	ShipType.Fishing

In [6]:
```
df_raw_static.tail(3)
```

Out[6]:

	timestamp	MMSI	IMO	to_bow	to_stern	to_port	to_starboard	draught	shiptype
93850345	1675205546	229863000	1012476	37	19	5	5	3.1	ShipType.PleasureCraft
93850346	1675205546	314537000	9573921	146	23	19	8	5.6	ShipType.Cargo
93850347	1675205547	538071332	1013004	60	47	7	7	4.3	ShipType.PleasureCraft

Mensajes con MMSI repetidos

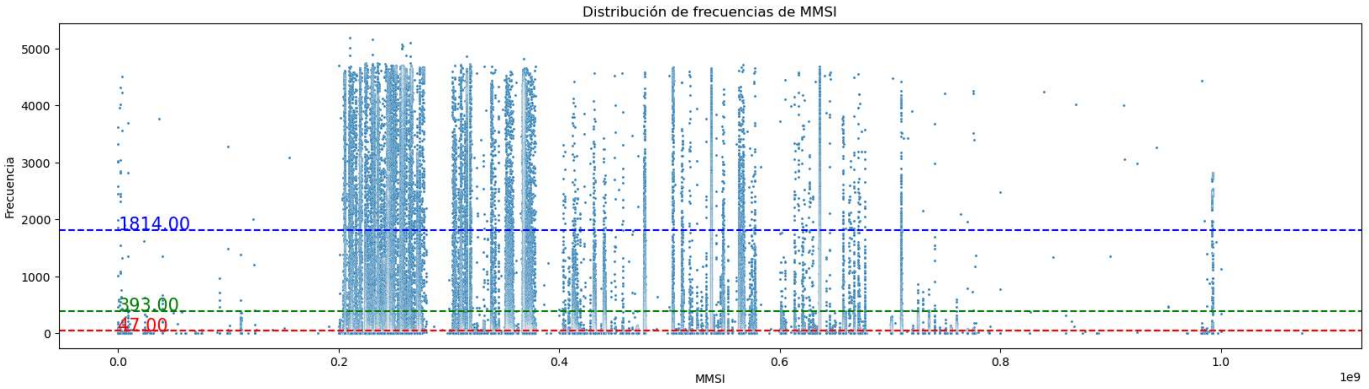
In [5]:
```
print(df_raw_static['MMSI'].value_counts())
```

```
MMSI
210231000    5185
230938210    5164
264900021    5100
257088680    5070
258020030    5042
             ...
525125012       1
273336280       1
657211400       1
244060589       1
355401000       1
Name: count, Length: 85834, dtype: int64
```

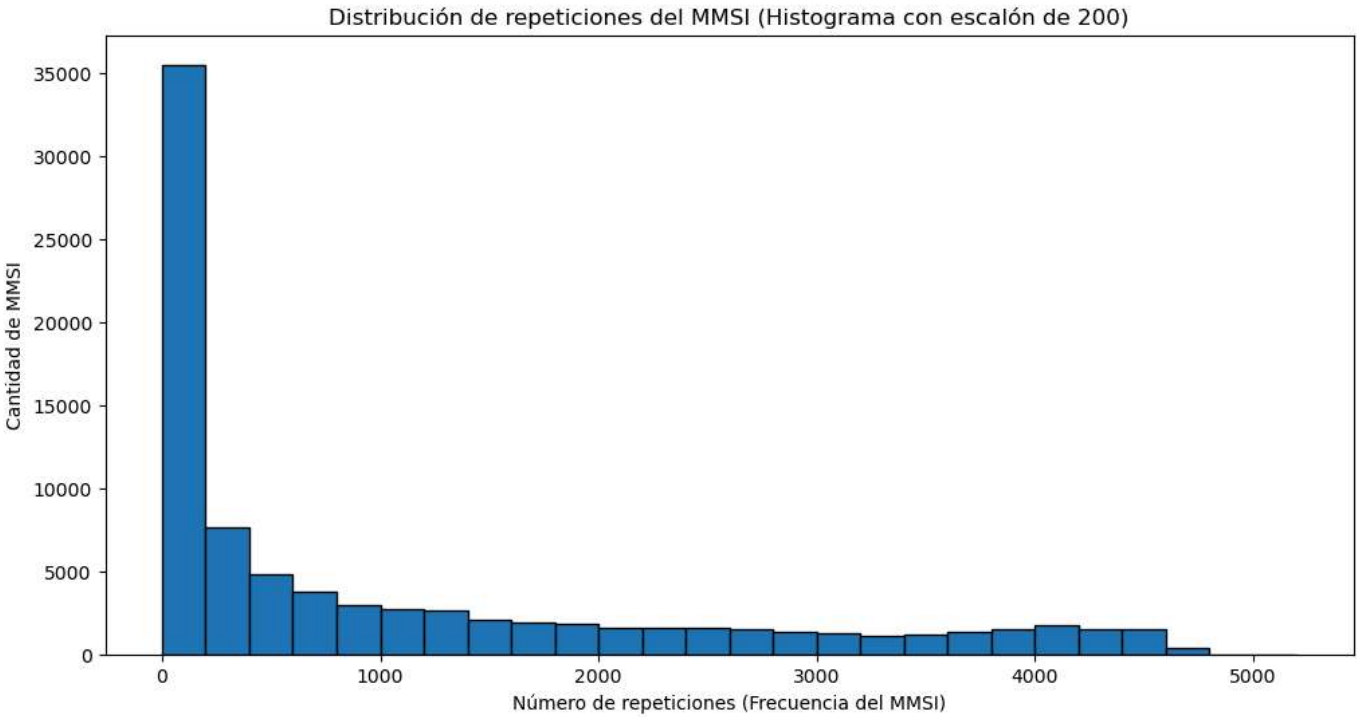
In [8]:
```
df_raw_static['MMSI'].value_counts().head(42917)
```

Out[8]:
```
MMSI
210231000    5185
230938210    5164
264900021    5100
257088680    5070
258020030    5042
             ...
377489000     393
357952000     393
228794000     393
374298000     393
538002510     393
Name: count, Length: 42917, dtype: int64
```

In [9]:
```
plt.figure(figsize=(20, 5))
sns.scatterplot(
    x=df_raw_static['MMSI'].value_counts().index,
    y=df_raw_static['MMSI'].value_counts().values,
    alpha=0.9, s=5)
quartiles = df_raw_static['MMSI'].value_counts().quantile([0.25, 0.5, 0.75])
plt.axhline(quartiles[0.25], linestyle='--', color='red', label='Cuartil 1')
plt.axhline(quartiles[0.5], linestyle='--', color='green', label='Cuartil 2 (Mediana)')
plt.axhline(quartiles[0.75], linestyle='--', color='blue', label='Cuartil 3')
# Añadir etiquetas de texto para mostrar los valores de los cuartiles
plt.text(0.9, quartiles[0.25], f'{quartiles[0.25]:.2f}', color='red', fontsize=15)
plt.text(0.9, quartiles[0.5], f'{quartiles[0.5]:.2f}', color='green', fontsize=15)
plt.text(0.9, quartiles[0.75], f'{quartiles[0.75]:.2f}', color='blue', fontsize=15)

plt.xlabel('MMSI')
plt.ylabel('Frecuencia')
plt.title('Distribución de frecuencias de MMSI')
plt.show()
```

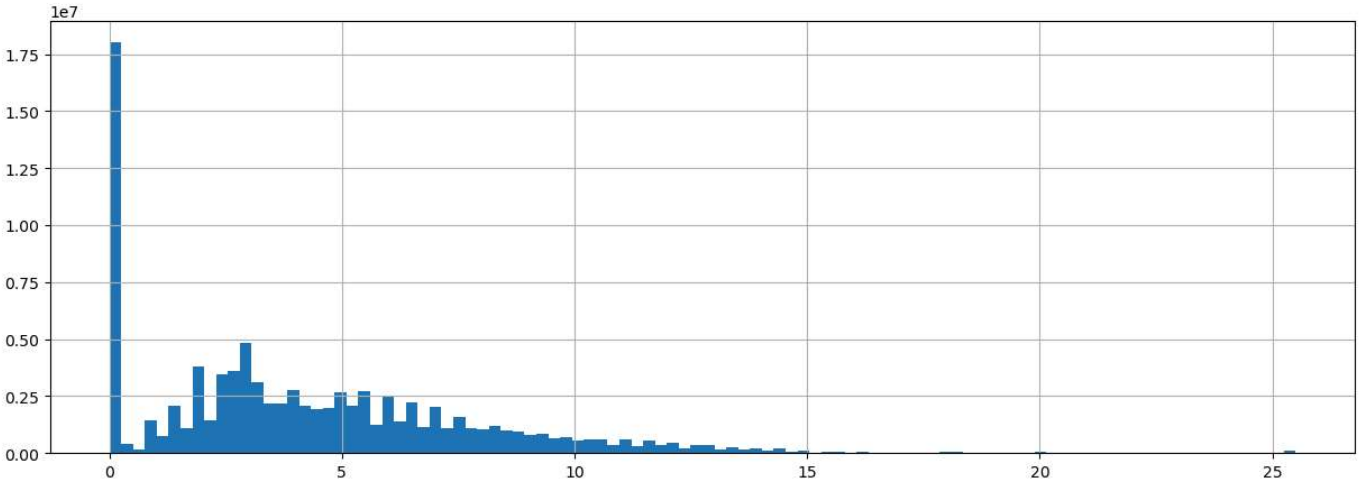


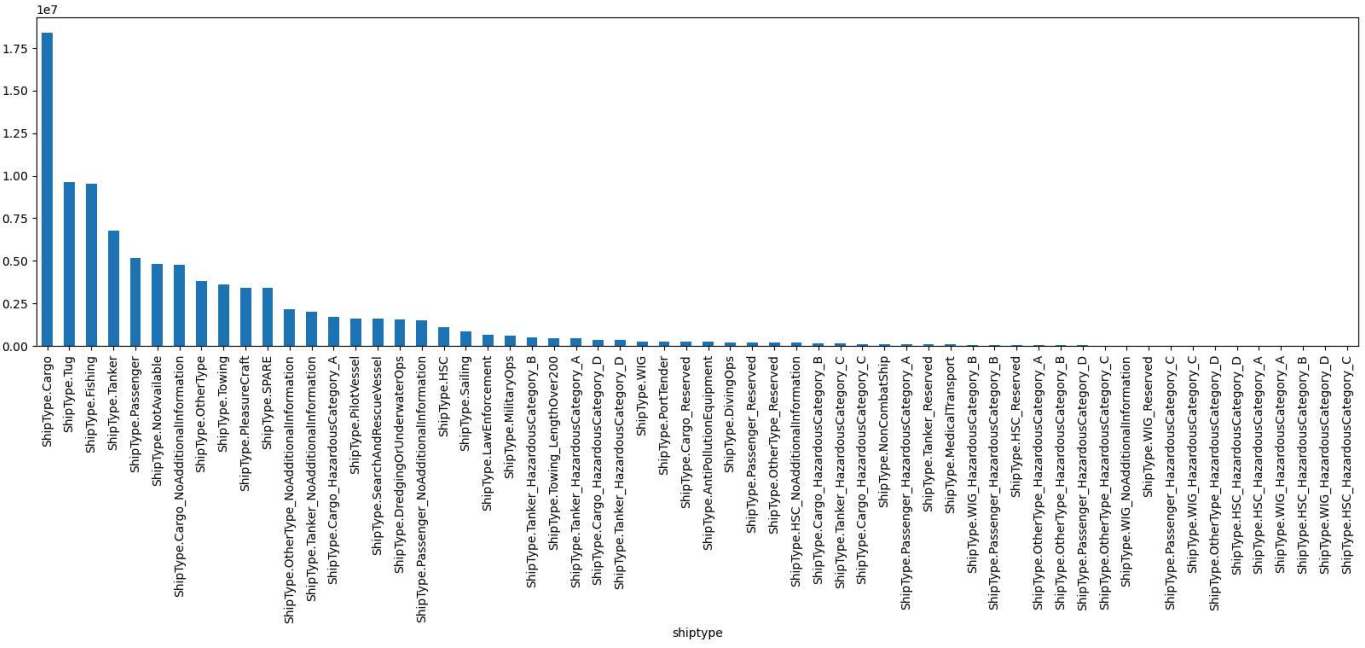
In [10]:
```
mmsi_counts = df_raw_static['MMSI'].value_counts()
plt.figure(figsize=(12, 6))
plt.hist(mmsi_counts.values, bins=range(0, max(mmsi_counts.values) + 200, 200), edgecolor='black')
plt.xlabel('Número de repeticiones (Frecuencia del MMSI)')
plt.ylabel('Cantidad de MMSI')
plt.title('Distribución de repeticiones del MMSI (Histograma con escalón de 200)')
plt.show()
```



Distribución de calados

In [11]:
```
df_raw_static['draught'].hist(bins=100, figsize=(15,5))
plt.show()
df_raw_static['shiptype'].value_counts().plot(kind='bar', figsize=(20,5))
plt.show()
```



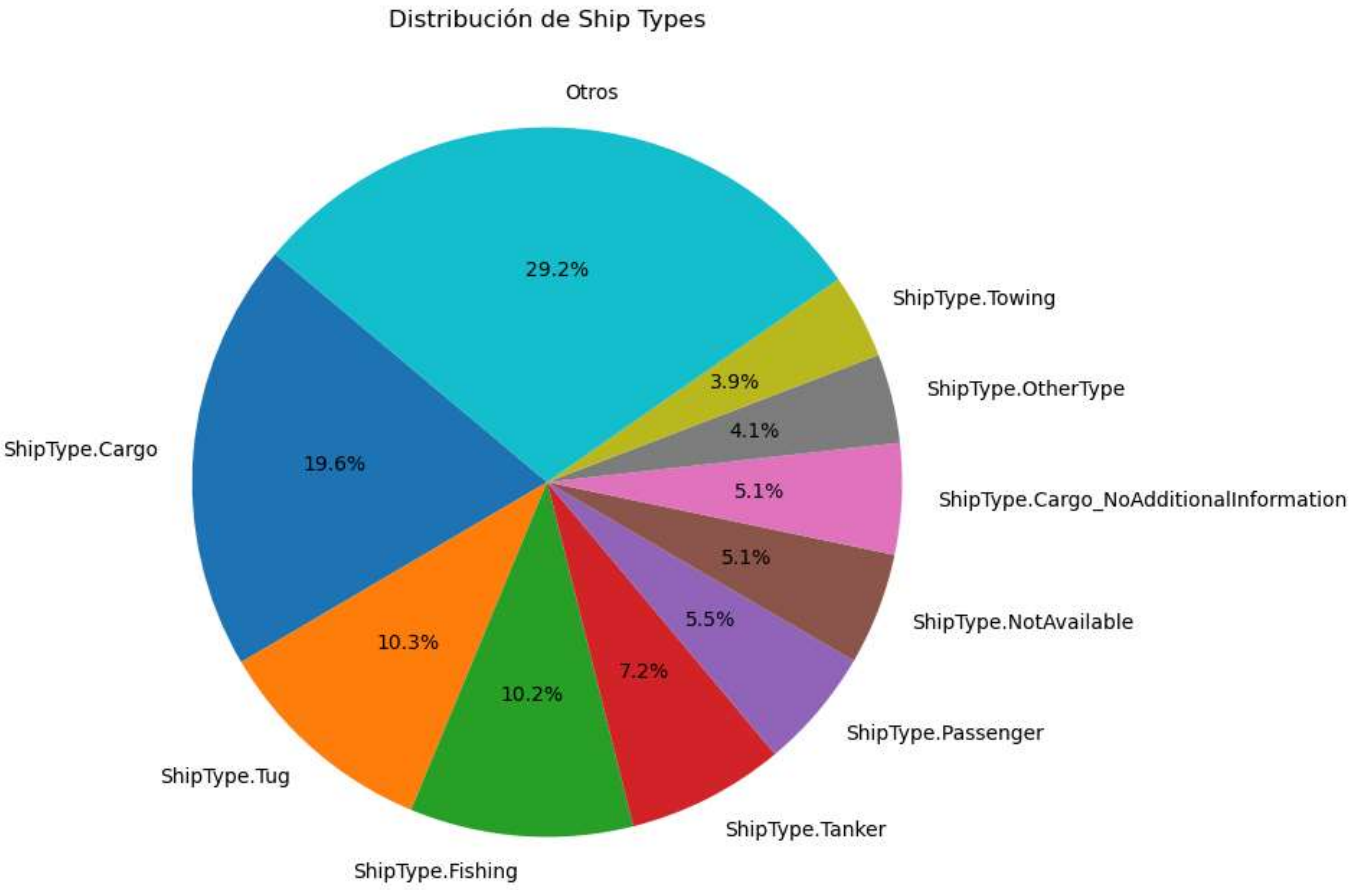


Distribución de Shiptypes

In [12]:
```
# Calcula la distribución de Ship Types y sus porcentajes
ship_type_counts = df_raw_static['shiptype'].value_counts()
ship_type_percentages = (ship_type_counts / len(df_raw_static) * 100).reset_index()

# Renombra las columnas
ship_type_percentages.columns = ['Ship Type', 'Percentage']

# Muestra la tabla directamente con pandas
print(ship_type_percentages.sort_values(by='Percentage', ascending=False).to_string(index=False))
```

```
                                Ship Type  Percentage
                           ShipType.Cargo   19.583932
                             ShipType.Tug   10.262081
                         ShipType.Fishing   10.166100
                          ShipType.Tanker    7.192190
                       ShipType.Passenger    5.504481
                    ShipType.NotAvailable    5.111624
    ShipType.Cargo_NoAdditionalInformation    5.062283
                        ShipType.OtherType    4.062052
                           ShipType.Towing    3.853443
                    ShipType.PleasureCraft    3.641912
                           ShipType.SPARE    3.614028
ShipType.OtherType_NoAdditionalInformation    2.294338
   ShipType.Tanker_NoAdditionalInformation    2.163035
         ShipType.Cargo_HazardousCategory_A    1.825507
                       ShipType.PilotVessel    1.729630
             ShipType.SearchAndRescueVessel    1.704077
           ShipType.DredgingOrUnderwaterOps    1.652333
ShipType.Passenger_NoAdditionalInformation    1.596450
                               ShipType.HSC    1.185486
                           ShipType.Sailing    0.902428
                     ShipType.LawEnforcement    0.694332
                        ShipType.MilitaryOps    0.663279
        ShipType.Tanker_HazardousCategory_B    0.530997
              ShipType.Towing_LengthOver200    0.465419
        ShipType.Tanker_HazardousCategory_A    0.461671
         ShipType.Cargo_HazardousCategory_D    0.397480
        ShipType.Tanker_HazardousCategory_D    0.354696
                               ShipType.WIG    0.295565
                        ShipType.PortTender    0.290491
                    ShipType.Cargo_Reserved    0.276160
             ShipType.AntiPollutionEquipment    0.268554
                          ShipType.DivingOps    0.233824
                ShipType.Passenger_Reserved    0.230591
                ShipType.OtherType_Reserved    0.202404
       ShipType.HSC_NoAdditionalInformation    0.200848
         ShipType.Cargo_HazardousCategory_B    0.179649
        ShipType.Tanker_HazardousCategory_C    0.169166
         ShipType.Cargo_HazardousCategory_C    0.133699
                     ShipType.NonCombatShip    0.112623
    ShipType.Passenger_HazardousCategory_A    0.104387
                   ShipType.Tanker_Reserved    0.091303
                ShipType.MedicalTransport    0.087074
         ShipType.WIG_HazardousCategory_B    0.056106
    ShipType.Passenger_HazardousCategory_B    0.049088
                     ShipType.HSC_Reserved    0.044408
    ShipType.OtherType_HazardousCategory_A    0.038532
    ShipType.OtherType_HazardousCategory_B    0.036569
    ShipType.Passenger_HazardousCategory_D    0.032259
    ShipType.OtherType_HazardousCategory_C    0.028760
      ShipType.WIG_NoAdditionalInformation    0.028743
                     ShipType.WIG_Reserved    0.026818
    ShipType.Passenger_HazardousCategory_C    0.024184
         ShipType.WIG_HazardousCategory_C    0.017185
    ShipType.OtherType_HazardousCategory_D    0.016851
         ShipType.HSC_HazardousCategory_D    0.015686
         ShipType.HSC_HazardousCategory_A    0.012341
         ShipType.WIG_HazardousCategory_A    0.007162
         ShipType.HSC_HazardousCategory_B    0.007003
         ShipType.WIG_HazardousCategory_D    0.005593
         ShipType.HSC_HazardousCategory_C    0.001090
```

In [13]:
```
#Diagrama de sectores
ship_type_counts = df_raw_static['shiptype'].value_counts()

top_ship_types = ship_type_counts.head(9)
top_ship_types['Otros'] = ship_type_counts[9:].sum()

plt.figure(figsize=(8, 8))
plt.pie(top_ship_types, labels=top_ship_types.index, autopct='%1.1f%%', startangle=140)
plt.title('Distribución de Ship Types')
plt.show()
```



Comprobación de infinitos y nulos

In [15]:
```
def verificar_infinitos_nulos(df, nombre_df):
    if df.replace([np.inf, -np.inf], np.nan).isna().any().any():
        print(f"El DataFrame {nombre_df} contiene infinitos y/o valores nulos")
    else:
        print(f"El DataFrame {nombre_df} no contiene ni infinitos ni valores nulos")

# Comprobación de infinitos (y nulos) en los df
verificar_infinitos_nulos(df_raw_static, "df_raw_static")
#verificar_infinitos_nulos(df_clean_static, "df_clean_static")
#verificar_infinitos_nulos(df_processed_static, "df_processed_static")
```

```
El DataFrame df_raw_static no contiene ni infinitos ni valores nulos
```

PREPROCESADO DE DATOS

Aglomeración de subcategorías

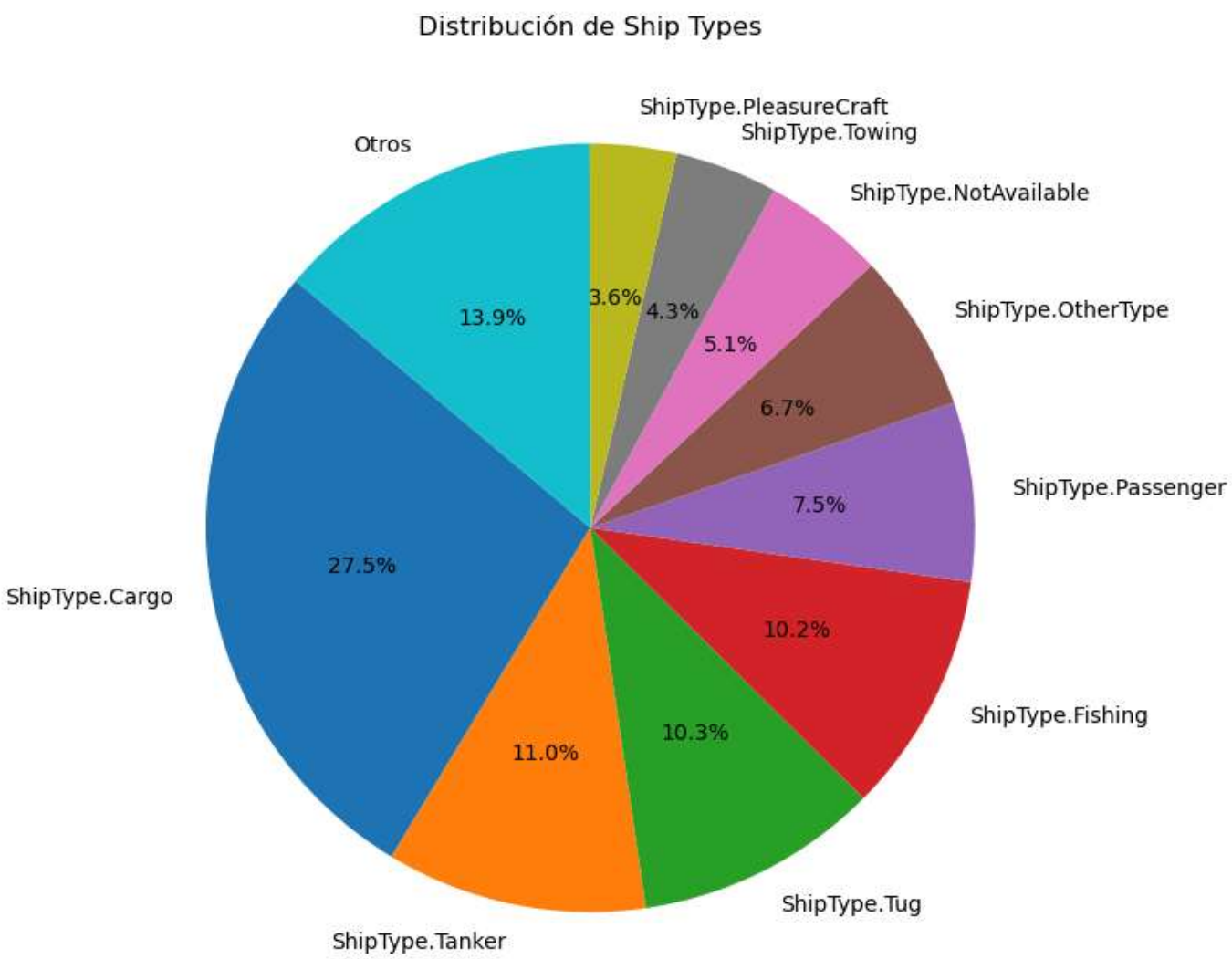
In [6]:
```
# Agrupa las subcategorías de buques en las categorías principales.
# Renombra los campos eliminando lo que viene después del guion bajo
df_reduced_static = df_raw_static.copy()
df_reduced_static['shiptype'] = df_reduced_static['shiptype'].apply(lambda x: re.sub(r'_.*$', '', x))
```

In [7]:
```
# Calcula la distribución de Ship Types y sus porcentajes
ship_type_counts = df_reduced_static['shiptype'].value_counts()
ship_type_percentages = (ship_type_counts / len(df_reduced_static) * 100).reset_index()
```

```
# Renombra las columnas
ship_type_percentages.columns = ['Ship Type', 'Percentage']

# Muestra la tabla directamente con pandas
print(ship_type_percentages.sort_values(by='Percentage', ascending=False).to_string(index=False))
```

```
                      Ship Type  Percentage
                  ShipType.Cargo   27.458709
                 ShipType.Tanker   10.963058
                    ShipType.Tug   10.262081
                ShipType.Fishing   10.166100
              ShipType.Passenger    7.541440
              ShipType.OtherType    6.679505
           ShipType.NotAvailable    5.111624
                 ShipType.Towing    4.318862
          ShipType.PleasureCraft    3.641912
                  ShipType.SPARE    3.614028
           ShipType.PilotVessel    1.729630
  ShipType.SearchAndRescueVessel    1.704077
ShipType.DredgingOrUnderwaterOps    1.652333
                   ShipType.HSC    1.466862
               ShipType.Sailing    0.902428
        ShipType.LawEnforcement    0.694332
           ShipType.MilitaryOps    0.663279
                   ShipType.WIG    0.437173
            ShipType.PortTender    0.290491
 ShipType.AntiPollutionEquipment    0.268554
             ShipType.DivingOps    0.233824
          ShipType.NonCombatShip    0.112623
       ShipType.MedicalTransport    0.087074
```

In [18]:
```
#Diagrama de sectores
ship_type_counts = df_reduced_static['shiptype'].value_counts()

top_ship_types = ship_type_counts.head(9)
top_ship_types['Otros'] = ship_type_counts[9:].sum()

plt.figure(figsize=(8, 8))
plt.pie(top_ship_types, labels=top_ship_types.index, autopct='%1.1f%%', startangle=140)
plt.title('Distribución de Ship Types')
plt.show()
```



Eliminación de filas con shiptype no deseado

In [8]:
```
# Eliminación de las filas con shiptype no deseado
df_reduced_static = df_reduced_static.loc[df_raw_static['shiptype'].isin(['ShipType.Cargo',
                                                                           'ShipType.Tug',
                                                                           'ShipType.Fishing',
                                                                           'ShipType.Tanker',
                                                                           'ShipType.Passenger'])]
```

In [9]:
```
print(df_reduced_static.shape)
print(len(df_reduced_static)/len(df_raw_static))
```

```
(49467377, 9)
0.5270878377563395
```

Eliminación de columnas irrelevantes para el algoritmo

In [10]:
```
df_reduced_static = df_reduced_static.drop(columns=['IMO', 'timestamp'])
```

In [11]:
```
df_reduced_static.head()
```

Out[11]:

	MMSI	to_bow	to_stern	to_port	to_starboard	draught	shiptype
0	352489000	276	57	26	34	21.2	ShipType.Tanker
1	273810100	37	68	17	3	8.8	ShipType.Fishing
2	273448240	29	65	12	3	6.0	ShipType.Fishing
6	412602000	161	29	14	18	7.1	ShipType.Cargo
7	512009984	13	12	6	7	0.0	ShipType.Tug

In [12]:
```
df_reduced_static.shape
```

Out[12]: (49467377, 7)

Filtrado de incoherencias de MMSI

In [13]:
```
df_clean_static = (
    df_reduced_static
    .groupby('MMSI')
    # Filtra valores del mismo grupo (MMSI) con shiptype diferente
    .filter(lambda x: len(set(x['shiptype'])) == 1)
)
```

In [25]:
```
# Almacena las filas filtradas para comprobar que son incoherentes
df_trash = df_reduced_static[~df_reduced_static.index.isin(df_clean_static.index)]
df_trash = df_trash.sort_values(by='MMSI')
df_trash.shape
```

Out[25]: (461172, 7)

In [26]:
```
print(len(df_trash)/len(df_reduced_static))
```

```
0.009322750223849548
```

In [27]:
```
df_trash.head(1)
```

Out[27]:

	MMSI	to_bow	to_stern	to_port	to_starboard	draught	shiptype
49563853	1	44	44	9	9	4.0	ShipType.Passenger

In [28]:
```
df_trash_example = df_trash[df_trash['MMSI'] == 1]
```

In [29]:
```
# Número de filas por cada shiptype
conteo_por_shiptype = df_trash_example['shiptype'].value_counts()

# Gráfico del número de filas por cada shiptype
conteo_por_shiptype.plot(kind='pie', figsize=(5, 5), autopct='%1.1f%%',
                         labels=['']*len(conteo_por_shiptype))
plt.title('Proporción de filas por shiptype para el MMSI 1')
plt.ylabel('')
plt.legend(labels=conteo_por_shiptype.index, loc='upper right')

plt.show()
```

In [14]:
```
df_clean_static = df_clean_static.sort_values(by='MMSI')
```

In [15]:
```
df_clean_static.shape
```

Out[15]: (49006205, 7)

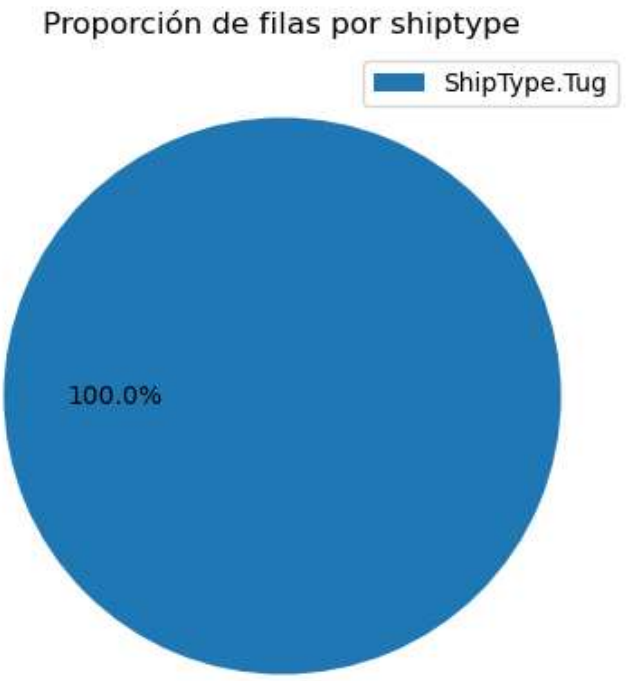
In [32]:
```
df_clean_static.head(1)
```

Out[32]:

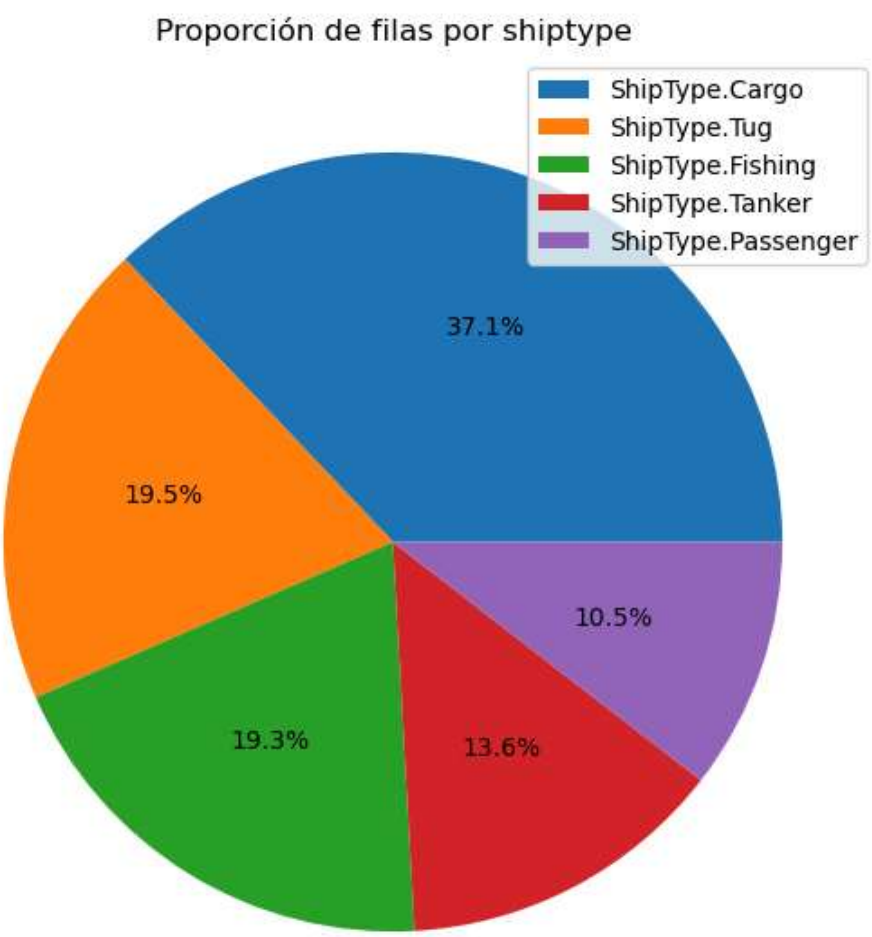
	MMSI	to_bow	to_stern	to_port	to_starboard	draught	shiptype
13563579	5	10	27	5	5	3.4	ShipType.Tug

In [33]:
```
df_clean_example = df_clean_static[df_clean_static['MMSI'] == 5]
```

In [34]:
```
# Número de filas por cada shiptype
conteo_por_shiptype = df_clean_example['shiptype'].value_counts()

# Gráfico del número de filas por cada shiptype
conteo_por_shiptype.plot(kind='pie', figsize=(5, 5), autopct='%1.1f%%',
                         labels=['']*len(conteo_por_shiptype))
plt.title('Proporción de filas por shiptype')
plt.ylabel('')
plt.legend(labels=conteo_por_shiptype.index, loc='upper right')

plt.show()
```



In [35]:
```
# Distribución de shiptype antes de eliminar calado, eslora y manga = 0
# Número de filas por cada shiptype
conteo_por_shiptype = df_clean_static['shiptype'].value_counts()

# Gráfico del número de filas por cada shiptype
conteo_por_shiptype.plot(kind='pie', figsize=(7, 7), autopct='%1.1f%%',
                         labels=['']*len(conteo_por_shiptype))
plt.title('Proporción de filas por shiptype')
plt.ylabel('')
plt.legend(labels=conteo_por_shiptype.index, loc='upper right')

plt.show()
```



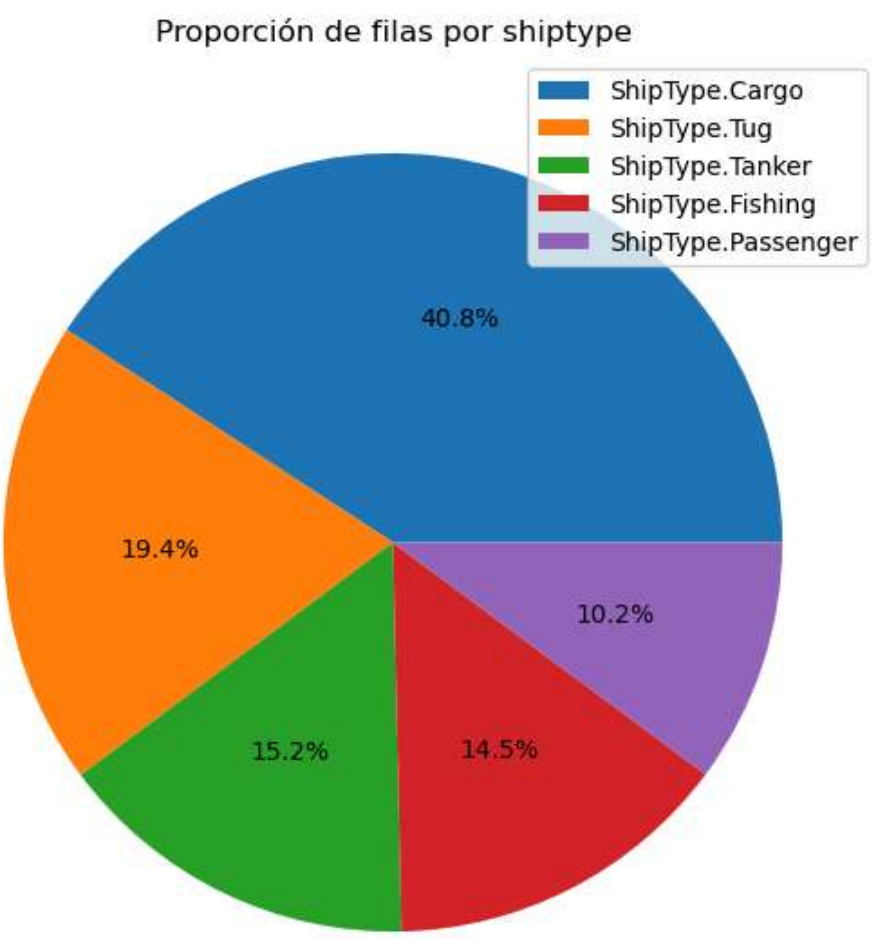
Filtrado de Incoherencias de MMSI (2)

In [16]:
```
# Elimina los valores de calado, eslora y manga = 0
df_clean_static = (
    df_clean_static
    .query('draught != 0\
        and not (to_bow == 0 and to_stern == 0)\
        and not (to_port == 0 and to_starboard == 0)')
)
```

In [17]:
```
df_clean_static.shape
```

Out[17]:
```
(40576316, 7)
```

In [38]:
```
# Distribución de shiptype despues de eliminar calado, eslora y manga = 0
# Número de filas por cada shiptype
conteo_por_shiptype = df_clean_static['shiptype'].value_counts()

# Gráfico del número de filas por cada shiptype
conteo_por_shiptype.plot(kind='pie', figsize=(7, 7), autopct='%1.1f%%',
                         labels=['']*len(conteo_por_shiptype))
plt.title('Proporción de filas por shiptype')
plt.ylabel('')
plt.legend(labels=conteo_por_shiptype.index, loc='upper right')

plt.show()
```



PROCESADO

In [18]:
```
# Aplica la función de procesado al dataframe limpio para crear el dataframe procesado.
df_processed_static = statics_processing(df_clean_static.copy())
```

In [19]:
```
df_processed_static.head()
```

Out[19]:

	MMSI	to_bow	to_stern	to_port	to_starboard	draught	shiptype	len	wid	ldivw	ldivd	wdivd	area
13563579	5	10	27	5	5	3.4	ShipType.Tug	37	10	3.7	10.882353	2.941176	370
13353516	5	10	27	5	5	3.4	ShipType.Tug	37	10	3.7	10.882353	2.941176	370
91661395	5	10	27	5	5	3.4	ShipType.Tug	37	10	3.7	10.882353	2.941176	370
21799361	5	10	27	5	5	3.4	ShipType.Tug	37	10	3.7	10.882353	2.941176	370
10257977	5	10	27	5	5	3.4	ShipType.Tug	37	10	3.7	10.882353	2.941176	370

In [20]:
```
df_processed_static.shape
```

Out[20]:
```
(40576316, 18)
```

In [ ]:
```
df_processed_static['len'].hist(bins=100, figsize=(15,5))
plt.show()
df_processed_static['wid'].hist(bins=100, figsize=(15,5))
plt.show()
```

In [ ]:
```
plt.figure(figsize=(15, 8))
sns.set(style="whitegrid")
sns.violinplot(x='shiptype', y='len', data=df_processed_static, inner='quartile')
plt.title('Distribución de Esloras por Tipo de Buque')
plt.show()
```

In [ ]:
```
plt.figure(figsize=(15, 8))
sns.set(style="whitegrid")
sns.violinplot(x='shiptype', y='wid', data=df_processed_static, inner='quartile')
plt.title('Distribución de Mangas por Tipo de Buque')
plt.show()
```

In [ ]:
```
plt.figure(figsize=(15, 8))
sns.set(style="whitegrid")
sns.violinplot(x='shiptype', y='draught', data=df_processed_static)
plt.title('Distribución de Calados por Tipo de Buque')
plt.show()
```

Eliminación de outliers

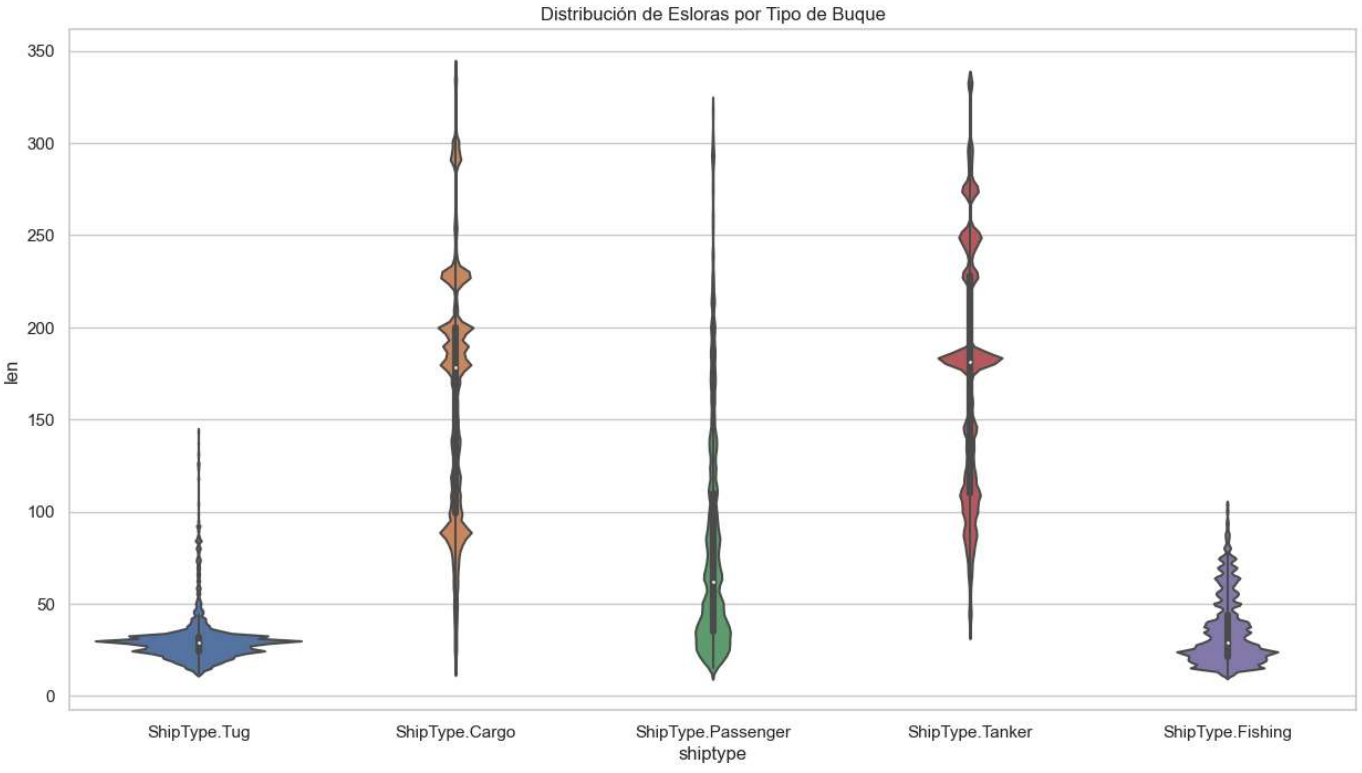
In [21]:
```
atributes = ['len', 'wid', 'draught']

# Calcular cuantiles para los atributos definidos para cada tipo de barco
q_low = df_processed_static.groupby('shiptype')[atributes].transform(lambda x: x.quantile(0.01))
q_high = df_processed_static.groupby('shiptype')[atributes].transform(lambda x: x.quantile(0.99))

# Establecer a NaN los valores por debajo del cuantil del 1% o por encima del cuantil del 99%
df_processed_static[atributes] = df_processed_static[atributes].where(
    (df_processed_static[atributes] >= q_low) & (df_processed_static[atributes] <= q_high)
)
# Eliminar filas con al menos un valor NaN en 'len', 'wid' o 'draught'
df_processed_static = df_processed_static.dropna(subset=atributes, how='any')
```

In [22]:
```
df_processed_static.shape
```

Out[22]:
```
(38947774, 18)
```

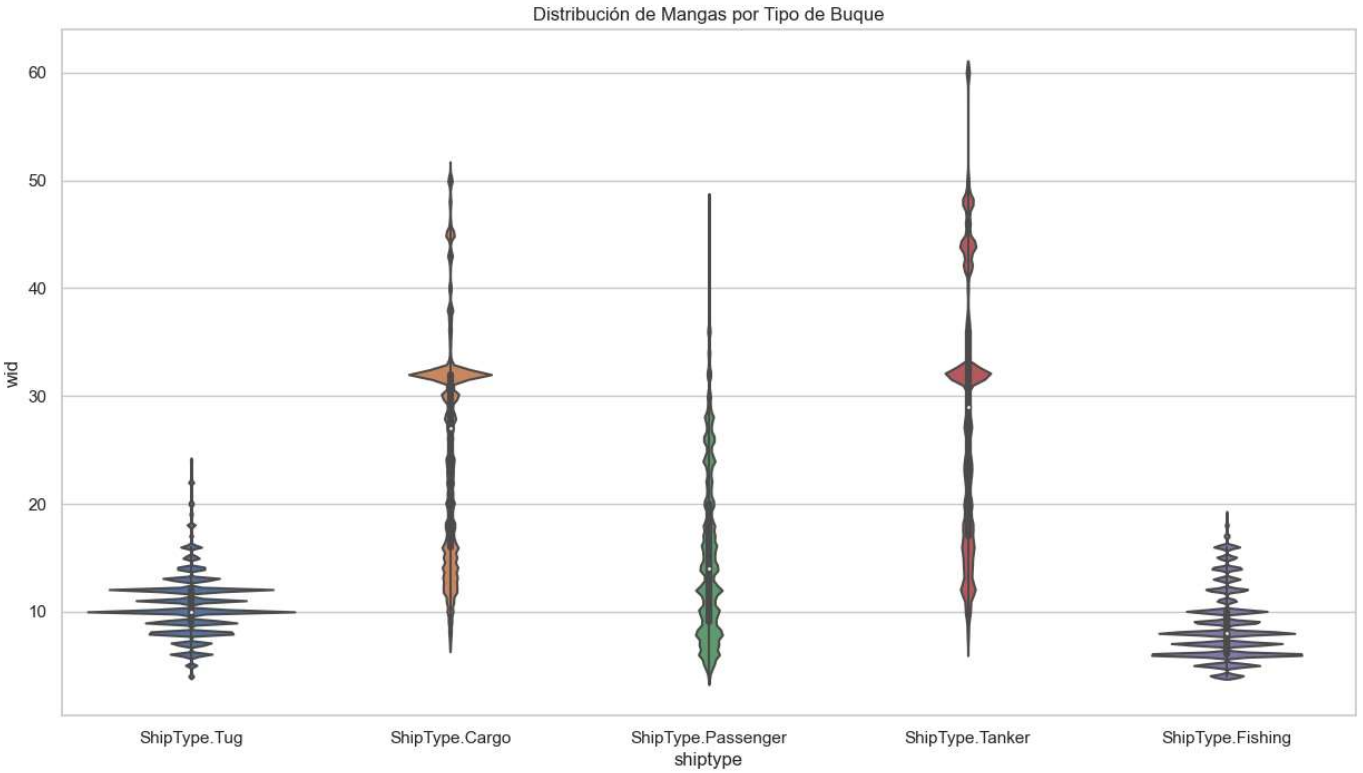
Comprobación de que se han eliminado los outliers

In [42]:
```
plt.figure(figsize=(15, 8))
sns.set(style="whitegrid")
sns.violinplot(x='shiptype', y='len', data=df_processed_static)
plt.title('Distribución de Esloras por Tipo de Buque')
plt.show()
```

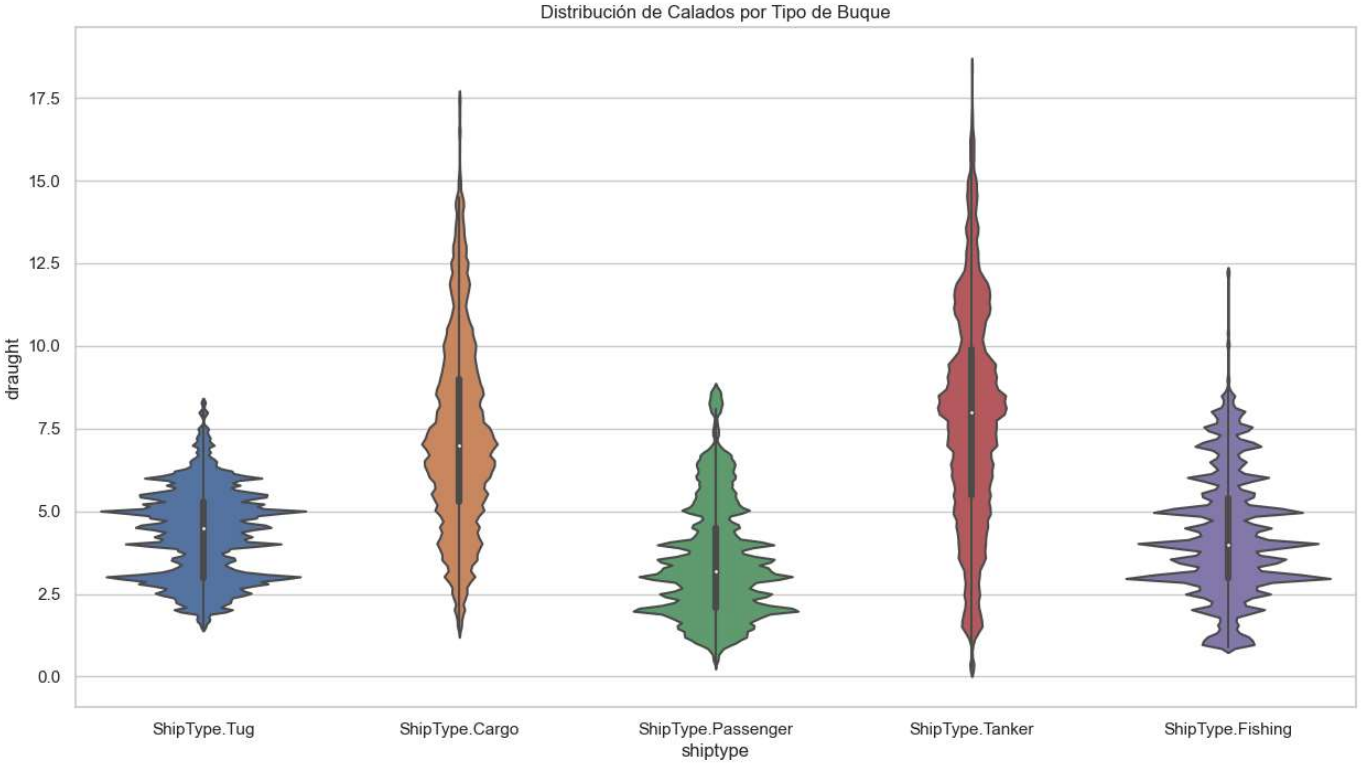


In [43]:
```
plt.figure(figsize=(15, 8))
sns.set(style="whitegrid")
sns.violinplot(x='shiptype', y='wid', data=df_processed_static)
```

```
plt.title('Distribución de Mangas por Tipo de Buque')
plt.show()
```



In [44]:
```
plt.figure(figsize=(15, 8))
sns.set(style="whitegrid")
sns.violinplot(x='shiptype', y='draught', data=df_processed_static)
plt.title('Distribución de Calados por Tipo de Buque')
plt.show()
```



Aproximación por medias de MMSI iguales

In [23]:
```
# Elimina el campo shiptype para calcular las medias por MMSI
df_means = df_processed_static.groupby('MMSI')[['draught', 'to_bow', 'to_stern', 'to_port',
                                                'to_starboard', 'len', 'wid', 'ldivw',
                                                'ldivd', 'wdivd', 'area', 'grith',
                                                'aml', 'amt', 'vs', 'aol']].mean().reset_index()
```

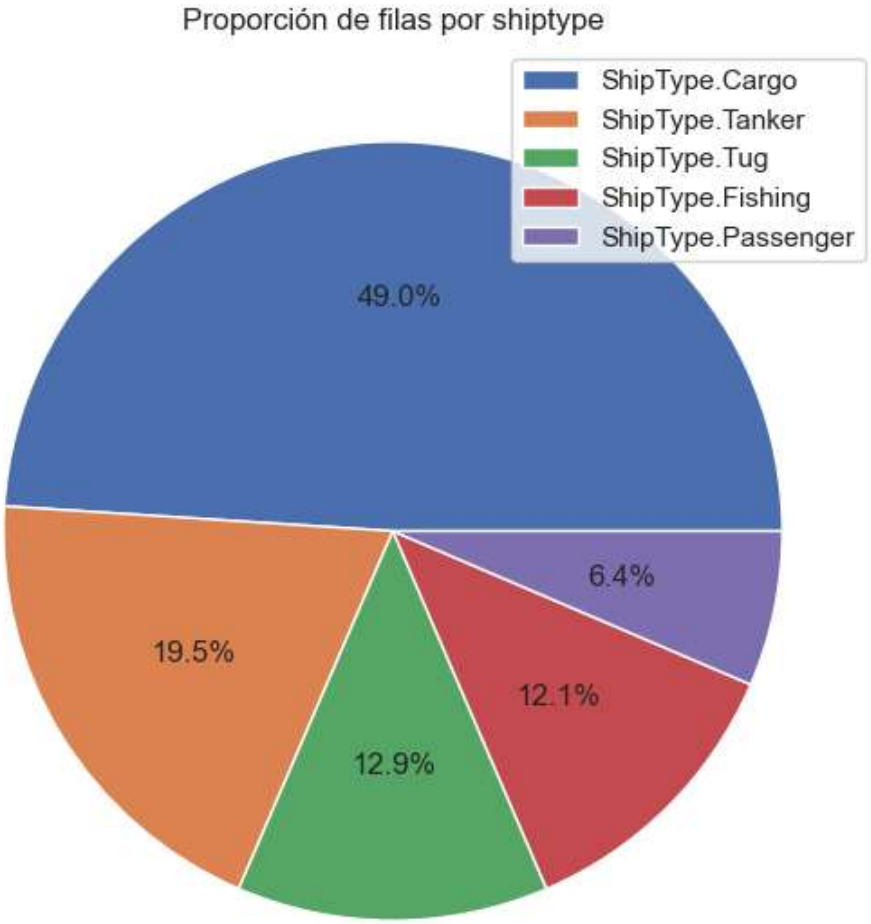
In [24]:
```
# Merge en base a MMSI y eliminamos las filas duplicadas.
df_final_static = pd.merge(df_means, df_processed_static[['MMSI', 'shiptype']], on='MMSI', how='left')
```

```
df_final_static = df_final_static.drop_duplicates(subset='MMSI').reset_index(drop=True)
```

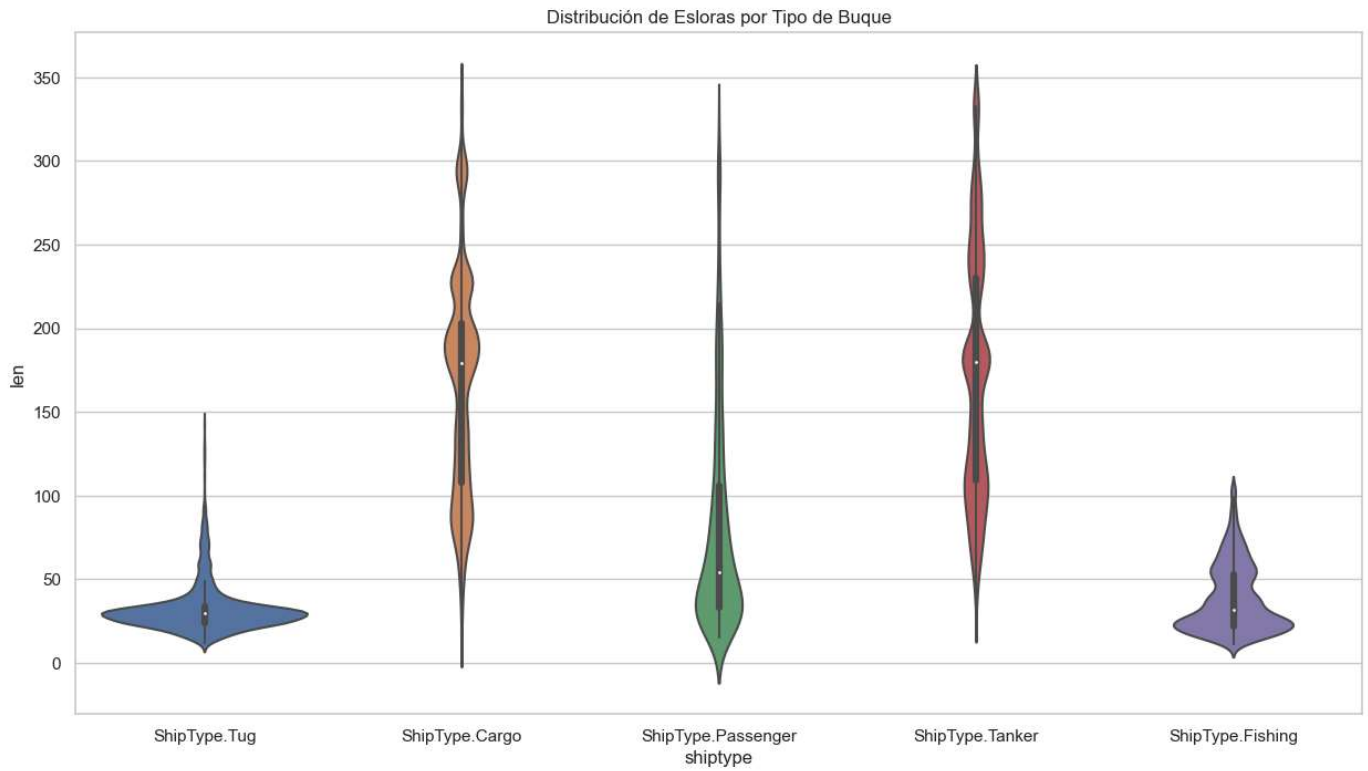
In [25]:
```
df_final_static.shape
```

Out[25]:
```
(40859, 18)
```

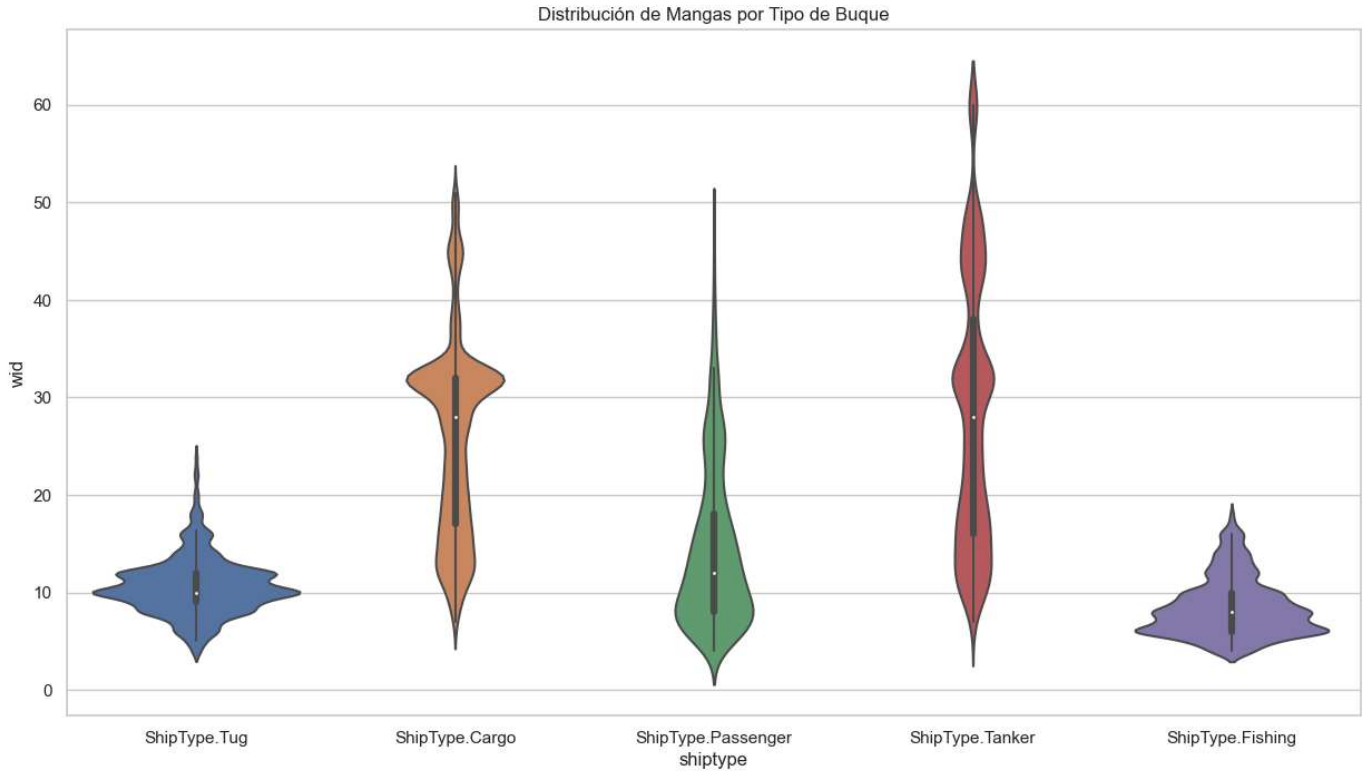
In [60]:
```
# Distribución de shiptype despues de agrupar duplicados
# Número de filas por cada shiptype
conteo_por_shiptype = df_final_static['shiptype'].value_counts()

# Gráfico del número de filas por cada shiptype
conteo_por_shiptype.plot(kind='pie', figsize=(7, 7), autopct='%1.1f%%',
                         labels=['']*len(conteo_por_shiptype))
plt.title('Proporción de filas por shiptype')
plt.ylabel('')
plt.legend(labels=conteo_por_shiptype.index, loc='upper right')

plt.show()
```



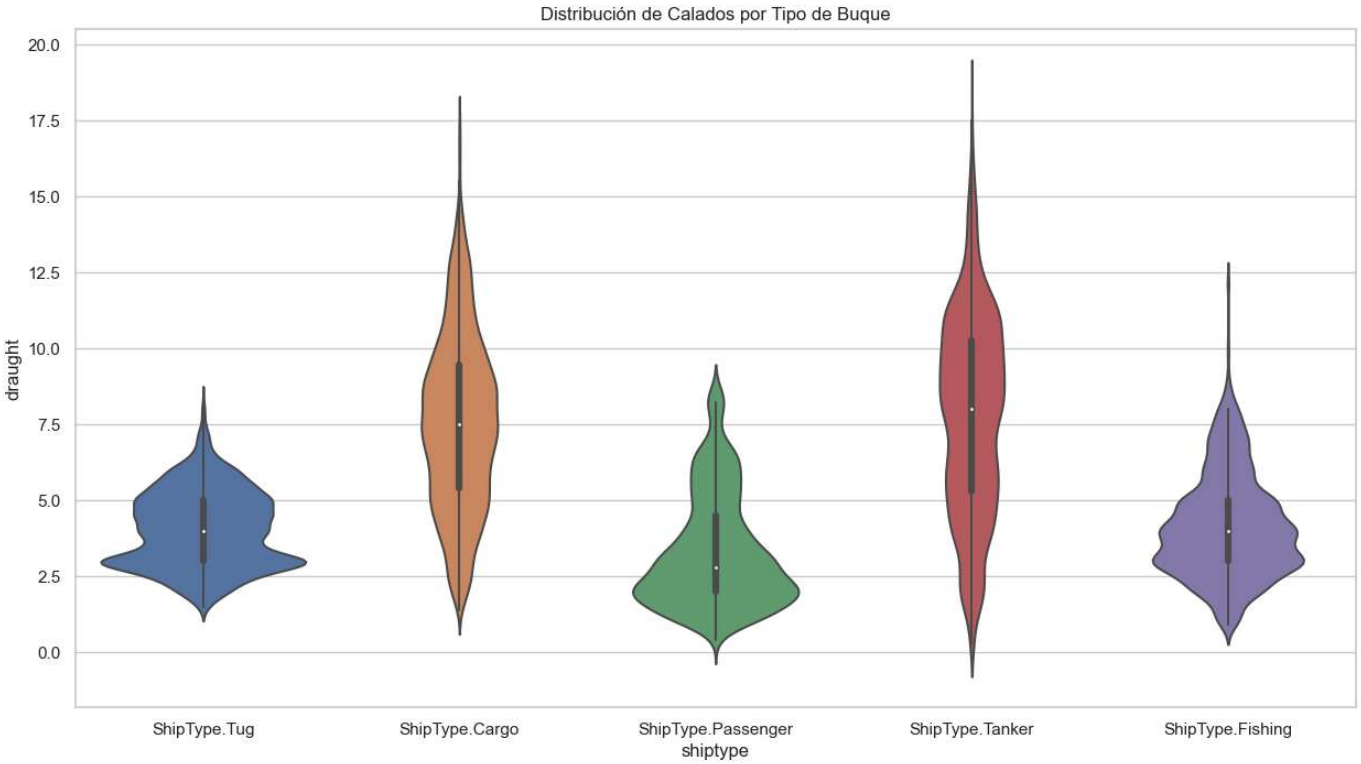
In [61]:
```
plt.figure(figsize=(15, 8))
sns.set(style="whitegrid")
sns.violinplot(x='shiptype', y='len', data=df_final_static)
plt.title('Distribución de Esloras por Tipo de Buque')
plt.show()
```



In [62]:
```
plt.figure(figsize=(15, 8))
sns.set(style="whitegrid")
sns.violinplot(x='shiptype', y='wid', data=df_final_static)
plt.title('Distribución de Mangas por Tipo de Buque')
plt.show()
```



In [63]:
```
plt.figure(figsize=(15, 8))
sns.set(style="whitegrid")
sns.violinplot(x='shiptype', y='draught', data=df_final_static)
plt.title('Distribución de Calados por Tipo de Buque')
plt.show()
```



Guardado en CSV

In [65]:
```
%%time
#guardar df en csv
df_final_static.to_csv('final_static_data_jan.csv', index = False)
#cargar df_clean_static
df_final_static = pd.read_csv('final_static_data_jan.csv')
```

```
CPU times: total: 453 ms
Wall time: 618 ms
```

In [66]:
```
df_final_static.head()
```

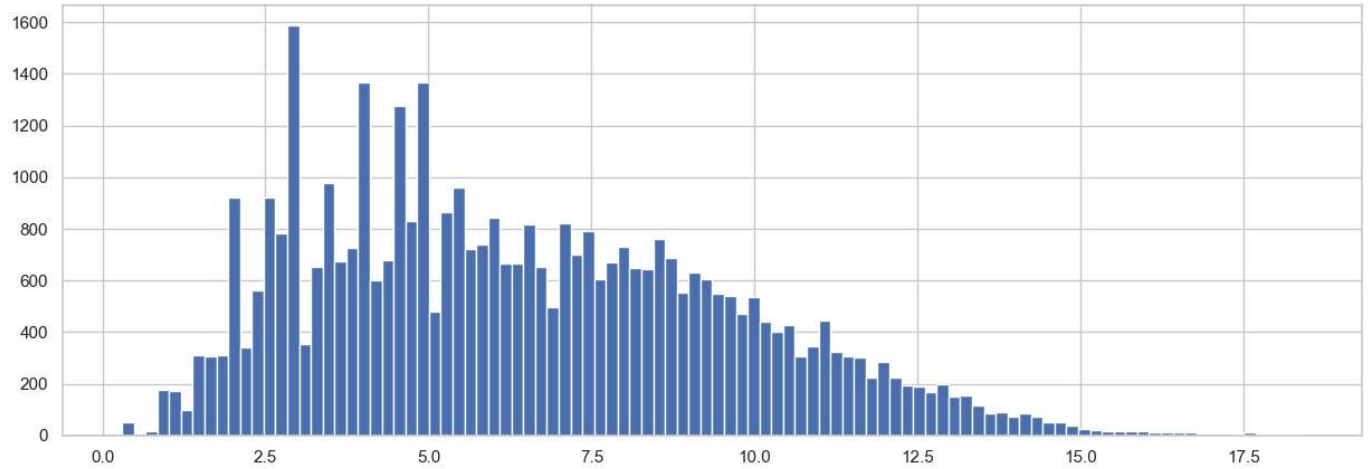
Out[66]:

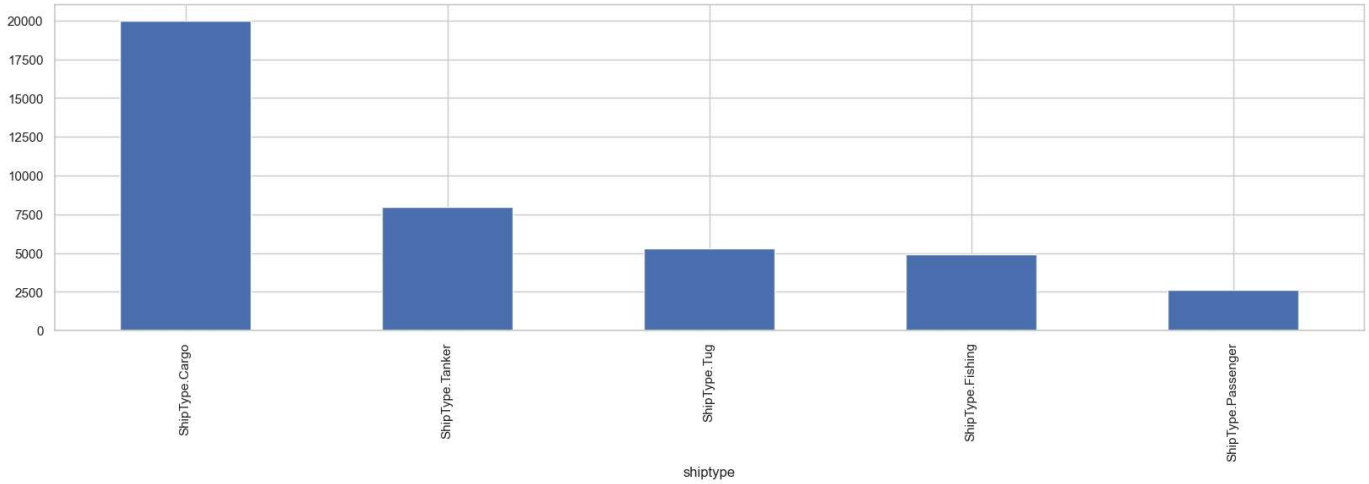
	MMSI	draught	to_bow	to_stern	to_port	to_starboard	len	wid	ldivw	ldivd	wdivd	area	grith	aml	a
0	5	3.4	10.0	27.0	5.0	5.0	37.0	10.0	3.700000	10.882353	2.941176	370.0	47.0	125.8	3
1	6	5.3	7.0	27.0	5.0	5.0	34.0	10.0	3.400000	6.415094	1.886792	340.0	44.0	180.2	5
2	2829	3.4	6.0	90.0	6.0	8.0	96.0	14.0	6.857143	28.235294	4.117647	1344.0	110.0	326.4	4
3	3247	4.5	50.0	10.0	6.0	6.0	60.0	12.0	5.000000	13.333333	2.666667	720.0	72.0	270.0	5
4	8086	3.5	50.0	32.0	12.0	10.0	82.0	22.0	3.727273	23.428571	6.285714	1804.0	104.0	287.0	7

In [67]:
```
df_final_static.shape
```

Out[67]:
```
(40859, 18)
```

In [68]:
```
df_final_static['draught'].hist(bins=100, figsize=(15,5))
plt.show()
df_final_static['shiptype'].value_counts().plot(kind='bar', figsize=(20,5))
plt.show()
```





ENTRENAMIENTO DEL MODELO Y RESULTADOS

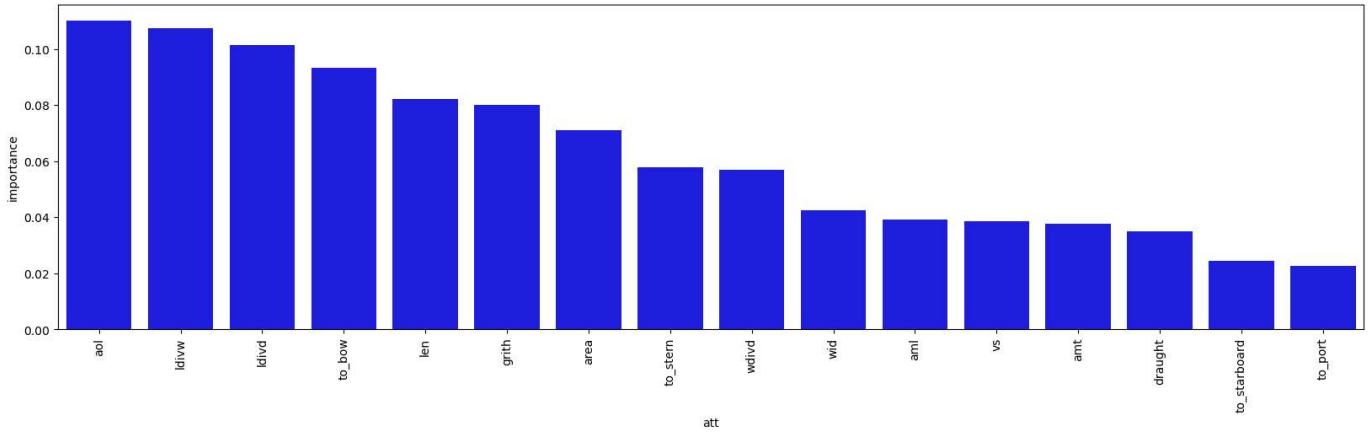
Los resultados proceden del csv final_static_data (Experimento 1)

In [32]:
```
#División entrenamiento y test
df_train, df_test = train_test_split(df_final_static, random_state = 43)
x_train = df_train.drop(columns=['shiptype'])
y_train = df_train.shiptype
x_test = df_test.drop(columns=['shiptype'])
y_test = df_test.shiptype

#Creacion del modelo
pipe = Pipeline([('scaler', MinMaxScaler()),
                 ('classifier',RandomForestClassifier(n_estimators=100))])
model = pipe.fit(x_train, y_train)
y_pred = model.predict(x_test)

#Resultados
print(classification_report(y_test, y_pred, zero_division=0))
forest = model['classifier']
importances = forest.feature_importances_
std = np.std([tree.feature_importances_ for tree in forest.estimators_], axis=0)
fi = pd.Series(importances, index=x_train.columns).reset_index()
fi.columns = ['att', 'importance']
fi = fi.sort_values(by='importance', ascending=False).reset_index(drop=True)
g = sns.barplot(data=fi, x='att', y='importance', color='b')
g.figure.set_size_inches(20, 5)
g.set_xticklabels(g.get_xticklabels(), rotation=90)
```

```
                    precision    recall  f1-score   support

     ShipType.Cargo       0.89      0.92      0.90      4997
   ShipType.Fishing       0.80      0.83      0.81      1275
ShipType.Passenger       0.75      0.72      0.73       698
    ShipType.Tanker       0.84      0.77      0.80      1918
       ShipType.Tug       0.83      0.84      0.84      1327

          accuracy                           0.85     10215
         macro avg       0.82      0.81      0.82     10215
      weighted avg       0.85      0.85      0.85     10215
```

Out[32]:

```
[Text(0, 0, 'aol'),
 Text(1, 0, 'ldivw'),
 Text(2, 0, 'ldivd'),
 Text(3, 0, 'to_bow'),
 Text(4, 0, 'len'),
 Text(5, 0, 'grith'),
 Text(6, 0, 'area'),
 Text(7, 0, 'to_stern'),
 Text(8, 0, 'wdivd'),
 Text(9, 0, 'wid'),
 Text(10, 0, 'aml'),
 Text(11, 0, 'vs'),
 Text(12, 0, 'amt'),
 Text(13, 0, 'draught'),
 Text(14, 0, 'to_starboard'),
 Text(15, 0, 'to_port')]
```



PREPARACIÓN DE DINÁMICOS (Prep_Dynamics.ipynb)

LIBRERÍAS

In [1]:
```
import csv
# Para leer y escribir archivos CSV
import os
# Para manipular archivos y directorios
import pandas as pd
# Para manipulación y análisis de datos
import geopandas as gpd
# Para trabajar con datos geoespaciales
import movingpandas as mpd
# Para el análisis de datos de movimiento
import numpy as np
# Para computación científica y operaciones matemáticas
from tqdm import tqdm
# Para mostrar barras de progreso en bucles iterativos
import seaborn as sns
# Para visualización de datos estadísticos
import matplotlib.pyplot as plt
# Para visualización de datos en gráficos
from sklearn.model_selection import train_test_split, GridSearchCV, GroupShuffleSplit
# Para dividir conjuntos de datos y búsqueda de hiperparámetros
from sklearn.preprocessing import MinMaxScaler
# Para escalar características
from sklearn.metrics import classification_report
# Para evaluar el rendimiento del modelo
from sklearn.ensemble import RandomForestClassifier
# Para entrenar modelos de clasificación de bosques aleatorios
from sklearn.pipeline import Pipeline
# Para encadenar pasos de procesamiento y modelado
import hvplot.pandas
# Para visualización interactiva de pandas DataFrames
from shapely.geometry import Point
# Para trabajar con geometría en Python
from datetime import timedelta
# Para manipular fechas y tiempos
import geoviews as gv
# Para visualización de datos geoespaciales
from ydata_profiling import ProfileReport
# Para generar informes de perfil de datos
import holoviews as hv
# Para visualización interactiva de datos
from geopy.distance import great_circle, geodesic
# Para calcular distancias geoespaciales
from shapely.geometry import MultiPoint, LineString, Polygon
# Para trabajar con geometría en Python
from matplotlib.colors import LinearSegmentedColormap
# Para personalizar mapas de colores en Matplotlib
import xarray as xr
# Para trabajar con estructuras de datos multidimensionales
import cartopy.crs as ccrs
# Para visualización de mapas utilizando Cartopy
import plotly.express as px
# Para visualización interactiva de datos
import time
# Para medir el tiempo de ejecución de código
```

LECTURA ARCHIVOS DINÁMICOS

ARCHIVOS DINÁMICOS POR MMSI

UN SOLO MMSI POR ITERACIÓN (PARA ESTUDIO DE TRAYECTORIAS INDIVIDUALES)

In [2]:
```
# Obtiene la lista de archivos CSV en el directorio
lista_archivos_dynamic = (
    [archivo for archivo in os.listdir('Dynamic_by_MMSI') if archivo.endswith('.csv')])
# Lee el primer archivo CSV y lo almacena en un DataFrame
df_raw_dynamic = pd.read_csv(os.path.join('Dynamic_by_MMSI', lista_archivos_dynamic[1]))
```

In [ ]:
```
print(df_raw_dynamic)
```

ASOCIACIÓN DE MMSI Y SHIPTYPE

CREACIÓN DE DF CON VALORES DE MMSI Y SHIPTYPE

In [ ]:
```
df_MMSI_shiptype = (
    pd.read_csv('final_static_data.csv')
    .filter(items=['MMSI', 'shiptype'])
    .sort_values(by='MMSI')
    .reset_index(drop=True)
)
df_MMSI_shiptype.to_csv('MMSI_shiptype_data.csv', index = False)
```

In [3]:
```
df_MMSI_shiptype = pd.read_csv('MMSI_shiptype_data.csv')
```

In [5]:
```
print (df_MMSI_shiptype)
```

```
            MMSI            shiptype
0              5        ShipType.Tug
1              6        ShipType.Tug
2           2829      ShipType.Cargo
3           3247  ShipType.Passenger
4           8086  ShipType.Passenger
...          ...                 ...
40854  827002226        ShipType.Tug
40855  848290500      ShipType.Cargo
40856  900004001      ShipType.Cargo
40857  924467387    ShipType.Fishing
40858  985524411        ShipType.Tug

[40859 rows x 2 columns]
```

PREPARACIÓN DEL DF

In [4]:
```
def clean(df):
    # Eliminar columna 'MMSI'
    df.drop(columns=['mmsi'], inplace=True)
    # Ordenar por timestamp
    df.sort_values(by='timestamp', inplace=True)
    # Convertir timestamp a datetime
    df['timestamp'] = pd.to_datetime(df['timestamp'], unit='s')
    # Reemplazar valores de latitud fuera de rango con NaN
    df['lat'] = df['lat'].where((df['lat'] >= -90) & (df['lat'] <= 90), np.nan)
    # Reemplazar valores de longitud fuera de rango con NaN
    df['lon'] = df['lon'].where((df['lon'] >= -180) & (df['lon'] <= 180), np.nan)
    # Hacer copias de lat y lon
    df = df.assign(lat_copy=df['lat'], lon_copy=df['lon'])
    return df
```

CREACIÓN DF MERGED

In [6]:
```
# Realiza el merge basado en las columnas 'mmsi' y 'MMSI'
df_merged_dynamic = pd.merge(df_raw_dynamic, df_MMSI_shiptype, left_on='mmsi', right_on='MMSI',
                             how='inner')
# Elimina la columna 'MMSI' duplicada
df_merged_dynamic.drop(columns=['MMSI'], inplace=True)
# Elimina filas con valores NaN en la columna 'shiptype'
df_merged_dynamic.dropna(subset=['shiptype'], inplace=True)
#Ordena por timestamp
df_merged_dynamic = df_merged_dynamic.sort_values(by='timestamp')

print (df_merged_dynamic)
```

```
       timestamp       mmsi       lat         lon  speed  course  heading  \
0     1672527471  100003661  8.578080  124.758660    0.0    37.9      511
1     1672527857  100003661  8.578085  124.758670    0.0    37.9      511
2     1672528158  100003661  8.578097  124.758668    0.0    37.9      511
3     1672528460  100003661  8.578082  124.758690    0.0    37.9      511
4     1672528770  100003661  8.578080  124.758643    0.0    37.9      511
...          ...        ...       ...         ...    ...     ...      ...
4441  1675203943  100003661  8.578278  124.758882    0.1   242.3      511
4442  1675204245  100003661  8.578275  124.758893    0.1   242.3      511
4443  1675204845  100003661  8.578263  124.758885    0.1   242.3      511
4444  1675205164  100003661  8.578258  124.758887    0.2   242.3      511
4445  1675205484  100003661  8.578275  124.758882    0.1   242.3      511

          shiptype
0     ShipType.Tug
1     ShipType.Tug
2     ShipType.Tug
3     ShipType.Tug
4     ShipType.Tug
...            ...
4441  ShipType.Tug
4442  ShipType.Tug
4443  ShipType.Tug
4444  ShipType.Tug
4445  ShipType.Tug

[4446 rows x 8 columns]
```

ANÁLISIS DATOS DINÁMICOS

In [7]:
```
print(df_merged_dynamic.isna().any().any())
df_merged_dynamic.dtypes
```

Out[7]:
```
False
timestamp      int64
mmsi           int64
lat          float64
lon          float64
speed        float64
course       float64
heading        int64
shiptype      object
dtype: object
```

In [ ]:
```
# Inicializa un DataFrame para almacenar las longitudes según el shiptype
data = {'shiptype': [], 'longitud': []}

for MMSI, row in df_MMSI_shiptype.iterrows():
    # Busca el archivo CSV correspondiente al MMSI actual
    csv_file = os.path.join('Dynamic_by_MMSI', f"dynamic_{row['MMSI']}.csv")

    # Verifica si el archivo CSV existe
    if os.path.exists(csv_file):
        df_raw_dynamic = pd.read_csv(csv_file)

        # Calcula la longitud del DataFrame actual y añádela al DataFrame de datos
        shiptype = row['shiptype']
        longitud_df = len(df_raw_dynamic)
        data['shiptype'].append(shiptype)
        data['longitud'].append(longitud_df)

# Crea un DataFrame a partir de los datos recopilados
df_longitudes = pd.DataFrame(data)
```

In [ ]:
```
# Crea el violinplot para mostrar las distribuciones de longitudes según el shiptype,
# usando px (interactivo)
fig = px.violin(df_longitudes, y='longitud', x='shiptype', box=True, points="all")
fig.update_layout(title='Distribución de longitudes según el shiptype',
                  xaxis_title='Shiptype',
                  yaxis_title='Longitud del DataFrame')
fig.show()
```

In [ ]:
```
# Crea el violinplot para mostrar las distribuciones de longitudes según el shiptype
plt.figure(figsize=(10, 6))
sns.violinplot(data=df_longitudes, x='shiptype', y='longitud')
plt.title('Distribución de longitudes según el shiptype')
plt.xlabel('Shiptype')
plt.ylabel('Longitud del DataFrame')

# Marca el cuartil 1 y el primer 10% de mensajes para cada shiptype
for i, shiptype in enumerate(df_longitudes['shiptype'].unique()):
    df_shiptype = df_longitudes[df_longitudes['shiptype'] == shiptype]
    q1 = df_shiptype['longitud'].quantile(0.25)
    plt.text(i, q1, f'Q1: {q1:.2f}', color='black', ha='right', va='bottom')
    p10 = df_shiptype['longitud'].quantile(0.1)
    plt.text(i, p10, f'P10: {p10:.2f}', color='black', ha='right', va='bottom')
```

```
plt.show()
```

CREACIÓN DE TRAYECTORIAS

Directamente con MPD

In [9]:
```
%%time
# Cambia el timestamp a formato fecha y hora (más espacio pero más legible)
df_merged_dynamic['timestamp'] = pd.to_datetime(df_merged_dynamic['timestamp'], unit='s')
df_merged_dynamic = df_merged_dynamic.assign(lat_copy=df_merged_dynamic['lat'],
                                             lon_copy=df_merged_dynamic['lon'])

print(df_merged_dynamic)
df_merged_dynamic['timestamp'].min(), df_merged_dynamic['timestamp'].max()
```

```
               timestamp       mmsi       lat         lon  speed  course  \
0    2022-12-31 22:57:51  100003661  8.578080  124.758660    0.0    37.9
1    2022-12-31 23:04:17  100003661  8.578085  124.758670    0.0    37.9
2    2022-12-31 23:09:18  100003661  8.578097  124.758668    0.0    37.9
3    2022-12-31 23:14:20  100003661  8.578082  124.758690    0.0    37.9
4    2022-12-31 23:19:30  100003661  8.578080  124.758643    0.0    37.9
...                  ...        ...       ...         ...    ...     ...
4441 2023-01-31 22:25:43  100003661  8.578278  124.758882    0.1   242.3
4442 2023-01-31 22:30:45  100003661  8.578275  124.758893    0.1   242.3
4443 2023-01-31 22:40:45  100003661  8.578263  124.758885    0.1   242.3
4444 2023-01-31 22:46:04  100003661  8.578258  124.758887    0.2   242.3
4445 2023-01-31 22:51:24  100003661  8.578275  124.758882    0.1   242.3

      heading      shiptype  lat_copy    lon_copy
0         511  ShipType.Tug  8.578080  124.758660
1         511  ShipType.Tug  8.578085  124.758670
2         511  ShipType.Tug  8.578097  124.758668
3         511  ShipType.Tug  8.578082  124.758690
4         511  ShipType.Tug  8.578080  124.758643
...       ...           ...       ...         ...
4441      511  ShipType.Tug  8.578278  124.758882
4442      511  ShipType.Tug  8.578275  124.758893
4443      511  ShipType.Tug  8.578263  124.758885
4444      511  ShipType.Tug  8.578258  124.758887
4445      511  ShipType.Tug  8.578275  124.758882

[4446 rows x 10 columns]
CPU times: total: 0 ns
Wall time: 20.1 ms
```

Out[9]:
```
(Timestamp('2022-12-31 22:57:51'), Timestamp('2023-01-31 22:51:24'))
```

In [10]:
```
%%time
# Genera las trayectorias sin pasar por un gdf, le pasamos los parámetros directamente
month_trajectories = mpd.Trajectory(df_merged_dynamic,'Remolcador 1',
                                    t='timestamp', x='lon_copy', y='lat_copy', crs=4326)
print(month_trajectories)
month_trajectories.df
```

```
Trajectory Remolcador 1 (2022-12-31 22:57:51 to 2023-01-31 22:51:24) | Size: 4446 | Length: 725152.3m
Bounds: (123.6318, 8.575555, 124.759902, 9.617302)
LINESTRING (124.75866 8.57808, 124.75867 8.578085, 124.758668 8.578097, 124.75869 8.578082, 124.7586
CPU times: total: 297 ms
Wall time: 350 ms
```

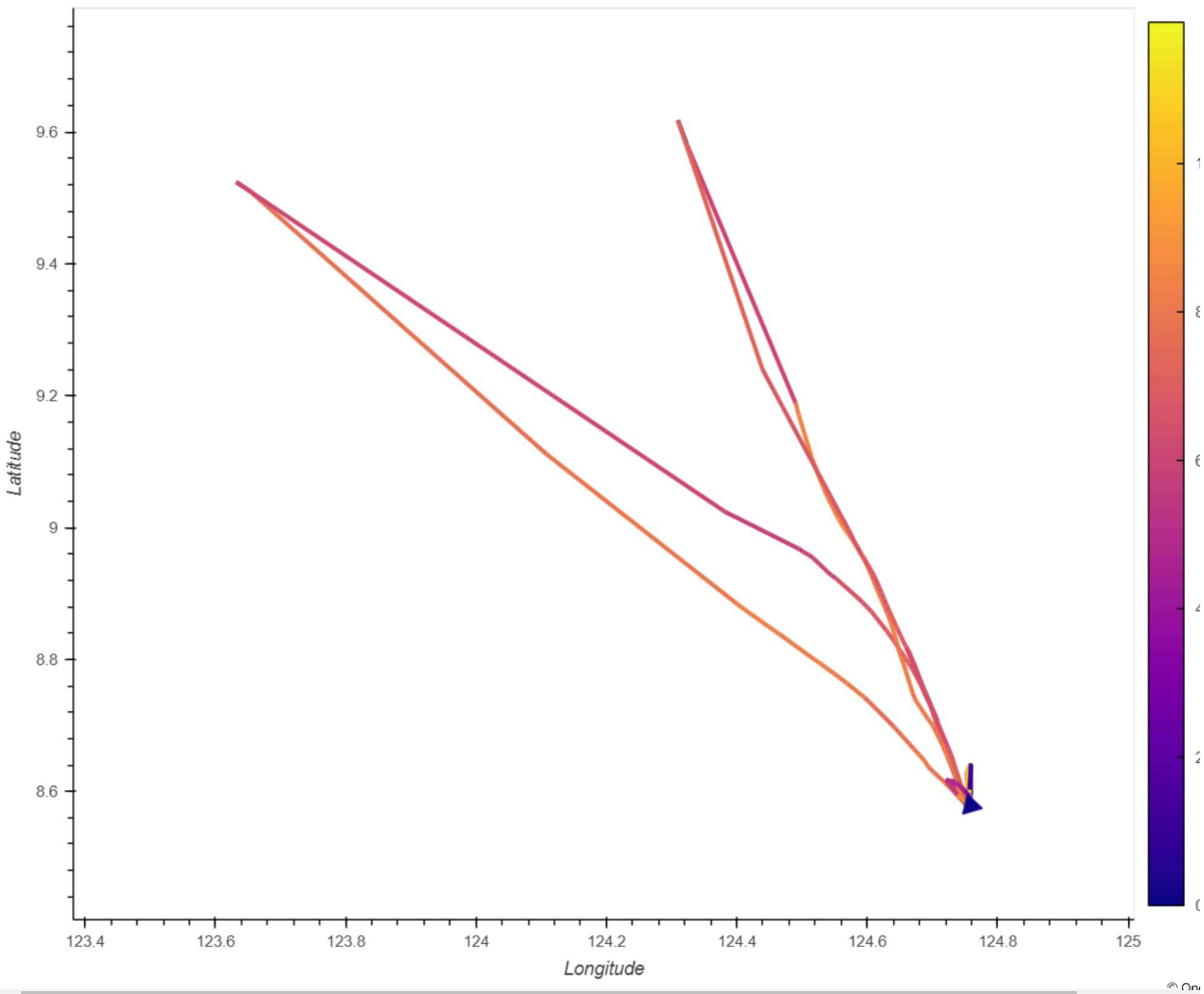
Out[10]:

timestamp	mmsi	lat	lon	speed	course	heading	shiptype	geometry	traj_id
2022-12-31 22:57:51	100003661	8.578080	124.758660	0.0	37.9	511	ShipType.Tug	POINT (124.75866 8.57808)	Remolcador 1
2022-12-31 23:04:17	100003661	8.578085	124.758670	0.0	37.9	511	ShipType.Tug	POINT (124.75867 8.57808)	Remolcador 1
2022-12-31 23:09:18	100003661	8.578097	124.758668	0.0	37.9	511	ShipType.Tug	POINT (124.75867 8.57810)	Remolcador 1
2022-12-31 23:14:20	100003661	8.578082	124.758690	0.0	37.9	511	ShipType.Tug	POINT (124.75869 8.57808)	Remolcador 1
2022-12-31 23:19:30	100003661	8.578080	124.758643	0.0	37.9	511	ShipType.Tug	POINT (124.75864 8.57808)	Remolcador 1
...	...	...	...	...	...	...	...	...	...
2023-01-31 22:25:43	100003661	8.578278	124.758882	0.1	242.3	511	ShipType.Tug	POINT (124.75888 8.57828)	Remolcador 1
2023-01-31 22:30:45	100003661	8.578275	124.758893	0.1	242.3	511	ShipType.Tug	POINT (124.75889 8.57827)	Remolcador 1
2023-01-31 22:40:45	100003661	8.578263	124.758885	0.1	242.3	511	ShipType.Tug	POINT (124.75889 8.57826)	Remolcador 1
2023-01-31 22:46:04	100003661	8.578258	124.758887	0.2	242.3	511	ShipType.Tug	POINT (124.75889 8.57826)	Remolcador 1
2023-01-31 22:51:24	100003661	8.578275	124.758882	0.1	242.3	511	ShipType.Tug	POINT (124.75888 8.57827)	Remolcador 1

4446 rows × 9 columns

In [11]:
```
month_trajectories.hvplot(c='speed', colorbar=True, tiles=True)
```

Out[11]:



DISCRETIZACIÓN DE TRAYECTORIAS

DETECCION DE PARADAS CON MPD

In [ ]:
```
# Umbral de 50 m durante 1 hora (puntos de parada)
mpd.TrajectoryStopDetector(month_trajectories).get_stop_points\
```

```
(min_duration=timedelta(seconds=3600), max_diameter=50)\
.hvplot(geo=True, tiles=True, width=700, height=1000)
```

In [13]:
```
# Umbral de 50 m durante 1 hora (segmentos de parada)
month_stops = mpd.TrajectoryStopDetector(month_trajectories).get_stop_segments\
(min_duration=timedelta(seconds=3600), max_diameter=50)
```

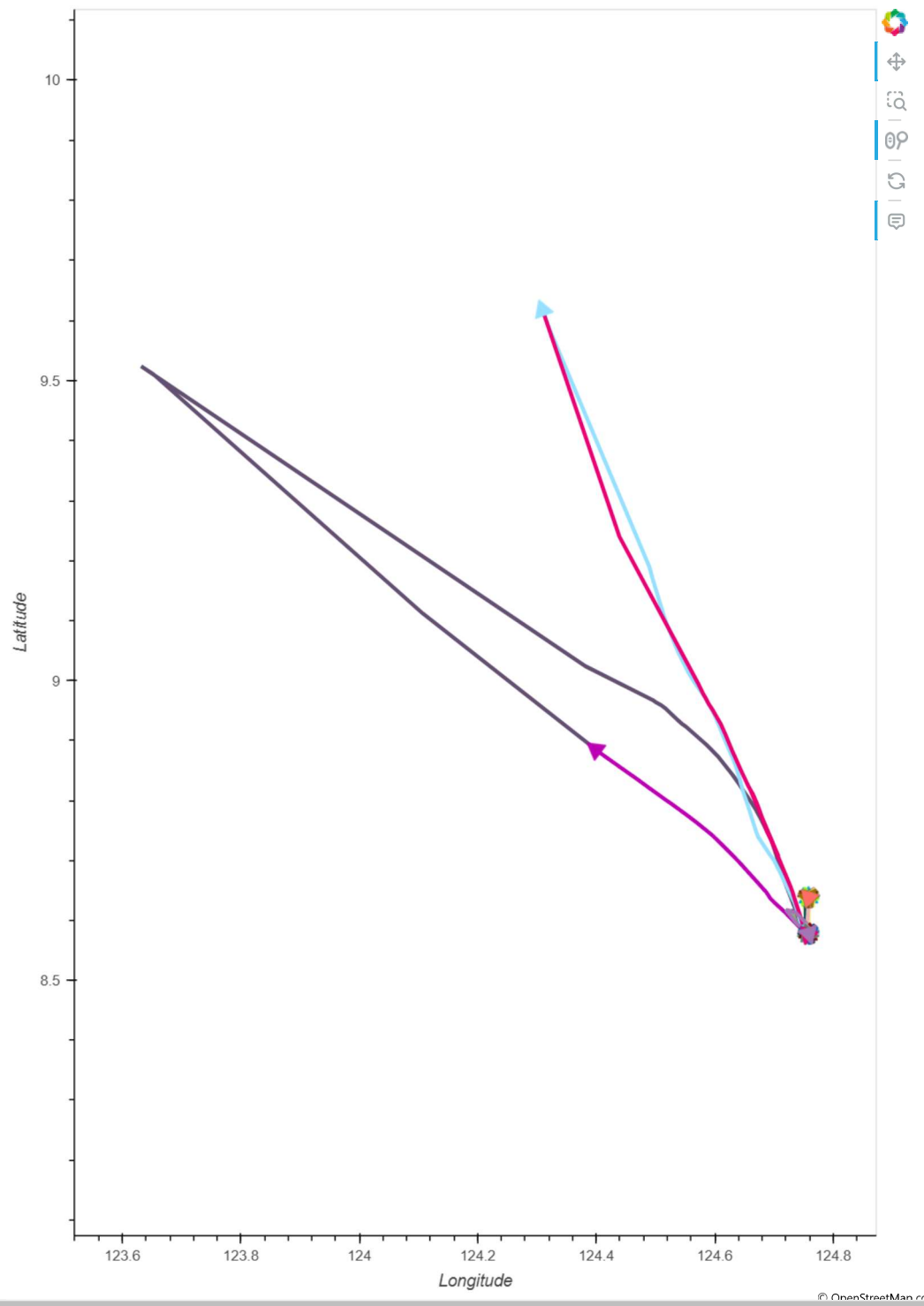
In [14]:
```
month_stops
```

Out[14]:
```
TrajectoryCollection with 39 trajectories
```

In [ ]:
```
month_stops.hvplot(geo=True, tiles=True)
```

In [16]:
```
# Trayectorias separadas por paradas
mpd.StopSplitter(month_trajectories).split(max_diameter=50, min_duration=timedelta(seconds=3600)
                                           ).hvplot(geo=True, tiles=True, width=700, height=1000)
```

Out[16]:



In [17]:
```
month_separated_trajectories = mpd.StopSplitter(month_trajectories
                                                ).split(max_diameter=50,
                                                        min_duration=timedelta(seconds=3600))
```

In [18]:
```
print(month_separated_trajectories)
```

```
TrajectoryCollection with 38 trajectories
```

Compresión de rutas

Generalizador de Timedelta (No usado en el bucle)

In [ ]:
```
trayectoria_compimida = mpd.MinTimeDeltaGeneralizer(trayectoria_sin_comprimir)
.generalize(tolerance=timedelta(minutes=5))
# Útil para tasas de transmisión altas (no es el caso), reduce mucho el tamaño del df.
```

Comprobador de velocidad (No usado en el bucle)

In [19]:
```
# Útil para comprobar fiabilidad de datos de trayectorias
# Válido para altas tasass de transmisión (no es el caso)
ejemplo_raw = pd.read_csv(os.path.join('Dynamic_by_MMSI', lista_archivos_dynamic[243]))
print(ejemplo_raw)
```

```
      timestamp       mmsi        lat       lon  speed  course  heading
0     1672527408  205278890  51.943305  4.175350    0.0    90.9      511
1     1672528129  205278890  51.943290  4.175340    0.0   104.0      511
2     1672528489  205278890  51.943330  4.175305    0.0    98.4      511
3     1672529031  205278890  51.943305  4.175380    0.0   115.5      511
4     1672529751  205278890  51.943320  4.175315    0.0   182.6      511
...          ...        ...        ...       ...    ...     ...      ...
3605  1675203957  205278890  51.101055  3.747345    0.0   101.8      511
3606  1675204317  205278890  51.101040  3.747355    0.0   104.5      511
3607  1675204676  205278890  51.100980  3.747360    0.0    70.3      511
3608  1675205038  205278890  51.101050  3.747445    0.0    82.2      511
3609  1675205397  205278890  51.101040  3.747420    0.0   124.6      511

[3610 rows x 7 columns]
```

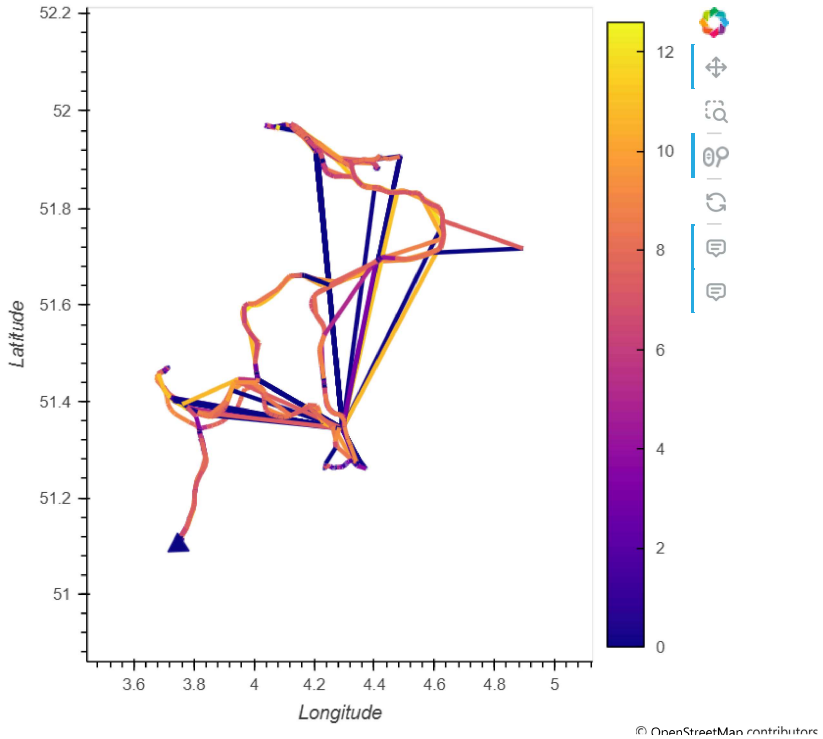
In [20]:
```
ejemplo_merged = pd.merge(ejemplo_raw, df_MMSI_shiptype, left_on='mmsi', right_on='MMSI',
                          how='inner')
ejemplo_merged.sort_values(by='timestamp', inplace=True)
ejemplo_merged.drop(columns=['MMSI'], inplace=True)
ejemplo_merged.dropna(subset=['shiptype'], inplace=True)
```

In [21]:
```
ejemplo_merged['timestamp'] = pd.to_datetime(ejemplo_merged['timestamp'], unit='s')
print(ejemplo_merged)
```

```
               timestamp       mmsi        lat       lon  speed  course  \
0    2022-12-31 22:56:48  205278890  51.943305  4.175350    0.0    90.9
1    2022-12-31 23:08:49  205278890  51.943290  4.175340    0.0   104.0
2    2022-12-31 23:14:49  205278890  51.943330  4.175305    0.0    98.4
3    2022-12-31 23:23:51  205278890  51.943305  4.175380    0.0   115.5
4    2022-12-31 23:35:51  205278890  51.943320  4.175315    0.0   182.6
...                  ...        ...        ...       ...    ...     ...
3605 2023-01-31 22:25:57  205278890  51.101055  3.747345    0.0   101.8
3606 2023-01-31 22:31:57  205278890  51.101040  3.747355    0.0   104.5
3607 2023-01-31 22:37:56  205278890  51.100980  3.747360    0.0    70.3
3608 2023-01-31 22:43:58  205278890  51.101050  3.747445    0.0    82.2
3609 2023-01-31 22:49:57  205278890  51.101040  3.747420    0.0   124.6

      heading         shiptype
0         511  ShipType.Tanker
1         511  ShipType.Tanker
2         511  ShipType.Tanker
3         511  ShipType.Tanker
4         511  ShipType.Tanker
...       ...              ...
3605      511  ShipType.Tanker
3606      511  ShipType.Tanker
3607      511  ShipType.Tanker
3608      511  ShipType.Tanker
3609      511  ShipType.Tanker

[3610 rows x 8 columns]
```

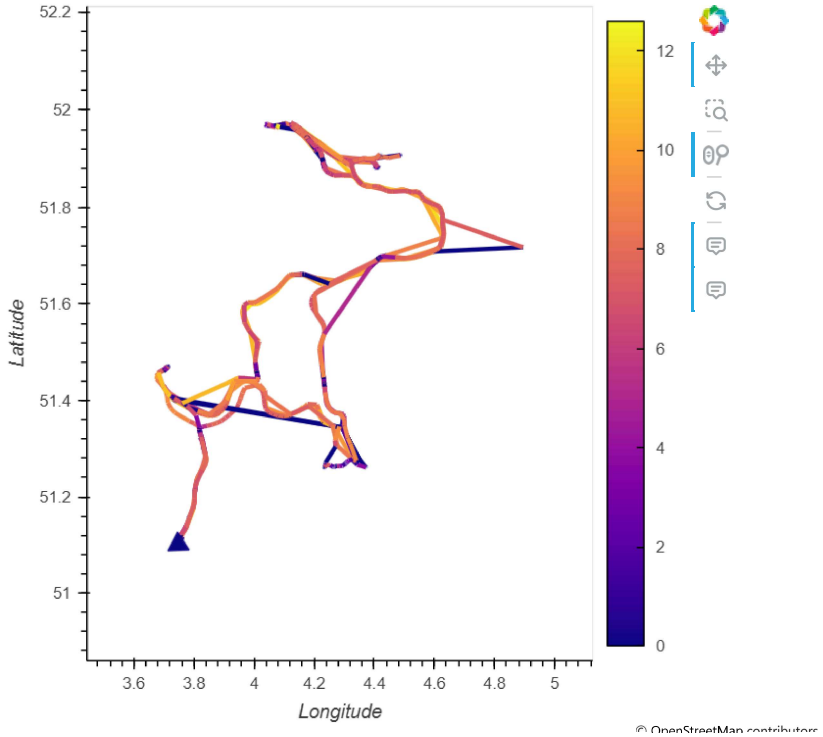
In [22]:
```
ejemplo_month_trajectories = mpd.Trajectory(ejemplo_merged,'ejemplo',
                                    t='timestamp', x='lon', y='lat', crs=4326)
ejemplo_month_trajectories.hvplot(c='speed', colorbar=True, tiles=True, width=500, height=500)
```

Out[22]:



In [23]:
```
ejemplo_month_trajectories = mpd.OutlierCleaner(ejemplo_month_trajectories
                                                ).clean(v_max=25, units=("nm", "h"))
ejemplo_month_trajectories.hvplot(c='speed', colorbar=True, tiles=True, width=500, height=500)
```

Out[23]:



BUCLES DE GENERACIÓN DE PARÁMETROS

Funciones iniciales

Cálculo de primeros parámetros con umbrales de 50m 3600s y longitud de DF>100)

In [ ]:
```
def calcular_parametros_trayectoria(month_separated_trajectories):
# DataFrame vacío con las columnas correspondientes
    traj_parameters = ['traj_N',
                       'traj_max_lat', 'traj_min_lat', 'traj_mean_lat',
                       'traj_max_lon', 'traj_min_lon', 'traj_mean_lon',
                       'traj_max_dif_lat', 'traj_max_dif_lon',
                       'traj_max_sep', 'traj_dist',
                       'traj_dist_0-0.5', 'traj_dist_0.5-1', 'traj_dist_1-1.5',
                       'traj_dist_1.5-2', 'traj_dist_2-5', 'traj_dist_5-10',
```

```
                    'traj_dist_10-20', 'traj_dist_>20',
                    'traj_time', 'traj_msg_rate',
                    'traj_time_at_speed_0', 'traj_time_over_speed_0',
                    'traj_mean_time_between_msg', 'traj_max_time_between_msg',
                    'traj_min_time_between_msg',
                    'traj_mean_speed', 'traj_max_speed', 'traj_min_speed',
                    'traj_mean_speed_var', 'traj_max_speed_var', 'traj_min_speed_var',
                    'traj_max_acc', 'traj_min_acc', 'traj_mean_acc',
                    'traj_max_dec', 'traj_min_dec', 'traj_mean_dec',
                    'traj_max_c_h_error', 'traj_min_c_h_error', 'traj_mean_c_h_error',
                    'traj_max_course_rate', 'traj_min_course_rate', 'traj_mean_course_rate',
                    'traj_max_head_rate', 'traj_min_head_rate', 'traj_mean_head_rate']

df_traj_parameters = pd.DataFrame(columns=traj_parameters)
# Comprobación de tiempos (usado para pruebas de rendimiento)
#time_geo = 0
#time_dist = 0
#time_temp = 0
#time_vel = 0
#time_acdec = 0
#time_ch = 0

for traj_N, trajectory in enumerate(month_separated_trajectories):

    # Parámetros geográficos
    #start_time = time.time()  # Captura el tiempo de inicio

    traj_max_lat = trajectory.df['lat'].max()
    traj_min_lat = trajectory.df['lat'].min()
    traj_mean_lat = trajectory.df['lat'].mean()

    traj_max_lon = trajectory.df['lon'].max()
    traj_min_lon = trajectory.df['lon'].min()
    traj_mean_lon = trajectory.df['lon'].mean()

    if (traj_max_lat >= 0 and traj_min_lat >= 0) or (traj_max_lat <= 0 and traj_min_lat <= 0):
        traj_max_dif_lat = abs(traj_max_lat) - abs(traj_min_lat)
    else:
        traj_max_dif_lat = abs(traj_max_lat) + abs(traj_min_lat)

    if (traj_max_lon >= 0 and traj_min_lon >= 0 or traj_max_lon <= 0 and traj_min_lon <= 0):
        traj_max_dif_lon = abs(traj_max_lon) - abs(traj_min_lon)
    else:
        traj_max_dif_lon = abs(traj_max_lon) + abs(traj_min_lon)
        if (traj_max_dif_lon>180):
            traj_max_dif_lon = 360 - traj_max_dif_lon

    #print("Tiempo después de calcular parametros geograficos", traj_N, ":", time.time() - start_time, "segundos")
    #time_geo = time_geo + time.time() - start_time

    # Distancias
    #start_time = time.time()  # Captura el tiempo de inicio
    # Distancias a punto más alejado

    # traj_max_sep requiere mucho procesamiento, se ha dejado a 0
    separations = 0 #trajectory.df.geometry.apply(lambda x: trajectory.df.geometry.distance(x))
    traj_max_sep = 0 #separations.apply(lambda x: x[x != 0].max()).max()

    traj_dist = trajectory.get_length()

    #print("Tiempo después de calcular parametros de distancias", traj_N, ":", time.time() - start_time, "segundos")
    #time_dist = time_dist + time.time() - start_time

    # Aspectos Temporales
    #start_time = time.time()  # Captura el tiempo de inicio

    trajectory.df['times_between_msgs'] = trajectory.df.index.to_series().diff().dt.total_seconds()

    traj_time = trajectory.df['times_between_msgs'].sum()

    traj_msg_rate = len(trajectory.df) / traj_time


    speed_ranges = [0, 0.5, 1, 1.5, 2, 5, 10, 20, 300]
    msg_by_speed_range = [((trajectory.df['speed'] >= inicio) & (trajectory.df['speed'] < fin)).sum()
                          for inicio, fin in zip(speed_ranges, speed_ranges[1:])]

    traj_time_at_speed_0 =
    (((trajectory.df['speed'].shift() == 0)
      & (trajectory.df['speed'] == 0))*trajectory.df['times_between_msgs']).sum()
    traj_time_over_speed_0 = traj_time - traj_time_at_speed_0

    traj_mean_time_between_msg = trajectory.df['times_between_msgs'].mean()
    traj_max_time_between_msg = trajectory.df['times_between_msgs'].max()
    traj_min_time_between_msg = trajectory.df['times_between_msgs'].min()

    #print("Tiempo después de calcular parametros temporales", traj_N, ":", time.time() - start_time, "segundos")
    #time_temp = time_temp + time.time() - start_time
```

```
        # Velocidades
        #start_time = time.time()  # Captura el tiempo de inicio

        traj_mean_speed = trajectory.df['speed'].mean()
        traj_max_speed = trajectory.df['speed'].max()
        traj_min_speed = trajectory.df['speed'].min()

        traj_mean_speed_var = trajectory.df['speed'].diff().mean()
        traj_max_speed_var = trajectory.df['speed'].diff().max()
        traj_min_speed_var = trajectory.df['speed'].diff().min()

        #print("Tiempo después de calcular parametros de velocidades", traj_N, ":", time.time() - start_time, "segundos
        #time_vel= time_vel + time.time() - start_time

        # Aceleraciones y deceleraciones
        #start_time = time.time()  # Captura el tiempo de inicio

        trajectory.df['acc_dec'] = trajectory.df['speed'].diff() / trajectory.df['times_between_msgs']
        trajectory.df['accelerations'] = trajectory.df[trajectory.df['acc_dec'] > 0]['acc_dec']
        trajectory.df['decelerations'] = trajectory.df[trajectory.df['acc_dec'] < 0]['acc_dec']

        traj_max_acc =
        trajectory.df['accelerations'].max() if not trajectory.df['accelerations'].empty else None
        traj_min_acc =
        trajectory.df['accelerations'].min() if not trajectory.df['accelerations'].empty else None
        traj_mean_acc =
        trajectory.df['accelerations'].mean() if not trajectory.df['accelerations'].empty else None

        traj_max_dec =
        trajectory.df['decelerations'].max() if not trajectory.df['decelerations'].empty else None
        traj_min_dec =
        trajectory.df['decelerations'].min() if not trajectory.df['decelerations'].empty else None
        traj_mean_dec =
        trajectory.df['decelerations'].mean() if not trajectory.df['decelerations'].empty else None

        #print("Tiempo después de calcular parametros de aceleraciones", traj_N, ":", time.time() - start_time, "segundo
        #time_acdec = time_acdec + time.time() - start_time

        # Errores de rumbo/dirección
        #start_time = time.time()  # Captura el tiempo de inicio

        trajectory.df['c_h_errors'] = trajectory.df['course'] - trajectory.df['heading']

        traj_max_c_h_error = trajectory.df['c_h_errors'].max()
        traj_min_c_h_error = trajectory.df['c_h_errors'].min()
        traj_mean_c_h_error = trajectory.df['c_h_errors'].mean()

        # Tasas de cambio de rumbo y dirección
        trajectory.df['course_rates'] =
        trajectory.df['course'].diff() / trajectory.df['times_between_msgs']

        traj_max_course_rate = trajectory.df['course_rates'].max()
        traj_min_course_rate = trajectory.df['course_rates'].min()
        traj_mean_course_rate = trajectory.df['course_rates'].mean()

        trajectory.df['head_rates'] =
        trajectory.df['heading'].diff() / trajectory.df['times_between_msgs']

        traj_max_head_rate = trajectory.df['head_rates'].max()
        traj_min_head_rate = trajectory.df['head_rates'].min()
        traj_mean_head_rate = trajectory.df['head_rates'].mean()

    #print("Tiempo después de calcular parametros de rumbo y direccion", traj_N, ":", time.time() - start_time, "segundo
        #time_ch = time_ch + time.time() - start_time

        # Agregar los resultados al DataFrame 'df_traj_parameters'
        df_traj_parameters.loc[traj_N] = [traj_N+1, traj_max_lat, traj_min_lat, traj_mean_lat,
                                  traj_max_lon, traj_min_lon, traj_mean_lon,
                                  traj_max_dif_lat, traj_max_dif_lon,
                                  traj_max_sep, traj_dist,
                                  msg_by_speed_range[0], msg_by_speed_range[1],
                                  msg_by_speed_range[2],
                                  msg_by_speed_range[3], msg_by_speed_range[4],
                                  msg_by_speed_range[5],
                                  msg_by_speed_range[6], msg_by_speed_range[7],
                                  traj_time, traj_msg_rate,
                                  traj_time_at_speed_0, traj_time_over_speed_0,
                                  traj_mean_time_between_msg, traj_max_time_between_msg,
                                  traj_min_time_between_msg,
                                  traj_mean_speed, traj_max_speed, traj_min_speed,
                                  traj_mean_speed_var, traj_max_speed_var, traj_min_speed_var,
                                  traj_max_acc, traj_min_acc, traj_mean_acc,
                                  traj_max_dec, traj_min_dec, traj_mean_dec,
                                  traj_max_c_h_error, traj_min_c_h_error, traj_mean_c_h_error,
                                  traj_max_course_rate, traj_min_course_rate, traj_mean_course_rate,
                                  traj_max_head_rate, traj_min_head_rate, traj_mean_head_rate]
    #print("TIEMPOS TOTALES:")
```

```
    #print("Tiempo en calcular geoparámetros:", time_geo)
    #print("Tiempo en calcular distancias:", time_dist)
    #print("Tiempo en calcular aspectos temporales:", time_temp)
    #print("Tiempo en calcular velocidades:", time_vel)
    #print("Tiempo en calcular aceleraciones y deceleraciones:", time_acdec)
    #print("Tiempo en calcular rumbos:", time_ch)
    return df_traj_parameters
```

In [ ]:
```
%%time
# Comprobación de la función para un MMSI (descomentar cálculos de tiempos en la función anterior)
df_traj_parameters = calcular_parametros_trayectoria(month_separated_trajectories)
pd.set_option('display.max_columns', None)
df_traj_parameters.head(9)
```

PARADAS

In [ ]:
```
def calcular_parametros_paradas (month_stops):
    stop_parameters = ['stop_N', 'stop_dist', 'stop_time']
    df_stop_parameters = pd.DataFrame(columns=stop_parameters)

    for stop_N, stop in enumerate(month_stops):

        stop_dist = stop.get_length()

        stop.df['times_between_msgs'] = stop.df.index.to_series().diff().dt.total_seconds()
        stop_time = stop.df['times_between_msgs'].sum()

        df_stop_parameters.loc[stop_N] = [stop_N+1, stop_dist, stop_time]
    return df_stop_parameters
```

In [ ]:
```
# Prueba para un MMSI
df_month_stops = calcular_parametros_paradas(month_stops)
df_month_stops.head()
```

CREACIÓN DE ATRIBUTOS GENÉRICOS A PARTIR DE LA RUTA COMPLETA Y DE LOS PARÁMETROS DE LAS INDIVIDUALES

In [ ]:
```
def calcular_parametros_cinematicos(df_traj_parameters, month_trajectories):
# DataFrame vacío con las columnas correspondientes
    parameters = ['N_Trajectories',
                  'traj_max_lat_mean', 'traj_min_lat_mean', 'traj_mean_lat_mean',
                  'traj_max_lon_mean', 'traj_min_lon_mean', 'traj_mean_lon_mean',
                  'month_max_lat', 'month_min_lat', 'month_mean_lat',
                  'month_max_lon', 'month_min_lon', 'month_mean_lon',
                  'traj_max_dif_lat_mean', 'traj_max_dif_lon_mean',
                  'month_max_dif_lat', 'month_max_dif_lon',
                  'traj_max_sep_mean', 'traj_dist_max', 'traj_dist_min', 'traj_dist_mean',
                  'month_dist', 'traj_time_mean', 'traj_msg_rate_mean',
                  'traj_msg_rate_max', 'traj_msg_rate_min',
                  'traj_time_at_speed_0_mean', 'traj_time_at_speed_0_max', 'traj_time_at_speed_0_min',
                  'traj_time_over_speed_0_mean', 'traj_time_over_speed_0_max',
                  'traj_time_over_speed_0_min',
                  'traj_mean_time_between_msg_mean', 'traj_max_time_between_msg_max',
                  'traj_min_time_between_msg_min',
                  'traj_msg_0_0dec5_mean', 'traj_msg_0dec5_1_mean', 'traj_msg_1_1dec5_mean',
                  'traj_msg_1dec5_2_mean', 'traj_msg_2_5_mean', 'traj_msg_5_10_mean',
                  'traj_msg_10_20_mean',
                  'traj_msg_over20_mean', 'traj_mean_speed_mean', 'month_mean_speed',
                  'traj_max_speed_max', 'month_max_speed',
                  'traj_min_speed_min', 'month_min_speed', 'traj_mean_speed_var_mean',
                  'traj_max_speed_var_max', 'traj_min_speed_var_min',
                  'traj_max_acc_max', 'traj_min_acc_min', 'traj_mean_acc_mean',
                  'traj_max_dec_max', 'traj_min_dec_min', 'traj_mean_dec_mean',
                  'traj_max_c_h_error_max', 'traj_min_c_h_error_min', 'traj_mean_c_h_error_mean',
                  'traj_max_course_rate_max', 'traj_min_course_rate_min', 'traj_mean_course_rate_mean',
                  'traj_max_head_rate_max', 'traj_min_head_rate_min', 'traj_mean_head_rate_mean'
                 ]
    time_geo = 0
    time_dist = 0
    time_temp = 0
    time_vel = 0
    time_acdec = 0
    time_ch = 0
    df_parameters_temp = pd.DataFrame(columns=parameters)

    N_Trajectories = df_traj_parameters['traj_N'].max()

    # Parámetros geográficos
    #start_time = time.time()  # Captura el tiempo de inicio

    traj_max_lat_mean = df_traj_parameters['traj_max_lat'].mean()
    traj_min_lat_mean = df_traj_parameters['traj_min_lat'].mean()
    traj_mean_lat_mean = df_traj_parameters['traj_mean_lat'].mean()
```

```
traj_max_lon_mean = df_traj_parameters['traj_max_lon'].mean()
traj_min_lon_mean = df_traj_parameters['traj_min_lon'].mean()
traj_mean_lon_mean = df_traj_parameters['traj_mean_lon'].mean()

month_max_lat = month_trajectories.df['lat'].max()
month_min_lat = month_trajectories.df['lat'].min()
month_mean_lat = month_trajectories.df['lat'].mean()

month_max_lon = month_trajectories.df['lon'].max()
month_min_lon = month_trajectories.df['lon'].min()
month_mean_lon = month_trajectories.df['lon'].mean()

traj_max_dif_lat_mean = df_traj_parameters['traj_max_dif_lat'].mean()
traj_max_dif_lon_mean = df_traj_parameters['traj_max_dif_lon'].mean()
# Diferencia de latitudes máxima en todo el mes
if ((month_max_lat >= 0 and month_min_lat >= 0) or (month_max_lat <= 0 and month_min_lat <= 0)):
    month_max_dif_lat = abs(month_max_lat) - abs(month_min_lat)
else:
    month_max_dif_lat = abs(month_max_lat) + abs(month_min_lat)
# Diferencia de longitudes máxima en todo el mes
if ((month_max_lon >= 0 and month_min_lon >= 0) or (month_max_lon <= 0 and month_min_lon <= 0)):
    month_max_dif_lon = abs(month_max_lon) - abs(month_min_lon)
else:
    month_max_dif_lon = abs(month_max_lon) + abs(month_min_lon)
    if (month_max_dif_lon>180):
        month_max_dif_lon = 360 - month_max_dif_lon

#print("Tiempo después de calcular parametros geograficos:", time.time() - start_time, "segundos")
#time_geo = time_geo + time.time() - start_time

# Distancias
#start_time = time.time()  # Captura el tiempo de inicio

# Distancias a punto más alejado (se deja a 0 pero se calcula como viene debajo)
traj_max_sep_mean = 0
#df_traj_parameters['traj_max_sep'].mean()
separations = 0
#month_trajectories.df.geometry.apply(lambda x: month_trajectories.df.geometry.distance(x))
month_max_sep = 0
#separations.apply(lambda x: x[x != 0].max()).max()

traj_dist_max = df_traj_parameters['traj_dist'].max()
traj_dist_min = df_traj_parameters['traj_dist'].min()
traj_dist_mean = df_traj_parameters['traj_dist'].mean()

month_dist = month_trajectories.get_length()

#print("Tiempo después de calcular parametros de distancias:", time.time() - start_time, "segundos")
#time_dist = time_dist + time.time() - start_time

# Aspectos Temporales
#start_time = time.time()  # Captura el tiempo de inicio

traj_time_mean = df_traj_parameters['traj_time'].mean()

traj_msg_rate_mean = df_traj_parameters['traj_msg_rate'].mean()
traj_msg_rate_max = df_traj_parameters['traj_msg_rate'].max()
traj_msg_rate_min = df_traj_parameters['traj_msg_rate'].min()

traj_msg_0_0dec5_mean = df_traj_parameters['traj_dist_0-0.5'].mean()
traj_msg_0dec5_1_mean = df_traj_parameters['traj_dist_0.5-1'].mean()
traj_msg_1_1dec5_mean = df_traj_parameters['traj_dist_1-1.5'].mean()
traj_msg_1dec5_2_mean = df_traj_parameters['traj_dist_1.5-2'].mean()
traj_msg_2_5_mean = df_traj_parameters['traj_dist_2-5'].mean()
traj_msg_5_10_mean = df_traj_parameters['traj_dist_5-10'].mean()
traj_msg_10_20_mean = df_traj_parameters['traj_dist_10-20'].mean()
traj_msg_over20_mean = df_traj_parameters['traj_dist_>20'].mean()

traj_time_at_speed_0_mean = df_traj_parameters['traj_time_at_speed_0'].mean()
traj_time_at_speed_0_max = df_traj_parameters['traj_time_at_speed_0'].max()
traj_time_at_speed_0_min = df_traj_parameters['traj_time_at_speed_0'].min()

traj_time_over_speed_0_mean = df_traj_parameters['traj_time_over_speed_0'].mean()
traj_time_over_speed_0_max = df_traj_parameters['traj_time_over_speed_0'].max()
traj_time_over_speed_0_min = df_traj_parameters['traj_time_over_speed_0'].min()

traj_mean_time_between_msg_mean = df_traj_parameters['traj_mean_time_between_msg'].mean()
traj_max_time_between_msg_max = df_traj_parameters['traj_max_time_between_msg'].max()
traj_min_time_between_msg_min = df_traj_parameters['traj_min_time_between_msg'].min()

#print("Tiempo después de calcular parametros temporales:", time.time() - start_time, "segundos")
#time_temp = time_temp + time.time() - start_time

# Velocidades
#start_time = time.time()  # Captura el tiempo de inicio

traj_mean_speed_mean = df_traj_parameters['traj_mean_speed'].mean()
```

```
    month_mean_speed = month_trajectories.df['speed'].mean()

    traj_max_speed_max = df_traj_parameters['traj_max_speed'].max()
    month_max_speed = month_trajectories.df['speed'].max()

    traj_min_speed_min = df_traj_parameters['traj_min_speed'].min()
    month_min_speed = month_trajectories.df['speed'].min()

    traj_mean_speed_var_mean = df_traj_parameters['traj_mean_speed_var'].mean()
    traj_max_speed_var_max = df_traj_parameters['traj_max_speed_var'].max()
    traj_min_speed_var_min = df_traj_parameters['traj_min_speed_var'].min()

    #print("Tiempo después de calcular parametros de velocidades:", time.time() - start_time, "segundos")
    #time_vel= time_vel + time.time() - start_time

    # Aceleraciones y deceleraciones
    #start_time = time.time()  # Captura el tiempo de inicio


    traj_max_acc_max = df_traj_parameters['traj_max_acc'].max()
    traj_min_acc_min = df_traj_parameters['traj_min_acc'].min()
    traj_mean_acc_mean = df_traj_parameters['traj_mean_acc'].mean()

    traj_max_dec_max = df_traj_parameters['traj_max_dec'].max()
    traj_min_dec_min = df_traj_parameters['traj_min_dec'].min()
    traj_mean_dec_mean = df_traj_parameters['traj_mean_dec'].mean()

    #print("Tiempo después de calcular parametros de aceleraciones:", time.time() - start_time, "segundos")
    #time_acdec = time_acdec + time.time() - start_time

    # Errores de rumbo/dirección
    #start_time = time.time()  # Captura el tiempo de inicio

    traj_max_c_h_error_max = df_traj_parameters['traj_max_c_h_error'].max()
    traj_min_c_h_error_min = df_traj_parameters['traj_min_c_h_error'].min()
    traj_mean_c_h_error_mean = df_traj_parameters['traj_mean_c_h_error'].mean()

    # Tasas de cambio de rumbo y dirección
    traj_max_course_rate_max = df_traj_parameters['traj_max_course_rate'].max()
    traj_min_course_rate_min = df_traj_parameters['traj_min_course_rate'].min()
    traj_mean_course_rate_mean = df_traj_parameters['traj_mean_course_rate'].mean()

    traj_max_head_rate_max = df_traj_parameters['traj_max_head_rate'].max()
    traj_min_head_rate_min = df_traj_parameters['traj_min_head_rate'].min()
    traj_mean_head_rate_mean = df_traj_parameters['traj_mean_head_rate'].mean()

    #print("Tiempo después de calcular parametros de rumbo y direccion:", time.time() - start_time, "segundos")
    #time_ch = time_ch + time.time() - start_time

    # Agregar los resultados al DataFrame 'df_traj_parameters'
    parameters_temp = {
    'N_Trajectories': [N_Trajectories],
    'traj_max_lat_mean': [traj_max_lat_mean],
    'traj_min_lat_mean': [traj_min_lat_mean],
    'traj_mean_lat_mean': [traj_mean_lat_mean],
    'traj_max_lon_mean': [traj_max_lon_mean],
    'traj_min_lon_mean': [traj_min_lon_mean],
    'traj_mean_lon_mean': [traj_mean_lon_mean],
    'month_max_lat': [month_max_lat],
    'month_min_lat': [month_min_lat],
    'month_mean_lat': [month_mean_lat],
    'month_max_lon': [month_max_lon],
    'month_min_lon': [month_min_lon],
    'month_mean_lon': [month_mean_lon],
    'traj_max_dif_lat_mean': [traj_max_dif_lat_mean],
    'traj_max_dif_lon_mean': [traj_max_dif_lon_mean],
    'month_max_dif_lat': [month_max_dif_lat],
    'month_max_dif_lon': [month_max_dif_lon],
    'traj_max_sep_mean': [traj_max_sep_mean],
    'traj_dist_max': [traj_dist_max],
    'traj_dist_min': [traj_dist_min],
    'traj_dist_mean': [traj_dist_mean],
    'month_dist': [month_dist],
    'traj_time_mean': [traj_time_mean],
    'traj_msg_rate_mean': [traj_msg_rate_mean],
    'traj_msg_rate_max': [traj_msg_rate_max],
    'traj_msg_rate_min': [traj_msg_rate_min],
    'traj_msg_0_0dec5_mean': [traj_msg_0_0dec5_mean],
    'traj_msg_0dec5_1_mean': [traj_msg_0dec5_1_mean],
    'traj_msg_1_1dec5_mean': [traj_msg_1_1dec5_mean],
    'traj_msg_1dec5_2_mean': [traj_msg_1dec5_2_mean],
    'traj_msg_2_5_mean': [traj_msg_2_5_mean],
    'traj_msg_5_10_mean': [traj_msg_5_10_mean],
    'traj_msg_10_20_mean': [traj_msg_10_20_mean],
    'traj_msg_over20_mean': [traj_msg_over20_mean],
    'traj_time_at_speed_0_mean': [traj_time_at_speed_0_mean],
    'traj_time_at_speed_0_max': [traj_time_at_speed_0_max],
    'traj_time_at_speed_0_min': [traj_time_at_speed_0_min],
```

```
    'traj_time_over_speed_0_mean': [traj_time_over_speed_0_mean],
    'traj_time_over_speed_0_max': [traj_time_over_speed_0_max],
    'traj_time_over_speed_0_min': [traj_time_over_speed_0_min],
    'traj_mean_time_between_msg_mean': [traj_mean_time_between_msg_mean],
    'traj_max_time_between_msg_max': [traj_max_time_between_msg_max],
    'traj_min_time_between_msg_min': [traj_min_time_between_msg_min],
    'traj_mean_speed_mean': [traj_mean_speed_mean],
    'month_mean_speed': [month_mean_speed],
    'traj_max_speed_max': [traj_max_speed_max],
    'month_max_speed': [month_max_speed],
    'traj_min_speed_min': [traj_min_speed_min],
    'month_min_speed': [month_min_speed],
    'traj_mean_speed_var_mean': [traj_mean_speed_var_mean],
    'traj_max_speed_var_max': [traj_max_speed_var_max],
    'traj_min_speed_var_min': [traj_min_speed_var_min],
    'traj_max_acc_max': [traj_max_acc_max],
    'traj_min_acc_min': [traj_min_acc_min],
    'traj_mean_acc_mean': [traj_mean_acc_mean],
    'traj_max_dec_max': [traj_max_dec_max],
    'traj_min_dec_min': [traj_min_dec_min],
    'traj_mean_dec_mean': [traj_mean_dec_mean],
    'traj_max_c_h_error_max': [traj_max_c_h_error_max],
    'traj_min_c_h_error_min': [traj_min_c_h_error_min],
    'traj_mean_c_h_error_mean': [traj_mean_c_h_error_mean],
    'traj_max_course_rate_max': [traj_max_course_rate_max],
    'traj_min_course_rate_min': [traj_min_course_rate_min],
    'traj_mean_course_rate_mean': [traj_mean_course_rate_mean],
    'traj_max_head_rate_max': [traj_max_head_rate_max],
    'traj_min_head_rate_min': [traj_min_head_rate_min],
    'traj_mean_head_rate_mean': [traj_mean_head_rate_mean]
    }
    #print("TIEMPOS TOTALES:")
    #print("Tiempo en calcular geoparámetros:", time_geo)
    #print("Tiempo en calcular distancias:", time_dist)
    #print("Tiempo en calcular aspectos temporales:", time_temp)
    #print("Tiempo en calcular velocidades:", time_vel)
    #print("Tiempo en calcular aceleraciones y deceleraciones:", time_acdec)
    #print("Tiempo en calcular rumbos:", time_ch)
    # Crear el DataFrame
    df_parameters_temp = pd.DataFrame(parameters_temp)
    return df_parameters_temp
```

In [ ]:
```
%%time
# Comprobación con un MMSI
prueba = calcular_parametros_cinematicos(df_traj_parameters, month_trajectories)
prueba.head()
```

BUCLE DE CREACIÓN DEL DF_CINEMATIC_PARAMETERS PARA TODOS LOS MMSI

ITERACIÓN 1

UMBRALES: 100m 3600s DF>100

In [ ]:
```
# Leer el DataFrame que contiene la lista de MMSI y SHIPTYPE
df_MMSI_shiptype = pd.read_csv('MMSI_shiptype_data.csv')
total_time = 0
containing_nan = 0
too_short_csv = 0
non_associated_shiptypes = 0

# DataFrame vacío para almacenar los parámetros globales
df_cinematic_parameters = df_MMSI_shiptype.copy()
# Itera sobre cada fila del DataFrame que contiene MMSI y SHIPTYPE
for MMSI, row in df_cinematic_parameters.iterrows():
    # Busca el archivo CSV correspondiente al MMSI actual
    csv_file = os.path.join('Dynamic_by_MMSI', f"dynamic_{row['MMSI']}.csv")

    # Verifica si el archivo CSV existe
    if os.path.exists(csv_file):
        start_time = time.time()  # Captura el tiempo de inicio de procesamiento total de MMSI

        # Lee el archivo CSV y crear el DataFrame
        df_raw_dynamic = pd.read_csv(csv_file)
        #  Verificar la longitud del DataFrame
        if len(df_raw_dynamic) >= 100:
            # Merge basado en las columnas 'mmsi' y 'MMSI' y preparación del df
            df_merged_dynamic = pd.merge(df_raw_dynamic, df_MMSI_shiptype,
                                         left_on='mmsi', right_on='MMSI', how='inner')

            # Llamar a la función para preprocesar df_merged_dynamic:
            df_merged_dynamic = clean(df_merged_dynamic)
```

```
            if not(df_merged_dynamic.isnull().any().any()):
                # Crear trayectorias
                month_trajectories =
                mpd.Trajectory(df_merged_dynamic,'MMSI',
                               t='timestamp', x='lon_copy', y='lat_copy', crs=4326)

                # Separar las trayectorias
                month_separated_trajectories =
                mpd.StopSplitter(month_trajectories).split(max_diameter=50,
                                                           min_duration=timedelta(seconds=3600))

                # Calcular parámetros de trayectoria
                df_traj_parameters = pd.DataFrame()
                df_traj_parameters = calcular_parametros_trayectoria(month_separated_trajectories)

                # Calcular parámetros globales
                df_cinematic_parameters_temp =
                calcular_parametros_cinematicos(df_traj_parameters, month_trajectories)
                df_cinematic_parameters.loc[MMSI, df_cinematic_parameters_temp.columns] =
                df_cinematic_parameters_temp.iloc[0].values
                # Asociar los parámetros globales al MMSI y SHIPTYPE actual
                df_cinematic_parameters.loc[df_cinematic_parameters.index[-1], ['MMSI', 'shiptype']] =
                [row['MMSI'], row['shiptype']]
                print("MMSI", row['MMSI'], "finished in", time.time() - start_time, "seconds")
                total_time = total_time + time.time() - start_time

            # En caso de que existan NaN
            else:
                containing_nan = containing_nan + 1
        # En caso de que la longitud sea menor de 100
        else:
            too_short_csv = too_short_csv + 1
    # En caso de que no se haya asociado un shiptype al MMSI
    else:
        non_associated_shiptypes = non_associated_shiptypes + 1

print( "Existen", containing_nan, "MMSI cuyo df contiene Nan.")
print( "Existen", too_short_csv, "MMSI que no cumplen con la longitud requerida.")
print( "Existen", non_associated_shiptypes, "MMSI que no se han asociado a ningún shiptype.")
print( "Tiempo total en extraer parámetros:", total_time, "segundos.")

# Guardar los parámetros globales en un archivo CSV
#df_cinematic_parameters.to_csv('cinematic_parameters.csv', index=False)
```

Segunda Iteración

Se eliminan parámetros y se calcula porcentaje de mensajes por trayectoria a cierta velocidad. (No se cambian rangos) UMBRALES: 100m 3600s DF>100

In [ ]:
```
def clean(df):
    # Eliminar columna 'MMSI'
    df.drop(columns=['mmsi'], inplace=True)
    # Ordenar por timestamp
    df.sort_values(by='timestamp', inplace=True)
    # Convertir timestamp a datetime
    df['timestamp'] = pd.to_datetime(df['timestamp'], unit='s')
    # Reemplazar valores de latitud fuera de rango con NaN
    df['lat'] = df['lat'].where((df['lat'] >= -90) & (df['lat'] <= 90), np.nan)
    # Reemplazar valores de longitud fuera de rango con NaN
    df['lon'] = df['lon'].where((df['lon'] >= -180) & (df['lon'] <= 180), np.nan)
    # Hacer copias de lat y lon
    df = df.assign(lat_copy=df['lat'], lon_copy=df['lon'])
    return df
```

In [ ]:
```
def calcular_parametros_trayectoria(month_separated_trajectories):
# DataFrame vacío con las columnas correspondientes
    traj_parameters = ['traj_N', 'traj_dist',
                       'traj_dist_0-0.5', 'traj_dist_0.5-1', 'traj_dist_1-1.5',
                       'traj_dist_1.5-2', 'traj_dist_2-5', 'traj_dist_5-10',
                       'traj_dist_10-20', 'traj_dist_>20',
                       'traj_time',
                       'traj_mean_speed', 'traj_max_speed',
                       'traj_mean_acc', 'traj_mean_dec',
                       ]

    df_traj_parameters = pd.DataFrame(columns=traj_parameters)

    for traj_N, trajectory in enumerate(month_separated_trajectories):

        # Distancias

        traj_dist = trajectory.get_length()
```

```
        # Aspectos Temporales
        trajectory.df['times_between_msgs'] = trajectory.df.index.to_series().diff().dt.total_seconds()

        traj_time = trajectory.df['times_between_msgs'].sum()

        speed_ranges = [0, 0.5, 1, 1.5, 2, 5, 10, 20, 300]
        msg_by_speed_range =
        [((((trajectory.df['speed'] >= inicio)
           & (trajectory.df['speed'] < fin)).sum() )/len(trajectory.df)
                            for inicio, fin in zip(speed_ranges, speed_ranges[1:])]

        # Velocidades
        traj_mean_speed = trajectory.df['speed'].mean()
        traj_max_speed = trajectory.df['speed'].max()

        # Aceleraciones y deceleraciones

        trajectory.df['acc_dec'] = trajectory.df['speed'].diff() / trajectory.df['times_between_msgs']
        trajectory.df['accelerations'] = trajectory.df[trajectory.df['acc_dec'] > 0]['acc_dec']
        trajectory.df['decelerations'] = trajectory.df[trajectory.df['acc_dec'] < 0]['acc_dec']

        traj_mean_acc =
        trajectory.df['accelerations'].mean() if not trajectory.df['accelerations'].empty else None

        traj_mean_dec =
        trajectory.df['decelerations'].mean() if not trajectory.df['decelerations'].empty else None

        # Agregar los resultados al DataFrame 'df_traj_parameters'
        df_traj_parameters.loc[traj_N] = [traj_N+1, traj_dist,
                                    msg_by_speed_range[0], msg_by_speed_range[1], msg_by_speed_range[2],
                                    msg_by_speed_range[3], msg_by_speed_range[4], msg_by_speed_range[5],
                                    msg_by_speed_range[6], msg_by_speed_range[7],
                                    traj_time,
                                    traj_mean_speed, traj_max_speed, traj_mean_acc, traj_mean_dec,
                                    ]

    return df_traj_parameters
```

In [ ]:
```
def calcular_parametros_paradas (month_stops):
    stop_parameters = ['stop_N', 'stop_dist', 'stop_time']
    df_stop_parameters = pd.DataFrame(columns=stop_parameters)

    for stop_N, stop in enumerate(month_stops):

        stop_dist = stop.get_length()

        stop.df['times_between_msgs'] = stop.df.index.to_series().diff().dt.total_seconds()
        stop_time = stop.df['times_between_msgs'].sum()

        df_stop_parameters.loc[stop_N] = [stop_N+1, stop_dist, stop_time]
```

In [ ]:
```
def calcular_parametros_cinematicos(df_traj_parameters, month_trajectories):
# DataFrame vacío con las columnas correspondientes
    parameters = ['N_Trajectories',
                  'month_max_lat', 'month_min_lat', 'month_mean_lat',
                  'month_max_lon', 'month_min_lon', 'month_mean_lon',
                  'month_max_dif_lat', 'month_max_dif_lon',
                  'traj_dist_max', 'traj_dist_mean',
                  'month_dist', 'traj_time_mean',
                  'traj_msg_0_0dec5_mean', 'traj_msg_0dec5_1_mean', 'traj_msg_1_1dec5_mean',
                  'traj_msg_1dec5_2_mean', 'traj_msg_2_5_mean', 'traj_msg_5_10_mean',
                  'traj_msg_10_20_mean',
                  'traj_msg_over20_mean', 'traj_mean_speed_mean', 'month_mean_speed',
                  'traj_max_speed_max', 'month_max_speed',
                  'traj_mean_acc_mean', 'traj_mean_dec_mean',
                 ]

    df_parameters_temp = pd.DataFrame(columns=parameters)

    N_Trajectories = df_traj_parameters['traj_N'].max()

    # Parámetros geográficos

    month_max_lat = month_trajectories.df['lat'].max()
    month_min_lat = month_trajectories.df['lat'].min()
    month_mean_lat = month_trajectories.df['lat'].mean()

    month_max_lon = month_trajectories.df['lon'].max()
    month_min_lon = month_trajectories.df['lon'].min()
    month_mean_lon = month_trajectories.df['lon'].mean()

    # Diferencia de latitudes máxima en todo el mes
    if ((month_max_lat >= 0 and month_min_lat >= 0) or (month_max_lat <= 0 and month_min_lat <= 0)):
        month_max_dif_lat = abs(month_max_lat) - abs(month_min_lat)
    else:
        month_max_dif_lat = abs(month_max_lat) + abs(month_min_lat)
    # Diferencia de longitudes máxima en todo el mes
```

```
    if ((month_max_lon >= 0 and month_min_lon >= 0) or (month_max_lon <= 0 and month_min_lon <= 0)):
        month_max_dif_lon = abs(month_max_lon) - abs(month_min_lon)
    else:
        month_max_dif_lon = abs(month_max_lon) + abs(month_min_lon)
        if (month_max_dif_lon>180):
            month_max_dif_lon = 360 - month_max_dif_lon

    # Distancias

    traj_dist_max = df_traj_parameters['traj_dist'].max()
    traj_dist_mean = df_traj_parameters['traj_dist'].mean()

    month_dist = month_trajectories.get_length()

    # Aspectos Temporales
    traj_time_mean = df_traj_parameters['traj_time'].mean()
    traj_time_max = df_traj_parameters['traj_time'].max()
    traj_time_min = df_traj_parameters['traj_time'].min()

    traj_msg_0_0dec5_mean = df_traj_parameters['traj_dist_0-0.5'].mean()
    traj_msg_0dec5_1_mean = df_traj_parameters['traj_dist_0.5-1'].mean()
    traj_msg_1_1dec5_mean = df_traj_parameters['traj_dist_1-1.5'].mean()
    traj_msg_1dec5_2_mean = df_traj_parameters['traj_dist_1.5-2'].mean()
    traj_msg_2_5_mean = df_traj_parameters['traj_dist_2-5'].mean()
    traj_msg_5_10_mean = df_traj_parameters['traj_dist_5-10'].mean()
    traj_msg_10_20_mean = df_traj_parameters['traj_dist_10-20'].mean()
    traj_msg_over20_mean = df_traj_parameters['traj_dist_>20'].mean()

    # Velocidades

    traj_mean_speed_mean = df_traj_parameters['traj_mean_speed'].mean()
    month_mean_speed = month_trajectories.df['speed'].mean()

    traj_max_speed_max = df_traj_parameters['traj_max_speed'].max()
    month_max_speed = month_trajectories.df['speed'].max()

    # Aceleraciones y deceleraciones

    traj_mean_acc_mean = df_traj_parameters['traj_mean_acc'].mean()

    traj_mean_dec_mean = df_traj_parameters['traj_mean_dec'].mean()

    # Agrega los resultados al DataFrame 'df_traj_parameters'
    parameters_temp = {
    'N_Trajectories': [N_Trajectories],
    'month_max_lat': [month_max_lat],
    'month_min_lat': [month_min_lat],
    'month_mean_lat': [month_mean_lat],
    'month_max_lon': [month_max_lon],
    'month_min_lon': [month_min_lon],
    'month_mean_lon': [month_mean_lon],
    'month_max_dif_lat': [month_max_dif_lat],
    'month_max_dif_lon': [month_max_dif_lon],
    'traj_dist_max': [traj_dist_max],
    'traj_dist_mean': [traj_dist_mean],
    'month_dist': [month_dist],
    'traj_time_mean': [traj_time_mean],
    'traj_msg_0_0dec5_mean': [traj_msg_0_0dec5_mean],
    'traj_msg_0dec5_1_mean': [traj_msg_0dec5_1_mean],
    'traj_msg_1_1dec5_mean': [traj_msg_1_1dec5_mean],
    'traj_msg_1dec5_2_mean': [traj_msg_1dec5_2_mean],
    'traj_msg_2_5_mean': [traj_msg_2_5_mean],
    'traj_msg_5_10_mean': [traj_msg_5_10_mean],
    'traj_msg_10_20_mean': [traj_msg_10_20_mean],
    'traj_msg_over20_mean': [traj_msg_over20_mean],
    'traj_mean_speed_mean': [traj_mean_speed_mean],
    'month_mean_speed': [month_mean_speed],
    'traj_max_speed_max': [traj_max_speed_max],
    'month_max_speed': [month_max_speed],
    'traj_mean_acc_mean': [traj_mean_acc_mean],
    'traj_mean_dec_mean': [traj_mean_dec_mean]
    }

    # Crear el DataFrame
    df_parameters_temp = pd.DataFrame(parameters_temp)
    return df_parameters_temp
```

In [ ]:

```
# Leer el DataFrame que contiene la lista de MMSI y SHIPTYPE
df_MMSI_shiptype = pd.read_csv('MMSI_shiptype_data.csv')
total_time = 0
containing_nan = 0
too_short_csv = 0
non_associated_shiptypes = 0
tiempos_procesamiento = {'MMSI': [], 'tiempo_procesamiento': []}


# DataFrame vacío para almacenar los parÃ¡metros globales
```

```
df_cinematic_parameters = df_MMSI_shiptype.copy()
# Itera sobre cada fila del DataFrame que contiene MMSI y SHIPTYPE
for MMSI, row in tqdm(df_cinematic_parameters.iterrows(),
                      total=len(df_cinematic_parameters), desc="Processing MMSIs"):

    start_time = time.time()  # Captura el tiempo de inicio de procesamiento total de MMSI
    # Busca el archivo CSV correspondiente al MMSI actual
    csv_file = os.path.join('Dynamic_by_MMSI', f"csvEnero2023/dynamic_{row['MMSI']}.csv")

    # Verifica si el archivo CSV existe
    if os.path.exists(csv_file):

        # Lee el archivo CSV y crear el DataFrame
        df_raw_dynamic = pd.read_csv(csv_file)
        #  Verificar la longitud del DataFrame
        if len(df_raw_dynamic) >= 100:
            # Merge basado en las columnas 'mmsi' y 'MMSI' y preparación del df
            df_merged_dynamic = pd.merge(df_raw_dynamic, df_MMSI_shiptype,
                                         left_on='mmsi', right_on='MMSI', how='inner')

            # Llamar a la función para preprocesar df_merged_dynamic:
            df_merged_dynamic = clean(df_merged_dynamic)

            if not(df_merged_dynamic.isnull().any().any()):
                # Crear trayectorias
                month_trajectories =
                mpd.Trajectory(df_merged_dynamic,'MMSI',
                               t='timestamp', x='lon_copy', y='lat_copy', crs=4326)

                # Separar las trayectorias
                month_separated_trajectories =
                mpd.StopSplitter(month_trajectories).split(max_diameter=100,
                                                           min_duration=timedelta(seconds=3600))

                # Calcular parámetros de trayectoria
                df_traj_parameters = pd.DataFrame()
                df_traj_parameters = calcular_parametros_trayectoria(month_separated_trajectories)

                # Calcular parámetros globales
                df_cinematic_parameters_temp =
                calcular_parametros_cinematicos(df_traj_parameters, month_trajectories)
                df_cinematic_parameters.loc[MMSI, df_cinematic_parameters_temp.columns] =
                df_cinematic_parameters_temp.iloc[0].values

                # Asociar los parámetros globales al MMSI y SHIPTYPE actual
                df_cinematic_parameters.loc[df_cinematic_parameters.index[-1], ['MMSI', 'shiptype']] =
                [row['MMSI'], row['shiptype']]

            # En caso de que existan NaN
            else:
                containing_nan = containing_nan + 1
        # En caso de que la longitud sea menor de 100
        else:
            too_short_csv = too_short_csv + 1
    # En caso de que no se haya asociado un shiptype al MMSI
    else:
        non_associated_shiptypes = non_associated_shiptypes + 1

    total_time = total_time + time.time() - start_time
    tiempos_procesamiento['MMSI'].append(row['MMSI'])
    tiempos_procesamiento['tiempo_procesamiento'].append(time.time() - start_time)
    df_tiempos_procesamiento = pd.DataFrame(tiempos_procesamiento)
    df_cinematic_parameters.to_csv('cinematic_parameters_2.csv', index=False)
    df_tiempos_procesamiento.to_csv('tiempos_procesamiento_2.csv', index=False)
print( "Existen", containing_nan, "MMSI cuyo df contiene Nan.")
print( "Existen", too_short_csv, "MMSI que no cumplen con la longitud requerida.")
print( "Existen", non_associated_shiptypes, "MMSI que no se han asociado a ningÃºn shiptype.")
print( "Tiempo total en extraer parÃ¡metros:", total_time, "segundos.")

# Guardar los parámetros globales en un archivo CSV
#df_cinematic_parameters.to_csv('cinematic_parameters.csv', index=False)
```

TERCERA ITERACIÓN

Se cambian umbrales a 3600 y 50

In [ ]:

```
def clean(df):
    # Eliminar columna 'MMSI'
    df.drop(columns=['mmsi', 'course', 'heading'], inplace=True)
    # Ordenar por timestamp
    df.sort_values(by='timestamp', inplace=True)
    # Convertir timestamp a datetime
    df['timestamp'] = pd.to_datetime(df['timestamp'], unit='s')
    # Reemplazar valores de latitud fuera de rango con NaN
```

```
    df['lat'] = df['lat'].where((df['lat'] >= -90) & (df['lat'] <= 90), np.nan)
    # Reemplazar valores de longitud fuera de rango con NaN
    df['lon'] = df['lon'].where((df['lon'] >= -180) & (df['lon'] <= 180), np.nan)
    # Hacer copias de lat y lon
    df = df.assign(lat_copy=df['lat'], lon_copy=df['lon'])
    return df
```

In [ ]:
```
def calcular_parametros_trayectoria(month_separated_trajectories):
# DataFrame vacío con las columnas correspondientes
    traj_parameters = ['traj_N', 'traj_dist',
                       'traj_dist_0-0.5', 'traj_dist_0.5-1', 'traj_dist_1-1.5',
                       'traj_dist_1.5-2', 'traj_dist_2-5', 'traj_dist_5-10',
                       'traj_dist_10-20', 'traj_dist_>20',
                       'traj_time',
                       'traj_mean_speed', 'traj_max_speed',
                       'traj_mean_acc', 'traj_mean_dec',
                       ]

    df_traj_parameters = pd.DataFrame(columns=traj_parameters)

    for traj_N, trajectory in enumerate(month_separated_trajectories):

        # Distancias

        traj_dist = trajectory.get_length()

        # Aspectos Temporales
        trajectory.df['times_between_msgs'] = trajectory.df.index.to_series().diff().dt.total_seconds()

        traj_time = trajectory.df['times_between_msgs'].sum()

        speed_ranges = [0, 0.5, 1, 1.5, 2, 5, 10, 20, 300]
        msg_by_speed_range =
        [(((trajectory.df['speed'] >= inicio)
           & (trajectory.df['speed'] < fin)).sum() )/len(trajectory.df)
                            for inicio, fin in zip(speed_ranges, speed_ranges[1:])]

        # Velocidades
        traj_mean_speed = trajectory.df['speed'].mean()
        traj_max_speed = trajectory.df['speed'].max()

        # Aceleraciones y deceleraciones

        trajectory.df['acc_dec'] = trajectory.df['speed'].diff() / trajectory.df['times_between_msgs']
        trajectory.df['accelerations'] = trajectory.df[trajectory.df['acc_dec'] > 0]['acc_dec']
        trajectory.df['decelerations'] = trajectory.df[trajectory.df['acc_dec'] < 0]['acc_dec']

        traj_mean_acc =
        trajectory.df['accelerations'].mean() if not trajectory.df['accelerations'].empty else None

        traj_mean_dec =
        trajectory.df['decelerations'].mean() if not trajectory.df['decelerations'].empty else None

        # Agregar los resultados al DataFrame 'df_traj_parameters'
        df_traj_parameters.loc[traj_N] = [traj_N+1, traj_dist,
                                    msg_by_speed_range[0], msg_by_speed_range[1],
                                    msg_by_speed_range[2],
                                    msg_by_speed_range[3], msg_by_speed_range[4],
                                    msg_by_speed_range[5],
                                    msg_by_speed_range[6], msg_by_speed_range[7],
                                    traj_time,
                                    traj_mean_speed, traj_max_speed, traj_mean_acc, traj_mean_dec,
                                    ]

    return df_traj_parameters
```

In [ ]:
```
def calcular_parametros_paradas (month_stops):
    stop_parameters = ['stop_N', 'stop_dist', 'stop_time']
    df_stop_parameters = pd.DataFrame(columns=stop_parameters)

    for stop_N, stop in enumerate(month_stops):

        stop_dist = stop.get_length()

        stop.df['times_between_msgs'] = stop.df.index.to_series().diff().dt.total_seconds()
        stop_time = stop.df['times_between_msgs'].sum()

        df_stop_parameters.loc[stop_N] = [stop_N+1, stop_dist, stop_time]
```

In [ ]:
```
def calcular_parametros_cinematicos(df_traj_parameters, month_trajectories):
# DataFrame vacío con las columnas correspondientes
    parameters = ['N_Trajectories',
                  'month_max_lat', 'month_min_lat', 'month_mean_lat',
                  'month_max_lon', 'month_min_lon', 'month_mean_lon',
                  'month_max_dif_lat', 'month_max_dif_lon',
                  'traj_dist_max', 'traj_dist_mean',
```

```
                'month_dist', 'traj_time_mean',
                'traj_msg_0_0dec5_mean', 'traj_msg_0dec5_1_mean',
                'traj_msg_1_1dec5_mean', 'traj_msg_1dec5_2_mean',
                'traj_msg_2_5_mean', 'traj_msg_5_10_mean', 'traj_msg_10_20_mean',
                'traj_msg_over20_mean', 'traj_mean_speed_mean',
                'month_mean_speed', 'traj_max_speed_max', 'month_max_speed',
                'traj_mean_acc_mean', 'traj_mean_dec_mean',
               ]

df_parameters_temp = pd.DataFrame(columns=parameters)

#Parámetros de paradas

#N_stops = df_stop_parameters['stop_N'].max()
#stop_dist_max =df_stop_parameters['stop_dist'].max()
#stop_dist_min =df_stop_parameters['stop_dist'].min()
#stop_dist_mean =df_stop_parameters['stop_dist'].mean()
#stop_time_max =df_stop_parameters['stop_time'].max()
#stop_time_min =df_stop_parameters['stop_time'].min()
#stop_time_mean =df_stop_parameters['stop_time'].mean()


N_Trajectories = df_traj_parameters['traj_N'].max()

# Parámetros geográficos

month_max_lat = month_trajectories.df['lat'].max()
month_min_lat = month_trajectories.df['lat'].min()
month_mean_lat = month_trajectories.df['lat'].mean()

month_max_lon = month_trajectories.df['lon'].max()
month_min_lon = month_trajectories.df['lon'].min()
month_mean_lon = month_trajectories.df['lon'].mean()

# Diferencia de latitudes máxima en todo el mes
if ((month_max_lat >= 0 and month_min_lat >= 0) or (month_max_lat <= 0 and month_min_lat <= 0)):
    month_max_dif_lat = abs(month_max_lat) - abs(month_min_lat)
else:
    month_max_dif_lat = abs(month_max_lat) + abs(month_min_lat)
# Diferencia de longitudes máxima en todo el mes
if ((month_max_lon >= 0 and month_min_lon >= 0) or (month_max_lon <= 0 and month_min_lon <= 0)):
    month_max_dif_lon = abs(month_max_lon) - abs(month_min_lon)
else:
    month_max_dif_lon = abs(month_max_lon) + abs(month_min_lon)
    if (month_max_dif_lon>180):
        month_max_dif_lon = 360 - month_max_dif_lon

# Distancias

traj_dist_max = df_traj_parameters['traj_dist'].max()
traj_dist_mean = df_traj_parameters['traj_dist'].mean()

month_dist = month_trajectories.get_length()

# Aspectos Temporales
traj_time_mean = df_traj_parameters['traj_time'].mean()
traj_time_max = df_traj_parameters['traj_time'].max()
traj_time_min = df_traj_parameters['traj_time'].min()

traj_msg_0_0dec5_mean = df_traj_parameters['traj_dist_0-0.5'].mean()
traj_msg_0dec5_1_mean = df_traj_parameters['traj_dist_0.5-1'].mean()
traj_msg_1_1dec5_mean = df_traj_parameters['traj_dist_1-1.5'].mean()
traj_msg_1dec5_2_mean = df_traj_parameters['traj_dist_1.5-2'].mean()
traj_msg_2_5_mean = df_traj_parameters['traj_dist_2-5'].mean()
traj_msg_5_10_mean = df_traj_parameters['traj_dist_5-10'].mean()
traj_msg_10_20_mean = df_traj_parameters['traj_dist_10-20'].mean()
traj_msg_over20_mean = df_traj_parameters['traj_dist_>20'].mean()

# Velocidades

traj_mean_speed_mean = df_traj_parameters['traj_mean_speed'].mean()
month_mean_speed = month_trajectories.df['speed'].mean()

traj_max_speed_max = df_traj_parameters['traj_max_speed'].max()
month_max_speed = month_trajectories.df['speed'].max()

# Aceleraciones y deceleraciones

traj_mean_acc_mean = df_traj_parameters['traj_mean_acc'].mean()

traj_mean_dec_mean = df_traj_parameters['traj_mean_dec'].mean()

# Agregar los resultados al DataFrame 'df_traj_parameters'
parameters_temp = {
'N_Trajectories': [N_Trajectories],
'month_max_lat': [month_max_lat],
'month_min_lat': [month_min_lat],
'month_mean_lat': [month_mean_lat],
```

```
    'month_max_lon': [month_max_lon],
    'month_min_lon': [month_min_lon],
    'month_mean_lon': [month_mean_lon],
    'month_max_dif_lat': [month_max_dif_lat],
    'month_max_dif_lon': [month_max_dif_lon],
    'traj_dist_max': [traj_dist_max],
    'traj_dist_mean': [traj_dist_mean],
    'month_dist': [month_dist],
    'traj_time_mean': [traj_time_mean],
    'traj_msg_0_0dec5_mean': [traj_msg_0_0dec5_mean],
    'traj_msg_0dec5_1_mean': [traj_msg_0dec5_1_mean],
    'traj_msg_1_1dec5_mean': [traj_msg_1_1dec5_mean],
    'traj_msg_1dec5_2_mean': [traj_msg_1dec5_2_mean],
    'traj_msg_2_5_mean': [traj_msg_2_5_mean],
    'traj_msg_5_10_mean': [traj_msg_5_10_mean],
    'traj_msg_10_20_mean': [traj_msg_10_20_mean],
    'traj_msg_over20_mean': [traj_msg_over20_mean],
    'traj_mean_speed_mean': [traj_mean_speed_mean],
    'month_mean_speed': [month_mean_speed],
    'traj_max_speed_max': [traj_max_speed_max],
    'month_max_speed': [month_max_speed],
    'traj_mean_acc_mean': [traj_mean_acc_mean],
    'traj_mean_dec_mean': [traj_mean_dec_mean]
    }

    # Crear el DataFrame
    df_parameters_temp = pd.DataFrame(parameters_temp)
    return df_parameters_temp
```

In [ ]:

```
# Leer el DataFrame que contiene la lista de MMSI y SHIPTYPE
df_MMSI_shiptype = pd.read_csv('MMSI_shiptype_data.csv')
total_time = 0
containing_nan = 0
too_short_csv = 0
non_associated_shiptypes = 0
tiempos_procesamiento = {'MMSI': [], 'tiempo_procesamiento': []}


# DataFrame vacío para almacenar los parámetros globales
df_cinematic_parameters = df_MMSI_shiptype.copy()
# Itera sobre cada fila del DataFrame que contiene MMSI y SHIPTYPE
for MMSI, row in tqdm(df_cinematic_parameters.iterrows(), total=
                      len(df_cinematic_parameters), desc="Processing MMSIs"):

    start_time = time.time()  # Captura el tiempo de inicio de procesamiento total de MMSI
    # Busca el archivo CSV correspondiente al MMSI actual
    csv_file = os.path.join('Dynamic_by_MMSI', f"dynamic_{row['MMSI']}.csv")
    # Verifica si el archivo CSV existe
    if os.path.exists(csv_file):

        # Lee el archivo CSV y crear el DataFrame
        df_raw_dynamic = pd.read_csv(csv_file)
        #  Verificar la longitud del DataFrame
        if len(df_raw_dynamic) >= 100:
            # Merge basado en las columnas 'mmsi' y 'MMSI' y preparación del df
            df_merged_dynamic = pd.merge(df_raw_dynamic, df_MMSI_shiptype,
                                         left_on='mmsi', right_on='MMSI', how='inner')

            # Llamar a la función para preprocesar df_merged_dynamic:
            df_merged_dynamic = clean(df_merged_dynamic)

            if not(df_merged_dynamic.isnull().any().any()):
                # Crear trayectorias
                month_trajectories = mpd.Trajectory(df_merged_dynamic,'MMSI', t='timestamp',
                                                    x='lon_copy', y='lat_copy', crs=4326)

                # Separar las trayectorias
                month_separated_trajectories =
                    mpd.StopSplitter(month_trajectories).split(max_diameter=50, min_duration=timedelta(seconds=3600))

                # Calcular parámetros de trayectoria
                df_traj_parameters = pd.DataFrame()
                df_traj_parameters = calcular_parametros_trayectoria(month_separated_trajectories)

                # Calcular parámetros globales
                df_cinematic_parameters_temp =
                calcular_parametros_cinematicos(df_traj_parameters, month_trajectories)
                df_cinematic_parameters.loc[MMSI, df_cinematic_parameters_temp.columns] =
                df_cinematic_parameters_temp.iloc[0].values

                # Asociar los parámetros globales al MMSI y SHIPTYPE actual
                df_cinematic_parameters.loc[df_cinematic_parameters.index[-1], ['MMSI', 'shiptype']] =
                [row['MMSI'], row['shiptype']]

            # En caso de que existan NaN
            else:
                containing_nan = containing_nan + 1
```

```
        # En caso de que la longitud sea menor de 100
        else:
            too_short_csv = too_short_csv + 1
    # En caso de que no se haya asociado un shiptype al MMSI
    else:
        non_associated_shiptypes = non_associated_shiptypes + 1

    total_time = total_time + time.time() - start_time
    tiempos_procesamiento['MMSI'].append(row['MMSI'])
    tiempos_procesamiento['tiempo_procesamiento'].append(time.time() - start_time)
    df_tiempos_procesamiento = pd.DataFrame(tiempos_procesamiento)
    df_cinematic_parameters.to_csv('cinematic_parameters_2.csv', index=False)
    df_tiempos_procesamiento.to_csv('tiempos_procesamiento_2.csv', index=False)
print( "Existen", containing_nan, "MMSI cuyo df contiene Nan.")
print( "Existen", too_short_csv, "MMSI que no cumplen con la longitud requerida.")
print( "Existen", non_associated_shiptypes, "MMSI que no se han asociado a ningún shiptype.")
print( "Tiempo total en extraer parÃ¡metros:", total_time, "segundos.")

# Guardar los parámetros globales en un archivo CSV
#df_cinematic_parameters.to_csv('cinematic_parameters.csv', index=False)
```

Cuarta Iteración

Se Implemantan parámetros de paradas y se añaden rangos de velocidades.

In [ ]:
```
def clean(df):
    # Eliminar columna 'MMSI'
    df.drop(columns=['mmsi'], inplace=True)
    # Ordenar por timestamp
    df.sort_values(by='timestamp', inplace=True)
    # Convertir timestamp a datetime
    df['timestamp'] = pd.to_datetime(df['timestamp'], unit='s')
    # Reemplazar valores de latitud fuera de rango con NaN
    df['lat'] = df['lat'].where((df['lat'] >= -90) & (df['lat'] <= 90), np.nan)
    # Reemplazar valores de longitud fuera de rango con NaN
    df['lon'] = df['lon'].where((df['lon'] >= -180) & (df['lon'] <= 180), np.nan)
    # Hacer copias de lat y lon
    df = df.assign(lat_copy=df['lat'], lon_copy=df['lon'])
    return df
```

In [ ]:
```
def calcular_parametros_trayectoria(month_separated_trajectories):
# DataFrame vaco con las columnas correspondientes
    traj_parameters = ['traj_N', 'traj_dist',
                    # Rangos de velocidad de 0 a 5 en incrementos de 0.5
                    'traj_percent_0-0.5', 'traj_percent_0.5-1', 'traj_percent_1-1.5',
                    'traj_percent_1.5-2', 'traj_percent_2-2.5', 'traj_percent_2.5-3',
                    'traj_percent_3-3.5',
                    'traj_percent_3.5-4', 'traj_percent_4-4.5', 'traj_percent_4.5-5',
                    # Rangos de velocidad de 0 a 10 en incrementos de 1
                    'traj_percent_0-1', 'traj_percent_1-2', 'traj_percent_2-3',
                    'traj_percent_3-4', 'traj_percent_4-5', 'traj_percent_5-6',
                    'traj_percent_6-7', 'traj_percent_7-8', 'traj_percent_8-9',
                    'traj_percent_9-10',
                    # Rangos de velocidad de 0 a 20 en incrementos de 2
                    'traj_percent_0-2', 'traj_percent_2-4', 'traj_percent_4-6',
                    'traj_percent_6-8', 'traj_percent_8-10', 'traj_percent_10-12',
                    'traj_percent_12-14', 'traj_percent_14-16', 'traj_percent_16-18',
                    'traj_percent_18-20',
                    # Rangos de velocidad de 0 a 20 en incrementos de 5
                    'traj_percent_0-5', 'traj_percent_5-10', 'traj_percent_10-15',
                    'traj_percent_15-20',
                    # Rango de velocidad mayor de 20
                    'traj_dist_>20',
                    'traj_time', 'traj_mean_speed', 'traj_max_speed',
                    'traj_mean_acc', 'traj_mean_dec']

    df_traj_parameters = pd.DataFrame(columns=traj_parameters)

    for traj_N, trajectory in enumerate(month_separated_trajectories):

        # Distancias

        traj_dist = trajectory.get_length()

        # Aspectos Temporales
        trajectory.df['times_between_msgs'] = trajectory.df.index.to_series().diff().dt.total_seconds()

        traj_time = trajectory.df['times_between_msgs'].sum()

        # Rangos de velocidades de 0,5
        speed_ranges_0dec5 = [0, 0.5, 1, 1.5, 2, 2.5, 3, 3.5, 4, 4.5, 5]
        percent_by_speed_range_0dec5 = [(((trajectory.df['speed'] >= inicio)
                                          & (trajectory.df['speed'] < fin)).sum() )/len(trajectory.df)
                                        for inicio, fin in zip(speed_ranges_0dec5, speed_ranges_0dec5[1:])]
```

```
        # Rangos de velocidades de 1
        speed_ranges_1 = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
        percent_by_speed_range_1 = [(((trajectory.df['speed'] >= inicio)
                                      & (trajectory.df['speed'] < fin)).sum() )/len(trajectory.df)
                                    for inicio, fin in zip(speed_ranges_1, speed_ranges_1[1:])]
        # Rangos de velocidades de 2
        speed_ranges_2 = [0, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20]
        percent_by_speed_range_2 = [(((trajectory.df['speed'] >= inicio)
                                      & (trajectory.df['speed'] < fin)).sum() )/len(trajectory.df)
                                    for inicio, fin in zip(speed_ranges_2, speed_ranges_2[1:])]
        # Rangos de velocidades de 5
        speed_ranges_5 = [0, 5, 10, 15, 20]
        percent_by_speed_range_5 = [(((trajectory.df['speed'] >= inicio)
                                      & (trajectory.df['speed'] < fin)).sum() )/len(trajectory.df)
                                    for inicio, fin in zip(speed_ranges_5, speed_ranges_5[1:])]
        # Velocidades mayores de 20
        speed_ranges_gt20 = [20, 300]
        percent_by_speed_range_gt20 = [(((trajectory.df['speed'] >= inicio)
                                         & (trajectory.df['speed'] < fin)).sum() )/len(trajectory.df)
                                       for inicio, fin in zip(speed_ranges_gt20, speed_ranges_gt20[1:])]


        # Velocidades
        traj_mean_speed = trajectory.df['speed'].mean()
        traj_max_speed = trajectory.df['speed'].max()

        # Aceleraciones y deceleraciones

        trajectory.df['acc_dec'] = trajectory.df['speed'].diff() / trajectory.df['times_between_msgs']
        trajectory.df['accelerations'] = trajectory.df[trajectory.df['acc_dec'] > 0]['acc_dec']
        trajectory.df['decelerations'] = trajectory.df[trajectory.df['acc_dec'] < 0]['acc_dec']

        traj_mean_acc = trajectory.df['accelerations'].mean()
        if not trajectory.df['accelerations'].empty else None

        traj_mean_dec = trajectory.df['decelerations'].mean()
        if not trajectory.df['decelerations'].empty else None

        # Agregar los resultados al DataFrame 'df_traj_parameters'
        # Agregar los resultados al DataFrame 'df_traj_parameters'
        df_traj_parameters.loc[traj_N] = [traj_N+1, traj_dist,
                                          *percent_by_speed_range_0dec5,
                                          *percent_by_speed_range_1,
                                          *percent_by_speed_range_2,
                                          *percent_by_speed_range_5,
                                          *percent_by_speed_range_gt20,
                                          traj_time,
                                          traj_mean_speed, traj_max_speed,
                                          traj_mean_acc, traj_mean_dec]

    return df_traj_parameters
```

In [ ]:

```
def calcular_parametros_paradas (month_stops):
    stop_parameters = ['stop_N', 'stop_dist', 'stop_time']
    df_stop_parameters = pd.DataFrame(columns=stop_parameters)

    for stop_N, stop in enumerate(month_stops):

        stop_dist = stop.get_length()

        stop.df['times_between_msgs'] = stop.df.index.to_series().diff().dt.total_seconds()
        stop_time = stop.df['times_between_msgs'].sum()

        df_stop_parameters.loc[stop_N] = [stop_N+1, stop_dist, stop_time]
    return df_stop_parameters
```

In [ ]:

```
def calcular_parametros_cinematicos(df_traj_parameters, month_trajectories):
    # DataFrame vacío con las columnas correspondientes
    parameters = ['N_Trajectories',
                  'month_max_lat', 'month_min_lat', 'month_mean_lat',
                  'month_max_lon', 'month_min_lon', 'month_mean_lon',
                  'month_max_dif_lat', 'month_max_dif_lon',
                  'traj_dist_max', 'traj_dist_mean',
                  'month_dist', 'traj_time_mean',
                  'traj_percent_0_0dec5_mean', 'traj_percent_0dec5_1_mean', 'traj_percent_1_1dec5_mean',
                  'traj_percent_1dec5_2_mean', 'traj_percent_2_2dec5_mean', 'traj_percent_2dec5_3_mean',
                  'traj_percent_3_3dec5_mean', 'traj_percent_3dec5_4_mean', 'traj_percent_4_4dec5_mean',
                  'traj_percent_4dec5_5_mean', 'traj_percent_0_1_mean', 'traj_percent_1_2_mean',
                  'traj_percent_2_3_mean', 'traj_percent_3_4_mean', 'traj_percent_4_5_mean',
                  'traj_percent_5_6_mean', 'traj_percent_6_7_mean', 'traj_percent_7_8_mean',
                  'traj_percent_8_9_mean', 'traj_percent_9_10_mean', 'traj_percent_0_2_mean',
                  'traj_percent_2_4_mean', 'traj_percent_4_6_mean', 'traj_percent_6_8_mean',
                  'traj_percent_8_10_mean', 'traj_percent_10_12_mean', 'traj_percent_12_14_mean',
                  'traj_percent_14_16_mean', 'traj_percent_16_18_mean', 'traj_percent_18_20_mean',
                  'traj_percent_0_5_mean', 'traj_percent_5_10_mean', 'traj_percent_10_15_mean',
                  'traj_percent_15_20_mean', 'traj_dist_over20_mean', 'traj_mean_speed_mean',
                  'month_mean_speed', 'traj_max_speed_max', 'month_max_speed',
```

```
                'traj_mean_acc_mean', 'traj_mean_dec_mean',
                'N_stops', 'stop_dist_max', 'stop_dist_min', 'stop_dist_mean',
                'stop_time_max', 'stop_time_min','stop_time_mean'
               ]

df_parameters_temp = pd.DataFrame(columns=parameters)

# Parámetros de paradas

N_stops = df_stop_parameters['stop_N'].max()
stop_dist_max = df_stop_parameters['stop_dist'].max()
stop_dist_min = df_stop_parameters['stop_dist'].min()
stop_dist_mean = df_stop_parameters['stop_dist'].mean()
stop_time_max = df_stop_parameters['stop_time'].max()
stop_time_min = df_stop_parameters['stop_time'].min()
stop_time_mean = df_stop_parameters['stop_time'].mean()

N_Trajectories = df_traj_parameters['traj_N'].max()

# Parámetros geográficos

month_max_lat = month_trajectories.df['lat'].max()
month_min_lat = month_trajectories.df['lat'].min()
month_mean_lat = month_trajectories.df['lat'].mean()

month_max_lon = month_trajectories.df['lon'].max()
month_min_lon = month_trajectories.df['lon'].min()
month_mean_lon = month_trajectories.df['lon'].mean()

# Diferencia de latitudes máxima en todo el mes
if ((month_max_lat >= 0 and month_min_lat >= 0) or (month_max_lat <= 0 and month_min_lat <= 0)):
    month_max_dif_lat = abs(month_max_lat) - abs(month_min_lat)
else:
    month_max_dif_lat = abs(month_max_lat) + abs(month_min_lat)
# Diferencia de longitudes máxima en todo el mes
if ((month_max_lon >= 0 and month_min_lon >= 0) or (month_max_lon <= 0 and month_min_lon <= 0)):
    month_max_dif_lon = abs(month_max_lon) - abs(month_min_lon)
else:
    month_max_dif_lon = abs(month_max_lon) + abs(month_min_lon)
    if (month_max_dif_lon > 180):
        month_max_dif_lon = 360 - month_max_dif_lon

# Distancias

traj_dist_max = df_traj_parameters['traj_dist'].max()
traj_dist_mean = df_traj_parameters['traj_dist'].mean()

month_dist = month_trajectories.get_length()

# Aspectos Temporales
traj_time_mean = df_traj_parameters['traj_time'].mean()
traj_time_max = df_traj_parameters['traj_time'].max()
traj_time_min = df_traj_parameters['traj_time'].min()

# Rangos de Velocidades
traj_percent_0_0dec5_mean = df_traj_parameters['traj_percent_0-0.5'].mean()
traj_percent_0dec5_1_mean = df_traj_parameters['traj_percent_0.5-1'].mean()
traj_percent_1_1dec5_mean = df_traj_parameters['traj_percent_1-1.5'].mean()
traj_percent_1dec5_2_mean = df_traj_parameters['traj_percent_1.5-2'].mean()
traj_percent_2_2dec5_mean = df_traj_parameters['traj_percent_2-2.5'].mean()
traj_percent_2dec5_3_mean = df_traj_parameters['traj_percent_2.5-3'].mean()
traj_percent_3_3dec5_mean = df_traj_parameters['traj_percent_3-3.5'].mean()
traj_percent_3dec5_4_mean = df_traj_parameters['traj_percent_3.5-4'].mean()
traj_percent_4_4dec5_mean = df_traj_parameters['traj_percent_4-4.5'].mean()
traj_percent_4dec5_5_mean = df_traj_parameters['traj_percent_4.5-5'].mean()

traj_percent_0_1_mean = df_traj_parameters['traj_percent_0-1'].mean()
traj_percent_1_2_mean = df_traj_parameters['traj_percent_1-2'].mean()
traj_percent_2_3_mean = df_traj_parameters['traj_percent_2-3'].mean()
traj_percent_3_4_mean = df_traj_parameters['traj_percent_3-4'].mean()
traj_percent_4_5_mean = df_traj_parameters['traj_percent_4-5'].mean()
traj_percent_5_6_mean = df_traj_parameters['traj_percent_5-6'].mean()
traj_percent_6_7_mean = df_traj_parameters['traj_percent_6-7'].mean()
traj_percent_7_8_mean = df_traj_parameters['traj_percent_7-8'].mean()
traj_percent_8_9_mean = df_traj_parameters['traj_percent_8-9'].mean()
traj_percent_9_10_mean = df_traj_parameters['traj_percent_9-10'].mean()

traj_percent_0_2_mean = df_traj_parameters['traj_percent_0-2'].mean()
traj_percent_2_4_mean = df_traj_parameters['traj_percent_2-4'].mean()
traj_percent_4_6_mean = df_traj_parameters['traj_percent_4-6'].mean()
traj_percent_6_8_mean = df_traj_parameters['traj_percent_6-8'].mean()
traj_percent_8_10_mean = df_traj_parameters['traj_percent_8-10'].mean()
traj_percent_10_12_mean = df_traj_parameters['traj_percent_10-12'].mean()
traj_percent_12_14_mean = df_traj_parameters['traj_percent_12-14'].mean()
traj_percent_14_16_mean = df_traj_parameters['traj_percent_14-16'].mean()
traj_percent_16_18_mean = df_traj_parameters['traj_percent_16-18'].mean()
traj_percent_18_20_mean = df_traj_parameters['traj_percent_18-20'].mean()
```

```
    traj_percent_0_5_mean = df_traj_parameters['traj_percent_0-5'].mean()
    traj_percent_5_10_mean = df_traj_parameters['traj_percent_5-10'].mean()
    traj_percent_10_15_mean = df_traj_parameters['traj_percent_10-15'].mean()
    traj_percent_15_20_mean = df_traj_parameters['traj_percent_15-20'].mean()

    traj_dist_over20_mean = df_traj_parameters['traj_dist_>20'].mean()

    # Velocidades

    traj_mean_speed_mean = df_traj_parameters['traj_mean_speed'].mean()
    month_mean_speed = month_trajectories.df['speed'].mean()

    traj_max_speed_max = df_traj_parameters['traj_max_speed'].max()
    month_max_speed = month_trajectories.df['speed'].max()

    # Aceleraciones y deceleraciones

    traj_mean_acc_mean = df_traj_parameters['traj_mean_acc'].mean()

    traj_mean_dec_mean = df_traj_parameters['traj_mean_dec'].mean()

    # Agregar los resultados al DataFrame 'df_traj_parameters'
    parameters_temp = {
        'N_Trajectories': [N_Trajectories],
        'month_max_lat': [month_max_lat],
        'month_min_lat': [month_min_lat],
        'month_mean_lat': [month_mean_lat],
        'month_max_lon': [month_max_lon],
        'month_min_lon': [month_min_lon],
        'month_mean_lon': [month_mean_lon],
        'month_max_dif_lat': [month_max_dif_lat],
        'month_max_dif_lon': [month_max_dif_lon],
        'traj_dist_max': [traj_dist_max],
        'traj_dist_mean': [traj_dist_mean],
        'month_dist': [month_dist],
        'traj_time_mean': [traj_time_mean],
        'traj_percent_0_0dec5_mean': [traj_percent_0_0dec5_mean],
        'traj_percent_0dec5_1_mean': [traj_percent_0dec5_1_mean],
        'traj_percent_1_1dec5_mean': [traj_percent_1_1dec5_mean],
        'traj_percent_1dec5_2_mean': [traj_percent_1dec5_2_mean],
        'traj_percent_2_2dec5_mean': [traj_percent_2_2dec5_mean],
        'traj_percent_2dec5_3_mean': [traj_percent_2dec5_3_mean],
        'traj_percent_3_3dec5_mean': [traj_percent_3_3dec5_mean],
        'traj_percent_3dec5_4_mean': [traj_percent_3dec5_4_mean],
        'traj_percent_4_4dec5_mean': [traj_percent_4_4dec5_mean],
        'traj_percent_4dec5_5_mean': [traj_percent_4dec5_5_mean],
        'traj_percent_0_1_mean': [traj_percent_0_1_mean],
        'traj_percent_1_2_mean': [traj_percent_1_2_mean],
        'traj_percent_2_3_mean': [traj_percent_2_3_mean],
        'traj_percent_3_4_mean': [traj_percent_3_4_mean],
        'traj_percent_4_5_mean': [traj_percent_4_5_mean],
        'traj_percent_5_6_mean': [traj_percent_5_6_mean],
        'traj_percent_6_7_mean': [traj_percent_6_7_mean],
        'traj_percent_7_8_mean': [traj_percent_7_8_mean],
        'traj_percent_8_9_mean': [traj_percent_8_9_mean],
        'traj_percent_9_10_mean': [traj_percent_9_10_mean],
        'traj_percent_0_2_mean': [traj_percent_0_2_mean],
        'traj_percent_2_4_mean': [traj_percent_2_4_mean],
        'traj_percent_4_6_mean': [traj_percent_4_6_mean],
        'traj_percent_6_8_mean': [traj_percent_6_8_mean],
        'traj_percent_8_10_mean': [traj_percent_8_10_mean],
        'traj_percent_10_12_mean': [traj_percent_10_12_mean],
        'traj_percent_12_14_mean': [traj_percent_12_14_mean],
        'traj_percent_14_16_mean': [traj_percent_14_16_mean],
        'traj_percent_16_18_mean': [traj_percent_16_18_mean],
        'traj_percent_18_20_mean': [traj_percent_18_20_mean],
        'traj_percent_0_5_mean': [traj_percent_0_5_mean],
        'traj_percent_5_10_mean': [traj_percent_5_10_mean],
        'traj_percent_10_15_mean': [traj_percent_10_15_mean],
        'traj_percent_15_20_mean': [traj_percent_15_20_mean],
        'traj_dist_over20_mean': [traj_dist_over20_mean],
        'traj_mean_speed_mean': [traj_mean_speed_mean],
        'month_mean_speed': [month_mean_speed],
        'traj_max_speed_max': [traj_max_speed_max],
        'month_max_speed': [month_max_speed],
        'traj_mean_acc_mean': [traj_mean_acc_mean],
        'traj_mean_dec_mean': [traj_mean_dec_mean],
        'N_stops': [N_stops],
        'stop_dist_max': [stop_dist_max],
        'stop_dist_min': [stop_dist_min],
        'stop_dist_mean': [stop_dist_mean],
        'stop_time_max': [stop_time_max],
        'stop_time_min': [stop_time_min],
        'stop_time_mean': [stop_time_mean]
    }
    # Crear el DataFrame
    df_parameters_temp = pd.DataFrame(parameters_temp)
    return df_parameters_temp
```

In [ ]:

```
# Leer el DataFrame que contiene la lista de MMSI y SHIPTYPE
df_MMSI_shiptype = pd.read_csv('MMSI_shiptype_data.csv')
total_time = 0
containing_nan = 0
too_short_csv = 0
non_associated_shiptypes = 0
tiempos_procesamiento = {'MMSI': [], 'tiempo_procesamiento': []}


# DataFrame vao para almacenar los par¡metros globales
df_cinematic_parameters = df_MMSI_shiptype.copy()
# Itera sobre cada fila del DataFrame que contiene MMSI y SHIPTYPE
for MMSI, row in tqdm(df_cinematic_parameters.head(20).iterrows(),
                      total=len(df_cinematic_parameters), desc="Processing MMSIs"):

    start_time = time.time()  # Captura el tiempo de inicio de procesamiento total de MMSI
    # Busca el archivo CSV correspondiente al MMSI actual
    csv_file = os.path.join('Dynamic_by_MMSI', f"dynamic_{row['MMSI']}.csv")

    # Verifica si el archivo CSV existe
    if os.path.exists(csv_file):

        # Lee el archivo CSV y crear el DataFrame
        df_raw_dynamic = pd.read_csv(csv_file)
        #  Verificar la longitud del DataFrame
        if len(df_raw_dynamic) >= 100:
            # Merge basado en las columnas 'mmsi' y 'MMSI' y preparacn del df
            df_merged_dynamic = pd.merge(df_raw_dynamic, df_MMSI_shiptype,
                                         left_on='mmsi', right_on='MMSI', how='inner')

            # Llamar a la funcin para preprocesar df_merged_dynamic:
            df_merged_dynamic = clean(df_merged_dynamic)

            if not(df_merged_dynamic.isnull().any().any()):
                # Crear trayectorias
                month_trajectories =
                mpd.Trajectory(
                    df_merged_dynamic,'MMSI', t='timestamp', x='lon_copy', y='lat_copy', crs=4326)

                # Separar las trayectorias
                month_separated_trajectories =
                mpd.StopSplitter(
                    month_trajectories).split(max_diameter=50, min_duration=timedelta(seconds=3600))

                # Calcular par¡metros de trayectoria
                df_traj_parameters = pd.DataFrame()
                df_traj_parameters = calcular_parametros_trayectoria(month_separated_trajectories)

                # Obtener Paradas
                month_stops =
                mpd.TrajectoryStopDetector(
                    month_trajectories).get_stop_segments(
                        min_duration=timedelta(seconds=3600), max_diameter=100)

                #Calcular parámetros de paradas
                df_stop_parameters = pd.DataFrame()
                df_stop_parameters = calcular_parametros_paradas(month_stops)

                # Calcular par¡metros globales
                df_cinematic_parameters_temp =
                calcular_parametros_cinematicos(df_traj_parameters, month_trajectories)
                df_cinematic_parameters.loc[MMSI, df_cinematic_parameters_temp.columns] =
                df_cinematic_parameters_temp.iloc[0].values

                # Asociar los parmetros globales al MMSI y SHIPTYPE actual
                df_cinematic_parameters.loc[df_cinematic_parameters.index[-1], ['MMSI', 'shiptype']] =
                [row['MMSI'], row['shiptype']]

            # En caso de que existan NaN
            else:
                containing_nan = containing_nan + 1
        # En caso de que la Longitud sea menor de 100
        else:
            too_short_csv = too_short_csv + 1
    # En caso de que no se haya asociado un shiptype al MMSI
    else:
        non_associated_shiptypes = non_associated_shiptypes + 1

    total_time = total_time + time.time() - start_time
    tiempos_procesamiento['MMSI'].append(row['MMSI'])
    tiempos_procesamiento['tiempo_procesamiento'].append(time.time() - start_time)
    df_tiempos_procesamiento = pd.DataFrame(tiempos_procesamiento)

print( "Existen", containing_nan, "MMSI cuyo df contiene Nan.")
print( "Existen", too_short_csv, "MMSI que no cumplen con la longitud requerida.")
print( "Existen", non_associated_shiptypes, "MMSI que no se han asociado a ningún shiptype.")
```

```
print( "Tiempo total en extraer parrámetros:", total_time, "segundos.")

# Guardar los parámetros globales en un archivo CSV
#df_cinematic_parameters.to_csv('cinematic_parameters.csv', index=False)
```

RESULTADOS

Iteración 0

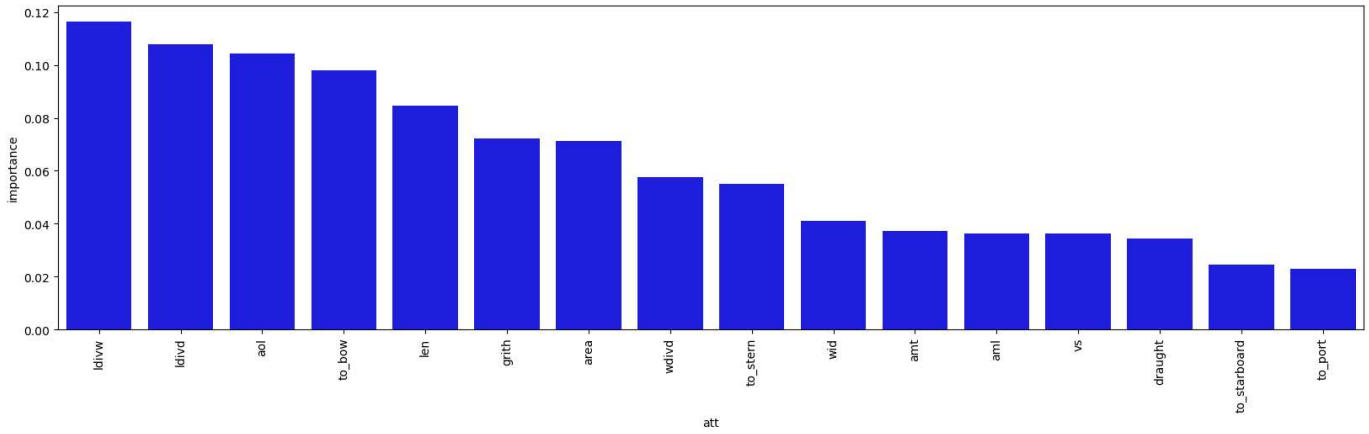
In [2]:
```
df_final_static = pd.read_csv('final_static_data_jan.csv')
df_final_static.shape
```

Out[2]:
```
(40859, 18)
```

In [3]:
```
#### Experimento 1 (Estáticos enero)
df_final_static = df_final_static.drop(columns=['MMSI'])
#División entrenamiento y test
df_train, df_test = train_test_split(df_final_static, random_state = 47)
x_train = df_train.drop(columns=['shiptype'])
y_train = df_train.shiptype
x_test = df_test.drop(columns=['shiptype'])
y_test = df_test.shiptype

#Creacion del modelo
pipe = Pipeline([('scaler', MinMaxScaler()),
                 ('classifier',RandomForestClassifier(n_estimators=100))])
model = pipe.fit(x_train, y_train)
y_pred = model.predict(x_test)

#Resultados
print(classification_report(y_test, y_pred, zero_division=0))
forest = model['classifier']
importances = forest.feature_importances_
std = np.std([tree.feature_importances_ for tree in forest.estimators_], axis=0)
fi = pd.Series(importances, index=x_train.columns).reset_index()
fi.columns = ['att', 'importance']
fi = fi.sort_values(by='importance', ascending=False).reset_index(drop=True)
g = sns.barplot(data=fi, x='att', y='importance', color='b')
g.figure.set_size_inches(20, 5)
g.set_xticklabels(g.get_xticklabels(), rotation=90)
```

```
                   precision    recall  f1-score   support

    ShipType.Cargo       0.88      0.92      0.90      5072
  ShipType.Fishing       0.81      0.81      0.81      1225
ShipType.Passenger       0.75      0.74      0.74       627
   ShipType.Tanker       0.86      0.75      0.80      2029
      ShipType.Tug       0.83      0.85      0.84      1262

          accuracy                           0.86     10215
         macro avg       0.83      0.82      0.82     10215
      weighted avg       0.86      0.86      0.85     10215
```

Out[3]:
```
[Text(0, 0, 'ldivw'),
 Text(1, 0, 'ldivd'),
 Text(2, 0, 'aol'),
 Text(3, 0, 'to_bow'),
 Text(4, 0, 'len'),
 Text(5, 0, 'grith'),
 Text(6, 0, 'area'),
 Text(7, 0, 'wdivd'),
 Text(8, 0, 'to_stern'),
 Text(9, 0, 'wid'),
 Text(10, 0, 'amt'),
 Text(11, 0, 'aml'),
 Text(12, 0, 'vs'),
 Text(13, 0, 'draught'),
 Text(14, 0, 'to_starboard'),
 Text(15, 0, 'to_port')]
```



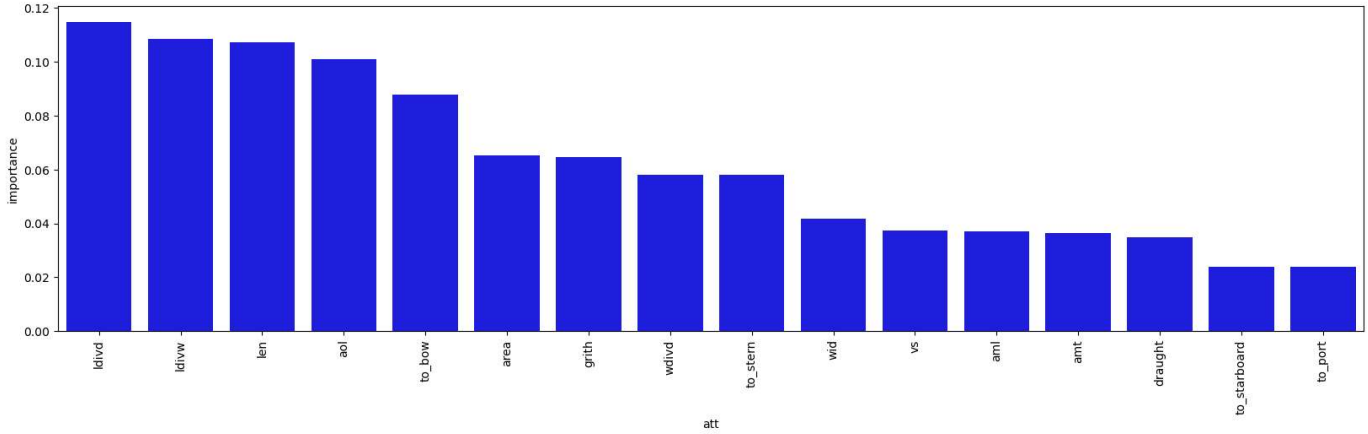
In [4]:
```
df_final_static = pd.read_csv('final_static_data_feb.csv')
df_final_static.shape
```

Out[4]:
```
(40034, 18)
```

In [5]:
```
#### Experimento 2 (Estáticos febrero)
df_final_static = df_final_static.drop(columns=['MMSI'])
#División entrenamiento y test
df_train, df_test = train_test_split(df_final_static, random_state = 47)
x_train = df_train.drop(columns=['shiptype'])
y_train = df_train.shiptype
x_test = df_test.drop(columns=['shiptype'])
y_test = df_test.shiptype

#Creacion del modelo
pipe = Pipeline([('scaler', MinMaxScaler()),
                 ('classifier',RandomForestClassifier(n_estimators=100))])
model = pipe.fit(x_train, y_train)
y_pred = model.predict(x_test)

#Resultados
print(classification_report(y_test, y_pred, zero_division=0))
forest = model['classifier']
importances = forest.feature_importances_
std = np.std([tree.feature_importances_ for tree in forest.estimators_], axis=0)
fi = pd.Series(importances, index=x_train.columns).reset_index()
fi.columns = ['att', 'importance']
fi = fi.sort_values(by='importance', ascending=False).reset_index(drop=True)
g = sns.barplot(data=fi, x='att', y='importance', color='b')
g.figure.set_size_inches(20, 5)
g.set_xticklabels(g.get_xticklabels(), rotation=90)
```

```
                   precision    recall  f1-score   support

     ShipType.Cargo       0.88      0.92      0.90      4919
   ShipType.Fishing       0.81      0.82      0.81      1253
ShipType.Passenger       0.73      0.74      0.73       607
    ShipType.Tanker       0.86      0.76      0.81      1986
       ShipType.Tug       0.83      0.82      0.83      1244

          accuracy                           0.85     10009
         macro avg       0.82      0.81      0.82     10009
      weighted avg       0.85      0.85      0.85     10009
```

Out[5]:
```
[Text(0, 0, 'ldivd'),
 Text(1, 0, 'ldivw'),
 Text(2, 0, 'len'),
 Text(3, 0, 'aol'),
 Text(4, 0, 'to_bow'),
 Text(5, 0, 'area'),
 Text(6, 0, 'grith'),
 Text(7, 0, 'wdivd'),
 Text(8, 0, 'to_stern'),
 Text(9, 0, 'wid'),
 Text(10, 0, 'vs'),
 Text(11, 0, 'aml'),
 Text(12, 0, 'amt'),
 Text(13, 0, 'draught'),
 Text(14, 0, 'to_starboard'),
 Text(15, 0, 'to_port')]
```



Iteración 1

Enero

In [6]:
```
df_final_static = pd.read_csv('final_static_data_jan.csv')
df_final_static.shape
```

Out[6]:
```
(40859, 18)
```

In [7]:
```
df_cinematic_parameters = pd.read_csv('cinematic_parameters_1_jan.csv')
df_cinematic_parameters.shape
```

Out[7]:
```
(40859, 69)
```

In [8]:
```
df_cinematic_parameters = df_cinematic_parameters.dropna()
df_cinematic_parameters.shape
```

Out[8]:
```
(31300, 69)
```

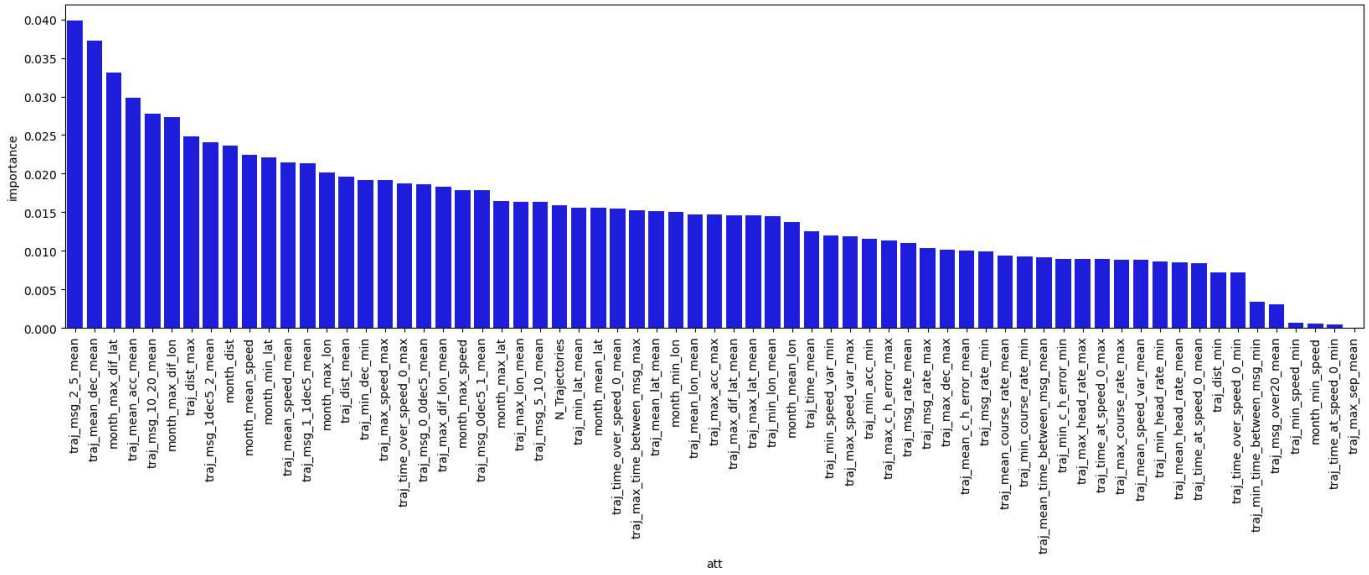
In [9]:
```
df_global_parameters = pd.merge(df_cinematic_parameters, df_final_static, on='MMSI', how='inner')
df_global_parameters.rename(columns={'shiptype_x': 'shiptype'}, inplace=True)
df_global_parameters.drop(columns=['shiptype_y','MMSI'], inplace=True)
df_global_parameters.shape
```

Out[9]:
```
(31300, 84)
```

In [10]:
```
#### Experimento 3
df_cinematic_parameters = df_cinematic_parameters.drop(columns=['MMSI'])
#División entrenamiento y test
df_train, df_test = train_test_split(df_cinematic_parameters, random_state = 47)
x_train = df_train.drop(columns=['shiptype'])
y_train = df_train.shiptype
x_test = df_test.drop(columns=['shiptype'])
y_test = df_test.shiptype

#Creacion del modelo
pipe = Pipeline([('scaler', MinMaxScaler()),
                 ('classifier',RandomForestClassifier(n_estimators=100))])
model = pipe.fit(x_train, y_train)
y_pred = model.predict(x_test)

#Resultados
print(classification_report(y_test, y_pred, zero_division=0))
forest = model['classifier']
importances = forest.feature_importances_
std = np.std([tree.feature_importances_ for tree in forest.estimators_], axis=0)
fi = pd.Series(importances, index=x_train.columns).reset_index()
fi.columns = ['att', 'importance']
fi = fi.sort_values(by='importance', ascending=False).reset_index(drop=True)
g = sns.barplot(data=fi, x='att', y='importance', color='b')
g.figure.set_size_inches(20, 5)
g.set_xticklabels(g.get_xticklabels(), rotation=90)
```

	precision	recall	f1-score	support
ShipType.Cargo	0.76	0.94	0.84	3890
ShipType.Fishing	0.87	0.84	0.86	934
ShipType.Passenger	0.89	0.70	0.78	485
ShipType.Tanker	0.79	0.39	0.52	1516
ShipType.Tug	0.83	0.83	0.83	1000
accuracy			0.79	7825
macro avg	0.83	0.74	0.77	7825
weighted avg	0.80	0.79	0.77	7825

Out[10]:
```
[Text(0, 0, 'traj_msg_2_5_mean'),
 Text(1, 0, 'traj_mean_dec_mean'),
 Text(2, 0, 'month_max_dif_lat'),
 Text(3, 0, 'traj_mean_acc_mean'),
 Text(4, 0, 'traj_msg_10_20_mean'),
 Text(5, 0, 'month_max_dif_lon'),
 Text(6, 0, 'traj_dist_max'),
 Text(7, 0, 'traj_msg_1dec5_2_mean'),
 Text(8, 0, 'month_dist'),
 Text(9, 0, 'month_mean_speed'),
 Text(10, 0, 'month_min_lat'),
 Text(11, 0, 'traj_mean_speed_mean'),
 Text(12, 0, 'traj_msg_1_1dec5_mean'),
 Text(13, 0, 'month_max_lon'),
 Text(14, 0, 'traj_dist_mean'),
 Text(15, 0, 'traj_min_dec_min'),
 Text(16, 0, 'traj_max_speed_max'),
 Text(17, 0, 'traj_time_over_speed_0_max'),
 Text(18, 0, 'traj_msg_0_0dec5_mean'),
 Text(19, 0, 'traj_max_dif_lon_mean'),
 Text(20, 0, 'month_max_speed'),
 Text(21, 0, 'traj_msg_0dec5_1_mean'),
 Text(22, 0, 'month_max_lat'),
 Text(23, 0, 'traj_max_lon_mean'),
 Text(24, 0, 'traj_msg_5_10_mean'),
 Text(25, 0, 'N_Trajectories'),
 Text(26, 0, 'traj_min_lat_mean'),
 Text(27, 0, 'month_mean_lat'),
 Text(28, 0, 'traj_time_over_speed_0_mean'),
 Text(29, 0, 'traj_max_time_between_msg_max'),
 Text(30, 0, 'traj_mean_lat_mean'),
 Text(31, 0, 'month_min_lon'),
 Text(32, 0, 'traj_mean_lon_mean'),
 Text(33, 0, 'traj_max_acc_max'),
 Text(34, 0, 'traj_max_dif_lat_mean'),
 Text(35, 0, 'traj_max_lat_mean'),
 Text(36, 0, 'traj_min_lon_mean'),
 Text(37, 0, 'month_mean_lon'),
 Text(38, 0, 'traj_time_mean'),
 Text(39, 0, 'traj_min_speed_var_min'),
 Text(40, 0, 'traj_max_speed_var_max'),
 Text(41, 0, 'traj_min_acc_min'),
 Text(42, 0, 'traj_max_c_h_error_max'),
 Text(43, 0, 'traj_msg_rate_mean'),
 Text(44, 0, 'traj_msg_rate_max'),
 Text(45, 0, 'traj_max_dec_max'),
 Text(46, 0, 'traj_mean_c_h_error_mean'),
 Text(47, 0, 'traj_msg_rate_min'),
 Text(48, 0, 'traj_mean_course_rate_mean'),
 Text(49, 0, 'traj_min_course_rate_min'),
 Text(50, 0, 'traj_mean_time_between_msg_mean'),
 Text(51, 0, 'traj_min_c_h_error_min'),
 Text(52, 0, 'traj_max_head_rate_max'),
 Text(53, 0, 'traj_time_at_speed_0_max'),
 Text(54, 0, 'traj_max_course_rate_max'),
 Text(55, 0, 'traj_mean_speed_var_mean'),
 Text(56, 0, 'traj_min_head_rate_min'),
 Text(57, 0, 'traj_mean_head_rate_mean'),
 Text(58, 0, 'traj_time_at_speed_0_mean'),
 Text(59, 0, 'traj_dist_min'),
 Text(60, 0, 'traj_time_over_speed_0_min'),
 Text(61, 0, 'traj_min_time_between_msg_min'),
 Text(62, 0, 'traj_msg_over20_mean'),
 Text(63, 0, 'traj_min_speed_min'),
 Text(64, 0, 'month_min_speed'),
 Text(65, 0, 'traj_time_at_speed_0_min'),
 Text(66, 0, 'traj_max_sep_mean')]
```

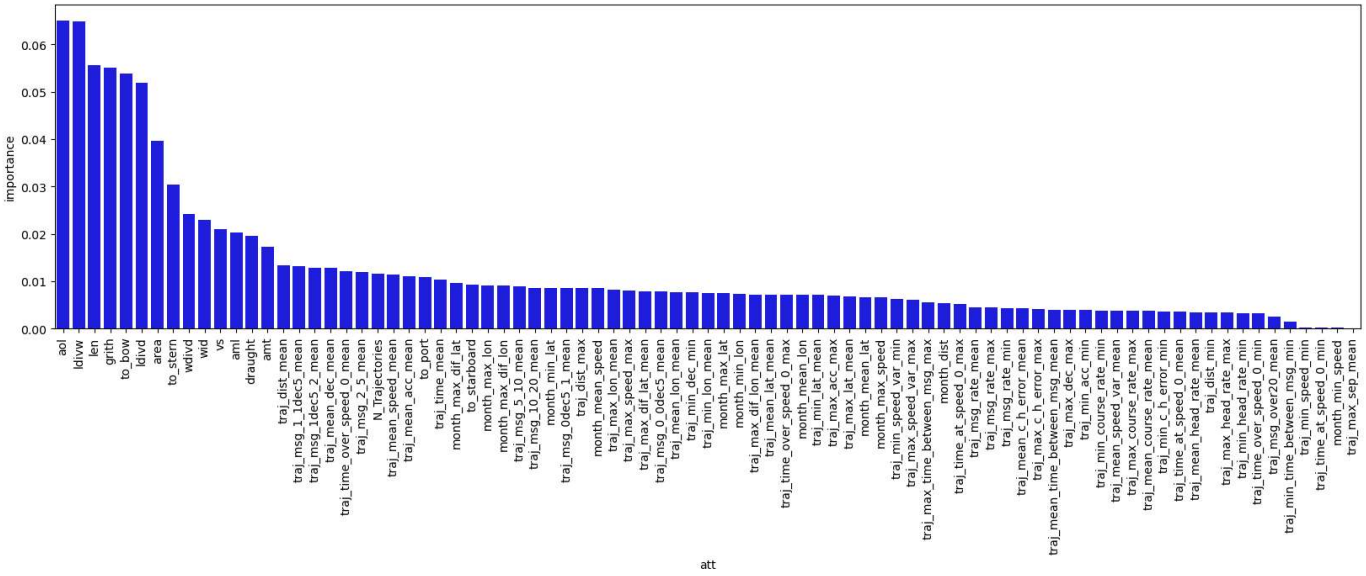


In [11]:
```
#### Experimento 4
#División entrenamiento y test
df_train, df_test = train_test_split(df_global_parameters, random_state = 47)
x_train = df_train.drop(columns=['shiptype'])
y_train = df_train.shiptype
x_test = df_test.drop(columns=['shiptype'])
y_test = df_test.shiptype

#Creacion del modelo
pipe = Pipeline([('scaler', MinMaxScaler()),
                 ('classifier',RandomForestClassifier(n_estimators=100))])
model = pipe.fit(x_train, y_train)
y_pred = model.predict(x_test)

#Resultados
print(classification_report(y_test, y_pred, zero_division=0))
forest = model['classifier']
importances = forest.feature_importances_
std = np.std([tree.feature_importances_ for tree in forest.estimators_], axis=0)
fi = pd.Series(importances, index=x_train.columns).reset_index()
fi.columns = ['att', 'importance']
fi = fi.sort_values(by='importance', ascending=False).reset_index(drop=True)
g = sns.barplot(data=fi, x='att', y='importance', color='b')
g.figure.set_size_inches(20, 5)
g.set_xticklabels(g.get_xticklabels(), rotation=90)
```

```
                   precision    recall  f1-score   support

    ShipType.Cargo       0.90      0.96      0.93      3890
  ShipType.Fishing       0.93      0.94      0.94       934
ShipType.Passenger       0.89      0.82      0.86       485
   ShipType.Tanker       0.92      0.78      0.84      1516
      ShipType.Tug       0.92      0.92      0.92      1000

         accuracy                           0.91      7825
        macro avg       0.91      0.89      0.90      7825
     weighted avg       0.91      0.91      0.91      7825
```

Out[11]:
```
[Text(0, 0, 'aol'),
 Text(1, 0, 'ldivw'),
 Text(2, 0, 'len'),
 Text(3, 0, 'grith'),
 Text(4, 0, 'to_bow'),
 Text(5, 0, 'ldivd'),
 Text(6, 0, 'area'),
 Text(7, 0, 'to_stern'),
 Text(8, 0, 'wdivd'),
 Text(9, 0, 'wid'),
 Text(10, 0, 'vs'),
 Text(11, 0, 'aml'),
 Text(12, 0, 'draught'),
 Text(13, 0, 'amt'),
 Text(14, 0, 'traj_dist_mean'),
 Text(15, 0, 'traj_msg_1_1dec5_mean'),
 Text(16, 0, 'traj_msg_1dec5_2_mean'),
 Text(17, 0, 'traj_mean_dec_mean'),
 Text(18, 0, 'traj_time_over_speed_0_mean'),
 Text(19, 0, 'traj_msg_2_5_mean'),
 Text(20, 0, 'N_Trajectories'),
 Text(21, 0, 'traj_mean_speed_mean'),
 Text(22, 0, 'traj_mean_acc_mean'),
 Text(23, 0, 'to_port'),
 Text(24, 0, 'traj_time_mean'),
 Text(25, 0, 'month_max_dif_lat'),
 Text(26, 0, 'to_starboard'),
 Text(27, 0, 'month_max_lon'),
 Text(28, 0, 'month_max_dif_lon'),
 Text(29, 0, 'traj_msg_5_10_mean'),
 Text(30, 0, 'traj_msg_10_20_mean'),
 Text(31, 0, 'month_min_lat'),
 Text(32, 0, 'traj_msg_0dec5_1_mean'),
 Text(33, 0, 'traj_dist_max'),
 Text(34, 0, 'month_mean_speed'),
 Text(35, 0, 'traj_max_lon_mean'),
 Text(36, 0, 'traj_max_speed_max'),
 Text(37, 0, 'traj_max_dif_lat_mean'),
 Text(38, 0, 'traj_msg_0_0dec5_mean'),
 Text(39, 0, 'traj_mean_lon_mean'),
 Text(40, 0, 'traj_min_dec_min'),
 Text(41, 0, 'traj_min_lon_mean'),
 Text(42, 0, 'month_max_lat'),
 Text(43, 0, 'month_min_lon'),
 Text(44, 0, 'traj_max_dif_lon_mean'),
 Text(45, 0, 'traj_mean_lat_mean'),
 Text(46, 0, 'traj_time_over_speed_0_max'),
 Text(47, 0, 'month_mean_lon'),
 Text(48, 0, 'traj_min_lat_mean'),
 Text(49, 0, 'traj_max_acc_max'),
 Text(50, 0, 'traj_max_lat_mean'),
 Text(51, 0, 'month_mean_lat'),
 Text(52, 0, 'month_max_speed'),
 Text(53, 0, 'traj_min_speed_var_min'),
 Text(54, 0, 'traj_max_speed_var_max'),
 Text(55, 0, 'traj_max_time_between_msg_max'),
 Text(56, 0, 'month_dist'),
 Text(57, 0, 'traj_time_at_speed_0_max'),
 Text(58, 0, 'traj_msg_rate_mean'),
 Text(59, 0, 'traj_msg_rate_max'),
 Text(60, 0, 'traj_msg_rate_min'),
 Text(61, 0, 'traj_mean_c_h_error_mean'),
 Text(62, 0, 'traj_max_c_h_error_max'),
 Text(63, 0, 'traj_mean_time_between_msg_mean'),
 Text(64, 0, 'traj_max_dec_max'),
 Text(65, 0, 'traj_min_acc_min'),
 Text(66, 0, 'traj_min_course_rate_min'),
 Text(67, 0, 'traj_mean_speed_var_mean'),
 Text(68, 0, 'traj_max_course_rate_max'),
 Text(69, 0, 'traj_mean_course_rate_mean'),
 Text(70, 0, 'traj_min_c_h_error_min'),
 Text(71, 0, 'traj_time_at_speed_0_mean'),
 Text(72, 0, 'traj_mean_head_rate_mean'),
 Text(73, 0, 'traj_dist_min'),
 Text(74, 0, 'traj_max_head_rate_max'),
 Text(75, 0, 'traj_min_head_rate_min'),
```

```
Text(76, 0, 'traj_time_over_speed_0_min'),
Text(77, 0, 'traj_msg_over20_mean'),
Text(78, 0, 'traj_min_time_between_msg_min'),
Text(79, 0, 'traj_min_speed_min'),
Text(80, 0, 'traj_time_at_speed_0_min'),
Text(81, 0, 'month_min_speed'),
Text(82, 0, 'traj_max_sep_mean')]
```



Febrero

In [12]:
```
df_final_static = pd.read_csv('final_static_data_feb.csv')
df_final_static.shape
```

Out[12]:
```
(40034, 18)
```

In [13]:
```
df_cinematic_parameters = pd.read_csv('cinematic_parameters_1_feb.csv')
df_cinematic_parameters.shape
```

Out[13]:
```
(40034, 69)
```

In [14]:
```
df_cinematic_parameters = df_cinematic_parameters.dropna()
df_cinematic_parameters.shape
```

Out[14]:
```
(30462, 69)
```

In [15]:
```
df_global_parameters = pd.merge(df_cinematic_parameters, df_final_static, on='MMSI', how='inner')
df_global_parameters.rename(columns={'shiptype_x': 'shiptype'}, inplace=True)
df_global_parameters.drop(columns=['shiptype_y','MMSI'], inplace=True)
df_global_parameters.shape
```

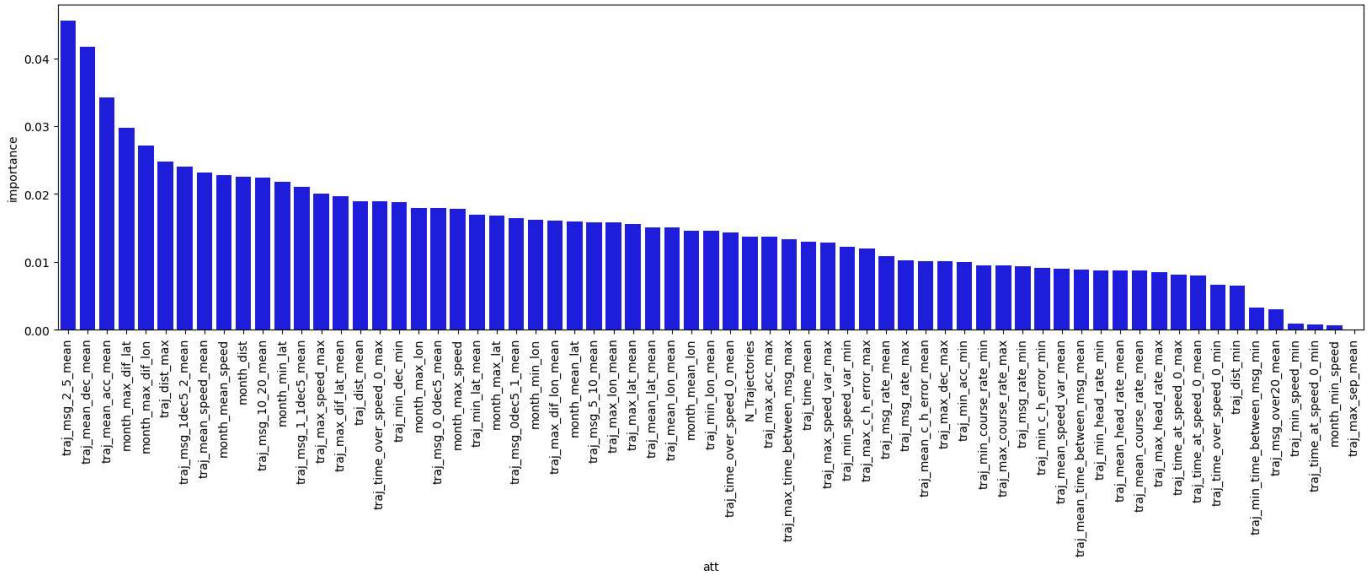
Out[15]:
```
(30462, 84)
```

In [16]:
```
#### Experimento 5
df_cinematic_parameters = df_cinematic_parameters.drop(columns=['MMSI'])
#División entrenamiento y test
df_train, df_test = train_test_split(df_cinematic_parameters, random_state = 47)
x_train = df_train.drop(columns=['shiptype'])
y_train = df_train.shiptype
x_test = df_test.drop(columns=['shiptype'])
y_test = df_test.shiptype

#Creacion del modelo
pipe = Pipeline([('scaler', MinMaxScaler()),
                 ('classifier',RandomForestClassifier(n_estimators=100))])
model = pipe.fit(x_train, y_train)
y_pred = model.predict(x_test)

#Resultados
print(classification_report(y_test, y_pred, zero_division=0))
forest = model['classifier']
importances = forest.feature_importances_
std = np.std([tree.feature_importances_ for tree in forest.estimators_], axis=0)
fi = pd.Series(importances, index=x_train.columns).reset_index()
```

```
fi.columns = ['att', 'importance']
fi = fi.sort_values(by='importance', ascending=False).reset_index(drop=True)
g = sns.barplot(data=fi, x='att', y='importance', color='b')
g.figure.set_size_inches(20, 5)
g.set_xticklabels(g.get_xticklabels(), rotation=90)
```

	precision	recall	f1-score	support
ShipType.Cargo	0.77	0.93	0.84	3820
ShipType.Fishing	0.89	0.87	0.88	948
ShipType.Passenger	0.89	0.68	0.77	437
ShipType.Tanker	0.77	0.41	0.54	1452
ShipType.Tug	0.84	0.84	0.84	959
accuracy			0.80	7616
macro avg	0.83	0.74	0.77	7616
weighted avg	0.80	0.80	0.78	7616

Out[16]:
```
[Text(0, 0, 'traj_msg_2_5_mean'),
 Text(1, 0, 'traj_mean_dec_mean'),
 Text(2, 0, 'traj_mean_acc_mean'),
 Text(3, 0, 'month_max_dif_lat'),
 Text(4, 0, 'month_max_dif_lon'),
 Text(5, 0, 'traj_dist_max'),
 Text(6, 0, 'traj_msg_1dec5_2_mean'),
 Text(7, 0, 'traj_mean_speed_mean'),
 Text(8, 0, 'month_mean_speed'),
 Text(9, 0, 'month_dist'),
 Text(10, 0, 'traj_msg_10_20_mean'),
 Text(11, 0, 'month_min_lat'),
 Text(12, 0, 'traj_msg_1_1dec5_mean'),
 Text(13, 0, 'traj_max_speed_max'),
 Text(14, 0, 'traj_max_dif_lat_mean'),
 Text(15, 0, 'traj_dist_mean'),
 Text(16, 0, 'traj_time_over_speed_0_max'),
 Text(17, 0, 'traj_min_dec_min'),
 Text(18, 0, 'month_max_lon'),
 Text(19, 0, 'traj_msg_0_0dec5_mean'),
 Text(20, 0, 'month_max_speed'),
 Text(21, 0, 'traj_min_lat_mean'),
 Text(22, 0, 'month_max_lat'),
 Text(23, 0, 'traj_msg_0dec5_1_mean'),
 Text(24, 0, 'month_min_lon'),
 Text(25, 0, 'traj_max_dif_lon_mean'),
 Text(26, 0, 'month_mean_lat'),
 Text(27, 0, 'traj_msg_5_10_mean'),
 Text(28, 0, 'traj_max_lon_mean'),
 Text(29, 0, 'traj_max_lat_mean'),
 Text(30, 0, 'traj_mean_lat_mean'),
 Text(31, 0, 'traj_mean_lon_mean'),
 Text(32, 0, 'month_mean_lon'),
 Text(33, 0, 'traj_min_lon_mean'),
 Text(34, 0, 'traj_time_over_speed_0_mean'),
 Text(35, 0, 'N_Trajectories'),
 Text(36, 0, 'traj_max_acc_max'),
 Text(37, 0, 'traj_max_time_between_msg_max'),
 Text(38, 0, 'traj_time_mean'),
 Text(39, 0, 'traj_max_speed_var_max'),
 Text(40, 0, 'traj_min_speed_var_min'),
 Text(41, 0, 'traj_max_c_h_error_max'),
 Text(42, 0, 'traj_msg_rate_mean'),
 Text(43, 0, 'traj_msg_rate_max'),
 Text(44, 0, 'traj_mean_c_h_error_mean'),
 Text(45, 0, 'traj_max_dec_max'),
 Text(46, 0, 'traj_min_acc_min'),
 Text(47, 0, 'traj_min_course_rate_min'),
 Text(48, 0, 'traj_max_course_rate_max'),
 Text(49, 0, 'traj_msg_rate_min'),
 Text(50, 0, 'traj_min_c_h_error_min'),
 Text(51, 0, 'traj_mean_speed_var_mean'),
 Text(52, 0, 'traj_mean_time_between_msg_mean'),
 Text(53, 0, 'traj_min_head_rate_min'),
 Text(54, 0, 'traj_mean_head_rate_mean'),
 Text(55, 0, 'traj_mean_course_rate_mean'),
 Text(56, 0, 'traj_max_head_rate_max'),
 Text(57, 0, 'traj_time_at_speed_0_max'),
 Text(58, 0, 'traj_time_at_speed_0_mean'),
 Text(59, 0, 'traj_time_over_speed_0_min'),
 Text(60, 0, 'traj_dist_min'),
 Text(61, 0, 'traj_min_time_between_msg_min'),
 Text(62, 0, 'traj_msg_over20_mean'),
 Text(63, 0, 'traj_min_speed_min'),
 Text(64, 0, 'traj_time_at_speed_0_min'),
 Text(65, 0, 'month_min_speed'),
 Text(66, 0, 'traj_max_sep_mean')]
```

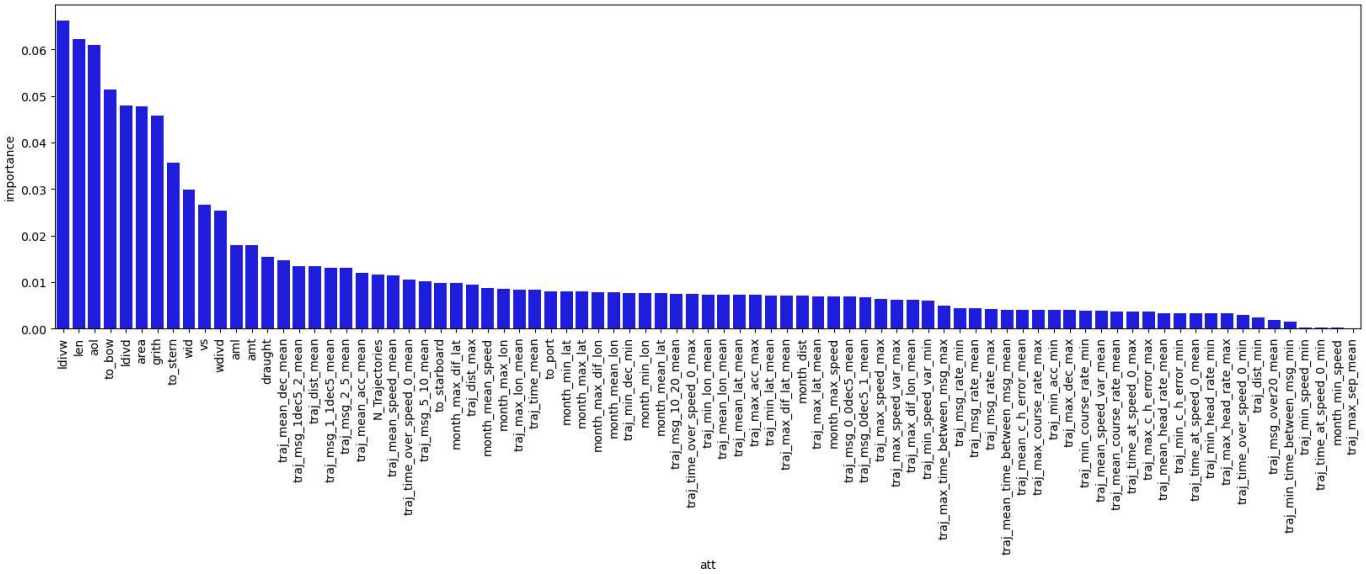


In [17]:
```
#### Experimento 6
#División entrenamiento y test
df_train, df_test = train_test_split(df_global_parameters, random_state = 47)
x_train = df_train.drop(columns=['shiptype'])
y_train = df_train.shiptype
x_test = df_test.drop(columns=['shiptype'])
y_test = df_test.shiptype

#Creacion del modelo
pipe = Pipeline([('scaler', MinMaxScaler()),
                 ('classifier',RandomForestClassifier(n_estimators=100))])
model = pipe.fit(x_train, y_train)
y_pred = model.predict(x_test)

#Resultados
print(classification_report(y_test, y_pred, zero_division=0))
forest = model['classifier']
importances = forest.feature_importances_
std = np.std([tree.feature_importances_ for tree in forest.estimators_], axis=0)
fi = pd.Series(importances, index=x_train.columns).reset_index()
fi.columns = ['att', 'importance']
fi = fi.sort_values(by='importance', ascending=False).reset_index(drop=True)
g = sns.barplot(data=fi, x='att', y='importance', color='b')
g.figure.set_size_inches(20, 5)
g.set_xticklabels(g.get_xticklabels(), rotation=90)
```

```
                   precision    recall  f1-score   support

    ShipType.Cargo       0.90      0.96      0.93      3820
  ShipType.Fishing       0.92      0.96      0.94       948
ShipType.Passenger       0.89      0.84      0.86       437
   ShipType.Tanker       0.94      0.78      0.85      1452
      ShipType.Tug       0.92      0.92      0.92       959

         accuracy                           0.91      7616
        macro avg       0.91      0.89      0.90      7616
     weighted avg       0.91      0.91      0.91      7616
```

Out[17]:
```
[Text(0, 0, 'ldivw'),
 Text(1, 0, 'len'),
 Text(2, 0, 'aol'),
 Text(3, 0, 'to_bow'),
 Text(4, 0, 'ldivd'),
 Text(5, 0, 'area'),
 Text(6, 0, 'grith'),
 Text(7, 0, 'to_stern'),
 Text(8, 0, 'wid'),
 Text(9, 0, 'vs'),
 Text(10, 0, 'wdivd'),
 Text(11, 0, 'aml'),
 Text(12, 0, 'amt'),
 Text(13, 0, 'draught'),
 Text(14, 0, 'traj_mean_dec_mean'),
 Text(15, 0, 'traj_msg_1dec5_2_mean'),
 Text(16, 0, 'traj_dist_mean'),
 Text(17, 0, 'traj_msg_1_1dec5_mean'),
 Text(18, 0, 'traj_msg_2_5_mean'),
 Text(19, 0, 'traj_mean_acc_mean'),
 Text(20, 0, 'N_Trajectories'),
 Text(21, 0, 'traj_mean_speed_mean'),
 Text(22, 0, 'traj_time_over_speed_0_mean'),
 Text(23, 0, 'traj_msg_5_10_mean'),
 Text(24, 0, 'to_starboard'),
 Text(25, 0, 'month_max_dif_lat'),
 Text(26, 0, 'traj_dist_max'),
 Text(27, 0, 'month_mean_speed'),
 Text(28, 0, 'month_max_lon'),
 Text(29, 0, 'traj_max_lon_mean'),
 Text(30, 0, 'traj_time_mean'),
 Text(31, 0, 'to_port'),
 Text(32, 0, 'month_min_lat'),
 Text(33, 0, 'month_max_lat'),
 Text(34, 0, 'month_max_dif_lon'),
 Text(35, 0, 'month_mean_lon'),
 Text(36, 0, 'traj_min_dec_min'),
 Text(37, 0, 'month_min_lon'),
 Text(38, 0, 'month_mean_lat'),
 Text(39, 0, 'traj_msg_10_20_mean'),
 Text(40, 0, 'traj_time_over_speed_0_max'),
 Text(41, 0, 'traj_min_lon_mean'),
 Text(42, 0, 'traj_mean_lon_mean'),
 Text(43, 0, 'traj_mean_lat_mean'),
 Text(44, 0, 'traj_max_acc_max'),
 Text(45, 0, 'traj_min_lat_mean'),
 Text(46, 0, 'traj_max_dif_lat_mean'),
 Text(47, 0, 'month_dist'),
 Text(48, 0, 'traj_max_lat_mean'),
 Text(49, 0, 'month_max_speed'),
 Text(50, 0, 'traj_msg_0_0dec5_mean'),
 Text(51, 0, 'traj_msg_0dec5_1_mean'),
 Text(52, 0, 'traj_max_speed_max'),
 Text(53, 0, 'traj_max_speed_var_max'),
 Text(54, 0, 'traj_max_dif_lon_mean'),
 Text(55, 0, 'traj_min_speed_var_min'),
 Text(56, 0, 'traj_max_time_between_msg_max'),
 Text(57, 0, 'traj_msg_rate_min'),
 Text(58, 0, 'traj_msg_rate_mean'),
 Text(59, 0, 'traj_msg_rate_max'),
 Text(60, 0, 'traj_mean_time_between_msg_mean'),
 Text(61, 0, 'traj_mean_c_h_error_mean'),
 Text(62, 0, 'traj_max_course_rate_max'),
 Text(63, 0, 'traj_min_acc_min'),
 Text(64, 0, 'traj_max_dec_max'),
 Text(65, 0, 'traj_min_course_rate_min'),
 Text(66, 0, 'traj_mean_speed_var_mean'),
 Text(67, 0, 'traj_mean_course_rate_mean'),
 Text(68, 0, 'traj_time_at_speed_0_max'),
 Text(69, 0, 'traj_max_c_h_error_max'),
 Text(70, 0, 'traj_mean_head_rate_mean'),
 Text(71, 0, 'traj_min_c_h_error_min'),
 Text(72, 0, 'traj_time_at_speed_0_mean'),
 Text(73, 0, 'traj_min_head_rate_min'),
 Text(74, 0, 'traj_max_head_rate_max'),
 Text(75, 0, 'traj_time_over_speed_0_min'),
```

```
Text(76, 0, 'traj_dist_min'),
Text(77, 0, 'traj_msg_over20_mean'),
Text(78, 0, 'traj_min_time_between_msg_min'),
Text(79, 0, 'traj_min_speed_min'),
Text(80, 0, 'traj_time_at_speed_0_min'),
Text(81, 0, 'month_min_speed'),
Text(82, 0, 'traj_max_sep_mean')]
```



Iteración 2

In [18]:
```
df_final_static = pd.read_csv('final_static_data_jan.csv')
df_final_static.shape
```

Out[18]:
```
(40859, 18)
```

In [19]:
```
df_cinematic_parameters = pd.read_csv('cinematic_parameters_2_jan.csv')
df_cinematic_parameters.shape
```

Out[19]:
```
(40859, 29)
```

In [20]:
```
df_cinematic_parameters = df_cinematic_parameters.dropna()
df_cinematic_parameters.shape
```

Out[20]:
```
(35725, 29)
```

In [21]:
```
df_global_parameters = pd.merge(df_cinematic_parameters, df_final_static, on='MMSI', how='inner')
df_global_parameters.rename(columns={'shiptype_x': 'shiptype'}, inplace=True)
df_global_parameters.drop(columns=['shiptype_y','MMSI'], inplace=True)
df_global_parameters.shape
```

Out[21]:
```
(35725, 44)
```

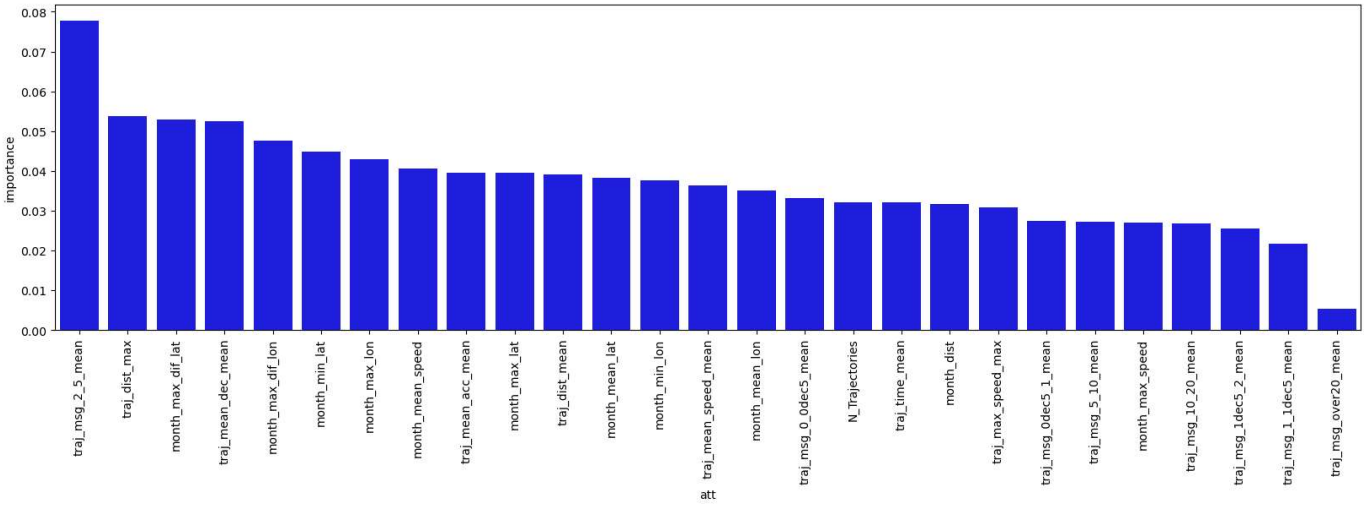
In [22]:
```
#### Experimento 7
df_cinematic_parameters = df_cinematic_parameters.drop(columns=['MMSI'])
#División entrenamiento y test
df_train, df_test = train_test_split(df_cinematic_parameters, random_state = 47)
x_train = df_train.drop(columns=['shiptype'])
y_train = df_train.shiptype
x_test = df_test.drop(columns=['shiptype'])
y_test = df_test.shiptype

#Creacion del modelo
pipe = Pipeline([('scaler', MinMaxScaler()),
                 ('classifier',RandomForestClassifier(n_estimators=100))])
model = pipe.fit(x_train, y_train)
y_pred = model.predict(x_test)

#Resultados
print(classification_report(y_test, y_pred, zero_division=0))
forest = model['classifier']
importances = forest.feature_importances_
std = np.std([tree.feature_importances_ for tree in forest.estimators_], axis=0)
fi = pd.Series(importances, index=x_train.columns).reset_index()
```

```
fi.columns = ['att', 'importance']
fi = fi.sort_values(by='importance', ascending=False).reset_index(drop=True)
g = sns.barplot(data=fi, x='att', y='importance', color='b')
g.figure.set_size_inches(20, 5)
g.set_xticklabels(g.get_xticklabels(), rotation=90)
```

```
                    precision    recall  f1-score   support

     ShipType.Cargo       0.76      0.92      0.83      4556
   ShipType.Fishing       0.86      0.82      0.84       962
ShipType.Passenger       0.82      0.68      0.75       519
    ShipType.Tanker       0.74      0.38      0.51      1738
       ShipType.Tug       0.82      0.82      0.82      1157

          accuracy                           0.78      8932
         macro avg       0.80      0.72      0.75      8932
      weighted avg       0.78      0.78      0.76      8932
```

Out[22]:

```
[Text(0, 0, 'traj_msg_2_5_mean'),
 Text(1, 0, 'traj_dist_max'),
 Text(2, 0, 'month_max_dif_lat'),
 Text(3, 0, 'traj_mean_dec_mean'),
 Text(4, 0, 'month_max_dif_lon'),
 Text(5, 0, 'month_min_lat'),
 Text(6, 0, 'month_max_lon'),
 Text(7, 0, 'month_mean_speed'),
 Text(8, 0, 'traj_mean_acc_mean'),
 Text(9, 0, 'month_max_lat'),
 Text(10, 0, 'traj_dist_mean'),
 Text(11, 0, 'month_mean_lat'),
 Text(12, 0, 'month_min_lon'),
 Text(13, 0, 'traj_mean_speed_mean'),
 Text(14, 0, 'month_mean_lon'),
 Text(15, 0, 'traj_msg_0_0dec5_mean'),
 Text(16, 0, 'N_Trajectories'),
 Text(17, 0, 'traj_time_mean'),
 Text(18, 0, 'month_dist'),
 Text(19, 0, 'traj_max_speed_max'),
 Text(20, 0, 'traj_msg_0dec5_1_mean'),
 Text(21, 0, 'traj_msg_5_10_mean'),
 Text(22, 0, 'month_max_speed'),
 Text(23, 0, 'traj_msg_10_20_mean'),
 Text(24, 0, 'traj_msg_1dec5_2_mean'),
 Text(25, 0, 'traj_msg_1_1dec5_mean'),
 Text(26, 0, 'traj_msg_over20_mean')]
```



In [23]:

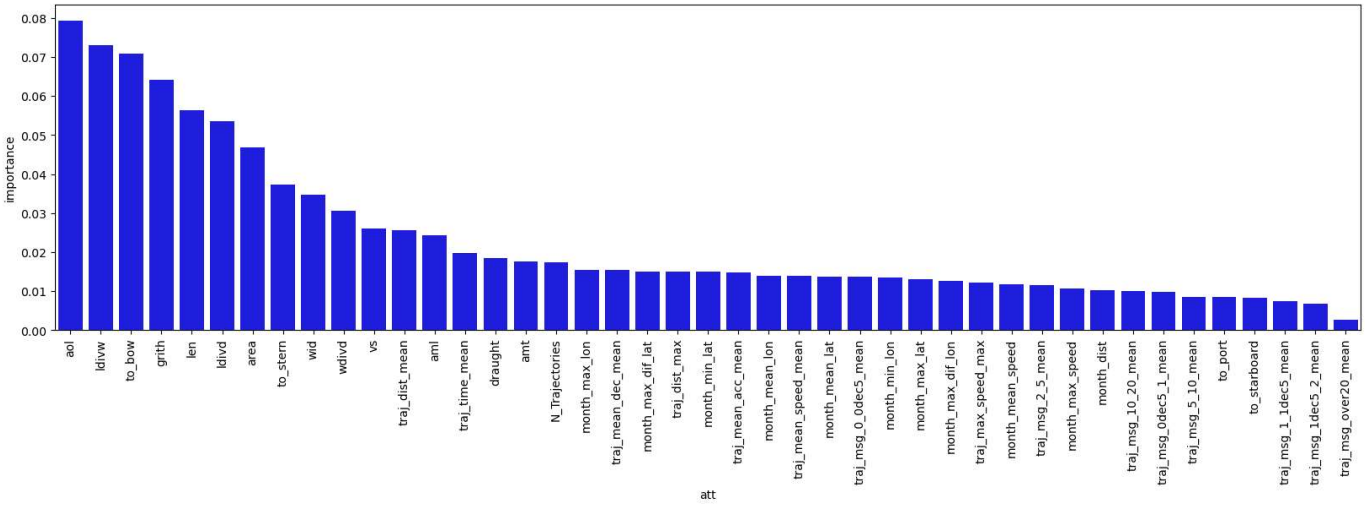
```
#### Experimento 8
#División entrenamiento y test
df_train, df_test = train_test_split(df_global_parameters, random_state = 47)
x_train = df_train.drop(columns=['shiptype'])
y_train = df_train.shiptype
x_test = df_test.drop(columns=['shiptype'])
y_test = df_test.shiptype

#Creacion del modelo
pipe = Pipeline([('scaler', MinMaxScaler()),
                 ('classifier',RandomForestClassifier(n_estimators=100))])
model = pipe.fit(x_train, y_train)
y_pred = model.predict(x_test)
```

```
#Resultados
print(classification_report(y_test, y_pred, zero_division=0))
forest = model['classifier']
importances = forest.feature_importances_
std = np.std([tree.feature_importances_ for tree in forest.estimators_], axis=0)
fi = pd.Series(importances, index=x_train.columns).reset_index()
fi.columns = ['att', 'importance']
fi = fi.sort_values(by='importance', ascending=False).reset_index(drop=True)
g = sns.barplot(data=fi, x='att', y='importance', color='b')
g.figure.set_size_inches(20, 5)
g.set_xticklabels(g.get_xticklabels(), rotation=90)
```

```
                    precision    recall  f1-score   support

    ShipType.Cargo       0.90      0.95      0.93      4556
  ShipType.Fishing       0.91      0.92      0.91       962
ShipType.Passenger       0.87      0.85      0.86       519
   ShipType.Tanker       0.91      0.77      0.84      1738
      ShipType.Tug       0.92      0.93      0.92      1157

         accuracy                           0.90      8932
        macro avg       0.90      0.88      0.89      8932
     weighted avg       0.90      0.90      0.90      8932
```

Out[23]:

```
[Text(0, 0, 'aol'),
 Text(1, 0, 'ldivw'),
 Text(2, 0, 'to_bow'),
 Text(3, 0, 'grith'),
 Text(4, 0, 'len'),
 Text(5, 0, 'ldivd'),
 Text(6, 0, 'area'),
 Text(7, 0, 'to_stern'),
 Text(8, 0, 'wid'),
 Text(9, 0, 'wdivd'),
 Text(10, 0, 'vs'),
 Text(11, 0, 'traj_dist_mean'),
 Text(12, 0, 'aml'),
 Text(13, 0, 'traj_time_mean'),
 Text(14, 0, 'draught'),
 Text(15, 0, 'amt'),
 Text(16, 0, 'N_Trajectories'),
 Text(17, 0, 'month_max_lon'),
 Text(18, 0, 'traj_mean_dec_mean'),
 Text(19, 0, 'month_max_dif_lat'),
 Text(20, 0, 'traj_dist_max'),
 Text(21, 0, 'month_min_lat'),
 Text(22, 0, 'traj_mean_acc_mean'),
 Text(23, 0, 'month_mean_lon'),
 Text(24, 0, 'traj_mean_speed_mean'),
 Text(25, 0, 'month_mean_lat'),
 Text(26, 0, 'traj_msg_0_0dec5_mean'),
 Text(27, 0, 'month_min_lon'),
 Text(28, 0, 'month_max_lat'),
 Text(29, 0, 'month_max_dif_lon'),
 Text(30, 0, 'traj_max_speed_max'),
 Text(31, 0, 'month_mean_speed'),
 Text(32, 0, 'traj_msg_2_5_mean'),
 Text(33, 0, 'month_max_speed'),
 Text(34, 0, 'month_dist'),
 Text(35, 0, 'traj_msg_10_20_mean'),
 Text(36, 0, 'traj_msg_0dec5_1_mean'),
 Text(37, 0, 'traj_msg_5_10_mean'),
 Text(38, 0, 'to_port'),
 Text(39, 0, 'to_starboard'),
 Text(40, 0, 'traj_msg_1_1dec5_mean'),
 Text(41, 0, 'traj_msg_1dec5_2_mean'),
 Text(42, 0, 'traj_msg_over20_mean')]
```



Iteración 3

In [24]:
```
df_final_static = pd.read_csv('final_static_data_jan.csv')
df_final_static.shape
```

Out[24]:
```
(40859, 18)
```

In [25]:
```
df_cinematic_parameters = pd.read_csv('cinematic_parameters_3_jan.csv')
df_cinematic_parameters.shape
```

Out[25]:
```
(40859, 29)
```

In [26]:
```
df_cinematic_parameters = df_cinematic_parameters.dropna()
df_cinematic_parameters.shape
```

Out[26]:
```
(35816, 29)
```

In [27]:
```
df_global_parameters = pd.merge(df_cinematic_parameters, df_final_static, on='MMSI', how='inner')
df_global_parameters.rename(columns={'shiptype_x': 'shiptype'}, inplace=True)
df_global_parameters.drop(columns=['shiptype_y','MMSI'], inplace=True)
df_global_parameters.shape
```

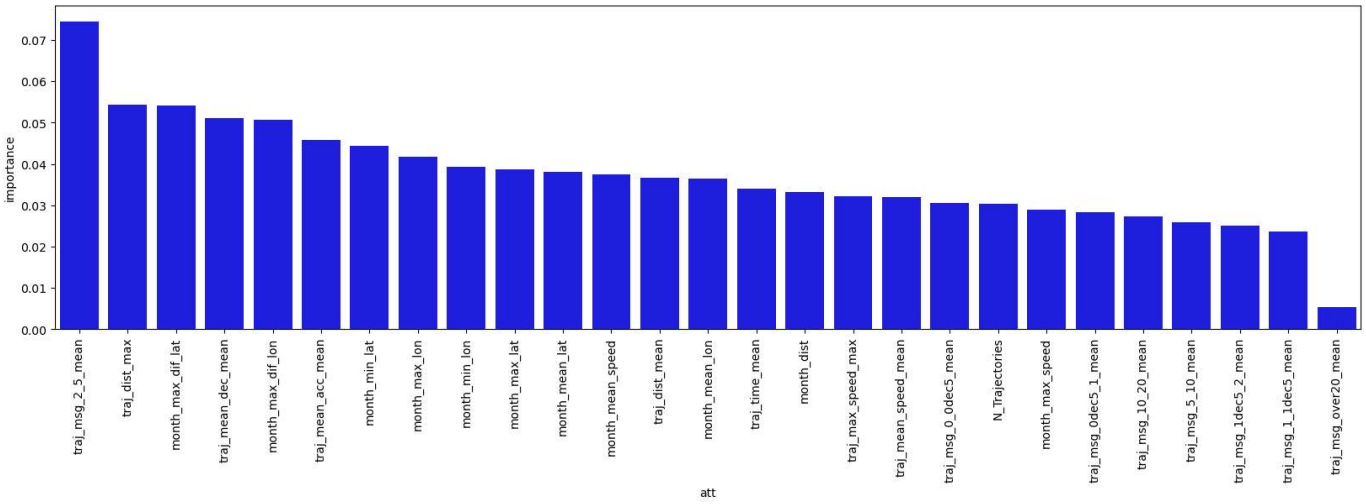
Out[27]:
```
(35816, 44)
```

In [28]:
```
#### Experimento 9
df_cinematic_parameters = df_cinematic_parameters.drop(columns=['MMSI'])
#División entrenamiento y test
df_train, df_test = train_test_split(df_cinematic_parameters, random_state = 47)
x_train = df_train.drop(columns=['shiptype'])
y_train = df_train.shiptype
x_test = df_test.drop(columns=['shiptype'])
y_test = df_test.shiptype

#Creacion del modelo
pipe = Pipeline([('scaler', MinMaxScaler()),
                 ('classifier',RandomForestClassifier(n_estimators=100))])
model = pipe.fit(x_train, y_train)
y_pred = model.predict(x_test)

#Resultados
print(classification_report(y_test, y_pred, zero_division=0))
forest = model['classifier']
importances = forest.feature_importances_
std = np.std([tree.feature_importances_ for tree in forest.estimators_], axis=0)
fi = pd.Series(importances, index=x_train.columns).reset_index()
fi.columns = ['att', 'importance']
fi = fi.sort_values(by='importance', ascending=False).reset_index(drop=True)
g = sns.barplot(data=fi, x='att', y='importance', color='b')
g.figure.set_size_inches(20, 5)
g.set_xticklabels(g.get_xticklabels(), rotation=90)
```

```
                    precision    recall  f1-score   support

     ShipType.Cargo       0.74      0.92      0.82      4499
   ShipType.Fishing       0.86      0.82      0.84      1002
ShipType.Passenger       0.86      0.64      0.74       537
    ShipType.Tanker       0.74      0.37      0.49      1795
       ShipType.Tug       0.80      0.80      0.80      1121

          accuracy                           0.77      8954
         macro avg       0.80      0.71      0.74      8954
      weighted avg       0.77      0.77      0.75      8954
```

Out[28]:

```
[Text(0, 0, 'traj_msg_2_5_mean'),
 Text(1, 0, 'traj_dist_max'),
 Text(2, 0, 'month_max_dif_lat'),
 Text(3, 0, 'traj_mean_dec_mean'),
 Text(4, 0, 'month_max_dif_lon'),
 Text(5, 0, 'traj_mean_acc_mean'),
 Text(6, 0, 'month_min_lat'),
 Text(7, 0, 'month_max_lon'),
 Text(8, 0, 'month_min_lon'),
 Text(9, 0, 'month_max_lat'),
 Text(10, 0, 'month_mean_lat'),
 Text(11, 0, 'month_mean_speed'),
 Text(12, 0, 'traj_dist_mean'),
 Text(13, 0, 'month_mean_lon'),
 Text(14, 0, 'traj_time_mean'),
 Text(15, 0, 'month_dist'),
 Text(16, 0, 'traj_max_speed_max'),
 Text(17, 0, 'traj_mean_speed_mean'),
 Text(18, 0, 'traj_msg_0_0dec5_mean'),
 Text(19, 0, 'N_Trajectories'),
 Text(20, 0, 'month_max_speed'),
 Text(21, 0, 'traj_msg_0dec5_1_mean'),
 Text(22, 0, 'traj_msg_10_20_mean'),
 Text(23, 0, 'traj_msg_5_10_mean'),
 Text(24, 0, 'traj_msg_1dec5_2_mean'),
 Text(25, 0, 'traj_msg_1_1dec5_mean'),
 Text(26, 0, 'traj_msg_over20_mean')]
```



In [29]:

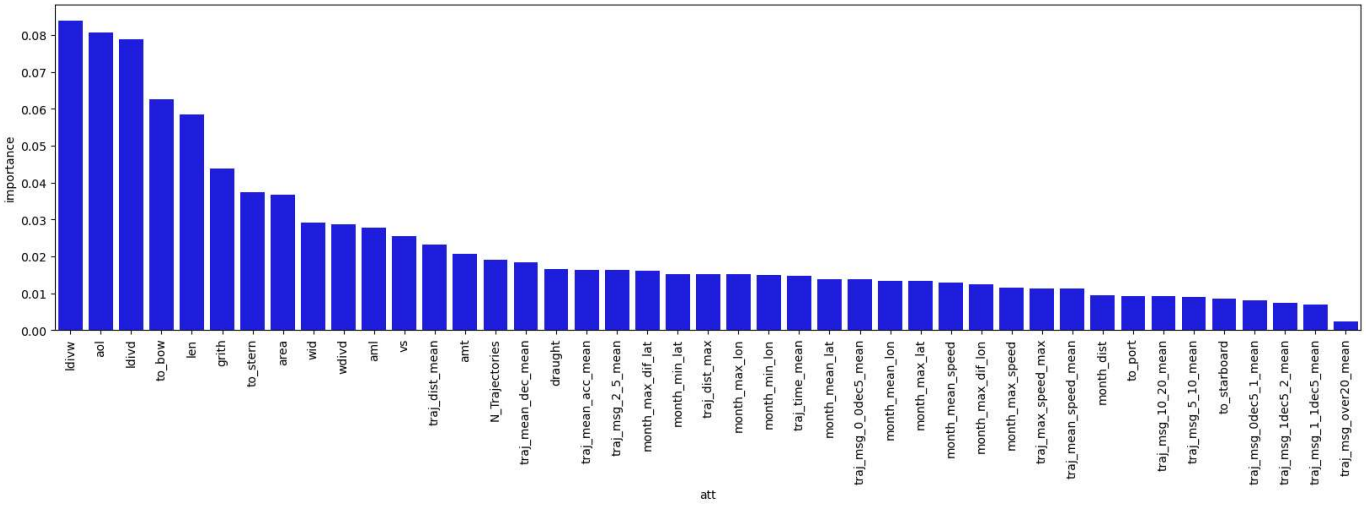
```
#### Experimento 10
#División entrenamiento y test
df_train, df_test = train_test_split(df_global_parameters, random_state = 47)
x_train = df_train.drop(columns=['shiptype'])
y_train = df_train.shiptype
x_test = df_test.drop(columns=['shiptype'])
y_test = df_test.shiptype

#Creacion del modelo
pipe = Pipeline([('scaler', MinMaxScaler()),
                 ('classifier',RandomForestClassifier(n_estimators=100))])
model = pipe.fit(x_train, y_train)
y_pred = model.predict(x_test)

#Resultados
print(classification_report(y_test, y_pred, zero_division=0))
forest = model['classifier']
importances = forest.feature_importances_
std = np.std([tree.feature_importances_ for tree in forest.estimators_], axis=0)
```

```
fi = pd.Series(importances, index=x_train.columns).reset_index()
fi.columns = ['att', 'importance']
fi = fi.sort_values(by='importance', ascending=False).reset_index(drop=True)
g = sns.barplot(data=fi, x='att', y='importance', color='b')
g.figure.set_size_inches(20, 5)
g.set_xticklabels(g.get_xticklabels(), rotation=90)
```

```
                   precision    recall  f1-score   support

    ShipType.Cargo       0.90      0.96      0.93      4499
  ShipType.Fishing       0.89      0.93      0.91      1002
ShipType.Passenger       0.89      0.82      0.86       537
   ShipType.Tanker       0.92      0.79      0.85      1795
      ShipType.Tug       0.93      0.91      0.92      1121

         accuracy                           0.90      8954
        macro avg       0.90      0.88      0.89      8954
     weighted avg       0.90      0.90      0.90      8954
```

Out[29]:

```
[Text(0, 0, 'ldivw'),
 Text(1, 0, 'aol'),
 Text(2, 0, 'ldivd'),
 Text(3, 0, 'to_bow'),
 Text(4, 0, 'len'),
 Text(5, 0, 'grith'),
 Text(6, 0, 'to_stern'),
 Text(7, 0, 'area'),
 Text(8, 0, 'wid'),
 Text(9, 0, 'wdivd'),
 Text(10, 0, 'aml'),
 Text(11, 0, 'vs'),
 Text(12, 0, 'traj_dist_mean'),
 Text(13, 0, 'amt'),
 Text(14, 0, 'N_Trajectories'),
 Text(15, 0, 'traj_mean_dec_mean'),
 Text(16, 0, 'draught'),
 Text(17, 0, 'traj_mean_acc_mean'),
 Text(18, 0, 'traj_msg_2_5_mean'),
 Text(19, 0, 'month_max_dif_lat'),
 Text(20, 0, 'month_min_lat'),
 Text(21, 0, 'traj_dist_max'),
 Text(22, 0, 'month_max_lon'),
 Text(23, 0, 'month_min_lon'),
 Text(24, 0, 'traj_time_mean'),
 Text(25, 0, 'month_mean_lat'),
 Text(26, 0, 'traj_msg_0_0dec5_mean'),
 Text(27, 0, 'month_mean_lon'),
 Text(28, 0, 'month_max_lat'),
 Text(29, 0, 'month_mean_speed'),
 Text(30, 0, 'month_max_dif_lon'),
 Text(31, 0, 'month_max_speed'),
 Text(32, 0, 'traj_max_speed_max'),
 Text(33, 0, 'traj_mean_speed_mean'),
 Text(34, 0, 'month_dist'),
 Text(35, 0, 'to_port'),
 Text(36, 0, 'traj_msg_10_20_mean'),
 Text(37, 0, 'traj_msg_5_10_mean'),
 Text(38, 0, 'to_starboard'),
 Text(39, 0, 'traj_msg_0dec5_1_mean'),
 Text(40, 0, 'traj_msg_1dec5_2_mean'),
 Text(41, 0, 'traj_msg_1_1dec5_mean'),
 Text(42, 0, 'traj_msg_over20_mean')]
```



Iteración 4

In [30]:
```
df_final_static = pd.read_csv('final_static_data_jan.csv')
df_final_static.shape
```

Out[30]:
```
(40859, 18)
```

In [31]:
```
df_cinematic_parameters = pd.read_csv('cinematic_parameters_4_jan.csv')
df_cinematic_parameters.shape
```

Out[31]:
```
(40859, 63)
```

In [32]:
```
df_cinematic_parameters = df_cinematic_parameters.dropna()
df_cinematic_parameters.shape
```

Out[32]:
```
(31889, 63)
```

In [33]:
```
df_global_parameters = pd.merge(df_cinematic_parameters, df_final_static, on='MMSI', how='inner')
df_global_parameters.rename(columns={'shiptype_x': 'shiptype'}, inplace=True)
df_global_parameters.drop(columns=['shiptype_y','MMSI'], inplace=True)
df_global_parameters.shape
```

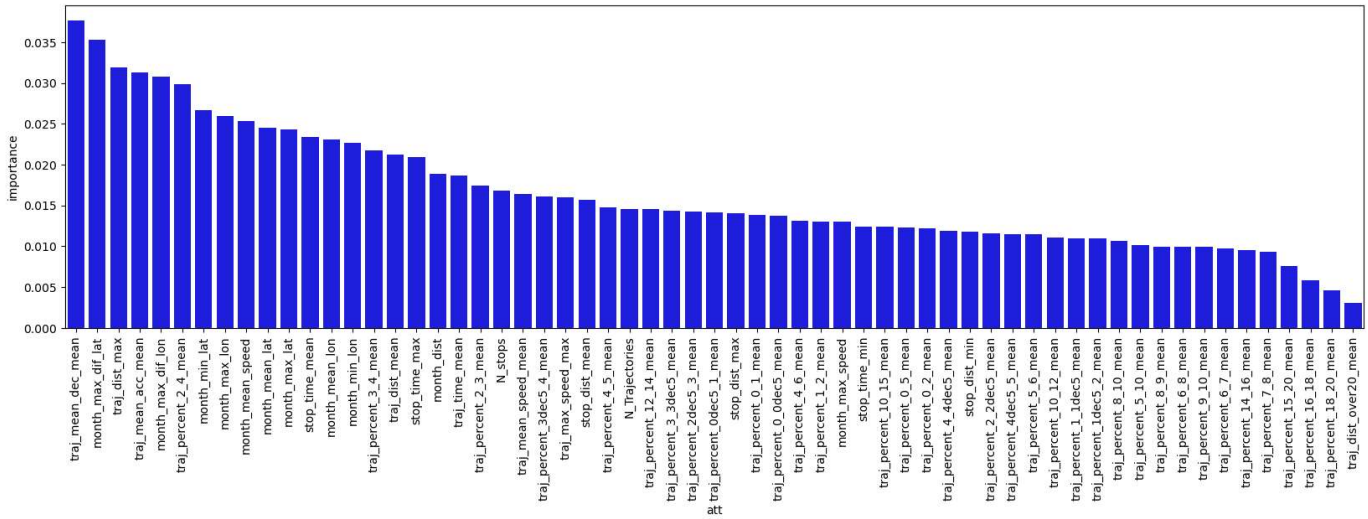
Out[33]:
```
(31889, 78)
```

In [34]:
```
#### Experimento 11
df_cinematic_parameters = df_cinematic_parameters.drop(columns=['MMSI'])
#División entrenamiento y test
df_train, df_test = train_test_split(df_cinematic_parameters, random_state = 47)
x_train = df_train.drop(columns=['shiptype'])
y_train = df_train.shiptype
x_test = df_test.drop(columns=['shiptype'])
y_test = df_test.shiptype

#Creacion del modelo
pipe = Pipeline([('scaler', MinMaxScaler()),
                 ('classifier',RandomForestClassifier(n_estimators=100))])
model = pipe.fit(x_train, y_train)
y_pred = model.predict(x_test)

#Resultados
print(classification_report(y_test, y_pred, zero_division=0))
forest = model['classifier']
importances = forest.feature_importances_
std = np.std([tree.feature_importances_ for tree in forest.estimators_], axis=0)
fi = pd.Series(importances, index=x_train.columns).reset_index()
fi.columns = ['att', 'importance']
fi = fi.sort_values(by='importance', ascending=False).reset_index(drop=True)
g = sns.barplot(data=fi, x='att', y='importance', color='b')
g.figure.set_size_inches(20, 5)
g.set_xticklabels(g.get_xticklabels(), rotation=90)
```

```
                    precision    recall  f1-score   support

      ShipType.Cargo       0.75      0.91      0.83      3950
    ShipType.Fishing       0.81      0.80      0.81       876
  ShipType.Passenger       0.85      0.65      0.74       487
     ShipType.Tanker       0.76      0.40      0.52      1508
        ShipType.Tug       0.80      0.80      0.80      1152

           accuracy                           0.77      7973
          macro avg       0.79      0.71      0.74      7973
       weighted avg       0.77      0.77      0.76      7973
```

Out[34]:

```
[Text(0, 0, 'traj_mean_dec_mean'),
 Text(1, 0, 'month_max_dif_lat'),
 Text(2, 0, 'traj_dist_max'),
 Text(3, 0, 'traj_mean_acc_mean'),
 Text(4, 0, 'month_max_dif_lon'),
 Text(5, 0, 'traj_percent_2_4_mean'),
 Text(6, 0, 'month_min_lat'),
 Text(7, 0, 'month_max_lon'),
 Text(8, 0, 'month_mean_speed'),
 Text(9, 0, 'month_mean_lat'),
 Text(10, 0, 'month_max_lat'),
 Text(11, 0, 'stop_time_mean'),
 Text(12, 0, 'month_mean_lon'),
 Text(13, 0, 'month_min_lon'),
 Text(14, 0, 'traj_percent_3_4_mean'),
 Text(15, 0, 'traj_dist_mean'),
 Text(16, 0, 'stop_time_max'),
 Text(17, 0, 'month_dist'),
 Text(18, 0, 'traj_time_mean'),
 Text(19, 0, 'traj_percent_2_3_mean'),
 Text(20, 0, 'N_stops'),
 Text(21, 0, 'traj_mean_speed_mean'),
 Text(22, 0, 'traj_percent_3dec5_4_mean'),
 Text(23, 0, 'traj_max_speed_max'),
 Text(24, 0, 'stop_dist_mean'),
 Text(25, 0, 'traj_percent_4_5_mean'),
 Text(26, 0, 'N_Trajectories'),
 Text(27, 0, 'traj_percent_12_14_mean'),
 Text(28, 0, 'traj_percent_3_3dec5_mean'),
 Text(29, 0, 'traj_percent_2dec5_3_mean'),
 Text(30, 0, 'traj_percent_0dec5_1_mean'),
 Text(31, 0, 'stop_dist_max'),
 Text(32, 0, 'traj_percent_0_1_mean'),
 Text(33, 0, 'traj_percent_0_0dec5_mean'),
 Text(34, 0, 'traj_percent_4_6_mean'),
 Text(35, 0, 'traj_percent_1_2_mean'),
 Text(36, 0, 'month_max_speed'),
 Text(37, 0, 'stop_time_min'),
 Text(38, 0, 'traj_percent_10_15_mean'),
 Text(39, 0, 'traj_percent_0_5_mean'),
 Text(40, 0, 'traj_percent_0_2_mean'),
 Text(41, 0, 'traj_percent_4_4dec5_mean'),
 Text(42, 0, 'stop_dist_min'),
 Text(43, 0, 'traj_percent_2_2dec5_mean'),
 Text(44, 0, 'traj_percent_4dec5_5_mean'),
 Text(45, 0, 'traj_percent_5_6_mean'),
 Text(46, 0, 'traj_percent_10_12_mean'),
 Text(47, 0, 'traj_percent_1_1dec5_mean'),
 Text(48, 0, 'traj_percent_1dec5_2_mean'),
 Text(49, 0, 'traj_percent_8_10_mean'),
 Text(50, 0, 'traj_percent_5_10_mean'),
 Text(51, 0, 'traj_percent_8_9_mean'),
 Text(52, 0, 'traj_percent_6_8_mean'),
 Text(53, 0, 'traj_percent_9_10_mean'),
 Text(54, 0, 'traj_percent_6_7_mean'),
 Text(55, 0, 'traj_percent_14_16_mean'),
 Text(56, 0, 'traj_percent_7_8_mean'),
 Text(57, 0, 'traj_percent_15_20_mean'),
 Text(58, 0, 'traj_percent_16_18_mean'),
 Text(59, 0, 'traj_percent_18_20_mean'),
 Text(60, 0, 'traj_dist_over20_mean')]
```

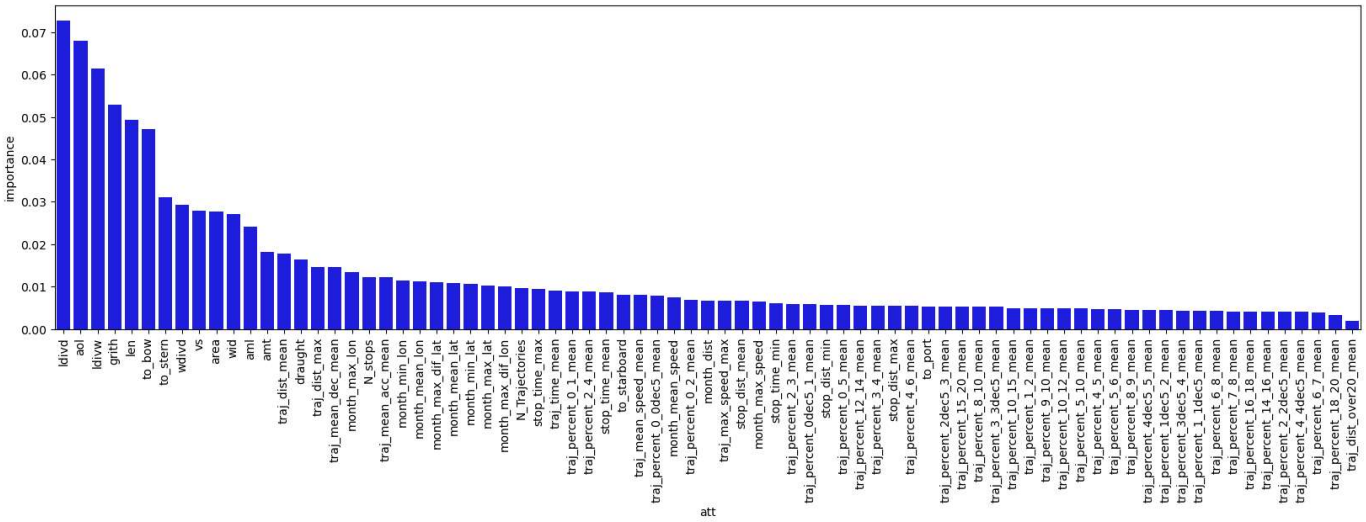


In [35]:
```
#### Experimento 12
#División entrenamiento y test
df_train, df_test = train_test_split(df_global_parameters, random_state = 47)
x_train = df_train.drop(columns=['shiptype'])
y_train = df_train.shiptype
x_test = df_test.drop(columns=['shiptype'])
y_test = df_test.shiptype

#Creacion del modelo
pipe = Pipeline([('scaler', MinMaxScaler()),
                 ('classifier',RandomForestClassifier(n_estimators=100))])
model = pipe.fit(x_train, y_train)
y_pred = model.predict(x_test)

#Resultados
print(classification_report(y_test, y_pred, zero_division=0))
forest = model['classifier']
importances = forest.feature_importances_
std = np.std([tree.feature_importances_ for tree in forest.estimators_], axis=0)
fi = pd.Series(importances, index=x_train.columns).reset_index()
fi.columns = ['att', 'importance']
fi = fi.sort_values(by='importance', ascending=False).reset_index(drop=True)
g = sns.barplot(data=fi, x='att', y='importance', color='b')
g.figure.set_size_inches(20, 5)
g.set_xticklabels(g.get_xticklabels(), rotation=90)
```

```
                   precision    recall  f1-score   support

    ShipType.Cargo       0.89      0.96      0.92      3950
  ShipType.Fishing       0.88      0.94      0.91       876
ShipType.Passenger       0.88      0.80      0.84       487
   ShipType.Tanker       0.92      0.76      0.83      1508
      ShipType.Tug       0.92      0.90      0.91      1152

         accuracy                           0.90      7973
        macro avg       0.90      0.87      0.88      7973
     weighted avg       0.90      0.90      0.90      7973
```

Out[35]:
```
[Text(0, 0, 'ldivd'),
 Text(1, 0, 'aol'),
 Text(2, 0, 'ldivw'),
 Text(3, 0, 'grith'),
 Text(4, 0, 'len'),
 Text(5, 0, 'to_bow'),
 Text(6, 0, 'to_stern'),
 Text(7, 0, 'wdivd'),
 Text(8, 0, 'vs'),
 Text(9, 0, 'area'),
 Text(10, 0, 'wid'),
 Text(11, 0, 'aml'),
 Text(12, 0, 'amt'),
 Text(13, 0, 'traj_dist_mean'),
 Text(14, 0, 'draught'),
 Text(15, 0, 'traj_dist_max'),
 Text(16, 0, 'traj_mean_dec_mean'),
 Text(17, 0, 'month_max_lon'),
 Text(18, 0, 'N_stops'),
 Text(19, 0, 'traj_mean_acc_mean'),
 Text(20, 0, 'month_min_lon'),
 Text(21, 0, 'month_mean_lon'),
 Text(22, 0, 'month_max_dif_lat'),
 Text(23, 0, 'month_mean_lat'),
 Text(24, 0, 'month_min_lat'),
 Text(25, 0, 'month_max_lat'),
 Text(26, 0, 'month_max_dif_lon'),
 Text(27, 0, 'N_Trajectories'),
 Text(28, 0, 'stop_time_max'),
 Text(29, 0, 'traj_time_mean'),
 Text(30, 0, 'traj_percent_0_1_mean'),
 Text(31, 0, 'traj_percent_2_4_mean'),
 Text(32, 0, 'stop_time_mean'),
 Text(33, 0, 'to_starboard'),
 Text(34, 0, 'traj_mean_speed_mean'),
 Text(35, 0, 'traj_percent_0_0dec5_mean'),
 Text(36, 0, 'month_mean_speed'),
 Text(37, 0, 'traj_percent_0_2_mean'),
 Text(38, 0, 'month_dist'),
 Text(39, 0, 'traj_max_speed_max'),
 Text(40, 0, 'stop_dist_mean'),
 Text(41, 0, 'month_max_speed'),
 Text(42, 0, 'stop_time_min'),
 Text(43, 0, 'traj_percent_2_3_mean'),
 Text(44, 0, 'traj_percent_0dec5_1_mean'),
 Text(45, 0, 'stop_dist_min'),
 Text(46, 0, 'traj_percent_0_5_mean'),
 Text(47, 0, 'traj_percent_12_14_mean'),
 Text(48, 0, 'traj_percent_3_4_mean'),
 Text(49, 0, 'stop_dist_max'),
 Text(50, 0, 'traj_percent_4_6_mean'),
 Text(51, 0, 'to_port'),
 Text(52, 0, 'traj_percent_2dec5_3_mean'),
 Text(53, 0, 'traj_percent_15_20_mean'),
 Text(54, 0, 'traj_percent_8_10_mean'),
 Text(55, 0, 'traj_percent_3_3dec5_mean'),
 Text(56, 0, 'traj_percent_10_15_mean'),
 Text(57, 0, 'traj_percent_1_2_mean'),
 Text(58, 0, 'traj_percent_9_10_mean'),
 Text(59, 0, 'traj_percent_10_12_mean'),
 Text(60, 0, 'traj_percent_5_10_mean'),
 Text(61, 0, 'traj_percent_4_5_mean'),
 Text(62, 0, 'traj_percent_5_6_mean'),
 Text(63, 0, 'traj_percent_8_9_mean'),
 Text(64, 0, 'traj_percent_4dec5_5_mean'),
 Text(65, 0, 'traj_percent_1dec5_2_mean'),
 Text(66, 0, 'traj_percent_3dec5_4_mean'),
 Text(67, 0, 'traj_percent_1_1dec5_mean'),
 Text(68, 0, 'traj_percent_6_8_mean'),
 Text(69, 0, 'traj_percent_7_8_mean'),
 Text(70, 0, 'traj_percent_16_18_mean'),
 Text(71, 0, 'traj_percent_14_16_mean'),
 Text(72, 0, 'traj_percent_2_2dec5_mean'),
 Text(73, 0, 'traj_percent_4_4dec5_mean'),
 Text(74, 0, 'traj_percent_6_7_mean'),
```

```
Text(75, 0, 'traj_percent_18_20_mean'),
Text(76, 0, 'traj_dist_over20_mean')]
```



Iteración 5

In [36]:
```
df_final_static = pd.read_csv('final_static_data_jan.csv')
df_final_static.shape
```

Out[36]:
```
(40859, 18)
```

In [37]:
```
df_cinematic_parameters = pd.read_csv('cinematic_parameters_4_jan.csv')
df_cinematic_parameters.shape
```

Out[37]:
```
(40859, 63)
```

In [38]:
```
columnas_a_eliminar = ['traj_percent_0_0dec5_mean',
                       'traj_percent_0dec5_1_mean',
                       'traj_percent_1_1dec5_mean',
                       'traj_percent_1dec5_2_mean',
                       'traj_percent_2_2dec5_mean',
                       'traj_percent_2dec5_3_mean',
                       'traj_percent_3_3dec5_mean',
                       'traj_percent_3dec5_4_mean',
                       'traj_percent_4_4dec5_mean',
                       'traj_percent_4dec5_5_mean']

# Eliminar las columnas especificadas
df_cinematic_parameters = df_cinematic_parameters.drop(columns=columnas_a_eliminar)
```

In [39]:
```
columnas_a_eliminar = ['traj_dist_over20_mean',
                       'traj_percent_18_20_mean',
                       'traj_percent_16_18_mean',
                       'traj_percent_14_16_mean',
                       'traj_percent_12_14_mean',
                       'traj_percent_12_14_mean',
                       'traj_percent_10_12_mean',
                       'traj_percent_8_10_mean',
                       'traj_percent_6_8_mean']

# Eliminar las columnas especificadas
df_cinematic_parameters = df_cinematic_parameters.drop(columns=columnas_a_eliminar)
```

In [40]:
```
columnas_a_eliminar = [
    'traj_percent_0_1_mean',
    'traj_percent_1_2_mean',
    'traj_percent_2_3_mean',
    'traj_percent_3_4_mean',
    'traj_percent_4_5_mean',
    'traj_percent_5_6_mean',
    'traj_percent_6_7_mean',
    'traj_percent_7_8_mean',
    'traj_percent_8_9_mean',
    'traj_percent_9_10_mean',]
```

```
# Eliminar las columnas especificadas
df_cinematic_parameters = df_cinematic_parameters.drop(columns=columnas_a_eliminar)
```

In [41]:
```
columnas_a_eliminar = [
    'traj_percent_0_5_mean',
    'traj_percent_5_10_mean',
    'traj_percent_10_15_mean',
    'traj_percent_15_20_mean']

# Eliminar las columnas especificadas
df_cinematic_parameters = df_cinematic_parameters.drop(columns=columnas_a_eliminar)
```

In [42]:
```
columnas_a_eliminar = [
    'stop_time_min',
    'stop_dist_min']

# Eliminar las columnas especificadas
df_cinematic_parameters = df_cinematic_parameters.drop(columns=columnas_a_eliminar)
```

In [43]:
```
df_cinematic_parameters = df_cinematic_parameters.dropna()
df_cinematic_parameters.shape
```

Out[43]:
```
(31889, 29)
```

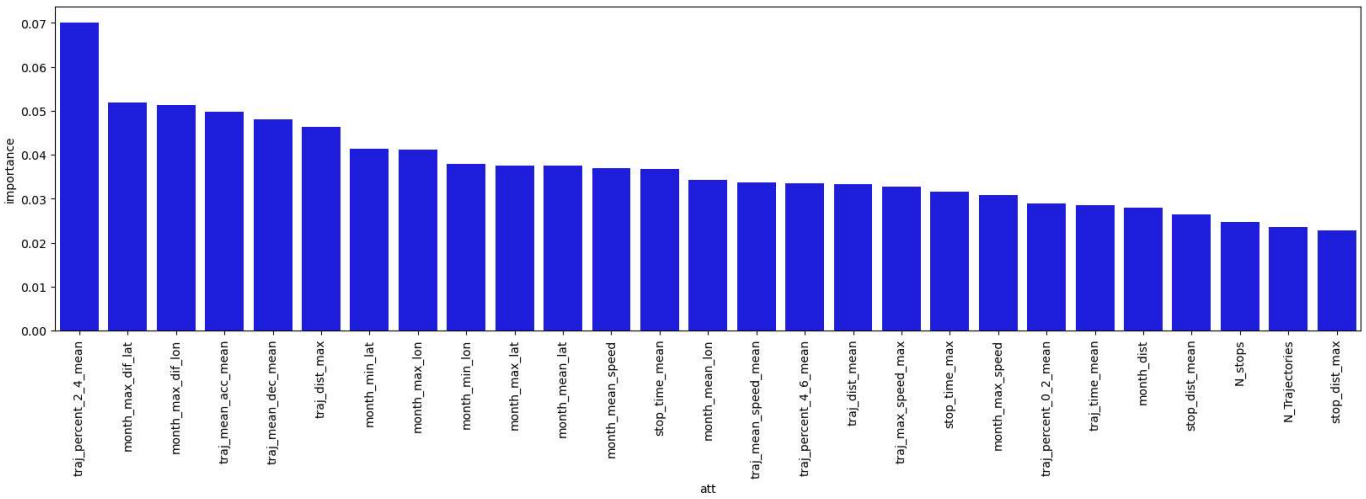
In [44]:
```
df_global_parameters = pd.merge(df_cinematic_parameters, df_final_static, on='MMSI', how='inner')
df_global_parameters.rename(columns={'shiptype_x': 'shiptype'}, inplace=True)
df_global_parameters.drop(columns=['shiptype_y','MMSI'], inplace=True)
df_global_parameters.shape
```

Out[44]:
```
(31889, 44)
```

In [45]:
```
#### Experimento 13
df_cinematic_parameters = df_cinematic_parameters.drop(columns=['MMSI'])
#División entrenamiento y test
df_train, df_test = train_test_split(df_cinematic_parameters, random_state = 47)
x_train = df_train.drop(columns=['shiptype'])
y_train = df_train.shiptype
x_test = df_test.drop(columns=['shiptype'])
y_test = df_test.shiptype

#Creacion del modelo
pipe = Pipeline([('scaler', MinMaxScaler()),
                 ('classifier',RandomForestClassifier(n_estimators=100))])
model = pipe.fit(x_train, y_train)
y_pred = model.predict(x_test)

#Resultados
print(classification_report(y_test, y_pred, zero_division=0))
forest = model['classifier']
importances = forest.feature_importances_
std = np.std([tree.feature_importances_ for tree in forest.estimators_], axis=0)
fi = pd.Series(importances, index=x_train.columns).reset_index()
fi.columns = ['att', 'importance']
fi = fi.sort_values(by='importance', ascending=False).reset_index(drop=True)
g = sns.barplot(data=fi, x='att', y='importance', color='b')
g.figure.set_size_inches(20, 5)
g.set_xticklabels(g.get_xticklabels(), rotation=90)
```

```
                   precision    recall  f1-score   support

    ShipType.Cargo       0.77      0.91      0.83      3950
  ShipType.Fishing       0.84      0.82      0.83       876
ShipType.Passenger       0.85      0.68      0.76       487
   ShipType.Tanker       0.75      0.44      0.56      1508
      ShipType.Tug       0.80      0.81      0.80      1152

         accuracy                           0.78      7973
        macro avg       0.80      0.73      0.76      7973
     weighted avg       0.78      0.78      0.77      7973
```

Out[45]:
```
[Text(0, 0, 'traj_percent_2_4_mean'),
 Text(1, 0, 'month_max_dif_lat'),
 Text(2, 0, 'month_max_dif_lon'),
 Text(3, 0, 'traj_mean_acc_mean'),
 Text(4, 0, 'traj_mean_dec_mean'),
 Text(5, 0, 'traj_dist_max'),
 Text(6, 0, 'month_min_lat'),
 Text(7, 0, 'month_max_lon'),
 Text(8, 0, 'month_min_lon'),
 Text(9, 0, 'month_max_lat'),
 Text(10, 0, 'month_mean_lat'),
 Text(11, 0, 'month_mean_speed'),
 Text(12, 0, 'stop_time_mean'),
 Text(13, 0, 'month_mean_lon'),
 Text(14, 0, 'traj_mean_speed_mean'),
 Text(15, 0, 'traj_percent_4_6_mean'),
 Text(16, 0, 'traj_dist_mean'),
 Text(17, 0, 'traj_max_speed_max'),
 Text(18, 0, 'stop_time_max'),
 Text(19, 0, 'month_max_speed'),
 Text(20, 0, 'traj_percent_0_2_mean'),
 Text(21, 0, 'traj_time_mean'),
 Text(22, 0, 'month_dist'),
 Text(23, 0, 'stop_dist_mean'),
 Text(24, 0, 'N_stops'),
 Text(25, 0, 'N_Trajectories'),
 Text(26, 0, 'stop_dist_max')]
```



In [46]:
```
#### Experimento 14
#División entrenamiento y test
df_train, df_test = train_test_split(df_global_parameters, random_state = 47)
x_train = df_train.drop(columns=['shiptype'])
y_train = df_train.shiptype
x_test = df_test.drop(columns=['shiptype'])
y_test = df_test.shiptype

#Creacion del modelo
pipe = Pipeline([('scaler', MinMaxScaler()),
                 ('classifier',RandomForestClassifier(n_estimators=100))])
model = pipe.fit(x_train, y_train)
y_pred = model.predict(x_test)

#Resultados
print(classification_report(y_test, y_pred, zero_division=0))
forest = model['classifier']
importances = forest.feature_importances_
std = np.std([tree.feature_importances_ for tree in forest.estimators_], axis=0)
fi = pd.Series(importances, index=x_train.columns).reset_index()
fi.columns = ['att', 'importance']
fi = fi.sort_values(by='importance', ascending=False).reset_index(drop=True)
g = sns.barplot(data=fi, x='att', y='importance', color='b')
g.figure.set_size_inches(20, 5)
g.set_xticklabels(g.get_xticklabels(), rotation=90)
```

	precision	recall	f1-score	support
ShipType.Cargo	0.90	0.96	0.93	3950
ShipType.Fishing	0.88	0.93	0.90	876
ShipType.Passenger	0.87	0.83	0.85	487
ShipType.Tanker	0.92	0.78	0.84	1508
ShipType.Tug	0.93	0.91	0.92	1152
accuracy			0.91	7973
macro avg	0.90	0.88	0.89	7973
weighted avg	0.91	0.91	0.90	7973

Out[46]:
```
[Text(0, 0, 'aol'),
 Text(1, 0, 'to_bow'),
 Text(2, 0, 'ldivd'),
 Text(3, 0, 'ldivw'),
 Text(4, 0, 'grith'),
 Text(5, 0, 'len'),
 Text(6, 0, 'area'),
 Text(7, 0, 'to_stern'),
 Text(8, 0, 'wdivd'),
 Text(9, 0, 'aml'),
 Text(10, 0, 'wid'),
 Text(11, 0, 'vs'),
 Text(12, 0, 'amt'),
 Text(13, 0, 'traj_dist_mean'),
 Text(14, 0, 'traj_mean_dec_mean'),
 Text(15, 0, 'traj_mean_acc_mean'),
 Text(16, 0, 'month_max_lon'),
 Text(17, 0, 'draught'),
 Text(18, 0, 'month_max_dif_lat'),
 Text(19, 0, 'month_mean_lon'),
 Text(20, 0, 'N_stops'),
 Text(21, 0, 'month_min_lon'),
 Text(22, 0, 'month_max_lat'),
 Text(23, 0, 'traj_time_mean'),
 Text(24, 0, 'N_Trajectories'),
 Text(25, 0, 'month_mean_lat'),
 Text(26, 0, 'traj_percent_2_4_mean'),
 Text(27, 0, 'month_min_lat'),
 Text(28, 0, 'traj_mean_speed_mean'),
 Text(29, 0, 'traj_dist_max'),
 Text(30, 0, 'stop_time_max'),
 Text(31, 0, 'month_max_dif_lon'),
 Text(32, 0, 'traj_max_speed_max'),
 Text(33, 0, 'traj_percent_0_2_mean'),
 Text(34, 0, 'stop_time_mean'),
 Text(35, 0, 'traj_percent_4_6_mean'),
 Text(36, 0, 'month_mean_speed'),
 Text(37, 0, 'month_max_speed'),
 Text(38, 0, 'month_dist'),
 Text(39, 0, 'stop_dist_mean'),
 Text(40, 0, 'to_starboard'),
 Text(41, 0, 'stop_dist_max'),
 Text(42, 0, 'to_port')]
```