NTNU

Kunnskap for en bedre verden

TDT4173 - Modern Machine Learning in Practice

140 - Los Paperos

Authors:

Member	**Student ID**	**Email**
Gimmillaro Gabriele	132543	gabriegi@stud.ntnu.no
Peiretti Riccardo	133373	riccap@stud.ntnu.no
Rocchi Ilaria	132938	ilariaro@stud.ntnu.no

10 November, 2024

Table of Contents

1 Introduction

The report describes the final result of a group project for the course TDT4173 - Modern Machine Learning in Practice at the Norwegian University of Science and Technology (NTNU). We begin by introducing the context and defining the problem statement. This is followed by an overview of the exploratory data analysis (EDA) conducted, a description of the feature engineering process, the ensemble modeling approach used, and an interpretation of the final model.

Objective

Given the Automatic Identification System (AIS) data from 1st January to 7th May 2024, predict the future positions of vessels at given timestamps for five days into the future. The goal is to develop predictive models that account for various factors such as congestion, port calls, and other events affecting the vessels' journeys.

Task Description

- Given an AIS dataset containing the positions of 689 vessels per time from January to May 2024, predict the next positions of a select 216 of these vessels at their given timestamps.
- One must utilize the AIS dataset; the *vessels*, *schedules*, and *ports* datasets are optional.

Dataset Overview

The *required* dataset are:

- *ais_train.csv*: contains the AIS data for training. This contains the positions of 689 vessels. The dataset was sampled every 20 minutes, but the timestamps for each vessel are irregular.
- *ais_test.csv*: contains AIS data without longitudes and latitudes for testing for 216 vessels that will be used to evaluate the score of the models.

The *optional* dataset are:

- *schedules_to_may_2024.csv*: contains the planned arrival destinations and time as communicated from the shipping lines for a select 252 vessels.
- *vessels*: Contains some information about each vessel.
- *ports.csv*: contains information about the ports referenced in *schedules_to_may_2024.csv*.

2 Exploratory Data Analysis

In this section, we present the exploratory data analysis (EDA) performed to understand the dataset, identify potential challenges, and prepare the data for the machine learning task. This process includes exploring the dataset's structure, verifying its intuitiveness, and investigating individual and combined features. We also clean and pre-process the data to ensure that it is in an optimal state for model training and prediction.

2.1 Domain Knowledge Exploration

In this analysis, domain knowledge about the maritime industry and vessel movement is crucial. A solid understanding of the variables that affect vessel positions - such as historical latitude and longitude, course over ground (COG), speed over ground (SOG), and rate of turn (ROT) - helped guide the feature engineering process, reported in Section 3. For a better understating, at the beginning, we combined the training dataset with the ports dataset to visualize the geographical distribution of vessels and ports, Figure 1.

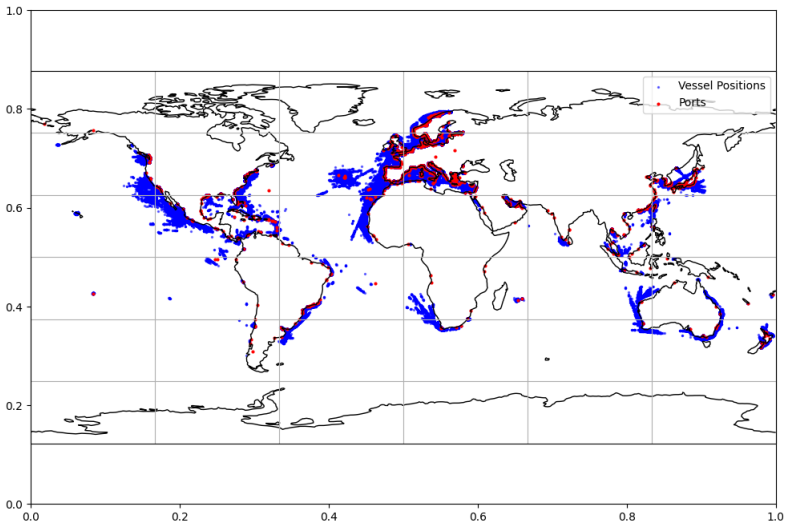

Figure 1: Geographical Distribution of Vessel Positions and Ports

A scatter plot was created to display the positions of the vessels based on their latitude and longitude, alongside the coordinates of the ports, offering a valuable insights into the general context of the problem.

2.2 Understand How the Data was Generated

Understanding how the data was generated is essential for identifying potential biases and ensuring data consistency. The dataset is created through the Automatic Identification System. AIS data provides real-time information about a vessel's position, speed, and course, among others. AIS is essential for navigation and collision avoidance, maritime traffic monitoring, and environmental protection. This data enhances safety, efficiency, and security in maritime operations, but its limitations include potential inaccuracies due to human error and signal interference, understanding these potential weaknesses in the data is essential for addressing any inconsistencies during the data cleaning and feature engineering processes.

2.3 Data Understanding and Feature Analysis

In this section, individual features, as well as pair of features, were explored.
Given the potential vulnerabilities in the AIS data, each feature was carefully examined to ensure its relevance and quality in relation to the problem at hand. This involved several steps, starting with checking for missing values (NaN) and identifying any incorrect or invalid entries that could skew the analysis. By addressing these issues, we ensured that the features used in the model were both accurate and meaningful.

2.3.1 Explore Individual Features

Indeed, it is important to check if the data aligns with intuitive expectations. For instance, features such as latitude, longitude, COG, SOG, ROT, heading (the direction in which the vessel's bow is pointing), and navigation status (which indicates the vessel's current operational state) are intuitive, as they directly describe a vessel's position and movement. To ensure the data follows these expected patterns, exploratory data analysis was conducted, using plots to visually inspect whether the relationships between these features made sense. This step allowed us to identify any inconsistencies or unexpected behaviors in the data, such as outliers, missing values, or unusual trends.
For example, in Figure 2, the correlation between NavStat and SOG (Speed over Ground) is shown.

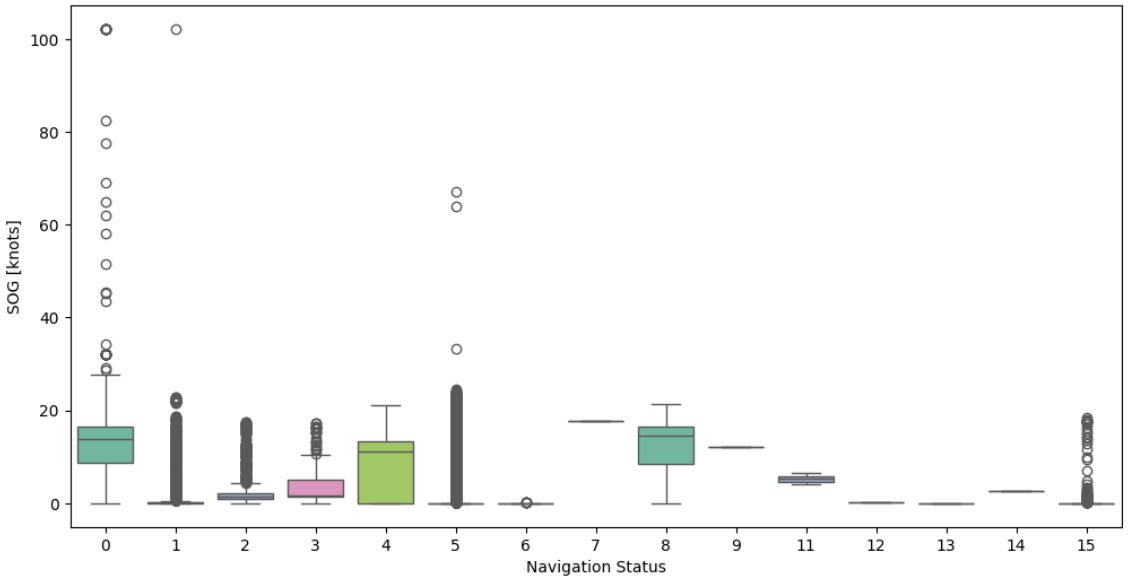

Figure 2: Distribution of Speed over Ground (SOG) by Navigation Status (NavStat)

Since NavStat represents the operational status of the vessel (e.g., whether it is underway, moored, anchored, etc.), it is useful to examine if the speed (SOG) varies according to the vessel's operational status. A box plot is used to visualize how the speed changes across different NavStat states. According to the AIS navigation status, the number 5 represents 'moored', which in theory should result in a speed close to zero. However, in practice, many vessels show a $SOG > 0$ even when their status is 'moored'.
Additionally, other plots were generated, such as histograms showing the distribution of SOG to examine its range of values, in Figure 3.

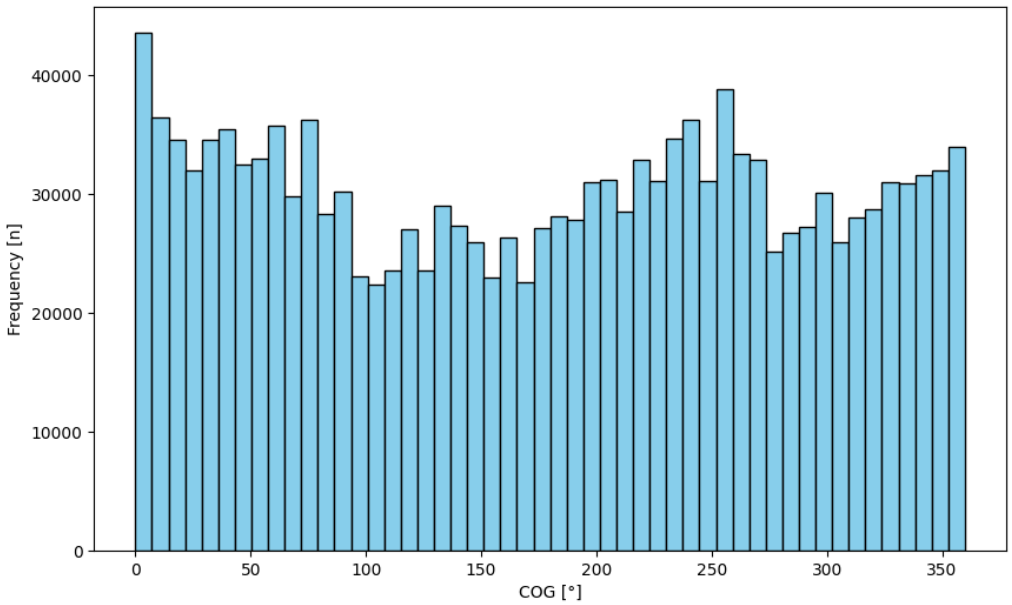

Figure 3: Distribution of COG

2.3.2 Explore Pairs of Features

As well as plots to explore the correlation between the different features such as ROT and SOG, in Figure 4. Most data points are close to zero ROT, indicating stable vessel trajectories. SOG ranges from 0 to 20 knots, with a concentration around 10-15 knots.

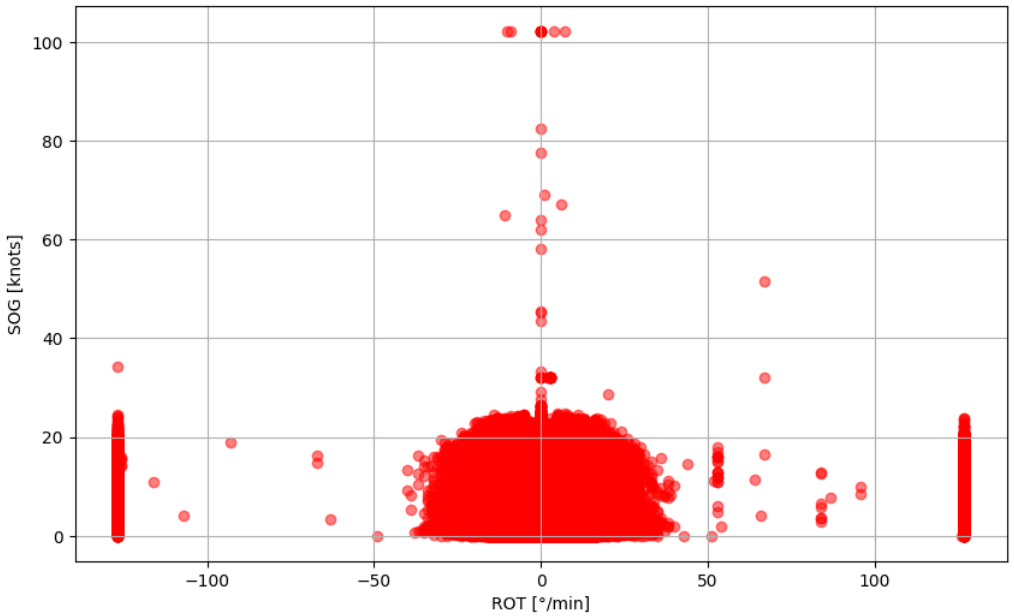

Figure 4: Correlation between ROT and SOG

These findings suggest further data cleaning, especially regarding outliers, and a need to explore other features that may better explain vessel dynamics and to understand how to address this inconsistency in the subsequent stages of Data Cleaning in Section 2.4 and Feature Engineering in Section 3. In general, such plots were created for the train, ports, and vessels datasets to gain a better understanding of the data being analyzed and to explore correlations between them.
Moreover, according to the official AIS documentation, certain values are flagged as default or null, which should not be considered for analysis as they do not provide meaningful information. These values were taken into account and excluded during the data cleaning process, as reported in the following Section 2.4.

2.4 Cleaning up Features

Feature cleaning is an essential step to ensure that the dataset is ready for modeling. The AIS data contains several features with potential inconsistencies, missing values, and default entries that need to be addressed to avoid skewing the analysis. Below, we outline the steps taken during the data cleaning process, based on the guidelines provided by the AIS documentation.

Handling Missing and Default Values

As mentioned in the AIS documentation, certain features may contain missing or default values were dropped from the training test. For example:

- Course Over Ground (COG): a value of 360 indicates that the data is not available, and values between 360.1 and 409.5 should not be used.
- Speed Over Ground (SOG): a value of $1,023$ indicates that the data is not available, and $1,022$ represents a speed greater than or equal to 102.2 knots.
- Rate of Turn (ROT): a value of -128 indicates no turn information is available, and values ± 127 represent uncertain turn information.
- Heading: a value of 511 indicates that the heading is not available.
- Navigational Status (NavStat): certain values like 0 ("Under way using engine") and 8 ("Under way sailing") are commonly used, but we need to account for the fact that vessels may report more than one status at different times. Moreover, 15 indicates that the status is not defined.

Validation of Geo-spatial Data

Features like latitude and longitude were cross-checked to ensure that they fell within the valid range for geo-spatial coordinates ($-90°$ to $+90°$ for latitude, $-180°$ to $+180°$ for longitude). Any data points outside of this range were flagged as errors and removed from the dataset.

In the following Table 1, the variables in the training set are listed, alongside information on any values that were dropped and the reasons for their exclusion.

Column Name	Action
time	check ranges to ensure data consistency, no data cleaning
vesselid	check ranges to ensure data consistency, no data cleaning
cog	dropped values following the documentation
sog	dropped values following the documentation
rot	dropped values following the documentation
heading	dropped values following the documentation
navstat	dropped values following the documentation and then the entire column due to inconsistency
etaraw	dropped the entire column due to inconsistency
latitude	check ranges to ensure data consistency, no data cleaning
longitude	check ranges to ensure data consistency, no data cleaning
portid	check ranges to ensure data consistency, no data cleaning

Table 1: Training Set

To conclude, *schedules* was not analyzed because of the complexity and uncertainty of the data. The *vessels* dataset was examined and cleaned, but due to a high number of NaN values, it did not contribute significantly to the analysis. On the other hand, the *ports* dataset was very useful, as it was merged with the *training* on `portid`, allowing the retrieval of `portlatitude` and `portlongitude`. Finally, from `portid`, NaN values were dropped initially, and after several attempts to implement other features from *ports* dataset, the remaining columns were dropped, as they were deemed not relevant to the analysis.

3 Feature Engineering

After a deep EDA reported in Section 2, the next step is feature engineering, which is a crucial step in improving the performance of machine learning models. In this stage, we focus on creating new features or modifying existing ones to better capture the underlying patterns in the data, while addressing the challenges identified during the data understanding and cleaning phases. Moreover, in order to use the models described in Section 4, we had to adapt the formats of the given data. For instance, we needed to convert all the data types into the formats expected (int, float, or boolean) by the used predictors (XGBoost, Random Forest, etc...). So we initially worked on the conversion of such variables into something the model could process.

Hash Function

First of all, we applied hashing techniques to handle categorical variables such as `vesselid` and `portid`. For instance, in Listing 1, to `vesselid` and `portid` we decided to use hash-lib's sha256 hashing function to have always the same conversion across different executions.

```python
from sklearn.feature_extraction import FeatureHasher
import hashlib

# To hash an alphanumeric to an integer #
def hash(df):

return int(hashlib.sha256(df.encode('utf-8')).hexdigest(), 16) % (10**18)
```

Listing 1: Function for hashing object types

Label Encoder

We then applied feature encoding to the port information, in Listing 2. Initially, we considered various port details as potentially useful; however, we later found that the model placed greater importance on the port's geographic position such as `portlatitude` and `portlongitude` rather than on general port characteristics.

```python
from sklearn.preprocessing import LabelEncoder

# Initialize LabelEncoder
le_name = LabelEncoder()
le_portlocation = LabelEncoder()
le_un_locode = LabelEncoder()
le_countryname = LabelEncoder()
le_iso = LabelEncoder()

# Apply Label Encoding to categorical columns
ports['name_enc'] = le_name.fit_transform(ports['name'])
ports['portlocation_enc'] = le_portlocation.fit_transform(ports['portlocation'])
ports['un_locode_enc'] = le_un_locode.fit_transform(ports['un_locode'])
ports['countryname_enc'] = le_countryname.fit_transform(ports['countryname'])
ports['iso_enc'] = le_iso.fit_transform(ports['iso'])
```

Listing 2: Function for encode categorical variables

Process Time

In the train set, the `time` column initially had a format of `"2024-05-08 00:03:16"`. To ensure that the time information was properly processed, both in the training and test sets, we first converted the `time` column into a pandas `datetime` object using the following code in Listing 3:

```python
ais_train['time'] = pd.to_datetime(ais_train['time'], format='%Y-%m-%d %H:%M:%S',
    errors='coerce')
test['time'] = pd.to_datetime(ais_test['time'], format='%Y-%m-%d %H:%M:%S', errors=
    'coerce')
```

Listing 3: Convert Time Basic

This conversion was crucial as it allowed us to work with time in a more structured format, enabling time-based feature extraction and calculations.
Initially, we implemented a simple function to format the `time` column into a more compact `int64` representation (YYYYMMDDHHMMSS), making it easier to use for model training. This function, `process_time`, also dropped the original `time` column after conversion, in Listing 4:

```python
def process_time(df):
    # Format the time as YYYYMMDDHHMMSS and convert it to 'int64'
    df['time_formatted'] = df['time'].dt.strftime('%Y%m%d%H%M%S').astype('int64')

return df
```

Listing 4: Process Time 1

However, after further consideration and not obtaining satisfactory results, we decided to refine the time processing. We created the `extract_time_features` function, which not only formatted the time but also extracted the hour from the timestamp and categorized the time of day into four distinct periods: night, morning, afternoon, and evening. This was aimed at helping the model understand potential patterns related to the time of day, in Listing 5:

```python
def extract_time_features(df):
    # Format the time as YYYYMMDDHHMMSS and convert it to 'int64'
    df['time_formatted'] = df['time'].dt.strftime('%Y%m%d%H%M%S').astype('int64')

    # Extract hour from the time for further feature engineering
    df['hour'] = df['time'].dt.hour

    # Define conditions for time of day categorization
    conditions = [
        (df['hour'] >= 0) & (df['hour'] < 6),
        (df['hour'] >= 6) & (df['hour'] < 12),
        (df['hour'] >= 12) & (df['hour'] < 18),
        (df['hour'] >= 18) & (df['hour'] < 24),
    ]
    choices = ['night', 'morning', 'afternoon', 'evening']
    df['time_of_day'] = np.select(conditions, choices)

return df
```

Listing 5: Process Time 2

Also with this function, we did not achieve good results, so we changed approach and we computed the time difference between consecutive observations for each vessel. This was important for understanding the temporal gaps between vessel movements. To do this, we first sorted the dataset by 'vesselid' and 'time', and then calculated the time difference (in seconds) between consecutive records for each vessel, in Listing 6:

```python
# Sort by 'vesselid' and 'time'
train = train.sort_values(by=['vesselid', 'time'])

# Calculate time difference between observations
train['time_difference'] = -train.groupby('vesselid')['time'].diff(-1)
train['time_difference'] = train['time_difference'].dt.total_seconds()
```

Listing 6: Time Difference

This new feature `time_difference`, as reported in Section 5, provided insight into how often data points were collected for each vessel and helped in predicting the vessel's future position.
Moreover, as a final step in processing the time feature, we first created *lag* features to calculate the time difference between the current observation and previous ones at different intervals, with lag values of 1, 50, and 100. Then, we computed the time differences between the current timestamp and the lagged timestamps, resulting in new features:

1. `time_diff_lag1`: The difference in seconds between the current time and the previous observation (`time_lag1`).
2. `time_diff_lag50`: The difference in seconds between the current time and the observation shifted by 50 records (`time_lag50`).
3. `time_diff_lag100`: The difference in seconds between the current time and the observation shifted by 100 records (`time_lag100`).

These lag features, along with the time differences, offer additional context regarding temporal patterns in the data and provide the model with more granular insight into the time intervals between vessel movements at different time scales.

Position

Additionally, since the task is to predict vessel positions, we focused on features related to location (latitude and longitude). Based on certain insights, including those from the `time` column, we introduced new features to improve the model's ability to predict vessel movement, including lags with values of 1, 50, and 100. Similar to how we applied lags to the `time` feature, we also created lag features for latitude and longitude to capture the vessel's positional changes over time, as reported in Listing 7:

```python
# Lag features for Latitude and Longitude
train['latitude_lag1'] = train.groupby('vesselid')['latitude'].shift(1)
train['longitude_lag1'] = train.groupby('vesselid')['longitude'].shift(1)
train['latitude_lag50'] = train.groupby('vesselid')['latitude'].shift(50)
train['longitude_lag50'] = train.groupby('vesselid')['longitude'].shift(50)
train['latitude_lag100'] = train.groupby('vesselid')['latitude'].shift(100)
train['longitude_lag100'] = train.groupby('vesselid')['longitude'].shift(100)
```

Listing 7: Process Lags Position

Distance to Port & Bearing to Port

To further enhance the model's ability to predict vessel movements, we introduced the concept of proximity to the port, which is crucial for understanding vessel behavior as they approach or depart from port locations. We utilized the *ports* dataset to obtain the latitude and longitude of each port, which allowed us to compute two important features: `distance_to_port` and `bearing_to_port`. The distance to the port was calculated using the Haversine function, in Listing 8, which computes the great-circle distance between two points on the Earth's surface, given their latitude and longitude coordinates. This feature provides valuable information about how close a vessel is to the port at any given moment, which can be a key indicator of the vessel's next movement, in Listing 9.

```python
# Distance Port - Position with Haversine distance
def haversine(lat1, lon1, lat2, lon2):
    R = 6371  # in km

    # Difference in radians between latitude and longitude
    dlat = np.radians(lat2 - lat1)
    dlon = np.radians(lon2 - lon1)

    # Compute the distance based on the difference
    a = np.sin(dlat / 2) ** 2 + np.cos(np.radians(lat1)) * np.cos(np.radians(lat2))
        * np.sin(dlon / 2) ** 2
    c = 2 * np.arctan2(np.sqrt(a), np.sqrt(1 - a))

# Output is distance in km
return R * c
```

Listing 8: Haversine Function

```python
# Apply Haversine function for 'distance_to_port'
train['distance_to_port'] = train.apply(lambda row:haversine(row['latitude'], row['
    longitude'], row['port_latitude'], row['port_longitude']), axis=1)
```

Listing 9: Distance to Port

Additionally, we calculated the bearing to the port, in Listing 10, which is the angle between the vessel's current position and the port, in terms of the direction the vessel would need to travel to reach the port. This was computed using the `arctan2` function, which accounts for the longitudinal and latitudinal differences between the vessel's location and the port's coordinates.

```python
# Bearing, heading angle respect to the port
train['bearing_to_port'] = np.arctan2(np.radians(train['port_longitude'] - train['
    longitude']), np.radians(train['port_latitude'] - train['latitude']))
```

Listing 10: Bearing to Port

COG Difference

From the feature `cog`, we also implement a useful variable called `cog_diff`. This features - Listing 11 - is basically the difference between the current course over ground (`cog`) and the bearing towards the port (`bearing_to_port`). The difference is computed as the absolute value of the difference between these two variables, resulting in a new feature called `cog_diff`:

```python
train['cog_diff'] = np.abs(train['cog'] - train['bearing_to_port'])
```

Listing 11: COG Difference

This feature represents the deviation of the vessel's current heading from the ideal heading towards the port. It can provide insights into how much the vessel's trajectory differs from the optimal route to the port.

Moving Average

To enhance prediction accuracy, we implemented a *moving average* smoothing technique, which helps to reduce noise by averaging data points over a specified time window. This method allowed us to smooth out fluctuations in the data, making underlying trends clearer and aiding in more stable model predictions. Specifically, we applied a moving average with a window size ranging from 1 to 10 observations for each vessel. This approach was crucial for achieving one of the best scores recorded in Model 1.
The moving average technique was particularly useful for capturing vessel behavior patterns in relation to port activities, as derived from the ports dataset. By smoothing the data, we were able to create features that significantly contributed to the model's accuracy in predicting vessel movements.

Standard Scaler & PCA

Another important implementation was to standardize the features using `StandardScaler` to ensure they were on a similar scale, as reported in Listing 12. This scaling step helped make sure that each feature contributed equally to the analysis, avoiding any one feature from dominating due to differences in units or ranges. Furthermore, we also tried to implement the Principal Component Analysis (PCA) process (Listing 12), where we began by selecting a set of features likely to be important for predicting vessel movement, including geographic coordinates (latitude and longitude), time and heading-related features (`cog`, `sog`, and `heading`), and engineered features like `latitude_lag` and `longitude_lag` at various intervals (`time_difference`). Additionally, we incorporated port-related information, including `distance_to_port` and `bearing_to_port`, as well as `port_longitude` and `port_latitude`, to capture the relationship between a vessel's position and port locations.

```python
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA

features = [] # Features
target = ['future_latitude', 'future_longitude'] # Target to predict

# Concatenate train and test data
X_all = np.concatenate([train[features], test[features]])

# Initialize the scaler
scaler = StandardScaler()

# Fit the scaler on the combined data (train + test)
X_all_scaled = scaler.fit_transform(X_all)

# Split the scaled data back into train and test
X_train_scaled = X_all_scaled[:len(train)]
X_test_scaled = X_all_scaled[len(train):]

# Initialize PCA
pca = PCA(n_components=5)
pca.fit(X_train_scaled)  # Fit PCA only on training data

# Apply PCA transformation to the scaled data
X_train_pca = pca.transform(X_train_scaled)
X_test_pca = pca.transform(X_test_scaled)

# Model definition
model = RandomForestRegressor(n_estimators=100, random_state=42)
```

```
30
31 # Fit the model using the PCA-transformed training data
32 model_rf.fit(X_train_pca, train[target])
33
34 # Prediction
35 y_pred = model_rf.predict(X_test_pca)
```

Listing 12: General PCA and Scaler implementation

In general, the implementation of the PCA did not lead to a significant improvement in model performance. Instead, standardization alone proved to be more effective for preparing the dataset for predictive modeling, as it maintained the original feature relationships without sacrificing key information. In every submission for the simple approach describe in Section 4.1, the standardization is used.

Iterative Imputer

We experimented with imputing missing values in the test set using the IterativeImputer, as reported in Listing 13, from the sklearn library. This method was applied specifically to the `cog`, `sog`, and `rot` columns to gain insights into the vessel's speed and direction, aiming to enrich our dataset with more consistent and complete information on navigational variables. However, despite these efforts, the imputation process did not yield improved results, as the imputed values failed to significantly enhance the model's predictive performance.

```
1  from sklearn.experimental import enable_iterative_imputer
2  from sklearn.impute import IterativeImputer
3  import pandas as pd
4
5  # Selecting the columns to impute
6  columns_to_impute = ['cog', 'sog', 'rot']
7
8  # IterativeImputer
9  imputer = IterativeImputer(random_state=42)
10
11 # Fit and transform the imputer on the train set
12 train[columns_to_impute] = imputer.fit_transform(train[columns_to_impute])
13
14 # Apply the fitted imputer to the test set
15 test[columns_to_impute] = imputer.transform(test[columns_to_impute])
16
17 # Check
18 print("Imputation completed. Checking for remaining NaN values:")
19 print(train[columns_to_impute].isna().sum())
20 print(test[columns_to_impute].isna().sum())
```

Listing 13: Iterative Imputer

4 Predictors

For all the features mentioned in this section, please refer to the Section 3.

After conducting the Exploratory Data Analysis in Section 2 and Feature Engineering in Section 3, we created the models for our submissions. Over the course of the project, a variety of models have been applied and tried in order to make predictions. In particular, the following have been tried and verified:

- LightGBM;
- XGBoost;
- Random forest.

The best predictions turned out to be provided by the latter. Over time, new features were introduced through our feature engineering process, and different decisions were made regarding the variables to keep into consideration during the prediction phase. Throughout this process,

different models were applied in order to verify their performance under the new settings, and **Random Forest** remained the best model with every provided configuration. For every model, we have the following structures in Listing 14.

```
features = [] # Features which depend on the used model
target = ['future_latitude', 'future_longitude'] # Target to predict

X = train[features]
y_train = train[target]
```

Listing 14: Features and Target

The logic of the prediction, as reported in the Pseudocode 1, is to predict the future latitude and longitude of the next observation for each vessel, after calculating the time difference between the current observation and the timestamp of the row to predict.

Algorithm 1 Predict Future Coordinates for Vessels

Require: *data* ▷ Training set

for each unique vessel in *data* **do**
    $train['time\_difference'] \leftarrow$ Compute the time difference between the current and next observation for each vessel
    Store this time difference in seconds
    $train['future\_latitude'], train['future\_longitude'] \leftarrow$ Derive the future latitude and longitude by associating each observation with the coordinates of the next observation
end for
Define the set of features used for prediction, which includes:
    - Current latitude and longitude
    - Time difference between consecutive observations
    - Previous latitudes and longitudes (lagged values)
    - Previous time differences (lagged values)
Define the prediction target as:
    - Future latitude and future longitude for each observation
return $features, target$

Based on this logic code, we tried two different approaches: simple approach and iterative approach.

4.1 Simple approach

In order to make a decision with regards to the model to adopt, we first used XGBoost (Listing 15), LightGBM (Listing 16), and Random Forest (Listing 17), with only `time` and `vesselid` as features.

```
import xgboost as xgb

# Train XGBoost regressor
xgb_model = xgb.XGBRegressor(objective='reg:squarederror')

# Fit the model
xgb_model.fit(X_train, y_train)

# Prediction
y_pred_xgb = xgb_model.predict(X_test)
```

Listing 15: Simple application of XGBoost

```
import lightgbm as lgb

# Single model for Latitude and Longitude
model_lgb_lat = lgb.LGBMRegressor(objective='regression', n_estimators=100,
    random_state=42)
model_lgb_lon = lgb.LGBMRegressor(objective='regression', n_estimators=100,
    random_state=42)

# Train
model_lgb_lat.fit(X_train, y_train['future_latitude'])
```

```
model_lgb_lon.fit(X_train, y_train['future_longitude'])

# Prediction
X_test = scaler.transform(test[features])
y_pred_lgb_lat = model_lgb_lat.predict(X_test)
y_pred_lgb_lon = model_lgb_lon.predict(X_test)
```

Listing 16: Simple application of LightGBM

The first model we tried was XGBoost. The adoption of such model was what first prompted us to work on the format of the data provided by the datasets (see Section 3). This first phase showed Random Forest to be the most effective model.
After picking said model, we started populating the *test set* by imputing the values we needed, as reported in Section 3. At first, such values were represented by `cog`, `sog`, `rot`, and heading, while, as seen in the EDA Section 2, `navstat` was omitted given it was too "dirty" to be used.
The imputing process described in Section 3 did not give satisfactory results, so the *test* was populated with the last observation in the *test set* relative to each vessel. The best result was achieved using the following code:

```
from sklearn.preprocessing import StandardScaler

# Features and Target
features = ['latitude', 'longitude', 'time_difference', 'vesselid', 'cog', 'sog', '
    heading', 'latitude_lag1', 'longitude_lag1', 'latitude_lag50', 'longitude_lag50
    ', 'latitude_lag25', 'longitude_lag25', 'distance_to_port', 'bearing_to_port',
    'cog_diff', 'port_longitude', 'port_latitude']
target = ['future_latitude','future_longitude']

# Initialize the scaler
scaler = StandardScaler()

X = train[features]
X_train_scaled = scaler.fit_transform(X)
y_train = train[target]

# Model definition
model_rf = RandomForestRegressor(n_estimators=50, random_state=42, n_jobs=-1)
model_rf.fit(X_train_scaled, y_train)

# Prediction
X_test = test[features]
X_test_scaled = scalar.transform(X_test)
y_pred_rf = model_rf.predict(X_test_scaled)
print('prediction done')

# Output
test['latitude_predicted'] = y_pred_rf[:, 0]
test['longitude_predicted'] = y_pred_rf[:, 1]

prediction = test.sort_values(by='ID')
prediction = prediction.reset_index(drop=True)
prediction[['ID','longitude_predicted','latitude_predicted']].to_csv('
    ais_submission.csv', index=False)
print('submission saved')
```

Listing 17: Random Forest - Best Model Simple Approach

4.2 Iterative approach

After analysing the obtained results, we decided to adopt yet another approach. Thus far, we had been using the most recent row (relative to a given `vesselid`) in the train set in order to predict all the instances for that vesselid in the test set.
Now, we instead use that row of the train set to only predict the first row of that vessel id in the test set. Then, the resulting prediction is used to make the predictions relative to the second row of that vessel in the test data, and so on. This gives place to an iterative approach, which results in a further improvement of the model.

Firstly, the adoption of this new method was used alongside new features, called `latitude_lag1`, `longitude_lag1`, `latitude_lag50`, `longitude_lag50`, `latitude_lag100`, `longitude_lag100` which allow the predictions to also take into consideration the `latitude` and the `longitude` of previous rows for a given portid (where the value specified in the name label represents the shift from the row currently being considered).

Later, additional features were added to achieve a better score:
`port_latitude`, `port_longitude`, `distance_to_port`, `bearing_to_port`. The latter refers to the angular direction between the current position of a vessel and a reference port location.

What did not work

When it came to predictors, we tried a variety of approaches which did not end up providing satisfactory results.
Automatic ML
Among them, was Automatic Machine Learning (specifically, AutoGluon), on which we had high expectations but turned out not to be a good fit for this specific problem configuration.
Multi-output on the iterative approach
In a further attempt to improve predictive performance, we applied a Multioutput Regression (Listing 18) approach with an XGBoost model. We used features including positional information (e.g., latitude, longitude), time differences, lags, and derived features like `distance_to_port` and `bearing_to_port`. To implement this, we employed a Multioutput XGBoost regressor, where each output (future latitude and longitude) is treated as a separate regression task. However, the resulting score did not meet our expectations.

```
import xgboost as xgb
from sklearn.multioutput import MultiOutputRegressor

# Features and Target
features = ['latitude', 'longitude', 'time_difference', 'latitude_lag1',  '
    longitude_lag1', 'latitude_lag50', 'longitude_lag50', 'latitude_lag100', '
    longitude_lag100', 'time_diff_lag1', 'time_diff_lag50', 'time_diff_lag100',  '
    distance_to_port', 'bearing_to_port']
target = ['future_latitude', 'future_longitude']

X = train[features]
y_train = train[target]

model = xgb.XGBRegressor(n_estimators=100, random_state=42)
model = MultiOutputRegressor(model)

# Fit
model.fit(X, y_train)
```

Listing 18: XGB Multioutput Regressor on Iterative Approach

RandomSearchCV
To optimize the performance of our Random Forest model, we implemented a hyper-parameter tuning strategy using a randomized search. In this approach, we specified a distribution of potential parameter values and allowed the model to randomly sample combinations within these distributions. The parameter grid included options as reported in Listing 19:

```
from sklearn.model_selection import RandomizedSearchCV

# Parameter distribution for Randomized Search
param_dist = {
    'bootstrap': [True, False],
    'max_depth': [10, 20, 30],
    'max_features': ['sqrt', 'log2'],
    'min_samples_leaf': [1, 2, 4],
    'min_samples_split': [2, 5, 10],
    'n_estimators': [100, 200, 300]
}

# Number of iterations for Randomized Search
n_iter_search = 30

```

```
# Randomized Search for Random Forest
random_search_rf = RandomizedSearchCV(
    estimator=model,
    param_distributions=param_dist,
    n_iter=n_iter_search,
    cv=3,  # 3-fold cross-validation
    scoring='neg_mean_squared_error',
    random_state=42,
    verbose=2
)
```

Listing 19: RandomSearchCV

5 Model Interpretation

In the following section, we present the model interpretation for `short_notebook_1` and `short_notebook_2`, both of which use a Random Forest Regressor with 50 estimators and a random state of 42 as predictors, as reported in Listing 20. However, the models differ in the set of features they utilize.

```
# Features and Target
X = train[features]
y_train = train[target]

# Model definition
model = RandomForestRegressor(n_estimators=50, random_state=42, n_jobs=-1)
model.fit(X, y_train)
```

Listing 20: Default Configuration

5.1 Model 1

To gain insights into the predictive performance of our best model, we conducted an interpretation analysis using *correlation matrix* and *feature importance*. This analysis highlights the impact of individual features on the model's predictions for the target variables `future_latitude` and `future_longitude`. The features used in this model included geographic positions, temporal differences, lagged positions, proximity to port, and smoothed coordinates based on moving averages.

- **Positional Features:** The features `latitude` and `longitude` represent the vessel's current location, serving as primary indicators of its spatial positioning.
- **Time-Based Features:** `time_difference` captures the interval between data points, allowing the model to incorporate temporal information that might affect movement patterns.
- **Lagged Features:** Lagged coordinates such as `latitude_lag1`, `longitude_lag1`, `latitude_lag50`, `longitude_lag50`, `latitude_lag100`, and `longitude_lag100` offer insights into the vessel's recent movements. These features help the model understand the trajectory over short to long intervals, which can enhance its ability to predict future positions.
- **Port Proximity Features:** `distance_to_port` and `bearing_to_port` capture the vessel's distance and relative direction to the nearest port. These features are crucial for understanding vessel behavior near port areas and may signal imminent docking or departure movements.
- **Moving Average Features:** `latitude_ma10` and `longitude_ma10` represent the 10-point moving average of the latitude and longitude, which helps to smooth out fluctuations. These smoothed features provide a more stable representation of the vessel's trajectory, which proved effective for achieving one of the best scores in this submission.

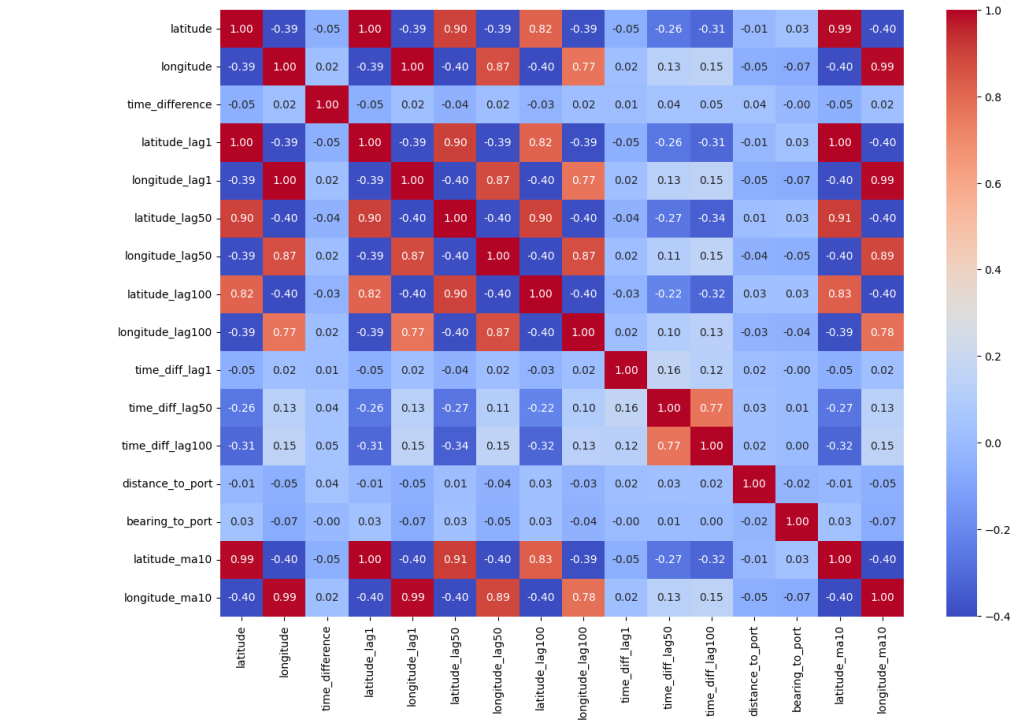

Figure 5: Correlation Matrix Model 1

The correlation matrix above illustrates the correlations between various features used in the model. Here's some considerations:

- The features `latitude_ma10` and `longitude_ma10`, representing a 10-step moving average of position, show near-perfect correlations with latitude and longitude, respectively. This is expected since moving averages smooth out recent values.
- The `distance_to_port` and `bearing_to_port` features are only weakly correlated with other features, indicating that they provide unique and complementary information. These features capture the vessel's proximity and direction relative to the port, which could be crucial for predicting behaviors associated with port arrivals or departures. Including these distinct features likely helps the model understand location-specific patterns that are not evident from latitude and longitude alone.
- It suggests that while many features are informative, some could be redundant due to high correlations, especially among the latitude and longitude lags. The idea was to select a few key lags to use the best lags for the task.

Furthermore, in the following Figure 6, it is reported the feature importance of the Model 2.

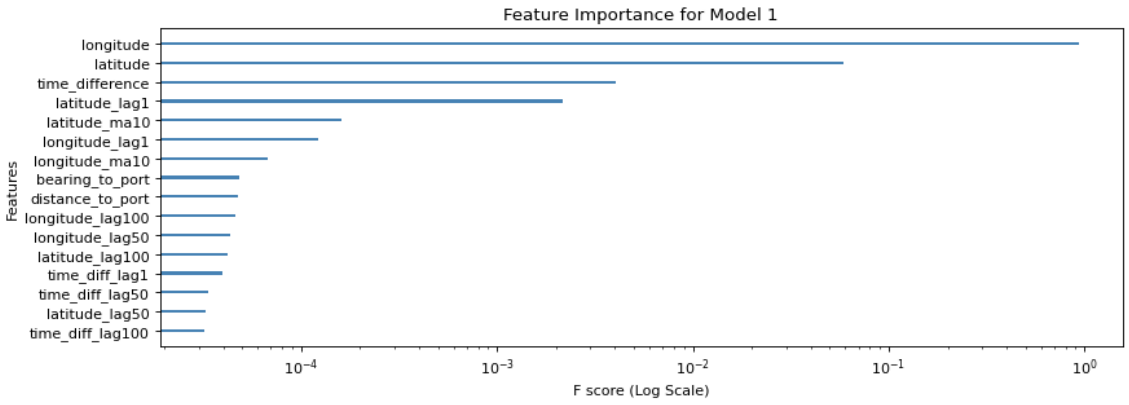

Figure 6: Feature Importance Model 1

To further analyze the spatial accuracy of our model's predictions, we leveraged the `geopandas` library. This allowed us to visualize the predicted positions of vessels on a map, providing a clear spatial context for each prediction. By converting the model's predicted latitude and longitude values into geometric points, we could plot these locations, as reported in Figure 7. The F-score metric quantifies each feature's impact by measuring how frequently it appears and what is the role in the model's predictions. In general, a thorough feature engineering process was carried out, particularly focusing on the deductions made through the introduction of the time window. This approach allowed us to better capture temporal patterns in the vessel's movement, enhancing the model's ability to predict future positions. The creation of lagged features and time differences provided valuable insights into the vessel's trajectory, improving overall predictive performance.

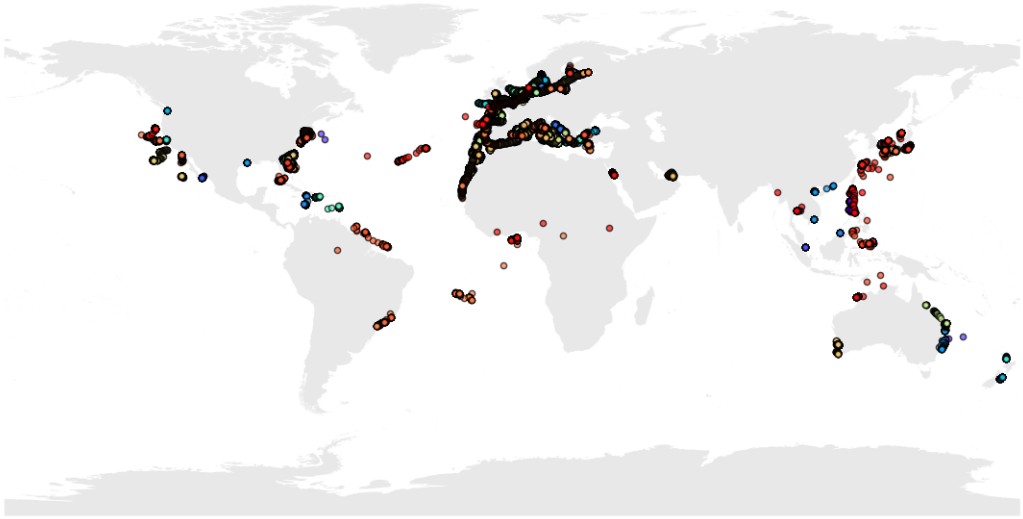

Figure 7: Distribution of Predictions for Model 1

5.2 Model 2

In the second model, we designed the feature set with a focus on including lagged positions and emphasizing the vessel's trajectory towards the port. To gain insights into the predictive performance of the second model, we conducted an interpretation analysis using *feature importance*, as reported in Figure 8. To achieve this, we retained `port_latitude` and `port_longitude` as features, alongside lagged coordinates (`latitude_lag1`, `longitude_lag1`, `latitude_lag50`, `longitude_lag50`, `latitude_lag100`, and `longitude_lag100`) and time differences. Our intention was to capture both the recent path of each vessel and its relation to the port's position.

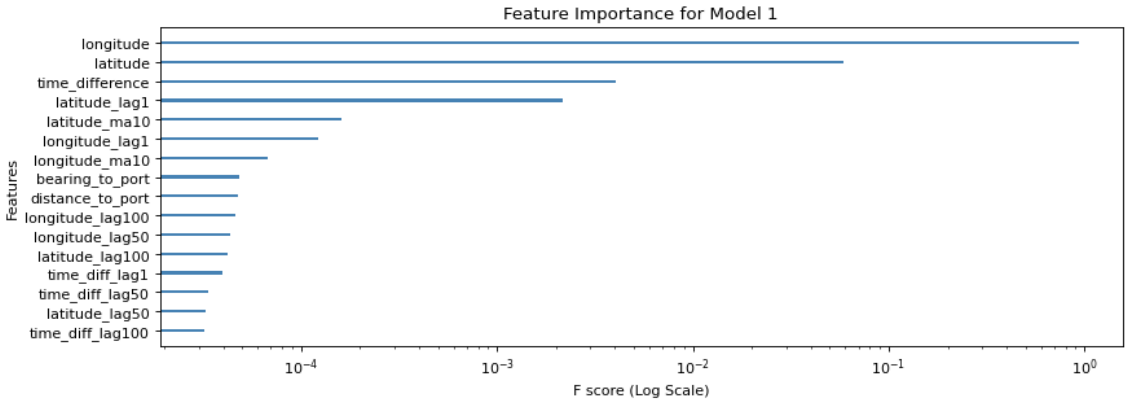

Figure 8: Feature Importance Model 2

The expectation was that by incorporating `port_latitude` and `port_longitude`, the model would gain a better understanding of the vessel's approach towards the port, which might have contributed to more accurate predictions. However, this approach did not yield the desired results. As seen in Model 5.1, excluding `port_latitude` and `port_longitude` while prioritizing the vessel's recent behavior over a 10-instance moving window achieved better predictive performance. This indicates that focusing on the vessel's short-term movement history, rather than its precise position relative to the port, provided more useful information for the model. However, to conclude, we also tried to implement in the final submission, we experimented with increasing the number of lagged features, given the significance of historical data in predictions. We extended the lags up to 12 instances, as well as additional intervals like 50 and 99, aiming to capture longer-term movement patterns, as shown in Listing 21.

```
# Lag features for Latitude and Longitude
for i in range(1, 13):  # Lag da 1 a 12
    train[f'latitude_lag{i}'] = train.groupby('vesselid')['latitude'].shift(i)
    train[f'longitude_lag{i}'] = train.groupby('vesselid')['longitude'].shift(i)

train['latitude_lag50'] = train.groupby('vesselid')['latitude'].shift(50)
train['longitude_lag50'] = train.groupby('vesselid')['longitude'].shift(50)

train['latitude_lag99'] = train.groupby('vesselid')['latitude'].shift(99)
train['longitude_lag99'] = train.groupby('vesselid')['longitude'].shift(99)

# Lag features for Time
for i in range(1, 13):  # Lag da 1 a 12
    train[f'time_lag{i}'] = train.groupby('vesselid')['time'].shift(i)

train['time_lag50'] = train.groupby('vesselid')['time'].shift(50)
train['time_lag99'] = train.groupby('vesselid')['time'].shift(99)

# Time Difference
for i in range(1, 13):  # Lag da 1 a 12
    train[f'time_diff_lag{i}'] = (train['time'] - train[f'time_lag{i}']).dt.
        total_seconds()

train['time_diff_lag50'] = (train['time'] - train['time_lag50']).dt.total_seconds()
train['time_diff_lag99'] = (train['time'] - train['time_lag99']).dt.total_seconds()
```

Listing 21: Last Submission

However, this approach did not result in improved performance, suggesting diminishing returns from deeper lagged features.