A PROJECT REPORT
on

“Track Classification of Ship Data”
Submitted to

Centre for Artificial Intelligence and Robotics(DRDO)

In Partial Fulfilment of the Requirement
for the Award of Internship
BY

AMIT KUMAR

under the guidance of
Dr. Loveneet Kumar

Acknowledgements

I would like to express my deepest gratitude to Dr. Loveneet Kumar sir of the Defence Research and Development Organisation (DRDO) for his invaluable guidance and support throughout this project. His expertise, insights, and encouragement were instrumental in the successful completion of this work.

Dr. Kumar's dedication to excellence and his willingness to share his knowledge and experience were a constant source of inspiration. His constructive feedback and thoughtful suggestions greatly enhanced the quality of this project, and his patience and understanding were greatly appreciated.

I am immensely grateful for the opportunity to work under his mentorship and for the generous support he provided at every stage of this project. This project would not have been possible without his assistance and commitment.

Thank you, Dr. Loveneet Kumar sir, for your unwavering support and for helping me navigate the complexities of this project.

ABSTRACT

Track Classification of Ship Data Using AIS with Time Series Analysis.

Automatic Identification System (AIS) data provides continuous, real-time tracking information of ships, which is critical for maritime traffic management and safety. This study explores the application of time series analysis for the classification of ship tracks using AIS data. By treating AIS data as time series, we extract temporal features such as velocity, heading, and positional changes over time. Advanced time series analysis techniques, including autoregressive integrated moving average (ARIMA) models, long short-term memory (LSTM) networks, and dynamic time warping (DTW), are employed to capture the underlying patterns and trends in ship movements. These techniques enable the differentiation of ship tracks into various categories such as commercial, fishing, and recreational activities. The proposed methodology is validated on a comprehensive AIS dataset, showcasing its ability to accurately classify ship tracks with high precision. This approach not only enhances maritime situational awareness but also aids in the detection of anomalous behavior, contributing to maritime safety and security. The findings demonstrate the potential of time series analysis as a powerful tool for the intelligent monitoring and classification of maritime traffic using AIS data.

CONTENTS

Project Overview

INTRODUCTION

Maritime navigation and safety are paramount in ensuring efficient and secure operations in the shipping industry. One critical aspect of maritime safety is the accurate classification of ship tracks, which involves monitoring and analyzing the paths taken by vessels. This project aims to leverage advanced time-series analysis techniques, specifically Fb Prophet and Long Short-Term Memory (LSTM) models, to classify ship tracks based on data from the Automatic Identification System (AIS).

My model helps in detecting the pattern of ship navigation and tracking and forecasting .

The coding enviornment used is Python and Machine Learning Technologies and Time-Series Analysis : Fbprophet library and Deep Learning Model : LSTM(Long Short Term Memory).

SCOPE AND OBJECTIVES

The scope of this project encompasses the following key areas:

Data Collection: Acquire ship tracking data from various sources, including Automatic Identification Systems (AIS), radar systems, and satellite imagery.Ensure data covers a wide geographical range and diverse types of ships.

Data Integration and Preprocessing: Integrate data from different sources into a unified dataset. Preprocess the data to handle missing values, outliers, noise, and ensure consistency in formats and coordinate systems.

Feature Engineering: Identify and extract relevant features such as speed, heading, position, timestamps, and movement patterns. Create additional features that may enhance model performance, such as environmental conditions.

Model Development: Develop machine learning models for classifying ship tracks into categories like cargo, passenger, fishing, and military vessels.Experiment with different algorithms, including both supervised and unsupervised learning methods.

Model Training and Validation: Train models using historical data and validate their performance through cross-validation and other techniques.Optimize model parameters to achieve high classification accuracy.

System Deployment: Implement the trained models in a real-time monitoring system.Ensure seamless integration with existing maritime surveillance infrastructure.

Performance Monitoring and Continuous Improvement: Continuously monitor the system's performance and update models with new data to maintain accuracy.Implement mechanisms for user feedback and operational insights to refine the system.

Objectives

The primary objectives of this project are:

Accurate Classification: Develop a system capable of accurately classifying ship tracks in real-time to enhance maritime situational awareness.

Enhanced Maritime Safety: Improve the ability to monitor and manage ship movements, thereby enhancing maritime safety and reducing the risk of accidents.

Detection and Prevention of Illegal Activities: Enhance the detection and response capabilities for illegal activities at sea, such as smuggling and piracy.

Optimized Maritime Traffic Management: Provide tools for better maritime traffic management and route planning, leading to more efficient and economical operations.

Scalability and Flexibility: Ensure the system can handle large volumes of data and is flexible enough to incorporate additional data sources and features in the future.

Stakeholder Collaboration: Facilitate collaboration with maritime authorities, shipping companies, and other stakeholders to standardize track classification methodologies and share best practices.

Existing System & Proposed System

To address the time series classification of ship data, we can break the project into two main sections: the existing system and the proposed system. The **existing** system involves data collection from various sensors on the ship, such as GPS, speed, direction, engine parameters, and weather conditions, with data recorded at regular intervals resulting in a time series dataset. This data is stored in central databases or distributed storage systems, using formats like SQL databases, NoSQL databases, or cloud storage solutions. Basic preprocessing steps, including data cleaning, handling missing values, and normalization, are performed with limited feature engineering. Classification in the current system is typically done using simple models or rule-based systems, employing basic machine learning models like decision trees or logistic regression. Visualization and reporting are achieved through basic dashboards and tools like Excel, basic BI tools, or custom dashboards.

The **proposed** system aims to enhance the existing capabilities by integrating more advanced sensors and IoT devices for richer and more granular data collection, with real-time capabilities for immediate processing. Advanced data storage solutions, such as time series databases or cloud-native solutions, will be adopted to ensure scalability, robustness, data redundancy, and high availability. The preprocessing pipeline will be upgraded with sophisticated techniques, including advanced cleaning, interpolation of missing values, robust normalization methods, and advanced feature engineering using domain knowledge and automated feature extraction techniques. Sophisticated classification models, such as Recurrent Neural Networks (RNNs), Long Short-Term Memory Networks (LSTMs), and Convolutional Neural Networks (CNNs) adapted for time series data, will be implemented along with ensemble methods and hybrid models for improved accuracy and robustness.

Real-time processing and classification will be facilitated through the implementation of streaming data processing frameworks like Apache Kafka and Apache Flink, ensuring immediate classification results. Advanced visualization tools and dashboards using platforms like Tableau, Power BI, or custom web-based dashboards will be utilized, along with real-time reporting capabilities and alert systems for anomalies or important classifications. Continuous evaluation of model

performance using metrics like accuracy, precision, recall, and F1-score will be established, along with a feedback loop to incorporate new data and continuously improve the models.

Implementation steps include data assessment to evaluate existing data quality and identify additional requirements, selection of an appropriate technology stack for data storage, processing, and modeling, and the development and training of advanced machine learning and deep learning models with hyperparameter tuning and model validation. System integration will ensure seamless data flow from collection to classification and visualization, followed by deployment in a production environment with continuous monitoring and updates. Finally, a feedback mechanism will be implemented for continuous learning and model improvement, ensuring regular system updates based on new data and evolving requirements. This comprehensive approach will significantly enhance the system's capability to classify ship data accurately and efficiently in real time.

Project Analysis

The classification of ship data through time series analysis is crucial for improving operational efficiency, safety, and predictive maintenance in maritime operations. This project aims to develop a robust system that can classify ship data accurately and in real-time by leveraging advanced data collection methods, sophisticated machine learning models, and real-time processing frameworks.

Time series analysis is a statistical approach that entails gathering data at consistent intervals to recognize patterns and trends. This methodology is employed for making well-informed decisions and precise forecasts by leveraging insights derived from historical data.

FBPROPHET Library :

Prophet is a procedure for forecasting time series data based on an additive model where non-linear trends are fit with yearly, weekly, and daily seasonality, plus holiday effects. It works best with time series that have strong seasonal effects and several seasons of historical data. Prophet is robust to missing data and shifts in the trend, and typically handles outliers well.

1. *Key Components and Features*

- Trend: Captures the overall direction of the data over time (upward or downward).
- Seasonality: Accounts for periodic fluctuations (daily, weekly, yearly) in the data.
- Holidays and Special Events: Allows for the inclusion of known events that can affect the time series.
- Changepoints: Points where the trend changes significantly.
- Uncertainty Intervals: Estimates the uncertainty of the forecast

2. Data Preparation

Prophet requires the input data in a specific format:

- 'ds': The date column (datetime format).
- 'y': The metric to be forecasted (numeric format).

Need of Facebook Prophet

Humans require it because, despite the fundamental decomposable additive model's seemingly straightforward appearance, the terms' calculations include complex mathematics. A lack of expertise can result in inaccurate forecasting, which could have catastrophic consequences in the actual world. Thus, we will use Prophet to automate this process. But first, let's look at how Prophet anticipates the data to grasp the math underneath this process and how it functions in reality.

Prophet provides us with two models (however, newer models can be written or extended according to specific requirements).

1. Logistic Growth Model

2. Piece-Wise Linear Model

By default, Prophet uses a piece-wise linear model, but it can be changed by specifying the model. Choosing a model is delicate as it is dependent on a variety of factors such as company size, growth rate, business model, etc., If the data to be forecasted, has saturating and non-linear data (grows non-linearly and after reaching the saturation point, shows little to no growth or shrink and only exhibits some seasonal changes), then logistic growth model is the best option. Nevertheless, if the data shows linear properties and had growth or shrink trends in the past then, the piece-wise linear model is a better choice. The logistic growth model is fit using the following statistical equation,

$$g(t) = \frac{C}{1+e^{-k(t-m)}}$$

where,
c is the carrying capacity
k is growth rate
m is an offest parameter

The piece-wise linear model is fit using the following statistical equations,

$$y = \begin{cases} \beta_0 + \beta_1 x & x \leq c \\ \beta_0 - \beta_2 c + (\beta_1 + \beta_2)\, x & x > c \end{cases}$$

where c is the trend change point (it defines the change in the trend).? is the trend parameter and can be tuned as per requirement for forecasting.

Implementation

1. Install and Import Necessary Libraries

2. Loading of Dataset.
3. Data Preprocessing and Visualization
4. Create an even interval function to store values sequentially for a time-frame.
5. Create Plot Forecast to check the forecast.
6.

a) Initialize combined_data DataFrame:Create an empty DataFrame to hold combined time series data.

b) Menu Display and User Choice:Continuously display a menu to the user with options to generate different types of time series data (SOG, COG, Latitude, Longitude) or exit the program.

c) Handle User Choice:If the user chooses to exit, break the loop and exit the program.If the user chooses a valid option (1-4), prompt the user to enter an MMSI (Maritime Mobile Service Identity) number to filter the dataset.Based on the user's choice, set the relevant columns and noise levels for the time series generation.

d) Generate Time Series Data:Call the create_time_series function with the filtered dataset and specified parameters to generate time series data.
Append the generated time series data to the combined_data DataFrame.

e) Split Data into Training and Testing Sets:Split the generated time series data into training (80%) and testing (20%) sets without shuffling to preserve the time sequence.

f) Prepare Data for Prophet Model:

Rename the relevant column to 'y' and ensure the 'ds' column is in datetime format for compatibility with the Prophet model.

g) Train Prophet Model:

Initialize and train a Prophet model using the training data.

h) Create Future Dataframe for Prediction:Create a future dataframe with the same frequency and period as the test data for making predictions.

i) Make Predictions:Use the Prophet model to make predictions on the future dataframe.

j) Evaluate Model Performance:Calculate and print the mean squared error (MSE) between the actual and predicted values.
Compute the deviation of the actual values from the predicted values and calculate the standard deviation of the deviations.

k) Calculate and Plot Moving Average of Deviation:Calculate the moving average of the deviation using a specified window size.Plot the deviation and moving average over time for visualization.

Explanation of Key Functions and Terms:

i. create_time_series: A function that generates time series data for the specified columns within a given date range and interval.
ii. Prophet: A forecasting tool developed by Facebook that is particularly effective for time series data.
iii. mean_squared_error: A metric used to evaluate the performance of the model by measuring the average squared difference between actual and predicted values.
iv. Standard Deviation (sd): A measure of the amount of variation or dispersion in a set of values.

v. Moving Average: A technique used to smooth out short-term fluctuations and highlight longer-term trends or cycles.

Workflow Summary:
The script prompts the user to select the type of time series data to generate and an MMSI number. It then generates the corresponding time series data, trains a Prophet model, makes predictions, evaluates the model's performance, and visualizes the deviations and moving average. This process continues until the user decides to exit the program.

7 . Merge the datasets

```
In [99]: df1_2 = pd.concat([combined_data, combined_data1], axis=0)
```

8.

- Time Series Generation :The function create_time_series12 creates time series data for the specified columns, either using existing data from df1_2 or generating synthetic data with noise.This allows for creating a consistent time series dataset with a specified frequency of data points.

- · Model Training and Forecasting: The time series data is split into training and test sets.The Prophet model is trained on the training set and forecasts future values based on this training.The results of the forecast are compared with the actual test data.

- Evaluation and Visualization: The Mean Squared Error (MSE) is calculated to measure the accuracy of the forecast.
  The standard deviation of the deviation between the forecast and actual values provides insight into the forecast's reliability.Visualizing the deviation helps identify periods where the model's forecast deviates significantly from the actual data.

9. Repeat this process for all the datasets, and check if the deviation is less than 5% or not.

Long Short Term Memory(LSTM)

Long Short-Term Memory (LSTM) networks are a type of Recurrent Neural Network (RNN) designed to capture long-term dependencies and

handle the vanishing gradient problem. They were introduced by Hochreiter and Schmidhuber in 1997 and have since become a fundamental building block for many sequence-based tasks.

Why LSTM?

Traditional RNNs suffer from the vanishing gradient problem, which makes it difficult to learn long-range dependencies. This problem arises because gradients of the loss function with respect to the parameters tend to either vanish (approaching zero) or explode (approaching infinity) as they are backpropagated through time. LSTMs address this issue by introducing a more complex unit structure that allows the network to maintain and update a cell state over long sequences.

LSTM Architecture

An LSTM network consists of a series of LSTM cells. Each LSTM cell contains several key components:

Cell State (): The cell state is the memory of the network. It is designed to carry information across many time steps, and can be updated or forgotten over time.

Hidden State (): The hidden state is the output of the LSTM cell at each time step.

Forget Gate (): This gate decides what information from the cell state should be discarded or kept. It is a sigmoid layer that takes the previous hidden state () and the current input () as inputs.

Input Gate (): This gate decides which new information will be added to the cell state. It also consists of a sigmoid layer.
Candidate Cell State (): This is the new candidate values that could be added to the cell state, created by a tanh layer.
Output Gate (): This gate decides the next hidden state. It is a sigmoid layer.

Working Mechanism

1. Initialization: At time t=0, initialize the cell state C0, and hidden state h0.
2. Sequence Processing: For each time step ttt in the sequence:
Compute the forget gate ft.
Compute the input gate it.
Compute the candidate cell state Ct.
Update the cell state Ct
Compute the output gate ot.
Update the hidden state ht.

Advantages of LSTM

- Long-Term Dependency: LSTMs can learn and remember over long sequences.

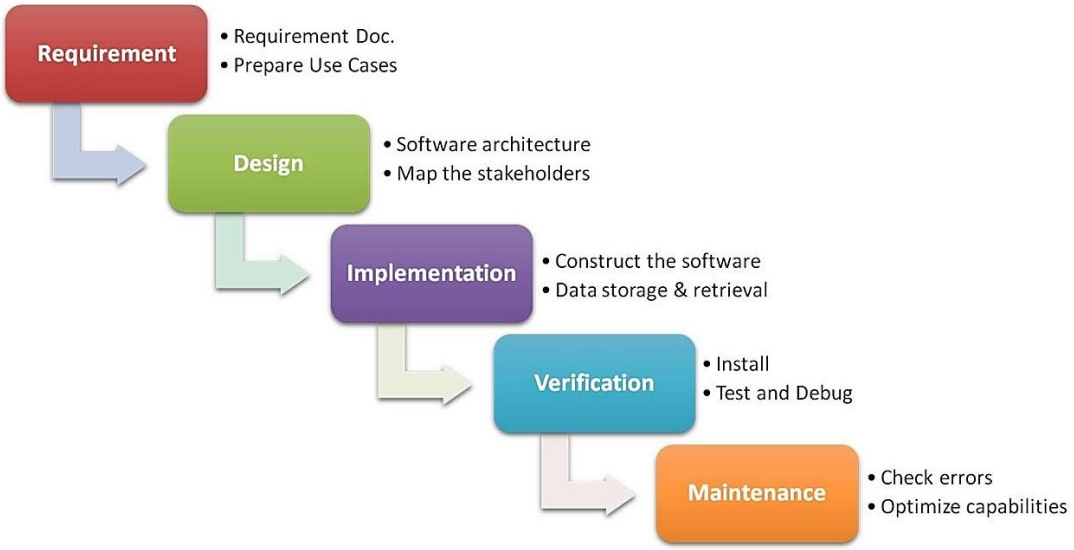
- Gradient Flow: LSTMs mitigate the vanishing gradient problem, allowing gradients to flow effectively through time.

- Flexibility: LSTMs are flexible and can be used in various applications like language modeling, speech recognition, and time series prediction.

Applications of LSTM

- Natural Language Processing (NLP): LSTMs are widely used in tasks like machine translation, text generation, and sentiment analysis.
- Speech Recognition: LSTMs help in recognizing and processing speech patterns over time.
- Time Series Forecasting: LSTMs are used to predict future values based on historical data.
- Video Analysis: LSTMs can analyze video sequences frame by frame, maintaining context over time.

Project Lifecycle



Description

The waterfall Model is a linear sequential flow. In which progress is seen as flowing steadily downwards (like a waterfall) through the phases of software implementation. This means that any phase in the development process begins only
if the previous phase is complete. The waterfall approach does not define the process to go back to the previous phase to handle changes in requirement. The waterfall approach is the earliest approach that was used for software development.

Project Design


PROJECT
PROJECT INITIATION
PROJECT INITIATION
DATA COLLECTION
MODEL
FB
FB PROPHET
MODEL
MODEL EVALUATION
MODEL EVALUATION

Project Snapshots

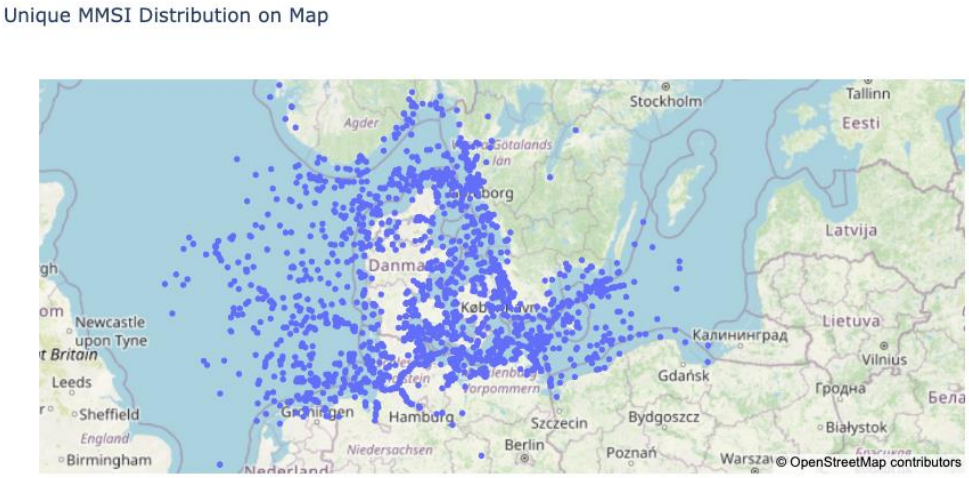


Fig 1. Unique Distribution on Map

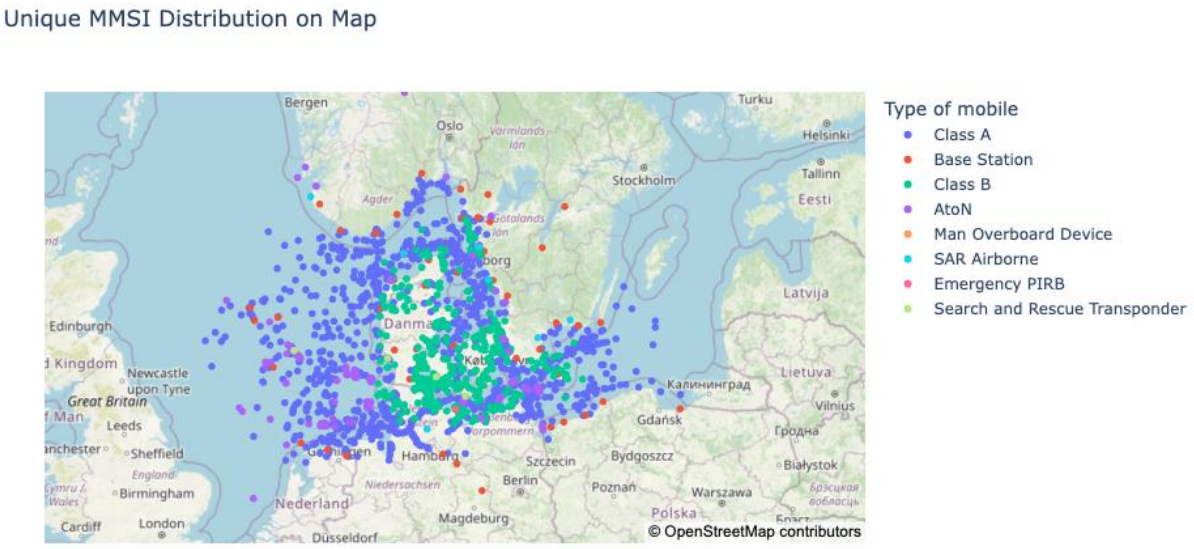


Fig 1.1. Unique DIstribution on Map (Type of Mobile)

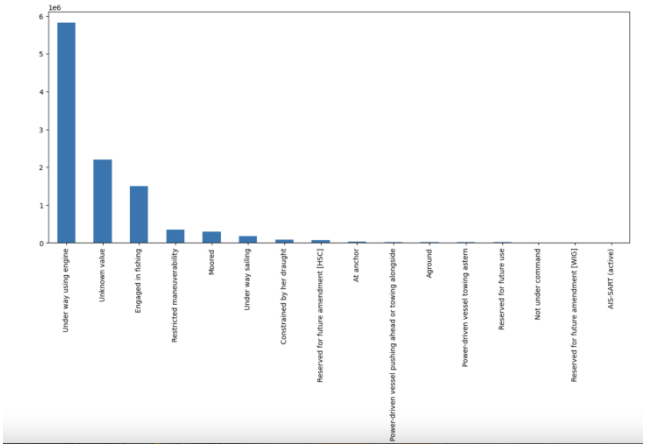


Fig 2. Navigational Status

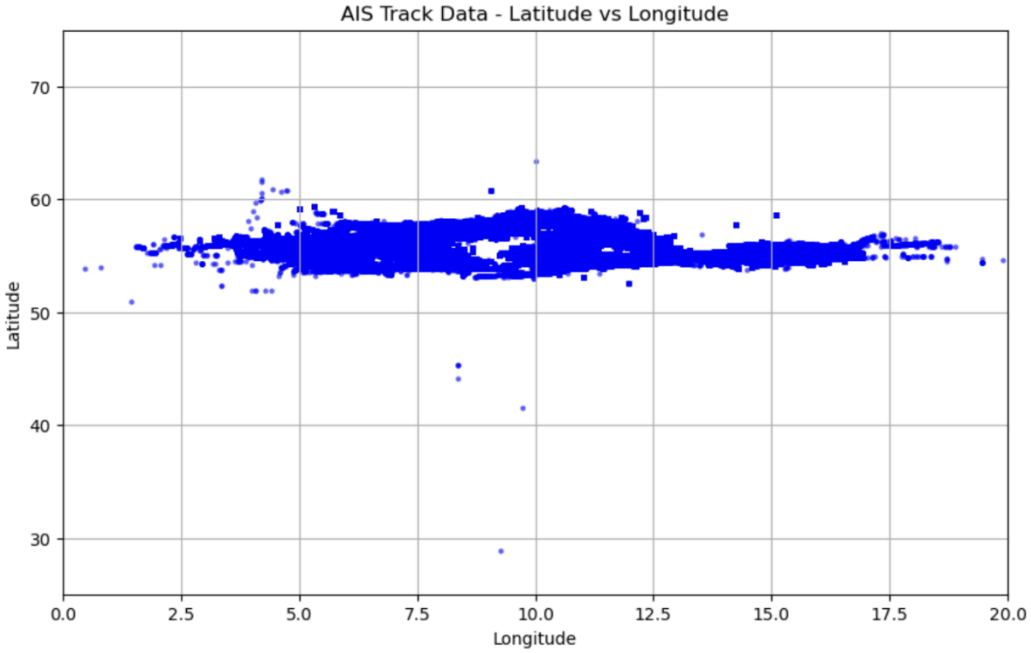


Fig 3. AIS Track Data - Latitude vs Longitude

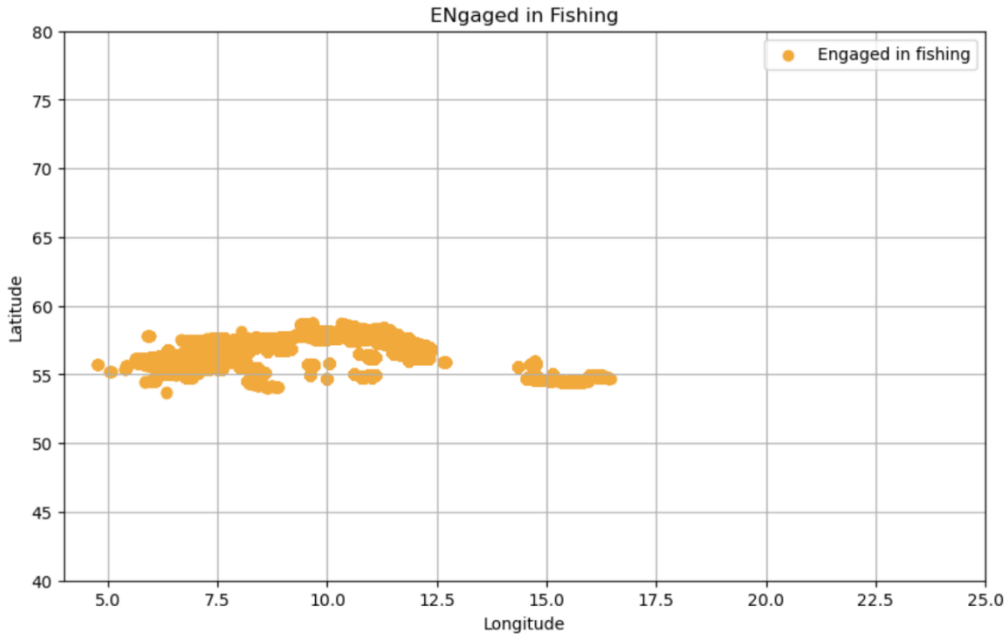


Fig 3.2 Engaged in Fishing

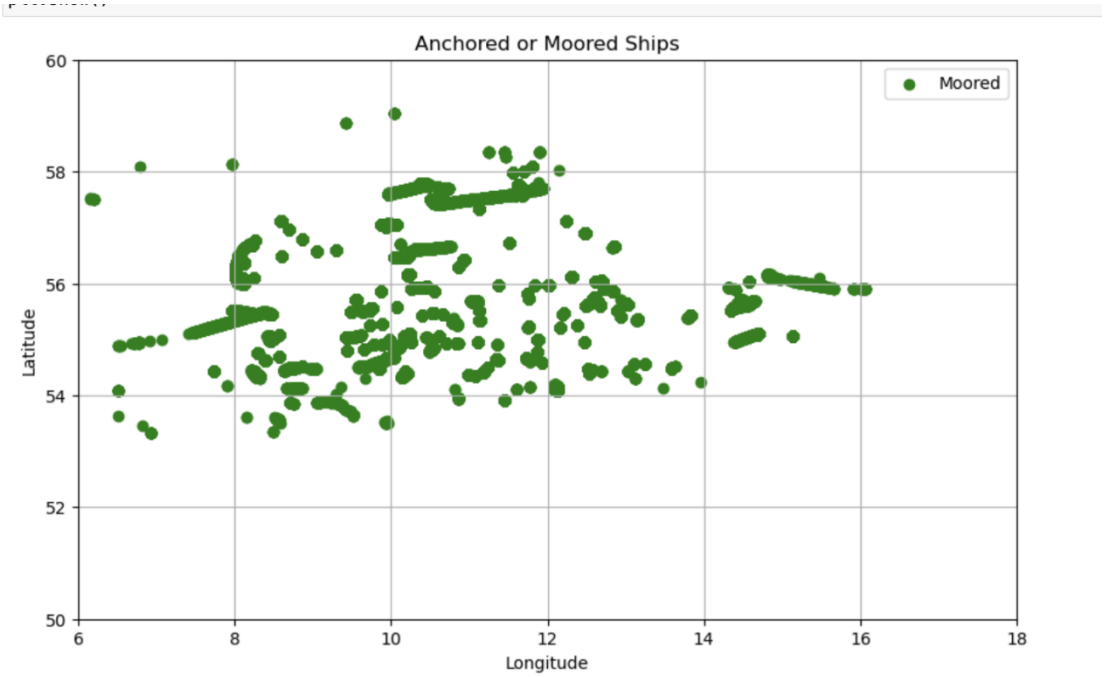


Fig 3.3 Anchored or Moored Ships

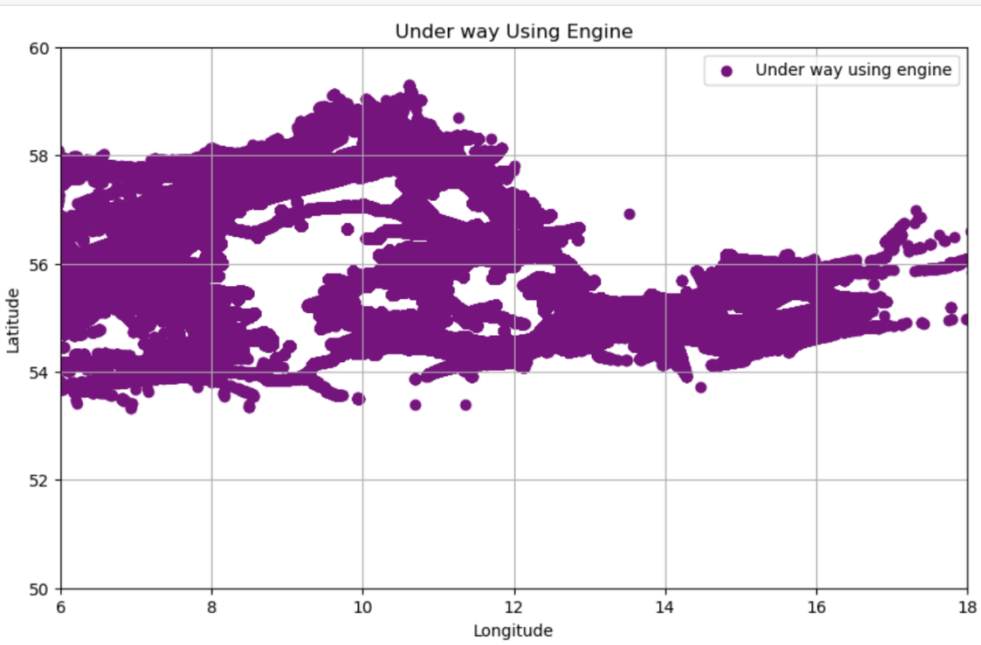


Fig 3.4 Under way Using Engine

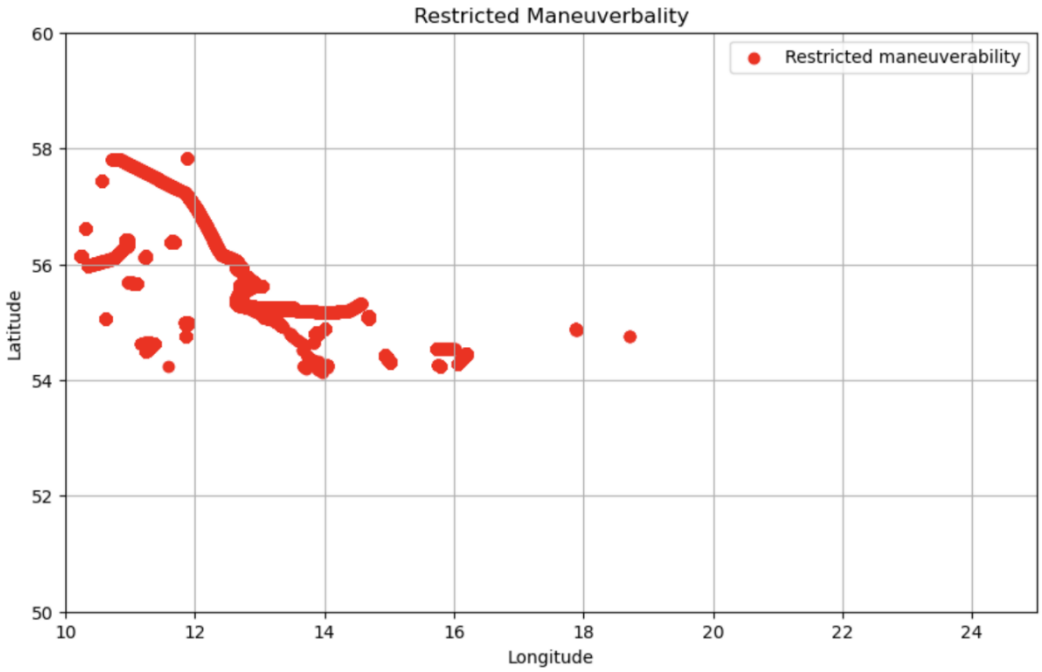


Fig 3.5 Restricted Maneuverbality

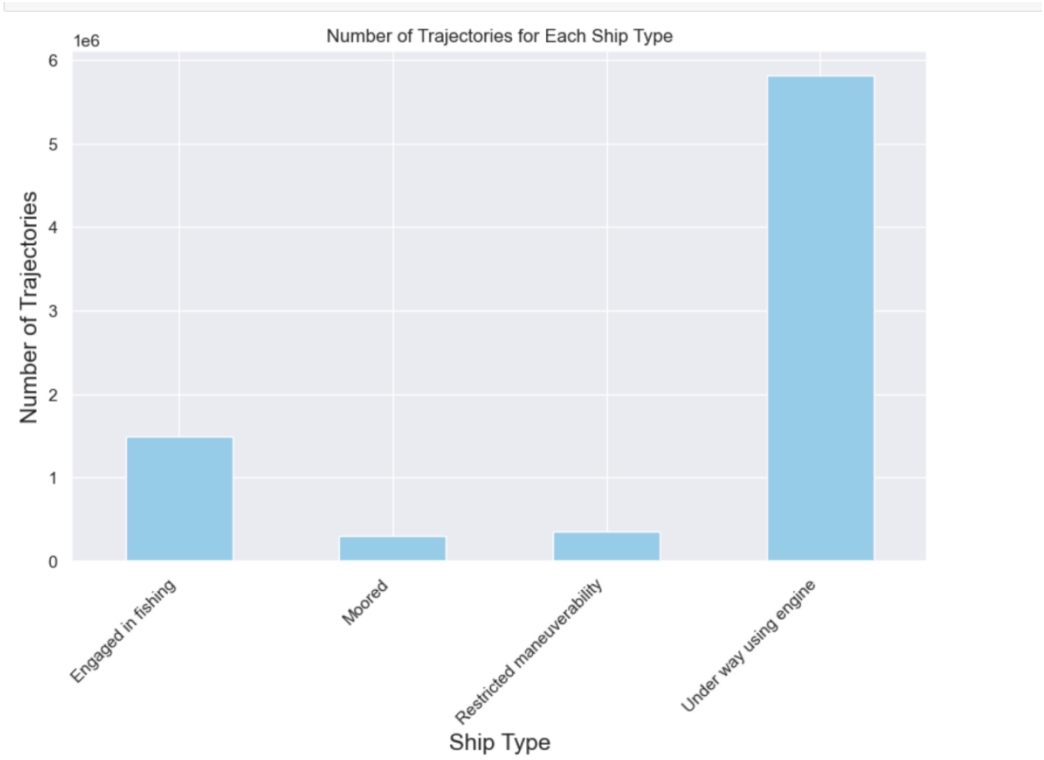


Fig 4. Number of Trajectories for Each Ship Type

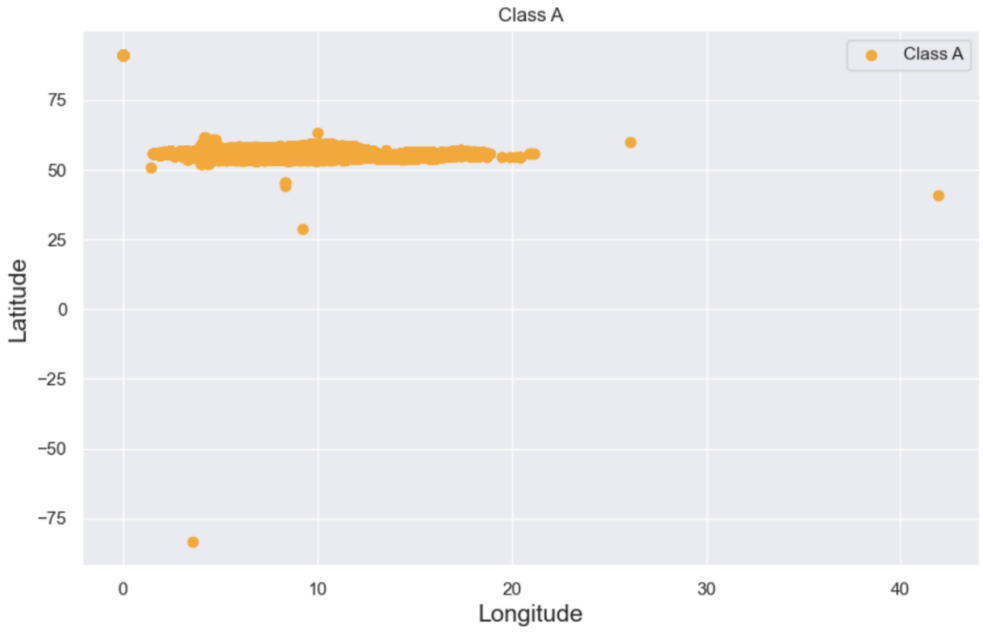


Fig 5. Class A

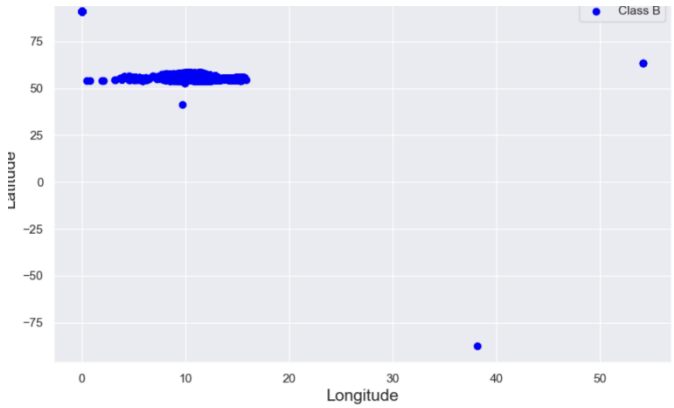


Fig 5.2 Class B

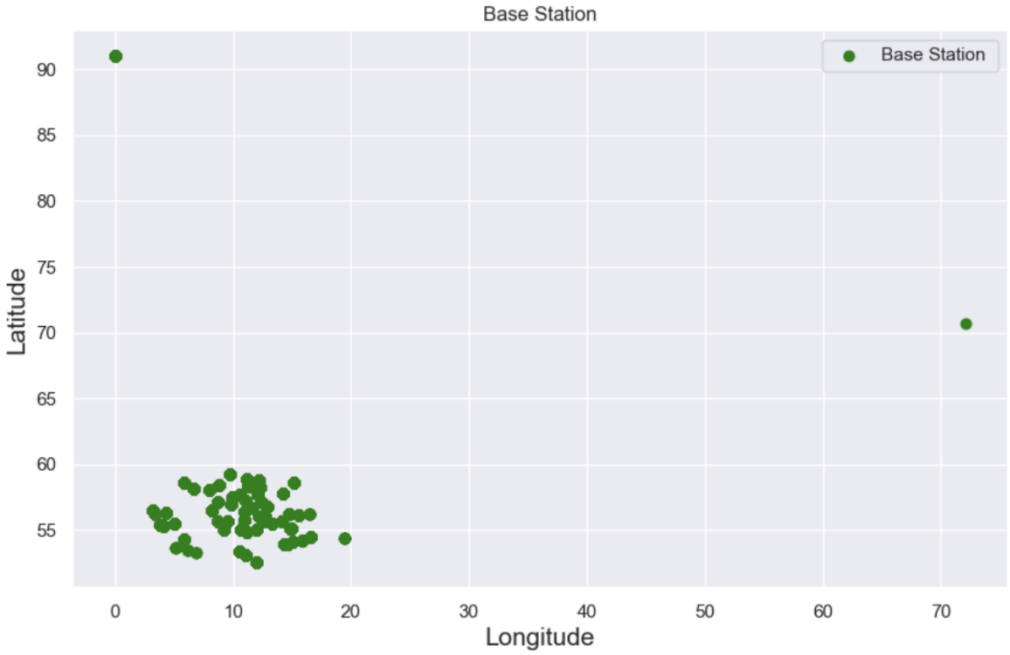


Fig 5.3 Base Station

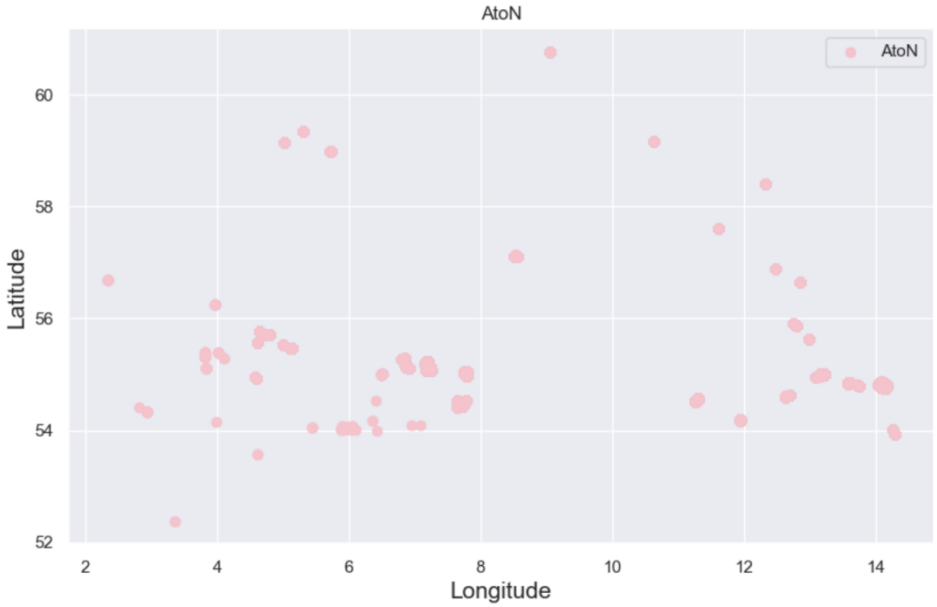


Fig 5.4 Aton

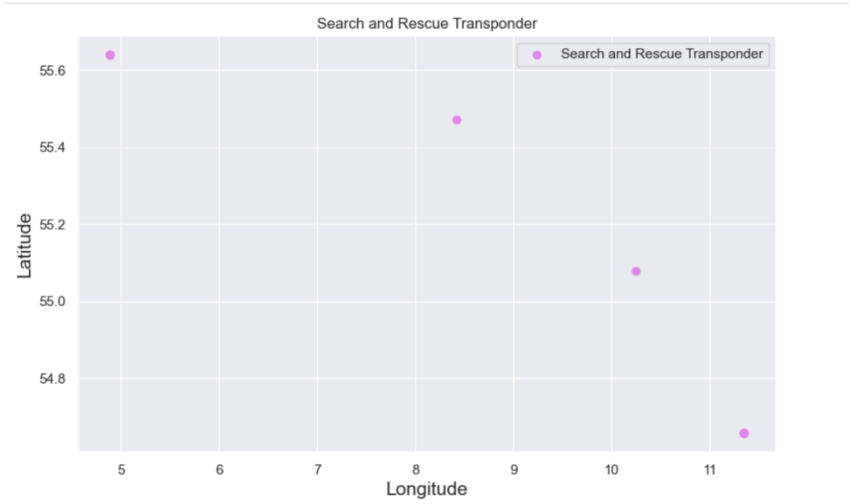


Fig 5.5 Search and Rescue Transponder

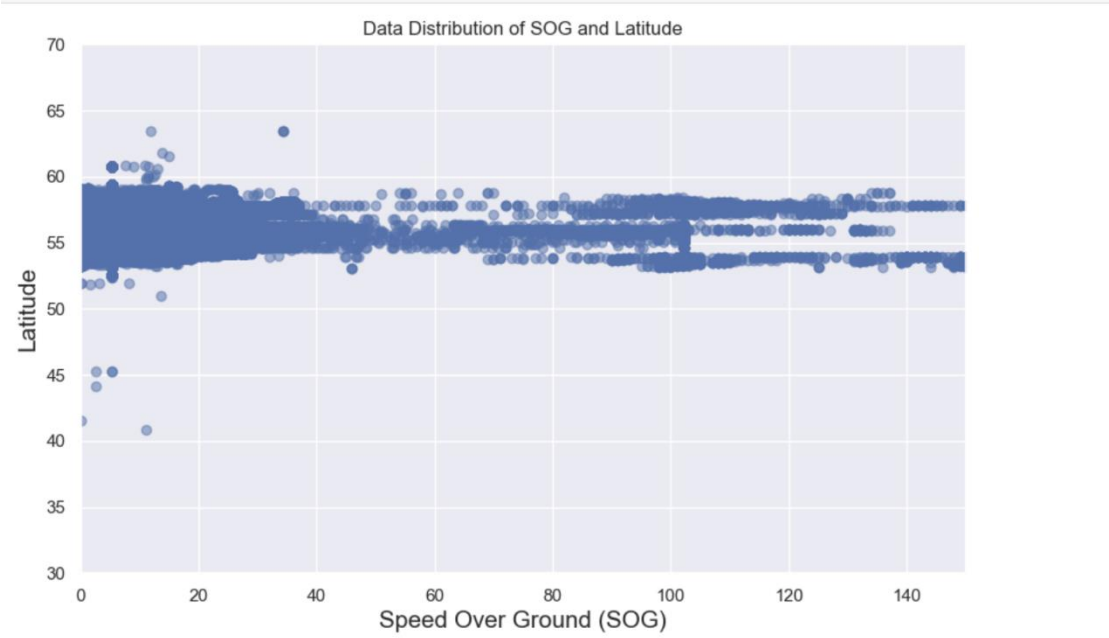
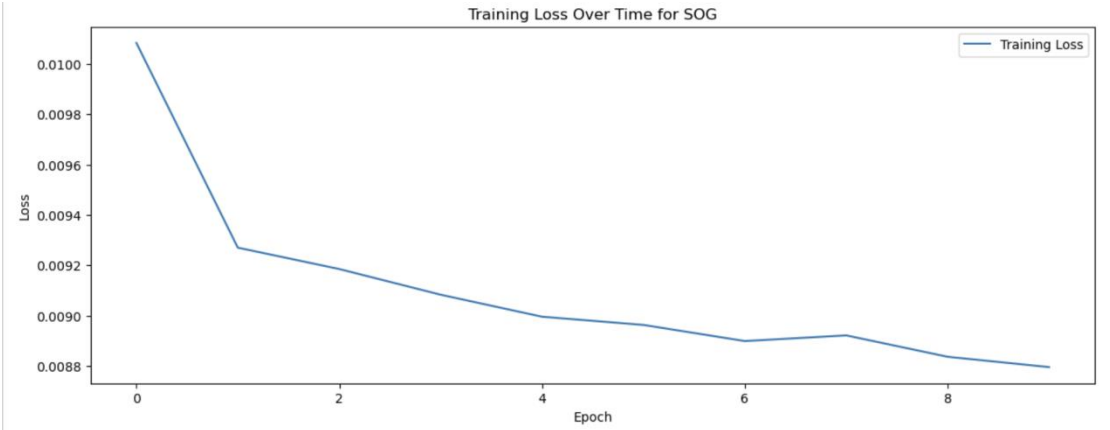


FIg 6. Data Distribution of SOG and Latitude

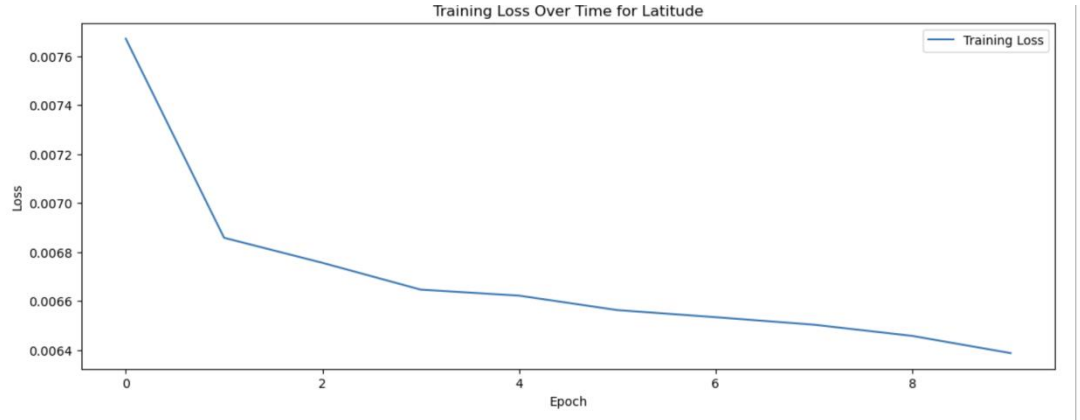



```
466/466 ━━━━━━━━━━━━━━━━━━━━ 4s 8ms/step
117/117 ━━━━━━━━━━━━━━━━━━━━ 1s 8ms/step
SOG - Train RMSE: 0.5332033951726334
SOG - Test RMSE: 0.5349242161526189
```

Fig 7.1 LSTM Loss PLot for SOG 246522000

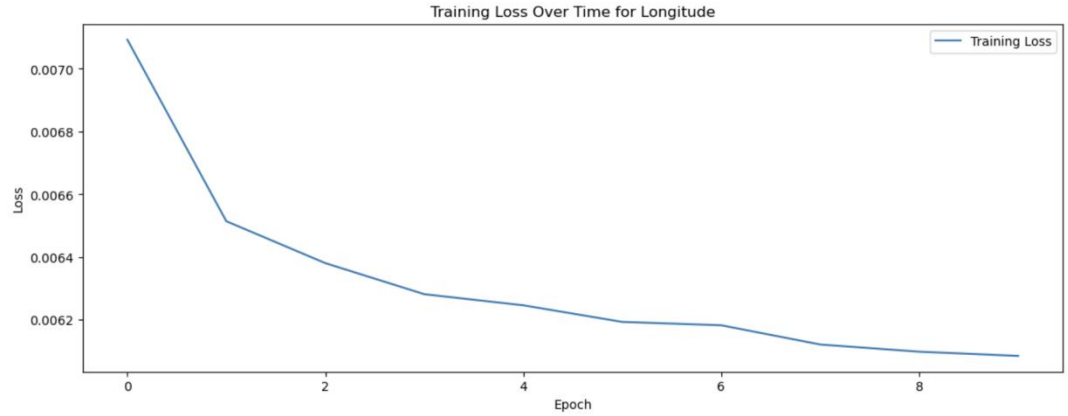

```
466/466 ━━━━━━━━━━━━━━━━━━━━ 4s 8ms/step
117/117 ━━━━━━━━━━━━━━━━━━━━ 1s 8ms/step
COG - Train RMSE: 0.30854763026771265
COG - Test RMSE: 0.30826247550315217
```

Fig 7.2. LSTM Loss PLot for COG 246522000




```
466/466 ━━━━━━━━━━━━━━━━━━━━ 4s 8ms/step
117/117 ━━━━━━━━━━━━━━━━━━━━ 1s 8ms/step
Latitude - Train RMSE: 0.3104457258956048
Latitude - Test RMSE: 0.31382695436728275
```

Fig 7.3 LSTM Loss PLot for Latitude 246522000




```
466/466 ━━━━━━━━━━━━━━━━━━━━ 4s 8ms/step
117/117 ━━━━━━━━━━━━━━━━━━━━ 1s 8ms/step
Longitude - Train RMSE: 0.3203463276416188
Longitude - Test RMSE: 0.31914801775082513
```

Fig 7.4 LSTM Loss PLot for Longitude 246522000

Project Implementation

Project Implementation Technology

The Project is designed and developed in Jupyter Notebook. We have used excel file to store the results of comaprison.

I. Hardware Requirement

i. Laptop or PC

- Windows 7 or higher
- I5 processor system or higher
- 4 GB RAM or higher
- 100 GB ROM or higher

II. Software Requirement

i. Laptop or PC

- Python
-

Coding

***FBPROPHET** SNAPSHOTS:*

In [82]:
```
def create_time_series(start_date, end_date, df, interval_minutes, columns, noise_levels=None):

    # Set the 'Timestamp' column as the index
    df.set_index('# Timestamp', inplace=True)

    # Resample the DataFrame to get one data point per minute
    resampled_df = df.resample(f'{interval_minutes}min').first().reset_index()

    resampled_df = df.groupby([pd.Grouper(freq=f'{interval_minutes}1min'), 'MMSI']).min().reset_index()

    # Generate datetime range with even intervals
    date_rng = pd.date_range(start=start_date, end=end_date, freq=f'{interval_minutes}T')

    # Create a DataFrame with datetime values
    new_df = pd.DataFrame(date_rng, columns=['ds'])

    # Merge resampled data with the new DataFrame on the 'ds' column
    new_df = pd.merge(new_df, resampled_df, how='left', left_on='ds', right_on='# Timestamp')

    # Drop the redundant '# Timestamp' column
    new_df.drop('# Timestamp', axis=1, inplace=True)

    # Generate time series values for each specified column
    for col, noise_level in zip(columns, noise_levels):
        if col.lower() == 'Latitude':
            new_df[col] = df[col]
        elif col.lower() == 'Longitude':
            new_df[col] = df[col]
        elif col.lower() =='SOG':
            new_df[col] =df[col]
        elif col.lower() =='COG':
            new_df[col] =df[col]
        else:
            new_df[col] = np.random.normal(0, noise_level, len(new_df)) + np.sin(np.linspace(0, 4 * np.pi, len(new_d

    return new_df
```

Fig. Even Intervals

In [83]:
```
def plot_time_series(df, columns, window=7):
    for col in columns:
        # Calculate moving average
        df['Moving Average'] = df[col].rolling(window=window).mean()

        plt.figure(figsize=(10, 6))
        plt.plot(df['ds'], df[col], label=col)
        plt.plot(df['ds'], df['Moving Average'], label=f'{col} - Moving Average', linestyle='--', color='red')
        plt.title(f'Time Series Plot for Dataset df - {col}')
        plt.xlabel('Time')
        plt.ylabel(col)
        plt.legend()
        plt.show()
```

Fig. Plot Time Series

In [84]:
```
## Menu-driven code
mmsi = input("Enter MMSI: ")  # Get the MMSI from the user
while True:
    print("\nMenu:")
    print("1. Generate Time Series - SOG ")
    print("2. Generate Time Series - COG")
    print("3. Generate Time Series - Latitude")
    print("4. Generate Time Series - Longitude")
    print("5. Exit")

    choice = input("Enter your choice (1-5): ")

    if choice in ['1', '2', '3', '4']:
        #mmsi = input("Enter MMSI: ")  # Get the MMSI from the user
        df_subset = df[df['MMSI'] == int(mmsi)]  # Subset the dataset based on the given MMSI
        columns = ['SOG'] if choice == '1' else ['COG'] if choice == '2' else ['Latitude'] if choice == '3' else ['L
        noise_levels = [0.5] if choice == '1' else [0.3]  # Adjust noise levels as needed

        time_series_data = create_time_series(start_date='2022-10-02',end_date='2022-10-03',df=df_subset, interval_m
        plot_time_series(time_series_data, columns)
    elif choice == '5':
        print("Exiting the program. Goodbye!")
        break
    else:
        print("Invalid choice. Please enter a number from 1 to 5.")
```

Fig. Implementation Even Intervals

In [86]:
```
from prophet.plot import plot_plotly, plot_components_plotly
from prophet import Prophet
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
```

In [87]:
```
def plot_prophet_forecast(train_data, test_data, forecast, columns, window=7):
    for col in columns:
        # Compute moving average
        test_data['Moving Average'] = test_data['y'].rolling(window=window).mean()

        plt.figure(figsize=(10, 6))
        plt.plot(train_data['ds'], train_data['y'], label=f'{col} - Training Data')
        plt.plot(test_data['ds'], test_data['y'], label=f'{col} - Actual Test Data')
        plt.plot(test_data['ds'], forecast[-len(test_data):]['yhat'], label=f'{col} - Forecast')
        plt.plot(test_data['ds'], test_data['Moving Average'], label=f'{col} - Moving Average', linestyle='--', colo
        plt.title(f'Time Series Forecast for DF - {col}')
        plt.xlabel('Time')
        plt.ylabel(col)
        plt.legend()
        plt.show()
```

Fig. Forecast

In [90]:
```
combined_data = pd.DataFrame()

while True:
    print("\nMenu:")
    print("1. Generate Time Series - SOG")
    print("2. Generate Time Series - COG")
    print("3. Generate Time Series - Latitude")
    print("4. Generate Time Series - Longitude")
    print("5. Exit")

    choice = input("Enter your choice (1-4): ")

    if choice == '5':
        print("Exiting the program. Goodbye!")
        break
    elif choice in ['1', '2', '3','4']:
        try:
            mmsi = input("Enter MMSI: ")  # Get the MMSI from the user
            df_subset = df[df['MMSI'] == int(mmsi)]  # Subset the dataset based on the given MMSI
        except ValueError:
            print("Invalid input. Please enter a valid integer for the interval.")
            continue

        if choice == '1':
            columns = ['SOG']
            noise_levels = [0.5]
        elif choice =='2':
            columns=['COG']
            noise_levels=[0.3]
        elif choice == '3':
            columns = ['Latitude']
            noise_levels = [0.3]
        elif choice == '4':
            columns = ['Longitude']
            noise_levels = [0.3]
    else:
        print("Invalid choice. Please enter a number from 1 to 4.")
        continue

    time_series_data = create_time_series(start_date='2023-10-02', end_date='2023-10-03',df=df_subset, interval_minu

    # Append to the DataFrame
    #df_all_time_series_data = pd.concat([df_all_time_series_data,time_series_data])
    combined_data = pd.concat([combined_data, time_series_data[columns]], axis=1)


    # Split data into training and testing sets (80% training, 20% testing)
    train_data, test_data = train_test_split(time_series_data, test_size=0.2, shuffle=False)

    # Rename columns to 'ds' and 'y' for Prophet
    train_data.rename(columns={columns[0]: 'y'}, inplace=True)
    test_data.rename(columns={columns[0]: 'y'}, inplace=True)
```

```
    # Train Prophet model
    prophet_model = Prophet()
    prophet_model.fit(train_data)

    #plot_prophet_loss(prophet_model)

    # Create a dataframe with future dates for prediction
    future = prophet_model.make_future_dataframe(periods=len(test_data), freq='1min')

    # Make predictions
    forecast = prophet_model.predict(future)

    # Plot the forecast and actual test data
    plot_prophet_forecast(train_data, test_data, forecast, columns)

    # Evaluate performance using mean squared error
    mse = mean_squared_error(test_data['y'], forecast[-len(test_data):]['yhat'])
    print(f'Mean Squared Error: {mse}')

    # Calculate deviation
    #test_data['deviation'] = ((test_data['y'] - forecast[-len(test_data):]['yhat'])/test_data['y']) * 100

    test_data['deviation'] = (test_data['y'] - forecast[-len(test_data):]['yhat'])

    # Calculate deviation percentage using stdev function
    sd = st.stdev(test_data['deviation'])
    print("Standard Deviation : ", sd)

    # Calculate moving average
    window_size = 12  # You can adjust the window size as needed
    test_data['moving_average'] = test_data['deviation'].rolling(window=window_size).mean()

    # Visualize deviation
    plt.figure(figsize=(10, 6))
    plt.plot(test_data['ds'], test_data['deviation'], label='Deviation')
    plt.plot(test_data['ds'], test_data['moving_average'], color='red', label='Moving Average')
    plt.axhline(y=0, color='r', linestyle='--')
    plt.title('Forecast Deviation')
    plt.xlabel('Time')
    plt.ylabel('Deviation')
    plt.legend()
    plt.show()



    #220068000
```

Fig.Training anf Testing

In [37]:
```
!pip install prophet
from prophet.plot import plot_plotly, plot_components_plotly
from prophet import Prophet
```

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib.axes as ax
import seaborn as sns
from sklearn.impute import SimpleImputer
import missingno as msno
from sklearn.preprocessing import StandardScaler
import statistics as st
!pip install prophet
from prophet.plot import plot_plotly, plot_components_plotly
from prophet import Prophet
```

Fig Libraries to be imported.

```
def create_time_series12(start_date,end_date,df1_2, interval_minutes, columns, noise_levels=None):
    # Generate datetime range with even intervals
    date_rng = pd.date_range(start=start_date, end=end_date, freq=f'{interval_minutes}min')

    # Create a DataFrame with datetime values
    new_df = pd.DataFrame(date_rng, columns=['ds'])

    # Generate time series values for each specified column
    for col, noise_level in zip(columns, noise_levels):
        #np.random.seed(42)
        if col.lower() == 'latitude':
            new_df[col] = df[col]
        elif col.lower() == 'longitude':
            new_df[col] = df[col]
        elif col.lower() == 'SOG':
            new_df[col] = df[col]
        elif col.lower() == 'COG':
            new_df[col] = df[col]
        else:
            new_df[col] = np.sin(np.linspace(0, 4 * np.pi, len(new_df))) + np.random.normal(0, noise_level, len(new_
    return new_df

def plot_prophet_forecast12(train_data, test_data, forecast, columns):
    for col in columns:
        plt.figure(figsize=(10, 6))
        plt.plot(train_data['ds'], train_data['y'], label=f'{col} - Training Data')
        plt.plot(test_data['ds'], test_data['y'], label=f'{col} - Actual Test Data')
        plt.plot(test_data['ds'], forecast[-len(test_data):]['yhat'], label=f'{col} - Forecast')
        plt.title(f'Time Series Forecast for Dataset1 - {col}')
        plt.xlabel('Time')
        plt.ylabel(col)
        plt.legend()
        plt.show()

## PRACTCIE


while True:
    print("\nMenu:")
    print("1. Generate Time Series - SOG")
    print("2. Generate Time Series - COG")
    print("3. Generate Time Series - Latitude")
    print("4. Generate Time Series - Longitude")
    print("5. Exit")

    choice = input("Enter your choice (1-4): ")

    if choice == '5':
        print("Exiting the program. Goodbye!")
        break
    elif choice in ['1', '2', '3','4']:
        if choice == '1':
            columns = ['SOG']
            noise_levels = [0.5]
        elif choice =='2':
            columns=['COG']
            noise_levels=[0.3]
        elif choice == '3':
            columns = ['Latitude']
            noise_levels = [0.3]
        elif choice == '4':
            columns = ['Longitude']
            noise_levels = [0.3]
    else:
        print("Invalid choice. Please enter a number from 1 to 4.")
        continue

     # First, you need to create the training and testing datasets separately
    # Let's assume your first dataset is stored in df and the second dataset is stored in df1
    time_series_data12 = create_time_series12(start_date='2023-10-02', end_date='2023-10-04', df1_2=df1_2, interval_


    train_data, test_data = train_test_split(time_series_data12, test_size=0.5, shuffle=False)
```

```python
# Rename columns to 'ds' and 'y' for Prophet

train_data.rename(columns={columns[0]: 'y'}, inplace=True)
test_data.rename(columns={columns[0]: 'y'}, inplace=True)

# Convert 'ds' column to datetime format
train_data['ds'] = pd.to_datetime(train_data['ds'])
test_data['ds'] = pd.to_datetime(test_data['ds'])

# Train Prophet model
prophet_model = Prophet()
prophet_model.fit(train_data)

# Create a dataframe with future dates for prediction
future = prophet_model.make_future_dataframe(periods=len(test_data), freq='min')

# Make predictions
forecast = prophet_model.predict(future)

# Plot the forecast and actual test data
plot_prophet_forecast12(train_data, test_data, forecast, columns)

# Evaluate performance using mean squared error
mse = mean_squared_error(test_data['y'], forecast[-len(test_data):]['yhat'])
print(f'Mean Squared Error: {mse}')

# Calculate deviation
test_data['deviation'] = test_data['y'] - forecast[-len(test_data):]['yhat']

# Calculate deviation percentage using stdev function
sd = np.std(test_data['deviation'])
print("Standard Deviation : ", sd)

# Calculate standard deviation percentage

# Visualize deviation
plt.figure(figsize=(10, 6))
plt.plot(test_data['ds'], test_data['deviation'], label='Deviation')
plt.axhline(y=0, color='r', linestyle='--')
plt.title('Forecast Deviation')
plt.xlabel('Time')
plt.ylabel('Deviation')
plt.legend()
plt.show()
```

Fig Prediction of 2 Datasets Combined.

LSTM

```python
def build_and_train_lstm(df, feature, time_step=100, epochs=10, batch_size=1):
    # Select the relevant feature
    df_values = df[[feature]].values

    # Normalize the data
    scaler = MinMaxScaler(feature_range=(0, 1))
    df_scaled = scaler.fit_transform(df_values)

    # Create the dataset
    def create_dataset(data, time_step=1):
        X, Y = [], []
        for i in range(len(data) - time_step - 1):
            a = data[i:(i + time_step), 0]
            X.append(a)
            Y.append(data[i + time_step, 0])
        return np.array(X), np.array(Y)

    X, Y = create_dataset(df_scaled, time_step)

    # Split the data into training and test sets
    train_size = int(len(X) * 0.8)
    test_size = len(X) - train_size
    X_train, X_test = X[:train_size], X[train_size:]
    Y_train, Y_test = Y[:train_size], Y[train_size:]

    # Reshape the input to be [samples, time steps, features]
    X_train = X_train.reshape(X_train.shape[0], X_train.shape[1], 1)
    X_test = X_test.reshape(X_test.shape[0], X_test.shape[1], 1)

    # Build the LSTM model
    model = Sequential()
    model.add(LSTM(50, return_sequences=True, input_shape=(time_step, 1)))
    model.add(LSTM(50, return_sequences=False))
    model.add(Dense(25))
    model.add(Dense(1))

    # Compile the model
    model.compile(optimizer='adam', loss='mean_squared_error')

    # Train the model and capture the history
    history = model.fit(X_train, Y_train, batch_size=batch_size, epochs=epochs)

    # Plot the training loss
    plt.figure(figsize=(14, 5))
    plt.plot(history.history['loss'], label='Training Loss')
    plt.xlabel('Epoch')
    plt.ylabel('Loss')
    plt.title(f'Training Loss Over Time for {feature}')
    plt.legend()
    plt.show()

    # Extract the final training loss
    final_loss = history.history['loss'][-1]

    # Predictions
    train_predict = model.predict(X_train)
    test_predict = model.predict(X_test)
```

```python
    # Inverse transform predictions
    train_predict = scaler.inverse_transform(train_predict)
    test_predict = scaler.inverse_transform(test_predict)

    # Inverse transform actual values
    Y_train = scaler.inverse_transform(Y_train.reshape(-1, 1))
    Y_test = scaler.inverse_transform(Y_test.reshape(-1, 1))

    # Calculate RMSE
    train_rmse = math.sqrt(mean_squared_error(Y_train, train_predict))
    test_rmse = math.sqrt(mean_squared_error(Y_test, test_predict))

    print(f'{feature} - Train RMSE: {train_rmse}')
    print(f'{feature} - Test RMSE: {test_rmse}')

    # Plot the results
    plt.figure(figsize=(14, 5))

    plt.plot(scaler.inverse_transform(df_scaled), label='Original Data')

    # Training data prediction
    train_predict_plot = np.empty_like(df_scaled)
    train_predict_plot[:, :] = np.nan
    train_predict_plot[time_step:len(train_predict) + time_step, :] = train_predict

    # Test data prediction
    test_predict_plot = np.empty_like(df_scaled)
    test_predict_plot[:, :] = np.nan
    test_predict_plot[len(train_predict) + (time_step) + 1:len(df_scaled), :] = test_predict

    # Plotting
    plt.plot(train_predict_plot, label='Train Prediction')
    plt.plot(test_predict_plot, label='Test Prediction')
    plt.legend()
    plt.show()

    # Save the model
    model.save(f'lstm_model_{feature}.h5')

    return train_rmse, test_rmse, final_loss

if __name__ == "__main__":
    # List of features to train on
    features = ['SOG', 'COG', 'Latitude', 'Longitude']

    # List of DataFrame names
    df_names = ['combined_data', 'combined_data1','combined_data2','combined_data3','combined_data4',
                'combined_data5','combined_data6','combined_data7','combined_data8','combined_data9',
                'combined_data10','combined_data11','combined_data12','combined_data13'] # Add more if needed

    # Create a list to store all results
    all_results = []

    for df_name in df_names:
        print(f"Processing {df_name}...")
        df = globals()[df_name]
        for feature in features:
```

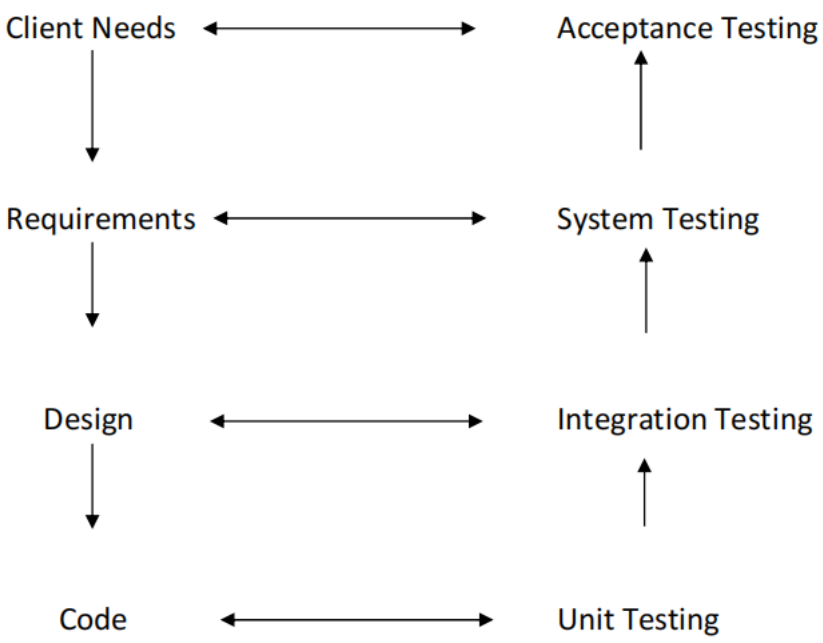
```python
        for feature in features:
            train_rmse, test_rmse, final_loss = build_and_train_lstm(df, feature, time_step=100, epochs=10, batch_si
            all_results.append({'DataFrame': df_name, 'Feature': feature, 'Train RMSE': train_rmse, 'Test RMSE': tes

    # Convert all results to a DataFrame
    all_results_df = pd.DataFrame(all_results)

    # Print the results
    print(all_results_df)

    # Save the results to an Excel file
    all_results_df.to_excel('LSTM_Tanker_229881000.xlsx', index=False)
    print("Results have been saved to LSTM_Tanker_229881000.xlsx")
```

Fig. LSTM Prediction

Testing

To ensure the success of our large-scale project, extensive testing is crucial. Each component must function correctly and deliver the desired output across various input scenarios. This comprehensive testing guarantees that the project will meet its objectives.
The testing performed here is System Testing, aimed at verifying that user requirements are met. The new system is entirely coded in Python, utilizing Machine Learning and Deep Learning techniques. It has undergone rigorous testing with the help of users, ensuring all applications have been meticulously verified from every possible angle. While some issues were initially discovered, they were promptly addressed and corrected before implementation. The flow of processes within the system aligns well with the actual data flow, confirming the system's robustness and reliability.

Levels of Testing



A series of testing is done for the proposed system before the system is ready for the user acceptance testing.

The steps involved in Testing are:

Unit Testing

Unit testing focuses verification efforts on the smallest unit of the software design, the module. This is also known as “Module Testing”. The modules are tested separately. This testing carried out during programming stage itself. In this testing each module is found to be working satisfactorily as regards to the expected output from the module.

Integration Testing

Data can be grossed across an interface; one module can have adverse efforts on another. Integration testing is systematic testing for construction the program structure while at the same time conducting tests to uncover errors associated with in the interface. The objective is to take unit tested modules and build a program structure. All the modules are combined and tested as a whole. Here correction is difficult because the isolation of cause is complicate by the vast expense of the entire program. Thus in the integration testing stop, all the errors uncovered are corrected for the text testing steps.

System testing

System testing is the stage of implementation that is aimed at ensuring that the system works accurately and efficiently for live operation commences. Testing is vital to the success of the system. System testing makes a logical assumption that if all the parts of the system are correct, then goal will be successfully achieved.

Validation Testing

At the conclusion of integration testing software is completely assembled as a package, interfacing errors have been uncovered and corrected and a final series of software tests begins, validation test begins. Validation test can be defined in many ways. But the simple definition is that validation succeeds when the software function in a manner that can reasonably expected by the customer. After validation test has been conducted one of two possible conditions exists.
One is the function or performance characteristics confirm to specifications and are accepted and the other is deviation from specification is uncovered and a deficiency list is created. Proposed system under consideration has been tested by using validation testing and found to be working satisfactorily.

-***Output Testing***

After performing validation testing, the next step is output testing of the proposed system since no system could be useful if it does not produce the required output in the specified format. Asking the users about the format required by them tests the outputs generated by the system under consideration. Here the output format is considered in two ways, one is on the screen and other is the printed format.
Theoutput format on the screen is found to be correct as the format was designed in the
system designed phase according to the user needs.
For the hard copy also the output comes as the specified requirements by the users.
Hence output testing does not result any corrections in the system.

User Acceptance Testing

User acceptance of a system is the key factor of the success of any system. The system under study is tested for the user acceptance by constantly keeping in touch with the prospective system users at the time of developing and making changes wherever required.

Test Cases :

1. FbProphet Model :

<u>Data Preparation:</u>

Create Even Intervals: Use the create_time_series12 function to generate a time series with specified intervals (1 min, 10 min, 60 min, etc.).
Generate Time Series: For each chosen column (e.g., SOG, COG, Latitude, Longitude), generate time series data with optional noise addition. Use the appropriate data from the provided datasets.

Merge Datasets:

Combine Data: Merge the generated time series from both datasets to create a comprehensive dataset covering the desired time frame.

Splitting Data:

Training and Testing: Split the combined dataset into training and testing datasets, typically with a 80-20 split while maintaining the time sequence (no shuffling).

Forecasting with Prophet:

Model Training: Train the Prophet model on the training dataset.
Future Dataframe: Create a future dataframe to predict values for the test period.
Prediction: Generate forecasted values using the Prophet model.

Evaluation:

Plot Forecast: Visualize the actual vs. forecasted values for the test dataset.
Calculate Deviation: Determine the deviation between actual and forecasted values.

Deviation Constraint: Ensure the deviation does not exceed **5%**.
Calculate the percentage deviation and check against this threshold.
Standard Deviation: Calculate the standard deviation of the deviation to ensure it aligns with the accuracy constraint.

FBProphet Testing Result :

MMSI (CARGO)	SOG	COG	LATITUDE	LONGITUDE
CARGO :215114000				
Dataset 1_2	1.7469	1.7112	8.1112	3.5087
Dataset 1_3	1.3502	1.2082	6.1733	2.9418
Dataset 1_4	1.8534	1.8811	6.1955	2.9579
Dataset 1_5	1.6001	1.6293	6.6741	3.0531
Dataset 1_6	1.9532	2.0006	1.9676	2.0432
Dataset 1_7	1.716	1.767	1.7976	1.8771
Dataset 1_8	1.8188	1.7748	1.8232	1.8
Dataset 1_9	1.8598	1.8671	1.899	1.8685
Dataset 1_10	1.8864	1.8864	1.9083	1.9078
Dataset 1_11	1.9129	1.9631	1.9876	1.9148
Dataset 1_12	1.9206	1.9553	1.929	1.9685
Dataset 1_13	1.0081	0.948	0.9387	0.9394
Dataset 1_14	1.9119	1.9774	1.9213	1.9284
CARGO :244246000				
Dataset 1_2	1.7311	1.8115	8.1112	3.5087
Dataset 1_3	1.3782	1.1586	6.1733	2.9418
Dataset 1_4	1.8304	1.8469	6.1955	2.9579
Dataset 1_5	1.5034	1.4604	6.6741	3.0531
Dataset 1_6	1.9424	1.8291	1.932	1.992
Dataset 1_7	1.926	1.8156	1.7799	1.7339
Dataset 1_8	1.7678	1.8342	1.8176	1.773
Dataset 1_9	0.8579	1.8203	1.902	1.8488
Dataset 1_10	1.8636	1.9401	1.8542	1.84
Dataset 1_11	1.9283	1.9372	1.982	1.8807
Dataset 1_12	1.9403	2.0102	1.8677	1.9267
Dataset 1_13	1.0241	0.9411	0.942	0.9397
Dataset 1_14	1.9568	1.9237	1.9473	1.933
CARGO :211718360				
Dataset 1_2	1.6452	1.7216	8.1112	3.5087
Dataset 1_3	1.2699	1.2699	1.2275	6.1733
Dataset 1_4	1.8718	1.8758	6.1955	2.9579
Dataset 1_5	1.5886	1.5024	6.6741	3.0531
Dataset 1_6	1.9102	2.097	1.9091	1.9985
Dataset 1_7	1.7791	1.7368	1.7806	1.7925
Dataset 1_8	1.8586	1.7812	1.9074	1.7415
Dataset 1_9	1.8843	1.9007	1.908	1.885
Dataset 1_10	1.8808	1.8661	1.8863	1.901
Dataset 1_11	1.9313	1.9275	1.9928	1.953

Dataset 1_12	1.9333	1.9119	1.9109	1.908
Dataset 1_13	1.0252	0.9365	0.9446	0.9434
Dataset 1_14	1.9483	1.8844	1.9071	1.9847
CARGO :231815000				
Dataset 1_2	1.7572	1.7189	8.1112	3.5087
Dataset 1_3	1.3464	1.2282	6.1733	2.9418
Dataset 1_4	1.9122	1.8587	6.1955	2.9579
Dataset 1_5	1.4122	1.497	6.6741	3.0531
Dataset 1_6	1.9556	1.9432	1.9857	1.9323
Dataset 1_7	1.9396	1.8657	1.7771	1.6901
Dataset 1_8	1.8722	1.8273	1.8705	1.8175
Dataset 1_9	1.8656	1.8594	1.9621	1.9326
Dataset 1_10	1.9239	1.8518	1.8709	1.8572
Dataset 1_11	1.8677	1.9552	1.9756	1.9214
Dataset 1_12	1.9548	1.9373	1.9204	1.9418
Dataset 1_13	1.0153	0.9435	0.9445	0.9391
Dataset 1_14	1.9384	1.9563	1.9179	1.9933

SHIPTYPE TANKER

MMSI (TANKER)	SOG	COG	LATITUDE	LONGITUDE
TANKER -209415000				
Dataset 1_2	1.8028	1.7542	8.1112	3.5087
Dataset 1_3	1.3923	1.2192	6.1733	2.9418
Dataset 1_4	1.921	1.7938	6.1955	2.9579
Dataset 1_5	1.5229	1.5641	6.6741	3.0531
Dataset 1_6	1.8912	1.9107	2.0215	1.9188
Dataset 1_7	1.7704	1.8442	1.7777	1.803
Dataset 1_8	1.7859	1.8368	1.9004	1.8262
Dataset 1_9	1.9438	1.9484	1.9428	1.9167
Dataset 1_10	1.9132	1.933	1.8992	1.862
Dataset 1_11	1.9265	1.8808	1.8959	1.9551
Dataset 1_12	1.9268	1.8677	1.9052	1.8743
Dataset 1_13	1.0092	0.9443	0.9548	0.9516
Dataset 1_14	1.8951	1.902	2.0413	1.9416
210051000				
Dataset 1_2	1.6934	1.6736	8.1112	3.5087
Dataset 1_3	1.3575	1.2214	6.1733	2.9418
Dataset1_4	1.7933	1.8037	6.1955	2.9579
Dataset1_5	1.5574	1.5207	6.6741	3.0531
Dataset1_6	2.20624	1.96405	1.9649	1.9696
Dataset1_7	1.9159	1.8497	1.789	1.776
Dataset1_8	1.8837	1.8309	1.7927	1.8177
Dataset1_9	1.8872	1.875	1.9442	1.9806
Dataset1_10	1.8707	1.8175	1.8252	1.8745
Dataset1_11	1.915	1.943	1.9327	1.9571
Dataset1_12	1.8697	1.9497	1.9169	1.9211
Dataset1_13	0.9905	0.9434	0.9556	0.9545
Dataset 1_14	1.9006	1.8704	1.962	1.9249

219008000				
Dataset 1_2	1.4942	1.723	8.1112	3.5087
Dataset 1_3	1.3593	1.14145	6.17336	2.9418
Dataset1_4	1.8234	1.7723	6.1955	2.9579
Dataset1_5	1.5144	1.5106	6.6741	3.0531
Dataset1_6	1.8844	2.017	2.0143	2.0088
Dataset1_7	1.8429	1.8086	1.7385	1.7635
Dataset1_8	1.9118	1.7588	1.7916	1.8033
Dataset1_9	1.9798	1.8742	1.9428	1.8866
Dataset1_10	1.9027	1.9358	1.9255	1.9302
Dataset1_11	1.8618	1.9545	1.9372	1.9447
Dataset1_12	1.9623	1.9535	1.9406	1.913
Dataset1_13	1.009	0.9422	0.9666	0.957
Dataset 1_14	1.9993	1.9713	1.95645	1.8998
219010252				
Dataset 1_2	1.6182	1.6789	8.1112	3.5087
Dataset 1_3	1.4202	1.217	6.1733	2.9418
Dataset1_4	1.7837	1.8746	6.1955	2.9579
Dataset1_5	1.5793	1.5757	6.6741	3.0531
Dataset1_6	1.9173	1.9063	2.0302	1.9939
Dataset1_7	1.8786	1.8827	1.836	1.8535
Dataset1_8	1.8988	1.789	1.8727	1.8414
Dataset1_9	1.9411	1.92136	1.8995	1.9003
Dataset1_10	1.7805	1.8468	1.8483	1.9057
Dataset1_11	1.9633	1.8819	1.8956	1.85913
Dataset1_12	1.8941	1.9396	1.8938	1.9233
Dataset1_13	0.9992	0.958	0.9447	0.9435
Dataset 1_14	1.9597	1.9765	1.9747	1.9336

SHIPTYPE : FISHING

MMSI (FISHING)	SOG	COG	LATITUDE	LONGITUDE
Fishing : **97000048**				
Dataset 1_2	5.963944	103.37369	8.097174	3.447692
Dataset 1_3	1.417013	1.159317	6.173369	2.941803
Dataset 1_4	1.725372	1.982258	6.195517	2.957979
Dataset 1_5	1.626367	1.496739	6.674134	6.674134
Dataset 1_6	2.001856	2.013315	2.02551	2.007505
Dataset 1_7	1.746914	1.863356	1.793396	1.860735
Dataset 1_8	0.951706	0.941006	0.925235	0.927667
Dataset 1_9	0.951706	0.941006	0.925235	0.927667
Dataset 1_10	1.89826	1.9437	1.9054	1.8367
Dataset 1_11	1.929374	1.940514	1.924682	1.947232
Dataset 1_12	1.961257	1.942995	1.902906	1.886124
Dataset 1_13	1.021684	0.939371	0.936341	0.966622

Dataset 1_14	1.873694	1.963955	1.894968	2.003505
211332750				
Dataset 1_2	5.963944	103.37369	8.097174	3.447692
Dataset 1_3	1.347493	1.181398	6.173369	2.941803
Dataset 1_4	1.835887	1.888438	6.195517	2.957979
Dataset 1_5	1.632475	1.610373	6.674134	3.053189
Dataset 1_6	1.976506	1.937851	1.93368	1.909631
Dataset 1_7	1.779251	1.812507	1.82385	1.792364
Dataset 1_8	1.781684	1.834749	1.81018	1.817925
Dataset 1_9	0.98767	0.947614	0.924465	0.931834
Dataset 1_10	1.854469	1.876323	1.878832	1.880563
Dataset 1_11	2.002099	1.91679	1.932664	1.943997
Dataset 1_12	1.984263	1.953051	1.930366	1.9461
Dataset 1_13				
Dataset 1_14				
211502210				
Dataset 1_2	5.963944	103.37369	8.097174	3.447692
Dataset 1_3	1.345658	1.183794	6.173369	2.941803
Dataset 1_4	1.717883	1.824597	6.195517	2.957979
Dataset 1_5	1.503838	1.545795	6.674134	3.053189
Dataset 1_6	1.846006	2.033963	1.955415	1.990835
Dataset 1_7	1.811863	1.816664	1.74537	1.796808
Dataset 1_8	1.756952	1.852653	1.767294	1.899773
Dataset 1_9	0.998411	0.941665	0.933067	0.941099
Dataset 1_10	1.900528	1.858981	1.924752	1.87419
Dataset 1_11	1.970223	1.934756	1.908664	1.946554
Dataset 1_12	1.978304	1.883317	1.966411	1.940611
Dataset 1_13	1.001741	0.947162	0.956496	0.924098
Dataset 1_14	1.975624	1.944397	1.917596	1.972085
219000907				
Dataset 1_2	1.6614	1.7662	8.1112	3.5087
Dataset 1_3	1.4068	1.2229	6.1733	2.9418
Dataset 1_4	1.7898	1.7963	6.1955	2.9579
Dataset 1_5	1.5518	1.5561	6.6741	3.0531
Dataset 1_6	1.9044	2.0285	1.99315	1.9987
Dataset 1_7	1.8116	1.7502	1.7751	1.7537
Dataset 1_8	1.8076	1.8346	1.7963	1.7215
Dataset 1_9	1.98501	1.8841	1.9025	1.8839
Dataset 1_10	1.8523	1.9008	1.7396	1.8702
Dataset 1_11	1.9138	1.9217	1.9282	1.9555
Dataset 1_12	1.8367	1.9661	1.887	1.9498
Dataset 1_13	1.0134	0.9217	0.9446	0.9407
Dataset 1_14	1.9229	1.9098	1.9349	1.9423

SHIPTYPE Passenger

MMSI (Passenger)	SOG	COG	LATITUDE	LONGITUDE

209764000				
Dataset 1_2	1.6625	1.7777	8.1112	3.5087
Dataset 1_3	1.3452	1.1895	6.1733	2.9418
Dataset 1_4	1.8632	1.7555	6.1955	2.9579
Dataset 1_5	1.692	1.5028	6.6741	3.0531
Dataset 1_6	1.9339	1.9539	2.048	1.9843
Dataset 1_7	1.7165	1.8144	1.8598	1.7855
Dataset 1_8	1.705	1.8147	1.7931	1.8502
Dataset 1_9	1.9299	1.9009	1.9248	1.9177
Dataset 1_10	1.8028	1.9383	1.8984	1.9575
Dataset 1_11	1.9627	1.9028	1.9223	1.9451
Dataset 1_12	1.8877	1.9443	1.8939	1.90469
Dataset 1_13	1.0164	0.9345	0.9384	0.9442
Dataset 1_14	1.9447	1.9512	1.9796	1.9424
211226940				
Dataset 1_2	1.6785	1.7412	8.1112	3.5087
Dataset 1_3	1.4008	1.1639	6.1733	2.9418
Dataset 1_4	1.8654	1.8047	6.1955	2.9579
Dataset 1_5	1.4758	1.4889	6.6741	3.0531
Dataset 1_6	1.9899	1.9989	1.9511	1.9569
Dataset 1_7	1.9133	1.7515	1.8404	1.8441
Dataset 1_8	1.7973	1.833	1.78417	1.8584
Dataset 1_9	1.889	1.8774	1.9741	1.8949
Dataset 1_10	1.84476	1.8691	1.84581	1.8944
Dataset 1_11	1.9554	1.8995	1.8872	1.9629
Dataset 1_12	1.9031	1.9961	1.9823	1.8496
Dataset 1_13	1.01485	0.9511	0.9469	0.9456
Dataset 1_14	1.9274	1.9126	1.9461	1.945
211543850				
Dataset 1_2	1.7145	1.7642	8.1112	3.5087
Dataset 1_3	1.3486	1.1925	6.1733	2.9418
Dataset 1_4	1.8973	1.8715	6.1955	2.9579
Dataset 1_5	1.57205	1.4878	6.6741	3.0531
Dataset 1_6	1.9227	1.9922	2.007	2.0226
Dataset 1_7	1.727	1.7592	1.8699	1.7747
Dataset 1_8	1.7338	1.8857	1.82746	1.8716
Dataset 1_9	1.8803	1.9211	1.88	1.975
Dataset 1_10	1.8792	1.90726	1.9259	1.8608
Dataset 1_11	1.9641	1.8919	1.9437	1.9434
Dataset 1_12	1.9079	1.8788	1.9354	1.8989
Dataset 1_13	1.0314	0.9342	0.9552	0.9607
Dataset 1_14	1.9426	1.943	1.9263	1.8854
219000675				
Dataset 1_2	1.8663	1.8115	8.1112	3.5087
Dataset 1_3	1.3636	1.2369	6.1733	2.9418
Dataset 1_4	1.8057	1.8318	6.1955	2.9579
Dataset 1_5	1.6882	1.5194	6.6741	3.0531
Dataset 1_6	1.8183	1.9737	1.8658	2.0513
Dataset 1_7	1.5772	1.7894	1.7794	1.7671

Dataset 1_8	1.8649	1.9137	1.82	1.828
Dataset 1_9	1.8539	1.8677	1.9648	1.9324
Dataset 1_10	1.8536	1.8474	1.884	1.8924
Dataset 1_11	1.8752	1.9088	1.9611	1.936
Dataset 1_12	1.9026	1.8437	1.9199	1.9406
Dataset 1_13	1.0189	0.9422	0.9644	0.9429
Dataset 1_14	1.9137	1.9608	1.8793	1.9345
219000734				
Dataset 1_2	1.623	1.6911	8.1112	3.5087
Dataset 1_3	1.3289	1.208	6.1733	2.9418
Dataset 1_4	1.8978	1.8644	6.1955	2.9579
Dataset 1_5	1.5998	1.6305	6.6741	3.0531
Dataset 1_6	2.0258	1.9764	1.9946	2.0224
Dataset 1_7	1.788	1.8514	1.8675	1.7946
Dataset 1_8	1.7733	1.7901	1.8587	1.85739
Dataset 1_9	1.798	1.9306	1.91	1.9005
Dataset 1_10	1.9686	1.8986	1.8856	1.9206
Dataset 1_11	1.8927	1.9428	1.9467	1.9626
Dataset 1_12	1.9225	1.8956	1.8766	1.9186
Dataset 1_13	0.993	0.9634	0.9634	0.9381
Dataset 1_14				

LSTM

Predictions: The trained model is used to make predictions on both the training and testing sets.

Inverse Transformation: Since the data was normalized, the predictions are inverse transformed to the original scale using the same scaler. This

step ensures that the error metrics are in the same scale as the original data.

RMSE Calculation: The Root Mean Squared Error (RMSE) is calculated for both the training and testing sets. RMSE is a commonly used metric to measure the difference between predicted and actual values. A lower RMSE indicates better performance.

MMSI : Cargo_246522000

DataFrame	Feature	Train RMSE	Test RMSE	Final Loss
combined_data	SOG	0.531782051	0.530402854	0.015256871
combined_data	COG	0.384372745	0.457533678	0.008973585
combined_data	Latitude	0.328261317	0.295136537	0.008911051
combined_data	Longitude	0.317819962	0.321434738	0.006997031
combined_data1	SOG	0.560794788	0.523941968	0.013992673
combined_data1	COG	0.31419478	0.311503273	0.006978431
combined_data1	Latitude	0.329152804	0.314807593	0.008742264
combined_data1	Longitude	0.319706975	0.327573376	0.007954257
combined_data2	SOG	0.557007746	0.478069108	0.015539441
combined_data2	COG	0.32017571	0.325234828	0.008206762
combined_data2	Latitude	0.313689844	0.302565728	0.007684048
combined_data2	Longitude	0.377351653	0.349212131	0.008930653
combined_data3	SOG	0.601070287	0.560165157	0.021563686
combined_data3	COG	0.41497894	0.451024322	0.007823397
combined_data3	Latitude	0.31828924	0.323177701	0.008092929
combined_data3	Longitude	0.32341057	0.310578044	0.009416166
combined_data4	SOG	0.53448937	0.564970158	0.011588188
combined_data4	COG	0.301991074	0.298949329	0.008920291
combined_data4	Latitude	0.319354848	0.293176745	0.009182189
combined_data4	Longitude	0.321554818	0.347975198	0.007957729
combined_data5	SOG	0.556521817	0.564780957	0.010118274
combined_data5	COG	0.326003037	0.334504058	0.007214226
combined_data5	Latitude	0.314559337	0.324621115	0.008697933
combined_data5	Longitude	0.337708624	0.333766605	0.011377106
combined_data6	SOG	0.582487158	0.583179068	0.012292297
combined_data6	COG	0.331703538	0.330205659	0.009233358
combined_data6	Latitude	0.315631476	0.312422134	0.008219576
combined_data6	Longitude	0.4045471	0.331743119	0.009768317
combined_data7	SOG	0.518795442	0.576529057	0.012404975
combined_data7	COG	0.322804896	0.316818115	0.008634413
combined_data7	Latitude	0.397204732	0.293063609	0.010111301
combined_data7	Longitude	0.306596108	0.317837978	0.009491628
combined_data8	SOG	0.532260259	0.484417906	0.012968828
combined_data8	COG	0.395302096	0.388398677	0.008542481
combined_data8	Latitude	0.314773081	0.318733693	0.00848751
combined_data8	Longitude	0.315630018	0.329521283	0.008552385

combined_data9	SOG	0.548116302	0.606659819	0.011323588
combined_data9	COG	0.329250627	0.305261464	0.008017823
combined_data9	Latitude	0.313947335	0.321694747	0.00884344
combined_data9	Longitude	0.33003181	0.336516215	0.009493891
combined_data10	SOG	0.536634926	0.561349443	0.011916388
combined_data10	COG	0.323416008	0.321783773	0.008595807
combined_data10	Latitude	0.312248237	0.317120838	0.00751234
combined_data10	Longitude	0.336515607	0.353719541	0.009277838
combined_data11	SOG	0.543122074	0.553613954	0.013108309
combined_data11	COG	0.323610977	0.330442452	0.008828669
combined_data11	Latitude	0.320833314	0.333867032	0.008677111
combined_data11	Longitude	0.324794666	0.358969551	0.009226592
combined_data12	SOG	0.569322579	0.65807178	0.012094017
combined_data12	COG	0.360215483	0.346918777	0.007218195
combined_data12	Latitude	0.32369524	0.339264109	0.008127584
combined_data12	Longitude	0.306633711	0.310861029	0.008699253
combined_data13	SOG	0.512932026	0.514739348	0.015924882
combined_data13	COG	0.309758997	0.345873859	0.007762609
combined_data13	Latitude	0.323231689	0.301279574	0.00958134
combined_data13	Longitude	0.324078197	0.313778263	0.00832831

MMSI : Tanker_219031429

DataFrame	Feature	Train RMSE	Test RMSE	Final Loss
combined_data	SOG	0.513729025	0.538409721	0.011512127
combined_data	COG	0.332425394	0.324618083	0.009307227
combined_data	Latitude	0.32327215	0.320793477	0.008695909
combined_data	Longitude	0.32891908	0.372161519	0.008916884
combined_data1	SOG	0.556065004	0.565144518	0.012922728
combined_data1	COG	0.309084381	0.323943967	0.007947213
combined_data1	Latitude	0.353653574	0.334472339	0.007339747
combined_data1	Longitude	0.33285509	0.365007169	0.008711657
combined_data2	SOG	0.568578456	0.588874699	0.012071289
combined_data2	COG	0.307866047	0.301453134	0.008094864
combined_data2	Latitude	0.326529383	0.304648351	0.008557048
combined_data2	Longitude	0.326901408	0.306596559	0.009559911
combined_data3	SOG	0.548767405	0.558169173	0.013016013
combined_data3	COG	0.327159785	0.324881899	0.010542939
combined_data3	Latitude	0.327228702	0.307813453	0.009896421
combined_data3	Longitude	0.357261976	0.344925034	0.007570061
combined_data4	SOG	0.521772803	0.500355041	0.013513216
combined_data4	COG	0.335251946	0.35043176	0.00872869
combined_data4	Latitude	0.314373409	0.318866237	0.00908343
combined_data4	Longitude	0.324891494	0.328439291	0.009322553
combined_data5	SOG	0.505927581	0.504734397	0.012554278
combined_data5	COG	0.335310276	0.304174073	0.008117051
combined_data5	Latitude	0.350156807	0.385285042	0.009854811

combined_data5	Longitude	0.309012958	0.30925127	0.009121252
combined_data6	SOG	0.579278614	0.56266668	0.012332225
combined_data6	COG	0.326596177	0.310417781	0.008504282
combined_data6	Latitude	0.319019201	0.321745255	0.009889354
combined_data6	Longitude	0.320631708	0.317554736	0.007978098
combined_data7	SOG	0.5416703	0.531428699	0.011993107
combined_data7	COG	0.314466245	0.301017275	0.008145426
combined_data7	Latitude	0.314919338	0.306994924	0.008980764
combined_data7	Longitude	0.310611587	0.331263591	0.009420959
combined_data8	SOG	0.527205055	0.525801017	0.01161092
combined_data8	COG	0.326750263	0.312590602	0.009245554
combined_data8	Latitude	0.3259032	0.312829109	0.007745928
combined_data8	Longitude	0.316054653	0.339742197	0.007802932
combined_data9	SOG	0.521728921	0.50433867	0.010802853
combined_data9	COG	0.329739669	0.33858104	0.009888996
combined_data9	Latitude	0.312458452	0.305066766	0.007815792
combined_data9	Longitude	0.346599989	0.358950491	0.008254871
combined_data10	SOG	0.515213028	0.516208078	0.011328698
combined_data10	COG	0.402849176	0.330892558	0.00802815
combined_data10	Latitude	0.32361797	0.331544609	0.008764472
combined_data10	Longitude	0.339062148	0.360827976	0.007559747
combined_data11	SOG	0.52072927	0.53809185	0.013448685
combined_data11	COG	0.346597755	0.327896858	0.008147939
combined_data11	Latitude	0.335741079	0.322146385	0.008679423
combined_data11	Longitude	0.346597562	0.326474124	0.008804177
combined_data12	SOG	0.517284535	0.54726182	0.011909939
combined_data12	COG	0.396010835	0.335136533	0.00904112
combined_data12	Latitude	0.32505508	0.296593927	0.007781379
combined_data12	Longitude	0.306845702	0.312845279	0.008783993
combined_data13	SOG	0.521004052	0.525085877	0.010286424
combined_data13	COG	0.313453734	0.331266994	0.00753905
combined_data13	Latitude	0.325451401	0.30004305	0.008419553
combined_data13	Longitude	0.308064575	0.320162939	0.008775009

MMSI : Passenger_219000734

DataFrame	Feature	Train RMSE	Test RMSE	Final Loss
combined_data	SOG	0.521802741	0.528862398	0.014131555
combined_data	COG	0.317729725	0.325951021	0.008015547
combined_data	Latitude	0.308353269	0.302963595	0.008970371
combined_data	Longitude	0.306064526	0.301365656	0.009453584

combined_data1	SOG	0.530379998	0.549247847	0.012788628
combined_data1	COG	0.319482676	0.327163888	0.00824883
combined_data1	Latitude	0.314441131	0.312127386	0.009649548
combined_data1	Longitude	0.31906542	0.329546692	0.00842722
combined_data2	SOG	0.509324964	0.521449906	0.013258701
combined_data2	COG	0.312855889	0.327685356	0.006871121
combined_data2	Latitude	0.324226221	0.33042246	0.008770317
combined_data2	Longitude	0.320846526	0.348628334	0.008687145
combined_data3	SOG	0.515305254	0.527895958	0.01416712
combined_data3	COG	0.30779378	0.312944537	0.007146379
combined_data3	Latitude	0.310833401	0.341889822	0.008031967
combined_data3	Longitude	0.33004437	0.32341811	0.008897994
combined_data4	SOG	0.562957492	0.522766716	0.009465592
combined_data4	COG	0.330403351	0.310687986	0.008951935
combined_data4	Latitude	0.312290282	0.320698296	0.008889067
combined_data4	Longitude	0.338894213	0.339006967	0.008839701
combined_data5	SOG	0.514144782	0.480010918	0.014902265
combined_data5	COG	0.399185197	0.323617181	0.009248813
combined_data5	Latitude	0.325965688	0.327577715	0.009002903
combined_data5	Longitude	0.31829987	0.336232499	0.008361487
combined_data6	SOG	0.528720996	0.525291879	0.013632633
combined_data6	COG	0.302470427	0.306994173	0.009407401
combined_data6	Latitude	0.326009647	0.301081154	0.008774209
combined_data6	Longitude	0.314019763	0.280335323	0.009120939
combined_data7	SOG	0.567977533	0.539405409	0.01344281
combined_data7	COG	0.335240701	0.332004751	0.009384307
combined_data7	Latitude	0.327798815	0.349912719	0.008808137
combined_data7	Longitude	0.351888036	0.326528072	0.008515266
combined_data8	SOG	0.542800201	0.578314139	0.012072396
combined_data8	COG	0.366069476	0.358907387	0.008709083
combined_data8	Latitude	0.324914012	0.343369153	0.009128083
combined_data8	Longitude	0.305077331	0.339457582	0.008275161
combined_data9	SOG	0.530087826	0.557054797	0.012037752
combined_data9	COG	0.321382529	0.309296322	0.00827384
combined_data9	Latitude	0.357925027	0.394412901	0.007433088
combined_data9	Longitude	0.446729251	0.443438188	0.006847822
combined_data10	SOG	0.510069065	0.524870128	0.012571019
combined_data10	COG	0.34136503	0.340513395	0.008326693
combined_data10	Latitude	0.312261371	0.312294319	0.008761147
combined_data10	Longitude	0.342591457	0.350831909	0.008629802
combined_data11	SOG	0.504547692	0.530439815	0.012410705
combined_data11	COG	0.3906723	0.328456043	0.008018997
combined_data11	Latitude	0.325439569	0.327161587	0.008530379
combined_data11	Longitude	0.325534125	0.331239135	0.008171837
combined_data12	SOG	0.52871656	0.522125981	0.012616402
combined_data12	COG	0.309275298	0.306901141	0.008427651
combined_data12	Latitude	0.316915062	0.323303522	0.008535532
combined_data12	Longitude	0.318843831	0.299455481	0.008697446
combined_data13	SOG	0.524490669	0.523947175	0.010546398

combined_data13	COG	0.316101175	0.296870102	0.008024559
combined_data13	Latitude	0.319377201	0.327431191	0.00740863
combined_data13	Longitude	0.334186567	0.332541965	0.008850036

MMSI : Fishing_219001039

DataFrame	Feature	Train RMSE	Test RMSE	Final Loss
combined_data	SOG	0.552255213	0.554475853	0.013228338
combined_data	COG	0.318425841	0.309361044	0.009446263
combined_data	Latitude	0.347196326	0.35870118	0.008666649
combined_data	Longitude	0.311063972	0.304799197	0.00786317
combined_data1	SOG	0.542693367	0.537726547	0.014795697
combined_data1	COG	0.317043737	0.318708196	0.009716554
combined_data1	Latitude	0.350011388	0.327716205	0.007764327
combined_data1	Longitude	0.316862009	0.315795104	0.00844486
combined_data2	SOG	0.536957727	0.485881815	0.013162413
combined_data2	COG	0.318967779	0.329102134	0.008708382
combined_data2	Latitude	0.314317358	0.345089171	0.007802815
combined_data2	Longitude	0.316086157	0.318991982	0.009005195
combined_data3	SOG	0.541765362	0.559801969	0.012744661
combined_data3	COG	0.32091412	0.345081444	0.008577492
combined_data3	Latitude	0.320250538	0.302329861	0.008947338
combined_data3	Longitude	0.330836534	0.322185809	0.008726036
combined_data4	SOG	0.524612012	0.522284325	0.009460267
combined_data4	COG	0.329841645	0.314996959	0.008341142
combined_data4	Latitude	0.315050892	0.321241721	0.007548017
combined_data4	Longitude	0.309796954	0.312175626	0.007415013
combined_data5	SOG	0.516364977	0.531532935	0.013026093
combined_data5	COG	0.320638802	0.296968045	0.00821449
combined_data5	Latitude	0.321298006	0.327396327	0.007592959
combined_data5	Longitude	0.325569773	0.30209241	0.007953157
combined_data6	SOG	0.535611546	0.510861086	0.01269612
combined_data6	COG	0.319951406	0.352548993	0.007162343
combined_data6	Latitude	0.30109059	0.312533646	0.00684659
combined_data6	Longitude	0.323490816	0.342540429	0.007692228
combined_data7	SOG	0.572602546	0.555286812	0.015155862
combined_data7	COG	0.331177201	0.343774687	0.008729514
combined_data7	Latitude	0.321388063	0.312217033	0.010693812
combined_data7	Longitude	0.319356161	0.305902908	0.008894627
combined_data8	SOG	0.522540877	0.540549675	0.01246758
combined_data8	COG	0.336117293	0.313072901	0.009960426
combined_data8	Latitude	0.338373585	0.303408004	0.00808376
combined_data8	Longitude	0.311263637	0.330714272	0.008105518
combined_data9	SOG	0.565124295	0.523774156	0.014265562
combined_data9	COG	0.346975369	0.33401921	0.009782163
combined_data9	Latitude	0.333272458	0.348496508	0.007745609

combined_data9	Longitude	0.307244351	0.306979493	0.00878828
combined_data10	SOG	0.514483429	0.493157684	0.012370663
combined_data10	COG	0.323810986	0.32700217	0.010105128
combined_data10	Latitude	0.343559458	0.318689096	0.009326718
combined_data10	Longitude	0.30958492	0.299253985	0.009166789
combined_data11	SOG	0.536998101	0.553130066	0.009795939
combined_data11	COG	0.348057486	0.298004842	0.009362615
combined_data11	Latitude	0.32422686	0.325453016	0.009188885
combined_data11	Longitude	0.348747369	0.305140292	0.008071301
combined_data12	SOG	0.54081409	0.548477026	0.012253512
combined_data12	COG	0.323195041	0.333110824	0.008862836
combined_data12	Latitude	0.317051475	0.296607968	0.009354276
combined_data12	Longitude	0.354676283	0.360324097	0.007664037
combined_data13	SOG	0.522565658	0.512670104	0.013265931
combined_data13	COG	0.306972013	0.318330959	0.009520981
combined_data13	Latitude	0.347057821	0.325402806	0.007366238
combined_data13	Longitude	0.36196015	0.375766305	0.008450794

MMSI : Fishing 211502210

DataFrame	Feature	Train RMSE	Test RMSE	Final Loss
combined_data	SOG	0.521023223	0.550168763	0.011567821
combined_data	COG	0.313909454	0.331091807	0.007358402
combined_data	Latitude	0.319501404	0.31181471	0.009306066
combined_data	Longitude	0.359067522	0.32416538	0.008524626
combined_data1	SOG	0.542824323	0.508912729	0.011657542
combined_data1	COG	0.356696671	0.328717808	0.008775422
combined_data1	Latitude	0.373492317	0.322844376	0.008731845
combined_data1	Longitude	0.324752109	0.301087633	0.007814573
combined_data2	SOG	0.54780238	0.558704856	0.014309078
combined_data2	COG	0.323695157	0.322013981	0.008180675
combined_data2	Latitude	0.308168579	0.323091599	0.009171241
combined_data2	Longitude	0.337879821	0.315369786	0.010747142
combined_data3	SOG	0.533026747	0.566525494	0.014494464
combined_data3	COG	0.341754171	0.334739981	0.008015702
combined_data3	Latitude	0.309433471	0.313421643	0.007780156
combined_data3	Longitude	0.330996471	0.297112371	0.01020043
combined_data4	SOG	0.541738993	0.617419685	0.011865682
combined_data4	COG	0.319030391	0.371290325	0.007851479
combined_data4	Latitude	0.314553768	0.316114291	0.009012832
combined_data4	Longitude	0.387321988	0.305571186	0.007472782
combined_data5	SOG	0.539249361	0.544064186	0.014022445
combined_data5	COG	0.313302217	0.313226163	0.008305928
combined_data5	Latitude	0.304536006	0.291707336	0.007558335
combined_data5	Longitude	0.426641614	0.333121587	0.009870371
combined_data6	SOG	0.600479694	0.56048231	0.012686756
combined_data6	COG	0.32826443	0.330402773	0.009151436

combined_data6	Latitude	0.329076927	0.341879147	0.009012617
combined_data6	Longitude	0.330149082	0.329575919	0.008716821
combined_data7	SOG	0.521474122	0.558240175	0.014680915
combined_data7	COG	0.347188354	0.345840988	0.009182895
combined_data7	Latitude	0.34405892	0.289603689	0.01011716
combined_data7	Longitude	0.317820016	0.30607503	0.008491992
combined_data8	SOG	0.561625015	0.594653266	0.010587945
combined_data8	COG	0.346016215	0.32129062	0.008453528
combined_data8	Latitude	0.318113737	0.284971145	0.009008216
combined_data8	Longitude	0.313843065	0.344949503	0.008222591
combined_data9	SOG	0.576613459	0.583223841	0.01283145
combined_data9	COG	0.303939571	0.298322576	0.008660068
combined_data9	Latitude	0.321801709	0.321693234	0.009374867
combined_data9	Longitude	0.326669144	0.344813267	0.009752628
combined_data10	SOG	0.520462888	0.525977771	0.013416714
combined_data10	COG	0.320239802	0.314934418	0.007951609
combined_data10	Latitude	0.312636328	0.339861396	0.009249602
combined_data10	Longitude	0.346391722	0.334047724	0.008408472
combined_data11	SOG	0.547231421	0.519364873	0.017678138
combined_data11	COG	0.34749429	0.32786736	0.010177217
combined_data11	Latitude	0.323051234	0.302740566	0.008808641
combined_data11	Longitude	0.312080297	0.31050161	0.007904876
combined_data12	SOG	0.573485055	0.556933084	0.016153531
combined_data12	COG	0.325780127	0.370248859	0.008364994
combined_data12	Latitude	0.32991465	0.33933899	0.007451331
combined_data12	Longitude	0.327837168	0.305060054	0.009409451
combined_data13	SOG	0.509389201	0.519607495	0.014449974
combined_data13	COG	0.309799501	0.328539319	0.007792538
combined_data13	Latitude	0.309035877	0.329183354	0.008760123
combined_data13	Longitude	0.327410187	0.336686873	0.008228495

MMSI : Fishing 219002661

DataFrame	Feature	Train RMSE	Test RMSE	Final Loss
combined_data	SOG	0.532022174	0.503263105	0.015001829
combined_data	COG	0.315297668	0.289908558	0.008948912
combined_data	Latitude	0.343701604	0.352619733	0.008338631
combined_data	Longitude	0.358984261	0.309245861	0.0091052
combined_data1	SOG	0.51219826	0.52390819	0.013791798
combined_data1	COG	0.321454882	0.308628265	0.009344631
combined_data1	Latitude	0.334344619	0.291913007	0.008846055
combined_data1	Longitude	0.318653706	0.353996546	0.008916453
combined_data2	SOG	0.521354852	0.489505883	0.014293894
combined_data2	COG	0.345998533	0.294345153	0.01015452
combined_data2	Latitude	0.365978705	0.33463161	0.008473287
combined_data2	Longitude	0.331390881	0.338574756	0.007873267
combined_data3	SOG	0.51641077	0.524374184	0.014543088

combined_data3	COG	0.303112708	0.335939395	0.008864844
combined_data3	Latitude	0.30164696	0.320125992	0.007906511
combined_data3	Longitude	0.345888913	0.330148659	0.008538917
combined_data4	SOG	0.540853943	0.480326107	0.014600732
combined_data4	COG	0.302729966	0.316246634	0.006716934
combined_data4	Latitude	0.309302218	0.295335719	0.00882076
combined_data4	Longitude	0.356512665	0.318481485	0.009245794
combined_data5	SOG	0.53581012	0.49423478	0.011803024
combined_data5	COG	0.337967642	0.335113398	0.008510303
combined_data5	Latitude	0.31630143	0.32083853	0.007011015
combined_data5	Longitude	0.325027063	0.339042138	0.009565908
combined_data6	SOG	0.554507546	0.574434913	0.013060366
combined_data6	COG	0.360063831	0.348200247	0.007980043
combined_data6	Latitude	0.324963916	0.309668938	0.008840065
combined_data6	Longitude	0.31960798	0.29679201	0.009583106
combined_data7	SOG	0.527932565	0.53019394	0.011713534
combined_data7	COG	0.34108945	0.310993668	0.008568792
combined_data7	Latitude	0.324004629	0.298577618	0.008590844
combined_data7	Longitude	0.341431684	0.314116357	0.00819428
combined_data8	SOG	0.535792042	0.585377283	0.013948288
combined_data8	COG	0.316955139	0.307730016	0.008988313
combined_data8	Latitude	0.338085987	0.304548895	0.008444393
combined_data8	Longitude	0.347724207	0.390225541	0.008211444
combined_data9	SOG	0.537133721	0.529079007	0.013297549
combined_data9	COG	0.399217264	0.318617723	0.007729973
combined_data9	Latitude	0.345333586	0.323018912	0.007890903
combined_data9	Longitude	0.333604711	0.305984805	0.009990515
combined_data10	SOG	0.542281564	0.546396846	0.012973715
combined_data10	COG	0.379404542	0.365348057	0.009667597
combined_data10	Latitude	0.302685411	0.316544374	0.008536748
combined_data10	Longitude	0.338813953	0.321247946	0.009697222
combined_data11	SOG	0.531274663	0.543455545	0.012367392
combined_data11	COG	0.331902836	0.318656353	0.008350634
combined_data11	Latitude	0.326113348	0.320460364	0.008795903
combined_data11	Longitude	0.307837622	0.316827101	0.0082422
combined_data12	SOG	0.516570619	0.526129706	0.014030773
combined_data12	COG	0.313371941	0.324776806	0.009868968
combined_data12	Latitude	0.325896947	0.289357226	0.008401786
combined_data12	Longitude	0.320409005	0.317365499	0.008948877
combined_data13	SOG	0.516905357	0.483984074	0.012165791
combined_data13	COG	0.321224995	0.297396491	0.00799159
combined_data13	Latitude	0.314366716	0.345305064	0.008133003
combined_data13	Longitude	0.327573842	0.343348198	0.008536902

MMSI : Passenger 211543850

DataFrame	Feature	Train RMSE	Test RMSE	Final Loss
combined_data	SOG	0.530729369	0.487013529	0.015631774
combined_data	COG	0.352889154	0.316915193	0.009244478
combined_data	Latitude	0.306286803	0.332638178	0.007403702

combined_data	Longitude	0.310845168	0.301593248	0.008927942
combined_data1	SOG	0.519293199	0.521346643	0.012356374
combined_data1	COG	0.326698076	0.323907381	0.008924375
combined_data1	Latitude	0.305825516	0.338499701	0.008093282
combined_data1	Longitude	0.334770816	0.343281082	0.007591947
combined_data2	SOG	0.525348342	0.537744814	0.014826177
combined_data2	COG	0.32190756	0.317076274	0.009379666
combined_data2	Latitude	0.312707983	0.2958543	0.008197113
combined_data2	Longitude	0.306623176	0.314662306	0.008981941
combined_data3	SOG	0.559177668	0.534371573	0.012092126
combined_data3	COG	0.312668979	0.324163167	0.009049941
combined_data3	Latitude	0.321594354	0.344615333	0.008323693
combined_data3	Longitude	0.400026221	0.381161946	0.008062035
combined_data4	SOG	0.517004666	0.521482677	0.013321246
combined_data4	COG	0.316122623	0.314102794	0.009322097
combined_data4	Latitude	0.327888134	0.302784731	0.007795618
combined_data4	Longitude	0.347011504	0.332406001	0.009108407
combined_data5	SOG	0.540787796	0.513583268	0.013014605
combined_data5	COG	0.313959566	0.307917762	0.009617024
combined_data5	Latitude	0.330943101	0.362118481	0.007342423
combined_data5	Longitude	0.32575302	0.308911482	0.008973216
combined_data6	SOG	0.524905884	0.550248954	0.01159073
combined_data6	COG	0.355746962	0.322683515	0.008594256
combined_data6	Latitude	0.328303866	0.341956777	0.008248459
combined_data6	Longitude	0.348549865	0.328417381	0.009636804
combined_data7	SOG	0.542162396	0.582325377	0.015991313
combined_data7	COG	0.324013639	0.331640459	0.010311712
combined_data7	Latitude	0.33239494	0.315989759	0.008645598
combined_data7	Longitude	0.331761312	0.347648334	0.009131872
combined_data8	SOG	0.521215103	0.507190175	0.01304166
combined_data8	COG	0.315334201	0.321818691	0.008615087
combined_data8	Latitude	0.313501669	0.321530254	0.008766502
combined_data8	Longitude	0.325543633	0.344962638	0.008688944
combined_data9	SOG	0.562717548	0.542626383	0.013691521
combined_data9	COG	0.304911444	0.303086665	0.008734142
combined_data9	Latitude	0.310854542	0.328232675	0.007710784
combined_data9	Longitude	0.31370262	0.304134097	0.00665817
combined_data10	SOG	0.595935514	0.561362164	0.012706422
combined_data10	COG	0.316221191	0.288514663	0.008870103
combined_data10	Latitude	0.315200198	0.292766086	0.008469792
combined_data10	Longitude	0.318944455	0.345893662	0.009169628
combined_data11	SOG	0.520725817	0.536631468	0.013878004
combined_data11	COG	0.311155677	0.30289023	0.007076644
combined_data11	Latitude	0.307094005	0.314039239	0.007481931
combined_data11	Longitude	0.328101268	0.326633689	0.008831435
combined_data12	SOG	0.50450807	0.520381805	0.015496506
combined_data12	COG	0.342941396	0.353831482	0.00837288
combined_data12	Latitude	0.332703347	0.335650326	0.00874646
combined_data12	Longitude	0.316034905	0.325486831	0.008340772

combined_data13	SOG	0.591805178	0.533714066	0.012893383
combined_data13	COG	0.321897534	0.303334665	0.009709371
combined_data13	Latitude	0.325272297	0.33429645	0.00817299
combined_data13	Longitude	0.333623539	0.32960136	0.006981645

MMSI : Tanker 224985000

DataFrame	Feature	Train RMSE	Test RMSE	Final Loss
combined_data	SOG	0.51981689	0.577317172	0.01164394
combined_data	COG	0.32711325	0.307175337	0.00822298
combined_data	Latitude	0.329035001	0.314782396	0.009145629
combined_data	Longitude	0.329163366	0.326689726	0.009526901
combined_data1	SOG	0.589904703	0.574340958	0.014466517
combined_data1	COG	0.357412554	0.399401569	0.006724854
combined_data1	Latitude	0.352493288	0.323265629	0.008446361
combined_data1	Longitude	0.376361877	0.32916673	0.007871728
combined_data2	SOG	0.513893432	0.517735057	0.009677215
combined_data2	COG	0.308498226	0.320174049	0.00852785
combined_data2	Latitude	0.357564462	0.323591345	0.008551644
combined_data2	Longitude	0.321593042	0.327389367	0.00791333
combined_data3	SOG	0.565965043	0.527154978	0.012305646
combined_data3	COG	0.345175156	0.296977257	0.009536516
combined_data3	Latitude	0.294097737	0.312208648	0.008414179
combined_data3	Longitude	0.315393038	0.323337747	0.007763226
combined_data4	SOG	0.540995307	0.601281958	0.012003627
combined_data4	COG	0.332770362	0.342529193	0.008652367
combined_data4	Latitude	0.341572149	0.341575615	0.008068696
combined_data4	Longitude	0.321480542	0.292011106	0.008965438
combined_data5	SOG	0.550465993	0.531619908	0.010838856
combined_data5	COG	0.336027827	0.326422403	0.008745415
combined_data5	Latitude	0.313046233	0.324201955	0.00835731
combined_data5	Longitude	0.336485842	0.341368397	0.008494279
combined_data6	SOG	0.533157528	0.485460403	0.015219954
combined_data6	COG	0.324151111	0.331927404	0.008616946
combined_data6	Latitude	0.316855655	0.324714183	0.007836491
combined_data6	Longitude	0.307801336	0.336718674	0.009116421
combined_data7	SOG	0.532197301	0.518350938	0.013385822
combined_data7	COG	0.316283205	0.336068911	0.007917585
combined_data7	Latitude	0.372290058	0.347072641	0.00993681
combined_data7	Longitude	0.307485664	0.300768822	0.010123846
combined_data8	SOG	0.549291143	0.546673155	0.012623794
combined_data8	COG	0.327122274	0.337679415	0.008911182
combined_data8	Latitude	0.303461797	0.337843404	0.008069044
combined_data8	Longitude	0.325055469	0.323664807	0.008538378
combined_data9	SOG	0.539194385	0.507981921	0.013489256
combined_data9	COG	0.330926471	0.354546803	0.009080153
combined_data9	Latitude	0.319788814	0.343985918	0.007616515

combined_data9	Longitude	0.380439012	0.337454528	0.008500831
combined_data10	SOG	0.538607349	0.562839372	0.012487713
combined_data10	COG	0.420532823	0.328812403	0.008382712
combined_data10	Latitude	0.317514591	0.308263119	0.008680791
combined_data10	Longitude	0.322780046	0.333909504	0.008334213
combined_data11	SOG	0.548947547	0.572369546	0.012485257
combined_data11	COG	0.3570698	0.312636314	0.008814654
combined_data11	Latitude	0.318404641	0.327814934	0.006783118
combined_data11	Longitude	0.336019863	0.373100443	0.007861481
combined_data12	SOG	0.490076102	0.490149068	0.013017326
combined_data12	COG	0.328114448	0.328300926	0.008367854
combined_data12	Latitude	0.338533665	0.307990815	0.009642228
combined_data12	Longitude	0.334648485	0.35073942	0.007953851
combined_data13	SOG	0.588032331	0.588403813	0.013668538
combined_data13	COG	0.314479461	0.306513103	0.007580153
combined_data13	Latitude	0.329022622	0.30082925	0.008361335
combined_data13	Longitude	0.372704699	0.34189676	0.008334095

MMSI : Cargo 244246000

DataFrame	Feature	Train RMSE	Test RMSE	Final Loss
combined_data	SOG	0.535596934	0.546550833	0.015437343
combined_data	COG	0.333456	0.287904587	0.009479699
combined_data	Latitude	0.318748766	0.324722299	0.008080964
combined_data	Longitude	0.345274451	0.362252089	0.008252644
combined_data1	SOG	0.516864575	0.493995754	0.014475299
combined_data1	COG	0.322379756	0.326720166	0.008029214
combined_data1	Latitude	0.314241803	0.312148091	0.008913786
combined_data1	Longitude	0.328831712	0.308732876	0.008331728
combined_data2	SOG	0.502634479	0.514820335	0.012840131
combined_data2	COG	0.318638405	0.344226975	0.009279503
combined_data2	Latitude	0.308252166	0.320404557	0.008655079
combined_data2	Longitude	0.318207392	0.32650729	0.007959321
combined_data3	SOG	0.51412506	0.508575718	0.014683404
combined_data3	COG	0.33233996	0.326619934	0.008436335
combined_data3	Latitude	0.321454413	0.308565799	0.008907432
combined_data3	Longitude	0.314669385	0.338946244	0.009016315
combined_data4	SOG	0.53829873	0.535480706	0.012632659
combined_data4	COG	0.329915302	0.323158464	0.008450123
combined_data4	Latitude	0.366279045	0.312645535	0.009990133
combined_data4	Longitude	0.33754197	0.339533802	0.00841604
combined_data5	SOG	0.527376508	0.516440723	0.011929975
combined_data5	COG	0.31623477	0.309607917	0.008867748
combined_data5	Latitude	0.31150392	0.302454914	0.008678146
combined_data5	Longitude	0.348045489	0.335558415	0.007779285
combined_data6	SOG	0.498300127	0.554658126	0.013073271
combined_data6	COG	0.328975156	0.305771667	0.008206828

combined_data6	Latitude	0.34050521	0.328938579	0.008798022
combined_data6	Longitude	0.337018479	0.350098214	0.009392906
combined_data7	SOG	0.522945189	0.499060606	0.013594268
combined_data7	COG	0.324824973	0.315754569	0.007438129
combined_data7	Latitude	0.312287498	0.310965905	0.008125179
combined_data7	Longitude	0.344884241	0.32674245	0.007513186
combined_data8	SOG	0.594663832	0.611377115	0.011024627
combined_data8	COG	0.3744641	0.323627416	0.00867446
combined_data8	Latitude	0.337976643	0.339513031	0.007308627
combined_data8	Longitude	0.3301069	0.31776524	0.009784699
combined_data9	SOG	0.510754552	0.52585892	0.013037534
combined_data9	COG	0.326269706	0.331573106	0.007815704
combined_data9	Latitude	0.326933266	0.337513658	0.01026552
combined_data9	Longitude	0.366191255	0.36265529	0.008675611
combined_data10	SOG	0.541062374	0.563609939	0.015373721
combined_data10	COG	0.310658867	0.312274283	0.007976374
combined_data10	Latitude	0.327176373	0.368921369	0.008991843
combined_data10	Longitude	0.33292034	0.321479126	0.007018614
combined_data11	SOG	0.506581524	0.507359907	0.010137464
combined_data11	COG	0.33842363	0.355181773	0.008009914
combined_data11	Latitude	0.324060175	0.336507278	0.008266361
combined_data11	Longitude	0.347603072	0.31264765	0.008941998
combined_data12	SOG	0.516889597	0.544942369	0.01547095
combined_data12	COG	0.329130482	0.375643108	0.008498034
combined_data12	Latitude	0.317223533	0.303018458	0.009412598
combined_data12	Longitude	0.337882818	0.334764407	0.007265632
combined_data13	SOG	0.557554814	0.512476681	0.013514252
combined_data13	COG	0.356737322	0.355485634	0.006966317
combined_data13	Latitude	0.348381141	0.305209997	0.009953302
combined_data13	Longitude	0.333553729	0.29956199	0.009144279

Here is the google drive link which contains all the testing for FbProphet and LSTM.

https://drive.google.com/drive/folders/1B0PecH9aIiLDsNVQkJNZOrm8qDA2EE-_?usp=sharing

VALIDATION CRITERIA

1. Test the model's robustness to changes in data distribution, such as new trends or seasonal patterns.
2. Ensure proper data preprocessing, including handling missing values, outliers, and feature scaling.
3. Compare the performance of LSTM and FBProphet models to identify the best model for your specific use case.
4. Consider domain-specific validation criteria relevant to the application (e.g., finance, weather forecasting).
5. Evaluate the explainability and interpretability of the model predictions.
6. Test the model's robustness to changes in data distribution, such as new trends or seasonal patterns.
7. . In each form, no field which is not null able should be left blank.
8. All numeric fields should be checked for non-numeric values
9. Use of error handling to filter out datas for wrong MMSI.

ADVANTAGES

FBProphet Model

I. It is designed for ease of use, allowing users to get started quickly with forecasting without needing to understand time series modelling in depth. This makes it applicable even to non-experts.

II. With respect to missing data and outliers, FBProphet is resistant. The algorithm can handle gaps in the data and automatically detect and model outliers which can be a big challenge when performing time series analysis.

III. Big Data has become common hence the need for scalable machine learning models that work on a wide variety of applications. This paradigm shift means that there is place for models like Prophet that occupy a large area at once. A lot of businesses as well as researchers take part in working with huge amounts of data over a long period of time.

IV. Seasonal patterns are well captured by this model making. Because it can capture daily, weekly and yearly seasonal changes on its own, it has many uses ranging from sales predictions to traffic prediction.

V. FBProphet acknowledges the existence of special events such as holidays and allows users to input their own custom seasonality terms including other specific days into the model which makes it accommodative enough for peculiar-patterning dataset forecasts.

VI. This makes it particularly useful for companies or researchers who have extensive time series datasets they are dealing with.

VII. FBProphet automatically recognizes trends in your data whether they are linear or nonlinear.

Disadvantages

i. FBProphet,however, is mostly for time series with a clear seasonality trend. It may not work well in certain situations such as when the data lacks temporal pattern or has complex, non-seasonal patterns.

ii. There are some conditions assumed by this model which include presence of additive or multiplicative seasonality. If these assumptions do not hold for the real data, then model accuracy can be undermined.

iii. On one hand, it may be ideal to use FBProphet to track long-term trends and capture seasonal variance. On the other hand, because of its high volatility; FBProphet may not always be suitable for short term forecasting.

LSTM

Advantages

i. LSTMs are impeccable at capturing extended periods of time in series of data.

ii. This can fix the problem of diminishing gradient that is often faced by other types of RNNs.

iii. Sequential data tasks have a wide application for LSTMs

iv. They capture complicated, non-linear relations within the data.

v. For varying length sequences an LSTM is a flexible model.

vi. Compared to traditional models, they are more resistant to noise.

vii. LSTM automatically learns and extracts features from raw data.

viii. These architectures can be expanded to accommodate large datasets, given their high computational resources.

DISADVANTAGES

1) The LSTM models have complex computations and require many resources to execute them well

2) It will take you longer times doing this than simpler models would do

3) In cases where only small sets are applied, these may lead to overfitting condition most especially in the case with LSTMS.

4) Thus, it needs careful hyperparameter tuning for optimal performance especially during training process

5) Information on its memory capacity and computational demand should be disclosed before purchase.

6) Traditional models are more understandable when compared to LSTMs.

7) Mostly large amount of data is required for good performances of LSTMs.

8) Weight initialization can affect performance outcomes.

Features

LSTM

Sequential Data Handling: LSTMs are meant for sequential data and as such can be perfect in time series forecasting.

Memory Cells: They have memory cells that enable them to retain information over long periods thereby overcoming the vanishing gradient problem which is common with traditional RNNs.

Gating Mechanism: It presents input, output and forget gates that control the flow of information leading to better learning of long term dependencies.

Versatility: Effective on various tasks including time series prediction, speech recognition and language modeling.

FBProphet

Automated Forecasting: FBProphet is a forecasting tool designed to handle missing data and seasonal effects with minimal data preprocessing.

Additive Model: Uses an additive model where non-linear trends are fit with yearly, weekly, and daily seasonality plus holiday effects.

Parameter Tuning: Offers intuitive parameters for manual tuning so users may include domain knowledge into the forecasting process.

Interactive Plots: Provides interactive plots which show forecast components such as trends, seasonality, and holidays.

CONCLUSION

For time series forecasting, this research project focused on comparing the FbProphet and LSTM models to give important insights into the strong points and limitations of both. In order to evaluate the models, data were sampled at 1 minute interval, 10 minutes interval, and 60

minutes interval for developing and testing these models in predicting variables such as Latitude, Longitude, Course Over Ground (COG), Speed Over Ground (SOG).

FbProphet Model:

The FbProphet model has a user-friendly interface and it is capable of modeling seasonal effects. It gave high accuracy when forecasting within a 300-minute horizon. For instance, its predictions had a standard deviation less than 5%, which implied that short term forecasts are reliable. On the other hand, outside this kind of timeline the model's accuracy reduced indicating that it cannot handle complex non-linear relationships over time.

LSTM Model:

LSTM (Long Short-Term Memory) model loss function decreased over time showing continuous learning ability since it designed with advanced architecture for learning long-term dependencies in sequential data. It also maintained prediction standard deviation at less than 5% until up to 300 minutes just like fbprophet did.

Although the LSTM is superior at modeling complex patterns, it also struggled with accuracy for extended forecasts without further tuning and data refining.

Comparative Insights:

Short-Term Forecasting: However, both models performed impressively in short-term forecasting with an accuracy of less than 5% within a standard deviation up to 300 minutes. The selection between FbProphet and LSTM for short term tasks depends on specific application needs and implementation complexity.

Long-Term Forecasting: Both models' performances dropped off beyond 300 minutes. FbProphet's simpler approach did not work well over longer periods while LSTM, although complex, also showed degraded performance suggesting additional data and tuning might be needed for long-term predictions.

Model Adaptability and Learning: This ability of LSTM to minimize its loss function over time shows that it is adaptable and has room for improvement as more data is fed into it and fine-tuning done. On the other hand, FbProphet being easier to implement may need complementary strategies to improve its long-term forecasting capabilities.

Recommendations:

When looking for short-term forecast solutions; both FbProphet as well as LSTM are useful options whose choice depends on ease of implementation as well as specific use cases.

There is a need for hybrid approaches to long-range forecasting or the improvement of current models.

Further studies should include issues such as incorporation of more variables, optimal hyperparameter search, sophisticated pre-processing techniques to deal with long-term forecasting problems.

The project was an invaluable learning experience that provided deep insights into the intricacies of time series prediction. In fact, it reinforced the importance of adaptability, problem-solving and ongoing improvement in data analysis. It is important to note that this module has also helped me acquire knowledge beyond technical skills necessary for accomplishment of tasks such as teamwork, management of projects, and bridging technology with application domains.

This project looks forward to future undertakings in a hopeful way. The groundwork laid and lessons learned open avenues for further exploration, methodology enhancement and relevant contributions within the changing realm of data science and forecasting technology.

References

Websites

- https://web.ais.dk/aisdata/
- https://facebook.github.io/prophet/
- https://facebook.github.io/prophet/docs/diagnostics.html

- https://www.sciencedirect.com/topics/computer-science/long-short-term-memory-network#:~:text=2%20Long%2Dshort%20Term%20Model,used%20in%20memory%2Dbased%20problems.

- https://www.geeksforgeeks.org/deep-learning-introduction-to-long-short-term-memory/

- https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html

The below link contains all the testing and testing code used :

- https://drive.google.com/drive/folders/1B0PecH9aIiLDsNVQkJNZOrm8qDA2EE-_?usp=sharing

Track Classification of Ship Data

Amit Kumar

Abstract : This observe makes a speciality of the classification of maritime tracks the usage of Automatic Identification System (AIS) records thru time series analysis techniques. The number one goal is to beautify maritime situational attention with the aid of correctly classifying ship tracks based on their AIS transmissions. We employed superior time series forecasting fashions, FbProphet and Long Short-Term

Memory (LSTM) networks, to analyze and expect key navigational parameters which includes Speed Over Ground (SOG), Course Over Ground (COG), Latitude, and Longitude. Data became amassed at various intervals (1 minute, 10 mins, and 60 minutes) to evaluate the fashions' performance over one-of-a-kind time horizons. Our findings suggest that each fashions deliver predictions with a trendy deviation of much less than five% as much as 300 minutes, past which the prediction accuracy diminishes. The consequences reveal the capability of those models in supplying dependable quick-term forecasts for ship tune classification, thereby contributing to enhanced navigation safety and operational performance.

Individual Contribution and Findings : In my project, I have designed the whole model from scratch.

Individual contribution to project report preparation: prepared the entire documentation of the report.