**Data Analysis and Modeling for Vessel Performance – Documentation**

**Project Overview**  
This document outlines the data analysis and modeling work aimed at understanding and predicting vessel performance. The project includes data cleaning, feature engineering, visualization, and the application of regression models to provide insights and make predictions based on operational and fuel data.

**Data Assumptions and Cleaning Steps**

**1. Data Loading and Initial Exploration**

**Objective:** Load the dataset and understand its structure, data types, and basic statistics.

**Process:**  
The primary dataset, flattened\_vesselPerformance\_copy.csv, was loaded into a pandas DataFrame.  
Initial exploration involved displaying the first few rows and data types of each column using df.head() and df.dtypes.

**2. Handling Missing Values**

**Assumption:** Columns with more than 70% missing data are unreliable for analysis or modeling.

**Process:**

* Calculated the percentage of missing values for each column.
* Dropped columns with more than 70% missing data.

**Code:**

threshold = len(df) \* 0.7

df\_cleaned = df.dropna(thresh=threshold, axis=1)

**Rationale:**  
Columns with excessive missing data are removed to ensure the analysis is based on relatively complete information.

**3. Imputing Missing Values**

**Assumption:** Key columns like Engine.PowerAtShaft, Engine.RPM, etc., may have missing values that can be inferred or imputed.

**Process:**

* Imputed missing values in skewed numerical columns (Engine.PowerAtShaft, Engine.RPM) using the median.
* Forward-filled missing values in columns such as BallastWater, CargoMetricTons, etc., assuming that missing data follows the last valid observation.
* Dropped rows with missing values in Longitude.Minutes.

**Code:**

# Median imputation for skewed numerical columns

median\_power = df['Engine.PowerAtShaft'].median()

median\_rpm = df['Engine.RPM'].median()

df['Engine.PowerAtShaft'].fillna(median\_power, inplace=True)

df['Engine.RPM'].fillna(median\_rpm, inplace=True)

# Forward fill for other columns

columns\_to\_ffill = ['BallastWater', 'BoilerIsExhaustBypassActive', 'CargoMetricTons',

'ChartererSpeedOrder', 'ChiefEngineerName']

df[columns\_to\_ffill] = df[columns\_to\_ffill].fillna(method='ffill')

# Dropping rows with missing values in 'Longitude.Minutes'

df.dropna(subset=['Longitude.Minutes'], inplace=True)

**4. Data Parsing and Transformation**

**Assumption:** The Consumptions and FuelRobs columns contain string representations of lists of dictionaries that need parsing.

**Process:**

* Parsed the string representations into actual lists of dictionaries using ast.literal\_eval.
* Exploded these columns to separate the lists of dictionaries into individual rows.
* Normalized the JSON data into separate columns for each attribute.

**Code:**

import ast

df['Consumptions'] = df['Consumptions'].apply(lambda x: ast.literal\_eval(x) if isinstance(x, str) else x)

df\_exploded = df.explode('Consumptions')

consumptions = pd.json\_normalize(df\_exploded['Consumptions'])

**5. True/False Encoding**

**Process:**  
Replaced True/TRUE with 1, False/FALSE with 0, and null values with 0 in relevant columns.

**Code:**

columns\_to\_modify = ['IsCombinatorMode', 'IsFuelChangeover', 'IsPositionWarningOverridden',

'IsShaftGeneratorRunning', 'IsSlowSteaming', 'IsTurboChargerCutOut', 'WindDirectionIsVariable']

df[columns\_to\_modify] = df[columns\_to\_modify].replace({'True': 1, 'TRUE': 1, 'False': 0, 'FALSE': 0, pd.NA: 0, None: 0}).fillna(0)

**Feature Engineering Steps and Rationale**

**1. Fuel Consumption per Nautical Mile (Fuel\_Consumption\_per\_NM)**

**Calculation:**  
Fuel\_Consumption\_per\_NM is computed by dividing Consumptions\_Parsed (that is the amount of fuel consumed) by SailedDistanceGPS. (we have all the values of this as we have found out the missing values using the Linear Regression model)

**Rationale:**  
This feature provides a normalized measure of fuel efficiency, showing fuel consumption per unit of distance traveled.

**Process:**

* Created the new feature.
* Replaced infinite values (due to division by zero) with NaN and imputed missing values with the median.

**Code:**

df['Fuel\_Consumption\_per\_NM'] = df['Consumptions\_Parsed'] / df['SailedDistanceGPS']

df['Fuel\_Consumption\_per\_NM'] = df['Fuel\_Consumption\_per\_NM'].replace([np.inf, -np.inf], np.nan)

median\_fuel\_consumption\_per\_nm = df['Fuel\_Consumption\_per\_NM'].median()

df['Fuel\_Consumption\_per\_NM'].fillna(median\_fuel\_consumption\_per\_nm, inplace=True)

**2. Principal Component Analysis (PCA)**

**Objective:** Reduce the dimensionality of air pressure and air temperature data.

**Rationale:**  
Strong correlations between certain features (e.g., AirPressure, EngineRoomAirPressure, AirPressure\_Z) suggested that PCA could help combine them into principal components.

**Process:**

* Standardized the air pressure and air temperature feature sets.
* Applied PCA to extract the first principal component for each set, creating PCA\_AirPressure and PCA\_AirTemperature.

**Code:**

pca\_set1 = ["AirPressure", "EngineRoomAirPressure", "AirPressure\_Z"]

pca\_set2 = ["AirTemperature", "AirTemperature\_Z", "EngineRoomAirTemperature"]

scaler = StandardScaler()

df[pca\_set1] = scaler.fit\_transform(df[pca\_set1])

df[pca\_set2] = scaler.fit\_transform(df[pca\_set2])

pca1 = PCA(n\_components=1)

mask\_set1 = df[pca\_set1].notnull().all(axis=1)

df.loc[mask\_set1, "PCA\_AirPressure"] = pca1.fit\_transform(df.loc[mask\_set1, pca\_set1])

pca2 = PCA(n\_components=1)

mask\_set2 = df[pca\_set2].notnull().all(axis=1)

df.loc[mask\_set2, "PCA\_AirTemperature"] = pca2.fit\_transform(df.loc[mask\_set2, pca\_set2])

df.loc[~mask\_set1, "PCA\_AirPressure"] = np.nan

df.loc[~mask\_set2, "PCA\_AirTemperature"] = np.nan

**Data Visualization and Analysis**

**Line Plots and Histograms**

**Purpose:** To visualize trends and distributions for key variables like AirPressure and AirTemperature.

**Observations:**

* Line plots showed trends over time.
* Histograms revealed the distribution of these variables, helping identify patterns or anomalies.

**Box Plots**

**Purpose:** To visualize the variability and potential outliers in air pressure and temperature.

**Observations:**

* Air pressure displayed outliers on both ends.
* Air temperature showed outliers, particularly on the lower end.

**Fuel Consumption by Engine Type and Fuel Type**

* **Analysis:** Grouped by ConsumptionTypeIDCode and FuelTypeIDCode to examine fuel consumption patterns.
* **Visualization:** Bar plots showing total fuel consumption by engine type and fuel type.

**Model Choices and Hyperparameter Tuning Strategy**

**1. Linear Regression Model**

* **Objective:** Predicted SailedDistanceGPS based on Consumptions\_Parsed.
* **Rationale:** Linear patterns observed between variables made this model a good candidate.

**Evaluation Metrics:**

* **Mean Squared Error (MSE):** 1775.19
* **R-squared (R²):** 0.7444

**Conclusion:** The linear regression model performed well, explaining around 74.44% of the variance in SailedDistanceGPS. It was used to impute missing values for this column.

**2. Neural Network Model**

* **Objective:** For predicting Engine.PowerAtShaft based on Consumptions\_Parsed.
* **Conclusion:** The neural network model performed poorly, as indicated by its negative R² value, leading to its exclusion in favor of simpler models. Thus, we have used other methods to fill the missing data.

**Real-Time Fuel Management Recommendations for Varuna Marine Services**

**The following strategies are proposed to optimize fuel consumption and improve operational efficiency for Varuna Marine Services based on fuel type analysis and real-time data:**

**1. Optimize HFO Consumption:**

* **Real-Time Adjustments:  
  HFO requires high viscosity management and heating. Adjust engine settings based on consumption patterns to avoid overuse.** 
  + **Action: Implement predictive analytics to monitor engine load and speed, optimizing HFO use and reducing emissions.**
  + **MGOLS offers lower sulfur content, making it environmentally friendly. Switch between MGOLS modes based on operational conditions for better fuel efficiency**

**2. Dynamic Fuel Switching Based on Engine Load:**

* **Automated Fuel Management: Engine fuel consumption varies significantly with operational conditions. Automating fuel switching based on real-time data as different engines need different fuels (main vs auxiliary). ensures that the most appropriate fuel is used for each engine.**
  + **Action: Implement a fuel management system that switches fuel types automatically depending on the engine load, reducing waste and optimizing efficiency for both main and auxiliary engines.**

**3. Monitor Engine Power and Fuel Consumption Correlation:**

* **Engine Performance Optimization: Main engines consume most of the fuel, and their efficiency is key to overall fuel optimization. Monitoring the relationship between power output and fuel consumption can highlight inefficiencies.**
  + **Action: Use power consumption data to identify underperforming engines. Adjust engine power output settings to balance fuel consumption and maintain operational efficiency.**

**4. Auxiliary Engine Efficiency:**

* **Low Auxiliary Engine Fuel Consumption: Auxiliary engines are not as fuel-intensive as the main engine, but their operational efficiency can still be optimized. The auxiliary engines' operational hours, load, and performance data should be continuously monitored.**
  + **Recommendation: Monitor auxiliary engine performance and operational hours to avoid underutilization or unnecessary idling. Implement load balancing strategies to ensure they are running efficiently during off-peak periods.**

**Conclusion:**

**By adopting real-time fuel management strategies, Varuna Marine Services can optimize fuel consumption, reduce emissions, and lower operational costs. Integrating predictive maintenance, dynamic fuel switching, and IoT-based monitoring systems will improve overall efficiency and sustainability in maritime operations.**

**Deployment Instructions**

1. **Build Docker Image:**

docker build -t jupyter-container .

1. **Run Docker Container:**

docker run -p 8888:8888 jupyter-container

1. **Access Jupyter Notebook:**
   * Open the URL displayed in the terminal (e.g., http://127.0.0.1:8888/?token=your\_token\_here) in your browser to access the notebook.

Ensure Requirements.txt and try.ipynb are in the same directory as the Dockerfile.