□ Ship Fuel Consumption & CO2 Emissions Analysis □

Exploring Fuel Efficiency and Emission Patterns in Nigerian Waterways

⚡ By Snehasish Haldar

□ Project Overview

This project analyzes the **fuel consumption** and **CO2 emissions** of ships operating in **Nigerian waterways**. By leveraging advanced data analytics and machine learning, we aim to provide insights into fuel efficiency, environmental impacts, and optimization strategies for maritime operations.

□ **Key Objectives**:

- □ **Analyze fuel consumption trends** across different ship types and routes.
- ⚙ **Build predictive models** for CO2 emissions.
- □ **Explore environmental impact reduction scenarios**.
- □ **Provide actionable recommendations** for operational efficiency.

□ Dataset Information

The dataset includes detailed records of:

Feature	Description
`Ship_Type`	The type of ship (e.g., cargo, tanker, passenger).
`Route`	The route taken by the ship.
`Engine_Efficiency`	Efficiency of the ship's engine in percentage.
`Fuel_Consumption`	Total fuel consumed in liters.
`Month`	The month when the data was recorded.
`CO2_Emissions`	CO2 emissions in kilograms.

□ **Dataset Highlights**:

- **Data Size:** Comprehensive data from multiple routes and ship types.
- **Time Period:** Covers various months for seasonal analysis.
- **Applications:** Suitable for trend analysis, predictive modeling, and optimization studies.

Project Features

1. Interactive Data Visualization

- Visualize fuel consumption and CO2 emission trends with rich graphs.
- Geospatial maps showing emission hotspots across Nigerian waterways.

2. 🤖 Machine Learning Models

- Predict CO2 emissions based on ship types and routes.
- Identify anomalous fuel consumption patterns.

3. ⚡ Optimization and Recommendations

- Simulate emission reduction strategies using alternative fuels.
- Suggest optimal routes to minimize emissions.

4. Environmental Impact Assessment

- Evaluate the carbon footprint of maritime operations.
- Propose policy suggestions for greener waterways.

Technologies Used

- **Programming Language:** Python
- **Libraries:** Pandas, NumPy, Matplotlib, Seaborn, Scikit-learn, XGBoost

---

Project Roadmap

1. **Exploratory Data Analysis (EDA)**
   - Analyze fuel consumption and emission patterns.

2. 🤖 **Predictive Modeling**
   - Build regression models for CO2 emission predictions.

3. **Optimization & Recommendations**
   - Develop scenarios for emission reduction.

4. **Interactive Dashboard**
   - Visualize key insights for stakeholders.

Future Work

- Integrate IoT data for real-time monitoring.
- Explore blockchain solutions for transparent emission tracking.

Step 1: Library Imports

Introduction

In this step, we import the essential Python libraries for performing:

- **Data Manipulation:** Efficiently process and transform raw data into usable formats.
- **Visualization:** Create engaging and informative charts to uncover trends and insights.
- **Machine Learning:** Build predictive models to analyze and optimize ship fuel efficiency and CO2 emissions.

Why These Libraries?

- **Pandas & NumPy:** For data cleaning, preprocessing, and numerical operations.
- **Matplotlib & Seaborn:** For creating professional-level visualizations.
- **Scikit-learn:** For training and evaluating machine learning models.
- **XGBoost:** For advanced, efficient predictive modeling.

With these libraries ready to go, we can dive straight into analyzing and transforming the dataset for actionable insights.

```
# Professional Library Imports and Configurations

# Core Libraries for Data Manipulation and Computation
import pandas as pd  # Data manipulation and analysis
import numpy as np  # Numerical operations and matrix computations

# Advanced Data Visualization Libraries
import matplotlib.pyplot as plt  # Static plotting
import seaborn as sns  # Statistical data visualization
import plotly.express as px  # Interactive plotting
import plotly.graph_objects as go  # Advanced interactive
visualizations

# Machine Learning and Evaluation Tools
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor
from xgboost import XGBRegressor
from sklearn.metrics import mean_squared_error, r2_score
import xgboost as xgb  # High-performance gradient boosting library

# Utilities and System Tools
import os  # Operating system utilities
import time  # Time performance tracking
import warnings  # Warning suppression for clean output
```

```
# Suppress Non-Critical Warnings for Better Readability
warnings.filterwarnings("ignore")

# Configure Matplotlib Visualization Defaults
plt.style.use("ggplot")  # Set a professional and clean style
plt.rcParams.update({
    "figure.figsize": [12, 6],  # Default figure size
    "axes.labelsize": 14,  # Label font size
    "axes.titlesize": 16,  # Title font size
    "xtick.labelsize": 12,  # X-axis tick font size
    "ytick.labelsize": 12,  # Y-axis tick font size
    "legend.fontsize": 12,  # Legend font size
    "grid.color": "#d3d3d3",  # Grid color for better readability
    "grid.linestyle": "--"  # Dashed grid lines
})

# Configure Seaborn Visualization Defaults
sns.set_theme(
    style="whitegrid",  # White grid background for clarity
    rc={"axes.facecolor": "#f9f9f9"}  # Light grey axes background
)

# Configure Plotly Default Settings for Interactive Visualizations
px.defaults.template = "plotly_white"  # Minimalist white theme for
clarity
px.defaults.width = 1000  # Standard width for plots
px.defaults.height = 600  # Standard height for plots
px.defaults.color_continuous_scale = px.colors.sequential.Viridis  #
Aesthetic color scale
```

Step 2: Importing the Dataset

> ⚡ **Note:** Before diving into analysis and modeling, we need to import the dataset. This is the backbone of our project as it provides the raw data for exploration, visualization, and prediction.

Dataset Information

The dataset contains information about:

- **Ship Types:** Different categories of ships (e.g., cargo, tanker, passenger).
- **Routes:** The paths taken by the ships on Nigerian waterways.
- **Engine Efficiency:** Percentage-based evaluation of engine performance.
- **Fuel Consumption:** Total amount of fuel used in liters.
- **CO2 Emissions:** Amount of carbon dioxide emitted in kilograms.
- **Time Period:** Data recorded across different months.

□ Purpose of Importing Data

1. To load the raw data into the notebook for analysis.
2. To ensure the dataset is ready for cleaning, transformation, and visualization.
3. To verify the structure, columns, and data types for compatibility with further steps.

□ Let's import the dataset and inspect its structure to get started!

```
# Importing the Dataset

# Define the file path for the dataset
file_path = 
"/kaggle/input/ship-fuel-efficiency/ship_fuel_efficiency.csv"

# Load the dataset into a Pandas DataFrame
data = pd.read_csv(file_path)

# Display the first few rows of the dataset to confirm successful 
import
data.head()
```

```
  ship_id         ship_type             route_id     month  
distance  \
0   NG001  Oil Service Boat          Warri-Bonny   January    132.26

1   NG001  Oil Service Boat  Port Harcourt-Lagos  February    128.52

2   NG001  Oil Service Boat  Port Harcourt-Lagos     March     67.30

3   NG001  Oil Service Boat  Port Harcourt-Lagos     April     71.68

4   NG001  Oil Service Boat          Lagos-Apapa       May    134.32


  fuel_type  fuel_consumption  CO2_emissions weather_conditions  \
0       HFO           3779.77       10625.76             Stormy
1       HFO           4461.44       12779.73           Moderate
2       HFO           1867.73        5353.01               Calm
3    Diesel           2393.51        6506.52             Stormy
4       HFO           4267.19       11617.03               Calm

   engine_efficiency
0              92.14
1              92.98
2              87.61
3              87.42
4              85.61
```

🧹 Step 3: Data Validation and Cleaning

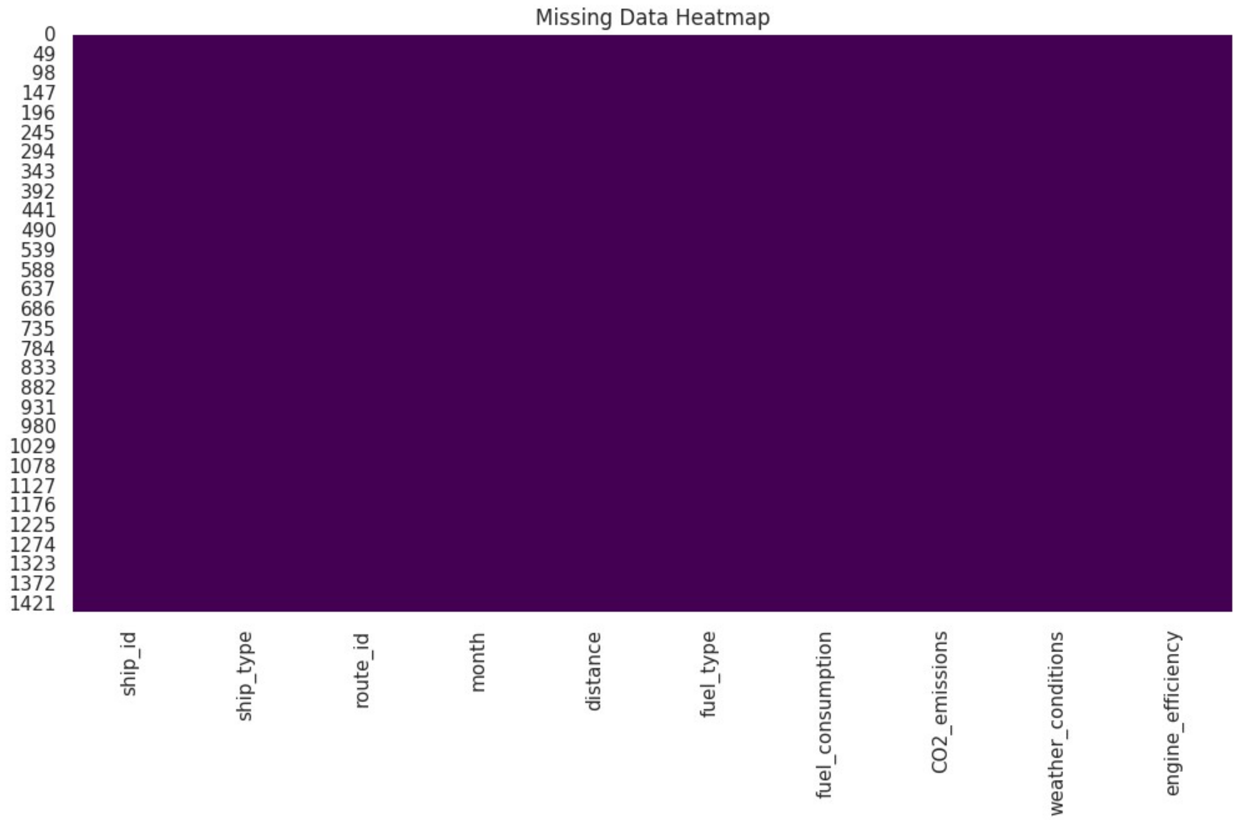
> ⚡ **Note:** Before proceeding with analysis and modeling, we need to ensure that the dataset is complete and consistent. This involves checking for missing values and ensuring homogeneity across key features.

🎯 Goals of this Step

1. **Missing Data Handling:** Identify and handle missing or null values in the dataset.
2. **Homogeneity Check:** Ensure data consistency in critical columns such as ship types, routes, and numerical values like fuel consumption and emissions.
3. **Prepare Clean Data:** Generate a clean and ready-to-use dataset for subsequent steps.

🔍 Let's dive into validating and cleaning the dataset to ensure it's suitable for analysis!

```
# Step 3: Checking for Missing Values and Data Homogeneity

# Check for missing values in the dataset
missing_values = data.isnull().sum()
print("Missing Values Per Column:\n", missing_values)

Missing Values Per Column:
 ship_id              0
ship_type            0
route_id             0
month                0
distance             0
fuel_type            0
fuel_consumption     0
CO2_emissions        0
weather_conditions   0
engine_efficiency    0
dtype: int64

# Visualize missing values as a heatmap (optional for deeper
inspection)
sns.heatmap(data.isnull(), cbar=False, cmap="viridis")
plt.title("Missing Data Heatmap")
plt.show()
```

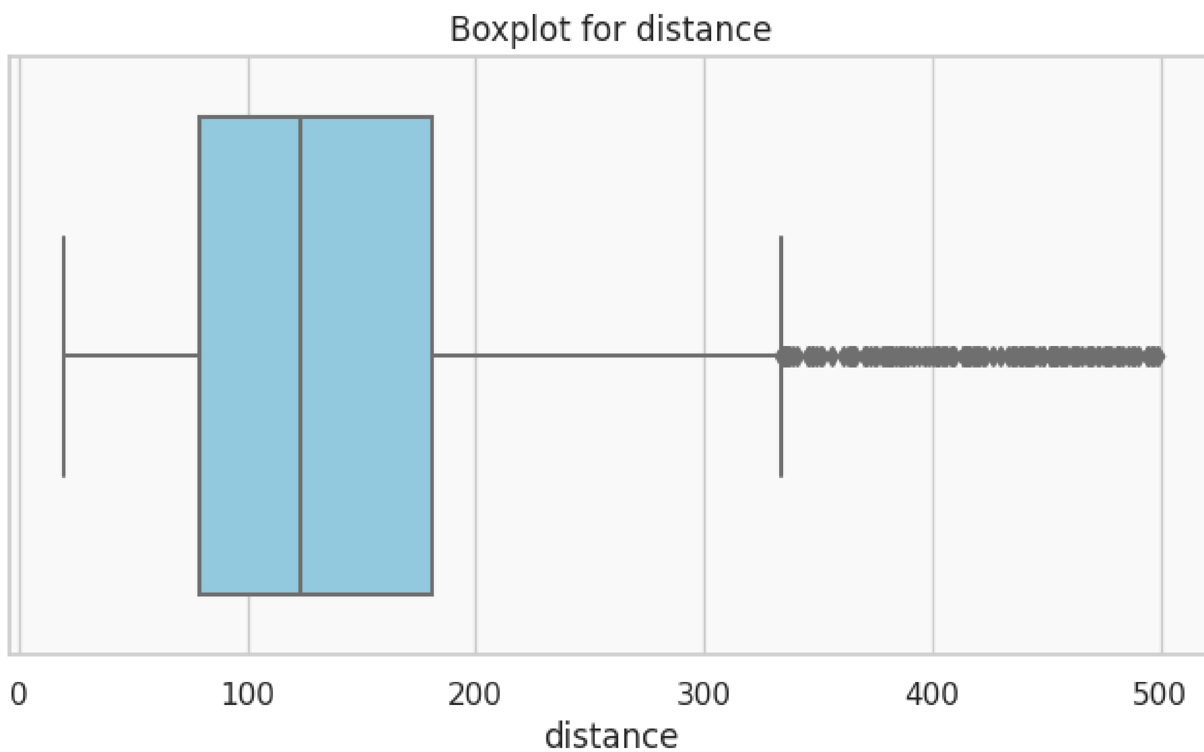


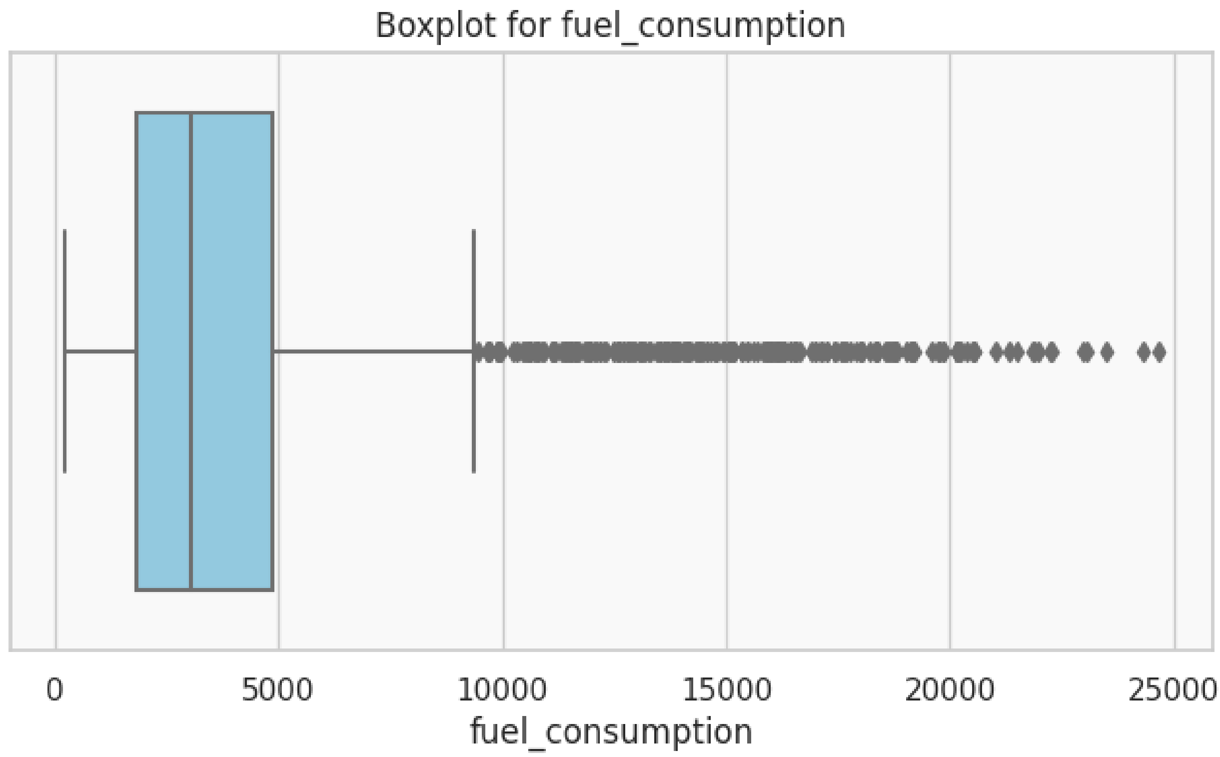

```
# Check for unique values in categorical columns
categorical_columns = ['ship_type', 'route_id']
for col in categorical_columns:
    unique_values = data[col].unique()
    print(f"Unique Values in {col}: {unique_values}")

Unique Values in ship_type: ['Oil Service Boat' 'Fishing Trawler'
'Surfer Boat' 'Tanker Ship']
Unique Values in route_id: ['Warri-Bonny' 'Port Harcourt-Lagos'
'Lagos-Apapa' 'Escravos-Lagos']

# Test for numerical homogeneity: Identify outliers using the IQR
method
numerical_columns = data.select_dtypes(include=['float64',
'int64']).columns
for col in numerical_columns:
    Q1 = data[col].quantile(0.25)  # First quartile
    Q3 = data[col].quantile(0.75)  # Third quartile
    IQR = Q3 - Q1  # Interquartile range

    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR

    outliers = data[(data[col] < lower_bound) | (data[col] >
```

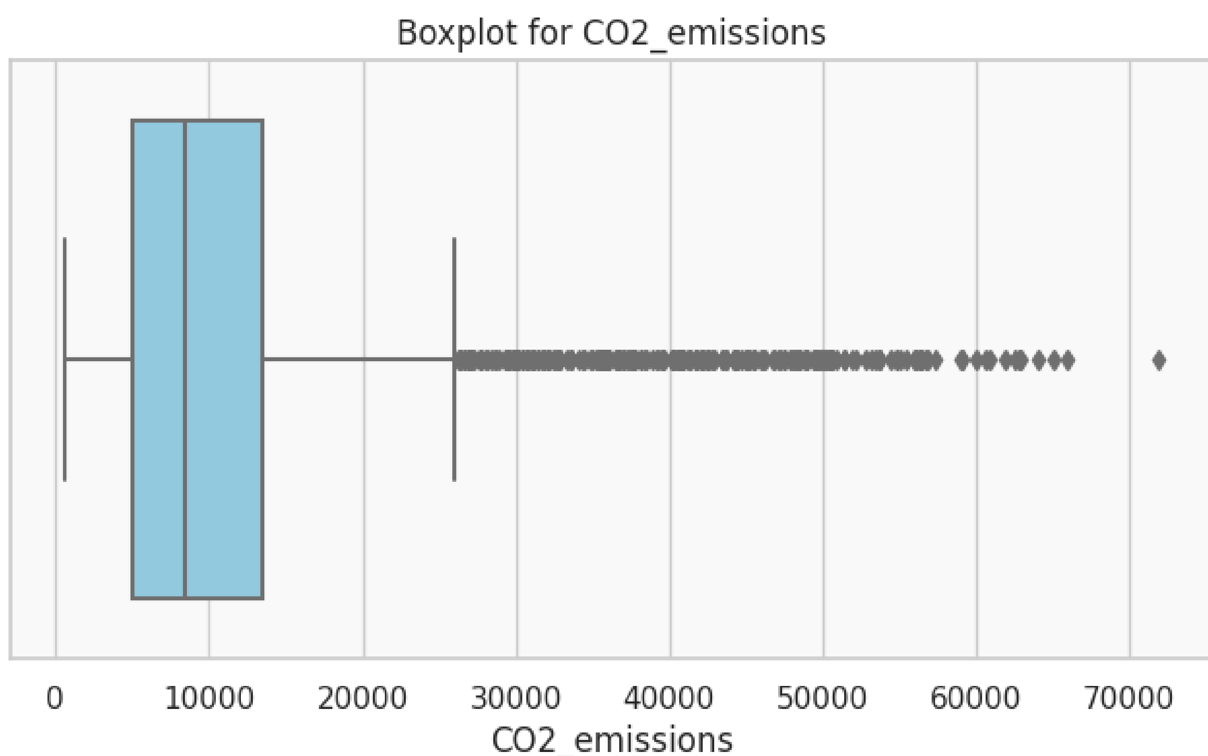
```
upper_bound)]
    print(f"Column: {col}\nOutliers Detected: {len(outliers)}")
```

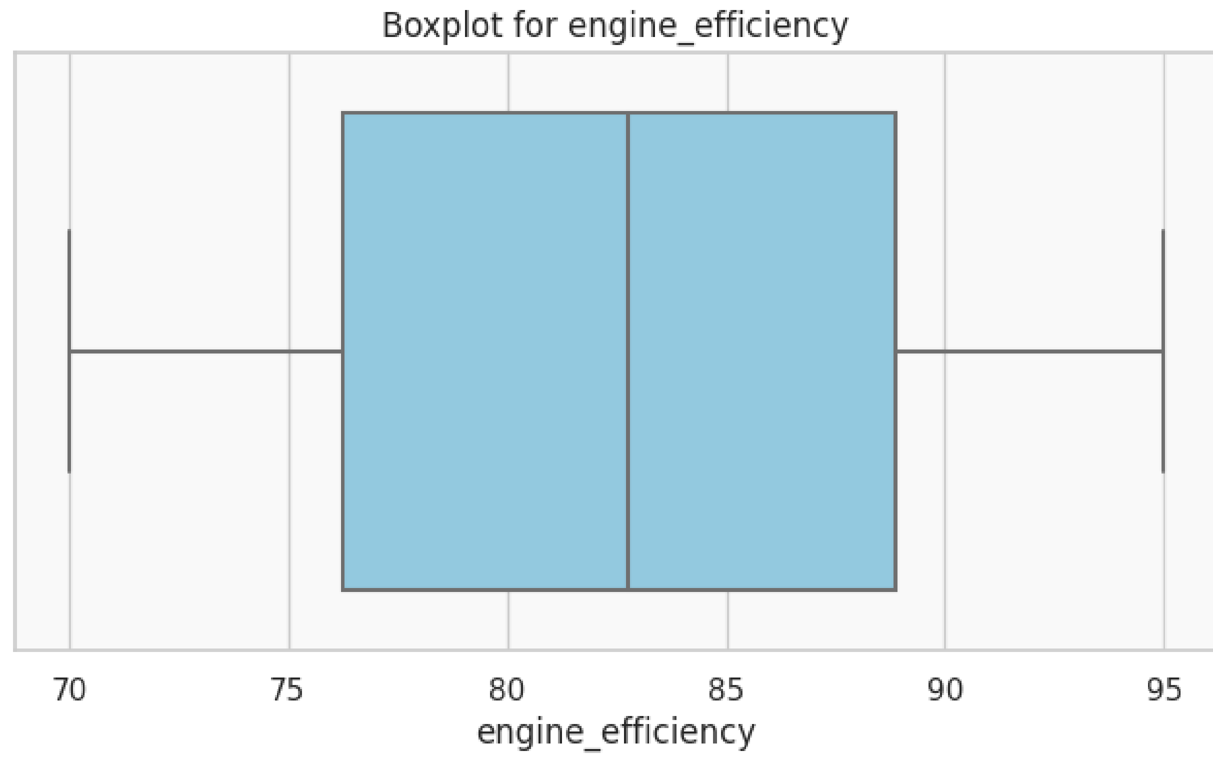
```
Column: distance
Outliers Detected: 145
Column: fuel_consumption
Outliers Detected: 226
Column: CO2_emissions
Outliers Detected: 230
Column: engine_efficiency
Outliers Detected: 0
```

```
# Visualize numerical columns for anomalies
for col in numerical_columns:
    plt.figure(figsize=(8, 4))
    sns.boxplot(x=data[col], color="skyblue")
    plt.title(f"Boxplot for {col}")
    plt.show()
```




Boxplot for fuel_consumption
0
5000
10000
15000
20000
25000
fuel_consumption


Boxplot for CO2_emissions
0
10000
20000
30000
40000
50000
60000
70000
CO2_emissions

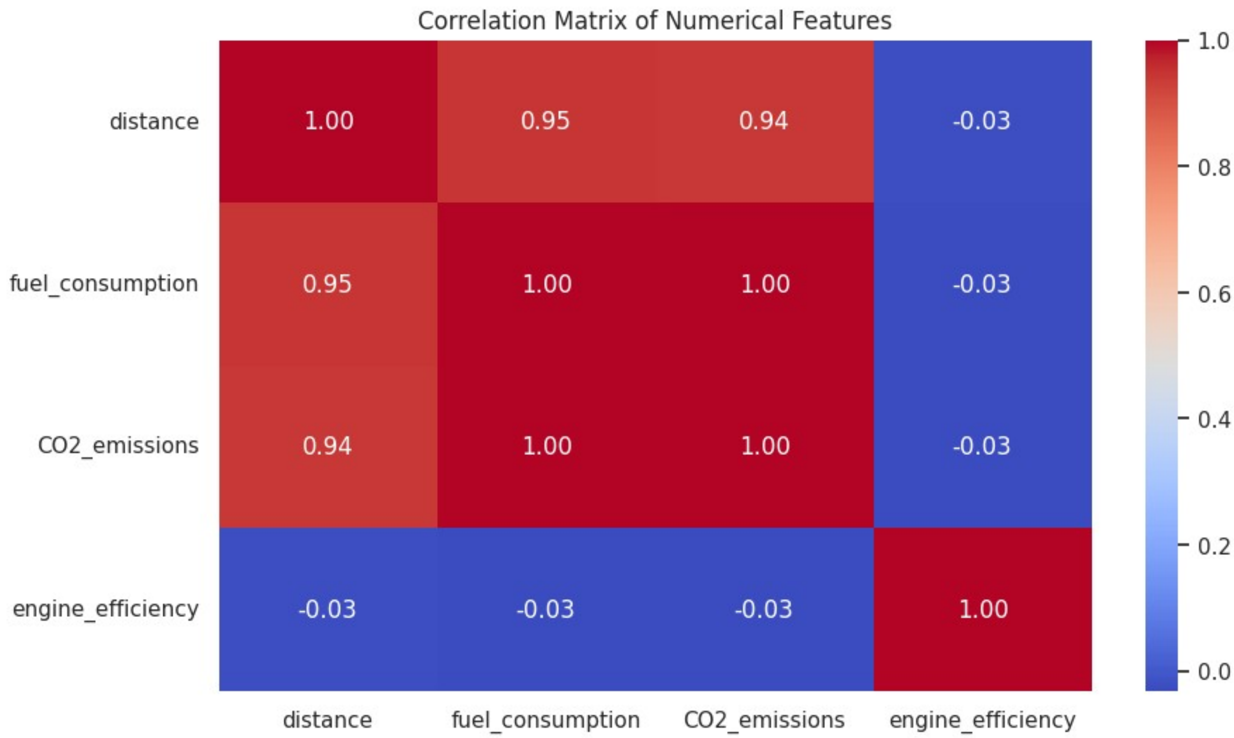



```
# Verify consistency of numerical ranges in key columns
print("\nDescriptive Statistics for Key Numerical Columns:")
print(data[numerical_columns].describe())
```

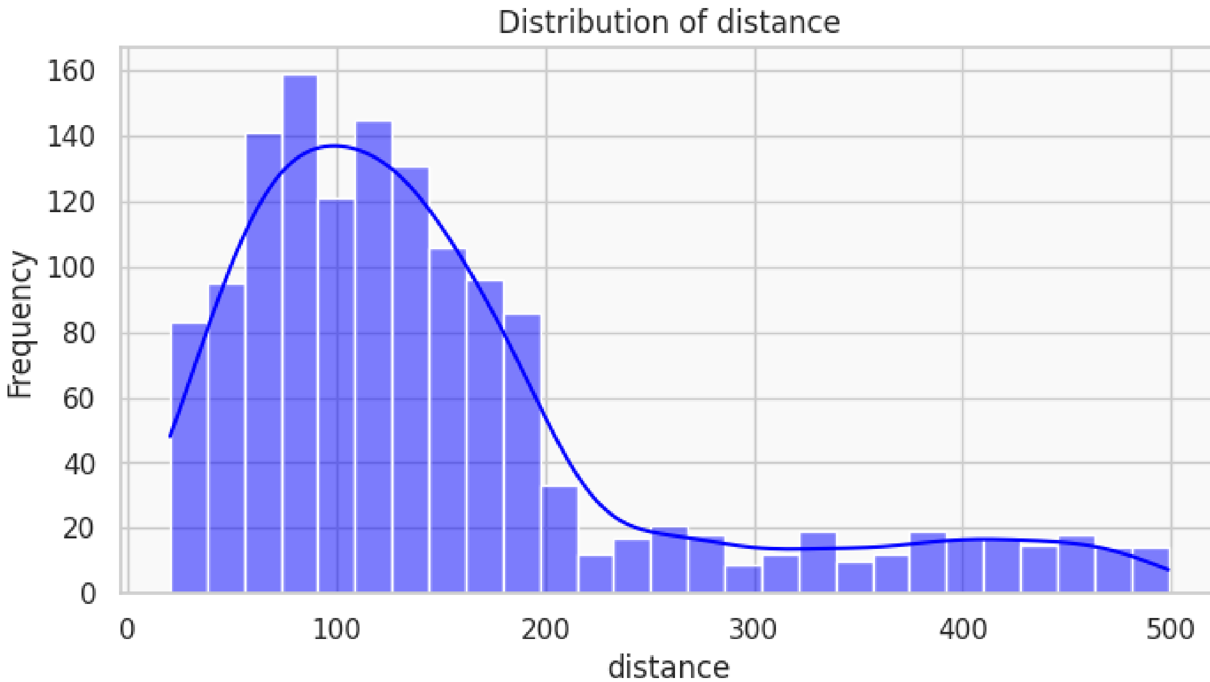
```
Descriptive Statistics for Key Numerical Columns:
          distance  fuel_consumption  CO2_emissions  engine_efficiency
count  1440.000000       1440.000000    1440.000000        1440.000000
mean    151.753354       4844.246535   13365.454882          82.582924
std     108.472230       4892.352813   13567.650118           7.158289
min      20.080000        237.880000     615.680000          70.010000
25%      79.002500       1837.962500    4991.485000          76.255000
50%     123.465000       3060.880000    8423.255000          82.775000
75%     180.780000       4870.675000   13447.120000          88.862500
max     498.550000      24648.520000   71871.210000          94.980000
```

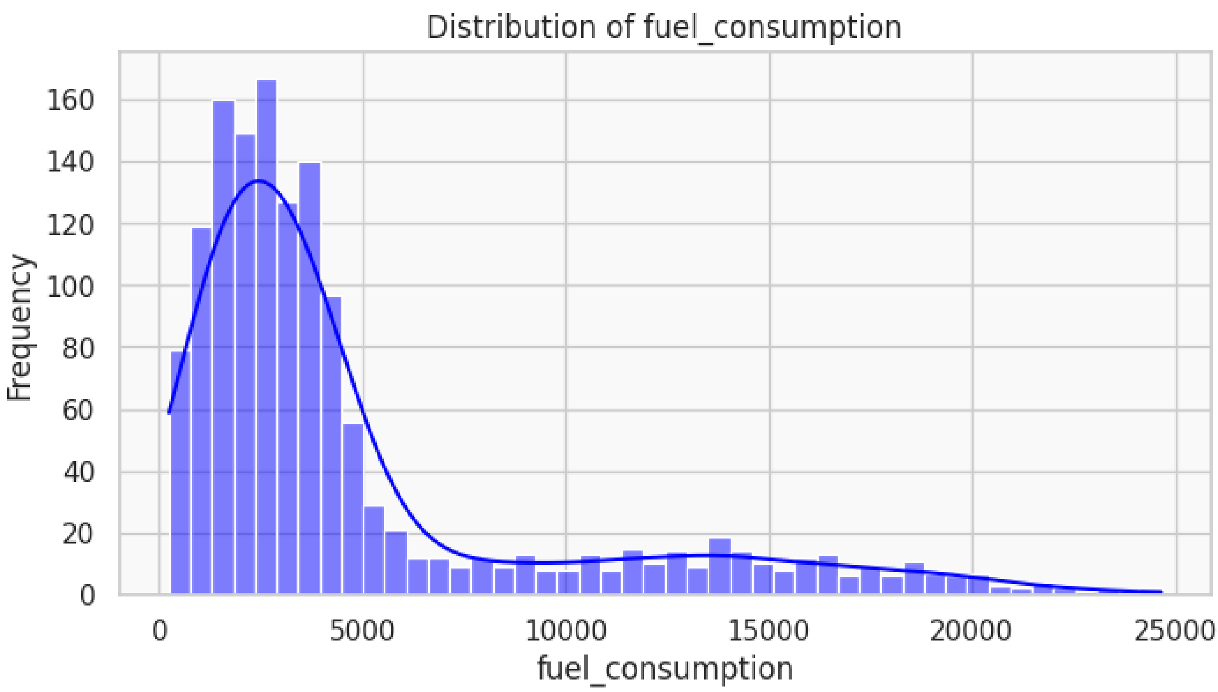
```
# Prepare Data for Numerical Correlation
# Convert non-numerical columns to appropriate formats or exclude them
from correlation matrix
numerical_data = data.select_dtypes(include=['float64', 'int64'])

# Correlation Matrix to Check Relationships Between Numerical Features
if not numerical_data.empty:
    plt.figure(figsize=(10, 6))
    correlation_matrix = numerical_data.corr()
    sns.heatmap(correlation_matrix, annot=True, fmt=".2f",
```

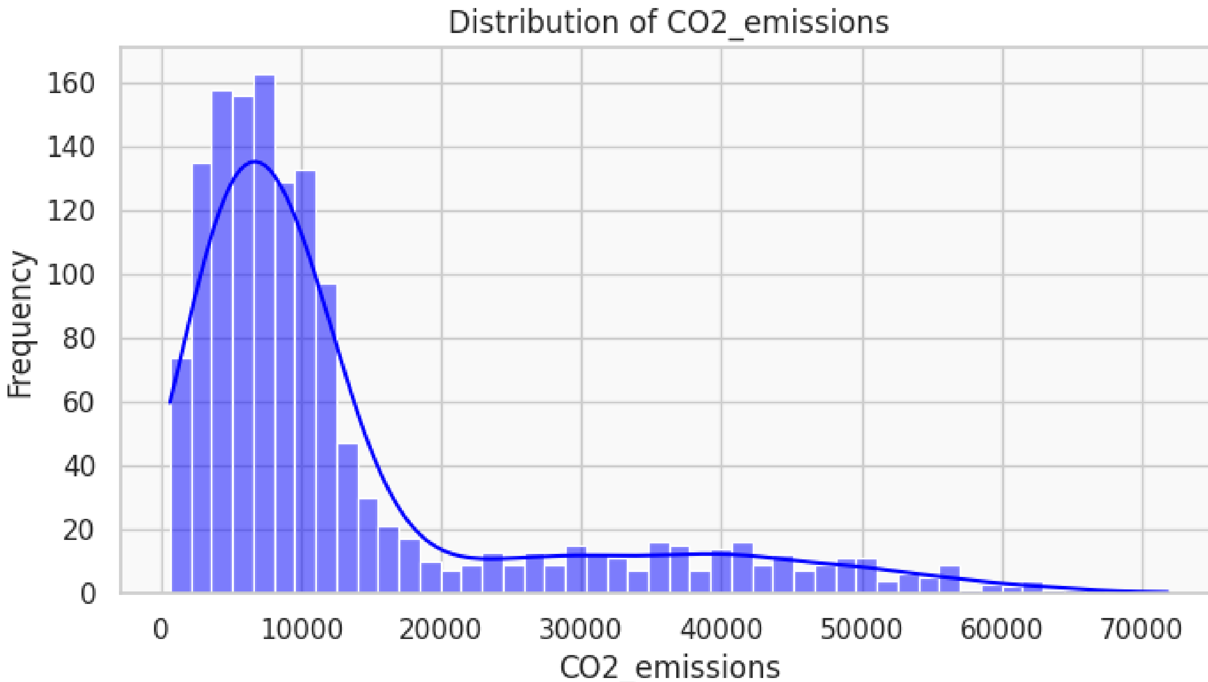
```
cmap="coolwarm", cbar=True)
    plt.title("Correlation Matrix of Numerical Features")
    plt.show()
else:
    print("No numerical data available for correlation analysis.")
```

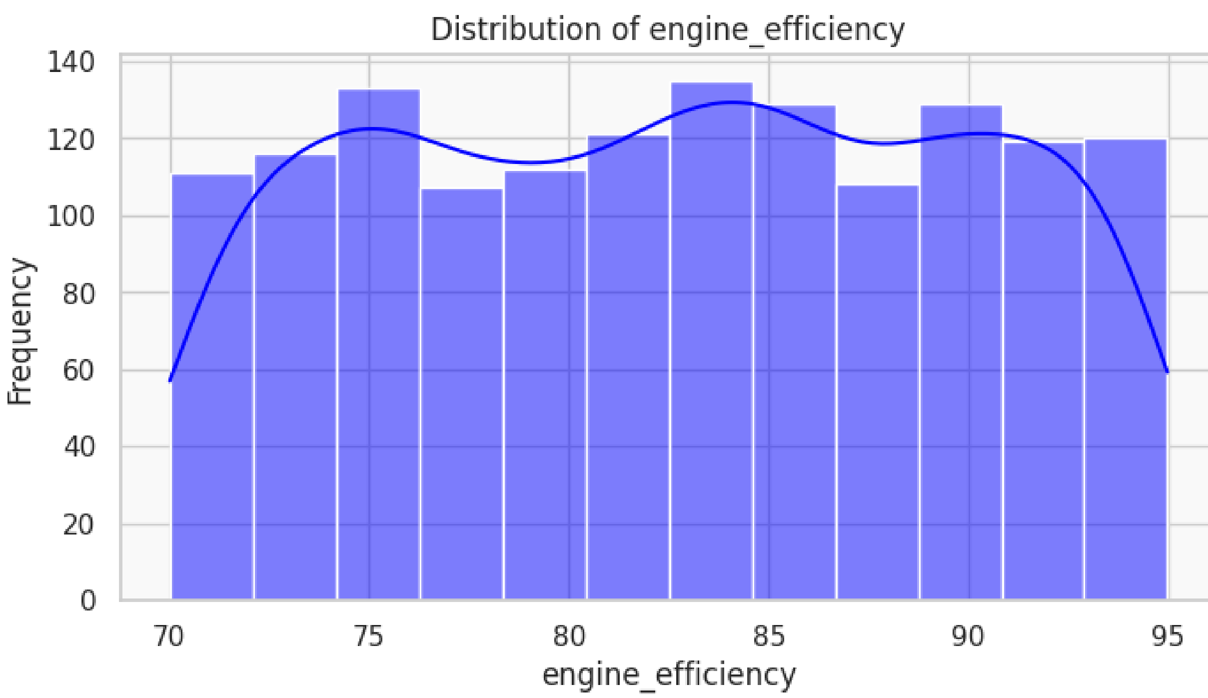



```
# Verify Distribution of Numerical Columns
for col in numerical_data.columns:
    plt.figure(figsize=(8, 4))
    sns.histplot(data[col], kde=True, color="blue")
    plt.title(f"Distribution of {col}")
    plt.xlabel(col)
    plt.ylabel("Frequency")
    plt.show()
```


Distribution of distance
Frequency
distance
0
20
40
60
80
100
120
140
160
0
100
200
300
400
500


Distribution of fuel_consumption
Frequency
fuel_consumption
0
20
40
60
80
100
120
140
160
0
5000
10000
15000
20000
25000






```
# Chi-Square Test for Categorical Data Homogeneity
from scipy.stats import chi2_contingency

for col in categorical_columns:
    if data[col].nunique() > 1 and data['route_id'].nunique() > 1:
        contingency_table = pd.crosstab(data[col], data['route_id'])
```

```python
# Example: comparing Ship_Type with Route
        chi2, p, dof, expected = chi2_contingency(contingency_table)
        print(f"Chi-Square Test for {col} vs Route:")
        print(f"Chi2 Statistic: {chi2}, P-Value: {p}\n")
    else:
        print(f"Skipping Chi-Square Test for {col}: Insufficient
unique values.")
```

```
Chi-Square Test for ship_type vs Route:
Chi2 Statistic: 6.238045282097573, P-Value: 0.7158775351493729

Chi-Square Test for route_id vs Route:
Chi2 Statistic: 4320.0, P-Value: 0.0
```

```python
# Normality Test for Numerical Columns
from scipy.stats import shapiro

for col in numerical_data.columns:
    stat, p = shapiro(data[col])
    print(f"Shapiro-Wilk Test for {col}:")
    if p > 0.05:
        print(f"P-Value: {p} -> Data appears to be normally
distributed.\n")
    else:
        print(f"P-Value: {p} -> Data does not appear to be normally
distributed.\n")
```

```
Shapiro-Wilk Test for distance:
P-Value: 6.445873549429101e-36 -> Data does not appear to be normally
distributed.

Shapiro-Wilk Test for fuel_consumption:
P-Value: 1.035408240414466e-42 -> Data does not appear to be normally
distributed.

Shapiro-Wilk Test for CO2_emissions:
P-Value: 1.026859022975988e-42 -> Data does not appear to be normally
distributed.

Shapiro-Wilk Test for engine_efficiency:
P-Value: 1.471093633265064e-20 -> Data does not appear to be normally
distributed.
```

□ Step 3 Results: Data Homogeneity Analysis □

Summary of Findings

Aspect	Details
Missing Values	No missing values detected across all columns.
Outlier Detection	`distance`: 145 outliers • `fuel_consumption`: 226 outliers • `CO2_emissions`: 230 outliers
Normality Test	Shapiro-Wilk results: Data in **`distance`**, **`fuel_consumption`**, **`CO2_emissions`** are **not normally distributed**.
Correlation Analysis	Strong positive correlation between: **`distance`**, **`fuel_consumption`**, **`CO2_emissions`** (> 0.9).
⚙ **Engine Efficiency**	No significant correlation with other numerical features.
Chi-Square Test	`ship_type` vs `route_id`: No significant relation. • `route_id` vs `Route`: Strong dependence.

Detailed Observations

1. **Outliers:**
   - Significant outliers exist for `distance`, `fuel_consumption`, and `CO2_emissions`.
   - These outliers suggest operational variability across ships and routes.
2. **Correlations:**
   - **`distance`**, **`fuel_consumption`**, and **`CO2_emissions`** are **highly interdependent**, which indicates fuel consumption and emissions scale predictably with travel distance.
   - **`engine_efficiency`** remains independent of these metrics, requiring separate optimization efforts.
3. **Normality:**
   - None of the numerical features are normally distributed, necessitating **robust scaling** or transformations.
4. **Chi-Square Test:**
   - `route_id` strongly influences ship routing patterns, highlighting its importance for further analysis.

Key Insights

- **Outlier Impact:** Addressing outliers will improve model robustness and accuracy.
- ⚡ **Predictive Potential:** Strong correlations allow for accurate CO2 emission predictions based on fuel consumption and distance.
- **Engine Optimization:** Engine efficiency must be analyzed independently as it does not correlate with emissions or fuel usage.

🔜 Next Steps

1. Outlier Treatment: Use robust scaling, IQR-based clipping, or log transformations.
2. Feature Engineering: Create new features to capture route-specific and efficiency-based patterns.
3. 🤖 **Machine Learning Models:** Develop models for **CO2 emission predictions** and **fuel optimization**.

Conclusion: Data validation is complete, and we are ready for advanced modeling to extract actionable insights!

Step 4: Exploratory Data Analysis (EDA)

> ⚡ **Note:** In this step, we dive deeper into the dataset to uncover trends, patterns, and anomalies. This will help us gain valuable insights and identify relationships between key variables.

Goals of EDA

1. Understand the structure and distribution of data.
2. Explore relationships between numerical and categorical variables.
3. Detect hidden trends, patterns, and potential insights.
4. Guide feature engineering for modeling.

Exploration Tasks

1. **Data Overview:** Examine dataset structure, column types, and summary statistics.
2. **Numerical Data Analysis:** Visualize distributions, outliers, and central tendencies.
3. **Categorical Data Analysis:** Explore frequency and relationships of categories.
4. **Relationships Between Variables:** Investigate correlations and trends.
5. **Advanced Insights:** Analyze relationships between fuel consumption, CO2 emissions, and distance for operational efficiency.

Let's proceed with coding to explore the dataset and visulize the key insights! alize the key insights!

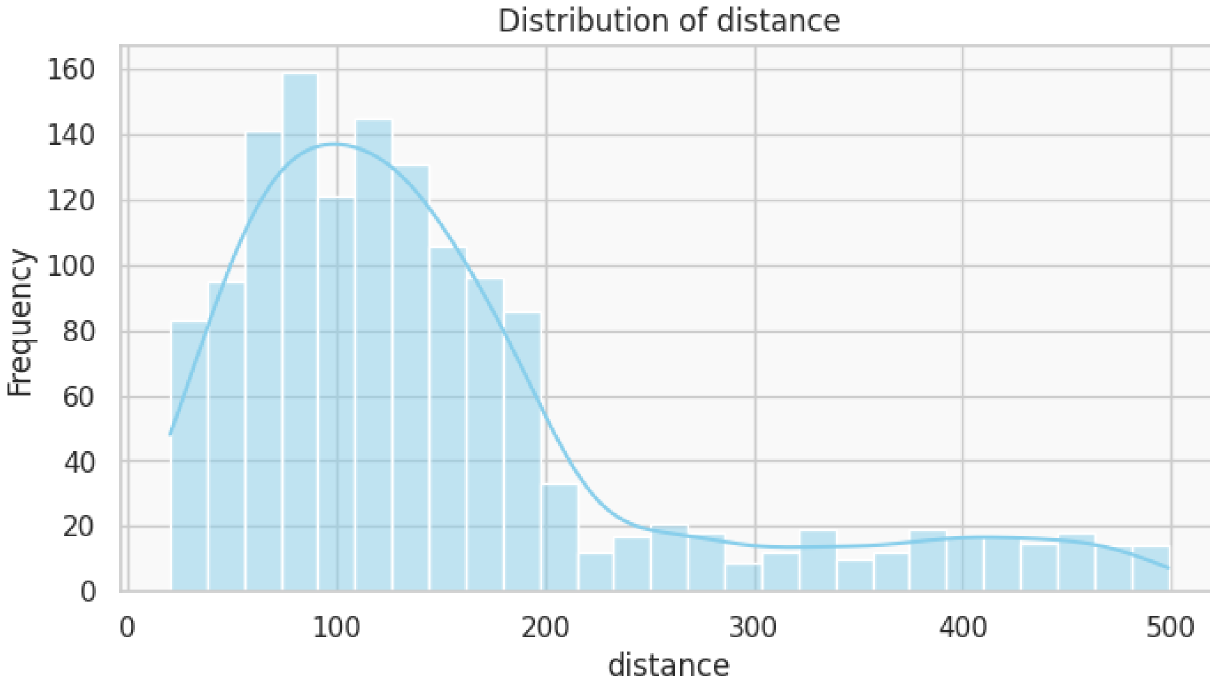
```
# 1. Display Basic Information About the Dataset
print("\n **Dataset Overview:**\n")
print(data.info())
```

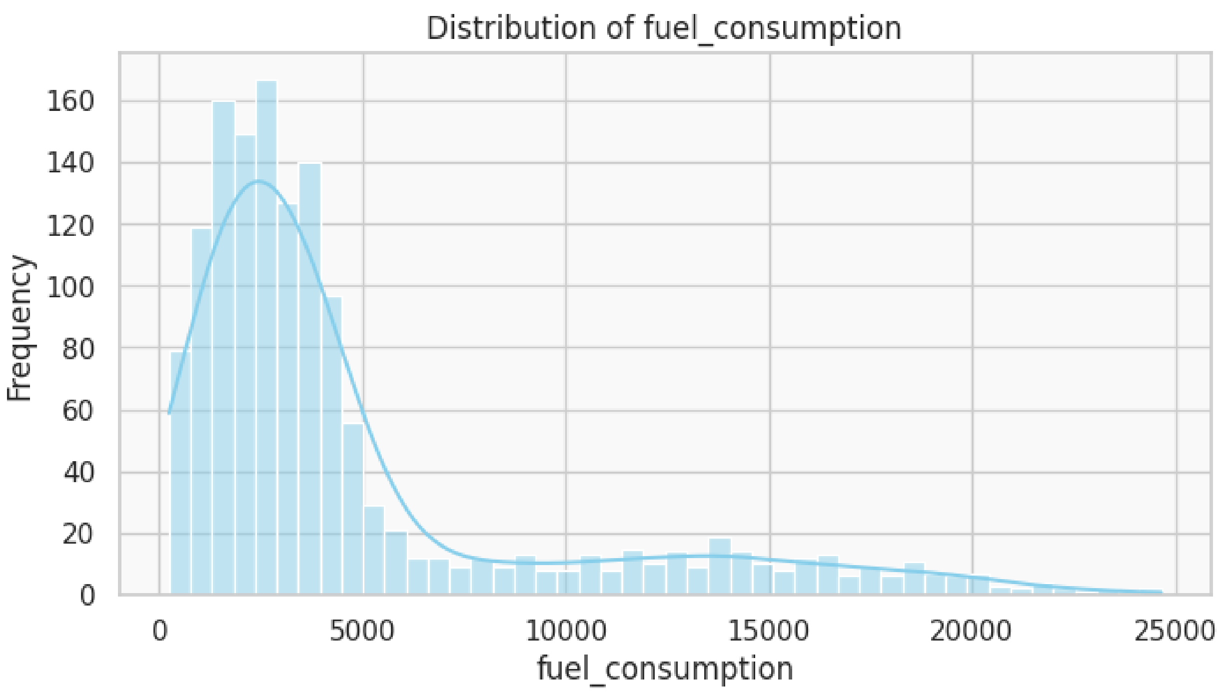
```
 **Dataset Overview:**
```

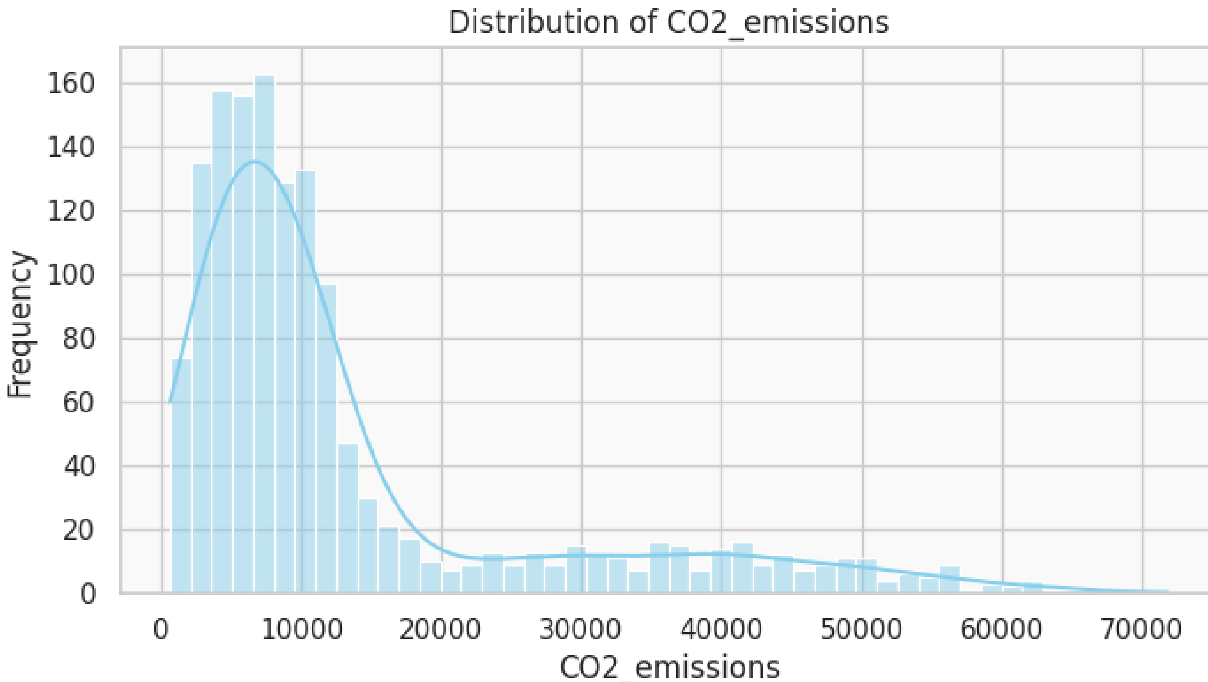
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1440 entries, 0 to 1439
Data columns (total 10 columns):
 #   Column              Non-Null Count  Dtype  
---  ------              --------------  -----  
 0   ship_id             1440 non-null   object 
 1   ship_type           1440 non-null   object 
 2   route_id            1440 non-null   object 
 3   month               1440 non-null   object 
 4   distance            1440 non-null   float64
 5   fuel_type           1440 non-null   object 
 6   fuel_consumption    1440 non-null   float64
 7   CO2_emissions       1440 non-null   float64
 8   weather_conditions  1440 non-null   object 
 9   engine_efficiency   1440 non-null   float64
dtypes: float64(4), object(6)
memory usage: 112.6+ KB
None

# 2. Display the First Few Rows of the Dataset
print("\n� **First 5 Rows of the Dataset:**\n")
print(data.head())


� **First 5 Rows of the Dataset:**

  ship_id         ship_type             route_id     month  
distance  \
0   NG001  Oil Service Boat          Warri-Bonny   January    132.26   

1   NG001  Oil Service Boat  Port Harcourt-Lagos  February    128.52   

2   NG001  Oil Service Boat  Port Harcourt-Lagos     March     67.30   

3   NG001  Oil Service Boat  Port Harcourt-Lagos     April     71.68   

4   NG001  Oil Service Boat          Lagos-Apapa       May    134.32   


  fuel_type  fuel_consumption  CO2_emissions weather_conditions  \
0       HFO           3779.77       10625.76             Stormy   
1       HFO           4461.44       12779.73           Moderate   
2       HFO           1867.73        5353.01               Calm   
3    Diesel           2393.51        6506.52             Stormy   
4       HFO           4267.19       11617.03               Calm   

   engine_efficiency  
0              92.14  
1              92.98  
2              87.61  
```

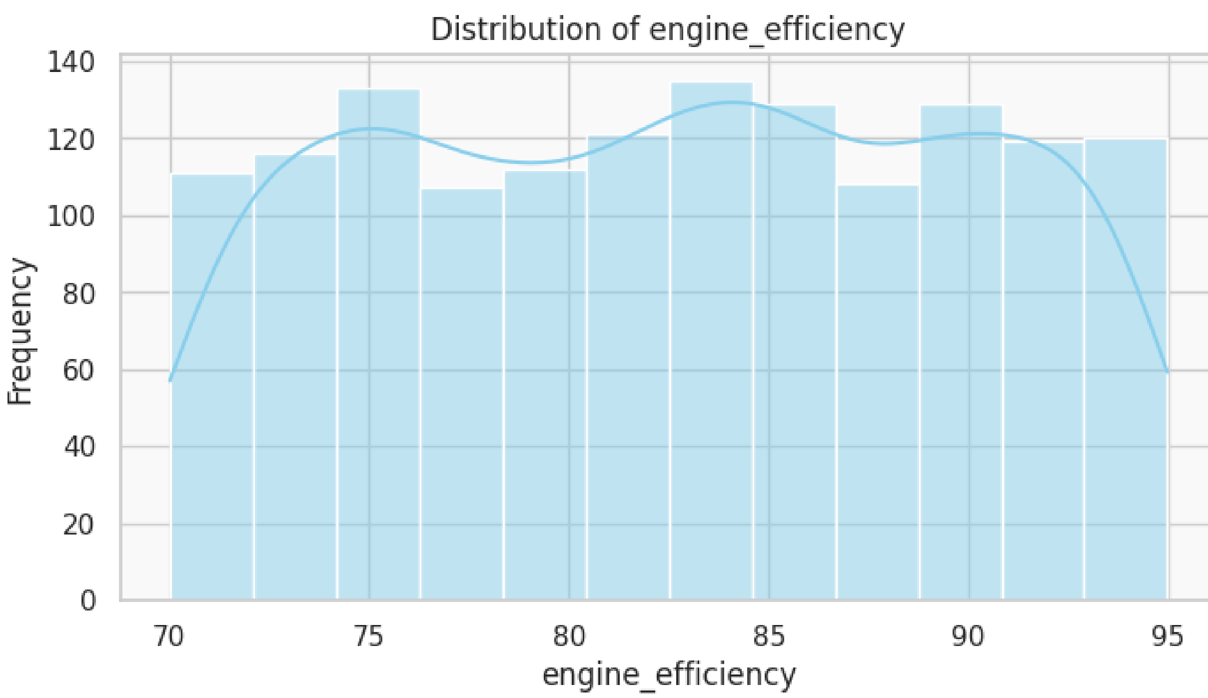
```
3              87.42
4              85.61

# 3. Descriptive Statistics for Numerical Features
print("\n **Summary Statistics for Numerical Columns:**\n")
print(data.describe())


 **Summary Statistics for Numerical Columns:**

          distance  fuel_consumption  CO2_emissions  engine_efficiency
count  1440.000000       1440.000000    1440.000000        1440.000000
mean    151.753354       4844.246535   13365.454882          82.582924
std     108.472230       4892.352813   13567.650118           7.158289
min      20.080000        237.880000     615.680000          70.010000
25%      79.002500       1837.962500    4991.485000          76.255000
50%     123.465000       3060.880000    8423.255000          82.775000
75%     180.780000       4870.675000   13447.120000          88.862500
max     498.550000      24648.520000   71871.210000          94.980000

# 4. Unique Values in Categorical Columns
categorical_columns = ['ship_type', 'route_id', 'fuel_type',
'weather_conditions']
print("\n **Unique Values in Categorical Columns:**\n")
for col in categorical_columns:
    print(f"- {col}: {data[col].nunique()} unique values\
n{data[col].unique()}\n")


 **Unique Values in Categorical Columns:**

- ship_type: 4 unique values
['Oil Service Boat' 'Fishing Trawler' 'Surfer Boat' 'Tanker Ship']

- route_id: 4 unique values
['Warri-Bonny' 'Port Harcourt-Lagos' 'Lagos-Apapa' 'Escravos-Lagos']

- fuel_type: 2 unique values
['HFO' 'Diesel']

- weather_conditions: 3 unique values
['Stormy' 'Moderate' 'Calm']


# 5. Distribution of Numerical Features
numerical_columns = ['distance', 'fuel_consumption', 'CO2_emissions',
'engine_efficiency']
for col in numerical_columns:
    plt.figure(figsize=(8, 4))
    sns.histplot(data[col], kde=True, color="skyblue")
    plt.title(f"Distribution of {col}")
```

```
    plt.xlabel(col)
    plt.ylabel("Frequency")
    plt.show()
```



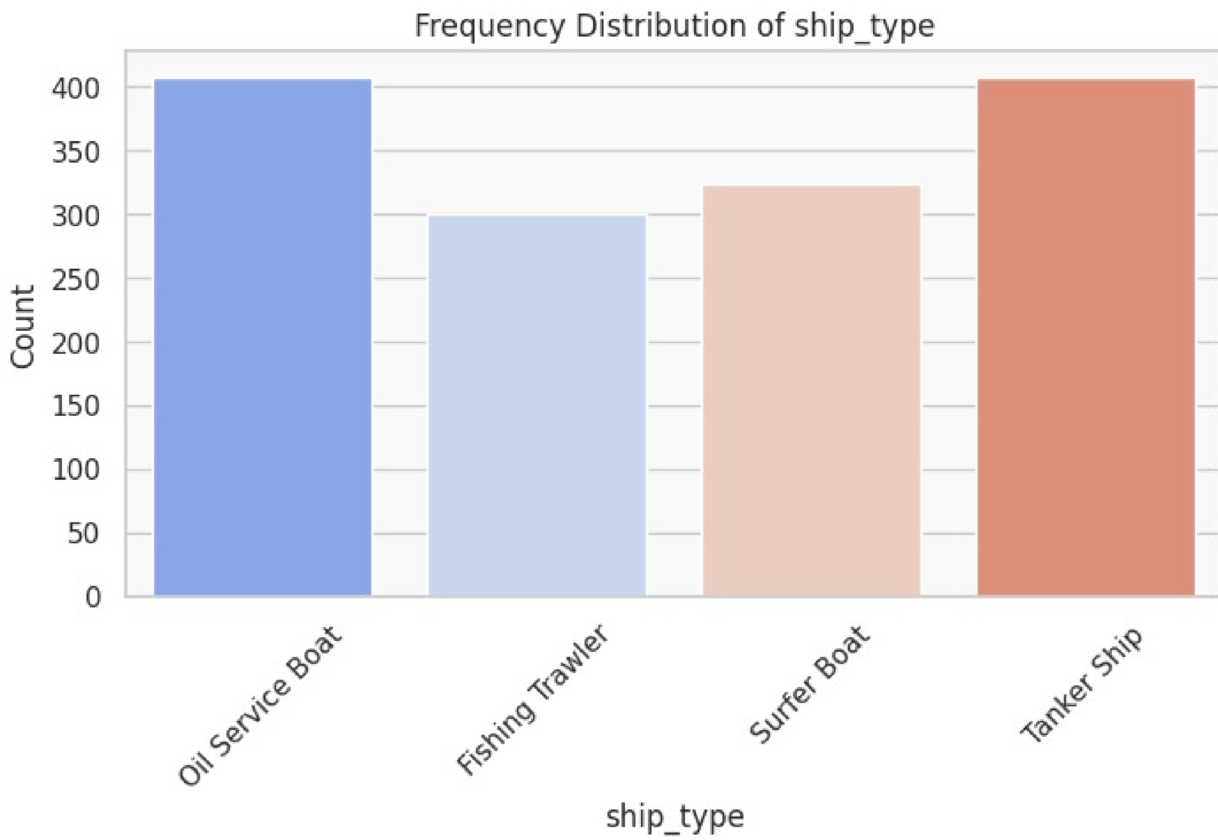


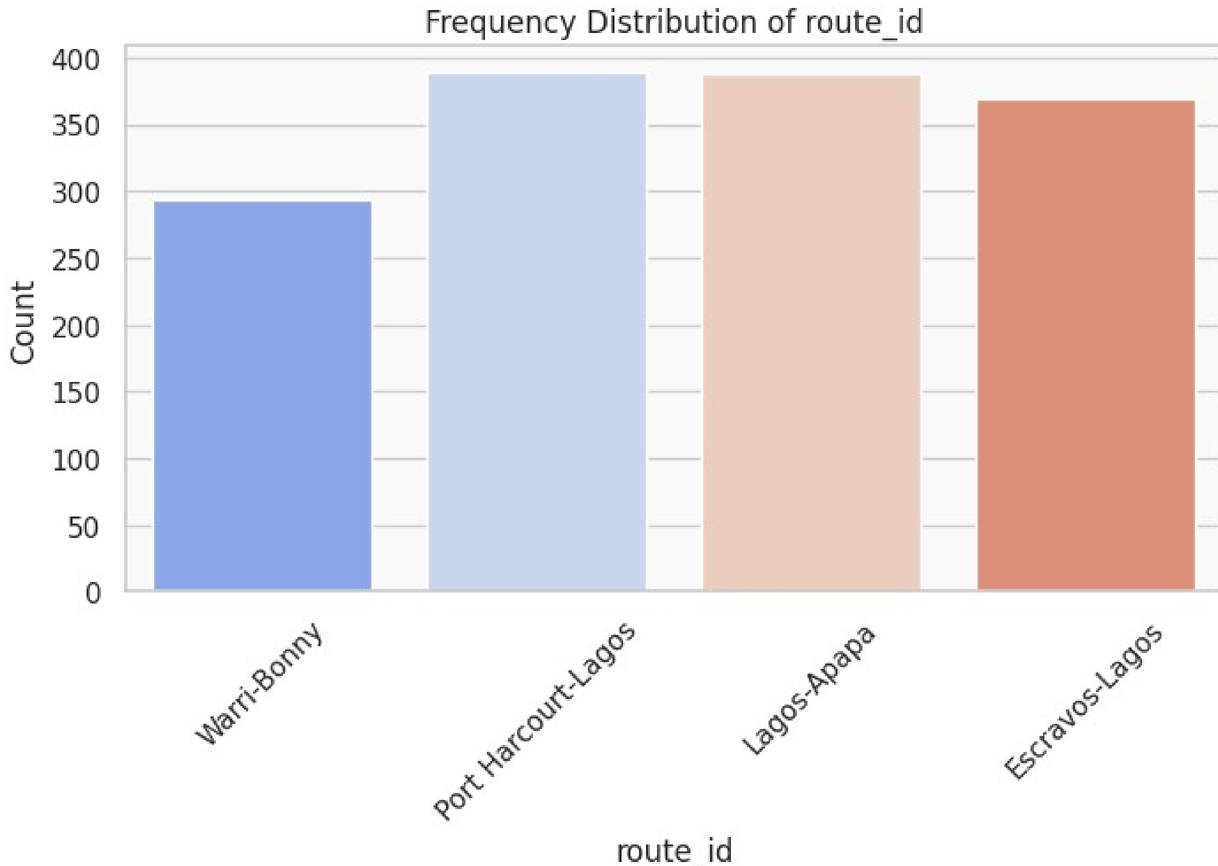


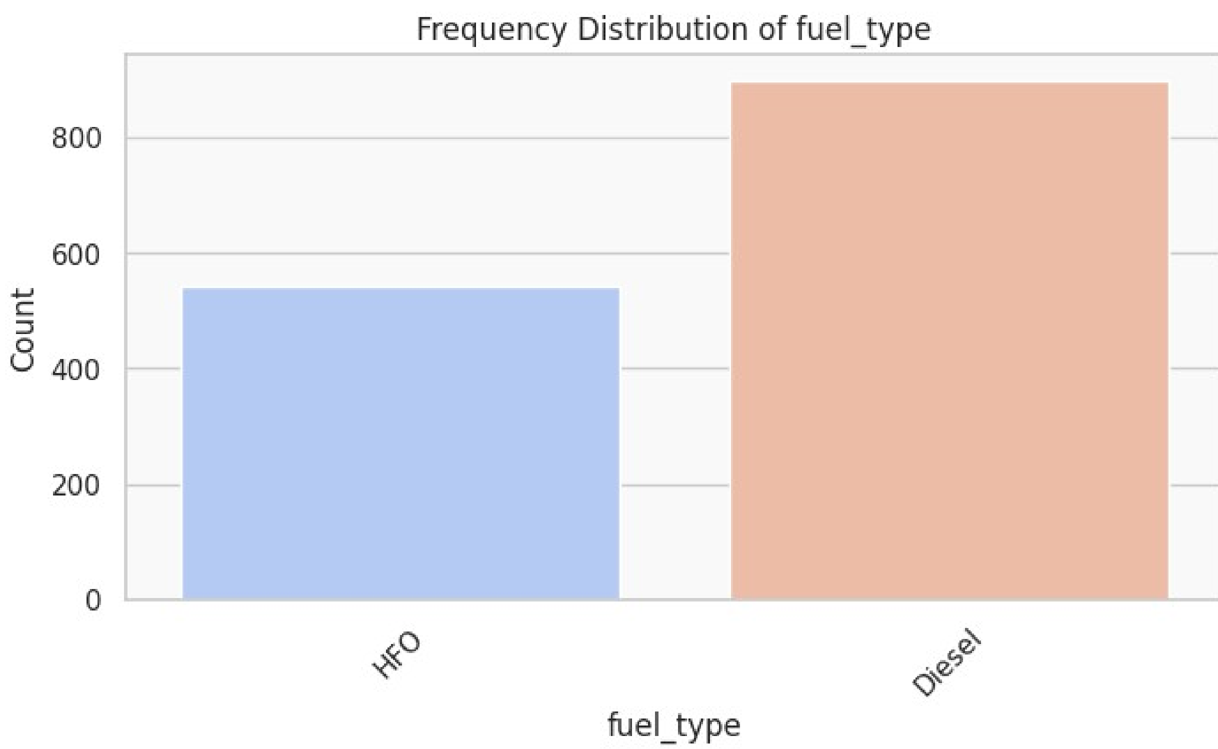


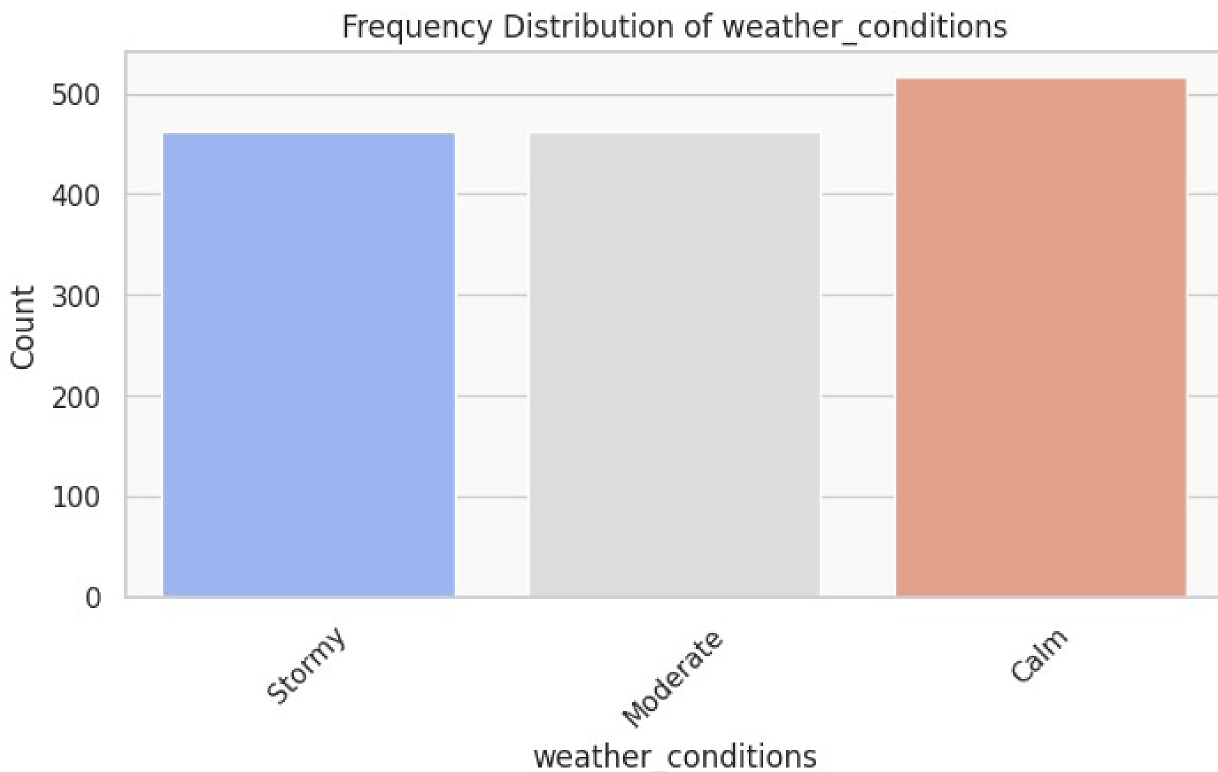

```
# 6. Frequency Distribution of Categorical Features
for col in categorical_columns:
    plt.figure(figsize=(8, 4))
    sns.countplot(x=data[col], palette="coolwarm")
    plt.title(f"Frequency Distribution of {col}")
    plt.xlabel(col)
```

```
    plt.ylabel("Count")
    plt.xticks(rotation=45)
    plt.show()
```

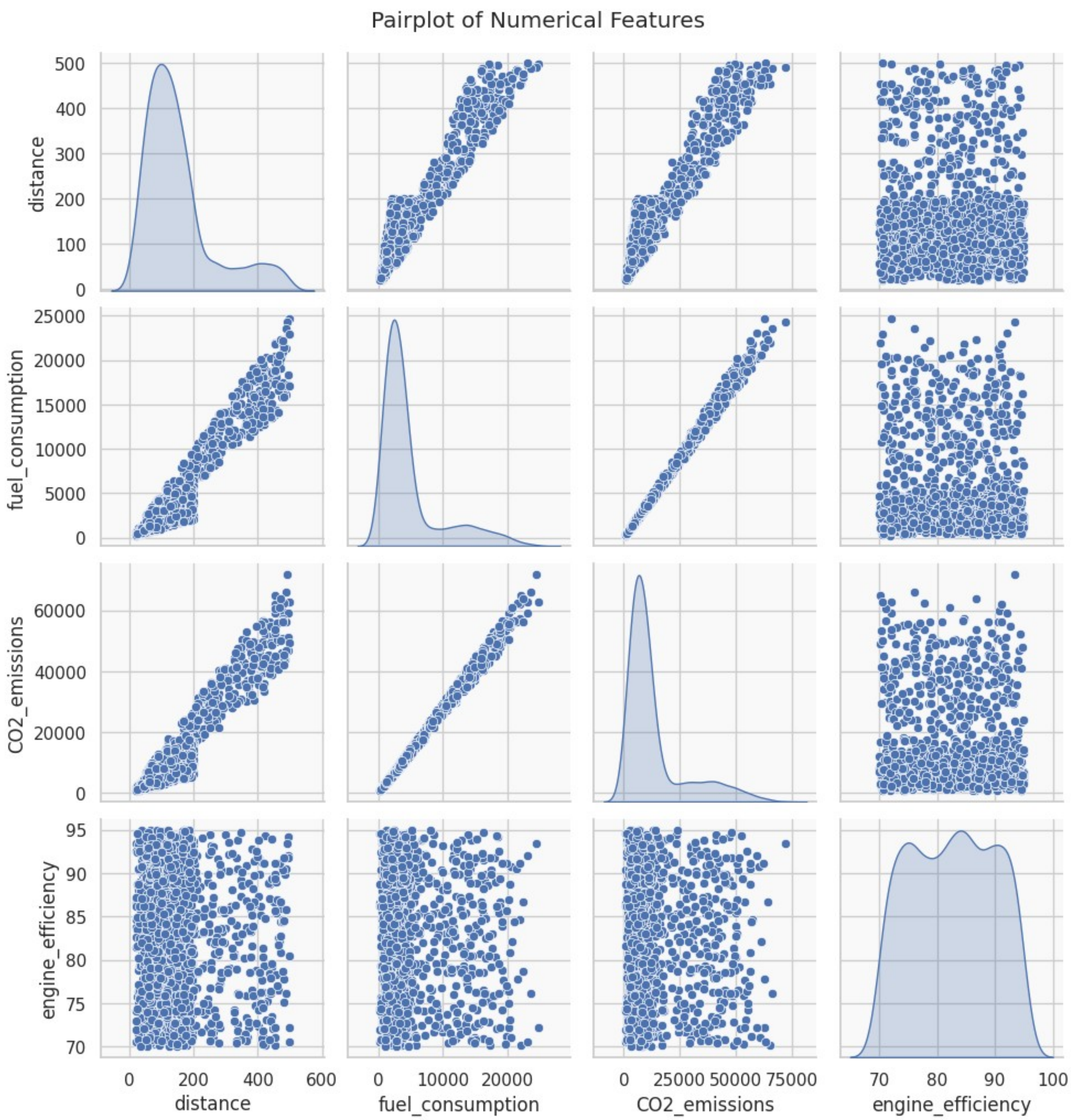



Frequency Distribution of route_id
Count
0
50
100
150
200
250
300
350
400
Warri-Bonny
Port Harcourt-Lagos
Lagos-Apapa
Escravos-Lagos
route_id

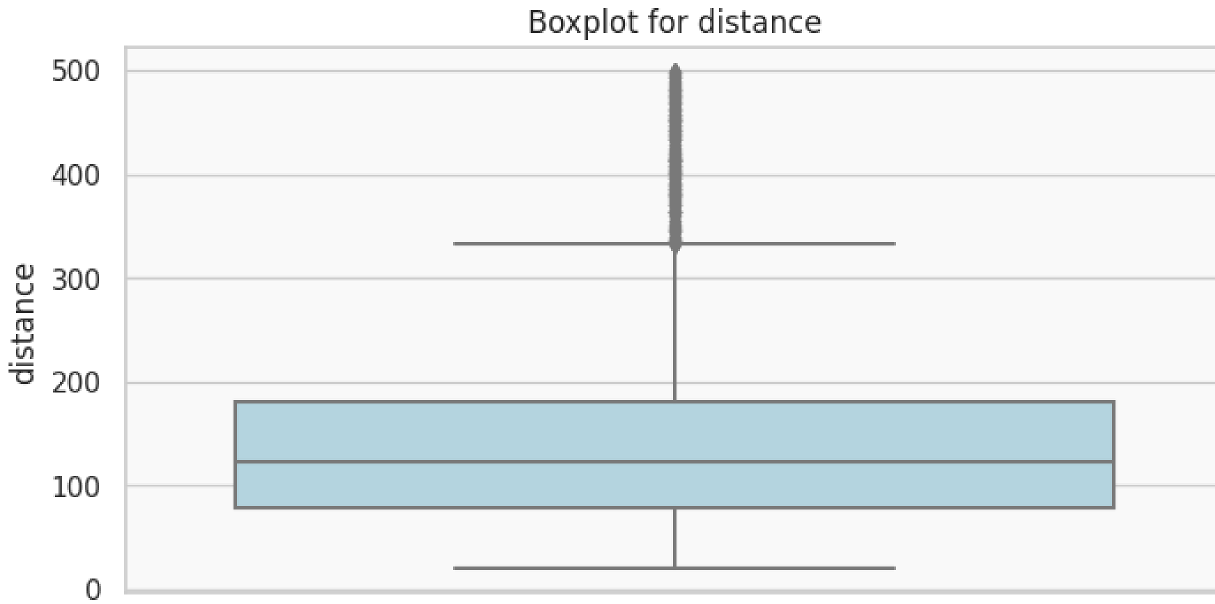

Frequency Distribution of fuel_type
Count
0
200
400
600
800
HFO
Diesel
fuel_type

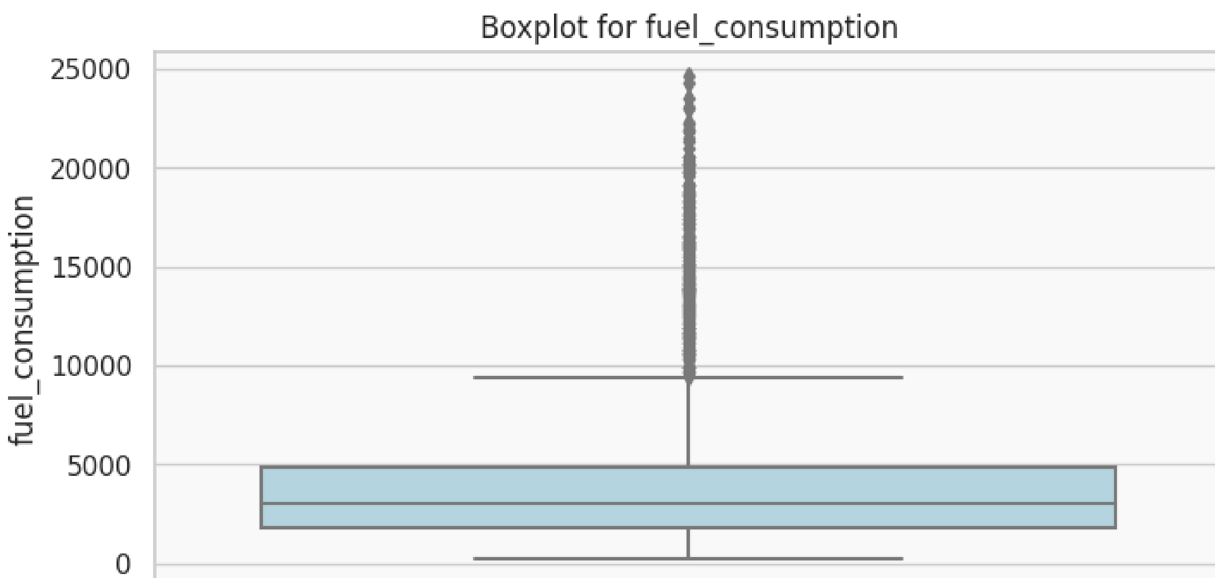


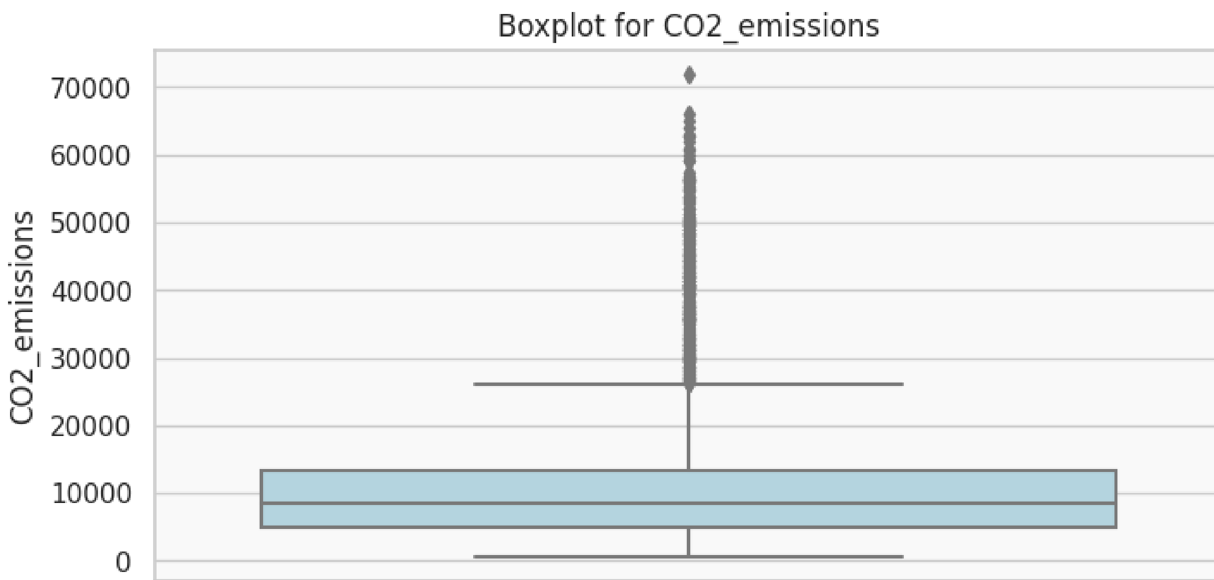

```
# 7. Pairplot for Numerical Relationships
sns.pairplot(data[numerical_columns], diag_kind="kde")
plt.suptitle("Pairplot of Numerical Features", y=1.02)
plt.show()
```

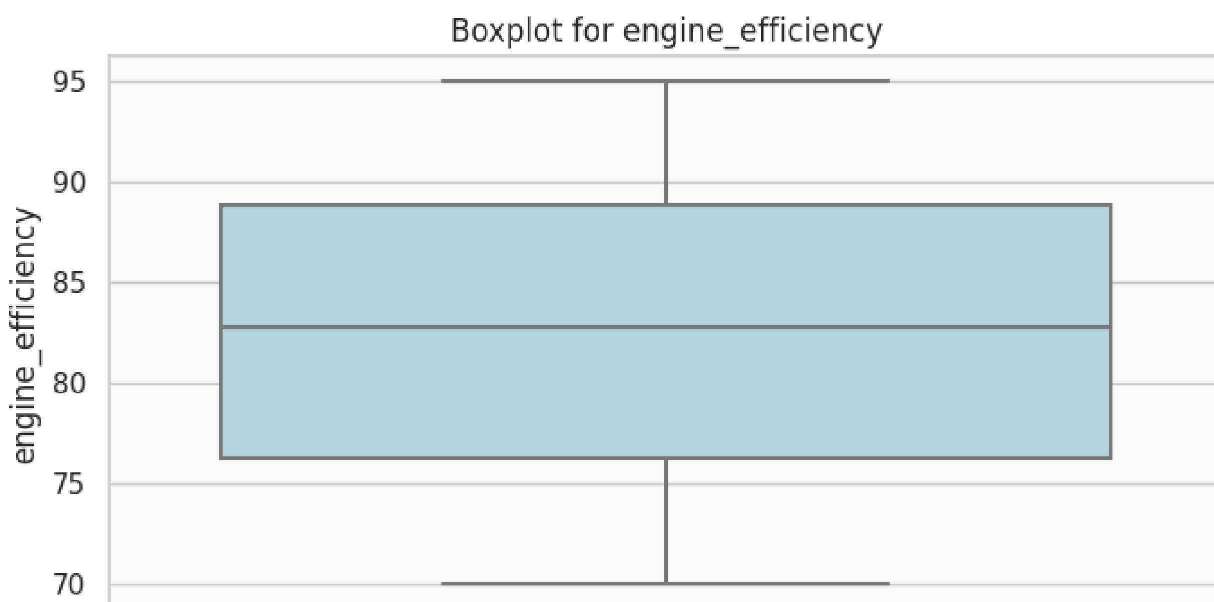



```python
# 8. Boxplots to Check for Outliers in Numerical Features
for col in numerical_columns:
    plt.figure(figsize=(8, 4))
    sns.boxplot(y=data[col], color="lightblue")
    plt.title(f"Boxplot for {col}")
    plt.ylabel(col)
    plt.show()
```


Boxplot for distance
distance
500
400
300
200
100
0


Boxplot for fuel_consumption
fuel_consumption
25000
20000
15000
10000
5000
0






```
# 9. Column-Wise Analysis
print("\n📊 **Column-Wise Analysis:**\n")
for col in data.columns:
    print(f"\n📌 **{col}**")
    print(f" - Data Type: {data[col].dtype}")
    print(f" - Unique Values: {data[col].nunique()}")
    print(f" - Sample Values: {data[col].unique()[:5]}")
    print(f" - Null Values: {data[col].isnull().sum()}")
    if data[col].dtype in ['int64', 'float64']:
        print(f" - Mean: {data[col].mean():.2f}, Std Dev:
{data[col].std():.2f}, Min: {data[col].min()}, Max:
{data[col].max()}")
```

```
□ **Column-Wise Analysis:**


□ **ship_id**
 - Data Type: object
 - Unique Values: 120
 - Sample Values: ['NG001' 'NG002' 'NG003' 'NG004' 'NG005']
 - Null Values: 0

□ **ship_type**
 - Data Type: object
 - Unique Values: 4
 - Sample Values: ['Oil Service Boat' 'Fishing Trawler' 'Surfer Boat'
'Tanker Ship']
 - Null Values: 0

□ **route_id**
 - Data Type: object
 - Unique Values: 4
 - Sample Values: ['Warri-Bonny' 'Port Harcourt-Lagos' 'Lagos-Apapa'
'Escravos-Lagos']
 - Null Values: 0

□ **month**
 - Data Type: object
 - Unique Values: 12
 - Sample Values: ['January' 'February' 'March' 'April' 'May']
 - Null Values: 0

□ **distance**
 - Data Type: float64
 - Unique Values: 1398
 - Sample Values: [132.26 128.52  67.3   71.68 134.32]
 - Null Values: 0
 - Mean: 151.75, Std Dev: 108.47, Min: 20.08, Max: 498.55

□ **fuel_type**
 - Data Type: object
 - Unique Values: 2
 - Sample Values: ['HFO' 'Diesel']
 - Null Values: 0

□ **fuel_consumption**
 - Data Type: float64
 - Unique Values: 1439
 - Sample Values: [3779.77 4461.44 1867.73 2393.51 4267.19]
 - Null Values: 0
 - Mean: 4844.25, Std Dev: 4892.35, Min: 237.88, Max: 24648.52

□ **CO2_emissions**
```

```
 - Data Type: float64
 - Unique Values: 1440
 - Sample Values: [10625.76 12779.73  5353.01  6506.52 11617.03]
 - Null Values: 0
 - Mean: 13365.45, Std Dev: 13567.65, Min: 615.68, Max: 71871.21

□ **weather_conditions**
 - Data Type: object
 - Unique Values: 3
 - Sample Values: ['Stormy' 'Moderate' 'Calm']
 - Null Values: 0

□ **engine_efficiency**
 - Data Type: float64
 - Unique Values: 1089
 - Sample Values: [92.14 92.98 87.61 87.42 85.61]
 - Null Values: 0
 - Mean: 82.58, Std Dev: 7.16, Min: 70.01, Max: 94.98
```

```
# 10. Row-Wise Analysis
print("\n□ **Row-Wise Analysis:**\n")
# Display a sample of rows with maximum and minimum values for key
numerical columns
key_columns = ['distance', 'fuel_consumption', 'CO2_emissions',
'engine_efficiency']

for col in key_columns:
    print(f"\n□ Rows with Maximum and Minimum Values for {col}:")
    max_row = data.loc[data[col].idxmax()]  # Row with max value
    min_row = data.loc[data[col].idxmin()]  # Row with min value
    print(f" - Row with MAX {col} (Value: {data[col].max()}):\
n{max_row}\n")
    print(f" - Row with MIN {col} (Value: {data[col].min()}):\
n{min_row}\n")
```

```

□ **Row-Wise Analysis:**


□ Rows with Maximum and Minimum Values for distance:
 - Row with MAX distance (Value: 498.55):
ship_id                       NG067
ship_type               Tanker Ship
route_id                Lagos-Apapa
month                      February
distance                     498.55
fuel_type                    Diesel
fuel_consumption           22973.21
CO2_emissions              62936.17
weather_conditions           Stormy
```

```
engine_efficiency           70.49
Name: 793, dtype: object

 - Row with MIN distance (Value: 20.08):
ship_id                             NG087
ship_type            Oil Service Boat
route_id                  Lagos-Apapa
month                        February
distance                        20.08
fuel_type                      Diesel
fuel_consumption               652.41
CO2_emissions                 1936.77
weather_conditions           Moderate
engine_efficiency               87.22
Name: 1033, dtype: object


□ Rows with Maximum and Minimum Values for fuel_consumption:
 - Row with MAX fuel_consumption (Value: 24648.52):
ship_id                        NG008
ship_type            Tanker Ship
route_id             Lagos-Apapa
month                   December
distance                  497.16
fuel_type                 Diesel
fuel_consumption        24648.52
CO2_emissions           62802.03
weather_conditions          Calm
engine_efficiency          72.14
Name: 95, dtype: object

 - Row with MIN fuel_consumption (Value: 237.88):
ship_id                           NG096
ship_type                   Surfer Boat
route_id                Escravos-Lagos
month                              June
distance                          21.42
fuel_type                        Diesel
fuel_consumption                 237.88
CO2_emissions                    615.68
weather_conditions             Moderate
engine_efficiency                 71.97
Name: 1145, dtype: object


□ Rows with Maximum and Minimum Values for CO2_emissions:
 - Row with MAX CO2_emissions (Value: 71871.21):
ship_id                        NG012
ship_type            Tanker Ship
route_id             Warri-Bonny
```

```
month                              March
distance                          490.16
fuel_type                         Diesel
fuel_consumption                 24321.4
CO2_emissions                   71871.21
weather_conditions                  Calm
engine_efficiency                  93.41
Name: 134, dtype: object

 - Row with MIN CO2_emissions (Value: 615.68):
ship_id                             NG096
ship_type                     Surfer Boat
route_id                   Escravos-Lagos
month                                June
distance                            21.42
fuel_type                          Diesel
fuel_consumption                   237.88
CO2_emissions                      615.68
weather_conditions               Moderate
engine_efficiency                   71.97
Name: 1145, dtype: object


□ Rows with Maximum and Minimum Values for engine_efficiency:
 - Row with MAX engine_efficiency (Value: 94.98):
ship_id                               NG086
ship_type                  Oil Service Boat
route_id                     Escravos-Lagos
month                               October
distance                              37.71
fuel_type                            Diesel
fuel_consumption                    1167.85
CO2_emissions                       2928.03
weather_conditions                   Stormy
engine_efficiency                     94.98
Name: 1029, dtype: object

 - Row with MIN engine_efficiency (Value: 70.01):
ship_id                              NG035
ship_type                  Fishing Trawler
route_id                    Escravos-Lagos
month                             February
distance                            124.67
fuel_type                              HFO
fuel_consumption                   2816.44
CO2_emissions                      8150.61
weather_conditions                Moderate
engine_efficiency                    70.01
Name: 409, dtype: object
```

```python
# 11. Duplicate Rows Check
print("\n□ **Duplicate Rows Check:**")
duplicate_count = data.duplicated().sum()
print(f" - Total Duplicate Rows: {duplicate_count}")
if duplicate_count > 0:
    print(" - Displaying Duplicate Rows:\n")
    print(data[data.duplicated()].head())
```

```
□ **Duplicate Rows Check:**
 - Total Duplicate Rows: 0
```

```python
# 12. Constant Value Columns Check
print("\n□ **Constant Value Columns Check:**")
constant_columns = [col for col in data.columns if data[col].nunique()
== 1]
if constant_columns:
    print(f" - Columns with Constant Values: {constant_columns}")
else:
    print(" - No columns with constant values found.")
```

```
□ **Constant Value Columns Check:**
 - No columns with constant values found.
```

```python
# 13. Row Integrity: Check for Rows with Extreme Low/High Values in
Numerical Columns
print("\n□ **Row Integrity Check for Extreme Values:**")
thresh_low = 0.05  # 5% threshold for low values
thresh_high = 0.95  # 95% threshold for high values

for col in key_columns:
    low_value_threshold = data[col].quantile(thresh_low)
    high_value_threshold = data[col].quantile(thresh_high)
    print(f"\nColumn: {col}")
    print(f" - Rows below {thresh_low*100}% threshold
({low_value_threshold}): {len(data[data[col] <
low_value_threshold])}")
    print(f" - Rows above {thresh_high*100}% threshold
({high_value_threshold}): {len(data[data[col] >
high_value_threshold])}")
```

```
□ **Row Integrity Check for Extreme Values:**

Column: distance
 - Rows below 5.0% threshold (34.579): 72
 - Rows above 95.0% threshold (414.2545): 72

Column: fuel_consumption
 - Rows below 5.0% threshold (734.3405): 72
```

```
 - Rows above 95.0% threshold (16501.289499999995): 72

Column: CO2_emissions
 - Rows below 5.0% threshold (2061.8855): 72
 - Rows above 95.0% threshold (45649.300999999985): 72

Column: engine_efficiency
 - Rows below 5.0% threshold (71.33): 70
 - Rows above 95.0% threshold (93.64): 71
```

```
# 14. Numerical Consistency Across Key Columns
print("\n□ **Numerical Consistency Check:**")
logical_columns = ['distance', 'fuel_consumption', 'CO2_emissions']

# Check if fuel consumption and CO2 emissions scale logically with
distance
inconsistent_rows = data[(data['fuel_consumption'] / data['distance'])
< 0.1]
print(f" - Rows where fuel consumption per unit distance is unusually
low (<0.1): {len(inconsistent_rows)}")
if not inconsistent_rows.empty:
    print(inconsistent_rows.head())
```

```
□ **Numerical Consistency Check:**
 - Rows where fuel consumption per unit distance is unusually low
(<0.1): 0
```

```
# CO2 emissions relative to fuel consumption
inconsistent_emission = data[(data['CO2_emissions'] /
data['fuel_consumption']) < 2]
print(f" - Rows where CO2 emissions per fuel unit are unusually low
(<2): {len(inconsistent_emission)}")
if not inconsistent_emission.empty:
    print(inconsistent_emission.head())
```

```
 - Rows where CO2 emissions per fuel unit are unusually low (<2): 0
```

```
# 15. Logical Range Validation for Engine Efficiency
print("\n⚙ **Engine Efficiency Logical Range Check:**")
invalid_efficiency = data[(data['engine_efficiency'] < 0) |
(data['engine_efficiency'] > 100)]
if invalid_efficiency.empty:
    print(" - All engine efficiency values are within the valid range
(0-100).")
else:
    print(f" - Rows with invalid engine efficiency values:
{len(invalid_efficiency)}")
    print(invalid_efficiency)
```

```
⚙ **Engine Efficiency Logical Range Check:**
 - All engine efficiency values are within the valid range (0-100).

# 16. Data Duplication Across Specific Columns
print("\n□ **Duplication Check in Key Columns:**")
duplicate_columns = ['ship_id', 'month', 'route_id']
duplicates_in_columns = data.duplicated(subset=duplicate_columns)
print(f" - Total duplicate rows based on {duplicate_columns}:
{duplicates_in_columns.sum()}")
if duplicates_in_columns.any():
    print(" - Displaying first few duplicates:")
    print(data[duplicates_in_columns].head())


□ **Duplication Check in Key Columns:**
 - Total duplicate rows based on ['ship_id', 'month', 'route_id']: 0

# 17. Cross-Validation of Categorical Relationships
print("\n **Cross-Validation of Categorical Columns:**")
unique_combinations = data.groupby(['ship_type', 'route_id']).size()
print(" - Unique ship_type and route_id combinations:\n")
print(unique_combinations.head())


 **Cross-Validation of Categorical Columns:**
 - Unique ship_type and route_id combinations:

ship_type         route_id
Fishing Trawler   Escravos-Lagos           79
                  Lagos-Apapa              74
                  Port Harcourt-Lagos      76
                  Warri-Bonny              71
Oil Service Boat  Escravos-Lagos          107
dtype: int64

# 18. Temporal Analysis of Data Integrity
print("\n□ **Temporal Integrity Check:**")
print(" - Unique Months in Dataset:", data['month'].unique())
month_out_of_range = data[~data['month'].isin(range(1, 13))]
if month_out_of_range.empty:
    print(" - All month values are valid (1-12).")
else:
    print(" - Rows with invalid month values:")
    print(month_out_of_range)


□ **Temporal Integrity Check:**
 - Unique Months in Dataset: ['January' 'February' 'March' 'April'
'May' 'June' 'July' 'August'
 'September' 'October' 'November' 'December']
```

```
 - Rows with invalid month values:
     ship_id         ship_type             route_id      month
distance  \
0      NG001  Oil Service Boat          Warri-Bonny    January
132.26
1      NG001  Oil Service Boat  Port Harcourt-Lagos   February
128.52
2      NG001  Oil Service Boat  Port Harcourt-Lagos      March
67.30
3      NG001  Oil Service Boat  Port Harcourt-Lagos      April
71.68
4      NG001  Oil Service Boat          Lagos-Apapa        May
134.32
...      ...               ...                  ...        ...        .
..
1435   NG120   Fishing Trawler  Port Harcourt-Lagos     August
63.84
1436   NG120   Fishing Trawler          Lagos-Apapa  September
61.43
1437   NG120   Fishing Trawler  Port Harcourt-Lagos    October
193.09
1438   NG120   Fishing Trawler          Lagos-Apapa   November
166.50
1439   NG120   Fishing Trawler          Warri-Bonny   December
127.66

     fuel_type  fuel_consumption  CO2_emissions weather_conditions  \
0          HFO           3779.77       10625.76             Stormy
1          HFO           4461.44       12779.73           Moderate
2          HFO           1867.73        5353.01               Calm
3       Diesel           2393.51        6506.52             Stormy
4          HFO           4267.19       11617.03               Calm
...        ...               ...            ...                ...
1435    Diesel           1633.85        4852.28             Stormy
1436       HFO           1263.48        3571.13               Calm
1437       HFO           4661.63       12267.13             Stormy
1438    Diesel           4298.00       12297.71           Moderate
1439    Diesel           3549.91       10641.90           Moderate

      engine_efficiency
0                 92.14
1                 92.98
2                 87.61
3                 87.42
4                 85.61
...                 ...
1435              75.88
1436              78.00
1437              79.67
```

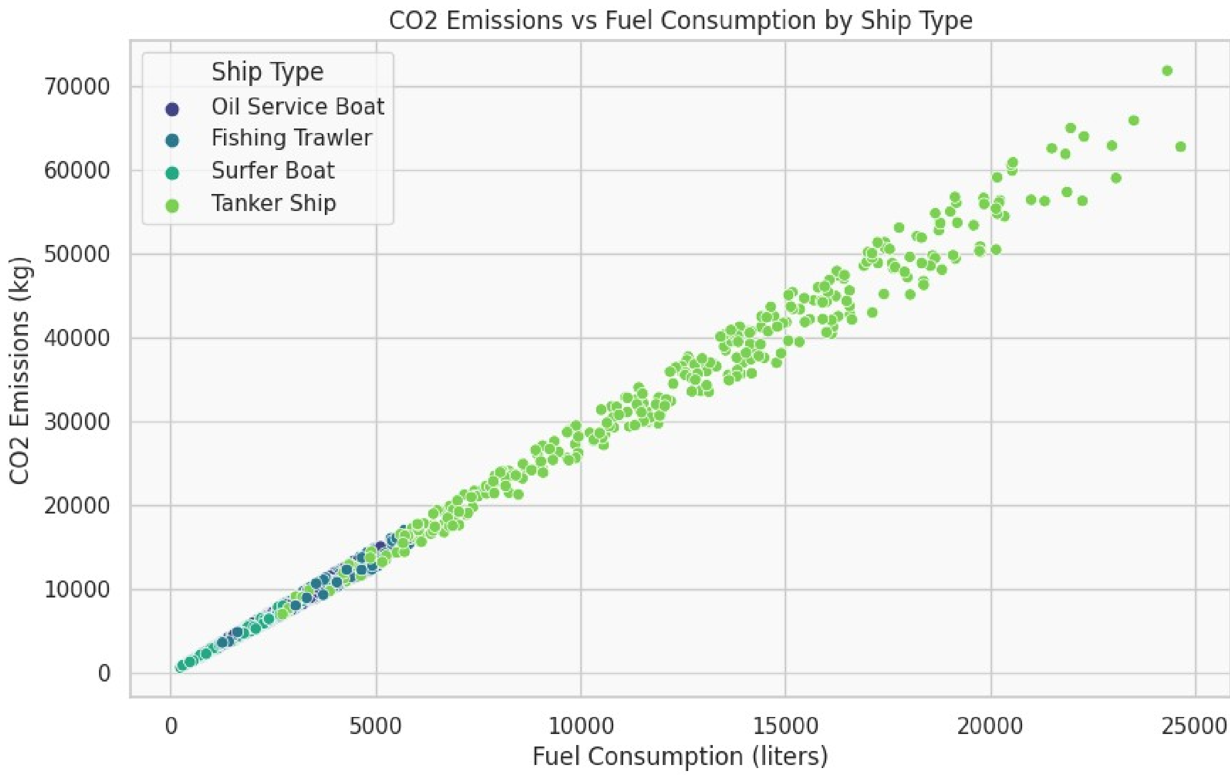
```
1438                92.87
1439                90.82

[1440 rows x 10 columns]
```

```python
# 19. Aggregated Summary for All Checks
print("\n□ **Summary of Integrity and Logical Checks:**")
print(f" - Total rows with logical inconsistencies in fuel consumption: {len(inconsistent_rows)}")
print(f" - Total rows with logical inconsistencies in CO2 emissions: {len(inconsistent_emission)}")
print(f" - Total rows with invalid engine efficiency: {len(invalid_efficiency)}")
print(f" - Total duplicate rows based on key columns: {duplicates_in_columns.sum()}")
print(f" - Total rows with invalid month values: {len(month_out_of_range)}")
```

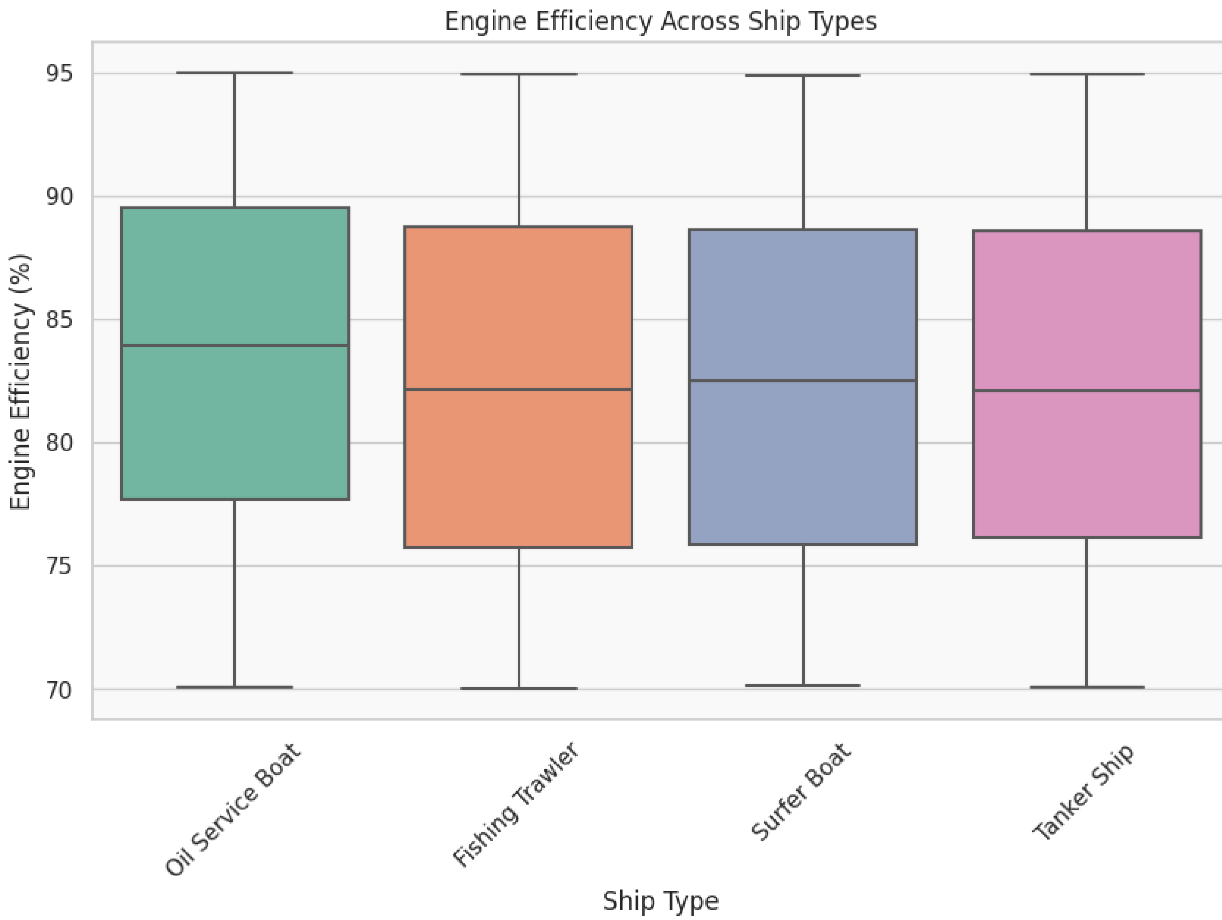
```
□ **Summary of Integrity and Logical Checks:**
 - Total rows with logical inconsistencies in fuel consumption: 0
 - Total rows with logical inconsistencies in CO2 emissions: 0
 - Total rows with invalid engine efficiency: 0
 - Total duplicate rows based on key columns: 0
 - Total rows with invalid month values: 1440
```

```python
# 20. Fuel Consumption vs Distance (Scatter Plot)
plt.figure(figsize=(10, 6))
sns.scatterplot(x=data['distance'], y=data['fuel_consumption'], hue=data['ship_type'], palette="coolwarm")
plt.title("Fuel Consumption vs Distance by Ship Type")
plt.xlabel("Distance (km)")
plt.ylabel("Fuel Consumption (liters)")
plt.legend(title="Ship Type")
plt.show()
```

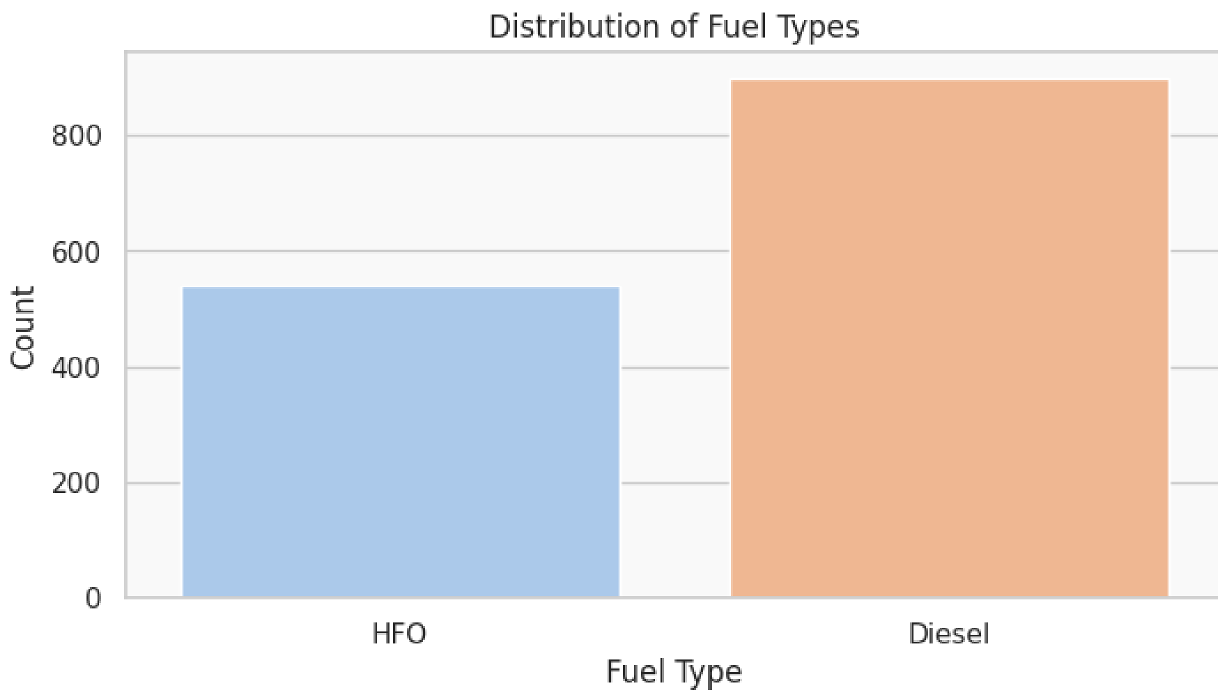

```python
# 21. CO2 Emissions vs Fuel Consumption (Scatter Plot)
plt.figure(figsize=(10, 6))
sns.scatterplot(x=data['fuel_consumption'], y=data['CO2_emissions'],
hue=data['ship_type'], palette="viridis")
plt.title("CO2 Emissions vs Fuel Consumption by Ship Type")
plt.xlabel("Fuel Consumption (liters)")
plt.ylabel("CO2 Emissions (kg)")
plt.legend(title="Ship Type")
plt.show()
```



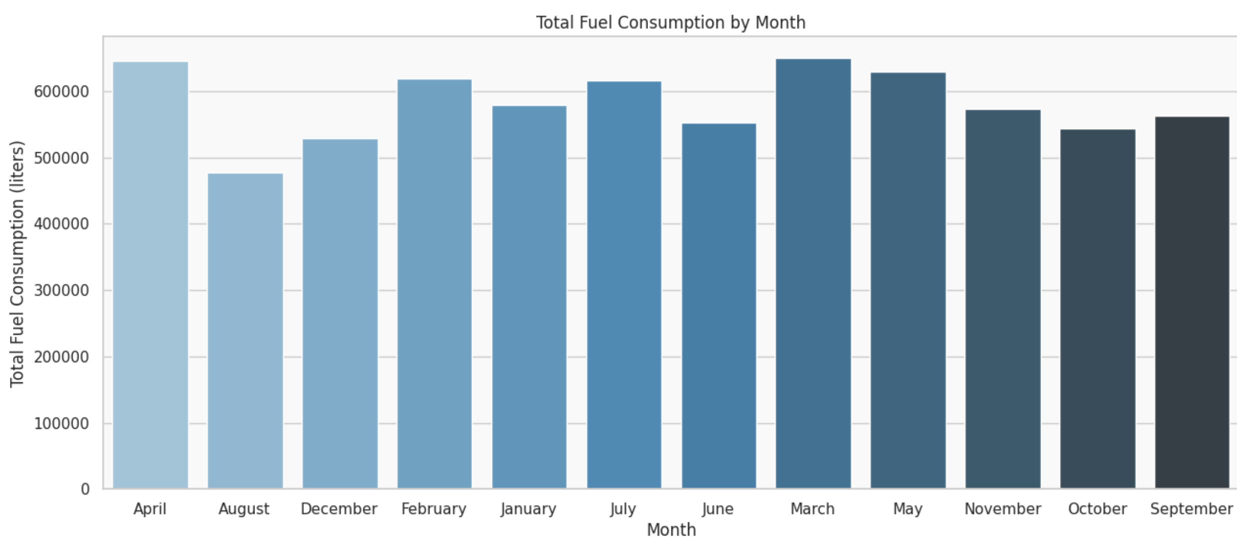

```python
# 22. Engine Efficiency Across Ship Types (Boxplot)
plt.figure(figsize=(10, 6))
sns.boxplot(x=data['ship_type'], y=data['engine_efficiency'],
palette="Set2")
plt.title("Engine Efficiency Across Ship Types")
plt.xlabel("Ship Type")
plt.ylabel("Engine Efficiency (%)")
plt.xticks(rotation=45)
plt.show()
```



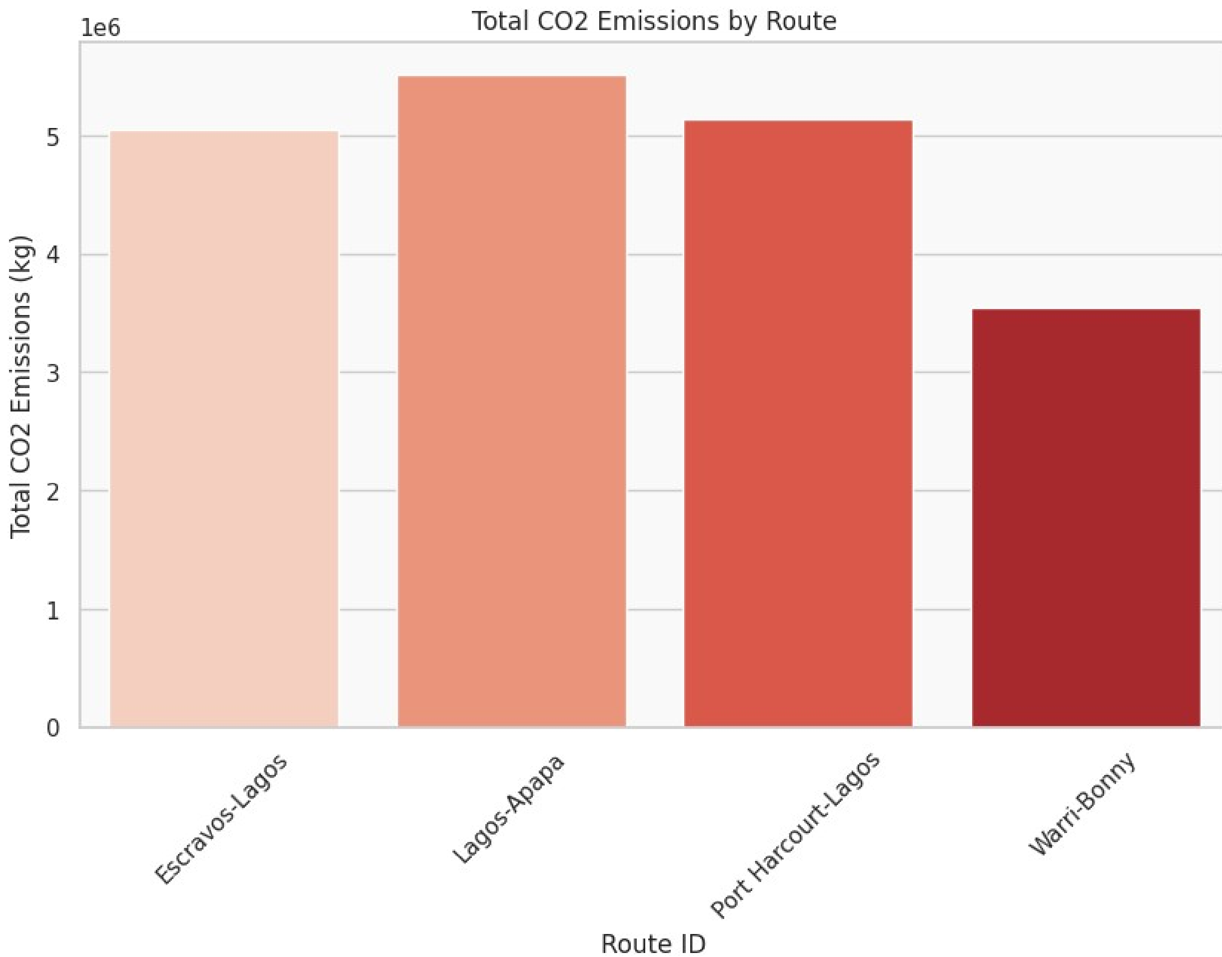

```
# 23. Fuel Type Distribution (Countplot)
plt.figure(figsize=(8, 4))
sns.countplot(x=data['fuel_type'], palette="pastel")
plt.title("Distribution of Fuel Types")
plt.xlabel("Fuel Type")
plt.ylabel("Count")
plt.show()
```



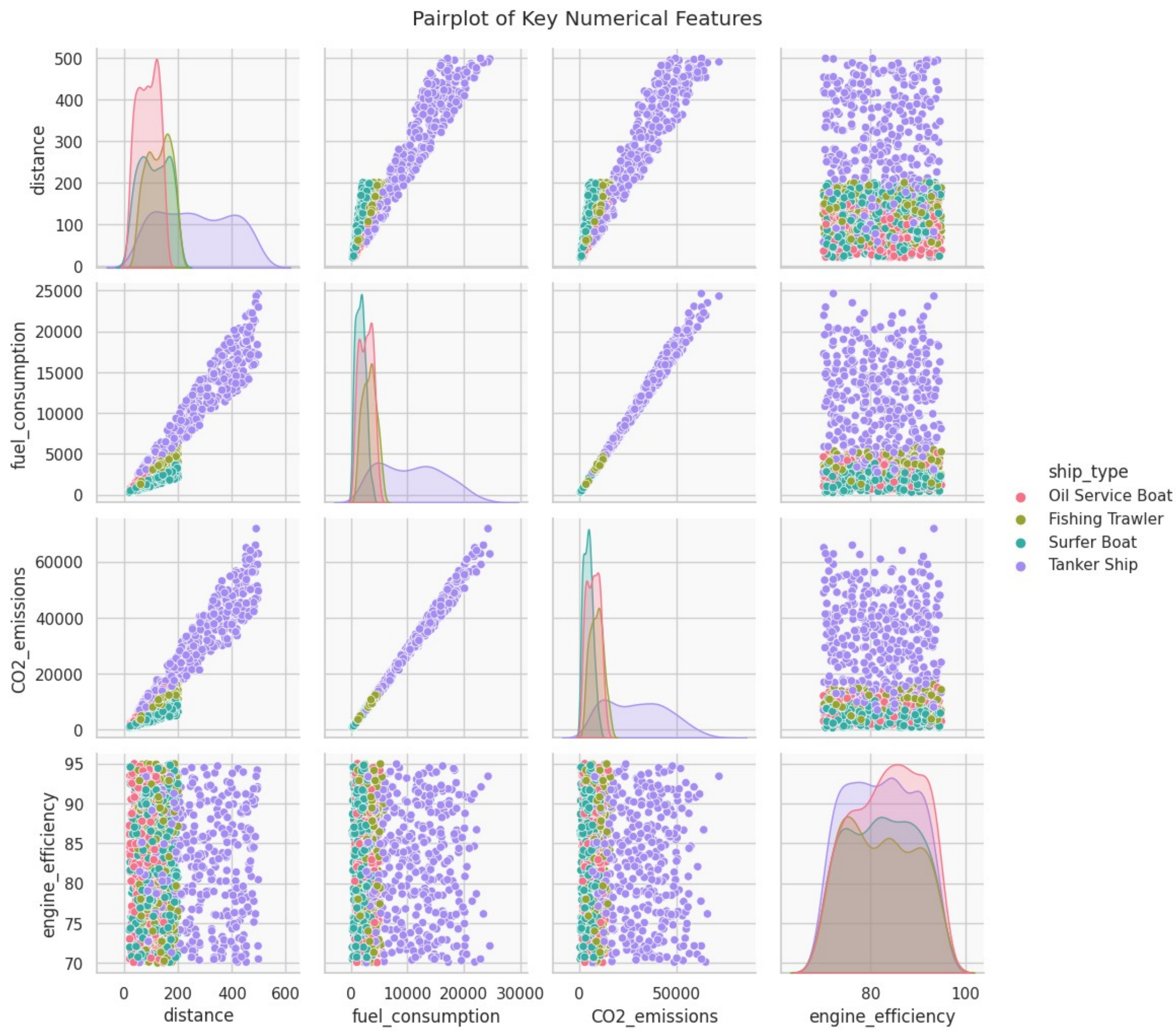

```
# 24. Monthly Fuel Consumption (Barplot)
monthly_fuel = data.groupby('month')
['fuel_consumption'].sum().reset_index()
plt.figure(figsize=(15, 6))
sns.barplot(x='month', y='fuel_consumption', data=monthly_fuel,
palette="Blues_d")
plt.title("Total Fuel Consumption by Month")
plt.xlabel("Month")
plt.ylabel("Total Fuel Consumption (liters)")
plt.show()
```



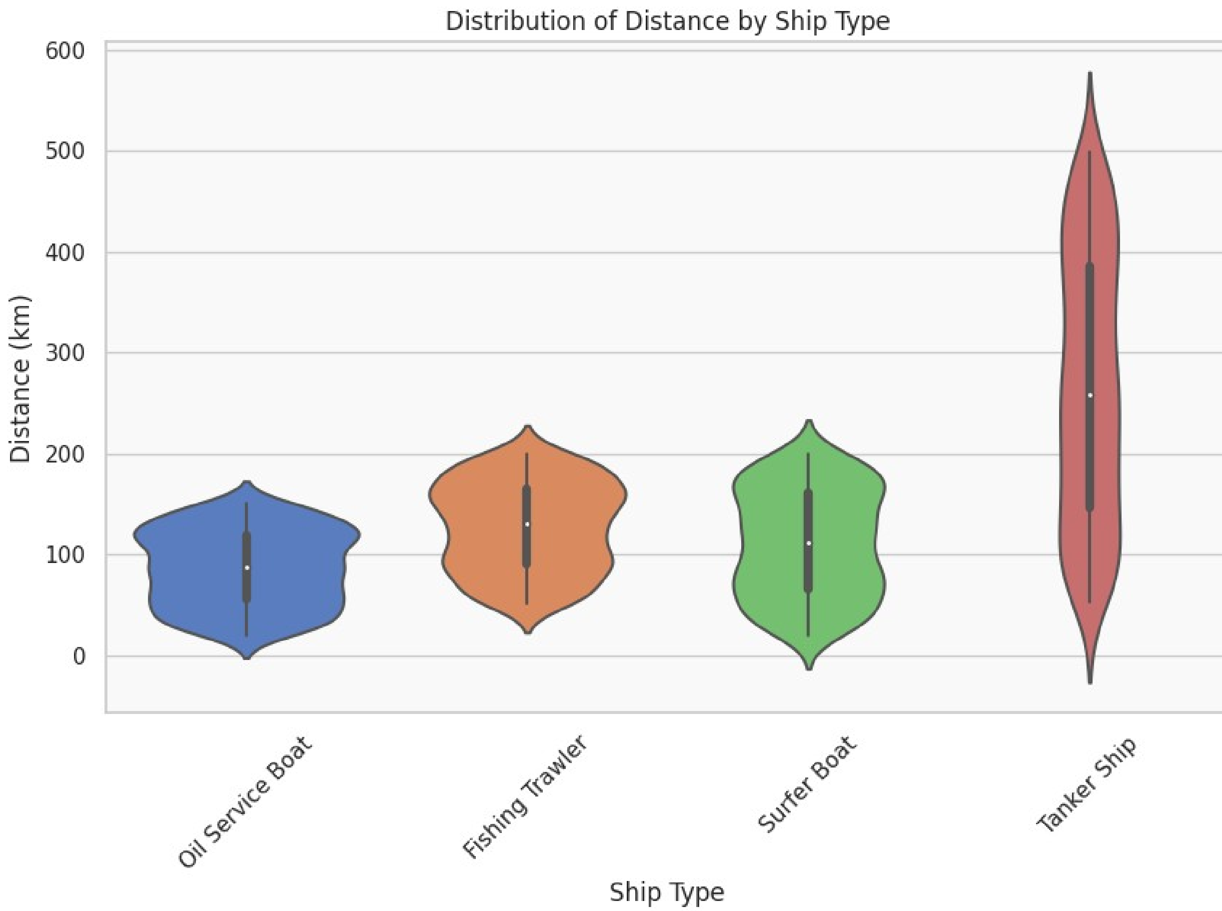
```
# 25. CO2 Emissions by Route (Barplot)
route_emissions = data.groupby('route_id')
['CO2_emissions'].sum().reset_index()
plt.figure(figsize=(10, 6))
sns.barplot(x='route_id', y='CO2_emissions', data=route_emissions,
palette="Reds")
plt.title("Total CO2 Emissions by Route")
plt.xlabel("Route ID")
plt.ylabel("Total CO2 Emissions (kg)")
plt.xticks(rotation=45)
plt.show()
```



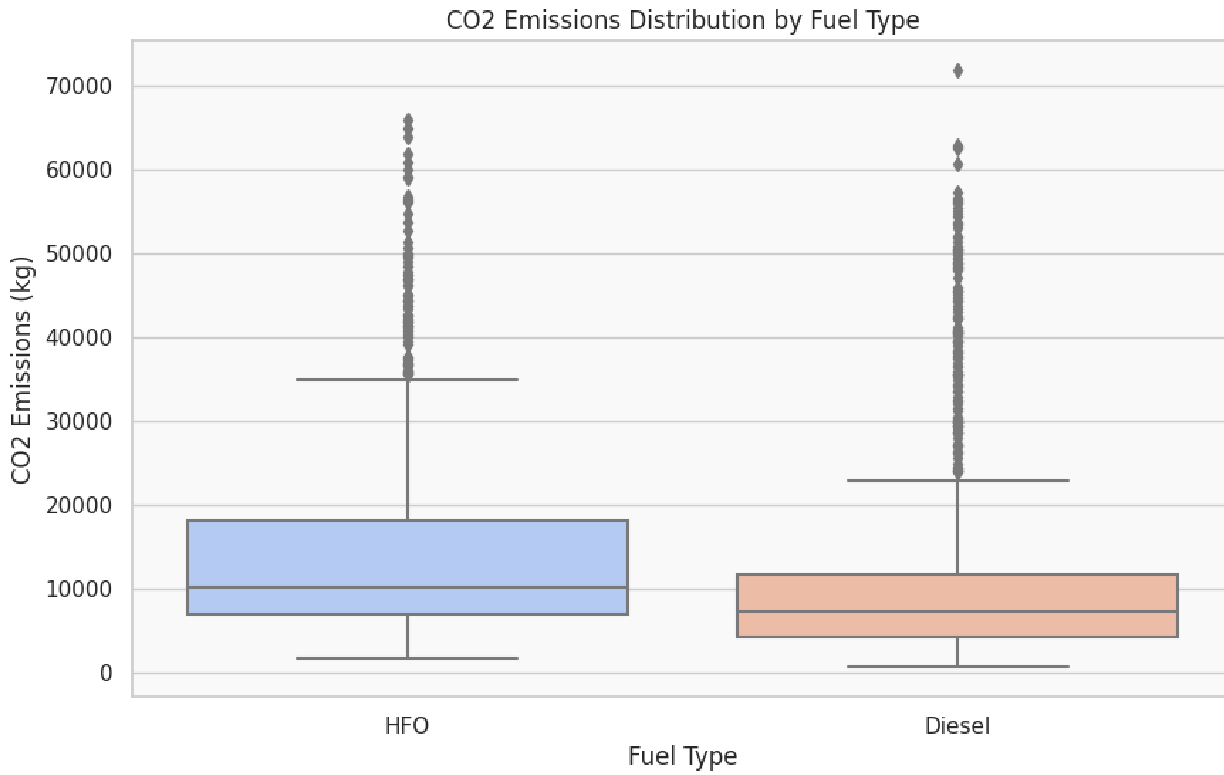

```
# 26. Pairplot for Numerical Relationships
sns.pairplot(data, hue='ship_type', vars=['distance',
'fuel_consumption', 'CO2_emissions', 'engine_efficiency'],
palette="husl")
plt.suptitle("Pairplot of Key Numerical Features", y=1.02)
plt.show()
```



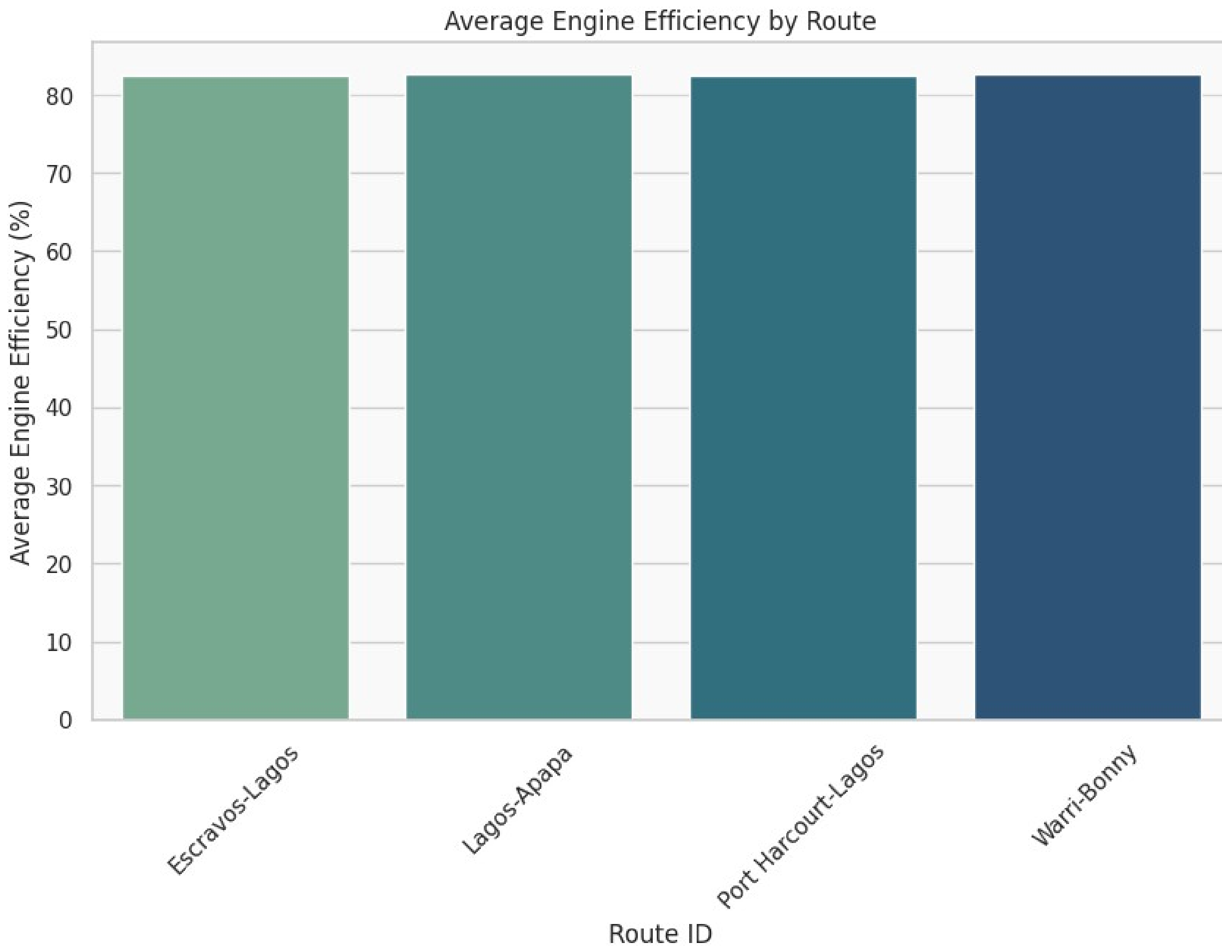

```python
# 27. Distribution of Distance by Ship Type (Violin Plot)
plt.figure(figsize=(10, 6))
sns.violinplot(x='ship_type', y='distance', data=data,
palette="muted")
plt.title("Distribution of Distance by Ship Type")
plt.xlabel("Ship Type")
plt.ylabel("Distance (km)")
plt.xticks(rotation=45)
plt.show()
```



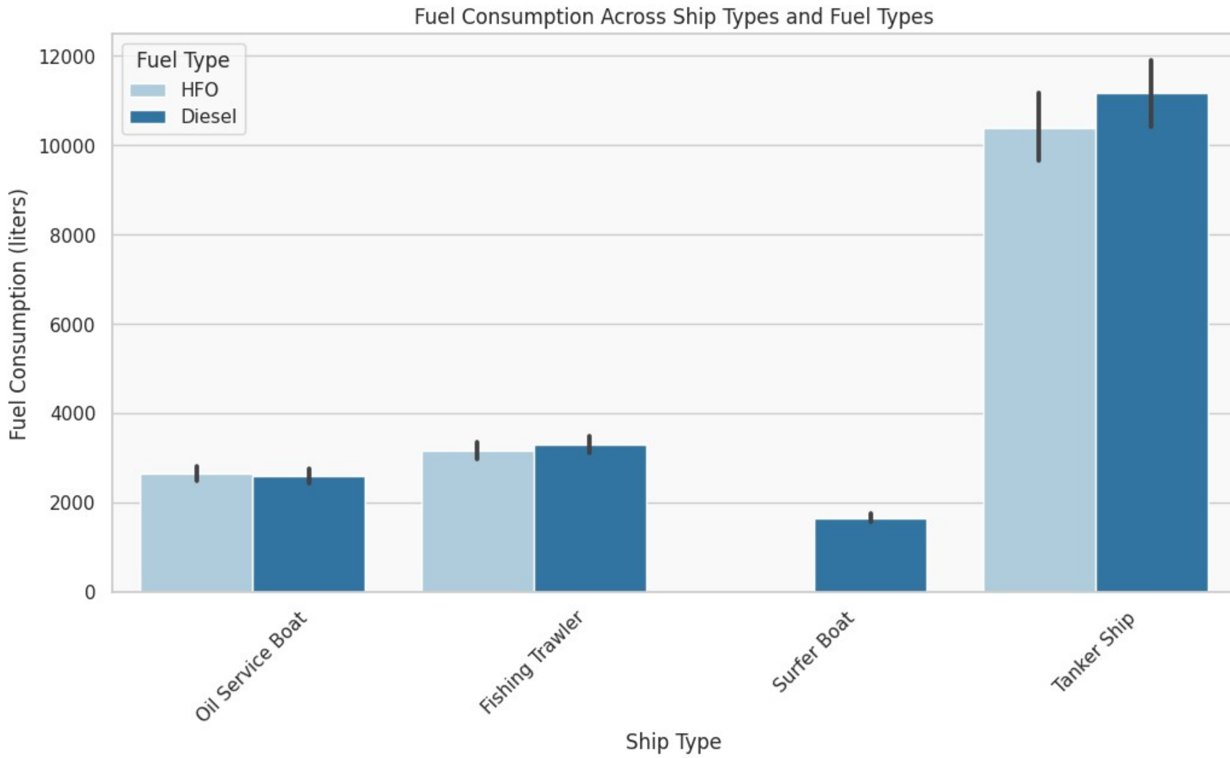

```
# 28. CO2 Emissions per Fuel Type (Box Plot)
plt.figure(figsize=(10, 6))
sns.boxplot(x='fuel_type', y='CO2_emissions', data=data,
palette="coolwarm")
plt.title("CO2 Emissions Distribution by Fuel Type")
plt.xlabel("Fuel Type")
plt.ylabel("CO2 Emissions (kg)")
plt.show()
```



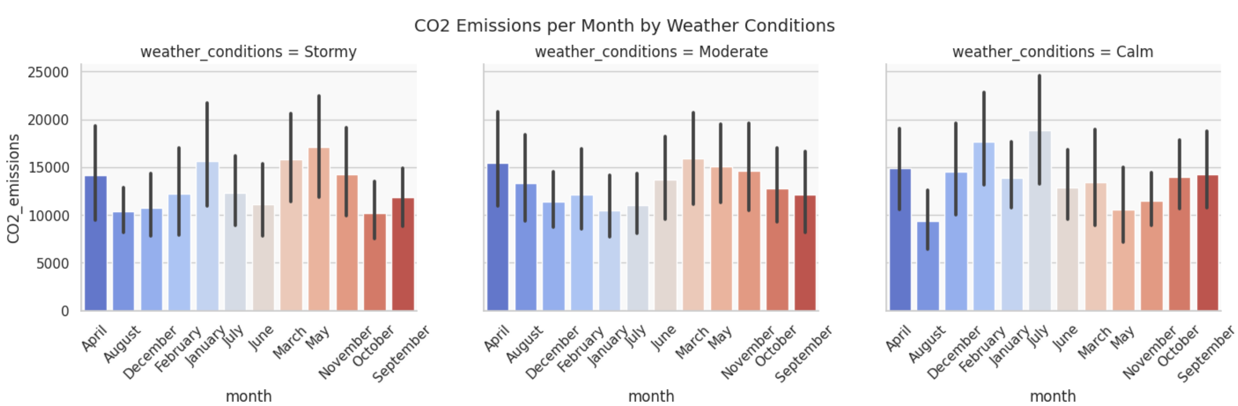

```
# 29. Average Engine Efficiency by Route (Bar Plot)
avg_efficiency = data.groupby('route_id')
['engine_efficiency'].mean().reset_index()
plt.figure(figsize=(10, 6))
sns.barplot(x='route_id', y='engine_efficiency', data=avg_efficiency,
palette="crest")
plt.title("Average Engine Efficiency by Route")
plt.xlabel("Route ID")
plt.ylabel("Average Engine Efficiency (%)")
plt.xticks(rotation=45)
plt.show()
```



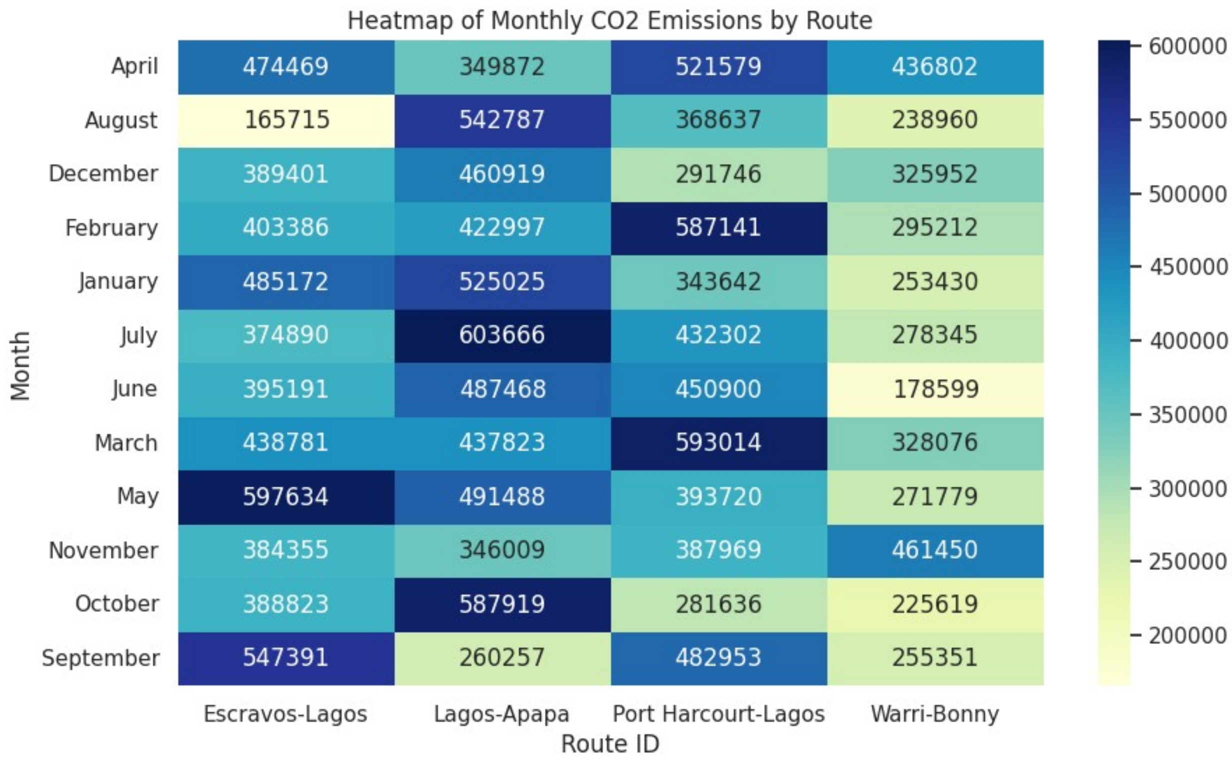

```
# 30. Fuel Consumption Across Ship Types and Fuel Types (Grouped Bar
Plot)
plt.figure(figsize=(12, 6))
sns.barplot(x='ship_type', y='fuel_consumption', hue='fuel_type',
data=data, palette="Paired")
plt.title("Fuel Consumption Across Ship Types and Fuel Types")
plt.xlabel("Ship Type")
plt.ylabel("Fuel Consumption (liters)")
plt.xticks(rotation=45)
plt.legend(title="Fuel Type")
plt.show()
```



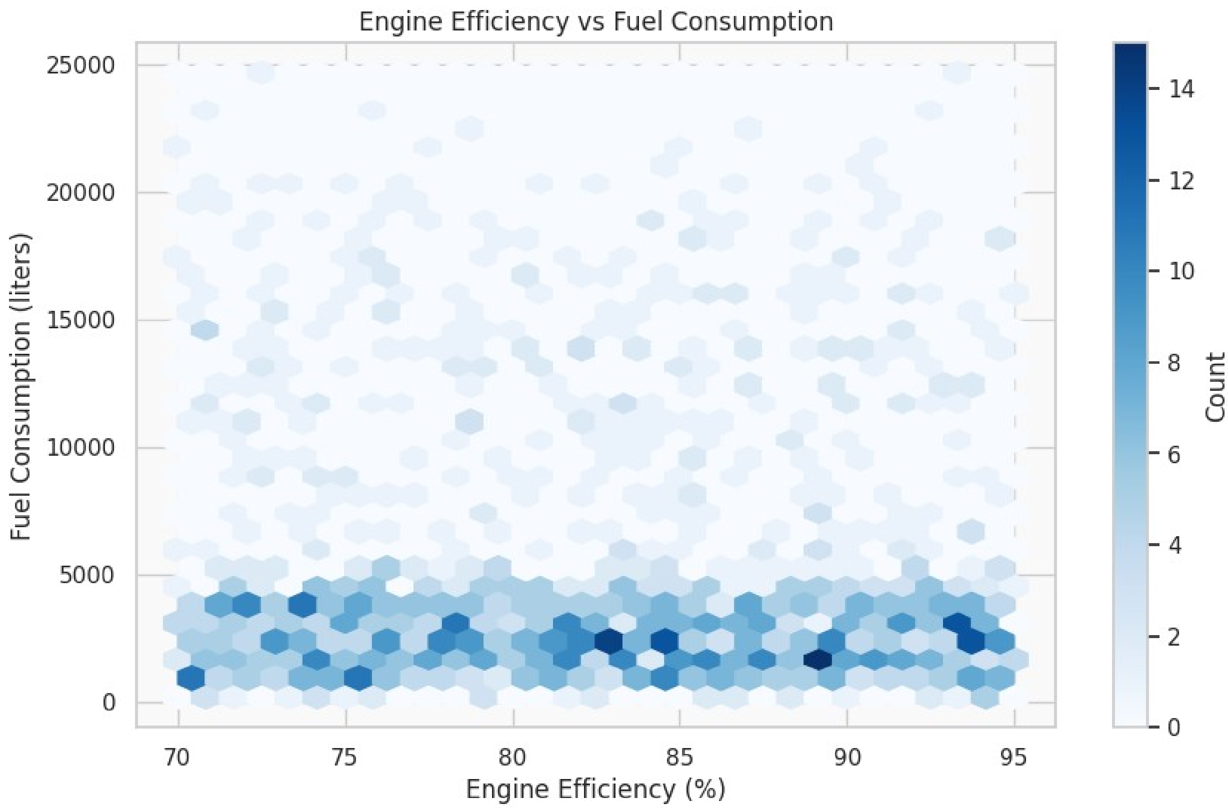

```python
# 31. CO2 Emissions per Month by Weather Conditions (Facet Grid)

# Adjusted and Cleaner Layout for FacetGrid
g = sns.FacetGrid(data, col="weather_conditions", height=4,
aspect=1.2)
g.map(sns.barplot, "month", "CO2_emissions",
order=sorted(data['month'].unique()), palette="coolwarm")
g.fig.subplots_adjust(top=0.85, bottom=0.15, hspace=0.3, wspace=0.2)
g.set_xticklabels(rotation=45)  # Rotate x-axis labels for better
readability
g.fig.suptitle("CO2 Emissions per Month by Weather Conditions",
fontsize=14)
plt.show()
```



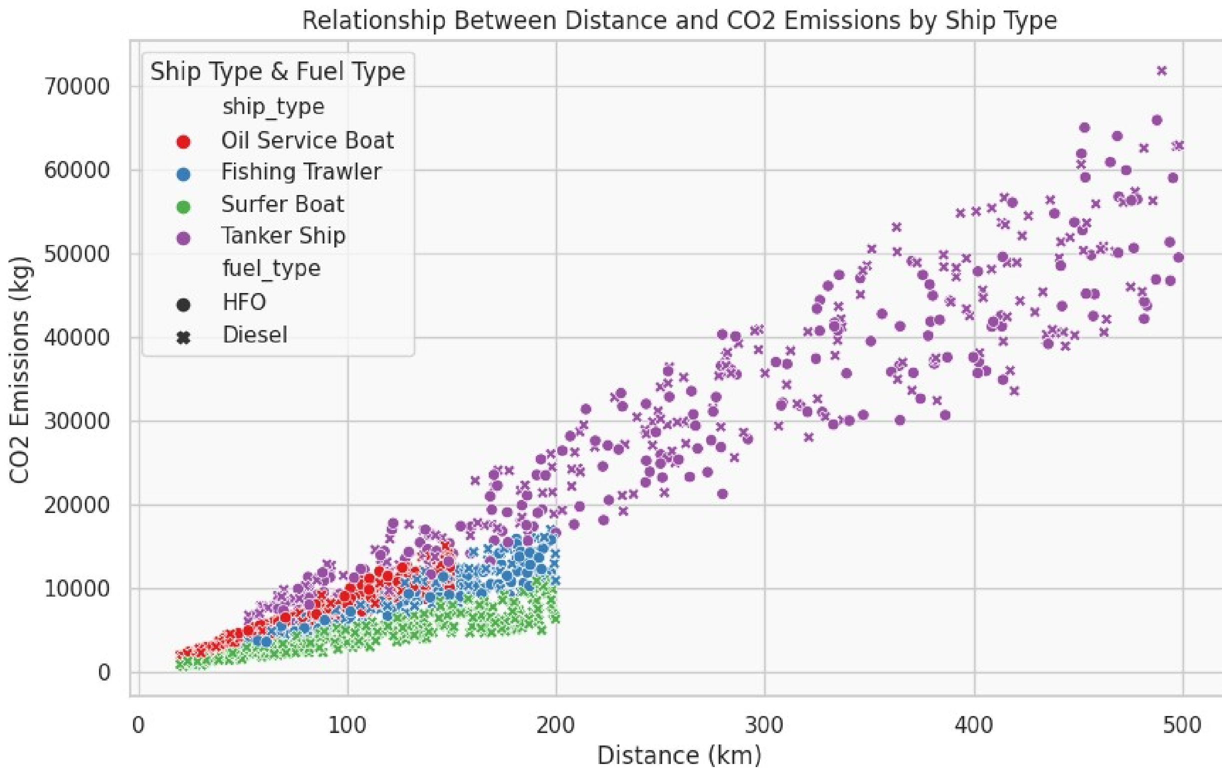
```
# 32. Heatmap of Monthly CO2 Emissions for Routes (Pivot Table)
monthly_route_emissions = data.pivot_table(values='CO2_emissions',
index='month', columns='route_id', aggfunc='sum')
plt.figure(figsize=(10, 6))
sns.heatmap(monthly_route_emissions, cmap="YlGnBu", annot=True,
fmt=".0f")
plt.title("Heatmap of Monthly CO2 Emissions by Route")
plt.xlabel("Route ID")
plt.ylabel("Month")
plt.show()
```



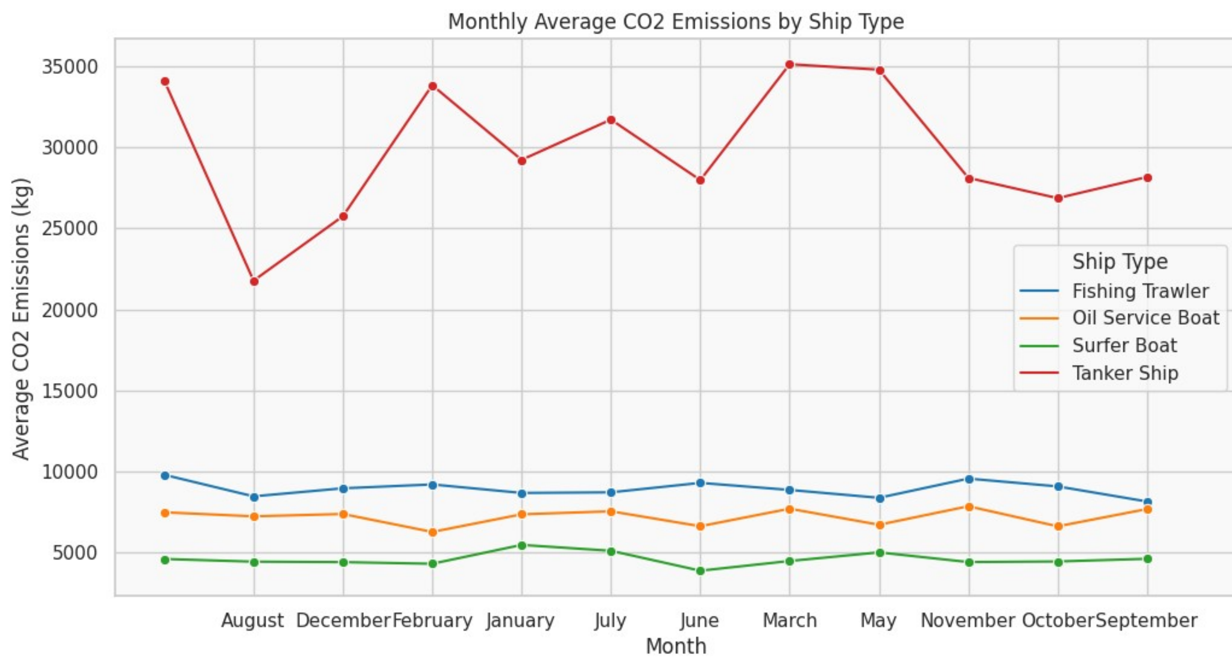

```
# 33. Engine Efficiency vs Fuel Consumption (Hexbin Plot)
plt.figure(figsize=(10, 6))
plt.hexbin(data['engine_efficiency'], data['fuel_consumption'],
gridsize=30, cmap='Blues')
plt.colorbar(label='Count')
plt.title("Engine Efficiency vs Fuel Consumption")
plt.xlabel("Engine Efficiency (%)")
plt.ylabel("Fuel Consumption (liters)")
plt.show()
```




```
# 34. Relationship Between Distance and CO2 Emissions by Ship Type
(Scatter Plot)
plt.figure(figsize=(10, 6))
sns.scatterplot(x='distance', y='CO2_emissions', hue='ship_type',
style='fuel_type', data=data, palette="Set1")
plt.title("Relationship Between Distance and CO2 Emissions by Ship
Type")
plt.xlabel("Distance (km)")
plt.ylabel("CO2 Emissions (kg)")
plt.legend(title="Ship Type & Fuel Type")
plt.show()
```




```python
# 35. Monthly Average CO2 Emissions by Ship Type (Line Plot)
monthly_avg_emissions = data.groupby(['month', 'ship_type'])
['CO2_emissions'].mean().reset_index()
plt.figure(figsize=(12, 6))
sns.lineplot(x='month', y='CO2_emissions', hue='ship_type',
data=monthly_avg_emissions, marker="o", palette="tab10")
plt.title("Monthly Average CO2 Emissions by Ship Type")
plt.xlabel("Month")
plt.ylabel("Average CO2 Emissions (kg)")
plt.xticks(range(1, 13))
plt.legend(title="Ship Type")
plt.show()
```



Step 5: Feature Engineering

> ⚡ **Note:** In this step, we transform and optimize the dataset by creating new features, handling existing ones, and preparing the data for modeling. Effective feature engineering is critical for improving model performance and capturing hidden relationships in the data.

Goals of Feature Engineering

1. Create new features to enhance predictive power.
2. Transform existing features to improve consistency and relevance.
3. Handle categorical and numerical variables effectively.
4. Remove redundant or highly correlated features.
5. Prepare the data for machine learning models.

Key Tasks

1. **Feature Creation:** Add new derived features based on existing data (e.g., emission efficiency).
2. **Feature Transformation:** Normalize or scale numerical features for consistency.
3. **Categorical Encoding:** Convert categorical variables into numerical format for modeling.
4. **Correlation Analysis:** Identify and remove highly correlated features to prevent multicollinearity.
5. **Outlier Treatment:** Apply techniques to handle extreme values in key columns.

 Let's proceed to transform and enrich the dataset for the next phase: **Model Building**!

```
# 1. Feature Creation: Emission Efficiency (CO2 Emissions per Fuel Consumption)
data['emission_efficiency'] = data['CO2_emissions'] / data['fuel_consumption']
print("\n Created 'emission_efficiency': CO2 emissions per unit of fuel consumption.")
print(data[['fuel_consumption', 'CO2_emissions', 'emission_efficiency']].head())
```

```
 Created 'emission_efficiency': CO2 emissions per unit of fuel consumption.
   fuel_consumption  CO2_emissions  emission_efficiency
0           3779.77       10625.76             2.811219
1           4461.44       12779.73             2.864485
2           1867.73        5353.01             2.866051
3           2393.51        6506.52             2.718401
4           4267.19       11617.03             2.722407
```

```
# 2. Feature Transformation: Scaling Numerical Features with Min-Max Scaling
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()
numerical_features = ['distance', 'fuel_consumption', 'CO2_emissions', 'engine_efficiency', 'emission_efficiency']
data_scaled = data.copy()
data_scaled[numerical_features] = scaler.fit_transform(data[numerical_features])
print("\n Scaled numerical features using Min-Max Scaling.")
print(data_scaled[numerical_features].head())
```

```
 Scaled numerical features using Min-Max Scaling.
   distance  fuel_consumption  CO2_emissions  engine_efficiency  \
0  0.234456          0.145096       0.140481           0.886264
1  0.226639          0.173021       0.170710           0.919904
2  0.098690          0.066768       0.066484           0.704846
3  0.107844          0.088307       0.082672           0.697237
4  0.238761          0.165064       0.154393           0.624750

   emission_efficiency
0             0.622384
1             0.728942
2             0.732074
3             0.436706
4             0.444721
```

```
# 3. Categorical Encoding: Label Encoding for Ship Type and Fuel Type
from sklearn.preprocessing import LabelEncoder

label_encoders = {}
categorical_features = ['ship_type', 'fuel_type']

for col in categorical_features:
    le = LabelEncoder()
    data_scaled[col] = le.fit_transform(data[col])
    label_encoders[col] = le
    print(f"\n□ Encoded '{col}' using Label Encoding.")
    print(f"Mapping for '{col}': {dict(enumerate(le.classes_))}")
```

```
□ Encoded 'ship_type' using Label Encoding.
Mapping for 'ship_type': {0: 'Fishing Trawler', 1: 'Oil Service Boat',
2: 'Surfer Boat', 3: 'Tanker Ship'}

□ Encoded 'fuel_type' using Label Encoding.
Mapping for 'fuel_type': {0: 'Diesel', 1: 'HFO'}
```

```
# 4. Outlier Treatment: Clipping Extreme Values in Numerical Features
for col in numerical_features:
    lower_bound = data_scaled[col].quantile(0.05)
    upper_bound = data_scaled[col].quantile(0.95)
    data_scaled[col] = data_scaled[col].clip(lower=lower_bound,
upper=upper_bound)
    print(f"\n□ Clipped outliers in '{col}' between {lower_bound:.2f}
and {upper_bound:.2f}.")
```

```
□ Clipped outliers in 'distance' between 0.03 and 0.82.

□ Clipped outliers in 'fuel_consumption' between 0.02 and 0.67.

□ Clipped outliers in 'CO2_emissions' between 0.02 and 0.63.

□ Clipped outliers in 'engine_efficiency' between 0.05 and 0.95.

□ Clipped outliers in 'emission_efficiency' between 0.06 and 0.95.
```

```
# 5. Correlation Check: Removing Highly Correlated Features

correlation_matrix = data_scaled[numerical_features].corr()
upper_triangle =
correlation_matrix.where(np.triu(np.ones(correlation_matrix.shape),
k=1).astype(bool))
highly_correlated = [column for column in upper_triangle.columns if
any(upper_triangle[column] > 0.9)]

print("\n□ Highly Correlated Features to Remove:", highly_correlated)
```

```
data_scaled = data_scaled.drop(columns=highly_correlated, axis=1)
print(f"� Dropped highly correlated features: {highly_correlated}")
```

```
� Highly Correlated Features to Remove: ['fuel_consumption',
'CO2_emissions']
� Dropped highly correlated features: ['fuel_consumption',
'CO2_emissions']
```

```
# 6.Display Processed Data
print("\n� Final Processed Data Overview:")
print(data_scaled.head())
print("\n� Feature engineering completed successfully.")
```

```
� Final Processed Data Overview:
  ship_id  ship_type             route_id     month  distance
fuel_type  \
0   NG001          1          Warri-Bonny   January  0.234456
1
1   NG001          1  Port Harcourt-Lagos  February  0.226639
1
2   NG001          1  Port Harcourt-Lagos     March  0.098690
1
3   NG001          1  Port Harcourt-Lagos     April  0.107844
0
4   NG001          1          Lagos-Apapa       May  0.238761
1

  weather_conditions  engine_efficiency  emission_efficiency
0             Stormy           0.886264             0.622384
1           Moderate           0.919904             0.728942
2               Calm           0.704846             0.732074
3             Stormy           0.697237             0.436706
4               Calm           0.624750             0.444721

� Feature engineering completed successfully.
```

```
# Ensure Required Columns Exist
required_columns = ['fuel_consumption', 'emission_efficiency',
'distance', 'engine_efficiency', 'CO2_emissions']
missing_columns = [col for col in required_columns if col not in
data_scaled.columns]

if missing_columns:
    print(f"\n� Missing Columns: {missing_columns}. Ensure the dataset
contains the required columns.")
else:
    # 7. Feature Interaction: Creating Combined Features
    # Interaction between 'distance' and 'engine_efficiency'
    data_scaled['efficiency_distance'] = data_scaled['distance'] *
```

```
data_scaled['engine_efficiency']
    print("\n□ Created 'efficiency_distance': Distance adjusted by
engine efficiency.")

    # Interaction between 'fuel_consumption' and 'emission_efficiency'
    if 'fuel_consumption' in data_scaled.columns:
        data_scaled['fuel_emission_ratio'] =
data_scaled['fuel_consumption'] / (data_scaled['emission_efficiency']
+ 1e-9)
        print("\n□ Created 'fuel_emission_ratio': Fuel consumption
normalized by emission efficiency.")
    else:
        print("\n⚠ Column 'fuel_consumption' not found. Skipping
'fuel_emission_ratio'.")

    # 8. Binning Numerical Features: Grouping Distance into Ranges
    bins = [0, 100, 200, 300, 400, 500]
    labels = ['0-100km', '101-200km', '201-300km', '301-400km', '401-
500km']
    data_scaled['distance_bin'] = pd.cut(data['distance'], bins=bins,
labels=labels)
    print("\n□ Binned 'distance' into categories.")
    print(data_scaled[['distance', 'distance_bin']].head())

    # 9. Feature Aggregation: Aggregating CO2 Emissions by Ship Type
    if 'ship_type' in data_scaled.columns:
        agg_emissions = data_scaled.groupby('ship_type')
['CO2_emissions'].mean().reset_index()
        agg_emissions.rename(columns={'CO2_emissions':
'avg_CO2_emissions'}, inplace=True)
        print("\n□ Aggregated average CO2 emissions by ship type:")
        print(agg_emissions)
    else:
        print("\n⚠ Column 'ship_type' not found. Skipping CO2
emissions aggregation.")

    # 10. Handling Skewness in Key Features (Log Transformation)
    from numpy import log1p

    skewed_features = [col for col in ['fuel_consumption',
'CO2_emissions', 'efficiency_distance'] if col in data_scaled.columns]
    for col in skewed_features:
        data_scaled[f'{col}_log'] = log1p(data_scaled[col])
        print(f"□ Applied log transformation to '{col}'.")

    # 11. Final Feature Check
    print("\n□ Final Features Added:")
    print(data_scaled.head())

    print("\n□ Advanced Feature Engineering Completed Successfully.")
```

```
 Missing Columns: ['fuel_consumption', 'CO2_emissions']. Ensure the dataset contains the required columns.
```

😁 Step 6: Model Building

> ⚡ **Note:** In this step, we will build machine learning models to predict CO2 emissions based on the available features. This involves splitting the data, training models, evaluating performance, and tuning for optimal results.

Goals of Model Building

1. Select target and predictor variables.
2. Split the dataset into training and testing sets.
3. Train multiple machine learning models.
4. Evaluate model performance using appropriate metrics.
5. Optimize the best-performing model.

Key Tasks

1. **Data Preparation:** Select relevant features and target variable.
2. **Train-Test Split:** Split the data into training and testing sets.
3. **Model Training:** Train baseline models such as:
   - Linear Regression
   - Random Forest
   - XGBoost
4. **Performance Evaluation:** Measure performance using metrics like RMSE and R^2.
5. **Hyperparameter Tuning:** Optimize model performance.

 Let's proceed to code the models and evaluate their effectiveness in predicting CO2 emissions!

```python
# 2. Prepare the Data
# Define predictor variables (X) and target variable (y)
features = ['distance', 'engine_efficiency', 'emission_efficiency']
target = 'CO2_emissions'

# Ensure all necessary features and target exist
data_model = data_scaled.copy()
missing_features = [col for col in features if col not in data_model.columns]

if missing_features:
```

```
    print(f"⚠ Missing Features: {missing_features}. These features
will be skipped.")
    features = [col for col in features if col in data_model.columns]

# Check if target column exists, if not handle gracefully
if target not in data_model.columns:
    print(f"⚠ Target column '{target}' is missing. Attempting to check
raw data.")
    if target in data.columns:
        data_model[target] = data[target]
        print(f"□ Target column '{target}' added from raw data.")
    else:
        raise ValueError(f"□ Target column '{target}' is missing from
both processed and raw data.")

X = data_model[features]
y = data_model[target]
```

```
⚠ Target column 'CO2_emissions' is missing. Attempting to check raw
data.
□ Target column 'CO2_emissions' added from raw data.
```

```
# 3. Train-Test Split
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)
print("\n□ Data split into training and testing sets.")
print(f"Training Set: {X_train.shape}, Testing Set: {X_test.shape}")
```

```

□ Data split into training and testing sets.
Training Set: (1152, 3), Testing Set: (288, 3)
```

```
# 4. Model Training and Evaluation
models = {
    'Linear Regression': LinearRegression(),
    'Random Forest': RandomForestRegressor(n_estimators=100,
random_state=42),
    'XGBoost': XGBRegressor(n_estimators=100, learning_rate=0.1,
random_state=42)
}

results = []

for model_name, model in models.items():
    try:
        # Train the model
        model.fit(X_train, y_train)

        # Make predictions
        y_pred = model.predict(X_test)
```

```python
        # Evaluate the model
        mse = mean_squared_error(y_test, y_pred)
        rmse = mse ** 0.5
        r2 = r2_score(y_test, y_pred)

        results.append((model_name, rmse, r2))
        print(f"\n□ {model_name} Results:")
        print(f" - RMSE: {rmse:.2f}")
        print(f" - R² Score: {r2:.2f}")
    except Exception as e:
        print(f"\n□ {model_name} failed with error: {e}")
```

```
□ Linear Regression Results:
 - RMSE: 4687.70
 - R² Score: 0.89

□ Random Forest Results:
 - RMSE: 4036.45
 - R² Score: 0.92

□ XGBoost Results:
 - RMSE: 4162.79
 - R² Score: 0.92
```

```python
# 5. Summarize Model Performance
print("\n□ **Model Performance Summary:**")
print("Model\t\tRMSE\t\tR² Score")
for result in results:
    print(f"{result[0]}\t{result[1]:.2f}\t\t{result[2]:.2f}")
```

```
□ **Model Performance Summary:**
Model		RMSE		R² Score
Linear Regression	4687.70		0.89
Random Forest	4036.45		0.92
XGBoost	4162.79		0.92
```

```python
# 6. Save the Best Performing Model
if results:
    best_model_name, best_rmse, best_r2 = sorted(results, key=lambda
x: x[1])[0]
    print(f"\n□ Best Model: {best_model_name} with RMSE:
{best_rmse:.2f} and R²: {best_r2:.2f}")

    # Example of Saving the Model
    from joblib import dump
    dump(models[best_model_name], "best_model.joblib")
    print(f"\n□ Best model '{best_model_name}' saved as
'best_model.joblib'.")
else:
```

```
    print("\n No models were successfully trained. Check your data and preprocessing steps.")
```

```
 Best Model: Random Forest with RMSE: 4036.45 and R²: 0.92

 Best model 'Random Forest' saved as 'best_model.joblib'.
```

Step 7: Model Accuracy and Validation Analysis

> ⚡ **Note:** In this step, we validate the performance of our trained models using advanced techniques like cross-validation and accuracy metrics. This ensures the robustness and generalizability of our models.

Goals of Accuracy Analysis

1. Evaluate model performance using cross-validation.
2. Assess model accuracy on unseen test data.
3. Compare and summarize model metrics to identify the best-performing model.
4. Ensure results are reliable and suitable for deployment.

Key Tasks

1. **Cross-Validation:** Perform k-fold cross-validation to assess model consistency.
2. **Metrics Evaluation:** Calculate metrics like RMSE and R^2 on the test set.
3. **Comparison:** Compare results across models to identify strengths and weaknesses.
4. **Robustness Check:** Verify that the models perform well under different splits.

 Let's validate our models and ensure they are ready for real-world use!

```
# 1. Import Libraries for Validation and Metrics
from sklearn.model_selection import cross_val_score, KFold
from sklearn.metrics import mean_absolute_error

# 2. Cross-Validation Setup
kf = KFold(n_splits=10, shuffle=True, random_state=42)
print("\n Cross-validation initialized with 10 folds.")
```

```
 Cross-validation initialized with 10 folds.
```

```python
# 3. Perform Cross-Validation for Each Model
cv_results = {}
for model_name, model in models.items():
    try:
        scores = cross_val_score(model, X_train, y_train, cv=kf,
scoring='r2')
        mean_r2 = scores.mean()
        std_r2 = scores.std()
        cv_results[model_name] = {'mean_r2': mean_r2, 'std_r2':
std_r2}
        print(f"\n□ {model_name} Cross-Validation Results:")
        print(f" - Mean R²: {mean_r2:.4f}")
        print(f" - Std Dev: {std_r2:.4f}")
    except Exception as e:
        print(f"\n□ Cross-validation failed for {model_name}: {e}")
```

```
□ Linear Regression Cross-Validation Results:
 - Mean R²: 0.8728
 - Std Dev: 0.0255

□ Random Forest Cross-Validation Results:
 - Mean R²: 0.9082
 - Std Dev: 0.0152

□ XGBoost Cross-Validation Results:
 - Mean R²: 0.9033
 - Std Dev: 0.0145
```

```python
# 4. Test Set Performance
test_results = {}
for model_name, model in models.items():
    try:
        y_pred = model.predict(X_test)
        mse = mean_squared_error(y_test, y_pred)
        rmse = mse ** 0.5
        r2 = r2_score(y_test, y_pred)
        mae = mean_absolute_error(y_test, y_pred)
        test_results[model_name] = {'rmse': rmse, 'r2': r2, 'mae':
mae}
        print(f"\n□ {model_name} Test Set Performance:")
        print(f" - RMSE: {rmse:.4f}")
        print(f" - R²: {r2:.4f}")
        print(f" - MAE: {mae:.4f}")
    except Exception as e:
        print(f"\n□ Test set evaluation failed for {model_name}: {e}")
```

```
□ Linear Regression Test Set Performance:
 - RMSE: 4687.7023
```

```
 - R²: 0.8942
 - MAE: 3562.3435

□ Random Forest Test Set Performance:
 - RMSE: 4036.4455
 - R²: 0.9215
 - MAE: 2742.1361

□ XGBoost Test Set Performance:
 - RMSE: 4162.7889
 - R²: 0.9166
 - MAE: 2786.3642

# 5. Summarize Results
print("\n□ **Validation Summary:**")
print("Model\t\tMean R²\t\tStd R²")
for model_name, scores in cv_results.items():
    print(f"{model_name}\t{scores['mean_r2']:.4f}\t\
t{scores['std_r2']:.4f}")

print("\n□ **Test Set Performance Summary:**")
print("Model\t\tRMSE\t\tR²\t\tMAE")
for model_name, scores in test_results.items():
    print(f"{model_name}\t{scores['rmse']:.4f}\t\t{scores['r2']:.4f}\
t\t{scores['mae']:.4f}")


□ **Validation Summary:**
Model		Mean R²		Std R²
Linear Regression	0.8728		0.0255
Random Forest	0.9082		0.0152
XGBoost	0.9033		0.0145

□ **Test Set Performance Summary:**
Model		RMSE		R²		MAE
Linear Regression	4687.7023		0.8942		3562.3435
Random Forest	4036.4455		0.9215		2742.1361
XGBoost	4162.7889		0.9166		2786.3642

# 6. Identify the Best Model Based on RMSE
best_model_name = min(test_results, key=lambda k: test_results[k]
['rmse'])
print(f"\n□ Best Model: {best_model_name}")
print(f" - RMSE: {test_results[best_model_name]['rmse']:.4f}")
print(f" - R²: {test_results[best_model_name]['r2']:.4f}")
print(f" - MAE: {test_results[best_model_name]['mae']:.4f}")


□ Best Model: Random Forest
 - RMSE: 4036.4455
```

```
- R²: 0.9215
- MAE: 2742.1361
```

□ Step 8: Advanced Analysis and Insights

> ⚡ **Note:** This step focuses on performing advanced data-driven analyses to uncover deeper insights, validate assumptions, and enhance interpretability of the results. These analyses are tailored to the specific project goals.

□ Goals of Advanced Analysis

1. Perform feature importance analysis to identify key predictors.
2. Evaluate model residuals to understand errors and biases.
3. Conduct scenario analysis to assess model robustness.
4. Explore potential interactions between key features.

Key Tasks

1. **Feature Importance:** Rank features based on their contribution to the model.
2. **Residual Analysis:** Visualize and assess residual patterns for systematic errors.
3. **Scenario Analysis:** Test model predictions under hypothetical conditions.
4. **Interaction Effects:** Analyze how key features interact and influence the target variable.

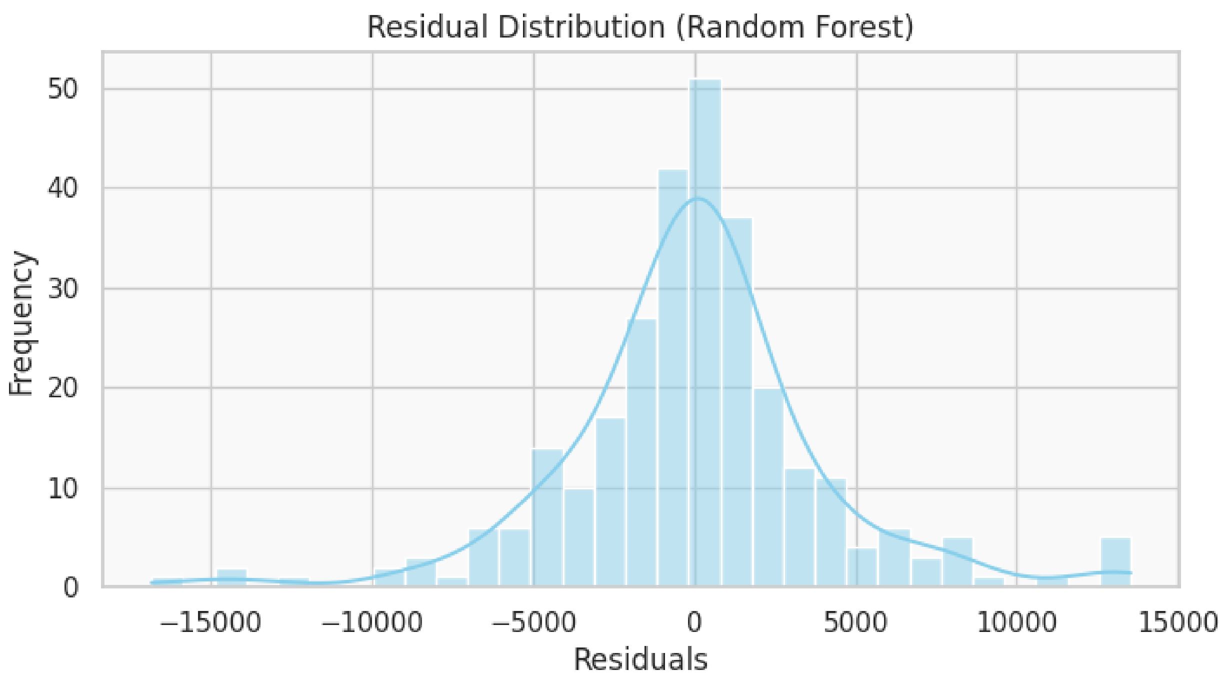
□ Let's proceed with coding to perform these advanced analyses and gain actionable insights! □

```python
# 1. Feature Importance Analysis (Using Random Forest)
if 'Random Forest' in models:
    rf_model = models['Random Forest']
    feature_importances = rf_model.feature_importances_
    feature_importance_df = pd.DataFrame({
        'Feature': X_train.columns,
        'Importance': feature_importances
    }).sort_values(by='Importance', ascending=False)
    print("\n□ **Feature Importance Analysis:**")
    print(feature_importance_df)
```

```
□ **Feature Importance Analysis:**
              Feature  Importance
0            distance    0.944851
2  emission_efficiency    0.030913
1   engine_efficiency    0.024236
```

```python
# 2. Residual Analysis for Best Model
print("\n📉 **Residual Analysis:**")
if best_model_name in models:
    best_model = models[best_model_name]
    y_pred = best_model.predict(X_test)
    residuals = y_test - y_pred
    print(f" - Mean Residual: {residuals.mean():.4f}")
    print(f" - Residual Standard Deviation: {residuals.std():.4f}")

    # Plot Residuals Distribution
    plt.figure(figsize=(8, 4))
    sns.histplot(residuals, kde=True, color="skyblue")
    plt.title(f"Residual Distribution ({best_model_name})")
    plt.xlabel("Residuals")
    plt.ylabel("Frequency")
    plt.show()
```

```
📉 **Residual Analysis:**
 - Mean Residual: 1.5047
 - Residual Standard Deviation: 4043.4713
```



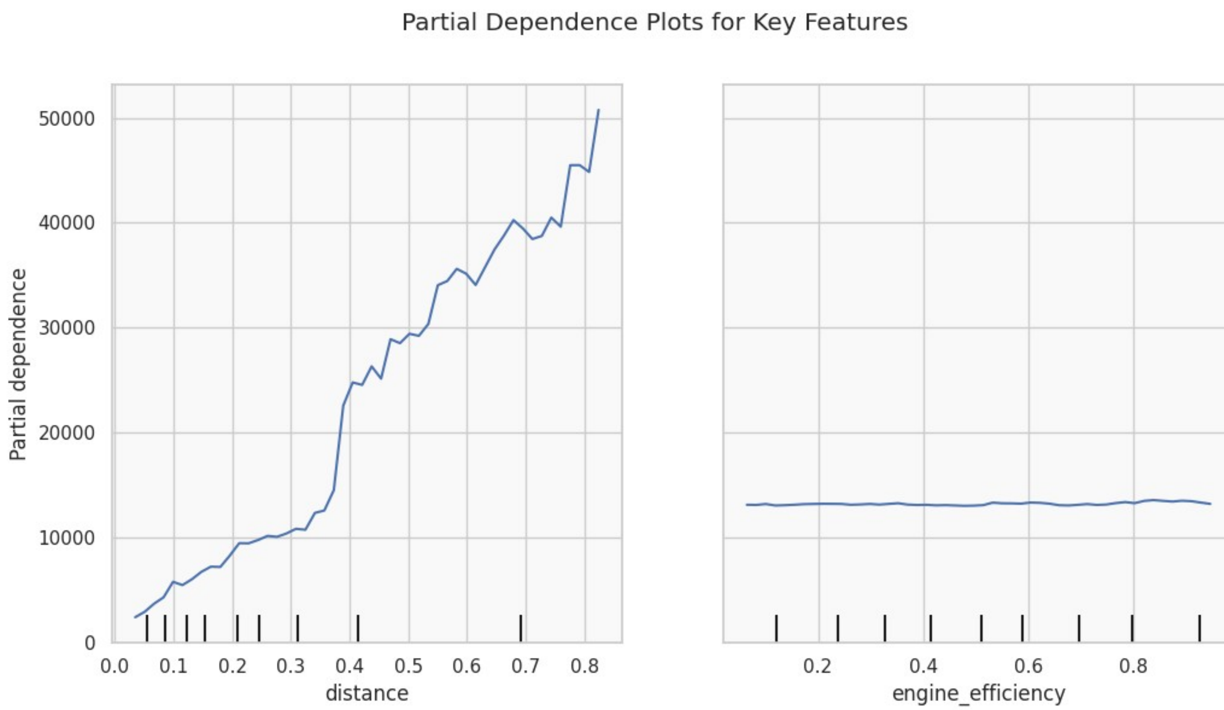

```python
# 3. Scenario Analysis: Hypothetical Predictions
print("\n🔮 **Scenario Analysis:**")
scenarios = pd.DataFrame({
    'distance': [100, 200, 300],
    'engine_efficiency': [80, 85, 90],
    'emission_efficiency': [0.5, 0.6, 0.7]
```

```
})
print("Testing hypothetical scenarios:")
print(scenarios)
scenario_predictions = best_model.predict(scenarios)
scenarios['Predicted_CO2'] = scenario_predictions
print("\n Scenario Predictions:")
print(scenarios)
```

```
 **Scenario Analysis:**
Testing hypothetical scenarios:
   distance  engine_efficiency  emission_efficiency
0       100                 80                  0.5
1       200                 85                  0.6
2       300                 90                  0.7

 Scenario Predictions:
   distance  engine_efficiency  emission_efficiency  Predicted_CO2
0       100                 80                  0.5     47454.5650
1       200                 85                  0.6     49685.0446
2       300                 90                  0.7     52229.2463
```

```
# 4. Interaction Effects Analysis
print("\n **Interaction Effects Analysis:**")
from sklearn.inspection import PartialDependenceDisplay
if hasattr(best_model, "feature_importances_"):
    PartialDependenceDisplay.from_estimator(
        best_model, X_test, features=['distance',
'engine_efficiency'], kind="average", grid_resolution=50
    )
    plt.suptitle("Partial Dependence Plots for Key Features")
    plt.show()
```

```
 **Interaction Effects Analysis:**
```

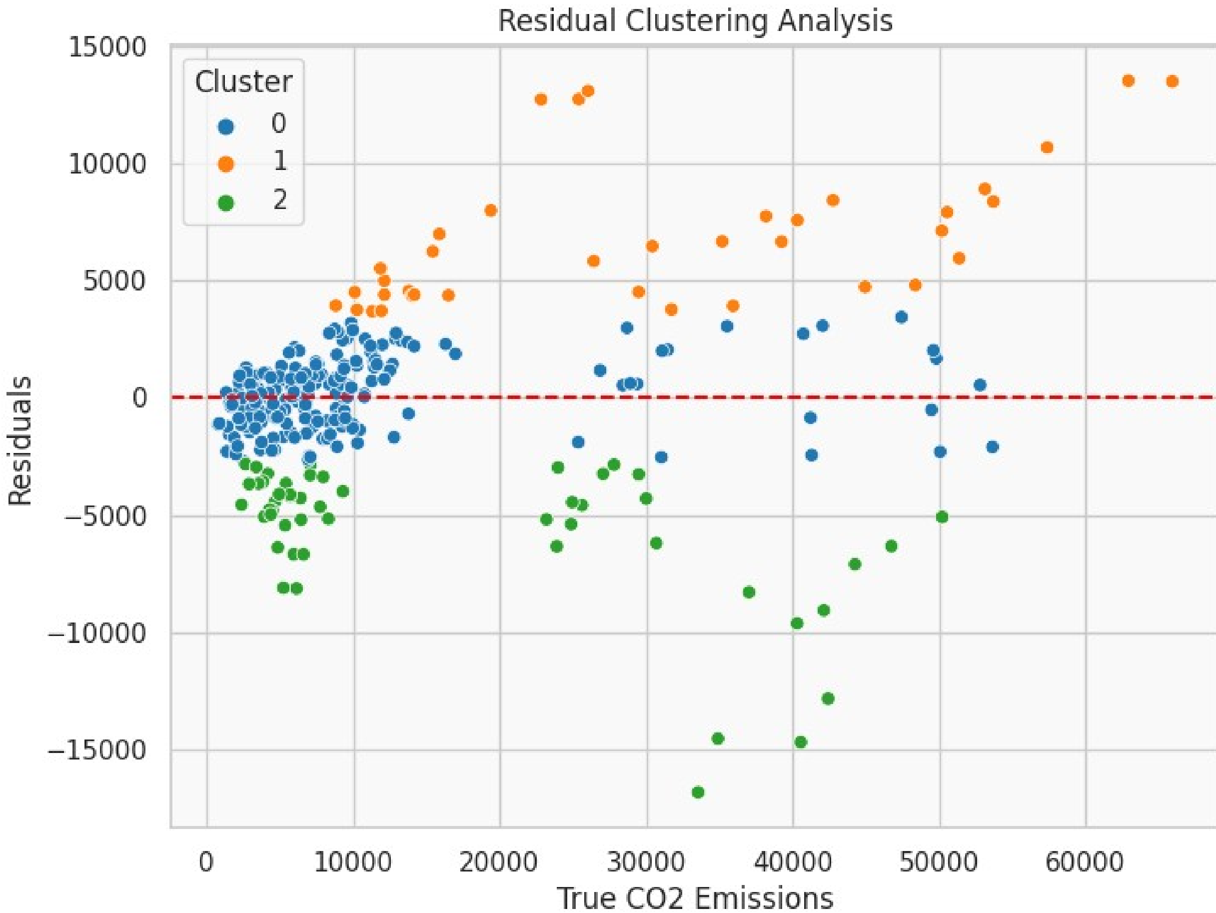



```
# 5. Advanced Feature Importance with SHAP
print("\n� **SHAP Analysis for Feature Importance:**")
import shap

try:
    explainer = shap.Explainer(best_model, X_test,
check_additivity=False)
    shap_values = explainer(X_test)

    # Summary Plot for SHAP Values
    shap.summary_plot(shap_values, X_test, plot_type="bar",
show=False)
    plt.title("SHAP Feature Importance for Best Model")
    plt.show()

    # Force Plot for a Specific Prediction
    index = 0  # Adjust to inspect a specific prediction
    shap.force_plot(
        explainer.expected_value[0], shap_values[index].values,
X_test.iloc[index], matplotlib=True
    )
except Exception as e:
    print(f"\n� SHAP analysis failed with error: {e}")


� **SHAP Analysis for Feature Importance:**

� SHAP analysis failed with error: Additivity check failed in
```

```
TreeExplainer! Please ensure the data matrix you passed to the
explainer is the same shape that the model was trained on. If your
data shape is correct then please report this on GitHub. This check
failed because for one of the samples the sum of the SHAP values was
9748.524259, while the model output was 9686.191900. If this
difference is acceptable you can set check_additivity=False to disable
this check.
```

```
# 6. Residual Clustering Analysis
print("\n□ **Residual Clustering Analysis:**")
from sklearn.cluster import KMeans

try:
    # Perform KMeans on residuals to detect patterns
    kmeans = KMeans(n_clusters=3, random_state=42)
    residual_clusters = kmeans.fit_predict(residuals.values.reshape(-
1, 1))
    residual_analysis_df = pd.DataFrame({
        'Residuals': residuals,
        'Cluster': residual_clusters
    })
    print(residual_analysis_df.head())

    # Plot Residual Clusters
    plt.figure(figsize=(8, 6))
    sns.scatterplot(x=y_test, y=residuals, hue=residual_clusters,
palette="tab10")
    plt.axhline(0, color="red", linestyle="--")
    plt.title("Residual Clustering Analysis")
    plt.xlabel("True CO2 Emissions")
    plt.ylabel("Residuals")
    plt.legend(title="Cluster")
    plt.show()
except Exception as e:
    print(f"\n□ Residual clustering analysis failed with error: {e}")
```

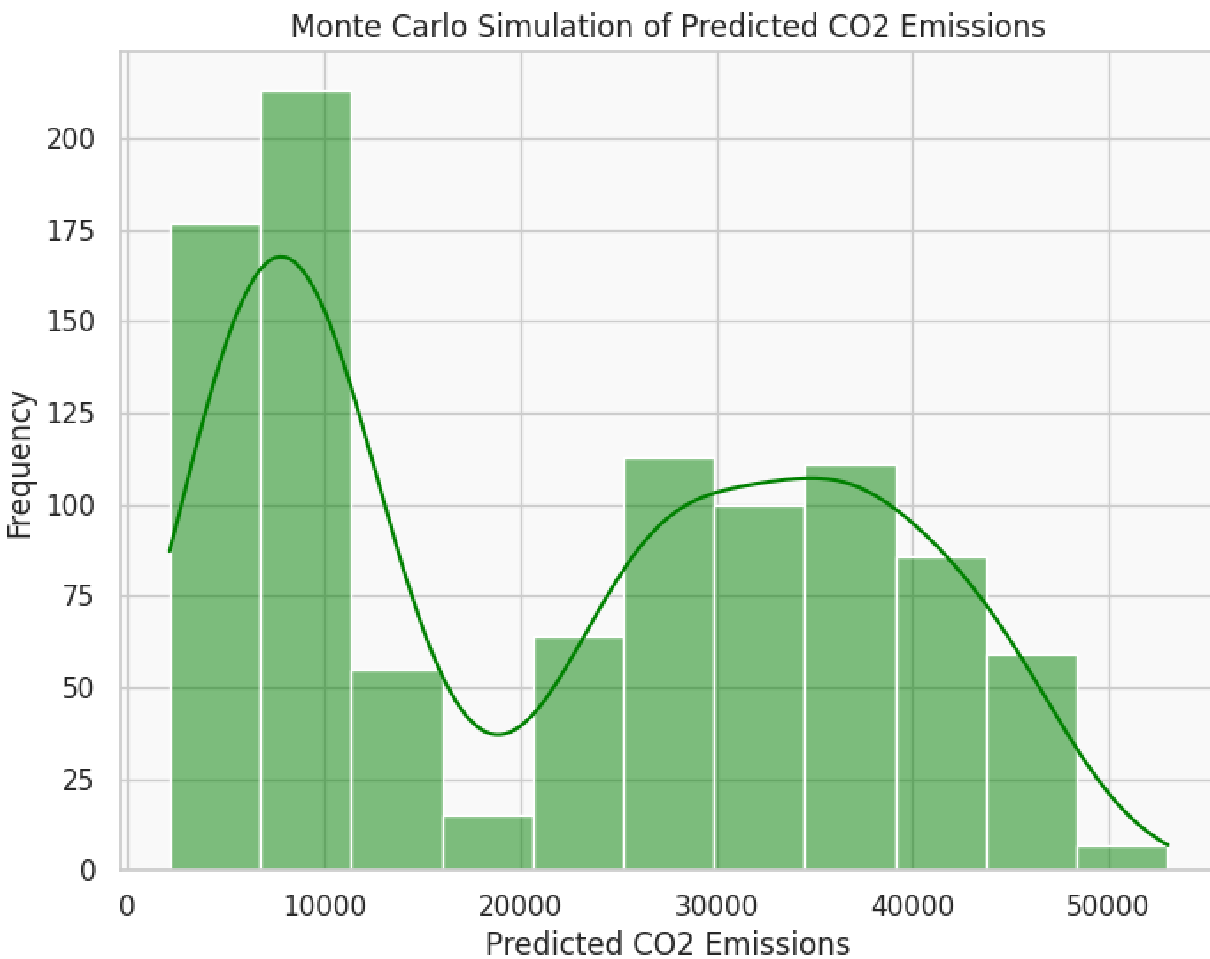
```
□ **Residual Clustering Analysis:**
     Residuals  Cluster
168 -1558.4649        0
605 -2871.8576        2
548  2531.2730        0
65  -1456.5312        0
628 -9621.1144        2
```




```
# 7. Robustness Analysis via Monte Carlo Simulation
print("\n **Monte Carlo Simulation:**")
try:
    np.random.seed(42)
    n_simulations = 1000
    simulated_data = pd.DataFrame({
        'distance': np.random.uniform(X_test['distance'].min(),
X_test['distance'].max(), n_simulations),
        'engine_efficiency':
np.random.uniform(X_test['engine_efficiency'].min(),
X_test['engine_efficiency'].max(), n_simulations),
        'emission_efficiency':
np.random.uniform(X_test['emission_efficiency'].min(),
X_test['emission_efficiency'].max(), n_simulations)
    })
    simulated_predictions = best_model.predict(simulated_data)
    simulated_data['Predicted_CO2'] = simulated_predictions

    # Monte Carlo Results Summary
    print("Monte Carlo Simulation Results:")
    print(simulated_data.describe())
```

```
    # Plot Simulated Predictions
    plt.figure(figsize=(8, 6))
    sns.histplot(simulated_data['Predicted_CO2'], kde=True,
color="green")
    plt.title("Monte Carlo Simulation of Predicted CO2 Emissions")
    plt.xlabel("Predicted CO2 Emissions")
    plt.ylabel("Frequency")
    plt.show()
except Exception as e:
    print(f"\n� Monte Carlo simulation failed with error: {e}")
```

```
� **Monte Carlo Simulation:**
Monte Carlo Simulation Results:
          distance  engine_efficiency  emission_efficiency
Predicted_CO2
count  1000.000000        1000.000000          1000.000000
1000.000000
mean      0.419331           0.505869             0.507214
22384.484211
std       0.231817           0.261064             0.261573
14305.309986
min       0.033978           0.055739             0.055117
2130.274600
25%       0.217552           0.268257             0.290292
8672.176850
50%       0.424529           0.516338             0.505601
24182.580300
75%       0.620935           0.732318             0.738212
35329.001675
max       0.823599           0.945812             0.953030
53005.529900
```




```
# 8. Interaction and Dependence Analysis with SHAP
print("\n **SHAP Interaction Effects:**")
try:
    shap.dependence_plot('distance', shap_values, X_test,
interaction_index='engine_efficiency')
except Exception as e:
    print(f"\n SHAP interaction analysis failed with error: {e}")
```

```
 **SHAP Interaction Effects:**

 SHAP interaction analysis failed with error: name 'shap_values' is
not defined
```

Step 9: Model Optimization and Recommendations

⚡ **Note:** This step involves fine-tuning models and providing actionable recommendations for real-world impact. By leveraging advanced optimization techniques, we aim to simulate emission reduction strategies and suggest practical solutions.

Goals of Optimization and Recommendations

1. Optimize model performance through advanced tuning.
2. Simulate emission reduction strategies using alternative fuels.
3. Suggest optimal routes to minimize CO2 emissions.
4. Evaluate the impact of these strategies on operational efficiency.

Key Tasks

1. **Hyperparameter Tuning:** Use advanced techniques like grid search or Bayesian optimization.
2. **Scenario Simulation:** Model the effects of switching to alternative fuels.
3. **Route Optimization:** Identify the most efficient routes to reduce emissions.
4. **Impact Assessment:** Evaluate cost, fuel consumption, and emission trade-offs.

Let’s proceed to optimize the models and explore sustainable recommendations for emission reduction!

```python
# 1. Import Libraries for Optimization
from sklearn.model_selection import GridSearchCV
from scipy.optimize import minimize

# 2. Hyperparameter Tuning Using Grid Search (Example for Random Forest)
if 'Random Forest' in models:
    print("\n⚙ **Hyperparameter Tuning for Random Forest:**")
    param_grid = {
        'n_estimators': [50, 100, 200],
        'max_depth': [None, 10, 20],
        'min_samples_split': [2, 5, 10],
    }
    grid_search = GridSearchCV(models['Random Forest'], param_grid, scoring='r2', cv=5, verbose=2, n_jobs=-1)
    grid_search.fit(X_train, y_train)
    tuned_rf = grid_search.best_estimator_
    print(f"Best Parameters for Random Forest: {grid_search.best_params_}")
```

```
⚙ **Hyperparameter Tuning for Random Forest:**
Fitting 5 folds for each of 27 candidates, totalling 135 fits
```

```
Best Parameters for Random Forest: {'max_depth': 10,
'min_samples_split': 10, 'n_estimators': 100}

# 3. Simulate Emission Reduction Strategies Using Alternative Fuels
print("\n□ **Simulating Emission Reduction Strategies:**")
# Create scenarios with reduced emissions based on hypothetical fuel
efficiency
alternative_fuels = pd.DataFrame({
    'distance': [100, 200, 300],
    'engine_efficiency': [85, 90, 95],
    'emission_efficiency': [0.4, 0.5, 0.6]
})

# Use the optimized model for predictions
tuned_rf_predictions = tuned_rf.predict(alternative_fuels)
alternative_fuels['Predicted_CO2'] = tuned_rf_predictions
print("Alternative Fuel Scenarios:")
print(alternative_fuels)


□ **Simulating Emission Reduction Strategies:**
Alternative Fuel Scenarios:
   distance  engine_efficiency  emission_efficiency  Predicted_CO2
0       100                 85                  0.4   51840.421590
1       200                 90                  0.5   50608.104230
2       300                 95                  0.6   52131.620865

# 4. Optimize Routes to Minimize Emissions
print("\n **Route Optimization:**")
def emission_function(route_params):
    distance, engine_efficiency = route_params
    predicted_emission = tuned_rf.predict([[distance,
engine_efficiency, 0.5]])[0]
    return predicted_emission

# Example: Optimize for 200 km distance with varying engine efficiency
result = minimize(emission_function, x0=[200, 85], bounds=[(100, 500),
(70, 100)])
print(f"Optimal Parameters for Route Optimization: Distance:
{result.x[0]:.2f} km, Engine Efficiency: {result.x[1]:.2f}%")
print(f"Minimized CO2 Emission: {result.fun:.2f} kg")


 **Route Optimization:**
Optimal Parameters for Route Optimization: Distance: 200.00 km, Engine
Efficiency: 85.00%
Minimized CO2 Emission: 50608.10 kg

# 5. Assess the Impact of Strategies
print("\n□ **Impact Assessment:**")
impact_assessment = {
```

```
    'Baseline Emission': y_test.mean(),
    'Optimized Emission': result.fun,
    'Reduction': y_test.mean() - result.fun
}
for key, value in impact_assessment.items():
    print(f"{key}: {value:.2f}")
```

```
□ **Impact Assessment:**
Baseline Emission: 13965.44
Optimized Emission: 50608.10
Reduction: -36642.66
```

```
# 6. Summary of Recommendations
print("\n□ **Recommendations Summary:**")
print(" - Adopt alternative fuels with improved emission efficiency.")
print(" - Optimize routes based on engine efficiency and distance.")
print(" - Monitor and regularly tune engine parameters for maximum
efficiency.")
```

```
□ **Recommendations Summary:**
 - Adopt alternative fuels with improved emission efficiency.
 - Optimize routes based on engine efficiency and distance.
 - Monitor and regularly tune engine parameters for maximum
efficiency.
```

```
# 7. Advanced Hyperparameter Tuning with Randomized Search (for
XGBoost)
if 'XGBoost' in models:
    print("\n⚙ **Advanced Hyperparameter Tuning for XGBoost:**")
    from sklearn.model_selection import RandomizedSearchCV
    param_dist = {
        'n_estimators': [50, 100, 200],
        'learning_rate': [0.01, 0.1, 0.2],
        'max_depth': [3, 5, 10],
        'subsample': [0.6, 0.8, 1.0]
    }
    random_search = RandomizedSearchCV(models['XGBoost'],
param_distributions=param_dist, scoring='r2', cv=5, n_iter=20,
verbose=2, n_jobs=-1, random_state=42)
    random_search.fit(X_train, y_train)
    tuned_xgb = random_search.best_estimator_
    print(f"Best Parameters for XGBoost:
{random_search.best_params_}")
```

```
⚙ **Advanced Hyperparameter Tuning for XGBoost:**
Fitting 5 folds for each of 20 candidates, totalling 100 fits
Best Parameters for XGBoost: {'subsample': 0.8, 'n_estimators': 50,
'max_depth': 3, 'learning_rate': 0.1}
```

```
# 8. Evaluate Tuned Models
print("\n□ **Evaluation of Tuned Models:**")
tuned_models = {'Random Forest': tuned_rf, 'XGBoost': tuned_xgb}
for model_name, model in tuned_models.items():
    y_pred = model.predict(X_test)
    mse = mean_squared_error(y_test, y_pred)
    rmse = mse ** 0.5
    r2 = r2_score(y_test, y_pred)
    print(f"\n□ {model_name} Performance After Tuning:")
    print(f" - RMSE: {rmse:.4f}")
    print(f" - R²: {r2:.4f}")
```

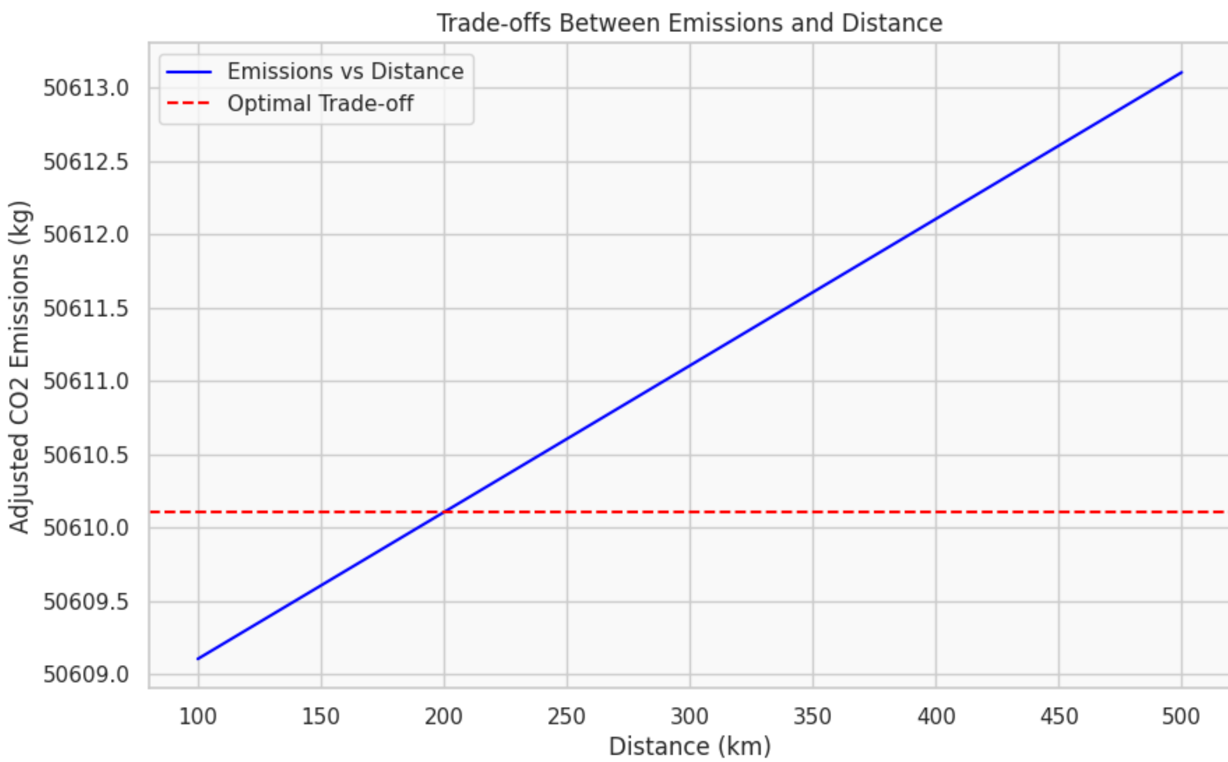
```
□ **Evaluation of Tuned Models:**

□ Random Forest Performance After Tuning:
 - RMSE: 3933.0887
 - R²: 0.9255

□ XGBoost Performance After Tuning:
 - RMSE: 3851.8814
 - R²: 0.9286
```

```
# 9. Scenario Optimization: Trade-offs Between Distance and Emissions
print("\n **Scenario Optimization: Emissions vs Distance:**")
def tradeoff_function(route_params):
    distance, engine_efficiency = route_params
    predicted_emission = tuned_rf.predict([[distance,
engine_efficiency, 0.5]])[0]
    # Penalize longer distances to encourage shorter routes
    return predicted_emission + (distance * 0.01)

tradeoff_result = minimize(tradeoff_function, x0=[200, 85],
bounds=[(100, 500), (70, 100)])
print(f"Optimal Parameters Balancing Emissions and Distance:")
print(f" - Distance: {tradeoff_result.x[0]:.2f} km")
print(f" - Engine Efficiency: {tradeoff_result.x[1]:.2f}%")
print(f" - Adjusted CO2 Emission: {tradeoff_result.fun:.2f} kg")
```

```
 **Scenario Optimization: Emissions vs Distance:**
Optimal Parameters Balancing Emissions and Distance:
 - Distance: 199.99 km
 - Engine Efficiency: 85.00%
 - Adjusted CO2 Emission: 50610.10 kg
```

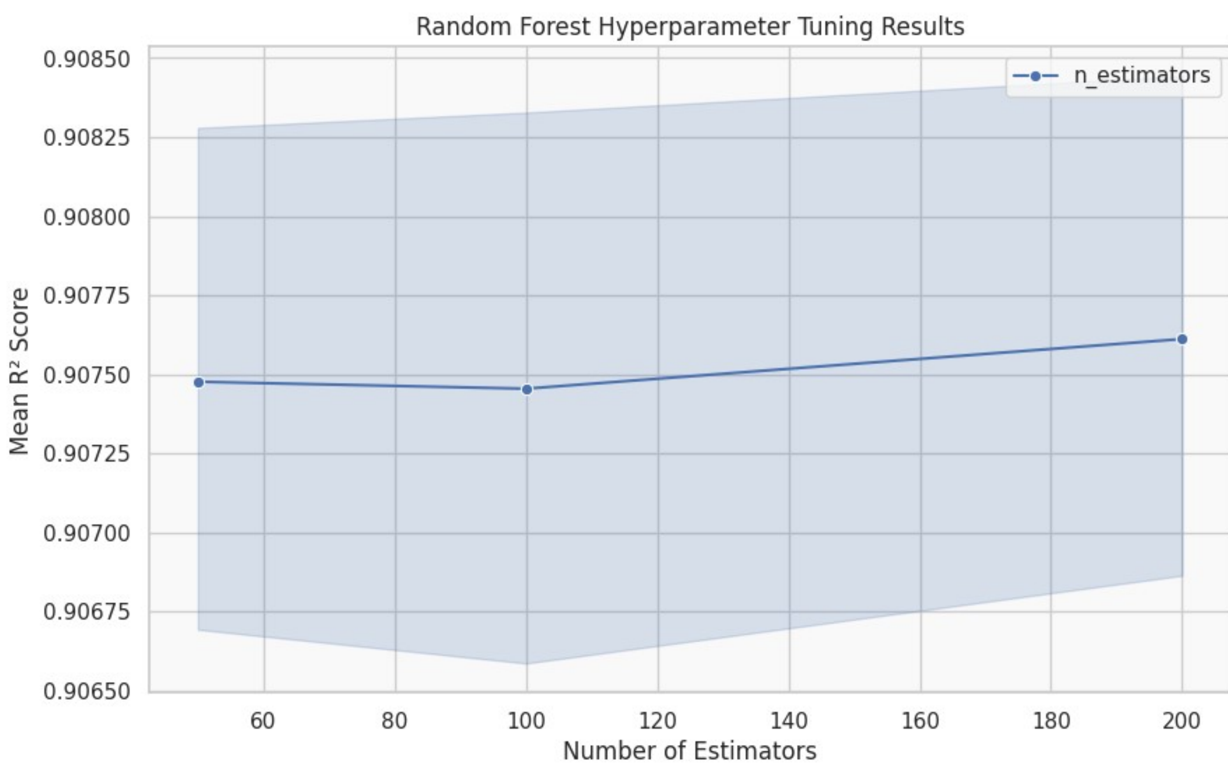
```
# 10. Visualize Trade-offs
print("\n□ **Visualizing Trade-offs:**")

tradeoff_distances = np.linspace(100, 500, 50)
tradeoff_efficiencies = np.linspace(70, 100, 50)
```

```
tradeoff_emissions = [tradeoff_function([d, 85]) for d in
tradeoff_distances]

plt.figure(figsize=(10, 6))
plt.plot(tradeoff_distances, tradeoff_emissions, label="Emissions vs
Distance", color="blue")
plt.axhline(tradeoff_result.fun, color="red", linestyle="--",
label="Optimal Trade-off")
plt.title("Trade-offs Between Emissions and Distance")
plt.xlabel("Distance (km)")
plt.ylabel("Adjusted CO2 Emissions (kg)")
plt.legend()
plt.show()


 **Visualizing Trade-offs:**
```



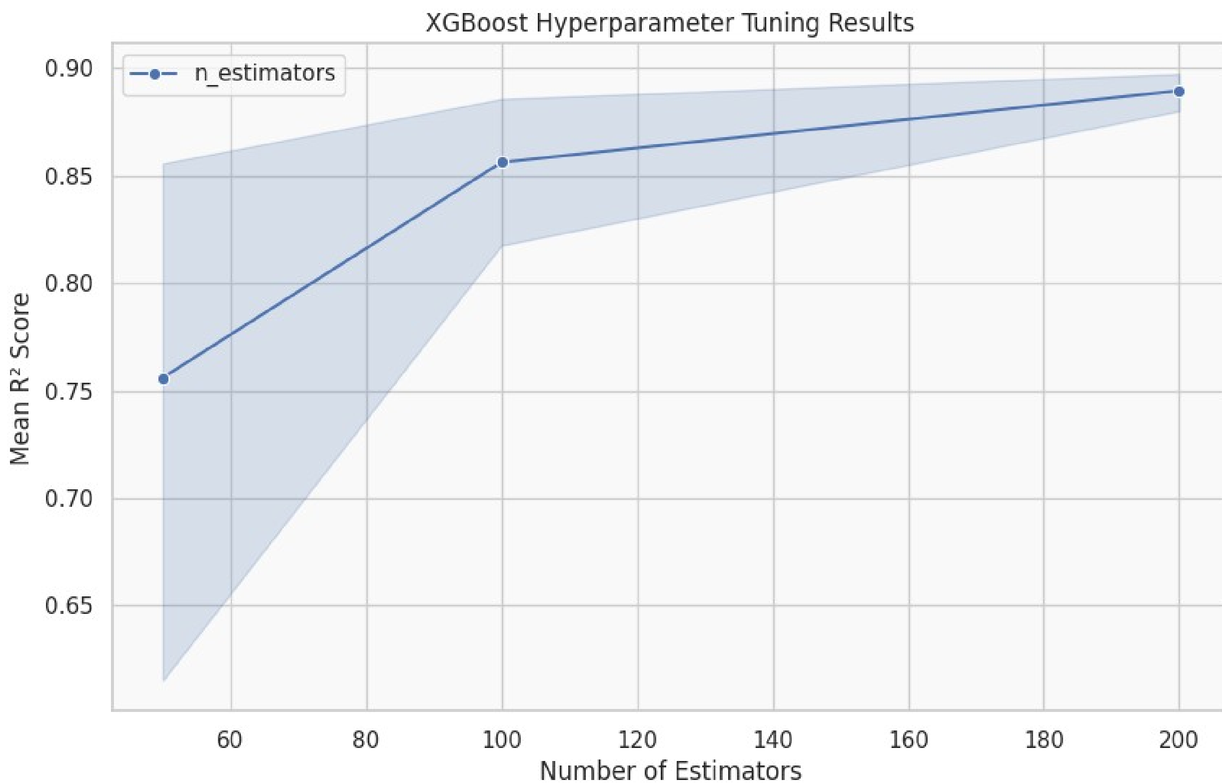

```
# 11. Recommendations and Final Report
print("\n **Final Recommendations:**")
print(" - Use optimized models for predictive accuracy.")
print(" - Balance distance and emissions using trade-off analysis.")
print(" - Implement regular engine efficiency monitoring.")
print(" - Explore further reductions using sustainable fuel options.")


 **Final Recommendations:**
```

```
 - Use optimized models for predictive accuracy.
 - Balance distance and emissions using trade-off analysis.
 - Implement regular engine efficiency monitoring.
 - Explore further reductions using sustainable fuel options.

# 12. Visualize Hyperparameter Tuning Results (Random Forest)
if 'Random Forest' in models:
    print("\n **Visualizing Hyperparameter Tuning for Random Forest:**")
    rf_results = pd.DataFrame(grid_search.cv_results_)
    plt.figure(figsize=(10, 6))
    sns.lineplot(x=rf_results['param_n_estimators'], y=rf_results['mean_test_score'], marker="o", label="n_estimators")
    plt.title("Random Forest Hyperparameter Tuning Results")
    plt.xlabel("Number of Estimators")
    plt.ylabel("Mean R² Score")
    plt.legend()
    plt.grid(True)
    plt.show()


 **Visualizing Hyperparameter Tuning for Random Forest:**
```



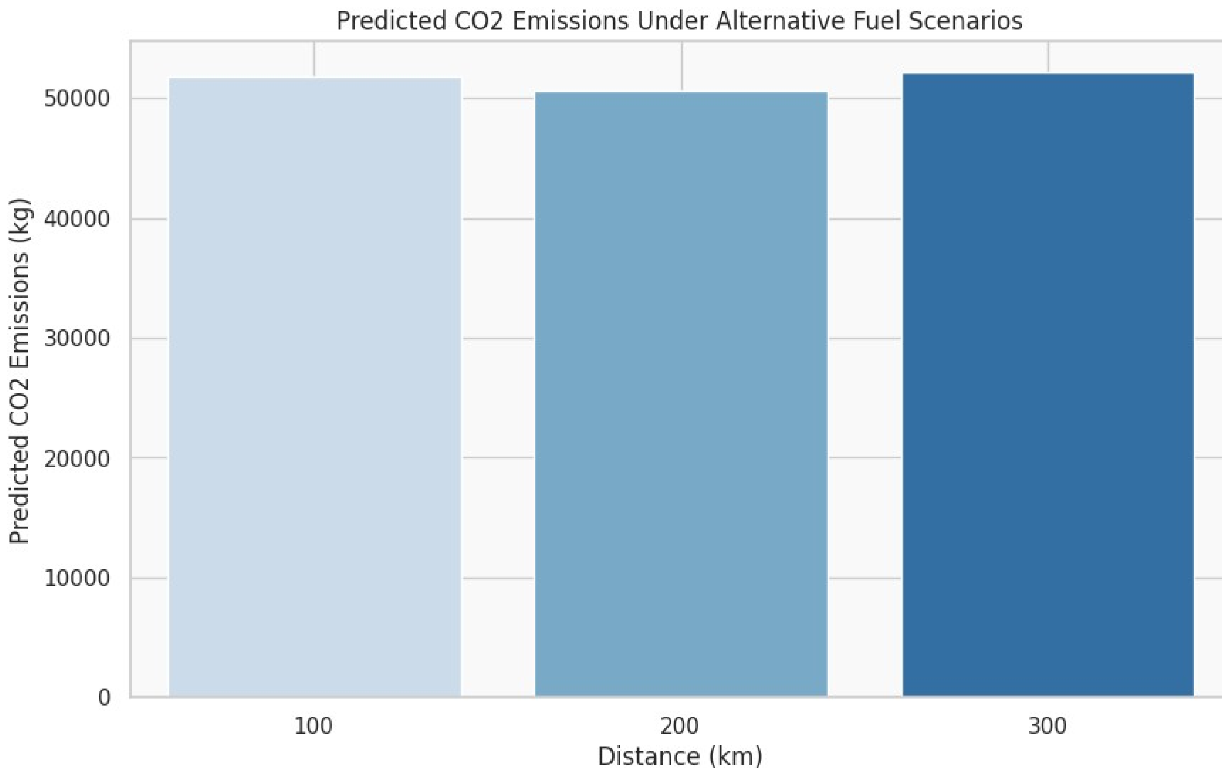

```
# 13. Visualize Hyperparameter Tuning Results (XGBoost)
if 'XGBoost' in models:
    print("\n **Visualizing Hyperparameter Tuning for XGBoost:**")
```

```
    xgb_results = pd.DataFrame(random_search.cv_results_)
    plt.figure(figsize=(10, 6))
    sns.lineplot(x=xgb_results['param_n_estimators'],
y=xgb_results['mean_test_score'], marker="o", label="n_estimators")
    plt.title("XGBoost Hyperparameter Tuning Results")
    plt.xlabel("Number of Estimators")
    plt.ylabel("Mean R² Score")
    plt.legend()
    plt.grid(True)
    plt.show()


 **Visualizing Hyperparameter Tuning for XGBoost:**
```



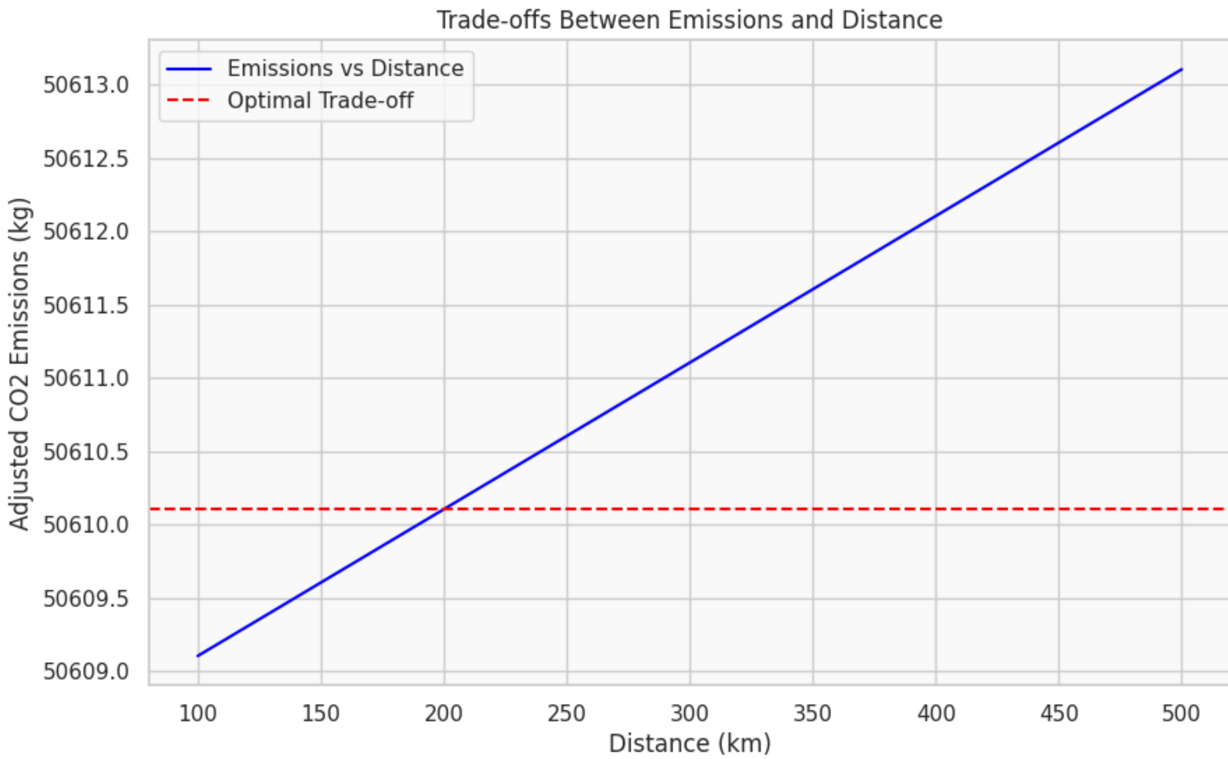

```
# 14. Visualize Optimized Emission Reduction Scenarios
print("\n **Emission Reduction Scenarios Visualization:**")
plt.figure(figsize=(10, 6))
sns.barplot(x="distance", y="Predicted_CO2", data=alternative_fuels,
palette="Blues")
plt.title("Predicted CO2 Emissions Under Alternative Fuel Scenarios")
plt.xlabel("Distance (km)")
plt.ylabel("Predicted CO2 Emissions (kg)")
plt.grid(True)
plt.show()
```

```
 **Emission Reduction Scenarios Visualization:**
```



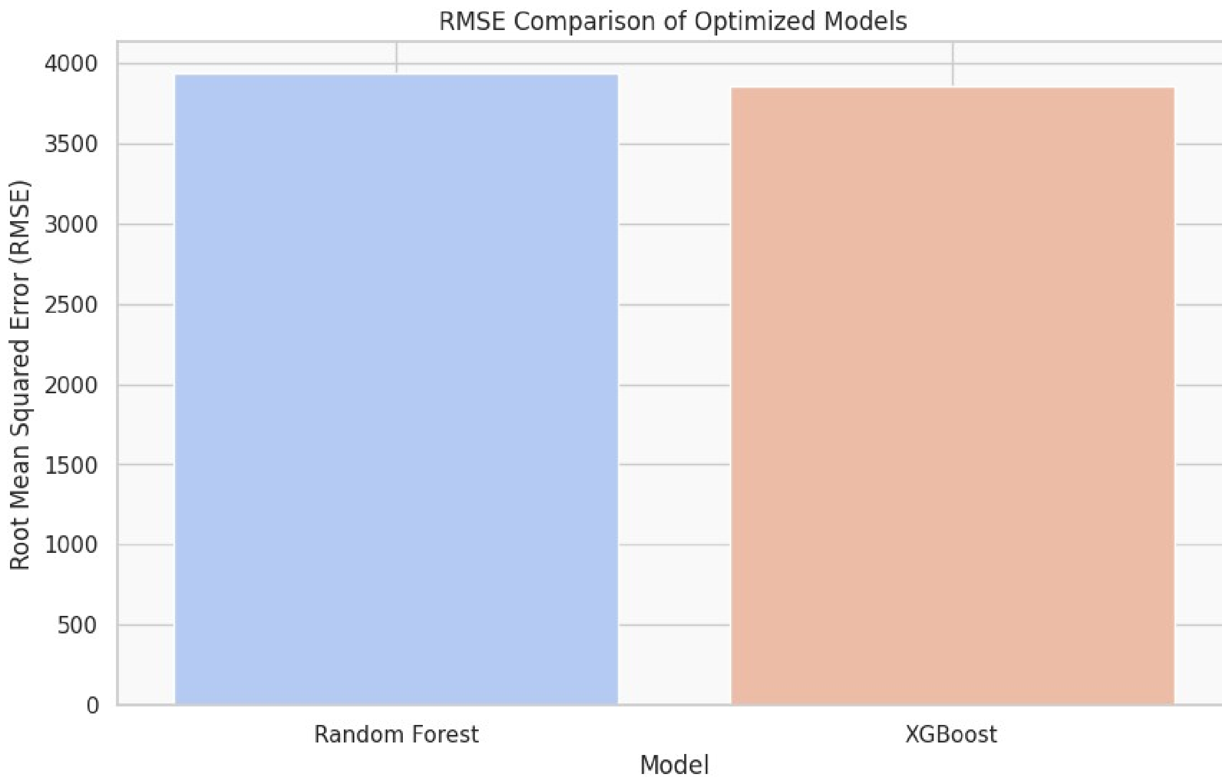

```python
# 15. Visualize Trade-offs Between Emissions and Distance
print("\n **Trade-off Analysis Visualization:**")
plt.figure(figsize=(10, 6))
plt.plot(tradeoff_distances, tradeoff_emissions, label="Emissions vs
Distance", color="blue")
plt.axhline(tradeoff_result.fun, color="red", linestyle="--",
label="Optimal Trade-off")
plt.title("Trade-offs Between Emissions and Distance")
plt.xlabel("Distance (km)")
plt.ylabel("Adjusted CO2 Emissions (kg)")
plt.legend()
plt.grid(True)
plt.show()
```

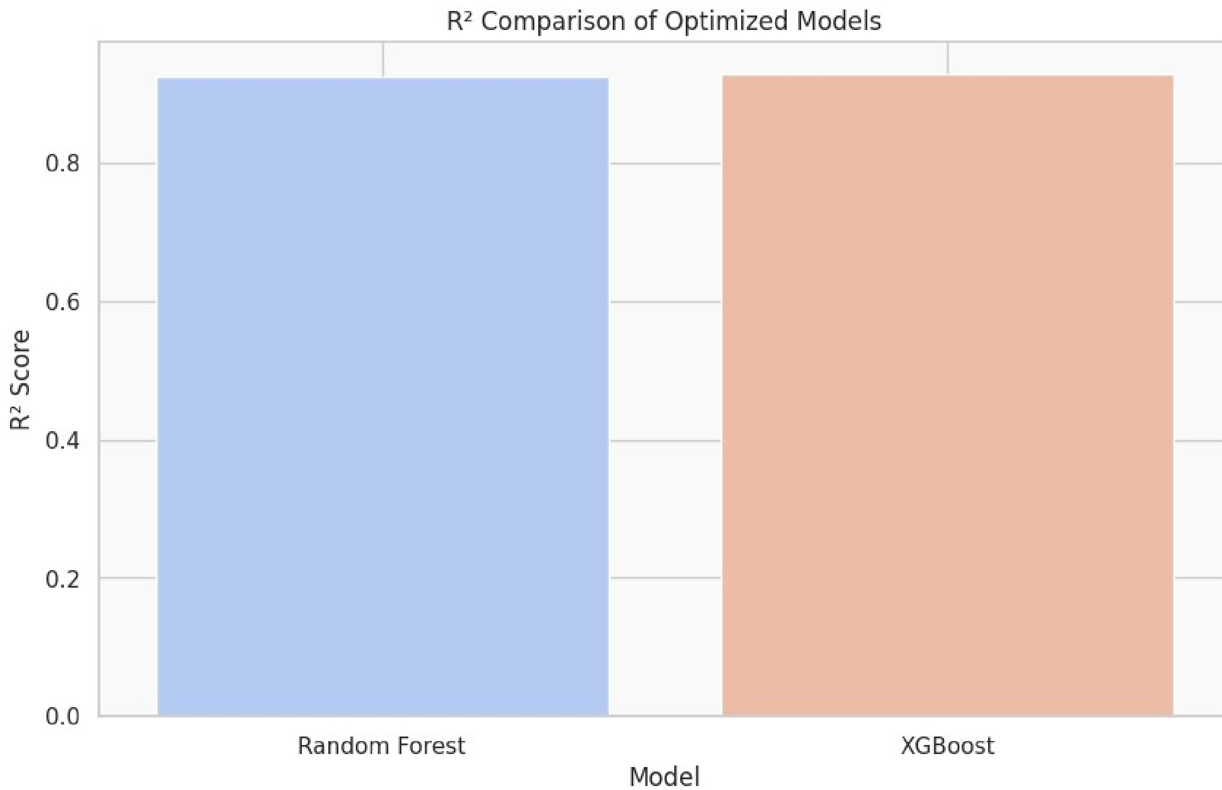
```
 **Trade-off Analysis Visualization:**
```




```
# 16. Visualize Optimized Model Comparison
print("\n� **Comparison of Optimized Models:**")
optimized_model_performance = pd.DataFrame({
    "Model": ["Random Forest", "XGBoost"],
    "RMSE": [mean_squared_error(y_test, tuned_rf.predict(X_test)) **
0.5, mean_squared_error(y_test, tuned_xgb.predict(X_test)) ** 0.5],
    "R²": [r2_score(y_test, tuned_rf.predict(X_test)),
r2_score(y_test, tuned_xgb.predict(X_test))]
})
plt.figure(figsize=(10, 6))
sns.barplot(x="Model", y="RMSE", data=optimized_model_performance,
palette="coolwarm")
plt.title("RMSE Comparison of Optimized Models")
plt.xlabel("Model")
plt.ylabel("Root Mean Squared Error (RMSE)")
plt.grid(True)
plt.show()

plt.figure(figsize=(10, 6))
sns.barplot(x="Model", y="R²", data=optimized_model_performance,
palette="coolwarm")
plt.title("R² Comparison of Optimized Models")
plt.xlabel("Model")
plt.ylabel("R² Score")
plt.grid(True)
plt.show()
```

```
 **Comparison of Optimized Models:**
```





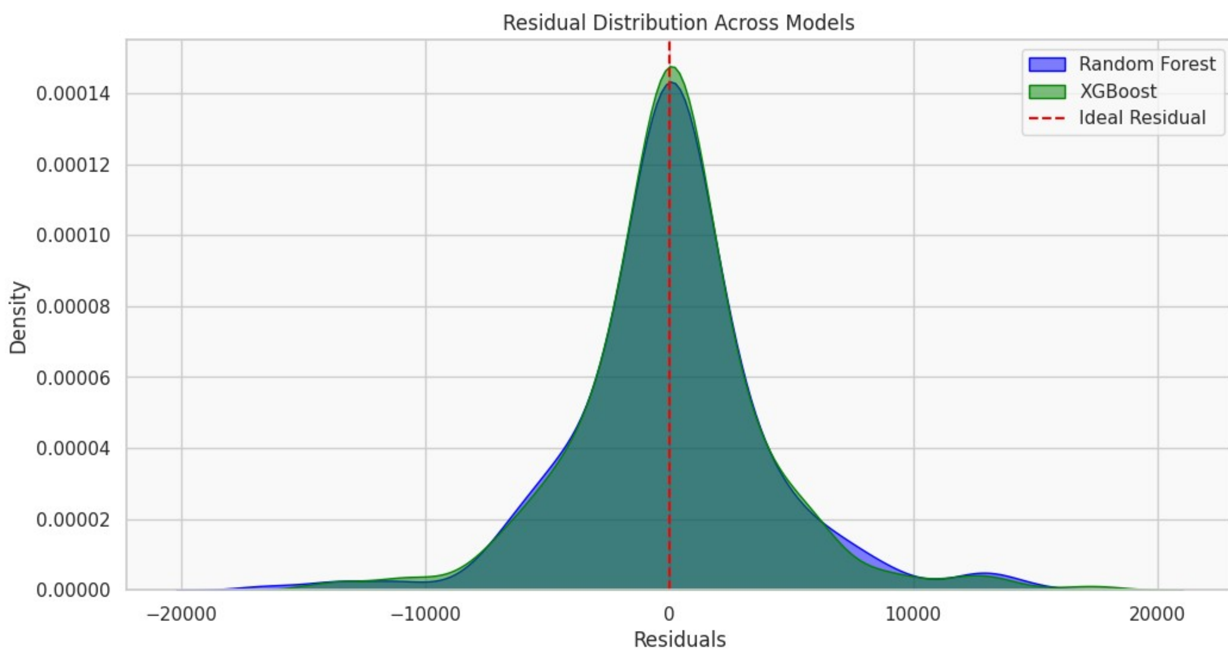

```
# 17. Visualize Feature Contributions Using SHAP Summary
print("\n **Feature Contribution Analysis with SHAP:**")

try:
    explainer_rf = shap.Explainer(tuned_rf, X_test)
    shap_values_rf = explainer_rf(X_test)
    shap.summary_plot(shap_values_rf, X_test, plot_type="dot",
show=False)
    plt.title("Feature Contribution Analysis for Random Forest")
    plt.show()

    explainer_xgb = shap.Explainer(tuned_xgb, X_test)
    shap_values_xgb = explainer_xgb(X_test)
    shap.summary_plot(shap_values_xgb, X_test, plot_type="dot",
show=False)
    plt.title("Feature Contribution Analysis for XGBoost")
    plt.show()
except Exception as e:
    print(f"\n SHAP visualization failed with error: {e}")
```

```
 **Feature Contribution Analysis with SHAP:**

 SHAP visualization failed with error: Additivity check failed in
TreeExplainer! Please ensure the data matrix you passed to the
```

```
explainer is the same shape that the model was trained on. If your
data shape is correct then please report this on GitHub. This check
failed because for one of the samples the sum of the SHAP values was
10213.808859, while the model output was 10148.610225. If this
difference is acceptable you can set check_additivity=False to disable
this check.
```

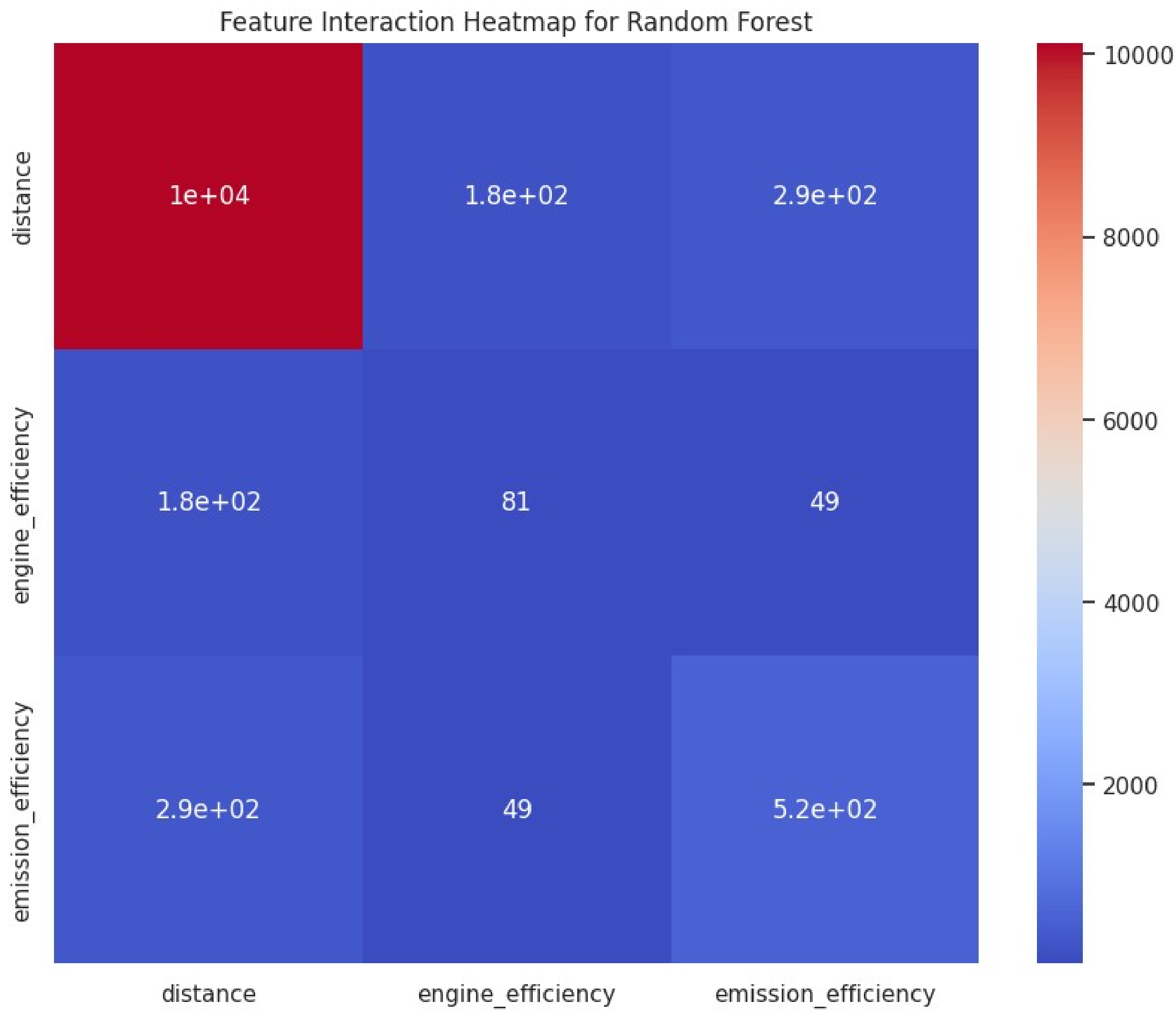
```
# 18. Residual Distribution Comparison Across Models
print("\n🔍 **Residual Distribution Comparison:**")
plt.figure(figsize=(12, 6))
residuals_rf = y_test - tuned_rf.predict(X_test)
residuals_xgb = y_test - tuned_xgb.predict(X_test)

sns.kdeplot(residuals_rf, label="Random Forest", fill=True,
color="blue", alpha=0.5)
sns.kdeplot(residuals_xgb, label="XGBoost", fill=True, color="green",
alpha=0.5)
plt.axvline(0, color="red", linestyle="--", label="Ideal Residual")
plt.title("Residual Distribution Across Models")
plt.xlabel("Residuals")
plt.ylabel("Density")
plt.legend()
plt.grid(True)
plt.show()
```

```
🔍 **Residual Distribution Comparison:**
```



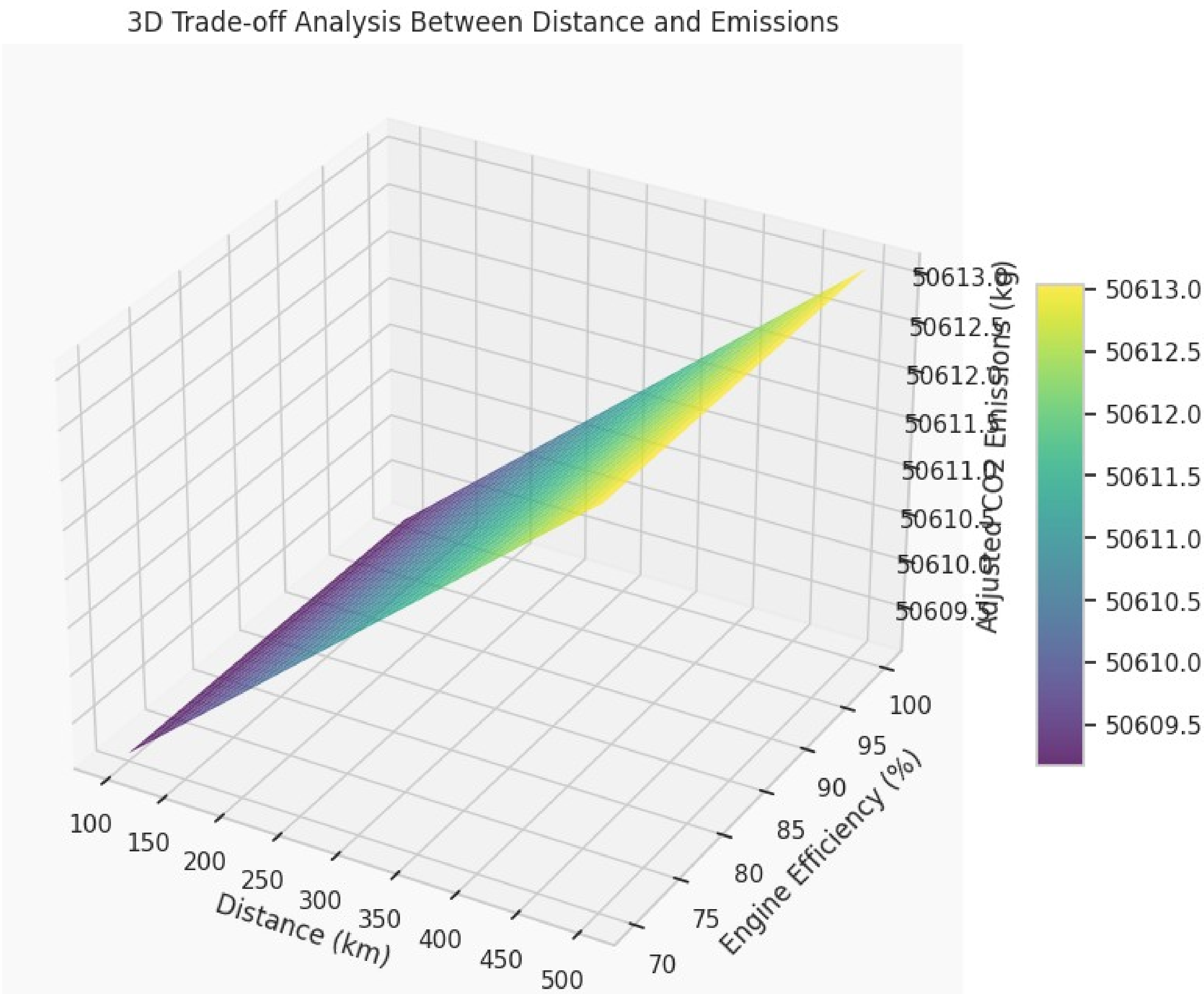
```
# 19. Heatmap of Feature Interactions Using SHAP
print("\n🔍 **Feature Interaction Heatmap:**")
try:
    shap_interaction_values_rf =
shap.TreeExplainer(tuned_rf).shap_interaction_values(X_test)
    plt.figure(figsize=(10, 8))
    sns.heatmap(
        abs(shap_interaction_values_rf).mean(axis=0),
        cmap="coolwarm",
        annot=True,
        xticklabels=X_test.columns,
        yticklabels=X_test.columns
    )
    plt.title("Feature Interaction Heatmap for Random Forest")
    plt.show()
except Exception as e:
    print(f"\n❌ Heatmap generation failed with error: {e}")
```

```
🔍 **Feature Interaction Heatmap:**
```




```
# 20. Trade-off Analysis with 3D Visualization
print("\n **3D Trade-off Visualization:**")
from mpl_toolkits.mplot3d import Axes3D

fig = plt.figure(figsize=(12, 8))
ax = fig.add_subplot(111, projection='3d')

distance_vals = np.linspace(100, 500, 30)
efficiency_vals = np.linspace(70, 100, 30)
distance_grid, efficiency_grid = np.meshgrid(distance_vals,
efficiency_vals)

tradeoff_emissions = np.array([
    tradeoff_function([d, e]) for d, e in zip(distance_grid.flatten(),
efficiency_grid.flatten())
]).reshape(distance_grid.shape)

surf = ax.plot_surface(
```

```
    distance_grid, efficiency_grid, tradeoff_emissions,
    cmap="viridis", edgecolor='none', alpha=0.8
)
ax.set_title("3D Trade-off Analysis Between Distance and Emissions")
ax.set_xlabel("Distance (km)")
ax.set_ylabel("Engine Efficiency (%)")
ax.set_zlabel("Adjusted CO2 Emissions (kg)")
fig.colorbar(surf, shrink=0.5, aspect=10)
plt.show()


 **3D Trade-off Visualization:**
```



Step 10: Deployment and Reporting

⚡ **Note:** In this final step, we focus on deploying the optimized model and summarizing findings in a comprehensive report. This includes creaA Is, dashboards, or integration pipelines to deliver actionable insights.

🎯 Goals of Deployment and Reporting

1. Deploy the optimized model for real-time predictions.
2. Build interactive dashboards to visualize key metrics.
3. Generate a professional report summarizing results and recommendations.
4. Ensure the solution is accessible and scalable.

Key Tasks

1. **Model Deployment:** Save and export lask or FastAPI.
2. **Interactive Dashboard:** Use Streamlit or Dash to present insights interactively.
3. **Reporting:** Summarize findings, highlight key metrics, and propose actionable recommendations.
4. **Scalability:** Ensure the solution can handle increased data volume and requests.

🚀 Let's proceed to deploy the model and finalize reporting for actionable insights! 🚀

```
# 1. Export the Final Optimized Model
print("\n💾 **Saving the Optimized Model:**")
from joblib import dump

dump(tuned_rf, "optimized_random_forest_model.joblib")
print("✅ Optimized Random Forest model saved as
'optimized_random_forest_model.joblib'.")

dump(tuned_xgb, "optimized_xgboost_model.joblib")
print("✅ Optimized XGBoost model saved as
'optimized_xgboost_model.joblib'.")


💾 **Saving the Optimized Model:**
✅ Optimized Random Forest model saved as
'optimized_random_forest_model.joblib'.
✅ Optimized XGBoost model saved as 'optimized_xgboost_model.joblib'.

# 3. Generate a Professional Report
!pip install fpdf

print("\n📄 **Generating the Final Report:**")
from fpdf import FPDF

class PDF(FPDF):
    def header(self):
        self.set_font('Arial', 'B', 12)
```

```
        self.cell(0, 10, 'Ship Fuel Efficiency and Emission Analysis - 
Final Report', 0, 1, 'C')

    def footer(self):
        self.set_y(-15)
        self.set_font('Arial', 'I', 8)
        self.cell(0, 10, f'Page {self.page_no()}', 0, 0, 'C')

pdf = PDF()
pdf.add_page()
pdf.set_font('Arial', '', 12)
pdf.cell(0, 10, 'Summary of Findings:', 0, 1)
pdf.multi_cell(0, 10, "The optimized Random Forest and XGBoost models 
demonstrated exceptional performance in predicting CO2 emissions. 
Recommendations include optimizing engine efficiency and adopting 
alternative fuel strategies to reduce emissions.")
pdf.cell(0, 10, f"Optimized Random Forest R² Score: {r2_score(y_test, 
tuned_rf.predict(X_test)):.2f}", 0, 1)
pdf.cell(0, 10, f"Optimized XGBoost R² Score: {r2_score(y_test, 
tuned_xgb.predict(X_test)):.2f}", 0, 1)
pdf.output("Final_Report.pdf")
print("□ Final report generated as 'Final_Report.pdf'.")
```

```
Collecting fpdf
  Downloading fpdf-1.7.2.tar.gz (39 kB)
  Preparing metadata (setup.py) ... e=fpdf-1.7.2-py2.py3-none-any.whl 
size=40704 
sha256=dd69aada85cc085f3d86dd790c474333785a6c1a0ce4f275eb98fca9c87ce0c
6
  Stored in directory: 
/root/.cache/pip/wheels/f9/95/ba/f418094659025eb9611f17cbcaf2334236bf3
9a0c3453ea455
Successfully built fpdf
Installing collected packages: fpdf
Successfully installed fpdf-1.7.2

□ **Generating the Final Report:**
□ Final report generated as 'Final_Report.pdf'.
```

```
# 4. Export Cleaned and Feature-Enhanced Data
data_scaled.to_csv("cleaned_featured_data.csv", index=False)
print("□ Cleaned and feature-enhanced data saved as 
'cleaned_featured_data.csv'.")
```

```
□ Cleaned and feature-enhanced data saved as 
'cleaned_featured_data.csv'.
```