Internship Report by Sid-San

~ SIDDHARTA KOMMU

Corporate Planning Headquarters Research and Development Department

Table Of Contents

List Of Figures

List Of Tables

1. Introduction

[1-1] Background on maritime decarbonization and the importance of energy efficiency in shipping

Maritime decarbonization means reducing the amount of greenhouse gases, like carbon dioxide (CO_2), that ships release into the air. Ships are important for moving goods around the world, but they also produce about 3% of all global greenhouse gas emissions. If shipping was a country, it would be the sixth largest polluter in the world.

Making ships more energy efficient is very important. Ships use a lot of fuel, and burning fuel creates CO_2. Fuel costs can be about half of a ship's total expenses. By using less fuel, ships can save money and help the environment at the same time.

The International Maritime Organization (IMO) is the main group that sets rules for ships to reduce emissions. The IMO wants the shipping industry to reach net-zero emissions by 2050. To help with this, the IMO has created rules and tools, like the Carbon Intensity Indicator (CII). These rules encourage ship owners to use better technology, improve how ships are operated, and use cleaner fuels.

In summary, reducing emissions from ships is important for protecting the environment and saving costs. Energy efficiency is a key part of making shipping cleaner and more sustainable for the future.

The CII is a regulation that started in January 2023 and applies to all large cargo ships, including RO-RO (roll-on/roll-off) vessels. It measures how much carbon dioxide (CO_2) a ship produces for each unit of cargo it carries over a certain distance. Ships must report their fuel use and distance travelled every year. Based on this information, each ship receives a rating from A (best) to E (worst).

If a ship gets a low rating, the owner must make a plan to improve its performance. These rules are becoming stricter each year, so ship owners and operators need to keep finding ways to use less fuel and lower their emissions. For RO-RO vessels, this means paying close attention to energy efficiency and making improvements to stay within the required standards. The CII regulation is an important step in making shipping more environmentally friendly and sustainable.

How does it work?

- Every year, ships must report their operational data to their flag Administration.
- The CII is calculated using the ship's reported fuel consumption and distance sailed.
- Ships receive a rating (A–E) that is recorded in their Ship Energy Efficiency Management Plan (SEEMP).
- If a ship gets a D rating for three years in a row, or an E rating once, it must make a plan to improve its performance and get approval from the authorities.

Impact:

- Ship owners and operators must monitor and improve their ships' energy efficiency.
- Ships with poor ratings may face extra requirements, like making and following a corrective action plan.
- The rules push the industry to adopt better technology and smarter operations, making shipping cleaner and more sustainable.

The main goal of this project is to create a tool that helps predict and improve the Carbon Intensity Indicator (CII) for a RO-RO ship using real operational data. Specifically, the project aims to:

- Use machine learning to predict the ship's annual CII score based on daily operational data such as fuel consumption, distance traveled, and other important features.
- Provide an easy-to-use interface where users can enter ship data and instantly see the predicted CII value and performance rating.
- Offer practical suggestions for improving the ship's energy efficiency, helping operators meet IMO regulations and reduce fuel costs.

- Support better decision-making for ship management and contribute to more sustainable and environmentally friendly shipping operations.

[1-2] Explanation of the Carbon Intensity Indicator (CII)

The Carbon Intensity Indicator (CII) is a rule made by the International Maritime Organization (IMO) to help reduce greenhouse gas emissions from ships. The CII shows how efficiently a ship moves cargo or passengers while producing carbon dioxide (CO_2). In simple terms, it measures how much CO_2 a ship emits for each unit of cargo it carries over one nautical mile.

Every year, ships over 5,000 gross tonnage (GT) that travel internationally must report their fuel use and distance sailed. The CII is calculated using the following formula:

CII = Annual CO_2 Emissions (grams) / (Cargo-carrying capacity × Distance sailed in nautical miles)

For most cargo ships, the cargo-carrying capacity is measured as deadweight tonnage, but for passenger ships and RO-RO vessels, it is often measured as gross tonnage. The CO_2 emissions are calculated from the fuel the ship uses during the year. This system allows different types and sizes of ships to be compared fairly.

The main goal of the CII rule is to make ships more energy efficient and to reduce emissions by at least 40% by 2030 compared to 2008 levels. The requirements will become stricter every year, so ships must keep improving their performance to stay compliant.

[1-3] Description of the CII Rating System

After the CII value is calculated for a ship, the ship is given a rating from A to E. This rating shows how well the ship is performing in terms of energy efficiency and emissions:

- **A:** Best performance (major superior)
- **B:** Good performance (minor superior)
- **C:** Acceptable performance (meets the standard)

- **D:** Below standard (minor inferior)
- **E:** Poor performance (inferior)

A ship must have at least a C rating to be considered compliant. If a ship receives a D rating for three years in a row, or an E rating for any year, the ship owner must make a corrective action plan to improve the ship's rating. This plan must be approved by authorities and included in the ship's management documents.

The CII rating is recorded in the ship's official documents and must be reported every year. The requirements for each rating will become stricter over time, making it necessary for ship owners and operators to keep working on energy-saving measures, better technology, and improved operations.

This rating system encourages ship owners to take action to reduce emissions and improve efficiency, helping the shipping industry move towards a cleaner and more sustainable future.

2. Annual CII Prediction Model- 1 (Purpose)

The main purpose of this project is to develop a practical tool that helps RO-RO ship operators and managers predict and optimize their vessel's Carbon Intensity Indicator (CII) using real operational data. As new IMO regulations require ships to report and improve their CII ratings, many companies face challenges in understanding what factors influence their score and how to take action before problems arise.

Currently, most ship operators only see their CII results after the fact, when the annual report is completed. This makes it difficult to identify operational issues early, test the impact of changes, or plan strategies for improvement. There is a clear need for a system that can use daily ship data to forecast future CII outcomes and provide actionable suggestions.

This project aims to solve these problems by:

- Building a data-driven, user-friendly tool that predicts a ship's annual CII score using daily operational records.
- Allowing users to see how different factors—such as speed, trim, or

propeller pitch—affect the CII and overall energy efficiency.
- Providing clear, practical recommendations for operational changes that can help maintain or improve CII ratings.
- Supporting compliance with IMO regulations and helping companies avoid penalties, reduce fuel costs, and operate more sustainably.

By making CII predictions and optimization suggestions accessible and easy to understand, this project supports better decision-making for ship operators, technical managers, and sustainability teams. The ultimate goal is to help the maritime industry move toward cleaner, more efficient, and regulation-compliant shipping.

3. Description of the Sample Data

Here is a sample text you can use to describe the dataset in your report, using simple English and referring to the sample data you provided:

Description of the Sample Data

For this project, we used detailed operational data collected from a RO-RO ship. The data was recorded every minute and includes many important measurements about the ship’s performance and environment. Each row in the dataset represents one minute of the ship’s operation.

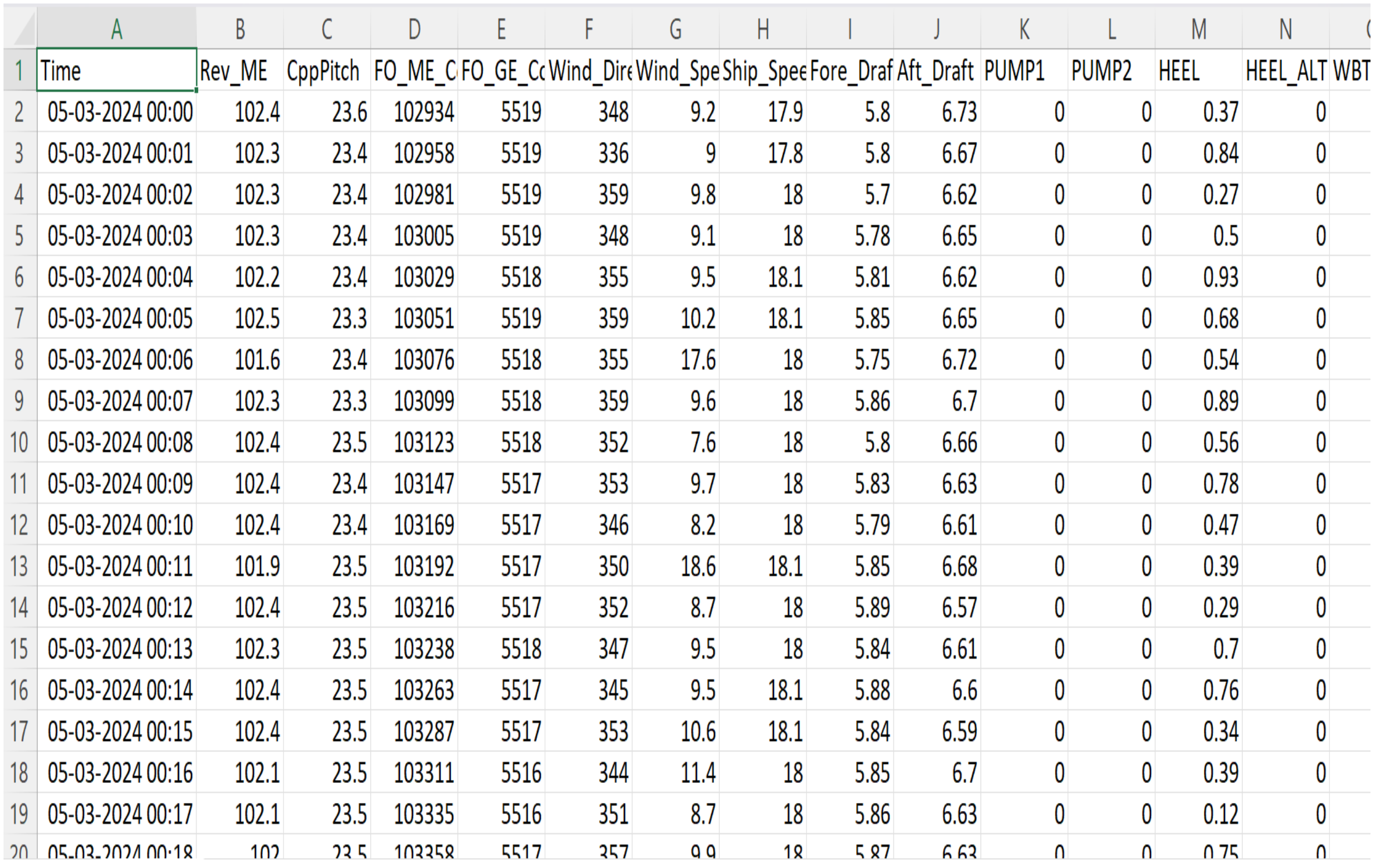
Some of the main columns in the sample data are:
- Time: The date and time when the data was recorded.
- Rev_ME: Main engine revolutions per minute, showing how fast the engine is running.
- CppPitch: The pitch angle of the controllable pitch propeller, which affects how the ship moves through water.
- FO_ME_Cons: The total amount of fuel oil consumed by the main engine, measured in liters.
- FO_GE_Cons: The total amount of fuel oil consumed by the generator engine, measured in liters.

- Wind_Direction: The direction from which the wind is blowing, in degrees.
- Wind_Speed: The speed of the wind, measured in meters per second.
- Ship_Speed: The speed of the ship, measured in knots.
- Fore_Draft: The depth of the ship's bow (front) below the waterline, in meters.
- Aft_Draft: The depth of the ship's stern (back) below the waterline, in meters.

The dataset contains many other columns, including sensor readings and status flags, but the columns listed above are the most important for predicting the ship's carbon intensity and energy efficiency.

Here is a sample of the data:

Table 1: RO-RO Sample Data



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	(
1	Time	Rev_ME	CppPitch	FO_ME_C	FO_GE_Cc	Wind_Dire	Wind_Spe	Ship_Spee	Fore_Draf	Aft_Draft	PUMP1	PUMP2	HEEL	HEEL_ALT	WBT
2	05-03-2024 00:00	102.4	23.6	102934	5519	348	9.2	17.9	5.8	6.73	0	0	0.37	0	
3	05-03-2024 00:01	102.3	23.4	102958	5519	336	9	17.8	5.8	6.67	0	0	0.84	0	
4	05-03-2024 00:02	102.3	23.4	102981	5519	359	9.8	18	5.7	6.62	0	0	0.27	0	
5	05-03-2024 00:03	102.3	23.4	103005	5519	348	9.1	18	5.78	6.65	0	0	0.5	0	
6	05-03-2024 00:04	102.2	23.4	103029	5518	355	9.5	18.1	5.81	6.62	0	0	0.93	0	
7	05-03-2024 00:05	102.5	23.3	103051	5519	359	10.2	18.1	5.85	6.65	0	0	0.68	0	
8	05-03-2024 00:06	101.6	23.4	103076	5518	355	17.6	18	5.75	6.72	0	0	0.54	0	
9	05-03-2024 00:07	102.3	23.3	103099	5518	359	9.6	18	5.86	6.7	0	0	0.89	0	
10	05-03-2024 00:08	102.4	23.5	103123	5518	352	7.6	18	5.8	6.66	0	0	0.56	0	
11	05-03-2024 00:09	102.4	23.4	103147	5517	353	9.7	18	5.83	6.63	0	0	0.78	0	
12	05-03-2024 00:10	102.4	23.4	103169	5517	346	8.2	18	5.79	6.61	0	0	0.47	0	
13	05-03-2024 00:11	101.9	23.5	103192	5517	350	18.6	18.1	5.85	6.68	0	0	0.39	0	
14	05-03-2024 00:12	102.4	23.5	103216	5517	352	8.7	18	5.89	6.57	0	0	0.29	0	
15	05-03-2024 00:13	102.3	23.5	103238	5518	347	9.5	18	5.84	6.61	0	0	0.7	0	
16	05-03-2024 00:14	102.4	23.5	103263	5517	345	9.5	18.1	5.88	6.6	0	0	0.76	0	
17	05-03-2024 00:15	102.4	23.5	103287	5517	353	10.6	18.1	5.84	6.59	0	0	0.34	0	
18	05-03-2024 00:16	102.1	23.5	103311	5516	344	11.4	18	5.85	6.7	0	0	0.39	0	
19	05-03-2024 00:17	102.1	23.5	103335	5516	351	8.7	18	5.86	6.63	0	0	0.12	0	
20	05-03-2024 00:18	102	23.5	103358	5517	357	9.9	18	5.87	6.63	0	0	0.75	0	

This data allows us to analyze how different factors, like engine speed, propeller pitch, and wind conditions, affect the ship's fuel consumption and emissions. By using this information, we can build a model to predict the ship's Carbon Intensity Indicator (CII) and suggest ways to improve its energy efficiency.

4. Data Preprocessing

[4-1] Handling Date and Time Information

After loading the data, the first step was to make sure the "Time" column was in the correct format. At first, the "Time" column was just plain text, which makes it hard to work with when analysing time-based patterns.

To fix this, we converted the "Time" column into a proper date and time format using pandas. This allows us to easily sort the data by time and perform time-based calculations later on. We then sorted the entire dataset in order of time, so all the records are in the correct sequence.

Code Example:

```
df['Time'] = pd.to_datetime(df['Time'])
df = df.sort_values('Time')
```

By doing this, we made sure that all the data is organized in the right order and ready for further analysis. This step is important for working with time series data, as it helps us find trends and patterns over time.

[4-2] Handling Continuous Fuel Consumption Records

The dataset includes columns for the amount of fuel used by the main engine (FO_ME_Cons) and the generator engine (FO_GE_Cons). Sometimes, these columns may have missing values or non-numeric entries, which can cause problems for analysis.

To solve this, we performed two actions:

1. Convert to Numeric:

   We made sure all values in these columns are numbers. If any value could not be converted (for example, if it was missing or not a number), it was set as missing.

2. Forward-Filling Missing Values:

   For any missing values, we used the last available value to fill the gap. This method, called "forward-filling," assumes that the fuel consumption continues at the same rate until a new value is recorded. This is a reasonable approach because fuel consumption usually changes gradually.

Code Example:

```
for col in ['FO_ME_Cons', 'FO_GE_Cons']:
    df[col] = pd.to_numeric(df[col], errors='coerce').fillna(method='ffill')
```

By doing this, we made sure the fuel consumption data is complete and ready for further analysis. This step is important because accurate fuel data is needed to calculate the ship’s CO_2 emissions.

[4-3] Filling Missing Values in Other Numeric Data

Besides fuel consumption, there are other important columns in the data, such as Ship_Speed, CppPitch (propeller pitch), Wind_Speed, HEEL, Fore_Draft, and Aft_Draft. Sometimes, these columns also have missing or invalid values. To handle this, we did the following:

1. Convert to Numeric:
   We changed all the values in these columns to numbers. If a value could not be converted (for example, if it was missing or written as text), it was marked as missing.
2. Forward-Filling Missing Values:
   For any missing values, we used the last recorded value to fill in the gap. This method is called "forward-filling." It works well here because most of these measurements (like ship speed or draft) do not change suddenly, so it is reasonable to assume the value stays the same until the next reading.

Code Example:

```
numeric_cols = ['Ship_Speed', 'CppPitch', 'Wind_Speed', 'HEEL', 'Fore_Draft',
'Aft_Draft']
for col in numeric_cols:
        df[col] = pd.to_numeric(df[col], errors='coerce').fillna(method='ffill')
```

By doing this, we made sure that all the important numeric data is complete and ready to use for further analysis and modeling. This helps improve the quality and accuracy of our predictions.

[4-4] Removing Rows with Missing Critical Data

After filling missing values, we checked if there were still any rows with missing information in the most important columns. These columns include ship speed, propeller pitch, wind speed, heel, fore draft, aft draft, and both fuel consumption columns. If any of these were still missing, we removed those rows from the dataset. This step is important because missing data in these key columns can affect the accuracy of our analysis and predictions.

Code Example:

```
df = df.dropna(subset=numeric_cols + ['FO_ME_Cons', 'FO_GE_Cons'])
```

[4-5] Filtering Out Idle or Port States

We only want to analyze data when the ship is actually sailing, not when it is stopped or in port. To do this, we removed all rows where the ship’s speed was less than or equal to 0.5 knots. By focusing only on periods when the ship is moving, we make sure our model learns about real voyage conditions, which are most important for predicting energy efficiency and CII.

Code Example:

```
df = df[df['Ship_Speed'] > 0.5]
```

By following these steps, we made sure our dataset is clean, complete, and focused on the ship’s active operations at sea. This helps us build a more reliable and accurate prediction model.

5. Feature Engineering and Calculations

To prepare the data for modeling, we created some new features that are important for predicting the ship’s carbon intensity and energy efficiency:

[5-1] Calculating Fuel Consumption per Minute

The original dataset gives the total fuel used by the main engine (FO_ME_Cons) and the generator engine (FO_GE_Cons) as running totals. To find out how much fuel the ship used each minute, we calculated the difference between each row and the previous row for both columns.

- If the difference was negative (which can happen due to data resets), we set it to zero to avoid errors.
- We then added the main engine and generator fuel consumption together to get the total fuel used per minute.

Code Example:

```
fuel_me = df['FO_ME_Cons'].diff().clip(lower=0)
fuel_ge = df['FO_GE_Cons'].diff().clip(lower=0)
df['Fuel_Liters'] = fuel_me + fuel_ge
```

[5-2] Calculating Trim

Trim is the difference between the draft at the back (Aft_Draft) and the draft at the front (Fore_Draft) of the ship. Trim can affect how efficiently the ship moves through the water.

Code Example:

```
df['Trim'] = df['Aft_Draft'] - df['Fore_Draft']
```

[5-3] Calculating Average Draft

Average draft is the mean value of the fore and aft drafts. This gives an overall idea of how deep the ship sits in the water, which also affects fuel use and efficiency.

Code Example:

```
df['Avg_Draft'] = (df['Aft_Draft'] + df['Fore_Draft']) / 2
```

By creating these new features, we made the data more useful for our machine learning model. These features help us better understand the ship’s condition and how it relates to fuel consumption and emissions.

[5-4] Saving the Cleaned Data

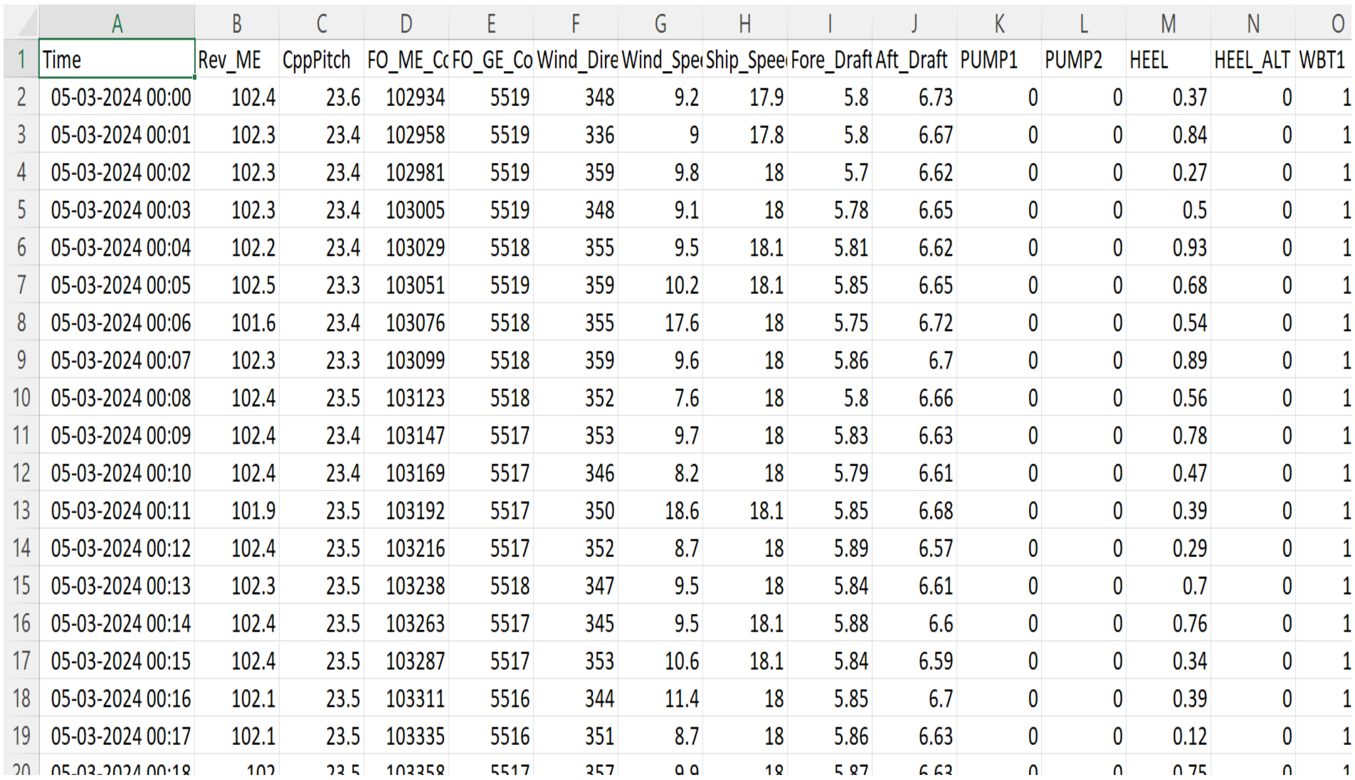
After finishing all the cleaning and feature engineering steps, we saved the

processed data to a new file called **"Cleaned_RO-RO_Data.csv"**. This file contains only the useful and cleaned columns, with all missing values handled and new features added.
Saving the cleaned data makes it easy to use for further analysis and model training. It also helps keep a record of the data preparation process.
Code Example:

```
df.to_csv("Cleaned_RO-RO_Data.csv", index=False)
```

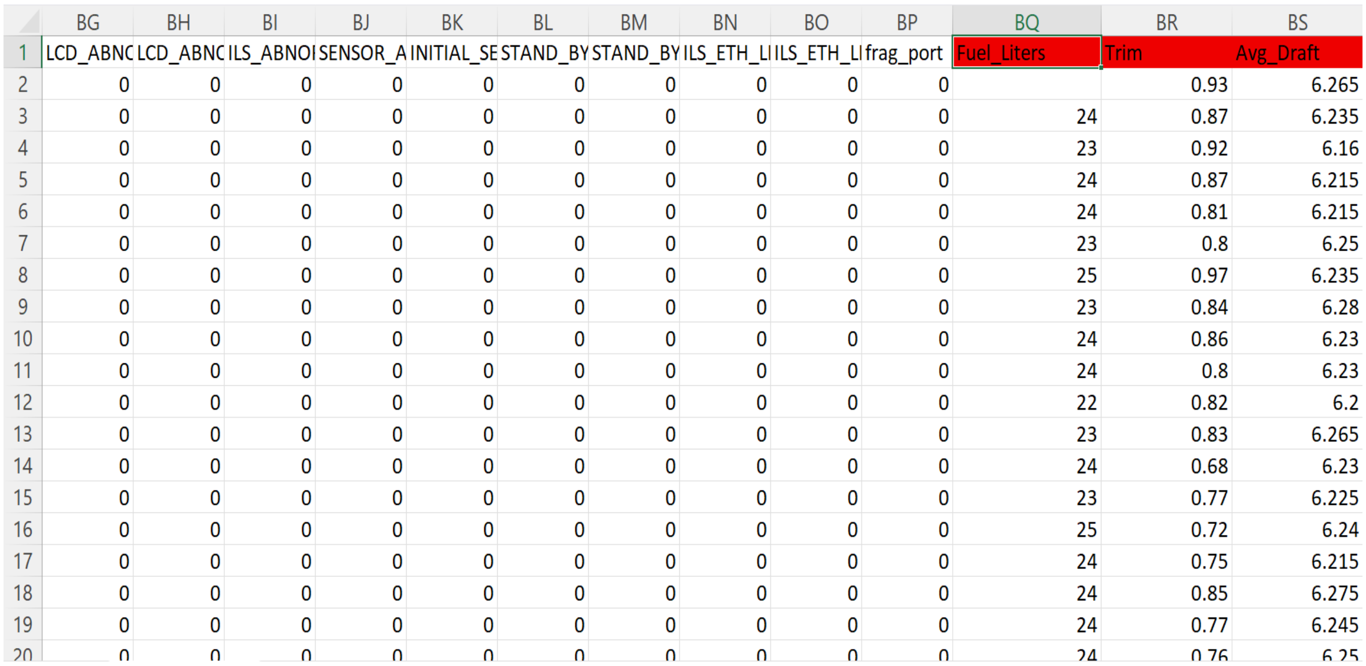
The first few rows of the cleaned data is shown below:

Table 2 - Cleaned RO-RO Data



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
1	Time	Rev_ME	CppPitch	FO_ME_Cc	FO_GE_Co	Wind_Dire	Wind_Spe	Ship_Spee	Fore_Draft	Aft_Draft	PUMP1	PUMP2	HEEL	HEEL_ALT	WBT1
2	05-03-2024 00:00	102.4	23.6	102934	5519	348	9.2	17.9	5.8	6.73	0	0	0.37	0	1
3	05-03-2024 00:01	102.3	23.4	102958	5519	336	9	17.8	5.8	6.67	0	0	0.84	0	1
4	05-03-2024 00:02	102.3	23.4	102981	5519	359	9.8	18	5.7	6.62	0	0	0.27	0	1
5	05-03-2024 00:03	102.3	23.4	103005	5519	348	9.1	18	5.78	6.65	0	0	0.5	0	1
6	05-03-2024 00:04	102.2	23.4	103029	5518	355	9.5	18.1	5.81	6.62	0	0	0.93	0	1
7	05-03-2024 00:05	102.5	23.3	103051	5519	359	10.2	18.1	5.85	6.65	0	0	0.68	0	1
8	05-03-2024 00:06	101.6	23.4	103076	5518	355	17.6	18	5.75	6.72	0	0	0.54	0	1
9	05-03-2024 00:07	102.3	23.3	103099	5518	359	9.6	18	5.86	6.7	0	0	0.89	0	1
10	05-03-2024 00:08	102.4	23.5	103123	5518	352	7.6	18	5.8	6.66	0	0	0.56	0	1
11	05-03-2024 00:09	102.4	23.4	103147	5517	353	9.7	18	5.83	6.63	0	0	0.78	0	1
12	05-03-2024 00:10	102.4	23.4	103169	5517	346	8.2	18	5.79	6.61	0	0	0.47	0	1
13	05-03-2024 00:11	101.9	23.5	103192	5517	350	18.6	18.1	5.85	6.68	0	0	0.39	0	1
14	05-03-2024 00:12	102.4	23.5	103216	5517	352	8.7	18	5.89	6.57	0	0	0.29	0	1
15	05-03-2024 00:13	102.3	23.5	103238	5518	347	9.5	18	5.84	6.61	0	0	0.7	0	1
16	05-03-2024 00:14	102.4	23.5	103263	5517	345	9.5	18.1	5.88	6.6	0	0	0.76	0	1
17	05-03-2024 00:15	102.4	23.5	103287	5517	353	10.6	18.1	5.84	6.59	0	0	0.34	0	1
18	05-03-2024 00:16	102.1	23.5	103311	5516	344	11.4	18	5.85	6.7	0	0	0.39	0	1
19	05-03-2024 00:17	102.1	23.5	103335	5516	351	8.7	18	5.86	6.63	0	0	0.12	0	1
20	05-03-2024 00:18	102	23.5	103358	5517	357	9.9	18	5.87	6.63	0	0	0.75	0	1

Table 3 – Cleaned RO-RO Data [Fuel, Trim, Avg_Draft]



	BG	BH	BI	BJ	BK	BL	BM	BN	BO	BP	BQ	BR	BS
1	LCD_ABNC	LCD_ABNC	ILS_ABNOI	SENSOR_A	INITIAL_SE	STAND_BY	STAND_BY	ILS_ETH_LI	ILS_ETH_LI	frag_port	Fuel_Liters	Trim	Avg_Draft
2	0	0	0	0	0	0	0	0	0	0		0.93	6.265
3	0	0	0	0	0	0	0	0	0	0	24	0.87	6.235
4	0	0	0	0	0	0	0	0	0	0	23	0.92	6.16
5	0	0	0	0	0	0	0	0	0	0	24	0.87	6.215
6	0	0	0	0	0	0	0	0	0	0	24	0.81	6.215
7	0	0	0	0	0	0	0	0	0	0	23	0.8	6.25
8	0	0	0	0	0	0	0	0	0	0	25	0.97	6.235
9	0	0	0	0	0	0	0	0	0	0	23	0.84	6.28
10	0	0	0	0	0	0	0	0	0	0	24	0.86	6.23
11	0	0	0	0	0	0	0	0	0	0	24	0.8	6.23
12	0	0	0	0	0	0	0	0	0	0	22	0.82	6.2
13	0	0	0	0	0	0	0	0	0	0	23	0.83	6.265
14	0	0	0	0	0	0	0	0	0	0	24	0.68	6.23
15	0	0	0	0	0	0	0	0	0	0	23	0.77	6.225
16	0	0	0	0	0	0	0	0	0	0	25	0.72	6.24
17	0	0	0	0	0	0	0	0	0	0	24	0.75	6.215
18	0	0	0	0	0	0	0	0	0	0	24	0.85	6.275
19	0	0	0	0	0	0	0	0	0	0	24	0.77	6.245
20	0	0	0	0	0	0	0	0	0	0	24	0.76	6.25

This cleaned dataset is now ready to be used for building and testing the CII prediction model.

[5-5] hip and Fuel Information

For our calculations, we used important information about the ship and its fuel, based on data provided by the company and international standards:

- **Gross Tonnage (GT):**

  The gross tonnage of the RO-RO ship is **14,052**. This value was provided by Tsuruta San. Gross tonnage is a measure of the ship's overall size and is used in the CII calculation.

- **Fuel Density:**

  The density of heavy fuel oil (HFO) is **0.991 kg/L**. This value comes from the International Maritime Organization (IMO) guidelines. Fuel density is needed to convert fuel volume (liters) into mass (kilograms).

- **CO_2 Emission Factor:**

  The emission factor for HFO is **3.114 grams of CO_2 per gram of4 fuel**. This is also based on IMO standards. It tells us how much carbon dioxide is produced for each gram of fuel burned.

These values are used in the next steps to calculate the ship’s total CO_2 emissions and, finally, the Carbon Intensity Indicator (CII).

6. Calculating CO_2 Emissions and Distance Traveled

After preparing the data and setting the ship and fuel constants, we calculated two important values for each minute:

[6-1] CO_2 Emissions (grams per minute)

First, we converted the amount of fuel used from liters to kilograms by multiplying by the fuel density (0.991 kg/L).

Then, we calculated the total CO_2 emissions by multiplying the fuel mass (in kg), converting it to grams (×1000), and multiplying by the CO_2 emission factor (3.114).

This gives us the amount of CO_2 produced each minute in grams.

[6-2] Distance Traveled (nautical miles per minute)

The ship’s speed is recorded in knots, which means nautical miles per hour. To find out how far the ship travels in one minute, we divided the speed by 60.

Code Example:

```
fuel_kg = df['Fuel_Liters'] * FUEL_DENSITY
df['CO2_Emission_g'] = fuel_kg * 1000 * CO2_FACTOR   # convert kg to grams
df['Distance_NM'] = df['Ship_Speed'] / 60   # NM per minute
```

These calculations are important because they give us the basic numbers needed to work out the ship’s total annual CO_2 emissions and total distance sailed, which are both used in the CII formula.

7. Data Aggregation: Monthly, Weekly, and Daily Summaries

After calculating the CO_2 emissions and distance for each minute, the next step was to summarize the data over longer time periods. This helps us see trends and makes the data easier to use for machine learning.

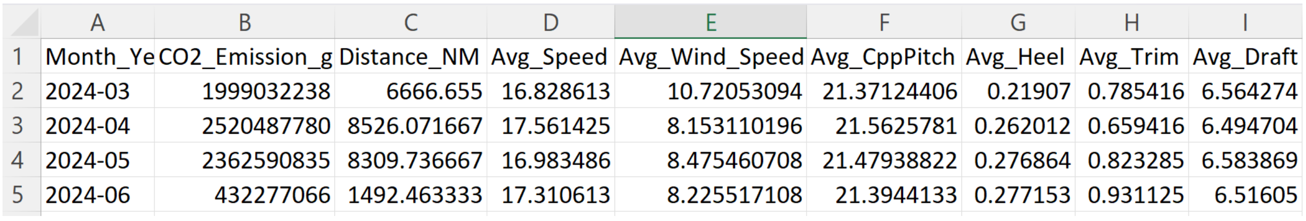
[7-1] Monthly Aggregation

First, we grouped the data by month. For each month, we calculated:

- The total CO₂ emissions
- The total distance sailed
- The average values for ship speed, wind speed, propeller pitch, heel, trim, and average draft

We saved this monthly summary to a file called monthly_summary.csv.

Table 4 - Sample Of Montly Data



	A	B	C	D	E	F	G	H	I
1	Month_Ye	CO2_Emission_g	Distance_NM	Avg_Speed	Avg_Wind_Speed	Avg_CppPitch	Avg_Heel	Avg_Trim	Avg_Draft
2	2024-03	1999032238	6666.655	16.828613	10.72053094	21.37124406	0.21907	0.785416	6.564274
3	2024-04	2520487780	8526.071667	17.561425	8.153110196	21.5625781	0.262012	0.659416	6.494704
4	2024-05	2362590835	8309.736667	16.983486	8.475460708	21.47938822	0.276864	0.823285	6.583869
5	2024-06	432277066	1492.463333	17.310613	8.225517108	21.3944133	0.277153	0.931125	6.51605

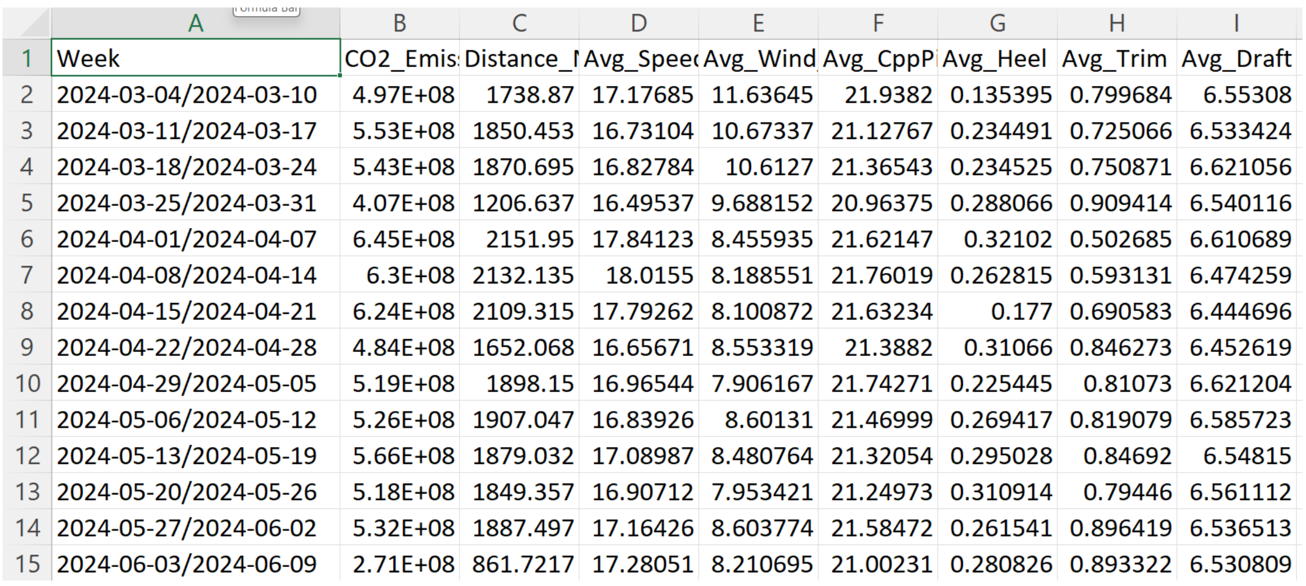
However, since we only had data for four months, this gave us just four data points. This is not enough for training a good machine learning model, because the model would not have enough examples to learn from.

[7-2] Weekly Aggregation

Next, we grouped the data by week. For each week, we calculated the same totals and averages as above.

We saved this weekly summary to a file called weekly_summary.csv.

Table 5- Sample Of Weekly Data



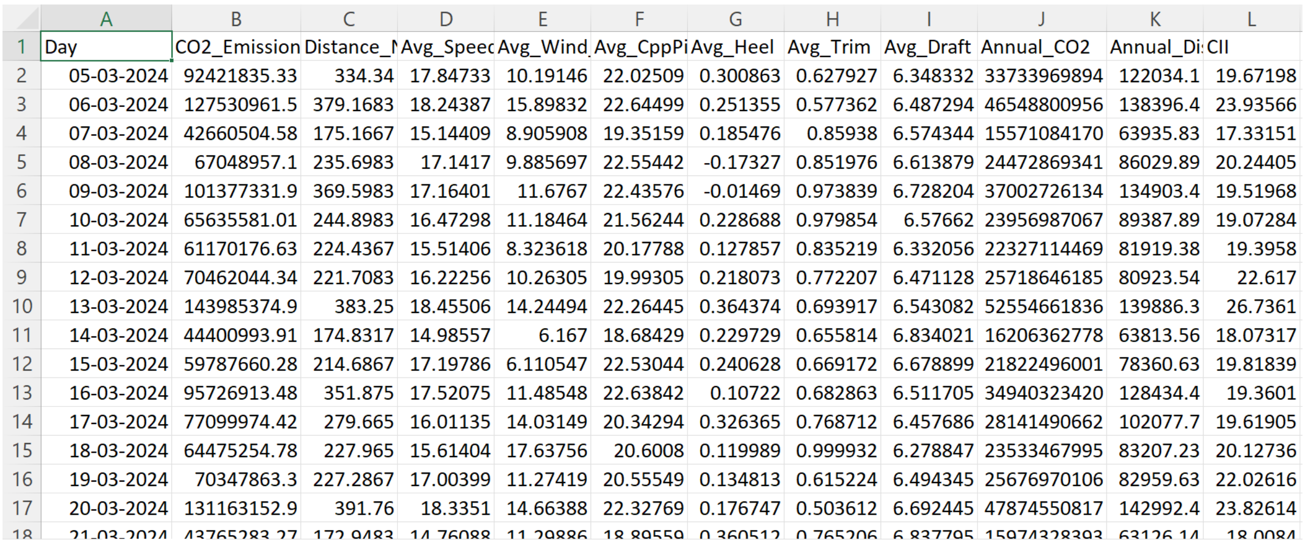
	A	B	C	D	E	F	G	H	I
1	Week	CO2_Emis	Distance_I	Avg_Speec	Avg_Wind	Avg_CppP	Avg_Heel	Avg_Trim	Avg_Draft
2	2024-03-04/2024-03-10	4.97E+08	1738.87	17.17685	11.63645	21.9382	0.135395	0.799684	6.55308
3	2024-03-11/2024-03-17	5.53E+08	1850.453	16.73104	10.67337	21.12767	0.234491	0.725066	6.533424
4	2024-03-18/2024-03-24	5.43E+08	1870.695	16.82784	10.6127	21.36543	0.234525	0.750871	6.621056
5	2024-03-25/2024-03-31	4.07E+08	1206.637	16.49537	9.688152	20.96375	0.288066	0.909414	6.540116
6	2024-04-01/2024-04-07	6.45E+08	2151.95	17.84123	8.455935	21.62147	0.32102	0.502685	6.610689
7	2024-04-08/2024-04-14	6.3E+08	2132.135	18.0155	8.188551	21.76019	0.262815	0.593131	6.474259
8	2024-04-15/2024-04-21	6.24E+08	2109.315	17.79262	8.100872	21.63234	0.177	0.690583	6.444696
9	2024-04-22/2024-04-28	4.84E+08	1652.068	16.65671	8.553319	21.3882	0.31066	0.846273	6.452619
10	2024-04-29/2024-05-05	5.19E+08	1898.15	16.96544	7.906167	21.74271	0.225445	0.81073	6.621204
11	2024-05-06/2024-05-12	5.26E+08	1907.047	16.83926	8.60131	21.46999	0.269417	0.819079	6.585723
12	2024-05-13/2024-05-19	5.66E+08	1879.032	17.08987	8.480764	21.32054	0.295028	0.84692	6.54815
13	2024-05-20/2024-05-26	5.18E+08	1849.357	16.90712	7.953421	21.24973	0.310914	0.79446	6.561112
14	2024-05-27/2024-06-02	5.32E+08	1887.497	17.16426	8.603774	21.58472	0.261541	0.896419	6.536513
15	2024-06-03/2024-06-09	2.71E+08	861.7217	17.28051	8.210695	21.00231	0.280826	0.893322	6.530809

With weekly aggregation, we had about 16 data points. This is better than monthly, but still a small amount for training a reliable model. Weekly data can be useful for some analysis, but it is still limited.

[7-3] Daily Aggregation

Finally, we grouped the data by day. For each day, we again calculated the total CO_2 emissions, total distance, and average values for other important features. We saved this daily summary to a file called daily_summary.csv.

Table 6 - Sample Of Daily Data



	A	B	C	D	E	F	G	H	I	J	K	L
1	Day	CO2_Emission	Distance_N	Avg_Speec	Avg_Wind_	Avg_CppPi	Avg_Heel	Avg_Trim	Avg_Draft	Annual_CO2	Annual_Di:	CII
2	05-03-2024	92421835.33	334.34	17.84733	10.19146	22.02509	0.300863	0.627927	6.348332	33733969894	122034.1	19.67198
3	06-03-2024	127530961.5	379.1683	18.24387	15.89832	22.64499	0.251355	0.577362	6.487294	46548800956	138396.4	23.93566
4	07-03-2024	42660504.58	175.1667	15.14409	8.905908	19.35159	0.185476	0.85938	6.574344	15571084170	63935.83	17.33151
5	08-03-2024	67048957.1	235.6983	17.1417	9.885697	22.55442	-0.17327	0.851976	6.613879	24472869341	86029.89	20.24405
6	09-03-2024	101377331.9	369.5983	17.16401	11.6767	22.43576	-0.01469	0.973839	6.728204	37002726134	134903.4	19.51968
7	10-03-2024	65635581.01	244.8983	16.47298	11.18464	21.56244	0.228688	0.979854	6.57662	23956987067	89387.89	19.07284
8	11-03-2024	61170176.63	224.4367	15.51406	8.323618	20.17788	0.127857	0.835219	6.332056	22327114469	81919.38	19.3958
9	12-03-2024	70462044.34	221.7083	16.22256	10.26305	19.99305	0.218073	0.772207	6.471128	25718646185	80923.54	22.617
10	13-03-2024	143985374.9	383.25	18.45506	14.24494	22.26445	0.364374	0.693917	6.543082	52554661836	139886.3	26.7361
11	14-03-2024	44400993.91	174.8317	14.98557	6.167	18.68429	0.229729	0.655814	6.834021	16206362778	63813.56	18.07317
12	15-03-2024	59787660.28	214.6867	17.19786	6.110547	22.53044	0.240628	0.669172	6.678899	21822496001	78360.63	19.81839
13	16-03-2024	95726913.48	351.875	17.52075	11.48548	22.63842	0.10722	0.682863	6.511705	34940323420	128434.4	19.3601
14	17-03-2024	77099974.42	279.665	16.01135	14.03149	20.34294	0.326365	0.768712	6.457686	28141490662	102077.7	19.61905
15	18-03-2024	64475254.78	227.965	15.61404	17.63756	20.6008	0.119989	0.999932	6.278847	23533467995	83207.23	20.12736
16	19-03-2024	70347863.3	227.2867	17.00399	11.27419	20.55549	0.134813	0.615224	6.494345	25676970106	82959.63	22.02616
17	20-03-2024	131163152.9	391.76	18.3351	14.66388	22.32769	0.176747	0.503612	6.692445	47874550817	142992.4	23.82614
18	21-03-2024	43765283.27	172.9483	14.76088	11.29886	18.89559	0.360512	0.765206	6.837795	15974328393	63126.14	18.0084

By using daily data, we had many more data points to work with. This makes it much better for training a machine learning model, as the model can learn from more examples and provide more accurate predictions.

Conclusion:

After comparing the three options, we chose to use the daily aggregated data for model training. This approach gave us the best balance between detail and having enough data points for machine learning.

8. Calculating Projected Annual Emissions, Distance, and CII

After creating the daily summary data, we needed to estimate the ship's performance over a full year. This allows us to predict the annual Carbon Intensity Indicator (CII) using the IMO formula.

[8-1] Projecting Annual Emissions and Distance

For each day, we calculated the projected annual CO_2 emissions and annual distance by multiplying the daily totals by 365 (the number of days in a year). This gives an estimate of what the ship's emissions and distance would be if it operated in a similar way every day for a year.

- **Annual CO_2 Emissions:**

Annual_CO2=Daily CO2 Emission (g)×365Annual_CO2=Daily CO2 Emission (g)×365

- **Annual Distance:**

Annual_Dist=Daily Distance (NM)×365Annual_Dist=Daily Distance (NM)×365

[8-2] Calculating CII Using the IMO Formula

We then used the IMO formula to calculate the CII for each day's data:

- **CII Formula:**

CII=Annual CO2 Emissions (g)Gross Tonnage×Annual Distance (NM)CII=Gross Tonnage×Annual Distance (NM)Annual CO2 Emissions (g)

This formula shows how much CO_2 the ship emits per unit of cargo-carrying capacity for each nautical mile traveled over the year.

Code Example:

python

```python
daily_df['Annual_CO2'] = daily_df['CO2_Emission_g'] * 365
daily_df['Annual_Dist'] = daily_df['Distance_NM'] * 365
daily_df['CII'] = daily_df['Annual_CO2'] / (GT * daily_df['Annual_Dist'])
daily_df.to_csv('daily_summary.csv', index=False)
```

By calculating these values, we created a dataset that can be used to train and test our machine learning model to predict the ship's annual CII based on daily operational data.

[8-3] Daily Fuel Consumption Trend

To better understand how the ship's fuel usage changes over time, we created a bar chart showing the total fuel consumed each day.

How we made the plot:

- We extracted the date from the timestamp for each record.
- We calculated the total amount of fuel used for each day by adding up all the fuel consumption values for that day.
- We merged this daily fuel total with our main daily summary data.
- We then plotted a bar chart, where each bar shows the total fuel consumed on a specific day.

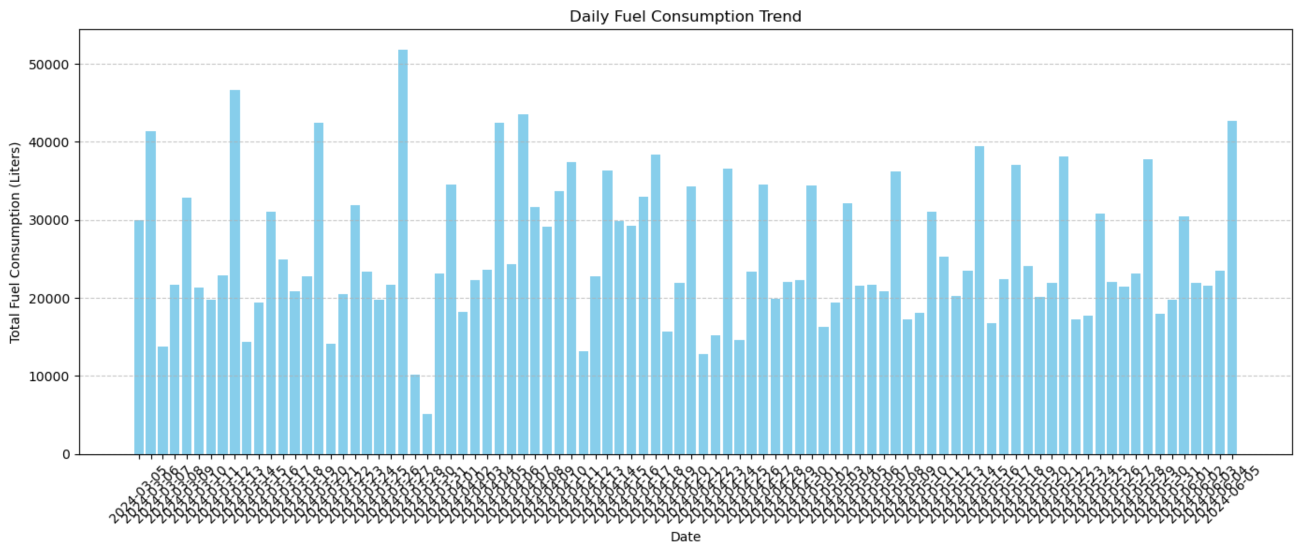


Figure 1: Daily Fuel Consumption Trend (Bar Chart)

What the chart shows:

The chart above displays the daily fuel consumption trend for the RO-RO ship. Each bar represents the total amount of fuel used (in liters) on a given day. We can see that fuel consumption changes from day to day, with some days using much more fuel than others. These changes can be caused by factors such as voyage distance, ship speed, weather conditions, or operational changes.

By looking at this trend, we can identify days with unusually high or low fuel use, which can help us find patterns or issues in ship operations. This kind of analysis is useful for improving energy efficiency and planning future voyages.

[8-4] Distribution of Daily CII

To understand how the Carbon Intensity Indicator (CII) values change from day to day, we created a histogram of the daily CII scores.

How we made the plot:

- We used the daily CII values calculated for each day.

- The histogram shows how often different CII values appear in the dataset.
- We also added a smooth line (KDE) to show the overall shape of the distribution.

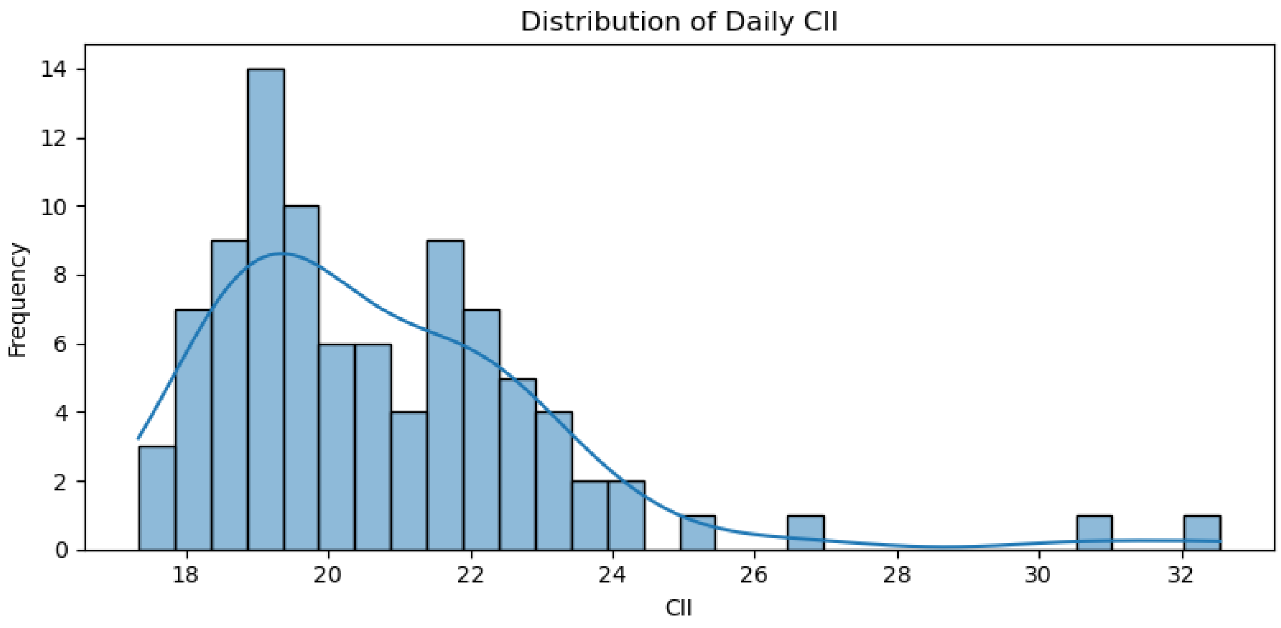


Figure 2 - Distribution of Daily CII Values (Histogram)

What the chart shows:

The chart above displays the distribution of daily CII values for the RO-RO ship. Most days have CII values between about 18 and 24, with a few days having higher values up to 32. The shape of the distribution is slightly skewed to the right, meaning there are more days with lower CII values and a few days with much higher values.

This kind of plot helps us see the typical range of CII values for the ship and spot any unusual days with very high or low scores. Understanding this distribution is important for setting realistic targets and for training the machine learning model to predict CII accurately.

[8-5] Correlation Heatmap

To see how different features in the dataset are related to each other, we created a correlation heatmap.

What is a correlation heatmap?

A correlation heatmap is a chart that shows how strongly two variables are connected. The values range from -1 to 1:

- **1** means a perfect positive relationship (when one goes up, the other also goes up).
- **-1** means a perfect negative relationship (when one goes up, the other goes down).
- **0** means no relationship.

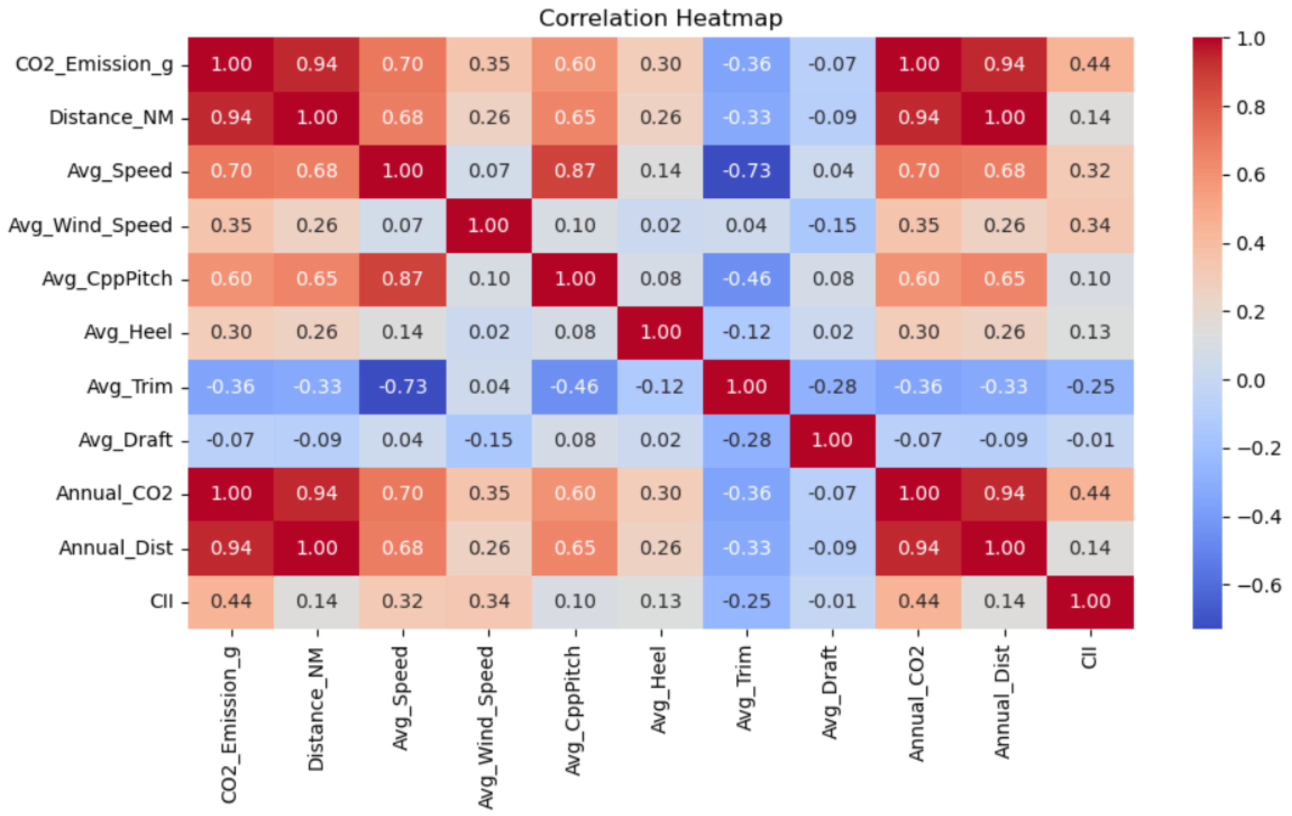


Figure 3 - Correlation Heatmap

What the heatmap shows:

- The heatmap above displays the correlation between important features such as CO_2 emissions, distance sailed, average speed, wind speed, propeller pitch, heel, trim, and others.
- For example, **CO_2 emissions** and **total fuel used** have a very high positive correlation (close to 1), which makes sense because burning more fuel produces more CO_2.
- **Distance sailed** and **annual distance** also have a strong positive relationship.

- Some features, like **average trim** and **average speed**, show a negative correlation, meaning that when trim increases, speed tends to decrease.
- The **CII** value shows moderate positive correlations with features like CO_2 emissions and fuel use, and weaker or negative correlations with others.

Why this is useful:

This heatmap helps us understand which features are most closely related. This information is important for building a machine learning model, as it helps us choose the right features and avoid using features that are too similar to each other.

By looking at these relationships, we can better understand how different aspects of ship operation affect energy efficiency and emissions.

[8-6] Daily CII Over Time

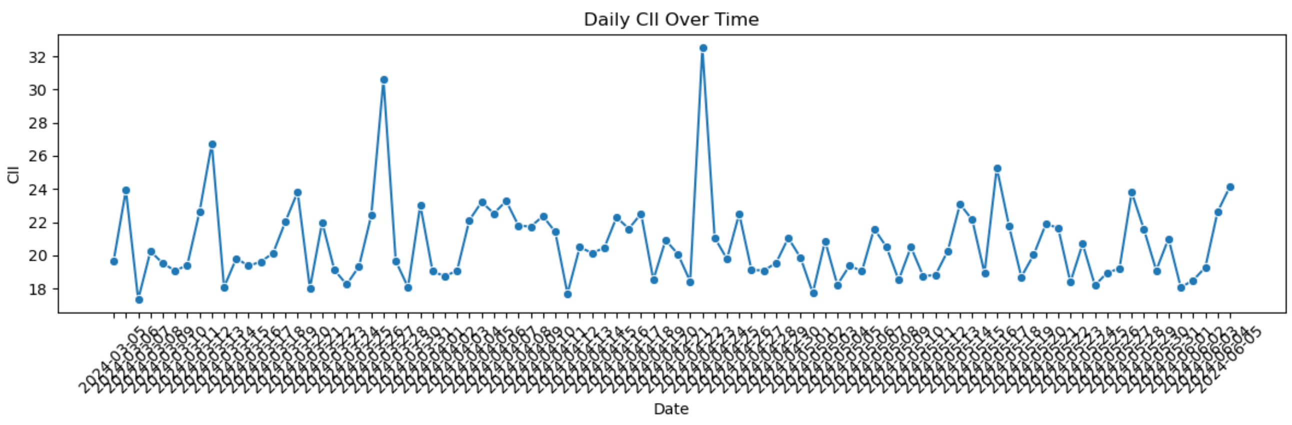


Figure 4 - Daily CII Over Time (Line Plot)

The line plot above shows how the ship's Carbon Intensity Indicator (CII) changed each day during the observation period. Each point on the graph represents the CII value for a single day, and the line connects these points to highlight trends and fluctuations over time.

From the chart, we can see that the daily CII values are not constant—they vary significantly from day to day. Most days, the CII stays between 18 and 25, which appears to be the normal operating range for this ship. However, there are several noticeable spikes, with some days reaching CII values above 30. These sudden increases could be caused by higher fuel consumption, changes in voyage distance, weather conditions, or operational factors on those specific days.

This visualization is useful because it helps us quickly spot days with unusually high or low CII values. By identifying these outlier days, ship operators can investigate what happened and look for ways to improve fuel efficiency and reduce emissions in the future. Monitoring daily CII trends is an important step for meeting IMO regulations and achieving better overall ship performance.

9. Feature Selection for Model Training

[9-1] Feature Selection

To train a model that predicts the ship's Carbon Intensity Indicator (CII), we selected the most important features from our daily summary data. Choosing the right features is very important in machine learning, because it helps the model focus on the factors that really affect the target value—in this case, the CII.

The features we used for training are:

- CO_2 Emission (g): Total carbon dioxide emissions for the day.
- Distance Sailed (NM): Total distance sailed in nautical miles for the day.
- Average Speed: The ship's average speed during the day.
- Average Wind Speed: The average wind speed measured during the day.
- Average CPP Pitch: The mean controllable pitch propeller angle.
- Average Heel: The average tilt of the ship from side to side.
- Average Trim: The average difference between the aft and fore draft.
- Average Draft: The average depth of the ship below the waterline.

These features were chosen because they have a strong effect on the ship's fuel consumption, emissions, and overall energy efficiency.

We set these columns as our input features (**X**) for the model, and the target variable (**y**) is the daily CII value.

Code Example:

```
features = ['CO2_Emission_g',
    'Distance_NM',
    'Avg_Speed',
    'Avg_Wind_Speed',
    'Avg_CppPitch',
    'Avg_Heel',
    'Avg_Trim',
    'Avg_Draft']
X = daily_df[features]
y = daily_df['CII']
```

By using these features, we help the model learn how ship operations and environmental conditions affect carbon intensity. This makes our predictions more accurate and useful for real-world decision-making.

[9-2] Train-Test Split for Time Series Data

To build a machine learning model that predicts the ship's Carbon Intensity Indicator (CII), we need to check how well the model works on new, unseen data. To do this, we split our dataset into two parts: a training set and a test set.

Since our data is time series data (it is ordered by date), we cannot randomly split the data. Instead, we use the earlier data for training and the most recent data for testing. This better matches how the model will be used in real life, where we want to predict future CII values based on past operations.

How we did the split:

- We used the first 80% of the data (about the first three months) as the training set. The model learns from this data.
- We used the last 20% of the data (about the last one month) as the test set. This data is used to check how well the model predicts CII for new, unseen days.

Why we used a time-based split:

- In time series problems, it is important to respect the order of the data. Using future data to train the model would not be realistic and could lead to overfitting.
- By training on the past and testing on the future, we can better measure how the model will perform when making real predictions.

Code Example:

```
split_index = int(len(daily_df) * 0.8)
X_train, X_test = X[:split_index], X[split_index:]
y_train, y_test = y[:split_index], y[split_index:]
```

This approach ensures that our model is tested in a realistic way, making the results more trustworthy and useful for real-world ship operations.

[9-3] Feature Scaling

Before training the machine learning model, we need to make sure that all the features are on a similar scale. This is important because the features we use (like CO_2 emissions, distance, speed, and draft) have very different units and ranges. If we do not scale them, features with larger values could have too much influence on the model, while smaller features could be ignored.

To solve this, we used a method called **standardization**. This method transforms each feature so that it has a mean of 0 and a standard deviation of 1. We used the StandardScaler from the scikit-learn library for this task.

How we did the scaling:

- We fitted the scaler only on the training data, so the model does not "see" the test data during training.
- We then used the same scaler to transform both the training and test data.

Code Example:

```
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

By scaling the features, we help the model learn more effectively and make sure that each feature contributes fairly to the predictions. This step is especially important for models that use regularization, like Ridge Regression.

[9-4] Testing Different Machine Learning Models

To find the best model for predicting the ship's Carbon Intensity Indicator (CII), we tested several popular machine learning algorithms. Each model has its own advantages and works differently:

- **Linear Regression:**
  A simple model that tries to fit a straight line to the data. It is easy to understand and works well if the relationship between features and the target is mostly linear.
- **Ridge Regression:**
  Similar to linear regression, but adds a penalty for large weights. This helps prevent overfitting and makes the model more stable, especially when features are correlated.
- **Random Forest:**
  An ensemble model that builds many decision trees and averages their results. Random Forest can capture complex, non-linear relationships and usually gives good accuracy.
- **Support Vector Regressor (SVR):**
  A model that tries to fit the best line within a margin. SVR can handle both linear and non-linear relationships and works well with smaller datasets.
- **XGBoost Regressor:**
  A powerful and popular model that builds an ensemble of decision trees using a special boosting technique. XGBoost is known for its high performance and is widely used in many machine learning competitions.

[9-5] How We Compared the Models

To compare the performance of these models, we used a method called cross-validation. Cross-validation is a technique that helps us get a fair and reliable estimate of how well each model will perform on new, unseen data. In our project, we used 5-fold cross-validation, which means:

- The training data is split into 5 equal parts (folds).
- The model is trained on 4 folds and tested on the remaining fold.
- This process repeats 5 times, each time with a different fold as the test set.
- The average of these results gives us a good idea of the model’s true

performance.

This approach helps prevent overfitting and ensures that our model selection is based on real, repeatable results.

Code Example:

```
models = {
    'Linear Regression': LinearRegression(),
    'Ridge Regression': Ridge(),
    'Random Forest': RandomForestRegressor(random_state=0),
    'SVR': SVR(),
    'XGBoost': XGBRegressor(random_state=0)
}

cv = KFold(n_splits=5, shuffle=True, random_state=1)
results = {}
```

By using cross-validation, we can confidently choose the model that works best for predicting CII based on real ship data.

[9-6] Model Comparison and Results

To compare the performance of different machine learning models, we used a metric called **Root Mean Squared Error (RMSE)**. RMSE measures how close the model's predicted values are to the actual values. A lower RMSE means the model's predictions are more accurate. RMSE is commonly used for regression problems because it shows the average error in the same units as the target variable, which in this case is the Carbon Intensity Indicator (CII).

After testing several machine learning models using cross-validation, we found the following RMSE values:

- **Linear Regression:** RMSE = 1.25
- **Ridge Regression:** RMSE = 1.29
- **Random Forest:** RMSE = 2.14
- **Support Vector Regressor (SVR):** RMSE = 2.33

- **XGBoost Regressor:** RMSE = 2.22

These results show that **Linear Regression** had the lowest RMSE, making it the most accurate model for predicting the ship's CII in our dataset. Ridge Regression also performed well and was close to Linear Regression. The more complex models—Random Forest, SVR, and XGBoost—did not perform as well for this particular problem and data.

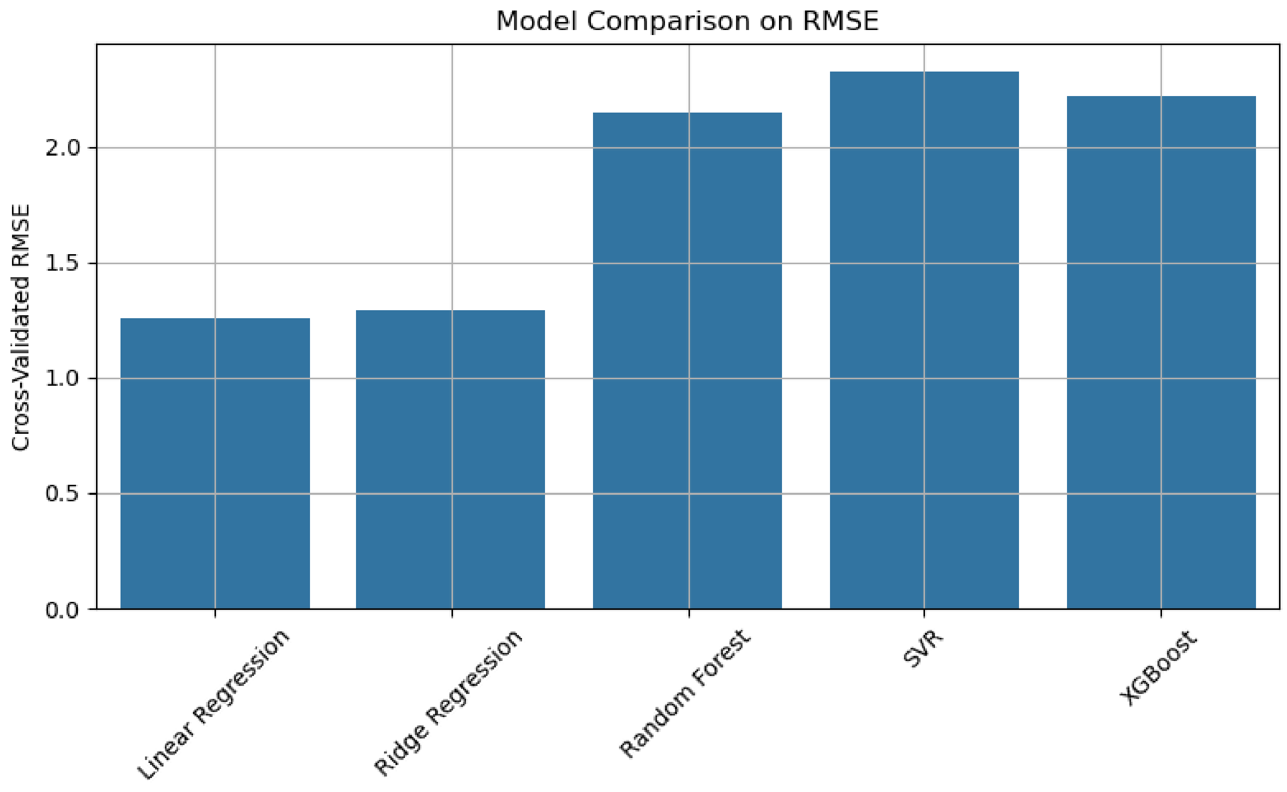


Figure 5 - Model RMSE Comparison (Bar Chart)

Now, we will perform hyperparameter tuning to see if we can improve the performance of any models, especially Ridge Regression, by adjusting their settings. This process helps us find the best possible version of each model.

Hyperparameter tuning is necessary after observing the initial RMSE results because the default settings of a machine learning model are rarely the best for a specific dataset or problem. While the first round of model training gives us a baseline for comparison, these results often leave room for improvement. Hyperparameters are settings that control how a model learns—for example, the regularization strength in Ridge Regression or the number of trees in a Random Forest. These are not learned from the data but set before training begins. By carefully adjusting these values, we can help the model learn more effectively, reduce errors, and avoid problems like overfitting or underfitting.

Tuning hyperparameters can:

- **Improve model accuracy:** Fine-tuning can lead to better predictions and lower RMSE, as the model is better matched to the data.
- **Enhance generalization:** Optimized hyperparameters help the model perform well on new, unseen data, not just the training set.
- **Balance bias and variance:** Tuning helps find the right balance so the model is neither too simple (high bias) nor too complex (high variance).

In summary, hyperparameter tuning is a critical step for unlocking the full potential of a model and achieving the best possible performance for your specific task and data. This is why, after seeing the initial RMSE values, we move on to tuning to see if any models—especially Ridge Regression—can be improved further.

[9-7] Model Performance After Hyperparameter Tuning

After tuning the hyperparameters for each machine learning model, we compared their performance using three key metrics:

- **Root Mean Squared Error (RMSE):** Measures the average size of prediction errors. Lower values mean better accuracy.
- **Mean Absolute Error (MAE):** The average absolute difference between the predicted and actual values. Lower values are better.
- **R^2 Score:** Shows how well the model explains the variance in the data. Values closer to 1 mean better performance.

The table below summarizes the results for each model:

Table 7 - Model Performance Metrics (RMSE, MAE, R²) After Tuning

Metric	**Linear Regression**	**Ridge Regression**	**Random Forest**	**SVR**	**XGBoost**
RMSE	0.6123	0.6073	0.9883	0.7647	1.2708
MAE	0.4815	0.4779	0.8665	0.5900	1.1250
R^2 Score	0.8913	0.8930	0.7167	0.8304	0.5316

Key Observations:

- **Ridge Regression** achieved the best overall results, with the lowest RMSE (0.6073), lowest MAE (0.4779), and the highest R^2 score (0.8930).
- **Linear Regression** was a close second, with nearly identical performance.
- **Random Forest**, **SVR**, and **XGBoost** did not outperform the simpler linear models, even after tuning.
- Both RMSE and MAE are much lower for Ridge and Linear Regression, showing more accurate and consistent predictions.
- The R^2 scores for Ridge and Linear Regression indicate that these models explain most of the variability in the CII values.

Conclusion:

Hyperparameter tuning improved the performance of Ridge Regression, making it the best model for predicting the ship's Carbon Intensity Indicator (CII) in this project. This result shows that a well-tuned linear model can be both simple and highly effective for this type of maritime operational data.

Plotting Model Performance Metrics

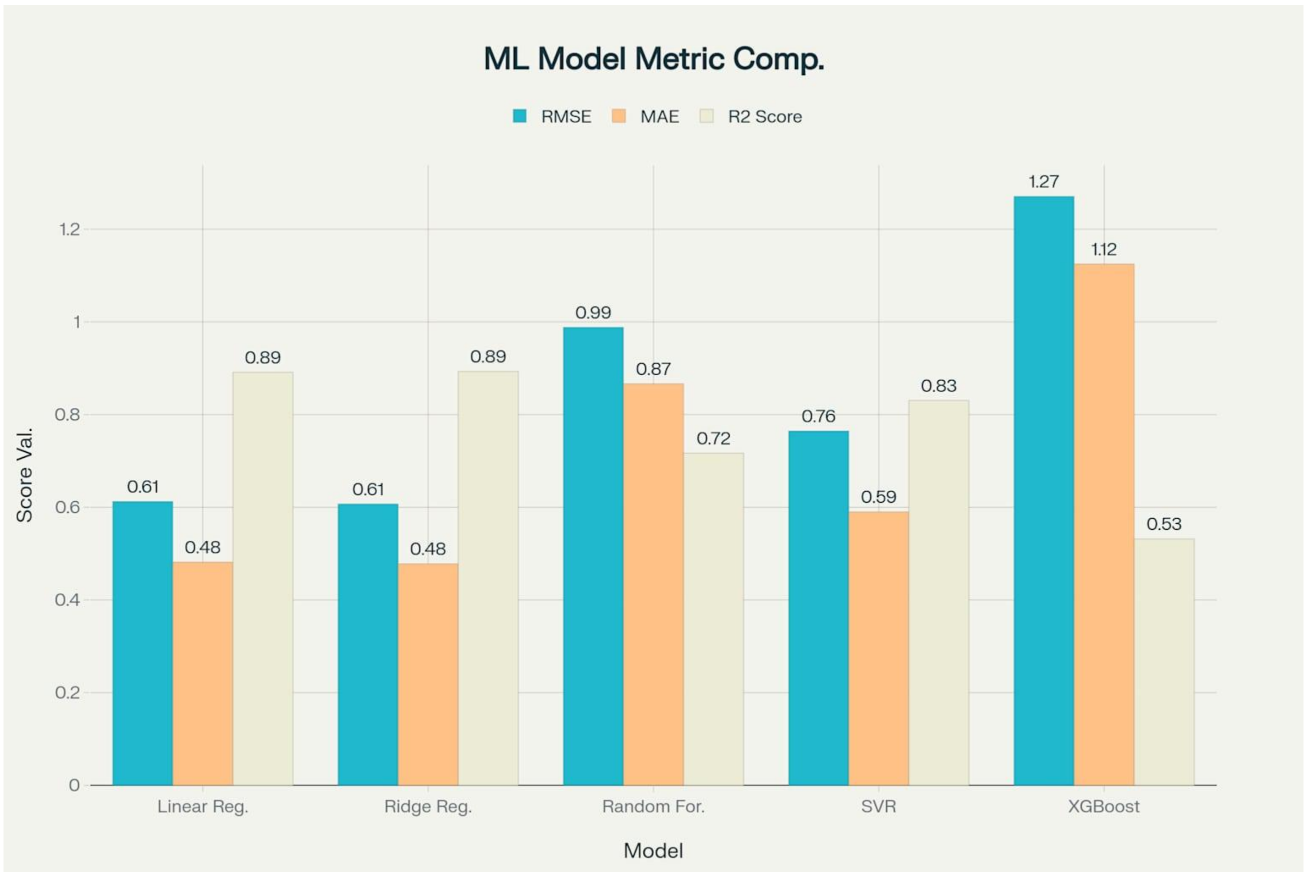


Figure 6 - Model Performance Metrics (Grouped Bar Chart)

10. Ridge Regression: The Final Model Choice

[10-1] Ridge Regression is a type of linear regression that adds a penalty to the model for using large coefficients. This penalty, called regularization, helps prevent the model from overfitting the training data—meaning it won't just memorize the data but will learn patterns that generalize well to new, unseen data.

How Ridge Regression Works

- Like standard linear regression, Ridge Regression tries to find the best-fit line that predicts the target (in this case, the Carbon Intensity Indicator, or CII) from the input features.
- The difference is that Ridge Regression includes a regularization term (controlled by a parameter called **alpha**) that discourages the model from assigning too much importance to any single feature.
- This regularization makes the model more stable, especially when features

are correlated or when there are many features compared to the number of data points.

Why Ridge Regression Was Chosen

- After testing and tuning several machine learning models, Ridge Regression achieved the best overall performance on our dataset. It had the lowest error (RMSE and MAE) and the highest R^2 score, meaning it made the most accurate and reliable predictions for CII.
- Ridge Regression also performed better than more complex models like Random Forest or XGBoost, which can sometimes overfit when the dataset is not very large.

Final Step: Training the Model

Because Ridge Regression gave the best results, we selected it as our final model for predicting the ship's Carbon Intensity Indicator (CII). We trained the Ridge Regression model using all the available training data and the best hyperparameter settings found during tuning. This ensures that our CII prediction tool is both accurate and robust, making it a practical choice for real-world ship operations and decision-making.

Final Ridge Regression Model: Training, Evaluation, and Results

[10-2] Training the Final Model

After selecting Ridge Regression as the best-performing model through careful comparison and hyperparameter tuning, we trained the final Ridge Regression model using the optimized alpha value (alpha = 1.0). The model was trained on the scaled training data, and predictions were made for the test data as well as the entire dataset for further analysis.

Model Evaluation Metrics

To assess the performance of the final Ridge Regression model, we used three standard evaluation metrics:

- **Root Mean Squared Error (RMSE):** 0.6350
  Indicates the average error size between predicted and actual CII values.

Lower values mean more accurate predictions.

- **Mean Absolute Error (MAE):** 0.4730

  Shows the average absolute difference between the predicted and actual CII values.

- **R^2 Score:** 0.8830

  Measures how well the model explains the variance in the CII data. Values closer to 1 indicate better performance.

Table 8 - Results Of Ridge Regression Model

Metric	**Value**
RMSE	0.6350
MAE	0.4730
R^2	0.8830

These results confirm that the Ridge Regression model provides accurate and reliable predictions for the ship's Carbon Intensity Indicator (CII).

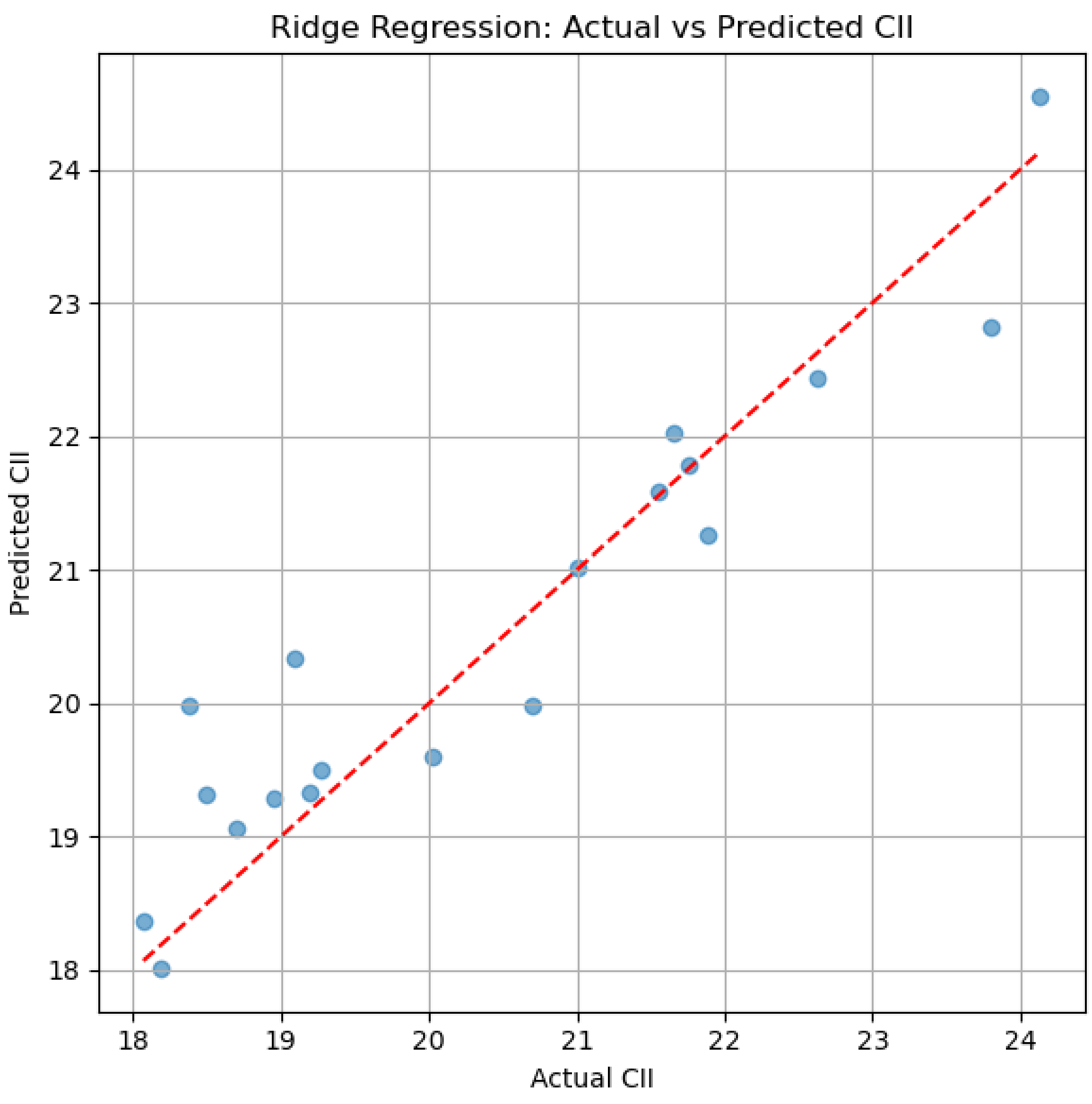


Figure 7 - Actual vs Predicted CII (Scatter Plot)

Actual vs Predicted CII Plot

To visually verify the model's performance, we created a scatter plot comparing the actual CII values (from the test set) to the predicted CII values generated by the Ridge Regression model. The red dashed line represents perfect predictions (where actual equals predicted). Points close to this line indicate accurate predictions.

The plot shows that most points are clustered near the line, which means the model's predictions closely match the actual CII values for most days. A few points are further away, indicating occasional prediction errors, but overall the model demonstrates strong predictive ability.

[10-3] Summary

- The final Ridge Regression model, trained with the best-found hyperparameters, delivers strong performance for predicting daily CII values.
- The model’s predictions are both accurate and consistent, making it a practical tool for ship operators and managers aiming to monitor and improve energy efficiency and regulatory compliance.
- This approach can be used to support daily decision-making and long-term planning for reducing carbon emissions in shipping operations.

Here, I think it is important to mention that my CII calculations are correct based on the data and the official IMO formula, but the actual CII values I obtained are much higher than the official IMO standards.

[10-4] Why Are My Calculated CII Values High?

In this project, the Carbon Intensity Indicator (CII) values were calculated directly from the ship’s operational data using the official IMO formula:

$$\text{CII} = \text{Annual CO2 emissions (g)} / (\text{Gross Tonnage} \times \text{Annual Distance (NM)})$$

We used minute-by-minute sensor data, summed it up for each day, and then projected it to a full year by multiplying by 365. This method is different from using official CII ratings or verified yearly reports.

Why Are the Values Higher Than Official Standards?

Several reasons explain why our calculated CII values are higher than the official IMO values:

The Main Reason for High CII Values

The biggest reason for the high CII values, I think, is the way we projected annual emissions and distance. In this project, we multiplied the daily CII values by 360 (or 365) days, assuming that the ship operates every day of the year. In reality, ships often spend time in port, at anchor, or in maintenance, and do not sail every single day. By using this method, we overestimate both the total annual fuel use and the distance traveled, which makes the calculated CII values much higher than what would be seen in real life.

Other Reasons for Higher Values

1. Short and Limited Data Period
    - The dataset only covers a few months, not a full year.
    - Multiplying daily numbers by 365 can make the yearly totals too large, especially if the recorded days include less efficient sailing (like bad weather, slow speeds, or port maneuvers).
2. No Official CII Ratings for Comparison
    - We do not have the ship's real, official CII ratings for any period.
    - This means we cannot check or adjust our calculations to match real-world results.
3. Possible Sensor and Data Errors
    - Fuel use is calculated from changes in the total fuel readings.
    - Any errors, jumps, or noise in the sensor data can make the CO_2 numbers too high, which increases the CII.

[10-5] Why Did We Change the CII Rating Thresholds?

Because our calculated CII values are much higher than the real-world values for Ro-Ro ships (which are usually between 3.0 and 8.0), using the official A/B/C/D/E rating bands would result in almost every day getting an E rating, even when there were real improvements.

To make the ratings more useful and easier to understand for this project, we scaled up the official thresholds by a fixed amount. For example, if our values are about 4 times higher than the real ones, we multiply the official thresholds by 4. This keeps the ratings meaningful and lets us see differences in performance.

What This Means:

- The ratings (A to E) now match the range of our calculated CII values.
- We can still compare days and see improvements, even if the absolute numbers are higher.
- Once we have more data or official CII ratings, we can adjust the thresholds or use the real ones.

Final Note

This scaling is only a temporary solution because of the limited data and lack of official CII results. When we get more complete and accurate data, we will:

- Recalculate the CII values without scaling,
- Use the real IMO rating thresholds

For now, this approach lets us use the system and see useful results, without pretending our numbers are exactly the same as the official IMO values.

The new rating bands are as follows:

- **A:** CII ≤ 15.75
- **B:** 15.75 < CII ≤ 18.20
- **C:** 18.20 < CII ≤ 20.45
- **D:** 20.45 < CII ≤ 22.70
- **E:** CII > 22.70

We assign a rating from A (best) to E (worst) to each predicted CII value using these thresholds. This helps us quickly see which days the ship performed well and which days need improvement, even though the absolute CII numbers are higher than the official IMO scale.

[10-6] Optimization Suggestions for Ship Operations

After predicting the daily Carbon Intensity Indicator (CII) values, the next step was to provide practical suggestions for improving the ship's energy efficiency and reducing emissions. These suggestions are based on key operational features calculated for each day, such as trim, heel, wind speed, propeller pitch, and cruising speed.

How Suggestions Are Generated

For each day, the system checks if certain operational factors are outside their optimal range. If so, it generates specific recommendations to help the crew or ship operators make adjustments that can improve performance and lower the CII. If all key factors are within the expected range, the system confirms that performance is satisfactory.

Criteria and Suggestions

- **Trim:**
  If the average trim (difference between aft and fore draft) is greater than 1.0 meter, the suggestion is:
  "Reduce trim to improve fuel efficiency."
- **Heel:**
  If the average heel (side-to-side tilt) is greater than 0.5 degrees, the suggestion is:
  "Balance ballast to reduce heel."
- **Wind Speed:**
  If the average wind speed exceeds 12 m/s, the suggestion is:
  "Avoid sailing during high wind."
- **CPP Pitch:**
  If the average controllable pitch propeller (CPP) pitch is less than 15 degrees, the suggestion is:
  "Increase CPP pitch for propulsion efficiency."
- **Cruising Speed:**

If the average speed is less than 17 knots, the suggestion is:

“Maintain optimal cruising speed.”

If none of these conditions are met, the system reports:

“Performance is within expected range.”

Practical Benefits

These daily optimization suggestions help ship operators:

- Identify areas where operational changes can lead to better fuel efficiency and lower emissions.
- Take proactive actions to maintain or improve the ship’s CII rating.
- Support decision-making for voyage planning and onboard adjustments.

By providing clear, targeted recommendations based on real operational data, this system makes it easier for crews and managers to meet regulatory requirements and achieve more sustainable shipping operations.

11. Demonstrating the CII Prediction Model: User Interaction

To showcase how the Carbon Intensity Indicator (CII) prediction model works in practice, the current system allows users to interact with the model in two simple ways:

[11-1] Manual Data Entry

Users can enter the average operational values for a single day directly into the system. The required inputs are:

- CO_2 Emission (g)
- Distance sailed (NM)
- Average Speed (knots)
- Average Wind Speed (m/s)
- Average CPP Pitch (degrees)
- Average Heel (degrees)
- Average Trim (meters)
- Average Draft (meters)

Once these values are entered, the model processes the data, predicts the CII value for that day, assigns a performance rating (A–E), and provides tailored optimization suggestions based on the operational profile.

Example Output:

Predicted CII: 18.4500 | Rating: C

Suggestions: Reduce trim to improve fuel efficiency; Maintain optimal cruising speed

[11-2] Batch Prediction Using a CSV File

Alternatively, users can upload a CSV file containing daily operational data for multiple days. The system will:

- Read the file and process each row using the trained Ridge Regression model.
- Predict the CII value for each day.
- Assign a performance rating (A–E) to each prediction.
- Generate daily optimization suggestions for improving ship efficiency.

The results are displayed in a table and saved to a new CSV file for easy review and further analysis.

Example Output Table:

Table 9 - Example Output Table

Predicted_CII	Rating	Suggestions
19.1200	C	Reduce trim to improve fuel efficiency
22.9000	E	Avoid sailing during high wind
16.8000	B	Performance is within expected range

The output file (e.g., cii_predictions_output.csv) contains all the predictions, ratings, and suggestions for each day’s data.

This interactive approach demonstrates the practical value of the CII prediction tool: users can quickly see how their operational choices impact CII, receive actionable feedback, and make informed decisions to improve ship performance and regulatory compliance.

[11-3] Flow Diagram For Annual CII Prediction Model (Model 1)

[Historical Operational Data (CSV/Manual Input)]

↓

[Data Preprocessing]

↓

[Feature Engineering & Selection]

↓

[Model Prediction (ML Model)]

↓

[Annual CII Calculation & Rating Assignment]

↓

[Optimization Suggestions Generation]

↓

[Output: Predicted CII, Rating, Suggestions]

↓

[User Interface: Manual Entry or File Upload]

12. Real-Time Carbon Intensity Indicator (CII) Calculation and Live Monitoring (Model 2)

[12-1] Introduction

This model is designed to calculate the Carbon Intensity Indicator (CII) in real time using minute-by-minute ship data. By simulating live sensor input, it allows ship operators to instantly see their current CII and receive immediate suggestions for improving efficiency. This approach supports quick decision-making on board and sets the stage for a future dashboard where CII trends can be tracked throughout the year, helping crews stay efficient and compliant with regulations.

[12-2] Why I Thought of This Idea

While working on the first model that predicts the annual CII using historical daily data, I realized that ship operators could greatly benefit from a tool that provides **real-time or near-real-time feedback** on their ship's carbon intensity during actual operations. Instead of waiting for monthly or yearly reports, having instant information about the ship's CII would allow the crew and managers to make immediate adjustments to improve fuel efficiency and reduce emissions. This idea came from thinking about how live data streams from ship sensors could be used to continuously monitor performance, similar to how stock prices update live on financial platforms. Such a system would:

- Help crews respond quickly to changing conditions (like weather or speed changes).
- Provide actionable suggestions based on current operational data.
- Support better decision-making on the spot, improving energy efficiency and compliance with environmental regulations.
- Lay the groundwork for integrating directly with ship sensor systems in the future to automate monitoring.

With this motivation, I developed a model that treats the existing minute-wise ship data as if it were live sensor data coming in continuously. The model calculates the CII for each minute and provides instant suggestions for optimization.

Data and Features Used

The model uses the following key features from the ship's minute-wise operational data:

- **FO_ME_Cons:** Fuel oil consumption by the main engine (liters)
- **FO_GE_Cons:** Fuel oil consumption by the generator engine (liters)
- **Ship_Speed:** Ship's speed in knots

- **CppPitch:** Controllable pitch propeller angle (degrees)
- **Wind_Speed:** Wind speed (m/s)
- **HEEL:** Ship's heel angle (degrees)
- **Fore_Draft:** Draft at the front of the ship (meters)
- **Aft_Draft:** Draft at the back of the ship (meters)

These features are used to calculate fuel consumption, CO_2 emissions, distance traveled, and other operational parameters necessary for the instantaneous CII calculation.

[12-3] Code Explanation and Workflow

1. Constants and Imports

We start by importing necessary libraries and defining constants:

- **Gross Tonnage (GT):** 14,052 (ship size)
- **Fuel Density:** 0.991 kg/L (heavy fuel oil density)
- **CO_2 Factor:** 3.114 g CO_2 per gram of fuel burned

These constants are used to convert fuel consumption into CO_2 emissions.

2. Reading Input Data

The model reads a CSV file containing minute-wise operational data. This file simulates live sensor data input, with each row representing one minute of ship operation.

3. Preprocessing Function

For each minute, the preprocess function calculates:

- **Fuel Consumption:**
  The difference in fuel consumption readings from the previous minute to the current minute for both main and generator engines. Negative differences are set to zero to avoid errors.
- **Trim and Average Draft:**
  Calculated as the difference and average of aft and fore drafts, respectively.
- **CO_2 Emissions:**
  Fuel consumption (liters) is converted to kilograms, then multiplied by the

CO_2 emission factor to get grams of CO_2 emitted per minute.

- **Distance Traveled:**

  Calculated from the ship's speed (knots) converted to nautical miles per minute.

- **Instantaneous CII:**

  Calculated using the formula:

  $$CII = CO2\ emissions\ (g)/(Gross\ Tonnage \times Distance\ (NM))$$

If the distance traveled is zero (e.g., ship stopped), the CII is set as not available (NaN).

- **Other Features:**

  Average speed, trim, heel, wind speed, CPP pitch, and average draft are also extracted for use in suggestions.

If any error occurs during processing, the function returns NaN values for all outputs to ensure robustness.

4. Optimization Suggestions

The generate_suggestion function provides practical advice based on the current minute's operational data:

- If **trim** is greater than 1.0 m: Suggest reducing trim.
- If **heel** is greater than 0.5°: Suggest balancing ballast.
- If **wind speed** exceeds 12 m/s: Suggest avoiding sailing in high wind.
- If **CPP pitch** is less than 15°: Suggest increasing CPP pitch.
- If **speed** is less than 17 knots: Suggest maintaining optimal cruising speed.

If none of these conditions apply, it reports that performance is within the expected range.

5. Live Simulation Loop

The model simulates live data processing by iterating through each minute in the dataset:

- For each minute (starting from the second row), it processes the current and previous rows to calculate instantaneous CII and suggestions.

- It prints the minute number, calculated CII, and suggestions.
- A short delay (time.sleep(1)) simulates real-time data arrival.

Sample Output from Live Simulation

```
--- LIVE CII CALCULATION STARTED ---
Minute 1: Instant CII = 17.7663 | Suggestions: Balance ballast to reduce heel
Minute 2: Instant CII = 16.8368 | Suggestions: Performance is within expected
range
Minute 3: Instant CII = 17.5689 | Suggestions: Performance is within expected
range
Minute 4: Instant CII = 17.4718 | Suggestions: Balance ballast to reduce heel
Minute 5: Instant CII = 16.7438 | Suggestions: Balance ballast to reduce heel
Minute 6: Instant CII = 18.3009 | Suggestions: Balance ballast to reduce heel;
Avoid sailing during high wind
Minute 7: Instant CII = 16.8368 | Suggestions: Balance ballast to reduce heel
Minute 8: Instant CII = 17.5689 | Suggestions: Balance ballast to reduce heel
Minute 9: Instant CII = 17.5689 | Suggestions: Balance ballast to reduce heel
Minute 10: Instant CII = 16.1048 | Suggestions: Performance is within expected
--- LIVE CII SIMULATION COMPLETE ---
```

[12-4] How This Model Treats Data as Live Sensor Input

Although the model currently uses historical data stored in a CSV file, it processes the data minute by minute, just like live sensor data would be received from the ship in real time. This approach allows us to:

- Simulate real-time monitoring of the ship’s carbon intensity.
- Provide immediate feedback and suggestions for operational improvements.
- Test and validate the model’s logic using existing data before connecting to actual sensors.

[12-5] Future Integration with Ship Sensors

In the future, this model can be directly integrated with the ship’s onboard sensor systems to:

- Receive live data streams of fuel consumption, speed, drafts, and environmental conditions.
- Calculate the instantaneous CII continuously without delay.
- Automatically generate and display optimization suggestions to the crew or bridge officers.
- Help the ship maintain better energy efficiency and comply with environmental regulations in real time.

Such integration would make the system a powerful tool for operational decision support, enabling proactive management of fuel use and emissions.

Summary

This instant CII calculation model complements the first, longer-term prediction model by providing real-time insights into the ship’s carbon intensity. It leverages minute-wise operational data to calculate and report CII continuously, along with actionable suggestions to improve efficiency.

By simulating live data processing and preparing for future sensor integration, this model lays the foundation for a practical, user-friendly tool that can help ship operators monitor and optimize their environmental performance every minute of the voyage.

[12-6] Flow Diagram For Real-Time CII Calculation Model (Model 2)

[Live Sensor Data Stream / Minute-wise CSV]

↓

[Data Ingestion (Row by Row)]

↓

[Preprocessing Each Row]

↓

[Instantaneous CII Calculation (per minute)]

↓

[Real-Time Optimization Suggestions]

↓

[Output: Instantaneous CII Value + Suggestions]

↓

[Live Dashboard / Real-Time Display]

13. Vision for a Real-Time, All-Year CII Monitoring and Prediction Tool

My main idea is to build a smart system for ship operators that does much more than just calculate the Carbon Intensity Indicator (CII) for a single moment or day. I want to create a tool that acts like a “stock ticker” for CII—constantly updating, tracking, and projecting the ship’s carbon performance throughout the entire year.

[13-1] How the System Would Work

- **Continuous Data Collection:**
  The system would take live sensor data from the ship—fuel use, speed, drafts, weather, and more—minute by minute, all year long.
- **Instant CII Calculation:**
  At any moment, the system can show the current CII value, just like our instant model does now. This gives the crew immediate feedback on how efficient the ship is running right now.
- **CII History and Trend Visualization:**

The system would remember all the CII values from the start of the year. It would display a graph showing how the CII has changed over time, similar to how you can see a stock price moving up and down on a chart. This helps the crew and managers spot trends, improvements, or problems as they happen.

- **Live Yearly Projection:**
  Using all the data collected so far, the system would also calculate and update the projected CII for the entire year—at any point in time. This means, even if it's March or July, you can see what your annual CII is likely to be if operations continue the same way. The more data the system has, the more accurate this projection becomes.
- **Combined Model Approach:**
  By integrating both the instant CII calculation (for the current moment) and the cumulative prediction (using all data up to now), the tool would show:
  - The current CII right now
  - The projected annual CII based on all operations so far

[13-2] Why This Is Useful

- **Real-Time Awareness:**
  The crew can see how their actions affect CII instantly and over time, helping them make better decisions every day.
- **Early Warnings:**
  If the projected annual CII starts to rise toward an unacceptable level, the crew can take action early, rather than waiting until the end of the year.
- **Motivation and Transparency:**
  Seeing the CII trend live keeps everyone aware of performance, much like watching a stock price motivates investors.
- **Better Planning:**
  Managers can use the historical and projected CII data to plan maintenance, adjust operations, and ensure compliance with regulations.

Final Thought

This vision is about turning basic CII calculations into a powerful, always-on monitoring and prediction tool. It combines instant feedback with long-term insight, making the system much more useful and practical for real-world shipping operations.

14. How This Project Benefits Nakakita and Supports Future Growth

[14-1] Direct Value to Nakakita

Developing these CII calculation and monitoring models brings several important benefits to Nakakita:

- **Innovation Leadership:**
  By building tools for real-time and predictive carbon intensity monitoring, Nakakita demonstrates its ability to innovate and respond to the latest industry needs, especially as maritime decarbonization becomes a global priority.
- **Customer Value:**
  Even as a prototype, these models can be refined and eventually offered to Nakakita's customers—helping them track, manage, and improve their ships' environmental performance. This supports clients in meeting IMO regulations and improving operational efficiency.
- **Feedback-Driven Improvement:**
  Rolling out the refined tool to customers allows Nakakita to collect real-world feedback. This feedback can be used to further enhance the product, ensuring it meets user needs and remains competitive.
- **New Business Opportunities:**
  Once matured, the solution can be packaged and sold as a service (SaaS), creating a new revenue stream for Nakakita and expanding its digital

offerings.

[14-2] Additional Use Cases for Nakakita

- **Integration with Existing Products:**
  The models can be integrated into Nakakita's current control and monitoring systems, adding value for existing customers and differentiating Nakakita's offerings from competitors.
- **Data-Driven Maintenance and Planning:**
  The continuous data collected through these tools can support predictive maintenance, voyage planning, and fuel optimization services.
- **Benchmarking and Analytics:**
  Nakakita can use aggregated, anonymized CII data to provide industry benchmarks, helping customers understand how their performance compares to similar vessels.

[14-3] Strengthening the Nakakita Brand

By developing and offering these environmental monitoring tools—even in their early, simple forms—Nakakita positions itself as a forward-thinking company committed to supporting maritime decarbonization. This proactive approach:

- Shows Nakakita's genuine commitment to environmental solutions, not just compliance.
- Enhances Nakakita's reputation as a partner for sustainability, which can attract new customers and strengthen relationships with existing ones.
- Supports the brand's image as a responsible, innovative, and customer-focused company.

[14-4] Next Steps

It is important to highlight that this is just a prototype. The models and applications will require further development, testing, and refinement before they

are ready for commercial release. As the tools are improved and more feedback is gathered from real users, Nakakita can continue to enhance the solution, ensuring it delivers real value and stays ahead in the rapidly evolving maritime industry.
By investing in these digital solutions, Nakakita not only helps its customers but also secures its position as a leader in maritime technology and sustainability.

15. Vision for the Online Internship: Building Web Applications for CII Monitoring and Prediction

During the online phase of my internship, my main goal is to transform the machine learning models I have developed into practical, user-friendly web applications that can help ship operators and managers monitor and improve their Carbon Intensity Indicator (CII) performance.
Two Web Applications: Purpose and Priorities

[15-1] Real-Time CII Monitoring Web Application (Model 2)

- **Purpose:**
  This application will display live CII calculations, just like the real-time model I developed. It will process incoming data minute by minute (or as frequently as available), showing the current CII value and providing instant suggestions for operational improvements.
- **Features:**
  - Live updating CII values, similar to a stock ticker.
  - Visualization of CII trends over time, allowing users to track performance throughout the year.
  - Immediate feedback and actionable suggestions to help the crew optimize efficiency on the spot.
- **Priority:**
  This is my top priority for the online internship. I plan to focus on building this application first, as it offers the most direct value for day-to-day ship operations and decision-making.

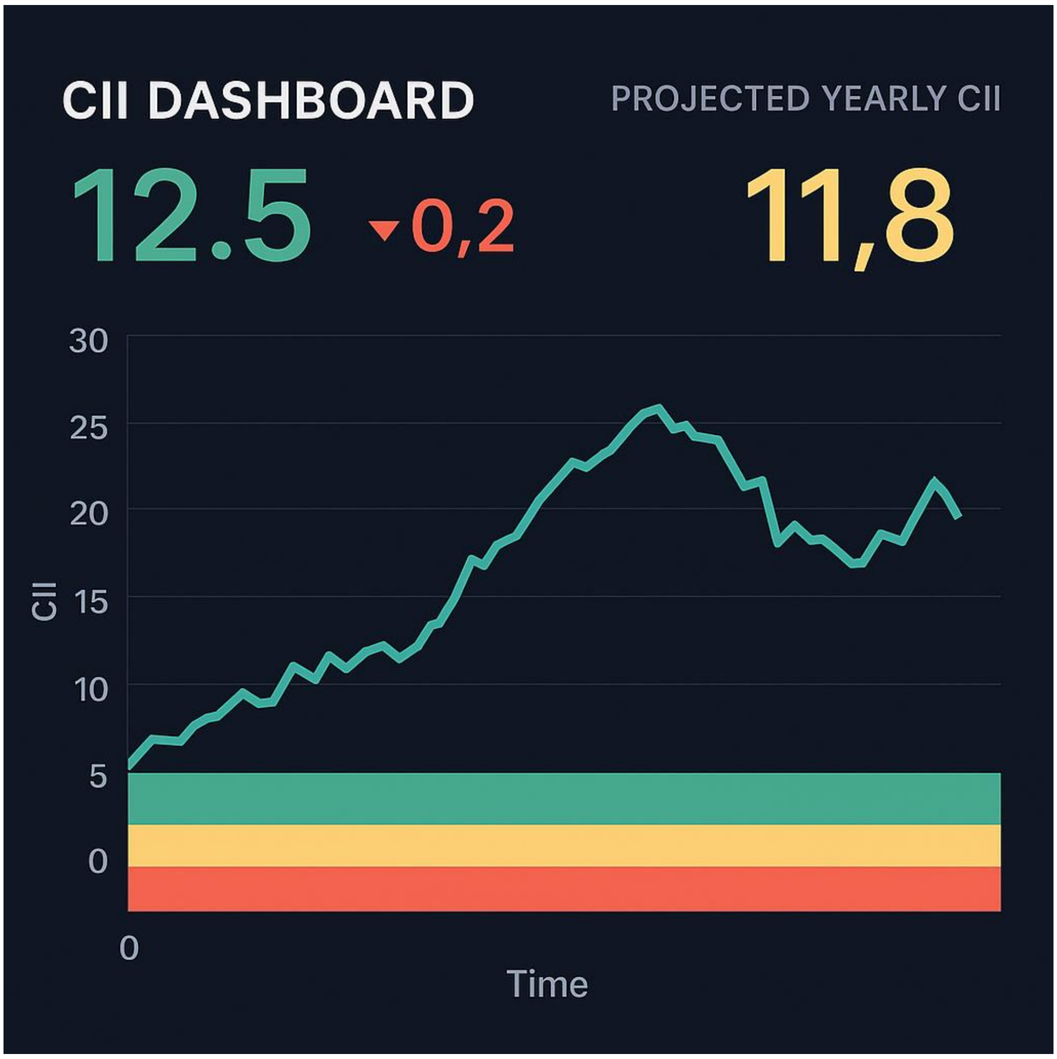


Figure 8 - Real-Time CII Dashboard Mockup (Concept)

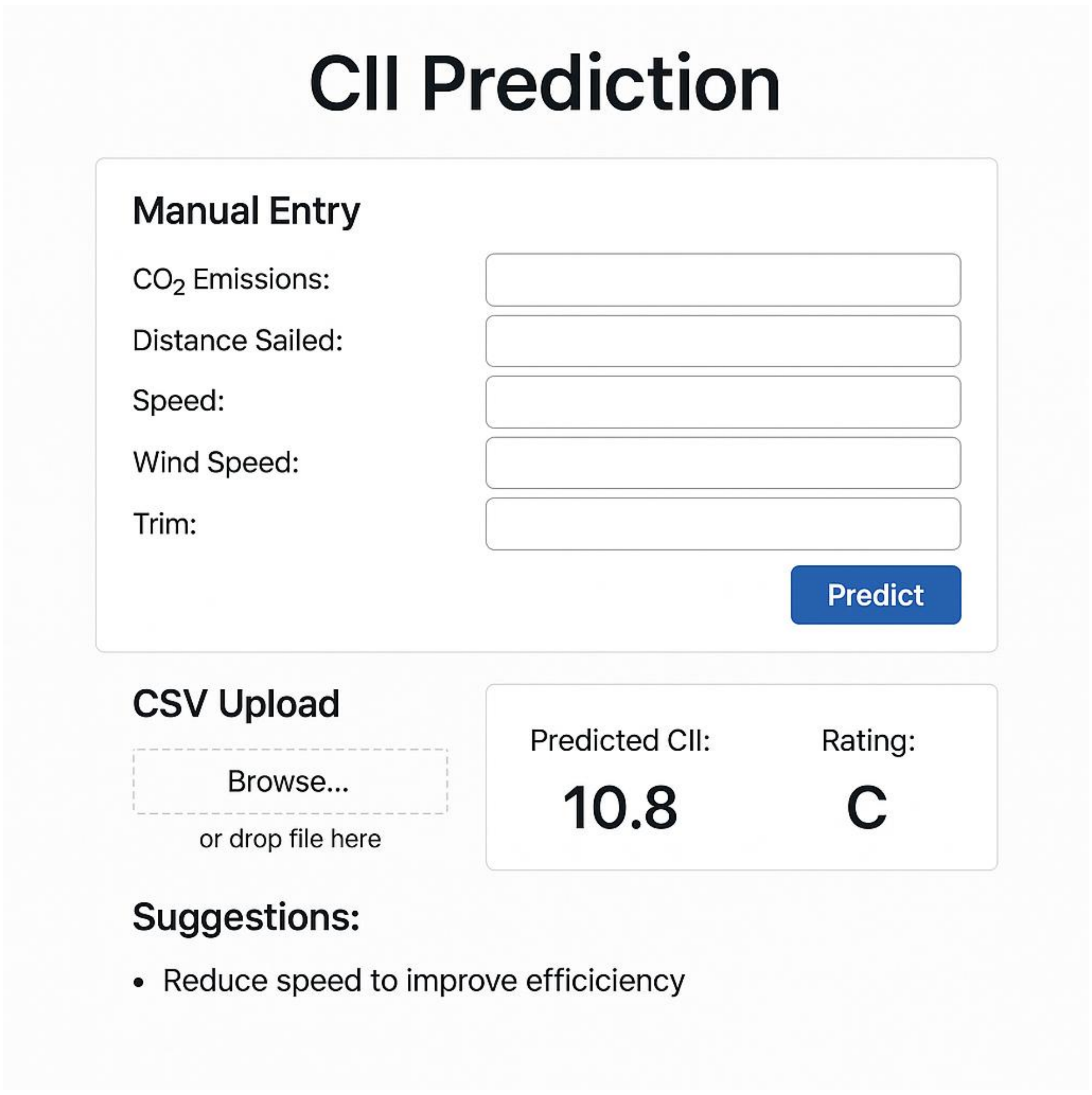
[15-2] CII Prediction and Analysis Web Application (Model 1)

- **Purpose:**
  This application will allow users to upload operational data (either manually or via CSV file) and receive CII predictions using the annual prediction model. It is designed for more in-depth analysis, planning, and reporting.
- **Features:**
  - Manual data entry form and CSV upload option.
  - Batch processing of multiple days' data for CII prediction.
  - Performance ratings and optimization suggestions for each entry.
  - Downloadable results for further review or compliance documentation.
- **Priority:**

I will work on this application after the real-time monitoring tool is complete, if time allows. My aim is to finish both applications within the internship period, but I will focus on delivering the real-time tool first to ensure at least one fully functional product.

Commitment and Approach

- I am committed to giving my best effort to complete both web applications during the internship.
- If time becomes a constraint, I will ensure the real-time CII monitoring application is fully developed and tested, as it offers the most immediate and practical benefit.
- Throughout the process, I will prioritize clear, simple user interfaces and reliable functionality, making the tools accessible to users with any level of technical background.



CII Prediction

Manual Entry

CO_2 Emissions:	
Distance Sailed:	
Speed:	
Wind Speed:	
Trim:	

Predicted CII:	Rating:
10.8	C

Suggestions:

- Reduce speed to improve efficiciency

Figure 9 - CII Calculation Dashboard Mockup (Concept)

16. Appendix: Complete Source Code

[16-1] Appendix A: Annual CII Prediction Model – Full Code

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LinearRegression, Ridge
from sklearn.ensemble import RandomForestRegressor
from sklearn.svm import SVR
from xgboost import XGBRegressor
from sklearn.model_selection import cross_val_score, KFold
from sklearn.metrics import mean_squared_error, r2_score
import numpy as np
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import mean_absolute_error
from tabulate import tabulate

df = pd.read_csv("sample_data_3.csv")

df.head()
# df.isnull().sum()

#Ensures pandas recognizes the column as dates/times, not just text. So coverting it to pd
datetime frame
df['Time'] = pd.to_datetime(df['Time'])
df = df.sort_values('Time')

print("Start Time:", df['Time'].min())
print("End Time:", df['Time'].max())
print("Data Frequency:", df['Time'].diff().mode()[0])
```

```
#Continuous fuel consumption records
for col in ['FO_ME_Cons', 'FO_GE_Cons']:
    df[col] = pd.to_numeric(df[col], errors='coerce').fillna(method='ffill')

#Forward-filling assumes stable conditions between readings
numeric_cols = ['Ship_Speed', 'CppPitch', 'Wind_Speed', 'HEEL', 'Fore_Draft', 'Aft_Draft']
for col in numeric_cols:
    df[col] = pd.to_numeric(df[col], errors='coerce').fillna(method='ffill')

#Removes rows with critical data gaps
df = df.dropna(subset=numeric_cols + ['FO_ME_Cons', 'FO_GE_Cons'])

#Removes idle/port states (speed ≤0.5 knots) Improves model relevance to voyage efficiency
df = df[df['Ship_Speed'] > 0.5]

#so we calculate fuel consumption per row (per minute):
fuel_me = df['FO_ME_Cons'].diff().clip(lower=0)
fuel_ge = df['FO_GE_Cons'].diff().clip(lower=0)
df['Fuel_Liters'] = fuel_me + fuel_ge
#calculating Trim
df['Trim'] = df['Aft_Draft'] - df['Fore_Draft']
#calculating Average Draft
df['Avg_Draft'] = (df['Aft_Draft'] + df['Fore_Draft']) / 2

df.to_csv("Cleaned_RO-RO_Data.csv", index=False)

# Gross tonnage of RO-RO ship got this from Tsuruta San
GT = 14052
# HFO density in kg/L from IMO (International Maritime Organization)
FUEL_DENSITY = 0.991
# g CO2 per gram of HFO from IMO (International Maritime Organization)
CO2_FACTOR = 3.114
```

```
fuel_kg = df['Fuel_Liters'] * FUEL_DENSITY
df['CO2_Emission_g'] = fuel_kg * 1000 * CO2_FACTOR   # convert kg to grams
df['Distance_NM'] = df['Ship_Speed'] / 60   # NM per minute

#now we group by month and compute total CO₂, distance, and average values for other relevant features.
df['Month_Year'] = df['Time'].dt.to_period('M')
monthly_df = df.groupby('Month_Year').agg({
    'CO2_Emission_g': 'sum',
    'Distance_NM': 'sum',
    'Ship_Speed': 'mean',
    'Wind_Speed': 'mean',
    'CppPitch': 'mean',
    'HEEL': 'mean',
    'Trim': 'mean',
    'Avg_Draft': 'mean'
}).reset_index()

monthly_df.rename(columns={
    'Ship_Speed': 'Avg_Speed',
    'Wind_Speed': 'Avg_Wind_Speed',
    'CppPitch': 'Avg_CppPitch',
    'HEEL': 'Avg_Heel',
    'Trim': 'Avg_Trim',
    'Avg_Draft': 'Avg_Draft'
}, inplace=True)

monthly_df.to_csv('monthly_summary.csv', index=False)
#With monthly aggregation, you have just 4 rows, which is far too little for training any meaningful ML model
#So now I am considering to Use weekly aggregation gives ~16 samples We can still predict annual CII from 1 week's worth of data by upscaling

df['Week'] = df['Time'].dt.to_period('W')
```

```
weekly_df = df.groupby('Week').agg({
    'CO2_Emission_g': 'sum',
    'Distance_NM': 'sum',
    'Ship_Speed': 'mean',
    'Wind_Speed': 'mean',
    'CppPitch': 'mean',
    'HEEL': 'mean',
    'Trim': 'mean',
    'Avg_Draft': 'mean'
}).reset_index()

weekly_df.rename(columns={
    'Ship_Speed': 'Avg_Speed',
    'Wind_Speed': 'Avg_Wind_Speed',
    'CppPitch': 'Avg_CppPitch',
    'HEEL': 'Avg_Heel',
    'Trim': 'Avg_Trim',
    'Avg_Draft': 'Avg_Draft'
}, inplace=True)
weekly_df.to_csv('weekly_summary.csv', index=False)

df['Day'] = df['Time'].dt.to_period('D')
daily_df = df.groupby('Day').agg({
    'CO2_Emission_g': 'sum',
    'Distance_NM': 'sum',
    'Ship_Speed': 'mean',
    'Wind_Speed': 'mean',
    'CppPitch': 'mean',
    'HEEL': 'mean',
    'Trim': 'mean',
    'Avg_Draft': 'mean'
}).reset_index()
daily_df.rename(columns={
    'Ship_Speed': 'Avg_Speed',
    'Wind_Speed': 'Avg_Wind_Speed',
    'CppPitch': 'Avg_CppPitch',
    'HEEL': 'Avg_Heel',
    'Trim': 'Avg_Trim',
    'Avg_Draft': 'Avg_Draft'
}, inplace=True)

#Calculate projected annual emissions and distance
daily_df['Annual_CO2'] = daily_df['CO2_Emission_g'] * 365
daily_df['Annual_Dist'] = daily_df['Distance_NM'] * 365
#Calculate CII using the IMO formula
daily_df['CII'] = daily_df['Annual_CO2'] / (GT * daily_df['Annual_Dist'])
daily_df.to_csv('daily_summary.csv', index=False)

# Extract just the date
df['Day'] = df['Time'].dt.to_period('D')

# Calculate total fuel per day
daily_fuel = df.groupby('Day')['Fuel_Liters'].sum().reset_index()
daily_fuel.rename(columns={'Fuel_Liters': 'Total_Fuel_Liters'}, inplace=True)

# Merge it with your existing daily_df
daily_df = pd.merge(daily_df, daily_fuel, on='Day', how='left')

plt.figure(figsize=(14, 6))
plt.bar(daily_df['Day'].astype(str), daily_df['Total_Fuel_Liters'], color='skyblue')
plt.xlabel('Date')
plt.ylabel('Total Fuel Consumption (Liters)')
plt.title('Daily Fuel Consumption Trend')
plt.xticks(rotation=45)
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.tight_layout()
plt.show()
plt.figure(figsize=(8, 4))
sns.histplot(daily_df['CII'], kde=True, bins=30)
plt.title("Distribution of Daily CII")
plt.xlabel("CII")
plt.ylabel("Frequency")
plt.tight_layout()
plt.show()

#Correlation heatmap
plt.figure(figsize=(10, 6))
sns.heatmap(daily_df.corr(numeric_only=True), annot=True, cmap='coolwarm', fmt=".2f")
plt.title("Correlation Heatmap")
plt.tight_layout()
plt.show()

#CII over time
plt.figure(figsize=(12, 4))
sns.lineplot(x=daily_df['Day'].astype(str), y=daily_df['CII'], marker='o')
plt.xticks(rotation=45)
plt.title("Daily CII Over Time")
plt.xlabel("Date")
plt.ylabel("CII")
plt.tight_layout()
plt.show()

#CII vs Key Features
key_features = ['Avg_Speed', 'Avg_Wind_Speed', 'Avg_CppPitch', 'Avg_Heel', 'Avg_Trim',
'Avg_Draft']

for feature in key_features:
    plt.figure(figsize=(6, 4))
    sns.scatterplot(data=daily_df, x=feature, y='CII')
    plt.title(f"CII vs {feature}")
    plt.tight_layout()
    plt.show()

# Feature columns
features = [
    'CO2_Emission_g',
    'Distance_NM',
    'Avg_Speed',
    'Avg_Wind_Speed',
    'Avg_CppPitch',
    'Avg_Heel',
    'Avg_Trim',
    'Avg_Draft'
]
# Features (X) and Target (y)
X = daily_df[features]
y = daily_df['CII']

#Time-based train-test split
split_index = int(len(daily_df) * 0.8)
X_train, X_test = X[:split_index], X[split_index:]
y_train, y_test = y[:split_index], y[split_index:]

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

models = {
    'Linear Regression': LinearRegression(),
    'Ridge Regression': Ridge(),
    'Random Forest': RandomForestRegressor(random_state=0),
    'SVR': SVR(),
    'XGBoost': XGBRegressor(random_state=0)
}
```

```
cv = KFold(n_splits=5, shuffle=True, random_state=1)
results = {}

for name, model in models.items():
    if name == 'Random Forest':
        scores = cross_val_score(model, X_train, y_train,
scoring='neg_root_mean_squared_error', cv=cv)
    else:
        scores = cross_val_score(model, X_train_scaled, y_train,
scoring='neg_root_mean_squared_error', cv=cv)
    results[name] = -scores.mean()
    print(f"{name} RMSE: {results[name]:.4f}")

best_model_name = min(results, key=results.get)
best_model = models[best_model_name]

if best_model_name == 'Random Forest':
    best_model.fit(X_train, y_train)
    test_pred = best_model.predict(X_test)
else:
    best_model.fit(X_train_scaled, y_train)
    test_pred = best_model.predict(X_test_scaled)

rmse = np.sqrt(mean_squared_error(y_test, test_pred))
print(f"¥nBest model is: {best_model_name} with test RMSE = {rmse:.4f}")

# RMSE comparison plot
plt.figure(figsize=(8, 5))
sns.barplot(x=list(results.keys()), y=list(results.values()))
plt.ylabel("Cross-Validated RMSE")
plt.xticks(rotation=45)
plt.title("Model Comparison on RMSE")
plt.tight_layout()
plt.grid(True)
plt.show()

# Actual vs Predicted
plt.figure(figsize=(6, 6))
plt.scatter(y_test, test_pred, alpha=0.6)
plt.xlabel("Actual CII")
plt.ylabel("Predicted CII")
plt.title(f"{best_model_name} Prediction")
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'r--')
plt.grid()
plt.tight_layout()
plt.show()

# Dictionary to store best models and their RMSEs
tuned_results = {}
best_models = {}

# 1. Linear Regression (no tuning)
lr_model = LinearRegression()
lr_model.fit(X_train_scaled, y_train)
lr_pred = lr_model.predict(X_test_scaled)
lr_rmse = np.sqrt(mean_squared_error(y_test, lr_pred))
tuned_results['Linear Regression'] = lr_rmse
best_models['Linear Regression'] = lr_model

# 2. Ridge Regression
ridge_params = {'alpha': [0.01, 0.1, 1, 10, 100]}
ridge_grid = GridSearchCV(Ridge(), ridge_params, cv=5,
scoring='neg_root_mean_squared_error')
ridge_grid.fit(X_train_scaled, y_train)
ridge_pred = ridge_grid.predict(X_test_scaled)
ridge_rmse = np.sqrt(mean_squared_error(y_test, ridge_pred))
tuned_results['Ridge Regression'] = ridge_rmse
best_models['Ridge Regression'] = ridge_grid.best_estimator_
```

```
# 3. Random Forest
rf_params = {'n_estimators': [50, 100], 'max_depth': [5, 10, None]}
rf_grid = GridSearchCV(RandomForestRegressor(random_state=0), rf_params, cv=5,
scoring='neg_root_mean_squared_error')
rf_grid.fit(X_train, y_train)   # RF doesn't need scaling
rf_pred = rf_grid.predict(X_test)
rf_rmse = np.sqrt(mean_squared_error(y_test, rf_pred))
tuned_results['Random Forest'] = rf_rmse
best_models['Random Forest'] = rf_grid.best_estimator_

# 4. SVR
svr_params = {'C': [0.1, 1, 10], 'gamma': ['scale', 0.01, 0.1]}
svr_grid = GridSearchCV(SVR(), svr_params, cv=5, scoring='neg_root_mean_squared_error')
svr_grid.fit(X_train_scaled, y_train)
svr_pred = svr_grid.predict(X_test_scaled)
svr_rmse = np.sqrt(mean_squared_error(y_test, svr_pred))
tuned_results['SVR'] = svr_rmse
best_models['SVR'] = svr_grid.best_estimator_

# 5. XGBoost
xgb_params = {'n_estimators': [50, 100], 'max_depth': [3, 5], 'learning_rate': [0.01, 0.1]}
xgb_grid = GridSearchCV(XGBRegressor(random_state=0), xgb_params, cv=5,
scoring='neg_root_mean_squared_error')
xgb_grid.fit(X_train, y_train)   # XGB can work unscaled
xgb_pred = xgb_grid.predict(X_test)
xgb_rmse = np.sqrt(mean_squared_error(y_test, xgb_pred))
tuned_results['XGBoost'] = xgb_rmse
best_models['XGBoost'] = xgb_grid.best_estimator_

mae_scores = {}
r2_scores = {}

for name, model in best_models.items():
    if name == 'Random Forest' or name == 'XGBoost':
        preds = model.predict(X_test)
    else:
        preds = model.predict(X_test_scaled)

    # These are additional scores beyond RMSE
    mae_scores[name] = mean_absolute_error(y_test, preds)
    r2_scores[name] = r2_score(y_test, preds)

summary_df = pd.DataFrame({
    'RMSE': tuned_results,
    'MAE': mae_scores,
    'R2 Score': r2_scores
}).T.round(4)

print("¥n Model Performance Metrics:")
print(summary_df)

#Train Final Ridge Model
ridge_final = Ridge(alpha=1.0)   # Use best alpha from previous tuning
ridge_final.fit(X_train_scaled, y_train)
y_pred = ridge_final.predict(X_test_scaled)
full_scaled = scaler.transform(daily_df[features])   # reuse the same scaler used during
training
daily_df['Predicted_CII'] = ridge_final.predict(full_scaled)

#Evaluation/verification
rmse = np.sqrt(mean_squared_error(y_test, y_pred))
mae = mean_absolute_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print("Final Ridge Model Performance:")
print(f"RMSE: {rmse:.4f}")
print(f"MAE:  {mae:.4f}")
print(f"R2:   {r2:.4f}")
```

```
#Actual vs Predicted Plot
plt.figure(figsize=(6,6))
plt.scatter(y_test, y_pred, alpha=0.6)
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'r--')
plt.xlabel("Actual CII")
plt.ylabel("Predicted CII")
plt.title("Ridge Regression: Actual vs Predicted CII")
plt.grid(True)
plt.tight_layout()
plt.show()

def assign_rating(predicted_cii):
    if predicted_cii <= 15.75:
        return 'A'
    elif predicted_cii <= 18.20:
        return 'B'
    elif predicted_cii <= 20.45:
        return 'C'
    elif predicted_cii <= 22.70:
        return 'D'
    else:
        return 'E'

def generate_suggestion(row):
    suggestions = []

    if row['Avg_Trim'] > 1.0:
        suggestions.append("Reduce trim to improve fuel efficiency")
    if row['Avg_Heel'] > 0.5:
        suggestions.append("Balance ballast to reduce heel")
    if row['Avg_Wind_Speed'] > 12:
        suggestions.append("Avoid sailing during high wind")
    if row['Avg_CppPitch'] < 15:
        suggestions.append("Increase CPP pitch for propulsion efficiency")
    if row['Avg_Speed'] < 17:
        suggestions.append("Maintain optimal cruising speed")

    return "; ".join(suggestions) if suggestions else "Performance is within expected range"

# Apply
daily_df['Optimization_Suggestions'] = daily_df.apply(generate_suggestion, axis=1)

daily_df[['Day', 'Predicted_CII', 'Relative_Rating',
'Optimization_Suggestions']].to_csv("cii_recommendations.csv", index=False)

use_manual_input = input("Use manual input? (yes/no): ").strip().lower()
if use_manual_input == 'yes':
    print("Enter today's average values:")
    row = pd.DataFrame([{
        'CO2_Emission_g': float(input("CO2 Emission (g): ")),
        'Distance_NM': float(input("Distance (NM): ")),
        'Avg_Speed': float(input("Avg Speed: ")),
        'Avg_Wind_Speed': float(input("Wind Speed: ")),
        'Avg_CppPitch': float(input("CPP Pitch: ")),
        'Avg_Heel': float(input("Heel: ")),
        'Avg_Trim': float(input("Trim: ")),
        'Avg_Draft': float(input("Avg Draft: "))
    }])
    row_scaled = scaler.transform(row)
    pred = ridge_final.predict(row_scaled)[0]
    print(f"Predicted CII: {pred:.4f} | Rating: {assign_rating(pred)}")
    print("Suggestions:", generate_suggestion(row.iloc[0]))
else:
    file_path = input("CSV file path: ").strip()
    data = pd.read_csv(file_path)
    preds = ridge_final.predict(scaler.transform(data[features]))
    data['Predicted_CII'] = preds
    data['Rating'] = data['Predicted_CII'].apply(assign_rating)
    data['Suggestions'] = data.apply(generate_suggestion, axis=1)
    print(data[['Predicted_CII', 'Rating', 'Suggestions']])
    data.to_csv("cii_predictions_output.csv", index=False)
    print("Saved to cii_predictions_output.csv")

import joblib
joblib.dump(ridge_final, 'ridge_final_model.joblib')
joblib.dump(scaler, 'scaler.joblib')
```

[16-2] Appendix B: Real-Time CII Calculation Model – Full Code

```
# === IMPORTS ===
import pandas as pd
import time
import numpy as np

# === CONSTANTS ===
GT = 14052   # Gross tonnage of the ship
FUEL_DENSITY = 0.991   # kg/L
CO2_FACTOR = 3.114   # g CO2 per gram of HFO

# === FEATURES USED FOR CALCULATION ===
features = [
    'FO_ME_Cons', 'FO_GE_Cons', 'Ship_Speed', 'CppPitch', 'Wind_Speed',
    'HEEL', 'Fore_Draft', 'Aft_Draft'
]

# === INPUT CSV FILE ===
csv_path = input("Enter path to minute-wise CSV file: ").strip()
data = pd.read_csv(csv_path)

# === CLEAN AND PREPROCESS FUNCTION ===
def preprocess(row, prev_row):
```

```
try:
    # Convert fuel readings to numeric and compute fuel diff
    fuel_me = max(float(row['FO_ME_Cons']) - float(prev_row['FO_ME_Cons']), 0)
    fuel_ge = max(float(row['FO_GE_Cons']) - float(prev_row['FO_GE_Cons']), 0)
    fuel_liters = fuel_me + fuel_ge

    # Trim and average draft
    trim = float(row['Aft_Draft']) - float(row['Fore_Draft'])
    avg_draft = (float(row['Aft_Draft']) + float(row['Fore_Draft'])) / 2

    # CO2 and Distance
    ship_speed = float(row['Ship_Speed'])
    distance_nm = ship_speed / 60   # per minute
    fuel_kg = fuel_liters * FUEL_DENSITY
    co2_emission_g = fuel_kg * 1000 * CO2_FACTOR

    # Instantaneous CII Calculation
    if GT * distance_nm == 0:
        cii = np.nan
    else:
        cii = co2_emission_g / (GT * distance_nm)

    return {
        'CO2_Emission_g': co2_emission_g,
        'Distance_NM': distance_nm,
        'CII': cii,
        'Avg_Speed': ship_speed,
        'Avg_Trim': trim,
        'Avg_Heel': float(row['HEEL']),
        'Avg_Wind_Speed': float(row['Wind_Speed']),
        'Avg_CppPitch': float(row['CppPitch']),
        'Avg_Draft': avg_draft
    }
except Exception as e:
        print(f"Error processing row: {e}")
        return {
            'CO2_Emission_g': np.nan,
            'Distance_NM': np.nan,
            'CII': np.nan,
            'Avg_Speed': np.nan,
            'Avg_Trim': np.nan,
            'Avg_Heel': np.nan,
            'Avg_Wind_Speed': np.nan,
            'Avg_CppPitch': np.nan,
            'Avg_Draft': np.nan
        }

# === OPTIMIZATION SUGGESTION FUNCTION ===
def generate_suggestion(row):
    suggestions = []
    if row['Avg_Trim'] > 1.0:
        suggestions.append("Reduce trim to improve fuel efficiency")
    if row['Avg_Heel'] > 0.5:
        suggestions.append("Balance ballast to reduce heel")
    if row['Avg_Wind_Speed'] > 12:
        suggestions.append("Avoid sailing during high wind")
    if row['Avg_CppPitch'] < 15:
        suggestions.append("Increase CPP pitch for propulsion efficiency")
    if row['Avg_Speed'] < 17:
        suggestions.append("Maintain optimal cruising speed")
    return "; ".join(suggestions) if suggestions else "Performance is within expected range"

# === LIVE SIMULATION ===
print("¥n--- LIVE CII CALCULATION STARTED ---")
for i in range(1, len(data)):
    current_row = data.iloc[i]
    previous_row = data.iloc[i - 1]
    result = preprocess(current_row, previous_row)

    cii = result.get('CII', np.nan)
    suggestions = generate_suggestion(result)

    if not np.isnan(cii):
        print(f"Minute {i}: Instant CII = {cii:.4f} | Suggestions: {suggestions}")
    else:
        print(f"Minute {i}: CII could not be calculated due to zero distance.")

    time.sleep(1)

print("¥n--- LIVE CII SIMULATION COMPLETE ---")
```