

# Configuration Manual

MSc Research Project  
Msc Data Analytics

Sushmita Gupta  
Student ID: x22219455

School of Computing  
National College of Ireland

Supervisor: Dr. Anderson Simiscuka

**National College of Ireland**  
**MSc Project Submission Sheet**  
**School of Computing**



**Student Name:** Sushmita Ghanshyam Gupta  
.....  
x22219455  
**Student ID:** .....  
MSc Data Analytics 2023 – 2024  
**Programme:** ..... **Year:** .....  
MSc Research Project  
**Module:** .....  
Dr Anderson Simiscuka  
**Supervisor:** .....  
**Submission Due Date:** 12-08-2024  
.....  
Configuration Manual  
**Project Title:** .....  
774  
**Word Count:** .....

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

**Signature:** Sushmita Ghanshyam Gupta  
.....  
12-08-2024  
**Date:** .....

**PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST**

Attach a completed copy of this sheet to each project (including multiple copies)	<input type="checkbox"/>
<b>Attach a Moodle submission receipt of the online project submission,</b> to each project (including multiple copies).	<input type="checkbox"/>
<b>You must ensure that you retain a HARD COPY of the project,</b> both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	<input type="checkbox"/>

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

<b>Office Use Only</b>	
Signature:	
Date:	
Penalty Applied (if applicable):	

# Configuration Manual

Sushmita Ghanshyam Gupta

X22219455

## 1 Introduction

This document provides a step-by-step guide to implementing the models built to solve the research question: *"How effective is Ant Colony Optimization (ACO) compared to Genetic Algorithms (GA) & Hybrid Models such as ACO-DRL in providing more efficient solutions for improving logistics routes to reduce CO2 emissions & fuel consumption while maintaining logistics efficiency?"* It addresses the hardware and software specifications of the study, environment, and system setting necessary for the research. The workflow of the three models, namely ACO, GA and ACO-DRL, is explained in this manual.

## 2 System Configuration

### 2.1 Hardware Requirements

- **Machine:** Windows 11 and above
- **Processor:** Intel Core i5 (12th Gen) or above
- **RAM:** 8 GB (Minimum)
- **Storage:** 16GB and above

### 2.2 Software Requirements

- **Python Version:** Python 3.11
- **Integrated Development Environment (IDE):** Jupyter Notebook
- **Package Manager:** Anaconda Navigator
- **Microsoft Excel:** For initial EDA

## 3 Environment Setup

### 3.1 Installing Anaconda

- **Download Anaconda** from [Anaconda Official Website](https://www.anaconda.com/) the installer for Windows.

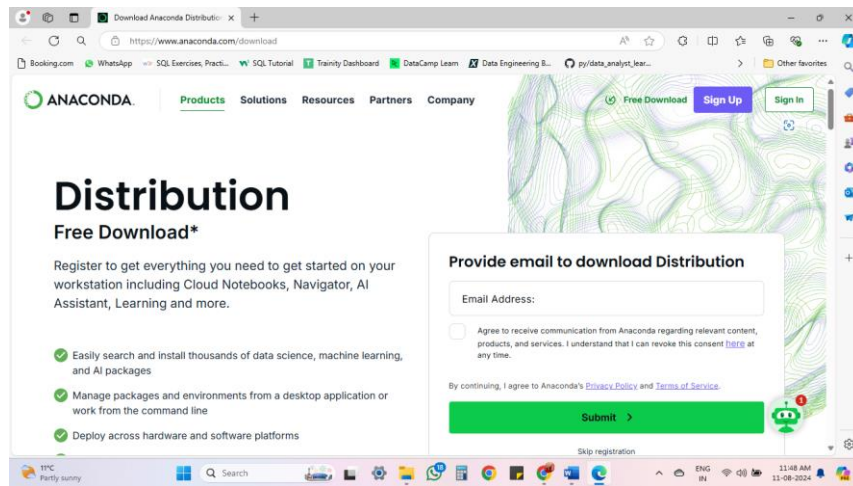


Figure 1 (a) : Anaconda Website

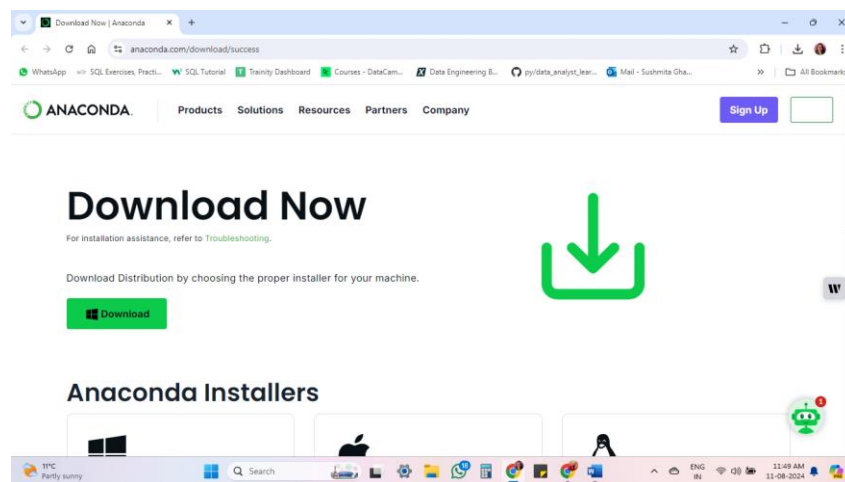


Figure 1 (b) : Anaconda Website

- **To Install** Run the installer and follow the instructions. add Anaconda to system's PATH.

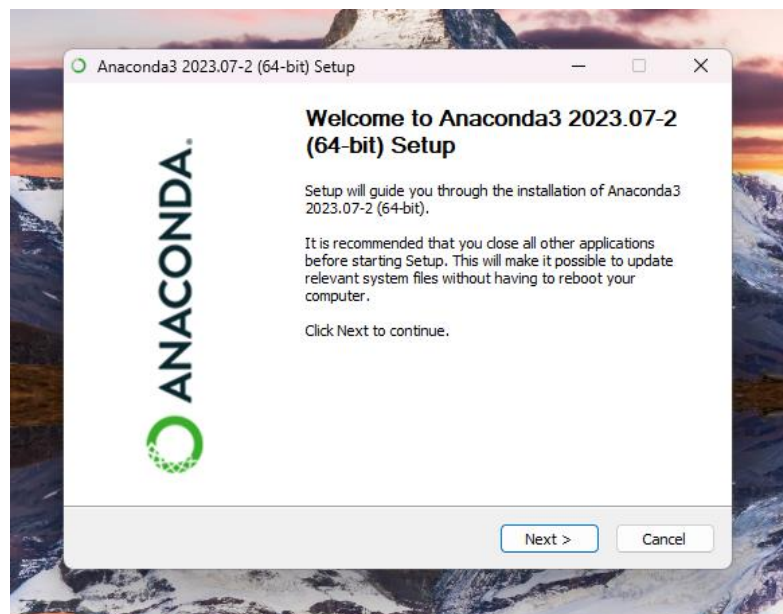


Figure 2 (a) : Anaconda Setup

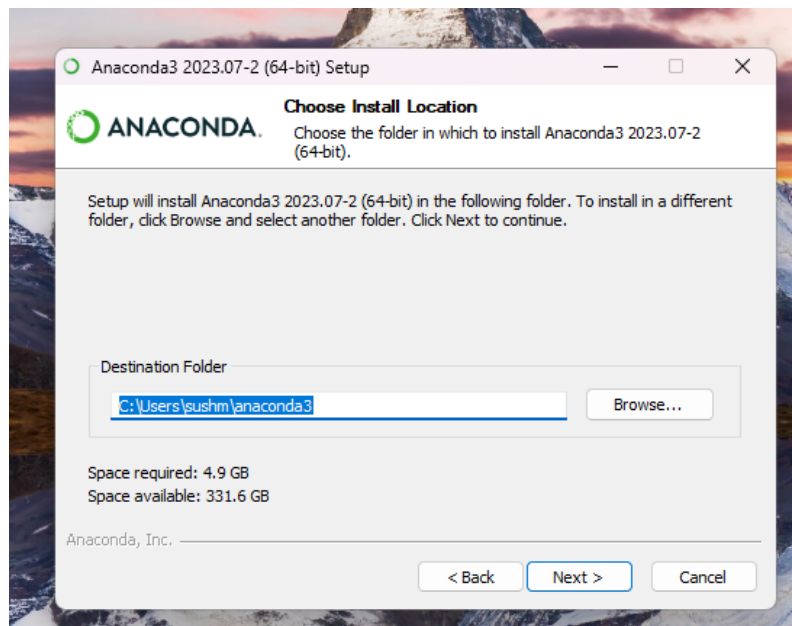


Figure 2 (b) : Anaconda Setup

- **Open Anaconda Navigator:** to manage your Python environments. **Launch Jupyter Notebook:** Use Anaconda cmd and launch Jupyter
- 

```

Anaconda Prompt - jupyter n x
+ v

(base) C:\Users\sushm>jupyter notebook
[I 2024-08-11 11:52:26.437 ServerApp] Extension package jupyter_server_ydoc took 0.3878s to import
[W 2024-08-11 11:52:27.079 ServerApp] A '_jupyter_server_extension_points' function was not found in nbclassic. Instead,
a '_jupyter_server_extension_paths' function was found and will be used for now. This function name will be deprecated
in future releases of Jupyter Server.
[W 2024-08-11 11:52:27.079 ServerApp] A '_jupyter_server_extension_points' function was not found in notebook_shim. Inst
ead, a '_jupyter_server_extension_paths' function was found and will be used for now. This function name will be depreca
ted in future releases of Jupyter Server.
[I 2024-08-11 11:52:30.848 ServerApp] Extension package panel.io.jupyter_server_extension took 3.7619s to import
[I 2024-08-11 11:52:30.849 ServerApp] jupyter_lsp | extension was successfully linked.
[I 2024-08-11 11:52:30.857 ServerApp] jupyter_server_fileid | extension was successfully linked.
[I 2024-08-11 11:52:30.858 ServerApp] jupyter_server_terminals | extension was successfully linked.
[I 2024-08-11 11:52:30.872 ServerApp] jupyter_server_ydoc | extension was successfully linked.
[I 2024-08-11 11:52:30.878 ServerApp] jupyterlab | extension was successfully linked.
[I 2024-08-11 11:52:30.890 ServerApp] nbclassic | extension was successfully linked.
[I 2024-08-11 11:52:30.898 ServerApp] notebook | extension was successfully linked.
[I 2024-08-11 11:52:31.908 ServerApp] notebook_shim | extension was successfully linked.
[I 2024-08-11 11:52:31.908 ServerApp] panel.io.jupyter_server_extension | extension was successfully linked.
[I 2024-08-11 11:52:31.988 ServerApp] notebook_shim | extension was successfully loaded.
[I 2024-08-11 11:52:31.988 ServerApp] jupyter_lsp | extension was successfully loaded.
[I 2024-08-11 11:52:31.988 FileIdExtension] Configured File ID manager: ArbitraryFileIdManager
[I 2024-08-11 11:52:31.996 FileIdExtension] ArbitraryFileIdManager : Configured root dir: C:/Users/sushm
[I 2024-08-11 11:52:31.996 FileIdExtension] ArbitraryFileIdManager : Configured database path: C:\Users\sushm\AppData\Ro
aming\jupyter\file_id_manager.db
[I 2024-08-11 11:52:31.999 FileIdExtension] ArbitraryFileIdManager : Successfully connected to database file.
[I 2024-08-11 11:52:32.008 FileIdExtension] ArbitraryFileIdManager : Creating File ID tables and indices with journal_mo
de = DELETE
[I 2024-08-11 11:52:32.012 FileIdExtension] Attached event listeners.
[I 2024-08-11 11:52:32.017 ServerApp] jupyter_server_fileid | extension was successfully loaded.
```

Figure 3 (a) : Anaconda Prompt

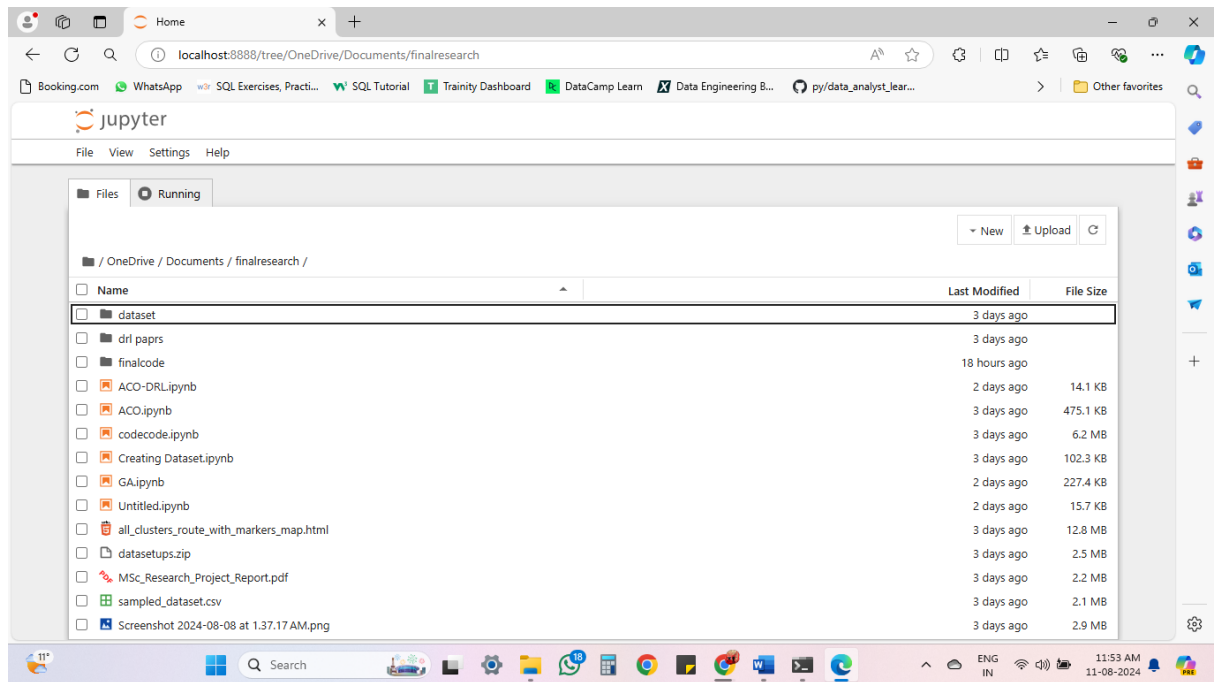


Figure 3 (b) : Anaconda Prompt

## 3.2 Installing Important Libraries

The following Python libraries are essential for executing the models in this research:

- NumPy: for numerical computations
- Pandas: for data manipulation and analysis
- SciPy: for scientific and technical computing
- Scikit-learn: for machine learning algorithms
- Matplotlib: for data visualization
- OpenAI Gym: To create and to work the TSP (Traveling Salesman Problem) environment

To install these libraries, open the Anaconda command prompt and run the following command:

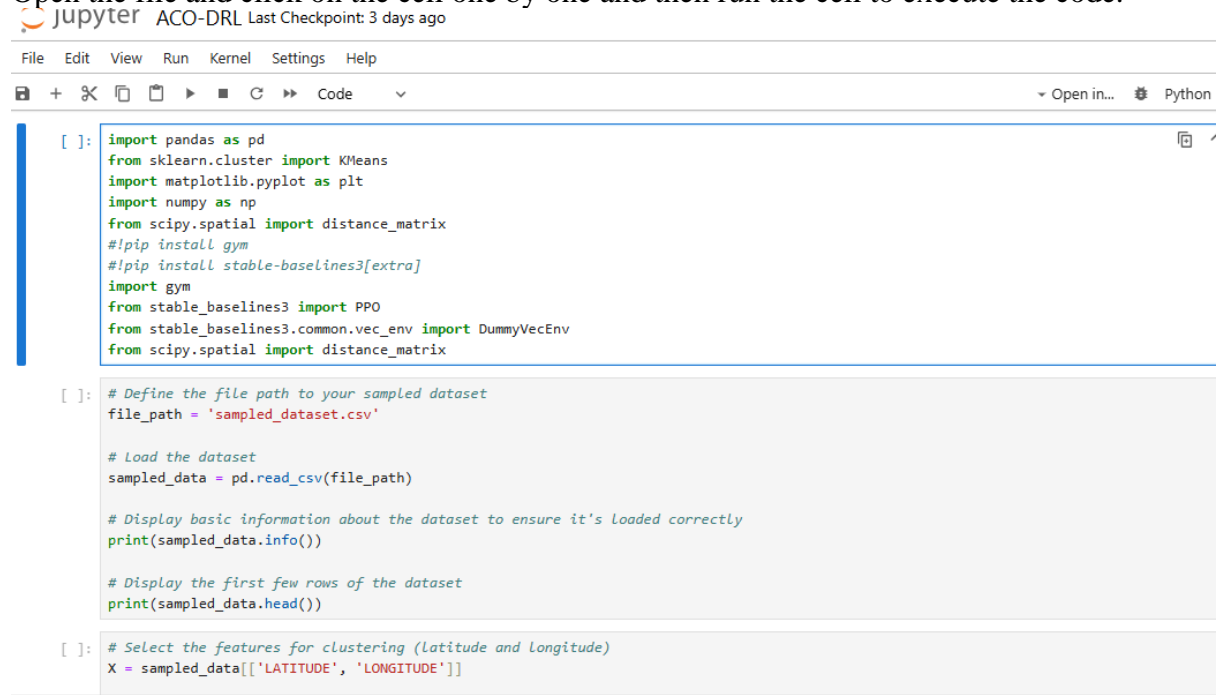
```
!pip install numpy pandas scipy scikit-learn matplotlib gym
```

```
[1]: !pip install numpy pandas scipy scikit-learn matplotlib gym

Requirement already satisfied: numpy in c:\users\sushm\anaconda3\lib\site-packages (1.24.3)
Requirement already satisfied: pandas in c:\users\sushm\anaconda3\lib\site-packages (1.5.3)
Requirement already satisfied: scipy in c:\users\sushm\anaconda3\lib\site-packages (1.10.1)
Requirement already satisfied: scikit-learn in c:\users\sushm\anaconda3\lib\site-packages (1.3.0)
Requirement already satisfied: matplotlib in c:\users\sushm\anaconda3\lib\site-packages (3.7.1)
Requirement already satisfied: gym in c:\users\sushm\anaconda3\lib\site-packages (0.26.2)
Requirement already satisfied: python-dateutil>=2.8.1 in c:\users\sushm\anaconda3\lib\site-packages (from pandas) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in c:\users\sushm\anaconda3\lib\site-packages (from pandas) (2022.7)
Requirement already satisfied: joblib>=1.1.1 in c:\users\sushm\anaconda3\lib\site-packages (from scikit-learn) (1.2.0)
Requirement already satisfied: threadpoolctl>=2.0.0 in c:\users\sushm\anaconda3\lib\site-packages (from scikit-learn) (2.2.0)
Requirement already satisfied: contourpy>=1.0.1 in c:\users\sushm\anaconda3\lib\site-packages (from matplotlib) (1.0.5)
Requirement already satisfied: cycler>=0.10 in c:\users\sushm\anaconda3\lib\site-packages (from matplotlib) (0.11.0)
Requirement already satisfied: fonttools>=4.22.0 in c:\users\sushm\anaconda3\lib\site-packages (from matplotlib) (4.25.0)
Requirement already satisfied: kiwisolver>=1.0.1 in c:\users\sushm\anaconda3\lib\site-packages (from matplotlib) (1.4.4)
Requirement already satisfied: packaging>=20.0 in c:\users\sushm\anaconda3\lib\site-packages (from matplotlib) (23.0)
Requirement already satisfied: pillow>=6.2.0 in c:\users\sushm\anaconda3\lib\site-packages (from matplotlib) (9.4.0)
Requirement already satisfied: pyparsing>=2.3.1 in c:\users\sushm\anaconda3\lib\site-packages (from matplotlib) (3.0.9)
Requirement already satisfied: cloudpickle>=1.2.0 in c:\users\sushm\anaconda3\lib\site-packages (from gym) (2.2.1)
Requirement already satisfied: gym-notices>=0.0.4 in c:\users\sushm\anaconda3\lib\site-packages (from gym) (0.0.8)
Requirement already satisfied: six>=1.5 in c:\users\sushm\anaconda3\lib\site-packages (from python-dateutil>=2.8.1->pandas) (1.16.0)
```

Figure 4: Importing required libraries

Open the file and click on the cell one by one and then run the cell to execute the code.



```
[ ]: import pandas as pd
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt
import numpy as np
from scipy.spatial import distance_matrix
#!pip install gym
#!pip install stable-baselines3[extra]
import gym
from stable_baselines3 import PPO
from stable_baselines3.common.vec_env import DummyVecEnv
from scipy.spatial import distance_matrix

[ ]: # Define the file path to your sampled dataset
file_path = 'sampled_dataset.csv'

# Load the dataset
sampled_data = pd.read_csv(file_path)

# Display basic information about the dataset to ensure it's loaded correctly
print(sampled_data.info())

# Display the first few rows of the dataset
print(sampled_data.head())

[ ]: # Select the features for clustering (Latitude and Longitude)
X = sampled_data[['LATITUDE', 'LONGITUDE']]
```

Figure 5: Required libraries

## 4 Data Gathering & Pre-processing

This section focuses on data collection and data preparation processes so that replication of the research study can be done. The data set is from Kaggle link :

<https://www.kaggle.com/datasets/thedevastator/ups-facilities-locations-dataset>

### Downloading the Datasets

Make sure that the obtained dataset is stored in CSV format in the directory of the local computer.

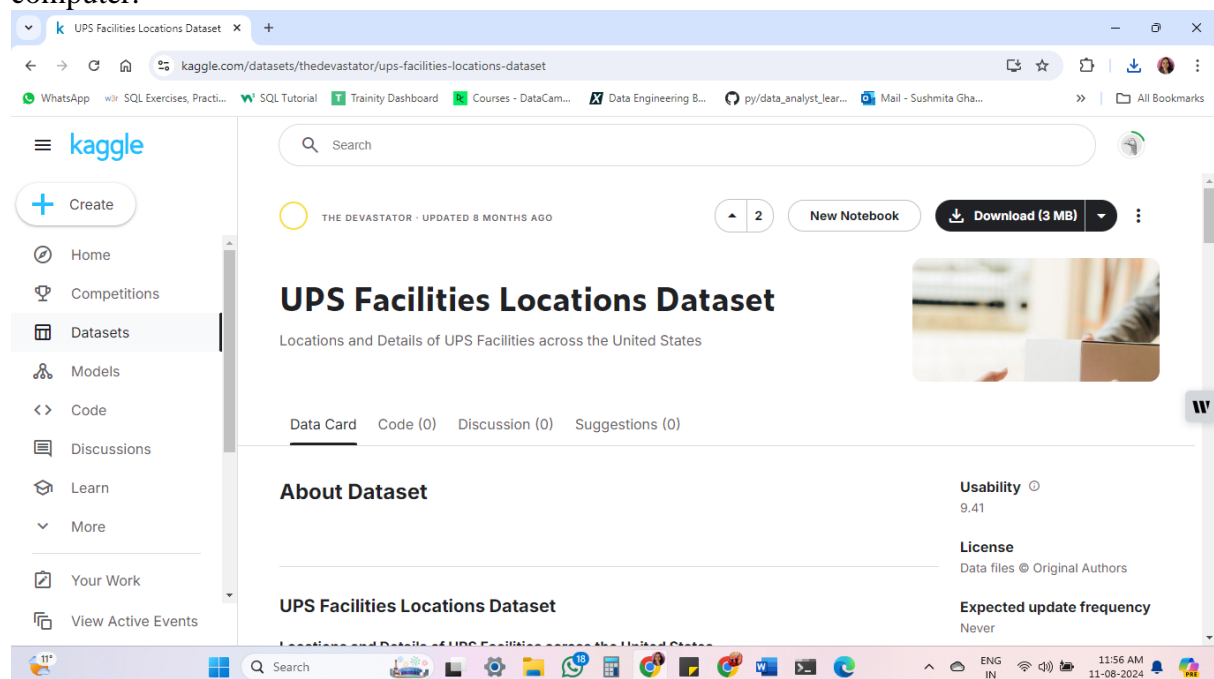




Figure 6: UPS Kaggle Dataset

## 4.1 Data Collection

### A) Data Sources:

- **UPS Facilities Data:** Geographical information of the location of warehouses and drop boxes such as latitude and longitude.

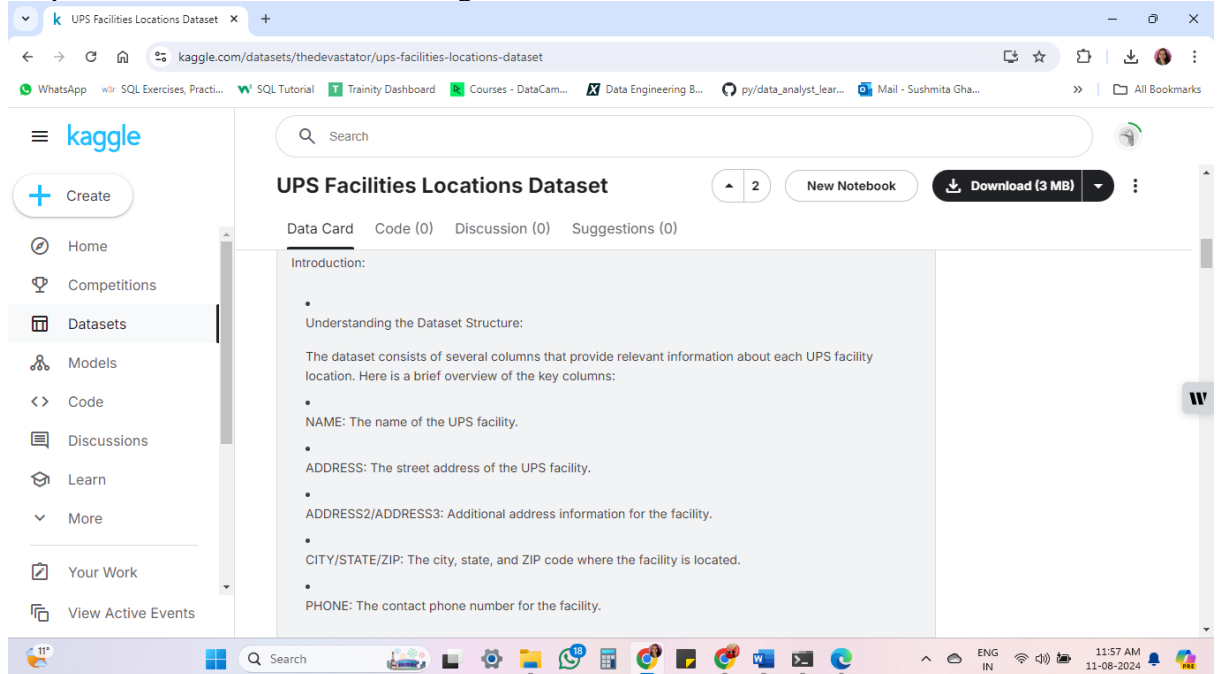


Figure 7: UPS meta data

- **Steps to Load and Prepare Data:**

**Load Data:** Make sure the dataset is in CSV format, and import it into Jupyter Notebook and **Display Basic Information:** Find out more about the data set:

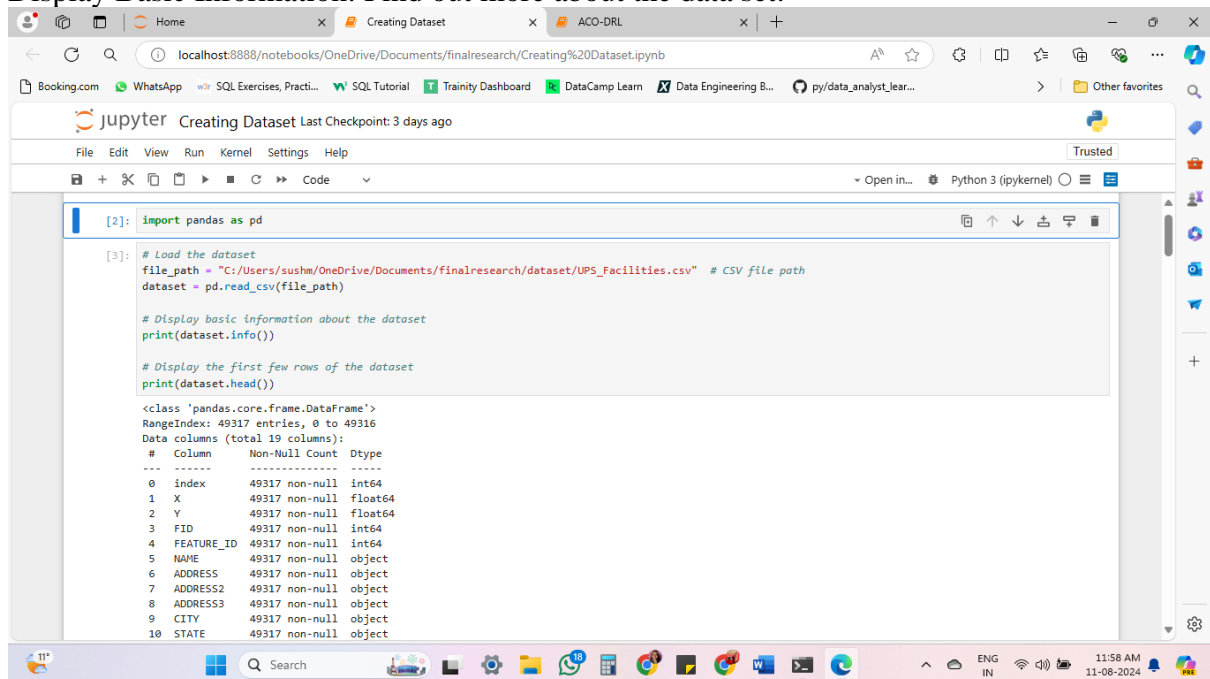




Figure 8: Loading Dataset

## 4.2 Data Filtering

- Remove Unwanted Entries: Exclusion of entries from Hawaii and Alaska because they could skew the results:

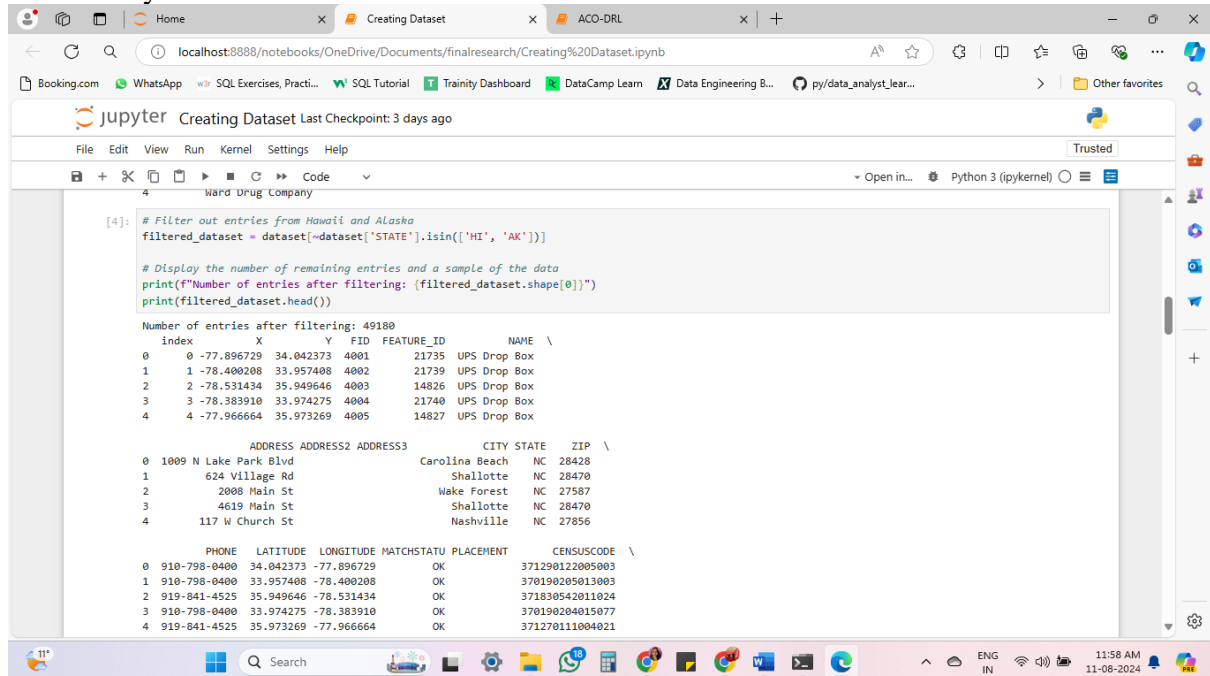


Figure 9: Data Filtering

- Filter by Facility Type: Retain only those rows that are to “Authorized UPS Facility” or “UPS Drop Box”.

```
# Filter the dataset to keep only rows with "Authorized UPS Facility" or "UPS Drop Box" in the 'NAME' column
filtered_facilities = filtered_dataset[filtered_dataset['NAME'].isin(['Authorized Shipping Outlet', 'UPS Drop Box'])]

# Display the number of remaining entries and a sample of the data
print(f"Number of entries after filtering: {filtered_facilities.shape[0]}")
print(filtered_facilities.head())
```

Figure 10: Refining Rows

## 4.3 Data Cleaning

- Remove Missing Values: One of the recommendations for the dataset is to clean the data by deleting the rows with missing values.

```
# Remove rows with missing values
cleaned_dataset = filtered_facilities.dropna()

# Display the number of remaining entries after removing missing values
print(f"Number of entries after removing missing values: {cleaned_dataset.shape[0]}")

Number of entries after removing missing values: 42297
```

Figure 11: Remove missing values

2. Outlier Removal: Apply the Interquartile Range (IQR) technique to Trim latitude and longitude data:

```
] : # Define a function to calculate IQR and filter out outliers
def remove_outliers_iqr(df, column):
    Q1 = df[column].quantile(0.25)
    Q3 = df[column].quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR
    return df[(df[column] >= lower_bound) & (df[column] <= upper_bound)]

# Apply the IQR method to the Latitude and Longitude columns
filtered_cleaned_dataset = remove_outliers_iqr(cleaned_dataset, 'LATITUDE')
filtered_cleaned_dataset = remove_outliers_iqr(filtered_cleaned_dataset, 'LONGITUDE')

# Display the number of remaining entries after removing outliers using IQR
print(f"Number of entries after removing outliers using IQR: {filtered_cleaned_dataset.shape[0]}")

Number of entries after removing outliers using IQR: 42248
```

Figure 12: Making the dataset short

## 4.4 Sampling

1. Proportional Sampling: To make sure the data collected is a sample of the entire population, the proportions have to be calculated and the samples obtained as follows;

```
# Define the target number of entries
target_entries = 12000

# Calculate the total entries to ensure proportional sampling
total_entries = filtered_cleaned_dataset.shape[0]

# Calculate the proportion of each type within each state
proportions = filtered_cleaned_dataset.groupby(['STATE', 'NAME']).size().reset_index(name='count')
proportions['proportion'] = proportions['count'] / total_entries

# Calculate target samples per group based on the proportions
proportions['target_count'] = (proportions['proportion'] * target_entries).round().astype(int)

# Sample the data according to the target counts
sampled_data = pd.DataFrame()

for _, row in proportions.iterrows():
    subset = filtered_cleaned_dataset[
        (filtered_cleaned_dataset['STATE'] == row['STATE']) &
        (filtered_cleaned_dataset['NAME'] == row['NAME'])
    ]
    sampled_subset = subset.sample(n=row['target_count'], random_state=42) # Use a fixed random_state
    sampled_data = pd.concat([sampled_data, sampled_subset])

# Shuffle the final sampled dataset
sampled_data = sampled_data.sample(frac=1, random_state=42).reset_index(drop=True)

# Display the number of entries in the sampled dataset
```

Figure 13: Sampling Dataset

2. Save Sampled Dataset: Export the last sampled dataset to the CSV file:

Total number of entries in the sampled dataset: 12004

```
|: # Define the file path where you want to save the CSV
output_file_path = 'sampled_dataset.csv'

# Save the sampled dataset to a CSV file
sampled_data.to_csv(output_file_path, index=False)

print(f"Sampled dataset saved as {output_file_path}")
```

Sampled dataset saved as sampled\_dataset.csv

Figure 13: Saving sampled dataset Dataset

### Clustering of the sampled dataset

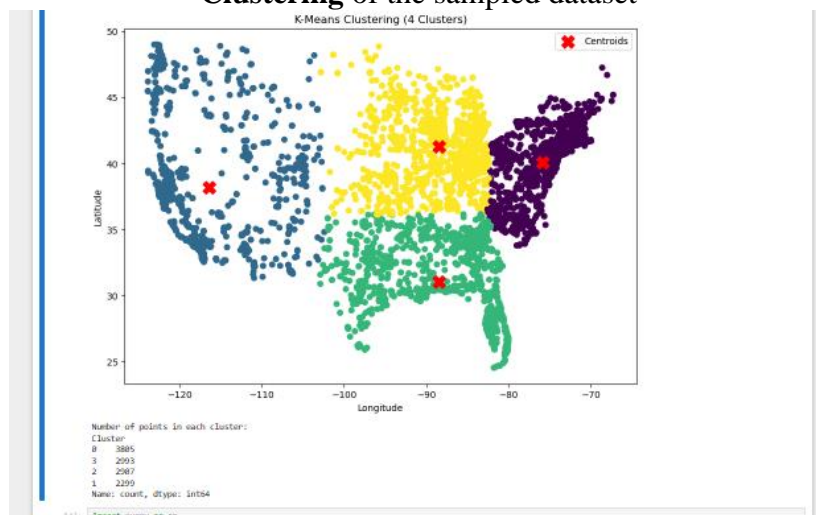


Figure 14: Clustering Dataset

```
[3]: # Select the features for clustering (Latitude and Longitude)
X = sampled_data[['LATITUDE', 'LONGITUDE']]

# Apply K-Means clustering with 4 clusters
kmeans = KMeans(n_clusters=4, random_state=42)
sampled_data['Cluster'] = kmeans.fit_predict(X)

# Centroids of the clusters
centroids = kmeans.cluster_centers_

# Visualize the clusters
plt.figure(figsize=(10, 7))
plt.scatter(sampled_data['LONGITUDE'], sampled_data['LATITUDE'], c=sampled_data['Cluster'], cmap='viridis', marker='o')
plt.scatter(centroids[:, 1], centroids[:, 0], c='red', marker='x', s=200, label='Centroids')
plt.title('K-Means Clustering (4 Clusters)')
plt.xlabel('Longitude')
plt.ylabel('Latitude')
plt.legend()
plt.show()

# Display the number of points in each cluster
cluster_counts = sampled_data['Cluster'].value_counts()
print("Number of points in each cluster:")
print(cluster_counts)
```

Figure 15: k-means clustering the Dataset

- **Model Configuration**

### 5.1 ACO Model Setup

#### Steps:

In the case of the ACO algorithm, the shortest paths within each cluster are obtained.

#### Steps:

##### 1. Import Necessary Libraries:

```
[ ]: import pandas as pd
      from sklearn.cluster import KMeans
      import matplotlib.pyplot as plt
      import numpy as np
      from scipy.spatial import distance_matrix
      #!/pip install gym
      #!/pip install stable-baselines3[extra]
      import gym
      from stable_baselines3 import PPO
      from stable_baselines3.common.vec_env import DummyVecEnv
      from scipy.spatial import distance_matrix
```

Figure 16

##### 2. Define ACO Parameters:

```
# Apply ACO on this cluster's data
n_ants = 10
n_best = 3
n_iterations = 100
decay = 0.95
```

Figure 17

### 3. Run ACO Algorithm:

```
return patn

def pick_move(self, pheromone, dist, visited):
    pheromone = np.copy(pheromone)
    pheromone[list(visited)] = 0
    with np.errstate(divide='ignore', invalid='ignore'):
        row = pheromone ** self.alpha * ((1.0 / dist) ** self.beta)
        row = safe_normalize(row) # Ensure the row is normalized and valid
    move = np.random.choice(self.all_inds, 1, p=row)[0]
    return move

def calculate_path_distance(self, path):
    total_dist = 0
    for ele in path:
        total_dist += self.dist_matrix[ele]
    return total_dist

try:
    aco = ACO(dist_matrix, n_ants, n_best, n_iterations, decay, alpha=1, beta=2)
    shortest_path = aco.run()

    print(f"Cluster {cluster_id}: Shortest path: {shortest_path[0]}")
    print(f"Cluster {cluster_id}: Shortest path distance: {shortest_path[1]} km")
except ValueError as e:
    print(f"Error in cluster {cluster_id}: {e}")
```

Figure 18

- Record Results:.

```
[ ]: # Define constants
fuel_consumption_per_km = 0.12 # in liters per km
co2_emissions_factor = 2.31 # kg CO2 per liter of petrol

# Cluster distances
cluster_distances = [442.67743356278237, 431.07633811873967, 481.28795997749353, 564.8667002559663]

# Calculate CO2 emissions for each cluster
co2_emissions = [distance * fuel_consumption_per_km * co2_emissions_factor for distance in cluster_distances]

# Display CO2 emissions for each cluster
for i, emissions in enumerate(co2_emissions):
    print(f"Cluster {i}: Estimated CO2 Emissions: {emissions:.2f} kg")

# Sum distances for a single total distance
total_distance = sum(cluster_distances)

# Sum CO2 emissions for a single total value
total_co2_emissions = sum(co2_emissions)

print(f"Total Distance: {total_distance:.2f} km")
print(f"Total CO2 Emissions: {total_co2_emissions:.2f} kg")
```

Figure 19

## 5.2 GA Model Setup

**Overview:** The GA is used to apply the idea of evolution in order to find the best routes for distributing the logistics.

Steps:

1. Import Necessary Libraries:

```
[1]: import pandas as pd
      from sklearn.cluster import KMeans
      import matplotlib.pyplot as plt
      import numpy as np
      from scipy.spatial import distance_matrix
```

Figure 20

3. Run GA Iterations: Crossover and mutation should be used to develop the population and to identify the best routes. Record Best Route: Store the best solution found by the GA.

```
# Example: Apply GA on one of the clusters with fixed start and end points
for cluster_id in sampled_data['Cluster'].unique():
    cluster_data = sampled_data[sampled_data['Cluster'] == cluster_id]

    # Identify the warehouse and drop boxes within the cluster
    warehouse_index = cluster_data[cluster_data['NAME'] == 'Authorized Shipping Outlet'].index[0]
    drop_box_index = cluster_data[cluster_data['NAME'] == 'UPS Drop Box'].index[0]

    # Combine warehouse and drop boxes for the distance matrix calculation
    locations = cluster_data[['LATITUDE', 'LONGITUDE']].values
    dist_matrix = distance_matrix(locations, locations)

    # Apply GA on this cluster's data
    best_route, best_distance = genetic_algorithm_fixed_start_end(
        dist_matrix, start_point=warehouse_index, end_point=drop_box_index,
        num_generations=100, population_size=50, mutation_rate=0.01
    )

    print(f"Cluster {cluster_id}: Shortest path distance using GA: {best_distance:.2f} km")
    print("Best route order of locations:", best_route)
```

Figure 21

### 5.3 Hybrid ACO-DRL Model Setup

**Overview:** The Hybrid ACO-DRL model integrates ACO's proficiency in identifying the first routes with DRL's adaptability for improvements.

Steps to Load and Prepare Data:

Model Setup

Overview:

1. **Initialize TSP Environment:** Set up the Traveling Salesman Problem (TSP) environment using OpenAI Gym:

```

print('Total CO2 Emissions: ', total_co2_emissions, 'kg')

[ ]: import gym
import numpy as np
class TSPEnv(gym.Env):
    def __init__(self, distance_matrix, initial_route):
        super(TSPEnv, self).__init__()
        self.distance_matrix = distance_matrix
        self.num_locations = len(distance_matrix)
        self.start_location = initial_route[0] # Starting point (warehouse)
        self.end_location = initial_route[-1] # Ending point (drop box)
        self.visited = [self.start_location] # Start with the warehouse index only
        self.current_location = self.start_location

        # Define action and observation space
        self.action_space = gym.spaces.Discrete(self.num_locations)
        self.observation_space = gym.spaces.Box(low=0, high=1, shape=(self.num_locations,), dtype=np.int32)

    def reset(self):
        self.visited = [self.start_location] # Reset to start from the warehouse
        self.current_location = self.start_location
        return self._get_observation()

    def _get_observation(self):

```

Figure 22

2. **Train DRL Model:** Use Proximal Policy Optimization (PPO) to refine ACO-derived routes. **Refine Routes:** Iterate through the environment to optimize the route  
**Evaluate and Save Results:** Compare refined routes with the original and save results.

```

# Process each cluster
for cluster_id in sampled_data['Cluster'].unique():
    cluster_data = sampled_data[sampled_data['Cluster'] == cluster_id]

    initial_route = shortest_path # Contains both the path and distance
    initial_route_indices = [pair[0] for pair in initial_route[0]]

    coordinates = cluster_data[['LATITUDE', 'LONGITUDE']].values
    dist_matrix = distance_matrix(coordinates, coordinates)

    env = DummyVecEnv([lambda: TSPEnv(dist_matrix, initial_route_indices)])

    model = PPO("MlpPolicy", env, verbose=1, learning_rate=1e-4, n_steps=2048, gamma=0.99)
    model.learn(total_timesteps=100000) # Increase timesteps

    obs = env.reset()
    for _ in range(100): # Increase iterations for a more thorough search
        action, _states = model.predict(obs)
        obs, rewards, dones, info = env.step(action)
        env.render()

    refined_route = env.envs[0].visited
    refined_distance = np.sum([dist_matrix[refined_route[i], refined_route[i+1]] for i in range(len(refined_route) - 1)])

    print(f"Cluster {cluster_id}: Shortest path distance using ACO + DRL: {refined_distance:.2f} km")
    print(f"Cluster {cluster_id}: Refined route order of locations: {refined_route}")

```

Figure 23

## 6. Model Evaluation

### 4. 6.1 Performance Metrics

The models are evaluated using the following metrics:

- **CO2 Emissions:** Calculate CO2 emissions based on distance and fuel consumption:  
 $\text{CO2\_Emissions} = \text{Distance\_km} * \text{Fuel\_Consumption\_per\_km} * \text{CO2\_Emission\_Factor}$  and  
**Total Distance:** Measure the total distance covered by the optimized routes.



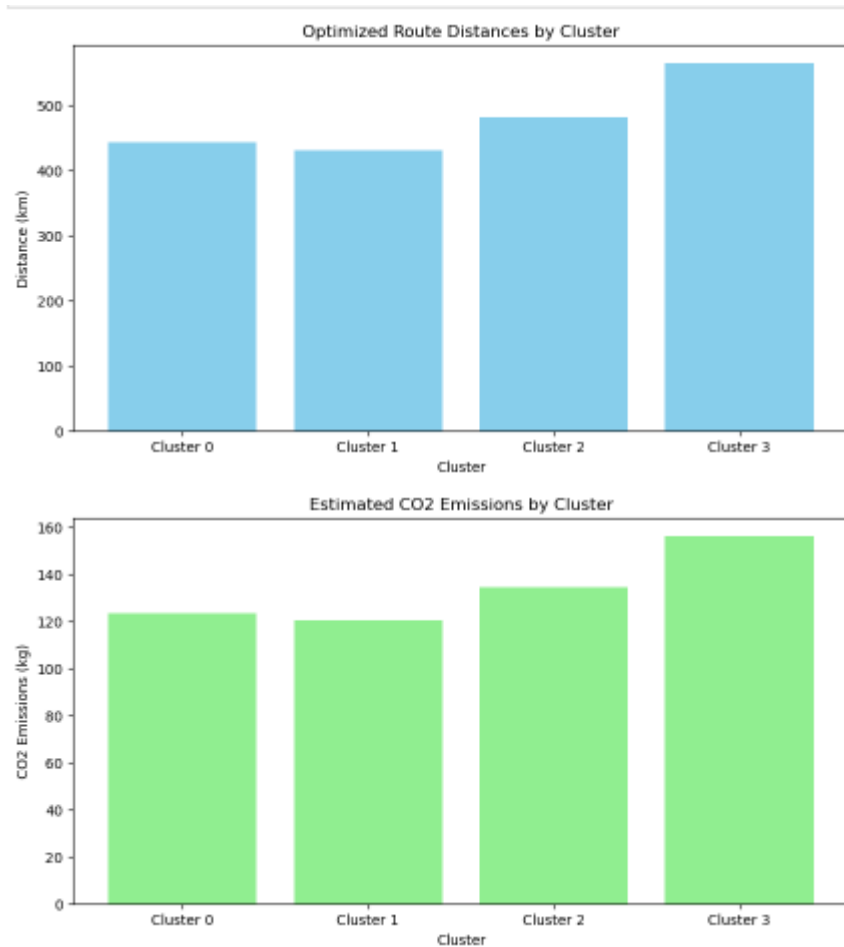


Figure 24

```
# Calculate CO2 emissions for each cluster
co2_emissions = [distance * fuel_consumption_per_km * co2_emissions_factor for distance in cluster_distances]

# Display CO2 emissions for each cluster
for i, emissions in enumerate(co2_emissions):
    print(f"Cluster {i}: Estimated CO2 Emissions: {emissions:.2f} kg")

# Sum distances for a single total distance
total_distance = sum(cluster_distances)

# Sum CO2 emissions for a single total value
total_co2_emissions = sum(co2_emissions)

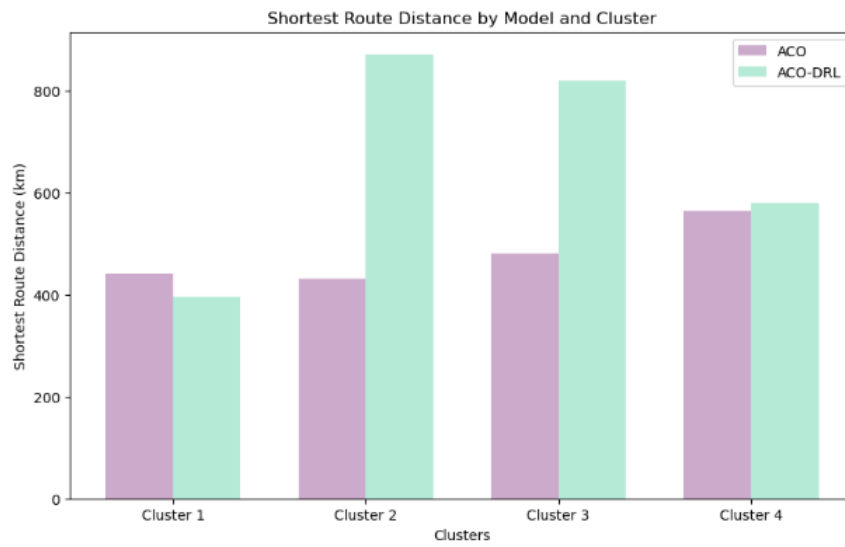
print(f"Total Distance: {total_distance:.2f} km")
print(f"Total CO2 Emissions: {total_co2_emissions:.2f} kg")
```

Figure 25

## 6.2 Visualization

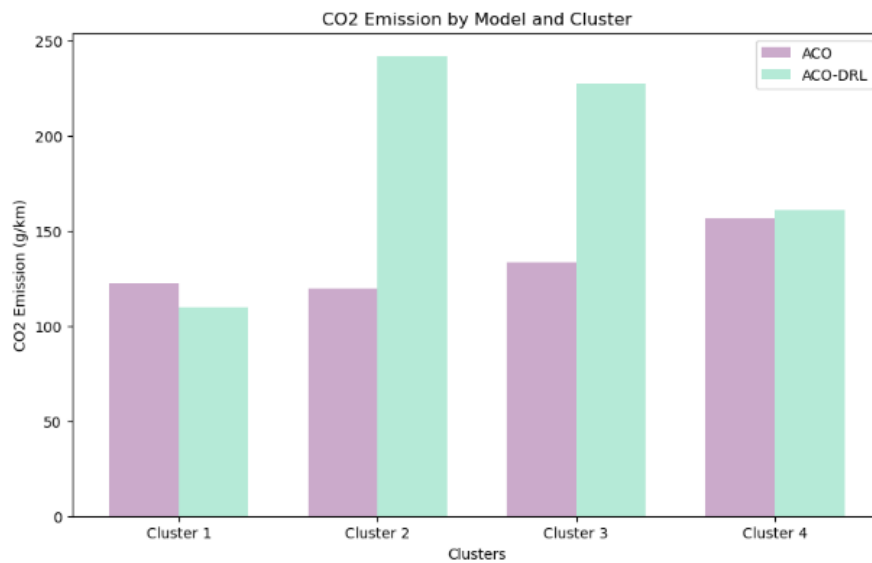
Visualize the results

**Bar Plots for Distances:**



**Figure 26**

- **Bar Plots for CO Emissions:**



**Figure 27**