In [ ]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error

# Load your dataset
df = pd.read_csv(r"C:\Users\MANICKA MEENAKSHI.S\OneDrive\Desktop\SIH\Fuel_data")

# Features and target
X = df[['Feature1', 'Feature2']]  # Replace with actual feature names
y = df['Target']  # Replace with actual target name

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta

# Initialize and train the model
model = RandomForestRegressor(n_estimators=100, random_state=42)
model.fit(X_train, y_train)

# Make predictions
y_pred = model.predict(X_test)

# Evaluate the model
mse = mean_squared_error(y_test, y_pred)
print(f"Mean Squared Error: {mse}")

# Load or define your waypoints data here
# Replace with the actual waypoints dataset path
waypoints_df = pd.read_csv('waypoints_data.csv')  # Ensure this file includes 'Feat

# Predict fuel consumption or other target variable for each waypoint
waypoints_df['Predicted_Target'] = model.predict(waypoints_df[['Feature1', 'Feature
```

In [8]:
```
pip install folium
```

```
Collecting folium
  Obtaining dependency information for folium from https://files.pythonhosted.org/pa
ckages/ae/6d/18a7546e1748ecdd6ed7cd00d3f183faf1df08bd4f5e5e0eb3e72458b862/folium-0.1
7.0-py2.py3-none-any.whl.metadata
  Downloading folium-0.17.0-py2.py3-none-any.whl.metadata (3.8 kB)
Collecting branca>=0.6.0 (from folium)
  Obtaining dependency information for branca>=0.6.0 from https://files.pythonhoste
d.org/packages/75/ca/6074ab4a04dd1a503201c18091b3426f3709670115fae316907a97f98d75/br
anca-0.7.2-py3-none-any.whl.metadata
  Downloading branca-0.7.2-py3-none-any.whl.metadata (1.5 kB)
Requirement already satisfied: jinja2>=2.9 in c:\users\manicka meenakshi.s\anaconda3
\lib\site-packages (from folium) (3.1.4)
Requirement already satisfied: numpy in c:\users\manicka meenakshi.s\anaconda3\lib\s
ite-packages (from folium) (1.26.4)
Requirement already satisfied: requests in c:\users\manicka meenakshi.s\anaconda3\li
b\site-packages (from folium) (2.32.3)
Requirement already satisfied: xyzservices in c:\users\manicka meenakshi.s\anaconda3
\lib\site-packages (from folium) (2022.9.0)
Requirement already satisfied: MarkupSafe>=2.0 in c:\users\manicka meenakshi.s\anaco
nda3\lib\site-packages (from jinja2>=2.9->folium) (2.1.3)
Requirement already satisfied: charset-normalizer<4,>=2 in c:\users\manicka meenaksh
i.s\anaconda3\lib\site-packages (from requests->folium) (2.0.4)
Requirement already satisfied: idna<4,>=2.5 in c:\users\manicka meenakshi.s\anaconda
3\lib\site-packages (from requests->folium) (3.7)
Requirement already satisfied: urllib3<3,>=1.21.1 in c:\users\manicka meenakshi.s\an
aconda3\lib\site-packages (from requests->folium) (2.2.2)
Requirement already satisfied: certifi>=2017.4.17 in c:\users\manicka meenakshi.s\an
aconda3\lib\site-packages (from requests->folium) (2024.7.4)
Downloading folium-0.17.0-py2.py3-none-any.whl (108 kB)
   ---------------------------------------- 0.0/108.4 kB ? eta -:--:--
   --- ------------------------------------ 10.2/108.4 kB ? eta -:--:--
   ----------- ---------------------------- 30.7/108.4 kB 435.7 kB/s eta 0:00:01
   -------------- ------------------------- 41.0/108.4 kB 326.8 kB/s eta 0:00:01
   ---------------------------------------- 108.4/108.4 kB 696.3 kB/s eta 0:00:00
Downloading branca-0.7.2-py3-none-any.whl (25 kB)
Installing collected packages: branca, folium
Successfully installed branca-0.7.2 folium-0.17.0
Note: you may need to restart the kernel to use updated packages.
```

```
WARNING: Skipping C:\Users\MANICKA MEENAKSHI.S\anaconda3\Lib\site-packages\PyQt5-5.1
5.10.dist-info due to invalid metadata entry 'name'
WARNING: Skipping C:\Users\MANICKA MEENAKSHI.S\anaconda3\Lib\site-packages\PyQt5-5.1
5.10.dist-info due to invalid metadata entry 'name'
WARNING: Skipping C:\Users\MANICKA MEENAKSHI.S\anaconda3\Lib\site-packages\PyQt5-5.1
5.10.dist-info due to invalid metadata entry 'name'
WARNING: Skipping C:\Users\MANICKA MEENAKSHI.S\anaconda3\Lib\site-packages\PyQt5-5.1
5.10.dist-info due to invalid metadata entry 'name'
```

In [9]:

```
import pandas as pd
import numpy as np
import networkx as nx

# Load waypoints data
waypoints_df = pd.read_csv(r"C:\Users\MANICKA MEENAKSHI.S\OneDrive\Desktop\SIH\wayp

# Create a graph for Dijkstra's Algorithm
G = nx.Graph()
```

```
# Add nodes with positions
for idx, row in waypoints_df.iterrows():
    G.add_node(row['Waypoint_ID'], pos=(row['Latitude'], row['Longitude']))

# Add edges with distances
for i, row1 in waypoints_df.iterrows():
    for j, row2 in waypoints_df.iterrows():
        if i != j:
            lat1, lon1 = row1['Latitude'], row1['Longitude']
            lat2, lon2 = row2['Latitude'], row2['Longitude']
            distance = np.sqrt((lat1 - lat2)**2 + (lon1 - lon2)**2)
            G.add_edge(row1['Waypoint_ID'], row2['Waypoint_ID'], weight=distance)
```

In [11]:

```
import pandas as pd
import numpy as np
import networkx as nx
import folium
import random

# Load waypoints data
waypoints_df = pd.read_csv(r"C:\Users\MANICKA MEENAKSHI.S\OneDrive\Desktop\SIH\wayp

# Convert Waypoint_ID to numeric indices
waypoints_df['Waypoint_Index'] = np.arange(len(waypoints_df))
waypoint_index_map = dict(zip(waypoints_df['Waypoint_ID'], waypoints_df['Waypoint_I

# Create a graph for Dijkstra's Algorithm
G = nx.Graph()

# Add nodes with positions
for idx, row in waypoints_df.iterrows():
    G.add_node(row['Waypoint_ID'], pos=(row['Latitude'], row['Longitude']))

# Add edges with distances
for i, row1 in waypoints_df.iterrows():
    for j, row2 in waypoints_df.iterrows():
        if i != j:
            lat1, lon1 = row1['Latitude'], row1['Longitude']
            lat2, lon2 = row2['Latitude'], row2['Longitude']
            distance = np.sqrt((lat1 - lat2)**2 + (lon1 - lon2)**2)
            G.add_edge(row1['Waypoint_ID'], row2['Waypoint_ID'], weight=distance)

# Parameters for Q-learning
n_states = len(waypoints_df)  # Number of waypoints
n_actions = n_states  # Actions could be moving to any other waypoint
q_table = np.zeros((n_states, n_actions))
learning_rate = 0.1
discount_factor = 0.9
epsilon = 0.1  # Exploration rate

def get_reward(state, action):
    waypoint_id = waypoints_df.iloc[action]['Waypoint_ID']
    waypoint_index = waypoint_index_map.get(waypoint_id, -1)
    if waypoint_index == -1:
        return -float('inf')  # Penalize invalid waypoint IDs
```

```
    return -waypoint_index  # Example reward logic, adjust as needed

# Train the Q-learning model
for episode in range(1000):  # Number of episodes
    state = random.randint(0, n_states - 1)
    for _ in range(100):  # Steps per episode
        if random.uniform(0, 1) < epsilon:  # Exploration
            action = random.randint(0, n_actions - 1)
        else:  # Exploitation
            action = np.argmax(q_table[state])

        reward = get_reward(state, action)
        next_state = action
        best_next_action = np.argmax(q_table[next_state])

        # Update Q-table
        q_table[state, action] = (1 - learning_rate) * q_table[state, action] + \
                                 learning_rate * (reward + discount_factor * q_table
        state = next_state

# Implement Dijkstra's algorithm
start_waypoint = waypoints_df['Waypoint_ID'].iloc[0]  # Start from the first waypoi
end_waypoint = waypoints_df['Waypoint_ID'].iloc[-1]  # End at the last waypoint

# Initial route using Dijkstra's algorithm
initial_path = nx.dijkstra_path(G, source=start_waypoint, target=end_waypoint, weig

# Optimize route based on Q-table
optimized_route = [start_waypoint]  # Start from the first waypoint
current_state = waypoint_index_map[start_waypoint]

for _ in range(n_states - 1):
    next_action = np.argmax(q_table[current_state])
    next_waypoint = waypoints_df.iloc[next_action]['Waypoint_ID']
    optimized_route.append(next_waypoint)
    current_state = waypoint_index_map.get(next_waypoint, -1)
    if current_state == -1:
        break

# Create a map centered around the mean latitude and longitude
map_center = [waypoints_df['Latitude'].mean(), waypoints_df['Longitude'].mean()]
mymap = folium.Map(location=map_center, zoom_start=10)

# Add waypoints to the map
for idx, row in waypoints_df.iterrows():
    folium.Marker(
        location=[row['Latitude'], row['Longitude']],
        popup=row['Waypoint_ID'],
        icon=folium.Icon(color='blue', icon='info-sign')
    ).add_to(mymap)

# Add optimized route as a polyline
route_coords = [(waypoints_df.loc[waypoints_df['Waypoint_ID'] == wp, 'Latitude'].va
                 waypoints_df.loc[waypoints_df['Waypoint_ID'] == wp, 'Longitude'].v
                for wp in optimized_route]
folium.PolyLine(locations=route_coords, color='red', weight=2.5, opacity=1).add_to(
```

```
# Save the map to an HTML file
mymap.save(r"C:\Users\MANICKA MEENAKSHI.S\OneDrive\Desktop\SIH\combined_route_map.h

print("Combined route map has been saved to 'combined_route_map.html'.")
```

```
Combined route map has been saved to 'combined_route_map.html'.
```

In [12]:
```
import pandas as pd
import numpy as np
import folium
import time  # Import this for the sleep function

# Define the waypoints directly
waypoints_data = {
    'Waypoint_ID': [f"WP_{i+1}" for i in range(11)],
    'Latitude': [9.2886, 9.3000, 9.2500, 9.0000, 8.7500, 8.5000, 8.2000, 7.8000, 7.
    'Longitude': [79.3128, 79.2000, 79.0500, 78.8000, 78.6000, 78.4000, 78.7000, 79
}
waypoints_df = pd.DataFrame(waypoints_data)

# Example fuel efficiency calculation (customize this according to your model)
def calculate_fuel_efficiency(lat, lon):
    return np.random.uniform(1, 10)  # Random values between 1 and 10

# Apply the function to calculate fuel efficiency
waypoints_df['Fuel_Efficiency'] = waypoints_df.apply(
    lambda row: calculate_fuel_efficiency(row['Latitude'], row['Longitude']),
    axis=1
)

# Parameters for Q-learning
n_states = len(waypoints_df)  # Number of waypoints
n_actions = n_states  # Actions could be moving to any other waypoint
q_table = np.zeros((n_states, n_actions))
learning_rate = 0.1
discount_factor = 0.9
epsilon = 0.1  # Exploration rate

def get_reward(state, action):
    fuel_efficiency = waypoints_df.iloc[action]['Fuel_Efficiency']
    return -fuel_efficiency  # Reward based on fuel efficiency (lower is better)

# Training Q-learning model
for episode in range(1000):  # Number of episodes
    state = np.random.randint(0, n_states)  # Random initial state
    for _ in range(100):  # Steps per episode
        if np.random.uniform(0, 1) < epsilon:  # Exploration
            action = np.random.randint(0, n_actions)
        else:  # Exploitation
            action = np.argmax(q_table[state])

        reward = get_reward(state, action)
        next_state = action
        best_next_action = np.argmax(q_table[next_state])
```

```
        # Update Q-table
        q_table[state, action] = (1 - learning_rate) * q_table[state, action] + \
                                 learning_rate * (reward + discount_factor * q_table
        state = next_state

# Get the optimized route based on Q-learning
current_state = 0
optimized_route = [waypoints_df.iloc[current_state]]
for _ in range(n_states - 1):
    next_state = np.argmax(q_table[current_state])
    optimized_route.append(waypoints_df.iloc[next_state])
    current_state = next_state

# Create a map with Folium
map_center = [waypoints_df['Latitude'].mean(), waypoints_df['Longitude'].mean()]
mymap = folium.Map(location=map_center, zoom_start=10)

# Add waypoints to the map with fuel efficiency as a tooltip
for idx, row in waypoints_df.iterrows():
    folium.Marker(
        location=[row['Latitude'], row['Longitude']],
        popup=f"ID: {row['Waypoint_ID']}<br>Fuel Efficiency: {row['Fuel_Efficiency'
        icon=folium.Icon(color='blue', icon='info-sign')
    ).add_to(mymap)

# Add optimized route as a polyline
route_coords = [(row['Latitude'], row['Longitude']) for row in optimized_route]
folium.PolyLine(locations=route_coords, color='red', weight=2.5, opacity=1).add_to(

# Save the initial map
initial_map_path = r"C:\Users\MANICKA MEENAKSHI.S\OneDrive\Desktop\SIH\initial_rout
mymap.save(initial_map_path)
print(f"Initial optimized route map has been saved to '{initial_map_path}'.")

# Simulate ship movement and update the map dynamically
current_route = [waypoints_df.iloc[0]]
for i in range(1, len(waypoints_df)):
    # Update waypoint dynamically
    current_route.append(waypoints_df.iloc[i])

    # Update the map with the new route
    updated_map = folium.Map(location=map_center, zoom_start=10)
    for idx, row in waypoints_df.iterrows():
        folium.Marker(
            location=[row['Latitude'], row['Longitude']],
            popup=f"ID: {row['Waypoint_ID']}<br>Fuel Efficiency: {row['Fuel_Efficie
            icon=folium.Icon(color='blue', icon='info-sign')
        ).add_to(updated_map)

    # Add updated route as a polyline
    route_coords = [(row['Latitude'], row['Longitude']) for row in current_route]
    folium.PolyLine(locations=route_coords, color='green', weight=2.5, opacity=1).a

    # Save updated map
    updated_map_path = f"C:\\Users\\MANICKA MEENAKSHI.S\\OneDrive\\Desktop\\SIH\\up
    updated_map.save(updated_map_path)
```

```
    print(f"Updated route map for step {i} has been saved to '{updated_map_path}'."

    # Simulate a delay to mimic real-time updates
    time.sleep(5)  # Delay in seconds
```

```
Initial optimized route map has been saved to 'C:\Users\MANICKA MEENAKSHI.S\OneDrive
\Desktop\SIH\initial_route_map.html'.
Updated route map for step 1 has been saved to 'C:\Users\MANICKA MEENAKSHI.S\OneDriv
e\Desktop\SIH\updated_route_map_1.html'.
Updated route map for step 2 has been saved to 'C:\Users\MANICKA MEENAKSHI.S\OneDriv
e\Desktop\SIH\updated_route_map_2.html'.
Updated route map for step 3 has been saved to 'C:\Users\MANICKA MEENAKSHI.S\OneDriv
e\Desktop\SIH\updated_route_map_3.html'.
Updated route map for step 4 has been saved to 'C:\Users\MANICKA MEENAKSHI.S\OneDriv
e\Desktop\SIH\updated_route_map_4.html'.
Updated route map for step 5 has been saved to 'C:\Users\MANICKA MEENAKSHI.S\OneDriv
e\Desktop\SIH\updated_route_map_5.html'.
Updated route map for step 6 has been saved to 'C:\Users\MANICKA MEENAKSHI.S\OneDriv
e\Desktop\SIH\updated_route_map_6.html'.
Updated route map for step 7 has been saved to 'C:\Users\MANICKA MEENAKSHI.S\OneDriv
e\Desktop\SIH\updated_route_map_7.html'.
Updated route map for step 8 has been saved to 'C:\Users\MANICKA MEENAKSHI.S\OneDriv
e\Desktop\SIH\updated_route_map_8.html'.
Updated route map for step 9 has been saved to 'C:\Users\MANICKA MEENAKSHI.S\OneDriv
e\Desktop\SIH\updated_route_map_9.html'.
Updated route map for step 10 has been saved to 'C:\Users\MANICKA MEENAKSHI.S\OneDri
ve\Desktop\SIH\updated_route_map_10.html'.
```