In [3]:
```
import pandas as pd
import numpy as np
from sklearn.preprocessing import MinMaxScaler

# Load the data
data = pd.read_csv("data.csv")
```

In [4]:
```
data.head()
```

Out[4]:

	Unnamed: 0	latitude	longitude	Wind_Speed	Land_Sea_Mask	distance_to_nearest_seaport	distance_to_nearest_transmittor	sosaline	sea_height	wave
0	0	36.0	26.00	7.338949	0.000000	160.455323	360.294858	39.292670	-0.568770	
1	1	36.0	26.25	7.329445	0.000839	145.510995	338.489107	39.287334	-0.561336	
2	2	36.0	26.50	7.301338	0.000397	132.662391	316.763420	39.276382	-0.555107	
3	3	36.0	26.75	7.314985	0.003494	122.570451	295.135640	39.269196	-0.553612	
4	4	36.0	27.00	7.058693	0.005402	115.957248	273.629146	39.268070	-0.554766	

In [5]:
```
data.drop("Unnamed: 0",axis=1,inplace=True)
```

In [6]:
```
# Extract the coordinates from the data
data_coordinates = data[["latitude", "longitude"]]

# Extract the features to be used in AHP from the data
data_features = data.drop(["latitude", "longitude"], axis=1)

# Normalize the features using MinMaxScaler
scaler = MinMaxScaler()
data_norm = pd.DataFrame(scaler.fit_transform(data_features), columns=data_features.columns)

# Define the criteria weights
criteria_weights = np.array([0.317, 0.317, 0.167, 0.093, 0.048, 0.034, 0.024])

# Define the pairwise comparison matrix
pairwise_matrix = np.array([[1, 1, 3, 5, 7, 8, 9],
                            [1, 1, 3, 5, 7, 8, 9],
```

```
                            [1/3, 1/3, 1, 3, 5, 6, 7],
                            [1/5, 1/5, 1/3, 1, 3, 4, 5],
                            [1/7, 1/7, 1/5, 1/3, 1, 2, 3],
                            [1/8, 1/8, 1/6, 1/4, 1/2, 1, 2],
                            [1/9, 1/9, 1/7, 1/5, 1/3, 1/2, 1]])

# Normalize the pairwise comparison matrix to get the criteria weights
criteria_matrix = pairwise_matrix / pairwise_matrix.sum(axis=1, keepdims=True)

# Calculate the criteria weights by taking the row-wise means of the normalized pairwise comparison matrix
criteria_weights = criteria_matrix.mean(axis=1)

# Calculate the weighted sum for each location
weighted_sums = np.sum(criteria_weights * data_norm, axis=1)

# Normalize the weighted sums
weighted_sums_norm = weighted_sums / weighted_sums.max()

# Find the index of the location with the highest weighted sum
best_location_idx = np.argmax(weighted_sums_norm)

# Extract the latitude and longitude of the best location
best_location = data_coordinates.iloc[best_location_idx]

print("The best location is: Latitude {}, Longitude {}".format(best_location["latitude"], best_location["longitude"]))
```

```
The best location is: Latitude 31.25, Longitude 26.0
```

In [7]:
```
best_location
```

Out[7]:
```
latitude     31.25
longitude    26.00
Name: 703, dtype: float64
```

In [21]:
```
weighted_sums_norm
```

Out[21]:
```
0      0.704809
1      0.696327
2      0.694160
3      0.689608
4      0.672620
         ...
772    0.753143
773    0.749529
774    0.766117
775    0.766173
776    0.755220
Length: 777, dtype: float64
```

In [11]:
```
top_10_locations = weighted_sums_norm.nlargest(10)
```

In [15]:
```
top_10_locations.index
```

Out[15]:
```
Int64Index([703, 740, 704, 741, 705, 666, 742, 667, 629, 706], dtype='int64')
```

In [16]:
```
best_locations = data_coordinates.iloc[top_10_locations.index]
```

In [17]:
```
best_locations
```

Out[17]:

	latitude	longitude
703	31.25	26.00
740	31.00	26.00
704	31.25	26.25
741	31.00	26.25
705	31.25	26.50
666	31.50	26.00
742	31.00	26.50
667	31.50	26.25
629	31.75	26.00
706	31.25	26.75

In [23]:
```
data_coordinates["sums"] = weighted_sums_norm
```

```
C:\Users\MustafAi\AppData\Local\Temp\ipykernel_6536\3577858177.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  data_coordinates["sums"] = weighted_sums_norm
```

In [24]:
```
data_coordinates
```

Out[24]:

	latitude	longitude	sums
0	36.0	26.00	0.704809
1	36.0	26.25	0.696327
2	36.0	26.50	0.694160
3	36.0	26.75	0.689608
4	36.0	27.00	0.672620
...	...	...	...
772	31.0	34.00	0.753143
773	31.0	34.25	0.749529
774	31.0	34.50	0.766117
775	31.0	34.75	0.766173
776	31.0	35.00	0.755220

777 rows × 3 columns

In [27]:
```
top_20 = data_coordinates.loc[data_coordinates['latitude'] > 31.5].nlargest(20, 'sums')
```

In [28]:
```
top_20
```

Out[28]:

	latitude	longitude	sums
629	31.75	26.00	0.913122
630	31.75	26.25	0.876192
371	33.50	26.25	0.861482
334	33.75	26.25	0.860229
370	33.50	26.00	0.858979
333	33.75	26.00	0.858710
297	34.00	26.25	0.855687
335	33.75	26.50	0.852528
298	34.00	26.50	0.850939
296	34.00	26.00	0.849079
372	33.50	26.50	0.845979
407	33.25	26.00	0.840907
408	33.25	26.25	0.837369
336	33.75	26.75	0.836581
299	34.00	26.75	0.835560
337	33.75	27.00	0.832066
444	33.00	26.00	0.830623
409	33.25	26.50	0.830108
631	31.75	26.50	0.829958
373	33.50	26.75	0.828795

In [29]:

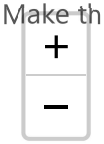
```
import folium

# create a map centered on the Mediterranean Sea
map_center = [35, 25]
zoom_level = 5
m = folium.Map(location=map_center, zoom_start=zoom_level)
```

```
# add a marker for each location
for index, row in top_20.iterrows():
    if row['latitude'] > 32:
        folium.Marker(
            location=[row['latitude'], row['longitude']],
            popup=f"Lat: {row['latitude']}, Lon: {row['longitude']}, Score: {row['sums']:.3f}",
            tooltip=row.name,
            icon=folium.Icon(color='green')
        ).add_to(m)

# display the map
m
```

Out[29]: Make this Notebook Trusted to load map: File -> Trust Notebook



In [ ]: