Test for Pole Star KTP associate

Introduction:

In this report, we have analyzed two datasets provided by Pole Star KTP Associate. The first dataset contains port visit information for several ships around the world, and the second dataset contains the positions of one ship at different timestamps. We have performed exploratory data analysis, data visualization, and predictive modeling to answer the questions provided in the instructions.

```
df = pd.read_csv('port_calls_prod_60_8.csv', encoding= ' cp1252')
df2 = pd.read_csv('Ship9794850.csv')
df.head()
```

	ship_name	imo_number	mmsi	ship_status	ship_type	flag_name	entered	departed	latitude	longitude	...	port_name	port_co
0	IL VAGABONDO	1001984	319574000.0	In Service/Commission	Yacht	Cayman Islands	2020-12-03T18:41:43Z	NaN	43.563351	7.078405	...	Juan-les-Pins	
1	IL VAGABONDO	1001984	319574000.0	In Service/Commission	Yacht	Cayman Islands	2020-12-03T14:40:15Z	2020-12-03T18:41:43Z	43.532131	7.123899	...	Antibes	
2	IL VAGABONDO	1001984	319574000.0	In Service/Commission	Yacht	Cayman Islands	2019-09-30T08:38:49Z	2020-12-03T14:40:15Z	43.563324	7.078390	...	Juan-les-Pins	
3	IL VAGABONDO	1001984	319574000.0	In Service/Commission	Yacht	Cayman Islands	2019-09-29T06:59:40Z	2019-09-30T08:38:49Z	43.528362	7.118890	...	Antibes	
4	IL VAGABONDO	1001984	319574000.0	In Service/Commission	Yacht	Cayman Islands	2019-09-28T13:42:29Z	2019-09-29T06:59:40Z	43.730396	7.456732	...	Monaco	

5 rows × 21 columns

	timestamp	mmsi	latitude	longitude	speed	course	heading	draught	destination	status	collection_type
0	2022-10-10 01:34:21.8 UTC	636018764	19.980405	119.877287	13.5	223	219	8.5	JPKSM > AEFJR	15	dynamic
1	2022-10-10 02:09:01 UTC	636018764	19.885000	119.785000	13.0	222	219	8.5	JPKSM > AEFJR	0	satellite
2	2022-10-10 02:12:01 UTC	636018764	19.876667	119.778333	13.0	221	219	8.5	JPKSM > AEFJR	0	satellite
3	2022-10-10 02:41:36.55 UTC	636018764	19.798242	119.700982	13.1	223	225	8.5	JPKSM > AEFJR	15	dynamic
4	2022-10-10 02:52:16.27 UTC	636018764	19.770292	119.673700	12.6	224	225	8.5	JPKSM > AEFJR	15	dynamic

Data Exploration:

We used the pandas library in Python to read the CSV files and performed the following data exploration tasks:

We calculated the mean hours in port for all the data.

```
df.hours_in_port.mean()
```

We have got the mean of hours in port **28.85 hours**; however, we have noticed above that we have some values in negative as well so we have two options either we will try to fix it or we will exclude it. Let's try both the methods.

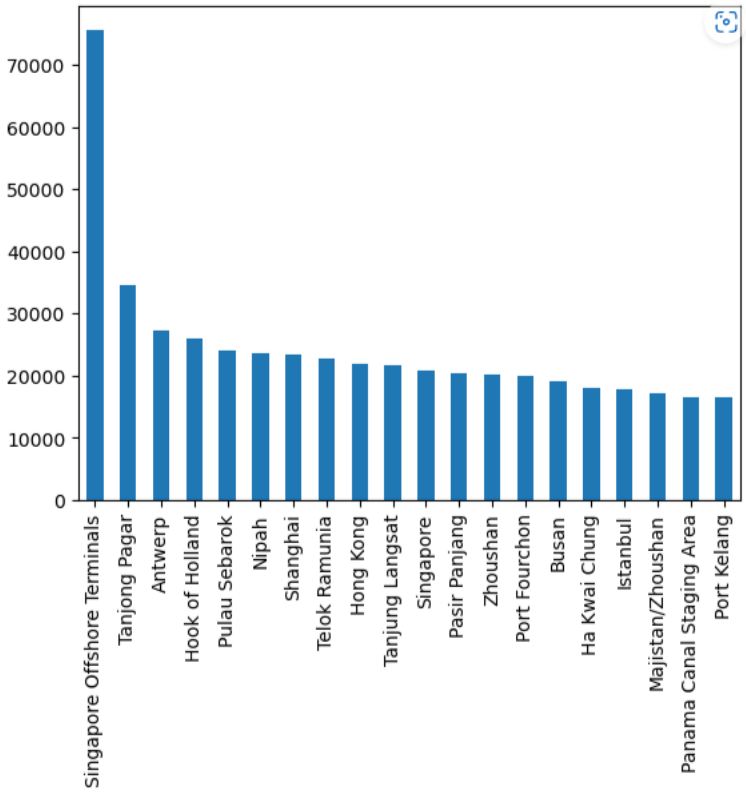
```
df_hours_exclude_negative = df[df['hours_in_port'] >= 0]
df_hours_exclude_negative.describe()
df_hours_exclude_negative.hours_in_port.mean()
(len(df[df['hours_in_port'] <= 0]) / len(df))* 100
```

New Mean of hours is **29.57** hours, we have calculated the percentage of data we have to delete to remove the negative values which was **2%.** Since, we do not see any major data deletion in the process so we are considering the **mean hours to be 29.57.**

We identified the most visited port in the dataset.

```
df.port_name.value_counts()[:10]
df.port_name.value_counts()[:20].plot(kind='bar', x='port_name')
```

```
Singapore Offshore Terminals    75574
Tanjong Pagar                   34613
Antwerp                         27369
Hook of Holland                 25910
Pulau Sebarok                   24058
Nipah                           23706
Shanghai                        23514
Telok Ramunia                   22846
Hong Kong                       21920
Tanjung Langsat                 21752
Name: port_name, dtype: int64
```



According to the graph and the data we concluded **Singapore Offshore Terminals** is the **most visited port** followed by **Tanjong Pagar** and **Antwerp.**

We calculated the mean, max, and min hours in port for all ships that entered US ports.

```
us_ports = df[df['port_country_name'] == 'United States of America'].groupby('port_country_name')['hours_in_port'].agg(['mean', 'max', 'min

# Print the results
us_ports
```

port_country_name	mean	max	min
United States of America	36.211294	15582.2	-7762.8

We discussed how we could remove outliers from the dataset.

Z-Score

```
def remove_outliers_zscore(df, col_name, threshold=3):
    mean = df[col_name].mean()
    std = df[col_name].std()
    df['z_score'] = (df[col_name] - mean) / std
    return df[df['z_score'].abs() <= threshold].drop('z_score', axis=1)
df_z_score = remove_outliers_zscore(df, 'hours_in_port')

# Calculate the number of value deleted and what percentage is it?
print(100 - ((len(df_z_score)/len(df))*100),'% of values removed from the data')
print(len(df) - len(df_z_score), 'values were removed')
```

0.7% of the outliers were removed with the help of **Z-Score** which are **22888.**

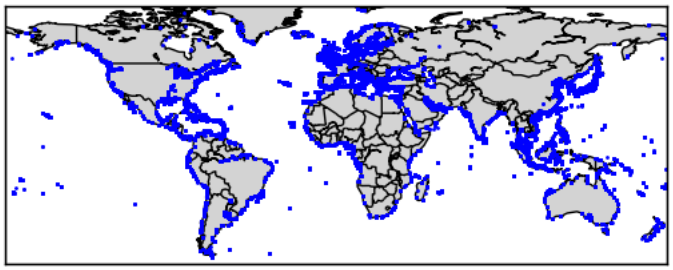
Interquartile Range (IQR) Method:

```
def remove_outliers_iqr(df, col_name):
    Q1 = df[col_name].quantile(0.25)
    Q3 = df[col_name].quantile(0.75)
    IQR = Q3 - Q1
    lower = Q1 - 1.5 * IQR
    upper = Q3 + 1.5 * IQR
    return df[(df[col_name] >= lower) & (df[col_name] <= upper)]
df_iqr = remove_outliers_iqr(df, 'hours_in_port')
print(100 - ((len(df_iqr)/len(df))*100),'% of values removed from the data')
print(len(df) - len(df_iqr), 'values were removed')
```

We visualized some port entries on a world map using the Basemap library.

```
import matplotlib.pyplot as plt
from mpl_toolkits.basemap import Basemap

# Define the map boundaries
lllon = -180 # lower left longitude
lllat = -60  # lower left latitude
urlon = 180 # upper right longitude
urlat = 80  # upper right latitude

# Create a new map
m = Basemap(projection='cyl', resolution='c',
            llcrnrlon=lllon, llcrnrlat=lllat,
            urcrnrlon=urlon, urcrnrlat=urlat)
```

```
# Add some features to the map
m.drawcoastlines(linewidth=1)
m.drawcountries(linewidth=1)
m.fillcontinents(color='lightgray', lake_color='white')
m.drawmapboundary(fill_color='white')

# Plot the port locations
lats = df.latitude.to_numpy()
lons = df.longitude.to_numpy()
x, y = m(lons, lats)
m.plot(x, y, 'bo', markersize=1)

# Show the map
plt.show()
```



We grouped the port calls by ship type.

```
# Group port calls by ship_type and count the number of entries
port_calls_by_ship_type = df.groupby('ship_type')['imo_number'].count()

# Print the result
pd.DataFrame(port_calls_by_ship_type)
```

ship_type	imo_number
Aggregates Carrier	1068
Air Cushion Vehicle Passenger	853
Anchor Handling Vessel	1378
Asphalt/Bitumen Tanker	6280
Bulk Carrier	369664
...	...
Vehicles Carrier	48533
Water Injection Dredger	219
Well Stimulation Vessel	960
Wood Chips Carrier	974
Yacht	3988

91 rows × 1 columns

Prediction:

Cleaned the data, dropped unnecessary columns:

```
train_df =  df_iqr.drop(['ship_name', 'mmsi', 'entered', 'departed', 'speed',
       'port_name', 'port_country_name', 'status',
       'hours_in_port'], axis=1)
test_df = df_iqr['hours_in_port']
```

TEST/TRAIN Split

We need to clean the data before splitting into training and testing set.

First, we will remove the feature which are not necessary for the classification or being repeated

```
drop_df =  df_iqr.drop(['ship_name', 'speed','mmsi',
       'port_name', 'port_country_name', 'status'], axis=1)
```

ship_name: We have imo number which tell us about the ship type already

speed: It contains more than 50% of null values

mmsi: We have imo_number which gives us the same imformation

port_name: It gives us the same information as port_code

port_country_name: It gives us the same information as port_code

status: It has lot's of null values.

	imo_number	ship_status	ship_type	flag_name	entered	departed	latitude	longitude	port_code	hours_in_port	gross_tonnage	length_overall_loa	draught	breadth
1	1001984	In Service/Commission	Yacht	Cayman Islands	2020-12-03T14:40:15Z	2020-12-03T18:41:43Z	43.532131	7.123899	FRANT	4.0	1143	61.27	3.29	10.98
3	1001984	In Service/Commission	Yacht	Cayman Islands	2019-09-29T06:59:40Z	2019-09-30T08:38:49Z	43.528362	7.118890	FRANT	25.7	1143	61.27	3.29	10.98
4	1001984	In Service/Commission	Yacht	Cayman Islands	2019-09-28T13:42:29Z	2019-09-29T06:59:40Z	43.730396	7.456732	MCMON	17.3	1143	61.27	3.29	10.98
5	1001984	In Service/Commission	Yacht	Cayman Islands	2019-09-28T12:57:32Z	2019-09-28T17:59:40Z	43.736800	7.431200	MCMCM	5.0	1143	61.27	3.29	10.98
6	1001984	In Service/Commission	Yacht	Cayman Islands	2019-09-27T09:25:53Z	2019-09-28T12:57:32Z	43.731758	7.457350	MCMON	27.5	1143	61.27	3.29	10.98

We need to convert the remaining categorical values into float

We will be using **Target Encoding**. Target encoding involves replacing each categorical value with the mean target value of the corresponding label (i.e., the proportion of positive class instances) for that category. This technique helps to capture the relationship between the categorical feature and the target variable, without increasing the number of features.

For example, suppose we have a categorical feature "Color" with values "Red", "Blue", and "Green", and a binary target variable (0 or 1). We can calculate the mean target value for each category as follows:

- Mean target value for "Red" = Number of positive instances with "Red" / Total number of instances with "Red"
- Mean target value for "Blue" = Number of positive instances with "Blue" / Total number of instances with "Blue"
- Mean target value for "Green" = Number of positive instances with "Green" / Total number of instances with "Green"

Then we can replace each category with its corresponding mean target value. So, for example, if the mean target value for "Red" is 0.6, then all instances with "Red" would be replaced with 0.6.

Target encoding can be a powerful tool for categorical feature encoding, but it is important to be careful when using it, as it can lead to overfitting if not properly regularized. Regularization techniques such as smoothing or cross-validation can be used to address this issue.

```
from sklearn.feature_extraction import FeatureHasher
import pandas as pd

# Define the columns to encode
cat_cols = ['ship_status',  'ship_type', 'flag_name', 'port_code' ]

target_col = 'hours_in_port'

# Define the regularization parameter
alpha = 5

# Encode the categorical columns using target encoding
for col in cat_cols:
    target_mean = drop_df.groupby(col)[target_col].mean()
    nrows = drop_df.groupby(col)[target_col].count()
    smooth_mean = (nrows * target_mean + alpha * drop_df[target_col].mean()) / (nrows + alpha)
    drop_df[col] = drop_df[col].map(smooth_mean)

# Show the encoded dataset
drop_df
```

	imo_number	ship_status	ship_type	flag_name	entered	departed	latitude	longitude	port_code	hours_in_port	gross_tonnage	length_overall_loa	draught	breadth
1	1001984	14.716175	14.023123	14.868933	2020-12-03T14:40:15Z	2020-12-03T18:41:43Z	43.532131	7.123899	6.448383	4.0	1143	61.27	3.29	10.98
3	1001984	14.716175	14.023123	14.868933	2019-09-29T06:59:40Z	2019-09-30T08:38:49Z	43.528362	7.118890	6.448383	25.7	1143	61.27	3.29	10.98
4	1001984	14.716175	14.023123	14.868933	2019-09-28T13:42:29Z	2019-09-29T06:59:40Z	43.730396	7.456732	10.278576	17.3	1143	61.27	3.29	10.98
5	1001984	14.716175	14.023123	14.868933	2019-09-28T12:57:32Z	2019-09-28T17:59:40Z	43.736800	7.431200	11.399052	5.0	1143	61.27	3.29	10.98
6	1001984	14.716175	14.023123	14.868933	2019-09-27T09:25:53Z	2019-09-28T12:57:32Z	43.731758	7.457350	10.278576	27.5	1143	61.27	3.29	10.98

```
# Splitting labels and categorical data
y_df = drop_df.hours_in_port
X_df = drop_df.drop(['hours_in_port'], axis =1)
X_df

# Converting'%Y-%m-%dT%H:%M:%S' to UNIX which is float value



from datetime import datetime

# Convert the datetime column into a datetime object
X_df['entered'] = pd.to_datetime(X_df['entered'], format='%Y-%m-%dT%H:%M:%S', utc=True)

# Convert the datetime column into Unix timestamps
X_df['entered_unix'] = (X_df['entered'] - pd.Timestamp("1970-01-01", tz='UTC')) // pd.Timedelta('1s')

# Convert the datetime column into a datetime object
X_df['departed'] = pd.to_datetime(X_df['departed'], format='%Y-%m-%dT%H:%M:%S', utc=True)

# Convert the datetime column into Unix timestamps
X_df['departed_unix'] = (X_df['departed'] - pd.Timestamp("1970-01-01", tz='UTC')) // pd.Timedelta('1s')
X_df = drop_df.drop(['entered'], axis =1)
# Print the DataFrame
X_df
```

	imo_number	ship_status	ship_type	flag_name	latitude	longitude	port_code	hours_in_port	gross_tonnage	length_overall_loa	draught	breadth
1	1001984	14.716175	14.023123	14.868933	43.532131	7.123899	6.448383	4.0	1143	61.27	3.29	10.98
3	1001984	14.716175	14.023123	14.868933	43.528362	7.118890	6.448383	25.7	1143	61.27	3.29	10.98
4	1001984	14.716175	14.023123	14.868933	43.730396	7.456732	10.278576	17.3	1143	61.27	3.29	10.98
5	1001984	14.716175	14.023123	14.868933	43.736800	7.431200	11.399052	5.0	1143	61.27	3.29	10.98
6	1001984	14.716175	14.023123	14.868933	43.731758	7.457350	10.278576	27.5	1143	61.27	3.29	10.98
...	...	...	...	...	...	...	...	...	...	...	...	...
2941491	9953016	14.716175	13.694463	14.151596	29.939243	122.205666	9.943775	12.3	8187	138.38	7.80	19.00
2941492	9953016	14.716175	13.694463	14.151596	30.007483	122.099517	16.422528	14.7	8187	138.38	7.80	19.00
2941493	9953016	14.716175	13.694463	14.151596	29.940598	122.202324	9.943775	0.0	8187	138.38	7.80	19.00
2941494	9953016	14.716175	13.694463	14.151596	21.700000	108.600000	21.795786	35.3	8187	138.38	7.80	19.00
2941495	9953016	14.716175	13.694463	14.151596	21.700000	108.600000	21.795786	14.3	8187	138.38	7.80	19.00

2625848 rows × 12 columns

```
# Train - Test Split

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X_df.to_numpy(), y_df.to_numpy(), test_size=0.20, random_state=42)
```

```
X_train shape:  (2100678, 12)
X_test shape:  (525170, 12)
y_train shape:  (2100678,)
y_test shape:  (525170,)
```

```
# Here we have trained our model with RandomForestRegressor
```

```
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error

# Create a Random Forest Regressor object
rf_regressor = RandomForestRegressor()

# Train the model on the training data
rf_regressor.fit(X_train, y_train)

# Make predictions on the testing data
y_pred = rf_regressor.predict(X_test)

# Evaluate the model's performance using mean squared error
mse = mean_squared_error(y_test, y_pred)

print("Mean Squared Error: {:.2f}".format(mse))
```

PREDICTION II

To predict the latitude and longitude of the ship at the given time points, we can use a machine learning regression model. Here are the steps that can be followed:

Visualizing the data

We can see that the data is giving us the coordinates of the ship, and comparatively the data is small so we will be able to gather most of the information just by looking up the values

```
# Import required libraries
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor

# Load the dataset
df = pd.read_csv('Ship9794850.csv')
df.head()
```

	timestamp	mmsi	latitude	longitude	speed	course	heading	draught	destination	status	collection_type
0	2022-10-10 01:34:21.8 UTC	636018764	19.980405	119.877287	13.5	223	219	8.5	JPKSM > AEFJR	15	dynamic
1	2022-10-10 02:09:01 UTC	636018764	19.885000	119.785000	13.0	222	219	8.5	JPKSM > AEFJR	0	satellite
2	2022-10-10 02:12:01 UTC	636018764	19.876667	119.778333	13.0	221	219	8.5	JPKSM > AEFJR	0	satellite
3	2022-10-10 02:41:36.55 UTC	636018764	19.798242	119.700982	13.1	223	225	8.5	JPKSM > AEFJR	15	dynamic
4	2022-10-10 02:52:16.27 UTC	636018764	19.770292	119.673700	12.6	224	225	8.5	JPKSM > AEFJR	15	dynamic

Pre Processing

We need to convert the date style into float for that we will be converting it to UNIX which is float value

```
df['timestamp'] = pd.to_datetime(df['timestamp'])

# convert datetime to float format
df['date_float'] = df['timestamp'].apply(lambda x: x.timestamp() / 86400)

df
```

	timestamp	mmsi	latitude	longitude	speed	course	heading	draught	destination	status	collection_type	date_float
0	2022-10-10 01:34:21.800000+00:00	636018764	19.980405	119.877287	13.5	223	219	8.5	JPKSM > AEFJR	15	dynamic	19275.065530
1	2022-10-10 02:09:01+00:00	636018764	19.885000	119.785000	13.0	222	219	8.5	JPKSM > AEFJR	0	satellite	19275.089595
2	2022-10-10 02:12:01+00:00	636018764	19.876667	119.778333	13.0	221	219	8.5	JPKSM > AEFJR	0	satellite	19275.091678
3	2022-10-10 02:41:36.550000+00:00	636018764	19.798242	119.700982	13.1	223	225	8.5	JPKSM > AEFJR	15	dynamic	19275.112229
4	2022-10-10 02:52:16.270000+00:00	636018764	19.770292	119.673700	12.6	224	225	8.5	JPKSM > AEFJR	15	dynamic	19275.119633

Feature Selection and Train Test Split

We need to pick up the important features only, so we have dropped **mmsi**, **destination** and **collection_type** as they are being repeated and holds the same value.

Further we have divided the data into train and test data. We have 2 labels **latitude** and **longitude.** So it was divided accordingly.

```
X = df[['date_float', 'speed', 'course', 'heading', 'draught', 'status']]
y_lat = df['latitude']
y_lon = df['longitude']

# Split data into training and testing sets
X_train, X_test, y_lat_train, y_lat_test, y_lon_train, y_lon_test = train_test_split(X, y_lat, y_lon, test_size=0.2, random_state=42)
print('X_train shape: ', X_train.shape)
print('X_test shape: ', X_test.shape)
print('y_lat_train shape: ', y_lat_train.shape)
print('y_lat_test shape: ', y_lat_test.shape)
print('y_lon_train shape: ', y_lon_train.shape)
print('y_lon_test shape: ', y_lon_test.shape)
```

```
X_train shape:  (36, 6)
X_test shape:  (9, 6)
y_lat_train shape:  (36,)
y_lat_test shape:  (9,)
y_lon_train shape:  (36,)
y_lon_test shape:  (9,)
```

Training and Testing the model

```
# Train a random forest regression model
rf = RandomForestRegressor(n_estimators=100, random_state=42)
rf.fit(X_train, y_lat_train)

# Evaluate model performance on testing set
lat_pred = rf.predict(X_test)
print("Latitude R-squared: ", rf.score(X_test, y_lat_test))

rf.fit(X_train, y_lon_train)
lon_pred = rf.predict(X_test)
print("Longitude R-squared: ", rf.score(X_test, y_lon_test))
```

```
Latitude R-squared:  0.9976868055958996
Longitude R-squared:  0.9973407498924802
```

Prediction

For prediction, we have to convert the input variable which is **Timestamp** into same **UNIX format** for correct prediction.

Further, we have to fill in the other 5 features as well, below explained the method used for filling it.

Speed: Taking the average speed of the ship by calculating the **mean** of the speeds provided in the dataset

Course: Taking the average course of the ship by calculating the **mode** of the Course provided in the dataset

Heading: Taking the average speed of the ship by calculating the **mode** of the heading provided in the dataset

Draught: Taking the average speed of the ship by calculating the **mode** of the draught provided in the dataset

Status: Taking the average speed of the ship by calculating the **mode** of the status provided in the dataset

```
pred = pd.DataFrame({'timestamp': ['2022-10-10 19:30:05 UTC', '2022-10-10 19:33:06 UTC', '2022-10-10 20:30:08 UTC']})
pred
pred['timestamp'] = pd.to_datetime(df['timestamp'])

# convert datetime to float format
pred['timestamp'] = pred['timestamp'].apply(lambda x: x.timestamp() / 86400)

# Use the trained model to predict the latitude and longitude of the ship at the given timestamps
X_pred = pd.DataFrame({'date_float': pred['timestamp'], 'speed': 12.5, 'course': 223, 'heading': 222, 'draught': 8.5, 'status': 15})
X_pred
```

	date_float	speed	course	heading	draught	status
0	19275.065530	12.96	222	222	8.5	15
1	19275.089595	12.96	222	222	8.5	15
2	19275.091678	12.96	222	222	8.5	15

Feeding our Random Forest for the prediction

```
lat_pred = rf.predict(X_pred)
lon_pred = rf.predict(X_pred)

print("Predicted latitude at given timestamps: ", lat_pred)
print("Predicted longitude at given timestamps: ", lon_pred)
```

```
Predicted latitude at given timestamps:  [119.72733925 119.71963899 119.71917232]
Predicted longitude at given timestamps:  [119.72733925 119.71963899 119.71917232]
```