About the Project

This project description is based on our team's final presentation slide and plus my own detail description on data preprocessing. Our team presented the project results in front of 20+ professional people from Orbital Insight and Cornell Tech.

This is the link of our final presentation video:

> https://youtu.be/bGd3z2A_Sko

AI Studio Project Overview

Team Challenge Goal

> To build and test a model that will accurately predit dwell times of ships heading to Shanghai during and outside of typhoon season

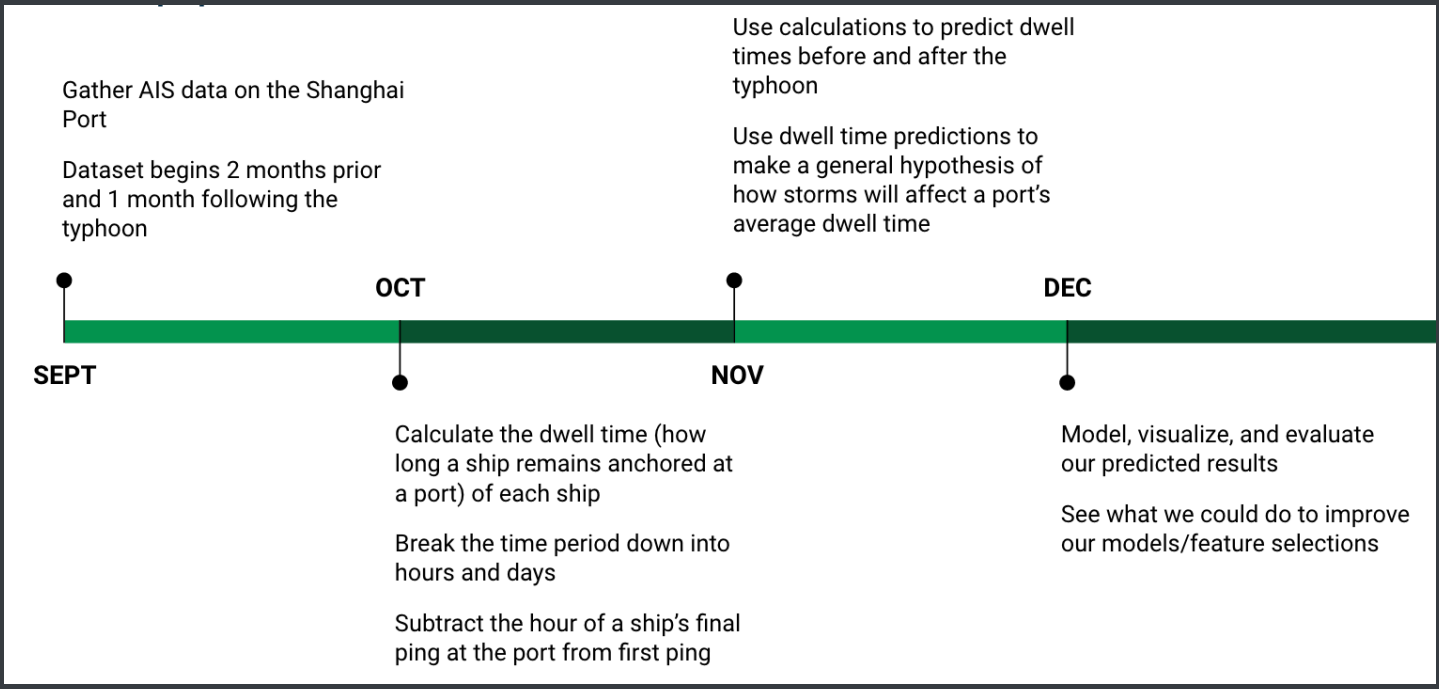
What is *dwell time*? Time spent in same position, area, stage of process

Business Impact

- Why we chose this goal?
  - Merchants and businesses need to know *when shipments arrive*
  - ***Shanghai*** is one of the busiest ports in the world
    - *Delays* here impact the global supply chain, making this important to understand
  - Ship dwell times are difficult to predict in typhoon-affected waters

- See the effect that worsening typhoons (caused by climate change) could have on shipping
- Utilize geo-spatial data rather than typical dataset, AOI (drawing polygons, QGIST)
- Important economic impact, provides insight to businesses that can only be drived from AI

Our Approach



Resources we used for this project

At first, to decide our project topic and look around vessels' activities, we used Orbital Insight's own program "***Orbital Insight Go***". Based on that, we asked Olivia to gather **AIS data** and used **QGIST** to visualize it.

To work on our project, we used **Google Colab** to collaborate with and used **Pandas** and **Scikit-Learn** for implementing data preprocessing and making a machine learning model.

Data Preprocessing and EDA

First of all, we had to deal with **Null values**, drop unnecessary values, and change String values to some dummy values.

```
df.drop("imo", axis = 1, inplace = True)
```

```
df.drop("vessel_name", axis = 1, inplace = True)
```

```
df.drop(df.loc[df['nav_status_code'] == 16].index, inplace=True)
```

```
df.drop(df.loc[df['length'] == 0].index, inplace=True)
df.drop(df.loc[df['width'] == 0].index, inplace=True)
df.drop(df.loc[df['draught'] == 0].index, inplace=True)
```

First, we dropped the "**imo**" column. **IMO** stands for "*IMO ship identification number*" and this number is used as a unique ship identifier. However, in our data, there were many 0 values and we already had the "**device_id**" column for identifying unique ships, so we decided to drop it.

We also dropped the "**vessel_name**" column for the same reason. This column was a string value and we had to either drop it or change it to a dummy value, but we already had the "**device_id**" column that could be used instead of "vessel_name".

Next, we had to deal with "**null values**". The first null values that we dealt with were "**df['nav_status_code'] == 16**". "nav_status_code" means **AIS Navigational Status** and this is a form of *signal reported by the vessel, which describes **the status of the vessel** in real-time*. The codes have their own meanings from 0 to 15, but we noticed that some of our data had 16 as its value. After discussing with our advisor, we decided that this did not have any meaning, so we dropped the rows that had 16 as its "nav_status_code" value.

The next null values we had to take care of were "**df['length'] == 0, df['width'] == 0, df['draught'] == 0**". The columns 'length', 'width', 'draught' do not have meaning separately, but if you multiply all of them (**length x width x draught**), it becomes "**tonnage**" (*the capacity of the ships in tons, in our case, the vessel's capacity under the water*). To be valid vessel data, this "tonnage" value should be larger than 0, but if one of the values is 0, "tonnage" value becomes 0, so we decided to drop these values. The reason why we just dropped them rather than substituting with other values was the amount of data that had invalid values was way smaller than the original value. So, we decided to drop it considering our time constraints.

Next, we tried to calculate the tonnage of the vessel under the water (ocean) because we thought that the vessel's tonnage under the ocean might have a connection with predicting the "dwell time" of the ships.

```
# Calculate Tonnage
df['tonnage'] = 0
df['tonnage'] = df['length'] * df['width'] * df['draught']
df = df[df['tonnage'].notna()]
```

Now, using the "**unixtime**" of our AIS data, we first tried to prove our assumption that a *typhoon would affect a vessel's journey, especially the dwell time*. If there is a *big difference* before and after the typhoon period, we can say that a *typhoon can affect a vessel's journey*. Since we know that a typhoon can have a big impact on our vessel's journey, it would also affect the vessel's dwell time. Therefore, when we make our model that can predict the "dwell time" of a ship, we can tell the model:

> Oh, if vessels are doing a journey when a typhoon happens, the dwell time would be different than normal time. The vessels' dwell time during the typhoon season are like this. Please be aware of this when you calculate the dwell time.

This was our plan, so now let's begin with data analysis.

We converted "unixtime" to "datetime" for our better understanding.

```
# Convert Unixtime
df['unix_to_date'] = 0
df['unix_to_date'] = pd.to_datetime(df['unixtime'], unit='s').apply(lambda
x: x.to_datetime64())
```

And then, among our AIS data, we grabbed "**df['nav_status_code'] == 1**". As I mentioned before, we want to check whether or not a typhoon would affect a vessel's journey, especially the dwell time. There must be a fewer number of "anchored" vessels before the typhoon and an unusual huge number of "anchored" vessels after the typhoon if a typhoon really "affects" a vessel's journey. Therefore, to only get anchored ships, **we only have to get the "anchored" vessel's data** and *"1" value for "nav_status_code" means in AIS data means that the vessel is anchored*.

```
df_anchored = df[(df['nav_status_code']==1)].copy()
```

With this anchored vessel's data, we tabulated unique vessels by hours and date to prove our assumption and it actually was true. Our data analysis shows that ***there is a huge difference in the number of vessels that are anchored at a port before and after a typhoon***.

```
# Function we used for "tabulating" unique vessels by hours and date

def hourlyData(df):
    df['hour'] = (df['unixtime'] / 3600).astype(int)
    # re-index hours to 1
    df['hour'] = df['hour'] - df['hour'].min() + 1
    return df.groupby('hour')['device_id'].agg('nunique')

  def dailyData(df):
    df['day'] = (df['unixtime'] / 86400).astype(int)
    # re-index hours to 1
    df['day'] = df['day'] - df['day'].min() + 1
    return df.groupby('day')['device_id'].agg('nunique')
```

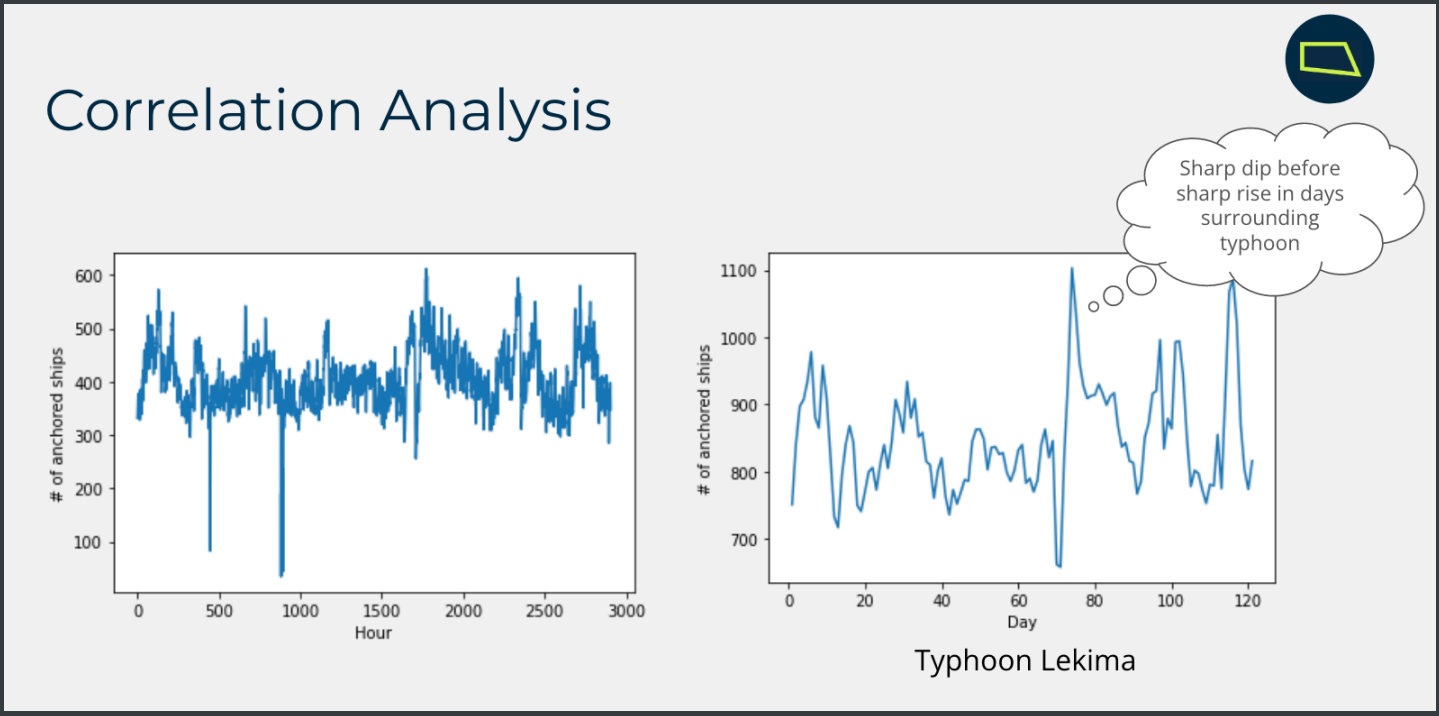
```
# tabulate unique ships by hour
hourly_anchored = hourlyData(df_anchored)
hourly_anchored
```

```
df_anchored['num_anchored_hour'] = df_anchored['hour'].apply(lambda x:
hourly_anchored[x] if (x > 0) else 0)
```

```
#tabulate unique ships by day
daily_anchored = dailyData(df_anchored)
daily_anchored
```

```
df_anchored['num_anchored_day'] = df_anchored['day'].apply(lambda x:
daily_anchored[x] if (x > 0) else 0)
```

This is correlation analysis of our data



After that, we started calculating the "dwell time" of the ships. This step was the hardest because **dwell time** is calculated by "**vessels leaving - vessels arriving**". It looks simple, but in our AIS data, ***some of the vessels had multiple trips***. Therefore, we needed to differentiate between the **end of a first trip** and the **start of another trip**.

To differentiate the multiple trips, first, we got **all the time data** ('hour' columns in our dataset) of **each unique vessel** and sorted all the time by its value in **ascending order**. After that, if there was a **big time difference** between the trip, we **considered that as another trip** and saved it in a separate list to use it to get multiple dwell times.

> How we differentiate multiple trips

```
# All anchored time for each unique ships
```

```
  current_time = sorted_anchored.loc[sorted_anchored['device_id'] == i]
[['hour']]
  array_hour = current_time.values.tolist()
  hour_len = len(array_hour)
  biggest_dif = 0
  # it contains index of list_hour(list unique ship's "hour" column) that
seems to start new trips
  trip_change = [0]
  list_hours = []

  for x in array_hour:
    list_hours.append(x[0])

  for j in list_hours:
    next_index = list_hours.index(j) + 1
    if(next_index < hour_len):
      biggest_dif = list_hours[next_index] - j
    # This means this ships had done multiple trips
    if(biggest_dif > 12):
      trip_change.append(next_index)
```

> Get the dwell time of the multiple trips

```
  dwell = []
  start_times = []

  for h in range(len(trip_change)):
    # This "if" condition for avoiding index range error
    if((len(array_hour) != trip_change[h])):
      # the case when there are only two different "trip changes"
      if(len(trip_change) == 2 and h == 1):
```

```
        dwell.append((list_hours[trip_change[h]-1] -
list_hours[trip_change[h-1]]) + 1) #trip 1
        start_times.append(list_hours[trip_change[h-1]])

        dwell.append(list_hours[-1] - list_hours[trip_change[h]] + 1) #trip
2
        start_times.append(list_hours[trip_change[h]])

      #Last index
      elif(h + 1 == len(trip_change)):
        dwell.append(list_hours[-1] - list_hours[trip_change[h]] + 1)
        start_times.append(list_hours[trip_change[h]])

      # Other index
      elif (h != 0):
        dwell.append((list_hours[trip_change[h]-1] -
list_hours[trip_change[h-1]]) + 1)
        start_times.append(list_hours[trip_change[h-1]])

  dwell_times[i] = dwell
  trip_start[i] = start_times
```

We got "dwell time" by *using two data points*. One for the "vessel arrival" time and one for the "vessel leave" time. And eventually, *both indicate the same trip*. Since our data is duplicated, if we keep both, we decided to keep the data for "vessel arrival" only and assigned the dwell time to that data.

```
new_sorted_anchored = sorted_anchored.copy()
# key == vessel's device id
dwell_times_keys = dwell_times.keys()
for i in dwell_times_keys:
  # Get all trips of one unique vessels
  target = sorted_anchored.loc[sorted_anchored['device_id'] == i]
```

```
    # Get the 'hour' data of that unique vessel
    target_hours = target['hour']
    hours_array = np.array(target_hours)
    for j in hours_array:
      # basically in our AIS data,
      if that hour is in "trip_start" column of that vessel, add "dwell time"
column,
      # s
      if (j in trip_start[i]):
        # get the row index of that target data
        row_num = (target[target['hour'] == j].index)[0]
        # get the dwell time index of that target data
        dwell_idx = trip_start[i].index(j)
        #df.loc[rowIndex, 'New Column Title'] = "some value"
        new_sorted_anchored.loc[row_num, 'dwell_time'] = dwell_times[i]
[dwell_idx]
```

```
# Only keep one data
final_sorted_anchored =
new_sorted_anchored[new_sorted_anchored['dwell_time'] > 0]
```

Now, we need another column that can tell our machine learning model that this is the dwell time data of the vessels when a typhoon happened. Let's create a "typhoon" column that can indicate whether a typhoon happened at this time period or not.

```
final_sorted_anchored['typhoon'] = 0
final_sorted_anchored['typhoon'] =
final_sorted_anchored['unixtime'].apply(lambda x: 1 if (x <= 1565459100 and
x >= 1565372700) else 0)
```

Actually, we ran our machine learning model after this, but our models seemed like they were overfitting. Therefore, with advice from our advisor, we did a little bit more feature engineering.

Here what we'd tried after

1. Drop the data that has unusual dwell time

```
df = df.loc[df['dwell_time'] <= 600
```

2. We only wanted to care about "Cargo" and "Tanker" type of vessels. and to only get "Cargo" and "Tanker" vessels type, our data's "vessel_type_code" should between 70 to 89.

```
df = df[(df['vessel_type_code'] >= 70) & (df['vessel_type_code'] <= 89)
```

3. Add a new column that indicate number of ships in backlog at time of each ship's first anchored ping

```
# sort pings by device_id and time
df = df.sort_values(by=['device_id', 'unixtime'], ascending=True)
```

```
# max time between pings to consider part of the same AOI visit
max_gap = 3600*6

# flag whether successive pings are from the same vessel
df['same_vessel'] = (df['device_id'].astype(int).diff() == 0)

# find time gap between pings
df['time_gap'] = (np.abs(df['unixtime'].astype(int).diff()) > max_gap)

# flag as new trip when time since previous ping is greater than
max_gap (or previous ping is from a different vessel)
df['new_trip'] = (np.invert(df['same_vessel']) | df['time_gap'])

# add unique trip id
df['trip_id'] = df['new_trip'].cumsum()
```

```
# get first and last ping for each trip and it also require a ~0 cog (course-over-ground) to filter out the small subset of pings that erroneously report that the ship is ancored/moored while it is still moving
# by first filtering down to anchored/moored pings, then groupby trip_id with first/last agg
df_anchored = df[df['cog']<0.1]
first_anchored = df_anchored.groupby('trip_id').agg('first')
last_anchored = df_anchored.groupby('trip_id').agg('last')
```

```
# only fields we care about from last anchored pings are trip_id and unixtime
df_trips = pd.merge(first_anchored, last_anchored[['unixtime']], suffixes=('_first', '_last'), left_index=True, right_index=True)
df_trips.sort_values('unixtime_first', inplace=True)
```

```
 # calculate number of ships in backlog at time of each ship's first anchored ping
df_trips['new_trip'] = True
df_trips['cumulative_ships_arrived'] =
df_trips.sort_values('unixtime_first')   ['new_trip'].cumsum()
df_trips['cumulative_ships_departed'] =
[len(df_trips[df_trips['unixtime_last']<=arrival]) for arrival in df_trips['unixtime_first'].values]
df_trips['n_ships_anchored'] = df_trips['cumulative_ships_arrived'] - df_trips['cumulative_ships_departed']
```

Model Selection and Evaluation

Model Selection Process

- ***Features***:
  - Vessel type code
  - Latitude & Longtitude
  - Typhoon Occurence
  - Number of ships anchored

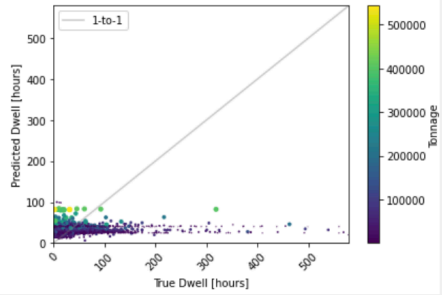
> Some people might notice that we didn't use "tonnage" as one of our features! It's because we assumed that it might cause overfitting, so we just dropped it at the end!

- ***Label***:
  - Dwell Time
- ***Problem***:
  - *Regression* - Supervised learning problem where the label is real number
- ***Our Chosen Models***:
  - Linear Regression
  - Random Forest
  - Gradient Boosting

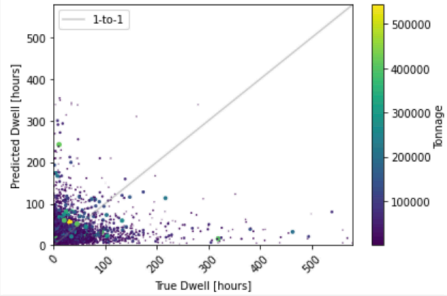
Model Comparison

Model Name	Description	Results	Pros	Cons
Linear regression	Model where straight line represents the data and proximity of points to line shows relationship between feature and label	Mean absolute error: **26.36** R^2: **0.01**	- Faster, more efficient, and simple - Overfitting can be reduced	- Prone to underfitting - Sensitive to outliers - Assumes data is independent
Random forest regressor	Regression model that involves multiple decision trees contained within one set and combined into an ensemble method	Mean absolute error: **29.55** R^2: **-0.34**	- High accuracy - Easy data preparation - Can handle larger data	- Biased towards more complex variables - Little control over the model
Gradient boosting regressor	Iterative model that progressively refines its predictions by combining multiple decision trees	Mean absolute error: **26.34** R^2: **0.00**	- Generally more accurate compare to other models - Lots of flexibility	- Prone to overfitting - Hard to interpret the final models

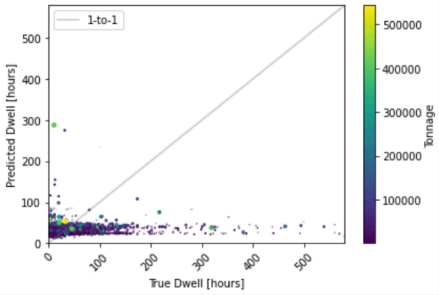
Model Visualization



Linear Regression



Random Forest Regressor



Gradient Boosting Regressor

Insights and Key Findings

- ***Final Model Selection***
  - *Gradient Boosting Regressor*
    - Why? Lowest mean absolute error (only by a negligible amount)
- ***Mean Absolute Error is very high***
  - Overfitted Model
  - Features may not lead to accurate predictions
- ***Potential Solutions***
  - Spend more tinkering with features
  - Try using different and more datasets
  - Potentially look at approaching as classification problem

Final Thought

Trial and Error

Lessons Learned:

- How to decide our reasonable thesis and scale of our project
- How to deal with AIS dataDeeper understanding of python libraries (Numpy, Pandas, Scikit-Learn) & ML tools (Google Colab)
- How important the data preparation process is
- How to format the data to be suitable for our Machine Learning model

- Why correct understanding of input and output feature is important
- How to work as a teamImportance of understanding the business/economic impact of our project

Takeaways:

- Exposure to a new side of machine learning
- Gaining hands on experience for Python in a quick-paced learning environment
- To not be afraid to ask questions!

Obstacles & Potential Next Steps

Obstacles:

- Scheduling a time when we are all available
- Limited in-person meetings
- Beginner knowledge in Python
- New exposure to machine learning and data science
- Never done any industrial scale project before
- Geospatial data vs clean data

Next Steps:

- Scale up to different region or different typhoon
- Apply this model to other natural disaster
- Apply this model to different transportation such as truck, airplane, train, and so on
- Focus on a specific company? (Amazon, Ebay, etc)