Module 3

Data Analytics Lab

Team 2: Analysis of Uber/Lyft rides

Team Members:

Akash Sundaresan
Ankit Bhawsar
Anmol Gupta
Raushan Khullar
Serah Varghese
Sonia Sun

Introduction

In the first Module, we analyzed the Uber/Lyft Dataset to identify important factors such as the frequency, price, and its correlation with the weather conditions, day of the week, Month, etc.

This analysis is vital for module 2.

Module 2 covered important machine learning methods to predict future demand prediction, price fare prediction, and Clustering. It included optimization to identify the best possible hub to open.

Our primary stakeholders are customers, the board of Directors, and other Business owners who would like to partner with Uber/Lyft for cost reduction and operational efficiency.

In Module 3, we cover conveying our analysis to a broader audience using interactive graphs and charts to those who may need to become more familiar with the technical concepts. It gives a holistic and interactive view that can be used to make better decisions by the leadership.

Part 1

Interactive Graph To View Ride V/S Hour Using Different Graph Types

Importing the dataset

In [2]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
```

In [4]:
```
import datetime
# Calling the fromtimestamp() function to
# extract datetime from the given timestamp
# from google.colab import drive
# drive.mount('/content/drive')
ride = pd.read_csv("rideshare_kaggle.csv")
# ride = pd.read_csv("/content/drive/MyDrive/Colab Notebooks/rideshare_kaggle.csv")
ride['timestamp']=pd.to_datetime(ride['timestamp'], unit='s')
ride.dropna(axis=0,inplace=True)
```

In [5]:
```
frequency=ride.groupby(["hour","source","destination"]).size().reset_index(name="Ride_Frequency")
```

In [6]:
```
frequency
```

Out[6]:

	hour	source	destination	Ride_Frequency
0	0	Back Bay	Boston University	442
1	0	Back Bay	Fenway	376
2	0	Back Bay	Haymarket Square	410
3	0	Back Bay	North End	471
4	0	Back Bay	Northeastern University	395
...	...	...	...	...
1723	23	West End	Fenway	424
1724	23	West End	Haymarket Square	365
1725	23	West End	North End	388
1726	23	West End	Northeastern University	376
1727	23	West End	South Station	462

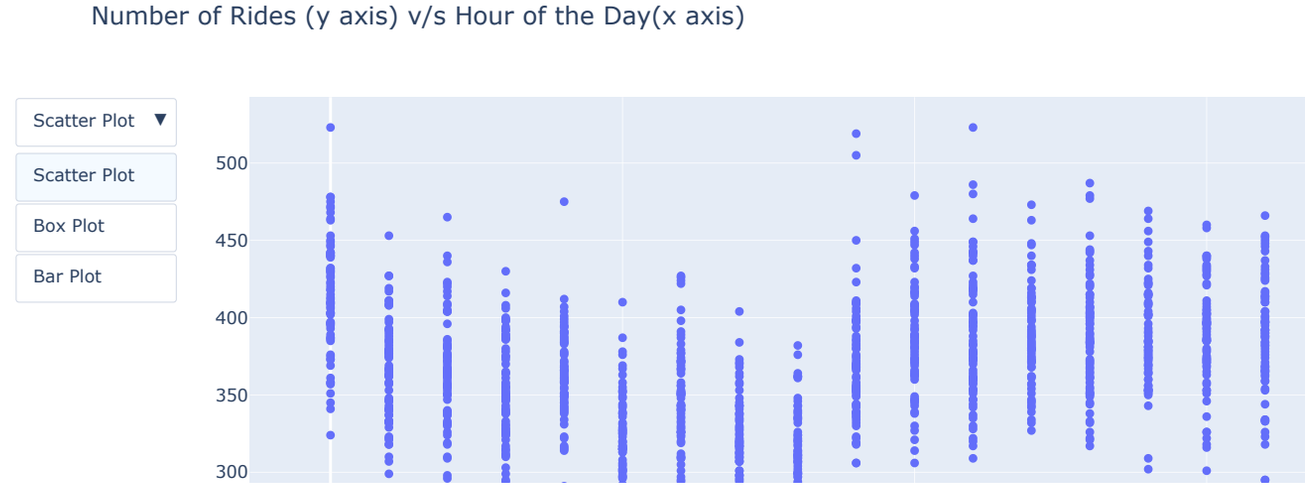
1728 rows × 4 columns

The code given below is to plot the interactive graph using plotly. The drop down menu gives us 3 options ie

1. Box plot- Useful for identifying median deman at a given time and the outliers
2. Scatter Plot- Helping identify the spread of the deamnd
3. Bar graph - Giving a quick over view of maximum demand recorded for a given hour.

In [7]:
```
import plotly.graph_objects as px
import numpy as np


np.random.seed(52)

random_x = frequency['hour']
random_y = frequency['Ride_Frequency']

fig = px.Figure(data=[px.Scatter(
        x=random_x,
        y=random_y,
        mode='markers',)
])

fig.update_layout(
        updatemenus=[
                dict(
                buttons=list([
                        dict(
                        args=["type", "scatter"],
                        label="Scatter Plot",
                        method="restyle"
                        ),
                        dict(
                        args=["type", "box"],
                        label="Box Plot",
                        method="restyle"
                        ),
                        dict(
```

```
                        args=["type", "bar"],
                        label="Bar Plot",
                        method="restyle"
                        )
                        ]),
                        direction="down",
                    ),
                ],
      title = "Number of Rides (y axis) v/s Hour of the Day(x axis)"
)

fig.show()
```



In [ ]:

In [ ]:

Representing SARIMA Forecast Using Plotly Interactive Graphs

Here we are making an attempt to come upwith SARIMA Model to forecast ride demand for managers. As discussed in the previous modules. But here we will presented the forecast results in a much more user friendly manner with interactive graph with an option to drill down to as deep as viewing daily data.The main objective of this section is to give the stakeholders a comprehensive overview of the demnad forecast data and makemit more interpretable. The aim is to answer as many question as possible, with the help over interactivity which helps the user to drill down or roll up data plotted in the graph.

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
from statsmodels.tsa.stattools import adfuller
```

Importing data from the drive

In [3]:
```
import datetime
# Calling the fromtimestamp() function to
# extract datetime from the given timestamp
# from google.colab import drive
# drive.mount('/content/drive')
ride = pd.read_csv("rideshare_kaggle.csv")
# ride = pd.read_csv("/content/drive/MyDrive/Colab Notebooks/rideshare_kaggle.csv")
ride['timestamp']=pd.to_datetime(ride['timestamp'], unit='s')
ride.dropna(axis=0,inplace=True)
ride["Date"] = ride["timestamp"].dt.date
ride["Time"] = ride["timestamp"].dt.time
ride["Weekday"]=ride["timestamp"].dt.day_name()
```

In [4]:
```
ride['Date'] = pd.to_datetime(ride['Date'], format='%Y-%m-%d')
```

In [5]:
```
filtered_df = ride.loc[(ride['Date'] == '2018-11-27')]
```

In [6]:
```
len(filtered_df)
```

Out[6]:
```
70135
```

Dropping NULL values

In [7]:
```
ride['datetime']=pd.to_datetime(ride['datetime'])
ride=ride.dropna()
```

In [8]:
```
ride['datetime']
```

Out[8]:
```
0         2018-12-16 09:30:07
1         2018-11-27 02:00:23
2         2018-11-28 01:00:22
3         2018-11-30 04:53:02
4         2018-11-29 03:49:20
                  ...
693065    2018-12-01 23:53:05
693066    2018-12-01 23:53:05
693067    2018-12-01 23:53:05
693069    2018-12-01 23:53:05
693070    2018-12-01 23:53:05
Name: datetime, Length: 637976, dtype: datetime64[ns]
```

In [9]:
```
grouped = ride.groupby(ride['datetime'].dt.floor('h'))['id'].count().reset_index()

grouped.rename(columns={'id': 'total_rides'}, inplace=True)
```

In [10]:
```
ride['datetime'].dt.floor('h')
```

Out[10]:
```
0         2018-12-16 09:00:00
1         2018-11-27 02:00:00
2         2018-11-28 01:00:00
3         2018-11-30 04:00:00
4         2018-11-29 03:00:00
                  ...
693065    2018-12-01 23:00:00
693066    2018-12-01 23:00:00
693067    2018-12-01 23:00:00
693069    2018-12-01 23:00:00
693070    2018-12-01 23:00:00
Name: datetime, Length: 637976, dtype: datetime64[ns]
```

In [11]:
```
grouped.head()
```

Out[11]:

	datetime	total_rides
0	2018-11-26 03:00:00	77
1	2018-11-26 04:00:00	390
2	2018-11-26 05:00:00	616
3	2018-11-26 06:00:00	1462
4	2018-11-26 07:00:00	925

Setting timestamp as the index for corresponding ride demand

In [12]:
```
grouped.set_index('datetime',inplace=True)
```

This is the final dataset to be used in ARIMA forecasting

In [13]:
```
grouped['total_rides']=grouped['total_rides'].astype(float)
```

An overview of the dataset we are about to use

In [14]:
```
grouped
```

Out[14]:

	total_rides
datetime	
2018-11-26 03:00:00	77.0
2018-11-26 04:00:00	390.0
2018-11-26 05:00:00	616.0
2018-11-26 06:00:00	1462.0
2018-11-26 07:00:00	925.0
...	...
2018-12-18 15:00:00	1709.0
2018-12-18 16:00:00	1712.0
2018-12-18 17:00:00	1712.0
2018-12-18 18:00:00	1745.0
2018-12-18 19:00:00	584.0

332 rows × 1 columns

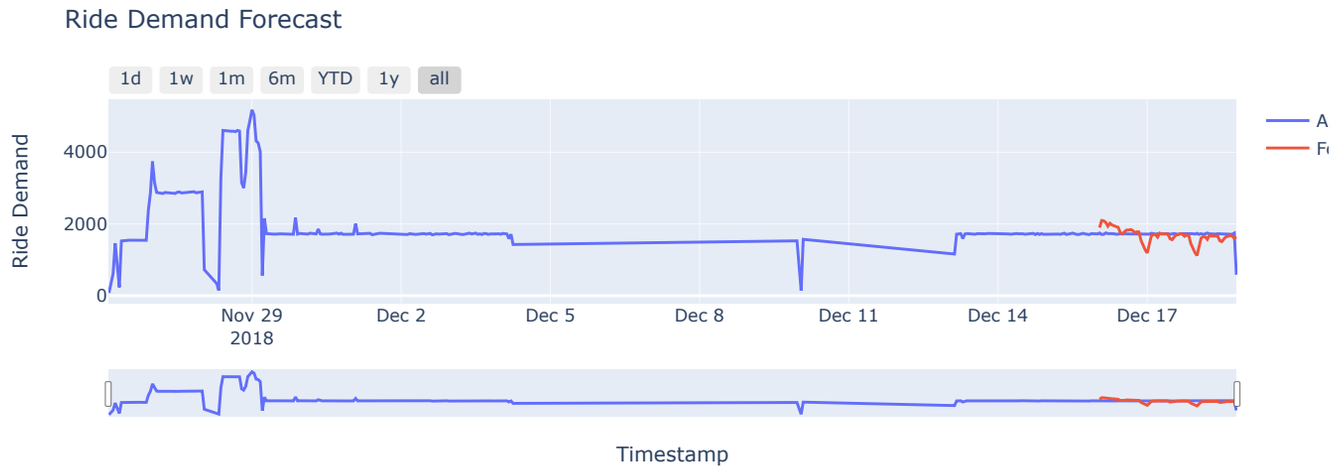
In the code block given below, the pieces of code essentially help us

1. Firstly use the dataset to run SARIMA forecasting model
2. Predict(forecast) the ride demand for the last week of December 2018.
3. Create Interactive plot using plotly library and plot forecasted and already known values (both test and train data).
4. The last part of code helps in adding interactivity to the graph by allowing user to select the aggregation level of data ie weekly, daily, yearly etc. The code helps in defining the buttons and their functions with respect to plotting data in the graph.

In [15]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from statsmodels.tsa.statespace.sarimax import SARIMAX
from sklearn.metrics import mean_squared_error
import plotly.graph_objects as go
from plotly.subplots import make_subplots




# Split into training and test sets
train_size = int(len(grouped) * 0.8)
train, test = grouped[:train_size], grouped[train_size:]

# Fit SARIMA model
model = SARIMAX(train, order=(1, 1, 1), seasonal_order=(0, 1, 1, 24))
model_fit = model.fit()

# Forecast
start_index = len(train)
end_index = len(train) + len(test) - 1
forecast = model_fit.predict(start=start_index, end=end_index)







# Evaluate model
mse = mean_squared_error(test, forecast)
rmse = np.sqrt(mse)
print('Test RMSE: %.3f' % rmse)

# Create interactive plot
fig = make_subplots(rows=1, cols=1, shared_xaxes=True)

fig.add_trace(go.Scatter(x=grouped.index, y=grouped['total_rides'], name='Actual'), row=1, col=1)
fig.add_trace(go.Scatter(x=test.index, y=forecast, name='Forecast'), row=1, col=1)

fig.update_layout(title='Ride Demand Forecast',
                  xaxis_title='Timestamp',
                  yaxis_title='Ride Demand',
                  height=400)

fig.update_layout(xaxis=dict(rangeselector=dict(buttons=list([
                                dict(count=1, label="1d", step="day", stepmode="backward"),
                                dict(count=7, label="1w", step="day", stepmode="backward"),
                                dict(count=1, label="1m", step="month", stepmode="backward"),
                                dict(count=6, label="6m", step="month", stepmode="backward"),
                                dict(count=1, label="YTD", step="year", stepmode="todate"),
                                dict(count=1, label="1y", step="year", stepmode="backward"),
                                dict(step="all")])),
                   rangeslider=dict(visible=True),
                   type="date"))

fig.show()
```

```
/Users/serahverg/opt/miniconda3/lib/python3.9/site-packages/statsmodels/tsa/base/tsa_model.py:471: ValueWarnin
g: A date index has been provided, but it has no associated frequency information and so will be ignored when
e.g. forecasting.
  self._init_dates(dates, freq)
/Users/serahverg/opt/miniconda3/lib/python3.9/site-packages/statsmodels/tsa/base/tsa_model.py:471: ValueWarnin
g: A date index has been provided, but it has no associated frequency information and so will be ignored when
e.g. forecasting.
  self._init_dates(dates, freq)
 This problem is unconstrained.
```

```
RUNNING THE L-BFGS-B CODE

           * * *

Machine precision = 2.220D-16
 N =            4     M =           10

At X0         0 variables are exactly at the bounds

At iterate    0    f=  6.99022D+00    |proj g|=  2.13767D-01

At iterate    5    f=  6.91983D+00    |proj g|=  8.96163D-03

At iterate   10    f=  6.91604D+00    |proj g|=  1.92923D-02

At iterate   15    f=  6.90910D+00    |proj g|=  6.87951D-04

At iterate   20    f=  6.85316D+00    |proj g|=  2.51542D-03

At iterate   25    f=  6.85198D+00    |proj g|=  1.18056D-04

At iterate   30    f=  6.83599D+00    |proj g|=  1.44062D-05

           * * *

Tit   = total number of iterations
Tnf   = total number of function evaluations
Tnint = total number of segments explored during Cauchy searches
Skip  = number of BFGS updates skipped
Nact  = number of active bounds at final generalized Cauchy point
Projg = norm of the final projected gradient
F     = final function value

           * * *

   N    Tit     Tnf  Tnint  Skip  Nact     Projg        F
    4     31     59      1     0     0   4.370D-08   6.836D+00
  F =   6.8359885173454185

CONVERGENCE: NORM_OF_PROJECTED_GRADIENT_<=_PGTOL
Test RMSE: 240.829

/Users/serahverg/opt/miniconda3/lib/python3.9/site-packages/statsmodels/tsa/base/tsa_model.py:834: ValueWarnin
g: No supported index is available. Prediction results will be given with an integer index beginning at `start
`.
  return get_prediction_index(
```



The Interactive Graph

Above one can see the interactive graoh generated. The users can view the data at different aggregation levels. Though some might seem irrelevant here due to nature of data ie the dataset used in the project spans for only a period of 2 months.

The Y-Axis depicts the ride demand at a given time and X-Axis depicts the timestamp

If you hover over the plotted lines of the graph, you will get values of timestamp and ride demand at each point on the plotted line.

For the ease of convinience of the user, there's a small window availalble at the bottom of the graph, which can be used to adjust the size of time window between which you want to see and observe the data.

The top right corner of the graph shows some self explanatory icons when you hover in that area. It provides option to download the graph, zoom in or out, reset the graph to original state

Thhis graph aims at given the decision makers a comprehensive overview of the data and ensure that it answers as many question as possible, pictorially.

In [ ]:

In [ ]:

Part 2

XgBoost for demand forecasting

In [164]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
```

In [165]:
```
import xgboost as xgb
from sklearn.metrics import mean_squared_error
```

In [166]:
```
import datetime
# Calling the fromtimestamp() function to
# extract datetime from the given timestamp
# from google.colab import drive
# drive.mount('/content/drive')
ride = pd.read_csv("rideshare_kaggle.csv")
# ride = pd.read_csv("/content/drive/MyDrive/Colab Notebooks/rideshare_kaggle.csv")
ride['timestamp']=pd.to_datetime(ride['timestamp'], unit='s')
ride.dropna(axis=0,inplace=True)
ride["Date"] = ride["timestamp"].dt.date
ride["Time"] = ride["timestamp"].dt.time
ride["Weekday"]=ride["timestamp"].dt.day_name()
```

Data preprocessing and cleaning

In [167]:
```
ride['datetime']=pd.to_datetime(ride['datetime'])
```

In [168]:
```
ride['Date']=pd.to_datetime(ride['Date'])
```

In [169]:
```
grouped = ride.groupby(ride['datetime'].dt.floor('h'))['id'].count().reset_index()
grouped.rename(columns={'id': 'total_rides'}, inplace=True)
```

In [170]:
```
grouped.head()
```

Out[170]:

	datetime	total_rides
0	2018-11-26 03:00:00	77
1	2018-11-26 04:00:00	390
2	2018-11-26 05:00:00	616
3	2018-11-26 06:00:00	1462
4	2018-11-26 07:00:00	925

In [171]:
```

grouped.set_index('datetime',inplace=True)
```

In [172]:
```
len(grouped)
```

Out[172]: 332

Xgboost

In [ ]:
```

```

This graph above shows the distribution of count of rides by hour for each day of the two month dataset

In [173]:
```
grouped.index=pd.to_datetime(grouped.index)
grouped.head()
```

Out[173]:

	total_rides
datetime	
2018-11-26 03:00:00	77
2018-11-26 04:00:00	390
2018-11-26 05:00:00	616
2018-11-26 06:00:00	1462
2018-11-26 07:00:00	925

In [174]:
```
train = grouped.loc[grouped.index < '2018-12-16']
test = grouped.loc[grouped.index >= '2018-12-16']
```

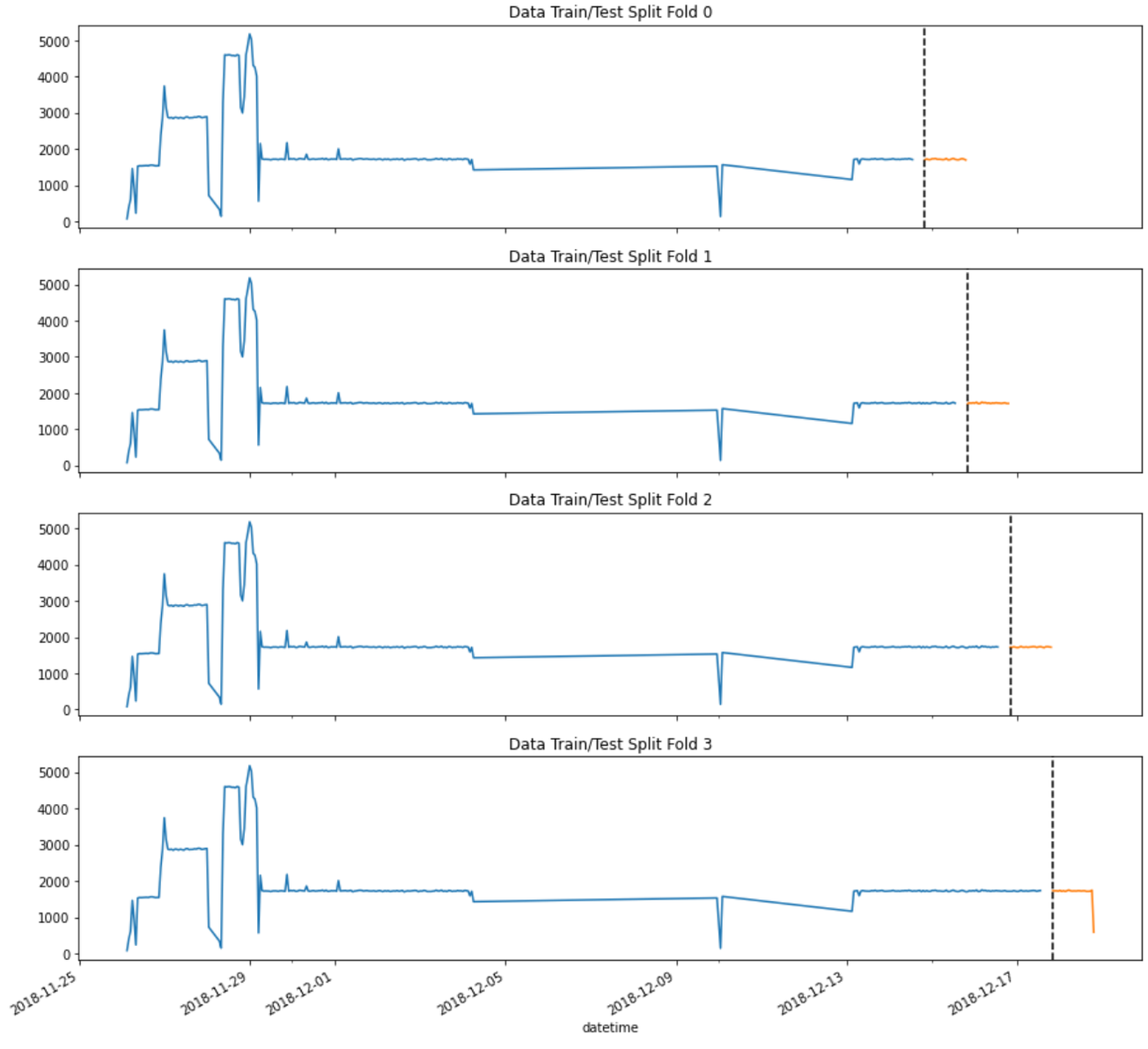
In [175]:
```
def create_features(df):
    """
    Create time series features based on time series index.
    """
    df = df.copy()
    df['hour'] = df.index.hour
    df['dayofweek'] = df.index.dayofweek
    df['quarter'] = df.index.quarter
    df['month'] = df.index.month
    df['dayofmonth'] = df.index.day

    return df

df = create_features(grouped)
```

In [176]:
```
train = create_features(train)
test = create_features(test)

FEATURES = ['hour', 'dayofweek', 'quarter', 'month']
TARGET = 'total_rides'

X_train = train[FEATURES]
y_train = train[TARGET]

X_test = test[FEATURES]
y_test = test[TARGET]
```

In [177]:
```
from sklearn.model_selection import TimeSeriesSplit

tss = TimeSeriesSplit(n_splits=4, test_size=24, gap=6)
df = df.sort_index()
```

Here we will split the data using timeseries split which is more suitable for timeseries data

In [178]:
```
fig, axs = plt.subplots(4, 1, figsize=(15, 15), sharex=True)

fold = 0
for train_idx, val_idx in tss.split(df):
    train = df.iloc[train_idx]
    test = df.iloc[val_idx]
    train['total_rides'].plot(ax=axs[fold],
                          label='Training Set',
                          title=f'Data Train/Test Split Fold {fold}')
    test['total_rides'].plot(ax=axs[fold],
                         label='Test Set')
    axs[fold].axvline(test.index.min(), color='black', ls='--')
    fold += 1
plt.show()
```



In [179]:
```
#adding the lags for timeseries
```

In [180]:
```
def add_lags(df):
    target_map = df['total_rides'].to_dict()

    df['lag1'] = (df.index - pd.Timedelta('24 hours')).map(target_map)
    df['lag2'] = (df.index - pd.Timedelta('48 hours')).map(target_map)
    df['lag3'] = (df.index - pd.Timedelta('72 hours')).map(target_map)
    return df
```

In [181]:
```
df = add_lags(df)
```

Running XGboost with CV-4 folds

In [182]:
```
fold = 0
preds = []
scores = []
for train_idx, val_idx in tss.split(df):
    train = df.iloc[train_idx]
    test = df.iloc[val_idx]

    X_train = train.drop(['total_rides'],axis=1)
    y_train = train['total_rides']

    X_test = test.drop(['total_rides'],axis=1)
    y_test = test['total_rides']

    reg = xgb.XGBRegressor(base_score=0.5, booster='gbtree',
                           n_estimators=1000,
                           early_stopping_rounds=50,
                           objective='reg:linear',
                           max_depth=2,
                           learning_rate=0.01)
    reg.fit(X_train, y_train,
            eval_set=[(X_train, y_train), (X_test, y_test)],
            verbose=100)

    y_pred = reg.predict(X_test)
    preds.extend(y_pred)
    score = np.sqrt(mean_squared_error(y_test, y_pred))
    scores.append(score)
```

```
[18:48:42] WARNING: /Users/runner/work/xgboost/xgboost/python-package/build/temp.macosx-11.0-arm64-cpython-38/
xgboost/src/objective/regression_obj.cu:213: reg:linear is now deprecated in favor of reg:squarederror.
[0]     validation_0-rmse:2181.71447    validation_1-rmse:1704.38670
[100]   validation_0-rmse:963.92043     validation_1-rmse:657.85133
[200]   validation_0-rmse:562.68902     validation_1-rmse:211.29765
[254]   validation_0-rmse:486.86099     validation_1-rmse:242.55130
[18:48:43] WARNING: /Users/runner/work/xgboost/xgboost/python-package/build/temp.macosx-11.0-arm64-cpython-38/
xgboost/src/objective/regression_obj.cu:213: reg:linear is now deprecated in favor of reg:squarederror.
[0]     validation_0-rmse:2141.22729    validation_1-rmse:1706.70551
[100]   validation_0-rmse:939.61204     validation_1-rmse:645.69093
[200]   validation_0-rmse:555.70149     validation_1-rmse:236.22796
[300]   validation_0-rmse:446.46264     validation_1-rmse:137.79013
[400]   validation_0-rmse:388.65751     validation_1-rmse:130.52979
[416]   validation_0-rmse:382.84399     validation_1-rmse:132.27299
[18:48:43] WARNING: /Users/runner/work/xgboost/xgboost/python-package/build/temp.macosx-11.0-arm64-cpython-38/
xgboost/src/objective/regression_obj.cu:213: reg:linear is now deprecated in favor of reg:squarederror.
[0]     validation_0-rmse:2107.16266    validation_1-rmse:1705.25756
[100]   validation_0-rmse:917.20887     validation_1-rmse:644.85178
[200]   validation_0-rmse:541.66789     validation_1-rmse:259.02237
[300]   validation_0-rmse:429.67495     validation_1-rmse:123.92924
[400]   validation_0-rmse:383.07769     validation_1-rmse:82.35604
[500]   validation_0-rmse:353.49827     validation_1-rmse:72.40632
[548]   validation_0-rmse:342.59889     validation_1-rmse:73.48144
[18:48:43] WARNING: /Users/runner/work/xgboost/xgboost/python-package/build/temp.macosx-11.0-arm64-cpython-38/
xgboost/src/objective/regression_obj.cu:213: reg:linear is now deprecated in favor of reg:squarederror.
[0]     validation_0-rmse:2077.98023    validation_1-rmse:1674.10524
[100]   validation_0-rmse:897.45380     validation_1-rmse:633.48225
[200]   validation_0-rmse:525.32540     validation_1-rmse:304.03599
[300]   validation_0-rmse:415.35458     validation_1-rmse:239.53361
[400]   validation_0-rmse:371.84084     validation_1-rmse:231.64806
[444]   validation_0-rmse:354.13786     validation_1-rmse:231.92640
```

In [183]:
```
X_test = test.drop(['total_rides'],axis=1)
len(X_test)
```

Out[183]: 24

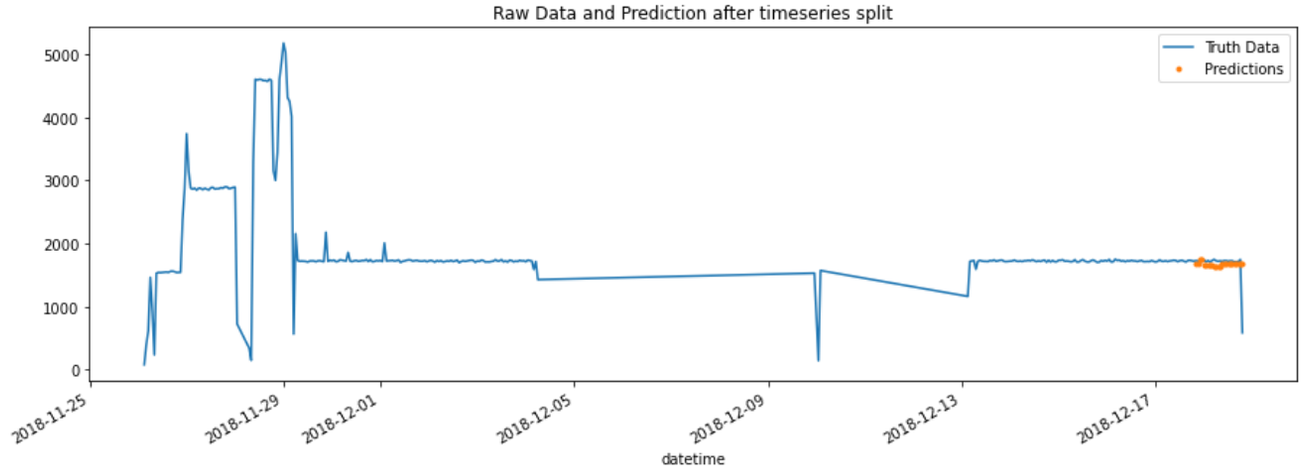
In [184]:
```
print(f'Score across folds {np.mean(scores):0.4f}')
print(f'Fold scores:{scores}')
```

```
Score across folds 159.8264
Fold scores:[207.14867310622387, 128.99194837891062, 71.52720065302476, 231.63772666583952]
```

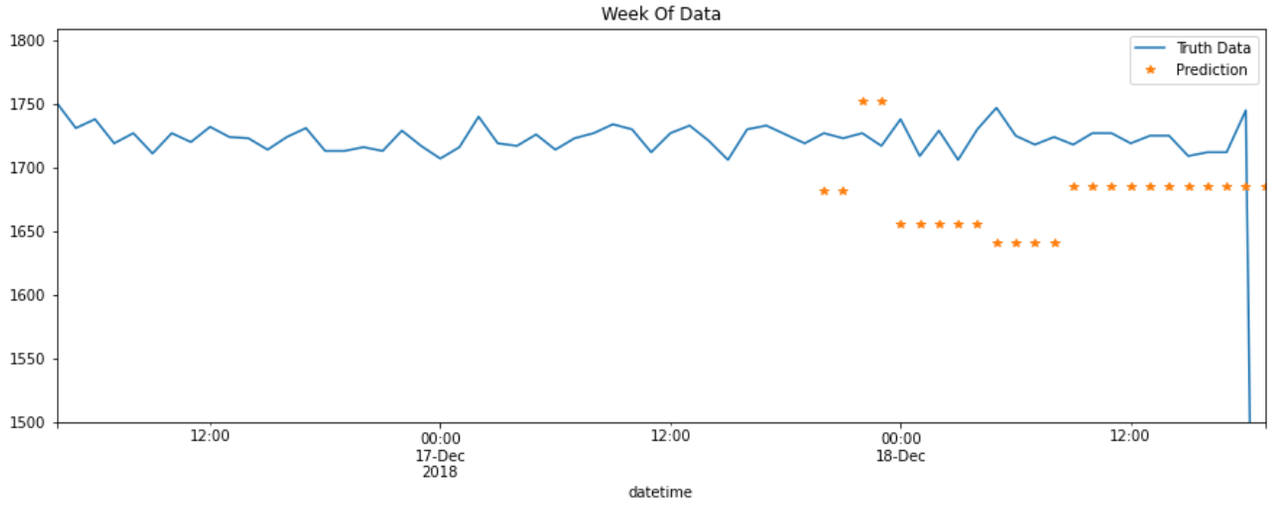
Average of the four folds gives a RMSE score of 159.82- Better than previous model but still can be improved

In [185]:
```
test['prediction'] = reg.predict(X_test)
# df = df.merge(test[['prediction']], how='left', left_index=True, right_index=True)
ax = df[['total_rides']].plot(figsize=(15, 5))
test['prediction'].plot(ax=ax, style='.')
plt.legend(['Truth Data', 'Predictions'])
ax.set_title('Raw Data and Prediction after timeseries split')
plt.show()
```

```
/var/folders/1j/vhrxwjz94tn0tsz00qlnqw8m0000gn/T/ipykernel_50466/577023577.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 (https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
```



In [186]:
```
ax = df.loc[(df.index > '2018-12-16 03:00:00 ') & (df.index < '2018-12-25 00:00:00')]['total_rides'] \
    .plot(figsize=(15, 5), title='Week Of Data').set_ylim(1500)
test.loc[(test.index > '2018-12-16 03:00:00') & (test.index < '2018-12-25 00:00:00')]['prediction'] \
    .plot(style='*')
plt.legend(['Truth Data','Prediction'])
plt.show()
```



Hyper parameter tuning and outlier handling

Removing data lesser than 750 counts and greater that 4500

In [187]:
```
df = df.query('total_rides > 750').copy()
df=df.query('total_rides < 4500').copy()
```

In [188]:
```
df.head()
```

Out[188]:

datetime	total_rides	hour	dayofweek	quarter	month	dayofmonth	lag1	lag2	lag3
2018-11-26 06:00:00	1462	6	0	4	11	26	NaN	NaN	NaN
2018-11-26 07:00:00	925	7	0	4	11	26	NaN	NaN	NaN
2018-11-26 09:00:00	1526	9	0	4	11	26	NaN	NaN	NaN
2018-11-26 10:00:00	1541	10	0	4	11	26	NaN	NaN	NaN
2018-11-26 11:00:00	1538	11	0	4	11	26	NaN	NaN	NaN

In [189]:
```
# Running the model with reduced dataset --> removed few outliers
```

hyperparameter tuning for XGboost

In [190]:
```
from sklearn.model_selection import GridSearchCV
```

In [191]:
```
# param_grid = {
#     'max_depth': [3, 5, 7],
#     'learning_rate': [0.1, 0.01, 0.001],
#     'n_estimators': [500,1000],
#     'subsample': [0.5, 0.7, 1.0],
#     'gamma': [0, 1, 5]
# }



# # Instantiate the XGBRegressor model
# xgb_model = xgb.XGBRegressor()

# # Instantiate the GridSearchCV object
# grid_search = GridSearchCV(estimator=xgb_model, param_grid=param_grid, cv=5, n_jobs=-1)

# # Fit the grid search object to the data
# grid_search.fit(X_train, y_train,
#             eval_set=[(X_train, y_train), (X_test, y_test)],
#             verbose=100)

# # Print the best hyperparameters and the best score
# print("Best Hyperparameters:", grid_search.best_params_)
# print("Best Score:", grid_search.best_score_)
```

In [192]:
```
fold = 0
preds = []
scores = []
for train_idx, val_idx in tss.split(df):
    train = df.iloc[train_idx]
    test = df.iloc[val_idx]

    X_train = train.drop(['total_rides'],axis=1)
    y_train = train['total_rides']

    X_test = test.drop(['total_rides'],axis=1)
    y_test = test['total_rides']

    reg = xgb.XGBRegressor(base_score=0.5, booster='gbtree',
                           n_estimators=1000,
                           early_stopping_rounds=50,
                           objective='reg:linear',
                           max_depth=3,
                           learning_rate=0.01)
    reg.fit(X_train, y_train,
            eval_set=[(X_train, y_train), (X_test, y_test)],
            verbose=100)

    y_pred = reg.predict(X_test)
    preds.append(y_pred)
    score = np.sqrt(mean_squared_error(y_test, y_pred))
    scores.append(score)
```

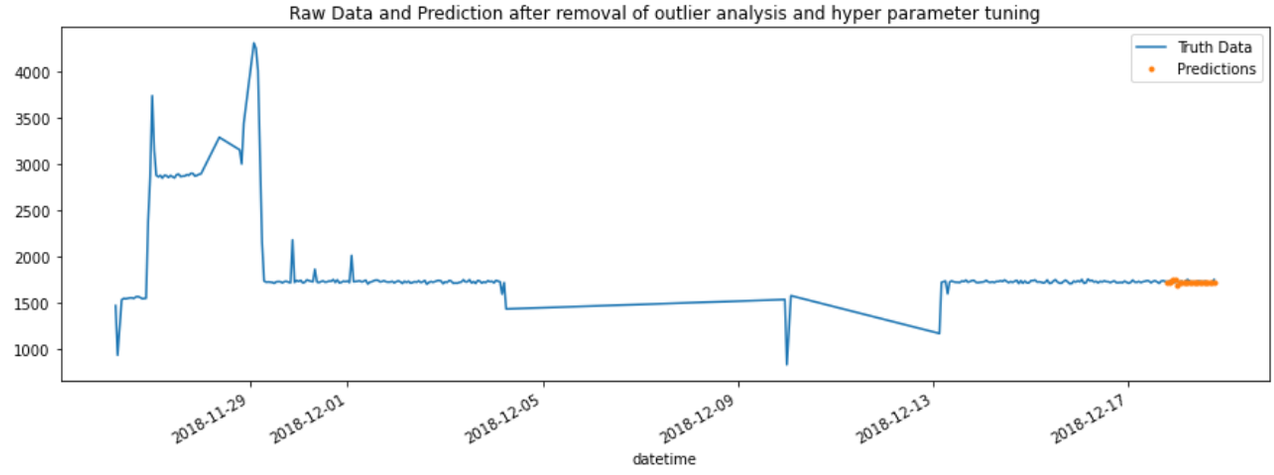
```
[18:48:43] WARNING: /Users/runner/work/xgboost/xgboost/python-package/build/temp.macosx-11.0-arm64-cpython-38/
xgboost/src/objective/regression_obj.cu:213: reg:linear is now deprecated in favor of reg:squarederror.
[0]     validation_0-rmse:1975.06649    validation_1-rmse:1705.22225
[100]   validation_0-rmse:766.86968     validation_1-rmse:646.10623
[200]   validation_0-rmse:328.73144     validation_1-rmse:251.69250
[300]   validation_0-rmse:178.62684     validation_1-rmse:99.80584
[400]   validation_0-rmse:142.03118     validation_1-rmse:39.40804
[500]   validation_0-rmse:123.45327     validation_1-rmse:20.21245
[600]   validation_0-rmse:110.02609     validation_1-rmse:18.89406
[625]   validation_0-rmse:107.54439     validation_1-rmse:21.34378
[18:48:44] WARNING: /Users/runner/work/xgboost/xgboost/python-package/build/temp.macosx-11.0-arm64-cpython-38/
xgboost/src/objective/regression_obj.cu:213: reg:linear is now deprecated in favor of reg:squarederror.
[0]     validation_0-rmse:1948.73540    validation_1-rmse:1705.82727
[100]   validation_0-rmse:753.34738     validation_1-rmse:643.42144
[200]   validation_0-rmse:320.84429     validation_1-rmse:247.91900
[300]   validation_0-rmse:171.41097     validation_1-rmse:97.78880
[400]   validation_0-rmse:134.91290     validation_1-rmse:38.81671
[500]   validation_0-rmse:120.73975     validation_1-rmse:20.88188
[600]   validation_0-rmse:109.69807     validation_1-rmse:18.80126
[689]   validation_0-rmse:104.81529     validation_1-rmse:19.48189
[18:48:44] WARNING: /Users/runner/work/xgboost/xgboost/python-package/build/temp.macosx-11.0-arm64-cpython-38/
xgboost/src/objective/regression_obj.cu:213: reg:linear is now deprecated in favor of reg:squarederror.
[0]     validation_0-rmse:1927.13091    validation_1-rmse:1704.63727
[100]   validation_0-rmse:742.19174     validation_1-rmse:639.74495
[200]   validation_0-rmse:313.22194     validation_1-rmse:244.04385
[300]   validation_0-rmse:164.15734     validation_1-rmse:101.42436
[400]   validation_0-rmse:128.05705     validation_1-rmse:56.81602
[500]   validation_0-rmse:116.32331     validation_1-rmse:41.89932
[580]   validation_0-rmse:107.57256     validation_1-rmse:49.49365
[18:48:44] WARNING: /Users/runner/work/xgboost/xgboost/python-package/build/temp.macosx-11.0-arm64-cpython-38/
xgboost/src/objective/regression_obj.cu:213: reg:linear is now deprecated in favor of reg:squarederror.
[0]     validation_0-rmse:1908.92583    validation_1-rmse:1705.82005
[100]   validation_0-rmse:732.76504     validation_1-rmse:639.18481
[200]   validation_0-rmse:307.19728     validation_1-rmse:240.88337
[300]   validation_0-rmse:159.15177     validation_1-rmse:93.40121
[400]   validation_0-rmse:123.36282     validation_1-rmse:36.30854
[500]   validation_0-rmse:109.71175     validation_1-rmse:17.98449
[600]   validation_0-rmse:102.18787     validation_1-rmse:16.08712
[636]   validation_0-rmse:100.19123     validation_1-rmse:16.67866
```

In [193]:
```
print(f'Score across folds {np.mean(scores):0.4f}')
print(f'Fold scores:{scores}')
```

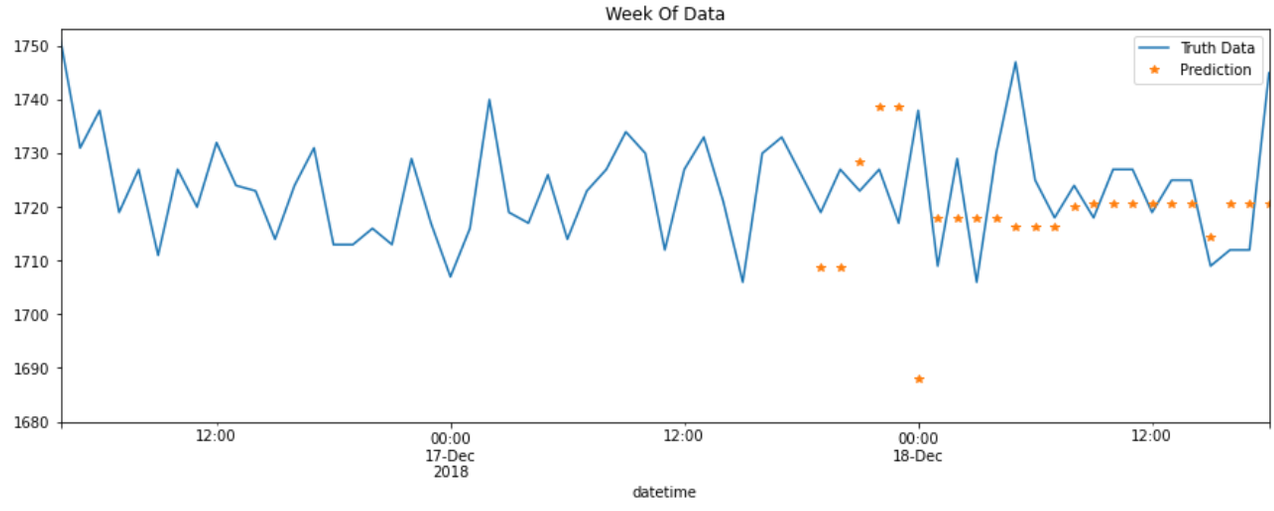
```
Score across folds 23.1732
Fold scores:[18.046861944204206, 18.67970362522241, 40.149977999915876, 15.81624221445832]
```

In [194]:
```
test['prediction'] = reg.predict(X_test)
# df = df.merge(test[['prediction']], how='left', left_index=True, right_index=True)
ax = df[['total_rides']].plot(figsize=(15, 5))
test['prediction'].plot(ax=ax, style='.')
plt.legend(['Truth Data', 'Predictions'])
ax.set_title('Raw Data and Prediction after removal of outlier analysis and hyper parameter tuning')
plt.show()
```

```
/var/folders/1j/vhrxwjz94tn0tsz00qlnqw8m0000gn/T/ipykernel_50466/2478601107.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 (https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
```



In [195]:
```
ax = df.loc[(df.index > '2018-12-16 03:00:00 ') & (df.index < '2018-12-25 00:00:00')]['total_rides'] \
    .plot(figsize=(15, 5), title='Week Of Data').set_ylim(1680)
test.loc[(test.index > '2018-12-16 03:00:00') & (test.index < '2018-12-25 00:00:00')]['prediction'] \
    .plot(style='*')
plt.legend(['Truth Data','Prediction'])
plt.show()
```



As we can see above that the model is able to predict much better and is giving an RMSE score of 23 ---> way better than Sarimax

In [196]:
```
X_test.head()
```

Out[196]:

datetime	hour	dayofweek	quarter	month	dayofmonth	lag1	lag2	lag3
2018-12-17 19:00:00	19	0	4	12	17	1713.0	1702.0	1734.0
2018-12-17 20:00:00	20	0	4	12	17	1716.0	1703.0	1707.0
2018-12-17 21:00:00	21	0	4	12	17	1713.0	1729.0	1730.0
2018-12-17 22:00:00	22	0	4	12	17	1729.0	1719.0	1711.0
2018-12-17 23:00:00	23	0	4	12	17	1717.0	1732.0	1711.0

In [197]:
```
print(f'Score across folds {np.mean(scores):0.4f}')
print(f'Fold scores:{scores}')
```

```
Score across folds 23.1732
Fold scores:[18.046861944204206, 18.67970362522241, 40.149977999915876, 15.81624221445832]
```

In [198]:
```
df.index.max()
```

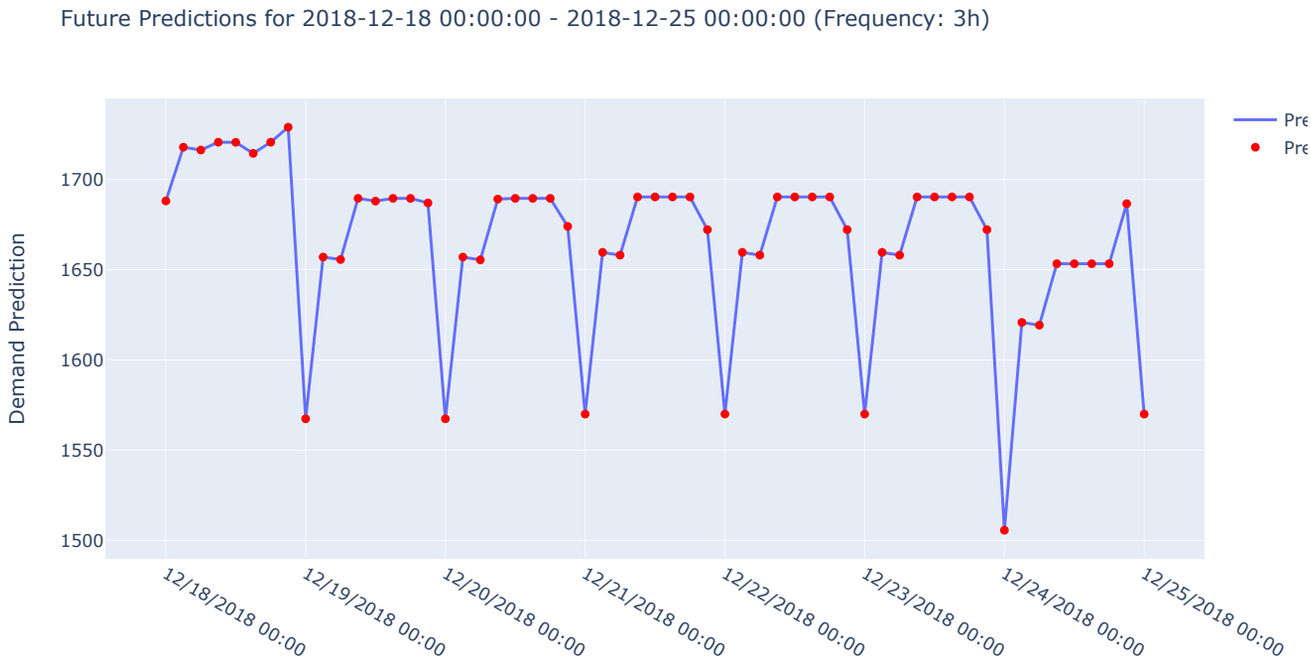
Out[198]:
```
Timestamp('2018-12-18 18:00:00')
```

Future Predictions - Interactive graph Xgboost

In [199]:
```
stall plotly==5.2.1
stall ipywidgets

ywidgets as widgets
otly.graph_objs as go
otly.io as pio
ly.subplots import make_subplots

date range slider
e_slider = widgets.SelectionRangeSlider(
ns=future_w_features.index,
=(0, len(future_w_features.index) - 1),
iption='Date Range:',
t=widgets.Layout(width='80%')


frequency dropdown
down = widgets.Dropdown(
ns=['1h', '2h', '3h', '4h', '6h', '12h', '24h'],
='3h',
iption='Frequency:',
t=widgets.Layout(width='20%')


update function for the plot
e_plot(date_range, freq):
 the selected date range
_date, end_date = date_range

ample the dataframe at the selected frequency
red_df = future_w_features.resample(freq).mean()
red_df = filtered_df.loc[start_date:end_date]

ate traces for the line plot and scatter plot
trace = go.Scatter(x=filtered_df.index, y=filtered_df['pred'], mode='lines', name='Prediction')
er_trace = go.Scatter(x=filtered_df.index, y=filtered_df['pred'], mode='markers', marker=dict(color='red'), name

ate subplot with the line plot and scatter plot
 make_subplots(rows=1, cols=1, shared_xaxes=True)
dd_trace(line_trace, row=1, col=1)
dd_trace(scatter_trace, row=1, col=1)

 x-axis and y-axis labels
pdate_xaxes(title_text='Date', tickformat='%m/%d/%Y %H:%M')
pdate_yaxes(title_text='Demand Prediction')

 plot title
_text = f"Future Predictions for {start_date} - {end_date} (Frequency: {freq})"
pdate_layout(title_text=title_text, title_font_size=15)
w plot
enderers.default = "notebook"
.renderers.default = "browser"
how()

 the date range slider and frequency dropdown
nteract(update_plot, date_range=date_range_slider, freq=freq_dropdown);
```

Date Range: 2018-12-18 00:00:00-201

Frequency: 3h



Steps to interact: The user can use the date range slider to select the range of the dates for which he would like the model to predict future demand.
The user can also change the frequency of the data points in time for the prediction(eg 3h,6h etc)*

This Graph shows the future demand prediction using XGBoost- best model.

It is created using plotly and ipwidgets

In [ ]:

Interactive Predictor for Price

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

In [2]:
```
# from google.colab import drive
# drive.mount('/content/drive')
```

In [3]:
```
ride = pd.read_csv("rideshare_kaggle.csv")
# ride = pd.read_csv("/content/drive/MyDrive/Colab Notebooks/rideshare_kaggle.csv")
```

In [4]:
```
#dropping rows with missing values
ride.dropna(axis=0,inplace=True)
```

In [5]:
```
import datetime

# Calling the fromtimestamp() function to
# extract datetime from the given timestamp
ride['timestamp']=pd.to_datetime(ride['timestamp'], unit='s')
timestamp = ride['timestamp']
timestamp
```

Out[5]:
```
0         2018-12-16 09:30:07.890000128
1         2018-11-27 02:00:23.676999936
2         2018-11-28 01:00:22.197999872
3         2018-11-30 04:53:02.749000192
4         2018-11-29 03:49:20.223000064
                      ...
693065    2018-12-01 23:53:06.000000000
693066    2018-12-01 23:53:06.000000000
693067    2018-12-01 23:53:06.000000000
693069    2018-12-01 23:53:06.000000000
693070    2018-12-01 23:53:06.000000000
Name: timestamp, Length: 637976, dtype: datetime64[ns]
```

In [6]:
```
ride.head(3)
```

Out[6]:

	id	timestamp	hour	day	month	datetime	timezone	source	destination	cab_type	...	precipIntensityMax	uvIndex
0	424553bb-7174-41ea-aeb4-fe06d4f4b9d7	2018-12-16 09:30:07.890000128	9	16	12	2018-12-16 09:30:07	America/New_York	Haymarket Square	North Station	Lyft	...	0.1276	154497
1	4bd23055-6827-41c6-b23b-3c491f24e74d	2018-11-27 02:00:23.676999936	2	27	11	2018-11-27 02:00:23	America/New_York	Haymarket Square	North Station	Lyft	...	0.1300	154325
2	981a3613-77af-4620-a42a-0c0866077d1e	2018-11-28 01:00:22.197999872	1	28	11	2018-11-28 01:00:22	America/New_York	Haymarket Square	North Station	Lyft	...	0.1064	154333

3 rows × 57 columns

In [7]:
```
# drop unnecessary column
ride = ride.drop(columns = ['id','timezone','timestamp', 'datetime', 'hour','month', 'day','product_id','latitud
```

In [8]:
```
numeric = ride._get_numeric_data()
numeric
```

Out[8]:

	price	distance	surge_multiplier	temperature	apparentTemperature	precipIntensity	precipProbability	humidity	windSpeed	windGust	...	ozo
0	5.0	0.44	1.0	42.34	37.12	0.0000	0.0	0.68	8.66	9.17	...	30
1	11.0	0.44	1.0	43.58	37.35	0.1299	1.0	0.94	11.98	11.98	...	29
2	7.0	0.44	1.0	38.33	32.93	0.0000	0.0	0.75	7.33	7.33	...	31
3	26.0	0.44	1.0	34.38	29.63	0.0000	0.0	0.73	5.28	5.28	...	29
4	9.0	0.44	1.0	37.44	30.88	0.0000	0.0	0.70	9.14	9.14	...	34
...	...	...	...	...	...	...	...	...	...	...	...	
693065	9.5	1.00	1.0	37.05	37.05	0.0000	0.0	0.74	2.34	2.87	...	27
693066	13.0	1.00	1.0	37.05	37.05	0.0000	0.0	0.74	2.34	2.87	...	27
693067	9.5	1.00	1.0	37.05	37.05	0.0000	0.0	0.74	2.34	2.87	...	27
693069	27.0	1.00	1.0	37.05	37.05	0.0000	0.0	0.74	2.34	2.87	...	27
693070	10.0	1.00	1.0	37.05	37.05	0.0000	0.0	0.74	2.34	2.87	...	27

637976 rows × 31 columns

Encoding Categorical Variables

In [9]:
```
# get all categorical variables
obj = ride.dtypes == object
categorical_cols = ride.columns[obj]
print(categorical_cols)
```

```
Index(['source', 'destination', 'cab_type', 'name', 'icon'], dtype='object')
```

In [10]:
```
# Determine how many extra columns would be created
num_ohc_cols = (ride[categorical_cols]
                .apply(lambda x: x.nunique())
                .sort_values(ascending=False))
# No need to encode if there is only one value
small_num_ohc_cols = num_ohc_cols.loc[num_ohc_cols>1]

# Number of one-hot columns is one less than the number of categories
small_num_ohc_cols -= 1

small_num_ohc_cols.sum()
```

Out[10]:
```
40
```

In [11]:

```
%%time

# encode categorical variables
from sklearn.preprocessing import OneHotEncoder, LabelEncoder

# The encoders
le = LabelEncoder()
ohc = OneHotEncoder()

for col in num_ohc_cols.index:

    # Integer encode the string categories
    dat = le.fit_transform(ride[col]).astype(int)

    # One hot encode the data--this returns a sparse array
    new_dat = ohc.fit_transform(dat.reshape(-1,1))

    # Create unique column names
    n_cols = new_dat.shape[1]
    col_names = ['_'.join([col, str(x)]) for x in range(n_cols)]

    # Create the new dataframe
    new_df = pd.DataFrame(new_dat.toarray(),
                          index = ride.index,
                          columns = col_names)
    value_colname_mapping = sorted(list(zip(ride[col].unique(), col_names)), key=lambda x: x[0])

    # Print original values and their one-hot encoded versions in the same order
    print(f"Original values and their one-hot encoded versions for {col}:")
    for original_value, one_hot_colname in value_colname_mapping:
        one_hot_value = np.zeros(n_cols)
        one_hot_value[col_names.index(one_hot_colname)] = 1
        print(f"{original_value}: {one_hot_value}")
    print("\n")

    # Append the new data to the dataframe
    ride = pd.concat([ride, new_df], axis=1)

    # Remove the original column from the dataframe
    ride = ride.drop(col, axis=1)
```

```
Original values and their one-hot encoded versions for source:
Back Bay: [0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
Beacon Hill: [0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0.]
Boston University: [0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0.]
Fenway: [0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0.]
Financial District: [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0.]
Haymarket Square: [1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
North End: [0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
North Station: [0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0.]
Northeastern University: [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1.]
South Station: [0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0.]
Theatre District: [0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0.]
West End: [0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0.]


Original values and their one-hot encoded versions for destination:
Back Bay: [0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0.]
Beacon Hill: [0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0.]
Boston University: [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1.]
Fenway: [0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0.]
Financial District: [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0.]
Haymarket Square: [0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
North End: [0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0.]
North Station: [1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
Northeastern University: [0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
South Station: [0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0.]
Theatre District: [0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0.]
West End: [0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]


Original values and their one-hot encoded versions for name:
Black: [0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0.]
Black SUV: [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0.]
Lux: [0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
Lux Black: [0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0.]
Lux Black XL: [0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
Lyft: [0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
Lyft XL: [0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0.]
Shared: [1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
UberPool: [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1.]
UberX: [0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0.]
UberXL: [0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0.]
WAV: [0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0.]


Original values and their one-hot encoded versions for icon:
 clear-day : [0. 0. 0. 0. 0. 1. 0.]
 clear-night : [0. 0. 1. 0. 0. 0. 0.]
 cloudy : [0. 0. 0. 1. 0. 0. 0.]
 fog : [0. 0. 0. 0. 1. 0. 0.]
 partly-cloudy-day : [0. 0. 0. 0. 0. 0. 1.]
 partly-cloudy-night : [1. 0. 0. 0. 0. 0. 0.]
 rain : [0. 1. 0. 0. 0. 0. 0.]


Original values and their one-hot encoded versions for cab_type:
Lyft: [1. 0.]
Uber: [0. 1.]


CPU times: user 1.02 s, sys: 856 ms, total: 1.88 s
Wall time: 2.19 s
```

In [12]:
```
ride['timestamp'] = timestamp
```

In [13]:
```
# Get the date of every ride
ride["Date"] = ride["timestamp"].dt.date
del ride['timestamp']
```

In [14]:
```
# split the data
import statsmodels.api as sm

ride_train = ride.sample(frac=0.8, random_state=1)
ride_test = ride.drop(ride_train.index)
```

In [15]:
```
# Choose the features to be used

del ride_train['Date']
del ride_test['Date']

X_train = ride_train.drop(['price'], axis=1) # was ride_train[features]
y_train = ride_train['price']

X_test = ride_test.drop(['price'], axis=1) #col was ride_test[features]
y_test = ride_test['price']

# We must add an intercept as the standard model doesn't automatically fit one
X_train = sm.add_constant(X_train)
X_test = sm.add_constant(X_test) #added this line
# fit the data to the model
model = sm.OLS(y_train, X_train).fit()

print(model.summary())
```

OLS Regression Results

Dep. Variable:	price	R-squared:	0.929
Model:	OLS	Adj. R-squared:	0.929
Method:	Least Squares	F-statistic:	1.005e+05
Date:	Thu, 27 Apr 2023	Prob (F-statistic):	0.00
Time:	21:05:03	Log-Likelihood:	-1.1905e+06
No. Observations:	510381	AIC:	2.381e+06
Df Residuals:	510314	BIC:	2.382e+06
Df Model:	66		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	248.4859	105.392	2.358	0.018	41.921	455.051
distance	2.8837	0.004	719.381	0.000	2.876	2.892
surge_multiplier	18.4984	0.037	493.482	0.000	18.425	18.572
temperature	0.0078	0.013	0.583	0.560	-0.019	0.034
apparentTemperature	-0.0031	0.005	-0.666	0.505	-0.012	0.006
precipIntensity	-0.3131	0.282	-1.109	0.267	-0.866	0.240
precipProbability	0.0093	0.058	0.162	0.871	-0.104	0.122
humidity	0.0384	0.419	0.091	0.927	-0.783	0.860
windSpeed	-0.0087	0.005	-1.586	0.113	-0.019	0.002
windGust	0.0044	0.003	1.731	0.083	-0.001	0.009
visibility	-0.0012	0.002	-0.685	0.493	-0.005	0.002
temperatureHigh	0.1007	0.122	0.825	0.410	-0.139	0.340
temperatureLow	0.0031	0.004	0.811	0.417	-0.004	0.011
apparentTemperatureHigh	-0.0680	0.071	-0.957	0.338	-0.207	0.071
apparentTemperatureLow	-0.0025	0.003	-0.779	0.436	-0.009	0.004
dewPoint	-0.0025	0.012	-0.210	0.834	-0.025	0.020
pressure	0.0003	0.001	0.334	0.739	-0.002	0.002
windBearing	0.0001	6.46e-05	1.769	0.077	-1.24e-05	0.000
cloudCover	-0.0394	0.027	-1.442	0.149	-0.093	0.014
uvIndex	-0.0091	0.009	-0.961	0.336	-0.028	0.009
visibility.1	-0.0012	0.002	-0.685	0.493	-0.005	0.002
ozone	-0.0002	0.000	-0.408	0.683	-0.001	0.001
sunriseTime	0.0004	0.000	2.394	0.017	7.12e-05	0.001
sunsetTime	-0.0004	0.000	-2.381	0.017	-0.001	-6.97e-05
moonPhase	-0.0926	0.060	-1.534	0.125	-0.211	0.026
precipIntensityMax	0.0234	0.144	0.162	0.871	-0.259	0.306
uvIndexTime	8.03e-07	3.88e-06	0.207	0.836	-6.81e-06	8.41e-06
temperatureMin	0.0058	0.004	1.362	0.173	-0.003	0.014
temperatureMax	-0.1111	0.122	-0.914	0.361	-0.349	0.127
apparentTemperatureMin	-0.0003	0.005	-0.067	0.947	-0.010	0.010
apparentTemperatureMax	0.0715	0.071	1.011	0.312	-0.067	0.210
source_0	20.6130	8.804	2.341	0.019	3.357	37.869
source_1	20.3093	8.804	2.307	0.021	3.053	37.565
source_2	20.4314	8.804	2.321	0.020	3.176	37.687
source_3	20.6229	8.804	2.342	0.019	3.367	37.879
source_4	20.9634	8.804	2.381	0.017	3.707	38.219
source_5	21.0806	8.804	2.394	0.017	3.825	38.336
source_6	21.2619	8.804	2.415	0.016	4.006	38.518
source_7	20.6979	8.804	2.351	0.019	3.442	37.954
source_8	20.4113	8.804	2.318	0.020	3.155	37.667
source_9	20.8743	8.804	2.371	0.018	3.618	38.130
source_10	21.1327	8.804	2.400	0.016	3.877	38.389
source_11	20.6951	8.804	2.351	0.019	3.439	37.951
destination_0	20.6952	8.804	2.351	0.019	3.439	37.951
destination_1	20.3877	8.804	2.316	0.021	3.132	37.644
destination_2	20.7434	8.804	2.356	0.018	3.488	37.999
destination_3	20.4278	8.804	2.320	0.020	3.172	37.684
destination_4	21.1007	8.804	2.397	0.017	3.845	38.357
destination_5	20.9639	8.804	2.381	0.017	3.708	38.220
destination_6	20.8081	8.804	2.363	0.018	3.552	38.064
destination_7	20.9032	8.804	2.374	0.018	3.647	38.159
destination_8	20.7504	8.804	2.357	0.018	3.495	38.006
destination_9	20.7179	8.804	2.353	0.019	3.462	37.974
destination_10	20.9284	8.804	2.377	0.017	3.672	38.184
destination_11	20.6673	8.804	2.347	0.019	3.411	37.923
name_0	25.4262	8.804	2.888	0.004	8.170	42.682
name_1	35.1815	8.804	3.996	0.000	17.926	52.437
name_2	21.1437	8.804	2.402	0.016	3.888	38.400
name_3	26.4210	8.804	3.001	0.003	9.165	43.677
name_4	35.6969	8.804	4.055	0.000	18.441	52.953
name_5	12.9609	8.804	1.472	0.141	-4.295	30.217
name_6	18.6668	8.804	2.120	0.034	1.411	35.923
name_7	10.0828	8.804	1.145	0.252	-7.173	27.339
name_8	13.6425	8.804	1.550	0.121	-3.613	30.898
name_9	14.6498	8.804	1.664	0.096	-2.606	31.906
name_10	20.5761	8.804	2.337	0.019	3.320	37.832
name_11	14.6456	8.804	1.663	0.096	-2.610	31.901
icon_0	35.5577	15.094	2.356	0.018	5.974	65.141
icon_1	35.5659	15.090	2.357	0.018	5.989	65.142
icon_2	35.5971	15.094	2.358	0.018	6.013	65.181
icon_3	35.5585	15.090	2.356	0.018	5.982	65.135
icon_4	35.5943	15.093	2.358	0.018	6.012	65.177
icon_5	35.6068	15.094	2.359	0.018	6.023	65.190
icon_6	35.6136	15.094	2.359	0.018	6.030	65.197

```
cab_type_0                124.9721     52.825      2.366       0.018       21.437      228.507
cab_type_1                124.1217     52.825      2.350       0.019       20.587      227.657
==============================================================================
Omnibus:                   196527.542   Durbin-Watson:                   1.996
Prob(Omnibus):                  0.000   Jarque-Bera (JB):          3697891.816
Skew:                           1.379   Prob(JB):                         0.00
Kurtosis:                      15.895   Cond. No.                     4.90e+23
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The smallest eigenvalue is 1.52e-23. This might indicate that there are
strong multicollinearity problems or that the design matrix is singular.
```

In [16]:
```
# Linear Regression
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error

LR = LinearRegression()
LR = LR.fit(X_train, y_train)
y_train_pred = LR.predict(X_train)
y_test_pred = LR.predict(X_test)

# MSE
print('train mse: ', np.sqrt(mean_squared_error(y_train, y_train_pred)))
print('test mse: ',  np.sqrt(mean_squared_error(y_test, y_test_pred)))

# R2 score
from sklearn.metrics import r2_score
print('r2 score: ', r2_score(y_test, y_test_pred))
```

```
train mse:  2.4931385221179987
test mse:  2.4840929981446926
r2 score:  0.9286870042006168
```

XG BOOST

In [17]:
```
import xgboost as xg
from sklearn.metrics import mean_squared_error as MSE
```

In [18]:
```
#This is to deal with the error assocoated with uniquness of our features(duplicate columns)
X_train = X_train.loc[:,~X_train.columns.duplicated()].copy()
X_test = X_test.loc[:,~X_test.columns.duplicated()].copy()
```

In [19]:
```
X_train
```

Out[19]:

	const	distance	surge_multiplier	temperature	apparentTemperature	precipIntensity	precipProbability	humidity	windSpeed	windGust	...	na
658311	1.0	2.16	1.0	41.34	34.55	0.0000	0.00	0.78	11.92	15.18	...	
547209	1.0	2.25	1.0	51.84	51.84	0.0000	0.00	0.80	3.58	5.30	...	
225076	1.0	1.56	1.0	36.95	36.95	0.0000	0.00	0.86	1.63	2.32	...	
360104	1.0	0.74	1.0	41.35	35.62	0.0000	0.00	0.60	9.29	11.60	...	
635115	1.0	3.14	1.0	37.71	31.92	0.0000	0.00	0.90	7.78	7.78	...	
...	...	...	...	...	...	...	...	...	...	...	...	...
450255	1.0	3.58	1.0	40.77	35.14	0.0000	0.00	0.63	8.76	14.90	...	
299568	1.0	3.39	1.0	47.87	47.87	0.0000	0.00	0.93	2.77	4.54	...	
642890	1.0	3.14	1.0	40.38	35.18	0.0000	0.00	0.71	7.73	11.57	...	
608760	1.0	2.87	1.0	31.71	22.64	0.0025	0.17	0.61	11.52	20.61	...	
278771	1.0	3.07	1.0	24.71	12.26	0.0000	0.00	0.51	15.00	24.98	...	

510381 rows × 76 columns

In [21]:
```
from sklearn.model_selection import GridSearchCV
```

In [1]:
```
# Various hyper-parameters to tune
# xgb1 = xg.XGBRegressor()
# parameters = {'nthread':[-1], #when use hyperthread, xgboost may become slower
#               'objective':['reg:squarederror'],
#               'learning_rate': [.03, 0.05, .07], #so called `eta` value
#               'max_depth': [5, 6, 7],
#               'min_child_weight': [4],
#               'silent': [1],
#               'subsample': [0.7],
#               'colsample_bytree': [0.7],
#               'n_estimators': [500]}

# xgb_grid = GridSearchCV(xgb1,
#                         parameters,
#                         cv = 2,
#                         n_jobs = 10,
#                         verbose=False)

# xgb_grid.fit(X_train,
#          y_train)
```

In [ ]:
```
print('CV R2:', xgb_grid.best_score_, 5)
XG_OSR2 = r2_score(y_test, xgb_grid.predict(X_test))
print('OSR2:', round(XG_OSR2, 5))
print('Best parameters', xgb_grid.best_params_)
```

In [ ]:
```
xgb1.fit(X_train,
         y_train)

model_file = 'xgb_model.bin'
xgb1.save_model(model_file)
```

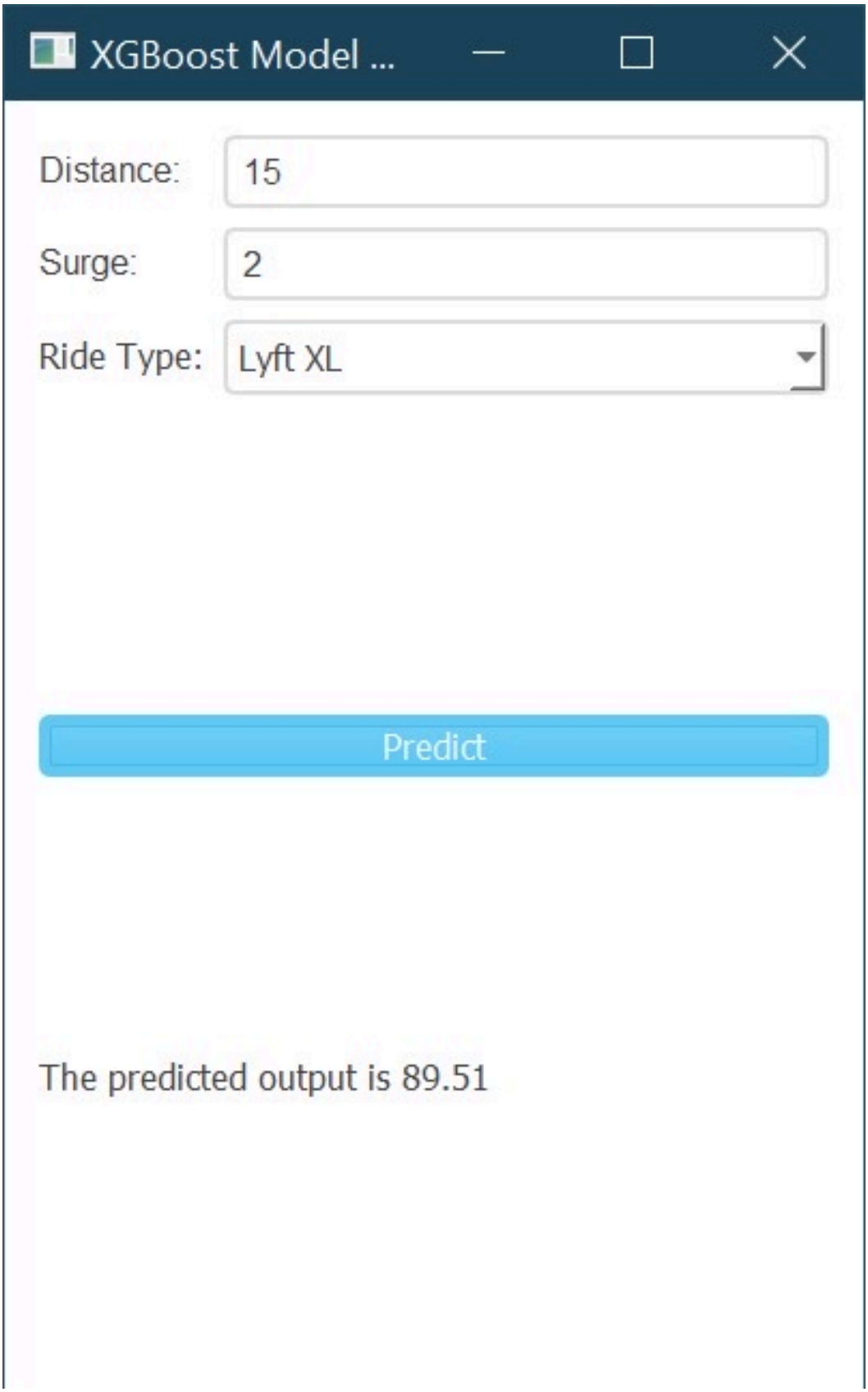
Since XG Boost has the best accuracy (96%), we are using this ML to generate our interactive predictor

In [ ]:

```
import sys
import xgboost as xgb
import pandas as pd
from PyQt5.QtWidgets import QApplication, QWidget, QLabel, QLineEdit, QPushButton, QVBoxLayout, QFormLayout, QC
from PyQt5.QtGui import QFont, QIcon

# Define the input variables
input_vars = ['Distance', 'Surge', 'Lyft Shared', 'Lyft Lux', 'Lyft', 'Lyft Lux Black XL', 'Lyft XL', 'Lyft Lux

# Load the trained XGBoost model
xgb_model = xgb.Booster()
xgb_model.load_model(model_file)

class MainWindow(QWidget):
    def __init__(self):
        super().__init__()

        # Set custom font
        font = QFont("Arial", 16)

        # Create the GUI elements
        self.labels = []
        self.line_edits = []
        self.button = QPushButton('Predict')
        self.result_label = QLabel()
        self.ride_type_combobox = QComboBox()

        # Set the style sheet for the window
        self.setStyleSheet('''
            QWidget {
                background-color: #FFF;
                color: #444;
            }

            QLabel {
                font-size: 16px;
            }

            QLineEdit {
                font-size: 16px;
                padding: 5px;
                border: 2px solid #DDD;
                border-radius: 5px;
            }

            QPushButton {
                font-size: 16px;
                padding: 5px;
                border: none;
                border-radius: 5px;
                background-color: #5FCCF5;
                color: #FFF;
            }

            QPushButton:hover {
                background-color: #3FA6C5;
            }

            QLabel#result_label {
                font-size: 20px;
                color: #444;
                margin-top: 20px;
            }

            QComboBox {
                font-size: 16px;
                padding: 5px;
                border: 2px solid #DDD;
                border-radius: 5px;
            }
        ''')

        # Create the input elements
        for var in input_vars:
            label = QLabel(f'{var}:')
            label.setFont(font)
            self.labels.append(label)

        self.distance_edit = QLineEdit()
        self.distance_edit.setFont(font)
        self.surge_edit = QLineEdit()
        self.surge_edit.setFont(font)
        self.line_edits = [self.distance_edit, self.surge_edit]

        for var in input_vars[2:]:
            self.ride_type_combobox.addItem(var)

        # Create a form layout
```

```
        form_layout = QFormLayout()

        # Add the input elements to the layout
        form_layout.addRow(self.labels[0], self.distance_edit)
        form_layout.addRow(self.labels[1], self.surge_edit)
        form_layout.addRow(QLabel('Ride Type:'), self.ride_type_combobox)

        # Create a vertical layout
        layout = QVBoxLayout()

        # Add the form layout and other elements to the layout
        layout.addLayout(form_layout)
        layout.addWidget(self.button)
        layout.addWidget(self.result_label)

        # Set the layout for the window
        self.setLayout(layout)

        # Connect the button to the predict function
        self.button.clicked.connect(self.predict_output)

    def predict_output(self):
        # Get user inputs
        input_data = [0] * len(input_vars)
        input_data[0] = float(self.distance_edit.text())
        input_data[1] = float(self.surge_edit.text())

        # Set the selected ride type to 1
        selected_ride_type = self.ride_type_combobox.currentIndex() + 2
        input_data[selected_ride_type] = 1

        # Prepare input data as a DataFrame
        input_df = pd.DataFrame([input_data], columns=input_vars)

         # Make the prediction
        prediction = xgb_model.predict(xgb.DMatrix(input_df))[0]

        # Set the result label text
        self.result_label.setText(f"The predicted output is {prediction:.2f}")

# Create the QApplication and MainWindow
app = QApplication(sys.argv)
app.setStyle('Fusion') # Set the Fusion style for the application
app.setWindowIcon(QIcon("app_icon.png"))  # Set the app icon
window = MainWindow()

# Set the window title and dimensions
window.setWindowTitle('XGBoost Model Predictor')
window.setGeometry(100, 100, 400, 600)

# Show the MainWindow
window.show()

# Run the QApplication event loop
sys.exit(app.exec_())
```

In [ ]:


XGBoost Model ...
Distance:
15
Surge:
2
Ride Type:
Lyft XL
Predict
The predicted output is 89.51

Part 4

Import Libraries

In [26]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from geopy.distance import geodesic
from IPython.display import Image
```

In [27]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, cal
l drive.mount("/content/drive", force_remount=True).
```

Parsing Location Coordinates

In [28]:
```
ride = pd.read_csv("/content/drive/MyDrive/Colab Notebooks/rideshare_kaggle.cs

# ride = pd.read_csv('rideshare_kaggle.csv')

x = ride[["latitude","longitude"]]
```

In [29]:
```
coords = {
    'Haymarket Square': [42.364,-71.060],
    'Back Bay' : [42.3503,-71.0810],
    'North End': [42.3647,-71.0542],
    'North Station': [42.3661,-71.0631],
    'Beacon Hill': [42.3588,-71.0707],
    'Boston University': [42.3505,-71.1054],
    'Fenway': [42.3505,-71.1054],
    'South Station': [42.3519,-71.0551],
    'Theatre District': [42.352,-71.065],
    'West End': [42.3661,-71.0631],
    'Financial District' : [42.3559,-71.0550],
    'Northeastern University': [42.3398,-71.0892]
}

def parse_lat(document):

  ret = [0,0]
  if document:
    lat = coords[document][0]
  return lat

def parse_long(document):

  ret = [0,0]
  if document:
    lon = coords[document][1]
  return lon
```

In [30]:
```
s_coords1 = ride["source"].apply(parse_lat)
s_coords2 = ride["source"].apply(parse_long)
d_coords1 = ride["destination"].apply(parse_lat)
d_coords2 = ride["destination"].apply(parse_long)

l1 = s_coords1.append(d_coords1, ignore_index = True)
l2 = s_coords2.append(d_coords2, ignore_index = True)
```

```
<ipython-input-30-a8d0bef7471d>:6: FutureWarning: The series.append method i
s deprecated and will be removed from pandas in a future version. Use panda
s.concat instead.
  l1 = s_coords1.append(d_coords1, ignore_index = True)
<ipython-input-30-a8d0bef7471d>:7: FutureWarning: The series.append method i
s deprecated and will be removed from pandas in a future version. Use panda
s.concat instead.
  l2 = s_coords2.append(d_coords2, ignore_index = True)
```

In [31]:
```
x = pd.concat([l1,l2],axis=1)
print(x)
```

```
                0        1
0         42.3640 -71.0600
1         42.3640 -71.0600
2         42.3640 -71.0600
3         42.3640 -71.0600
4         42.3640 -71.0600
...           ...      ...
1386137   42.3647 -71.0542
1386138   42.3647 -71.0542
1386139   42.3647 -71.0542
1386140   42.3647 -71.0542
1386141   42.3647 -71.0542

[1386142 rows x 2 columns]
```

In [32]:
```
pip install -U googlemaps
```

```
Looking in indexes: https://pypi.org/simple, (https://pypi.org/simple,) http
s://us-python.pkg.dev/colab-wheels/public/simple/ (https://us-python.pkg.de
v/colab-wheels/public/simple/)
Requirement already satisfied: googlemaps in /usr/local/lib/python3.9/dist-p
ackages (4.10.0)
Requirement already satisfied: requests<3.0,>=2.20.0 in /usr/local/lib/pytho
n3.9/dist-packages (from googlemaps) (2.27.1)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.9/dist
-packages (from requests<3.0,>=2.20.0->googlemaps) (3.4)
Requirement already satisfied: charset-normalizer~=2.0.0 in /usr/local/lib/p
ython3.9/dist-packages (from requests<3.0,>=2.20.0->googlemaps) (2.0.12)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.
9/dist-packages (from requests<3.0,>=2.20.0->googlemaps) (2022.12.7)
Requirement already satisfied: urllib3<1.27,>=1.21.1 in /usr/local/lib/pytho
n3.9/dist-packages (from requests<3.0,>=2.20.0->googlemaps) (1.26.15)
```

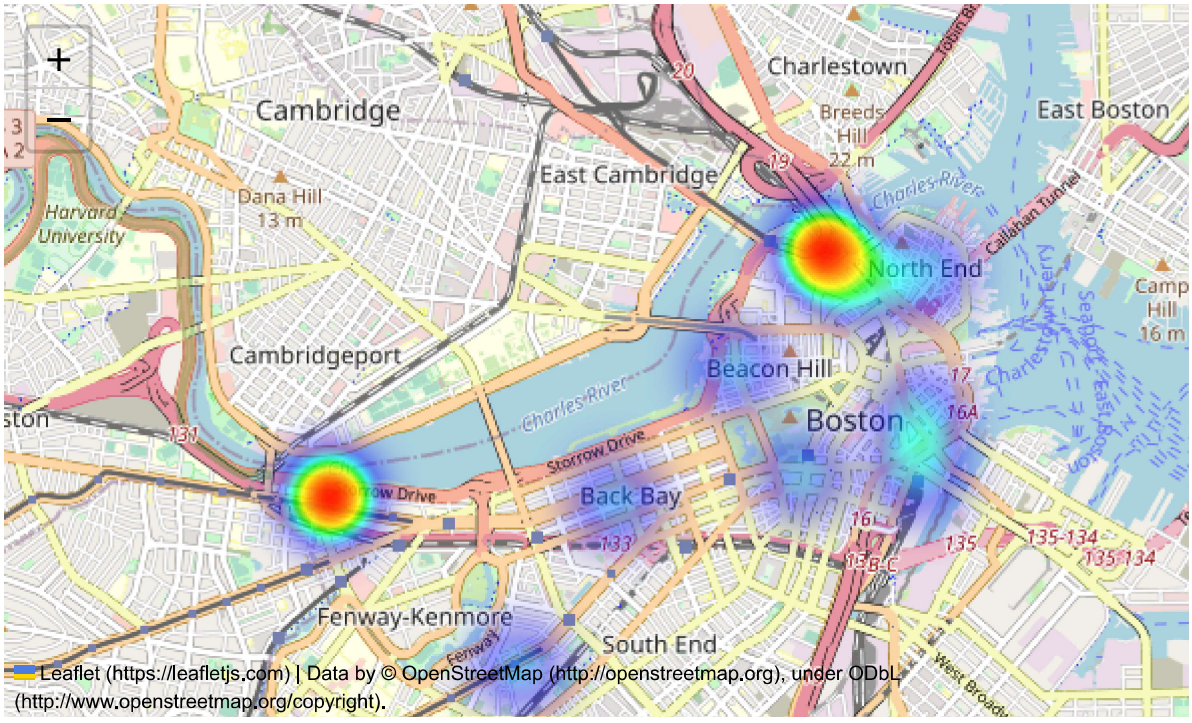
In [33]:
```
import googlemaps

gmaps = googlemaps.Client(key='AIzaSyB-Dvs504EXQNMt1L-S79Yil2I58dCejLQ')
```

Plotting Hubs using Heatmap

In [34]:
```
l1 = s_coords1.append(d_coords1, ignore_index = True)
l2 = s_coords2.append(d_coords2, ignore_index = True)

import folium
from folium.plugins import HeatMap
hmap = folium.Map(location=[42.35, -71.07], zoom_start=13, title="Most Common

hm_wide = HeatMap( list(zip(l1,l2)),
                   min_opacity=0.5,
                   radius=25, blur=20,
                   max_zoom=1,
                 )

hmap.add_child(hm_wide)
```

```
<ipython-input-34-5e371f4aa757>:1: FutureWarning: The series.append method i
s deprecated and will be removed from pandas in a future version. Use panda
s.concat instead.
  l1 = s_coords1.append(d_coords1, ignore_index = True)
<ipython-input-34-5e371f4aa757>:2: FutureWarning: The series.append method i
s deprecated and will be removed from pandas in a future version. Use panda
s.concat instead.
  l2 = s_coords2.append(d_coords2, ignore_index = True)
```

Out[34]:



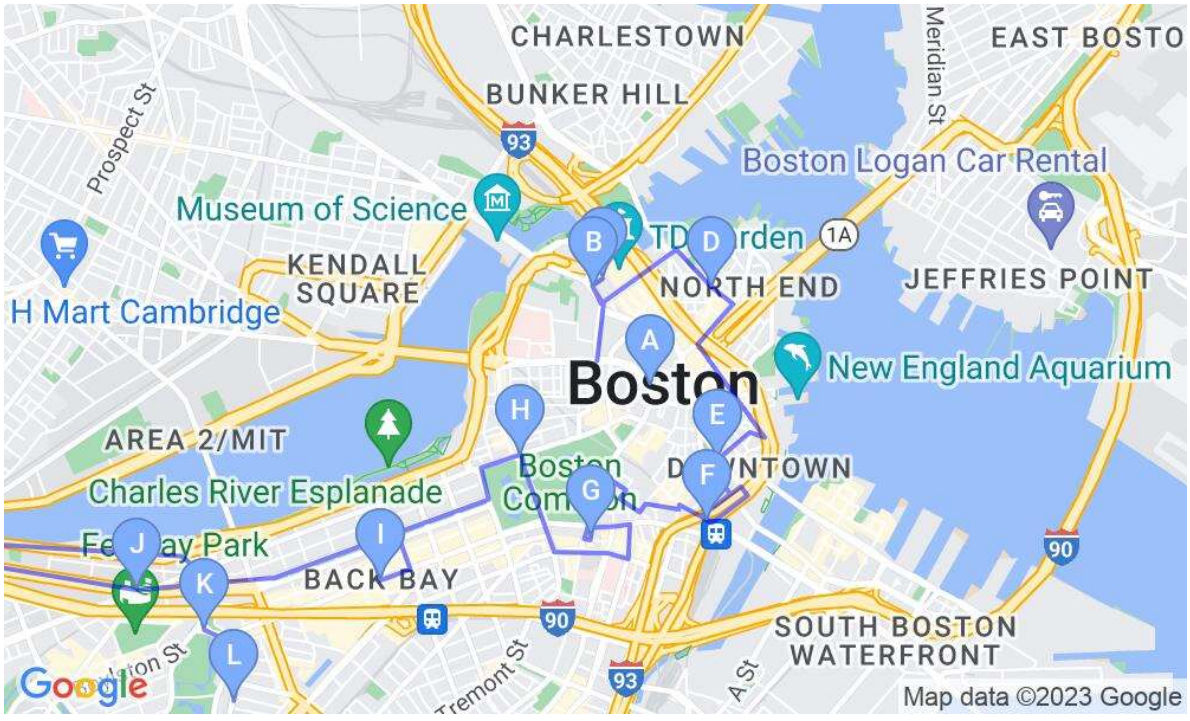
Example Route Through all Hubs

In [35]:
```
waypoints = [
    'Back Bay, Boston',
    'North End, Boston',
    'North Station, Boston',
    'Beacon Hill, Boston',
    'Boston University, Boston',
    'Fenway, Boston',
    'South Station, Boston',
    'Theatre District, Boston',
    'West End, Boston',
    'Financial District, Boston',
    ]

results = gmaps.directions(origin = 'Haymarket Square, Boston',
                                        destination = 'Northeastern Universit
                                        waypoints = waypoints,
                                        optimize_waypoints = True)

for i, leg in enumerate(results[0]["legs"]):
    print("Stop:" + str(i),
        leg["start_address"],
        "==> ",
        leg["end_address"],
        "distance: ",
        leg["distance"]["value"],
        "traveling Time: ",
        leg["duration"]["value"]
    )
```

```
Stop:0 Boston, MA, USA ==>  West End, Boston, MA, USA distance:  843 traveli
ng Time:  286
Stop:1 West End, Boston, MA, USA ==>  North Station, 135 Causeway St, Bosto
n, MA 02114, USA distance:  88 traveling Time:  33
Stop:2 North Station, 135 Causeway St, Boston, MA 02114, USA ==>  North End,
Boston, MA, USA distance:  1091 traveling Time:  522
Stop:3 North End, Boston, MA, USA ==>  Financial District, Boston, MA, USA d
istance:  1565 traveling Time:  494
Stop:4 Financial District, Boston, MA, USA ==>  South Station, 700 Atlantic
Ave, Boston, MA 02110, USA distance:  543 traveling Time:  157
Stop:5 South Station, 700 Atlantic Ave, Boston, MA 02110, USA ==>  Boston Th
eater District, Boston, MA, USA distance:  1948 traveling Time:  636
Stop:6 Boston Theater District, Boston, MA, USA ==>  Beacon Hill, Boston, M
A, USA distance:  1564 traveling Time:  535
Stop:7 Beacon Hill, Boston, MA, USA ==>  Back Bay, Boston, MA, USA distance:
1453 traveling Time:  346
Stop:8 Back Bay, Boston, MA, USA ==>  Boston, MA 02215, USA distance:  2044
traveling Time:  443
Stop:9 Boston, MA 02215, USA ==>  Fenway-Kenmore, Boston, MA, USA distance:
2970 traveling Time:  326
Stop:10 Fenway-Kenmore, Boston, MA, USA ==>  360 Huntington Ave, Boston, MA
02115, USA distance:  542 traveling Time:  97
```

In [36]:
```
marker_points = []
waypoints = []

#extract the location points from the previous directions function

for leg in results[0]["legs"]:
    leg_start_loc = leg["start_location"]
    marker_points.append(f'{leg_start_loc["lat"]},{leg_start_loc["lng"]}')
    for step in leg["steps"]:
        end_loc = step["end_location"]
        waypoints.append(f'{end_loc["lat"]},{end_loc["lng"]}')
last_stop = results[0]["legs"][-1]["end_location"]
marker_points.append(f'{last_stop["lat"]},{last_stop["lng"]}')

markers = [ "color:blue|size:mid|label:" + chr(65+i) + "|"
           + r for i, r in enumerate(marker_points)]
result_map = gmaps.static_map(
                center = waypoints[0],
                scale=2,
                zoom=13,
                size=[500, 300],
                format="jpg",
                maptype="roadmap",
                markers=markers,
                path="color:0x0000ff|weight:2|" + "|".join(waypoints))
```

In [37]:
```
with open('driving_route_map.jpg', 'wb') as img:
    for chunk in result_map:
        img.write(chunk)

i = Image('driving_route_map.jpg')
display(i)
```



Simple Route Between 2 Hubs

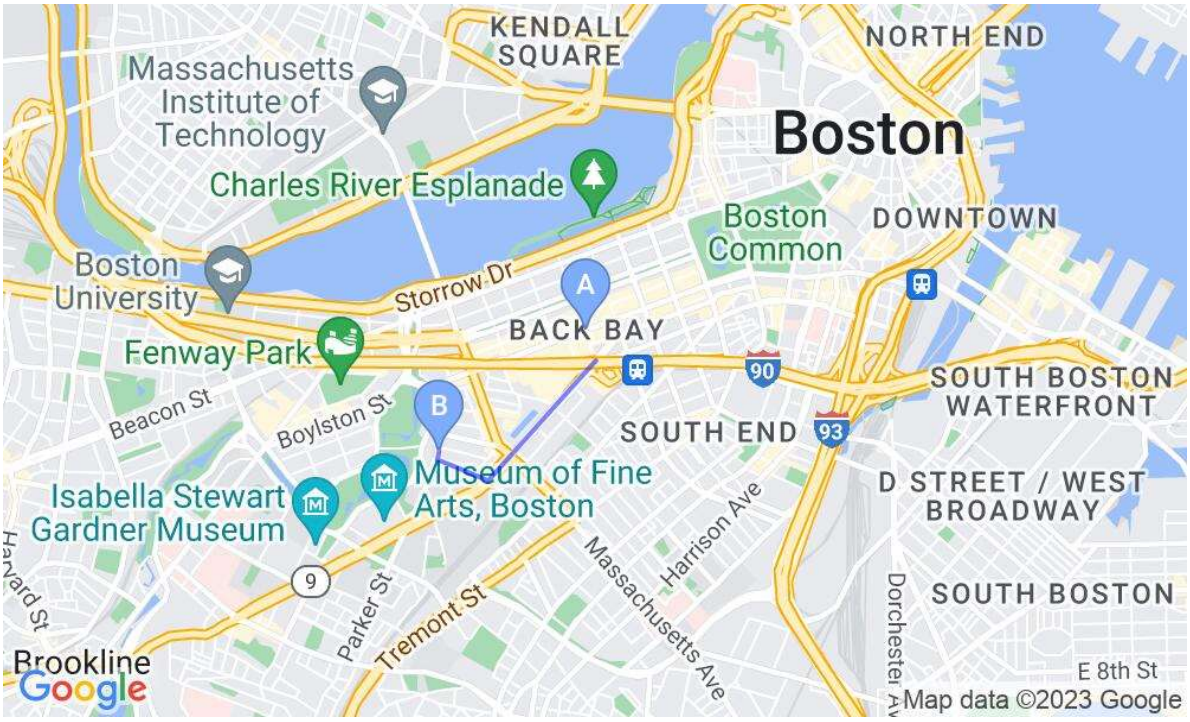
In [38]:
```
locations = [
    'Haymarket Square, Boston',
    'Back Bay, Boston',
    'North End, Boston',
    'North Station, Boston',
    'Beacon Hill, Boston',
    'Boston University, Boston',
    'Fenway, Boston',
    'South Station, Boston',
    'Theatre District, Boston',
    'West End, Boston',
    'Financial District, Boston',
    'Northeastern University, Boston']


location_geocodes = {}

for loc in locations:
  location_geocodes[loc] = gmaps.geocode(loc)
```

In [39]:

```
print('1. Haymarket Square')
print('2. Back Bay')
print('3. North End')
print('4. North Station')
print('5. Beacon Hill')
print('6. Boston University')
print('7. Fenway')
print('8. South Station')
print('9. Theatre District')
print('10. West End')
print('11. Financial District')
print('12. Northeastern University')

print("-----------------------------------------")
s = input("Select Source Option: ")
d = input("Select Destination Option: ")
source = locations[int(s)-1]
destination = locations[int(d)-1]
print("-----------------------------------------")
print('Selected Source: ',source)
print('Selected Destination: ',destination)


results = gmaps.directions(origin = source,
                                        destination = destination,
                                        optimize_waypoints = True)

marker_points = []
waypoints = []

#extract the location points from the previous directions function

for leg in results[0]["legs"]:
    leg_start_loc = leg["start_location"]
    marker_points.append(f'{leg_start_loc["lat"]},{leg_start_loc["lng"]}')
    for step in leg["steps"]:
        end_loc = step["end_location"]
        waypoints.append(f'{end_loc["lat"]},{end_loc["lng"]}')
last_stop = results[0]["legs"][-1]["end_location"]
marker_points.append(f'{last_stop["lat"]},{last_stop["lng"]}')

markers = [ "color:blue|size:mid|label:" + chr(65+i) + "|"
           + r for i, r in enumerate(marker_points)]

result_map2 = gmaps.static_map(
                 center = waypoints[0],
                 scale=2,
                 zoom=13,
                 size=[500, 300],
                 format="jpg",
                 maptype="roadmap",
                 markers=markers,
                 path="color:0x0000ff|weight:2|" + "|".join(waypoints))

with open('driving_route_map2.jpg', 'wb') as img:
    for chunk in result_map2:
        img.write(chunk)
```

```
i = Image('driving_route_map2.jpg')
display(i)
```

```
1. Haymarket Square
2. Back Bay
3. North End
4. North Station
5. Beacon Hill
6. Boston University
7. Fenway
8. South Station
9. Theatre District
10. West End
11. Financial District
12. Northeastern University
-----------------------------------------
Select Source Option: 2
Select Destination Option: 12
-----------------------------------------
Selected Source:  Back Bay, Boston
Selected Destination:  Northeastern University, Boston
```



Routing Any Possible Rideshare Routes Through Hubs

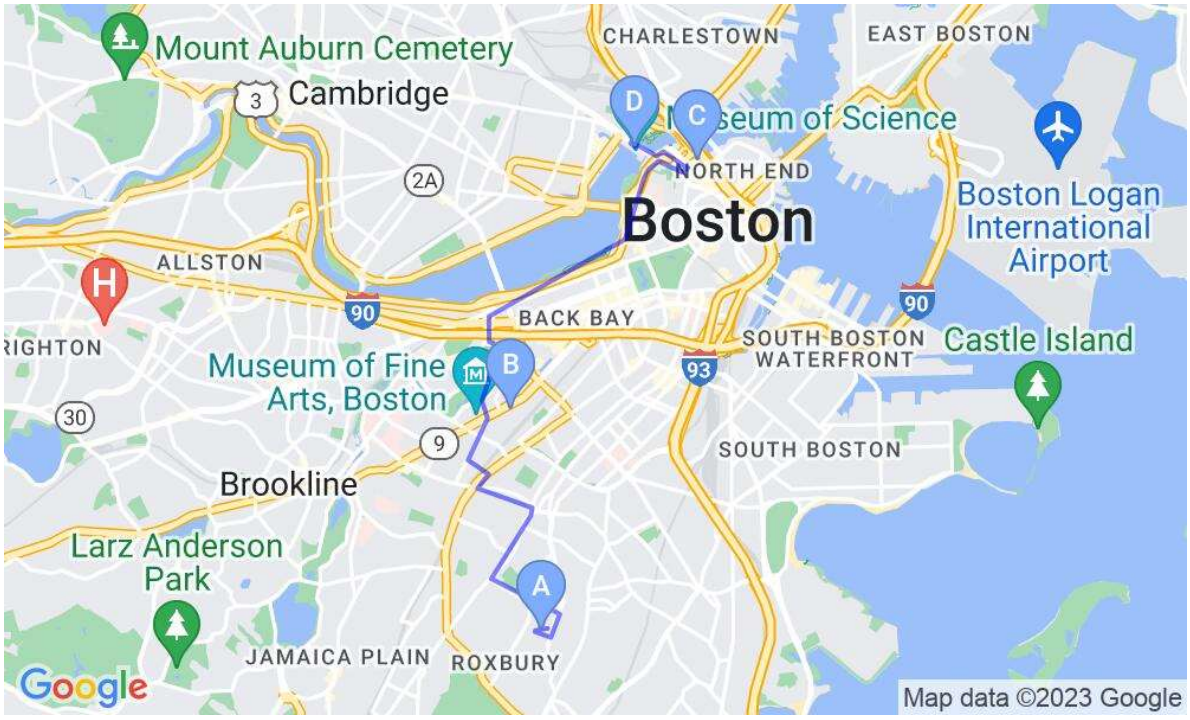
In [40]:

```
def route():

  print("-----------------------------------------")
  s = input("Select Source Address: ")
  d = input("Select Destination Address: ")
  print("-----------------------------------------")

  # s = "Nickerson Field"
  # d = "John F. Kennedy Presidential Library and Museum"


  source = gmaps.geocode(s)
  destination = gmaps.geocode(d)
  print(source[0]["formatted_address"])
  print(source[0]["geometry"]["location"]["lat"])
  print(source[0]["geometry"]["location"]["lng"])
  print()
  print(destination[0]["formatted_address"])
  print(destination[0]["geometry"]["location"]["lat"])
  print(destination[0]["geometry"]["location"]["lng"])
  print()


  max_s_dist = float('inf')
  max_d_dist = float('inf')
  closest_source_hub = locations[0]
  closest_dest_hub = locations[0]


  for loc,loc_geo in location_geocodes.items():
    curr_dist1 = gmaps.distance_matrix(origins=source[0]['formatted_address'],
                         destinations=loc_geo[0]["formatted_address"])['rows'][
    curr_dist2 = gmaps.distance_matrix(origins=loc_geo[0]['formatted_address']
                         destinations=destination[0]["formatted_address"])['rov

    if curr_dist1 < max_s_dist:
      closest_source_hub = loc
      max_s_dist = curr_dist1
    if curr_dist2 < max_d_dist:
      closest_dest_hub = loc
      max_d_dist = curr_dist2

  print("Closest hub to source = ",closest_source_hub)
  print("Closest hub to destination = ",closest_dest_hub)

  waypoints = [source[0]['formatted_address'],closest_source_hub,closest_dest_


  results = gmaps.directions(origin = source[0]['formatted_address'],
                                          destination = destination[0]['format
                                          waypoints = waypoints,
                                          optimize_waypoints = True)

  w =waypoints
  marker_points = []
  waypoints = []
```

```
#extract the location points from the previous directions function

for leg in results[0]["legs"]:
    leg_start_loc = leg["start_location"]
    marker_points.append(f'{leg_start_loc["lat"]},{leg_start_loc["lng"]}')
    for step in leg["steps"]:
        end_loc = step["end_location"]
        waypoints.append(f'{end_loc["lat"]},{end_loc["lng"]}')
last_stop = results[0]["legs"][-1]["end_location"]
marker_points.append(f'{last_stop["lat"]},{last_stop["lng"]}')

markers = [ "color:blue|size:mid|label:" + chr(65+i) + "|"
          + r for i, r in enumerate(w)]

result_map = gmaps.static_map(
                scale=2,
                zoom=12,
                size=[500, 300],
                format="jpg",
                maptype="roadmap",
                markers=markers,
                path="color:0x0000ff|weight:2|" + "|".join(waypoints))


with open('driving_route_map2.jpg', 'wb') as img:
    for chunk in result_map:
        img.write(chunk)

i = Image('driving_route_map2.jpg')
display(i)
```

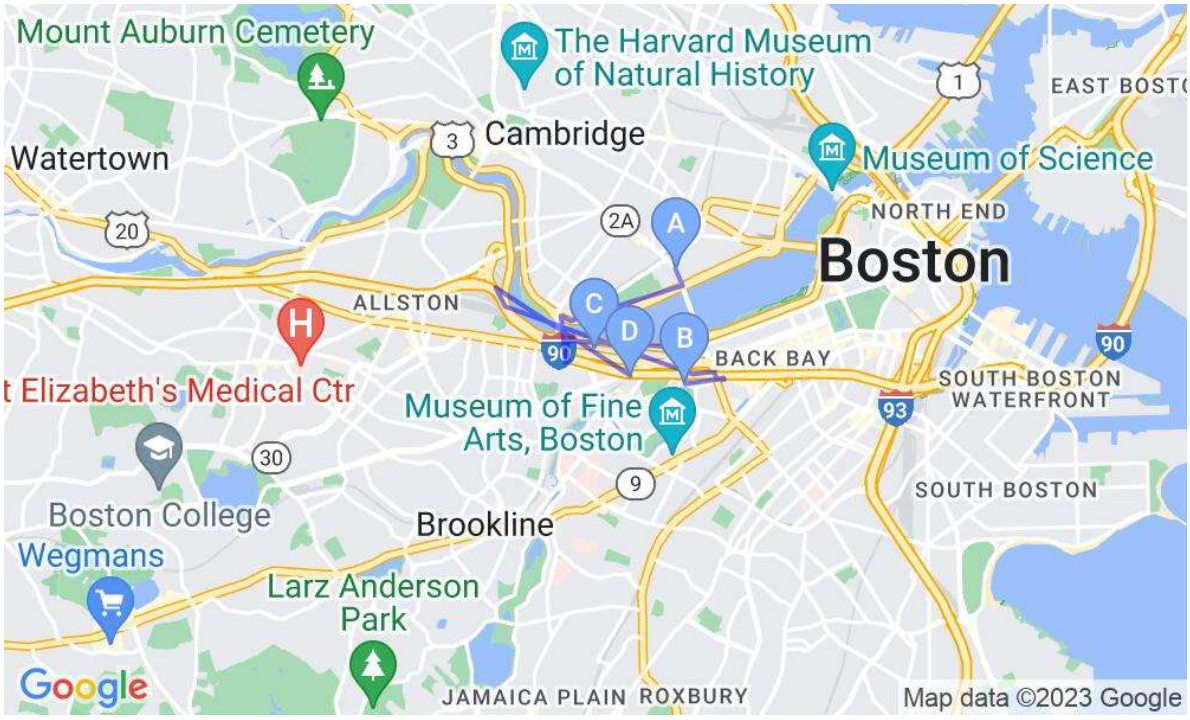
In [41]:
```
route()
```

```
-------------------------------------------
Select Source Address: boston latin academy
Select Destination Address: museum of science
-------------------------------------------
205 Townsend St, Boston, MA 02121, USA
42.316119
-71.0845953

Museum Of Science Driveway, Boston, MA 02114, USA
42.367714
-71.0709771

Closest hub to source =  Northeastern University, Boston
Closest hub to destination =  North Station, Boston
```



In [44]:
```
route()
```

```
-------------------------------------------
Select Source Address: MIT
Select Destination Address: boston university
-------------------------------------------
77 Massachusetts Ave, Cambridge, MA 02139, USA
42.360091
-71.09416

Boston, MA 02215, USA
42.3504997
-71.1053991

Closest hub to source =  Fenway, Boston
Closest hub to destination =  Boston University, Boston
```



In [42]:
```
route()
```

```
-----------------------------------------
Select Source Address: MIT
Select Destination Address: BU medical campus
-----------------------------------------
77 Massachusetts Ave, Cambridge, MA 02139, USA
42.360091
-71.09416

715 Albany St #437, Boston, MA 02118, USA
42.3355579
-71.0711474

Closest hub to source =  Fenway, Boston
Closest hub to destination =  Northeastern University, Boston
```

In [ ]:
```
```