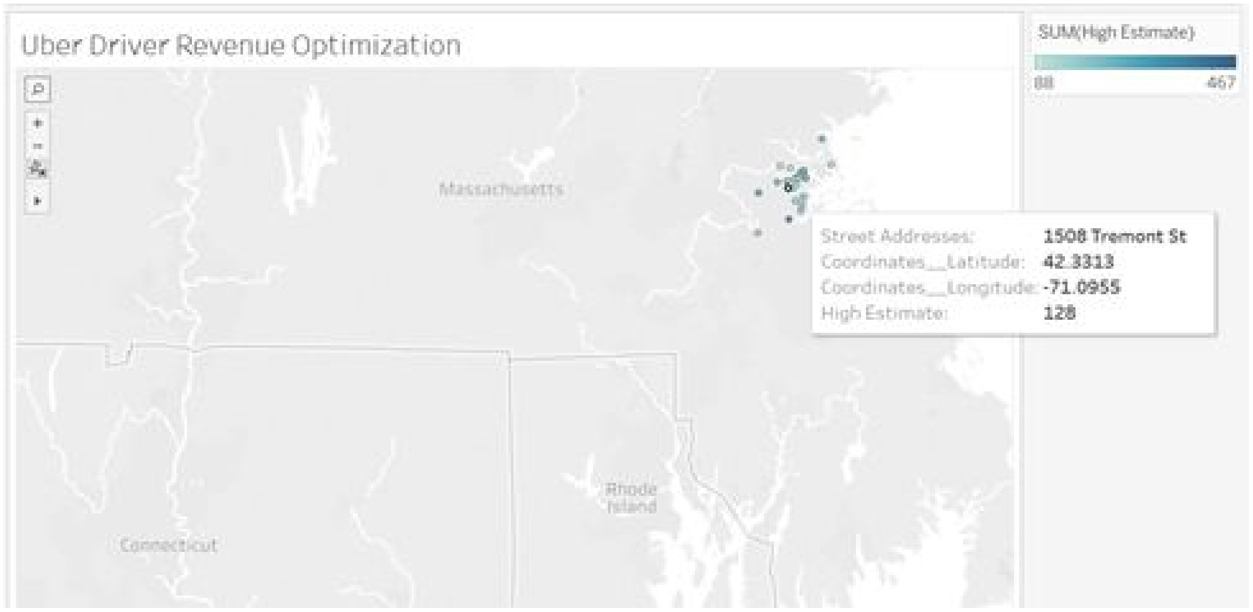
Eugene Cheah | Mahalakshmi Arunachalam
cheah.eug@husky.neu.edu | arunachalam.m@husky.neu.edu
INFO 7390 Advances in Data Sciences and Architecture
Fall 2017, Northeastern University College of Engineering

Uber Data Optimization

Abstract

Being an Uber driver may sound lucrative, but how real is it and can the time utilization be maximized as they also need to pay for expenses for running the car or its insurance. An Uber driver may happen to be driving at a particular location, but there is no guarantee that there are passengers there, if any. Logically speaking, it is better to have accurate predictions for an Uber driver to decide upon his/her own schedule, maximize earnings in a shorter time, and on top of that knowing all the best driving locations where ride-hailing customers are densest. Conducting data analysis, training predictive models, and building an application would come in handy for the drivers, all in which are the aim of the project.


Uber Driver Revenue Optimization
SUM(High Estimate)
88
467
Massachusetts
Rhode Island
Connecticut
Street Addresses: 1508 Tremont St
Coordinates__Latitude: 42.3313
Coordinates__Longitude: -71.0955
High Estimate: 128

Introduction

The purpose of the research would be to build an application to optimize Uber drivers' rides in Boston. With this application Uber drivers should have a higher driving frequency, leading to higher profit margins in less time. Factors and assumptions to take into consideration include but are not limited to customer pickup frequency, time of day, weather, and popular events happening in the city. Through this project, we aim to learn and apply predictive models like time series analysis, as well as machine learning methods such as Support Vector Machines, K-Nearest Neighbors, and Naïve Bayes. Uber drivers only have a finite amount time for driving, so why not make the most out of their driving time?

Datasets Sources

1. Uber API - https://developer.uber.com/dashboard/
2. Uber Rides Python SDK (beta) - https://github.com/uber/rides-python-sdk
3. Yelp API - https://www.yelp.com/developers/documentation/v3/business_search

Selecting Locations

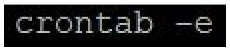
In this study, we focus on the Uber ride in Boston and Cambridge areas. We have restricted our study to these two small geographic regions due to practical issues such as analysis on a smaller region will help us to predict better with the help of other factors such as weather or popularity of places. This would provide us to provide efficient analysis with available resources.

Approach to Obtaining the Uber Dataset

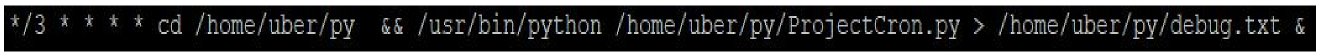
The main programming language used for this project, unless specified otherwise, is Python. As indicated in the Datasets section of this report, we queried the Uber API. In order to do that, we must first register an application on Uber's developer dashboard and install the uber-rides SDK, which is also available on Github. After successfully creating an Uber session with a server token, we found out that the Uber Ride API accepts a pair of coordinates (latitudes and longitudes) as parameters to return data on its various types of services and their price estimates.

Instead of inputting random coordinates in Boston and Cambridge, which are the areas we are parameterizing, we opted to query Yelp's API as well because it returns actual businesses' location and thankfully the returned data contains longitudes and latitudes. A similar process to querying any API, we needed a server token and a client ID to access it. Once that was done, we can specify a term, such as theater or food, location, and a search limit to request businesses and their details. We wrote a Python script that appends the resulting data based on a set of zip codes in Boston and Cambridge and the key term "food" to a JSON file. Subsequently, the script loads the Yelp data onto a SQLite3 database in which we can extract the coordinates and pass them onto the Uber query, providing that those coordinates are at least a straight line mile apart from each other, validated by using an if statement and the GeoPy package, because we imagine one does not simply hail a ride for a distance shorter than a mile. The script continues to append the ride details to an uber.json file along with the added variables start_latitude, start_longitude, end_latitude, end_longitude, time, as well as datetime using the datetime.datetime.now function. With that, we finally have an Uber dataset based in Boston and Cambridge, Massachusetts and the timing of said ride queries. The next step was to automate the process of running the script.

Both of us are using Windows laptops and we all know the Windows command line is not the best because it lacks functionalities when compared to its Apple and Linux counterpart. Hence we decided to employ Amazon Web Services (AWS) and set up an Ubuntu server to perform a Cron Job. The images below are snippets of the commands we used to set up Cron Job that executes our Python script every three minute via crontab, which continuously appends data to and increases the file size of the uber.json file.

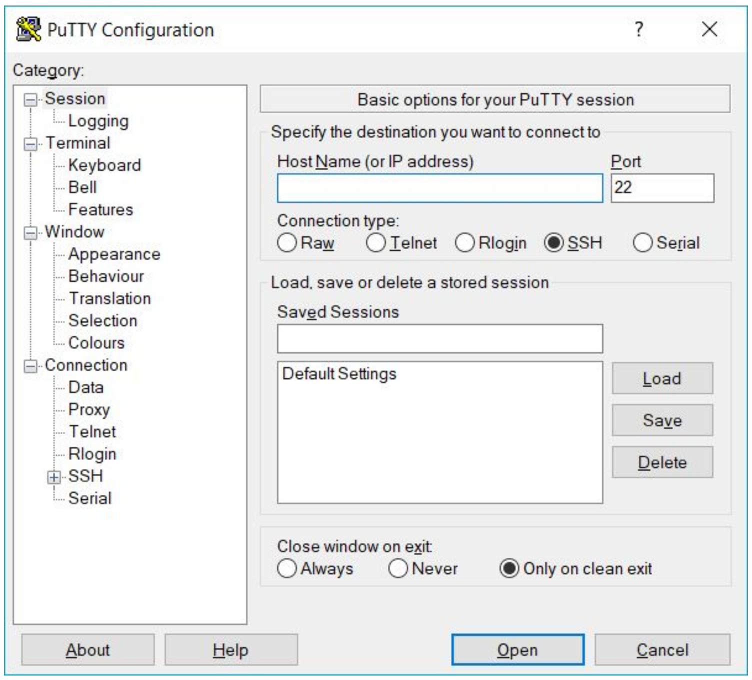


```
crontab -e
```



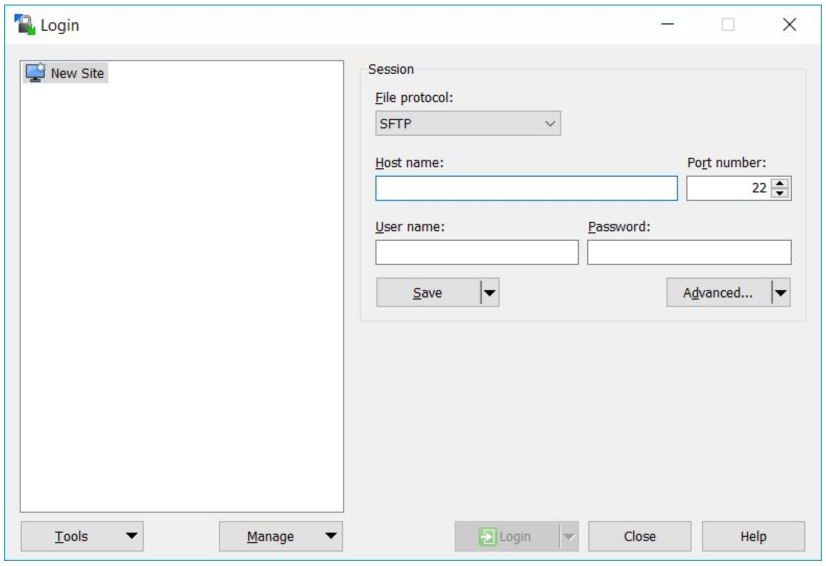
```
*/3 * * * * cd /home/uber/py  && /usr/bin/python /home/uber/py/ProjectCron.py > /home/uber/py/debug.txt &
```

Softwares we installed to access and modify the Ubuntu server are PuTTY and WinSCP. PuTTY is an SSH and telnet client developed for the Windows platform, whereas WinSCP offers a GUI that allows easier secure file transfer between a local and a remote computer/server.



Specify the destination you want to connect to
Host Name (or IP address):
Port: 22
Connection type: Raw Telnet Rlogin [✓ SSH] Serial
Load, save or delete a stored session
Saved Sessions:
Default Settings
Close window on exit: Always Never [✓ Only on clean exit]

A screenshot of PuTTY



A screenshot of WinSCP

Once we have sufficient data on Uber in around Boston and Cambridge, it was time to start conducting our analyses and applying prediction models for the benefit of Uber drivers.

Analysis Basis

We have tried to analysis the uber data collected broadly on two main categories i.e. the high estimate and low estimate for all different types of uber ride services offered by the company namely, uberPOOL, uberX, uberSUV, uberBlack and uberWAV.

Based on the data collected, we are analyzing the high and low cost rides based on time and other factors such as distance, duration, end latitude, end longitude, Weather, day_of_week, end_latitude, end_longitude

All About the Input Dataset

Below are the description of each column in the dataset:

|-- Date-time: string -Date and the time when the request was made
|-- currency_code: string – Currency used for payment(USD)
|-- display_name: string – Type of the ride i.e. uberPOOL, uberX, uberSUV, uberBlack, uberXL, uberWAV and Taxi
|-- distance: double – Distance between the start and end location

|-- duration: long – The time for which the ride lasted
|-- end_latitude: double – End location latitude of particular ride
|-- end_longitude: double – Start location latitude of a particular ride
|-- estimate: string – Price range of the payment to be made for particular ride
|-- high_estimate: double – Estimate of the ride cost on the
|-- localized_display_name: string (nullable = true)
|-- low_estimate: double (nullable = true)
|-- product_id: string (nullable = true)
|-- start_latitude: double - Start location latitude of a particular ride
|-- start_longitude: double - Start location longitude of a particular ride
|-- time: string (nullable = true) – Time at which request was initiated

Time Series Analysis

According to Wikipedia, a time series is a series of data points indexed in time order. Most commonly, a time series is a sequence taken at successive equally spaced points in time. Thus it is a sequence of discrete-time data [1]. When it comes to looking at historical data and predicting the future, time series analysis is the go-to analytical method to employ. However before that, we ought to carry out some exploratory data analysis to better understand our dataset and make changes to our dataframe if needed. The Pandas package allows for easy creation of dataframes and a ton of its user-friendly functionalities include time-related series.

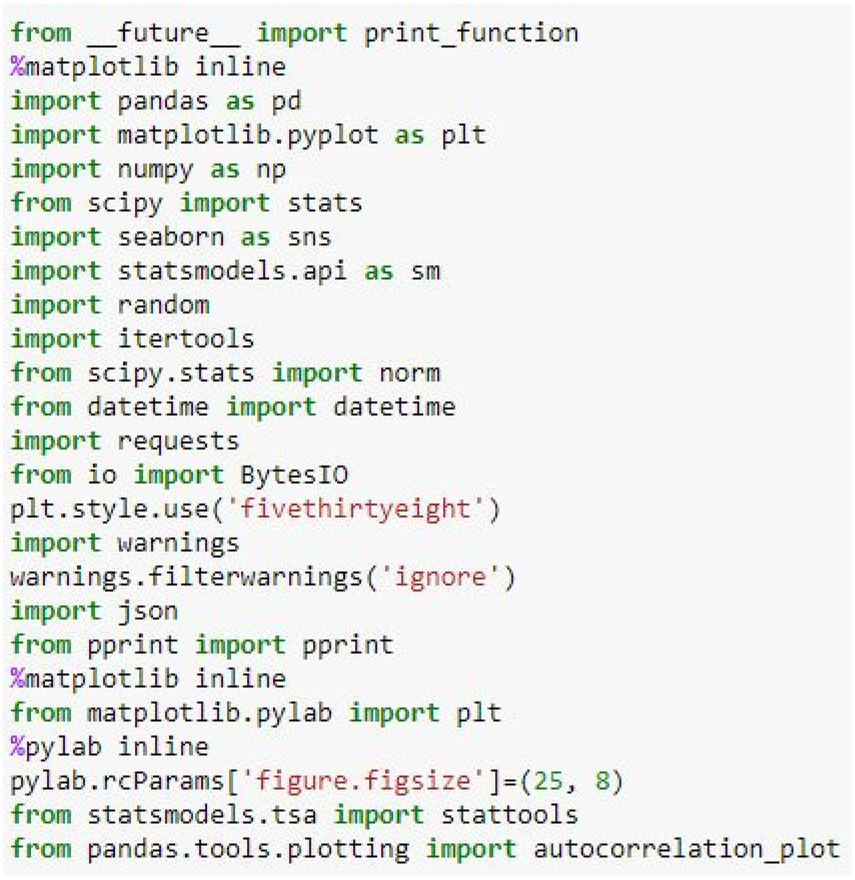
Using Pandas to read the JSON file, we noticed there are seven unique types of Uber rides, namely 'uberPOOL', 'uberX', 'uberSUV', 'uberXL', 'UberBLACK', 'uberWAV', and 'TAXI'. The first step we took was to look for null values and we realized that the null values all lie on the 'TAXI' rows. We proceeded to remove rows with 'TAXI' as they are not related to predicting Uber prices in any way because it is stated on the data itself to be metered just like a regular taxi.

Next, we looked at datatypes of the data columns. We wanted to change the datatype of column 'date_time' from object to datetime and Pandas supports the changing of datatypes with a simple line of code. We then proceeded to display the price estimates’ mean, max, and min by ride types to get a sense of the numbers. After successfully changing the 'date_time' column datatype, the next crucial step is to set the numeric index to a DatetimeIndex based on the date and time on the 'date_time' column. We also rounded the DatetimeIndex to nearest minute to make the data look cleaner.

Once we were done exploring the dataset, we decided to split the six different types of Uber services into their own data frames. Starting with uberPOOL, the following are what we did for the time series analysis:

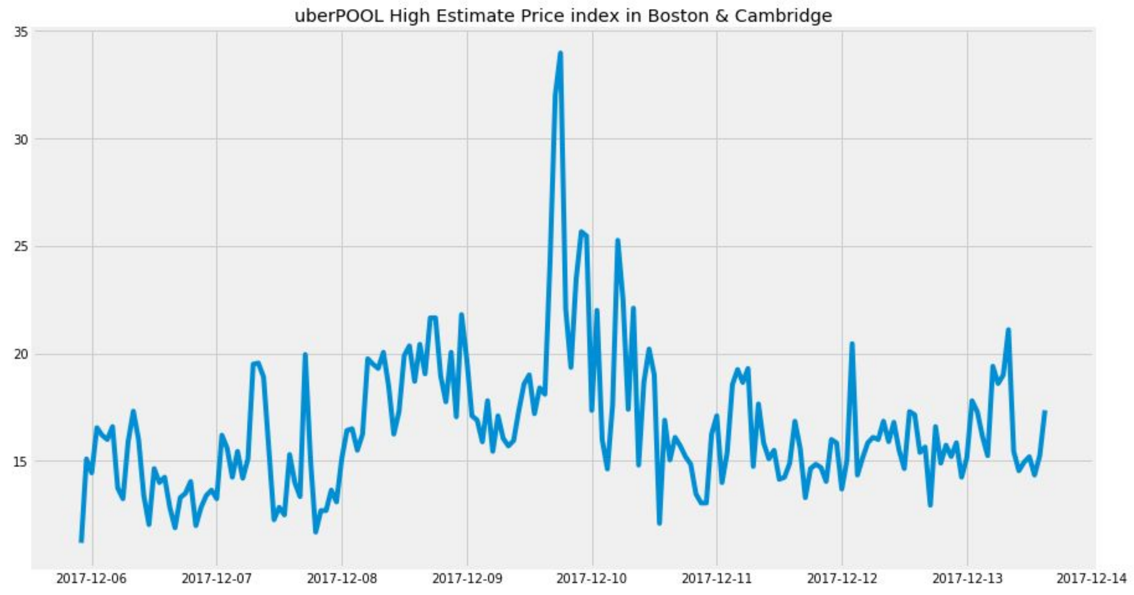
1. List an array of column names with what we want to include to form a new dataframe for the time series analysis, namely 'high_estimate' and 'low_estimate' columns. Note that we do not have to include the 'date_time' column because it was already indexed.
2. Remove duplicated DatetimeIndex by marking duplicates as true except for the first occurrence [2].
3. Plot a preliminary graph based on the new dataframe.
4. Plot an autocorrelation plot for price estimates to check for randomness in the dataframe.
5. Plot the autocorrelation function for price estimates with varying degrees of lag.
6. Resample the DatetimeIndex to an hourly mean of the data using Pandas' resample functionality.
7. Build an Autoregressive Integrated Moving Average ARIMA(p,d,q) Model based on price estimates and subsequently trying the model for different p,d,q orders in a for loop.
8. Apply NumPy's logarithm function, which is the inverse of the exponential function, to the price estimates and create a new column to store the result [3].
9. Get the log difference and create another new column to store the result.
10. Plot two plots in price estimates and difference of logs.
11. Plot a new autocorrelation function and a partial autocorrelation function.
12. Build a Seasonal AutoRegressive Integrated Moving Average with eXogenous regressors model SARIMAX based on price estimate logs with AR model (1,1,1).
13. Specify our endogenous and exogenous data indexer variables for price estimates and difference of price estimate logs respectively.
14. Build a new SARIMAX model based on them with order (1,0,1).
15. Plot a graph on observed price estimates data, one-step-ahead forecast, and dynamic forecast based on in-sample prediction and out-of-sample forecasting, coupled with confidence intervals based on respective T-distributions.
16. Plot a graph matching up in-sample one-step-ahead predictions and 95% confidence intervals as well as dynamic predictions and 95% confidence intervals.
17. Repeat step 1 through 16 for 'uberX', 'uberSUV', 'uberXL', 'UberBLACK', and 'uberWAV'.

List of Python packages used for Time Series Analysis:

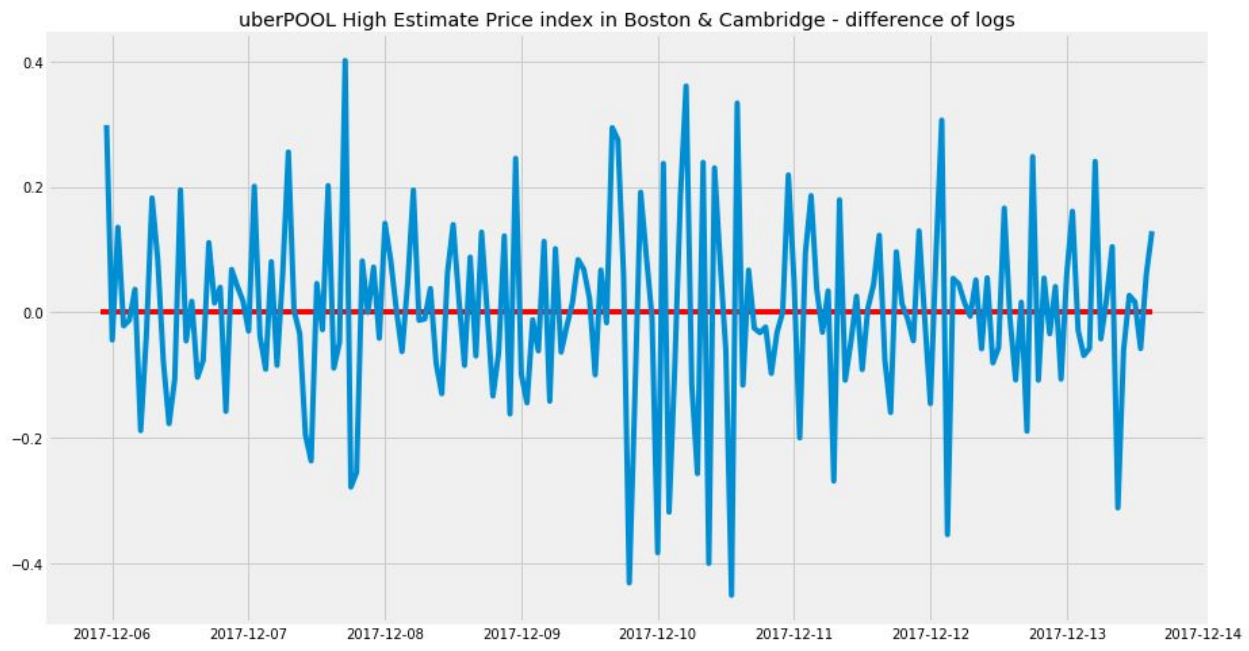


```
from __future__ import print_function
%matplotlib inline
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
from scipy import stats
import seaborn as sns
import statsmodels.api as sm
import random
import itertools
from scipy.stats import norm
from datetime import datetime
import requests
from io import BytesIO
plt.style.use('fivethirtyeight')
import warnings
warnings.filterwarnings('ignore')
import json
from pprint import pprint
%matplotlib inline
from matplotlib.pylab import plt
%pylab inline
pylab.rcParams['figure.figsize']=(25, 8)
from statsmodels.tsa import stattools
from pandas.tools.plotting import autocorrelation_plot
```

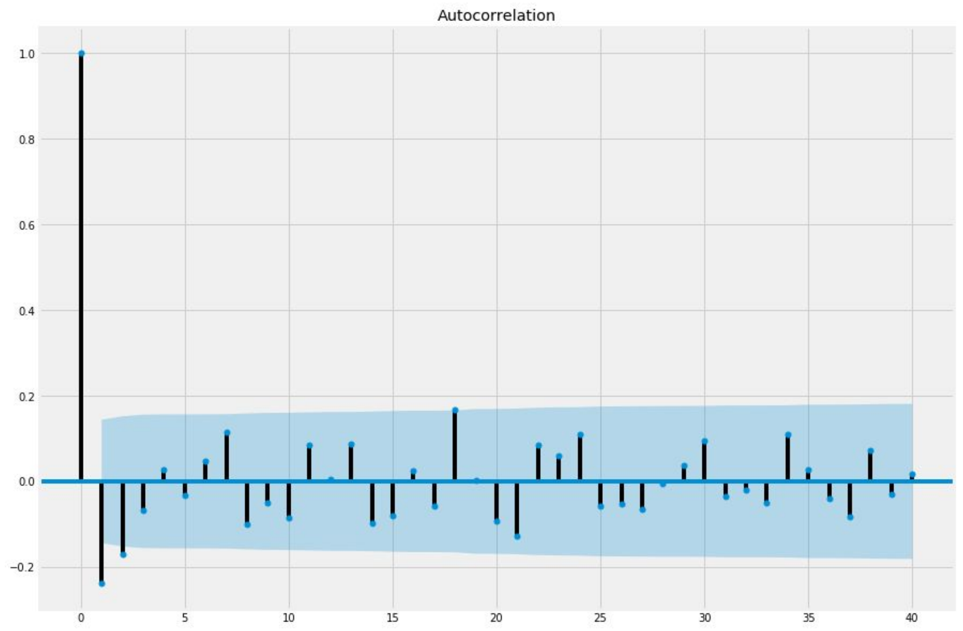
To prevent a bloated version of the report, we will just include images of uberPOOL's high price estimate time-series analysis. The rest of the ride types' analysis as well as low price estimates are similar, though not the same, and can be found in our Jupyter notebook.

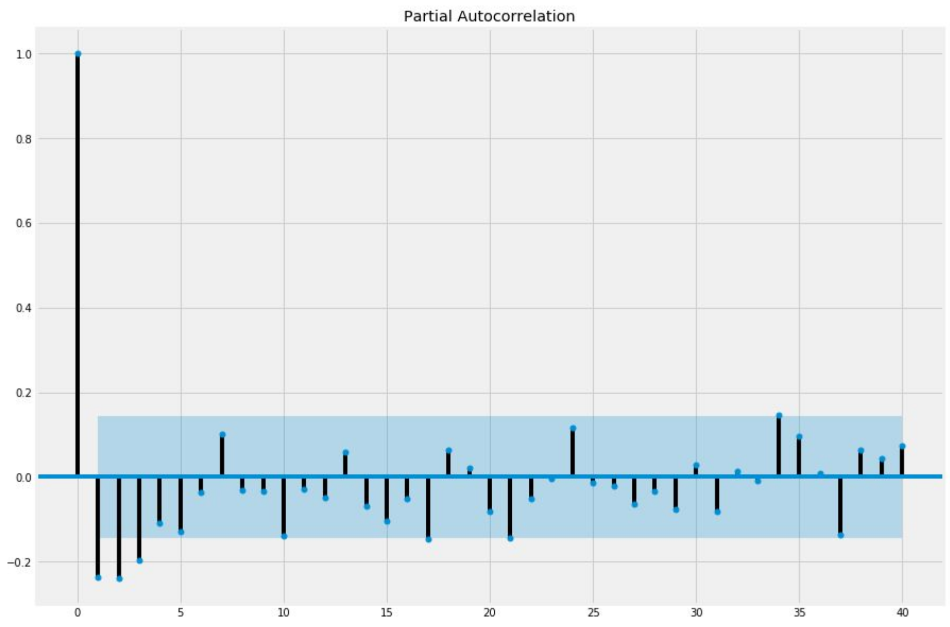


The image above shows a general plot on uberPOOL's high price estimate. All six Uber ride types have a similar trend around Boston and Cambridge where price estimates spiked on 9th of December, because the couple inches of snow of the season fell that day.



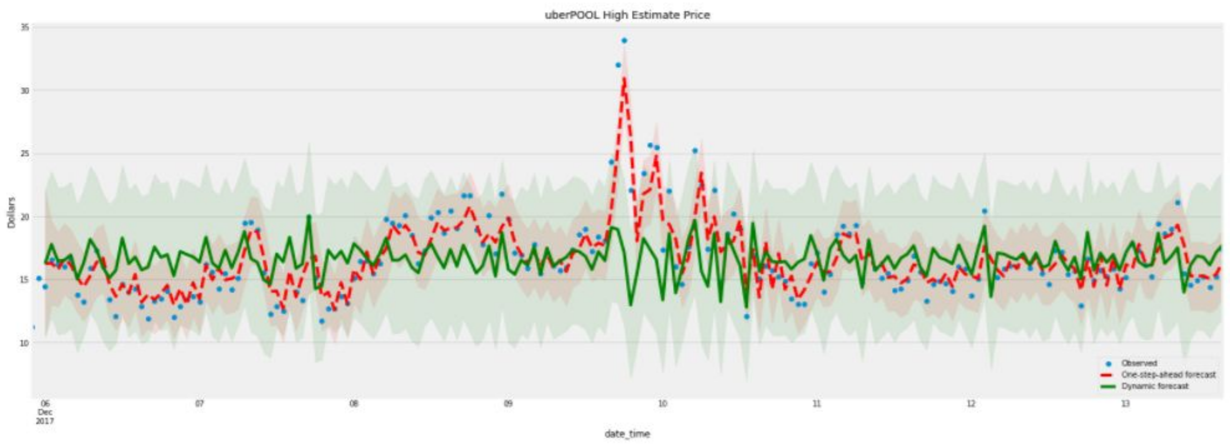
A plot of difference of high price estimate logs.



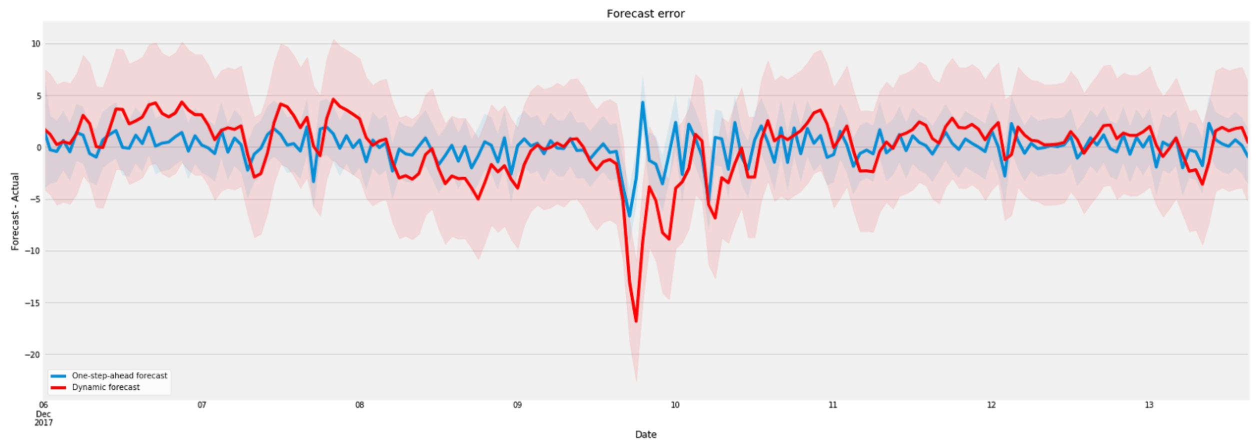


Plots showing autocorrelation and partial autocorrelation function on high price estimates with a lag of 40. They indicate a higher order autoregressive term in the data and thus an AR(1 model) would be feasible [4].

Conclusion



The plot shows observed price estimates data, one-step-ahead forecast, and dynamic forecast. The one-step ahead forecast works better than the dynamic forecast, which makes sense since Uber constantly updates its price estimates by the minute. In fact, the one-step ahead forecast follows the observed data very closely.



The plot shows a graph matching up forecast errors between one-step-ahead forecast and dynamic forecast. Dynamic forecast generally did poorly across all six ride types when the actual data has a sudden spike on the snowy day of December, whereas one-step-ahead forecast manages to forecast with less discrepancy.

Overview of Supervised Learning Methods Used

The methods that were followed are listed below:

1) Support Vector Machines
2) K- Nearest Neighbors
3) Naïve Bayes

Support Vector Machines

It is based on the idea of a hyperplane or a decision plane dividing variables having the different properties and grouping together similar trends.

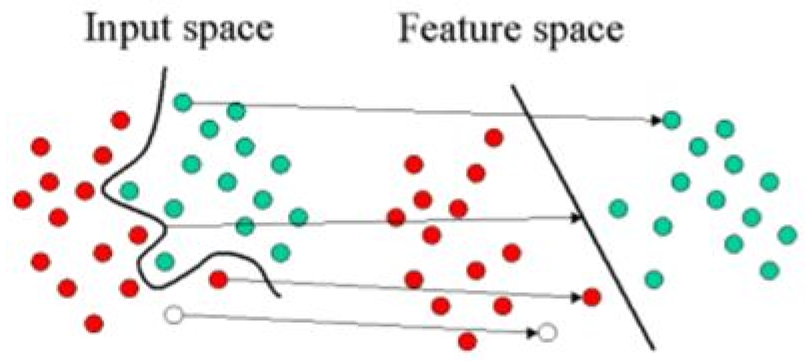


fig: Support vector machines concept

As the figure above depicts how the algorithm works, i.e. the objects having same properties i.e color are grouped together in the feature space. SVM is classified on the basis of the kernels used. Kernels are the similarity functions used to point out the similarity between the input variables considered.

Kernel functions used are the dot product of input data points projected on the output or classification plane.

Kernel used here for analysis is the Radial basis function as it has finite responses for wide range if input variations.

RBF kernel on two samples x and x' is defined as:

$$K(\mathbf{x}, \mathbf{x'}) = \exp\left(-\frac{||\mathbf{x} - \mathbf{x'}||^2}{2\sigma^2}\right)$$

Where ||x-x'||2 is the Euclidean distance between the sample points and generally, = 1/22.

<u>K- Nearest Neighbor</u>

KNN is a non-parametric method used for classification.

The similarity is based on the measured by the closest distance the object is from its neighbours and generally, it is calculated using the Euclidean distance.

The optimal value of k used here is 1, it means that the object is assigned to the classification type of that single nearest neighbor. The predicted output contains the properties of all the inputs considered.

It is a lazy algorithm as it predicts output based on the training data.

<u>Naïve Bayes Algorithm</u>

$$\underset{\text{Posterior Probability}}{P(c\mid x)} = \frac{\overset{\text{Likelihood}}{P(x\mid c)}\,\overset{\text{Class Prior Probability}}{P(c)}}{\underset{\text{Predictor Prior Probability}}{P(x)}}$$

$$P(c\mid X) = P(x_1\mid c)\times P(x_2\mid c)\times\cdots\times P(x_n\mid c)\times P(c)$$

Above,

- *P(c|x)* is the posterior probability of *class* (c, *target*) given *predictor* (x, *attributes*).
- *P(c)* is the prior probability of *class*.
- *P(x|c)* is the likelihood which is the probability of *predictor* given *class*.
- *P(x)* is the prior probability of *predictor*.

Naïve Bayes is a probabilistic prediction algorithm. It predicts the output based on the number of occurrences. It assumes that the input variables are independent of each other.

Two types used here are listed below:
[i] Gaussian naive Bayes – The basic assumption made here is that each class of objects considered for prediction is distributed normally.
[ii] Multinomial Naïve Bayes – It calculates probability based on the frequency of occurrence or counts.

The approach followed:

1. Read the uber response json file with the help of dataframes in python.
2. Based on the date, the weather information was gathered and was appended into the dataframe.
3. Convert Date-Time to Date and then to Day of the week to measure the ride frequency on a particular week day.
4. Plot graphs to analyze the data obtained.
5. Remove the columns which are least significant and redundant in further predictions. As per our analysis, the redundant columns removed were namely, 'currency_code', 'estimate', 'localized_display_name', 'product_id', 'time', 'date', 'start_latitude', 'start_longitude'.
6. Now, convert the Weekday categorical data into indices by assigning them a integer values so that it could be used in further prediction.

7. Also, convert the four categories of weather into indices by assigning them appropriate values using the 'date' column
8. Now, remove the 'date' column as it does not further aid in analysis.
9. Create a new dataframe to contain only a particular ride type for better prediction of results.
10. Remove any duplicates in the above dataframe.
11. Calculate the mean, variance and standard deviation of the high and low price estimates.
12. Now map the prices which are above mean and below mean for both high and low estimates. Terminologies used:
    - HH: High estimate above mean high estimate price
    - HL : High estimate below mean high estimate price
    - LH : Low estimate above mean low estimate price
    - LL : Low estimate below mean low estimate price
13. Plot box plot to check for outliers and scale the prediction variables to be used.
14. The predictors considered are:
    - distance', 'duration', 'low_estimate', 'Weather', 'day_of_week', 'end_latitude', 'end_longitude' for high estimates.
    - 'distance', 'duration', 'high_estimate', 'Weather', 'day_of_week', 'end_latitude', 'end_longitude' for low estimate prediction
15. Divide the dataset into train and test sets.
16. Apply the machine learning algorithm i.e. SVM, K-NN, or Gaussian Naïve Bayes and Multimodal Naïve Bayes
17. Analyze results based on accuracy, results from confusion matrix and classification reports.
18. Repeat from step 8-17 for each type of uber ride offered for both high estimate and low estimate price prediction.

(Note, in this project all steps were repeated as they were done on different Jupyter notebooks but it is not necessary)

Result Analysis

(A) For High estimate prediction

Comparison for	**High Estimate Models**			
Model Accuracy / Ride Type	**RBF SVM**	**KNN**	**MultinomialNB**	**GaussianNB**
uberPOOL	0.801912568	0.803961749	0.739754098	0.899590164

uberX	0.799863388	0.831284153	0.764344262	0.93920765
uberSUV	0.862704918	0.871584699	0.780054645	0.984972678
uberXL	0.834016393	0.836748634	0.784836066	0.950136612
uberBlack	0.864754098	0.864071038	0.786202186	0.974043716
uberWAV	0.81010929	0.828551913	0.758196721	0.928278689

(B) For low estimate prediction:

Comparison for	**Low Estimation Models**			
Model Accuracy / Ride Type	**RBF SVM**	**KNN**	**MultinomialNB**	**GaussianNB**
uberPOOL	0.792349727	0.800546448	0.728825137	0.901639344
uberX	0.800546448	0.824453552	0.760245902	0.932377049
uberSUV	0.858606557	0.866803279	0.782786885	0.980191257
uberXL	0.862021858	0.851775956	0.839480874	0.942622951
uberBlack	0.866803279	0.866803279	0.783469945	0.975409836
uberWAV	0.800546448	0.824453552	0.760245902	0.932377049

Based on the values, the Gaussian Naïve bayes is the best model amongst the four models compared in the project. Then, K – nearest neighbors is able to predict best, followed by RBF SVM and then Multinomial Naïve Bayes.

<u>List of qualifiers used for best method prediction in confusion matrix:</u>

- Accuracy: Overall, how often is the classifier correct?
  - (TP+TN)/total
- Misclassification Rate: Overall, how often is it wrong?
  - (FP+FN)/total
  - equivalent to 1 minus Accuracy
  - also known as "Error Rate"
- True Positive Rate: When it's actually yes, how often does it predict yes?
  - TP/actual yes
  - also known as "Sensitivity" or "Recall"
- False Positive Rate: When it's actually no, how often does it predict yes?

    - FP/actual no
- Specificity: When it's actually no, how often does it predict no?
    - TN/actual no
    - equivalent to 1 minus False Positive Rate
- Precision: When it predicts yes, how often is it correct?
    - TP/predicted yes
- Prevalence: How often does the yes condition actually occur in our sample?
    - actual yes/total

<u>List of qualifiers used for best method prediction in classification report:</u>

- The precision is the ratio tp / (tp + fp) where tp is the number of true positives and fp the number of false positives. The precision is intuitively the ability of the classifier not to label as positive a sample that is negative.
- The recall is the ratio tp / (tp + fn) where tp is the number of true positives and fn the number of false negatives. The recall is intuitively the ability of the classifier to find all the positive samples.
- The F-beta score can be interpreted as a weighted harmonic mean of the precision and recall, where an F-beta score reaches its best value at 1 and worst score at 0..
- The support is the number of occurrences of each class in which is truth (correct) target values.

<u>Features of the Trend Observed</u>

The sole purpose of the prediction was to answer the question, can the most effective ride location and time be predicted on a given day of the year. So, analysis of the data was done to predict the trend of Uber users in Boston and Cambridge. So, in the models used, time, day and weather have been used to analyze the high estimate or low estimate of a particular type of Uber ride. Also, only the end latitude and longitude were used as the cost of the ride will depend on the distance from destination and its location and is independent of the start location. As discussed earlier, the duration of the ride also does play a important part and seen linearly affecting the cost of the ride. SO the driver should look for a longer ride.

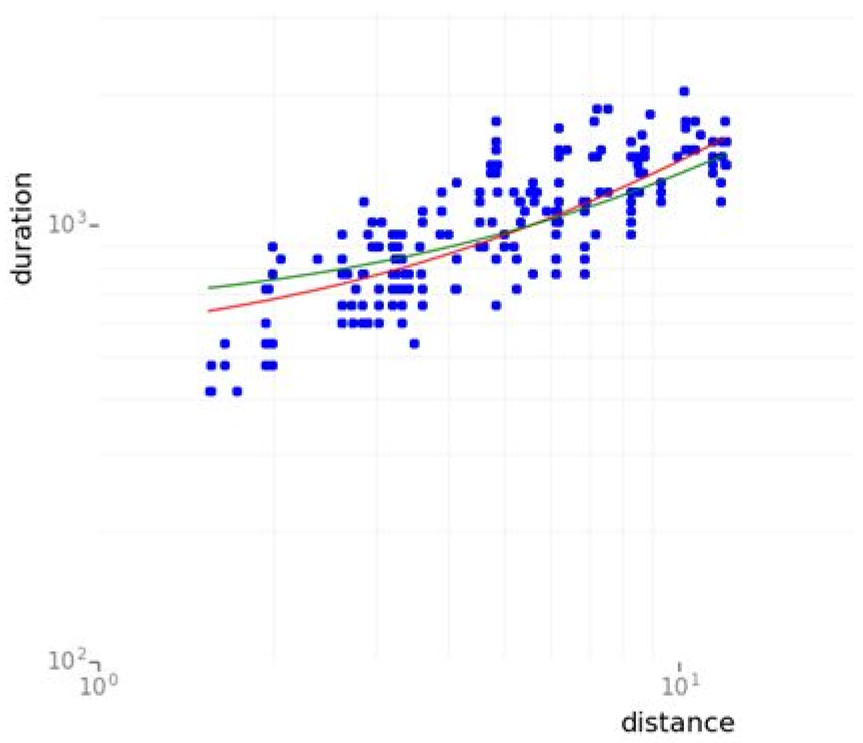


Fig: Duration v/s distance graph using linear regression.

Mid week has the highest low - price and high price estimate of all So uber drivers should be looking for rides on Wednesday with slight profits on Tuesday, Saturday and Sunday.

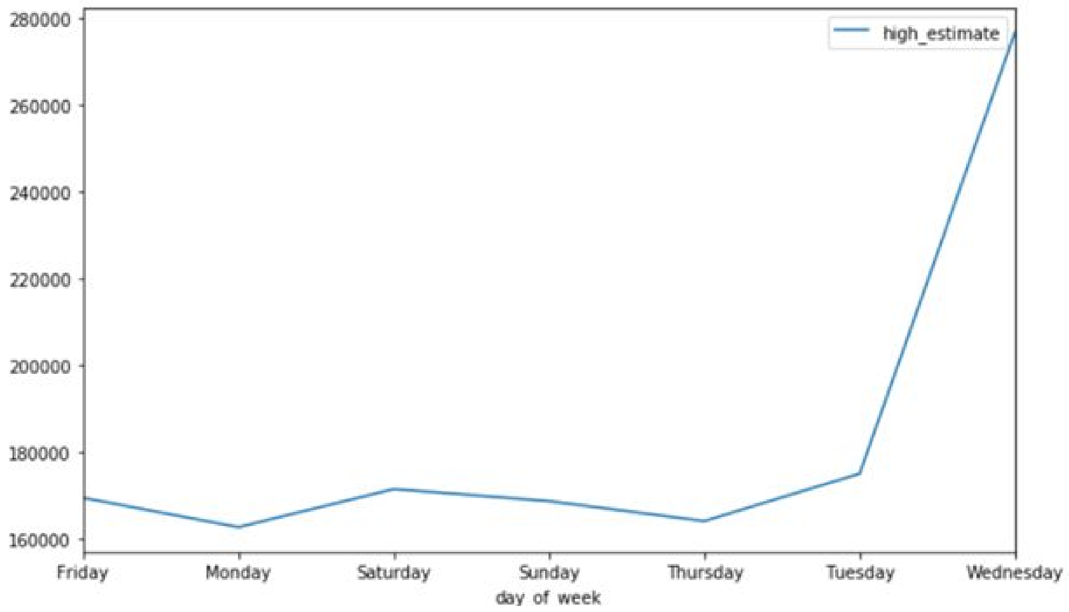


Fig: Plot of sum of high estimates on a weekday

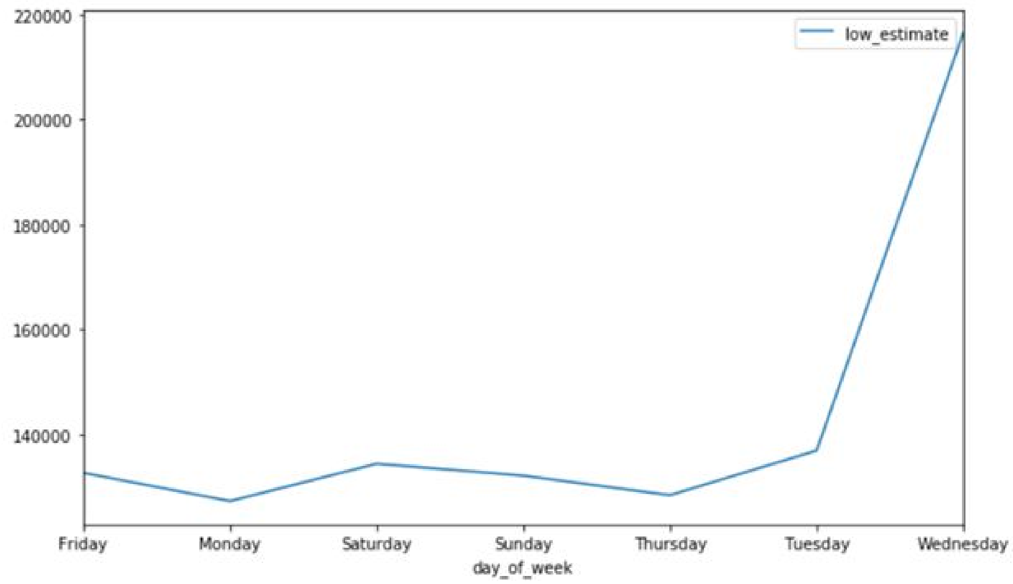


Fig: Plot of sum of low estimates on Weekday

The price estimates on snowy and rainy days are lesser compared to Sunny and Cloudy days. So uber drivers can look out for more rides on Cloudy days and then, sunny days but can be casual on Rainy and then Snowy days.

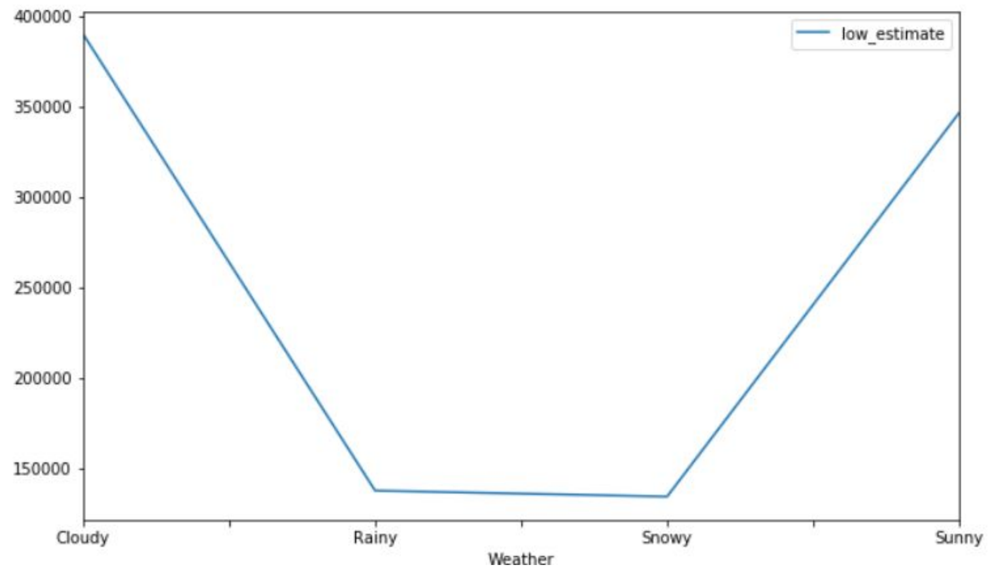


Fig: Plot of sum of low estimates based on weather

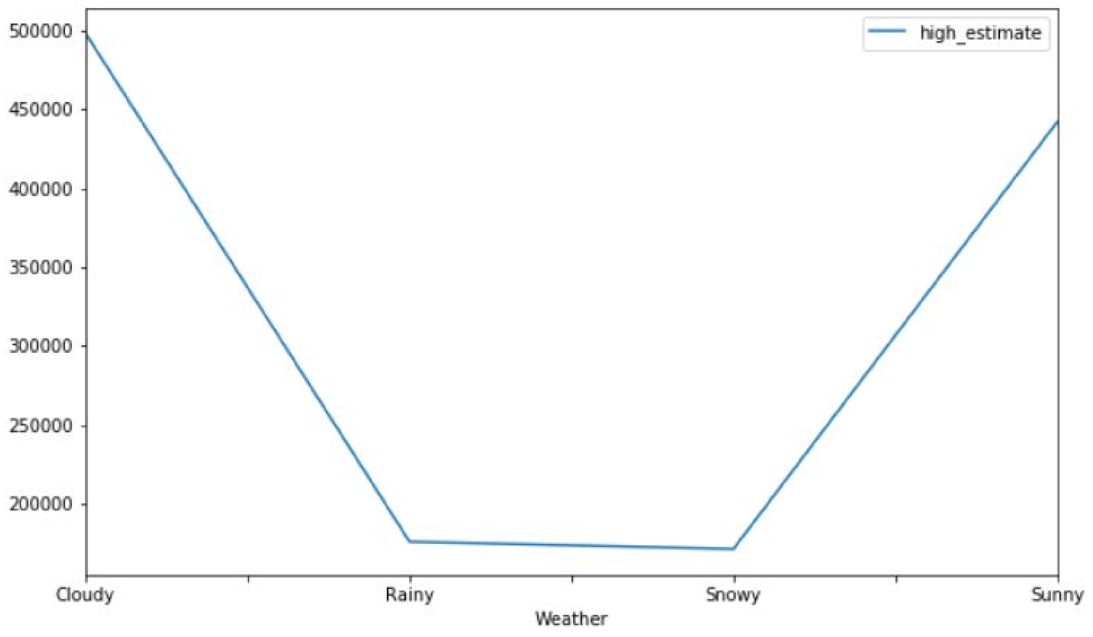
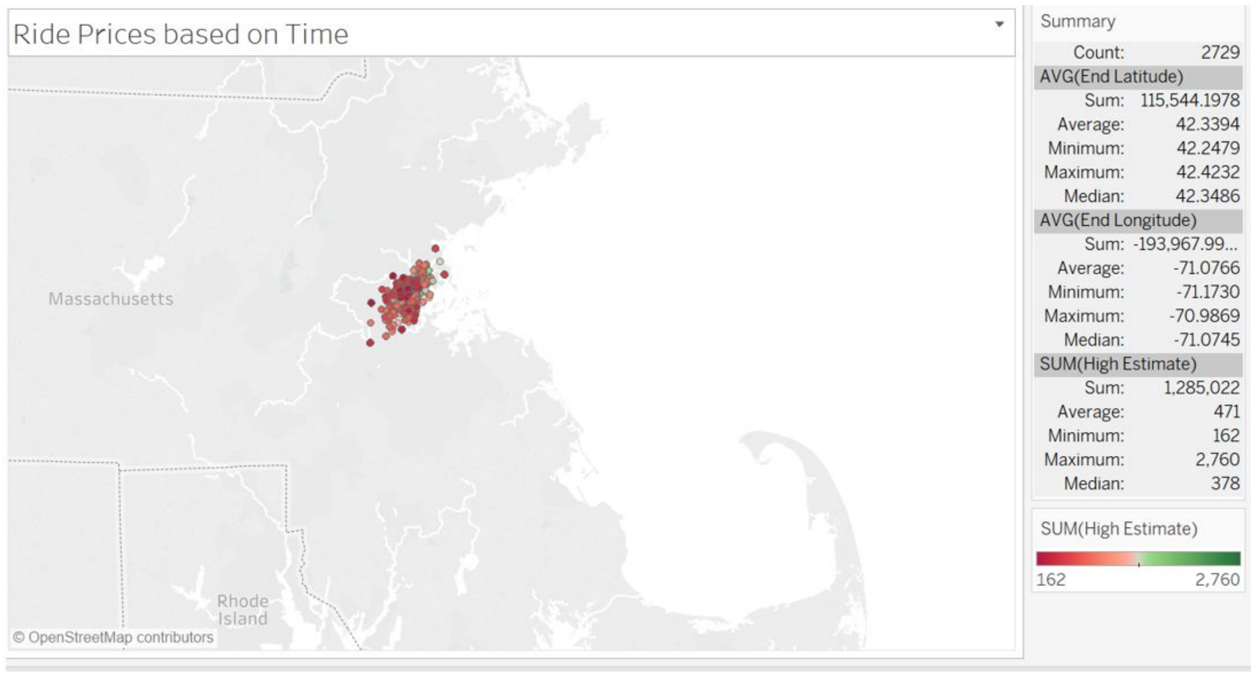


Fig: Plot of sum of high estimates based on weather

<u>Other trends that were seen:</u>



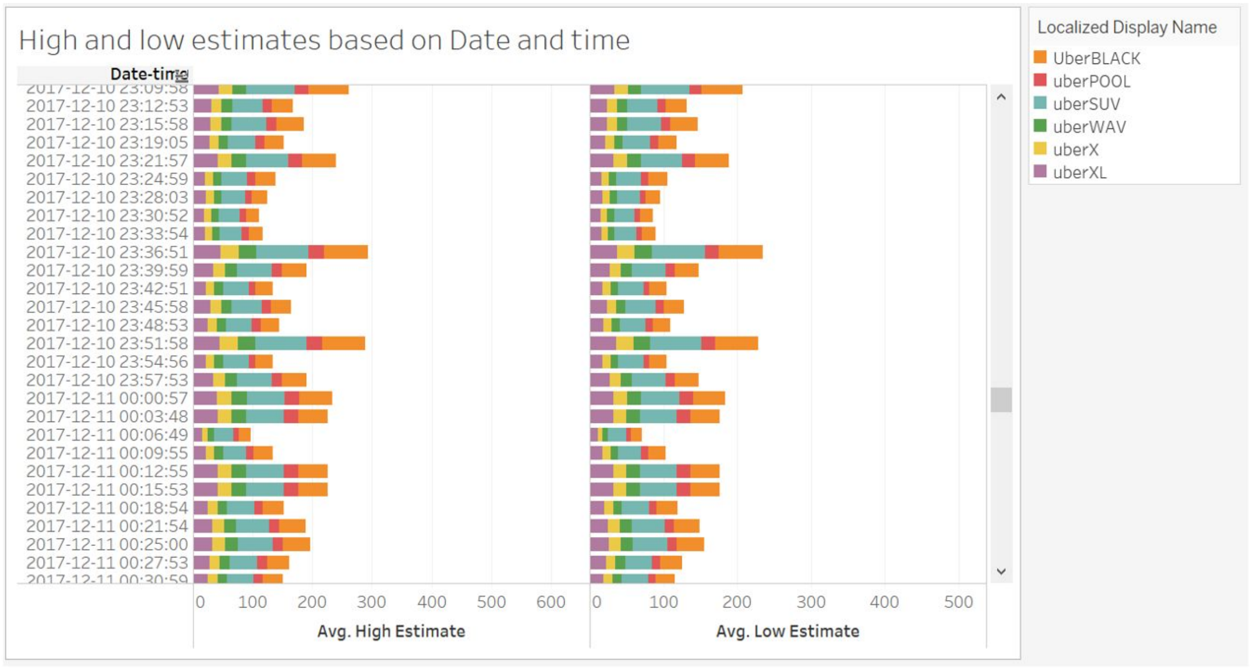
Summary

Count:	2729
AVG(End Latitude)	
Sum:	115,544.1978
Average:	42.3394
Minimum:	42.2479
Maximum:	42.4232
Median:	42.3486
AVG(End Longitude)	
Sum:	-193,967.99...
Average:	-71.0766
Minimum:	-71.1730
Maximum:	-70.9869
Median:	-71.0745
SUM(High Estimate)	
Sum:	1,285,022
Average:	471
Minimum:	162
Maximum:	2,760
Median:	378

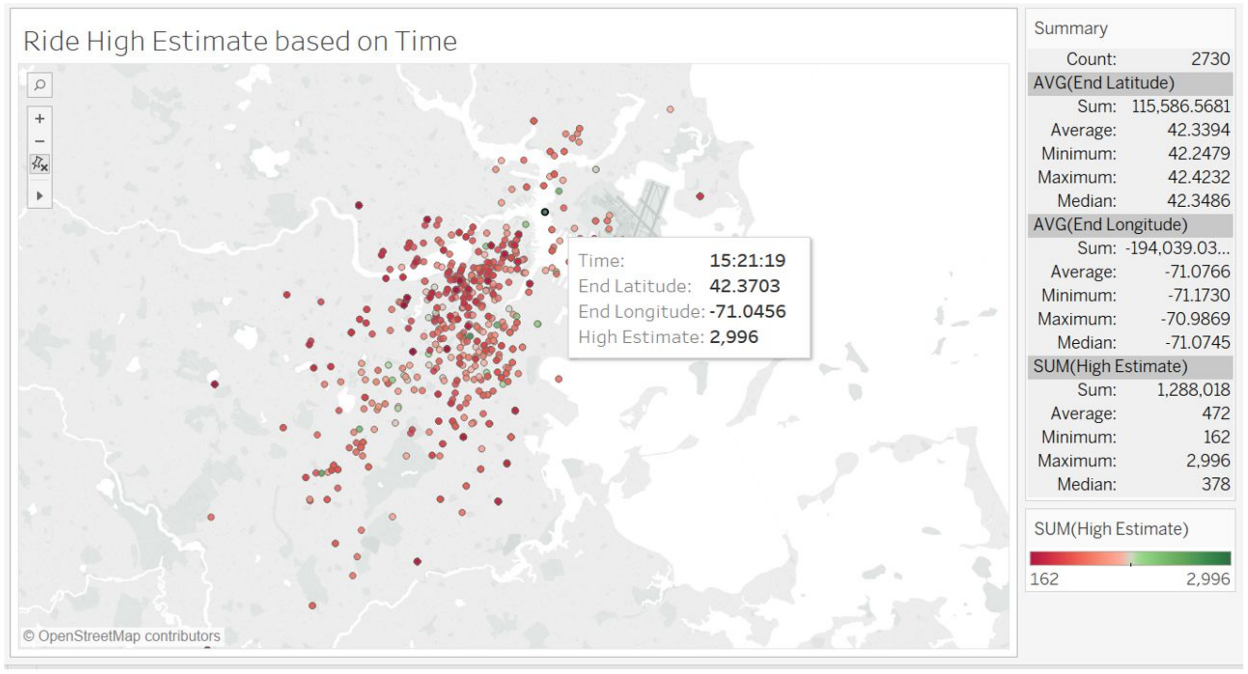
SUM(High Estimate)

162 2,760

As depicted in the figure below, it is more profitable to be a uberSUV and uberBLACK driver, as there average estimates are always greater than other rides.



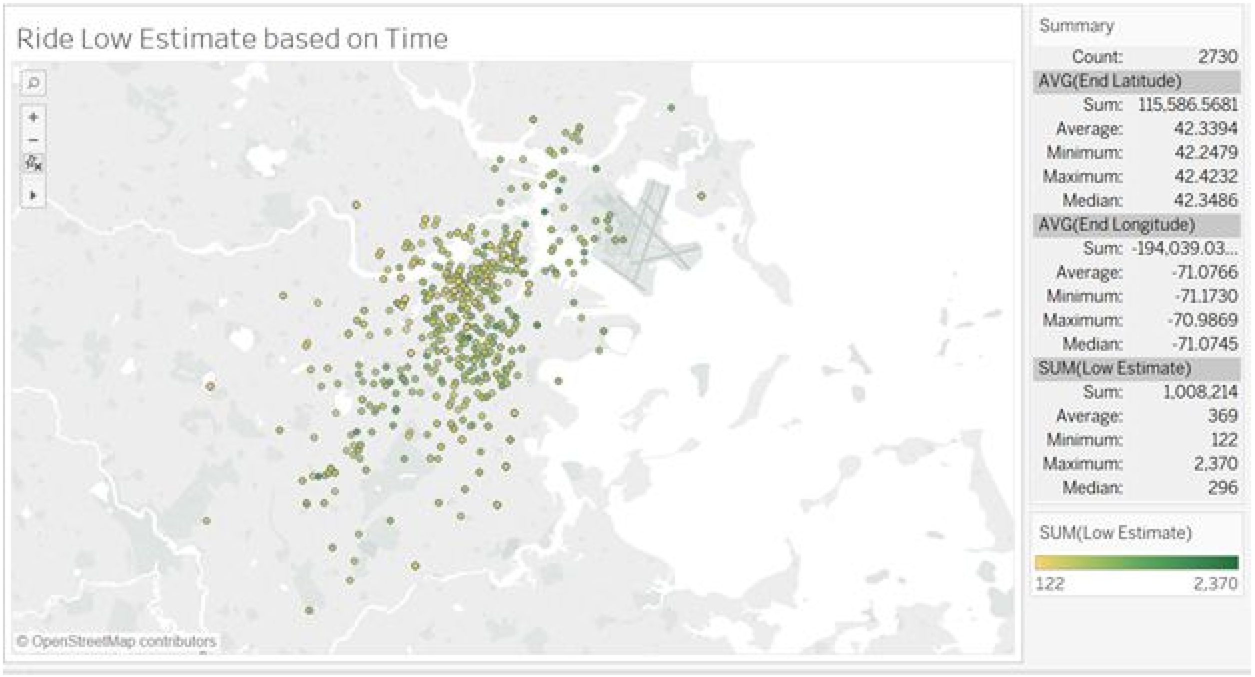
Again, the figure below shows that early morning and midnight rides were more profitable.



Summary

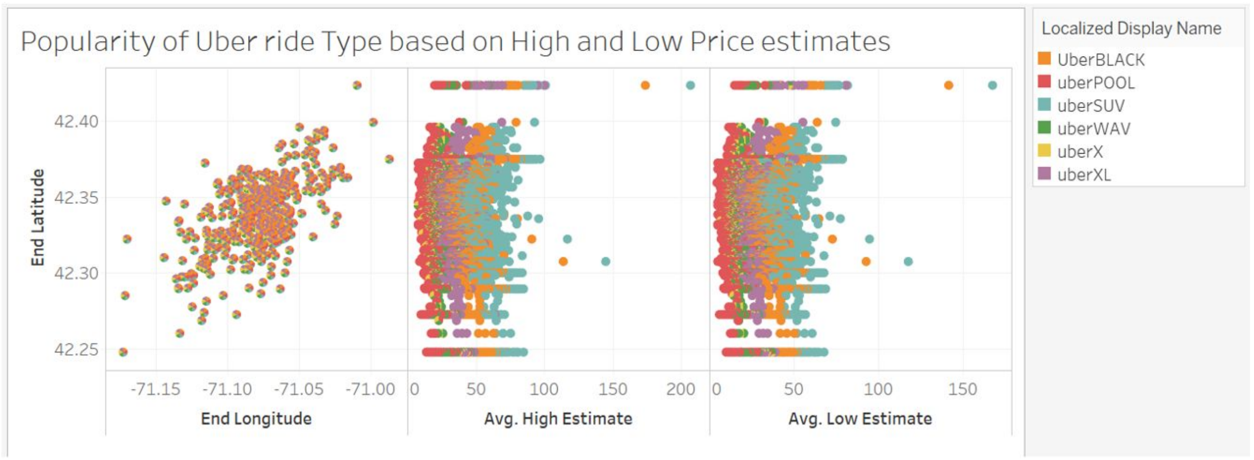
Count:	2730
AVG(End Latitude)	
Sum:	115,586.5681
Average:	42.3394
Minimum:	42.2479
Maximum:	42.4232
Median:	42.3486
AVG(End Longitude)	
Sum:	-194,039.03...
Average:	-71.0766
Minimum:	-71.1730
Maximum:	-70.9869
Median:	-71.0745
SUM(High Estimate)	
Sum:	1,288,018
Average:	472
Minimum:	162
Maximum:	2,996
Median:	378

The figure below shows that low price estimate follow the same trend as the high price trend i.e. late night and early morning ferries are more profitable.



Summary	
Count:	2730
AVG(End Latitude)	
Sum:	115,586.5681
Average:	42.3394
Minimum:	42.2479
Maximum:	42.4232
Median:	42.3486
AVG(End Longitude)	
Sum:	-194,039.03...
Average:	-71.0766
Minimum:	-71.1730
Maximum:	-70.9869
Median:	-71.0745
SUM(Low Estimate)	
Sum:	1,008,214
Average:	369
Minimum:	122
Maximum:	2,370
Median:	296

Also, the popularity of the uber ride could be guessed. UberBLACK is the most popular ride but with profits lesser than uberSUV which has high returns in price estimate but is not popular.



Acknowledgment

A special thank you to Professor Nik Bear Brown for providing guidance and thoughtful insights throughout the course of the project.

Reference

[1] Time series - Wikipedia
https://en.wikipedia.org/wiki/Time_series

[2] pandas.Index.duplicated — pandas 0.21.1 documentation
https://pandas.pydata.org/pandas-docs/stable/generated/pandas.Index.duplicated.html

[3] numpy.log — NumPy v1.13 Manual - Numpy and Scipy Documentation
https://docs.scipy.org/doc/numpy-1.13.0/reference/generated/numpy.log.html

[4] Autocorrelation function (ACF) - Minitab
https://support.minitab.com/en-us/minitab/18/help-and-how-to/modeling-statistics/time-series/how-to/autocorrelation/interpret-the-results/autocorrelation-function-acf/

[5] http://www.statsoft.com/Textbook/Support-Vector-Machines

[6]
https://saravananthirumuruganathan.wordpress.com/2010/05/17/a-detailed-introduction-to-k-nearest-neighbor-knn-algorithm

[7] https://www.analyticsvidhya.com

[8] http://www.statsoft.com/textbook

[9] http://www.dataschool.io