TEQUED LABS INTERNSHIP PROGRAM AUG 2021

PROJECT REPORT ON:

UBER PICKUPS DATASET ANALYSIS

Done By:

Name: YASH B. JOSHI

Internship ID: TLS21A014

E-mail: ymbj28112000@gmail.com

ABSTRACT:

This report explains the working of an Uber dataset, which contains data produced by Uber for New York City. Uber is defined as a P2P platform. The platform links you to drivers who can take you to your destination. The dataset includes primary data on Uber pickups with details including the date, time of the ride as well as longitude-latitude information , Using the information, the paper explains the use of the k-means clustering algorithm on the set of data and classify the various parts of New York City. Since the industry is booming and expected to grow shortly. Effective taxi dispatching will facilitate each driver and passenger to reduce the wait time to seek out one another. The model is employed to predict the demand on points of the city.

INTRODUCTION:

The Uber platform connects you with drivers who can take you to your destination or location. This dataset includes primary data on Uber collections with details that include the date, time of travel, as well as information on longitude and latitude in San Francisco and has operations in over 900 metropolitan areas worldwide.

The prediction of the frequency of trips of data is by implementing a part of k-means clustering algorithm the standard algorithm describes the maximum variance within the group as the number of square distances Euclidean distances between the points and the corresponding centroid. The use of the digital computer has since moved to technology where the program involves the use of neural networks, Examples of RNN (Recurrent Neural Network) and TDNN (Time delay Neural Network) for importing data from uber dataset which takes the data for forecasting on a time horizon.

The ultimate aim of the project is to predict the pickup of the cab on the basis of clusters defined by the k-means clustering algorithm. This algorithm is used to divide the dataset into k-groups. Where k is defined as the number of groups provided by the user. The standard algorithm describes the maximum variance within the group as the number of square distances Euclidean distances between the points and the corresponding centroid. The important packages used in the project are Pandas, numpy, seaborn, k-means, yellowbrick and folium.

TASKS:

- ✓ Data Acquisition and cleaning
- ✓ Data Visualization
- ✓ Data Modelling
- ✓ Testing
- ✓ Comparison and Measurement

DATA ACQUISITION AND CLEANING:

A huge amount of trip data will be collected from Uber for training and testing data. From the collected dataset the latitude and latitude will be clustered and classified based on the frequency of trips travelled by the cab during the day. When these criteria are considered, and data preprocess will be done on these datasets.

IMPORTING LIBRARIES:

```
import pandas as pd
import collections
import itertools
import os

# data visualization
import matplotlib as mpl
import matplotlib.pyplot as plt
import seaborn as sns

from plotly.offline import download_plotlyjs, init_notebook_mode, plot, iplot
import plotly as py
import plotly.graph_objs as go

import scipy.stats as stats
from scipy.stats import norm
from scipy.special import boxcox1p
from sklearn import neighbors
from sklearn.metrics import confusion_matrix, classification_report, precision_score
from sklearn.model_selection import train_test_split

# machine learning
from sklearn.preprocessing import StandardScaler

import sklearn.linear_model as skl_lm
from sklearn.linear_model import LinearRegression
from sklearn import preprocessing

import statsmodels
import statsmodels.api as sm
from statsmodels.tsa.arima_model import ARMA
from statsmodels.tsa.stattools import adfuller
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
from statsmodels.tsa.arima_process import ArmaProcess
from statsmodels.tsa.arima_model import ARIMA

import warnings
warnings.filterwarnings("ignore", category=FutureWarning)
```

RAW DATASET:

	A	B	C	D
1	Date/Tim	Lat	Lon	Base
2	#########	40.769	-73.9549	B02512
3	#########	40.7267	-74.0345	B02512
4	#########	40.7316	-73.9873	B02512
5	#########	40.7588	-73.9776	B02512
6	#########	40.7594	-73.9722	B02512
7	#########	40.7383	-74.0403	B02512
8	#########	40.7223	-73.9887	B02512
9	#########	40.762	-73.979	B02512
10	#########	40.7524	-73.996	B02512
11	#########	40.7575	-73.9846	B02512
12	#########	40.7256	-73.9869	B02512
13	#########	40.7591	-73.9684	B02512
14	#########	40.7271	-73.9803	B02512
15	#########	40.6463	-73.7896	B02512
16	#########	40.7564	-73.9167	B02512
17	#########	40.7666	-73.9531	B02512
18	#########	40.758	-73.9761	B02512
19	#########	40.7238	-73.9821	B02512
20	#########	40.7531	-74.0039	B02512
21	#########	40.7389	-74.0393	B02512
22	#########	40.7619	-73.9715	B02512
23	#########	40.753	-74.0042	B02512

PROCESSED AND CLEANED DATA:

DATA READING AND PREPROCESSING

```
df = pd.read_csv('./data/uber-raw-data-apr14.csv', parse_dates=['Date/Time'])
df = pd.concat([df,pd.read_csv('./data/uber-raw-data-may14.csv', parse_dates=['Date/Time'])], axis=0)
df = pd.concat([df,pd.read_csv('./data/uber-raw-data-jun14.csv', parse_dates=['Date/Time'])], axis=0)
df = pd.concat([df,pd.read_csv('./data/uber-raw-data-jul14.csv', parse_dates=['Date/Time'])], axis=0)
df = pd.concat([df,pd.read_csv('./data/uber-raw-data-aug14.csv', parse_dates=['Date/Time'])], axis=0)
df = pd.concat([df,pd.read_csv('./data/uber-raw-data-sep14.csv', parse_dates=['Date/Time'])], axis=0)
print(df)
```

```
                  Date/Time      Lat      Lon    Base
0       2014-04-01 00:11:00  40.7690 -73.9549  B02512
1       2014-04-01 00:17:00  40.7267 -74.0345  B02512
2       2014-04-01 00:21:00  40.7316 -73.9873  B02512
3       2014-04-01 00:28:00  40.7588 -73.9776  B02512
4       2014-04-01 00:33:00  40.7594 -73.9722  B02512
...                     ...      ...      ...     ...
1028131 2014-09-30 22:57:00  40.7668 -73.9845  B02764
1028132 2014-09-30 22:57:00  40.6911 -74.1773  B02764
1028133 2014-09-30 22:58:00  40.8519 -73.9319  B02764
1028134 2014-09-30 22:58:00  40.7081 -74.0066  B02764
1028135 2014-09-30 22:58:00  40.7140 -73.9496  B02764

[4534327 rows x 4 columns]
```

PROCESSED AND CLEANED DATA:

```
df['weekday']=df['Date/Time'].dt.day_name()
df['day']=df['Date/Time'].dt.day
df['minute']=df['Date/Time'].dt.minute
df['month']=df['Date/Time'].dt.month
df['hour']=df['Date/Time'].dt.hour
df
```

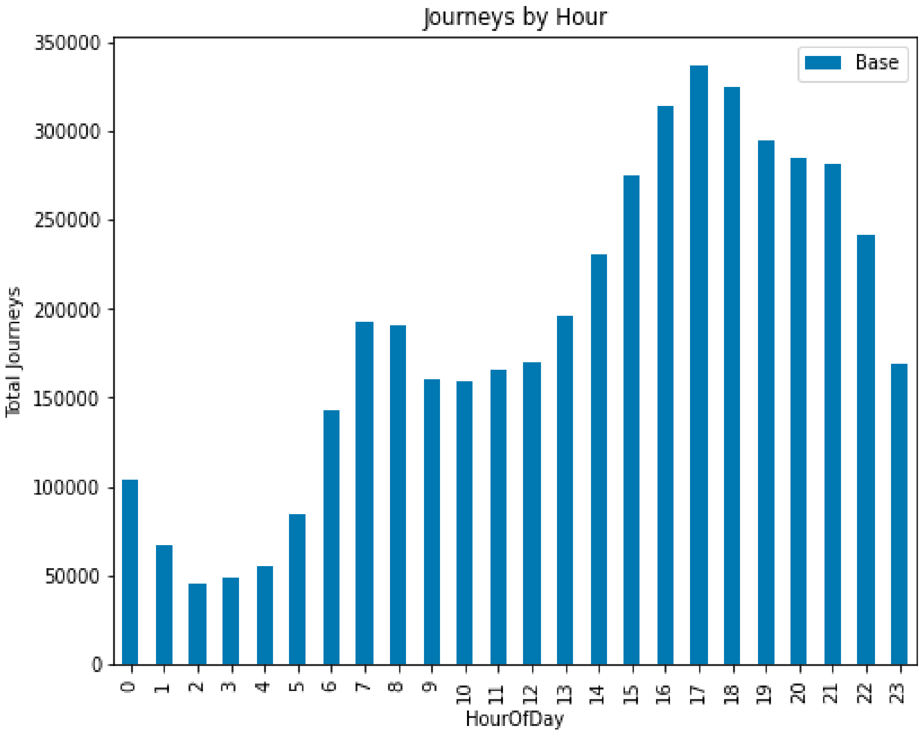
	Date/Time	Lat	Lon	Base	weekday	day	minute	month	hour
0	2014-04-01 00:11:00	40.7690	-73.9549	B02512	Tuesday	1	11	4	0
1	2014-04-01 00:17:00	40.7267	-74.0345	B02512	Tuesday	1	17	4	0
2	2014-04-01 00:21:00	40.7316	-73.9873	B02512	Tuesday	1	21	4	0
3	2014-04-01 00:28:00	40.7588	-73.9776	B02512	Tuesday	1	28	4	0
4	2014-04-01 00:33:00	40.7594	-73.9722	B02512	Tuesday	1	33	4	0
...	...	...	...	...	...	...	...	...	...
1028131	2014-09-30 22:57:00	40.7668	-73.9845	B02764	Tuesday	30	57	9	22
1028132	2014-09-30 22:57:00	40.6911	-74.1773	B02764	Tuesday	30	57	9	22
1028133	2014-09-30 22:58:00	40.8519	-73.9319	B02764	Tuesday	30	58	9	22
1028134	2014-09-30 22:58:00	40.7081	-74.0066	B02764	Tuesday	30	58	9	22
1028135	2014-09-30 22:58:00	40.7140	-73.9496	B02764	Tuesday	30	58	9	22

4534327 rows × 9 columns

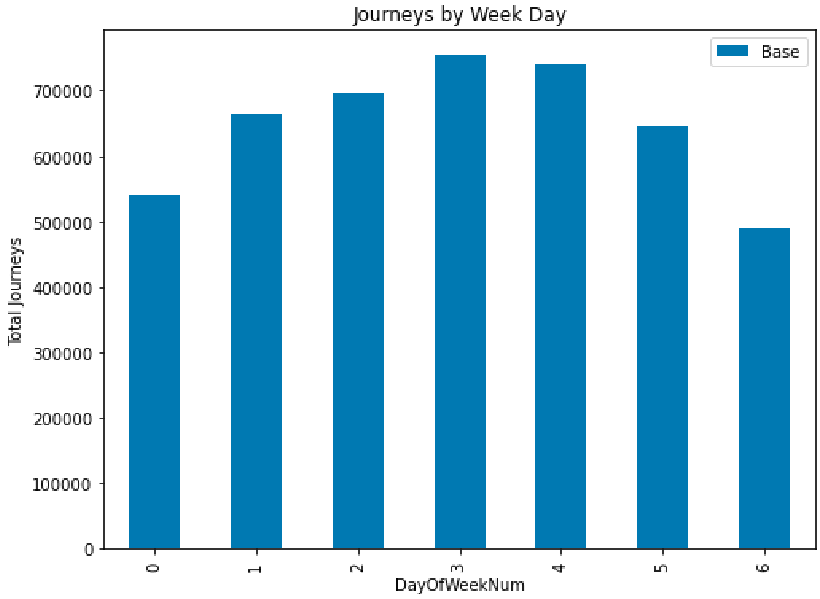
DATA VISUALIZATION:

Data visualization is defined as to evaluate the performance of a model by using graphs and metrics that calculate performance. Data visualization can be mainly used to categorize the data into new levels such that the algorithm used can be generalized to an observation of each output variable derived by an observed input variable.

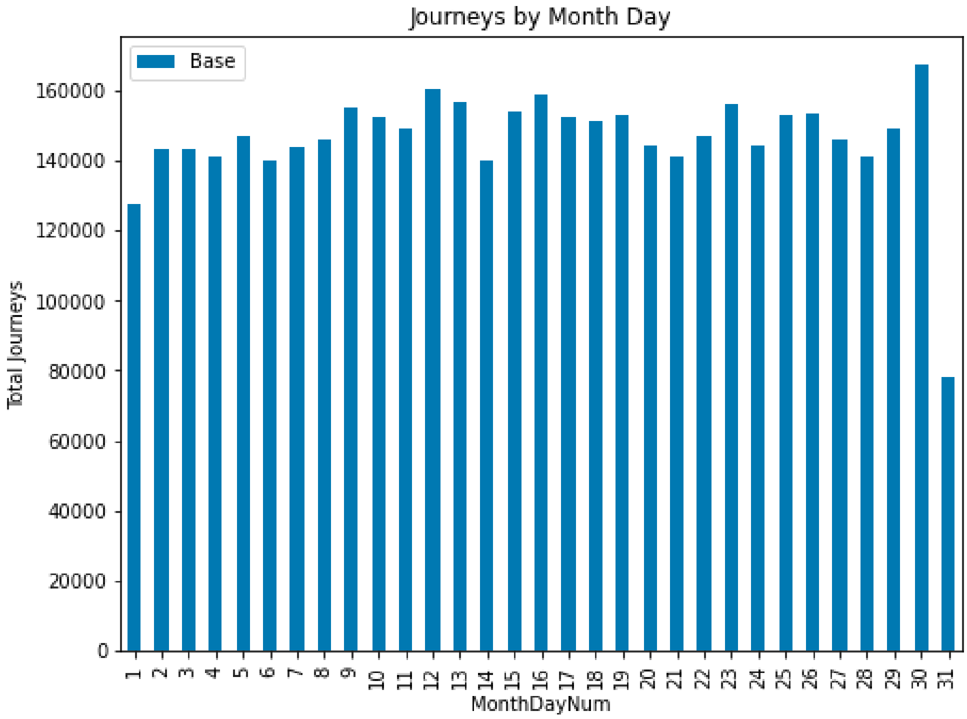
In [8]:
```
uber_hour = df.pivot_table(index=['HourOfDay'],
                                  values='Base',
                                  aggfunc='count')
uber_hour.plot(kind='bar', figsize=(8,6))
plt.ylabel('Total Journeys')
plt.title('Journeys by Hour');
```



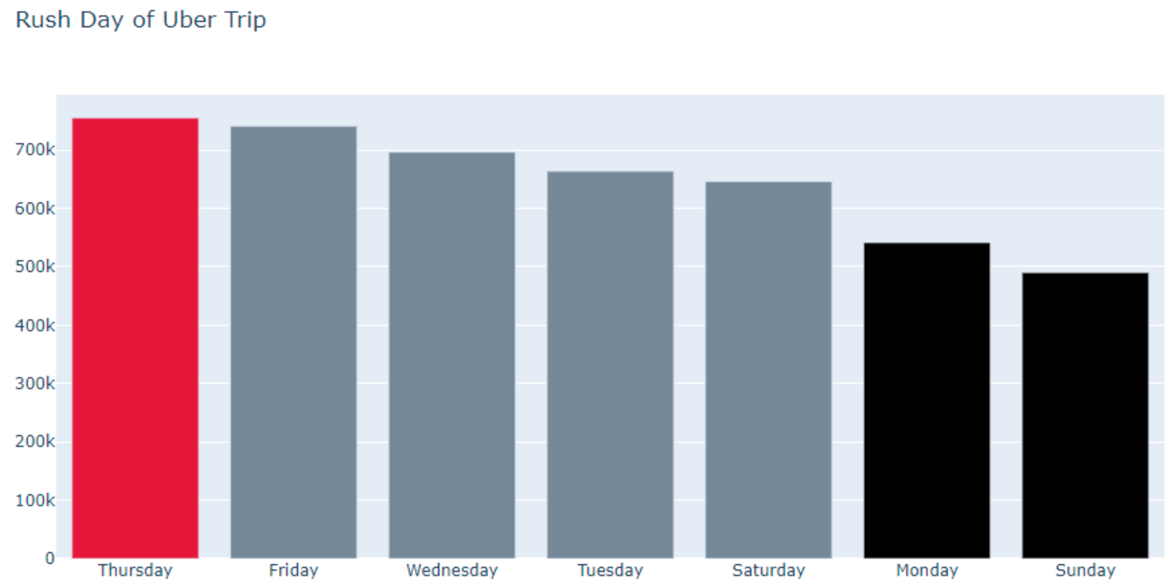
In [6]:
```
uber_weekdays = df.pivot_table(index=['DayOfWeekNum',],
                               values='Base',
                               aggfunc='count')
uber_weekdays.plot(kind='bar', figsize=(8,6))
plt.ylabel('Total Journeys')
plt.title('Journeys by Week Day');
```



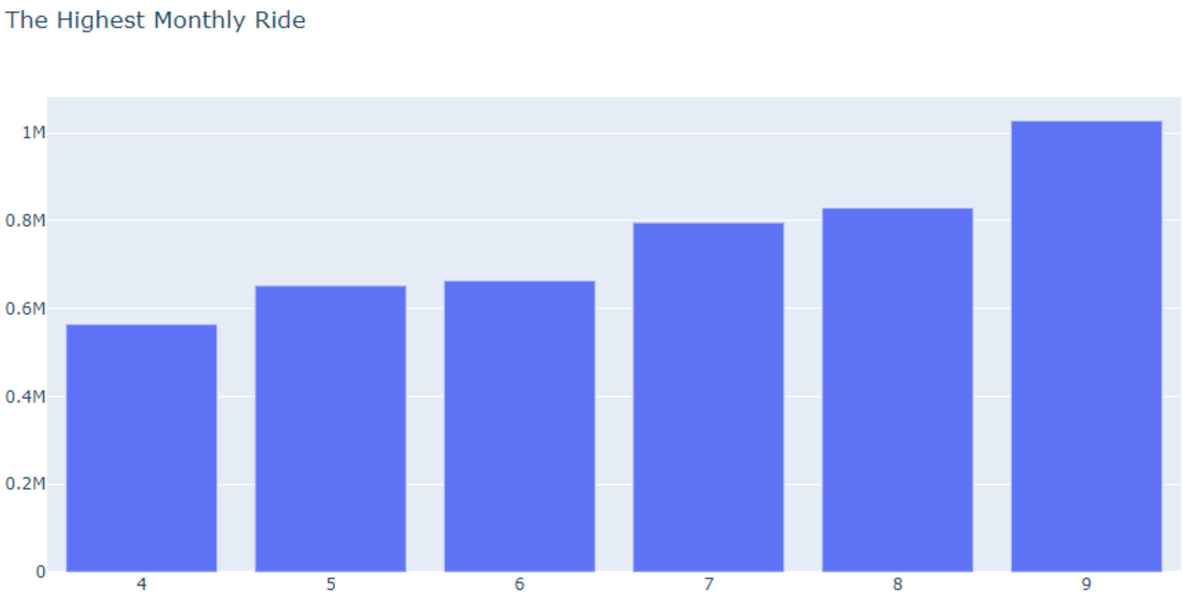

```
uber_monthdays = df.pivot_table(index=['MonthDayNum'],
                                values='Base',
                                aggfunc='count')
uber_monthdays.plot(kind='bar', figsize=(8,6))
plt.ylabel('Total Journeys')
plt.title('Journeys by Month Day');
```



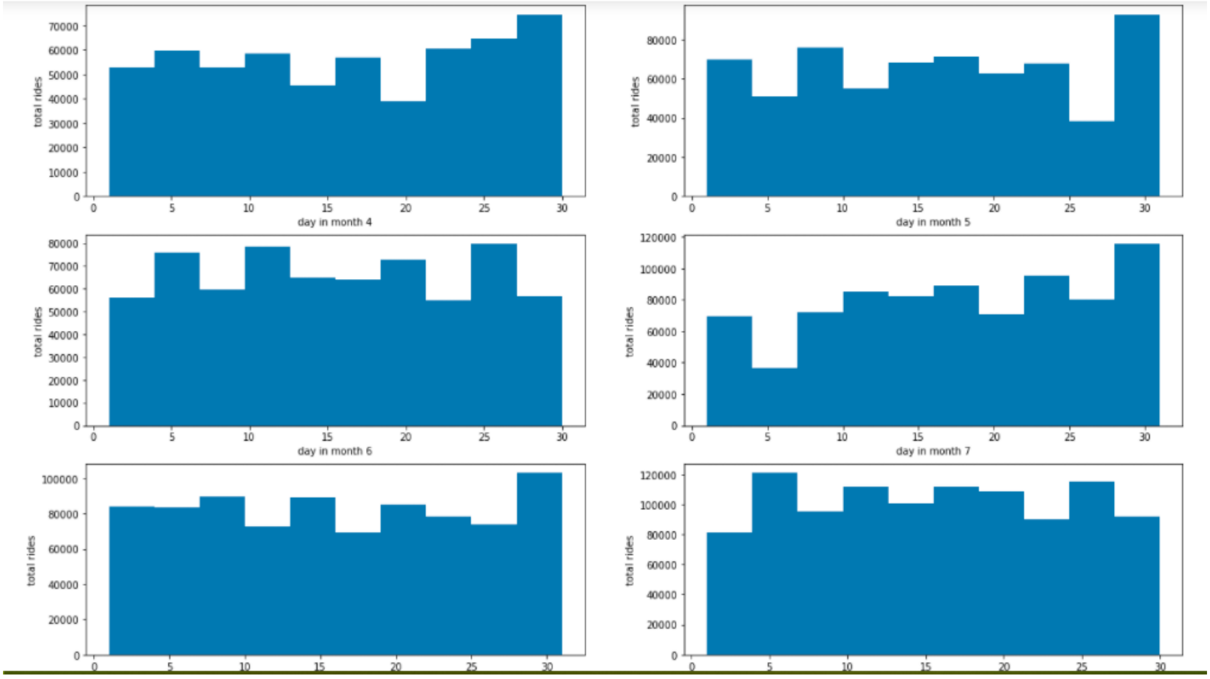
```
colors = ['lightslategray',] * 5
colors[0] = 'crimson'

fig = go.Figure(data=[go.Bar(
    x=df['weekday'].value_counts().index,
    y=df['weekday'].value_counts().values,
    marker_color=colors # marker color can be a single color value or an iterable
)])
fig.update_layout(title_text='Rush Day of Uber Trip')
```




```
fig = go.Figure(data=[go.Bar(
    x = df.groupby('month')['hour'].count().index,
    y = df.groupby('month')['hour'].count(),
    #marker_color=colors # marker color can be a single color value or an iterable
)])
fig.update_layout(title_text='The Highest Monthly Ride')
```

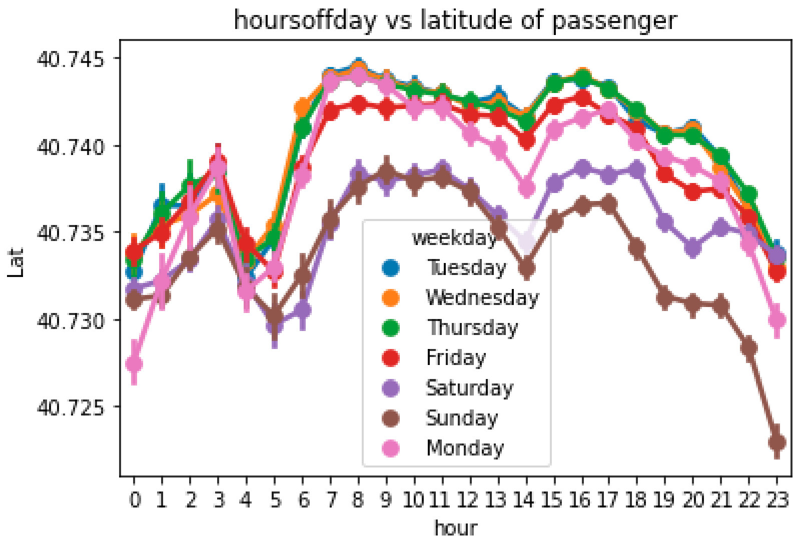


```
plt.figure(figsize=(20,12))
for i, month in enumerate(df['month'].unique(),1):
  plt.subplot(3,2,i)
  df_out=df[df['month']==month]
  plt.hist(df_out['day'])
  plt.xlabel('day in month {}'.format(month))
  plt.ylabel('total rides')
```



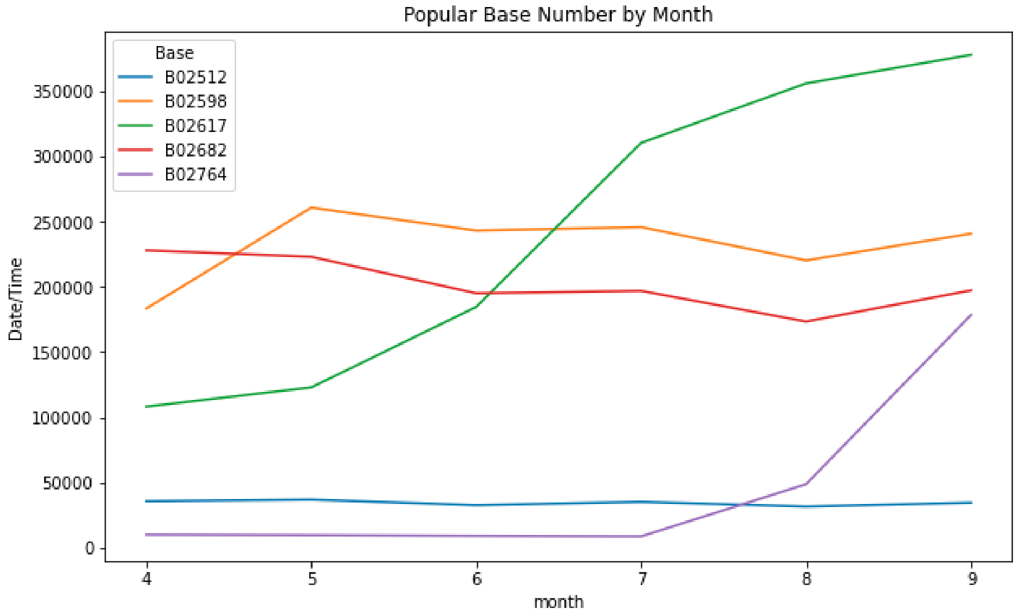
```
ax=sns.pointplot(x='hour',y='Lat', data=df, hue='weekday')
ax.set_title('hoursoffday vs latitude of passenger')
```

```
Text(0.5, 1.0, 'hoursoffday vs latitude of passenger')
```



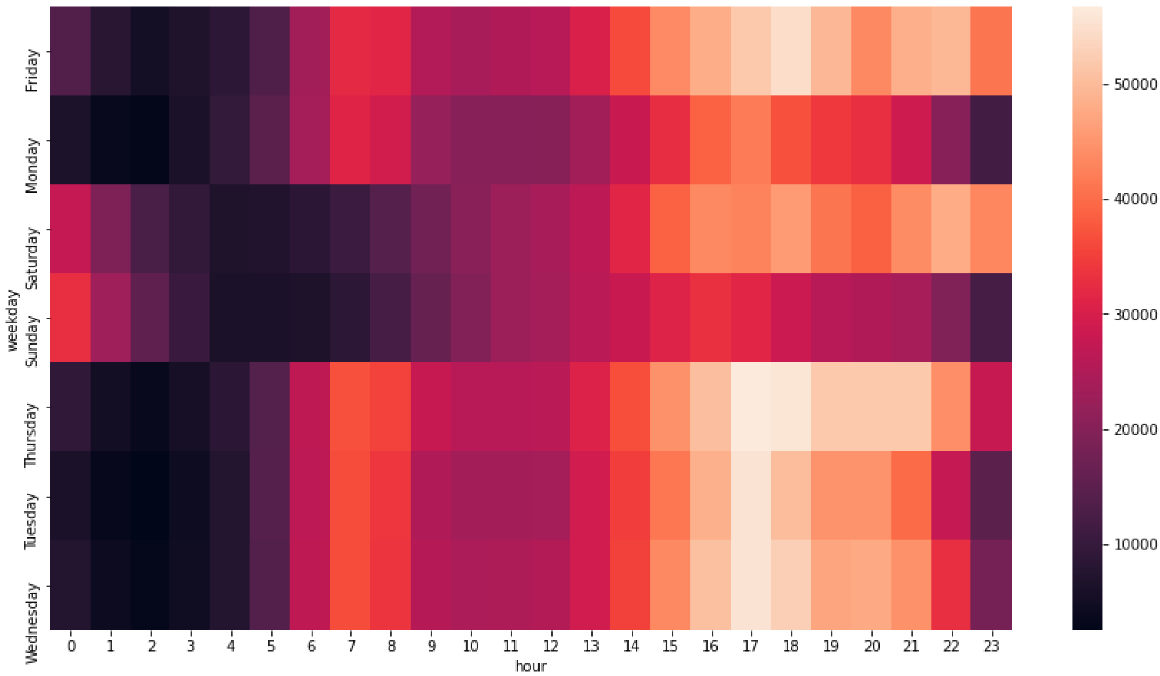
```
plt.figure(figsize=(10,6))
ax = sns.lineplot(x='month',y='Date/Time', hue='Base',data=base)
ax.set_title('Popular Base Number by Month')
```

```
Text(0.5, 1.0, 'Popular Base Number by Month')
```




```
plt.figure(figsize=(15,8))
sns.heatmap(pivot)
```

```
<AxesSubplot:xlabel='hour', ylabel='weekday'>
```



DATA MODELLING:

Based on the problems of forecasting errors and risk of overfitting due to large datasets. The data analyzed and sent to the company is resulted as inefficient and ineffective. Thus to overcome the problem we are going to predict the pickup of cab from a coordinated cluster of points predicted by using applied k-means clustering algorithm. The k-means clustering algorithm adopted will effectively dispatch taxis to the cluster. This facilitates each driver and passenger to attenuate the wait-time to search out one another. Drivers don't have enough info concerning wherever passengers and different taxis area unit and shall move. Therefore, a cab center will organize the taxicab fleet and with efficiency give out consistent request to the whole town. The system uses the latitude and longitude of the cab scheduled and also the day of the travel and the month. An unsupervised learning model is trained with this dataset and the model is employed to predict the pickup of the cab on the cluster.

LINEAR REGRESSION:

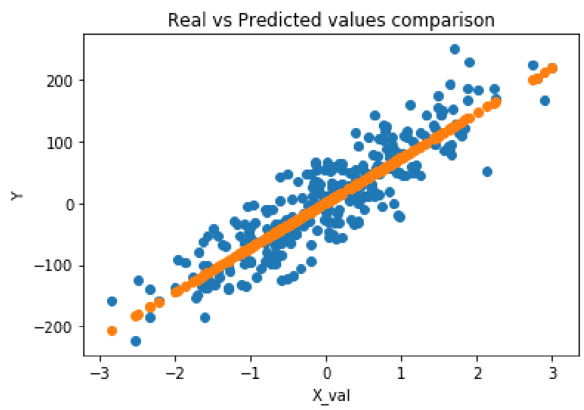
In statistics, linear regression is a linear approach for modelling the relationship between a scalar response and one or more explanatory variables (also known as dependent and independent variables). The case of one explanatory variable is called *simple linear regression*; for more than one, the process is called multiple linear regression. This term is distinct from multivariate linear regression, where multiple correlated dependent variables are predicted, rather than a single scalar variable. In linear regression, the relationships are modeled using linear predictor functions whose unknown model parameters are estimated from the data. Such models are called linear models. Most commonly, the conditional mean of the response given the values of the explanatory variables (or predictors) is assumed to be an affine function of those values less commonly, the conditional median or some other quantile is used. Like all forms of regression analysis, linear regression focuses on the conditional probability distribution of the response given the values of the predictors, rather than on the joint probability distribution of all of these variables, which is the domain of multivariate analysis.

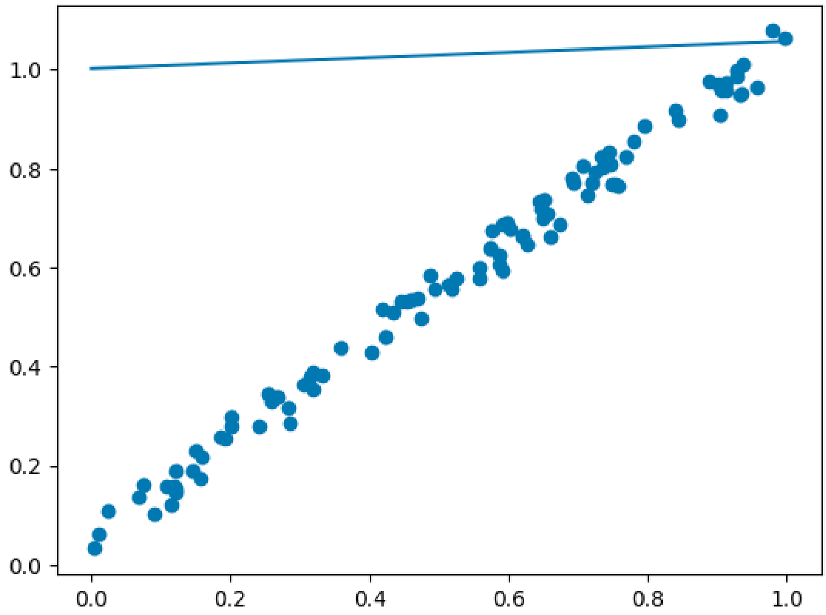
```
In [36]: from sklearn.linear_model import LinearRegression
         # fit the model on the train dataset

         model = LinearRegression()
         model.fit(X_train, Y_train)

Out[36]: LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)
```

```
In [37]: # Predicting for the X_val points

         Y_pred = model.predict(X_val)
```

```
In [38]: from sklearn.metrics import mean_squared_error
         print(f'MSE on the validation set: {mean_squared_error(Y_val, Y_pred)}')

         MSE on the validation set: 1515.7821465595791
```

```
In [39]: plt.xlabel('X_val')
         plt.ylabel('Y')
         plt.title('Real vs Predicted values comparison')

         plt.scatter(X_val, Y_val)
         plt.scatter(X_val, Y_pred)

Out[39]: <matplotlib.collections.PathCollection at 0xf7cb427e80>
```





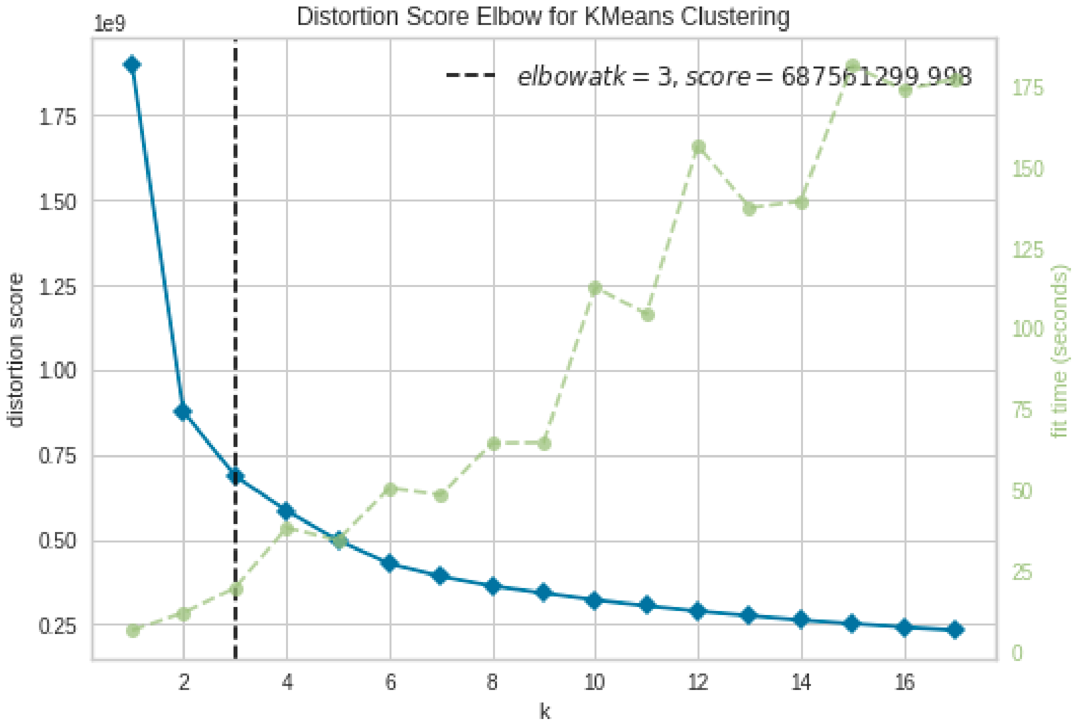
K-MEANS CLUSTERING:

K-Means Clustering is an Unsupervised Learning algorithm, which groups the unlabeled dataset into different clusters. Here K defines the number of pre-defined clusters that need to be created in the process, as if K=2, there will be two clusters, and for K=3, there will be three clusters, and so on. It allows us to cluster the data into different groups and a convenient way to discover the categories of groups in the unlabeled dataset on its own without the need for any training. It is a centroid-based algorithm, where each cluster is associated with a centroid. The main aim of this algorithm is to minimize the sum of distances between the data point and their corresponding clusters.

The k-means clustering algorithm mainly performs two tasks:

a. Determines the best value for K center points or centroids by an iterative process.

b. Assigns each data point to its closest k-center. Those data points which are near to the particular k-center, create a cluster.



TESTING:

The main step after visualizing data in an algorithm is to test the data, the test set can be defined as a set of observations which is used to evaluate the performance of a model by using performance metrics. The program that uses the test set must be able to generalize and effectively perform with the dataset to yield the predicted data accurately such that the program is effective in nature. Moreover when the program memorizes the dataset it is termed overfitting hence to balance overfitting we use regularization which is applied to the model to reduce it.

```
from sklearn.model_selection import train_test_split

np.random.seed(0)
df_train, df_test = train_test_split(df, train_size = 0.70, test_size = 0.30, random_state = 333)
```

```
data_x = df.iloc[:,0:-1].values
data_y = df.iloc[:,-1].values
```

```
X_train,X_test,y_train,y_test = train_test_split(data_x,data_y,test_size=0.3,random_state=0)
```

```
# Create a string for the formula
cols = df.columns.drop('trips')
formula = 'trips ~ ' + ' + '.join(cols)
print(formula, '\n')
```

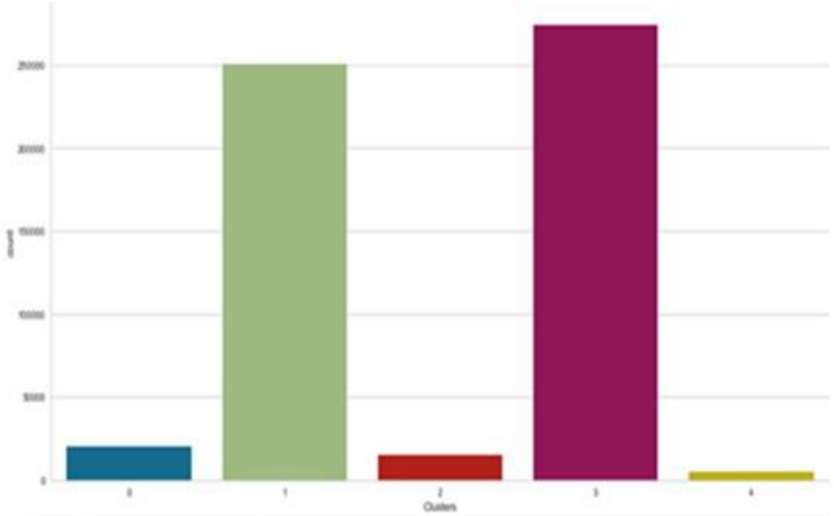
```
# Run the model and report the results
model = smf.glm(formula=formula, data=X_train, family=sm.families.Binomial())
logistic_fit = model.fit()

print(logistic_fit.summary())
```

```
==============================================================================
Dep. Variable:                  trips   No. Observations:                  247
Model:                            GLM   Df Residuals:                      187
Model Family:                Binomial   Df Model:                           59
Link Function:                  logit   Scale:                          1.0000
Method:                          IRLS   Log-Likelihood:                   -inf
Date:                Fri, 24 Sep 2021   Deviance:                   2.6912e+08
Time:                        09:49:27   Pearson chi2:                 2.87e+26
No. Iterations:                     2
Covariance Type:            nonrobust
=====================================================================================
                        coef    std err          z      P>|z|      [0.025      0.975]
-------------------------------------------------------------------------------------
Intercept          -5.452e+18    3.4e+07   -1.6e+11      0.000   -5.45e+18   -5.45e+18
date[T.1/10/2015]   2.502e+19   4.75e+07   5.27e+11      0.000     2.5e+19     2.5e+19
date[T.1/11/2015]   3.963e+18   4.75e+07   8.35e+10      0.000    3.96e+18    3.96e+18
date[T.1/12/2015]  -2.974e+18   4.33e+07  -6.86e+10      0.000   -2.97e+18   -2.97e+18
date[T.1/13/2015]   9.584e+17    4.5e+07   2.13e+10      0.000    9.58e+17    9.58e+17
date[T.1/14/2015]  -1.404e+18   4.33e+07  -3.24e+10      0.000    -1.4e+18    -1.4e+18
date[T.1/15/2015]   1.236e+17   4.33e+07   2.85e+09      0.000    1.24e+17    1.24e+17
date[T.1/16/2015]   4.256e+18    4.5e+07   9.45e+10      0.000    4.26e+18    4.26e+18
date[T.1/17/2015]   1.311e+19    4.5e+07   2.91e+11      0.000    1.31e+19    1.31e+19
date[T.1/18/2015]   2.401e+19   5.13e+07   4.68e+11      0.000     2.4e+19     2.4e+19
date[T.1/19/2015]  -2.233e+18   4.34e+07  -5.15e+10      0.000   -2.23e+18   -2.23e+18
date[T.1/2/2015]   -6.923e+18   5.82e+07  -1.19e+11      0.000   -6.92e+18   -6.92e+18
date[T.1/20/2015]  -1.019e+19    4.5e+07  -2.26e+11      0.000   -1.02e+19   -1.02e+19
date[T.1/21/2015]  -7.099e+18   4.33e+07  -1.64e+11      0.000    -7.1e+18    -7.1e+18
date[T.1/22/2015]   -5.11e+18   4.75e+07  -1.08e+11      0.000   -5.11e+18   -5.11e+18
date[T.1/23/2015]   3.834e+18   5.82e+07   6.59e+10      0.000    3.83e+18    3.83e+18
date[T.1/24/2015]   1.167e+19   4.75e+07   2.46e+11      0.000    1.17e+19    1.17e+19
date[T.1/29/2015]  -1.513e+18   4.75e+07  -3.19e+10      0.000   -1.51e+18   -1.51e+18
date[T.1/3/2015]    7.086e+18   4.75e+07   1.49e+11      0.000    7.09e+18    7.09e+18
date[T.1/30/2015]   9.985e+18   4.75e+07    2.1e+11      0.000    9.99e+18    9.99e+18
date[T.1/31/2015]   2.721e+19    4.5e+07   6.04e+11      0.000    2.72e+19    2.72e+19
date[T.1/4/2015]   -6.217e+18   5.81e+07  -1.07e+11      0.000   -6.22e+18   -6.22e+18
date[T.1/5/2015]   -3.571e+18   5.14e+07  -6.95e+10      0.000   -3.57e+18   -3.57e+18
date[T.1/6/2015]   -7.377e+18   4.33e+07   -1.7e+11      0.000   -7.38e+18   -7.38e+18
date[T.1/7/2015]    1.598e+17   4.75e+07   3.37e+09      0.000     1.6e+17     1.6e+17
date[T.1/8/2015]    4.706e+18   4.75e+07   9.92e+10      0.000    4.71e+18    4.71e+18
date[T.1/9/2015]    5.688e+18   5.13e+07   1.11e+11      0.000    5.69e+18    5.69e+18
date[T.2/1/2015]    1.658e+19    4.5e+07   3.68e+11      0.000    1.66e+19    1.66e+19
date[T.2/10/2015]   2.772e+18   5.14e+07    5.4e+10      0.000    2.77e+18    2.77e+18
date[T.2/11/2015]   1.503e+18   4.75e+07   3.17e+10      0.000     1.5e+18     1.5e+18
date[T.2/12/2015]   8.073e+18   5.13e+07   1.57e+11      0.000    8.07e+18    8.07e+18
date[T.2/13/2015]   7.252e+18   5.82e+07   1.25e+11      0.000    7.25e+18    7.25e+18
date[T.2/14/2015]   2.811e+19   5.81e+07   4.84e+11      0.000    2.81e+19    2.81e+19
date[T.2/15/2015]   3.119e+19    4.5e+07   6.93e+11      0.000    3.12e+19    3.12e+19
date[T.2/16/2015]   1.124e+19    4.5e+07    2.5e+11      0.000    1.12e+19    1.12e+19
date[T.2/17/2015]   3.091e+18   4.75e+07   6.51e+10      0.000    3.09e+18    3.09e+18
date[T.2/18/2015]   1.915e+18   4.75e+07   4.03e+10      0.000    1.91e+18    1.91e+18
date[T.2/19/2015]    1.65e+19   4.75e+07   3.47e+11      0.000    1.65e+19    1.65e+19
date[T.2/2/2015]    1.773e+19   5.82e+07   3.05e+11      0.000    1.77e+19    1.77e+19
date[T.2/20/2015]   2.343e+19    4.5e+07    5.2e+11      0.000    2.34e+19    2.34e+19
date[T.2/21/2015]    2.28e+19   4.75e+07    4.8e+11      0.000    2.28e+19    2.28e+19
date[T.2/22/2015]   4.115e+18   4.75e+07   8.67e+10      0.000    4.11e+18    4.11e+18
date[T.2/23/2015]   3.493e+18   4.75e+07   7.36e+10      0.000    3.49e+18    3.49e+18
date[T.2/24/2015]   9.203e+18   4.75e+07   1.94e+11      0.000     9.2e+18     9.2e+18
date[T.2/25/2015]    -5.3e+17   4.75e+07  -1.12e+10      0.000    -5.3e+17    -5.3e+17
date[T.2/26/2015]   7.662e+18   4.33e+07   1.77e+11      0.000    7.66e+18    7.66e+18
date[T.2/27/2015]    1.31e+19   5.13e+07   2.55e+11      0.000    1.31e+19    1.31e+19
date[T.2/28/2015]   1.831e+19    4.5e+07   4.07e+11      0.000    1.83e+19    1.83e+19
date[T.2/3/2015]    6.917e+18    4.5e+07   1.54e+11      0.000    6.92e+18    6.92e+18
```

COMPARISON AND MEASUREMENT:

The scheduling of the cab can be predicted on the basis of the location given by the user and the proposed method finds the nearest hotspot which is defined as a cluster of points analyzed by k-means clustering and gives info to the cab on the hotspot nearest to the location of the user and is booked to pick up the user.

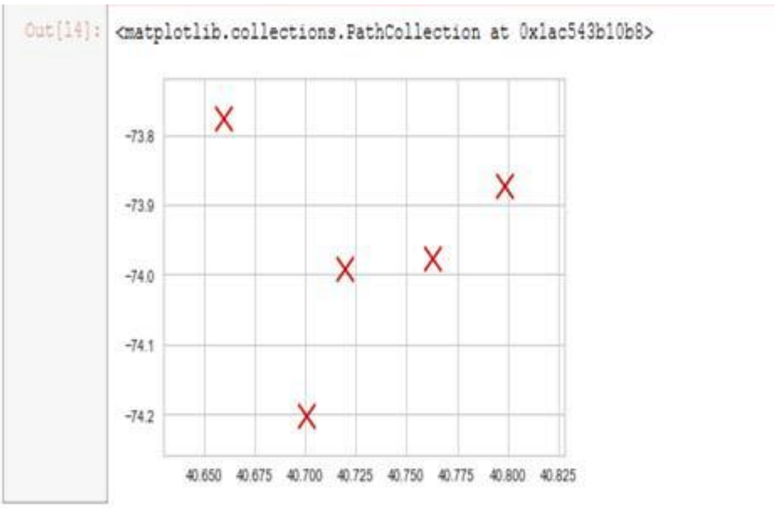
```
In [21]: seaborn.factorplot(data=data_new,x="Clusters",kind="count",size=7,aspect=2)

C:\Users\Rishibingo7\Anaconda3\lib\site-packages\seaborn\categorical.py:3666: UserWarning: The `factorplot` function has be
en renamed to `catplot`. The original name will be removed in a future release. Please update your code. Note that the defa
ult `kind` in `factorplot` (`'point'`) has changed `'strip'` in `catplot`.
  warnings.warn(msg)
C:\Users\Rishibingo7\Anaconda3\lib\site-packages\seaborn\categorical.py:3672: UserWarning: The `size` paramter has been ren
amed to `height`; please update your code.
  warnings.warn(msg, UserWarning)

Out[21]: <seaborn.axisgrid.FacetGrid at 0x1ac54409c50>
```



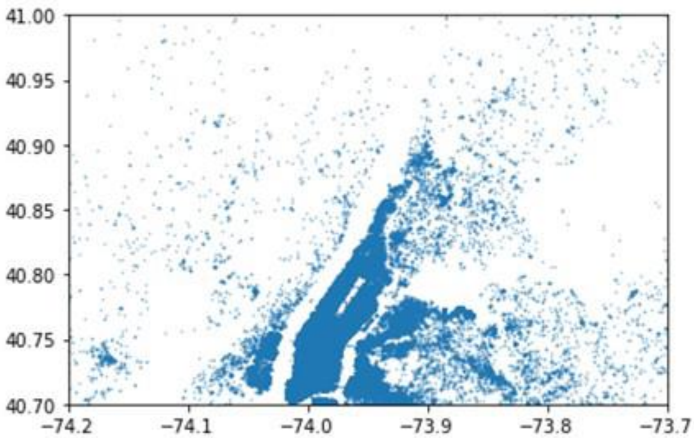

```
In [1]: import matplotlib.pyplot as plt
        from sklearn.cluster import KMeans
        from yellowbrick.cluster import KElbowVisualizer
```

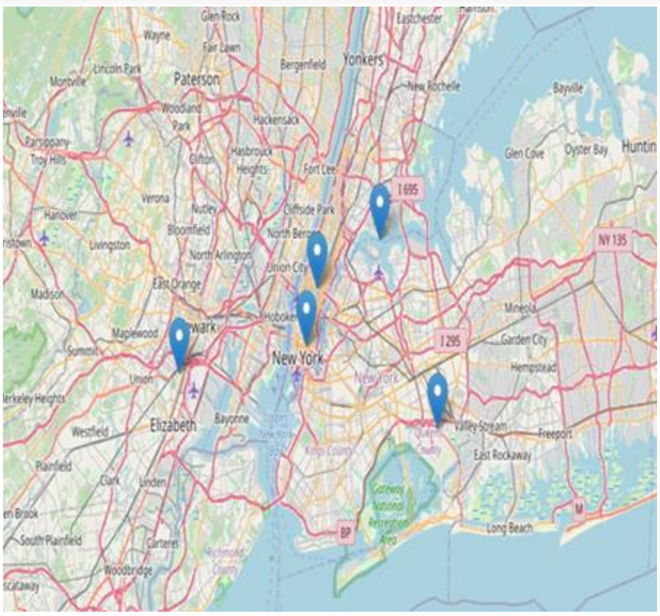
```
In [8]: kmeans = KMeans(n_clusters = 5, random_state = 0)
        kmeans.fit(clus)

Out[8]: KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=300,
               n_clusters=5, n_init=10, n_jobs=None, precompute_distances='auto',
               random_state=0, tol=0.0001, verbose=0)
```

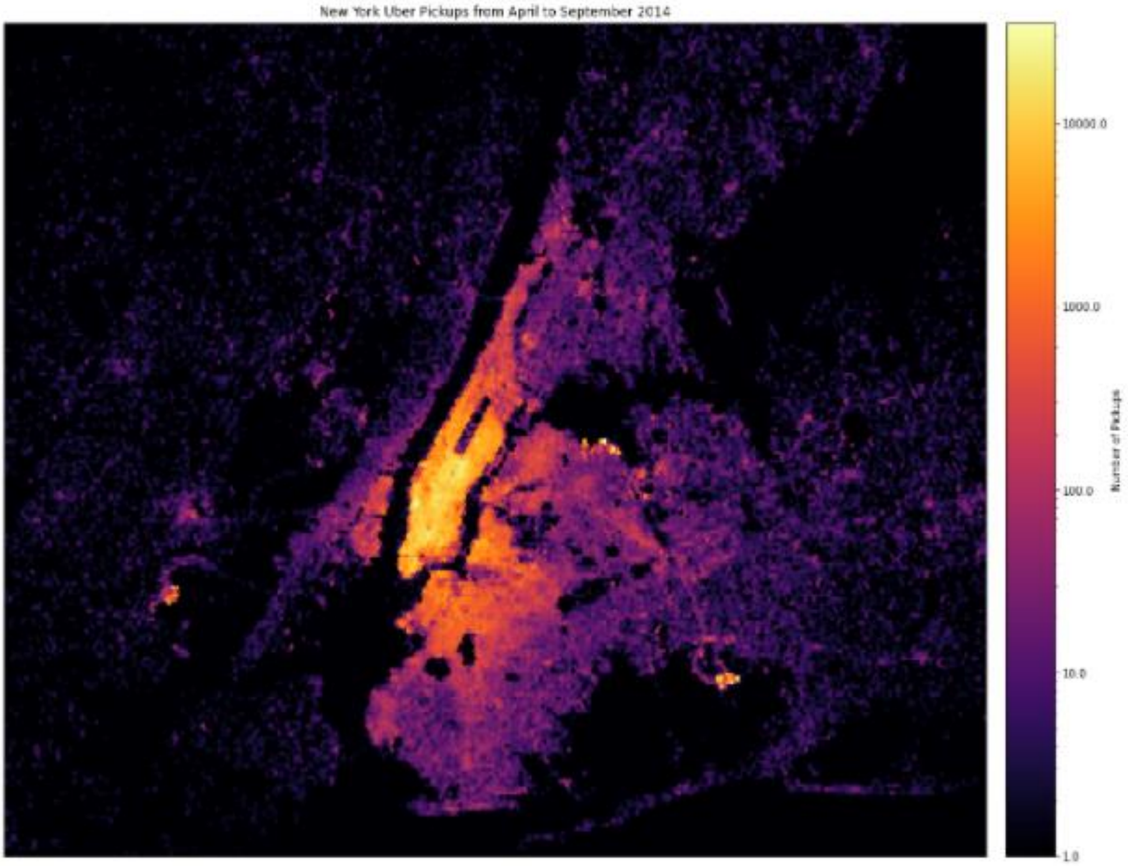
```
In [9]: centroids = kmeans.cluster_centers_
        centroids

Out[9]: array([[ 40.79813773, -73.87204835],
               [ 40.76302051, -73.97574403],
               [ 40.6599309 , -73.77672246],
               [ 40.71968154, -73.99233502],
               [ 40.70048692, -74.20152276]])
```

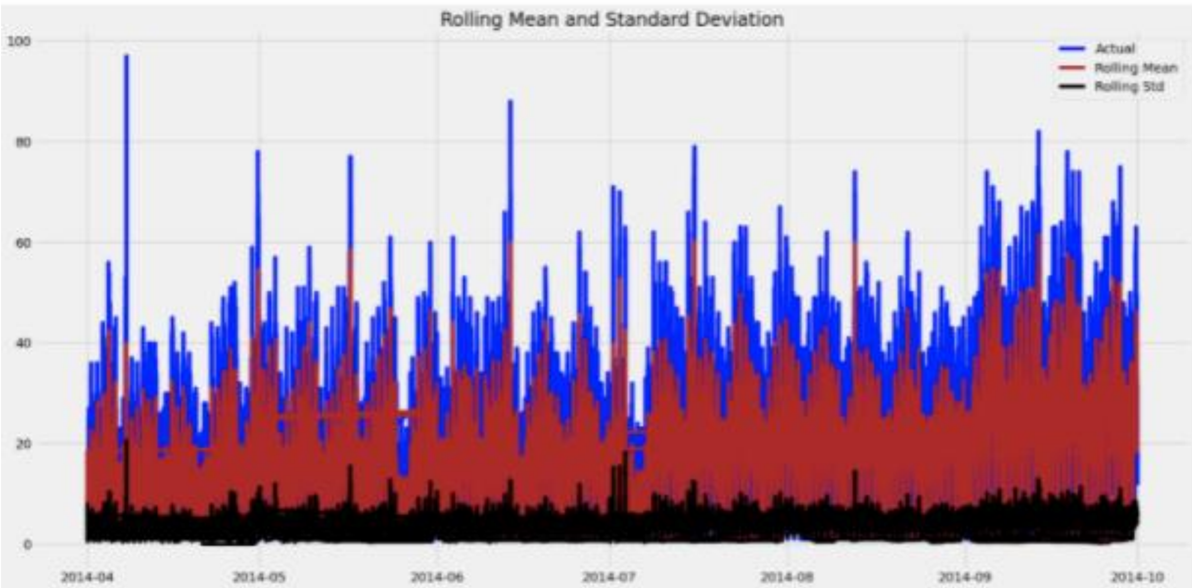


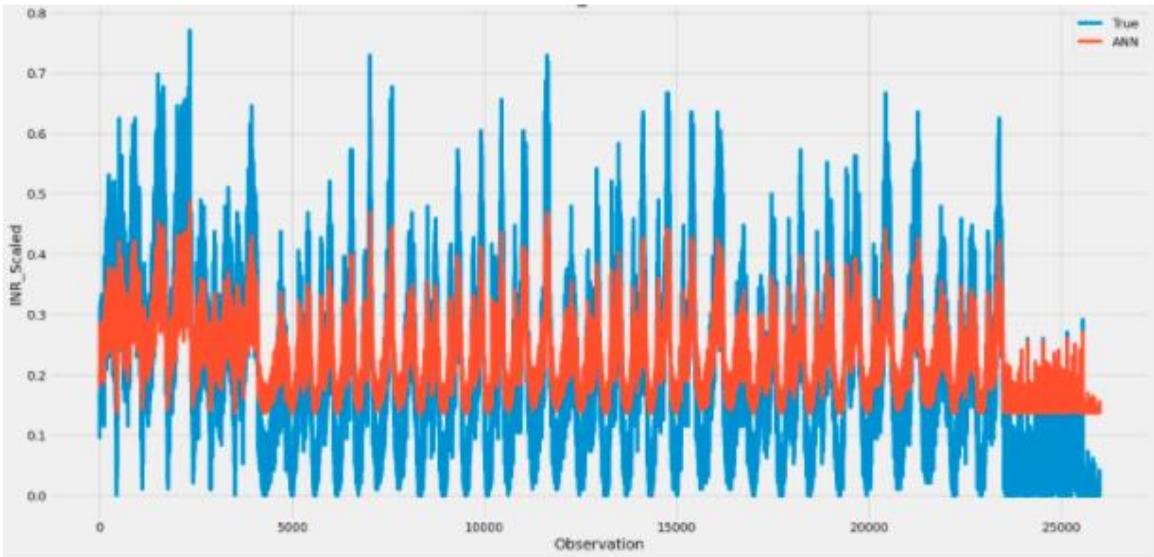
The program predicts the pickup location of theca based on the centroids plotted using applied by k-means clustering for appropriate cab scheduled for pickup. The results discussed are based on the following figures below.



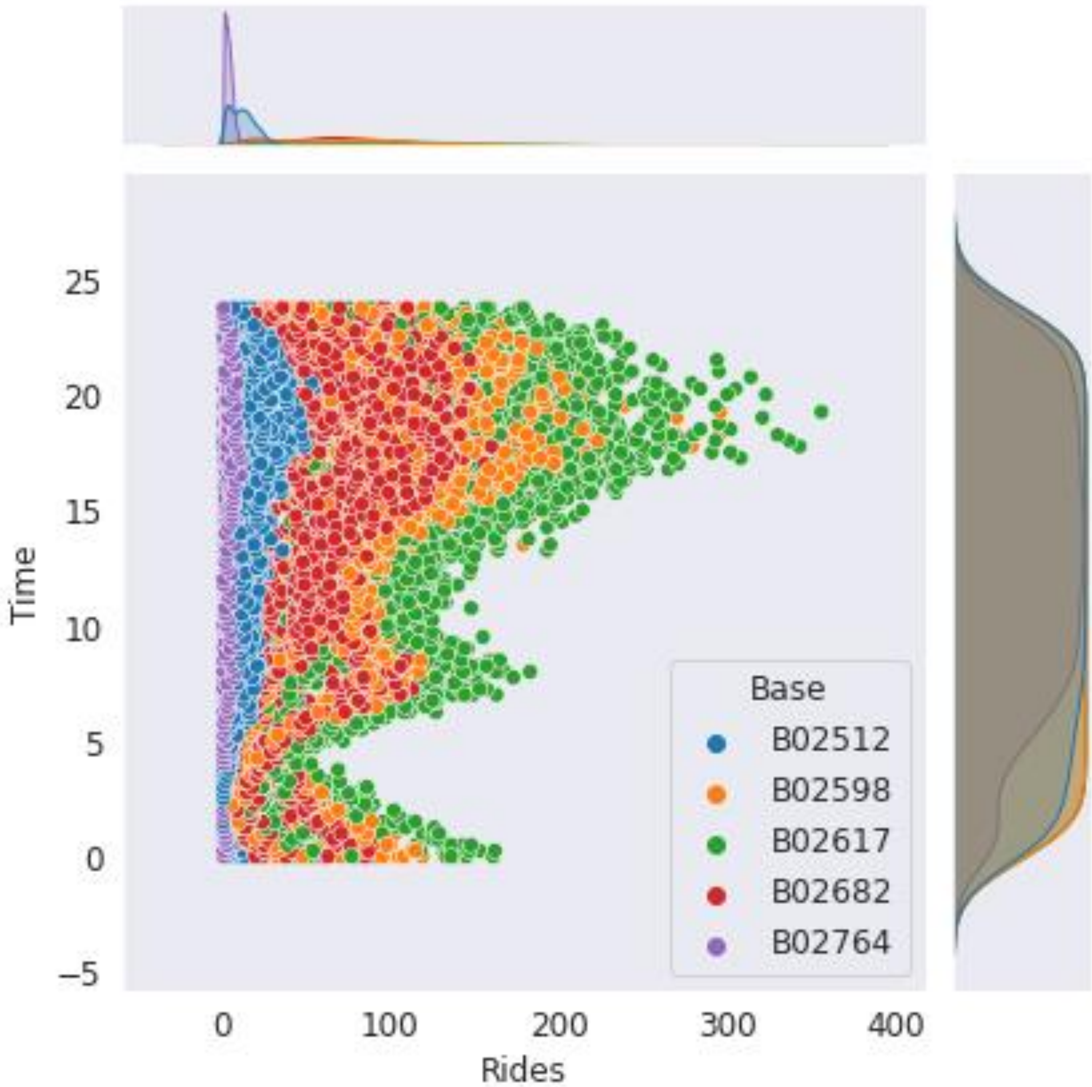



New York Uber Pickups from April to September 2014
10000.0
1000.0
100.0
10.0
1.0
Number of Pickups

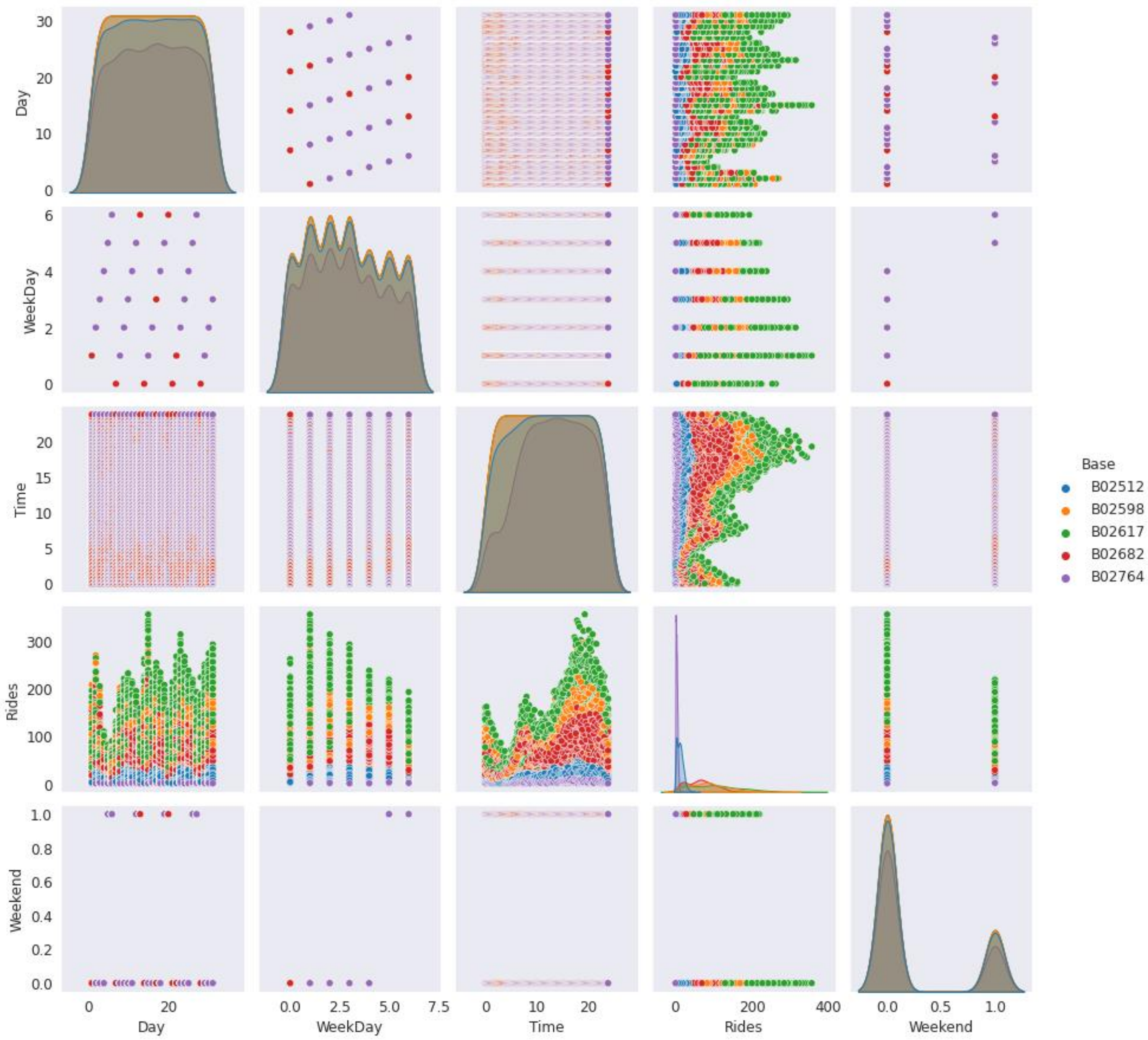

Rolling Mean and Standard Deviation
Actual
Rolling Mean
Rolling Std
100
80
60
40
20
0
2014-04
2014-05
2014-06
2014-07
2014-08
2014-09
2014-10

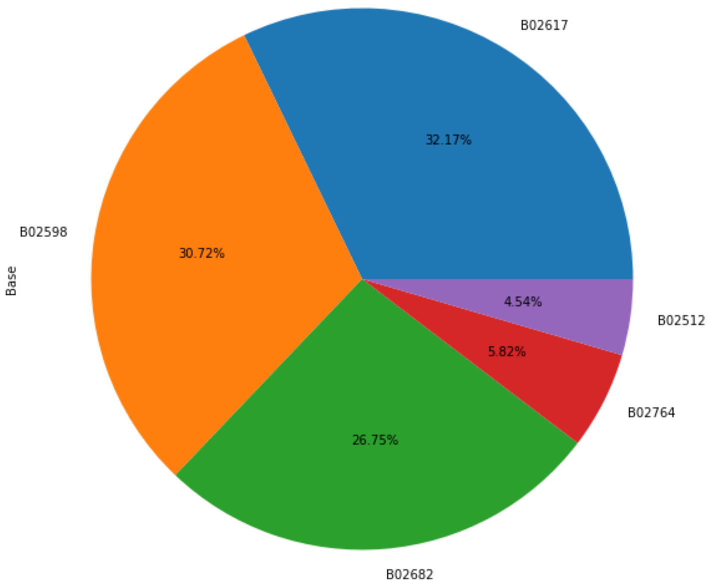


True
ANN
INR_Scaled
Observation

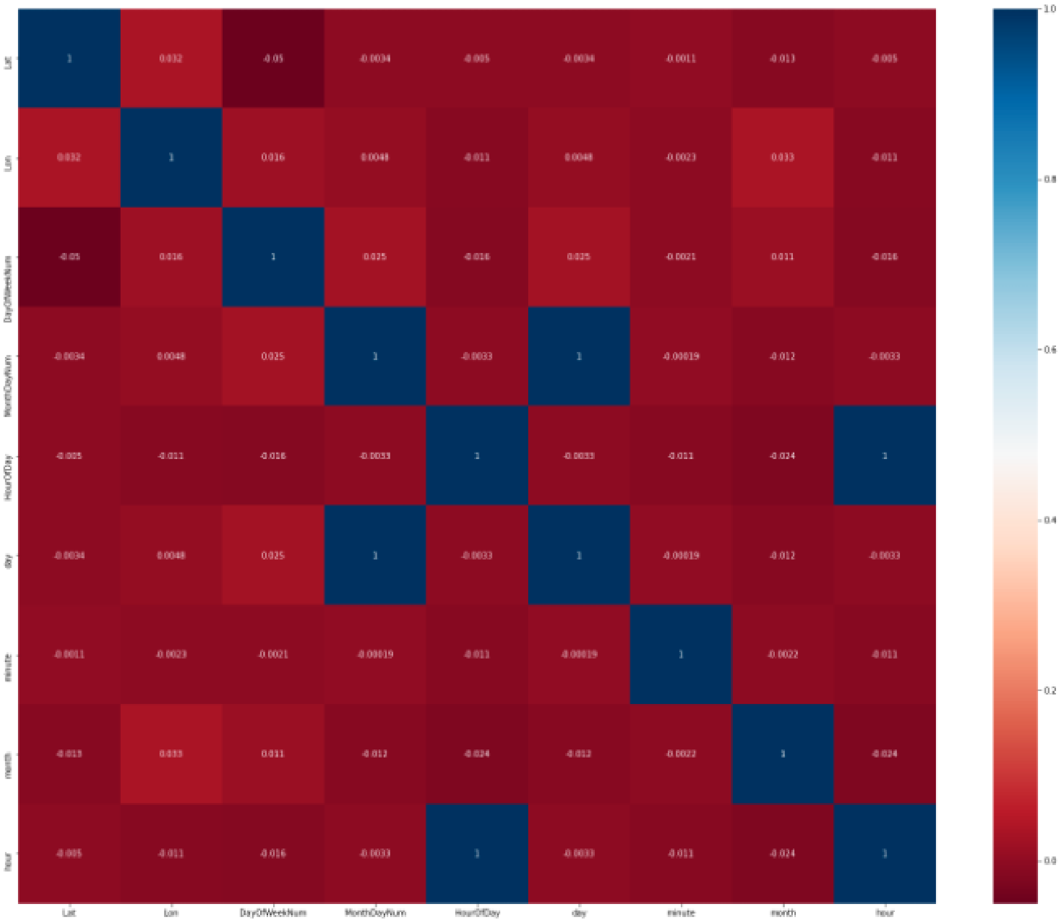


Base
B02512
B02598
B02617
B02682
B02764
Time
Rides


Day
WeekDay
Time
Rides
Weekend
Base
B02512
B02598
B02617
B02682
B02764


Base
B02617
32.17%
B02598
30.72%
4.54%
B02512
5.82%
B02764
26.75%
B02682

```python
plt.figure(figsize = (25,20))
sns.heatmap(df.corr(), annot = True, cmap="RdBu")
plt.show()
```



In [44]:

```python
X = df.iloc[:, 0].values.reshape(-1, 1)
Y = df.iloc[:, 1].values.reshape(-1, 1)
```

In [45]:

```python
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.25, random_state = 42, shuffle = Tru
e)
```

In [46]:

```python
regressor = LinearRegression()
regressor.fit(X_train, Y_train) #training the algorithm
#To retrieve the intercept:
print(regressor.intercept_)

#For retrieving the slope:
print(regressor.coef_)
```

```
[40.90744471]
[[-1.19688232e-19]]
```

PYTHON CODE:

```python
import pandas as pd
import collections
import itertools
import os

# data visualization
import matplotlib as mpl
import matplotlib.pyplot as plt
import seaborn as sns

from plotly.offline import download_plotlyjs, init_notebook_mode, plot, iplot
import plotly as py
import plotly.graph_objs as go

import scipy.stats as stats
from scipy.stats import norm
from scipy.special import boxcox1p
from sklearn import neighbors
from sklearn.metrics import confusion_matrix, classification_report, precision_score
from sklearn.model_selection import train_test_split

# machine learning
from sklearn.preprocessing import StandardScaler

import sklearn.linear_model as skl_lm
from sklearn.linear_model import LinearRegression
from sklearn import preprocessing

import statsmodels
import statsmodels.api as sm
from statsmodels.tsa.arima_model import ARMA
from statsmodels.tsa.stattools import adfuller
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
from statsmodels.tsa.arima_process import ArmaProcess
from statsmodels.tsa.arima_model import ARIMA

import warnings
warnings.filterwarnings("ignore", category=FutureWarning)
```

```python
df = pd.read_csv('./data/uber-raw-data-apr14.csv', parse_dates=['Date/Time'])
df = pd.concat([df,pd.read_csv('./data/uber-raw-data-may14.csv', parse_dates=['Date/Time'])], axis=0)
df = pd.concat([df,pd.read_csv('./data/uber-raw-data-jun14.csv', parse_dates=['Date/Time'])], axis=0)
df = pd.concat([df,pd.read_csv('./data/uber-raw-data-jul14.csv', parse_dates=['Date/Time'])], axis=0)
df = pd.concat([df,pd.read_csv('./data/uber-raw-data-aug14.csv', parse_dates=['Date/Time'])], axis=0)
df = pd.concat([df,pd.read_csv('./data/uber-raw-data-sep14.csv', parse_dates=['Date/Time'])], axis=0)
print(df)
```

Out:

```
                     Date/Time      Lat      Lon    Base
0       2014-04-01 00:11:00  40.7690 -73.9549  B02512
1       2014-04-01 00:17:00  40.7267 -74.0345  B02512
2       2014-04-01 00:21:00  40.7316 -73.9873  B02512
3       2014-04-01 00:28:00  40.7588 -73.9776  B02512
4       2014-04-01 00:33:00  40.7594 -73.9722  B02512
...                     ...      ...      ...     ...
1028131 2014-09-30 22:57:00  40.7668 -73.9845  B02764
1028132 2014-09-30 22:57:00  40.6911 -74.1773  B02764
1028133 2014-09-30 22:58:00  40.8519 -73.9319  B02764
1028134 2014-09-30 22:58:00  40.7081 -74.0066  B02764
1028135 2014-09-30 22:58:00  40.7140 -73.9496  B02764

[4534327 rows x 4 columns]
```

```
df.head()
```

	Date/Time	Lat	Lon	Base
0	2014-04-01 00:11:00	40.7690	-73.9549	B02512
1	2014-04-01 00:17:00	40.7267	-74.0345	B02512
2	2014-04-01 00:21:00	40.7316	-73.9873	B02512
3	2014-04-01 00:28:00	40.7588	-73.9776	B02512
4	2014-04-01 00:33:00	40.7594	-73.9722	B02512

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 4534327 entries, 0 to 1028135
Data columns (total 4 columns):
 #   Column     Dtype
---  ------     -----
 0   Date/Time  datetime64[ns]
 1   Lat        float64
 2   Lon        float64
 3   Base       object
dtypes: datetime64[ns](1), float64(2), object(1)
memory usage: 173.0+ MB
```

```
df = pd.read_csv('./data/uber-raw-data-apr14.csv', parse_dates=['Date/Time'])
df = pd.concat([df,pd.read_csv('./data/uber-raw-data-may14.csv', parse_dates=['Date/Time'])], axis=0)
df = pd.concat([df,pd.read_csv('./data/uber-raw-data-jun14.csv', parse_dates=['Date/Time'])], axis=0)
df = pd.concat([df,pd.read_csv('./data/uber-raw-data-jul14.csv', parse_dates=['Date/Time'])], axis=0)
df = pd.concat([df,pd.read_csv('./data/uber-raw-data-aug14.csv', parse_dates=['Date/Time'])], axis=0)
df = pd.concat([df,pd.read_csv('./data/uber-raw-data-sep14.csv', parse_dates=['Date/Time'])], axis=0)
print(df)
```


```
                     Date/Time      Lat      Lon    Base
0       2014-04-01 00:11:00  40.7690 -73.9549  B02512
1       2014-04-01 00:17:00  40.7267 -74.0345  B02512
2       2014-04-01 00:21:00  40.7316 -73.9873  B02512
3       2014-04-01 00:28:00  40.7588 -73.9776  B02512
4       2014-04-01 00:33:00  40.7594 -73.9722  B02512
...                     ...      ...      ...     ...
1028131 2014-09-30 22:57:00  40.7668 -73.9845  B02764
1028132 2014-09-30 22:57:00  40.6911 -74.1773  B02764
1028133 2014-09-30 22:58:00  40.8519 -73.9319  B02764
1028134 2014-09-30 22:58:00  40.7081 -74.0066  B02764
1028135 2014-09-30 22:58:00  40.7140 -73.9496  B02764

[4534327 rows x 4 columns]
```

```python
df['weekday']=df['Date/Time'].dt.day_name()
df['day']=df['Date/Time'].dt.day
df['minute']=df['Date/Time'].dt.minute
df['month']=df['Date/Time'].dt.month
df['hour']=df['Date/Time'].dt.hour
df
```

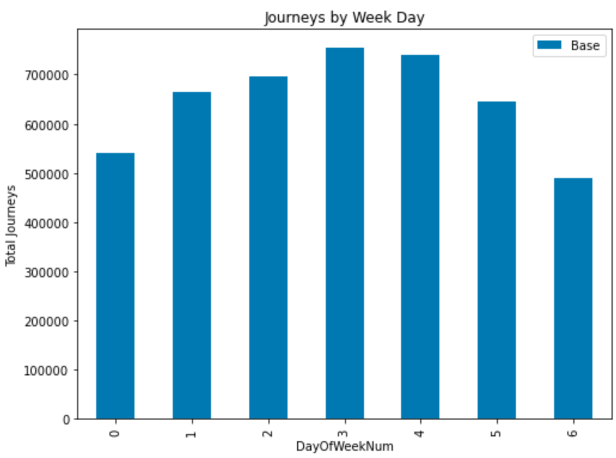
	Date/Time	Lat	Lon	Base	weekday	day	minute	month	hour
0	2014-04-01 00:11:00	40.7690	-73.9549	B02512	Tuesday	1	11	4	0
1	2014-04-01 00:17:00	40.7267	-74.0345	B02512	Tuesday	1	17	4	0
2	2014-04-01 00:21:00	40.7316	-73.9873	B02512	Tuesday	1	21	4	0
3	2014-04-01 00:28:00	40.7588	-73.9776	B02512	Tuesday	1	28	4	0
4	2014-04-01 00:33:00	40.7594	-73.9722	B02512	Tuesday	1	33	4	0
...	...	...	...	...	...	...	...	...	...
1028131	2014-09-30 22:57:00	40.7668	-73.9845	B02764	Tuesday	30	57	9	22
1028132	2014-09-30 22:57:00	40.6911	-74.1773	B02764	Tuesday	30	57	9	22
1028133	2014-09-30 22:58:00	40.8519	-73.9319	B02764	Tuesday	30	58	9	22
1028134	2014-09-30 22:58:00	40.7081	-74.0066	B02764	Tuesday	30	58	9	22
1028135	2014-09-30 22:58:00	40.7140	-73.9496	B02764	Tuesday	30	58	9	22

4534327 rows × 9 columns

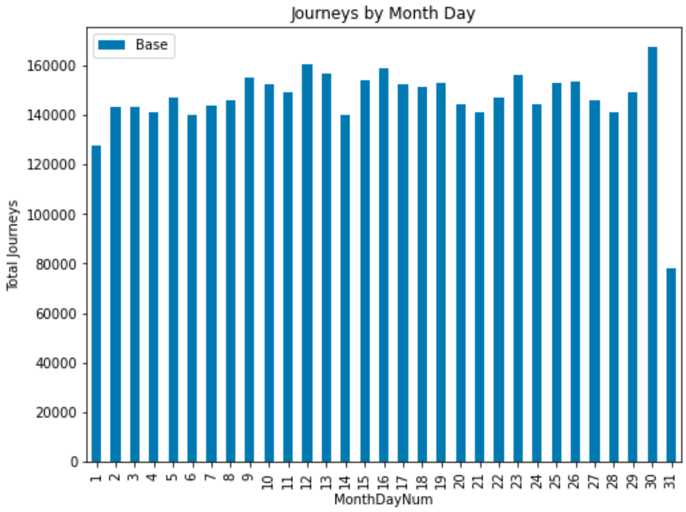
```python
:  df['Date/Time'] = pd.to_datetime(df['Date/Time'], format="%m/%d/%Y %H:%M:%S
   df['DayOfWeekNum'] = df['Date/Time'].dt.dayofweek
   df['MonthDayNum'] = df['Date/Time'].dt.day
   df['HourOfDay'] = df['Date/Time'].dt.hour
```

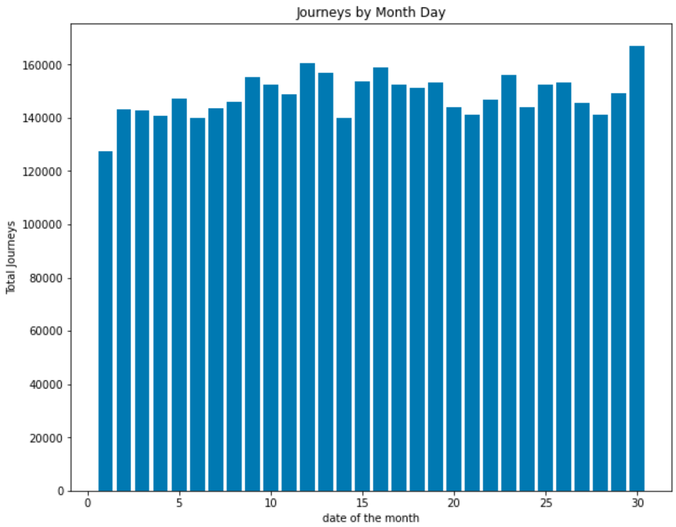
```python
:  uber_weekdays = df.pivot_table(index=['DayOfWeekNum',],
                                  values='Base',
                                  aggfunc='count')
   uber_weekdays.plot(kind='bar', figsize=(8,6))
   plt.ylabel('Total Journeys')
   plt.title('Journeys by Week Day');
```


```python
In [6]: uber_weekdays = df.pivot_table(index=['DayOfWeekNum',],
                                       values='Base',
                                       aggfunc='count')
        uber_weekdays.plot(kind='bar', figsize=(8,6))
        plt.ylabel('Total Journeys')
        plt.title('Journeys by Week Day');
```

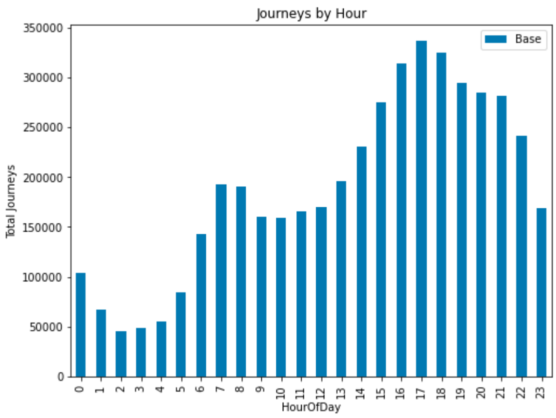



```python
uber_monthdays = df.pivot_table(index=['MonthDayNum'],
                                values='Base',
                                aggfunc='count')
uber_monthdays.plot(kind='bar', figsize=(8,6))
plt.ylabel('Total Journeys')
plt.title('Journeys by Month Day');
```





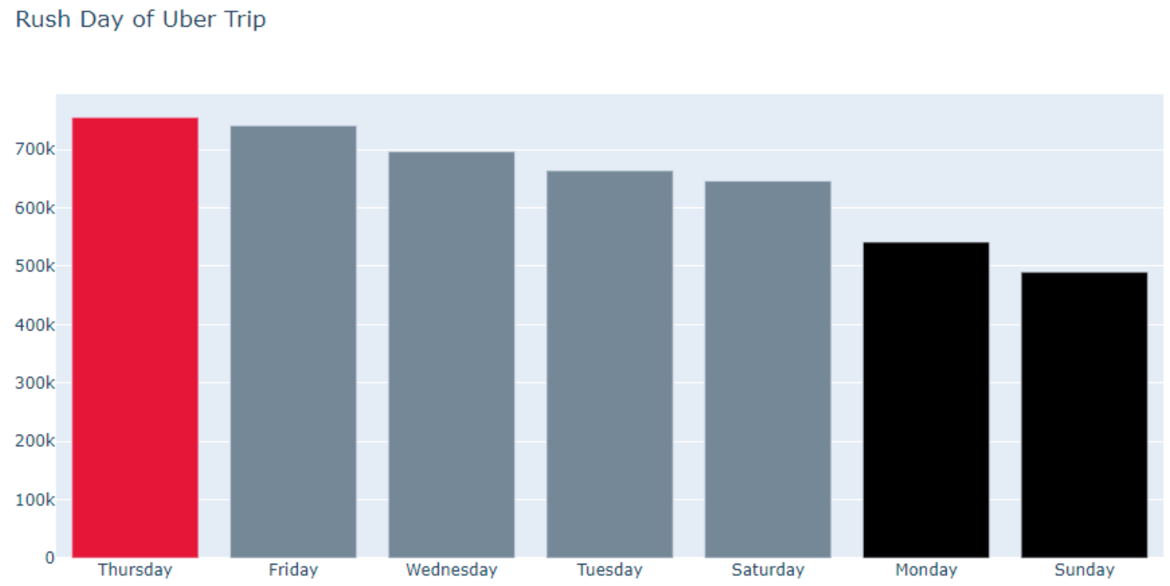
In [8]:
```python
uber_hour = df.pivot_table(index=['HourOfDay'],
                           values='Base',
                           aggfunc='count')
uber_hour.plot(kind='bar', figsize=(8,6))
plt.ylabel('Total Journeys')
plt.title('Journeys by Hour');
```

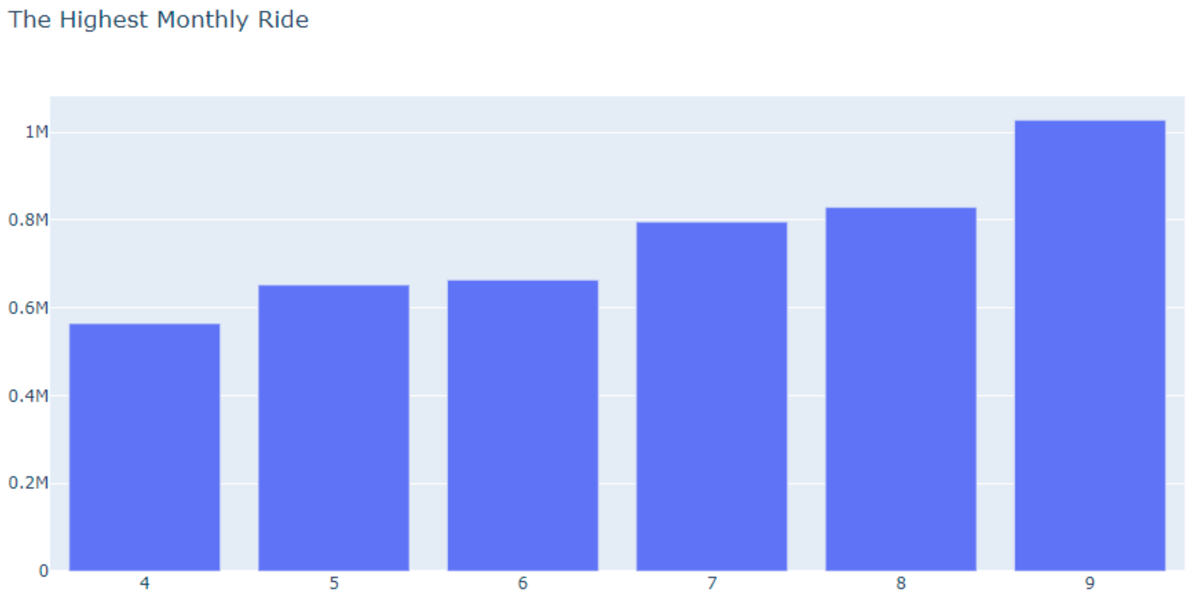


```
colors = ['lightslategray',] * 5
colors[0] = 'crimson'

fig = go.Figure(data=[go.Bar(
    x=df['weekday'].value_counts().index,
    y=df['weekday'].value_counts().values,
    marker_color=colors # marker color can be a single color value or an iterable
)])
fig.update_layout(title_text='Rush Day of Uber Trip')
```

```
fig = go.Figure(data=[go.Bar(
    x = df.groupby('month')['hour'].count().index,
    y = df.groupby('month')['hour'].count(),
    #marker_color=colors # marker color can be a single color value or an iterable
)])
fig.update_layout(title_text='The Highest Monthly Ride')
```

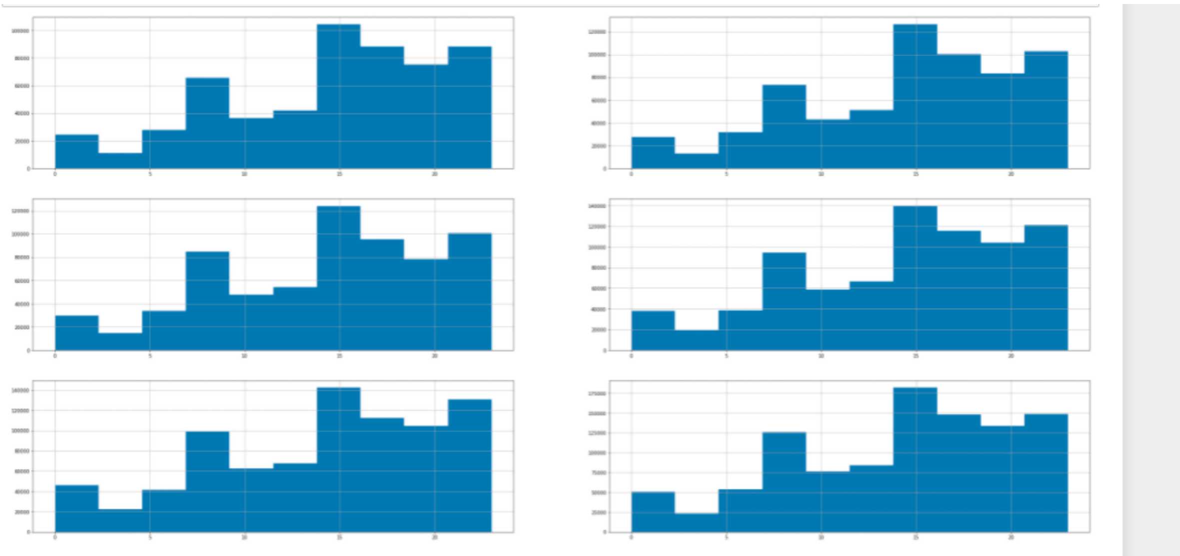
<u>Out:</u>





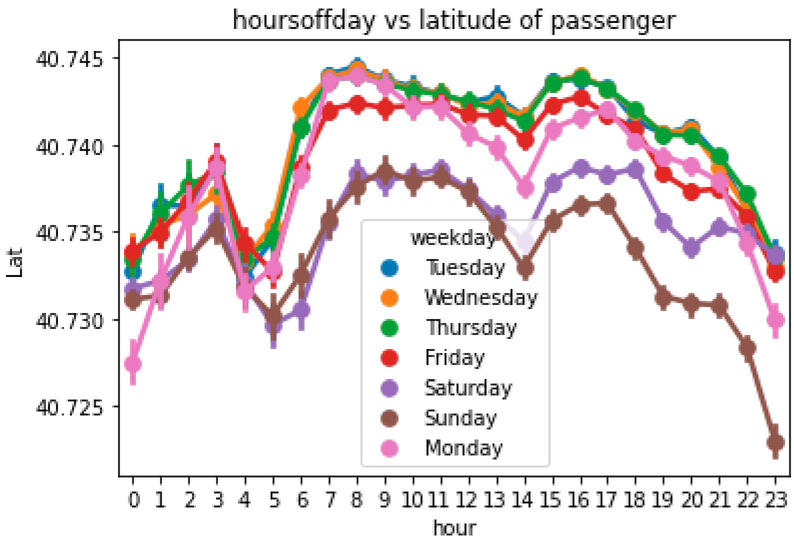
```
plt.figure(figsize=(20,12))
for i, month in enumerate(df['month'].unique(),1):
  plt.subplot(3,2,i)
  df_out=df[df['month']==month]
  plt.hist(df_out['day'])
  plt.xlabel('day in month {}'.format(month))
  plt.ylabel('total rides')
```

<u>Out:</u>



```
ax=sns.pointplot(x='hour',y='Lat', data=df, hue='weekday')
ax.set_title('hoursoffday vs latitude of passenger')
```

```
Text(0.5, 1.0, 'hoursoffday vs latitude of passenger')
```

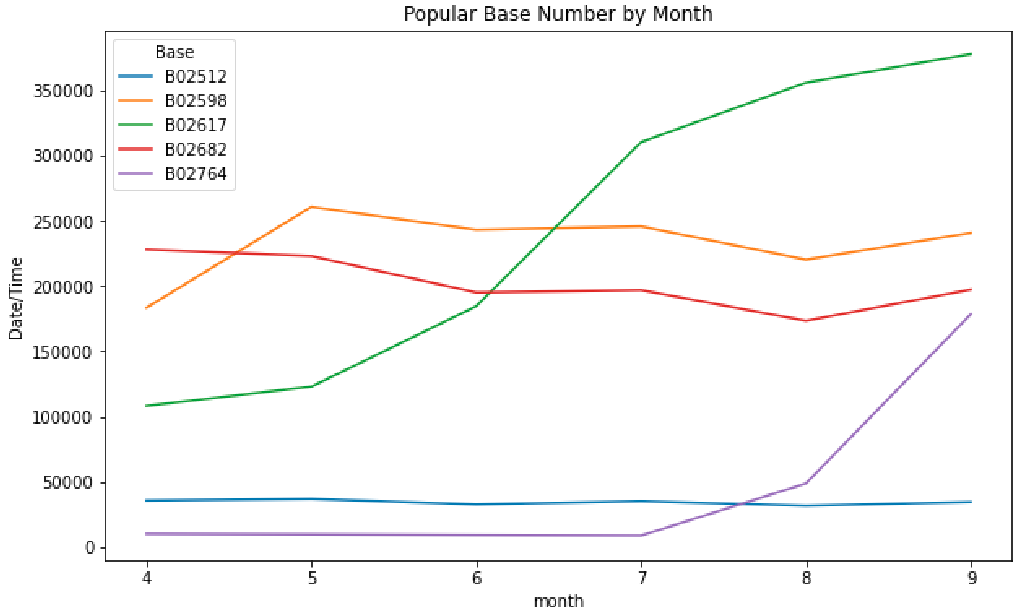


```python
base=df.groupby(['Base','month'])['Date/Time'].count().reset_index()
base
```

	Base	month	Date/Time
0	B02512	4	35536
1	B02512	5	36765
2	B02512	6	32509
3	B02512	7	35021
4	B02512	8	31472
5	B02512	9	34370
6	B02598	4	183263
7	B02598	5	260549
8	B02598	6	242975
9	B02598	7	245597
10	B02598	8	220129
11	B02598	9	240600
12	B02617	4	108001
13	B02617	5	122734
14	B02617	6	184460
15	B02617	7	310160
16	B02617	8	355803
17	B02617	9	377695
18	B02682	4	227808
19	B02682	5	222883
20	B02682	6	194926
21	B02682	7	196754
22	B02682	8	173280
23	B02682	9	197138
24	B02764	4	9908
25	B02764	5	9504
26	B02764	6	8974
27	B02764	7	8589
28	B02764	8	48591
29	B02764	9	178333

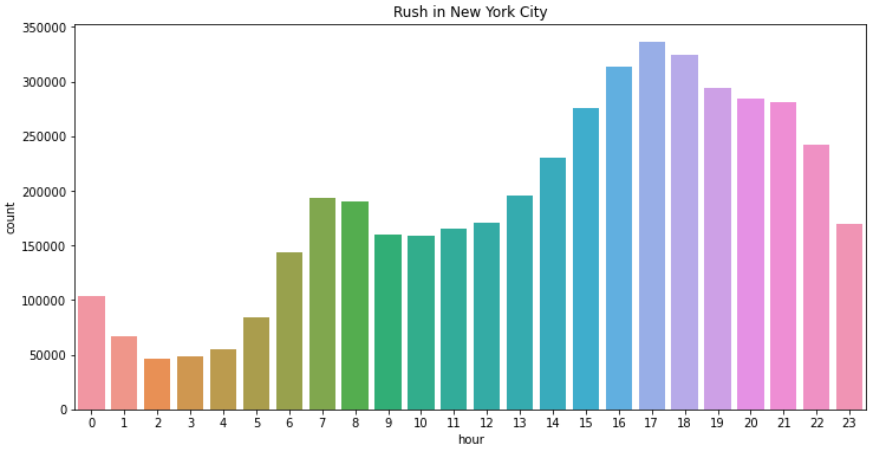
```python
plt.figure(figsize=(10,6))
ax = sns.lineplot(x='month',y='Date/Time', hue='Base',data=base)
ax.set_title('Popular Base Number by Month')
```

```
Text(0.5, 1.0, 'Popular Base Number by Month')
```



```
plt.figure(figsize=(12,6))
sns.countplot(df['hour'])
plt.title("Rush in New York City")
```

```
Text(0.5, 1.0, 'Rush in New York City')
```




```
#Heatmap by hour and weekday
def count_rows(rows):
  return len(rows)
by_cross = df.groupby(['weekday','hour']).apply(count_rows)
by_cross
```

```
weekday    hour
Friday     0       13716
           1        8163
           2        5350
           3        6930
           4        8806
                   ...
Wednesday  19      47017
           20      47772
           21      44553
           22      32868
           23      18146
Length: 168, dtype: int64
```

```
pivot=by_cross.unstack()
pivot
```

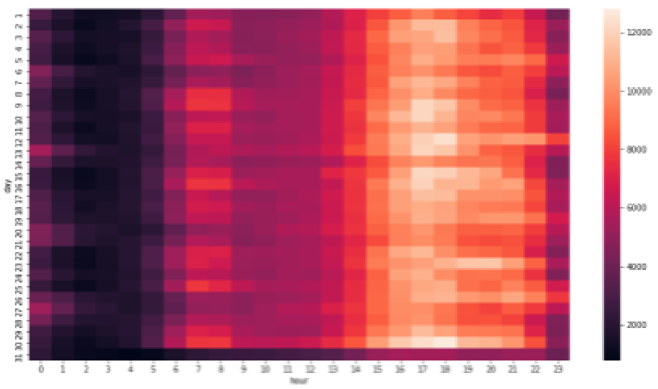
hour weekday	0	1	2	3	4	5	6	7	8	9	...	14	15	16	17	18	19	20	21	22	2
Friday	13716	8163	5350	6930	8806	13450	23412	32061	31509	25230	...	38208	43673	48169	51961	54762	49595	43542	48323	49409	4128
Monday	6436	3737	2938	6232	9640	15032	23746	31159	29265	22197	...	28157	32744	38770	42023	37000	34159	32849	28925	20158	1181
Saturday	27633	19189	12710	9542	6846	7084	8579	11014	14411	17669	...	31418	38769	43512	42844	45883	41098	38714	43826	47951	4317
Sunday	32877	23015	15436	10597	6374	6169	6596	8728	12128	16401	...	28151	31112	33038	31521	28291	25948	25076	23967	19566	1218
Thursday	9293	5290	3719	5637	8505	14169	27065	37038	35431	27812	...	38699	44442	50560	56704	55825	51907	51990	51953	44194	2776
Tuesday	6237	3509	2571	4494	7548	14241	26872	36599	33934	25023	...	34846	41338	48667	55500	50186	44789	44661	39913	27712	1486
Wednesday	7644	4324	3141	4855	7511	13794	26943	36495	33828	25635	...	35148	43388	50684	55637	52732	47017	47772	44553	32868	1814

7 rows × 24 columns

```
#Heatmap by hour and day, month and day, month and weekday
def heatmap(col1, col2):
  by_cross = df.groupby([col1,col2]).apply(count_rows)
  pivot=by_cross.unstack()
  plt.figure(figsize=(15,8))
  return sns.heatmap(pivot)
```

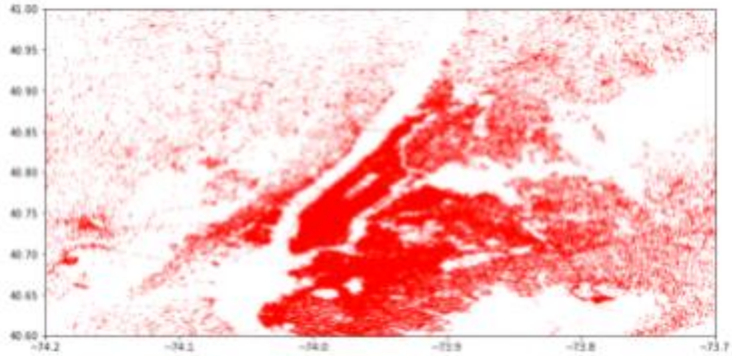
```
#Heatmap by hour and day
heatmap('day','hour')
```

```
<AxesSubplot:xlabel='hour', ylabel='day'>
```



```
plt.figure(figsize=(12,6))
plt.plot(df['Lon'],df['Lat'],'r+',ms=0.5)
plt.xlim(-74.2,-73.7)
plt.ylim(40.6,41)
```

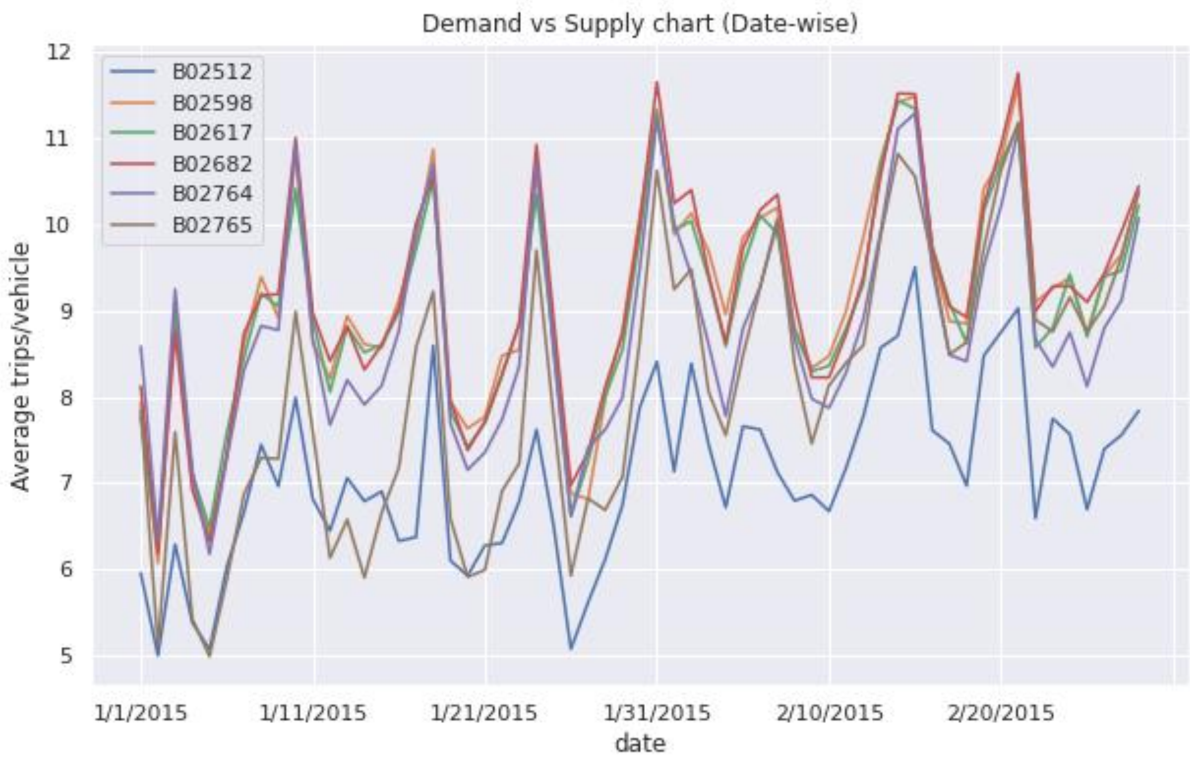
```
(40.6, 41.0)
```



```
df_out=df[df['weekday']=='Sunday']
df_out
```

	Date/Time	Lat	Lon	Base	DayOfWeekNum	MonthDayNum	HourOfDay	weekday	day	minute	month	hour
6265	2014-04-06 00:00:00	40.6547	-74.3033	B02512	6	6	0	Sunday	6	0	4	0
6266	2014-04-06 00:00:00	40.7358	-74.0008	B02512	6	6	0	Sunday	6	0	4	0
6267	2014-04-06 00:00:00	40.7421	-74.0041	B02512	6	6	0	Sunday	6	0	4	0
6268	2014-04-06 00:00:00	40.7401	-74.0053	B02512	6	6	0	Sunday	6	0	4	0
6269	2014-04-06 00:01:00	40.7368	-73.9877	B02512	6	6	0	Sunday	6	1	4	0
...	...	...	...	...	...	...	...	...	...	...	...	...
1014138	2014-09-28 23:57:00	40.8447	-73.7821	B02764	6	28	23	Sunday	28	57	9	23
1014139	2014-09-28 23:57:00	40.7513	-73.9941	B02764	6	28	23	Sunday	28	57	9	23
1014140	2014-09-28 23:57:00	40.6875	-74.1824	B02764	6	28	23	Sunday	28	57	9	23
1014141	2014-09-28 23:57:00	40.6482	-73.7823	B02764	6	28	23	Sunday	28	57	9	23
1014142	2014-09-28 23:59:00	40.6483	-73.7824	B02764	6	28	23	Sunday	28	59	9	23

490180 rows × 12 columns



In [36]:
```python
from sklearn.linear_model import LinearRegression
# fit the model on the train dataset

model = LinearRegression()
model.fit(X_train, Y_train)
```

Out[36]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)
```

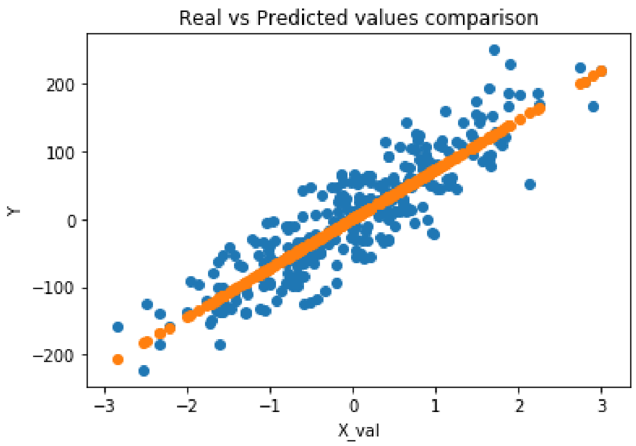
In [37]:
```python
# Predicting for the X_val points

Y_pred = model.predict(X_val)
```

In [38]:
```python
from sklearn.metrics import mean_squared_error
print(f'MSE on the validation set: {mean_squared_error(Y_val, Y_pred)}')
```

```
MSE on the validation set: 1515.7821465595791
```

In [39]:
```python
plt.xlabel('X_val')
plt.ylabel('Y')
plt.title('Real vs Predicted values comparison')

plt.scatter(X_val, Y_val)
plt.scatter(X_val, Y_pred)
```

Out[39]:
```
<matplotlib.collections.PathCollection at 0xf7cb427e80>
```



In [44]:
```python
X = df.iloc[:, 0].values.reshape(-1, 1)
Y = df.iloc[:, 1].values.reshape(-1, 1)
```

In [45]:
```python
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.25, random_state = 42, shuffle = Tru
e)
```

In [46]:
```python
regressor = LinearRegression()
regressor.fit(X_train, Y_train) #training the algorithm
#To retrieve the intercept:
print(regressor.intercept_)

#For retrieving the slope:
print(regressor.coef_)
```

```
[40.90744471]
[[-1.19688232e-19]]
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                      y   R-squared:                       0.990
Model:                            OLS   Adj. R-squared:                  0.990
Method:                 Least Squares   F-statistic:                     9798.
Date:                Sat, 31 Jul 2021   Prob (F-statistic):          5.01e-100
Time:                        08:40:14   Log-Likelihood:                 217.43
No. Observations:                 100   AIC:                            -430.9
Df Residuals:                      98   BIC:                            -425.7
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const          0.0547      0.006      9.154      0.000       0.043       0.067
x1             1.0020      0.010     98.985      0.000       0.982       1.022
==============================================================================
Omnibus:                       12.685   Durbin-Watson:                   2.215
Prob(Omnibus):                  0.002   Jarque-Bera (JB):                5.318
Skew:                          -0.311   Prob(JB):                       0.0700
Kurtosis:                       2.057   Cond. No.                         4.70
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

CONCLUSION:

The conclusion of the project is to project a basic outline of trips travelled with respect to latitude and longitude of locations and pinpoint the locations travelled with respect to the frequency of trips travelled by a uber cab during the day and also based on the cross analyzing of the dataset based on the latitude and longitude of the point travelled by the cab which is then analyzed by deploying k-means clustering which classifies the locations on the basis of centroids and then orders the frequency of trips based on labels or clusters. By the location given by the user, the algorithm predicts the cluster nearest to the location so that cab can be assigned to the user for pickup. The merit of the project is that it explains the functioning of how cabs are assigned to passengers based on an unsupervised algorithm and also explains the key concepts of machine learning. The limitations of the Project are that the algorithm deployed may be inefficient for huge data for over 10 years. The future work suggests that the system will provide the location to the user. The algorithm then records the time, latitude, longitude of the trip and assigns it to a cluster nearest to the passenger location where a cab is scheduled for pickup. We can also predict the passenger count on each district to deploy more cabs to the clustered coordinates using convolutional neural networks (CNN).