Final Project Report

April 15, 2020

0.0.1 Importing the libraries

[2]:
```
import pandas as pd # Pandas is used for data manipulation
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.pyplot import figure
import seaborn as sns
from sklearn.metrics.pairwise import haversine_distances
%matplotlib inline
import plotly.express as px
import folium
from folium import FeatureGroup, LayerControl, Map, Marker
from folium.plugins import HeatMap
from folium.plugins import TimestampedGeoJson
from folium.plugins import MarkerCluster

plt.style.use('seaborn-whitegrid')
```

[3]:
```
nyc = pd.read_csv('train.csv', nrows = 50000,parse_dates=["pickup_datetime"])

# Let's see data of first few rows of the dataset
nyc.head(10)
```

[3]:
```
                             key  fare_amount           pickup_datetime  \
0    2009-06-15 17:26:21.0000001          4.5 2009-06-15 17:26:21+00:00
1    2010-01-05 16:52:16.0000002         16.9 2010-01-05 16:52:16+00:00
2   2011-08-18 00:35:00.00000049          5.7 2011-08-18 00:35:00+00:00
3    2012-04-21 04:30:42.0000001          7.7 2012-04-21 04:30:42+00:00
4  2010-03-09 07:51:00.000000135          5.3 2010-03-09 07:51:00+00:00
5    2011-01-06 09:50:45.0000002         12.1 2011-01-06 09:50:45+00:00
6    2012-11-20 20:35:00.0000001          7.5 2012-11-20 20:35:00+00:00
7   2012-01-04 17:22:00.00000081         16.5 2012-01-04 17:22:00+00:00
8  2012-12-03 13:10:00.000000125          9.0 2012-12-03 13:10:00+00:00
9   2009-09-02 01:11:00.00000083          8.9 2009-09-02 01:11:00+00:00

   pickup_longitude  pickup_latitude  dropoff_longitude  dropoff_latitude  \
0        -73.844311        40.721319         -73.841610         40.712278
1        -74.016048        40.711303         -73.979268         40.782004
```

```
2        -73.982738        40.761270        -73.991242         40.750562
3        -73.987130        40.733143        -73.991567         40.758092
4        -73.968095        40.768008        -73.956655         40.783762
5        -74.000964        40.731630        -73.972892         40.758233
6        -73.980002        40.751662        -73.973802         40.764842
7        -73.951300        40.774138        -73.990095         40.751048
8        -74.006462        40.726713        -73.993078         40.731628
9        -73.980658        40.733873        -73.991540         40.758138

   passenger_count
0                1
1                1
2                2
3                1
4                1
5                1
6                1
7                1
8                1
9                2
```

0.1 Data Exploration

Checking the datatypes of the features of the dataset

[4]:
```
nyc.dtypes
```

[4]:
```
key                                  object
fare_amount                         float64
pickup_datetime      datetime64[ns, UTC]
pickup_longitude                    float64
pickup_latitude                     float64
dropoff_longitude                   float64
dropoff_latitude                    float64
passenger_count                       int64
dtype: object
```

Checking the statistics of the data

[5]:
```
nyc.describe()
```

[5]:
```
        fare_amount  pickup_longitude  pickup_latitude  dropoff_longitude  \
count  50000.000000      50000.000000     50000.000000       50000.000000
mean      11.364171        -72.509756        39.933759         -72.504616
std        9.685557         10.393860         6.224857          10.407570
min       -5.000000        -75.423848       -74.006893         -84.654241
25%        6.000000        -73.992062        40.734880         -73.991152
50%        8.500000        -73.981840        40.752678         -73.980082
```

```
75%       12.500000        -73.967148        40.767360          -73.963584
max      200.000000         40.783472       401.083332           40.851027

       dropoff_latitude  passenger_count
count      50000.000000     50000.000000
mean          39.926251         1.667840
std            6.014737         1.289195
min          -74.006377         0.000000
25%           40.734371         1.000000
50%           40.753372         1.000000
75%           40.768167         2.000000
max           43.415190         6.000000
```

0.1.1 From the statistical summary we can conclude these points :

- The fare amount has a minimum value of -44.9, which cannot be true. The base fare for New York City is \$2.50.So, We will be removing records where the fare is less than \$2.50.
- The minimum value of passenger count is zero. This is not possible.
- The longitude and latitude values are totally different as we can see it from the maximum and minimum values of pickup_latitude and dropoff_longitude.

[6]:
```
nyc = nyc[nyc['fare_amount']>2.50]
```

[7]:
```
nyc = nyc[nyc['passenger_count']>0]
```

[8]:
```
nyc.shape
```

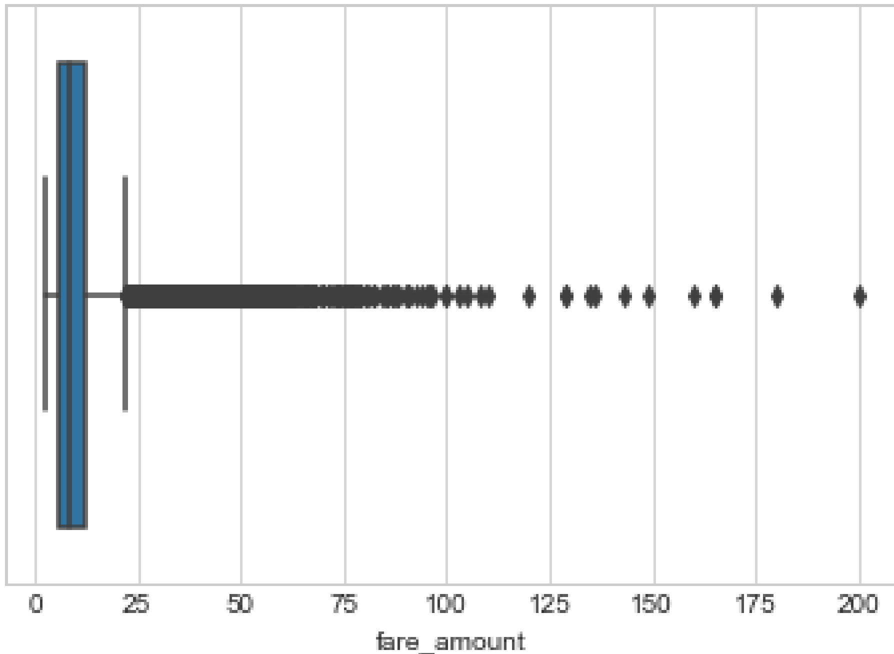
[8]:
```
(49623, 8)
```

[9]:
```
#Now checking for missing data
nyc.isnull().sum()
```

[9]:
```
key                  0
fare_amount          0
pickup_datetime      0
pickup_longitude     0
pickup_latitude      0
dropoff_longitude    0
dropoff_latitude     0
passenger_count      0
dtype: int64
```

[10]:
```
sns.boxplot(nyc['fare_amount'])
```

[10]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1c2eb0f250>
```



From the boxplot we can see that there are a lot of outliers. We will be removing them in the below steps.

[11]:
```
# Calculating the mean and the standard deviation of the 'fare_amount' in the
↪dataset.
mean_df = np.mean(nyc.fare_amount)
std_df = np.std(nyc.fare_amount)
# Filtering the rows of from outliers
nyc = nyc[(nyc.fare_amount > (mean_df - 3*std_df)) & (nyc.fare_amount <
↪(mean_df + 3*std_df))]
nyc.head()
```

[11]:
```
                             key  fare_amount           pickup_datetime  \
0    2009-06-15 17:26:21.0000001          4.5 2009-06-15 17:26:21+00:00
1    2010-01-05 16:52:16.0000002         16.9 2010-01-05 16:52:16+00:00
2   2011-08-18 00:35:00.00000049          5.7 2011-08-18 00:35:00+00:00
3    2012-04-21 04:30:42.0000001          7.7 2012-04-21 04:30:42+00:00
4  2010-03-09 07:51:00.000000135          5.3 2010-03-09 07:51:00+00:00

   pickup_longitude  pickup_latitude  dropoff_longitude  dropoff_latitude  \
0        -73.844311        40.721319         -73.841610         40.712278
1        -74.016048        40.711303         -73.979268         40.782004
2        -73.982738        40.761270         -73.991242         40.750562
3        -73.987130        40.733143         -73.991567         40.758092
4        -73.968095        40.768008         -73.956655         40.783762
```

```
   passenger_count
0                1
1                1
2                2
3                1
4                1
```

0.1.2 Location Data

New York city coordinates are (https://www.travelmath.com/cities/New+York,+NY):

- longitude = -74.3 to -72.9
- lattitude = 40.5 to 41.8

We will be deleting all records where th pickup as well as dropoff longitude and latitude doesn't lie in between the above values.

[12]:
```python
nyc = nyc[((nyc['pickup_longitude'] >= -74.3)
             & (nyc['pickup_longitude'] <= -72.9))
            & ((nyc['dropoff_longitude'] >= -74.3)
               & (nyc['dropoff_longitude'] <= -72.9))
            & ((nyc['pickup_latitude'] >= 40.5)
               & (nyc['pickup_latitude'] <= 41.8))
            & ((nyc['dropoff_latitude'] >= 40.5)
               & (nyc['dropoff_latitude'] <= 41.8))]
```

[13]:
```python
nyc.describe()
```

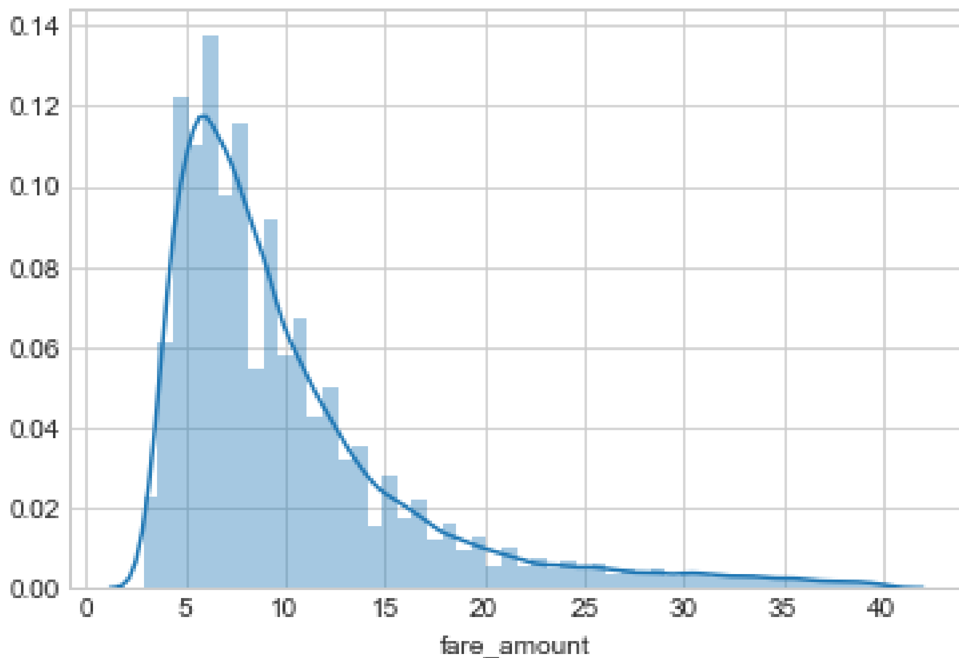
[13]:
```
       fare_amount  pickup_longitude  pickup_latitude  dropoff_longitude  \
count  47278.00000      47278.000000     47278.000000       47278.000000
mean      10.23105        -73.977579        40.752252         -73.975356
std        6.42082          0.031571         0.027737           0.031947
min        2.90000        -74.290833        40.522263         -74.294613
25%        6.00000        -73.992431        40.737252         -73.991348
50%        8.50000        -73.982264        40.753740         -73.980629
75%       12.10000        -73.969241        40.767897         -73.966042
max       40.33000        -73.137393        41.650000         -73.137393

       dropoff_latitude  passenger_count
count      47278.000000     47278.000000
mean          40.752602         1.674034
std            0.030800         1.288916
min           40.531637         1.000000
25%           40.736804         1.000000
50%           40.754457         1.000000
75%           40.768684         2.000000
max           41.543217         6.000000
```

[14]:
```
nyc.shape
```

[14]: (47278, 8)

[15]:
```
sns.distplot(nyc.fare_amount)
```

[15]: <matplotlib.axes._subplots.AxesSubplot at 0x1c2cbf8410>



Calculating the distance Since, we need to calculate the distance between two points where their latitude and longitude points are given, we will be using the haversine formula. The haversine formula determines the great-circle distance between two points on a sphere given their longitudes and latitudes.

[16]:
```
#To calculate the distance in miles we use a formula called "HAVERSINE FORMULA"

def distance(lat1, lon1, lat2, lon2):
    p = 0.017453292519943295 # Pi/180
    a = 0.5 - np.cos((lat2 - lat1) * p)/2 + np.cos(lat1 * p) * np.cos(lat2 * p)␣
 ↪* (1 - np.cos((lon2 - lon1) * p)) / 2
    return 0.6213712 * 12742 * np.arcsin(np.sqrt(a))
```

Adding a new distance column to dataframe storing the haversine distance of the corresponding trips

[17]:
```
nyc['distance'] = distance(nyc.pickup_latitude, nyc.pickup_longitude, \
                                      nyc.dropoff_latitude, nyc.
 ↪dropoff_longitude)
nyc.distance.describe()
```

[17]:
```
count    47278.000000
mean         1.858303
std          1.835652
min          0.000000
25%          0.781311
50%          1.313407
75%          2.324054
max         62.203770
Name: distance, dtype: float64
```

Adding Year, Month and Day and time as a separate column

[18]:
```
nyc['year'] = nyc.pickup_datetime.apply(lambda x : x.year)
nyc['month'] = nyc.pickup_datetime.apply(lambda x : x.month)
nyc['day'] = nyc.pickup_datetime.apply(lambda x : x.day)
nyc['time'] = nyc.pickup_datetime.apply(lambda x : x.time)
nyc['hour'] = nyc.time.apply(lambda x : x.hour)
nyc.head()
```

[18]:
```
                             key  fare_amount           pickup_datetime  \
0    2009-06-15 17:26:21.0000001          4.5 2009-06-15 17:26:21+00:00
1    2010-01-05 16:52:16.0000002         16.9 2010-01-05 16:52:16+00:00
2   2011-08-18 00:35:00.00000049          5.7 2011-08-18 00:35:00+00:00
3    2012-04-21 04:30:42.0000001          7.7 2012-04-21 04:30:42+00:00
4  2010-03-09 07:51:00.000000135          5.3 2010-03-09 07:51:00+00:00

   pickup_longitude  pickup_latitude  dropoff_longitude  dropoff_latitude  \
0        -73.844311        40.721319         -73.841610         40.712278
1        -74.016048        40.711303         -73.979268         40.782004
2        -73.982738        40.761270         -73.991242         40.750562
3        -73.987130        40.733143         -73.991567         40.758092
4        -73.968095        40.768008         -73.956655         40.783762

   passenger_count  distance  year  month  day      time  hour
0                1  0.640487  2009      6   15  17:26:21    17
1                1  5.250670  2010      1    5  16:52:16    16
2                2  0.863411  2011      8   18  00:35:00     0
3                1  1.739386  2012      4   21  04:30:42     4
4                1  1.242218  2010      3    9  07:51:00     7
```

[19]:
```
nyc_boroughs={
    'manhattan':{
```

```
        'min_lng':-74.0479,
        'min_lat':40.6829,
        'max_lng':-73.9067,
        'max_lat':40.8820
    },

    'queens':{
        'min_lng':-73.9630,
        'min_lat':40.5431,
        'max_lng':-73.7004,
        'max_lat':40.8007

    },

    'brooklyn':{
        'min_lng':-74.0421,
        'min_lat':40.5707,
        'max_lng':-73.8334,
        'max_lat':40.7395

    },

    'bronx':{
        'min_lng':-73.9339,
        'min_lat':40.7855,
        'max_lng':-73.7654,
        'max_lat':40.9176

    },

    'staten_island':{
        'min_lng':-74.2558,
        'min_lat':40.4960,
        'max_lng':-74.0522,
        'max_lat':40.6490

    }



}
```

[20]:

```
def getBorough(lat,lng):

    locs=nyc_boroughs.keys()
    for loc in locs:
```

```
        if lat>=nyc_boroughs[loc]['min_lat'] and␣
↪lat<=nyc_boroughs[loc]['max_lat'] and lng>=nyc_boroughs[loc]['min_lng'] and␣
↪lng<=nyc_boroughs[loc]['max_lng']:
            return loc
    return 'others'
```

[21]:
```
nyc['pickup_borough']=nyc.apply(lambda row:
↪getBorough(row['pickup_latitude'],row['pickup_longitude']),axis=1)
nyc['dropoff_borough']=nyc.apply(lambda row:
↪getBorough(row['dropoff_latitude'],row['dropoff_longitude']),axis=1)
```

[68]:
```
nyc.head()
```

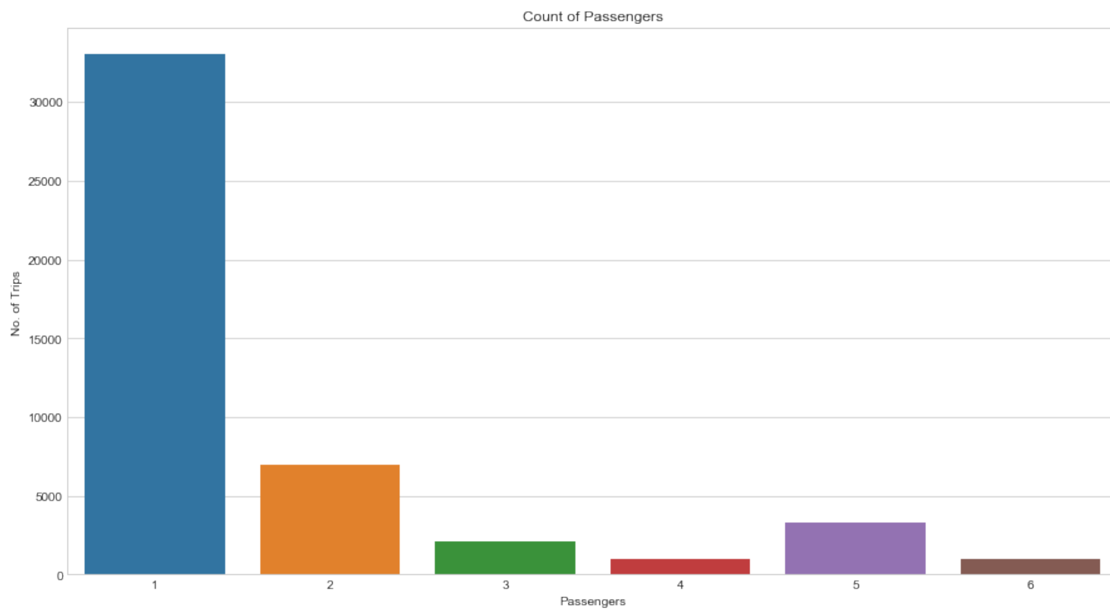
[68]:
```
                              key  fare_amount          pickup_datetime  \
0    2009-06-15 17:26:21.0000001          4.5 2009-06-15 17:26:21+00:00
1    2010-01-05 16:52:16.0000002         16.9 2010-01-05 16:52:16+00:00
2   2011-08-18 00:35:00.00000049          5.7 2011-08-18 00:35:00+00:00
3    2012-04-21 04:30:42.0000001          7.7 2012-04-21 04:30:42+00:00
4  2010-03-09 07:51:00.000000135          5.3 2010-03-09 07:51:00+00:00

   pickup_longitude  pickup_latitude  dropoff_longitude  dropoff_latitude  \
0        -73.844311        40.721319         -73.841610         40.712278
1        -74.016048        40.711303         -73.979268         40.782004
2        -73.982738        40.761270         -73.991242         40.750562
3        -73.987130        40.733143         -73.991567         40.758092
4        -73.968095        40.768008         -73.956655         40.783762

   passenger_count  distance  year  month  day      time  hour pickup_borough  \
0                1  0.640487  2009      6   15  17:26:21    17         queens
1                1  5.250670  2010      1    5  16:52:16    16      manhattan
2                2  0.863411  2011      8   18  00:35:00     0      manhattan
3                1  1.739386  2012      4   21  04:30:42     4      manhattan
4                1  1.242218  2010      3    9  07:51:00     7      manhattan

  dropoff_borough
0          queens
1       manhattan
2       manhattan
3       manhattan
4       manhattan
```

0.2 Data Exploration

[22]:
```
pass_count = nyc.groupby('passenger_count').count()
plt.subplots(figsize=(15,8))
sns.barplot(pass_count.index,pass_count.key)
```

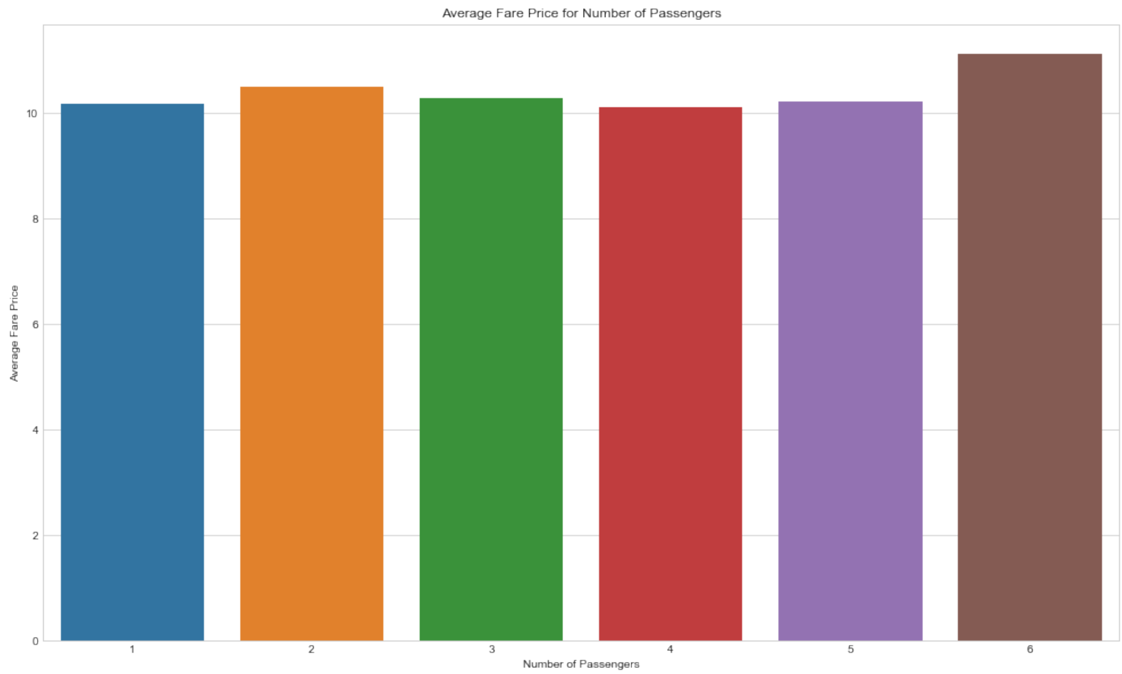
```
plt.xlabel('Passengers')
plt.ylabel('No. of Trips')
plt.title('Count of Passengers')
```

[22]: Text(0.5, 1.0, 'Count of Passengers')



From the graph we can see that no. of trips having a passenger count of 1 exceeds 30000, which accounts for more than 60% of the trips recorded in the given dataset.

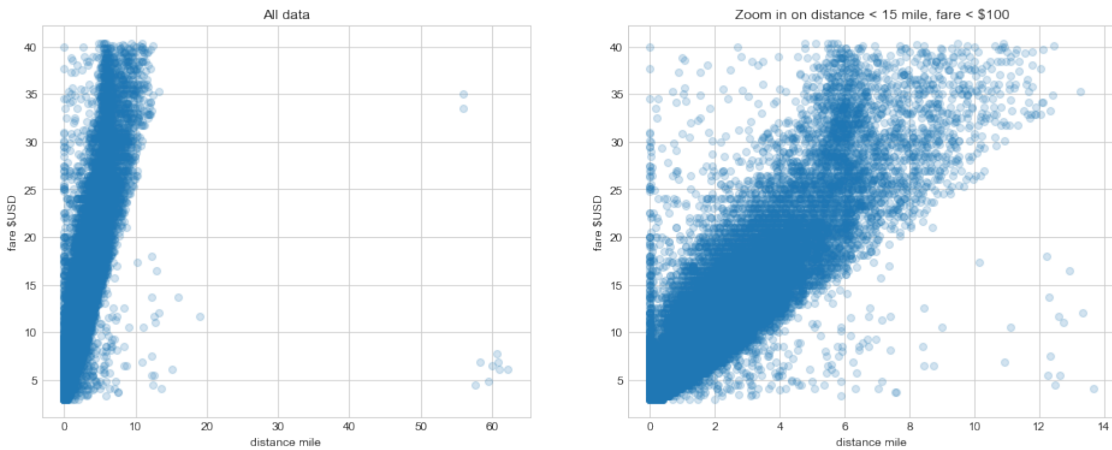
[23]:
```
passenger_fare = nyc.groupby(['passenger_count']).mean()

fig, ax = plt.subplots(figsize=(17,10))

sns.barplot(passenger_fare.index, passenger_fare['fare_amount'])
plt.xlabel('Number of Passengers')
plt.ylabel('Average Fare Price')
plt.title('Average Fare Price for Number of Passengers')
plt.show()
```



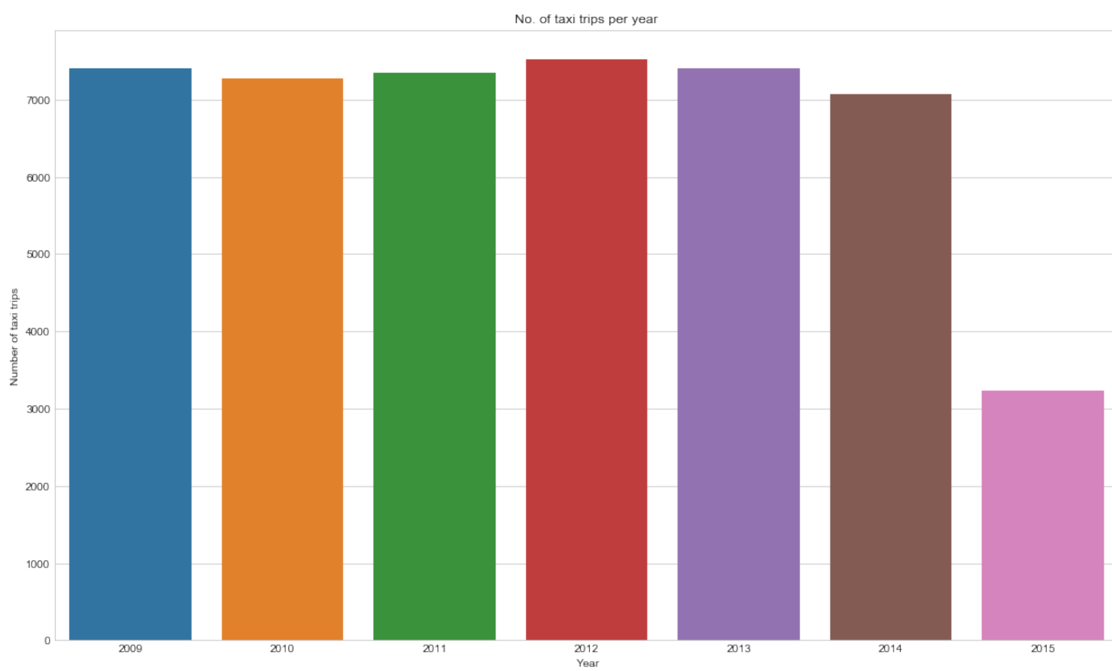
[24]:
```
print("Average ride cost in USD/ : {}".format(nyc.fare_amount.sum()/
 ↪nyc["distance"].sum()))
```

```
Average ride cost in USD/ : 5.5055888146580765
```

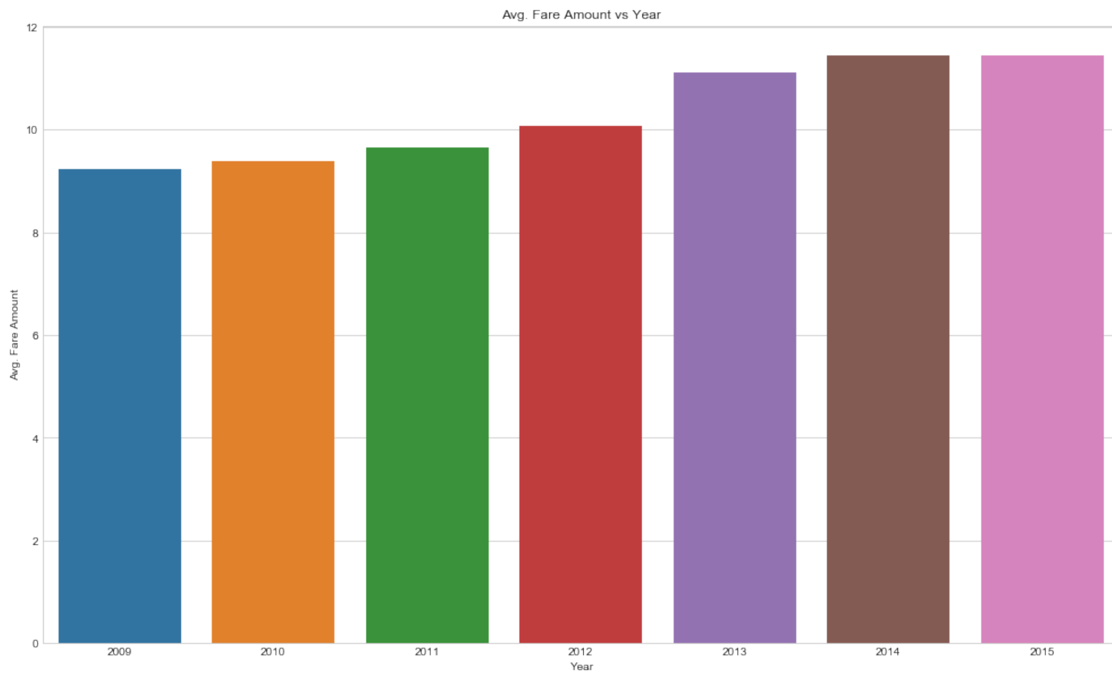
[25]:
```
#Scatterplot of distance-Fare

fig, axs = plt.subplots(1, 2, figsize=(16,6))
axs[0].scatter(nyc.distance, nyc.fare_amount, alpha=0.2)
axs[0].set_xlabel('distance mile')
axs[0].set_ylabel('fare $USD')
axs[0].set_title('All data')

# zoom in on part of data
idx = (nyc.distance < 15)
axs[1].scatter(nyc[idx].distance, nyc[idx].fare_amount, alpha=0.2)
axs[1].set_xlabel('distance mile')
axs[1].set_ylabel('fare $USD')
axs[1].set_title('Zoom in on distance < 15 mile, fare < $100');
```



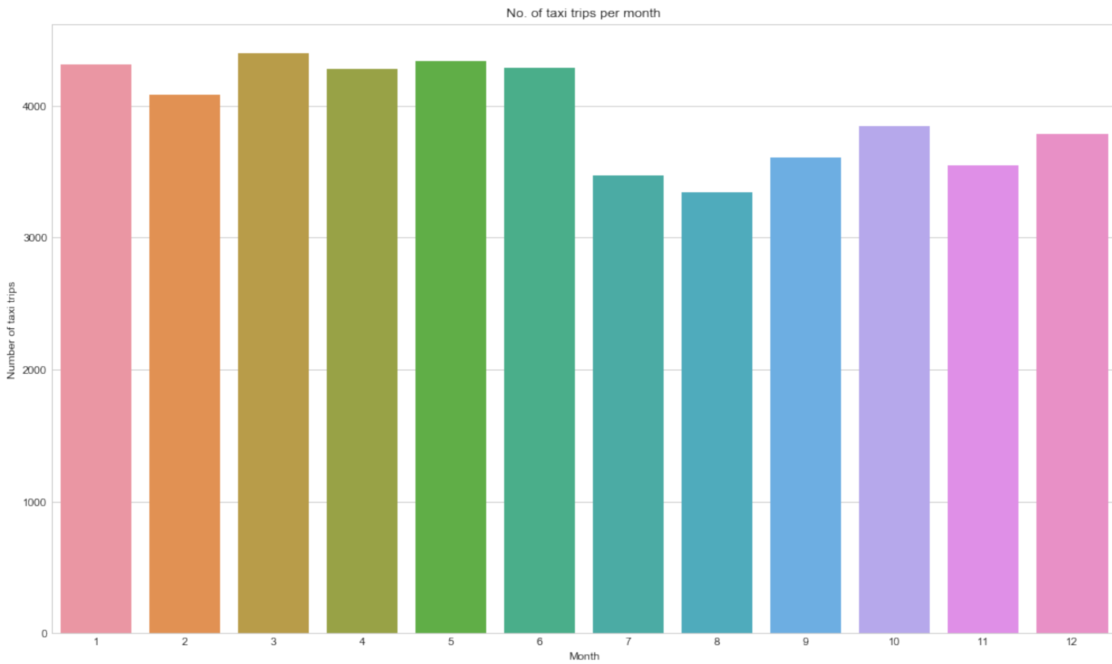
[26]:
```
fig, ax = plt.subplots(figsize=(17,10))
year_count = nyc.groupby('year').count()
sns.barplot(year_count.index,year_count.key)
plt.xlabel('Year')
plt.ylabel('Number of taxi trips')
plt.title('No. of taxi trips per year')
plt.show()
```



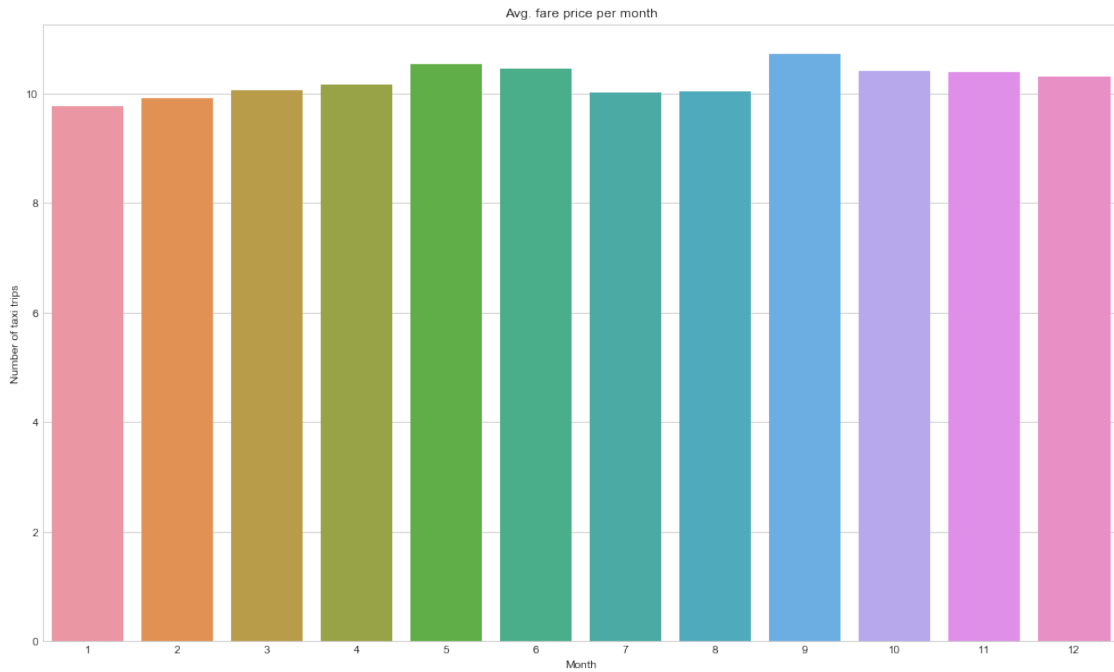
[27]:
```
fig, ax = plt.subplots(figsize=(17,10))
avg_fare_years = nyc.groupby('year').mean()
avg_fare_years.head()
sns.barplot(avg_fare_years.index,avg_fare_years.fare_amount)
plt.xlabel('Year')
plt.ylabel('Avg. Fare Amount')
plt.title('Avg. Fare Amount vs Year')
plt.show()
```



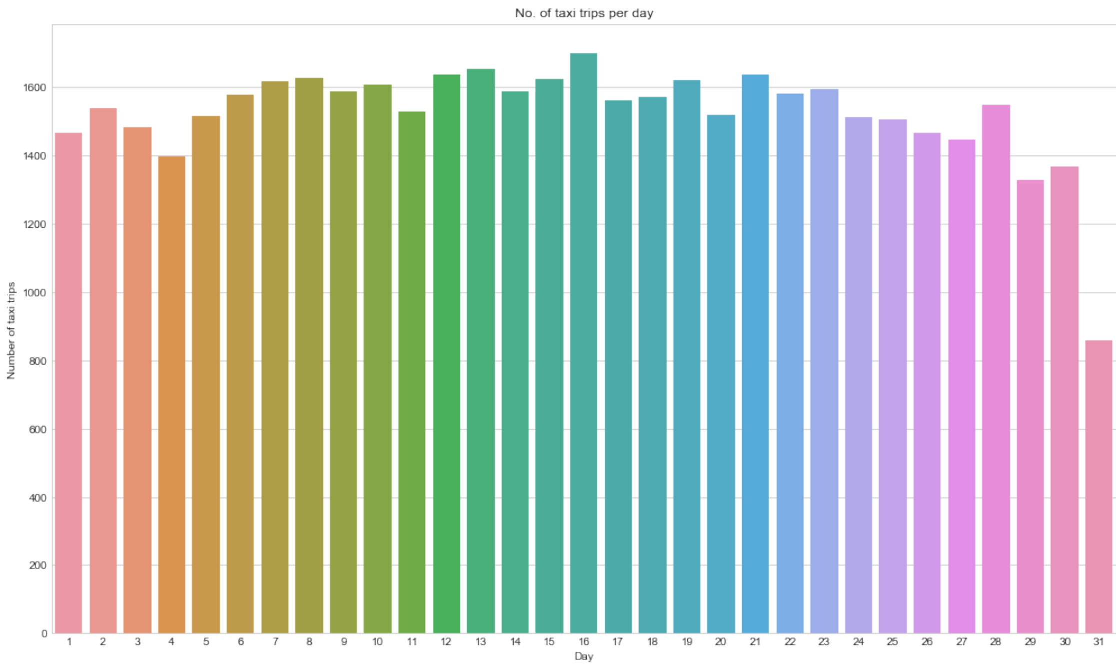
[28]:
```
fig, ax = plt.subplots(figsize=(17,10))
month_count = nyc.groupby('month').count()
sns.barplot(month_count.index,month_count.key)
plt.xlabel('Month')
plt.ylabel('Number of taxi trips')
plt.title('No. of taxi trips per month')
plt.show()
```



[29]:
```
fig, ax = plt.subplots(figsize=(17,10))
month_mean = nyc.groupby('month').mean()
sns.barplot(month_mean.index,month_mean.fare_amount)
plt.xlabel('Month')
plt.ylabel('Avg. Fare Price')
plt.title('Avg. fare price per month')
plt.show()
```

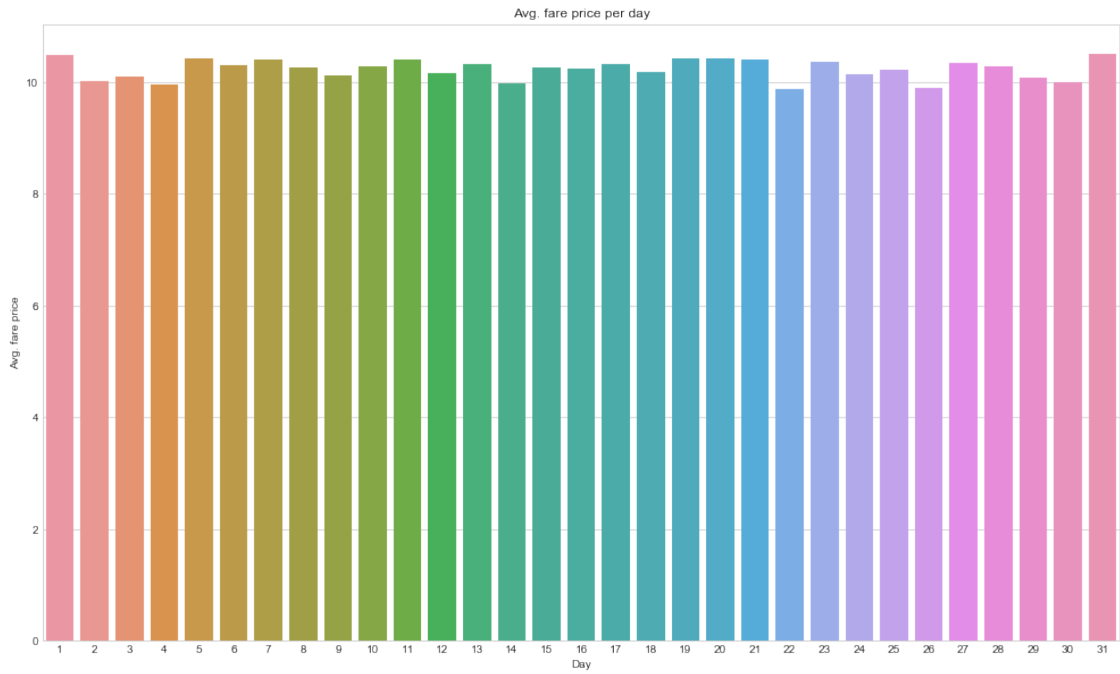


[30]:
```
fig, ax = plt.subplots(figsize=(17,10))
day_count = nyc.groupby('day').count()
sns.barplot(day_count.index,day_count.key)
plt.xlabel('Day')
plt.ylabel('Number of taxi trips')
plt.title('No. of taxi trips per day')
plt.show()
```

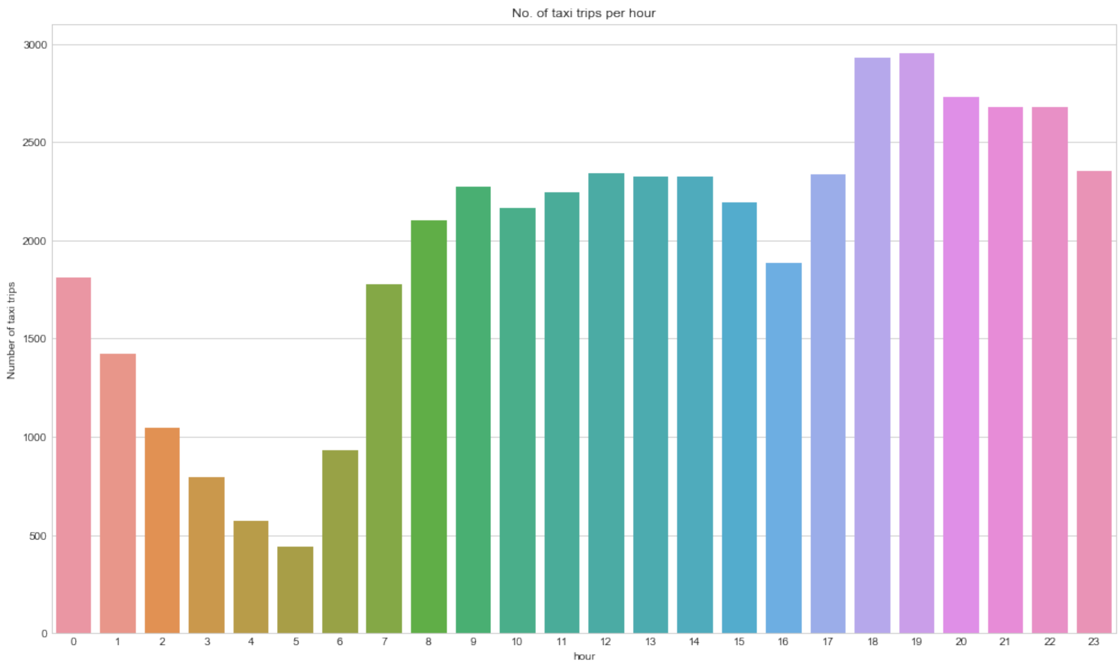


[63]:

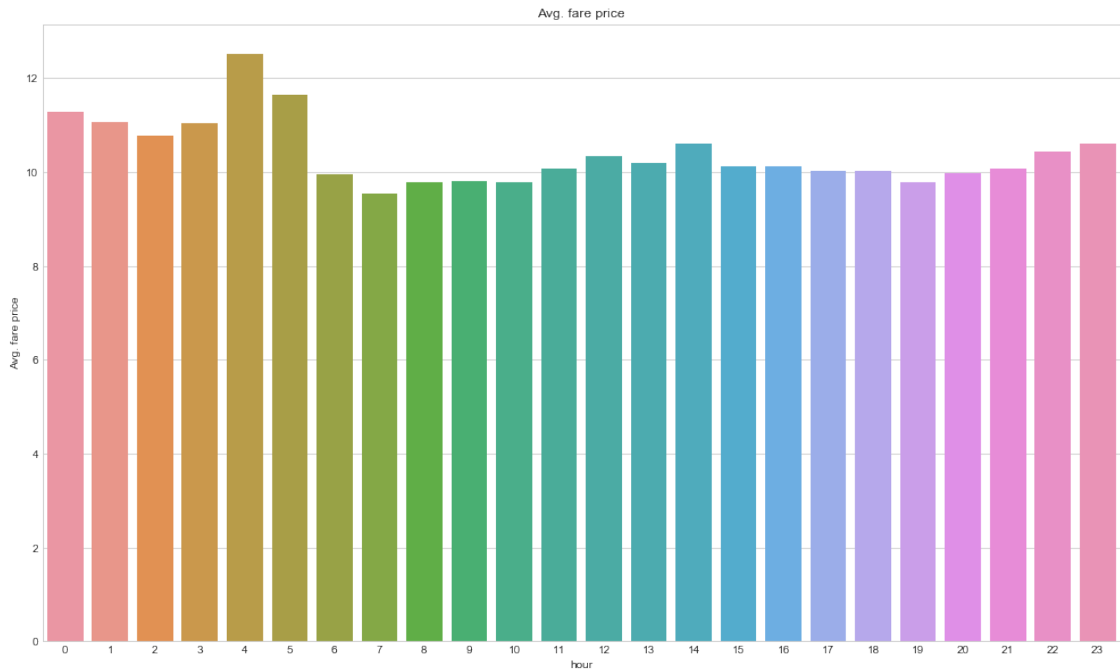
```
fig, ax = plt.subplots(figsize=(17,10))
day_mean = nyc.groupby('day').mean()
sns.barplot(day_mean.index,day_mean.fare_amount)
plt.xlabel('Day')
plt.ylabel('Avg. fare price')
plt.title('Avg. fare price per day')
plt.show()
```



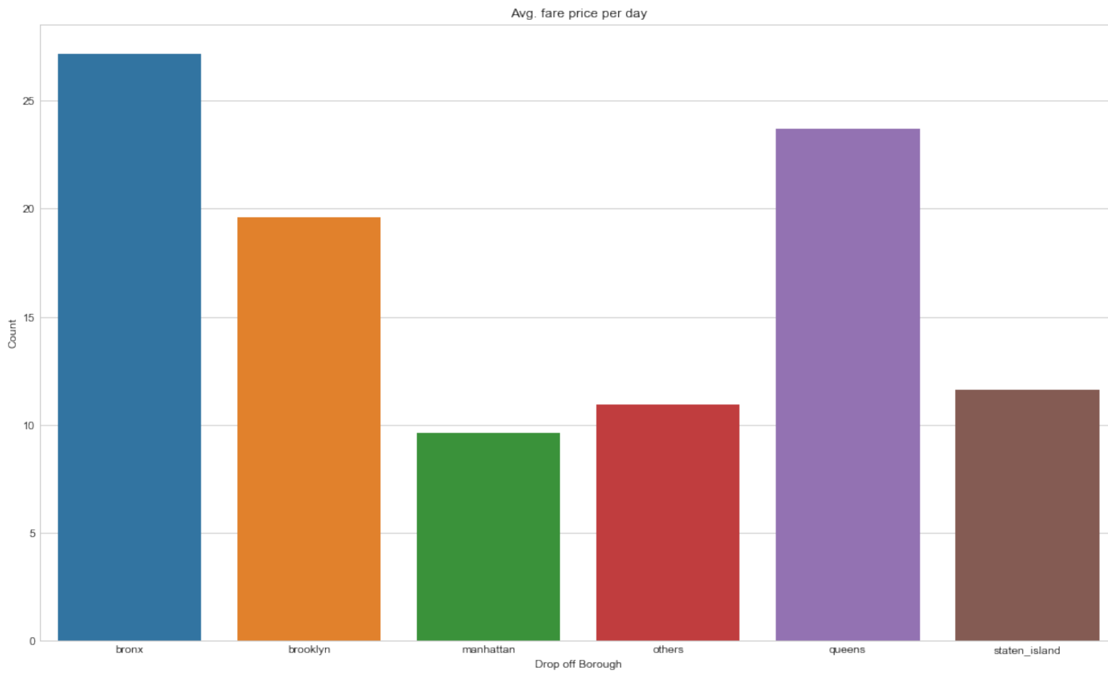
[65]:
```
fig, ax = plt.subplots(figsize=(17,10))
hour_count = nyc.groupby('hour').count()
sns.barplot(hour_count.index,hour_count.key)
plt.xlabel('hour')
plt.ylabel('Number of taxi trips')
plt.title('No. of taxi trips per hour')
plt.show()
```



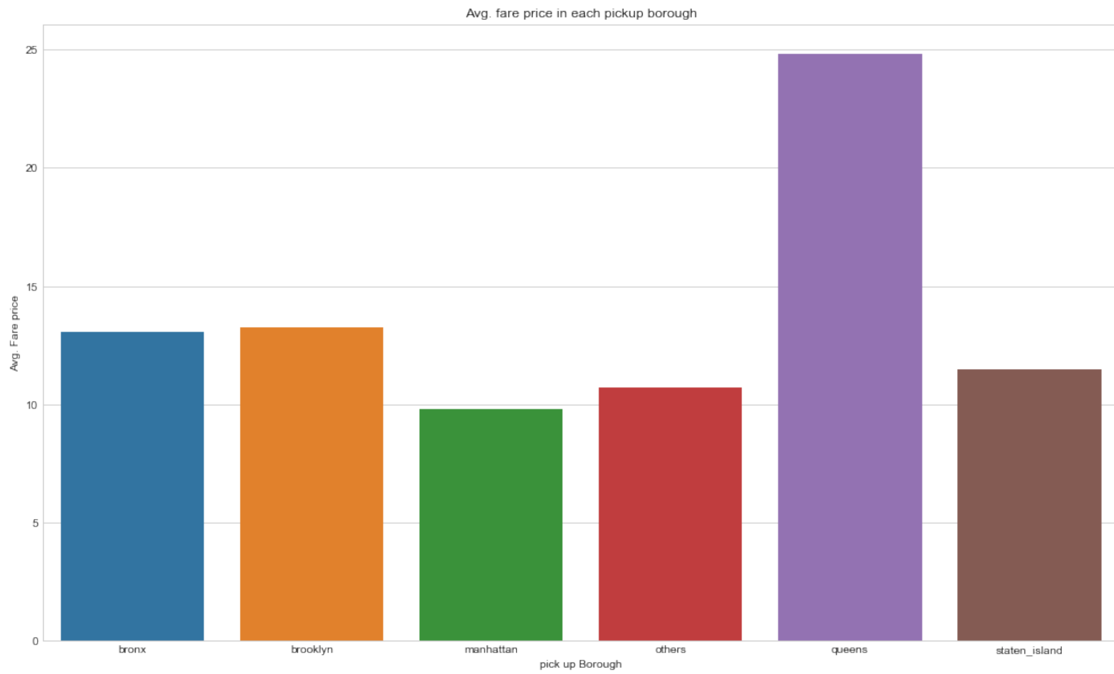
[67]:
```
fig, ax = plt.subplots(figsize=(17,10))
hour_fare = nyc.groupby('hour').mean()
sns.barplot(hour_fare.index,hour_fare.fare_amount)
plt.xlabel('hour')
plt.ylabel('Avg. fare price')
plt.title('Avg. fare price')
plt.show()
```



[55]:
```
fig, ax = plt.subplots(figsize=(17,10))
dropborofare = nyc.groupby('dropoff_borough').mean()
sns.barplot(dropborofare.index,dropborofare.fare_amount)
plt.xlabel('Drop off Borough')
plt.ylabel('Avg. Fare price')
plt.title('Avg. fare price in each dropoff borough')
plt.show()
```




```
[56]: fig, ax = plt.subplots(figsize=(17,10))
      pickborofare = nyc.groupby('pickup_borough').mean()
      sns.barplot(pickborofare.index,pickborofare.fare_amount)
      plt.xlabel('pick up Borough')
      plt.ylabel('Avg. Fare price')
      plt.title('Avg. fare price in each pickup borough')
      plt.show()
```



0.2.1 Plot Heatmap of Pickups and Dropoffs within NYC

[34]:
```
import plotly
import chart_studio.plotly as py
import plotly.offline as offline
import plotly.graph_objs as go
from plotly.offline import download_plotlyjs, init_notebook_mode, plot, iplot
init_notebook_mode(connected=True)
import cufflinks as cf
from plotly.graph_objs import Scatter, Figure, Layout
cf.set_config_file(offline=True)
```

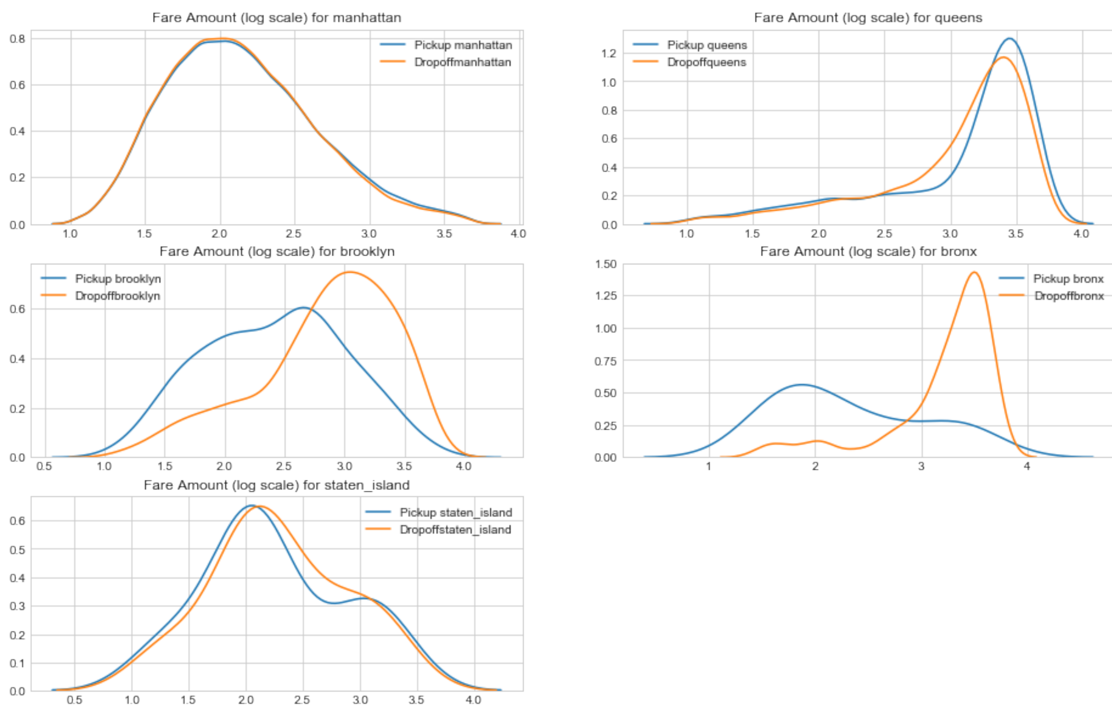
[59]:
```
pickuplocation = [go.Scattermapbox(
            lat= nyc['pickup_latitude'] ,
            lon= nyc['pickup_longitude'],
            customdata = nyc['key'],
            mode='markers',
            marker=dict(
                size= 5,
                color = 'red',
                opacity = .2,
            ),
          )]
layoutpan = go.Layout(autosize=False,
```

```
                   mapbox= dict(accesstoken="pk.
↪eyJ1Ijoic2hhejEzIiwiYSI6ImNqYXA3NjhmeDR4d3Iyd2w5M2phM3E2djQifQ.
↪yyxsAzT94VGYYEEOhxy87w",
                                bearing=10,
                                pitch=10,
                                zoom=13,
                                center= dict(
                                         lat=40.721319,
                                         lon=-73.987130),
                                style= "mapbox://styles/mapbox/streets-v11"),
                    width=800,
                    height=700, title = " Customer Pickup Visualization in NYC")
figure = dict(data=pickuplocation, layout=layoutpan)
iplot(figure)
```

[57]:
```
#Now we visualize the dropoff locations of customers in NYC



dropofflocation = [go.Scattermapbox(
            lat= nyc['dropoff_latitude'] ,
            lon= nyc['dropoff_longitude'],
            customdata = nyc['key'],
            mode='markers',
            marker=dict(
                size= 5,
                color = 'green',
                opacity = .2,
            ),
          )]
layoutpan = go.Layout(autosize=False,
                   mapbox= dict(accesstoken="pk.
↪eyJ1Ijoic2hhejEzIiwiYSI6ImNqYXA3NjhmeDR4d3Iyd2w5M2phM3E2djQifQ.
↪yyxsAzT94VGYYEEOhxy87w",
                                bearing=10,
                                pitch=5,
                                zoom=10,
                                center= dict(
                                         lat=40.721319,
                                         lon=-73.987130),
                                style= "mapbox://styles/mapbox/streets-v11"),
                    width=900,
                    height=700, title = "Customer Dropoff Visualization in NYC")
figure = dict(data=dropofflocation, layout=layoutpan)
iplot(figure)
```

[37]:

```
plt.figure(figsize=(16,10))
plt.title("Distribution of Fare Amount Across Buroughs")
i=1
for key in nyc_boroughs.keys():
    plt.subplot(3,2,i)
    sns.kdeplot(np.log(nyc.loc[nyc['pickup_borough']==key,'fare_amount'].
 ↪values),label='Pickup '+ key)
    sns.kdeplot(np.log(nyc.loc[nyc['dropoff_borough']==key,'fare_amount'].
 ↪values),label='Dropoff'+ key).set_title("Fare Amount (log scale) for "+key)

    i=i+1
```



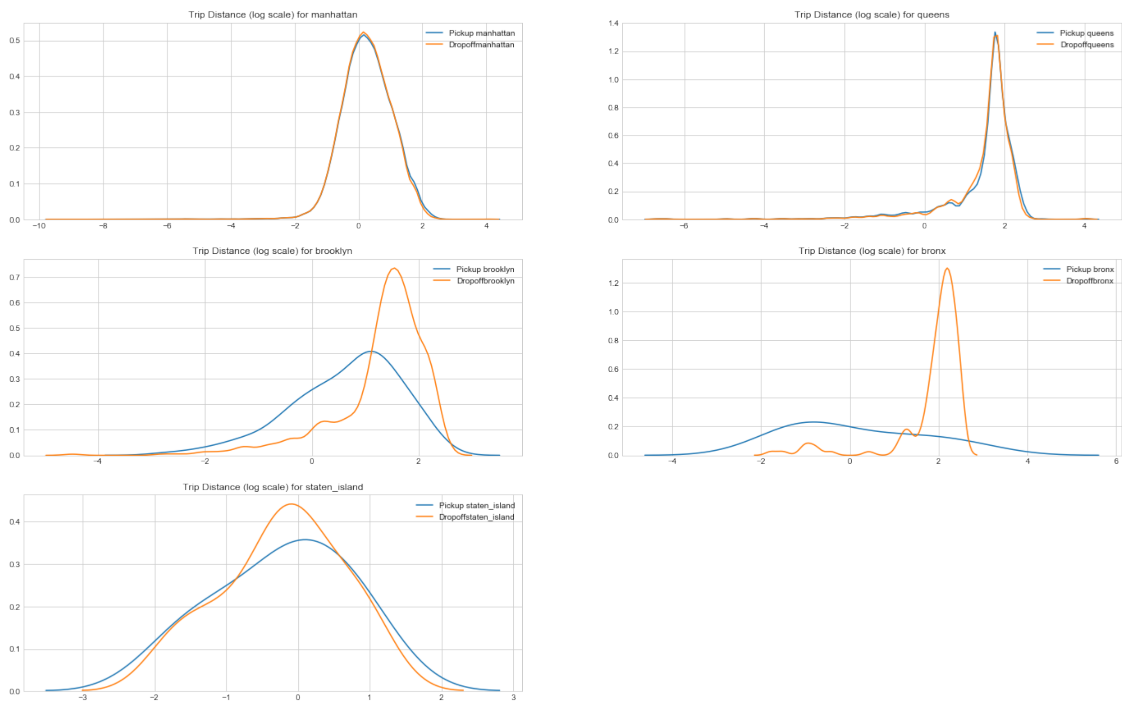
There is a significant difference in pickups and dropoffs fare amount for each burough except Manhattan.

[38]:

```
plt.figure(figsize=(24,15))
plt.title("Distribution of Trip Distances Across Buroughs")
i=1
for key in nyc_boroughs.keys():
    plt.subplot(3,2,i)
    sns.kdeplot(np.log(nyc.loc[nyc['pickup_borough']==key,'distance'].
 ↪values),label='Pickup '+ key)
    sns.kdeplot(np.log(nyc.loc[nyc['dropoff_borough']==key,'distance'].
 ↪values),label='Dropoff'+ key).set_title("Trip Distance (log scale) for "+key)
```

```
    i=i+1
```

```
/Users/ashwinjohnchempolil/opt/anaconda3/lib/python3.7/site-
packages/ipykernel_launcher.py:6: RuntimeWarning:

divide by zero encountered in log

/Users/ashwinjohnchempolil/opt/anaconda3/lib/python3.7/site-
packages/ipykernel_launcher.py:7: RuntimeWarning:

divide by zero encountered in log
```



Dropoffs to Bronx and Brooklyn are long trips.

0.3 Model Implementation

We will be implementing Multiple Linear Regression, Decision Trees, Random Forest and Boosted Trees.

Splitting the nyc data to train data as well as validation data

[39]:
```
# Labels are the values we want to predict
labels = np.array(nyc['fare_amount'])

# Remove the labels from the nyc
```

```
# axis 1 refers to the columns
features= nyc.drop(['fare_amount','key',␣
↪'pickup_datetime','time','pickup_borough','dropoff_borough'],axis = 1)
# Saving feature names for later use
feature_list = list(features.columns)
# Convert to numpy array
features = np.array(features)
```

[40]:
```
# Using Skicit-learn to split data into training and testing sets
from sklearn.model_selection import train_test_split

# Split the data into training and testing sets
train_nyc, valid_nyc, train_labels, valid_labels = train_test_split(features,␣
↪labels, test_size = 0.25, random_state = 42)

# Looking at the shape of the training data and validation data
print('Training Features Shape:', train_nyc.shape)
print('Training Labels Shape:', train_labels.shape)
print('Testing Features Shape:', valid_nyc.shape)
print('Testing Labels Shape:', valid_labels.shape)
```

```
Training Features Shape: (35458, 10)
Training Labels Shape: (35458,)
Testing Features Shape: (11820, 10)
Testing Labels Shape: (11820,)
```

0.3.1 Multiple Linear Regression

[41]:
```
# Importing the Linear Regression model from the sklearn
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error

lm = LinearRegression()
lm.fit(train_nyc,train_labels)
y_pred=np.round(lm.predict(valid_nyc),2)
lm_rmse=np.sqrt(mean_squared_error(y_pred, valid_labels))
lm_train_rmse=np.sqrt(mean_squared_error(lm.predict(train_nyc), train_labels))
lm_variance=abs(lm_train_rmse - lm_rmse)
print("Test RMSE for Linear Regression is ",lm_rmse)
print("Train RMSE for Linear Regression is ",lm_train_rmse)
print("Variance for Linear Regression is ",lm_variance)
```

```
Test RMSE for Linear Regression is  3.943219679175923
Train RMSE for Linear Regression is  3.913177184536855
Variance for Linear Regression is  0.030042494639068273
```

0.3.2 Establishing Baseline

0.3.3 Random Forest Regression

[42]:
```
# Importing the Random Forest from scikit learn package
from sklearn.ensemble import RandomForestRegressor
# Importing GridSearchCV which checks for the optimal n_estimator parameter
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import r2_score

est = range(50,100,50)
params_to_test = {'n_estimators': est}


rf = RandomForestRegressor(random_state = 101)

grid_search = GridSearchCV(rf, param_grid=params_to_test, cv=10,␣
↪scoring='neg_mean_squared_error')

grid_search.fit(train_nyc, train_labels)

best_model = grid_search.best_estimator_
```

[43]:
```
# Use the forest's predict method on the test data
predictions = best_model.predict(valid_nyc)
# Calculate the absolute errors
errors = abs(predictions - valid_labels)
# Print out the mean absolute error (mae)
print('Mean Absolute Error:', round(np.mean(errors), 2), 'degrees.')
```

```
Mean Absolute Error: 1.62 degrees.
```

[44]:
```
rf_rmse=np.sqrt(mean_squared_error(predictions, valid_labels))
rf_train_rmse=np.sqrt(mean_squared_error(best_model.predict(train_nyc),␣
↪train_labels))
rf_variance=abs(rf_train_rmse - rf_rmse)
print("Test RMSE for Random Forest Regression is ",rf_rmse)
print("Train RMSE for Random Forest Regression is ",rf_train_rmse)
print("Variance for Random Forest Regression is ",rf_variance)
```

```
Test RMSE for Random Forest Regression is  2.4954782376399693
Train RMSE for Random Forest Regression is  1.007790536555357
Variance for Random Forest Regression is  1.4876877010846123
```

0.3.4 Gradient Boosting Regression

[45]:
```
from sklearn.ensemble import GradientBoostingRegressor

est = range(50,100,50)
params_to_test = {'n_estimators': est}

gb = GradientBoostingRegressor(learning_rate=1, max_depth=3, random_state = 1)

grid_search_gb = GridSearchCV(gb, param_grid=params_to_test, cv=10,␣
↪scoring='neg_mean_squared_error')

grid_search_gb.fit(train_nyc,train_labels)

best_model_gb = grid_search_gb.best_estimator_
```

[46]:
```
# Use the forest's predict method on the test data
predictions_gb = best_model_gb.predict(valid_nyc)
# Calculate the absolute errors
errors_gb = abs(predictions_gb - valid_labels)
# Print out the mean absolute error (mae)
print('Mean Absolute Error:', round(np.mean(errors_gb), 2), 'degrees.')
```

```
Mean Absolute Error: 1.67 degrees.
```

[47]:
```
gb_rmse=np.sqrt(mean_squared_error(predictions_gb, valid_labels))
gb_train_rmse=np.sqrt(mean_squared_error(best_model_gb.predict(train_nyc),␣
↪train_labels))
gb_variance=abs(gb_train_rmse - gb_rmse)
print("Test RMSE for Gradient Boost Regression is ",gb_rmse)
print("Train RMSE for Gradient Boost Regression is ",gb_train_rmse)
print("Variance for Gradient Boost Regression is ",gb_variance)
```

```
Test RMSE for Gradient Boost Regression is  2.6719085932133186
Train RMSE for Gradient Boost Regression is  2.4137350247513076
Variance for Gradient Boost Regression is  0.258173568462011
```

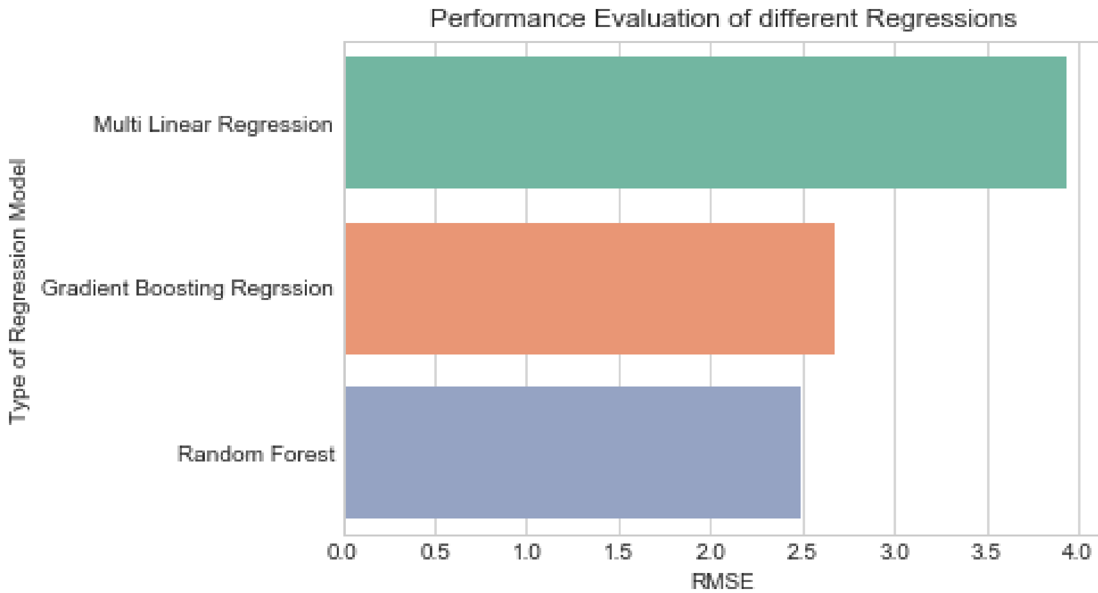
0.4 Performance Metrics

[50]:
```
regression = pd.DataFrame({"regression": ['Multi Linear Regression','Random␣
↪Forest',  'Gradient Boosting Regrssion'],
                           "rmse": [lm_rmse,rf_rmse,gb_rmse]},columns =␣
↪['regression','rmse'])
```

[51]:
```
regression = regression.sort_values(by='rmse', ascending = False)
```

[52]:
```
sns.barplot(regression['rmse'], regression['regression'], palette = 'Set2')
plt.xlabel("RMSE")
plt.ylabel('Type of Regression Model')
plt.title('Performance Evaluation of different Regressions')
```

[52]:
```
Text(0.5, 1.0, 'Performance Evaluation of different Regressions')
```



1 Project Results

- Implemented three regression based machine learning models - Multiple Linear Regression, Random Forest Regression and Gradient Boost Regression to predict the fare of a taxi ride in NYC.
- For Predictive Measures, we have used RMSE of the predicted fare with the actual fare amount. The RMSE for the Random Forest Regression was at 2.495 which was the lowest among the machine learning model that we employed. The RMSE for Gradient Boosting Regression was at 2.67 and for the Linear Regression Model the RMSE is at 3.94.
- We have used GridSearchCV for finding the optimal parameters for Random Forest Regression and Gradient Boosting Regression.
- The variance for the Random Forest Regression model is at 1.48 while variance for Gradient Boosting Regression Model is at 0.258 and for the Multiple Linear Regression, the variance is at 0.03

2 Insights for Decision Making

- The machine learning model that we can use to predict the NYC Taxi fare amount is the Gradient Boosting Regression. Even though the RMSE value of Gradient Boosting Regression is greater by 0.174 when we are comparing it with the RMSE of Random Forest Regression, the variance of the both the models are different. The variance of the Gradient Boosting Regression is at 0.258 and for the Random Forest Regression is at 1.48, we will be choosing the Gradient Boosting Regression model for prediction as it shows that it hasn't overfit the model(low variance).

- From the map, we can see that the number of drop off locations and pick up locations are in Manhattan. Also, the dropoff and pickup locations are also more concentrated at JFK Airport and La Guardia Airport.
- Most of the trips to Bronx, Brooklyn and Queen are long distance trips.
- The fare amount for the trips to and from Bronx, Queens and Brooklyn are much higher than the trips to and from other boroughs.
- Average Taxi fare is increasing per year.
- During the day, the number of taxi trips made are lowest during 12-5 am in the morning. As a result, the Average taxi fare is highest at these hours.

3 Impact of the Project Outcomes

- From the data we explored, the average taxi fare is increasing per year.
- The taxi fare is highest during the hours 12-5 in the morning. So, its better advised to not to hail a taxi during these hours.
- Most of the taxi pickup and dropoff points are in Manhattan borough(almost 60% of the trips), which shows that Manhattan is the commercial and shopping district of New York City.
- There is a significant difference in pickups and drop offs fare amount for Queens, Brooklyn, and Bronx boroughs. Also the distance travelled to Brooklyn, Bronx and Queens are longer. So, this might reflect why the fare amount for these boroughs are a bit higher than the rest.
- From the map, we can see that dropoffs and pickups are more concentrated on Manhattan borough and the two airports in New York City, i.e., La Guardia Airport and John F. Kennedy Airport.