KaggleDSF

September 25, 2018

1 New York City Taxi Fare Prediction

The aim is to predict the taxi fare for the customer taking cab service in New York City

Data Import and Exploration

```
In [1]: import pandas as pd
        import numpy as np
        import matplotlib.pyplot as plt
        import gc
        import os
        import seaborn as sb
        %matplotlib inline
```

```
In [2]: print(os.listdir('./'))
```

```
['.config', 'Anaconda3-5.2.0-Linux-x86_64.sh', 'train.csv', 'anaconda3', '.jupyter', 'output_37_
```

```
In [3]: train_df =  pd.read_csv('./train.csv')
        train_df.dtypes
```

```
Out[3]: key                    object
        fare_amount           float64
        pickup_datetime        object
        pickup_longitude      float64
        pickup_latitude       float64
        dropoff_longitude     float64
        dropoff_latitude      float64
        passenger_count         int64
        dtype: object
```

```
In [4]: print(train_df.shape[0])
```

```
55423856
```

In [5]:
```
#Drop any values that are NaN
train_df = train_df.dropna(how = 'any', axis = 'rows')
print('New size: %d' % len(train_df))
```

```
New size: 55423480
```

In [6]:
```
#Drop all the rows that have value = 0 in them
train_df = train_df[(train_df != 0).all(1)]
print('New size: %d' % len(train_df))
```

```
New size: 54127059
```

In [7]:
```
#Take only those Fares of passengers that have a postive value
train_df = train_df[train_df.fare_amount > 0]
print('New size: %d' % len(train_df))
```

```
New size: 54124853
```

In [8]:
```
#Using the general fare_amount from the given data set, it seems the fare of more than 1
train_df = train_df[train_df.fare_amount < 175]
print('New size: %d' % len(train_df))
```

```
New size: 54122340
```

In [9]:
```
#We are taking percentile of the data in order to identify the outliers and make sure to
train_df['pickup_longitude'].quantile([0, 0.01, 0.05, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7,
```

Out[9]:
```
0.00   -3442.059565
0.01     -74.014384
0.05     -74.006905
0.10     -74.002911
0.20     -73.994792
0.30     -73.990328
0.40     -73.986191
0.50     -73.982087
0.60     -73.977850
0.70     -73.972158
0.80     -73.963565
0.90     -73.953194
0.95     -73.934200
0.99     -73.785999
1.00    3457.625683
Name: pickup_longitude, dtype: float64
```

- From the above values we can see that the majority of data lies between 1%tile and 99%tile. So we are now going to filter the data according to their percentile ranges

```
In [10]: print('Size of training data: %d' % len(train_df) )

         train_df = train_df[(train_df['pickup_longitude'] <= train_df['pickup_longitude'].quant
         train_df = train_df[(train_df['pickup_longitude'] >= train_df['pickup_longitude'].quant

         train_df = train_df[(train_df['pickup_latitude'] <= train_df['pickup_latitude'].quantil
         train_df = train_df[(train_df['pickup_latitude'] >= train_df['pickup_latitude'].quantil

         train_df = train_df[(train_df['dropoff_longitude'] <= train_df['dropoff_longitude'].qua
         train_df = train_df[(train_df['dropoff_longitude'] >= train_df['dropoff_longitude'].qua

         train_df = train_df[(train_df['dropoff_latitude'] <= train_df['dropoff_latitude'].quant
         train_df = train_df[(train_df['dropoff_latitude'] >= train_df['dropoff_latitude'].quant

         print('New size: %d' % len(train_df))
```

```
Size of training data: 54122340
New size: 52852731
```

We tried taking different values of the percentile, and taking the data between 0.5%tile and 99.9%tile gives us the best data set for training.

```
In [11]: eu_cal = (train_df['dropoff_latitude'] - train_df['pickup_latitude']) **2  + (train_df[

         eu_dist = np.sqrt(eu_cal)
```

1.0.1 Pearson Correlation between Eucledian Distance and Fare Amount

```
In [12]: eu_dist.corr(train_df['fare_amount'])
```

```
Out[12]: 0.8806340289981661
```

```
In [13]: #Get the given pickup time in a new format of date of Y-M-D:H-M-S
         train_df['pickup_datetime'] = train_df['pickup_datetime'].str.replace(" UTC", "")
         #replace the given date time in a new format
         train_df['pickup_datetime'] = pd.to_datetime(train_df['pickup_datetime'], format='%Y-%m

In [14]: journey_time = (train_df['pickup_datetime'].dt.hour)*60 + train_df['pickup_datetime'].d
```

1.0.2 Pearson Correlation between Eucledian Distance and the Time

```
In [15]: eu_dist.corr(journey_time)
```

```
Out[15]: -0.032817644627646574
```

1.0.3 Pearson Correlation between Fare Amount and the Time

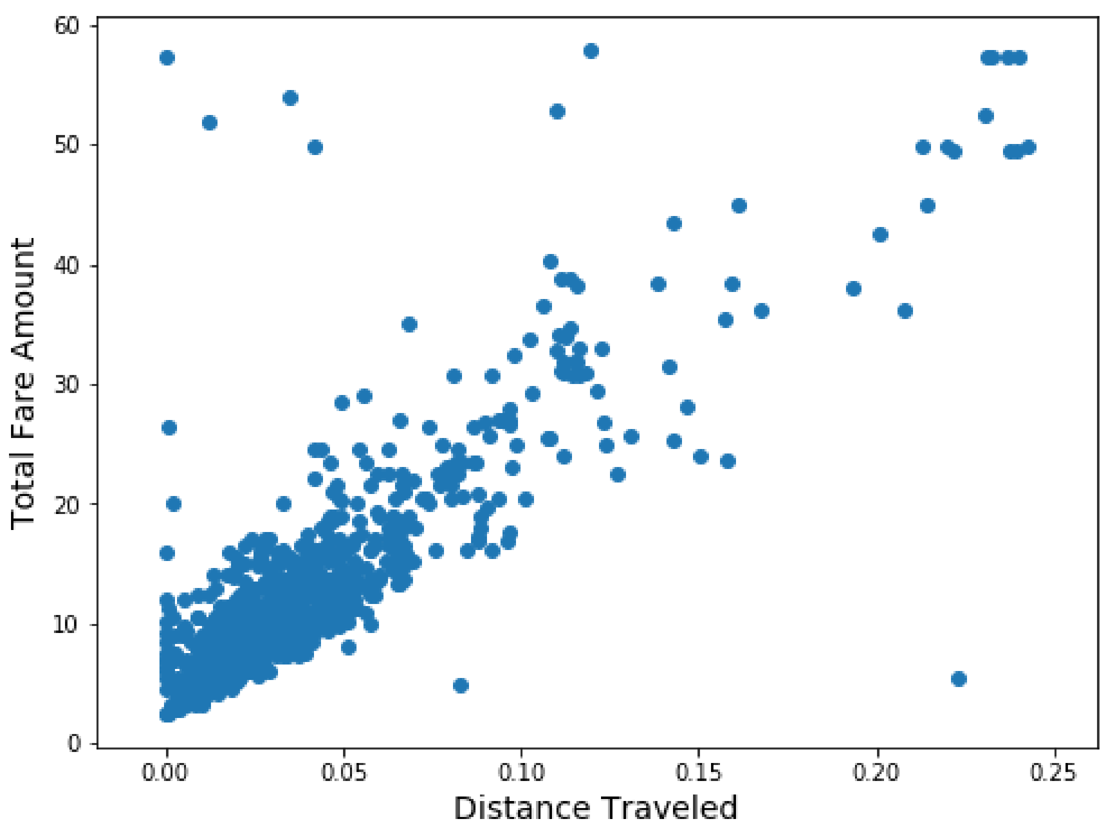
```
In [16]: journey_time.corr(train_df['fare_amount'])
```

```
Out[16]: -0.01736950684640317
```

```
In [17]: train_df['distance'] = eu_dist
         train_df['journey_time'] = journey_time
```

1.0.4 Plot between Distance and Total Fare Amount

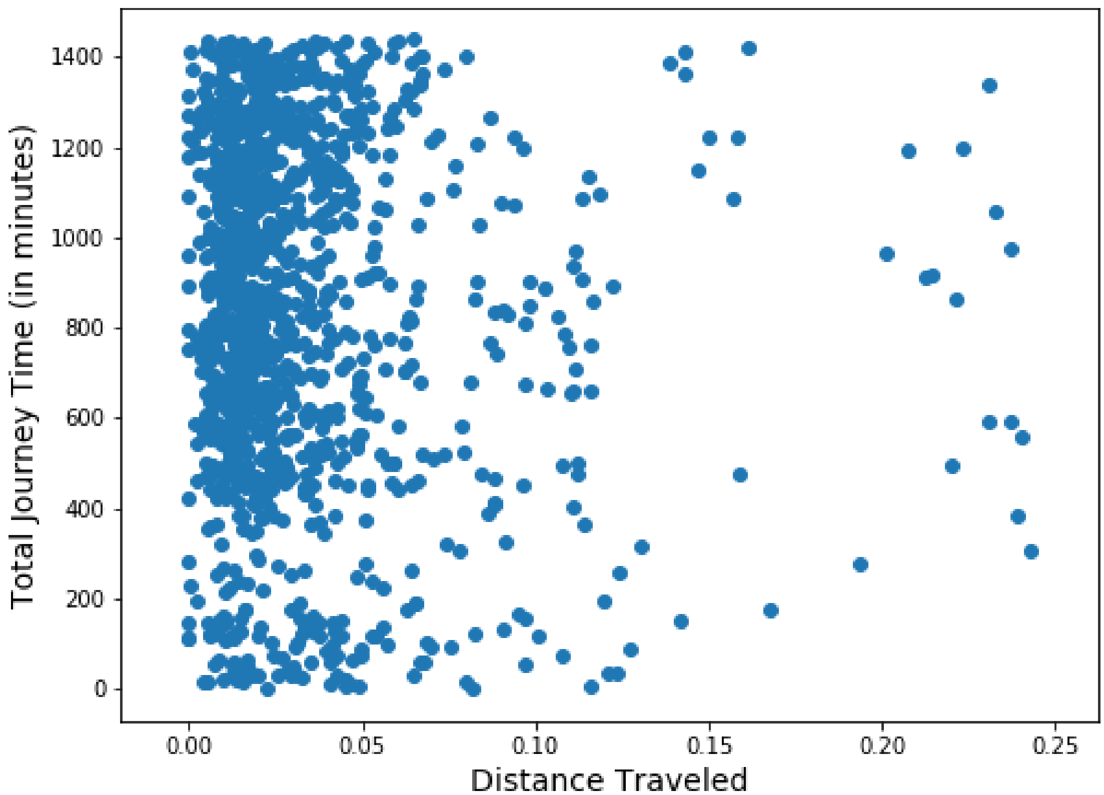
```
In [18]: plt.figure(figsize = (8,6))
         plt.xlabel('Distance Traveled', fontsize = 14)
         plt.ylabel('Total Fare Amount', fontsize = 14)
         plt.scatter(train_df[:1200].distance, train_df[:1200].fare_amount)
         plt.show()
```



- The plot between Distance traveled and the Total Fare Amount generates a linear relationship

1.0.5 Plot between Distance and Time

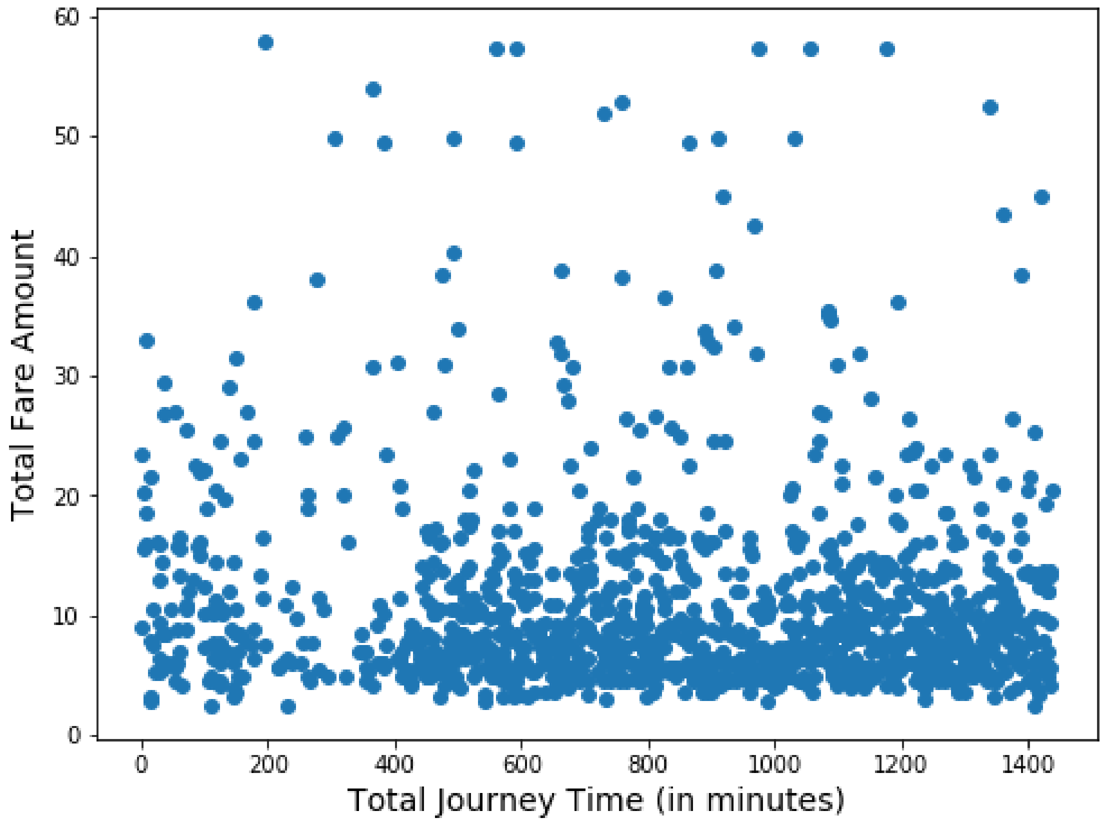
In [19]:
```
plt.figure(figsize = (8,6))
plt.xlabel('Distance Traveled', fontsize = 14)
plt.ylabel('Total Journey Time (in minutes)', fontsize = 14)
plt.scatter(train_df[:1200].distance, train_df[:1200].journey_time)
plt.show()
```



- The plot between Distance traveled and the total journey time of a passenger generates a non - linear relationship. This plot doesn't tell us the exact relationship between the variables

1.0.6 Plot between Total Fare Amount and Time

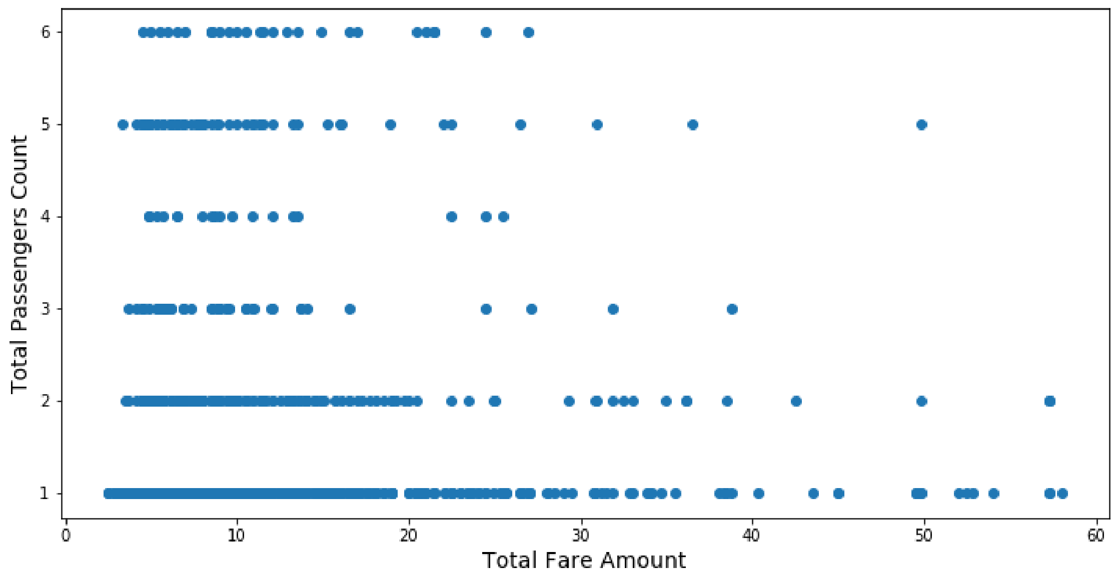
In [20]:
```
plt.figure(figsize = (8,6))
plt.xlabel('Total Journey Time (in minutes)', fontsize = 14)
plt.ylabel('Total Fare Amount', fontsize = 14)
plt.scatter(train_df[:1200].journey_time, train_df[:1200].fare_amount)
plt.show()
```



- The plot between Distance traveled and the total journey time of a passenger generates a non - linear relationship. The plot gives a scattered data and hence we cannot infer anything from it

1.0.7 Plot between Total Fare Amount and Total Passenger Count

```
In [21]: plt.figure(figsize = (12,6))
         plt.xlabel('Total Fare Amount', fontsize = 14)
         plt.ylabel('Total Passengers Count', fontsize = 14)
         plt.scatter(train_df[:1200].fare_amount, train_df[:1200].passenger_count)
         plt.show()
```

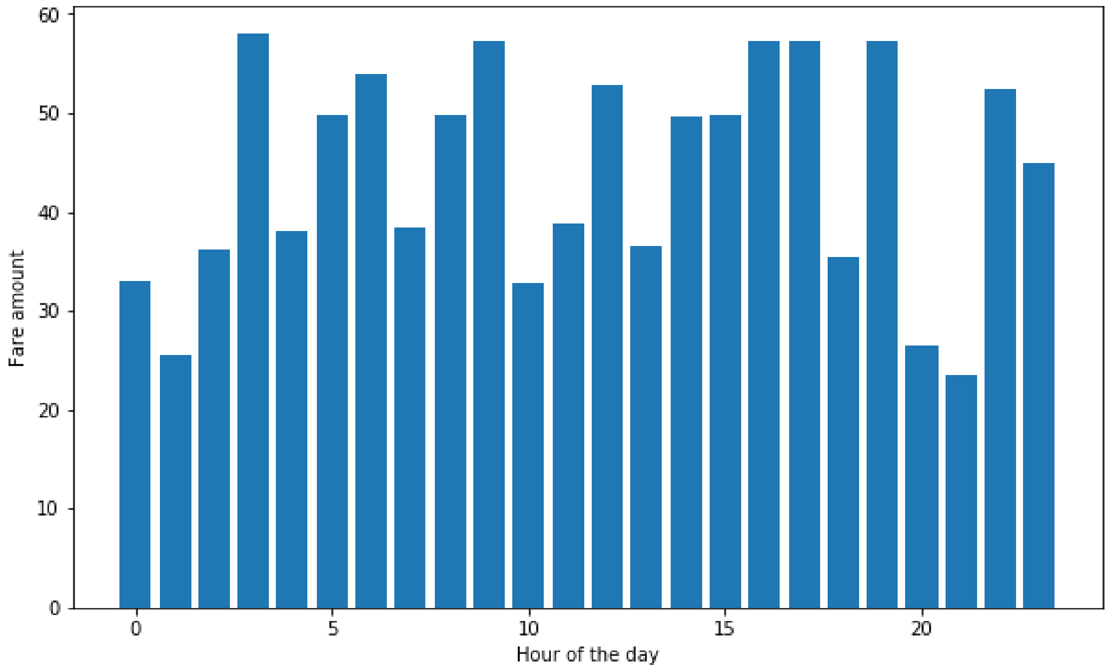


- The plot between total fare and the number of passengers taking the cab, we see that generally people in New York City are spending less than 30$ while they take the cab service, although it is a very generic statement

1.0.8 Bar Chart comparing the fare amount depending on the hour of the day

```
In [22]: pickup_hour_time = (train_df['pickup_datetime'].dt.hour)[:1200]
         fare = train_df[:1200].fare_amount

         plt.figure(figsize = (10,6))
         plt.bar(pickup_hour_time, fare)
         plt.xlabel('Hour of the day')
         plt.ylabel('Fare amount')
         plt.show()
```



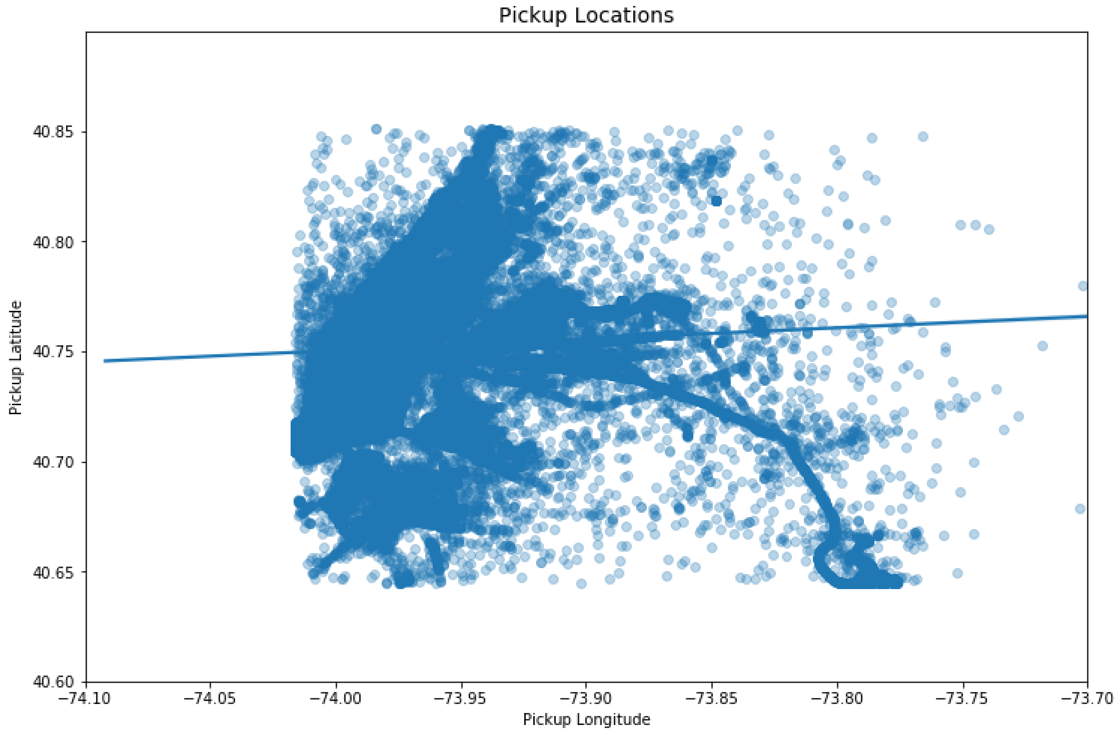
- This plot shows relationship between the fare amount at particular hours of the day. We can see at what times during the day, we have highest amount of fare. It seems that during early morning hours(3 AM, 9 AM), and the evening at 5PM, 6PM and 8 PM we are able to see the highest fare amount throughout the day.

1.0.9 Data plot based on Pickup locations

In [23]:
```
plt.figure(figsize = (12,8))
plt.title('Pickup Locations', fontsize=14)
p_long, p_lat = pd.Series(train_df[:1000000].pickup_longitude, name="Pickup Longitude")

pickup = sb.regplot(x=p_long, y=p_lat, scatter_kws={'alpha':0.3})
pickup.set(xlim = (-74.1, -73.7))
```

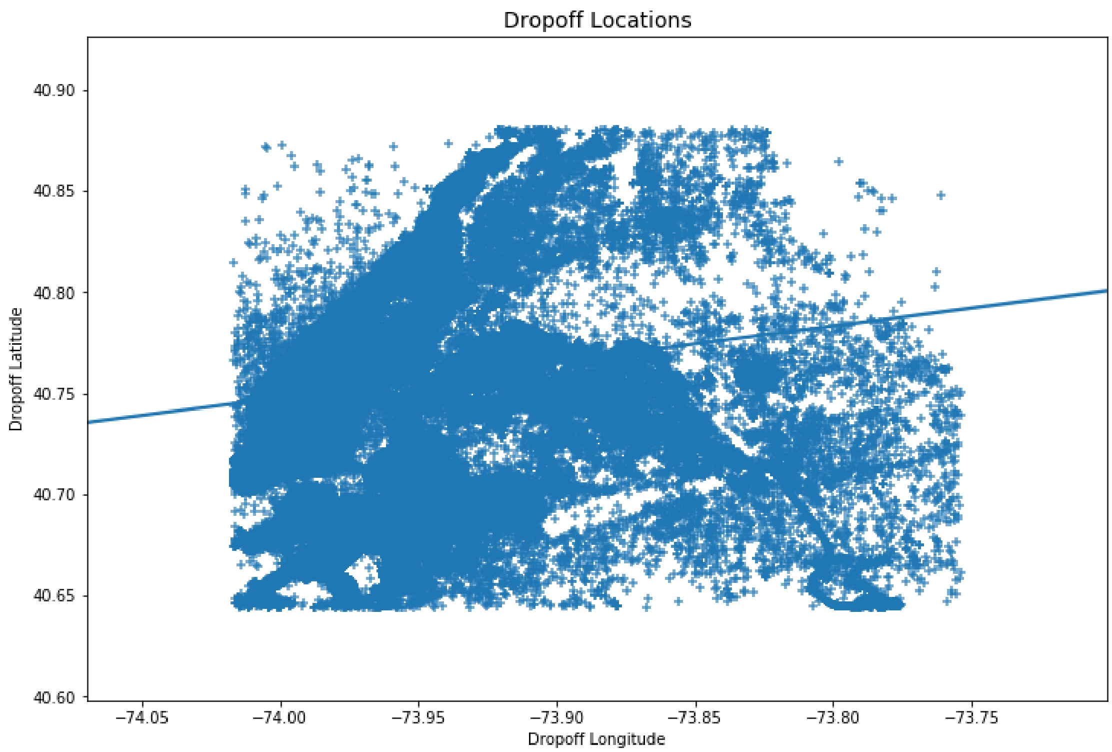
Out[23]:
```
[(-74.1, -73.7)]
```



- From the given plot we get to know the various locations of pickups based on their latitude and longitude coordinates. Since we have a huge dataset, we can see the whole map of New York City from the marking locations of pickup. We can also see there are a few points for the pickup in the right bottom corner, because those coordinates are of JFK airport, while a few scattered points are also present which represent the pickup locations in the New York's boroughs as well. So we can take the pickup based on these locations as well.

1.0.10 Data plot based on DropOff locations

```
In [24]: plt.figure(figsize = (12,8))
         plt.title('Dropoff Locations', fontsize=14)
         d_long, d_lat = pd.Series(train_df[:1000000].dropoff_longitude, name="Dropoff Longitude
         dropoff = sb.regplot(x=d_long, y=d_lat, marker="+")
```



- From the given plot we get to know the various locations of dropoff points based on their latitude and longitude coordinates. We can see the map of New York City from the marking locations of dropoff. We can also see there are a few points for the pickup in the right bottom corner, because those coordinates are of JFK airport. Here we see a lot of scattered points across Manhattan, so we can say that, many of the taxi cab's customers have a drop off location outside of New York City.

1.0.11 Addtional Feature Extraction

In [25]:
```
train_df['day'] = train_df['pickup_datetime'].dt.day;
train_df['month'] = train_df['pickup_datetime'].dt.month;
train_df['hour'] = train_df['pickup_datetime'].dt.hour;
train_df['minute'] = train_df['pickup_datetime'].dt.minute;
train_df.head()
```

Out[25]:
```
                              key  fare_amount     pickup_datetime  \
0    2009-06-15 17:26:21.0000001          4.5 2009-06-15 17:26:21
2   2011-08-18 00:35:00.00000049          5.7 2011-08-18 00:35:00
3    2012-04-21 04:30:42.0000001          7.7 2012-04-21 04:30:42
4  2010-03-09 07:51:00.000000135          5.3 2010-03-09 07:51:00
5    2011-01-06 09:50:45.0000002         12.1 2011-01-06 09:50:45

   pickup_longitude  pickup_latitude  dropoff_longitude  dropoff_latitude  \
```

```
0       -73.844311       40.721319       -73.841610        40.712278
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

   passenger_count  distance  journey_time  day  month  hour  minute
0                1  0.009436          1046   15      6    17      26
2                2  0.013674            35   18      8     0      35
3                1  0.025340           270   21      4     4      30
4                1  0.019470           471    9      3     7      51
5                1  0.038675           590    6      1     9      50
```

1.0.12 Training Data - Linear Regression

In [38]:
```
data2 = train_df[:5000000]

#We are considering three features for the data selection. These three features could b
features = ['passenger_count', 'distance','journey_time']
X = data2[features]
y = data2['fare_amount']

#Source for Linear Regression: https://towardsdatascience.com/linear-regression-in-pyth
from sklearn.cross_validation import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn import metrics

#We are splitting the data for training and testing according to the ratio of 80 : 20 r
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)

print(X_train.shape, X_test.shape, y_train.shape, y_test.shape)

lr = LinearRegression()
lr.fit(X_train, y_train)
```

```
(4000000, 3) (1000000, 3) (4000000,) (1000000,)
```

Out[38]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normalize=False)
```

In [39]:
```
print(lr.intercept_)
print(lr.coef_)
zip(features, lr.coef_)
```

```
3.680830954073837
[3.75550574e-02 2.14014535e+02 2.38247046e-04]
```

Out[39]:
```
<zip at 0x7f56051f1ac8>
```

```
In [40]: y_pred = lr.predict(X_test)
         print(y_pred)
         print("RMS: ", np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
```

```
[10.47004978 20.54124327  6.02887277 ... 11.37778047  4.58511403
  7.04375718]
RMS:  3.91867263627871
```

We are getting RMS value = 3.912 from the training data set

1.0.13 Actual Data Prediction using Linear Regression

```
In [41]: actual_data =  pd.read_csv('./test.csv')
         actual_data.dtypes
```

```
Out[41]: key                    object
         pickup_datetime        object
         pickup_longitude      float64
         pickup_latitude       float64
         dropoff_longitude     float64
         dropoff_latitude      float64
         passenger_count         int64
         dtype: object
```

```
In [42]: eu_cal = (actual_data['dropoff_latitude'] - actual_data['pickup_latitude']) **2  + (act

         eu_dist = np.sqrt(eu_cal)
```

```
In [43]: actual_data['pickup_datetime'] = actual_data['pickup_datetime'].str.replace(" UTC", "")
         #replace the given date time in a new format
         actual_data['pickup_datetime'] = pd.to_datetime(actual_data['pickup_datetime'], format=
```

```
In [44]: journey_time = (actual_data['pickup_datetime'].dt.hour)*60 + actual_data['pickup_dateti
```

```
In [45]: actual_data['distance'] = eu_dist
         actual_data['journey_time'] = journey_time
```

```
In [46]: actual_data['day'] = actual_data['pickup_datetime'].dt.day;
         actual_data['month'] = actual_data['pickup_datetime'].dt.month;
         actual_data['hour'] = actual_data['pickup_datetime'].dt.hour;
         actual_data['minute'] = actual_data['pickup_datetime'].dt.minute;
         actual_data.head()
```

```
Out[46]:                              key     pickup_datetime  pickup_longitude  \
         0  2015-01-27 13:08:24.0000002 2015-01-27 13:08:24        -73.973320
         1  2015-01-27 13:08:24.0000003 2015-01-27 13:08:24        -73.986862
         2  2011-10-08 11:53:44.0000002 2011-10-08 11:53:44        -73.982524
         3  2012-12-01 21:12:12.0000002 2012-12-01 21:12:12        -73.981160
```

```
4  2012-12-01 21:12:12.0000003 2012-12-01 21:12:12       -73.966046

   pickup_latitude  dropoff_longitude  dropoff_latitude  passenger_count  \
0        40.763805         -73.981430         40.743835                1
1        40.719383         -73.998886         40.739201                1
2        40.751260         -73.979654         40.746139                1
3        40.767807         -73.990448         40.751635                1
4        40.789775         -73.988565         40.744427                1

   distance  journey_time  day  month  hour  minute
0  0.021554           788   27      1    13       8
1  0.023180           788   27      1    13       8
2  0.005870           713    8     10    11      53
3  0.018649          1272    1     12    21      12
4  0.050631          1272    1     12    21      12
```

```
In [47]: features1 = ['passenger_count', 'distance','journey_time']
         X1 = actual_data[features1]
```

```
In [49]: predict_value = lr.predict(X1)
         print(predict_value)
```

```
[ 8.51897634  8.86693473  5.14460608 ... 50.6405486  21.23970975
  6.95370272]
```

```
In [50]: final_data = pd.DataFrame()
         final_data['key'] = actual_data['key']
         final_data['fare_amount'] = predict_value
         final_data.to_csv('final_result.csv',sep=',', index = False)
```

1.0.14 Training Data - Random Forest Regressor

```
In [167]: data3 = train_df[:5000000]

          from sklearn.ensemble import RandomForestRegressor
          from sklearn.datasets import make_regression
```

```
In [168]: regressor = RandomForestRegressor(max_depth= 12, random_state=0,n_estimators=5)

          #We are considering three features for the data selection. These three features could
          #I considered other features as well, but they didn't improve the score at all. These
          features3 = ['passenger_count', 'distance','journey_time']

          X3 = data3[features3]
          y3 = data3['fare_amount']

          regressor.fit(X3, y3)

          #Source: http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomFores
```

Out[168]:
```
RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=12,
           max_features='auto', max_leaf_nodes=None,
           min_impurity_decrease=0.0, min_impurity_split=None,
           min_samples_leaf=1, min_samples_split=2,
           min_weight_fraction_leaf=0.0, n_estimators=5, n_jobs=1,
           oob_score=False, random_state=0, verbose=0, warm_start=False)
```

In [169]:
```
print(regressor.feature_importances_)
```

```
[6.96457525e-04 9.87749333e-01 1.15542096e-02]
```

In [170]:
```
#We are taking the testing data from the current given data set - for local prediction
X3_test = train_df[5000001: 6000000][features3]

predict3 = regressor.predict(X3_test)
print(predict3)
```

```
[ 5.67247324  5.78835029  5.36457978 ...  6.13259908 11.51195472
  5.36457978]
```

1.0.15 Data Prediction using Random Forest

In [171]:
```
actual_data3 =  pd.read_csv('./test.csv')
actual_data3.dtypes
```

Out[171]:
```
key                   object
pickup_datetime       object
pickup_longitude     float64
pickup_latitude      float64
dropoff_longitude    float64
dropoff_latitude     float64
passenger_count        int64
dtype: object
```

In [172]:
```
eu_cal3 = (actual_data3['dropoff_latitude'] - actual_data3['pickup_latitude']) **2  +

eu_dist3 = np.sqrt(eu_cal3)
```

In [173]:
```
actual_data3['pickup_datetime'] = actual_data3['pickup_datetime'].str.replace(" UTC",
#replace the given date time in a new format
actual_data3['pickup_datetime'] = pd.to_datetime(actual_data3['pickup_datetime'], form
```

In [174]:
```
#Calculating the time using hour and minute of the given time stamp
journey_time3 = (actual_data3['pickup_datetime'].dt.hour)*60 + actual_data3['pickup_da
```

In [175]:
```
actual_data3['distance'] = eu_dist3
actual_data3['journey_time'] = journey_time3
```

In [176]:
```
#Updating the data and adding new columns to accomodate new features
actual_data3['day'] = actual_data3['pickup_datetime'].dt.day;
actual_data3['month'] = actual_data3['pickup_datetime'].dt.month;
actual_data3['hour'] = actual_data3['pickup_datetime'].dt.hour;
actual_data3['minute'] = actual_data3['pickup_datetime'].dt.minute;
actual_data3.head()
```

Out[176]:
```
                            key     pickup_datetime  pickup_longitude  \
0  2015-01-27 13:08:24.0000002 2015-01-27 13:08:24        -73.973320
1  2015-01-27 13:08:24.0000003 2015-01-27 13:08:24        -73.986862
2  2011-10-08 11:53:44.0000002 2011-10-08 11:53:44        -73.982524
3  2012-12-01 21:12:12.0000002 2012-12-01 21:12:12        -73.981160
4  2012-12-01 21:12:12.0000003 2012-12-01 21:12:12        -73.966046

   pickup_latitude  dropoff_longitude  dropoff_latitude  passenger_count  \
0        40.763805         -73.981430         40.743835                1
1        40.719383         -73.998886         40.739201                1
2        40.751260         -73.979654         40.746139                1
3        40.767807         -73.990448         40.751635                1
4        40.789775         -73.988565         40.744427                1

   distance  journey_time  day  month  hour  minute
0  0.021554           788   27      1    13       8
1  0.023180           788   27      1    13       8
2  0.005870           713    8     10    11      53
3  0.018649          1272    1     12    21      12
4  0.050631          1272    1     12    21      12
```

In [177]:
```
X31 = actual_data3[features3]
predict_value3 = regressor.predict(X31)
```

In [178]:
```
final_data3 = pd.DataFrame()
final_data3['key'] = actual_data3['key']
final_data3['fare_amount'] = predict_value3
final_data3.to_csv('final_result3.csv',sep=',', index = False)
```