Project Name - Cab Fare Prediction

Somesh Ugar

28 July 2019

Chapter 1 : Introduction

1.1 Problem Statement

You are a cab rental start-up company. You have successfully run the pilot project and now want to launch your cab service across the country. You have collected the historical data from your pilot project and now have a requirement to apply analytics for fare prediction. You need to design a system that predicts the fare amount for a cab ride in the city.

1.2 Data Preprocessing

Our task is to build a models which will predict the Cab fare based on the date-time and other given features. Given below is a sample of the data set that we are using to predict the Cab fare:

Number of attributes:

pickup_datetime - timestamp value indicating when the cab ride started.
pickup_longitude - float for longitude coordinate of where the cab ride started.
pickup_latitude - float for latitude coordinate of where the cab ride started.
dropoff_longitude - float for longitude coordinate of where the cab ride ended.
dropoff_latitude - float for latitude coordinate of where the cab ride ended.
passenger_count - an integer indicating the number of passengers in the cab ride.

Table 1.1: Cab fare Train Data (Columns: 1-7)

	fare_amount	pickup_datetime	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_count
0	4.5	2009-06-15 17:26:21 UTC	-73.844311	40.721319	-73.841610	40.712278	1.0
1	16.9	2010-01-05 16:52:16 UTC	-74.016048	40.711303	-73.979268	40.782004	1.0
2	5.7	2011-08-18 00:35:00 UTC	-73.982738	40.761270	-73.991242	40.750562	2.0
3	7.7	2012-04-21 04:30:42 UTC	-73.987130	40.733143	-73.991567	40.758092	1.0
4	5.3	2010-03-09 07:51:00 UTC	-73.968095	40.768008	-73.956655	40.783762	1.0

Table 1.2: Cab fare Test Data (Columns: 1-6)

	pickup_datetime	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_count
0	2015-01-27 13:08:24 UTC	-73.973320	40.763805	-73.981430	40.743835	1
1	2015-01-27 13:08:24 UTC	-73.986862	40.719383	-73.998886	40.739201	1
2	2011-10-08 11:53:44 UTC	-73.982524	40.751260	-73.979654	40.746139	1
3	2012-12-01 21:12:12 UTC	-73.981160	40.767807	-73.990448	40.751635	1
4	2012-12-01 21:12:12 UTC	-73.966046	40.789775	-73.988565	40.744427	1

1.3 Data Types of all variables

Train data set contains 7 variables and 16067 observations in total.
$ fare_amount : Factor w/ 469 levels "","0","0.01",..: 339 57 403 441 386 24 435 55 1 460 ...
$ pickup_datetime : Factor w/ 16021 levels "2009-01-01 01:31:49 UTC",..: 1115 2509 6550 8252 2919 4986 9743 7479 9838 1606 ...
$ pickup_longitude : num -73.8 -74 -74 -74 -74 ...
$ pickup_latitude : num 40.7 40.7 40.8 40.7 40.8 ...
$ dropoff_longitude: num -73.8 -74 -74 -74 -74 ...
$ dropoff_latitude : num 40.7 40.8 40.8 40.8 40.8 ...
$ passenger_count : num 1 1 2 1 1 1 1 1 1 2 ...

Test data set contains 6 variables and 9914 observations in total.
$ pickup_datetime : Factor w/ 1753 levels "2009-01-01 11:04:24 UTC",..: 1648 1648 747 1041 1041 1041 744 744 744 1384 ...
$ pickup_longitude : num -74 -74 -74 -74 -74 ...
$ pickup_latitude : num 40.8 40.7 40.8 40.8 40.8 ...
$ dropoff_longitude: num -74 -74 -74 -74 -74 ...
$ dropoff_latitude : num 40.7 40.7 40.7 40.8 40.7 ...
$ passenger_count : int 1 1 1 1 1 1 1 1 1 1 ...

1.4 Summary of all variables

Train Data

	fare_amount	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_count
count	16043.000000	16067.000000	16067.000000	16067.000000	16067.000000	16012.000000
mean	15.041919	-72.462787	39.914725	-72.462328	39.897906	2.625070
std	430.459960	10.578384	6.826587	10.575062	6.187087	60.844122
min	0.000000	-74.438233	-74.006893	-74.429332	-74.006377	0.000000
25%	6.000000	-73.992156	40.734927	-73.991182	40.734651	1.000000
50%	8.500000	-73.981698	40.752603	-73.980172	40.753567	1.000000
75%	12.500000	-73.966838	40.767381	-73.963643	40.768013	2.000000
max	54343.000000	40.766125	401.083332	40.802437	41.366138	5345.000000

When we look at statistic summary, we have several discoveries:

- The minimum fare amount is 0 and maximum is 54345
- Minimum passenger count is 0 and maximum is 5345

We are going to fix them.

- Remove passenger count equal to 0 and greater than 6

```
Train_data = Train_data[(Train_data.passenger_count>=1) & (Train_data.passenger_count<=6)]
```

```
Train_data = Train_data.drop(Train_data[Train_data.passenger_count == 1.3].index, axis=0)
```

- We are removing fare amount above 0.999 quantile and below 0.001 quantile

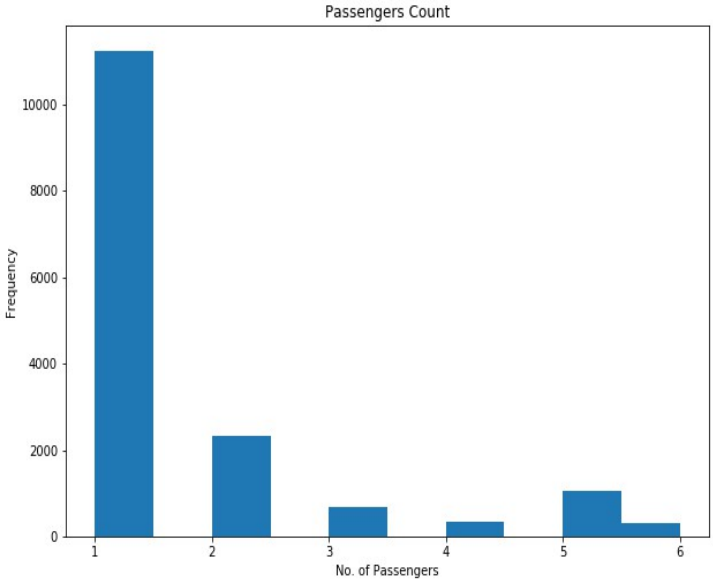
```
Train_data = Train_data[(Train_data.fare_amount>Train_data.fare_amount.quantile(.001)) &
                        (Train_data.fare_amount<Train_data.fare_amount.quantile(.999))]
```

Test Data

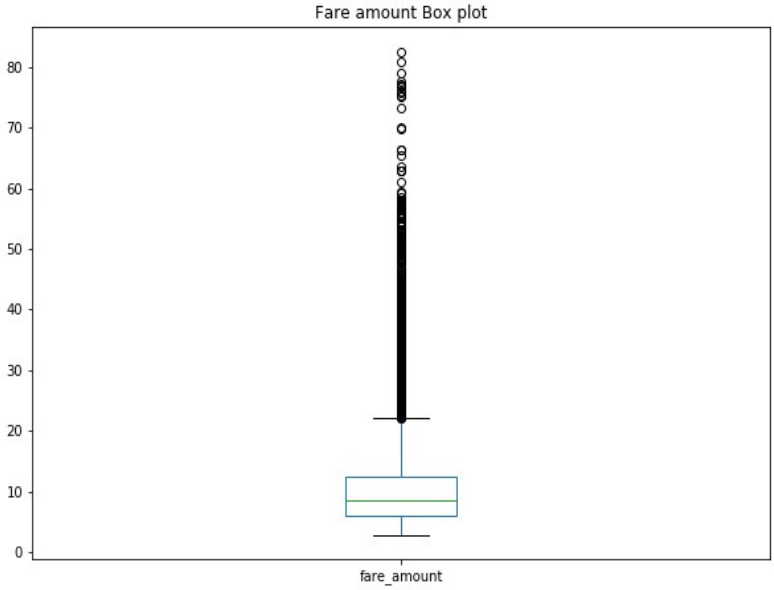
	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_count
count	9914.000000	9914.000000	9914.000000	9914.000000	9914.000000
mean	-73.974722	40.751041	-73.973657	40.751743	1.671273
std	0.042774	0.033541	0.039072	0.035435	1.278747
min	-74.252193	40.573143	-74.263242	40.568973	1.000000
25%	-73.992501	40.736125	-73.991247	40.735254	1.000000
50%	-73.982326	40.753051	-73.980015	40.754065	1.000000
75%	-73.968013	40.767113	-73.964059	40.768757	2.000000
max	-72.986532	41.709555	-72.990963	41.696683	6.000000

Chapter 2 : Exploratory Data Analysis :

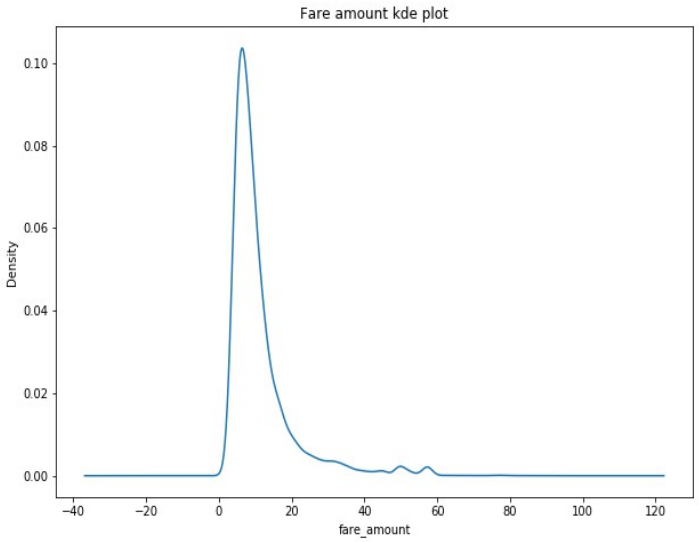
2.1 Data Visualization



From the above figure, it seems that the number of passengers follow a nearly normal distribution. The first plot shows the frequency of passenger count, the frequency of passenger count is decreased from 1 to 6 gradually .

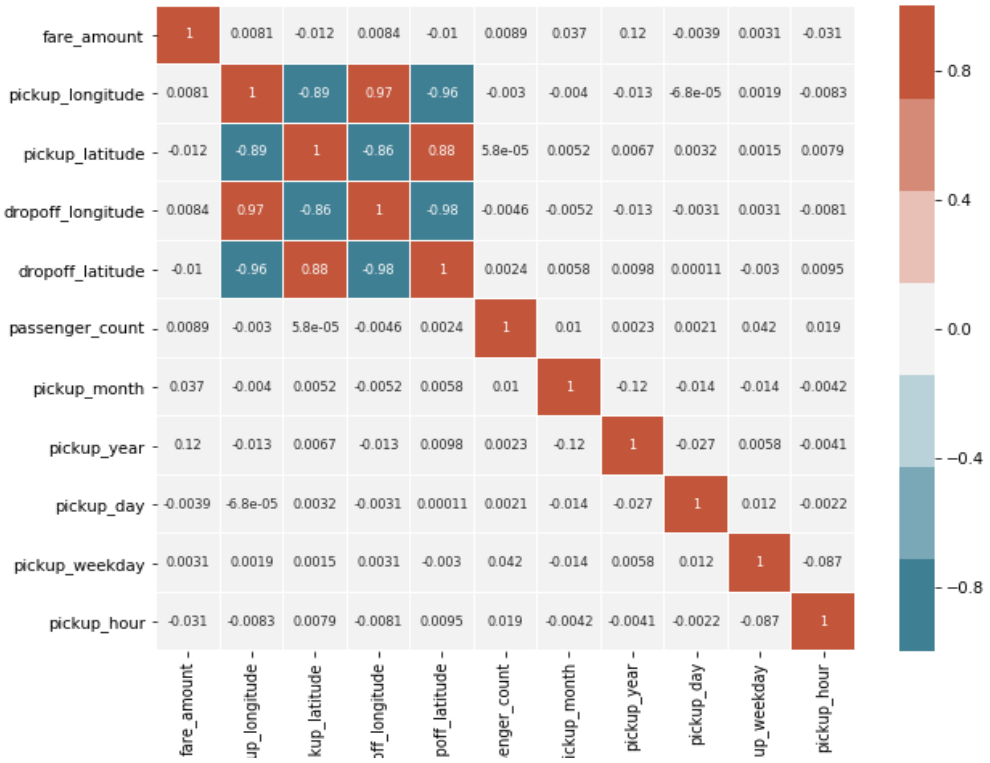


The second plot shows the box plot of fare amount, which confirms that that most fare amount are very small.



The third plot shows the Kernel density estimation(KDE) of fare amount, which confirms that that most fare amount are very small and leis between 10 to 20 .

2.2 Correlation Analysis:



Above plot shows the correlation between the dataset variables. A plot of their distributions highlighting the value of the target variable might also reveal some patterns. Correlation analysis will allow to identify relationships between the dataset variables

2.3 Feature Engineering

Extract information from datetime (day of week,year, month, hour, day). Taxi fares change during different hours of the day and on weekdays/weekends/holidays.

```
Train_data['pickup_datetime'] = pd.to_datetime(Train_data['pickup_datetime'],format='%Y-%m-%d %H:%M:%S %Z',
                                               errors='coerce')
```

```
Train_data['pickup_month'] = Train_data['pickup_datetime'].dt.month
Train_data['pickup_year'] = Train_data['pickup_datetime'].dt.year
Train_data['pickup_day'] = Train_data['pickup_datetime'].dt.day
Train_data['pickup_weekday'] = Train_data['pickup_datetime'].dt.weekday
Train_data['pickup_hour'] = Train_data['pickup_datetime'].dt.hour
```

Calculate the distance from pick up to drop off. The longer the trip, the higher the price.
Create a new variable log of a distance.

Finding distances based on Latitude and Longitude

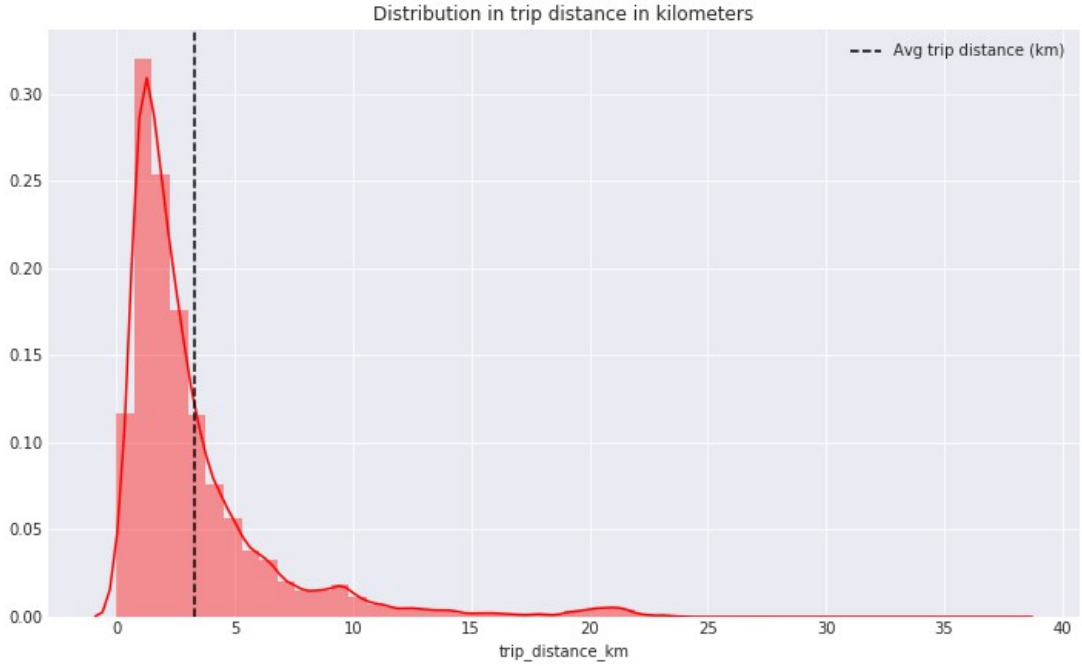
The haversine formula determines the great-circle distance between two points on a sphere given their longitudes and latitudes.

Formula:

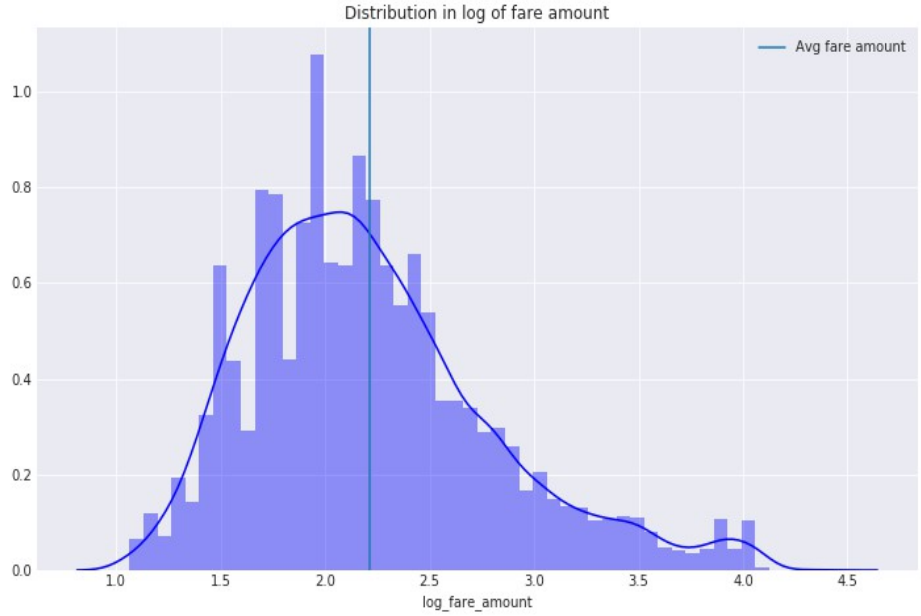
dlon = lon2 - lon1
dlat = lat2 - lat1
a = (sin(dlat/2))^2 + cos(lat1) cos(lat2) (sin(dlon/2))^2
c = 2 * atan2(sqrt(a), sqrt(1-a))
d = R * c (where R is the radius of the Earth)

```
#radius of earth in kilometers
R = 6373.0

pickup_lat  = np.radians(Train_data["pickup_latitude"])
pickup_lon  = np.radians(Train_data["pickup_longitude"])
dropoff_lat = np.radians(Train_data["dropoff_latitude"])
dropoff_lon = np.radians(Train_data["dropoff_longitude"])

dist_lon = dropoff_lon - pickup_lon
dist_lat = dropoff_lat - pickup_lat

#Formula
a = (np.sin(dist_lat/2))**2 + np.cos(pickup_lat) * np.cos(dropoff_lat) * (np.sin(dist_lon/2))**2
c = 2 * np.arctan2( np.sqrt(a), np.sqrt(1-a) )
d = R * c #(where R is the radius of the Earth)

Train_data["trip_distance_km"] = d

#create new variable log of distance
Train_data["log_trip_distance"] = np.log(Train_data["trip_distance_km"])
```

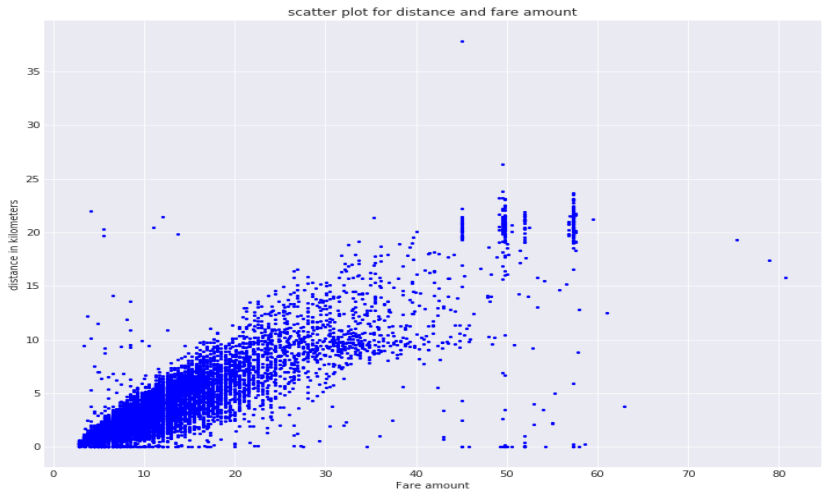
Let’s visualize one of the new feature, —trip_distance_km



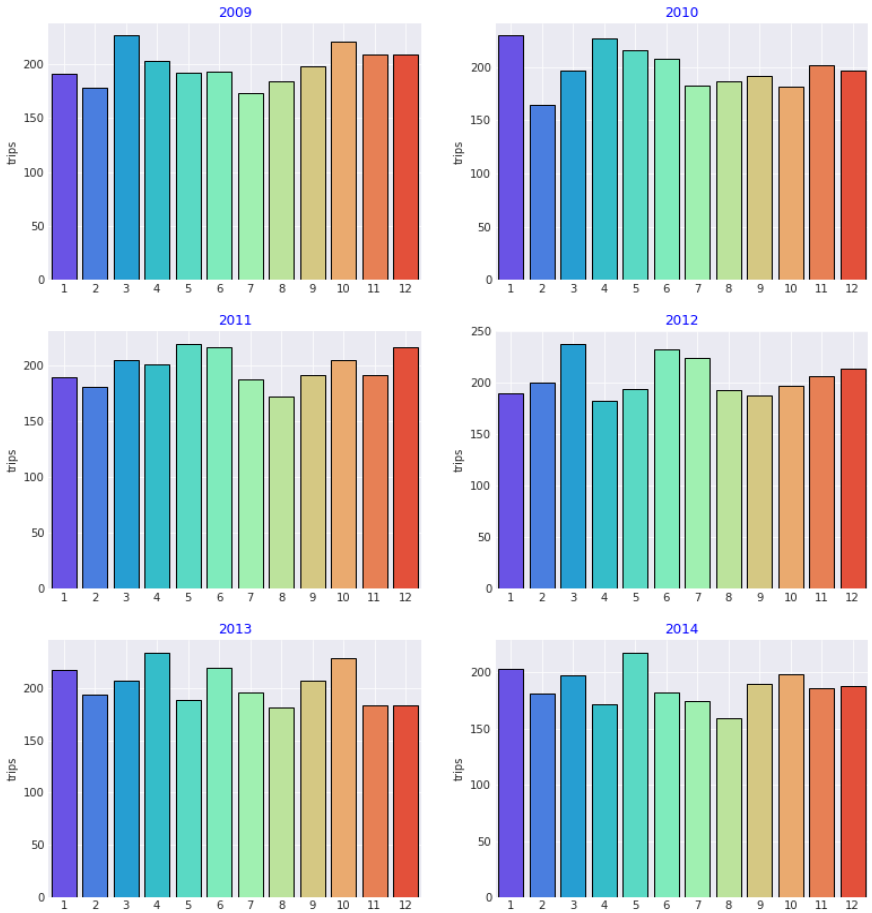
The below plot shows the Distribution in log of fare amount



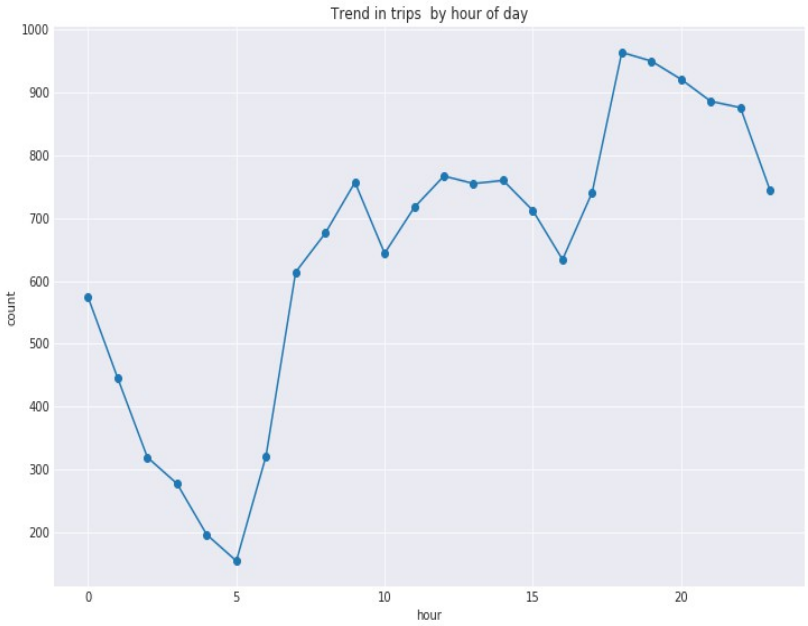
The below scatter plot shows the relationship between distance and fare amount



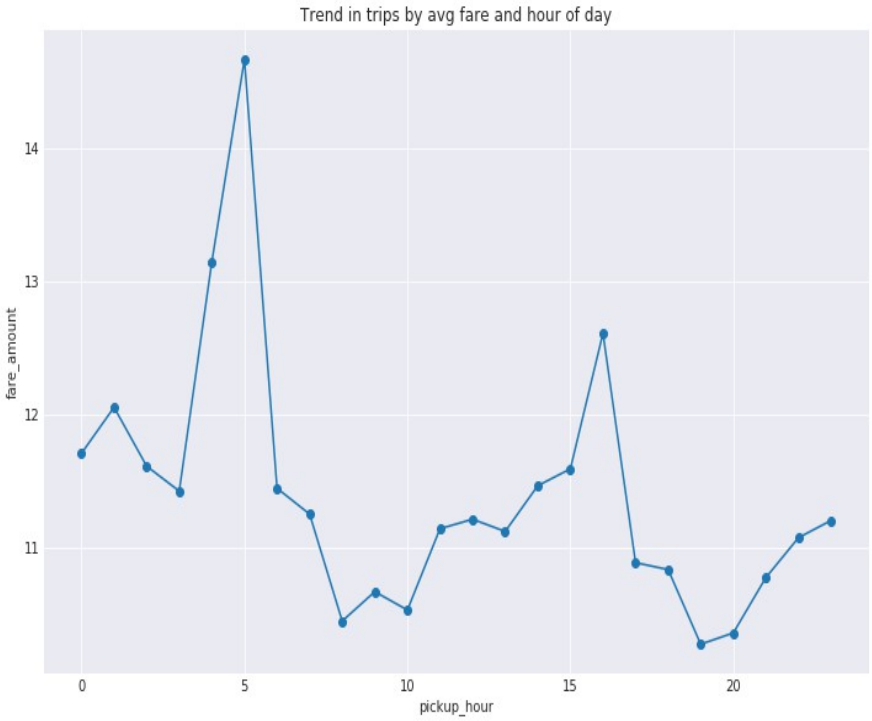
These six plots show the trip_counts of different months with pickup_year from 2009 to 2014



The below line plot shows the trend of trips by hour of a day



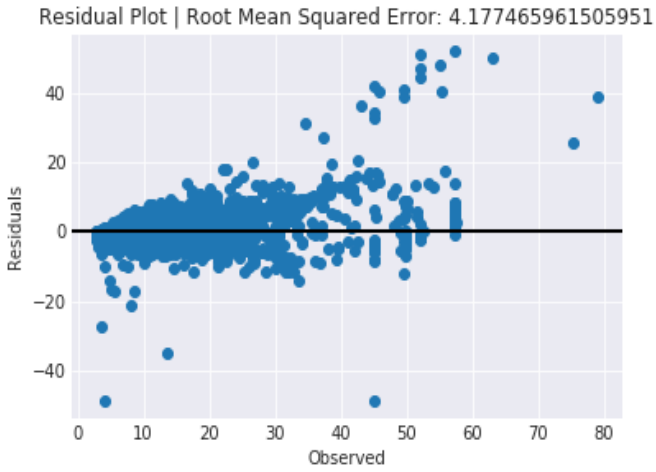
The below line plot shows the trend in trips by average hour of a day and fare amount



Chapter 3: Model Development

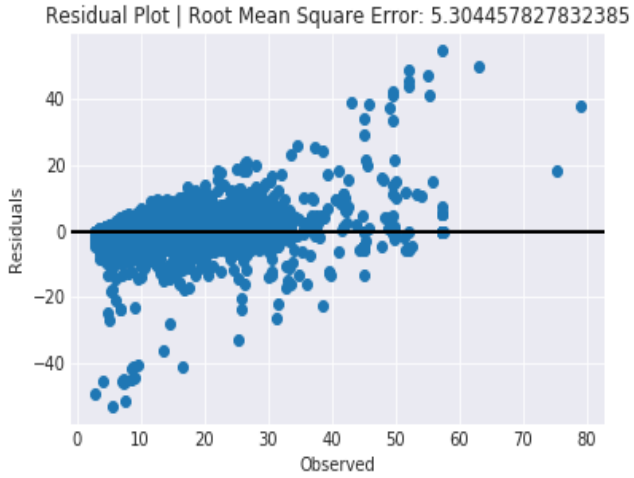
Model 1 - Linear Regression Model

```
y = train.fare_amount
X = train.drop(['fare_amount'],axis=1)

X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.30,random_state=42)

model= LinearRegression()
model.fit(X_train,y_train)
y_pred_lm = model.predict(X_test)
```



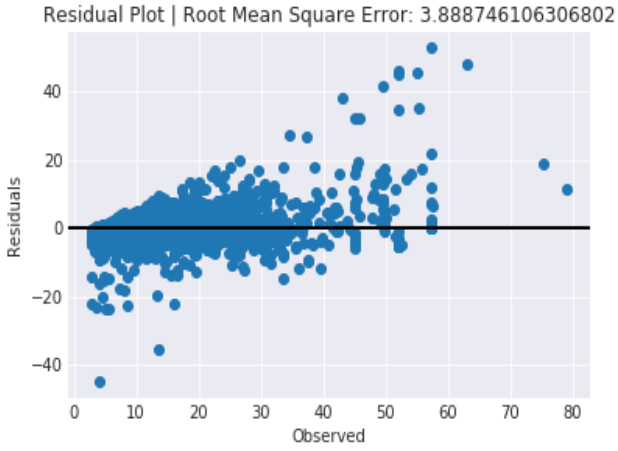
Model 2 – Decision Tree Regressor Model

```
y = df_dummy.fare_amount
X = df_dummy.drop(['fare_amount'],axis=1)

X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.30,random_state=42)

model= DecisionTreeRegressor()
model.fit(X_train,y_train)
y_pred_dec = model.predict(X_test)
```



Model 3 – Random Forest Regressor Model

```
y = df_dummy.fare_amount
X = df_dummy.drop(['fare_amount'],axis=1)

X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.30,random_state=42)

model= RandomForestRegressor()
model.fit(X_train,y_train)
y_pred_rf = model.predict(X_test)
```



4.1 Model Selection

We'll do a performance test of all three models with the same number of variables. Let's check how the model behaves by checking its Root Mean Squared Error . We see that we have an error in Linear Regression is 4.177 , Random Forest Regressor is 3.888 and Decision Tree Regressor is 5.304.

Random Forest Regressor algorithm is good, because it has less Root Mean Squared Error compared to other models.

```
#Predict from test set
y_train = train.fare_amount
X_train = train.drop(['fare_amount'],axis=1)

X_test = test

model= RandomForestRegressor()
model.fit(X_train,y_train)
y_pred = model.predict(X_test)
```

```
prediction = pd.DataFrame(y_pred, columns=['Fare Amount']).to_csv('Cabfare.csv',index=False)
```

Python Code:

 # In[1]:

 # Import all the required libraries

import pandas as pd

import matplotlib.pyplot as plt

import numpy as np

from sklearn.model_selection import train_test_split

from sklearn.tree import DecisionTreeRegressor

from sklearn.ensemble import RandomForestRegressor,ExtraTreesRegressor

from sklearn.metrics import mean_absolute_error

import seaborn as sns

from sklearn.metrics import accuracy_score

import statsmodels.api as sm

from sklearn.linear_model import LinearRegression

from sklearn.metrics import mean_squared_error, r2_score

```
import datetime
import os
# In[2]:
# Set current directory
newpath = os.chdir("/home/someshugar/Project2")
dirpath = os.getcwd()
print("current directory is : " + dirpath)
# In[3]:
# Get file path
Train_file = "/home/someshugar/Project2/train_cab.csv"
Test_file = "/home/someshugar/Project2/test.csv"

# In[4]:
# Read file  using pandas and check shape
Train_data = pd.read_csv(Train_file)
print(Train_data.shape)
Test_data = pd.read_csv(Test_file)
print(Test_data.shape)
# In[5]:
# View and analyse head of the Dataframe
Train_data.head()
# In[6]:
Test_data.head()
# In[7]:
Train_data.info()
# In[8]:
Train_data.isnull().sum()
# In[9]:
Train_data.describe()
# In[10]:
Test_data.info()
# In[11]:
Test_data.describe()
```

```
# In[12]:
Train_data.passenger_count.value_counts()
# In[13]:
```

Remove invalid passenger_count values

```
Train_data = Train_data[(Train_data.passenger_count>=1) & (Train_data.passenger_count<=6)]
# In[14]:
Train_data = Train_data.drop(Train_data[Train_data.passenger_count == 1.3].index, axis=0)
# In[15]:
Train_data.passenger_count.isnull().sum()
# In[16]:
Train_data.passenger_count.value_counts()
# In[17]:
```

 # Plot of Passengers Count

```
fig, ax = plt.subplots(figsize=(10,7))
Train_data.passenger_count.plot(kind='hist')
ax.set(xlabel='No. of Passengers', ylabel='Frequency', title='Passengers Count')
plt.savefig('Figure1.png')
plt.show()

# In[18]:
```

Replace – and convert 'fare_amount' as type float

```
Train_data['fare_amount'] = Train_data['fare_amount'].map(lambda x :  str(x).replace("-",""))
Train_data['fare_amount'] = Train_data['fare_amount'].astype(float)
# In[19]:
Train_data.fare_amount.dtype
# In[20]:
Train_data.fare_amount.describe()
# In[21]:
Train_data.isnull().sum()
# In[22]:
```

Select Fare amount between quantile of 0.001 and 0.999

```
Train_data = Train_data[(Train_data.fare_amount>Train_data.fare_amount.quantile(.001)) &
                        (Train_data.fare_amount<Train_data.fare_amount.quantile(.999))]
# In[23]:
Train_data.fare_amount.describe()
# In[24]:
#Box Plot of fare_amount
get_ipython().run_line_magic('matplotlib', 'inline')
fig, ax = plt.subplots(figsize=(10,7))
Train_data['fare_amount'].plot(kind='box')
ax.set(title='Fare amount Box plot')
plt.savefig('Figure2.png')
plt.show()
# In[25]:
# Fare amount kde plot
get_ipython().run_line_magic('matplotlib', 'inline')
fig, ax = plt.subplots(figsize=(10,7))
Train_data['fare_amount'].plot(kind='kde')
ax.set(xlabel='fare_amount',title='Fare amount kde plot')
plt.savefig('Figure3.png')
plt.show()
# In[26]:
Train_data.info()
# In[27]:
# Date time conversion
Train_data['pickup_datetime'] = pd.to_datetime(Train_data['pickup_datetime'],format='%Y-%m-%d
%H:%M:%S %Z', errors='coerce')
# In[28]:
Train_data.isnull().sum()
# In[29]:
Train_data = Train_data.dropna()
# In[30]:
Train_data.pickup_datetime.dtype
# In[31]:
```

 # Date time conversion

Test_data['pickup_datetime'] = pd.to_datetime(Test_data['pickup_datetime'],format='%Y-%m-%d %H:%M:%S %Z',errors='coerce')

 # In[32]:

Test_data.pickup_datetime.dtype

 # In[33]:

 # Generate the five new time variables

Train_data['pickup_month'] = Train_data['pickup_datetime'].dt.month

Train_data['pickup_year'] = Train_data['pickup_datetime'].dt.year

Train_data['pickup_day'] = Train_data['pickup_datetime'].dt.day

Train_data['pickup_weekday'] = Train_data['pickup_datetime'].dt.weekday

Train_data['pickup_hour'] = Train_data['pickup_datetime'].dt.hour

 # In[34]:

Train_data = Train_data.drop('pickup_datetime',axis=1)

 # In[35]:

 # Generate the five new time variables

Test_data['pickup_month'] = Test_data['pickup_datetime'].dt.month

Test_data['pickup_year'] = Test_data['pickup_datetime'].dt.year

Test_data['pickup_day'] = Test_data['pickup_datetime'].dt.day

Test_data['pickup_weekday'] = Test_data['pickup_datetime'].dt.weekday

Test_data['pickup_hour'] = Test_data['pickup_datetime'].dt.hour

 # In[36]:

Test_data = Test_data.drop('pickup_datetime',axis=1)

 # In[37]:

Train_data.info()

 # In[38]:

Train_data.describe()

 # In[39]:

 # # Correlation Analysis

plt.figure(figsize=(10, 8))

sns.heatmap(Train_data.corr(),cmap=sns.diverging_palette(220, 20, n=7), vmax=1.0, vmin=-1.0, linewidths=0.1,

annot=True, annot_kws={"size": 8}, square=True)

```
plt.savefig('Figure4.png')
plt.show()



# In[40]:
```

Select ['pickup_longitude','pickup_latitude', 'dropoff_longitude', 'dropoff_latitude'] between quantile of 0.001 and 0.999

```
coord = ['pickup_longitude','pickup_latitude',
        'dropoff_longitude', 'dropoff_latitude']
for i in coord:
    Train_data = Train_data[(Train_data[i]   > Train_data[i].quantile(.001)) &
                  (Train_data[i] < Train_data[i].quantile(.999))]
```

#create new variable log of fare amount

```
Train_data["log_fare_amount"] = np.log(Train_data["fare_amount"])
 Train_data.head()
```

 # # Finding distances based on Latitude and Longitude

 # The haversine formula determines the great-circle distance between two points on a sphere given their longitudes and latitudes.

 # Formula

 # dlon = lon2 - lon1

#

 # dlat = lat2 - lat1

#

 # a = (sin(dlat/2))^2 + cos(lat1) cos(lat2) (sin(dlon/2))^2

 # i

 # c = 2 * atan2(sqrt(a), sqrt(1-a))

#

 # d = R * c (where R is the radius of the Earth)

```
# In[41]:
```

```
#radius of earth in kilometers
R = 6373.0

pickup_lat  = np.radians(Train_data["pickup_latitude"])
pickup_lon  = np.radians(Train_data["pickup_longitude"])
dropoff_lat = np.radians(Train_data["dropoff_latitude"])
dropoff_lon = np.radians(Train_data["dropoff_longitude"])

dist_lon = dropoff_lon - pickup_lon
dist_lat = dropoff_lat - pickup_lat

#Formula
a = (np.sin(dist_lat/2))**2 + np.cos(pickup_lat) * np.cos(dropoff_lat) * (np.sin(dist_lon/2))**2
c = 2 * np.arctan2( np.sqrt(a), np.sqrt(1-a) )
d = R * c #(where R is the radius of the Earth)

Train_data["trip_distance_km"] = d

#create new variable log of distance
Train_data["log_trip_distance"] = np.log(Train_data["trip_distance_km"])

Train_data[coord + ["trip_distance_km"]].head(7)


# In[42]:
pickup_lat  = np.radians(Test_data["pickup_latitude"])
pickup_lon  = np.radians(Test_data["pickup_longitude"])
dropoff_lat = np.radians(Test_data["dropoff_latitude"])
dropoff_lon = np.radians(Test_data["dropoff_longitude"])

dist_lon = dropoff_lon - pickup_lon
dist_lat = dropoff_lat - pickup_lat
```

```
#Formula
a = (np.sin(dist_lat/2))**2 + np.cos(pickup_lat) * np.cos(dropoff_lat) * (np.sin(dist_lon/2))**2
c = 2 * np.arctan2( np.sqrt(a), np.sqrt(1-a) )
d = R * c #(where R is the radius of the Earth)

Test_data["trip_distance_km"] = d

#create new variable log of distance
Test_data["log_trip_distance"] = np.log(Test_data["trip_distance_km"])
Test_data[coord + ["trip_distance_km"]].head(7)
# In[43]:
# Dist plot with mean of 'fare_amount','passenger_count', 'pickup_longitude',
'dropoff_longitude','pickup_latitude', 'dropoff_latitude'
import itertools
cols = ['fare_amount','passenger_count',
       'pickup_longitude', 'dropoff_longitude',
       'pickup_latitude', 'dropoff_latitude']
length = len(cols)
cs     = [(0.8941176470588236, 0.10196078431372549, 0.10980392156862745),
        (0.21568627450980393, 0.49411764705882355, 0.7215686274509804),
        (0.30196078431372547, 0.6862745098039216, 0.2901960784313726),
        (0.596078431372549, 0.3058823529411765, 0.6392156862745098),
        (1.0, 0.4980392156862745, 0.0),"b"]

sns.set_style("darkgrid")
plt.figure(figsize = (13,15))
for i,j,k in itertools.zip_longest(cols,range(length),cs) :
   plt.subplot(length/2,length/3,j+1)
   sns.distplot(Train_data[i],color = k)
   plt.axvline(Train_data[i].mean(),linewidth  = 2 ,
               linestyle = "dashed",color = "k" ,
               label = "Mean")
   plt.legend(loc = "best")
```

```
    plt.title(i,color = "b")
    plt.xlabel("")
plt.savefig('Figure5.png')

# In[44]:
#Distribution in log of fare amount
plt.figure(figsize = (12,7))
sns.distplot(Train_data["log_fare_amount"],color = "b")
plt.axvline(Train_data["log_fare_amount"].mean(),label = "Avg fare amount")
plt.title("Distribution in log of fare amount")
plt.legend()
plt.savefig('Figure6.png')
plt.show()

# In[45]:
#Distribution Plot in trip distance in kilometers
plt.figure(figsize = (12,7))
sns.distplot(Train_data["trip_distance_km"],color = "r")
plt.axvline(Train_data["trip_distance_km"].mean(),color = "k",
        linestyle = "dashed",label = "Avg trip distance (km)")
plt.title("Distribution in trip distance in kilometers")
plt.legend()
plt.savefig('Figure7.png')
plt.show()

# In[46]:
#Scatter plot for distance and fare amount
plt.figure(figsize = (12,10))
plt.scatter(Train_data["fare_amount"],
        Train_data["trip_distance_km"],s = 5,
        linewidths=1, c = "b")
plt.ylabel("distance in kilometers")
plt.xlabel("Fare amount")
```

```
plt.title("scatter plot for distance and fare amount")
plt.savefig('Figure8.png')
plt.show()


# In[47]:
# Barplot of trip_counts vs pickup_months of a year
yrs = [i for i in Train_data["pickup_year"].unique().tolist() if i not in [2015]]
#subset data without year 2015
complete_dat = Train_data[Train_data["pickup_year"].isin(yrs)]
plt.figure(figsize = (13,15))
for i,j in itertools.zip_longest(yrs,range(len(yrs))) :
    plt.subplot(3,2,j+1)
    trip_counts_mn = complete_dat[complete_dat["pickup_year"] == i]["pickup_month"].value_counts()
    trip_counts_mn = trip_counts_mn.reset_index()
    sns.barplot(trip_counts_mn["index"],trip_counts_mn["pickup_month"],
                palette = "rainbow",linewidth = 1,
                edgecolor = "k"*complete_dat["pickup_month"].nunique()
               )
    plt.title(i,color = "b",fontsize = 12)
    plt.grid(True)
    plt.xlabel("")
    plt.ylabel("trips")
plt.savefig('Figure9.png')

# In[48]:
#Plot of trips_hr["pickup_hour"] vs trips_hr["count"]
fig, ax = plt.subplots(figsize=(12,8))
trips_hr = Train_data["pickup_hour"].value_counts().reset_index()
trips_hr.columns = ["pickup_hour","count"]
trips_hr = trips_hr.sort_values(by = "pickup_hour",ascending = True)

plt.plot(trips_hr["pickup_hour"],trips_hr["count"],'-o')
```

```
ax.set(title = "Trend in trips  by hour of day",
                xlabel = "hour",
                ylabel = "count")
plt.savefig('Figure10.png')


# In[49]:
#Plot of  Trend in trips by avg fare and hour of day
avg_fare_hr = Train_data.groupby("pickup_hour")["fare_amount"].mean().reset_index()
fig, ax = plt.subplots(figsize=(12,8))

plt.plot(avg_fare_hr["pickup_hour"],avg_fare_hr["fare_amount"],'-o')
ax.set(title = "Trend in trips by avg fare and hour of day",
                xlabel = "pickup_hour",
                ylabel = "fare_amount")
plt.savefig('Figure11.png')

# In[50]:
Train_data.columns
# In[51]:
train = Train_data
# In[52]:
Drop log_fare_amount' and 'log_trip_distance'
train = train.drop(['log_fare_amount','log_trip_distance'],axis=1)
# In[53]:
train.shape
# In[56]:
train.shape
# In[57]:
train.columns
```

 # # LinearRegression Model

 # In[58]:

```
y = train.fare_amount
X = train.drop(['fare_amount'],axis=1)

X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.30,random_state=42)

model= LinearRegression()
model.fit(X_train,y_train)
y_pred_lm = model.predict(X_test)
```

 # In[59]:

 # Plot the residuals

```
residuals = y_test-y_pred_lm
fig, ax = plt.subplots()
ax.scatter(y_test, residuals)
ax.axhline(lw=2,color='black')
ax.set_xlabel('Observed')
ax.set_ylabel('Residuals')
ax.title.set_text('Residual Plot | Root Mean Squared Error: ' + str(mean_squared_error(y_test,
y_pred_lm) ** 0.5))
plt.savefig('Figure12.png')
plt.show()
```

 # In[60]:

```
df_dummy = train
```

 # In[61]:

```
df_dummy.head()
```

 # # DecisionTreeRegressor

 # In[62]:

```
y = df_dummy.fare_amount
X = df_dummy.drop(['fare_amount'],axis=1)

X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.30,random_state=42)
```

```
model= DecisionTreeRegressor()
model.fit(X_train,y_train)
y_pred_dec = model.predict(X_test)
# In[63]:
#Residual Plot
residuals = y_test-y_pred_dec
fig, ax = plt.subplots()
ax.scatter(y_test, residuals)
ax.axhline(lw=2,color='black')
ax.set_xlabel('Observed')
ax.set_ylabel('Residuals')
ax.title.set_text('Residual Plot | Root Mean Square Error: ' + str(mean_squared_error(y_test, y_pred_dec) ** 0.5))
plt.savefig('Figure13.png')
plt.show()


# # RandomForestRegressor
# In[64]:
y = df_dummy.fare_amount
X = df_dummy.drop(['fare_amount'],axis=1)

X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.30,random_state=42)

model= RandomForestRegressor()
model.fit(X_train,y_train)
y_pred_rf = model.predict(X_test)
# In[65]:
#Residual Plot
residuals = y_test-y_pred_rf
fig, ax = plt.subplots()
ax.scatter(y_test, residuals)
```

ax.axhline(lw=2,color='black')

ax.set_xlabel('Observed')

ax.set_ylabel('Residuals')

*ax.title.set_text('Residual Plot | Root Mean Square Error: ' + str(mean_squared_error(y_test, y_pred_rf) ** 0.5))*

plt.savefig('Figure14.png')

plt.show()

#Model Selection

 # Hence Random forest has less RMSE we choose Random forest model to make Predictions from test

 # In[66]:

Test_data.columns

 # In[67]:

 # In[68]:

test = Test_data.drop(['log_trip_distance'],axis=1)

 # In[69]:

train.columns

 # In[70]:

test.columns

 # In[71]:

train.shape

 # In[72]:

test.shape

 # In[74]:

#Predict from test set

y_train = train.fare_amount

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

X_test = test

model= RandomForestRegressor()

model.fit(X_train,y_train)

y_pred = model.predict(X_test)

 # In[75]:

prediction = pd.DataFrame(y_pred, columns=['Fare Amount']).to_csv('Cabfare.csv',index=False)

 # In[76]:

Write result to Cabfare.csv

Cabfare = pd.read_csv("Cabfare.csv")

print(Cabfare.shape)

print(Cabfare.head())