

Cab Fare Prediction

Abstraction:

In this project we need to predict the fare_amount of the cab using attributes like drop-off latitude, drop-off longitude, pickup date-time, passenger_count, pickup longitude, pickup latitude. Our project helps the organisation to take the right decisions, so that the organisation may run successfully without any losses.

Here we used R studio and Jupyter Notebook as platforms to work on the project. At first, we need to clean data using Exploratory Data Analysis like check for Missing Values, Outliners, then Feature Extraction, Feature Selection, Feature Scaling. Achieving the goal was quite challenging. Using the help of Machine Learning Algorithms like Multi Linear Regression, Random Forest, Decision Tree, enhance the probability of reaching goal early.

Project Index

TOPIC	Page number
1. Introduction	3
2. Problem Statement	3 - 4
3. Loading Data	4
4. Data Visualisation & Processing	5 - 7
5. Missing Value Analysis	8 - 10
6. Outliner Analysis	11 - 12
7. Feature Extraction	12 - 16
8. Feature Selection	16 - 21
9. Feature Scaling	22 - 23
10. Splitting train and test data	24
11. Model Development & Testing	25 - 26
12. Conclusion	27

1. INTRODUCTION:

Cab rental system is becoming a new frontier in business, particularly in major cities all over the world. There are many cab rental organisations who are competing with one another in race to achieve profit and fame.

In this project, our objective is to predict the cab fare-amount. The test data contains 16067 observations and 7 variables i.e fare-amount which is our target variable, pickup latitude, pickup longitude, drop-off latitude, drop-off longitude, passenger-count, pickup date-time.

Our aim is to develop a model that helps in predicting the fare amount, for future references using the past data.

2. 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.

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.

3. Loading Data in R:

After installing important packages.

Here we are using R studio and we are loading data into R-environment. Using following command:

- `Original_train=pd.read_csv("E:/Sproject/train_cab.csv",header=T)`
- `Original_test =pd.read_csv("E:/Sproject/test.csv",header=T)`

The screenshot shows a Jupyter Notebook with the following code and output:

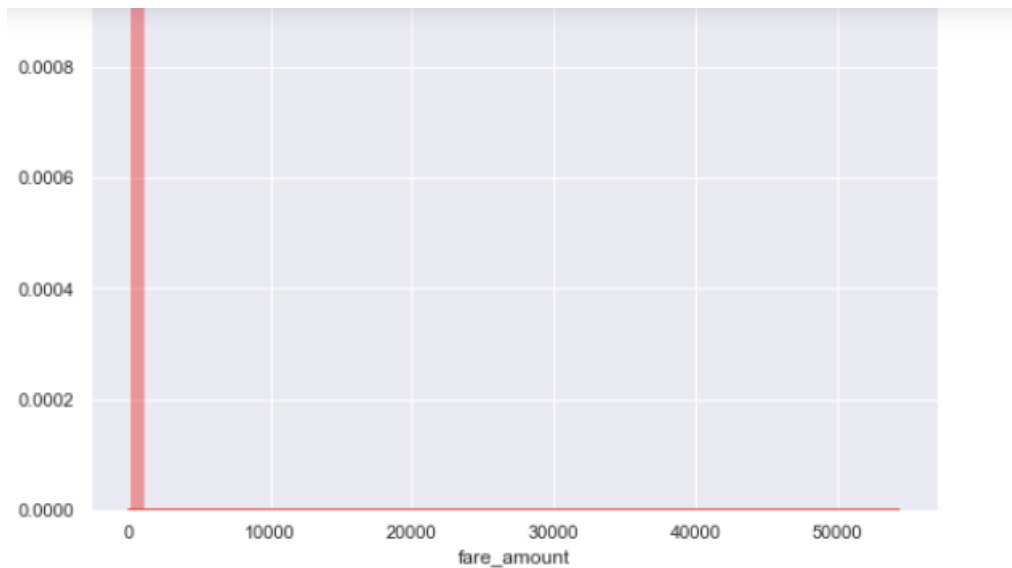
```
In [2]: train = pd.read_csv('E:/Sproject/train_cab.csv', dtype={'fare_amount': np.float64}, na_values={'fare_amount': '430'})
In [3]: j = train
In [4]: test = pd.read_csv("E:/Sproject/test.csv")
In [5]: k = test
In [6]: train.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 16067 entries, 0 to 16066
Data columns (total 7 columns):
 fare_amount      16042 non-null float64
 pickup_datetime  16067 non-null object
 pickup_longitude 16067 non-null float64
 pickup_latitude  16067 non-null float64
 dropoff_longitude 16067 non-null float64
 dropoff_latitude 16067 non-null float64
 passenger_count  16012 non-null float64
dtypes: float64(6), object(1)
memory usage: 878.7+ KB

In [7]: test.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9914 entries, 0 to 9913
Data columns (total 6 columns):
 pickup_datetime  9914 non-null object
 pickup_longitude 9914 non-null float64
 pickup_latitude  9914 non-null float64
 dropoff_longitude 9914 non-null float64
 dropoff_latitude 9914 non-null float64
```

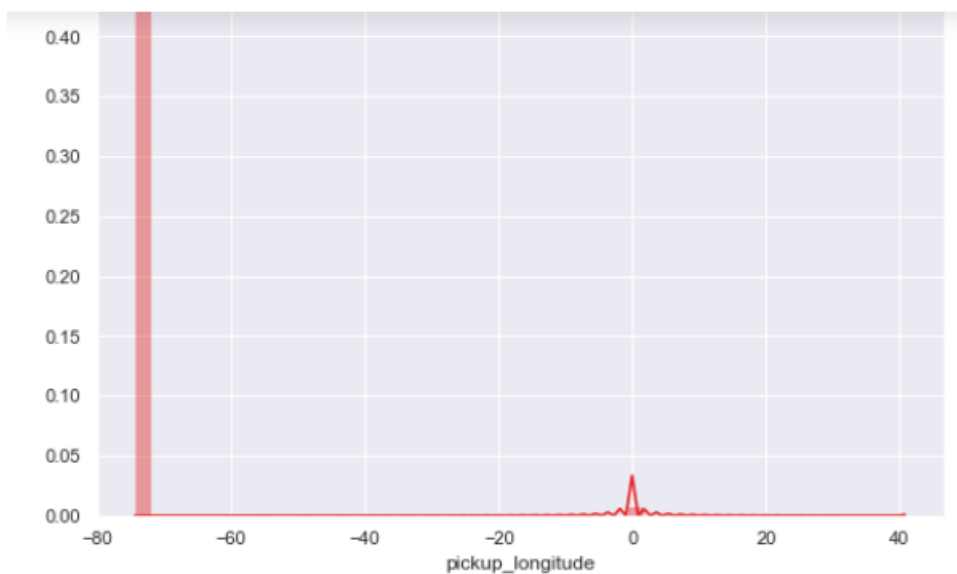
4. Data Visualisation & Processing:

Some histogram plots on different variables of train data.

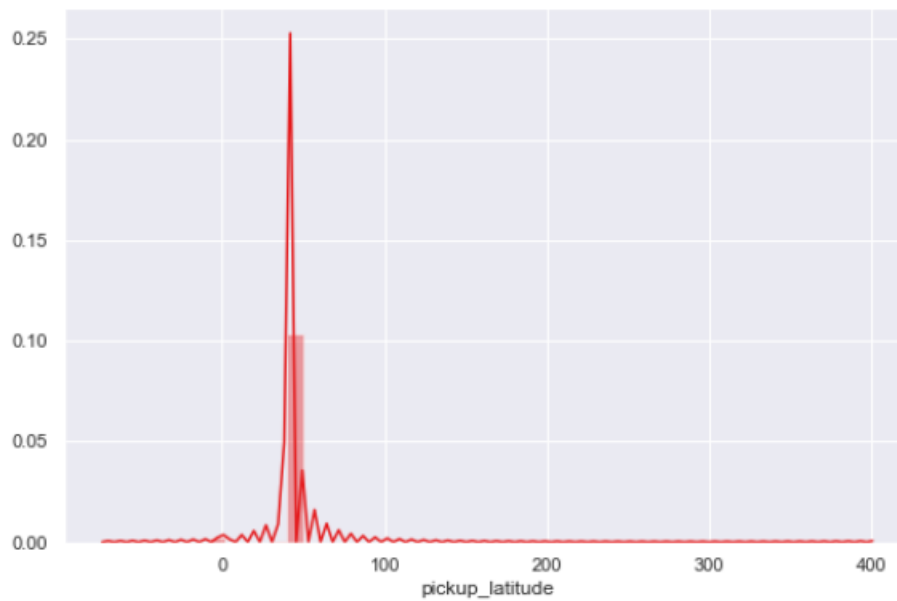
Histogram plot on fare-amount:



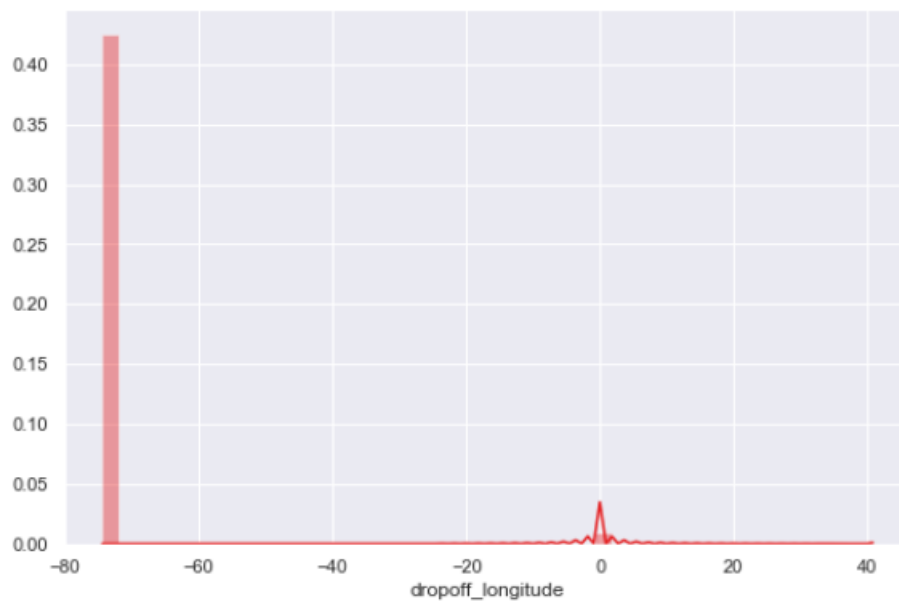
Histogram plot on pickup-Longitude:



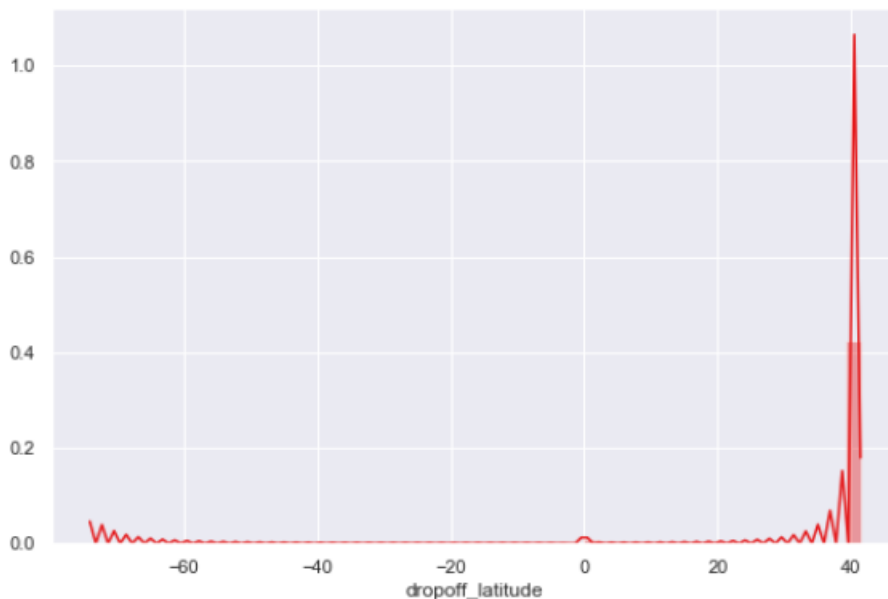
Histogram plot on pickup-latitude:



Histogram plot on drop off-longitude:



Histogram plot on drop off latitude:



4. Data Processing / Exploratory Data Analysis:

- I. Fare amount has some negative values. it is also having 0 values, so we need to remove these fields. We used following code to overcome this:

```
sum(j['fare_amount']<1)
```

here j is original train data. There are 5 negative values which are less than 1. We need to remove them.

- II. The passenger count in a cab should be below 6, but there are values more than 6 values.

```
j[j['passenger_count']>6]
```

There are 20 values which are greater than 6.

- III. We need to check for negative values in passenger-count variable using following command:

```
j[j['passenger_count']<1]
```

There are 58 negative values.

Now we are removing the negative values from fare-amount, passenger-count and the values greater than 6 from passenger-count variables, using following commands:

- `j = j.drop(j[j['fare_amount']<1].index, axis=0)`
- `j = j.drop(j[j['passenger_count']>6].index, axis=0)`
- `j = j.drop(j[j['passenger_count']<1].index, axis=0)`

5. Missing value Analysis:

Here we are check for missing values in the dataset like empty rows which was filled with Na. we found some missing values in our dataset. Now missing values can be found by using different techniques like mean, median and Knn imputation.

First, we are selecting 1000 rows randomly and performing mean, median, Knn imputation, so that we can choose any one by comparing actual value and predicted value.

There are 3 types of methods to predict the missing values.

1. MEAN method
2. MEDIAN method
3. KNN

We need to decide which method is suitable for our dataset, for that we are randomly taking actual value of observation of passenger count , fare_amount variables and check the predicted values of above 3 methods with actual value and decide which method is suitable .

1. For passenger-count variable the location, of 121 observation has actual value =1, here for this variable we can only use knn imputation as, it is not suitable for mean and median method.
2. For fare-amount variable of location 121, the actual value is 4, the mean value is 15, the median value is 8, but knn value is 3 which is nearer. So, we use knn method.

From above experience we decide to use knn method.

Here are the missing value percentage for variables of dataset.

```

missing_val
missing_val['Missing_percentage'] = (missing_val['Missing_percentage']/len(train))*100
missing_val = missing_val.sort_values('Missing_percentage', ascending = False).reset_index(drop = True)

In [32]: missing_val
Out[32]:

```

	Variables	Missing_percentage
0	passenger_count	0.342317
1	fare_amount	0.136927
2	pickup_datetime	0.006224
3	pickup_longitude	0.000000
4	pickup_latitude	0.000000
5	dropoff_longitude	0.000000
6	dropoff_latitude	0.000000

There are 3 types of methods to predict the missing values.

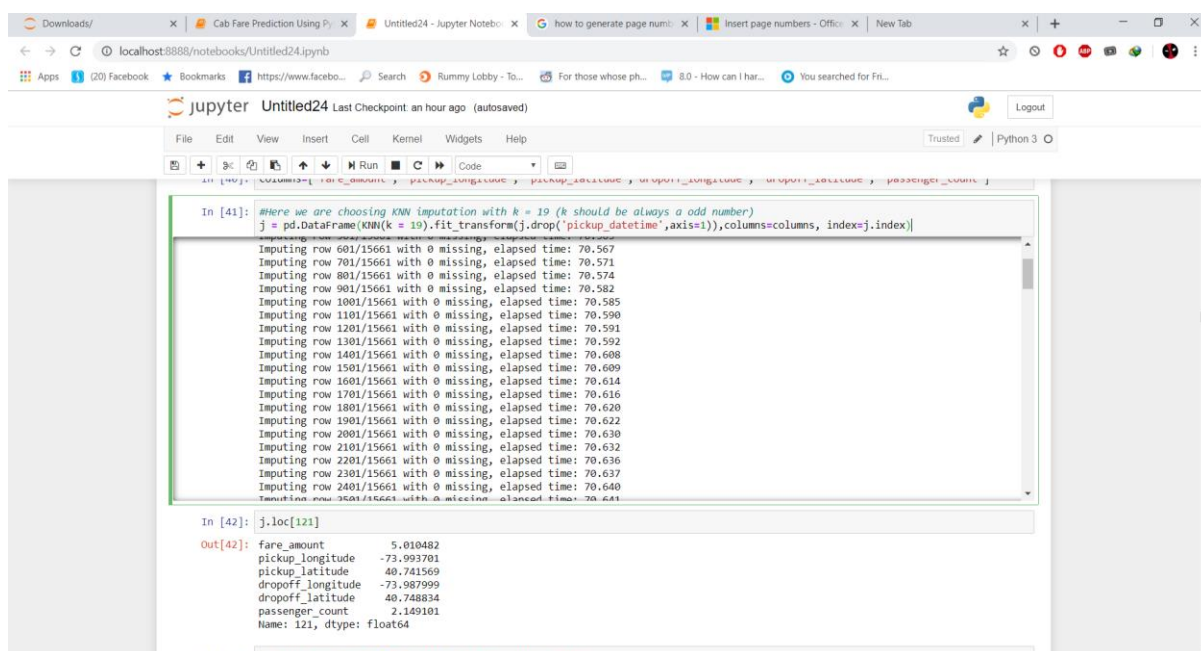
1. MEAN method
2. MEDIAN method
3. KNN

We need to decide which method is suitable for our dataset, for that we are randomly taking actual value of a observation of passenger count, fare_amount variables and check the predicted values of above 3 methods with actual value and decide which method is suitable.

As passenger-count variable has 34% of missing values & fare-amount has 14% of missing values.

We are now using knn imputation to replace missing values, using following command:

```
j=pd.DataFrame(KNN(k=19).fit_transform(j.drop('pickup_datetime',axis=1)),columns=columns, index=j.index)
```



```

In [41]: #Here we are choosing KNN imputation with k = 19 (k should be always a odd number)
j = pd.DataFrame(KNN(k = 19).fit_transform(j.drop('pickup_datetime',axis=1)),columns=columns, index=j.index)

Imputing row 601/15661 with 0 missing, elapsed time: 70.567
Imputing row 701/15661 with 0 missing, elapsed time: 70.571
Imputing row 801/15661 with 0 missing, elapsed time: 70.574
Imputing row 901/15661 with 0 missing, elapsed time: 70.582
Imputing row 1001/15661 with 0 missing, elapsed time: 70.585
Imputing row 1101/15661 with 0 missing, elapsed time: 70.590
Imputing row 1201/15661 with 0 missing, elapsed time: 70.591
Imputing row 1301/15661 with 0 missing, elapsed time: 70.592
Imputing row 1401/15661 with 0 missing, elapsed time: 70.608
Imputing row 1501/15661 with 0 missing, elapsed time: 70.609
Imputing row 1601/15661 with 0 missing, elapsed time: 70.614
Imputing row 1701/15661 with 0 missing, elapsed time: 70.616
Imputing row 1801/15661 with 0 missing, elapsed time: 70.620
Imputing row 1901/15661 with 0 missing, elapsed time: 70.622
Imputing row 2001/15661 with 0 missing, elapsed time: 70.630
Imputing row 2101/15661 with 0 missing, elapsed time: 70.632
Imputing row 2201/15661 with 0 missing, elapsed time: 70.636
Imputing row 2301/15661 with 0 missing, elapsed time: 70.637
Imputing row 2401/15661 with 0 missing, elapsed time: 70.640
Imputing row 2501/15661 with 0 missing, elapsed time: 70.641

In [42]: j.loc[121]
Out[42]: fare_amount      5.010482
pickup_longitude -73.993701
pickup_latitude   40.741569
dropoff_longitude -73.987999
dropoff_latitude  40.748834
passenger_count    2.149101
name: 121, dtype: float64

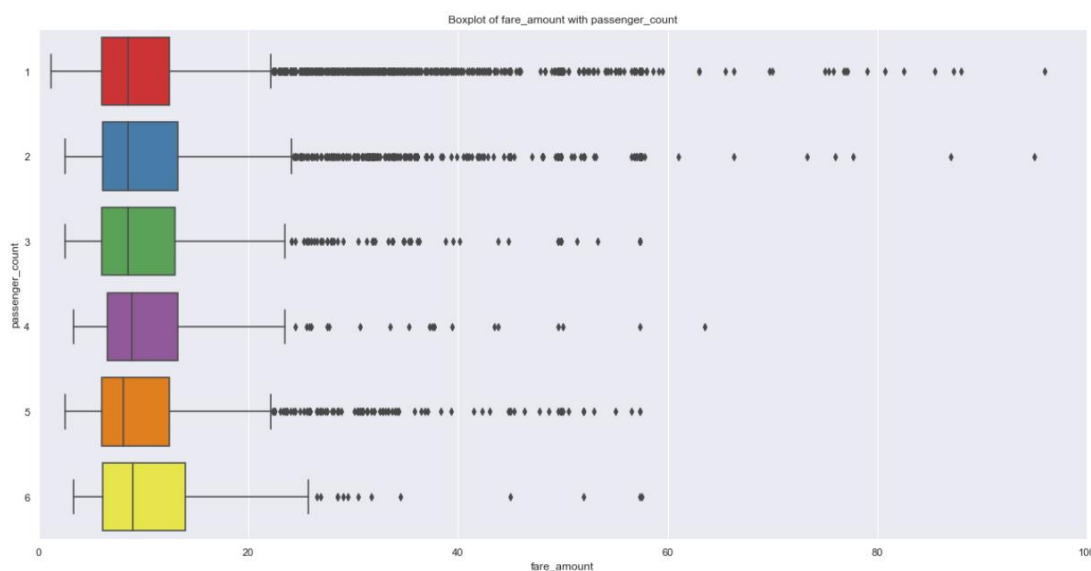
```

6. Outlier Analysis:

Here we are performing **Outlier Analysis** only on fare-amount which is our dependent variable.

Using the below code, we are representing the outlier graphically.

```
plt.figure(figsize=(20,10))
plt.xlim(0,100)_ =
sns.boxplot(x=j['fare_amount'],y=j['passenger_count'],data=j,orient='
h')
plt.title('Boxplot of fare_amount with passenger_count')
plt.show()
```



We have found 1358 outliers in fare-amount variable.

We have successfully replaced them with NA and removed them using KNN imputation.

Imputing with missing values using KNN, using following command:

```
j = pd.DataFrame(KNN(k = 3).fit_transform(j), columns = j.columns,
index=j.index)
```

```
In [55]: #Imputing with missing values using KNN
j = pd.DataFrame(KNN(k = 3).fit_transform(j), columns = j.columns, index=j.index)
```

```
Imputing row 1/15661 with 0 missing, elapsed time: 69.084
Imputing row 101/15661 with 0 missing, elapsed time: 69.091
Imputing row 201/15661 with 0 missing, elapsed time: 69.094
Imputing row 301/15661 with 1 missing, elapsed time: 69.094
Imputing row 401/15661 with 0 missing, elapsed time: 69.109
Imputing row 501/15661 with 0 missing, elapsed time: 69.114
Imputing row 601/15661 with 1 missing, elapsed time: 69.117
Imputing row 701/15661 with 0 missing, elapsed time: 69.117
Imputing row 801/15661 with 0 missing, elapsed time: 69.130
Imputing row 901/15661 with 0 missing, elapsed time: 69.131
Imputing row 1001/15661 with 0 missing, elapsed time: 69.131
Imputing row 1101/15661 with 1 missing, elapsed time: 69.131
Imputing row 1201/15661 with 0 missing, elapsed time: 69.131
Imputing row 1301/15661 with 0 missing, elapsed time: 69.147
Imputing row 1401/15661 with 0 missing, elapsed time: 69.147
Imputing row 1501/15661 with 0 missing, elapsed time: 69.147
Imputing row 1601/15661 with 0 missing, elapsed time: 69.147
Imputing row 1701/15661 with 0 missing, elapsed time: 69.147
Imputing row 1801/15661 with 0 missing, elapsed time: 69.147
Imputing row 1901/15661 with 0 missing, elapsed time: 69.163
```

7. Feature Extraction:

As we know that pickup date-time is a categorical variable of type time-series. We can extract year, month, day from pickup date-time. We can use following command to extract above variables:

```
data = [j,k]
```

```
for i in data:
```

```
    i["year"] = i["pickup_datetime"].apply(lambda row: row.year)
```

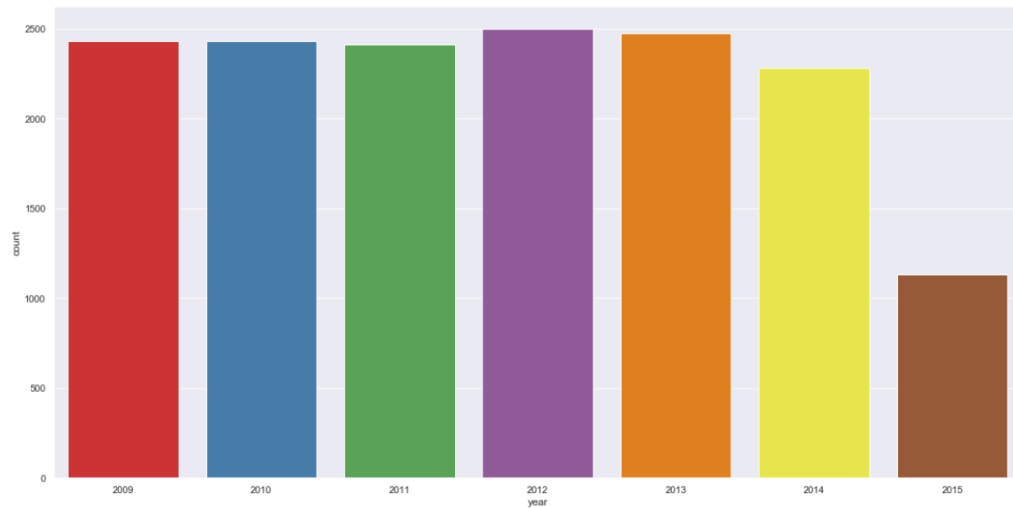
```
    i["month"] = i["pickup_datetime"].apply(lambda row: row.month)
```

```
    i["day_of_week"] = i["pickup_datetime"].apply(lambda row:
row.dayofweek)
```

```
    i["hour"] = i["pickup_datetime"].apply(lambda row: row.hour)
```

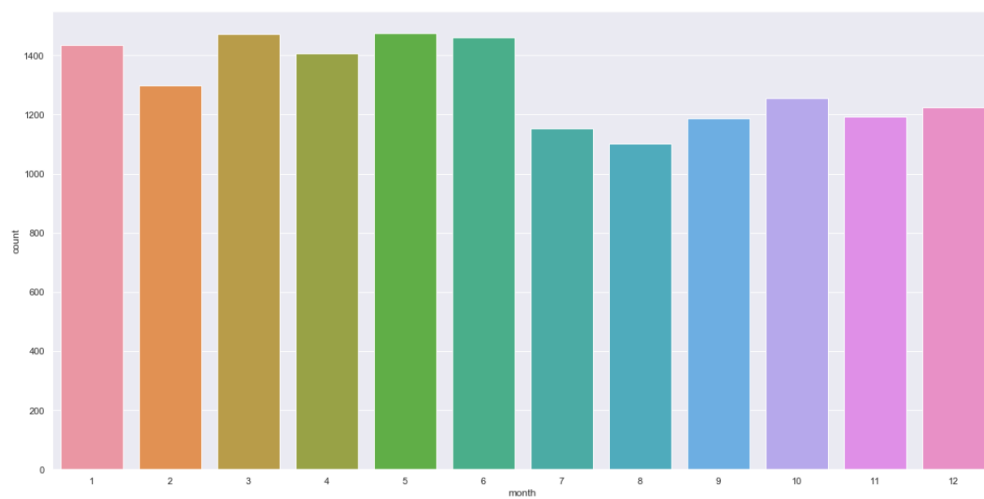
Plots for year:

```
Out[68]: <matplotlib.axes._subplots.AxesSubplot at 0x257fc3add30>
```

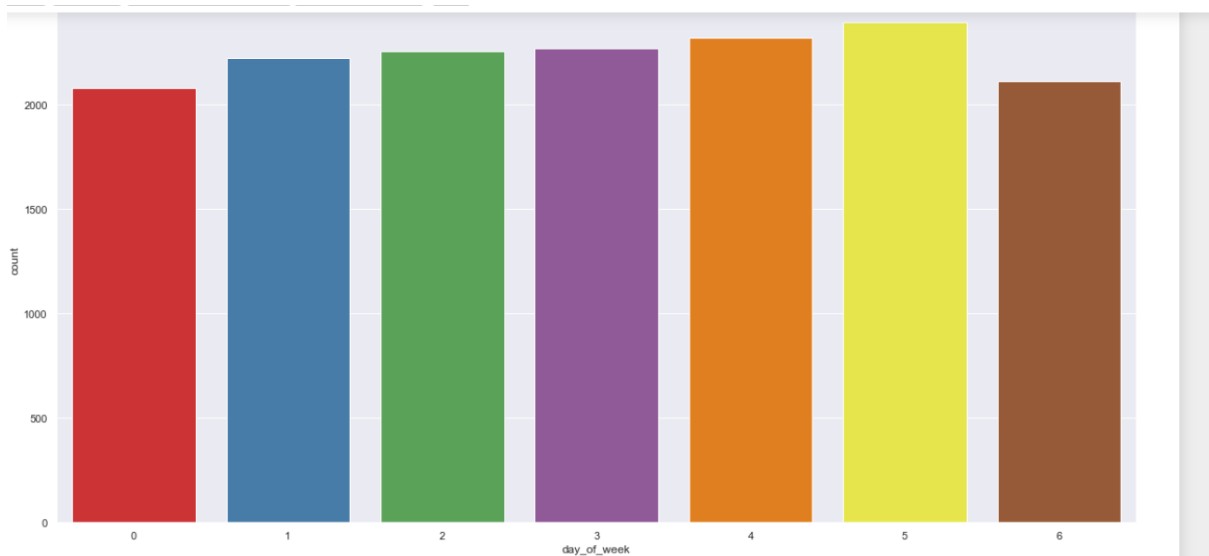


Plots for month:

```
Out[69]: <matplotlib.axes._subplots.AxesSubplot at 0x257fd3ace10>
```

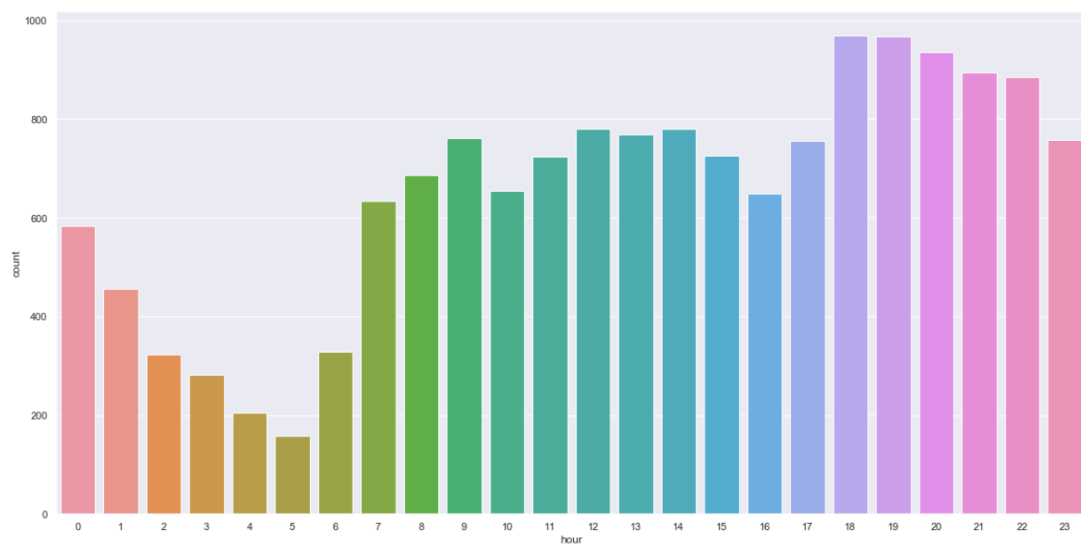


Plots for day:



Plots for hour:

jt[71]: <matplotlib.axes._subplots.AxesSubplot at 0x257fd42ddd8>



We need to calculate distance using longitudes and latitudes:

Here we calculate the distance from drop-off longitude and latitude, pickup longitude and pickup latitude. Using following command:

```
data = [j,k]
```

```
for i in data:
```

```
    i['great_circle']=i.apply(lambda x:
great_circle((x['pickup_latitude'],x['pickup_longitude']),
(x['dropoff_latitude'], x['dropoff_longitude'])).miles, axis=1)
```

```
    i['geodesic']=i.apply(lambda x:
geodesic((x['pickup_latitude'],x['pickup_longitude']),
(x['dropoff_latitude'], x['dropoff_longitude'])).miles, axis=1)
```

We create a new variable called distance from above longitudes and latitudes, here is the box plot for distance.



Now we are dropping unwanted variables like pickup longitude, pickup latitude, drop off longitude, drop off latitude. Using following commands:

```
j=j.drop(['pickup_datetime','pickup_longitude', 'pickup_latitude',  
         'dropoff_longitude', 'dropoff_latitude', 'passenger_count', 'year',  
         'month', 'day_of_week', 'hour', 'session', 'seasons',  
         'week','great_circle'],axis=1)
```

8. Feature Selection:

In this stage we select the variables which are used for target variable prediction. The variables which are not relevant can be removed, by doing so, we can create model which hold high accuracy of data prediction.

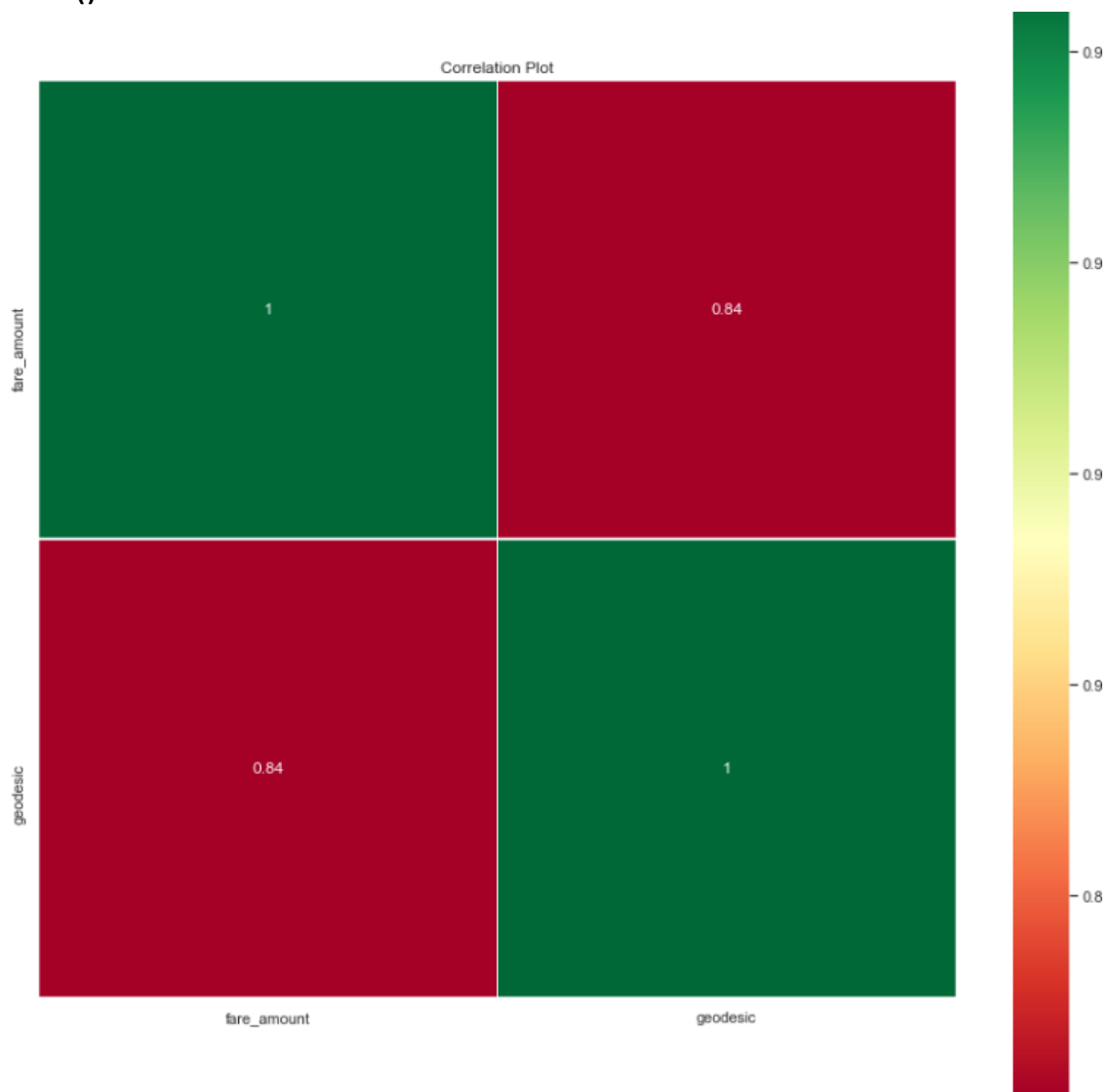
As our dataset contains both categorical and numerical variables. We use **Correlation** for numeric data and **Anova & Chi-square test** for categorical data.

We conduct correlation on numerical data i.e fare-amount and passenger-count.

Correlation Analysis: It helps to find the correlation between two independent variables. The correlation between independent variable and dependent variable must be high. If two independent variables are correlated to each other, then we need to choose any one of it.

We correlate numeric variables using following command:

```
plt.figure(figsize=(15,15))  
  
_ = sns.heatmap(j[num_var].corr(), square=True,  
cmap='RdYlGn',linewidths=0.5,linecolor='w',annot=True)  
  
plt.title('Correlation Plot')  
  
plt.show()
```



Anova: Here for categorical variables, we are using anova. If p-value is greater than 0.05 then we accept our Null hypothesis saying that two variables are independent. If p-values is less than 0.05, we reject the Null hypothesis saying that two variables are dependent.

We using following command to conduct Anova test on categorical variables

```
model = ols('fare_amount ~
C(passenger_count_2)+C(passenger_count_3)+C(passenger_count_4
)+C(passenger_count_5)+C(passenger_count_6)+C(season_spring)+C
(season_summer)+C(season_winter)+C(week_weekend)+C(session_n
ight_AM)+C(session_night_PM)+C(session_evening)+C(session_morn
ing)+C(year_2010)+C(year_2011)+C(year_2012)+C(year_2013)+C(yea
r_2014)+C(year_2015)',data=j).fit()
```

```
aov_table = sm.stats.anova_lm(model)
```

	df	sum_sq	mean_sq	F	PR(>F)
C(passenger_count_2)	1.0	10.211738	10.211738	0.527319	4.677473e-01
C(passenger_count_3)	1.0	17.010131	17.010131	0.878377	3.486610e-01
C(passenger_count_4)	1.0	63.864841	63.864841	3.297883	6.938839e-02
C(passenger_count_5)	1.0	21.361766	21.361766	1.103089	2.936054e-01
C(passenger_count_6)	1.0	145.705549	145.705549	7.524011	6.095127e-03
C(season_spring)	1.0	28.909259	28.909259	1.492830	2.217963e-01
C(season_summer)	1.0	26.961287	26.961287	1.392240	2.380452e-01
C(season_winter)	1.0	482.175550	482.175550	24.898807	6.106870e-07
C(week_weekend)	1.0	131.096812	131.096812	6.769639	9.280899e-03
C(session_night_AM)	1.0	2129.756471	2129.756471	109.977362	1.205681e-25
C(session_night_PM)	1.0	185.962424	185.962424	9.602815	1.946239e-03
C(session_evening)	1.0	0.981165	0.981165	0.050666	8.219116e-01
C(session_morning)	1.0	48.835012	48.835012	2.521765	1.123050e-01
C(year_2010)	1.0	1507.742306	1507.742306	77.857504	1.222792e-18
C(year_2011)	1.0	1332.103411	1332.103411	68.787780	1.184971e-16
C(year_2012)	1.0	431.200585	431.200585	22.266538	2.393599e-06
C(year_2013)	1.0	341.393053	341.393053	17.629014	2.699462e-05
C(year_2014)	1.0	1496.720279	1496.720279	77.288343	1.628849e-18
C(year_2015)	1.0	2587.829762	2587.829762	133.631566	8.773536e-31
Residual	15640.0	302874.978066	19.365408	NaN	NaN

- I. It is carried out to compare between each group in a categorical variable.
- II. ANOVA only lets us know the means for different groups are same or not. It doesn't help us identify which mean is different.

Null Hypothesis: mean of all categories in a variable are same.

Alternate Hypothesis: mean of at least one category in a variable is different.

If p-value is less than 0.05 then we reject the null hypothesis.

And if p-value is greater than 0.05 then we accept the null hypothesis.

Checking for multi-collinearity: It is the process of having collinearity that means two or more independent variables are co-related with each other. We use following command to check multi-collinearity between the variables.

```
outcome, predictors = dmatrices('fare_amount ~
geodesic+passenger_count_2+passenger_count_3+passenger_count_4+passenger_count_5+passenger_count_6+season_spring+season_summer+season_winter+week_weekend+session_night_AM+session_night_PM+session_evening+session_morning+year_2010+year_2011+year_2012+year_2013+year_2014+year_2015',j,
return_type='dataframe')
```

calculating VIF for each individual Predictors

```
vif = pd.DataFrame()
```

```
vif["VIF"] = [variance_inflation_factor(predictors.values, i) for i in
range(predictors.shape[1])]
```

```
vif["features"] = predictors.columns
```

```
vif
```

VIF is always greater or equal to 1.

- i. if VIF is 1, then Not correlated to any of the variables.
- ii. if VIF is between 1 to 5, then Moderately correlated.
- iii. if VIF is above 5, then Highly correlated. If there are multiple variables with VIF greater than 5, only remove the variable with the highest VIF.

	VIF	features
0	15.268469	Intercept
1	1.040740	passenger_count_2[T.1.0]
2	1.019517	passenger_count_3[T.1.0]
3	1.011841	passenger_count_4[T.1.0]
4	1.024998	passenger_count_5[T.1.0]
5	1.017210	passenger_count_6[T.1.0]
6	1.642257	season_spring[T.1.0]
7	1.552425	season_summer[T.1.0]
8	1.587591	season_winter[T.1.0]
9	1.050819	week_weekend[T.1.0]
10	1.376205	session_night_AM[T.1.0]
11	1.423280	session_night_PM[T.1.0]
12	1.524792	session_evening[T.1.0]
13	1.559076	session_morning[T.1.0]
14	1.691361	year_2010[T.1.0]
15	1.687796	year_2011[T.1.0]
16	1.711102	year_2012[T.1.0]
17	1.709339	year_2013[T.1.0]
18	1.665000	year_2014[T.1.0]
19	1.406917	year_2015[T.1.0]
20	1.025428	geodesic

9. Feature Scaling:

It helps in scaling/measuring data on same units. As, we are aware that different dataset contains different observations of different units, in order to scale them on same units, we use scaling.

Normalisation: It is calculated by dividing the data by its length. It ranges from 0 to 1. It can be checked by using following command:

for i in names:

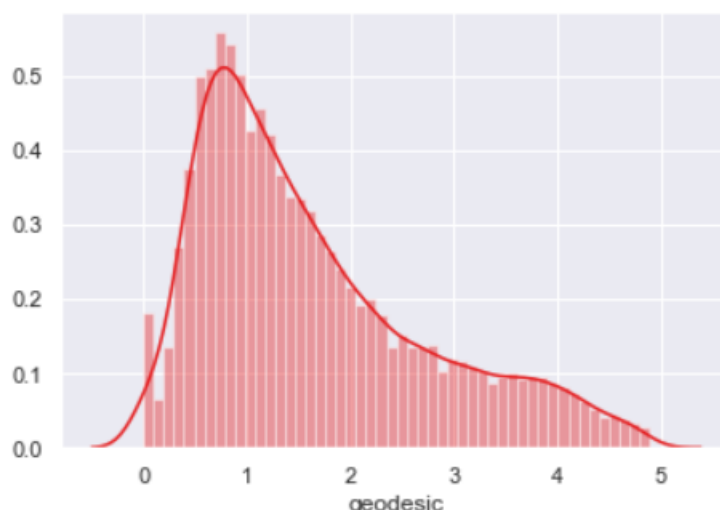
 print(i)

 j[i] = (j[i] - j[i].min())/(j[i].max() - j[i].min())

geodisc before Normalisation:

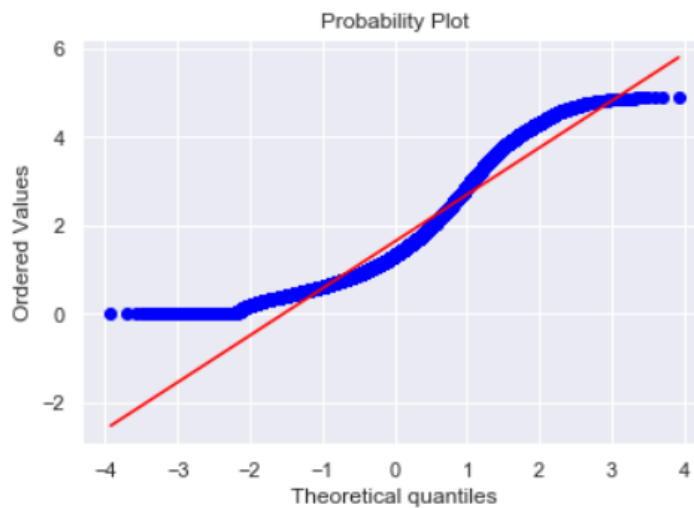
```
In [109]: sns.distplot(j['geodesic'],bins=50)
```

```
Out[109]: <matplotlib.axes._subplots.AxesSubplot at 0x257fd4d8cc0>
```



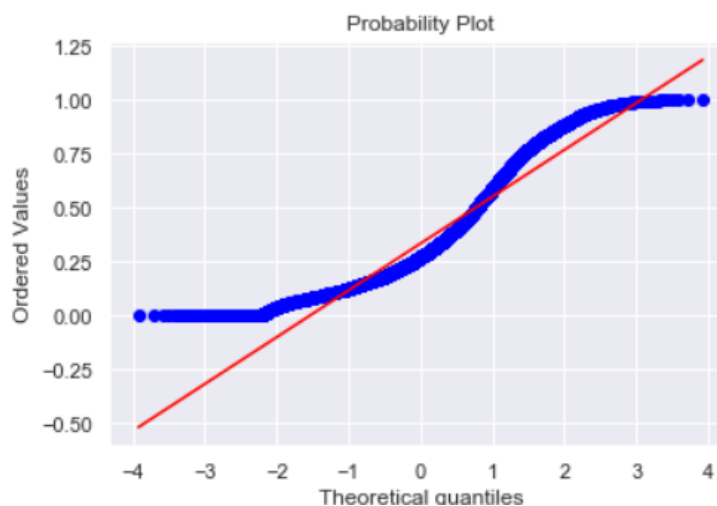
qq probability before Normalisation:

```
Out[110]: ((array([-3.92007182, -3.70085568, -3.58076887, ...,  3.58076887,
                  3.70085568,  3.92007182])),
          array([0.          , 0.          , 0.          , ..., 4.88300333, 4.88468361,
                  4.88875044])),
          (1.061852615888654, 1.6274749563025295, 0.956311159369912))
```



qq probability after Normalisation:

```
Out[115]: ((array([-3.92007182, -3.70085568, -3.58076887, ...,  3.58076887,
                  3.70085568,  3.92007182])),
          array([0.          , 0.          , 0.          , ..., 0.99882442, 0.99916812,
                  1.          ])),
          (0.21720327684112845, 0.33290203197356816, 0.9563111593699122))
```



10. Splitting data into train and test:

Now before creating model, we need to split the data into train and test data. Here train data has 75% of original train data and test data has 25% of original train data. Using following command:

```
#Divide data into train and test
```

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


11. Model Development:

Here as we need to predict the data, which has continuous type of values, we are using Regression. We are using 3 types of Regression to predict the values.

1. Multiple Linear Regression
2. Decision Tree Regression
3. Random Forest Regression

The error metrics we are using to evaluate our model is RMSE not MAPE because as our data contains Time-series type of variable called pickup date-time. The RMSE value should be as low as possible for a good model. In the same time, we are also trying to generate values for other metrics like MAE, MSE, RMSE, MAPE, r square, adjusted r square.

Using Multiple Linear Regression:

When we try to build a model using Multiple Linear Regression, here are the results:

Error Metrics	r square	Adjusted r square	MAPE	MSE	RMSE	RMSLE
Train data	0.7343	0.7339	18.7287	5.2833	2.2985	0.2165

Using Decision Tree for Regression:

When we try to build a model, using Decision Tree Regression. Here are the results:

Error Metrics	r square	Adjusted r square	MAPE	MSE	RMSE	RMSLE
Train data	0.6738	0.6732	22.6318	6.487	2.5470	0.2427

Using Random Forest Regression:

Error Metrics	r square	Adjusted r square	MAPE	MSE	RMSE	RMSLE
Train data	0.8952	0.8950	10.506	2.0841	1.4436	0.135

The RMSE for above models is less for Random Forest model. Therefore, we can predict the data using Random Forest which has less RMSE.

12. Conclusion:

The overall Project is quite challenging, as it takes so much time to sort out the variables and extract the date (year, month, hour, day) from pickup date-time and pickup, dropoff -longitudes and latitudes which helps in calculating distance.

The model we developed can be used for future purpose to predict the cab fare-amount. All we need is to enter the valid data belonging to respected variable.