Cab Fare Prediction

Project By,

Gaurav D Patil

Contents

Chapter 1

Introduction

1.1 Problem Statement

The objective of this project is to predict Cab Fare amount.
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

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.

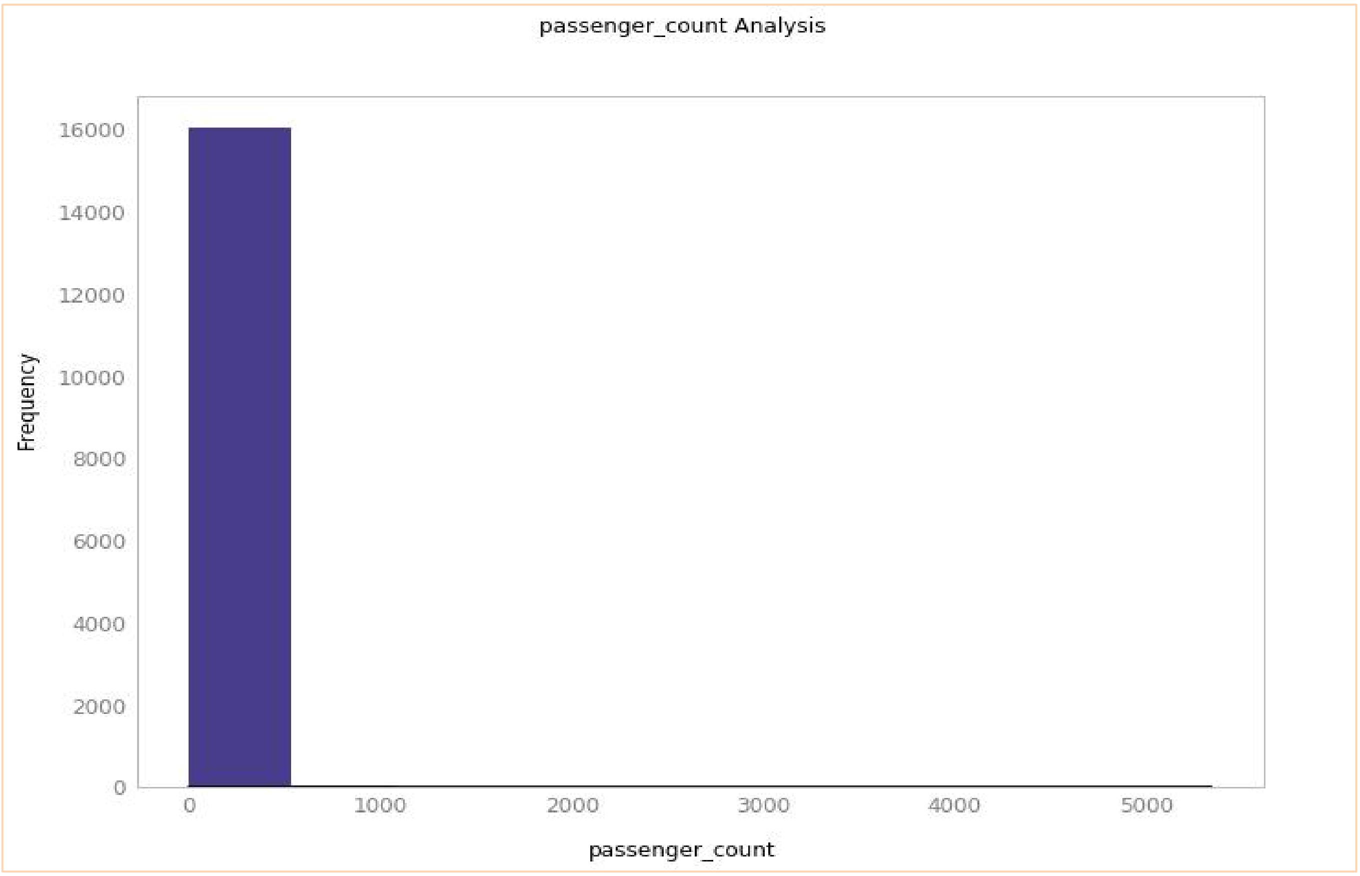
Chapter 2

Methodology

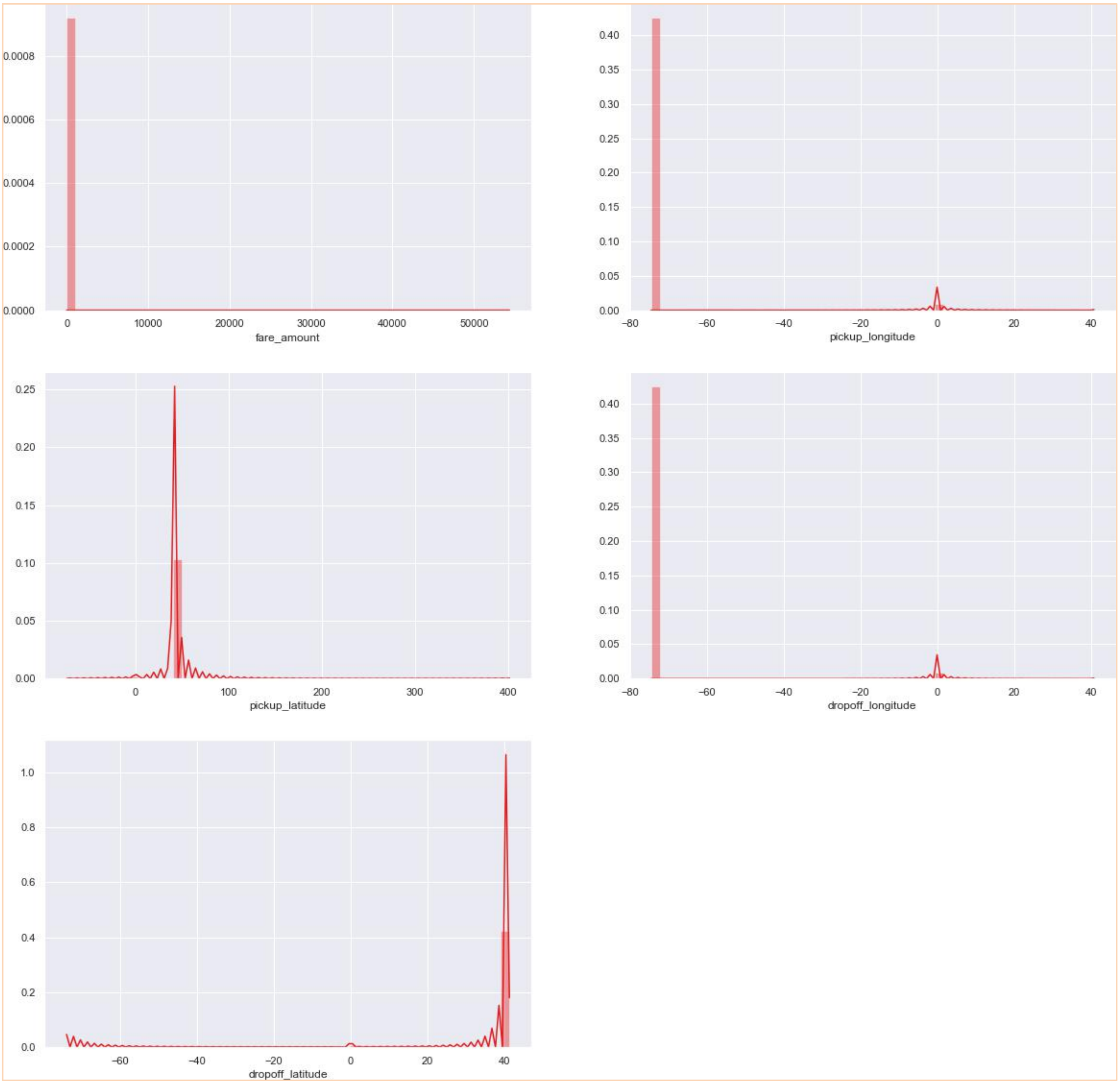
2.1 Pre-Processing

Data pre-processing is the first stage of any type of project. In this stage we get the feel of the data. We do this by looking at plots of independent variables vs target variables. If the data is messy, we try to improve it by sorting deleting extra rows and columns. This stage is called as Exploratory Data Analysis. This stage generally involves data cleaning, merging, sorting, looking for outlier analysis, looking for missing values in the data, imputing missing values if found by various methods such as mean, median, mode, KNN imputation, etc.
Further we will look into what Pre-Processing steps do this project was involved in.

Getting feel of data via visualization:

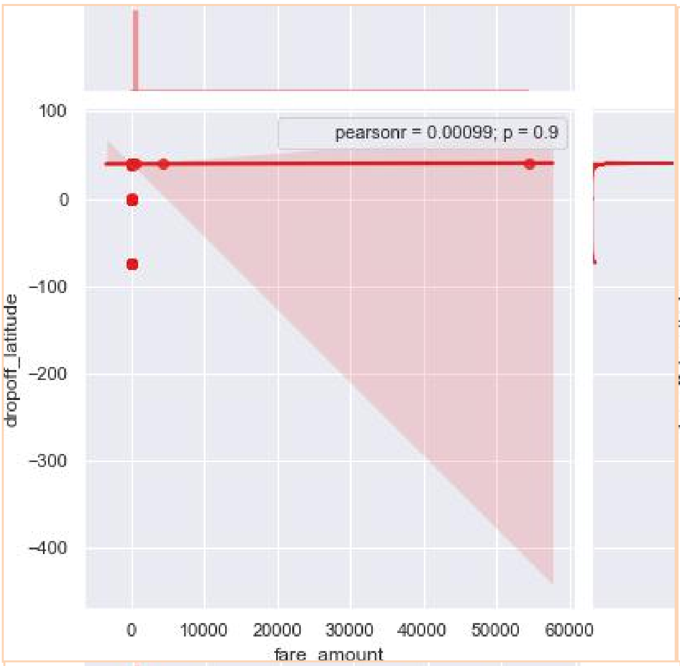


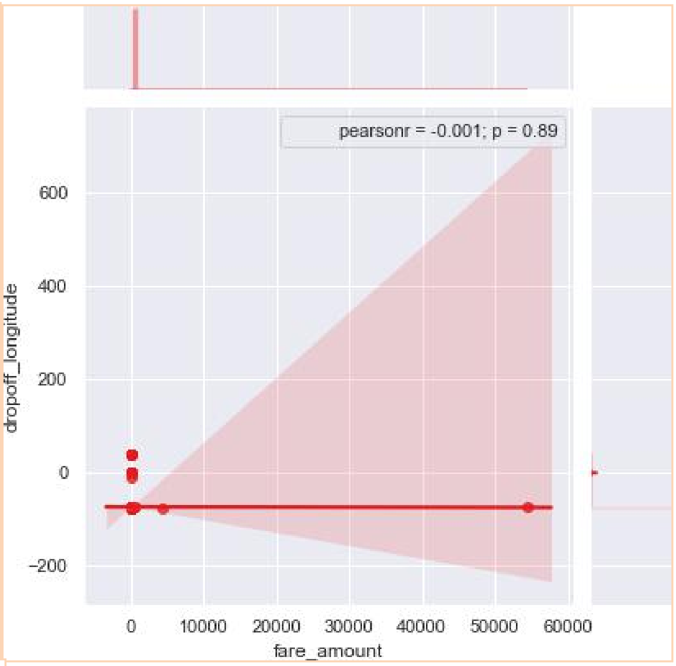
Some Histogram plots from seaborn library for each individual variable created using distplot() method.

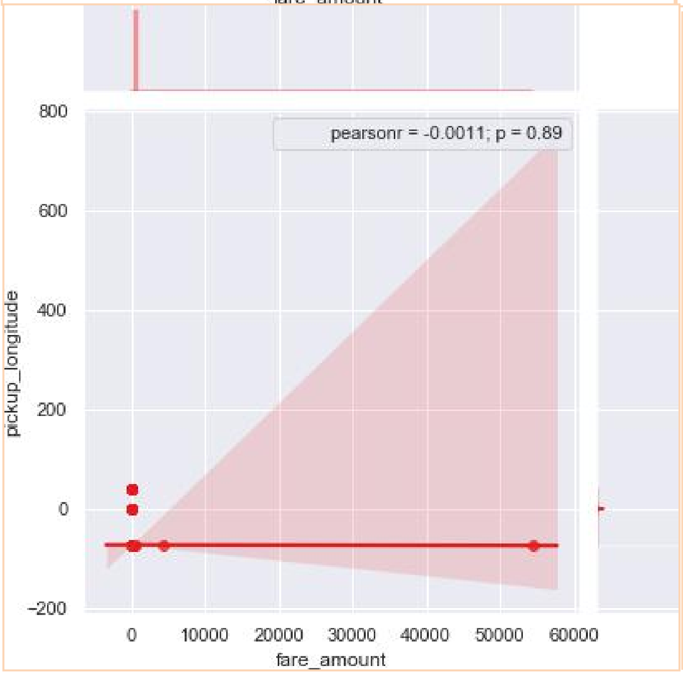


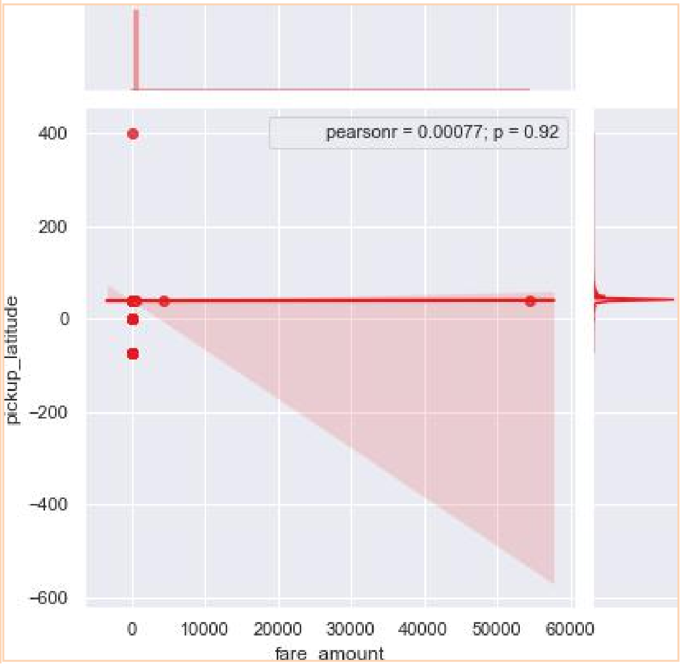
Some Jointplots:

- They are used for Bivariate Analysis.
- Here we have plotted Scatter plot with Regression line between 2 variables along with separate Bar plots of both variables.
- Also, we have annotated Pearson correlation coefficient and p value.
- Plotted only for numerical/continuous variables
- Target variable 'fare_amount' Vs each numerical variable.

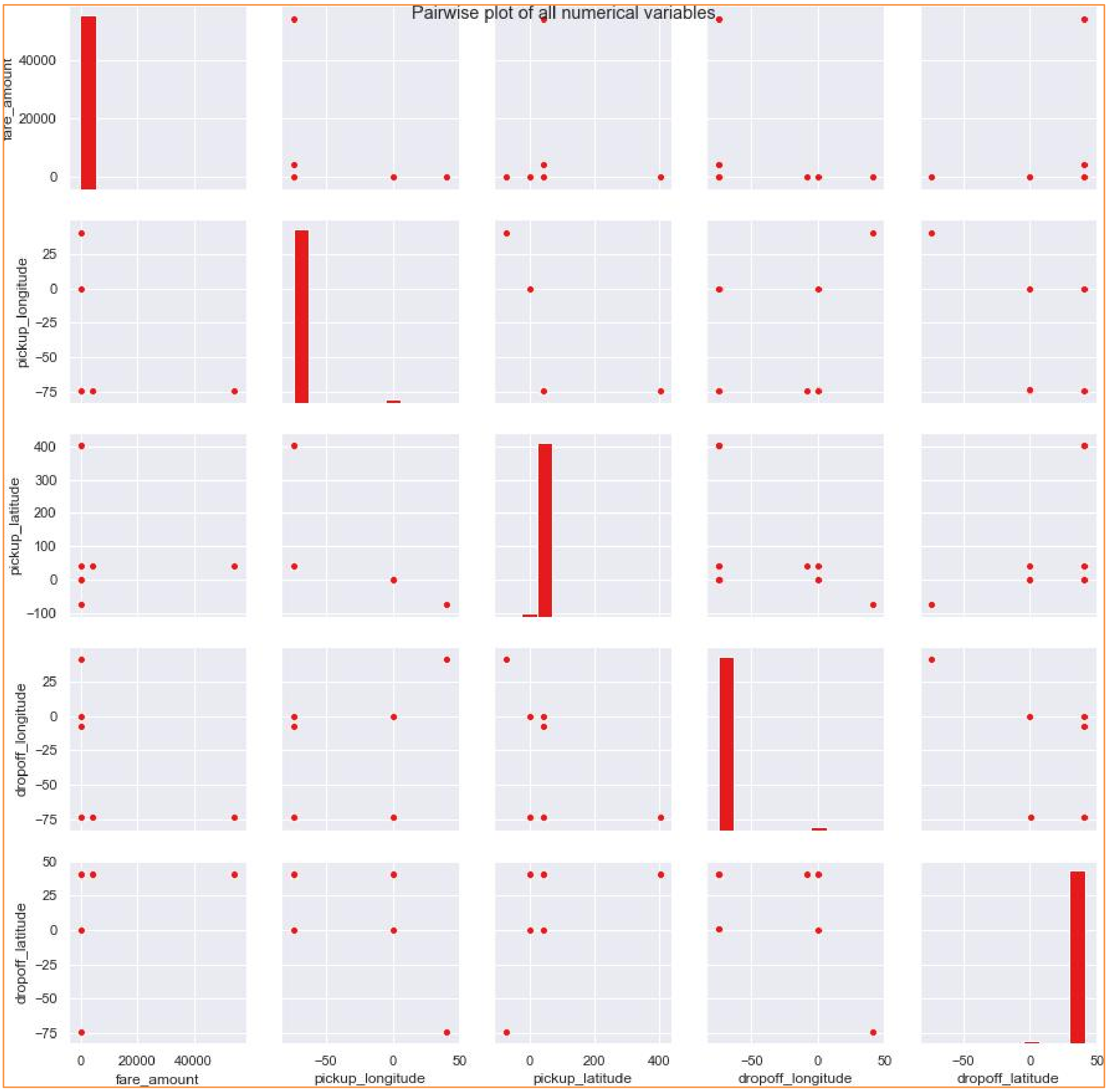








Pairwise Plots for all Numerical variables:



2.1.1 Removing values which are not within desired range(outlier) depending upon basic understanding of dataset.

this step we will remove values in each variable which are not within desired range and we will consider them as outliers depending upon basic understanding of all the variables. Youwouldthinkwhyhaven'tmadethosevaluesNAinsteadofremovingthemwellIdidmade them NA but it turned out to be a lot of missing values(NA's) in the dataset. Missing values percentage becomes very much high and then there will be no point of using that imputed data. Take a look at below 3 scenarios--

- If everything beyond range is made nan also except latitudes and longitudes then:

Variables	Missing_percentage	
0	passenger_count	29.563702
1	pickup_latitude	1.966764
2	pickup_longitude	1.960540
3	dropoff_longitude	1.954316
4	dropoff_latitude	1.941868
5	fare_amount	0.186718
6	pickup_datetime	0.006224

After imputing above mentioned missing values kNN algorithm imputes every value to 0 at a particular row which was made nan using np.nan method:

fare_amount	0.0
pickup_longitude	0.0
pickup_latitude	0.0
dropoff_longitude	0.0
dropoff_latitude	0.0
passenger_count	0.0

Name: 1000, dtype: float64

- And If everything is dropped which are beyond range then below are the missing percentages for each variable:

Variables	Missing_percentage	
0	passenger_count	0.351191
1	fare_amount	0.140476
2	pickup_datetime	0.006385
3	pickup_longitude	0.000000
4	pickup_latitude	0.000000
5	dropoff_longitude	0.000000
6	dropoff_latitude	0.000000

After imputing above mentioned missing values kNN algorithm values at a particular row which was made nan using np.nan method

```
fare_amount          7.3698
pickup_longitude    -73.9954
pickup_latitude     40.7597
dropoff_longitude  -73.9876
dropoff_latitude     40.7512
passenger_count           2
Name: 1000, dtype: object
```

- If everything beyond range is made nan except passenger_count:

Variables	Missing_percentag e	
0	pickup_latitude	1.951342
1	dropoff_longitude	1.951342
2	pickup_longitude	1.945087
3	dropoff_latitude	1.938833
4	passenger_count	0.343986
5	fare_amount	0.181375
6	pickup_datetime	0.006254

After imputing above mentioned missing values kNN algorithm imputes every value to 0 at a particular row which was made nan using np.nan method:

```
fare_amount         0.0
pickup_longitude    0.0
pickup_latitude     0.0
dropoff_longitude   0.0
dropoff_latitude    0.0
passenger_count     0.0
Name: 1000, dtype: float64
```

2.1.2 Missing value Analysis

In this step we look for missing values in the dataset like empty row column cell which was left after removing special characters and punctuation marks. Some missing values are in form of NA. missing values left behind after outlier analysis; missing values can be in any form. Unfortunately, in this dataset we have found some missing values. Therefore, we will do some missing value analysis. Before imputed we selected random row no-1000 and made it NA, so that we will compare original value with imputed value and choose best method which will impute value closer to actual value.

	index	0
0	fare_amount	22
1	pickup_datetime	1
2	pickup_longitude	0
3	pickup_latitude	0
4	dropoff_longitude	0
5	dropoff_latitude	0
6	passenger_count	55

We will impute values for fare_amount and passenger_count both of them has missing values 22 and 55 respectively. We will drop 1 value in pickup_datetime i.e it will be an entire row to drop.

Below are the missing value percentage for each variable:

Variables	Missing_percentage	
0	passenger_count	0.351191
1	fare_amount	0.140476
2	pickup_datetime	0.006385
3	pickup_longitude	0.000000
4	pickup_latitude	0.000000
5	dropoff_longitude	0.000000
6	dropoff_latitude	0.000000

And below is the Standard deviation of particular variable which has missing values in them:
fare_amount 435. 982171

```
passenger_count       1.266096
dtype: float64
```

We'dtriedcentralstatisticalmethodsandalgorithmicmethod--KNN to impute missing values in the dataset:

1. **For**

   Passenger_count:

   Actual value = 1

   Mode = 1

   KNN = 2

We will choose the KNN method here because it maintains the standard deviation of variable. We will not use Mode method because whole variable will be more biased towards 1 passenger_count also passenger_count has maximum value equals to 1

2. **For fare_amount**:

   Actual value = 7.0,

   Mean = 15.117,

   Median = 8.5,

   KNN = 7.369801

We will Choose KNN method here because it imputes value closest to actual value also it maintains the Standard deiviation of the variable.

Standard deviation for passenger_count and fare_amount after KNN imputation:

```
fare_amount         435.661995
passenger_count       1.264322
dtype: float64
```

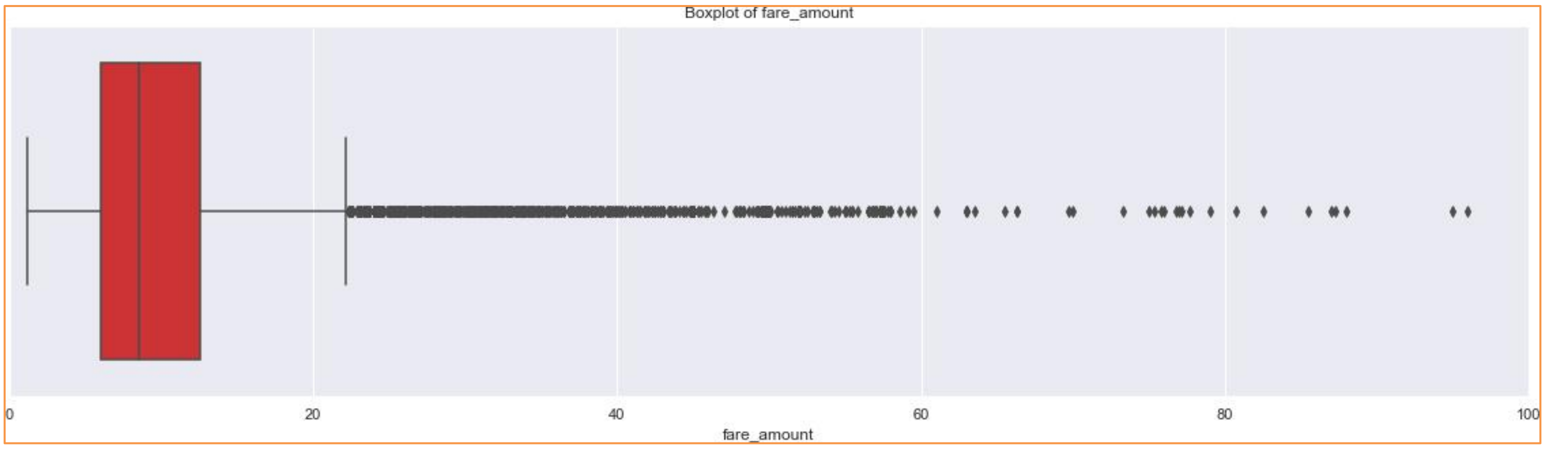
2.1.3 Outlier Analysis

We look for outlier in the dataset by plotting Boxplots. There are outliers present in the data. we have removed these outliers. This is how we done,

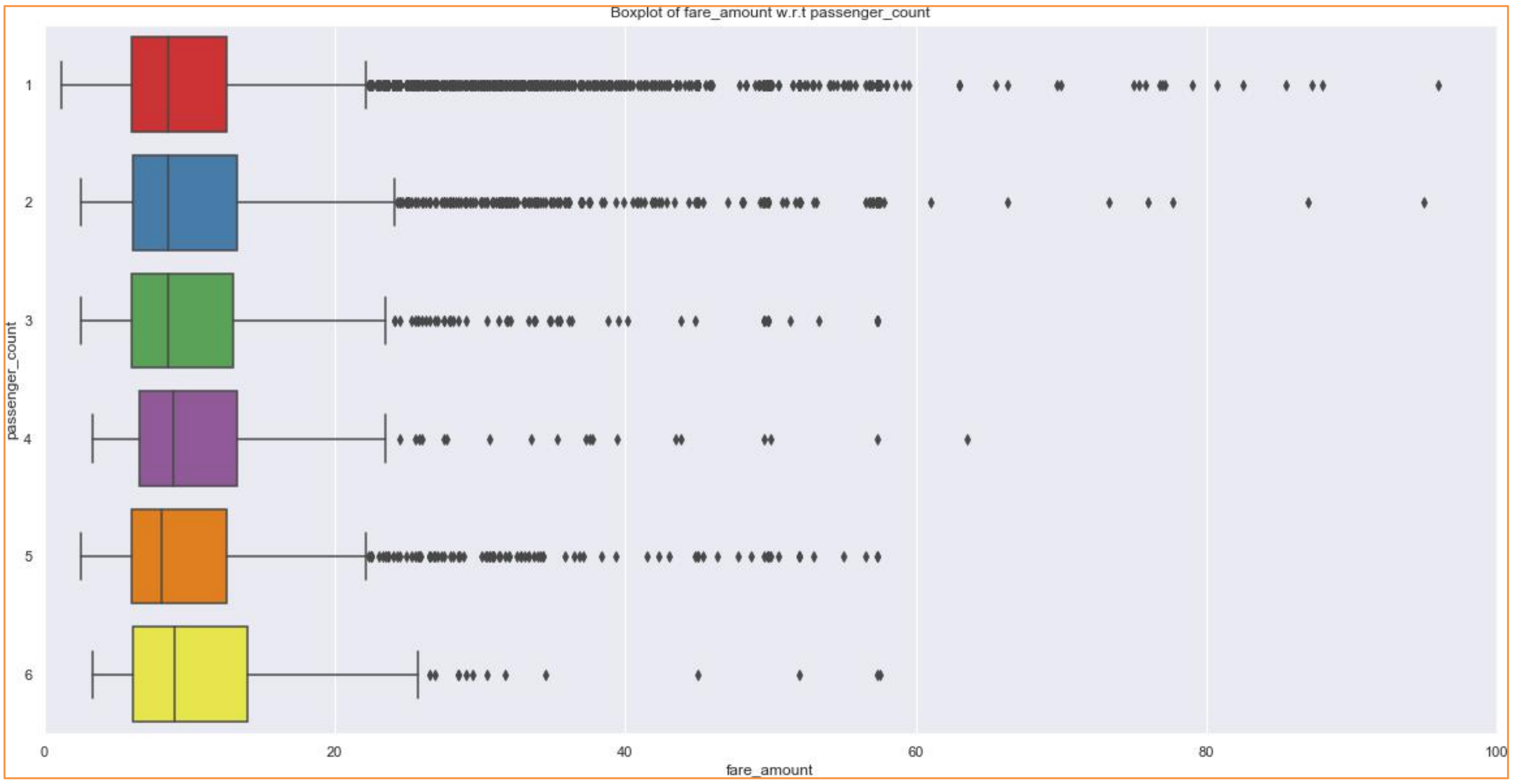
I. We replaced them with Nan values or we can say created missing values.
II. Then we imputed those missing values with KNN method.

- We Will do Outlier Analysis only on Fare_amount just for now and we will do outlier analysis after feature engineering laitudes and longitudes.
- Univariate Boxplots: Boxplots for target variable.

Univariate Boxplots: Boxplots for all Numerical Variables also for target variable



Bivariate Boxplots: Boxplots for all fare_amount Variables Vs all passenger_count variable.



From above Boxplots we see that ‘fare_amount’have outliers in it:

‘fare_amount’ has 1359 outliers.

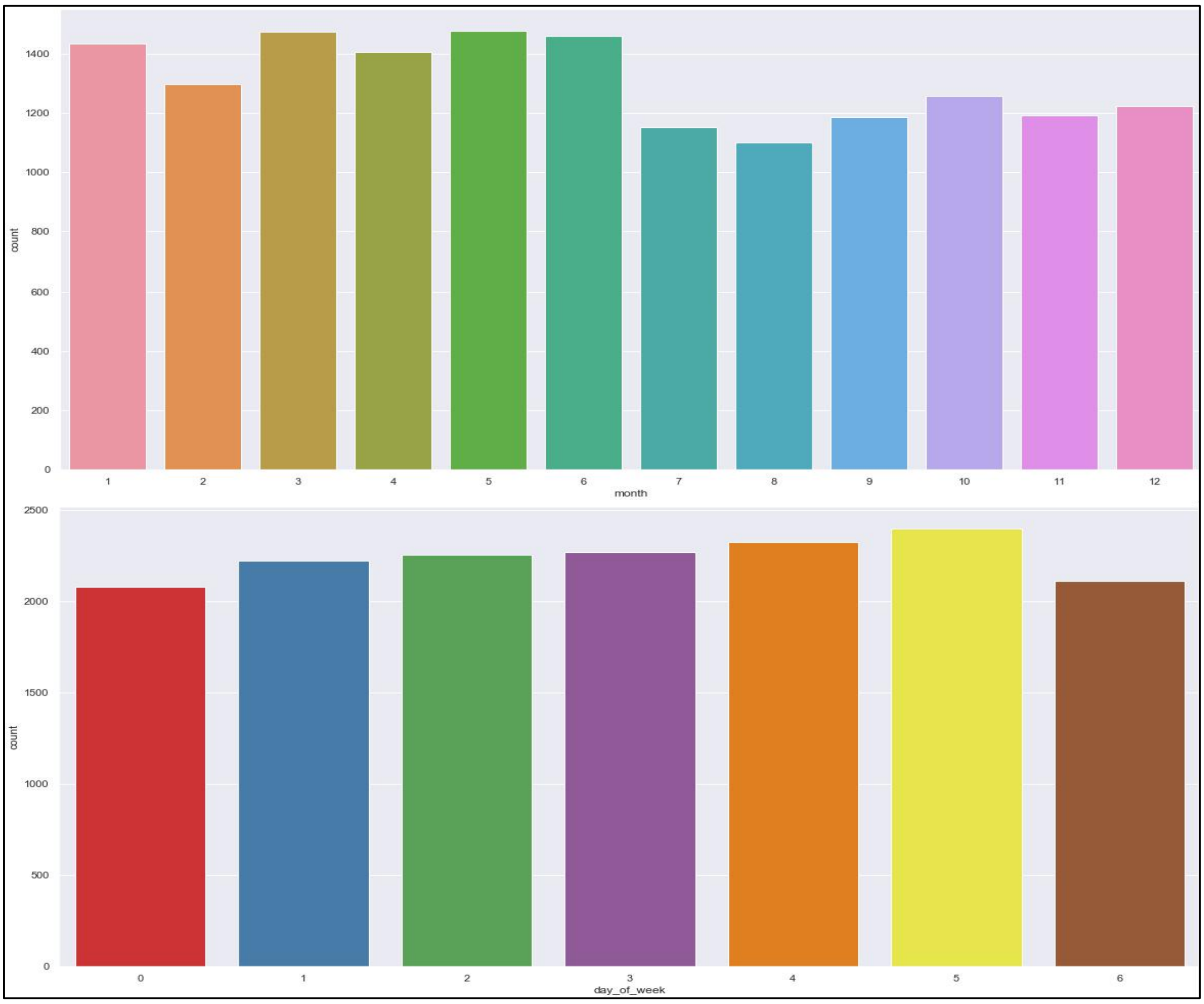
We successfully imputed these outliers with KNN and K value is 3

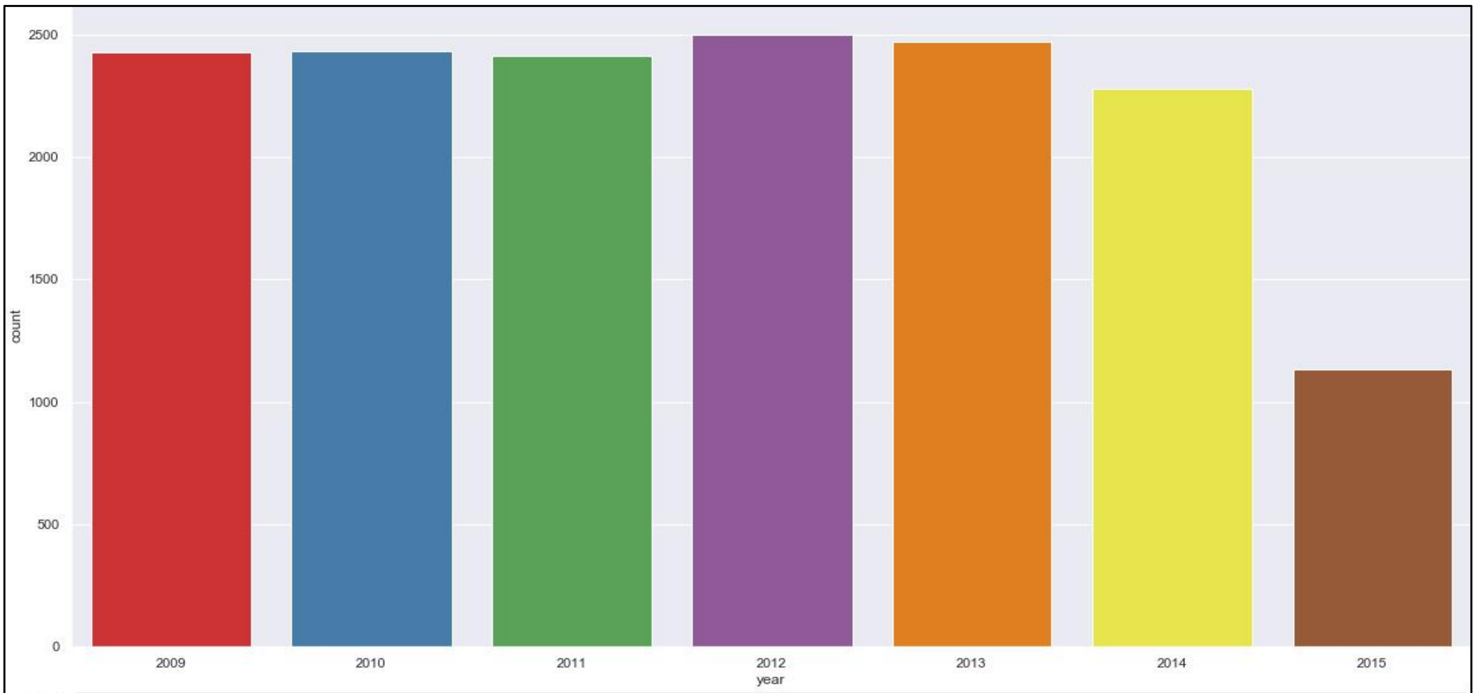
2.1.4 Feature Engineering

Feature Engineering is used to drive new features from existing features.

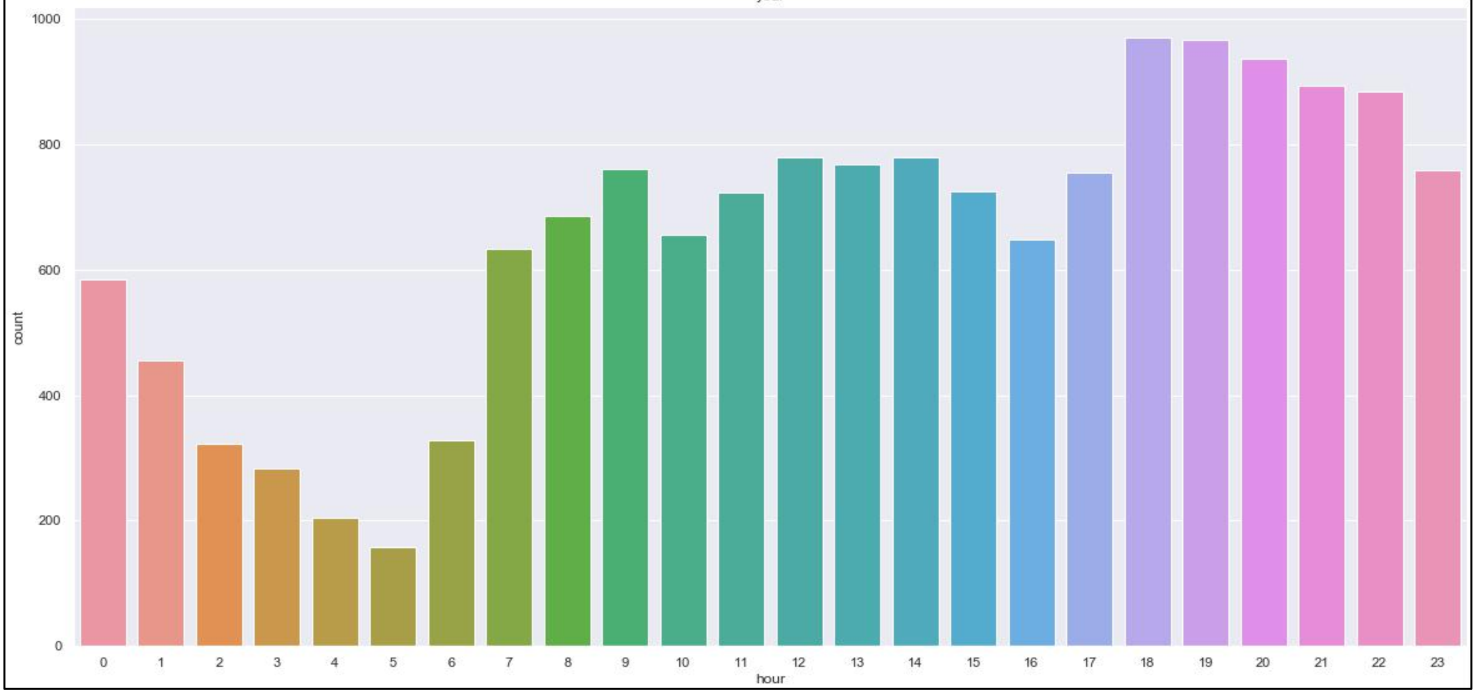
1. <u>For 'pickup_datetime' variable</u>:

We will use this timestamp variable to create new variables.
New features will be year, month, day_of_week, hour.
'year' will contain only years from pickup_datetime. For ex. 2009, 2010, 2011, etc.
'month' will contain only months from pickup_datetime. For ex. 1 for January, 2 for February, etc.
'day_of_week' will contain only week from pickup_datetime. For ex. 1 which is for Monday,2 for Tuesday,etc.
'hour' will contain only hours from pickup_datetime. For ex. 1, 2, 3, etc.





2500
2000
1500
1000
500
0
count
2009
2010
2011
2012
2013
2014
2015
year



1000
800
600
400
200
0
count
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
hour

As we have now these new variables we will categorize them to new variables like Session from hour column, seasons from month column, week:weekday/weekend from day_of_week variable.

So, session variable which will contain categories—morning, afternoon, evening, night_PM, night_AM.

Seasons variable will contain categories—spring, summer, fall, winter. Week will contain categories—weekday, weekend.

We will one-hot-encode session, seasons, week variable.

2. **<u>For 'passenger_count' variable</u>:**

As passenger_count is a categorical variable we will one-hot-encode it.

3. **<u>For 'Latitudes' and 'Longitudes' variables:</u>**

As we have latitude and longitude data for pickup and dropoff, we will find the distance the cab travelled from pickup and dropoff location.

We will use both haversine and vincenty methods to calculate distance. For haversine, variable name will be 'great_circle' and for vincenty, new variable name will be 'geodesic'.

As Vincenty is more accurate than haversine. Also, vincenty is prefered for short distances. Therefore, we will drop great_circle.

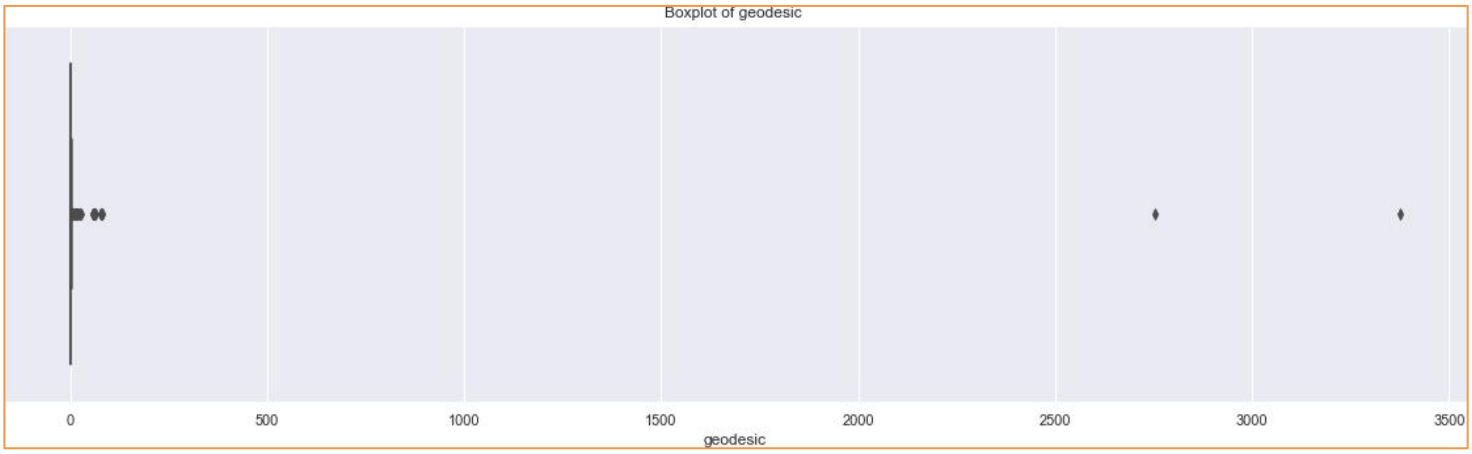
Columns in training data after feature engineering:

```
Index(['fare_amount', 'passenger_count_2',
'passenger_count_3',
       'passenger_count_4', 'passenger_count_5',
       'passenger_count_6', 'season_spring', 'season_summer',
       'season_winter', 'week_weekend', 'session_evening',
       'session_morning', 'session_night_AM', 'session_night_PM',
       'year_2010', 'year_2011', 'year_2012', 'year_2013',
       'year_2014', 'year_2015', 'geodesic'],
      dtype='object')
```

Columns in testing data after feature engineering:

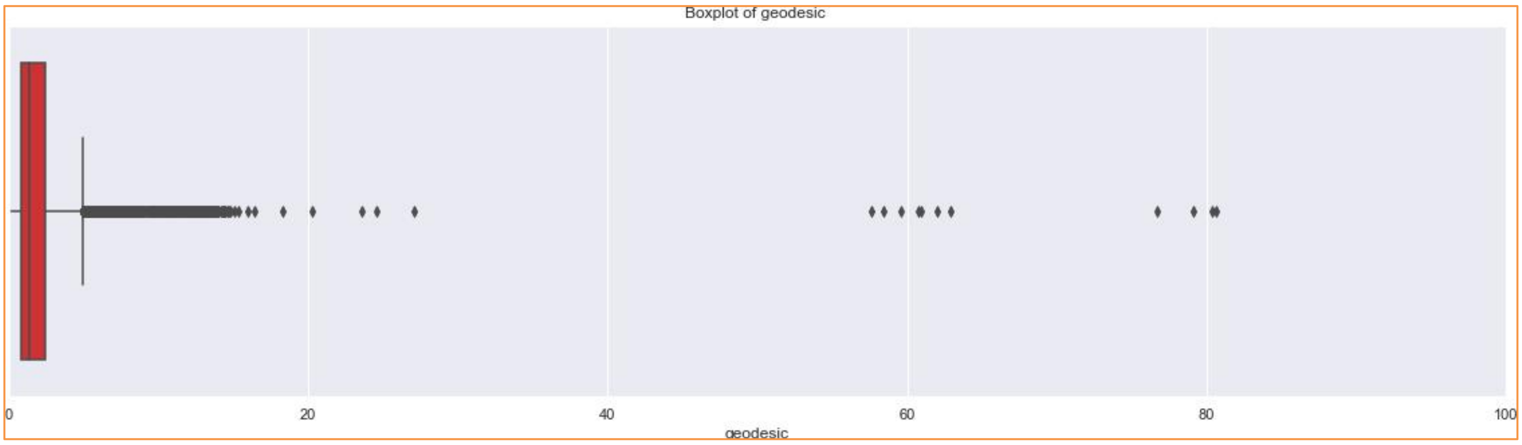
```
Index(['passenger_count_2', 'passenger_count_3',
'passenger_count_4',
       'passenger_count_5', 'passenger_count_6', 'season_spring',
       'season_summer', 'season_winter', 'week_weekend',
       'session_evening', 'session_morning', 'session_night_AM',
       'session_night_PM', 'year_2010', 'year_2011', 'year_2012',
       'year_2013', 'year_2014', 'year_2015', 'geodesic'],
      dtype='object')
```

we will plot boxplot for our new variable ‘geodesic’:



We see that there are outliers in ‘geodesic’ and also a cab cannot go upto 3400 miles.

Boxplot of ‘geodesic’ for range 0 to 100 miles.



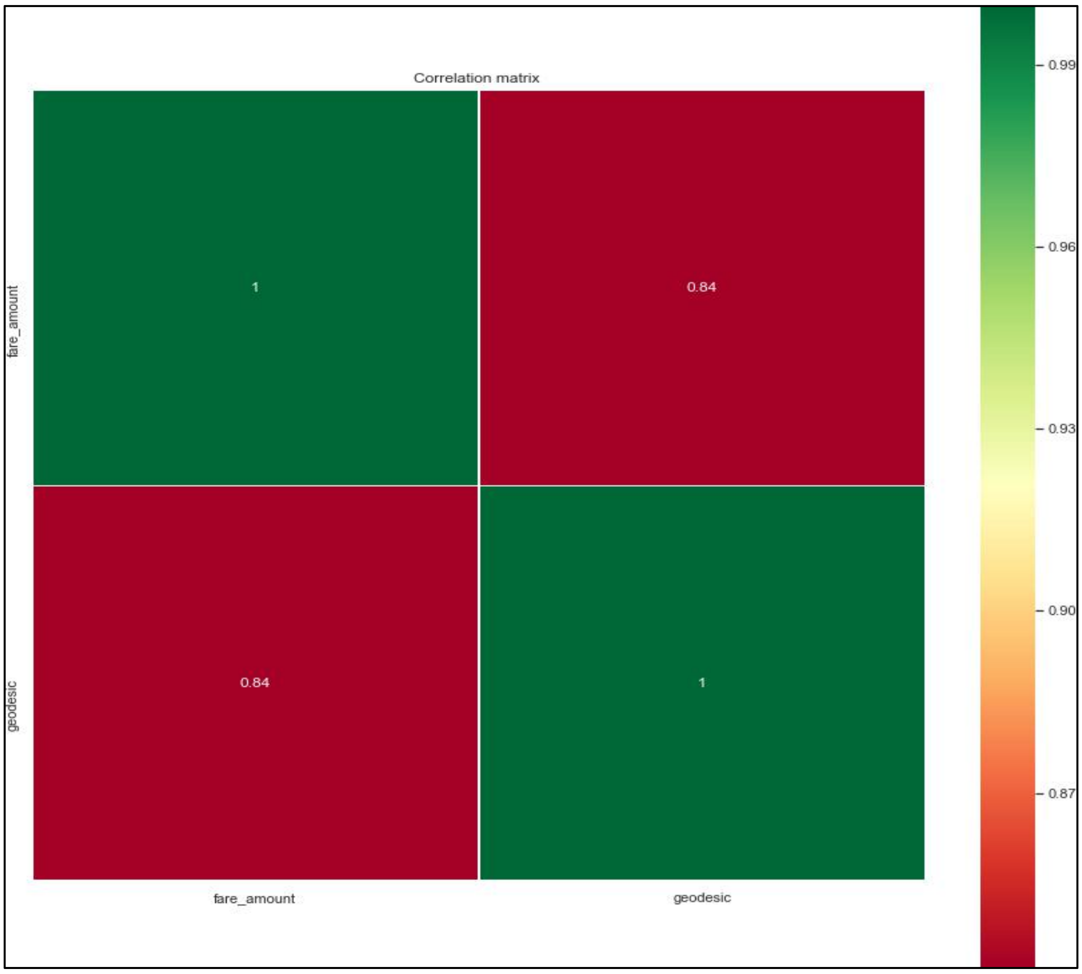
We will treat these outliers like we previously did.

2.1.5 Feature Selection

In this step we would allow only to pass relevant features to further steps. We remove irrelevant features from the dataset. We do this by some statistical techniques, like we look for features which will not be helpful in predicting the target variables. In this dataset we have to predict the fare_amount.
Further below are some types of test involved for feature selection:

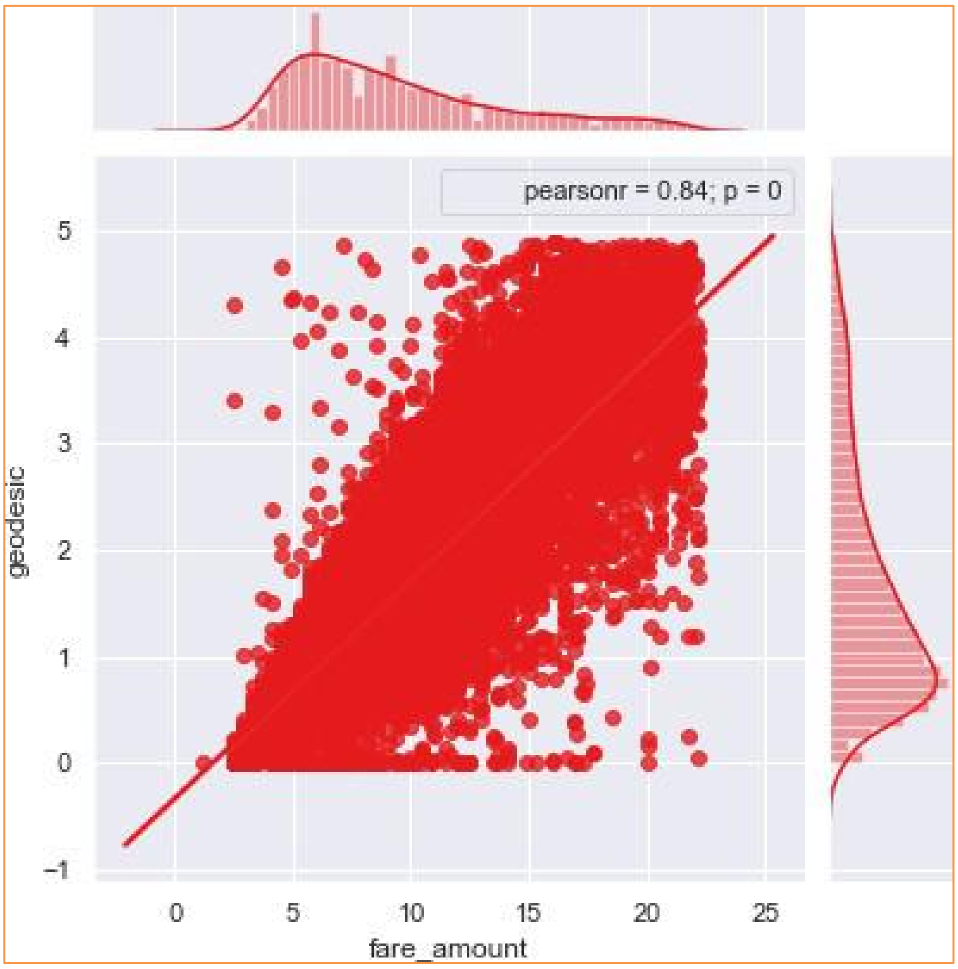
1. **Correlation analysis** – This requires only numerical variables. Therefore, we will filter out only numerical variables and feed it to correlation analysis. We do this by plotting correlation plot for all numerical variables. There should be no correlation between independent variables but there should be high correlation between independent variable and dependent variable. So, we plot the correlation plot. we can see that in correlation plot faded colour like skin colour indicates that 2 variables are highly correlated with each other. As the colour fades correlation values increases. From below correlation plot we see that:
   - 'fare_amount' and 'geodesic' are very highly correlated with each other.
   - As fare_amount is the target variable and 'geodesic' is independent variable we will

     Correlation Plot:



keep 'geodesic' because it will help to explain variation in fare_amount.

Jointplot between ‘geodesic’ and ‘fare_amount’:



2 Chi-Square test of independence –

Unlike correlation analysis we will filter out only categorical variables and pass it to Chi-Square test. Chi-square test compares 2 categorical variables in a contingency table to see if they are related or not.

I. Assumption for chi-square test: Dependency between Independent variable and dependent variable should be high and there should be no dependency among independent variables.

II. Before proceeding to calculate chi-square statistic, we do the hypothesis testing:
Null hypothesis: 2 variables are independent.
Alternate hypothesis: 2 variables are not independent.
The interpretation of chi-square test:

I. For theorical or excel sheet purpose: If chi-square statistics is greater than critical value then reject the null hypothesis saying that 2 variables are dependent and if it’s less, then accept the null hypothesis saying that 2 variables are independent.

II. While programming: If p-value is less than 0.05 then we reject the null hypothesis saying that 2 variables are dependent and if p-value is greater than 0.05 then we accept the null hypothesis saying that 2 variables are independent.

Here we did the test between categorical independent variables pairwise.

- If p-value<0.05 then remove the variable,
- If p-value>0.05 then keep the variable.

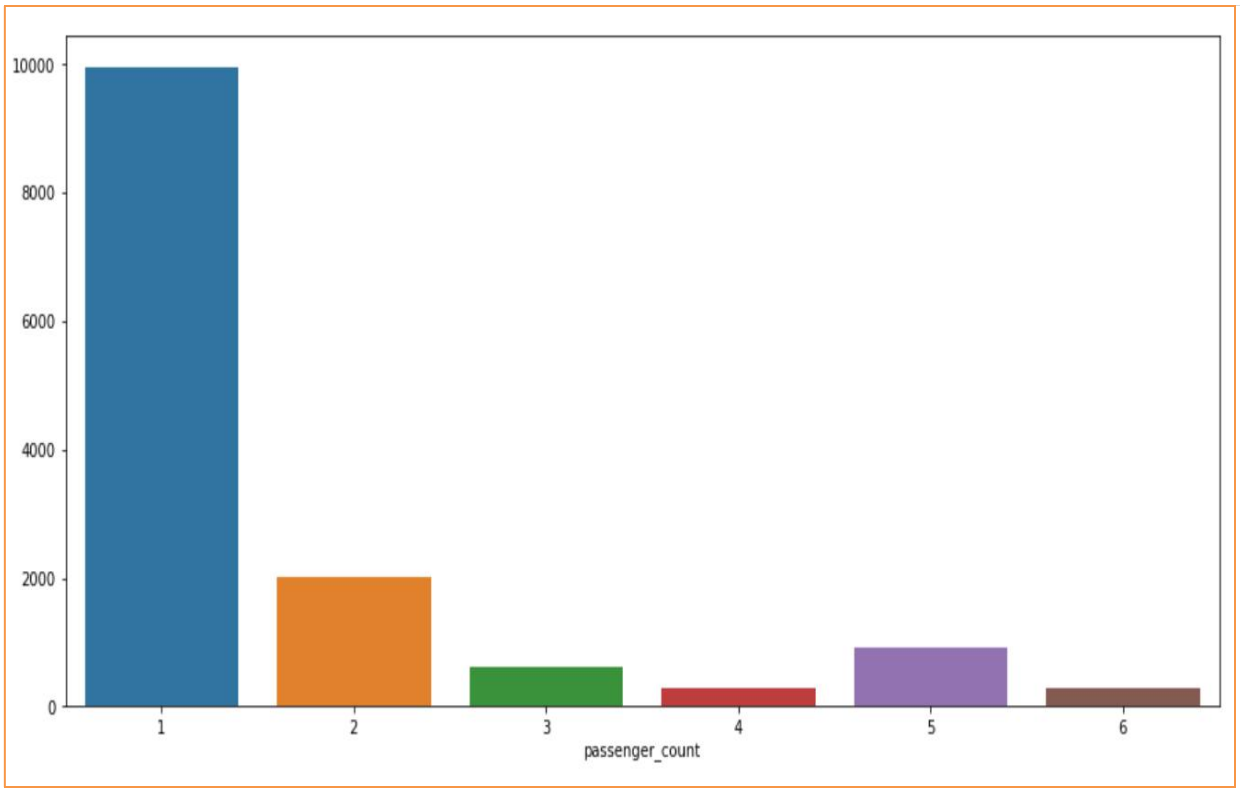
Multicollinearity–

In regression, "multicollinearity" refers to predictors that are correlated with other predictors. Multicollinearity occurs when your model includes multiple factors that are correlated not just to your response variable, but also to each other.\

I. Multicollinearity increases the standard errors of the coefficients.
II. Increased standard errors in turn means that coefficients for some independent variables may be found not to be significantly different from 0.
III. In other words, by overinflating the standard errors, multicollinearity makes some variables statistically insignificant when they should be significant. Without multicollinearity (and thus, with lower standard errors), those coefficients might be significant.
IV. VIF is always greater or equal to 1.
if VIF is 1 --- Not correlated to any of the variables.
if VIF is between 1-5 --- Moderately correlated.
if VIF is above 5 --- Highly correlated.
If there are multiple variables with VIF greater than 5, only remove the variable with the highest VIF.
V. And if the VIF goes above 10, you can assume that the regression coefficients are poorly estimated due to multicollinearity.

Below is the table for VIF analysis for each independent variable:

	VIF	features
0	15.268789	Intercept
1	1.040670	passenger_count_2[T.1.0]
2	1.019507	passenger_count_3[T.1.0]
3	1.011836	passenger_count_4[T.1.0]
4	1.024990	passenger_count_5[T.1.0]
5	1.017206	passenger_count_6[T.1.0]
6	1.642247	season_spring[T.1.0]
7	1.552411	season_summer[T.1.0]
8	1.587588	season_winter[T.1.0]
9	1.050786	week_weekend[T.1.0]
10	1.376197	session_night_AM[T.1.0]
11	1.423255	session_night_PM[T.1.0]
12	1.524790	session_evening[T.1.0]
13	1.559080	session_morning[T.1.0]
14	1.691361	year_2010[T.1.0]
15	1.687794	year_2011[T.1.0]
16	1.711100	year_2012[T.1.0]
17	1.709348	year_2013[T.1.0]
18	1.665000	year_2014[T.1.0]
19	1.406916	year_2015[T.1.0]
20	1.025425	geodesic



We have checked for multicollinearity in our Dataset and all VIF values are below 5.

2.1.6 Feature Scaling

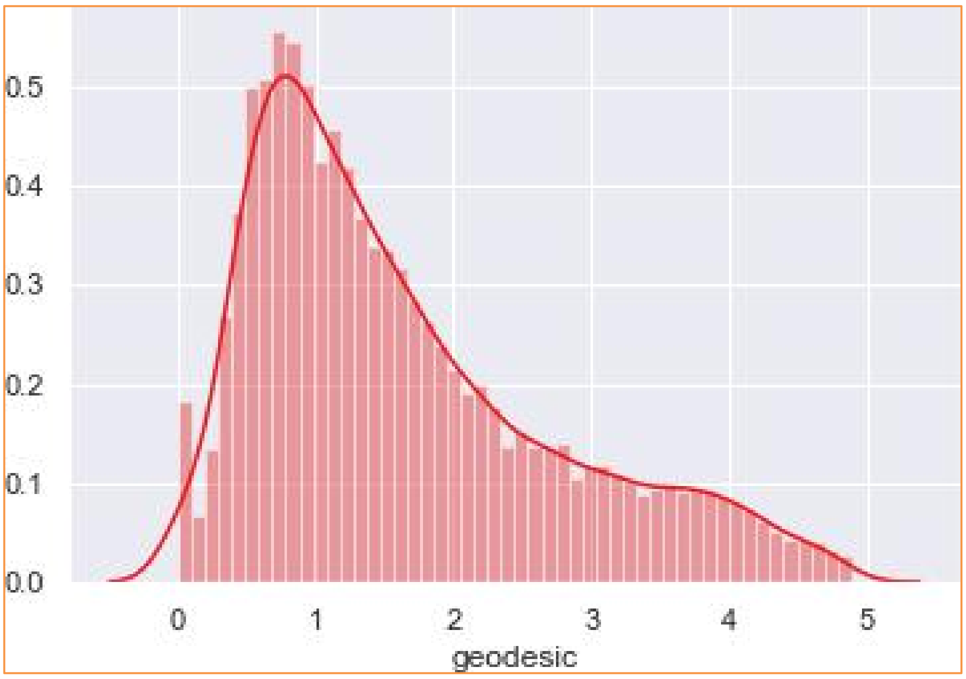
Data Scaling methods are used when we want our variables in data to scaled on common ground. It is performed only on continuous variables.

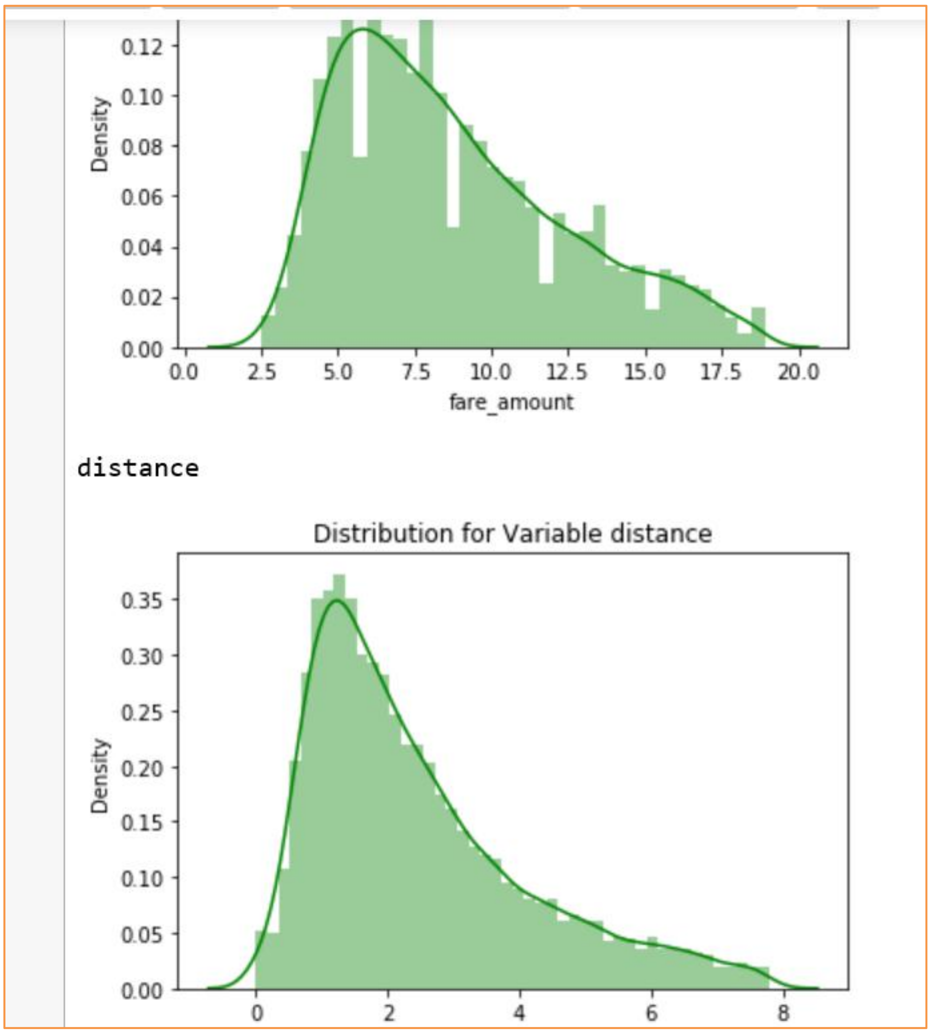
- **Normalization**: Normalization refer to the dividing of a vector by its length. normalization normalizes the data in the range of 0 to 1. It is generally used when we are planning to use distance method for our model development purpose such as KNN. Normalizing the data improves convergence of such algorithms. Normalisation of data scales the data to a very small interval, where outliers can be loosed.

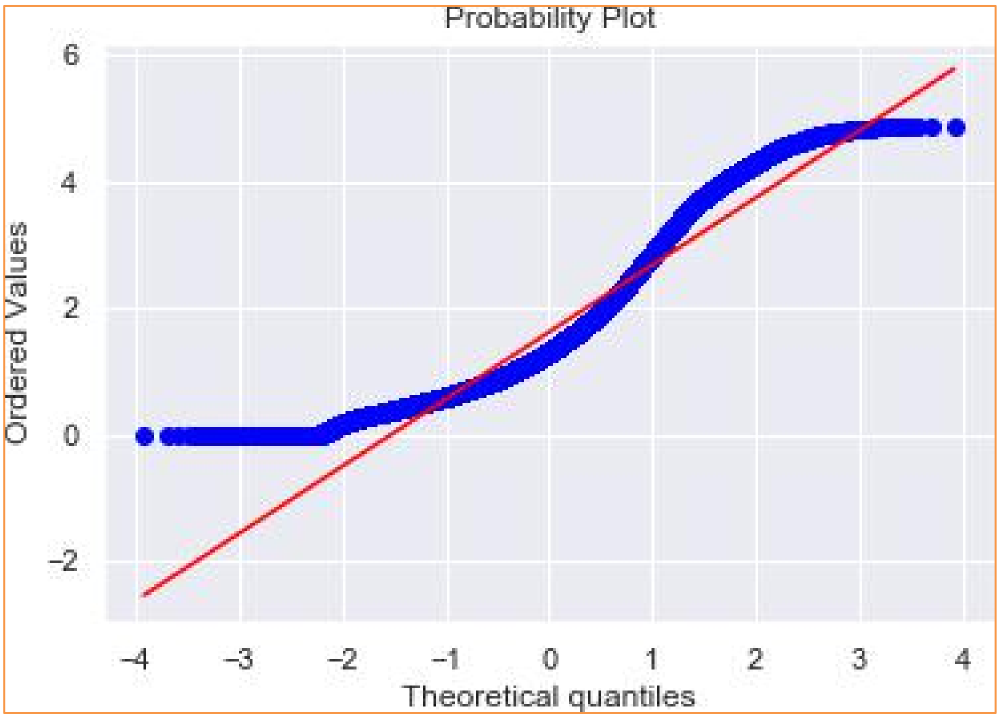
- **Standardization**: Standardization refers to the subtraction of mean from individual point and then dividing by its SD. Z is negative when the raw score is below the mean and Z is positive when above mean. When the data is distributed normally you should go for standardization.

Linear Models assume that the data you are feeding are related in a linear fashion, or can be measured with a linear distance metric.
Also, our independent numerical variable 'geodesic' is not distributed normally so we had chosen normalization over standardization.
- We have checked variance for each column in dataset before Normalisation
- High variance will affect the accuracy of the model. So, we want to normalise that variance.

Graphs based on which standardization was chosen:
Note: It is performed only on Continuous variables.
distplot() for 'geodesic' feature before normalization:

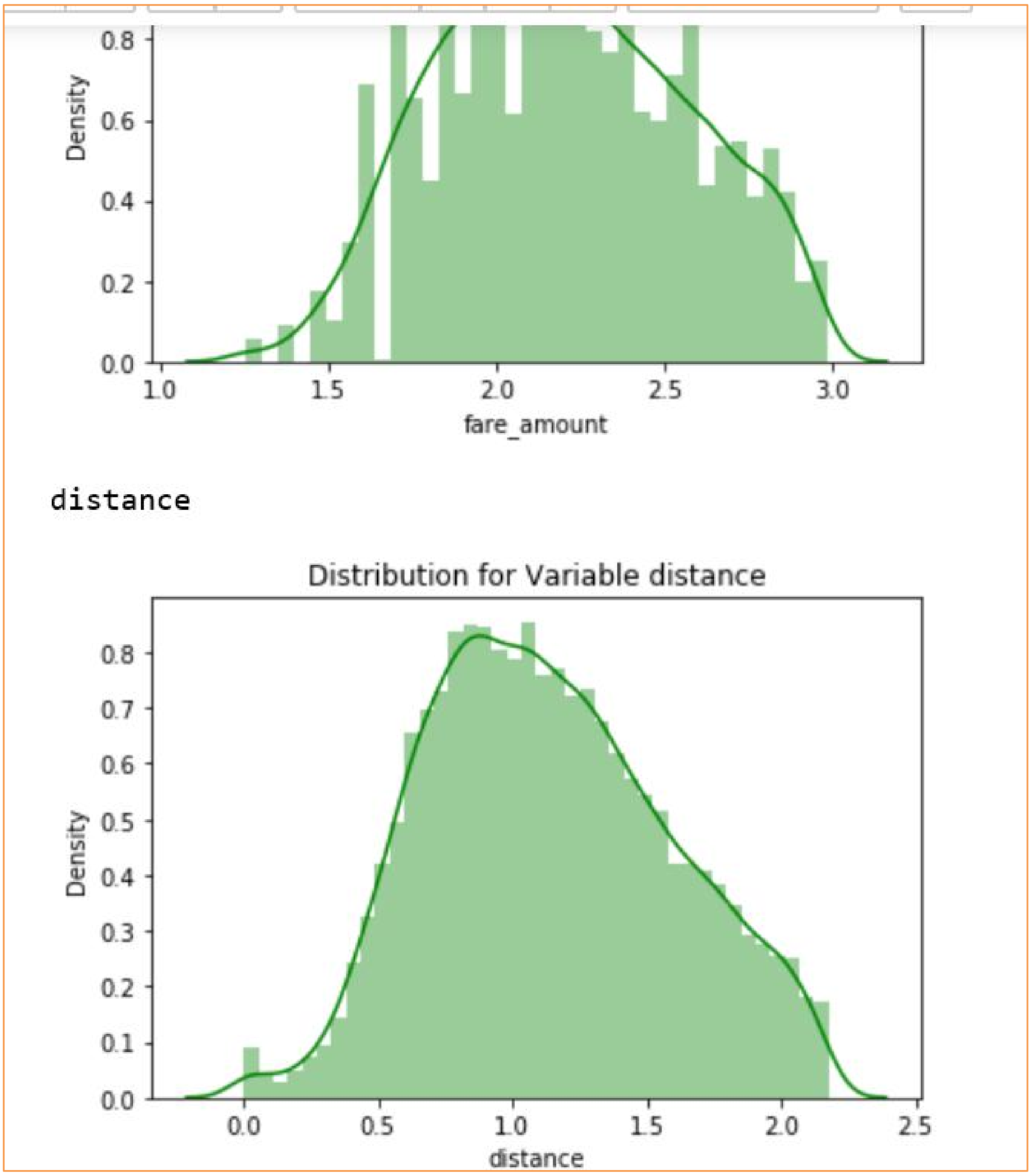

0.5
0.4
0.3
0.2
0.1
0.0
0
1
2
3
4
5
geodesic



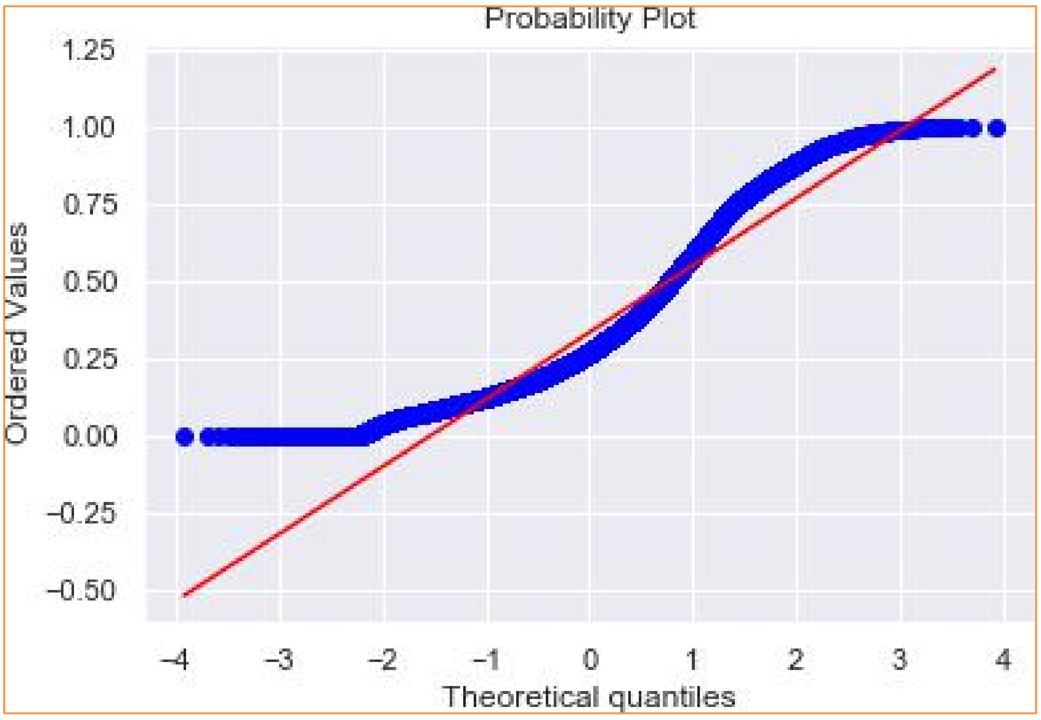


qq probability plot before normalization:

distplot() for 'geodesic' feature after normalization:


distance

qq probability plot after normalization:



Chapter 3

Splitting train and Validation Dataset

a) Wehaveusedsklearn'strain_test_split()methodtodividewholeDatasetintotrainandvalidation datset.
b) 25% is in validation dataset and 75% is in training data.
c) 11745 observations in training and 3915 observations in validation dataset.
d) We will test the performance of model on validation datset.
e) The model which performs best will be chosen to perform on test dataset provided along with original train dataset.
f) X_train y_train--are train subset.
g) X_test y_test--are validation subset.

Chapter 4

Model Development

Our problem statement wants us to predict the fare_amount. This is a Regression problem. So, we are going to build regression models on training data and predict it on test data. In this project I have built models using 5 Regression Algorithms:

I. Linear Regression
II. Decision Tree Regression
III. Random Forest Regression
IV. Gradient Boosting
V. Xgboost Regression

We will evaluate performance on validation dataset which was generated using Sampling. We will deal with specific error metrics like –
Regression metrics for our Models:

- r square
- Adjusted r square
- MAPE(Mean Absolute Percentage Error)
- MSE(Mean square Error)
- RMSE(Root Mean Square Error)
  RMSLE(Root Mean Squared Log E

Chapter 5

ML Model Selection

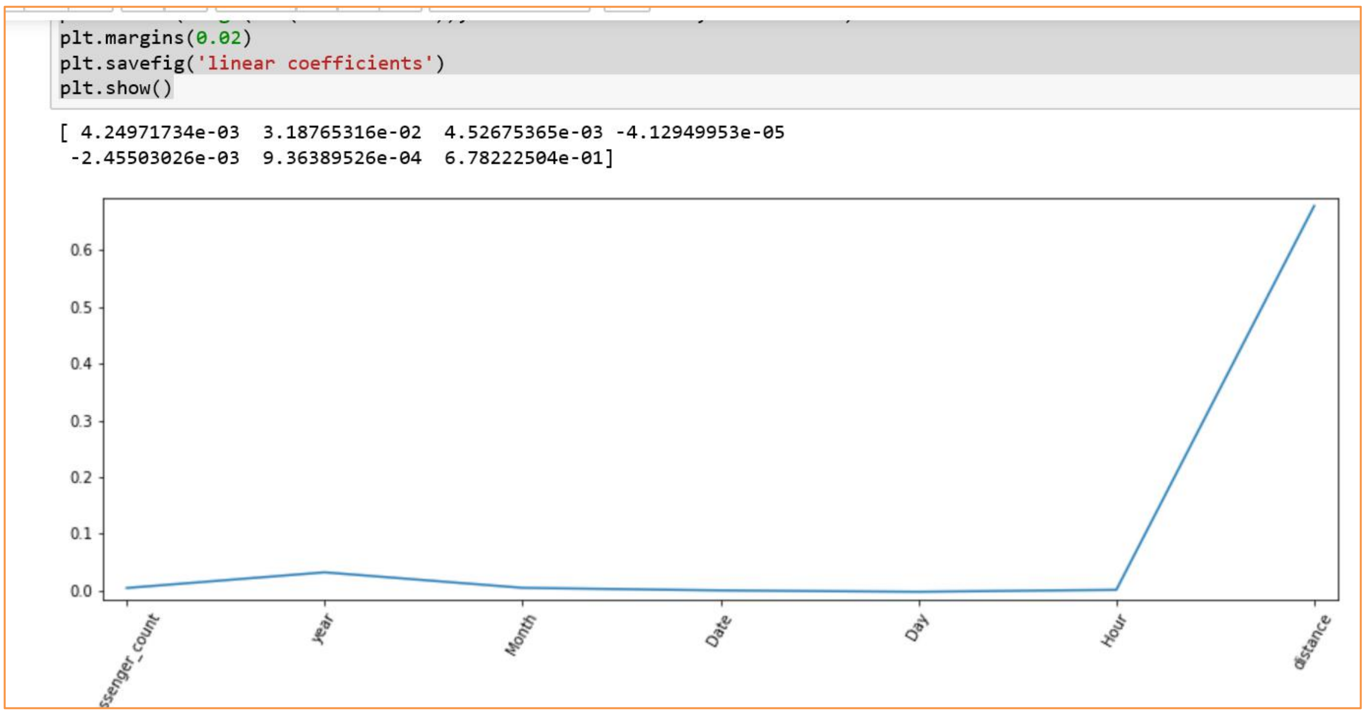
2.3.1 Model Performance

Here, we will evaluate the performance of different Regression models based on different Error Metrics

I. Multiple Linear Regression:

Error Metrics	r square	Adj r sq	MAPE	MSE	RMSE	RMSLE
Train	0.734	0.733	18.73	5.28	0.188221	0.21
Validation	0.719	0.7406	18.96	5.29	0.193667	0.21

Line Plot for Coefficients of Multiple Linear regression:



```
plt.margins(0.02)
plt.savefig('linear coefficients')
plt.show()
```

```
[ 4.24971734e-03  3.18765316e-02  4.52675365e-03 -4.12949953e-05
 -2.45503026e-03  9.36389526e-04  6.78222504e-01]
```

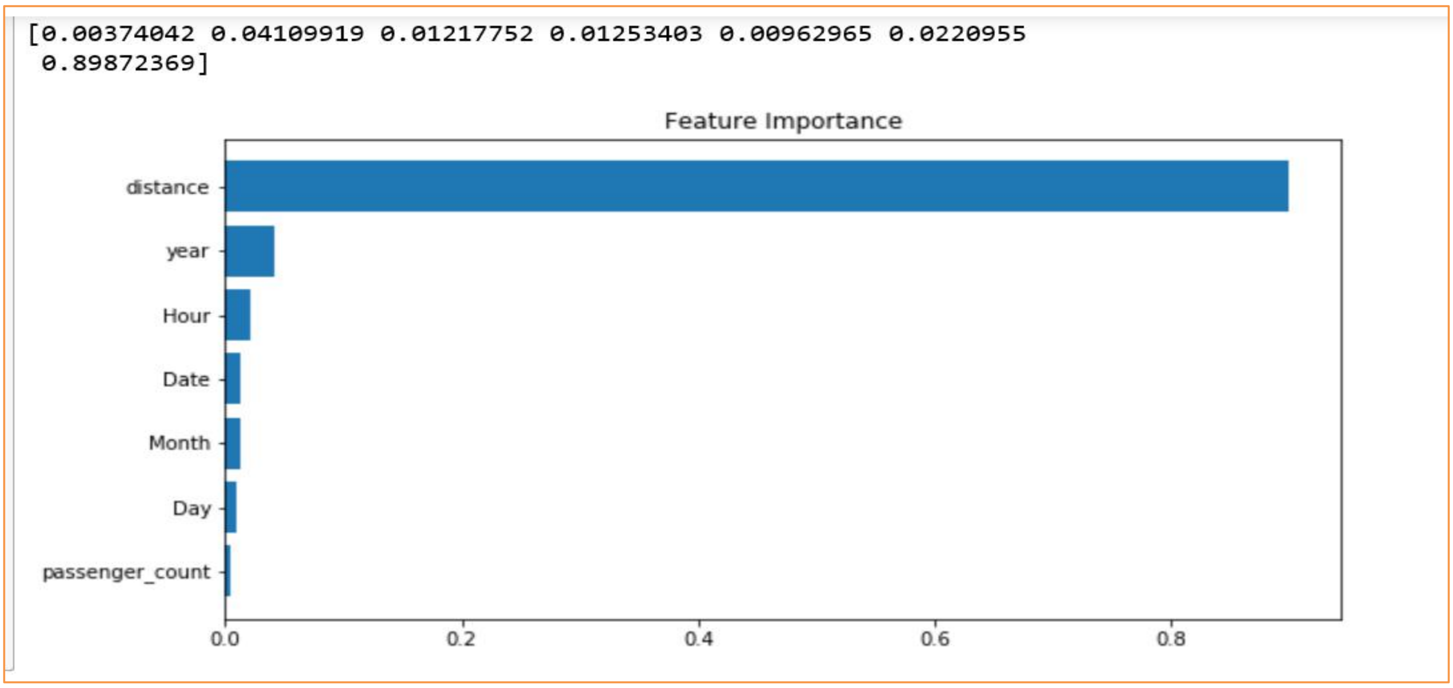
II. Decision Tree Regression:

Error Metrics	r square	Adj r sq	MAPE	MSE	RMSE	RMSLE
Train	0.65100	0.733	18.74	5.28	0.21942	0.21
validation	0.6628	0.7406	18.96	5.29	0.214411	0.21

III. Random Forest Regression:

Error Metrics	r square	Adj r sq	MAPE	MSE	RMSE	RMSLE
Train	0.96151	0.7337	18.75	5.28	0.07286	0.21
Validation	0.737321	0.7415	18.95	5.27	0.18955	0.21

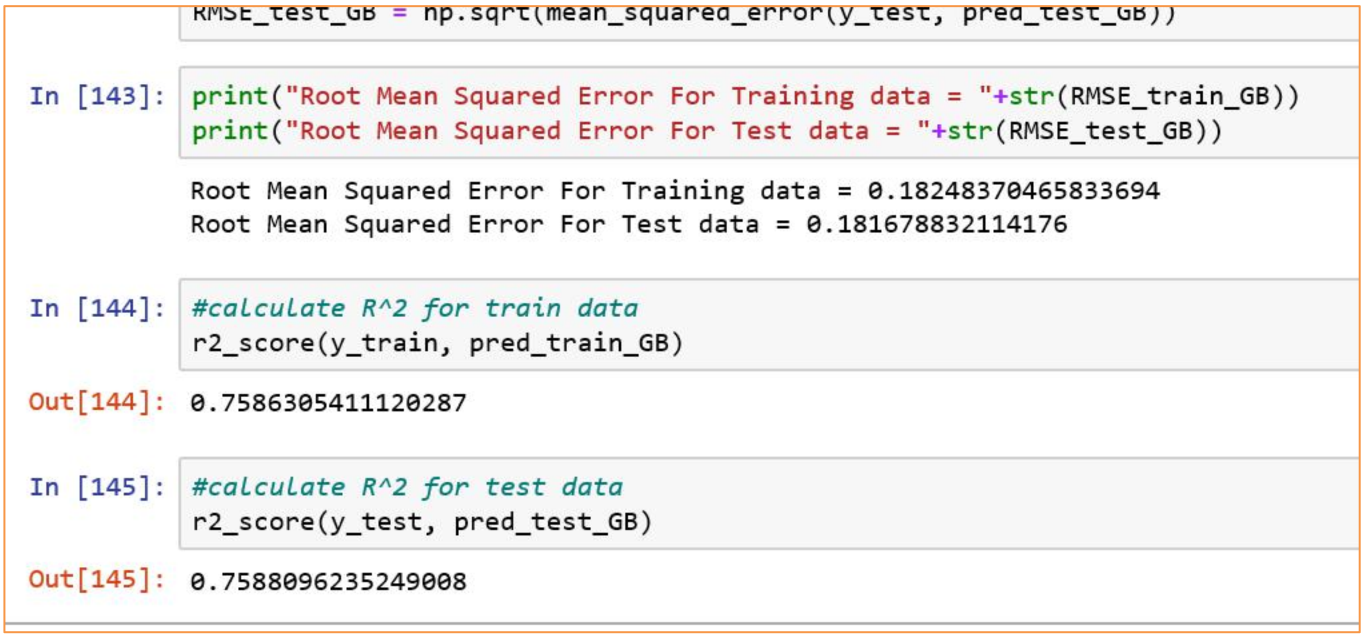
Feature Importance :-



```
[0.00374042 0.04109919 0.01217752 0.01253403 0.00962965 0.0220955
 0.89872369]
```

IV. Gradient Boosting Regression:

Error Metrics	r square	Adj r sq	MAPE	MSE	RMSE	RMSLE
Train	0.7586	0.7467	18.54	5.02	0.1824	0.20
Validation	0.75880	0.7396	19.07	5.31	0.18167	0.21



```
RMSE_test_GB = np.sqrt(mean_squared_error(y_test, pred_test_GB))

In [143]: print("Root Mean Squared Error For Training data = "+str(RMSE_train_GB))
          print("Root Mean Squared Error For Test data = "+str(RMSE_test_GB))

          Root Mean Squared Error For Training data = 0.18248370465833694
          Root Mean Squared Error For Test data = 0.181678832114176

In [144]: #calculate R^2 for train data
          r2_score(y_train, pred_train_GB)

Out[144]: 0.7586305411120287

In [145]: #calculate R^2 for test data
          r2_score(y_test, pred_test_GB)

Out[145]: 0.7588096235249008
```

Chapter 6

Improving accuracy

Optimizing the Models results with parameters tuning :

1. CV techniques = Random Search CV & Grid Search CV,
2. Apply both on our best both models ie 1) Random S CV on Random Forest & Gradient Boosting 2) Grid Search CV on Random Forest & Gradient Boosting

a. To find the optimal hyperparameter we have used ,
sklearn.model_selection. GridSearchCV. and
sklearn.model_selection. RandomizedSearchCV

b. GridSearchCV tries all the parameters that we provide it and then returns the best suited parameter for data.

c. We gave parameter dictionary to GridSearchCV which contains keys which are parameter names and values are the values of parameters which we want to try for.

Below are best hyperparameter we found for different models:

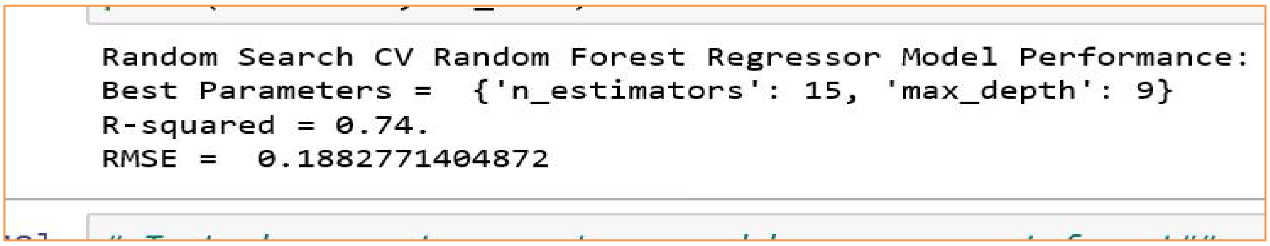
I. Multiple Linear Regression:
Tuned Decision reg Parameters: {'copy_X': True, 'fit_intercept': True}
Best score is 0.7354470072210966

II. Decision Tree Regression:
Tuned Decision Parameters: {'alpha': 0.0005428675439323859
, 'max_iter': 500, 'normalize': True}
Best score is 0.7354637543642097

III. Random Forest Model Regression:
Tuned Parameters: {'alpha': 0.00021209508879201905
, 'max_iter': 1000, 'normalize': False}
Best score is 0.740677751497154

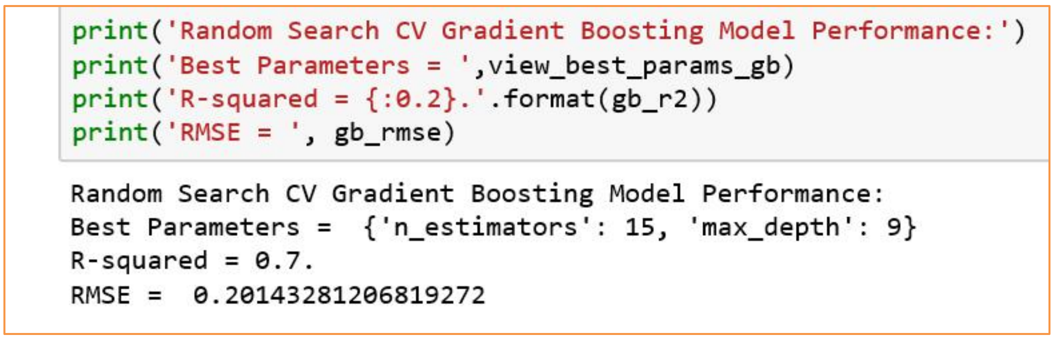
IV. Gradient Boosting :
Tuned Gradient Boosting Parameters: {'max_depth': 6, 'min_samples_split': 2}
Best score is 0.758813489270203365

- `Random Search CV on Random Forest Model:`



```
Random Search CV Random Forest Regressor Model Performance:
Best Parameters =  {'n_estimators': 15, 'max_depth': 9}
R-squared = 0.74.
RMSE =  0.1882771404872
```

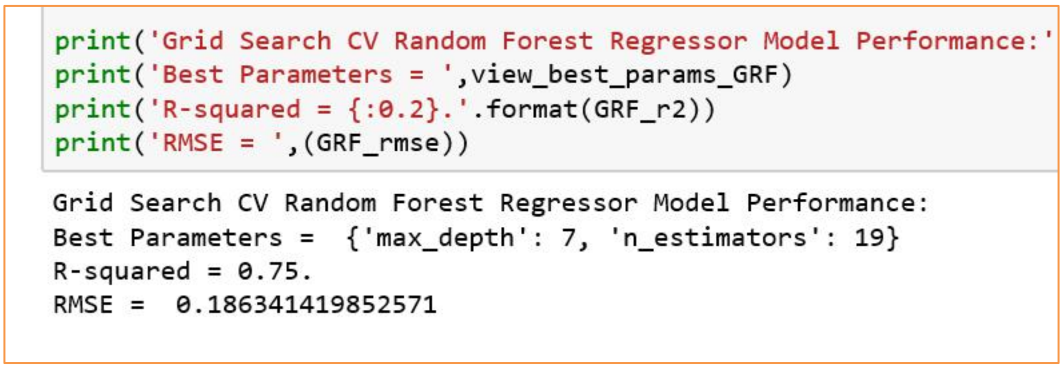
- Random Search CV on gradient boosting model



```
print('Random Search CV Gradient Boosting Model Performance:')
print('Best Parameters = ',view_best_params_gb)
print('R-squared = {:0.2}.'.format(gb_r2))
print('RMSE = ', gb_rmse)
```

```
Random Search CV Gradient Boosting Model Performance:
Best Parameters =  {'n_estimators': 15, 'max_depth': 9}
R-squared = 0.7.
RMSE =  0.20143281206819272
```

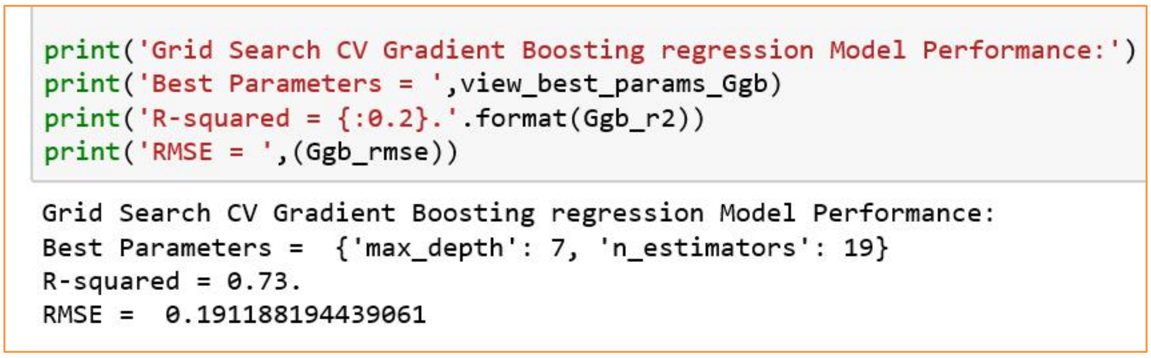
- Grid Search CV for random Forest model



```
print('Grid Search CV Random Forest Regressor Model Performance:'
print('Best Parameters = ',view_best_params_GRF)
print('R-squared = {:0.2}.'.format(GRF_r2))
print('RMSE = ',(GRF_rmse))
```

```
Grid Search CV Random Forest Regressor Model Performance:
Best Parameters =  {'max_depth': 7, 'n_estimators': 19}
R-squared = 0.75.
RMSE =  0.186341419852571
```

- Grid Search CV for gradinet boosting



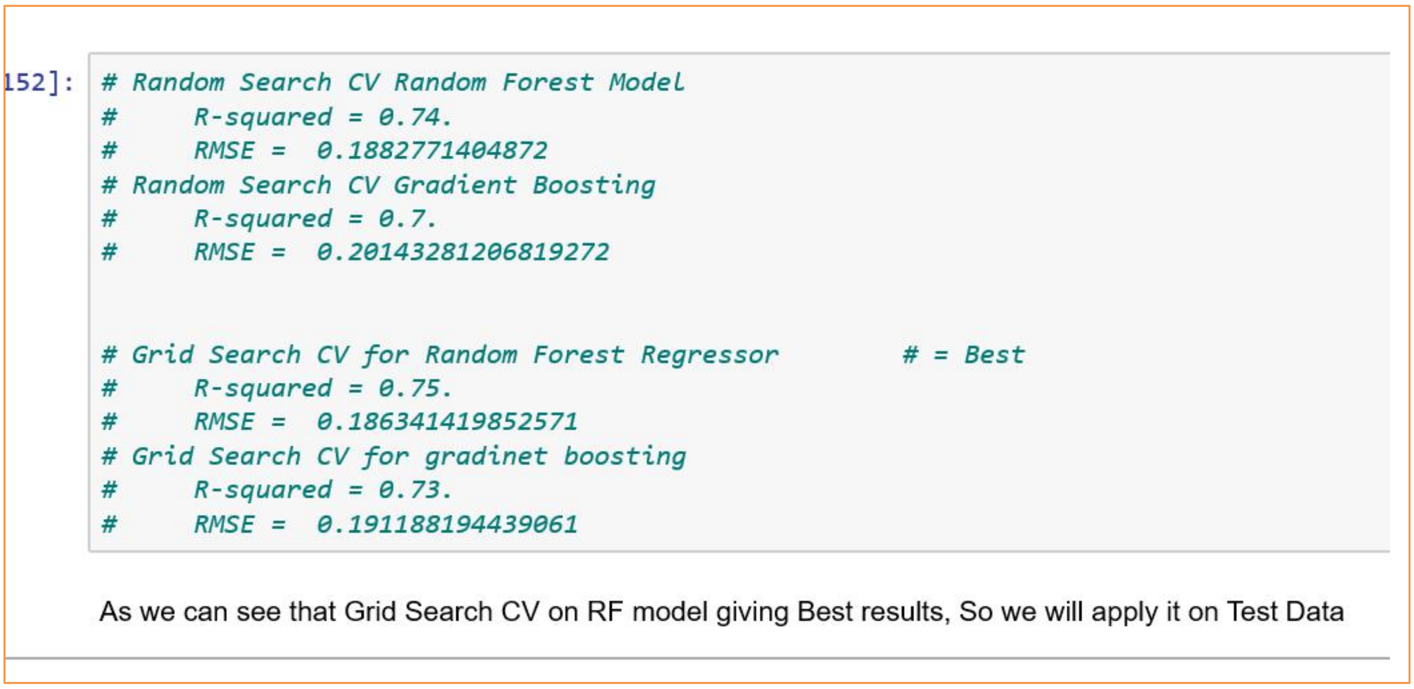
```
print('Grid Search CV Gradient Boosting regression Model Performance:')
print('Best Parameters = ',view_best_params_Ggb)
print('R-squared = {:0.2}.'.format(Ggb_r2))
print('RMSE = ',(Ggb_rmse))
```

```
Grid Search CV Gradient Boosting regression Model Performance:
Best Parameters =  {'max_depth': 7, 'n_estimators': 19}
R-squared = 0.73.
RMSE =  0.191188194439061
```

Chapter 7
Finalize model

- Create standalone model on entire training dataset
- Save model for later use

Conclusion :-



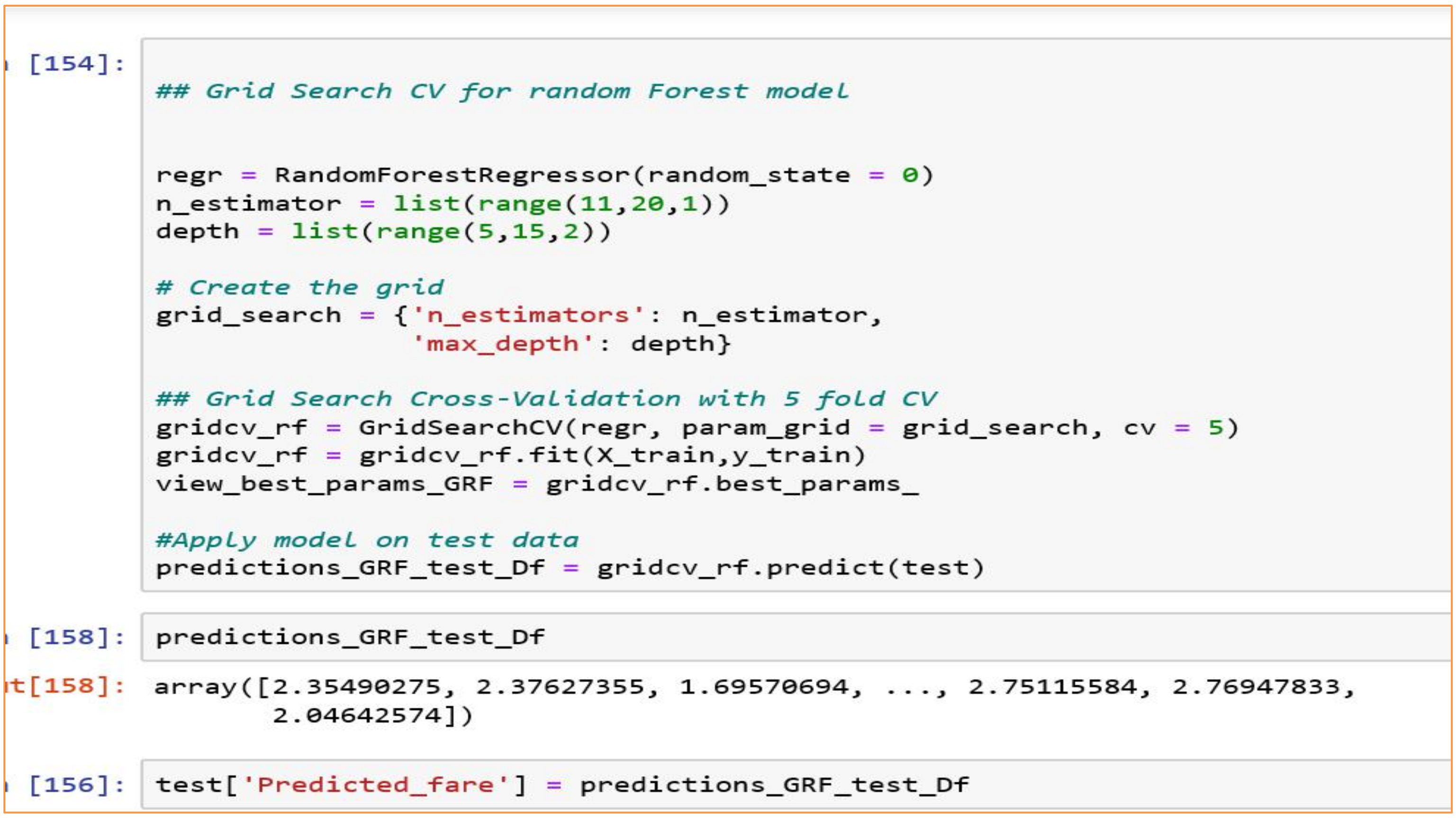
```
152]: # Random Search CV Random Forest Model
      #     R-squared = 0.74.
      #     RMSE =  0.1882771404872
      # Random Search CV Gradient Boosting
      #     R-squared = 0.7.
      #     RMSE =  0.20143281206819272


      # Grid Search CV for Random Forest Regressor        # = Best
      #     R-squared = 0.75.
      #     RMSE =  0.186341419852571
      # Grid Search CV for gradinet boosting
      #     R-squared = 0.73.
      #     RMSE =  0.191188194439061
```

As we can see that Grid Search CV on RF model giving Best results, So we will apply it on Test Data

As we can see that Grid Search CV on RF model giving Best results, So we will apply it on Test Data.

Prediction of fare from provided test dataset :



```
 [154]:
## Grid Search CV for random Forest model


regr = RandomForestRegressor(random_state = 0)
n_estimator = list(range(11,20,1))
depth = list(range(5,15,2))

# Create the grid
grid_search = {'n_estimators': n_estimator,
               'max_depth': depth}

## Grid Search Cross-Validation with 5 fold CV
gridcv_rf = GridSearchCV(regr, param_grid = grid_search, cv = 5)
gridcv_rf = gridcv_rf.fit(X_train,y_train)
view_best_params_GRF = gridcv_rf.best_params_

#Apply model on test data
predictions_GRF_test_Df = gridcv_rf.predict(test)
```

```
 [158]: predictions_GRF_test_Df
t[158]: array([2.35490275, 2.37627355, 1.69570694, ..., 2.75115584, 2.76947833,
               2.04642574])
```

```
 [156]: test['Predicted_fare'] = predictions_GRF_test_Df
```