

# Cab Fare Prediction

*Sagar Raj Roul*

# Contents

	<b>Introduction</b>	<b>3</b>
	Problem Statement	3
	Data	3
	<b>Methodology</b>	<b>4</b>
	Pre-Processing	4
	Removing values which are not within desired range(outlier) depending upon basic understanding of dataset.	9
	Missing Value Analysis	11
	Outlier Analysis	13
	Feature Engineering	15
	Feature Selection	19
	Feature Scaling	24
	<b>Splitting train and test Dataset</b>	<b>26</b>
	<b>Hyperparameter Optimization</b>	<b>27</b>
	<b>Model Development</b>	<b>28</b>
	Improving accuracy	34
	<b>Finalize Model</b>	<b>36</b>
	<b>Complete Python Code</b>	<b>37</b>

# 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

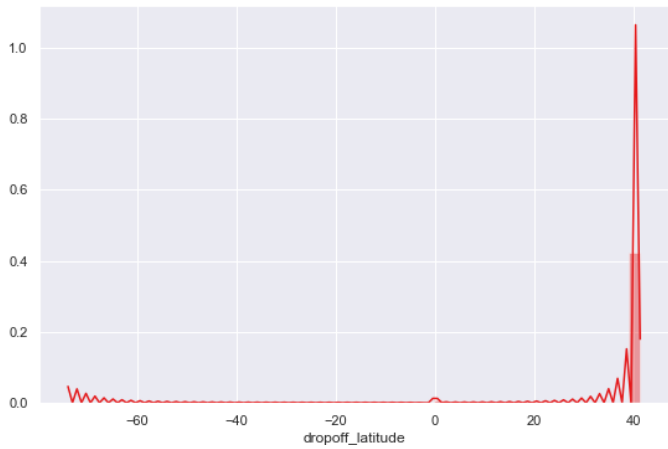
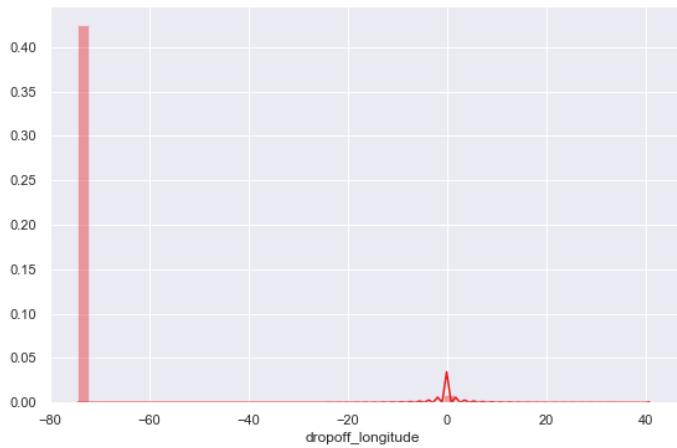
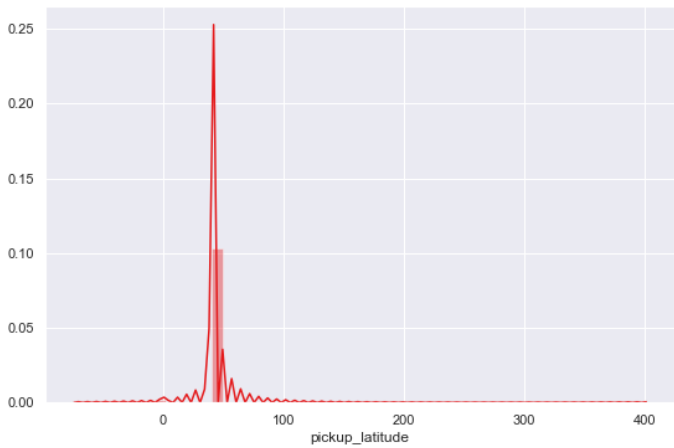
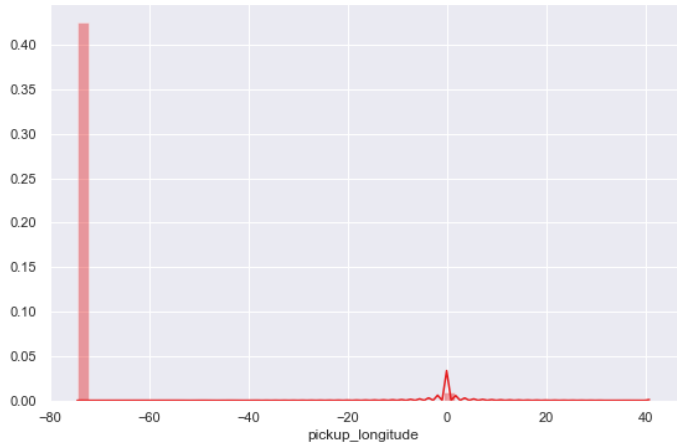
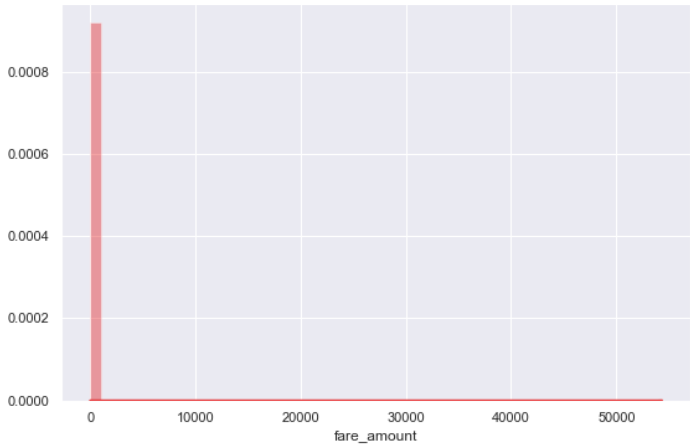
---

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.

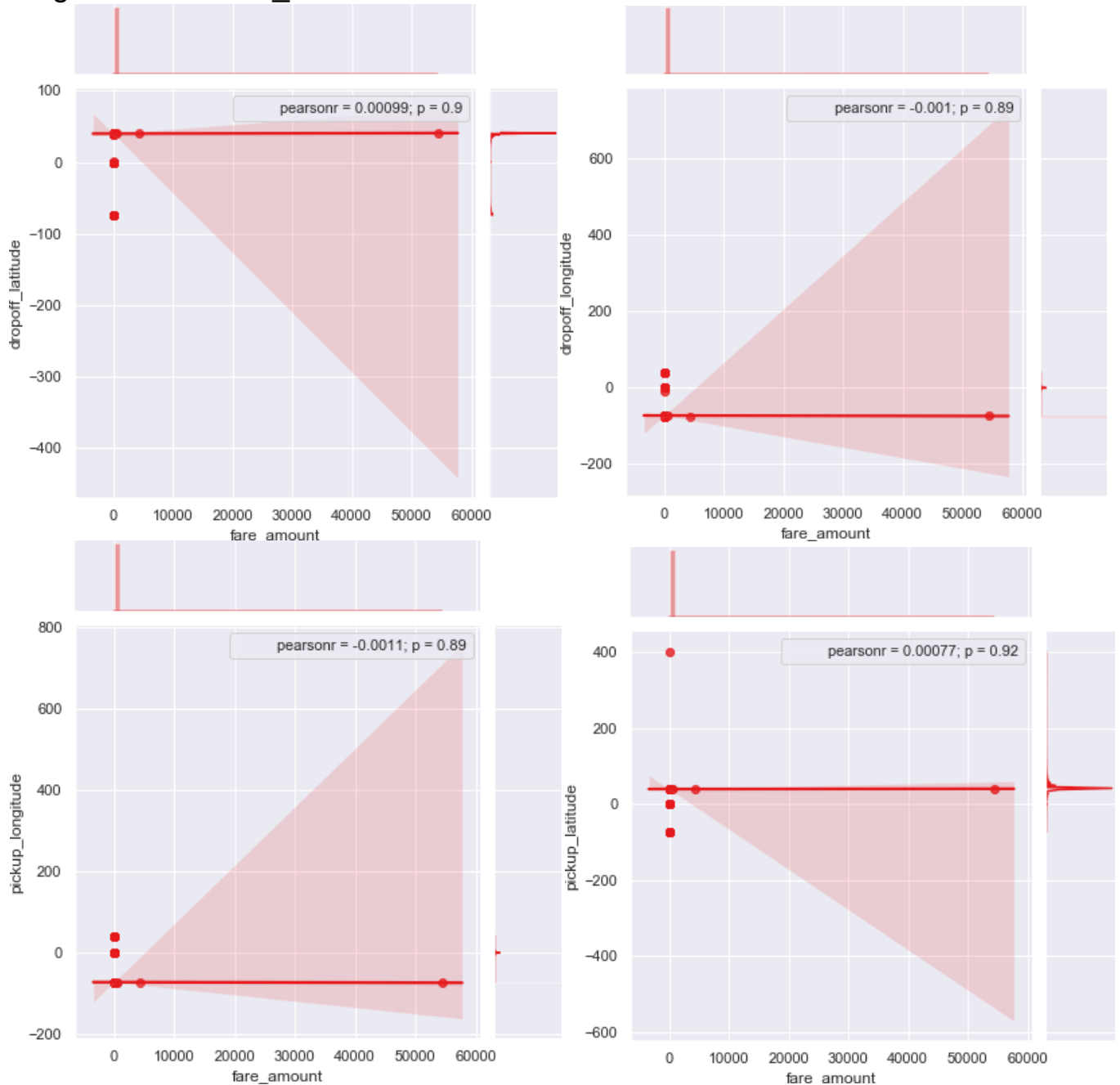
Getting feel of data via visualization:

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

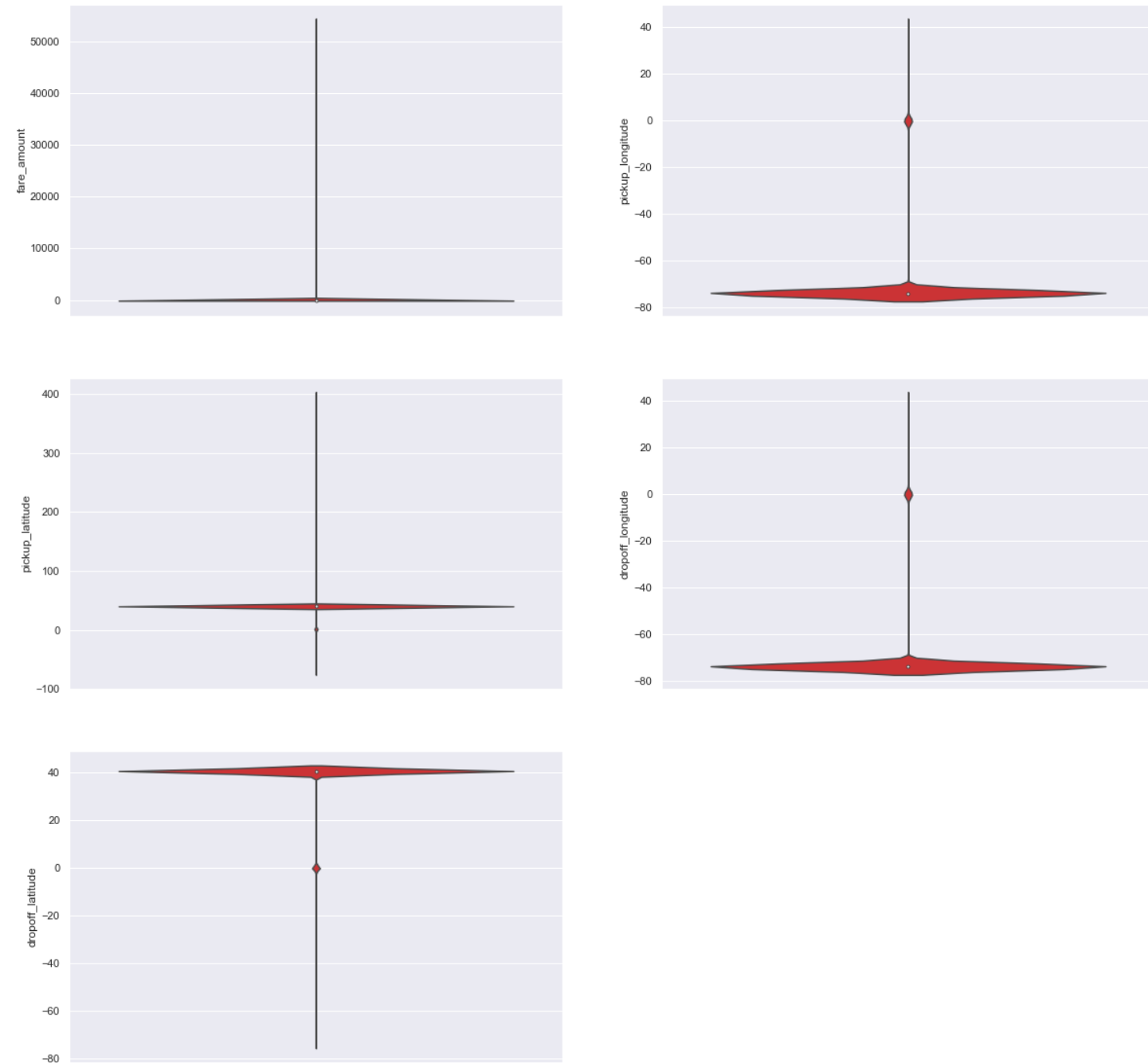


### Some Jointplots:

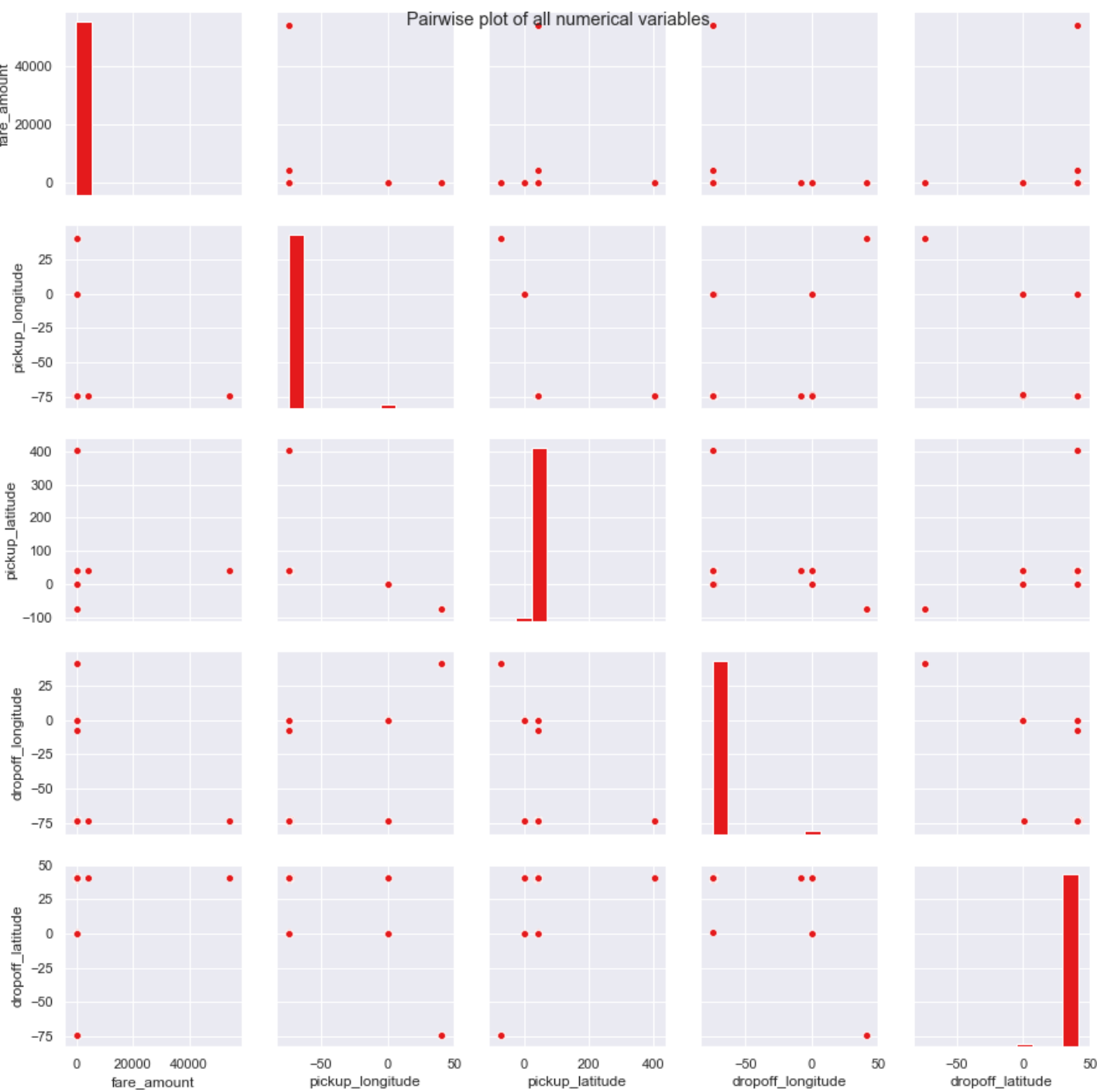
- They are used for Bivariate Analysis.
- Here we have plotted Scatter plot with Regression line between 2 variables along with separate Bar plots of both variables.
- Also, we have annotated Pearson correlation coefficient and p value.
- Plotted only for numerical/continuous variables
- Target variable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



Some Violin Plots to get the idea about till what range is the variables is spread.



Pairwise Plots for all Numerical variables:





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

In 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. You would think why haven't made those values NA instead of removing them well I did made 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



Variables	Missing_percentage	
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_percentage	
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'd tried central statistical methods and algorithmic method--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 deviation of the variable.

Standard deviation for passenger\_count and fare\_amount after KNN imputation:

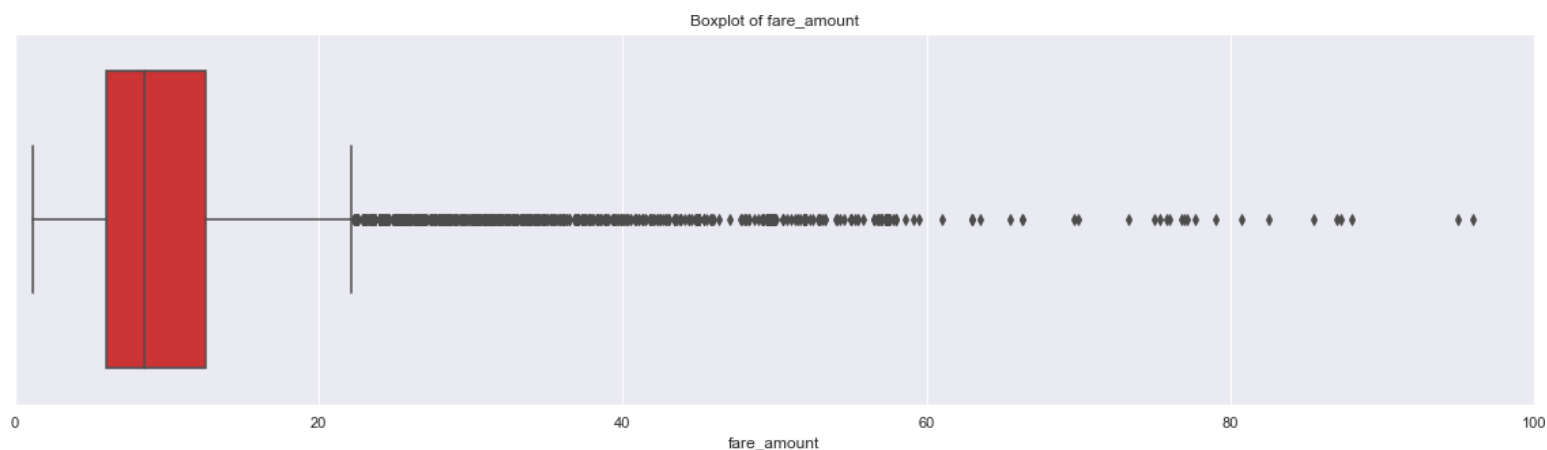
```
fare_amount    435.661995  
passenger_count  1.264322  
dtype: float64
```

## 2.1.3 Outlier Analysis

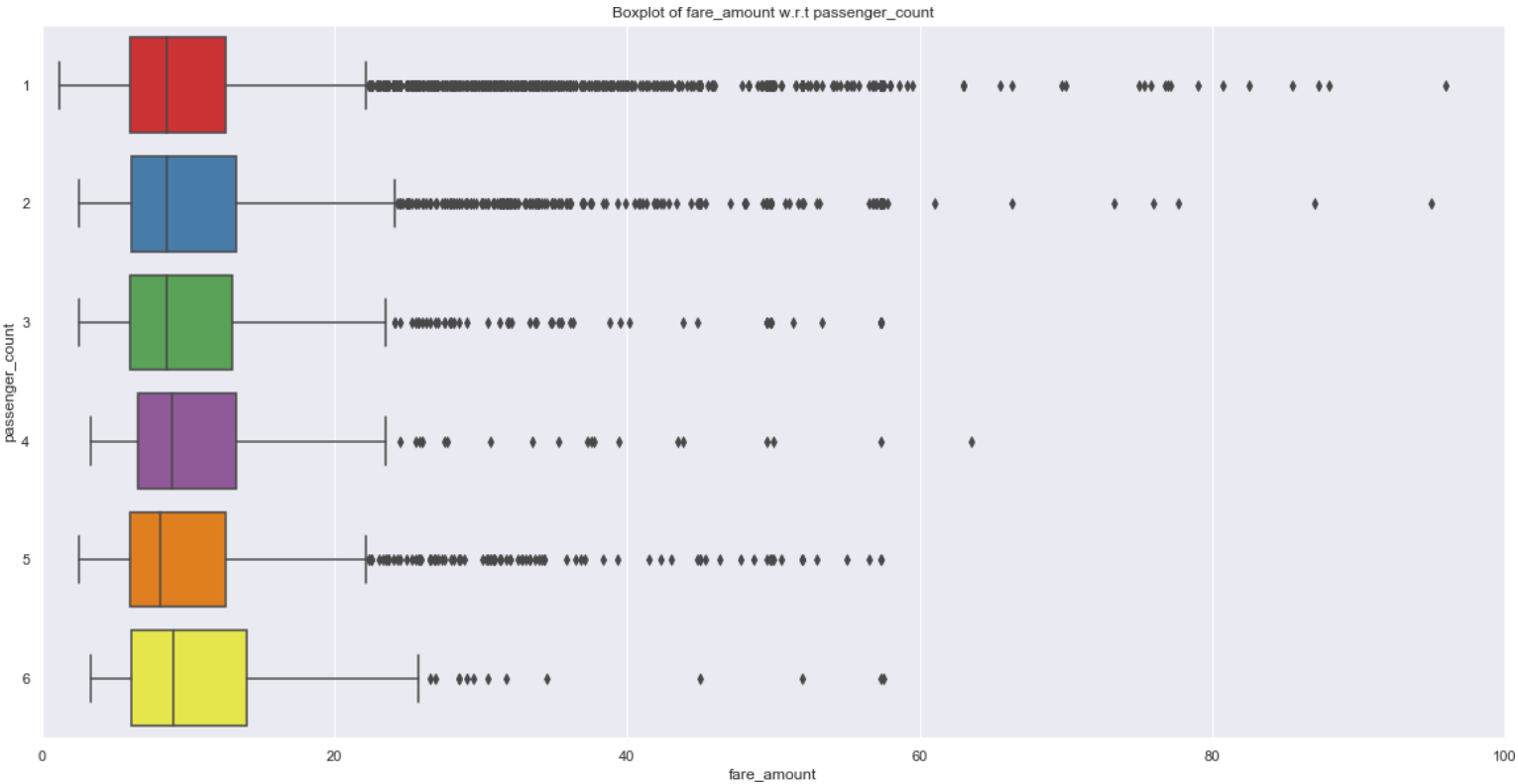
We look for outlier in the dataset by plotting Boxplots. There are outliers present in the data. we have removed these outliers. This is how we done,

- 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 latitudes 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 1359outliers.

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. For 'pickup\_datetime' variable:

We will use this timestamp variable to create new variables.

New features will be year, month, day\_of\_week, hour.

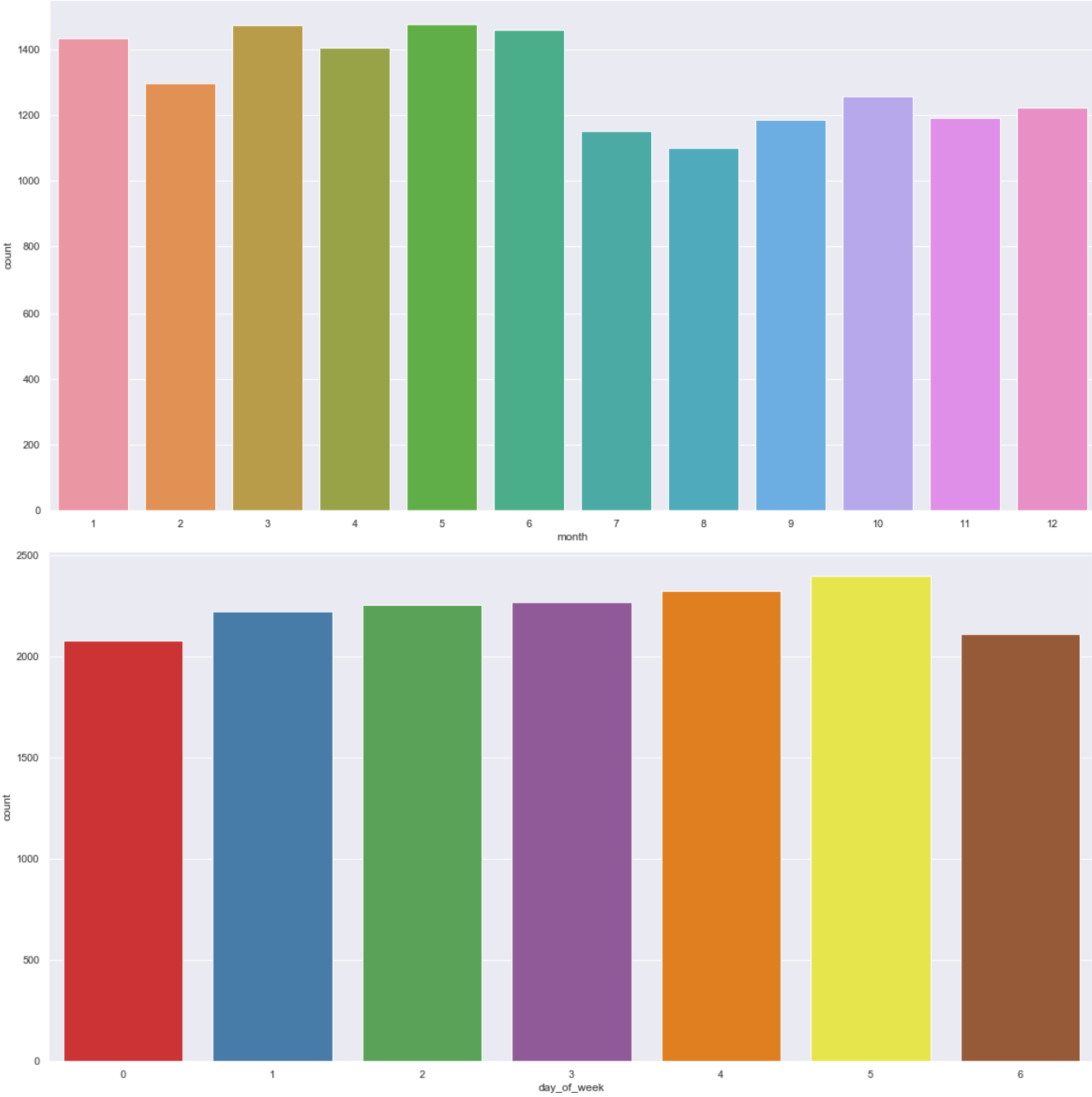
'year' will contain only years from pickup\_datetime. For ex. 2009, 2010, 2011, etc.

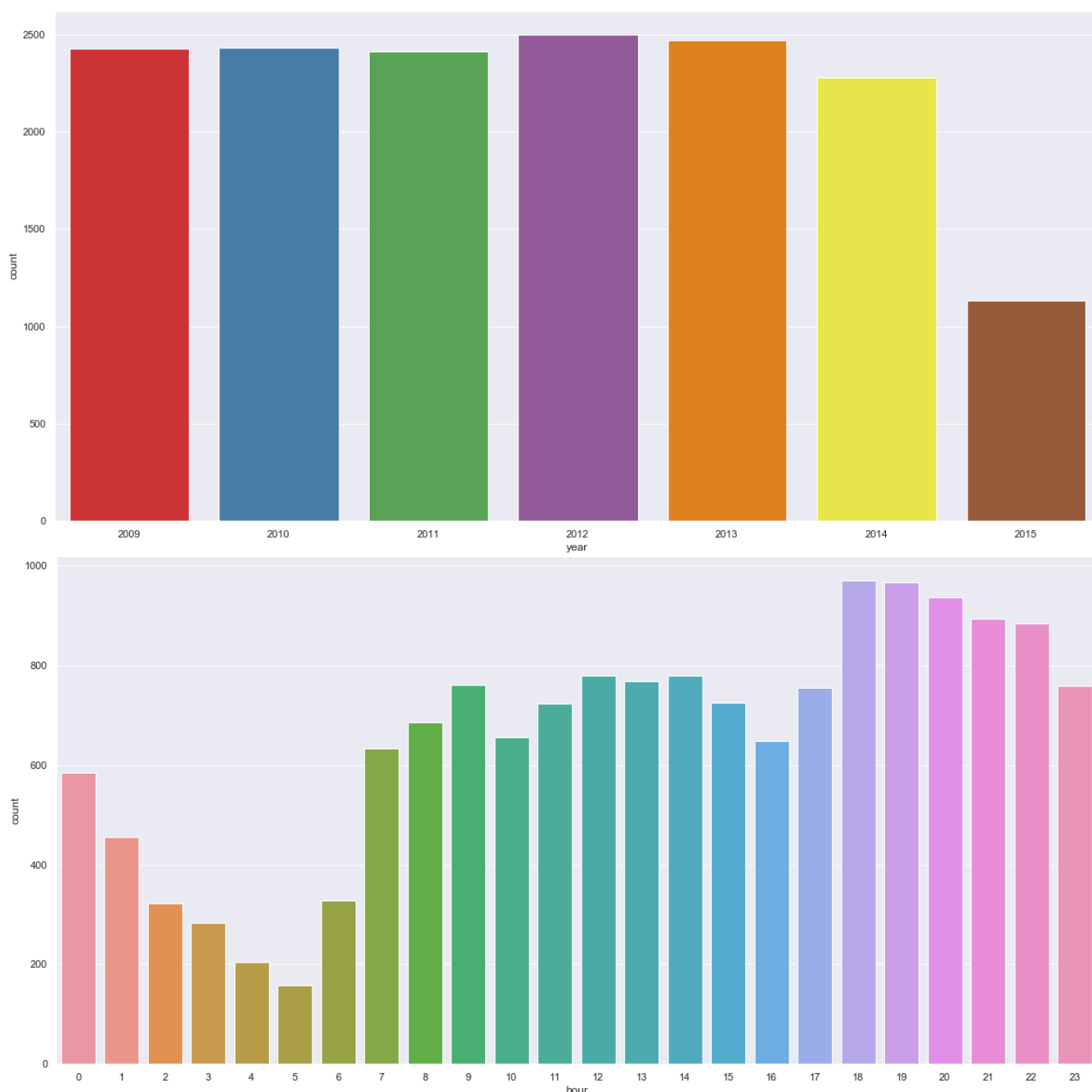
'month' will contain only months from pickup\_datetime. For ex. 1 for January, 2 for February, etc.

'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.







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. For 'passenger\_count' variable:

As passenger\_count is a categorical variable we will one-hot-encode it.

### 3. For 'Latitudes' and 'Longitudes' variables:

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 preferred for short distances.

Therefore, we will drop great\_circle.

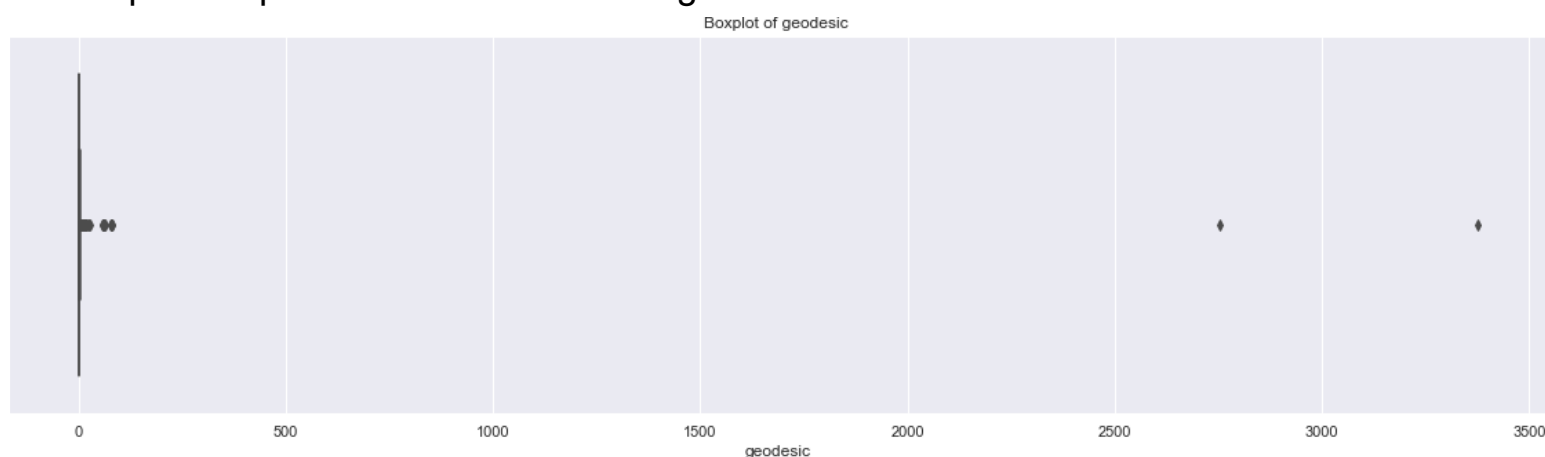
Columns in training data after feature engineering:

```
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:

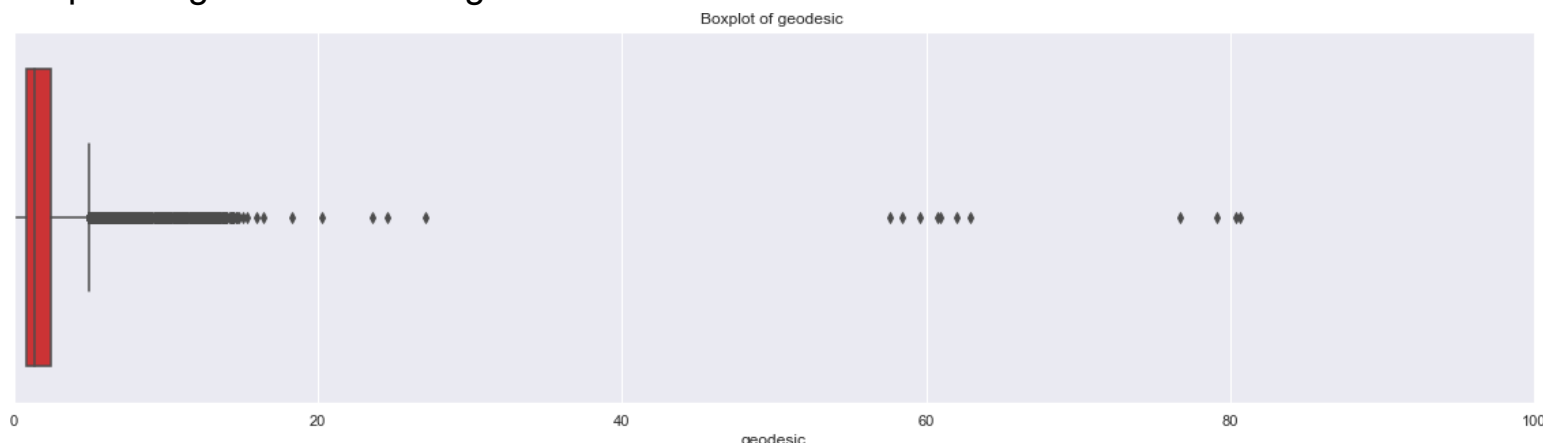
```
Index(['passenger_count_2', 'passenger_count_3', 'passenger_count_4',
      'passenger_count_5', 'passenger_count_6', 'season_spring',
      'season_summer', 'season_winter', 'week_weekend', 'session_evening',
      'session_morning', 'session_night_AM', 'session_night_PM', 'year_2010',
      'year_2011', '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.

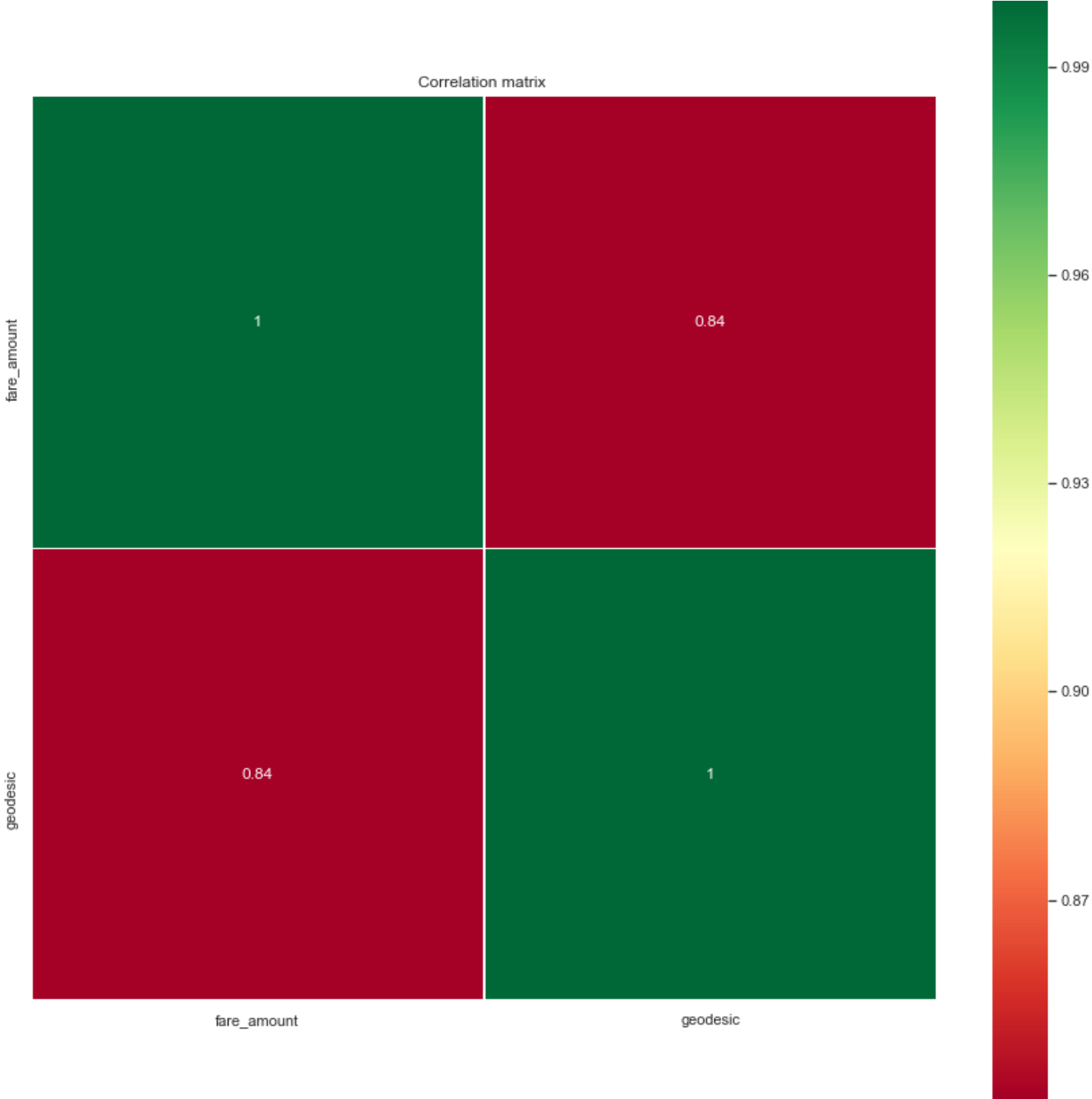
Further below are some types of test involved for feature selection:

- 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.

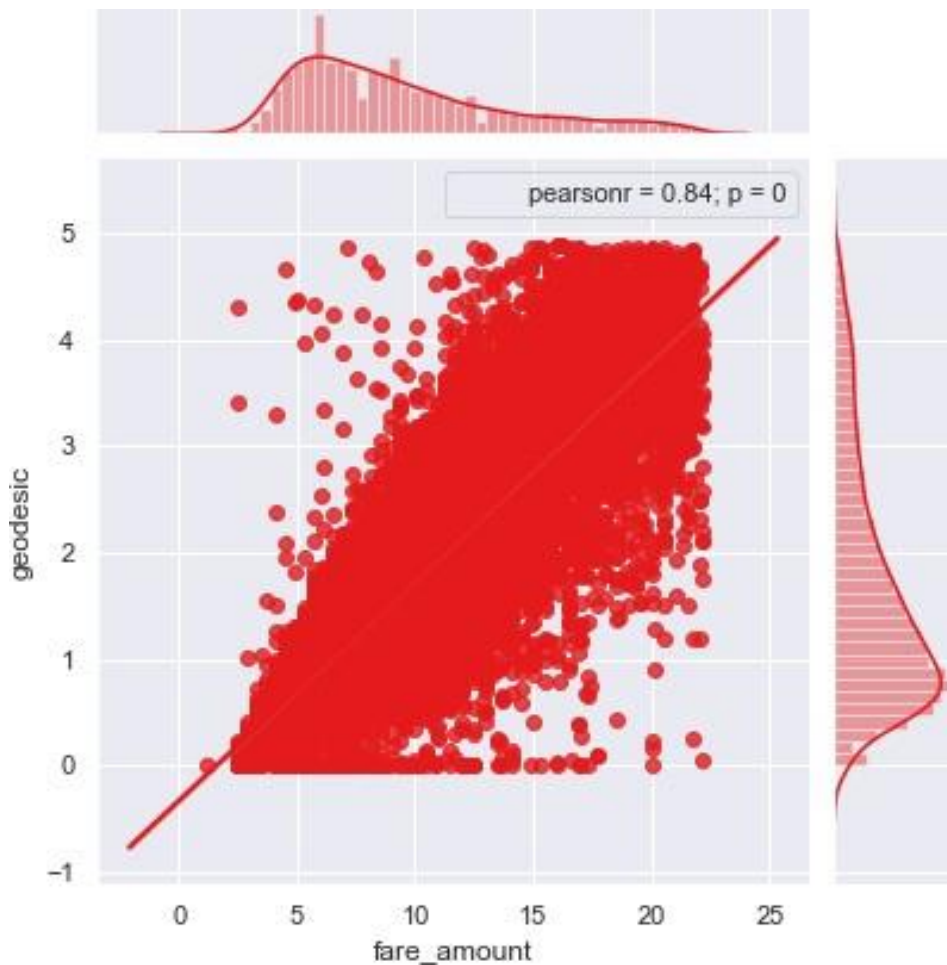
From below correlation plot we see that:

- '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 keep 'geodesic' because it will help to explain variation in fare\_amount.

Correlation Plot:



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 theoretical 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  $p\text{-value} < 0.05$  then remove the variable,

- If  $p\text{-value} > 0.05$  then keep the variable.

### 3 Analysis of Variance(Anova) Test –

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

Hypothesis testing:

- **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  $p\text{-value}$  is less than 0.05 then we reject the null hypothesis.
- And if  $p\text{-value}$  is greater than 0.05 then we accept the null hypothesis.

Below is the anova analysis table for each categorical variable:

	df	sum_sq	mean_sq	F	PR(>F)
C(passenger_count_2)	1.0	10.881433	10.881433	0.561880	4.535152e-01
C(passenger_count_3)	1.0	17.098139	17.098139	0.882889	3.474262e-01
C(passenger_count_4)	1.0	63.987606	63.987606	3.304099	6.912635e-02
C(passenger_count_5)	1.0	21.227640	21.227640	1.096122	2.951349e-01
C(passenger_count_6)	1.0	145.904989	145.904989	7.534030	6.061341e-03
C(season_spring)	1.0	28.961298	28.961298	1.495461	2.213894e-01
C(season_summer)	1.0	26.878639	26.878639	1.387920	2.387746e-01
C(season_winter)	1.0	481.664803	481.664803	24.871509	6.193822e-07
C(week_weekend)	1.0	130.676545	130.676545	6.747686	9.395730e-03
C(session_night_AM)	1.0	2130.109284	2130.109284	109.991494	1.197176e-25
C(session_night_PM)	1.0	185.382247	185.382247	9.572500	1.978619e-03
C(session_evening)	1.0	0.972652	0.972652	0.050224	8.226762e-01
C(session_morning)	1.0	48.777112	48.777112	2.518682	1.125248e-01
C(year_2010)	1.0	1507.533635	1507.533635	77.843835	1.231240e-18
C(year_2011)	1.0	1332.003332	1332.003332	68.780056	1.189600e-16
C(year_2012)	1.0	431.018841	431.018841	22.256326	2.406344e-06
C(year_2013)	1.0	340.870175	340.870175	17.601360	2.738958e-05
C(year_2014)	1.0	1496.882424	1496.882424	77.293844	1.624341e-18
C(year_2015)	1.0	2587.637234	2587.637234	133.616659	8.839097e-31
Residual	15640.0	302886.232626	19.366127	NaN	NaN

Looking at above table every variable has  $p$  value less than 0.05 so reject the null hypothesis.

- ### 4 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.

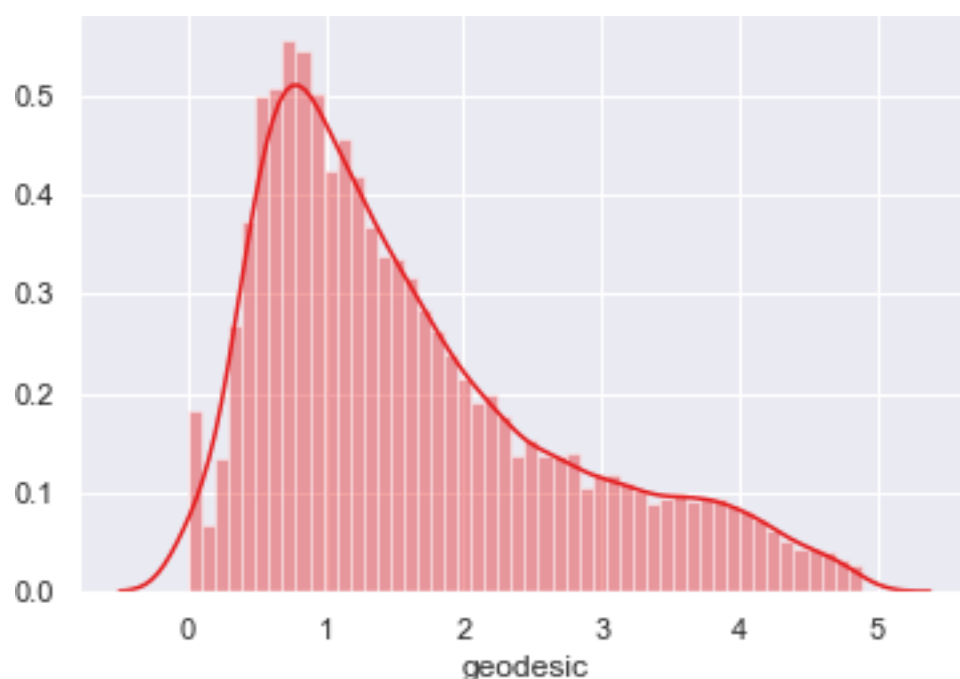
Also, our independent numerical variable 'geodesic' is not distributed normally so we had chosen normalization over standardization.

- 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.

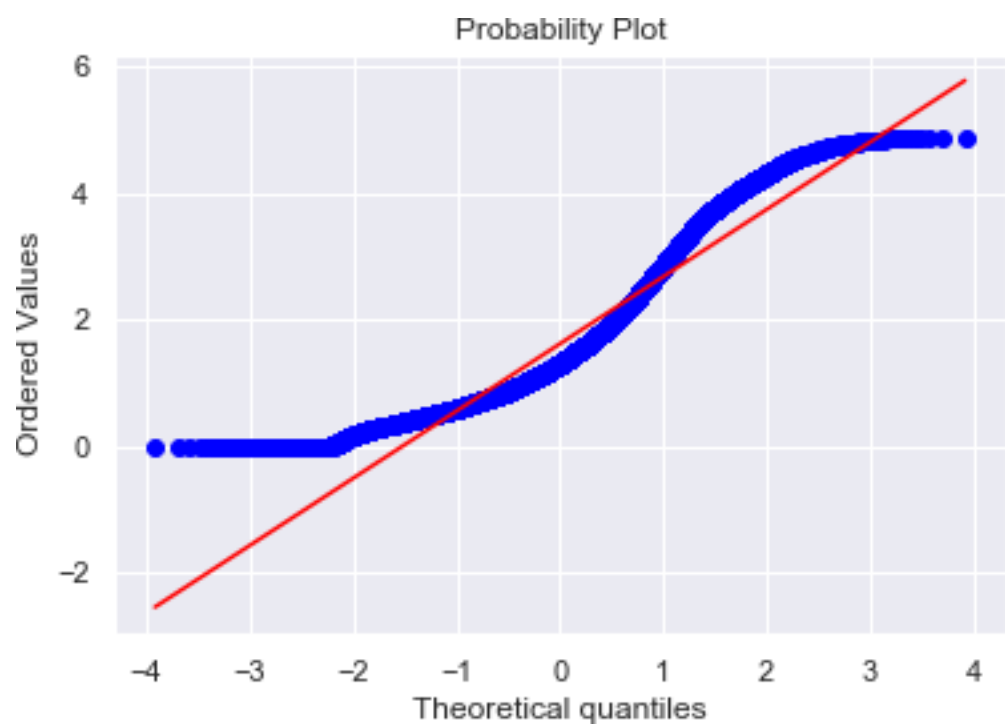
Graphs based on which standardization was chosen:

Note: It is performed only on Continuous variables.

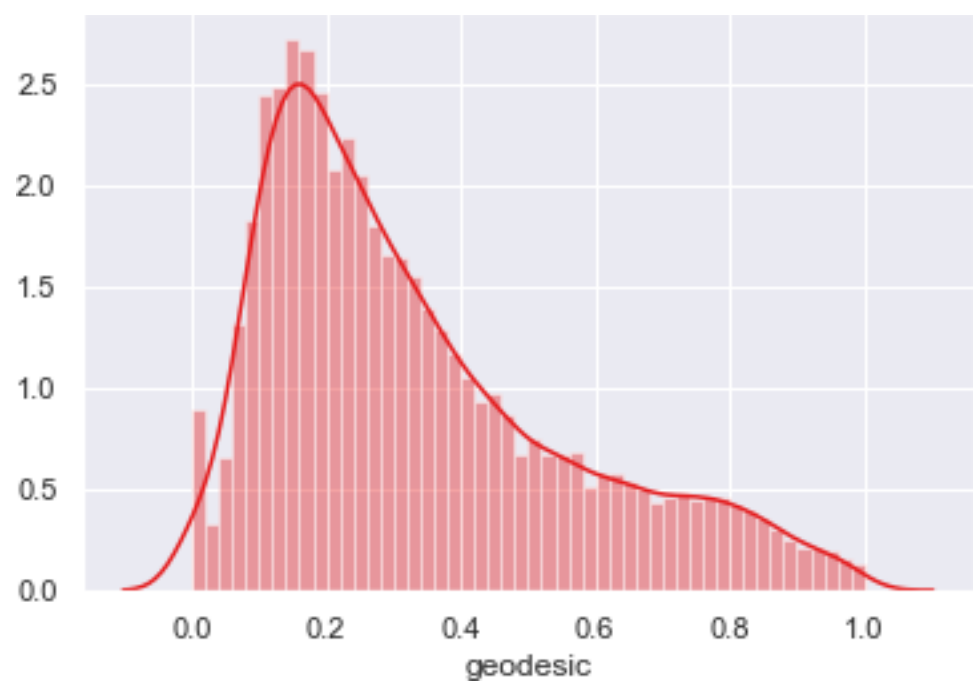
distplot() for 'geodesic' feature before normalization:



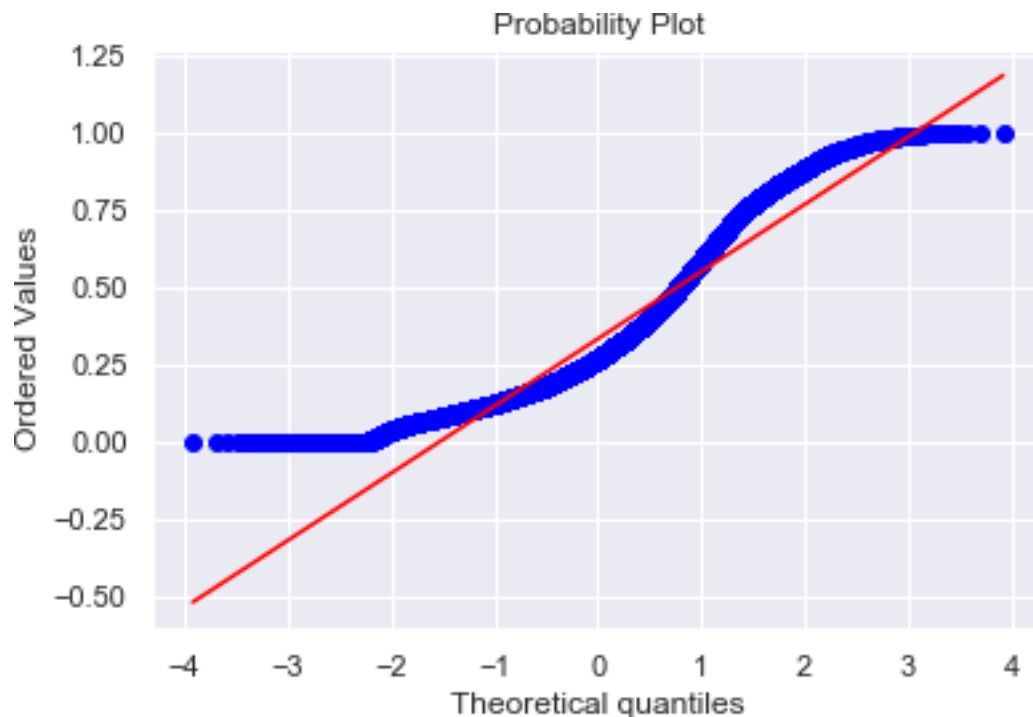
qq probability plot before normalization:



distplot() for 'geodesic' feature after normalization:



qq probability plot after normalization:



## Chapter 3

# Splitting train and Validation Dataset

- We have used sklearn's `train_test_split()` method to divide whole Dataset into train and validation dataset.
- 25% is in validation dataset and 75% is in training data.
- 11745 observations in training and 3915 observations in validation dataset.
- We will test the performance of model on validation dataset.
- The model which performs best will be chosen to perform on test dataset provided along with original train dataset.
- `X_train y_train`--are train subset.
- `X_test y_test`--are validation subset.

## Chapter 4

# Hyperparameter Optimization

---

- 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 Tree Parameters: {'max\_depth': 6, 'min\_samples\_split': 2}

Best score is 0.7313489270203365

III. Random Forest Regression:

Tuned Decision Forest Parameters: {'n\_estimators': 100, 'min\_samples\_split': 2, 'min\_samples\_leaf': 4, 'max\_features': 'auto', 'max\_depth': 9, 'bootstrap': True}

Best score is 0.7449373558797026

## IV. Xgboost regression:

Tuned Xgboost Parameters: {'subsample': 0.1, 'reg\_alpha': 0.08685113737513521, 'n\_estimators': 200, 'max\_depth': 3, 'learning\_rate': 0.05, 'colsample\_bytree': 0.7000000000000001, 'colsample\_bynode': 0.7000000000000001, 'colsample\_bylevel': 0.9000000000000001}

Best score is 0.7489532917329004

## Chapter 5

# 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
- III. Random Forest
- IV. 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 Error)

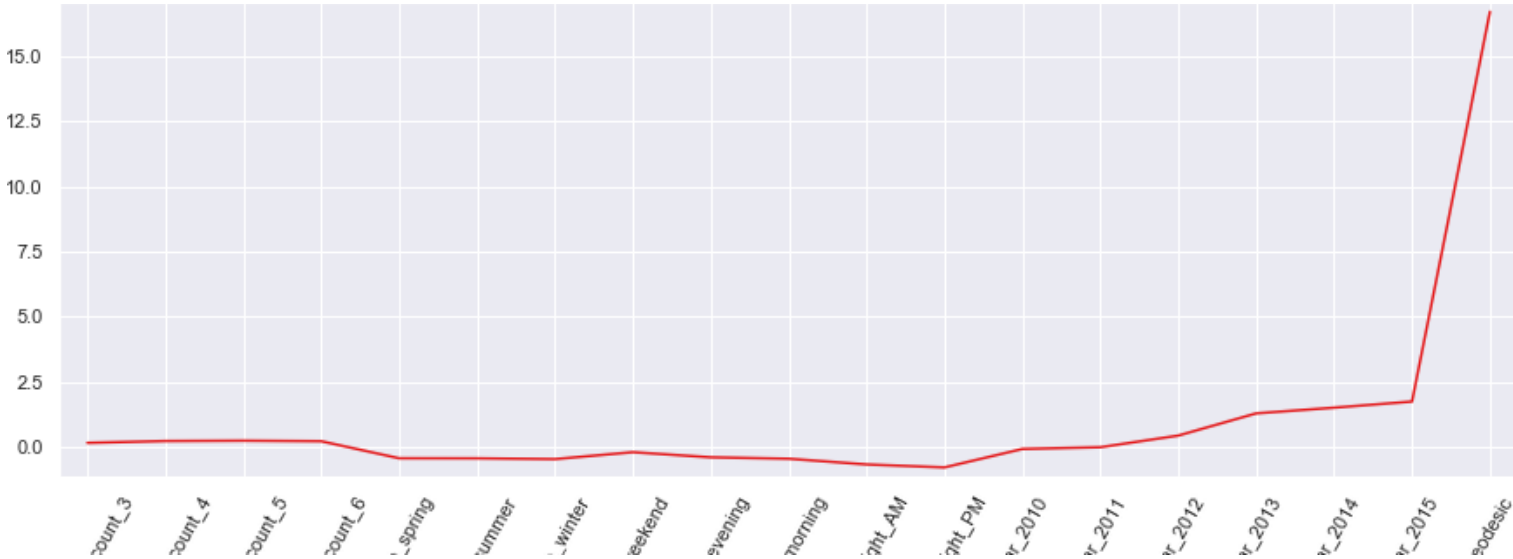
### 2.3.1 Model Performance

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

Linear Regression:

Error Metrics	r square	Adj rsq	MAPE	MSE	RMSE	RMSLE
Train	0.734	0.733	18.73	5.28	2.29	0.21
Validation	0.719	0.7406	18.96	5.29	2.30	0.21

Line Plot for Coefficients of Multiple Linear regression:





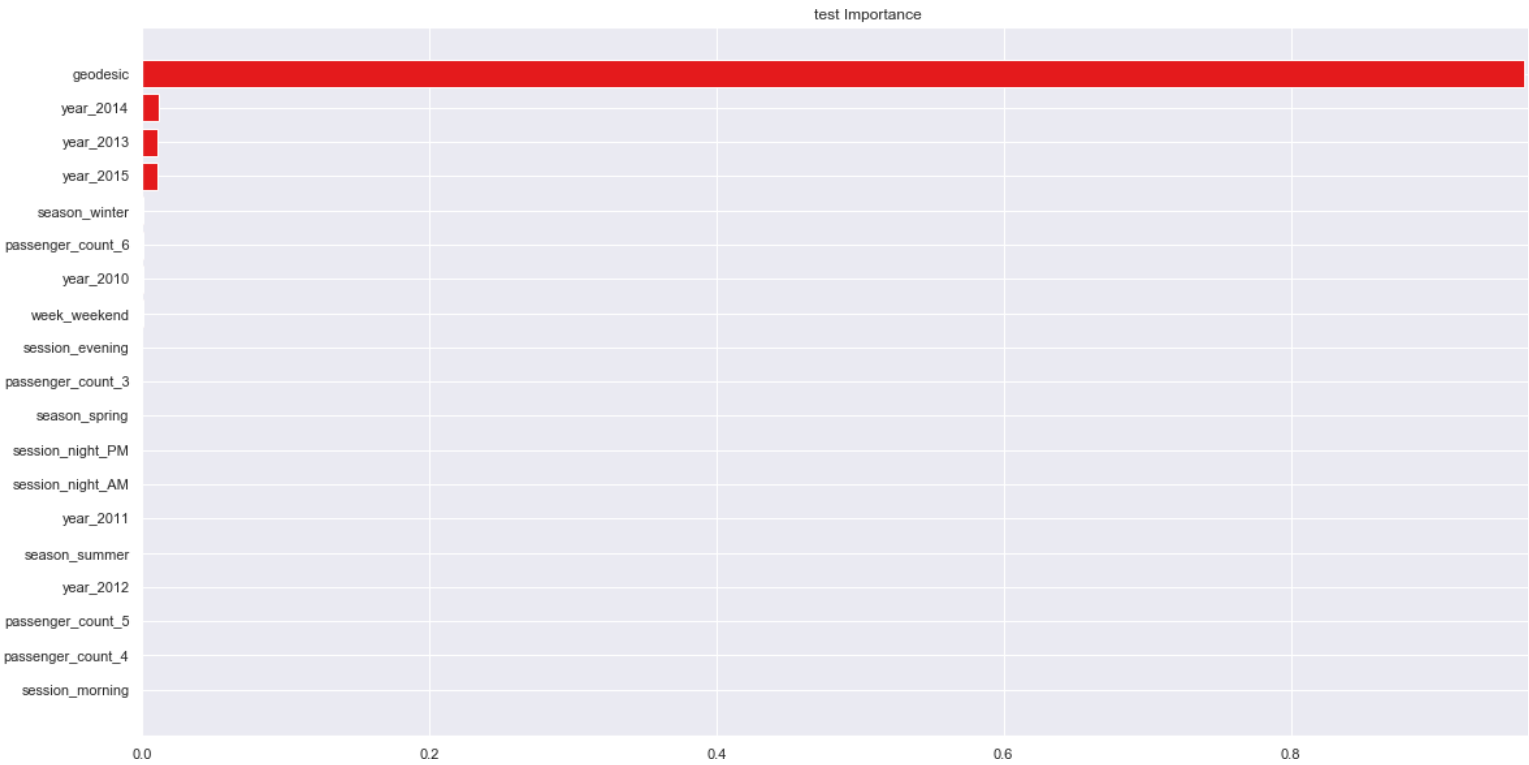




Decision Tree Regression:

Error Metrics	r square	Adj rsq	MAPE	MSE	RMSE	RMSLE
Train	0.7471	0.7467	18.54	5.02	2.24	0.20
Validation	0.7408	0.7396	19.07	5.31	2.30	0.21

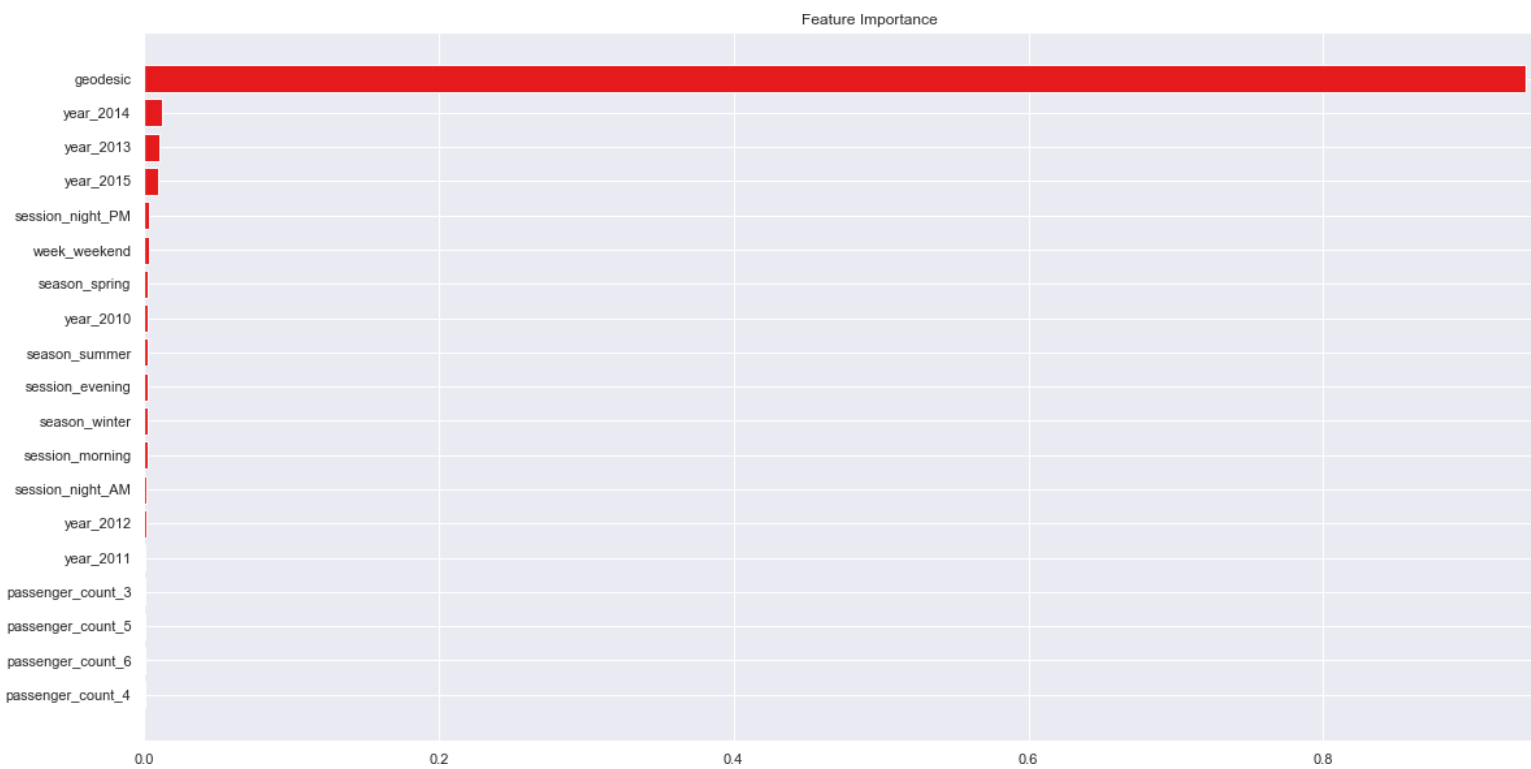
Bar Plot of Decision tree Feature Importance:



Random Forest Regression:

Error Metrics	r square	Adj rsq	MAPE	MSE	RMSE	RMSLE
Train	0.7893	0.7889	16.95	4.19	2.04	0.19
Validation	0.7542	0.7530	18.56	5.09	2.24	0.20

Bar Plot of Random Forest Feature Importance:



Cross validation scores: [-5.19821639 -5.18058997 -5.11306209 -5.15194135 -5.14644304]  
Average 5-Fold CV Score: -5.158050568861664

# Chapter 6

## Improving accuracy

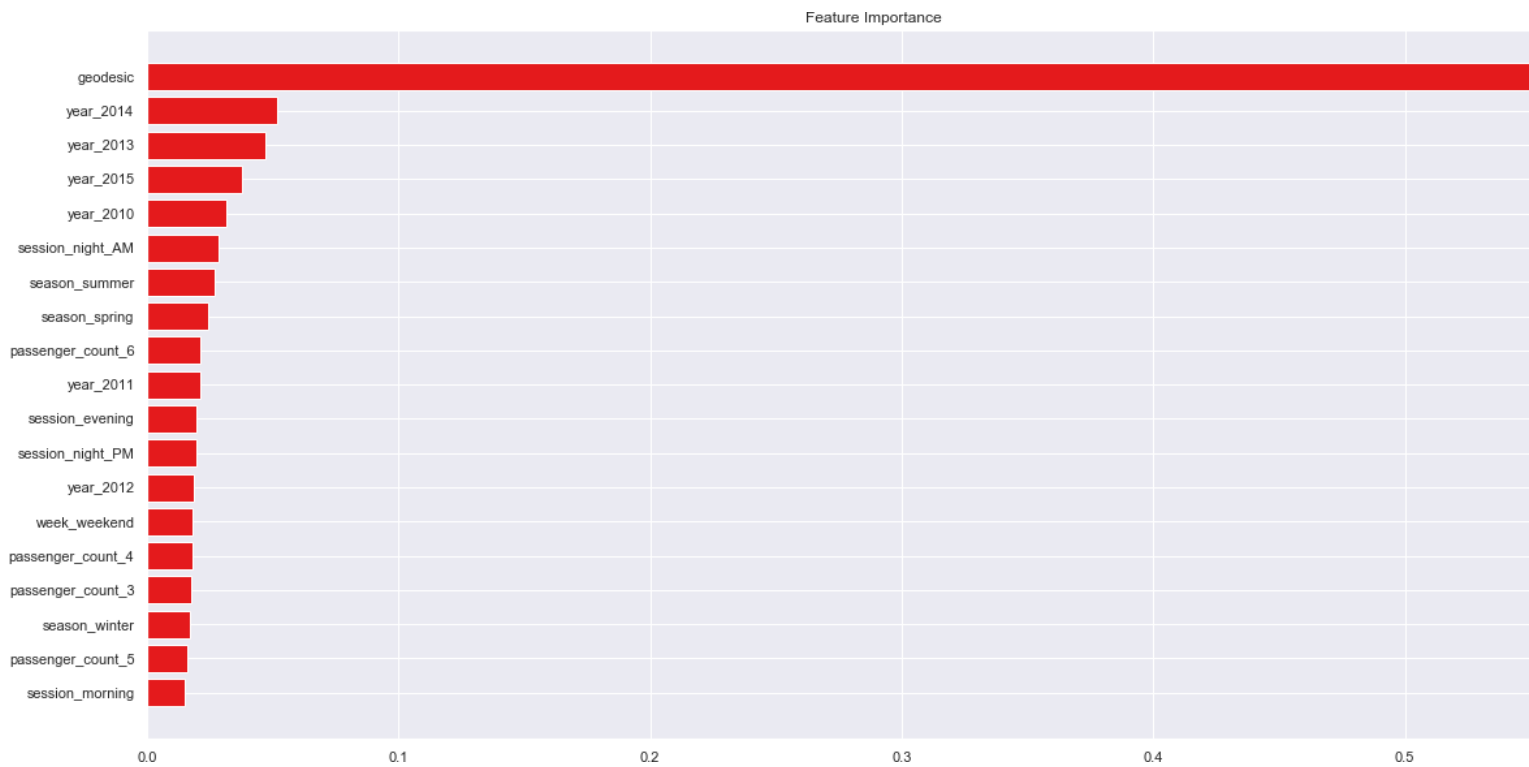
- Improve Accuracy a) Algorithm Tuning b) Ensembles
- We have used xgboost as a ensemble technique.

Xgboost hyperparameters tuned parameters: Tuned Xgboost Parameters: {'subsample': 0.1, 'reg\_alpha': 0.08685113737513521, 'n\_estimators': 200, 'max\_depth': 3, 'learning\_rate': 0.05, 'colsample\_bytree': 0.70000000000000001, 'colsample\_bynode': 0.70000000000000001, 'colsample\_bylevel': 0.90000000000000001}

Xgboost Regression:

Error Metrics	r square	Adj rsq	MAPE	MSE	RMSE	RMSLE
Train	0.7542	0.7538	18.15	4.88	2.21	0.20
Validation	0.7587	0.7575	18.37	4.96	2.22	0.20

Bar Plot of Xgboost Feature Importance:



# Chapter 7

## Finalize model

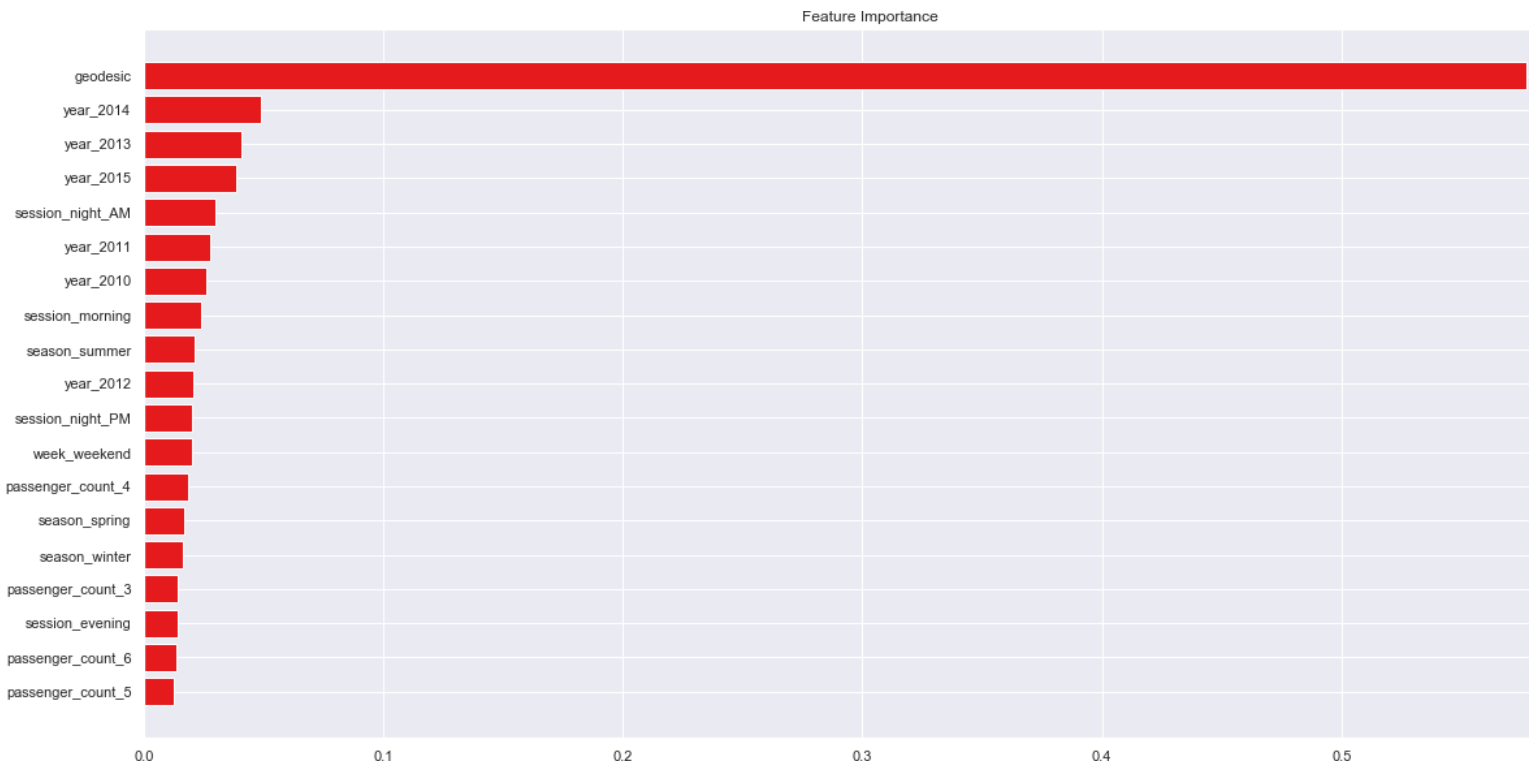
- Create standalone model on entire training dataset
- Save model for later use

We have trained a Xgboost model on entire training dataset and used that model to predict on test data. Also, we have saved model for later use.

<<<----- Training Data Score ----->

r square 0.7564292952182666  
Adjusted r square:0.7561333973032505  
MAPE:18.100202501103993  
MSE: 4.881882644209386  
RMSE: 2.2094982788428204  
RMSLE: 0.2154998534679604  
RMSLE: 0.20415655796958632

Feature importance:



# Chapter 8

## Python-Code

---



# Cab Fare Prediction

##### 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.

# loading the required libraries

```
import os
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import matplotlib.pyplot as plt
import scipy.stats as stats
from fancyimpute import KNN
import warnings
warnings.filterwarnings('ignore')
from geopy.distance import geodesic
from geopy.distance import great_circle
from scipy.stats import chi2_contingency
import statsmodels.api as sm
from statsmodels.formula.api import ols
from patsy import dmatrices
from statsmodels.stats.outliers_influence import variance_inflation_factor
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
from sklearn import metrics
from sklearn.linear_model import LinearRegression, Ridge, Lasso
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import RandomizedSearchCV
from sklearn.model_selection import cross_val_score
from sklearn.ensemble import RandomForestRegressor
from sklearn.tree import DecisionTreeRegressor
from xgboost import XGBRegressor
import xgboost as xgb
```

```
from sklearn.externals import joblib
```

```
# set the working directory
os.chdir('C:/Users/admin/Documents/Python Files')
os.getcwd()
```

The details of data attributes in the dataset are as follows:

- 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.

predictive modeling machine learning project can be broken down into below workflow:

1. Prepare Problem

a) Load libraries b) Load dataset

2. Summarize Data a) Descriptive statistics b) Data visualizations

3. Prepare Data a) Data Cleaning b) Feature Selection c) Data Transforms

4. Evaluate Algorithms a) Split-out validation dataset b) Test options and evaluation metric c) Spot Check Algorithms d) Compare Algorithms

5. Improve Accuracy a) Algorithm Tuning b) Ensembles

6. Finalize Model a) Predictions on validation dataset b) Create standalone model on entire training dataset c) Save model for later use

# Importing data

```
train = pd.read_csv('train_cab.csv', dtype={'fare_amount': np.float64}, na_values={'fare_amount': '430-'})
```

```
test = pd.read_csv('test.csv')
```

```
data=[train,test]
```

```
for i in data:
```

```
    i['pickup_datetime'] = pd.to_datetime(i['pickup_datetime'], errors='coerce')
```

```
train.head(5)
```

```
train.info()
```

```
test.head(5)
```

```
test.info()
```

```
test.describe()
```

```
train.describe()
```

## EDA

- we will convert passenger\_count into a categorical variable because passenger\_count is not a continuous variable.
- passenger\_count cannot take continuous values. and also they are limited in number if its a cab.

```
cat_var=['passenger_count']
```

```
num_var=['fare_amount', 'pickup_longitude', 'pickup_latitude', 'dropoff_longitude', 'dropoff_latitude']
```

## Graphical EDA - Data Visualization

# setting up the sns for plots

```
sns.set(style='darkgrid', palette='Set1')
```

Some histogram plots from seaborn library

```
plt.figure(figsize=(20,20))
plt.subplot(321)
_ = sns.distplot(train['fare_amount'],bins=50)
plt.subplot(322)
_ = sns.distplot(train['pickup_longitude'],bins=50)
plt.subplot(323)
_ = sns.distplot(train['pickup_latitude'],bins=50)
plt.subplot(324)
_ = sns.distplot(train['dropoff_longitude'],bins=50)
plt.subplot(325)
_ = sns.distplot(train['dropoff_latitude'],bins=50)
# plt.savefig('hist.png')
plt.show()
```

Some Bee Swarmplots

```
# plt.figure(figsize=(25,25))
# _ = sns.swarmplot(x='passenger_count',y='fare_amount',data=train)
# plt.title('Cab Fare w.r.t passenger_count')
```

- Jointplots for Bivariate Analysis.
- Here Scatter plot has regression line between 2 variables along with separate Bar plots of both variables.
- Also its annotated with pearson correlation coefficient and p value.

```
_ = sns.jointplot(x='fare_amount',y='pickup_longitude',data=train,kind = 'reg')
_.annotate(stats.pearsonr)
# plt.savefig('jointfplo.png')
plt.show()
```

```
_ = sns.jointplot(x='fare_amount',y='pickup_latitude',data=train,kind = 'reg')
_.annotate(stats.pearsonr)
# plt.savefig('jointfpla.png')
plt.show()
```

```
_ = sns.jointplot(x='fare_amount',y='dropoff_longitude',data=train,kind = 'reg')
_.annotate(stats.pearsonr)
# plt.savefig('jointfdlo.png')
plt.show()
```

```
_ = sns.jointplot(x='fare_amount',y='dropoff_latitude',data=train,kind = 'reg')
_.annotate(stats.pearsonr)
# plt.savefig('jointfdla.png')
plt.show()
```

Some Violinplots to see spread of variables

```
plt.figure(figsize=(20,20))
plt.subplot(321)
_ = sns.violinplot(y='fare_amount',data=train)
plt.subplot(322)
_ = sns.violinplot(y='pickup_longitude',data=train)
plt.subplot(323)
```



```

_=sns.violinplot(y='pickup_latitude',data=train)
plt.subplot(324)
_=sns.violinplot(y='dropoff_longitude',data=train)
plt.subplot(325)
_=sns.violinplot(y='dropoff_latitude',data=train)
plt.savefig('violin.png')
plt.show()

```

Pairplot for all numerical variables

```

_=sns.pairplot(data=train[num_var],kind='scatter',dropna=True)
_.fig.suptitle('Pairwise plot of all numerical variables')
# plt.savefig('Pairwise.png')
plt.show()

```

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

1.Fare amount has a negative value, which doesn't make sense. A price amount cannot be -ve and also cannot be 0. So we will remove these fields.

```

sum(train['fare_amount']<1)

train[train['fare_amount']<1]

train = train.drop(train[train['fare_amount']<1].index,axis=0)

#train.loc[train['fare_amount']<1,'fare_amount']=np.nan

```

2.Passenger\_count variable

```

for i in range(4,11):
    print('passenger_count above ' +str(i)+'={}'.format(sum(train['passenger_count']>i)))

```

so 20 observations of passenger\_count is consistently above from 6,7,8,9,10 passenger\_counts, let's check them.

```
train[train['passenger_count']>6]
```

Also we need to see if there are any passenger\_count<1

```
train[train['passenger_count']<1]
```

```
len(train[train['passenger_count']<1])
```

```
test['passenger_count'].unique()
```

- passenger\_count variable contains values which are equal to 0.
- And test data does not contain passenger\_count=0 . So if we feature engineer passenger\_count of train dataset then it will create a dummy variable for passenger\_count=0 which will be an extra feature compared to test dataset.
- So, we will remove those 0 values.
- Also, We will remove 20 observation which are above 6 value because a cab cannot hold these number of passengers.

```

train = train.drop(train[train['passenger_count']>6].index, axis=0)
train = train.drop(train[train['passenger_count']<1].index, axis=0)

```

```
# train.loc[train['passenger_count'] >6,'passenger_count'] = np.nan
```

```
#train.loc[train['passenger_count']>1,'passenger_count']=np.nan
```

```
sum(train['passenger_count']>6)
```

3.Latitudes range from -90 to 90.Longitudes range from -180 to 180.

Removing which does not satisfy these ranges

```
print('pickup_longitude above 180={}'.format(sum(train['pickup_longitude']>180)))
print('pickup_longitude below -180={}'.format(sum(train['pickup_longitude']<-180)))
print('pickup_latitude above 90={}'.format(sum(train['pickup_latitude']>90)))
print('pickup_latitude below -90={}'.format(sum(train['pickup_latitude']<-90)))
print('dropoff_longitude above 180={}'.format(sum(train['dropoff_longitude']>180)))
print('dropoff_longitude below -180={}'.format(sum(train['dropoff_longitude']<-180)))
print('dropoff_latitude below -90={}'.format(sum(train['dropoff_latitude']<-90)))
print('dropoff_latitude above 90={}'.format(sum(train['dropoff_latitude']>90)))
```

- There's only one outlier which is in variable pickup\_latitude. So we will remove it with nan.
- Also we will see if there are any values equal to 0.

```
for i in ['pickup_longitude','pickup_latitude','dropoff_longitude','dropoff_latitude']:
    print(i,'equal to 0={}'.format(sum(train[i]==0)))
```

there are values which are equal to 0. we will remove them.

```
train = train.drop(train[train['pickup_latitude']>90].index, axis=0)
for i in ['pickup_longitude','pickup_latitude','dropoff_longitude','dropoff_latitude']:
    train = train.drop(train[train[i]==0].index, axis=0)
```

```
# for i in ['pickup_longitude','pickup_latitude','dropoff_longitude','dropoff_latitude']:
#     train.loc[train[i]==0,i] = np.nan
# train.loc[train['pickup_latitude']>90,'pickup_latitude'] = np.nan
```

```
train.shape
```

So, we lossed 16067-15661=406 observations because of non-sensical values.

```
df=train.copy()
# train=df.copy()
```

## Missing Value Analysis

```
#Create dataframe with missing percentage
missing_val = pd.DataFrame(train.isnull().sum())
#Reset index
missing_val = missing_val.reset_index()
missing_val
```

- As we can see there are some missing values in the data.
- Also pickup\_datetime variable has 1 missing value.
- We will impute missing values for fare\_amount,passenger\_count variables except pickup\_datetime.
- And we will drop that 1 row which has missing value in pickup\_datetime.

```
#Rename variable
missing_val = missing_val.rename(columns = {'index': 'Variables', 0: 'Missing_percentage'})
missing_val
```

```
#Calculate percentage
missing_val['Missing_percentage'] = (missing_val['Missing_percentage']/len(train))*100
#descending order
missing_val = missing_val.sort_values('Missing_percentage', ascending = False).reset_index(drop = True)
missing_val
```

1.For Passenger\_count:

- Actual value = 1
- Mode = 1
- KNN = 2

```
# Choosing a random values to replace it as NA
train['passenger_count'].loc[1000]
```

```
# Replacing 1.0 with NA
train['passenger_count'].loc[1000] = np.nan
train['passenger_count'].loc[1000]
```

```
# Impute with mode
train['passenger_count'].fillna(train['passenger_count'].mode()[0]).loc[1000]
```

We can't use mode method because data will be more biased towards passenger\_count=1

2.For fare\_amount:

- Actual value = 7.0,
- Mean = 15.117,
- Median = 8.5,
- KNN = 7.369801

```
# for i in ['fare_amount','pickup_longitude','pickup_latitude','dropoff_longitude','dropoff_latitude']:
#   # Choosing a random values to replace it as NA
#   a=train[i].loc[1000]
#   print(i,'at loc-1000:{}'.format(a))
#   # Replacing 1.0 with NA
#   train[i].loc[1000] = np.nan
#   print('Value after replacing with nan:{}'.format(train[i].loc[1000]))
#   # Impute with mean
#   print('Value if imputed with mean:{}'.format(train[i].fillna(train[i].mean()).loc[1000]))
#   # Impute with median
#   print('Value if imputed with median:{}'.format(train[i].fillna(train[i].median()).loc[1000]))
```

```
# Choosing a random values to replace it as NA
a=train['fare_amount'].loc[1000]
print('fare_amountatloc-1000:{}'.format(a))
# Replacing 1.0 with NA
train['fare_amount'].loc[1000] = np.nan
print('Value after replacing with nan:{}'.format(train['fare_amount'].loc[1000]))
# Impute with mean
print('Value if imputed with mean:{}'.format(train['fare_amount'].fillna(train['fare_amount'].mean()).loc[1000]))
# Impute with median
print('Value if imputed with median:{}'.format(train['fare_amount'].fillna(train['fare_amount'].median()).loc[1000]))
```

```
train.std()
```

```
columns=['fare_amount', 'pickup_longitude', 'pickup_latitude','dropoff_longitude', 'dropoff_latitude', 'passenger_count']
```

we will separate pickup\_datetime into a different dataframe and then merge with train in feature engineering step.

```
pickup_datetime=pd.DataFrame(train['pickup_datetime'])
```

```
# Imputing with missing values using KNN
```

```
# Use 19 nearest rows which have a feature to fill in each row's missing features
```

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

```
train.std()
```

```
train.loc[1000]
```

```
train['passenger_count'].head()
```

```
train['passenger_count']=train['passenger_count'].astype('int')
```

```
train.std()
```

```
train['passenger_count'].unique()
```

```
train['passenger_count']=train['passenger_count'].round().astype('object').astype('category',ordered=True)
```

```
train['passenger_count'].unique()
```

```
train.loc[1000]
```

- Now about missing value in pickup\_datetime

```
pickup_datetime.head()
```

```
#Create dataframe with missing percentage
```

```
missing_val = pd.DataFrame(pickup_datetime.isnull().sum())
```

```
#Reset index
```

```
missing_val = missing_val.reset_index()
```

```
missing_val
```

```
pickup_datetime.shape
```

```
train.shape
```

- We will drop 1 row which has missing value for pickup\_datetime variable after feature engineering step because if we drop now, pickup\_datetime dataframe will have 16040 rows and our train has 1641 rows, then if we merge these 2 dataframes then pickup\_datetime variable will gain 1 missing value.

- And if we merge and then drop now then we would require to split again before outlier analysis and then merge again in feature engineering step.

- So, instead of doing the work 2 times we will drop 1 time i.e. after feature engineering process.

```
# df1 = train.copy()
```

```
train=df1.copy()
```

```
train['passenger_count'].describe()
```

```
train.describe()
```

### ## Outlier Analysis using Boxplot

- We Will do Outlier Analysis only on Fare\_amount just for now and we will do outlier analysis after feature engineering latitudes and longitudes.

- Univariate Boxplots: Boxplots for all Numerical Variables including target variable.

```
plt.figure(figsize=(20,5))
plt.xlim(0,100)
sns.boxplot(x=train['fare_amount'],data=train,orient='h')
plt.title('Boxplot of fare_amount')
# plt.savefig('bp of fare_amount.png')
plt.show()

# sum(train['fare_amount']<22.5)/len(train['fare_amount'])*100
```

- Bivariate Boxplots: Boxplot for Numerical Variable Vs Categorical Variable.

```
plt.figure(figsize=(20,10))
plt.xlim(0,100)
_ = sns.boxplot(x=train['fare_amount'],y=train['passenger_count'],data=train,orient='h')
plt.title('Boxplot of fare_amount w.r.t passenger_count')
# plt.savefig('Boxplot of fare_amount w.r.t passenger_count.png')
plt.show()
```

```
train.describe()
```

```
train['passenger_count'].describe()
```

### ## Outlier Treatment

- As we can see from the above Boxplots there are outliers in the train dataset.  
- Reconsider pickup\_longitude, etc.

```
def outlier_treatment(col):
    ''' calculating outlier indices and replacing them with NA '''
    #Extract quartiles
    q75, q25 = np.percentile(train[col], [75, 25])
    print(q75, q25)
    #Calculate IQR
    iqr = q75 - q25
    #Calculate inner and outer fence
    minimum = q25 - (iqr*1.5)
    maximum = q75 + (iqr*1.5)
    print(minimum, maximum)
    #Replace with NA
    train.loc[train[col] < minimum, col] = np.nan
    train.loc[train[col] > maximum, col] = np.nan
```

```
# for i in num_var:
    outlier_treatment('fare_amount')
# outlier_treatment('pickup_longitude')
# outlier_treatment('pickup_latitude')
# outlier_treatment('dropoff_longitude')
# outlier_treatment('dropoff_latitude')
```

```
pd.DataFrame(train.isnull().sum())
```

```
train.std()
```

```
#Imputing with missing values using KNN
```

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

```
train.std()
```

```
train['passenger_count'].describe()
```

```
train['passenger_count']=train['passenger_count'].astype('int').round().astype('object').astype('category')
```

```
train.describe()
```

```
train.head()
```

```
df2 = train.copy()
```

```
# train=df2.copy()
```

```
train.shape
```

```
## Feature Engineering
```

```
#### 1.Feature Engineering for timestamp variable
```

- we will derive new features from pickup\_datetime variable
- new features will be year,month,day\_of\_week,hour

```
# we will Join 2 Dataframes pickup_datetime and train
```

```
train = pd.merge(pickup_datetime,train,right_index=True,left_index=True)
```

```
train.head()
```

```
train.shape
```

```
train=train.reset_index(drop=True)
```

As we discussed in Missing value imputation step about dropping missing value, we will do it now.

```
pd.DataFrame(train.isna().sum())
```

```
train=train.dropna()
```

```
data = [train,test]
```

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

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

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

```
# train_nodummies=train.copy()
```

```
# train=train_nodummies.copy()
```

```
plt.figure(figsize=(20,10))
```

```
sns.countplot(train['year'])
```

```
# plt.savefig('year.png')

plt.figure(figsize=(20,10))
sns.countplot(train['month'])
# plt.savefig('month.png')

plt.figure(figsize=(20,10))
sns.countplot(train['day_of_week'])
# plt.savefig('day_of_week.png')

plt.figure(figsize=(20,10))
sns.countplot(train['hour'])
# plt.savefig('hour.png')
```

Now we will use month, day\_of\_week, hour to derive new features like sessions in a day, seasons in a year, week:weekend/weekday

```
def f(x):
    """for sessions in a day using hour column"""
    if (x >= 5) and (x <= 11):
        return 'morning'
    elif (x >= 12) and (x <= 16):
        return 'afternoon'
    elif (x >= 17) and (x <= 20):
        return 'evening'
    elif (x >= 21) and (x <= 23):
        return 'night_PM'
    elif (x >= 0) and (x <= 4):
        return 'night_AM'

def g(x):
    """for seasons in a year using month column"""
    if (x >= 3) and (x <= 5):
        return 'spring'
    elif (x >= 6) and (x <= 8):
        return 'summer'
    elif (x >= 9) and (x <= 11):
        return 'fall'
    elif (x >= 12) and (x <= 2):
        return 'winter'

def h(x):
    """for week:weekday/weekend in a day_of_week column"""
    if (x >= 0) and (x <= 4):
        return 'weekday'
    elif (x >= 5) and (x <= 6):
        return 'weekend'

train['session'] = train['hour'].apply(f)
test['session'] = test['hour'].apply(f)
# train_nodummies['session'] = train_nodummies['hour'].apply(f)

train['seasons'] = train['month'].apply(g)
test['seasons'] = test['month'].apply(g)
# train['seasons'] = test['month'].apply(g)
```

```
train['week'] = train['day_of_week'].apply(h)
test['week'] = test['day_of_week'].apply(h)
```

```
train.shape
```

```
test.shape
```

```
##### 2.Feature Engineering for passenger_count variable
```

- Because models in scikit learn require numerical input,if dataset contains categorical variables then we have to encode them.
- We will use one hot encoding technique for passenger\_count variable.

```
train['passenger_count'].describe()
```

```
#Creating dummies for each variable in passenger_count and merging dummies dataframe to both train and test dataframe
```

```
temp=pd.get_dummies(train['passenger_count'],prefix='passenger_count')
```

```
train = train.join(temp)
```

```
temp=pd.get_dummies(test['passenger_count'],prefix='passenger_count')
```

```
test = test.join(temp)
```

```
temp=pd.get_dummies(train['seasons'],prefix='season')
```

```
train = train.join(temp)
```

```
temp=pd.get_dummies(test['seasons'],prefix='season')
```

```
test = test.join(temp)
```

```
temp = pd.get_dummies(train['week'], prefix = 'week')
```

```
train = train.join(temp)
```

```
temp = pd.get_dummies(test['week'], prefix = 'week')
```

```
test = test.join(temp)
```

```
temp=pd.get_dummies(train['session'],prefix='session')
```

```
train = train.join(temp)
```

```
temp=pd.get_dummies(test['session'],prefix='session')
```

```
test = test.join(temp)
```

```
temp = pd.get_dummies(train['year'], prefix = 'year')
```

```
train = train.join(temp)
```

```
temp = pd.get_dummies(test['year'], prefix = 'year')
```

```
test = test.join(temp)
```

```
train.head()
```

```
test.head()
```

we will drop one column from each one-hot-encoded variables

```
train.columns
```

```
train=train.drop(['passenger_count_1','season_fall','week_weekday','session_afternoon','year_2009'],axis=1)
```

```
test=test.drop(['passenger_count_1','season_fall','week_weekday','session_afternoon','year_2009'],axis=1)
```

```
##### 3.Feature Engineering for latitude and longitude variable
```

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

```
# train.sort_values('pickup_datetime')
```



```

# def haversine(coord1, coord2):
#     """Calculate distance the cab travelled from pickup and dropoff location using the Haversine Formula"""
#     data = [train, test]
#     for i in data:
#         lon1, lat1 = coord1
#         lon2, lat2 = coord2
#         R=6371000 #radius of Earth in meters
#         phi_1 = np.radians(i[lat1])
#         phi_2 = np.radians(i[lat2])
#         delta_phi = np.radians(i[lat2] - i[lat1])
#         delta_lambda = np.radians(i[lon2] - i[lon1])
#         a = np.sin(delta_phi/2.0)**2 + np.cos(phi_1) * np.cos(phi_2) * np.sin(delta_lambda/2.0)**2
#         c = 2 * np.arctan2(np.sqrt(a), np.sqrt(1 - a))
#         meters = R * c # output distance in meters
#         km = meters/1000.0 # output distance in kilometers
#         miles = round(km, 3)/1.609344
#         i['distance'] = miles
#     print(f"Distance: {miles}miles")
#     return miles

# haversine(['pickup_longitude', 'pickup_latitude'], ['dropoff_longitude', 'dropoff_latitude'])

# Calculate distance the cab travelled from pickup and dropoff location using great_circle from geopy library
data = [train, test]
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)

train.head()

test.head()

As Vincenty is more accurate than haversine. Also vincenty is preferred for short distances. Therefore we will drop
great_circle. we will drop them together with other variables which were used to feature engineer.

pd.DataFrame(train.isna().sum())

pd.DataFrame(test.isna().sum())

#### We will remove the variables which were used to feature engineer new variables

# train_nodummies=train_nodummies.drop(['pickup_datetime', 'pickup_longitude', 'pickup_latitude',
#     'dropoff_longitude', 'dropoff_latitude', 'great_circle'], axis = 1)
# test_nodummies=test.drop(['pickup_datetime', 'pickup_longitude', 'pickup_latitude',
#     'dropoff_longitude', 'dropoff_latitude', 'passenger_count_1', 'passenger_count_2', 'passenger_count_3',
#     'passenger_count_4', 'passenger_count_5', 'passenger_count_6',
#     'season_fall', 'season_spring', 'season_summer', 'season_winter',
#     'week_weekday', 'week_weekend', 'session_afternoon', 'session_evening',
#     'session_morning', 'session_night (AM)', 'session_night (PM)',
#     'year_2009', 'year_2010', 'year_2011', 'year_2012', 'year_2013',
#     'year_2014', 'year_2015', 'great_circle'], axis = 1)

train=train.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)
test=test.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)

train.shape, test.shape

# test_nodummies.columns

# train_nodummies.columns

train.columns

test.columns

train.head()

test.head()

plt.figure(figsize=(20,5))
sns.boxplot(x=train['geodesic'], data=train, orient='h')
plt.title('Boxplot of geodesic ')
# plt.savefig('bp geodesic.png')
plt.show()

plt.figure(figsize=(20,5))
plt.xlim(0,100)
sns.boxplot(x=train['geodesic'], data=train, orient='h')
plt.title('Boxplot of geodesic ')
# plt.savefig('bp geodesic.png')
plt.show()

outlier_treatment('geodesic')

pd.DataFrame(train.isnull().sum())

#Imputing with missing values using KNN
train = pd.DataFrame(KNN(k = 3).fit_transform(train), columns = train.columns, index=train.index)

## Feature Selection
1. Correlation Analysis

Statistically correlated: features move together directionally.
Linear models assume feature independence.
And if features are correlated that could introduce bias into our models.

cat_var=['passenger_count_2',
'passenger_count_3', 'passenger_count_4', 'passenger_count_5',
'passenger_count_6', 'season_spring', 'season_summer',
'session_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']
num_var=['fare_amount', 'geodesic']

```

```
train[cat_var]=train[cat_var].apply(lambda x:x.astype('category'))
test[cat_var]=test[cat_var].apply(lambda x:x.astype('category'))
```

- We will plot a Heatmap of correlation whereas, correlation measures how strongly 2 quantities are related to each other.

```
# heatmap using correlation matrix
plt.figure(figsize=(15,15))
_ = sns.heatmap(train[num_var].corr(), square=True, cmap='RdYlGn',linewidths=0.5,linecolor='w',annot=True)
plt.title('Correlation matrix ')
# plt.savefig('correlation.png')
plt.show()
```

As we can see from above correlation plot fare\_amount and geodesic is correlated to each other.

- Jointplots for Bivariate Analysis.
- Here Scatter plot has regression line between 2 variables along with separate Bar plots of both variables.
- Also its annotated with pearson correlation coefficient and p value.

```
_ = sns.jointplot(x='fare_amount',y='geodesic',data=train,kind = 'reg')
_.annotate(stats.pearsonr)
# plt.savefig('jointct.png')
plt.show()
```

### ### Chi-square test of Independence for Categorical Variables/Features

- Hypothesis testing :
  - Null Hypothesis: 2 variables are independent.
  - Alternate Hypothesis: 2 variables are not independent.
- 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.
- There should be no dependencies between Independent variables.
- So we will remove that variable whose p-value with other variable is low than 0.05.
- And we will keep that variable whose p-value with other variable is high than 0.05

```
#loop for chi square values
for i in cat_var:
    for j in cat_var:
        if(i != j):
            chi2, p, dof, ex = chi2_contingency(pd.crosstab(train[i], train[j]))
            if(p < 0.05):
                print(i,"and",j,"are dependent on each other with",p,'----Remove')
            else:
                print(i,"and",j,"are independent on each other with",p,'----Keep')
```

### ## Analysis of Variance(Anova) Test

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

```
train.columns
```

```
#ANOVA
_1)+C(passenger_count_2)+C(passenger_count_3)+C(passenger_count_4)+C(passenger_count_5)+C(passenger_count_6)
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(sea
season_spring)+C(season_summer)+C(season_winter)+C(week_weekend)+C(session_night_AM)+C(session_night_PM)+C(s
ession_evening)+C(session_morning)+C(year_2010)+C(year_2011)+C(year_2012)+C(year_2013)+C(year_2014)+C(year_20
15)',data=train).fit()

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

Every variable has p-value less than 0.05 therefore we reject the null hypothesis.

### ## Multicollinearity Test

- 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.

```
#_1+passenger_count_2+passenger_count_3+passenger_count_4+passenger_count_5+passenger_count_6
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+sessio
n_morning+year_2010+year_2011+year_2012+year_2013+year_2014+year_2015',train, 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
```

So we have no or very low multicollinearity

### ## Feature Scaling Check with or without normalization of standard scalar

```
train[num_var].var()

sns.distplot(train['geodesic'],bins=50)
# plt.savefig('distplot.png')

plt.figure()
stats.probplot(train['geodesic'], dist='norm', fit=True,plot=plt)
# plt.savefig('qq prob plot.png')
```

### #Normalization

```
train['geodesic'] = (train['geodesic'] - min(train['geodesic']))/(max(train['geodesic']) - min(train['geodesic']))
test['geodesic'] = (test['geodesic'] - min(test['geodesic']))/(max(test['geodesic']) - min(test['geodesic']))
```

```
train['geodesic'].var()
```

```
sns.distplot(train['geodesic'],bins=50)
plt.savefig('distplot.png')
```

```

plt.figure()
stats.probplot(train['geodesic'], dist='norm', fit=True, plot=plt)
# plt.savefig('qq prob plot.png')

train.columns

#df4=train.copy()
train=df4.copy()
# f4=test.copy()
test=f4.copy()

train=train.drop(['passenger_count_2'],axis=1)
test=test.drop(['passenger_count_2'],axis=1)

train.columns

## Splitting train into train and validation subsets
- X_train y_train--are trainsubset
- X_test y_test--are validationsubset

X = train.drop('fare_amount',axis=1).values
y = train['fare_amount'].values
X_train,X_test,y_train,y_test=train_test_split(X,y, test_size = 0.25, random_state=42)
print(train.shape, X_train.shape, X_test.shape,y_train.shape,y_test.shape)

def rmsle(y,y_):
    log1=np.nan_to_num(np.array([np.log(v+1) for v in y]))
    log2=np.nan_to_num(np.array([np.log(v+1) for v in y_]))
    calc = (log1 - log2) ** 2
    return np.sqrt(np.mean(calc))

def scores(y, y_):
    print('r square ', metrics.r2_score(y, y_))
    print('Adjusted r square:{}'.format(1 - (1-metrics.r2_score(y, y_))*(len(y)-1)/(len(y)-X_train.shape[1]-1)))
    print('MAPE:{}'.format(np.mean(np.abs((y - y_) / y))*100))
    print('MSE:', metrics.mean_squared_error(y, y_))
    print('RMSE:', np.sqrt(metrics.mean_squared_error(y, y_)))

def test_scores(model):
    print('<<<----- Training Data Score ----->')
    print()
    #Predicting result on Training data
    y_pred = model.predict(X_train)
    scores(y_train,y_pred)
    print('RMSLE:',rmsle(y_train,y_pred))
    print()
    print('<<<----- Test Data Score ----->')
    print()
    # Evaluating on Test Set
    y_pred = model.predict(X_test)
    scores(y_test,y_pred)
    print('RMSLE:',rmsle(y_test,y_pred))

## Multiple Linear Regression

# Setup the parameters and distributions to sample from: param_dist
param_dist = {'copy_X':[True, False],

```

```
'fit_intercept':[True,False]]
# Instantiate a Decision reg classifier: reg
reg = LinearRegression()

# Instantiate the gridSearchCV object: reg_cv
reg_cv = GridSearchCV(reg, param_dist, cv=5,scoring='r2')

# Fit it to the data
reg_cv.fit(X,y)

# Print the tuned parameters and score
print("Tuned Decision reg Parameters: {}".format(reg_cv.best_params_))
print("Best score is {}".format(reg_cv.best_score_))

# Create the regressor: reg_all
reg_all = LinearRegression(copy_X= True, fit_intercept=True)

#Fit the regressor to the training data
reg_all.fit(X_train,y_train)

#Predict on the test data: y_pred
y_pred= reg_all.predict(X_test)

# Compute and print R^2 and RMSE
print("R^2: {}".format(reg_all.score(X_test, y_test)))
rmse = np.sqrt(mean_squared_error(y_test,y_pred))
print("Root Mean Squared Error: {}".format(rmse))
test_scores(reg_all)

# Compute and print the coefficients
reg_coef = reg_all.coef_
print(reg_coef)

# Plot the coefficients
plt.figure(figsize=(15,5))
plt.plot(range(len(test.columns)), reg_coef)
plt.xticks(range(len(test.columns)), test.columns.values, rotation=60)
plt.margins(0.02)
plt.savefig('linear coefficients')
plt.show()

from sklearn.model_selection import cross_val_score
# Create a linear regression object: reg
reg = LinearRegression()

# Compute 5-fold cross-validation scores: cv_scores
cv_scores = cross_val_score(reg,X,y,cv=5,scoring='neg_mean_squared_error')

# Print the 5-fold cross-validation scores
print(cv_scores)

print("Average 5-Fold CV Score: {}".format(np.mean(cv_scores)))

## Ridge Regression
```

```
# Setup the parameters and distributions to sample from: param_dist
param_dist = {'alpha':np.logspace(-4, 0, 50),
              'normalize':[True,False],
              'max_iter':range(500,5000,500)}
# Instantiate a Decision ridge classifier: ridge
ridge = Ridge()

# Instantiate the gridSearchCV object: ridge_cv
ridge_cv = GridSearchCV(ridge, param_dist, cv=5,scoring='r2')

# Fit it to the data
ridge_cv.fit(X,y)

# Print the tuned parameters and score
print("Tuned Decision ridge Parameters: {}".format(ridge_cv.best_params_))
print("Best score is {}".format(ridge_cv.best_score_))

# Instantiate a ridge regressor: ridge
ridge = Ridge(alpha=0.0005428675439323859, normalize=True,max_iter = 500)

# Fit the regressor to the data
ridge.fit(X_train,y_train)

# Compute and print the coefficients
ridge_coef = ridge.coef_
print(ridge_coef)

# Plot the coefficients
plt.figure(figsize=(15,5))
plt.plot(range(len(test.columns)), ridge_coef)
plt.xticks(range(len(test.columns)), test.columns.values, rotation=60)
plt.margins(0.02)
# plt.savefig('ridge coefficients')
plt.show()
test_scores(ridge)
```

lasso can be used feature selection

### ## Lasso Regression

```
# Setup the parameters and distributions to sample from: param_dist
param_dist = {'alpha':np.logspace(-4, 0, 50),
              'normalize':[True,False],
              'max_iter':range(500,5000,500)}
# Instantiate a Decision lasso classifier: lasso
lasso = Lasso()

# Instantiate the gridSearchCV object: lasso_cv
lasso_cv = GridSearchCV(lasso, param_dist, cv=5,scoring='r2')

# Fit it to the data
lasso_cv.fit(X,y)

# Print the tuned parameters and score
print("Tuned Decision lasso Parameters: {}".format(lasso_cv.best_params_))
```

```
print("Best score is {}".format(lasso_cv.best_score_))

# Instantiate a lasso regressor: lasso
lasso = Lasso(alpha=0.00021209508879201905, normalize=False,max_iter = 500)

# Fit the regressor to the data
lasso.fit(X,y)

# Compute and print the coefficients
lasso_coef = lasso.coef_
print(lasso_coef)

# Plot the coefficients
plt.figure(figsize=(15,5))
plt.ylim(-1,10)
plt.plot(range(len(test.columns)), lasso_coef)
plt.xticks(range(len(test.columns)), test.columns.values, rotation=60)
plt.margins(0.02)
plt.savefig('lasso coefficients')
plt.show()
test_scores(lasso)

## Decision Tree Regression

train.info()

# Setup the parameters and distributions to sample from: param_dist
param_dist = {'max_depth': range(2,16,2),
              'min_samples_split': range(2,16,2)}

# Instantiate a Decision Tree classifier: tree
tree = DecisionTreeRegressor()

# Instantiate the gridSearchCV object: tree_cv
tree_cv = GridSearchCV(tree, param_dist, cv=5)

# Fit it to the data
tree_cv.fit(X, y)

# Print the tuned parameters and score
print("Tuned Decision Tree Parameters: {}".format(tree_cv.best_params_))
print("Best score is {}".format(tree_cv.best_score_))

# Instantiate a tree regressor: tree
tree = DecisionTreeRegressor(max_depth= 6, min_samples_split=2)

# Fit the regressor to the data
tree.fit(X_train,y_train)

# Compute and print the coefficients
tree_features = tree.feature_importances_
print(tree_features)

# Sort test importances in descending order
indices = np.argsort(tree_features)[::-1]
```



```
# Rearrange test names so they match the sorted test importances
names = [test.columns[i] for i in indices]

# Creating plot
fig = plt.figure(figsize=(20,10))
plt.title("test Importance")

# Add horizontal bars
plt.barh(range(pd.DataFrame(X_train).shape[1]),tree_features[indices],align = 'center')
plt.yticks(range(pd.DataFrame(X_train).shape[1]), names)
plt.savefig('tree test importance')
plt.show()
# Make predictions and cal error
test_scores(tree)

## Random Forest Regression

# Create the random grid
random_grid = {'n_estimators': range(100,500,100),
               'max_depth': range(5,20,1),
               'min_samples_leaf':range(2,5,1),
               'max_features':['auto','sqrt','log2'],
               'bootstrap': [True, False],
               'min_samples_split': range(2,5,1)}
# Instantiate a Decision Forest classifier: Forest
Forest = RandomForestRegressor()

# Instantiate the gridSearchCV object: Forest_cv
Forest_cv = RandomizedSearchCV(Forest, random_grid, cv=5)

# Fit it to the data
Forest_cv.fit(X, y)

# Print the tuned parameters and score
print("Tuned Random Forest Parameters: {}".format(Forest_cv.best_params_))
print("Best score is {}".format(Forest_cv.best_score_))

# Instantiate a Forest regressor: Forest
Forest = RandomForestRegressor(n_estimators=100, min_samples_split= 2, min_samples_leaf=4, max_features='auto',
max_depth=9, bootstrap=True)

#Fit the regressor to the data
Forest.fit(X_train,y_train)

# Compute and print the coefficients
Forest_features = Forest.feature_importances_
print(Forest_features)

# Sort feature importances in descending order
indices = np.argsort(Forest_features)[::-1]

# Rearrange feature names so they match the sorted feature importances
names = [test.columns[i] for i in indices]
```

```
# Creating plot
fig = plt.figure(figsize=(20,10))
plt.title("Feature Importance")

# Add horizontal bars
plt.barh(range(pd.DataFrame(X_train).shape[1]),Forest_features[indices],align = 'center')
plt.yticks(range(pd.DataFrame(X_train).shape[1]), names)
plt.savefig('Random forest feature importance')
plt.show()# Make predictions
test_scores(Forest)

from sklearn.model_selection import cross_val_score
#Create a random forest regression object: Forest
Forest = RandomForestRegressor(n_estimators=400, min_samples_split= 2, min_samples_leaf=4, max_features='auto',
max_depth=12, bootstrap=True)

# Compute 5-fold cross-validation scores: cv_scores
cv_scores = cross_val_score(Forest,X,y,cv=5,scoring='neg_mean_squared_error')

# Print the 5-fold cross-validation scores
print(cv_scores)

print("Average 5-Fold CV Score: {}".format(np.mean(cv_scores)))

## Improving accuracy using XGBOOST
- Improve Accuracy a) Algorithm Tuning b) Ensembles

data_dmatrix = xgb.DMatrix(data=X,label=y)
dtrain = xgb.DMatrix(X_train, label=y_train)
dtest = xgb.DMatrix(X_test)

dtrain,dtest,data_dmatrix

params={"objective":"reg:linear",'colsample_bytree':0.3,'learning_rate':0.1,
        'max_depth': 5, 'alpha': 10}

cv_results = xgb.cv(dtrain=data_dmatrix, params=params, nfold=5,
                    num_boost_round=50,early_stopping_rounds=10,metrics="rmse", as_pandas=True, seed=123)
cv_results.head()

# the final boosting round metric
print((cv_results["test-rmse-mean"]).tail(1))

Xgb = XGBRegressor()
Xgb.fit(X_train,y_train)
# pred_xgb = model_xgb.predict(X_test)
test_scores(Xgb)

# Create the random grid
para = {'n_estimators': range(100,500,100),
        'max_depth': range(3,10,1),
        'reg_alpha':np.logspace(-4, 0, 50),
        'subsample': np.arange(0.1,1,0.2),
        'colsample_bytree': np.arange(0.1,1,0.2),
        'colsample_bylevel': np.arange(0.1,1,0.2),
```

```

'colsample_bynode': np.arange(0.1,1,0.2),
'learning_rate': np.arange(.05, 1, .05)}
# Instantiate a Decision Forest classifier: Forest
Xgb = XGBRegressor()

# Instantiate the gridSearchCV object: Forest_cv
xgb_cv = RandomizedSearchCV(Xgb, para, cv=5)

# Fit it to the data
xgb_cv.fit(X, y)

# Print the tuned parameters and score
print("Tuned Xgboost Parameters: {}".format(xgb_cv.best_params_))
print("Best score is {}".format(xgb_cv.best_score_))

# Instantiate a xgb regressor: xgb
Xgb = XGBRegressor(subsample= 0.1, reg_alpha= 0.08685113737513521, n_estimators= 200, max_depth= 3,
learning_rate=0.05, colsample_bytree= 0.7000000000000001, colsample_bynode=0.7000000000000001,
colsample_bylevel=0.9000000000000001)

# Fit the regressor to the data
Xgb.fit(X_train,y_train)

# Compute and print the coefficients
xgb_features = Xgb.feature_importances_
print(xgb_features)

# Sort feature importances in descending order
indices = np.argsort(xgb_features)[::-1]

# Rearrange feature names so they match the sorted feature importances
names = [test.columns[i] for i in indices]

# Creating plot
fig = plt.figure(figsize=(20,10))
plt.title("Feature Importance")

# Add horizontal bars
plt.barh(range(pd.DataFrame(X_train).shape[1]),xgb_features[indices],align = 'center')
plt.yticks(range(pd.DataFrame(X_train).shape[1]), names)
plt.savefig(' xgb feature importance')
plt.show()# Make predictions
test_scores(Xgb)

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

def rmsle(y,y_):
    log1=np.nan_to_num(np.array([np.log(v+1)for viny]))
    log2 = np.nan_to_num(np.array([np.log(v + 1) for v in y_]))
    calc = (log1 - log2) ** 2
    return np.sqrt(np.mean(calc))

def score(y, y_):

```

```

print('r square ', metrics.r2_score(y, y_))
print('Adjusted r square:{}'.format(1 - (1 - metrics.r2_score(y, y_)) * (len(y) - 1) / (len(y) - X_train.shape[1] - 1)))
print('MAPE:{}'.format(np.mean(np.abs((y - y_) / y)) * 100))
print('MSE:', metrics.mean_squared_error(y, y_))
print('RMSE:', np.sqrt(metrics.mean_squared_error(y, y_)))
print('RMSLE:', rmsle(y_test, y_pred))
def scores(model):
    print('<<<----- Training Data Score ----->')
    print()
    #Predicting result on Training data
    y_pred = model.predict(X)
    score(y, y_pred)
    print('RMSLE:', rmsle(y, y_pred))

test.columns

train.columns

train.shape

test.shape

a=pd.read_csv('test.csv')

test_pickup_datetime=a['pickup_datetime']

# Instantiate a xgb regressor: xgb
Xgb = XGBRegressor(subsample= 0.1, reg_alpha= 0.08685113737513521, n_estimators= 200, max_depth= 3,
learning_rate=0.05, colsample_bytree= 0.7000000000000001, colsample_bynode=0.7000000000000001,
colsample_bylevel=0.9000000000000001)

#Fit the regressor to the data
Xgb.fit(X,y)

# Compute and print the coefficients
xgb_features = Xgb.feature_importances_
print(xgb_features)

# Sort feature importances in descending order
indices = np.argsort(xgb_features)[::-1]

# Rearrange feature names so they match the sorted feature importances
names = [test.columns[i] for i in indices]

# Creating plot
fig = plt.figure(figsize=(20,10))
plt.title("Feature Importance")

# Add horizontal bars
plt.barh(range(pd.DataFrame(X_train).shape[1]),xgb_features[indices],align = 'center')
plt.yticks(range(pd.DataFrame(X_train).shape[1]), names)
plt.savefig(' xgb1 feature importance')
plt.show()
scores(Xgb)

```

```
# Predictions
```

```
pred = Xgb.predict(test.values)
```

```
pred_results_wrt_date = pd.DataFrame({"pickup_datetime":test_pickup_datetime,"fare_amount":pred})
```

```
pred_results_wrt_date.to_csv("predictions_xgboost.csv",index=False)
```

```
pred_results_wrt_date
```

```
# Save the model as a pickle in a file
```

```
joblib.dump(Xgb, 'cab_fare_xgboost_model.pkl')
```

```
# # Load the model from the file
```

```
# Xgb_from_joblib = joblib.load('cab_fare_xgboost_model.pkl')
```