Cab Fare Prediction

Submitted by: -
Ganesh Karthik Kommineni

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

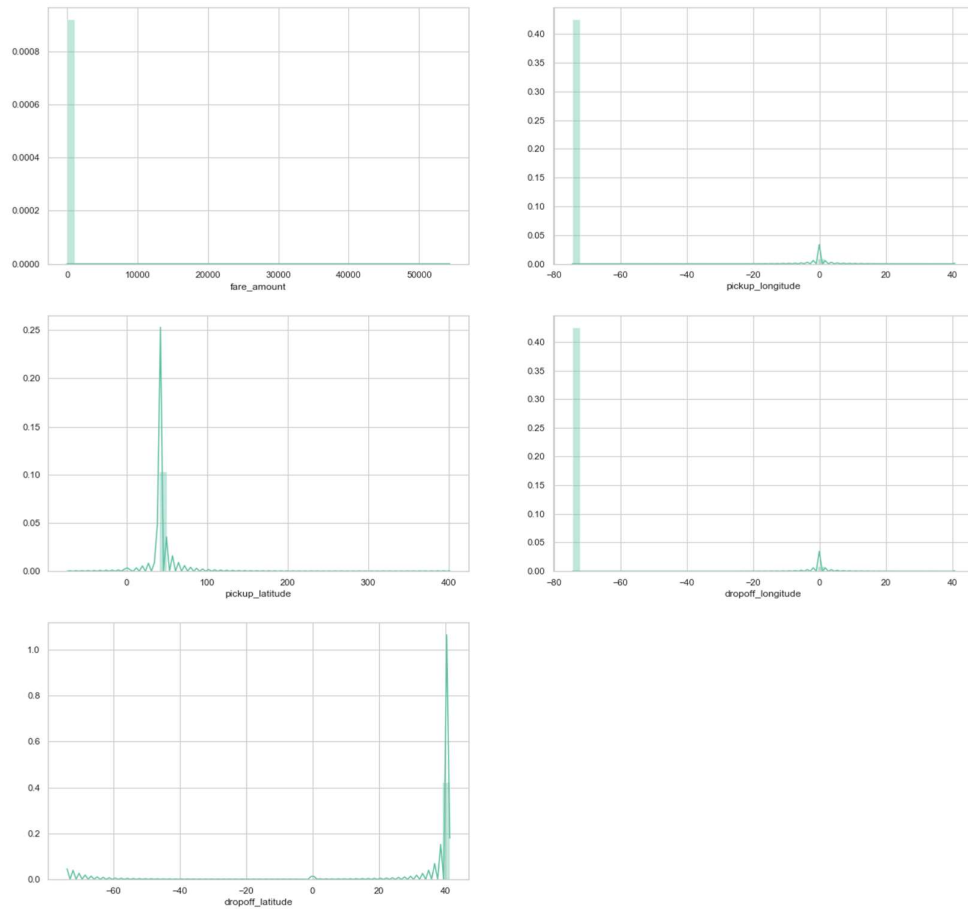
2.1 Data Pre-Processing

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.

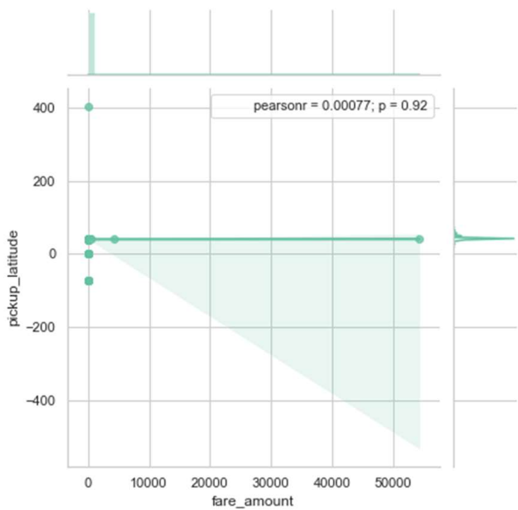
Data Visualization for better understanding:-

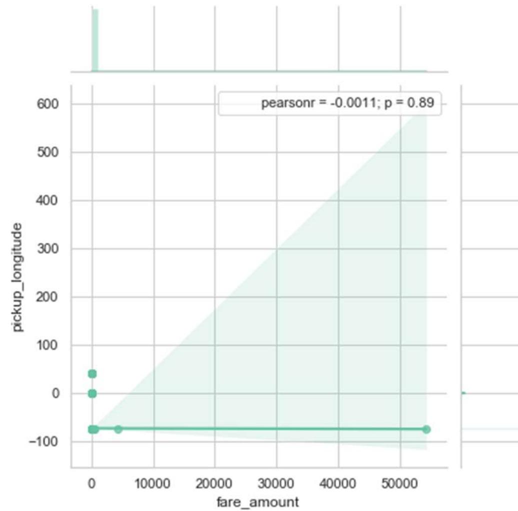
Some Histogram plots from seaborn library for each individual variable created using distplot() method.

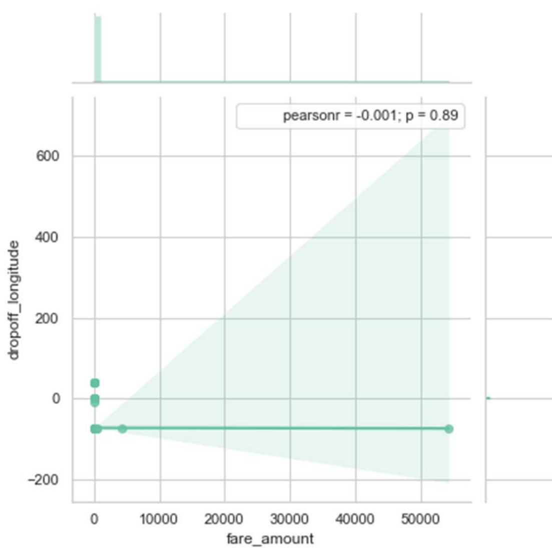


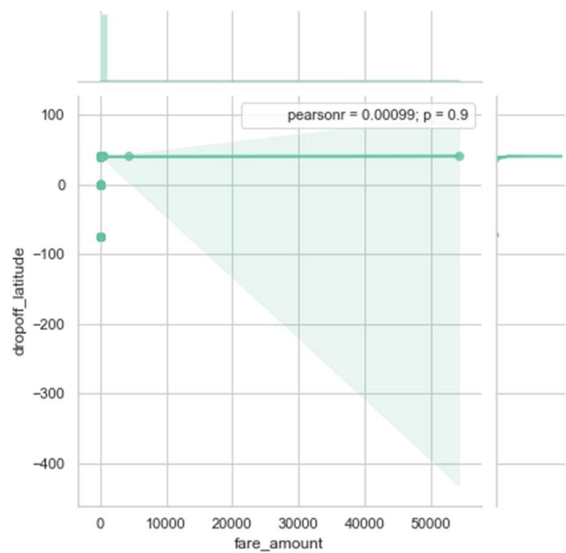
Some jointplots:

- 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.

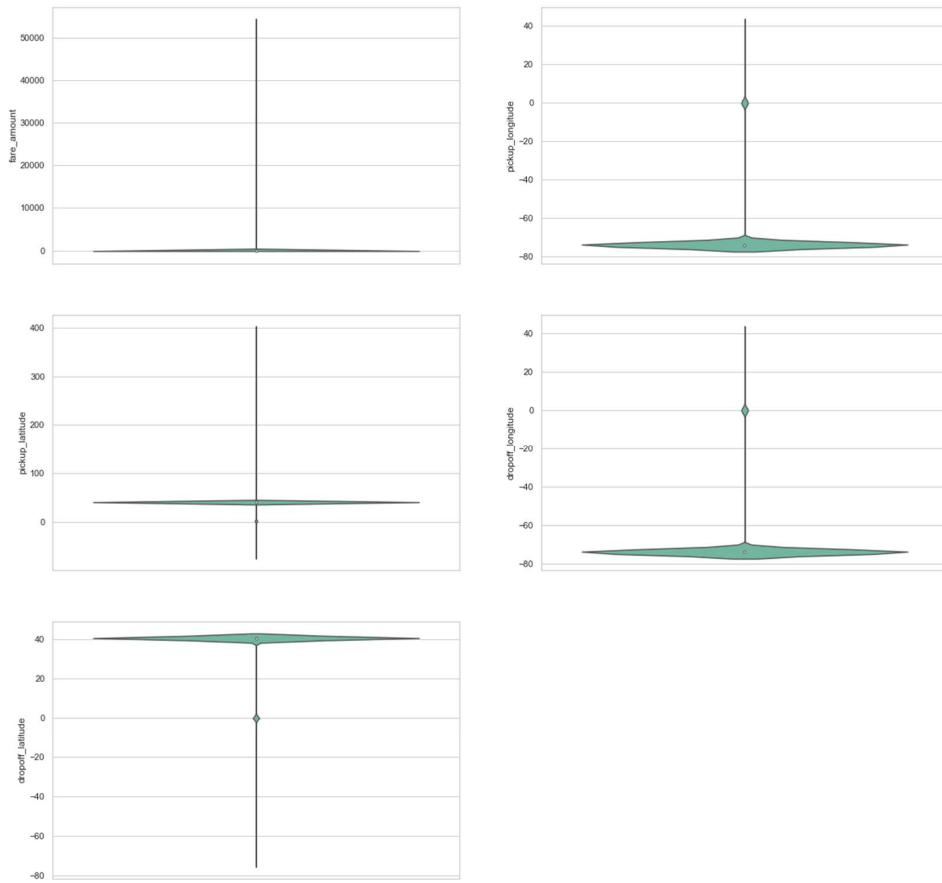

pearsonr = 0.00077; p = 0.92
pickup_latitude
fare_amount


pearsonr = -0.0011; p = 0.89
pickup_longitude
fare_amount

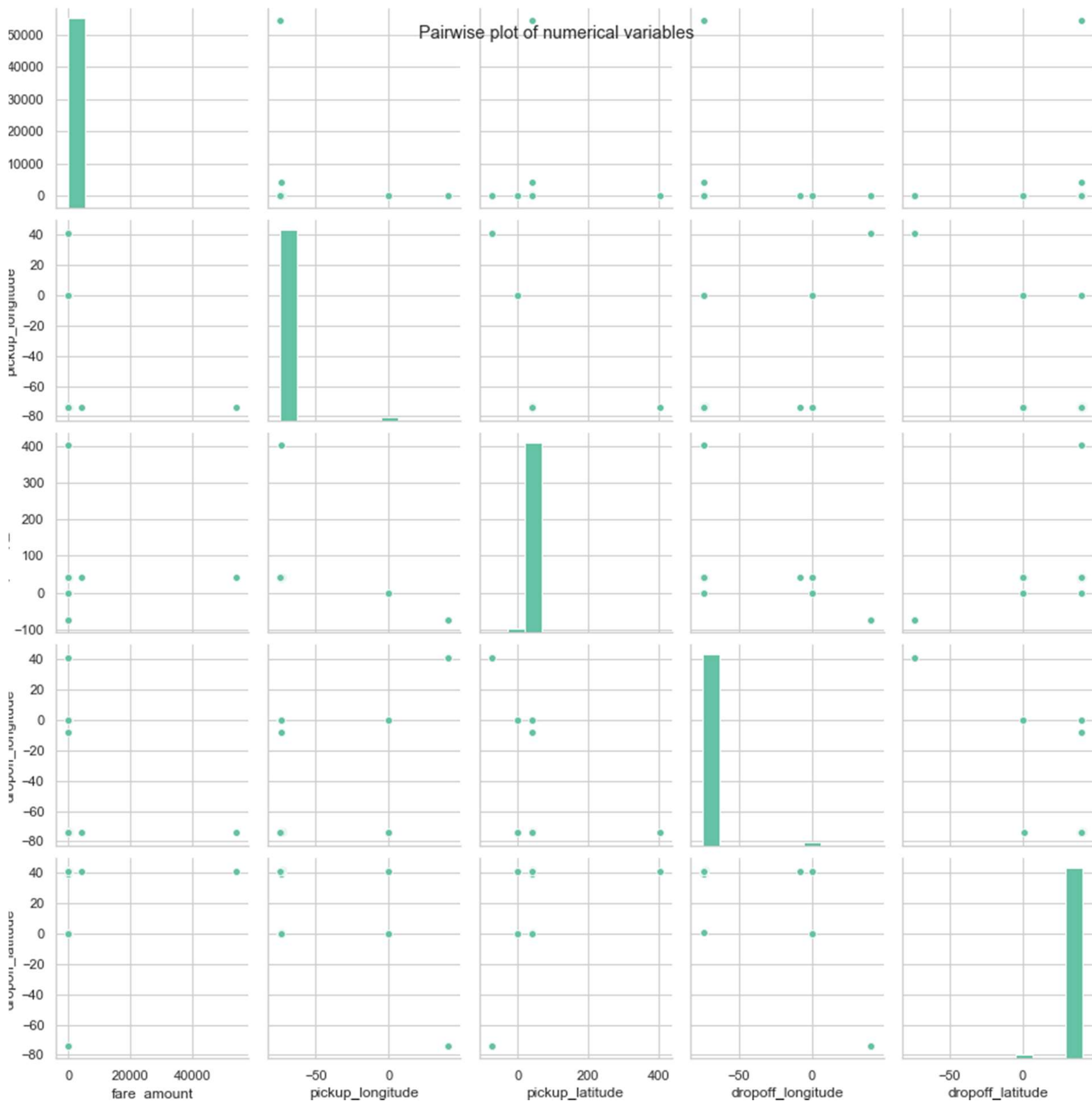

pearsonr = -0.001; p = 0.89
dropoff_longitude
fare_amount


pearsonr = 0.00099; p = 0.9
dropoff_latitude
fare_amount

Some violin plots to get the idea about till what range is the variables are spread.



Pairwise plots for all the Numerical variables:



2.2 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
passenger_count	29.563702
fare_amount	0.186718
pickup_datetime	0.006224
pickup_latitude	1.966764
pickup_longitude	1.960540
dropoff_longitude	1.954316
dropoff_latitude	1.941868

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:

passenger_count	0.0
fare_amount	0.0
pickup_longitude	0.0
pickup_latitude	0.0
dropoff_latitude	0.0
dropoff_longitude	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**
passenger_count	0.351191
fare_amount	0.140476
pickup_datetime	0.006385
pickup_longitude	0.000000
pickup_latitude	0.000000
dropoff_longitude	0.000000
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	3.1428

Name: 1000, dtype: object

If everything beyond range is made nan except passenger_count:

Variables	Missing_percentag
pickup_latitude	1.951342
dropoff_longitude	1.951342
pickup_longitude	1.945087
dropoff_latitude	1.938833
passenger_count	0.343986
fare_amount	0.181375
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

2.3 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
fare_amount	22
pickup_datetime	1
pickup_longitude	0
pickup_latitude	0
dropoff_longitude	0
dropoff_latitude	0
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
passenger_count	0.351191
fare_amount	0.140476
pickup_datetime	0.006385
pickup_longitude	0.000000
pickup_latitude	0.000000
dropoff_longitude	0.000000
dropoff_latitude	0.000000

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

fare_amount 435. 662006

passenger_count 1.264248

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

For fare_amount:

Actual value = 7.0,

Mean = 15.118,

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 435.662006

passenger_count 1.264248

dtype: float64

2.4 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,

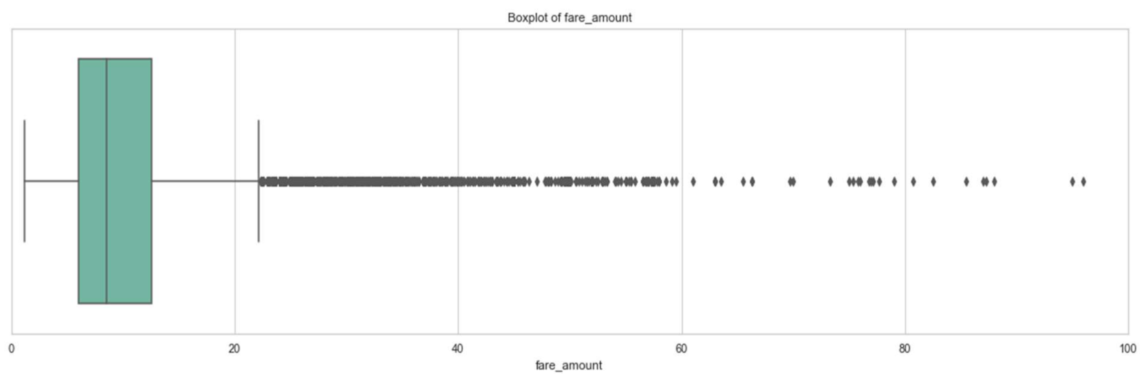
- We replaced them with Nan values or we can say created missing values.

- Then we imputed those missing values with KNN method.

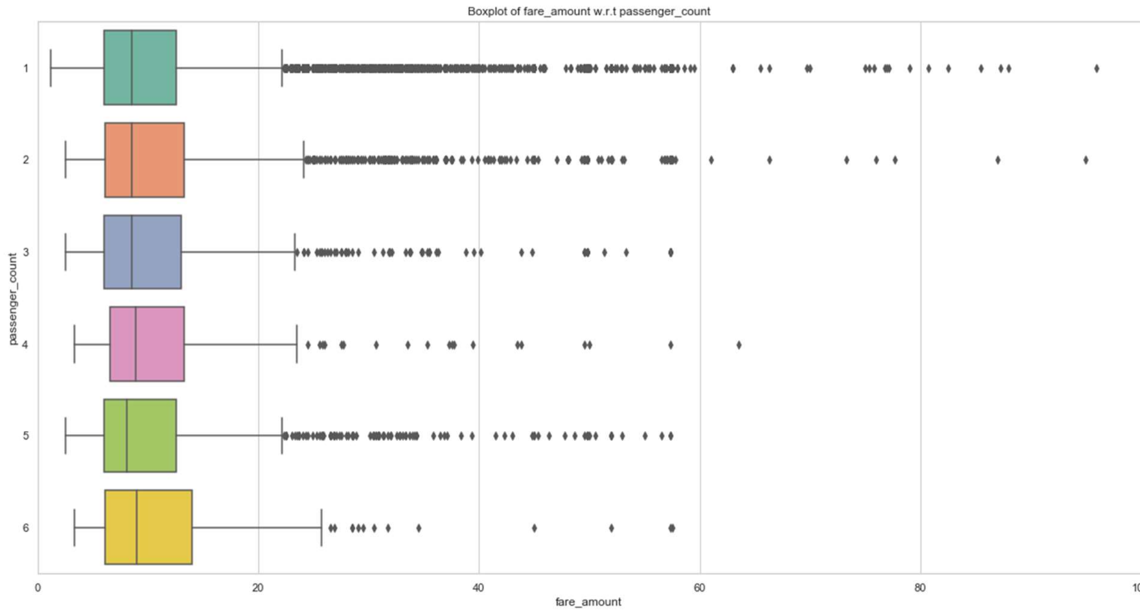
- We Will do Outlier Analysis only on fare_amount just for now and we will do outlier analysis after feature engineering latitudes and longitudes.

- **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 1359 outliers.

We successfully imputed these outliers with KNN and K value is 7

2.5 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.

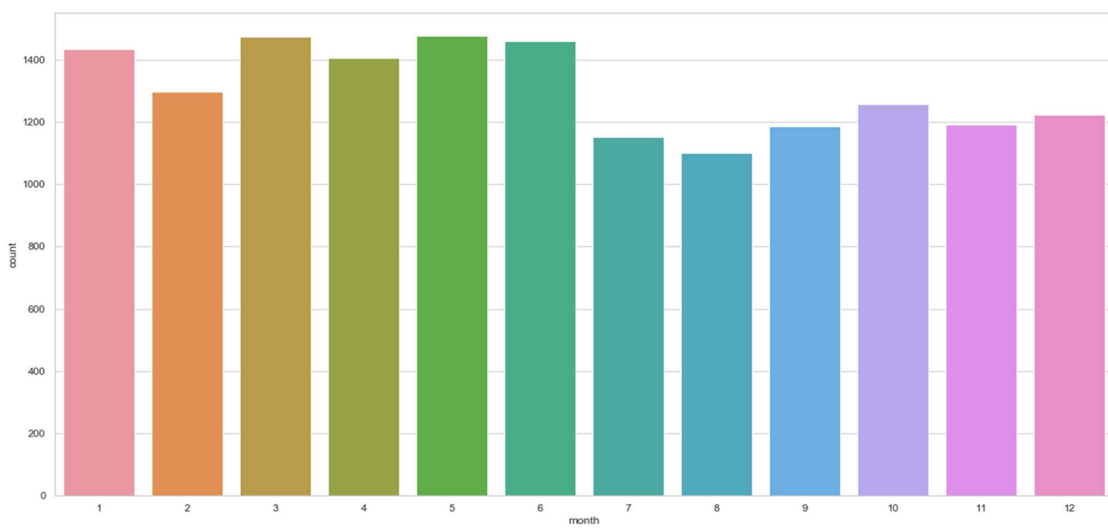
New features will be year, month, day_of_week, and hour.

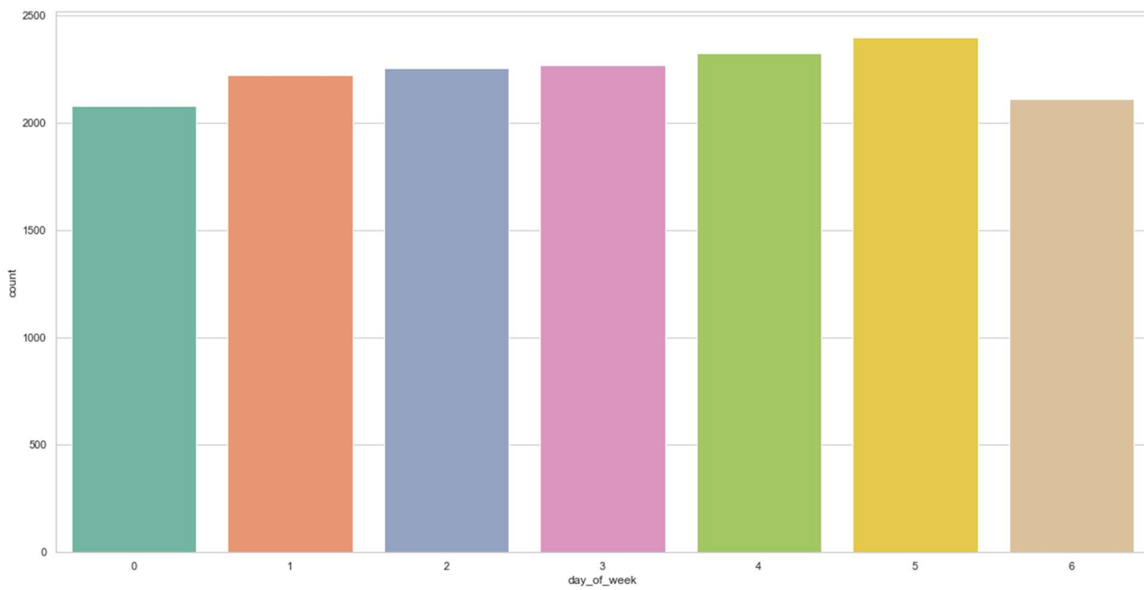
'year' will contain only years from pickup_datetime. For ex. 2009, 2010, 2011, etc.

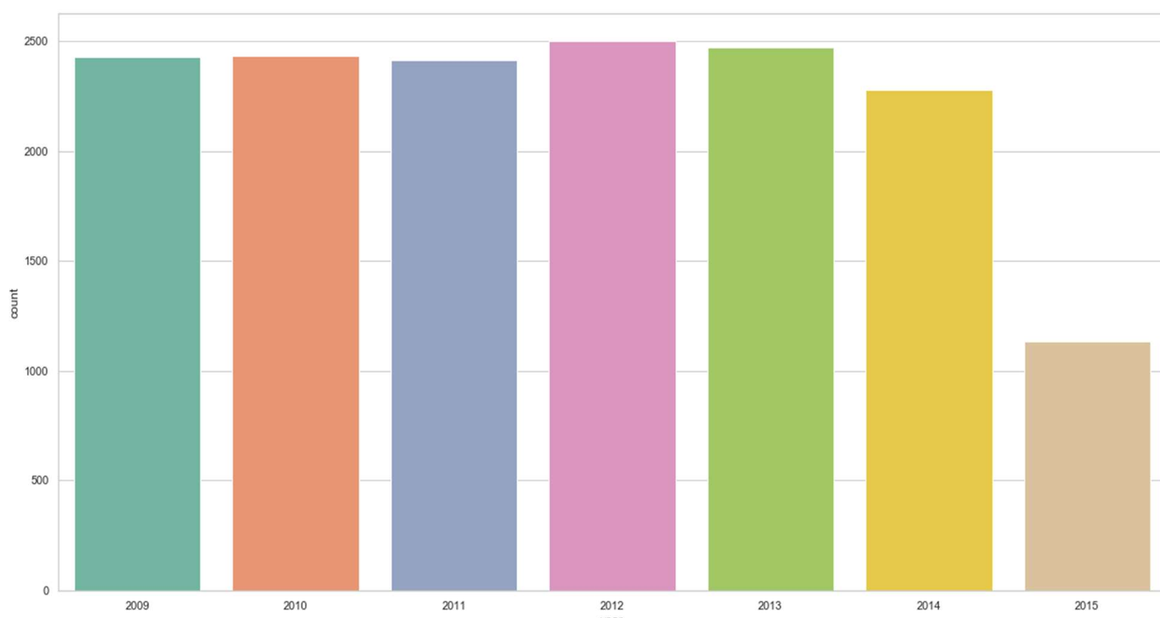
'month' will contain only months from pickup_datetime. For ex. 1 for January, 2 for February, etc.

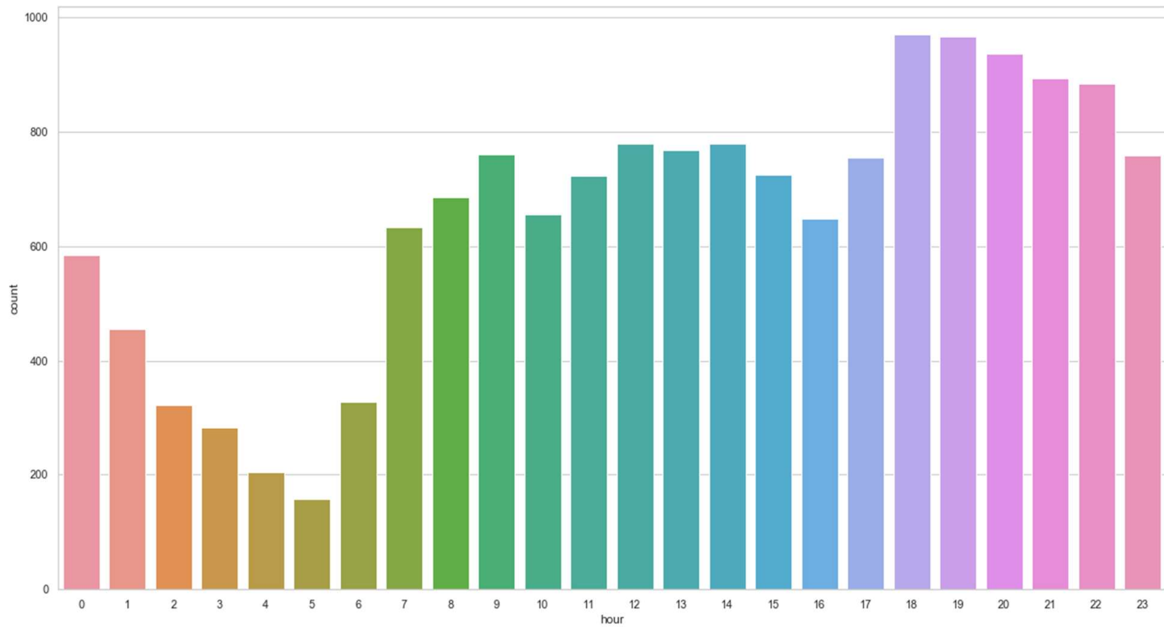
'day_of_week' will contain only week from pickup_datetime. For ex. 1 which is for Monday,2 for Tuesday,etc.

'hour' will contain only hours from pickup_datetime. For ex- 1, 2, 3, etc,




2500
2000
1500
1000
500
0
count
0
1
2
3
4
5
6
day_of_week


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



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.

2. **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 prefered for short distances.

Therefore, we will drop great_circle.

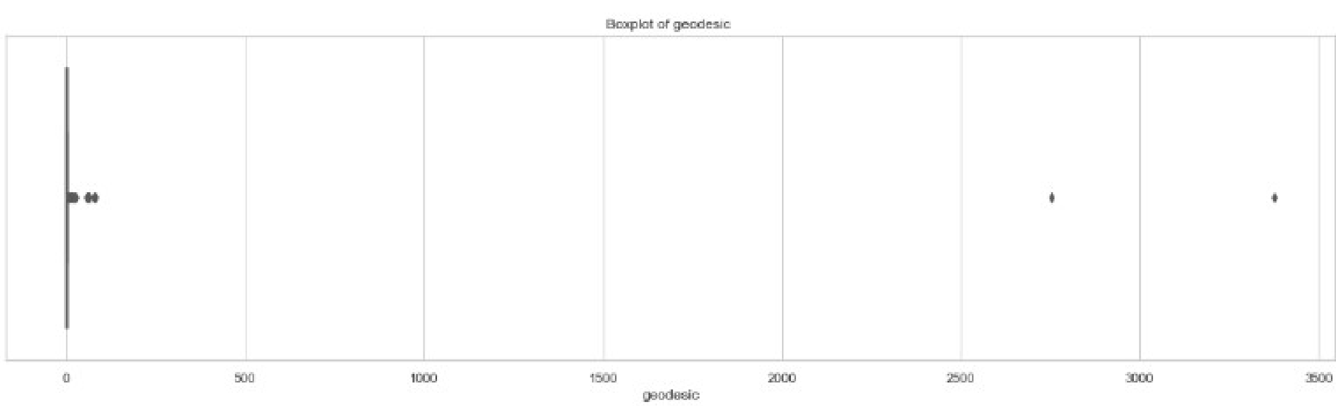
Columns in training data after feature engineering:

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:

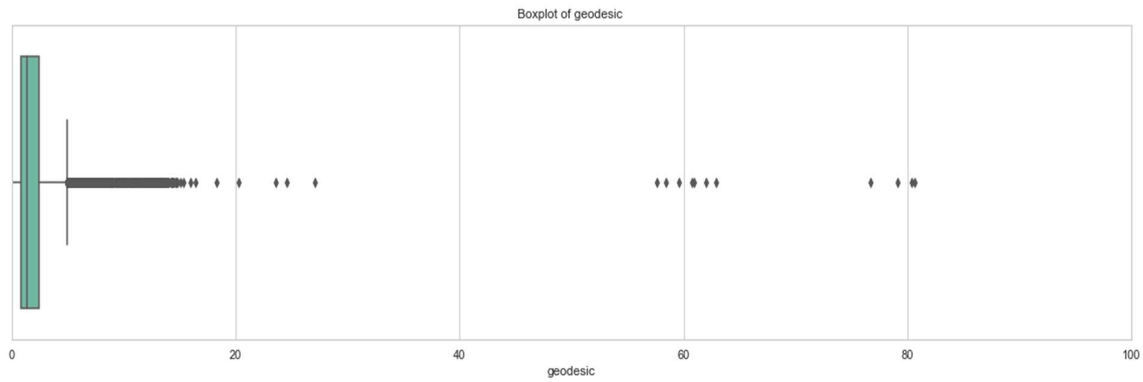
Index(['passenger_count_2', 'passenger_count_3', 'passenger_count_4',
'passenger_count_5', 'passenger_count_6','season_spring',
'season_summer','season_winter','week_weekend','session_evening',
'session_morning','session_night_AM','session_night_PM','year_2010','year_2011',
'year_2012', 'year_2013', '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 did before.

2.6 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.

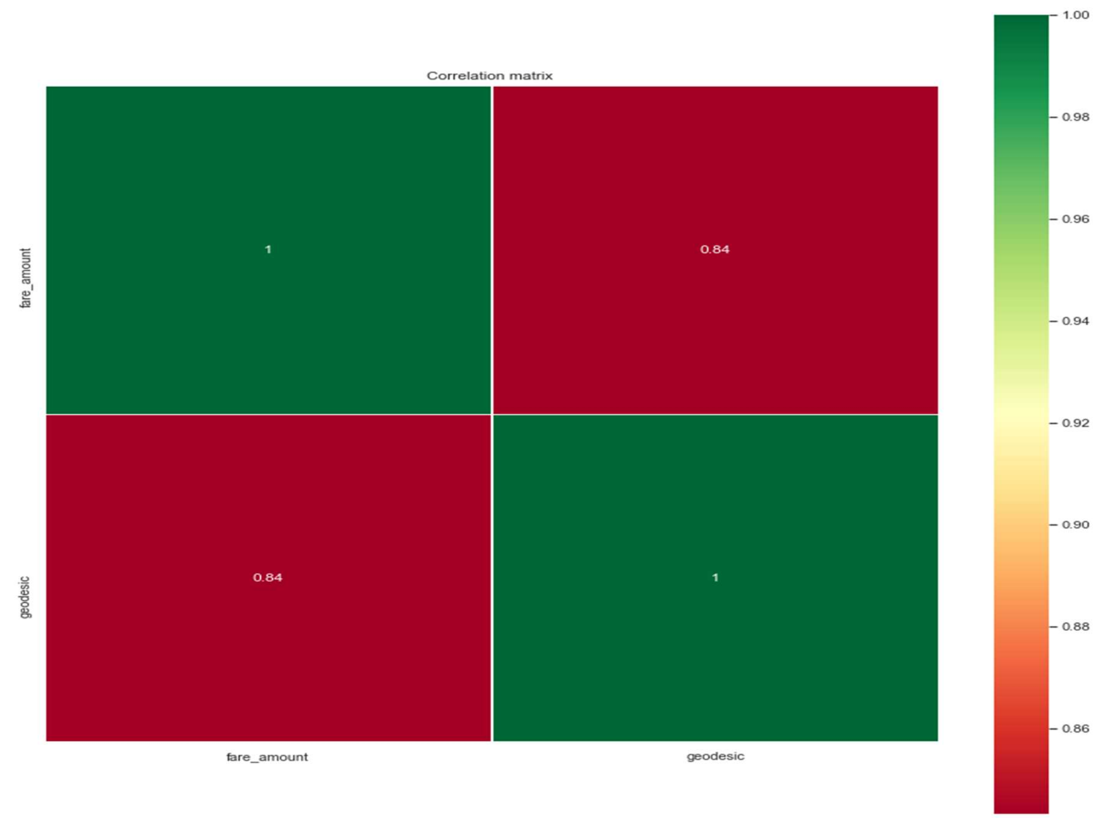
Further below are some types of test involved for feature selection:

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.

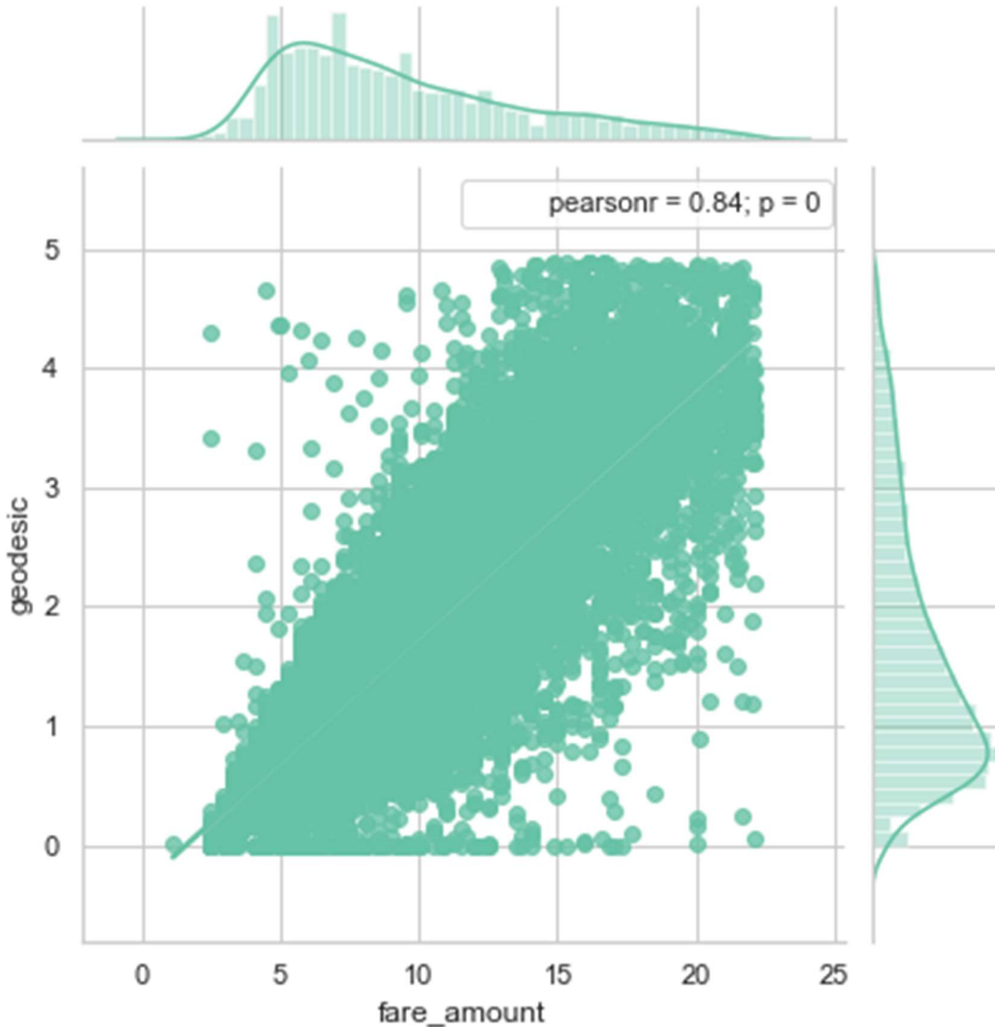
From below correlation plot we see that:

- '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’:



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.

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

- 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:

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.

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.

Analysis of Variance (Anova) Test –

I. It is carried out to compare between each group in a categorical variable.

II. ANOVA only lets us know the means for different groups are same or not. It doesn’t help us identify which mean is different.

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.

Below is the anova analysis table for each categorical variable:

	df	sum_sq	mean_sq	F	PR(>F)
C(passenger_count_2)	1.0	40.528165	40.528165	2.171219	1.406348e-01
C(passenger_count_3)	1.0	1.404893	1.404893	0.075264	7.838241e-01
C(passenger_count_4)	1.0	33.989929	33.989929	1.820946	1.772203e-01
C(passenger_count_5)	1.0	67.803517	67.803517	3.632444	5.668185e-02
C(passenger_count_6)	1.0	108.758839	108.758839	5.826548	1.579749e-02
C(season_spring)	1.0	34.661352	34.661352	1.856916	1.730008e-01
C(season_summer)	1.0	25.770171	25.770171	1.380588	2.400189e-01
C(season_winter)	1.0	535.303391	535.303391	28.677859	8.668339e-08
C(week_weekend)	1.0	100.937832	100.937832	5.407552	2.006254e-02
C(session_night_AM)	1.0	1832.485282	1832.485282	98.171908	4.485907e-23
C(session_night_PM)	1.0	165.596621	165.596621	8.871523	2.900977e-03
C(session_evening)	1.0	0.409856	0.409856	0.021957	8.822028e-01
C(session_morning)	1.0	81.177216	81.177216	4.348915	3.704871e-02
C(year_2010)	1.0	1478.878362	1478.878362	79.228091	6.131134e-19
C(year_2011)	1.0	1314.108887	1314.108887	70.400880	5.249322e-17
C(year_2012)	1.0	376.171000	376.171000	20.152645	7.201213e-06
C(year_2013)	1.0	292.014180	292.014180	15.644103	7.678762e-05
C(year_2014)	1.0	1433.569515	1433.569515	76.800756	2.082448e-18
C(year_2015)	1.0	2506.956546	2506.956546	134.305421	6.266490e-31
Residual	15640.0	291937.586214	18.666086	NaN	NaN

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

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.

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	15.357283	Intercept
1	1.040681	passenger_count_2[T.1.0]
2	1.019570	passenger_count_3[T.1.0]
3	1.011770	passenger_count_4[T.1.0]
4	1.025106	passenger_count_5[T.1.0]
5	1.017198	passenger_count_6[T.1.0]
6	1.642264	season_spring[T.1.0]
7	1.552403	season_summer[T.1.0]
8	1.587609	season_winter[T.1.0]
9	1.050873	week_weekend[T.1.0]
10	1.374308	session_night_AM[T.1.0]
11	1.422471	session_night_PM[T.1.0]
12	1.524670	session_evening[T.1.0]
13	1.558837	session_morning[T.1.0]
14	1.691370	year_2010[T.1.0]
15	1.687873	year_2011[T.1.0]
16	1.711219	year_2012[T.1.0]
17	1.709327	year_2013[T.1.0]
18	1.664974	year_2014[T.1.0]
19	1.406924	year_2015[T.1.0]
20	1.023846	geodesic

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

2.7 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. Normalization 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.

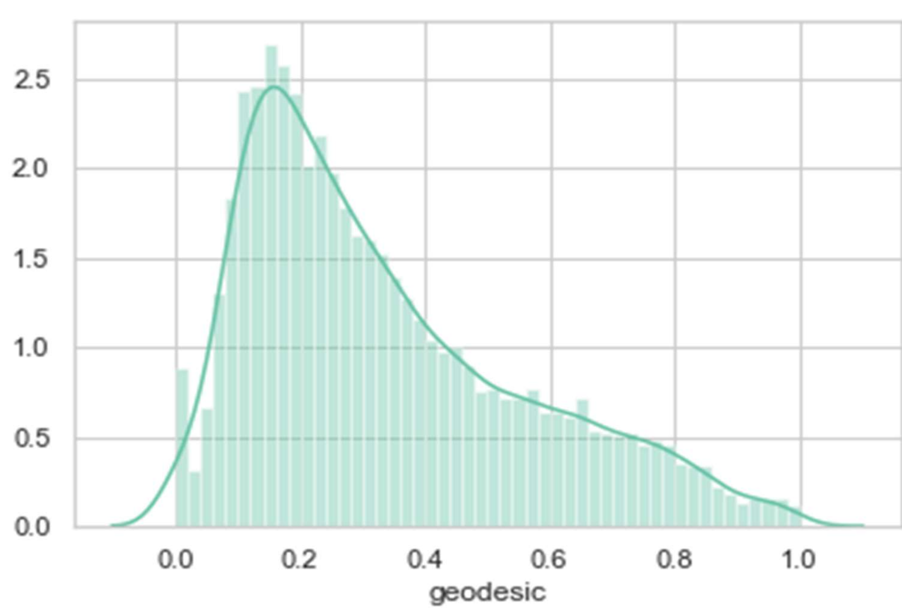
Also, our independent numerical variable 'geodesic' is not distributed normally so we had chosen normalization over standardization.

- 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.

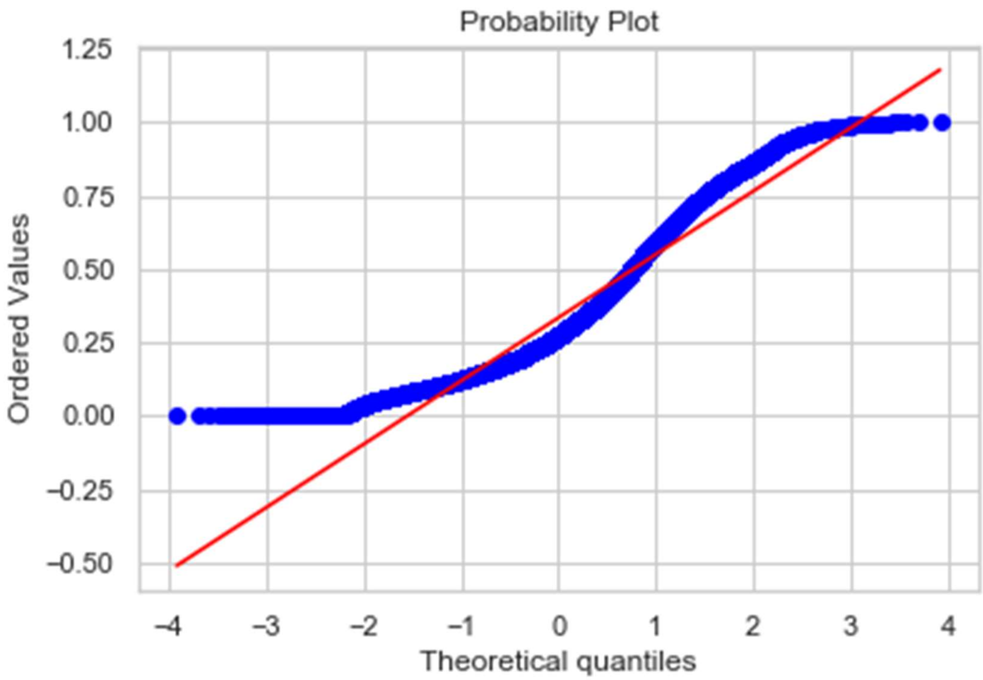
Graphs based on which standardization was chosen

Note: It is performed only on Continuous variables.

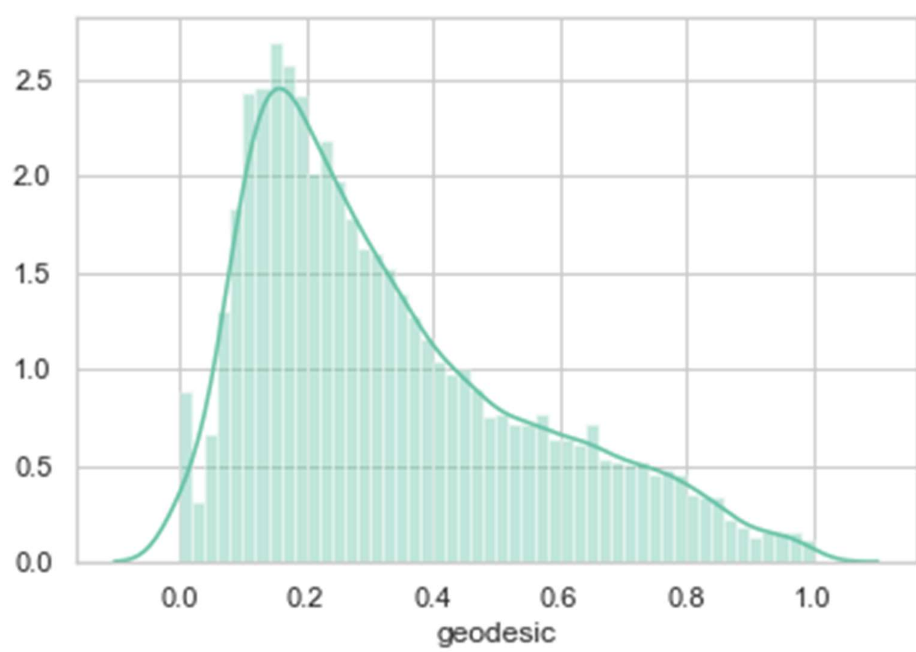
distplot() for 'geodesic' feature before normalization:



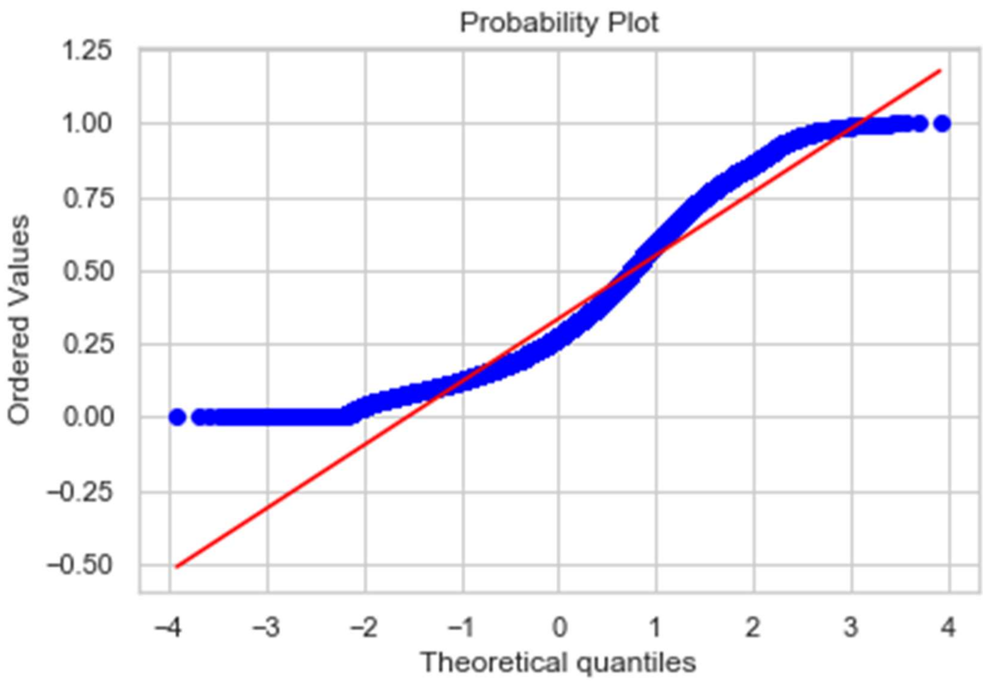
qq probability plot before normalization:



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 datset.
- 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 datset.
- 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

- To find the optimal hyperparameter we have used sklearn.model_selection.GridSearchCV. and sklearn.model_selection.RandomizedSearchCV
- GridSearchCV tries all the parameters that we provide it and then returns the best suited parameter for data.
- 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 the best hyperparameter we found for different models:

1. **Multiple Linear Regression:** Tuned Decision reg Parameters: {'copy_X': True, 'fit_intercept': True}
Best score is 0.7390030176362054
2. **Ridge Regression:** Tuned Decision ridge Parameters: {'alpha': 0.0004498432668969444, 'max_iter': 500, 'normalize': True}
Best score is 0.739003222488128
3. **Lasso Regression:** Tuned Decision lasso Parameters: {'alpha': 0.00014563484775012445, 'max_iter': 500, 'normalize': False}
Best score is 0.7390033729590965
4. **Decision Tree Regression:** Tuned Decision Tree Parameters: {'max_depth': 6, 'min_samples_split': 12}
Best score is 0.7345289221287127
5. **Random Forest Regression:** Tuned Random Forest Parameters: {'n_estimators': 300, 'min_samples_split': 2, 'min_samples_leaf': 3, 'max_features': 'log2', 'max_depth': 17, 'bootstrap': True }
Best score is 0.7445350335198329
6. **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 0.7554023297588686

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:

- Linear Regression
- Ridge Regression
- Lasso Regression
- Decision Tree
- Random Forest
- 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)

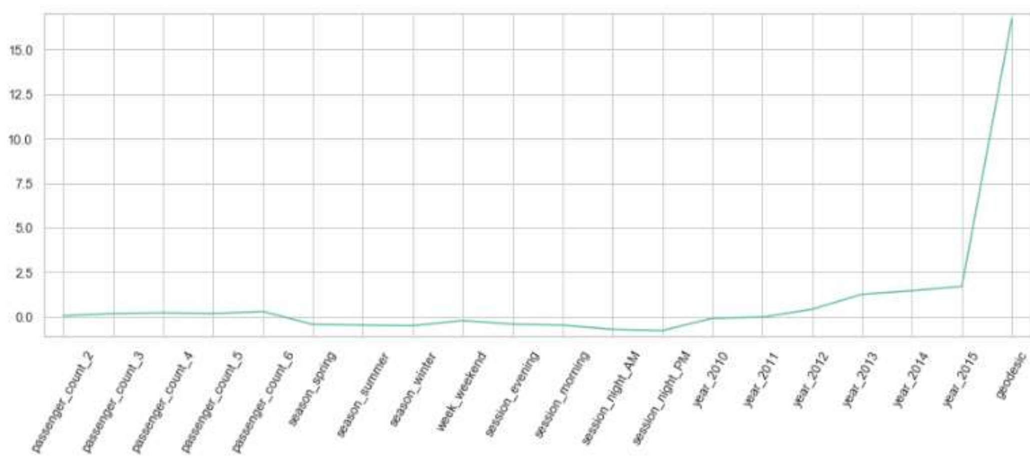
Model Performance

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

1. Multiple Linear Regression:

Error Metrics	r square	Adj r sq	MAPE	MSE	RMSE	RMSLE
Train	0.7386	0.7381	18.489	5.015	2.239	0.212
Validation	0.7433	0.7420	18.71	5.044	2.246	0.211

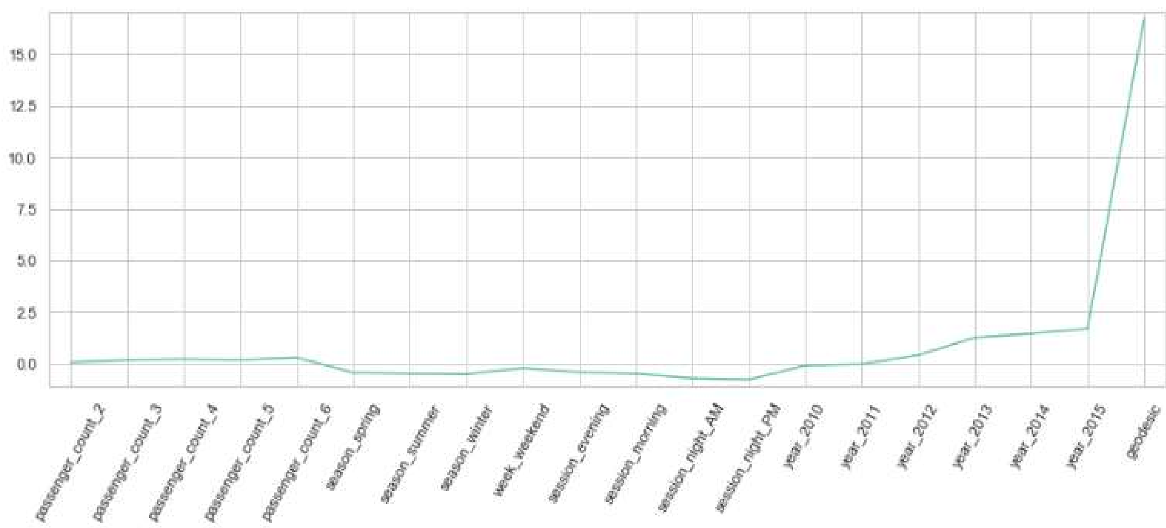
Line Plot for Coefficients of Multiple Linear regression:



2. Ridge Regression:

Error Metrics	r square	Adj r sq	MAPE	MSE	RMSE	RMSLE
Train	0.7386	0.738	18.49	5..015	2.239	0.212
validation	0.7433	0.7420	18.72	5.044	2.24	0.211

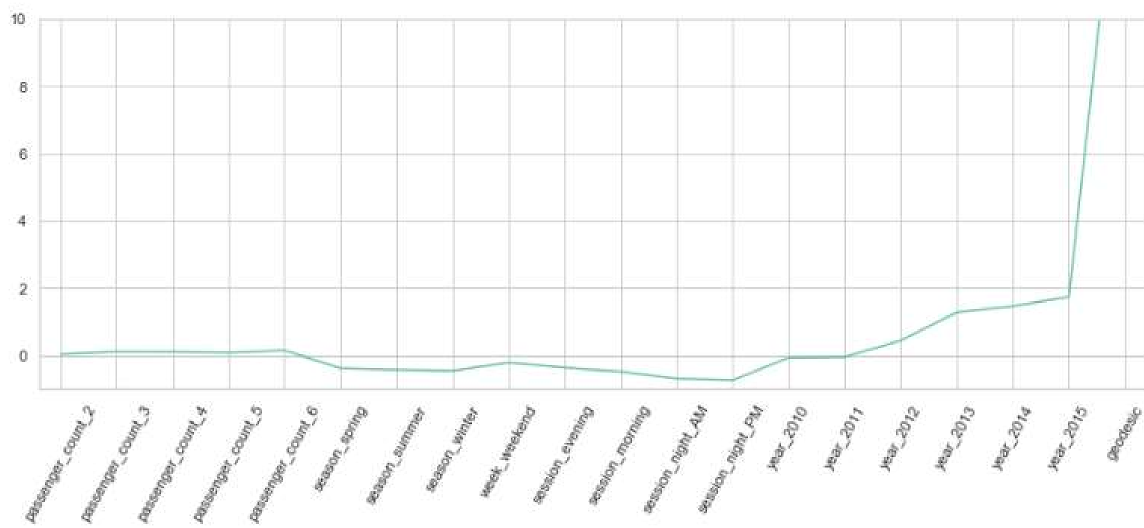
Line Plot for Coefficients of Ridge regression:



3. Lasso Regression:

Error Metrics	r square	Adj r sq	MAPE	MSE	RMSE	RMSLE
Train	0.7385	0.7380	18.50	5.018	2.240	0.212
Validation	0.7442	0.7429	18.71	5.028	2.242	0.211

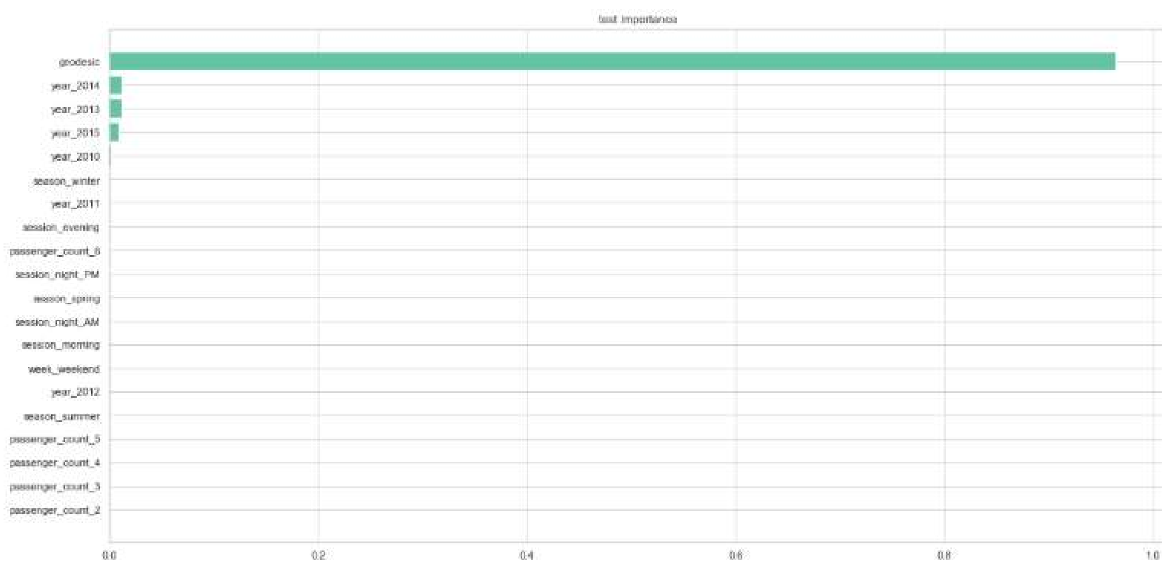
Line Plot for Coefficients of Lasso regression



4. Decision Tree Regression:

Error Metrics	r square	Adj r sq	MAPE	MSE	RMSE	RMSLE
Train	0.7510	0.7506	18.22	4.777	2.185	0.205
Validation	0.7408	0.7396	19.07	5.31	2.30	0.21

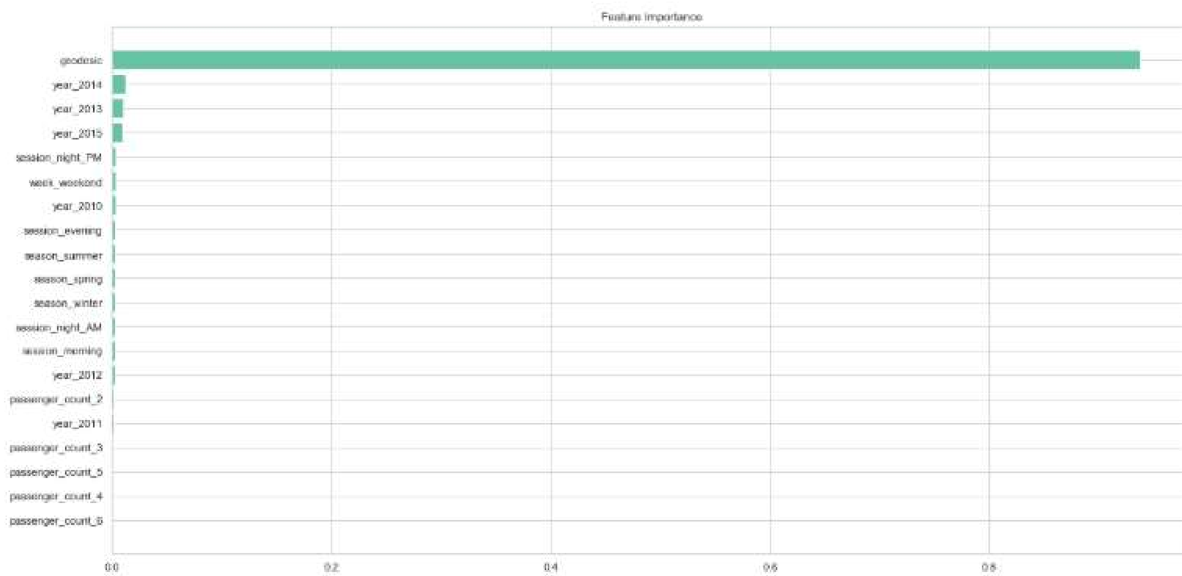
Bar Plot of Decision tree Feature Importance:



5. Random Forest Regression:

Error Metrics	r square	Adj r sq	MAPE	MSE	RMSE	RMSLE
Train	0.7948	0.7944	16.67	3.937	1.984	0.187
Validation	0.7581	0.7568	18.30	4.75	2.180	0.203

Bar Plot of Random Forest Feature Importance:



Cross validation scores:[-4.87339458 -4.82793972 -4.82389384 -4.90688533 -4.90933122]

Average 5-Fold CV Score: -4.8682889380351915

Chapter 6
Improving accuracy

- Improve Accuracy

  a) Algorithm Tuning

  b) Ensembles

- We have used xgboost as a ensemble technique.

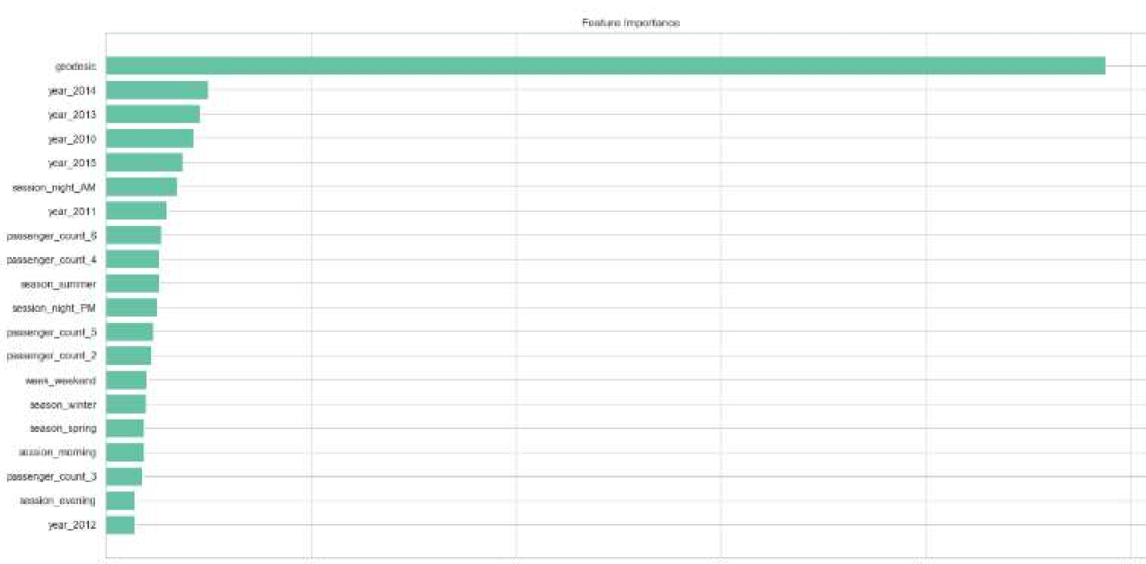
Xgboost hyperparameters tuned parameters: Tuned Xgboost Parameters: {'subsample': 0.1, 'reg_alpha': 0.08685113737513521, 'n_estimators': 200, 'max_depth': 3, 'learning_rate': 0.05, 'colsample_bytree': 0.7000000000000001, 'colsample_bynode': 0.7000000000000001, 'colsample_bylevel': 0.9000000000000001}

Best score is 0.7554023297588686

Xgboost Regression:

Error Metrics	r square	Adj r sq	MAPE	MSE	RMSE	RMSLE
Train	0.7606	0.7602	17.79	4.59	2.14	0.200
Validation	0.7638	0.7626	17.98	4.64	2.15	0.201

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   0.762537233321031

Adjusted r square:0.7622335530771803

MAPE:17.781764206110427

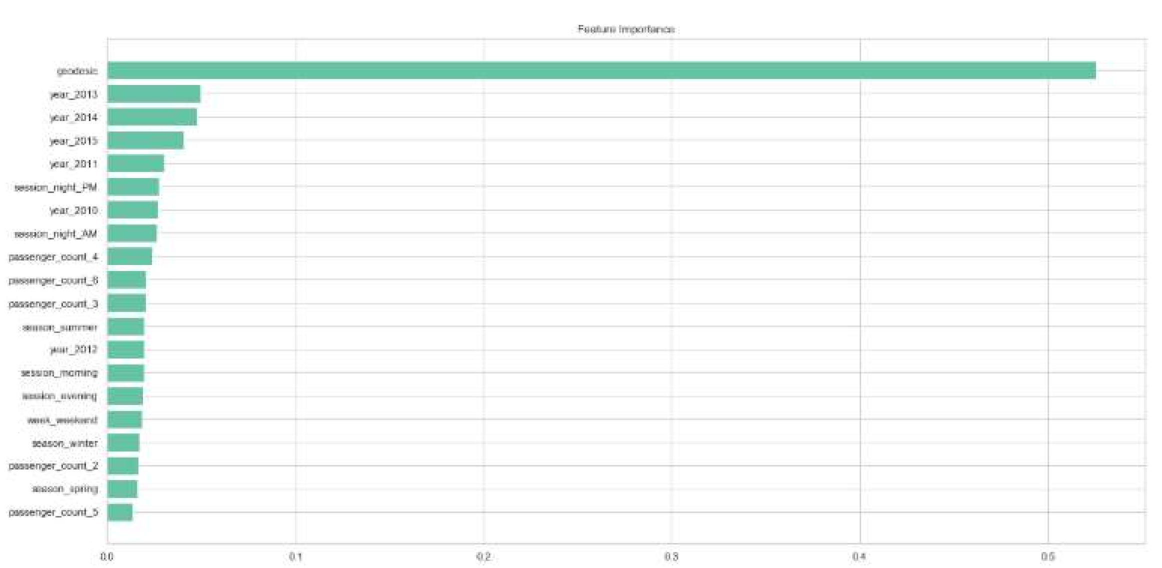
MSE: 4.585004366285271

RMSE: 2.1412623300953273

RMSLE: 0.2118407879472014

RMSLE: 0.2000174614876937

Feature importance:



Python Code

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 thehistorical data from your pilot project and now have a requirement to apply analytics forfare prediction. You need to design a system that predicts the fare amount for a cab ride in the city.

```
#loading 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 sklearn.impute import KNNImputer
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
```

```
working directory
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 metrics 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 future use

```
#importing data
train_data = pd.read_csv('train_cab.csv',dtype={'fare_amount':np.float64},na_values={'fare_amount':'430-'})
test_data = pd.read_csv('test.csv')
data = [train_data,test_data]
for i in data:
    i['pickup_datetime'] = pd.to_datetime(i['pickup_datetime'], errors = 'coerce')
train_data.head()
train_data.info()
test_data.head(5)
test_data.info()
test_data.describe()
train_data.describe()
```

Exploratory Data Analysis

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

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

Graphical Exploratory Data Analysis - Data Visualization

```
#making sns for plots
sns.set(style = 'whitegrid', palette = 'Set2' )
plt.figure(figsize=(20,20))
plt.subplot(321)
_ = sns.distplot(train_data['fare_amount'], bins=50)
plt.subplot(322)
_ = sns.distplot(train_data['pickup_longitude'], bins=50)
plt.subplot(323)
_ = sns.distplot(train_data['pickup_latitude'], bins=50)
plt.subplot(324)
_ = sns.distplot(train_data['dropoff_longitude'], bins=50)
plt.subplot(325)
_ = sns.distplot(train_data['dropoff_latitude'], bins=50)
plt.savefig('histogram.png')
plt.show()
```

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

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

Violin plots

```
plt.figure(figsize=(20,20))
plt.subplot(321)
_ = sns.violinplot(y='fare_amount', data=train_data)
plt.subplot(322)
_ = sns.violinplot(y='pickup_longitude', data=train_data)
plt.subplot(323)
_ =sns.violinplot(y='pickup_latitude', data=train_data)
plt.subplot(324)
_ = sns.violinplot(y='dropoff_longitude', data=train_data)
plt.subplot(325)
_ = sns.violinplot(y='dropoff_latitude', data=train_data)
#plt.savefig('violinplot.png')
plt.show()
```

Pair plots

```
_ =sns.pairplot(data=train_data[nume_var],kind='scatter',dropna=True)
_.fig.suptitle('Pairwise plot of numerical variables')
#plt.savefig('Pairwiseplot.png')
plt.show()
sum(train_data['fare_amount']<1)
train_data[train_data['fare_amount']<1]
train_data = train_data.drop(train_data[train_data['fare_amount']<1].index, axis=0)
for i in range(4,11):
    print('passenger_count above' +str(i)+'={}'.format(sum(train_data['passenger_count']>i)))
train_data[train_data['passenger_count']>6]
train_data[train_data['passenger_count']<1]
len(train_data[train_data['passenger_count']<1])
test_data['passenger_count'].unique()
train_data = train_data.drop(train_data[train_data['passenger_count']>6].index, axis=0)
train_data = train_data.drop(train_data[train_data['passenger_count']<1].index, axis=0)
sum(train_data['passenger_count']>6)
```

```
print('pickup_longitude above 180={}'.format(sum(train_data['pickup_longitude']>180)))
print('pickup_longitude below -180={}'.format(sum(train_data['pickup_longitude']<-180)))
print('pickup_latitude above 90={}'.format(sum(train_data['pickup_latitude']>90)))
print('pickup_latitude below -90={}'.format(sum(train_data['pickup_latitude']<-90)))
print('dropoff_longitude above 180={}'.format(sum(train_data['dropoff_longitude']>180)))
print('dropoff_longitude below -180={}'.format(sum(train_data['dropoff_longitude']<-180)))
print('dropoff_latitude below -90={}'.format(sum(train_data['dropoff_latitude']<-90)))
print('dropoff_latitude above 90={}'.format(sum(train_data['dropoff_latitude']>90)))
for i in ['pickup_longitude','pickup_latitude','dropoff_longitude','dropoff_latitude']:
    print(i,'equal to 0={}'.format(sum(train_data[i]==0)))
train_data = train_data.drop(train_data[train_data['pickup_latitude']>90].index, axis=0)
for i in ['pickup_longitude','pickup_latitude','dropoff_longitude','dropoff_latitude']:
    train_data = train_data.drop(train_data[train_data[i]==0].index, axis=0)
train_data.shape
#df=train_data.copy()
train_data=df.copy()
```

Missing Value Analysis

```
#Create dataframe with missing percentage
missing_val = pd.DataFrame(train_data.isnull().sum())
#Reset index
missing_val = missing_val.reset_index()
missing_val
#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_data))*100
#descending order
missing_val = missing_val.sort_values('Missing_percentage', ascending = False).reset_index(drop = True)
missing_val
# Choose random values to replace it as NA
train_data['passenger_count'].loc[1000]
# Replacing 1.0 with NA
train_data['passenger_count'].loc[1000] = np.nan
train_data['passenger_count'].loc[1000]
# Impute with mode
train_data['passenger_count'].fillna(train_data['passenger_count'].mode()[0]).loc[1000]
# Choosing a random values to replace it as NA
a=train_data['fare_amount'].loc[1000]
```

```
print('fare_amount at loc-1000:{}'.format(a))
# Replacing 1.0 with NA
train_data['fare_amount'].loc[1000] = np.nan
print('Value after replacing with nan:{}'.format(train_data['fare_amount'].loc[1000]))
# Impute with mean
print('Value if imputed with mean:{}'.format(train_data['fare_amount'].fillna(train_data['fare_amount'].mean()).loc
[1000]))
# Impute with median
print('Value if imputed with median:{}'.format(train_data['fare_amount'].fillna(train_data['fare_amount'].median())
.loc[1000]))
train_data.std()
columns=['fare_amount', 'pickup_longitude', 'pickup_latitude','dropoff_longitude', 'dropoff_latitude', 'passenger_co
unt']
pickup_datetime=pd.DataFrame(train_data['pickup_datetime'])
# Imputing with missing values using KNN
train_data = pd.DataFrame(KNNImputer(n_neighbors = 7).fit_transform(train_data.drop('pickup_datetime',axis=1)
),columns=columns, index=train_data.index)
train_data.std()
train_data.loc[1000]
train_data['passenger_count'].head()
train_data['passenger_count']=train_data['passenger_count'].astype('int')
train_data.std()
train_data['passenger_count'].unique()

train_data['passenger_count']=train_data['passenger_count'].round().astype('object').astype('category')
train_data['passenger_count'].unique()
train_data.loc[1000]
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_data.shape
df1 = train_data.copy()
#train_data = df1.copy()
train_data['passenger_count'].describe()
train_data.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 laitudes 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_data['fare_amount'],data=train_data,orient='h')
plt.title('Boxplot of fare_amount')
#plt.savefig('boxplot of fare_amount.png')
plt.show()
plt.figure(figsize=(20,10))
plt.xlim(0,100)
_ = sns.boxplot(x=train_data['fare_amount'],y=train_data['passenger_count'],data=train_data,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_data.describe()
train_data['passenger_count'].describe()
```

Outlier Treatment

- From the above Boxplots we can say that there are outliers in the train dataset.
- Reconsider pickup_longitude,etc.

```
def outliers_treatment(col):

  ##calculating outlier indices and replace them with NA
  #Extract quartiles

  q75, q25 = np.percentile(train_data[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)

    #Replacing with NA

    train_data.loc[train_data[col] < minimum,col] = np.nan
    train_data.loc[train_data[col] > maximum,col] = np.nan
for i in nume_var:
    outliers_treatment('fare_amount')
    #outliers_treatment('pickup_longitude')
    #outliers_treatment('pickup_latitude')
    #outliers_treatment('dropoff_longitude')
    #outliers_treatment('dropoff_latitude')
pd.DataFrame(train_data.isnull().sum())
train_data.std()
#Imputing with missing values using KNN
train_data = pd.DataFrame(KNNImputer(n_neighbors = 7).fit_transform(train_data), columns = train_data.column
s, index=train_data.index)
train_data.std()
train_data['passenger_count'].describe()
train_data['passenger_count']=train_data['passenger_count'].astype('int').round().astype('object').astype('category')
train_data.describe()
train_data.head()
df2 = train_data.copy()
#train=df2.copy()
train_data.shape
```

Feature Engineering

Feature Engineering for timestamp variable

- Let's us derive new features from pickup_datetime variable
- New features will be year,month,day_of_week,hour

```
# Let's Join 2 Dataframes pickup_datetime and train
train_data = pd.merge(pickup_datetime,train_data,right_index=True,left_index=True)
train_data.head()
train_data.shape
train_data = train_data.reset_index(drop=True)
pd.DataFrame(train_data.isna().sum())
train_data = train_data.dropna()
data = [train_data,test_data]
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_data.copy()
# train_data=train_nodummies.copy()
plt.figure(figsize=(20,10))
sns.countplot(train_data['year'])
plt.savefig('year.png')

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

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

plt.figure(figsize=(20,10))
sns.countplot(train_data['hour'])
plt.savefig('hour.png')
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)|(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_data['session'] = train_data['hour'].apply(f)
test_data['session'] = test_data['hour'].apply(f)
#train_nodummies['session'] = train_nodummies['hour'].apply(f)
train_data['seasons'] = train_data['month'].apply(g)
test_data['seasons'] = test_data['month'].apply(g)
#train_data['seasons'] = test_data['month'].apply(g)
train_data['week'] = train_data['day_of_week'].apply(h)
test_data['week'] = test_data['day_of_week'].apply(h)
train_data.shape
test_data.shape
```

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.
- Let's use one hot encoding technique for passenger_count variable.

```
train_data['passenger_count'].describe()
#Creating dummies for each variable in passenger_count and merging dummies dataframe to both train and test dat
aframe
temp = pd.get_dummies(train_data['passenger_count'], prefix = 'passenger_count')
train_data = train_data.join(temp)
temp = pd.get_dummies(test_data['passenger_count'], prefix = 'passenger_count')
test_data = test_data.join(temp)
temp = pd.get_dummies(train_data['seasons'], prefix = 'season')
train_data = train_data.join(temp)
temp = pd.get_dummies(test_data['seasons'], prefix = 'season')
test_data = test_data.join(temp)
temp = pd.get_dummies(train_data['week'], prefix = 'week')
train_data = train_data.join(temp)
temp = pd.get_dummies(test_data['week'], prefix = 'week')
test_data = test_data.join(temp)
temp = pd.get_dummies(train_data['session'], prefix = 'session')
train_data = train_data.join(temp)
temp = pd.get_dummies(test_data['session'], prefix = 'session')
```

```
test_data = test_data.join(temp)
temp = pd.get_dummies(train_data['year'], prefix = 'year')
train_data = train_data.join(temp)
temp = pd.get_dummies(test_data['year'], prefix = 'year')
test_data = test_data.join(temp)
train_data.head()
test_data.head()
train_data.columns
train_data = train_data.drop(['passenger_count_1','season_fall','week_weekday','session_afternoon','year_2009'],axis=1)
test_data=test_data.drop(['passenger_count_1','season_fall','week_weekday','session_afternoon','year_2009'],axis=1)
```

Feature Engineering for latitude and longitude variable

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

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

#def haversine(coord1, coord2):

    ###Calculate distance the cab travelled from pickup and dropoff location using the Haversine Formula

 #    data = [train_data, test_data]

  #   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

# Calculate distance the cab travelled from pickup and dropoff location using great_circle from geopy library

data = [train_data, test_data]

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)

pd.DataFrame(train_data.isna().sum())

pd.DataFrame(test_data.isna().sum())

train_data=train_data.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_data=test_data.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)

plt.figure(figsize=(20,5))

sns.boxplot(x=train_data['geodesic'],data=train_data,orient='h')
```

```
plt.title('Boxplot of geodesic ')

#plt.savefig('boxplot of geodesic.png')

plt.show()

plt.figure(figsize=(20,5))

plt.xlim(0,100)

sns.boxplot(x=train_data['geodesic'],data=train_data,orient='h')

plt.title('Boxplot of geodesic ')

#plt.savefig('boxplot of geodesic.png')

plt.show()

outliers_treatment('geodesic')

pd.DataFrame(train_data.isnull().sum())

#Imputing with missing values using KNN

train_data = pd.DataFrame(KNNImputer(n_neighbors = 7).fit_transform(train_data), columns = train_data.columns, index=train_data.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 bia s into our models.

```
cate_var=['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']

nume_var=['fare_amount','geodesic']

train_data[cate_var]=train_data[cate_var].apply(lambda x: x.astype('category') )

test_data[cate_var]=test_data[cate_var].apply(lambda x: x.astype('category') )

# heatmap using correlation matrix

plt.figure(figsize=(15,15))

_ = sns.heatmap(train_data[nume_var].corr(), square=True, cmap='RdYlGn',linewidths=0.5,linecolor='w',annot=True)

plt.title('Correlation matrix ')

plt.savefig('correlation.png')

plt.show()

_ = sns.jointplot(x='fare_amount',y='geodesic',data=train_data,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 two variables are dependent.
- And if p-value is greater than 0.05 then we accept the null hypothesis saying that two variables are independent.
- There should be no dependencies between Independent variables.
- So let's remove that variable whose p-value with other variable is low than 0.05.
- And let's keep that variable whose p-value with other variable is high than 0.05

```
#loop for chi square values

for i in cate_var:

    for j in cate_var:

        if(i != j):

            chi2, p, dof, ex = chi2_contingency(pd.crosstab(train_data[i], train_data[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_data.columns

#ANOVA
_1)+C(passenger_count_2)+C(passenger_count_3)+C(passenger_count_4)+C(passenger_count_5)+C(passen
ger_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(season_spring)+C(season_summer)+C(season_winter)+C(week_weekend)+C(session_night_A
M)+C(session_night_PM)+C(session_evening)+C(session_morning)+C(year_2010)+C(year_2011)+C(year_
2012)+C(year_2013)+C(year_2014)+C(year_2015)',data=train_data).fit()
```

aov_table = sm.stats.anova_lm(model)

aov_table

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+session_morning+year_2010+year_2011+year_2012+year_2013+year_2014+year_2015',train_data, 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

Feature Scaling Check with or without normalization of standard scalar

train_data[nume_var].var()

sns.distplot(train_data['geodesic'],bins=50)

#plt.savefig('distplot.png')

plt.figure()

stats.probplot(train_data['geodesic'], dist='norm', fit=True,plot=plt)

```
#plt.savefig('qq prob plot.png')

#Normalization

train_data['geodesic'] = (train_data['geodesic'] - min(train_data['geodesic']))/(max(train_data['geodesic']) - min(train_data['geodesic']))

test_data['geodesic'] = (test_data['geodesic'] - min(test_data['geodesic']))/(max(test_data['geodesic']) - min(test_data['geodesic']))

train_data['geodesic'].var()

sns.distplot(train_data['geodesic'],bins=50)

#plt.savefig('distplot.png')

plt.figure()

stats.probplot(train_data['geodesic'], dist='norm', fit=True,plot=plt)

plt.savefig('qq prob plot1.png')

train_data.columns

#df4=train_data.copy()

train_data=df4.copy()

#f4=test_data.copy()

test_data=f4.copy()

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

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

train_data.columns
```

Splitting train into train and validation subsets

- X_train y_train--are train subset
- X_test y_test--are validation subset

```
X = train_data.drop('fare_amount',axis=1).values
```

```
y = train_data['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_data.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 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

```
# 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

- Algorithm Tuning
- 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)
```

Finalizing the model

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

```
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 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_data.columns

train_data.columns

train_data.shape

test_data.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')
```