“Cab Fare Prediction”

(A data science project report)

Arun Kumar

Submitted By

Arun Kumar

arunk.recs.cse@gmail.com

To

edWisor.com

Contents

Arun Kumar

Chapter 1

Introduction

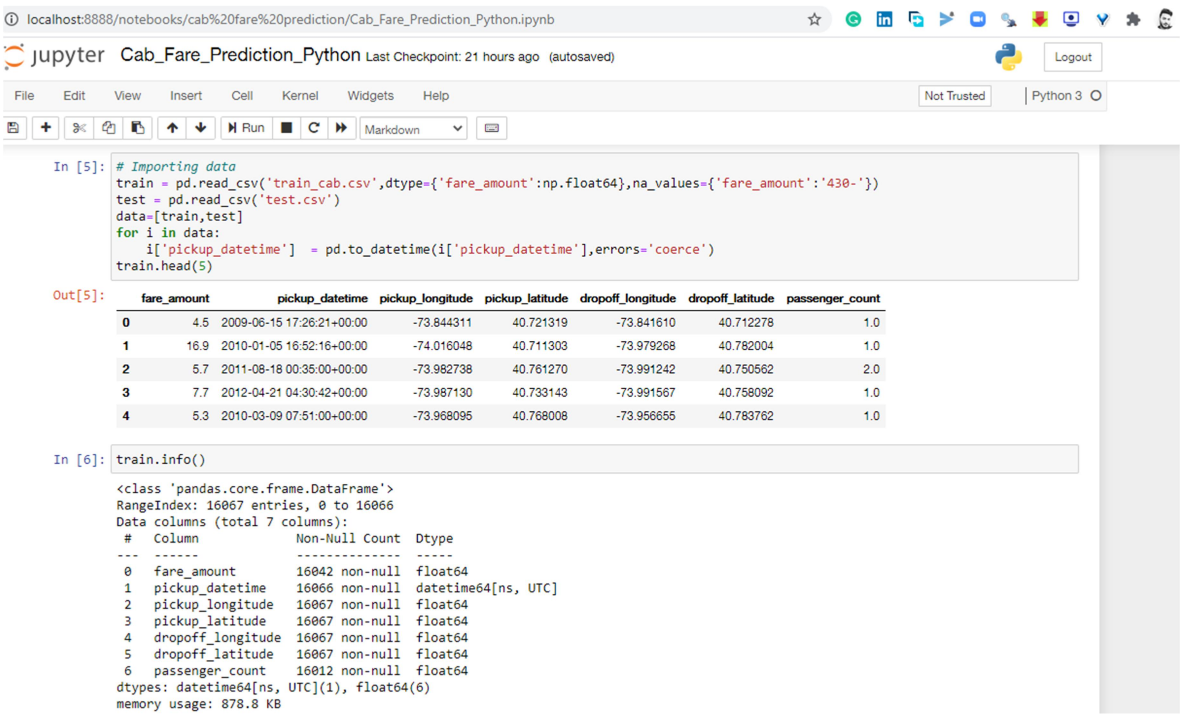
1.1 Problem Statement

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

We have total 7 data columns in train data and 6 data columns in test data. For more details look below pictures.

Arun Kumar




```
# 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)
```

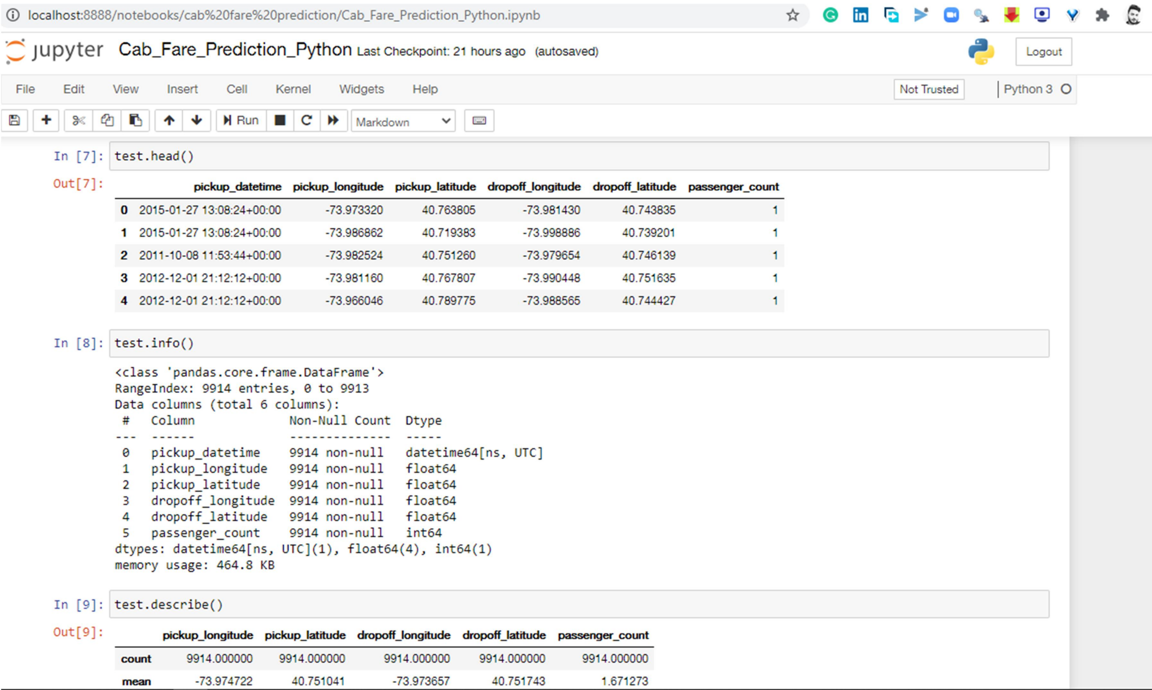
Out[5]:

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

In [6]:

```
train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 16067 entries, 0 to 16066
Data columns (total 7 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   fare_amount        16042 non-null  float64
 1   pickup_datetime    16066 non-null  datetime64[ns, UTC]
 2   pickup_longitude   16067 non-null  float64
 3   pickup_latitude    16067 non-null  float64
 4   dropoff_longitude  16067 non-null  float64
 5   dropoff_latitude   16067 non-null  float64
 6   passenger_count    16012 non-null  float64
dtypes: datetime64[ns, UTC](1), float64(6)
memory usage: 878.8 KB
```



localhost:8888/notebooks/cab%20fare%20prediction/Cab_Fare_Prediction_Python.ipynb

jupyter Cab_Fare_Prediction_Python Last Checkpoint: 21 hours ago (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help

Not Trusted | Python 3

Run | Markdown

In [7]:
```
test.head()
```
Out[7]:

	pickup_datetime	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_count
0	2015-01-27 13:08:24+00:00	-73.973320	40.763805	-73.981430	40.743835	1
1	2015-01-27 13:08:24+00:00	-73.986862	40.719383	-73.998886	40.739201	1
2	2011-10-08 11:53:44+00:00	-73.982524	40.751260	-73.979654	40.746139	1
3	2012-12-01 21:12:12+00:00	-73.981160	40.767807	-73.990448	40.751635	1
4	2012-12-01 21:12:12+00:00	-73.966046	40.789775	-73.988565	40.744427	1

In [8]:
```
test.info()
```
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9914 entries, 0 to 9913
Data columns (total 6 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   pickup_datetime    9914 non-null   datetime64[ns, UTC]
 1   pickup_longitude   9914 non-null   float64
 2   pickup_latitude    9914 non-null   float64
 3   dropoff_longitude  9914 non-null   float64
 4   dropoff_latitude   9914 non-null   float64
 5   passenger_count    9914 non-null   int64
dtypes: datetime64[ns, UTC](1), float64(4), int64(1)
memory usage: 464.8 KB
```

In [9]:
```
test.describe()
```
Out[9]:

	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_count
count	9914.000000	9914.000000	9914.000000	9914.000000	9914.000000
mean	-73.974722	40.751041	-73.973657	40.751743	1.671273

Arun Kumar

Attributes:-

- pickup_datetime - timestamp value indicating when the cab ride started.
- pickup_longitude - float for longitude coordinate of where the cab rides started.
- pickup_latitude - float for latitude coordinate of where the cab rides 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.

2.1.0 Exploratory Data Analysis

Exploratory Data Analysis (EDA) is an approach to analysing data sets and Summarize their main characteristics. Our train data consists 16067 observation and 7 variables. Data type of all variables is either int64 or float64 or datetime64. As per the data analysis we hav e to find which variables are the categorical variables, continuous variables and target variable. Data types need to be change accordingly. We have distributed the variables on the basis of conti nuous and categorical variables. Target variable is continuous. We have total 16067 observation , but as per above summary tables total observation is <16067 in some variables. Its means there is missing values present in our dataset. Missing value analysis is required to further understand t he data.

```
Train data summery
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 16067 entries, 0 to 16066
Data columns (total 7 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   fare_amount        16042 non-null  float64
 1   pickup_datetime    16066 non-null  datetime64[ns, UTC]
 2   pickup_longitude   16067 non-null  float64
 3   pickup_latitude    16067 non-null  float64
 4   dropoff_longitude  16067 non-null  float64
 5   dropoff_latitude   16067 non-null  float64
 6   passenger_count    16012 non-null  float64
dtypes: datetime64[ns, UTC](1), float64(6)
memory usage: 878.8 KB
```

```
test data summery
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9914 entries, 0 to 9913
Data columns (total 6 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   pickup_datetime    9914 non-null   datetime64[ns, UTC]
 1   pickup_longitude   9914 non-null   float64
 2   pickup_latitude    9914 non-null   float64
 3   dropoff_longitude  9914 non-null   float64
 4   dropoff_latitude   9914 non-null   float64
 5   passenger_count    9914 non-null   int64
dtypes: datetime64[ns, UTC](1), float64(4), int64(1)
memory usage: 464.8 KB
```

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.
2. Passenger_count variable drop where passenger count less than 0 or greater than 6.
3. Latitudes range from -90 to 90.Longitudes range from -180 to 180. Removing which does not satisfy these ranges. And there is only one value and just drop it.

So, we lost 16067-15661 = 406 observations because of non-sensual values.

2.1.1 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

2.1.2. Outlier Analysis

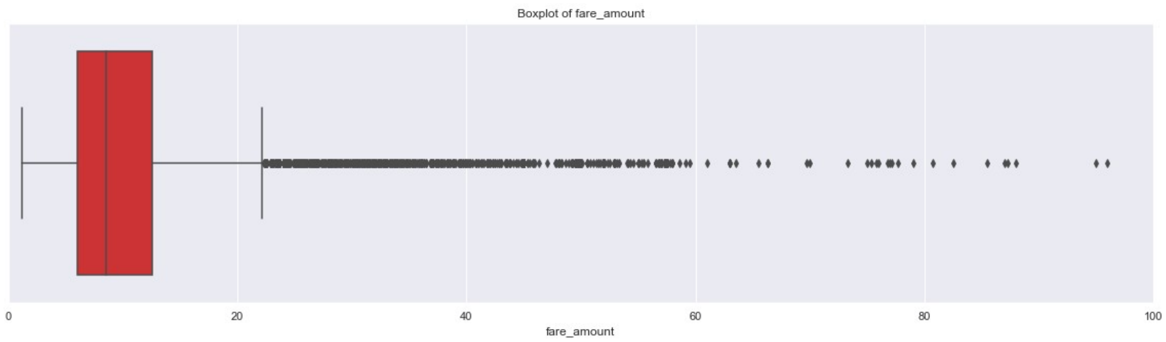
We look for outlier in the dataset by plotting Boxplots. There are outliers present in the data. We have removed these outliers. This is how we done,

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

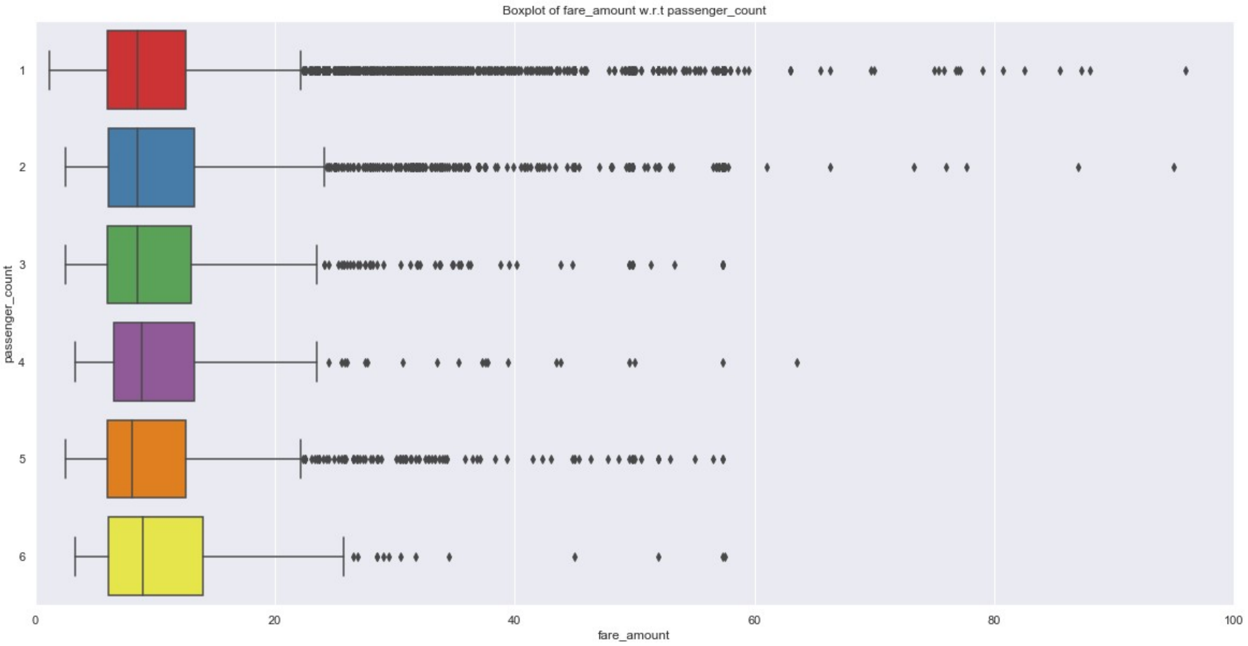
II. Then we imputed those missing values with KNN method.

- 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 1359 outliers. We successfully imputed these outliers with KNN and K value is 3

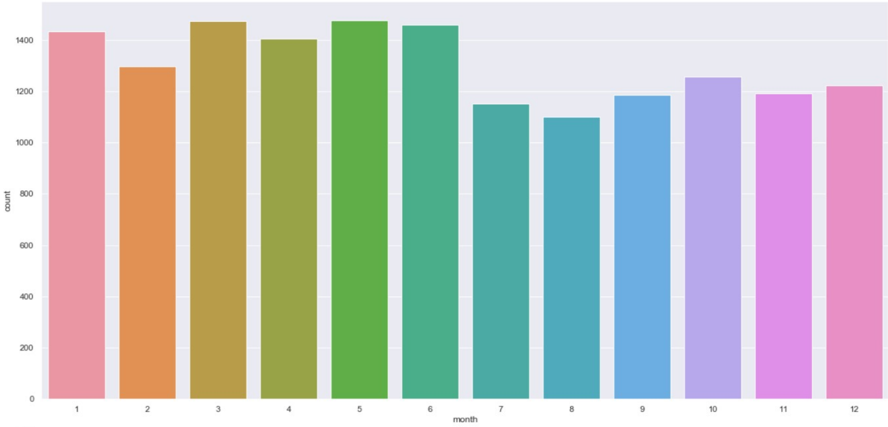
2.1.3. Feature Engineering

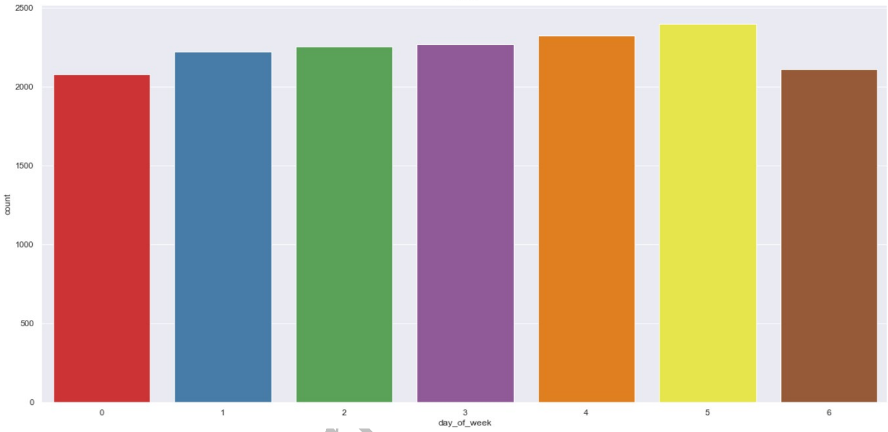
Feature Engineering is used to drive new features from existing features.

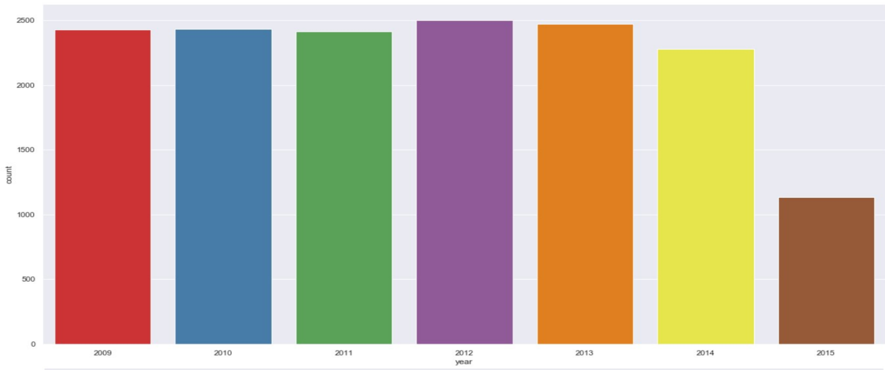
1. **<u>For ‘pickup_datetime’ variable</u>:**

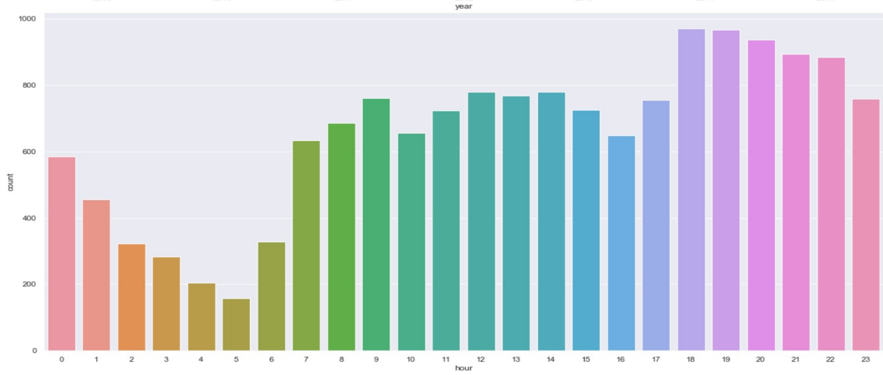
We will use this timestamp variable to create new variables.
New features will be year, month, day_of_week, and hour.
‘year’ will contain only years from pickup_datetime. For ex. 2009, 2010, 2011, etc.
‘month’ will contain only months from pickup_datetime. For ex. 1 for January, 2 for February, etc.
‘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.









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.

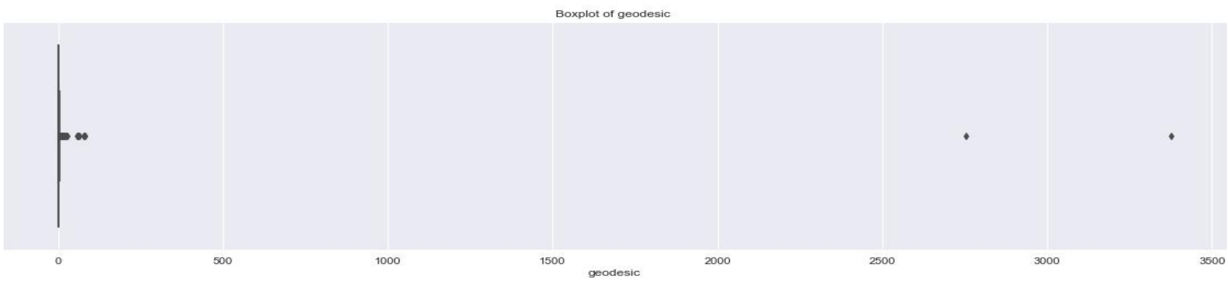
We will use both haversine and vincenty methods to calculate distance. For haversine, variable name will be ‘great_circle’ and for vincenty, new variable name will be ‘geodesic’.
As Vincenty is more accurate than haversine. Also, vincenty is prefered for short distances. Therefore, we will drop great_circle.
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')
```

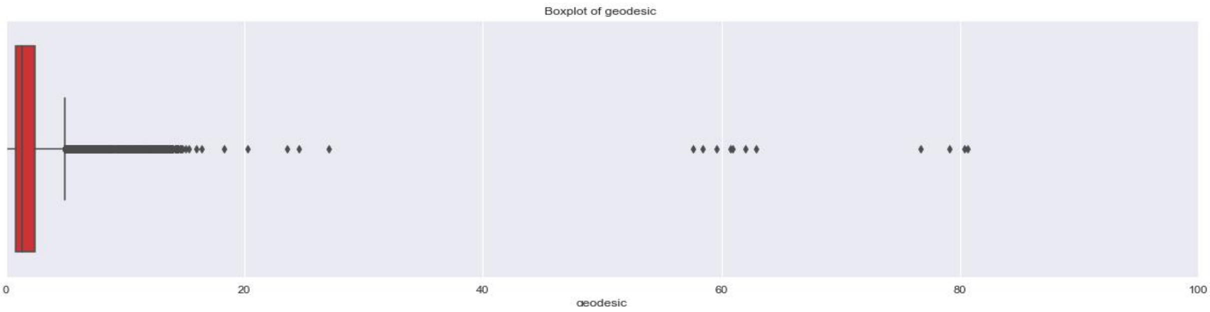
We will plot boxplot for our new variable ‘geodesic’:



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

Arun Kumar

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



We will treat these outliers like we previously did.

2.1.4. 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:

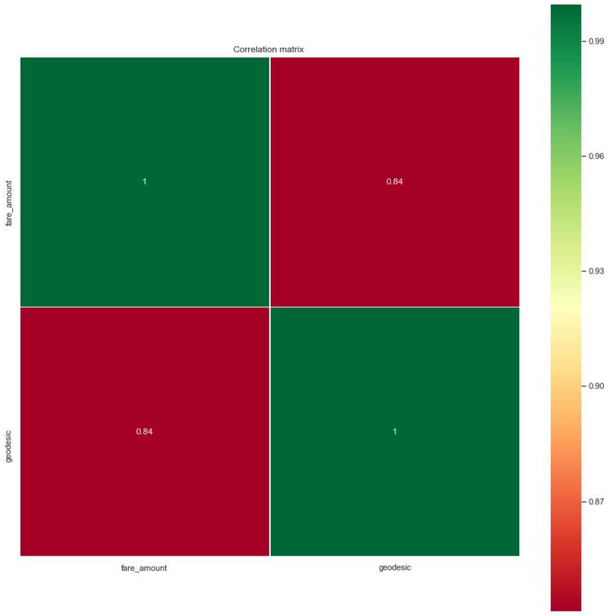
Correlation analysis – This requires only numerical variables. Therefore, we will filter out only numerical variables and feed it to correlation analysis. We do this by plotting correlation plot for all numerical variables. There should be no correlation between independent variables but there should 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:

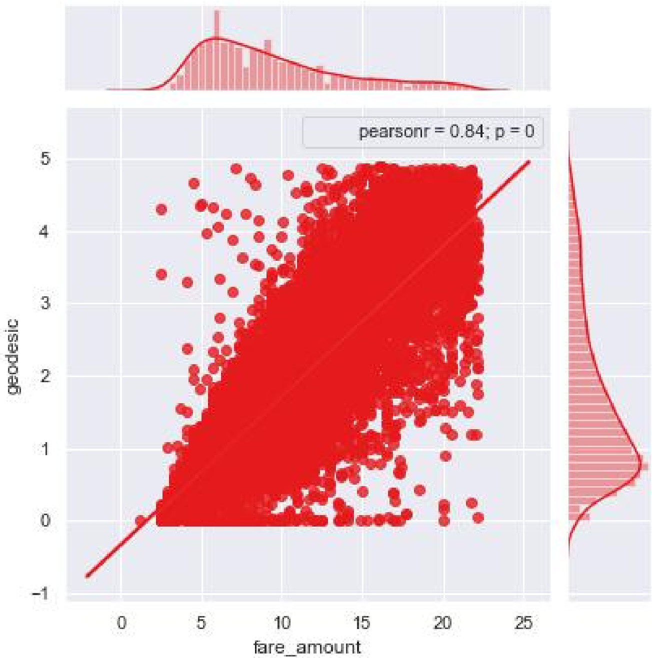
- 'fare_amount' and 'geodesic' are very highly correlated with each other.
- As fare_amount is the target variable and ‘geodesic’ is independent variable we will keep ‘geodesic’ because it will help to explain variation in fare_amount.

Arun Kumar

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

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

Here we did the test between categorical independent variables pairwise.

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

3 **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	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

Arun Kumar

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 error 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.

Arun Kumar

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

Arun Kumar

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

2.1.5. Feature Scaling

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

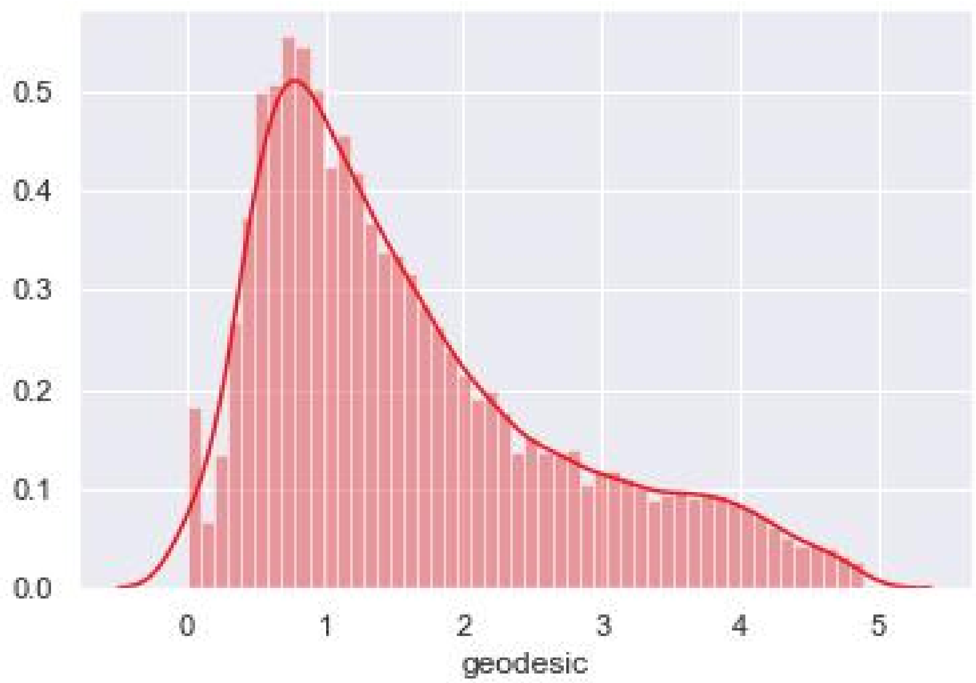
Normalization: Normalization refers to the dividing of a vector by its length. Normalization normalizes the data in the range of 0 to 1. It is generally used when we are planning to use distance method for our model development purpose such as KNN. Normalizing the data improves convergence of such algorithms. 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.

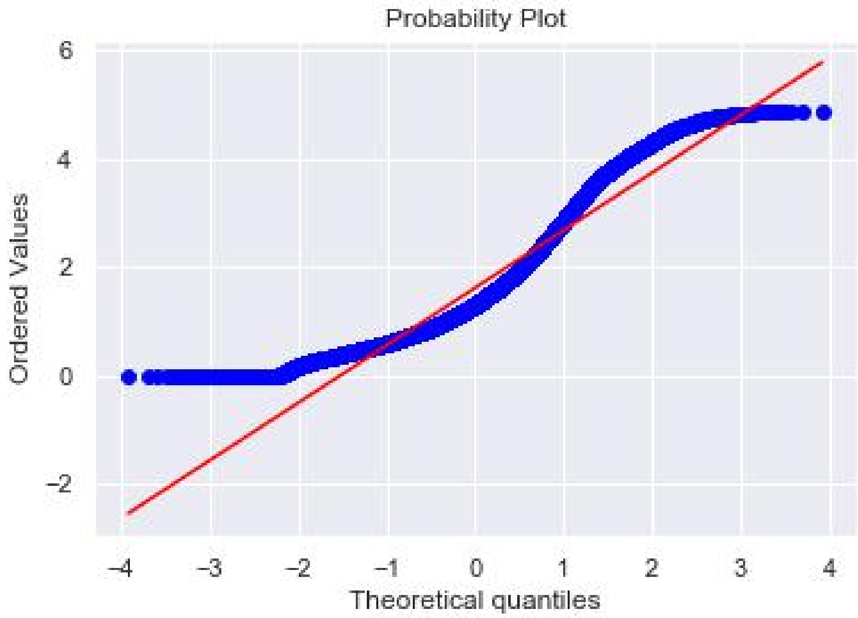
- We have checked variance for each column in dataset before Normalisation
- High variance will affect the accuracy of the model. So, we want to normalise that variance.

Graphs based on which standardization was chosen:
Note: It is performed only on Continuous variables.

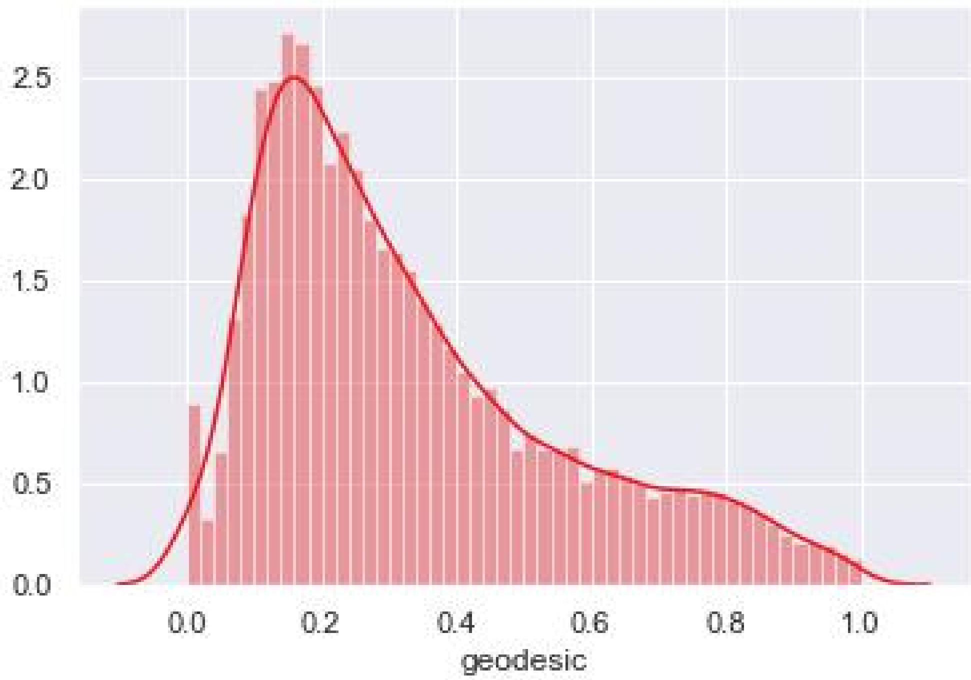
distplot() for 'geodesic' feature before normalization:



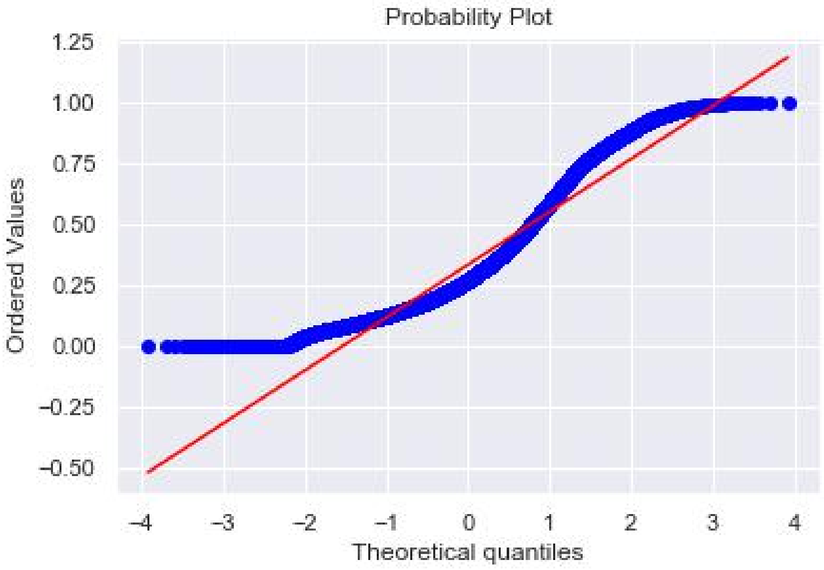
qq probability plot before normalization:



distplot() for ‘geodesic’ feature after normalization:



qq probability plot after normalization:



2.1.6. Data after EDA and pre-processing

Save this data after EDA and data pre-processing for further steps.

Arun Kumar

2.2. 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. Ridge Regression
III. Lasso Regression
IV. Decision Tree
V. Random Forest
VI. 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:

I. r square
II. Adjusted r square
III. MAPE(Mean Absolute Percentage Error)
IV. MSE(Mean square Error)
V. RMSE(Root Mean Square Error)
VI. RMSLE(Root Mean Squared Log Error)

Chapter 3

Conclusion

3.1. Model Evaluation

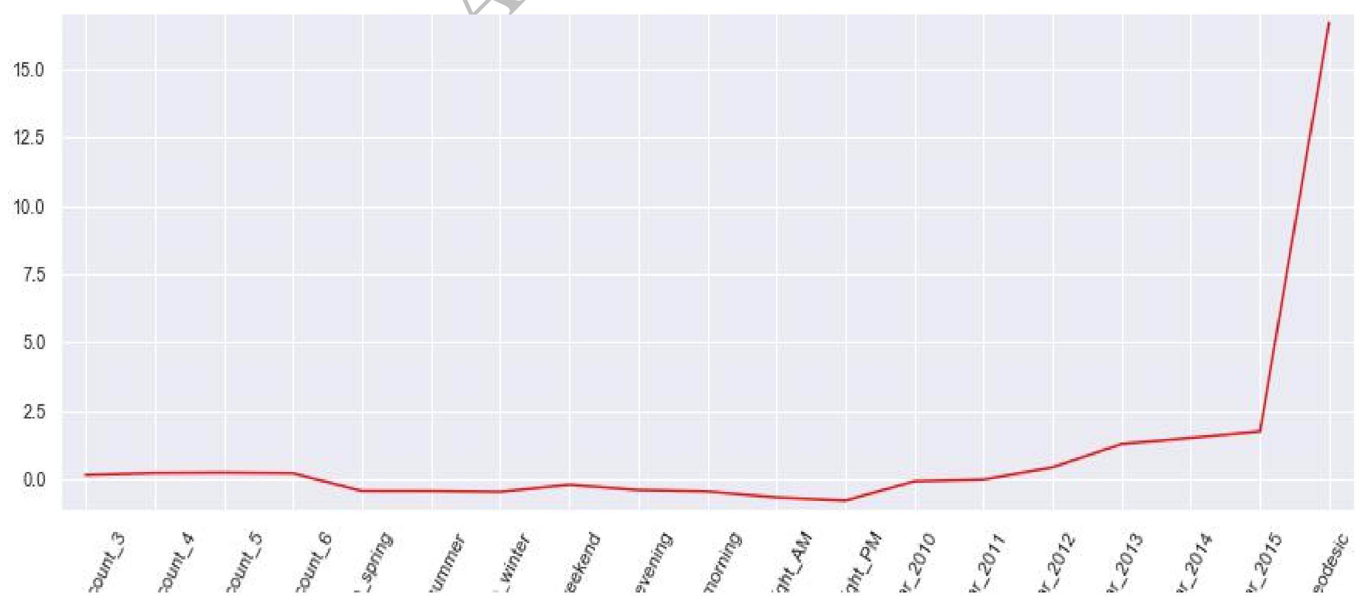
Here, we will evaluate the performance of different Regression models based on different Error Metrics

I. Multiple Linear Regression:

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

Arun Kumar

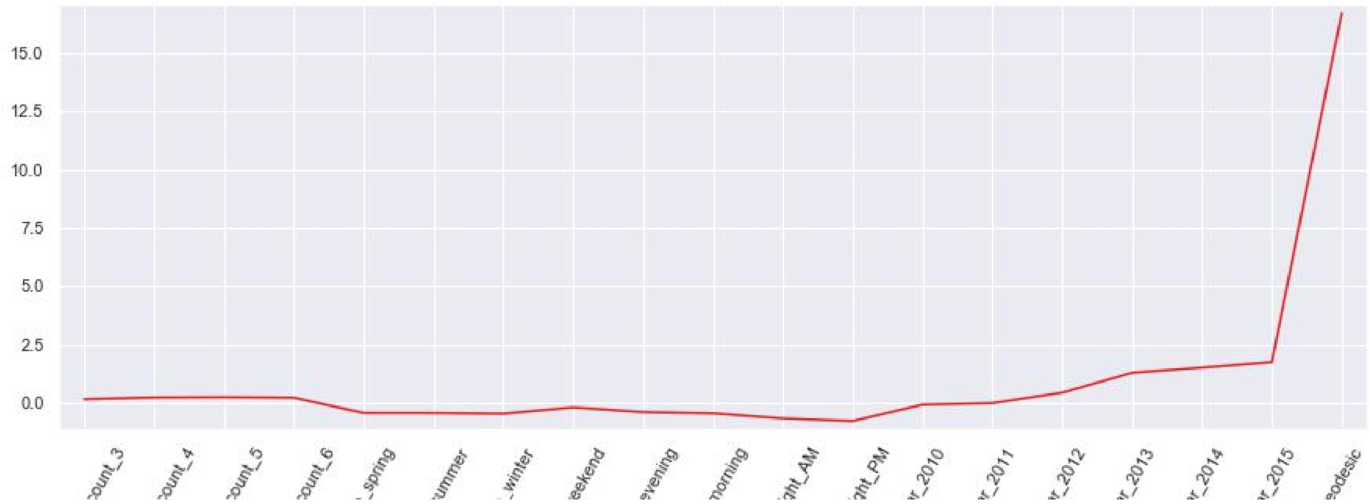
Line Plot for Coefficients of Multiple Linear regression:



II. Ridge Regression:

Error Metrics	r square	Adj r sq	MAPE	MSE	RMSE	RMSLE
Train	0.7343	0.733	18.74	5.28	2.29	0.21
validation	0.7419	0.7406	18.96	5.29	2.3	0.21

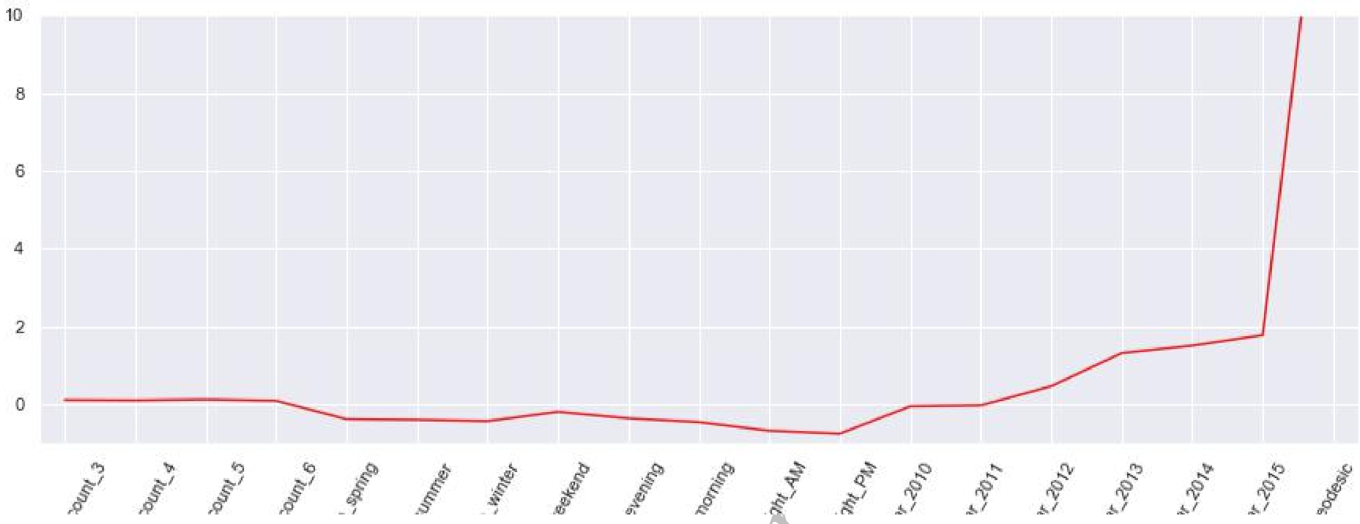
Line Plot for Coefficients of Ridge regression:



III. Lasso Regression:

Error Metrics	r square	Adj r sq	MAPE	MSE	RMSE	RMSLE
Train	0.7341	0.7337	18.75	5.28	2.29	0.21
Validation	0.7427	0.7415	18.95	5.27	2.29	0.21

Line Plot for Coefficients of Lasso regression:

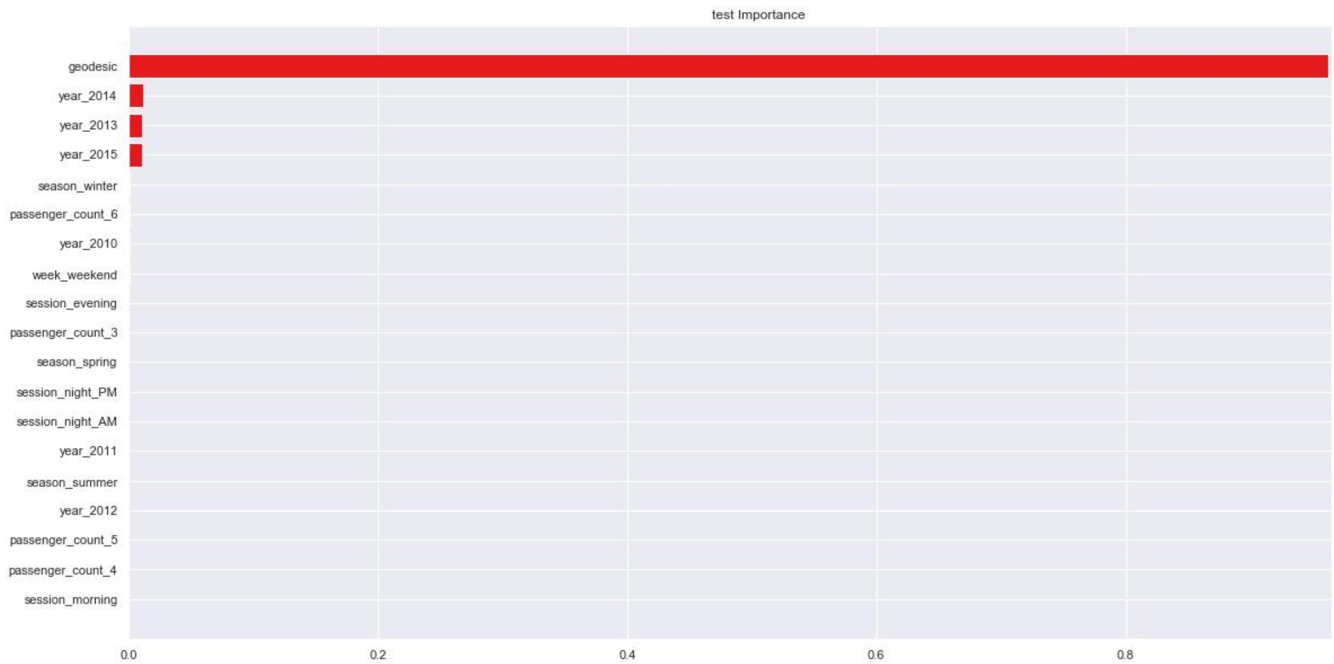


IV. Decision Tree Regression:

Error Metrics	r square	Adj r sq	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

Arun Kumar

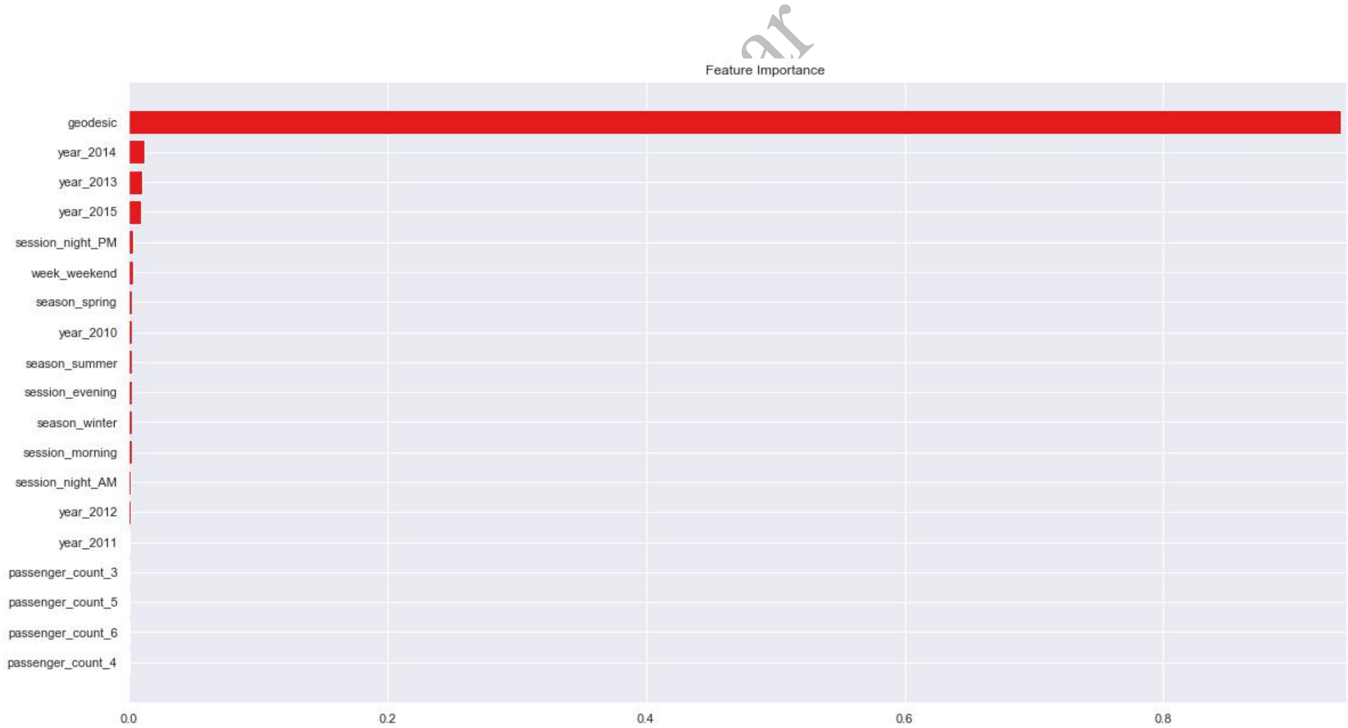
Bar Plot of Decision tree Feature Importance:



V. Random Forest Regression:

Error Metrics	r square	Adj r sq	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:



Improving accuracy

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

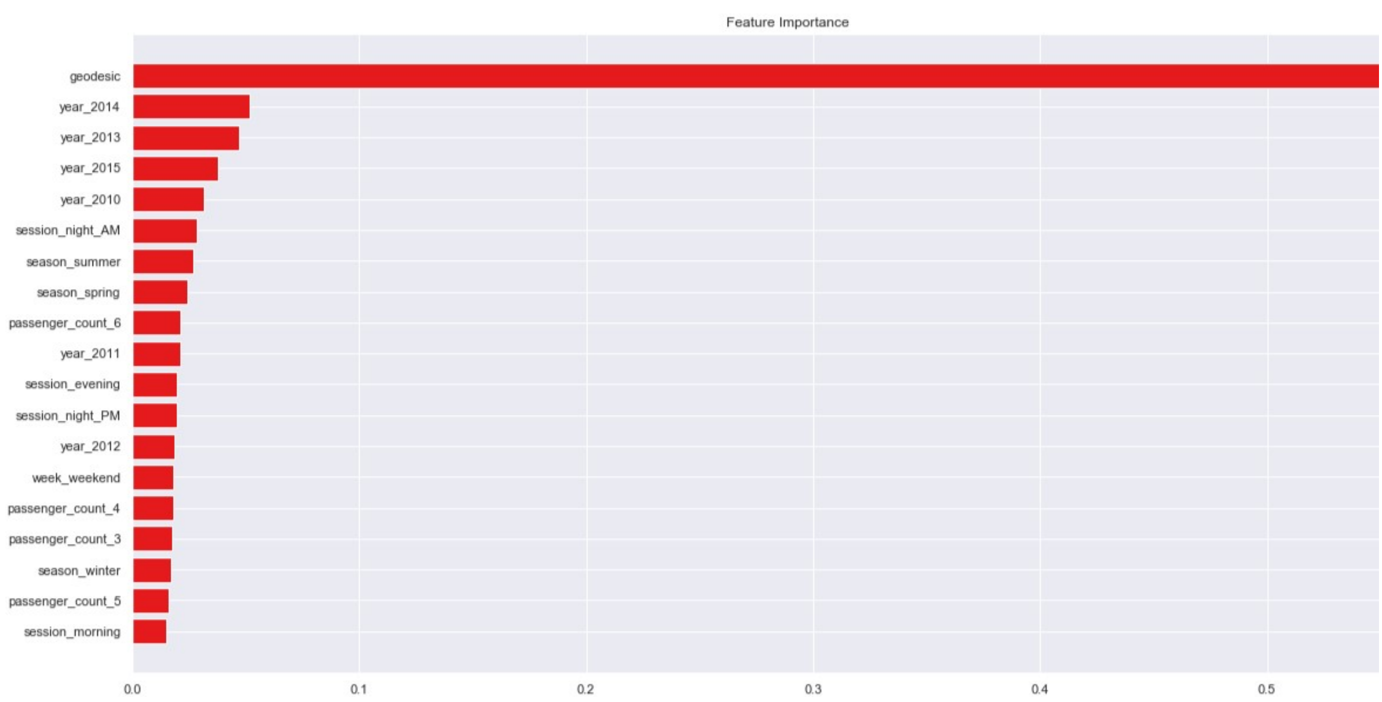
<u>Xgboost hyperparameters tuned parameters:</u>Tuned Xgboost Parameters: {'subsample': 0.1, 'reg_alpha': 0.08685113737513521, 'n_estimators': 200, 'max_depth': 3, 'learning_rate': 0.05, 'colsample_bytree': 0.7000000000000001, 'colsample_bynode': 0.7000000000000001, 'colsample_bylevel': 0.9000000000000001}

Xgboost Regression:

Error Metrics	r square	Adj r sq	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

Arun Kumar

Bar Plot of Xgboost Feature Importance:



Chapter 3

Conclusion

3.2. Model Selection

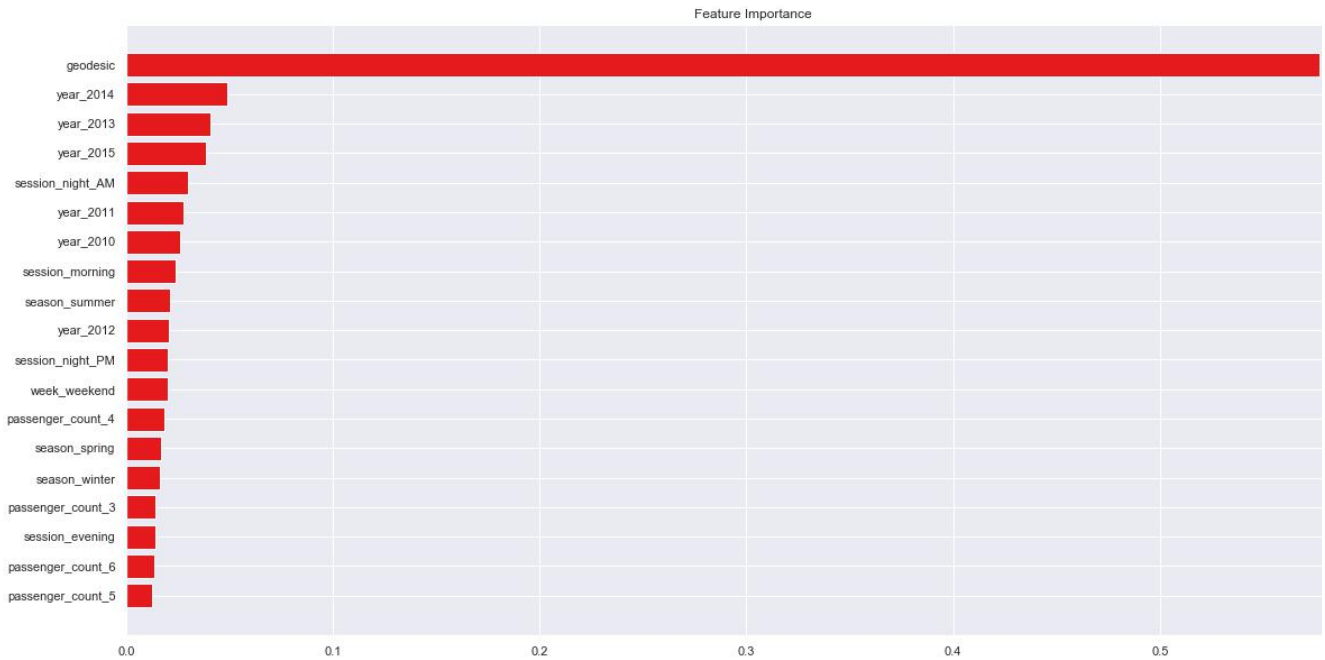
- Create standalone model on entire training dataset
- Save model for later use

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

```
<<<------------------- Test Data Score --------------------->

r square   0.7536990357805177
Adjusted r square:0.7521952702534707
MAPE:18.477615127632426
MSE: 5.044311731381777
RMSE: 2.24595452567094
RMSLE: 0.20634512637457236
```

Feature importance:



Chapter 4

Codes

4.1. 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 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.

# In[4]:


# 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


# In[5]:
```

```
#Get working dirctory
os.getcwd()


# In[6]:


# set the working directory
os.chdir('C:\\Users\\Arun Kumar\\Desktop\\cab fare prediction')


# 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

# In[7]:


# 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)


# In[8]:


train.info()


# In[9]:


test.head()
```

```
# In[10]:


test.info()


# In[11]:


test.describe()


# In[12]:


train.describe()


# ## EDA

# -  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.

# In[13]:


cat_var=['passenger_count']
num_var=['fare_amount','pickup_longitude','pickup_latitude','dropoff_longitude','dropoff_latitude']


# ## Graphical EDA - Data Visualization

# In[14]:


# setting up the sns for plots
sns.set(style='darkgrid',palette='Set1')


# Some histogram plots from seaborn library

# In[15]:


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

# In[16]:


# 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.

# In[17]:


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


# In[18]:


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


# In[19]:


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


# In[20]:


_ = 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

# In[21]:
```

```
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

# In[22]:


_ =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.

# In[23]:


sum(train['fare_amount']<1)


# In[24]:


train[train['fare_amount']<1]


# In[25]:


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


# In[26]:


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


# 2.Passenger_count variable

# In[27]:
```

```
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 consistenly above from 6,7,8,9,10
passenger_counts, let's check them.

# In[28]:


train[train['passenger_count']>6]


# Also we need to see if there are any passenger_count<1

# In[29]:


train[train['passenger_count']<1]


# In[30]:


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


# In[31]:


test['passenger_count'].unique()


# -  passenger_count variable conatins 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.

# In[32]:


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


# In[33]:


# train.loc[train['passenger_count'] >6,'passenger_count'] = np.nan
# train.loc[train['passenger_count'] >1,'passenger_count'] = np.nan


# In[34]:
```

```
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

# In[35]:


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.

# In[36]:


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.

# In[37]:


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)


# In[38]:


# 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
```

```
# In[39]:


train.shape


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

# In[40]:


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


# ## Missing Value Analysis

# In[41]:


#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.

# In[43]:


#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

# In[44]:


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

```
# In[45]:


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


# In[46]:


# 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

# In[47]:


# for i in
['fare_amount','pickup_longitude','pickup_latitude','dropoff_longitude','drop
off_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:{}\n'.format(train[i].fillna(train[i].median()).loc[1000]))


# In[48]:


# Choosing a random values to replace it as NA
a=train['fare_amount'].loc[1000]
print('fare_amount at loc-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]))


# In[49]:


train.std()


# In[50]:


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.

# In[51]:


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


# In[52]:


# 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)


# In[53]:


train.std()


# In[54]:


train.loc[1000]


# In[55]:


train['passenger_count'].head()


# In[56]:


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

```
# In[57]:


train.std()


# In[58]:


train['passenger_count'].unique()


# In[59]:


train['passenger_count']=pd.Categorical(train['passenger_count'],
categories=[1,2,3,4,5,6],ordered=True)


# In[60]:


train['passenger_count'].unique()


# In[61]:


train.loc[1000]


# -  Now about missing value in pickup_datetime

# In[62]:


pickup_datetime.head()


# In[63]:


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


# In[64]:


pickup_datetime.shape


# In[65]:


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.

# In[66]:


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


# In[67]:


train['passenger_count'].describe()


# In[68]:


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 laitudes and longitudes.

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

# In[69]:


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()


# In[70]:


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


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

# In[71]:


plt.figure(figsize=(20,10))
plt.xlim(0,100)
_ =
```

```
sns.boxplot(x=train['fare_amount'],y=train['passenger_count'],data=train,orie
nt='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()


# In[72]:


train.describe()


# In[73]:


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.

# In[74]:


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


# In[75]:


# 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')


# In[76]:


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


# In[77]:
```

```
train.std()


# In[78]:


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


# In[79]:


train.std()


# In[80]:


train['passenger_count'].describe()


# In[81]:


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


# In[82]:


train.describe()


# In[83]:


train.head()


# In[84]:


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


# In[85]:


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

# In[86]:


# we will Join 2 Dataframes pickup_datetime and train
train = pd.merge(pickup_datetime,train,right_index=True,left_index=True)
train.head()


# In[87]:


train.shape


# In[88]:


train=train.reset_index(drop=True)


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

# In[89]:


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


# In[90]:


train=train.dropna()


# In[91]:


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)


# In[92]:


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


# In[93]:


plt.figure(figsize=(20,10))
sns.countplot(train['year'])
```

```python
# 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

# In[94]:


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'


# In[95]:


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'


# In[96]:


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'
```

```
# In[97]:


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


# In[98]:


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


# In[99]:


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


# In[100]:


train.shape


# In[101]:


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.

# In[102]:


train['passenger_count'].describe()


# In[103]:


#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)


# In[104]:


train.head()


# In[105]:


test.head()


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

# In[106]:


train.columns


# In[107]:


train=train.drop(['passenger_count_1','season_fall','week_weekday','session_a
fternoon','year_2009'],axis=1)
test=test.drop(['passenger_count_1','season_fall','week_weekday','session_aft
ernoon','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.

# In[108]:


# train.sort_values('pickup_datetime')


# In[109]:


# 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


# In[110]:


#
haversine(['pickup_longitude','pickup_latitude'],['dropoff_longitude','dropof
f_latitude'])


# In[111]:


# 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)


# In[112]:


train.head()


# In[113]:


test.head()


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

# In[114]:


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

```
# In[115]:


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


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

# In[116]:


# 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)


# In[117]:


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)


# In[118]:


train.shape,test.shape


# In[119]:


# test_nodummies.columns


# In[120]:


# train_nodummies.columns


# In[121]:
```

```
train.columns


# In[122]:


test.columns


# In[123]:


train.head()


# In[124]:


test.head()


# In[125]:


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()


# In[126]:


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()


# In[127]:


outlier_treatment('geodesic')


# In[128]:


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


# In[129]:


#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.

# In[130]:


cat_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']
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.

# In[131]:


# 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.

# In[132]:


_ = 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

# In[133]:


#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.

# In[134]:


train.columns


# In[135]:


#ANOVA
_1)+C(passenger_count_2)+C(passenger_count_3)+C(passenger_count_4)+C(passenge
r_count_5)+C(passenger_count_6)
model = ols('fare_amount ~ 
C(passenger_count_2)+C(passenger_count_3)+C(passenger_count_4)+C(passenger_co
unt_5)+C(passenger_count_6)+C(season_spring)+C(season_summer)+C(season_winter
)+C(week_weekend)+C(session_night_AM)+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).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.

# In[136]:


#
_1+passenger_count_2+passenger_count_3+passenger_count_4+passenger_count_5+pa
ssenger_count_6
outcome, predictors = dmatrices('fare_amount ~
geodesic+passenger_count_2+passenger_count_3+passenger_count_4+passenger_coun
t_5+passenger_count_6+season_spring+season_summer+season_winter+week_weekend+
session_night_AM+session_night_PM+session_evening+session_morning+year_2010+y
ear_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 standarscalar

# In[137]:


train[num_var].var()


# In[138]:


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


# In[139]:


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


# In[140]:
```

```
#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']))


# In[141]:


train['geodesic'].var()


# In[142]:


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


# In[143]:


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


# In[144]:


train.columns


# In[145]:


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


# In[146]:


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


# In[147]:


train.columns


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

# In[149]:
```

```
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 = 0.20,
random_state=42)
print(train.shape, X_train.shape, X_test.shape,y_train.shape,y_test.shape)


# In[150]:


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

# In[151]:


# 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_))
```

```
# In[152]:


# 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()


# In[153]:


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

# In[154]:


# 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_))


# In[155]:


# 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

# In[156]:


# 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_))


# In[157]:
```

```
# Instantiate a lasso regressor: lasso
lasso = Lasso(alpha=0.0002120950887920190 5, 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

# In[159]:


# 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_))


# In[160]:


# 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

# In[163]:


# 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_))


# In[164]:


# 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)


# In[165]:


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
#
# - a) Algorithm Tuning
# - b) Ensembles

# In[166]:


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


# In[167]:


dtrain,dtest,data_dmatrix


# In[168]:


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()


# In[169]:


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


# In[170]:


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


# In[171]:


# 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_))


# In[172]:


# 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

# In[173]:


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))


# In[174]:


test.columns


# In[175]:


train.columns
```

```
# In[176]:


train.shape


# In[177]:


test.shape


# In[178]:


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


# In[179]:


test_pickup_datetime=a['pickup_datetime']


# In[180]:


# 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)


# In[181]:


pred_results_wrt_date


# In[182]:


# 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')


# In[ ]:
```

Arun Kumar

Chapter 4

Codes

4.2 R Code

```
# Cab Fare Prediction
#clear envernment
rm(list = ls())
#get working dir
getwd()
#set working dir
setwd("C:/Users/Arun Kumar/Desktop/cab fare prediction")
# #loading Libraries
x = c("ggplot2", "corrgram", "DMwR", "usdm", "caret", "randomForest", "e1071",
      "DataCombine", "doSNOW", "inTrees", "rpart.plot", "rpart",'MASS','xgboost','stats')
#load Packages
lapply(x, require, character.only = TRUE)
rm(x)

# 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.


# loading datasets
train = read.csv("train_cab.csv", header = T, na.strings = c(" ", "", "NA"))
test = read.csv("test.csv")
test_pickup_datetime = test["pickup_datetime"]
# Structure of data
str(train)
str(test)
summary(train)
summary(test)
head(train,5)
head(test,5)
```

Arun Kumar

```
#############                Exploratory Data Analysis               #######################
# Changing the data types of variables
train$fare_amount = as.numeric(as.character(train$fare_amount))
train$passenger_count=round(train$passenger_count)

### 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.
train[which(train$fare_amount < 1 ),]
nrow(train[which(train$fare_amount < 1 ),])
train = train[-which(train$fare_amount < 1 ),]

#2.Passenger_count variable
for (i in seq(4,11,by=1)){
  print(paste('passenger_count above ' ,i,nrow(train[which(train$passenger_count > i ),])))
  }
# so 20 observations of passenger_count is consistenly above from 6,7,8,9,10 passenger_counts, let's check them.
train[which(train$passenger_count > 6 ),]
# Also we need to see if there are any passenger_count==0
train[which(train$passenger_count <1 ),]
nrow(train[which(train$passenger_count <1 ),])
# We will remove these 58 observations and 20 observation which are above 6 value because a cab cannot hold these number of passengers.
train = train[-which(train$passenger_count < 1 ),]
train = train[-which(train$passenger_count > 6),]
# 3.Latitudes range from -90 to 90.Longitudes range from -180 to 180.Removing which does not satisfy these ranges
print(paste('pickup_longitude above 180=',nrow(train[which(train$pickup_longitude >180 ),])))
print(paste('pickup_longitude above -180=',nrow(train[which(train$pickup_longitude < -180 ),])))
print(paste('pickup_latitude above 90=',nrow(train[which(train$pickup_latitude > 90 ),])))
print(paste('pickup_latitude above -90=',nrow(train[which(train$pickup_latitude < -90 ),])))
print(paste('dropoff_longitude above 180=',nrow(train[which(train$dropoff_longitude > 180 ),])))
print(paste('dropoff_longitude above -180=',nrow(train[which(train$dropoff_longitude < -180 ),])))
print(paste('dropoff_latitude above -90=',nrow(train[which(train$dropoff_latitude < -90 ),])))
print(paste('dropoff_latitude above 90=',nrow(train[which(train$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.
nrow(train[which(train$pickup_longitude == 0 ),])
nrow(train[which(train$pickup_latitude == 0 ),])
nrow(train[which(train$dropoff_longitude == 0 ),])
nrow(train[which(train$pickup_latitude == 0 ),])
# there are values which are equal to 0. we will remove them.
train = train[-which(train$pickup_latitude > 90),]
train = train[-which(train$pickup_longitude == 0),]
```

Arun Kumar

```
train = train[-which(train$dropoff_longitude == 0),]

# Make a copy
df=train
# train=df

#############                    Missing Value Analysis              #############
missing_val = data.frame(apply(train,2,function(x){sum(is.na(x))}))
missing_val$Columns = row.names(missing_val)
names(missing_val)[1] =  "Missing_percentage"
missing_val$Missing_percentage = (missing_val$Missing_percentage/nrow(train)) * 100
missing_val = missing_val[order(-missing_val$Missing_percentage),]
row.names(missing_val) = NULL
missing_val = missing_val[,c(2,1)]
missing_val

unique(train$passenger_count)
unique(test$passenger_count)
train[,'passenger_count'] = factor(train[,'passenger_count'], labels=(1:6))
test[,'passenger_count'] = factor(test[,'passenger_count'], labels=(1:6))
# 1.For Passenger_count:
# Actual value = 1
# Mode = 1
# KNN = 1
train$passenger_count[1000]
train$passenger_count[1000] = NA
getmode <- function(v) {
  uniqv <- unique(v)
  uniqv[which.max(tabulate(match(v, uniqv)))]
}

# Mode Method
getmode(train$passenger_count)
# We can't use mode method because data will be more biased towards passenger_count=1

# 2.For fare_amount:
# Actual value = 18.1,
# Mean = 15.117,
# Median = 8.5,
# KNN = 18.28
sapply(train, sd, na.rm = TRUE)
# fare_amount   pickup_datetime  pickup_longitude
# 435.968236       4635.700531          2.659050
# pickup_latitude dropoff_longitude  dropoff_latitude
# 2.613305          2.710835          2.632400
# passenger_count
# 1.266104
train$fare_amount[1000]
train$fare_amount[1000]= NA
```

Arun Kumar

```
# Mean Method
mean(train$fare_amount, na.rm = T)

#Median Method
median(train$fare_amount, na.rm = T)

# kNN Imputation
train = knnImputation(train, k = 181)
train$fare_amount[1000]
train$passenger_count[1000]
sapply(train, sd, na.rm = TRUE)
# fare_amount   pickup_datetime  pickup_longitude
# 435.661952       4635.700531         2.659050
# pickup_latitude dropoff_longitude  dropoff_latitude
# 2.613305         2.710835         2.632400
# passenger_count
# 1.263859
sum(is.na(train))
str(train)
summary(train)

df1=train
# train=df1
######################                    Outlier Analysis                ##################

# We Will do Outlier Analysis only on Fare_amount just for now and we will do outlier analysis
      after feature engineering laitudes and longitudes.
# Boxplot for fare_amount
pl1 = ggplot(train,aes(x = factor(passenger_count),y = fare_amount))
pl1 + geom_boxplot(outlier.colour="red", fill = "grey" ,outlier.shape=18,outlier.size=1,
      notch=FALSE)+ylim(0,100)

# Replace all outliers with NA and impute
vals = train[,"fare_amount"] %in% boxplot.stats(train[,"fare_amount"])$out
train[which(vals),"fare_amount"] = NA

#lets check the NA's
sum(is.na(train$fare_amount))

#Imputing with KNN
train = knnImputation(train,k=3)

# lets check the missing values
sum(is.na(train$fare_amount))
str(train)

df2=train
# train=df2
##################         Feature Engineering                 ###########################
# 1.Feature Engineering for timestamp variable
```

Arun Kumar

```
# we will derive new features from pickup_datetime variable
# new features will be year,month,day_of_week,hour
#Convert pickup_datetime from factor to date time
train$pickup_date = as.Date(as.character(train$pickup_datetime))
train$pickup_weekday = as.factor(format(train$pickup_date,"%u"))# Monday = 1
train$pickup_mnth = as.factor(format(train$pickup_date,"%m"))
train$pickup_yr = as.factor(format(train$pickup_date,"%Y"))
pickup_time = strptime(train$pickup_datetime,"%Y-%m-%d %H:%M:%S")
train$pickup_hour = as.factor(format(pickup_time,"%H"))

#Add same features to test set
test$pickup_date = as.Date(as.character(test$pickup_datetime))
test$pickup_weekday = as.factor(format(test$pickup_date,"%u"))# Monday = 1
test$pickup_mnth = as.factor(format(test$pickup_date,"%m"))
test$pickup_yr = as.factor(format(test$pickup_date,"%Y"))
pickup_time = strptime(test$pickup_datetime,"%Y-%m-%d %H:%M:%S")
test$pickup_hour = as.factor(format(pickup_time,"%H"))

sum(is.na(train))# there was 1 'na' in pickup_datetime which created na's in above feature
      engineered variables.
train = na.omit(train) # we will remove that 1 row of na's

train = subset(train,select = -c(pickup_datetime,pickup_date))
test = subset(test,select = -c(pickup_datetime,pickup_date))
# Now we will use month,weekday,hour to derive new features like sessions in a day,seasons in
      a year,week:weekend/weekday
# f = function(x){
#   if ((x >=5)& (x <= 11)){
#     return ('morning')
#   }
#   if ((x >=12) & (x <= 16)){
#     return ('afternoon')
#   }
#   if ((x >=17) & (x <= 20)){
#     return ('evening')
#   }
#   if ((x >=21) & (x <= 23)){
#     return ('night (PM)')
#   }
#   if ((x >=0) & (x <= 4)){
#     return ('night (AM)')
#   }
# }
# 2.Calculate the distance travelled using longitude and latitude
deg_to_rad = function(deg){
  (deg * pi) / 180
}
haversine = function(long1,lat1,long2,lat2){
  #long1rad = deg_to_rad(long1)
  phi1 = deg_to_rad(lat1)
```

Arun Kumar

```
  #long2rad = deg_to_rad(long2)
  phi2 = deg_to_rad(lat2)
  delphi = deg_to_rad(lat2 - lat1)
  dellamda = deg_to_rad(long2 - long1)

  a = sin(delphi/2) * sin(delphi/2) + cos(phi1) * cos(phi2) *
    sin(dellamda/2) * sin(dellamda/2)

  c = 2 * atan2(sqrt(a),sqrt(1-a))
  R = 6371e3
  R * c / 1000 #1000 is used to convert to meters
}
# Using haversine formula to calculate distance fr both train and test
train$dist =
        haversine(train$pickup_longitude,train$pickup_latitude,train$dropoff_longitude,train$dr
        opoff_latitude)
test$dist =
        haversine(test$pickup_longitude,test$pickup_latitude,test$dropoff_longitude,test$dropof
        f_latitude)

# We will remove the variables which were used to feature engineer new variables
train = subset(train,select = -
        c(pickup_longitude,pickup_latitude,dropoff_longitude,dropoff_latitude))
test = subset(test,select = -
        c(pickup_longitude,pickup_latitude,dropoff_longitude,dropoff_latitude))

str(train)
summary(train)

################                    Feature selection             ###################
numeric_index = sapply(train,is.numeric) #selecting only numeric

numeric_data = train[,numeric_index]

cnames = colnames(numeric_data)
#Correlation analysis for numeric variables
corrgram(train[,numeric_index],upper.panel=panel.pie, main = "Correlation Plot")

#ANOVA for categorical variables with target numeric variable

#aov_results = aov(fare_amount ~ passenger_count * pickup_hour * pickup_weekday,data =
        train)
aov_results = aov(fare_amount ~ passenger_count + pickup_hour + pickup_weekday +
        pickup_mnth + pickup_yr,data = train)

summary(aov_results)

# pickup_weekdat has p value greater than 0.05
train = subset(train,select=-pickup_weekday)
```

Arun Kumar

```
#remove from test set
test = subset(test,select=-pickup_weekday)

######################## 	Feature Scaling 	################################
#Normality check
# qqnorm(train$fare_amount)
# histogram(train$fare_amount)
library(car)
# dev.off()
par(mfrow=c(1,2))
qqPlot(train$fare_amount)                           # qqPlot, it has a x values derived from gaussian
          distribution, if data is distributed normally then the sorted data points should lie very
          close to the solid reference line
truehist(train$fare_amount)                         # truehist() scales the counts to give an estimate of
          the probability density.
lines(density(train$fare_amount))  # Right skewed     # lines() and density() functions to overlay
          a density plot on histogram

#Normalisation

print('dist')
train[,'dist'] = (train[,'dist'] - min(train[,'dist']))/
   (max(train[,'dist'] - min(train[,'dist'])))

# #check multicollearity
# library(usdm)
# vif(train[,-1])
#
# vifcor(train[,-1], th = 0.9)

#################### Splitting train into train and validation subsets ###################
set.seed(1000)
tr.idx = createDataPartition(train$fare_amount,p=0.75,list = FALSE) # 75% in trainin and 25%
          in Validation Datasets
train_data = train[tr.idx,]
test_data = train[-tr.idx,]

rmExcept(c("test","train","df",'df1','df2','df3','test_data','train_data','test_pickup_datetime'))
###################Model Selection################
#Error metric used to select model is RMSE

############# 	Linear regression 	#################
lm_model = lm(fare_amount ~.,data=train_data)

summary(lm_model)
str(train_data)
plot(lm_model$fitted.values,rstandard(lm_model),main = "Residual plot",
    xlab = "Predicted values of fare_amount",
    ylab = "standardized residuals")
```

Arun Kumar

```
lm_predictions = predict(lm_model,test_data[,2:6])

qplot(x = test_data[,1], y = lm_predictions, data = test_data, color = I("blue"), geom = "point")

regr.eval(test_data[,1],lm_predictions)
# mae        mse       rmse       mape
# 3.5303114 19.3079726  4.3940838  0.4510407


#############                    Decision Tree          #####################

Dt_model = rpart(fare_amount ~ ., data = train_data, method = "anova")

summary(Dt_model)
#Predict for new test cases
predictions_DT = predict(Dt_model, test_data[,2:6])

qplot(x = test_data[,1], y = predictions_DT, data = test_data, color = I("blue"), geom = "point")

regr.eval(test_data[,1],predictions_DT)
# mae       mse       rmse      mape
# 1.8981592 6.7034713 2.5891063 0.2241461


#############                    Random forest          #####################
rf_model = randomForest(fare_amount ~.,data=train_data)

summary(rf_model)

rf_predictions = predict(rf_model,test_data[,2:6])

qplot(x = test_data[,1], y = rf_predictions, data = test_data, color = I("blue"), geom = "point")

regr.eval(test_data[,1],rf_predictions)
# mae       mse       rmse      mape
# 1.9053850 6.3682283 2.5235349 0.2335395

######## Improving Accuracy by using Ensemble technique ---- XGBOOST    ############
train_data_matrix = as.matrix(sapply(train_data[-1],as.numeric))
test_data_data_matrix = as.matrix(sapply(test_data[-1],as.numeric))

xgboost_model = xgboost(data = train_data_matrix,label = train_data$fare_amount,nrounds =
        15,verbose = FALSE)

summary(xgboost_model)
xgb_predictions = predict(xgboost_model,test_data_data_matrix)

qplot(x = test_data[,1], y = xgb_predictions, data = test_data, color = I("blue"), geom = "point")
```

Arun Kumar

```
regr.eval(test_data[,1],xgb_predictions)
# mae       mse      rmse      mape
# 1.6183415 5.1096465 2.2604527 0.1861947

############# Finalizing and Saving Model for later use            ##############
# In this step we will train our model on whole training Dataset and save that model for later use
train_data_matrix2 = as.matrix(sapply(train[-1],as.numeric))
test_data_matrix2 = as.matrix(sapply(test,as.numeric))

xgboost_model2 = xgboost(data = train_data_matrix2,label = train$fare_amount,nrounds =
          15,verbose = FALSE)

# Saving the trained model
saveRDS(xgboost_model2, "./final_Xgboost_model_using_R.rds")

# loading the saved model
super_model <- readRDS("./final_Xgboost_model_using_R.rds")
print(super_model)

# Lets now predict on test dataset
xgb = predict(super_model,test_data_matrix2)

xgb_pred = data.frame(test_pickup_datetime,"predictions" = xgb)

# Now lets write(save) the predicted fare_amount in disk as .csv format
write.csv(xgb_pred,"xgb_predictions_R.csv",row.names = FALSE)
```

Arun Kumar

Chapter 5

Reference

https://stackoverflow.com/
https://www.kaggle.com/
https://www.edwisor.com/

Arun Kumar