Cab Fare Prediction

Problem Statement:

You are a cab rental start-up company. You have successfully run the pilot project and now want to launch your cab service across the country.You have collected the historical data from your pilot project and now have a requirement to apply analytics for fare prediction.You need to design a system that predicts the fare amount for a cab ride in the city.

Data :

The Dataset has 7 variables namely

1.pickup_datetime
2.pickup_longitude
3.pickup_latitude
4.dropoff_longitude
5.dropoff_latitude
6.Passenger_count
7.fare_amount
Missing Values :Yes.

Explanation : The project is about predicting the fare amount for a cab ride in the city based on the historical data that is collected all across the country.

Steps involved in developing an analytics model for a dataset:

1.Importing the required libraries

```
import os
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

2. Importing both the training and test dataset

```
train = pd.read_csv('train_cab.csv')
test = pd.read_csv('test.csv')
data=[train,test]
```

3. Finding the mean median and std of both train and test data

```
train.describe()
 test.describe()
```

-

Exploratory Data analysis

1.Fare_amount variable has values in negative.we will drop those rows as the values cannot be negative or 0.

```
sum(train['fare_amount']<1)
 # Drop the fare amount rows whose fare amount is less than 1
train = train.drop(train[train['fare_amount']<1].index, axis=0)
```

2.Check if the passenger count has < 1 passenger in any of its row

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

3. We will remove all the passenger count rows whose values is above 6 as the cab cannot hold these number of passengers

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

4.Latitude ranges from -90 to 90 and longitude ranges from -180 to 180

```
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)))
```

5.**Check if there are any longitudes and latitudes whose value is 0**

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

6. **Drop the latitude and longitude values whose values are out of range**

```
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)
```

7. Convert fare_amount from object to numeric

```
train["fare_amount"] = pd.to_numeric(train["fare_amount"],errors = "coerce")
```

8. We will remove the rows having fare amounting more that 454 as considering them as outliers

```
train = train.drop(train[train["fare_amount"]> 454 ].index, axis=0)
Train.shape
```

9.test["pickup_datetime"]=pd.to_datetime(test["pickup_datetime"],format="%Y-%m-%d%H:%M:%S UTC")

10.train = train.drop(train[train['fare_amount'].isnull()].index, axis=0)

```
print(train.shape)
print(train['fare_amount'].isnull().sum())
```

11.**We will separate the Pickup_datetime column into separate field like year, month, day of the week**

```
train['year'] = train['pickup_datetime'].dt.year
```

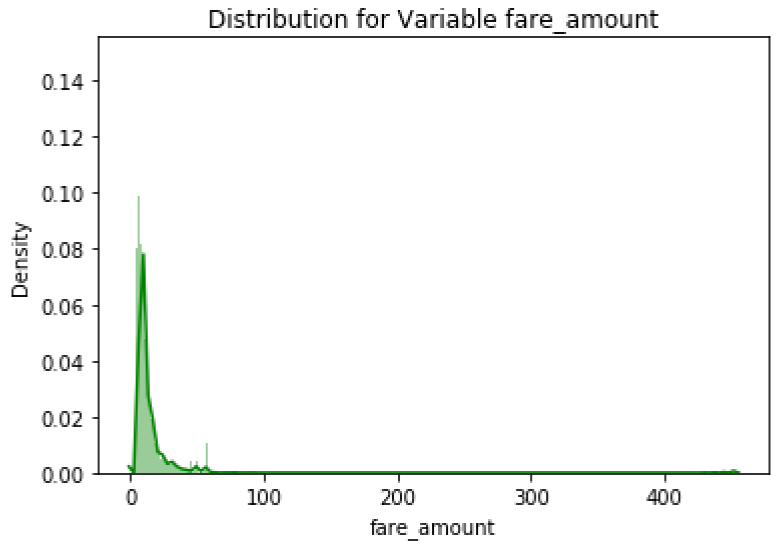
```
train['Month'] = train['pickup_datetime'].dt.month
train['Date'] = train['pickup_datetime'].dt.day
train['Day'] = train['pickup_datetime'].dt.dayofweek
train['Hour'] = train['pickup_datetime'].dt.hour
train['Minute'] = train['pickup_datetime'].dt.minute
```

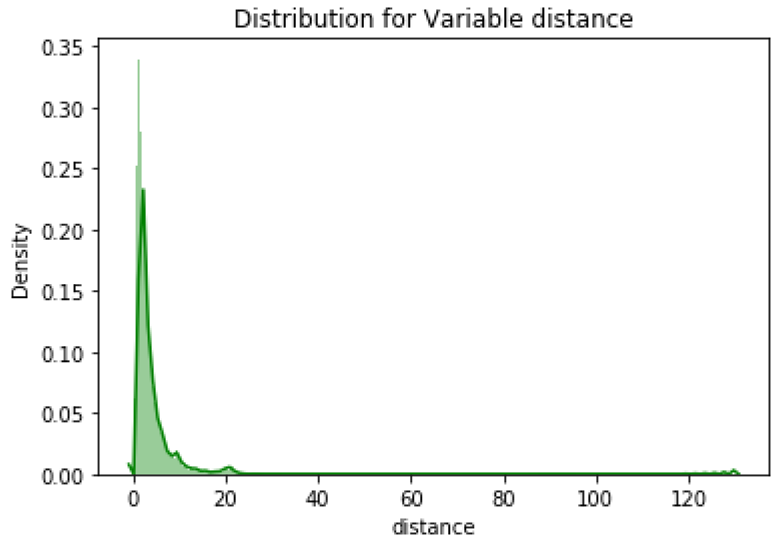
12.We will separate the Pickup_datetime column into separate field like year, month, day of the week

```
test['year'] = test['pickup_datetime'].dt.year
test['Month'] = test['pickup_datetime'].dt.month
test['Date'] = test['pickup_datetime'].dt.day
test['Day'] = test['pickup_datetime'].dt.dayofweek
test['Hour'] = test['pickup_datetime'].dt.hour
test['Minute'] = test['pickup_datetime'].dt.minute
```

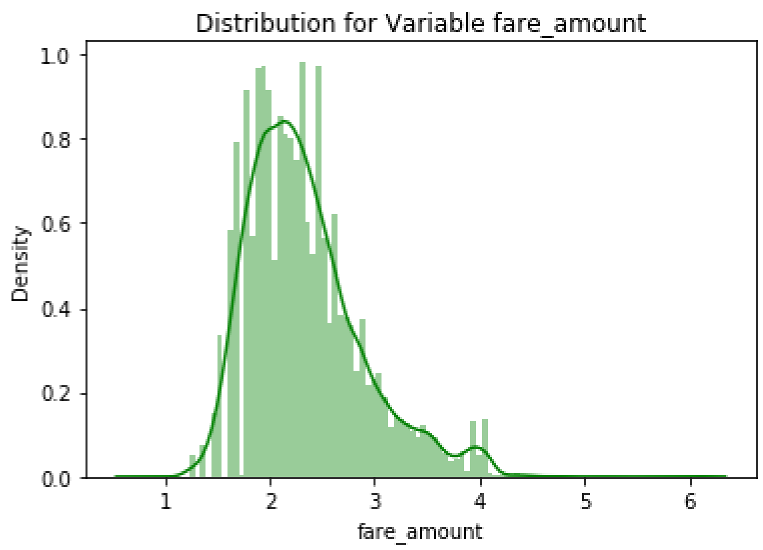
Feature Scaling:

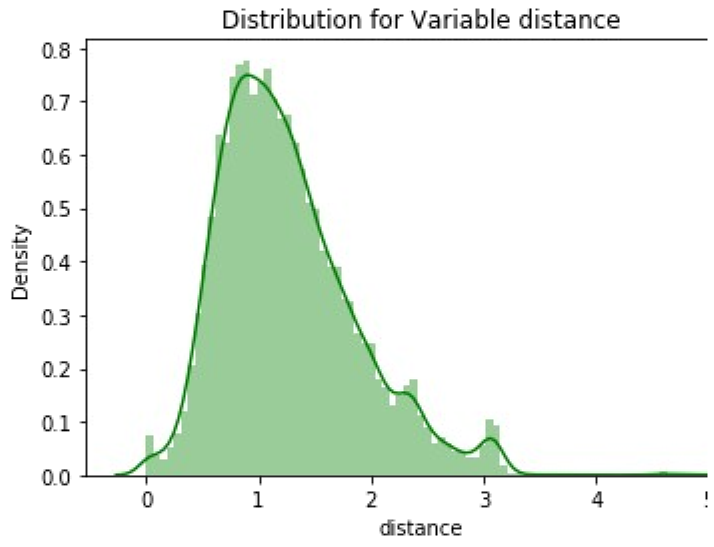
Below mentioned graphs shows the probability distribution plot to check distribution before log transformation:





Below mentioned graphs shows the probability distribution plot to check distribution after l
transformation:

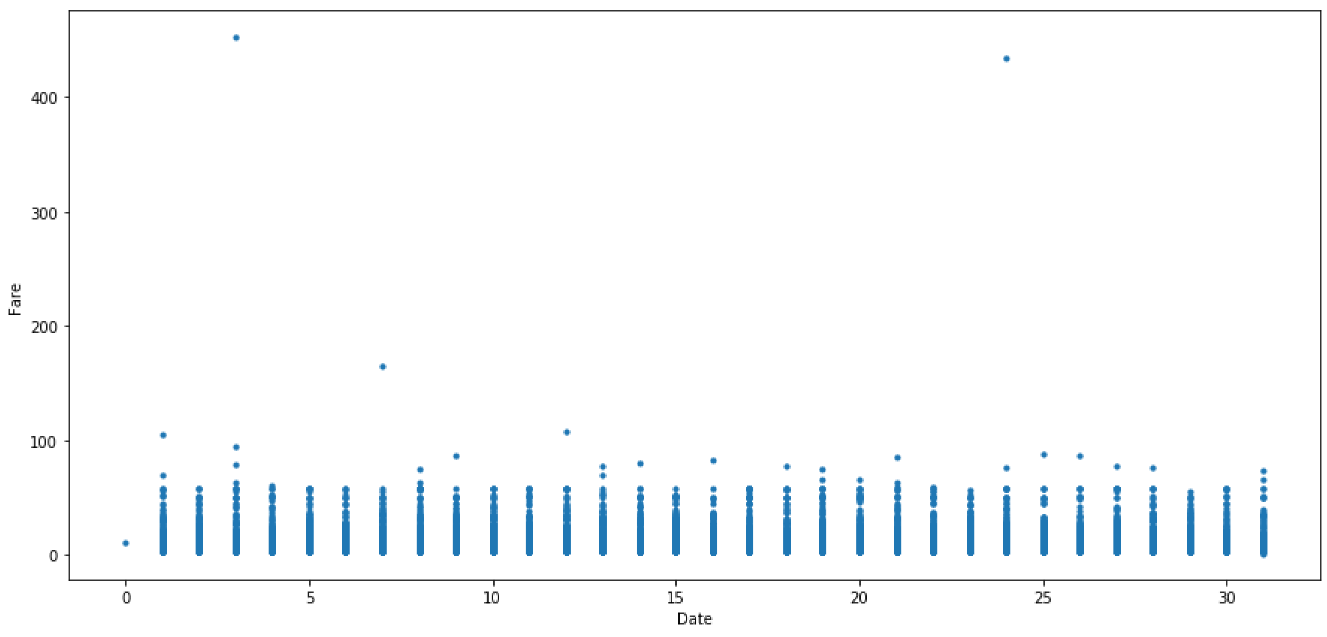




Visualisation :

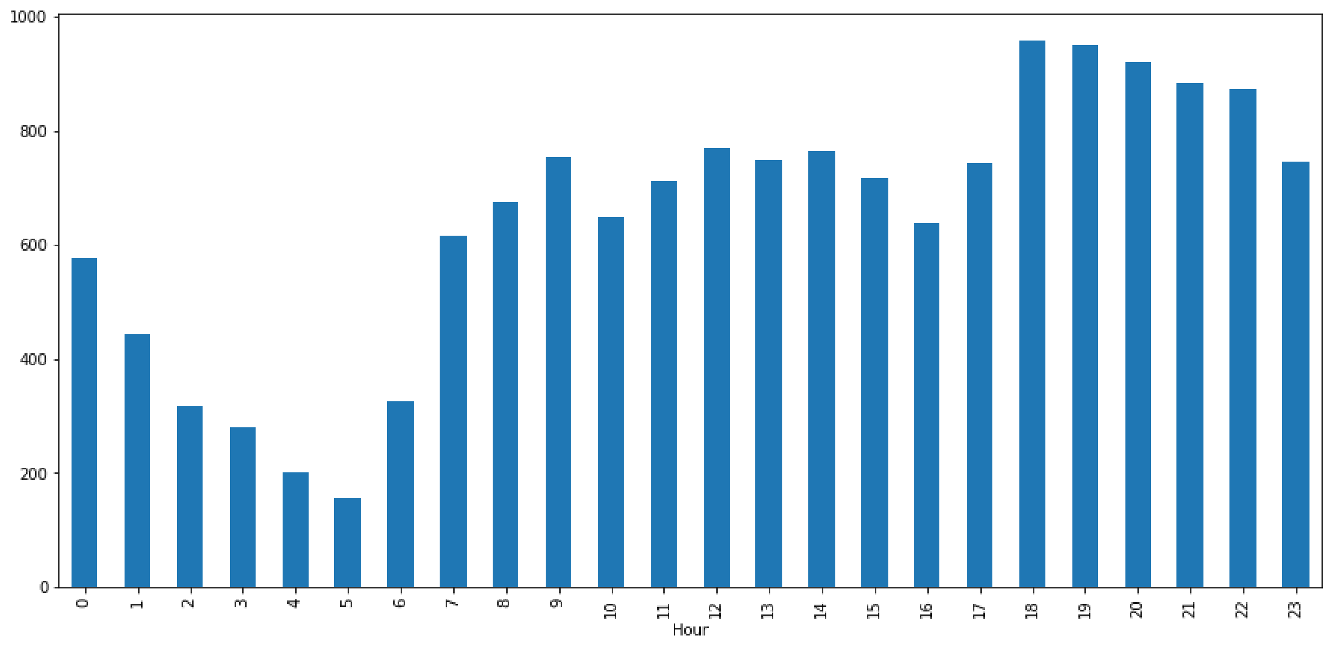
Date of month and fares

The fares throughout the month mostly seem uniform.



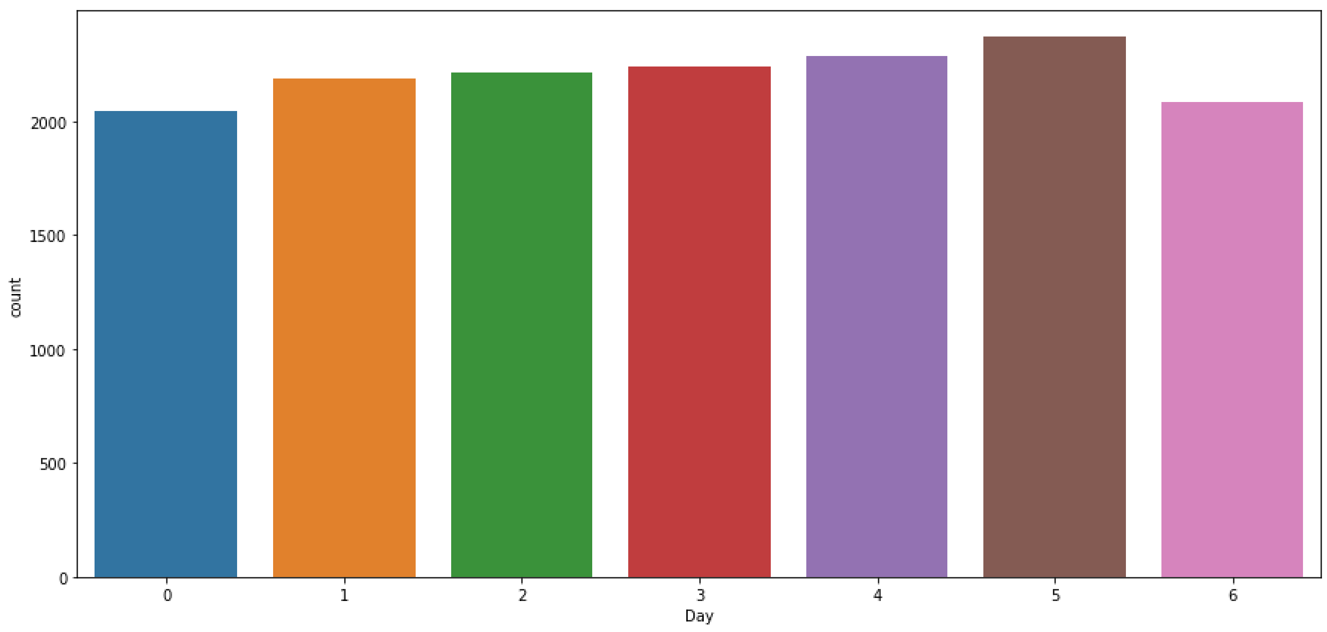
2.Hours and Fares

Lowest cabs at 5 AM and highest at and around 7 PM i.e the office rush hours



3. Impact of Day on the Number of Cab rides :

The day of the week does not seem to have much influence on the number of cabs ride



Before running any model, we will split our data into two parts which is train and test data. Here in our case we have taken 80% of the data as our train data.

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split( train.iloc[:, train.columns != 'fare_amount'],
                        train.iloc[:, 0], test_size = 0.20, random_state = 1)
```

Modelling: After thorough EDA process,The following models are built on the processed data

Multiple Linear Regression
RMSE =0.28
R^2 score =73%

Decision Tree Regressor
RMSE =0.3
R^2 Score=69%

Random Forest Regressor
RMSE =0.24
R^2 Score =79%

XG Boosting Regression
RMSE =0.24
R^2 Score =80%

From the above models,it is clearly visible that both Random Forest Regressor and Gradient

Boosting Algorithm gave a very close results of 80%.Lets perform Grid search on both models to finetune the parameters

Grid Search CV Random Forest Regressor Model Performance:
R-squared = 0.79.
RMSE = 0.25

Grid Search CV Gradient Boosting regression Model Performance:
R-squared = 0.78.
RMSE = 0.23

R

Multiple Linear Regression:
MAPE :0.6
RMSE :0.97

Decision Tree Regression:
MAPE :0.4
RMSE :0.98

Random Forest Regression:
MAPE :0.3
RMSE :0.97

XG Boosting

MAPE :0.4
RMSE :0.97

Model Selection :

As a good model should have a least RMSE And Max R Squared value,
From the above table it can be concluded that :
→ Both the models- Gradient Boosting Default and Random Forest perform comparatively well while comparing their RMSE and R-Squared value.
→ After this, I choose Random Forest CV and Grid Search CV to apply cross validation technique and see changes brought by them.
→After applying tuning Random forest model showed best results compared to gradient boosting.
→ we can say that Random forest model is the best method to make prediction for this project with highest explained variance of the target variables and lowest error chances with parameter tuning technique Grid Search CV.
Finally Cab fare is predicted using the RandomForest Regressor and is stored in **Predictedfare.csv**

--The End--