Cab Fare Prediction

Chaitrali A Deokar

June 2019

Contents

Chapter 1

INTRODUCTION

1.1 Problem Statement

With technological advancements, there is a rise in the number of cabs one takes for commuting. One of the reason for growing use of cab uses are how accurately the fair is predicted for a particular distance without charging extra by middlemen.

With increasing number of cab rental startup it is necessary to be able to accurately predict the cab fare for any distance, otherwise it would have inverse output

1. Prediction < correct fare: result in loss of revenue to the company
2. Prediction > correct fare: result in passenger being over charged and hence losing faith in the company resulting in loss of customers and creation of bad image.

Hence it is of utmost necessity to be able to predict the cab fare accurately and precisely to ensure that both the company and the passengers are benefited.

1.2 Data

The Analysis involves building a Regression model to predict the Cab Fare depending on a number of input parameters. The given train data is a CSV file that consists 6 variables and 16067 Observation. Given below is a sample of the data that we develop our model on:

fare_amount	pickup_datetime	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_count
4.5	2009-06-15 17:26:21 UTC	-73.844311	40.721319	-73.84161	40.71228	1
16.9	2010-01-05 16:52:16 UTC	-74.016048	40.711303	-73.97927	40.782	1
5.7	2011-08-18 00:35:00 UTC	-73.982738	40.76127	-73.99124	40.75056	2
7.7	2012-04-21 04:30:42 UTC	-73.98713	40.733143	-73.99157	40.75809	1
5.3	2010-03-09 07:51:00 UTC	-73.968095	40.768008	-73.95666	40.78376	1

Table 1.1 -train cab.csv

The given attributes are:

a. **Fare_amount –** The total fare amount of the given cab ride.
b. **pickup_datetime** - timestamp value indicating when the cab ride started.
c. **pickup_longitude** - float for longitude coordinate of where the cab ride started.
d. **pickup_latitude** - float for latitude coordinate of where the cab ride started.
e. **dropoff_longitude** - float for longitude coordinate of where the cab ride ended.
f. **dropoff_latitude** - float for latitude coordinate of where the cab ride ended.
g. **passenger_count** - an integer indicating the number of passengers in the cab ride

Among the given variables the target variable is the 'fare_amount' variable. The other variables help us identify the pickup point and the drop point depend on the latitude and longitude point on a particular date and time and the number of passengers for a particular ride.

Hence the Independent variables are as follows:

Independent Variables
pickup_datetime
pickup_longitude
pickup_latitude
dropoff_longitude
dropoff_latitude
passenger_count

Table 1.2 Independent Variables

The target/ dependent variable is:

Independent Variables
pickup_longitude

Table 1.3 Target Variable

Chapter 2

Methodology

2.1 CRISP DM Process

CRISP-DM stands for cross-industry process for data mining. The CRISP-DM methodology provides a structured approach to planning a data mining project. The project follows CRISP Dm process to develop the model for the given problem. It involves the following steps:

1. Business understanding
2. Data understanding
3. Data preparation
4. Modeling
5. Evaluation
6. Deployment

2.2 Business Understanding

1. **Set objectives** - This primary objective from a business perspective would be " How to accurately predict the Cab fare for a particular ride?"
2. **Produce project plan** – The plan will involve Understanding the data and converting into a proper shape , i.e., a data free from any missing values, outliers that can possibly produce errors; scaling the variables; applying various models and choosing the best model to predict the bike rental count
3. **Business success criteria** – The success would be related to able to accurately predict the cab fare for a particular input , with maximum accuracy . On being able to predict it accurately.

2.3 Data Understanding

The given train data is a CSV file that consists 6 variables and 16067 Observation. Given below is a sample of the data that we develop our model on:

The given attributes are:

a. **fare_amount** – The total fare amount for a particular ride.
b. **pickup_datetime** - timestamp value indicating when the cab ride started.
c. **pickup_longitude** - float for longitude coordinate of where the cab ride started.
d. **pickup_latitude** - float for latitude coordinate of where the cab ride started.
e. **dropoff_longitude** - float for longitude coordinate of where the cab ride ended.
f. **dropoff_latitude** - float for latitude coordinate of where the cab ride ended.
g. **passenger_count** - an integer indicating the number of passengers in the cab ride

Among the given variables the target variable is the 'fare_amount' variable. The other variables help us identify the pickup point and the drop point depend on the latitude and longitude point on a particular date and time and the number of passengers for a particular ride.

Also, there is a separate test data that is provided as a CSV file containing 9514 observations and 6 variables. All of them are the Independent variables.

pickup_datetime	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_count
2015-01-27 13:08:24 UTC	-73.97332	40.7638054	-73.9814301	40.7438355	1
2015-01-27 13:08:24 UTC	-73.9868622	40.7193832	-73.9988861	40.7392006	1
2011-10-08 11:53:44 UTC	-73.982524	40.75126	-73.979654	40.746139	1
2012-12-01 21:12:12 UTC	-73.98116	40.767807	-73.990448	40.751635	1

Table 2.1 test.csv data

Once the model is developed, it will be tested on this train data.

2.4 Data Pre Processing

Data preprocessing is a data mining technique that involves transforming raw data into an understandable format. Real-world data is often incomplete, inconsistent, and/or lacking in certain behaviors or trends, and is likely to contain many errors. Data preprocessing is a proven method of resolving such issues. Data preprocessing prepares raw data for further processing.

2.4.1 Missing Value Analysis

Missing value is the occurrence of incomplete observations in asset of data. Missing values can arise due to many reasons, error in uploading data, unable to measure a particular observation etc.

Due to presence of missing values, that observation either gets value as 0 or NA. These affect the accuracy of model. Hence it is necessary to check for any missing values in the given data.

Missing Value Analysis in R:

On primary check of the dataset it is seen that there are a lot of values that are missing. The missing values are in different forms as below,

1. Values containing '0'- These will first need to be changed to NA and Nan in R and python respectively.
2. Blank spaces that are taken as NA and NaN in R and Python respectively.
3. Also, there are few places where the start location(pickup_longitude and pickup_latitude) is same as the drop location(dropoff_longitude and dropoff_latitude). Such entries will not provide any meaningful analysis, hence rows that contain such data are removed from the dataset to improve the performance.

Depending on the percentage of missing values we can decide if we need to keep a particular variable or drrop it based on the following conditions:

1. Missing value percentage < 30% - We can include the variable.
2. Missing value percentage > 30 % - We need to remove the particular variable. This is because even if we impute values n the variable, since they are not the actual values, the variable will not contain original values and will not contribute effectively to creating the model.

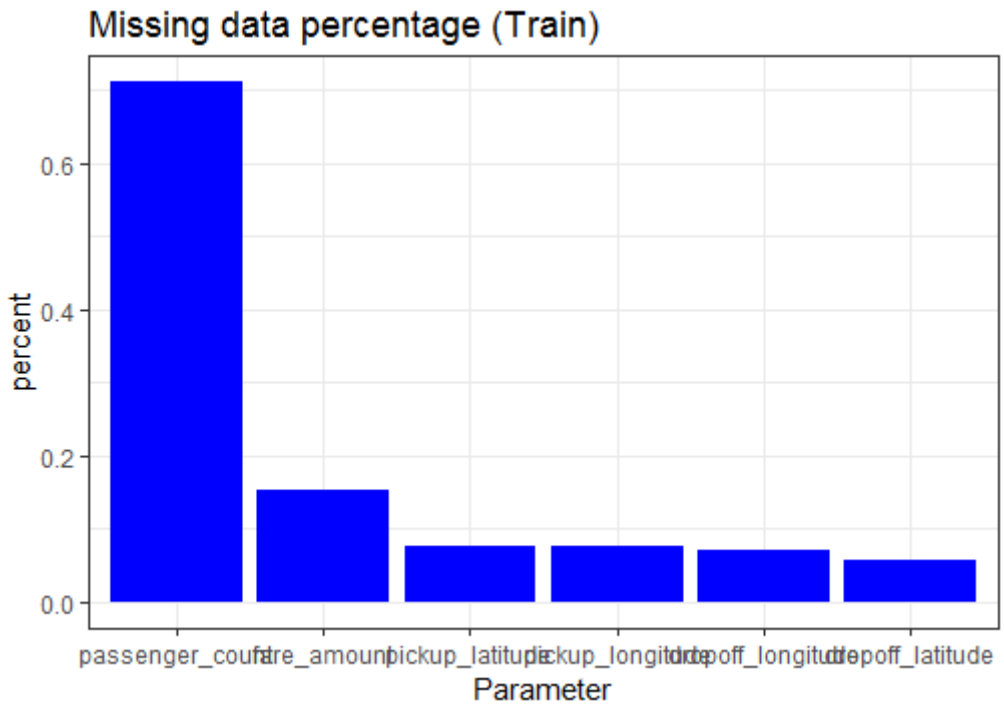

Fig 2.1 Missing Value Percentage of the variables

It can be seen from the above graph that the highest number of missing values are present in Passenger count(approx. 0.7 percent)

Since the percentage of missing values of all the variables is less than the standard percentage of 30%, we do not exclude it.

Impute the missing value:

The missing values can be imputed using the following methods

1. Fill with central tendencies, i.e, with Mean , Median or Mode
2. Distance based or Data mining method like KNN imputation
3. Prediction Based, i.e based on Predictive Machine Learning Algorithm

To choose which one of these methods can be used, we can try these methods on a subset and check which predicts values close to the original data. For our data, KNN imputation worked the best.

Hence, KNN method was used to impute missing Values. After KNN imputation, the missing values were not there

	variables	Missing_val_count	percent
1	fare_amount	0	0
2	pickup_longitude	0	0
3	pickup_latitude	0	0
4	dropoff_longitude	0	0
5	dropoff_latitude	0	0
6	passenger_count	0	0

Table 2.1 Missing value after imputation

2.4.2 Outlier Analysis

An Outlier is any inconsistent or abnormal observation that deviates from the rest of the observations. These inconsistent observation can be due to manual error, poor quality/ malfunctioning equipments or correct but exceptional data. It can cause an error in prediciting the target variables.

Hence we need to check for the outliers and either remove the observations containing them or replace them with NA and later impute values.

Outliers in Fare_amount

Usually fare amount for any cab ride (no matter how short the distance is) is always a positive number. But, in our training data it is seen that there are few instances of fare amount which are negative. Given below are such instances:

fare_amount	pickup_datetime	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_count
-2.9	2010-03-09 23:37:10 UTC	-73.7895	40.6435	-73.7887	40.64195	1
-2.5	2015-03-22 05:14:27 UTC	-74	40.72063	-73.9998	40.72054	1
-3	2013-08-30 08:57:10 UTC	-73.9951	40.74076	-73.9959	40.74136	4

Table 2.4.1 Outliers in 'fare_amount'

Outliers in passenger_count:

Usually cabs can be 4 seater to a maximum of 8 seaters. Cabs cannot accommodate more than that. On preliminary check of the data it is seen that the passenger_cout has very large values.These are outliers and need to be eliminated.

fare_amount	pickup_datetime	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_count
4.9	2010-07-12 09:44:33 UTC	-73.9832	40.73466	-73.9913	40.73892	456
6.1	2011-01-18 23:48:00 UTC	-74.0066	40.73893	-74.0108	40.71791	5334
8.5	2013-06-18 10:27:05 UTC	-73.9921	40.7642	-73.973	40.7627	535
8.1	2009-08-21 19:35:05 UTC	-73.9609	40.76156	-73.9763	40.74836	354

Table 2.4.2 Outliers in 'passenger_count'

Outliers in Location points:

On viewing the data it is clear that most of the longitude points are within the 70 degree frame whereas most of the latitude points are withing the 40 degree frame.

An outlier in this would mean any point that is very far from this. To be able to analyse if there are outliers let us look at the range of the location points.

The below table contains the range of the location points:

Variable	Minimum Value	Maximum Value
pickup_longitude	-74.43823	40.76613
pickup_latitude	-74.00689	401.08333
dropoff_longitude	-74.22705	40.80244
dropoff_latitude	-74.00638	41.36614

Table 2.4.3 Range of location points

It can be seen that despite most of the values being similar, the range is exceeded most of the actual range. The reason is due to the presence of outlier values.

The below is a view few the values:

fare_amount	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_count
15	40.72913	-74.0069	40.76337	-73.9616	1
52	40.73688	-74.0062	40.73689	-74.0064	6
15.5	40.76442	-73.9929	40.80244	-73.9507	1
6.5	40.74826	-73.9918	40.74037	-73.979	1
3.3	-73.9472	401.0833	-73.9514	40.77893	1

Table 2.4.4 Abnormal Data in location points

It can be seen that these values are probably due to manual error while recording the values. Probably the values of the Longitude and latitude points are interchanged and also 401.08 a value not possible for both the latitude and longitude.

Hence these inconsistent values are to be replaced by NA and hen imputed. The total number of outliers present in each of the variable is given below:

variables	Missing_val_count	percent
fare_amount	3	0.01922091
pickup_longitude	780	4.99743721
pickup_latitude	497	3.18426448
dropoff_longitude	897	5.74705279
dropoff_latitude	736	4.7155305
passenger_count	18	0.11532547

Table 2.4.5 Missing Value count

After Removal of Outliers:

After removal of outliers, the range of the variables is as shown below

Variables	Minimum Value	Maximum Value
pickup_longitude	-74.01811	-73.93187
pickup_latitude	40.68974	40.81453
dropoff_longitude	-74.02946	-73.92674
dropoff_latitude	40.68823	40.81639

Table 2.4.6 Range after outlier removal

2.4.3 Feature Selection

Before creating a model we need to assess the importance of each predictor variable in our analysis. There is a possibility that many variables in our analysis are not important at all to the problem of prediction. Hence, feature selection can help in reducing the time for computation of model as well as the complexity of the model. Also, few models require the independent variables to be free from multicollinear effect, hence it is needed to perform various procedures to ensure that the independent variables are not collinear.

The 1st step involved is to do the correlation analysis for the categorical variables and continuous variables. Since our Predictor variable is also continuous we will plot a correlation table that would predict the correlation strength between the dependent variable and the 'fare_amount' variable

Creating a new Variable 'dist'

The given data contain only the pick up and drop points in longitude and latitude. The fare_amount would depend mailnly on the distance between the two (pick up and dropoff locations) rather than the points themselves. Hence before analyzing the correlation between the variables, it is necessary to create a new variable that will be a numeric value of the dstance between the pickup and dropoff locations.

The haversine formula determines the great-circle distance between two points on a sphere given their longitudes and latitudes. The **Haversine** formula calculates the shortest distance between two points on a sphere using their latitudes and longitudes measured along the surface. It is important for use in navigation.

```
# vectorized haversine function
def haversine(lat1, lon1, lat2, lon2, to_radians=True, earth_radius=6371):
    if to_radians:
        lat1, lon1, lat2, lon2 = np.radians([lat1, lon1, lat2, lon2])
    a = np.sin((lat2-lat1)/2.0)**2 + \
        np.cos(lat1) * np.cos(lat2) * np.sin((lon2-lon1)/2.0)**2

    return earth_radius * 2 * np.arcsin(np.sqrt(a))
```

The few rows with the dist columns are below:

fare_amount	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_count	Dist
4.5	-73.98172	40.72132	-73.98011	40.71228	1	1.0156499
16.9	-74.01605	40.7113	-73.97927	40.782	1	8.4595997
5.7	-73.98274	40.76127	-73.99124	40.75056	2	1.3910818
7.7	-73.98713	40.73314	-73.99157	40.75809	1	2.8024061
5.3	-73.9681	40.76801	-73.95665	40.78376	1	2.0013963

Table 2.4.7 New column containing distance between two points

Correlation Plot:
Now on creating the correlation plot:

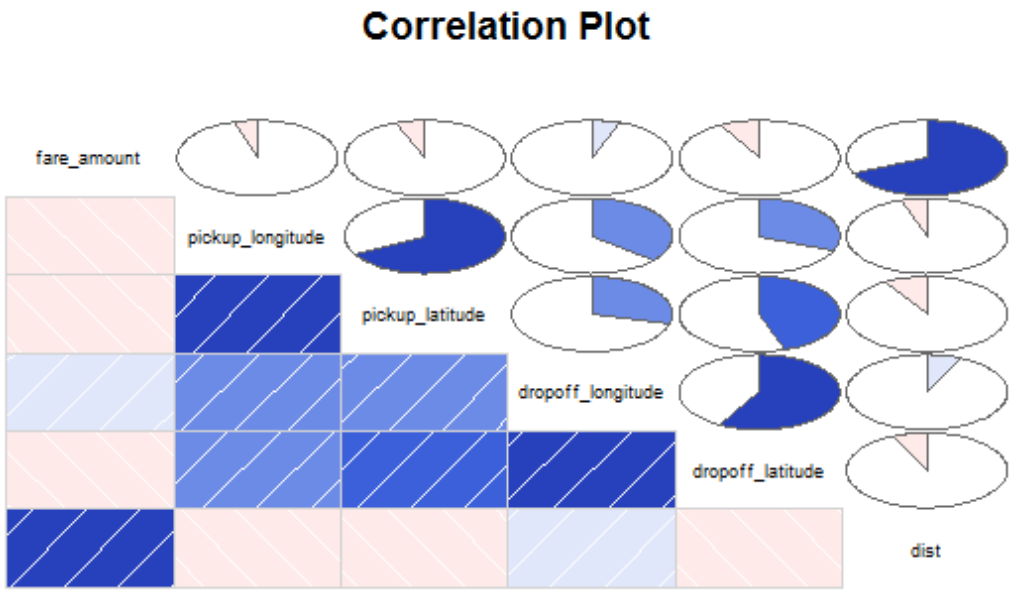

Fig 2 Correlation plot of numeric Variables

1.5 Modeling

We know that the target variable is “fare_amount”. The model is to predict a numeric value, hence it is a **Regression** problem statement.

The various models that can be used for prediction problem statement are Decision trees, Random Forest, Linear Regression and KNN imputation.

2.5.1 Decision Tree

A Decision Tree is a supervised learning predictive model that uses a set of binary rules to calculate a target value.

Decision trees have three main parts:

- **Root Node :** The node that performs the first split
- **Terminal Nodes/Leaves :**Nodes that predict the outcome.
- **Branches :** arrows connecting nodes, showing the flow from question to answer.

The root node is the starting point of the tree, and both root and terminal nodes contain questions or criteria to be answered.Each node typically has two or more nodes extending from it. For example, if the question in the first node requires a “yes” or “no” answer, there will be one leaf node for a “yes” response, and another node for “no.”

Decision Tree in R

The Decision tree algorithm is applied in R with all the input variables except the pickup_datetime variable. The model developed is as below:

```
> fit
n= 11820

node), split, n, deviance, yval
      * denotes terminal node

 1) root 11820 314955.00  9.554943
   2) dist< 2.727234 7735 108831.50  7.394478
     4) dist< 1.601553 4468  48100.35  6.243319 *
     5) dist>=1.601553 3267  46712.84  8.968822 *
   3) dist>=2.727234 4085 101656.20 13.645810
     6) dist< 4.49323 2530  45660.74 11.998300 *
     7) dist>=4.49323 1555  37955.49 16.326320
      14) dist< 7.033306 1293  28794.96 15.600950 *
      15) dist>=7.033306 262   5122.71 19.906110 *
```

The above shows the rules of splitting of threes. The main root splits into 2 nodes having dist< 2.727234 and dist>=2.727234 as their conditions. Similarly the nodes further split. We apply these rules to predict the test cases.

The line with * shows that it is the terminal node. These rules are then applied on the test data to predict values.

14.19502

Decision Tree in Python

```
DecisionTreeRegressor(criterion='mse', max_depth=2, max_features=None,
      max_leaf_nodes=None, min_impurity_decrease=0.0,
      min_impurity_split=None, min_samples_leaf=1,
      min_samples_split=2, min_weight_fraction_leaf=0.0,
      presort=False, random_state=None, splitter='best')
```

The above shows the criteria that are used in building the decision tree in Python. To build the mode l I python no specific criteria of the maximum number of leaf nodes etc are provided. Hence the defa ult values are chosen in developing the model.

2.5.2 Random Forest

Random forest is a way of averaging multiple deep decision trees, trained on different parts of the sa me training set, with the goal of overcoming over-fitting problem of individual decision tree.

In other words, random forests are an ensemble learning method for classification and regression th at operate by constructing a lot of decision trees at training time and outputting the class that is the mode of the classes output by individual trees.

Random Forest in R

On creating a RandomForest model the importance contributed by individual variables can be seen u sing importance function.

```
> importance(RF_model, type = 1)
                %IncMSE
pickup_longitude  18.714544
pickup_latitude   23.009147
dropoff_longitude 24.859336
dropoff_latitude  25.979808
passenger_count   -2.089081
dist              78.319964
```

The above shows that the most important variable for predicting the fare_amount is dist and the least important is passenger_count

Random Forest in Python

```
RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=None,
           max_features='auto', max_leaf_nodes=None,
           min_impurity_decrease=0.0, min_impurity_split=None,
           min_samples_leaf=1, min_samples_split=2,
           min_weight_fraction_leaf=0.0, n_estimators=10, n_jobs=None,
           oob_score=False, random_state=None, verbose=0, warm_start=Fa
lse)
```

Similar to Decision tree these are all the criteria values that are passed to develop the Random Forest model in python.

2.5.3 Linear Regression

Linear regression is used to predict the value of an outcome variable *Y* based on one or more input predictor variables *X*. The aim is to establish a linear relationship (a mathematical formula) between the predictor variable(s) and the response variable, so that, we can use this formula to estimate the value of the response *Y*, when only the predictors (*Xs*) values are known.

Linear regression in R

On developing the model on the train data the inference is as follows:

```
Call:
lm(formula = fare_amount ~ ., data = train1)

Residuals:
    Min      1Q  Median      3Q     Max
-19.4376  -1.9307  -0.9371   0.7004  23.7188

Coefficients:
                   Estimate Std. Error t value Pr(>|t|)
(Intercept)      689.54056  329.36283   2.094  0.03632 *
pickup_longitude  -11.32824    2.98773  -3.792  0.00015 ***
pickup_latitude    10.25032    2.30562   4.446 8.84e-06 ***
dropoff_longitude  16.63405    2.59908   6.400 1.61e-10 ***
dropoff_latitude  -17.42745    2.03736  -8.554  < 2e-16 ***
passenger_count2    0.02274    0.10072   0.226  0.82138
passenger_count3    0.21386    0.17344   1.233  0.21758
passenger_count4    0.12708    0.24494   0.519  0.60390
passenger_count5   -0.13596    0.14368  -0.946  0.34402
passenger_count6    1.05594    0.26574   3.974 7.12e-05 ***
dist                1.99605    0.02034  98.148  < 2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 3.777 on 11809 degrees of freedom
Multiple                                    R-squared:                                                                  0.4651,
       Adjusted R-squared:  0.4647
F-statistic:  1027 on 10 and 11809 DF,  p-value: < 2.2e-16
```

The above shows how the target variable,i.e, the fare_amount varies with change in each individual input.

For eg, with one unit change in the pickup_longitude, the fare_amount decreases by 11 units, provided all other input variables are left constant.

Also, the R squared and adjusted R squared values show how much variance of the output variable is explained by the independent or input variables. Here the adjusted r square value is 46.47%, which indicated that only 46% of the variance of fare_amount is explained by the input variables. This shows that the model is relatedly poor in performance.

This can be due to the fact that one or more input to the model might not be normal , and the relationship between the independent and dependent variable might be non linear.

Linear Regression in Python

In python before developing the model, we need to spilt the categorical variables so that each category becomes a column and the position where it is present it get a value 1 and where it is not present it gets value 0.
In our data only 1 categorical variable existes- passenger_count with 6 categories. They are split into 6 columns. If row 1 has 1 passenger then the column under passenger_1 row will be set as one and all other columns of passenger will be set to 0.
After this the model is developed

OLS Regression Results

Dep. Variable:	fare_amount	**R-squared:**	0.456
Model:	OLS	**Adj. R-squared:**	0.455
Method:	Least Squares	**F-statistic:**	987.8
Date:	Fri, 07 Jun 2019	**Prob (F-statistic):**	0.00
Time:	15:05:29	**Log-Likelihood:**	-32497.
No. Observations:	11816	**AIC:**	6.502e+04
Df Residuals:	11805	**BIC:**	6.510e+04
Df Model:	10		
Covariance Type:	nonrobust		

Here, theR squared and adjusted R squared values show how much variance of the output variable is explained by the independent or input variables. Here the adjusted r square value is 46.47%, which indicated that only 46% of the variance of fare_amount is explained by the input variables. This shows that the model is relatedly poor in performance.

This can be due to the fact that one or more input to the moel might not be normal , and the relationship between the independent and dependent variable might be non linear.

The "F value'' and "Prob(F)'' statistics test the overall significance of the regression model. Specifically, they test the null hypothesis that all of the regression coefficients are equal to zero. This tests the full model against a model with no variables and with the estimate of the dependent variable being the mean of the values of the dependent variable. The F value is the ratio of the mean regression sum of squares divided by the mean error sum of squares. Its value will range from zero to an arbitrarily large number.

The value of Prob(F) is the probability that the null hypothesis for the full model is true (i.e., that all of the regression coefficients are zero). For example, if Prob(F) has a value of 0.01000 then there is 1 chance in 100 that all of the regression parameters are zero. This low a value would imply that at least some of the regression parameters are nonzero and that the regression equation does have some validity in fitting the data (i.e., the independent variables are not purely random with respect to the dependent variable).

	coef	std err	t	P>|t|	[0.025	0.975]
pickup_longitude	-15.6361	3.040	-5.143	0.000	-21.595	-9.677
pickup_latitude	10.9286	2.339	4.673	0.000	6.344	15.513
dropoff_longitude	18.9589	2.686	7.059	0.000	13.694	24.223
dropoff_latitude	-17.9903	2.104	-8.552	0.000	-22.114	-13.867
dist	1.9919	0.021	96.339	0.000	1.951	2.032
passenger_count_1	538.1594	336.838	1.598	0.110	-122.098	1198.417
passenger_count_2	538.2359	336.840	1.598	0.110	-122.025	1198.497
passenger_count_3	538.2980	336.842	1.598	0.110	-121.969	1198.565
passenger_count_4	538.3609	336.842	1.598	0.110	-121.905	1198.627
passenger_count_5	538.1210	336.839	1.598	0.110	-122.138	1198.380
passenger_count_6	539.3901	336.839	1.601	0.109	-120.870	1199.650

Omnibus:	5960.846	**Durbin-Watson:**	1.999
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	44870.186
Skew:	2.320	**Prob(JB):**	0.00
Kurtosis:	11.344	**Cond. No.**	2.83e+06

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 2.83e+06. This might indicate that there are strong multicollinearity or other numerical problems.

2.5.4 KNN imputation moel

The KNN model finds the nearest neigbhors and tries to predict target value.
The algorithm uses '**feature similarity**' to predict values of any new data points. This means that the new point is assigned a value based on how closely it resembles the points in the training set.

These are the various models developed on the given data. The Data is divided into train and test. The model is developed on the train data and the test model is fit into it to predict the target variables. Later the actual and predicted values of target variable are compare to get the error and accuracy.

The model with least error and best accuracy will be taken as the model for future use.

Chapter 3

Conclusion

3.1 Model Evaluation

Now that we have a few models for predicting the target variable, we need to decide which one to choose. There are several criteria that exist for evaluating and comparing models. We can compare the models using predictive performance as the criteria to compare and evaluate models.

Predictive performance can be measured by comparing Predictions of the models with real values of the target variables, and calculating some average error measure.

3.1.1 Mean Absolute Error (MAE)

MAE is one of the error measures used to calculate the predictive performance of the model. We will apply this measure to our models that we have generated in the previous section.

In R, Define MAPE usinf function

```
> MAPE = function(y, yhat){
    mean(abs((y - yhat)/y)*100)
  }
```

R	
Method	Mape Error(in Percentage)
Decision Tree	27.0018
Random Forest	22.43
Linear Regression	25.87
KNN Imputation	32.61

Table 3.1 MAPE Error in R

Python	
Method	Mape Error(in Percentage)
Decision Tree	29.7161746
Random Forest	25.70792047
Linear Regressio	34.58694638
KNN Imputation	27.01811016

Table 3.1 MAPE Error in Python

3.1.2 Accuracy

It is the ratio of number of correct predictions to the total number of predictions made.

Accuracy= number of correct predictions / Total predicitions made

It can also be calculated from MAE as

Accuracy = 100 – MAE

R	
Method	Accuracy (in Percentage)
Decision Tree	72.9982
Random Forest	77.57
Linear Regression	74.13
KNN Imputation	67.39

Table 3.3 Accuracy in R

Python	
Method	Accuracy (in Percentage)
Decision Tree	70.2838254
Random Forest	74.29207953
Linear Regression	65.41305362
KNN Imputation	72.98188984

Table 3.4 Accuracy in Python

3.2 Model Selection

From the values of Error and accuracy, it is seen that all the models perform similar in terms of Error. As such any model can be used, but **Random forest** gives better results compared to other algorithms.

APPENDIX -I
COMPLETE R CODE

```
rm(list=ls())

# set the working directory
setwd("C:/Users/Chaitrali/Downloads/Data/data01s2l1/Edwisor-Project/CabfarePred")
getwd()

#load libraries
x=c("ggplot2", "DMwR", "corrgram", "Hmisc", "rpart", "randomForest", "geosphere")
lapply(x, require, character.only = TRUE)
rm(x)

# Load the input data
train= read.csv("train_cab.csv", header = T)[,-2]
pre_test = read.csv("train_cab.csv", header = T)

str(train)
#convert fare amount into proper type
train$fare_amount=as.numeric(as.character(train$fare_amount))
train$passenger_count=as.integer(train$passenger_count)
#replace all "0" with NA
train=subset(train, !(train$pickup_longitude==train$dropoff_longitude &
train$pickup_latitude==train$dropoff_latitude))
train[train==0]= NA

#create a function to calculate missing values

missingvalue= function(data){
 mv=data.frame(apply(data, 2 , function(x){sum(is.na(x))}))
 colnames(mv)="Missing_val_count"
 mv$percent=apply(mv , 1 , function(x){x/nrow(train)*100})
 mv=cbind(row.names(mv), mv)
 row.names(mv)=NULL
 colnames(mv)[1]="variables"
 print(mv)


 #plot Missing Values
 ggplot(data = mv, aes(x=reorder(variables , -percent),y = percent))+
 geom_bar(stat = "identity",fill = "blue")+xlab("Parameter")+
 ggtitle("Missing data percentage (Train)") + theme_bw()
}


#Calculate Missing Values
missingvalue(train)

#Since PAssenger_count is categorical, hence replace it using mode
#create a function to calculate mode
mode= function(data){
```

```
 uniq=unique(data)
 as.numeric(as.character(uniq[which.max(tabulate(match(data,uniq)))]))
 #print(mode_d)
}

mode(train$passenger_count)

#impute with the mode
train$passenger_count[is.na(train$passenger_count)] = mode(train$passenger_count)


#Check suitable methods to impute Missing value for numerical data
#train[1,2]=-73.84431
#Mean= -73.91186
#Median=-73.98204
#KNN= -73.84371
df=train
train=train[complete.cases(train[,1]),]
#Since KNN is giving the value closest to the original one, We choose KNN for missing value imputation
train=knnImputation(train, k=1)

missingvalue(train)

######################Outlier Analysis#################################################33
df=train
#outliers in fare_amount
#Remove negative values from 'fare_amount'
train$fare_amount=ifelse(train$fare_amount<0, NA, train$fare_amount)
train$fare_amount=ifelse(train$fare_amount>30,NA, train$fare_amount)
#outliers in passenger_count
#all values greater than 8 are converted to NA
unique(train$passenger_count)

for (i in 1:nrow(train)){
 if (as.integer(train$passenger_count[i]) > 8){
   train$passenger_count[i]=NA
 }
}

train['passenger_count'][train['passenger_count']<0 | train['passenger_count']>8]= NA

train[train==0]=NA

#Outliers in Pickup_latitude

#Get the range of the locations
range(train$pickup_longitude)
range(train$pickup_latitude)
range(train$dropoff_longitude)
range(train$dropoff_latitude)
```

```
cnames=colnames(train[,c(2:5)])

for (i in 1:length(cnames))
{
 assign(paste0("gn",i), ggplot(aes_string(y = (cnames[i]), x = "fare_amount"), data = train)+
        stat_boxplot(geom = "errorbar", width = 0.5) +
        geom_boxplot(outlier.colour="red", fill = "grey" ,outlier.shape=18,
                     outlier.size=1, notch=FALSE) +
        theme(legend.position="bottom")+
        labs(y=cnames[i],x="y")+
        ggtitle(paste("Box plot of fare amount",cnames[i])))
}

## Plotting plots together
gridExtra::grid.arrange(gn1, gn2, ncol=2)
gridExtra::grid.arrange(gn3, gn4,gn5, ncol=3)

#Replace all outliers with NA and impute
#create NA on outliers
for(i in cnames){
 val = train[,i][train[,i] %in% boxplot.stats(train[,i])$out]
 print(length(val))
 train[,i][train[,i] %in% val] = NA
}

missingvalue(train)

#replace missinf value with mode
mode(train$passenger_count)
train$passenger_count[is.na(train$passenger_count)] = mode(train$passenger_count)
train=train[complete.cases(train[,1]), ]

#replace all other missing value with mean
train$fare_amount[is.na(train$fare_amount)] = mean(train$fare_amount, na.rm=T)
train$pickup_longitude[is.na(train$pickup_longitude)] = mean(train$pickup_longitude, na.rm=T)
train$pickup_latitude[is.na(train$pickup_latitude)] = mean(train$pickup_latitude, na.rm=T)
train$dropoff_longitude[is.na(train$dropoff_longitude)] = mean(train$dropoff_longitude,
na.rm=T)
train$dropoff_latitude[is.na(train$dropoff_latitude)] = mean(train$dropoff_latitude, na.rm=T)

range(train$fare_amount)
range(train$pickup_longitude)
range(train$pickup_latitude)
range(train$dropoff_longitude)
range(train$dropoff_latitude)
unique(train$passenger_count)

train$passenger_count=as.factor(train$passenger_count)
str(train)
missingvalue(train)
```

```
#########################Feature Selection#####################
df=train

#create new variable
library(geosphere)
train$dist= distHaversine(cbind(train$pickup_longitude, train$pickup_latitude),
cbind(train$dropoff_longitude,train$dropoff_latitude))
#the output is in metres, Change it to kms
train$dist=as.numeric(train$dist)/1000
df=train
train=df

#correlation analysis
corrgram(train[,-6], order = F,
       upper.panel=panel.pie, text.panel=panel.txt, main = "Correlation Plot")

#Finding the correlation between the numeric variables
num_cor=round(cor(train[,-6]), 3)


#check if there are any duplicate rows
abc=subset(train,(train$pickup_longitude==train$dropoff_longitude &
train$pickup_latitude==train$dropoff_latitude))
#Hence, there are no duplicates
rm(abc,df,gn1,gn2,gn3,gn4,cnames,i,val)

#########################MODELDEvelopment#########################################
##############3

#create sampling and divide data into train and test
set.seed(123)
train_index = sample(1:nrow(train), 0.8 * nrow(train))

train1 = train[train_index,]#do not add column if already removed
test1 = train[-train_index,]#do not add column if already removed

write.csv(train1, "Trial5.csv", row.names = F)
write.csv(test1, "Test5.csv", row.names = F)

MAPE = function(y, yhat){
  mean(abs((y - yhat)/y*100))
}

#Decision Tree
fit = rpart(fare_amount ~. , data = train1, method = "anova", minsplit=5)
summary(fit)
predictions_DT = predict(fit, test1[,-1])
MAPE(test1[,1], predictions_DT)
write.csv(predictions_DT, "DT_R_PRed5.csv", row.names = F)
#Error 27.0018
#Accuracy
```

```
#Random Forest
RF_model = randomForest(fare_amount ~.  , train1, importance = TRUE, ntree=100)
RF_Predictions = predict(RF_model, test1[,-1])
MAPE(test1[,1], RF_Predictions)
write.csv(RF_Predictions, "RF_r_pred5.csv", row.names = F)
importance(RF_model, type = 1)
#error 22.43 for n=100
#Accuracy

#Linear Regression
lm_model = lm(fare_amount ~. , data = train1)
summary(lm_model)
predictions_LR = predict(lm_model, test1[,-1])
MAPE(test1[,1], predictions_LR)
write.csv(predictions_LR, "LR_R_pred5.csv", row.names = F)
#error 25.87
#accuracy


##KNN Implementation
library(class)
#Predict test data
KNN_Predictions = knn(train1[, 2:7], test1[, 2:7], train1$fare_amount, k = 1)
KNN_Predictions=as.numeric(as.character((KNN_Predictions)))
#Calculate MAPE
MAPE(test1[,1], KNN_Predictions)
write.csv(KNN_Predictions, "KNN_r_pred5.csv", row.names = F)
#error 32.61
#accuracy

###############FIne Tuning##############################3333

#Random Forest
RF_model = randomForest(fare_amount ~.  , train1, importance = TRUE, ntree=200, mtry=2)
RF_Predictions = predict(RF_model, test1[,-1])
MAPE(test1[,1], RF_Predictions)
write.csv(RF_Predictions, "RF_r_pred5.csv", row.names = F)
importance(RF_model, type = 1)
#error 22.38 for n=100
#Accuracy

rm(a, num_cor,pre, i)
#####################################Predict VAlues###################3

pred_data=read.csv("test.csv", header= T)[,-1]

#create dist variable
pred_data=subset(pred_data,   !(pred_data$pickup_longitude==pred_data$dropoff_longitude   &
pred_data$pickup_latitude==pred_data$dropoff_latitude))
pred_data[pred_data==0]= NA
```

```
str(pred_data)
pred_data$passenger_count=as.factor(pred_data$passenger_count)
pred_data$dist=  distHaversine(cbind(pred_data$pickup_longitude,  pred_data$pickup_latitude),
cbind(pred_data$dropoff_longitude,pred_data$dropoff_latitude))
#the output is in metres, Change it to kms
pred_data$dist=as.numeric(pred_data$dist)/1000
pred_data$fare_amount=0
pred_data=pred_data[,c(7,1,2,3,4,5,6)]
#Random Forest
RF_model = randomForest(fare_amount ~.  , train, importance = TRUE, ntree=200, mtry=2)
pred_data$fare_amount = predict(RF_model, pred_data[,-1])
write.csv(pred_data,"pred_data.csv",row.names = F)
```

APPENDIX -II

COMPLETE PYTHON CODE

```
# In[1]:
#Load libraries
import os
import pandas as pd
import numpy as np
from fancyimpute import KNN
import matplotlib.pyplot as plt
from scipy.stats import chi2_contingency
import seaborn as sns
from random import randrange, uniform
# In[2]:
os.chdir("C:/Users/Chaitrali/Downloads/Data/data01s2l1/Edwisor-Project/CabfarePred")
# In[33]:
cab=(pd.read_csv('train_cab.csv', header = 0 )).drop(columns="pickup_datetime")


# In[34]:
#Remove all entires where the pickup and drop locations are same
cab=cab[np.logical_and(cab['pickup_longitude'] != cab['dropoff_longitude'], cab['pickup_latitude']
!= cab['dropoff_latitude'])]

# In[35]:
#replace 0 with NA in the variables
cab['fare_amount']= cab['fare_amount'].apply(pd.to_numeric, errors='coerce')
cab['fare_amount']= cab['fare_amount'].replace({0:np.nan})
cab['passenger_count']=cab['passenger_count'].fillna(0)
cab['passenger_count']= cab['passenger_count'].astype(int)
cab['passenger_count']=cab['passenger_count'].replace({0: np.nan})
cab['pickup_longitude']= cab['pickup_longitude'].replace({0:np.nan})
cab['pickup_latitude']= cab['pickup_latitude'].replace({0:np.nan})
cab['dropoff_longitude']= cab['dropoff_longitude'].replace({0:np.nan})
cab['dropoff_latitude']= cab['dropoff_latitude'].replace({0:np.nan})


# In[36]:
cab.shape

# In[37]:
#creaet a function to calculate missing values
def missingval(data):
   missing_val = pd.DataFrame(data.isnull().sum())

#Reset index
   missing_val = missing_val.reset_index()

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

```
#Calculate percentage
   missing_val['Missing_percentage'] = (missing_val['count']/len(cab)*100)

#descending order
   missing_val = missing_val.sort_values('Missing_percentage', ascending = False).reset_index(drop
= True)
   print(missing_val)

# In[38]:
missingval(cab)

# In[24]:
#Impute with mean
# cab['pickup_longitude'] = cab['pickup_longitude'].fillna(cab['pickup_longitude'].mean())
# -73.85475529027732
#Impute with median
# cab['pickup_longitude'] = cab['pickup_longitude'].fillna(cab['pickup_longitude'].median())
# -73.9820495
#Apply KNN imputation algorithm
# cab = pd.DataFrame(KNN(k = 1).fit_transform(cab), columns = cab.columns)
# -73.890529

# In[39]:
#Since Mean is the best method, we impute missing values with mean
# cab['fare_amount'] = cab['fare_amount'].fillna(cab['fare_amount'].mean())
cab['pickup_longitude']= cab['pickup_longitude'].fillna(cab['pickup_longitude'].mean())
cab['pickup_latitude']= cab['pickup_latitude'].fillna(cab['pickup_latitude'].mean())
cab['dropoff_longitude']= cab['dropoff_longitude'].fillna(cab['dropoff_longitude'].mean())
cab['dropoff_latitude']= cab['dropoff_latitude'].fillna(cab['dropoff_latitude'].mean())


#for category variables we impute with mode
cab['passenger_count'] = cab['passenger_count'].fillna(int(cab['passenger_count'].mode()))


# In[40]:
missingval(cab)

# In[41]:
#Imputing the NAs in target variables will bias the model, hence remove them
cab=cab.dropna()

# In[42]:
convert_dic={'fare_amount' : 'float','passenger_count': 'int'}
cab=cab.astype(convert_dic)


# # Outlier Analysis
# In[21]:
#save it in another name, incase we need it again
df = cab.copy()
```

```
cab = df.copy()
# In[43]:


# fare_amount at the higher end are converted to na
cab.loc[cab['fare_amount']<0 , 'fare_amount']=np.nan
cab.loc[cab['fare_amount'] > 30, 'fare_amount']=np.nan
cab=cab.dropna()


# In[44]:


#there are few passenger counts that are greater than 8. convert them intoNAN
cab.loc[cab['passenger_count'] > 8,'passenger_count'] = np.nan


# In[45]:


#save numeric names
coutliers = [ 'pickup_longitude', 'pickup_latitude', 'dropoff_longitude', 'dropoff_latitude']
for list in coutliers:
    #Detect and replace with NA
    # #Extract quartiles
    q75, q25 = np.percentile(cab[list], [75 ,25])

    #Calculate IQR
    iqr = q75 - q25

    # #Calculate inner and outer fence
    minimum = q25 - (iqr*1.5)
    maximum = q75 + (iqr*1.5)

    # #Replace with NA
    cab.loc[cab[list] < minimum,list] = np.nan
    cab.loc[cab[list] > maximum,list] = np.nan

    # #Calculate missing value
    missing_val = pd.DataFrame(cab.isnull().sum())



# In[46]:


missingval(cab)


# In[47]:
```

```
#Since Mean is the best method, we impute missing values with mean

cab['pickup_longitude'] = cab['pickup_longitude'].fillna(cab['pickup_longitude'].mean())
cab['pickup_latitude'] = cab['pickup_latitude'].fillna(cab['pickup_latitude'].mean())
cab['dropoff_longitude'] = cab['dropoff_longitude'].fillna(cab['dropoff_longitude'].mean())
cab['dropoff_latitude'] = cab['dropoff_latitude'].fillna(cab['dropoff_latitude'].mean())

#for category variables we impute with mode
cab['passenger_count'] = cab['passenger_count'].fillna(int(cab['passenger_count'].mode()))


# In[48]:


missingval(cab)


# In[49]:


#first convert passenger_count to int to round to whole number and then to category
cab['passenger_count']=cab['passenger_count'].astype('int')
cab['passenger_count']=cab['passenger_count'].astype('category')


# In[50]:


cab['passenger_count'].unique()


# # Feature Selection

# In[51]:


# vectorized haversine function
def haversine(lat1, lon1, lat2, lon2, to_radians=True, earth_radius=6371):

    if to_radians:
        lat1, lon1, lat2, lon2 = np.radians([lat1, lon1, lat2, lon2])

    a = np.sin((lat2-lat1)/2.0)**2 +        np.cos(lat1) * np.cos(lat2) * np.sin((lon2-lon1)/2.0)**2

    return earth_radius * 2 * np.arcsin(np.sqrt(a))


# In[52]:
```

```
cab['dist'] =     haversine( cab['pickup_latitude'], cab['pickup_longitude'],
              cab['dropoff_latitude'], cab['dropoff_longitude'])


# In[53]:


##Correlation analysis
#Correlation plot
numeric=['fare_amount','pickup_longitude','pickup_latitude','dropoff_longitude','dropoff_latitud
e', 'dist']
cab_corr = cab.loc[:,numeric]


# In[54]:


#Set the width and hieght of the plot
f, ax = plt.subplots(figsize=(7, 5))

#Generate correlation matrix
corr = cab_corr.corr()
print(corr)
#Plot using seaborn library
sns.heatmap(corr, mask=np.zeros_like(corr, dtype=np.bool), cmap=sns.diverging_palette(220, 10,
as_cmap=True),
          square=True, ax=ax)


# In[55]:


#Before Creatinng the mode, Check for duplicates
cab1=cab[np.logical_and(cab['pickup_longitude']                ==                cab['dropoff_longitude'],
cab['pickup_latitude'] == cab['dropoff_latitude'])]
cab1


# In[ ]:


#This shows that there are no duplicate entries in the location


# # Model Development

# # Decision Tree

# In[56]
#Load libraries
from sklearn.model_selection import train_test_split
```

```
from sklearn.tree import DecisionTreeRegressor

# In[57]:

# Divide the data into train and test
train, test = train_test_split(cab, test_size=0.2)
# Already loaded

# In[63]:
# Decision tree for regression
fit_DT = DecisionTreeRegressor(max_depth=2).fit(train.iloc[:, 1:7], train.iloc[:,0])

# In[64]
fit_DT
# In[65]:


#Apply model on test data
predictions_DT = fit_DT.predict(test.iloc[:,1:7])



# In[66]:
#Calculate MAPE
def MAPE(y_true, y_pred):
   mape = np.mean(np.abs((y_true - y_pred) / y_true))*100
   return mape

# In[67]:
MAPE(test.iloc[:,0], predictions_DT)


# In[ ]:
#Error 29.71
# In[55]:
#Save_pred as csv
pd.DataFrame(predictions_DT, columns = ['DT_Predictions']).to_csv("DT_pred-Py5.csv", index =
False)
# # Random Fores
# In[68]:
#Random Forest
from sklearn.ensemble import RandomForestRegressor
# In[69]:
RF_model = RandomForestRegressor(n_estimators = 10).fit(train.iloc[:, 1:7], train.iloc[:,0])
# In[70]
RF_model
# In[71]:
RF_Predictions = RF_model.predict(test.iloc[:, 1:7])
# In[72]:
MAPE(test.iloc[:,0], RF_Predictions)
# In[ ]
```

```
#error is 25.71
#accuracy 75.29
# In[63]:
#Save_pred as csv
pd.DataFrame(RF_Predictions, columns = ['PF_Predictions']).to_csv("RF_pred-Py5.csv", index =
False)
# # KNN Imputation
# In[73]:
#KNN implementation
from sklearn.neighbors import KNeighborsRegressor
KNN_model = KNeighborsRegressor(n_neighbors = 1).fit(train.iloc[: , 1:7], train.iloc[:, 0])
# In[74]
#predict test cases
KNN_Predictions = KNN_model.predict(test.iloc[: , 1:7])
# In[75]:
MAPE(test.iloc[:,0], KNN_Predictions)
# In[ ]:
#error is 35.21
#accuracy is 65
# In[67]:
#Save_pred as csv
pd.DataFrame(KNN_Predictions, columns = ['KNN_Predictions']).to_csv("KNN_pred-Py5.csv",
index = False)
# # Linear Regression
# In[76]:
value=['fare_amount', 'pickup_longitude','pickup_latitude', 'dropoff_longitude', 'dropoff_latitude',
'dist']
# In[77]:
linear = cab[value]
# In[78]:
cat_names = ['passenger_count']
for i in cat_names:
    temp = pd.get_dummies(cab[i], prefix= i)
    linear = linear.join(temp)
# In[79]:
linear.head(2)
# In[80]:
#already loaded
train1, test1 = train_test_split(linear, test_size=0.2)
# In[81]:
#Import libraries for LR
import statsmodels.api as sm
# In[82]:
# Train the model using the training sets
model = sm.OLS(train1.iloc[:, 0], train1.iloc[:, 1:12]).fit()
# In[83]:
# Print out the statistics
model.summary()
# In[84]:
# make the predictions by the model
predictions_LR = model.predict(test1.iloc[:,1:31])
```

```
# In[85]:
#Calculate MAPE
MAPE(test1.iloc[:,0], predictions_LR)
# In[163]:
#MAPE 27.08
# In[164]:
train1.to_csv("py_LR_train15.csv", index = False)
test1.to_csv("py_LR_test15.csv", index= False)
# In[86]:
#Save_pred as csv
pd.DataFrame(predictions_LR, columns = ['LR_Predictions']).to_csv("LR_pred-Py_final.csv", index =
False)
# In[87]:
train.to_csv("py_LR_train1final.csv", index = False)
test.to_csv("py_LR_test1final.csv", index= False)
# In[88]:
# create a df
pre_val=pd.DataFrame(test['fare_amount']).reset_index(drop=True)
pre_val['DT']=(predictions_DT)
pre_val['RF']=(RF_Predictions)
pre_val['KNN']=(KNN_Predictions)
# In[89]:
pre_val.to_csv("Pred_val_pyfinal.csv", index= False)
# # Predict on new data
# In[90]:
pred=(pd.read_csv('test.csv', header = 0 )).drop(columns="pickup_datetime")
# In[91]:
#create Dist variable
pred['dist'] =     haversine( pred['pickup_latitude'], pred['pickup_longitude'],
              pred['dropoff_latitude'], pred['dropoff_longitude'])
pred['fare_amount']=0
pred['passenger_count']=pred['passenger_count'].astype('category')
# In[92]:
pred.head(5)
# In[100]:
cab.head(0)
# In[93]:
# Build model on the entire cab data
RF_model = RandomForestRegressor(n_estimators = 10).fit(cab.iloc[:, 1:7], cab.iloc[:,0])
#predict value
pred['fare_amount'] = RF_model.predict(pred.iloc[:, 0:6])
# In[94]:
#Download
pred.to_csv("test_Pred_val_pyfinal.csv", index= False)
```

Reference

- ✓ Websites:
  - www.edwisor.com
  - www.analyticsvidhya.com
  - www.tuitorialspoint.com
  - https://stackoverflow.com/questions/51488949/use-haversine-package-to-compare-all-distances-possibilities-of-a-csv-list-of-lo
  - https://www.r-bloggers.com/great-circle-distance-calculations-in-r/

- ✓ Books
  - R in action- Robert Kabarcoff