EDWISOR

Project: Prediction of fare amount for a cab ride in the city.

Submitted by: Anurag Pratap Singh Chauhan
Date: 18/05/2019

CONTENTS

1. INTRODUCTION

1.1Problem Statement

Our file "test_cab.csv" contains the fare of the cab that is charged along with the pickup and drop latitudes and longitudes. Also we are given the date and time of the travel so need to develop a model that would help us predict the fares for future. We need to predict the cab fares for our test file which is "test.csv".

1.2 Data

Our task is to build Regression model which will give the fare amount of the cab based on various factors such as distance , month of the year, week day, year. Given below is a sample of the data set that we are using to predict the cab fare.

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

And our test file for which the cab fare is to be predicted is as follows:

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

Below are the variables we used to predict the count of cab fare as per our modified data:

S.No	Variables
1	Passenger Count
2	Pickup DateTime
3	Pickup Latitude
4	Pickup longitude
5	Dropoff latitude
6	Dropoff Longitude

Keeping these variables in mind we are going to develop our model using various pre-processing techniques, predictive analysis and model evaluation to find the relationship between these variables and the target variable which is monthly rental bike counts.

2.METHODOLOGY

2.1 Pre-processing

Data preprocessing describes any type of processing performed on raw data to prepare it for another processing procedure. Commonly used as a preliminary data mining practice, data preprocessing transforms the data into a format that will be more easily and effectively processed for the purpose of the user.

It refers to exploring the data, cleaning the data as well as visualizing the data through graphs and plots. This is often called as Exploratory Data Analysis. To start this process we will first try and look at all the distributions of the Numeric variables. Most analysis like regression, require the data to be normally distributed.

2.1.1 Missing Value Analysis

Missing values in data is a common phenomenon in real world problems. Knowing how to handle missing values effectively is a required step to reduce bias and to produce powerful models.

Below table illustrates that missing values are present in the data.

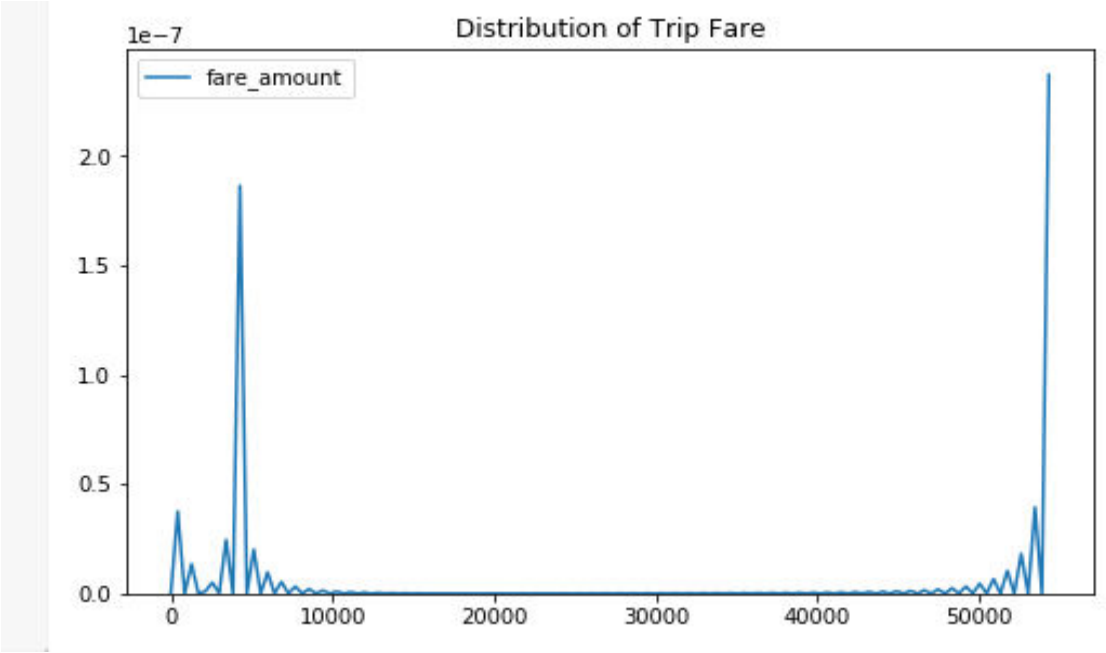
	Variables	Missing_percentage
0	passenger_count	0.342338
1	fare_amount	0.149384
2	pickup_datetime	0.000000
3	pickup_longitude	0.000000
4	pickup_latitude	0.000000
5	dropoff_longitude	0.000000
6	dropoff_latitude	0.000000

FIG 2.9: Missing Value analysis.

This missing data has to be taken care of since there are very high undulations in the data in few of the data sets, we would go for **median imputation** so that the median values take up the missing places.

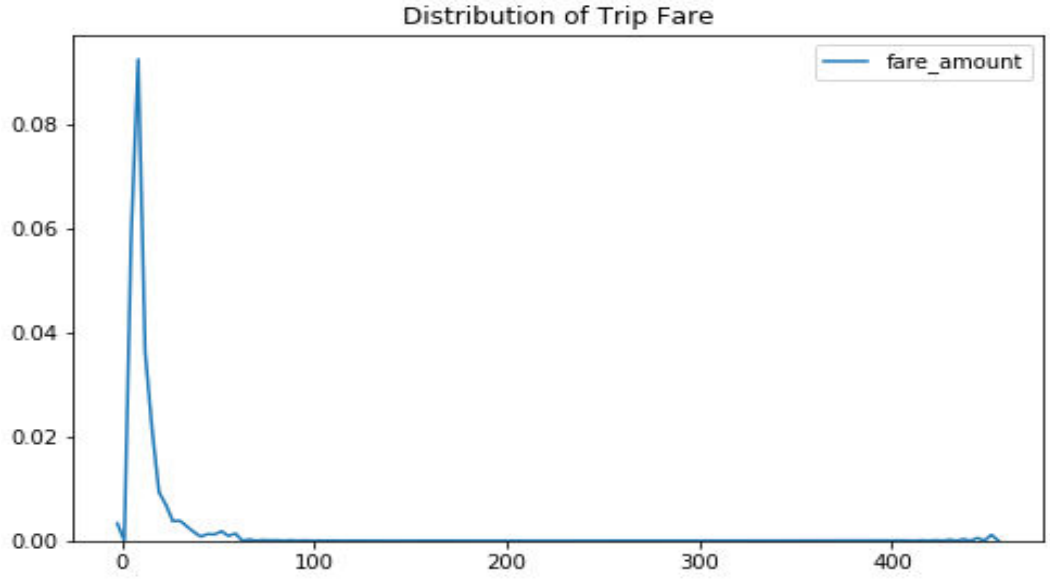
2.1.2 Outlier Analysis

In statistics, an outlier is an observation point that is distant from other observations. An outlier may be due to variability in the measurement or it may indicate experimental error; the latter are sometimes excluded from the data set. An outlier can cause serious problems in statistical analyses. Since most of the variables are categorical therefore we have removed the outliers taking different independent variables one at a time.



Above shown is the distribution of the trip fare, clearly we can see some outliers here, therefore we have removed the negative values from the data set. Also we have taken care of the fares greater than 500 by removing them.

The following is the trip fare distribution after treating the outliers.



Apart from the fare amount pickup and drop-off latitudes and longitudes also have some incorrect data sets therefore we need to remove them too.
The figure below shows us the values of the pickup and drop-off latitudes and longitudes as follows

```
Range of Pickup Latitude is  (-74.006893, 401.083332)
Range of Dropoff Latitude is  (-74.006377, 41.366138)
Range of Pickup Longitude is  (-74.438233, 40.766125)
Range of Dropoff Longitude is  (-74.42933199999999, 40.802437)
```

:

So after removing the outlying values which we have analyzed using the filters option in the "test.csv" file (by using the filters option) we have new range of the given latitudes and longitudes.

```
Range of Pickup Latitude is  (40.573143, 41.709555)
Range of Dropoff Latitude is  (40.568973, 41.696683)
Range of Pickup Longitude is  (-74.252193, -72.986532)
Range of Dropoff Longitude is  (-74.263242, -72.990963)
```

Passenger count also shows some very high values such as 5435 in many cases so we have removed the value greater than 6 as a cab cannot accommodate more than 6 people at a time. There were 6 such data sets therefore we can get rid of them.

2.1.3 Conversion of independent factors

A) Pickup_datetime to its components.

The first step to analyze how the fares have changed over time, is to create features like hour, day of the week, day, month, year from pickup datetime. The code to extract these features is as below:

```
#Create datetime features based on pickup_datetime
train['pickup_date']= train['pickup_datetime'].dt.date
train['pickup_day']=train['pickup_datetime'].apply(lambda x:x.day)
train['pickup_hour']=train['pickup_datetime'].apply(lambda x:x.hour)
train['pickup_day_of_week']=train['pickup_datetime'].apply(lambda x:calendar.day_name[x.weekday()])
train['pickup_month']=train['pickup_datetime'].apply(lambda x:x.month)
train['pickup_year']=train['pickup_datetime'].apply(lambda x:x.year)
```

After conversion we can relate the relationship between various independent factors also we can relate these parameters with our dependent variable which is cab fare. For example, we have

shown couple of plots below:

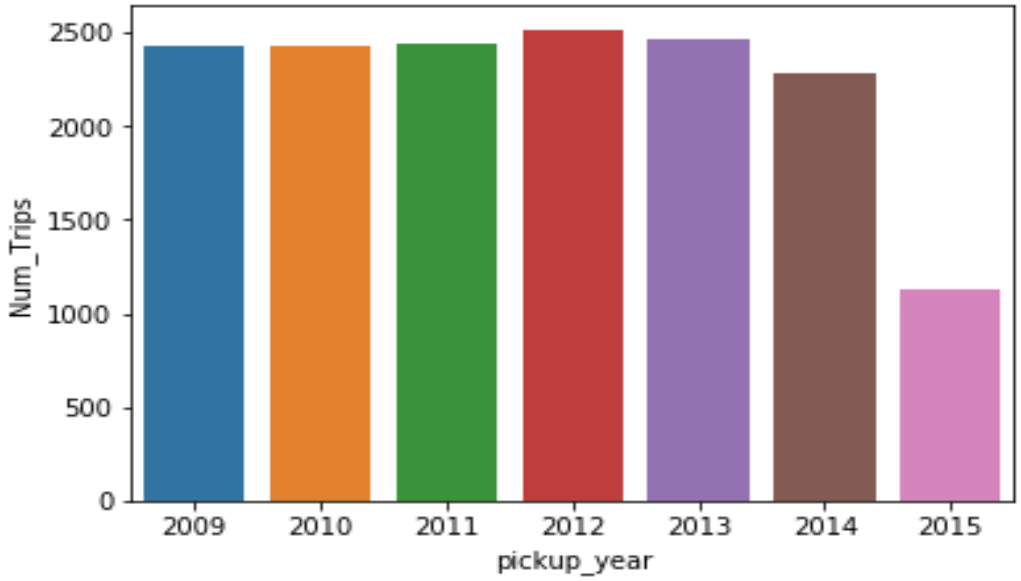


Fig: Number of trips vs Pickup Year

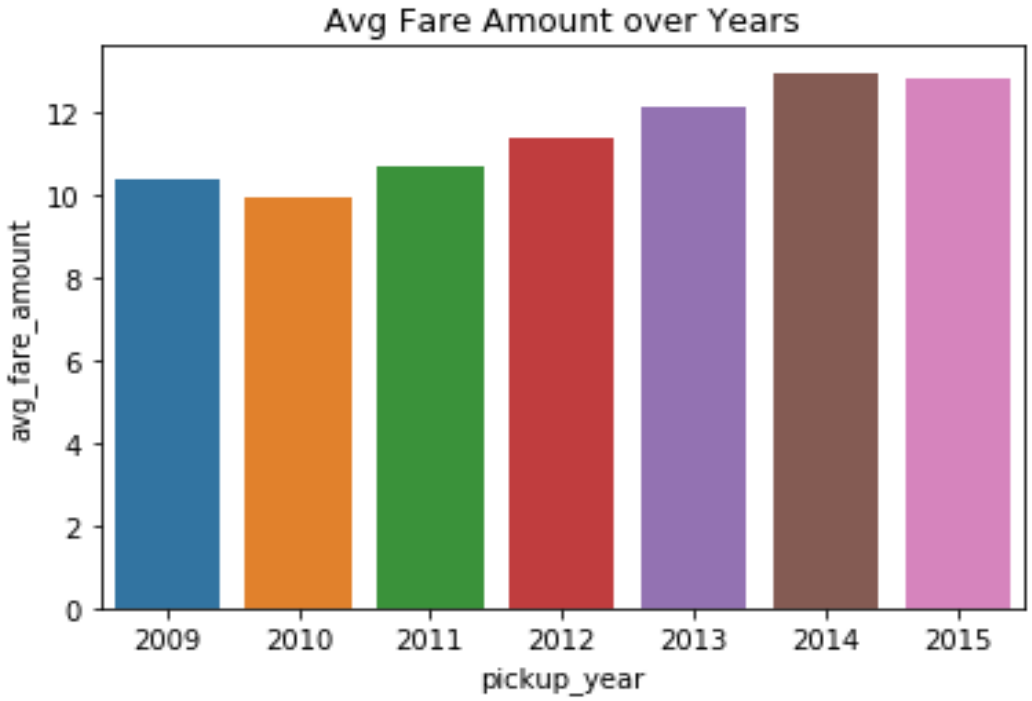


Fig: Average fare amount vs Pickup Year

B) Pickup and drop-off latitudes and longitudes to Distance.

Using the pickup and drop-off coordinates we calculate the trip distance in miles based on **Haversine Distance.** We will calculte the total distance travelled on the sphere using the haversine distance/ great circle distance.
Formulas used to calculate(phi = latitudes of the 2 pts, lambda = longitudes): haversine(θ) = $\sin^2(\theta/2)$
$a = \sin^2(\varphi B - \varphi A/2) + \cos \varphi A * \cos \varphi B * \sin^2(\lambda B - \lambda A/2)$
$c = 2 * \text{atan2}(\sqrt{a}, \sqrt{(1-a)})$
$d = R * c$

After the above two conversions we have the following new variables in the data set:

pickup_date	pickup_day	pickup_hour	pickup_day_of_week	pickup_month	pickup_year	H_Distance
2011-02-11	11	21	Friday	2	2011	129.950482
2011-04-03	3	12	Sunday	4	2011	129.560455
2011-05-27	27	21	Friday	5	2011	127.509261
2011-04-26	26	23	Tuesday	4	2011	123.561157
2009-02-25	25	7	Wednesday	2	2009	101.094619
2009-05-02	2	19	Saturday	5	2009	99.771579

2.1.4 Feature Selection

Before performing any type of modeling 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 class prediction. There are several methods of doing it.
Machine learning works on a simple rule – if you put garbage in, you will only get garbage to come out. By garbage here, I mean noise in data.
This becomes even more important when the number of features are very large. You need not use every feature at your disposal for creating an algorithm. You can assist your algorithm by feeding in only those features that are really important. I have myself witnessed feature subsets giving better results than complete set of feature for the same algorithm or – "Sometimes, less is better!".

We should consider the selection of feature for model based on below criteria

i. The relationship between two independent variable should be less and

ii. The relationship between Independent and Target variables should be high.

Below figure illustrates that relationship between all numeric variables using Corrgram plot.

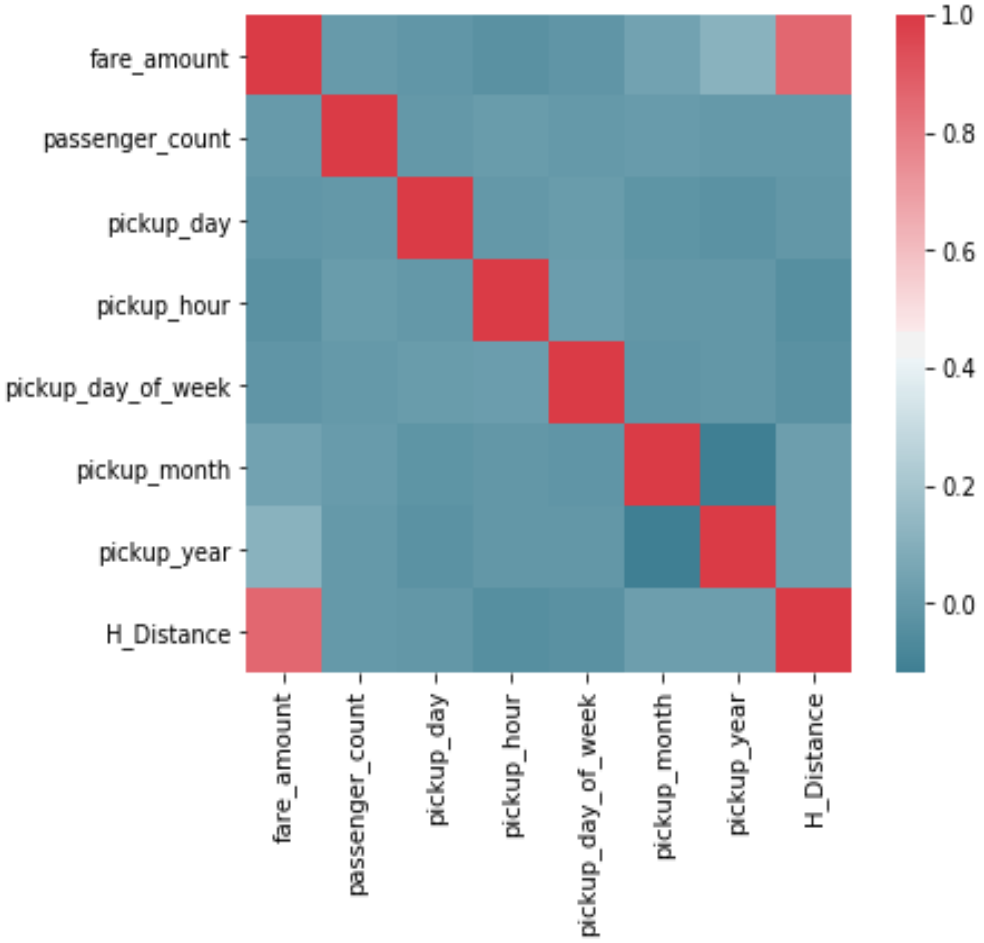


FIG 2.11 Correlation plot of numeric variables

The following inference can be made from the above correlation plot.

a) none of the dependent variables are negatively correlated with each other or with the dependent variable.

b) H_distance is highly related with the fare_amount.

c) there is no multi- co linearity between the independent variables. Hence we would not go for dimensional reduction we will consider all the variables in our analysis.

2.1.5 Feature Scaling

The word "normalization" is used informally in statistics, and so the term normalized data can have multiple meanings. In most cases, when you normalize data you eliminate the units of measurement for data, enabling you to more easily compare data from different places. Some of the more common ways to normalize data include:

Rescaling data to have values between 0 and 1. This is usually called feature scaling. One possible formula to achieve this is.

$$x_{new} = \frac{x - x_{min}}{x_{max} - x_{min}}$$

But in our data set we have no such requirement as the data we have is in proper limits and we need to scale the data, as it could affect our model adversely.

2.2.Modelling

2.2.1 Model Selection

In out earlier stage of analysis we have come to understand that few variables like "Year", "Haversine Distance" , "hour of the day", "weekday" are going to play key role in model development , for model development dependent variable may fall under below categories

i. Nominal
ii. Ordinal
iii. Interval
iv. Ratio

In our case dependent variable is interval so, the predictive analysis that we can perform is Regression Analysis

We will start our model building from Multiple Linear Regression

2.2.2 Evaluating Regression Model

The main concept of looking at what is called **residuals** or difference between our predictions f(x[I,]) and actual outcomes y[i].

We are using two methods to evaluating performance of our model.

i. **MAPE** : (Mean Absolute Percent Error) measures the size of the error in percentage terms. It is calculated as the average of the unsigned percentage error.

$$\left(\frac{1}{n}\sum\frac{|Actual - Forecast|}{|Actual|}\right)*100$$

ii. **RMSE :**(Root Mean Square Error) is a frequently used measure of the difference between values predicted by a model and the values actually observed from the environment that is being modelled. We are going to use RMSE for our model evaluation because our data is time based.

$$RMSE = \sqrt{\frac{\sum_{i=1}^{n}(X_{obs,i} - X_{model,i})^2}{n}}$$

2.2.3 Linear Regression

Multiple linear regression is the most common form of linear regression analysis. As a predictive analysis, the multiple linear regressions are used to explain the relationship between one continuous dependent variable and two or more independent variables. The independent variables can be continuous or categorical.

As Linear regression will work well if multicollinearity between the Independent variables are less.

```
In [70]: X = train.drop(['fare_amount','pickup_datetime','pickup_date'],axis=1)
         y = train['fare_amount']

In [71]: #Split train set into test and train subsets
         X_train, X_test, y_train, y_test = train_test_split(X,y, test_size=0.2)

         #Drop columns from test dataset we're not going to use
         test_pred = test.drop(['pickup_datetime','pickup_date'],axis=1)

In [37]: ##LINEAR REGESSION MODEL
         #Initilise a linear regression model, fit the data and get scores
         lm = LinearRegression()
         lm.fit(X_train,y_train)
         print(lm.score(X_train,y_train))
         print(lm.score(X_test,y_test))

         0.7573114950538957
         0.7630316704972073

In [38]: #Predict fares and get a rmse for them
         y_pred = lm.predict(X)
         lrmse = np.sqrt(metrics.mean_squared_error(y_pred, y))
         lrmse
         #4.663127011970703 whcih is quite good

Out[38]: 4.663127011970703
```

FIG 3.7 : Development of Linear Regression Model.

2.2.4 Decision Tree

A tree has many analogies in real life, and turns out that it has influenced a wide area of **machine learning**, covering both **classification and regression**. In decision analysis, a decision tree can be used to visually and explicitly represent decisions and decision making. As the name goes, it uses a tree-like model of decisions.

```
: ##DECISION TREE REGRESSOR
  fit_DT = DecisionTreeRegressor(max_depth=8)
  fit_DT.fit(X_train,y_train)
```

```
: DecisionTreeRegressor(criterion='mse', max_depth=8, 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')
```

```
: #Predict fares and get a rmse for them
  predictions_DT = fit_DT.predict(X)
  lrmse = np.sqrt(metrics.mean_squared_error(predictions_DT, y))
  lrmse
  #3.4871212085852865 better than linear regression model.
```

```
: 3.4871212085852865
```

```
: #Predict final fares for test data
  DT_Predictions = fit_DT.predict(test_pred)
  DT_Predictions = np.round(DT_Predictions, decimals=2)

  #Set up predictions for a submittable dataframe
  DT_submission = pd.DataFrame({"fare_amount": DT_Predictions},columns = ['fare_amount'])
  DT_submission
```

FIG 3.1 Decision Tree Model

2.2.5 Random Forest

Random forests or random decision forests are an ensemble learning method for classification, regression and other tasks, that operate by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees.

In [81]:
```
### RANDOM FOREST

rf = RandomForestRegressor(n_estimators = 100, random_state = 800,n_jobs=-1)
rf.fit(X_train,y_train)
rf_pred= rf.predict(X_test)
rf_rmse=np.sqrt(metrics.mean_squared_error(rf_pred, y_test))
print("RMSE for Random Forest is ",rf_rmse)

##RMSE for Random Forest is  4.148241041585081. Multiple linear regression and decision tree form better m
```

```
RMSE for Random Forest is  4.187533080652855
```

In [83]:
```
#Predict final fares for test data
RF_Predictions = rf.predict(test_pred)
RF_Predictions = np.round(RF_Predictions, decimals=2)

#Set up predictions for a submittable dataframe
RF_Predictions = pd.DataFrame({"fare_amount": RF_Predictions},columns = ['fare_amount'])
RF_Predictions
```

FIG 3.5 Development of Random Forest

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 Root Mean Squared Error because it is time based data and RMSE is suitable for such data types.

We have calculated the RMSE for all the three models and we will compare them to select the best model for cab rental prediction.

a) Multiple Linear Regressions

```
#Predict fares and get a rmse for them
y_pred = lm.predict(X)
lrmse = np.sqrt(metrics.mean_squared_error(y_pred, y))
lrmse
#4.663127011970703 whcih is quite good
```

```
#Predict final fares for test data
LinearPredictions = lm.predict(test_pred)
LinearPredictions = np.round(LinearPredictions, decimals=2)
LinearPredictions
```

```
array([10.48, 10.01,  5.4 , ..., 48.94, 22.74,  7.24])
```

RMSE value for multiple linear regressions is 4.663 and the predicted values for our test data is shown in figure above.

b) Decision Tree

```
#Predict fares and get a rmse for them
predictions_DT = fit_DT.predict(X)
lrmse = np.sqrt(metrics.mean_squared_error(predictions_DT, y))
lrmse
#3.4871212085852865 better than linear regression model.
```

```
#Predict final fares for test data
DT_Predictions = fit_DT.predict(test_pred)
DT_Predictions = np.round(DT_Predictions, decimals=2)

#Set up predictions for a submittable dataframe
DT_submission = pd.DataFrame({"fare_amount": DT_Predictions},columns = ['fare_amount'])
DT_submission
```

	fare_amount
0	8.91
1	8.91
2	5.05
3	9.81
4	20.53
5	9.98
6	5.87
7	43.12
8	10.59
9	5.87
10	8.91

RMSE value for Decision tree is 3.48 and the predicted cab fare values for the test file are shown above.

Also Decision tree model has least RMSE value.

c) Random Forest Regression

```
rf_rmse=np.sqrt(metrics.mean_squared_error(rf_pred, y_test))
print("RMSE for Random Forest is ",rf_rmse)

##RMSE for Random Forest is  4.148241041585081. Multiple linear regression and decisio
```

```
#Set up predictions for a submittable dataframe
RF_Predictions = pd.DataFrame({"fare_amount": RF_Predictions},columns = ['fare_amount'])
RF_Predictions
```

51]:

	fare_amount
0	10.36
1	9.97
2	4.53
3	9.44
4	18.82
5	9.84
6	6.19
7	48.98
8	11.10
9	6.33
10	10.03

RMSE value for Random Forest is 4.14 and the predicted cab fare values for the test file are shown above.

3.2 Model Selection

We can see that both models perform comparatively on average and therefore we can select any of the three models without any loss of information.

References

- **WWW.EDWISOR.COM**
- **WWW.ANALTICSVIDHYA.COM**
- **WWW.GOOGLE.COM**
- **WWW.YOUTUBE.COM**