EDWISOR

PROJECT REPORT

Cab Fare Prediction

Author:
Alok Misra

A report submitted in fulfillment of the requirements for the degree of Data Science Certification

in the

February 12, 2020

Declaration of Authorship

I, Alok Misra, declare that this thesis titled, “Cab Fare Prediction ” and the work presented in it are my own. I confirm that:
Signed:

Date:

"Thanks to my solid academic training, today I can write hundreds of words on virtually any topic without possessing a shred of information, which is how I got a good job in journalism."

Dave Barry

EDWISOR

Abstract

Data Science Certification

Cab Fare Prediction

by Alok Misra

Predictive analytics uses archival data to predict the future events. Typically, past data is used to build a mathematical model that captures important trends. That predictive model is then used on current data to predict the future or to suggest actions to take for optimal outcomes. Predictive analytics has received a lot of attention in recent years due to advances in supporting technology, particularly in the areas of big data and machine learning. Companies also use predictive analytics to create more accurate forecasts, such as forecasting the fare amount for a cab ride in the city. These forecasts enable resource planning for instance, scheduling of various cab rentals to be done more effectively. For a cab rental start-up company, the fare amount is dependent on a lot of factors. This research aims to understand all patterns and to apply analytics for fare prediction. The proposed work is to design a system that predicts the fare amount for a cab ride in the city. The aim is to build regression models, which will predict the continuous fare amount for each cab ride and help prediction depending on multiple time-based, positional and general factors.

Acknowledgements

I would also like to thank all of my friends who supported me in writing, and incented me to strive towards my goal. At the end I would like express appreciation to my beloved wife Mamta who spent sleepless nights with and was always my support in the moments when there was no one to answer my queries.

Contents

List of Figures

List of Tables

1 Introduction

In any city taxi rides paint a vibrant picture of life in the city. The millions of rides taken each month can provide insight into traffic patterns, road blockage, or large-scale events that attract many people. With ridesharing apps gaining popularity, it is increasingly important for taxi companies to provide visibility to their estimated fare and ride duration, since the competing apps provide these metrics upfront. Predicting fare and duration of a ride can help passengers decide when is the optimal time to start their commute, or help drivers decide which of two potential rides will be more profitable, for example. Furthermore, this visibility into fare will attract customers during times when ridesharing services are implementing surge pricing. In order to predict duration and fare, only data which would be available at the beginning of a ride was used. This includes pickup and dropoff coordinates, trip distance, start time, number of passengers, and a rate code detailing whether the standard rate or the airport rate was applied. Linear regression with model selection, lasso, and random forest models were used to predict duration and fare amount.

In current scenarios cab rental services are expanding with the multiplier rate. The ease of using the services and flexibility gives their customer a great experience with competitive prices. Machine learning (ML) is closely related to computational statistics, which focuses on making predictions using computers. Data mining (DM) is a field of study within ML and focuses on exploratory data analysis through**book-minimal** unsupervised learning. In its application across business problems, machine learning is also referred to as predictive analytics. Machine learning tasks are classified into several broad categories. In supervised learning, the algorithm builds a mathematical model from a set of data that contains both the inputs and the desired outputs. Classification algorithms and regression algorithms are examples of supervised learning Regression algorithms are named for their continuous outputs, meaning they may have any value within a range **book-minimal**. In unsupervised learning, the algorithm builds a mathematical model from a set of data that contains only inputs and no desired output labels. Unsupervised learning algorithms are used to find structure in the data, like grouping or clustering of data points. Unsupervised learning can discover patterns in the data and can group the inputs into categories, as in feature learning. Dimensionality reduction is the process of reducing the number of "features", or inputs, in a set of data. Machine learning and data mining often employ the same methods and overlap significantly, but while ML focuses on prediction, based on known properties learned from the training data, data mining focuses on the discovery of (previously) unknown properties in the data. This is the analysis step of knowledge discovery in databases (KDD) [1]. DM uses many ML methods, but with different goals; on the other hand, ML also employs data mining methods as "unsupervised learning" or as a Raschka, 2016preprocessing step to improve learner accuracy.

1.1 Problem Statement

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

1.2 Data

The aim is to build regression modelsHawthorn, Weber, and Scholten, 2001 that will predict the continuous fare amount for each of the cab-rides depending on multiple time-based, positional and generic factors. This problem statement falls under the category of forecasting which deals with predicting continuous values for the future (the continuous value is the fare amount of the cab ride).Fig.1 shows a sample of the data set[2] that will be used to predict the fare amount of a cab ride. There are six predictor variables and one target variable which are listed as follows: Predictors:

1. $Pickup_datetime : timestampvalueindicatingwhenthecabridestarted$
2. $Pickup_longitude : floatforlongitudecoordinateofwherethecabridestarted.$
3. $Pickup_latitude : floatforlatitudecoordinateofwherethecabridestarted.$
4. $Dropoff_longitude : floatforlongitudecoordinateofwherethecabrideended$
5. $Dropoff_latitude : floatforlatitudecoordinateofwherethecabrideended.$
6. $Passenger_count : anintegerindicatingthenumberofpassengersinthecabride.$

2 Methodology

In any city taxi rides paint a vibrant picture of life in the city. The millions of rides taken each month can provide insight into traffic patterns, road blockage, or large-scale events that attract many people. With ridesharing apps gaining popularity, it is increasingly important for taxi companies to provide visibility to their estimated fare and ride duration, since the competing apps provide these metrics upfront. Predicting fare and duration of a ride can help passengers decide when is the optimal time to start their commute, or help drivers decide which of two potential rides will be more profitable, for example. Furthermore, this visibility into fare will attract customers during times when ridesharing services are implementing surge pricing. In order to predict duration and fare, only data which would be available at the beginning of a ride was used. This includes pickup and dropoff coordinates, trip distance, start time, number of passengers, and a rate code detailing whether the standard rate or the airport rate was applied. Linear regression with model selection, lasso, and random forest models were used to predict duration and fare amount.

In current scenarios cab rental services are expanding with the multiplier rate. The ease of using the services and flexibility gives their customer a great experience with competitive prices.

2.1 Pre-Processing

When we required to build a predictive model, we require to look and manipulate the data before we start modelling which includes multiple preprocessing"Data Science course" steps such as exploring the data, cleaning the data as well as visualizing the data through graph and plots, all these steps is combined under one shed which is Exploratory Data Analysis, which includes following steps:

1. $DataexplorationandCleaning$
2. $Missingvaluestreament$
3. $Outlier Analysis$
4. $FeatureSelection$
5. $FeaturesScaling$
   (a) Skewness
   (b) Log transformation
6. Visualization

2.2 Modelling

Once all the Pre-Processing steps has been done on our data set, we will now further move to our next step which is modelling. Modelling plays an important role to find out the good inferences from the data. Choice of models depends upon the problem statement and data set. As per our problem statement and dataset, we will try some models on our preprocessed data and post comparing the output results we will select the best suitable model for our problem. As per our data set following models need to be tested:

1. Linear regression
2. Decision Tree
3. Random forest
4. Gradient Boosting

We have also used hyper parameter tunings to check the parameters on which our model runs best. Following are two techniques of hyper parameter tuning we have used:

1. Random Search CV
2. Grid Search CV

3 Data Pre-Processing:Pre-Processing

3.1 Data exploration and Cleaning (Missing Values and Outliers)

The very first step which comes with any data science project is data exploration and cleaning which includes following points as per this project:

- Separate the combined variables.
- As we know we have some negative values in fare amount so we have to remove those values.
- Passenger count would be max 6 if it is a SUV vehicle not more than that. We have to remove the rows having passengers counts more than 6 and less than 1.
- There are some outlier figures in the fare (like top 3 values) so we need to remove those.
- Latitudes range from -90 to 90. Longitudes range from -180 to 180. We need to remove the rows if any latitude and longitude lies beyond the ranges

3.2 Creating new variables from the given variables.

Here in our data set our variable name$pickup_datetime$ contains date and time for pickup. So we tried to extract some important variables from $pickup_datetime$:

- Year
- $Month$
- $Date$
- $DayofWeek$
- $Hour$
- $Minute$

Also, we tried to find out the distance using the haversine formula which says: The haversine formula determines the great-circle distance between two points on a sphere given their longitudes and latitudes. Important in navigation, it is a special case of a more general formula in spherical trigonometry, the law of haversines, that relates the sides and angles of spherical triangles as shown in figure.

So our new extracted variables are:

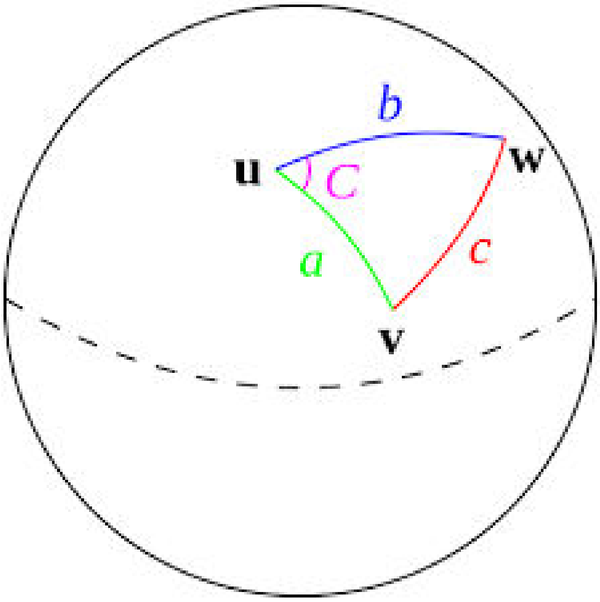

FIGURE 3.1: harvesine

1. $fare_amount$
2. $pickup_datetime$
3. $pickup_longitude$
4. $pickup_latitude$
5. $dropoff_longitude$
6. $dropoff_latitude$
7. $passenger_count$
8. $year$
9. $Month$
10. $Date$
11. $DayofWeek$
12. $Hour$

The formula used to calculate great-circle distance between two points on a sphere given their longitudes and latitudes is shown in the figure below.

3.3 Dropping variables from the given variables.

Now as we know that all the following variables are of no use so we will drop the redundant variables as following:

1. *pickup_datetime*
2. *pickup_longitude*

$$a = \sin^2\left(\frac{\Delta\varphi}{2}\right) + \cos\varphi 1 \cdot \cos\varphi 2 \cdot \sin^2\left(\frac{\Delta\lambda}{2}\right)$$

$$c = 2 \cdot \operatorname{atan2}\left(\sqrt{a}, \sqrt{(1-a)}\right)$$

$$d = R \cdot c$$

FIGURE 3.2: formula to calculate distance

3. $pickup\_latitude$
4. $dropoff\_longitude$
5. $dropoff\_latitude$
6. $Minute$

3.4 Final variables from the given variables

Variable List	
Variable Names	Variable data Types
fare_amount	float64
passenger_count	object
year	object
month	object
date	object
Dayofweek	object
Hour	object
distance	object

3.5 Some more data exploration

In this report we are trying to predict the fare prices of a cab rental company. So here we have a data set of 16067 observations with 8 variables including one dependent variable.

3.5.1 independent variables

Below are the names of Independent variables:

1. $passenger_{c}ount$
2. $year$
3. $Month$

4. *Date*
5. *DayofWeek*
6. *Hour*
7. *distance*

3.5.2 dependent variables

Our Dependent variable is

- $fare\_amount$

3.5.3 Uniqueness of a variables

We need to look at the unique number in the variables which help us to decide whether the variable is categorical or numeric. So, by using python script 'nunique' we tried to find out the unique values in each variable. We have also added the table below:

Uniqueness in Variable List	
Variable Names	Unique count
fare_amount	450
passenger_count	7
year	7
month	12
date	31
Dayofweek	7
Hour	24
distance	15424

3.5.4 Dividing the variables based on their datatypes

Continous

Following are the continous variables

- $fare\_amount$
- distance

Categorical

Following are the categorical variables

- $passenger_count$
- *year*
- *Month*
- *Date*
- *DayofWeek*
- *Hour*

3.6 Feature Scaling

Skewness is asymmetry in a statistical distribution, in which the curve appears distorted or skewed either to the left or to the right. Skewness can be quantified to define the extent to which a distribution differs from a normal distribution. Here we tried to show the skewness of our variables and we find that our target variable absenteeism in hours having is one sided skewed so by using log transform technique we tried to reduce the skewness of the same. Below mentioned graphs shows the probability distribution plot to check distribution before log transformation:

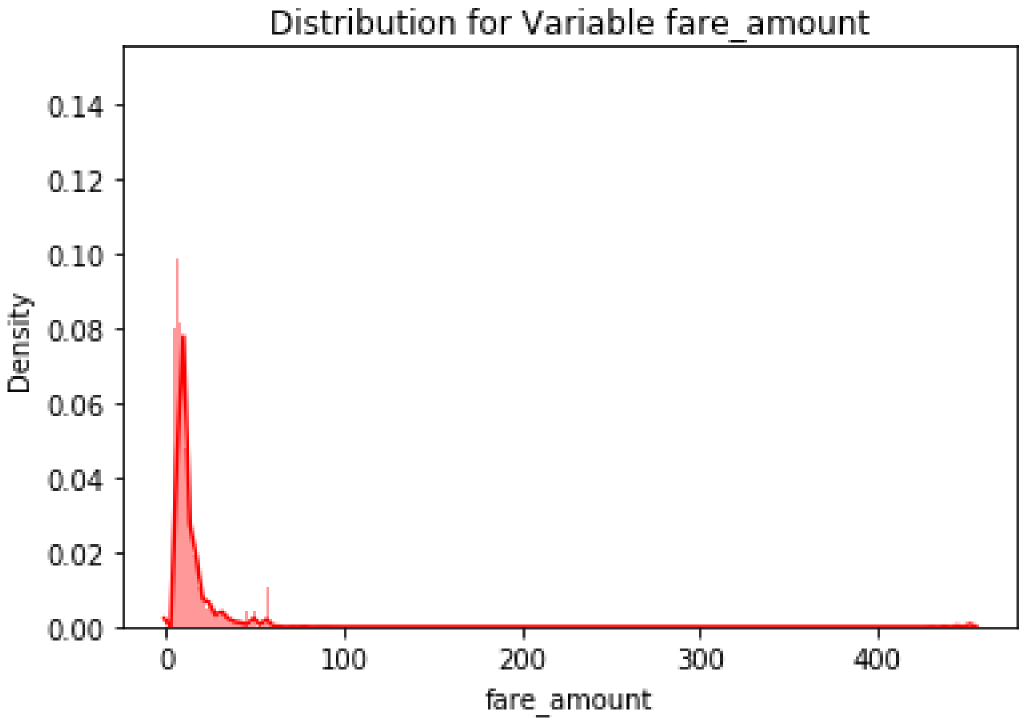

FIGURE 3.3: featurescalingfare

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

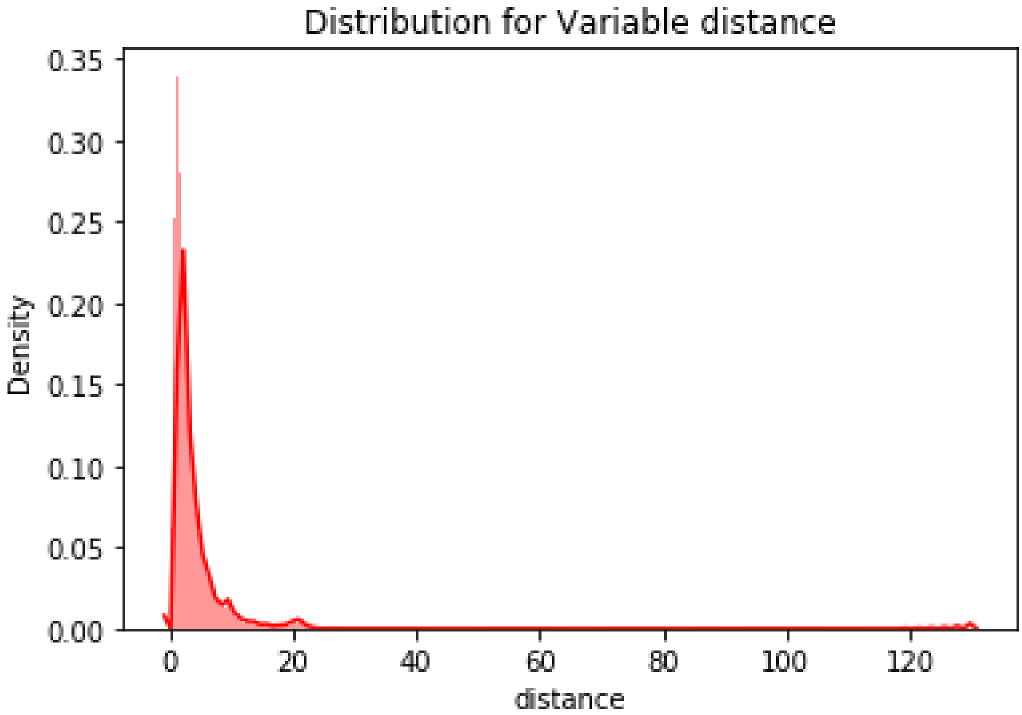

FIGURE 3.4: featurescalingdistance

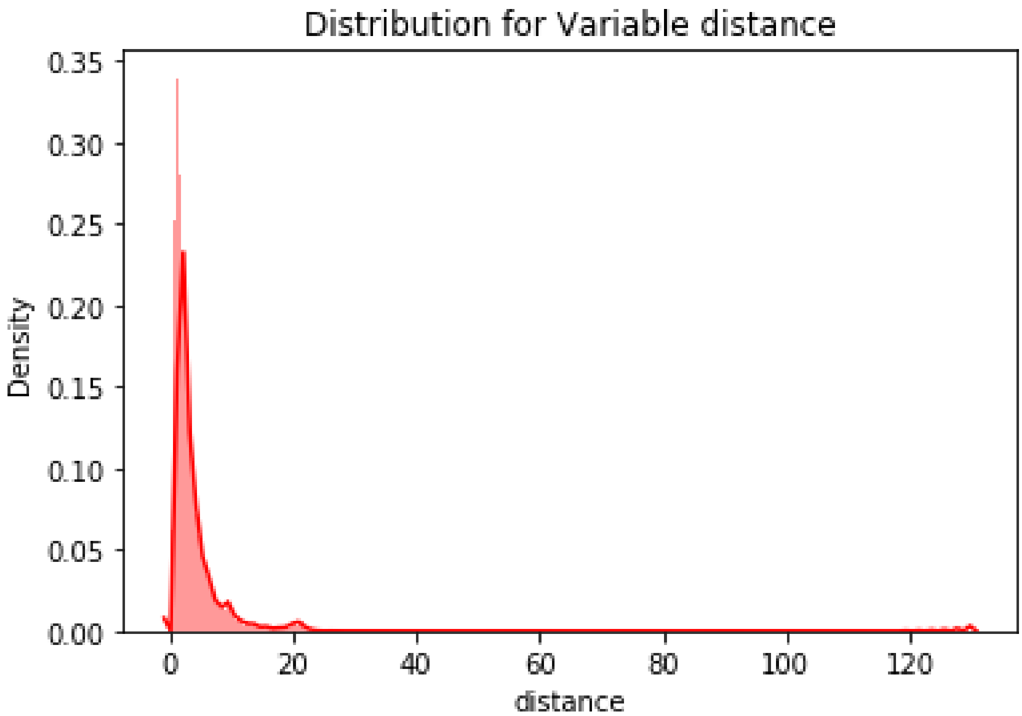

FIGURE 3.5: correctedfeaturescalingdistance

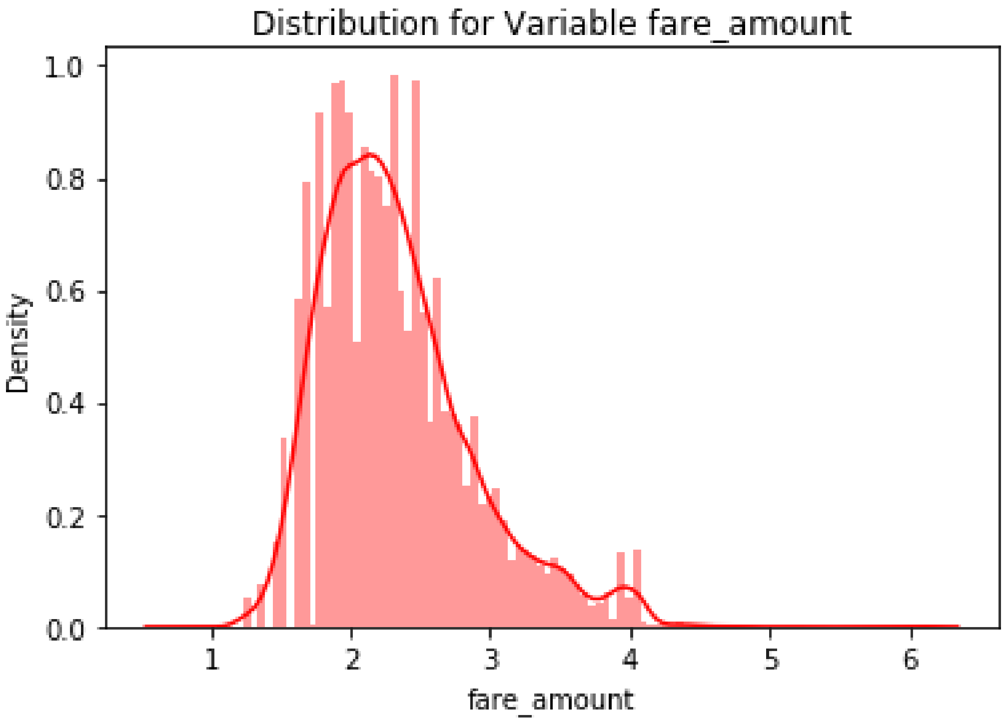

FIGURE 3.6: correctedfeaturescalingfare

4 Modelling

After a thorough pre processing, we will use some regression models on our processed data to predict the target variable. Following are the models which we have built –

- Linear Regression
- Decision Tree
- Random Forest
- Gradient Boosting

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

We need to split our train data into two parts

```
In [56]: from sklearn.tree import DecisionTreeRegressor
         from sklearn.metrics import mean_squared_error

In [60]: ##train test split for further modelling
         X_train, X_test, y_train, y_test = train_test_split( train_df.iloc[:, train_df.columns != 'fare_amount'],
                                  train_df.iloc[:, 0], test_size = 0.20, random_state = 1)

In [61]: print(X_train.shape)
         print(X_test.shape)

         (12339, 7)
         (3085, 7)
```

FIGURE 4.1: traintestsplit

4.1 Linear Regression

Multiple linear regression is the most common form of linear regression analysis. Multiple regression is an extension of simple linear regression. It is used as a predictive analysis, when we want to predict the value of a variable based on the value of two or more other variables. The variable we want to predict is called the dependent variable (or sometimes, the outcome, target or criterion variable).

Below is a screenshot of the model we build and its output:

4.2 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 **inproceedings-full**. In

1. Linear Regeression Model

```
In [62]: # Importing libraries for Linear Regression
         from sklearn.linear_model import LinearRegression

In [63]: # Building model on top of training dataset
         fit_LR = LinearRegression().fit(X_train , y_train)

In [64]: #prediction on train data
         pred_train_LR = fit_LR.predict(X_train)

In [65]: #prediction on test data
         pred_test_LR = fit_LR.predict(X_test)

In [66]: ##calculating RMSE for test data
         RMSE_test_LR = np.sqrt(mean_squared_error(y_test, pred_test_LR))

In [67]: print("Root Mean Squared Error For Test data = "+str(RMSE_test_LR))

         Root Mean Squared Error For Test data = 0.2503511796785927

In [68]: from sklearn.metrics import r2_score
         #calculate R^2 for train data
         r2_score(y_train, pred_train_LR)

Out[68]: 0.746855951097612

In [69]: r2_score(y_test, pred_test_LR)
Out[69]: 0.7778537029821875
```

FIGURE 4.2: Linear Regression

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.

Below is the screenshot of the query we executed and the result shown, we will compare the results of each model in a combined table later on.

Decision Tree Algorithm

2. Decision Tree Model

```
In [71]: fit_DT = DecisionTreeRegressor(max_depth = 2).fit(X_train,y_train)

In [72]: #prediction on train data
         pred_train_DT = fit_DT.predict(X_train)

         #prediction on test data
         pred_test_DT = fit_DT.predict(X_test)

In [73]: ##calculating RMSE for train data
         RMSE_train_DT = np.sqrt(mean_squared_error(y_train, pred_train_DT))

         ##calculating RMSE for test data
         RMSE_test_DT = np.sqrt(mean_squared_error(y_test, pred_test_DT))

In [74]: print("Root Mean Squared Error For Training data = "+str(RMSE_train_DT))
         print("Root Mean Squared Error For Test data = "+str(RMSE_test_DT))

         Root Mean Squared Error For Training data = 0.301206387471297796
         Root Mean Squared Error For Test data = 0.28969521517125973

In [75]: ## R^2 calculation for train data
         r2_score(y_train, pred_train_DT)

Out[75]: 0.70012186420221

In [76]: ## R^2 calculation for test data
         r2_score(y_test, pred_test_DT)

Out[76]: 0.7025442022580384
```

FIGURE 4.3: Decision Tree

4.3 Random Forest

Random forests or random decision forests are an ensemble learning method for classification, regression and other task, 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. Random decision forests correct for decision trees' habit of overfitting to their training set.

To say it in simple words: Random forest builds multiple decision trees and merges them together to get a more accurate and stable prediction.

3. Random Forest Model

```
In [77]: # Importing libraries for Random Forest
         from sklearn.ensemble import RandomForestRegressor

In [78]: fit_RF = RandomForestRegressor(n_estimators = 200).fit(X_train,y_train)

In [79]: #prediction on train data
         pred_train_RF = fit_RF.predict(X_train)
         #prediction on test data
         pred_test_RF = fit_RF.predict(X_test)

In [80]: ##calculating RMSE for train data
         RMSE_train_RF = np.sqrt(mean_squared_error(y_train, pred_train_RF))
         ##calculating RMSE for test data
         RMSE_test_RF = np.sqrt(mean_squared_error(y_test, pred_test_RF))

In [81]: print("Root Mean Squared Error For Training data = "+str(RMSE_train_RF))
         print("Root Mean Squared Error For Test data = "+str(RMSE_test_RF))

         Root Mean Squared Error For Training data = 0.09594886483675674
         Root Mean Squared Error For Test data = 0.23918653965477282

In [82]: ## calculate R^2 for train data

         r2_score(y_train, pred_train_RF)

Out[82]: 0.9695704079865043

In [83]: #calculate R^2 for test data
         r2_score(y_test, pred_test_RF)

Out[83]: 0.7972255343157659
```

FIGURE 4.4: Random Forest

4.4 Gradient Boosting

Gradient boosting is a machine learning technique for regression and classification problems, which produces a prediction model in the form of an ensemble of weak prediction models, typically decision trees. It builds the model in a stage-wise fashion like other boosting methods do, and it generalizes them by allowing optimization of an arbitrary differentiable loss function. **Below is a screenshot of the model we build and its output:**

3. Random Forest Model

```
In [77]: # Importing libraries for Random Forest
         from sklearn.ensemble import RandomForestRegressor

In [78]: fit_RF = RandomForestRegressor(n_estimators = 200).fit(X_train,y_train)

In [79]: #prediction on train data
         pred_train_RF = fit_RF.predict(X_train)
         #prediction on test data
         pred_test_RF = fit_RF.predict(X_test)

In [80]: ##calculating RMSE for train data
         RMSE_train_RF = np.sqrt(mean_squared_error(y_train, pred_train_RF))
         ##calculating RMSE for test data
         RMSE_test_RF = np.sqrt(mean_squared_error(y_test, pred_test_RF))

In [81]: print("Root Mean Squared Error For Training data = "+str(RMSE_train_RF))
         print("Root Mean Squared Error For Test data = "+str(RMSE_test_RF))

         Root Mean Squared Error For Training data = 0.09594886483675674
         Root Mean Squared Error For Test data = 0.23918653965477282

In [82]: ## calculate R^2 for train data

         r2_score(y_train, pred_train_RF)

Out[82]: 0.9695704079865043

In [83]: #calculate R^2 for test data
         r2_score(y_test, pred_test_RF)

Out[83]: 0.7972255343157659
```

FIGURE 4.5: Gradient Boosting

4.5 Hyper Parameters Tunings for optimizing the results

Model hyperparameters are set by the data scientist ahead of training and control implementation aspects of the model. The weights learned during training of a linear regression model are parameters while the number of trees in a random forest is a model hyperparameter because this is set by the data scientist. Hyperparameters can be thought of as model settings. These settings need to be tuned for each problem because the best model hyperparameters for one particular dataset will not be the best across all datasets. The process of hyperparameter tuning (also called hyperparameter optimization) means finding the combination of hyperparameter values for a machine learning model that performs the best - as measured on a validation dataset - for a problem Here we have used two hyper parameters tuning techniques

- Random Search CV
- Grid Search CV

1. Random Search CV: This algorithm set up a grid of hyperparameter values and select random combinations to train the model and score. The number of search iterations is set based on time/resources.
2. Grid Search CV: This algorithm set up a grid of hyperparameter values and for each combination, train a model and score on the validation data. In this approach, every single combination of hyperparameters values is tried which can be very inefficient.

Check results after using Random Search CV on Random forest and gradient boosting model

4. Gradient Boosting

```
In [86]: # Importing library for GradientBoosting
         from sklearn.ensemble import GradientBoostingRegressor

In [87]: # Building model on top of training dataset
         fit_GB = GradientBoostingRegressor().fit(X_train, y_train)

In [88]: #prediction on train data
         pred_train_GB = fit_GB.predict(X_train)

         #prediction on test data
         pred_test_GB = fit_GB.predict(X_test)

In [89]: ##calculating RMSE for train data
         RMSE_train_GB = np.sqrt(mean_squared_error(y_train, pred_train_GB))
         ##calculating RMSE for test data
         RMSE_test_GB = np.sqrt(mean_squared_error(y_test, pred_test_GB))
```

FIGURE 4.6: Hyper Parameters Tunings for optimizing the results

4.6 Randon Search CV and Grid Search CV

Check results after using Grid Search CV on Random forest and gradient boosting model:

```
In [90]: print("Root Mean Squared Error For Training data = "+str(RMSE_train_GB))
         print("Root Mean Squared Error For Test data = "+str(RMSE_test_GB))

         Root Mean Squared Error For Training data = 0.22921680482502263
         Root Mean Squared Error For Test data = 0.22939164285908767

In [92]: #calculate R^2 for test data
         r2_score(y_test, pred_test_GB)

Out[92]: 0.813493068270751

In [93]: #calculate R^2 for train data
         r2_score(y_train, pred_train_GB)

Out[93]: 0.8263361773771449
```

FIGURE 4.7: Hyper Parameters Tunings for optimizing the results1

```
In [101]: ##Random Search CV on Random Forest Model

          RRF = RandomForestRegressor(random_state = 0)
          n_estimator = list(range(1,20,2))
          depth = list(range(1,100,2))

          # Create the random grid
          rand_grid = {'n_estimators': n_estimator,
                       'max_depth': depth}

          randomcv_rf = RandomizedSearchCV(RRF, param_distributions = rand_grid, n_iter = 5, cv = 5, random_state=0)
          randomcv_rf = randomcv_rf.fit(X_train,y_train)
          predictions_RRF = randomcv_rf.predict(X_test)

          view_best_params_RRF = randomcv_rf.best_params_

          best_model = randomcv_rf.best_estimator_

          predictions_RRF = best_model.predict(X_test)

          #R^2
          RRF_r2 = r2_score(y_test, predictions_RRF)
          #Calculating RMSE
          RRF_rmse = np.sqrt(mean_squared_error(y_test,predictions_RRF))

          print('Random Search CV Random Forest Regressor Model Performance:')
          print('Best Parameters = ',view_best_params_RRF)
          print('R-squared = {:0.2}.'.format(RRF_r2))
          print('RMSE = ',RRF_rmse)

          Random Search CV Random Forest Regressor Model Performance:
          Best Parameters =  {'n_estimators': 15, 'max_depth': 9}
          R-squared = 0.79.
          RMSE =  0.2414849508921194
```

FIGURE 4.8: Check results

```
In [103]: ##Random Search CV on gradient boosting model

          gb = GradientBoostingRegressor(random_state = 0)
          n_estimator = list(range(1,20,2))
          depth = list(range(1,100,2))

          # Create the random grid
          rand_grid = {'n_estimators': n_estimator,
                       'max_depth': depth}

          randomcv_gb = RandomizedSearchCV(gb, param_distributions = rand_grid, n_iter = 5, cv = 5, random_state=0)
          randomcv_gb = randomcv_gb.fit(X_train,y_train)
          predictions_gb = randomcv_gb.predict(X_test)

          view_best_params_gb = randomcv_gb.best_params_

          best_model = randomcv_gb.best_estimator_

          predictions_gb = best_model.predict(X_test)

          #R^2
          gb_r2 = r2_score(y_test, predictions_gb)
          #Calculating RMSE
          gb_rmse = np.sqrt(mean_squared_error(y_test,predictions_gb))

          print('Random Search CV Gradient Boosting Model Performance:')
          print('Best Parameters = ',view_best_params_gb)
          print('R-squared = {:0.2}.'.format(gb_r2))
          print('RMSE = ', gb_rmse)

          Random Search CV Gradient Boosting Model Performance:
          Best Parameters =  {'n_estimators': 15, 'max_depth': 9}
          R-squared = 0.77.
          RMSE =  0.2550691086098142
```

FIGURE 4.9: Check results1

```
In [104]: from sklearn.model_selection import GridSearchCV
          ## Grid Search CV for random Forest model
          regr = RandomForestRegressor(random_state = 0)
          n_estimator = list(range(11,20,1))
          depth = list(range(5,15,2))

          # Create the grid
          grid_search = {'n_estimators': n_estimator,
                         'max_depth': depth}

          ## Grid Search Cross-Validation with 5 fold CV
          gridcv_rf = GridSearchCV(regr, param_grid = grid_search, cv = 5)
          gridcv_rf = gridcv_rf.fit(X_train,y_train)
          view_best_params_GRF = gridcv_rf.best_params_

          #Apply model on test data
          predictions_GRF = gridcv_rf.predict(X_test)

          #R^2
          GRF_r2 = r2_score(y_test, predictions_GRF)
          #Calculating RMSE
          GRF_rmse = np.sqrt(mean_squared_error(y_test,predictions_GRF))

          print('Grid Search CV Random Forest Regressor Model Performance:')
          print('Best Parameters = ',view_best_params_GRF)
          print('R-squared = {:0.2}.'.format(GRF_r2))
          print('RMSE = ',(GRF_rmse))

          Grid Search CV Random Forest Regressor Model Performance:
          Best Parameters =  {'max_depth': 5, 'n_estimators': 12}
          R-squared = 0.8.
          RMSE =  0.2398346306918429
```

FIGURE 4.10: Check results2

```
In [105]: ## Grid Search CV for gradinet boosting
          gb = GradientBoostingRegressor(random_state = 0)
          n_estimator = list(range(11,20,1))
          depth = list(range(5,15,2))

          # Create the grid
          grid_search = {'n_estimators': n_estimator,
                         'max_depth': depth}

          ## Grid Search Cross-Validation with 5 fold CV
          gridcv_gb = GridSearchCV(gb, param_grid = grid_search, cv = 5)
          gridcv_gb = gridcv_gb.fit(X_train,y_train)
          view_best_params_Ggb = gridcv_gb.best_params_

          #Apply model on test data
          predictions_Ggb = gridcv_gb.predict(X_test)

          #R^2
          Ggb_r2 = r2_score(y_test, predictions_Ggb)
          #Calculating RMSE
          Ggb_rmse = np.sqrt(mean_squared_error(y_test,predictions_Ggb))

          print('Grid Search CV Gradient Boosting regression Model Performance:')
          print('Best Parameters = ',view_best_params_Ggb)
          print('R-squared = {:0.2}.'.format(Ggb_r2))
          print('RMSE = ',(Ggb_rmse))

          Grid Search CV Gradient Boosting regression Model Performance:
          Best Parameters =  {'max_depth': 5, 'n_estimators': 19}
          R-squared = 0.79.
          RMSE =  0.2417391489664249
```

FIGURE 4.11: Check resultsand Compare

5 Conclusion

5.1 Model Evaluation

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

In general, most data scientists use two methods to evaluate the performance of the model

1. RMSE Raschka, 2016(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.

2. R Square: is a statistical measure of how close the data are to the fitted regression line. It is also known as the coefficient of determination, or the coefficient of multiple determination for multiple regression. In other words, we can say it explains as to how much of the variance of the target variable is explained.

Below table shows the model results before applying hyper tuning:

Model Evaluation				
ModelName	RMSEtrain	RMSETest	RSquareTrain	RSquareTest
Linear Regression	.27	.25	0.74	0.77
Decision Tree	0.30	0. 28	0.70	0.70
Random Forest Model	0.09	0.23	0.96	0.79
Gradient Boosting	0.22	0.22	0.82	0.81

Below table shows results post using hyper parameter tuning techniques

Model Evaluation			
ModelName	Parameter	RMSE(Test)	RSquare(Test)
Random Search CV	Random Forest	.24	0.79
Random Search CV	Gradient Boosting	0.25	0.77
Grid Search CV	Random Forest	0.23	0.80
Grid Search CV	Gradient Boosting	0.24	0.79

Above table shows the results after tuning the parameters of our two best suited models i.e. Random Forest and Gradient Boosting. For tuning the parameters, we have used Random Search CV and Grid Search CV under which we have given the range of nestimators, depth and CV folds.

5.2 Model Selection

On the basis RMSE and R Squared results a good model should have least RMSE and max R Squared value. So, from above tables we can see: From the observation of all RMSE"Machine Learning in Python" Value and R-Squared Value we have concluded that,

1. Both the models- Gradient Boosting Default and Random Forest perform comparatively well while comparing their RMSE and R-Squared value.
2. After this, I chose Random Forest CV a
3. After applying tunings Random forest model shows best results compared to gradient boosting.
4. So finally, 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, I used this method to predict the target variable for the test data file shared in the problem statement. Results that I found are attached with my submissions.

5.3 More Visualisation

5.3.1 Relation between Number of Passengers and Fare

We can see in below graph that single passengers are the most frequent travelers, and the highest fare also seems to come from cabs which carry just 1 passenger

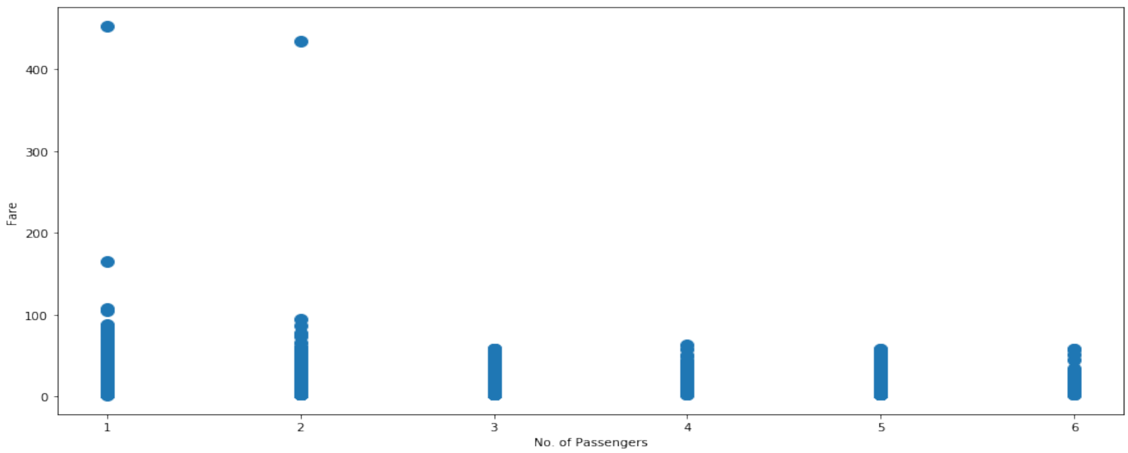

FIGURE 5.1: Number of Passengers and Fare

5.3.2 Relation between Hours and Fare

We can see in below graph that fares

1. During hours 6 PM to 11PM the frequency of cab boarding is very due to peak hours

2. Fare prices during 2PM to 8PM is bit high compared to all other time might be due to high demands.

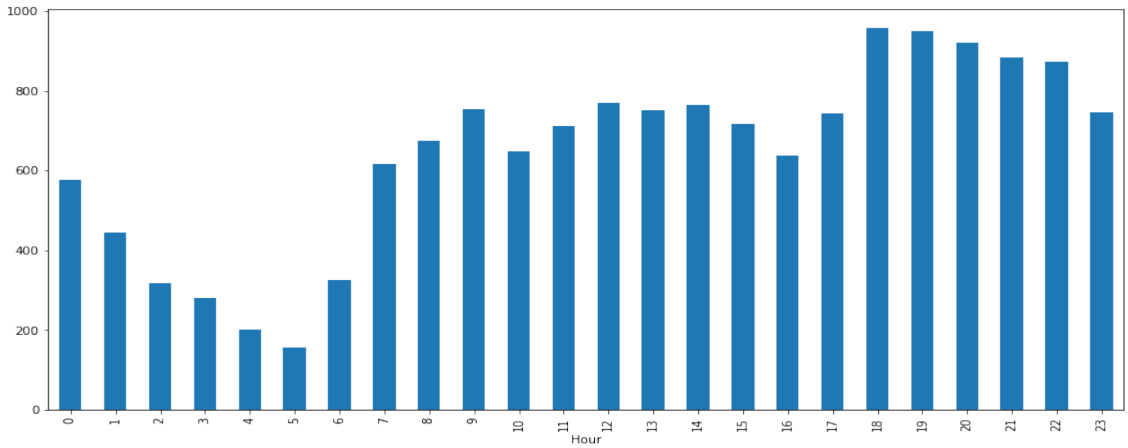

FIGURE 5.2: Relation between Hours and Fare

5.3.3 Relation between Weekday and Fare

We can see in below graph that Cab fare is high on Friday, Saturday and Monday, may be during weekend and first day of the working day they charge high fares because of high demands of cabs.

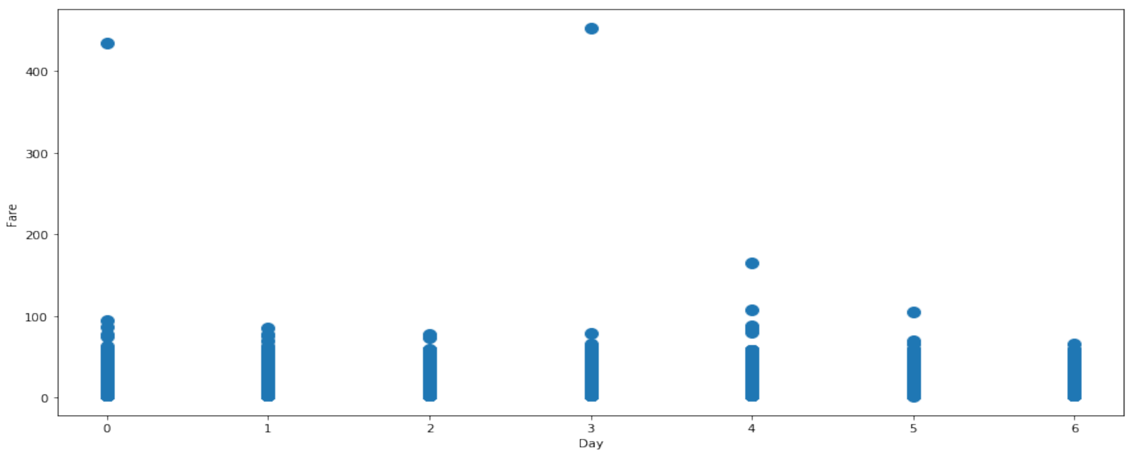

FIGURE 5.3: Relation between Weekday and Fare

5.3.4 Relation between day and Number of rides

We can see in below graph that The day of the week does not seem to have much influence on the number of cabs ride

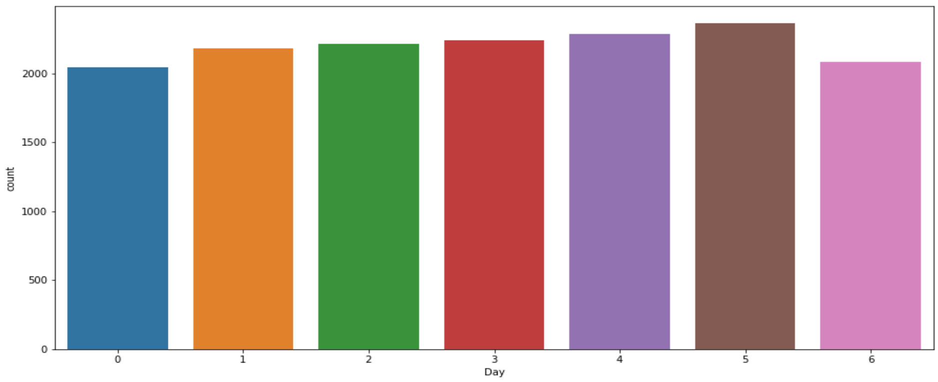

FIGURE 5.4: Relation between day and Number of rides

6 Deployment Conclusion and Future Scope

6.1 Deployment

1. Creating a Simple Web Application using Flask
2. HTML Form
3. Flask script
4. Deploying Flask app using Heroku

Creating a Simple Web Application using Flask There are a number of web dev frameworks like Angular.js, React.js, Node.js written in javascript and others like PHP, ASP.net and many more. But here I have used python to training our machine learning model why not create a web app using the same as well. Flask is a python based microframework used for developing small scale websites.

HTML Form For predicting the fare from various attributes we first need to collect the data(new attribute values) and then use the model we build above to predict cab fare . Therefore, in order to collect the data we create html form which would contain all the different variables from each attribute. Here, I have created a simple form using html only.

Flask script Before starting with the coding part, I need to download flask and some other libraries. Here, we make use of virtual environment, where all the libraries are managed and makes both the development and deployment job easier. Create script.py file in the project folder and copy the following code.

- importing libraries
- import os
- import numpy as np
- import flask
- import pickle
- from flask import Flask, render$_{t}emplate, request$
- creating instance of the class
- app=Flask($_{name)}$
- to tell flask what url shoud trigger the function index()

- @app.route('/')
- @app.route('/index')
- def index():
- return flask.render$_t emplate('index.html')$

T his should run the application and launch a simple server. Open *http://127.0.0.1:5000/ to see the html form.*

Deploying Flask app using Heroku Heroku is a platform as a service (PaaS) that enables developers to build, run, and operate applications entirely in the cloud. In this project we deploy using heroku git.

1. Step 1:At first we need to download gunicorn to our virtual environment venv. We can use pip to download it.
2. Step 2:pip freeze > requirements.txt
3. Step 3:Procfile is a text file in the root directory of your application, to explicitly declare what command should be executed to start your app. This is an essential requirement for heroku.
4. Step 4:we create a .gitignore file.
5. Step 5:heroku login
6. Step 6:push the entire app on heroku and open the url in the browser.

6.2 Conclusion

The quality of a regression model depends on the matchup of predictions against actual values. In regression problems, the dependent variable is continuous. In classification problems, the dependent variable is categorical. Random Forest can be used to solve both regression and classification problems. The K-NN algorithm is a simple, easy-to-implement supervised machine learning algorithm that can be used to solve both classification and regression problems. Decision trees are nonlinear; unlike linear regression, there is no equation to express the relationship between independent and dependent variables. Out of the three models left, Random Forest is the best model as it has the lowest RMSE score and highest R-Squared score, which explains the highest variability and tells us how well the model fits in this data.

6.3 Future Scope

As is known, with an increase in the number of features; underlying equations become a higher-order polynomial equation, and it leads to overfitting of the data. Generally, it is seen that an overfitted model performs worse on the testing data set, and it is also observed that the overfitted model performs worse on additional new test data set as well. A kind of normalized regression type - Ridge Regression may be further considered.

A Appendix

A.1 R code

Cab Fare Prediction

rm(list = ls()) setwd("D:/edwisor/edwisorproject/rproject/edwisorproject/cabfareprediction")

getwd() 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) install.packages(x[4],x[11]) rm(x)

The details of data attributes in the dataset are as follows: pickup$_d$$atetime -$ $timestampvalueindicatingwhenthecabridestarted.pickup_longitude - floatforlongitudecoordinateofwherethec$ $floatforlatitudecoordinateofwherethecabridestarted.dropoff_longitude - floatforlongitudecoordinateofwhere$ $floatforlatitudecoordinateofwherethecabrideended.passenger_count - anintegerindicatingthenumberofpassen$

loading datasets train = read.csv("train$_c$$ab.csv", header = T, na.strings = c("", "", "NA"))test =$ $read.csv("test.csv")test_pickup_datetime = test["pickup_datetime"]Structureofdatastr(train)str(test)summary($ $as.numeric(as.character(train$fare$_a$$mount))train$passenger$_c$$ount = round(train$passenger$_c$$ount)$

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$_a$$mount < 1),])train = train[-which(train$fare$_a$$mount <$ $1),]$

2.Passenger$_c$$ountvariablefor(iinseq(4, 11, by = 1))print(paste('passenger_countabove', i, nrow(train[which$ $i),])))so20observationsofpassenger_countisconsistenlyabovefrom6, 7, 8, 9, 10passenger_counts, let'scheckthem.t$ $6),]$

Also we need to see if there are any passenger$_c$$ount == 0train[which(train$passenger$_c$$ount <$ $1),]nrow(train[which(train$passenger$_c$$ount < 1),])Wewillremovethese58observationsand20observationwhich$ $train[-which(train$passenger$_c$$ount < 1),]train = train[-which(train$passenger$_c$$ount >$ $6),]$

nrow(train[which(train$passenger_count > 6),])nrow(train[which(train$passenger$_c$$ount <$ $1),])3.Latitudesrangefrom - 90to90.Longitudesrangefrom - 180to180.Removingwhichdoesnotsatisfythesera$ $, nrow(train[which(train$pickup$_l$$ongitude > 180),])))print(paste('pickup_longitudeabove -$ $180 =', nrow(train[which(train$pickup$_l$$ongitude < -180),])))print(paste('pickup_latitudeabove90 ='$ $, nrow(train[which(train$pickup$_l$$atitude > 90),])))print(paste('pickup_latitudeabove -$ $90 =', nrow(train[which(train$pickup$_l$$atitude < -90),])))print(paste('dropoff_longitudeabove180 ='$ $, nrow(train[which(train$dropoff$_l$$ongitude > 180),])))print(paste('dropoff_longitudeabove -$ $180 =', nrow(train[which(train$dropoff$_l$$ongitude < -180),])))print(paste('dropoff_latitudeabove -$ $90 =', nrow(train[which(train$dropoff$_l$$atitude < -90),])))print(paste('dropoff_latitudeabove90 ='$ $, nrow(train[which(train$dropoff$_l$$atitude > 90),])))There'sonlyoneoutlierwhichisinvariablepickup_latitude.So$ $0),])nrow(train[which(train$pickup$_l$$atitude == 0),])nrow(train[which(train$dropoff$_l$$ongitude ==$ $0),])nrow(train[which(train$pickup$_l$$atitude == 0),])therearevalueswhichareequalto0.wewillremovethem.trai$ $train[-which(train$pickup$_l$$atitude > 90),]train = train[-which(train$pickup$_l$$ongitude ==$ $0),]train = train[-which(train$dropoff$_l$$ongitude == 0),]$

Make a copy df=train train=df

Missing Value Analysis missing$_{v}al = data.frame(apply(train, 2, function(x)sum(is.na(x))))missing_{v}al$
= row.names(missing$_{v}al)names(missing_{v}al)[1] = "Missing_{p}ercentage"missing_{v}al$Missing$_{p}ercentage =$
$(missing_{v}al$Missing$_{p}ercentage/nrow(train)) * 100 missing_{v}al = missing_{v}al[order(-missing_{v}al$Missing$_{p}ercen$
$NULL missing_{v}al = missing_{v}al[, c(2, 1)]missing_{v}al$

unique(train$passenger_{c}ount)unique(test$passenger$_{c}ount)train[,'passenger_{c}ount'] =$
$factor(train[,'passenger_{c}ount'], labels = (1:6))test[,'passenger_{c}ount'] = factor(test[,'passenger_{c}ount'], lab$
$(1:6))1.ForPassenger_{c}ount : Actualvalue = 1 Mode = 1 KNN = 1 train$passenger$_{c}ount[1000]train$passenger
$NA getmode < -function(v)uniqv < -unique(v)uniqv[which.max(tabulate(match(v, uniqv)))]$

Mode Method getmode(train$passenger_{c}ount)Wecan'tusemodemethodbecausedatawillbemorebiasedtoward$
1

2.For fare$_{a}mount : Actualvalue = 18.1, Mean = 15.117, Median = 8.5, KNN =$
$18.28 sapply(train, sd, na.rm = TRUE)fare_{a}mountpickup_{d}atetimepickup_{l}ongitude 435.9682364635.7005312.6$
NA

Mean Method mean(train$fare_{a}mount, na.rm = T)$

Median Method median(train$fare_{a}mount, na.rm = T)$

kNN Imputation train = knnImputation(train, k = 181) train$fare_{a}mount[1000]train$passenger$_{c}ount[1000]$
$TRUE)fare_{a}mountpickup_{d}atetimepickup_{l}ongitude 435.6619524635.7005312.659050 pickup_{l}atitudedropoff_{l}on$

df1=train train=df1 Outlier Analysis

We Will do Outlier Analysis only on Fare$_{a}mountjustfornowandwewilldooutlieranalysisafterfeatureengi$
$ggplot(train, aes(x = factor(passenger_{c}ount), y = fare_{a}mount))pl1 + geom_{b}oxplot(outlier.colour =$
$"red", fill = "grey", outlier.shape = 18, outlier.size = 1, notch = FALSE) + ylim(0, 100)$

Replace all outliers with NA and impute vals = train[,"fare$_{a}mount"]train[which(vals), "fare_{a}mount"] =$
NA

lets check the NA's sum(is.na(train$fare_{a}mount))$

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

lets check the missing values sum(is.na(train$fare_{a}mount))str(train)$

df2=train train=df2 Feature Engineering 1.Feature Engineering for timestamp variable we will derive new features from pickup$_{d}atetimevariablenewfeatureswillbeyear, month, day_{o}f_{w}eek,$
$as.Date(as.character(train$pickup$_{d}atetime))train$pickup$_{w}eekday = as.factor(format(train$pickup$_{d}ate, "train$
$as.factor(format(train$pickup$_{d}ate, "train$pickup$_{y}r = as.factor(format(train$pickup$_{d}ate, "pickup_{t}ime =$
$strptime(train$pickup$_{d}atetime, "train$pickup$_{h}our = as.factor(format(pickup_{t}ime, "$

Add same features to test set test$pickup_{d}ate = as.Date(as.character(test$pickup$_{d}atetime))test$pickup$_{w}eek$
$as.factor(format(test$pickup$_{d}ate, "test$pickup$_{m}nth = as.factor(format(test$pickup$_{d}ate, "test$pickup$_{y}r =$
$as.factor(format(test$pickup$_{d}ate, "pickup_{t}ime = strptime(test$pickup$_{d}atetime, "test$pickup$_{h}our =$
$as.factor(format(pickup_{t}ime, "$

sum(is.na(train)) there was 1 'na' in pickup$_{d}atetimewhichcreatedna'sinabovefeatureengineeredvariables.t$
$na.omit(train)wewillremovethat1rowofna's$

train = subset(train,select = -c(pickup$_{d}atetime, pickup_{d}ate))test = subset(test, select =$
$-c(pickup_{d}atetime, pickup_{d}ate))Nowwewillusemonth, weekday, hourtoderivenewfeatureslikesessionsinaday,$
$weekend/weekdayf = function(x)if((x >= 5)(x <= 11))return('morning')if((x >= 12)(x <= 16))retur$
$function(deg)(deg * pi)/180 haversine = function(long1, lat1, long2, lat2)long1rad = deg_{t}o_{r}ad(long1)phi1$

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$_{l}ongitude, train$pickup$_{l}atitude, train$dropoff$_{l}ongitude, train$dropoff$_{l}atitud$
= haversine(test$pickup_{l}ongitude, test$pickup$_{l}atitude, test$dropoff$_{l}ongitude, test$dropoff$_{l}atitude)$

We will remove the variables which were used to feature engineer new variables train = subset(train,select = -c(pickup$_{l}ongitude, pickup_{l}atitude, dropoff_{l}ongitude, dropoff_{l}atitude))test =$
$subset(test, select = -c(pickup_{l}ongitude, pickup_{l}atitude, dropoff_{l}ongitude, dropoff_{l}atitude))$

str(train) summary(train)

Feature selection numeric$_{i}ndex = sapply(train, is.numeric)selectingonlynumeric$

numeric$_{d}ata = train[, numeric_{i}ndex]$

cnames = colnames(numeric$_{d}ata) Correlation analysis for numeric variables corrgram(train[, numeric_{i}ndex]$
$panel.pie, main = "CorrelationPlot")$

ANOVA for categorical variables with target numeric variable

aov$_{r}esults = aov(fare_{a}mount\ passenger_{c}ount * pickup_{h}our * pickup_{w}eekday, data = train) aov_{r}esults = aov(fare_{a}mount\ passenger_{c}ount + pickup_{h}our + pickup_{w}eekday + pickup_{m}nth + pickup_{y}r, data = train)$

summary(aov$_{r}esults)$

pickup$_{w}eekdathaspvaluegreaterthan 0.05 train = subset(train, select = -pickup_{w}eekday)$

remove from test set test = subset(test,select=-pickup$_{w}eekday)$

Feature Scaling Normality check qqnorm(train$fare_{a}mount) histogram(train$fare$_{a}mount) library(car) dev.$
$c(1, 2)) qqPlot(train$fare$_{a}mount) qqPlot, ithasaxvaluesderivedfromgaussiandistribution, ifdataisdistributedno$

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$fare_{a}mou$
$0.75, list = FALSE) 75 train_{d}ata = train[tr.idx,] test_{d}ata = train[-tr.idx,]$

rmExcept(c("test","train","df",'df1','df2','df3','test$_{d}ata',' train_{d}ata',' test_{p}ickup_{d}atetime')) ModelSelectionE$

Linear regression lm$_{m}odel = lm(fare_{a}mount\ ., data = train_{d}ata)$

summary(lm$_{m}odel) str(train_{d}ata) plot(lm_{m}odel$fitted.values,rstandard(lm$_{m}odel), main = "Residualplot", xlab = "Predictedvaluesoffare_{a}mount", ylab = "standardizedresiduals")$

lm$_{p}redictions = predict(lm_{m}odel, test_{d}ata[, 2 : 6])$

qplot(x = test$_{d}ata[, 1], y = lm_{p}redictions, data = test_{d}ata, color = I("blue"), geom = "point")$

regr.eval(test$_{d}ata[, 1], lm_{p}redictions) maemsermsemape 3.530311419.30797264.39408380.4510407$

Decision Tree

Dt$_{m}odel = rpart(fare_{a}mount\ ., data = train_{d}ata, method = "anova")$

summary(Dt$_{m}odel) Predictfornewtestcasespredictions_{D}T = predict(Dt_{m}odel, test_{d}ata[, 2 : 6])$

qplot(x = test$_{d}ata[, 1], y = predictions_{D}T, data = test_{d}ata, color = I("blue"), geom = "point")$

regr.eval(test$_{d}ata[, 1], predictions_{D}T) maemsermsemape 1.89815926.70347132.58910630.2241461$

Random forest rf$_{m}odel = randomForest(fare_{a}mount\ ., data = train_{d}ata)$

summary(rf$_{m}odel)$

rf$_{p}redictions = predict(rf_{m}odel, test_{d}ata[, 2 : 6])$

qplot(x = test$_{d}ata[, 1], y = rf_{p}redictions, data = test_{d}ata, color = I("blue"), geom = "point")$

regr.eval(test$_{d}ata[, 1], rf_{p}redictions) maemsermsemape 1.90538506.36822832.52353490.2335395$

Improving Accuracy by using Ensemble technique —- XGBOOST train$_{d}ata_{m}atrix = as.matrix(sapply(train_{d}ata[-1], as.numeric)) test_{d}ata_{d}ata_{m}atrix = as.matrix(sapply(test_{d}ata[-1], as.numeric$

xgboost$_{m}odel = xgboost(data = train_{d}ata_{m}atrix, label = train_{d}ata$fare$_{a}mount, nrounds = 15, verbose = FALSE)$

summary(xgboost$_{m}odel) xgb_{p}redictions = predict(xgboost_{m}odel, test_{d}ata_{d}ata_{m}atrix)$

qplot(x = test$_{d}ata[, 1], y = xgb_{p}redictions, data = test_{d}ata, color = I("blue"), geom = "point")$

regr.eval(test$_{d}ata[, 1], xgb_{p}redictions) maemsermsemape 1.61834155.10964652.26045270.1861947$

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$_{d}ata_{m}atrix 2 = as.matrix(sapply(train[-1], as.numeric)) test_{d}ata_{m}atrix 2 = as.matrix(sapply(test, as.numeric))$

xgboost$_m odel2 = xgboost(data = train_d ata_m atrix2, label = train$fare$_a mount, nrounds =$ 15, $verbose = FALSE)$

Saving the trained model saveRDS(xgboost$_m odel2, "./final_X gboost_m odel_u sing_R.rds")$

loading the saved model super$_m odel < -readRDS("./final_X gboost_m odel_u sing_R.rds")print(super_m odel)$

Lets now predict on test dataset xgb = predict(super$_m odel, test_d ata_m atrix2)$

xgb$_p red = data.frame(test_p ickup_d atetime, "predictions" = xgb)$

Now lets write(save) the predicted fare$_a mountindiskas.csvformatwrite.csv(xgb_p red, "xgb_p redictions_R.csv$ $FALSE)$

B Appendix

B.1 Python code

!/usr/bin/env python coding: utf-8

Jupyter Notebook for Cab fare Prediction

In[40]:

Importing required libraries import os getting access to input files import pandas as pd Importing pandas for performing EDA import numpy as np Importing numpy for Linear Algebric operations import matplotlib.pyplot as plt Importing for Data Visualization import seaborn as sns Importing for Data Visualization from collections import Counter from sklearn.tree import DecisionTreeRegressor from sklearn.ensemble import RandomForestRegressor from sklearn.ensemble import GradientBoostingRegressor from sklearn.linear$_m odelimportLinearRegressionMLalgorithmfromsklearn$

get$_i python().run_l ine_m agic('matplotlib', 'inline')$

In[41]:

Setting the working directory

os.chdir("D:/edWisor/edwisorproject/pythonproject") print(os.getcwd())

The details of data attributes in the dataset are as follows: - pickup$_d atetime -$
$timestampvalueindicatingwhenthecabridestarted. - pickup_l ongitude - floatforlongitudecoordinateofwhereth$
$pickup_l atitude - floatforlatitudecoordinateofwherethecabridestarted. - dropoff_l ongitude -$
$floatforlongitudecoordinateofwherethecabrideended. - dropoff_l atitude - floatforlatitudecoordinateofwhere$
$passenger_c ount - anintegerindicatingthenumberofpassengersinthecabride.$

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[42]:

Since one of values in pickup$_d atetimecolumnis43soreplacingitbyNANLoadingthedata:$
$train = pd.read_c sv("train_c ab.csv")makingdataframefromcsvfiletrain = pd.read_c sv("train_c ab.csv", na_v alues$
$"pickup_d atetime" : "43")$

test = pd.read$_c sv("test.csv")$

In[43]:

understanding data train.head() checking first five rows of the training dataset print(train)

In[44]:

print("shape of training data is: ",train.shape) checking the number of rows and columns in training data print("shape of test data is: ",test.shape) checking the number of rows and columns in test data

In[45]:

checking the data-types in training dataset train.dtypes

In[46]:
checking the data-types in test dataset test.dtypes
In[47]:
test.head(5)
In[48]:
train.describe()
In[49]:
test.describe()
Data cleaning and missing value analysis
In[50]:
Convert fare$_amount fromobjecttonumericUsingerrors =' coerce'.Itwillreplaceallnon - numericvalueswithNaN.train["fare_amount"] = pd.to_numeric(train["fare_amount"], errors = "coerce")$
In[51]:
Checking the data-types in training dataset train.dtypes
In[52]:
train.shape
In[53]:
train['pickup$_datetime'].isnull().sum()$
In[54]:
There is only one row which is having NAN in pickup$_datetimecolumnsodeleteit.Removingpickup_datetime$
Dropping NA values in datetime column train=train.dropna(subset= ["pickup$_datetime"])train["pickup_d$ $0, how =' any')$
In[55]:
train['pickup$_datetime'].isnull().sum()$
In[56]:
train.shape
In[57]:
Here pickup$_datetimevariableisinobjectsoweneedtochangeitsdatatypetodatetimetrain['pickup_datetime'] = pd.to_datetime(train['pickup_datetime'], format ='$
In[58]:
Here pickup$_datetimevariableisinobjectsoweneedtochangeitsdatatypetodatetimetest['pickup_datetime'] = pd.to_datetime(test['pickup_datetime'], format ='$
In[59]:
train.dtypes
In[60]:
test.dtypes
In[61]:
test.head()
In[62]:
we will saperate the Pickup$_datetimecolumnintoseparatefieldlikeyear, month, dayoftheweek, etcSeries.dtqu$
train['year'] = train['pickup$_datetime'].dt.yeartrain['Month'] = train['pickup_datetime'].dt.monthtrain['Da$ $train['pickup_datetime'].dt.daytrain['Day'] = train['pickup_datetime'].dt.dayofweektrain['Hour'] = train['pickup_datetime'].dt.hourtrain['Minute'] = train['pickup_datetime'].dt.minute$
In[63]:
we will saperate the Pickup$_datetimecolumnintoseparatefieldlikeyear, month, dayoftheweek, etcSeries.dtqu$
test['year'] = test['pickup$_datetime'].dt.yeartest['Month'] = test['pickup_datetime'].dt.monthtest['Date'] = test['pickup_datetime'].dt.daytest['Day'] = test['pickup_datetime'].dt.dayofweektest['Hour'] = test['pickup_datetime'].dt.hourtest['Minute'] = test['pickup_datetime'].dt.minute$
In[64]:
test.head(5)

In[65]:

train.dtypes Re-checking datatypes after conversion

Observations: - point1: An outlier value of 43 in pickup$_{d}atetime. - point2 : passengercountshouldnotexceedthan6(evenforSUV). - point3 : Latitudesrangefrom - 90to + 90andLongitudesrangefrom - 180to180. - point4 : Veryfewmissingvaluesandhighvaluesoffareandpa$

In[66]:

print(train.shape) print(train['pickup$_{d}atetime'].isnull().sum())$

In[68]:

print(test.shape) print(test['pickup$_{d}atetime'].isnull().sum())$

Now there is no NAN values with respect to pickup$_{d}atetimecolumn$

In[69]:

checking the passenger count value train["passenger$_{c}ount"].describe()$

In[70]:

we see max value of passenger$_{c}ountis5345whichisactuallynotfeasiblesoreducingitto6train = train.drop(train[train["passenger_{c}ount"] > 6].index, axis = 0)Alsoremovingthevalueswithpassengercountof$ $train.drop(train[train["passenger_{c}ount"] == 0].index, axis = 0)$

In[71]:

train["passenger$_{c}ount"].describe()$

In[72]:

train["passenger$_{c}ount"].sort_{v}alues(ascending = True)$

In[73]:

removing passanger$_{c}ountmissingvaluesrowstrain = train.drop(train[train['passenger_{c}ount'].isnull()].in$ $0)print(train.shape)print(train['passenger_{c}ount'].isnull().sum())$

In[75]:

There is one passenger count value of 0.12 which is not possible. Hence we will remove fractional passenger value train = train.drop(train[train["passenger$_{c}ount"] == 0.12].index, axis = 0)train.shape$

In[76]:

Now analyzing fare amount variable finding decending order of fare to get to know whether the outliers are present or not train["fare$_{a}mount"].sort_{v}alues(ascending = False)$

In[77]:

Counter(train["fare$_{a}mount"] < 0)$

In[78]:

train = train.drop(train[train["fare$_{a}mount"] < 0].index, axis = 0)train.shape$

In[79]:

make sure there is no negative values in the fare$_{a}mountvariablecolumntrain["fare_{a}mount"].min()$

In[80]:

Also remove the row where fare amount is zero train = train.drop(train[train["fare$_{a}mount"] < 1].index, axis = 0)train.shape$

In[81]:

Now we can see that there is a huge difference in 1st 2nd and 3rd position in decending order of fare amount so we will remove the rows having fare amounting more that 454 as considering them as outliers

train = train.drop(train[train["fare$_{a}mount"] > 454].index, axis = 0)train.shape$

In[82]:

eliminating rows for which value of "fare$_{a}mount"ismissingtrain = train.drop(train[train['fare_{a}mount'].i$ $0)print(train.shape)print(train['fare_{a}mount'].isnull().sum())$

In[83]:

train["fare$_{a}mount"].describe()$

In[ ]:

now checking the latitude andlongitude Lattitude—-(-90 to 90) Longitude—-(-180 to 180)

we need to drop the rows having pickup lattitute and longitute out the range mentioned above

In[84]:

Hence dropping the values train = train.drop((train[train['pickup$_l atitude'] < -90]).index, axis = 0) train = train.drop((train[train['pickup_l atitude'] > 90]).index, axis = 0)$

In[86]:

Hence dropping the values train = train.drop((train[train['pickup$_l ongitude'] < -180]).index, axis = 0) train = train.drop((train[train['pickup_l ongitude'] > 180]).index, axis = 0)$

In[87]:

Hence dropping the values train = train.drop((train[train['dropoff$_l atitude'] < -90]).index, axis = 0) train = train.drop((train[train['dropoff_l atitude'] > 90]).index, axis = 0) Hencedroppingthevaluestrain = train.drop((train[train['dropoff_l ongitude'] < -180]).index, axis = 0) train = train.drop((train[train['dropoff_l ongitude'] > 180]).index, axis = 0)$

In[88]:

train.shape

In[89]:

train.isnull().sum()

In[90]:

test.isnull().sum()

In[91]:

now we cleaned our both datasets and do further operations

In[92]:

calculating distance between coordinates As we know that we have given pickup longitute and latitude values and same for drop. So we need to calculate the distance Using the haversine formula and we will create a new variable called distance from math import radians, cos, sin, asin, sqrt

def haversine(a): lon1=a[0] lat1=a[1] lon2=a[2] lat2=a[3] """ Calculate the great circle distance between two points on the earth (specified in decimal degrees) """ convert decimal degrees to radians lon1, lat1, lon2, lat2 = map(radians, [lon1, lat1, lon2, lat2])

haversine formula dlon = lon2 - lon1 dlat = lat2 - lat1 a = sin(dlat/2)**2 + cos(lat1) * cos(lat2) * sin(dlon/2)**2 c = 2 * asin(sqrt(a)) Radius of earth in kilometers is 6371 km = 6371* c return km

In[93]:

train['distance'] = train[['pickup$_l ongitude',' pickup_l atitude',' dropoff_l ongitude',' dropoff_l atitude']].apply$ 1)

In[94]:

test['distance'] = test[['pickup$_l ongitude',' pickup_l atitude',' dropoff_l ongitude',' dropoff_l atitude']].apply(h$ 1)

In[96]:

train.head(5)

In[97]:

test.head(5)

In[98]:

train.nunique()

In[211]:

test.nunique()

In[99]:

finding decending order of fare to get to know whether the outliers are presented or not train['distance'].sort$_{v}alues(ascending = False)$

In[100]:

train["distance"].describe()

In[101]:

train["distance"].sort$_{v}alues(ascending = False).head(25)$

In[ ]:

As we can see that top 23 values in the distance variables are very high It means more than 1000 Kms distance they have travelled Also just after 23rd value from the top, the distance goes down to 129, which means these values are showing some outliers We need to remove these outliers.

In[102]:

Counter(train['distance'] == 0)

In[103]:

Counter(test['distance'] == 0)

In[104]:

Counter(train['fare$_{a}mount'] == 0)$

In[105]:

we will remove the rows whose distance value is zero

train = train.drop(train[train['distance']== 0].index, axis=0) train.shape

In[106]:

we will remove the rows whose distance values is very high which is more than 129kms train = train.drop(train[train['distance'] > 130].index, axis=0) train.shape

In[107]:

train.head()

In[ ]:

Now we have splitted the pickup$_{d}atetimevariableintodifferentvaraibleslikemonth, year, dayetcsonowwed$

In[108]:

drop = ['pickup$_{d}atetime',' pickup_{l}ongitude',' pickup_{l}atitude',' dropoff_{l}ongitude',' dropoff_{l}atitude',' Minu$
$train.drop(drop, axis = 1)test = test.drop(drop, axis = 1)$

In[109]:

train.head()

In[110]:

test.head()

In[111]:

train['passenger$_{c}ount'] = train['passenger_{c}ount'].astype('int64')train['year'] = train['year'].astype('int64')train['Month'] = train['Month'].astype('int64')train['Date'] = train['Date'].astype('int64')train['Day'] = train['Day'].astype('int64')train['Hour'] = train['Hour'].astype('int64')$

In[112]:

test['passenger$_{c}ount'] = test['passenger_{c}ount'].astype('int64')test['year'] = test['year'].astype('int64')tes$
$test['Month'].astype('int64')test['Date'] = test['Date'].astype('int64')test['Day'] = test['Day'].astype('int64'$
$test['Hour'].astype('int64')$

In[113]:

train.dtypes

In[114]:

test.dtypes

In[115]:

test.describe()

In[116]:

test.head(5)

In[306]:

test.head()

In[307]:

test.dtypes

DATA VISUALIZATION VisuaLISE of following: - 1. Number of Passengers effects the the fare - 2. Pickup date and time effects the fare - 3. Day of the week does effects the fare - 4. Distance effects the fare

In[123]:

Count plot on passenger count plt.figure(figsize=(15,7)) sns.countplot(x="passenger$_count", data = train) plt.savefig('Figures/passengercount.png') plt.savefig('Figures/passengercount.pdf') plt.savefig('Fig$

In[124]:

train['passenger$_count'].describe()$

In[125]:

Counter(train["passenger$_count"])$

In[126]:

Relationship beetween number of passengers and Fare

plt.figure(figsize=(15,7)) plt.scatter(x=train['passenger$_count'], y = train['fare_amount'], s = 100) plt.xlabel('No.of Passengers') plt.ylabel('Fare') plt.show() plt.savefig('Figures/Numberofpassengerand$

Observations : By seeing the above plots we can easily conclude that: - 1. single travelling passengers are most frequent travellers. - 2. At the sametime we can also conclude that highest Fare are coming from single double travelling passengers.

In[127]:

Relationship between date and Fare plt.figure(figsize=(15,7)) plt.scatter(x=train['Date'], y=train['fare$_amount'], s = 100) plt.xlabel('Date') plt.ylabel('Fare') plt.show() plt.savefig('Figures/dateandfa$

In[128]:

plt.figure(figsize=(15,7)) train.groupby(train["Hour"])['Hour'].count().plot(kind="bar")

plt.show() plt.savefig('Figures/hourcount.eps') plt.savefig('Figures/hourcount.jpg') plt.savefig('Figures/hourcount.pdf') plt.savefig('Figures/hourcount.png')

Analyzing above graph we can predict that Lowest cabs at 5 AM and highest at and around 7 PM to8 PM i.e the office rush hours

In[129]:

Relationship between Time and Fare plt.figure(figsize=(15,7)) plt.scatter(x=train['Hour'], y=train['fare$_amount'], s = 100) plt.xlabel('Hour') plt.ylabel('Fare') plt.show() plt.savefig('Figures/timeandf$

From the above plot We can observe that the cabs taken at 7 am and 23 Pm are the costliest. Hence we can assume that cabs taken early in morning and late at night are costliest

In[130]:

impact of Day on the number of cab rides plt.figure(figsize=(15,7)) sns.countplot(x="Day", data=train) plt.savefig('Figures/impactofdayonnumbercabride.eps') plt.savefig('Figures/impactofdayonn plt.savefig('Figures/impactofdayonnumbercabride.pdf') plt.savefig('Figures/impactofdayonnumbercabri

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

In[131]:

Relationships between day and Fare plt.figure(figsize=(15,7)) plt.scatter(x=train['Day'], y=train['fare$_amount'], s = 100) plt.xlabel('Day') plt.ylabel('Fare') plt.show() plt.savefig('Figures/impactofd$

The highest fares seem to be on a Sunday, Monday and Thursday, and the low on Wednesday and Saturday. May be due to low demand of the cabs on saturdays the cab fare is low and high demand of cabs on sunday and monday shows the high fare prices

In[132]:

Relationship between distance and fare plt.figure(figsize=(15,10)) plt.scatter(x = train['distance'],y = train['fare$_a mount'], c = "r")plt.xlabel('Distance')plt.ylabel('Fare')plt.show()plt.savefig$

observation It is quite obvious that distance will effect the amount of fare.

Feature Scaling :

In[133]:

Normality check of training data is uniformly distributed or not-

for i in ['fare$_a mount','distance'] : print(i)sns.distplot(train[i], bins =' auto', color =' red')plt.title("Distribution for Variable" + i)plt.ylabel("Density")plt.show()plt.savefig('Figures/featuresca$

In[109]:

since skewness of target variable is high, apply log transform to reduce the skewness- train['fare$_a mount'] = np.log1p(train['fare_a mount'])$

In[134]:

Normality Re-check to check data is uniformly distributed or not after log transformartion

for i in ['fare$_a mount','distance'] : print(i)sns.distplot(train[i], bins =' auto', color =' red')plt.title("Distribution for Variable" + i)plt.ylabel("Density")plt.show()plt.savefig('Figures/correctedf$

In[318]:

since skewness of distance variable is high, apply log transform to reduce the skewness- train['distance'] = np.log1p(train['distance'])

In[135]:

Normality Re-check to check data is uniformly distributed or not after log transformartion

for i in ['fare$_a mount','distance'] : print(i)sns.distplot(train[i], bins =' auto', color =' green')plt.title("Distribution for Variable" + i)plt.ylabel("Density")plt.show()plt.savefig('Figures/finalfe$

In[ ]:

Here we can see bell shaped distribution. Hence our continous variables are now normally distributed, we will use not use any Feature Scalling technique. i.e, Normalization or Standarization for our training data

In[136]:

Normality check for test data is uniformly distributed or not-

sns.distplot(test['distance'],bins='auto',color='yellow') plt.title("Distribution for Variable "+i) plt.ylabel("Density") plt.show() plt.savefig('Figures/testdistribution.eps') plt.savefig('Figures/testdistribution.jpg') plt.savefig('Figures/testdistribution.pdf') plt.savefig('Figures/testdistribution.png')

In[137]:

since skewness of distance variable is high, apply log transform to reduce the skewness- test['distance'] = np.log1p(test['distance'])

In[138]:

rechecking the distribution for distance sns.distplot(test['distance'],bins='auto',color='violet')

plt.title("Distribution for Variable "+i) plt.ylabel("Density") plt.show() plt.savefig('Figures/finaltestdistri plt.savefig('Figures/finaltestdistribution.jpg') plt.savefig('Figures/finaltestdistribution.pdf') plt.savefig('Figures/finaltestdistribution.png')

In[ ]:

As we can see a bell shaped distribution. Hence our continous variables are now normally distributed, we will use not use any Feature Scalling technique. i.e, Normalization or Standarization for our test data

In[ ]:

Applying ML ALgorithms:

In[143]:

train test split for further modelling x$_t rain, x_t est, y_t rain, y_t est = train_t est_s plit(train.iloc[:, train.columns! =' fare_a mount'], train.iloc[:, 0], test_s ize = 0.20, random_s tate = 1)$

In[144]:
x$_t rain.head(5)$
In[325]:
y$_t rain.head(5)$
In[145]:
print(x$_t rain.shape) print(x_t est.shape)$
In[146]:
print(y$_t rain.shape) print(y_t est.shape)$
In[332]:
y$_t est.head()$
In[ ]:
In[333]:
print(y$_t rain.shape) print(y_t est.shape)$
In[ ]:
Linear Regression Model :
In[147]:
Building model on top of training dataset fit$_L R = LinearRegression().fit(x_t rain, y_t rain)$
In[148]:
prediction on train data pred$_t rain_L R = fit_L R.predict(x_t rain)$
In[149]:
prediction on train data pred$_t rain_L R = fit_L R.predict(x_t rain)$
In[151]:
prediction on test data pred$_t est_L R = fit_L R.predict(x_t est)$
In[152]:
calculating RMSE for test data RMSE$_t est_L R = np.sqrt(mean_s quared_e rror(y_t est, pred_t est_L R))$
calculating RMSE for train data RMSE$_t rain_L R = np.sqrt(mean_s quared_e rror(y_t rain, pred_t rain_L R))$
In[153]:
print("Root Mean Squared Error For Training data = "+str(RMSE$_t rain_L R)) print(''RootMeanSquaredErro$
$'' + str(RMSE_t est_L R))$
In[154]:
calculate R$^2 fortraindatafromsklearn.metricsimportr2_s corer2_s core(y_t rain, pred_t rain_L R)$
In[155]:
r2$_s core(y_t est, pred_t est_L R)$
Decision Tree Model
In[156]:
fit$_D T = DecisionTreeRegressor(max_d epth = 2).fit(x_t rain, y_t rain)$
In[157]:
prediction on train data pred$_t rain_D T = fit_D T.predict(x_t rain)$
prediction on test data pred$_t est_D T = fit_D T.predict(x_t est)$
In[158]:
calculating RMSE for train data RMSE$_t rain_D T = np.sqrt(mean_s quared_e rror(y_t rain, pred_t rain_D T))$
calculating RMSE for test data RMSE$_t est_D T = np.sqrt(mean_s quared_e rror(y_t est, pred_t est_D T))$
In[159]:
print("Root Mean Squared Error For Training data = "+str(RMSE$_t rain_D T)) print(''RootMeanSquaredErro$
$'' + str(RMSE_t est_D T))$
In[160]:
R$^2 calculationfortraindatar2_s core(y_t rain, pred_t rain_D T)$
In[145]:
RANDOM FOREST MODEL
In[161]:
fit$_R F = RandomForestRegressor(n_e stimators = 200).fit(x_t rain, y_t rain)$

In[162]:

prediction on train data pred$_{t}rain_{R}F = fit_{R}F.predict(x_{t}rain) predictionontestdatapred_{t}est_{R}F = fit_{R}F.predict(x_{t}est)$

In[163]:

calculating RMSE for train data RMSE$_{t}rain_{R}F = np.sqrt(mean_{s}quared_{e}rror(y_{t}rain, pred_{t}rain_{R}F))calculat$ $np.sqrt(mean_{s}quared_{e}rror(y_{t}est, pred_{t}est_{R}F))$

In[165]:

print("Root Mean Squared Error For Training data = "+str(RMSE$_{t}rain_{R}F))print(''RootMeanSquaredErro$ $'' + str(RMSE_{t}est_{R}F))$

In[168]:

calculate R$^{2}fortraindata$

r2$_{s}core(y_{t}rain, pred_{t}rain_{R}F)$

In[167]:

calculate R$^{2}fortestdatar2_{s}core(y_{t}est, pred_{t}est_{R}F)$

In[ ]:

GRADIENT Boosting

In[169]:

fit$_{G}B = GradientBoostingRegressor().fit(x_{t}rain, y_{t}rain)$

In[170]:

prediction on train data pred$_{t}rain_{G}B = fit_{G}B.predict(x_{t}rain)$

prediction on test data pred$_{t}est_{G}B = fit_{G}B.predict(x_{t}est)$

In[171]:

calculating RMSE for train data RMSE$_{t}rain_{G}B = np.sqrt(mean_{s}quared_{e}rror(y_{t}rain, pred_{t}rain_{G}B))calcula$ $np.sqrt(mean_{s}quared_{e}rror(y_{t}est, pred_{t}est_{G}B))$

In[172]:

print("Root Mean Squared Error For Training data = "+str(RMSE$_{t}rain_{G}B))print(''RootMeanSquaredErro$ $'' + str(RMSE_{t}est_{G}B))$

In[173]:

calculate R$^{2}fortestdatar2_{s}core(y_{t}est, pred_{t}est_{G}B)$

In[174]:

calculate R$^{2}fortraindatar2_{s}core(y_{t}rain, pred_{t}rain_{G}B)$

In[159]:

OPTIMIZING THE RESULTS WITH PARAMETERS TUNING:

In[175]:

from sklearn.ensemble import RandomForestRegressor rf = RandomForestRegressor(random$_{s}tate = 42)frompprintimportpprintLookatparametersusedbyourcurrentforestprint('Parameterscurrentlyinuse :')pprint(rf.get_{p}arams())$

In[ ]:

Random Hyperparameter Grid

In[176]:

from sklearn.model$_{s}electionimporttrain_{t}est_{s}plit, RandomizedSearchCV$

In[177]:

Random Search CV on Random Forest Model

RRF = RandomForestRegressor(random$_{s}tate = 0)n_{e}stimator = list(range(1, 20, 2))depth = list(range(1, 100, 2))gb = GradientBoostingRegressor(random_{s}tate = 42)frompprintimportpprintLookatpar)pprint(gb.get_{p}arams())Createtherandomgridrand_{g}rid = 'n_{e}stimators' : n_{e}stimator,' max_{d}epth' : depth$

randomcv$_{r}f = RandomizedSearchCV(RRF, param_{d}istributions = rand_{g}rid, n_{i}ter = 5, cv = 5, random_{s}tate = 0)randomcv_{r}f = randomcv_{r}f.fit(x_{t}rain, y_{t}rain)predictions_{R}RF = randomcv_{r}f.predict(x_{t}est)$

view$_{b}est_{p}arams_{R}RF = randomcv_{r}f.best_{p}arams$

best$_{m}odel = randomcv_{r}f.best_{e}stimator$

predictions$_R RF = best_m odel.predict(x_t est)$

R$^2 RRF_r 2 = r2_s core(y_t est, predictions_R RF) Calculating RMSE RRF_r mse = np.sqrt(mean_s quared_e rror(y_t est,$

print('Random Search CV Random Forest Regressor Model Performance:') print('Best Parameters = ',view$_b est_p arams_R RF) print('R - squared = : 0.2.'.format(RRF_r 2)) print('RMSE =' , RRF_r mse)$

In[178]:

gb = GradientBoostingRegressor(random$_s tate = 42) from pprint import pprint Look at parameters used by you$ $) pprint(gb.get_p arams())$

In[179]:

Random Search CV on gradient boosting model

gb = GradientBoostingRegressor(random$_s tate = 0) n_e stimator = list(range(1, 20, 2)) depth = list(range(1, 100, 2))$

Create the random grid rand$_g rid = 'n_e stimators' : n_e stimator, ' max_d epth' : depth$

randomcv$_g b = RandomizedSearchCV(gb, param_d istributions = rand_g rid, n_i ter = 5, cv = 5, random_s tate = 0) randomcv_g b = randomcv_g b.fit(x_t rain, y_t rain) predictions_g b = randomcv_g b.predict(x_t est)$

view$_b est_p arams_g b = randomcv_g b.best_p arams$

best$_m odel = randomcv_g b.best_e stimator$

predictions$_g b = best_m odel.predict(x_t est)$

R$^2 gb_r 2 = r2_s core(y_t est, predictions_g b) Calculating RMSE gb_r mse = np.sqrt(mean_s quared_e rror(y_t est, predic$

print('Random Search CV Gradient Boosting Model Performance:') print('Best Parameters = ',view$_b est_p arams_g b) print('R - squared = : 0.2.'.format(gb_r 2)) print('RMSE =' , gb_r mse)$

In[180]:

from sklearn.model$_s election import GridSearchCV GridSearchCV for random Forest model regr = RandomForestRegressor(random_s tate = 0) n_e stimator = list(range(11, 20, 1)) depth = list(range(5, 15, 2))$

Create the grid grid$_s earch = 'n_e stimators' : n_e stimator, ' max_d epth' : depth$

Grid Search Cross-Validation with 5 fold CV gridcv$_r f = GridSearchCV(regr, param_g rid = grid_s earch, cv = 5) gridcv_r f = gridcv_r f.fit(x_t rain, y_t rain) view_b est_p arams_G RF = gridcv_r f.best_p arams$

Apply model on test data predictions$_G RF = gridcv_r f.predict(x_t est)$

R$^2 GRF_r 2 = r2_s core(y_t est, predictions_G RF) Calculating RMSE GRF_r mse = np.sqrt(mean_s quared_e rror(y_t est$

print('Grid Search CV Random Forest Regressor Model Performance:') print('Best Parameters = ',view$_b est_p arams_G RF) print('R - squared = : 0.2.'.format(GRF_r 2)) print('RMSE =' , (GRF_r mse))$

In[181]:

Grid Search CV for gradinet boosting gb = GradientBoostingRegressor(random$_s tate = 0) n_e stimator = list(range(11, 20, 1)) depth = list(range(5, 15, 2))$

Create the grid grid$_s earch = 'n_e stimators' : n_e stimator, ' max_d epth' : depth$

Grid Search Cross-Validation with 5 fold CV gridcv$_g b = GridSearchCV(gb, param_g rid = grid_s earch, cv = 5) gridcv_g b = gridcv_g b.fit(x_t rain, y_t rain) view_b est_p arams_G gb = gridcv_g b.best_p arams$

Apply model on test data predictions$_G gb = gridcv_g b.predict(x_t est)$

R$^2 Ggb_r 2 = r2_s core(y_t est, predictions_G gb) Calculating RMSE Ggb_r mse = np.sqrt(mean_s quared_e rror(y_t est,$

print('Grid Search CV Gradient Boosting regression Model Performance:') print('Best Parameters = ',view$_b est_p arams_G gb) print('R - squared = : 0.2.'.format(Ggb_r 2)) print('RMSE =' , (Ggb_r mse))$

Prediction of fare from provided test dataset :

We have already cleaned and processed our test dataset along with our training dataset. Hence we will be predicting using grid search CV for random forest model

In[182]:

Grid Search CV for random Forest model regr = RandomForestRegressor(random$_s tate = 0) n_e stimator = list(range(11, 20, 1)) depth = list(range(5, 15, 2))$

Create the grid grid$_s earch = 'n_e stimators' : n_e stimator,' max_d epth' : depth$

Grid Search Cross-Validation with 5 fold CV gridcv$_r f = GridSearchCV(regr, param_g rid = grid_s earch, cv = 5) gridcv_r f = gridcv_r f.fit(x_t rain, y_t rain) view_b est_p arams_G RF = gridcv_r f.best_p arams$

Apply model on test data predictions$_G RF_t est_D f = gridcv_r f.predict(test)$

In[183]:

predictions$_G RF_t est_D f = gridcv_r f.predict(test)$

In[184]:

test1=test

In[187]:

test['Predicted$_f are'] = predictions_G RF_t est_D f$

In[188]:

test.head()

In[189]:

test.to$_c sv('test.csv')$

In[ ]:

C Appendix

codeapp.py import numpy as np from flask import Flask, request, jsonify, render$_{t}emplate$
import pickle
app = Flask($_{n}ame_{)model=pickle.load(open('model.pkl','rb'))}$
@app.route('/') def home(): return render$_{t}emplate('index.html')$
@app.route('/predict',methods=['POST']) def predict(): "' For rendering results on HTML GUI "' int$_{f}eatures = [int(x) for x in request.form.values()] print(int_{f}eatures) final_{f}eatures = [np.array(int_{f}eatures)] prediction = model.predict(final_{f}eatures)$
output = round(prediction[0], 2)
return render$_{t}emplate('index.html', prediction_{t}ext =' Predicted cab fare is '$.format(output))
@app.route('/predict$_{a}pi', methods = ['POST']) def predict_{a}pi() :''' For direct API calls trought request''' data$ $request.get_{j}son(force = True) prediction = model.predict([np.array(list(data.values()))]) model.predict([[1, 2$
output = prediction[0] return jsonify(output)
if $_{n}ame_{=="}{}_{main_{":app.run(debug=True)}}$

C.1 model.py

Importing required libraries import os getting access to input files import pandas as pd Importing pandas for performing EDA import numpy as np Importing numpy for Linear Algebric operations import matplotlib.pyplot as plt Importing for Data Visualization import seaborn as sns Importing for Data Visualization from collections import Counter from sklearn.tree import DecisionTreeRegressor from sklearn.ensemble import RandomForestRegressor from sklearn.ensemble import GradientBoostingRegressor from sklearn.linear$_{m}odel import LinearRegression ML algorithm from sklearn.model_{se}$
os.chdir("D:/edWisor/edwisorproject/pythonproject") train1=pd.read$_{c}sv("train_{c}ab.csv") Loading the da$ $train = pd.read_{c}sv("train_{c}ab.csv", na_{v}alues = "pickup_{d}atetime" : "43") test = pd.read_{c}sv("test.csv") Convert$ $pd.to_{n}umeric(train["fare_{a}mount"], errors = "coerce") Using errors =' coerce'. It will replace all non- numeric values with NaN. train.dropna(subset = ["pickup_{d}atetime"])$
dropping NA values in datetime column Here pickup$_{d}atetime variable is in object so we need to change its data$ $pd.to_{d}atetime(train['pickup_{d}atetime'], format =' we will saperate the Pickup_{d}atetime column into separate field li$
train['year'] = train['pickup$_{d}atetime'].dt.year train['Month'] = train['pickup_{d}atetime'].dt.month train['Da$ $train['pickup_{d}atetime'].dt.day train['Day'] = train['pickup_{d}atetime'].dt.dayofweek train['Hour'] = train['pickup_{d}atetime'].dt.hour train['Minute'] = train['pickup_{d}atetime'].dt.minute$
removing datetime missing values rows train = train.drop(train[train['pickup$_{d}atetime'].isnull()].index, a$ $0) we see max value of passenger_{c}ount is 5345 which is actually not feasible so reducing it to 6 train = train.drop(train[train["passenger_{c}ount"] > 6].index, axis = 0) Also removing the values with passenger count of$ $train.drop(train[train["passenger_{c}ount"] == 0].index, axis = 0)$
removing passanger$_{c}ount missing values rows train = train.drop(train[train['passenger_{c}ount'].isnull()].in$ $0)$
There is one passenger count value of 0.12 which is not possible. Hence we will remove fractional passenger value train = train.drop(train[train["passenger$_{c}ount"] == 0.12].index, axis = 0) train = train.drop(train[train["fare_{a}mount"] < 0].index, axis = 0)$

Also remove the row where fare amount is zero train = train.drop(train[train["fare$_amount"] <$ $1].index, axis = 0) Nowwecanseethatthereisahugedifferencein1st2ndand3rdpositionindecendingorderoffarea$

train = train.drop(train[train["fare$_amount"] > 454].index, axis = 0) eliminatingrowsforwhichvalueof" fa$
$train.drop(train[train['fare_amount'].isnull()].index, axis = 0)$

Hence dropping the values train = train.drop((train[train['pickup$_latitude'] < -90]).index, axis =$ $0) train = train.drop((train[train['pickup_latitude'] > 90]).index, axis = 0)$

Hence dropping the values train = train.drop((train[train['pickup$_longitude'] <$ $-180]).index, axis = 0) train = train.drop((train[train['pickup_longitude'] > 180]).index, axis =$ $0)$

8ween coordinates As we know that we have given pickup longitute and latitude values and same for drop. So we need to calculate the distance Using the haversine formula and we will create a new variable called distance from math import radians, cos, sin, asin, sqrt

def haversine(a): lon1=a[0] lat1=a[1] lon2=a[2] lat2=a[3] """ Calculate the great circle distance between two points on the earth (specified in decimal degrees) """ convert decimal degrees to radians lon1, lat1, lon2, lat2 = map(radians, [lon1, lat1, lon2, lat2])

haversine formula dlon = lon2 - lon1 dlat = lat2 - lat1 a = sin(dlat/2)**2 + cos(lat1) * cos(lat2) * sin(dlon/2)**2 c = 2 * asin(sqrt(a)) Radius of earth in kilometers is 6371 km = 6371* c return km

train['distance'] = train[['pickup$_longitude',' pickup_latitude',' dropoff_longitude',' dropoff_latitude']].apply$
$1) test['distance'] = test[['pickup_longitude',' pickup_latitude',' dropoff_longitude',' dropoff_latitude']].apply(ha$
$1) wewillremovetherowswhosedistancevalueiszero$

train = train.drop(train[train['distance']== 0].index, axis=0) we will remove the rows whose distance values is very high which is more than 129kms train = train.drop(train[train['distance'] > 130].index, axis=0)

drop = ['pickup$_datetime',' pickup_longitude',' pickup_latitude',' dropoff_longitude',' dropoff_latitude',' Minu$
$train.drop(drop, axis = 1)$

test["pickup$_datetime"] = pd.to_datetime(test["pickup_datetime"], format = "wewillsaperatethePickup_datet$

test['year'] = test['pickup$_datetime'].dt.yeartest['Month'] = test['pickup_datetime'].dt.monthtest['Date'] =$
$test['pickup_datetime'].dt.daytest['Day'] = test['pickup_datetime'].dt.dayofweektest['Hour'] =$
$test['pickup_datetime'].dt.hourtest['Minute'] = test['pickup_datetime'].dt.minute$

drop$_test = ['pickup_datetime',' pickup_longitude',' pickup_latitude',' dropoff_longitude',' dropoff_latitude',' M$
$test.drop('pickup_datetime', axis = 1)$

since skewness of distance variable is high, apply log transform to reduce the skewness- train['distance'] = np.log1p(train['distance'])

train test split for further modelling x$_train, x_test, y_train, y_test = train_test_split(train.iloc[:$
$, train.columns! = ' fare_amount'], train.iloc[:, 0], test_size = 0.20, random_state = 1)$

Building model on top of training dataset fit$_LR = LinearRegression().fit(x_train, y_train) Fittingmodelwit$

import pickle Saving model to disk pickle.dump(fit$_LR, open('model.pkl',' wb'))$

Loading model to compare the results model = pickle.load(open('model.pkl','rb')) print(model.predict([[2,2011,6,17,4,9,6]]))

prediction on train data ored$_train_LR = fit_LR.predict(x_train)$

prediction on train data pred$_train_LR = fit_LR.predict(x_train)$

prediction on test data pred$_test_LR = fit_LR.predict(x_test)$

calculating RMSE for test data RMSE$_test_LR = np.sqrt(mean_squared_error(y_test, pred_test_LR))$

calculating RMSE for train data RMSE$_train_LR = np.sqrt(mean_squared_error(y_train, pred_train_LR))$

print("Root Mean Squared Error For Training data = "+str(RMSE$_train_LR)) print("RootMeanSquaredErro$
$" + str(RMSE_test_LR))$

calculate R$^2 fortraindatafromsklearn.metricsimportr2_scorer2_score(y_train, pred_train_LR)$

C.2 requirement.txt

Flask==1.1.1 gunicorn==19.9.0 itsdangerous==1.1.0 Jinja2==2.10.1 MarkupSafe==1.1.1 Werkzeug==0.15.5 numpy>=1.9.2 scipy>=0.15.1 scikit-learn>=0.18 matplotlib>=1.4.3 pandas>=0.19

C.3 Procfile

web: gunicorn app:app

C.4 index.html

<!DOCTYPE html> <html > <!–From https://codepen.io/frytyler/pen/EGdtg–> <head> <meta charset="UTF-8"> <title>ML API</title> <link href='https://fonts.googleapis.com/css?fam rel='stylesheet' type='text/css'> <link href='https://fonts.googleapis.com/css?family=Arimo' rel='stylesheet' type='text/css'> <link href='https://fonts.googleapis.com/css?family=Hind:300' rel='stylesheet' type='text/css'> <link href='https://fonts.googleapis.com/css?family=Open+Sans+Conde rel='stylesheet' type='text/css'> <link rel="stylesheet" href=" url$_{f}or('static', filename =' css/style.css')" >$

</head>

<body> <div class="login"> <h1>Predict Cab Fare Analysis</h1>

<!– Main Input For Receiving Query to our ML –> <form action=" url$_{f}or('predict')"method = "post" >< inputtype = "text"name = "passenger_{c}ount"placeholder = "passenger_{c}ount"required = "required" / >< inputtype = "text"name = "year"placeholder = "year"required = "required" / >< inputtype = "text"name = "Month"placeholder = "Month"required = "required" / >< inputtype = "text"name = "Date"placeholder = "Date"required = "required" / >< inputtype = "text"name = "Day"placeholder = "Day"required = "required" / >< inputtype = "text"name = "Hour"placeholder = "Hour"required = "required" / >< inputtype = "text"name = "distance"placeholder = "distance"required = "required" / >$

<button type="submit" class="btn btn-primary btn-block btn-large">Predict</button> </form>

 prediction$_{t}ext$

</div>

</body> </html>

r2$_{s}core(y_{t}est, pred_{t}est_{L}R)$

Bibliography

"Data Science course". In: (). URL: https://www.edwisor.com/.

Hawthorn, C. J., K. P. Weber, and R. E. Scholten (Dec. 2001). "Littrow Configuration Tunable External Cavity Diode Laser with Fixed Direction Output Beam". In: *Review of Scientific Instruments* 72.12, pp. 4477–4479. URL: http://link.aip.org/link/?RSI/72/4477/1.

"Machine Learning in Python". In: (). URL: https://scikit-learn.org/stable/.

Raschka, Sebastian (2016). "Model Evaluation, Model Selection, and Algorithm Selection in Machine Learning". In: URL: https://arxiv.org/abs/1811.12808, 2016.