Project Report – Cab Fare Prediction

By: Tarun Kumar Agarwal

Contents

Chapter 1 Introduction

1.1 Problem:

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

1.2 Data:

We have two datasets:

a. Train_cab
b. Test_cab

Number of attributes:

- pickup_datetime - timestamp value indicating when the cab ride started.
- pickup_longitude - float for longitude coordinate of where the cab ride started.
- pickup_latitude - float for latitude coordinate of where the cab ride started.
- dropoff_longitude - float for longitude coordinate of where the cab ride ended.
- dropoff_latitude - float for latitude coordinate of where the cab ride ended.
- passenger_count - an integer indicating the number of passengers in the cab ride.

Missing Values: Yes

Chapter 2 Pre-Processing of Data

In this chapter, we manipulate the data before we start modeling which includes multiple preprocessing steps such as exploring the data, cleaning the data as well as visualizing the data through graph and plots.

2.1 Data exploration and Missing Values analysis:

Firstly, we perform data exploration and cleaning which includes following points as per this project:

- Convert the data types into appropriate data types.
- We have some negative values in fare amount so we have to remove those values.
- Passenger count cannot be less than 1 and more than 6, so we remove the rows having passenger's counts more than 6 and less than 1.
- Latitude ranges from -90 to 90. Longitude ranges from -180 to 180. So, we remove the rows if any latitude and longitude lies beyond the ranges.

2.2 Creating some new variables:

Variable pickup_datetime contains date and time for pickup. So we extract some important variables from pickup_datetime:

- Year
- Month
- Date
- Day
- Hour
- Minute

```
# separate the pickup_datetime column into separate fields like year, month,day, day of the week, hour etc.
train['year'] = train['pickup_datetime'].dt.year
train['Month'] = train['pickup_datetime'].dt.month
train['Date'] = train['pickup_datetime'].dt.day
train['Day'] = train['pickup_datetime'].dt.dayofweek
train['Hour'] = train['pickup_datetime'].dt.hour
train['Minute'] = train['pickup_datetime'].dt.minute
```

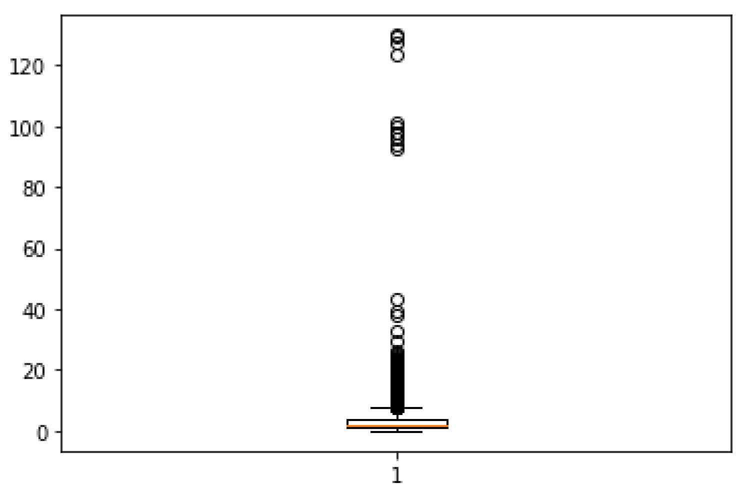
Also, we calculate distance from the latitude and longitude values using haversine formula:

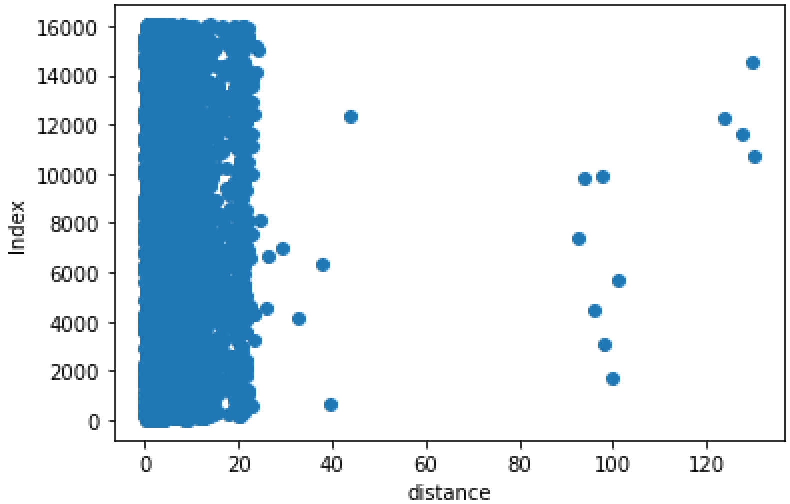
```python
# function for calculating the distance using haversine formula.
from math import radians, cos, sin, asin, sqrt

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
```

2.3 Outlier Analysis:

We performed boxplot and scatter plot analysis on continuous variables (fare_amount and distance) to check the outlier.

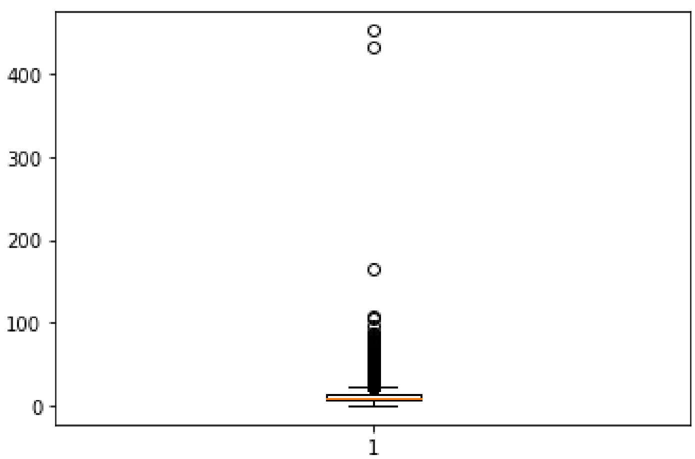
Distance variable:

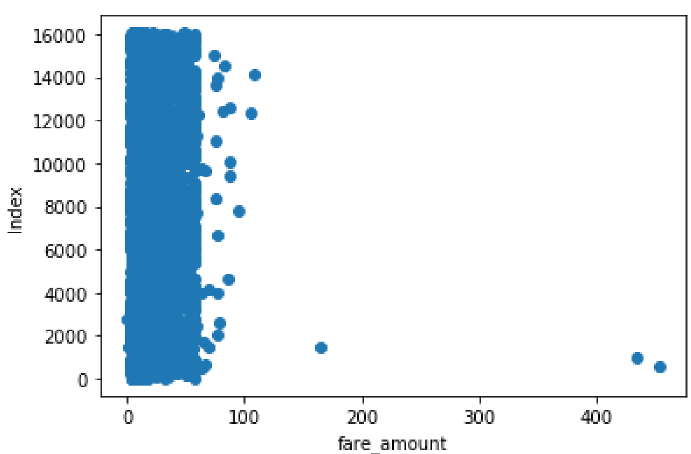




We consider distance greater than 30 km is outlier. So, we drop the rows which includes distance greater than 30 km.

Fare_amount:





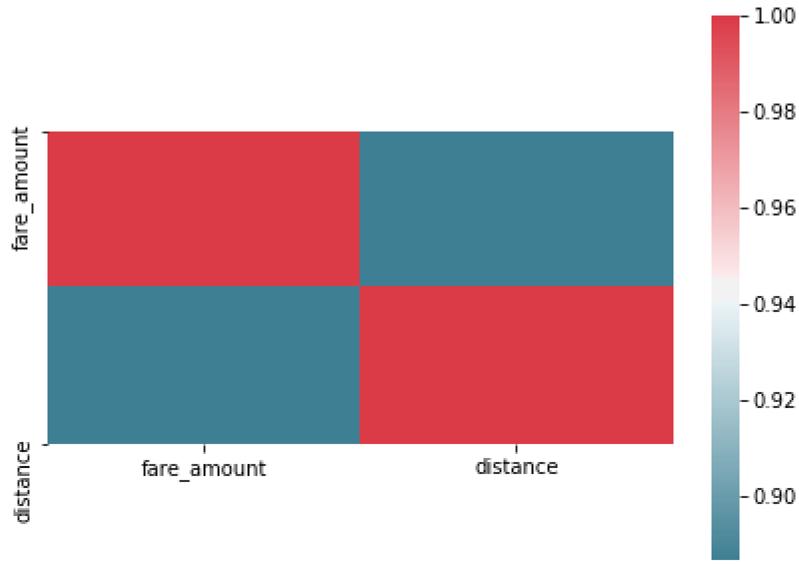
We consider fare greater than 80 is outlier. So, we drop the rows which includes fare greater than 80.

2.4 Feature Selection:

We drop the variables which were used for creating the new variables:

- pickup_datetime
- pickup_longitude
- pickup_latitude
- dropoff_longitude
- dropoff_latitude
- Minute

We also performed correlation matrix on continuous variables to check the multicolinerity.



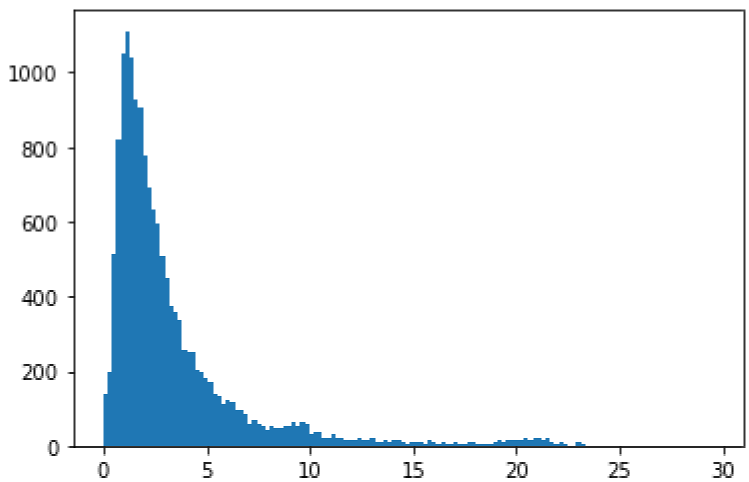
Also, we performed VIF on the data to check the multicolinerity.

```
# checking VIF for multicolinerity

from statsmodels.stats.outliers_influence import variance_inflation_factor
from statsmodels.tools.tools import add_constant

VIF_df = add_constant(train.iloc[:,1:8])
pd.Series([variance_inflation_factor(VIF_df.values, i)
               for i in range(VIF_df.shape[1])],
              index=VIF_df.columns)
```

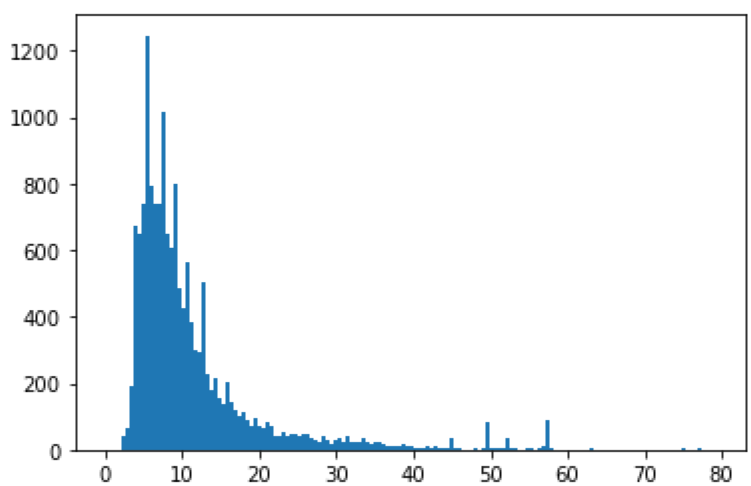
```
const             1.174942e+06
passenger_count   1.002393e+00
year              1.015236e+00
Month             1.015122e+00
Date              1.001269e+00
Day               1.010604e+00
Hour              1.010584e+00
distance          1.003526e+00
dtype: float64
```

2.5 Feature Scaling:

We check the data distribution of distance variable:



Also, we check the data distribution on fare_amount variable:



We performed the data normalization of both variables:

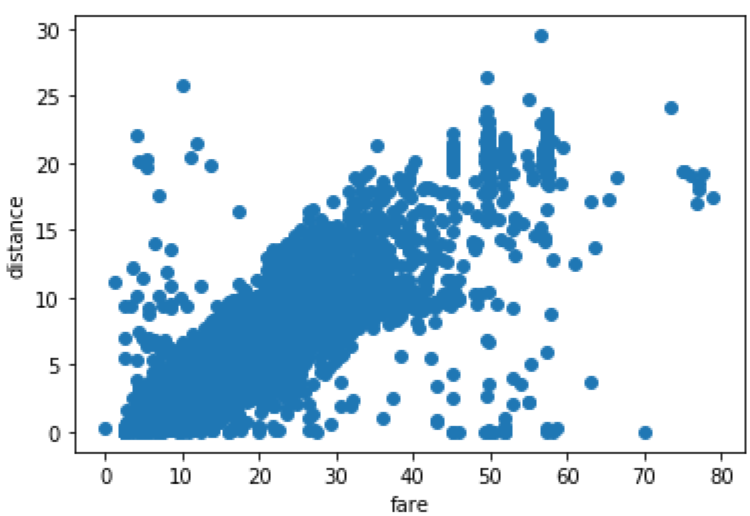
```
# performing normalization

cnames = ['fare_amount', 'distance']
for i in cnames:
    print(i)
    train[i] = (train[i] - train[i].min())/(train[i].max() - train[i].min())
```

	fare_amount	passenger_count	year	Month	Date	Day	Hour	distance
0	0.056843	1	2009	6	15	0	17	0.034963
1	0.213825	1	2010	1	5	1	16	0.286654
2	0.072034	2	2011	8	18	3	0	0.047134
3	0.097354	1	2012	4	21	5	4	0.094957
4	0.066971	1	2010	3	9	1	7	0.067814

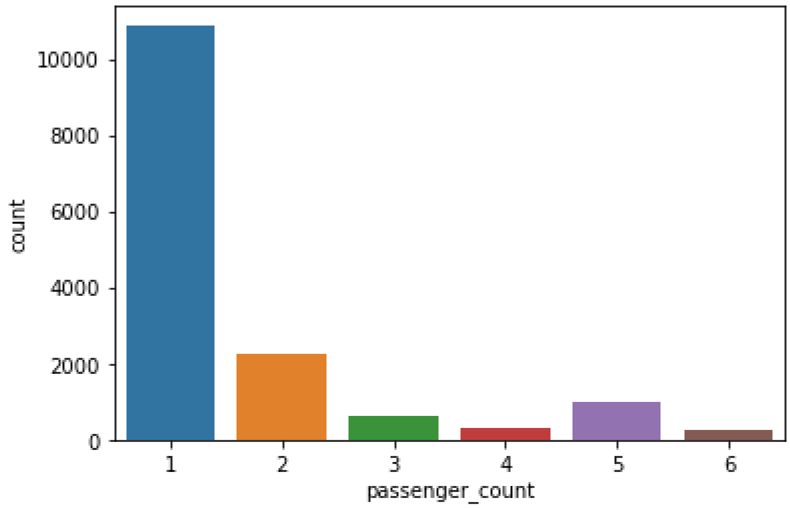
Visualization:

We also performed some visualization on the cleaned data.

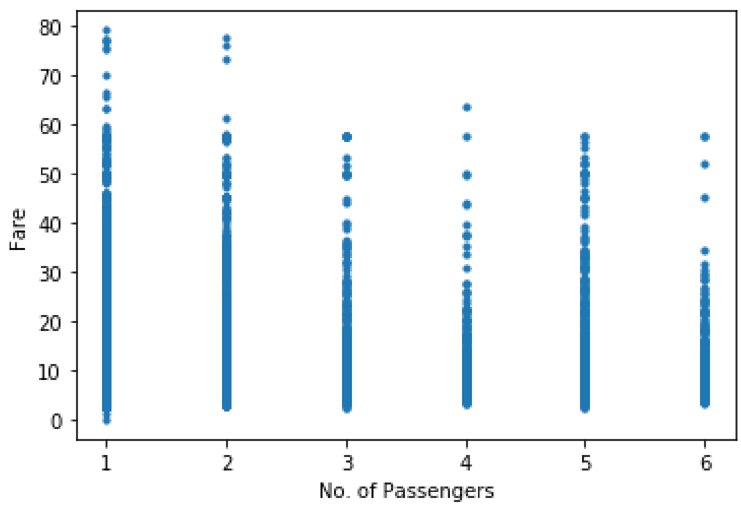
1. Plot for fare_amount variation across distance:



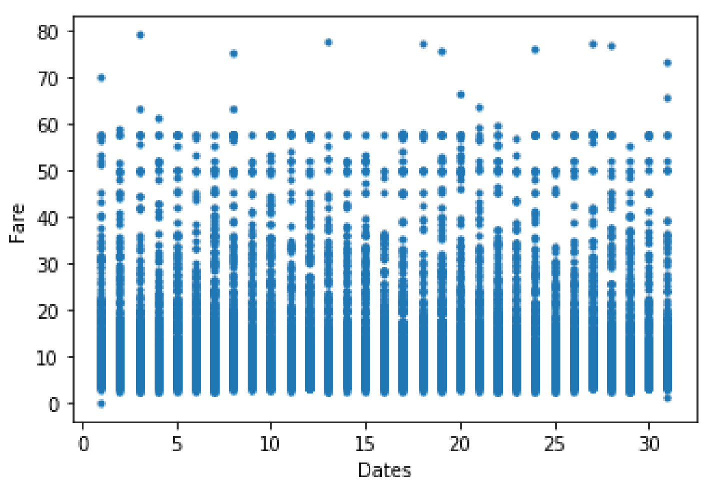
2. Count plot on passenger count:



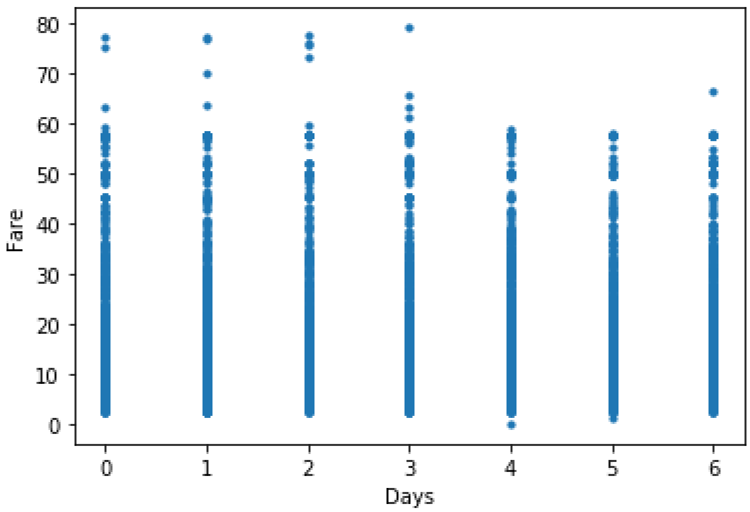
3. Relationship between fare and passengers:



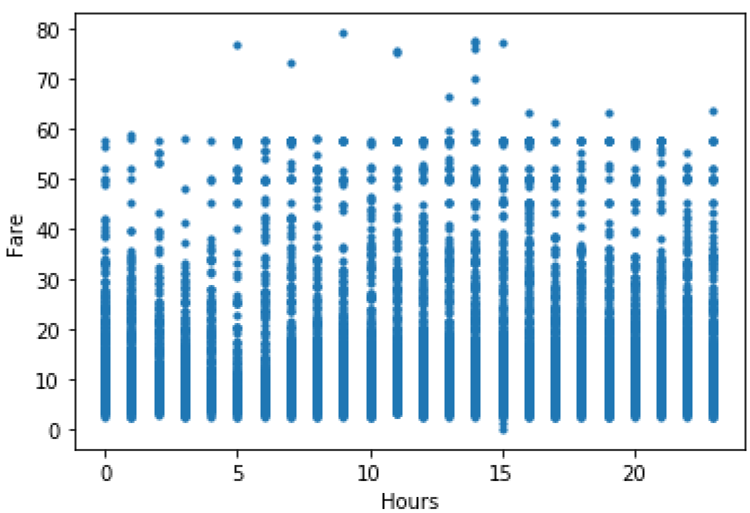
4. Relationship between fare and date:



5. Relationship between fare and day:



6. Relationship between fare and hour:



Chapter 3 Model Development

In this chapter, we firstly split the cleaned dataset into train and test and then developed different models and also performed hyper-parameter tuning.

3.1 Linear Regression Model:

```
# Build model on train data
LR = LinearRegression().fit(X_train , y_train)
```

```
# predict on train data
pred_train_LR = LR.predict(X_train)
```

```
# predict on test data
pred_test_LR = LR.predict(X_test)
```

3.2 Decision Tree Model:

```
# Build model on train data
DT = DecisionTreeRegressor(max_depth = 2).fit(X_train, y_train)
```

```
# predict on train data
pred_train_DT = DT.predict(X_train)

# predict on test data
pred_test_DT = DT.predict(X_test)
```

3.3 Random Forest Model:

```
# Build model on train data
RF = RandomForestRegressor(n_estimators = 300).fit(X_train, y_train)
```

```
# predict on train data
pred_train_RF = RF.predict(X_train)

# predict on test data
pred_test_RF = RF.predict(X_test)
```

3.4 Gradient Boosting Model:

```
# Build model on train data
GB = GradientBoostingRegressor().fit(X_train, y_train)
```

```
# predict on train data
pred_train_GB = GB.predict(X_train)

# predict on test data
pred_test_GB = GB.predict(X_test)
```

3.5 Hyper-parameter Tuning:

There are two ways to apply hyper-parameter tuning:

1. RandomizedSearchCV
2. GridSearchCV

```
# 1. RandomizedSearchCV

from sklearn.model_selection import RandomizedSearchCV

# RandomizedSearchCV on Random Forest Model

RFR = 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(RFR, param_distributions = rand_grid, n_iter = 5, cv = 5, random_state=0)
randomcv_rf = randomcv_rf.fit(X_train, y_train)
predictions_RFR = randomcv_rf.predict(X_test)

best_params_RFR = randomcv_rf.best_params_

best_estimator_RFR = randomcv_rf.best_estimator_

predictions_RFR = best_estimator_RFR.predict(X_test)
```

```python
# RandomizedSearchCV on gradient boosting model

GBR = 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(GBR, 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)

best_params_gb = randomcv_gb.best_params_

best_estimator_gb = randomcv_gb.best_estimator_

predictions_gb = best_estimator_gb.predict(X_test)
```

```python
# 2. GridSearchCV

from sklearn.model_selection import GridSearchCV

# GridSearchCV on Random Forest Model

rfr_gs = 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(rfr_gs, param_grid = grid_search, cv = 5)
gridcv_rf = gridcv_rf.fit(X_train,y_train)

best_params_GRF = gridcv_rf.best_params_
best_estimator_GRF = gridcv_rf.best_estimator_

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

```python
# GridSearchCV on gradient boosting model

gbr_gs = 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(gbr_gs, param_grid = grid_search, cv = 5)
gridcv_gb = gridcv_gb.fit(X_train,y_train)

best_params_Ggb = gridcv_gb.best_params_
best_estimator_Ggb = gridcv_gb.best_estimator_

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

Chapter 4 Results and Conclusion

4.1 Models Evaluation:

1. Linear Regression Model:

```
# Model Evaluation

# calculate RMSE on train data
RMSE_train_LR= np.sqrt(mean_squared_error(y_train, pred_train_LR))

# calculate RMSE on test data
RMSE_test_LR = np.sqrt(mean_squared_error(y_test, pred_test_LR))
```

```
print("RMSE on training data = "+str(RMSE_train_LR))
print("RMSE on test data = "+str(RMSE_test_LR))
```

```
RMSE on training data = 0.05225641654989230 5
RMSE on test data = 0.05209296672134858
```

```
# calculate R^2 on train data
r2_train_LR = r2_score(y_train, pred_train_LR)

# calculate R^2 on test data
r2_test_LR = r2_score(y_test, pred_test_LR)
```

```
print("r2 on training data = "+str(r2_train_LR))
print("r2 on test data = "+str(r2_test_LR))
```

```
r2 on training data = 0.7967162830547573
r2 on test data = 0.7996915170530652
```

2. Decision Tree Model:

```python
# Model Evaluation

# calculate RMSE on train data
RMSE_train_DT = np.sqrt(mean_squared_error(y_train, pred_train_DT))

# calculate RMSE on test data
RMSE_test_DT = np.sqrt(mean_squared_error(y_test, pred_test_DT))
```

```python
print("RMSE on training data = "+str(RMSE_train_DT))
print("RMSE on test data = "+str(RMSE_test_DT))
```

```
RMSE on training data = 0.05882202678524862
RMSE on test data = 0.0595765542682677
```

```python
# calculate R^2 on train data
r2_train_DT = r2_score(y_train, pred_train_DT)

# calculate R^2 on test data
r2_test_DT = r2_score(y_test, pred_test_DT)
```

```python
print("r2 on training data = "+str(r2_train_DT))
print("r2 on test data = "+str(r2_test_DT))
```

```
r2 on training data = 0.7424252350137317
r2 on test data = 0.7380056525207052
```

3. Random Forest Model:

```python
# Model Evaluation

# calculate RMSE on train data
RMSE_train_RF = np.sqrt(mean_squared_error(y_train, pred_train_RF))

# calculate RMSE on test data
RMSE_test_RF = np.sqrt(mean_squared_error(y_test, pred_test_RF))
```

```python
print("RMSE on training data = "+str(RMSE_train_RF))
print("RMSE on test data = "+str(RMSE_test_RF))
```

```
RMSE on training data = 0.0188235582953822
RMSE on test data = 0.053362956566434236
```

```python
# calculate R^2 on train data
r2_train_RF = r2_score(y_train, pred_train_RF)

# calculate R^2 on test data
r2_test_RF = r2_score(y_test, pred_test_RF)
```

```python
print("r2 on training data = "+str(r2_train_RF))
print("r2 on test data = "+str(r2_test_RF))
```

```
r2 on training data = 0.9736229155518966
r2 on test data = 0.7898057041771843
```

4. Gradient Booting Model:

```
# Model Evaluation

# calculate RMSE on train data
RMSE_train_GB = np.sqrt(mean_squared_error(y_train, pred_train_GB))

# calculate RMSE on test data
RMSE_test_GB = np.sqrt(mean_squared_error(y_test, pred_test_GB))
```

```
print("RMSE on training data = "+str(RMSE_train_GB))
print("RMSE on test data = "+str(RMSE_test_GB))
```

```
RMSE on training data = 0.04283819595073212
RMSE on test data = 0.05221474459579668
```

```
# calculate R^2 on train data
r2_train_GB = r2_score(y_train, pred_train_GB)

# calculate R^2 on test data
r2_test_GB = r2_score(y_test, pred_test_GB)
```

```
print("r2 on training data = "+str(r2_train_GB))
print("r2 on test data = "+str(r2_test_GB))
```

```
r2 on training data = 0.8633889940883133
r2 on test data = 0.7987538989912932
```

5. Hyper-parameter Tuning:

```
RandomizedSearchCV - Random Forest Regressor Model Performance:
Best Parameters =  {'n_estimators': 15, 'max_depth': 9}
R-squared = 0.79.
RMSE =  0.05360261946279395

RandomizedSearchCV - Gradient Boosting Model Performance:
Best Parameters =  {'n_estimators': 15, 'max_depth': 9}
R-squared = 0.75.
RMSE =  0.0579931618343988

GridSearchCV - Random Forest Regressor Model Performance:
Best Parameters =  {'max_depth': 5, 'n_estimators': 19}
R-squared = 0.8.
RMSE =  0.05243106129916764
```

```
Grid Search CV Gradient Boosting regression Model Performance:
Best Parameters =  {'max_depth': 5, 'n_estimators': 19}
R-squared = 0.78.
RMSE =  0.05445782146149547
```

From above results, it is clear that GridSearchCV on Random Forest Model is providing best results having R-squared = 0.8 and RMSE = 0.05243

4.2 Predicting Fare on Test Data:

We have already cleaned the test dataset, so we are applying GridSearchCV on Random Forest Model on Test data.

```python
# GridSearchCV for random Forest model - test data fare prediction

rfr_test = 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_test = GridSearchCV(rfr_test, param_grid = grid_search, cv = 5)
gridcv_rf_test = gridcv_rf_test.fit(X, y)

best_params_GRF_test = gridcv_rf_test.best_params_
best_estimator_GRF_test = gridcv_rf_test.best_estimator_

# Apply model on test data
predictions_GRF_test = best_estimator_GRF_test.predict(test)

print('Best Parameters = ',best_params_GRF_test)
```

```
Best Parameters =  {'max_depth': 7, 'n_estimators': 19}
```

	passenger_count	year	Month	Date	Day	Hour	distance	Predicted_fare
0	1	2015	1	27	1	13	2.323259	9.835905
1	1	2015	1	27	1	13	2.425353	10.380099
2	1	2011	10	8	5	11	0.618628	5.002537
3	1	2012	12	1	5	21	1.961033	7.878283
4	1	2012	12	1	5	21	5.387301	15.531962

References

1. https://stackoverflow.com/questions/43577086/pandas-calculate-haversine-distance-within-each-group-of-rows/43577275
2. https://www.r-bloggers.com/great-circle-distance-calculations-in-r/
3. https://www.analyticsvidhya.com/blog/2016/01/xgboost-algorithm-easy-steps/
4. http://benalexkeen.com/gradient-boosting-in-python-using-scikit-learn/
5. https://www.analyticsvidhya.com/blog/2016/02/complete-guide-parameter-tuning-gradient-boosting-gbm-python/
6. https://towardsdatascience.com/hyperparameter-tuning-the-random-forest-in-python-using-scikit-learn-28d2aa77dd74

Appendix

R code – Cab Fare Prediction

```
rm(list=ls())


# set working directory

setwd("F:/R_Programming/Edwisor")

getwd()


#################################################################


# loading Libraries

x = c("tidyr", "ggplot2", "corrgram", "usdm", "caret", "DMwR", "rpart", "randomForest",'xgboost')


# tidyr - drop_na

# ggplot2 - for visulization, boxplot, scatterplot

# corrgram - correlation plot

# usdm - vif

# caret - createDataPartition

# DMwR - regr.eval

# rpart - decision tree

# randomForest - random forest

# xgboost - xgboost


# load Packages

lapply(x, require, character.only = TRUE)
```

```
rm(x)

#############################################################

# loading datasets
train = read.csv("train_cab.csv", header = T, na.strings = c(" ", "", "NA"))
test = read.csv("test_cab.csv")

######################
# Exploring Datasets
######################

# Structure of data
str(train)
str(test)

# Summary of data
summary(train)
summary(test)

# Viewing the data
head(train,5)
head(test,5)

#####################################
```

```
# EDA, Missing value and Outlier analysis

#####################################

# Changing the data types of variables

train$fare_amount = as.numeric(as.character(train$fare_amount))

train$passenger_count = round(train$passenger_count)


# Checking Missing data

apply(train, 2, function(x) {sum(is.na(x))})


#Creating dataframe with missing values present in each variable

val = data.frame(apply(train, 2, function(x){sum(is.na(x))}))

val$Columns = row.names(val)

names(val)[1] = "null_percentage"


#Calculating percentage missing value

val$null_percentage = (val$null_percentage/nrow(train)) * 100


# Sorting null_val in Descending order

val = val[order(-val$null_percentage),]

row.names(val) = NULL

# Reordering columns

val = val[,c(2,1)]


#viewing the % of missing data for all variales

val
```

```
# delete the rows having missing values

train = drop_na(train)


# Verifying missing values after deletion

sum(is.na(train))


#Splitting Date and time on train data

train$pickup_date = as.Date(as.character(train$pickup_datetime))

train$weekday = as.factor(format(train$pickup_date,"%u")) # Monday = 1

train$month = as.factor(format(train$pickup_date,"%m"))

train$year = as.factor(format(train$pickup_date,"%Y"))

pickup_time = strptime(train$pickup_datetime,"%Y-%m-%d %H:%M:%S")

train$hour = as.factor(format(pickup_time,"%H"))


# Now drop the column pickup_datetime and pickup_date from train data

train = subset(train, select = -c(pickup_datetime))

train = subset(train, select = -c(pickup_date))


#Splitting Date and time on test data

test$pickup_date = as.Date(as.character(test$pickup_datetime))

test$weekday = as.factor(format(test$pickup_date,"%u")) # Monday = 1

test$month = as.factor(format(test$pickup_date,"%m"))

test$year = as.factor(format(test$pickup_date,"%Y"))

pickup_time_test = strptime(test$pickup_datetime,"%Y-%m-%d %H:%M:%S")
```

```
test$hour = as.factor(format(pickup_time_test,"%H"))

# Now drop the column pickup_datetime and pickup_date from test data
test = subset(test, select = -c(pickup_datetime))
test = subset(test, select = -c(pickup_date))

# Make a copy
df_train = train
df_test = test

#train = df_train
#test = df_test

# calculate distance
my_dist = function(long1, lat1, long2, lat2) {
  rad = pi/180
  a1 = lat1*rad
  a2 = long1*rad
  b1 = lat2*rad
  b2 = long2*rad
  dlon = b2 - a2
  dlat = b1 - a1
  a = (sin(dlat/2))^2 + cos(a1)*cos(b1)*(sin(dlon/2))^2
  c = 2*atan2(sqrt(a), sqrt(1 - a))
  R = 6371
```

```
  d = R*c

  return(d)

}


#Running the distance function for all rows in train dataframe

for (i in 1:nrow(train)){

  train$distance[i]=                    my_dist(train$pickup_longitude[i],                    train$pickup_latitude[i],
train$dropoff_longitude[i], train$dropoff_latitude[i])

}


#Running the distance function for all rows in test dataframe

for (i in 1:nrow(test)){

  test$distance[i]=  my_dist(test$pickup_longitude[i],  test$pickup_latitude[i],  test$dropoff_longitude[i],
test$dropoff_latitude[i])

}


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

train = subset(train,select = -c(pickup_longitude,pickup_latitude,dropoff_longitude,dropoff_latitude))

test = subset(test,select = -c(pickup_longitude,pickup_latitude,dropoff_longitude,dropoff_latitude))


summary(train)


# in train data, passangers count can not be zero or more than 6

# so, remove rows having passangers count zero or more than 6

train$passenger_count[train$passenger_count<1] = NA

train$passenger_count[train$passenger_count>6] = NA
```

```
sum(is.na(train))

train = drop_na(train)


# Make a copy

df_train1 = train

df_test1 = test


# creating boxplot of the continous variables to check outlier

ggplot(data = train, aes(x = "", y = distance)) +

  geom_boxplot() +

  coord_cartesian(ylim = c(0, 150))


ggplot(data = train, aes(x = "", y = fare_amount)) +

  geom_boxplot() +

  coord_cartesian(ylim = c(0, 150))


# sorting fare data in decending order to check outlier

train_fare_dec = train$fare_amount

train_fare_dec = sort(train_fare_dec, decreasing = TRUE, na.last = TRUE)


# fare_amount greater than 100 seems to be outlier

# fare_amount can not be zero  or negative

# drop rows having fare negative, zero and greater than 100.
```

```
train$fare_amount[train$fare_amount<1] = NA

train$fare_amount[train$fare_amount>100] = NA


sum(is.na(train))

train = drop_na(train)


# sorting distance in decending order to check outlier for train data

train_distance_dec = train$distance

train_distance_dec = sort(train_distance_dec, decreasing = TRUE, na.last = TRUE)


# distance greater than 30 km seems to be outlier

# distance can not be zero  or negative

# drop rows having distance negative, zero and greater than 30 km.


train$distance[train$distance<=0] = NA

train$distance[train$distance>30] = NA


sum(is.na(train))

train = drop_na(train)


# sorting distance in decending order to check outlier for test data

test_distance_dec = test$distance

test_distance_dec = sort(test_distance_dec, decreasing = TRUE, na.last = TRUE)


test$distance[test$distance<=0] = NA
```

```
test$distance[test$distance>30] = NA


sum(is.na(test))

test = drop_na(test)


# Make a copy

df_train2 = train

df_test2 = test

#train=df_train2

#test=df_test2


############### Visualization #######################

# Scatter plot between distance and fare on train data

ggplot(data = train, aes_string(x = train$distance, y = train$fare_amount))+

  geom_point()


# Scatter plot between passenger and fare on train data

ggplot(data = train, aes_string(x = train$passenger_count, y = train$fare_amount))+

  geom_point()


# Scatter plot between weekday and fare on train data

ggplot(data = train, aes_string(x = train$weekday, y = train$fare_amount))+

  geom_point()


# Scatter plot between hour and fare on train data
```

```r
ggplot(data = train, aes_string(x = train$hour, y = train$fare_amount))+
  geom_point()

#################### Feature Selection ######################
# generate correlation plot between numeric variables

numeric_index=sapply(train, is.numeric)
corrgram(train[,numeric_index], order=F, upper.panel=panel.pie,
         text.panel=panel.txt, main="Correlation plot")

# check VIF
vif(train[,-1])
# if vif is greater than 10 then variable is not suitable/multicollinerity

# creating dummy variables for categorical variables
# require(fastDummies)
# results = fastDummies::dummy_cols(train)

############### Feature Scaling ################
# check normality - single continous variable
qqnorm(train$fare_amount)
hist(train$fare_amount)

qqnorm(train$distance)
hist(train$distance)
```

```
train[,'distance'] = (train[,'distance'] - min(train[,'distance']))/
  (max(train[,'distance'] - min(train[,'distance'])))

test[,'distance'] = (test[,'distance'] - min(test[,'distance']))/
  (max(test[,'distance'] - min(test[,'distance'])))

######################### Model Development ###################

############ Splitting train into train and test ###################
set.seed(101)
split_index = createDataPartition(train$fare_amount, p = 0.75, list = FALSE)
train_data = train[split_index,]
test_data = train[-split_index,]

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

# summary of trained model
summary(lm_model)

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

```
# prediction on test_data
lm_predictions = predict(lm_model,test_data[,2:7])

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

regr.eval(test_data[,1],lm_predictions)
#       mae        mse       rmse       mape
#     2.1719005 17.8044656  4.2195338  0.2130411

# compute r^2
rss_lm = sum((lm_predictions - test_data$fare_amount) ^ 2)
tss_lm = sum((test_data$fare_amount - mean(test_data$fare_amount)) ^ 2)
rsq_lm = 1 - rss_lm/tss_lm
#    r^2 - 0.7966621

############## Decision Tree Model ###############
Dt_model = rpart(fare_amount ~ ., data=train_data, method = "anova")

# summary on trainned model
summary(Dt_model)

#Prediction on test_data
predictions_DT = predict(Dt_model, test_data[,2:7])
```

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

regr.eval(test_data[,1], predictions_DT)
#      mae       mse      rmse      mape
#    2.498668 20.471238  4.524515  0.255439

# compute r^2
rss_dt = sum((predictions_DT - test_data$fare_amount) ^ 2)
tss_dt = sum((test_data$fare_amount - mean(test_data$fare_amount)) ^ 2)
rsq_dt = 1 - rss_dt/tss_dt
#    r^2 - 0.766206

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

# summary on trained model
summary(rf_model)

# prediction of test_data
rf_predictions = predict(rf_model, test_data[,2:7])

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

regr.eval(test_data[,1], rf_predictions)
#       mae       mse       rmse       mape
```

```
#     2.2628045 18.4661108  4.2972213  0.2370227


# compute r^2

rss_rf = sum((rf_predictions - test_data$fare_amount) ^ 2)

tss_rf = sum((test_data$fare_amount - mean(test_data$fare_amount)) ^ 2)

rsq_rf = 1 - rss_rf/tss_rf

#    r^2 - 0.7891057


############  XGBOOST Model ###########################

train_data_matrix = as.matrix(sapply(train_data[-1],as.numeric))

test_data_matrix = as.matrix(sapply(test_data[-1],as.numeric))


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


# summary of trained model

summary(xgboost_model)


# prediction on test_data

xgb_predictions = predict(xgboost_model,test_data_matrix)


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


regr.eval(test_data[,1], xgb_predictions)

#       mae       mse       rmse       mape

#     2.0897725 18.5837037  4.3108820  0.2181592
```

```
# compute r^2

rss_xgb = sum((xgb_predictions - test_data$fare_amount) ^ 2)

tss_xgb = sum((test_data$fare_amount - mean(test_data$fare_amount)) ^ 2)

rsq_xgb = 1 - rss_xgb/tss_xgb

#    r^2 - 0.7977628


# from above models, it is clear that xgboost is best model

# so, we are using xgboost to predict test dataset


#############   Final Test data prediction   ###################

# we have already clean the test data

# we use whole training Dataset to predict the fare on test dataset

train_data_matrix2 = as.matrix(sapply(train[-1],as.numeric))

test_data_matrix2 = as.matrix(sapply(test,as.numeric))


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


# Lets now predict on test dataset

xgb = predict(xgboost_model2, test_data_matrix2)


test_xgb_pred = data.frame(df_test2$passenger_count, df_test2$distance,"predictions_fare" = xgb)


write.csv(test_xgb_pred,"test_cab_predicted_fare.csv",row.names = FALSE)
```