Project Report – Bike Rental Count Prediction

Submitted By: Megha Tapali

Contents

Chapter 1 Introduction

1.1 Problem:

The project is about a bike rental company who has its historical data, and now our objective of this Project is to predict the bike rental count on daily basis, considering the environmental and seasonal settings. These predicted values will help the business to meet the demand on those particular days by maintain the amount of supply.

Nowadays there are number of bike renting companies like, Ola Bikes, Rapido etc. And these bike renting companies deliver services to lakhs of customers daily. Now it becomes really important to manage their data properly to come up with new business ideas to get best results. In this case we have to identify in which days there can be most demand, such that we have enough strategies met to deal with such demand.

1.2 Data:

The details of data attributes in the dataset are as follows –

instant: Record index

dteday: Date

season: Season (1:springer, 2:summer, 3:fall, 4:winter)

yr: Year (0: 2011, 1:2012)

mnth: Month (1 to 12)

hr: Hour (0 to 23)

holiday: weather day is holiday or not (extracted from Holiday

Schedule) weekday: Day of the week

workingday: If day is neither weekend nor holiday is 1, otherwise is 0.

weathersit: (extracted from Freemeteo)

1: Clear, Few clouds, Partly cloudy, Partly cloudy

2: Mist + Cloudy, Mist + Broken clouds, Mist + Few clouds, Mist

3: Light Snow, Light Rain + Thunderstorm + Scattered clouds, Light Rain + Scattered clouds

4: Heavy Rain + Ice Pallets + Thunderstorm + Mist, Snow + Fog

temp: Normalized temperature in Celsius. The values are derived via (t-t_min)/(t_max-t_min), t_min=-8, t_max=+39 (only in hourly scale)

atemp: Normalized feeling temperature in Celsius. The values are derived via (tt_min)/(t_maxt_min), t_min=-16, t_max=+50 (only in hourly scale)

hum: Normalized humidity. The values are divided to 100 (max)

windspeed: Normalized wind speed. The values are divided to 67 (max)

casual: count of casual users

registered: count of registered users

cnt: count of total rental bikes including both casual and registered

Chapter 2 METHODOLOGY

After going through the dataset in detail and pre-understanding the data the next step is, Methodology that will help achieve our goal.

In Methodology following processes are followed:

Pre-processing: It includes missing value analysis, outlier analysis, feature selection and feature scaling.

Model development: It includes identifying suitable Machine learning Algorithms and applying those algorithms in our given dataset.

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

2.1 Data exploration, Missing Values and Outlier analysis:

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

- Data Cleaning and Convert the data types into appropriate data types.
- Check the missing values in the data.

1. Variable "instant" can be dropped as it simply represents the index.
2. casual and registered variables can be removed, as these two sums to dependent variable count
3. Variable "dteday" can be ignored as output is not based on time series analysis. And we already have data in year and month column

```
#Converting variables datatype to required datatypes

data['season'] = data['season'].astype(str)
data['yr']     = data['yr'].astype(str)
data['mnth']   = data['mnth'].astype(str)
data['holiday']= data['holiday'].astype(str)
data['weekday']= data['weekday'].astype(str)
data['workingday']= data['workingday'].astype(str)
data['weathersit']= data['weathersit'].astype(str)
```

```
data.isnull().sum()
```

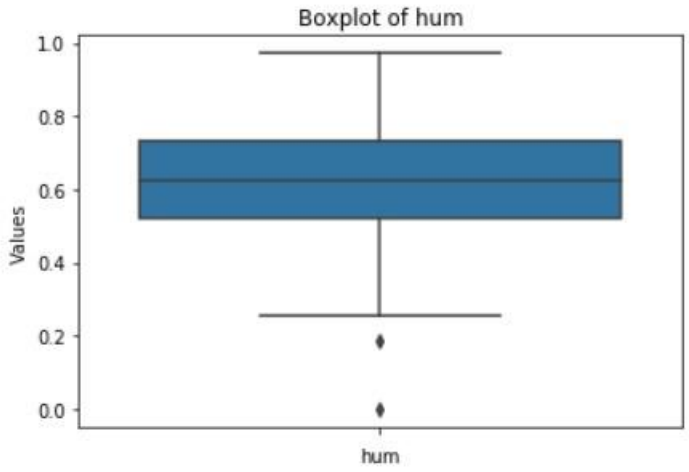
```
season        0
yr            0
mnth          0
holiday       0
weekday       0
workingday    0
weathersit    0
temp          0
atemp         0
hum           0
windspeed     0
cnt           0
dtype: int64
```

As there are no missing values found in our given data, thus we don't need to follow imputation processes here. So, we can directly move to our next step that is outlier analysis.

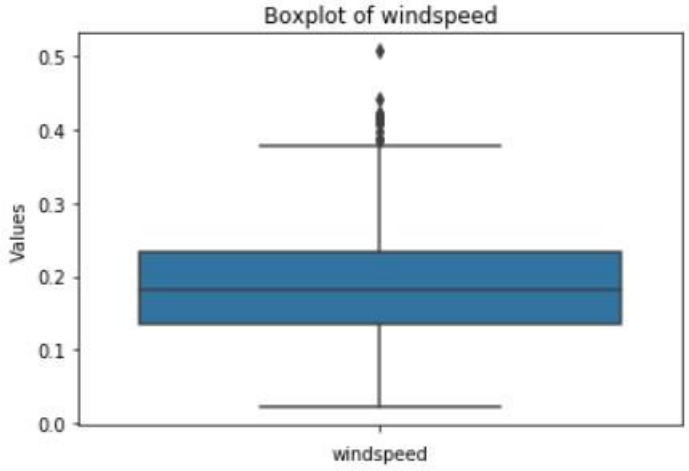
Outlier Analysis:

Outlier is an abnormal observation that stands or deviates away from other observations. These happens because of manual error, poor quality of data and it is correct but exceptional data. But, it can cause an error in predicting the target variables. So we have to check for outliers in our data set and also remove or replace the outliers wherever required.
In this project, outliers are found in only two variables this are Humidity and windspeed, following are the box plots for both the variables and dots outside the quartile ranges are outliers.

hum



windspeed



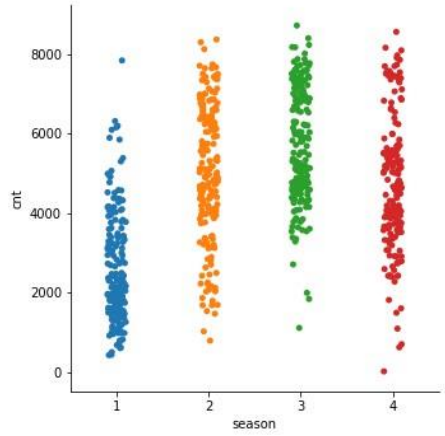
Plot: Outliers

All this outliers mentioned above happened because of manual error, or interchange of data, or may be correct data but exceptional. But all these outliers can hamper our data model. So there is a requirement to eliminate or replace such outliers, and impute with proper methods to get better accuracy of the model. In this project, I used mean method to impute the outliers in windspeed and humidity variables.

Data Understanding :

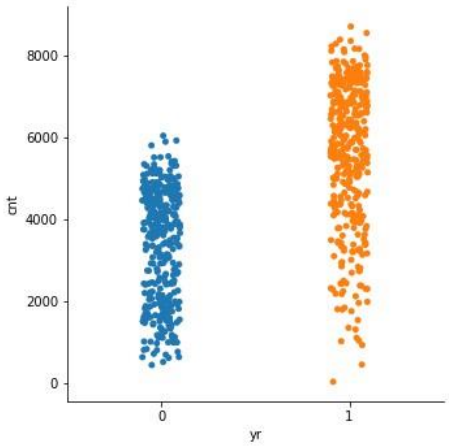
Data Understand is a process where we know our data in a better way by the help of visual representations and come up with initial ideas to develop our model. Here, the specific variables are plotted with respect to the target variable. In some cases two variables are compared, whereas in some cases three variables are plotted together for our better understanding and visualization.

a. Season



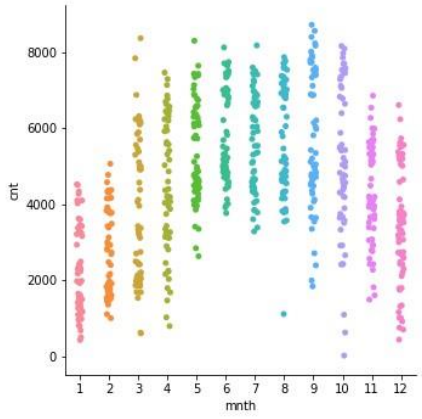
Here, it is found that in Season 2, 3 and 4 has the highest count

b. Year



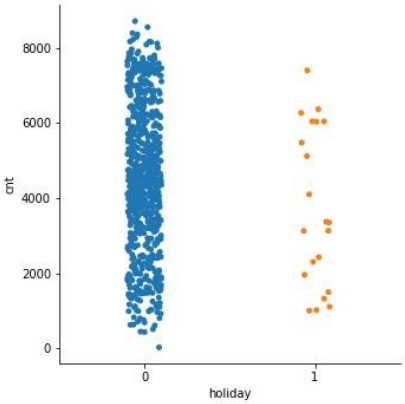
Here, it is found that in Year 1 has high count than 0

c. Month



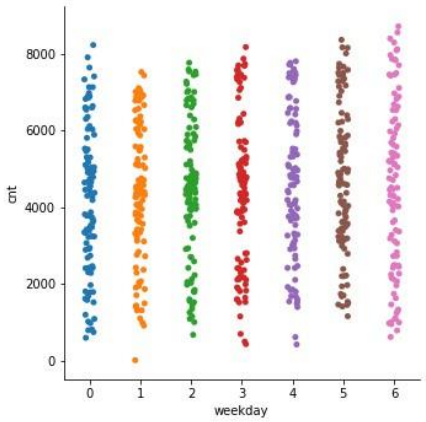
Here, it is observed that in Months 3 to 10 we got a good number of count

d. Holidays and Non-Holidays



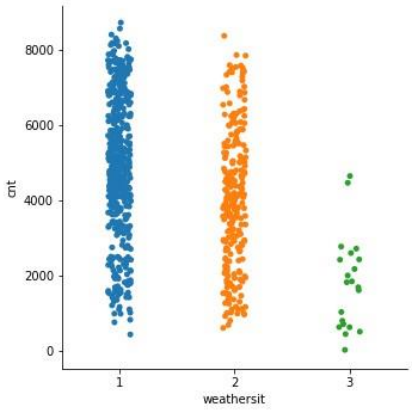
Here, it is found that, on holidays the count is higher when compared non-holidays

e. Weekdays



Here, it is observed that in weekdays, 0 and 6 i.e., Monday to Saturday the count is highest.

f. Weather

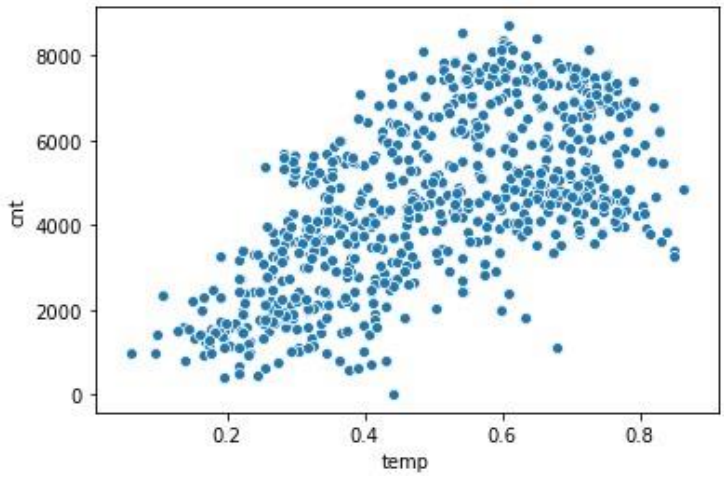


Here, in weather it is observed that, weather 1 has the highest count.

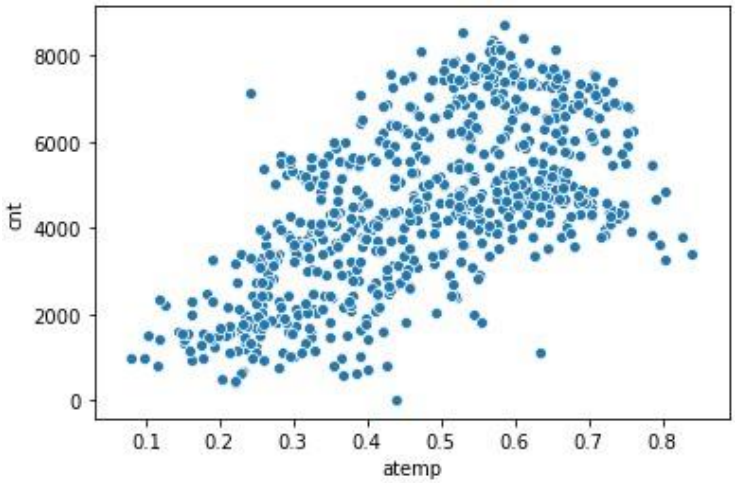
Visualization:

We also performed some visualization on the cleaned data.

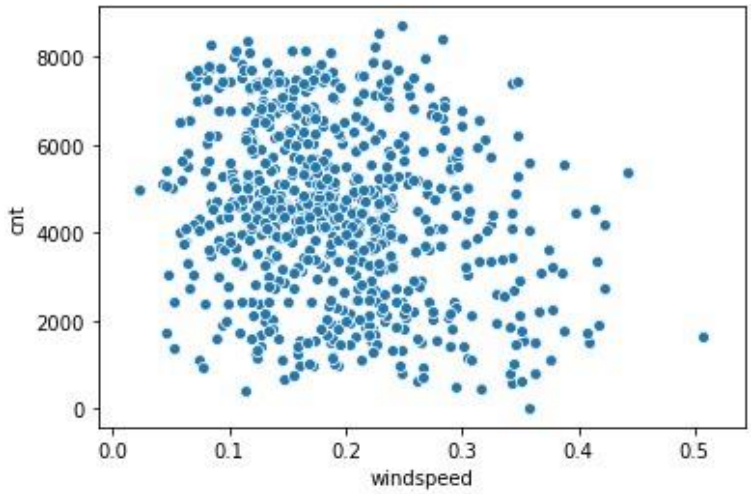
1. Scatter plot between temp and cnt:



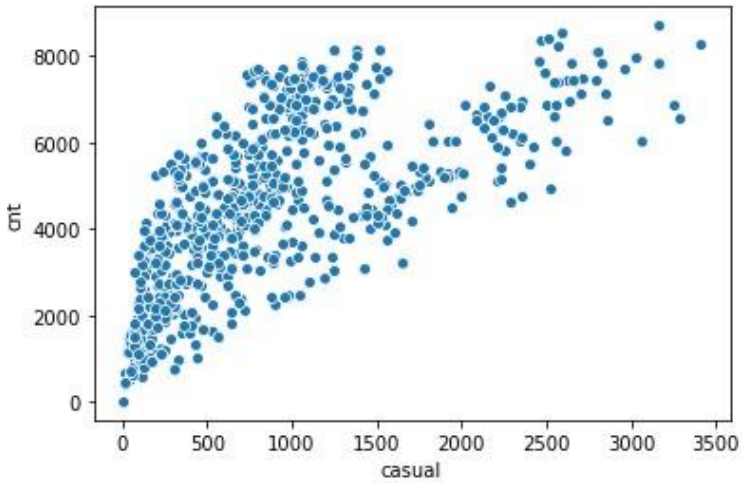
2. Scatter plot between atemp and cnt:



3. Scatter plot between windspeed and cnt:



4. Scatter plot between hum and cnt:



2.2 Feature Selection and Scaling:

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

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

```
# checking VIF for multicolinerity for continous variables

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

VIF_data = add_constant(data.iloc[:,7:11])
pd.Series([variance_inflation_factor(VIF_data.values, i)
               for i in range(VIF_data.shape[1])],
              index=VIF_data.columns)
```

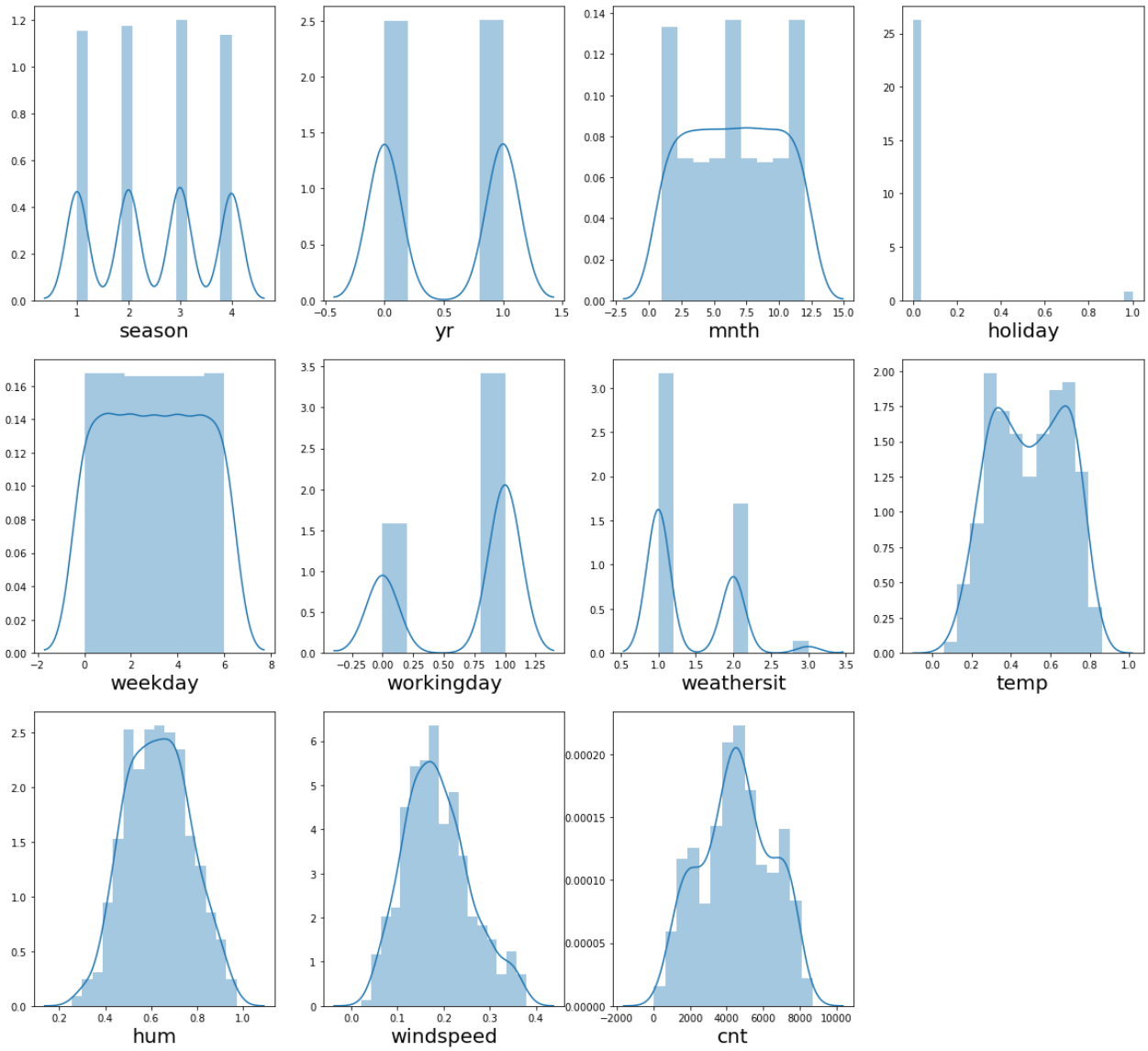
```
const        45.499530
temp         63.010048
atemp        63.632085
hum           1.059230
windspeed     1.097383
dtype: float64
```

Observations:

- From heatmap and VIF, we remove variables atemp because it is highly correlated with temp. So, we drop atemp column.

Visualizations:

Now, we plot distribution plot for the cleaned data to check the distribution of data.



Since the data is distributed normally, scaling isn't required. We can proceed with modelling.

Chapter 3 Model Development

The next step after Exploratory Data Analysis and Data Pre-Processing is Model Development. Now we have our data ready to be implemented to develop a model. There are number of models and Machine learning algorithms that are used to develop model, some are like decision tree, random forest, SVM, KNN, Naïve Bayes, Linear regression, Logistic Regression etc. So, before implementing any model we have to choose precisely our model. So, the first step in Model Development is selection of model.

Model Selection

As per industry standards, there are four categories of models that are derived by classifying problem statement and goal of the project. These categories are:

- Forecasting
- Classification
- Optimization
- Unsupervised Learning

The process of selecting precise model depends on our goal and the problem statement. In this project the problem statement is to predict the bike rental count on daily basis, considering the environmental and seasonal settings. Thus, the problem statement is an identified as regression problem and falls under the category of forecasting, where we have to forecast a numeric data or continuous variable for the target.

Basis of understanding the criteria and given data's problem statement. In this project we got our target variable as "count". The model has to predict a numeric value. Thus, it is identified that this is a Regression problem statement. And to develop a **regression** model, the various models that can be used are Decision trees, Random Forest and Linear Regression.

Splitting train and Validation Dataset

a) We have used sklearn's train_test_split() method to divide whole Dataset into train and validation dataset.
b) 20% is in validation dataset and 80% is in training data.
c) We will test the performance of model on validation dataset.
d) The model which performs best will be chosen to perform on test dataset provided along with original train dataset.
e) X_train y_train--are train subset.
f) X_test y_test--are validation subset.

EVALUATION METRICS OF THE MODEL

So, now we have developed few models for predicting the target variable, now the next step is evaluate the models and identify which one to choose for deployment. To decide these, error metrics are used. In this project MAPE, R Square and Accuracy are used.

1. Mean Absolute Percentage Error (MAPE)

MAE or Mean Absolute Error, it is one of the error measures that is used to calculate the predictive performance of the model. It is the sum of calculated errors. In this project we will apply this measure to our models. We choose the model with lowest MAPE as a suitable Model. But we need more error metrics to cross check this. So, we go for R Square which is a better error metric.

2. Accuracy

The second matric to identify or compare for better model is Accuracy. It is the ratio of number of correct predictions to the total number of predictions made.

Accuracy= number of correct predictions / Total predictions made

It can also be calculated from MAE as
Accuracy = 1- MAPE

As, Accuracy derives from MAE/MAPE its observations also suggest same models as better models as suggested by MAPE. Here, the models with highest accuracy are chosen.

3. R Squared

R Square is another metric that helps us to know about the Correlation between original and predicted values. R Square is identified as a better error metric to evaluate models.

3.1 Linear Regression

It is used to predict the value of variable Y based on one or more input predictor variables X. The goal of this method is to establish a linear relationship between the predictor variables and the response variable. Such that, we can use this formula to estimate the value of the response Y, when only the predictors (X- Values) are known. In this project Linear Regression is applied in both R and Python, details are described following.

Error Metrics in Python:

R- squared	0.8074472692401635
MAPE	24.046271082160825
Accuracy	75.95372891783917

Error metrics in R:

MAE	5.288626e+02
MSE	4.837572e+05
RMSE	6.955265e+02
MAPE	1.739758e-01
R- squared	0.8649468

3.2 Decision Tree

Decision Tree is a supervised learning predictive model that uses a set of binary rules to calculate the target value/dependent variable. Decision trees are divided into three main parts this are:

- Root Node: performs the first split
- Terminal Nodes: that predict the outcome, these are also called leaf nodes
- Branches: arrows connecting nodes, showing the flow from root to other leaves.

In this project Decision tree is applied in both R and Python, details are described following.

Error Metrics in Python:

R- squared	0.6464697716428666
MAPE	36.94809301452646
Accuracy	63.05190698547354

Error metrics in R: using regr.eval function

MAE	7.258519e+02
MSE	9.767130e+05
RMSE	9.882879e+02
MAPE	2.898936e-01
R- squared	0.7273256

3.3 Random Forest

The next model to be followed in this project is Random forest. It is a process where the machine follows an ensemble learning method for classification and regression that operates by developing a number of decision trees at training time and giving output as the class that is the mode of the classes of all the individual decision trees. For regression, it takes mean of output of all individual decision trees

In this project Random Forest is applied in both R and Python, details are described following.

Error Metrics in Python:

R- squared	0.8768854596234804
MAPE	21.03514074116917
Accuracy	78.96485925883083

Error metrics in R:

MAE	5.122251e+02
MSE	4.948365e+05
RMSE	7.034462e+02
MAPE	1.978135e-01
R-squared	0.8618538

3.4 Gradient Boosting:

The next model to be followed in this project is Gradient Boosting. Gradient boosting is a type of machine learning boosting. It relies on the intuition that the best possible next model, when combined with previous models, minimizes the overall prediction error. The key idea is to set the target outcomes for this next model in order to minimize the error.

In this project Gradient Boosting is applied in both R and Python, details are described following.

Error Metrics in Python:

R- squared	0.8839146143324043
MAPE	18.695706619022438
Accuracy	81.30429338097755

Error metrics in R:

MAE	4.876667e+02
MSE	4.252179e+05
RMSE	6.520873e+02
MAPE	1.583279e-01
R-squared	0.8812896

As observed in above 4 models, random forest and gradient boosting model has better performance. We will now tune the hyper parameters of Random Forest Model and gradient boosting model for more accuracy

3.5 Hyper parameter Tuning

1. Grid Search CV for Random Forest:

Grid search is an approach to hyperparameter tuning that will methodically build and evaluate a model for each combination of algorithm parameters specified in a grid. GridSearchCV is a library function that is a member of sklearn's model_selection package. In addition to that, we can specify the number of times for the cross-validation for each set of hyperparameters.

Here we are giving below ranges for the parameters:

n_estimator = list(range(11,20,1))

depth = list(range(5,15,2))

And cross validation with 5 fold CV

And we got below parameters as best parameters:

```
Grid Search CV Random Forest Regressor Model Performance:
Best Parameters =  {'max_depth': 13, 'n_estimators': 12}
R-squared = 0.87.
MAPE =21.308642569783053
Accuracy = 78.69%.
```

2. Grid Search CV for Gradient Boosting:

Here we are giving below ranges for the parameters:

n_estimator = list(range(100,120,2))

depth = list(range(1,10,2))

And cross validation with 5 fold CV

And we got below parameters as best parameters:

```
Grid Search CV Gradient Boosting Regressor Model Performance:
Best Parameters =  {'max_depth': 3, 'n_estimators': 110}
R-squared = 0.88.
MAPE =18.829153637435322
Accuracy = 81.17%.
```

3.6 Model Selection:

After comparison of the error matrix, the next step we come to is Selection of the most effective model. From the values of Error and accuracy, it is found that Random Forest and Gradient Boosting models perform close to each other. And after tuning the hyper parameters for Gradient boosting model, the accuracy remains almost same. So, we will prefer Gradient Boosting Model to be used for further processes.

From above models, it is clear that Gradient Boosting Model is providing best results having R-squared = 0.88 and Accuracy = 88.3%

References

1. https://www.analyticsvidhya.com/blog/2016/01/xgboost-algorithm-easy-steps/
2. http://benalexkeen.com/gradient-boosting-in-python-using-scikit-learn/
3. https://www.analyticsvidhya.com/blog/2016/02/complete-guide-parameter-tuning-gradientboosting-gbm-python/
4. https://towardsdatascience.com/hyperparameter-tuning-the-random-forest-in-python-usingscikit-learn-28d2aa77dd74

Appendix

R code – Bike Rental Count

```
rm(list=ls())


# set working directory
setwd("D:/Edwisor/Project 2")
getwd()

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

# loading Libraries
install.packages (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



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

# loading dataset
df = read.csv("day.csv", header = T, na.strings = c(" ", "", "NA"))

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

# Structure of data
str(df)
```

```
# Summary of data
summary(df)

# Viewing the data

head(df,5)

########################################
# EDA, Missing value and Outlier analysis
########################################

#Variable "instant" can be dropped as it simply represents the index.
#casual and registered variables can be removed, as these two sums to dependent variable
count
#Variable "dteday" can be ignored as output is not based on time series analysis. And we
already have data in year and month column

df = subset(df, select = -c(instant,casual, registered, dteday) )

#Converting variables datatype to required datatypes
catnames=c("season","yr","mnth","holiday","weekday","workingday","weathersit")
for(i in catnames){
  print(i)
  df[,i]=as.factor(df[,i])
}

#Checking datatypes
str(df)


#########Checking Missing data##########
sum(is.na(df))

# No missing values are present in the given data set.

#########Outlier Analysis############

num_index = sapply(df, is.numeric)
numeric_data = df[,num_index]
num_cnames = colnames(numeric_data)

for (i in 1:length(num_cnames))
{
  assign(paste0("gn",i), ggplot(aes_string(y = (num_cnames[i]), x = "cnt"), data = subset(df))+
           stat_boxplot(geom = "errorbar", width = 0.5) +
```

```
        geom_boxplot(outlier.colour="red", fill = "grey" ,outlier.shape=18,
                     outlier.size=1, notch=FALSE) +
        theme(legend.position="bottom")+
        labs(y=num_cnames[i],x="cnt")+
        ggtitle(paste("Box plot of count for",num_cnames[i])))
}

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


# continous variables hum and windspeed  includes outliers


outlier_var=c("hum","windspeed")

#Replace all outliers with NA

for(i in outlier_var){
  val = df[,i][df[,i] %in% boxplot.stats(df[,i])$out]
  print(length(val))
  df[,i][df[,i] %in% val] = NA
}

# Checking Missing data - after outlier
sum(is.na(df))

# hum includes 2 outliers and windspeed includes 13 outliers, so impute them with mean
df$hum[is.na(df$hum)] <- mean(df$hum, na.rm = TRUE)
df$windspeed[is.na(df$windspeed)] <- mean(df$windspeed, na.rm = TRUE)

# Checking Missing data - after imputation
sum(is.na(df))


################## Visualization ############################
# Scatter plot between temp and cnt
ggplot(data = df, aes_string(x = df$temp, y = df$cnt))+
  geom_point()

# Scatter plot between atemp and cnt
ggplot(data = df, aes_string(x = df$atemp, y = df$cnt))+
  geom_point()
```

```
# Scatter plot between hum and cnt
ggplot(data = df, aes_string(x = df$hum, y = df$cnt))+
  geom_point()

# Scatter plot between windspeed and cnt
ggplot(data = df, aes_string(x = df$windspeed, y = df$cnt))+
  geom_point()

# Scatter plot between season and cnt
ggplot(data = df, aes_string(x = df$season, y = df$cnt))+
  geom_point()

# Scatter plot between month and cnt
ggplot(data = df, aes_string(x = df$mnth, y = df$cnt))+
  geom_point()

# Scatter plot between weekday and cnt
ggplot(data = df, aes_string(x = df$weekday, y = df$cnt))+
  geom_point()




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

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

# check VIF
library(usdm)
vif(df[,8:10])
# if vif is greater than 10 then variable is not suitable/multicollinerity
#1      temp 61.087196
#2     atemp 61.308023
#3       hum  1.029048

#From heatmap and VIF, Removing variables atemp beacuse it is highly correlated with temp
df = subset(df, select = -c(atemp) )

str(df)
```

```
# generate histogram of continous variables
d=density(df$temp)
plot(d,main="distribution")
polygon(d,col="green",border="red")

e=density(df$hum)
plot(e,main="distribution")
polygon(e,col="green",border="red")


f=density(df$windspeed)
plot(f,main="distribution")
polygon(f,col="green",border="red")

#Since the data is distributed normally, scaling isn't required. We can proceed with data splitting


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

############# Splitting df into train and test ###################
set.seed(101)
install.packages("caret")
split_index = createDataPartition(df$cnt, p = 0.80, list = FALSE)
train_data = df[split_index,]
test_data = df[-split_index,]


##############  Linear regression Model  ##################

lm_model = lm(cnt ~., data=train_data)

# summary of trained model
summary(lm_model)

# prediction on test_data
lm_predictions = predict(lm_model,test_data[,1:10])

regr.eval(test_data[,11],lm_predictions)
#   mae          mse          rmse       mape
#5.288626e+02 4.837572e+05 6.955265e+02 1.739758e-01

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

```
print(rsq_lm)
#    r^2 = 0.8649468


############### Decision Tree Model ################

Dt_model = rpart(cnt ~ ., data=train_data, method = "anova")

# summary on trainned model
summary(Dt_model)

#Prediction on test_data
predictions_DT = predict(Dt_model, test_data[,1:10])

regr.eval(test_data[,11], predictions_DT)
#   mae          mse          rmse         mape
# 7.258519e+02 9.767130e+05 9.882879e+02 2.898936e-01

# compute r^2
rss_dt = sum((predictions_DT - test_data$cnt) ^ 2)
tss_dt = sum((test_data$cnt - mean(test_data$cnt)) ^ 2)
rsq_dt = 1 - rss_dt/tss_dt
print(rsq_dt)
#    r^2 - 0.7273256

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

# summary on trained model
summary(rf_model)

# prediction of test_data
rf_predictions = predict(rf_model, test_data[,1:10])

regr.eval(test_data[,11], rf_predictions)
#  mae          mse          rmse         mape
# 5.122251e+02 4.948365e+05 7.034462e+02 1.978135e-01

# compute r^2
rss_rf = sum((rf_predictions - test_data$cnt) ^ 2)
tss_rf = sum((test_data$cnt - mean(test_data$cnt)) ^ 2)
rsq_rf = 1 - rss_rf/tss_rf
print(rsq_rf)
#    r^2 - 0.8618538
```

```
############  XGBOOST Model ##############################
train_data_matrix = as.matrix(sapply(train_data[-11],as.numeric))
test_data_matrix = as.matrix(sapply(test_data[-11],as.numeric))

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

# summary of trained model
summary(xgboost_model)

# prediction on test_data
xgb_predictions = predict(xgboost_model,test_data_matrix)

regr.eval(test_data[,11], xgb_predictions)
#   mae          mse          rmse       mape
#4.876667e+02 4.252179e+05 6.520873e+02 1.583279e-01

# compute r^2
rss_xgb = sum((xgb_predictions - test_data$cnt) ^ 2)
tss_xgb = sum((test_data$cnt - mean(test_data$cnt)) ^ 2)
rsq_xgb = 1 - rss_xgb/tss_xgb
print(rsq_xgb)
#    r^2 = 0.8812896




# from above models, it is clear that xgboost performs better.
```

Python code – Bike Rental Count

```
#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 sklearn.metrics import r2_score,  mean_squared_error
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression

%matplotlib inline

#Setting working directory

os.chdir("D:/Edwisor/Project 2")

print(os.getcwd())

D:\Edwisor\Project 2

#Loading the data

data = pd.read_csv("day.csv")

Understanding the Data:

#checking first five rows of the  dataset

data.head()

instant
dteday
season
yr
mnth
holiday
weekday
workingday
weathersit
temp
atemp
hum
```

windspeed
casual
registered
cnt
0
1
2011-01-01
1
0
1
0
6
0
2
0.344167
0.363625
0.805833
0.160446
331
654
985
1
2
2011-01-02
1
0
1
0
0
0
2
0.363478
0.353739
0.696087
0.248539
131
670
801
2
3
2011-01-03
1

0
1
0
1
1
1
0.196364
0.189405
0.437273
0.248309
120
1229
1349
3
4
2011-01-04
1
0
1
0
2
1
1
0.200000
0.212122
0.590435
0.160296
108
1454
1562
4
5
2011-01-05
1
0
1
0
3
1
1
0.226957
0.229270

0.436957
0.186900
82
1518
1600

#checking the number of rows and columns in dataset
print("shape of data is: ",data.shape)

shape of data is: (731, 16)

#checking the data-types in dataset
data.dtypes

instant int64
dteday object
season int64
yr int64
mnth int64
holiday int64
weekday int64
workingday int64
weathersit int64
temp float64
atemp float64
hum float64
windspeed float64
casual int64
registered int64
cnt int64
dtype: object

data.describe()

instant
season
yr
mnth
holiday
weekday
workingday
weathersit
temp
atemp
hum
windspeed
casual
registered

cnt
count
731.000000
731.000000
731.000000
731.000000
731.000000
731.000000
731.000000
731.000000
731.000000
731.000000
731.000000
731.000000
731.000000
731.000000
731.000000
mean
366.000000
2.496580
0.500684
6.519836
0.028728
2.997264
0.683995
1.395349
0.495385
0.474354
0.627894
0.190486
848.176471
3656.172367
4504.348837
std
211.165812
1.110807
0.500342
3.451913
0.167155
2.004787
0.465233
0.544894

0.183051
0.162961
0.142429
0.077498
686.622488
1560.256377
1937.211452
min
1.000000
1.000000
0.000000
1.000000
0.000000
0.000000
0.000000
1.000000
0.059130
0.079070
0.000000
0.022392
2.000000
20.000000
22.000000
25%
183.500000
2.000000
0.000000
4.000000
0.000000
1.000000
0.000000
1.000000
0.337083
0.337842
0.520000
0.134950
315.500000
2497.000000
3152.000000
50%
366.000000
3.000000

1.000000
7.000000
0.000000
3.000000
1.000000
1.000000
0.498333
0.486733
0.626667
0.180975
713.000000
3662.000000
4548.000000
75%
548.500000
3.000000
1.000000
10.000000
0.000000
5.000000
1.000000
2.000000
0.655417
0.608602
0.730209
0.233214
1096.000000
4776.500000
5956.000000
max
731.000000
4.000000
1.000000
12.000000
1.000000
6.000000
1.000000
3.000000
0.861667
0.840896
0.972500
0.507463

3410.000000
6946.000000
8714.000000

Data Cleaning, Missing Value Analysis And Outliers Analysis:

1. Variable "instant" can be dropped as it simply represents the index.
2. casual and registered variables can be removed, as these two sums to dependent variable count
3. Variable "dteday" can be ignored as output is not based on time series analysis. And we alraedy have data in year and month column

```
data = data.drop(columns=['instant','dteday', 'casual', 'registered'])
```

#Converting variables datatype to required datatypes

```
data['season'] = data['season'].astype(str)
data['yr']     = data['yr'].astype(str)
data['mnth']   = data['mnth'].astype(str)
data['holiday']= data['holiday'].astype(str)
data['weekday']= data['weekday'].astype(str)
data['workingday']= data['workingday'].astype(str)
data['weathersit']= data['weathersit'].astype(str)
```

```
data.dtypes
```

```
season         object
yr             object
mnth           object
holiday        object
weekday        object
workingday     object
weathersit     object
temp          float64
atemp         float64
hum           float64
windspeed     float64
cnt             int64
dtype: object
```

#Defining numeric and categorical variables and saving in specific array

```
numeric_var = ['temp', 'atemp', 'hum', 'windspeed', 'cnt']

categorical_var = ['season', 'yr', 'mnth', 'holiday', 'weekday', 'workingday', 'weathersit']
```

Missing Value Analysis

```
data.isnull().sum()
```

```
season        0
yr            0
```

```
mnth          0
holiday       0
weekday       0
workingday    0
weathersit    0
temp          0
atemp         0
hum           0
windspeed     0
cnt           0
dtype: int64
```

No missing values found

Outlier Analysis

```
# checking boxplot of continous variables

for i in numeric_var:
    print(i)
    sns.boxplot(y = data[i])
    plt.xlabel(i)
    plt.ylabel("Values")
    plt.title("Boxplot of " + i)
    plt.show()
```

temp

png

png

atemp

png

png

hum

png

png

windspeed

png

png

cnt

png

png

Outliers are found in windspeed and humidity variables.

```
#Scatter plot for temp against cnt
sns.scatterplot(data=data,x='temp',y='cnt')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x192cffe6df0>
```

png

png

```
#Scatter plot for atemp against cnt
sns.scatterplot(data=data,x='atemp',y='cnt')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x192d3bb7580>
```

png

png

```
#Scatter plot for windspeed against cnt
sns.scatterplot(data=data,x='windspeed',y='cnt')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x192d3c0ef70>
```

png

png

```
#Scatter plot for hum against cnt
sns.scatterplot(data=data,x='hum',y='cnt')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x192d3c5a2e0>
```

png

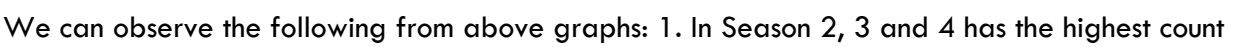
png

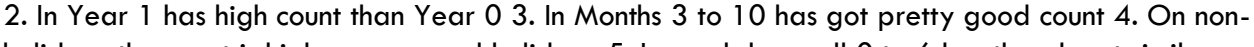
```
# from the boxplot analysis, it is clear that continous variables windspeed, hum and casual includes th
e outliers.

count_names = ['windspeed','hum']
for i in count_names:
    print (i)
    q75,q25 = np.percentile(data.loc[:,i],[75,25])
    iqr = q75-q25
    min = q25 - (iqr*1.5)
    max = q75 + (iqr*1.5)
    print (min)
    print (max)
```

```
    data.loc[data[i]<min,i]=np.nan
    data.loc[data[i]>max,i]=np.nan

windspeed
-0.012446750000000034
0.38061125
hum
0.20468725
1.0455212500000002

# checking missing values
data.isnull().sum()

season         0
yr             0
mnth           0
holiday        0
weekday        0
workingday     0
weathersit     0
temp           0
atemp          0
hum            2
windspeed     13
cnt            0
dtype: int64

# total 15 outliers found. Now, we will impute the values with mean as humidity and windspeed is al
most normally distributed

data['hum'] = data['hum'].fillna(data['hum'].mean())
data['windspeed'] = data['windspeed'].fillna(data['windspeed'].mean())

data.isnull().sum()

season         0
yr             0
mnth           0
holiday        0
weekday        0
workingday     0
weathersit     0
temp           0
atemp          0
hum            0
windspeed      0
cnt            0
dtype: int64
```

Data Understanding

```
for i in categorical_var:
    sns.catplot(x = i, y = "cnt", data=data)
```





We can observe the following from above graphs: 1. In Season 2, 3 and 4 has the highest count 2. In Year 1 has high count than Year 0 3. In Months 3 to 10 has got pretty good count 4. On non-holidays the count is higher compared holidays 5. In weekdays, all 0 to 6 has the almost similar count 6. In weather, 1 has the highest count

Feature Selection and Scaling

```
data.columns
```

```
Index(['season', 'yr', 'mnth', 'holiday', 'weekday', 'workingday',
       'weathersit', 'temp', 'atemp', 'hum', 'windspeed', 'cnt'],
      dtype='object')
```

```
# Correlation Analysis on continous variables
# generating heatmap

cnames = ['temp','atemp', 'hum', 'windspeed', 'cnt']
data_corr = data.loc[:,cnames]
f, ax = plt.subplots(figsize=(7, 5))
```

```
# correlation matrix
corr = data_corr.corr()

sns.heatmap(corr, ax=ax, linewidths=0.5, vmin= -1.0, annot=True)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x192d3aca490>
```

png


```
# checking VIF for multicolinerity for continous variables

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

VIF_data = add_constant(data.iloc[:,7:11])
pd.Series([variance_inflation_factor(VIF_data.values, i)
           for i in range(VIF_data.shape[1])],
          index=VIF_data.columns)
```

```
const        45.499530
temp         63.010048
atemp        63.632085
hum           1.059230
windspeed     1.097383
dtype: float64
```

From heatmap and VIF, Removing variables atemp beacuse it is highly correlated with temp

```
data = data.drop(columns=['atemp'])

# let's see how data is distributed for every column
plt.figure(figsize=(20,25), facecolor='white')
plotnumber = 1

for column in data:
    if plotnumber<=16 :
        ax = plt.subplot(4,4,plotnumber)
        sns.distplot(data[column])
        plt.xlabel(column,fontsize=20)
    plotnumber+=1
plt.show()
```

```
C:\Users\Megha\anaconda3\lib\site-packages\seaborn\distributions.py:369: UserWarning: De
fault bandwidth for data is 0; skipping density estimation.
  warnings.warn(msg, UserWarning)
```

png

Since the data is distributed normally, scaling isn't required. We can proceed with data splitting

categorical_var

['season', 'yr', 'mnth', 'holiday', 'weekday', 'workingday', 'weathersit']

MODEL DEVELOPMENT

 # Create dummy variables

```
data = pd.get_dummies(data, columns = categorical_var)
```

```
data.shape
```

(731, 36)

#predictors and target

```
X = data.drop(['cnt'], axis = "columns")
y = data['cnt']
```

#divide the data into train and test

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=.20, random_state=0)
```

```
# creating function to calulate MAPE
def MAPE(y_actual, y_predicted):
    MAPE = np.mean(np.abs(y_actual-y_predicted)/y_actual)*100
    return MAPE
```

Linear Regression

```
lr= LinearRegression()
```

```
lr.fit(X_train, y_train)
```

LinearRegression()

```
y_pred_LR= lr.predict(X_test)
```

```
r2_score(y_test, y_pred_LR)
```

0.8074472692401635

```
Mape_LR= MAPE(y_test, y_pred_LR)
print("MAPE ="+str(Mape_LR))
```

MAPE =24.046271082160825

```
Accuracy_LR = 100 - Mape_LR
print("Accuracy= " + str(Accuracy_LR))
```

Accuracy= 75.95372891783917

Decision Tree

```
fit_DT = DecisionTreeRegressor(max_depth = 2).fit(X_train,y_train)

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

## R^2 calculation for test data
r2_score(y_test, y_pred_DT)
```

0.6464697716428666

```
Mape_DT = MAPE(y_test, y_pred_DT)
print("MAPE ="+str(Mape_DT))
```

MAPE =36.94809301452646

```
Accuracy_DT = 100 - Mape_DT
print("Accuracy= " + str(Accuracy_DT))
```

Accuracy= 63.05190698547354

Random Forest

```
rf = RandomForestRegressor(n_estimators=10)

rf.fit(X_train,y_train)
```

RandomForestRegressor(n_estimators=10)

```
y_pred_RF= rf.predict(X_test)

r2_score(y_test, y_pred_RF)
```

0.8768854596234804

```
Mape_RF = MAPE(y_test, y_pred_RF)
print("MAPE ="+str(Mape_RF))
```

MAPE =21.035140741169 17

```
Accuracy_RF = 100 - Mape_RF
print("Accuracy= " + str(Accuracy_RF))
```

Accuracy= 78.96485925883083

Gradient Boosting

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

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

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

0.8841148608179593

```
Mape_GB = MAPE(y_test, pred_test_GB)
print("MAPE ="+str(Mape_GB))
```

MAPE =18.66493900147446

```
Accuracy_GB = 100 - Mape_GB
print("Accuracy= " + str(Accuracy_GB))
```

Accuracy= 81.33506099852553

GridSearchCV for Random Forest

```
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 MAPE
Mape_gRF = MAPE(y_test, predictions_GRF)

# Calculating Accuracy
accuracy = 100 - Mape_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("MAPE ="+str(Mape_gRF))
print('Accuracy = {:0.2f}%.'.format(accuracy))
```

Grid Search CV Random Forest Regressor Model Performance:
Best Parameters = {'max_depth': 13, 'n_estimators': 12}

R-squared = 0.87.
MAPE =21.308642569783053
Accuracy = 78.69%.

GridSearchCV for Gradient Boosting

```
gbr_gs = GradientBoostingRegressor(random_state = 0)
n_estimator = list(range(100,120,2))
depth = list(range(1,10,2))

# 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)

#R^2
GGB_r2 = r2_score(y_test, predictions_Ggb)

#Calculating MAPE
Mape_gGB = MAPE(y_test, predictions_Ggb)

# Calculating Accuracy
accuracy = 100 - Mape_gGB


print('Grid Search CV Random Forest Regressor Model Performance:')
print('Best Parameters = ',best_params_Ggb)
print('R-squared = {:0.2}.'.format(GGB_r2))
print("MAPE ="+str(Mape_gGB))
print('Accuracy = {:0.2f}%.'.format(accuracy))
```

Grid Search CV Random Forest Regressor Model Performance:
Best Parameters = {'max_depth': 3, 'n_estimators': 110}
R-squared = 0.88.
MAPE =18.829153637435322
Accuracy = 81.17%.

From all above models, we can see that Gradient Boosting Regressor performs well with with 81.33% accuracy

Example of output with a sample input.

#Using Gradient Boosting model
GB.predict([[0.2,0.4,0.1,1,0,0,0,1,0,1,0,0,0,0,0,0,0,0,0,0,0,1,0,1,0,0,0,0,0,0,0,1,0,0,1]])

array([1664.98890512])