Prediction of Bike Rental Count

Poorna Sai Manikanta Chimirala

July 2019

Contents

1.1 Problem Statement

The objective of this case study is the prediction of bike rental count on daily based on the environmental and seasonal settings. The dataset contains 731 observations, 15 predictors and 1 target variable. The predictors are describing various environment factors and settings like season, humidity etc. We need to build a prediction model to predict estimated count or demand of bikes on a particular day based on the environmental factors

1.2 Data

Our task is to build Regression model which will give the daily count of rental bikes based on weather and season Given below is a sample of the data set that we are using to predict the count: The data set consist of 731 observation recorded over a period of 2 years, between 2011 and 2012. It has 15 predictors or variables and 1 target variable..

Table 1.1: Bike Rental Sample Data (Columns: 1-8)

instant	dteday	Season	yr	mnth	holiday	weekday	workingday
1	1/1/2011	1	0	1	0	6	0
2	1/2/2011	1	0	1	0	0	0
3	1/3/2011	1	0	1	0	1	1
4	1/4/2011	1	0	1	0	2	1
5	1/5/2011	1	0	1	0	3	1

Table 1.2: Bike Rental Sample Data (Columns: 9-16)

weathersit	temp	atemp	Hum	windspeed	casual	registered	cnt
2	0.344167	0.363625	0.805833	0.160446	331	654	985
2	0.363478	0.353739	0.696087	0.248539	131	670	801
1	0.196364	0.189405	0.437273	0.248309	120	1229	1349
1	0.2	0.212122	0.590435	0.160296	108	1454	1562
1	0.226957	0.22927	0.436957	0.1869	82	1518	1600

Below are the variables we used to predict the count of bike rentals

Table 1.3: Bike Rental Predictors

s.no	Variables
1	Instant
2	Dteday
3	Season
4	Yr
5	Mnth
6	Holiday
7	Weekday
8	workingday
9	weathersit
10	Temp
11	Atemp
12	Hum
13	windspeed
14	Casual
15	registered

Chapter 2

Methodology

2.1 Pre Processing

Any predictive modeling requires that we look at the data before we start modeling. However, in data mining terms *looking at data* refers to so much more than just looking. Looking at data refers to exploring the data, cleaning the data as well as visualizing the data through graphs and plots. This is often called as Exploratory Data Analysis. To start this process, we will first try and look at all the distributions of the Numeric variables. Most analysis like regression, require the data to be normally distributed.

We have removed the fields instant, dteday (as all the required like month week day already present) casual and registered (because there sum is equal to target variable ie. 'cnt')

2.1.1 Univariate Analysis

In Figure 2.1 and 2.2 we have plotted the probability density functions numeric variables present in the data including target variable cnt..

i. Target variable cnt is normally distributed
ii. Independent variables like 'temp','atemp', is distributed normally.
iii. Other Independent variable 'hum' and windspeed data is slightly skewed

Figure 2.1 Distribution of target variable (CNT)

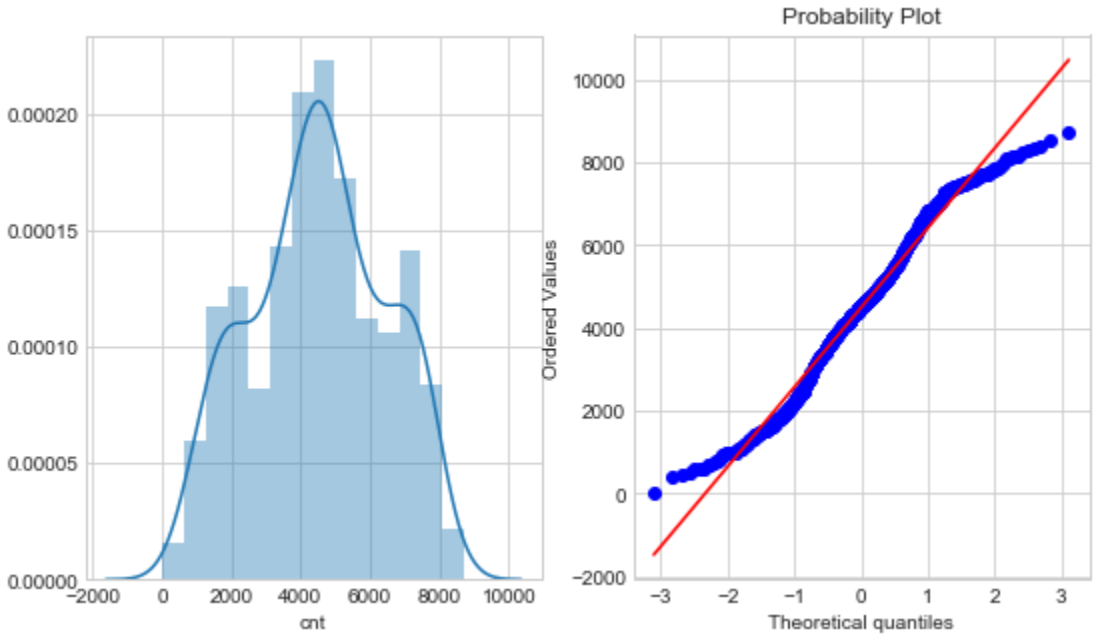


Figure 2.2 showing distribution of numeric variables

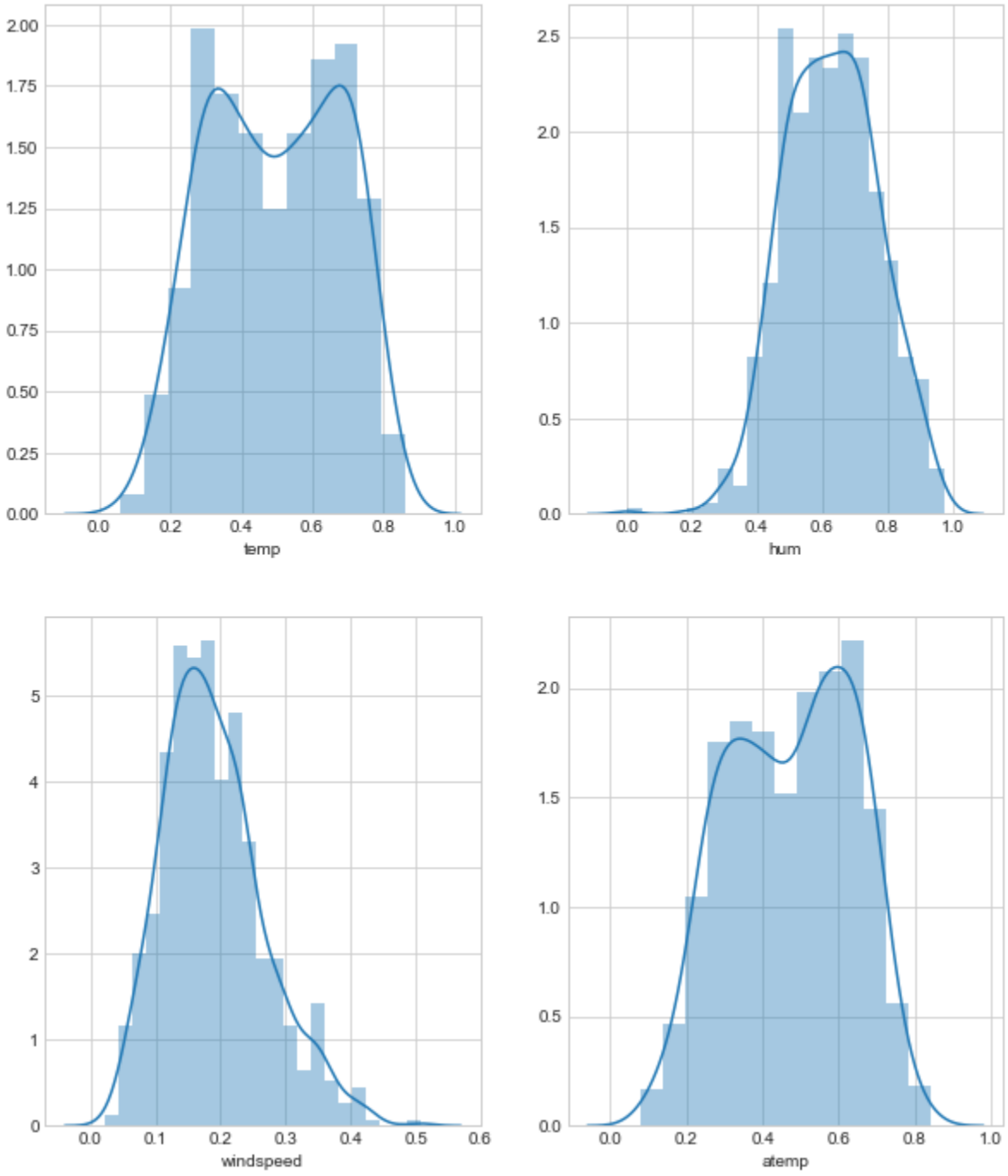
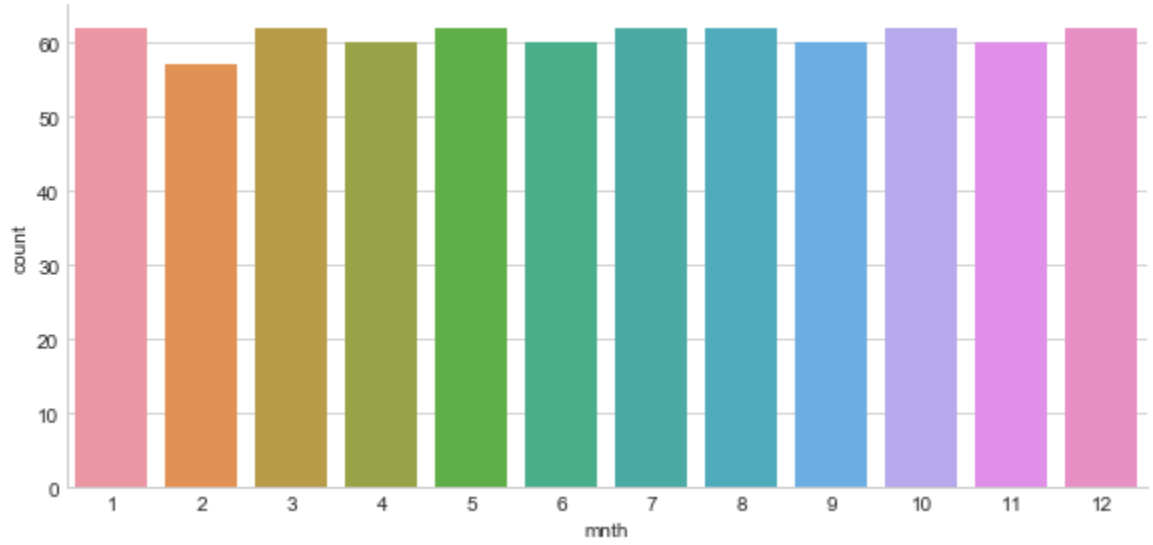
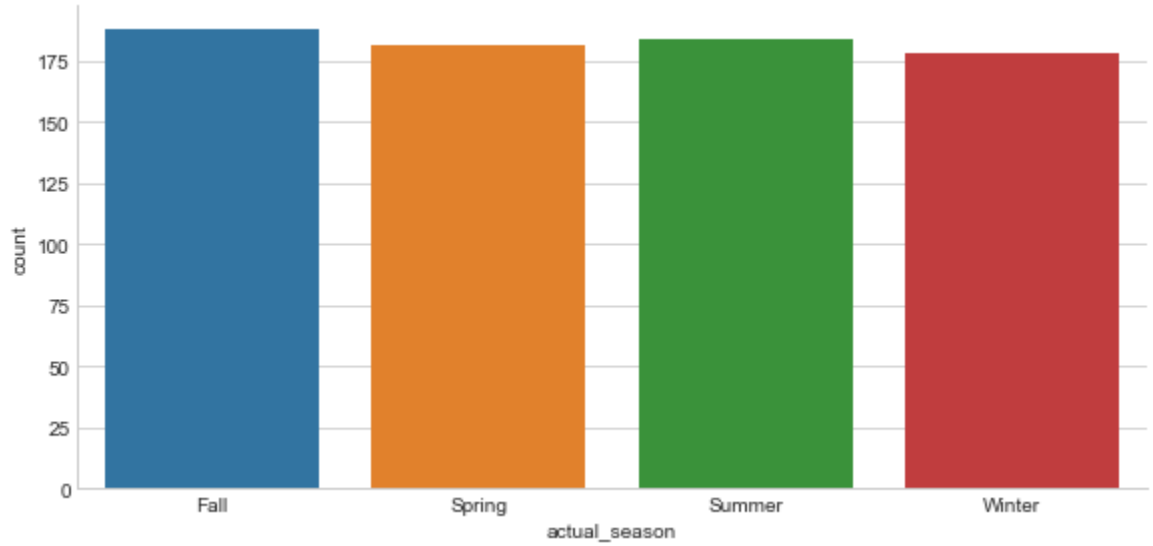
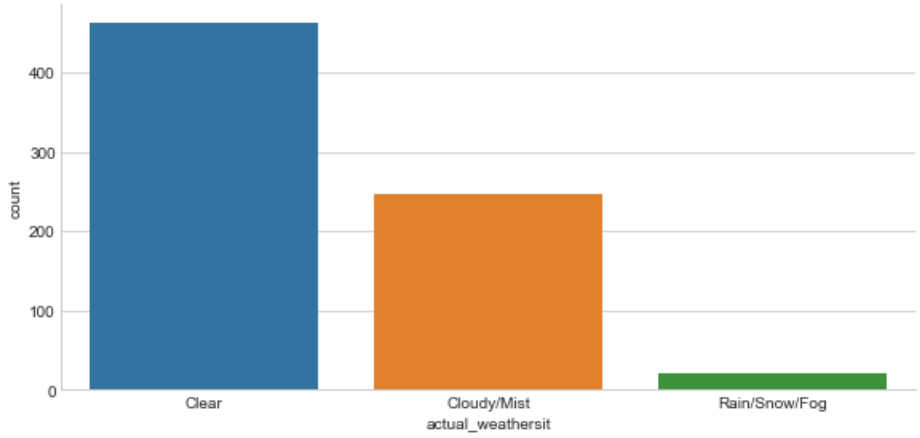
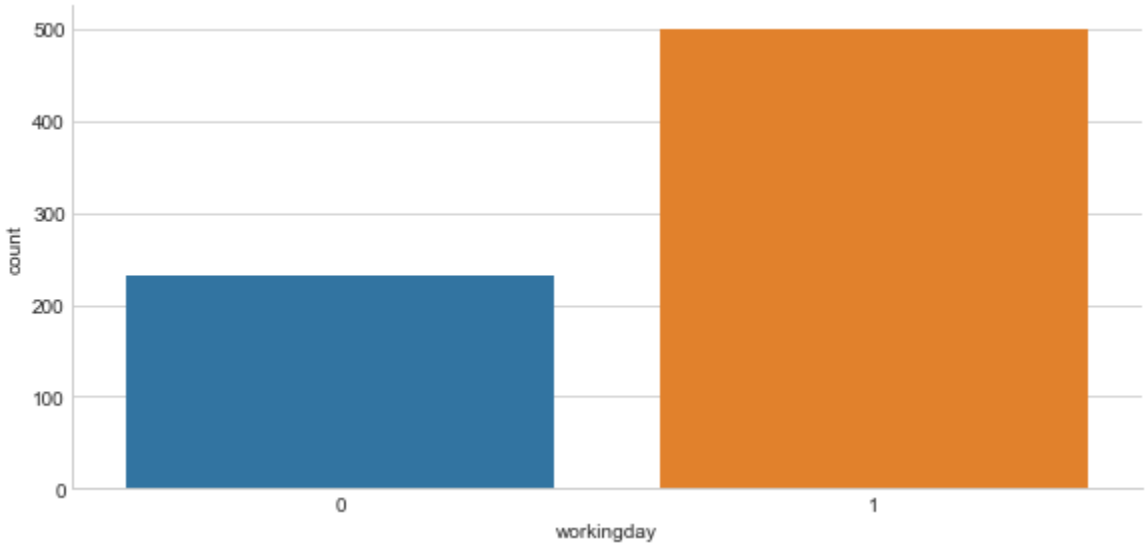


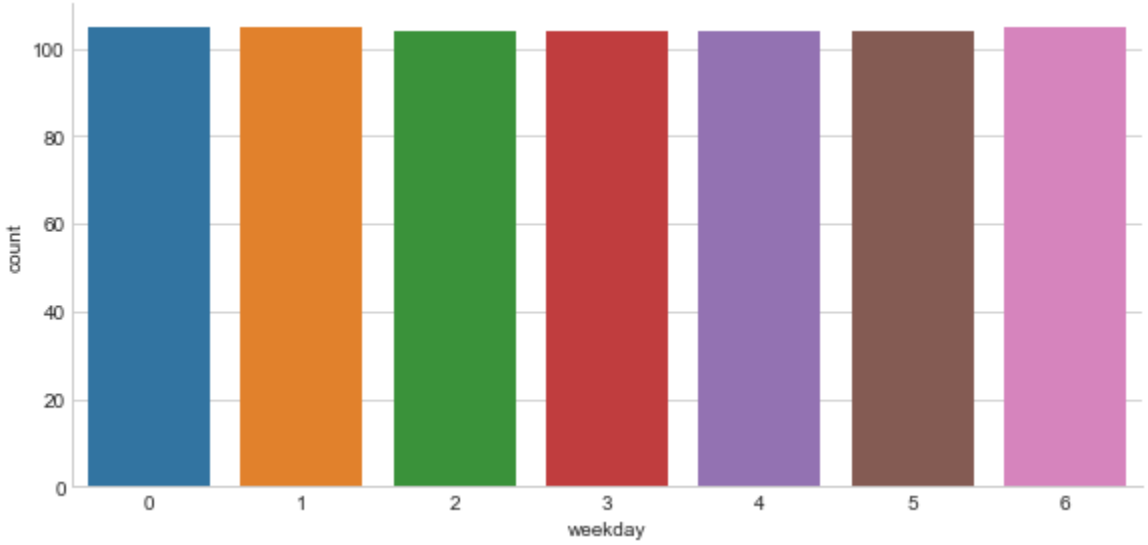
Figure 2.3 showing distribution of categorical variables using factorplot/bar graph

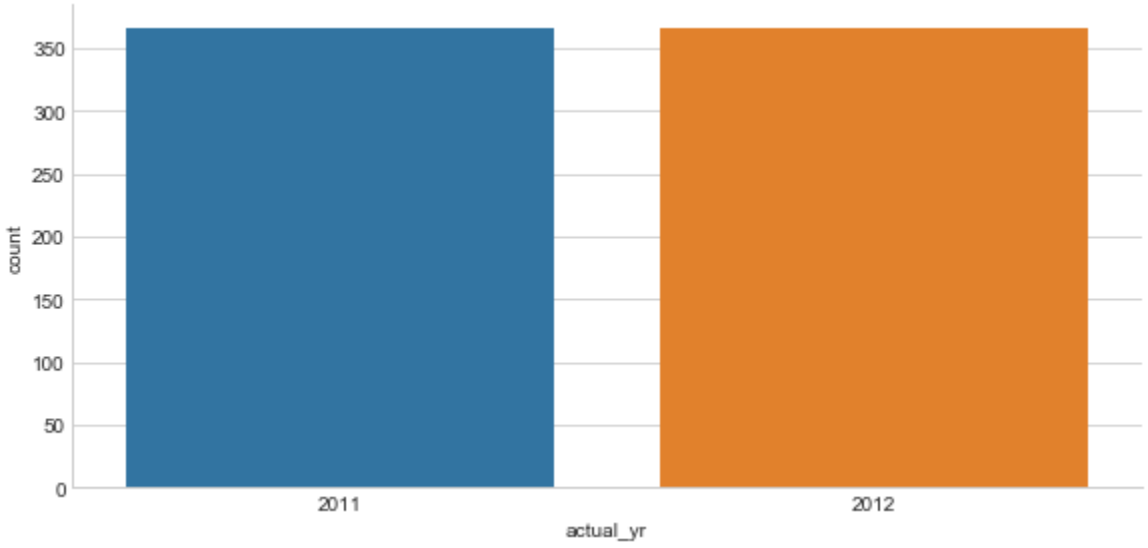


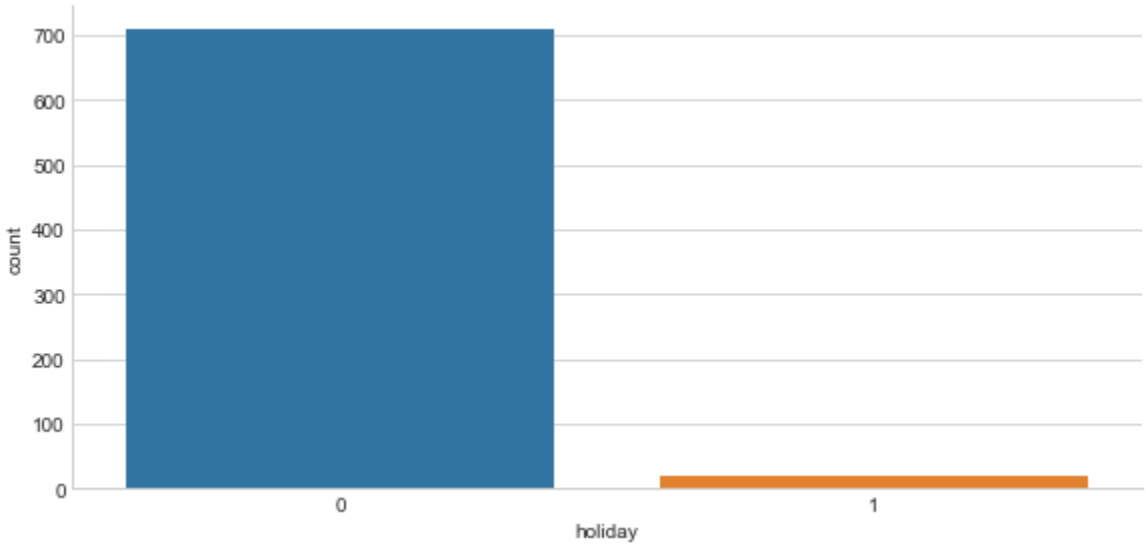





500
400
300
200
100
0
count
0
1
workingday


100
80
60
40
20
0
count
0
1
2
3
4
5
6
weekday


350
300
250
200
150
100
50
0
count
2011
2012
actual_yr



2.1.2 Bivariate Analysis

The below figure shows the relationship between continuous variables and the target variable using scatter plot. It can be observed that there exists a linear positive relationship between the variables temperature and feel temperature with the bike rental count. There also exists a negative linear relationship between the variable's humidity and windspeed with the bike rental count

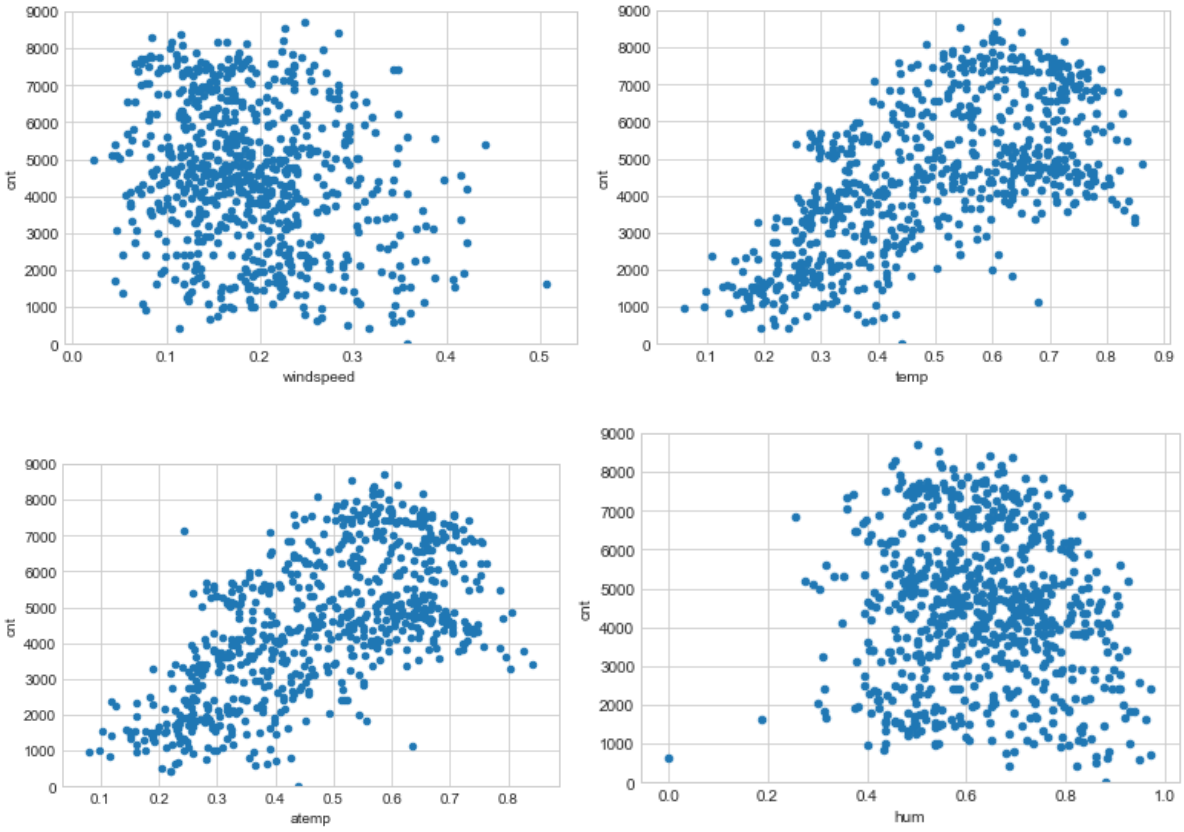


Figure 2.4 Showing relationship between numeric variables and cnt

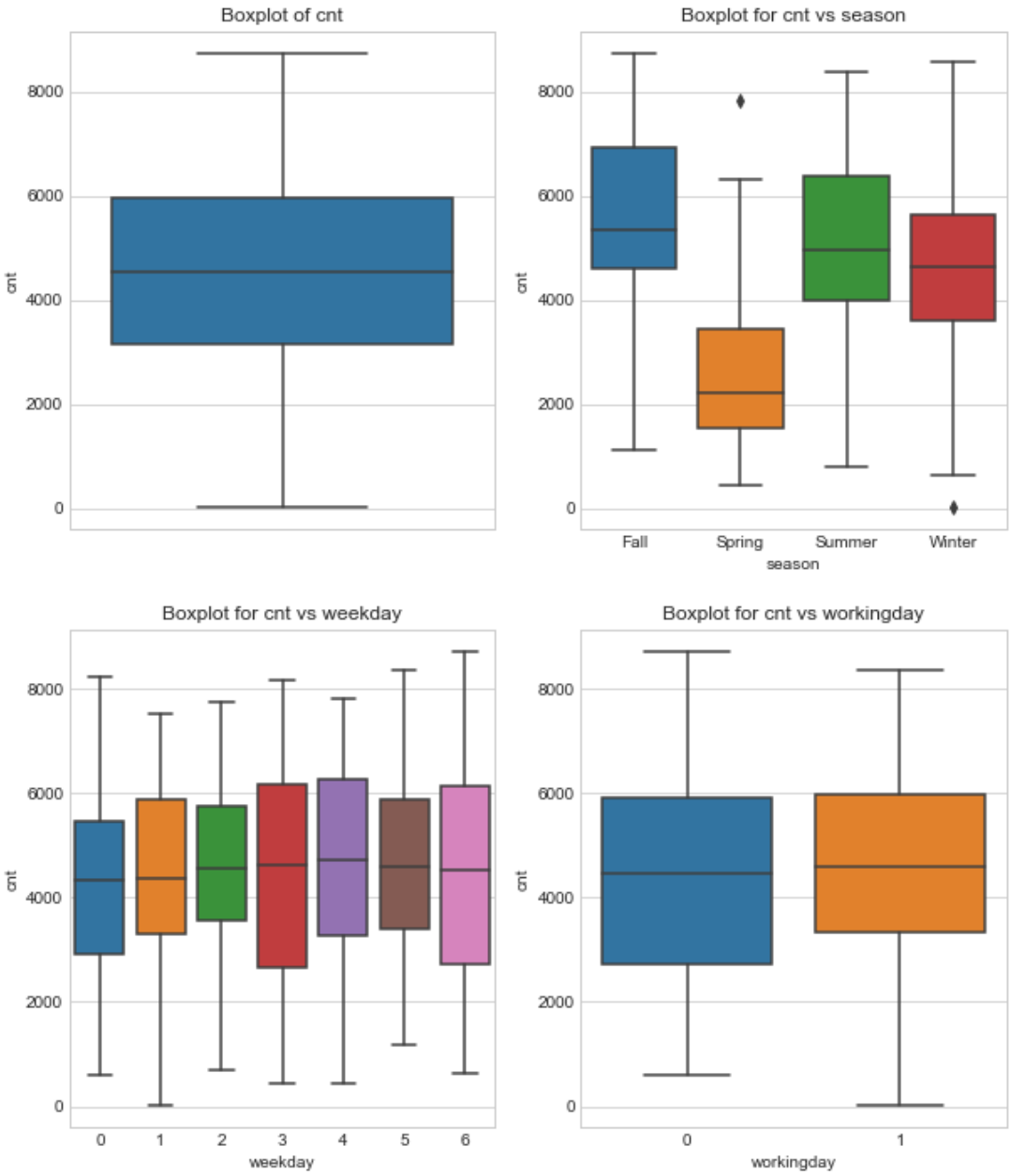


Figure 2.5 Showing relationship between Categoricalvariables and cnt

From Fig2.5,2.6 we can say that,

1. The count is highest for fall season and lowest for spring season. There is no significance difference between count for summer and fall.
2. Bike demand was higher in 2012 as compared with 2011.
3. count is high in the month of august, September and October.It is lowest count is for January ad February.
4. Count is higher on holidays. People prefer to rent bike on holidays.

5. There is not much variation in count on weekdays. Thecount is same for working and non-working days.The range is longer for nonworking days
6. The count is maximum when weather situation is good.

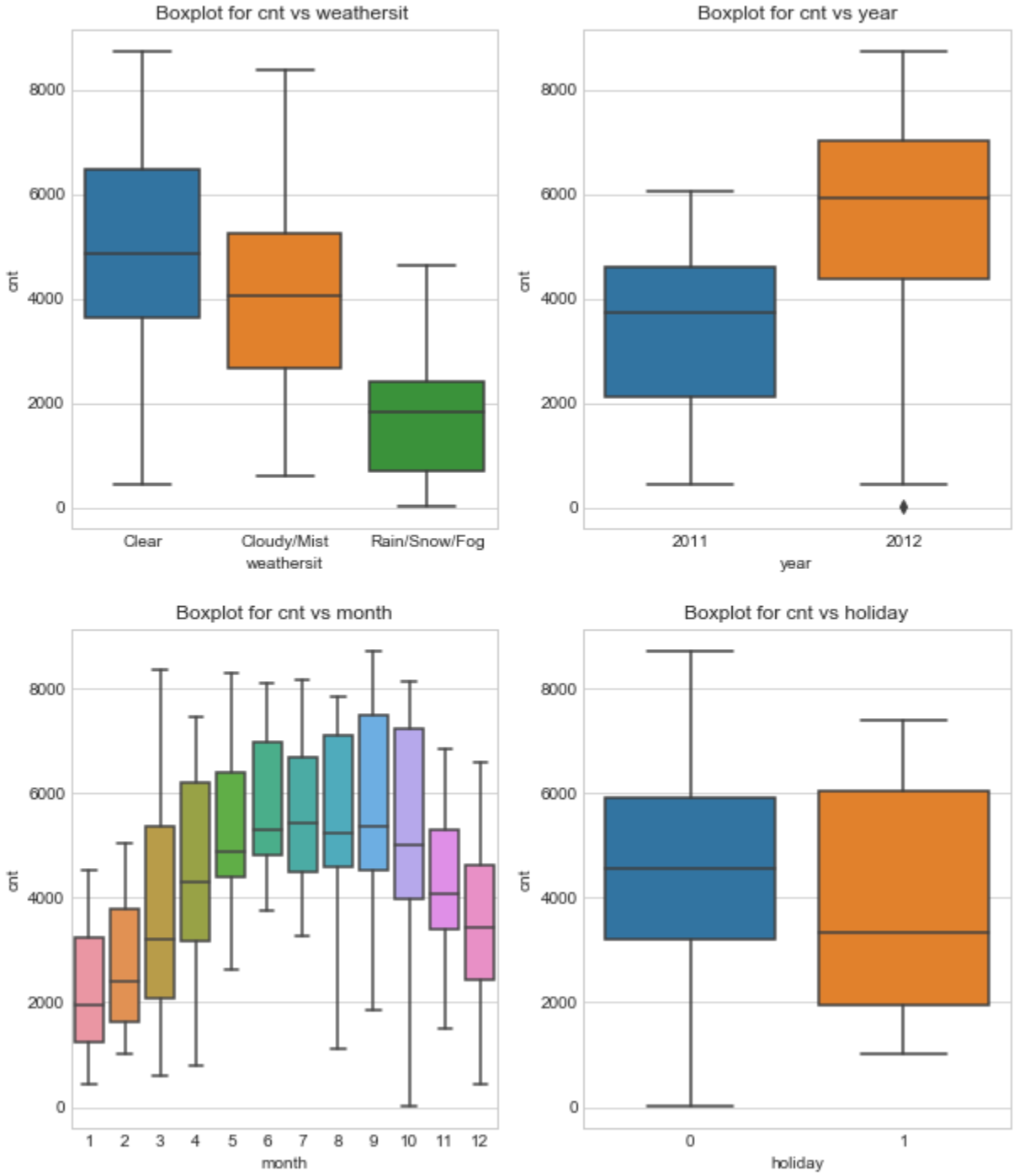


Figure 2.6 Showing relationship between Categoricalvariables and cnt

2.2.1 Missing Value Analysis

Missing values in data is a common phenomenon in real world problems. Knowing how to handle missing values effectively is a required step to reduce bias and to produce powerful models. Below table illustrate no missing value present in the data.

2.1 missing values

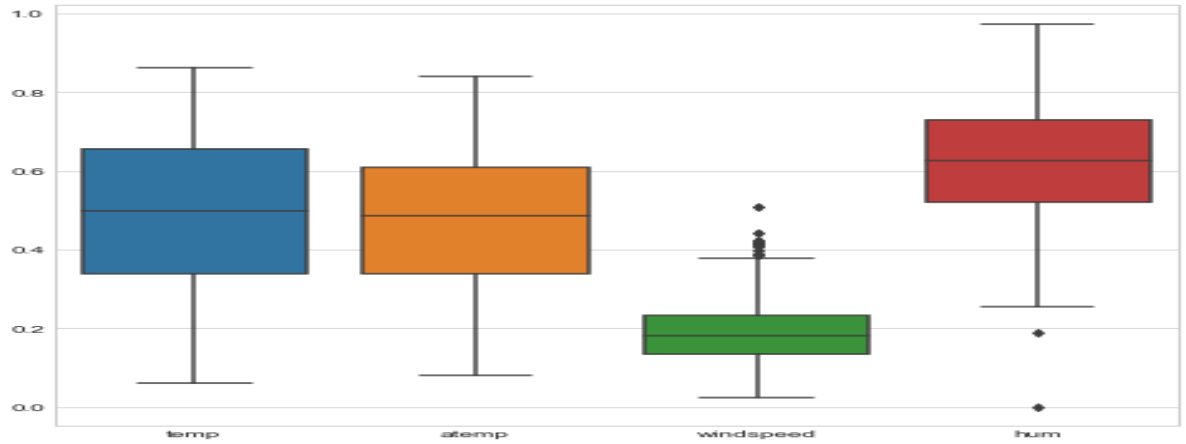
s.no	Variables	missing values
1	season	0
2	yr	0
3	mnth	0
4	holiday	0
5	weekday	0
6	workingday	0
7	weathersit	0
8	temp	0
9	atemp	0
10	hum	0
11	windspeed	0

2.2.2 Outlier Analysis

The Other steps of Preprocessing Technique is Outliers analysis , an outlier is an observation point that is distant from other observations. Outliers in data can distort predictions and affect the accuracy, if you don't detect and handle them appropriately especially in regression models.

Outliers can be removed using the Boxplot stats method, wherein the Inter Quartile Range (IQR) is calculated and the minimum and maximum value are calculated for the variables. Any value ranging outside the minimum and maximum value are discarded.

Below figure illustrates the boxplots for all continuous variables.



the

Fig 2.7 boxplots for all continuous variables

The Figure 2.7 shows that there are outliers in hum and windspeed. Using boxplot method the outliers are removed.

The boxplot of the continuous variable hum and windspeed after removing the outliers is shown in the below figure:

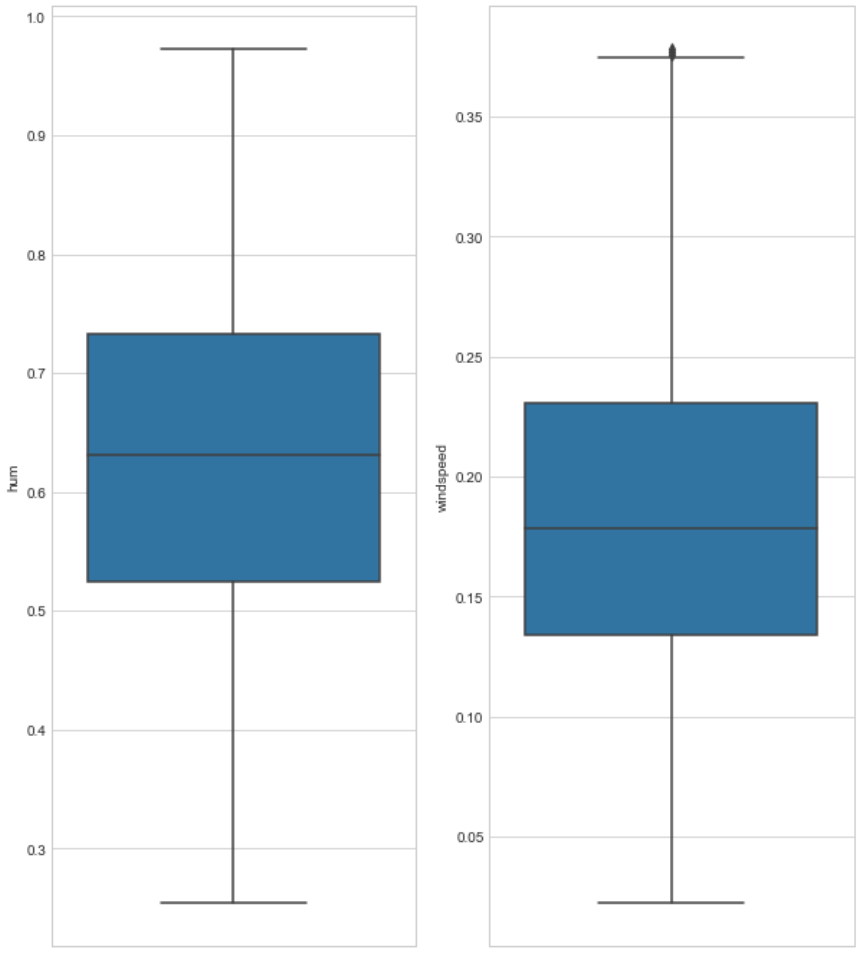


Fig 2.8 boxplots for continuous variables hum and speed after removing outliers

2.2.3 Features Selections

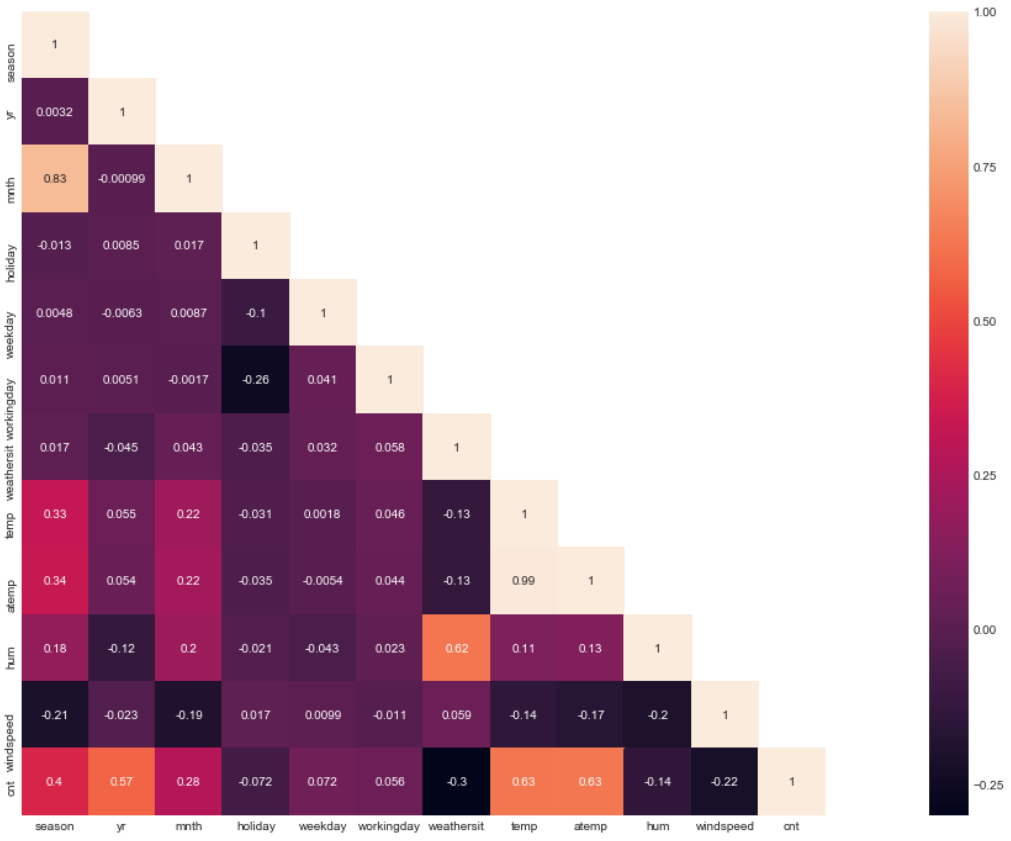
Machine learning works on a simple rule – if you put garbage in, you will only get garbage to come out. By garbage here, I mean noise in data.

This becomes even more important when the number of features are very large. You need not use every feature at your disposal for creating an algorithm. You can assist your algorithm by feeding in only those features that are really important. I have myself witnessed feature subsets giving better results than complete set of feature for the same algorithm or – “Sometimes, less is better!”.

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

i. The relationship between two independent variable should be less and
ii. The relationship between Independent and Target variables should be high.

Below fig 2.6 illustrates that t relationship between all numeric variables using Correlation plot .



Light Color indicates there is strong positive relationship and if darkness is increasing indicates relation between variables are decreasing. Dark Color Red indicates there is strong negative relationship and if darkness is decreasing indicates relationship between variables are decreasing.

The features ‘atemp’ is removed due to high collinearity with temp. There is a poor relation between Independent variables 'holiday','weekday','workingday' and dependent variable 'cnt' , hence removing these variables.

2.2.4 Features Scaling

The word “normalization” is used informally in statistics, and so the term normalized data can have multiple meanings. In most cases, when you normalize data you eliminate the units of measurement for data, enabling you to more easily compare data from different places. In dataset numeric variables like ‘temp’ ,’hum’ and ‘ windspeed’ are in normalization form. Hence no need of Feature scaling

Chapter 3

Modelling

3.1 Model Selection

3.1 Model Selection

The dependent variable in our model is a continuous variable i.e., Count of bike rentals. Hence the models that we choose are Linear Regression, Decision Tree and Random Forest. The error metric chosen for the problem statement is Mean Absolute Error (MAE).

3.1.1 Evaluating Regression Model

We are using two methods to evaluating performance of model

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

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

ii. RMSE : (Root Mean Square Error) is a frequently used measure of the difference between values predicted by a model and the values actually observed from the environment that is being modelled.

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

3.2 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 regression**. In decision analysis, a decision tree can be used to visually and explicitly represent decisions and decision making. As the name goes, it uses a tree-like model of decisions.

Figure 3.2.1 Decision Tree Algorithm

R code:

```r
#Divide the data into train and test
set.seed(12)
train_index = sample(1:nrow(df_day), 0.8 * nrow(df_day))
train = df_day[train_index,]
test = df_day[-train_index,]

#rpart for regression
dt_model = rpart(cnt ~ ., data = train, method = "anova")

#Predict the test cases
dt_predictions = predict(dt_model, test[,-8])

#Create dataframe for actual and predicted values
df = data.frame("actual"=test[,8], "pred"=dt_predictions)
head(df)
print(dt_model)
#  plotting decision tree
par(cex= 0.8)
plot(dt_model)
text(dt_model)
#calculate MAPE
regr.eval(trues = test[,8], preds = dt_predictions, stats = c("mae","mse","rmse","mape"))

#calculate MAPE
MAPE = function(actual, pred){
  print(mean(abs((actual - pred)/actual)) * 100)
}
```

Python Code:

```
In [25]: #********************************** Decision Tree  Regressor ************************************


         dt_model = DecisionTreeRegressor(random_state=123).fit(train.iloc[:,0:7], train.iloc[:,7])

         print(dt_model)

         DecisionTreeRegressor(criterion='mse', max_depth=None, max_features=None,
                    max_leaf_nodes=None, min_impurity_decrease=0.0,
                    min_impurity_split=None, min_samples_leaf=1,
                    min_samples_split=2, min_weight_fraction_leaf=0.0,
                    presort=False, random_state=123, splitter='best')

In [26]: dt_predictions = dt_model.predict(test.iloc[:,0:7])
         print(dt_predictions)

         [4484. 3830. 7282. 7112. 3614. 5572. 6273. 4294. 6569. 4352. 3322. 4120.
          3867. 2493. 5976. 3005. 3805. 1872.  920. 3644. 6153. 4010. 6855. 4098.
          1416. 4966. 3744. 3510. 4127.  959. 4333. 5611. 6966. 4010. 6230. 1472.
          3709. 4459. 4023. 5375. 6169. 6591. 7733. 1891. 4308. 4634. 4401. 2115.
          5046. 5713. 3005. 6889. 6043. 4905. 7509. 6043. 1977. 2914. 5445. 5582.
          5810. 6569. 2689. 7494. 6118. 5026. 4839. 3310. 3520. 7697. 3785. 2689.
          2660. 1815. 6569. 3239. 5260. 4864. 1530. 4717. 5713. 3542. 4579. 4308.
          4326. 3613. 7591. 6043. 5515. 4120. 3423. 3510. 5375. 7691.  822. 6230.
          4098. 7767. 5992. 4648. 2496. 2423. 4318. 4905. 1526. 4459. 4326. 3910.
          5810. 5936. 4840. 1985. 4333. 6230. 2294. 3663. 4258. 3717. 2177. 3574.
          3867. 7415. 4758. 5058. 4182. 7498. 2252. 4010. 5918. 7384. 3669. 2425.
          6460. 6889. 4539. 6227. 6133. 5342. 5582. 6639. 7359. 2425. 4352. 5180.]

In [27]: df_results = pd.DataFrame({'actual': test.iloc[:,7], 'pred': dt_predictions})
         df_results.head()
```

3.2.1 Evaluation of Decision Tree Model

Figure 3.2.3 Evaluation of Decision Tree using MAPE and RMSE

R code Results:

```
##########################################DECISION TREE#########################################
#MAPE: 19.42%
#MAE: 782
#RMSE: 953.8
#Accuracy: 81.58%
```

Python Results:

```
In [28]: #Calculate MAPE
         def MAPE(y_true, y_pred):
             MAE = np.mean(np.abs((y_true - y_pred)))
             mape = np.mean(np.abs((y_true - y_pred) / y_true))
             print("MAPE is: ",mape)
             print("MAE is: ",MAE)
             return mape

         def RMSE(y_test,y_predict):
             mse = np.mean((y_test-y_predict)**2)
             print("Mean Square : ",mse)
             rmse=np.sqrt(mse)
             print("Root Mean Square : ",rmse)
             return rmse

         MAPE(test.iloc[:,7], dt_predictions)
         RMSE(test.iloc[:,7], dt_predictions)

         MAPE is:  0.19246933238644948
         MAE is:  764.8472222222222
         Mean Square :  1012321.5277777778
         Root Mean Square :  1006.1419024063046

Out[28]: 1006.1419024063046
```

In Figure 3.2.3, Model Accuracy is 81.80% for python and for R it is 81.58.

3.3 Random Forest

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

Random forest functions in below way

i. Draws a bootstrap sample from training data.
ii. For each sample grow a decision tree and at each node of the tree
    a. Ramdomly draws a subset of mtry variable and p total of features that are available
    b. Picks the best variable and best split from the subset of mtry variable
    c. Continues until the tree is fully grown.

It was implemented in both R and python. After training with default setting, hyperparameter tuning was used for increase performance.

Figure 3.3.1 Random Forest Implementation

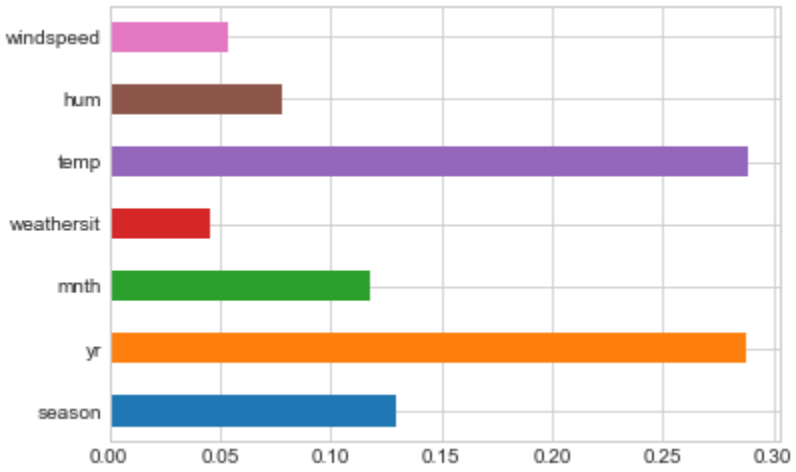
In Python:

```
In [272]: ##########################################Random Forest#########################################################################
          = RandomForestRegressor(random_state=12345)
          random.seed(12)

          electing best max_depth, maximum features, split criterion and number of trees
          am_dist = {'max_depth': [2,4,6,8,10],
                     'bootstrap': [True, False],
                     'max_features': ['auto', 'sqrt', 'log2',None],
                     "n_estimators" : [100 ,200 ,300 ,400 ,500]
                    }
          randomForest = RandomizedSearchCV(rf, cv = 10,
                             param_distributions = param_dist,
                             n_iter = 10)

          randomForest.fit(train.iloc[:,0:7], train.iloc[:,7])
          nt('Best Parameters using random search: \n',
             cv_randomForest.best_params_)

          Best Parameters using random search:
           {'n_estimators': 300, 'max_features': 'log2', 'max_depth': 8, 'bootstrap': False}

In [273]: # setting parameters

          # Set best parameters given by random search # Set be
          rf.set_params(max_features = 'log2',
                        max_depth =8,
                        n_estimators = 300,
                       bootstrap = 'False'
                         )

          rf.fit(train.iloc[:,0:7], train.iloc[:,7])

          # Use the forest's predict method on the test data
          dt_rfPredictions = rf.predict(test.iloc[:,0:7])
```

In R:

```
#Train the data using random forest
rf_model = randomForest(cnt~., data = train, ntree = 500 ,nodesize =8,importance=TRUE,mtry=4)

#Predict the test cases
rf_predictions = predict(rf_model, test[,-8])

#Create dataframe for actual and predicted values
df = cbind(df,rf_predictions)
head(df)

#Calculate MAPE
regr.eval(trues = test[,8], preds = rf_predictions, stats = c("mae","mse","rmse","mape"))
MAPE(test[,8], rf_predictions)


varimp <- importance(rf_model)


# sort variable

sort_var <- names(sort(varimp[,1],decreasing =T))

sort_var
# draw varimp plot

varImpPlot(rf_model,type = 2)
```

Variable importance using random forest in python



3.3.1 Evaluation of Random Forest

Figure 3.2.2 Random Forest Evaluation

In Python:

```
In [274]: MAPE(test.iloc[:,7], dt_rfPredictions)
          RMSE(test.iloc[:,7], dt_rfPredictions)
          rf_errors = abs(dt_rfPredictions - test.iloc[:,7])

          MAPE is:  0.13203573754389147
          MAE is:  487.0455405920468
          Mean Square :  441579.59408990276
          Root Mean Square :  664.5145552129786
```

In R:

```
########################################RANDOM FOREST#########################################
#MAPE: 15.68%
#MAE: 437
#RMSE: 724
#Accuracy: 84.32%
```

Fig 3.2.2 shows Random Forest model performs dramatically better than Decision tree on both training and test data and well also improve the Accuracy and decrease the RMSE of the model which is quite impressive.

Here the Accuracy in the Python is about 87.80, where in R is 84.32%

3.4 Linear Regression

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

Figure 3.4.2 Multiple Linear Regression Model

Python Implementation

```
In [270]: #develop Linear Regression model using sm.ols

          dt_lnr_model = sm.OLS(train_lr.iloc[:,0], train_lr.iloc[:,1:]).fit()

          #Summary of model
          print(dt_lnr_model.summary())

          #predict the  model

          dt_predict_LR = dt_lnr_model.predict(test_lr.iloc[:,1:])

          print(dt_predict_LR)
```

R code Implementation:

```
#Train the data using linear regression
lr_model = lm(formula = cnt~., data = train)

#Check the summary of the model
summary(lr_model)

#Predict the test cases
lr_predictions = predict(lr_model, test[,-8])

#Create dataframe for actual and predicted values
df = cbind(df,lr_predictions)
head(df)

#Calculate MAPE
regr.eval(trues = test[,8], preds = lr_predictions, stats = c("mae","mse","rmse","mape"))
MAPE(test[,8], lr_predictions)
```

3.4.2 Evaluation of Linear regression Model

Figure 3.4.3 Evaluation of Regression Model

Python Results:

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                    cnt   R-squared:                       0.847
Model:                            OLS   Adj. R-squared:                  0.841
Method:                 Least Squares   F-statistic:                     152.2
Date:                Fri, 19 Jul 2019   Prob (F-statistic):          1.49e-209
Time:                        11:28:40   Log-Likelihood:                -4616.4
No. Observations:                 573   AIC:                             9275.
Df Residuals:                     552   BIC:                             9366.
Df Model:                          20
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
```

```
In [271]: MAPE(test_lr.iloc[:,0], dt_predict_LR)
          RMSE(test_lr.iloc[:,0], dt_predict_LR)

          MAPE is:  0.16879845030544732
          MAE is:  625.407468333216
          Mean Square :  706297.5937208855
          Root Mean Square :  840.4151317776742
```

R Results:

```
#########################################LINEAR REGRESSION#####################################
#MAPE: 17.12%
#RMSE: 844
#Accuracy: 82.88%
#MAE: 647
#Adjusted R squared: 0.8483
#F-statistic: 183.2
```

Here the Accuracy in the Python is about 84.13%, where in R is 82.82%. It performs better than decision tree but not that the random forest method. According to the Adj R squared value, about 86% of data is using our linear regression model

3.5 Model Selection

Result and Performance measure RMSE (root mean square error) and MAE (mean absolute error) were used as error metric and measuring model performance.

Performance Measure

R implementation :
For measuring rmse, Metric package was used. For measuring MAE, a function was written.
The values for both the metric for linear regression and random forest are as follow.

Error Metric / Algorithm	RMSE	MAE	Accuracy
Linear Regression	844	647	82.86
Random Forest	724	437	84.32
Decision Tree	953	782	81.58

As from the table we can see that random forest performing better than others on all the error metric.

Python implementation
In python, both the error metric was calculated using python functions. No pre-built package or modules were used.
The values for both metric are given below.

Error Metric / Algorithm	RMSE	MAE	Accuracy
Linear Regression	840	625	83.2
Random Forest	664	487	86.8
Decision Tree	1006	764	80.76

As we can see random forest performing better than other.

Result:

From the error metric we can see that random forest is performing better than linear regression and Decision tree in both implementations. Random Forest is the better model for our analysis. Hence Random Forest is chosen as the model for prediction of bike rental count.

Appendix A – R Code

```
#Clean the environment
rm(list = ls())

#Set working directory
setwd("C:/Users/sai/Documents/New folder R")
getwd()

#Load the librarires
libraries = c("plyr","dplyr", "ggplot2","rpart","DMwR","randomForest","usdm","corrgram","DataCombine")
lapply(X = libraries,require, character.only = TRUE)
rm(libraries)



#Read the csv file
df_day = read.csv(file = "day.csv", header = T, sep = ",", na.strings = c(" ", "", "NA"))



##########################################EXPLORE THE
DATA##########################################
#First few rows
head(df_day)

#Dimensions of data
dim(df_day)

#Column names
names(df_day)

#Structure of variables
str(df_day)
```

```
##Droping variables which are not requried.


#instant - index number

#dteday- all the requried like month week day all ready present

#droping casual and registered because there sum is equal to target variable ie. 'cnt'


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


############################################################ Univariate
Analysis#####################################################################

#Create columns

df_day$actual_temp <- df_day$temp*39

df_day$actual_feel_temp <- df_day$atemp*50

df_day$actual_windspeed <- df_day$windspeed*67

df_day$actual_hum = df_day$hum * 100


df_day$actual_season = factor(x = df_day$season, levels = c(1,2,3,4), labels =
c("Spring","Summer","Fall","Winter"))

df_day$actual_yr = factor(x = df_day$yr, levels = c(0,1), labels = c("2011","2012"))

df_day$actual_holiday = factor(x = df_day$holiday, levels = c(0,1), labels = c("Working day","Holiday"))

df_day$actual_weathersit = factor(x = df_day$weathersit, levels = c(1,2,3,4),

                                  labels = c("Clear","Cloudy/Mist","Rain/Snow/Fog","Heavy Rain/Snow/Fog"))


df_day$weathersit = as.factor(df_day$weathersit)

df_day$season = as.factor(df_day$season)

df_day$mnth = as.factor(df_day$mnth)

df_day$weekday = as.factor(as.character(df_day$weekday))

df_day$workingday = as.factor(as.character(df_day$workingday))

df_day$yr = as.factor(df_day$yr)

df_day$holiday = as.factor(df_day$holiday)
```

```
# function to create univariate distribution of numeric  variables
univariate_numeric <- function(num_x) {


  ggplot(df_day)+
    geom_histogram(aes(x=num_x,y=..density..),
                   fill= "grey")+
    geom_density(aes(x=num_x,y=..density..))

}


# analyze the distribution of  target variable 'cnt'and other numeric independent variables
univariate_numeric(df_day$cnt)
univariate_numeric(df_day$temp)
univariate_numeric(df_day$atemp)
univariate_numeric(df_day$hum)
univariate_numeric(df_day$windspeed)

#Check the distribution of categorical Data using bar graph
bar1 = ggplot(data = df_day, aes(x = actual_season)) + geom_bar() + ggtitle("Count of Season")
bar2 = ggplot(data = df_day, aes(x = actual_weathersit)) + geom_bar() + ggtitle("Count of Weather")
bar3 = ggplot(data = df_day, aes(x = actual_holiday)) + geom_bar() + ggtitle("Count of Holiday")
bar4 = ggplot(data = df_day, aes(x = workingday)) + geom_bar() + ggtitle("Count of Working day")
# ## Plotting plots together
gridExtra::grid.arrange(bar1,bar2,bar3,bar4,ncol=2)

############################################ Bivariate  Relationship
############################################################

#relation between Numerical Variables and target variable 'cnt' using scatter plot
```

```
scat1 = ggplot(data = df_day, aes(x =actual_temp, y = cnt)) + ggtitle("Distribution of Temperature") +
geom_point() + xlab("Temperature") + ylab("Bike COunt")

scat2 = ggplot(data = df_day, aes(x =actual_hum, y = cnt)) + ggtitle("Distribution of Humidity") +
geom_point(color="red") + xlab("Humidity") + ylab("Bike COunt")

scat3 = ggplot(data = df_day, aes(x =actual_feel_temp, y = cnt)) + ggtitle("Distribution of Feel
Temperature") + geom_point() + xlab("Feel Temperature") + ylab("Bike COunt")

scat4 = ggplot(data = df_day, aes(x =actual_windspeed, y = cnt)) + ggtitle("Distribution of Windspeed") +
geom_point(color="red") + xlab("Windspeed") + ylab("Bike COunt")

gridExtra::grid.arrange(scat1,scat2,scat3,scat4,ncol=2)


#Box plot for Categorical Variables with 'CNT' to know the releation


cnames =
colnames(df_day[,c("actual_season","mnth","actual_yr","actual_holiday","actual_weathersit","weekday","
workingday")])

for (i in 1:length(cnames))

{

  assign(paste0("gn",i), ggplot(aes_string(x = cnames[i],y = 'cnt'), data = df_day)+

           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(x=cnames[i])+

           ggtitle(paste("Box plot for",cnames[i])))

}


gridExtra::grid.arrange(gn1,ncol=1)

gridExtra::grid.arrange(gn1,gn3,gn2,gn4,gn5,gn6,gn7,ncol=4)




##########################################MISSING VALUE
Analysis##########################################

missing_values = sapply(df_day, function(x){sum(is.na(x))})
```

```
#no Missing Values found in the data



########################################## Outlier Analysis
####################################
#Check for outliers in data using boxplot
cnames = colnames(df_day[,c("temp","atemp","windspeed","hum")])
for (i in 1:length(cnames))
{
  assign(paste0("gn",i), ggplot(aes_string(y = cnames[i]), data = df_day)+
           stat_boxplot(geom = "errorbar", width = 0.5) +
           geom_boxplot(outlier.colour="red", fill = "grey" ,outlier.shape=18,
                        outlier.size=1, notch=FALSE) +
           theme(legend.position="bottom")+
           labs(y=cnames[i])+
           ggtitle(paste("Box plot for",cnames[i])))
}
gridExtra::grid.arrange(gn1,gn3,gn2,gn4,ncol=2)

cor(df_day$windspeed,df_day$cnt)
cor(df_day$hum,df_day$cnt)
#Remove outliers in Windspeed
val = df_day$windspeed[df_day$windspeed %in% boxplot.stats(df_day$windspeed)$out]
df_day = df_day[which(!df_day$windspeed %in% val),]

#Remove outliers in hum
val = df_day$hum[df_day$hum %in% boxplot.stats(df_day$hum)$out]
df_day = df_day[which(!df_day$hum %in% val),]

# Boxplot after removing  outliers
cnames = colnames(df_day[,c("windspeed","hum")])
for (i in 1:length(cnames))
```

```
{
  assign(paste0("gn",i), ggplot(aes_string(y = cnames[i]), data = df_day)+
           stat_boxplot(geom = "errorbar", width = 0.5) +
           geom_boxplot(outlier.colour="red", fill = "grey" ,outlier.shape=18,
                        outlier.size=1, notch=FALSE) +
           theme(legend.position="bottom")+
           labs(y=cnames[i])+
           ggtitle(paste("Box plot for",cnames[i])))
}
gridExtra::grid.arrange(gn1,gn2,ncol=2)

cor(df_day$windspeed,df_day$cnt)
cor(df_day$hum,df_day$cnt)

############################################  Feature  Scaling
########################################

# In dataset  numeric  variables like 'temp' , 'atem' ,'hum' and ' windspeed' are in normalization form
#no Need of Feature scaling

################################################ feature  selection
########################################

#Check for multicollinearity using VIF

df_VIF = df_day[,c("temp","atemp","hum","windspeed")]
vifcor(df_VIF)

#Check for collinearity using corelation graph
corrgram(day, order = F, upper.panel=panel.pie, text.panel=panel.txt, main = "Correlation Plot")

#Remove the unwanted variables
```

```
df_day <- subset(df_day, select = -
c(holiday,atemp,weekday,workingday,actual_temp,actual_feel_temp,actual_windspeed,
                                actual_hum,actual_season,actual_yr,actual_holiday,actual_weathersit))


rmExcept(keepers = "df_day")


##########################################DECISION
TREE##########################################

#MAPE: 19.42%

#MAE: 782

#RMSE: 953.8

#Accuracy: 81.58%


#Divide the data into train and test

set.seed(12)

train_index = sample(1:nrow(df_day), 0.8 * nrow(df_day))

train = df_day[train_index,]

test = df_day[-train_index,]


#rpart for regression

dt_model = rpart(cnt ~ ., data = train, method = "anova")


#Predict the test cases

dt_predictions = predict(dt_model, test[,-8])


#Create dataframe for actual and predicted values

df = data.frame("actual"=test[,8], "pred"=dt_predictions)

head(df)

print(dt_model)

#  plotting decision tree

par(cex= 0.8)
```

```
plot(dt_model)
text(dt_model)
#calculate MAPE
regr.eval(trues = test[,8], preds = dt_predictions, stats = c("mae","mse","rmse","mape"))

#calculate MAPE
MAPE = function(actual, pred){
  print(mean(abs((actual - pred)/actual)) * 100)
}

MAPE(test[,8], dt_predictions)



##########################################RANDOM
FOREST##########################################
#MAPE: 15.68%
#MAE: 437
#RMSE: 724
#Accuracy: 84.32%

#Train the data using random forest
rf_model = randomForest(cnt~., data = train, ntree = 500 ,nodesize =8,importance=TRUE)

#Predict the test cases
rf_predictions = predict(rf_model, test[,-8])

#Create dataframe for actual and predicted values
df = cbind(df,rf_predictions)
head(df)

#Calculate MAPE
regr.eval(trues = test[,8], preds = rf_predictions, stats = c("mae","mse","rmse","mape"))
```

```
MAPE(test[,8], rf_predictions)

#variable importance  plot

varimp <- importance(rf_model)
# sort variable
sort_var <- names(sort(varimp[,1],decreasing =T))
sort_var
varImpPlot(rf_model,type = 2)

##########################################LINEAR
REGRESSION#########################################
#MAPE: 17.12%
#RMSE: 844
#Accuracy: 82.88%
#MAE: 647
#Adjusted R squared: 0.8483
#F-statistic: 183.2

#Train the data using linear regression
lr_model = lm(formula = cnt~., data = train)

#Check the summary of the model
summary(lr_model)

#Predict the test cases
lr_predictions = predict(lr_model, test[,-8])

#Create dataframe for actual and predicted values
df = cbind(df,lr_predictions)
head(df)
```

```
#Calculate MAPE
regr.eval(trues = test[,8], preds = lr_predictions, stats = c("mae","mse","rmse","mape"))
MAPE(test[,8], lr_predictions)

# Random forest is performing better than linear regression.

# Model input and output
write.csv(test_set, file = "InputtestR.csv")
write.csv(rf_predictions, file="outputR.csv")
```

APPENDIX B Python Code

```
# coding: utf-8


# In[40]:

import pandas as pd # data processing, CSV file I/O

import os #To Interact with local system directories

import numpy as np # linear algebra

import matplotlib.pyplot as plt # For Plotting the Visualizations/Graphs

import seaborn as sns #For the diffetent kind of vizualizations

from scipy import stats #iFor probability plot

from sklearn.ensemble import RandomForestRegressor # For Random forest algorithm

from sklearn.model_selection import train_test_split,RandomizedSearchCV #For model splitting and fit

from sklearn.tree import DecisionTreeRegressor #Importing Decision Tree Regressor from sklear.tree

import statsmodels.api as sm #for Linear Regression model

get_ipython().run_line_magic('matplotlib', 'inline')


# In[4]:

#Set working directory

os.chdir("C:/Users/poorna.chimirala/Documents/Python")

print(os.getcwd()

# In[5]:

#Read the csv file

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

# In[6]:

#Get first 5 rows

print(df_day.head())

#Get the number of rows and columns
```

```
print(df_day.shape)

#Get the data types of variables
print(df_day.dtypes)

#data  consist of Integers , Float  and Object(categorical) variables



# In[7]:



#Droping variables which are not requried.

#instant - index number
#dteday- all the requried like month week day all ready present
#droping casual and registered because there sum is equal to target variable ie. 'cnt'
df_day = df_day.drop(["instant","dteday","casual","registered"],axis = 1)

df_day.head()



# In[8]:



########################################################## Univariate
Analysis##############################################
# Target variable  analysis

#Check whether target variable is normal or not
```

```
fig,(x1,x2) = plt.subplots(ncols=2)
fig.set_size_inches(9,5)
sns.distplot(df_day['cnt'],ax = x1)
stats.probplot(df_day["cnt"], dist='norm', fit=True,plot = x2)

#descriptive statistics summary
print(df_day['cnt'].describe())
print("Skewness: %f" % df_day['cnt'].skew())



# In[9]:



#Create a new dataframe containing required columns and creating new columns
df = df_day.copy()
df.head()

#Create new columns
df['actual_temp'] = df_day['temp'] * 39
df['actual_feel_temp'] = df_day['atemp'] * 50
df['actual_windspeed'] = df_day['windspeed'] * 67
df['actual_hum'] = df_day['hum'] * 100
df['actual_season'] = df_day['season'].replace([1,2,3,4],["Spring","Summer","Fall","Winter"])
df['actual_yr'] = df_day['yr'].replace([0,1],["2011","2012"])
df['actual_holiday'] = df_day['holiday'].replace([0,1],["Working day","Holiday"])
df['actual_weathersit'] =
df_day['weathersit'].replace([1,2,3,4],["Clear","Cloudy/Mist","Rain/Snow/Fog","Heavy Rain/Snow/Fog"])

#Check the data types new variables
```

```
print(df.dtypes)

#Change the data types
df['weathersit'] = df['weathersit'].astype('category')
df['holiday'] = df['holiday'].astype('category')
df['yr'] = df['yr'].astype('category')
df['season'] = df['season'].astype('category')
df['workingday'] = df['workingday'].astype('category')
df['weekday'] = df['weekday'].astype('category')
df['mnth'] = df['mnth'].astype('category')
df['actual_season'] = df['actual_season'].astype('category')
df['actual_yr'] = df['actual_yr'].astype('category')
df['actual_holiday'] = df['actual_holiday'].astype('category')
df['actual_weathersit'] = df['actual_weathersit'].astype('category')

print(df.dtypes)


# In[10]:


#Check the bar graph of categorical Data using factorplot
sns.set_style("whitegrid")
sns.factorplot(data=df, x='actual_season', kind= 'count',size=4,aspect=2)
sns.factorplot(data=df, x='actual_weathersit', kind= 'count',size=4,aspect=2)
sns.factorplot(data=df, x='workingday', kind= 'count',size=4,aspect=2)
sns.factorplot(data=df, x='weekday', kind= 'count',size=4,aspect=2)
sns.factorplot(data=df, x='mnth', kind= 'count',size=4,aspect=2)
sns.factorplot(data=df, x='actual_yr', kind= 'count',size=4,aspect=2)
```

```
sns.factorplot(data=df, x='holiday', kind= 'count',size=4,aspect=2)
# In[11]:

#Distribution  independent numeric variables
#Check whether  independent numeric variables are normal or not
fig,axes = plt.subplots(nrows=2,ncols=2)
fig.set_size_inches(10,12)
sns.distplot(df_day['temp'],ax=axes[0][0])
sns.distplot(df_day['hum'],ax=axes[0][1])
sns.distplot(df_day['windspeed'],ax=axes[1][0])
sns.distplot(df_day['atemp'],ax=axes[1][1])


# In[12]:


############################################ Bivariate  Relationship
#########################################################


#relation between Numerical Variable 'temp' and target variable 'cnt'


df['temp'].value_counts()


#Now draw scatter plot between 'temp' and 'cnt' variables


var = 'temp'
data = pd.concat([df['cnt'], df[var]], axis=1)
data.plot.scatter(x=var, y='cnt', ylim=(0,9000));
```

```
# It is showing  there is good relation between 'temp' and 'cnt'


# In[13]:


#relation between Numerical Variable 'atemp' and target variable 'cnt'

df['atemp'].value_counts()

#Now draw scatter plot between 'temp' and 'cnt' variables

var = 'atemp'
data = pd.concat([df['cnt'], df[var]], axis=1)
data.plot.scatter(x=var, y='cnt', ylim=(0,9000));

# It is showing  there is good relation between 'atemp' and 'cnt'


# In[14]:


#relation between Numerical Variable 'hum' and target variable 'cnt'

df['hum'].value_counts()

#Now draw scatter plot between 'hum' and 'cnt' variables
```

```
var = 'hum'
data = pd.concat([df['cnt'], df[var]], axis=1)
data.plot.scatter(x=var, y='cnt', ylim=(0,9000));

# It is showing  there is average relation between 'atemp' and 'cnt'


# In[15]:


#relation between Numerical Variable 'windspeed' and target variable 'cnt'

df['windspeed'].value_counts()

#Now draw scatter plot between 'windspeed' and 'cnt' variables

var = 'windspeed'
data = pd.concat([df['cnt'], df[var]], axis=1)
data.plot.scatter(x=var, y='cnt', ylim=(0,9000));

# It is showing  there is nagative relation between 'windspeed' and 'cnt'


# In[16]:


# Box plot for Categorical Variables with 'CNT' to know the releation
fig,axes = plt.subplots(nrows=2,ncols=2)
fig.set_size_inches(10,12)
```

```
sns.boxplot(data=df,y="cnt",orient='v',ax=axes[0][0])
sns.boxplot(data=df,y="cnt",x="actual_season",orient='v',ax=axes[0][1])
sns.boxplot(data=df,y="cnt",x="weekday",orient="v",ax=axes[1][0])
sns.boxplot(data=df,y="cnt",x="workingday",orient="v",ax=axes[1][1])

axes[0][0].set(ylabel='cnt',title = "Boxplot of cnt")
axes[0][1].set(xlabel="season",ylabel="cnt",title="Boxplot for cnt vs season")
axes[1][0].set(xlabel="weekday", ylabel="cnt",title="Boxplot for cnt vs weekday")
axes[1][1].set(xlabel="workingday",ylabel="cnt",title="Boxplot for cnt vs workingday")



# In[17]:



fig,axes = plt.subplots(nrows=2,ncols=2)
fig.set_size_inches(10,12)
sns.boxplot(data=df,y="cnt",x="actual_weathersit",orient='v',ax=axes[0][0])
sns.boxplot(data=df,y="cnt",x="actual_yr",orient='v',ax=axes[0][1])
sns.boxplot(data=df,y="cnt",x="mnth",orient="v",ax=axes[1][0])
sns.boxplot(data=df,y="cnt",x="holiday",orient="v",ax=axes[1][1])

axes[0][0].set(xlabel='weathersit',ylabel="cnt",title = "Boxplot for cnt vs weathersit ")
axes[0][1].set(xlabel="year",ylabel="cnt",title="Boxplot for cnt vs year")
axes[1][0].set(xlabel="month", ylabel="cnt",title="Boxplot for cnt vs month")
axes[1][1].set(xlabel="holiday",ylabel="cnt",title="Boxplot for cnt vs holiday")



# In[18]:
```

```
############################################   Missing  value Analysis
#############################################

total = df_day.isnull().sum().sort_values(ascending=False)

percent = (df_day.isnull().sum()/df_day.isnull().count()).sort_values(ascending=False)

missing_data = pd.concat([total, percent], axis=1, keys=['Total', 'Percent'])

missing_data.head(30)



#no Missing Values found in the data



# In[19]:



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

sns.boxplot(data=df_day[['temp','atemp','windspeed','hum']])

fig=plt.gcf()

fig.set_size_inches(8,8)



# In[20]:



#removing outliers and checking corelation

print(df_day.shape)

print(df_day['hum'].corr(df_day['cnt']))
```

```
print(df_day['windspeed'].corr(df_day['cnt']))

q75, q25 = np.percentile(df_day.loc[:,'hum'], [75 ,25])
iqr = q75 - q25


min = q25 - (iqr*1.5)
max = q75 + (iqr*1.5)



df_day = df_day.drop(df_day[df_day.loc[:,'hum'] < min].index)
df_day = df_day.drop(df_day[df_day.loc[:,'hum'] > max].index)

q75, q25 = np.percentile(df_day.loc[:,'windspeed'], [75 ,25])
iqr = q75 - q25

min = q25 - (iqr*1.5)
max = q75 + (iqr*1.5)

df_day = df_day.drop(df_day[df_day.loc[:,'windspeed'] < min].index)
df_day = df_day.drop(df_day[df_day.loc[:,'windspeed'] > max].index)

fig,(x1,x2) = plt.subplots(ncols=2)
fig.set_size_inches(10,12)
# Boxplot for casual after  aoutlier removal

sns.boxplot(x=df_day['hum'],orient ='v',ax = x1)
sns.boxplot(x=df_day['windspeed'],orient ='v',ax = x2)
```

```
# Correlation between ''casual'hum,'windspeed' and 'cnt'  after  removal of  outliers
print(df_day['hum'].corr(df_day['cnt']))
print(df_day['windspeed'].corr(df_day['cnt']))
print(df_day.shape)


# In[21]:


######################################### Feature  Scaling
#######################################

# In dataset  numeric  variables like ‘temp’ , ‘atem’ ,’hum’ and ‘ windspeed’ are in normalization form
#no Need of Feature scaling


# In[22]:


############################################### feature  selection
#######################################
#Check for collinearity using corelation matrix.
cor_mat= df_day[:].corr()
mask = np.array(cor_mat)
mask[np.tril_indices_from(mask)] = False
fig=plt.gcf()
fig.set_size_inches(30,12)
```

```
sns.heatmap(data=cor_mat,mask=mask,square=True,annot=True,cbar=True)



# In[23]:



#As per above Correlation  graph, there is strong relation between Independent variables'temp' & 'atemp'.

# There is a poor relation between  Independent variables 'holiday','weekday','workingday' and dependent  variable 'cnt'

# so dropping the above variables for best feature selection

df_day_feature_selection = df_day.drop(['atemp','holiday','weekday','workingday'],axis = 1)

print(df_day_feature_selection.shape)

df_day_feature_selection.head()



# In[24]:



#diividing  Test and train data using skilearn train_test_split



train,test = train_test_split(df_day_feature_selection, test_size = 0.2,random_state = 123 )

print(train.head())

print(test.head())
```

```
# In[25]:


#********************************** Decision Tree  Regressor
**********************************


dt_model = DecisionTreeRegressor(random_state=123).fit(train.iloc[:,0:7], train.iloc[:,7])

print(dt_model)


# In[26]:


dt_predictions = dt_model.predict(test.iloc[:,0:7])
print(dt_predictions)


# In[27]:


df_results = pd.DataFrame({'actual': test.iloc[:,7], 'pred': dt_predictions})
df_results.head()


# In[32]:
```

```
#Calculate MAPE
def MAPE(y_true, y_pred):
    MAE = np.mean(np.abs((y_true - y_pred)))
    mape = np.mean(np.abs((y_true - y_pred) / y_true))
    print("MAPE is: ",mape)
    print("MAE is: ",MAE)
    return mape

def RMSE(y_test,y_predict):
    mse = np.mean((y_test-y_predict)**2)
    print("Mean Square : ",mse)
    rmse=np.sqrt(mse)
    print("Root Mean Square : ",rmse)
    return rmse

MAPE(test.iloc[:,7], dt_predictions)
RMSE(test.iloc[:,7], dt_predictions)
#MAPE is:  0.19246933238644948
#MAE is:  764.8472222222222
#Mean Square :  1012321.5277777778
#Root Mean Square :  1006.1419024063046


# In[33]:
```

```
############################################## Linear Regression
############################


#Create continuous data. Save target variable first
train_lr = train[['cnt','temp','hum','windspeed']]
test_lr = test[['cnt','temp','hum','windspeed']]

cat_names = ["season", "yr", "mnth", "weathersit"]


for i in cat_names:
    temp1 = pd.get_dummies(train[i], prefix = i)
    temp2 = pd.get_dummies(test[i], prefix = i)
    train_lr = train_lr.join(temp1)
    test_lr = test_lr.join(temp2)

train_lr.head()


# In[34]:


#develop Linear Regression model using sm.ols

dt_lnr_model = sm.OLS(train_lr.iloc[:,0], train_lr.iloc[:,1:]).fit()

#Summary of model
```

```
print(dt_lnr_model.summary())

#predict the  model

dt_predict_LR = dt_lnr_model.predict(test_lr.iloc[:,1:])

print(dt_predict_LR)



# In[35]:



MAPE(test_lr.iloc[:,0], dt_predict_LR)

RMSE(test_lr.iloc[:,0], dt_predict_LR)

#MAPE is:  0.16879845030544732

#MAE is:  625.407468333216

#Mean Square :  706297.5937208855

#Root Mean Square :  840.4151317776742



# In[41]:



##################################################Random
Forest#############################################################################

rf = RandomForestRegressor(random_state=12345)

np.random.seed(12)

# selecting best max_depth, maximum features, split criterion and number of trees
```

```
param_dist = {'max_depth': [2,4,6,8,10],
              'bootstrap': [True, False],
              'max_features': ['auto', 'sqrt', 'log2',None],
              "n_estimators" : [100 ,200 ,300 ,400 ,500]
             }
cv_randomForest = RandomizedSearchCV(rf, cv = 10,
                    param_distributions = param_dist,
                    n_iter = 10)

cv_randomForest.fit(train.iloc[:,0:7], train.iloc[:,7])
print('Best Parameters using random search: \n',
      cv_randomForest.best_params_)


# In[42]:


# setting parameters

# Set best parameters given by random search # Set be
rf.set_params(max_features = 'log2',
              max_depth =8,
              n_estimators = 300,
             bootstrap = 'False'
              )

rf.fit(train.iloc[:,0:7], train.iloc[:,7])

# Use the forest's predict method on the test data
```

```
dt_rfPredictions = rf.predict(test.iloc[:,0:7])

print(dt_rfPredictions)


# In[43]:


MAPE(test.iloc[:,7], dt_rfPredictions)
RMSE(test.iloc[:,7], dt_rfPredictions)
rf_errors = abs(dt_rfPredictions - test.iloc[:,7])
#MAPE is:  0.13203573754389147
#MAE is:  487.0455405920468
#Mean Square :  441579.59408990276
#Root Mean Square :  664.5145552129786


# In[275]:


#VariableImportanceInRandomForest
feature_importance =  pd.Series(rf.feature_importances_, index=train.iloc[:,0:7].columns)
feature_importance.plot(kind='barh')

# In[44]:

#Example of output with a sample input.
#model input and output
pd.DataFrame(test).to_csv('InputtestdataPyhon.csv', index = False)
```

```
pd.DataFrame(dt_rfPredictions, columns=['predictions']).to_csv('outputRandomForestPy.csv')
```