PROJECT

BIKE RENTING

Subhadeep chowdhury

CONTENTS:-

1. INTRODUCTION:-

1.1 PROBLEM STATEMENT :-

The objective of this Case is to Predict bike rental count on daily based on the environmental and seasonal settings. The details of data attributes in the dataset are as follows -

i. instant: Record index
ii. dteday: Date season:
iii. Season (1:springer, 2:summer, 3:fall, 4:winter)
iv. yr: Year (0: 2011, 1:2012)
v. mnth: Month (1 to 12)
vi. holiday: weather day is holiday or not (extracted fromHoliday Schedule)
vii. weekday: Day of the week workingday: If day is neither weekend nor holiday is 1, otherwise is 0.
viii. weathersit: (extracted fromFreemeteo)
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
ix. 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)
x. atemp: Normalized feeling temperature in Celsius. The values are derived via (t-t_min)/(t_maxt_min), t_min=-16, t_max=+50 (only in hourly scale)
xi. hum: Normalized humidity. The values are divided to 100 (max)
xii. windspeed: Normalized wind speed. The values are divided to 67 (max)
xiii. casual: count of casual users
xiv. registered: count of registered users
xv. cnt: count of total rental bikes including both casual and registered

1.2 DATA:-

Our task is to build regression model, where we can predict the bike rental count on daily basis.
Given below is a sample of the data set that we are using to predict.

	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

The data set consist of 731 observations with 16 variables.

2. METHODOLOGY :-

2.1 PRE PROCESSING :-

Any predictive modelling requires that we look at the data before we start modelling. 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 visualising 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 variables.

2.1.1. Missing Value Analysis:-

```
bike_data=pd.read_csv("/Users/subhadeep/Downloads/day.csv")
```

```
bike_data.isnull().sum()
# we can see that there are no missing Value in the given data.
```

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

First of all we have loaded the dataset under bike_data data frame.

Then we have checked missing values in the data.

There are no missing values in the dataset.

```
bike_data['dteday'] =  pd.to_datetime(bike_data['dteday'], format='%Y-%m-%d')
```

```
bike_data['day'] = bike_data['dteday'].apply(lambda r:r.day)
```

We have parsed the ‘dteday’ column and extracted the day only, as month and year is already been present in the data.

Next we are dropping 'dteday' and 'instant' column from the data as these 2 columns will not add any value to the model. Along with that we are dropping 'casual' and 'registered' column from the data as the summation of these 2 columns are there in 'cnt' column.

```
bike_data=bike_data.drop(['casual','registered'],axis=1)
```

```
bike_data=bike_data.drop(['dteday','instant'],axis=1)
```

2.1.2 Data Visualisation :-

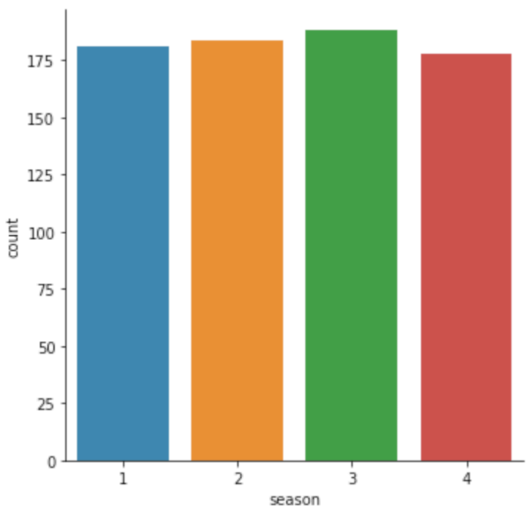
First of all we will se some distribution of categorical variables.

```
bike_data.season.value_counts()
```

```
3    188
2    184
1    181
4    178
Name: season, dtype: int64
```

```
sns.factorplot(x='season',data=bike_data,kind='count',size=5)
# here season 1= spring, 2= summer, 3= fall, 4= winter
```

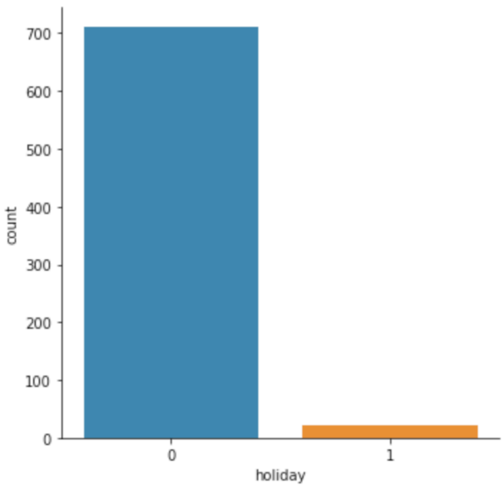
```
<seaborn.axisgrid.FacetGrid at 0x1104a5f98>
```



We can see that bike has been rented highest in 'fall' season.

```python
# Holiday:-
sns.factorplot(x='holiday',data=bike_data,kind='count',size=5)
# holiday as 1 and non holiday as 0. majority of the data is for non holiday
```

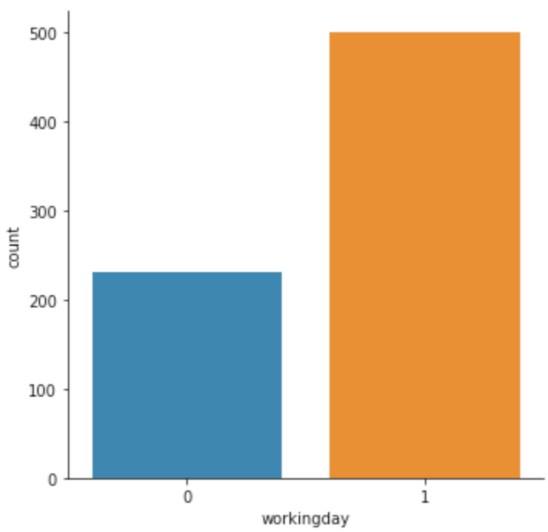
<seaborn.axisgrid.FacetGrid at 0x1105a0e48>



Here we can see that the bike has been rented heavily during working days.

```python
# Working day :-
sns.factorplot(x='workingday',data=bike_data,kind='count',size=5)
# weekend is 0 and for non weekend is 1
# cleary it s visible that majority of the data is for working day
```

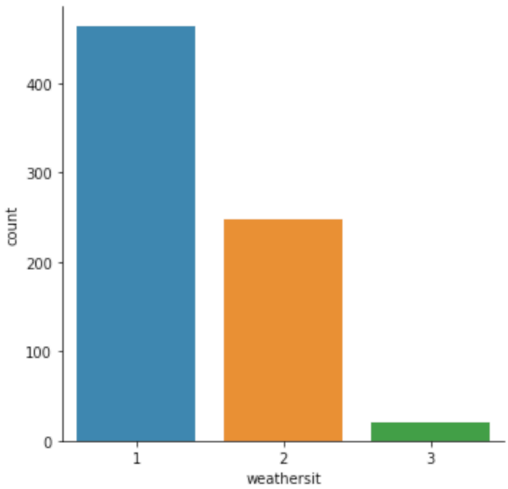
<seaborn.axisgrid.FacetGrid at 0x1105b6780>



We can see from the above graph , that bike was rented highly for non weekend day.

```
# Weathersit
# 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
sns.factorplot(x='weathersit',data=bike_data,kind='count',size=5)
```

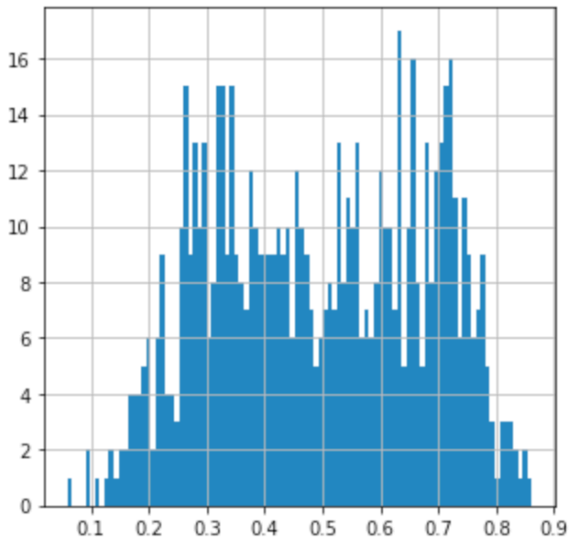
```
<seaborn.axisgrid.FacetGrid at 0x11057a908>
```



Bike was rented majorly when there was a clear or few clouds in it.

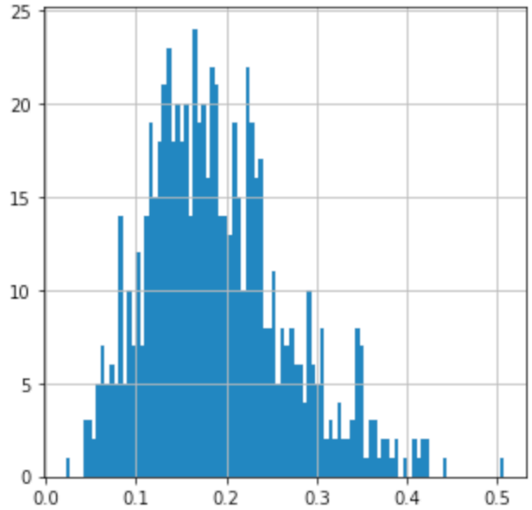
Next we will see some distribution of continuous variables.

```
%matplotlib inline
bike_data.temp.hist(figsize=(5,5),bins=100)
plt.show()
```



We can see that variable ‘temp’ is not normally distributed.

```
%matplotlib inline
bike_data.windspeed.hist(figsize=(5,5),bins=100)
plt.show()
```



Variable ‘windspeed’ is also not normally distributed.

2.1.3 Feature Selection :-

We have drawn a heatmap of correlation of all the training variables . We can see from the below correlation plot that there are correlation between tea and temp variable

```
## set the width and height of the plot
# subplot will decide the height and width of the plot
f, ax = plt.subplots(figsize=(8,6))

# generating correlation matrix
bike_corr = bike_data.loc[:,pnames]
corr=bike_corr.corr()

## mask will crate individual blocks in correlation matrix, np.zeroes will create square blocks in whole paine
##cmap. diverse_palette will set the colours
## plot using seaborn libraray
sns.heatmap(corr, mask=np.zeros_like(corr,dtype=np.bool),cmap=sns.diverging_palette(220,10,as_cmap=True),
            square= True,ax=ax)
# from the heatmap we can see that temp and atemp is highly correlated. so we will drop atemp
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1a140f5978>
```

```
bike_data=bike_data.drop(['atemp'],axis=1)
```

We have dropped variable atemp from the data because of multicollinearity.

We have run RandomForestRegressor with n_estimator taking as 100. Finding the feature importance form the regressor we can see that 'temp' is the most important variable followed by 'yr' and 'season'. We can see the from the below plot of the feature importance of the variables.

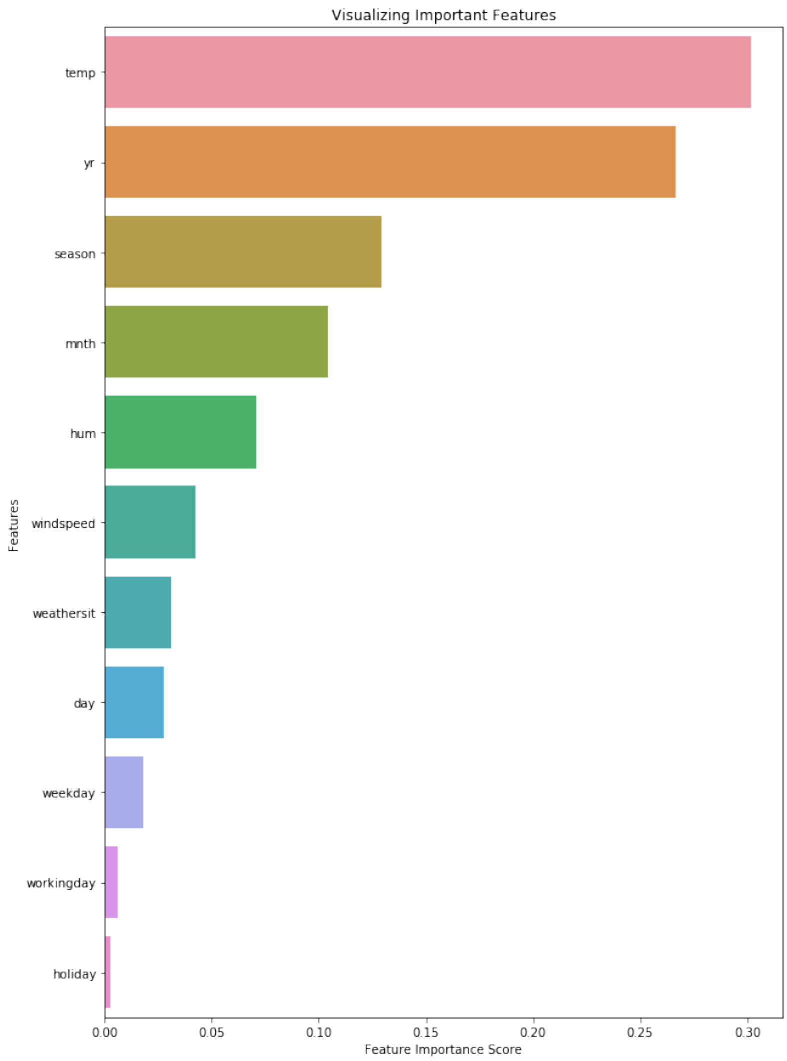
```
# feature importance
feature_imp = pd.Series(bike_rf.feature_importances_,index=x.columns).sort_values(ascending=False)
feature_imp
# we can see that temp variable is the most important variable as it is contributing 30.46%
```

```
temp          0.301520
yr            0.266396
season        0.128980
mnth          0.104003
hum           0.070794
windspeed     0.042547
weathersit    0.031228
day           0.027700
weekday       0.017866
workingday    0.006039
holiday       0.002926
dtype: float64
```

```
%matplotlib inline
# Creating a bar plot
f, ax = plt.subplots(figsize=(10,15))
sns.barplot(x=feature_imp, y=feature_imp.index)
# Add labels to the graph
plt.xlabel('Feature Importance Score')
plt.ylabel('Features')
plt.title("Visualizing Important Features")
#plt.legend()
plt.show()
```

We can also see the feature importance plot visualisation from the below graph.

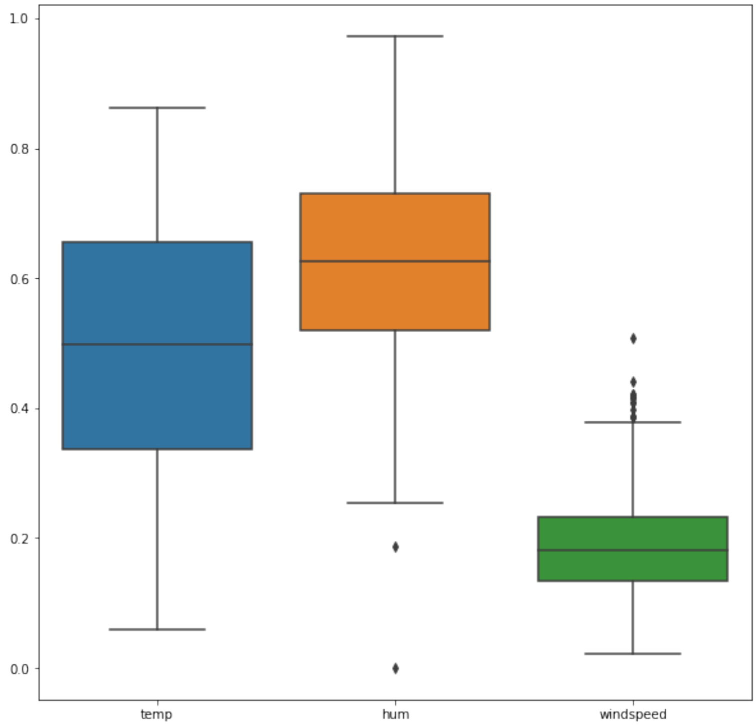
'temp' variable has highest importance of 30% where as holiday has importance of just 0.2%.



2.1.4 Outlier Analysis :-

We have seen the from the distribution of the variable that some the variables are skewed.The skew in these distributions can be most likely explained by the presence of outliers and extreme values in the data.

```
# just to visualize.
sns.boxplot(data=bike_data[['temp','hum','windspeed']])
fig=plt.gcf()
fig.set_size_inches(10,10)
# we can see that there are no outliers present in temp variable.
```



From the above outlier graph we can see that ‘temp’ variables has no outliers. The outliers for ‘winspeed’ is just hovering around the whiskers. 2 outliers are present in ‘hum’ variable. After extracting the IQR range for ‘hum’ variable we can see that one outlier for which the value recorded as 0 in ‘hum’ variable is present. So we are deleting the entire row records for data recorded for the ‘hum’ variable as 0.the rest of the data we kept in the data as it is.

```
bike_data.drop(bike_data[bike_data['hum'] == 0.000000].index, inplace = True)
```

2.1.5. Feature Scaling :-

As we have seen in the data distribution section that all the variables are not normally distributed. So we are using normalisation , just scale down the range of the data from 0 to 1.

So the above table shows the data after normalisation.

```
for i in col_names:
    print(i)
    bike_data[i]= (bike_data[i] - np.min(bike_data[i]))/(np.max(bike_data[i]) - np.min(bike_data[i]))
```

Where col names consists of ‘temp’ , ‘hum’ and ‘windspeed’ variable.

2.2 MODELING :-

2.2.1 Model Selection :-

We will be building different types of regression models, as the project is based on regression.

We will be using different regression models like linear regression, random forest, KNN . We will be using different metrics like RMSE,MAPE or MAE to choose the best model.

2.2.2 Linear Regression :-

```
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import KFold,train_test_split
```

```
x_new=bike_data.drop(['cnt'],axis=1)
y_new=bike_data['cnt']
```

```
# split the data, 75% into tarin and 25% into test data
x_train,x_test,y_train,y_test=train_test_split(x_new,y_new,test_size=0.25,random_state=2)
```

```
print (x_train.shape,y_train.shape)
print (x_test.shape,y_test.shape)
```

```
(547, 11) (547,)
(183, 11) (183,)
```

First of all we are splitting the data into train and test data with 75% as train data and the rest 25% as test data where 'cnt' is the dependent variable.

We build a linear regression based on these data. After fitting the training data we have predicted on testing data.

```
rmse=np.sqrt(mean_squared_error(predict_ln,y_test))
print(rmse)
```

```
869.3162345064748
```

```
mae=mean_absolute_error(predict_ln,y_test)
print(mae)
```

```
659.8206328581629
```

```
# calculate
def MAPE(y_true,y_pred):
    mape=np.mean(np.abs((y_true-y_pred)/y_true))*100
    return mape
```

```
MAPE(y_test,predict_ln)
```

```
17.441670489849336
```

Here we can see that MAPE is 17.44% i.e the model is 82.56% accurate.

Next we are trying with linear regression model with stats model linear regression.

```
model.summary()
```

OLS Regression Results

Dep. Variable:	cnt	**R-squared:**	0.964
Model:	OLS	**Adj. R-squared:**	0.963
Method:	Least Squares	**F-statistic:**	1302.
Date:	Thu, 15 Aug 2019	**Prob (F-statistic):**	0.00
Time:	09:01:28	**Log-Likelihood:**	-4511.7
No. Observations:	547	**AIC:**	9045.
Df Residuals:	536	**BIC:**	9093.
Df Model:	11		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
season	606.0035	62.568	9.686	0.000	483.096	728.911
yr	2121.1242	79.011	26.846	0.000	1965.914	2276.334
mnth	-36.4130	19.480	-1.869	0.062	-74.679	1.853
holiday	-470.7400	238.748	-1.972	0.049	-939.737	-1.743
weekday	88.2888	19.560	4.514	0.000	49.865	126.713
workingday	288.9704	87.157	3.316	0.001	117.758	460.182
weathersit	-564.1636	103.524	-5.450	0.000	-767.526	-360.801
temp	4530.0639	189.496	23.906	0.000	4157.818	4902.310
hum	39.1949	299.271	0.131	0.896	-548.694	627.083
windspeed	-126.3333	229.303	-0.551	0.582	-576.776	324.109
day	-1.7387	4.379	-0.397	0.691	-10.341	6.863

Omnibus:	96.267	**Durbin-Watson:**	1.740
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	218.194
Skew:	-0.932	**Prob(JB):**	4.17e-48
Kurtosis:	5.470	**Cond. No.**	160.

From the above summary we can interpret that the dependent variable is cnt.

No of observations is 547 and df residuals is no of observations - independent variable

Degrees of freedom is 11 which is no of independent variable.

Coefficient of temp is 4530.0639 that means 1 unit increase in temp 4530 unit will change in int variable.

We can take out some important variables based on p value. Like if pvalue< 0.5 we can say that this variable has significance in the model.

Ex-variables like season, temp, yr, weathersit are significant variables where variables like math, day, hum is not significant as their values are less than 0.5.

2.2.3 RandomForest :-

We have used a cross validation hyper parameter tuning n random forest. Its a Bagging Algorithm.

The technique of cross validation (CV) is best explained by example using the most common method, K-Fold CV. When we approach a machine learning problem, we make sure to split our data into a training and a testing set. In K-Fold CV, we further split our training set into K number of subsets, called folds. We then iteratively fit the model K times, each time training the data on K-1 of the folds and evaluating on the Kth fold (called the validation data).
For hyperparameter tuning, we perform many iterations of the entire K-Fold CV process, each time using different model settings. We then compare all of the models, select the best one, train it on the full training set, and then evaluate on the testing set.
Using Scikit-Learn's RandomizeeSearchCV method, we can define a grid of hyperparameter ranges, and randomly sample from the grid, performing Fold CV with each combination of values.
We will try adjusting the following set of hyperparameters:

- n_estimators = number of trees in the foreset
- max_features = max number of features considered for splitting a node
- max_depth = max number of levels in each decision tree
- min_samples_split = min number of data points placed in a node before the node is split
- min_samples_leaf = min number of data points allowed in a leaf node
- criterion = rules fro sampling (Gini or Entropy)

To use RandomizedSearchCV, we first need to create a parameter grid to sample from during fitting:

```
# making a param)dist , a dictonary for hyperparameter tuning.
param_dist = {"n_estimators":[20,50,100,200],
              "max_depth": [3,5,6,8],
              "max_features": [2,5,7,8],
              "min_samples_split": [2,5,8,10],
              "min_samples_leaf": [1,2,5,10]
             }

n_iter_search = 20
```

On each iteration, the algorithm will choose a difference combination of the features. Altogether, there are 4*4*4*4*4= 1024
However, the benefit of a random search is that we are not trying every combination, but selecting at random to sample a wide range of values.

The most important arguments in RandomizedSearchCV are n_iter, which controls the number of different combinations to try, and cv which is the number of folds to use for cross validation (we use 20 and 10 respectively).
After running RandomizedSearchCV we found the best combination of parameter for randomforestregressor. We build the model on top of that.

We found RMSE as 694.303 and MAPE as 14.17 . So the model is 85.83% accurate, which much better than the linear regression.

```
feature_imp_rf = pd.Series(model_rf.feature_importances_,index=x_train.columns).sort_values(ascending=False)
feature_imp_rf
```

```
temp          0.376097
yr            0.283647
season        0.160748
hum           0.068981
mnth          0.032330
windspeed     0.032052
day           0.016331
weathersit    0.016169
weekday       0.008351
workingday    0.002950
holiday       0.002345
dtype: float64
```

Form the importance feature we can see that 'temp','yr','season' is contributing around 81% to the model. SO what we have done is that we are building another random forest model and keeping the cutoff from the importance plot as 3%.
So we are dropping variables which are below 3% like 'day', 'weathersit', 'weekday', 'workigday',
'holiday'.
We build a model with that variables and found MAPE as 16.2 which nit good as the model created by all the original variables.

2.2.4 KNN :

We built a model based on KNN. First of all we have done parameter tuning of n_neighbours and weights.

```
params = {'n_neighbors':[3,4,5,6,7,8],
          'weights':['uniform', 'distance'],
          'n_jobs':[-1]}
```

After fitting the with the data with best parameter it found to be the results given below.
RMSE is 1422.16 and MAPE is 34.41. it is very bad compare to random forest model.

2.2.5 XGBOOST:-

XGBoost (Xtreme Gradient Boosting) is an advanced implementation of gradient boosting algorithm.
we are trying building this model with hyperparameter tuning.
Hyperparameter tuning:

1. max_depth [default=6] : Used to control over-fitting as higher depth will allow model to learn relations very specific to a particular sample. Typical value 3-10.
2. Subsample :Denotes the fraction of observations to be randomly samples for each tree. Lower values make the algorithm more conservative and prevents overfitting but too small values might lead to under-fitting.typical value 0.5 to 1
3. Colsample_bytree : Denotes the fraction of columns to be randomly samples for each tree.typical value 0.5 -1.
4. Learning_rate : This determines the impact of each tree on the final outcome. Lower values are generally preferred as they make the model robust to the specific characteristics of tree and thus allowing it to generalize well.
5. n_estimators : The number of sequential trees to be modeled

On each iteration, the algorithm will choose a difference combination of the features. Altogether, there are 3*4*3*3*3= 324 combination

However, the benefit of a random search is that we are not trying every combination, but selecting at random to sample a wide range of values.

The most important arguments in RandomizedSearchCV are n_iter, which controls the number of different combinations to try, and cv which is the number of folds to use for cross validation (we use 20 and 10 respectively).
After running RandomizedSearchCV we found the best combination of parameter for xgbregressor. We build the model on top of that.

```
# hyperparameter tuning set for param_dist
param_dist1 = {"learning_rate":[0.05,0.1,0.2],
               "n_estimators":[100,150,200,250],
               "max_depth":  [3,4,5],
               "subsample":[0.8,1,0.5],
               "colsample_bytree":[0.6,0.7,0.8]
              }
n_iter_search=20
```

```
# hyperparameter tuning for xgbooost
random_search = RandomizedSearchCV(xgb1, param_distributions=param_dist1,
                                   n_iter=n_iter_search,cv=10)
random_search.fit(x_train, y_train)
```

RMSE of this model is 634.77 and MAPE is 10.93 i.e. the model is 89.07% accurate which becomes the highest amongst all the models built.

MODEL BUILDING IN R :-

We have also built algorithm based on the same problem in R. We have imported the Data into R.

Loading the train and test data

bike_data=read.csv("/Users/subhadeep/Downloads/day.csv")

Looking into the data we can see that all the variables are of int type except dteday which is factor type.

We are dropping 'casual' and 'registered' variable as the summation of these 2 varibles are given in 'cnt' column.

bike_data_new= subset(bike_data,select = -c(casual,registered))

We have parsed the dteday column to extract date from the column.

bike_data_new$date=parse_date_time(bike_data_new$dteday,"ymd")
bike_data_new$day=day(bike_data_new$date)

We have also dropped variables instant and dteday.

bike_data_new=subset(bike_data_new,select=-c(instant,dteday,date))

further processing purpose we are keeping the origial data in df
df=bike_data_new

We are adding 4 new columns with different factor names into df just visualise it more.

df$actual_season = factor(x = df$season, levels = c(1,2,3,4), labels = c("Spring","Summer","Fall","Winter"))
df$actual_yr = factor(x = df$yr, levels = c(0,1), labels = c("2011","2012"))
df$actual_holiday = factor(x = df$holiday, levels = c(0,1), labels = c("Working day","Holiday"))
df$actual_weathersit = factor(x = df$weathersit, levels = c(1,2,3,4), labels = c("Clear","Cloudy/Mist","Rain/Snow/Fog","Heavy Rain/Snow/Fog"))

We are looking into the distribution respect dependent variables.

sort(xtabs(formula = cnt~actual_season,df))

actual_season
Spring Winter Summer   Fall
471348 841613 918589 1061129

We can see that highest no of bike rented in fall season.

xtabs(formula = cnt~actual_holiday,df)

actual_holiday
Working day   Holiday
3214244   78435

From the above chart we can see that maximum no of bike was rented during working day.

xtabs(formula = cnt~actual_weathersit,df)

actual_weathersit
Clear   Cloudy/Mist   Rain/Snow/Fog   Heavy Rain/Snow/Fog
2257952   996858   37869   0

We can see from the above table that highest no of bikes was rated during clear weather while no bike was rented during heavy rain/snow weather.

MISSSING VALUE ANAYSIS :-

#missing_values = sapply(bike_data, function(x){sum(is.na(x))})
There are no missing values present in the data.

OUTLIER ANALYSIS:-

We are making different data frame consist of numeric variables only.

cnames = colnames(bike_data_new[,c("temp","atemp","hum","windspeed")])

boxplot.stats(bike_data_new$temp)
$stats

[1] 0.0591304 0.3370835 0.4983330 0.6554165 0.8616670

$n
[1] 731

$conf
[1] 0.4797301 0.5169359

$out
numeric(0)

We can see that there o variables in 'temp' variable.

boxplot.stats(bike_data_new$windspeed)

$stats
[1] 0.0223917 0.1349500 0.1809750 0.2332145 0.3781080

$n
[1] 731

$conf
[1] 0.1752326 0.1867174

$out
[1] 0.417908 0.507463 0.385571 0.388067 0.422275 0.415429 0.409212 0.421642 0.441563
[10] 0.414800 0.386821 0.398008 0.407346

We can see that some outliers are present in the windspeed variable . But we are keeping it same as it is. maximum whiskers range is 0.37 where the maximum data point is 0.5. so we are keeping the data same.

boxplot.stats(bike_data_new$hum)

$stats
[1] 0.2541670 0.5200000 0.6266670 0.7302085 0.9725000

$n
[1] 731

$conf
[1] 0.6143827 0.6389513

$out
[1] 0.187917 0.000000

We are keeping those 2 outliers in the data and proceeding further.

boxplot.stats(bike_data_new$atemp)

No outlier present in the data.

FEATURE SCALING :-

We will be drawing correlation plot amongst numerical variables.

```
# library(corrgram)
# library(usdm)
# num_names=bike_data_new[,c("temp","atemp","hum","windspeed","day")]

# vifcor(num_names)
```

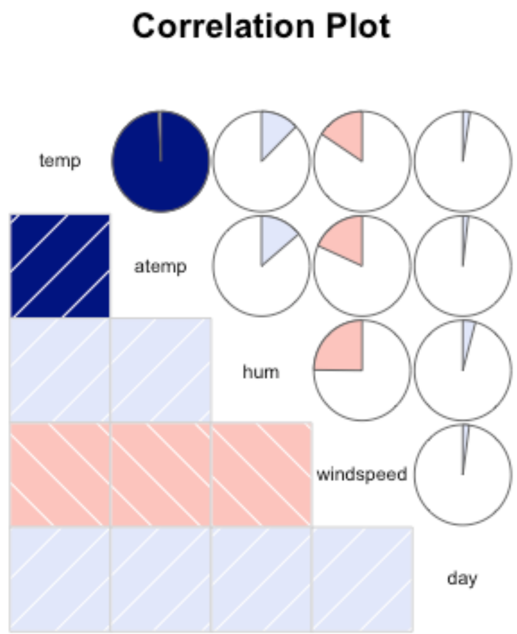
```
1 variables from the 5 input variables have collinearity problem:

atemp

After excluding the collinear variables, the linear correlation coefficients ranges
    between:
min correlation ( day ~ windspeed ):  0.02158805
max correlation ( windspeed ~ hum ):  -0.2484891

---------- VIFs of the remained variables --------
  Variables      VIF
1      temp 1.034859
2       hum 1.077264
3 windspeed 1.085996
4       day 1.003501
```

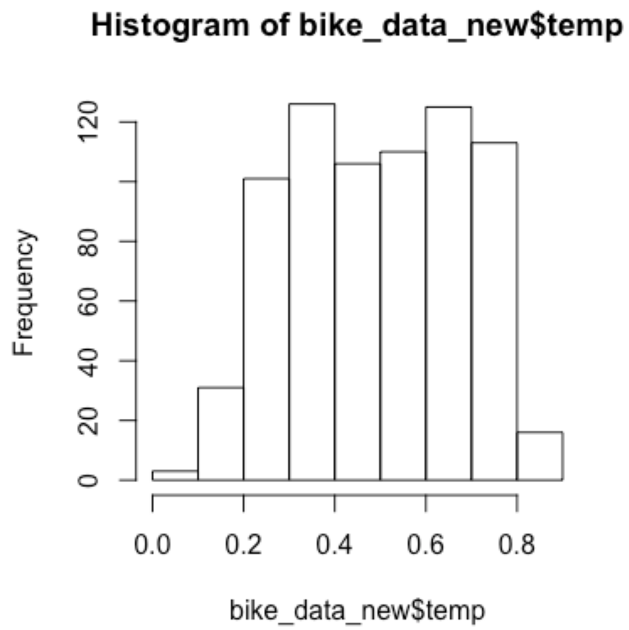
We can see that atemp has multicolinearity problem.
The correlation plot is given below.



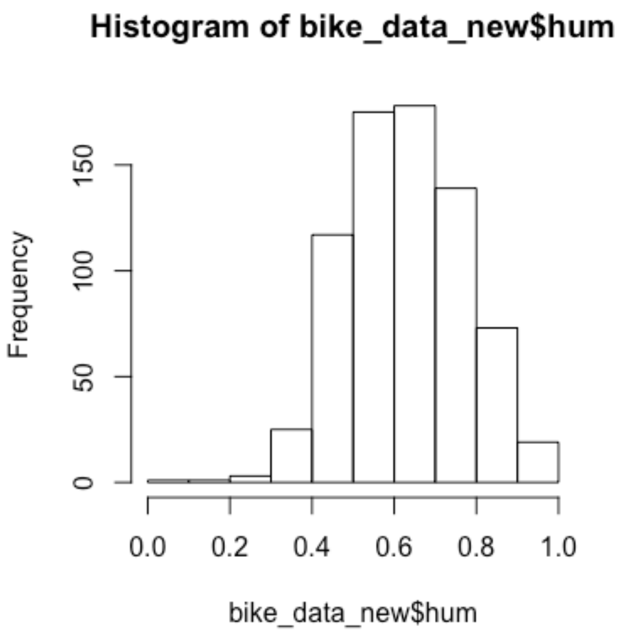
So we are dropping ‘atemp’ variable from the dataset.

NORMALISATION :-

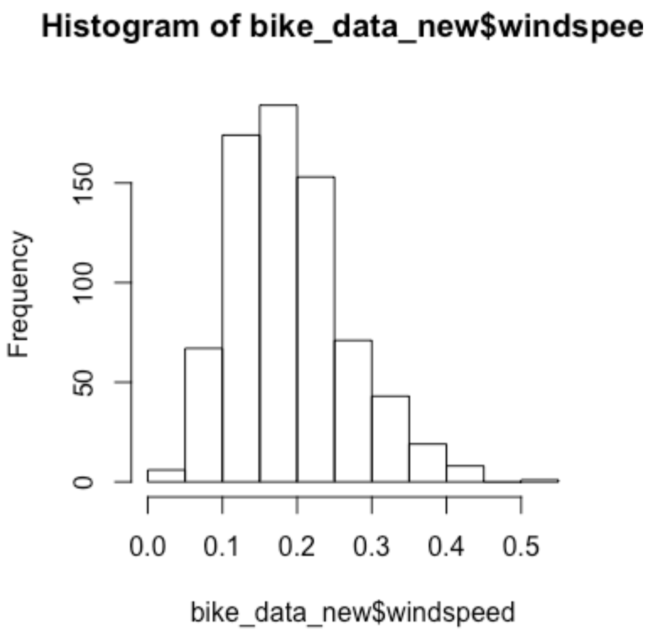
hist(bike_data_new$temp)



hist(bike_data_new$hum)



hist(bike_data_new$windspeed)



We can see that the data is not normally distributed, so we are normalizing the data

```
#    cnames1 = colnames(bike_data_new[,c("temp","hum","windspeed")])
#    for ( i in cnames1) {
```

```
#    print(i)
#    bike_data_new[,i]=(bike_data_new[,i]-min(bike_data_new[,i]))/
#    (max(bike_data_new[,i]-min(bike_data_new[,i])))
#     }
```

MODEL DEVELOPMENT :-

```
# set.seed(2)
# s=sample(1:nrow(bike_data_new),0.75*nrow(bike_data_new))
# train=bike_data_new[s,]
# test=bike_data_new[-s,]
```

We are dividing the dataset into train and test. 75% of the data ate take for training and 25% for testing purpose.

```
# fit=lm(cnt~.,data=train)
```

We have built a linear regression model.

Now we are checking multicollinearity using vif.

```
# library(car)
# vif(fit)
```

```
  season       yr     mnth   holiday   weekday workingday weathersit
 3.672133  1.023034  3.463203  1.097532  1.035953  1.084153  1.695276
     temp      hum windspeed       day
 1.260327  1.902968  1.177868  1.016393
```

We can see that vid score is less than 5 for all variables. So we are to dropping and variables.

```
# step(fit)
```

The stepwise regression (or stepwise selection) consists of iteratively adding and removing predictors, in the predictive model, in order to find the subset of variables in the data set resulting in the best performing model, that is a model that lowers prediction error.

```
Start:  AIC=7463.93
cnt ~ season + yr + mnth + holiday + weekday + workingday + weathersit +
    temp + hum + windspeed + day

             Df Sum of Sq       RSS    AIC
- workingday  1   1090155 432597571 7463.3
<none>                    431507417 7463.9
- day         1   2165079 433672496 7464.7
- hum         1   2925623 434433040 7465.6
- mnth        1   4950952 436458368 7468.2
```

```
- holiday     1   6165659 437673076 7469.7
- weekday     1   7941113 439448530 7471.9
- windspeed   1  22802598 454310015 7490.2
- weathersit  1  38544302 470051719 7508.8
- season      1  49808498 481315914 7521.8
- temp        1 374861964 806369381 7804.6
- yr          1 560744798 992252214 7918.2

Step:  AIC=7463.32
cnt ~ season + yr + mnth + holiday + weekday + weathersit + temp + 
    hum + windspeed + day

             Df Sum of Sq       RSS    AIC
<none>                    432597571 7463.3
- day         1   2132856 434730427 7464.0
- hum         1   3017990 435615561 7465.1
- mnth        1   4965743 437563315 7467.6
- weekday     1   8120158 440717729 7471.5
- holiday     1   8147395 440744967 7471.5
- windspeed   1  23263818 455861390 7490.0
- weathersit  1  38022840 470620412 7507.5
- season      1  49824033 482421605 7521.1
- temp        1 377057016 809654587 7804.8
- yr          1 561146977 993744548 7917.1

Call:
lm(formula = cnt ~ season + yr + mnth + holiday + weekday + weathersit + 
    temp + hum + windspeed + day, data = train)



Coefficients:
(Intercept)       season           yr         mnth      holiday      weekday  
   2053.609      526.114     2046.967      -52.427     -691.870       61.133  
 weathersit         temp          hum    windspeed          day  
   -628.429     4026.607     -682.735    -1397.446       -7.118  
```

We have started with 11 variables and we can see that based on AIC workingday Variable has been removed.

```
> summary(fit)

Call:
lm(formula = cnt ~ ., data = train)

Residuals:
    Min      1Q  Median      3Q     Max 
-3983.8  -484.6    55.4   558.6  2955.7 

Coefficients:
```

```
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  1987.429    263.219   7.550 1.88e-13 ***
season        526.033     66.876   7.866 2.03e-14 ***
yr           2046.291     77.535  26.392  < 2e-16 ***
mnth          -52.349     21.109  -2.480  0.01345 *
holiday      -623.468    225.287  -2.767  0.00585 **
weekday        60.481     19.257   3.141  0.00178 **
workingday     99.461     85.471   1.164  0.24507
weathersit   -633.419     91.543  -6.919 1.30e-11 ***
temp         4018.023    186.204  21.579  < 2e-16 ***
hum          -672.419    352.730  -1.906  0.05714 .
windspeed   -1384.743    260.189  -5.322 1.51e-07 ***
day            -7.172      4.374  -1.640  0.10161
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 897.2 on 536 degrees of freedom
Multiple R-squared:  0.7873,	Adjusted R-squared:  0.7829
F-statistic: 180.4 on 11 and 536 DF,  p-value: < 2.2e-16
```

Adjusted r square is 0.7829 that means that independent variables can explain 78% variance of the target variable.

The estimate are the co efficient of all the independent variables. Based on the p value we can drop working day and day variable as these 2 variables are not significant to the explanation of the target variable.

So we are making another linear regression model after dropping those 2 variables.

```
#fit_final=lm(cnt~season+yr+mnth+holiday+weekday+weathersit+temp+windspeed+
hum,data=train)
#summary(fit_final)

summary(fit_final)

Call:
lm(formula = cnt ~ season + yr + mnth + holiday + weekday + weathersit +
    temp + windspeed + hum, data = train)

Residuals:
    Min      1Q  Median      3Q     Max
-4044.1  -467.1    33.2   568.8  2869.6

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  1974.76     252.86   7.810 3.02e-14 ***
season        527.34      67.00   7.871 1.95e-14 ***
yr           2051.84      77.62  26.435  < 2e-16 ***
mnth          -52.59      21.15  -2.487  0.01319 *
holiday      -685.08     217.85  -3.145  0.00175 **
```

```
weekday        60.91      19.28   3.159  0.00167 **
weathersit   -620.13      91.47  -6.780 3.18e-11 ***
temp         4010.11     186.13  21.545  < 2e-16 ***
windspeed   -1421.47     260.02  -5.467 7.03e-08 ***
hum          -733.22     351.90  -2.084  0.03767 *
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 898.9 on 538 degrees of freedom
Multiple R-squared:  0.7857,	Adjusted R-squared:  0.7821
F-statistic: 219.2 on 9 and 538 DF,  p-value: < 2.2e-16
```

So removing those variables is not helping to improve the model.

Some metrics we ca use to evaluate our regression model.

R Square (Coefficient of Determination) - This metric explains the percentage of variance explained by covariates in the model. It ranges between 0 and 1. Usually, higher values are desirable but it rests on the data quality and domain. For example, if the data is noisy, you'd be happy to accept a model at low R^2 values. But it's a good practice to consider adjusted R^2 than R^2 to determine model fit.

Adjusted R^2- The problem with R^2 is that it keeps on increasing as you increase the number of variables, regardless of the fact that the new variable is actually adding new information to the model. To overcome that, we use adjusted R^2 which doesn't increase (stays same or decrease) unless the newly added variable is truly useful.
F Statistics - It evaluates the overall significance of the model. It is the ratio of explained variance by the model by unexplained variance. It compares the full model with an intercept only (no predictors) model. Its value can range between zero and any arbitrary large number. Naturally, higher the F statistics, better the model.

Overall pvalue of the model is less than 0.5 which is acceptable.

RMSE / MSE / MAE - Error metric is the crucial evaluation number we must check. Since all these are errors, lower the number, better the model. Let's look at them one by one:

- MSE - This is mean squared error. It tends to amplify the impact of outliers on the model's accuracy. For example, suppose the actual y is 10 and predictive y is 30, the resultant MSE would be $(30-10)^2 = 400$.
- MAE - This is mean absolute error. It is robust against the effect of outliers. Using the previous example, the resultant MAE would be (30-10) = 20
- RMSE - This is root mean square error. It is interpreted as how far on an average, the residuals are from zero. It nullifies squared effect of MSE by square root and provides the result in original units as data. Here, the resultant RMSE would be $\sqrt{(30-10)^2} = 20$.

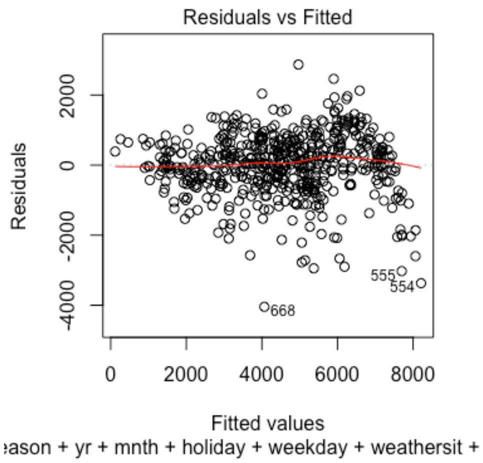
```
# library(DMwR)
# regr.eval(test$cnt,pred,stats = c('rmse','mape'))
```

rmse mape
812.7414680 0.1904365

Checking the performance of the model. RMSE is 812.74 while mate is 0.19. The model is 81% accurate.
We can draw some diagnostic plots .The diagnostic plots show residuals in four different ways.

1. Residual vs. Fitted Values Plot

plot(fit_final,which = 1)

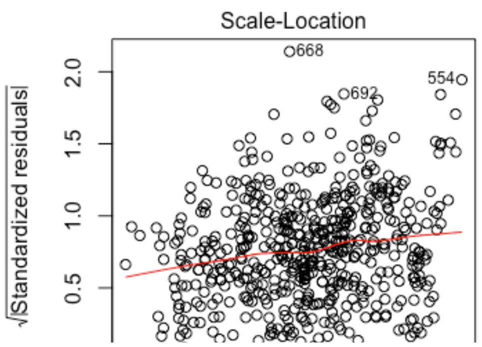


Ideally, this plot shouldn't show any pattern. But if you see any shape (curve, U shape), it suggests non-linearity in the data set. In addition, if you see a funnel shape pattern, it suggests your data is suffering from heteroskedasticity, i.e. the error terms have non-constant variance.

Here we can see that there no such pattern.

2. Normality Q-Q Plot

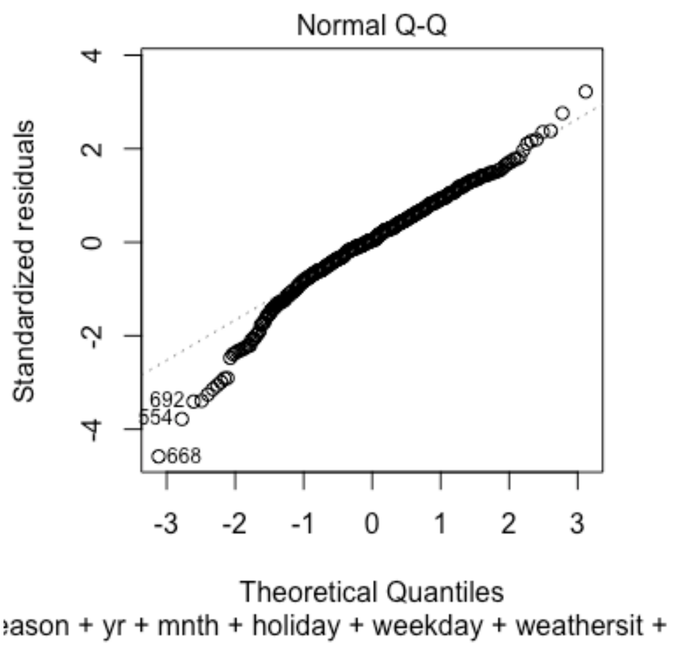
plot(fit_final,which = 2)



As the name suggests, this plot is used to determine the normal distribution of errors. It uses standardized values of residuals. Ideally, this plot should show a straight line. If you find a curved, distorted line, then your residuals have a non-normal distribution (problematic situation).
We can see almost a straight line.

3. Scale Location Plot

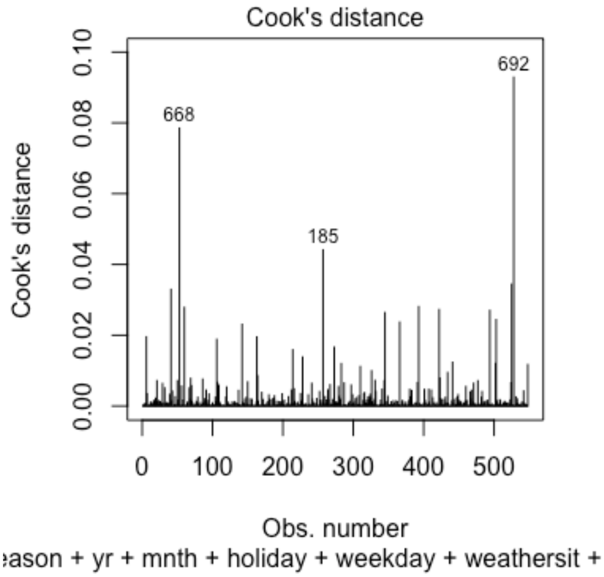
plot(fit_final,which = 3)



This plot is also useful to determine heteroskedasticity. Ideally, this plot shouldn't show any pattern. Presence of a pattern determine heteroskedasticity. Don't forget to corroborate the findings of this plot with the funnel shape in residual vs. fitted values.
We van see that there are no such pattern.

4. Cooks Distance (outliers detection)

plot(fit_final,which = 4)



This plot helps us to find influential cases (i.e., subjects) if any. Not all outliers are influential in linear regression analysis (whatever outliers mean). Even though data have extreme values, they might not be influential to determine a regression line. That means, the results wouldn't be much different if we either include or exclude them from analysis.
We can see that no points is there grater than p.1 which is good. Greater than 1 cooks distance is not acceptable.

We will try to build a new algorithm.

RANDOM FORSET:-

```
# control =trainControl(method="repeatedcv", number=10, repeats=3, search =
        'random')
# set.seed(2)
# rf =train(cnt~., data=train, method='rf',tuneLength=20 ,trControl=control)
```

We are doing randomizedsearcv cross validation of 3 X 10 times to find best mtry values.

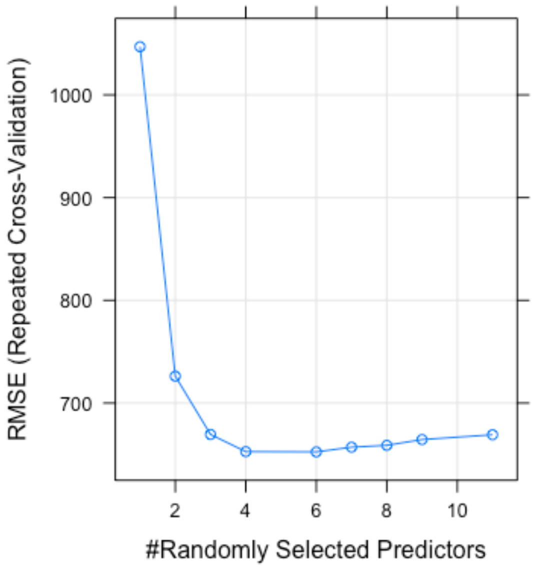
```
print(rf)
Random Forest

548 samples
 11 predictor

No pre-processing
Resampling: Cross-Validated (10 fold, repeated 3 times)
Summary of sample sizes: 492, 494, 494, 493, 493, 492, ...
Resampling results across tuning parameters:
```

mtry	RMSE	Rsquared	MAE
1	1046.9055	0.8428990	847.6682
2	726.2030	0.8836494	543.7346
3	669.4245	0.8902575	481.6546
4	652.7027	0.8914320	463.0729
6	652.4324	0.8881805	457.4399
7	657.0332	0.8860121	459.0452
8	658.8353	0.8846449	461.3609
9	664.4556	0.8824970	464.9264
11	669.1293	0.8802793	468.4354

RMSE was used to select the optimal model using the smallest value.
The final value used for the model was mtry = 6.
We are selecting mtry=6 as RMSE for this is the lowest.




```
# library(randomForest)
# rf_bike_rental=randomForest(cnt~.,train, importance=TRUE, ntree=100,mtry=6)
```

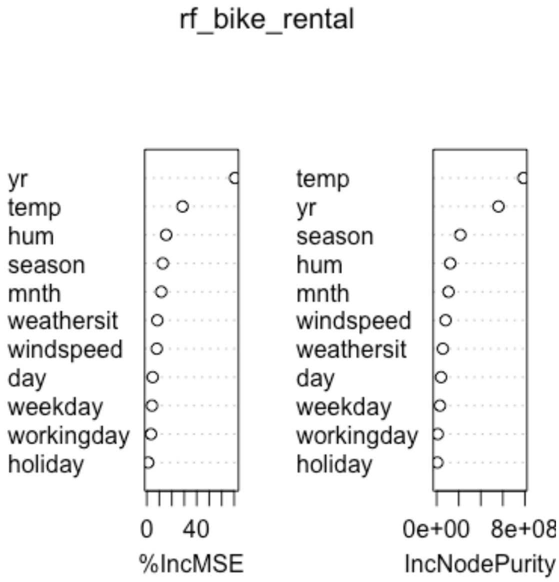
After building the model with ntree=100 we can see the important variables.

```
# importance(rf_bike_rental)
```

	%IncMSE	IncNodePurity
season	12.761319	214612003
yr	70.681655	560235810
mnth	11.570260	107466796
holiday	1.101394	6100805
weekday	4.128098	29555613
workingday	3.350137	9262647
weathersit	8.410737	55449229
temp	28.613028	787259450
hum	15.515859	122903462
windspeed	7.984124	80617038
day	4.696462	39577360

We can see that yr followed by temp, hum,season are the most important variable.

```
#varImp(rf_bike_rental)
            Overall
season     12.761319
yr         70.681655
mnth       11.570260
holiday     1.101394
weekday     4.128098
workingday  3.350137
weathersit  8.410737
temp       28.613028
hum        15.515859
windspeed   7.984124
day         4.696462
```



pred_rf=predict(rf_bike_rental,newdata = test)

regr.eval(test$cnt,pred_rf,stats = c('rmse','mape'))
rmse mape
674.40841 0.1614294

After predicting and evaluating the error metrics we can see that RMSE is 674.4 and make is 0.16 that means the model is 83.86% accurate.

3. Conclusion :-

3.1 Model Evaluation:-

Now that we have a few models for predicting the target variable, we need to decide which one to choose. There are several criteria that exist for evaluating and comparing models. We can compare the models using predictive performance of the models.

We have done several model building in R and in python. Based on different parameters we have chosen accuracy or MAPE and RMSE.

3.2 Model Selection :-

On based of above metrics, we will choose xgboost model in python which is 89.075 accurate.

3.3 Improvement of Model :-

We can do some other techniques and can used other different algorithms with different hyperparameter tuning for better model. Some way of improvements are as follows.

1. Trying building other models like LGB (Light Gradient boosting method) with hyperparameter tuning.

3. We could have done stacking model.

3. Based on important plot features we can drop some variables and create a new dataset and can build a model on top that.

4. SUMMARIZATION :-

In this project we have to predict the bike rent.

We have evaluated this model based on MAPE. We could have tried with different algorithm to increase the accuracy and Lowe down the RMSE as much as possible.

5. INSTRUCTION TO RUN PYTHON AND R CODE :-

PYTHON CODE:-

The 'Edwisor project- Bike Renting.ipynb' are attached along with the project. One can go to cell and click on the command 'run all' . The code will run automatically. Just in case while doing randomizedsearchcv if the result of the best parameter comes different from the existing one then just we have to change the parameters in the model based on tuning and run run the code.

While loading the file into Jupyter Notebook from the location of the data folder (day data), we have to pass the location.

```
bike_data=pd.read_csv("/Users/subhadeep/Downloads/day.csv")
```

R CODE:-

 # bike_data=read.csv("/Users/subhadeep/Downloads/day.csv")

After giving the right path of the training and testing data for downloading into R we can select all the codes will execute automatically.