Project Title : Seoul Bike Sharing Demand Prediction

Problem Description

Currently Rental bikes are introduced in many urban cities for the enhancement of mobility comfort. It is important to make the rental bike available and accessible to the public at the right time as it lessens the waiting time. Eventually, providing the city with a stable supply of rental bikes becomes a major concern. The crucial part is the prediction of bike count required at each hour for the stable supply of rental bikes.

Data Description

The dataset contains weather information (Temperature, Humidity, Windspeed, Visibility, Dewpoint, Solar radiation, Snowfall, Rainfall), the number of bikes rented per hour and date information.

Attribute Information:

- Date : year-month-day
- Rented Bike count - Count of bikes rented at each hour
- Hour - Hour of he day
- Temperature-Temperature in Celsius
- Humidity - %
- Windspeed - m/s
- Visibility - 10m
- Dew point temperature - Celsius
- Solar radiation - MJ/m2
- Rainfall - mm
- Snowfall - cm
- Seasons - Winter, Spring, Summer, Autumn
- Holiday - Holiday/No holiday
- Functional Day - NoFunc(Non Functional Hours), Fun(Functional hours)

Importing Libraries

```
#Import all library that will be used in entire project

%matplotlib inline
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
from scipy.stats import norm
from sklearn.preprocessing import StandardScaler
from scipy import stats
import warnings
warnings.filterwarnings('ignore')

#for date
import datetime

#for linear regression
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score
from sklearn.metrics import mean_squared_error
from numpy import math

#for decision tree
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import accuracy_score, auc
from sklearn.tree import DecisionTreeRegressor

#for random forest
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import roc_auc_score, confusion_matrix

#for gradient boosting
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import make_pipeline
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.decomposition import PCA

# for xg boost
import numpy as np
import pandas as pd
import xgboost as xg
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
```

Mount Drive And Import Data

```
#Mount google drive for access of the play store dataset
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
# Importing Dataset
File_path='/content/drive/MyDrive/Capstone project_2/'
data= pd.read_csv(File_path + 'SeoulBikeData.csv',encoding= 'unicode_escape')
```

```
# First Look
data.head()
```

	Date	Rented Bike Count	Hour	Temperature(°C)	Humidity(%)	Wind speed (m/s)	Visibility (10m)	De temperat
0	01/12/2017	254	0	-5.2	37	2.2	2000	
1	01/12/2017	204	1	-5.5	38	0.8	2000	
2	01/12/2017	173	2	-6.0	39	1.0	2000	
3	01/12/2017	107	3	-6.2	40	0.9	2000	
4	01/12/2017	78	4	-6.0	36	2.3	2000	

```
#tail of data
data.tail()
```

	Date	Rented Bike Count	Hour	Temperature(°C)	Humidity(%)	Wind speed (m/s)	Visibility (10m)	tempe
8755	30/11/2018	1003	19	4.2	34	2.6	1894	
8756	30/11/2018	764	20	3.4	37	2.3	2000	
8757	30/11/2018	694	21	2.6	39	0.3	1968	
8758	30/11/2018	712	22	2.1	41	1.0	1859	
8759	30/11/2018	584	23	1.9	43	1.3	1909	

```
#data information
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8760 entries, 0 to 8759
Data columns (total 14 columns):
 #   Column                     Non-Null Count  Dtype  
---  ------                     --------------  -----  
 0   Date                       8760 non-null   object 
 1   Rented Bike Count          8760 non-null   int64  
 2   Hour                       8760 non-null   int64  
 3   Temperature(°C)            8760 non-null   float64
 4   Humidity(%)                8760 non-null   int64  
 5   Wind speed (m/s)           8760 non-null   float64
 6   Visibility (10m)           8760 non-null   int64  
 7   Dew point temperature(°C)  8760 non-null   float64
 8   Solar Radiation (MJ/m2)    8760 non-null   float64
 9   Rainfall(mm)               8760 non-null   float64
 10  Snowfall (cm)              8760 non-null   float64
 11  Seasons                    8760 non-null   object 
 12  Holiday                    8760 non-null   object 
 13  Functioning Day            8760 non-null   object 
dtypes: float64(6), int64(4), object(4)
memory usage: 958.2+ KB
```

```
#Discription of Data
data.describe(include='all')
```

	Date	Rented Bike Count	Hour	Temperature(°C)	Humidity(%)	Wind speed (m/s)
count	8760	8760.000000	8760.000000	8760.000000	8760.000000	8760.000000
unique	365	NaN	NaN	NaN	NaN	NaN
top	01/12/2017	NaN	NaN	NaN	NaN	NaN
freq	24	NaN	NaN	NaN	NaN	NaN
mean	NaN	704.602055	11.500000	12.882922	58.226256	1.724909
std	NaN	644.997468	6.922582	11.944825	20.362413	1.036300
min	NaN	0.000000	0.000000	-17.800000	0.000000	0.000000
25%	NaN	191.000000	5.750000	3.500000	42.000000	0.900000
50%	NaN	504.500000	11.500000	13.700000	57.000000	1.500000
75%	NaN	1065.250000	17.250000	22.500000	74.000000	2.300000
max	NaN	3556.000000	23.000000	39.400000	98.000000	7.400000

Handling Missing Vaules

```
#checking for null
data.isnull().any()
```

```
Date                         False
Rented Bike Count            False
Hour                         False
Temperature(°C)              False
Humidity(%)                  False
Wind speed (m/s)             False
Visibility (10m)             False
Dew point temperature(°C)    False
Solar Radiation (MJ/m2)      False
Rainfall(mm)                 False
Snowfall (cm)                False
Seasons                      False
Holiday                      False
Functioning Day              False
dtype: bool
```

No null values in our data

Making Data In Proper Format

```
#type of date
type(data['Date'][0])
```

```
str
```

```
#converting date type in to Timestamp
data['Date'] = pd.to_datetime(data['Date'])
```

```
#creating new columns year,month and day
data['year'] = pd.DatetimeIndex(data['Date']).year
data['month'] = pd.DatetimeIndex(data['Date']).month_name()
data['day'] = pd.DatetimeIndex(data['Date']).day_name()
```

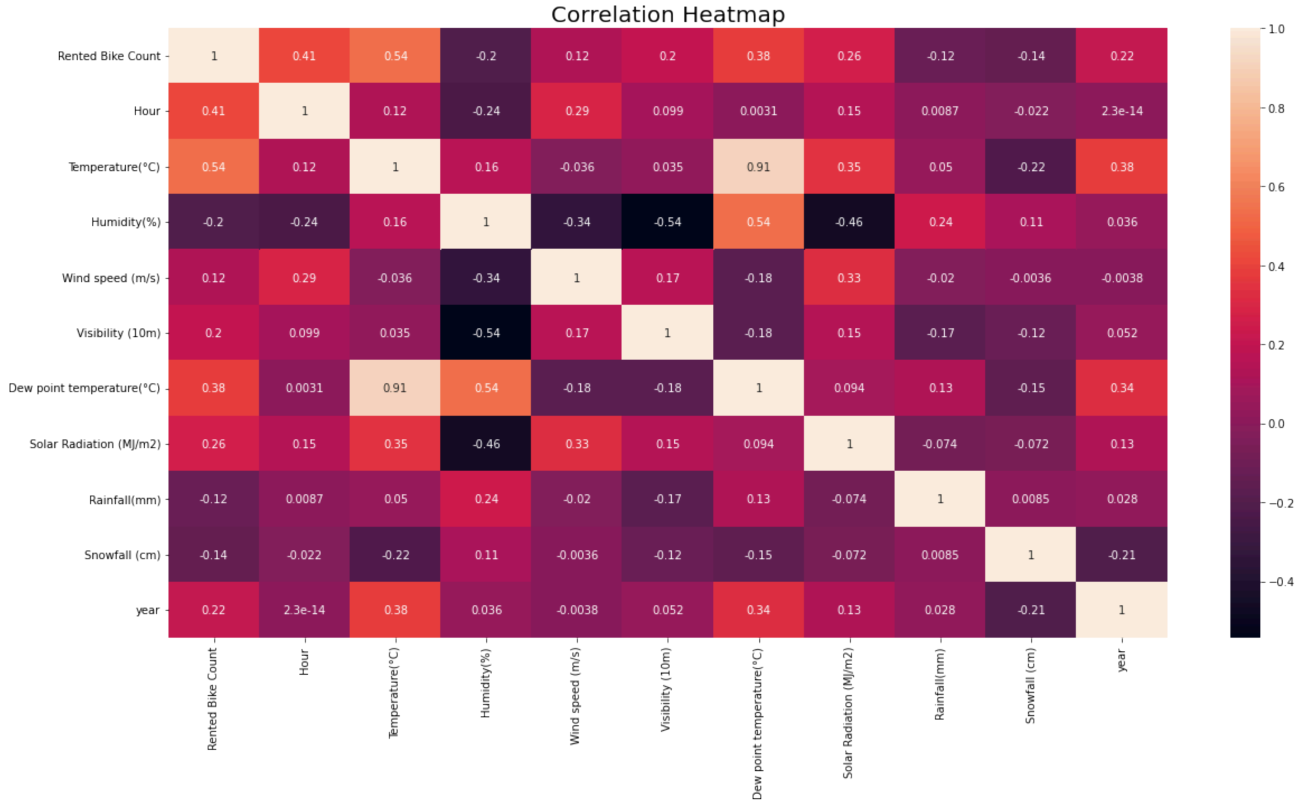
```
#data head
data.head(1)
```

	Date	Rented Bike Count	Hour	Temperature(°C)	Humidity(%)	Wind speed (m/s)	Visibility (10m)	Dew point temperature(°
0	2017-01-12	254	0	-5.2	37	2.2	2000	-1

Correlation Heatmap

```
#Correlation Heatmap
plt.figure(figsize = (20,10))
sns.heatmap(data.corr(), annot= True)
plt.title("Correlation Heatmap",fontsize=20)
```

```
Text(0.5, 1.0, 'Correlation Heatmap')
```



Drop column of Dew point temperature(°C) as there is high correlation in Temperature and Dew point temperature(°C)

```
# Temperature(°C) and Dew point temperature(°C)
data[["Temperature(°C)","Dew point temperature(°C)"]]
```

	Temperature(°C)	Dew point temperature(°C)
0	-5.2	-17.6
1	-5.5	-17.6
2	-6.0	-17.7
3	-6.2	-17.6
4	-6.0	-18.6
...	...	...
8755	4.2	-10.3
8756	3.4	-9.9
8757	2.6	-9.9
8758	2.1	-9.8
8759	1.9	-9.3

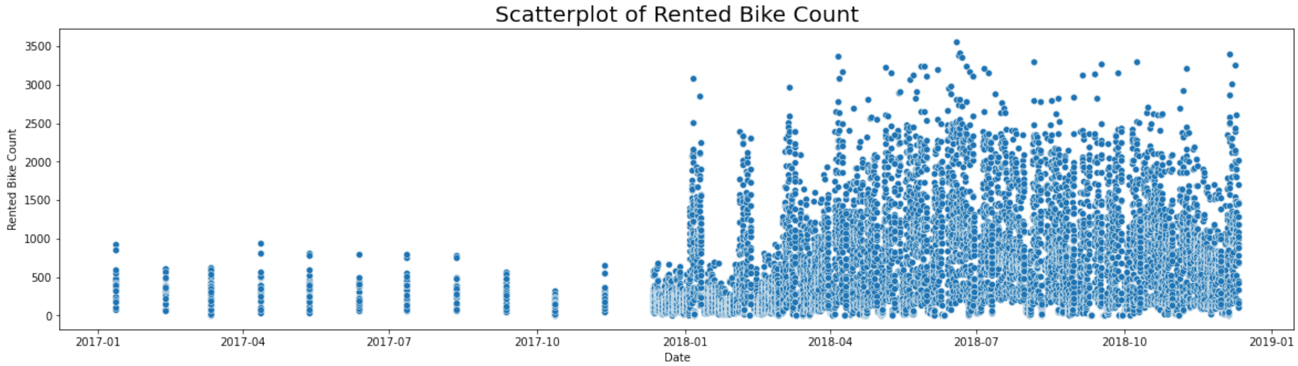
8760 rows × 2 columns

```
#drop Dew point temperature(°C)
data.drop(columns=['Dew point temperature(°C)'], axis=1,inplace=True)
```

EDA

```
# Scatterplot of Mean Rented Bike Count
plt.figure(figsize = (20,5))
sns.scatterplot(x="Date", y="Rented Bike Count", data=data)
plt.title("Scatterplot of Rented Bike Count",fontsize=20)
```

```
Text(0.5, 1.0, 'Scatterplot of Rented Bike Count')
```



We can see that there is high demand of Rented bike in year 2018 when compare with year 2017

```
# find categorical variables
categorical = [var for var in data.columns if data[var].dtype=='O']
print('There are {} categorical variables'.format(len(categorical)))
```

```
There are 5 categorical variables
```

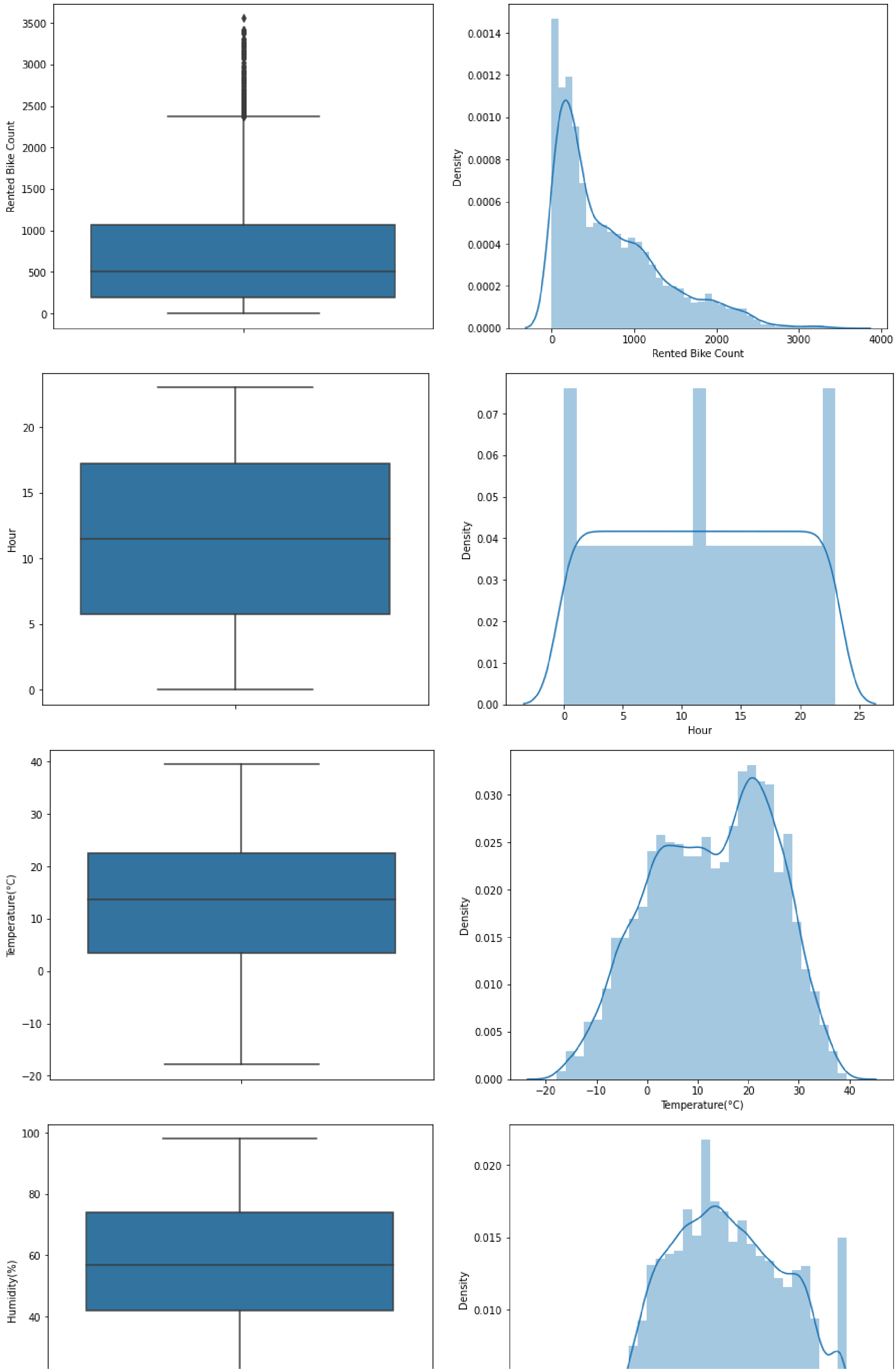
```
# find Numerical variables
numerical = [var for var in data.columns if data[var].dtype!='O']
print('There are {} numerical variables'.format(len(numerical)))
```

```
There are 11 numerical variables
```

```
#remove date
numerical.remove('Date')
```

Boxplot And Distribution Plot Of Numerical Variables

```
#Boxplot And Distribution Plot Of Numerical Variables
for var in numerical:
    plt.figure(figsize=(15,6))
    plt.subplot(1, 2, 1)
    fig = sns.boxplot(y=data[var])
    fig.set_title('')
    fig.set_ylabel(var)

    plt.subplot(1, 2, 2)
    fig = sns.distplot(data[var].dropna())
    fig.set_xlabel(var)

    plt.show()
```


Rented Bike Count
Density
Rented Bike Count
Hour
Density
Hour
Temperature(°C)
Density
Temperature(°C)
Humidity(%)
Density

```
# Number of labels: cardinality
#Let's now check if our categorical variables have a huge number of categories.
for var in categorical:
    print(var, ' contains ', len(data[var].unique()), ' labels')
```

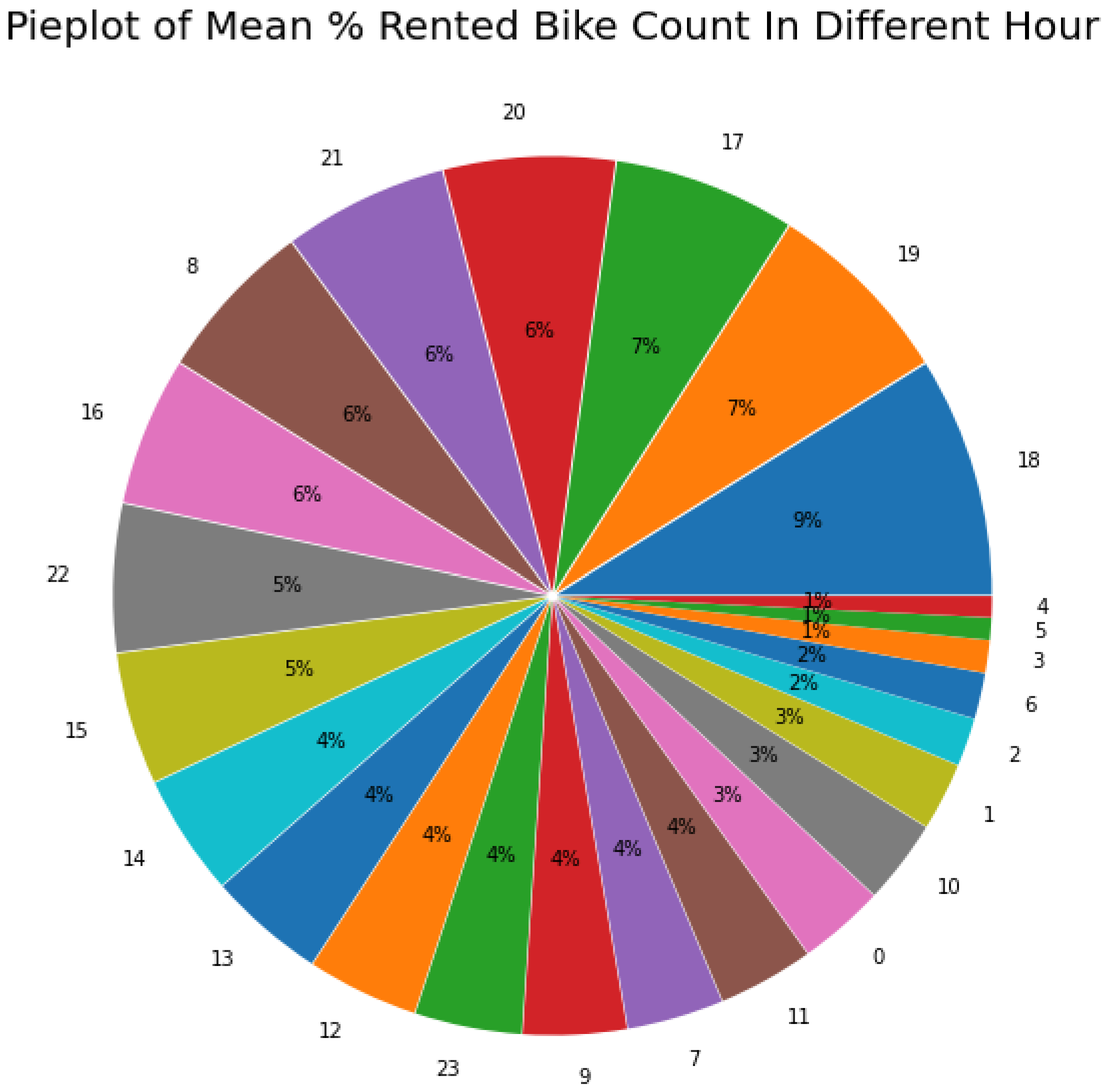
```
Seasons  contains  4  labels
Holiday  contains  2  labels
Functioning Day  contains  2  labels
month  contains  12  labels
day  contains  7  labels
```

Mean Rented Bike Count In Different Hour


```
#Mean Rented Bike Count By Hour
rented_bike_count_hour=data.groupby('Hour')['Rented Bike Count'].mean().reset_index(name=
rented_bike_count_hour
```

	Hour	Rented Bike Count
18	18	1502.926027
19	19	1195.147945
17	17	1138.509589
20	20	1068.964384
21	21	1031.449315
8	8	1015.701370
16	16	930.621918
22	22	922.797260
15	15	829.186301
14	14	758.824658
13	13	733.246575
12	12	699.441096
23	23	671.126027
9	9	645.983562
7	7	606.005479
11	11	600.852055
0	0	541.460274
10	10	527.821918
1	1	426.183562
2	2	301.630137
6	6	287.564384
3	3	203.331507
5	5	139.082192
4	4	132.591781

```
#Pieplot of Mean Rented Bike Count In Different Hour
plt.rcParams['figure.figsize'] = (20,10)
plt.pie(rented_bike_count_hour["Rented Bike Count"],labels=rented_bike_count_hour['Hour']
plt.title('Pieplot of Mean % Rented Bike Count In Different Hour',fontsize=20)
plt.show()
```



We can conclude from above pieplot that demand of rented bike is high in Hour 18

```
#sns.histplot(x='Hour', y="Rented Bike Count", data=rented_bike_count_hour)
```

Mean Rented Bike Count In Different Temperature

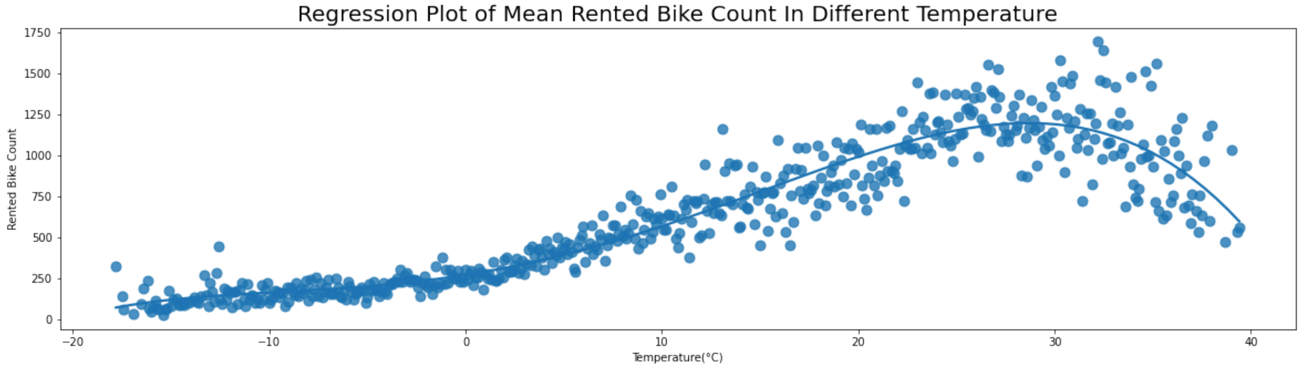
```
#Mean Rented Bike Count In Different Temperature
temp_and_rented_bike= data.groupby('Temperature(°C)')['Rented Bike Count'].mean().reset_i
temp_and_rented_bike
```

	Temperature(°C)	Rented Bike Count
485	32.2	1692.875000
488	32.5	1638.000000
466	30.3	1579.750000
515	35.2	1558.333333
429	26.6	1552.650000
...	...	...
14	-15.3	63.833333
12	-15.6	60.333333
8	-16.0	46.000000
3	-16.9	36.000000
13	-15.4	24.500000

546 rows × 2 columns

```
# regression plot of Mean Rented Bike Count In Different Temperature
plt.figure(figsize = (20,5))
sns.regplot(x="Temperature(°C)", y="Rented Bike Count", data=temp_and_rented_bike,
                 scatter_kws={"s": 80},
                 order=4, ci=None)
plt.title("Regression Plot of Mean Rented Bike Count In Different Temperature",fontsize=2
```

```
Text(0.5, 1.0, 'Regression Plot of Mean Rented Bike Count In Different Temperature')
```



From above plot we can see that as temperature increases count of rented bike also increases

```
#plt.figure(figsize = (50,50))
#temp_and_rented_bike.plot(x="Temperature(°C)",y="Rented Bike Count")
```

Mean Rented Bike Count In Different Humidity

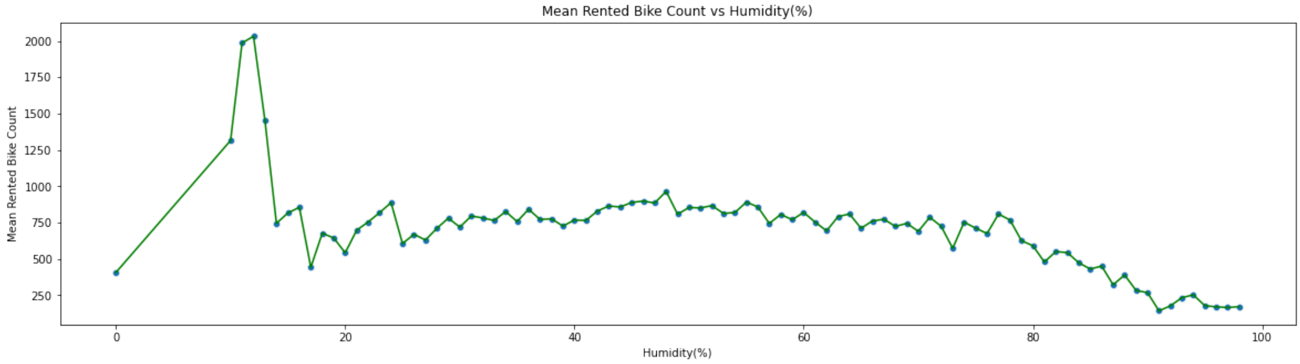
```
#Mean Rented Bike Count In Different Humidity
humidity_and_rented_bike= data.groupby('Humidity(%)')['Rented Bike Count'].mean().reset_i
humidity_and_rented_bike
```

	Humidity(%)	Rented Bike Count
3	12	2032.000000
2	11	1986.000000
4	13	1451.000000
1	10	1315.000000
39	48	965.284553
...	...	...
83	92	177.851852
89	98	172.320000
87	96	170.828829
88	97	166.069364
82	91	143.394737

90 rows × 2 columns

```
#Scatterplot and lineplot of Mean Rented Bike Count In Different Humidity
plt.figure(figsize = (20,5))
sns.scatterplot(x="Humidity(%)", y="Rented Bike Count", data=humidity_and_rented_bike)
sns.lineplot(x="Humidity(%)", y="Rented Bike Count", data=humidity_and_rented_bike, color
plt.title('Mean Rented Bike Count vs Humidity(%)')
plt.ylabel('Mean Rented Bike Count')
plt.xlabel('Humidity(%)')
```

```
Text(0.5, 0, 'Humidity(%)')
```



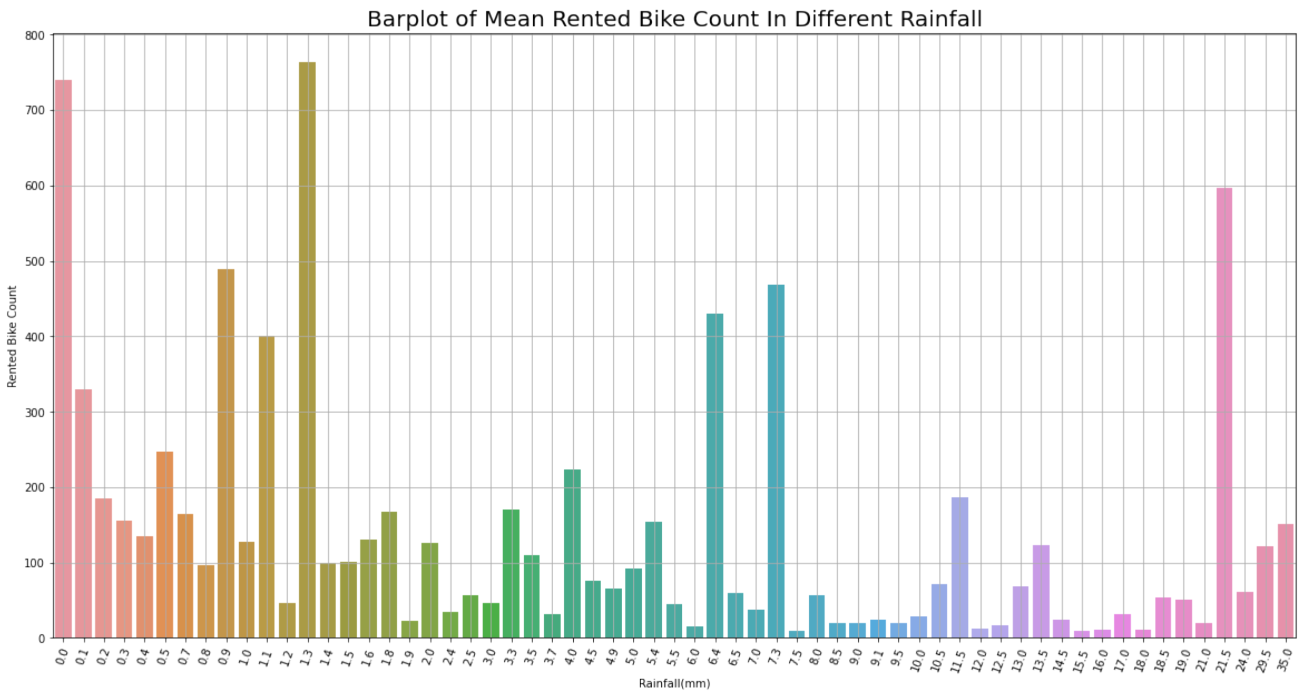
Mean Rented Bike Count In Different Rainfall

```
#Mean Rented Bike Count In Different Rainfall
rainfall_and_rented_bike= data.groupby('Rainfall(mm)')['Rented Bike Count'].mean().reset_
rainfall_and_rented_bike
```

	Rainfall(mm)	Rented Bike Count
12	1.3	764.000000
0	0.0	739.311103
57	21.5	596.000000
8	0.9	489.333333
35	7.3	468.000000
...	...	...
45	12.0	13.000000
51	16.0	11.000000
53	18.0	10.500000
50	15.5	10.000000
36	7.5	9.000000

61 rows × 2 columns

```
# Barplot of mean ranted bike coungt by rainfall
plt.figure(figsize = (20,10))
sns.barplot(x="Rainfall(mm)", y="Rented Bike Count", data=rainfall_and_rented_bike)
plt.title("Barplot of Mean Rented Bike Count In Different Rainfall",fontsize=20)
plt.xticks(rotation=70, horizontalalignment="center")
plt.grid()
```

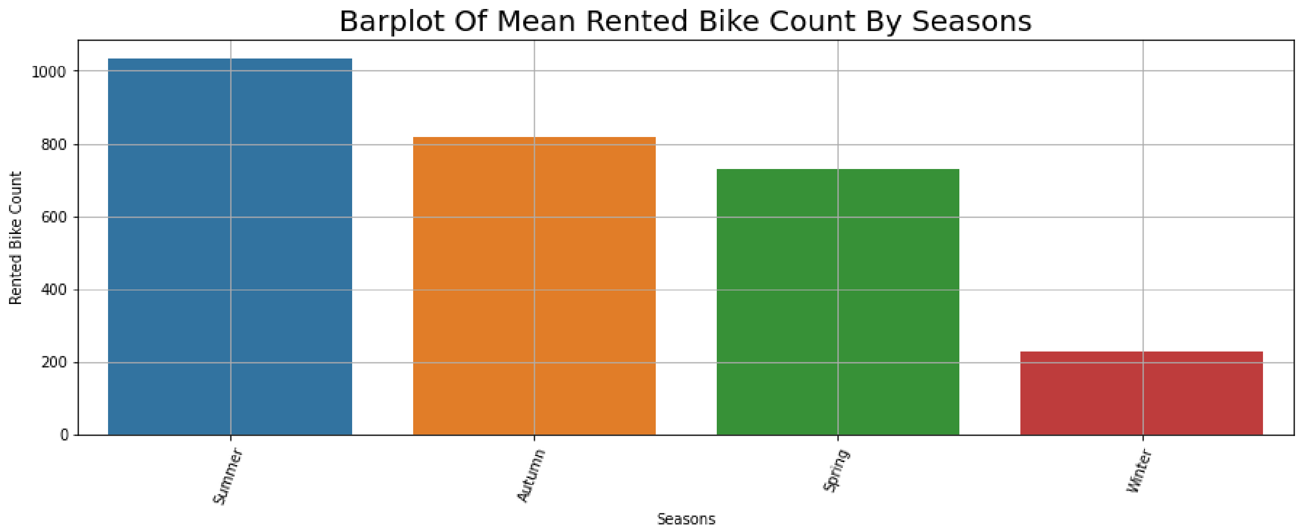


Mean Rented Bike Count In Different Seasons

```
#Mean Rented Bike Count In Different Seasons
seasons_and_rented_bike= data.groupby('Seasons')['Rented Bike Count'].mean().reset_index(
seasons_and_rented_bike
```

	Seasons	Rented Bike Count
2	Summer	1034.073370
0	Autumn	819.597985
1	Spring	730.031250
3	Winter	225.541204

```
#Barplot of mean ranted bike count In Different Seasons
plt.figure(figsize = (15,5))
sns.barplot(x="Seasons", y="Rented Bike Count", data=seasons_and_rented_bike)
plt.title("Barplot Of Mean Rented Bike Count By Seasons",fontsize=20)
plt.xticks(rotation=70, horizontalalignment="center")
plt.grid()
```



Count Of Rented Bike Is High In Summer

Mean Rented Bike Count In Holidays,Functioning day And In Different Year,Months And Days

```
#Mean Rented Bike Count In Holidays
holiday_and_rented_bike= data.groupby('Holiday')['Rented Bike Count'].mean().reset_index(
holiday_and_rented_bike
```

	Holiday	Rented Bike Count
1	No Holiday	715.228026
0	Holiday	499.756944

```
#Mean Rented Bike Count In Functioning day
Functioning_Day_and_rented_bike= data.groupby('Functioning Day')['Rented Bike Count'].mea
Functioning_Day_and_rented_bike
```

	Functioning Day	Rented Bike Count
1	Yes	729.156999
0	No	0.000000

```
#Mean Rented Bike Count In Different Year
year_and_rented_bike= data.groupby('year')['Rented Bike Count'].mean().reset_index(name="
year_and_rented_bike
```

	year	Rented Bike Count
1	2018	746.879242
0	2017	249.099462

```
#Mean Rented Bike Count In Different Month
month_and_rented_bike= data.groupby('month')['Rented Bike Count'].mean().reset_index(name
month_and_rented_bike
```

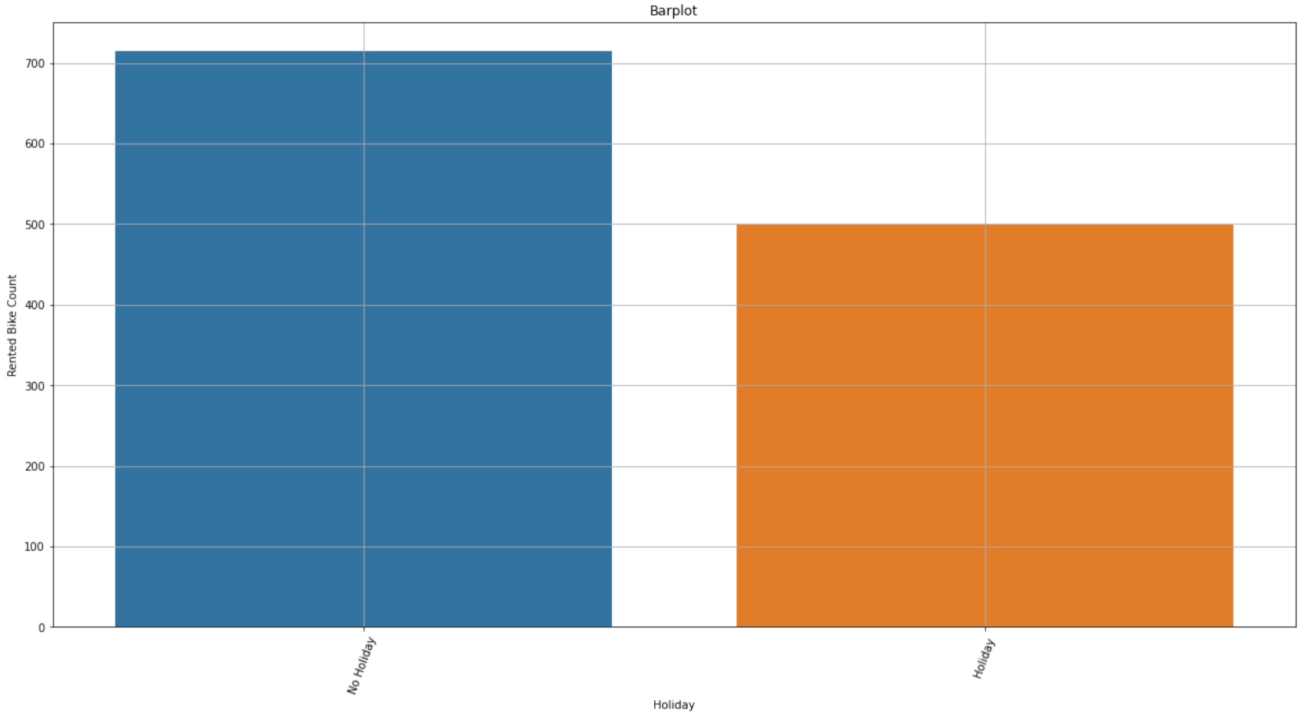
	month	Rented Bike Count
6	June	981.566667
5	July	929.219086
8	May	895.091398
10	October	842.725806
1	August	825.524194
0	April	772.526389
11	September	693.508333
9	November	685.294444
7	March	611.608871
2	December	419.047043
3	February	393.023810
4	January	386.080645

```
#Mean Rented Bike Count In Different Day
day_and_rented_bike= data.groupby('day')['Rented Bike Count'].mean().reset_index(name="Re
day_and_rented_bike
```

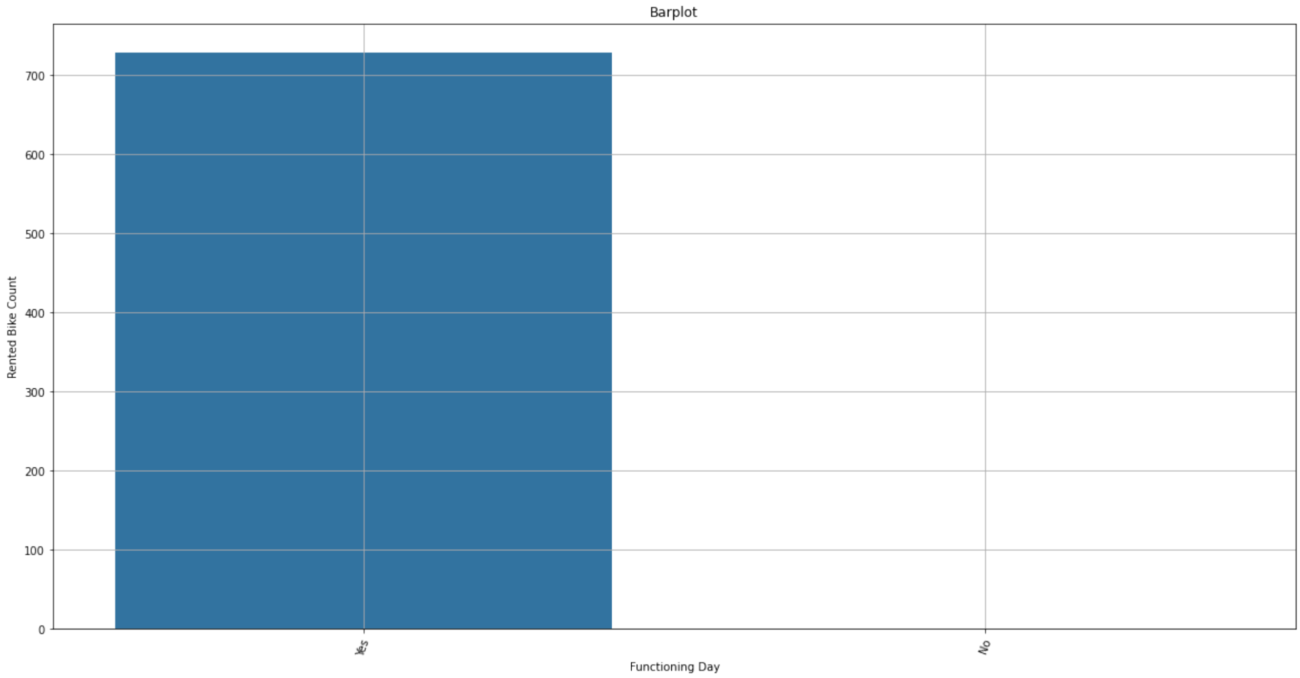
	day	Rented Bike Count
4	Thursday	743.803686
0	Friday	734.449346
2	Saturday	730.348558
1	Monday	719.635833
6	Wednesday	714.521226
5	Tuesday	678.362421
3	Sunday	615.968364

```
#Barplots of holiday_and_rented_bike,Functioning_Day_and_rented_bike,year_and_rented_bike
var=[holiday_and_rented_bike,Functioning_Day_and_rented_bike,year_and_rented_bike,month_a

for i in var:
    plt.figure(figsize=(10,5))
    plt.subplots(1,1)
    fig = sns.barplot(x=i.columns[0], y=i.columns[1], data=i)
    plt.xticks(rotation=70, horizontalalignment="center")
    fig.set_title('Barplot')
    plt.grid()

    plt.show()
```

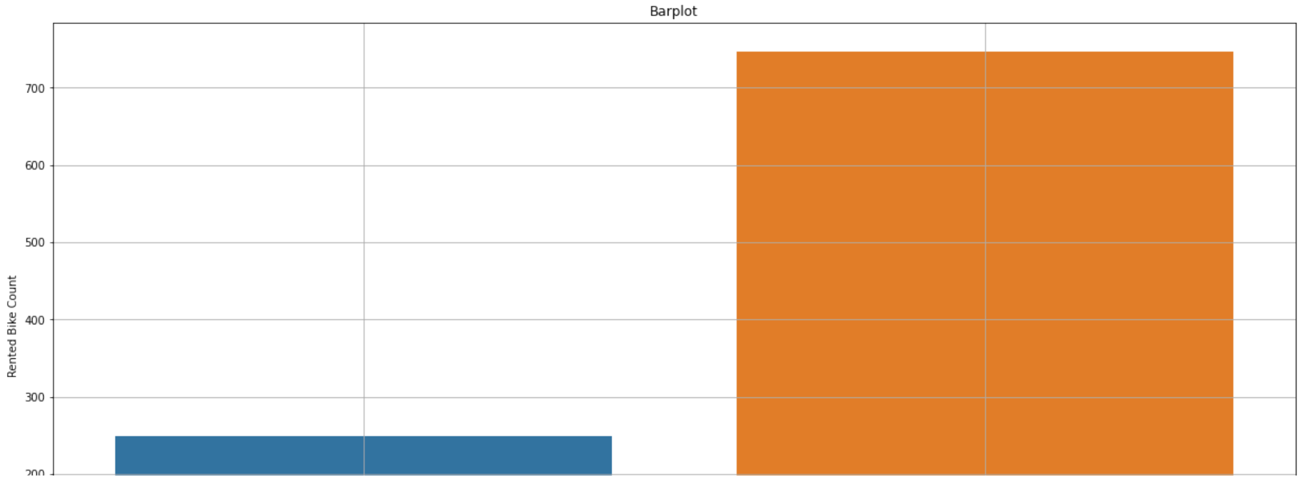
<Figure size 720x360 with 0 Axes>



<Figure size 720x360 with 0 Axes>

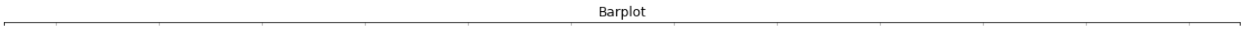


<Figure size 720x360 with 0 Axes>



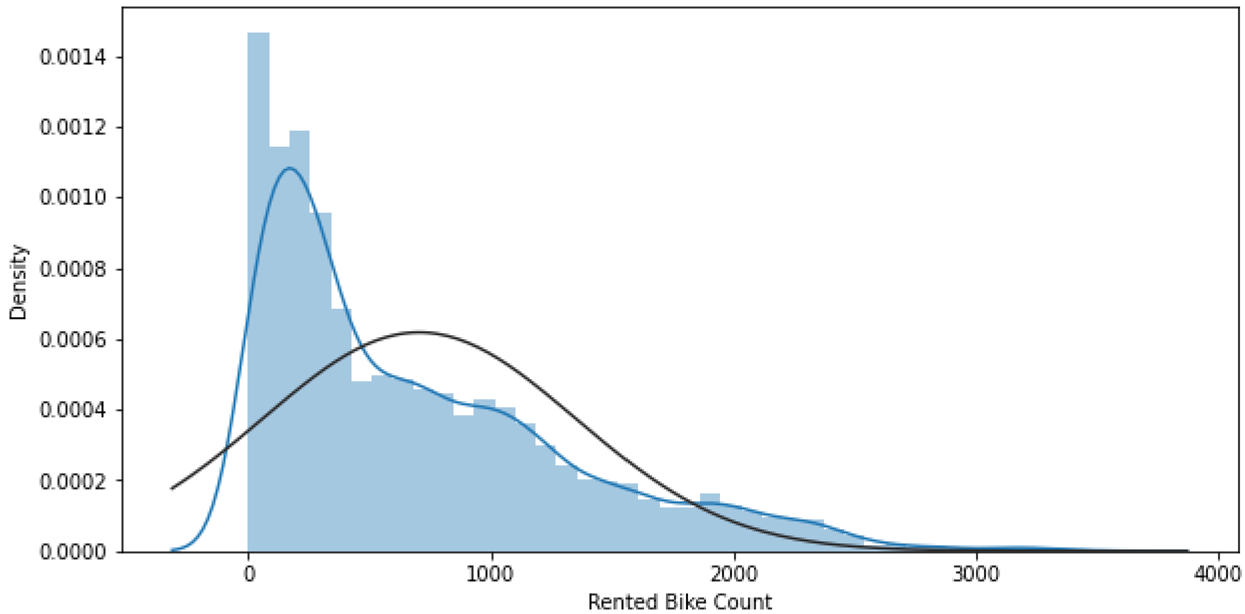
Count Of Rented Bike Is High In Non Holidays,Functioning Days, 2018 year and in June Month.

Final Data



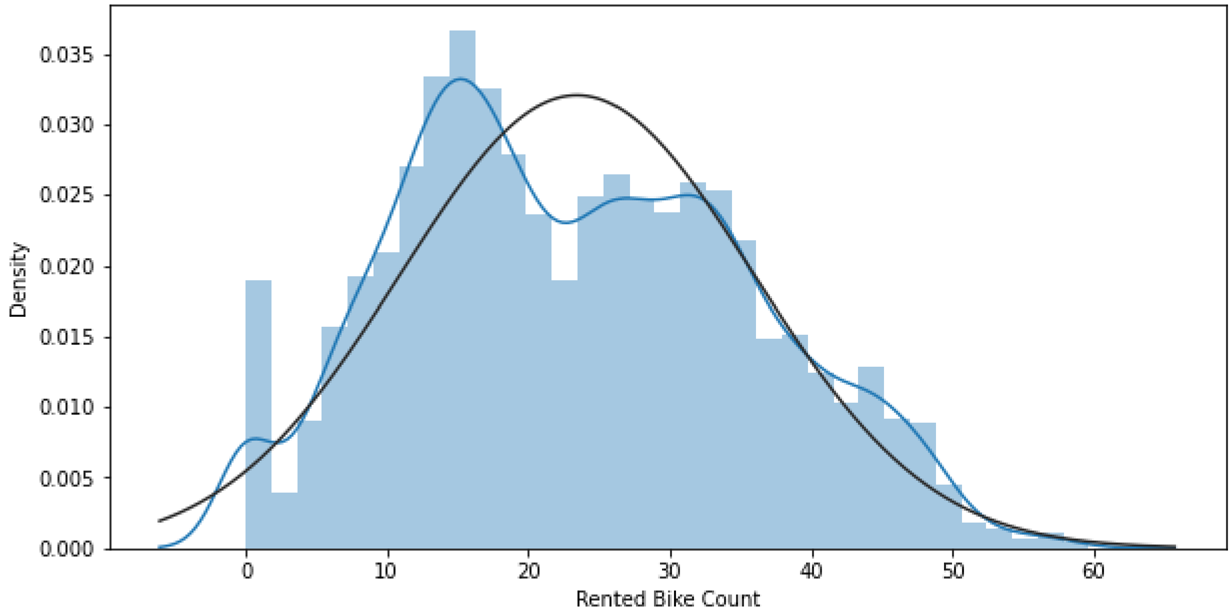

```
# Encode Categorical Variables - one hot encoding
from sklearn.preprocessing import OneHotEncoder

#creating instance of one-hot-encoder
encoder = OneHotEncoder(handle_unknown='ignore')

#perform one-hot encoding on 'team' column
encoder_df = pd.DataFrame(encoder.fit_transform(data[['Seasons','Holiday','Functioning Da

encoder_df.columns = encoder.get_feature_names(['Seasons','Holiday','Functioning Day','ye

#merge one-hot encoded columns back with original DataFrame
final_df = data.join(encoder_df)
```

```
# Target Variable Transformation
plt.figure(figsize=(10,5))
sns.distplot(final_df['Rented Bike Count'], fit=norm);
fig = plt.figure()
```




```
<Figure size 1440x720 with 0 Axes>
```

```
# It looks more normal now.
plt.figure(figsize=(10,5))
sns.distplot(np.sqrt(final_df['Rented Bike Count']), fit=norm);
fig = plt.figure()
```




```
<Figure size 1440x720 with 0 Axes>
```

```
#Dependant variable
Y =np.sqrt(final_df['Rented Bike Count'])
```

```
#Independant variable
final_df.drop(columns=['Rented Bike Count','Date','year','month','day'], axis=1,inplace=T
final_df.drop(columns=['Seasons','Holiday','Functioning Day'], axis=1,inplace=True)
X=final_df
```

Linear Regression

```
# Splitting the dataset into the Training set and Test set
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size = 0.2, random_state =
```

```
# Transforming data
scaler = MinMaxScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

```
# Fitting Multiple Linear Regression to the Training set
regressor = LinearRegression()
regressor.fit(X_train, y_train)
```

```
LinearRegression()
```

```
#regressor intercept,coefficients
print("Intercept:", regressor.intercept_)
print("Coefficients:",regressor.coef_)
```

```
Intercept: -102520444180419.03
Coefficients: [ 1.10346279e+01  2.97723946e+01 -1.48386086e+01  7.09373545e-01
  1.67971463e+00 -3.11574641e+00 -5.38233806e+01 -3.53716418e-02
  3.36893748e+14  3.36893748e+14  3.36893748e+14  3.36893748e+14
  2.09338342e+14  2.09338342e+14  5.69583580e+13  5.69583580e+13
 -2.05934417e+14 -2.05934417e+14 -5.15073021e+13 -5.15073021e+13
 -5.15073021e+13 -5.15073021e+13 -5.15073021e+13 -5.15073021e+13
 -5.15073021e+13 -5.15073021e+13 -5.15073021e+13 -5.15073021e+13
 -5.15073021e+13 -5.15073021e+13 -2.43228286e+14 -2.43228286e+14
 -2.43228286e+14 -2.43228286e+14 -2.43228286e+14 -2.43228286e+14
 -2.43228286e+14]
```

```
# Predicting the results
y_pred_train = regressor.predict(X_train)
y_pred_test = regressor.predict(X_test)
```

```
#Evaluation for train set
mean_squared_error_linear_train=mean_squared_error(y_train, y_pred_train)
r2_score_linear_train= r2_score(y_train, y_pred_train)
adjusted_r2_score_linear_train=1-(1-r2_score((y_train), (y_pred_train)))*((X_train.shape[
print("mean_squared_error_linear_train:",mean_squared_error_linear_train)
print("r2_score_linear_train:",r2_score_linear_train)
print("adjusted_r2_score_linear_train:",adjusted_r2_score_linear_train)
```

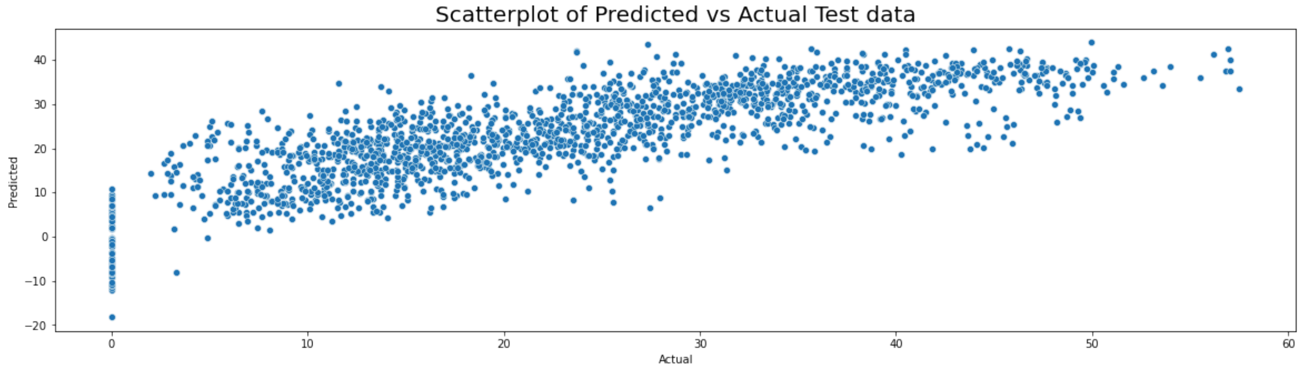
```
mean_squared_error_linear_train: 51.63196704135537
r2_score_linear_train: 0.6654511607517628
adjusted_r2_score_linear_train: 0.6636752199982212
```

```
#Evaluation for test set
mean_squared_error_linear_test=mean_squared_error(y_test, y_pred_test)
r2_score_linear_test= r2_score(y_test, y_pred_test)
adjusted_r2_score_linear_test=1-(1-r2_score((y_test), (y_pred_test)))*((X_test.shape[0]-1
print("mean_squared_error_linear_test:",mean_squared_error_linear_test)
print("r2_score_linear_test:",r2_score_linear_test)
print("adjusted_r2_score_linear_test:",adjusted_r2_score_linear_test)
```

```
mean_squared_error_linear_test: 53.377437083669705
r2_score_linear_test: 0.6610660635389255
adjusted_r2_score_linear_test: 0.6537495199863819
```

```
#Scatterplot of fitted vs Actual Test data
plt.figure(figsize = (20,5))
sns.scatterplot(x=y_test, y=y_pred_test)
plt.title('Scatterplot of Predicted vs Actual Test data',fontsize=20)
plt.ylabel('Predicted')
plt.xlabel('Actual')
```

```
Text(0.5, 0, 'Actual')
```



Decision Tree

```
# Splitting the dataset into the Training set and Test set
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.33, random_state=32
```

```
TreeRegressor= DecisionTreeRegressor(criterion='mse', random_state=0)
TreeRegressor.fit(X_train, y_train)
```

```
DecisionTreeRegressor(criterion='mse', random_state=0)
```

```
# Predicting the results
y_pred_train = TreeRegressor.predict(X_train)
y_pred_test =TreeRegressor.predict(X_test)
```

```
#Evaluation for train set
mean_squared_error_decision_tree_train=mean_squared_error(y_train, y_pred_train)
r2_score_decision_tree_train= r2_score(y_train, y_pred_train)
adjusted_r2_score_decision_tree_train=1-(1-r2_score((y_train), (y_pred_train)))*((X_train
print("mean_squared_error_decision_tree_train:",mean_squared_error_decision_tree_train)
print("r2_score_decision_tree_train:",r2_score_decision_tree_train)
print("adjusted_r2_score_decision_tree_train:",adjusted_r2_score_decision_tree_train)
```

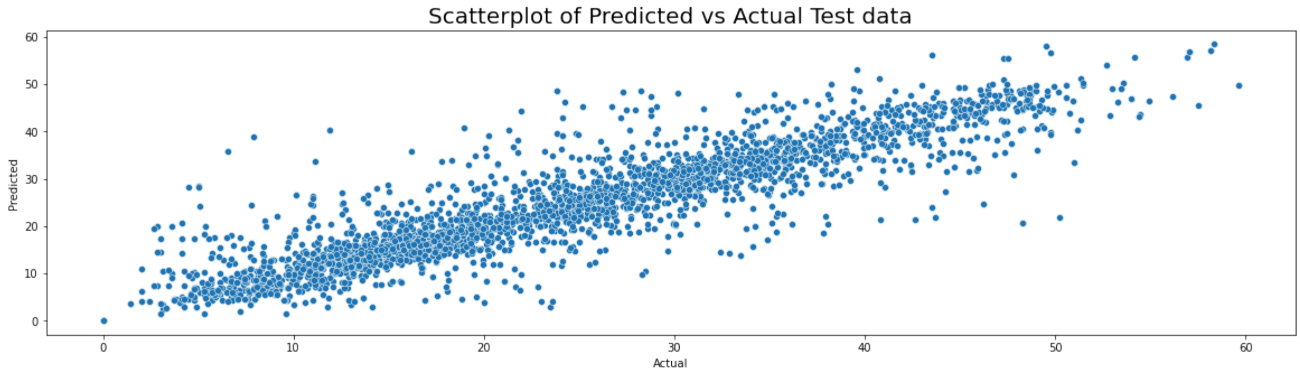
```
mean_squared_error_decision_tree_train: 0.0
r2_score_decision_tree_train: 1.0
adjusted_r2_score_decision_tree_train: 1.0
```

```
#Evaluation for test set
mean_squared_error_decision_tree_test=mean_squared_error(y_test, y_pred_test)
r2_score_decision_tree_test= r2_score(y_test, y_pred_test)
adjusted_r2_score_decision_tree_test=1-(1-r2_score((y_test), (y_pred_test)))*((X_test.sha
print("mean_squared_error_decision_tree_test:",mean_squared_error_decision_tree_test)
print("r2_score_decision_tree_test:",r2_score_decision_tree_test)
print("adjusted_r2_score_decision_tree_test:",adjusted_r2_score_decision_tree_test)
```

```
mean_squared_error_decision_tree_test: 26.043439393364014
r2_score_decision_tree_test: 0.8301089682610303
adjusted_r2_score_decision_tree_test: 0.8279056846387584
```

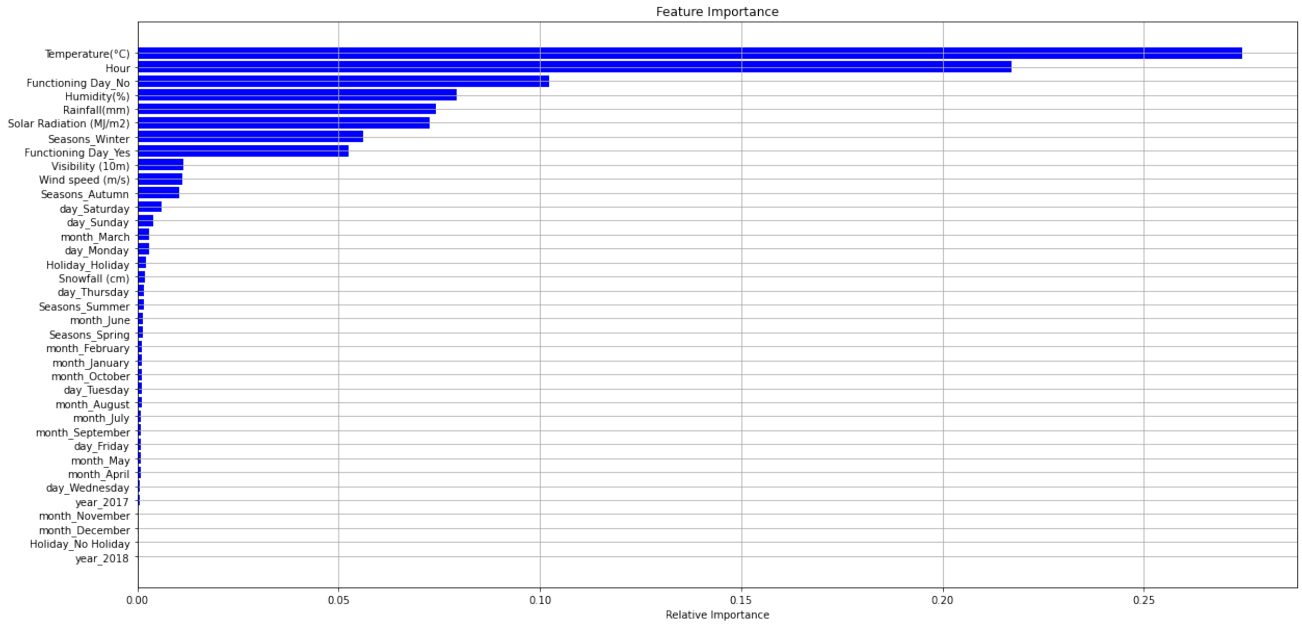
```
#Scatterplot of fitted vs Actual Test data
plt.figure(figsize = (20,5))
sns.scatterplot(x=y_test, y=y_pred_test)
plt.title('Scatterplot of Predicted vs Actual Test data',fontsize=20)
plt.ylabel('Predicted')
plt.xlabel('Actual')
```

```
Text(0.5, 0, 'Actual')
```




```
#storing features and there importance
features = X_train.columns
importances = TreeRegressor.feature_importances_
indices = np.argsort(importances)
```

```
#barh plot of features and there importance
plt.figure(figsize=(20,10))
plt.title('Feature Importance')
plt.barh(range(len(indices)), importances[indices], color='blue', align='center')
plt.yticks(range(len(indices)), [features[i] for i in indices])
plt.xlabel('Relative Importance')
plt.grid()
plt.show()
```



Decision Tree Using Grid Search CV

```
#Grid search CV
grid_values = {'max_depth':[3, 5, 7]}
TreeRegressorr= GridSearchCV(TreeRegressor, param_grid = grid_values, cv=5)

# Fit the object to train dataset
TreeRegressorr.fit(X_train, y_train)
```

```
GridSearchCV(cv=5,
             estimator=DecisionTreeRegressor(criterion='mse', random_state=0),
             param_grid={'max_depth': [3, 5, 7]})
```

```
#best found parameters
TreeRegressorr.get_params().keys()
```

```
dict_keys(['cv', 'error_score', 'estimator__ccp_alpha', 'estimator__criterion',
'estimator__max_depth', 'estimator__max_features', 'estimator__max_leaf_nodes',
'estimator__min_impurity_decrease', 'estimator__min_samples_leaf',
'estimator__min_samples_split', 'estimator__min_weight_fraction_leaf',
```

```
'estimator__random_state', 'estimator__splitter', 'estimator', 'n_jobs',
'param_grid', 'pre_dispatch', 'refit', 'return_train_score', 'scoring', 'verbose'])
```

```
# Predicting the results
y_pred_train = TreeRegressorr.predict(X_train)
y_pred_test =TreeRegressorr.predict(X_test)
```

```
#Evaluation for train set
mean_squared_error_decision_tree_gridcv_train=mean_squared_error(y_train, y_pred_train)
r2_score_decision_tree_gridcv_train= r2_score(y_train, y_pred_train)
adjusted_r2_score_decision_tree_gridcv_train=1-(1-r2_score((y_train), (y_pred_train)))*((
print("mean_squared_error_decision_tree_gridcv_train:",mean_squared_error_decision_tree_g
print("r2_score_decision_tree_gridcv_train:",r2_score_decision_tree_gridcv_train)
print("adjusted_r2_score_decision_tree_gridcv_train:",adjusted_r2_score_decision_tree_gri
```

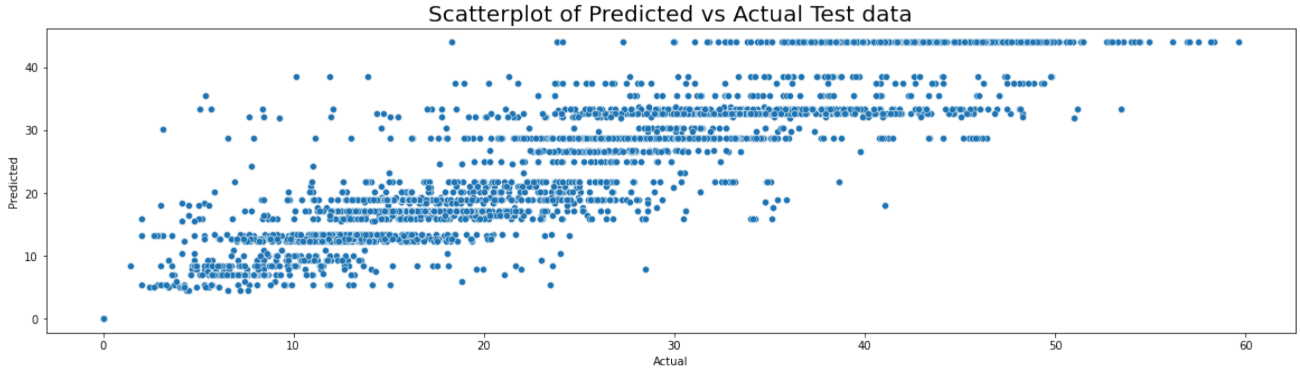
```
mean_squared_error_decision_tree_gridcv_train: 23.018425351508764
r2_score_decision_tree_gridcv_train: 0.8523024376521783
adjusted_r2_score_decision_tree_gridcv_train: 0.8513652382340906
```

```
#Evaluation for test set
mean_squared_error_decision_tree_gridcv_test=mean_squared_error(y_test, y_pred_test)
r2_score_decision_tree_gridcv_test= r2_score(y_test, y_pred_test)
adjusted_r2_score_decision_tree_gridcv_test=1-(1-r2_score((y_test), (y_pred_test)))*((X_t
print("mean_squared_error_decision_tree_gridcv_test:",mean_squared_error_decision_tree_gr
print("r2_score_decision_tree_gridcv_test:",r2_score_decision_tree_gridcv_test)
print("adjusted_r2_score_decision_tree_gridcv_test:",adjusted_r2_score_decision_tree_grid
```

```
mean_squared_error_decision_tree_gridcv_test: 30.21753469317526
r2_score_decision_tree_gridcv_test: 0.8028797937134317
adjusted_r2_score_decision_tree_gridcv_test: 0.8003233802424877
```

```
#Scatterplot of fitted vs Actual Test data
plt.figure(figsize = (20,5))
sns.scatterplot(x=y_test, y=y_pred_test)
plt.title('Scatterplot of Predicted vs Actual Test data',fontsize=20)
plt.ylabel('Predicted')
plt.xlabel('Actual')
```

```
Text(0.5, 0, 'Actual')
```



Random Forest

```
# Splitting the dataset into the Training set and Test set
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size = 0.2, random_state =
```

```
RF_Regressor = RandomForestRegressor()

# Fit the object to train dataset
RF_Regressor.fit(X_train, y_train)
```

```
RandomForestRegressor()
```

```
# Predicting the results
y_pred_train = RF_Regressor.predict(X_train)
y_pred_test =RF_Regressor.predict(X_test)
```

```
#Evaluation for train set
mean_squared_error_RF_Regressor_train=mean_squared_error(y_train, y_pred_train)
r2_score_RF_Regressor_train= r2_score(y_train, y_pred_train)
adjusted_r2_score_RF_Regressor_train=1-(1-r2_score((y_train), (y_pred_train)))*((X_train.
print("mean_squared_error_RF_Regressor_train:",mean_squared_error_RF_Regressor_train)
print("r2_score_RF_Regressor_train:",r2_score_RF_Regressor_train)
print("adjusted_r2_score_RF_Regressor_train:",adjusted_r2_score_RF_Regressor_train)
```

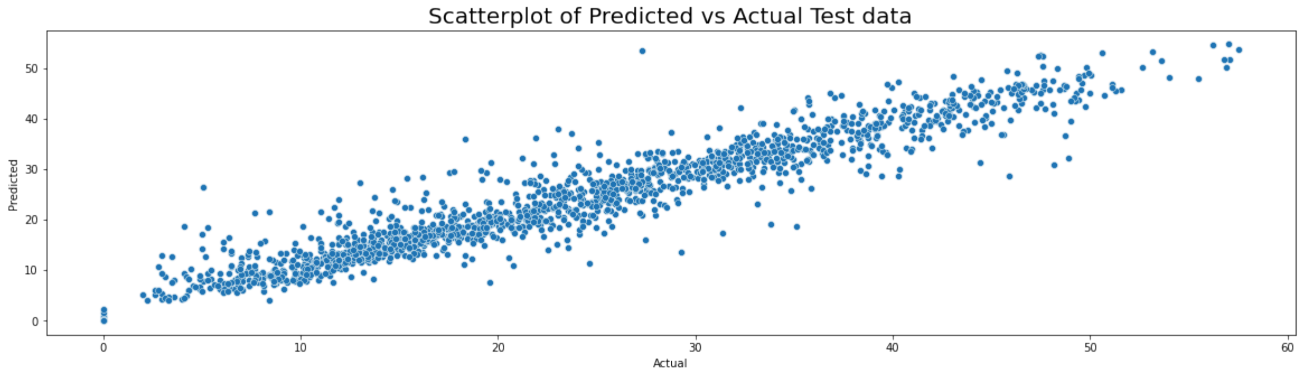
```
mean_squared_error_RF_Regressor_train: 1.7656482934114224
r2_score_RF_Regressor_train: 0.9885594986801821
adjusted_r2_score_RF_Regressor_train: 0.9884987671810669
```

```
#Evaluation for test set
mean_squared_error_RF_Regressor_test=mean_squared_error(y_test, y_pred_test)
r2_score_RF_Regressor_test= r2_score(y_test, y_pred_test)
adjusted_r2_score_RF_Regressor_test=1-(1-r2_score((y_test), (y_pred_test)))*((X_test.shap
print("mean_squared_error_RF_Regressor_test:",mean_squared_error_RF_Regressor_test)
print("r2_score_RF_Regressor_test:",r2_score_RF_Regressor_test)
print("adjusted_r2_score_RF_Regressor_test:",adjusted_r2_score_RF_Regressor_test)
```

```
mean_squared_error_RF_Regressor_test: 12.717787304570768
r2_score_RF_Regressor_test: 0.919245097746898
adjusted_r2_score_RF_Regressor_test: 0.917501847231516
```

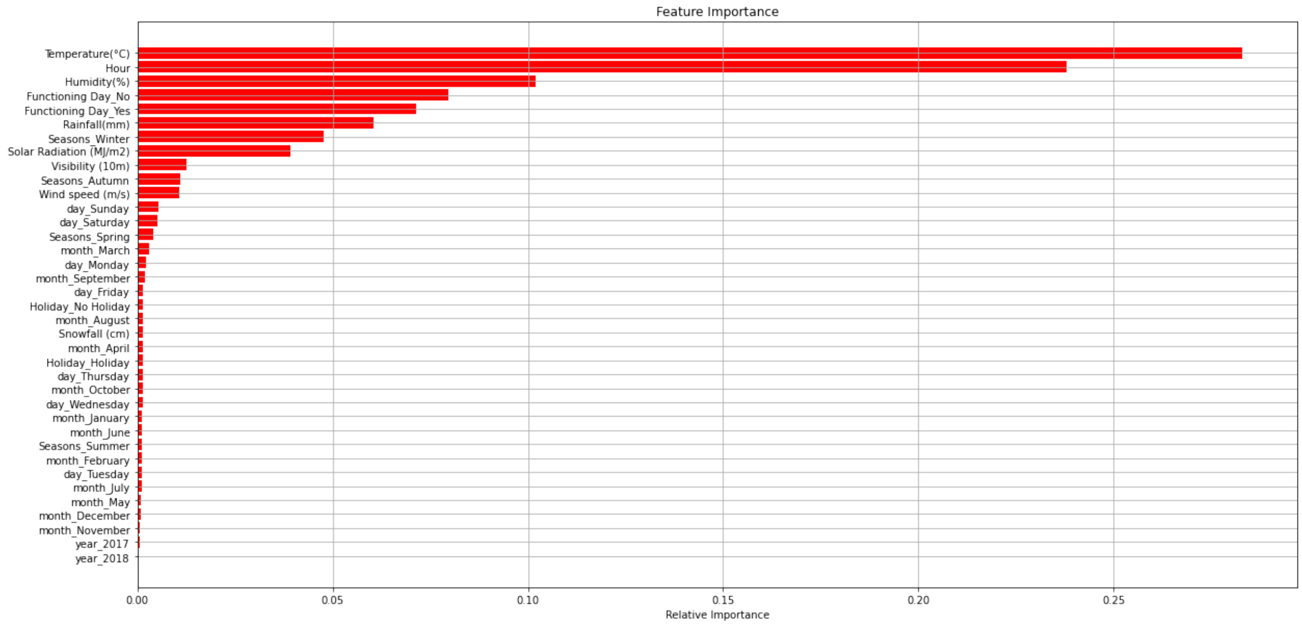
```
#Scatterplot of fitted vs Actual Test data
plt.figure(figsize = (20,5))
sns.scatterplot(x=y_test, y=y_pred_test)
plt.title('Scatterplot of Predicted vs Actual Test data',fontsize=20)
plt.ylabel('Predicted')
plt.xlabel('Actual')
```

```
Text(0.5, 0, 'Actual')
```




```
#storing features and there importance
features = X_train.columns
importances = RF_Regressor.feature_importances_
indices = np.argsort(importances)
```

```
#barh plot of features and there importance
plt.figure(figsize=(20,10))
plt.title('Feature Importance')
plt.barh(range(len(indices)), importances[indices], color='red', align='center')
plt.yticks(range(len(indices)), [features[i] for i in indices])
plt.xlabel('Relative Importance')
plt.grid()
plt.show()
```



Random Forest Using Grid Search CV

```
#Grid search CV
grid_values = {'n_estimators':[50, 80,  100], 'max_depth':[3, 5, 7]}
RF_Regressorr= GridSearchCV(RF_Regressor, param_grid = grid_values, cv=5)

# Fit the object to train dataset
RF_Regressorr.fit(X_train, y_train)
```

```
     GridSearchCV(cv=5, estimator=RandomForestRegressor(),
                  param_grid={'max_depth': [3, 5, 7], 'n_estimators': [50, 80, 100]})
```

```
# Predicting the results
y_pred_train = RF_Regressorr.predict(X_train)
y_pred_test =RF_Regressorr.predict(X_test)
```

```
#Evaluation for train set
mean_squared_error_RF_Regressor_gridcv_train=mean_squared_error(y_train, y_pred_train)
r2_score_RF_Regressor_gridcv_train= r2_score(y_train, y_pred_train)
adjusted_r2_score_RF_Regressor_gridcv_train=1-(1-r2_score((y_train), (y_pred_train)))*((X
print("mean_squared_error_RF_Regressor_gridcv_train:",mean_squared_error_RF_Regressor_gri
print("r2_score_RF_Regressor_gridcv_train:",r2_score_RF_Regressor_gridcv_train)
print("adjusted_r2_score_RF_Regressor_gridcv_train:",adjusted_r2_score_RF_Regressor_gridc
```

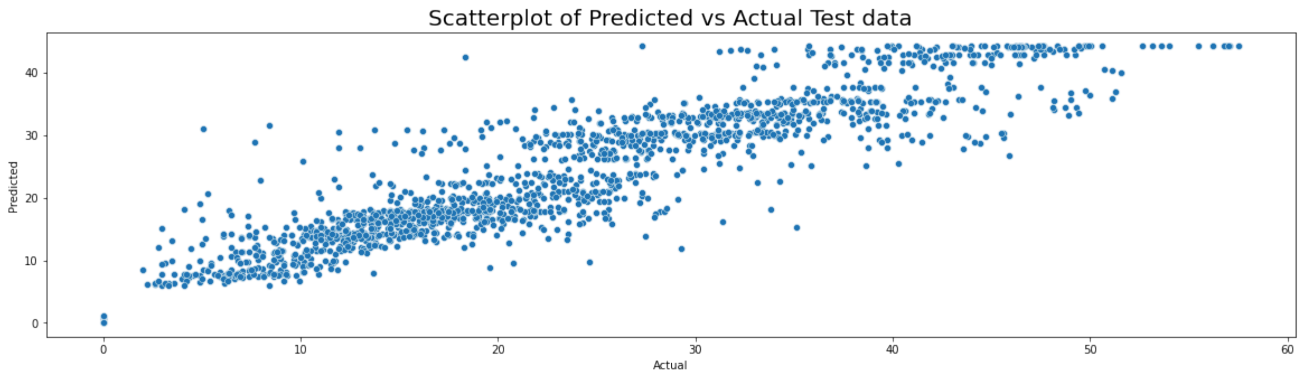
```
mean_squared_error_RF_Regressor_gridcv_train: 19.54390383485301
r2_score_RF_Regressor_gridcv_train: 0.8733654610312982
adjusted_r2_score_RF_Regressor_gridcv_train: 0.8726932260324687
```

```
#Evaluation for test set
mean_squared_error_RF_Regressor_gridcv_test=mean_squared_error(y_test, y_pred_test)
r2_score_RF_Regressor_gridcv_test= r2_score(y_test, y_pred_test)
adjusted_r2_score_RF_Regressor_gridcv_test=1-(1-r2_score((y_test), (y_pred_test)))*((X_te
print("mean_squared_error_RF_Regressor_gridcv_test:",mean_squared_error_RF_Regressor_grid
print("r2_score_RF_Regressor_gridcv_test:",r2_score_RF_Regressor_gridcv_test)
print("adjusted_r2_score_RF_Regressor_gridcv_test:",adjusted_r2_score_RF_Regressor_gridcv
```

```
mean_squared_error_RF_Regressor_gridcv_test: 23.103562647436284
r2_score_RF_Regressor_gridcv_test: 0.8532979127098945
adjusted_r2_score_RF_Regressor_gridcv_test: 0.8501310648512399
```

```
#Scatterplot of fitted vs Actual Test data
plt.figure(figsize = (20,5))
sns.scatterplot(x=y_test, y=y_pred_test)
plt.title('Scatterplot of Predicted vs Actual Test data',fontsize=20)
plt.ylabel('Predicted')
plt.xlabel('Actual')
```

```
Text(0.5, 0, 'Actual')
```



Gradient Boosting

```
# Splitting the dataset into the Training set and Test set
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size = 0.2, random_state =

# Standardize the dataset
sc = StandardScaler()
X_train_std = sc.fit_transform(X_train)
X_test_std = sc.transform(X_test)

# Hyperparameters for GradientBoostingRegressor

gbr_params = {'n_estimators': 1000,
          'max_depth': 3,
          'min_samples_split': 5,
          'learning_rate': 0.01,
          'loss': 'ls'}

# Create an instance of gradient boosting regressor

gbr = GradientBoostingRegressor(**gbr_params)

# Fit the model

gbr.fit(X_train_std, y_train)
```

```
GradientBoostingRegressor(learning_rate=0.01, loss='ls', min_samples_split=5,
                          n_estimators=1000)
```

```
# Predicting the results
y_pred_train = gbr.predict(X_train_std)
y_pred_test =gbr.predict(X_test_std)
```

```
#Evaluation for train set
mean_squared_error_gbr_train=mean_squared_error(y_train, y_pred_train)
r2_score_gbr_train= r2_score(y_train, y_pred_train)
adjusted_r2_score_gbr_train=1-(1-r2_score((y_train), (y_pred_train)))*((X_train_std.shape
print("mean_squared_error_gbr_train:",mean_squared_error_gbr_train)
print("r2_score_gbr_train:",r2_score_gbr_train)
print("adjusted_r2_score_gbr_train:",adjusted_r2_score_gbr_train)
```

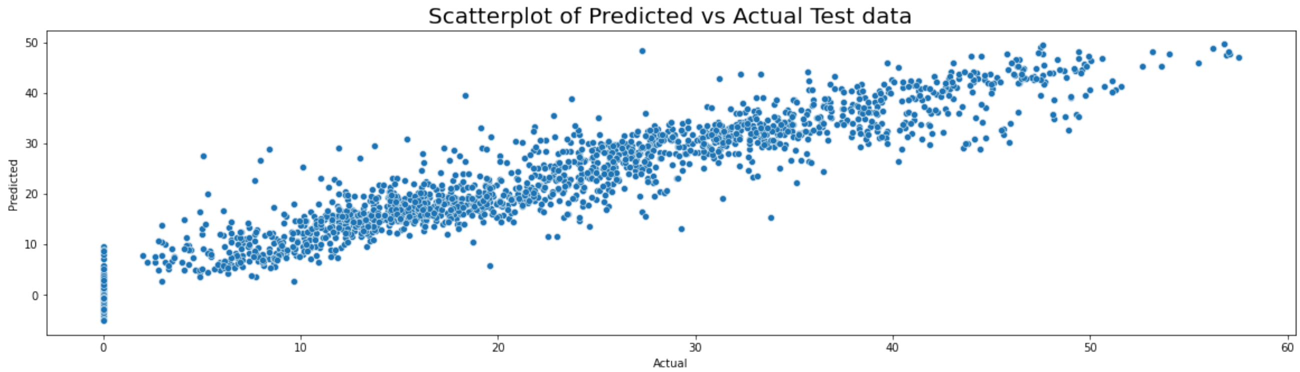
```
mean_squared_error_gbr_train: 16.73941988001001
r2_score_gbr_train: 0.8915370881364894
adjusted_r2_score_gbr_train: 0.8909613165814033
```

```
#Evaluation for test set
mean_squared_error_gbr_test=mean_squared_error(y_test, y_pred_test)
r2_score_gbr_test= r2_score(y_test, y_pred_test)
adjusted_r2_score_gbr_test=1-(1-r2_score((y_test), (y_pred_test)))*((X_test_std.shape[0]-
print("mean_squared_error_gbr_test:",mean_squared_error_gbr_test)
print("r2_score_gbr_test:",r2_score_gbr_test)
print("adjusted_r2_score_gbr_test:",adjusted_r2_score_gbr_test)
```

```
mean_squared_error_gbr_test: 19.364128692224796
r2_score_gbr_test: 0.8770424223720827
adjusted_r2_score_gbr_test: 0.8743881456088196
```

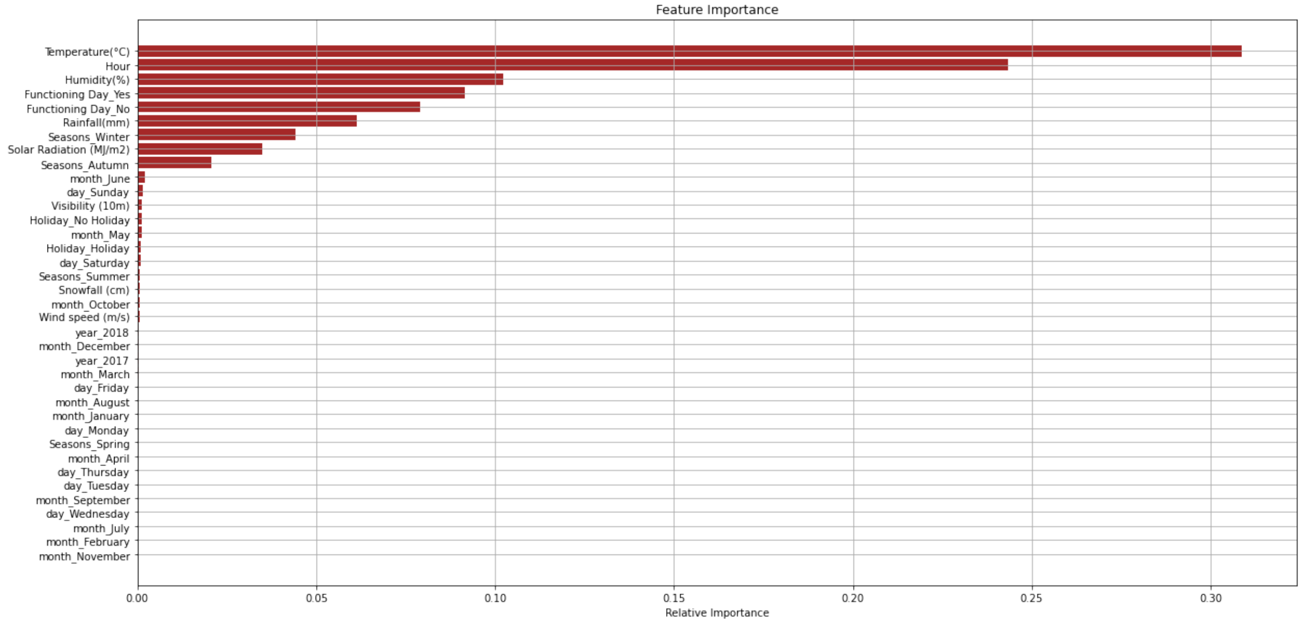
```
#Scatterplot of fitted vs Actual Test data
plt.figure(figsize = (20,5))
sns.scatterplot(x=y_test, y=y_pred_test)
plt.title('Scatterplot of Predicted vs Actual Test data',fontsize=20)
plt.ylabel('Predicted')
plt.xlabel('Actual')
```

```
Text(0.5, 0, 'Actual')
```




```
#storing features and there importance
features = X_train.columns
importances = gbr.feature_importances_
indices = np.argsort(importances)
```

```
#barh plot of features and there importance
plt.figure(figsize=(20,10))
plt.title('Feature Importance')
plt.barh(range(len(indices)), importances[indices], color='brown', align='center')
plt.yticks(range(len(indices)), [features[i] for i in indices])
plt.xlabel('Relative Importance')
plt.grid()
plt.show()
```



Gradient Boosting using Grid Search CV

```
#Grid Search CV
grid_values = {'n_estimators':[50, 80,  100], 'max_depth':[3, 5, 7]}
gbrr = GridSearchCV(gbr, param_grid = grid_values, cv=5)

# Fit the object to train dataset
gbrr.fit(X_train_std, y_train)
```

```
GridSearchCV(cv=5,
             estimator=GradientBoostingRegressor(learning_rate=0.01, loss='ls',
                                                 min_samples_split=5,
                                                 n_estimators=1000),
             param_grid={'max_depth': [3, 5, 7], 'n_estimators': [50, 80, 100]})
```

```
# Predicting the results
y_pred_train = gbrr.predict(X_train_std)
y_pred_test =gbrr.predict(X_test_std)
```

```
#Evaluation for train set
mean_squared_error_gbr_gridcv_train=mean_squared_error(y_train, y_pred_train)
r2_score_gbr_gridcv_train= r2_score(y_train, y_pred_train)
adjusted_r2_score_gbr_gridcv_train=1-(1-r2_score((y_train), (y_pred_train)))*((X_train_st
print("mean_squared_error_gbr_gridcv_train:",mean_squared_error_gbr_gridcv_train)
print("r2_score_gbr_gridcv_train:",r2_score_gbr_gridcv_train)
print("adjusted_r2_score_gbr_gridcv_train:",adjusted_r2_score_gbr_gridcv_train)
```

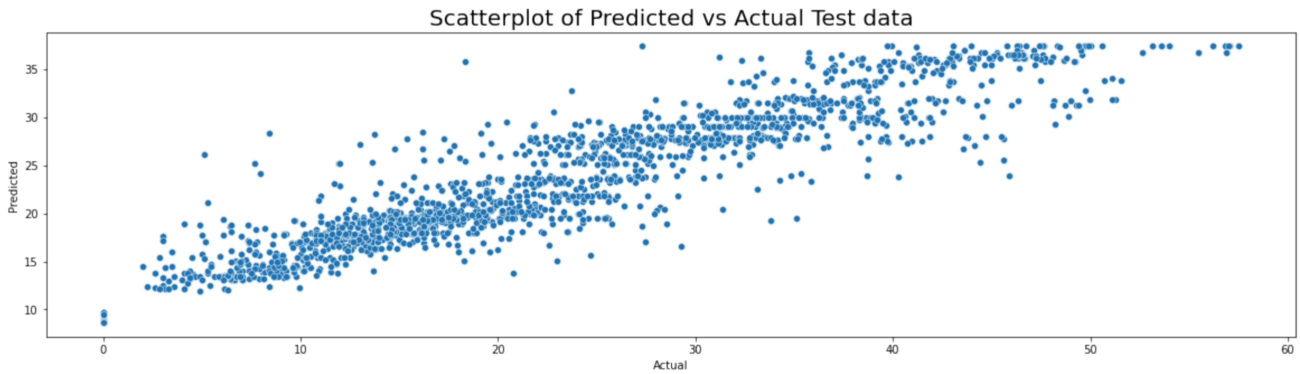
```
mean_squared_error_gbr_gridcv_train: 37.35238557918996
r2_score_gbr_gridcv_train: 0.757975572988308
adjusted_r2_score_gbr_gridcv_train: 0.7566907948248313
```

```
#Evaluation for test set
mean_squared_error_gbr_gridcv_test=mean_squared_error(y_test, y_pred_test)
r2_score_gbr_gridcv_test= r2_score(y_test, y_pred_test)
adjusted_r2_score_gbr_gridcv_test=1-(1-r2_score((y_test), (y_pred_test)))*((X_test_std.sh
print("mean_squared_error_gbr_gridcv_test:",mean_squared_error_gbr_gridcv_test)
print("r2_score_gbr_gridcv_test:",r2_score_gbr_gridcv_test)
print("adjusted_r2_score_gbr_gridcv_test:",adjusted_r2_score_gbr_gridcv_test)
```

```
mean_squared_error_gbr_gridcv_test: 41.015653608175995
r2_score_gbr_gridcv_test: 0.7395604267744804
adjusted_r2_score_gbr_gridcv_test: 0.7339383356371734
```

```
#Scatterplot of fitted vs Actual Test data
plt.figure(figsize = (20,5))
sns.scatterplot(x=y_test, y=y_pred_test)
plt.title('Scatterplot of Predicted vs Actual Test data',fontsize=20)
plt.ylabel('Predicted')
plt.xlabel('Actual')
```

```
Text(0.5, 0, 'Actual')
```



```
# Splitting
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size = 0.2, random_state =

# Instantiation
xgb_r = xg.XGBRegressor(objective ='reg:linear', seed = 123)


# Fitting the model
xgb_r.fit(X_train, y_train)
```

```
[08:51:05] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now
XGBRegressor(seed=123)
```