Project Type - Regression

Contribution - Individual

Team Member- Madhumoy Shaw

Project Title : Seoul Bike Sharing Demand Prediction

GitHub Link -

https://github.com/MADHUMOYSHAW/MACHINE-LEARNING-1

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)

Import Libraries and Data

```
#let's import the modules
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns

from datetime import datetime
import datetime as dt

from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import OneHotEncoder
from sklearn.preprocessing import MultiLabelBinarizer

from sklearn.linear_model import LinearRegression
from sklearn.linear_model import Lasso
from sklearn.linear_model import Ridge
from sklearn.linear_model import ElasticNet
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.ensemble import GradientBoostingRegressor
```

```
from sklearn.model_selection import cross_validate
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import RepeatedStratifiedKFold
from sklearn.model_selection import RandomizedSearchCV

from sklearn import metrics
from sklearn.metrics import r2_score
from sklearn.metrics import mean_squared_error
from sklearn.metrics import accuracy_score
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import log_loss

import warnings
warnings.filterwarnings('ignore')
```

Mount the drive and import the datset

```
#let's mount the google drive for import the dtaset
from google.colab import drive
drive.mount('/content/drive')
```

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

```
#load the seol bike data set from drive
dataset = pd.read_csv('/content/drive/MyDrive/Colab Notebooks/MACHINE LEARNING CHAPTER 1/CAPS
```

Understand More About The Data

summary of data

```
# Viewing the data of top 5 rows to take a glimps of the data
dataset.head()
```

	Date	Rented Bike Count	Hour	Temperature(°C)	Humidity(%)	Wind speed (m/s)	Visibility (10m)	Dew p temperature
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	

```
#Getting the shape of dataset with rows and columns
print(dataset.shape)
```

```
(8760, 14)
```

```
#Getting all the columns
print("Features of the dataset:")
dataset.columns
```

```
Features of the dataset:
Index(['Date', 'Rented Bike Count', 'Hour', 'Temperature(°C)', 'Humidity(%)',
       'Wind speed (m/s)', 'Visibility (10m)', 'Dew point temperature(°C)',
       'Solar Radiation (MJ/m2)', 'Rainfall(mm)', 'Snowfall (cm)', 'Seasons',
       'Holiday', 'Functioning Day'],
      dtype='object')
```

```
#check details about the data set
dataset.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
```

```
#print the unique value
dataset.nunique()
```

```
Date                          365
Rented Bike Count            2166
Hour                           24
Temperature(°C)               546
Humidity(%)                    90
Wind speed (m/s)               65
Visibility (10m)             1789
Dew point temperature(°C)     556
Solar Radiation (MJ/m2)       345
Rainfall(mm)                   61
Snowfall (cm)                  51
Seasons                         4
Holiday                         2
Functioning Day                 2
dtype: int64
```

```
#Looking for the description of the dataset to get insights of the data
dataset.describe(include='all').T
```

	count	unique	top	freq	mean	std	min	25%	
Date	8760	365	01/12/2017	24	NaN	NaN	NaN	NaN	I
Rented Bike Count	8760.0	NaN	NaN	NaN	704.602055	644.997468	0.0	191.0	5(
Hour	8760.0	NaN	NaN	NaN	11.5	6.922582	0.0	5.75	
Temperature(°C)	8760.0	NaN	NaN	NaN	12.882922	11.944825	-17.8	3.5	
Humidity(%)	8760.0	NaN	NaN	NaN	58.226256	20.362413	0.0	42.0	5
Wind speed (m/s)	8760.0	NaN	NaN	NaN	1.724909	1.0363	0.0	0.9	
Visibility (10m)	8760.0	NaN	NaN	NaN	1436.825799	608.298712	27.0	940.0	16
Dew point temperature(°C)	8760.0	NaN	NaN	NaN	4.073813	13.060369	-30.6	-4.7	
Solar Radiation (MJ/m2)	8760.0	NaN	NaN	NaN	0.569111	0.868746	0.0	0.0	(
Rainfall(mm)	8760.0	NaN	NaN	NaN	0.148687	1.128193	0.0	0.0	
Snowfall (cm)	8760.0	NaN	NaN	NaN	0.075068	0.436746	0.0	0.0	
S	8760	4	S	2208	NaN	NaN	NaN	NaN	I

- ***This Dataset contains 8760 lines and 14 columns.***
- **In a day we have 24 hours and we have 365 days a year so 365 multiplied by 24 = 8760, which represents the number of line in the dataset.***

Features description

Breakdown of Our Features:

Date : *The date of the day, during 365 days from 01/12/2017 to 30/11/2018, formating in DD/MM/YYYY, type : str*, we need to convert into datetime format.

Rented Bike Count : *Number of rented bikes per hour which our dependent variable and we need to predict that, type : int*

Hour: *The hour of the day, starting from 0-23 it's in a digital time format, type : int, we need to convert it into category data type.*

Temperature(°C): *Temperature in Celsius, type : Float*

Humidity(%): *Humidity in the air in %, type : int*

Wind speed (m/s) : *Speed of the wind in m/s, type : Float*

Visibility (10m): *Visibility in m, type : int*

Dew point temperature(°C): *Temperature at the beggining of the day, type : Float*

Solar Radiation (MJ/m2): *Sun contribution, type : Float*

Rainfall(mm): *Amount of raining in mm, type : Float*

Snowfall (cm): *Amount of snowing in cm, type : Float*

Seasons: *Season of the year, type : str, there are only 4 season's in data *.

Holiday: *If the day is holiday period or not, type: str*

Functioning Day: *If the day is a Functioning Day or not, type : str*

Preprocessing the dataset

Why do we need to handle missing values?

- ***The real-world data often has a lot of missing values. The cause of missing values can be data corruption or failure to record data. The handling of missing data is very important during the***

preprocessing of the dataset as many machine learning algorithms do not support missing values.that's why we check missing values first

Missing values

```
#check for count of missing values in each column.
dataset.isna().sum()
dataset.isnull().sum()
```

```
Date                       0
Rented Bike Count          0
Hour                       0
Temperature(°C)            0
Humidity(%)                0
Wind speed (m/s)           0
Visibility (10m)           0
Dew point temperature(°C)  0
Solar Radiation (MJ/m2)    0
Rainfall(mm)               0
Snowfall (cm)              0
Seasons                    0
Holiday                    0
Functioning Day            0
dtype: int64
```

- ***As we can see above there are no missing value presents thankfully***

Duplicate values

Why is it important to remove duplicate records from my data?

- "Duplication" just means that you have repeated data in your dataset. This could be due to things like data entry errors or data collection methods. by removing duplication in our data set, Time and money are saved by not sending identical communications multiple times to the same person.***

```
# Checking Duplicate Values
value=len(dataset[dataset.duplicated()])
print("The number of duplicate values in the data set is = ",value)
```

```
The number of duplicate values in the data set is =  0
```

- ***In the above data after count the missing and duplicate value we came to know that there are no missing and duplicate value present.***

- ***Some of the columns name in the dataset are too large ,complex and clumsy so we change the the into some simple name, and it don't affect our end results.***

Changing column name

```
#Rename the complex columns name
dataset=dataset.rename(columns={'Rented Bike Count':'Rented_Bike_Count',
                                'Temperature(°C)':'Temperature',
                                'Humidity(%)':'Humidity',
                                'Wind speed (m/s)':'Wind_speed',
                                'Visibility (10m)':'Visibility',
                                'Dew point temperature(°C)':'Dew_point_temperature',
                                'Solar Radiation (MJ/m2)':'Solar_Radiation',
                                'Rainfall(mm)':'Rainfall',
                                'Snowfall (cm)':'Snowfall',
                                'Functioning Day':'Functioning_Day'})
```

```
numeric_features = dataset.describe().columns
numeric_features
```

```
Index(['Rented_Bike_Count', 'Hour', 'Temperature', 'Humidity', 'Wind_speed',
       'Visibility', 'Dew_point_temperature', 'Solar_Radiation', 'Rainfall',
       'Snowfall'],
      dtype='object')
```

- ***Python read "Date" column as a object type basically it reads as a string, as the date column is very important to analyze the users behaviour so we need to convert it into datetime format then we split it into 3 column i.e 'year', 'month', 'day'as a category data type.***

Breaking date column

```
# Changing the "Date" column into three "year","month","day" column
dataset['Date'] = dataset['Date'].apply(lambda x:
                                        dt.datetime.strptime(x,"%d/%m/%Y"))
```

```
dataset['year'] = dataset['Date'].dt.year
dataset['month'] = dataset['Date'].dt.month
dataset['day'] = dataset['Date'].dt.day_name()
```

```
dataset.head(1)
```

	Date	Rented_Bike_Count	Hour	Temperature	Humidity	Wind_speed	Visibility	Dew_po
0	2017-12-01	254	0	-5.2	37	2.2	2000	

```
#creating a new column of "weekdays_weekend" and drop the column "Date","day","year"
dataset['weekdays_weekend']=dataset['day'].apply(lambda x : 1 if x=='Saturday' or x=='Sunday'
dataset=dataset.drop(columns=['Date','day','year'],axis=1)
```

- ***So we convert the "date" column into 3 different column i.e "year","month","day".***
- ***The "year" column in our data set is basically contain the 2 unique number contains the details of from 2017 december to 2018 november so if i consider this is a one year then we don't need the "year" column so we drop it***.
- ***The other column "day", it contains the details about the each day of the month, for our relevence we don't need each day of each month data but we need the data about, if a day is a weekday or a weekend so we convert it into this format and drop the "day" column***.

```
dataset.head(2)
```

	Rented_Bike_Count	Hour	Temperature	Humidity	Wind_speed	Visibility	Dew_point_tem
0	254	0	-5.2	37	2.2	2000	
1	204	1	-5.5	38	0.8	2000	

```
dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8760 entries, 0 to 8759
Data columns (total 15 columns):
 #   Column                 Non-Null Count  Dtype
```

```
---  ------                  --------------  -----
 0   Rented_Bike_Count       8760 non-null   int64
 1   Hour                    8760 non-null   int64
 2   Temperature             8760 non-null   float64
 3   Humidity                8760 non-null   int64
 4   Wind_speed              8760 non-null   float64
 5   Visibility              8760 non-null   int64
 6   Dew_point_temperature   8760 non-null   float64
 7   Solar_Radiation         8760 non-null   float64
 8   Rainfall                8760 non-null   float64
 9   Snowfall                8760 non-null   float64
 10  Seasons                 8760 non-null   object
 11  Holiday                 8760 non-null   object
 12  Functioning_Day         8760 non-null   object
 13  month                   8760 non-null   int64
 14  weekdays_weekend        8760 non-null   int64
dtypes: float64(6), int64(6), object(3)
memory usage: 1.0+ MB
```

```
dataset['weekdays_weekend'].value_counts()
```

```
0    6264
1    2496
Name: weekdays_weekend, dtype: int64
```

Changing data type

- ***As "Hour","month","weekdays_weekend" column are show as a integer data type but actually it is a category data tyepe. so we need to change this data tyepe if we not then, while doing the further anlysis and correleted with this then the values are not actually true so we can mislead by this.***

```
dataset.nunique()
```

```
Rented_Bike_Count        2166
Hour                       24
Temperature               546
Humidity                   90
Wind_speed                 65
Visibility               1789
Dew_point_temperature     556
Solar_Radiation           345
Rainfall                   61
Snowfall                   51
Seasons                     4
Holiday                     2
Functioning_Day             2
month                      12
```

```
weekdays_weekend            2
dtype: int64
```

```
#Change the int64 column into catagory column
cols=['Hour','month','weekdays_weekend']
for col in cols:
  dataset[col]=dataset[col].astype('category')
```

```
#let's check the result of data type
dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8760 entries, 0 to 8759
Data columns (total 15 columns):
 #   Column                 Non-Null Count  Dtype   
---  ------                 --------------  -----   
 0   Rented_Bike_Count      8760 non-null   int64   
 1   Hour                   8760 non-null   category
 2   Temperature            8760 non-null   float64 
 3   Humidity               8760 non-null   int64   
 4   Wind_speed             8760 non-null   float64 
 5   Visibility             8760 non-null   int64   
 6   Dew_point_temperature  8760 non-null   float64 
 7   Solar_Radiation        8760 non-null   float64 
 8   Rainfall               8760 non-null   float64 
 9   Snowfall               8760 non-null   float64 
 10  Seasons                8760 non-null   object  
 11  Holiday                8760 non-null   object  
 12  Functioning_Day        8760 non-null   object  
 13  month                  8760 non-null   category
 14  weekdays_weekend       8760 non-null   category
dtypes: category(3), float64(6), int64(3), object(3)
memory usage: 848.3+ KB
```

```
dataset.columns
```

```
Index(['Rented_Bike_Count', 'Hour', 'Temperature', 'Humidity', 'Wind_speed',
       'Visibility', 'Dew_point_temperature', 'Solar_Radiation', 'Rainfall',
       'Snowfall', 'Seasons', 'Holiday', 'Functioning_Day', 'month',
       'weekdays_weekend'],
      dtype='object')
```

```
dataset['weekdays_weekend'].unique()
```

```
[0, 1]
Categories (2, int64): [0, 1]
```

Exploratory Data Analysis Of The Data Set

Why do we perform EDA?

- ***An EDA is a thorough examination meant to uncover the underlying structure of a data set and is important for a company because it exposes trends, patterns, and relationships that are not readily apparent.***

Univariate Analysis

Why do you do univariate analysis?

- ***The key objective of Univariate analysis is to simply describe the data to find patterns within the data.***

Analysis of Dependent Variable:

What is a dependent variable in data analysis?

- ***we analyse our dependent variable,A dependent variable is a variable whose value will change depending on the value of another variable.***

Analysation of categorical variables

- ***Our dependent variable is "Rented Bike Count" so we need to analysis this column with the other columns by using some visualisation plot.first we analyze the category data tyep then we proceed with the numerical data type***

```
dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8760 entries, 0 to 8759
Data columns (total 15 columns):
 #   Column                 Non-Null Count  Dtype   
---  ------                 --------------  -----   
 0   Rented_Bike_Count      8760 non-null   int64   
 1   Hour                   8760 non-null   category
 2   Temperature            8760 non-null   float64 
 3   Humidity               8760 non-null   int64   
 4   Wind_speed             8760 non-null   float64 
 5   Visibility             8760 non-null   int64   
 6   Dew_point_temperature  8760 non-null   float64 
 7   Solar_Radiation        8760 non-null   float64 
```

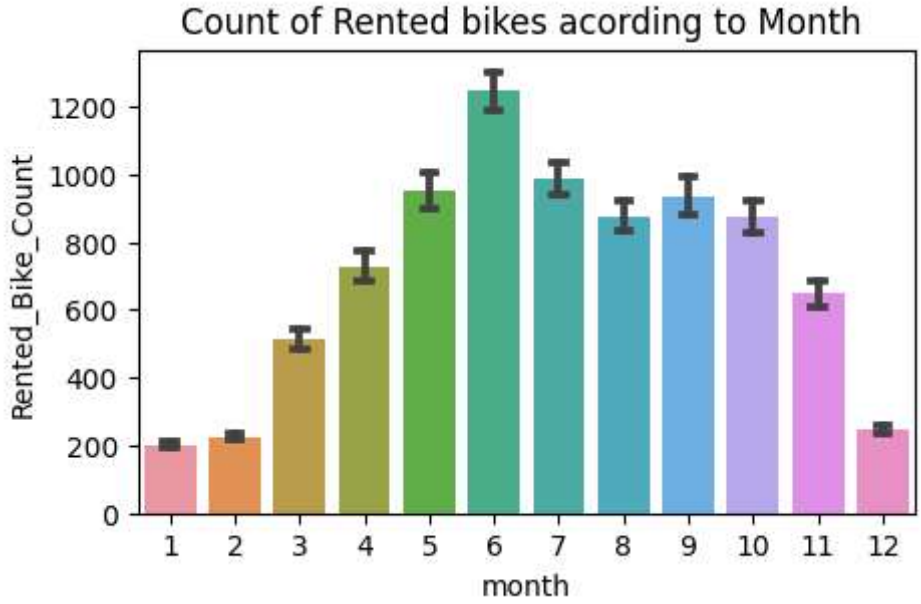
```
 8   Rainfall               8760 non-null   float64
 9   Snowfall               8760 non-null   float64
 10  Seasons                8760 non-null   object
 11  Holiday                8760 non-null   object
 12  Functioning_Day        8760 non-null   object
 13  month                  8760 non-null   category
 14  weekdays_weekend       8760 non-null   category
dtypes: category(3), float64(6), int64(3), object(3)
memory usage: 848.3+ KB
```

Month

CHART 1

```
#anlysis of data by vizualisation
fig,ax=plt.subplots(figsize=(5,3))
sns.barplot(data=dataset,x='month',y='Rented_Bike_Count',ax=ax,capsize=.2)
ax.set(title='Count of Rented bikes acording to Month ')
```

```
[Text(0.5, 1.0, 'Count of Rented bikes acording to Month ')]
```



1. Why did you pick the specific chart?

to see the relationship between rented bike count with respect to month.

2. What is/are the insight(s) found from the chart?

From the above bar plot we can clearly say that from the month 5 to 10 the demand of the rented bike is high as compare to other months.these months are comes inside the summer season.

3. Will the gained insights help creating a positive business impact?

Are there any insights that lead to negative growth? Justify with specific reason.

Yes.It will help to gain insight to help creating a positive business impact.

weekdays_weekend

```
#anlysis of data by vizualisation
fig,ax=plt.subplots(figsize=(5,3))
sns.barplot(data=dataset,x='weekdays_weekend',y='Rented_Bike_Count',ax=ax,capsize=.2)
ax.set(title='Count of Rented bikes acording to weekdays and weekend ')
```

```
[Text(0.5, 1.0, 'Count of Rented bikes acording to weekdays and weekend ')]
```

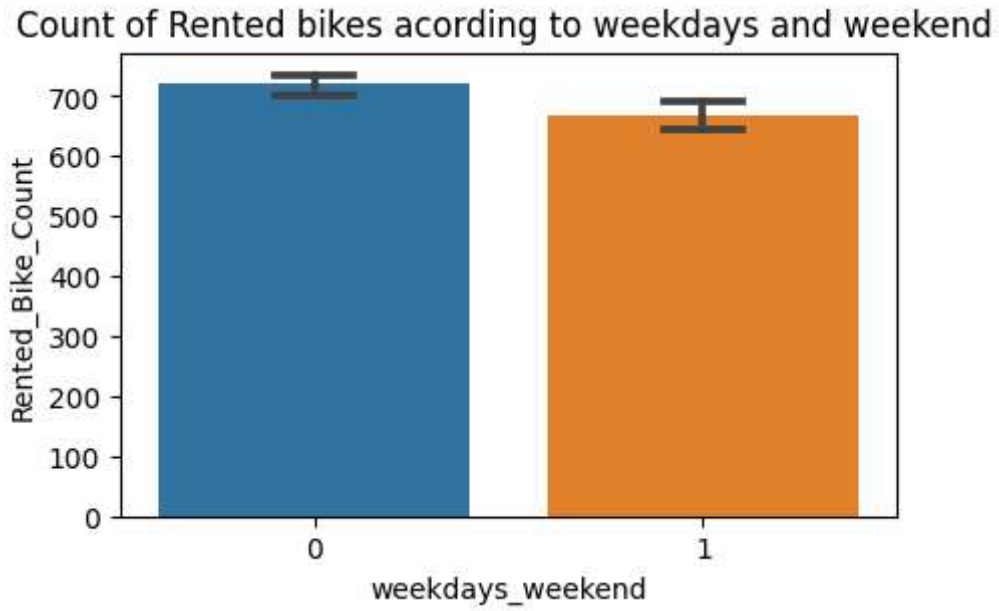
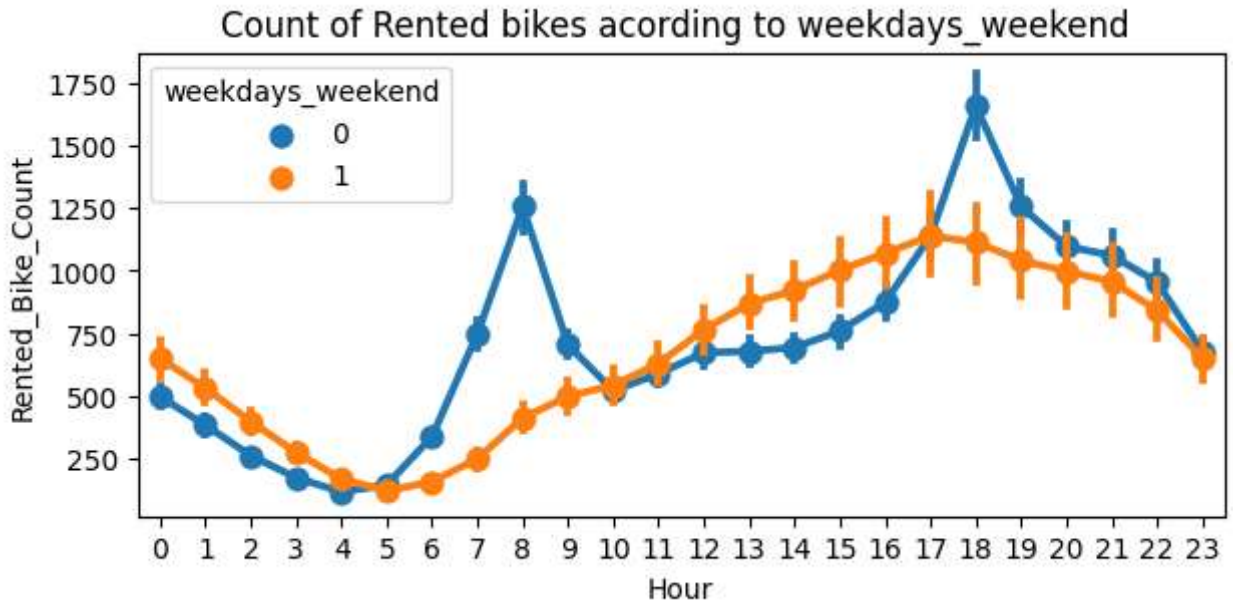


CHART 2

```
#anlysis of data by vizualisation
fig,ax=plt.subplots(figsize=(7,3))
sns.pointplot(data=dataset,x='Hour',y='Rented_Bike_Count',hue='weekdays_weekend',ax=ax)
ax.set(title='Count of Rented bikes acording to weekdays_weekend ')
```

```
[Text(0.5, 1.0, 'Count of Rented bikes acording to weekdays_weekend ')]
```



1. Why did you pick the specific chart?

to see the relationship between rented bike count and hour with respect to weekdays_weekend.

2. What is/are the insight(s) found from the chart?

- ***From the above point plot and bar plot we can say that in the week days which represent in blue colur show that the demand of the bike higher because of the office.***
- **Peak Time are 7 am to 9 am and 5 pm to 7 pm**.
- **The orange colur represent the weekend days, and it show that the demand of rented bikes are very low specially in the morning hour but when the evening start from 4 pm to 8 pm the demand slightly increases.***

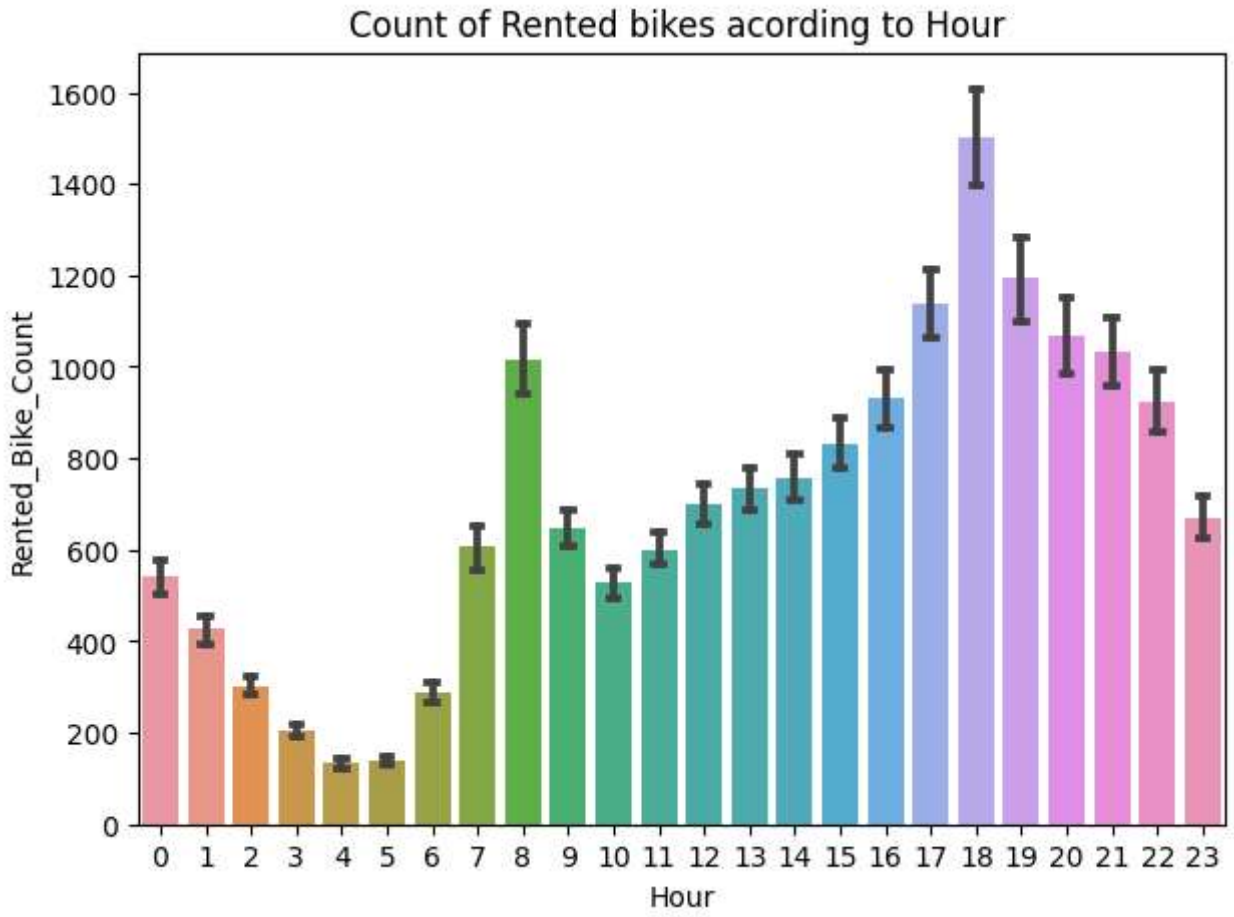
3. Will the gained insights help creating a positive business impact?

Are there any insights that lead to negative growth? Justify with specific reason.

Yes.It will help to gain insight to help creating a positive business impact.

```
#anlysis of data by vizualisation
fig,ax=plt.subplots(figsize=(7,5))
sns.barplot(data=dataset,x='Hour',y='Rented_Bike_Count',ax=ax,capsize=.2)
ax.set(title='Count of Rented bikes acording to Hour ')
```

```
[Text(0.5, 1.0, 'Count of Rented bikes acording to Hour ')]
```



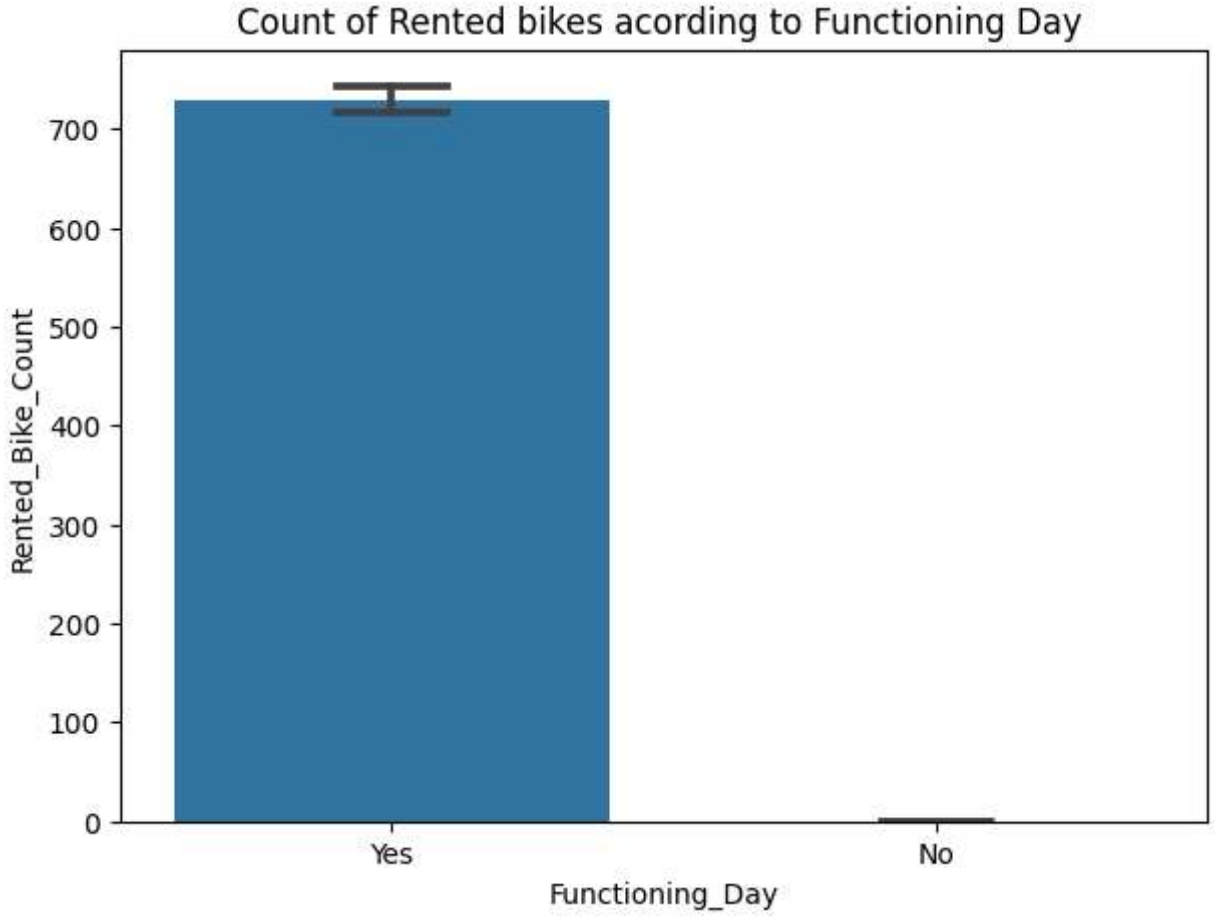
Hour

- ***In the above plot which shows the use of rented bike according the hours and the data are from all over the year.***
- ***generally people use rented bikes during their working hour from 7am to 9am and 5pm to 7pm.***

Functioning Day

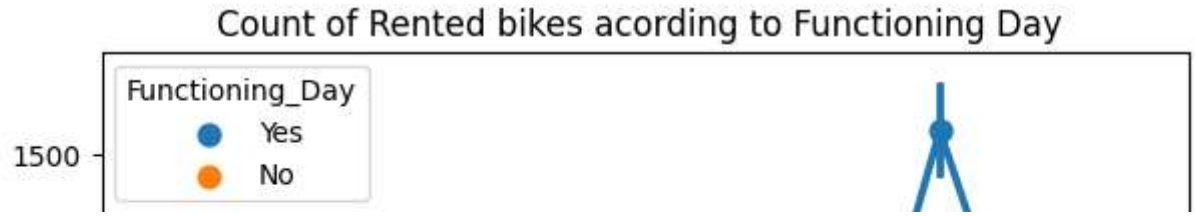
```
#anlysis of data by vizualisation
fig,ax=plt.subplots(figsize=(7,5))
sns.barplot(data=dataset,x='Functioning_Day',y='Rented_Bike_Count',ax=ax,capsize=.2)
ax.set(title='Count of Rented bikes acording to Functioning Day ')
```

```
[Text(0.5, 1.0, 'Count of Rented bikes acording to Functioning Day ')]
```




```
#anlysis of data by vizualisation
fig,ax=plt.subplots(figsize=(7,5))
sns.pointplot(data=dataset,x='Hour',y='Rented_Bike_Count',hue='Functioning_Day',ax=ax)
ax.set(title='Count of Rented bikes acording to Functioning Day ')
```

```
[Text(0.5, 1.0, 'Count of Rented bikes acording to Functioning Day ')]
```



- ***In the above bar plot and point plot which shows the use of rented bike in functioning daya or not, and it clearly shows that,***
- ***Peoples dont use reneted bikes in no functioning day.***

Seasons

```
#anlysis of data by vizualisation
fig,ax=plt.subplots(figsize=(7,5))
sns.barplot(data=dataset,x='Seasons',y='Rented_Bike_Count',ax=ax,capsize=.2)
ax.set(title='Count of Rented bikes acording to Seasons ')
```

```
[Text(0.5, 1.0, 'Count of Rented bikes acording to Seasons ')]
```

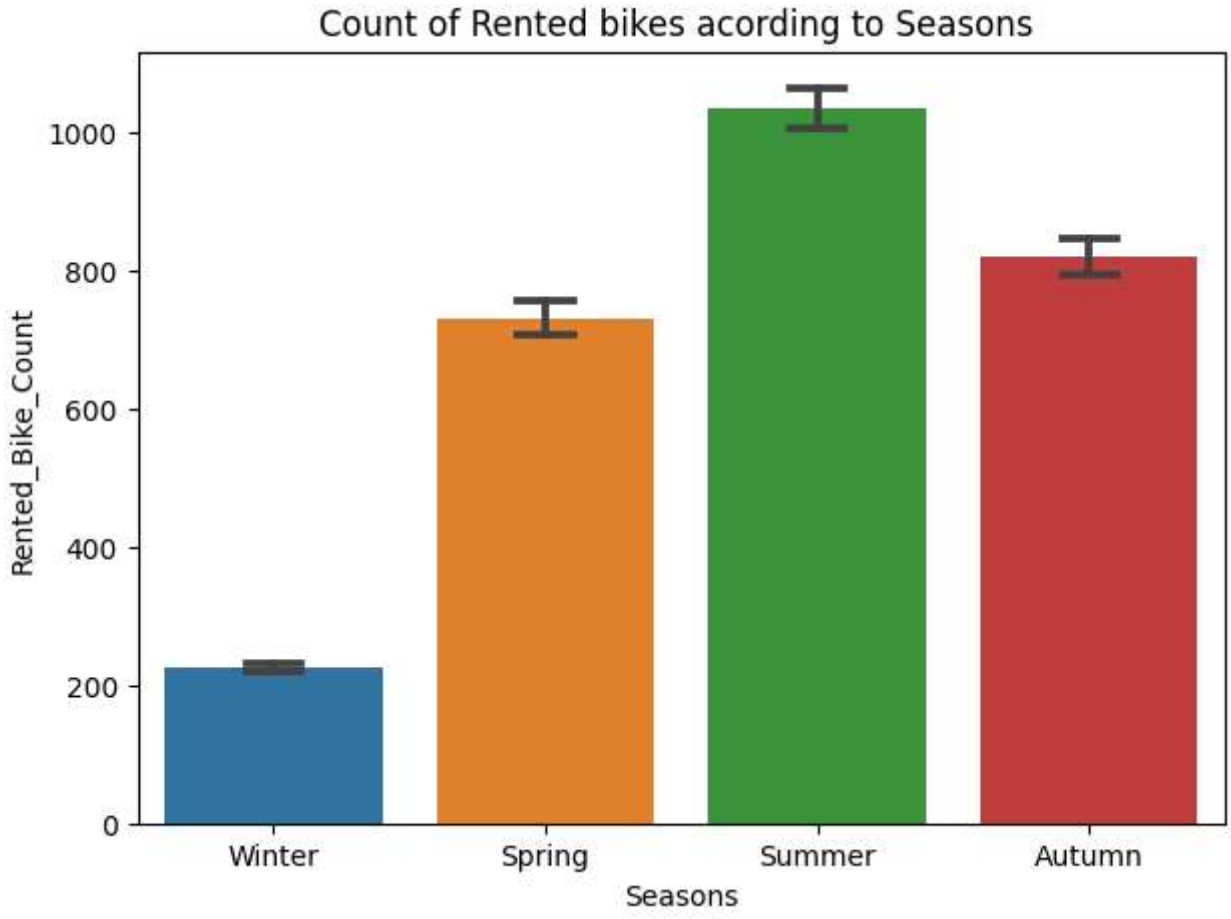
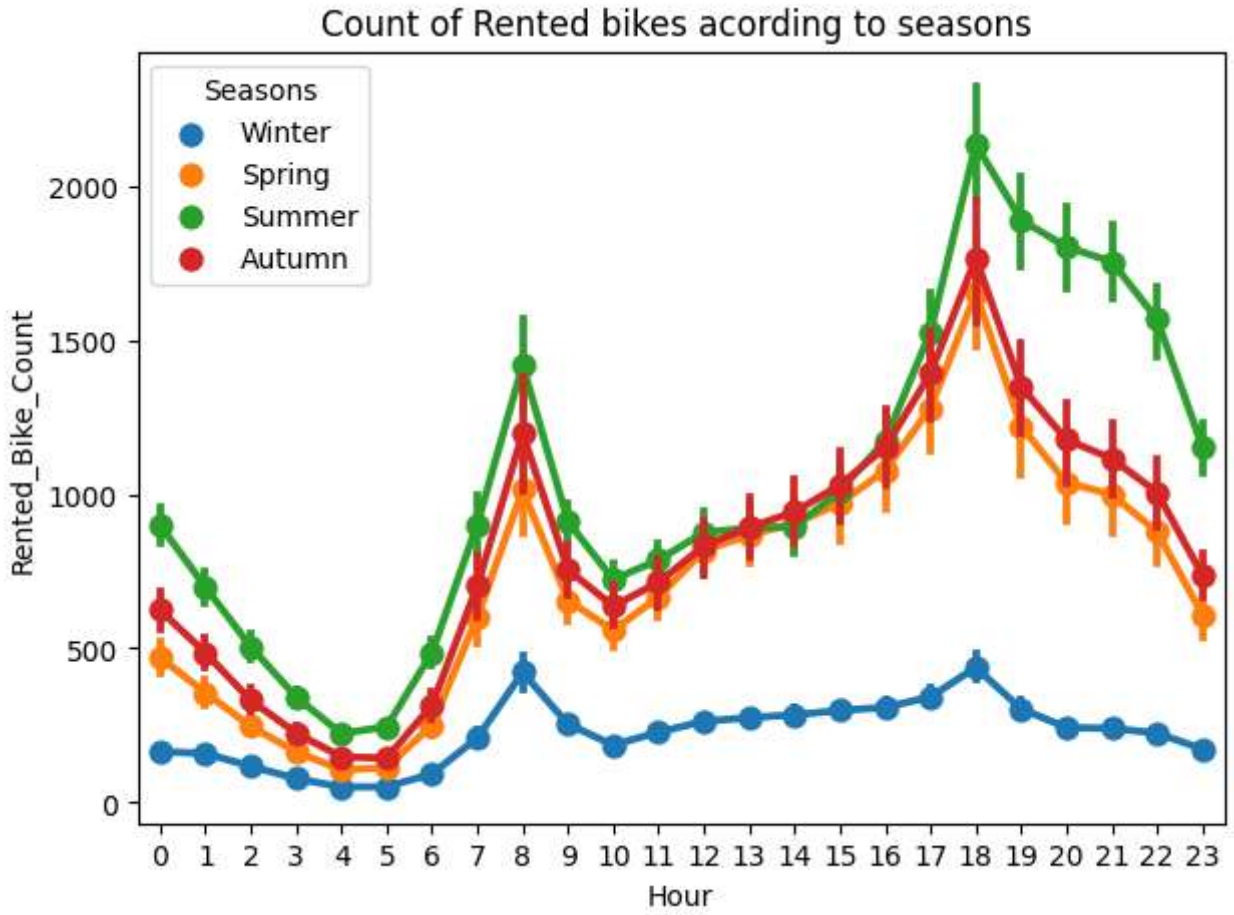


CHART 3

```
#anlysis of data by vizualisation
fig,ax=plt.subplots(figsize=(7,5))
sns.pointplot(data=dataset,x='Hour',y='Rented_Bike_Count',hue='Seasons',ax=ax)
ax.set(title='Count of Rented bikes acording to seasons ')
```

```
[Text(0.5, 1.0, 'Count of Rented bikes acording to seasons ')]
```



1. Why did you pick the specific chart?

to see the relationship between rented bike count and hour with respect to SEASONS.

2. What is/are the insight(s) found from the chart?

- ***In the above bar plot and point plot which shows the use of rented bike in in four different seasons, and it clearly shows that,***
- ***In summer season the use of rented bike is high and peak time is 7am-9am and 7pm-5pm.***
- ***In winter season the use of rented bike is very low because of snowfall.***

3. Will the gained insights help creating a positive business impact?

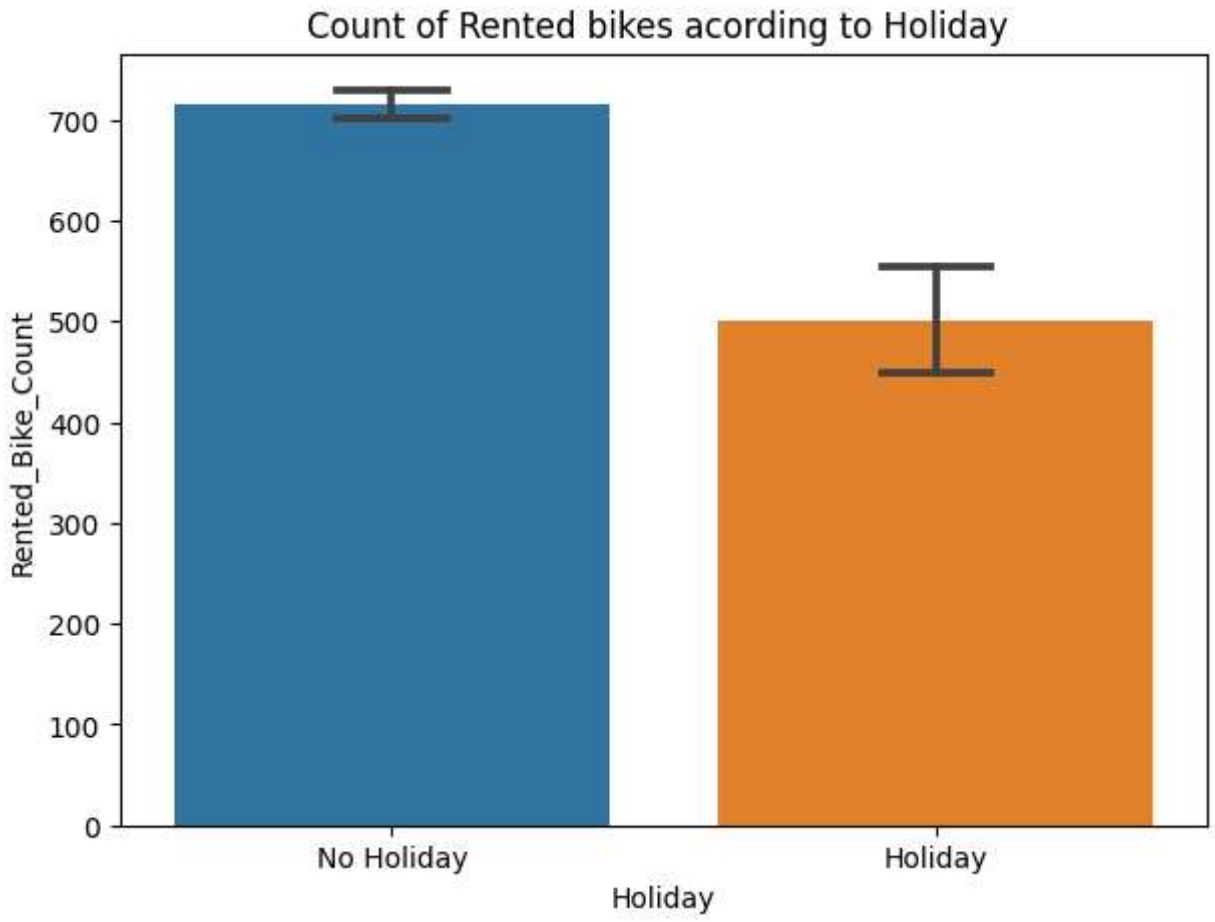
Are there any insights that lead to negative growth? Justify with specific reason.

Yes.It will help to gain insight to help creating a positive business impact.

Holiday

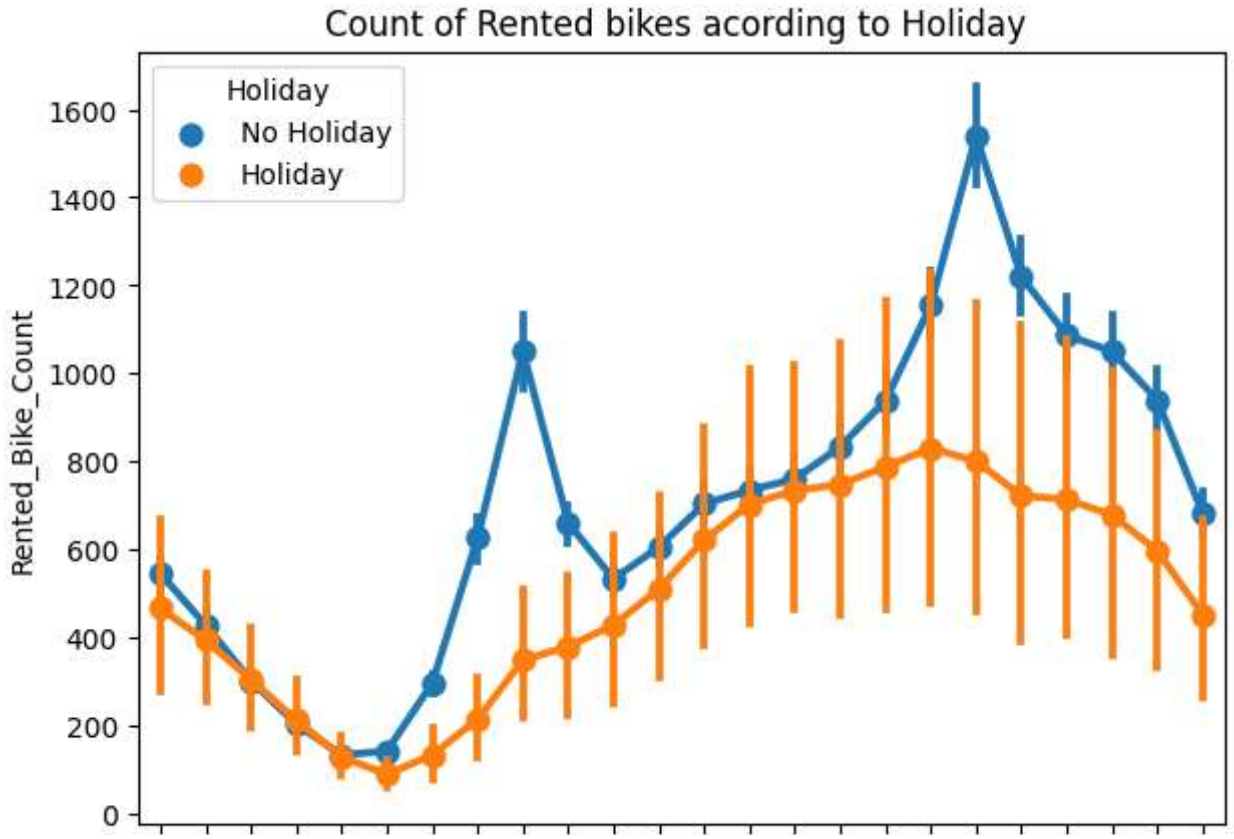
```
#anlysis of data by vizualisation
fig,ax=plt.subplots(figsize=(7,5))
sns.barplot(data=dataset,x='Holiday',y='Rented_Bike_Count',ax=ax,capsize=.2)
ax.set(title='Count of Rented bikes acording to Holiday ')
```

```
[Text(0.5, 1.0, 'Count of Rented bikes acording to Holiday ')]
```




```
#anlysis of data by vizualisation
fig,ax=plt.subplots(figsize=(7,5))
sns.pointplot(data=dataset,x='Hour',y='Rented_Bike_Count',hue='Holiday',ax=ax)
ax.set(title='Count of Rented bikes acording to Holiday ')
```

```
[Text(0.5, 1.0, 'Count of Rented bikes acording to Holiday ')]
```



- ***In the above bar plot and point plot which shows the use of rented bike in a holiday, and it clearly shows that,***
- ***plot shows that in holiday people uses the rented bike from 2pm-8pm***

```
categorical_features = dataset.describe(include=['object','category']).columns
```

```
categorical_features
```

```
Index(['Hour', 'Seasons', 'Holiday', 'Functioning_Day', 'month',
       'weekdays_weekend'],
      dtype='object')
```

CHART 4

```
# plot a bar plot for each categorical feature count
for col in categorical_features:
    counts = dataset[col].value_counts().sort_index()
    fig = plt.figure(figsize=(7, 5))
    ax = fig.gca()
    counts.plot.bar(ax = ax, color='red')
    ax.set_title(col + ' counts')
    ax.set_xlabel(col)
```

```
        ax.set_ylabel("Frequency")
    plt.show
```

```
<function matplotlib.pyplot.show(close=None, block=None)>
```

1. Why did you pick the specific chart?

to see the relationship between CATEGORICAL FEATURES AND FREQUENCY.

2. What is/are the insight(s) found from the chart?

From the above bar plot we can clearly say that from the month 5 to 10 the demand of the rented bike is high as compare to other months.these months are comes inside the summer season.

*** In weekdays_weekend which is having maximum count on sunday that is 0.**

3. Will the gained insights help creating a positive business impact?

Are there any insights that lead to negative growth? Justify with specific reason.

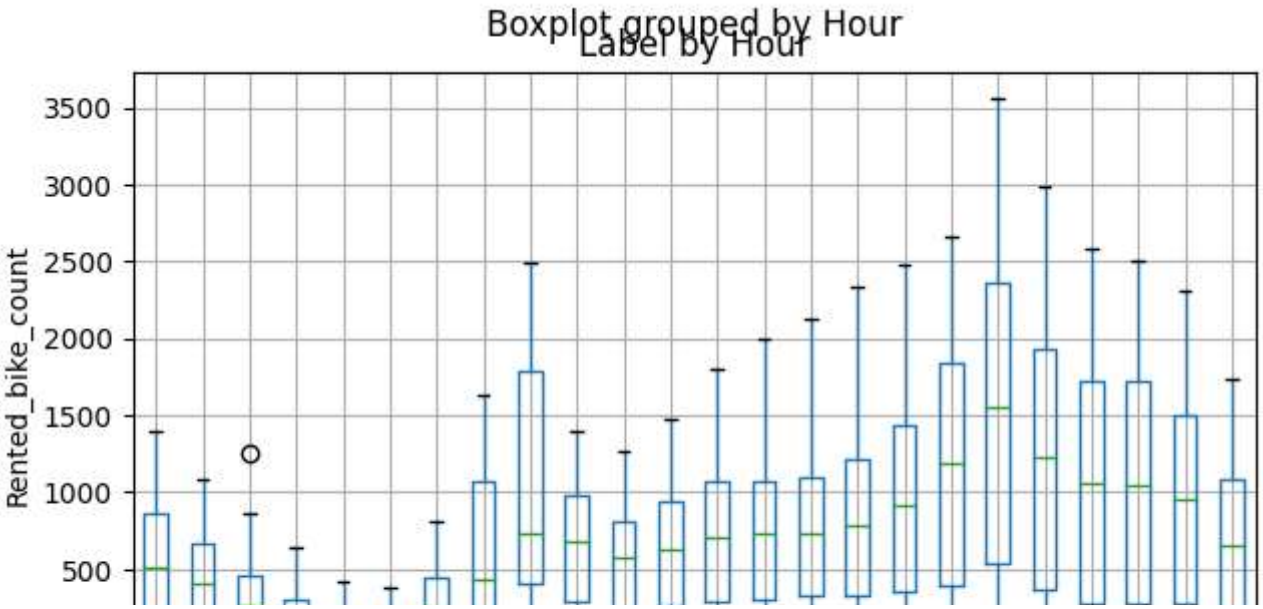
Yes.It will help to gain insight to help creating a positive business impact.

```
dataset.head(1)
```

	Rented_Bike_Count	Hour	Temperature	Humidity	Wind_speed	Visibility	Dew_point_tem
0	254	0	-5.2	37	2.2	2000	

CHART 5

```
# plot a boxplot for the label by each categorical feature
for col in categorical_features:
    fig = plt.figure(figsize=(7, 4))
    ax = fig.gca()
    dataset.boxplot(column = 'Rented_Bike_Count', by = col, ax = ax)
    ax.set_title('Label by ' + col)
    ax.set_ylabel("Rented_bike_count")
plt.show()
```



1. Why did you pick the specific chart?

to see the relationship between CATEGORICAL FEATURES AND RENTED_BIKE_COUNT.

2. What is/are the insight(s) found from the chart?

there is some categorical features are there where there is outliers so we want to remove that outliers.

3. Will the gained insights help creating a positive business impact?

Are there any insights that lead to negative growth? Justify with specific reason.

no.It will help to gain insight to help creating a negative business impact.

```
#Removal of outlier:
# plot a boxplot for the label by each categorical feature
for col in categorical_features:
    fig = plt.figure(figsize=(7, 4))
    ax = fig.gca()
    dataset.boxplot(column = np.sqrt(dataset['Rented_Bike_Count'], by = col, ax = ax)
    ax.set_title('Label by ' + col)
    ax.set_ylabel("Rented_bike_count")
plt.show()
```

```
#removal of outlier:
#for col in categorical_features:
    #fig = plt.figure(figsize=(9, 6))
    #ax = fig.gca()
    #q = dataset["Rented_bike_count"].quantile(0.95)
    #dataset = dataset[dataset['Rented_bike_count'] < q]
    #dataset.boxplot(column = 'Rented_bike_count', by = col, ax = ax)
    #ax.set_title('Label by ' + col)
    #ax.set_ylabel("Rented_bike_count")
#plt.show()
```

Double-click (or enter) to edit

Analyze of Numerical variables

What is Numerical Data

- ***Numerical data is a data type expressed in numbers, rather than natural language description. Sometimes called quantitative data, numerical data is always collected in number form. Numerical data differentiates itself from other number form data types with its ability to carry out arithmetic operations with these numbers.***

Analyze of Numerical variables distplots

```
dataset.describe().columns
```

```
Index(['Rented_Bike_Count', 'Temperature', 'Humidity', 'Wind_speed',
       'Visibility', 'Dew_point_temperature', 'Solar_Radiation', 'Rainfall',
       'Snowfall'],
      dtype='object')
```

```
pd.Index(list(dataset.select_dtypes(['int64','float64']).columns))
```

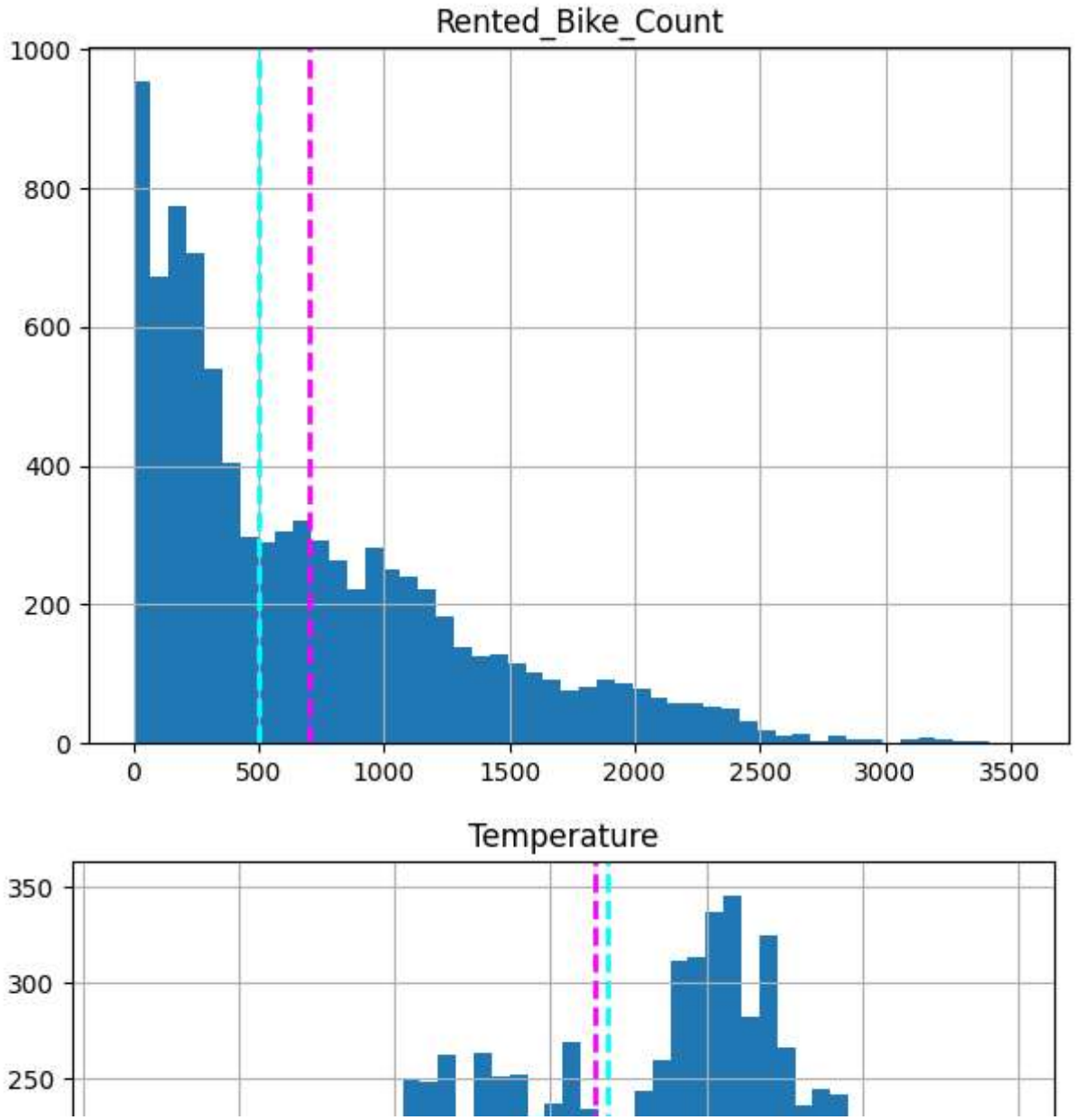
```
Index(['Rented_Bike_Count', 'Temperature', 'Humidity', 'Wind_speed',
       'Visibility', 'Dew_point_temperature', 'Solar_Radiation', 'Rainfall',
       'Snowfall'],
      dtype='object')
```


```
#assign the numerical coulmn to variavle
numerical_columns=list(dataset.select_dtypes(['int64','float64']).columns)
numerical_features=pd.Index(numerical_columns)
numerical_features
```

```
Index(['Rented_Bike_Count', 'Temperature', 'Humidity', 'Wind_speed',
       'Visibility', 'Dew_point_temperature', 'Solar_Radiation', 'Rainfall',
       'Snowfall'],
      dtype='object')
```

CHART 6


```
for col in numerical_features[:]:
    fig = plt.figure(figsize=(7, 5))
    ax = fig.gca()
    feature = dataset[col]
    feature.hist(bins=50, ax = ax)
    ax.axvline(feature.mean(), color='magenta', linestyle='dashed', linewidth=2)
    ax.axvline(feature.median(), color='cyan', linestyle='dashed', linewidth=2)
    ax.set_title(col)
plt.show()
```



1. Why did you pick the specific chart?

to see the relationship between numerical FEATURES AND density.

2. What is/are the insight(s) found from the chart?

there is some numerical features are there where there is positive skew,negative skews may be because of outliers.we have to convert them in the normal distibution.

3. Will the gained insights help creating a positive business impact?

Are there any insights that lead to negative growth? Justify with specific reason.

no.It will help to gain insight to help creating a negative business impact.

```
# we have to convert into normal distribution
for col in numerical_features[:]:
    fig = plt.figure(figsize=(7, 5))
    ax = fig.gca()
    feature = np.sqrt(dataset[col])
    feature.hist(bins=50, ax = ax)
    ax.axvline(feature.mean(), color='magenta', linestyle='dashed', linewidth=2)
    ax.axvline(feature.median(), color='cyan', linestyle='dashed', linewidth=2)
    ax.set_title(col)
plt.show()
```

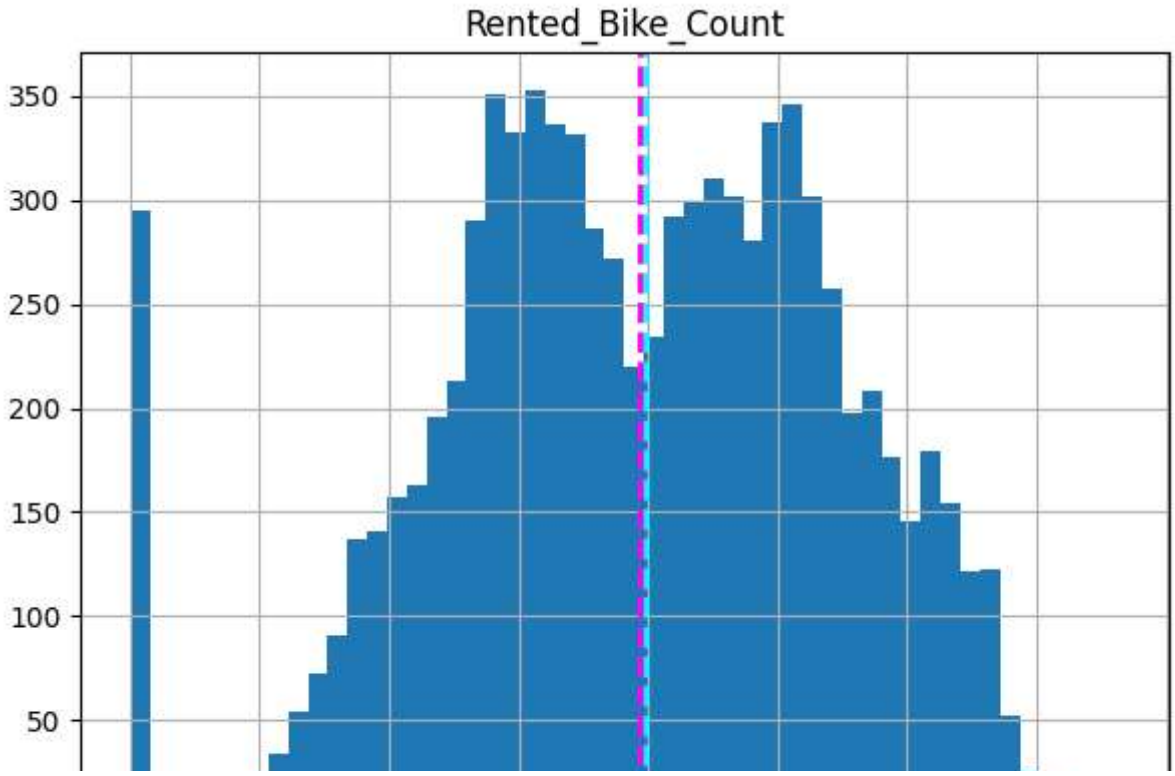
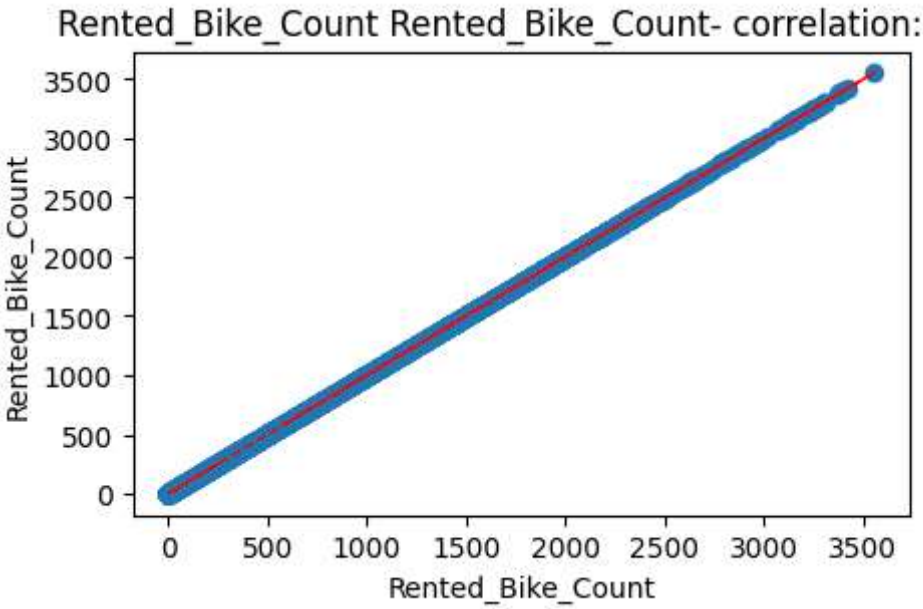
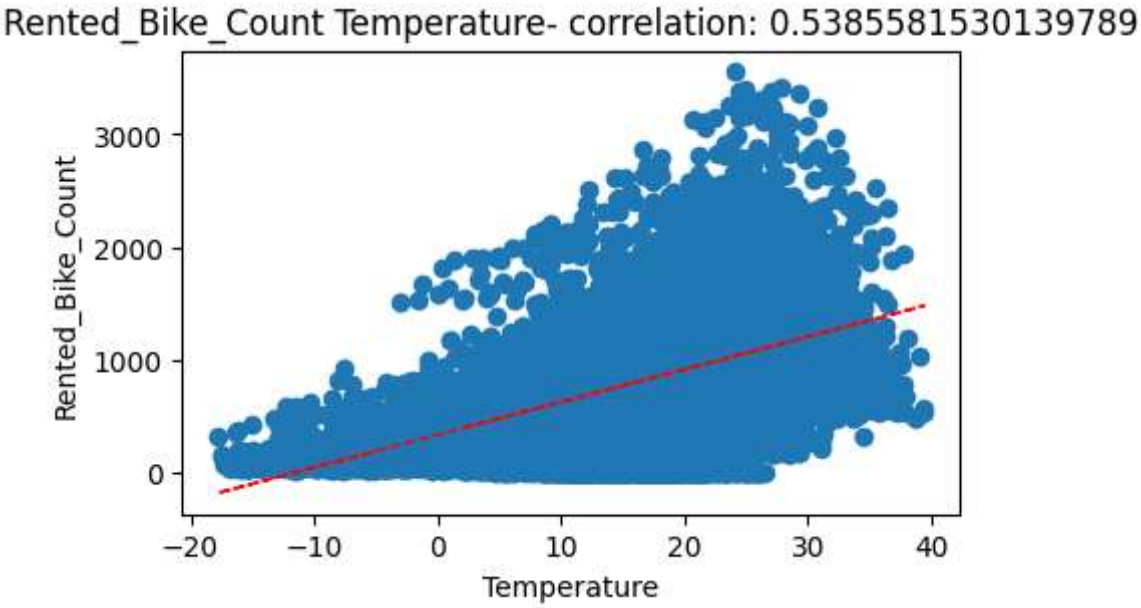
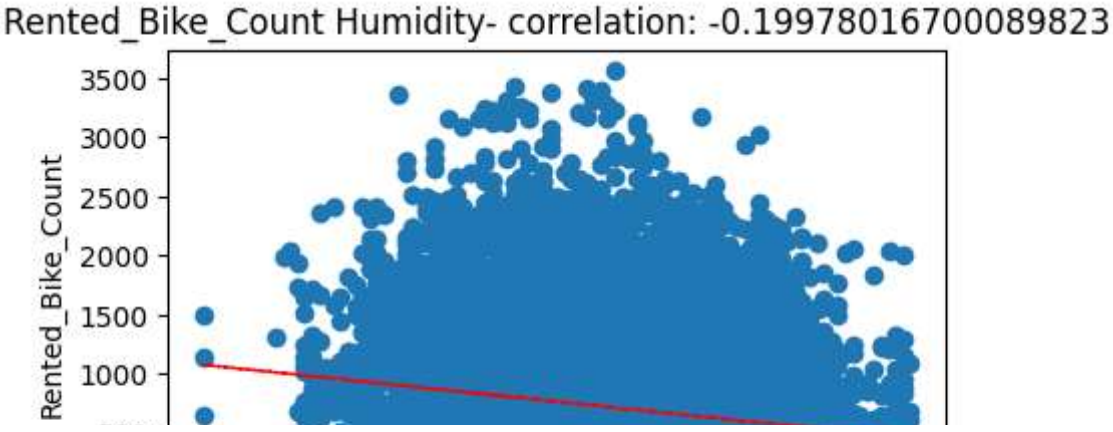


CHART 7


```
#plot scatterplot
for col in numerical_features:
   fig = plt.figure(figsize=(5, 3))
   ax = fig.gca()
   feature = dataset[col]
   label = dataset['Rented_Bike_Count']
   correlation = feature.corr(label)
   plt.scatter(x=feature, y=label)
   plt.xlabel(col)
   plt.ylabel('Rented_Bike_Count')
   ax.set_title('Rented_Bike_Count ' + col + '- correlation: ' + str(correlation))
   z = np.polyfit(dataset[col], dataset['Rented_Bike_Count'], 1)
   y_hat = np.poly1d(z)(dataset[col])

   plt.plot(dataset[col], y_hat, "r--", lw=1)

plt.show()
```







1. Why did you pick the specific chart?

to see the relationship between numerical FEATURES AND RENTED_BIKE_COUNT IN SCATTER PLOT.

2. What is/are the insight(s) found from the chart?

there is some numerical features are there where there is positive and negative corelation with rented_bike_counts.

- ***From the above scatter plot of all numerical features we see that the columns 'Temperature', 'Wind_speed','Visibility', 'Dew_point_temperature', 'Solar_Radiation' are positively relation to the target variable.***
- ***which means the rented bike count increases with increase of these features.***
- ***'Rainfall','Snowfall','Humidity' these features are negatively related with the target variaable which means the rented bike count decreases when these features increase.***

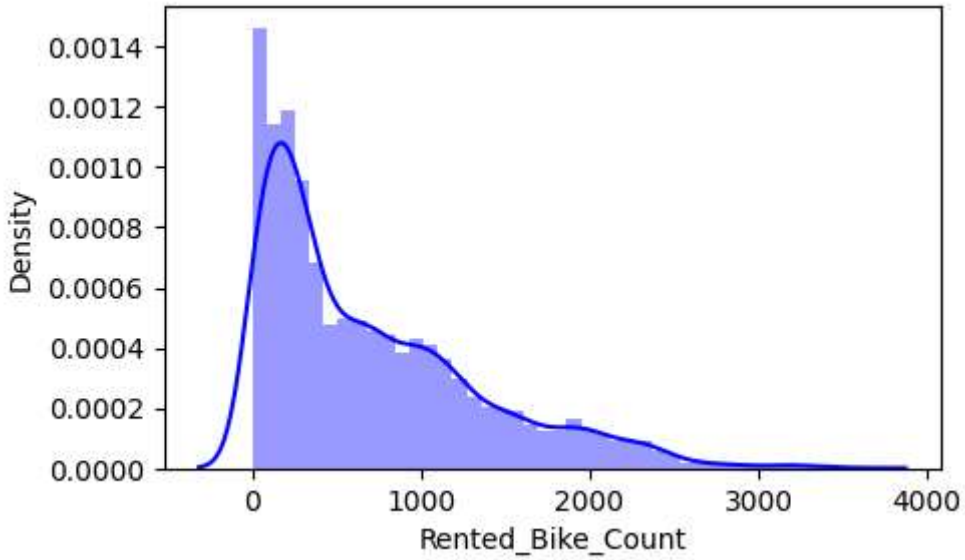
3. Will the gained insights help creating a positive business impact?

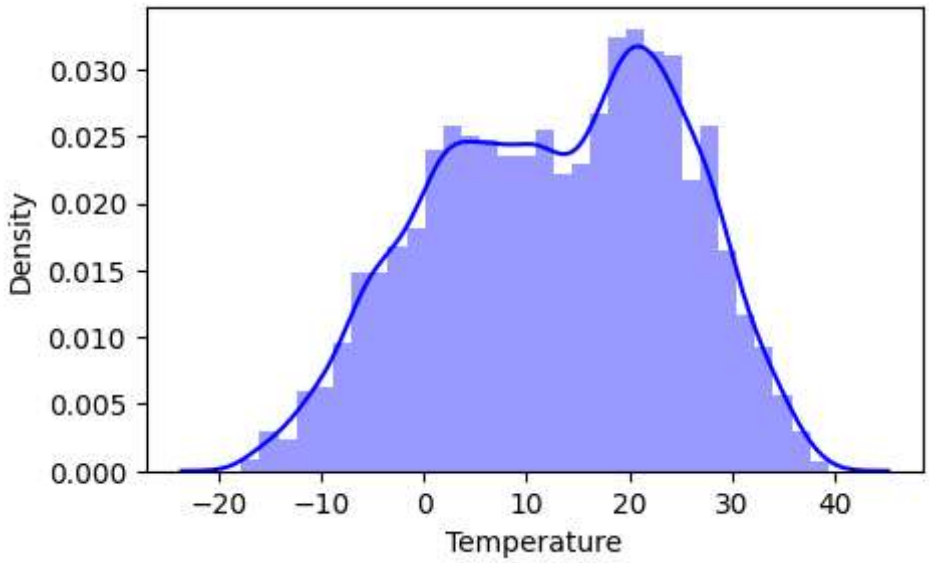
Are there any insights that lead to negative growth? Justify with specific reason.

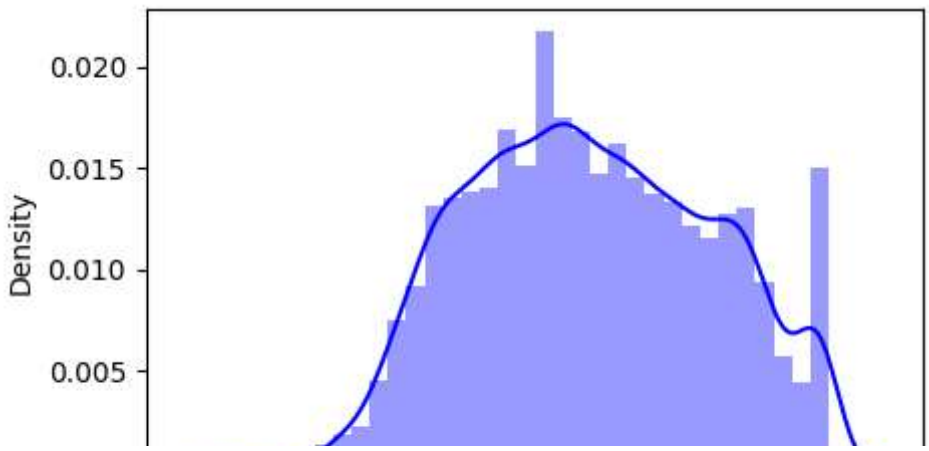
yes.It will help to gain insight to help creating a positive business impact.

CHART 8


```
#printing displots to analyze the distribution of all numerical features
for col in numerical_features:
  plt.figure(figsize=(5,3))
  sns.distplot(x=dataset[col],color='b')
  plt.xlabel(col)
plt.show()
```







1. Why did you pick the specific chart?

to see the relationship between numerical FEATURES AND density.

2. What is/are the insight(s) found from the chart?

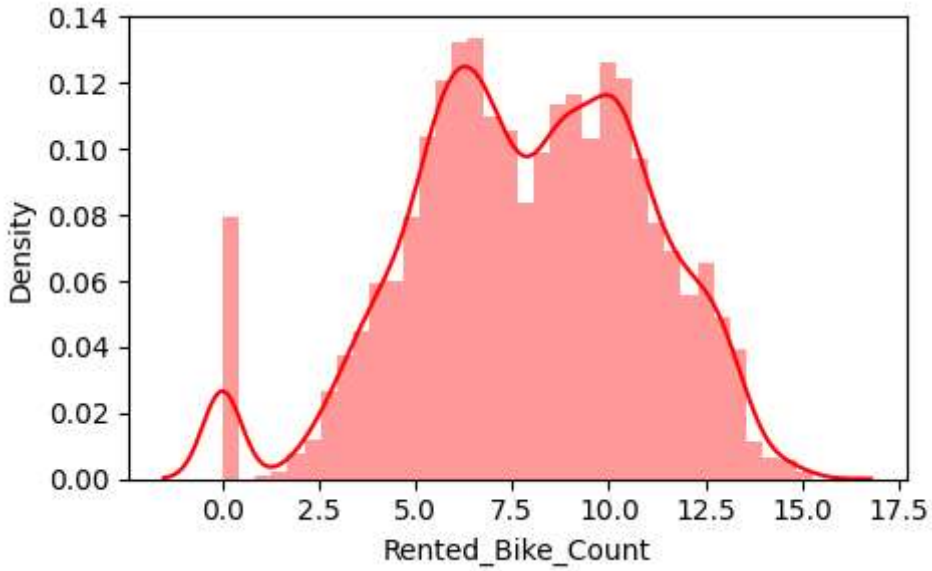
there is some numerical features are there where there is positive skew,negative skews may be because of outliers.we have to convert them in the normal distibution.

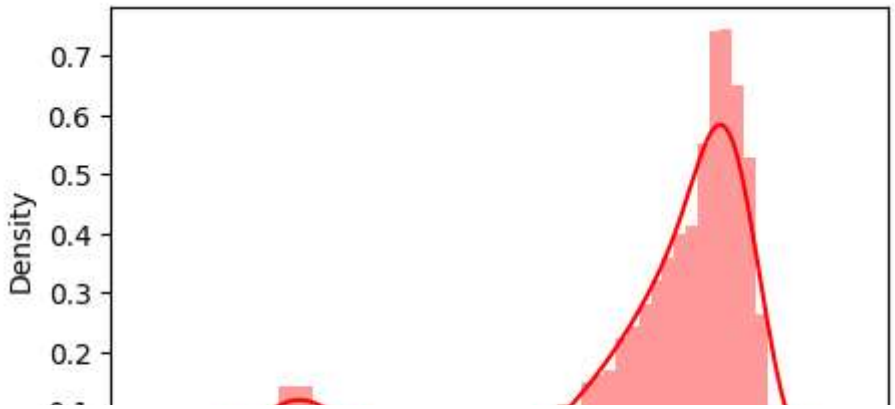
3. Will the gained insights help creating a positive business impact?

Are there any insights that lead to negative growth? Justify with specific reason.

no.It will help to gain insight to help creating a negative business impact.

```
# set upto normal distribution
for col in numerical_features:
  plt.figure(figsize=(5,3))
  sns.distplot(x=np.cbrt(dataset[col]),color ='red')
  plt.xlabel(col)
plt.show()
```





Numerical vs.Rented_Bike_Count

```
dataset.groupby('Temperature').mean()['Rented_Bike_Count'].reset_index()
```

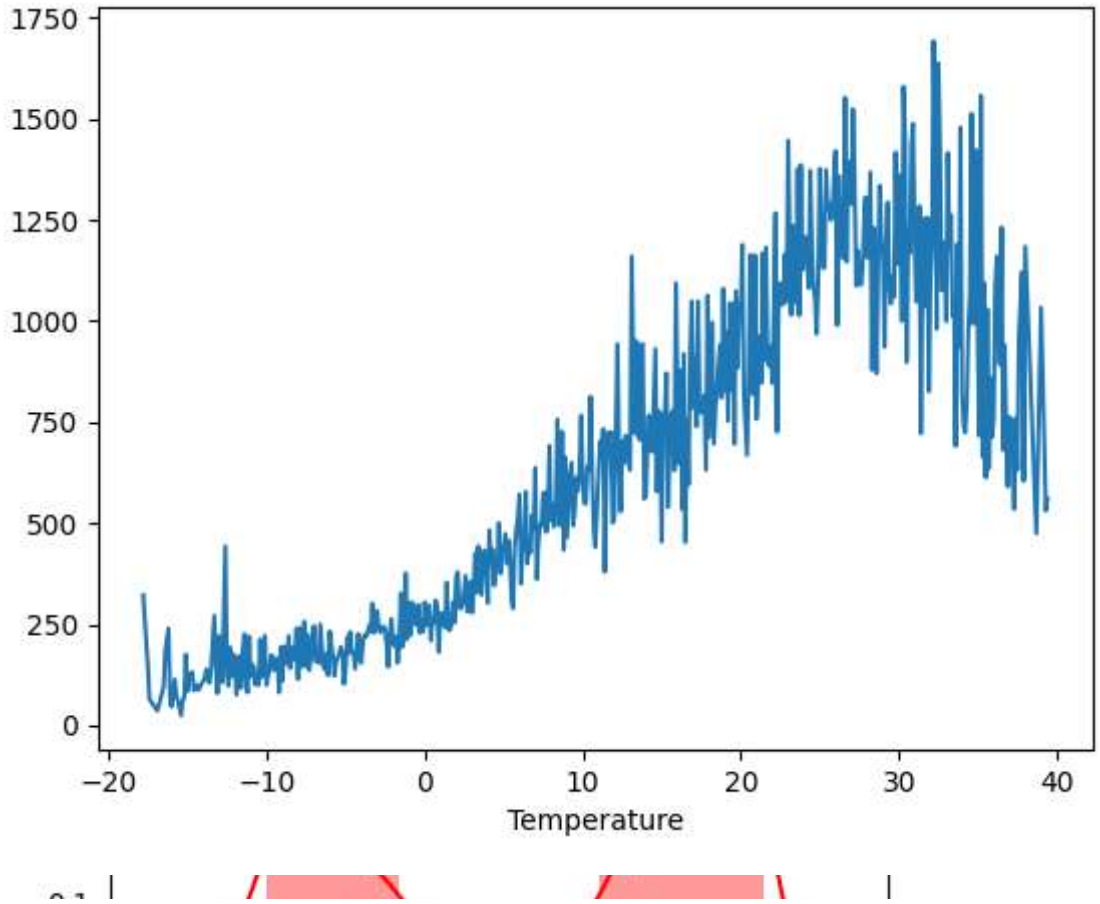
	Temperature	Rented_Bike_Count
0	-17.8	322.0

CHART 9

3	-16.9	36.0

```
#print the plot to analyze the relationship between "Rented_Bike_Count" and "Temperature"
dataset.groupby('Temperature').mean()['Rented_Bike_Count'].plot()
```

```
<Axes: xlabel='Temperature'>
```



1. Why did you pick the specific chart?

to see the relationship between Mean of Rented_bike _count AND dew_point_temperature.

2. What is/are the insight(s) found from the chart?

**** From the above plot we see that people like to ride bikes when it is pretty hot around 25°C in average***

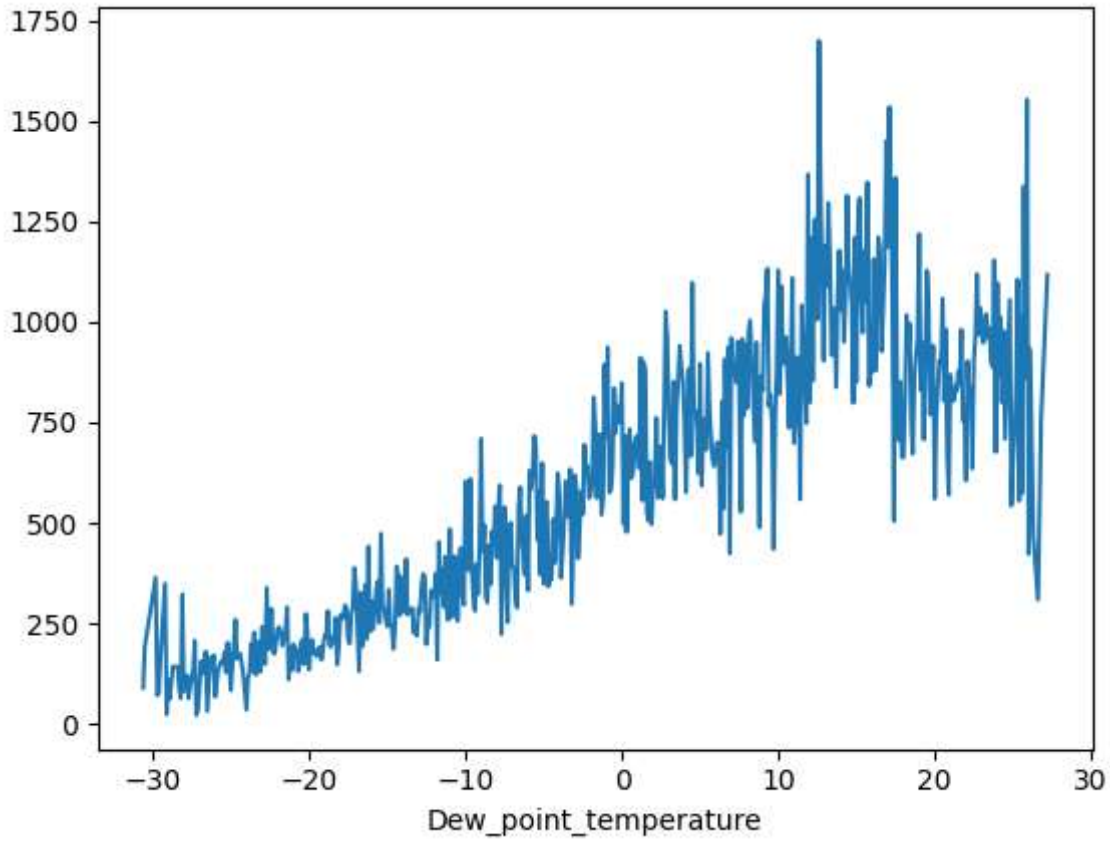
3. Will the gained insights help creating a positive business impact?

Are there any insights that lead to negative growth? Justify with specific reason.

yes.It will help to gain insight to help creating a positive business impact.

```
#print the plot to analyze the relationship between "Rented_Bike_Count" and "Dew_point_temper
dataset.groupby('Dew_point_temperature').mean()['Rented_Bike_Count'].plot()
```

```
<Axes: xlabel='Dew_point_temperature'>
```



1. Why did you pick the specific chart?

to see the relationship between Mean of Rented_bike _count AND dew_point_temperature.

2. What is/are the insight(s) found from the chart?

**** From the above plot of "Dew_point_temperature' is almost same as the 'temperature' there is some similarity present we***

can check it in our next step.

3. Will the gained insights help creating a positive business impact?

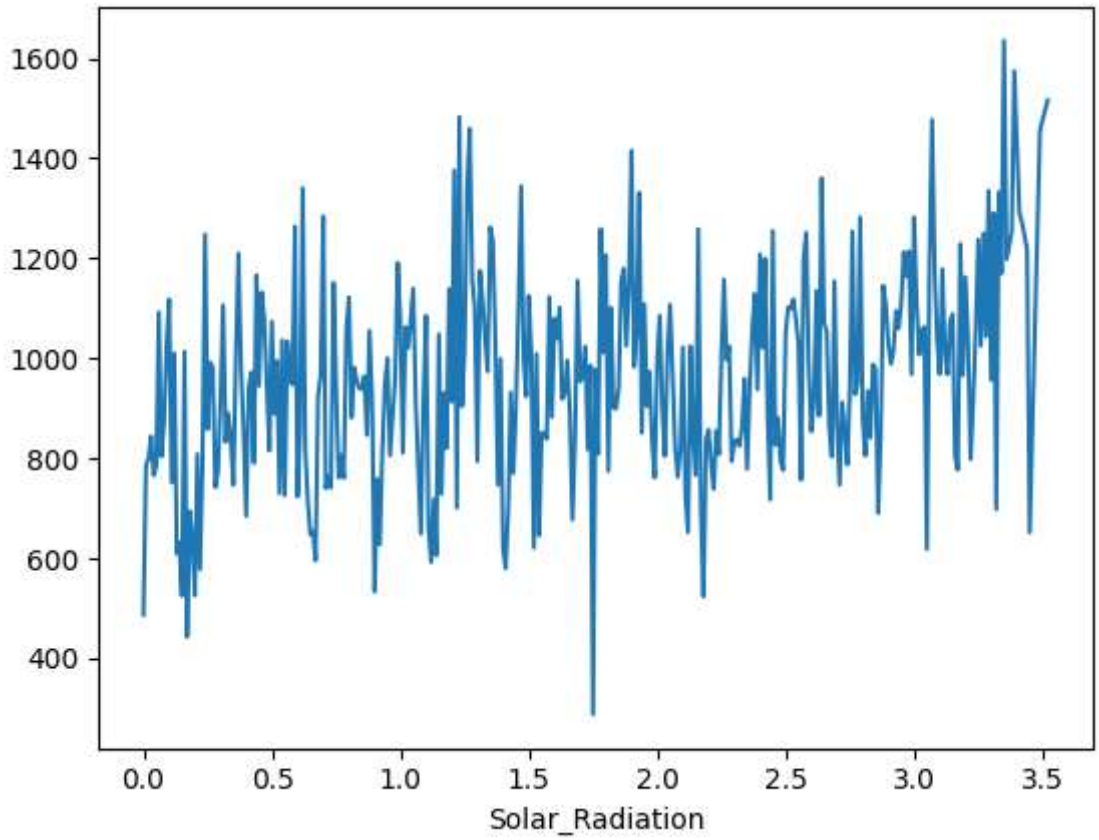
Are there any insights that lead to negative growth? Justify with specific reason.

no.It will help to gain insight to help creating a negative business impact.

- ***From the above plot of "Dew_point_temperature' is almost same as the 'temperature' there is some similarity present we can check it in our next step.***

```
#print the plot to analyze the relationship between "Rented_Bike_Count" and "Solar_Radiation"
dataset.groupby('Solar_Radiation').mean()['Rented_Bike_Count'].plot()
```

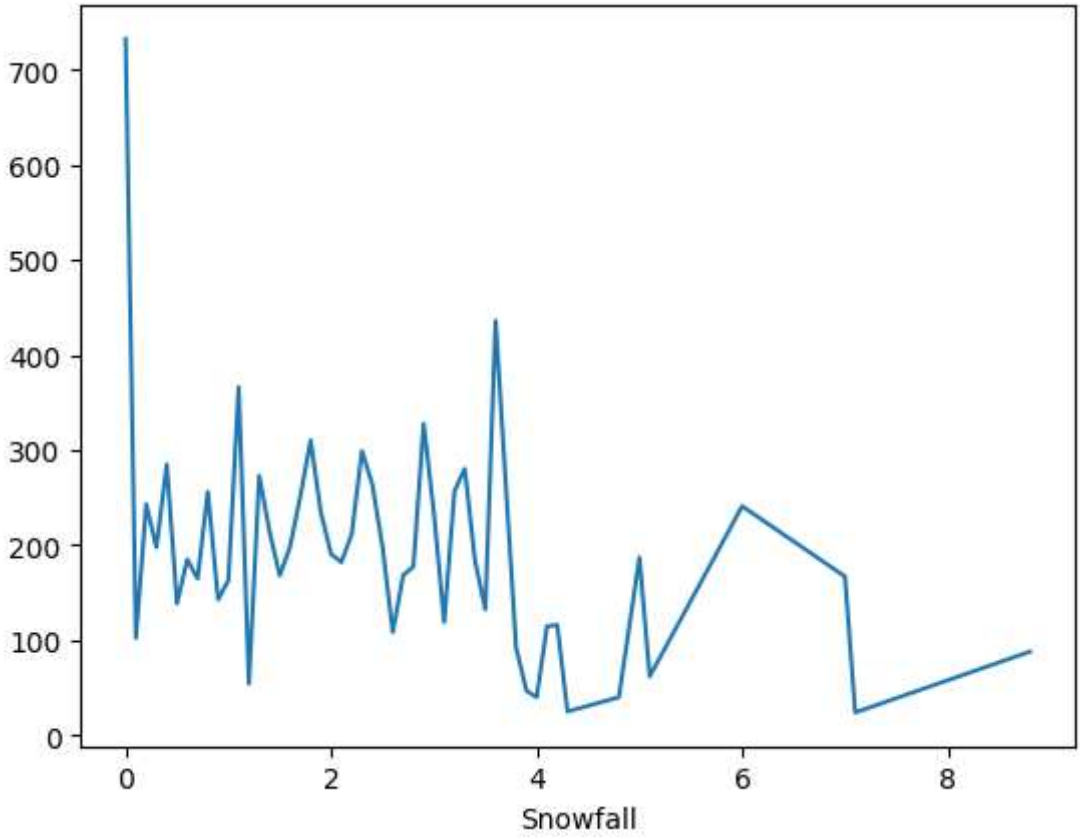
```
<Axes: xlabel='Solar_Radiation'>
```



- ***from the above plot we see that, the amount of rented bikes is huge, when there is solar radiation, the counter of rents is around 1000***

```
#print the plot to analyze the relationship between "Rented_Bike_Count" and "Snowfall"
dataset.groupby('Snowfall').mean()['Rented_Bike_Count'].plot()
```

```
<Axes: xlabel='Snowfall'>
```



- ***We can see from the plot that, on the y-axis, the amount of rented bike is very low When we have more than 4 cm of snow, the bike rents is much lower***

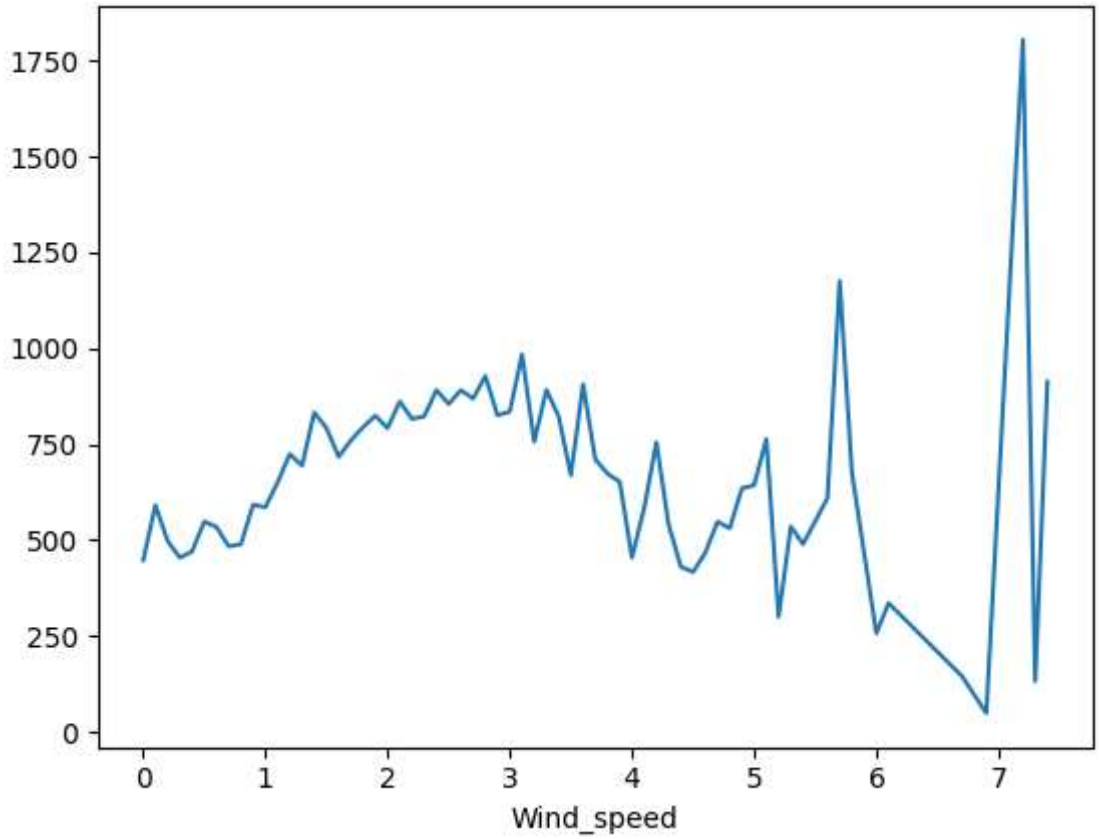
```
#print the plot to analyze the relationship between "Rented_Bike_Count" and "Rainfall"
dataset.groupby('Rainfall').mean()['Rented_Bike_Count'].plot()
```

```
<Axes: xlabel='Rainfall'>
```

- ***We can see from the above plot that even if it rains a lot the demand of of rent bikes is not decreasing, here for example even if we have 20 mm of rain there is a big peak of rented bikes***

```
#print the plot to analyze the relationship between "Rented_Bike_Count" and "Wind_speed"
dataset.groupby('Wind_speed').mean()['Rented_Bike_Count'].plot()
```

```
<Axes: xlabel='Wind_speed'>
```



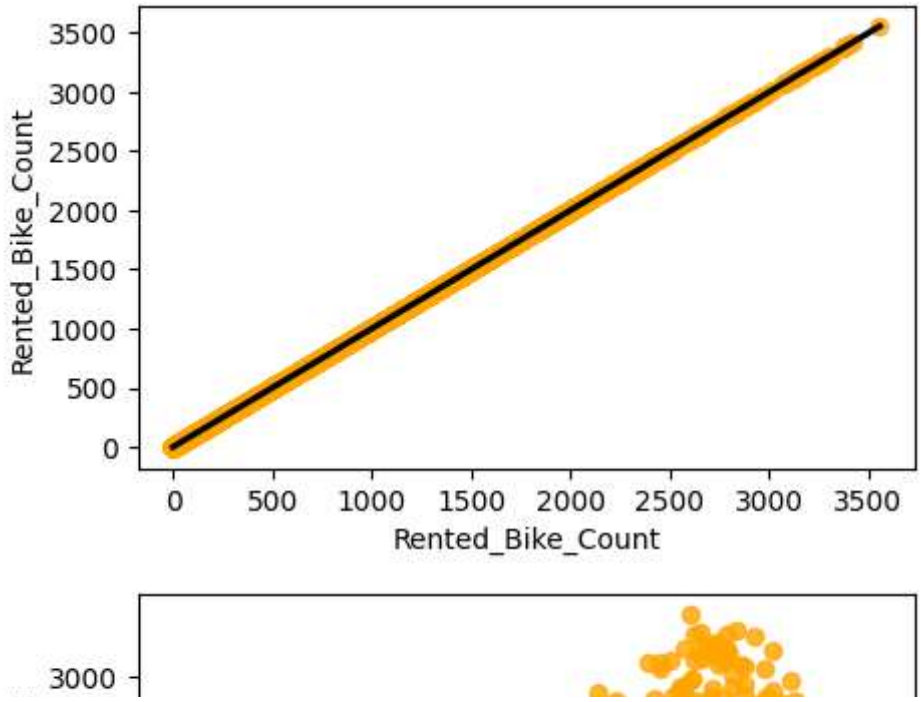
- ***We can see from the above plot that the demand of rented bike is uniformly distribute despite of wind speed but when the speed of wind was 7 m/s then the demand of bike also increase that clearly means peoples love to ride bikes when its little windy.***

Regression plot

- ***The regression plots in seaborn are primarily intended to add a visual guide that helps to emphasize patterns in a dataset during exploratory data analyses. Regression plots as the name suggests creates a regression line between 2 parameters and helps to visualize their linear relationships.***

CHART 10

```
#printing the regression plot for all the numerical features
for col in numerical_features:
  fig,ax=plt.subplots(figsize=(5,3))
  sns.regplot(x=dataset[col],y=dataset['Rented_Bike_Count'],scatter_kws={"color": 'orange'},
```



1. Why did you pick the specific chart?

to see the relationship between numerical FEATURES AND RENTED_BIKE_COUNT IN REGRESSION PLOT.

2. What is/are the insight(s) found from the chart?

there is some numerical features are there where there is positive and negative corelation with rented_bike_counts.

**** From the above regression plot of all numerical features we see that the columns 'Temperature', 'Wind_speed','Visibility', 'Dew_point_temperature', 'Solar_Radiation' are positively relation to the target variable.***

- ***which means the rented bike count increases with increase of these features.***
- ***'Rainfall','Snowfall','Humidity' these features are negatively related with the target variaable which means the rented bike count decreases when these features increase.***

3. Will the gained insights help creating a positive business impact?

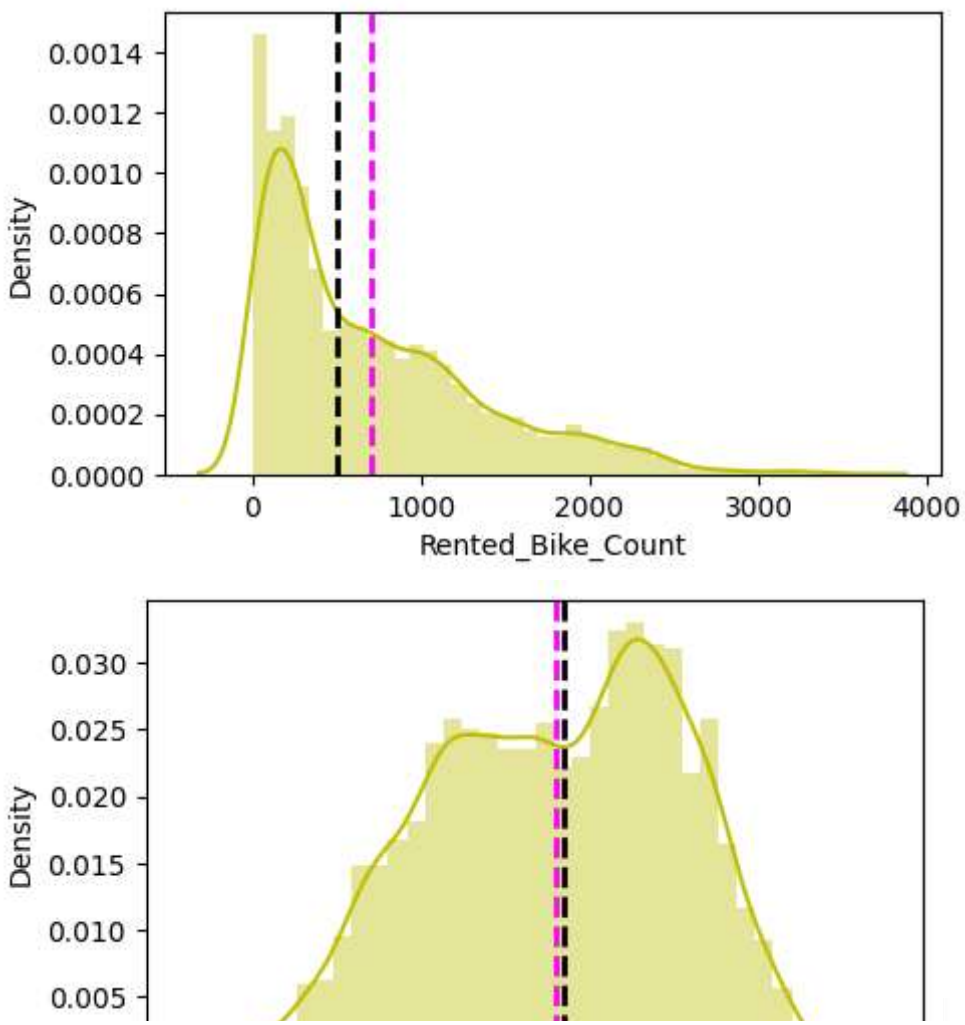
Are there any insights that lead to negative growth? Justify with specific reason.

yes.It will help to gain insight to help creating a positive business impact.

Normalise Rented_Bike_Count column data

- ***The data normalization (also referred to as data pre-processing) is a basic element of data mining. It means transforming the data, namely converting the source data in to another format that allows processing data effectively. The main purpose of data normalization is to minimize or even exclude duplicated data***

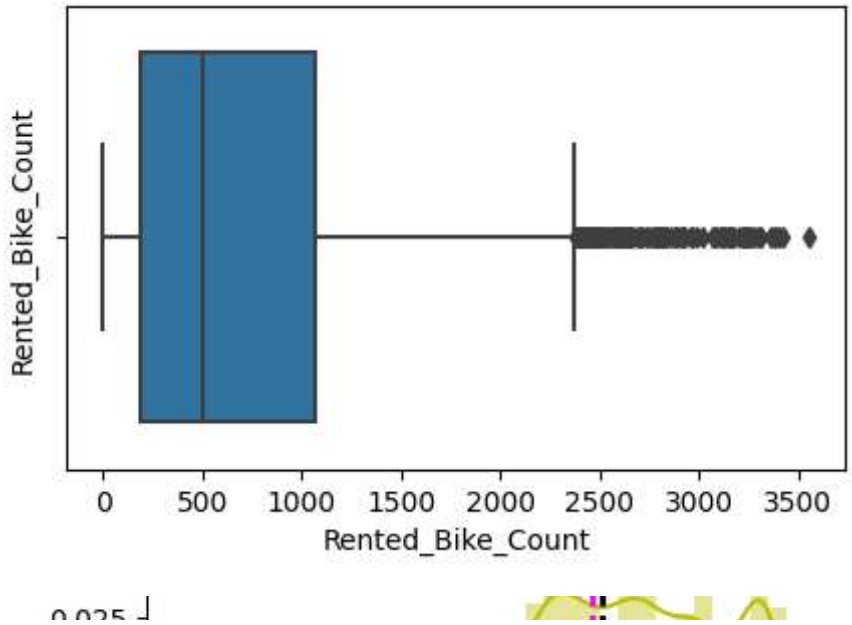
```
for col in numerical_features:
  plt.figure(figsize=(5,3))
  plt.xlabel('col')
  plt.ylabel('Density')
  ax=sns.distplot(dataset[col],hist=True ,color="y")
  ax.axvline(dataset[col].mean(), color='magenta', linestyle='dashed', linewidth=2)
  ax.axvline(dataset[col].median(), color='black', linestyle='dashed', linewidth=2)
plt.show()
```




```
#Distribution plot of Rented Bike Count
plt.figure(figsize=(5,3))
plt.xlabel('Rented_Bike_Count')
plt.ylabel('Density')
ax=sns.distplot(dataset['Rented_Bike_Count'],hist=True ,color="y")
ax.axvline(dataset['Rented_Bike_Count'].mean(), color='magenta', linestyle='dashed', linewidt
ax.axvline(dataset['Rented_Bike_Count'].median(), color='black', linestyle='dashed', linewidt
plt.show()
```

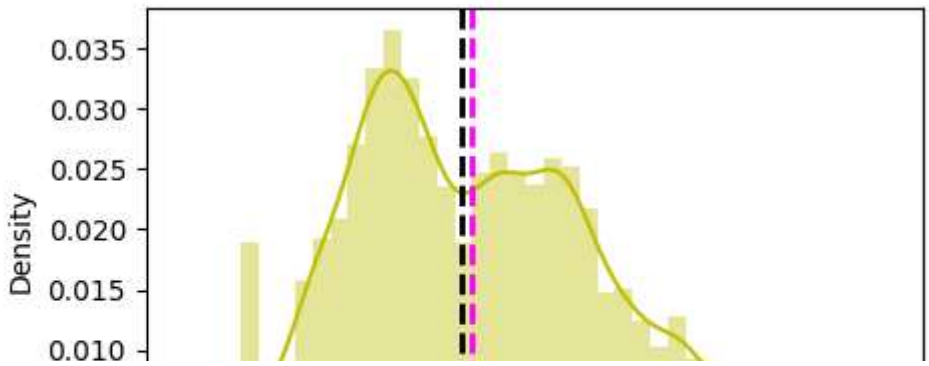
- ***The above graph shows that Rented Bike Count has moderate right skewness. Since the assumption of linear regression is that 'the distribution of dependent variable has to be normal', so we should perform some operation to make it normal.***

```
#Boxplot of Rented Bike Count to check outliers
plt.figure(figsize=(5,3))
plt.ylabel('Rented_Bike_Count')
sns.boxplot(x=dataset['Rented_Bike_Count'])
plt.show()
```



- ***The above boxplot shows that we have detect outliers in Rented Bike Count column***

```
#Applying square root to Rented Bike Count to improve skewness
plt.figure(figsize=(5,3))
plt.xlabel('Rented Bike Count')
plt.ylabel('Density')

ax=sns.distplot(np.sqrt(dataset['Rented_Bike_Count']), color="y")
ax.axvline(np.sqrt(dataset['Rented_Bike_Count']).mean(), color='magenta', linestyle='dashed',
ax.axvline(np.sqrt(dataset['Rented_Bike_Count']).median(), color='black', linestyle='dashed',

plt.show()
```



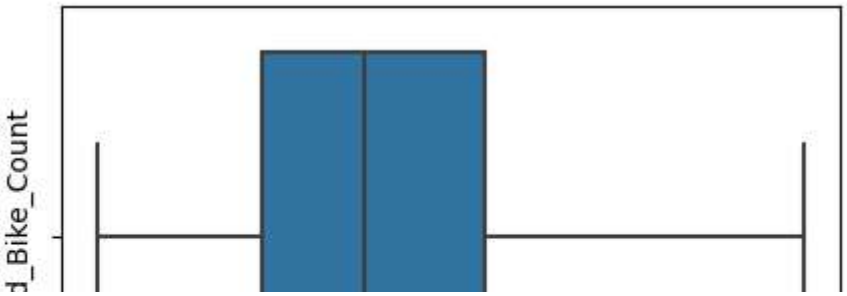

```
dataset.agg(['skew', 'kurtosis']).transpose()
```

	skew	kurtosis
Rented_Bike_Count	1.153428	0.853387
Temperature	-0.198326	-0.837786
Humidity	0.059579	-0.803559
Wind_speed	0.890955	0.727179
Visibility	-0.701786	-0.961980
Dew_point_temperature	-0.367298	-0.755430
Solar_Radiation	1.504040	1.126433
Rainfall	14.533232	284.991099
Snowfall	8.440801	93.803324

- ***Since we have generic rule of applying Square root for the skewed variable in order to make it normal .After applying Square root to the skewed Rented Bike Count, here we get almost normal distribution.***

Snowfall

```
#After applying sqrt on Rented Bike Count check wheater we still have outliers
plt.figure(figsize=(5,3))

plt.ylabel('Rented_Bike_Count')
sns.boxplot(x=np.sqrt(dataset['Rented_Bike_Count']))
plt.show()
```




```
dataset.corr()
```

	Rented_Bike_Count	Temperature	Humidity	Wind_speed	Visibility
Rented_Bike_Count	1.000000	0.538558	-0.199780	0.121108	0.199280
Temperature	0.538558	1.000000	0.159371	-0.036252	0.034794
Humidity	-0.199780	0.159371	1.000000	-0.336683	-0.543090
Wind_speed	0.121108	-0.036252	-0.336683	1.000000	0.171507
Visibility	0.199280	0.034794	-0.543090	0.171507	1.000000
Dew_point_temperature	0.379788	0.912798	0.536894	-0.176486	-0.176630
Solar_Radiation	0.261837	0.353505	-0.461919	0.332274	0.149738
Rainfall	-0.123074	0.050282	0.236397	-0.019674	-0.167629
Snowfall	-0.141804	-0.218405	0.108183	-0.003554	-0.121695

- ***After applying Square root to the Rented Bike Count column, we find that there is no outliers present.***

Checking of Correlation between variables

Checking in OLS Model

Ordinary least squares (OLS) regression is a statistical method of analysis that estimates the relationship between one or more independent variables and a dependent variable

```
#import the module
#assign the 'x','y' value
import statsmodels.api as sm
X = dataset[[ 'Temperature','Humidity',
       'Wind_speed', 'Visibility','Dew_point_temperature',
       'Solar_Radiation', 'Rainfall', 'Snowfall']]
```

```
Y = dataset['Rented_Bike_Count']
dataset.head()
```

	Rented_Bike_Count	Hour	Temperature	Humidity	Wind_speed	Visibility	Dew_point_tem
0	254	0	-5.2	37	2.2	2000	
1	204	1	-5.5	38	0.8	2000	
2	173	2	-6.0	39	1.0	2000	
3	107	3	-6.2	40	0.9	2000	
4	78	4	-6.0	36	2.3	2000	

```
#add a constant column
X = sm.add_constant(X)
X
```

	const	Temperature	Humidity	Wind_speed	Visibility	Dew_point_temperature	Solar
0	1.0	-5.2	37	2.2	2000	-17.6	
1	1.0	-5.5	38	0.8	2000	-17.6	
2	1.0	-6.0	39	1.0	2000	-17.7	
3	1.0	-6.2	40	0.9	2000	-17.6	
4	1.0	-6.0	36	2.3	2000	-18.6	
...	...	...	...	...	...	...	
8755	1.0	4.2	34	2.6	1894	-10.3	
8756	1.0	3.4	37	2.3	2000	-9.9	
8757	1.0	2.6	39	0.3	1968	-9.9	
8758	1.0	2.1	41	1.0	1859	-9.8	
8759	1.0	1.9	43	1.3	1909	-9.3	

8760 rows × 9 columns

```
## fit a OLS model
```

```
model= sm.OLS(Y, X).fit()
model.summary()
```

OLS Regression Results

Dep. Variable:	Rented_Bike_Count	**R-squared (uncentered):**	0.724
Model:	OLS	**Adj. R-squared (uncentered):**	0.723
Method:	Least Squares	**F-statistic:**	2863.
Date:	Sun, 04 Jun 2023	**Prob (F-statistic):**	0.00
Time:	07:14:29	**Log-Likelihood:**	-66909.
No. Observations:	8760	**AIC:**	1.338e+05
Df Residuals:	8752	**BIC:**	1.339e+05
Df Model:	8		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Temperature	66.9774	1.648	40.647	0.000	63.747	70.207
Humidity	-1.2248	0.196	-6.250	0.000	-1.609	-0.841
Wind_speed	55.1557	5.671	9.727	0.000	44.040	66.271
Visibility	0.0199	0.010	1.924	0.054	-0.000	0.040
Dew_point_temperature	-33.5953	1.530	-21.957	0.000	-36.595	-30.596
Solar_Radiation	-123.2678	8.683	-14.196	0.000	-140.289	-106.246
Rainfall	-56.0617	4.903	-11.434	0.000	-65.672	-46.451
Snowfall	33.4346	12.815	2.609	0.009	8.314	58.555

Omnibus:	964.502	**Durbin-Watson:**	0.338
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	1585.356
Skew:	0.778	**Prob(JB):**	0.00
Kurtosis:	4.387	**Cond. No.**	3.74e+03

Notes:
[1] R^2 is computed without centering (uncentered) since the model does not contain a constant.
[2] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[3] The condition number is large, 3.74e+03. This might indicate that there are
strong multicollinearity or other numerical problems.

- **R sqauare and Adj Square are near to each other. 40% of variance in the Rented Bike count is explained by the model.**
- **For F statistic , P value is less than 0.05 for 5% levelof significance.**
- **P value of dew point temp and visibility are very high and they are not significant.**
- **Omnibus tests the skewness and kurtosis of the residuals. Here the value of Omnibus is high., it shows we have skewness in our data.**
- **The condition number is large, 3.11e+04. This might indicate that there are strong multicollinearity or other numerical problems**

- **Durbin-Watson tests for autocorrelation of the residuals. Here value is less than 0.5. We can say that there exists a positive auto correlation among the variables.**

```
X.corr()
```

	const	Temperature	Humidity	Wind_speed	Visibility	Dew_point_t
const	NaN	NaN	NaN	NaN	NaN	
Temperature	NaN	1.000000	0.159371	-0.036252	0.034794	
Humidity	NaN	0.159371	1.000000	-0.336683	-0.543090	
Wind_speed	NaN	-0.036252	-0.336683	1.000000	0.171507	
Visibility	NaN	0.034794	-0.543090	0.171507	1.000000	
Dew_point_temperature	NaN	0.912798	0.536894	-0.176486	-0.176630	
Solar_Radiation	NaN	0.353505	-0.461919	0.332274	0.149738	
Rainfall	NaN	0.050282	0.236397	-0.019674	-0.167629	
Snowfall	NaN	-0.218405	0.108183	-0.003554	-0.121695	

- ***From the OLS model we find that the 'Temperature' and 'Dew_point_temperature' are highly correlated so we need to drop one of them.***
- ***for droping the we check the (P>|t|) value from above table and we can see that the 'Dew_point_temperature' value is higher so we need to drop Dew_point_temperature column***
- ***For clarity, we use visualisation i.e heatmap in next step***

CHART 11

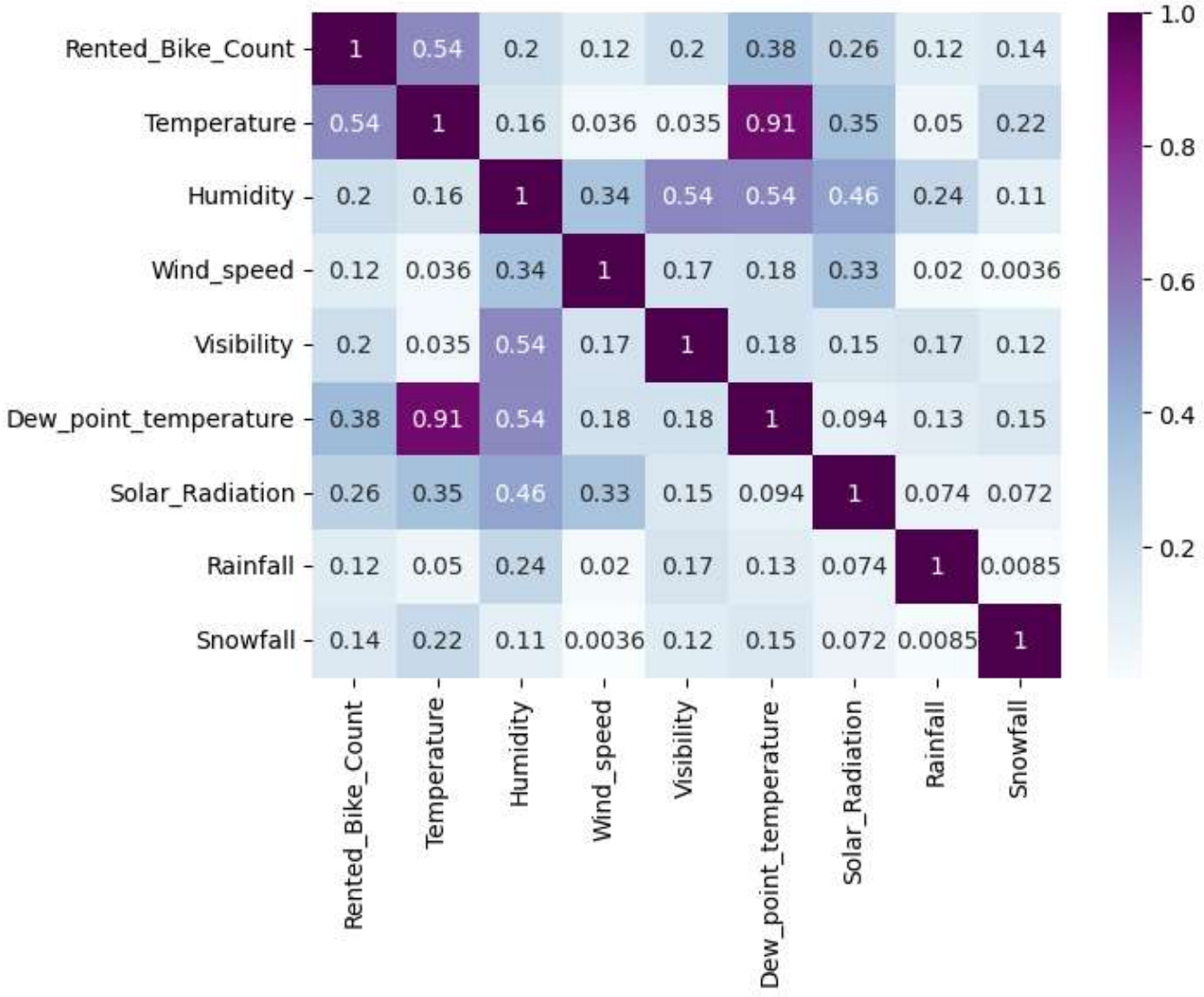
Heatmap

- **we check correletion betweeen variables using Correlation heatmap, it is graphical representation of correlation matrix representing correlation between different variables**

```
## plot the Correlation matrix
#plt.figure(figsize=(7,5))
#correlation=dataset.corr()
#mask = np.triu(np.ones_like(correlation, dtype=bool))
#sns.heatmap((correlation),mask=mask, annot=True,cmap='BuPu')
```

```
# Correlation Heatmap visualization code
#plt.figure(figsize=(15,12))
#sns.heatmap(df.corr(), cmap="Spectral", cbar_kws={'shrink': .6}, square=True, annot=True, fm
#plt.show()


plt.figure(figsize=(7,5))
correlation = dataset.corr()
sns.heatmap(abs(correlation),annot = True,cmap = 'BuPu')
```

```
<Axes: >
```



1. Why did you pick the specific chart?

Correlation heatmaps can be used to find potential relationships between variables and to understand the strength of these relationships. In addition, correlation plots can be used to identify outliers and to detect linear and nonlinear relationships.

2. What is/are the insight(s) found from the chart?

We can observe on the heatmap that on the target variable line the most positively correlated variables to the rent are :

- the temperature
- the dew point temperature
- the solar radiation

And most negatively correlated variables are:

- Humidity
- Rainfall
- ***From the above correlation heatmap, We see that there is a positive correlation between columns 'Temperature' and 'Dew point temperature' i.e 0.91 so even if we drop this column then it dont affects the outcome of our analysis. And they have the same variations.. so we can drop the column 'Dew point temperature(°C)'.***

Double-click (or enter) to edit

```
#Multicollinearity:
from  statsmodels.stats.outliers_influence import variance_inflation_factor
def calc_vif(X):
  #calculating vif
  vif = pd.DataFrame()
  vif['variables'] = X.columns
  vif['VIF']=[variance_inflation_factor(X.values,i) for i in range(X.shape[1])]
  return(vif)



calc_vif(dataset[[i for i in dataset.describe().columns if i not in ['Rented_Bike_Count']]])
```

variables	VIF

```
calc_vif(dataset[[i for i in dataset.describe().columns if i not in ['Rented_Bike_Count','Dew
```

	variables	VIF
0	Temperature	3.166007
1	Humidity	4.758651
2	Wind_speed	4.079926
3	Visibility	4.409448
4	Solar_Radiation	2.246238
5	Rainfall	1.078501
6	Snowfall	1.118901

```
#drop the Dew point temperature column
dataset=dataset.drop(['Dew_point_temperature'],axis=1)
```

```
dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8760 entries, 0 to 8759
Data columns (total 14 columns):
 #   Column             Non-Null Count  Dtype   
---  ------             --------------  -----   
 0   Rented_Bike_Count  8760 non-null   int64   
 1   Hour               8760 non-null   category
 2   Temperature        8760 non-null   float64 
 3   Humidity           8760 non-null   int64   
 4   Wind_speed         8760 non-null   float64 
 5   Visibility         8760 non-null   int64   
 6   Solar_Radiation    8760 non-null   float64 
 7   Rainfall           8760 non-null   float64 
 8   Snowfall           8760 non-null   float64 
 9   Seasons            8760 non-null   object  
 10  Holiday            8760 non-null   object  
 11  Functioning_Day    8760 non-null   object  
 12  month              8760 non-null   category
 13  weekdays_weekend   8760 non-null   category
dtypes: category(3), float64(5), int64(3), object(3)
memory usage: 779.8+ KB
```

5. Feature Engineering & Data Pre-processing

1. Handling Missing Values

```
# Handling Missing Values & Missing Value Imputation
#already handled
```

2. Handling Outliers

```
# Handling Outliers & Outlier treatments

#There is no outliers in this dataset.
```

3. Categorical Encoding

Create the dummy variables

A dataset may contain various type of values, sometimes it consists of categorical values. So, in-order to use those categorical value for programming efficiently we create dummy variables.

```
#Assign all catagoriacla features to a variable
categorical_features=list(dataset.select_dtypes(['object','category']).columns)
categorical_features=pd.Index(categorical_features)
categorical_features
```

```
Index(['Hour', 'Seasons', 'Holiday', 'Functioning_Day', 'month',
       'weekdays_weekend'],
      dtype='object')
```

one hot encoding and feature selection

Double-click (or enter) to edit

A one hot encoding allows the representation of categorical data to be more expressive. Many machine learning algorithms cannot work with categorical data directly. The categories must be converted into numbers. This is required for both input and output variables that are categorical.

```
#creat a copy
bike_df_copy = dataset

def one_hot_encoding(data, column):
    data = pd.concat([data, pd.get_dummies(data[column], prefix=column, drop_first=True)], ax
    data = data.drop([column], axis=1)
    return data

for col in categorical_features:
    bike_df_copy = one_hot_encoding(bike_df_copy, col)
bike_df_copy.head()
```

	Rented_Bike_Count	Temperature	Humidity	Wind_speed	Visibility	Solar_Radiation	Ra
0	254	-5.2	37	2.2	2000	0.0	
1	204	-5.5	38	0.8	2000	0.0	
2	173	-6.0	39	1.0	2000	0.0	
3	107	-6.2	40	0.9	2000	0.0	
4	78	-6.0	36	2.3	2000	0.0	

5 rows × 48 columns

```
dataset['weekdays_weekend'].value_counts()
```

```
0    6264
1    2496
Name: weekdays_weekend, dtype: int64
```

```
dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8760 entries, 0 to 8759
Data columns (total 14 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   Rented_Bike_Count  8760 non-null   int64
 1   Hour               8760 non-null   category
 2   Temperature        8760 non-null   float64
 3   Humidity           8760 non-null   int64
 4   Wind_speed         8760 non-null   float64
 5   Visibility         8760 non-null   int64
 6   Solar_Radiation    8760 non-null   float64
 7   Rainfall           8760 non-null   float64
 8   Snowfall           8760 non-null   float64
 9   Seasons            8760 non-null   object
 10  Holiday            8760 non-null   object
```

```
 11  Functioning_Day    8760 non-null   object
 12  month              8760 non-null   category
 13  weekdays_weekend   8760 non-null   category
dtypes: category(3), float64(5), int64(3), object(3)
memory usage: 779.8+ KB
```

Model Training

Train Test split for regression

Before, fitting any model it is a rule of thumb to split the dataset into a training and test set. This means some proportions of the data will go into training the model and some portion will be used to evaluate how our model is performing on any unseen data. The proportions may vary from 60:40, 70:30, 75:25 depending on the person but mostly used is 80:20 for training and testing respectively. In this step we will split our data into training and testing set using scikit learn library.

```
numerical_features
```

```
Index(['Rented_Bike_Count', 'Temperature', 'Humidity', 'Wind_speed',
       'Visibility', 'Dew_point_temperature', 'Solar_Radiation', 'Rainfall',
       'Snowfall'],
      dtype='object')
```

```
#Assign the value in X and Y
X = bike_df_copy.drop(columns=['Rented_Bike_Count'], axis=1)
y = np.sqrt(bike_df_copy['Rented_Bike_Count'])
```

```
X.head(2)
```

	Temperature	Humidity	Wind_speed	Visibility	Solar_Radiation	Rainfall	Snowfall	H
0	-5.2	37	2.2	2000	0.0	0.0	0.0	
1	-5.5	38	0.8	2000	0.0	0.0	0.0	

2 rows × 47 columns

```
y.head(2)
```

```
0    15.937377
1    14.282857
Name: Rented_Bike_Count, dtype: float64
```

```
#Creat test and train data
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size=0.2, random_state=0)
print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(7008, 47)
(1752, 47)
(7008,)
(1752,)
```

```
bike_df_copy.describe().columns
```

```
Index(['Rented_Bike_Count', 'Temperature', 'Humidity', 'Wind_speed',
       'Visibility', 'Solar_Radiation', 'Rainfall', 'Snowfall', 'Hour_1',
       'Hour_2', 'Hour_3', 'Hour_4', 'Hour_5', 'Hour_6', 'Hour_7', 'Hour_8',
       'Hour_9', 'Hour_10', 'Hour_11', 'Hour_12', 'Hour_13', 'Hour_14',
       'Hour_15', 'Hour_16', 'Hour_17', 'Hour_18', 'Hour_19', 'Hour_20',
       'Hour_21', 'Hour_22', 'Hour_23', 'Seasons_Spring', 'Seasons_Summer',
       'Seasons_Winter', 'Holiday_No Holiday', 'Functioning_Day_Yes',
       'month_2', 'month_3', 'month_4', 'month_5', 'month_6', 'month_7',
       'month_8', 'month_9', 'month_10', 'month_11', 'month_12',
       'weekdays_weekend_1'],
      dtype='object')
```

- The mean squared error (MSE) tells you how close a regression line is to a set of points. It does this by taking the distances from the points to the regression line (these distances are the “errors”) and squaring them. It’s called the mean squared error as you’re finding the average of a set of errors. The lower the MSE, the better the forecast.
- MSE formula = (1/n) * Σ(actual − forecast)2 Where:
- n = number of items,
- Σ = summation notation,
- Actual = original or observed y-value,
- Forecast = y-value from regression.
- Root Mean Square Error (RMSE) is the standard deviation of the residuals (prediction errors).

- Mean Absolute Error (MAE) are metrics used to evaluate a Regression Model. ... Here, errors are the differences between the predicted values (values predicted by our regression model) and the actual values of a variable.
- R-squared (R2) is a statistical measure that represents the proportion of the variance for a dependent variable that's explained by an independent variable or variables in a regression model.
- Formula for R-Squared

$$\mathrm{R}^2 = 1 - \frac{\text{Unexplained Variation}}{\text{Total Variation}}$$

- R 2 =1− Total Variation Unexplained Variation
- Adjusted R-squared is a modified version of R-squared that has been adjusted for the number of predictors in the model.

LINEAR REGRESSION

Regression models describe the relationship between variables by fitting a line to the observed data. Linear regression models use a straight line

Linear regression uses a linear approach to model the relationship between independent and dependent variables. In simple words its a best fit line drawn over the values of independent variables and dependent variable. In case of single variable, the formula is same as straight line equation having an intercept and slope.

$$\text{y\_pred} = \beta_0 + \beta_1 x$$

where

$$\beta_0 \text{ and } \beta_1$$

are intercept and slope respectively.

In case of multiple features the formula translates into:

$$\text{y\_pred} = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3 + \ldots..$$

where x_1,x_2,x_3 are the features values and

$$\beta_0, \beta_1, \beta_2 \ldots..$$

are weights assigned to each of the features. These become the parameters which the algorithm tries to learn using Gradient descent.

Gradient descent is the process by which the algorithm tries to update the parameters using a loss function . Loss function is nothing but the diffence between the actual values and predicted values(aka error or residuals). There are different types of loss function but this is the simplest one. Loss function summed over all observation gives the cost functions. The role of gradient descent is to update the parameters till the cost function is minimized i.e., a global minima is reached. It uses a hyperparameter 'alpha' that gives a weightage to the cost function and decides on how big the steps to take. Alpha is called as the learning rate. It is always necesarry to keep an optimal value of alpha as high and low values of alpha might make the gradient descent overshoot or get stuck at a local minima. There are also some basic assumptions that must be fulfilled before implementing this algorithm. They are:

1. No multicollinearity in the dataset.
2. Independent variables should show linear relationship with dv.
3. Residual mean should be 0 or close to 0.
4. There should be no heteroscedasticity i.e., variance should be constant along the line of best fit.

Let us now implement our first model. We will be using LinearRegression from scikit library.

```
#import the packages
from sklearn.linear_model import LinearRegression
reg= LinearRegression().fit(X_train, y_train)
```

```
#check the score
reg.score(X_train, y_train)
```

```
0.7745527328667352
```

```
#check the coefficeint
reg.coef_
```

```
array([ 5.13015001e-01, -1.22767835e-01, -4.13109228e-02,  1.11126068e-03,
        9.16727254e-01, -1.53255669e+00, -8.07225695e-02, -1.94872364e+00,
       -4.88230776e+00, -7.21855433e+00, -9.45852159e+00, -9.09121756e+00,
       -4.05091676e+00,  2.20684979e+00,  7.54238202e+00,  1.32579600e+00,
       -3.16495770e+00, -3.43098887e+00, -2.72035456e+00, -2.91898256e+00,
       -2.94598172e+00, -1.84312764e+00,  1.20075997e-01,  3.83601663e+00,
        1.03636669e+01,  6.80210980e+00,  6.02819334e+00,  6.32722762e+00,
        5.10059921e+00,  1.55360459e+00, -5.40167750e+09, -1.01218304e+10,
       -7.41235754e+09,  3.46321882e+00,  2.83463702e+01, -8.43494814e-01,
       -2.01068004e+09, -2.01068004e+09, -2.01068004e+09,  2.70947282e+09,
        2.70947281e+09,  2.70947281e+09, -7.41235754e+09, -7.41235753e+09,
       -7.41235753e+09,  2.05547188e+00, -1.44324643e+00])
```

```
#get the X_train and X-test value
y_pred_train=reg.predict(X_train)
y_pred_test=reg.predict(X_test)


#import the packages
from sklearn.metrics import mean_squared_error
#calculate MSE
MSE_lr= mean_squared_error((y_train), (y_pred_train))
print("MSE :",MSE_lr)

#calculate RMSE
RMSE_lr=np.sqrt(MSE_lr)
print("RMSE :",RMSE_lr)


#calculate MAE
MAE_lr= mean_absolute_error(y_train, y_pred_train)
print("MAE :",MAE_lr)



#import the packages
from sklearn.metrics import r2_score
#calculate r2 and adjusted r2
r2_lr= r2_score(y_train, y_pred_train)
print("R2 :",r2_lr)
Adjusted_R2_lr = (1-(1-r2_score(y_train, y_pred_train))*((X_test.shape[0]-1)/(X_test.shape[0]
print("Adjusted R2 :",1-(1-r2_score(y_train, y_pred_train))*((X_test.shape[0]-1)/(X_test.shap


     MSE : 34.79398073042247
     RMSE : 5.89864227856059
     MAE : 4.459079570875118
     R2 : 0.7745527328667352
     Adjusted R2 : 0.7683344103577778
```

Looks like our r2 score value is 0.77 that means our model is able to capture most of the data variance. Lets save it in a dataframe for later comparisons.

```
# storing the test set metrics value in a dataframe for later comparison
dict1={'Model':'Linear regression ',
       'MAE':round((MAE_lr),3),
       'MSE':round((MSE_lr),3),
       'RMSE':round((RMSE_lr),3),
       'R2_score':round((r2_lr),3),
       'Adjusted R2':round((Adjusted_R2_lr ),2)
       }
training_df=pd.DataFrame(dict1,index=[1])
```

```
training_df.head(1)
```

	Model	MAE	MSE	RMSE	R2_score	Adjusted R2
1	Linear regression	4.459	34.794	5.899	0.775	0.77

```
#import the packages
from sklearn.metrics import mean_squared_error
#calculate MSE
MSE_lr= mean_squared_error(y_test, y_pred_test)
print("MSE :",MSE_lr)

#calculate RMSE
RMSE_lr=np.sqrt(MSE_lr)
print("RMSE :",RMSE_lr)


#calculate MAE
MAE_lr= mean_absolute_error(y_test, y_pred_test)
print("MAE :",MAE_lr)


#import the packages
from sklearn.metrics import r2_score
#calculate r2 and adjusted r2
r2_lr= r2_score((y_test), (y_pred_test))
print("R2 :",r2_lr)
Adjusted_R2_lr = (1-(1-r2_score((y_test), (y_pred_test)))*((X_test.shape[0]-1)/(X_test.shape[
print("Adjusted R2 :",Adjusted_R2_lr )
```

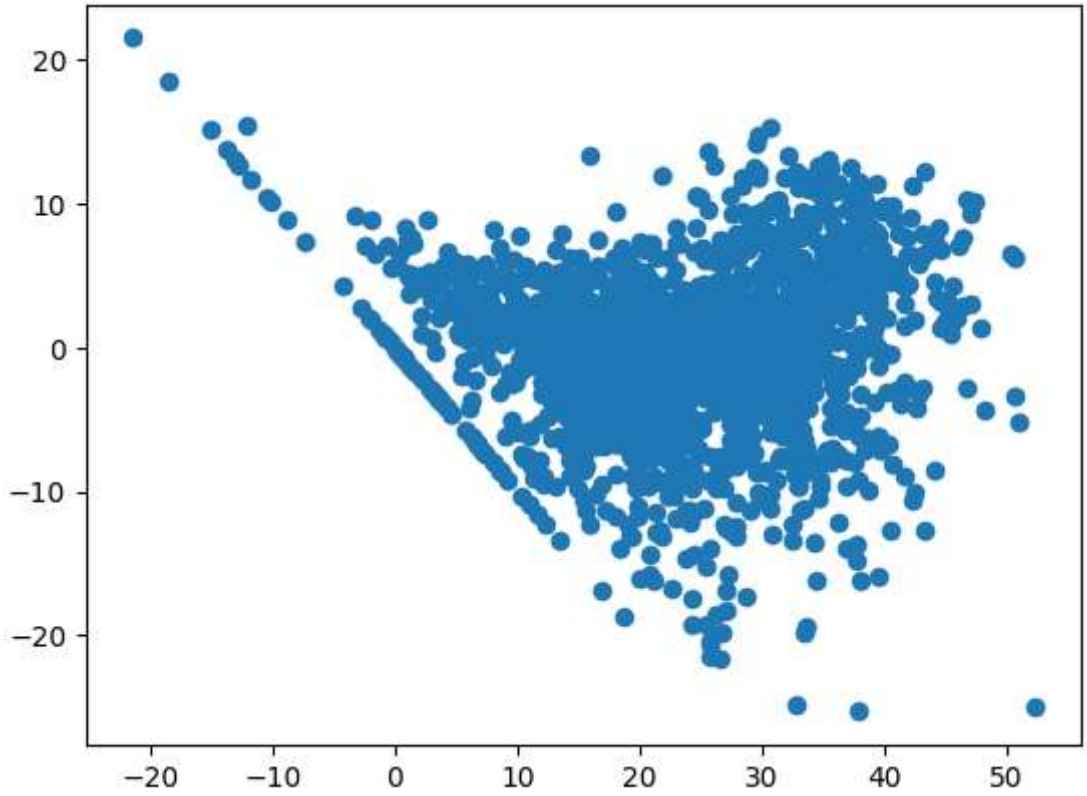
```
MSE : 33.894123347487565
RMSE : 5.821865967839483
MAE : 4.442356971270003
R2 : 0.7847804376397125
Adjusted R2 : 0.7788442173163947
```

The r2_score for the test set is 0.78. This means our linear model is performing well on the data. Let us try to visualize our residuals and see if there is heteroscedasticity(unequal variance or scatter).

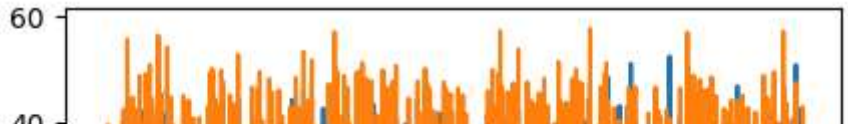
```
# storing the test set metrics value in a dataframe for later comparison
dict2={'Model':'Linear regression ',
       'MAE':round((MAE_lr),3),
       'MSE':round((MSE_lr),3),
       'RMSE':round((RMSE_lr),3),
       'R2_score':round((r2_lr),3),
       'Adjusted R2':round((Adjusted_R2_lr ),2)
```

```
        }
test_df=pd.DataFrame(dict2,index=[1])


### Heteroscadacity
plt.scatter((y_pred_test),(y_test)-(y_pred_test))
```

```
<matplotlib.collections.PathCollection at 0x7f7689804df0>
```

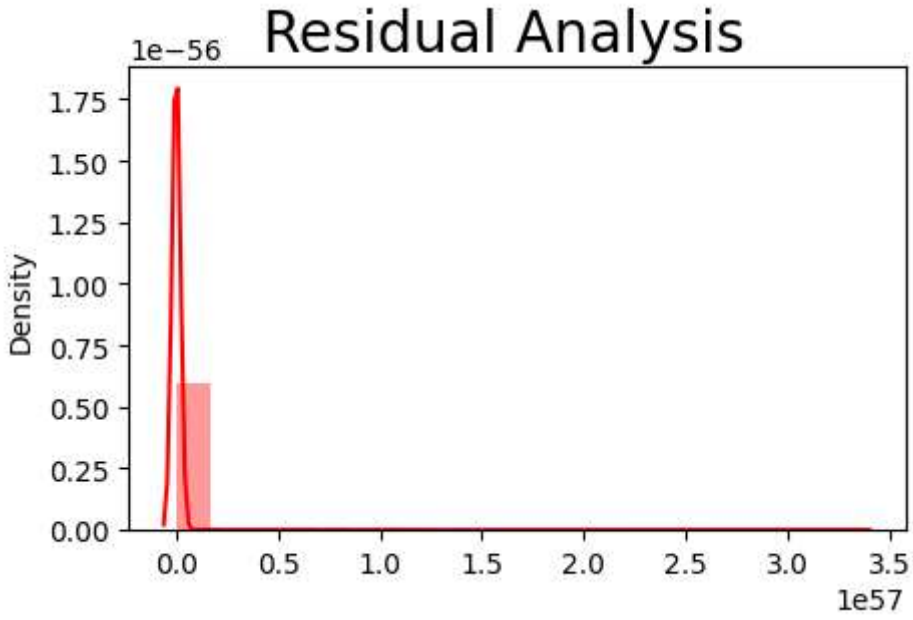



```
#Plot the figure
plt.figure(figsize=(5,3))
plt.plot(y_pred_test)
plt.plot(np.array(y_test))
plt.legend(["Predicted","Actual"])
plt.xlabel('No of Test Data')
plt.show()
```




```
#Resuldual Analysis
fig=plt.figure(figsize=(5,3))

sns.distplot((10**np.array(y_test)- 10**(y_pred_test)),bins=20,color='r')

#Plot Label
fig.suptitle('Residual Analysis', fontsize = 20)
```

```
Text(0.5, 0.98, 'Residual Analysis')
```



LASSO REGRESSION

```
# Create an instance of Lasso Regression implementation
from sklearn.linear_model import Lasso
lasso = Lasso(alpha=1.0, max_iter=3000)
# Fit the Lasso model
lasso.fit(X_train, y_train)
# Create the model score
print(lasso.score(X_test, y_test), lasso.score(X_train, y_train))
```

```
0.3873692800799008 0.40519624904934015
```

```
#get the X_train and X-test value
y_pred_train_lasso=lasso.predict(X_train)
y_pred_test_lasso=lasso.predict(X_test)
```

```
from sklearn.metrics import mean_squared_error
#calculate MSE
MSE_l= mean_squared_error((y_train), (y_pred_train_lasso))
print("MSE :",MSE_l)

#calculate RMSE
RMSE_l=np.sqrt(MSE_l)
print("RMSE :",RMSE_l)


#calculate MAE
MAE_l= mean_absolute_error(y_train, y_pred_train_lasso)
print("MAE :",MAE_l)


from sklearn.metrics import r2_score
#calculate r2 and adjusted r2
r2_l= r2_score(y_train, y_pred_train_lasso)
print("R2 :",r2_l)
Adjusted_R2_l = (1-(1-r2_score(y_train, y_pred_train_lasso))*((X_test.shape[0]-1)/(X_test.sha
print("Adjusted R2 :",1-(1-r2_score(y_train, y_pred_train_lasso))*((X_test.shape[0]-1)/(X_tes
```

```
MSE : 91.59423336097032
RMSE : 9.570487623991283
MAE : 7.255041571454952
R2 : 0.40519624904934015
Adjusted R2 : 0.3921449996120475
```

Looks like our r2 score value is 0.40 that means our model is not able to capture most of the data variance. Lets save it in a dataframe for later comparisons.

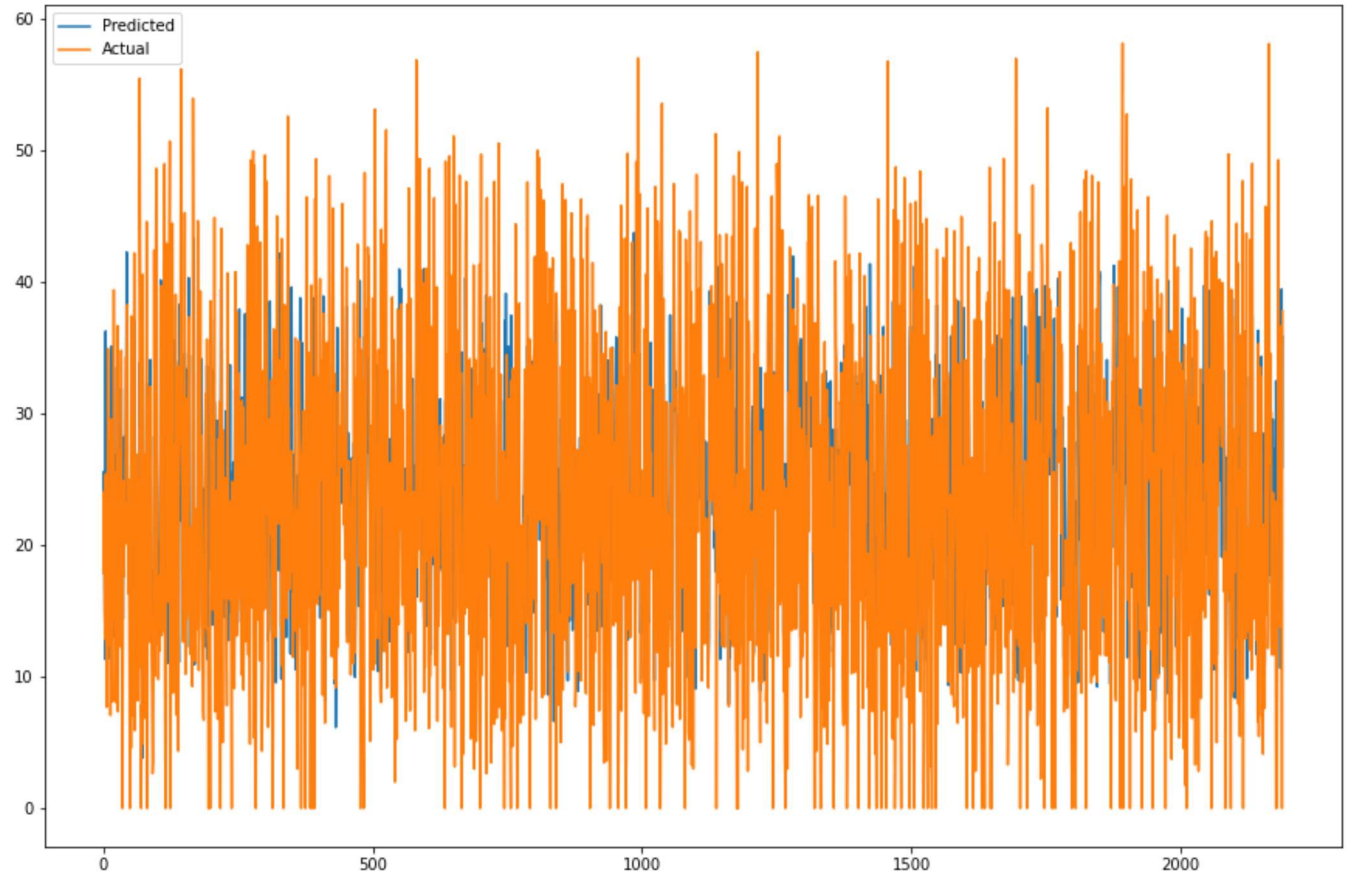
```
# storing the test set metrics value in a dataframe for later comparison
dict1={'Model':'Lasso regression ',
       'MAE':round((MAE_l),3),
       'MSE':round((MSE_l),3),
       'RMSE':round((RMSE_l),3),
       'R2_score':round((r2_l),3),
       'Adjusted R2':round((Adjusted_R2_l ),2)
       }
training_df=training_df.append(dict1,ignore_index=True)


from sklearn.metrics import mean_squared_error
#calculate MSE
MSE_l= mean_squared_error(y_test, y_pred_test_lasso)
print("MSE :",MSE_l)

#calculate RMSE
RMSE_l=np.sqrt(MSE_l)
```

```
print("RMSE :",RMSE_l)


#calculate MAE
MAE_l= mean_absolute_error(y_test, y_pred_test_lasso)
print("MAE :",MAE_l)


from sklearn.metrics import r2_score
#calculate r2 and adjusted r2
r2_l= r2_score((y_test), (y_pred_test_lasso))
print("R2 :",r2_l)
Adjusted_R2_l=(1-(1-r2_score((y_test), (y_pred_test_lasso)))*((X_test.shape[0]-1)/(X_test.sha
print("Adjusted R2 :",1-(1-r2_score((y_test), (y_pred_test_lasso)))*((X_test.shape[0]-1)/(X_t
```

```
MSE : 96.7750714044618
RMSE : 9.837432155011886
MAE : 7.455895061963607
R2 : 0.3873692800799008
Adjusted R2 : 0.37392686932535146
```

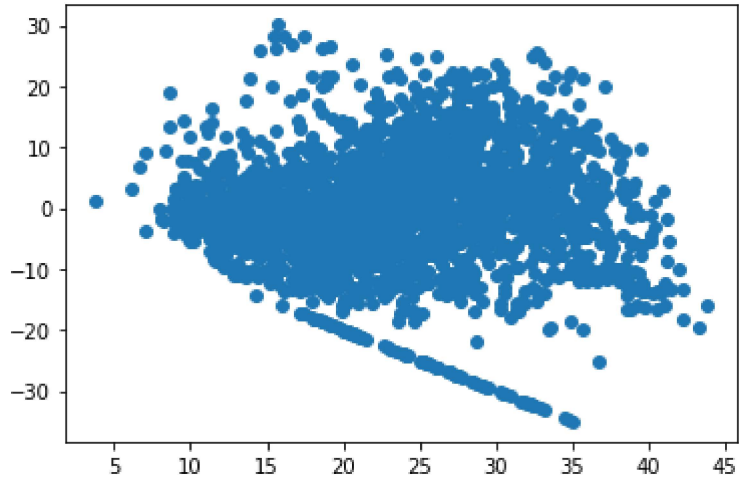
The r2_score for the test set is 0.38. This means our linear model is not performing well on the data. Let us try to visualize our residuals and see if there is heteroscedasticity(unequal variance or scatter).

```
# storing the test set metrics value in a dataframe for later comparison
dict2={'Model':'Lasso regression ',
       'MAE':round((MAE_l),3),
       'MSE':round((MSE_l),3),
       'RMSE':round((RMSE_l),3),
       'R2_score':round((r2_l),3),
       'Adjusted R2':round((Adjusted_R2_l ),2),
       }
test_df=test_df.append(dict2,ignore_index=True)


#Plot the figure
plt.figure(figsize=(15,10))
plt.plot(np.array(y_pred_test_lasso))
plt.plot(np.array((y_test)))
plt.legend(["Predicted","Actual"])
plt.show()
```




```
### Heteroscadacity
plt.scatter((y_pred_test_lasso),(y_test-y_pred_test_lasso))
```

```
<matplotlib.collections.PathCollection at 0x7f982da84ad0>
```



RIDGE REGRESSION

```
#import the packages
from sklearn.linear_model import Ridge
```

```
ridge= Ridge(alpha=0.1)


#FIT THE MODEL
ridge.fit(X_train,y_train)
```

```
     Ridge(alpha=0.1)
```

```
#check the score
ridge.score(X_train, y_train)
```

```
     0.7722100789802107
```

```
#get the X_train and X-test value
y_pred_train_ridge=ridge.predict(X_train)
y_pred_test_ridge=ridge.predict(X_test)


#import the packages
from sklearn.metrics import mean_squared_error
#calculate MSE
MSE_r= mean_squared_error((y_train), (y_pred_train_ridge))
print("MSE :",MSE_r)

#calculate RMSE
RMSE_r=np.sqrt(MSE_r)
print("RMSE :",RMSE_r)


#calculate MAE
MAE_r= mean_absolute_error(y_train, y_pred_train_ridge)
print("MAE :",MAE_r)


#import the packages
from sklearn.metrics import r2_score
#calculate r2 and adjusted r2
r2_r= r2_score(y_train, y_pred_train_ridge)
print("R2 :",r2_r)
Adjusted_R2_r=(1-(1-r2_score(y_train, y_pred_train_ridge))*((X_test.shape[0]-1)/(X_test.shape
print("Adjusted R2 :",1-(1-r2_score(y_train, y_pred_train_ridge))*((X_test.shape[0]-1)/(X_tes
```

```
     MSE : 35.07752456136463
     RMSE : 5.922628180239296
     MAE : 4.474125776125378
     R2 : 0.7722100789802107
     Adjusted R2 : 0.7672118874358922
```

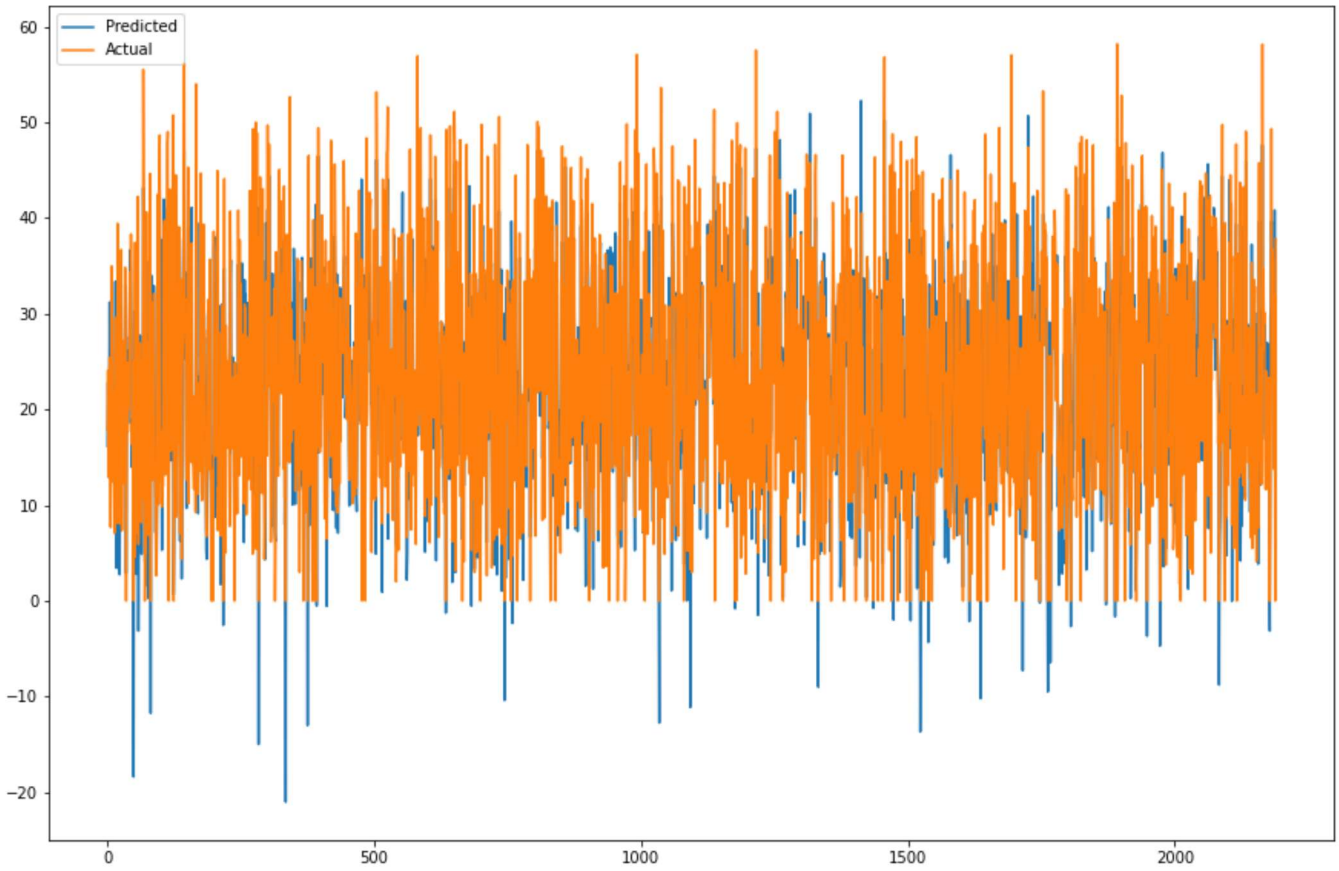
Looks like our r2 score value is 0.77 that means our model is able to capture most of the data variance. Lets save it in a dataframe for later comparisons.

```
# storing the test set metrics value in a dataframe for later comparison
dict1={'Model':'Ridge regression ',
       'MAE':round((MAE_r),3),
       'MSE':round((MSE_r),3),
       'RMSE':round((RMSE_r),3),
       'R2_score':round((r2_r),3),
       'Adjusted R2':round((Adjusted_R2_r ),2)}
training_df=training_df.append(dict1,ignore_index=True)


#import the packages
from sklearn.metrics import mean_squared_error
#calculate MSE
MSE_r= mean_squared_error(y_test, y_pred_test_ridge)
print("MSE :",MSE_r)

#calculate RMSE
RMSE_r=np.sqrt(MSE_r)
print("RMSE :",RMSE_r)


#calculate MAE
MAE_r= mean_absolute_error(y_test, y_pred_test_ridge)
print("MAE :",MAE_r)


#import the packages
from sklearn.metrics import r2_score
#calculate r2 and adjusted r2
r2_r= r2_score((y_test), (y_pred_test_ridge))
print("R2 :",r2_r)
Adjusted_R2_r=(1-(1-r2_score((y_test), (y_pred_test_ridge)))*((X_test.shape[0]-1)/(X_test.sha
print("Adjusted R2 :",1-(1-r2_score((y_test), (y_pred_test_ridge)))*((X_test.shape[0]-1)/(X_t
```

```
MSE : 33.27678426818438
RMSE : 5.768603320404722
MAE : 4.410414932539515
R2 : 0.7893426477812578
Adjusted R2 : 0.7847203809491939
```

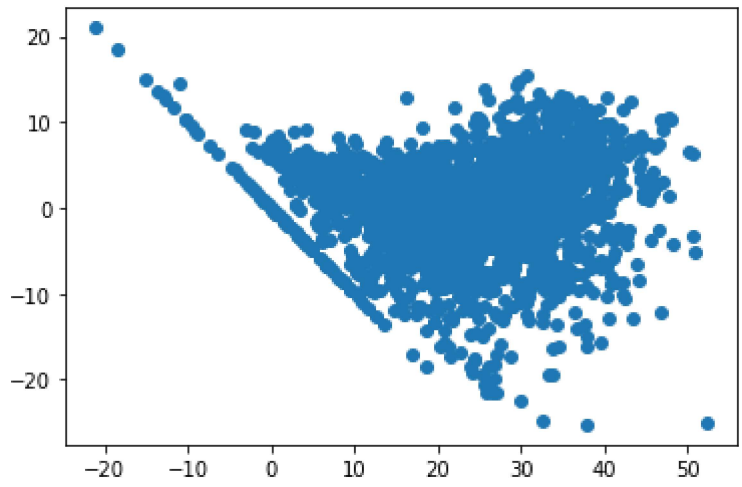
The r2_score for the test set is 0.78. This means our linear model is performing well on the data. Let us try to visualize our residuals and see if there is heteroscedasticity(unequal variance or scatter).

```
# storing the test set metrics value in a dataframe for later comparison
dict2={'Model':'Ridge regression ',
       'MAE':round((MAE_r),3),
       'MSE':round((MSE_r),3),
       'RMSE':round((RMSE_r),3),
       'R2_score':round((r2_r),3),
       'Adjusted R2':round((Adjusted_R2_r ),2)}
test_df=test_df.append(dict2,ignore_index=True)


#Plot the figure
plt.figure(figsize=(15,10))
plt.plot((y_pred_test_ridge))
plt.plot((np.array(y_test)))
plt.legend(["Predicted","Actual"])
plt.show()
```




```
### Heteroscadacity
plt.scatter((y_pred_test_ridge),(y_test)-(y_pred_test_ridge))
```

```
<matplotlib.collections.PathCollection at 0x7f982d98be10>
```



ELASTIC NET REGRESSION

```
#import the packages
from sklearn.linear_model import ElasticNet
#a * L1 + b * L2
#alpha = a + b and l1_ratio = a / (a + b)
elasticnet = ElasticNet(alpha=0.1, l1_ratio=0.5)
```

```
#FIT THE MODEL
elasticnet.fit(X_train,y_train)
```

```
ElasticNet(alpha=0.1)
```

```
#check the score
elasticnet.score(X_train, y_train)
```

```
0.6261189054494012
```

```
#get the X_train and X-test value
y_pred_train_en=elasticnet.predict(X_train)
y_pred_test_en=elasticnet.predict(X_test)
```

```
#import the packages
from sklearn.metrics import mean_squared_error
#calculate MSE
MSE_e= mean_squared_error((y_train), (y_pred_train_en))
print("MSE :",MSE_e)

#calculate RMSE
RMSE_e=np.sqrt(MSE_e)
print("RMSE :",RMSE_e)
```

```
#calculate MAE
MAE_e= mean_absolute_error(y_train, y_pred_train_en)
print("MAE :",MAE_e)


#import the packages
from sklearn.metrics import r2_score
#calculate r2 and adjusted r2
r2_e= r2_score(y_train, y_pred_train_en)
print("R2 :",r2_e)
Adjusted_R2_e=(1-(1-r2_score(y_train, y_pred_train_en))*((X_test.shape[0]-1)/(X_test.shape[0]
print("Adjusted R2 :",1-(1-r2_score(y_train, y_pred_train_en))*((X_test.shape[0]-1)/(X_test.s
```

```
MSE : 57.5742035398887
RMSE : 7.587766703048315
MAE : 5.792276538970546
R2 : 0.6261189054494012
Adjusted R2 : 0.6179151652795234
```

Looks like our r2 score value is 0.62 that means our model is able to capture most of the data variance. Lets save it in a dataframe for later comparisons.

```
# storing the test set metrics value in a dataframe for later comparison
dict1={'Model':'Elastic net regression ',
       'MAE':round((MAE_e),3),
       'MSE':round((MSE_e),3),
       'RMSE':round((RMSE_e),3),
       'R2_score':round((r2_e),3),
       'Adjusted R2':round((Adjusted_R2_e ),2)}
training_df=training_df.append(dict1,ignore_index=True)


#import the packages
from sklearn.metrics import mean_squared_error
#calculate MSE
MSE_e= mean_squared_error(y_test, y_pred_test_en)
print("MSE :",MSE_e)

#calculate RMSE
RMSE_e=np.sqrt(MSE_e)
print("RMSE :",RMSE_e)


#calculate MAE
MAE_e= mean_absolute_error(y_test, y_pred_test_en)
print("MAE :",MAE_e)
```

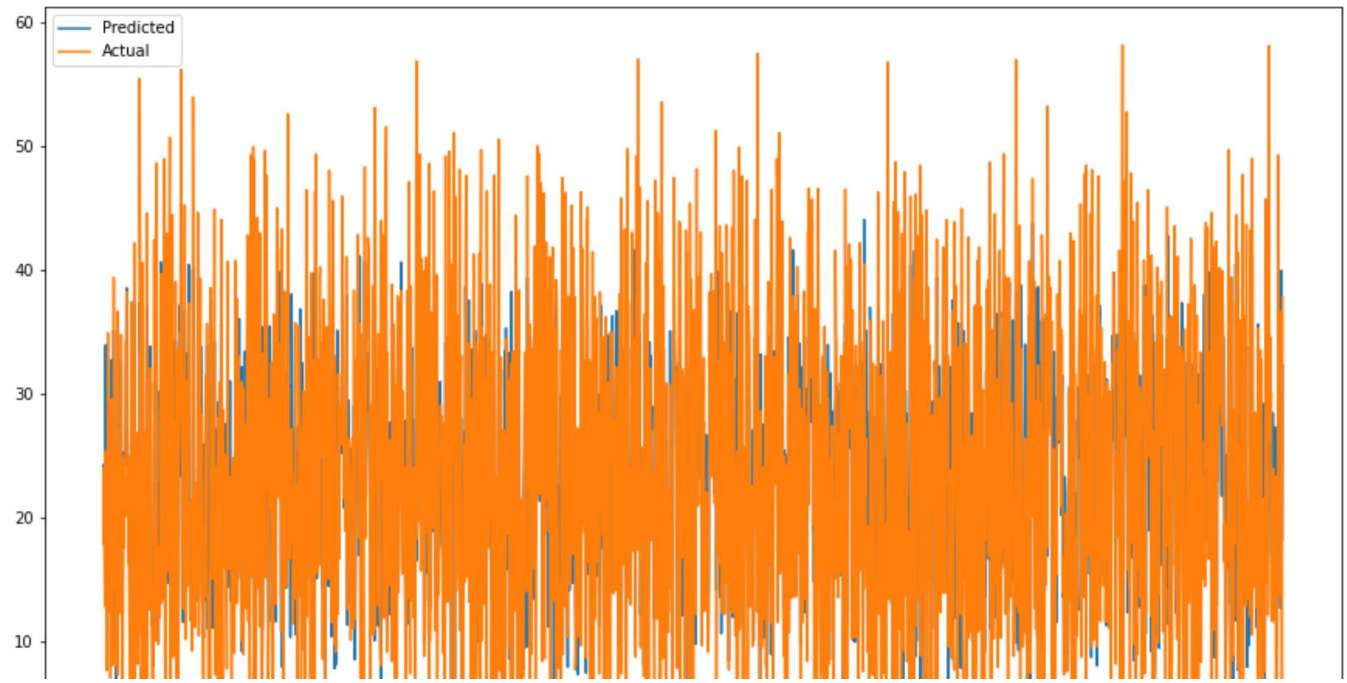
```
#import the packages
from sklearn.metrics import r2_score
#calculate r2 and adjusted r2
r2_e= r2_score((y_test), (y_pred_test_en))
print("R2 :",r2_e)
Adjusted_R2_e=(1-(1-r2_score((y_test), (y_pred_test_en)))*((X_test.shape[0]-1)/(X_test.shape[
print("Adjusted R2 :",1-(1-r2_score((y_test), (y_pred_test_en)))*((X_test.shape[0]-1)/(X_test
```

```
MSE : 59.45120536350042
RMSE : 7.710460775044538
MAE : 5.873612334800099
R2 : 0.6236465216363589
Adjusted R2 : 0.6153885321484546
```

The r2_score for the test set is 0.86. This means our linear model is performing well on the data. Let us try to visualize our residuals and see if there is heteroscedasticity(unequal variance or scatter).

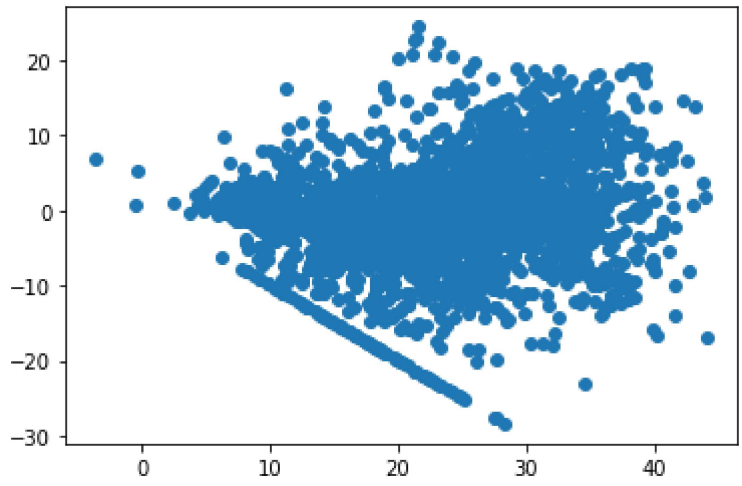
```
# storing the test set metrics value in a dataframe for later comparison
dict2={'Model':'Elastic net regression Test',
       'MAE':round((MAE_e),3),
       'MSE':round((MSE_e),3),
       'RMSE':round((RMSE_e),3),
       'R2_score':round((r2_e),3),
       'Adjusted R2':round((Adjusted_R2_e ),2)}
test_df=test_df.append(dict2,ignore_index=True)
```

```
#Plot the figure
plt.figure(figsize=(15,10))
plt.plot(np.array(y_pred_test_en))
plt.plot((np.array(y_test)))
plt.legend(["Predicted","Actual"])
plt.show()
```




```
### Heteroscadacity
plt.scatter((y_pred_test_en),(y_test)-(y_pred_test_en))
```

```
<matplotlib.collections.PathCollection at 0x7f982d88f7d0>
```



DECISION TREE

```
#import the packages
from sklearn.tree import DecisionTreeRegressor
decision_regressor = DecisionTreeRegressor(criterion='mse', max_depth=8,
                      max_features=9, max_leaf_nodes=100,)
decision_regressor.fit(X_train, y_train)
```

```
DecisionTreeRegressor(criterion='mse', max_depth=8, max_features=9,
                      max_leaf_nodes=100)
```

```
#get the X_train and X-test value
y_pred_train_d = decision_regressor.predict(X_train)
y_pred_test_d = decision_regressor.predict(X_test)


#import the packages
from sklearn.metrics import mean_squared_error
print("Model Score:",decision_regressor.score(X_train,y_train))

#calculate MSE
MSE_d= mean_squared_error(y_train, y_pred_train_d)
print("MSE :",MSE_d)

#calculate RMSE
RMSE_d=np.sqrt(MSE_d)
print("RMSE :",RMSE_d)


#calculate MAE
MAE_d= mean_absolute_error(y_train, y_pred_train_d)
print("MAE :",MAE_d)


#import the packages
from sklearn.metrics import r2_score
#calculate r2 and adjusted r2
r2_d= r2_score(y_train, y_pred_train_d)
print("R2 :",r2_d)
Adjusted_R2_d=(1-(1-r2_score(y_train, y_pred_train_d))*((X_test.shape[0]-1)/(X_test.shape[0]-
print("Adjusted R2 :",1-(1-r2_score(y_train, y_pred_train_d))*((X_test.shape[0]-1)/(X_test.sh
```

```
Model Score: 0.6799535533778143
MSE : 49.28416956249743
RMSE : 7.020268482223271
MAE : 5.170334160033761
R2 : 0.6799535533778143
Adjusted R2 : 0.6729310589841435
```

Looks like our r2 score value is 0.69 that means our model is able to capture most of the data variance. Lets save it in a dataframe for later comparisons.

```
# storing the test set metrics value in a dataframe for later comparison
dict1={'Model':'Dicision tree regression ',
       'MAE':round((MAE_d),3),
       'MSE':round((MSE_d),3),
       'RMSE':round((RMSE_d),3),
       'R2_score':round((r2_d),3),
       'Adjusted R2':round((Adjusted_R2_d),2)
```

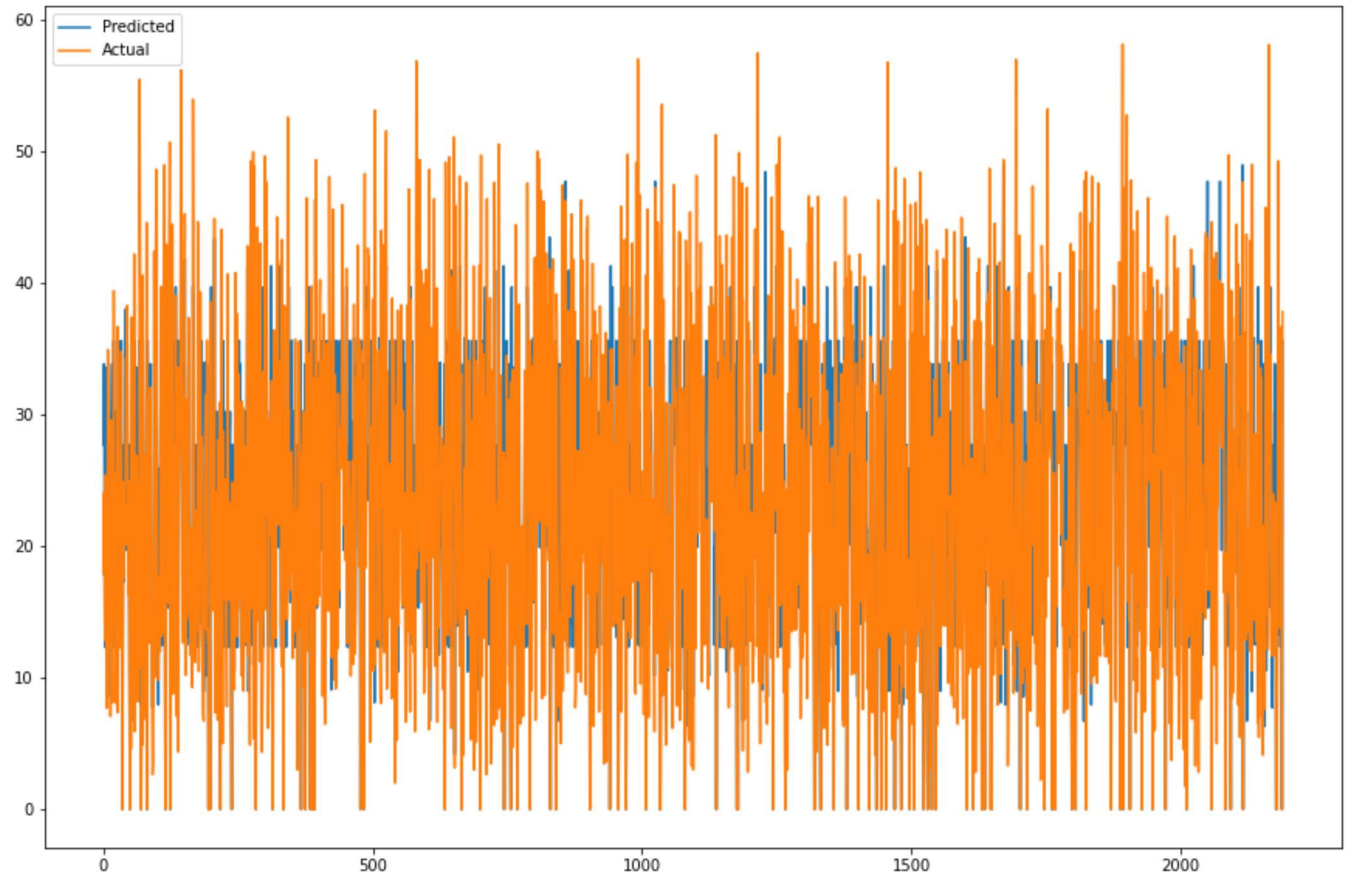
```
      }
training_df=training_df.append(dict1,ignore_index=True)


#import the packages
from sklearn.metrics import mean_squared_error
#calculate MSE
MSE_d= mean_squared_error(y_test, y_pred_test_d)
print("MSE :",MSE_d)

#calculate RMSE
RMSE_d=np.sqrt(MSE_d)
print("RMSE :",RMSE_d)


#calculate MAE
MAE_d= mean_absolute_error(y_test, y_pred_test_d)
print("MAE :",MAE_d)


#import the packages
from sklearn.metrics import r2_score
#calculate r2 and adjusted r2
r2_d= r2_score((y_test), (y_pred_test_d))
print("R2 :",r2_d)
Adjusted_R2_d=(1-(1-r2_score((y_test), (y_pred_test_d)))*((X_test.shape[0]-1)/(X_test.shape[0
print("Adjusted R2 :",1-(1-r2_score((y_test), (y_pred_test_d)))*((X_test.shape[0]-1)/(X_test.
```

```
MSE : 55.04537750487864
RMSE : 7.419257207084725
MAE : 5.454578699879592
R2 : 0.6515374387258508
Adjusted R2 : 0.643891434813673
```

The r2_score for the test set is 0.66. This means our linear model is performing well on the data. Let us try to visualize our residuals and see if there is heteroscedasticity(unequal variance or scatter).

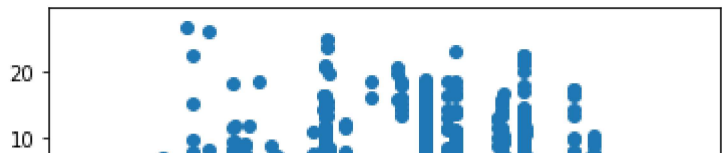
```
# storing the test set metrics value in a dataframe for later comparison
dict2={'Model':'Dicision tree regression ',
       'MAE':round((MAE_d),3),
       'MSE':round((MSE_d),3),
       'RMSE':round((RMSE_d),3),
       'R2_score':round((r2_d),3),
       'Adjusted R2':round((Adjusted_R2_d),2)
      }
test_df=test_df.append(dict2,ignore_index=True)
```

```
#Plot the figure
plt.figure(figsize=(15,10))
plt.plot((np.array(y_pred_test_d)))
plt.plot(np.array((y_test)))
plt.legend(["Predicted","Actual"])
plt.show()
```




```
### Heteroscadacity
plt.scatter((y_pred_test_d),(y_test)-(y_pred_test_d))
```

```
<matplotlib.collections.PathCollection at 0x7f982d770e50>
```



RANDOM FOREST

```
#import the packages
from sklearn.ensemble import RandomForestRegressor
# Create an instance of the RandomForestRegressor
rf_model = RandomForestRegressor()

rf_model.fit(X_train,y_train)
```

```
RandomForestRegressor()
```

```
# Making predictions on train and test data

y_pred_train_r = rf_model.predict(X_train)
y_pred_test_r = rf_model.predict(X_test)



#import the packages
from sklearn.metrics import mean_squared_error
print("Model Score:",rf_model.score(X_train,y_train))

#calculate MSE
MSE_rf= mean_squared_error(y_train, y_pred_train_r)
print("MSE :",MSE_rf)

#calculate RMSE
RMSE_rf=np.sqrt(MSE_rf)
print("RMSE :",RMSE_rf)


#calculate MAE
MAE_rf= mean_absolute_error(y_train, y_pred_train_r)
print("MAE :",MAE_rf)


#import the packages
from sklearn.metrics import r2_score
#calculate r2 and adjusted r2
r2_rf= r2_score(y_train, y_pred_train_r)
print("R2 :",r2_rf)
Adjusted_R2_rf=(1-(1-r2_score(y_train, y_pred_train_r))*((X_test.shape[0]-1)/(X_test.shape[0]
```

```
print("Adjusted R2 :",1-(1-r2_score(y_train, y_pred_train_r))*((X_test.shape[0]-1)/(X_test.sh
```

```
Model Score: 0.9895620801374047
MSE : 1.6073423648886425
RMSE : 1.2678100665670085
MAE : 0.8064650602500485
R2 : 0.9895620801374047
Adjusted R2 : 0.9893330501497567
```

Looks like our r2 score value is 0.98 that means our model is able to capture most of the data variance. Lets save it in a dataframe for later comparisons.

```
# storing the test set metrics value in a dataframe for later comparison
dict1={'Model':'Random forest regression ',
       'MAE':round((MAE_rf),3),
       'MSE':round((MSE_rf),3),
       'RMSE':round((RMSE_rf),3),
       'R2_score':round((r2_rf),3),
       'Adjusted R2':round((Adjusted_R2_rf ),2)}
training_df=training_df.append(dict1,ignore_index=True)
```

```
#import the packages
from sklearn.metrics import mean_squared_error
#calculate MSE
MSE_rf= mean_squared_error(y_test, y_pred_test_r)
print("MSE :",MSE_rf)

#calculate RMSE
RMSE_rf=np.sqrt(MSE_rf)
print("RMSE :",RMSE_rf)
```

```
#calculate MAE
MAE_rf= mean_absolute_error(y_test, y_pred_test_r)
print("MAE :",MAE_rf)
```

```
#import the packages
from sklearn.metrics import r2_score
#calculate r2 and adjusted r2
r2_rf= r2_score((y_test), (y_pred_test_r))
print("R2 :",r2_rf)
Adjusted_R2_rf=(1-(1-r2_score((y_test), (y_pred_test_r)))*((X_test.shape[0]-1)/(X_test.shape[
print("Adjusted R2 :",1-(1-r2_score((y_test), (y_pred_test_r)))*((X_test.shape[0]-1)/(X_test.
```

```
MSE : 12.587221457793135
RMSE : 3.5478474400392606
```

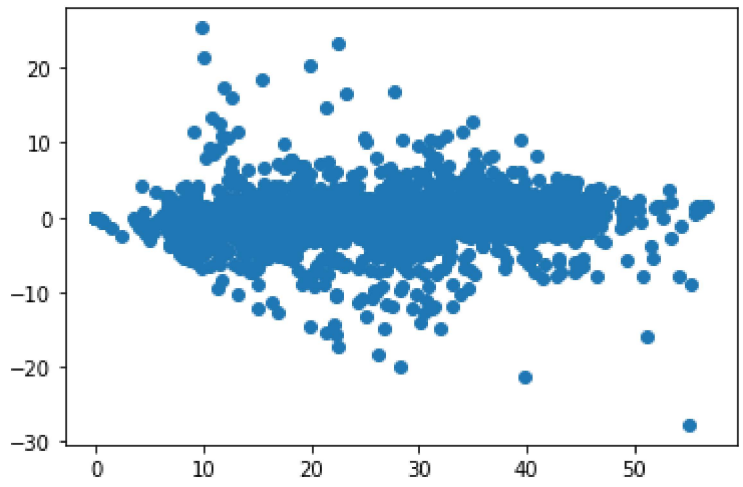
```
MAE : 2.2041627237103874
R2 : 0.9203170978685943
Adjusted R2 : 0.9185686868507716
```

The r2_score for the test set is 0.91. This means our linear model is performing well on the data. Let us try to visualize our residuals and see if there is heteroscedasticity(unequal variance or scatter).

```
# storing the test set metrics value in a dataframe for later comparison
dict2={'Model':'Random forest regression ',
       'MAE':round((MAE_rf),3),
       'MSE':round((MSE_rf),3),
       'RMSE':round((RMSE_rf),3),
       'R2_score':round((r2_rf),3),
       'Adjusted R2':round((Adjusted_R2_rf ),2)}
test_df=test_df.append(dict2,ignore_index=True)
```

```
### Heteroscadacity
plt.scatter((y_pred_test_r),(y_test)-(y_pred_test_r))
```

```
<matplotlib.collections.PathCollection at 0x7f982d770d50>
```




```
rf_model.feature_importances_
```

```
array([3.13095630e-01, 1.57425188e-01, 1.22506816e-02, 1.22065003e-02,
       3.09394663e-02, 3.36502329e-02, 1.45646961e-03, 4.75517271e-03,
       1.30784296e-02, 2.04903580e-02, 2.91665696e-02, 2.53530332e-02,
       7.97967533e-03, 3.99792916e-03, 1.39915183e-02, 1.05274131e-03,
       2.24460807e-03, 1.08132671e-03, 2.70569838e-04, 2.44446694e-04,
       3.89561067e-04, 1.54548696e-03, 3.79327133e-03, 1.03562376e-02,
       3.08757150e-02, 1.76195906e-02, 1.26537709e-02, 1.28310825e-02,
       1.00024903e-02, 4.23726947e-03, 4.29130093e-03, 8.07916762e-04,
       9.09621026e-03, 3.94657674e-03, 1.52104386e-01, 3.55800961e-04,
       3.86549510e-03, 1.27669276e-03, 1.67286673e-03, 2.34594474e-03,
       6.75963988e-04, 2.24619101e-03, 8.97728750e-04, 2.98259756e-03,
       4.92694727e-03, 7.84784654e-04, 1.86875727e-02])
```

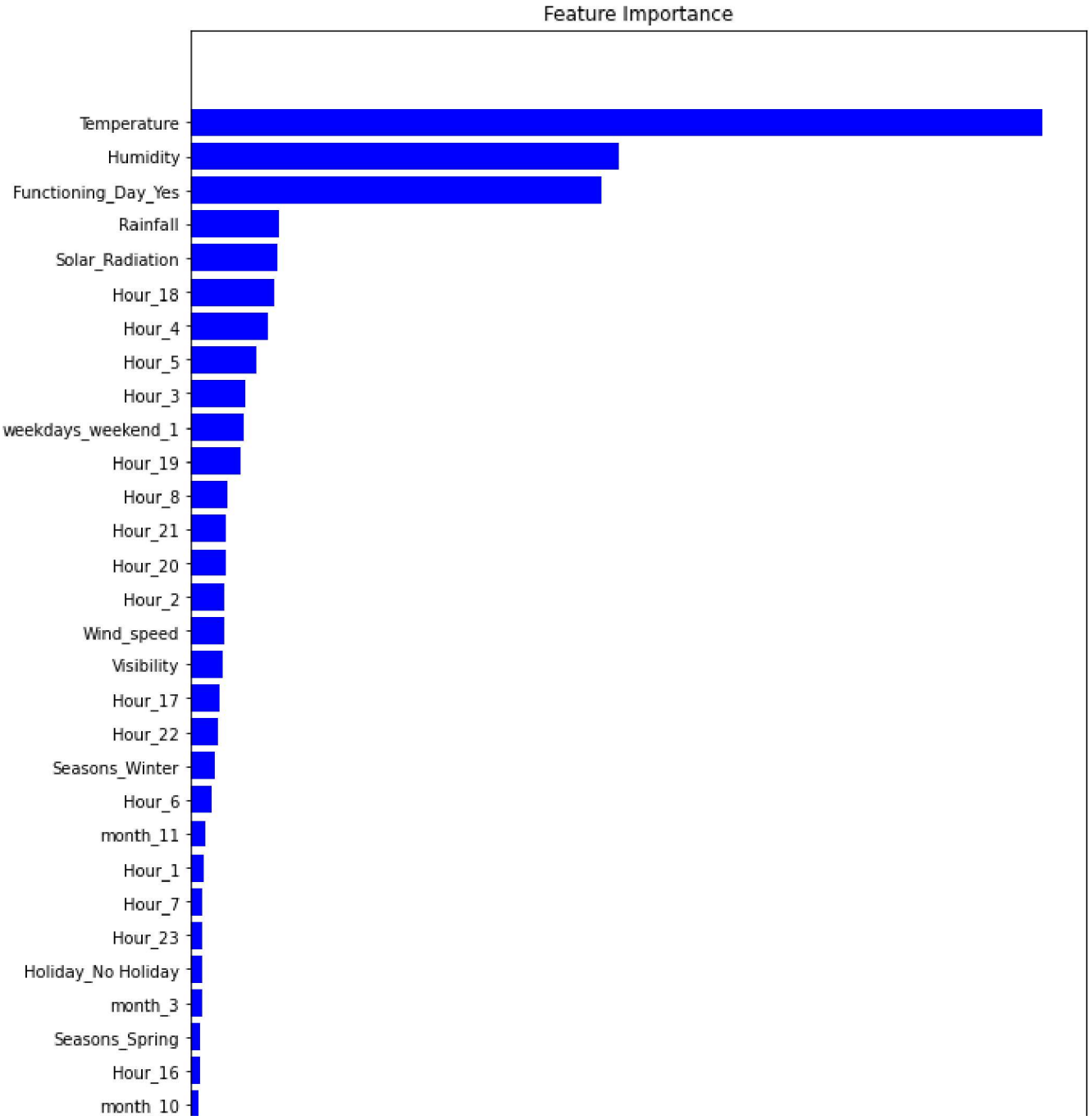
```
importances = rf_model.feature_importances_

importance_dict = {'Feature' : list(X_train.columns),
                   'Feature Importance' : importances}

importance_df = pd.DataFrame(importance_dict)


importance_df['Feature Importance'] = round(importance_df['Feature Importance'],2)


importance_df.sort_values(by=['Feature Importance'],ascending=False)
```

	Feature	Feature Importance
0	Temperature	0.31
1	Humidity	0.16
34	Functioning_Day_Yes	0.15
10	Hour_4	0.03
4	Solar_Radiation	0.03
5	Rainfall	0.03
24	Hour_18	0.03
11	Hour_5	0.03

```
#FIT THE MODEL
rf_model.fit(X_train,y_train)
```

```
RandomForestRegressor()
```

```
features = X_train.columns
importances = rf_model.feature_importances_
indices = np.argsort(importances)


#Plot the figure
plt.figure(figsize=(10,20))
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.show()
```



GRADIENT BOOSTING

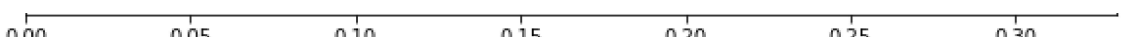
Snowfall

```
#import the packages
from sklearn.ensemble import GradientBoostingRegressor
# Create an instance of the GradientBoostingRegressor
gb_model = GradientBoostingRegressor()


gb_model.fit(X_train,y_train)
```

```
GradientBoostingRegressor()
```

Hour_14

```
# Making predictions on train and test data

y_pred_train_g = gb_model.predict(X_train)
y_pred_test_g = gb_model.predict(X_test)
```




```
#import the packages
from sklearn.metrics import mean_squared_error
print("Model Score:",gb_model.score(X_train,y_train))
#calculate MSE
MSE_gb= mean_squared_error(y_train, y_pred_train_g)
print("MSE :",MSE_gb)

#calculate RMSE
RMSE_gb=np.sqrt(MSE_gb)
print("RMSE :",RMSE_gb)


#calculate MAE
MAE_gb= mean_absolute_error(y_train, y_pred_train_g)
print("MAE :",MAE_gb)


#import the packages
from sklearn.metrics import r2_score
#calculate r2 and adjusted r2
r2_gb= r2_score(y_train, y_pred_train_g)
print("R2 :",r2_gb)
Adjusted_R2_gb = (1-(1-r2_score(y_train, y_pred_train_g))*((X_test.shape[0]-1)/(X_test.shape[
print("Adjusted R2 :",1-(1-r2_score(y_train, y_pred_train_g))*((X_test.shape[0]-1)/(X_test.sh


     Model Score: 0.8789016499095264
     MSE : 18.64801713184794
     RMSE : 4.3183349953249275
     MAE : 3.2690035692731247
     R2 : 0.8789016499095264
     Adjusted R2 : 0.8762444965695393
```

Looks like our r2 score value is 0.87 that means our model is able to capture most of the data variance. Lets save it in a dataframe for later comparisons.

```
# storing the test set metrics value in a dataframe for later comparison
dict1={'Model':'Gradient boosting regression ',
       'MAE':round((MAE_gb),3),
       'MSE':round((MSE_gb),3),
       'RMSE':round((RMSE_gb),3),
       'R2_score':round((r2_gb),3),
       'Adjusted R2':round((Adjusted_R2_gb ),2),
```

```
        }
training_df=training_df.append(dict1,ignore_index=True)


#import the packages
from sklearn.metrics import mean_squared_error
#calculate MSE
MSE_gb= mean_squared_error(y_test, y_pred_test_g)
print("MSE :",MSE_gb)

#calculate RMSE
RMSE_gb=np.sqrt(MSE_gb)
print("RMSE :",RMSE_gb)


#calculate MAE
MAE_gb= mean_absolute_error(y_test, y_pred_test_g)
print("MAE :",MAE_gb)


#import the packages
from sklearn.metrics import r2_score
#calculate r2 and adjusted r2
r2_gb= r2_score((y_test), (y_pred_test_g))
print("R2 :",r2_gb)
Adjusted_R2_gb = (1-(1-r2_score((y_test), (y_pred_test_g)))*((X_test.shape[0]-1)/(X_test.shap
print("Adjusted R2 :",1-(1-r2_score((y_test), (y_pred_test_g)))*((X_test.shape[0]-1)/(X_test.
```

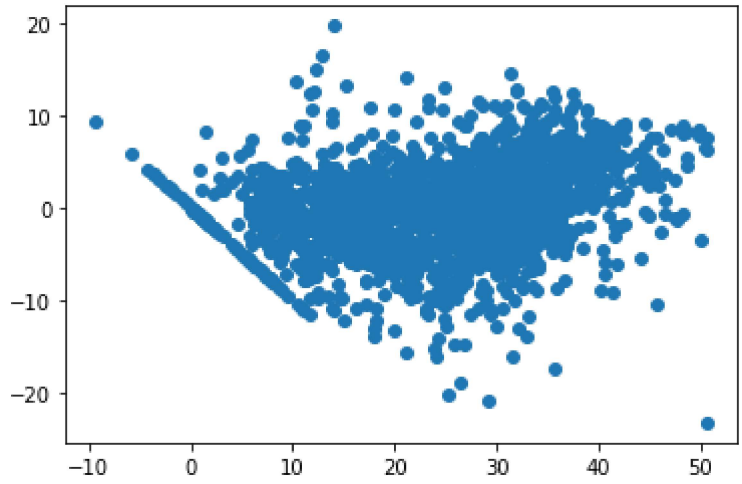
```
MSE : 21.289441842508687
RMSE : 4.614048313846387
MAE : 3.4928587865599914
R2 : 0.8652280396863458
Adjusted R2 : 0.8622708584843188
```

The r2_score for the test set is 0.86. This means our linear model is performing well on the data. Let us try to visualize our residuals and see if there is heteroscedasticity(unequal variance or scatter).

```
# storing the test set metrics value in a dataframe for later comparison
dict2={'Model':'Gradient boosting regression ',
       'MAE':round((MAE_gb),3),
       'MSE':round((MSE_gb),3),
       'RMSE':round((RMSE_gb),3),
       'R2_score':round((r2_gb),3),
       'Adjusted R2':round((Adjusted_R2_gb ),2),
       }
test_df=test_df.append(dict2,ignore_index=True)
```

```
### Heteroscadacity
plt.scatter((y_pred_test_g),(y_test)-(y_pred_test_g))
```

```
<matplotlib.collections.PathCollection at 0x7f982d6f73d0>
```




```
gb_model.feature_importances_
```

```
array([3.21437110e-01, 1.28054219e-01, 5.01908892e-04, 4.45596896e-04,
       4.82210501e-02, 6.50977064e-02, 3.00825496e-05, 4.41679040e-04,
       6.20291012e-03, 1.54723207e-02, 2.77923448e-02, 2.09197623e-02,
       5.54914116e-03, 1.38849071e-03, 9.51964101e-03, 0.00000000e+00,
       2.87933226e-03, 1.37560679e-03, 0.00000000e+00, 0.00000000e+00,
       0.00000000e+00, 0.00000000e+00, 4.08156531e-04, 7.57741782e-03,
       3.03117662e-02, 1.74699066e-02, 1.14240782e-02, 1.55664225e-02,
       1.20487441e-02, 2.45479420e-04, 1.42322866e-05, 0.00000000e+00,
       3.35174157e-02, 2.23294271e-03, 1.74747832e-01, 0.00000000e+00,
       1.72209214e-03, 9.33502112e-05, 2.82254756e-04, 4.18000398e-03,
       0.00000000e+00, 2.66904099e-03, 0.00000000e+00, 6.76468137e-03,
       5.12015005e-03, 2.89720073e-04, 1.79854096e-02])
```

```
importances = gb_model.feature_importances_

importance_dict = {'Feature' : list(X_train.columns),
                   'Feature Importance' : importances}

importance_df = pd.DataFrame(importance_dict)


importance_df['Feature Importance'] = round(importance_df['Feature Importance'],2)


importance_df.head()
```

	Feature	Feature Importance
0	Temperature	0.32
1	Humidity	0.13

```
importance_df.sort_values(by=['Feature Importance'],ascending=False)
```

	Feature	Feature Importance
0	Temperature	0.32
34	Functioning_Day_Yes	0.17
1	Humidity	0.13
5	Rainfall	0.07

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

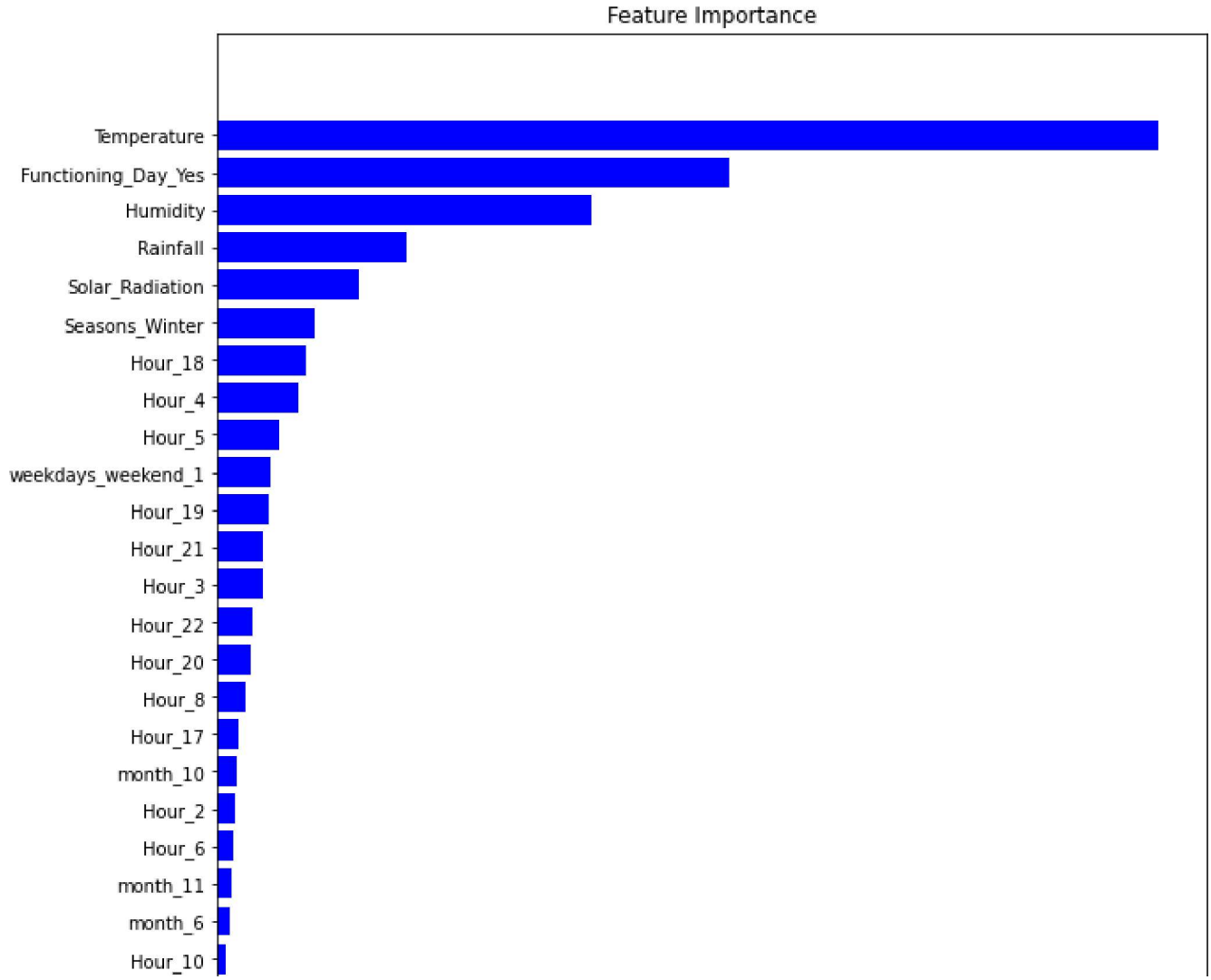
```
GradientBoostingRegressor()
```

24	Hour_18	0.03

```
features = X_train.columns
importances = gb_model.feature_importances_
indices = np.argsort(importances)
```

27	Hour 21	0.02

```
#Plot the figure
plt.figure(figsize=(10,20))
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.show()
```



Hyperparameter tuning

Before proceding to try next models, let us try to tune some hyperparameters and see if the performance of our model improves.

Hyperparameter tuning is the process of choosing a set of optimal hyperparameters for a learning algorithm. A hyperparameter is a model argument whose value is set before the learning process begins. The key to machine learning algorithms is hyperparameter tuning.

Using GridSearchCV

GridSearchCV helps to loop through predefined hyperparameters and fit the model on the training set. So, in the end, we can select the best parameters from the listed hyperparameters.

Gradient Boosting Regressor with GridSearchCV

Provide the range of values for chosen hyperparameters

Relative Importance

```
# Number of trees
n_estimators = [50,80,100]

# Maximum depth of trees
max_depth = [4,6,8]

# Minimum number of samples required to split a node
min_samples_split = [50,100,150]

# Minimum number of samples required at each leaf node
min_samples_leaf = [40,50]

# HYperparameter Grid
param_dict = {'n_estimators' : n_estimators,
              'max_depth' : max_depth,
              'min_samples_split' : min_samples_split,
              'min_samples_leaf' : min_samples_leaf}


param_dict
```

```
{'max_depth': [4, 6, 8],
 'min_samples_leaf': [40, 50],
 'min_samples_split': [50, 100, 150],
 'n_estimators': [50, 80, 100]}
```

Importing Gradient Boosting Regressor

```
from sklearn.model_selection import GridSearchCV
# Create an instance of the GradientBoostingRegressor
gb_model = GradientBoostingRegressor()

# Grid search
gb_grid = GridSearchCV(estimator=gb_model,
                       param_grid = param_dict,
                       cv = 5, verbose=2)

gb_grid.fit(X_train,y_train)
```

```
Fitting 5 folds for each of 54 candidates, totalling 270 fits
[CV] END max_depth=4, min_samples_leaf=40, min_samples_split=50, n_estimators=50; tot
[CV] END max_depth=4, min_samples_leaf=40, min_samples_split=50, n_estimators=50; tot
```

```
[CV] END max_depth=4, min_samples_leaf=40, min_samples_split=50, n_estimators=50; tot
[CV] END max_depth=4, min_samples_leaf=40, min_samples_split=50, n_estimators=50; tot
[CV] END max_depth=4, min_samples_leaf=40, min_samples_split=50, n_estimators=50; tot
[CV] END max_depth=4, min_samples_leaf=40, min_samples_split=50, n_estimators=80; tot
[CV] END max_depth=4, min_samples_leaf=40, min_samples_split=50, n_estimators=80; tot
[CV] END max_depth=4, min_samples_leaf=40, min_samples_split=50, n_estimators=80; tot
[CV] END max_depth=4, min_samples_leaf=40, min_samples_split=50, n_estimators=80; tot
[CV] END max_depth=4, min_samples_leaf=40, min_samples_split=50, n_estimators=80; tot
[CV] END max_depth=4, min_samples_leaf=40, min_samples_split=50, n_estimators=100; to
[CV] END max_depth=4, min_samples_leaf=40, min_samples_split=50, n_estimators=100; to
[CV] END max_depth=4, min_samples_leaf=40, min_samples_split=50, n_estimators=100; to
[CV] END max_depth=4, min_samples_leaf=40, min_samples_split=50, n_estimators=100; to
[CV] END max_depth=4, min_samples_leaf=40, min_samples_split=50, n_estimators=100; to
[CV] END max_depth=4, min_samples_leaf=40, min_samples_split=100, n_estimators=50; to
[CV] END max_depth=4, min_samples_leaf=40, min_samples_split=100, n_estimators=50; to
[CV] END max_depth=4, min_samples_leaf=40, min_samples_split=100, n_estimators=50; to
[CV] END max_depth=4, min_samples_leaf=40, min_samples_split=100, n_estimators=50; to
[CV] END max_depth=4, min_samples_leaf=40, min_samples_split=100, n_estimators=50; to
[CV] END max_depth=4, min_samples_leaf=40, min_samples_split=100, n_estimators=80; to
[CV] END max_depth=4, min_samples_leaf=40, min_samples_split=100, n_estimators=80; to
[CV] END max_depth=4, min_samples_leaf=40, min_samples_split=100, n_estimators=80; to
[CV] END max_depth=4, min_samples_leaf=40, min_samples_split=100, n_estimators=80; to
[CV] END max_depth=4, min_samples_leaf=40, min_samples_split=100, n_estimators=80; to
[CV] END max_depth=4, min_samples_leaf=40, min_samples_split=100, n_estimators=100; t
[CV] END max_depth=4, min_samples_leaf=40, min_samples_split=100, n_estimators=100; t
[CV] END max_depth=4, min_samples_leaf=40, min_samples_split=100, n_estimators=100; t
[CV] END max_depth=4, min_samples_leaf=40, min_samples_split=100, n_estimators=100; t
[CV] END max_depth=4, min_samples_leaf=40, min_samples_split=100, n_estimators=100; t
[CV] END max_depth=4, min_samples_leaf=40, min_samples_split=150, n_estimators=50; to
[CV] END max_depth=4, min_samples_leaf=40, min_samples_split=150, n_estimators=50; to
[CV] END max_depth=4, min_samples_leaf=40, min_samples_split=150, n_estimators=50; to
[CV] END max_depth=4, min_samples_leaf=40, min_samples_split=150, n_estimators=50; to
[CV] END max_depth=4, min_samples_leaf=40, min_samples_split=150, n_estimators=50; to
[CV] END max_depth=4, min_samples_leaf=40, min_samples_split=150, n_estimators=80; to
[CV] END max_depth=4, min_samples_leaf=40, min_samples_split=150, n_estimators=80; to
[CV] END max_depth=4, min_samples_leaf=40, min_samples_split=150, n_estimators=80; to
[CV] END max_depth=4, min_samples_leaf=40, min_samples_split=150, n_estimators=80; to
[CV] END max_depth=4, min_samples_leaf=40, min_samples_split=150, n_estimators=80; to
[CV] END max_depth=4, min_samples_leaf=40, min_samples_split=150, n_estimators=100; t
[CV] END max_depth=4, min_samples_leaf=40, min_samples_split=150, n_estimators=100; t
[CV] END max_depth=4, min_samples_leaf=40, min_samples_split=150, n_estimators=100; t
[CV] END max_depth=4, min_samples_leaf=40, min_samples_split=150, n_estimators=100; t
[CV] END max_depth=4, min_samples_leaf=40, min_samples_split=150, n_estimators=100; t
[CV] END max_depth=4, min_samples_leaf=50, min_samples_split=50, n_estimators=50; tot
[CV] END max_depth=4, min_samples_leaf=50, min_samples_split=50, n_estimators=50; tot
[CV] END max_depth=4, min_samples_leaf=50, min_samples_split=50, n_estimators=50; tot
[CV] END max_depth=4, min_samples_leaf=50, min_samples_split=50, n_estimators=50; tot
[CV] END max_depth=4, min_samples_leaf=50, min_samples_split=50, n_estimators=50; tot
[CV] END max_depth=4, min_samples_leaf=50, min_samples_split=50, n_estimators=80; tot
[CV] END max_depth=4, min_samples_leaf=50, min_samples_split=50, n_estimators=80; tot
[CV] END max_depth=4, min_samples_leaf=50, min_samples_split=50, n_estimators=80; tot
[CV] END max_depth=4, min_samples_leaf=50, min_samples_split=50, n_estimators=80; tot
[CV] END max_depth=4, min_samples_leaf=50, min_samples_split=50, n_estimators=80; tot
[CV] END max_depth=4, min_samples_leaf=50, min_samples_split=50, n_estimators=100; to
```

```
gb_grid.best_estimator_
```

```
GradientBoostingRegressor(max_depth=8, min_samples_leaf=40,
                          min_samples_split=50)
```

```
gb_optimal_model = gb_grid.best_estimator_
```

```
gb_grid.best_params_
```

```
{'max_depth': 8,
 'min_samples_leaf': 40,
 'min_samples_split': 50,
 'n_estimators': 100}
```

```
# Making predictions on train and test data

y_pred_train_g_g = gb_optimal_model.predict(X_train)
y_pred_g_g= gb_optimal_model.predict(X_test)
```

```
from sklearn.metrics import mean_squared_error
print("Model Score:",gb_optimal_model.score(X_train,y_train))
MSE_gbh= mean_squared_error(y_train, y_pred_train_g_g)
print("MSE :",MSE_gbh)

RMSE_gbh=np.sqrt(MSE_gbh)
print("RMSE :",RMSE_gbh)
```

```
MAE_gbh= mean_absolute_error(y_train, y_pred_train_g_g)
print("MAE :",MAE_gbh)
```

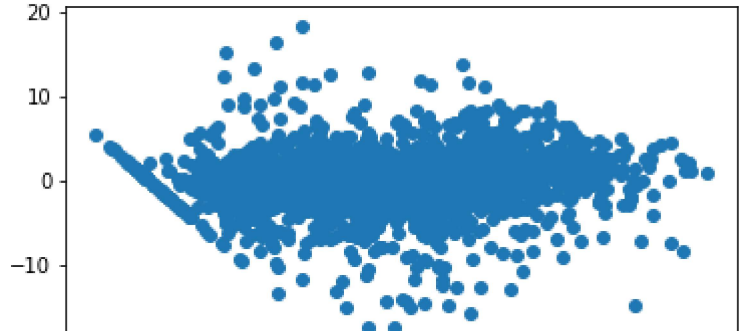
```
from sklearn.metrics import r2_score
r2_gbh= r2_score(y_train, y_pred_train_g_g)
print("R2 :",r2_gbh)
Adjusted_R2_gbh = (1-(1-r2_score(y_train, y_pred_train_g_g))*((X_test.shape[0]-1)/(X_test.sha
print("Adjusted R2 :",1-(1-r2_score(y_train, y_pred_train_g_g))*((X_test.shape[0]-1)/(X_test.
```

```
Model Score: 0.9515896672300013
MSE : 7.454740004128373
RMSE : 2.7303369762958516
MAE : 1.8489194833919358
R2 : 0.9515896672300013
Adjusted R2 : 0.9505274423746372
```

```
# storing the test set metrics value in a dataframe for later comparison
dict1={'Model':'Gradient Boosting gridsearchcv ',
       'MAE':round((MAE_gbh),3),
```

```
        'MSE':round((MSE_gbh),3),
        'RMSE':round((RMSE_gbh),3),
        'R2_score':round((r2_gbh),3),
        'Adjusted R2':round((Adjusted_R2_gbh ),2)
       }
training_df=training_df.append(dict1,ignore_index=True)


from sklearn.metrics import mean_squared_error
MSE_gbh= mean_squared_error(y_test, y_pred_g_g)
print("MSE :",MSE_gbh)

RMSE_gbh=np.sqrt(MSE_gbh)
print("RMSE :",RMSE_gbh)


MAE_gbh= mean_absolute_error(y_test, y_pred_g_g)
print("MAE :",MAE_gbh)


from sklearn.metrics import r2_score
r2_gbh= r2_score((y_test), (y_pred_g_g))
print("R2 :",r2_gbh)
Adjusted_R2_gbh = (1-(1-r2_score(y_test, y_pred_g_g))*((X_test.shape[0]-1)/(X_test.shape[0]-X
print("Adjusted R2 :",1-(1-r2_score((y_test), (y_pred_g_g)))*((X_test.shape[0]-1)/(X_test.sha


     MSE : 12.392760556291105
     RMSE : 3.520335290322657
     MAE : 2.4005915565405354
     R2 : 0.921548124829924
     Adjusted R2 : 0.9198267251413182


# storing the test set metrics value in a dataframe for later comparison
dict2={'Model':'Gradient Boosting gridsearchcv ',
        'MAE':round((MAE_gbh),3),
        'MSE':round((MSE_gbh),3),
        'RMSE':round((RMSE_gbh),3),
        'R2_score':round((r2_gbh),3),
        'Adjusted R2':round((Adjusted_R2_gbh ),2)
       }
test_df=test_df.append(dict2,ignore_index=True)


### Heteroscadacity
plt.scatter((y_pred_g_g),(y_test)-(y_pred_g_g))
```

```
<matplotlib.collections.PathCollection at 0x7f982d572b10>
```




```
gb_optimal_model.feature_importances_
```

```
array([3.12057438e-01, 1.52960270e-01, 6.20316780e-03, 6.03743312e-03,
       3.75324776e-02, 4.41384879e-02, 1.25380735e-03, 9.51937002e-04,
       9.98653930e-03, 1.96911560e-02, 2.89529919e-02, 2.45449359e-02,
       6.18872117e-03, 3.32550727e-03, 1.20634772e-02, 4.66509851e-04,
       3.73626049e-03, 2.06003352e-03, 6.36119042e-05, 1.39325973e-04,
       9.77140091e-05, 5.27337742e-04, 1.33847427e-03, 9.51593158e-03,
       3.11372494e-02, 1.65223193e-02, 1.21776066e-02, 1.44682226e-02,
       9.91879886e-03, 2.41023598e-03, 3.67911329e-03, 1.06796008e-03,
       1.62472784e-02, 2.18064007e-03, 1.57198472e-01, 7.75938658e-05,
       1.30043275e-03, 1.32742892e-04, 1.66051097e-03, 2.73793528e-03,
       1.57151366e-03, 6.25530621e-03, 7.62045639e-05, 6.35447256e-03,
       4.56464936e-03, 1.05534757e-03, 2.33718468e-02])
```

```
importances = gb_optimal_model.feature_importances_

importance_dict = {'Feature' : list(X_train.columns),
                   'Feature Importance' : importances}

importance_df = pd.DataFrame(importance_dict)


importance_df['Feature Importance'] = round(importance_df['Feature Importance'],2)


importance_df.head()
```

	Feature	Feature Importance
0	Temperature	0.31
1	Humidity	0.15
2	Wind_speed	0.01
3	Visibility	0.01
4	Solar_Radiation	0.04

```
importance_df.sort_values(by=['Feature Importance'],ascending=False)
```

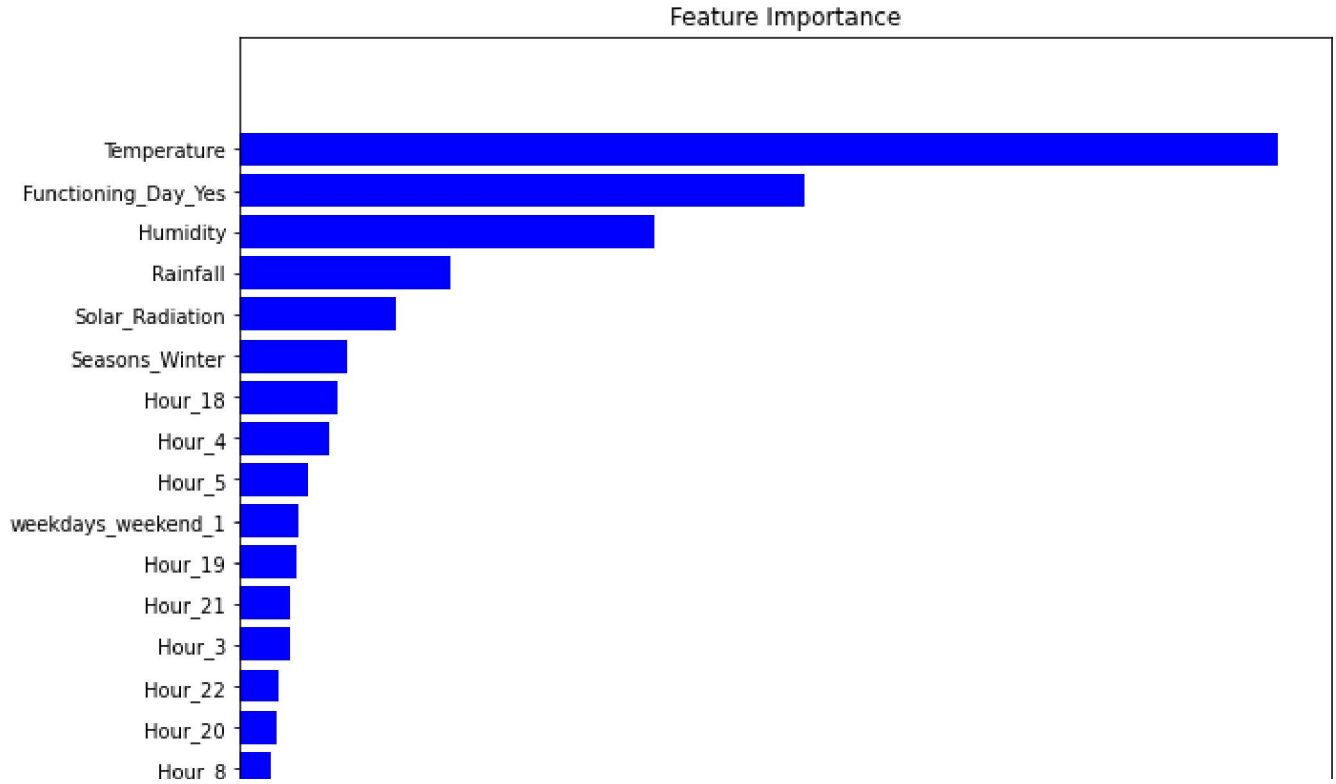
	Feature	Feature Importance
0	Temperature	0.31
34	Functioning_Day_Yes	0.16
1	Humidity	0.15
4	Solar_Radiation	0.04
5	Rainfall	0.04
10	Hour_4	0.03
24	Hour_18	0.03
32	Seasons_Winter	0.02
25	Hour_19	0.02
11	Hour_5	0.02
46	weekdays_weekend_1	0.02
9	Hour_3	0.02
8	Hour_2	0.01
43	month_10	0.01
41	month_8	0.01
2	Wind_speed	0.01
3	Visibility	0.01
12	Hour_6	0.01
28	Hour_22	0.01
27	Hour_21	0.01
26	Hour_20	0.01
23	Hour_17	0.01
14	Hour_8	0.01
22	Hour_16	0.00
35	month_2	0.00
45	month_12	0.00
44	month_11	0.00
13	Hour_7	0.00
42	month_9	0.00
40	month_7	0.00

39	month	6	0.00

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

```
GradientBoostingRegressor()
```

```
features = X_train.columns
importances = gb_model.feature_importances_
indices = np.argsort(importances)


#Plot the figure
plt.figure(figsize=(10,20))
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.show()
```



CONCLUSION

During the time of our analysis, we initially did EDA on all the features of our datset. We first analysed our dependent variable, 'Rented Bike Count' and also transformed it. Next we analysed categorical variable and dropped the variable who had majority of one class, we also analysed numerical variable, found out the correlation, distribution and their relationship with the dependent variable. We also removed some numerical features who had mostly 0 values and hot encoded the categorical variables.

Next we implemented 7 machine learning algorithms Linear Regression,lasso,ridge,elasticnet,decission tree, Random Forest and XGBoost. We did hyperparameter tuning to improve our model performance. The results of our evaluation are:

```
# displaying the results of evaluation metric values for all models
result=pd.concat([training_df,test_df],keys=['Training set','Test set'])
result
```

		Model	MAE	MSE	RMSE	R2_score	Adjusted R2
Training set	**0**	Linear regression	4.474	35.078	5.923	0.772	0.77
	1	Lasso regression	7.255	91.594	9.570	0.405	0.39
	2	Ridge regression	4.474	35.078	5.923	0.772	0.77
	3	Elastic net regression	5.792	57.574	7.588	0.626	0.62
	4	Dicision tree regression	5.170	49.284	7.020	0.680	0.67
	5	Random forest regression	0.806	1.607	1.268	0.990	0.99
	6	Gradient boosting regression	3.269	18.648	4.318	0.879	0.88
	7	Gradient Boosting gridsearchcv	1.849	7.455	2.730	0.952	0.95
Test set	**0**	Linear regression	4.410	33.275	5.768	0.789	0.78