# Supervised Machine Learning (Regression)

## Seoul Bike Sharing Demand Prediction


**By: Sanjay Verma**
**Poonam Shevkar**

# Introduction

Bike sharing systems are a means of renting bicycles where the process of obtaining membership, rental, and bike return is automated via a network of kiosk locations throughout a city. Using these Bike Sharing systems, people rent a bike from one location and return it to a different or same place on need basis. People can rent a bike through membership (mostly regular users) or on demand basis (mostly casual users). This process is controlled by a network of automated kiosk across the city.

- **<u>DATABASE:</u>**-

Seoul Bike Data

- From 2016 to 2017
- 8761 rows and 14 columns

# Problem Description-

The objective of this Project is to Predict bike rental count on daily based on the environmental and seasonal settings.

The crucial part is the prediction of bike count required at each hour for the stable supply of rental bikes.

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.

# Data Description-

The dataset contains  information about variables related to Weather,Seasons and Functional with the number of bikes rented per hour and date .

Attribute Information:

- Date : year-month-day
- Rented Bike count - Count of bikes rented at each hour
- Hour - Hour of the day
- Temperature-Temperature in Celsius
- Humidity - %
- Wind Speed - 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)

# DATA PIPELINE-

❖ Data processing-: At first phase verified the null values and changed the datetime containing column in dataset.

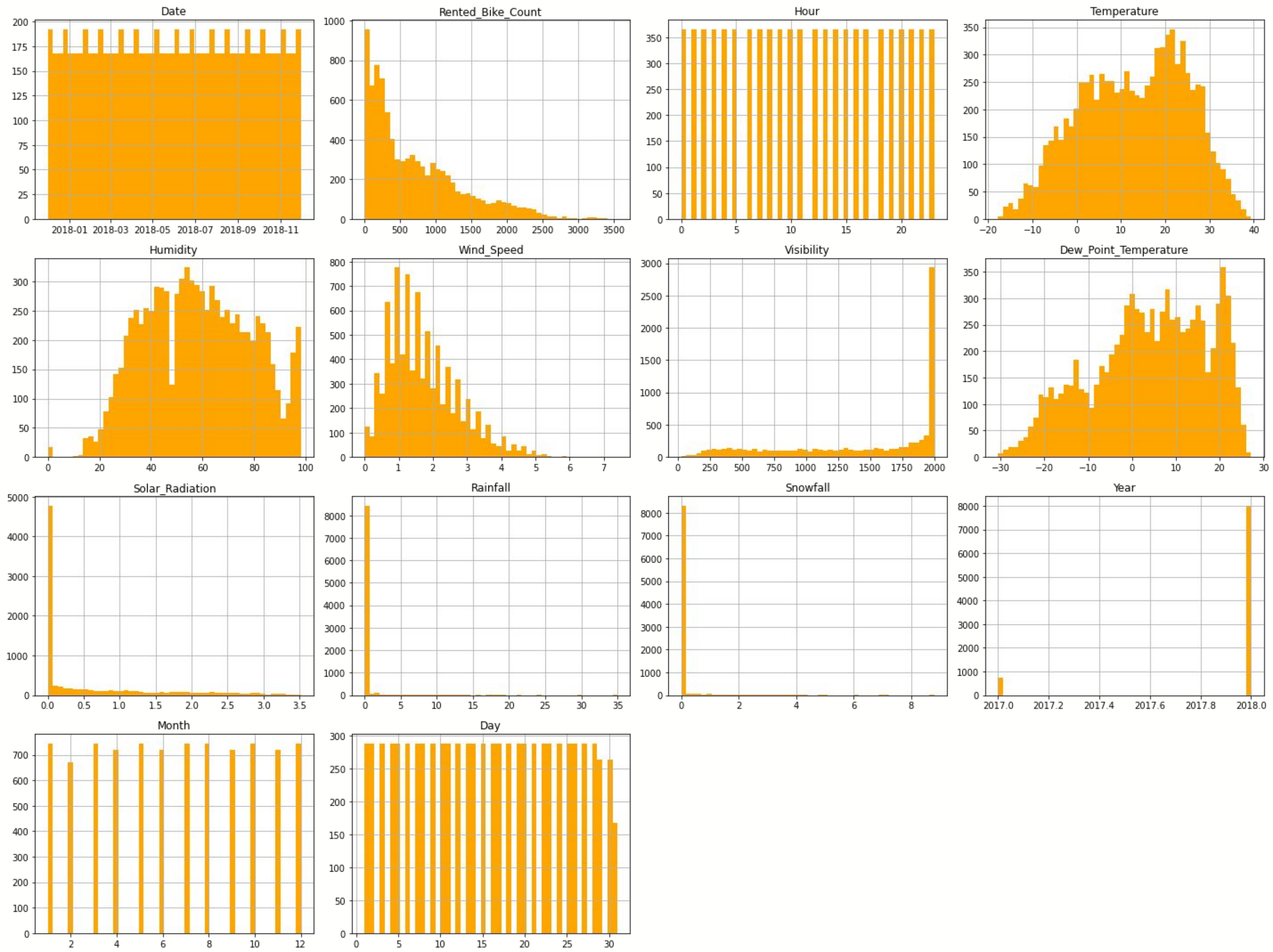
❖ Separate Features : Separate dependent and Independent variables. Divide them in different columns.

❖ EDA: Exploratory Data analysis was done on the features selected in the Phase ,feature distribution,Scatter plots/ Box plots and Outliers.

❖ Feature Scaling: Outlier treatment with median imputation,normalising the data and used of VIF to check the multi-collinearity among the variables.

❖ Data processing-2: Preparing the new dataframe with selected columns using dummy variables for one-hot encoding.

❖ Create a model: Finally in this part, creation of model is performed using trained and test set based on the splitting of the dataset.

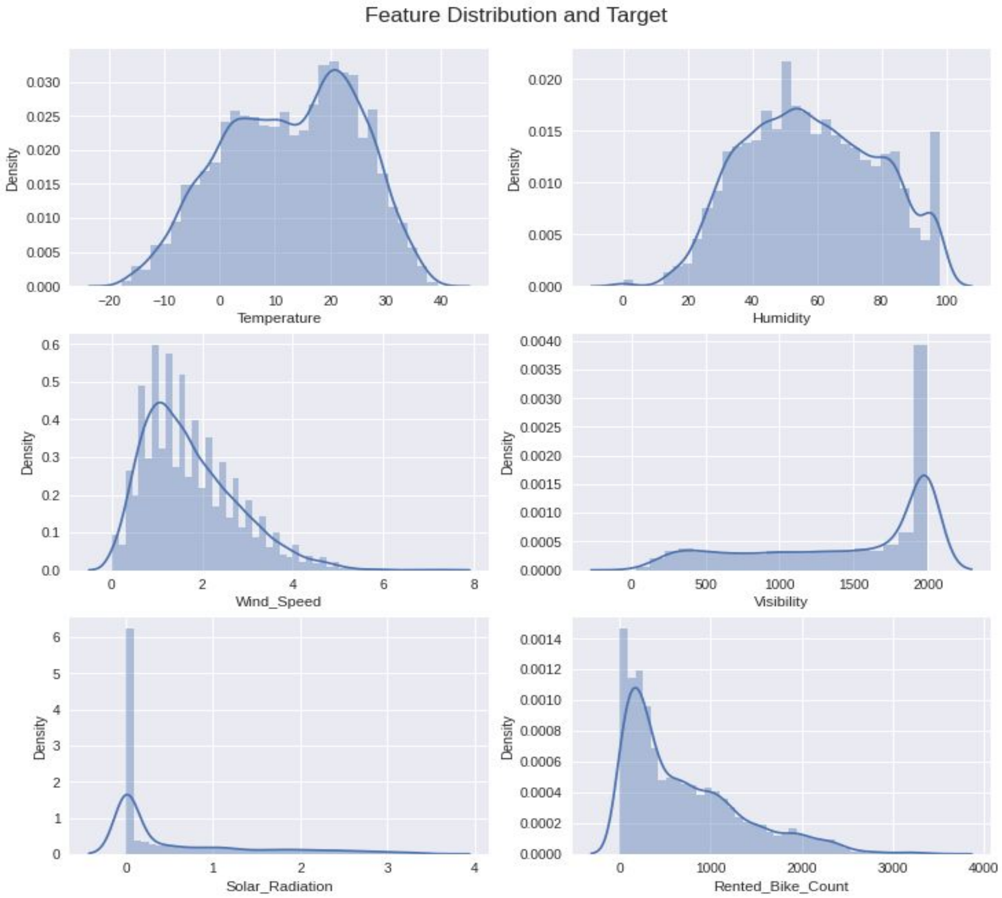



# EDA

Visualize the features of the dataset-

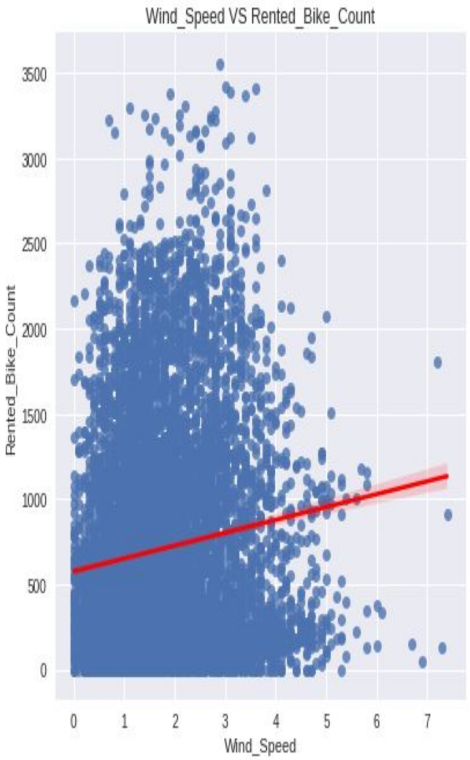
# EDA

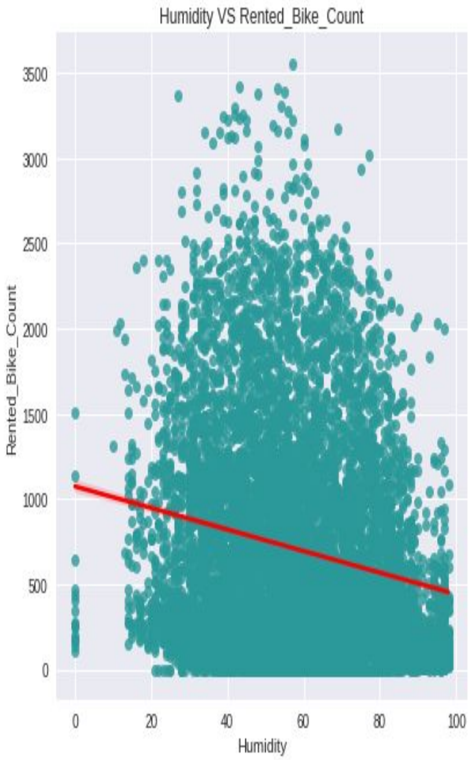
## Feature Distribution-

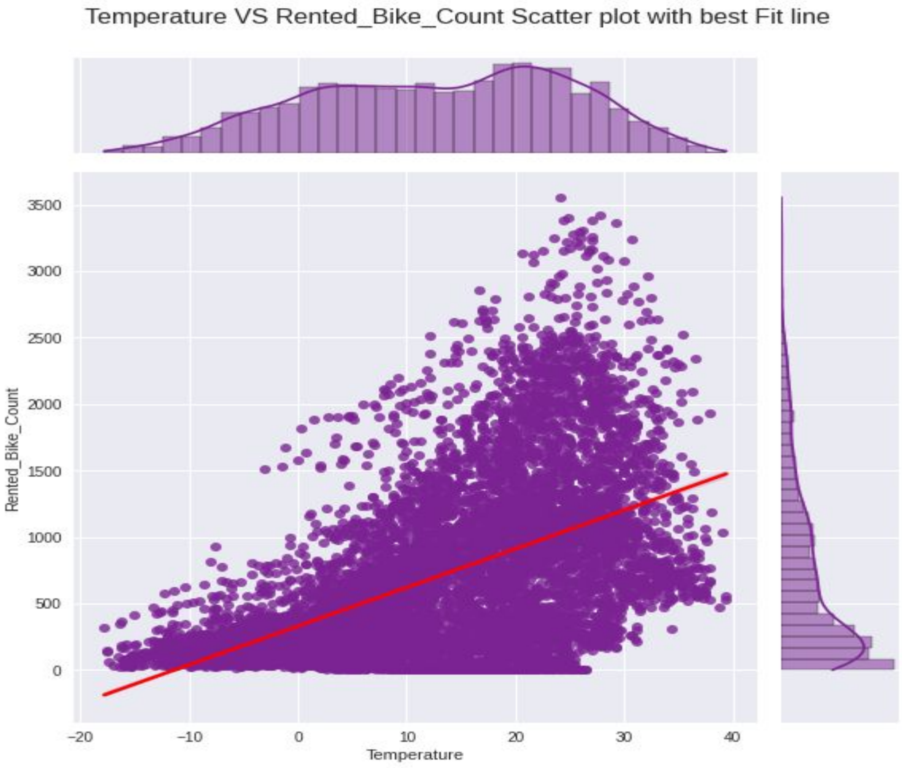



The above plot shows that the distribution of Wind speed, Solar Radiation, Rented bike count are positively skewed whereas visibility is negatively skewed.Also, Temperature and Humidity are nearly normally distributed.

# EDA-Best Fitting Line Between Feature And Target-

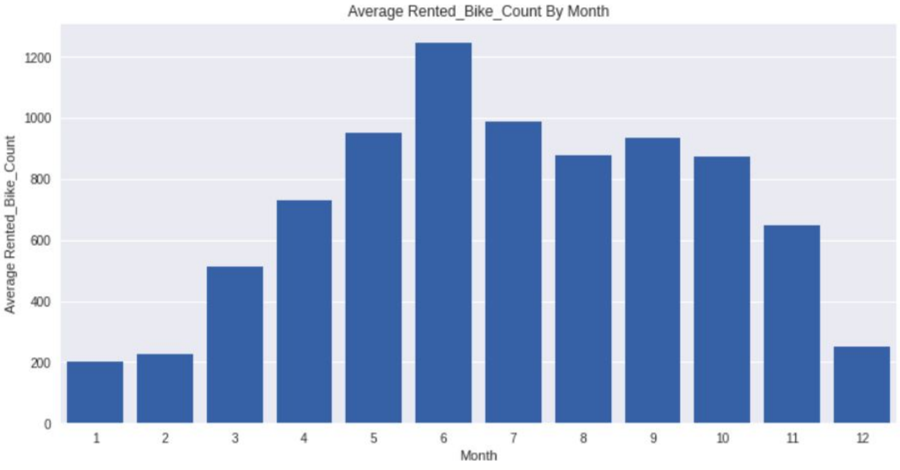


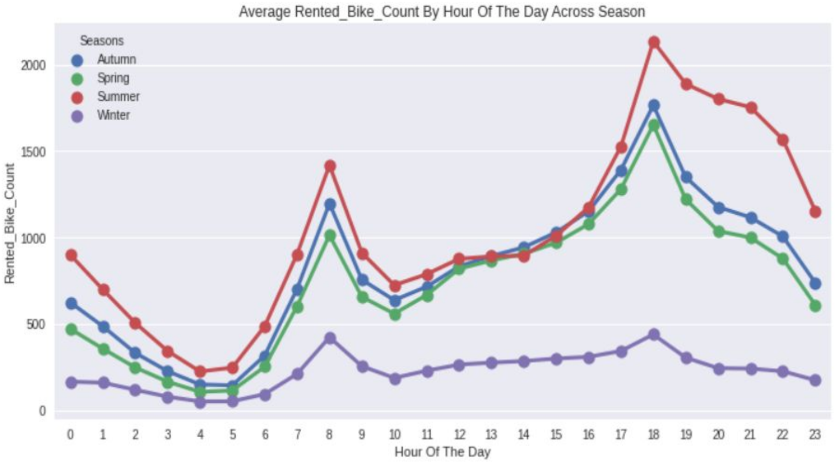



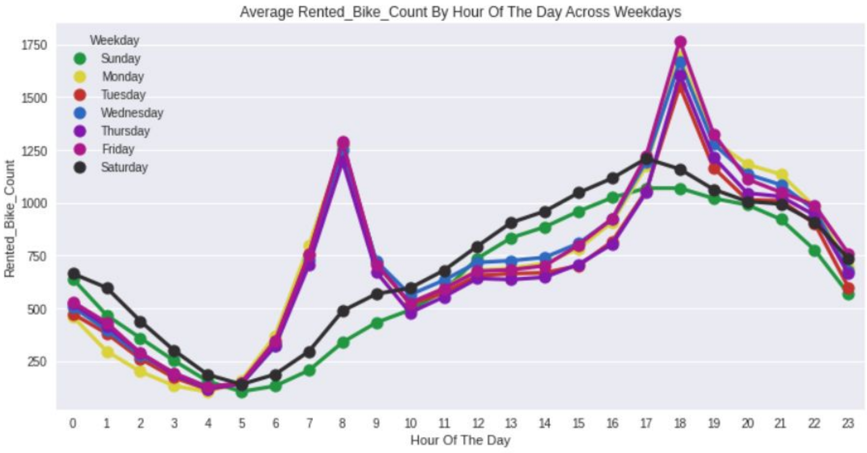




The best fit line describes- When there is Low Wind speed the rented bike count is maximum or populated.And, as humidity level decreases the count of rented bike increases.As the temperature feature is nearly normally distributed,the rented bike count is equally affected by the rise in temperature.

# Average Rented Bike Count



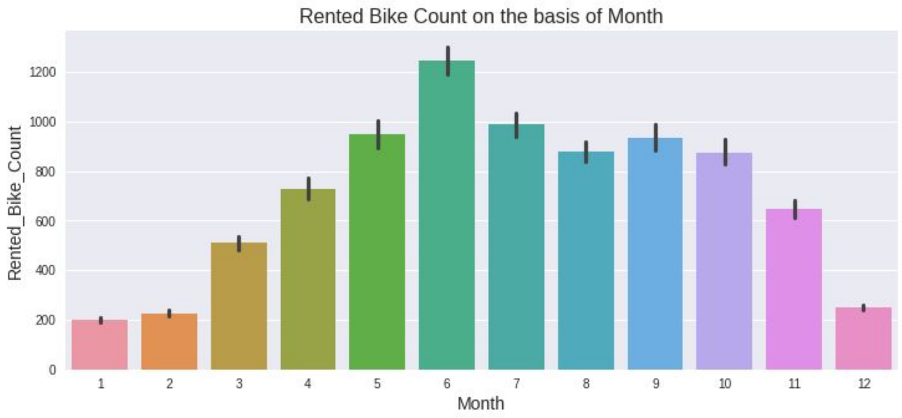



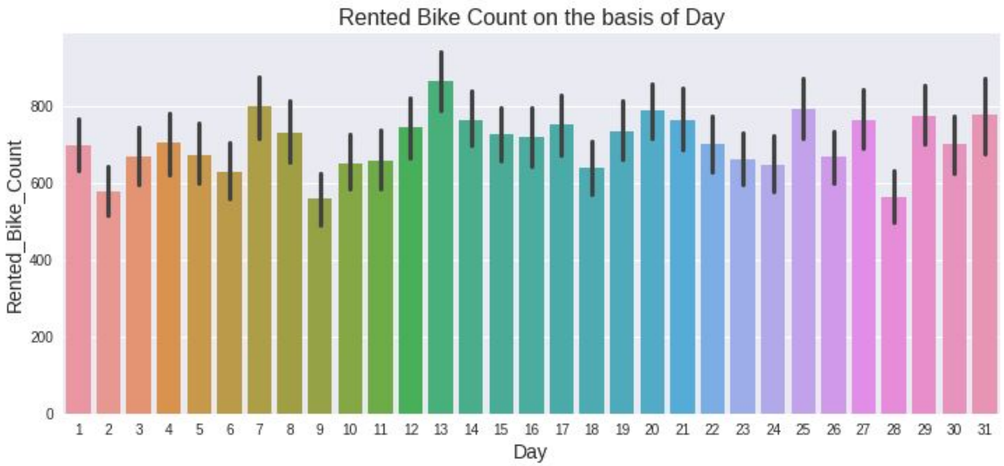



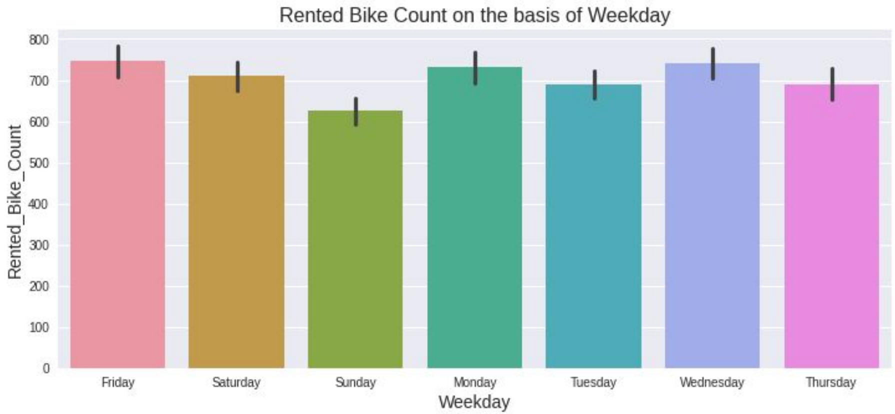

The average rented Bike Count is maximum in the month of June and minimum in January.We observe that,in Summer season the bike rental count is huge and low in Winter season as by hours of the day.Also,rented bike count is nearly same for all weekday except Saturday & Sunday.

# VISUALIZATION BY VALUE COUNTS-

In month, the count of rented bike is more in the middle i.e May, June & July whereas, low in December & January.And, Day wise the rented bike count depicts the similar behaviour everyday with slightly difference for weekoff(saturday/sunday).Also,Sunday has the lowest count of rented bike due to weekoff people avoid for rest.



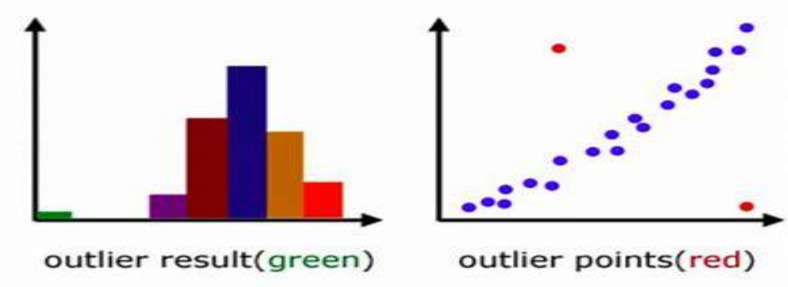
# Outlier Analysis

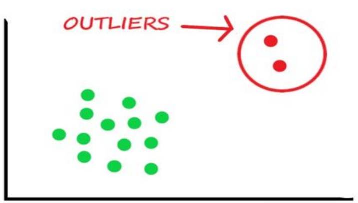
We look for outlier in the dataset by plotting Boxplots. There are outliers present in the data. Now we have removed these outliers. This is how we done,

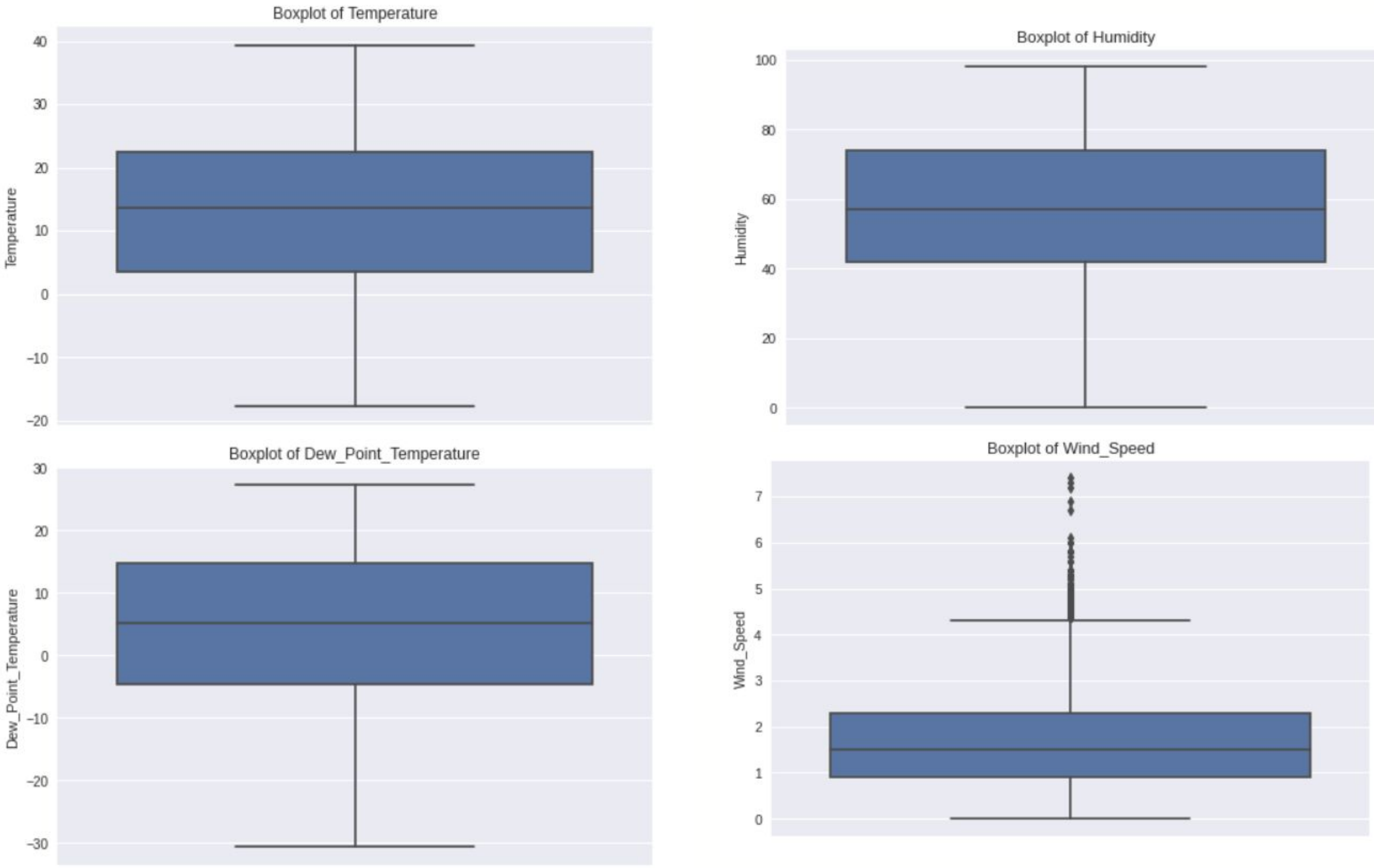
I. We replaced them with Nan values or we can say created missing values.

II. We tried three methods to impute the missing value: mean, median, KNN. Among this three methods we found that median imputation gives better treatment for outliers.

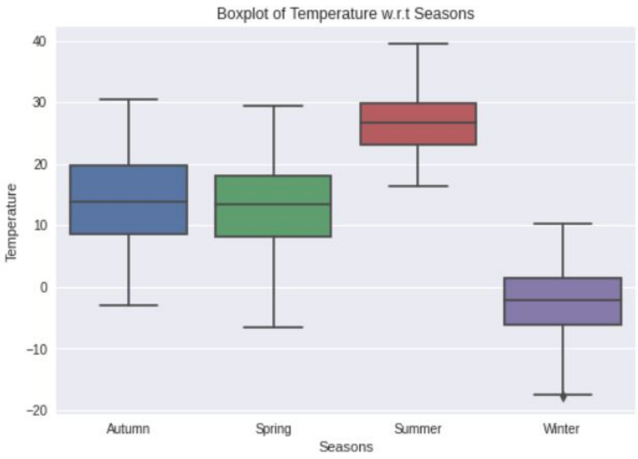
IV. We checked the performance of each method by checking Standard Deviation of that variable which has outliers before imputation and after imputation.

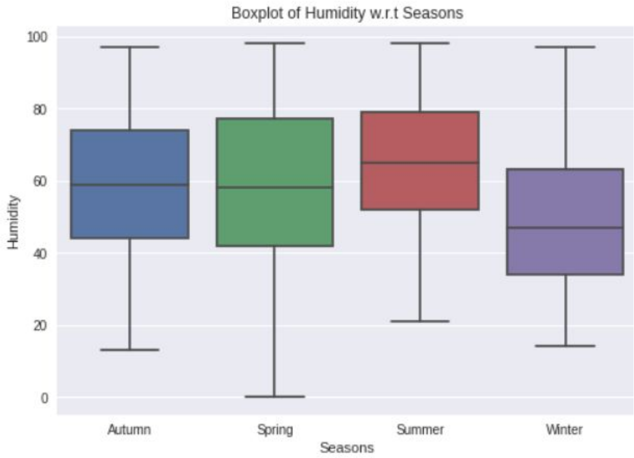


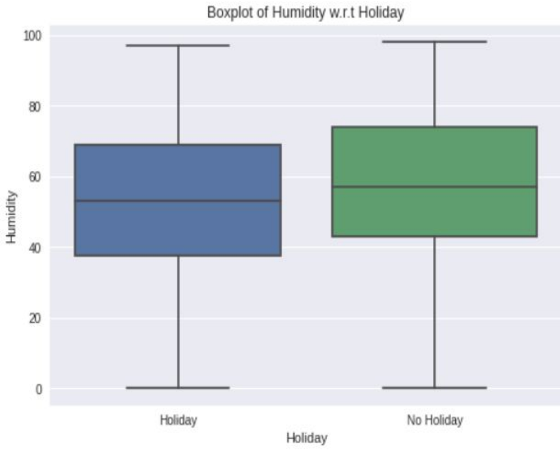



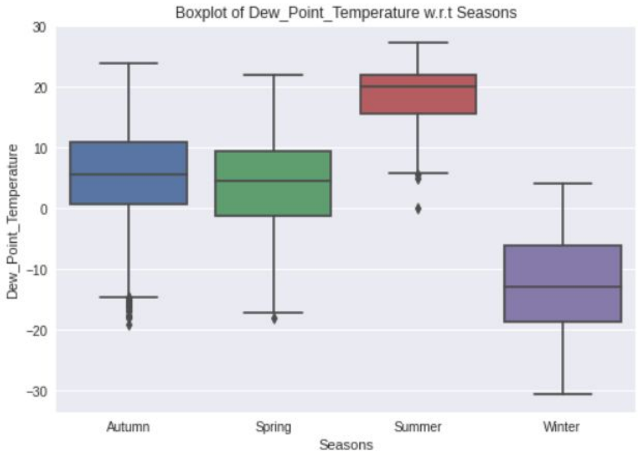

Boxplot of Temperature
Temperature
40
30
20
10
0
−10
−20
Boxplot of Humidity
Humidity
100
80
60
40
20
0
Boxplot of Dew_Point_Temperature
Dew_Point_Temperature
30
20
10
0
−10
−20
−30
Boxplot of Wind_Speed
Wind_Speed
7
6
5
4
3
2
1
0

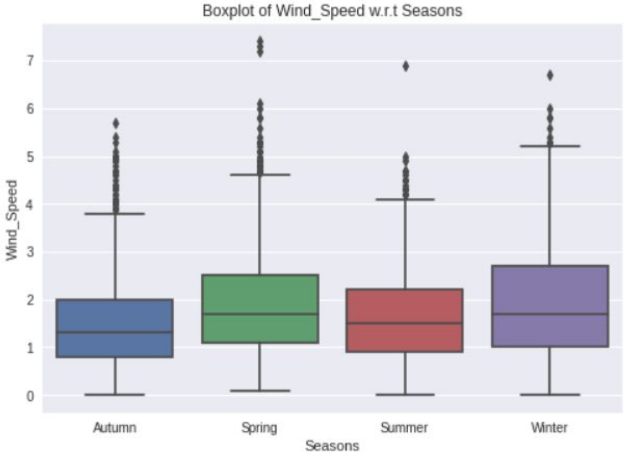
# Bivariate Boxplots: Boxplot for all Numerical Variables Vs all Categorical Variables-

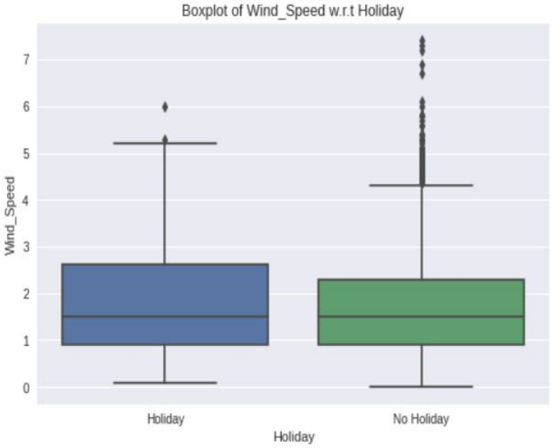


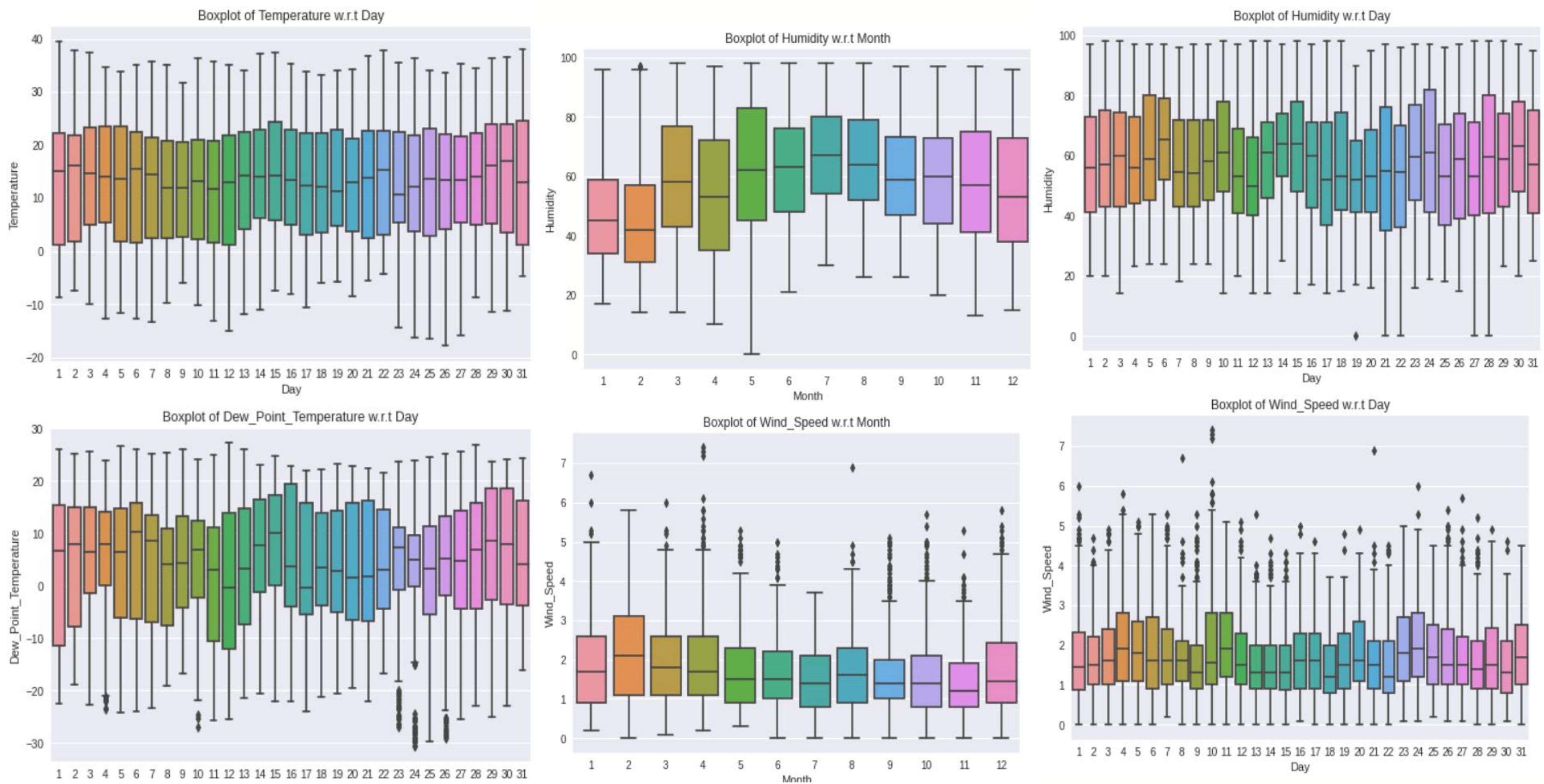






**As we can see from the above Boxplots-Univariate & Bivariate only 'Dew point temperature' and 'Windspeed' features in the dataset has the outliers.**

# Outlier Treatment-

```
# To evaluate Standard deviation for outlier treatment-
bike_df.std()
```

```
Date                     105 days 08:55:44.535820018
Rented_Bike_Count                         644.997468
Hour                                        6.922582
Temperature                                11.944825
Humidity                                   20.362413
Wind_Speed                                    1.0363
Visibility                                608.298712
Dew_Point_Temperature                      13.060369
Solar_Radiation                             0.868746
Rainfall                                    1.128193
Snowfall                                    0.436746
dtype: object
```

**Std Deviation before outlier treatment :** standard deviation for 'dew point temperature'= 13.060369 standard deviation for 'windspeed'= 1.0363

```
bike_df.std()
```

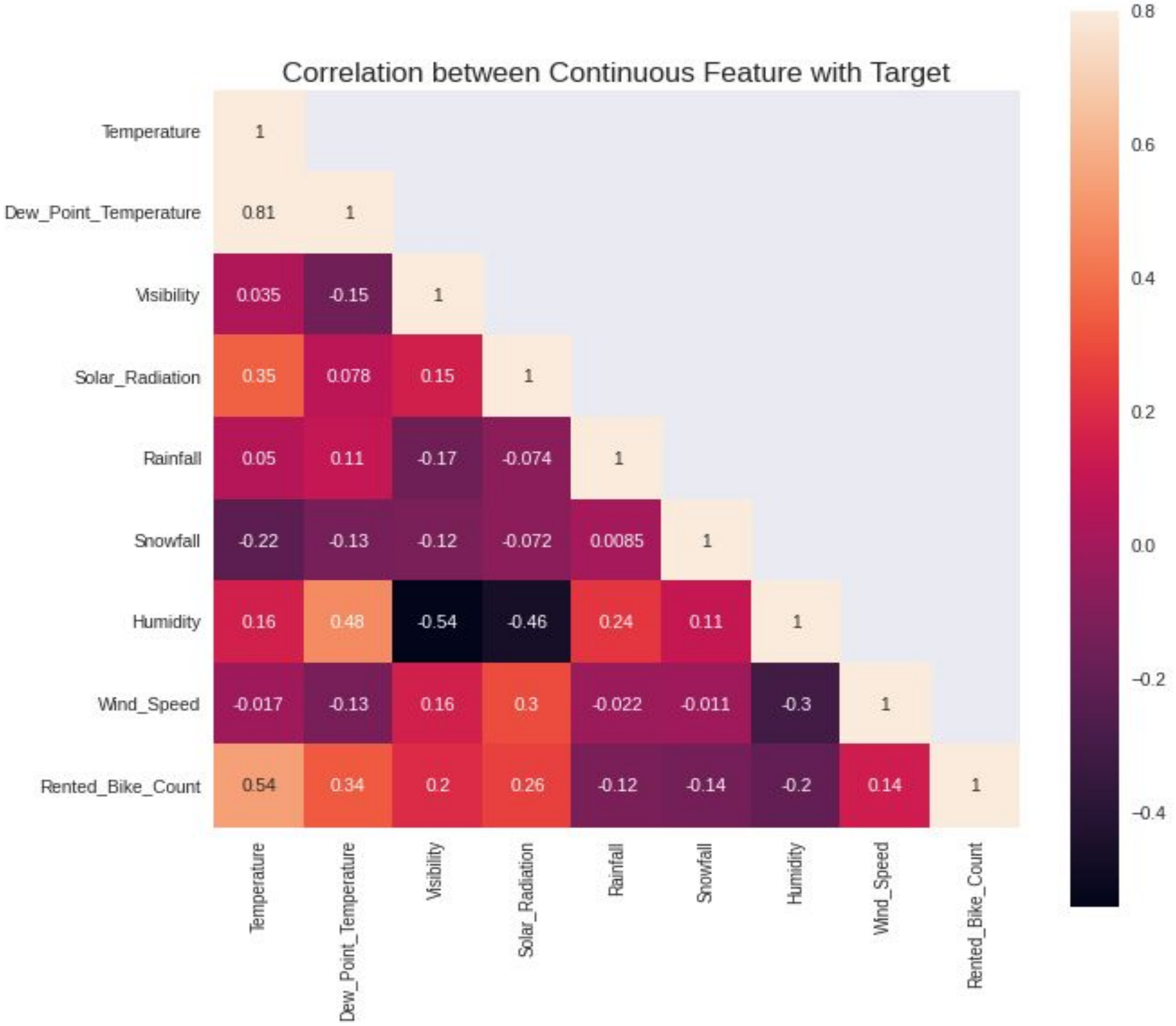
```
Date                     105 days 08:55:44.535820018
Rented_Bike_Count                         644.997468
Hour                                        6.922582
Temperature                                11.944825
Humidity                                   20.362413
Wind_Speed                                  0.947656
Visibility                                608.298712
Dew_Point_Temperature                      13.060369
Solar_Radiation                             0.868746
Rainfall                                    1.128193
Snowfall                                    0.436746
Wind speed                                   0.93917
Dew point temp                             13.060369
dtype: object
```

**Std Deviation after outlier treatment :** standard deviation for 'Dew point temperature(°C)'= 13.060369 standard deviation for 'windspeed'= 0.93917

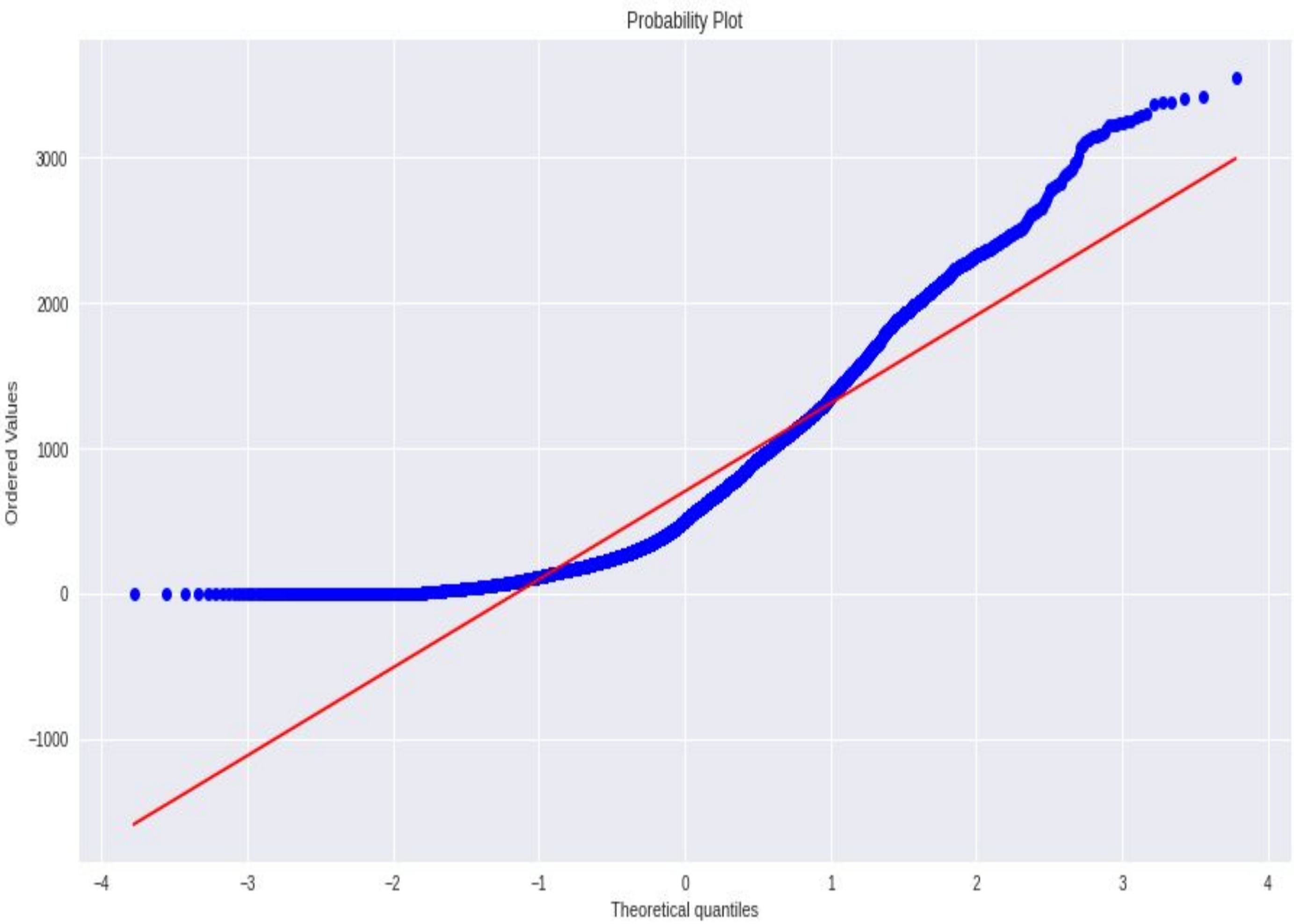
# Correlation matrix

'Temperature' and 'Dew point temperature' are very highly correlated with each other.

From correlation plot, we can observed that some features are positively correlated or some are negatively correlated to each other. The temp. and dew point temp. are highly positively correlated to each other, it means that both are carrying same information. So, we are going to ignore temp & dew point for further analysis.



# Normal Probability Plot-




**Rented_Bike_Count**

The above probability plot, the some target variable data points are deviates from normality.

# Multicollinearity Test-

**From above Dataframe, we see that there is Multicollinearity in our Data for- Dew point temperature(°C) and Humidity(%) has highest VIF value**

Now we will drop the highest multicollinearity column-

| | Variables | VIF |
|---|---|---|
| 0 | Rented_Bike_Count | 3.590435 |
| 1 | Hour | 4.161441 |
| 2 | Visibility | 4.361638 |
| 3 | Humidity | 11.548679 |
| 4 | Dew_Point_Temperature | 14.164964 |
| 5 | Solar_Radiation | 1.810299 |
| 6 | Rainfall | 1.094684 |
| 7 | Snowfall | 1.100190 |

# One Hot Encoding-

```
one_hot_var = ['Seasons','Holiday','Year']
```

```
#Creating dummies for categorical variables-
for i in one_hot_var:
    ''' Creating dummies for each variable in one_hot_var and merging dummies dataframe to our original dataframe '''
    temp= pd.get_dummies(bike_df[i], prefix = i)
    bike_df = bike_df.join(temp)
```

# Data Preprocessing-

```
#defining dependent and independent variables
dependent_variable = 'Rented_Bike_Count'
independent_variable = ['Month', 'Day', 'Hour', 'Visibility', 'Temperature', 'Solar_Radiation','Rainfall',
       'Snowfall', 'Wind_Speed', 'mean_Humidity', 'Seasons_Autumn', 'Seasons_Spring', 'Seasons_Summer' , 'Seasons_Winter',
       'Holiday_Holiday', 'Holiday_No Holiday', 'Year_2017', 'Year_2018']
```

```
#defining X and y varaibles
y = df[dependent_variable]
X = df[independent_variable]
```

# MODEL FITTING-

## MODEL FITTING FOR TRAIN AND TEST DATA:

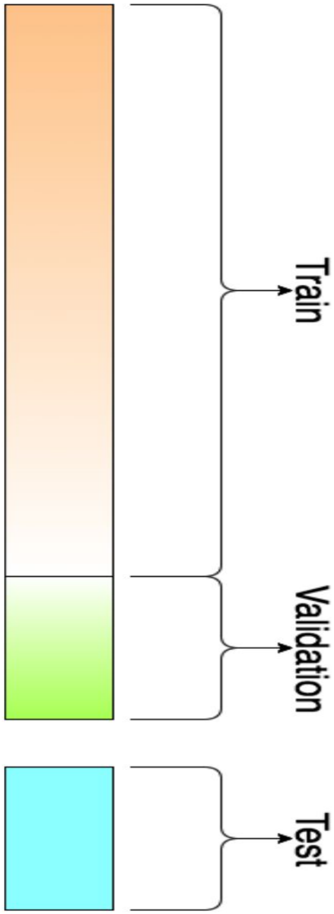
```
[ ]  from sklearn.model_selection import train_test_split
     from sklearn.preprocessing import StandardScaler, MinMaxScaler
     from sklearn.metrics import mean_squared_error
     from sklearn.metrics import r2_score
```

**Splitting the dataset into train and test in the ratio of 70:30**

```
[ ]  #splitting train and test data sets
     X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

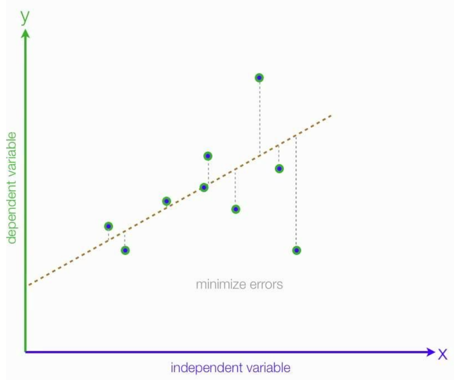
```
[ ]  #size of train and test datasets
     print(f'Size of X_train is: {X_train.shape}')
     print(f'Size of X_test is: {X_test.shape}')
     print(f'Size of y_train is: {y_train.shape}')
     print(f'Size of y_test is: {y_test.shape}')

     Size of X_train is: (6132, 18)
     Size of X_test is: (2628, 18)
     Size of y_train is: (6132,)
     Size of y_test is: (2628,)
```

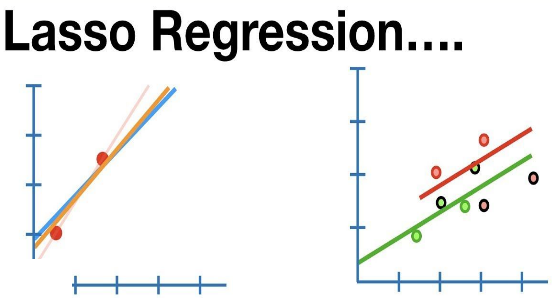
```
[ ]  #scaling the data
     scaler = StandardScaler()
     X_train = scaler.fit_transform(X_train)
     X_test = scaler.transform(X_test)
```

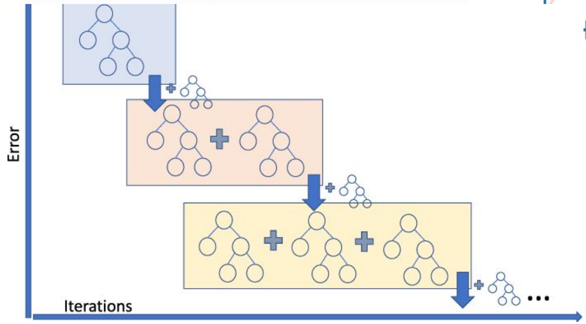


# MODELS USED-

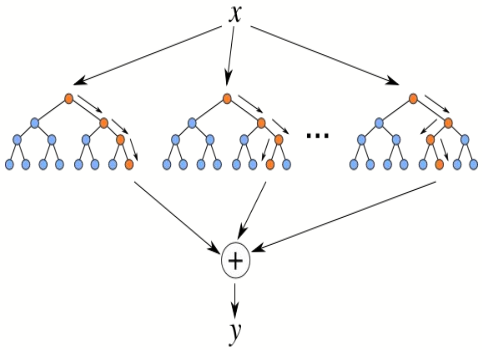
- **Linear Regression Model**
- **Lasso  Regression Model**
- **Decision Tree**
- **Random Forest Regressors**
- **Gradient Boosting Regressors**

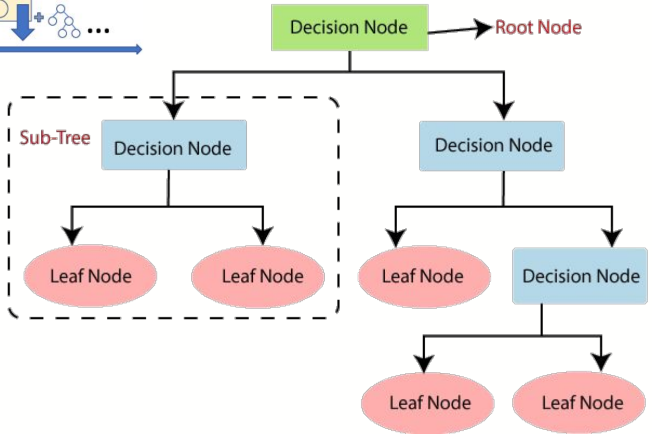



**In these above model fitting , we evaluate the model using Metrics as accuracy score,r-squared,MSE,RMSE.Also,Cross validation is used for all the models.**



# Let’s view the metrics of Train and test data set for all the models-

| | METRICS | MODELS | | | |
|---|---|---|---|---|---|
| | | LINEAR | LASSO | RANDOM FOREST | GRADIENT BOOSTING |
| TRAIN DATA SET | ACCURACY | 49% | 49% | 97% | 79% |
| | R2 | 0.4877 | 0.4877 | 0.4877 | 0.4877 |
| | MSE | 214442.27 | 214442.27 | 214442.27 | 214442.27 |
| | RMSE | 463.07 | 463.07 | 463.07 | 463.07 |
| TEST DATA SET | ACCURACY | 47% | 48% | 81% | 77% |
| | R2 | 0.4739 | 0.4755 | 0.8078 | 0.7660 |
| | MSE | 215581.40 | 214918.81 | 78739.49 | 95880.59 |
| | RMSE | 464.30 | 463.59 | 280.60 | 309.64 |

# Hyperparameter tuning & Cross validation (Gradient Boosting Regressors)-

```
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import make_scorer, r2_score

# Use a Gradient Boosting algorithm
alg = GradientBoostingRegressor()

# Trying these hyperparameter values
params = {
 'learning_rate': [0.1, 0.5, 1.0],
 'n_estimators' : [60, 120, 155]
 }

# Find the best hyperparameter combination to optimize the R2 metric
score = make_scorer(r2_score)
gridsearch = GridSearchCV(alg, params, scoring=score, cv=5, return_train_score=True)
gridsearch.fit(X_train, y_train)
print("Best parameter combination:", gridsearch.best_params_, "\n")
```

```
Best parameter combination: {'learning_rate': 0.5, 'n_estimators': 155}
```

**Without Hyperparameter tunning-**

1. Accuracy of the model of train data set is **79%**
2. Accuracy of the model of test data set is **77%**

**With Hyperparameter tunning-**

1. Accuracy of the model of train data set is **90%**
2. Accuracy of the model of test data set is **80%**

# Conclusions-

- Overall we observe that,Linear Regression model and Lasso Regression model are worst fitted model as there accuracy is less than 50% whereas, Random Forest Regressor and Gradient Boosting Regressor are the best fitted model for the train and test data set.
- Random Forest Regressor has the accuracy rate of train data set 98% and test data set 81%. Also,MSE is 463.08 for train data set and 280.61 for test data set. After,hyperparameter tuning the accuracy rate gives the similar result for train and test data set.
- Gradient Boosting Regressor has the accuracy rate of train data set 79% and test data set 77%. Also,MSE is 463.08 for train data set and 309.65 for test data set. With hyperparameter tuning the accuracy of the model increased and RMSE decreases which implies that the model fitted is the best model for higher accuracy rate of regression models with the predictions.

## With Hyperparameter tuning-

1. Accuracy of the model of train data set is 90%
2. Accuracy of the model of test data set is 80%
3. RMSE of the model of train data set is 463.08
4. RMSE of the model of test data set is 285.46

- **Among, all the above models we conclude that Gradient Boosting Regressor(With hyperparameter tuning) is the best fitted model for Seoul Bike Rental Prediction data set.**