# Capstone Project - 2

# Supervised – ML – Bike Sharing Demand Prediction


Team Members
Sri harish A
Sethupathy M

# Contents

# Addressing the problem

The goal of this study is to test a machine learning strategy for predicting **bike sharing demand** in Seoul given the hour, day, and weather information. This project includes the following components: exploratory data analysis, feature engineering, selecting relevant features, cross algorithms, cross validation, tweaking the hyper parameters, feature importance analysis, and model performance analysis. The dataset shown here is from the years 2017 and 2018. Predictions of future usage might aid in improved service management. Another viewpoint is to put machine learning algorithms to the test to see how well they handle this problem.

# Addressing the problem

## How Bike sharing works?


- **Step 1 :** Register through bike share app or at any bike station.
- **Step 2** : Pick out your bike from any bike port.
- **Step 3** : Get on your bike and take off.
- **Step 4** : Park your bike in any port at any station.

# Reasons why bike sharing is beneficial:

- **Reduces traffic congestion**

- Overall, traffic congestion costs reduces.
- The average commuter spends 50 hours every year stuck in traffic.

- **Improving public health through exercise**

- The average person loses 13 lbs. their first year of commuting by bike.
- At least 30 minutes of exercise is recommended at least 5 days a week.

- **Potentially reducing greenhouse gas emissions and air pollution**

- A short, four-mile round trip by bicycle keeps about 15 pounds of pollutants out of the air.
- By 2032 traffic delays will more than double and CO2 emissions traced to congestion will reach 60 million tons.

# Features Summary

**Independent variables:**

- **Date** - year-month-day.
- **Hour** - Hour of the day.
- **Temperature** - Temperature in Celsius.
- **Humidity** -  Humidity in percentage.
- **Windspeed** - Windspeed in m/s.
- **Visibility** - Visibility in meters.
- **Dew point temperature** - The dew point is the temperature at which air is saturated with water vapor.(Celsius)
- **Solar radiation** - Solar radiation is the heat and light and other radiation given off by the Sun.(MJ/m2)
- **Rainfall** – Rainfall in mm.

# Features Summary (continued)

**Independent variables:**

- **Snowfall –** Snowfall in cm
- **Seasons -** Winter, Spring, Summer, Autumn
- **Holiday -** Holiday/No holiday
- **Functional Day -** NoFunc(Non Functional Hours), Fun(Functional hours)

**Dependent variable:**

- **Rented Bike count -** Count of bikes rented at each hour.

# Outliers

An outlier is an **extremely high or extremely low data point** relative to the nearest data point and the rest of the neighboring co-existing values in a data graph or dataset you're working with.

## Ways to detect outliers:

- Interquartile range
- Box plot
- Scatter plot
- Z – score
- In the given dataset we have used Box plot to detect outliers.

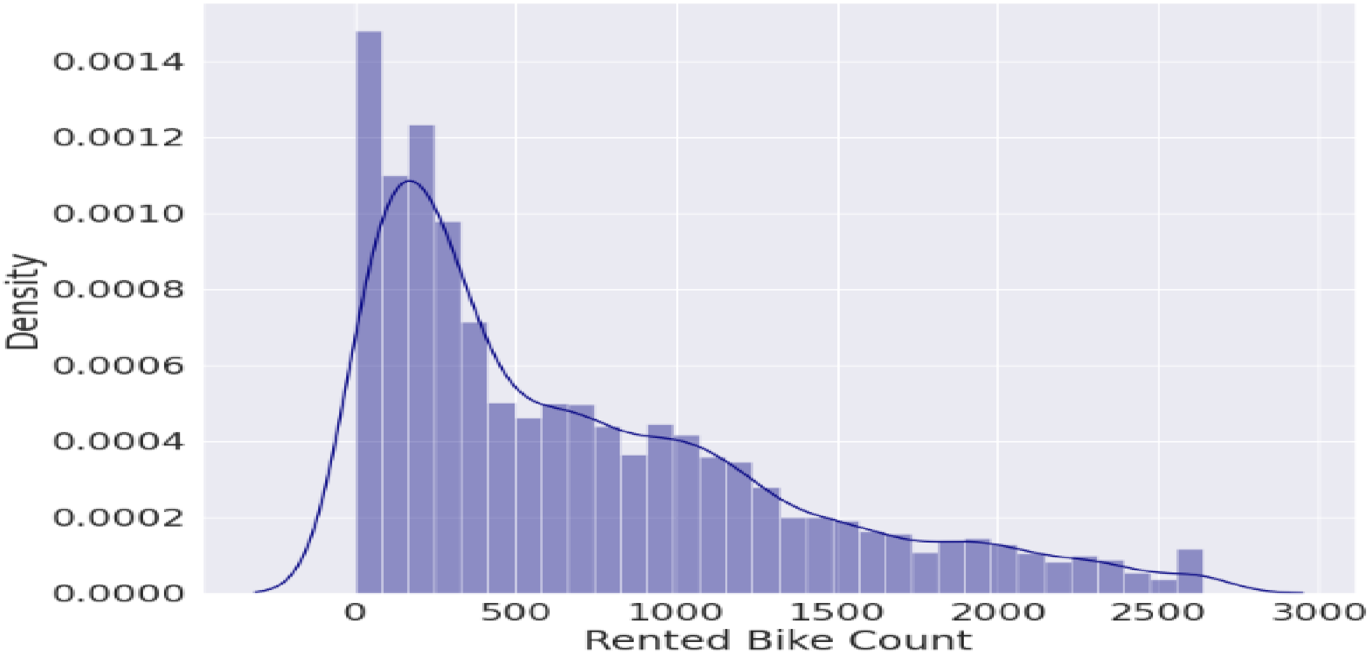
# Outliers (continued)

**Z–Score** – To handle outliers.

```
# Z Score based technique to remove outliers
lst = ['Rented Bike Count','Temperature(°C)', 'Humidity(%)',
       'Wind speed (m/s)', 'Visibility (10m)', 'Dew point temperature(°C)',
       'Solar Radiation (MJ/m2)']
for i in lst:
  lower_limit = df[i].mean() - 3*df[i].std()
  print(i+'Lower_limit:',round(lower_limit,2))
  upper_limit = df[i].mean() + 3*df[i].std()
  print(i+'Upper_limit:',round(upper_limit,2))
  df[i] = np.where(df[i]>upper_limit,upper_limit,np.where(df[i]<lower_limit,lower_limit,df[i]))
```

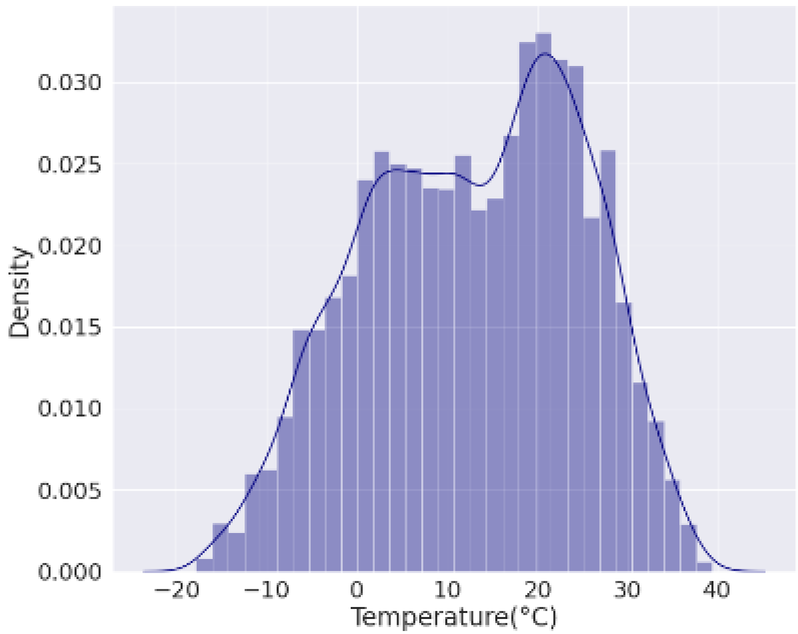
# Exploratory Data Analysis (continued)

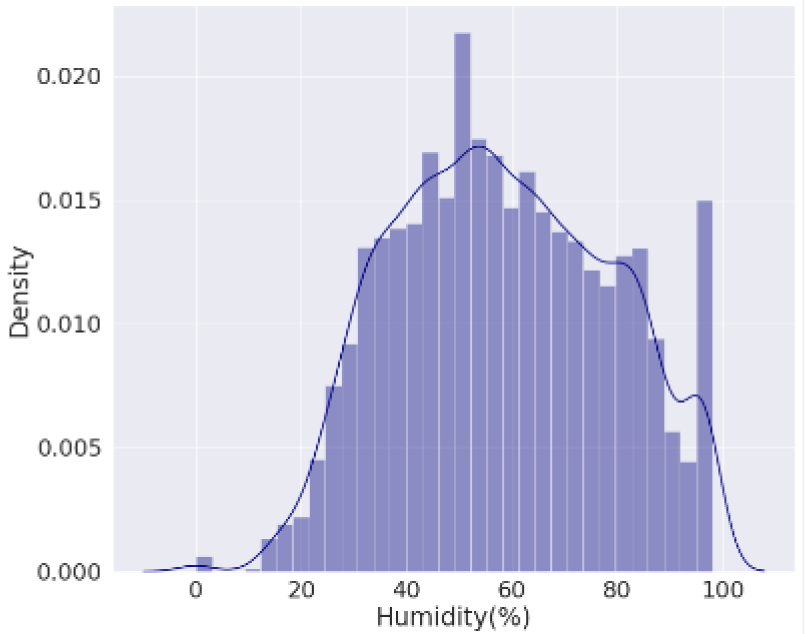
## Distribution plot for dependent variable.



# Exploratory Data Analysis

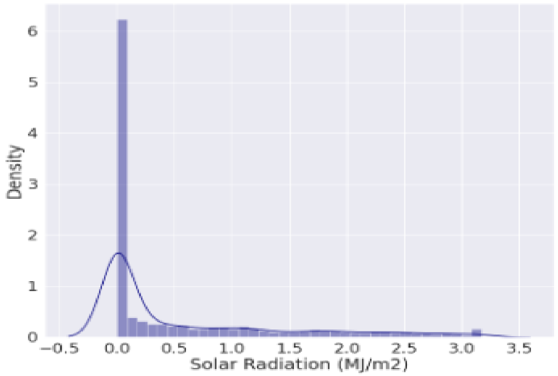
## Distribution plot for independent variables.

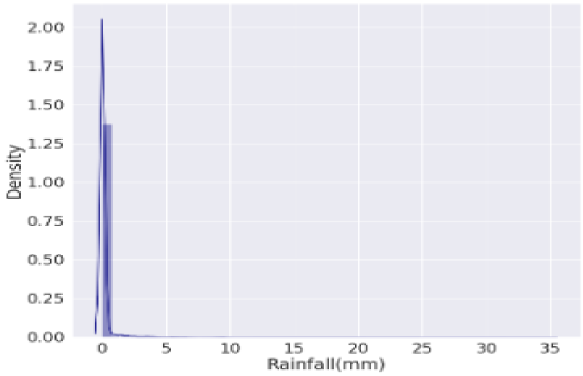


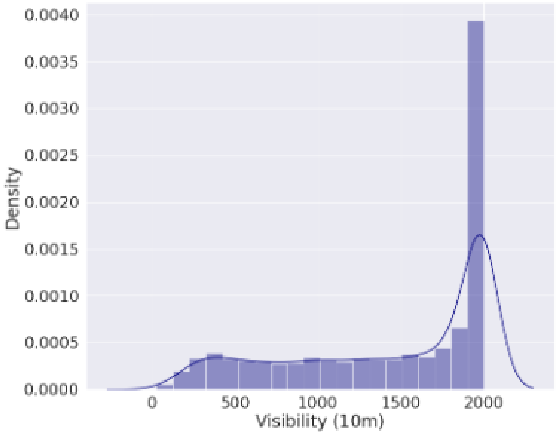



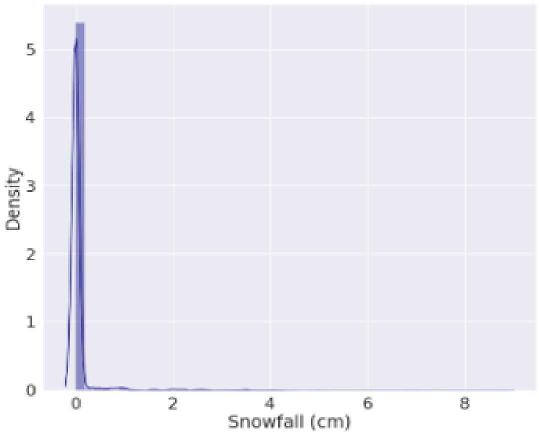
# Exploratory Data Analysis (continued)

## Distribution plot for skewed independent variable.



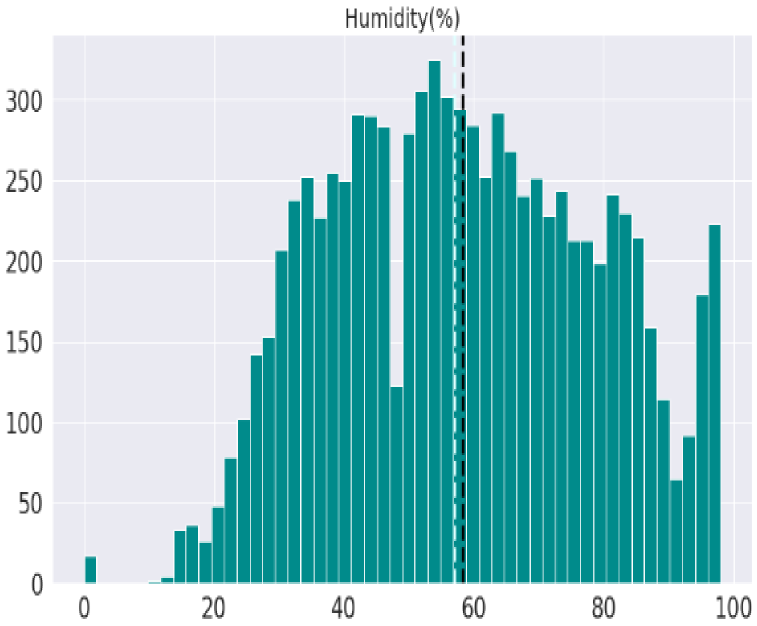



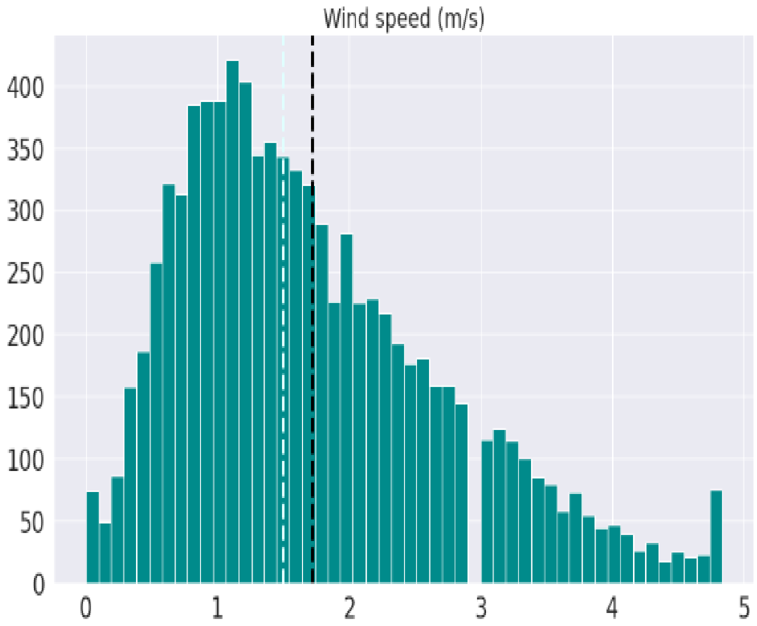



# Exploratory Data Analysis (continued)

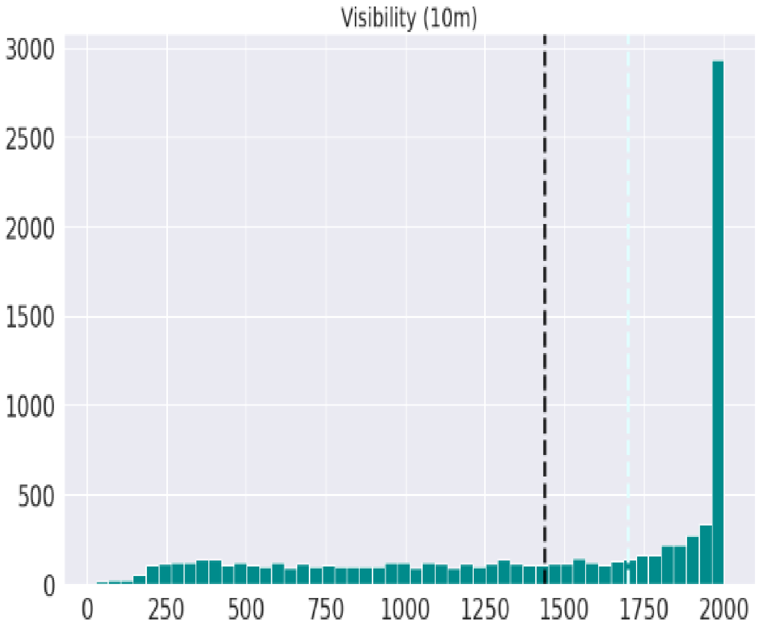
## Mean and Median plot for independent variables.

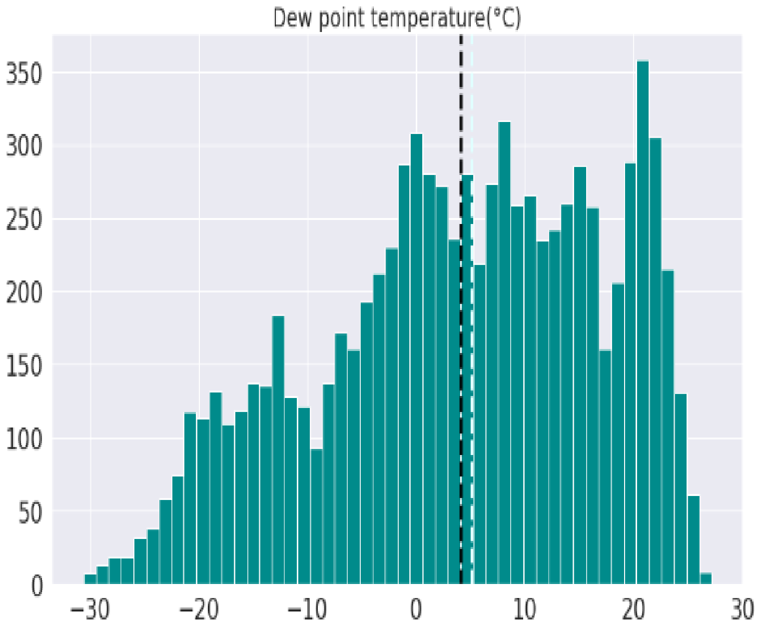


# Exploratory Data Analysis (continued)

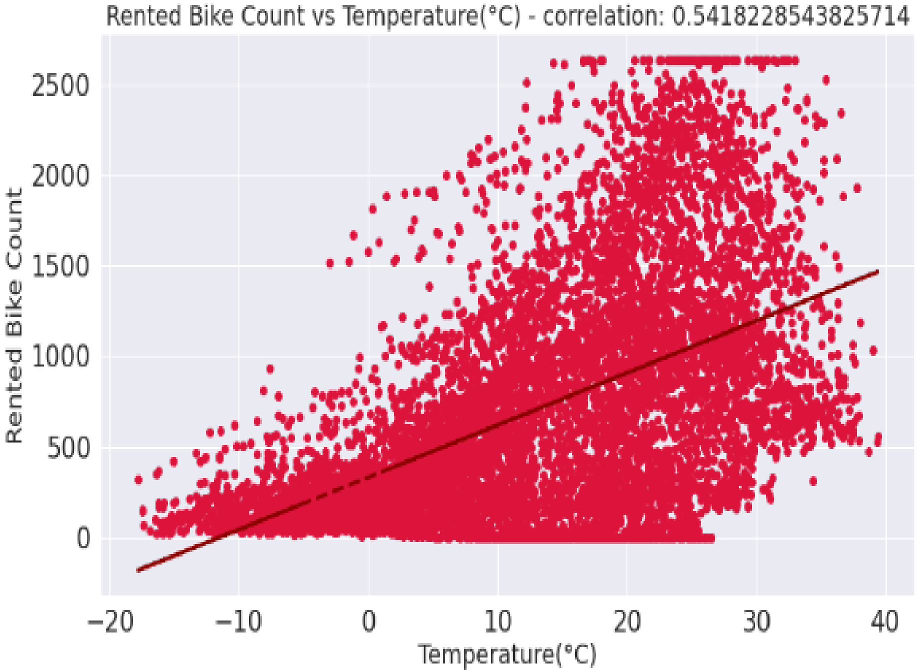
## Mean and Median plot for independent variables.

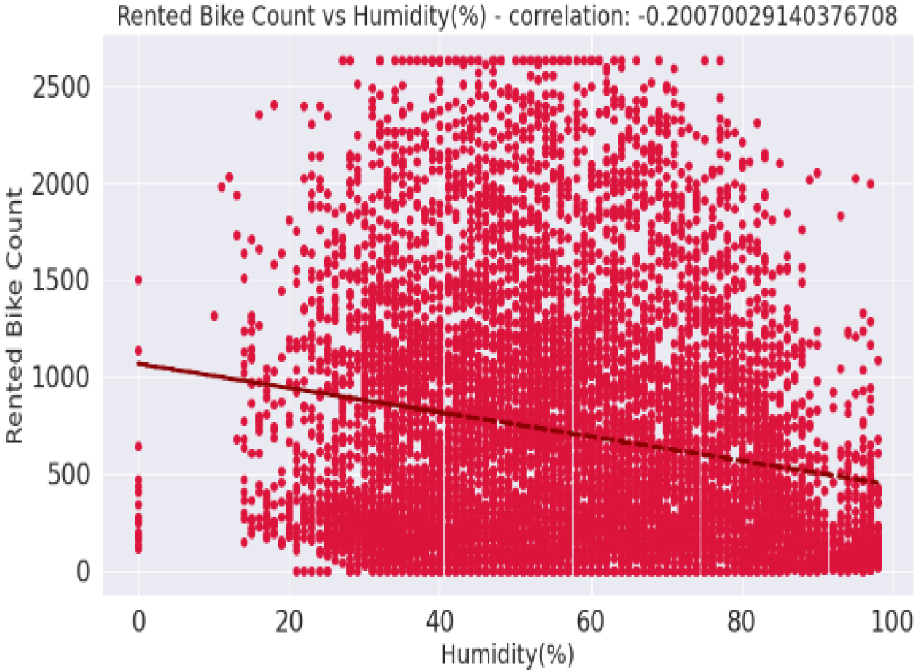


# Exploratory Data Analysis (continued)

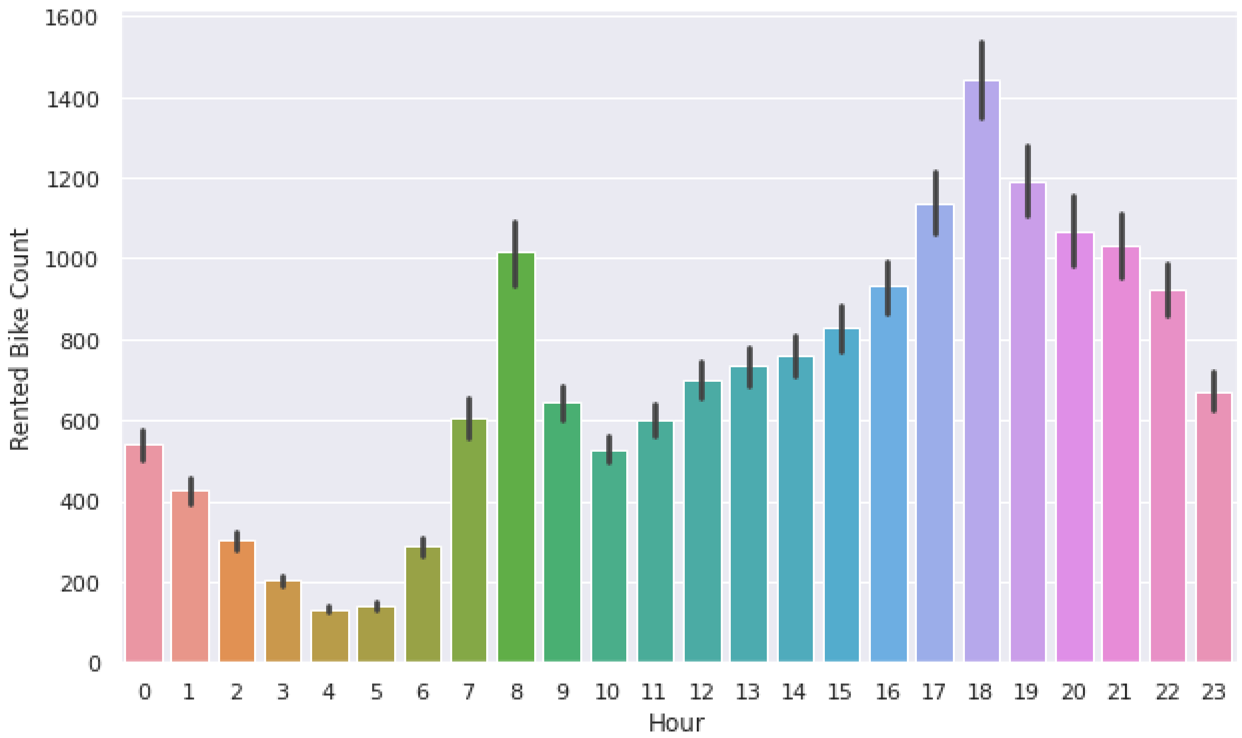
## Scatter and Correlation plot for independent variables



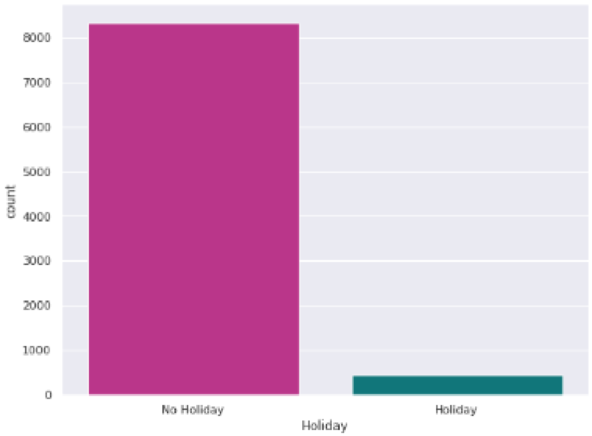
# Exploratory Data Analysis (continued)

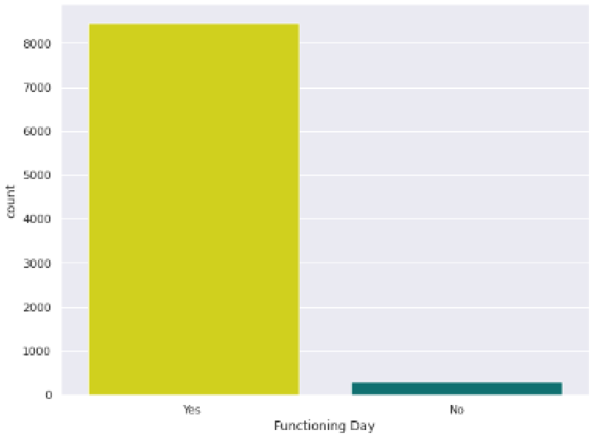
## Bar plot for Hourly bike count

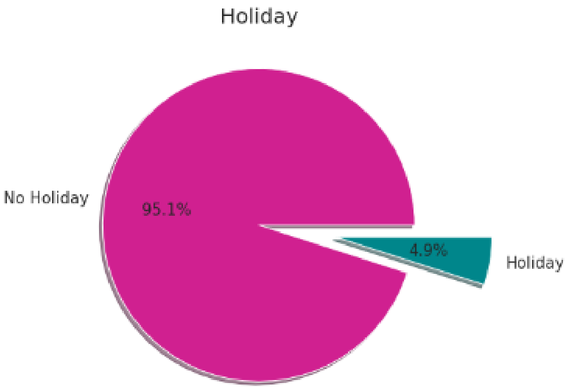


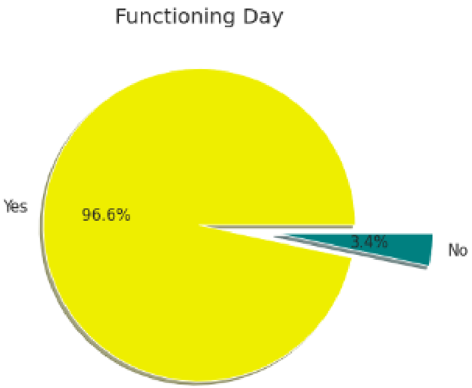
# Exploratory Data Analysis (continued)

## Independent categorical variables count plot and pie chart analysis



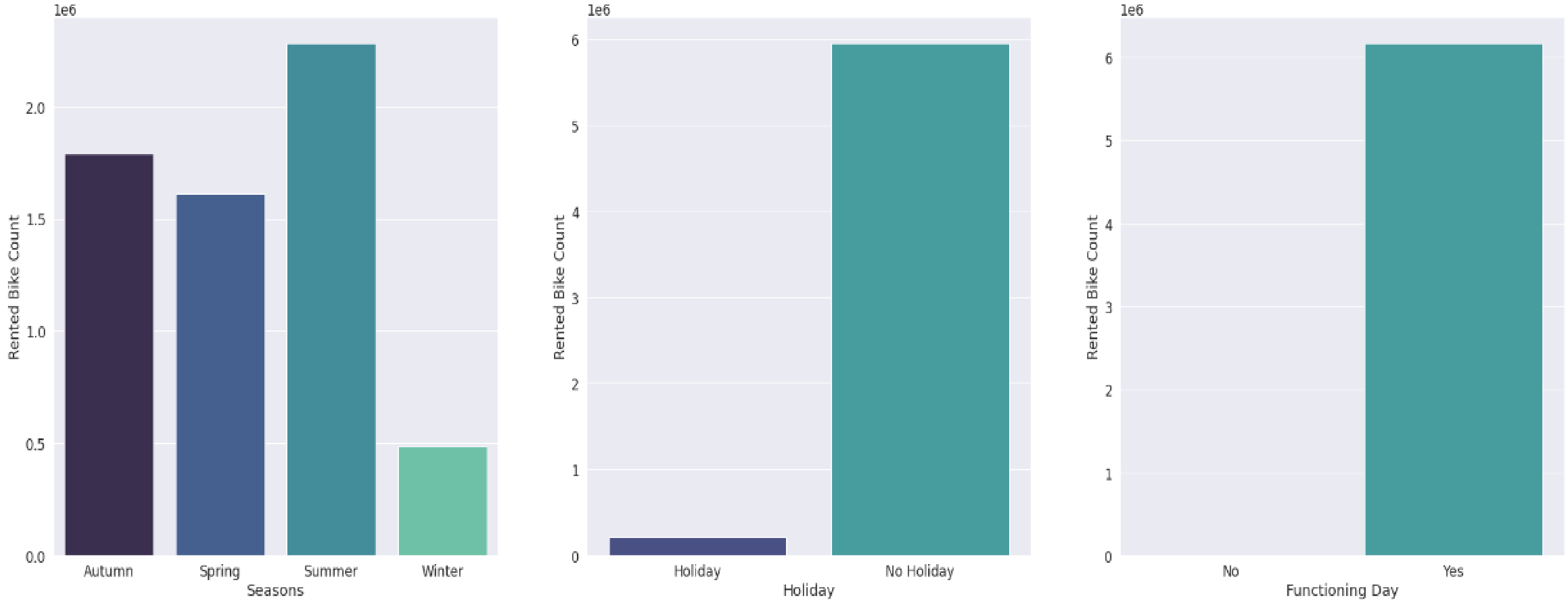



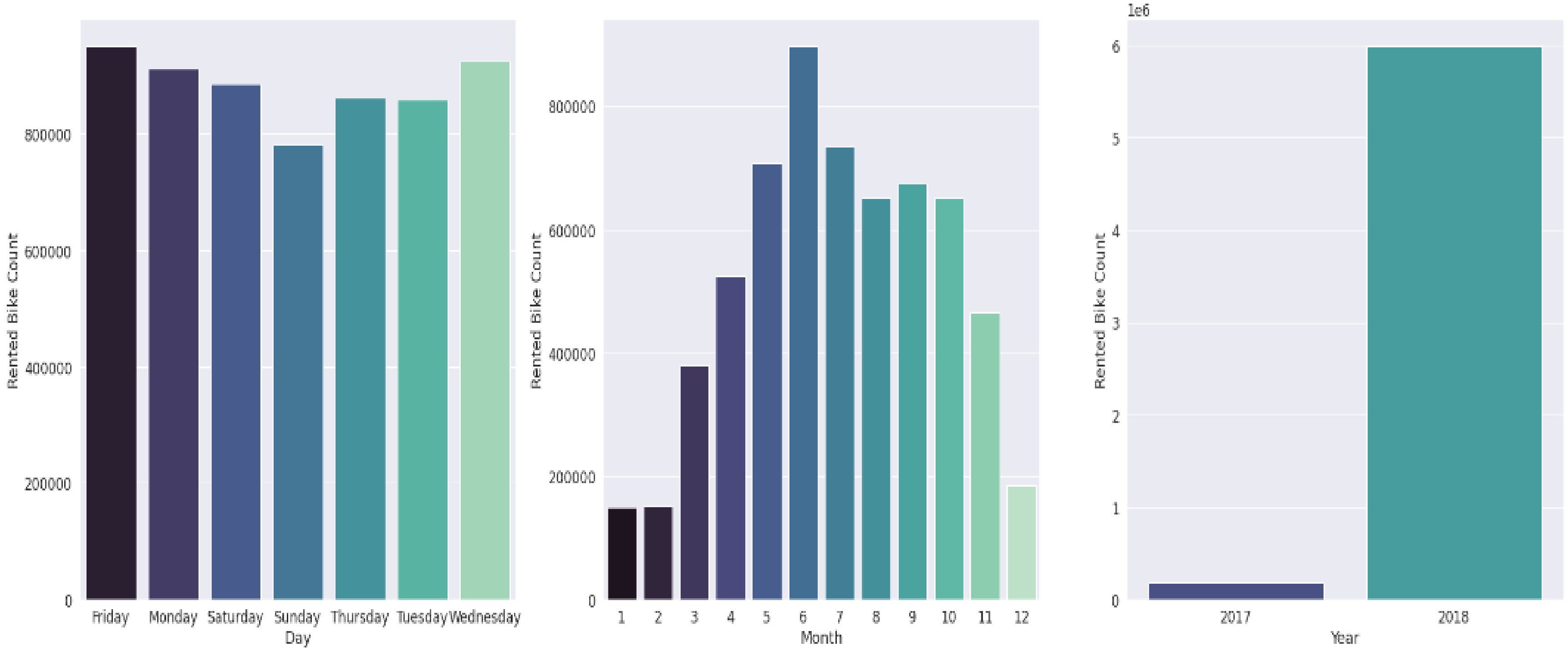
# Exploratory Data Analysis (continued)

## Sum plot for dependent variable.



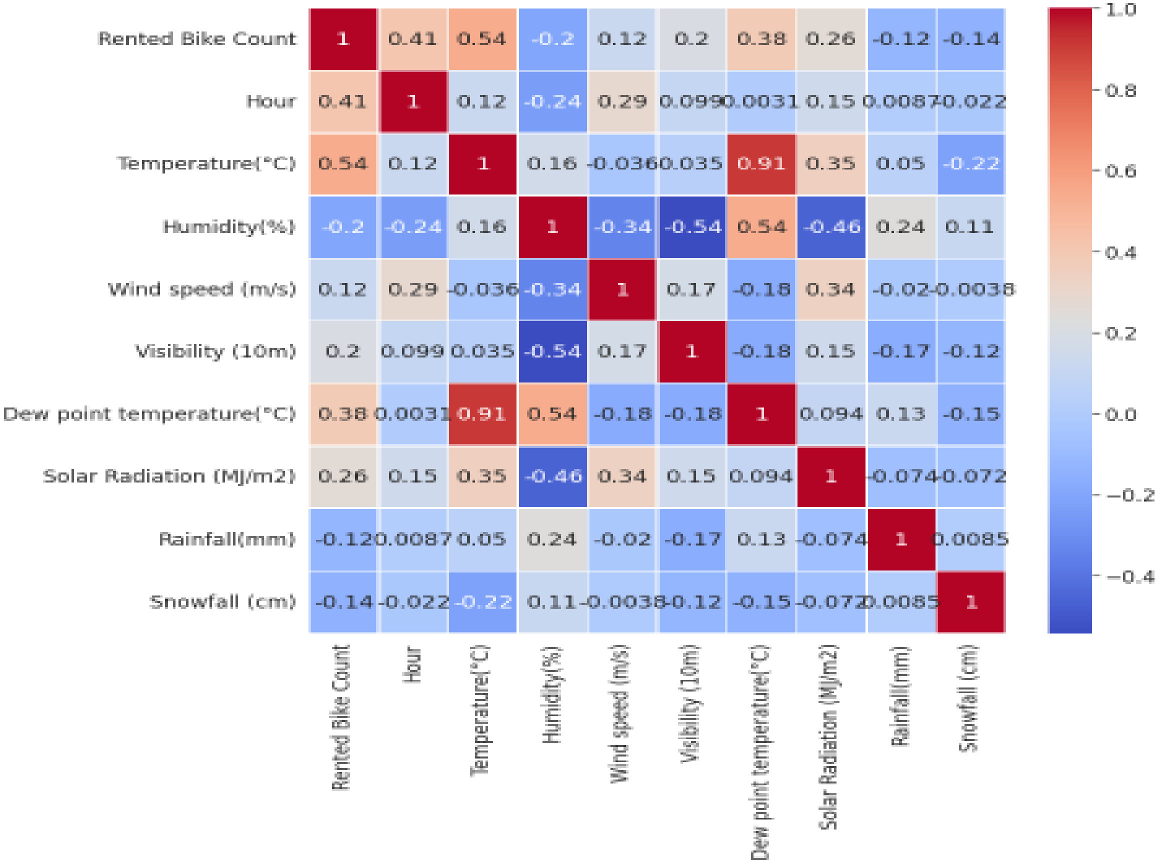
# Exploratory Data Analysis (continued)

## Sum plot for dependent variable.



# Exploratory Data Analysis (continued)

## Multicollinearity



# Feature Engineering

- There exists a high multicollinearity between ’Temperature(^C)’ and ’Dew poi nt Temperature(^C)’.
- Lets create a  new feature **’Temperature’** which comprises the addition of temperature and dew point temperature
- Then remove the features  ’Temperature(^C)’ and ’Dew point Temperature(^C)’ from the dataset.

```
# There exists a high multicollinearity between Temperature and Dew point Temperature
# Lets create a feature new feature Temperature which comprises the addition of temperature and dew point temperature
df['Temperature'] = df['Temperature(°C)'] + df['Dew point temperature(°C)']
df.drop('Temperature(°C)',axis = 1,inplace = True)
df.drop('Dew point temperature(°C)',axis = 1,inplace = True)
```

# Feature Engineering (continued)

## Variance Inflation Factor(VIF)

VIF measures the correlation and strength of correlation between the explanatory variables in a regression model

```
calc_vif(df[[i for i in df.describe().columns if i not in ['Date','Rented Bike Count']]])
```

| | variables | VIF |
|---|---|---|
| 0 | Hour | 3.877343 |
| 1 | Humidity(%) | 5.118281 |
| 2 | Wind speed (m/s) | 4.767596 |
| 3 | Visibility (10m) | 4.743852 |
| 4 | Solar Radiation (MJ/m2) | 2.073688 |
| 5 | Rainfall(mm) | 1.079532 |
| 6 | Snowfall (cm) | 1.117344 |
| 7 | Temperature | 2.183298 |

# Feature Engineering (continued)

## OneHotEncoder

- The input to this transformer should be an array-like of integers or strings, denoting the values taken on by categorical (discrete) features.
- This creates a binary column for each category and returns a sparse matrix or dense array (depending on the sparse parameter)

```
# One hot encoding
data = pd.get_dummies(df,columns = ['Hour','Seasons','Holiday','Functioning Day'])
data.drop(['Hour_0','Seasons_Autumn','Holiday_Holiday','Functioning Day_No'],axis = 1,inplace = True)
data.head()
```

# Feature Engineering (continued)

## Binning

Binning, also known as quantization is used for transforming continuous numeric features into discrete ones (categories).

There are two types of binning:

- Fixed-Width Binning
- Adaptive Binning

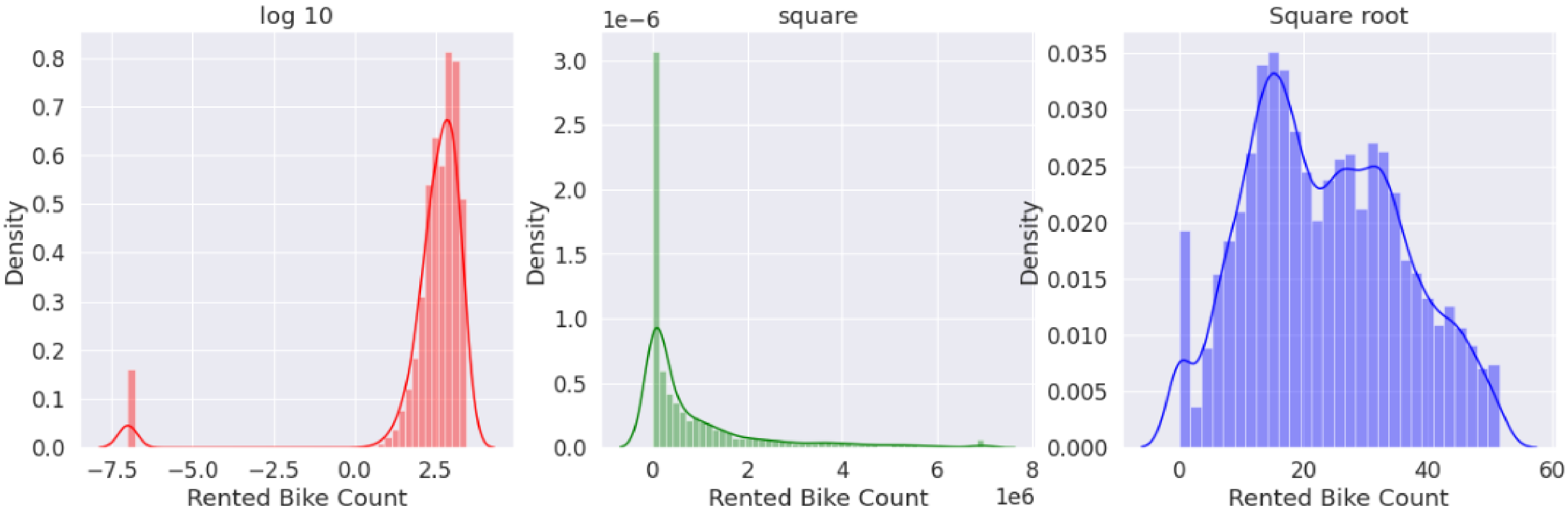
```
# Binning hours column
df3['morning_hours']=df['Hour'].apply(lambda x: 1 if x>=0 and x<8 else 0)
df3['afternoon_hours']=df['Hour'].apply(lambda x: 1 if x>=8 and x<16 else 0)
df3['evening_hours']=df['Hour'].apply(lambda x: 1 if x>=16 and x<24 else 0)

# Binning of highly imbalanced features

#df3['nVisibility']=df['Visibility (10m)'].apply(lambda x: 1 if x>=1650 else 0)
#df3['nRainfall']=df['Rainfall(mm)'].apply(lambda x:1 if x<2 else 0)
#df3['nSnowfall']=df['Snowfall (cm)'].apply(lambda x:1 if x<=1 else 0)
#df3['nSolar_Radiation']=df['Solar Radiation (MJ/m2)'].apply(lambda x:1 if x<=0.2 else 0)
```

# Dependent variable Transformation

Taking the square root of the dependent variable to transform it to a normal distribution since it is rightly skewed.



# Preparing dataset for modelling

## Train Test Split

Train test split is a model validation procedure that allows you to simulate how a model would perform on new/unseen data.

```
X = data[independent_variables].values
y = data[dependent_variable].values
```

```
# Splitting the dataset into the Training set and Test set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, random_state = 0)
```

- Train dataset: **(6570, 35)**
- Test dataset: **(2190, 35)**

# Preparing dataset for modelling

## Normalization:

**Normalization is a scaling technique in which values are shifted and rescaled so that they end up ranging between 0 and 1. It is also known as Min-Max scaling.**

```
[ ]  # Transforming data

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

$$X' = \frac{X - X_{min}}{X_{max} - X_{min}}$$

# Applying Model

## Linear Regression

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

```
LinearRegression()
```

## Model Validation

```
predict(LinearRegression(),X,y)
```

```
Train R^2 score is 0.7570950612010796
Train Adj R^2 is 0.7554195979506272
Train RMSE is: 6.109421984097278

Test R^2 is 0.7593713171799338
Test Adj R^2 is 0.7543208084453707
Test RMSE is: 6.064572434214715
```

# Applying Model (continued)

## Polynomial Regression

```
[ ]  # Polynomial Regression
     from sklearn.preprocessing import PolynomialFeatures
     poly_features = PolynomialFeatures(degree=2)
     X_train_poly = poly_features.fit_transform(X_train)
     poly_model = LinearRegression()
     poly_model.fit(X_train_poly, y_train)
     y_train_poly_predicted = poly_model.predict(X_train_poly)
     y_test_poly_predict = poly_model.predict(poly_features.fit_transform(X_test))
```

## Model Validation

```
Train R2_score            : 92.67282042922832
Adjusted train R2_score   : 92.51903167890896
Mean Squared Error        : 11.17600276448414
Train RMSE is             : 3.343052910811335

Test R2_score             : 90.0393171579194
Adjusted test R2_score    : 89.83025431841676
Mean Squared Error        : 15.560205340444622
Test RMSE is              : 3.944642612511889
```

# Applying Model (continued)

## Lasso Regression

```
[ ] # Cross validation and Hyperparameter tunning for Lasso
    from sklearn.model_selection import GridSearchCV
    lasso = Lasso()
    parameters = {'alpha': [0.0001,0.0002,0.0004,0.0007]}
    lasso_regressor = GridSearchCV(lasso, parameters, scoring='neg_mean_squared_error', cv=5)
    lasso_regressor.fit(X_train_poly, y_train)
```

## Model Validation

```
Train R2_score           : 65.1184431453987
Adjusted train R2_score  : 64.38632091664074
Mean Squared Error       : 53.204152030285044
Train RMSE is            : 7.294117632057016

Test R2_score            : 66.37852149273931
Adjusted test R2_score   : 65.67284680392088
Mean Squared Error       : 52.52221335791918
Test RMSE is             : 7.247221078311271
```

# Applying Model (continued)

## Ridge Regression

```
[ ]  # Cross validation and Hyperparameter tunning for Ridge
     from sklearn.model_selection import GridSearchCV
     ridge = Ridge()
     parameters = {'alpha': [1e-15,1e-10,1e-8,1e-5,1e-4,1e-3,1e-2,1,5,10,20,30,40,45,50,55,60,100]}
     ridge_regressor = GridSearchCV(ridge, parameters, scoring='neg_mean_squared_error', cv=3)
     ridge_regressor.fit(X_train_poly,y_train)
```

## Model Validation

```
Train R2_score            : 75.27734800806404
Adjusted train R2_score   : 74.75844906233777
Mean Squared Error        : 37.70897442031118
Train RMSE is             : 6.140763341825768

Test R2_score             : 77.23228894319769
Adjusted test R2_score    : 76.75442187344204
Mean Squared Error        : 35.56686472126996
Test RMSE is              : 5.96379616697871
```

# Applying Model (continued)

## Decision Tree

```
[ ] # Decision Tree Regressor

    decision_tree_reg = DecisionTreeRegressor(criterion = 'squared_error',splitter = 'best')
    decision_tree_reg.fit(X_train_dt,y_train_dt)
```

```
DecisionTreeRegressor()
```

## Model Validation

```
predict(DecisionTreeRegressor(),X,y)
```

```
Train R^2 score is 1.0
Train Adj R^2 is 1.0
Train RMSE is: 1.1157451977586e-15

Test R^2 is 0.8333628773654638
Test Adj R^2 is 0.8298653631310635
Test RMSE is: 5.046756716022452
```

# Applying Model (continued)

## Random Forest Regressor

```
# Cross validation and Hyperparameter tunning for Random Forest Regressor

rfr  = RandomForestRegressor(criterion='squared_error')
grid_values = {'n_estimators' : [50,100,150],'max_depth': [20,25,20],'min_samples_split' : [30,60,120],'min_samples_leaf':[1]
rfr = GridSearchCV(rfr,param_grid = grid_values ,cv = 5, verbose=2)
rfr.fit(X_train_dt,y_train_dt)
```

## Model Validation

```
Train R^2 score is 0.9367901717669347
Train Adj R^2 is 0.9363541750976385
Train RMSE is: 3.1165511270335733

Test R^2 is 0.8920805597252638
Test Adj R^2 is 0.8898154595329303
Test RMSE is: 4.061405314476434
```

# Applying Model (continued)

## Gradient Boost Regressor

```
# Cross validation and Hyperparameter tunning for Gradient Boosting Regressor

gbr = GradientBoostingRegressor(min_samples_leaf=1,criterion='squared_error')
grid_values = {'n_estimators' : [50,100,150],'max_depth': [50,60,70],'min_samples_split' : [60,90,120],'min_samples_leaf' :[1
gbr_gscv = GridSearchCV(gbr,param_grid = grid_values ,cv = 5)
```

## Model Validation

```
# Hyperparameter tuned GradientBoostingRegressor
predict(GradientBoostingRegressor(n_estimators=400,max_depth=15, min_samples_leaf=30,
                          min_samples_split=500),X,y)
gbrcv=GradientBoostingRegressor(n_estimators=400,max_depth=15, min_samples_leaf=30,
                          min_samples_split=500)
gbrcv.fit(X_train,y_train)
y_train_gbr_cv_pred = gbrcv.predict(X_train)
y_test_gbr_cv_pred = gbrcv.predict(X_test)
```

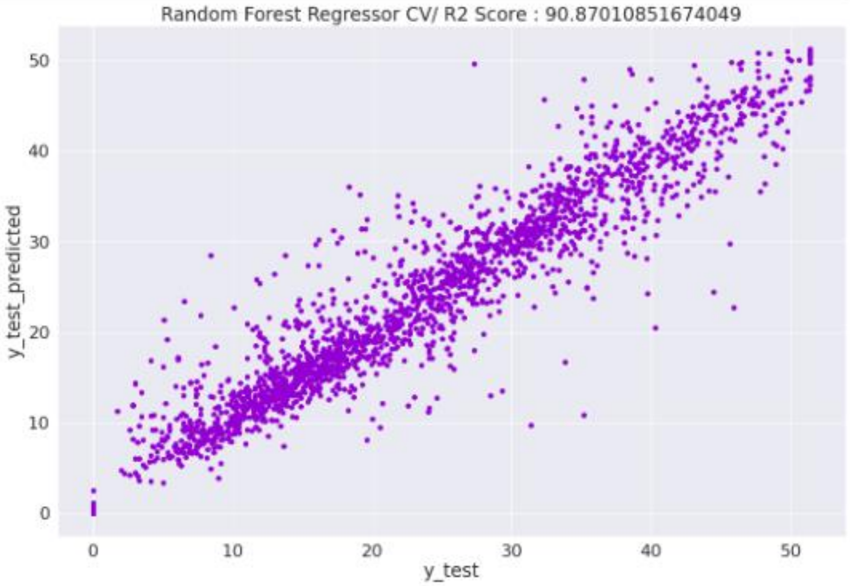
```
Train R^2 score is 0.9762463834103813
Train Adj R^2 is 0.9760825402548735
Train RMSE is: 1.9104990316317505

Test R^2 is 0.9409714242394717
Test Adj R^2 is 0.9397324849161397
Test RMSE is: 3.00370752821118
```

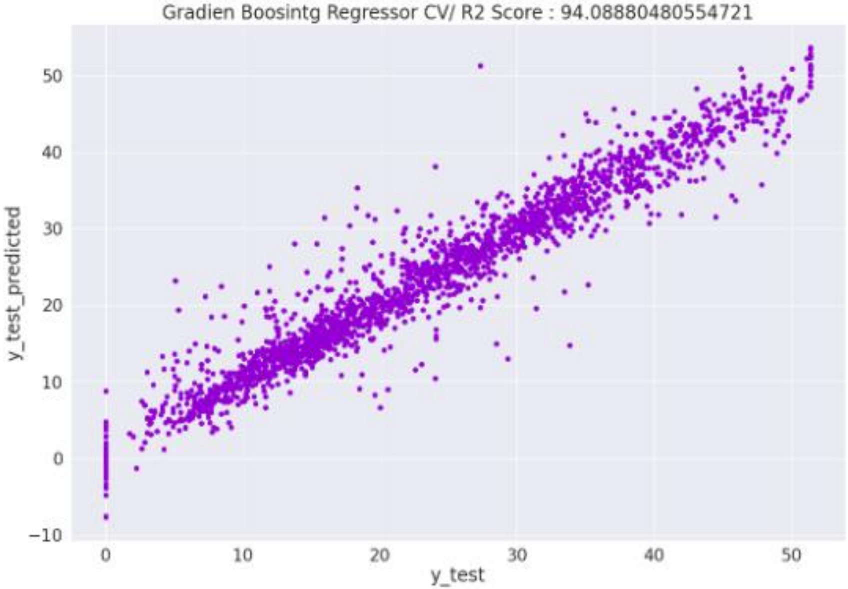
# Applying Model (continued)

## Visual representation of Decision Tree model's prediction

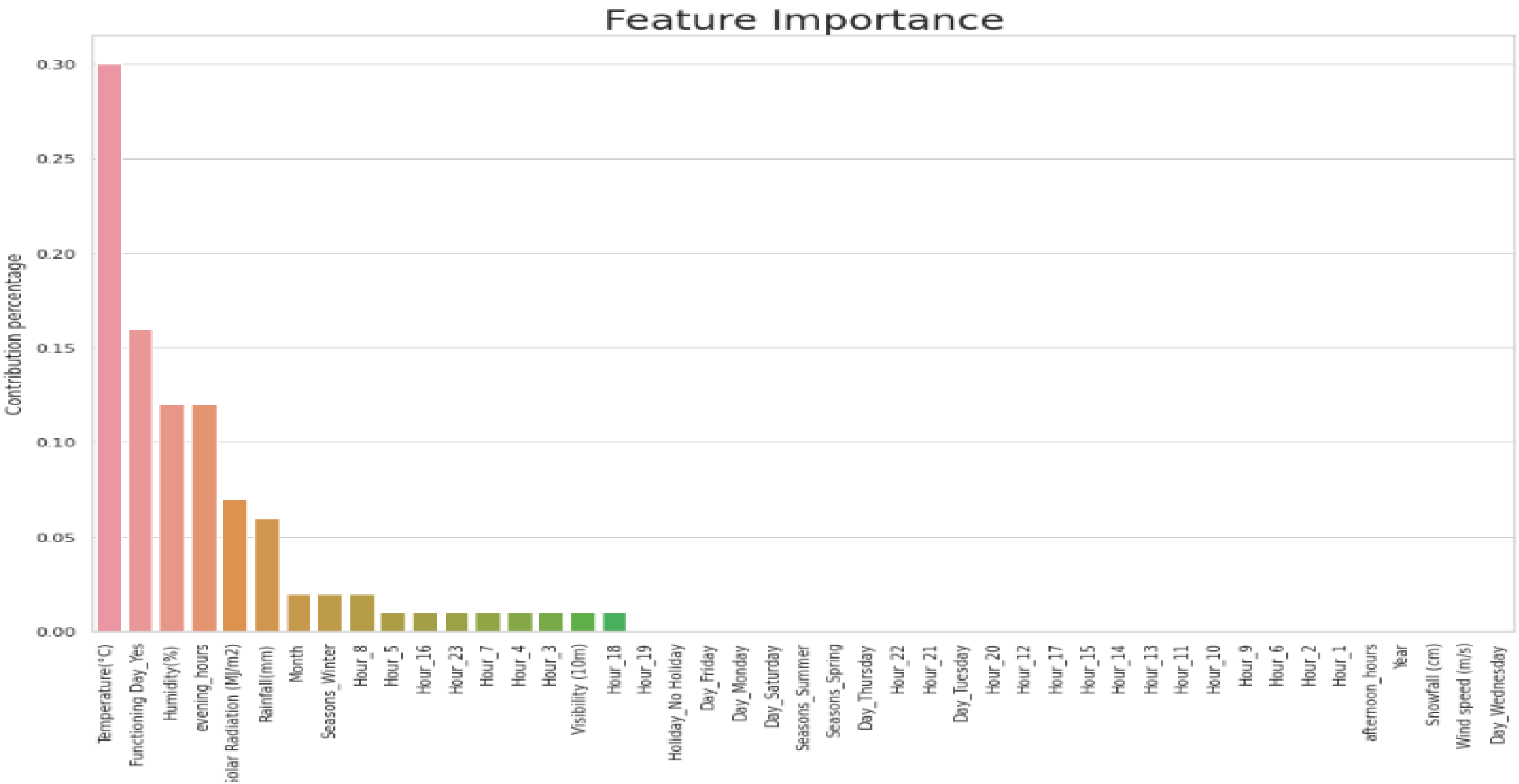
### Random Forest Regressor




### Gradient Boost Regressor

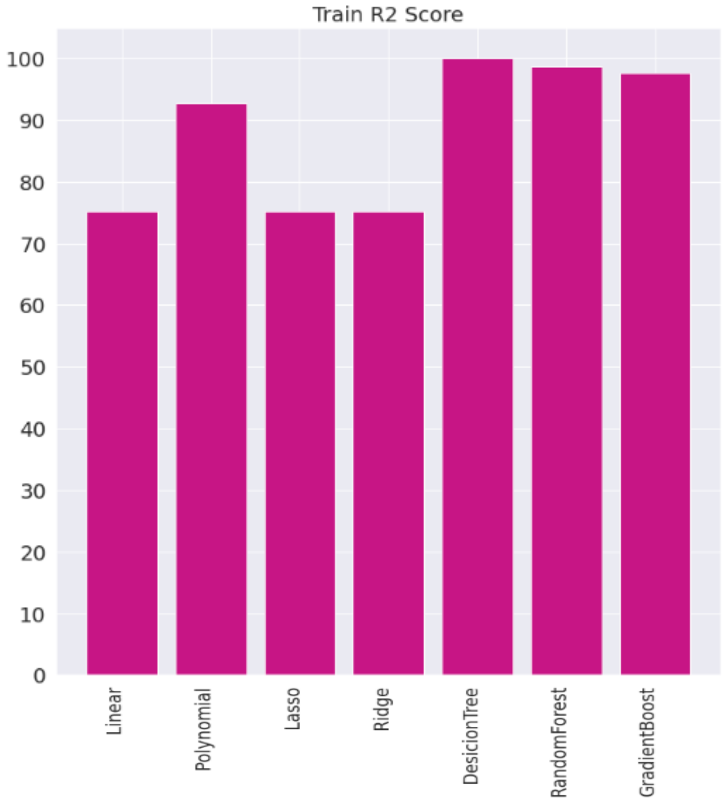


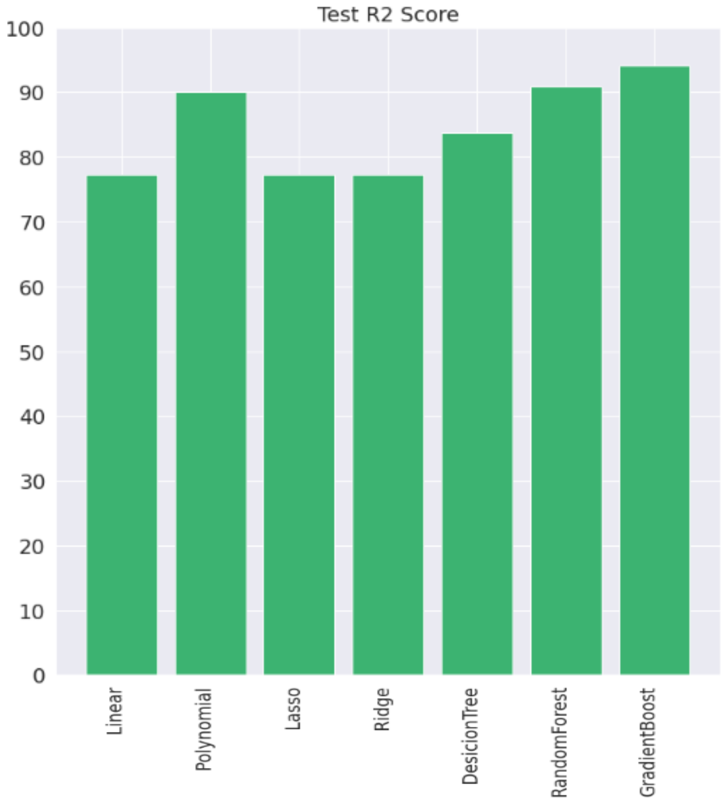
# Feature Importance



# Model Performance

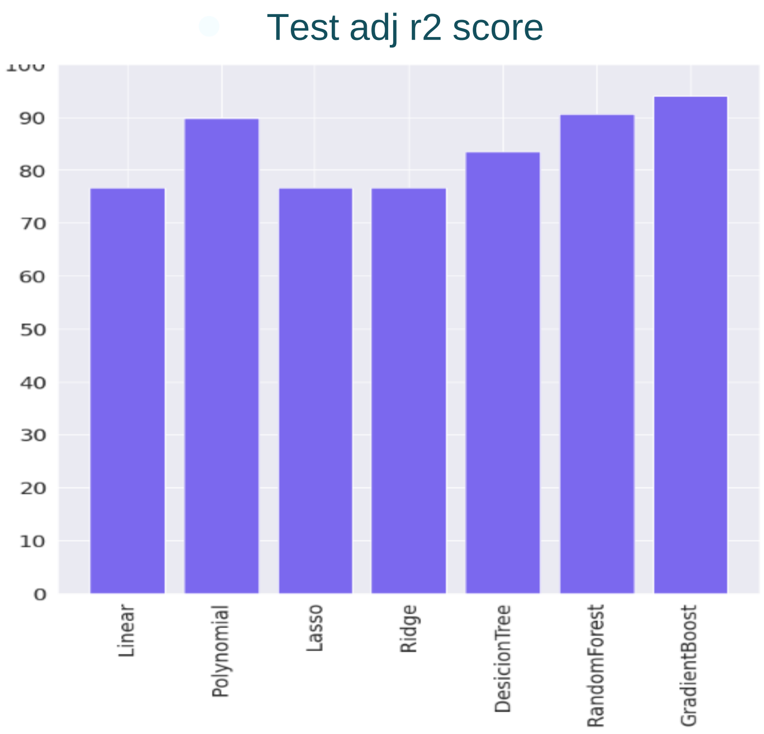
## Visualization of model score

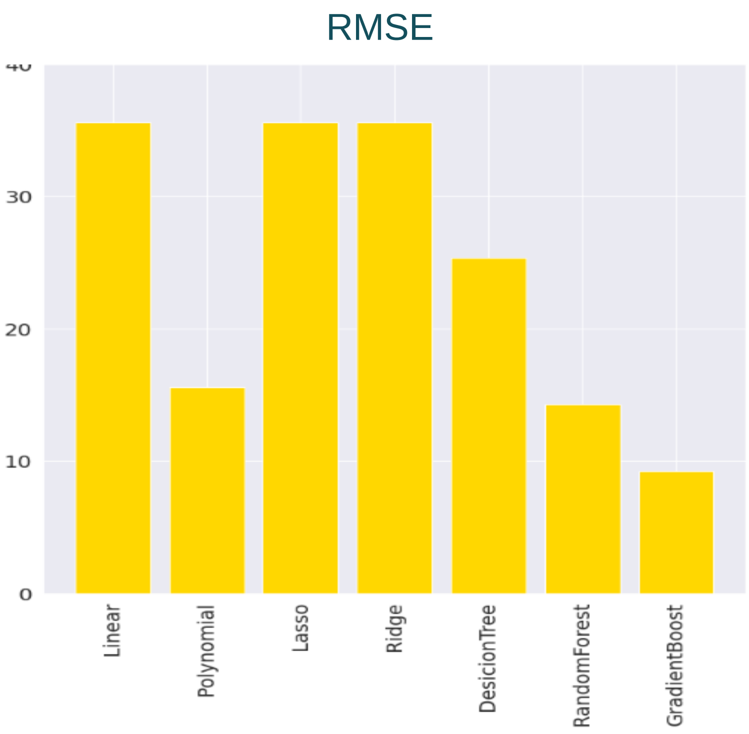


# Model Performance (continued)

## Visualization of model score



# Conclusion

- It’s quite obvious that most Bikes are rented during **Summer** season.
- And the least number of bikes are rented during **Winter** season.
- By using simple Polynomial Regressor algorithm we were able to get the **R2 score** of **90 percent.**
- Very little improvement in R2 score after using Lasso and Ridge with cross validation and hyper parameter tunning.
- By using simple Decision Tree algorithm, we couldn’t get the desired results as over fitting occurred.
- We got the best **R2 score** of **94 percent** using **GradientBoostRegressor** after cross validation and hyper parameter tunning.
- Top five most important features are **Temperature, Functioning day, Humidity, Solar Radiation, Evening hour.**

# Conclusion

- For the given dataset, **GradientBoostRegressor** has proven to be the best fit model with,

**Train r2 score : 97 %**
**Test r2 score : 94%**
**Test Adj r2 score : 94%**
**Test RMSE : 3.0037**

# Thank you