Regression Analysis on Seoul Bike Sharing Demand Prediction

Rohan s. Jagadale.
Data science trainee at Almabetter

Abstract

In recent days, Public rental bike sharing is becoming popular because of increased comfortableness and environmental sustainability. To predict the bike sharing demand, regression techniques are employed in this study to predict the demand on rental bikes in Seoul Bike sharing system. The prediction is carried out with the Seoul Bike data. The dataset contains weather information (Temperature, Humidity, Windspeed, Visibility, Dewpoint, Solar radiation, Snowfall, Rainfall), the number of bikes rented per hour and date information.

Problem Statement

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.

Introduction

In a span of few decade, the sharing of bicycle systems has seen enormous growth (Fishman, 2016). This system is a recently developed transportation system which provides people with bicycle for common use. Many countries have bike sharing systems, such as Ddareungi is a bike sharing system in South Korea, which started in the year 2015, known as Seoul bike in English. It was started to overcome issues like greater oil prices, congestion in traffic and pollution in the environment and to develop a healthy environment for citizens of Seoul to live.

Han River is the initial place where Ddareungi was first started in October 2015 in Seoul, a few months later, the total number of bike sharing stations touched 150 with as much as 1500 were there. In order to cover the entire people in Seoul, in

2016 there is a gradual incline in the number of docking stations. As large as 20,000 bikes were made available which was confirmed by Seoul Mayor Park won-soon

.

Managing a fleet of vehicles is an important task for rideshare or rental companies. The goal of this project was to build a machine learning model that could make an accurate prediction of how many bikes would be rented based on factors such as day of the week, time of day, temperature, humidity, precipitation, wind, and solar radiation. The dataset had 8,760 observations, with each observation representing one hour of one day. The target variable is 'Rented Bike Count' and there were 13 attributes to work with. The baseline rental count for any given day is 735 bikes..

Objective

The main objective of the project is to build a model to predict the demand of rental bikes and the bike count required at each hour for the stable supply of rental bikes.

Dataset Peeping

The dataset had 8,760 observations, with each observation representing one hour of one day. The target variable is 'Rented Bike Count' and there were 13 attributes to work with. The baseline rental count for any given day is 735 bikes.

1. We have NaN values in the dataset.
2. Changed the format of the Date.
3. Added some columns which are extracted from the Date column.

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 - a quantity representing the amount of water vapour in the atmosphere or in a gas.
- Wind speed - The rate at which air is moving in a particular area
- Visibility - a measure of the distance at which an object or light can be clearly discerned (10m)
- Dew point temperature - The temperature below which the water vapour in a volume of air at a constant pressure will condense into liquid water
- Solar radiation - The electromagnetic radiation emitted by the sun (MJ/m2)
- Rainfall - the quantity of rain falling within a given area in a given time (mm)
- Snowfall - the quantity of snow falling within a given area in a given time (cm)
- Seasons - Winter, Spring, Summer, Autumn
- Holiday - Holiday/No holiday
- Functional Day - NoFunc(Non Functional Hours), Fun(Functional hours

Challenges Faced

The following are the challenges faced in the data analysis:

- ➢ Conversion of Datetime features, categorical features.
- ➢ Added some of the columns
- ➢ Remove dummy variables
- ➢ Model Implementation

Approach

As the problem statement says, the main objective is to predict demand of rental bikes and the bike count required at each hour for the stable supply of rental bikes. We use supervised learning regression analysis Linear Regression, Lasso Regression, Random Forest Regression for the purpose of training the dataset to predict future supply and demand of bikes.

Tools Used

The whole project was done using python, in google colaboratory. Following libraries were used for analysing the data and visualizing it and to build the model to predict the bike count required at each hour for the stable supply of rental bikes.

- Pandas: Extensively used to load and wrangle with the dataset.
- Matplotlib: Used for visualization.
- Seaborn: Used for visualization.
- Datetime: Used for analysing the date variable.
- Warnings: For filtering and ignoring the warnings.
- Numpy: For some math operations in predictions.
- Sklearn: For the purpose of analysis and prediction.
- Datetime: For reading the date.
- Statsmodels: For outliers influence.

The below table shows the dataset in the form of Pandas DataFrame.

Date, Rented Bike count, Hour ,Temperature, Humidity ,Wind speed, Visibility, Dew point temperature, Solar radiation, Snowfall, etc.

Pandas DataFrame

	Date	Rented Bike Count	Hour	Temperature(°C)	Humidity(%)	Wind speed (m/s)	Visibility (10m)	Dew point temperature(°C)	So Radiati (MJ/m
0	01/12/2017	254	0	-5.2	37	2.2	2000	-17.6	(
1	01/12/2017	204	1	-5.5	38	0.8	2000	-17.6	(
2	01/12/2017	173	2	-6.0	39	1.0	2000	-17.7	(
3	01/12/2017	107	3	-6.2	40	0.9	2000	-17.6	(
4	01/12/2017	78	4	-6.0	36	2.3	2000	-18.6	(

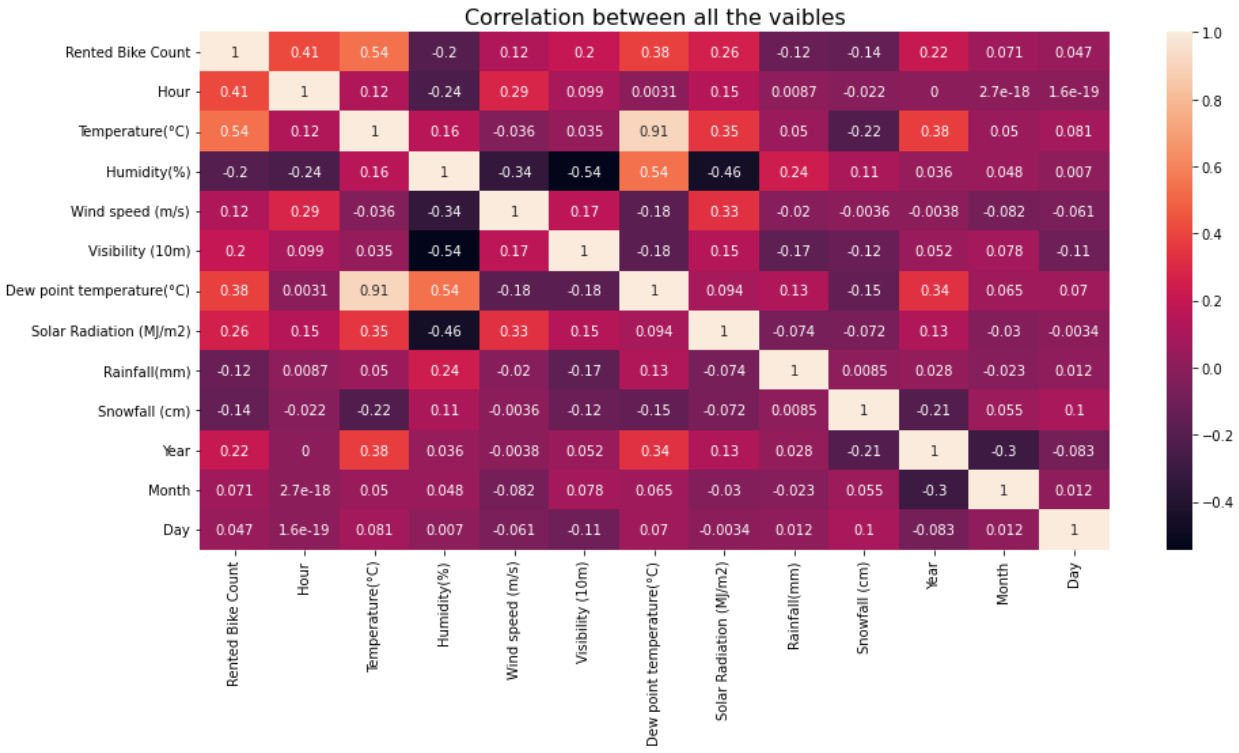
Description

The below table shows the mathematical calculations such as count, mean,standard deviation, minimum and percentiles of all features of the dataset.

	Rented Bike Count	Hour	Temperature(°C)	Humidity(%)	Wind speed (m/s)	Visibility (10m)	Dew pc temperature(
count	8760.000000	8760.000000	8760.000000	8760.000000	8760.000000	8760.000000	8760.000
mean	704.602055	11.500000	12.882922	58.226256	1.724909	1436.825799	4.073
std	644.997468	6.922582	11.944825	20.362413	1.036300	608.298712	13.060
min	0.000000	0.000000	-17.800000	0.000000	0.000000	27.000000	-30.600
25%	191.000000	5.750000	3.500000	42.000000	0.900000	940.000000	-4.700
50%	504.500000	11.500000	13.700000	57.000000	1.500000	1698.000000	5.100
75%	1065.250000	17.250000	22.500000	74.000000	2.300000	2000.000000	14.800
max	3556.000000	23.000000	39.400000	98.000000	7.400000	2000.000000	27.200

Correlation Heatmap

The above heatmap shows the correlation of all variables in the merged dataset.



Preparing the new data frame with selected columns (add few columns)

	Year	Month	Day	Day_name	Rented Bike Count	Hour	HoliDay	Visibility	Temperature
0	2017	1	12	Thursday	254	0	0	1	-5.2
1	2017	1	12	Thursday	204	1	0	1	-5.5
2	2017	1	12	Thursday	173	2	0	1	-6.0
3	2017	1	12	Thursday	107	3	0	1	-6.2
4	2017	1	12	Thursday	78	4	0	1	-6.0

Variance Inflation Factor

The Variance Inflation Factor is used when we have the multi-collinearity between the features. The factor helps in reducing the inflation between the features by dropping some of the features which are having the high correlation among them.

In the given dataset we don't have much correlation between the features and the features which are dropped in the data column to reduce the linearity among them and also the data in it was splitted.

Data Modelling

After the data preparation was completed it is ready for the purpose of building the model to predict the sales. Only numerical valued features are taken into consideration. The data was combined and labelled as X and y as independent and dependent variables respectively. The sales feature is taken as dependent variable (y) and remaining all are considered as independent variables (X).

Splitting the data

The train_test_split was imported from the sklearn.model_selection. The data is now divided into 80% and 20% as train and test splits respectively. 80% of the data is taken for training the model and 20% is for test and the random state was taken as 42.

Metrics

The metrics are tools used for evaluating the performance of a regression model. The following are the metrics that have been used in the analysis of the data.

Mean Squared Error

The mean squared error (MSE) is a commonly used metric for estimating errors in regression models. It provides a positive value as the error gets closer to zero. It is simply the average of the squared difference between the target value and the value predicted by the regression model.

$$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2$$

Root Mean Squared Error

The root-mean-Square error or RMSE is a frequently used measure to evaluate the differences between the values that a model has predicted and the values that were observed. It is computed by taking the second sample moment and dividing it by the quadratic mean of the differences.

RMSD is a non-zero measure that shows a perfect fit to the data, and it is generally better than a higher one. It is not used to evaluate the relationships among different types of data.

RMSD is the sum of the average of all errors. It is sensitive to outliers.

$$\mathrm{RMSD} = \sqrt{\frac{\sum_{i=1}^{N}\left(x_i - \hat{x}_i\right)^2}{N}}$$

R2 score

The goodness-of-fit evaluation should not be performed on the R2 linear regression because it quantifies the degree of linear correlation between the two values. Instead, the linear correlation should only be taken into account when evaluating the Ypred-Yobs relationship.

The metric helps us to compare our current model with a constant baseline and tells us how much our model is better. The constant baseline is chosen by taking the mean of the data and drawing a line at the mean. R^2 is a scale-free score that implies it doesn't matter whether the values are too large or too small, the R^2 will always be less than or equal to 1.

$$R^2 = 1 - \frac{RSS}{TSS}$$

Scaling the data

To normalise the data standardscaler was used from sklearn.preprocessing. It scales the data in the form of standard deviation of the feature divided from the difference of variable and its mean of the feature. At first the training data was made fit into the scaling function and test data is transformed now. The output we get are X_train, X_test, y_train, y_test. The below figure shows the shape of each dataset.

```
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: (7008, 12)
Size of X_test is: (1752, 12)
Size of y_train is: (7008,)
Size of y_test is: (1752,)
```

Linear Regression

The next step is implementing and training the model. A linear regression is a type of statistical procedure that involves finding a relationship between a linearly-related variable and a prediction. Mathematically it solves problem in the form of:

$$Y_i = f(X_i, \beta) + e_i$$

The first thing we do is to fit the training set and train the model. The score we obtained in training the model is 48.79%. To get the predicted data we fit the independent variables in the regression model and define the predicted variable. The available variables in hand at last are the actual y values and the predicted values. With these we can make the visualizations as follows.

Training data

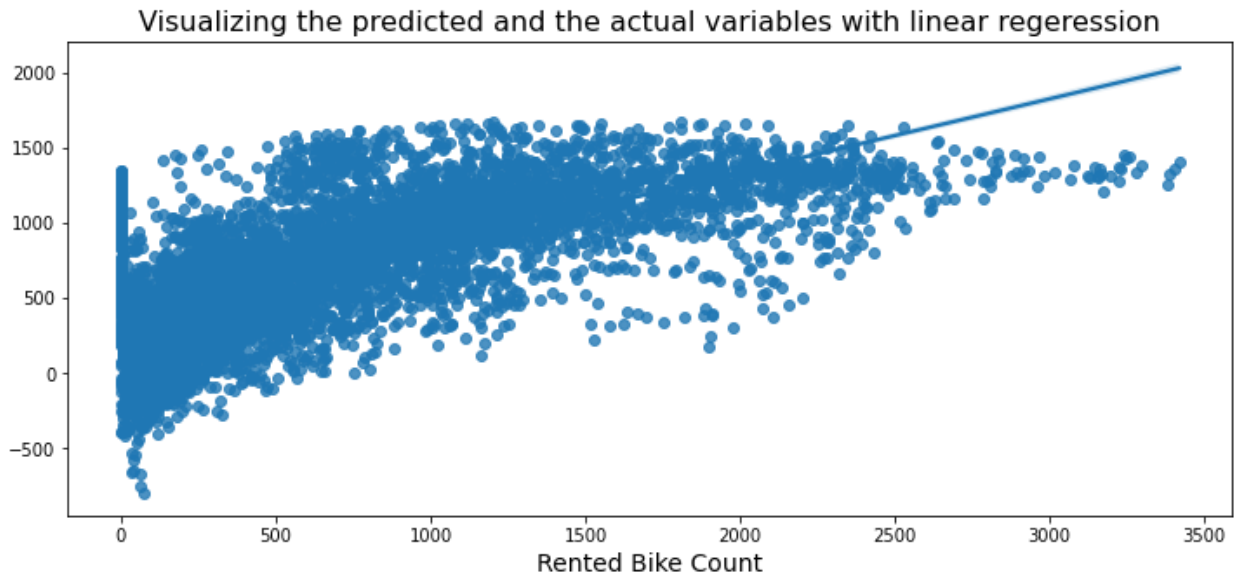
```
MSE_train = mean_squared_error(y_train, pred_train)
print(f'MSE= {MSE_train}')

RMSE_train = np.sqrt(MSE_train)
print(f'RMSE= {RMSE_train}')

R2_Score_train = r2_score(y_train, pred_train)
print(f'R2_Score= {R2_Score_train}')

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

```
MSE= 212922.60609394923
RMSE= 461.435375858797
R2_Score= 0.4879281400858403
Adjusted R2 :  0.48439457923536877
```



The above figure shows how the model was trained and fit with the data. It consists of both predicted and actual training dataset in it. With the accuracy of 48% of the model can predict correctly the percentage on numbers of bikes will be rented in the near future.

Test dataset

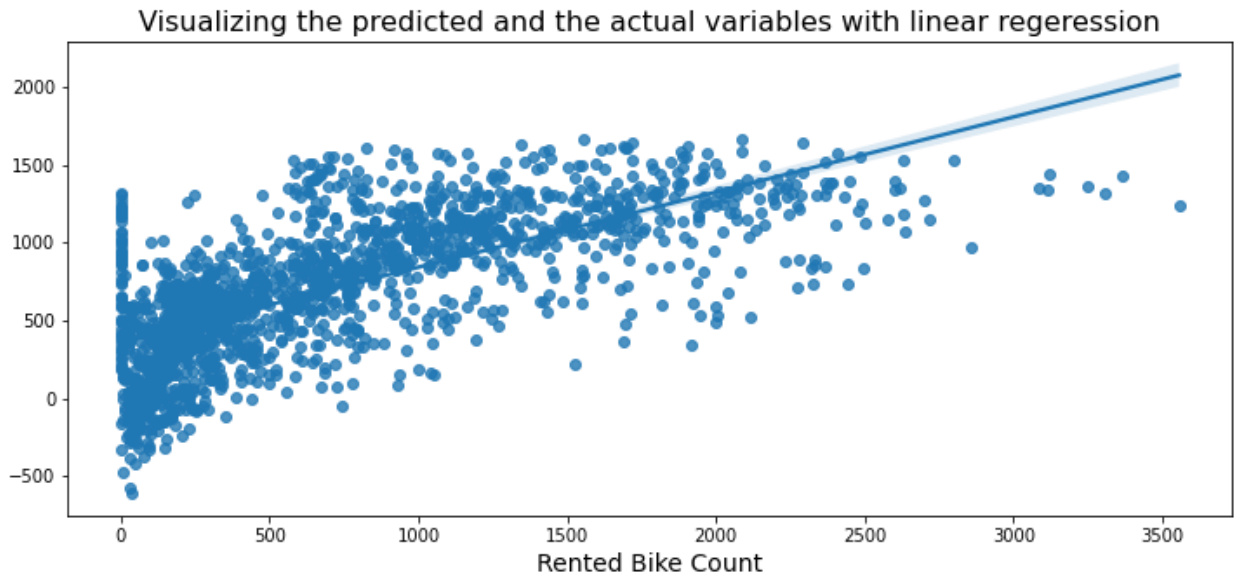
```
MSE_test = mean_squared_error(y_test, pred_test)
print(f'MSE= {MSE_test}')

RMSE_test = np.sqrt(MSE_test)
print(f'RMSE= {RMSE_test}')

R2_Score_test = r2_score(y_test, pred_test)
print(f'R2_Score= {R2_Score_test}')

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

```
MSE= 217066.86427979858
RMSE= 465.9043509989991
R2_Score= 0.4790139820446465
Adjusted R2 :  0.47541890889026794
```



The above figure shows how the model was trained and fit with the data. It consists of both predicted and actual training dataset in it. With the accuracy of 48% of the model can predict correctly the percentage on numbers of bikes will be rented in the near future.

Lasso Regression

Lasso Regression is a type of linear Regression that uses shrinkage**.** Shrinkage is where data values are shrunk towards a central point , like the mean. The lasso encourages simple, sparse models (i.e. model the fewer parameters). The acronym "LASSO" stands for Least Absolute Shrinkage and Selection operator.

The first thing we do is to fit the training set and train the model. The score we obtained in training the model is 48.79%. To get the predicted data we fit the independent variables in the regression model and define the predicted variable.

Training data

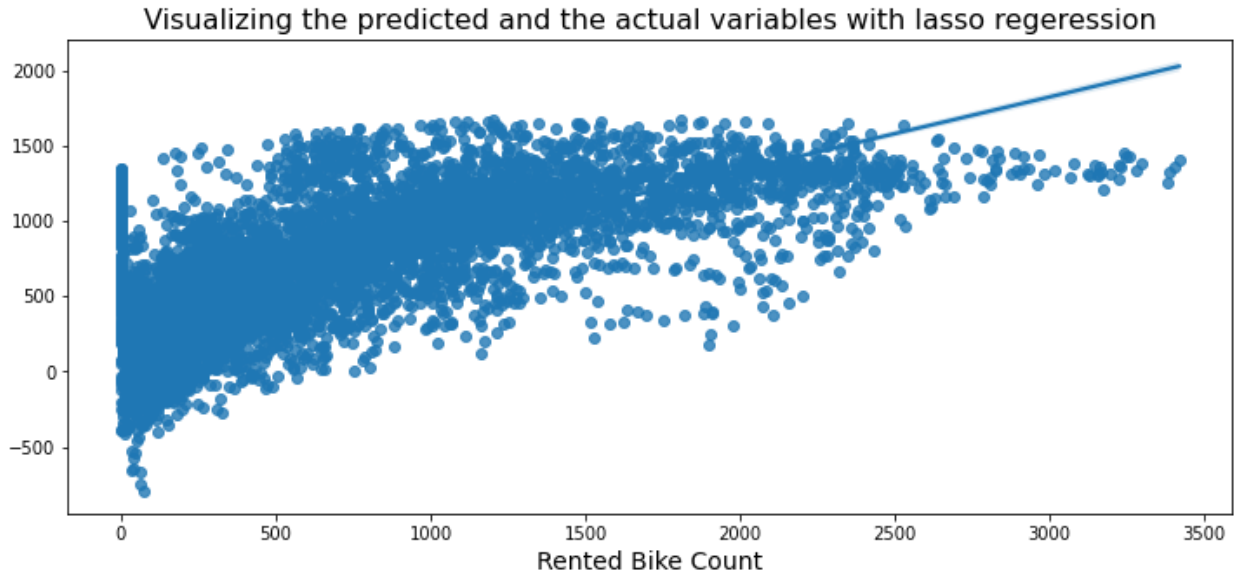
```
MSE_train = mean_squared_error(y_train, pred_train)
print(f'MSE= {MSE_train}')

RMSE_train = np.sqrt(MSE_train)
print(f'RMSE= {RMSE_train}')

R2_Score_train = r2_score(y_train, pred_train)
print(f'R2_Score= {R2_Score_train}')

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

```
MSE= 212922.60609394923
RMSE= 461.435375858797
R2_Score= 0.4879281400858403
Adjusted R2 :  0.48439457923536877
```



The above figure shows how the model was trained and fit with the data. It consists of both predicted and actual training dataset in it. With the accuracy of 49% of the model can predict correctly the percentage on numbers of bikes will be rented in the near future.

Test dataset

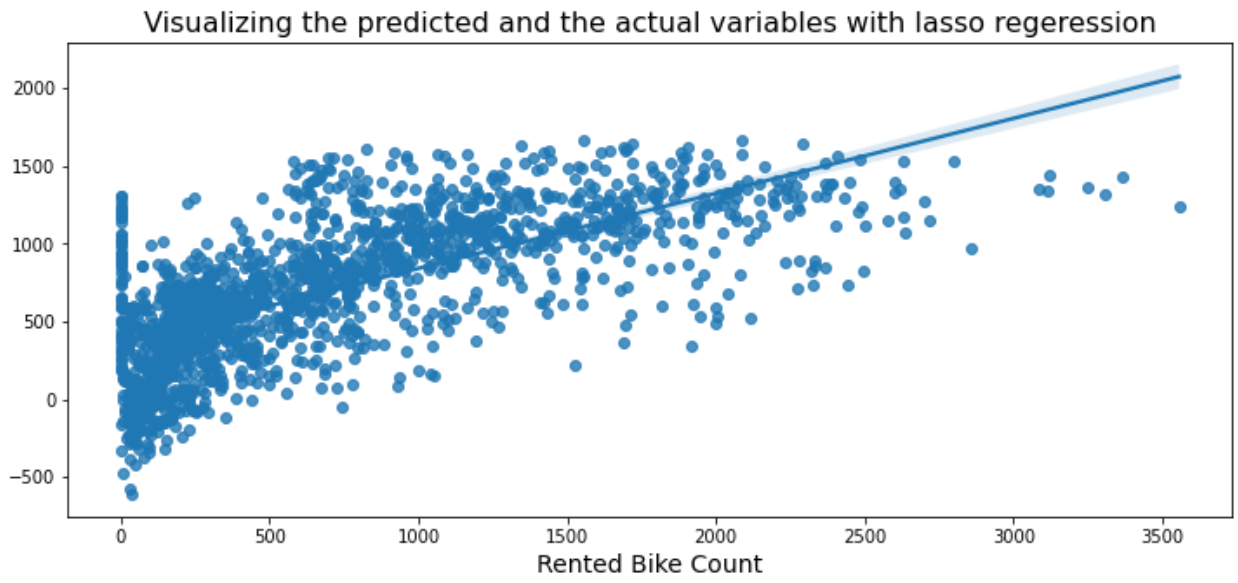
```
MSE_test = mean_squared_error(y_test, pred_test)
print(f'MSE= {MSE_test}')

RMSE_test = np.sqrt(MSE_test)
print(f'RMSE= {RMSE_test}')

R2_Score_test = r2_score(y_test, pred_test)
print(f'R2_Score= {R2_Score_test}')

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

```
MSE= 217066.86427979858
RMSE= 465.9043509989991
R2_Score= 0.4790139820446465
Adjusted R2 :  0.47541890889026794
```



The above figure shows how the model was trained and fit with the data. It consists of both predicted and actual training dataset in it. With the accuracy of 48% of the model can predict correctly the percentage on numbers of bikes will be rented in the near future.

Random Forest Regression

Random Forest is a supervised learning algorithm which uses ensemble learning methods for statistical regression. Ensemble learning method is a technique that combines the predictions from multiple machine learning algorithms to make a more accurate prediction. A random forest operates by constructing several decision trees during training time and outputs the mean of the classes at the prediction of all the trees.

The model was imported and trained with the data available in the training dataset. Defined the predicted variable and checked the score of the model.

Training data

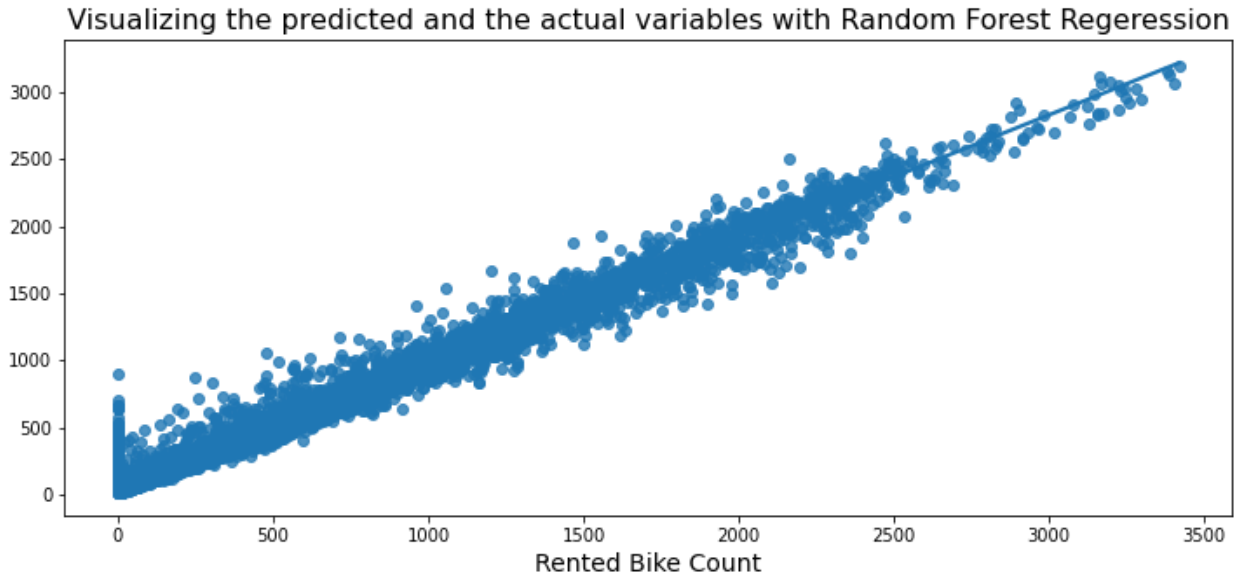
```
MSE_train = mean_squared_error(y_train, pred_train)
print(f'MSE= {MSE_train}')

RMSE_train = np.sqrt(MSE_train)
print(f'RMSE= {RMSE_train}')

R2_Score_train = r2_score(y_train, pred_train)
print(f'R2_Score= {R2_Score_train}')

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

```
MSE= 10424.451827965326
RMSE= 102.10020483801844
R2_Score= 0.9749295364449164
Adjusted R2 :  0.9747565372714483
```



From the above figure we can see that the best fit line passes through the points and shows how good the model is trained and fits with the data. The model with the random forest regression has an accuracy of **97.40%** on the training dataset. The below figure shows the result of the evaluation of the model.

Test dataset

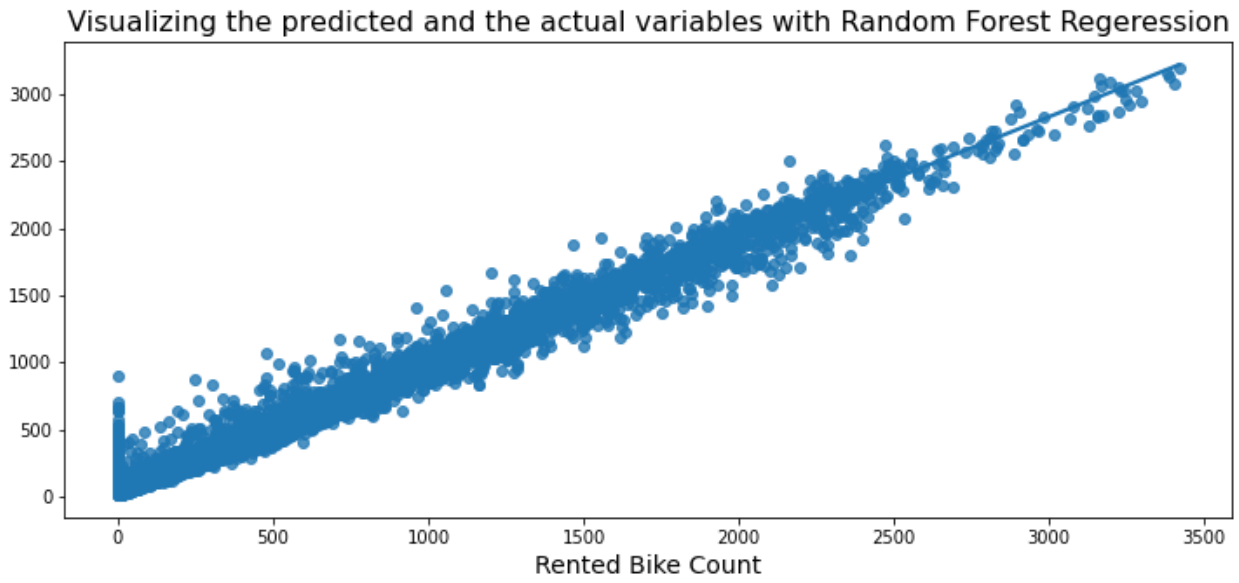
```
MSE_test = mean_squared_error(y_test, pred_test)
print(f'MSE= {MSE_test}')

RMSE_test = np.sqrt(MSE_test)
print(f'RMSE= {RMSE_test}')

R2_Score_test = r2_score(y_test, pred_test)
print(f'R2_Score= {R2_Score_test}')

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

```
MSE= 74899.85868671804
RMSE= 273.67838549421117
R2_Score= 0.8202315251934862
Adjusted R2 :  0.8189910296801577
```



The above figure shows how good the model has predicted and performed on the test data. It has an accuracy of **81.40 %** on the test dataset..

Hyperparameter Tuning

In machine learning, hyperparameter optimization or tuning is the problem of choosing a set of optimal hyperparameters for learning algorithm. A hyperparameter is a parameter whose value is used to control the learning process. By contrast, the values of other parameters are learned.

Random Forest Hyperparameters we'll be looking at:

- Max_depth : The max_depth of a tree in Random Forest is defined as the longest path between the leaf root node and the left node.
- Min_samples_leaf : This hyperparameter specifies the minimum number of samples that should be in the leaf node after splitting a node.
- Min_samples_split : A parameter that tells the decision tree in a random forest the minimum required number of observations in any given node.
- N_estimator : We know that a Random forest algorithm is a nothing but number of trees.
- Max_features : This resembles the number of maximum features provided to each tree in a Random Forest.

Build Random Forest Model with hyperparameters

```
n_estimators= [160,210,10]
max_depth = [25,35,1]
min_samples_split = [2,5,1]
min_samples_leaf = [1,5,1]
max_features= [4,10,1]
```

Cross Validation on Random Forest

Cross-validation is a resampling method that uses different portions of the data to test and train a model on different iterations. It is mainly used in settings where the goal is prediction, and one wants to estimate how accurately a predictive model will perform in practice.

Cross-validation gives a more accurate measure of model quality, which is especially important if you are making a lot of modeling decisions.

```
rf_grid = RandomizedSearchCV(estimator= rf_model , param_distributions= random_grid , cv = 5
```

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

```
Fitting 5 folds for each of 50 candidates, totalling 250 fits
RandomizedSearchCV(cv=5, estimator=RandomForestRegressor(), n_iter=50,
                   n_jobs=-1,
                   param_distributions={'max_depth': [25, 35, 1],
                                        'max_features': [4, 10, 1],
                                        'min_samples_leaf': [1, 5, 1],
                                        'min_samples_split': [2, 5, 1],
                                        'n_estimators': [160, 210, 10]},
```

Made a cross validation on the model with 5 CV.

Best parameters and the Accuracy of model

```
rf_grid.best_params_
```

```
{'max_depth': 25,
 'max_features': 4,
 'min_samples_leaf': 1,
 'min_samples_split': 2,
 'n_estimators': 160}
```

```
print(f' Train Acuuracy : {rf_grid.score(X_train,y_train):.3f}')
print(f' Test Acuuracy : {rf_grid.score(X_test,y_test):.3f}')
```

```
 Train Acuuracy : 0.976
 Test Acuuracy : 0.826
```

The above figure shows how good the model has predicted and performed on the test data. It has an accuracy of 97.60% on the training data set and 82.60% on the test dataset.

Conclusion :

- ➔ The dataset had 8,760 observations, with each observation representing one hour of one day.The target variable is ‘Rented Bike Count’ and there were 13 attributes to work with.
- ➔ We calculate the numbers of bikes rented on Function day, holidays, monthly and day wise.
- ➔ In 2017, the count on rented bikes was only 8.5% (744) but within one year use of these bikes increased to 91.5% (8016).
- ➔ In the rainy season and snow season use of these bikes was very less.
- ➔ In the months of May, July and October we have a high number of rented bikes counts in these 3 month periods. In all over months we have maximum rented bikes counts in the month of June.
- ➔ Use of these bikes in the first 10 days of every month is much higher than other days.
- ➔ It is assumed that 7 in the day of week as the Sunday and stores are mostly closed during Sundays.
- ➔ We assume All schools, some offices are closed on public holidays and weekends. It shows very less counts on holidays and high counts during working days.
- ➔ Linear Regression, Lasso Regression and Random Forest Regression are used to train the model.
- ➔ As per the evaluation it is better to implement the Random Forest Regression rather than going for Linear and Lasso Regression.
- ➔ When it comes to the accuracy the Random Forest Regression is performing well on the test dataset with the accuracy of 81.40% and with hyperparameter tuning used on the t dataset it shows slightly more accuracy of 82.60 %.
- ➔ As it can predict the daily bike count required at each hour for the stable supply of rental bikes with some accuracy.