Capstone Project - 2

Seoul Bike Sharing Demand Prediction

Zuber Ahmad

Contents :

- PREPROCESSING THE DATASHEET
- EXPLODATORY DATA ANALYSIS
  - DATA EXPLORATION
  - DATA ANALYSIS
  - UNIVARIATE ANALYSIS
  - Outlier Checking & Treatment
  - Correlation Heatmap
- Modelling
  - Linear Regression
  - Lasso & Ridge Regression
  - Elastic Net Regression
  - Decision Tree
  - Random Forest
  - Gradient Boosting
  - Hyperparameter
- EVALUATION METRICS
  - Mean Absolute Error
  - Mean Square Error
  - Route Mean Square Error
  - R2 Score
  - Comparison
  - Conclusion

INTRODUCTION

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

OBJECTIVES

- Primary Objective

  To build a superior statistical model that predict the number of bicycle that can be rented with availability of data.

  The project goal is to predict number of rental bikes required at a particular time of the day.

- Secondary Objective

  To learn how real time data is represented in datasets.

  To understand how to preprocess such data.

  To study comparison of results achieved by various Machine Learning techniques such as regression, decision trees, random forest, Gradient Boosting and Hyperparameter tuning.

Attributes Information

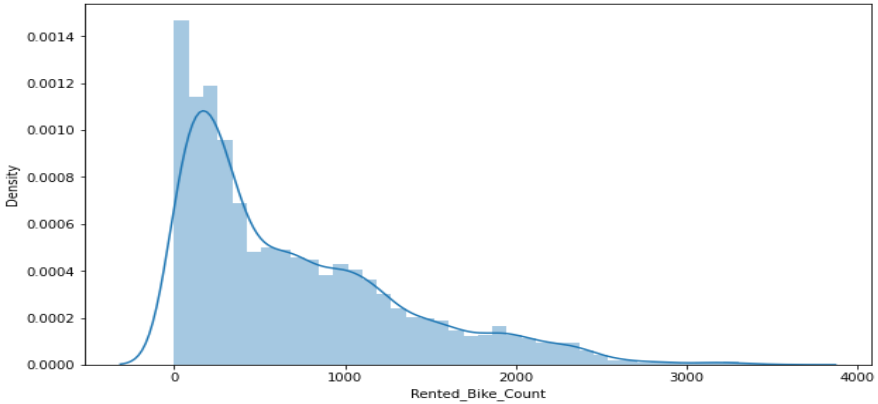
- **Date : year-month-day**
- **Rented Bike count - Count of bikes rented at each hour**
- **Hour - Hour of he day**
- **Temperature-Temperature in Celsius**
- **Humidity - %**
- **Windspeed - m/s**
- **Visibility - 10m**
- **Dew point temperature - Celsius**
- **Solar radiation - MJ/m2**
- **Rainfall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

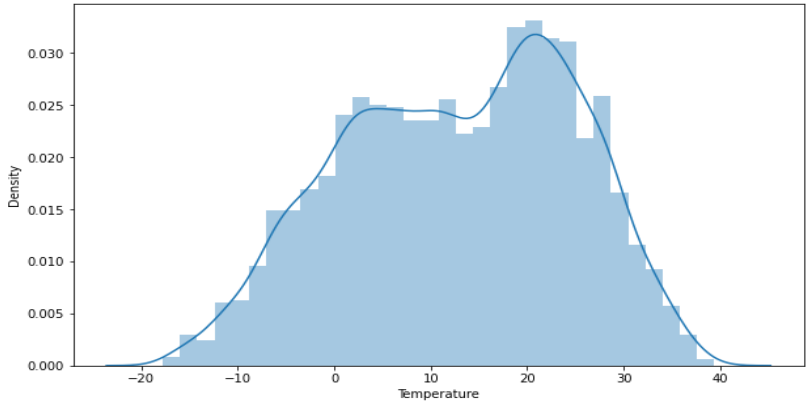
Exploratory Data Analysis

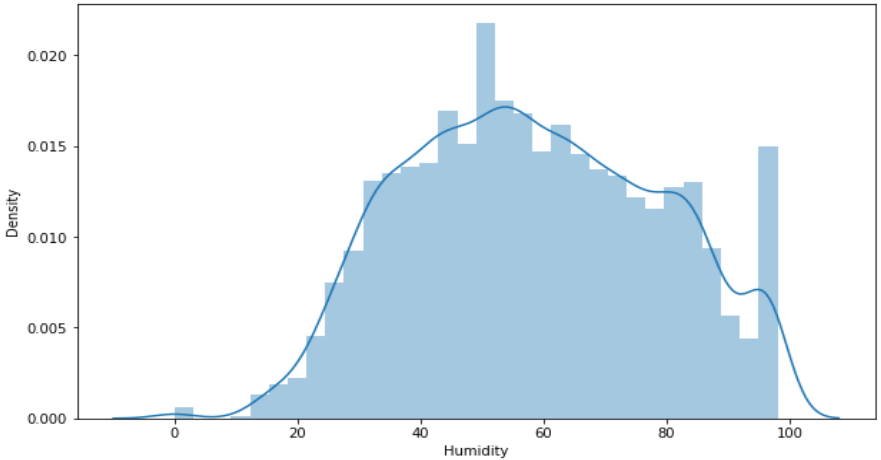
```
bike_df.head()
```

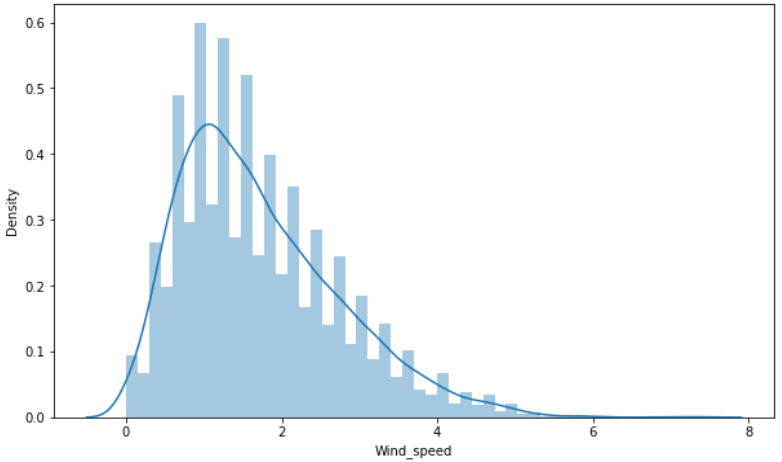
	Date	Rented Bike Count	Hour	Temperature(°C)	Humidity(%)	Wind speed (m/s)	Visibility (10m)	Dew point temperature(°C)	Solar Radiation (MJ/m2)	Rainfall(mm)	Snowfall (cm)	Seasons	Holiday	Functioning Day
0	01/12/2017	254	0	-5.2	37	2.2	2000	-17.6	0.0	0.0	0.0	Winter	No Holiday	Yes
1	01/12/2017	204	1	-5.5	38	0.8	2000	-17.6	0.0	0.0	0.0	Winter	No Holiday	Yes
2	01/12/2017	173	2	-6.0	39	1.0	2000	-17.7	0.0	0.0	0.0	Winter	No Holiday	Yes
3	01/12/2017	107	3	-6.2	40	0.9	2000	-17.6	0.0	0.0	0.0	Winter	No Holiday	Yes
4	01/12/2017	78	4	-6.0	36	2.3	2000	-18.6	0.0	0.0	0.0	Winter	No Holiday	Yes

Exploratory Data Analysis

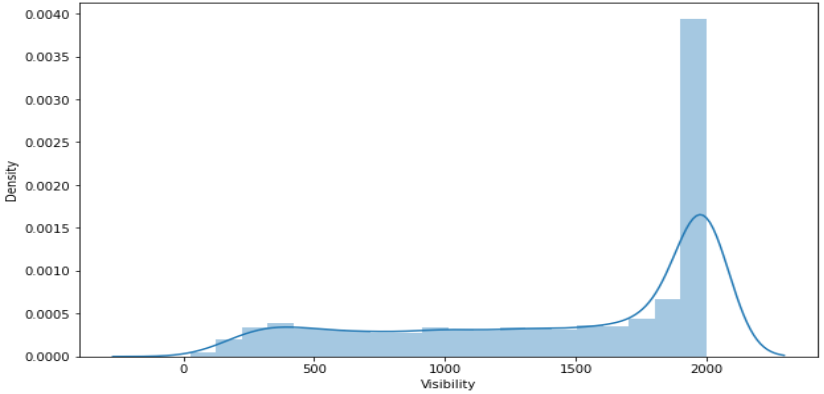


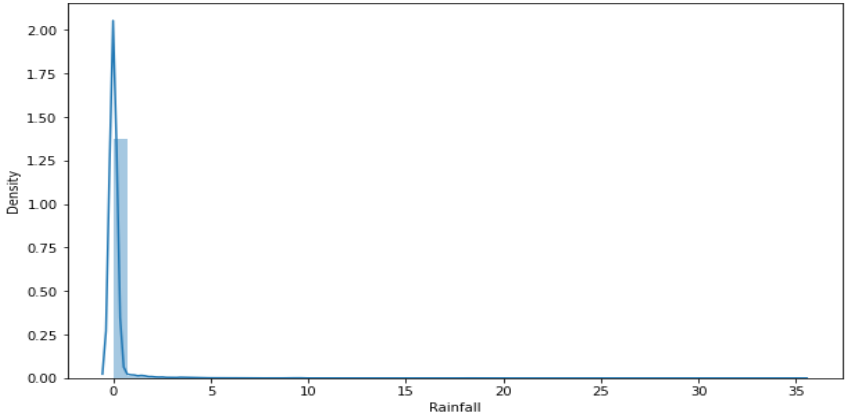


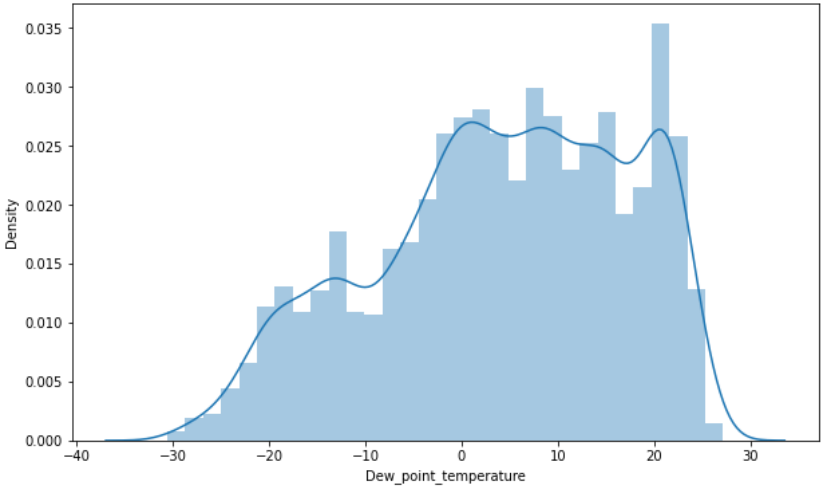


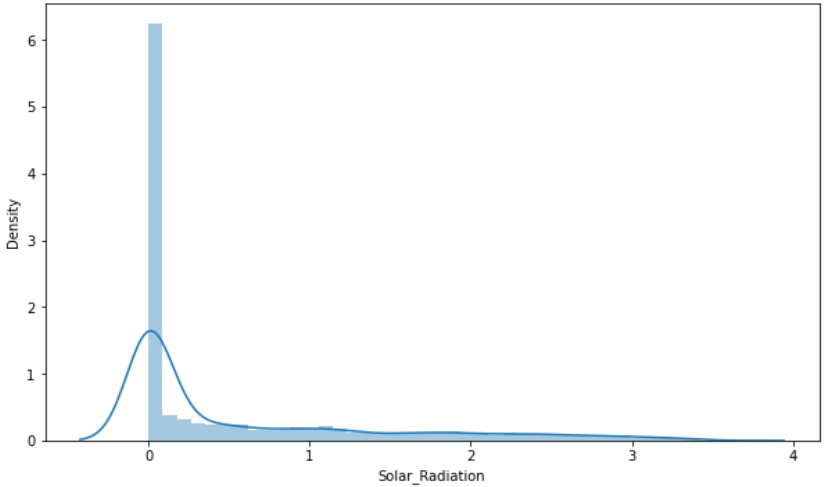


Exploratory Data Analysis

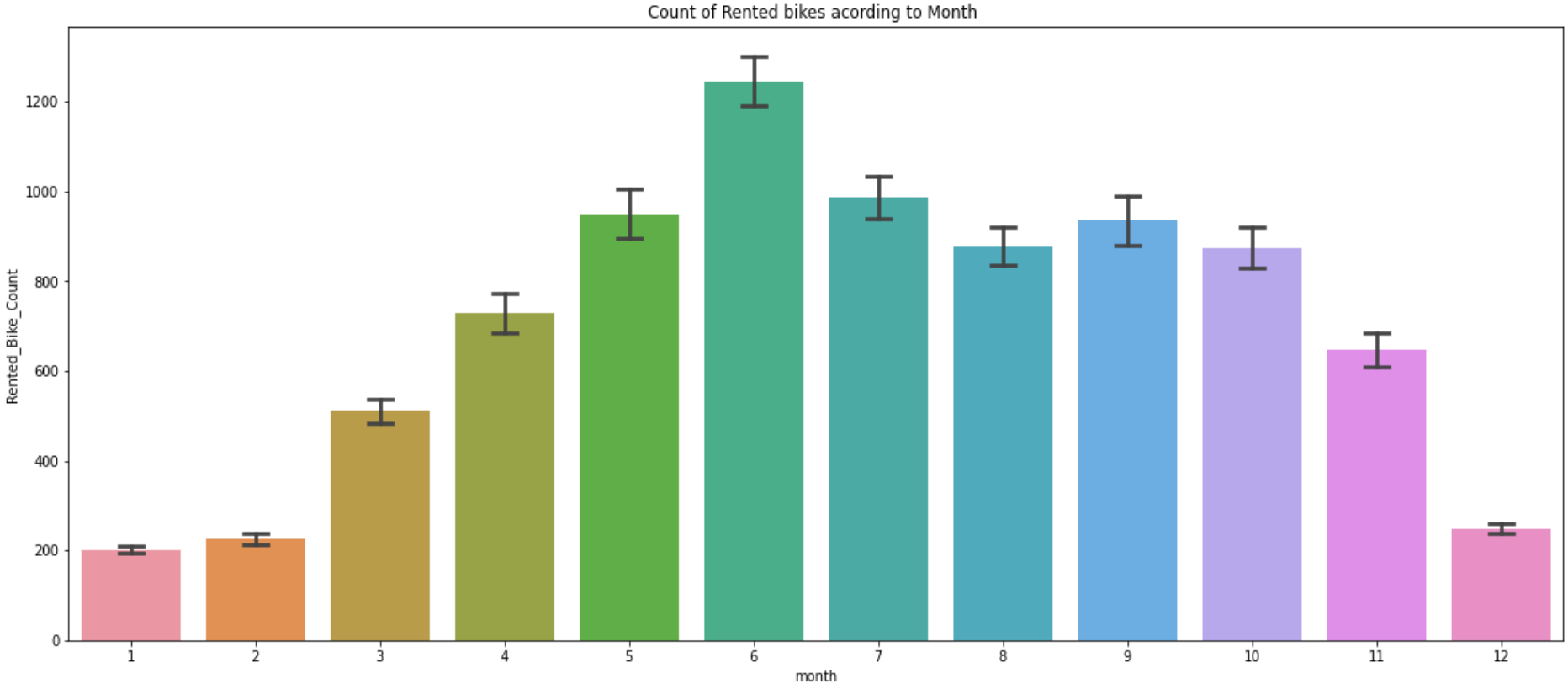




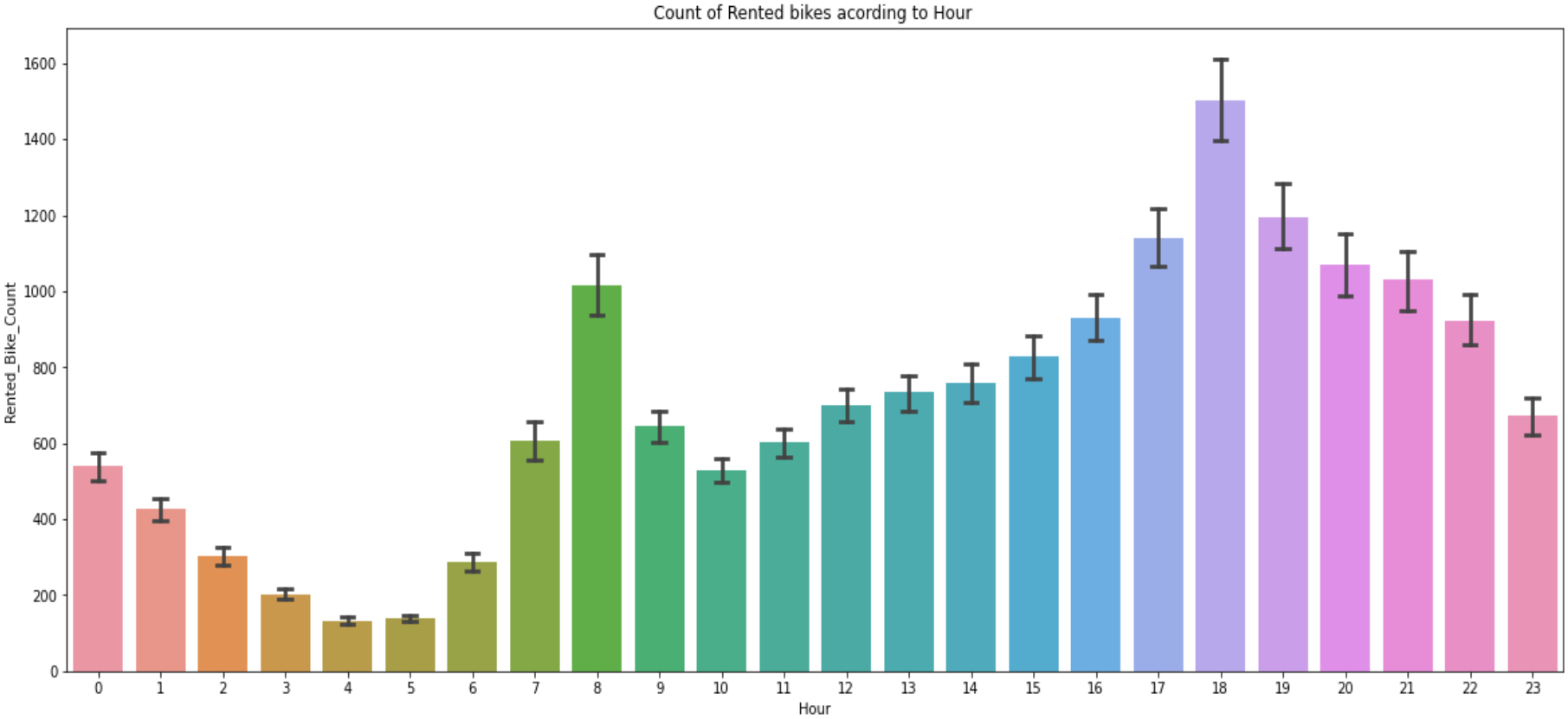




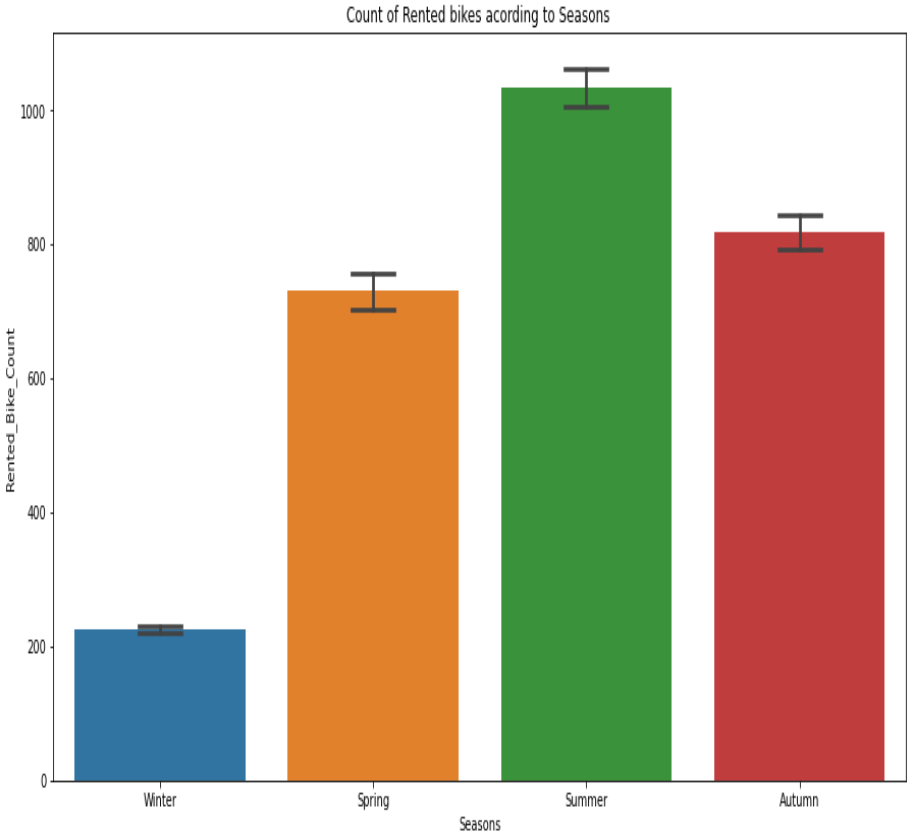
Exploratory Data Analysis

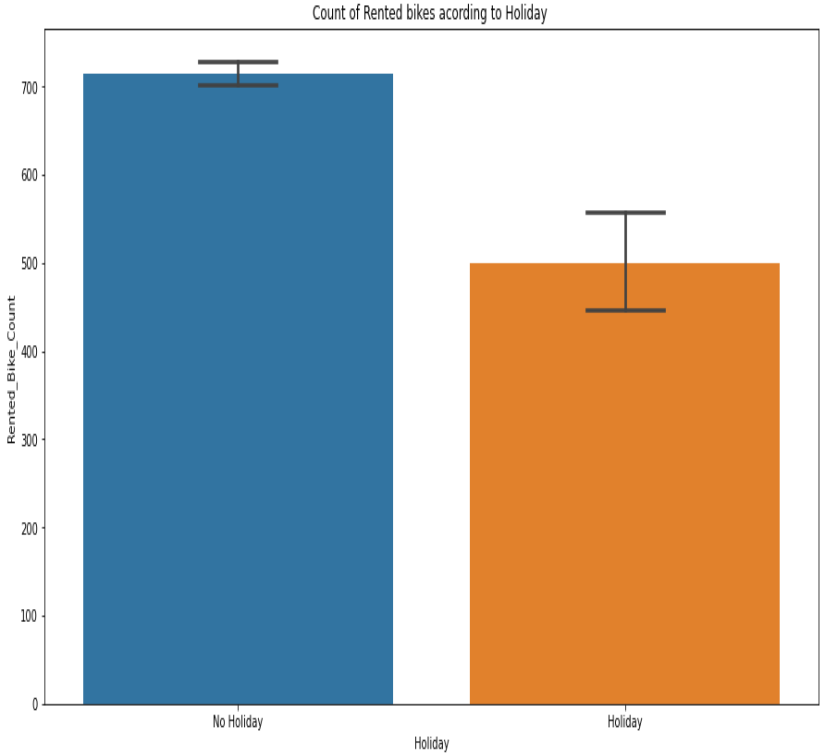


Exploratory Data Analysis

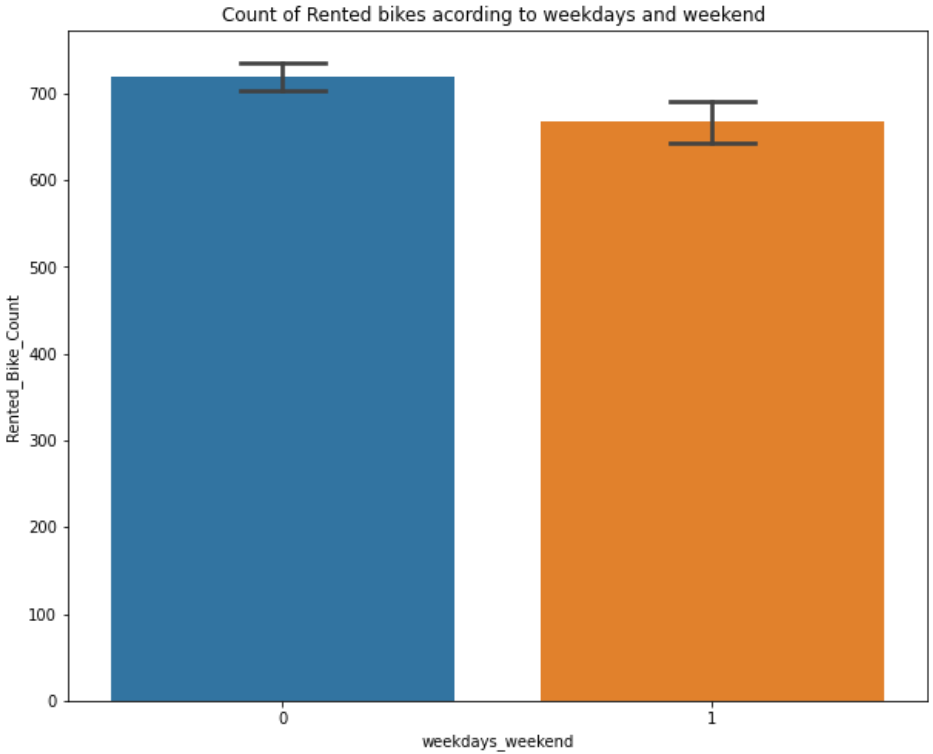


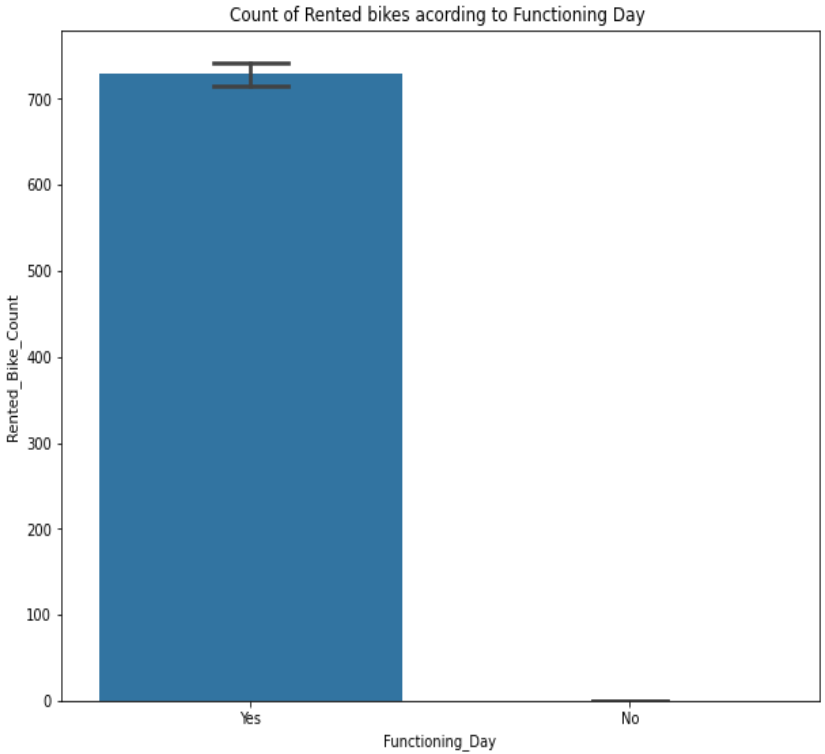
Exploratory Data Analysis



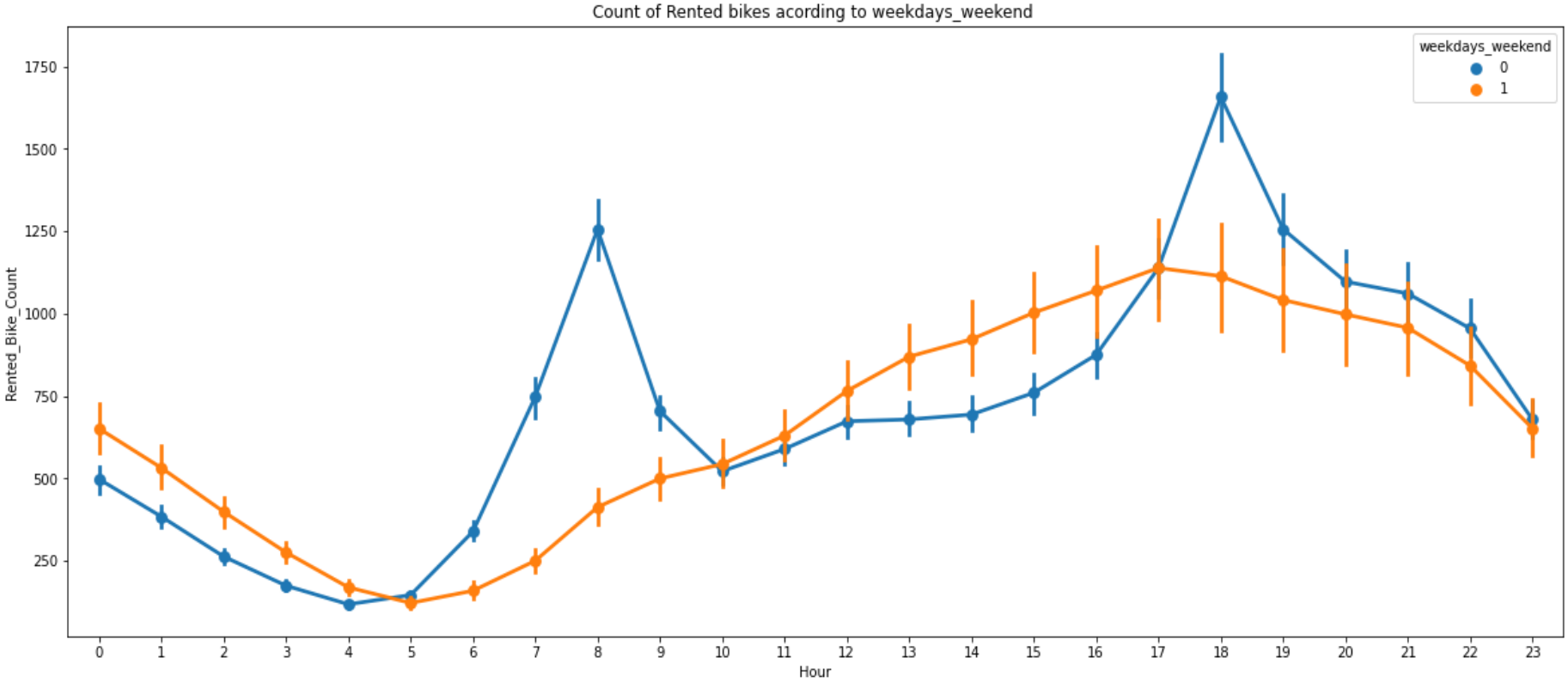


Exploratory Data Analysis

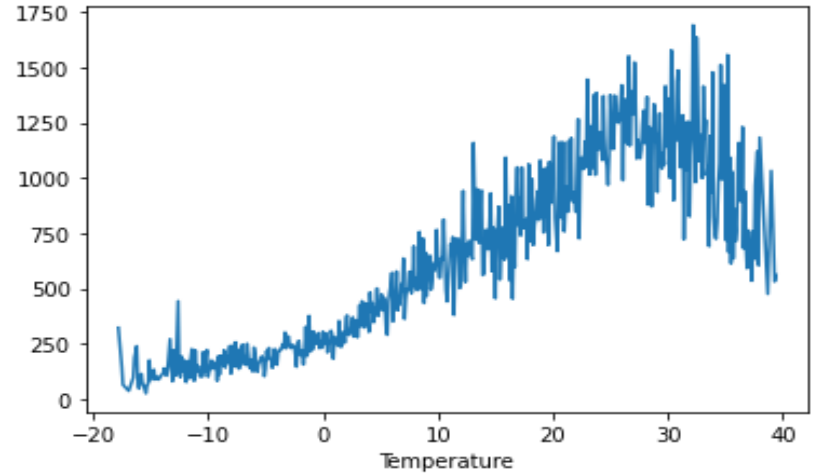


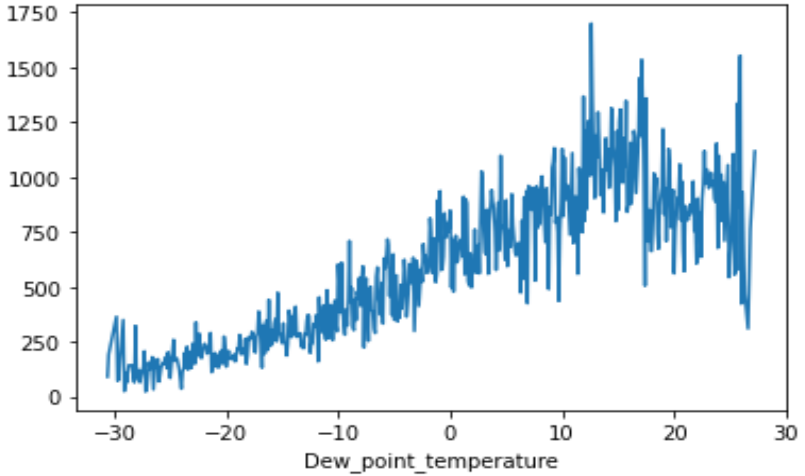


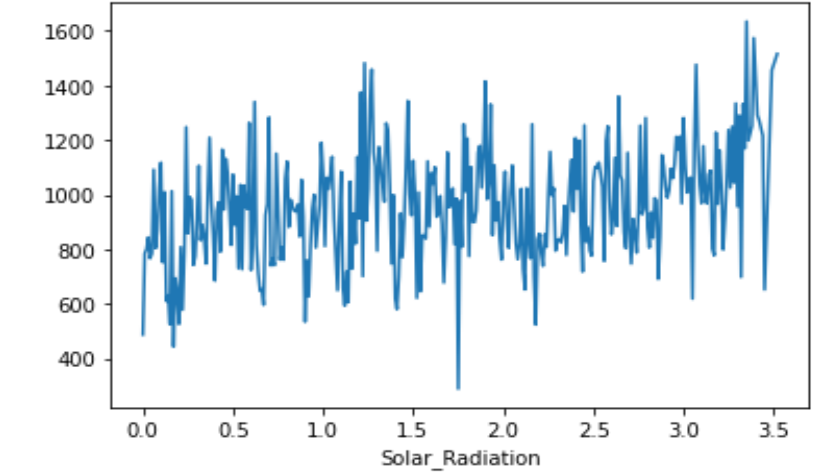
Exploratory Data Analysis

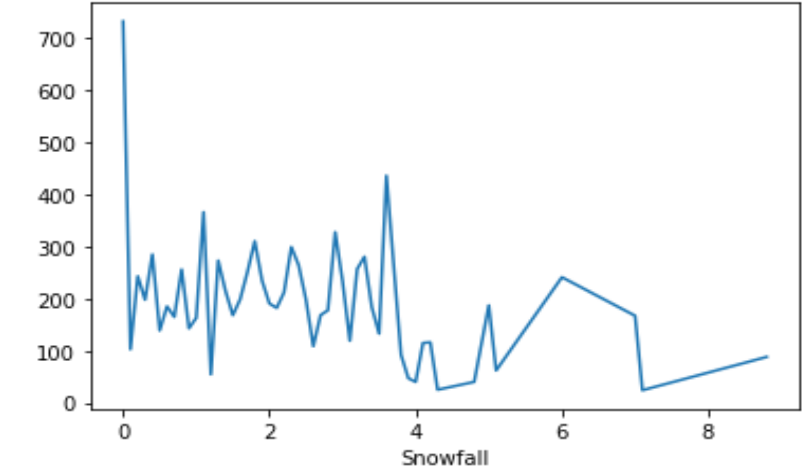


Exploratory Data Analysis

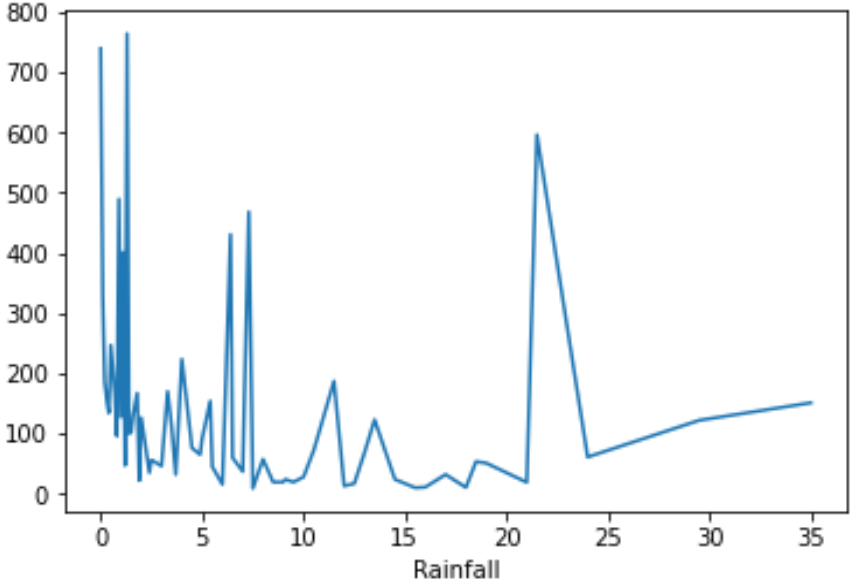


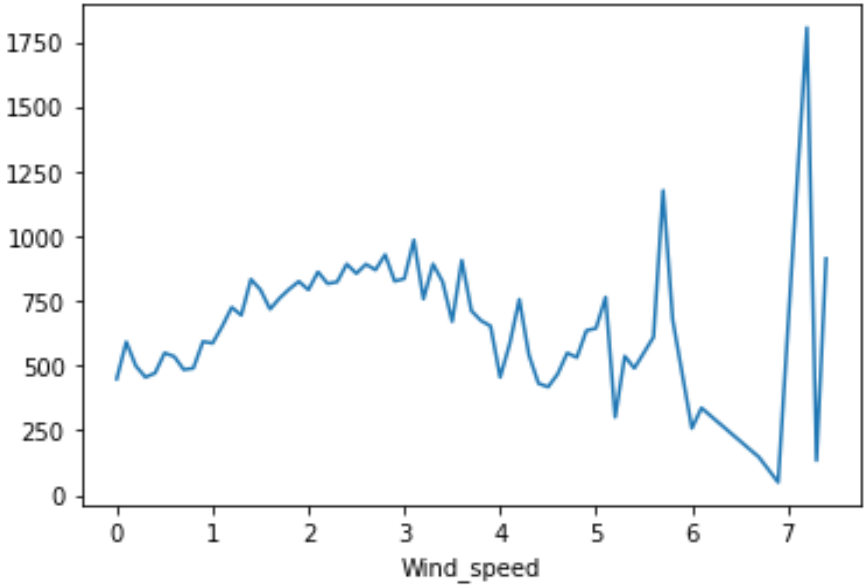






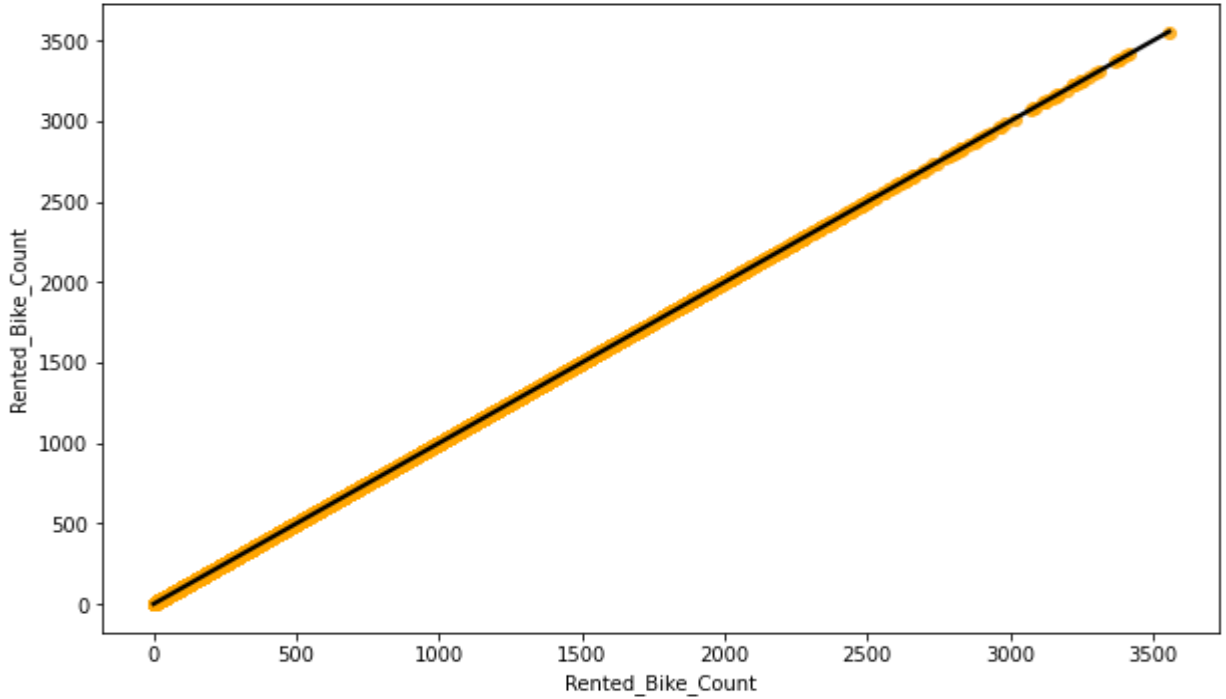
Exploratory Data Analysis

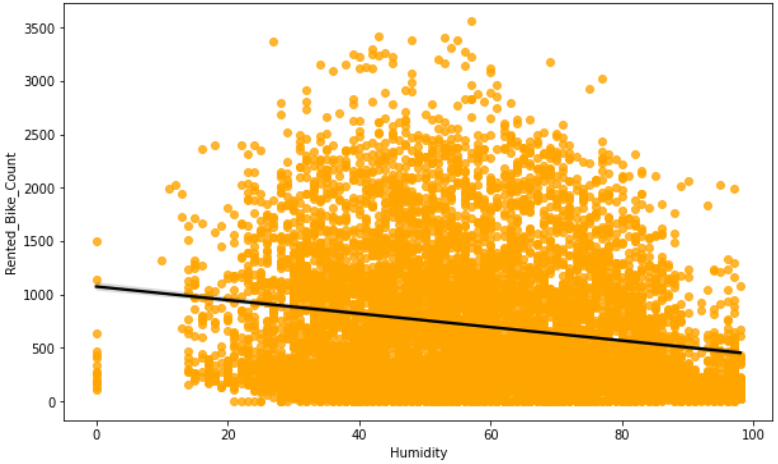


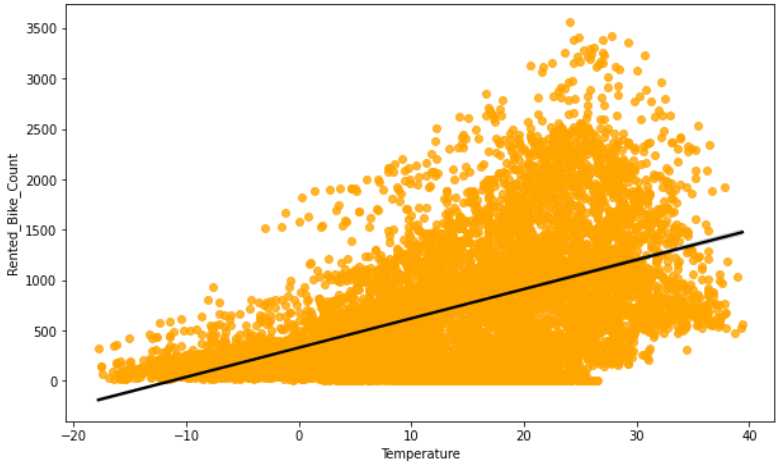


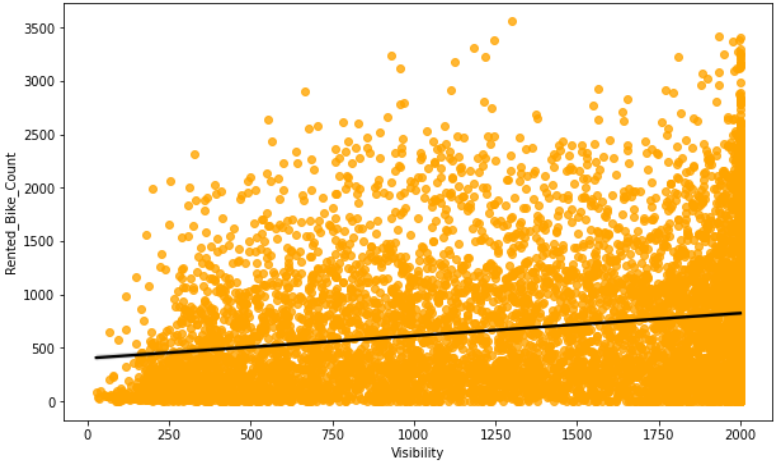
Regression Plot

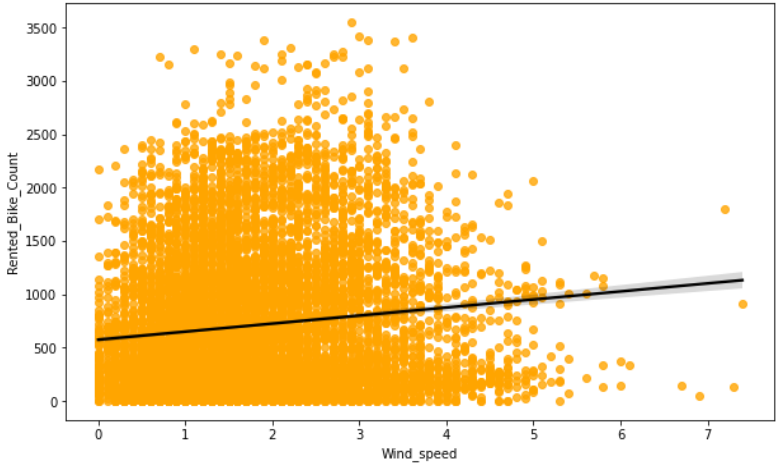
```
#printing the regression plot for all the numerical features
for col in numerical_features:
  fig,ax=plt.subplots(figsize=(10,6))
  sns.regplot(x=bike_df[col],y=bike_df['Rented_Bike_Count'],scatter_kws={"color": 'orange'}, line_kws={"color": "black"})
```

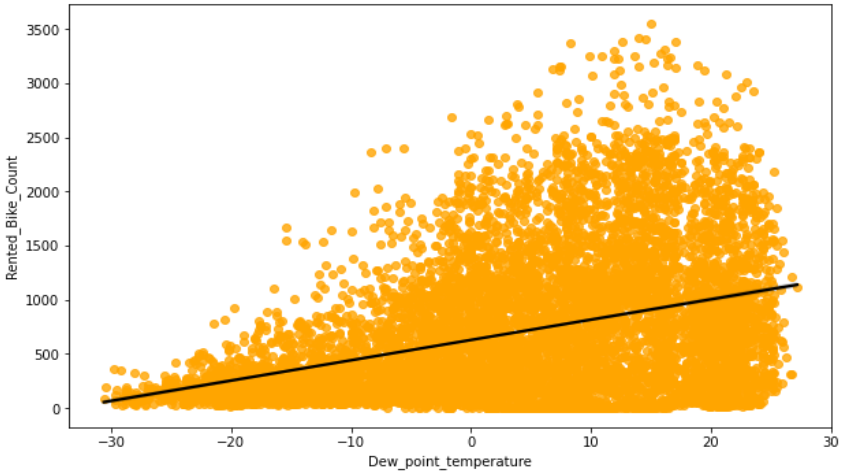


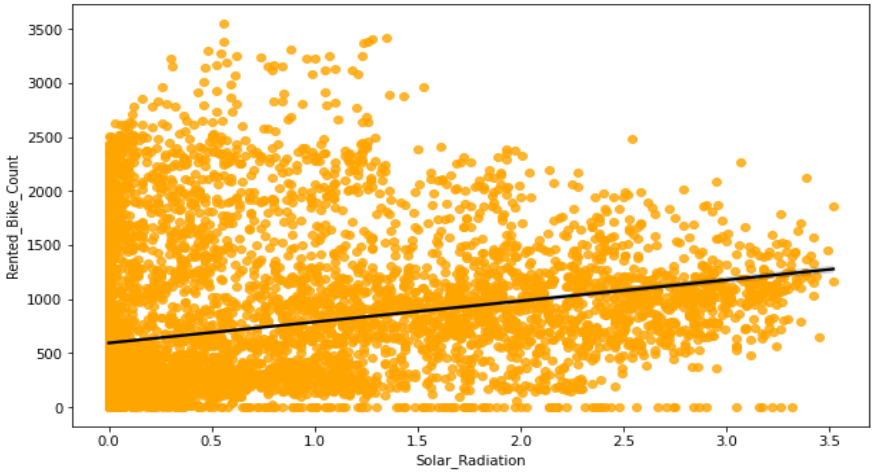

Rented_Bike_Count
Humidity
0
500
1000
1500
2000
2500
3000
3500
0
20
40
60
80
100

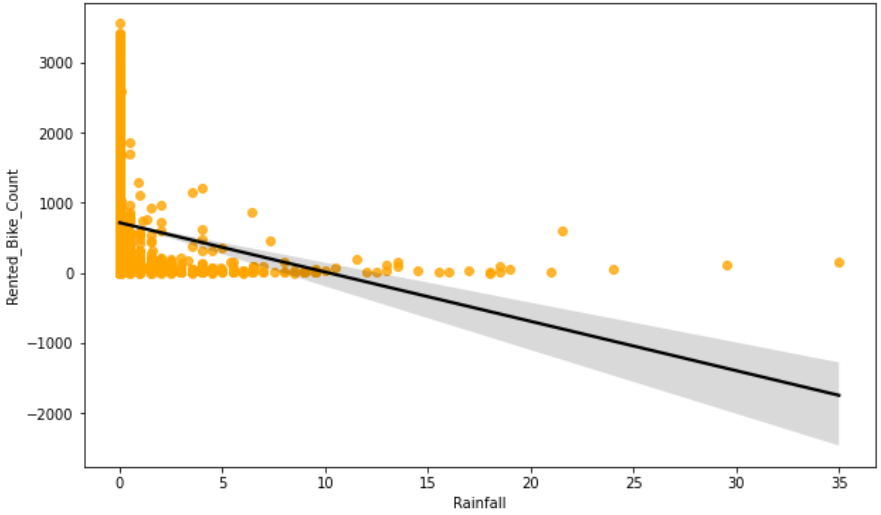

Rented_Bike_Count
Temperature
0
500
1000
1500
2000
2500
3000
3500
−20
−10
0
10
20
30
40

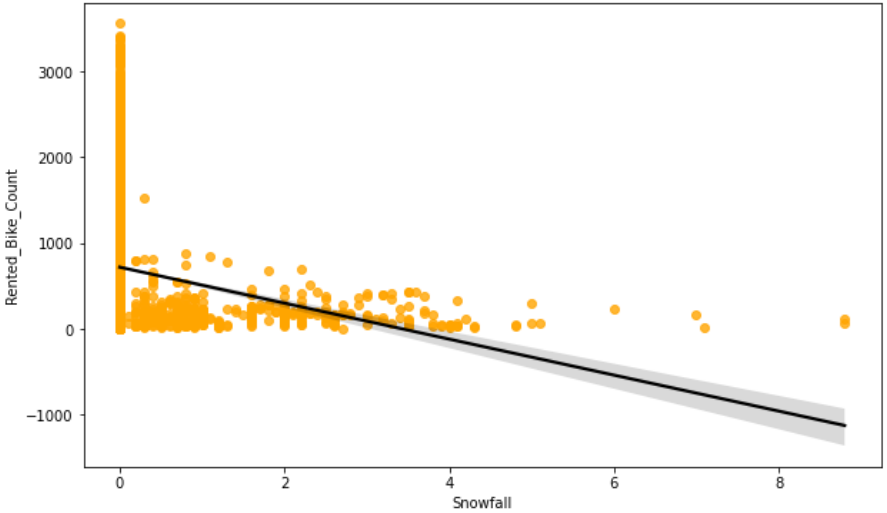

Rented_Bike_Count
Visibility
0
500
1000
1500
2000
2500
3000
3500
0
250
500
750
1000
1250
1500
1750
2000


Rented_Bike_Count
Wind_speed
0
500
1000
1500
2000
2500
3000
3500
0
1
2
3
4
5
6
7


Rented_Bike_Count
Dew_point_temperature

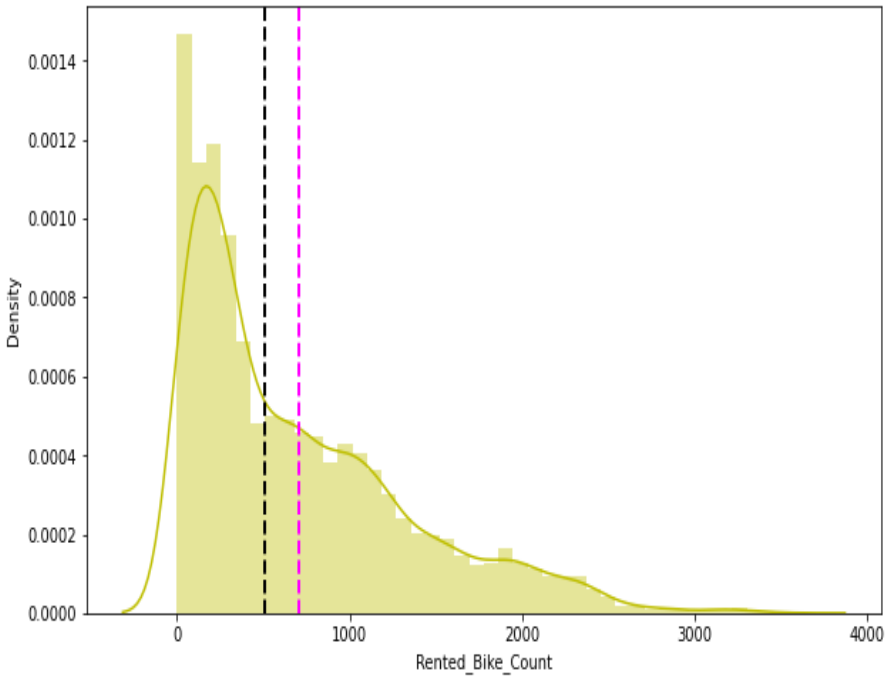

Rented_Bike_Count
Solar_Radiation


Rented_Bike_Count
Rainfall

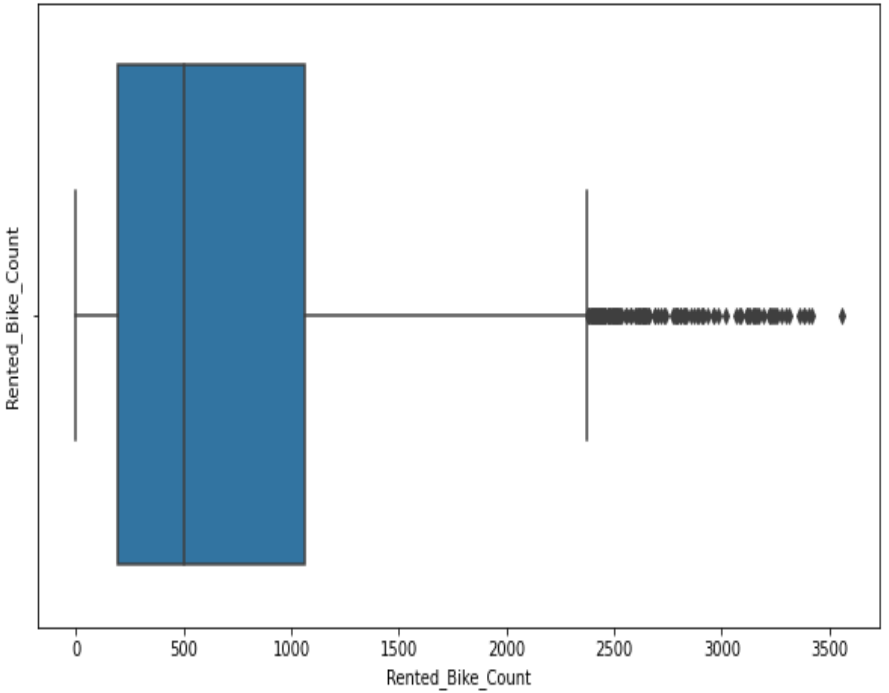

Rented_Bike_Count
Snowfall

Outliers Detection & Treatment

Normalise Rented Bikes Count

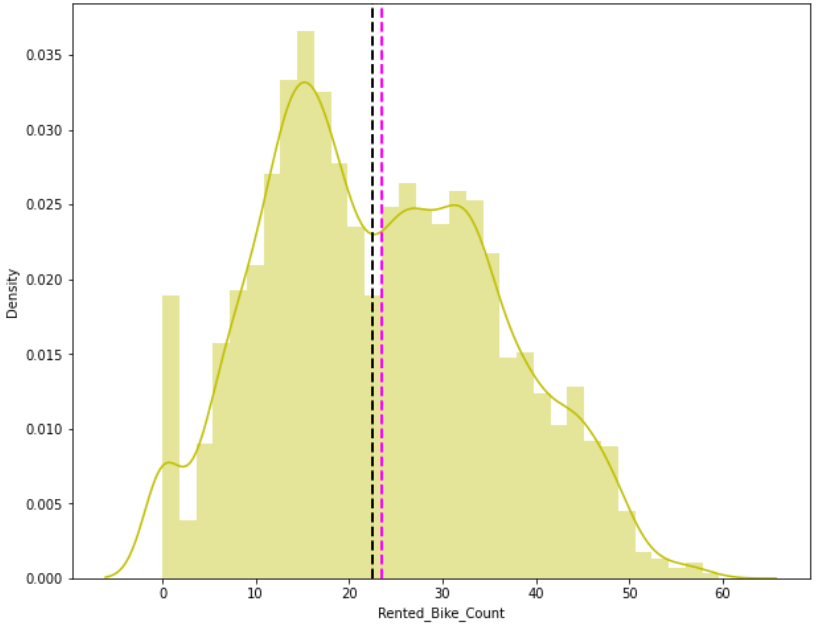


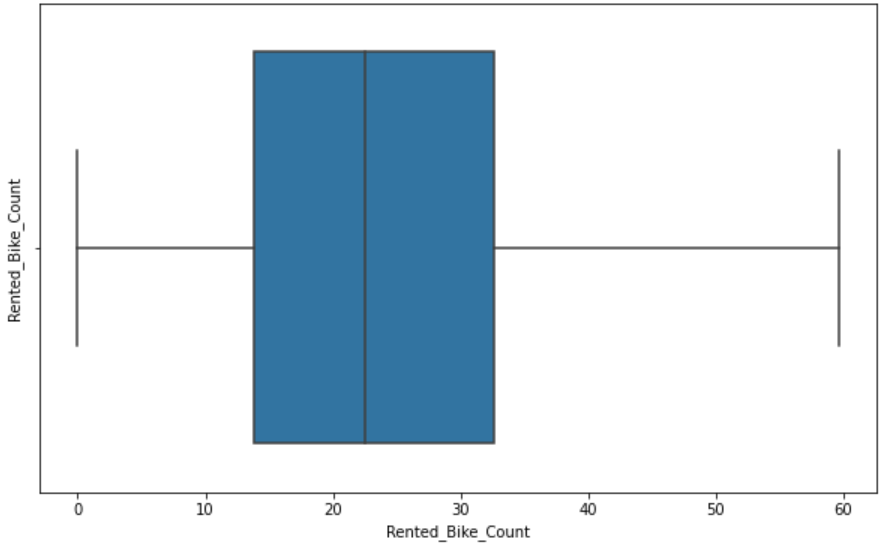
Outliers Detected



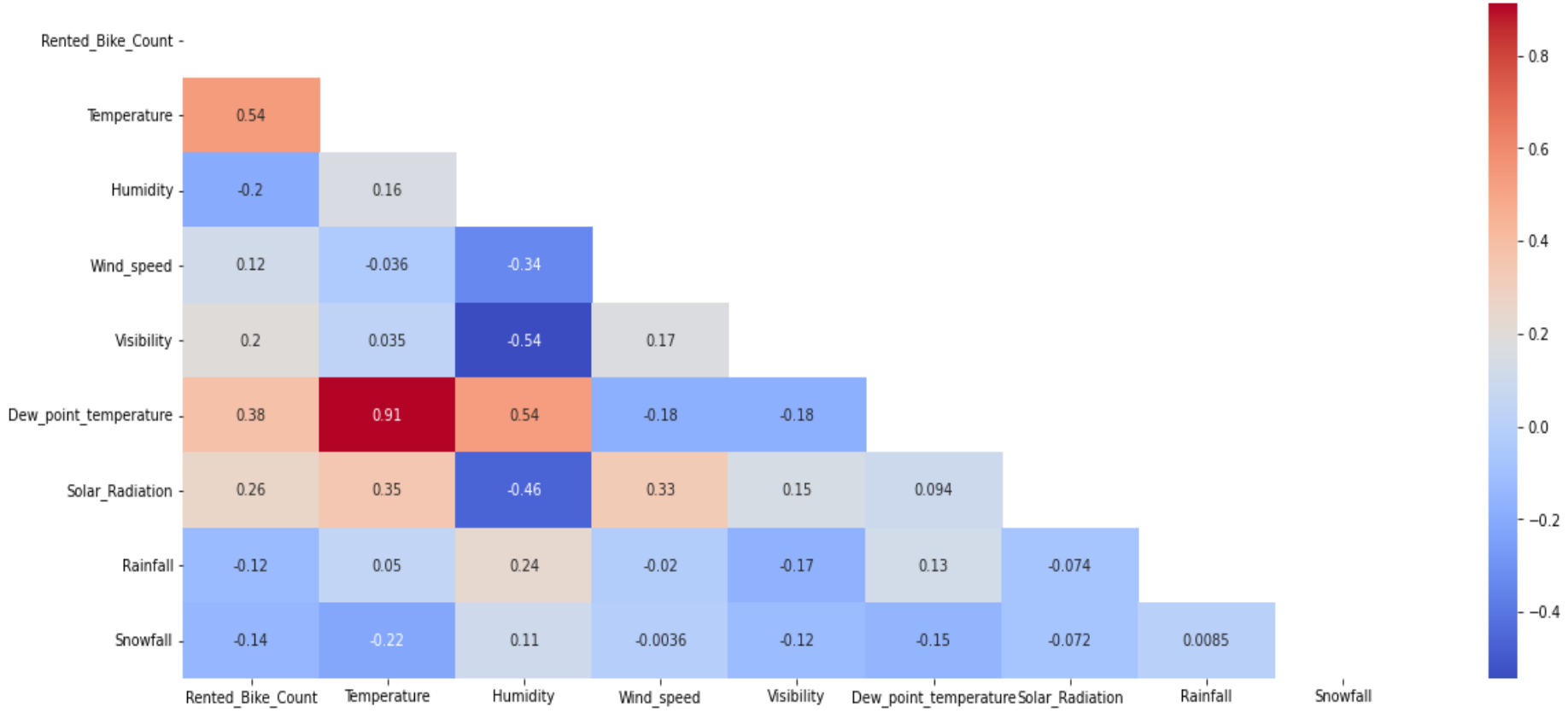
```
#Applying square root to Rented Bike Count to improve skewness
plt.figure(figsize=(10,8))
plt.xlabel('Rented Bike Count')
plt.ylabel('Density')

ax=sns.distplot(np.sqrt(bike_df['Rented_Bike_Count']), color="y")
ax.axvline(np.sqrt(bike_df['Rented_Bike_Count']).mean(), color='magenta', linestyle='dashed', linewidth=2)
ax.axvline(np.sqrt(bike_df['Rented_Bike_Count']).median(), color='black', linestyle='dashed', linewidth=2)

plt.show()
```

After applying sqrt on Rented Bike Count, the outliers were removed.





Correlation Heatmap



Machine Learning Modelling

1. Linear Regression

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.

$$y\_pred=\beta_0+\beta_1 x$$

Where β_0 and β_1 are intercept and slope respectively.
In case of multiple features the formula translates into:

$$y\_pred=\beta_0+\beta_1 x_1+\beta_2 x_2+\beta_3 x_3+.....$$

where x_1,x_2,x_3 are the features values and β_0,β_1,β_2.....
are weights assigned to each of the features. These become the parameters which the algorithm tries to learn using Gradient descent.

```
#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)
```

```
0.7722101548255267
```

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

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

```
array([ 5.11538263e-01, -1.27236196e-01, -2.90122073e-02,  9.90615715e-04,
        8.89701409e-01, -1.48171446e+00, -8.13629988e-02, -2.04211065e+00,
       -4.95822637e+00, -7.23630435e+00, -9.51882072e+00, -9.04457949e+00,
       -4.03213994e+00,  2.27462862e+00,  7.54438892e+00,  1.25491521e+00,
       -3.27047269e+00, -3.46014138e+00, -2.74135788e+00, -2.96341191e+00,
       -3.18317912e+00, -1.92136354e+00,  1.11918595e-01,  3.83216182e+00,
        1.02397844e+01,  6.78230326e+00,  6.08102846e+00,  6.20280481e+00,
        5.04597766e+00,  1.46736086e+00, -4.81648861e-01, -1.26348263e+00,
       -4.80391365e+00,  3.51130869e+00,  2.82758278e+01, -8.13449897e-01,
       -1.53946964e+00, -6.75094331e-01,  1.73291511e+00,  4.41327425e+00,
       -1.71047504e+00, -3.96628184e+00,  6.27209321e-01,  3.28833588e+00,
        2.63349995e+00,  2.07210333e+00, -1.42458875e+00])
```

Machine Learning Modelling

AI

1. Lasso & Ridge Regression

Regularized linear regression models are very similar to least squares, except that the coefficients are estimated by minimizing a slightly different objective function. we minimize the sum of RSS and a "penalty term" that penalizes coefficient size.

Ridge regression (or "L2 regularization") minimizes:

$$RSS+\lambda\sum_{j=1}^{p}\beta 2j$$

Lasso regression (or "L1 regularization") minimizes:

$$RSS+\lambda\sum_{j=1}^{p}|\beta j|$$

Where λ is a tuning parameter that seeks to balance between the fit of the model to the data and the magnitude of the model's coefficients:

A tiny λ imposes no penalty on the coefficient size, and is equivalent to a normal linear regression.

Increasing λ penalizes the coefficients and thus shrinks them towards zero.

```
# 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)
```

```
#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)
```

Machine Learning Modelling

2. Lasso & Ridge Regression

Regularized linear regression models are very similar to least squares, except that the coefficients are estimated by minimizing a slightly different objective function. we minimize the sum of RSS and a "penalty term" that penalizes coefficient size.

Ridge regression (or "L2 regularization") minimizes:

$$RSS+\lambda\sum_{j=1}^{p}\beta 2j$$

Lasso regression (or "L1 regularization") minimizes:

$$RSS+\lambda\sum_{j=1}^{p}|\beta j|$$

Where λ is a tuning parameter that seeks to balance between the fit of the model to the data and the magnitude of the model's coefficients:

A tiny λ imposes no penalty on the coefficient size, and is equivalent to a normal linear regression.

Increasing λ penalizes the coefficients and thus shrinks them towards zero.

```
# 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)
```

```
#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)
```

Machine Learning Modelling

3. Elastic New Regression

Elastic net is a penalized linear regression model that includes both the L1 and L2 penalties during training. Using the terminology from "The Elements of Statistical Learning," a hyperparameter "alpha" is provided to assign how much weight is given to each of the L1 and L2 penalties.

```
#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)
```

Machine Learning Modelling

4. Decision Tree

Decision tree is a type of supervised learning algorithm that is mostly used in classification problems. It works for both categorical and continuous input and output variables.

It is a flowchart-like structure in which each internal node represents a "test" on an attribute (e.g. whether a coin flip comes up heads or tails), each branch represents the outcome of the test, and each leaf node represents a class label (decision taken after computing all attributes).

The decision of making strategic splits heavily affects a tree's accuracy. The decision criteria is different for classification and regression trees.

Decision trees use multiple algorithms to decide to split a node in two or more sub-nodes. The creation of sub-nodes increases the homogeneity of resultant sub-nodes.

```
#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)
```

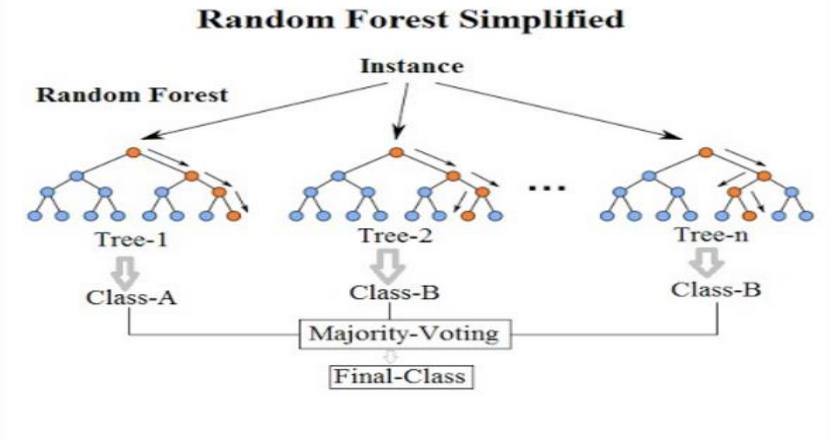
```
DecisionTreeRegressor(criterion='mse', max_depth=8, max_features=9,
                      max_leaf_nodes=100)
```

```
#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)
```

Machine Learning Modelling

5. Random Forest

Random Forest is a bagging type of Decision Tree Algorithm that creates a number of decision trees from a randomly selected subset of the training set, collects the labels from these subsets and then averages the final prediction depending on the most number of times a label has been predicted out of all.




```
#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()
```

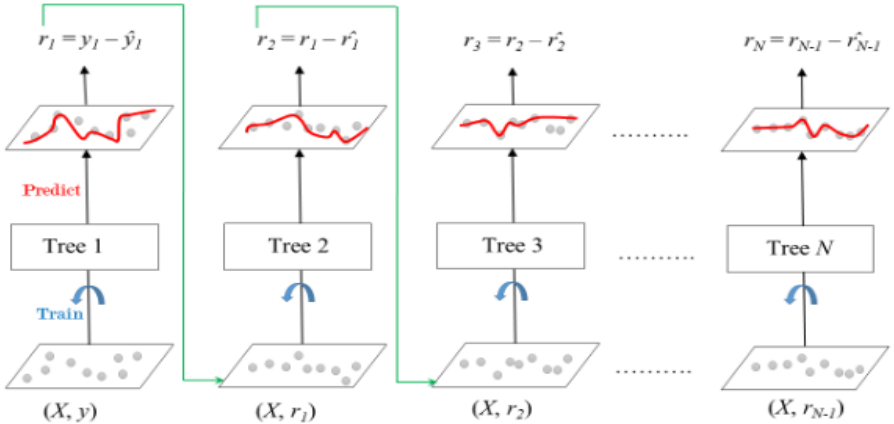
```
# Making predictions on train and test data

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

Machine Learning Modelling

6. Gardient Boosting

Gradient boosting is a machine learning technique used in regression and classification tasks, among others. It gives a prediction model in the form of an ensemble of weak prediction models, which are typically decision trees.

In gradient boosting, each predictor corrects its predecessor's error.




```
#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()
```

```
# 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)
```

Machine Learning Modelling

7. Hyperparameter Tuning

A hyperparameter is a model argument whose value is set before the learning process begins. The key to machine learning algorithms is hyperparameter tuning.

Hyperparameters are sets of information that are used to control the way of learning an algorithm.

Hyperparameters alter the way a model learns to trigger this training algorithm after parameters to generate outputs.

We used Grid Search CV, Randomized Search CV and Bayesian Optimization for hyperparameter tuning.

Grid-Search-CV 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.

```
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)
```

```
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)
```

Evaluation Metrics

- 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 **$R^2 = 1 - \text{Unexplained Variation/Total Variation}$**
- $$R^2 = 1 - \frac{SS_{RES}}{SS_{TOT}} = 1 - \frac{\sum_i (y_i - \hat{y}_i)^2}{\sum_i (y_i - \bar{y})^2}$$

- R2 =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.

Result Analysis

		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	5.404	52.832	7.269	0.657	0.65
	5	Random forest regression	0.799	1.594	1.263	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
	8	Decision tree regression	5.755	60.578	7.783	0.617	0.61
Test set	0	Linear regression	4.410	33.275	5.768	0.789	0.78
	1	Lasso regression	7.456	96.775	9.837	0.387	0.37
	2	Ridge regression	4.410	33.277	5.769	0.789	0.78
	3	Elastic net regression Test	5.874	59.451	7.710	0.624	0.62
	4	Dicision tree regression	5.755	60.578	7.783	0.617	0.61
	5	Random forest regression	2.217	12.889	3.590	0.918	0.92
	6	Gradient boosting regression	3.493	21.289	4.614	0.865	0.86
	7	Gradient Boosting gridsearchcv	2.401	12.393	3.520	0.922	0.92
	8	Decision tree regression	5.755	60.578	7.783	0.617	0.61

Conclusion

During the time of our analysis, we initially did EDA on all the features of our dataset. We first analyzed our dependent variable, 'Rented Bike Count' and also transformed it. Next we analyzed categorical variable and dropped the variable who had majority of one class, we also analyzed 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, elastic net, decision tree, Random Forest and XG-Boost. We did hyperparameter tuning to improve our model performance. The results of our evaluation are:

- No overfitting is seen.
- Random forest Regressor and Gradient Boosting grid-search-cv gives the highest R2 score of 99% and 95% respectively for Train Set and 92% for Test set.
- Feature Importance value for Random Forest and Gradient Boost are different.
- We can deploy this model.

However, this is not the ultimate end. As this data is time dependent, the values for variables like temperature, windspeed, solar radiation etc., will not always be consistent. Therefore, there will be scenarios where the model might not perform well. As Machine learning is an exponentially evolving field, we will have to be prepared for all contingencies and also keep checking our model from time to time.

References

1. Stacksoverflow.com
2. Geeks for Geeks
3. Kaggle.com
4. analyticsvidhya.com
5. Machine Learning using Python by Manaranjan Pradhan (Book)