Predicting a Stable Bike Rental Supply in Seoul Based on Weather and Holiday Information

Team 5: Harini Lakshmanan, Kyle Esteban Dalope, and John J Chen

University of San Diego

Master of Science, Applied Data Science

ADS-505 Applied Data Science for Business

Section 01

Oct 17, 2022

Predicting a Stable Bike Rental Supply in Seoul Based on Weather and Holiday

Rental bikes have become a popular transportation method that allows residents and visitors in urban cities to be mobile, minimize fuel consumption, and provide convenience for commutes. Bike sharing is a system that is popular in larger cities. It has many benefits as it is a green method of traveling and reduces traffic congestion. The system is easy to use and the individual renting a bike requires access with a mobile device to download an app that will unlock the bike; agreeing the ability to track the location of the bike while in service. This system allows the public to rent bikes from a location and then return them to a designated and convenient location on an as-needed basis. Thus, it is important to make the rental bike available and accessible to the public at the right time as it lessens the waiting time (UCI Machine Learning Repository).

The objective of this project is to apply machine learning techniques to effectively estimate the demand of rental bikes in Seoul, South Korea, based on thirteen different influencing variables including weather information, number of bikes rented per hour, and date information. Using predictive models and data mining methods to identify the demand would help in shaping the bike rental business and prepare for increase in supply, and if not met, could lead to an increase in wait times for customers. The proposed project would provide a better understanding of downtime periods, which can be attributed towards servicing the rental bikes for maintenance and safety precautions. The data that was used in this project was collected and sourced from the UCI Machine Learning Repository and directly from http://data.seoul.go.kr/. The dataset contains 8760 instances, 14 attributes, and with a data source of March 01, 2020.

EDA

To begin the project, the dataset titled "SeoulBikeData.csv", was first imported into a dataframe to view the data and to initiate exploratory data analysis (EDA). This will allow the ability to learn and identify whether or not relationships between the response variable (Rented_Bike_Count) and the predictor variables are present. The dataset contains 14 attributes with exactly 8,760 instances with no null values present. Figure 1 summarizes all the attributes within the dataset with a brief description for each as well.

Figure 1

Table Goes over the Parameters and Features

Table 1. Seoul Bike data variables and description.

Parameters/Features	Abbreviation	Type	Measurement
Date	Date	Year-month-day	2017-Dec-2017 to 2018-Dec-2018
Rented Bicycle count	Count	Continuous	0,1,2,3 3556
Hour	Hour	Continuous	0,1,2,3 23
Temperature	Temp	Continuous	°C
Humidity	Hum	Continuous	%
Windspeed	Wind	Continuous	m/s
Visibility	Visb	Continuous	10 m
Dew point temperature	Dew	Continuous	°C
Solar radiation	Solar	Continuous	MJ/m2
Rainfall	Rain	Continuous	Mm
Snowfall	Snow	Continuous	Cm
Holiday	Holiday	Categorical	Holiday, Workday
Functional Day	Fday	Categorical	NoFunc, Func
Week status	Wstatus	Categorical	Weekday (Wday), Weekend (Wend)
Day of the week	Dweek	Categorical	Sunday, Monday . . . . Saturday
Seasons	Season	Categorical	Spring, Summer, Autumn, Winter

(E, Sathishkumar & Cho, Yongyun, 2020)

Before starting EDA, we converted the 'Rented Bike Count' column to float data type and 'Hour' to object for later graphing purposes. The describe function was used to get descriptive Statistics that are seen in Figure 2.

Figure 2

Descriptive Statistics that Summarize the Central Tendency, Dispersion, and Shape of a Dataset's Distribution

	count	mean	std	min	25%	50%	75%	max
Rented_Bike_Count	8760.000000	704.602055	644.997468	0.000000	191.000000	504.500000	1065.250000	3556.000000
Temperature(°C)	8760.000000	12.882922	11.944825	-17.800000	3.500000	13.700000	22.500000	39.400000
Humidity(%)	8760.000000	58.226256	20.362413	0.000000	42.000000	57.000000	74.000000	98.000000
Wind_speed(m/s)	8760.000000	1.724909	1.036300	0.000000	0.900000	1.500000	2.300000	7.400000
Visibility(10m)	8760.000000	1436.825799	608.298712	27.000000	940.000000	1698.000000	2000.000000	2000.000000
Dew_point_temperature(°C)	8760.000000	4.073813	13.060369	-30.600000	-4.700000	5.100000	14.800000	27.200000
Solar_Radiation(MJ/m2)	8760.000000	0.569111	0.868746	0.000000	0.000000	0.010000	0.930000	3.520000
Rainfall(mm)	8760.000000	0.148687	1.128193	0.000000	0.000000	0.000000	0.000000	35.000000
Snowfall(cm)	8760.000000	0.075068	0.436746	0.000000	0.000000	0.000000	0.000000	8.800000
Holiday_No Holiday	8760.000000	0.950685	0.216537	0.000000	1.000000	1.000000	1.000000	1.000000
Seasons_Spring	8760.000000	0.252055	0.434217	0.000000	0.000000	0.000000	1.000000	1.000000
Seasons_Summer	8760.000000	0.252055	0.434217	0.000000	0.000000	0.000000	1.000000	1.000000
Seasons_Winter	8760.000000	0.246575	0.431042	0.000000	0.000000	0.000000	0.000000	1.000000
Functioning_Day_Yes	8760.000000	0.966324	0.180404	0.000000	1.000000	1.000000	1.000000	1.000000

Kicking off the EDA visualizations, we created bar graphs for some of the categorical features including holiday, functioning_day, and seasons. The holiday bar graph turned out to be what was expected with the majority of the column being non-holidays and only a few data points being holidays. Functioning_day predictor showed similar results with most being yes and only a few no's. A functioning day being yes is when the day is neither a weekend nor a holiday. Both columns showed highly unbalanced data, which makes sense since there's only a few holidays throughout the year and majority of work days or functional days. Next, we checked the value counts for seasons in Figure 3 and saw an almost identical number of values (about 2000) for each season with summer and spring slightly edging out the other two.

Figure 3

Count of Each Season

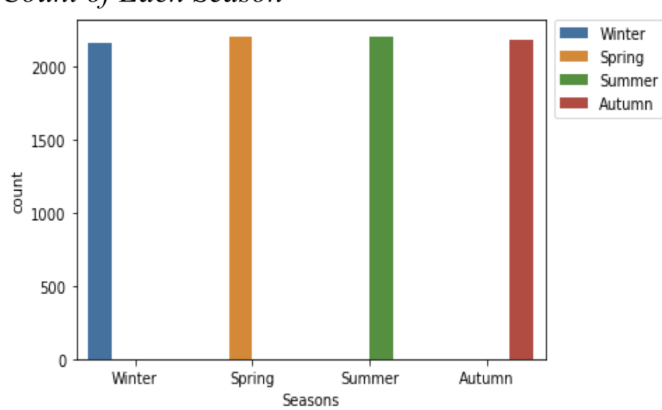
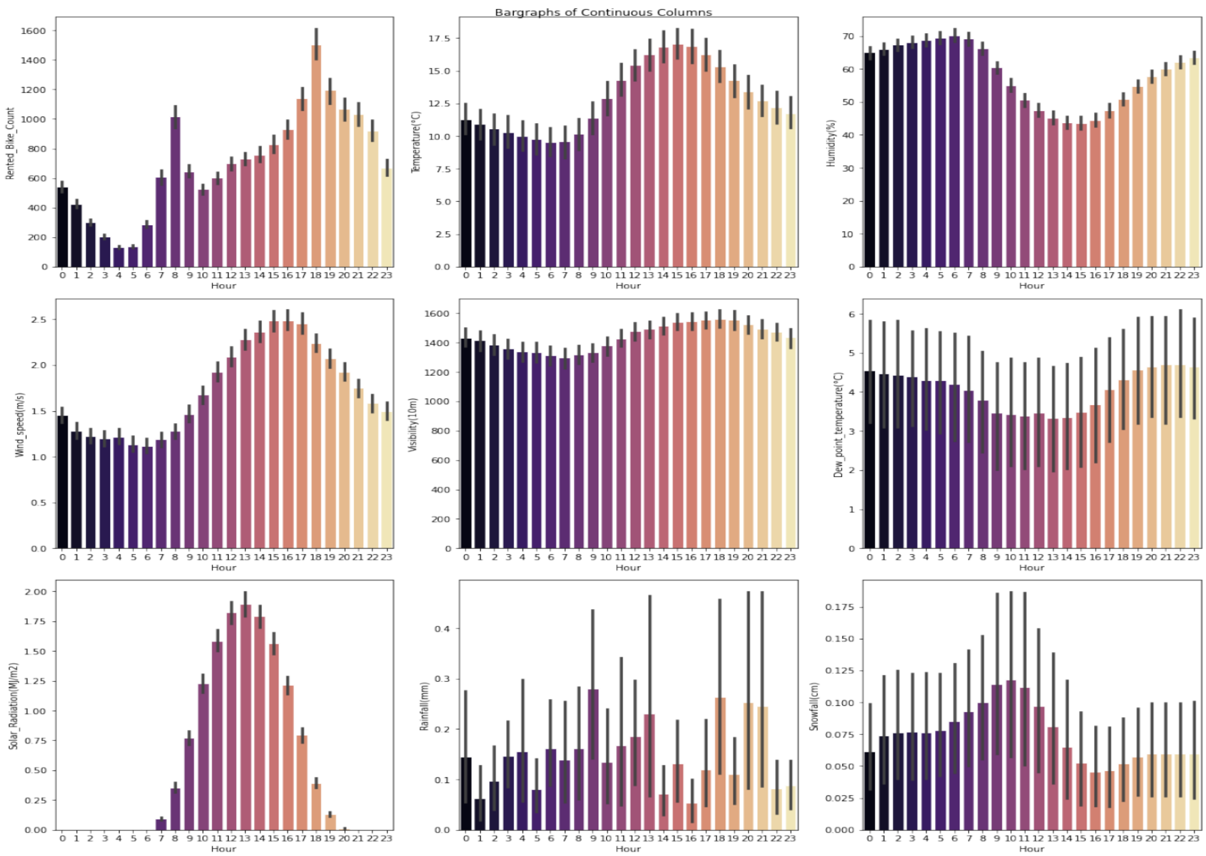


Figure 4

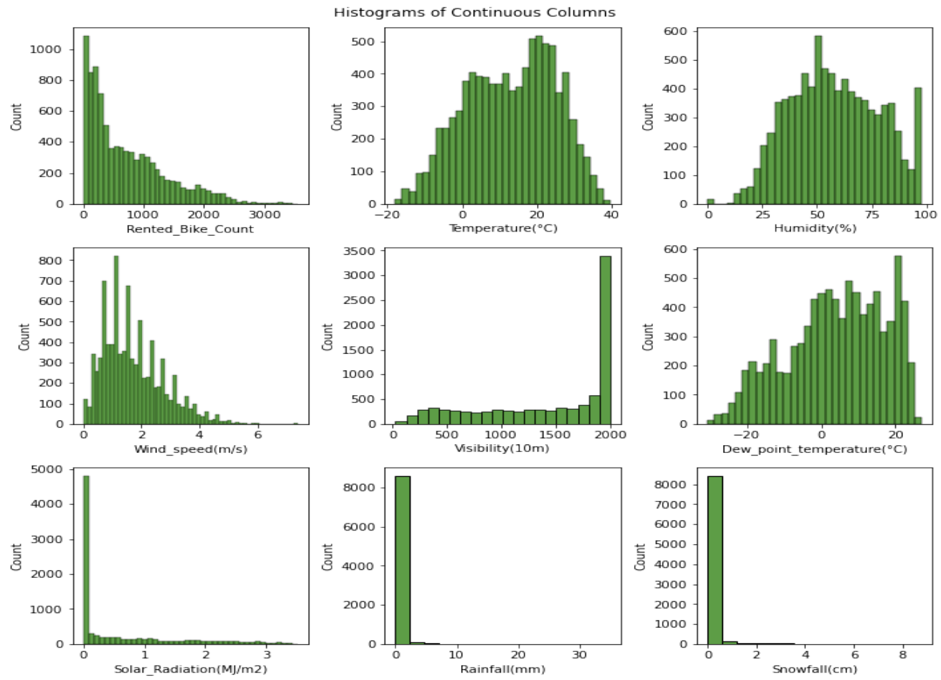
Bar Graph Showing Continuous Variables by the Hour



From there we created bar graphs in Figure 4 with hour as the x-axis variable and plotted the rest of the continuous variables against hours with most of them being the conditions of the weather when the bikes were rented. Observations we noticed were bike rental counts were frequent in the beginning of the day at 8 am and then picked up from around 5 pm and peaking at 6 pm with almost 1500 bikes rented in total from the dataset. Temperature was around the highest during the peak bike rentals and looked to have an inverse relationship with humidity. The graph of wind speed was identical to temperature rising in the morning and starting to drop off at 5 or 6 pm. Visibility was fairly stable throughout the day. The most inconsistent was rainfall. Figure 5 shows the histograms we built to get the counts of the data points from each continuous predictor column. The temperatures and humidity looked like normal distributions and the rest of the histograms were either skewed to the right or left.

Figure 5

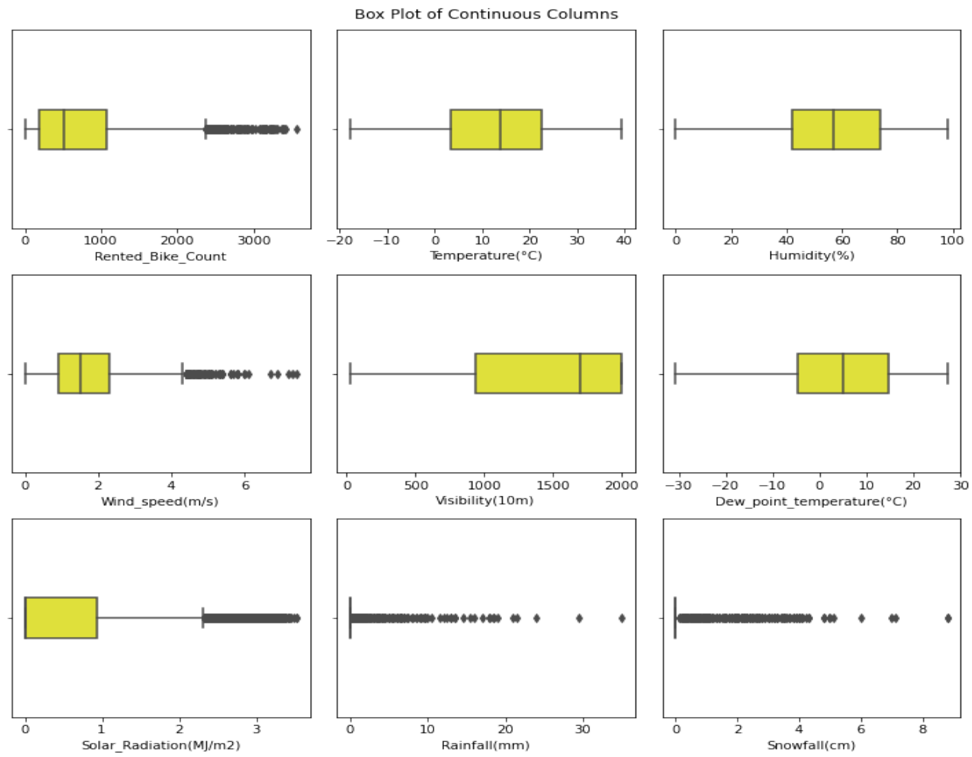
Histogram of Our Continuous Predictor Variables



Box plots were created for continuous columns in figure 6. These plots are used to look for outliers.

Figure 6

Boxplots of the Continuous Predictor Variables

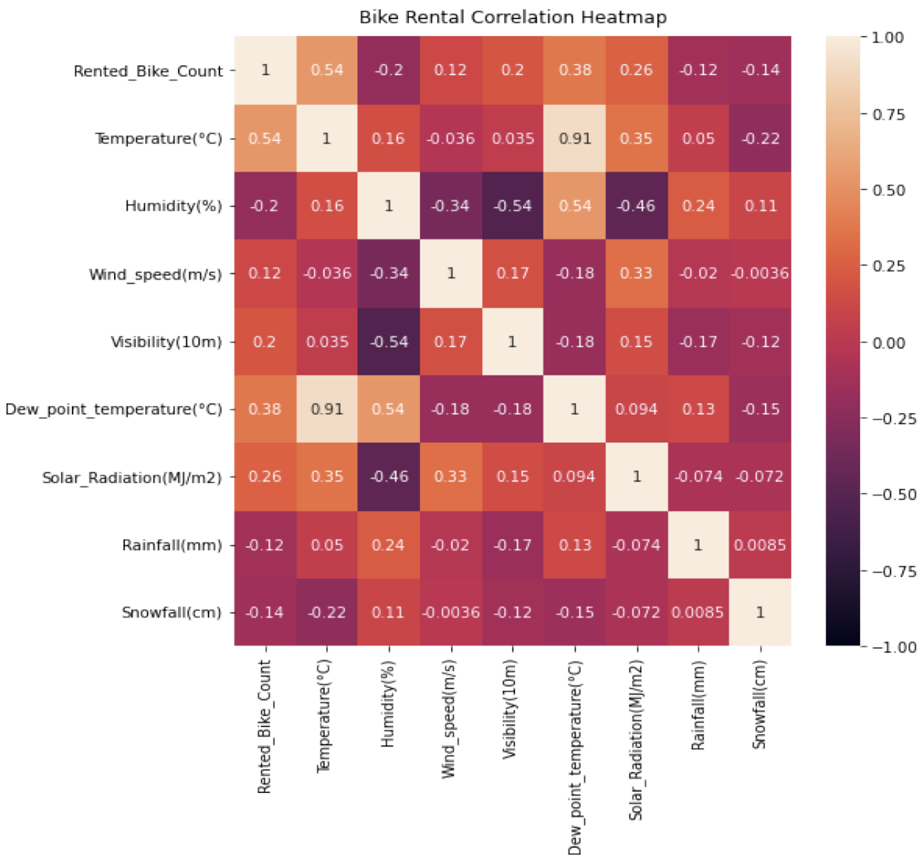


Data Pre-Processing and Splitting

Lucky for us there were no missing data in the entire dataset. From the boxplots, we were able to clearly see outliers in the 'Rented_Bike_Count', 'Wind_speed(m/s)', 'Solar_Radiation(MJ/m2)', 'Rainfall(mm)', and 'Snowfall(cm)' columns. An interesting outlier was a bike rental count of close to 3600 in a single hour. Because there were a relatively small number of outliers, we decided to leave them as-is.

Figure 7

Correlation Heatmap between Rented_Bike_Count and the Continuous Variables



In Figure 7, we used a correlation heatmap to find out the correlation between the rented bike count and all the other variables. Temperature, Dew_point_temperature, Solar_Radiation, Visibility, and Wind_speed all had positive correlation and Rainfall, Snowfall, Humidity were negatively correlated. The correlations were not very high with any of the variables with the exception of Temperature which had a moderate .5385 positive correlation with rented bike count. Our categorical variables: ‘Holiday’, ‘Seasons’, and ‘Functioning_Day’ were converted to dummy variables. For picking which variables to use we used Variance Inflation Factor on our non-categorical variables. We initially ran the VIF function on the Temperature, Dew_point_temperature , Visibility, Humidity, Wind_speed, Solar_Radiation, Snowfall, and Rainfall columns. A few of the variables returned high VIF values from Temperature to Humidity. We decided to drop Dew_point_temperature since it was a redundant temperature variable and not the popular one where most people would understand. It did the trick because after dropping it, all the remaining variables had VIF values of under 5.

Our data was split into 25% and 75% training using the train_test_split function from sklearn. After the split, we scaled the training and testing data using standardscaler function and are now ready for the modeling stage.

Model Strategies and Tuning

The ultimate goal of this project is to estimate the demand for rental bikes in Seoul, South Korea based on thirteen different influencing variables. The variable that helps us in understanding the demand is the ‘Rented Bike Count’ in this dataset. The objective is to predict the rented bike count as closely as possible and hence we need to perform a predictive analysis and not a classification problem as this data is not categorical but numerical in nature.

In this study, we conduct an empirical analysis of the performance of six popular data mining methods for predictive analysis namely, Linear regression, Decision Tree, Support vector machine, Neural Network, kNN algorithm and Random Forest for the prediction of rented bike cound, with the focus being to select the algorithm with the best fit and refine the model and deploy it.

As discussed in data preprocessing and splitting, the data was split in the ratio of 75% and 25% for train and test data. Standard scaler function was used in python to normalize the data for the seven different chosen independent variables after as discussed earlier namely, 'Temperature (°C)', 'Humidity (%)', 'Wind_speed (m/s)', 'Visibility (10m)', 'Solar_Radiation (MJ/m2)', 'Rainfall (mm)', 'Snowfall (cm)'.

Validation of the model and tuning

For the model development the train data from the earlier split was used and to evaluate the performance of the model, the test data was used. We performed prediction analysis using sklearn libraries for all the six mentioned data mining algorithms and estimated the R-squared value, mean absolute error, mean squared error and root mean squared error for the test data by comparing with the actual test data results and the predicted test data results. We chose to use R-squared value as an important performance metric to evaluate the model performance. An R-Squared above 0.7 would generally be seen as showing a high level of correlation, whereas a measure below 0.4 would show a low correlation. This is not a hard rule, however, and will depend on the specific analysis. The results of the model performance can be seen in Figure 8.

Figure 8

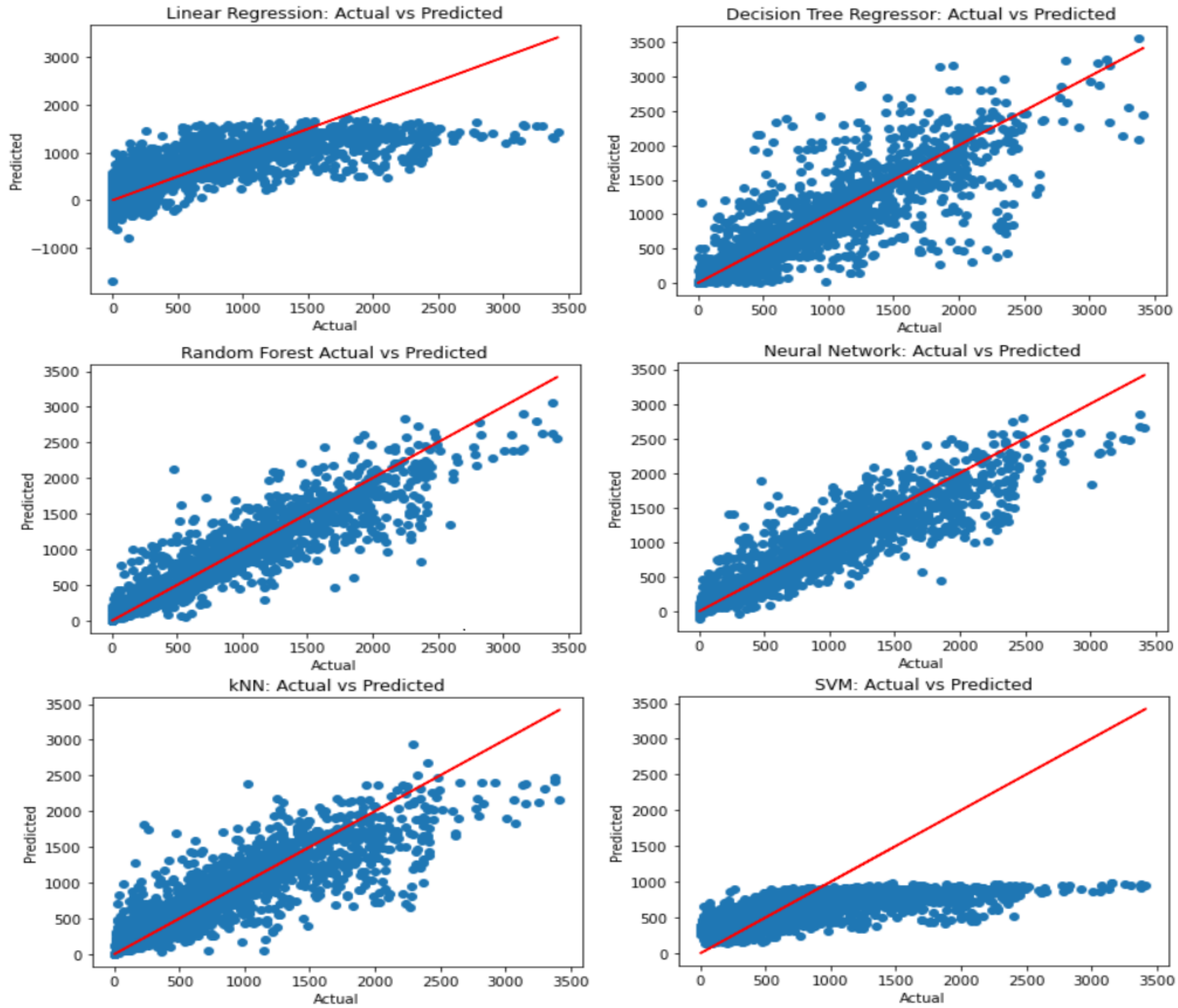
Model performance characteristics sorted by R-squared value

Model	R-Squared	MAE	MSE	RMSE
SVM	0.3373	353.3543	281388.0445	530.4602
Linear Regerssion	0.5397	326.7375	195443.5772	442.09
Decision Tree	0.7192	198.4635	119256.1356	345.3348
KNN	0.7747	201.818	95671.1062	309.3075
Neural Network	0.8548	158.2314	61641.2817	248.2766
Random Forest	0.8642	147.5391	57655.2537	240.1151

The model with the best R-squared value is random forest with a value of 0.8642 and the next closest to best model is that of a neural network model for the prediction of Rented bike count. The actual vs predicted scatter plot. To visually demonstrate how R-squared values represent the scatter around the regression line, you can plot the fitted values by observed values and can be seen in Figure 9.

Figure 9

Actual vs Predicted data for six data mining model



As we can see from Figure 8 and 9, Random Forest algorithm turned out to be the best fit for this predictive analysis for 'Rented Bike count' prediction with a low R-squared value and the least RMSE as well. Hence, we have chosen this model for our final deployment.

Conclusion and Final Model Selection

After conducting the model evaluation with the selected performance metric of R-squared and RMSE, the final model selected was the Random Forest. The model outperformed with a R-squared value of 0.8642 and a RMSE value of 240.11. To minimize the possibility of overfitting our model(s) as the data set only contained a year's worth of data, the random forest algorithm was beneficial as an ensemble technique in this manner. It must be noted, that model tuning was limited and held at a minimal due to time and computational constraints.

In conclusion, the objective for this project was to accurately predict the bike rental count (target variable) at each hour based on weather and date information (predictor variables). This concern has become a prominent topic for urban cities; such as Seoul, South Korea and other cities similar. As bike rentals have become a popular transportation method for leisure and business, it is important that the responsible entities for public transportation maintain a stable supply to minimize wait times and scheduled maintenance for accessibility. This initial project provides a strong starting point to identify key relationships between the weather and date information, periods of high volumes, demand and supply, and the utilization of data mining techniques for other alternative transportation methods for urban cities. The next steps recommended include increasing the length in time for data collection, detailed model tuning of the algorithms, and collaboration with the public transportation committee for the identified relationships to implement the random forest algorithm to predict bike rental supply.

References

UCI Machine Learning Repository: Seoul Bike Sharing Demand Data Set. (n.d.). Archive.ics.uci.edu.https://archive.ics.uci.edu/ml/datasets/Seoul+Bike+Sharing+Demand#

Long, A. (2020, February 2). Machine Learning with Datetime Feature Engineering: Predicting Healthcare Appointment No-Shows. Medium. https://towardsdatascience.com/machine-learning-with-datetime-feature-engineering-predicting-healthcare-appointment-no-shows-5e4ca3a85f96

Benton, J. (2020, July 22). Interpreting Coefficients in Linear and Logistic Regression. Medium. https://towardsdatascience.com/interpreting-coefficients-in-linear-and-logistic-regression-6ddf1295f6f1

E, Sathishkumar & Cho, Yongyun. (2020). A rule-based model for Seoul Bike sharing demand prediction using weather data. European Journal of Remote Sensing. 53. 1-18. 10.1080/22797254.2020.1725789.

Appendix

GitHub Repository link: https://github.com/jjchen-SEA/ADS-505-Seoul_Bike_Share

GitHub link to final code:

https://github.com/jjchen-SEA/ADS-505-Seoul_Bike_Share/blob/main/Group%205%20-%20Bike%20Rental%20Project%20ADS%20505.pdf

Group 5 - Bike Rental Project ADS 505

October 17, 2022

[867]:
```
# Import dependences

import pandas as pd
import numpy as np
import matplotlib as mpl
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
import statsmodels.api as sm
from statsmodels.stats.outliers_influence import variance_inflation_factor
from sklearn import preprocessing
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.model_selection import train_test_split, KFold, cross_val_score
import sklearn.metrics as metrics
from sklearn.metrics import accuracy_score, confusion_matrix, precision_score,␣
↪recall_score, f1_score
from sklearn.neural_network import MLPClassifier, MLPRegressor
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeClassifier, DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.neighbors import NearestNeighbors, KNeighborsClassifier,␣
↪KNeighborsRegressor
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression, LogisticRegressionCV
from sklearn.svm import SVR
import math
import operator
from prettytable import PrettyTable
warnings.filterwarnings("ignore")

# mpl.rc_file_defaults()
# plt.rcParams.update(plt.rcParamsDefault)
# plt.rcParams['axes.facecolor'] = 'black'

%matplotlib inline
```

[868]:
```
# parse_dates=[0]: We give the function a hint that data in the first column␣
  ↪contains dates that need to be parsed.
# This argument takes a list, so we provide it a list of one element, which is␣
  ↪the index of the first column

Seoul_Bike_df = pd.read_csv('/Users/JohnnyBlaze/Website Data Sets/SeoulBikeData.
  ↪csv', encoding='unicode_escape', parse_dates=[0])
```

[869]:
```
Seoul_Bike_df.head()
```

[869]:
```
        Date  Rented Bike Count  Hour  Temperature(°C)  Humidity(%)  \
0 2017-01-12                254     0             -5.2           37
1 2017-01-12                204     1             -5.5           38
2 2017-01-12                173     2             -6.0           39
3 2017-01-12                107     3             -6.2           40
4 2017-01-12                 78     4             -6.0           36

   Wind speed (m/s)  Visibility (10m)  Dew point temperature(°C)  \
0               2.2              2000                      -17.6
1               0.8              2000                      -17.6
2               1.0              2000                      -17.7
3               0.9              2000                      -17.6
4               2.3              2000                      -18.6

   Solar Radiation (MJ/m2)  Rainfall(mm)  Snowfall (cm) Seasons     Holiday  \
0                      0.0           0.0            0.0  Winter  No Holiday
1                      0.0           0.0            0.0  Winter  No Holiday
2                      0.0           0.0            0.0  Winter  No Holiday
3                      0.0           0.0            0.0  Winter  No Holiday
4                      0.0           0.0            0.0  Winter  No Holiday

  Functioning Day
0             Yes
1             Yes
2             Yes
3             Yes
4             Yes
```

[870]:
```
Seoul_Bike_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8760 entries, 0 to 8759
Data columns (total 14 columns):
 #   Column                     Non-Null Count  Dtype
---  ------                     --------------  -----
 0   Date                       8760 non-null   datetime64[ns]
 1   Rented Bike Count          8760 non-null   int64
 2   Hour                       8760 non-null   int64
```

```
 3   Temperature(°C)           8760 non-null   float64
 4   Humidity(%)               8760 non-null   int64
 5   Wind speed (m/s)          8760 non-null   float64
 6   Visibility (10m)          8760 non-null   int64
 7   Dew point temperature(°C) 8760 non-null   float64
 8   Solar Radiation (MJ/m2)   8760 non-null   float64
 9   Rainfall(mm)              8760 non-null   float64
 10  Snowfall (cm)             8760 non-null   float64
 11  Seasons                   8760 non-null   object
 12  Holiday                   8760 non-null   object
 13  Functioning Day           8760 non-null   object
dtypes: datetime64[ns](1), float64(6), int64(4), object(3)
memory usage: 958.2+ KB
```

[871]:

```
Seoul_Bike_df = Seoul_Bike_df.astype({'Rented Bike Count':'float','Hour':
 ↪'object'})
# Seoul_Bike_df.info()
```

[872]:

```
# Reformat Column Names
Seoul_Bike_df = Seoul_Bike_df.copy()

Seoul_Bike_df.columns = [d.replace(' ','_').replace('.','') for d in␣
 ↪Seoul_Bike_df.columns]

Seoul_Bike_df = Seoul_Bike_df.rename(columns={'Wind_speed_(m/s)':'Wind_speed(m/
 ↪s)','Visibility_(10m)':'Visibility(10m)',
                                              'Solar_Radiation_(MJ/m2)':
 ↪'Solar_Radiation(MJ/m2)','Snowfall_(cm)':'Snowfall(cm)'})

# Print Column Names
for col in Seoul_Bike_df.columns:
    print(col)
```

```
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
```

[873]:
```
# Check for Nulls
Seoul_Bike_df.isnull().sum()
```

[873]:
```
Date                         0
Rented_Bike_Count            0
Hour                         0
Temperature(°C)              0
Humidity(%)                  0
Wind_speed(m/s)              0
Visibility(10m)              0
Dew_point_temperature(°C)    0
Solar_Radiation(MJ/m2)       0
Rainfall(mm)                 0
Snowfall(cm)                 0
Seasons                      0
Holiday                      0
Functioning_Day              0
dtype: int64
```

[874]:
```
Seoul_Bike_df.describe().transpose().style.
 ↪background_gradient(cmap='brg',axis=None)
```

[874]:
```
<pandas.io.formats.style.Styler at 0x7fb0f8429b50>
```

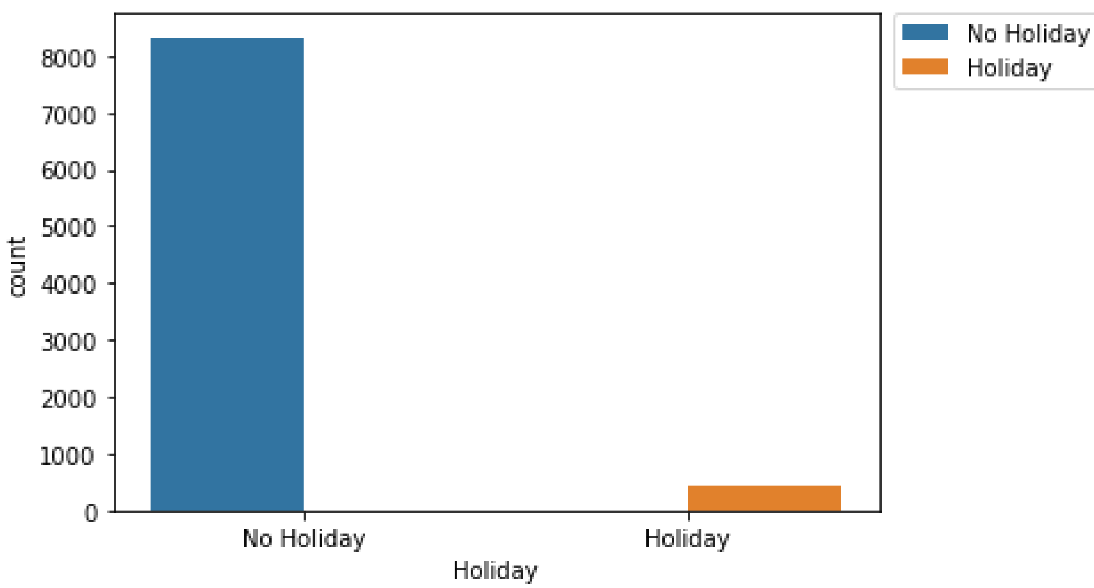
[875]:
```
# Count of Unique Values

Seoul_Bike_df.nunique().sort_values(ascending=False)
```

[875]:
```
Rented_Bike_Count            2166
Visibility(10m)              1789
Dew_point_temperature(°C)     556
Temperature(°C)               546
Date                          365
Solar_Radiation(MJ/m2)        345
Humidity(%)                    90
Wind_speed(m/s)                65
Rainfall(mm)                   61
Snowfall(cm)                   51
Hour                           24
Seasons                         4
Holiday                         2
Functioning_Day                 2
dtype: int64
```

[876]:
```
# Unique Object Dtype Values

print(Seoul_Bike_df.iloc[:, -3:].apply(lambda col: col.unique()))
```

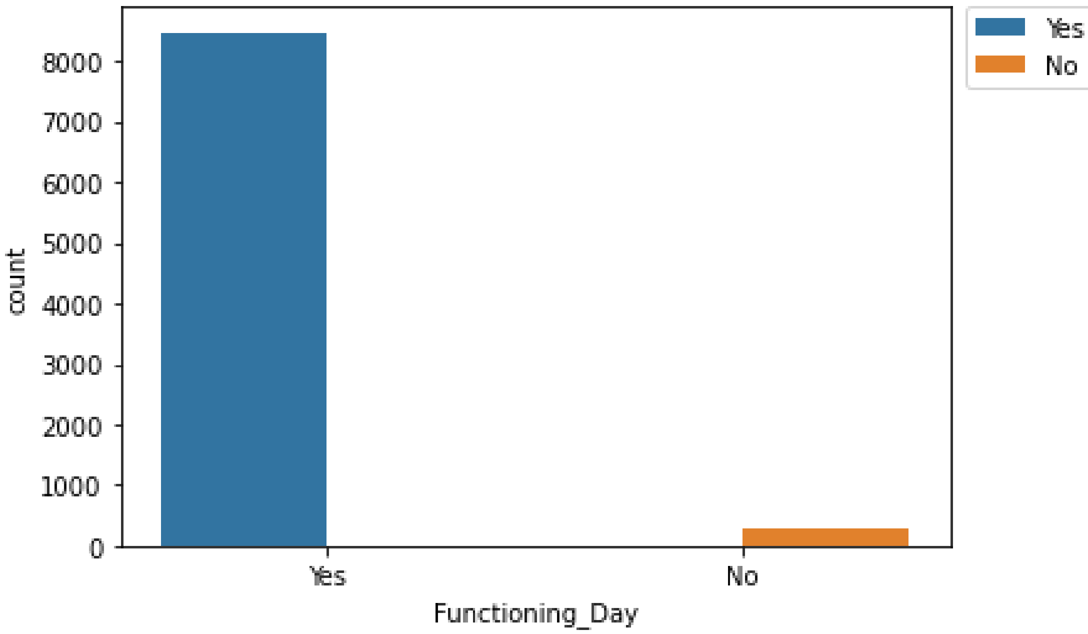
```
Seasons             [Winter, Spring, Summer, Autumn]
Holiday                        [No Holiday, Holiday]
Functioning_Day                            [Yes, No]
dtype: object
```

[877]:
```
# Counts of Holiday

sns.countplot(data=Seoul_Bike_df, x='Holiday', hue='Holiday')
plt.legend(bbox_to_anchor=(1.02, 1), loc='upper left', borderaxespad=0)
plt.show()

print(Seoul_Bike_df['Holiday'].value_counts())
print()
```




```
No Holiday    8328
Holiday        432
Name: Holiday, dtype: int64
```

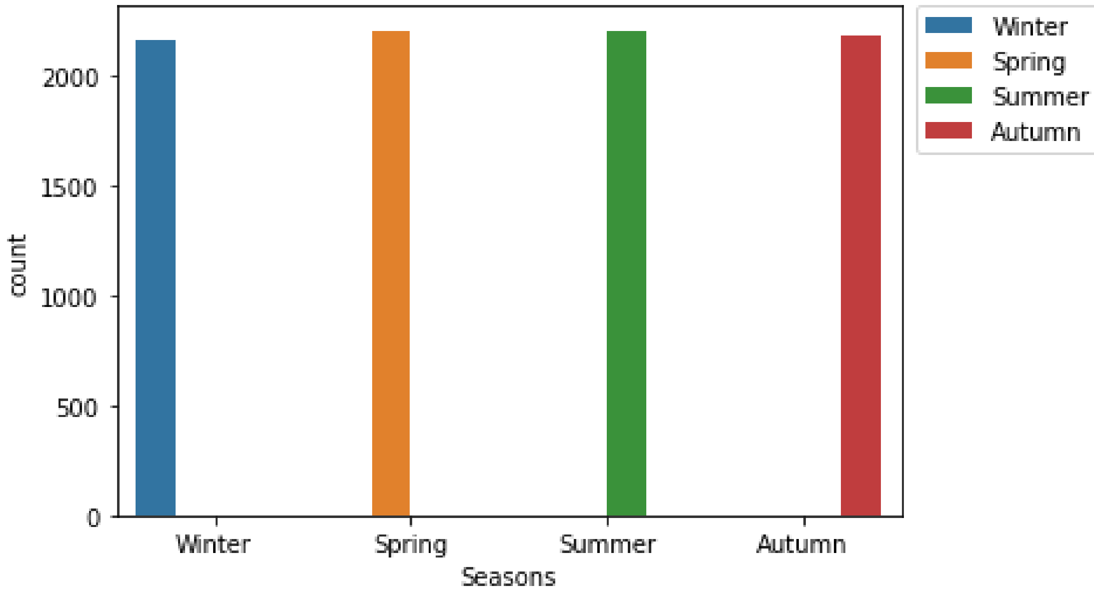
[878]:
```
# Counts of Functioning Day

sns.countplot(data=Seoul_Bike_df, x='Functioning_Day', hue='Functioning_Day')
plt.legend(bbox_to_anchor=(1.02, 1), loc='upper left', borderaxespad=0)
plt.show()

print(Seoul_Bike_df['Functioning_Day'].value_counts())
print()
```



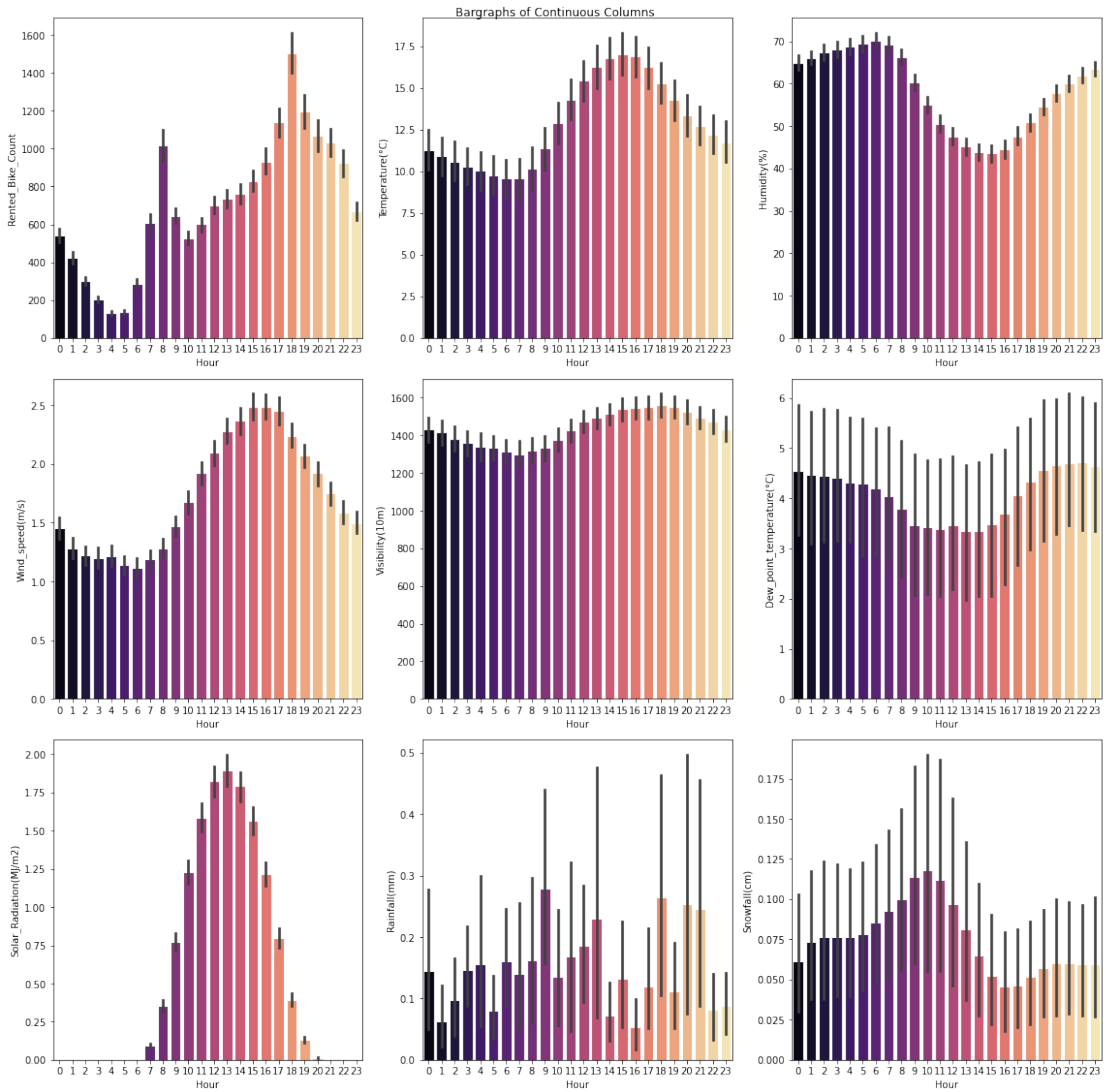

```
Yes    8465
No      295
Name: Functioning_Day, dtype: int64
```

[879]:
```
# Counts of Seasons

sns.countplot(data=Seoul_Bike_df, x='Seasons', hue='Seasons')
plt.legend(bbox_to_anchor=(1.02, 1), loc='upper left', borderaxespad=0)

print(Seoul_Bike_df['Seasons'].value_counts())
print()
plt.show()
```

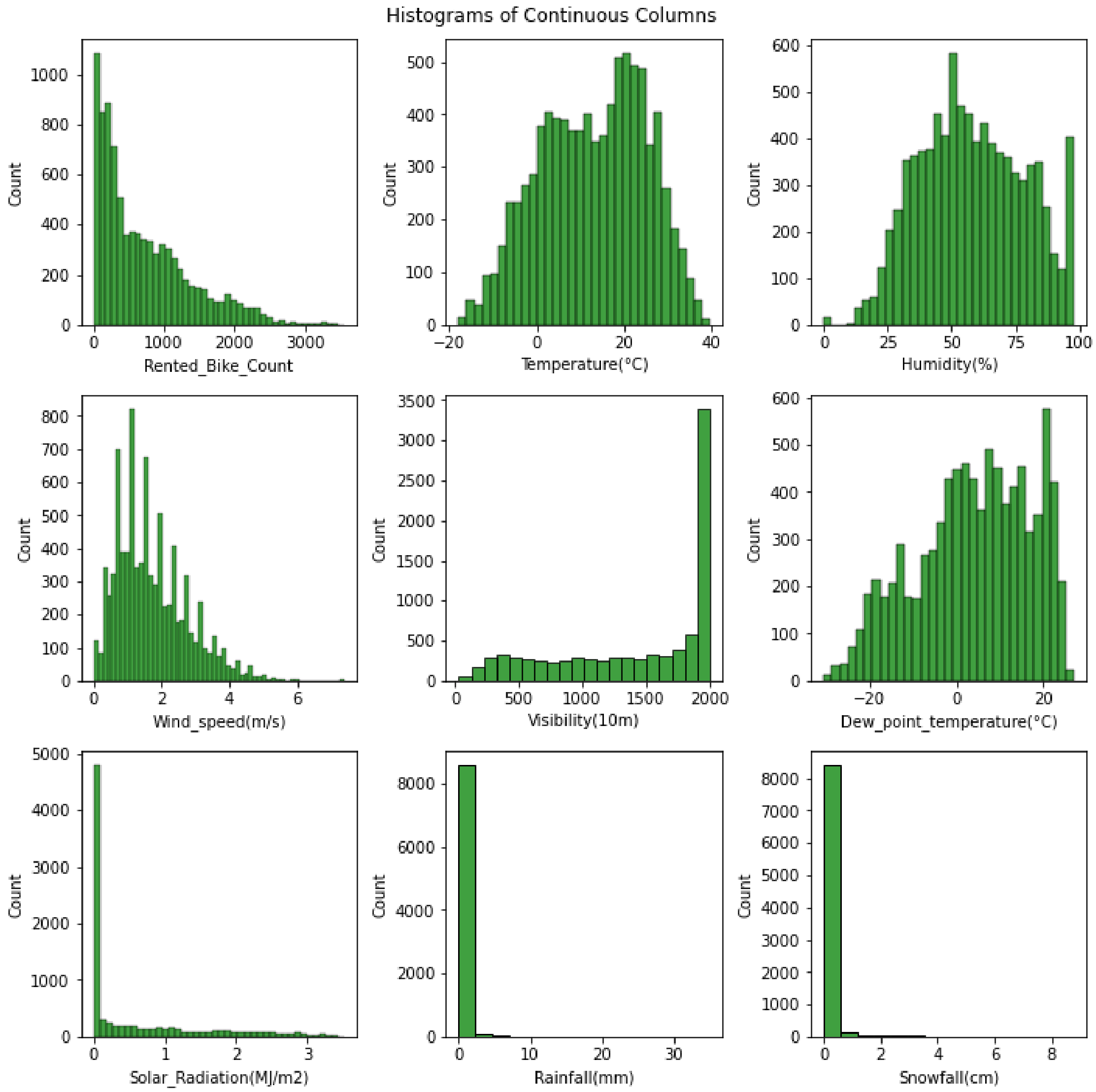
```
Spring    2208
Summer    2208
Autumn    2184
Winter    2160
Name: Seasons, dtype: int64
```



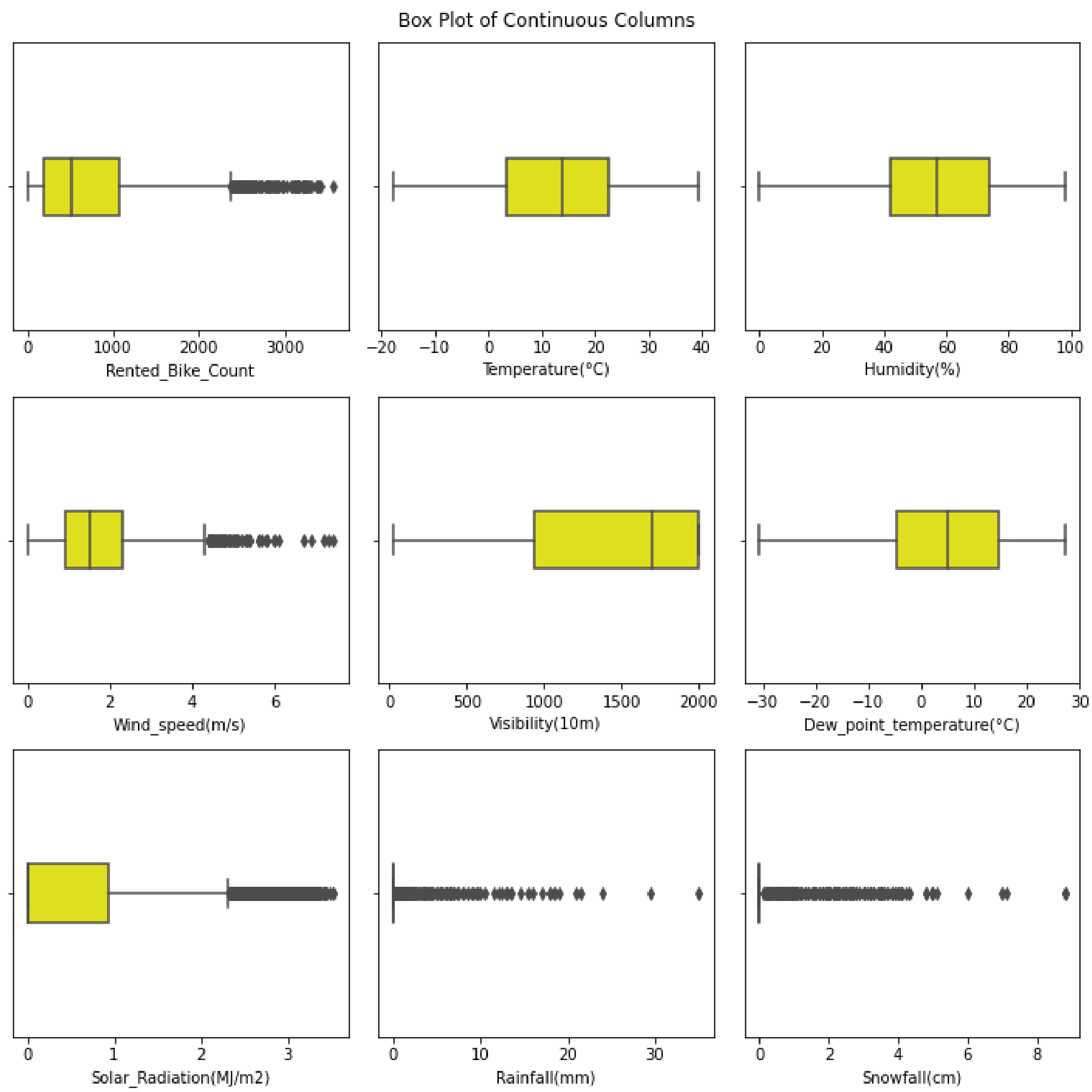
[880]:
```
# Bargraphs

plt.figure(figsize=(16, 16))
for i, col in enumerate(Seoul_Bike_df.
  ↪select_dtypes(exclude=['datetime64[ns]','object']).columns):
    ax = plt.subplot(3,3, i+1)
    sns.barplot(data=Seoul_Bike_df, x='Hour', y=col, ax=ax, edgecolor='white',␣
  ↪palette='magma')
plt.suptitle('Bargraphs of Continuous Columns')
plt.tight_layout()
plt.show()
```



[881]:
```
# Histograms

plt.figure(figsize=(10, 10))
for i, col in enumerate(Seoul_Bike_df.select_dtypes(include=['float','int']).
 ↪columns):
    ax = plt.subplot(3,3, i+1)
    sns.histplot(data=Seoul_Bike_df, x=col, ax=ax, color='green')
plt.suptitle('Histograms of Continuous Columns')
plt.tight_layout()
plt.show()
```



[882]:
```python
# Box & Whisker

plt.figure(figsize=(10, 10))
for i, col in enumerate(Seoul_Bike_df.select_dtypes(include=['float','int']).
 ↪columns):
    ax = plt.subplot(3,3, i+1)
    sns.boxplot(data=Seoul_Bike_df, x=col, ax=ax, color='yellow', width=0.2)
plt.suptitle('Box Plot of Continuous Columns')
plt.tight_layout()
plt.show()
```

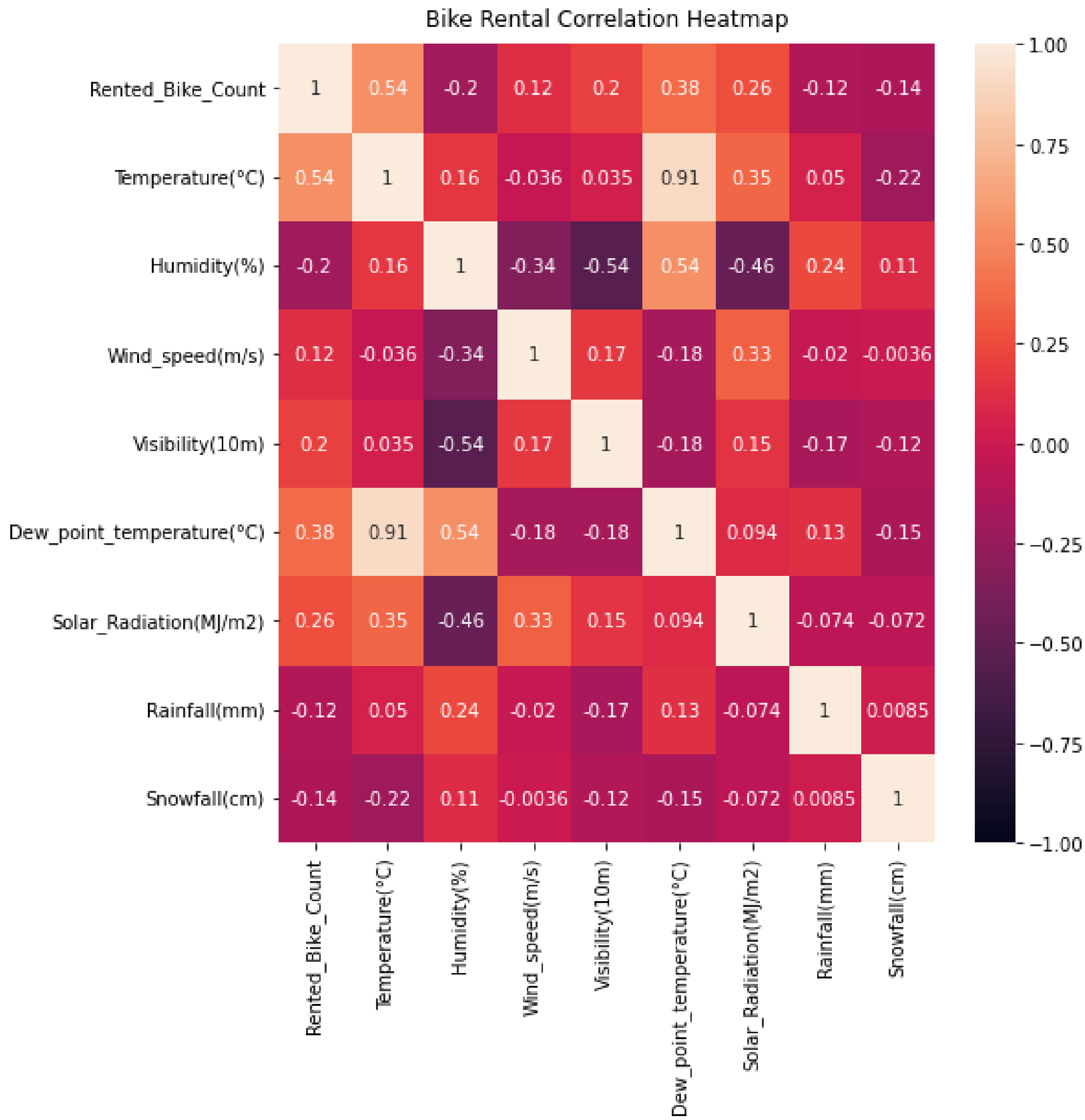


[883]:

```
# Count of Outliers

ContCols = Seoul_Bike_df.select_dtypes(include=['float','int'])

#ContCols.head()

Q1 = ContCols.quantile(0.25)
Q3 = ContCols.quantile(0.75)
IQR = Q3 - Q1

((ContCols < (Q1 - 1.5 * IQR)) | (ContCols > (Q3 + 1.5 * IQR))).sum()
```

[883]:
```
Rented_Bike_Count            158
Temperature(°C)                0
Humidity(%)                    0
Wind_speed(m/s)              161
Visibility(10m)                0
Dew_point_temperature(°C)      0
Solar_Radiation(MJ/m2)       641
Rainfall(mm)                 528
Snowfall(cm)                 443
dtype: int64
```

[884]:
```
# Correlation Heatmap

plt.figure(figsize=(8, 8))
heatmap = sns.heatmap(ContCols.corr(method='pearson'), vmin=-1, vmax=1,␣
↪annot=True)
heatmap.set_title('Bike Rental Correlation Heatmap', fontdict={'fontsize':12},␣
↪pad=10);
```



[885]:
```
# Sort Correlation Values

ContCols[ContCols.columns[:]].corr()['Rented_Bike_Count'][:].
 ↪sort_values(ascending=False)
```

[885]:
```
Rented_Bike_Count            1.000000
Temperature(°C)              0.538558
Dew_point_temperature(°C)    0.379788
Solar_Radiation(MJ/m2)       0.261837
Visibility(10m)              0.199280
Wind_speed(m/s)              0.121108
Rainfall(mm)                -0.123074
```

```
Snowfall(cm)                 -0.141804
Humidity(%)                  -0.199780
Name: Rented_Bike_Count, dtype: float64
```

[886]:
```
# Converting Categorical to Dummies

# Hour = pd.get_dummies(Seoul_Bike_df.index.hour, prefix='hour')
Seoul_Bike_df = pd.
 ↪get_dummies(Seoul_Bike_df,columns=['Holiday','Seasons','Functioning_Day'],drop_first=True)

# Seoul_Bike_df.head()
```

[887]:
```
X = Seoul_Bike_df[['Temperature(°C)', 'Humidity(%)', 'Wind_speed(m/
 ↪s)','Visibility(10m)','Dew_point_temperature(°C)',
    'Solar_Radiation(MJ/m2)','Rainfall(mm)','Snowfall(cm)']]
```

[888]:
```
# VIF Function

def _calc_vif(X):
    # Multicollinearity detection
    vif = pd.DataFrame()

    # point here suspicious variables or just all variables
    vif["VIF"] = [variance_inflation_factor(X.values, i) for i in range(X.
 ↪shape[1])]
    vif["variables"] = X.columns

    return(vif)
```

[889]:
```
# Run VIF

_calc_vif(X).sort_values(by=['VIF'], ascending=False) # High VIF from␣
 ↪temperature column and dew point
```

[889]:

	VIF	variables
0	29.075866	Temperature(°C)
4	15.201989	Dew_point_temperature(°C)
3	9.051931	Visibility(10m)
1	5.069743	Humidity(%)
2	4.517664	Wind_speed(m/s)
5	2.821604	Solar_Radiation(MJ/m2)
7	1.118903	Snowfall(cm)
6	1.079919	Rainfall(mm)

[890]:
```
# Remove Dew Point Column
```

```
X = Seoul_Bike_df[['Temperature(°C)', 'Humidity(%)', 'Wind_speed(m/
↪s)','Visibility(10m)',

                   'Solar_Radiation(MJ/m2)','Rainfall(mm)','Snowfall(cm)']]
```

[891]:
```
# VIF again

_calc_vif(X).sort_values(by=['VIF'], ascending=False)
```

[891]:

	VIF	variables
1	4.758651	Humidity(%)
3	4.409448	Visibility(10m)
2	4.079926	Wind_speed(m/s)
0	3.166007	Temperature(°C)
4	2.246238	Solar_Radiation(MJ/m2)
6	1.118901	Snowfall(cm)
5	1.078501	Rainfall(mm)

[892]:
```
# Define Predictor and Outcome

X = Seoul_Bike_df.iloc[:,2:]
y = Seoul_Bike_df['Rented_Bike_Count']

# X.head()
# X.shape
```

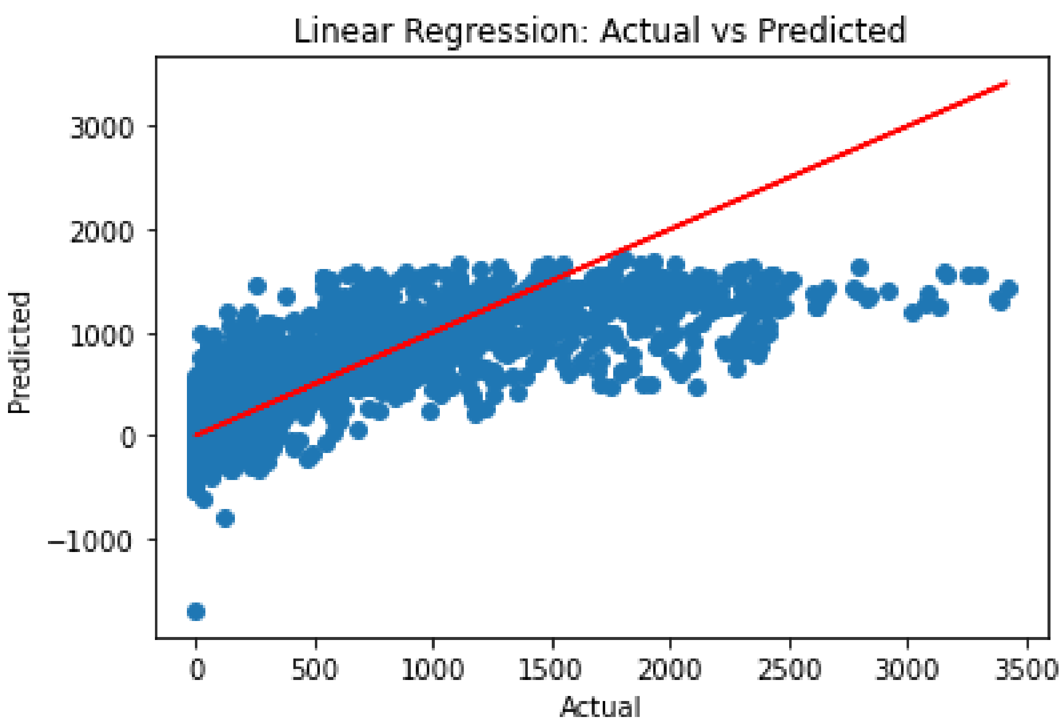
[893]:
```
# Split the Data - 75% train, 25% test

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25,␣
↪random_state=12345)
```

[894]:
```
# Scaling

sc = StandardScaler()
X_train_scaled = sc.fit_transform(X_train)
X_test_scaled = sc.transform(X_test)
```

[895]:
```
# Linear Regression

lin_reg = LinearRegression()
lin_reg.fit(X_train_scaled,y_train)

# Prediction
y_pred_lin = lin_reg.predict(X_test_scaled)
R2_lin = metrics.r2_score(y_test, y_pred_lin).round(4)
mae_lin = metrics.mean_absolute_error(y_test, y_pred_lin).round(4)
mse_lin = metrics.mean_squared_error(y_test, y_pred_lin).round(4)
```

```
rmse_lin = np.sqrt(mse_lin).round(4)

# Printing the metrics
print('Linear Regression Accuracy: ', lin_reg.score(X_test_scaled, y_test).
 ↪round(4))
print('R2 square:', R2_lin)
print('MAE: ', mae_lin)
print('MSE: ', mse_lin)
print('RMSE: ', rmse_lin)

# Scatterplot
plt.scatter(y_test, y_pred_lin)
plt.plot(y_test, y_test, color = 'red')
plt.xlabel('Actual')
plt.ylabel('Predicted')
plt.title('Linear Regression: Actual vs Predicted')
plt.show()
```

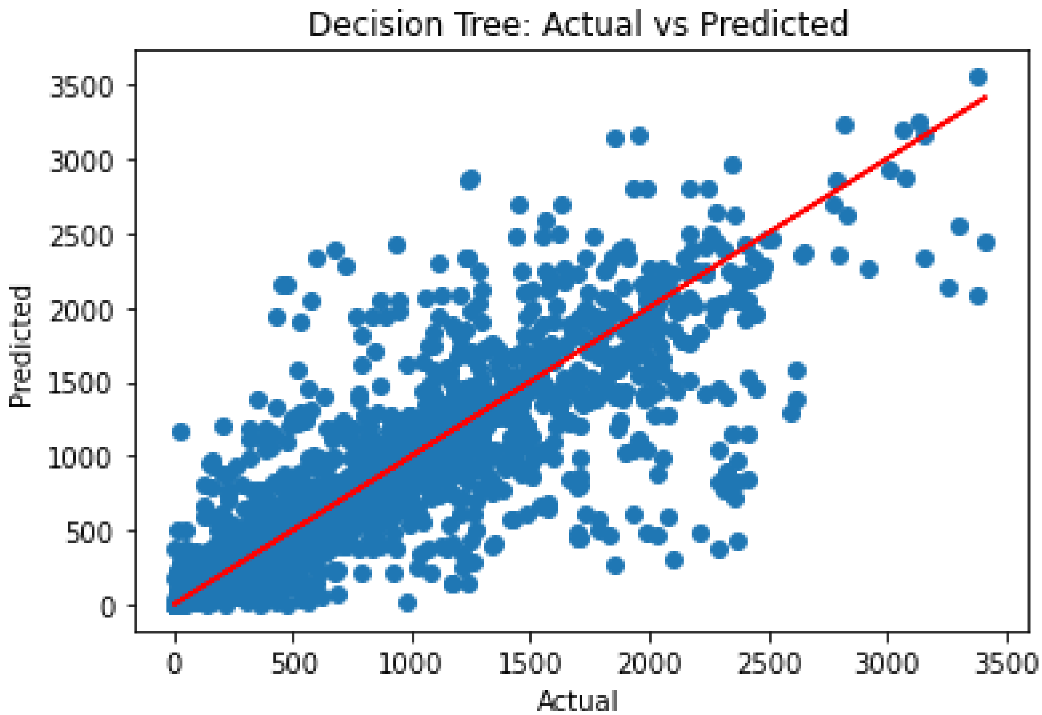
```
Linear Regression Accuracy:  0.5397
R2 square: 0.5397
MAE:  326.7375
MSE:  195443.5773
RMSE:  442.09
```



[896]:

```
# Decision Tree

dt_regressor = DecisionTreeRegressor(random_state = 12345)
dt_regressor.fit(X_train_scaled, y_train)

# Prediction
y_pred_dt = dt_regressor.predict(X_test_scaled)
R2_dt = metrics.r2_score(y_test, y_pred_dt).round(4)
mae_dt = metrics.mean_absolute_error(y_test, y_pred_dt).round(4)
mse_dt = metrics.mean_squared_error(y_test, y_pred_dt).round(4)
rmse_dt = np.sqrt(mse_dt).round(4)

# Printing the metrics
print('Decision Tree Regression Accuracy: ', dt_regressor.score(X_test_scaled,␣
 ↪y_test).round(4))
print('R2 square:', R2_dt)
print('MAE: ', mae_dt)
print('MSE: ', mse_dt)
print('RMSE: ', rmse_dt)

# Scatterplot
plt.scatter(y_test, y_pred_dt)
plt.plot(y_test, y_test, color = 'red')
plt.xlabel('Actual')
plt.ylabel('Predicted')
plt.title('Decision Tree: Actual vs Predicted')
plt.show()
```

```
Decision Tree Regression Accuracy:  0.7192
R2 square: 0.7192
MAE:  198.4635
MSE:  119256.1356
RMSE:  345.3348
```

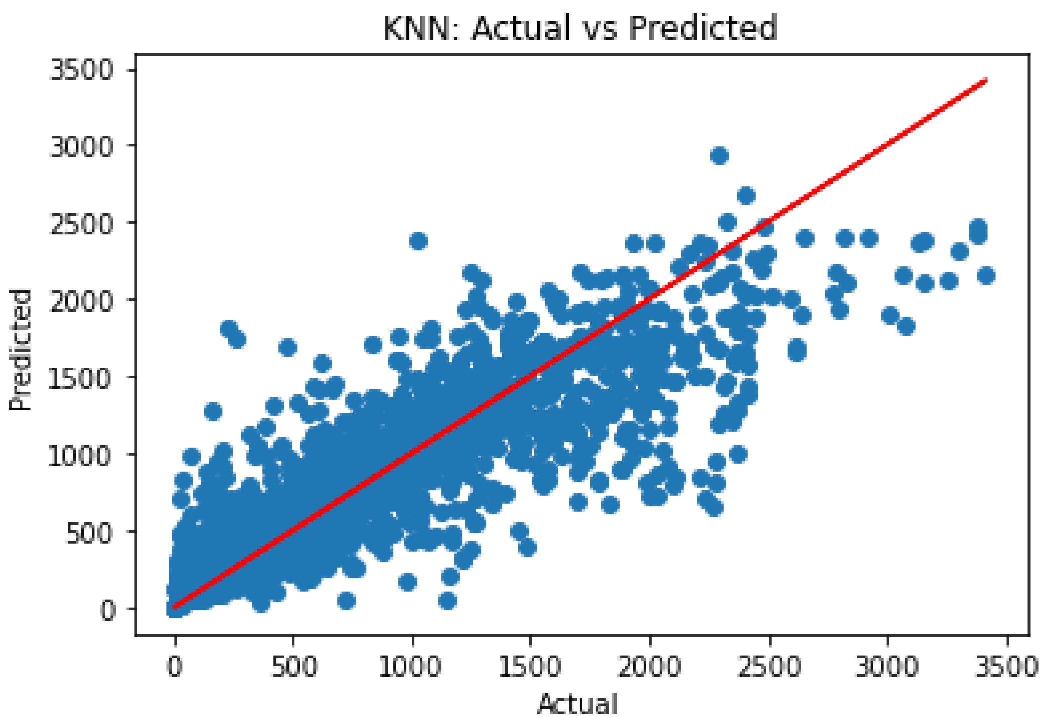


[897]:

```
# KNN

knn_9 = KNeighborsRegressor(n_neighbors=9)
KNeighborsRegressor(algorithm='auto', leaf_size=40, metric='minkowski',
                    metric_params=None, n_jobs=-1, n_neighbors=9, p=2,␣
 ↪weights='uniform')
knn_9.fit(X_train_scaled, y_train)
# print(knn_9)

# Prediction
y_pred_knn = knn_9.predict(X_test_scaled)
R2_knn = metrics.r2_score(y_test, y_pred_knn).round(4)
mae_knn = metrics.mean_absolute_error(y_test, y_pred_knn).round(4)
mse_knn = metrics.mean_squared_error(y_test, y_pred_knn).round(4)
rmse_knn = np.sqrt(mse_knn).round(4)

# Printing the metrics
print('KNN Regression Accuracy: ', knn_9.score(X_test_scaled, y_test).round(4))
print('R2 square:', R2_knn)
print('MAE: ', mae_knn)
print('MSE: ', mse_knn)
print('RMSE: ', rmse_knn)
```

```
# Scatterplot
plt.scatter(y_test, y_pred_knn)
plt.plot(y_test, y_test, color = 'red')
plt.xlabel('Actual')
plt.ylabel('Predicted')
plt.title('KNN: Actual vs Predicted')
plt.show()
```

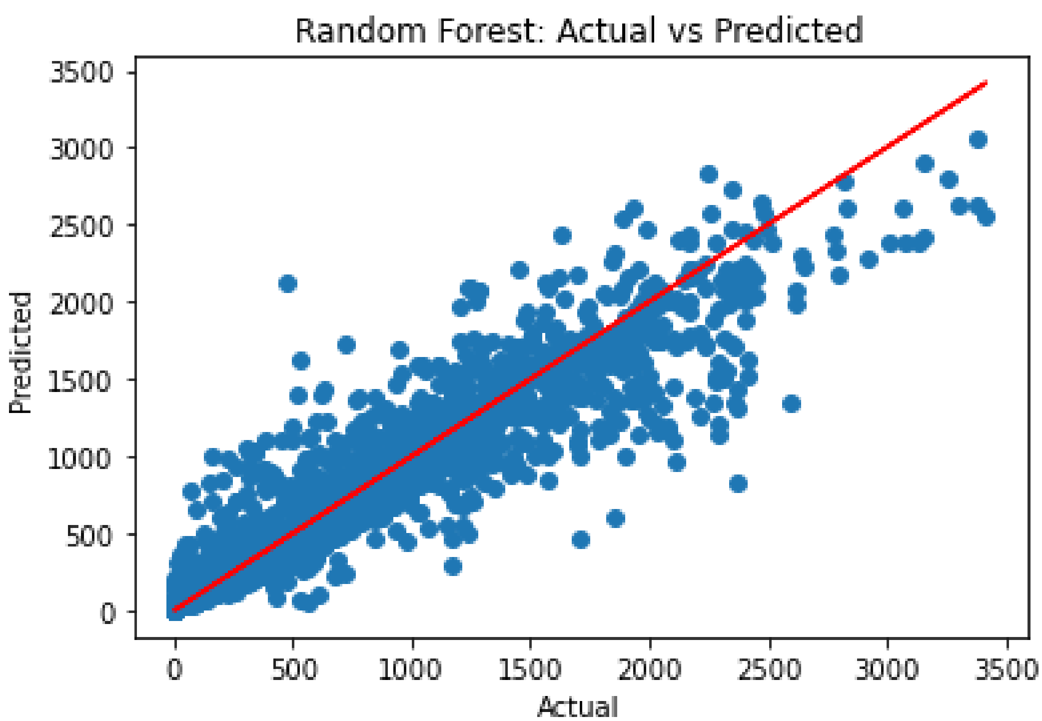
```
KNN Regression Accuracy:  0.7747
R2 square: 0.7747
MAE:  201.818
MSE:  95671.1062
RMSE:  309.3075
```



[898]:
```
# Random Forest Regression

rf_regressor = RandomForestRegressor(n_estimators = 300 ,  random_state = 12345)
rf_regressor.fit(X_train_scaled, y_train)

# Prediction
y_pred_rf = rf_regressor.predict(X_test_scaled)
R2_rf = metrics.r2_score(y_test, y_pred_rf).round(4)
mae_rf = metrics.mean_absolute_error(y_test, y_pred_rf).round(4)
mse_rf = metrics.mean_squared_error(y_test, y_pred_rf).round(4)
```

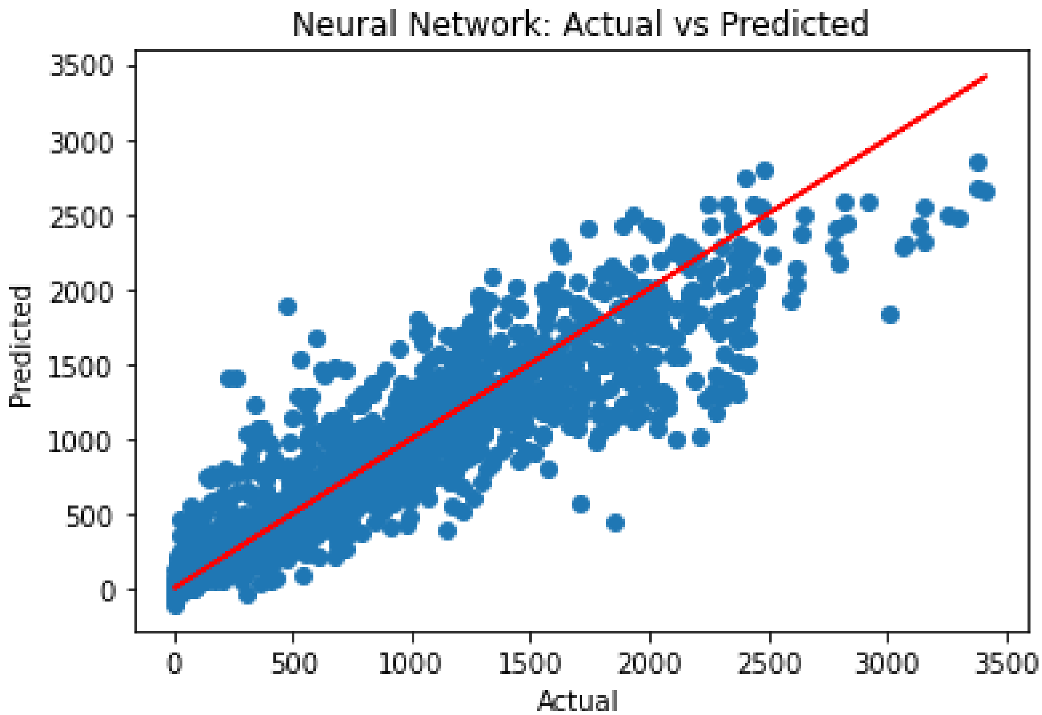
```python
rmse_rf = np.sqrt(mse_rf).round(4)

# Printing the metrics
print('Random Forest Regression Accuracy: ', rf_regressor.score(X_test_scaled,␣
 ↪y_test).round(4))
print('R2 square:', R2_rf)
print('MAE: ', mae_rf)
print('MSE: ', mse_rf)
print('RMSE: ', rmse_rf)

# Scatterplot
plt.scatter(y_test, y_pred_rf)
plt.plot(y_test, y_test, color = 'red')
plt.xlabel('Actual')
plt.ylabel('Predicted')
plt.title('Random Forest: Actual vs Predicted')
plt.show()
```

```
Random Forest Regression Accuracy:  0.8642
R2 square: 0.8642
MAE:  147.5391
MSE:  57655.2637
RMSE:  240.1151
```



[899]:
```
# Neural Network

mlp_reg = MLPRegressor(hidden_layer_sizes =(150,100,50), max_iter = 300,␣
↪activation = 'relu',
                       solver = 'adam', random_state = 12345)
mlp_reg.fit(X_train_scaled, y_train)

# Prediction
y_pred_nn = mlp_reg.predict(X_test_scaled)
R2_nn = metrics.r2_score(y_test, y_pred_nn).round(4)
mae_nn = metrics.mean_absolute_error(y_test, y_pred_nn).round(4)
mse_nn = metrics.mean_squared_error(y_test, y_pred_nn).round(4)
rmse_nn = np.sqrt(mse_nn).round(4)

# Printing the metrics
print('Neural Network Regression Accuracy: ', mlp_reg.score(X_test_scaled,␣
↪y_test).round(4))
print('R2 square:', R2_nn)
print('MAE: ', mae_nn)
print('MSE: ', mse_nn)
print('RMSE: ', rmse_nn)

# Scatterplot
plt.scatter(y_test, y_pred_nn)
plt.plot(y_test, y_test, color = 'red')
plt.xlabel('Actual')
plt.ylabel('Predicted')
plt.title('Neural Network: Actual vs Predicted')
plt.show()
```

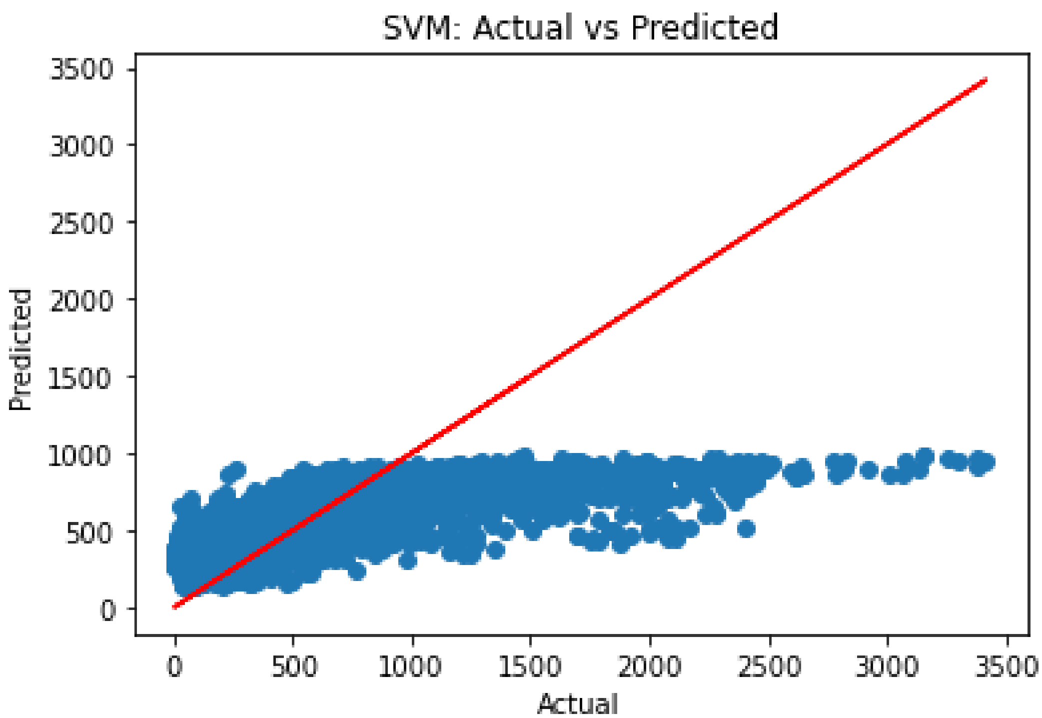
```
Neural Network Regression Accuracy:  0.8548
R2 square: 0.8548
MAE:  158.2314
MSE:  61641.2817
RMSE:  248.2766
```



[900]:
```
# SVM

regressor = SVR(kernel='rbf')
regressor.fit(X_train_scaled,y_train)

# Prediction
y_pred_svm = regressor.predict(X_test_scaled)
R2_svm = metrics.r2_score(y_test, y_pred_svm).round(4)
mae_svm = metrics.mean_absolute_error(y_test, y_pred_svm).round(4)
mse_svm = metrics.mean_squared_error(y_test, y_pred_svm).round(4)
rmse_svm = np.sqrt(mse_svm).round(4)

# Printing the metrics
print('Suppport Vector Regression Accuracy: ', regressor.score(X_test_scaled,␣
 ↪y_test).round(4))
print('R2 square:', R2_svm)
print('MAE: ', mae_svm)
print('MSE: ', mse_svm)
print('RMSE: ', rmse_svm)

# Scatterplot
plt.scatter(y_test, y_pred_svm)
plt.plot(y_test, y_test, color = 'red')
```

```
plt.xlabel('Actual')
plt.ylabel('Predicted')
plt.title('SVM: Actual vs Predicted')
plt.show()
```

```
Suppport Vector Regression Accuracy:  0.3373
R2 square: 0.3373
MAE:  353.3543
MSE:  281388.0445
RMSE:  530.4602
```



[901]:
```
# Table Results

Table = PrettyTable(["Model","R-Squared","MAE","MSE","RMSE"])
Table.add_row(["Linear Regression", R2_lin, mae_lin, mse_lin, rmse_lin])
Table.add_row(["Decision Tree", R2_dt, mae_dt, mse_dt, rmse_dt])
Table.add_row(["KNN", R2_knn, mae_knn, mse_knn, rmse_knn])
Table.add_row(["Random Forest", R2_rf, mae_rf, mse_rf, rmse_rf])
Table.add_row(["Neural Network", R2_nn, mae_nn, mse_nn, rmse_nn])
Table.add_row(["SVM", R2_svm, mae_svm, mse_svm, rmse_svm])
print("Models Performance Sorted by R-Squared Values")
Table.sortby = "R-Squared"
print(Table)
```

Models Performance Sorted by R-Squared Values

Model	R-Squared	MAE	MSE	RMSE
SVM	0.3373	353.3543	281388.0445	530.4602
Linear Regression	0.5397	326.7375	195443.5773	442.09
Decision Tree	0.7192	198.4635	119256.1356	345.3348
KNN	0.7747	201.818	95671.1062	309.3075
Neural Network	0.8548	158.2314	61641.2817	248.2766
Random Forest	0.8642	147.5391	57655.2637	240.1151

ADS-505 Team Project Form & Business Brief Templates Team Project Form

Fill out this form and business brief and submit it by the end of Module 3 in Blackboard (2 pages max for each). Reference the file, "Final Project Business Brief Requirements.doc."

Team Number: **5**

Team Leader/Representative: **Kyle Esteban Dalope**

Full Names of Team Members:

1. Harini Lakshmanan
2. Kyle Esteban Dalope
3. John J Chen

Title of Your Project: Predicting Bike Rental Counts in Seoul Based on the Weather and Holiday Information for a Stable Supply

Short Description of Your Project and Objectives: "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."

Name of Your Selected Dataset and Programming Language: **Python**

Description of Your Selected Dataset (source, number of variables, size of the dataset, etc.):
Source: UCI Machine Learning Repository
Number of variables: 14
Number of instances/size: 8760
Data donated: 2020-03-01
Link: https://archive.ics.uci.edu/ml/datasets/Seoul+Bike+Sharing+Demand#

Provide your team GitHub link here: https://github.com/jjchen-SEA/ADS-505-Seoul_Bike_Share

How many times have your team members met so far? **Twice**

What was the agreed-upon method of communication? Are you using any teamwork project management software, such as Deepnote, Trello, or Asana? If not, explain why?

Method of communication will be conducted through Slack as it provides a quick and easy channel for all team members. Project management will be discussed and monitored with Slack and through a shared Google folder for visibility.

Comments/ Roadblocks: Scheduling and time availability is limited

Team Project Business Brief

Purpose:
The main objective of this project is to effectively estimate the demand of rental bikes in Seoul, South Korea, based on thirteen different influencing variables including season, climatic conditions. time etc. Prediction of the demand would help in shaping the bike rental business and prepare for increase in demand, if not met could lead to increase in wait times for the customers, hence affecting accessibility and revenue loss for the company. It would also help to understand downtime periods which can be attributed towards servicing the vehicles to keep them ready.

Background:
Bike sharing is a system that is popular in large cities. It has many benefits as it is a green method of traveling and reduces traffic congestion. The system is easy to use and the person renting a bike just needs to download an app that will unlock the bike and be able to track the location of the bike. This system allows people to rent bikes from a location and then return them to a different spot on an as-needed basis.

Current Situation:
What is interesting is the data points that we can get from these rental systems. We have data on time, location, distance traveled, and more. This can help the company decide on how many bikes to stock in certain locations and even based on time of day perhaps more customers rent bikes during the day to get to work. The goal is to figure out the demand or count of bikes rented.

Conclusion:
Our conclusion is to successfully identify a final model that accurately predicts the number of rental bikes in Seoul throughout the year. Rental bikes have become a popular transportation

method that allows residents and visitors in urban cities to be mobile, minimize fuel consumption, and provide convenience for commutes. This dataset that contains weather information, seasons, holidays, and functional days will provide important insight into determining a stable supply of rental bikes throughout various locations in Seoul and for urban cities similar to it.