2019

Predicting Bike Rental

Omkar Annabathula

8/10/2019

Table of Contents

Introduction

Now a day's transportations are becoming very easy to commute from one place to another. Bike renting systems are one of the best solution where we can rent bicycles where the process of obtaining membership, rental, and bike return is automated via a network of kiosk locations throughout a city. Using these systems, people are able to rent a bike from a one location and return it to a different place on an as-needed basis. Currently, there are over 500 bike-sharing programs around the world.

Several bike/scooter ride sharing facilities (e.g., Vogo, Driverzy, Rapido, Bike Share) have started up lately especially in metropolitan cities and one of the most important problem from a business point of view is to predict the bike demand on any particular day. While having excess bikes results in wastage of resource (both with respect to bike maintenance and the land/bike stand required for parking and security), having fewer bikes leads to revenue loss (ranging from a short term loss due to missing out on immediate customers to potential longer term loss due to loss in future customer base), Thus, having an estimate on the demands would enable efficient functioning of these companies.

1.1 Problem Statement

The objective of this Case is to Predication of bike rental count on daily based on the environmental and seasonal settings. The details of data attributes in the dataset are as follows. The data set consists of day.csv, containing data to train and test the prediction algorithm.

1.2 Data Set

The data set consists of 731 observations recorded between the period of 2 Years, between 2011 and 2012. It has 15 variables or predictors and 1 target variable. The data fields in the given data file are enumerated below.

Variable Names	Description
instant	Record index
dteday	Date
Season	Season (1:springer, 2:summer, 3:fall, 4:winter)
yr	Year (0: 2011, 1:2012)
mnth	Month (1 to 12)
hr	Hour (0 to 23)
holiday	weather day is holiday or not (extracted from Holiday Schedule)
weekday	Day of the week
workingday	If day is neither weekend nor holiday is 1, otherwise is 0
weathersit	(extracted fromFreemeteo) 1: Clear, Few clouds, Partly cloudy, Partly cloudy 2: Mist + Cloudy, Mist + Broken clouds, Mist + Few clouds, Mist

	3: Light Snow, Light Rain + Thunderstorm + Scattered clouds, Light Rain + Scattered clouds 4: Heavy Rain + Ice Pallets + Thunderstorm + Mist, Snow + Fog
temp	Normalized temperature in Celsius. The values are derived via (t-t_min)/(t_max-t_min), t_min=-8, t_max=+39 (only in hourly scale)
atemp	Normalized feeling temperature in Celsius. The values are derived via (t-t_min)/(t_maxt_min), t_min=-16, t_max=+50 (only in hourly scale)
hum	Normalized humidity. The values are divided to 100 (max)
windspeed	Normalized wind speed. The values are divided to 67 (max)
casual	count of casual users
registered	count of registered users
cnt	count of total rental bikes including both casual and registered

Table 1. Description of variables

The given data set consists of 8 categorical, 7 continuous and 1 target Variable. sample data is as below.

	instant	dteday	season	yr	mnth	holiday	weekday	workingday	weathersit	temp	atemp	hum	windspeed	casual	registered	cnt
0	1	2011-01-01	1	0	1	0	6	0	2	0.344167	0.363625	0.805833	0.160446	331	654	985
1	2	2011-01-02	1	0	1	0	0	0	2	0.363478	0.353739	0.696087	0.248539	131	670	801
2	3	2011-01-03	1	0	1	0	1	1	1	0.196364	0.189405	0.437273	0.248309	120	1229	1349
3	4	2011-01-04	1	0	1	0	2	1	1	0.200000	0.212122	0.590435	0.160296	108	1454	1562
4	5	2011-01-05	1	0	1	0	3	1	1	0.226957	0.229270	0.436957	0.186900	82	1518	1600

Table 2. Instance of Sample Data

Methodology

The solution of this problem is divided into three parts. First was EDA (Exploratory Data analysis) and pre-processing, followed by modelling and performance tuning and comparison. During first part data pre-processing step like missing value analysis, outlier analysis, univariate and bi-variate analysis etc. were performed. After that data was split into train and test. The target variable is a continuous variable, so it a regression problem. Linear regression and Random forest regression were used for modelling and their performance comparison was performed. Both the algorithms were implemented in R and python.

2.1 Pre-processing

Pre-processing was performed in both R and python. The dataset consists of 731 observations, and 16 predictors. The process of pre-processing techniques was used for cleaning and reorder the data set in a proper format by changing into categorical variables and Variable (columns) names.

Index	Date	Season	Year	Month	Holiday	Weekday	Workingday	Weather	Temperature	Atemperature	Humidity	Windspeed	Casual Users	Registered Users	Count
1	2011-01-01	Spring	2011	Jan	0	6	0	Misty+Cloudy	0.344167	0.363625	0.805833	0.160446	331	654	985
2	2011-01-02	Spring	2011	Jan	0	0	0	Misty+Cloudy	0.363478	0.353739	0.696087	0.248539	131	670	801
3	2011-01-03	Spring	2011	Jan	0	1	1	Clear	0.196364	0.189405	0.437273	0.248309	120	1229	1349
4	2011-01-04	Spring	2011	Jan	0	2	1	Clear	0.200000	0.212122	0.590435	0.160296	108	1454	1562

Table 3. Instance of processed Data

2.1.1 Target Variable – 'cnt'

The target variable in the problem statement is the total count of registered and casual users of bikes on a single day. 'Count' is the combined value of 'Registered' and 'Casual' variables. The summary statistics of 'cnt' are as follow.

	Temperature	Atemperature	Humidity	Windspeed	Casual Users	Registered Users	Count
count	731.000000	731.000000	731.000000	731.000000	731.000000	731.000000	731.000000
mean	0.495385	0.474354	0.627894	0.190486	848.176471	3656.172367	4504.348837
std	0.183051	0.162961	0.142429	0.077498	686.622488	1560.256377	1937.211452
min	0.059130	0.079070	0.000000	0.022392	2.000000	20.000000	22.000000
25%	0.337083	0.337842	0.520000	0.134950	315.500000	2497.000000	3152.000000
50%	0.498333	0.486733	0.626667	0.180975	713.000000	3662.000000	4548.000000
75%	0.655417	0.608602	0.730209	0.233214	1096.000000	4776.500000	5956.000000
max	0.861667	0.840896	0.972500	0.507463	3410.000000	6946.000000	8714.000000

Table.4 Future Summary Statistics of Target Variable ('Count')

2.1.2 Missing value Analysis

Missing value analysis was performed on the dataset. No missing values were found. Missing values distribution can be seen below.

	0
Season	0
Year	0
Month	0
Holiday	0
Weekday	0
Workingday	0
Weather	0
Temperature	0
Atemperature	0
Humidity	0
Windspeed	0
Casual Users	0
Registered Users	0
Count	0

2.2 Exploratory Data Analysis (EDA)

2.2.1 Outlier Analysis

After missing value analysis, we check for outliers in target variable and predictors. There were no outliers present in the dataset. Some extreme values were present in the predictors but those seems to logical. So no observations were removed and no imputation was performed on the dataset.

Boxplot method was used to check for outliers. Below are the figures from the python implementation. Box plots from R implementation can be found in appendix.

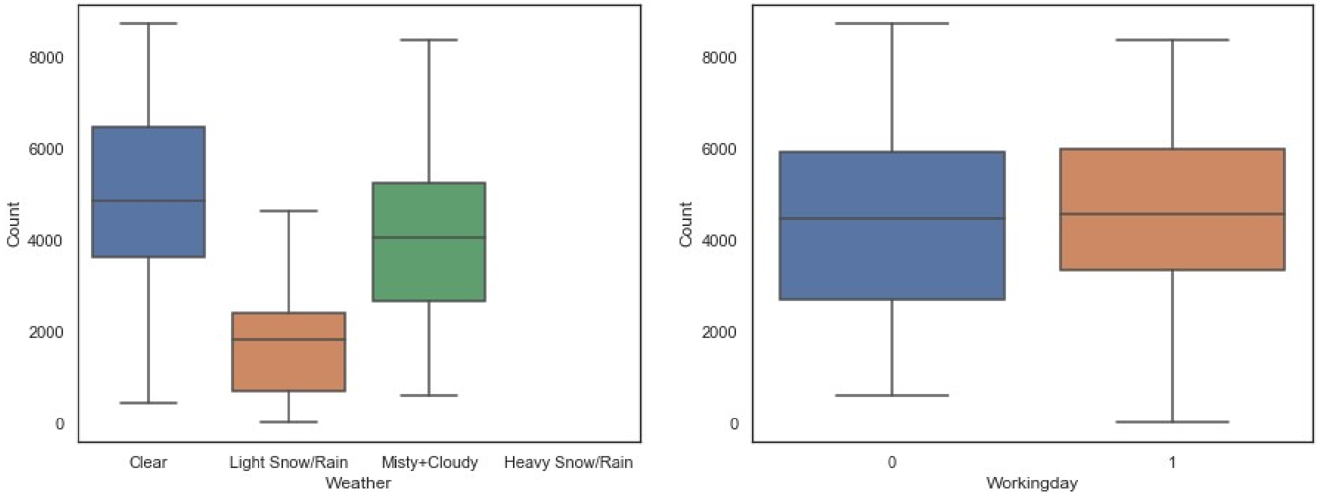


Fig.2.2.0 Box plot for 'Count vs Weather' & 'Count vs Workingday'

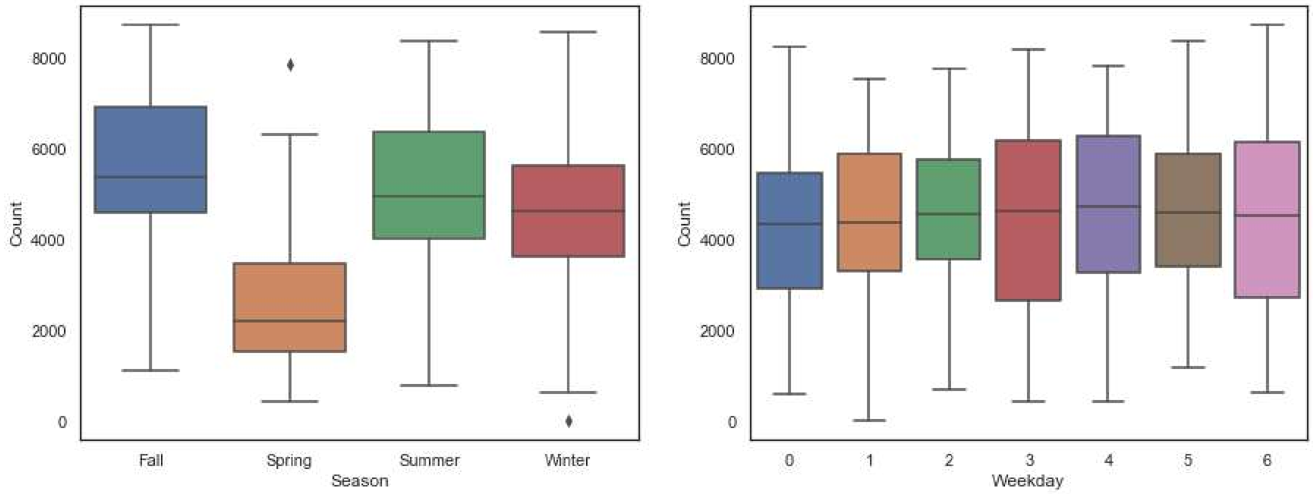


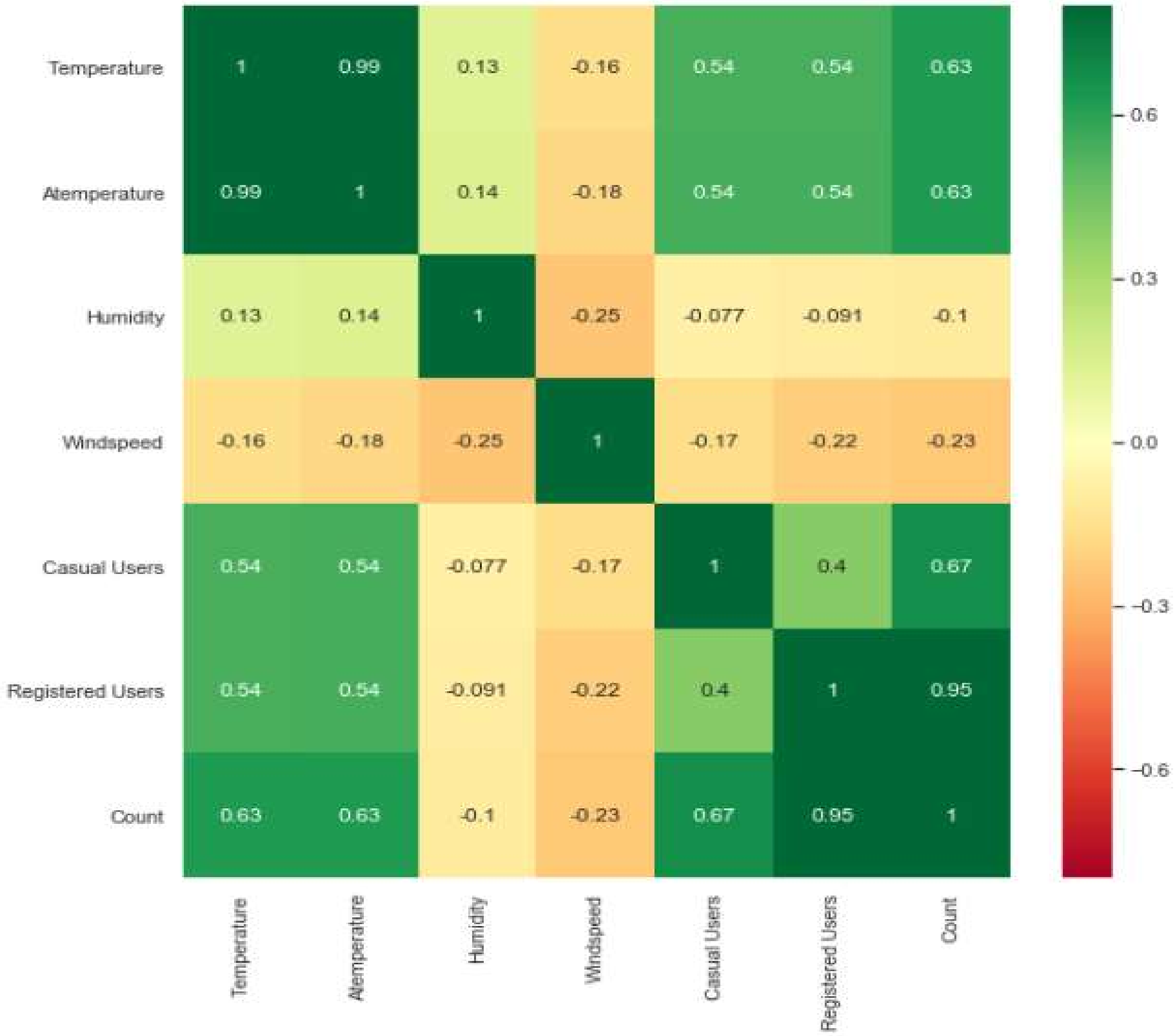
Fig.2.2.1 Box plot for 'Count vs Season' & 'Count vs Weekday'

After examining the above boxplots, we can see that there are some extreme values but no outliers. From these boxplots we can also infer that

- Bike demand count ('cnt') is low in spring (1) season.
- There is no effect on bike count('cnt') due to a holiday or a working day.
- Bikes are rented mostly in good weather (Clear, Few clouds, Partly cloudy, Partly cloudy) and least in bad (Light Snow, Light Rain + Thunderstorm + Scattered clouds, Light Rain + Scattered clouds) weather.

2.2.2 Correlation Analysis

Correlation analysis is used for checking a linear relationship between continuous predictor and target. It is also used to check for multicollinearity among predictors. Multicollinearity exists whenever two or more of the predictors in a regression model are moderately or highly correlated. Multicollinearity is the condition when one predictor can be used to predict other. The basic problem is multicollinearity results in unstable estimation of coefficients which makes it difficult to access the effect of independent variable on dependent variable. Figure6 is showing the correlation matrix for bike rent dataset.



'registered' and 'casual' were not included in correlation matrix because their sum is equal to the 'cnt' i.e. Target variable.

From the correlation matrix, it is revealed that

- Temperature and Atemperature (ambient temperature) are highly collinear. One of them should be removed before modelling.
- 'Count' have a strong and positive relationship with temperature and ambient temperature which is logical. People tends to rent bikes more which temperature is higher.
- 'Count' is negative relationship with Humidity and Windspeed. People tends to rent bike more when there is less humidity and wind speed.
- Also the relationship between 'Humidity', 'Windspeed' and 'Count' is very weak. These are not very strong predictors.

2.2.3 Univariate analysis

In univariate analysis, we look at the distribution and summary statistics of each variable.

- **Temperature**

```
Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
0.05913 0.33708 0.49833 0.49538 0.65542 0.86167
```

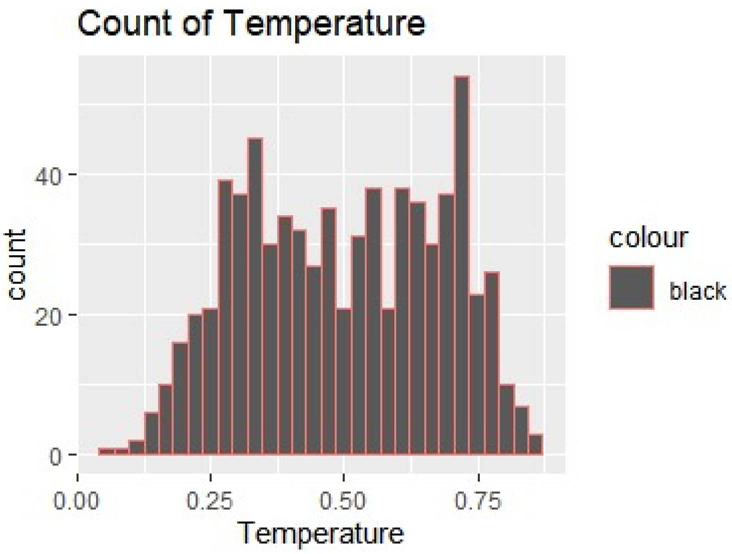


Fig. 2.2.3.0 Univariate analysis of Temperature

- **Atemperature**

```
Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
0.07907 0.33784 0.48673 0.47435 0.60860 0.84090
```

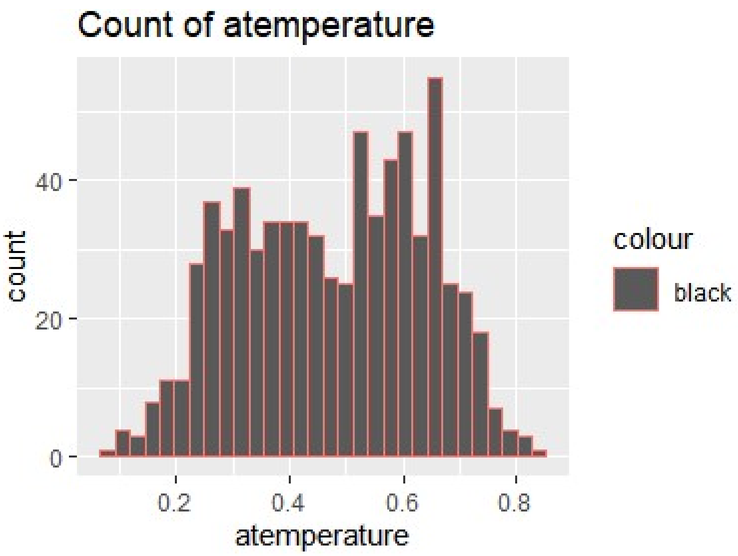


Fig. 2.2.3.1 Univariate analysis of Atemperature

➢ Humidity

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
0.0000  0.5200  0.6267  0.6279  0.7302  0.9725 
```

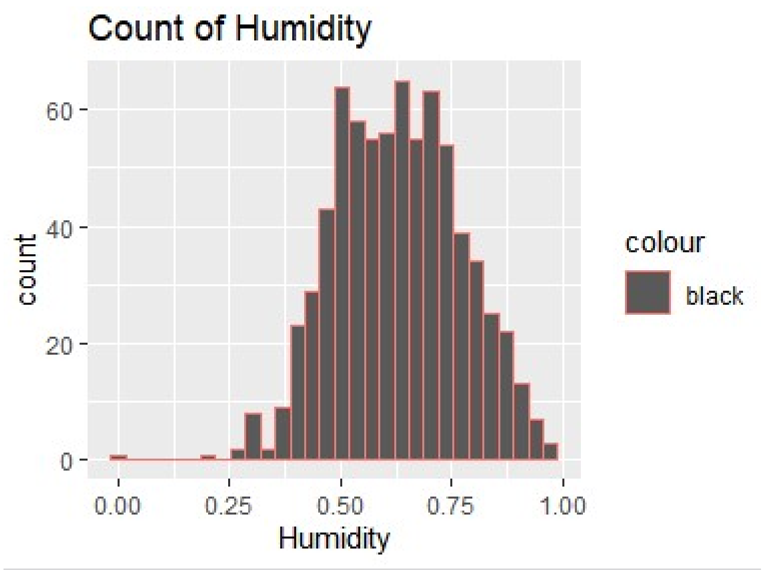


Fig. 2.2.3.2 Univariate analysis of Humidity

➢ Windspeed

```
Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
0.02239 0.13495 0.18097 0.19049 0.23321 0.50746 
```

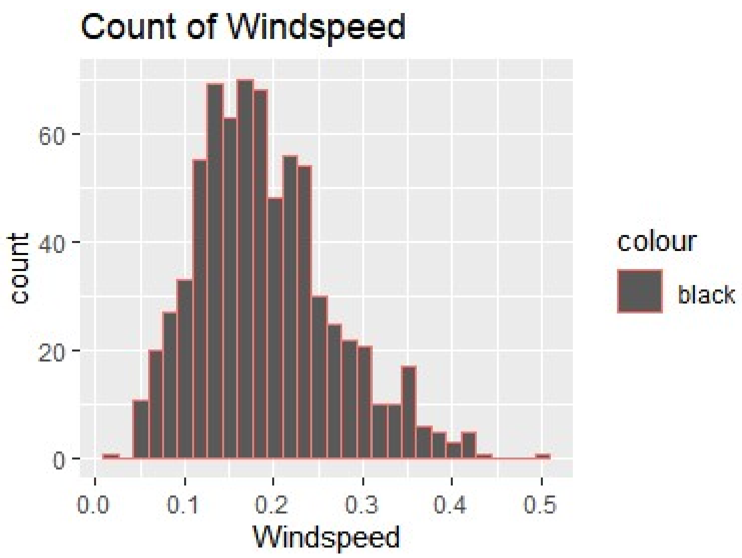


Fig. 2.2.3.3 Univariate analysis of Windspeed

➢ Season

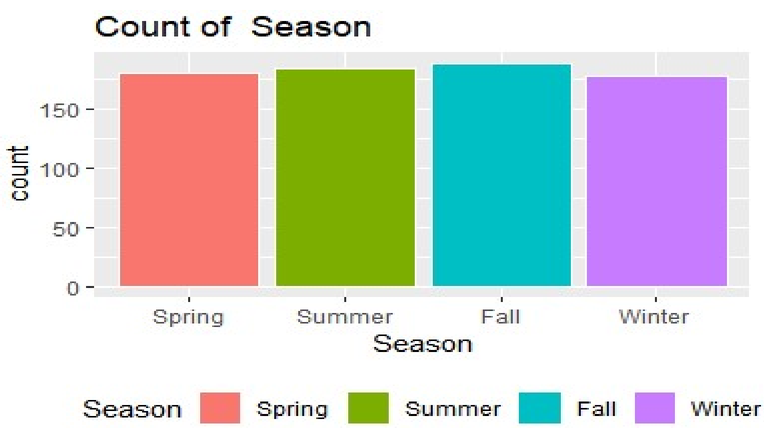


Fig. 2.2.3.4 Univariate analysis of Season

➢ **Weather**

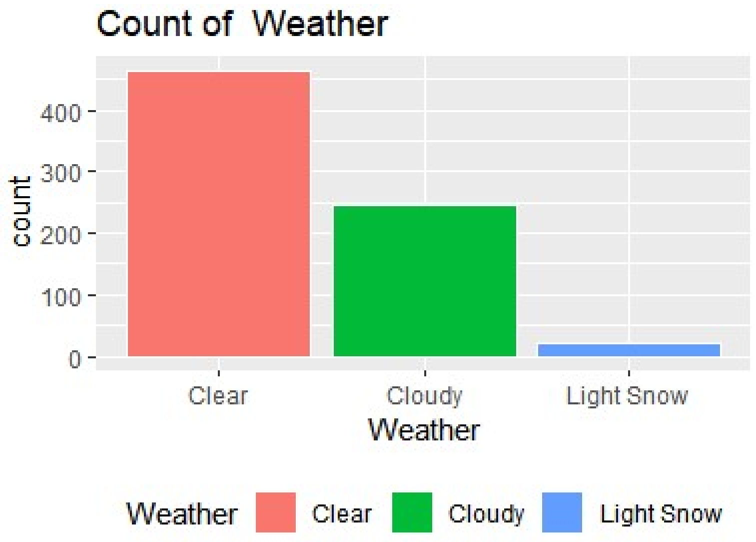


Fig. 2.2.3.5 Univariate analysis of Weather

➢ **Year**

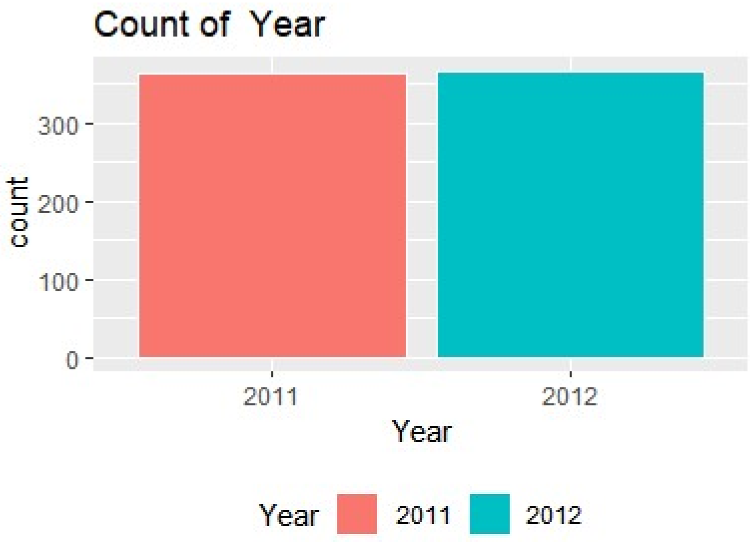


Fig. 2.2.3.6 Univariate analysis of Year

➢ **Month**

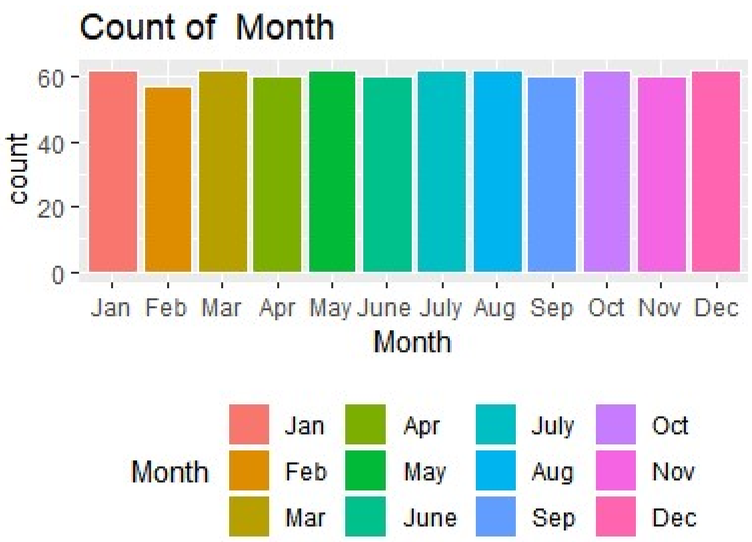


Fig. 2.2.3.7 Univariate analysis of Month

➢ **Weekday**

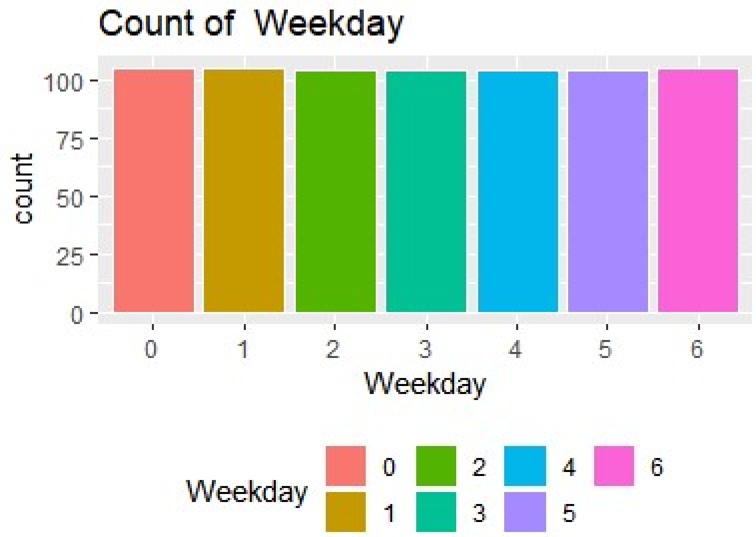


Fig. 2.2.3.8 Univariate analysis of Weekday

➢ **Workingday**

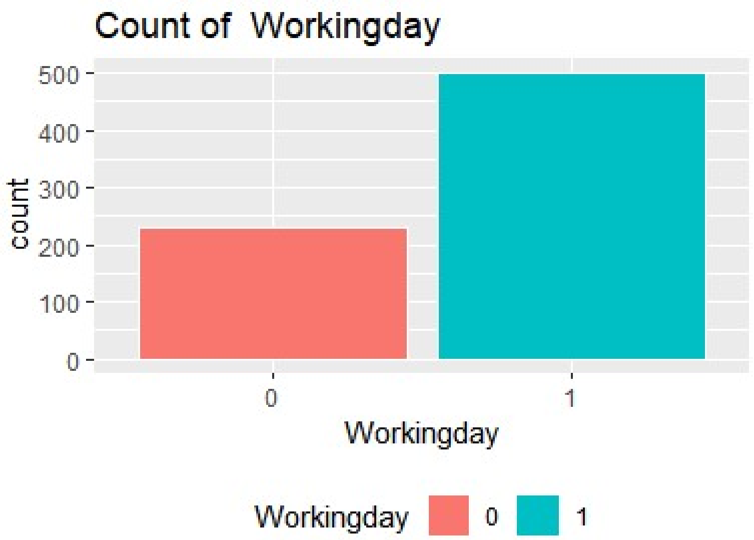


Fig. 2.2.3.9 Univariate analysis of Workingday

➢ **Holiday**

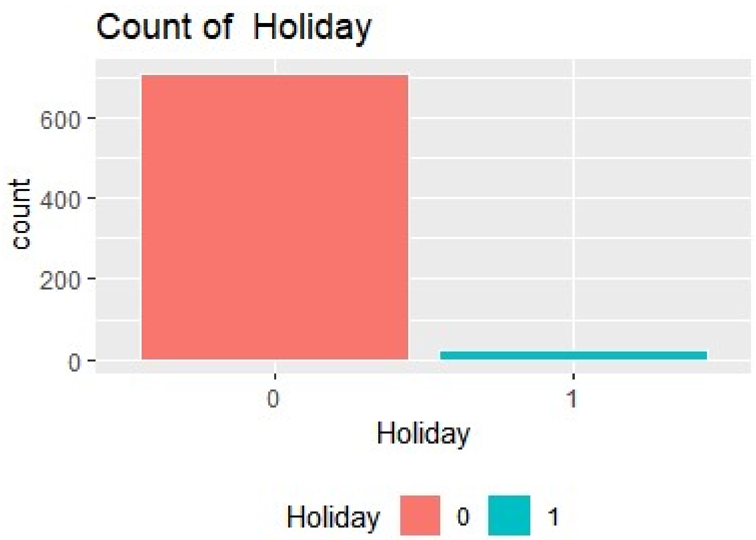


Fig. 2.2.3.10 Univariate analysis of Holiday

2.2.4 Bivariate Analysis

In bivariate analysis, we will look at the relationship between target variable and predictor. First we look for continuous variables.

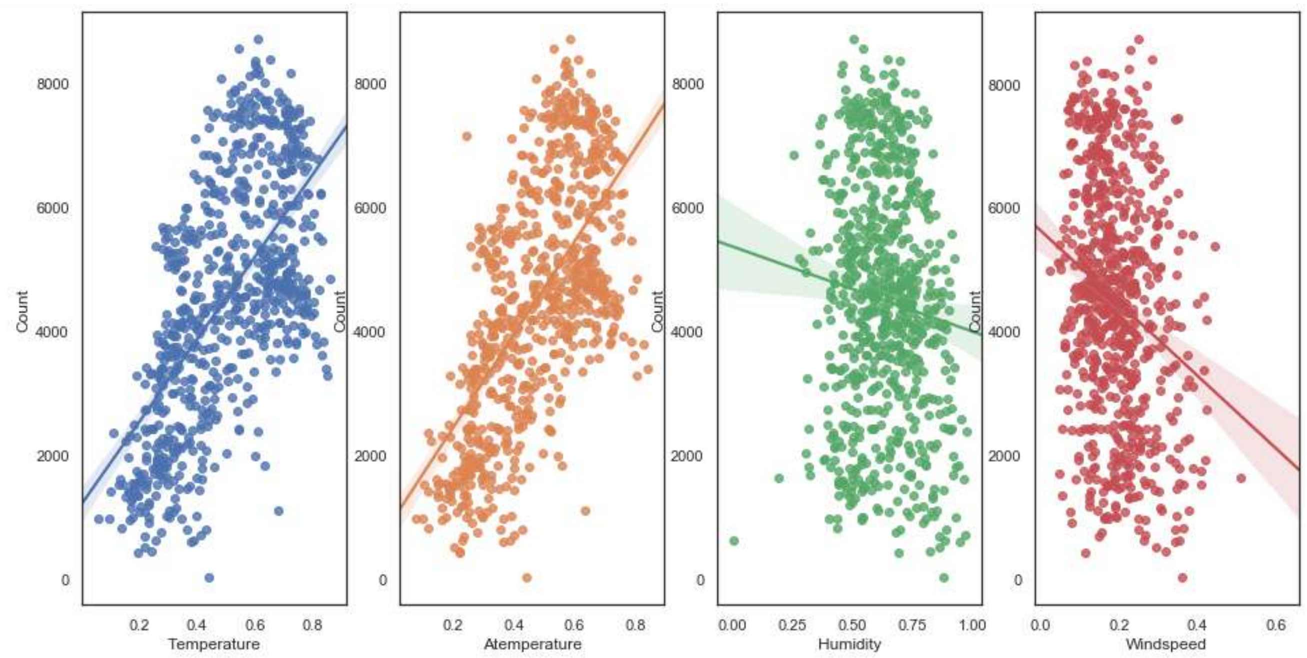


Fig 2.2.4.0. relationship between target variable and continuous predictors

From the above scatter plots, we can see that

- 'Count' and 'Temperature' have strong and positive relationship. It means that as the temperature rises, the bike demand also increase.
- 'Atemperature' and 'Count' have strong and positive relationship. It means that as the ambient temperature rise, demand for bikes also increases.
- Humidity' has a negative linear relationship with 'Count'. As humidity increases, count decreases. 'Windspeed' has negative linear relationship with 'Count'. With an increase in windspeed, bike count decreases.

➢ **Season vs Count**

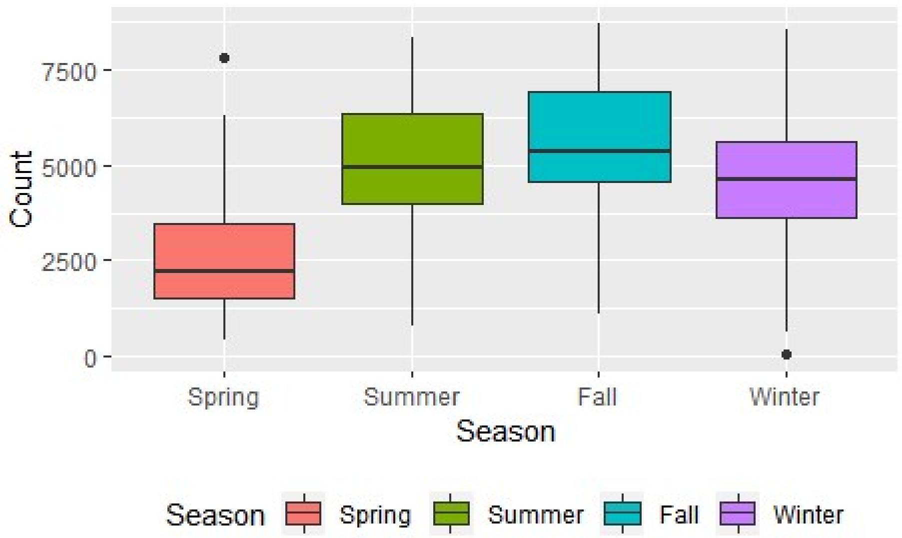


Fig. 2.2.4.1 Relation Between 'Season' and 'Count'

The above figure is showing relationship between count (demand) and season.

- count is highest for fall season and lowest for spring season.
- There is no significance difference between count for summer and fall.

➤ **Weather vs Count**

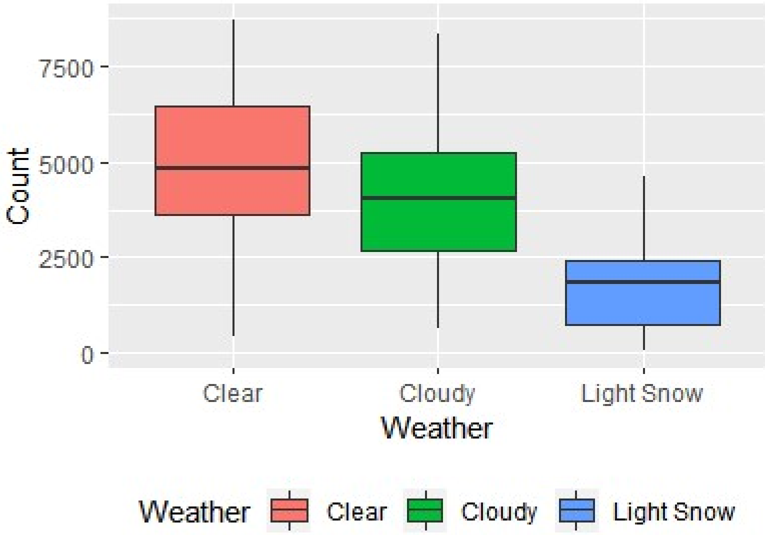


Fig. 2.2.4.2 Relation Between 'Weather' and 'Count'

- The count is maximum when weather situation is good.
- It is least when weather conditions are bad.

➤ **Year vs Count**

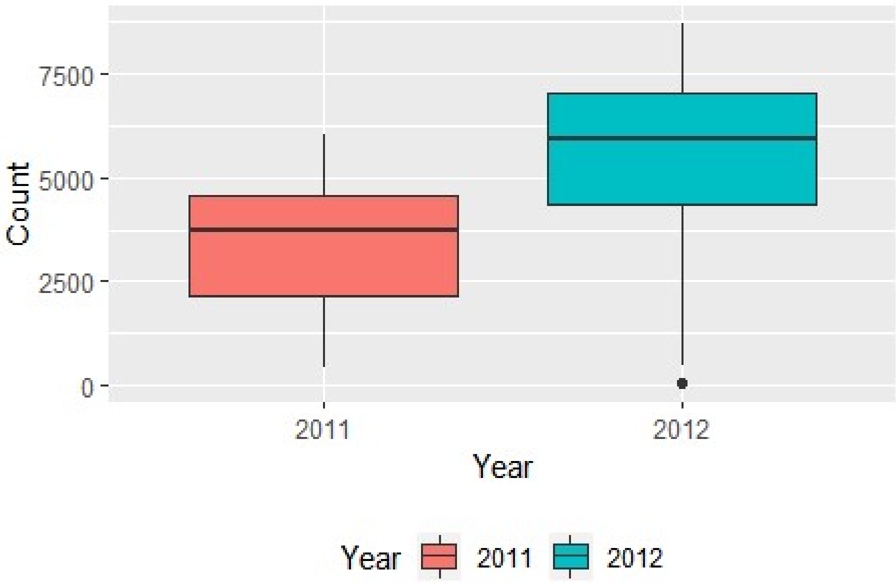


Fig. 2.2.4.3 Relation Between 'Year' and 'Count'

The above figure shows that bike demand was higher in 2012 as compared with 2011.

- **Month vs Count**

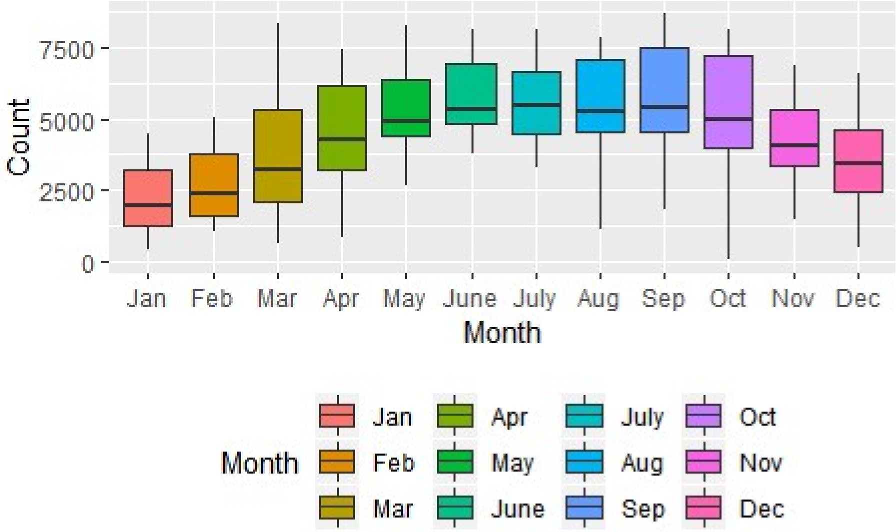


Fig. 2.2.4.4 Relation Between 'Year' and 'Count'

The above figure is showing relationship between count (demand) and Month.

- count is highest in the month of Aug, Sep and Oct.
- There is lowest count in Jan and Feb.

- **Weekday vs Count**

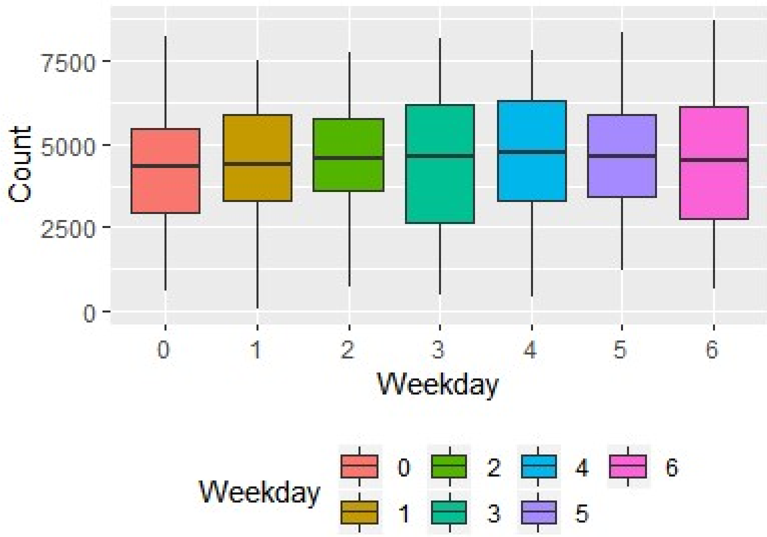


Fig. 2.2.4.5 Relation Between 'Year' and 'Count'

There is not much variation in median of count on weekdays. They are nearly similar on all weekdays.

➢ Workingday vs Count

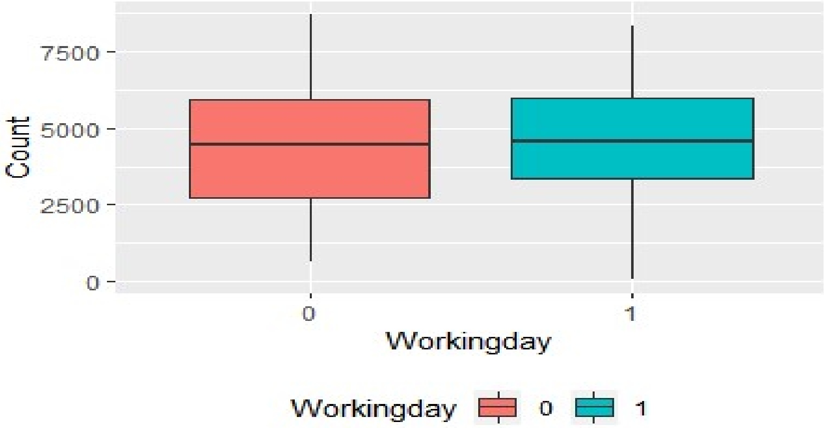


Fig. 2.2.4.6 Relation Between 'Year' and 'Count'

- There is median for count is same for working and non-working days.
- The range is longer for non-working days.

➢ Holiday vs Count

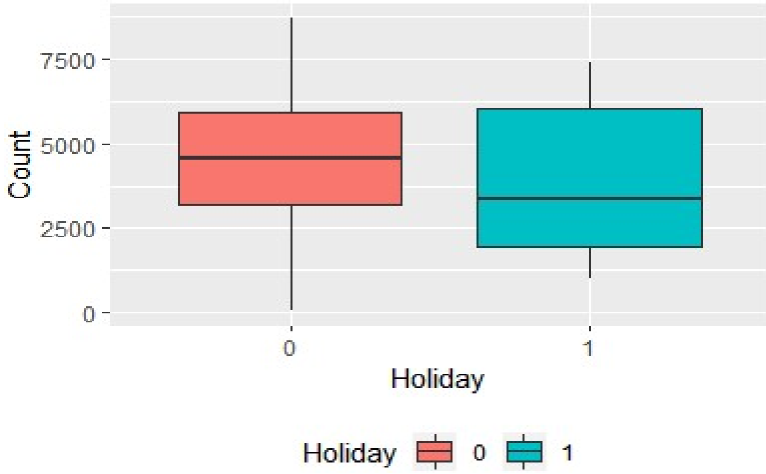


Fig. 2.2.4.7 Relation Between 'Year' and 'Count'

From the boxplot it is visible that count and it's median is higher on holidays. People prefer to rent bike on holidays.

Below are Some more illustrations

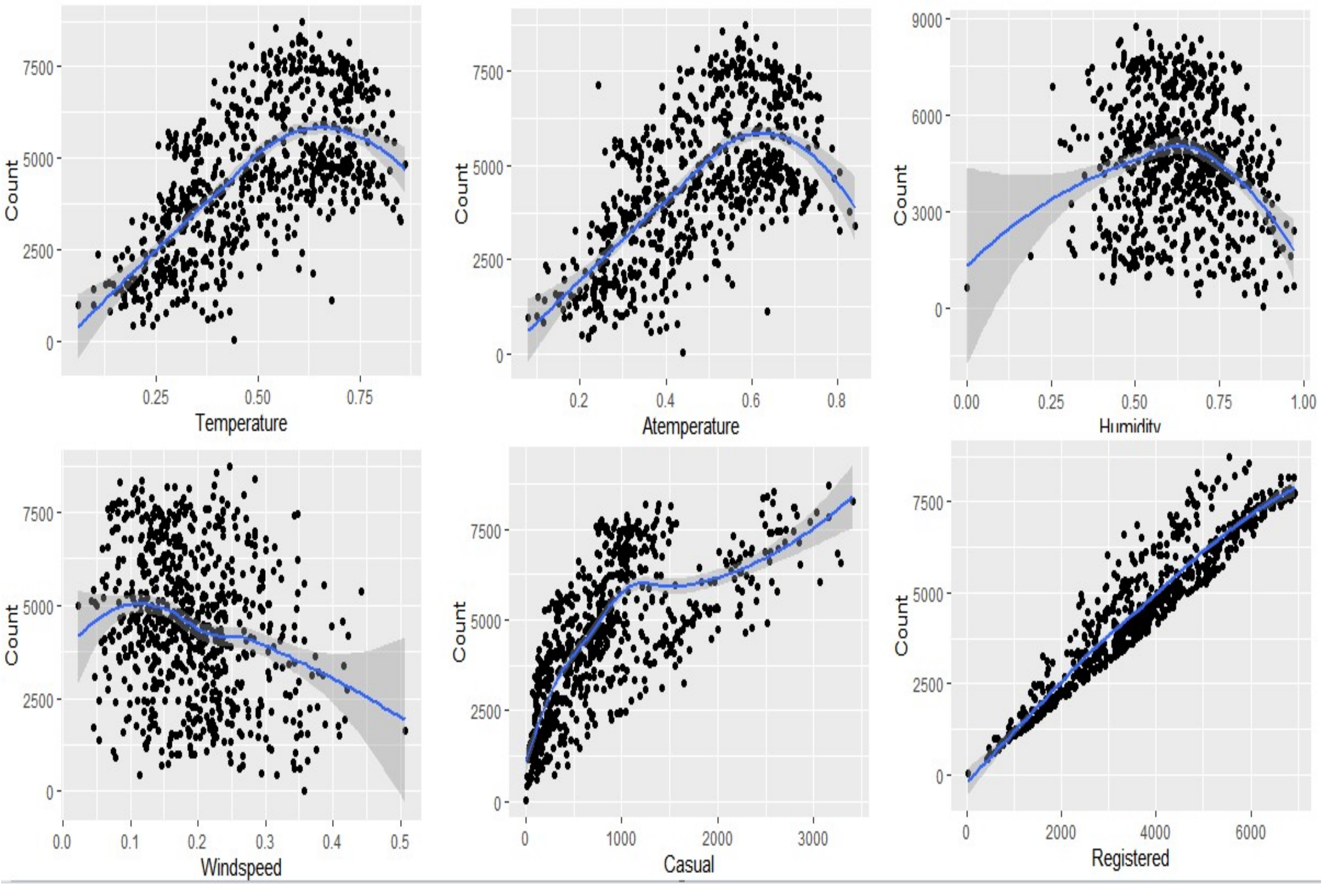


Fig. 2.2.4.8 Relation Between 'Variables' and 'Count'

2.2.5 Feature Scaling and Normalization

Data normalization is the process of rescaling one or more attributes to the range of [0, 1]. This means largest value of each attribute is 1 and smallest is 0. Normalization is a good technique to use when you know that your data distribution is not Gaussian.

Feature scaling was used in the R implementation using MLR package. It was not applied in python for the reason of performance comparison and the Scaled Data was shown below

	Season	Year	Month	Holiday	Weekday	Workingday	Weather	Temperature	Atemperature	Humidity	Windspeed	Casual	Registered	Count
1	Spring	2011	Jan	0	6	0	Cloudy	-0.82609651	-0.67948078	1.24931593	-0.38762628	-0.753218077	-1.9241532	985
2	Spring	2011	Jan	0	0	0	Cloudy	-0.72060131	-0.74014554	0.47878516	0.74908882	-1.044498954	-1.9138985	801
3	Spring	2011	Jan	0	1	1	Clear	-1.63353817	-1.74856976	-1.33835761	0.74612099	-1.060519402	-1.5556241	1349
4	Spring	2011	Jan	0	2	1	Clear	-1.61367485	-1.60916846	-0.26300148	-0.38956182	-1.077996254	-1.4114170	1562
5	Spring	2011	Jan	0	3	1	Clear	-1.46640988	-1.50394095	-1.34057625	-0.04627497	-1.115862768	-1.3703981	1600
6	Spring	2011	Jan	0	4	1	Clear	-1.58992191	-1.47976954	-0.76973783	-1.30224238	-1.107124342	-1.3703981	1606

Fig. 2.2.5.1 Future Scailing Data

2.3 Modeling

In bike renting case study, the target variable is continuous in nature. Our task is predicting the bike demand on a single day. This makes it a regression problem. Two machine learning algorithms were used for learning. Both were implemented in R and python.

1. Multivariate linear regression

2. Random forest regressor – an ensemble tree based regression

After EDA and pre-processing steps, data was divided into training and test dataset with 70 % and 30 % ratio. After modeling , diagnostic plots were used to check the assumptions of linear regression. For performance tuning of random forest, hyper parameter tuning was used.

2.3.1 Linear Regression

Linear regression is a technique in which we try to model a linear relationship with target and predictors. First linear regression was used.

- Data was divided into train and test.
- Linear regression was trained on training data.
- Backward and Forward elimination method was used on model with all predictors to select the best model.
- MAP and RMSE was used to check the performance of the model
- Prediction were done on the test data.

R Implementation:

First a model will all the predictors was trained in R. I.e. model1. Below is summary of model1.

```
model1 <- lm(Count ~ ., data = training_set)
> modelAIC <- stepAIC(model1, direction = "both")
Start:  AIC=6800.56
Count ~ Season + Year + Month + Holiday + Weekday + Workingday +
    Weather + Temperature + Atemperature + Humidity + Windspeed


Step:  AIC=6800.56
Count ~ Season + Year + Month + Holiday + Weekday + Weather +
    Temperature + Atemperature + Humidity + Windspeed

               Df Sum of Sq       RSS    AIC
- Atemperature  1    604568 275309771 6799.7
<none>                      274705203 6800.6
- Temperature   1   1680528 276385731 6801.7
- Holiday       1   2541499 277246702 6803.3
- Weekday       6   8931824 283637028 6804.9
- Humidity      1   8547280 283252484 6814.2
- Windspeed     1  15775500 290480703 6827.1
- Month        11  38792108 313497311 6846.1
- Weather       2  43451067 318156270 6871.6
- Season        3  45689214 320394417 6873.2
- Year          1 507678731 782383934 7333.4

Step:  AIC=6799.69
Count ~ Season + Year + Month + Holiday + Weekday + Weather +
    Temperature + Humidity + Windspeed
```

```
    Temperature + Humidity + Windspeed

                Df Sum of Sq       RSS    AIC
<none>                       275309771 6799.7
+ Atemperature   1    604568 274705203 6800.6
- Holiday        1   2726121 278035892 6802.7
- Weekday        6   8679904 283989675 6803.5
- Humidity       1   8284810 283594581 6812.8
- Windspeed      1  17582336 292892107 6829.3
- Month         11  38214582 313524353 6844.1
- Temperature    1  35748724 311058495 6860.1
- Weather        2  44428926 319738697 6872.1
- Season         3  45830789 321140560 6872.4
- Year           1 507074640 782384411 7331.4
> summary(modelAIC)

Call:
lm(formula = Count ~ Season + Year + Month + Holiday + Weekday +
    Weather + Temperature + Humidity + Windspeed, data = training_set)

Residuals:
    Min      1Q  Median      3Q     Max
-3479.9  -351.7    71.3   425.4  2418.5

Coefficients:
                  Estimate Std. Error t value Pr(>|t|)
(Intercept)        1514.61     293.24   5.165 3.52e-07 ***
SeasonSummer       1058.45     199.47   5.306 1.71e-07 ***
SeasonFall         1092.89     243.02   4.497 8.63e-06 ***
SeasonWinter       1740.57     209.33   8.315 9.41e-16 ***
Year2012           2054.43      68.88  29.826  < 2e-16 ***
MonthFeb            211.07     170.44   1.238 0.216161
MonthMar            505.08     195.72   2.581 0.010158 *
MonthApr            471.39     284.54   1.657 0.098240 .
MonthMay            897.34     310.55   2.889 0.004032 **
MonthJune           667.54     329.89   2.024 0.043568 *
MonthJuly            53.63     371.28   0.144 0.885217
MonthAug            488.20     357.27   1.366 0.172427
```

```
MonthSep             928.93     309.64   3.000 0.002839 **
MonthOct             612.68     285.72   2.144 0.032506 *
MonthNov             -71.15     268.27  -0.265 0.790960
MonthDec            -144.34     210.37  -0.686 0.492969
Holiday1            -493.88     225.83  -2.187 0.029226 *
Weekday1              84.97     132.12   0.643 0.520427
Weekday2             212.54     127.53   1.667 0.096254 .
Weekday3             344.98     126.02   2.738 0.006417 **
Weekday4             302.66     125.41   2.413 0.016180 *
Weekday5             365.09     125.24   2.915 0.003721 **
Weekday6             339.40     124.79   2.720 0.006767 **
WeatherCloudy       -412.23      94.14  -4.379 1.46e-05 ***
WeatherLight Snow -2059.08     234.47  -8.782  < 2e-16 ***
Temperature         3986.92     503.44   7.919 1.65e-14 ***
Humidity           -1398.37     366.79  -3.812 0.000155 ***
Windspeed          -2708.02     487.59  -5.554 4.62e-08 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 755 on 483 degrees of freedom
Multiple R-squared:  0.8558,    Adjusted R-squared:  0.8477
F-statistic: 106.1 on 27 and 483 DF,  p-value: < 2.2e-16
```

Fig. 2.3.1.0 Model 1 Summary

model1 all the predictors were included. Temp and atemp were multicollinear. They were also included in the model. The adjusted r square value was 0 .85 which is a good value with F-statistic 106.1.

After model1, step wise model selection was performed. Both forward and backward elimination technique were applied. The second models summary is given below.

```
> model2 <- lm(log(Count)~., data = training_set)
> stepwiseLogAICModel <- stepAIC(model2,direction = "both")
Start:  AIC=-1172.01
log(Count) ~ Season + Year + Month + Holiday + Weekday + Workingday +
    Weather + Temperature + Atemperature + Humidity + Windspeed
Step:  AIC=-1172.01
log(Count) ~ Season + Year + Month + Holiday + Weekday + Weather +
    Temperature + Atemperature + Humidity + Windspeed
               Df Sum of Sq    RSS     AIC
- Weekday       6    0.6975 46.728 -1176.32
- Atemperature  1    0.0220 46.053 -1173.77
<none>                      46.031 -1172.01
- Holiday       1    0.3205 46.351 -1170.46
- Temperature   1    0.4928 46.523 -1168.57
- Month        11    2.8682 48.899 -1163.12
- Humidity      1    1.3827 47.413 -1158.89
- Windspeed     1    2.0611 48.092 -1151.63
- Season        3    5.9065 51.937 -1116.32
- Weather       2    9.1973 55.228 -1082.92
- Year          1   24.7937 70.824  -953.82
Step:  AIC=-1176.32
log(Count) ~ Season + Year + Month + Holiday + Weather + Temperature +
    Atemperature + Humidity + Windspeed
               Df Sum of Sq    RSS     AIC
- Atemperature  1    0.0075 46.736 -1178.24
<none>                      46.728 -1176.32
+ Workingday    1    0.1100 46.618 -1175.53
- Holiday       1    0.5013 47.229 -1172.87
+ Weekday       6    0.6975 46.031 -1172.01
- Temperature   1    0.6271 47.355 -1171.51
- Month        11    2.8524 49.581 -1168.05
- Humidity      1    1.5565 48.285 -1161.58
- Windspeed     1    2.1192 48.847 -1155.66
- Season        3    5.9384 52.667 -1121.19
- Weather       2    9.1419 55.870 -1089.02
- Year          1   24.9092 71.637  -959.99
Step:  AIC=-1178.24
log(Count) ~ Season + Year + Month + Holiday + Weather + Temperature +
    Humidity + Windspeed
               Df Sum of Sq    RSS     AIC
<none>                      46.736 -1178.24
+ Workingday    1    0.1082 46.627 -1177.43
+ Atemperature  1    0.0075 46.728 -1176.32
- Holiday       1    0.5106 47.246 -1174.69
+ Weekday       6    0.6830 46.053 -1173.77
- Month        11    2.8514 49.587 -1169.98
- Humidity      1    1.5490 48.285 -1163.58
- Windspeed     1    2.2438 48.979 -1156.28
- Season        3    5.9438 52.679 -1123.07
- Temperature   1    6.7043 53.440 -1111.74
- Weather       2    9.2252 55.961 -1090.19
- Year          1   24.9068 71.642  -961.95
```

Fig. 2.3.1.1 Model 2 Summary

Python Implementation :

In python a single regression model was trained after all pre-processing. Python don't have step wise regression implementation. Same log transformation was performed to avoid negative prediction.

Linear Regression Model ¶

```
In [49]: X = training_set
         X = sm.add_constant(X)
         y= np.log(train_target)

         model = sm.OLS(y, X.astype(float)).fit()
```

```
In [48]: model.summary()
```

Out[48]:

OLS Regression Results

Dep. Variable:	y	R-squared:	0.654
Model:	OLS	Adj. R-squared:	0.647
Method:	Least Squares	F-statistic:	94.40
Date:	Sun, 11 Aug 2019	Prob (F-statistic):	2.20e-108
Time:	01:08:02	Log-Likelihood:	-184.70
No. Observations:	511	AIC:	391.4
Df Residuals:	500	BIC:	438.0
Df Model:	10		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	7.6112	0.110	69.224	0.000	7.395	7.827
Season	0.1286	0.026	4.865	0.000	0.077	0.180
Year	0.4818	0.031	15.339	0.000	0.420	0.543
Month	-0.0069	0.008	-0.834	0.405	-0.023	0.009
Holiday	-0.1800	0.099	-1.815	0.070	-0.375	0.015
Weekday	0.0133	0.008	1.670	0.095	-0.002	0.029
Workingday	0.0577	0.035	1.655	0.099	-0.011	0.126
Weather	-0.2331	0.037	-6.228	0.000	-0.307	-0.160
Temperature	1.5244	0.094	16.269	0.000	1.340	1.708
Humidity	-0.2495	0.149	-1.677	0.094	-0.542	0.043
Windspeed	-1.0399	0.218	-4.760	0.000	-1.469	-0.611

Omnibus:	654.553	Durbin-Watson:	2.039
Prob(Omnibus):	0.000	Jarque-Bera (JB):	128684.713
Skew:	-6.061	Prob(JB):	0.00
Kurtosis:	79.792	Cond. No.	131.

Fig. 2.3.1.2 Linear Regression – Python

2.2.2 Random Forest Regression

After linear regression, random forest was trained. It was implemented in both R and python. After training with default setting, hyper parameter tuning was used for increase performance.

R Implementation

```
Hit <Return> to see next plot: # ----------------- Model 2 Random forest -----------------------------------------------------#
Hit <Return> to see next plot:
Hit <Return> to see next plot: model1 <- randomForest(Count ~.,
Hit <Return> to see next plot:                          data = training_set,ntree = 500, mtry = 8, importance = TRUE)
> print(model1)

Call:
lm(formula = Count ~ ., data = training_set)

Coefficients:
      (Intercept)      SeasonSummer        SeasonFall      SeasonWinter          Year2012
          1465.98           1052.92           1090.08           1739.77           2056.81
         MonthFeb          MonthMar          MonthApr          MonthMay         MonthJune
           207.30            505.95            468.06            914.46            695.86
        MonthJuly          MonthAug          MonthSep          MonthOct          MonthNov
            74.15            537.96            954.10            611.19            -77.42
         MonthDec          Holiday1          Weekday1          Weekday2          Weekday3
          -149.70           -477.98             84.20            216.58            349.06
         Weekday4          Weekday5          Weekday6       Workingday1      WeatherCloudy
           304.14            374.08            342.14                NA           -409.63
WeatherLight Snow       Temperature      Atemperature          Humidity         Windspeed
         -2041.38           2548.79           1571.57          -1423.48          -2611.79

> par(mfrow = c(1,1))
> plot(model1)
Hit <Return> to see next plot:
Hit <Return> to see next plot:
Hit <Return> to see next plot: # 300 trees selected from the plot
Hit <Return> to see next plot:
> tumedmodel <- tuneRF(training_set[,1:11], training_set[,12], stepFactor = 0.5, plot = TRUE,
+                        ntreeTry = 250, trace = TRUE, improve = 0.05)
mtry = 3  OOB error = 482840.4
Searching left ...
mtry = 6    OOB error = 450199.4
0.06760194 0.05
mtry = 12   OOB error = 460451.7
-0.0227728 0.05
```

```
Searching right ...
mtry = 1    OOB error = 915072.8
-1.032594 0.05
Warning message:
In randomForest.default(x, y, mtry = mtryCur, ntree = ntreeTry,  :
  invalid mtry: reset to within valid range
>
> # selected mtry = 6 from the plot
>
> tuned_randomForest <-  randomForest(Count ~. - Atemperature,
+                                      data = training_set,ntree = 250, mtry = 6, importance = TRUE)
> tuned_randomForest

Call:
 randomForest(formula = Count ~ . - Atemperature, data = training_set,      ntree = 250, mtry = 6, importance = TRUE)
               Type of random forest: regression
                     Number of trees: 250
No. of variables tried at each split: 6

          Mean of squared residuals: 460970
                    % Var explained: 87.66
> # predicting using random forest model 1
> rf1_prediction <- predict(tuned_randomForest,test_set[,-12])
> rmse(rf1_prediction,test_set$Count)
[1] 749.583
> print(paste("Mean Absolute Error for Random forest regressor  is ",
+             MAE(test_set$Count,rf1_prediction)))
[1] "Mean Absolute Error for Random forest regressor  is  501.871369582841"
```

Python Implementation

In python random forest was trained and hyper parameters optimisation was done using following parameters. Default setting of random forest are given below.

Random Forest Model

```
In [99]: rf = RandomForestRegressor(random_state=12345)
         rf

Out[99]: RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=None,
                               max_features='auto', max_leaf_nodes=None,
                               min_impurity_decrease=0.0, min_impurity_split=None,
                               min_samples_leaf=1, min_samples_split=2,
                               min_weight_fraction_leaf=0.0, n_estimators='warn',
                               n_jobs=None, oob_score=False, random_state=12345,
                               verbose=0, warm_start=False)
```

Tuned parameters were selected with hyper tuning random grid search. Best parameters selected were as follows

In [102]:
```
np.random.seed(12)
start = time.time()

# selecting best max_depth, maximum features, split criterion and number of trees
parameter_dist = {'max_depth': [2,4,6,8,10],
              'bootstrap': [True, False],
              'max_features': ['auto', 'sqrt', 'log2',None],
              "n_estimators" : [100 ,200 ,300 ,400 ,500]
             }
RandomForest = RandomizedSearchCV(rf, cv = 10,
                     param_distributions = parameter_dist,
                     n_iter = 10)

RandomForest.fit(training_set, train_target)
print('Best Parameters using random search: \n',
      RandomForest.best_params_)
end = time.time()
print('Time taken in random search: {0: .2f}'.format(end - start))
```

```
Best Parameters using random search:
 {'n_estimators': 300, 'max_features': 'log2', 'max_depth': 8, 'bootstrap': False}
Time taken in random search:  64.67
```

Using above mention parameters, random forest regressor was trained.

In [103]:
```
# setting parameters

# Set best parameters given by random search # Set be
rf.set_params( max_features = 'log2',
               max_depth =8 ,
               n_estimators = 300
                )
```

Out[103]:
```
RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=8,
                      max_features='log2', max_leaf_nodes=None,
                      min_impurity_decrease=0.0, min_impurity_split=None,
                      min_samples_leaf=1, min_samples_split=2,
                      min_weight_fraction_leaf=0.0, n_estimators=300,
                      n_jobs=None, oob_score=False, random_state=12345,
                      verbose=0, warm_start=False)
```

In [105]:
```
rf.fit(training_set, train_target)
```

Out[105]:
```
RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=8,
                      max_features='log2', max_leaf_nodes=None,
                      min_impurity_decrease=0.0, min_impurity_split=None,
                      min_samples_leaf=1, min_samples_split=2,
                      min_weight_fraction_leaf=0.0, n_estimators=300,
                      n_jobs=None, oob_score=False, random_state=12345,
                      verbose=0, warm_start=False)
```

In [106]:
```
# Use the forest's predict method on the test data
rfPredictions = rf.predict(test_set)
# Calculate the absolute errors
rf_errors = abs(rfPredictions - test_target)
# Print out the mean absolute error (mae)
print('Mean Absolute Error:', round(np.mean(rf_errors), 2), 'degrees.')
```

```
Mean Absolute Error: 495.28 degrees.
```

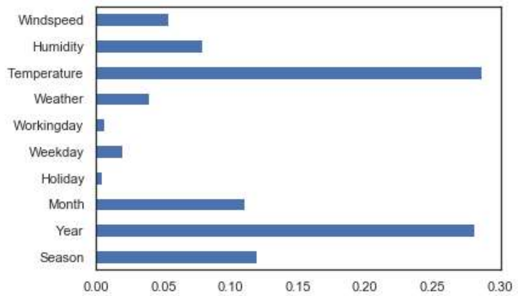
In [107]:
```
rmse_rf = sqrt(mean_squared_error(test_target, rfPredictions))
print("RMSE for test set in random forest regressor  is :" , rmse_rf)
```

```
RMSE for test set in random forest regressor  is : 649.6911207838448
```

Variable importance using random forest in python.

In [108]:
```
feature_importance =  pd.Series(rf.feature_importances_, index=training_set.columns)
feature_importance.plot(kind='barh')
```

Out[108]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x284b2f09048>
```



Result and Performance measure

RM SE (root mean square error) and MAE (mean absolute error) were used as error metric and measuring model performance.

3.1 Performance Measure

R implementation

For measuring rmse, Metric package was used. For measuring MAE, a function was written. The values for both the metric for linear regression and random forest are as follow.

Error metric / Algorithm	Linear Regression	Random Forest
RMSE	821.37	749.58
MAE	696.18	501.87

As from the table we can see that random forest performing better than linear regression on both the error metric.

Python implementation

In python, both the error metric was calculated using python functions. No pre-built package or modules were used. The values for both metric are given below.

Error metric / Algorithm	Linear Regression	Random Forest
RMSE	1222.15	649.69
MAE	899.5	495.28

As we can see random forest performing better than linear regression.

Result / Conclusion

From the error metric we can see that random forest is performing better than linear regression in both implementation s. The result for random forest is similar in both R and python. But in case of linear regression, R's implementation is performing better than python. The difference here is that data in R was normalized before regression.

Selection of model depends on use case. If we want to study the effects of predictors in details, we will go for linear regression and look at the regression equation. If we are care about more precise prediction, we will opt for random forest.

R Code Implementation

```
#Clear Environment-
rm(list=ls())

library(corrplot)
library(ggplot2)
library(dplyr)
library(rcompanion)
library(mlr)
library(caTools)
library(MASS)
library(Metrics)
library(randomForest)

#Set working directory-
setwd("F:/Edvisor Project/Bike_Rental")

#Check working directory-
getwd()

#load data-
bikedata= read.csv("day.csv")

#------------------------------Exploratory Data Analysis------------------------------------------#
class(bikedata)
dim(bikedata)
head(bikedata)
names(bikedata)
str(bikedata)
summary(bikedata)

#Remove the instant variable, as it is index in dataset.
bikedata= subset(bikedata,select=-(instant))

#Remove date variable as we have to predict count on seasonal basis not date basis-
bikedata= subset(bikedata,select=-(dteday))

```

```
#check the remaining variables-
names(bikedata)

#Rename the variables-
names(bikedata)[1]="Season"
names(bikedata)[2]="Year"
names(bikedata)[3]="Month"
names(bikedata)[4]="Holiday"
names(bikedata)[5]="Weekday"
names(bikedata)[6]="Workingday"
names(bikedata)[7]="Weather"
names(bikedata)[8]="Temperature"
names(bikedata)[9]="Atemperature"
names(bikedata)[10]="Humidity"
names(bikedata)[11]="Windspeed"
names(bikedata)[12]="Casual"
names(bikedata)[13]="Registered"
names(bikedata)[14]="Count"


#Seperate categorical and numeric variables-
names(bikedata)

#numeric variables-
cnames= c("Temperature","Atemperature","Humidity","Windspeed","Count")

#categorical varibles-
cat_cnames= c("Season","Year","Month","Holiday","Weekday","Workingday","Weather")
str(bikedata)

```

```
#=================================Data Pre-processing=========================================#

#--------------------------------Missing Vlaue Analysis----------------------------------------#
#Check missing values in dataset-
sum(is.na(bikedata))
#Missing value= 0
#No Missing values in data.
```

```
#convering categorical variables into factor

bikedata$Season <- as.factor(bikedata$Season)
levels(bikedata$Season)[levels(bikedata$Season) == 1] <- 'Spring'
levels(bikedata$Season)[levels(bikedata$Season) == 2] <- 'Summer'
levels(bikedata$Season)[levels(bikedata$Season) == 3] <- 'Fall'
levels(bikedata$Season)[levels(bikedata$Season) == 4] <- 'Winter'
```

```

bikedata$Month <- as.factor(bikedata$Month)
levels(bikedata$Month)[levels(bikedata$Month) == 1] <- 'Jan'
levels(bikedata$Month)[levels(bikedata$Month) == 2] <- 'Feb'
levels(bikedata$Month)[levels(bikedata$Month) == 3] <- 'Mar'
levels(bikedata$Month)[levels(bikedata$Month) == 4] <- 'Apr'
levels(bikedata$Month)[levels(bikedata$Month) == 5] <- 'May'
levels(bikedata$Month)[levels(bikedata$Month) == 6] <- 'June'
levels(bikedata$Month)[levels(bikedata$Month) == 7] <- 'July'
levels(bikedata$Month)[levels(bikedata$Month) == 8] <- 'Aug'
levels(bikedata$Month)[levels(bikedata$Month) == 9] <- 'Sep'
levels(bikedata$Month)[levels(bikedata$Month) == 10] <- 'Oct'
levels(bikedata$Month)[levels(bikedata$Month) == 11] <- 'Nov'
levels(bikedata$Month)[levels(bikedata$Month) == 12] <- 'Dec'

bikedata$Year <- as.factor(bikedata$Year)
levels(bikedata$Year)[levels(bikedata$Year) == 0] <- '2011'
levels(bikedata$Year)[levels(bikedata$Year) == 1] <- '2012'

bikedata$Holiday <- as.factor(bikedata$Holiday)
bikedata$Weekday <- as.factor(bikedata$Weekday)
bikedata$Workingday <- as.factor(bikedata$Workingday)

bikedata$Weather <- as.factor(bikedata$Weather)
levels(bikedata$Weather)[levels(bikedata$Weather) == 1] <-'Clear'
levels(bikedata$Weather)[levels(bikedata$Weather) == 2] <-'Cloudy'
levels(bikedata$Weather)[levels(bikedata$Weather) == 3] <-'Light Snow'
levels(bikedata$Weather)[levels(bikedata$Weather) == 4] <-' Heavy Rain'

```

```
#-----------------------------------Outlier Analysis---------------------------------------------#


#create Box-Plot for outlier analysis-

outlierKD <- function(dt, var) {
  var_name <- eval(substitute(var), eval(dt))
  na1 <- sum(is.na(var_name))
  m1 <- mean(var_name, na.rm = T)
  par(mfrow = c(1, 2), oma = c(0, 0, 3, 0))
  boxplot(var_name, main = "With outliers")
  hist(var_name,
       main = "With outliers",
       xlab = NA,
       ylab = NA)
  outlier <- boxplot.stats(var_name)$out
  mo <- mean(outlier)
  var_name <- ifelse(var_name %in% outlier, NA, var_name)
  boxplot(var_name, main = "Without outliers")
  hist(var_name,
       main = "Without outliers",
       xlab = NA,
       ylab = NA)
  title("Outlier Check", outer = TRUE)
  na2 <- sum(is.na(var_name))
  cat("Outliers identified:", na2 - na1, "n")
  cat("Propotion (%) of outliers:", round((na2 - na1) / sum(!is.na(var_name)) *
                                          100, 1), "n")
  cat("Mean of the outliers:", round(mo, 2), "n")
  m2 <- mean(var_name, na.rm = T)
  cat("Mean without removing outliers:", round(m1, 2), "n")
  cat("Mean if we remove outliers:", round(m2, 2), "n")

}

```

```


outlierKD(bikedata, Temperature) #no outliers
outlierKD(bikedata, Atemperature) #no outliers
outlierKD(bikedata, Humidity) # no extreme outlier detected
outlierKD(bikedata, Windspeed) #some extreme values are present but canot be considered as outlier
outlierKD(bikedata, Casual) # no logical outliers
outlierKD(bikedata, Registered)# no ouliers
outlierKD(bikedata, Count)# no ouliers


```

```r
#-----------------------------------------------------------------------------------------#
#                                                                                         #
#                              Correlation Analysis                                       #
#                                                                                         #
#-----------------------------------------------------------------------------------------#
par(mfrow = c(1, 1))
numeric_predictors <- unlist(lapply(bikedata, is.numeric))
numVarDataset <- bikedata[, numeric_predictors]
corr <- cor(numVarDataset)
corrplot(
  corr,
  method = "color",
  outline = TRUE,
  cl.pos = 'n',
  rect.col = "black",
  tl.col = "indianred4",
  addCoef.col = "black",
  number.digits = 2,
  number.cex = 0.60,
  tl.cex = 0.70,
  cl.cex = 1,
  col = colorRampPalette(c("green4", "white", "red"))(100)
)

# Findings :
# 1. temp and atemp are highly correlated

# Looking at target variable
ggplot(data = bikedata, aes(Count)) +
  geom_histogram(aes(
    y = ..density..,
    binwidth = .10,
    colour = "black"
  ))
# Target variable looks like normal distribution

```

```r
#-----------------------------------------------------------------------------------------#
#                                                                                         #
#                              Univariate Analysis                                        #
#                                                                                         #
#-----------------------------------------------------------------------------------------#
# 1. Continous predictors
univariate_continuous <- function(dataset, variable, variableName) {
  var_name = eval(substitute(variable), eval(dataset))
  print(summary(var_name))
  ggplot(data = dataset, aes(var_name)) +
    geom_histogram(aes(binwidth = .8, colour = "black")) +
    labs(x = variableName) +
    ggtitle(paste("count of", variableName))
}

univariate_continuous(bikedata, Count, "Count")
univariate_continuous(bikedata, Temperature, "Temperature")
univariate_continuous(bikedata, Atemperature, "Atemperature")
univariate_continuous(bikedata, Humidity, "Humidity") # skwed towards left
univariate_continuous(bikedata, Windspeed, "Windspeed") #skewed towards right
univariate_continuous(bikedata, Casual, "Casual") # skwed towards right
univariate_continuous(bikedata, Registered, "Registered")

#2. categorical variables
univariate_categorical  <- function(dataset, variable, variableName) {
  variable <- enquo(variable)

  percentage <- dataset %>%
    dplyr::select(!!variable) %>%
    group_by(!!variable) %>%
    summarise(n = n()) %>%
    mutate(percantage = (n / sum(n)) * 100)
  print(percentage)

```

```
  dataset %>%
    count(!!variable) %>%
    ggplot(mapping = aes_(
      x = rlang::quo_expr(variable),
      y = quote(n),
      fill = rlang::quo_expr(variable)
    )) +
    geom_bar(stat = 'identity',
             colour = 'white') +
    labs(x = variableName, y = "count") +
    ggtitle(paste("count of ", variableName)) +
    theme(legend.position = "bottom") -> p
  plot(p)
}

univariate_categorical(bikedata, Season, 'Season')
univariate_categorical(bikedata, Year, "Year")
univariate_categorical(bikedata, Month, "Month")
univariate_categorical(bikedata, Holiday, "Holiday")
univariate_categorical(bikedata, Weekday, "Weekday")
univariate_categorical(bikedata, Workingday, "Workingday")
univariate_categorical(bikedata, Weather, "Weather")

```

```
# ------------------------------------------------------------------------------------------------ #
#
#                                      bivariate Analysis
#
#------------------------------------------------------------------------------------------------- #

# bivariate analysis for categorical variables
bivariate_categorical <-
  function(dataset, variable, targetVariable) {
    variable <- enquo(variable)
    targetVariable <- enquo(targetVariable)

    ggplot(
      data = dataset,
      mapping = aes_(
        x = rlang::quo_expr(variable),
        y = rlang::quo_expr(targetVariable),
        fill = rlang::quo_expr(variable)
      )
    ) +
      geom_boxplot() +
      theme(legend.position = "bottom") -> p
    plot(p)

  }

bivariate_continous <-
  function(dataset, variable, targetVariable) {
    variable <- enquo(variable)
    targetVariable <- enquo(targetVariable)
    ggplot(data = dataset,
           mapping = aes_(
             x = rlang::quo_expr(variable),
             y = rlang::quo_expr(targetVariable)
           )) +
      geom_point() +
```

```
      geom_smooth() -> q
    plot(q)

  }

bivariate_categorical(bikedata, Season, Count)
bivariate_categorical(bikedata, Year, Count)
bivariate_categorical(bikedata, Month, Count)
bivariate_categorical(bikedata, Holiday, Count)
bivariate_categorical(bikedata, Weekday, Count)
bivariate_categorical(bikedata, Workingday, Count)
bivariate_categorical(bikedata, Weather, Count)

bivariate_continous(bikedata, Temperature, Count)
bivariate_continous(bikedata, Atemperature, Count)
bivariate_continous(bikedata, Humidity, Count)
bivariate_continous(bikedata, Windspeed, Count)
bivariate_continous(bikedata, Casual, Count)
bivariate_continous(bikedata, Registered, Count)

# removing instant and dteday
bikedata$instant <- NULL
bikedata$Date <- NULL
bikedata$Casual <- NULL
bikedata$Registered <- NULL
```

```r
# ------------------------------------------------------------------------------------------------ #
#
#                                     Feature scaling or Normalization                            #
#
#------------------------------------------------------------------------------------------------- #

scaledData <- normalizeFeatures(bikedata,'Count')

# Function for calculating Mean Absolute Error
MAE <- function(actual,predicted){
  error = actual - predicted
  mean(abs(error))
}

# ----------------- Model 1 Linear Regression -----------------------------------------------------#


set.seed(654)
split <- sample.split(bikedata$Count, SplitRatio = 0.70)
training_set <- subset(bikedata, split == TRUE)
test_set <- subset(bikedata, split == FALSE)


model1 <- lm(Count ~ ., data = training_set)

# step wise model selection

modelAIC <- stepAIC(model1, direction = "both")
summary(modelAIC)

# Apply prediction on test set
test_prediction <- predict(modelAIC, newdata = test_set)

test_rmse <- rmse(test_set$Count, test_prediction)
print(paste("root-mean-square error for linear regression model is ", test_rmse))
print(paste("Mean Absolute Error for linear regression model is ",MAE(test_set$Count,test_prediction)))
print("summary of predicted count values")
summary(test_prediction)
print("summary of actual Count values")
summary(test_set$Count)

# From the summary we can observe negative prediction values
#We will perform log transformation of trarget variable
model2 <- lm(log(Count)~., data = training_set)

stepwiseLogAICModel <- stepAIC(model2,direction = "both")
test_prediction_log<- predict(stepwiseLogAICModel, newdata = test_set)
predict_test_nonlog <- exp(test_prediction_log)

test_rmse2 <- rmse(test_set$Count, predict_test_nonlog)
print(paste("root-mean-square error between actual and predicted", test_rmse))
print(paste("Mean Absolute Error for linear regression model is ",
            MAE(test_set$Count,predict_test_nonlog)))

summary(predict_test_nonlog)
summary(test_set$Count)



par(mfrow = c(1,1))
plot(stepwiseLogAICModel)
```

```r
# ----------------- Model 2 Random forest -----------------------------------------------------#

model1 <- randomForest(Count ~.,
                           data = training_set,ntree = 500, mtry = 8, importance = TRUE)
print(model1)
par(mfrow = c(1,1))
plot(model1)


# 300 trees selected from the plot

tumedmodel <- tuneRF(training_set[,1:11], training_set[,12], stepFactor = 0.5, plot = TRUE,
                     ntreeTry = 250, trace = TRUE, improve = 0.05)

# selected mtry = 6 from the plot

tuned_randomForest <-  randomForest(Count ~. - Atemperature,
                                    data = training_set,ntree = 250, mtry = 6, importance = TRUE)
tuned_randomForest

# predicting using random forest model 1
rf1_prediction <- predict(tuned_randomForest,test_set[,-12])
rmse(rf1_prediction,test_set$Count)
print(paste("Mean Absolute Error for Random forest regressor  is ",
            MAE(test_set$Count,rf1_prediction)))

# Tuned Random Forest

varImpPlot(tuned_randomForest)

# Random forest is performing better than linear regression.

# Model input and output for linear regression and Random forest
write.csv(test_set, file = "InputLinearRegressionR.csv")
write.csv(test_set, file = "InputRandomForestR.csv")
write.csv(predict_test_nonlog, file="outputLogisticRegressionR.csv")
```

******************************End Of the Report **********************************