Predication of Bike Rentals

Vanusha Baregal
23rd October 2019

Table of Contents

Chapter 1

Introduction

1.1. Problem Statement

A bike rental business rents out bicycles for short periods of time provided mostly by bike shops as a side-line to their main businesses of sales and service and also by specialized shops in rentals. These rental shops primarily serve people who do not have access to a vehicle, typically travellers and particularly tourists. Specialized bicycle rental shops therefore typically operate at beaches, parks, or other locations that tourists frequent. These shops allow both registered and casual ('walk in') users to travel across cities, counties, and even to more remote destinations 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), Rental process is highly correlated to the environmental and seasonal settings. For instance, weather conditions, precipitation, day of week, season etc. can affect the rental behaviours. Data from the users is constantly being collected for analytics purposes to help prepare for a change in the demand of bike rental from their users. Thus, having an estimate on the demands would enable efficient functioning of these companies The objective of this Case is Predication of bike rental count on daily based on the environmental and seasonal settings.

1.2. Data

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. Given below is a sample of the data set that we are using to predict the bike rental counts:

Table 1.1: Bike Renting Sample Data (Columns: 1-5)

Instant	Date	Season	Year	Month	Holiday	Weekday	Workingday
1	01-01-2011	1	0	1	0	6	0
2	02-01-2011	1	0	1	0	0	0
3	03-01-2011	1	0	1	0	1	1
4	04-01-2011	1	0	1	0	2	1
5	05-01-2011	1	0	1	0	3	1

Table 1.2: Bike Renting Sample Data (Columns: 7-12)

Weathersit	Temp	Atemp	Humidity	Windspeed	Casual	Registered	Count
2	0.344	0.364	0.806	0.160	331	654	985
2	0.363	0.354	0.696	0.249	131	670	801
1	0.196	0.189	0.437	0.248	120	1229	1349
1	0.200	0.212	0.590	0.160	108	1454	1562
1	0.227	0.229	0.437	0.187	82	1518	1600

As you can see in the table below we have the following 16 variables, using which we have to predict the bike rental counts:

Variable	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	Whether day is holiday or not (extracted from holiday Schedule)
Weekday	Day of the week
Working day	If day is neither weekend nor holiday is 1, otherwise is 0.
Weathersit	(extracted from freemeteo) 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

The above data set consists of 8 Categorical, 7 Continuous and 1 Target Variable.

Chapter 2

Methodology

Before building any predictive model it is necessary to look at the raw data. Looking at data refers to exploring the data, cleaning the data as well as visualizing the data through graphs and plots. This is often called as Exploratory Data Analysis. As the target variable "Count" is continuous, our task is to build regression model to predict the count of bike rented depending on various environmental and seasonal settings. First step in EDA is to look at all the probability distributions of the variables. Most analysis like regression, require the data to be normally distributed. We can visualize that in a glance by looking at the probability distributions or probability density functions of the variable. 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.

Exploratory Data Analysis

In exploring the data, we have

- Rename variables
- Univariate analysis and variable consolidation
- Converted Season, Month, Working day, Weather into categorical variables
- Deleted instant variable as it is nothing but an index and Date variable as month and week are already included
- Omitted registered and casual variable as sum of registered and casual is the total count that is what we have to predict.

Summary

	Temperature	Atemperature	Humidity	Windspeed	Casual	Registered	Count
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

2.1.1 Missing value Analysis

In statistics, missing data, or missing values, occur when no data value is stored for the variable in an observation. Missing data are a common occurrence and can have a significant effect on the conclusions that can be drawn from the data. Missing value analysis was performed in both R and Python. It was found that there were no missing values in the data.

Variable	Missing Values
Season	0
Year	0
Month	0
Holiday	0
Weekday	0
Workingday	0
Weather	0
Temperature	0
Atemperature	0
Humidity	0
Windspeed	0
Casual	0
Registered	0
Count	0

2.1.2 (a) Univariate Analysis

In univariate analysis, we look at the probability density functions numeric variables present in the data including target variable

i. Target variable Count is normally distributed
ii. Independent variables like ‘Temperature’,’Atemperature’, and ‘Registered’ data is distributed normally.
iii. Independent variable ‘casual’ data is slightly skewed to the right so, there is chances of getting outliers.
iv. Other Independent variable ‘Humidity’ data is slightly skewed to the left, here data is already in normalized form.

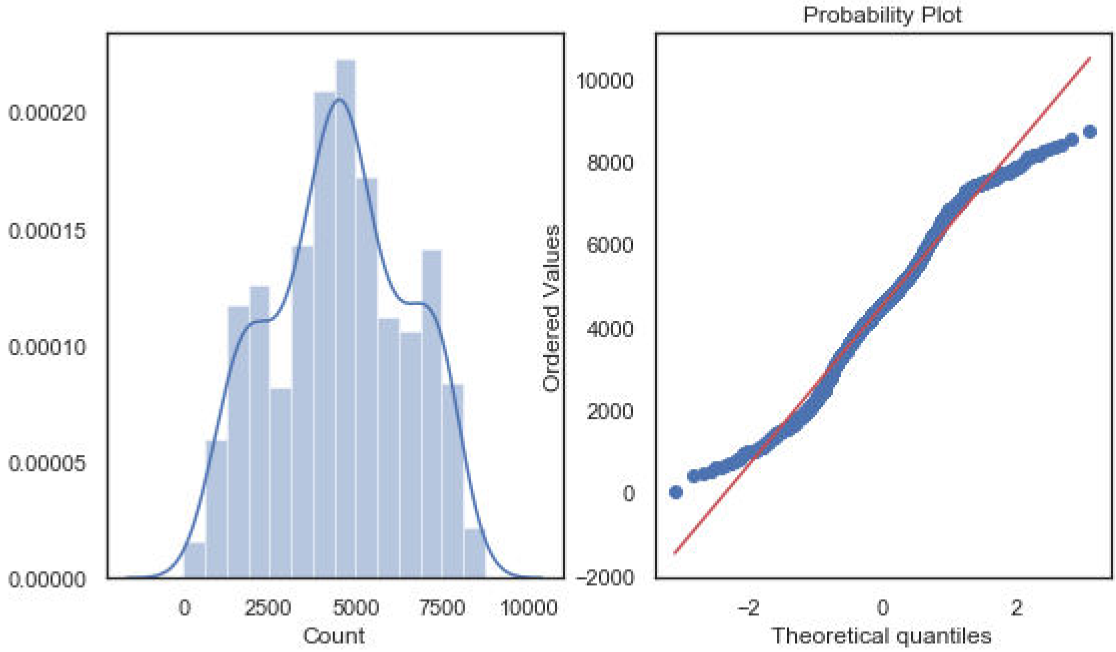
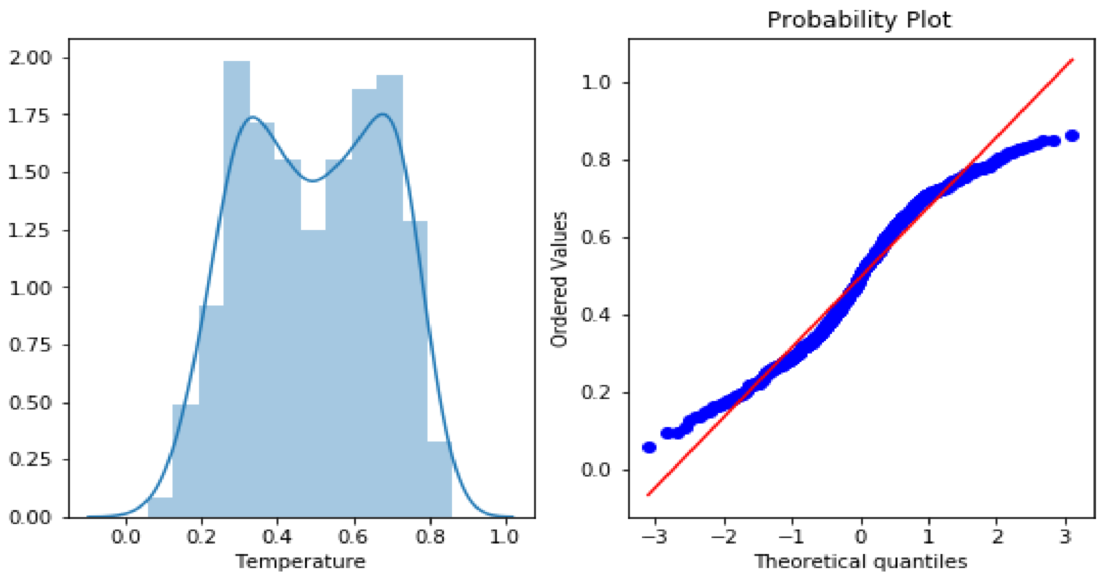
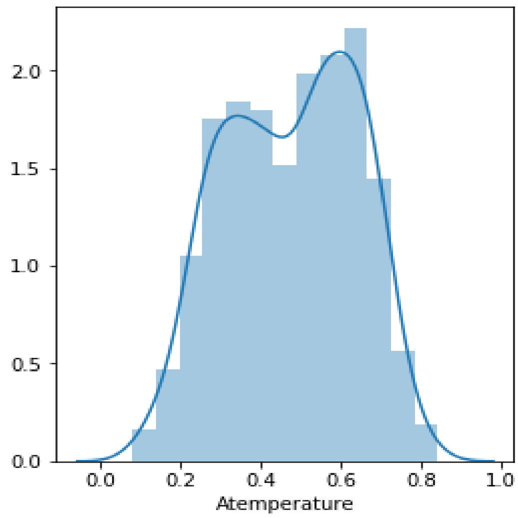
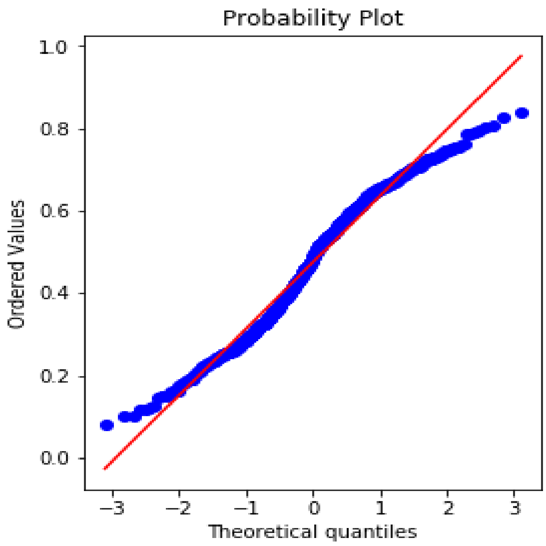
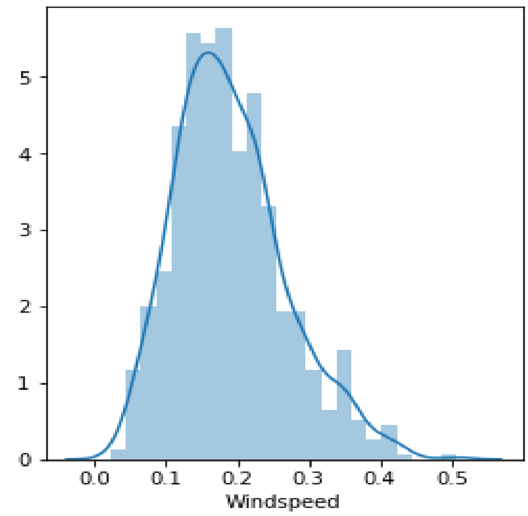


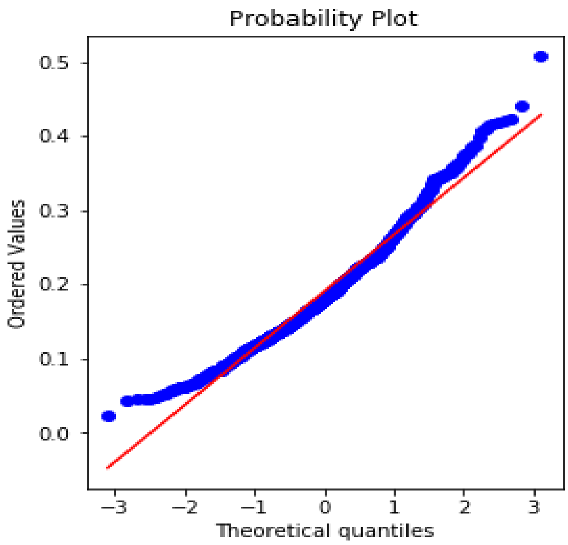
Figure 2.1 Distribution of target variable (Count)

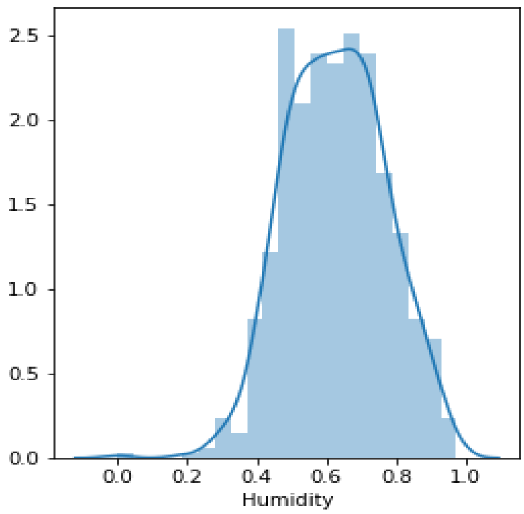


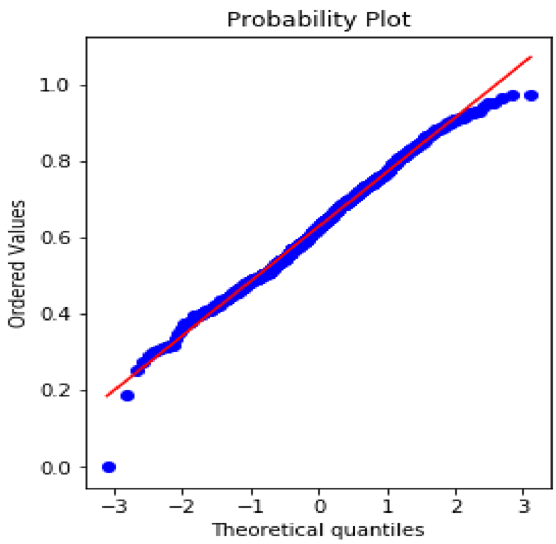

Atemperature


Probability Plot
Ordered Values
Theoretical quantiles


Windspeed


Probability Plot
Ordered Values
Theoretical quantiles


Humidity


Probability Plot
Ordered Values
Theoretical quantiles

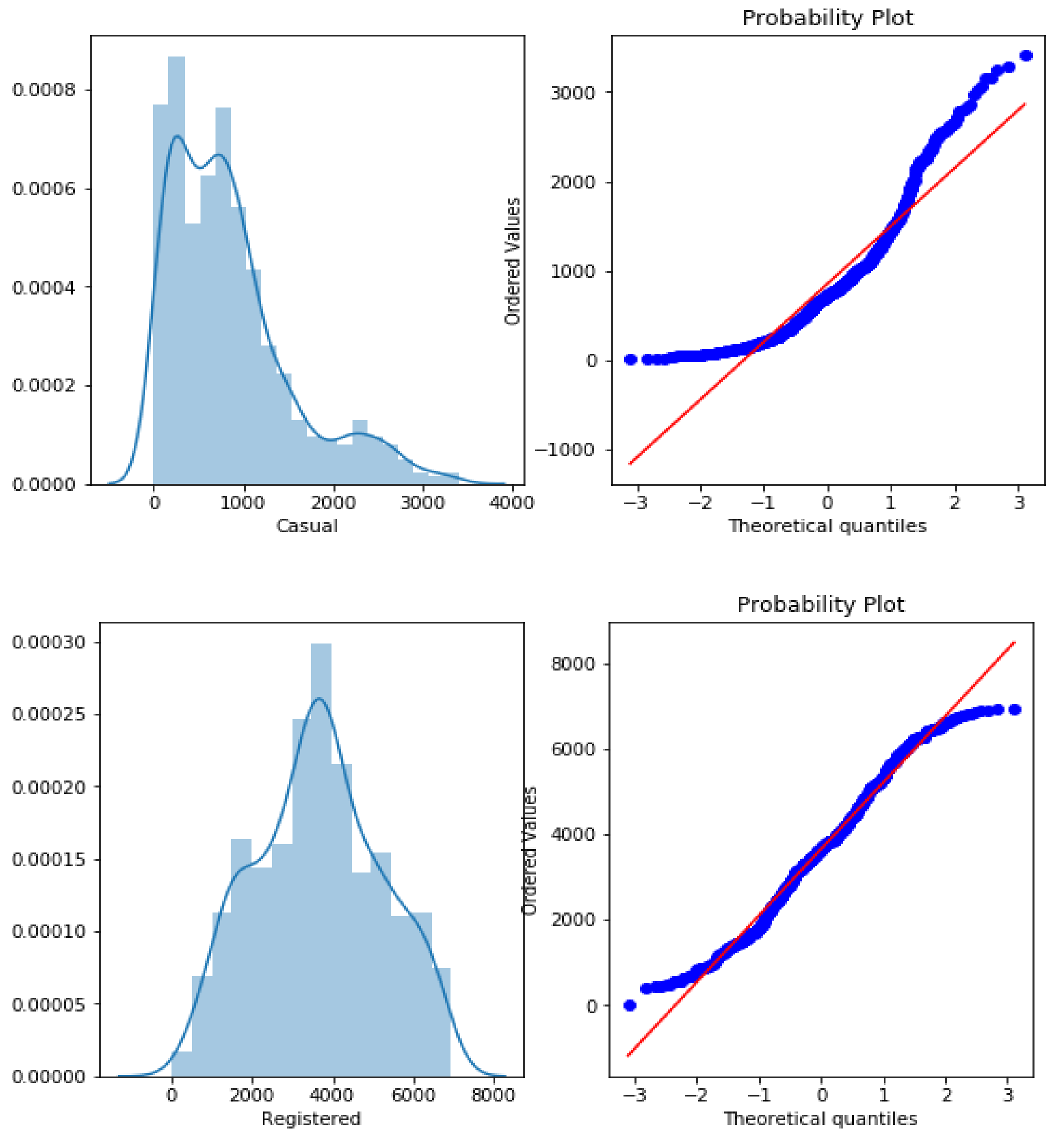


Figure 2.2. Distribution of temp, atemp, windspeed, humidity,casual,registered

2.1.2 (b) Bivariate analysis

In bivariate analysis, we will look at the relationship between target variable and predictor.
From the scatter plots, findings are:

- '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.
- 'Wind speed' has negative linear relationship with 'Count'. With an increase in wind speed, bike count decreases.

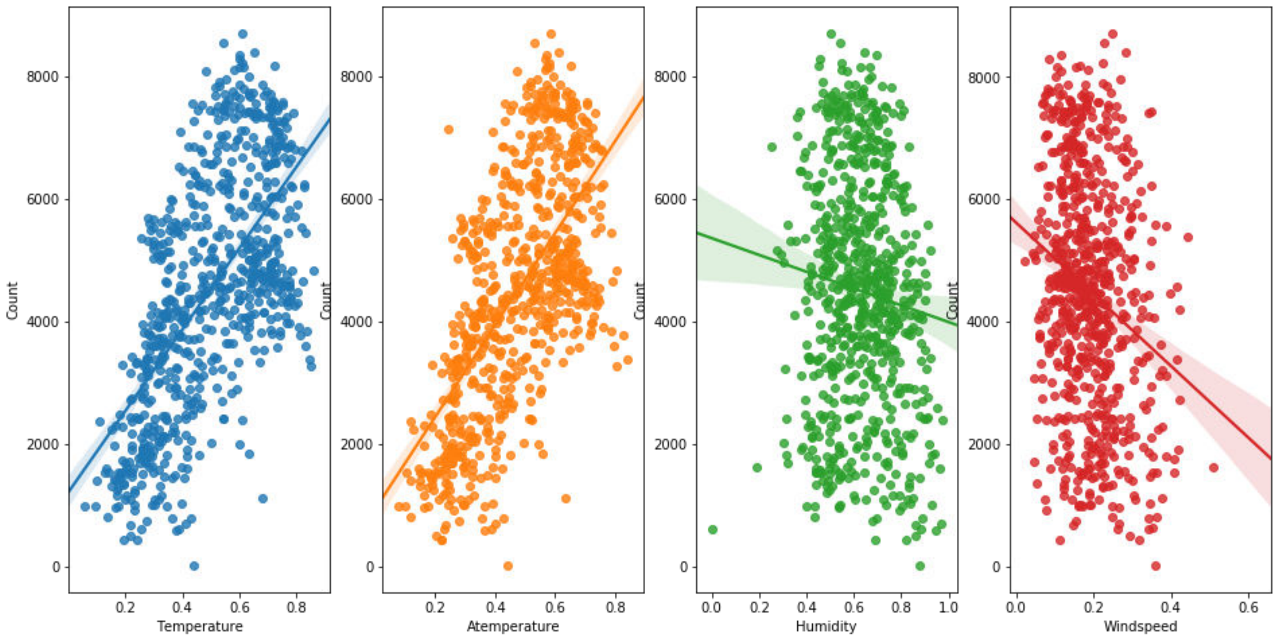
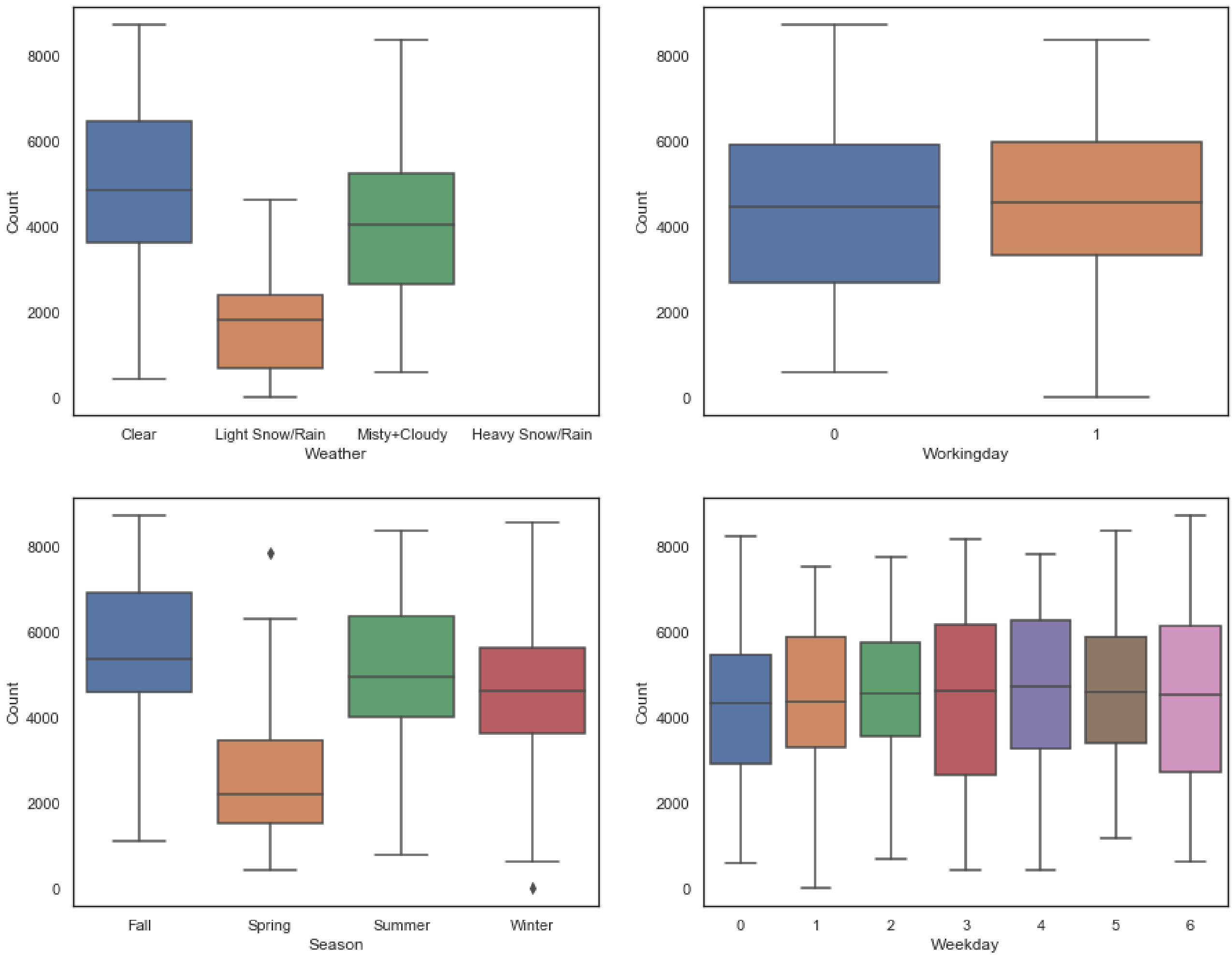


Fig 2.3. Relationship between target variable and continuous predictors



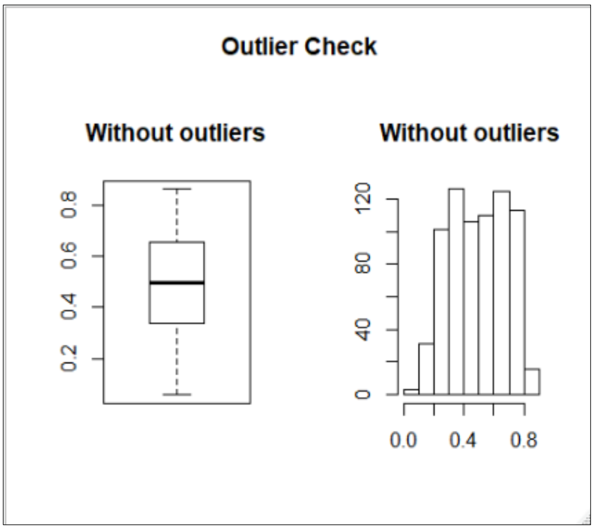
Few Observations

i. Count of bike rentals is higher during clear, few clouds, partly cloudy, cloudy weather and less during light and heavy rains
ii. No significant effect of either holiday or working day on count of bike rentals
iii. Season 'Fall' has seen good number of users renting bikes

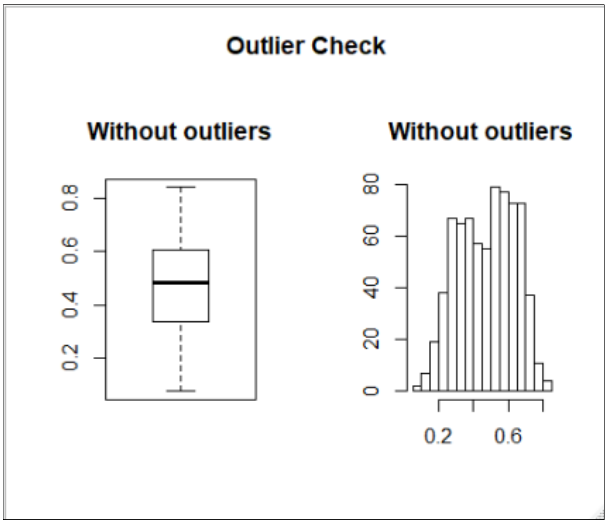
2.1.4 Outlier Analysis

After missing value analysis, we check for outliers in target variable and predictors. Outlier analysis is done to handle all inconsistent observations present in given dataset. As outlier analysis can only be done on continuous variable. There were no outliers present in the dataset. Some extreme values were present in the predictors but those seems to be 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 R implementation.

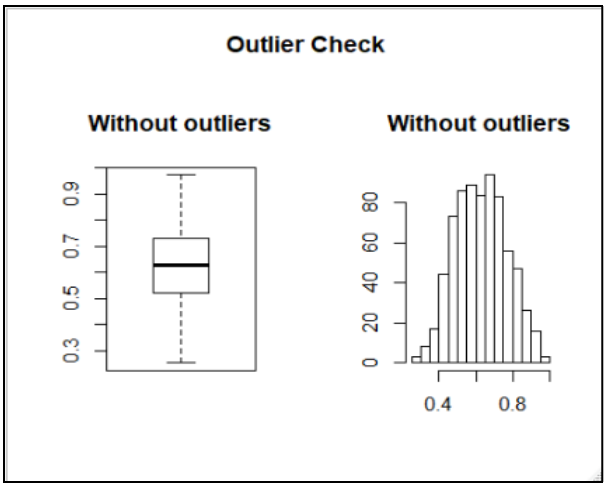
1. Temperature



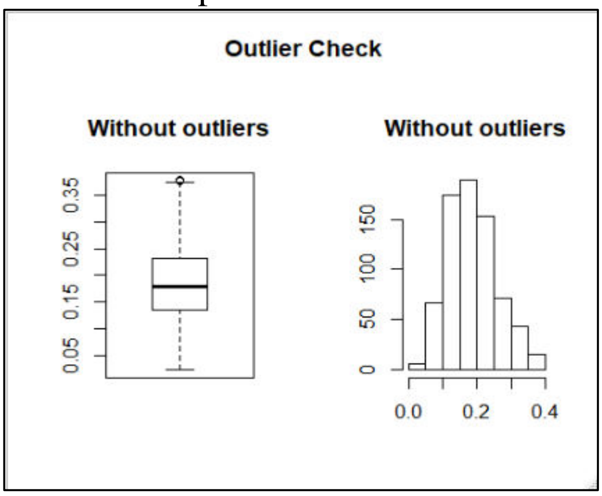
2. Atemperature



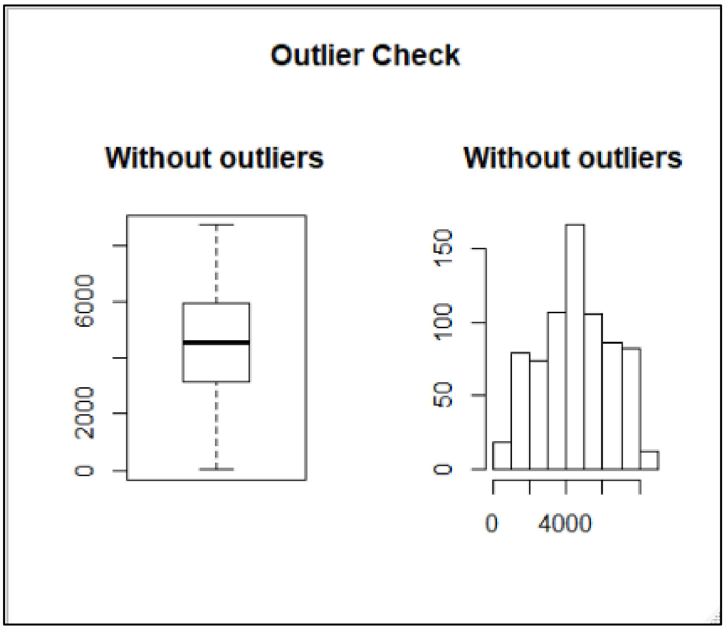
3.Humidity



4. Wind speed



5. Count



2.1.5 Correlation Analysis

Correlation analysis is a method of statistical evaluation used to study the strength of a relationship between two, numerically measured, continuous variables (e.g. height and weight). This particular type of analysis is useful when a researcher wants to establish if there are possible connections between variables. 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. 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.

Correlation Matrix

	Temperature	Atemperature	Humidity	Windspeed	Casual	Registered	Count
Temperature	1	0.99	0.13	-0.16	0.54	0.54	0.63
Atemperature	0.99	1	0.14	-0.18	0.54	0.54	0.63
Humidity	0.13	0.14	1	-0.25	-0.077	-0.091	-0.1
Windspeed	-0.16	-0.18	-0.25	1	-0.17	-0.22	-0.23

	Temperature	Atemperature	Humidity	Windspeed	Casual	Registered	Count
Casual	0.54	0.54	-0.077	-0.17	1	0.4	0.67
Registered	0.54	0.54	-0.091	-0.22	0.4	1	0.95
Count	0.63	0.63	-0.1	-0.23	0.67	0.95	1

Findings:

i. Temperature and Atemperature are highly correlated, thereby one variable will be dropped
ii. Sum of casual and registered users gives us the count which we are predicting thereby these two variables will be dropped.
iii. Count' have a strong and positive relationship with temperature and ambient temperature which is logical. People tend to rent bikes more when temperature is high
iv. Relationship between Humidity, Wind speed and count is very weak.

2.1.7 Chi-squared Test of Independence

The Chi-squared test is used to determine whether an association (or relationship) between 2 categorical variables in a sample is likely to reflect a real association between these 2 variables in the population. The result from the analysis showed that there was association between the dependent and independent categorical variables.

Null hypothesis: Variable (Season, Year, Month, Holiday, Weekday, Weather) and Target variable (Count) are independent
Alternate hypothesis: Variable (Season, Year, Month, Holiday, Weekday, Weather) and Target variable (Count) are correlated

As all the variables had p-value higher than 0.05, we rejected null hypothesis and concluded that the variables have relation with the target variable.

Variable	p-value
Season	0.5441
Year	0.3677
Month	0.4918
Holiday	0.6781
Weekday	0.4102
Working day	0.4544
Weather	0.4678

2.1.8 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.

As given in the problem statement Temperature, Atemperature, Humidity are already normalised. So here we use Normalisation technique on target variable "Count" for rescaling.

Normalised Values of Numerical Variables (Top 6 rows)

Temperature	Humidity	Windspeed	Count
0.3441670	0.805833	0.1604460	0.11079153
0.3634780	0.696087	0.2485390	0.08962264
0.1963640	0.437273	0.2483090	0.15266912
0.2000000	0.590435	0.1602960	0.17717441
0.2269570	0.436957	0.1869000	0.18154625
0.2043480	0.518261	0.0895652	0.18223654

2.2 Modeling

2.2.1 Model Selection

In our bike renting project the target variable is continuous in nature, hence the task of predicting the rentals is regression problem. Two Machine learning algorithms were used.

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 80 % and 20 % ratio. Model was built using the above two machine learning algorithms and after that the diagnostic plots were used to check the assumptions of linear regression. For performance tuning of random forest, hyper parameter tuning was used.

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

```
# ----------------- 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")
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

               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

> 
> # Apply prediction on test set
> test_prediction <- predict(modelAIC, newdata = test_set)
> 
> test_rmse <- rmse(test_set$Count, test_prediction)
> print(paste("root-mean-square error for linear regression model is ", test_rmse))
[1] "root-mean-square error for linear regression model is  821.372628075882"
> print(paste("Mean Absolute Error for linear regression model is ",MAE(test_set$Count,test_prediction)))
[1] "Mean Absolute Error for linear regression model is  575.459501759832"
> print("summary of predicted count values")
[1] "summary of predicted count values"
> summary(test_prediction)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
  -1334    3543    4716    4547    5903    7889 
> print("summary of actual Count values")
[1] "summary of actual Count values"
> summary(test_set$Count)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
    506    3112    4650    4550    5949    8395 
> 
> # 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")
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
> test_prediction_log<- predict(stepwiseLogAICModel, newdata = test_set)
> predict_test_nonlog <- exp(test_prediction_log)
>
> test_rmse2 <- rmse(test_set$Count, predict_test_nonlog)
> print(paste("root-mean-square error between actual and predicted", test_rms
e))
[1] "root-mean-square error between actual and predicted 821.372628075882"
> print(paste("Mean Absolute Error for linear regression model is ",
+             MAE(test_set$Count,predict_test_nonlog)))
[1] "Mean Absolute Error for linear regression model is  696.180959982148"
>
> summary(predict_test_nonlog)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
    486    3063    4381    4484    5822   10614
> summary(test_set$Count)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
    506    3112    4650    4550    5949    8395
>
>
```

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

---------------- Model 2 Random forest ---#

```
model1 <- randomForest(Count ~.,
>                              data = training_set,ntree = 500, mtry = 8, impor
tance = TRUE)

> print(model1)

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

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

```
In [42]: #selecting predictors
         train_feature_space = bikedata.iloc[:,bikedata.columns != 'Count']
         # selecting target class
         target_class = bikedata.iloc[:,bikedata.columns == 'Count']

In [43]: #droping Atemperature due to multicollinearity
         #droping Casual Users and Registered Users because there sum is equal to target variable ie. 'Count'

         train_feature_space = train_feature_space.drop(["Atemperature","Casual Users","Registered Users"],axis =
         1)

In [44]: train_feature_space.shape

Out[44]: (731, 10)

In [45]: train_feature_space.info()

         <class 'pandas.core.frame.DataFrame'>
         RangeIndex: 731 entries, 0 to 730
         Data columns (total 10 columns):
         Season         731 non-null object
         Year           731 non-null object
         Month          731 non-null object
         Holiday        731 non-null category
         Weekday        731 non-null category
         Workingday     731 non-null category
         Weather        731 non-null object
         Temperature    731 non-null float64
         Humidity       731 non-null float64
         Windspeed      731 non-null float64
         dtypes: category(3), float64(3), object(4)
         memory usage: 42.8+ KB
```

In [46]:
```
# creating training and test set
training_set, test_set, train_target, test_target = train_test_split(train_feature_space,
                                                                      target_class,
                                                                      test_size = 0.30,
                                                                      random_state = 456)

# Cleaning test sets to avoid future warning messages
train_target = train_target.values.ravel()
test_target = test_target.values.ravel()
```

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]:

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.

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

In [92]:
```
# Initialize linear regression model
Model = LinearRegression()
Model.fit(X = training_set,y = np.log(train_target))
```

In [95]:
```
#predicting using linear regression
bikedata_Predictions = Model.predict(X=test_set)
bikedata=pd.DataFrame(np.exp(bikedata_Predictions))
bikedata.describe()
```

Out[95]:

	0
count	220.000000
mean	4438.291596
std	2120.623071
min	1034.553984
25%	2730.336107
50%	4096.304410
75%	5708.079487
max	11000.217123

In [96]:
```
bikedata_errors = abs(np.exp(bikedata_Predictions) - test_target)
# Print out the mean absolute error (mae)
print('Mean Absolute Error:', round(np.mean(bikedata_errors), 2), 'degrees.')
```

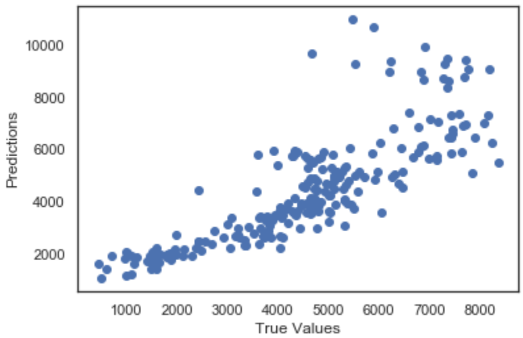
```
Mean Absolute Error: 899.5 degrees.
```

In [97]:
```
rmse = sqrt(mean_squared_error(test_target, np.exp(bikedata_Predictions)))
print("RMSE for test set in linear regression is :" , rmse)
```

```
RMSE for test set in linear regression is : 1222.1581373120362
```

In [98]:
```
## The line / model
plt.scatter(test_target, np.exp(bikedata_Predictions))
plt.xlabel("True Values")
plt.ylabel("Predictions")
```

Out[98]: Text(0, 0.5, 'Predictions')



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)
```

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],
```

```
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)
```


```
          print("RMSE for test set in random forest regressor  is :" , rmse_rf)
```


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

3.1 Model Evaluation

Now that we have a few models for predicting the target variable, we need to decide which one to choose. There are several criteria that exist for evaluating and comparing models.

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	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	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

From the error metric we can see that random forest is performing better than linear regression in both implementations. 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.

Model selection

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 implementation- Codes

```
#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:/EdwisorVanusha/Project/Master files")

#Check working directory-
getwd()

#load data-
bikedata= read.csv("bike_rental.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$Season) <- c("spring", "summer", "fall", "winter")
bikedata$Year <- as.factor(bikedata$Year)
levels(bikedata$Year) <- c(2011, 2012)
bikedata$Month <- as.factor(bikedata$Month)
levels(bikedata$Month) <- c("Jan", "Feb", "Mar", "Apr", "May", "Jun", "Jul", "Aug", "Sep",
"Oct", "Nov", "Dec")
bikedata$Holiday <- as.factor(bikedata$Holiday)
levels(bikedata$Holiday) <- c("Not Holiday", "Holiday")
bikedata$Weekday <- as.factor(bikedata$Weekday)
levels(bikedata$Weekday) <- c("Sun", "Mon", "Tues", "Wed", "Thurs", "Fri", "Sat")
bikedata$Workingday <- as.factor(bikedata$Workingday)
levels(bikedata$Workingday) <- c("Holiday", "Workingday")
bikedata$Weather<- as.factor(bikedata$Weather)
levels(bikedata$Weather) <- c("Clear", "Cloudy", "Rainy","Heavy rain")
str(bikedata)

#----------------------------------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


#-----------------------------------------------------------------------------------------------#
#                                                                                #
#                    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

#-----------------------------------------------------------------------------------------------#
#                                                                       #
#                    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

# ---------------------------------------------------------------------------------------------- #
#
#                        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)

# ---------------- 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")
```

In [52]:

```
# importing requried library

import os
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from scipy import stats
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split, RandomizedSearchCV
from sklearn import metrics
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error
import statsmodels.api as sm
from math import sqrt
import time
import random
import warnings
warnings.filterwarnings("ignore", category=DeprecationWarning)
pd.options.mode.chained_assignment = None

%matplotlib inline
```

Setting Working Directory

In [53]:

```
#set working directory-
os.chdir("F:/Edvisor Project/Bike_Rental")

#check current working directory-
os.getcwd()
```

Out[53]:

```
'F:\\Edvisor Project\\Bike_Rental'
```

In [54]:

```
# Read the data
bikedata = pd.read_csv('day.csv')
```

In [55]:

```
bikedata.head()
```

Out[55]:

	instant	dteday	season	yr	mnth	holiday	weekday	workingday	weathersit	temp	atemp	hum	windspeed	casual	re(
0	1	2011-01-01	1	0	1	0	6	0	2	0.344167	0.363625	0.805833	0.160446	331	
1	2	2011-01-02	1	0	1	0	0	0	2	0.363478	0.353739	0.696087	0.248539	131	
2	3	2011-01-03	1	0	1	0	1	1	1	0.196364	0.189405	0.437273	0.248309	120	
3	4	2011-01-04	1	0	1	0	2	1	1	0.200000	0.212122	0.590435	0.160296	108	
4	5	2011-01-05	1	0	1	0	3	1	1	0.226957	0.229270	0.436957	0.186900	82	

In [56]:

```
bikedata.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 731 entries, 0 to 730
Data columns (total 16 columns):
instant       731 non-null int64
dteday        731 non-null object
season        731 non-null int64
yr            731 non-null int64
mnth          731 non-null int64
holiday       731 non-null int64
weekday       731 non-null int64
workingday    731 non-null int64
weathersit    731 non-null int64
temp          731 non-null float64
atemp         731 non-null float64
hum           731 non-null float64
windspeed     731 non-null float64
casual        731 non-null int64
registered    731 non-null int64
cnt           731 non-null int64
dtypes: float64(4), int64(11), object(1)
memory usage: 91.5+ KB
```

Data Cleaning

In [57]:

```
# change the names of the columns
#Rename varaible
bikedata = bikedata.rename(columns = {'instant':'Index','dteday':'Date','season':'Season','yr':'Year','mnth':'Month','holiday':'Holiday','weekday':'Weekday','workingday':'Workingday','weathersit':'Weather','temp':'Temperature','atemp':'Atemperature','hum':'Humidity','windspeed':'Windspeed','casual':'Casual Users','registered':'Registered Users','cnt':'Count'})

bikedata.columns
```

Out[57]:

```
Index(['Index', 'Date', 'Season', 'Year', 'Month', 'Holiday', 'Weekday',
       'Workingday', 'Weather', 'Temperature', 'Atemperature', 'Humidity',
       'Windspeed', 'Casual Users', 'Registered Users', 'Count'],
      dtype='object')
```

In [58]:

```
# Mapping numbers to understandable text
season_dict = {1:'Spring', 2:'Summer', 3:'Fall', 4:'Winter'}
weather_dict = {1:'Clear', 2:'Misty+Cloudy', 3:'Light Snow/Rain', 4:'Heavy Snow/Rain'}
month_dict = {1: 'Jan', 2: 'Feb', 3: 'Mar', 4: 'Apr', 5: 'May', 6: 'June', 7:'July', 8: 'Aug', 9: 'Sep', 10: 'Oct', 11: 'Nov', 12: 'Dec'}
year_dict = {0: '2011', 1: '2012'}
bikedata['Season'] = bikedata['Season'].map(season_dict)
bikedata['Weather'] = bikedata['Weather'].map(weather_dict)
bikedata['Month'] = bikedata['Month'].map(month_dict)
bikedata['Year'] = bikedata['Year'].map(year_dict)

bikedata.head()
```

Out[58]:

	Index	Date	Season	Year	Month	Holiday	Weekday	Workingday	Weather	Temperature	Atemperature	Humidity	Windspee
0	1	2011-01-01	Spring	2011	Jan	0	6	0	Misty+Cloudy	0.344167	0.363625	0.805833	0.16044
1	2	2011-01-02	Spring	2011	Jan	0	0	0	Misty+Cloudy	0.363478	0.353739	0.696087	0.24853
2	3	2011-01-03	Spring	2011	Jan	0	1	1	Clear	0.196364	0.189405	0.437273	0.24830
3	4	2011-01-04	Spring	2011	Jan	0	2	1	Clear	0.200000	0.212122	0.590435	0.16029

	Index	Date	Season	Year	Month	Holiday	Weekday	Workingday	Weather	Temperature	Atemperature	Humidity	Windspee
4	5	2011-01-05	Spring	2011	Jan	0	3	1	Clear	0.226957	0.229270	0.436957	0.18690

In [60]:

```
#converting to categorical variable
#"season",
categorical_variable = ["Season","Year","Month","Holiday","Weekday","Workingday","Weather"]

for var in categorical_variable:
    bikedata[var] = bikedata[var].astype("category")
```

Shape of the Data

In [61]:

```
print('Shape of data: ', bikedata.shape)
bikedata.info()
```

```
Shape of data:  (731, 16)
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 731 entries, 0 to 730
Data columns (total 16 columns):
Index               731 non-null int64
Date                731 non-null object
Season              731 non-null category
Year                731 non-null category
Month               731 non-null category
Holiday             731 non-null category
Weekday             731 non-null category
Workingday          731 non-null category
Weather             731 non-null category
Temperature         731 non-null float64
Atemperature        731 non-null float64
Humidity            731 non-null float64
Windspeed           731 non-null float64
Casual Users        731 non-null int64
Registered Users    731 non-null int64
Count               731 non-null int64
dtypes: category(7), float64(4), int64(4), object(1)
memory usage: 57.8+ KB
```

We have 731 observations, 15 predictors and 1 target variable. Count is our target variable. Next examining variable types

In [29]:

```
print('Unique value count for each feature:')
for i in bikedata:
    print(i, '-->', bikedata[i].unique().size)
```

```
Unique value count for each feature:
Season --> 4
Year --> 2
Month --> 12
Holiday --> 2
Weekday --> 7
Workingday --> 2
Weather --> 3
Temperature --> 499
Atemperature --> 690
Humidity --> 595
Windspeed --> 650
Casual Users --> 606
Registered Users --> 679
Count --> 696
```

In [63]:

```
bikedata = bikedata.drop(['Index','Date'], axis = 1)
bikedata.head(4)
```

Out[63]:

	Season	Year	Month	Holiday	Weekday	Workingday	Weather	Temperature	Atemperature	Humidity	Windspeed	Casual Users	Re
0	Spring	2011	Jan	0	6	0	Misty+Cloudy	0.344167	0.363625	0.805833	0.160446	331	
1	Spring	2011	Jan	0	0	0	Misty+Cloudy	0.363478	0.353739	0.696087	0.248539	131	
2	Spring	2011	Jan	0	1	1	Clear	0.196364	0.189405	0.437273	0.248309	120	
3	Spring	2011	Jan	0	2	1	Clear	0.200000	0.212122	0.590435	0.160296	108	

In [65]:

```
bikedata.describe()
```

Out[65]:

	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

Missing value Analysis

In [64]:

```
#Check Missing values
missing_value = pd.DataFrame(bikedata.isnull().sum())
missing_value
```

Out[64]:

	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

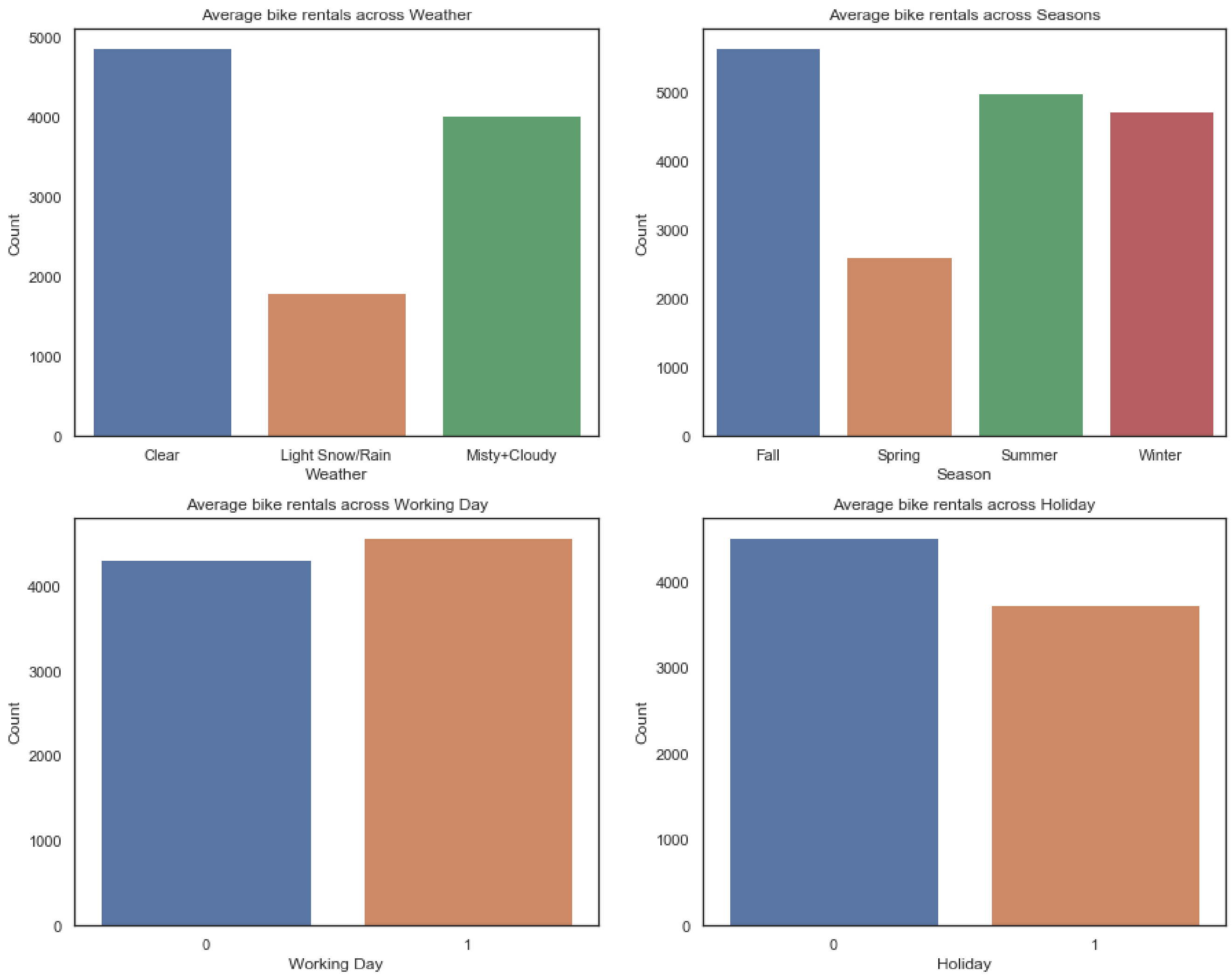
Exploratory Data Analysis

In [30]:

```
sns.set(style="white")
sns.set(style="white", color_codes=True)
```

In [31]:

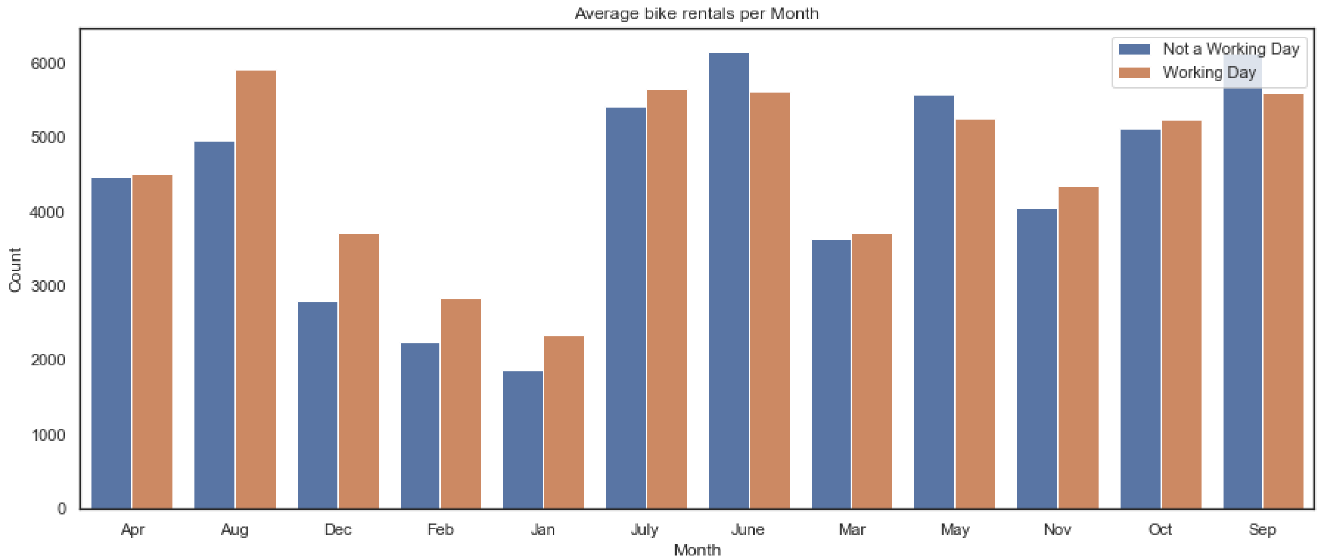
```
# Average values across each of the categorical columns
fig = plt.figure(figsize=(15, 12))
axes = fig.add_subplot(2, 2, 1)
group_weather = pd.DataFrame(bikedata.groupby(['Weather'])['Count'].mean()).reset_index()
sns.barplot(data=group_weather, x='Weather', y='Count', ax=axes)
axes.set(xlabel='Weather', ylabel='Count', title='Average bike rentals across Weather')

axes = fig.add_subplot(2, 2, 2)
group_season = pd.DataFrame(bikedata.groupby(['Season'])['Count'].mean()).reset_index()
sns.barplot(data=group_season, x='Season', y='Count', ax=axes)
axes.set(xlabel='Season', ylabel='Count', title='Average bike rentals across Seasons')

axes = fig.add_subplot(2, 2, 3)
group_workingday = pd.DataFrame(bikedata.groupby(['Workingday'])['Count'].mean()).reset_index()
sns.barplot(data=group_workingday, x='Workingday', y='Count', ax=axes)
axes.set(xlabel='Working Day', ylabel='Count', title='Average bike rentals across Working Day')

axes = fig.add_subplot(2, 2, 4)
group_season = pd.DataFrame(bikedata.groupby(['Holiday'])['Count'].mean()).reset_index()
sns.barplot(data=group_season, x='Holiday', y='Count', ax=axes)
axes.set(xlabel='Holiday', ylabel='Count', title='Average bike rentals across Holiday')

plt.show()
```



Monthly Distribution

In [32]:

```
# Average Monthly Count Distribution plot
f, axes = plt.subplots(nrows=1, ncols=1, figsize=(15, 6))
group_month = pd.DataFrame(bikedata.groupby(['Month', 'Workingday'])['Count'].mean()).reset_index(
)
sns.barplot(data=group_month, x='Month', y='Count', hue='Workingday', ax=axes)
axes.set(xlabel='Month', ylabel='Count', title='Average bike rentals per Month')
handles, _ = axes.get_legend_handles_labels()
axes.legend(handles, ['Not a Working Day', 'Working Day'])
plt.show()
```

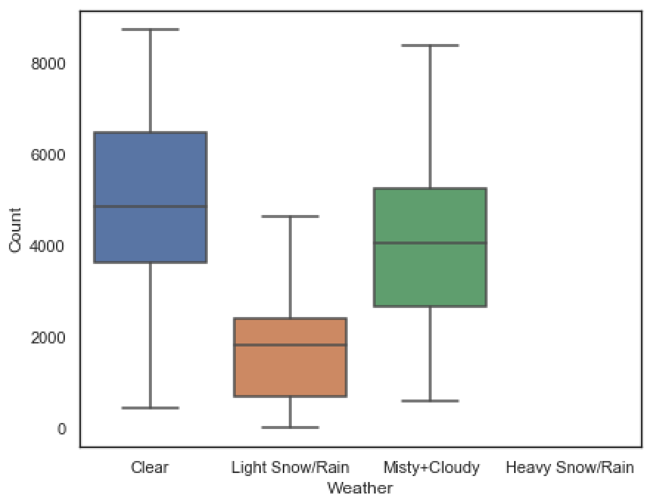


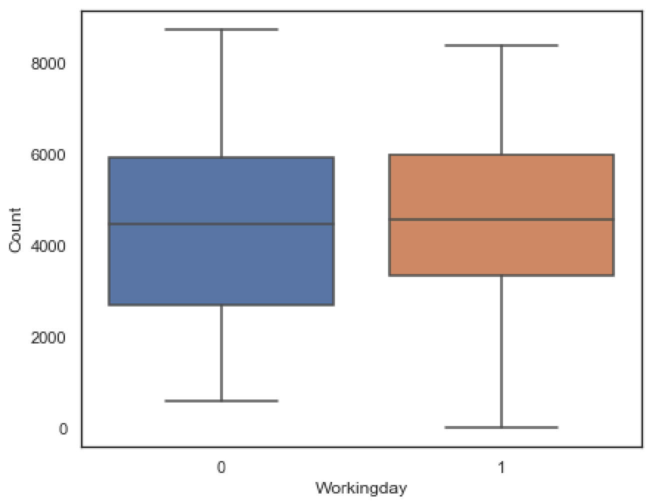
Using seaborn boxplots to get an idea of the distribution and outliers acorss various categorical features

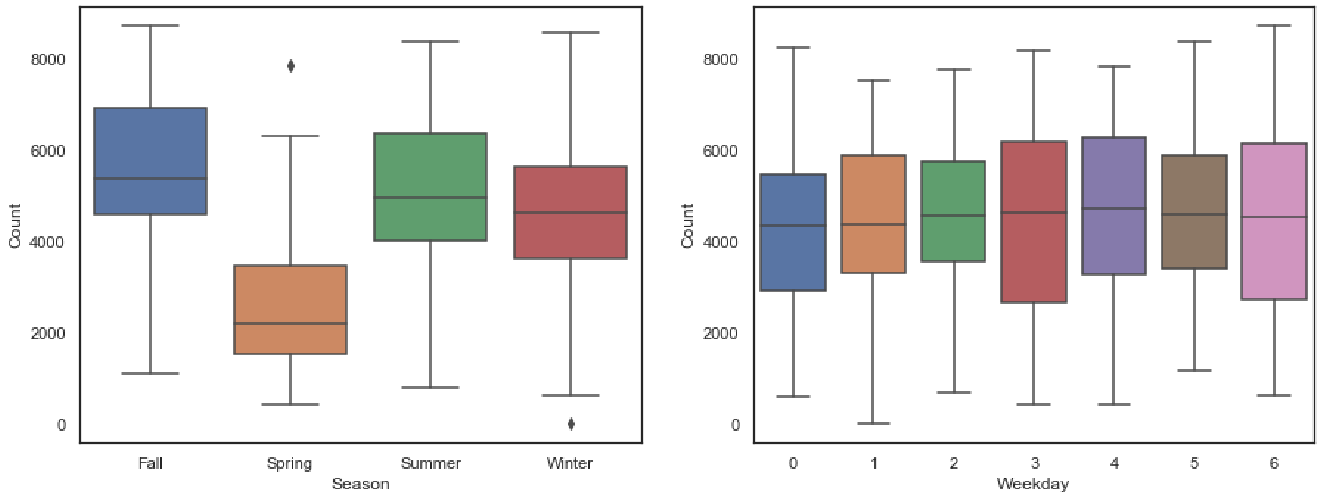
Outlier Analysis

In [33]:

```
# Seaborn boxplots to get an idea of the distribution/outliers
f, axes = plt.subplots(2, 2, figsize=(15, 12))
hue_order= ['Clear',  'Light Snow/Rain', 'Misty+Cloudy','Heavy Snow/Rain',]
sns.boxplot(data=bikedata, y='Count', x='Weather', ax=axes[0][0], order=hue_order)
sns.boxplot(data=bikedata, y='Count', x='Workingday', ax=axes[0][1])
hue_order= ['Fall', 'Spring', 'Summer', 'Winter']
sns.boxplot(data=bikedata, y='Count', x='Season', ax=axes[1][0], order=hue_order)
sns.boxplot(data=bikedata, y='Count', x='Weekday', ax=axes[1][1])

plt.show()
```







Correlation Analysis

In [35]:

```
churn_corr = bikedata.corr()
cmap = cmap=sns.diverging_palette(15, 250, as_cmap=True)

def magnify():
    return [dict(selector="th",
                 props=[("font-size", "12pt")]),
            dict(selector="td",
                 props=[('padding', "0em 0em")]),
            dict(selector="th:hover",
                 props=[("font-size", "12pt")]),
            dict(selector="tr:hover td:hover",
                 props=[('max-width', '200px'),
                        ('font-size', '12pt')])
]

churn_corr.style.background_gradient(cmap, axis=1)\
    .set_properties(**{'max-width': '90px', 'font-size': '12pt'})\
    .set_caption("Correlation matrix")\
    .set_precision(2)\
    .set_table_styles(magnify())
```

Out[35]:

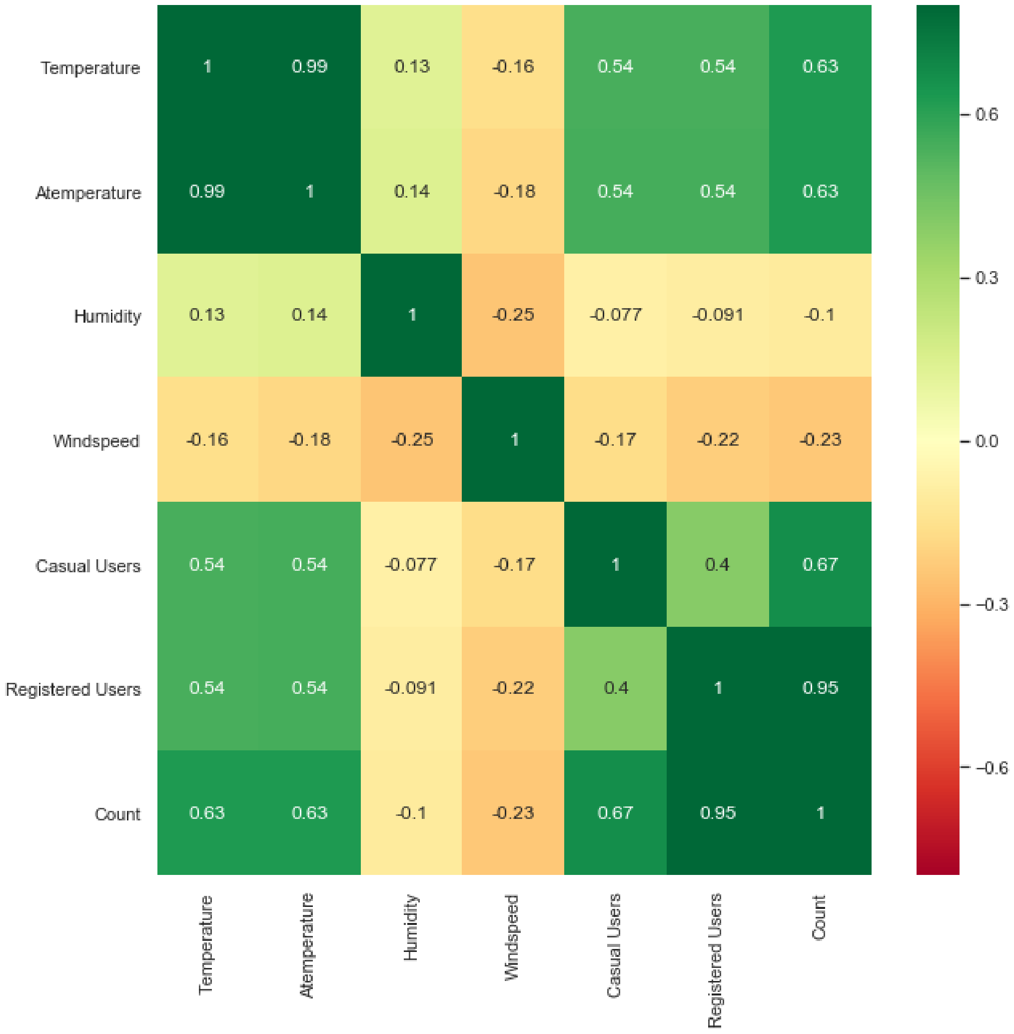
Correlation matrix

	Temperature	Atemperature	Humidity	Windspeed	Casual Users	Registered Users	Count
Temperature	1	0.99	0.13	-0.16	0.54	0.54	0.63
Atemperature	0.99	1	0.14	-0.18	0.54	0.54	0.63
Humidity	0.13	0.14	1	-0.25	-0.077	-0.091	-0.1
Windspeed	-0.16	-0.18	-0.25	1	-0.17	-0.22	-0.23
Casual Users	0.54	0.54	-0.077	-0.17	1	0.4	0.67
Registered Users	0.54	0.54	-0.091	-0.22	0.4	1	0.95
Count	0.63	0.63	-0.1	-0.23	0.67	0.95	1

In [38]:

```
corr = bikedata.corr()
mask = np.zeros_like(corr)
```

```
mask[np.tril_indices_from(mask)] = False
fig = plt.figure(figsize=(10, 10))
sns.heatmap(corr, mask=mask, annot=True, cbar=True, vmax=0.8, vmin=-0.8, cmap='RdYlGn')
plt.show()
```



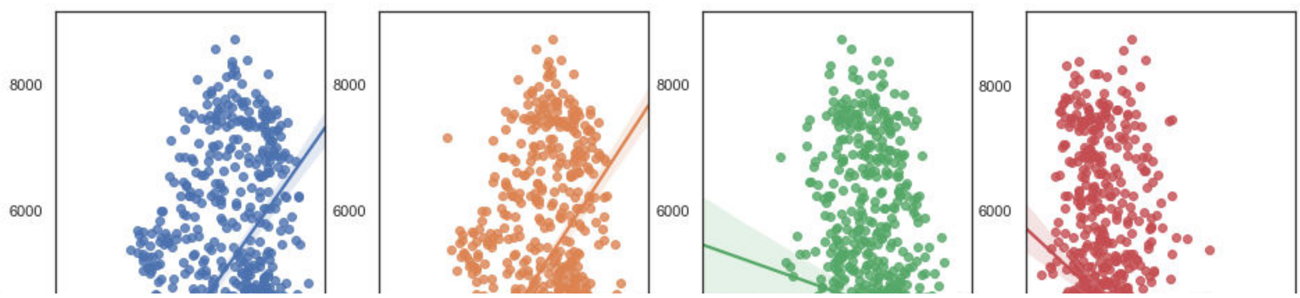
Bivariate analysis

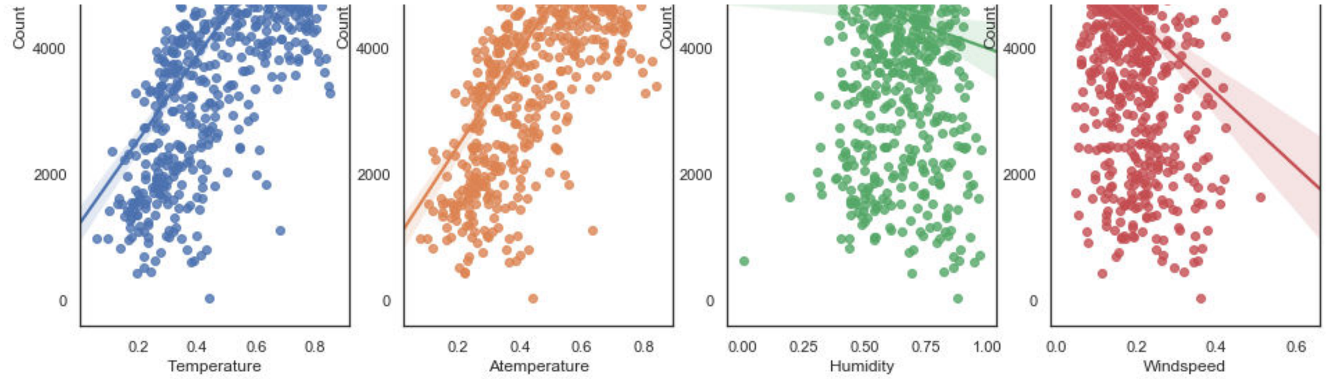
In [37]:

```
# Bivariate analysis of cnt and continous predictor

fig,(ax1,ax2,ax3,ax4) = plt.subplots(ncols=4)
fig.set_size_inches(16,8)

sns.regplot(x="Temperature",y="Count",data=bikedata,ax=ax1)
sns.regplot(x="Atemperature",y="Count",data=bikedata,ax=ax2)
sns.regplot(x="Humidity",y="Count",data=bikedata,ax=ax3)
sns.regplot(x="Windspeed",y="Count",data=bikedata,ax=ax4)
```

Out[37]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x223f31d6d30>
```





From the above plot, it is evident that cnt has a positive linear relationship with Temperature and Atemperature. On the other hand, cnt has a negative linear relationship with Windspeed. Humidity has a little negative linear relationship with Count.

Distribution of target Variable

In [40]:

```
fig,(ax1,ax2) = plt.subplots(ncols=2)
fig.set_size_inches(9,5)
sns.distplot(bikedata["Count"],ax=ax1)
stats.probplot(bikedata["Count"], dist='norm', fit=True, plot=ax2)
```

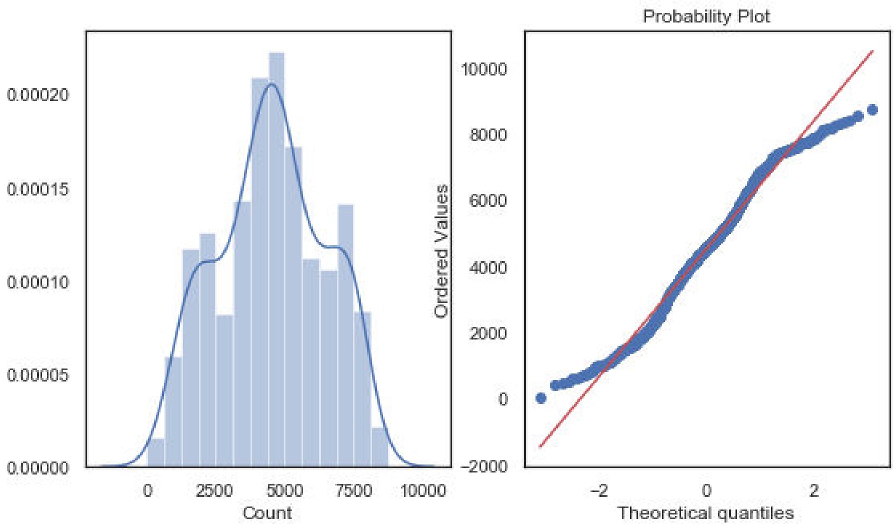
Out[40]:

```
((array([-3.10612952, -2.83371839, -2.68121219, -2.57340905, -2.48915191,
         -2.41955673, -2.36001798, -2.30782877, -2.26125818, -2.21912992,
         -2.18060696, -2.14507173, -2.11205508, -2.08119197, -2.05219258,
         -2.0248228 , -1.99889075, -1.97423711, -1.95072808, -1.92825019,
         -1.90670633, -1.88601273, -1.8660966 , -1.84689427, -1.82834975,
         -1.81041348, -1.79304141, -1.77619419, -1.75983653, -1.74393663,
         -1.72846577, -1.71339788, -1.69870925, -1.68437825, -1.67038506,
         -1.6567115 , -1.64334086, -1.63025771, -1.6174478 , -1.60489794,
         -1.59259587, -1.58053022, -1.56869036, -1.55706641, -1.54564912,
         -1.53442983, -1.52340042, -1.51255328, -1.50188124, -1.49137757,
         -1.4810359 , -1.47085025, -1.46081495, -1.45092464, -1.44117426,
         -1.431559  , -1.4220743 , -1.41271583, -1.40347947, -1.39436132,
         -1.38535765, -1.3764649 , -1.36767969, -1.35899879, -1.35041911,
         -1.3419377 , -1.33355173, -1.32525852, -1.31705546, -1.30894008,
         -1.30091001, -1.29296295, -1.28509673, -1.27730922, -1.26959842,
         -1.26196238, -1.25439922, -1.24690714, -1.2394844 , -1.23212934,
         -1.22484033, -1.21761582, -1.21045431, -1.20335435, -1.19631454,
         -1.18933352, -1.18240998, -1.17554267, -1.16873035, -1.16197185,
         -1.155266  , -1.14861171, -1.14200789, -1.13545351, -1.12894754,
         -1.12248901, -1.11607697, -1.10971049, -1.10338867, -1.09711064,
         -1.09087556, -1.08468261, -1.07853098, -1.07241989, -1.06634859,
         -1.06031635, -1.05432244, -1.04836618, -1.04244688, -1.03656388,
         -1.03071654, -1.02490423, -1.01912634, -1.01338227, -1.00767144,
         -1.0019933 , -0.99634727, -0.99073283, -0.98514945, -0.9795966 ,
         -0.97407381, -0.96858056, -0.96311639, -0.95768082, -0.9522734 ,
         -0.94689368, -0.94154123, -0.93621562, -0.93091643, -0.92564325,
         -0.92039569, -0.91517335, -0.90997585, -0.90480282, -0.89965388,
         -0.89452869, -0.88942689, -0.88434814, -0.87929209, -0.87425842,
         -0.86924681, -0.86425694, -0.85928849, -0.85434116, -0.84941466,
         -0.84450869, -0.83962296, -0.83475719, -0.8299111 , -0.82508442,
         -0.8202769 , -0.81548825, -0.81071823, -0.80596659, -0.80123308,
         -0.79651745, -0.79181947, -0.78713889, -0.7824755 , -0.77782907,
         -0.77319937, -0.76858618, -0.76398929, -0.75940848, -0.75484356,
         -0.75029432, -0.74576055, -0.74124205, -0.73673864, -0.73225012,
         -0.72777631, -0.72331702, -0.71887206, -0.71444126, -0.71002444,
         -0.70562143, -0.70123206, -0.69685616, -0.69249356, -0.6881441 ,
         -0.68380762, -0.67948396, -0.67517297, -0.67087448, -0.66658836,
         -0.66231445, -0.6580526 , -0.65380267, -0.64956452, -0.64533801,
         -0.64112299, -0.63691932, -0.63272689, -0.62854555, -0.62437516,
         -0.62021561, -0.61606676, -0.61192849, -0.60780067, -0.60368318,
         -0.59957591, -0.59547872, -0.5913915 , -0.58731414, -0.58324652,
         -0.57918853, -0.57514005, -0.57110098, -0.5670712 , -0.56305061,
         -0.5590391 , -0.55503657, -0.55104291, -0.54705802, -0.5430818 ,
         -0.53911414, -0.53515496, -0.53120414, -0.5272616 , -0.52332724,
```

```
-0.51940096, -0.51548267, -0.51157228, -0.5076697 , -0.50377484,
-0.4998876 , -0.4960079 , -0.49213565, -0.48827077, -0.48441317,
-0.48056276, -0.47671946, -0.4728832 , -0.46905388, -0.46523142,
-0.46141575, -0.45760679, -0.45380445, -0.45000867, -0.44621936,
-0.44243644, -0.43865984, -0.43488949, -0.43112532, -0.42736724,
-0.42361519, -0.41986909, -0.41612887, -0.41239447, -0.40866581,
-0.40494282, -0.40122544, -0.39751359, -0.39380721, -0.39010623,
-0.38641059, -0.38272022, -0.37903506, -0.37535503, -0.37168008,
-0.36801015, -0.36434516, -0.36068506, -0.35702979, -0.35337928,
-0.34973347, -0.34609231, -0.34245573, -0.33882367, -0.33519607,
-0.33157289, -0.32795405, -0.3243395 , -0.32072918, -0.31712304,
-0.31352101, -0.30992305, -0.3063291 , -0.3027391 , -0.299153  ,
-0.29557074, -0.29199227, -0.28841753, -0.28484648, -0.28127906,
-0.27771521, -0.27415488, -0.27059803, -0.2670446 , -0.26349453,
-0.25994779, -0.25640431, -0.25286405, -0.24932695, -0.24579297,
-0.24226206, -0.23873416, -0.23520924, -0.23168723, -0.2281681 ,
-0.22465178, -0.22113825, -0.21762744, -0.21411931, -0.21061382,
-0.20711091, -0.20361054, -0.20011267, -0.19661724, -0.19312421,
-0.18963354, -0.18614518, -0.18265908, -0.17917519, -0.17569349,
-0.17221391, -0.16873641, -0.16526095, -0.16178749, -0.15831598,
-0.15484638, -0.15137863, -0.14791271, -0.14444857, -0.14098615,
-0.13752543, -0.13406635, -0.13060888, -0.12715297, -0.12369857,
-0.12024565, -0.11679416, -0.11334407, -0.10989532, -0.10644788,
-0.1030017 , -0.09955675, -0.09611298, -0.09267034, -0.08922881,
-0.08578833, -0.08234887, -0.07891038, -0.07547282, -0.07203616,
-0.06860034, -0.06516534, -0.0617311 , -0.05829759, -0.05486477,
-0.0514326 , -0.04800103, -0.04457003, -0.04113955, -0.03770955,
-0.03428   , -0.03085085, -0.02742207, -0.0239936 , -0.02056542,
-0.01713748, -0.01370974, -0.01028217, -0.00685471, -0.00342734,
 0.        ,  0.00342734,  0.00685471,  0.01028217,  0.01370974,
 0.01713748,  0.02056542,  0.0239936 ,  0.02742207,  0.03085085,
 0.03428   ,  0.03770955,  0.04113955,  0.04457003,  0.04800103,
 0.0514326 ,  0.05486477,  0.05829759,  0.0617311 ,  0.06516534,
 0.06860034,  0.07203616,  0.07547282,  0.07891038,  0.08234887,
 0.08578833,  0.08922881,  0.09267034,  0.09611298,  0.09955675,
 0.1030017 ,  0.10644788,  0.10989532,  0.11334407,  0.11679416,
 0.12024565,  0.12369857,  0.12715297,  0.13060888,  0.13406635,
 0.13752543,  0.14098615,  0.14444857,  0.14791271,  0.15137863,
 0.15484638,  0.15831598,  0.16178749,  0.16526095,  0.16873641,
 0.17221391,  0.17569349,  0.17917519,  0.18265908,  0.18614518,
 0.18963354,  0.19312421,  0.19661724,  0.20011267,  0.20361054,
 0.20711091,  0.21061382,  0.21411931,  0.21762744,  0.22113825,
 0.22465178,  0.2281681 ,  0.23168723,  0.23520924,  0.23873416,
 0.24226206,  0.24579297,  0.24932695,  0.25286405,  0.25640431,
 0.25994779,  0.26349453,  0.2670446 ,  0.27059803,  0.27415488,
 0.27771521,  0.28127906,  0.28484648,  0.28841753,  0.29199227,
 0.29557074,  0.299153  ,  0.3027391 ,  0.3063291 ,  0.30992305,
 0.31352101,  0.31712304,  0.32072918,  0.3243395 ,  0.32795405,
 0.33157289,  0.33519607,  0.33882367,  0.34245573,  0.34609231,
 0.34973347,  0.35337928,  0.35702979,  0.36068506,  0.36434516,
 0.36801015,  0.37168008,  0.37535503,  0.37903506,  0.38272022,
 0.38641059,  0.39010623,  0.39380721,  0.39751359,  0.40122544,
 0.40494282,  0.40866581,  0.41239447,  0.41612887,  0.41986909,
 0.42361519,  0.42736724,  0.43112532,  0.43488949,  0.43865984,
 0.44243644,  0.44621936,  0.45000867,  0.45380445,  0.45760679,
 0.46141575,  0.46523142,  0.46905388,  0.4728832 ,  0.47671946,
 0.48056276,  0.48441317,  0.48827077,  0.49213565,  0.4960079 ,
 0.4998876 ,  0.50377484,  0.5076697 ,  0.51157228,  0.51548267,
 0.51940096,  0.52332724,  0.5272616 ,  0.53120414,  0.53515496,
 0.53911414,  0.5430818 ,  0.54705802,  0.55104291,  0.55503657,
 0.5590391 ,  0.56305061,  0.5670712 ,  0.57110098,  0.57514005,
 0.57918853,  0.58324652,  0.58731414,  0.5913915 ,  0.59547872,
 0.59957591,  0.60368318,  0.60780067,  0.61192849,  0.61606676,
 0.62021561,  0.62437516,  0.62854555,  0.63272689,  0.63691932,
 0.64112299,  0.64533801,  0.64956452,  0.65380267,  0.6580526 ,
 0.66231445,  0.66658836,  0.67087448,  0.67517297,  0.67948396,
 0.68380762,  0.6881441 ,  0.69249356,  0.69685616,  0.70123206,
 0.70562143,  0.71002444,  0.71444126,  0.71887206,  0.72331702,
 0.72777631,  0.73225012,  0.73673864,  0.74124205,  0.74576055,
 0.75029432,  0.75484356,  0.75940848,  0.76398929,  0.76858618,
 0.77319937,  0.77782907,  0.7824755 ,  0.78713889,  0.79181947,
 0.79651745,  0.80123308,  0.80596659,  0.81071823,  0.81548825,
 0.8202769 ,  0.82508442,  0.8299111 ,  0.83475719,  0.83962296,
 0.84450869,  0.84941466,  0.85434116,  0.85928849,  0.86425694,
 0.86924681,  0.87425842,  0.87929209,  0.88434814,  0.88942689,
 0.89452869,  0.89965388,  0.90480282,  0.90997585,  0.91517335,
 0.92039569,  0.92564325,  0.93091643,  0.93621562,  0.94154123,
```

```
        0.94689368,  0.9522734 ,  0.95768082,  0.96311639,  0.96858056,
        0.97407381,  0.9795966 ,  0.98514945,  0.99073283,  0.99634727,
        1.0019933 ,  1.00767144,  1.01338227,  1.01912634,  1.02490423,
        1.03071654,  1.03656388,  1.04244688,  1.04836618,  1.05432244,
        1.06031635,  1.06634859,  1.07241989,  1.07853098,  1.08468261,
        1.09087556,  1.09711064,  1.10338867,  1.10971049,  1.11607697,
        1.12248901,  1.12894754,  1.13545351,  1.14200789,  1.14861171,
        1.155266  ,  1.16197185,  1.16873035,  1.17554267,  1.18240998,
        1.18933352,  1.19631454,  1.20335435,  1.21045431,  1.21761582,
        1.22484033,  1.23212934,  1.2394844 ,  1.24690714,  1.25439922,
        1.26196238,  1.26959842,  1.27730922,  1.28509673,  1.29296295,
        1.30091001,  1.30894008,  1.31705546,  1.32525852,  1.33355173,
        1.3419377 ,  1.35041911,  1.35899879,  1.36767969,  1.3764649 ,
        1.38535765,  1.39436132,  1.40347947,  1.41271583,  1.4220743 ,
        1.431559  ,  1.44117426,  1.45092464,  1.46081495,  1.47085025,
        1.4810359 ,  1.49137757,  1.50188124,  1.51255328,  1.52340042,
        1.53442983,  1.54564912,  1.55706641,  1.56869036,  1.58053022,
        1.59259587,  1.60489794,  1.6174478 ,  1.63025771,  1.64334086,
        1.6567115 ,  1.67038506,  1.68437825,  1.69870925,  1.71339788,
        1.72846577,  1.74393663,  1.75983653,  1.77619419,  1.79304141,
        1.81041348,  1.82834975,  1.84689427,  1.8660966 ,  1.88601273,
        1.90670633,  1.92825019,  1.95072808,  1.97423711,  1.99889075,
        2.0248228 ,  2.05219258,  2.08119197,  2.11205508,  2.14507173,
        2.18060696,  2.21912992,  2.26125818,  2.30782877,  2.36001798,
        2.41955673,  2.48915191,  2.57340905,  2.68121219,  2.83371839,
        3.10612952]),
 array([  22,  431,  441,  506,  605,  623,  627,  683,  705,  754,  795,
         801,  822,  920,  959,  981,  985,  986, 1000, 1005, 1011, 1013,
        1027, 1096, 1096, 1098, 1107, 1115, 1162, 1162, 1167, 1204, 1248,
        1263, 1301, 1317, 1321, 1341, 1349, 1360, 1406, 1416, 1421, 1446,
        1450, 1461, 1471, 1472, 1495, 1501, 1510, 1526, 1529, 1530, 1536,
        1538, 1543, 1550, 1562, 1589, 1600, 1605, 1606, 1607, 1623, 1635,
        1650, 1683, 1685, 1685, 1693, 1708, 1712, 1746, 1749, 1787, 1795,
        1796, 1807, 1812, 1815, 1817, 1834, 1842, 1851, 1865, 1872, 1891,
        1913, 1917, 1927, 1944, 1951, 1969, 1977, 1977, 1985, 1996, 2028,
        2034, 2046, 2056, 2077, 2077, 2114, 2115, 2121, 2132, 2133, 2134,
        2162, 2169, 2177, 2192, 2209, 2210, 2227, 2236, 2252, 2277, 2294,
        2298, 2302, 2311, 2368, 2376, 2395, 2402, 2416, 2417, 2423, 2424,
        2424, 2425, 2425, 2429, 2431, 2432, 2455, 2471, 2475, 2485, 2493,
        2496, 2566, 2594, 2633, 2659, 2660, 2689, 2703, 2710, 2729, 2732,
        2739, 2743, 2744, 2765, 2792, 2802, 2808, 2832, 2843, 2895, 2913,
        2914, 2918, 2927, 2933, 2935, 2947, 2999, 3005, 3053, 3068, 3068,
        3071, 3095, 3115, 3117, 3126, 3129, 3141, 3163, 3190, 3194, 3204,
        3214, 3214, 3228, 3239, 3243, 3249, 3267, 3272, 3285, 3292, 3310,
        3322, 3331, 3333, 3348, 3351, 3351, 3368, 3372, 3376, 3387, 3389,
        3392, 3403, 3409, 3422, 3423, 3425, 3429, 3456, 3485, 3487, 3510,
        3520, 3523, 3542, 3544, 3570, 3574, 3577, 3598, 3606, 3613, 3614,
        3620, 3623, 3624, 3641, 3644, 3649, 3659, 3663, 3669, 3709, 3717,
        3727, 3740, 3744, 3747, 3750, 3761, 3767, 3777, 3784, 3784, 3785,
        3786, 3805, 3811, 3820, 3830, 3831, 3840, 3846, 3855, 3867, 3872,
        3873, 3894, 3907, 3910, 3915, 3922, 3926, 3940, 3944, 3956, 3958,
        3959, 3974, 3974, 3982, 4010, 4023, 4035, 4036, 4040, 4046, 4058,
        4066, 4067, 4068, 4073, 4073, 4075, 4086, 4094, 4097, 4098, 4098,
        4105, 4109, 4118, 4120, 4123, 4127, 4128, 4150, 4151, 4153, 4154,
        4169, 4182, 4186, 4187, 4189, 4191, 4195, 4195, 4205, 4220, 4258,
        4266, 4270, 4274, 4274, 4294, 4302, 4304, 4308, 4318, 4322, 4326,
        4332, 4333, 4334, 4338, 4339, 4342, 4352, 4359, 4362, 4363, 4367,
        4375, 4378, 4381, 4390, 4400, 4401, 4401, 4433, 4451, 4456, 4458,
        4459, 4459, 4460, 4475, 4484, 4486, 4492, 4507, 4509, 4511, 4521,
        4539, 4541, 4548, 4549, 4553, 4563, 4569, 4570, 4575, 4576, 4579,
        4585, 4586, 4590, 4592, 4595, 4602, 4608, 4629, 4630, 4634, 4639,
        4648, 4649, 4649, 4656, 4660, 4661, 4665, 4669, 4672, 4677, 4679,
        4687, 4694, 4708, 4713, 4714, 4717, 4725, 4727, 4744, 4748, 4758,
        4758, 4760, 4763, 4765, 4773, 4780, 4785, 4788, 4790, 4792, 4795,
        4803, 4826, 4833, 4835, 4839, 4840, 4844, 4845, 4862, 4864, 4866,
        4881, 4891, 4905, 4906, 4911, 4916, 4917, 4940, 4966, 4968, 4972,
        4978, 4985, 4990, 4991, 4996, 5008, 5010, 5020, 5026, 5035, 5041,
        5046, 5047, 5058, 5062, 5084, 5087, 5099, 5102, 5107, 5115, 5115,
        5117, 5119, 5119, 5130, 5138, 5146, 5169, 5170, 5180, 5191, 5191,
        5202, 5202, 5204, 5217, 5225, 5255, 5259, 5260, 5260, 5267, 5298,
        5302, 5305, 5312, 5312, 5315, 5319, 5323, 5336, 5342, 5345, 5362,
        5375, 5382, 5409, 5409, 5423, 5424, 5445, 5459, 5463, 5464, 5478,
        5495, 5499, 5501, 5511, 5515, 5531, 5532, 5538, 5557, 5558, 5566,
        5572, 5582, 5585, 5611, 5629, 5633, 5634, 5668, 5686, 5687, 5698,
        5698, 5713, 5728, 5729, 5740, 5743, 5786, 5805, 5810, 5823, 5847,
        5847, 5870, 5875, 5892, 5895, 5905, 5918, 5923, 5936, 5976, 5986,
        5992, 6031, 6034, 6041, 6043, 6043, 6053, 6073, 6093, 6118, 6133,
```

```
       6140, 6153, 6169, 6192, 6196, 6203, 6207, 6211, 6227, 6230, 6233,
       6234, 6235, 6241, 6269, 6273, 6290, 6296, 6299, 6304, 6312, 6359,
       6370, 6392, 6398, 6421, 6436, 6457, 6460, 6530, 6536, 6536, 6544,
       6565, 6569, 6572, 6591, 6591, 6597, 6598, 6606, 6624, 6639, 6660,
       6664, 6685, 6691, 6734, 6770, 6772, 6778, 6779, 6784, 6786, 6824,
       6824, 6825, 6830, 6852, 6855, 6857, 6861, 6864, 6869, 6871, 6879,
       6883, 6883, 6889, 6891, 6904, 6917, 6966, 6969, 6978, 6998, 7001,
       7006, 7013, 7030, 7040, 7055, 7058, 7105, 7109, 7112, 7129, 7132,
       7148, 7175, 7216, 7261, 7264, 7273, 7282, 7286, 7290, 7328, 7333,
       7335, 7338, 7347, 7350, 7359, 7363, 7375, 7384, 7393, 7403, 7410,
       7415, 7421, 7424, 7429, 7436, 7442, 7444, 7446, 7458, 7460, 7461,
       7466, 7494, 7498, 7499, 7504, 7509, 7525, 7534, 7534, 7538, 7570,
       7572, 7580, 7582, 7591, 7592, 7605, 7639, 7641, 7665, 7691, 7693,
       7697, 7702, 7713, 7720, 7733, 7736, 7765, 7767, 7804, 7836, 7852,
       7865, 7870, 7907, 7965, 8009, 8090, 8120, 8156, 8167, 8173, 8227,
       8294, 8362, 8395, 8555, 8714], dtype=int64)),
 (1925.1274361641943, 4504.3488372093025, 0.9908084868276723))
```



As we can see, out cnt variable is very close to normal distribution.

Preprocessing original data and Spliting into train and test data

In [41]:

```
# Rollback understandable text to numbers
season_dict = { 'Spring' :'1', 'Summer' :'2', 'Fall' : '3', 'Winter' : '4'}
weather_dict = { 'Clear' :'1', 'Misty+Cloudy' : '2', 'Light Snow/Rain' : '3', 'Heavy Snow/Rain' : '
4'}
month_dict = {'Jan' : '1', 'Feb' : '2', 'Mar' :'3', 'Apr' : '4', 'May' : '5', 'June' :'6', 'July' :
'7', 'Aug' : '8', 'Sep' : '9', 'Oct' : '10', 'Nov' : '11', 'Dec' :'12'}
year_dict = { '2011' : '0', '2012' : '1'}
bikedata['Season'] = bikedata['Season'].map(season_dict)
bikedata['Weather'] = bikedata['Weather'].map(weather_dict)
bikedata['Month'] = bikedata['Month'].map(month_dict)
bikedata['Year'] = bikedata['Year'].map(year_dict)

bikedata.head()
```

Out[41]:

	Season	Year	Month	Holiday	Weekday	Workingday	Weather	Temperature	Atemperature	Humidity	Windspeed	Casual Users	Registe Us
0	1	0	1	0	6	0	2	0.344167	0.363625	0.805833	0.160446	331	[illegible]
1	1	0	1	0	0	0	2	0.363478	0.353739	0.696087	0.248539	131	[illegible]
2	1	0	1	0	1	1	1	0.196364	0.189405	0.437273	0.248309	120	[illegible]
3	1	0	1	0	2	1	1	0.200000	0.212122	0.590435	0.160296	108	[illegible]
4	1	0	1	0	3	1	1	0.226957	0.229270	0.436957	0.186900	82	[illegible]

In [42]:

```
#selecting predictors
train_feature_space = bikedata.iloc[:,bikedata.columns != 'Count']
# selecting target class
target_class = bikedata.iloc[:,bikedata.columns == 'Count']
```

In [43]:

```
#droping Atemperature due to multicollinearity
#droping Casual Users and Registered Users because there sum is equal to target variable ie. 'Coun
t'

train_feature_space = train_feature_space.drop(["Atemperature","Casual Users","Registered Users"],
axis = 1)
```

In [44]:

```
train_feature_space.shape
```

Out[44]:

```
(731, 10)
```

In [45]:

```
train_feature_space.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 731 entries, 0 to 730
Data columns (total 10 columns):
Season         731 non-null object
Year           731 non-null object
Month          731 non-null object
Holiday        731 non-null category
Weekday        731 non-null category
Workingday     731 non-null category
Weather        731 non-null object
Temperature    731 non-null float64
Humidity       731 non-null float64
Windspeed      731 non-null float64
dtypes: category(3), float64(3), object(4)
memory usage: 42.8+ KB
```

In [46]:

```
# creating training and test set
training_set, test_set, train_target, test_target = train_test_split(train_feature_space,
                                                                     target_class,
                                                                     test_size = 0.30,
                                                                     random_state = 456)

# Cleaning test sets to avoid future warning messages
train_target = train_target.values.ravel()
test_target = test_target.values.ravel()
```

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.

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

In [92]:

```
# Initialize linear regression model
Model = LinearRegression()
Model.fit(X = training_set,y = np.log(train_target))
```

Out[92]:

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)
```

In [95]:

```
#predicting using linear regression
bikedata_Predictions = Model.predict(X=test_set)
bikedata=pd.DataFrame(np.exp(bikedata_Predictions))
bikedata.describe()
```

Out[95]:

	0

	0
count	220.000000
mean	4438.291596
std	2120.623071
min	1034.553984
25%	2730.336107
50%	4096.304410
75%	5708.079487
max	11000.217123

In [96]:

```
bikedata_errors = abs(np.exp(bikedata_Predictions) - test_target)
# Print out the mean absolute error (mae)
print('Mean Absolute Error:', round(np.mean(bikedata_errors), 2), 'degrees.')
```

```
Mean Absolute Error: 899.5 degrees.
```

In [97]:

```
rmse = sqrt(mean_squared_error(test_target, np.exp(bikedata_Predictions)))
print("RMSE for test set in linear regression is :" , rmse)
```

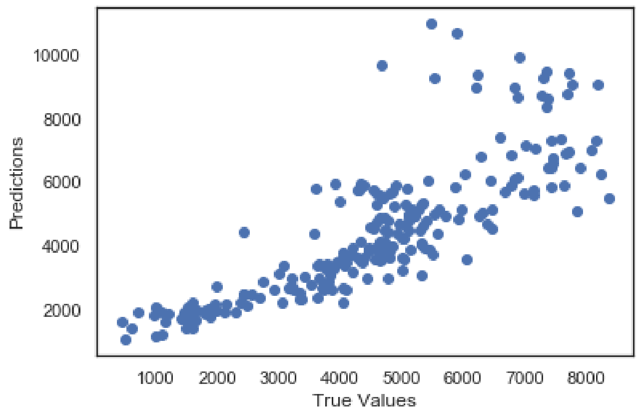
```
RMSE for test set in linear regression is : 1222.1581373120362
```

In [98]:

```
## The line / model
plt.scatter(test_target, np.exp(bikedata_Predictions))
plt.xlabel("True Values")
plt.ylabel("Predictions")
```

Out[98]:

```
Text(0, 0.5, 'Predictions')
```



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)
```

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
```

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
```

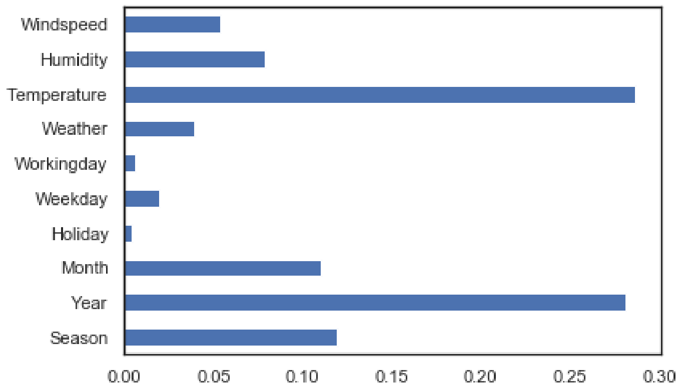
Variable importance for random forest

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>
```



In [109]:

```
#model input and output
pd.DataFrame(test_set).to_csv('InputLinearRegressionRandomForestPyhon.csv', index = False)
pd.DataFrame(np.exp(bikedata_Predictions),
columns=['predictions']).to_csv('outputLinearRegressionPython.csv')
pd.DataFrame(rfPredictions, columns=['predictions']).to_csv('outputRandomForestPython.csv')
```