8/3/2018

Bike Renting Prediction

Rohit Haritash
ROHIT.HARITASH@GMAIL.COM

Table of Contents

Chapter 1

Introduction

Whether it's to boost your fitness, health or bank balance, or as an environmental choice, taking up bicycle riding could be one of the best decisions you ever make. Remember the days of the bicycle built for two, when tourists rented bikes to explore island areas where cars either didn't exist or were blessedly limited? Those days are still here, but the majority of bicycle rental businesses are clustered around heavily trafficked tourist spots.

However, with increased rails-to-trails projects and traffic congestion there are many more bicycle paths away from resort areas, office space, residential area that are creating excellent new rental opportunities. Many bicycle rental shops are now featuring inline skate rentals as well, especially in places like USA, UK. The bike rental service has a great potential as a business opportunity.

1.1 Problem Statement

The objective of this case study is the prediction of bike rental count on daily based on the environmental and seasonal settings. The dataset contains 731 observations, 15 predictors and 1 target variable. The predictors are describing various environment factors and settings like season, humidity etc. We need to build a prediction model to predict estimated count or demand of bikes on a particular day based on the environmental factors.

1.2 Dataset

The data set consist of 731 observation recorded over a period of 2 years, between 2011 and 2012. It has 15 predictors or variables and 1 target variable. All the variables are described in table 1.

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	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- t_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

Table1. Description of variables

The data set consist of 7 continuous and 8 categorical variables. Sample data is shown below.

instant	dteday	season	yr	mnth	holiday	weekday	workingday	weathersit
1	1/1/2011	1	0	1	0	6	0	2
2	1/2/2011	1	0	1	0	0	0	2
3	1/3/2011	1	0	1	0	1	1	1
4	1/4/2011	1	0	1	0	2	1	1
5	1/5/2011	1	0	1	0	3	1	1
6	1/6/2011	1	0	1	0	4	1	1

temp	atemp	hum	windspeed	casual	registered	cnt
0.344167	0.363625	0.805833	0.160446	331	654	985
0.363478	0.353739	0.696087	0.248539	131	670	801
0.196364	0.189405	0.437273	0.248309	120	1229	1349
0.2	0.212122	0.590435	0.160296	108	1454	1562
0.226957	0.22927	0.436957	0.1869	82	1518	1600
0.204348	0.233209	0.518261	0.0895652	88	1518	1606

Table2. Sample data

Chapter 2

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 and EDA

Pre-processing and EDA was performed in both R and python. The dataset consists of 731 observations, and 15 predictors. The process of pre-processing and EDA is described below.

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. '*cnt'* is the combined value of *'registered'* and *'casual'* variables. The histogram, distribution and summary statistics of *'cnt'* are as follow.

Summary Stats	Values
count	731
mean	4504.34
std	1937
min	22
25%	3152
50%	4548
75%	5956
max	8714

Table3. Summary statistics of target variable '*cnt'*

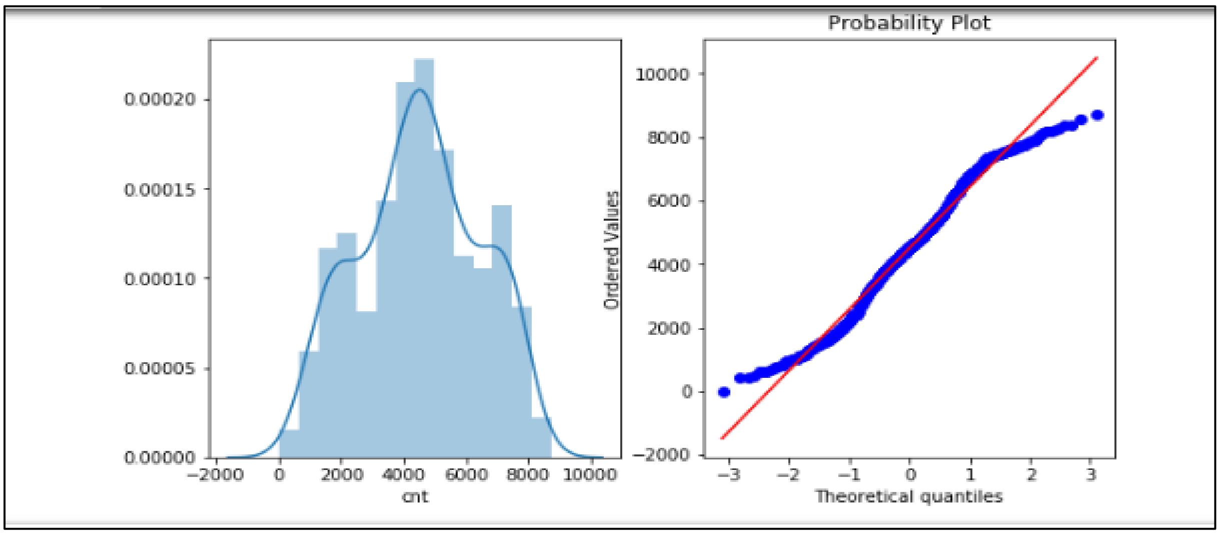


Figure1. Target variable distribution

From the figure it is can be seen that the target variable is close to normal distribution.

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.

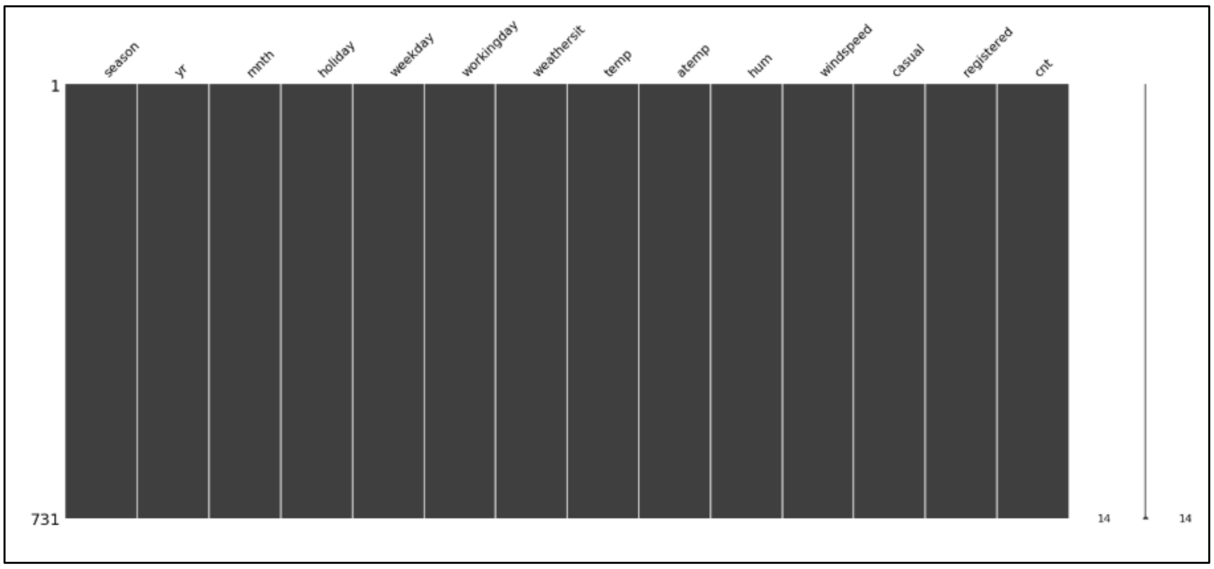


Figure2. Missing values distribution

2.1.3 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.

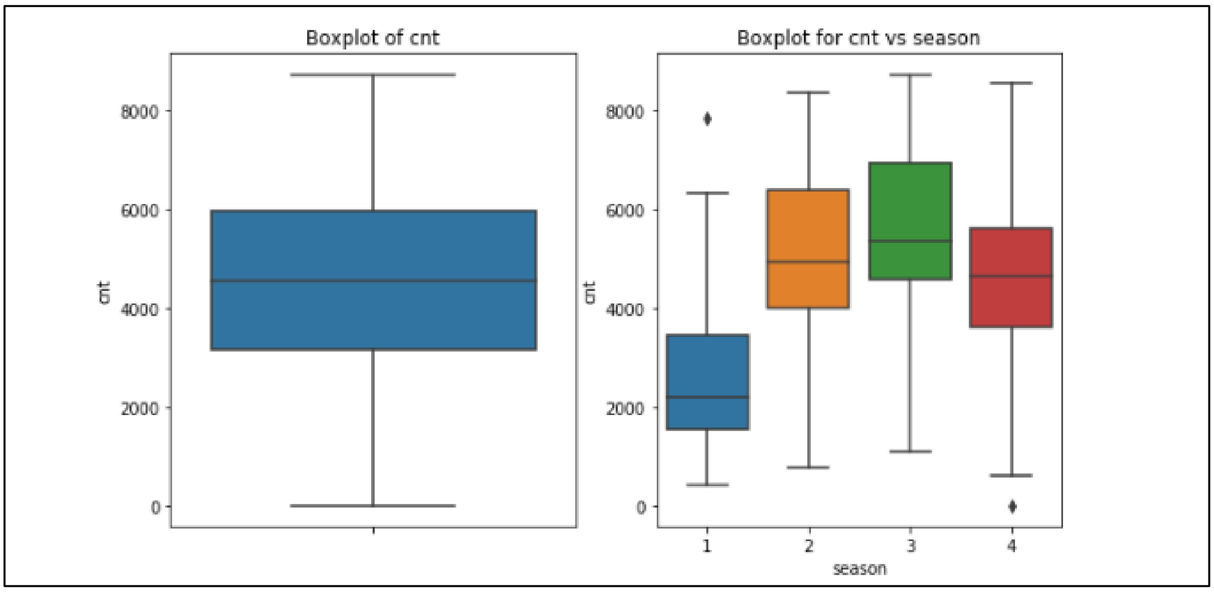


Figure3. Boxplot for cnt and boxplot for cnt vs season

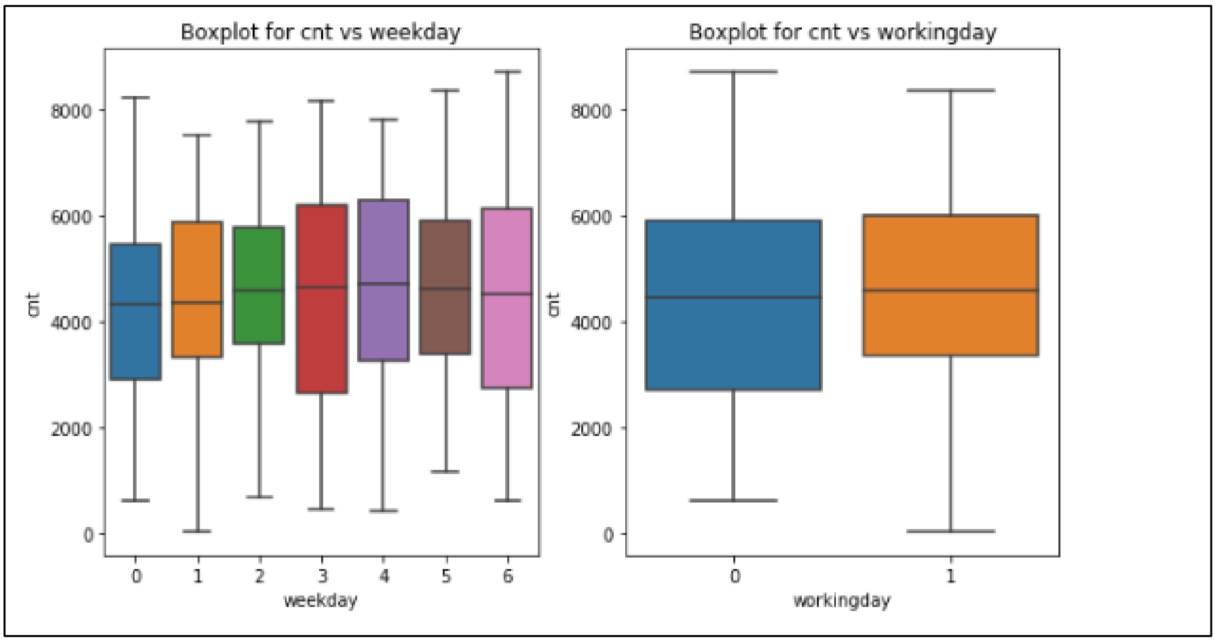


Figure4. Boxplot of cnt vs weekday and boxplot of cnt vs working day

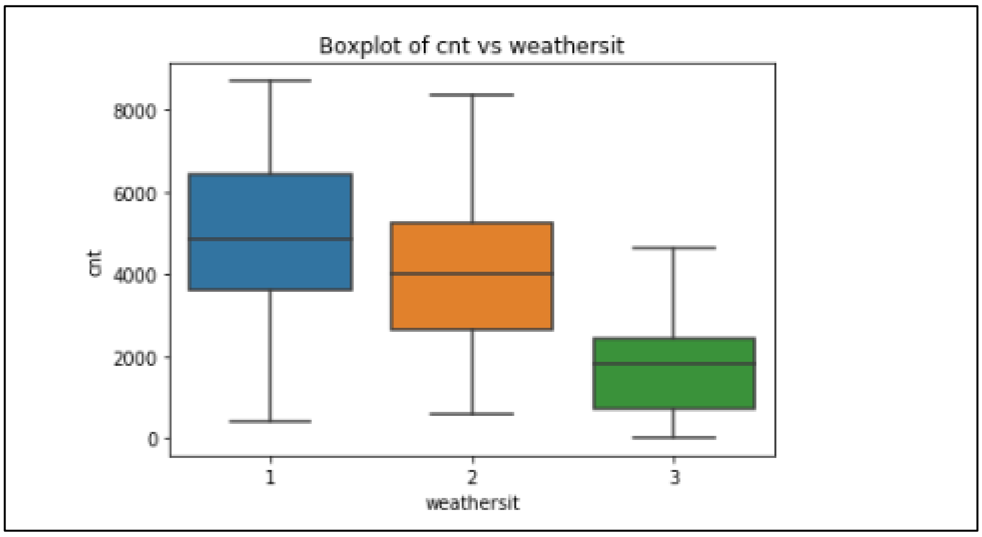


Figure5. Boxplot of cnt vs weathersit

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

1. 1.Bike demand count ('cnt') is low in spring (1) season.
2. There is no effect on bike count('cnt') due to a holiday or a working day.
3. Bikes are rented mostly in good weather (1: Clear, Few clouds, Partly cloudy, Partly cloudy) and least in bad (3: Light Snow, Light Rain + Thunderstorm + Scattered clouds, Light Rain + Scattered clouds) weather.

2.1.4 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.

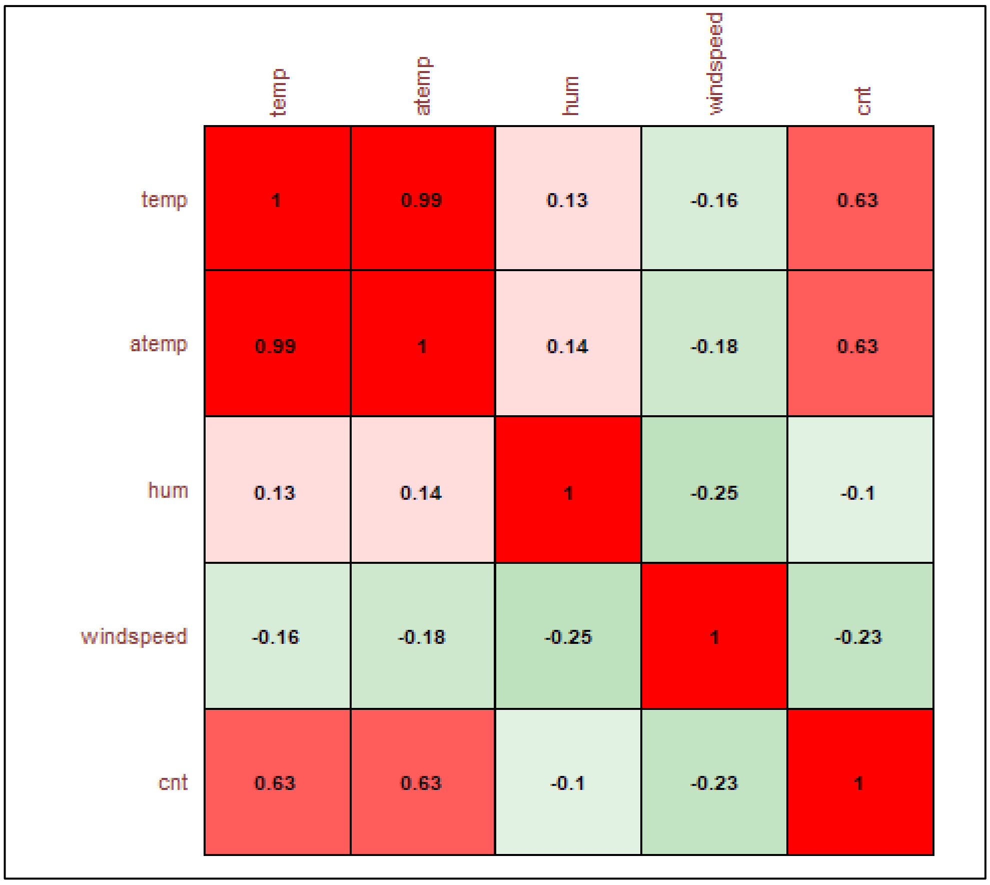


Figure6. Correlation matrix

‘*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

1. Temp (temperature)and atemp (ambient temperature) are highly collinear. One of them should be removed before modeling.
2. ‘cnt’ (demand 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.
3. ‘cnt’ (demand count) is negative relationship with hum(humidity) and windspeed. People tends to rent bike more when there is less humidity and wind speed.
4. Also the relationship between ‘hum’, ’windspeed’ and ‘cnt’ is very weak. These are not very strong predictors.

Python’s implementation of correlation matrix is in appendix.

2.1.5 Univariate analysis

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

a. temp

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

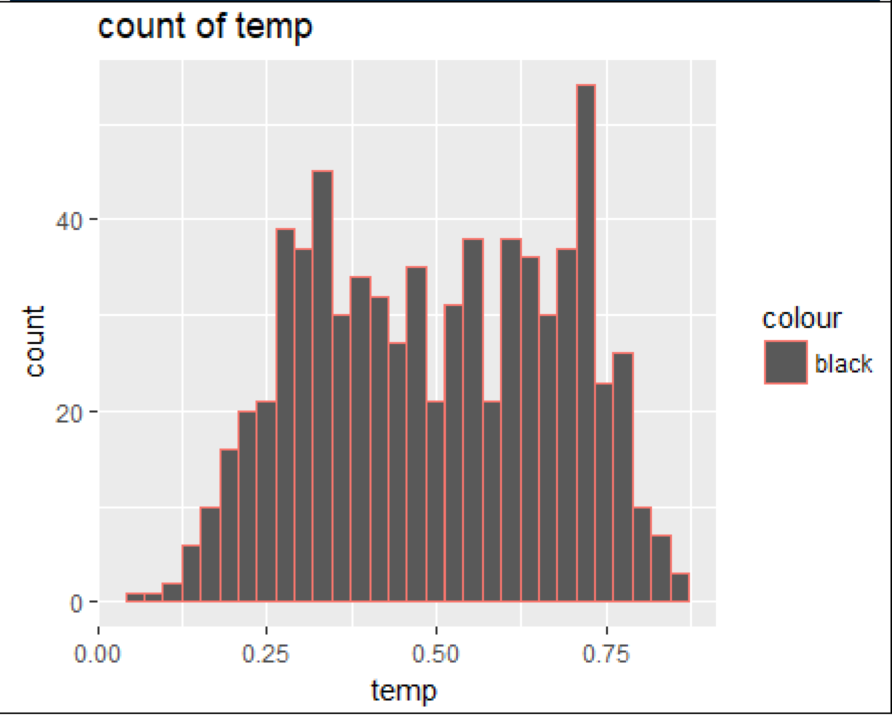


Table4. Univariate analysis of temp row1. Summary stats row2. Distribution

b. atemp

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

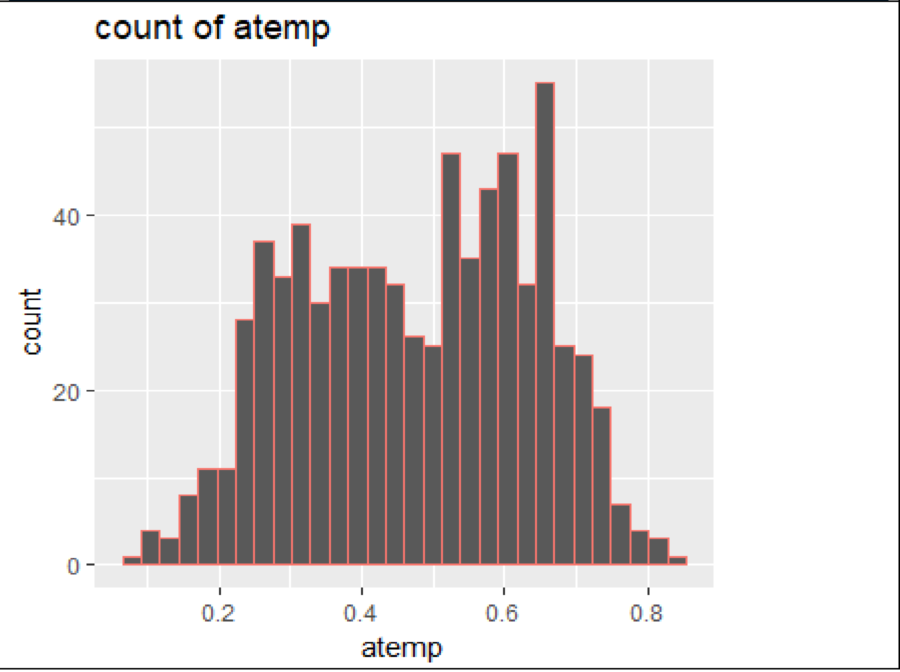
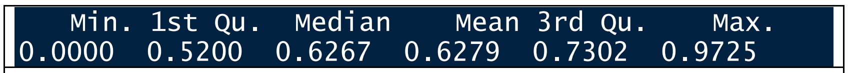


Table5. Univariate analysis of atemp row1. Summary stats row2. Distribution

c. hum



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

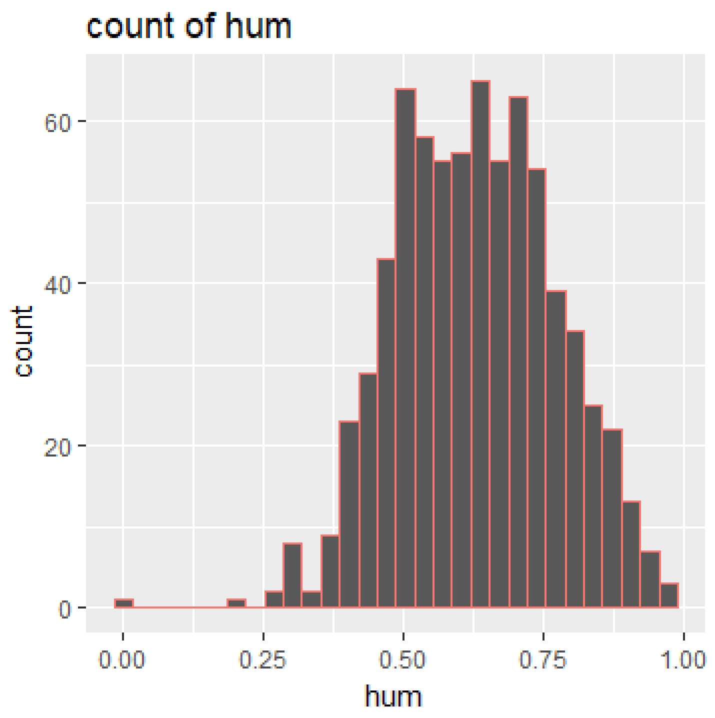
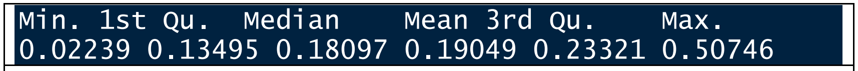


Table6. Univariate analysis of hum(humidity) row1. Summary stats row2. Distribution

d. windspeed



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

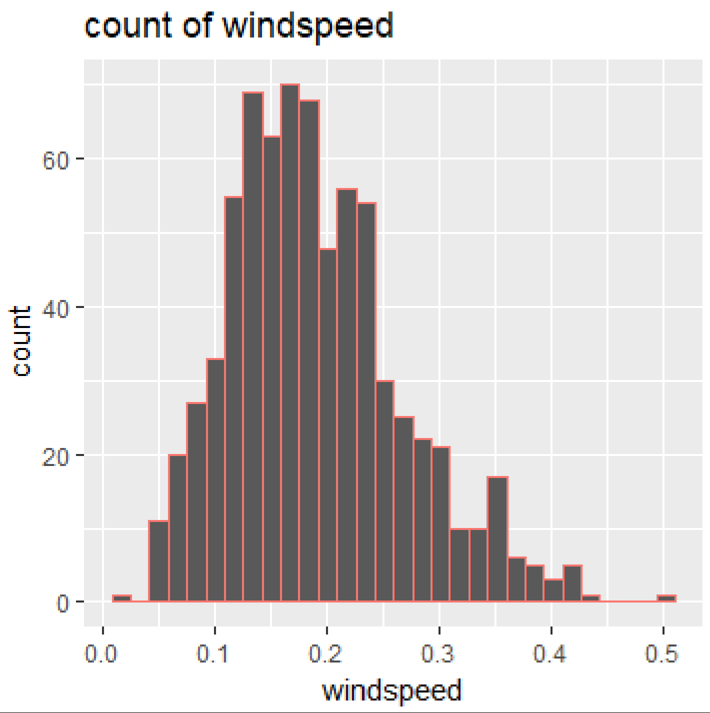
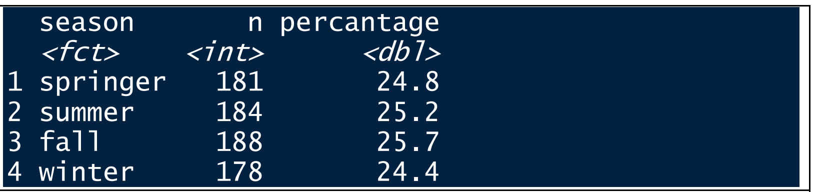


Table7. Univariate analysis of wind summary row1. Summary stats row2. Distribution

e. Seasons



```
  season        n percantage
  <fct>     <int>      <dbl>
1 springer    181       24.8
2 summer      184       25.2
3 fall        188       25.7
4 winter      178       24.4
```

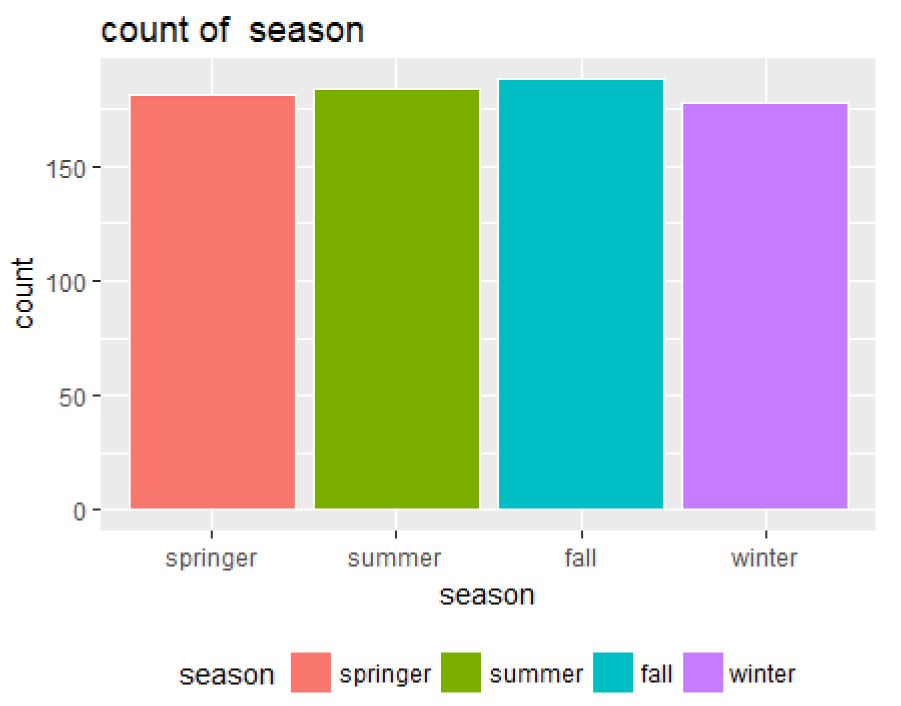
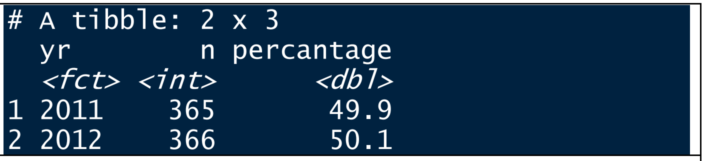


Table8. Univariate analysis of season row1. Summary stats row2. Count

f. Yr (year)



```
# A tibble: 2 x 3
  yr        n percantage
  <fct> <int>      <dbl>
1 2011    365       49.9
2 2012    366       50.1
```

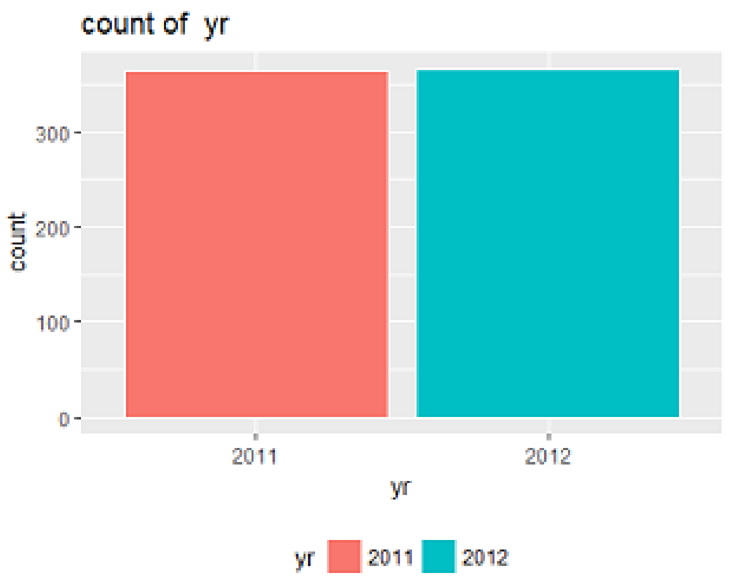


Table9. Univariate analysis of year row1. Summary stats row2. Count

g. Mnth (month)

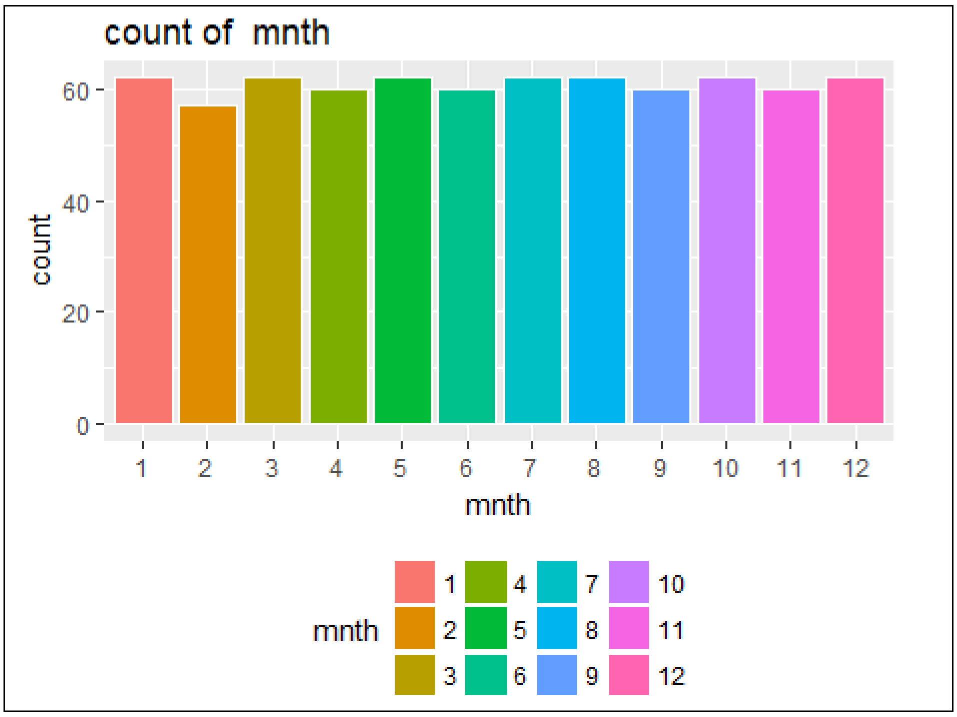


Figure6. count of months

h. Weekday

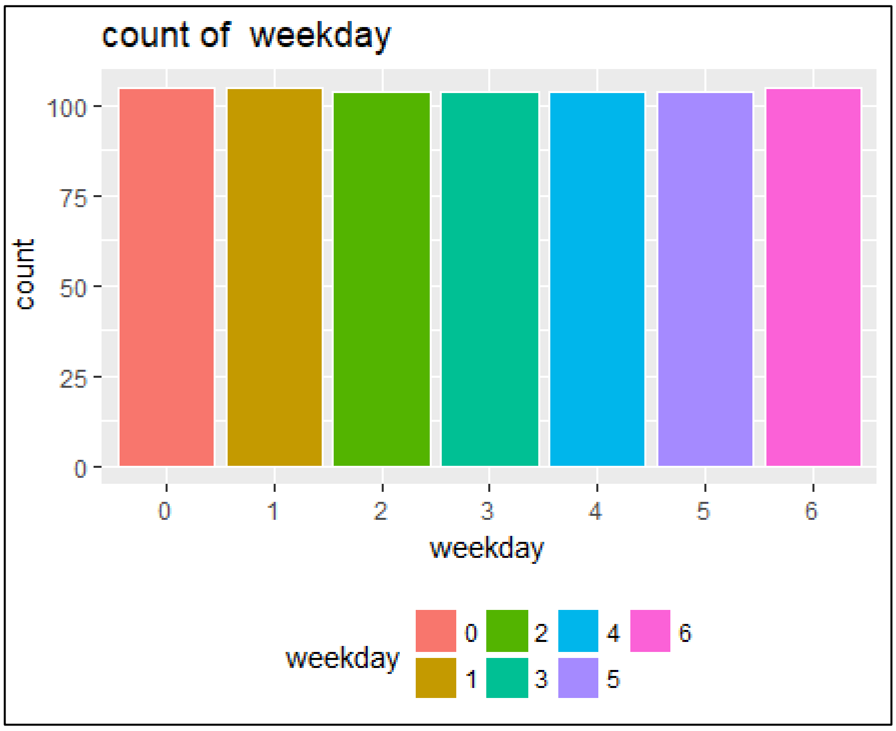


Figure7. count of weekdays

i. Working day

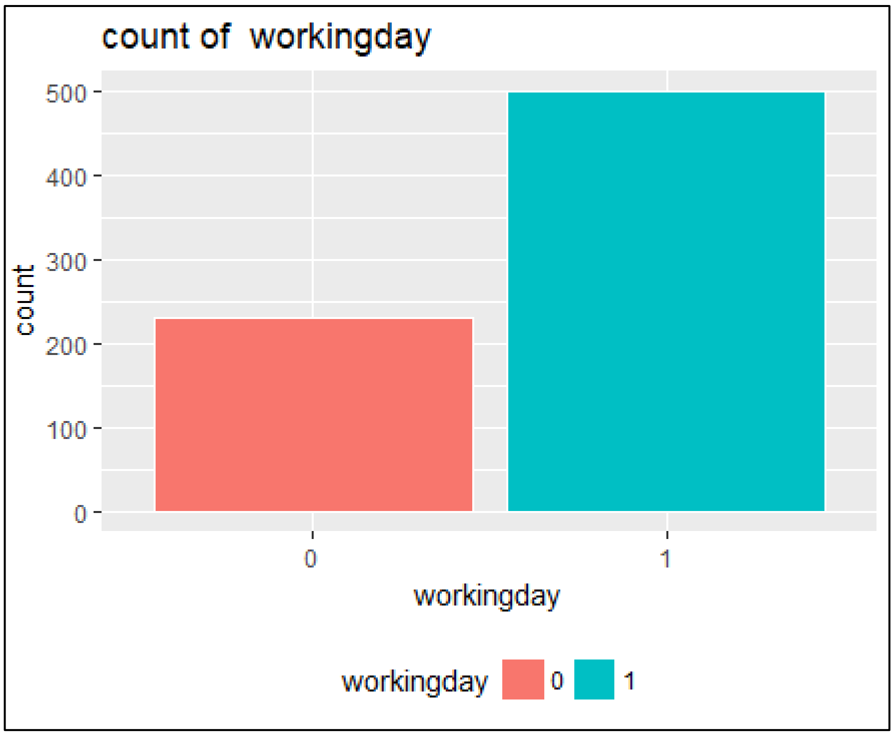


Figure8. count of working days

j. Weathersit (weather situation)

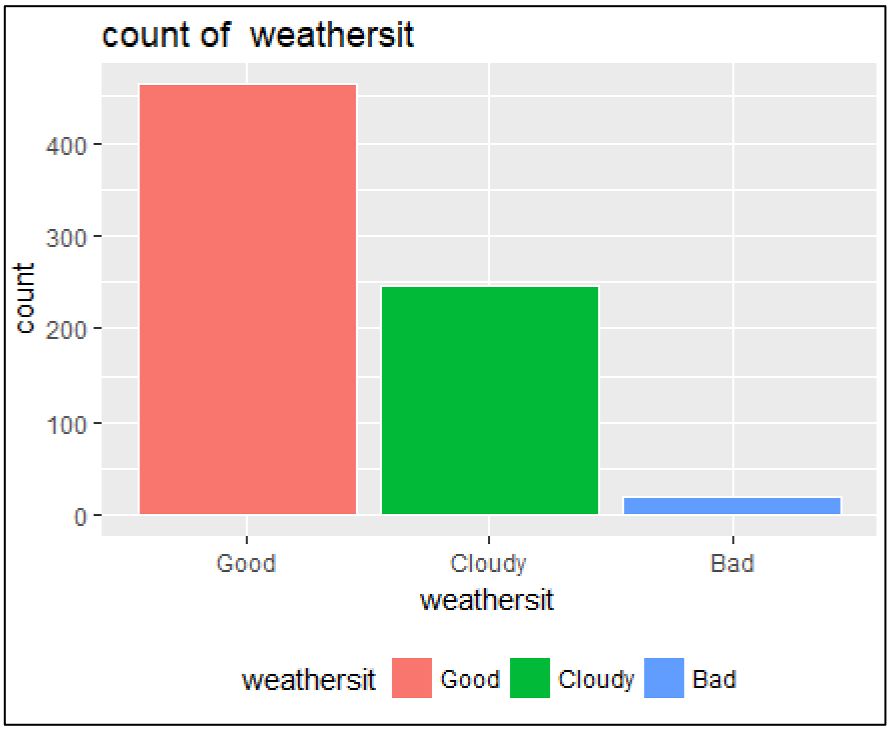


Figure9. count of weather situation

2.1.6 Bivariate Analysis

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

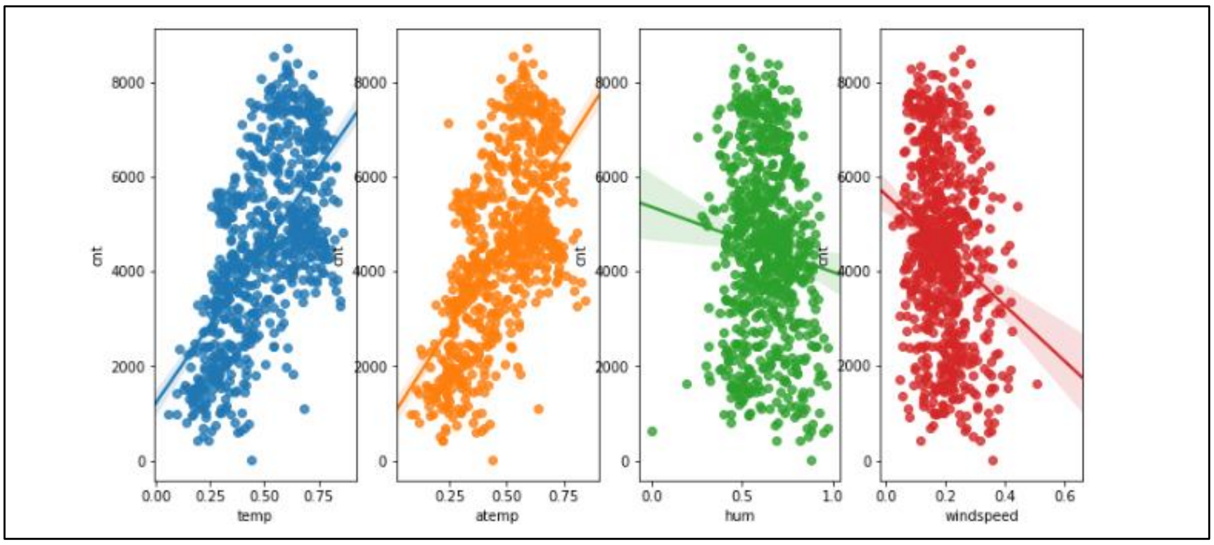


Figure 10. relationship between target variable and continuous predictors

From the above scatter plots, we can see that

1. 'cnt' and 'temp' have strong and positive relationship. It means that as the temperature rises, the bike demand also increase.
2. 'atemp' and 'cnt' have strong and positive relationship. It means that as the ambient temperature rise, demand for bikes also increases.
3. 'hum' (humidity) has a negative linear relationship with 'cnt'. As humidity increases, count decreases.
4. 'windspeed' has negative linear relationship with 'cnt'. With an increase in windspeed, bike count decreases.

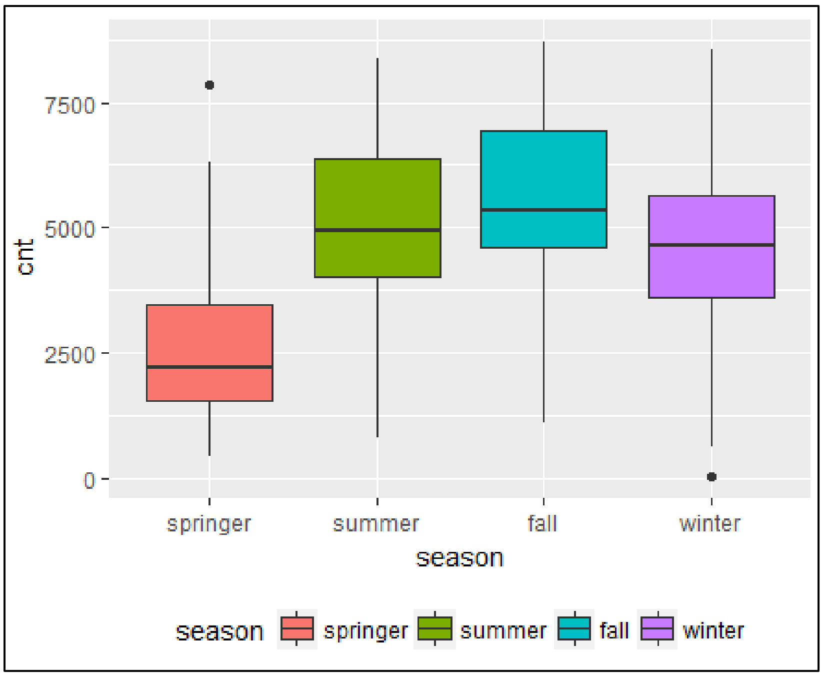


Figure 11. relationship between season and count

Figure 11 is showing relationship between count (demand) and season.

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

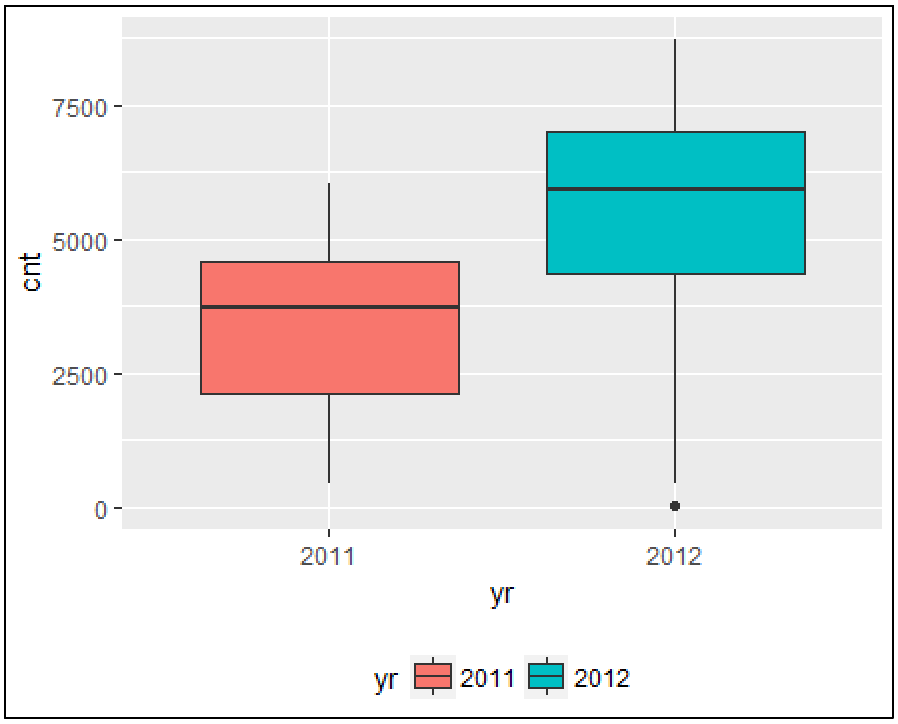
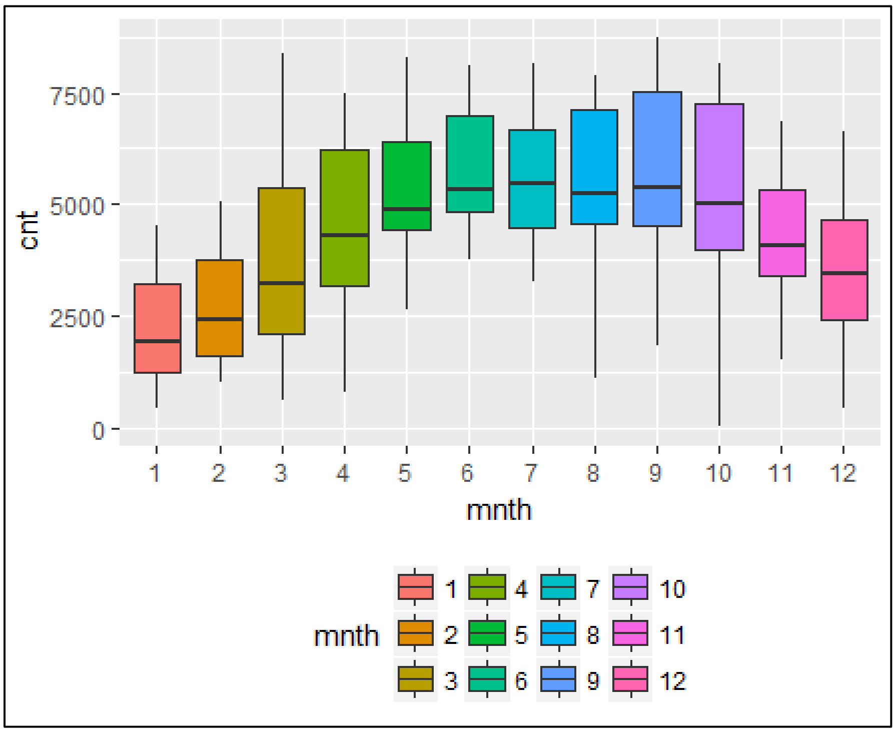


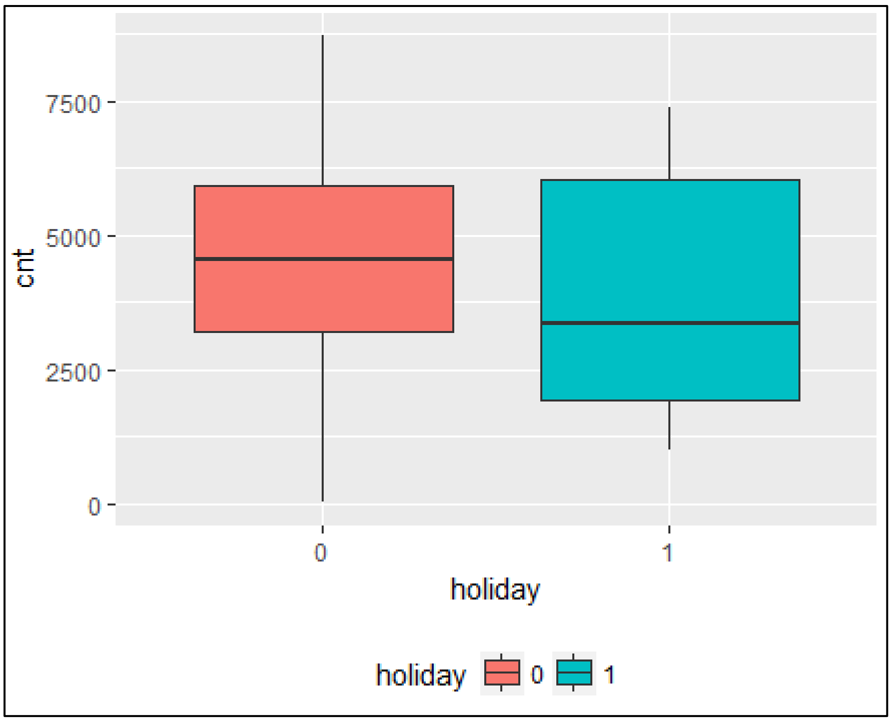
Figure 12 is showing that bike demand was higher in 2012 as compared with 2011.

Figure 12. relationship between year and count



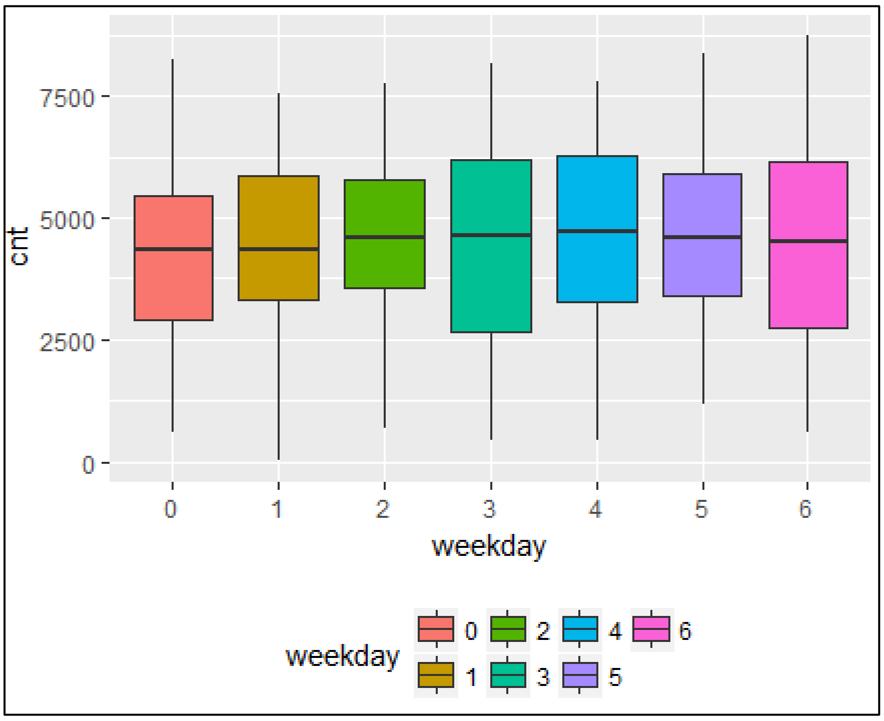
1. From figure 13 it can be inferred that count is high in the month of august, September and October.
2. It is lowest count is for January ad February.
3. We can see that as the weather changes from cold to hot, count also

Figure 13. relationship between months 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.

Figure 14. relationship between holidays and count



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

Figure 15. relationship between weekdays and count

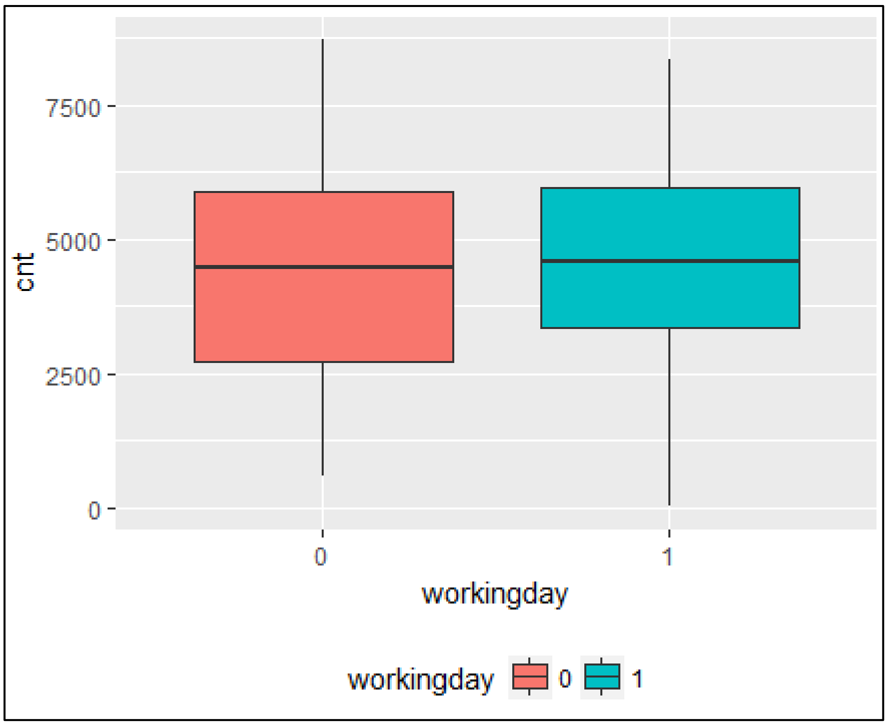


Figure 16. relationship between workingday and count

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

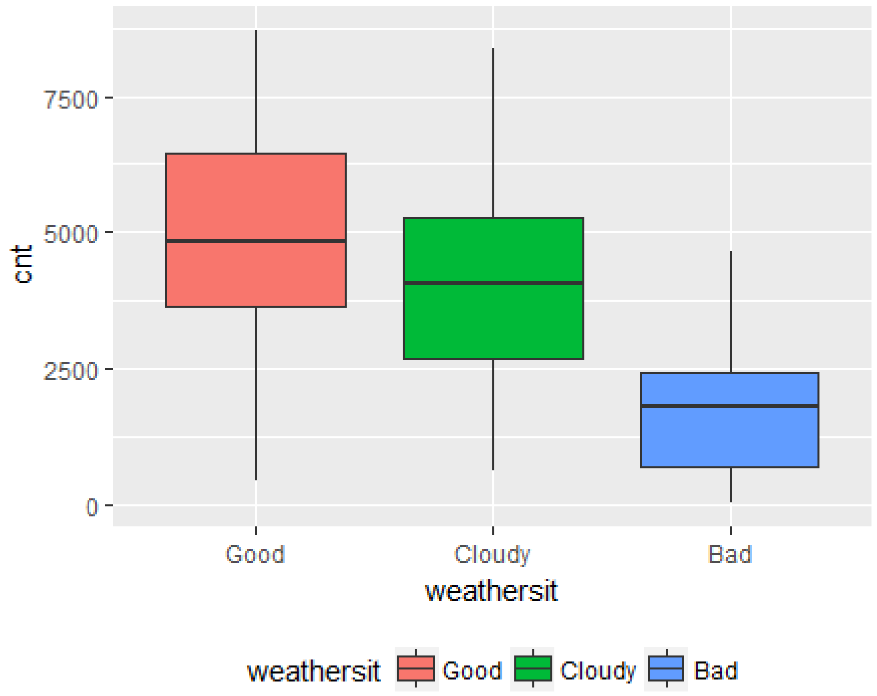


Figure 17. relationship between weathersit and count

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

2.1.7 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.

2.2 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, hyperparameter tuning was used.

2.2.1 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.

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

Residuals:
    Min      1Q  Median      3Q     Max
-3518.4  -351.7    63.2   426.9  2439.2

Coefficients: (1 not defined because of singularities)
                 Estimate Std. Error t value Pr(>|t|)
(Intercept)       1465.98     296.99   4.936 1.10e-06 ***
seasonsummer      1052.92     199.53   5.277 1.99e-07 ***
seasonfall        1090.08     243.02   4.486 9.10e-06 ***
```

```
seasonwinter      1739.77     209.32   8.312 9.69e-16 ***
yr2012            2056.81      68.91  29.846  < 2e-16 ***
mnth2              207.30     170.46   1.216 0.224539    
mnth3              505.95     195.71   2.585 0.010026 *  
mnth4              468.06     284.54   1.645 0.100636    
mnth5              914.46     310.98   2.941 0.003433 ** 
mnth6              695.86     331.01   2.102 0.036053 *  
mnth7               74.15     371.79   0.199 0.842003    
mnth8              537.96     360.50   1.492 0.136280    
mnth9              954.10     310.59   3.072 0.002247 ** 
mnth10             611.19     285.71   2.139 0.032920 *  
mnth11             -77.42     268.33  -0.289 0.773079    
mnth12            -149.70     210.42  -0.711 0.477154    
holiday1          -477.98     226.35  -2.112 0.035225 *  
weekday1            84.20     132.11   0.637 0.524209    
weekday2           216.58     127.58   1.698 0.090235 .  
weekday3           349.06     126.07   2.769 0.005844 ** 
weekday4           304.14     125.41   2.425 0.015670 *  
weekday5           374.08     125.54   2.980 0.003030 ** 
weekday6           342.14     124.81   2.741 0.006346 ** 
workingday1            NA         NA      NA       NA    
weathersitCloudy  -409.63      94.17  -4.350 1.66e-05 ***
weathersitBad    -2041.38     235.09  -8.684  < 2e-16 ***
temp              2548.79    1484.30   1.717 0.086591 .  
atemp             1571.57    1525.89   1.030 0.303554    
hum              -1423.48     367.58  -3.873 0.000123 ***
windspeed        -2611.79     496.43  -5.261 2.16e-07 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 754.9 on 482 degrees of freedom
Multiple R-squared:  0.8561,    Adjusted R-squared:  0.8477 
F-statistic: 102.4 on 28 and 482 DF,  p-value: < 2.2e-16
```

Table 10. model1 summary

In model1 all the predictors were included. Temp and atemp were multicollinear. They were also included in the model. The adjusted r square value was .84 which is a good value with F-statistic 102.4.

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

```
Call:
lm(formula = cnt ~ season + yr + mnth + holiday + weekday + weathersit + 
    temp + hum + 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 ***
yr2012            2054.43      68.88  29.826  < 2e-16 ***
mnth2              211.07     170.44   1.238 0.216161    
mnth3              505.08     195.72   2.581 0.010158 *  
mnth4              471.39     284.54   1.657 0.098240 .  
mnth5              897.34     310.55   2.889 0.004032 ** 
mnth6              667.54     329.89   2.024 0.043568 *  
```

```
mnth7              53.63     371.28   0.144 0.885217
mnth8             488.20     357.27   1.366 0.172427
mnth9             928.93     309.64   3.000 0.002839 **
mnth10            612.68     285.72   2.144 0.032506 *
mnth11            -71.15     268.27  -0.265 0.790960
mnth12           -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 **
weathersitCloudy  -412.23      94.14  -4.379 1.46e-05 ***
weathersitBad    -2059.08     234.47  -8.782  < 2e-16 ***
temp              3986.92     503.44   7.919 1.65e-14 ***
hum              -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
```

Table 11. Summary of AICModel

For second model, adjusted R-square was same, but a slight increase in F-statistics. It shows that this model has better relevant features. There was an issue with this model. If you look at the summary of model, it can be seen that there are negative predictions. We don't want negative values since they don't make any sense.

So to counter this, third model was trained with log transformation of target variable. The summary of third model is given below.

```
Call:
lm(formula = log(cnt) ~ season + yr + mnth + holiday + weathersit +
    temp + hum + windspeed, data = training_set)

Residuals:
    Min      1Q  Median      3Q     Max
-4.3758 -0.0998  0.0336  0.1445  0.9327

Coefficients:
                 Estimate Std. Error t value Pr(>|t|)
(Intercept)       7.51764    0.11423  65.814  < 2e-16 ***
seasonsummer      0.34336    0.08159   4.208 3.06e-05 ***
seasonfall        0.41883    0.09910   4.226 2.83e-05 ***
seasonwinter      0.64784    0.08562   7.566 1.92e-13 ***
yr2012            0.45360    0.02810  16.143  < 2e-16 ***
mnth2             0.12485    0.06974   1.790   0.0740 .
mnth3             0.13220    0.07991   1.654   0.0987 .
mnth4             0.04304    0.11588   0.371   0.7105
mnth5             0.09349    0.12656   0.739   0.4604
mnth6            -0.09060    0.13407  -0.676   0.4995
mnth7            -0.31026    0.15077  -2.058   0.0401 *
mnth8            -0.20306    0.14516  -1.399   0.1625
mnth9            -0.05415    0.12595  -0.430   0.6675
mnth10           -0.12779    0.11680  -1.094   0.2744
mnth11           -0.08442    0.10967  -0.770   0.4418
mnth12           -0.11101    0.08601  -1.291   0.1974
holiday1         -0.20572    0.08900  -2.311   0.0212 *
weathersitCloudy -0.04933    0.03825  -1.290   0.1978
```

```
weathersitBad    -0.90159    0.09560  -9.431  < 2e-16 ***
temp              1.70632    0.20373   8.375 5.86e-16 ***
hum              -0.59965    0.14895  -4.026 6.58e-05 ***
windspeed        -0.96584    0.19933  -4.845 1.70e-06 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 0.3092 on 489 degrees of freedom
Multiple R-squared:  0.7364,    Adjusted R-squared:  0.7251
F-statistic: 65.04 on 21 and 489 DF,  p-value: < 2.2e-16

> summary(predict_test_nonlog)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
    486    3063    4381    4484    5822   10614
```

Table 12. Summary of stepwiseLogAICModel

In stepwiseLogAICModel model, we lost some adjusted R-square. But after prediction, when the values were converted back using exponential transformation, we get all the positive prediction.

The summary of predictions are as below

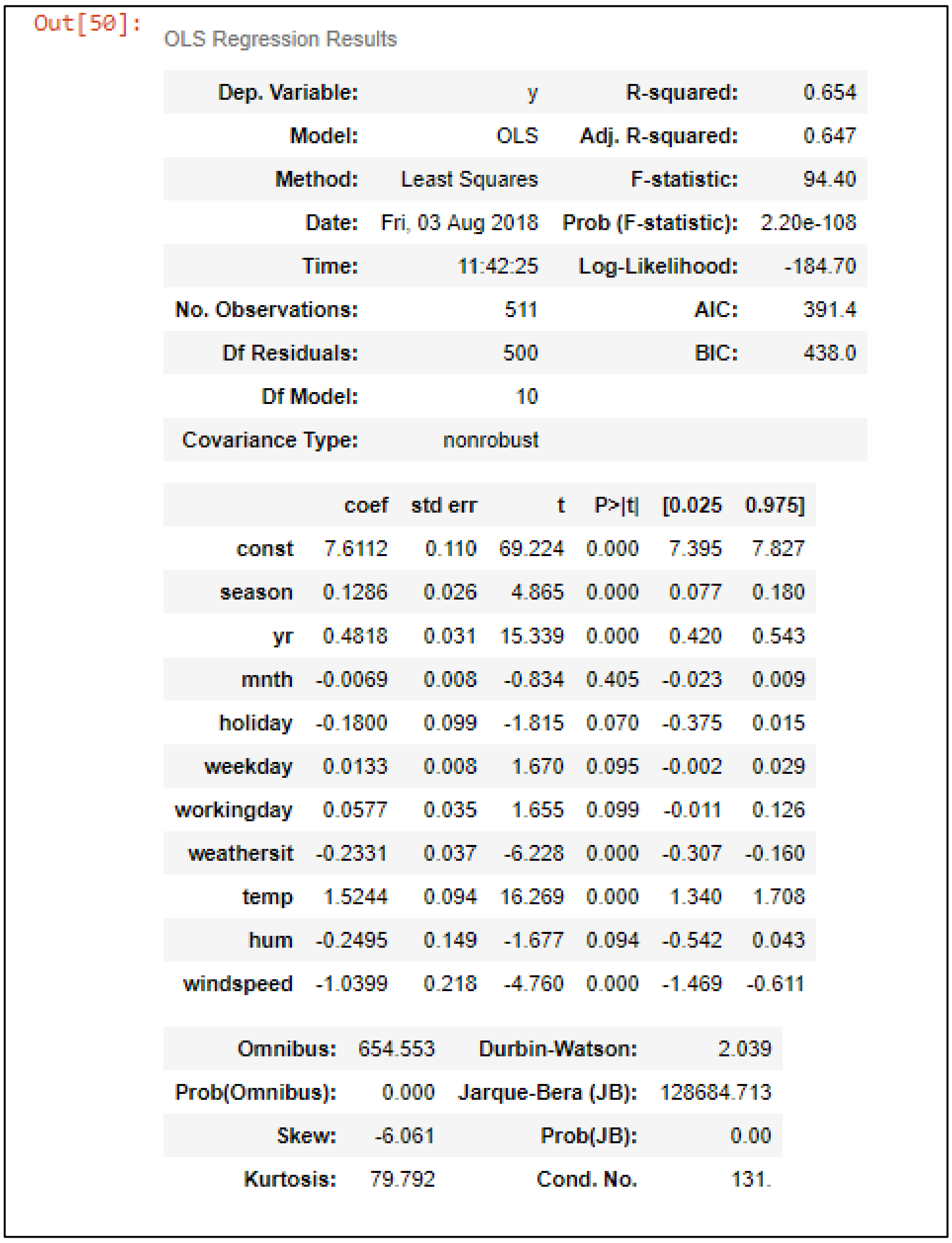
```
> summary(predict_test_nonlog)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
    486    3063    4381    4484    5822   10614
```

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.

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normalize=False)
```

Summary of python regression was examined used statsmodels.api. The summary is given below.



```
Out[50]:
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: Fri, 03 Aug 2018 Prob (F-statistic): 2.20e-108
               Time:         11:42:25    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
          yr  0.4818    0.031  15.339   0.000   0.420   0.543
        mnth -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
  weathersit -0.2331    0.037  -6.228   0.000  -0.307  -0.160
        temp  1.5244    0.094  16.269   0.000   1.340   1.708
         hum -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.
```

Figure 18. Linear Regression summary for python implementation

In python's linear regression, no data normalisation or scaling was performed.

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, hyperparameter tuning was used for increase performance.

R implementation

First random forest model was trained ‘rf_model_1’ with default setting.

```
Call:
 randomForest(formula = cnt ~ ., data = training_set, ntree = 500,      mtry = 8, i
mportance = TRUE)
               Type of random forest: regression
                     Number of trees: 500
No. of variables tried at each split: 8

          Mean of squared residuals: 452950.4
                    % Var explained: 87.87
```

After hyper tuning , a very litterimprovment was recoreded on variance explained.

```
Call:
 randomForest(formula = cnt ~ . - atemp, data = training_set,      ntree = 250, mtr
y = 6, importance = TRUE)
               Type of random forest: regression
                     Number of trees: 250
No. of variables tried at each split: 6

          Mean of squared residuals: 451228.2
                    % Var explained: 87.92
```

Mtry parameter was selected using the following plot.

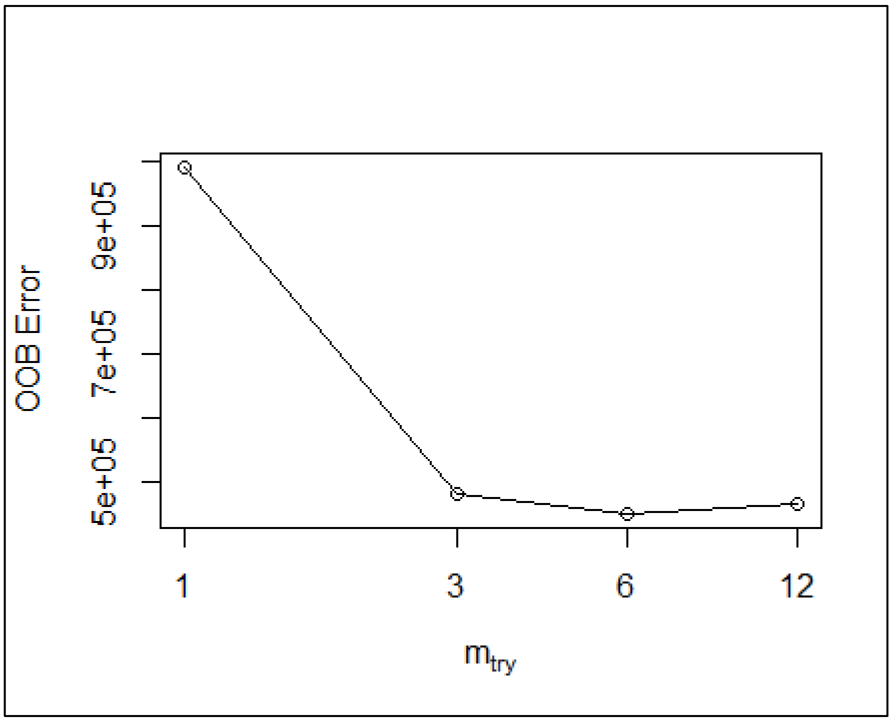


Figure 19. OOB error vs Mtry

Variable importance for R implementation of tuned random forest regressor is as follows.

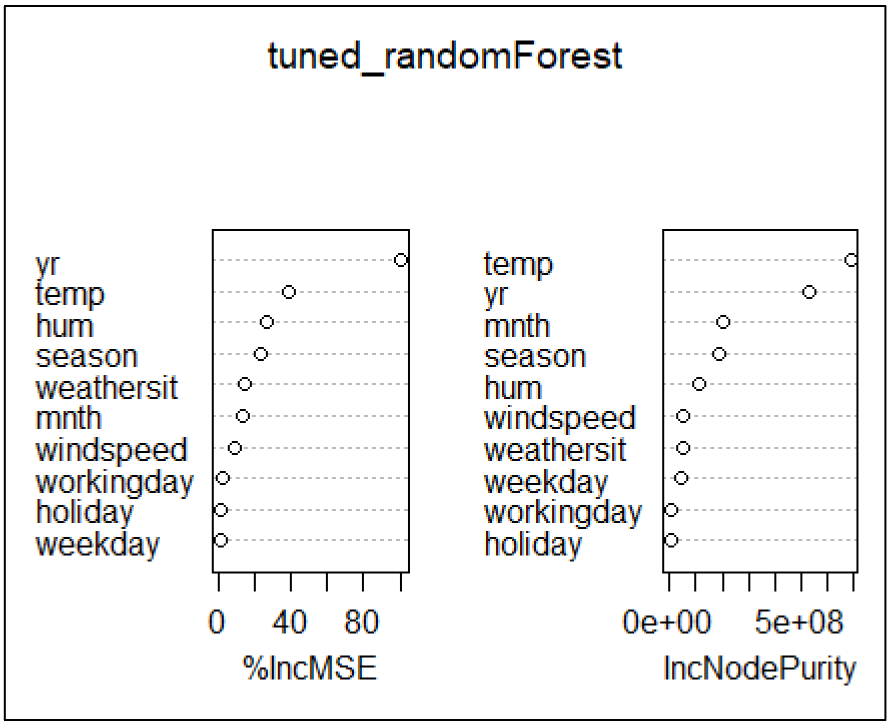


Figure 20. Feature importance R's random forest regressor

Python implementation

In python random forest was trained and hyperparameters optimisation was done using following parameters.

Default setting of random forest are given below.

```
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=10, n_jobs=1,
           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

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

Time taken in random search:  105.77
```

Using above mention parameters, random forest regressor was trained.

```
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=1,
           oob_score=False, random_state=12345, verbose=0,
           warm_start=False)
```

Variable importance using random forest in python.

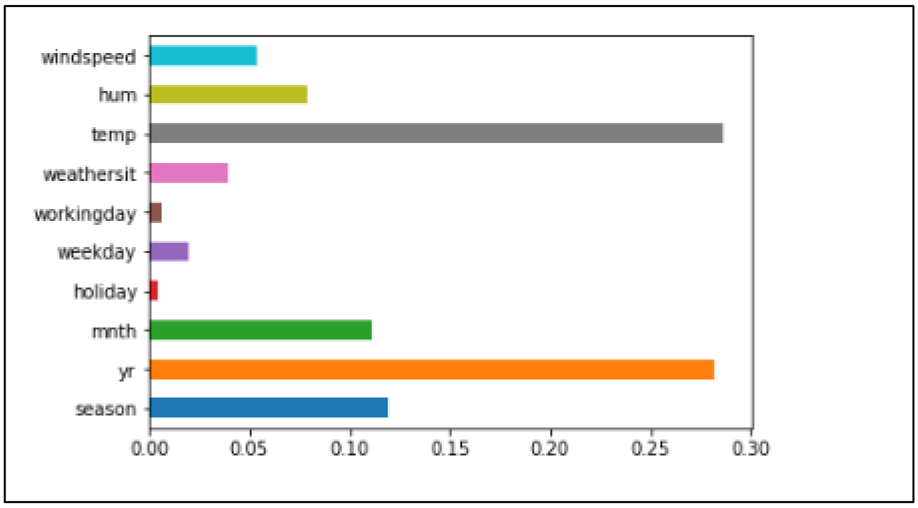


Figure 19. Variable importance random forest

Chapter 3

Result and Performance measure

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

3.1 Performance Measure

3.1.1 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	745.69
MAE	696.18	498.29

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

3.1.2 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	649.74
MAE	899.5	493.33

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

3.2 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.

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.

Appendix

Linear Regression Diagnostics

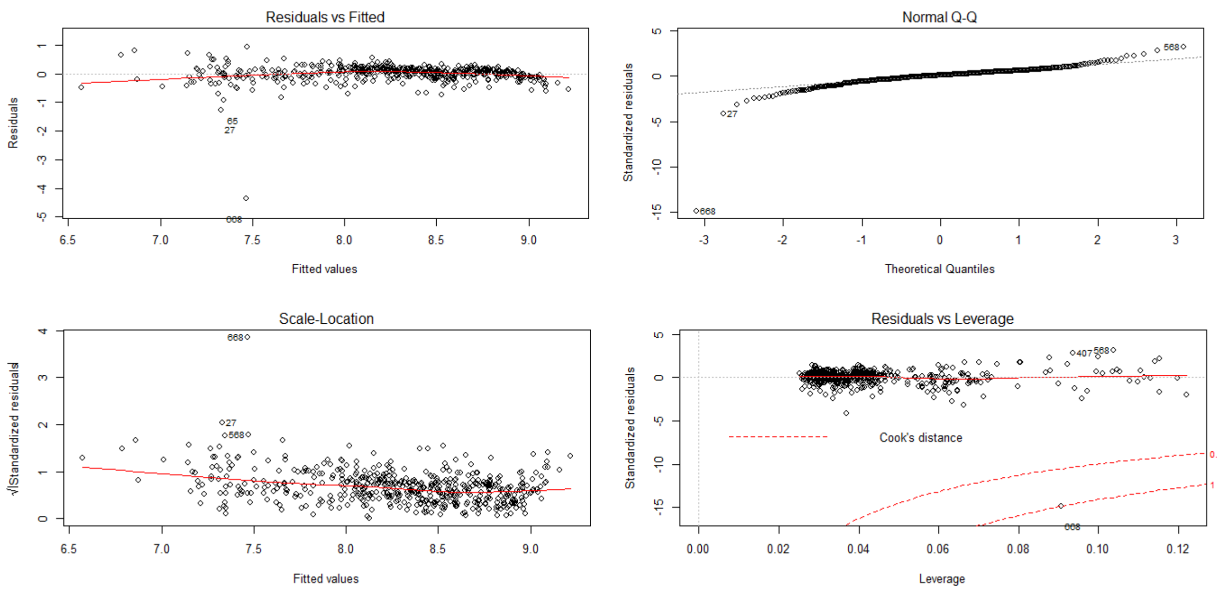


Figure 20. Linear regression diagnostics plot

Code

R Implementation

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


# reading csv file
raw_data <- read.csv("day.csv", header = TRUE)
head(raw_data)
dim(raw_data)
str(raw_data)

#checking for missing values
sapply(raw_data, function(x) {
  sum(is.na(x))
})
# There are no missing values

#convering categorical variables into factor
raw_data$season <- as.factor(raw_data$season)
levels(raw_data$season)[levels(raw_data$season) == 1] <- 'springer'
levels(raw_data$season)[levels(raw_data$season) == 2] <- 'summer'
levels(raw_data$season)[levels(raw_data$season) == 3] <- 'fall'
levels(raw_data$season)[levels(raw_data$season) == 4] <- 'winter'
```

```r

raw_data$yr <- as.factor(raw_data$yr)
levels(raw_data$yr)[levels(raw_data$yr) == 0] <- 2011
levels(raw_data$yr)[levels(raw_data$yr) == 1] <- 2012
raw_data$mnth <- as.factor(raw_data$mnth)
raw_data$holiday <- as.factor(raw_data$holiday)
raw_data$weekday <- as.factor(raw_data$weekday)
raw_data$workingday <- as.factor(raw_data$workingday)
raw_data$weathersit <- as.factor(raw_data$weathersit)
levels(raw_data$weathersit)[levels(raw_data$weathersit) == 1] <-
  'Good'
levels(raw_data$weathersit)[levels(raw_data$weathersit) == 2] <-
  'Cloudy'
levels(raw_data$weathersit)[levels(raw_data$weathersit) == 3] <-
  'Bad'


#-------------------------------------------------------------------------------------------------#
#                                                                                                 #
#                                    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(2, 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(raw_data, temp) #no outliers
outlierKD(raw_data, atemp) #no outliers
outlierKD(raw_data, hum) # no extreme outlier detected
outlierKD(raw_data, windspeed) #some extreme values are present but canot be considered as outlier
outlierKD(raw_data, casual) # no logical outliers
outlierKD(raw_data, registered)# no ouliers
outlierKD(raw_data, cnt)# no ouliers
```

```r


#--------------------------------------------------------------------------------------------------#
#                                                                                                  #
#                        Correlation Analysis                                                      #
#                                                                                                  #
#--------------------------------------------------------------------------------------------------#
par(mfrow = c(1, 1))
numeric_predictors <- unlist(lapply(raw_data, is.numeric))
numVarDataset <- raw_data[, 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 = raw_data, aes(cnt)) +
  geom_histogram(aes(
    y = ..density..,
    binwidth = .5,
    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 = .5, colour = "black")) +
    labs(x = variableName) +
    ggtitle(paste("count of", variableName))
}

univariate_continuous(raw_data, cnt, "cnt")
univariate_continuous(raw_data, temp, "temp")
univariate_continuous(raw_data, atemp, "atemp")
univariate_continuous(raw_data, hum, "hum") # skwed towards left
univariate_continuous(raw_data, windspeed, "windspeed") #skewed towards right
univariate_continuous(raw_data, casual, "casual") # skwed towards right
univariate_continuous(raw_data, 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(raw_data, season, 'season')
univariate_categorical(raw_data, yr, "yr")
univariate_categorical(raw_data, mnth, "mnth")
univariate_categorical(raw_data, holiday, "holiday")
univariate_categorical(raw_data, weekday, "weekday")
univariate_categorical(raw_data, workingday, "workingday")
univariate_categorical(raw_data, weathersit, "weathersit")

# -------------------------------------------------------------------------------------------------- #
#
#                                     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(raw_data, season, cnt)
bivariate_categorical(raw_data, yr, cnt)
bivariate_categorical(raw_data, mnth, cnt)
bivariate_categorical(raw_data, holiday, cnt)
bivariate_categorical(raw_data, weekday, cnt)
bivariate_categorical(raw_data, workingday, cnt)
bivariate_categorical(raw_data, weathersit, cnt)

bivariate_continous(raw_data, temp, cnt)
bivariate_continous(raw_data, atemp, cnt)
bivariate_continous(raw_data, hum, cnt)
bivariate_continous(raw_data, windspeed, cnt)
bivariate_continous(raw_data, casual, cnt)
bivariate_continous(raw_data, registered, cnt)


# removing instant and dteday
raw_data$instant <- NULL
raw_data$dteday <- NULL
raw_data$casual <- NULL
raw_data$registered <- NULL


# ------------------------------------------------------------------------------------------------------ #
#
#                                      Feature scaling or Normalization                               #
#
#------------------------------------------------------------------------------------------------------- #

scaledData <- normalizeFeatures(raw_data,'cnt')





# 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(raw_data$cnt, SplitRatio = 0.70)
```

```r
    training_set <- subset(raw_data, split == TRUE)
    test_set <- subset(raw_data, split == FALSE)


    model1 <- lm(cnt ~ ., 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$cnt, test_prediction)
    print(paste("root-mean-square error for linear regression model is ", test_rmse))
    print(paste("Mean Absolute Error for linear regression model is ",MAE(test_set$cnt,test_prediction)))
    print("summary of predicted count values")
    summary(test_prediction)
    print("summary of actual count values")
    summary(test_set$cnt)

    # From the summary we can observe negative prediction values
    #We will perform log transformation of trarget variable
    model2 <- lm(log(cnt)~., 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$cnt, predict_test_nonlog)
    print(paste("root-mean-square error between actual and predicted", test_rmse))
    print(paste("Mean Absolute Error for linear regression model is ",
                MAE(test_set$cnt,predict_test_nonlog)))

    summary(predict_test_nonlog)
    summary(test_set$cnt)



    par(mfrow = c(2, 2))
    plot(stepwiseLogAICModel)

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

    rf_model_1 <- randomForest(cnt ~.,
                                    data = training_set,ntree = 500, mtry = 8, importance = TRUE)
    print(rf_model_1)
    par(mfrow = c(1,1))
    plot(rf_model_1)

    cv_RF <- cv

    # 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(cnt ~. - atemp,
                                            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$cnt)
        print(paste("Mean Absolute Error for Random forest regressor  is ",
                    MAE(test_set$cnt,rf1_prediction)))

        #745

        varImpPlot(tuned_randomForest)


        # Random forest is performing better than linear regression.
```

Python Implementation

```
# coding: utf-8

# # Bike Demand Prediction

# In[1]:


# importing requried library

import pandas as pd
import numpy as np
import missingno as msno
import matplotlib.pyplot as plt
import seaborn as sn
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
from math import sqrt
import time
import random
import warnings
warnings.filterwarnings("ignore", category=DeprecationWarning)
pd.options.mode.chained_assignment = None

get_ipython().run_line_magic('matplotlib', 'inline')


# In[2]:


# Reading dataset
raw_data = pd.read_csv("day.csv")


# In[3]:

```

```python

raw_data.cnt.describe()


# In[4]:


raw_data.head(3)


# In[5]:


raw_data.shape


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

# In[6]:


raw_data.dtypes


# In the dataset season, yr, mnth, holiday, weekday, workingday, weathersit predictors should be categorical type, but they are int64. In the next step mapping and categorical transformatin will be performed.

# In[7]:


#raw_data["season"] = raw_data.season.map({1:"Spring",2:"Summer",3:"Fall",4:"winter"})
#raw_data["yr"] = raw_data.yr.map({0:"2011",1:"2012"})
#raw_data["weathersit"] = raw_data.weathersit.map({1: "Good",\
                                    #    2 : "OK ", \
                                     #   3 : " Bad"})


# In[8]:


#converting to categorical variable

categorical_variable = ["season","yr","mnth","holiday","weekday","workingday","weathersit"]

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


# In[9]:


raw_data.head(2)


# Droping variables which are not requried.
# 1. instant - index number
# 2. dteday- all the requried like month week day all ready present
# 3. casual and resgistered - their sum is equal to cnt ie. to the target variable

# In[10]:
```

```


raw_data = raw_data.drop(["instant","dteday"],axis = 1)


# ## Missing value Analysis

# In[11]:


# We will perform missing value andlysis using missingno package
msno.matrix(raw_data)


# There are no missing values present in the dataset

#    ## Outliers Analysis

# In[12]:


fig, axes = plt.subplots(nrows=2,ncols=2)
fig.set_size_inches(10,12)
sn.boxplot(data=raw_data,y="cnt",orient='v',ax=axes[0][0])
sn.boxplot(data=raw_data,y="cnt",x="season",orient='v',ax=axes[0][1])
sn.boxplot(data=raw_data,y="cnt",x="weekday",orient="v",ax=axes[1][0])
sn.boxplot(data=raw_data,y="cnt",x="workingday",orient="v",ax=axes[1][1])

axes[0][0].set(ylabel='cnt',title = "Boxplot of cnt")
axes[0][1].set(xlabel="season",ylabel="cnt",title="Boxplot for cnt vs season")
axes[1][0].set(xlabel="weekday", ylabel="cnt",title="Boxplot for cnt vs weekday")
axes[1][1].set(xlabel="workingday",ylabel="cnt",title="Boxplot for cnt vs workingday")


# In[13]:


fig.set_size_inches(8,12)
sn.boxplot(data=raw_data, x="weathersit",y="cnt").set_title("Boxplot of cnt vs weathersit")


# From the above boxplots, it is evident that there are no outliers present in the cnt. Two things are clear.
# 1. Cnt is very low in spring season.
# 2. Cnt is maximum when weather is good and its minimum weather is bab.

# ## Correlation Analysis

# In[14]:


churn_corr = raw_data.corr()
cmap = cmap=sn.diverging_palette(15, 250, as_cmap=True)

def magnify():
    return [dict(selector="th",
                 props=[("font-size", "7pt")]),
            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': '15pt'})    .set_caption("Correlation matrix")    .set_precision(2)    .set_table_styles(magnify())


# Finding of correlation analysis -
# 1. temp and atemp are highly correlated.
# 2. temp and atemp have positive and strong coorelation with cnt.
# 3. hum and windspeed have negative and weak correlation with cnt.
#

# ## Bivariate analysis

# In[15]:


# Bivariate analysis of cnt and continous predictor

fig,(ax1,ax2,ax3,ax4) = plt.subplots(ncols=4)
fig.set_size_inches(12,6)

sn.regplot(x="temp",y="cnt",data=raw_data,ax=ax1)
sn.regplot(x="atemp",y="cnt",data=raw_data,ax=ax2)
sn.regplot(x="hum",y="cnt",data=raw_data,ax=ax3)
sn.regplot(x="windspeed",y="cnt",data=raw_data,ax=ax4)


# From the above plot, it is evident that cnt has a positive linear relationship with temp and temp.
# On the other hand, cnt has a negative linear relationship with windspeed.

# Humidity(hum) has a little negative linear relationship with cnt.

# ## Distribution of target Variable

# In[16]:


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


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

# Preprocessing original data and Spliting into train and test data

# In[17]:


# selecting predictors
train_feature_space = raw_data.iloc[:,raw_data.columns != 'cnt']
# selecting target class
target_class = raw_data.iloc[:,raw_data.columns == 'cnt']


# In[18]:


#droping atemp due to multicollinearity
```

```python
#droping casual and registered because there sum is equal to target variable ie. 'cnt'

train_feature_space = train_feature_space.drop(["atemp","casual","registered"],axis = 1)


# In[19]:


train_feature_space.shape


# In[20]:


# creating training and test set
training_set, test_set, train_taget, 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_taget = train_taget.values.ravel()
test_target = test_target.values.ravel()


# ## Model1 Linear Regression Model
#

# In[21]:


import statsmodels.api as sm
X = training_set
X = sm.add_constant(X)
y= np.log(train_taget)

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


# In[22]:


model.summary()


# In[23]:


# Initialize logistic regression model
lModel = LinearRegression()
lModel.fit(X = training_set,y = np.log(train_taget))


# In[24]:


#predicting using linear regression
lmPredictions = lModel.predict(X=test_set)

```

```
# In[25]:


x=pd.DataFrame(np.exp(lmPredictions))


# In[26]:


x.describe()


# In[27]:


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


# In[28]:


rmse = sqrt(mean_squared_error(test_target, np.exp(lmPredictions)))


# In[29]:


print("RMSE for test set in linear regression is :" , rmse)


# In[30]:


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


# ## Model2 Random forest

# In[31]:


rf = RandomForestRegressor(random_state=12345)


# In[32]:


rf


# In[33]:


np.random.seed(12)
start = time.time()

# selecting best max_depth, maximum features, split criterion and number of trees
param_dist = {'max_depth': [2,4,6,8,10],
              'bootstrap': [True, False],
              'max_features': ['auto', 'sqrt', 'log2',None],
```

```python
              "n_estimators" : [100 ,200 ,300 ,400 ,500]
             }
cv_randomForest = RandomizedSearchCV(rf, cv = 10,
                     param_distributions = param_dist,
                     n_iter = 10)

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


# In[34]:


# setting parameters

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


# In[35]:


rf.fit(training_set, train_taget)


# In[36]:


# 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.')


# In[37]:


rmse_rf = sqrt(mean_squared_error(test_target, rfPredictions))


# In[38]:


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


# ### Variable importance for random forest

# In[39]:


feature_importance =  pd.Series(rf.feature_importances_, index=training_set.columns)
feature_importance.plot(kind='barh')
```