BIKE RENTING PREDICTION

Shankar Samidala

15 JULY 2018

Contents

Chapter 1

Introduction

1.1 Problem Description –

Our predictive task is to forecast bike rental demand of Bike renting program on historical usage patterns in relation with weather and seasonal settings.

1.2 Problem Statement –

The Objective of this Case is to Predication of bike rental count on daily based on the environmental and seasonal settings.
The details of data attributes in the dataset are as follows –

Table 1.1 Bike Renting Reduction Data (Columns 1-5)

Instant	Dteday	Season	Yr	Mnth
1	01-01-2011	1	0	1
2	02-01-2011	1	0	1
3	03-01-2011	1	0	1
4	04-01-2011	1	0	1
5	05-01-2011	1	0	1

Table 1.2 Bike Renting Reduction Data (Columns 6-10)

Holiday	Weekday	Working day	Weathersit	Temp
0	6	0	2	0.344
0	0	0	2	0.363
0	1	1	1	0.196
0	2	1	1	0.2
0	3	1	1	0.227

Table 1.3 Bike Renting Reduction Data (Columns 11-16)

Atempt	Hum	Windspeed	Casual	Registered	Cnt
0.364	0.806	0.160	331	654	985
0.354	0.696	0.249	131	670	801
0.189	0.437	0.248	120	1229	1349
0212	0.590	0.160	108	1454	1562
0.229	0.437	0.187	82	1518	1600

The Predictors provided are as follows

- 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 fromHoliday Schedule)
- weekday: Day of the week
- workingday: If day is neither weekend nor holiday is 1, otherwise is 0.
- weathersit: (extracted fromFreemeteo) 1: Clear, Few clouds, Partly cloudy, Partly cloudy 2: Mist + Cloudy, Mist + Broken clouds, Mist + Few clouds, Mist 3: Light Snow, Light Rain + Thunderstorm + Scattered clouds, Light Rain + Scattered clouds 4: Heavy Rain + Ice Pallets + Thunderstorm + Mist, Snow + Fog
- temp: Normalized temperature in Celsius. The values are derived via (t-t_min)/(t_max-t_min), t_min=-8, t_max=+39 (only in hourly scale)
- atemp: Normalized feeling temperature in Celsius. The values are derived via (t-t_min)/(t_max- 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 vaalues 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

Target Variable : cnt

We have to predict the count of bike rental on daily basis

Chapter 2

Methodology

2.1 Pre Processing

Data pre processing is a data mining technique that involves transforming raw data into an understandable format. Real-world data is often incomplete, inconsistent, and/or lacking in certain behaviours or trends, and is likely to contain many errors. Data pre processing is a proven method of resolving such issues. Data pre processing prepares raw data for further processing.

If there is much irrelevant and redundant information present or noisy and unreliable data, then knowledge discovery during the training phase is more difficult. Data preparation and filtering steps can take considerable amount of processing time. Data pre-processing includes cleaning ,outlier deduction, normalization, feature extraction and selection , etc. The product of data pre-processing is the final data. This is often called as Exploratory Data Analysis **.**

2.1.1 Missing value analysis

Missing data can occur because of nonresponse: no information is provided for one or more items or for a whole unit ("subject"). Some items are more likely to generate a nonresponse than others: for example items about private subjects such as income. Attrition ("Dropout") is a type of missingness that can occur in longitudinal studies - for instance studying development where a measurement is repeated after a certain period of time. Missingness occurs when participants drop out before the test ends and one or more measurements are missing.

Missing value analysis helps address several concerns caused by incomplete data. If cases with missing values are systematically different from cases without missing values, the results can be misleading. Also, missing data may reduce the precision of calculated statistics because there is less information than originally planned. Another concern is that

the assumptions behind many statistical procedures are based on complete cases, and missing values can complicate the theory required.

Example: In evaluating a treatment for cancer, several variables are measured. However, not all measurements are available for every patient. The patterns of missing data are displayed, tabulated, and found to be random. An EM (expectationmaximization) analysis is used to estimate the means, correlations, and covariances. It is also used to determine that the data are missing completely at random. Missing values are then replaced by imputed values and saved into a new data file for further analysis.

2.1.2 Outliers Analysis :

Outliers can occur by chance in any distribution, but they often indicate either measuring error .In the former case one wishes to discard them or use statistics that are robust to outliers, while in the latter case they indicate that the distribution has high skewness and that one should be very cautious in using tools or intuitions that assume a normal distribution A frequent cause of outliers is a mixture of two distributions, which may be two distinct sub-populations, or may indicate 'correct trial' versus 'measurement error'.

it is a point or an observation that deviates significantly from the other observations.

●Due to experimental errors or “special circumstances”

●Outlier detection tests to check for outliers

●Outlier treatment –

- Retention
- Exclusion
- Other treatment methods

We have “OUTLIER” package in R to detect and treat outliers in Data.

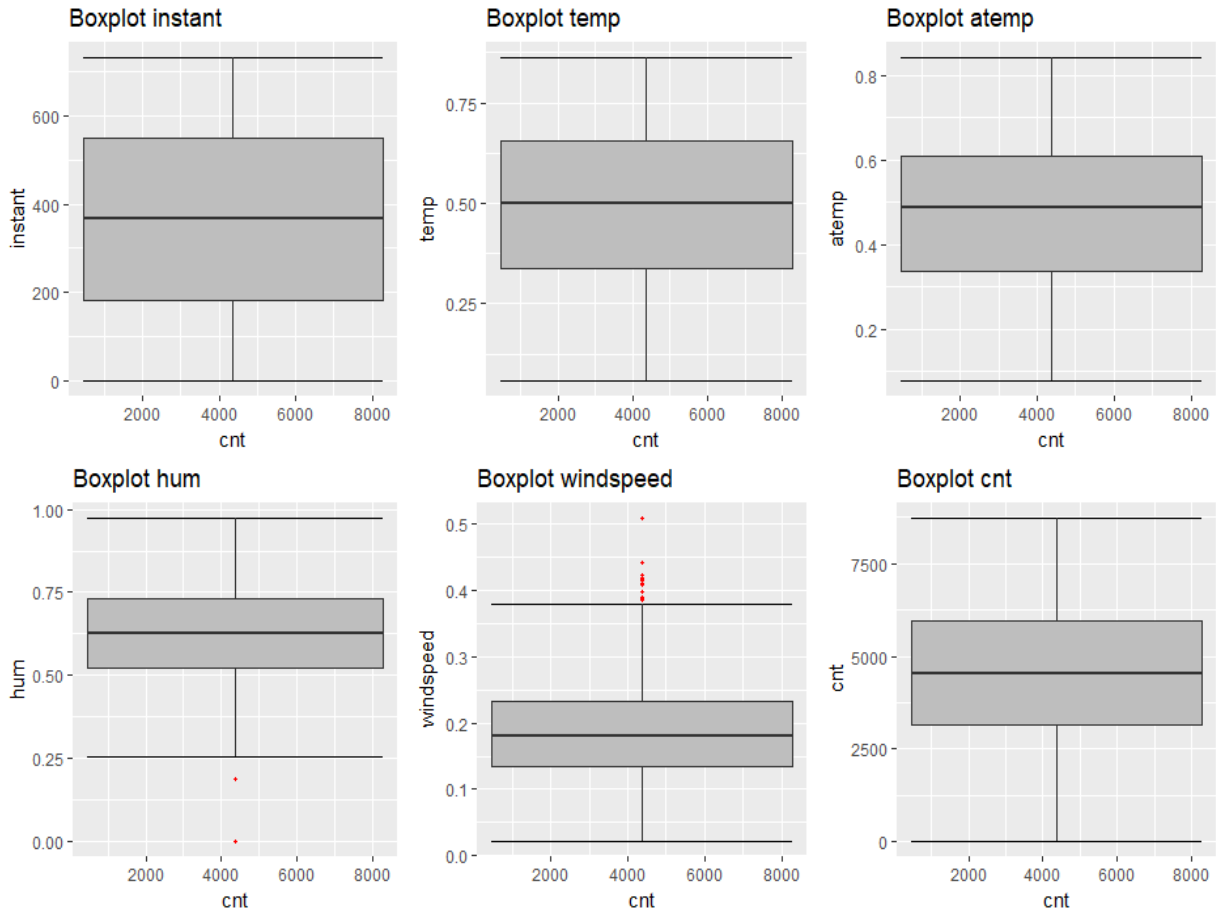
Normally we use BOX Plot and Scatter plot to find outliers from graphical representation.

Causes of Outliers :

• Poor data quality / contamination

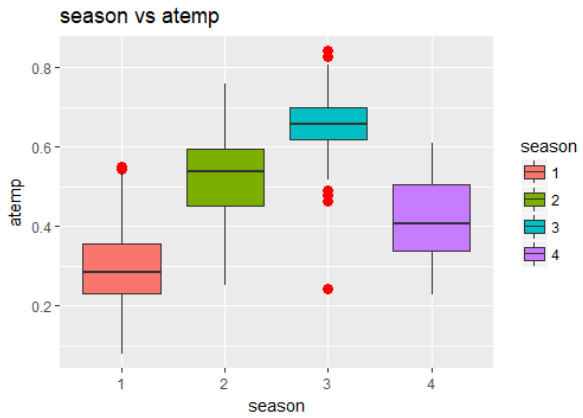
• Low quality measurements, malfunctioning equipment, manual error • Correct but exceptional data

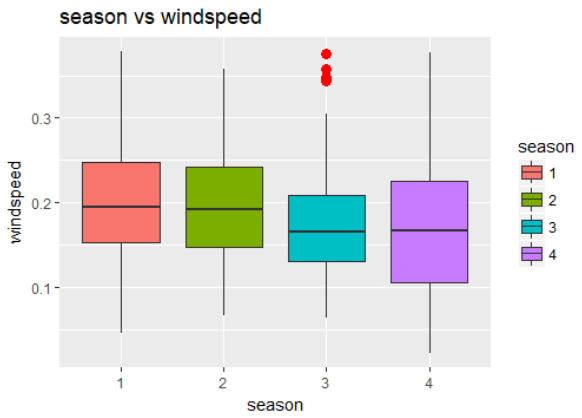
• Mean = Most Common measure of Central Tendency • Mean Affected by Extreme Values



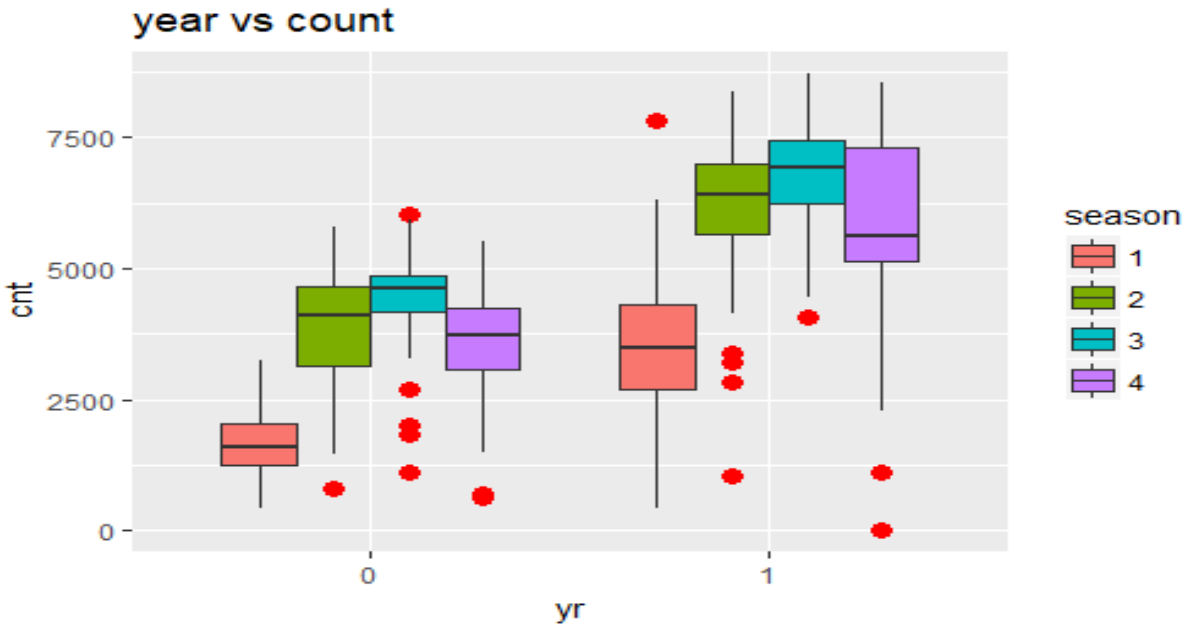
Outliers in atemp with season

outliers in windspeed with season





Outliers in year and count variables with season :



2.1.3 Feature Selection

Feature selection is critical to building a good model for several reasons. One is that feature selection implies some degree of cardinality reduction, to impose a cutoff on the number of attributes that can be considered when building a model. Data almost always contains more information than is needed to build the model, or the wrong kind of information. For example, you might have a dataset with 500 columns that describe the characteristics of customers; however, if the data in some of the columns is very sparse you would gain very little benefit from adding them to the model, and if some of the columns duplicate each other, using both columns could affect the model.

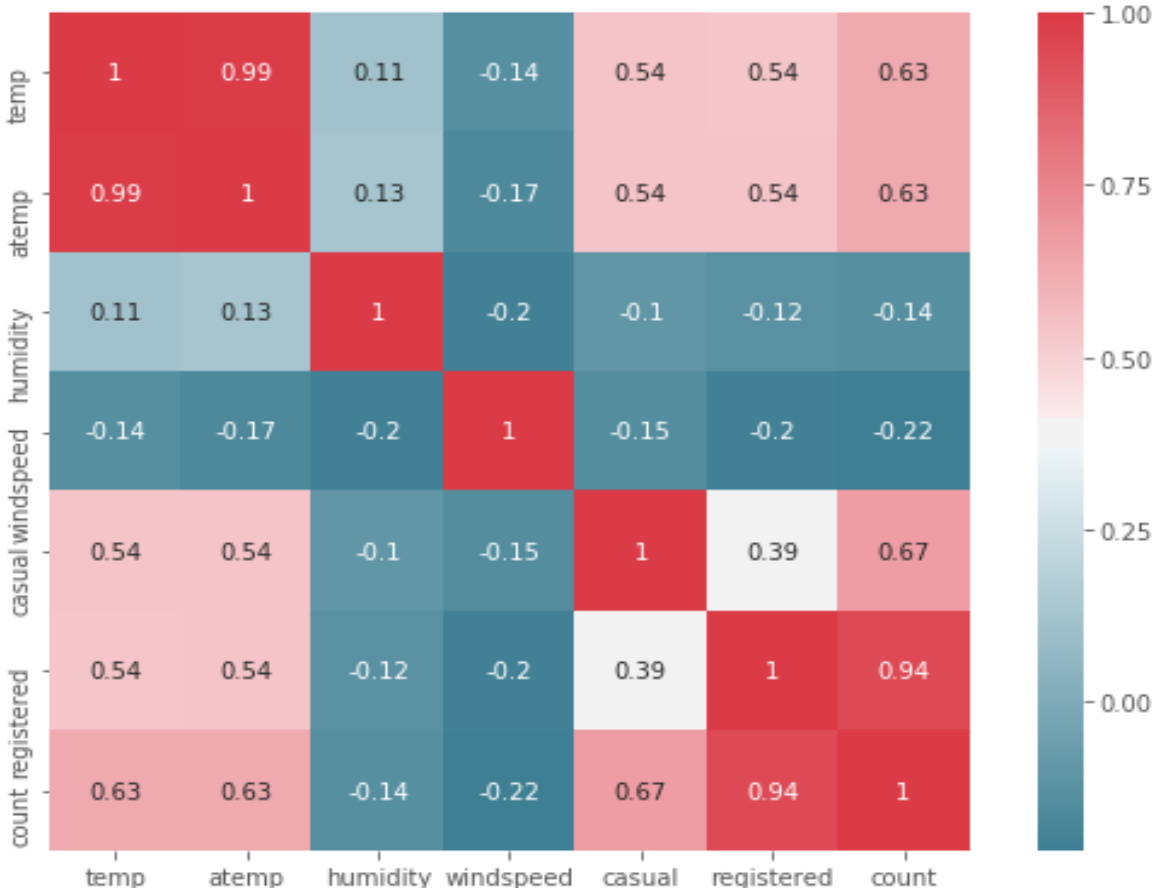
Not only does feature selection improve the quality of the model, it also makes the process of modeling more efficient. If you use unneeded columns while building a model, more CPU

and memory are required during the training process, and more storage space is required for the completed model. Even if resources were not an issue, you would still want to perform feature selection and identify the best columns, because unneeded columns can degrade the quality of the model in several ways:

 Noisy or redundant data makes it more difficult to discover meaningful patterns. If the data set is high-dimensional, most data mining algorithms require a much larger training data set.

(A)Correlation analysis

A correlation matrix is a table showing correlation coefficients between sets of variables. Each random variable (Xi) in the table is correlated with each of the other values in the table (Xj). This allows you to find and predict which pairs have the highest correlation. If two variables are highly correlated we have two drop one variable .



Correlation Matrix

The above correlation matrix shows some interesting results as follows :

1. Temp and atemp are very highly correlated.

2. Registered and count are very highly correlated.

3. Now we have to remove the highly correlated variable so that our model can perform well and it gives much accuracy.

(B) ANOVA Test

An ANOVA test is a way to find out if survey or experiment results are significant. In other words, they help you to figure out if you need to reject the null hypothesis or accept the alternate hypothesis. Basically, you're testing groups to see if there's a difference between them. Examples of when you might want to test different groups:

- A group of psychiatric patients are trying three different therapies: counseling, medication and biofeedback. You want to see if one therapy is better than the others.
- A manufacturer has two different processes to make light bulbs. They want to know if one process is better than the other.
- Students from different colleges take the same exam. You want to see if one college outperforms the other

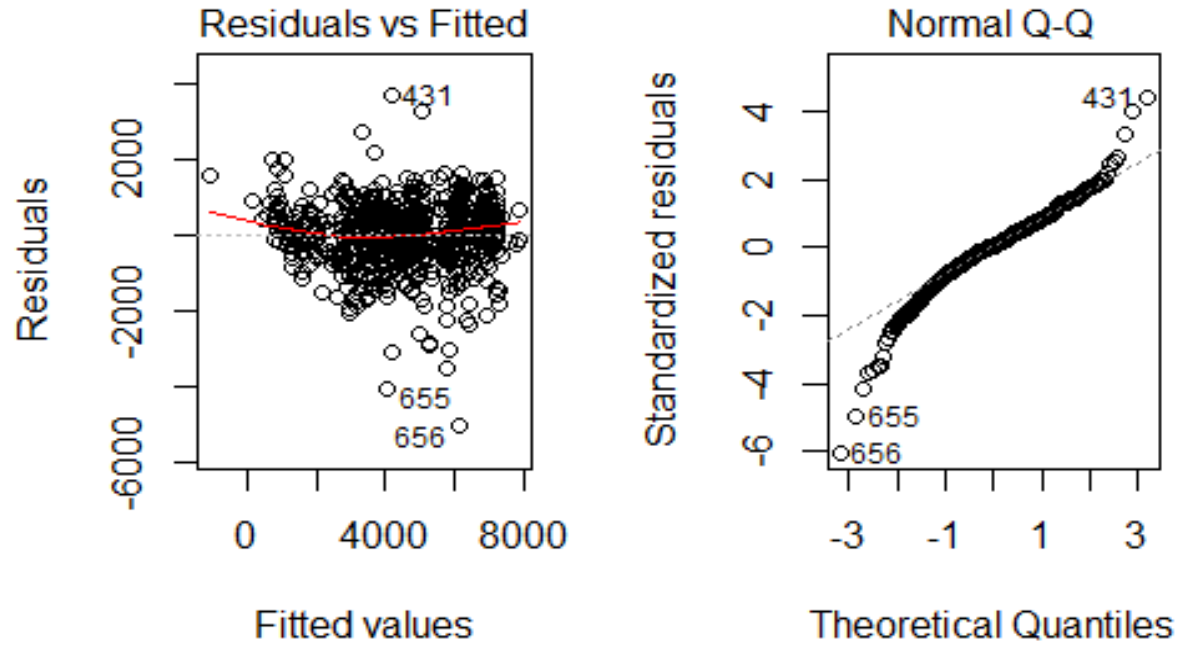
• ANOVA is a statistical technique used to compare the means of two or more groups of observations.

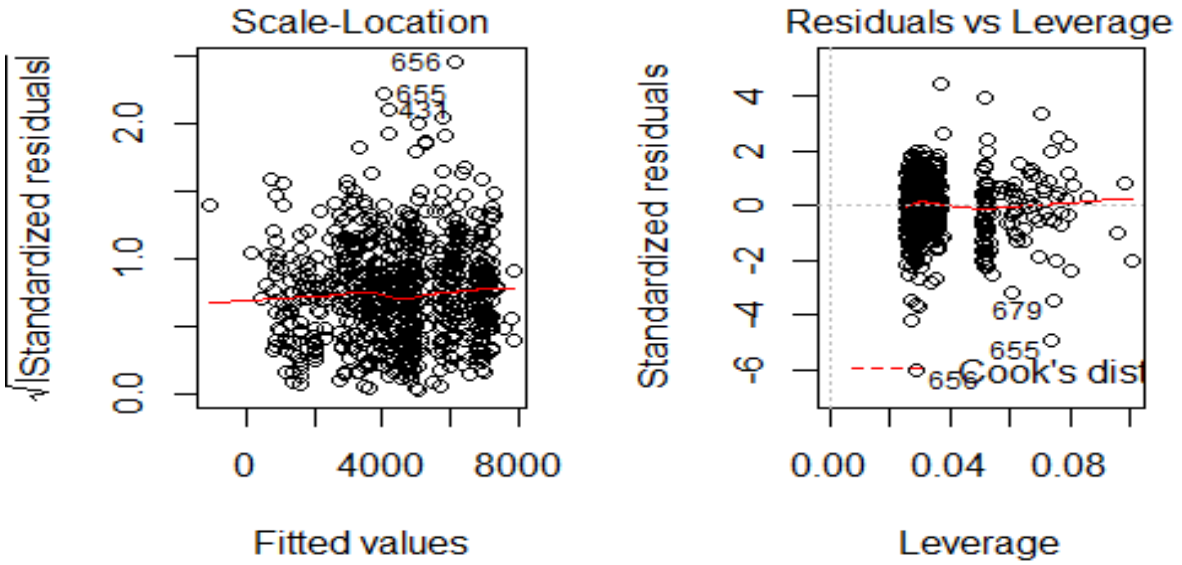
• ANOVA compares the means between the groups you are interested in and determines whether any of those means are significantly different from each other.

• Specifically, it tests the null hypothesis:

In anova we check the relationship between one continuous and one categorical variable

```
> m1=aov(cnt~season+yr+mnth+holiday+weekday+workingday+weathersit)
> summary(m1)
             Df    Sum Sq   Mean Sq  F value  Pr(>F)
season        3 921846604 307282201  427.956 < 2e-16 ***
yr            1 871757387 871757387 1214.108 < 2e-16 ***
mnth         11 184091208  16735564   23.308 < 2e-16 ***
holiday       1   3612964   3612964    5.032 0.02520 *
weekday       6  14576781   2429463    3.384 0.00269 **
weathersit    2 184071169  92035585  128.179 < 2e-16 ***
Residuals   692 496871695    718023
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```





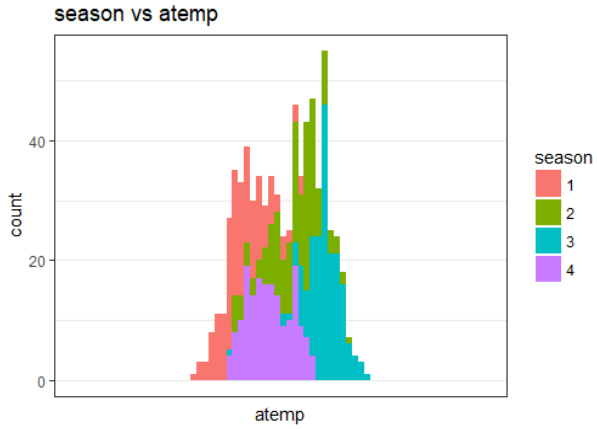
If the p value of the categorical variable is less than 0.05 it is representing the Alternative hypothesis. If the p value is grater than 0.05 it is representing Null Hypothesis. We have to drop the variables which are having null hypothesis because those are not carrying much information to explain the target variable. We will consider only the variables having Alternative hypothesis.

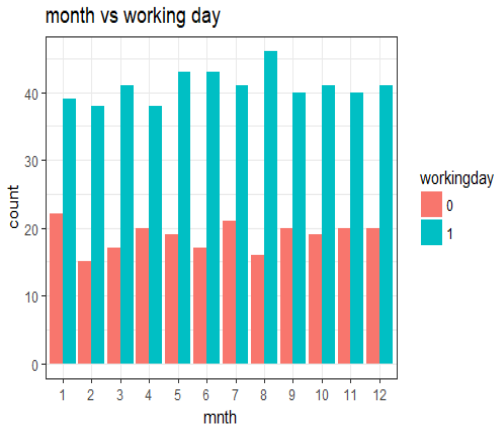
Therefore from both the correlation analysis and anova test of independence we got some variable which are not carrying much information. We have to delete those variables

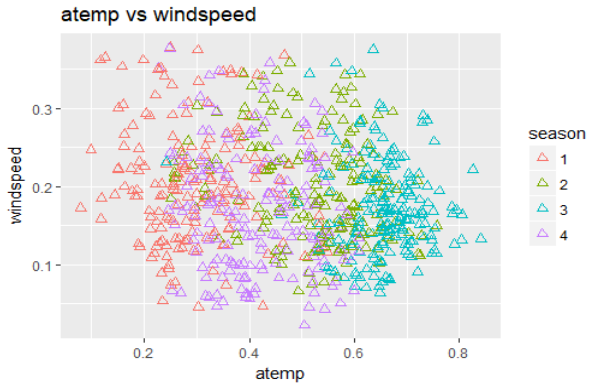
Numerical: temp, registered

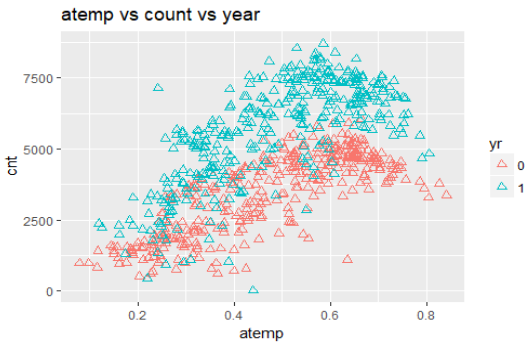
Categorical: holiday

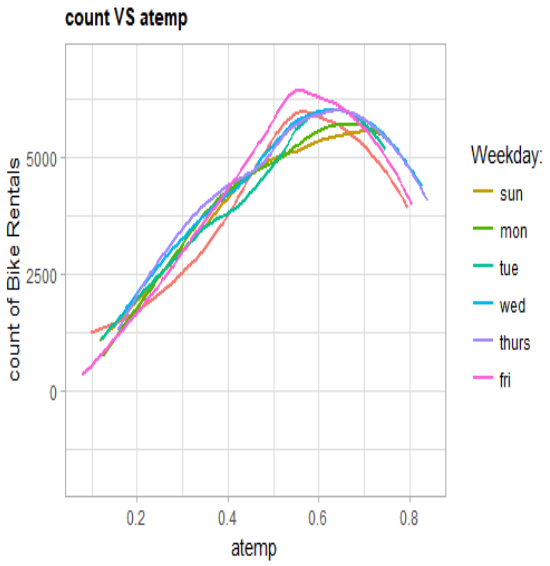
VISUALISATIONS :

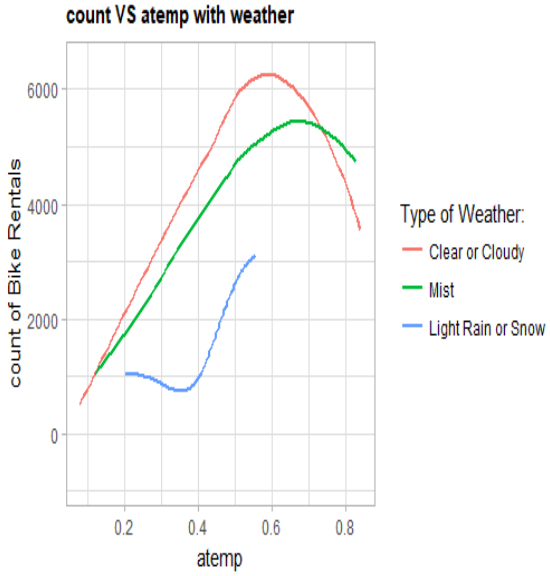








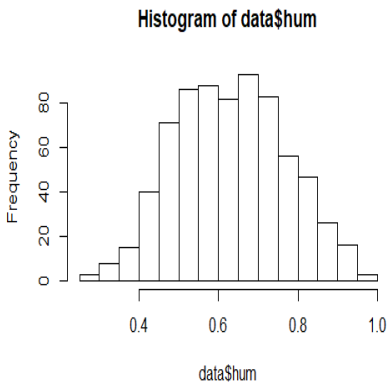




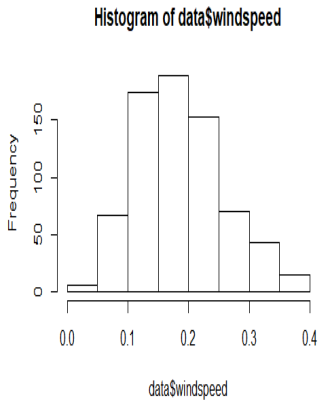
2.1.4 Feature Scaling

Feature scaling is a method used to standardize the range of independent variables or features of data. In data processing, it is also known as data normalization and is generally performed during the data pre processing steps. If training an algorithm using different features and some of them are off the scale in their magnitude, then the results might be dominated by them. Therefore, the range of all features should be normalized so that each feature contributes approximately proportionately to the final distance.

Her we are not applying feature scaling because the data is already normalized.



Distribution of humidity



Distribution of wind speed

2.2 Modeling

2.2.1 Model Selection

Before applying the model we have to divide the data into train and test sets

After pre processing we will be using some Regression models on our processed data to predict the target variable.

2.2.2 Decision Tree: Decision tree is a type of supervised learning algorithm (having a pre-defined target variable) that is mostly used in classification problems. It works for both categorical and continuous input and output variables. In this technique, we split the population or sample into two or more homogeneous sets (or sub-populations) based on most significant splitter / differentiator in input variables. Decision tree is a rule. Each branch connects nodes with “and” and multiple branches are connected by “or”. It can be used for classification and regression. It is a Supervised machine learning algorithm. Accept continuous and categorical variables as independent variables. Extremely easy to understand by the business users. There are many types of decision trees.

Categorical Variable Decision Tree: Decision Tree which has categorical target variable then it called as categorical variable decision tree. Example:- a scenario of student problem, where the target variable was “Student will Pass or not” i.e. YES or NO.

Continuous Variable Decision Tree: Decision Tree has continuous target variable then it is called as Continuous Variable Decision Tree.

We are using C5.0 model which is entropy based. When we applied this model on our train data, we got certain rules which is provided in a text file. You can also

visualize the decision tree with the dot in python . The accuracy of the model as following

```
> rg=rpart(cnt~.,train,method="anova")
> rg
n= 573

node), split, n, deviance, yval
      * denotes terminal node

 1) root 573 2086503000 4501.894
   2) atemp< 0.4308565 237  562925700 3152.895
     4) yr=0 123  124732800 2256.919
       8) season=1,2 83   28092990 1738.398 *
       9) season=4 40   28018980 3332.850 *
     5) yr=1 114  232915500 4119.605
      10) season=1 61   61620600 3213.016
        20) atemp< 0.294643 32   24177470 2598.250 *
        21) atemp>=0.294643 29   12004010 3891.379 *
      11) season=2,3,4 53   63454930 5163.038 *
   3) atemp>=0.4308565 336  788070500 5453.420
     6) yr=0 168  119516100 4283.631
      12) weathersit=3 7     805894 2257.000 *
      13) weathersit=1,2 161   88709570 4371.745 *
     7) yr=1 168  208770100 6623.208
      14) hum>=0.771458 22   53343840 5233.909 *
      15) hum< 0.771458 146  106564300 6832.555
        30) mnth=2,3,4,5,7,11,12 69   41226680 6408.754 *
        31) mnth=6,8,9,10 77   41839400 7212.325 *
> dt_prd=predict(rg,test[,-10])
> regr.eval(test[,10],dt_prd,stats = c("mape","rmse","mae"))
     mape      rmse       mae
  0.24172 932.03266 694.14541
> rmsle(test[,10],dt_prd)
[1] 0.3033795
```

2.2.3 Random Forest : Random Forest or decision tree forests is an ensemble learning method for classification, regression and other tasks. Random Forest is a supervised learning algorithm. Like you can already see from it's name, it creates a forest and makes it somehow random. The forest it builds, is an ensemble of Decision Trees, most of the time trained with the "bagging" method. The general idea of the bagging method is that a combination of learning models increases the overall result. One big advantage of random forest is, that it can be used for both classification and regression problems, which form the majority of current machine learning systems. I will talk about random forest in classification, since classification is sometimes considered the building block of machine learning. We can see certain rules of random forest in the R code. The accuracy of the model is as follows.

```
> rf=randomForest(cnt~.,train,ntree=100,importance=T,bootstrap=T)
> rf

Call:
 randomForest(formula = cnt ~ ., data = train, ntree = 100, importance = T,      bootstrap = T)
               Type of random forest: regression
                     Number of trees: 100
No. of variables tried at each split: 3

          Mean of squared residuals: 467970.3
                    % Var explained: 87.15
> rf_prd=predict(rf,test[,-10])
> summary(rf_prd)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
   1260    3566    4622    4709    6428    7662
> regr.eval(test[,10],rf_prd,stats = "rmse")
   rmse
753.163
> rmsle(test[,10],rf_prd)
[1] 0.2698992
```

2.2.4 Linear Regression :

Simple linear regression is useful for finding relationship between two continuous variables. One is predictor or independent variable and other is response or dependent variable. It looks for statistical relationship but not deterministic relationship. Relationship between two variables is said to be deterministic if one variable can be accurately expressed by the other. For example, using temperature in degree Celsius it is possible to accurately predict Fahrenheit. Statistical relationship is not accurate in determining relationship between two variables. For example, relationship between height and weight.

The core idea is to obtain a line that best fits the data. The best fit line is the one for which total prediction error (all data points) are as small as possible. Error is the distance between the point to the regression line.

Example:

We have a dataset which contains information about relationship between 'number of hours studied' and 'marks obtained'. Many students have been observed and their hours of study and grade are recorded. This will be our training data. Goal is to design a model that can predict marks if given the number of hours studied. Using the training data, a regression line is obtained which will give minimum error. This linear equation is then used for any new data. That is, if we give number of hours studied by a student as an input, our model should predict their mark with minimum error.

Y(pred) = b0 + b1*x

The values b0 and b1 must be chosen so that they minimize the error. If sum of squared error is taken as a metric to evaluate the model, then goal to obtain a line that best reduces the error.

$$\text{Error} = \sum_{i=1}^{n} (actual\_output - predicted\_output) ** 2$$

Figure 2: Error Calculation

If we don't square the error, then positive and negative point will cancel out each other.

For model with one predictor,

$$b_0 = \bar{y} - b_1\bar{x}$$

$$b_1 = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^{n}(x_i - \bar{x})^2}$$

Exploring 'b1'

- If b1 > 0, then x(predictor) and y(target) have a positive relationship. That is increase in x will increase y.
- If b1 < 0, then x(predictor) and y(target) have a negative relationship. That is increase in x will decrease y.

Exploring 'b0'

- If the model does not include x=0, then the prediction will become meaningless with only b0. For example, we have a dataset that relates height(x) and weight(y). Taking

x=0(that is height as 0), will make equation have only b0 value which is completely meaningless as in real-time height and weight can never be zero. This resulted due to considering the model values beyond its scope.

- If the model includes value 0, then 'b0' will be the average of all predicted values when x=0. But, setting zero for all the predictor variables is often impossible.
- The value of b0 guarantee that residual have mean zero. If there is no 'b0' term, then regression will be forced to pass over the origin. Both the regression co-efficient and prediction will be biased.

Metrics for model evaluation

R-Squared value :

This value ranges from 0 to 1. Value '1' indicates predictor perfectly accounts for all the variation in Y. Value '0' indicates that predictor 'x' accounts for no variation in 'y'.

1. Regression sum of squares (SSR)

This gives information about how far estimated regression line is from the horizontal 'no relationship' line (average of actual output).

$$\text{Error} = \sum_{i=1}^{n} (\text{Predicted\_output} - \text{average\_of\_actual\_output})\text{^}2$$

2. Sum of Squared error (SSE)

How much the target value varies around the regression line (predicted value).

$$\text{Error} = \sum_{i=1}^{n} (Actual\_output - predicted\_output)\text{^}2$$

3. Total sum of squares (SSTO)

This tells how much the data point move around the mean.

Correlation co-efficient (r)

This is related to value of ‘r-squared’ which can be observed from the notation itself. It ranges from -1 to 1.

$r = (+/-) \text{ sqrt}(r^2)$

If the value of b1 is negative, then ‘r’ is negative whereas if the value of ‘b1’ is positive then, ‘r’ is positive. It is unitless.

Null-Hypothesis and P-value

Null hypothesis is the initial claim that researcher specify using previous research or knowledge.

Low P-value: Rejects null hypothesis indicating that the predictor value is related to the response

High P-value: Changes in predictor are not associated with change in target

Obtained Regression line

```
> lm_mod=lm(cnt~.,train)
> summary(lm_mod)

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

Residuals:
    Min      1Q  Median      3Q     Max
-3530.4  -358.0    60.3   412.5  3071.8

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept)  1418.614    267.437   5.304 1.65e-07 ***
season2       962.679    197.231   4.881 1.39e-06 ***
season3       937.133    237.014   3.954 8.70e-05 ***
season4      1534.377    198.664   7.723 5.46e-14 ***
yr1          2045.404     65.436  31.258  < 2e-16 ***
mnth2         113.843    160.476   0.709 0.478373
mnth3         505.412    184.673   2.737 0.006407 **
mnth4         409.708    274.180   1.494 0.135674
mnth5         753.242    289.819   2.599 0.009603 **
mnth6         602.398    301.636   1.997 0.046311 *
mnth7         222.405    338.003   0.658 0.510818
mnth8         547.760    325.393   1.683 0.092875 .
mnth9        1140.335    288.089   3.958 8.55e-05 ***
mnth10        674.154    264.758   2.546 0.011161 *
mnth11         -1.993    251.857  -0.008 0.993689
mnth12        -37.916    197.591  -0.192 0.847900
weekday1     -481.147    201.394  -2.389 0.017229 *
weekday2     -283.929    227.116  -1.250 0.211780
weekday3     -224.815    228.558  -0.984 0.325737
weekday4     -201.503    227.152  -0.887 0.375424
weekday5     -167.503    223.530  -0.749 0.453966
weekday6      354.908    121.905   2.911 0.003746 **
workingday1   526.028    194.402   2.706 0.007026 **
weathersit2  -486.219     88.147  -5.516 5.37e-08 ***
weathersit3 -2179.339    226.074  -9.640  < 2e-16 ***

atemp        4253.436    479.972   8.862  < 2e-16 ***
hum         -1341.031    350.518  -3.826 0.000145 ***
windspeed   -1883.012    481.895  -3.908 0.000105 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 756.3 on 545 degrees of freedom
Multiple R-squared:  0.8506,    Adjusted R-squared:  0.8432
F-statistic: 114.9 on 27 and 545 DF,  p-value: < 2.2e-16

> lm_prd=predict(lm_mod,test[,-10])
> lm_prd=round(lm_prd)
> regr.eval(test[,10], lm_prd, stats = "rmse")
    rmse
850.4599
> summary(lm_prd)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
    649    3576    4908    4675    6321    7486
> rmsle(test[,10],pred)
[1] 0.3033795
```

2.2.5 XGBoost :

XGBoost stands for e**X**treme **G**radient **B**oosting.
XGBoost is an algorithm that has recently been dominating applied machine learning
XGBoost is an implementation of gradient boosted decision trees designed for speed and performance. It was created by Tianqi Chen It belongs to a broader collection of tools under the umbrella of the Distributed Machine Learning Community or DMLC who are also the creators of the popular mxnet deep learning library.
The library is laser focused on computational speed and model performance, as such there are few frills. Nevertheless, it does offer a number of advanced features.
Generally, XGBoost is fast. Really fast when compared to other implementations of gradient boosting.Szilard Pafka performed some objective benchmarks comparing the performance of XGBoost to other implementations of gradient boosting and bagged decision trees. He wrote up his results in May 2015 in the blog post titled “Benchmarking Random Forest Implementations“.He also provides all the code on GitHub and a more extensive report of results with hard numbers.

```
[1]     train-rmse:3479.930908
[2]     train-rmse:2497.775146
[3]     train-rmse:1811.246460
[4]     train-rmse:1332.831177
[5]     train-rmse:1000.755676
[6]     train-rmse:774.059021
[7]     train-rmse:620.907898
[8]     train-rmse:515.584900
[9]     train-rmse:444.604401
[10]    train-rmse:398.614075
> xgb_prd = getPrediction(list(xgb_mod),test,list(features))
> evaluate(test$cnt,xgb_prd)
[1] 0.2329852
> regr.eval(test[,10],xgb_prd,stats = "rmse")
    rmse
703.1444
```

SVM REGRESSION:

Support Vector Machine can also be used as a regression method, maintaining all the main features that characterize the algorithm (maximal margin). The Support Vector Regression (SVR) uses the same principles as the SVM for classification, with only a few minor differences. First of all, because output is a real number it becomes very difficult to predict the information at hand, which has infinite possibilities. In the case of regression, a margin of tolerance (epsilon) is set in approximation to the SVM which would have already requested from the problem. But besides this fact, there is also a more complicated reason, the algorithm is more complicated therefore to be taken in consideration. However, the main idea is always the same: to minimize error, individualizing the hyperplane which maximizes the margin, ***Support Vector Machine*** can be applied not only to classification problems but also to the case of regression. Still it contains all the main features that characterize maximum margin algorithm: a non-linear function is leaned by linear learning machine mapping into high dimensional kernel induced feature space. The capacity of the system is controlled by parameters that do not depend on the dimensionality of feature space.

```
Call:
svm(formula = cnt ~ ., data = train, kernel = "radial", epsilon = 0.1, gamma = 0.17,
    scale = TRUE)


Parameters:
   SVM-Type:  eps-regression
 SVM-Kernel:  radial
       cost:  1
      gamma:  0.17
    epsilon:  0.1


Number of Support Vectors:  386

> svm_prd = getPrediction(list(svm_mod),test,list(features))
> evaluate(test$cnt,svm_prd)
[1] 0.2512134
```

Chapter 3

Conclusion

3.1 Model Evaluation

MODEL EVALUATION :We will be evaluating our model on the basis of Root Mean Square Log Error (RMSLE). RMSLE is calculated as:

ε =√(1/n) (log(p) log(a)) ∑ n i=1 i+1 − i +1 2

Where, pi is the predicted value, ai is the actual value and n is the total number of samples. RMSLE is suitable for our problem because RMSLE penalises an under prediction more than an over prediction.The bike renting company would lose revenue if the number of bikes will be less than the demand for the bikes.

Model	RMSLE	RMSE
Decision tree	0.3033	932.03
Random forest	0.269	753.16
Linear regression	0.303	850.45
XGboost	0.23	703.14
SVM	0.25	603.34

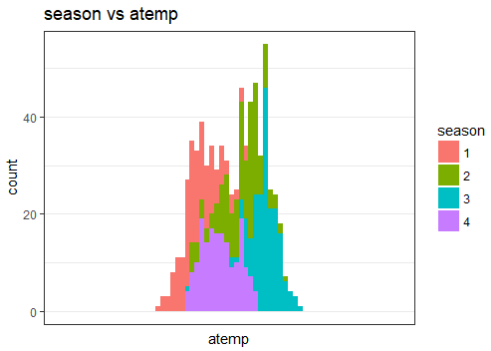
3.2 Model Selection

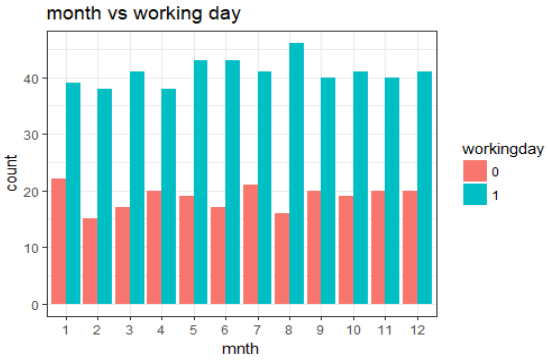
CONCLUSION

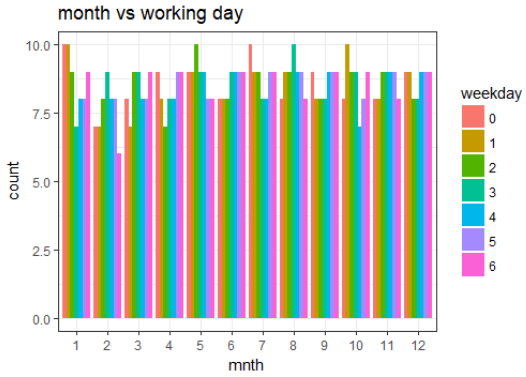
We established significant relationship between several independent variables and Bike rental. We developed a regression model that can be applied to predict daily demand. We also found that the usage of bike rental is far more high for registered users as compared to casual user .Also weather has significant effect on bike ridership. A clear and sunny weather invites more riders as compared to rainy and snow weather. We trained various different models and performed . we found that XGBoost , randomForest and svr are best to capture the variance and nonlinearity of the dataset. the explanation could be absence of trend and seasonality in the bike demand pattern. We can thus conclude that the developed model can be used to predict the bike demand.

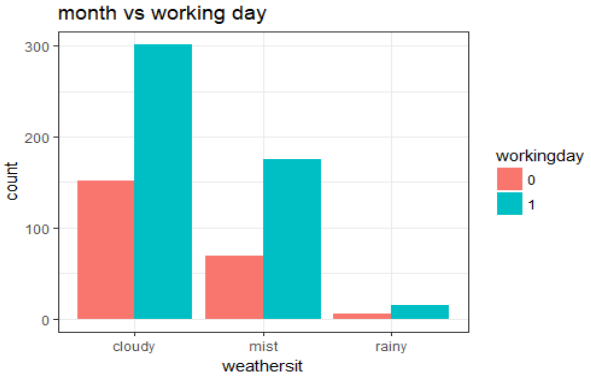
We can observe that from all the above models they are performing comparatively on average and therefore we have to select either XGboost , SVM or Random Forest Regressor models for better prediction

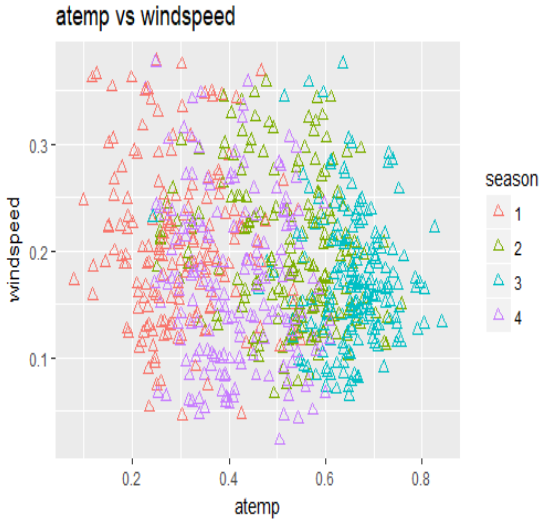
Appendix A - Extra Figures

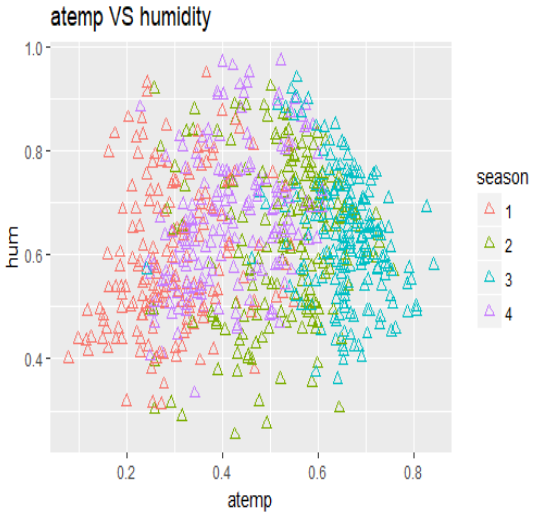


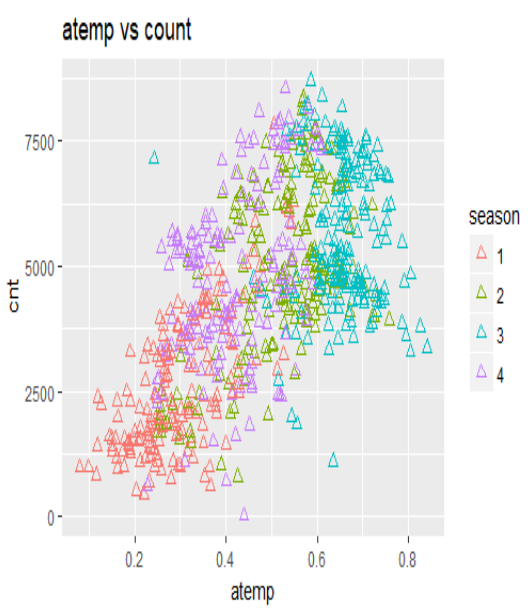


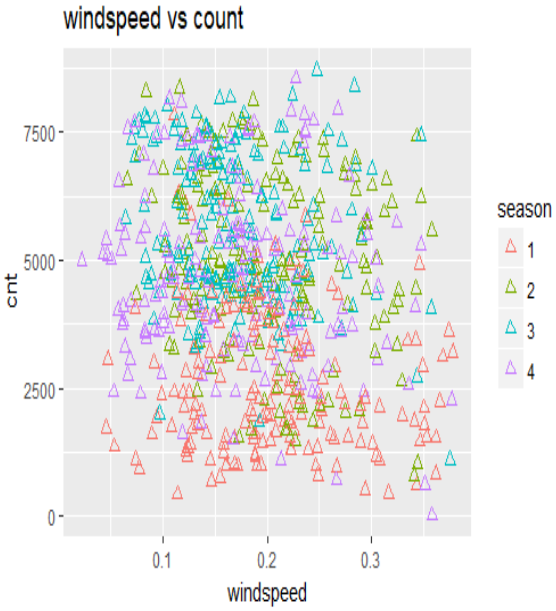


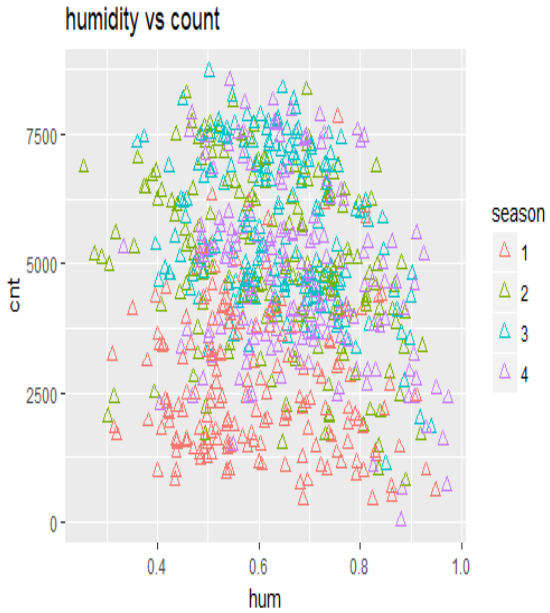


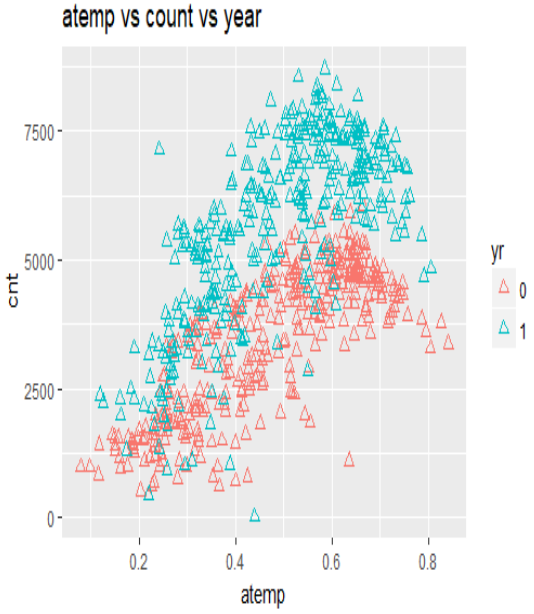


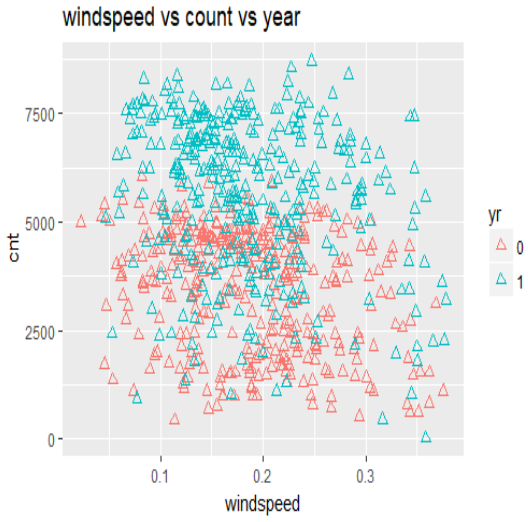


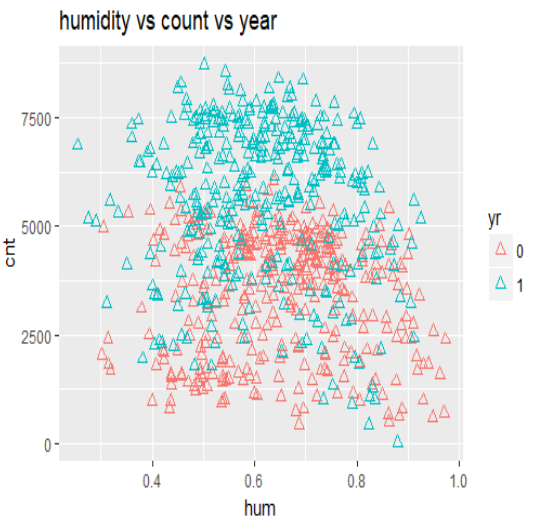

atemp vs count
cnt
atemp
season
1
2
3
4

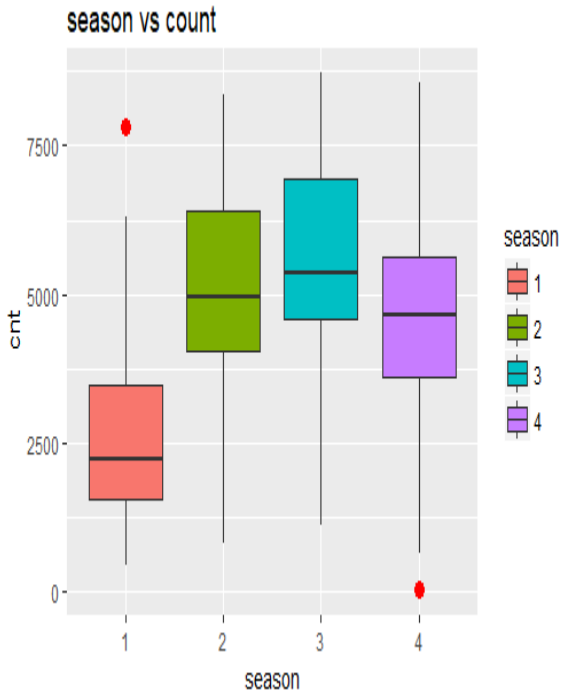

windspeed vs count
cnt
windspeed
season
1
2
3
4

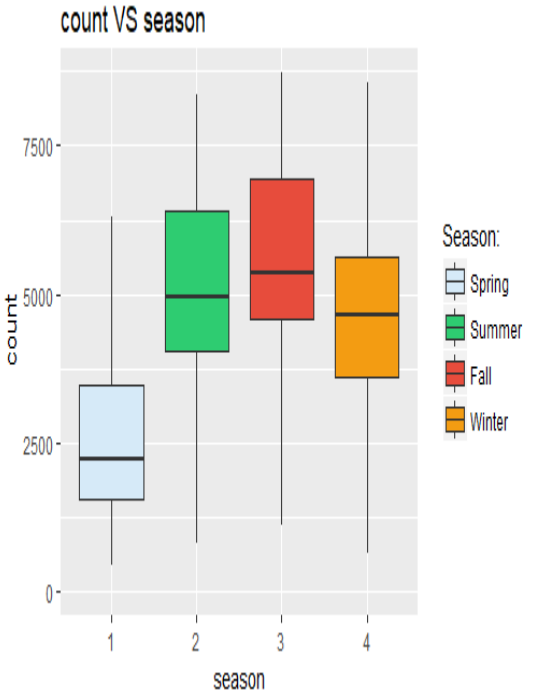

humidity vs count
cnt
hum
season
1
2
3
4

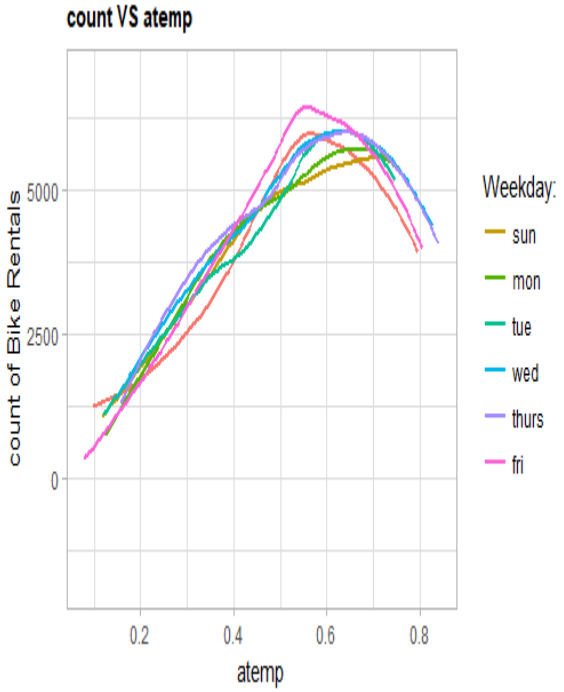

atemp vs count vs year
cnt
atemp
yr
0
1

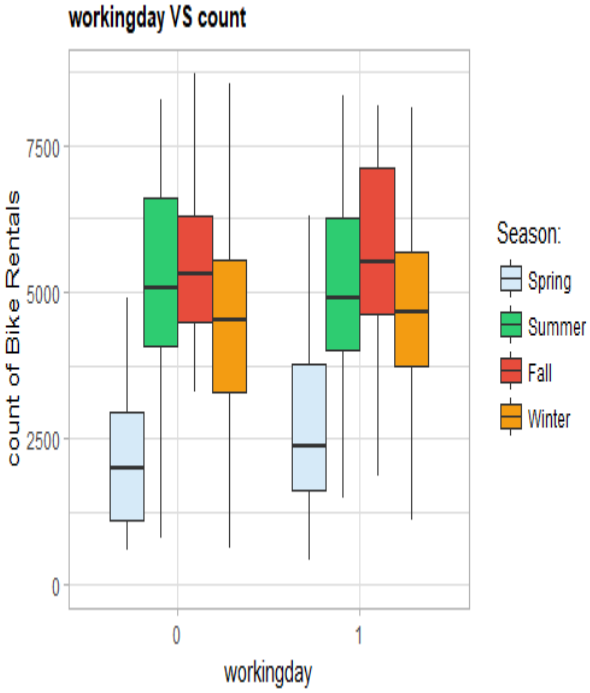

windspeed vs count vs year
cnt
windspeed
yr
0
1

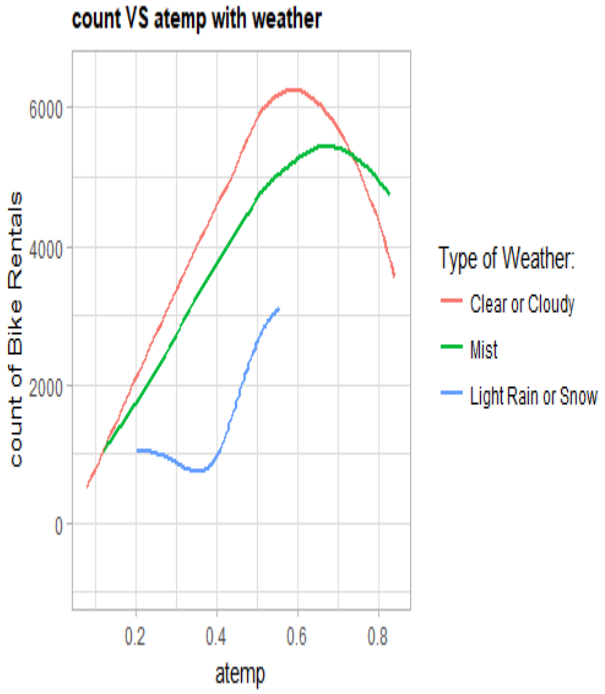

humidity vs count vs year
cnt
hum
yr
0
1

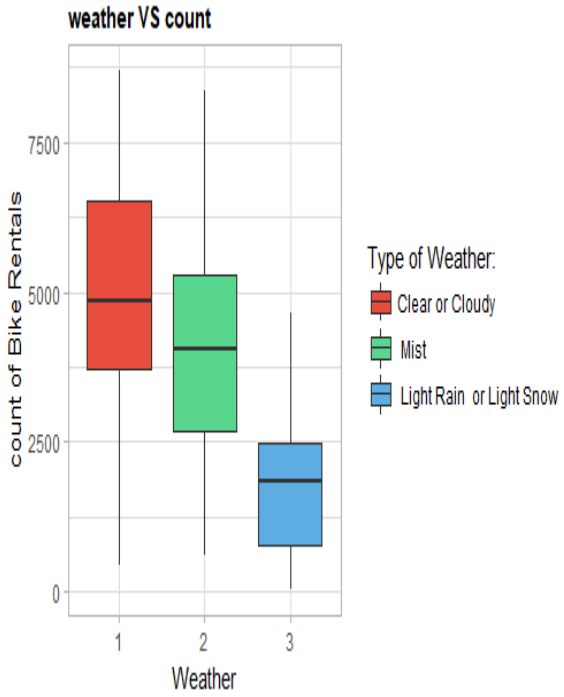

season vs count
cnt
7500
5000
2500
0
1
2
3
4
season
season
1
2
3
4

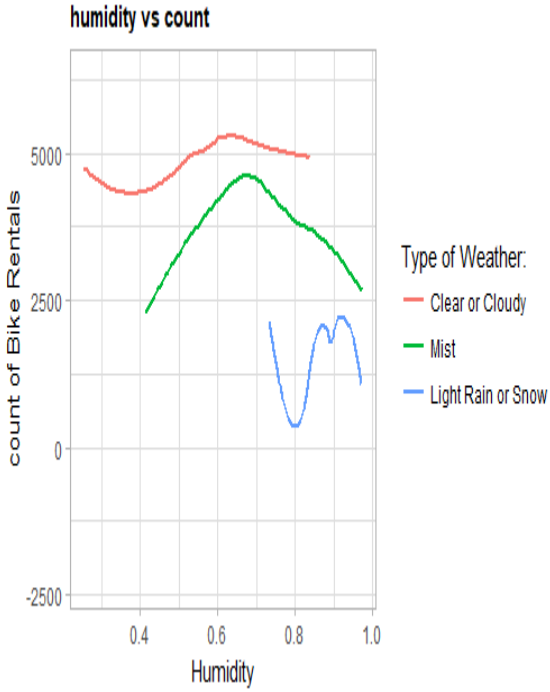

count VS season
count
7500
5000
2500
0
1
2
3
4
season
Season:
Spring
Summer
Fall
Winter

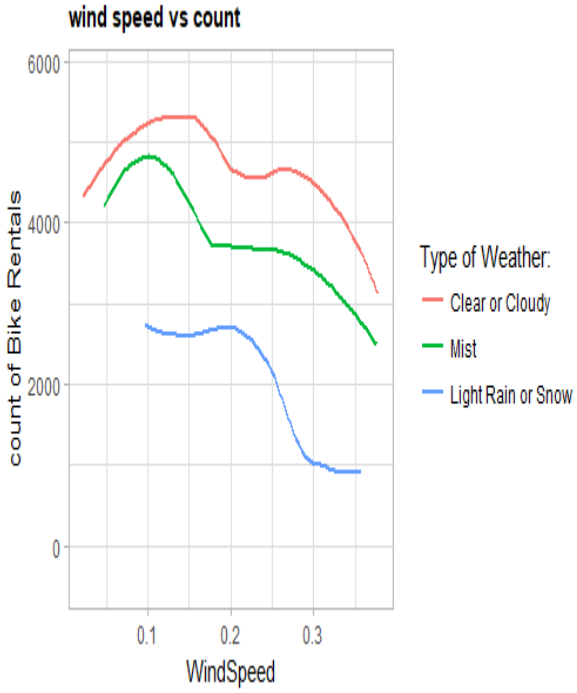

count VS atemp
count of Bike Rentals
5000
2500
0
0.2
0.4
0.6
0.8
atemp
Weekday:
sun
mon
tue
wed
thurs
fri

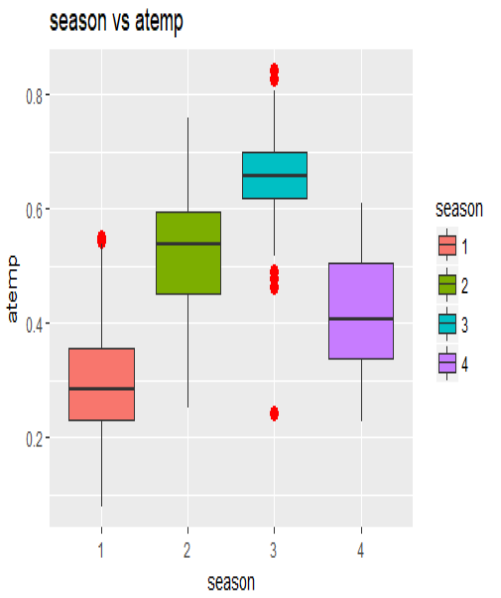

workingday VS count
count of Bike Rentals
7500
5000
2500
0
0
1
workingday
Season:
Spring
Summer
Fall
Winter

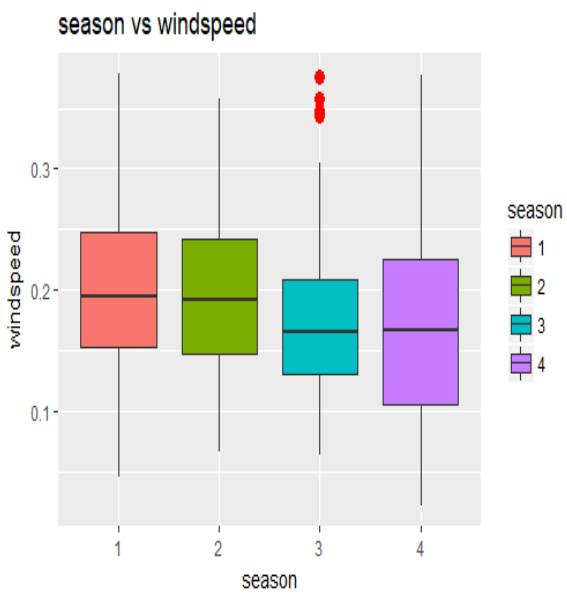

count VS atemp with weather
count of Bike Rentals
6000
4000
2000
0
0.2
0.4
0.6
0.8
atemp
Type of Weather:
Clear or Cloudy
Mist
Light Rain or Snow

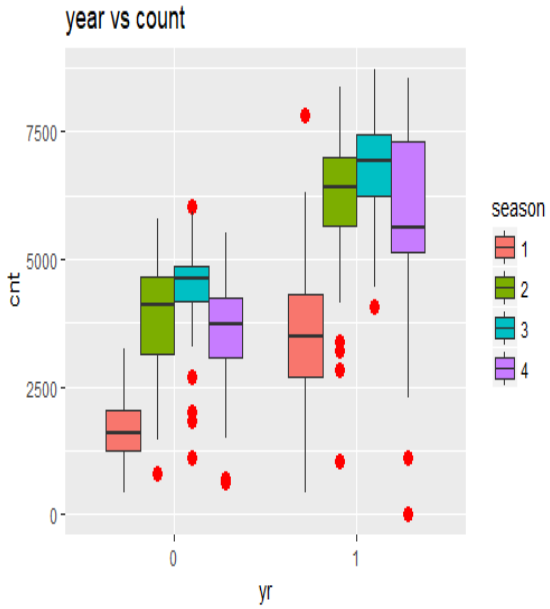

weather VS count
count of Bike Rentals
7500
5000
2500
0
1
2
3
Weather
Type of Weather:
Clear or Cloudy
Mist
Light Rain or Light Snow

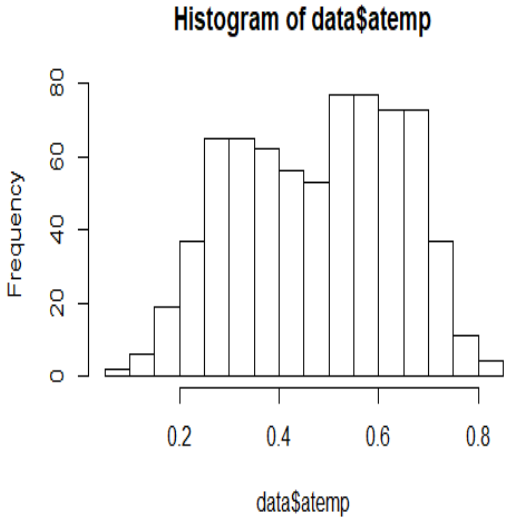

humidity vs count
count of Bike Rentals
5000
2500
0
-2500
0.4
0.6
0.8
1.0
Humidity
Type of Weather:
Clear or Cloudy
Mist
Light Rain or Snow

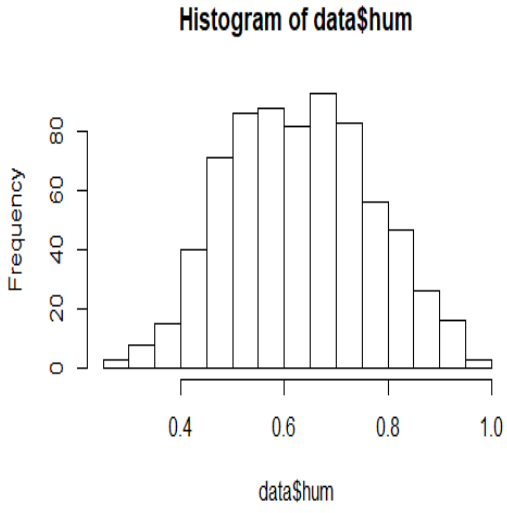

wind speed vs count
count of Bike Rentals
6000
4000
2000
0
0.1
0.2
0.3
WindSpeed
Type of Weather:
Clear or Cloudy
Mist
Light Rain or Snow

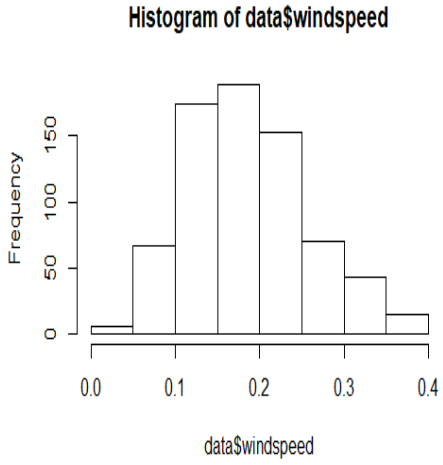

season vs atemp
atemp
season
1
2
3
4


season vs windspeed
windspeed
season
1
2
3
4


year vs count
cnt
yr
season
1
2
3
4


Histogram of data$atemp
Frequency
data$atemp


Histogram of data$hum
Frequency
data$hum


Histogram of data$windspeed
Frequency
data$windspeed

Appendix B – Code

R-code

```
rm(list=ls())

# setting working dir

getwd()

setwd("D:/project_2")


x = c("ggplot2", "corrgram", "DMwR", "caret", "randomForest", "unbalanced", "C50",
"dummies", "e1071", "Information",

    "MASS", "rpart", "gbm", "ROSE", 'sampling', 'DataCombine',
'inTrees',"readr","xgboost","class","Metrics")


lapply(x, require,character.only=T)

rm(x)

data=read_csv("day.csv")

dim(data)

class(data)

data=data.frame(data)

summary(data)

str(data)


# converting variablesinto their original data type


factor=c("season","yr","mnth","holiday","weekday","workingday","weathersit")
```

```
for(i in factor){
  data[,i]=as.factor(data[,i])}
```

#visualizations

```
ggplot(data,aes(x=atemp,fill=season))+geom_histogram()+scale_x_discrete(labels=c("2011",
"2012")) +
  theme_bw()+ggtitle("season vs atemp")
```

```
ggplot(data, aes(x=mnth,fill=workingday)) +
  geom_bar(position="dodge")+theme_bw()+ggtitle("month vs working day")
```

```
ggplot(data, aes(x=mnth,fill=weekday)) +
  geom_bar(position="dodge")+theme_bw()+ggtitle("month vs working day")
```

```
ggplot(data, aes(x=weekday,fill=weathersit)) +
  geom_bar(position="dodge")+theme_bw()+ggtitle("month vs working day")
```

```
ggplot(data, aes(x=weathersit,fill=workingday)) +
  geom_bar(position="dodge")+theme_bw()+ggtitle("month vs working
day")+scale_x_discrete(labels=c("cloudy","mist","rainy"))
```

```
ggplot(data, aes(x = atemp, y = hum, col= season)) +
  geom_point(shape = 2, size = 2)+ggtitle("atemp VS humidity")

ggplot(data, aes(x = atemp, y = windspeed, col= season)) +
  geom_point(shape = 2, size = 2)+ggtitle("atemp vs windspeed")


ggplot(data, aes(x = atemp, y = cnt, col= season)) +
  geom_point(shape = 2, size = 2)+ggtitle("atemp vs count")


ggplot(data, aes(x = windspeed, y = cnt, col= season)) +
  geom_point(shape = 2, size = 2)+ggtitle("windspeed vs count")


ggplot(data, aes(x = hum, y = cnt, col= season)) +
  geom_point(shape = 2, size = 2)+ggtitle("humidity vs count")


ggplot(data, aes(x = atemp, y = cnt, col= yr)) +
  geom_point(shape = 2, size = 2)+ggtitle("atemp vs count vs year")


ggplot(data, aes(x = windspeed, y = cnt, col= yr)) +
  geom_point(shape = 2, size = 2)+ggtitle("windspeed vs count vs year")
```

```
ggplot(data, aes(x = hum, y = cnt, col= yr)) +
  geom_point(shape = 2, size = 2)+ggtitle("humidity vs count vs year")



################################################################################


ggplot(data, aes(x = season, y = cnt, fill = season)) +
  geom_boxplot(outlier.color = adjustcolor("black", alpha.f = 0), na.rm = TRUE) +
  ylab("count") +
  ggtitle("count VS season") +
  scale_fill_manual(values = c("#D6EAF8", "#2ECC71", "#E74C3C", "#F39C12"),
                    name="Season:",
                    breaks=c(1, 2, 3, 4),
                    labels=c("Spring", "Summer", "Fall","Winter"))



################################################################################


ggplot(data, aes(x = atemp, y = cnt, color = weekday)) +
  geom_smooth(method = "loess", fill = NA, size = 1) +
  theme_light(base_size = 11) +
  xlab("atemp") +
  ylab("count of Bike Rentals") +
  ggtitle("count VS atemp") +


  scale_color_discrete(name = "Weekday:",
                       breaks = c(1, 2, 3, 4, 5, 6, 7),
```

```
                labels = c("sun","mon","tue","wed","thurs","fri","sat"))+
  theme(plot.title = element_text(size = 11, face="bold"))


##############################################################################


ggplot(data, aes(x = workingday, y = cnt, fill =season)) +
  geom_boxplot(outlier.color = adjustcolor("black", alpha.f = 0), na.rm = TRUE) +
  theme_light(base_size = 11) +
  xlab("workingday") +
  ylab("count of Bike Rentals") +
  ggtitle("workingday VS count") +
  scale_fill_manual(values=c("#D6EAF8", "#2ECC71", "#E74C3C", "#F39C12"),
                    name="Season:",
                    breaks=c(1, 2, 3, 4),
                    labels=c( "Spring", "Summer", "Fall","Winter")) +
  theme(plot.title = element_text(size = 11, face="bold"))

##############################################################################


ggplot(data, aes(x = atemp, y = cnt, color = weathersit)) +
  geom_smooth(fill = NA, size = 1) +
  theme_light(base_size = 11) +
  xlab("atemp") +
  ylab("count of Bike Rentals") +
  ggtitle("count VS atemp with weather") +
```

```
  scale_color_discrete(name = "Type of Weather:",
                       breaks = c(1, 2, 3, 4),
                       labels = c("Clear or Cloudy",
                                  "Mist",
                                  "Light Rain or Snow",
                                  " Heavy Rain + Ice Pallets ")) +
  theme(plot.title = element_text(size = 11, face="bold"))



##############################################################################


ggplot(data, aes(x = weathersit, y = cnt, fill = weathersit)) +
  geom_boxplot(outlier.color = adjustcolor("black", alpha.f = 0), na.rm = TRUE) +
  theme_light(base_size = 11) +
  xlab("Weather") +
  ylab("count of Bike Rentals") +
  ggtitle("weather VS count") +
  scale_fill_manual(values = c("#E74C3C", "#58D68D", "#5DADE2", "#F4D03F"),
                    name = "Type of Weather:",
                    breaks = c(1, 2, 3, 4),
                    labels = c("Clear or Cloudy ",
                               " Mist ",
                               " Light Rain  or Light Snow ",
                               "Heavy rain or ice pallets ")) +
  theme(plot.title = element_text(size = 11, face="bold"))


##############################################################################
```

```
ggplot(data, aes(x = hum, y = cnt, color = weathersit)) +
  geom_smooth(method = 'loess', fill = NA, size = 1) +
  theme_light(base_size = 11) +
  xlab("Humidity") +
  ylab("count of Bike Rentals") +
  ggtitle("humidity vs count") +
  scale_color_discrete(name = "Type of Weather:",
                       breaks = c(1, 2, 3, 4),
                       labels = c("Clear or Cloudy",
                                  "Mist",
                                  "Light Rain or Snow",
                                  "")) +
  theme(plot.title = element_text(size = 11, face="bold"))
##############################################################################


ggplot(data, aes(x = windspeed, y = cnt, color = weathersit)) +
  geom_smooth(fill = NA, size = 1) +
  theme_light(base_size = 11) +
  xlab("WindSpeed") +
  ylab("count of Bike Rentals") +
  ggtitle("wind speed vs count") +
  scale_color_discrete(name = "Type of Weather:",
                       breaks = c(1, 2, 3, 4),
                       labels = c("Clear or Cloudy",
                                  "Mist",
                                  "Light Rain or Snow",
```

```
                        "")) +
  theme(plot.title = element_text(size = 11, face="bold"))



################################################################################

ggplot(data,aes(x=season,y=cnt,fill=season))+
  geom_boxplot(outlier.color ="red",outlier.size = 3)+ggtitle("season vs count")

ggplot(data,aes(x=season,y=atemp,fill=season))+
  geom_boxplot(outlier.color ="red",outlier.size = 3)+ggtitle("season vs atemp")

ggplot(data,aes(x=season,y=windspeed,fill=season))+
  geom_boxplot(outlier.color ="red",outlier.size = 3)+ggtitle("season vs windspeed")

ggplot(data,aes(x=yr,y=cnt,fill=season))+
  geom_boxplot(outlier.color ="red",outlier.size = 3)+ggtitle("year vs count")

ggplot(data,aes(x=yr,y=atemp,fill=season))+
  geom_boxplot(outlier.color ="red",outlier.size = 3)+ggtitle("year vs count")



hist(data$cnt)
hist(data$atemp)
hist(data$hum)
```

```
hist(data$windspeed)
```

 # in the given data set the variables cnt is the sum of casual and registered

 # so we removing casual and registered

```
data=data[,-c(14,15)]
str(data)
```

 # checking for missing values

```
sum(is.na(data))
```

 # outlier analysis

```
ind=sapply(data,is.numeric)
data_num=data[,ind]
cn=colnames(data_num)
cn
for(i in cn){
  print(i)
  v=data[,i][data[,i]%in%boxplot.stats(data[,i])$out]
print(v)
}
```

#plotting for outliers

```
for (i in 1:length(cn)) {
  assign(paste0("gn",i), ggplot(aes_string( y = (cn[i]), x= "cnt") , data = subset(data)) +
          stat_boxplot(geom = "errorbar" , width = 0.5) +
          geom_boxplot(outlier.color = "red", fill = "grey", outlier.shape = 20, outlier.size = 1,
notch = FALSE)+
          theme(legend.position = "bottom")+
          labs(y = cn[i], x= "cnt")+
          ggtitle(paste("Boxplot" , cn[i])))
  #print(i)
}

gridExtra::grid.arrange(gn1,gn2,gn3,gn4,gn5,gn6,ncol=3,nrow=2)
```

 # removing outliers

```
cn
for(i in cn){
  v=data[,i][data[,i]%in%boxplot.stats(data[,i])$out]
  data=data[which(!data[,i]%in%v),]
}
```

#attaching data

```
attach(data)
```

 # correlation analysis

```
corrgram(data_num,order=F,upper.panel = panel.pie,text.panel = panel.txt,main="correlation plot")
```

 # checking anova

```
m1=aov(cnt~season+yr+mnth+holiday+weekday+workingday+weathersit)
summary(m1)
par(mfrow=c(1,2))
plot(m1)
```

 # removing the variables which are not carrying much information

```
data_del=subset(data,select=-c(instant,temp,holiday,dteday))
```

 # as the data is normally distributed we are not doing feature scaling

dividing the data into train and test sets

```
rmExcept("data_del")
data=data_del
dt_ind=sample(1:nrow(data),0.8*nrow(data))
train=data[dt_ind,]
test=data[-dt_ind,]
```

 # applying desicion tree

```
rg=rpart(cnt~.,train,method="anova")
rg
dt_prd=predict(rg,test[,-10])

dt_prd
```

 # as it is a time series data we have to use rmse and rmsle

```
regr.eval(test[,10],dt_prd,stats = c("mape","rmse","mae"))
actual=test[,10]
pred=data.frame(dt_prd)
err=actual-pred
rmse <- function(error)
{
  sqrt(mean(error^2))
}

rmse(err)
require(Metrics)
rmsle(test[,10],dt_prd)
rmsle_dT=rmsle(test[,10],dt_prd)
```

 # rmse=932.5

#rmsle=0.30367

building regression model

```
lm_mod=lm(cnt~.,train)
summary(lm_mod)
lm_prd=predict(lm_mod,test[,-10])

lm_prd=round(lm_prd)
lm_prd
regr.eval(test[,10], lm_prd, stats = "rmse")
summary(lm_prd)
```

 # as we can see there are some negetive values so we have to replace those values with min cnt values

#min(cnt)

#max(cnt)

```
#pred = lm_prd
#pred[pred<=0] = 22

rmsle(test[,10],pred)
rmsle_lm=rmsle(test[,10],pred)

#rmse = 850.49
#rmsle = 0.30398
```

#random forest

```
rf=randomForest(cnt~.,train,ntree=100,importance=T,bootstrap=T)
rf


rf_prd=predict(rf,test[,-10])
rf_prd
summary(rf_prd)
regr.eval(test[,10],rf_prd,stats = "rmse")
rmsle(test[,10],rf_prd)
rmsle_rf=rmsle(test[,10],rf_prd)

#for 100 trees
#rmse = 753.159
#rmsle = 0.264
```

 # applying XGboost

```
Prediction <- function(model,data,features){
  predictions <- numeric(nrow(data))
  counter <- 1
  for (model in model){
    if (class(model)[1]=="xgb.Booster"){
      usedata <- data.matrix(data[features[[counter]]])
    } else{
      usedata <- data[features[[counter]]]
    }
```

```
    pred_test <- predict(model,usedata)
    pred_test[pred_test<0]=0
          predictions <- predictions+pred_test
    counter <- counter + 1
  }
  return(predictions)
}

evaluate <- function(true,predictions){
  return(rmsle(true, predictions))
}

features=c("season","yr","mnth","weekday","workingday","weathersit","atemp","hum","win
dspeed")

xgb_mod <- xgboost(data = data.matrix(train[,features]),label = train$cnt,objective =
"reg:linear",eval_metric = "rmse",max.depth = 5,nround = 10)

xgb_prd = getPrediction(list(xgb_mod),test,list(features))
evaluate(test$cnt,xgb_prd)
rmsle_xg=evaluate(test$cnt,xgb_prd)
regr.eval(test[,10],xgb_prd,stats = "rmse")

#RMSE=702.32
#RMSLE=0.2325

xgb.ggplot.importance(xgb.importance(model=xgb_mod))
```

 # svm model

```
svm_mod<- svm(cnt~.,data=train,kernel="radial",scale=TRUE,epsilon=0.1,gamma=0.17)

svm_mod
svm_prd = Prediction(list(svm_mod),test,list(features))
evaluate(test$cnt,svm_prd)
rmsle_svm=evaluate(test$cnt,svm_prd)
regr.eval(test[,10],svm_prd,stats="rmse")

#RMSE=603.32
#RMSLE=0.2525

prds=data.frame(svm_prd,xgb_prd,lm_prd,rf_prd,dt_prd)
out2=cbind(round(xgb_prd),test)
names(out2)[1]="predicted"
out2=out2[,c(2:11,1)]
out2=out2[,-10]
write.csv(out2,"sample_output.csv", row.names = F)
```

Python-code :

```
import os
import pandas as pd
import numpy as np
from fancyimpute import KNN
import matplotlib.pyplot as plt
```

```
from scipy.stats import chi2_contingency

import seaborn as sns

from random import randrange, uniform

from subprocess import check_output

from sklearn.model_selection import train_test_split

from sklearn import metrics

from sklearn.model_selection import cross_val_score
```

In [ ]:

```
os.getcwd()
```

In [ ]:

```
data=pd.read_csv("day.csv")
```

In [ ]:

```
data.shape
```

In [ ]:

```
data.describe()
```

In [ ]:

```
data.corr()
```

In [ ]:

```
data=data.rename(columns={'dteday':"datetime",'yr':'year','mnth':'month','hum':'humidity','cnt':'co
unt'})
```

In [ ]:

```
data['season']=data['season'].astype('category')
```

In [ ]:

```
data=data.rename(columns={'weathersit':'weather'})
```

In [ ]:

```
data['year'] = data['year'].astype('category')

data['month'] = data['month'].astype('category')

data['holiday'] = data['holiday'].astype('category')
```

```
data['weekday'] = data['weekday'].astype('category')

data['workingday'] = data['workingday'].astype('category')

data['weather'] = data['weather'].astype('category')

In [ ]:

data.dtypes

In [ ]:

fig, ax = plt.subplots()

sns.barplot(data=data[['season','count']],

        x='season',

        y='count',

        ax=ax)


plt.title('count by Season')

plt.ylabel('count')

plt.xlabel('Season')


tick_val=[0, 1, 2, 3]

tick_lab=['Spring','Summer','Fall',"winter"]

plt.xticks(tick_val, tick_lab)


plt.show()

In [ ]:

fig, ax = plt.subplots()

sns.barplot(data=data[['month','count']], x='month', y='count', ax=ax)


plt.title('count by month')

plt.ylabel('count')
```

```
plt.xlabel('Month')


tick_val=[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]
tick_lab=['January', 'February', 'March', 'April', 'May', 'June', 'July', 'August', 'September', 'October',
'November', 'December']
plt.xticks(tick_val, tick_lab)


plt.show()
In [ ]:
fig = plt.subplots(figsize=(7,5))
sns.boxplot(data=data[['count', 'casual', 'registered']])
In [ ]:


fig, ax = plt.subplots(figsize=(7,7))
sns.pointplot(data=data[['month', 'casual', 'registered']],
        x='month',
        y='casual',
        ax=ax,
        color='red')


sns.pointplot(data=data[['month', 'casual', 'registered']],
        x='month',
        y='registered',
        ax=ax,
        color='black')


tick_val=[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]
```

```
tick_lab=['January', 'February', 'March', 'April', 'May', 'June', 'July', 'August', 'September', 'October',
'November', 'December']

plt.xticks(tick_val, tick_lab)


plt.title('Casual and Registered by Month')

plt.ylabel('count')

plt.xlabel('Month')


plt.show()

In [ ]:

plot= sns.FacetGrid(data=data,

        col='season',

        row='weather',hue='season')

plot.map(plt.hist,'count')


plt.subplots_adjust(top=1.0)

plot.fig.suptitle('count vs weather')


plot.set_xlabels('count')

plot.set_ylabels('Frequency')


plt.show()

In [ ]:

by_week = data.groupby(['weekday'])['count'].sum().reset_index()

ax = sns.barplot(x = by_week['weekday'], y = by_week['count'])

ax.set(xlabel='weekday', ylabel='count')

plt.show()

In [ ]:
```

```
by_month = data.groupby(['month'])['count'].sum().reset_index()
ax = sns.barplot(x = by_month['month'], y = by_month['count'])
ax.set(xlabel='month', ylabel='count')
plt.show()
In [ ]:
plt.style.use('ggplot')

data.boxplot(column='count', by=['year','month'])

plt.title('Number of bikes rented per month')
plt.xlabel('')

plt.ylabel('Number of bikes')

plt.show()
In [ ]:
data.dtypes
In [ ]:
data.head(4)
In [ ]:
```

checking for missing values

```
data.isnull().sum()
In [ ]:
```

checking for outliers

```
data.columns
In [ ]:
```

 # saving numeric variables

```
cnames=['temp', 'atemp', 'humidity', 'windspeed',
      'casual', 'registered', 'count']
```

In [ ]:

```
q75,q25= np.percentile(data['humidity'], [75,25])
```

In [ ]:

```
iqr=q75-q25


min=q25-(iqr*1.5)
max=q75+(iqr*1.5)


data=data.drop(data[data['humidity']<min].index)
data=data.drop(data[data['humidity']>max].index)
```

In [ ]:

```
q75,q25= np.percentile(data['windspeed'], [75,25])
```

In [ ]:

```
iqr=q75-q25


min=q25-(iqr*1.5)
max=q75+(iqr*1.5)


data=data.drop(data[data['windspeed']<min].index)
data=data.drop(data[data['windspeed']>max].index)
```

Feature selection

In [ ]:

 # corelation analysis

dt_corr=data.loc[:,cnames]

In [ ]:

 # set width and hight of the plot

f,ax=plt.subplots(figsize=(10,7))

#generating corr plot

corr=dt_corr.corr()

 # plot using seaborn lib

sns.heatmap(corr,mask=np.zeros_like(corr,dtype=np.bool),annot=True,cmap=sns.diverging_palette(220,10,as_cmap=True),square=True,ax=ax)

In [ ]:

from statsmodels.formula.api import ols

import statsmodels.api as sm

In [ ]:

aov=ols('count~season+year+month+holiday+weekday+workingday+weather',data=data).fit()

In [ ]:

aov_table=sm.stats.anova_lm(aov,typ=2)

In [ ]:

aov_table

In [ ]:

 # droping the variables which are not carrying useful information

data=data.drop(['instant','datetime','holiday','casual','registered','temp'],axis=1)

In [ ]:

```
data.shape
```

In [ ]:

as the given data is normalized so we are not applying feature scaling

In [ ]:

```
dep=data['count']
```

In [ ]:

```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split( data, dep, test_size=0.3, random_state=42)
```

In [ ]:

```
def rmsle(y, y_,convertExp=True):
    if convertExp:
        y = np.exp(y),
        y_ = np.exp(y_)
    log1 = np.nan_to_num(np.array([np.log(v + 1) for v in y]))
    log2 = np.nan_to_num(np.array([np.log(v + 1) for v in y_]))
    calc = (log1 - log2) ** 2
    return np.sqrt(np.mean(calc))
```

In [ ]:

#Random forest

```
from sklearn.ensemble import RandomForestRegressor
```

In [ ]:

```
rf_mod=RandomForestRegressor(n_estimators=500).fit(x_train,np.log1p(y_train))
```

In [ ]:

```
pred=rf_mod.predict(x_test)
```

In [ ]:

```
print ("RMSLE Value For Random Forest: ",rmsle(np.exp(np.log1p(y_test)),np.exp(pred),False))
```

In [ ]:

Decision tree

```
from sklearn.tree import DecisionTreeRegressor

dt_mod=DecisionTreeRegressor(max_depth=2).fit(x_train,np.log1p(y_train))
```

In [ ]:

```
dt_pred=dt_mod.predict(x_test)
```

In [ ]:

```
print ("RMSLE value For decision tree: ",rmsle(np.exp(np.log1p(y_test)),np.exp(dt_pred),False))
```

In [ ]:

```
from sklearn.linear_model import LinearRegression
```

In [ ]:

 # linear regression

```
lm_mod=LinearRegression().fit(X = x_train,y =np.log1p(y_train))
```

In [ ]:

```
lm_pred = lm_mod.predict(x_test)
```

In [ ]:

```
rmsle(np.exp(np.log1p(y_test)),np.exp(lm_pred),False)
```

In [ ]:

```
print ("RMSLE Value For Linear Regression :
",rmsle(np.exp(np.log1p(y_test)),np.exp(lm_pred),False))
```

In [ ]:

```
from sklearn.ensemble import GradientBoostingRegressor


gbm=GradientBoostingRegressor(n_estimators=100,alpha=0.01).fit(X = x_train,y = np.log1p(y_train))
```

In [ ]:

```
preds = gbm.predict(X= x_test)


rmsle(np.exp(np.log1p(y_test)),np.exp(preds),False)
```

In [ ]:

```
print ("RMSLE Value For Gradient Boost: ",rmsle(np.exp(np.log1p(y_test)),np.exp(preds),False))
```

In [ ]:

```
## support vector regressor
from sklearn.svm import SVR
from sklearn.metrics import mean_squared_log_error,mean_squared_error,
r2_score,mean_absolute_error
```

In [ ]:

```
sv=SVR().fit(x_train,y_train)
```

In [ ]:

```
svr_prd=sv.predict(x_test)
```

In [ ]:

```
rmsle=np.sqrt(mean_squared_log_error(svr_prd,y_test))
```

In [ ]:

```
rmsle
```

In [ ]:

```
print('RMSLE value for svr regressor :',rmsle)
```

In [ ]:

```
# knn
from sklearn.neighbors import KNeighborsRegressor
```

In [ ]:

```
knn=KNeighborsRegressor(n_neighbors=9).fit(x_train,y_train)
```

In [ ]:

```
knn_prd=knn.predict(x_test)
```

In [ ]:

```
rmsle=np.sqrt(mean_squared_log_error(knn_prd,y_test))
```

In [ ]:

```
rmsle
```

In [ ]:

```
print('RMSLE value for knn regressor :',rmsle)
```