Predication of bike rental count based on the environmental and seasonal settings.

Gideon Anthony D'Souza

Jan 2019

Contents

Chapter 1

Introduction

1.1 Problem Statement

The objective of this Case is to Predication of bike rental count on daily based on the environmental and seasonal settings.

1.2 Data

In order to predict future bike rental count based on the environmental and seasonal data, it is required to build a regressor model on the data set provided. Table 1.1 & 1.2 below is a sample of the data set that is using to predict the bike rental count.

Table 1.1: Bike rental count (Columns(1-8))

index	instant	dteday	season	yr	mnth	holiday	weekday	workingday
0	1	2011-01-01	1	0	1	0	6	0
1	2	2011-01-02	1	0	1	0	0	0
2	3	2011-01-03	1	0	1	0	1	1
3	4	2011-01-04	1	0	1	0	2	1
4	5	2011-01-05	1	0	1	0	3	1

Table 1.2: Bike rental count (Columns(9-16)

weathersit	temp	atemp	hum	windspeed	casual	registered	cnt
2	0.3441	0.3636	0.8058	0.1604	331	654	985
2	0.3634	0.3537	0.696	0.2485	131	670	801
1	0.1963	0.1894	0.4372	0.2483	120	1229	1229
1	0.2	0.2121	0.5904	0.1602	108	1454	1454
1	0.2269	0.2292	0.4369	0.1869	82	1518	1518

1.2.1 Data Overview:

As shown in the Table1.3 there are 13 features or predictor variables to predict the bike count which is the target or label. There are three target variables as shown in Table1.3.

Table 1.3:

S.No	Predictor	Target
1	instant	casual
2	dteday	registered
3	season	cnt
4	yr	
5	mnth	
6	holiday	
7	weekday	
8	workingday	
9	weathersit	
10	temp	
11	atemp	
12	hum	
13	windspeed	

1.2.2 Data Description

From the given in the metadata file, the feature and target variables names and descriptions are given in table 1.4

Table 1.4: Feature and target description. Detailed description for features with *, **, ***, ##, ###

S.No	Predictor/target variables	Description
1	instant	Record index
2	dteday	Date of record
3	season	Season (1:spring, 2:summer, 3:fall, 4:winter)
4	yr	Year (0: 2011, 1:2012)
5	mnth	Month (1 to 12)
6	holiday	weather day is holiday or not (extracted from Holiday Schedule)
7	weekday	Day of the week
8	workingday	If day is neither weekend nor holiday is 1, otherwise is 0.
9	weathersit	Weather site information extracted from Freemeteo**
10	temp	Normalized temperature in Celsius.**
11	atemp	Normalized real feel temperature in Celsius.**
12	hum	Normalized humidity.**
13	windspeed	Normalized wind speed.**
14	casual	Casual bike users.
15	registered	Registered bike users.
16	cnt	Total bike users.

- **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_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 values are divided to 67 (max).

From the information provided by the metadata. The continuous and categorical feature variables can clearly be separated. Table 1.5 shows the categorical and continuous variables provided in the data set. It can also be seen that there are three target variables which are all continuous and the factor variable dteday a date variable.

Table 1.5: List of categorical ans continuous features present in the data set.

S.No	Categorical features	Continuous features
1	season	instant
2	yr	temp
3	mnth	atemp
4	holiday	hum
5	weekday	windspeed
6	workingday	
7	weathersit	
8	temp	
9	atemp	
10	hum	
11	windspeed	

Chapter 2

Data Pre-Processing

2.1 Data Preparation

In order to create a model for the data it is first required to look into the data. First the data types of the feature and target variables are checked in order to see if it complies with the information given in the meta data file (data description file). In order to perform the aforementioned the variables are split according to their data types by the function listing 2.1.

Listing 2.1: Function to seperate the variables names into their respective data types.

```
def dtype_separator(df, col_names):
    '''
    ########################################################################
    This function will segregate the different data types present in the dataframe.
    ########################################################################
    Input -
    * df - The data frame to be analysed.
    *A list of the column names.
    Output -
    * obj - columns containing object data type.
    * num - columns containing numerical data type (int64/float64).
    * bool_d - columns containing bool data type.
    * unknown = columns containing data type such as datetime64, timedelta[ns] and others.
    ########################################################################
    '''
    obj = []
    num = []
    bool_d = []
    unknown = []
    for i in range(len(col_names)):
        if df.iloc[:,i].dtype == 'O':
            obj.append(col_names[i])
        elif df.iloc[:,i].dtype == 'int64' or df.iloc[:,i].dtype == '
```

```
            float64':
24              num.append(col_names[i])
25          elif df.iloc[:,i].dtype == 'bool':
26              bool_d.append(col_names[i])
27          else:
28              unknown.append(col_names[i])
29      return obj, num, bool_d, unknown
```

Function 2.1 outputs list of columns according to their data types. It was noted that the features season, yr, mnth, holiday, weekday, workingday and weathersit were incorrectly saved as numerical data type and hence they are to be converted back to categorical data type. From viewing the meta data file and the data set, it can seen that the feature instance is just the index of the records and causal and registered are the count of the respective type of users which is not required as we have the total count. Hence instance, causal and registered columns are dropped from the data set. The dteday is the date of the records which is not required as we are not conducting a time series analysis. Hence even dteday column is dropped from the data set. Now there is eleven features and one label. Next the probability distribution of each numerical column is plotted for visual examination to see if the data was normally distributed. Figure 2.1 is the probability distribution plots for each of the continuous variable. From figure 2.1 it can be see that they are all mostly normally distributed but windseed is slightly skewed.

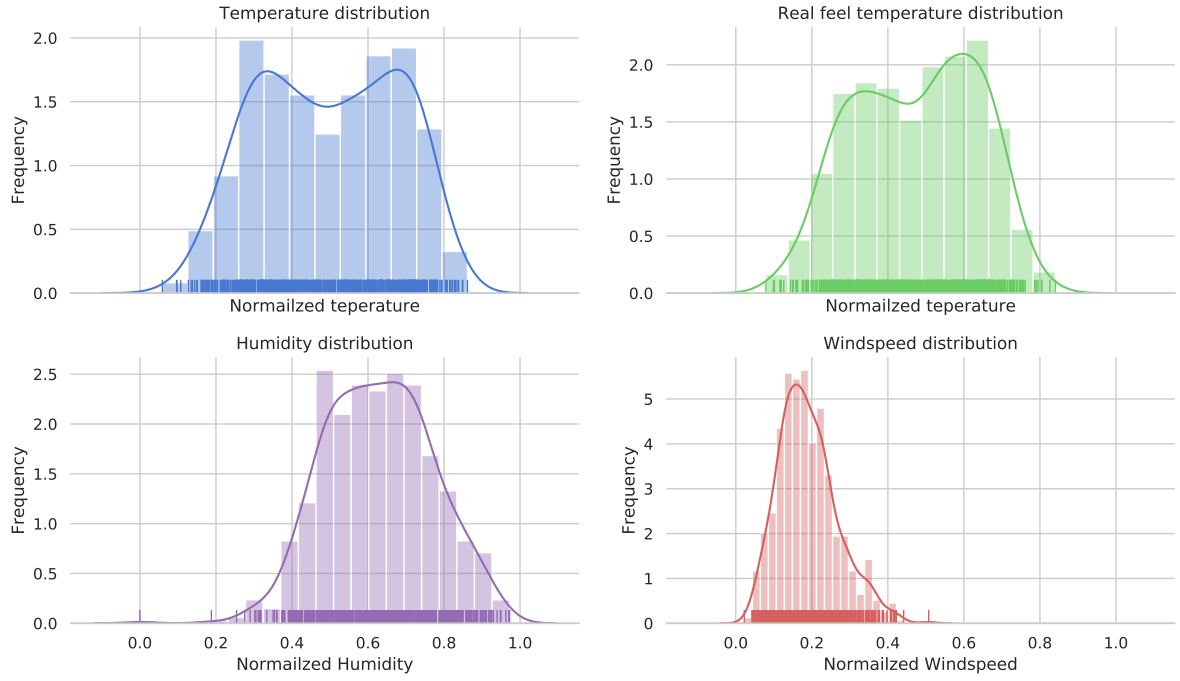


Figure 2.1: Distribution plots of the continuous variables.

The skewness of the variables are further analyzed by running the code in listing 2.2. The output for 2.2 is shown in figure 2.2.

Listing 2.2: Code to analyse the skewness of the continous variables distribution.

```
1 for i in num_dtype:
```

```
    from scipy import stats
    skewness = stats.describe(df.loc[:,i])
    print("statistical properities of {0:5s}:".format(i))
    print(skewness)
    print("****************************************************************")
```

Check the skewness of the continous variables

```
In [307]: print("****************************************************************")
          for i in num_ctype:
              from scipy import stats
              skewness = stats.describe(df.loc[:,i])
              print("statistical properities of {0:5s}:".format(i))
              print(skewness)
              print("****************************************************************")
```

```
****************************************************************
statistical properities of temp :
DescribeResult(nobs=731, minmax=(0.0591304, 0.861667), mean=0.495384788508892, variance=0.03350766717740828, skewness=-
0.05440902400571610, kurtosis=-1.1194225480473057)
****************************************************************
statistical properities of atemp:
DescribeResult(nobs=731, minmax=(0.0790696, 0.8408959999999999), mean=0.4743539886469569084, variance=0.026556345661055174, skewness=-0.1308188980737412, kurtosis=-0.9866019052943136)
****************************************************************
statistical properities of hum  :
DescribeResult(nobs=731, minmax=(0.0, 0.9725), mean=0.6278940629274967, variance=0.02028604714193028, skewness=-0.06964015783152368, kurtosis=-0.07228631791987006)
****************************************************************
statistical properities of windspeed:
DescribeResult(nobs=731, minmax=(0.0223917, 0.507463), mean=0.1904862116279070, variance=0.006005919960192755, skewness=0.6759547264275362, kurtosis=0.39992023832685497)
****************************************************************
statistical properities of cnt  :
DescribeResult(nobs=731, minmax=(22, 8714), mean=4504.3488372093025, variance=3752788.2082828926, skewness=-0.047255557553620630, kurtosis=-0.8145762269613592)
****************************************************************
```

Figure 2.2: Output for code in listing 2.2.

Hence from figure 2.2. It can said that the continuous variables are all mostly normally distributed but there is a slight skew in all of them. This could be due to some outliers present in the data set. Those outliers will be dealt with in further parts of this report. The data distribution of the categorical variables were also checked and it was found all the variables except workingdays, holiday and weathersit where almost equally distributed.

2.2 Missing Value Analysis

The data set was checked for missing values and was found that there were no missing values present in the data set.

2.3 Outlier Analysis

From figure2.1 & fig:skew is can be observed that most of the variables are skewed. This could be due to outliers present in the data set. In order to visualize outliers, the classic approach of using boxplot is used. Figure 2.3 & 2.4 Shows the outlier analysis using boxplot. it can be seen from figure 2.3 only hum which is humidity and windspeed variables have outliers present in them. In order to decide as to how to deal with the outliers in the data set, the percentage of outliers in the data set is first calculated by sample code in listing 2.3. the output of listing 2.3 shows that the **percentage of outliers present in the data set in 2.052%**. In such a case it would be best to omit the records with outliers from the data set as it would not

have much affect on the overall data. After removal of the outliers from the data set, the boxplot analysis is again conducted on the new data set. This is because when data is removed from the data set the mean of the data set and other statistical properties also changes and this may give is to new outliers.

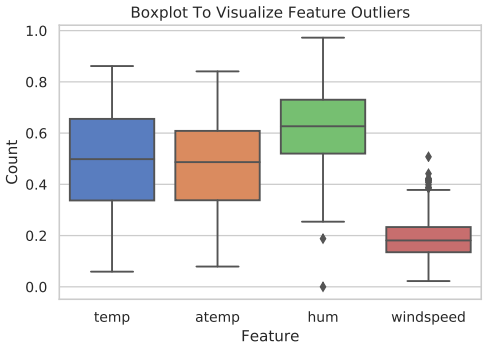


Figure 2.3: Outliers present in the feature variables.

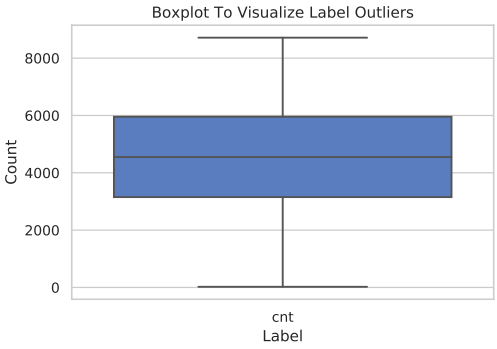


Figure 2.4: Outliers present in the target variable.

Listing 2.3: Function to seperate the variables names into their respective data types.

```
l_q = df.quantile(0.25)
u_q = df.quantile(0.75)
iqr = u_q - l_q
total_outliers = ((df<(l_q - 1.5*iqr))|(df>(u_q + 1.5*iqr))).sum()
outliers = total_outliers.sum()

r,c = df.shape
outlier_percentage = (outliers/r)*100
print("Outlier percentage :",round(outlier_percentage,3))
```

Box plot in figure 2.3 & 2.4 shows that there is still outliers present in windspeed variable. By running listing 2.3 on the new data set it can be seen that the **percentage of outliers present in the data set in 0.418%**.

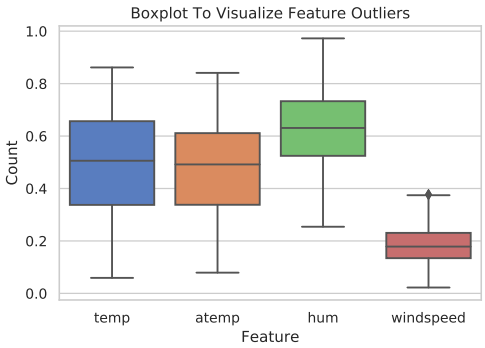


Figure 2.5: Outliers present in the feature variables.

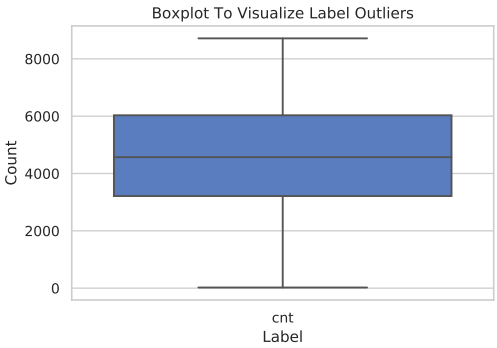


Figure 2.6: Outliers present in the target variable.

Now we remove these outliers from the data set and repeat the process of plotting the new data set

in box plot and examining for outliers. this time only one outlier is found in windspeed variable. That amounts to **0.14% percentage of outliers present.** This too is removed and the data set is again checked for outliers. Figure 2.7 & 2.8 shows that the data set is free from outliers. The data set is now reduced from 731 records to 713 records.

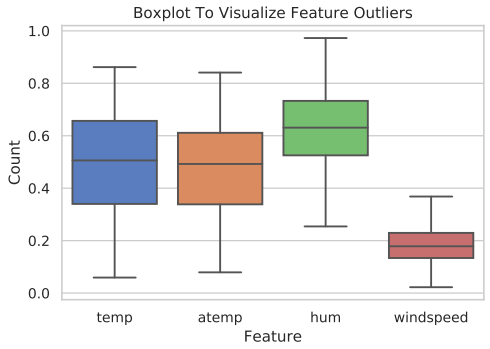


Figure 2.7: No outliers present in the feature variables.

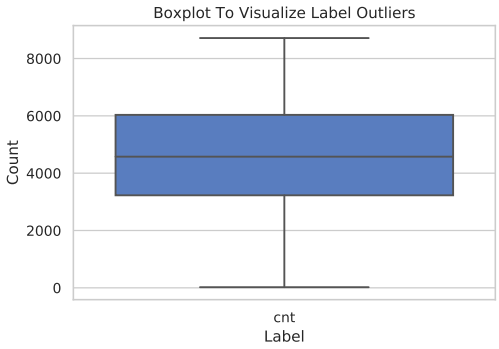


Figure 2.8: No outliers present in the target variable.

2.4 Data Understanding

In order to get further insight and understand of the data set. It is to see as to how different features interact with each other and the target. First the amount of bike rental counts for each day of the week is analyzed.

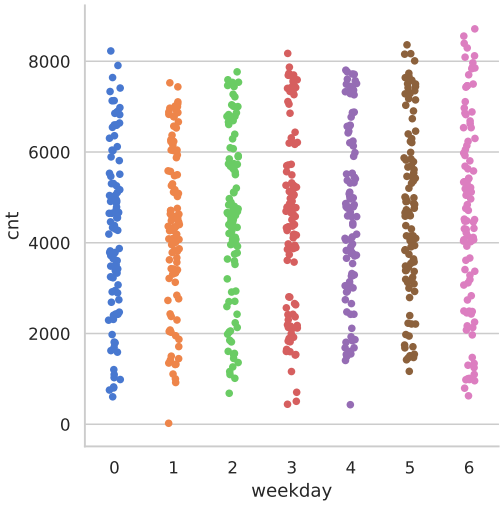


Figure 2.9: Variation in bike rental counts over the days of the week.

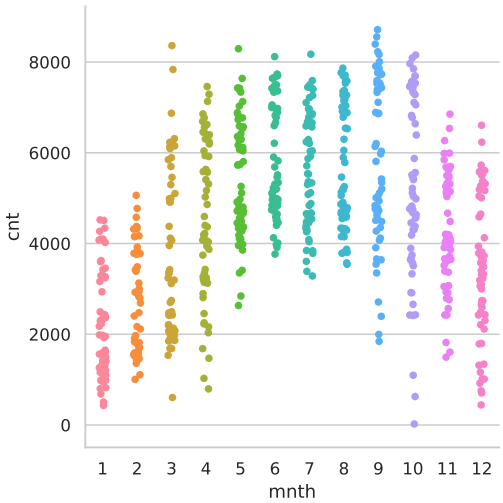


Figure 2.10: Variation in bike rental counts over the months.

From figure 2.9 it can be said that the majority of the bike rental each day is between 3500 - 6000. Whereas days 0, 5 and 6 is almost evenly distributed along the cnt ('bike count') axis. Figure 2.10 shows that the maximum amount of rentals happen in the months 4 - 10. These months could reach rentals counts of 4000 to 8000 and even more.

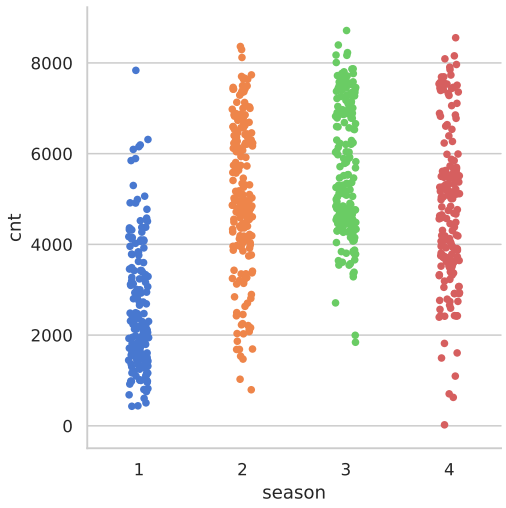


Figure 2.11: Variation in bike rental based on the season. (1:spring, 2:summer, 3:fall, 4:winter).

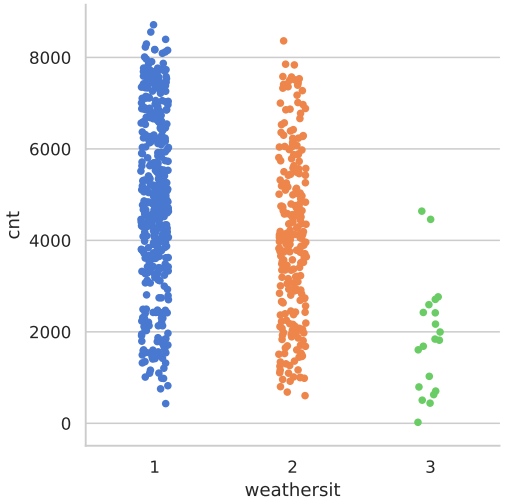


Figure 2.12: Variation in bike rental based on the weather site. (1: Clear, Few clouds, Partly cloudy; 2: Mist + Cloudy, Mist + Broken clouds, Mist + Few clouds, Mist.; 3: Light Snow, Light Rain + Thunderstorm + Scattered clouds, Light Rain + Scattered clouds.

Figure 2.11 shows the variation in bike rentals based on the season. It can be observed that season 3 which is fall is the most busiest, having majority of the daily bike count between 4000 - 8000 rentals. On the other hand season 1 which is spring is the least where the majority of the daily bike count is mostly in the range 500 - 4000. Figure 2.12 it can be seen that weather site forecast 3 has barley any bike rentals. This weather corresponds to Light Snow, Light Rain, Thunderstorm or or Scattered clouds.

Figure 2.13 it can be seen that majority of the bike rental counts are between humidity levels of 0.25 - 0.75 and normalized temperature levels of 0.4 - 0.8. For humidity levels of 0.75-1 there is significantly lesser bike rental counts. From figure 2.14 the variation in bike rental based on the change in normalized temperature, humidity and wind speed is shown. Based on both figures 2.13 & 2.14 it could be inferred that the optimal conditions for maximum bike count is when the normalized temperature is between 0.4 - 0.8, normalized humidity is below 0.75 and normalized winspeed is below 0.15.

To see how the bike rental count varies based on working day and the weather site information figure 2.15 was generated. Here bike rental count is plotted against normalized temperature and categorical variables 'workingday' and 'weathersit'. From figure 2.15 it can be seen that majority of the bike rental counts were on working day 1 which is neither a weekend nor a holiday. Therefor it can be said that majority of the bike rental counts are from working days Monday to Friday. It can also be noted that the weather site forecast is 2 (Mist + Cloudy, Mist + Broken clouds, Mist + Few clouds, Mist) is maximum between the region of rentals counts 0 - 6000 (y-axis) . However the weathersite forecast 1 (Clear, Few clouds, Partly cloudy) dominates the region of bike counts of 6000-8500. Weather site 3 (Light Snow, Light Rain + Thunderstorm + Scattered clouds, Light Rain + Scattered clouds.) has the least amount of bike rental counts.

Further analysis was done on the raw data and it is shown in **Appendix** A figure(**??**, **??**, **??** & **??**).

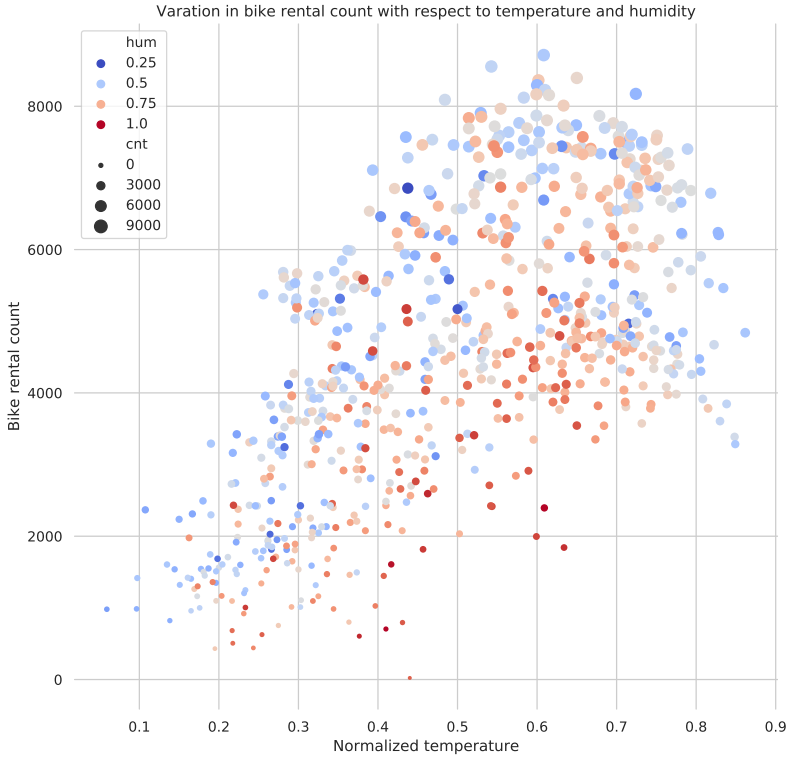


Figure 2.13: Variation in bike count based on normalized temperature and normalized humidity.

2.5 Feature Selection

To develop an efficient model, it is required to know which are the features which are of most importance in predicting the target variable. It is possible that many variable in our data set is less important in predicting the target than others or they my just be redundant. In this case it is required that we remove these variables from our data set to reduce its complexity. At times two features may carry the same information. In such case it is required to remove one of the features to avoid multi collinerity problems. There are many way to perform feature selection or dimensionality reduction, but the method selected is this report is **Correlation Analysis** for continuous features and **ANOVA test** for categorical features.

2.5.1 Correlation Analysis

Correlation Analysis is a technique which helps to determine how strongly two features are related to each other(i.e their co-variance). As the co-variance can vary from - infinity to + infinity, the correlation is used as it is a scaled version of the co-variance having values ranging from -1 to +1. A correlation plot is shown in figure 2.16. A correlation threshold of 0.8 is set and feature pairs of which exceeds this threshold, one of them is dropped. Table 2.1 corresponds to the values of the correlation figure 2.16. By examining table 2.1 it can be observed that features temp and atemp are highly correlated. Hence the feature atemp is dropped from the analysis.

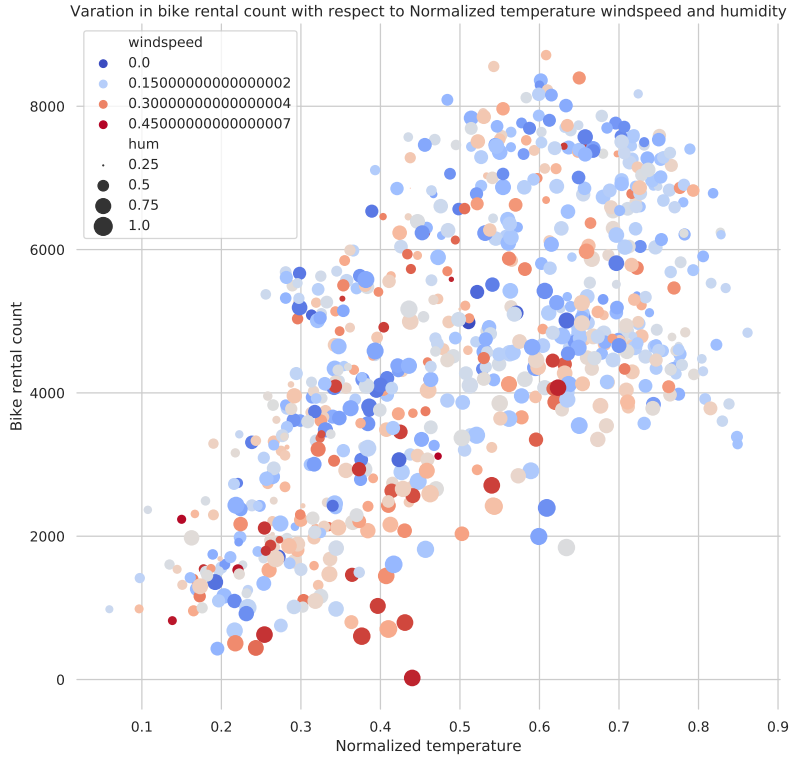


Figure 2.14: Variation in bike count based on normalized temperature humidity and windspeed.

Table 2.1: Correlation analysis table

	temp	atemp	hum	windspeed	cnt
temp	1.0	0.99	0.1	-0.13	0.62
atemp	0.99	1.0	0.11	-0.15	0.63
hum	0.1	0.11	1.0	-0.2	-0.13
winsdpeed	-0.13	-0.15	-0.2	1.0	-0.2
cnt	0.6	0.63	-0.13	-0.2	1.0

2.5.2 Analysis Of Variance (ANOVA)

Analysis of Variance (ANOVA) is a statistical technique used to compute and compare the mean between two or more groups of observations. ANOVA makes use of two variables which are categorical variables and numeric variables of the data set. The python code in listing 2.4 is used to compute the p values of the feature and target variables. listing 2.4 also the output p- values to the threshold value of 0.05 and saves the feature names to be dropped in drop_feat variable. The output of the code in listing 2.4 is shown in figure 2.17.

Listing 2.4: The python code used to comput the p values of the feature and target variables

```
1 ## ANOVA TEST FOR P VALUES
2 import statsmodels.api as sm
```

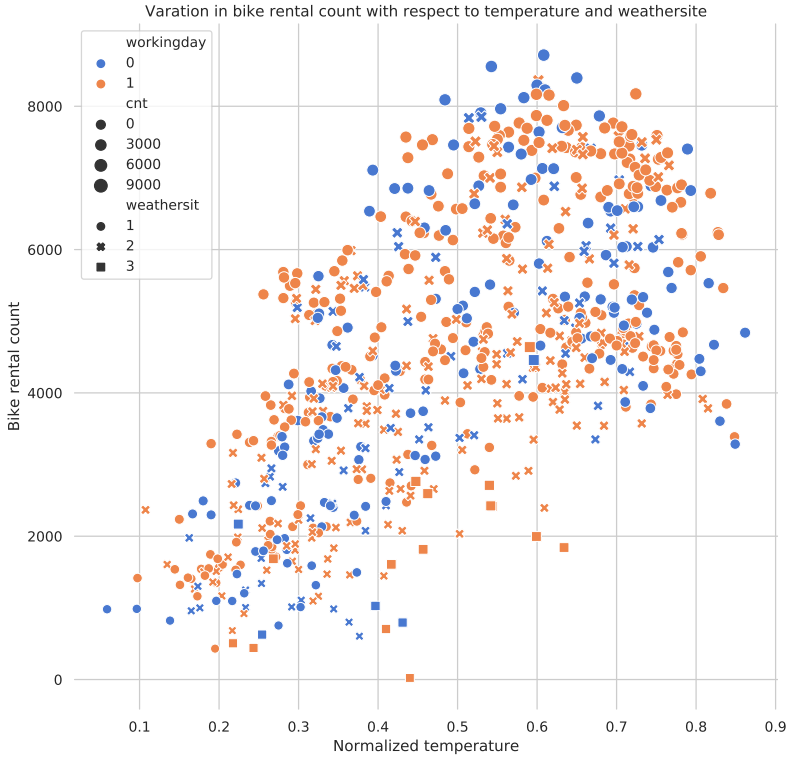


Figure 2.15: Variation in bike count based on normalized temperature, workingdays and weathersite. Workingdays : 1 - working; 0 - holiday | Weathersite: 1- Clear, Few clouds, Partly cloudy; 2- Mist + Cloudy, Mist + Broken clouds, Mist + Few clouds, Mist.; 3- Light Snow, Light Rain + Thunderstorm + Scattered clouds, Light Rain + Scattered clouds.

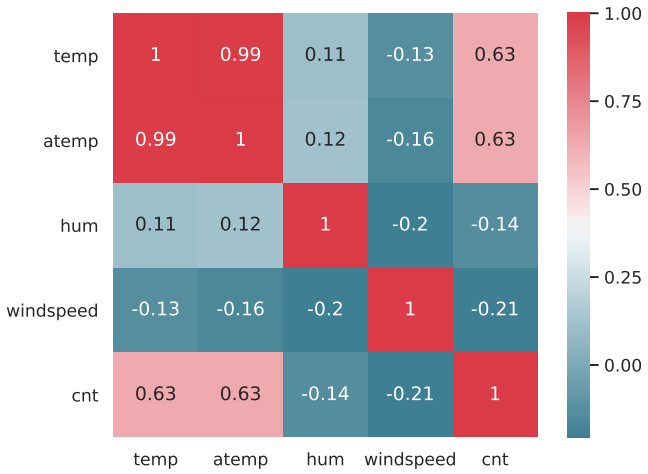


Figure 2.16: Correlation analysis of continuous variables.

```
from statsmodels.formula.api import ols

anova_p = []
for  i in obj_dtype:
    buf = label + ' ~ ' + i
    mod = ols(buf,data=df).fit()
    anova_op = sm.stats.anova_lm(mod, typ=2)
    print(anova_op)
    anova_p.append(anova_op.iloc[0:1,3:4])
    p = anova_op.loc[i,'PR(>F)']
    if p >= 0.05:
        drop_feat.append(i)
```

```
                sum_sq     df          F        PR(>F)
season    9.305377e+08    3.0  127.350479  5.482108e-66
Residual  1.726865e+09  709.0         NaN           NaN
                sum_sq     df          F        PR(>F)
yr        8.887398e+08    1.0  357.27215  7.173240e-65
Residual  1.768663e+09  711.0        NaN           NaN
                sum_sq     df          F        PR(>F)
mnth      1.049040e+09   11.0  41.565535  3.197860e-69
Residual  1.608363e+09  701.0        NaN           NaN
                sum_sq     df         F    PR(>F)
holiday   1.415864e+07    1.0  3.808499  0.051385
Residual  2.643244e+09  711.0       NaN       NaN
                sum_sq     df         F    PR(>F)
weekday   1.970414e+07    6.0  0.878994  0.509776
Residual  2.637698e+09  706.0       NaN       NaN
                sum_sq     df         F    PR(>F)
workingday  7.117269e+06    1.0  1.909371  0.167467
Residual    2.650285e+09  711.0       NaN       NaN
                sum_sq     df          F        PR(>F)
weathersit  2.672556e+08    2.0  39.694529  4.553787e-17
Residual    2.390147e+09  710.0        NaN           NaN
```

Figure 2.17: Analysis Of Variance (ANOVA).

From figure 2.17 we can see that features holiday, weekday and workingday are greater than the p value threshold (0.05) and hence they are dropped along with atemp. Now the number of features and target variables have reduced from 13 and 3 to 7 and 8 respectively.

2.6 Feature Engineering

Now that most of the pre processing is over the last thing left to do is to convert the categorical variables into their respective dummy variables but creating a feature for each category group in the categorical feature. As our categorical variables are in the form of numbers and not strings we need not convert them to numbers, but it is required that we create dummy variables for each categorical group. As the categorical features in our data set are nominal categorical data and not ordinal categories the model shouldn't give higher precedence to a higher number. For example, the categories present in the feature seasons is 1, 2, 3 and 4 which corresponds to spring, summer, fall and winter respectively. As the categories are numeric, the model would give a higher precedence to 4 as it is numerically greater than the rest. Which is not

so as they are not ordinal categories and each number refers to a specific season. Hence each group is converted to a binary category. Where 1 imply the presence of that group and 0 represents absence. To make it less complex the first dummy variable is dropped, as when all the other groups are absent it imply that the dropped group is present. This is achieved by running the command in listing 2.5.

Listing 2.5: Creation of dummy variables.

```
1
2 ## Creating dummy variables
3 # Where x is all the features in our analysis.
4
5 X_lr = pd.get_dummies(X, columns = obj_dtype, drop_first = True)
```

Chapter 3

Modeling

3.1 Model Selection

It has be noted in previous stages of our analysis that for different combinations of the independent variables, the count is different. It can be seen that the dependent variable is a continuous variable and hence the type model of model that would be developed for this problem is a regression model.

3.2 Methodology:

Model Evaluation is an integral part of the model development process as it helps us find the best model for representing our data. It also helps to evaluate as to how it would on new data. In order to develop an efficient and accurate model to predict our target variable we shall use a combination of three different methods The three different methods used in our analysis is given below.

- Hold-Out Method
- Hyper parameter Tunning
- Cross-Validation Technique

3.2.1 Hold-Out Method

As evaluating model performance on training data set may lead to develop an over fitted model. Due to this is is required to test the model on a separate data set. Hence the original data set is split into training and testing data. The training data set is used to build a predictive model and the testing data is used to evaluate the model performance.

3.2.2 Hyper parameter Tunning

Hyper parameter Tuning is a method in which the parameters of the model is to be set before training. Scikit-Learn implements a set of sensible default hyper parameters but it may not be optimal. In our analysis we shall use two types of hyper parameter tuning methodology which are *Random Search CV* and *Grid Search CV*. The major difference between them is the run time. As randomized search is drastically lower than grid search. Random search is faster than grid search as we do not provide a set of discrete

values to be searched. Rather we provide a statistical distribution for each parameter from which values may be randomly sampled. Where as in grid search a discrete set of values are provided for each parameter and several models are built on each of its combinations which makes it very laboursome.

3.2.3 K-Fold Cross Validation Technique

In K fold cross validation technique the data is divided into k subsets of equal size. We build the model K times, each time leaving out a subset from training. This sub set which was left out will be used for testing. This method helps us develop an unbiased estimate of the model performance.

3.3 Approach

At first thee data is split to training and testing data set as per the hold out method. Then Hyperparameter tuning and k fold cross validation are performed together. First Random Search CV is performed on a wide range of parameter values. This is performed first as its a hyperparameter tuning cross validation technique in which the run time required is less. Based on its results we can narrow are search parameters and perform grid search to come up with an optimal model.

3.4 Model Building

We Start building our model by using the simplest model to the most complex model. Therefor we start our model building using the multiple linear regression model.

3.4.1 Multivariate Linear Regression Model

Before the model is built in multivariate linear regression it is required to do some processing on the data before hand. The equation around which the multivariate regression in statsmodels works is shown below.

$$y = \beta_0 + \beta_1 * X_1 + \beta_2 * X_2 + ... + \beta_n * X_n \tag{3.1}$$

Where β_1, β_2 till β_n are the coefficients for columns X_1, X_2 till X_n respectively and β_0 represents the intercept. Unfortunately in statsmodel the β_0 column is not added by default to out model and hence we are required to add a column with a constant value of one to the data set. Now that this column is added and the dummy variables are created we can go ahead with building the model. Listing **??** shows the Multivariate Linear Regression Modeling. Figure 3.1 shows the model summary.

Listing 3.1: Multivarient Linear Regression Modeling.

```
1
2 ## Creating dummy variables for multivarient linear regression
3 X_l = pd.get_dummies(df, columns = obj_dtype, drop_first = True)
4 X_l["b0"]=1
5 col_name = X_l.columns.tolist()
6 col_name = col_name[21:22] + col_name[0:3]+ col_name[4:21]+ col_name[3:4]
7 X_l = X_l.loc[:, col_name]
8
```

```
#Divide data into train and test
train_stat, test_stat = train_test_split(X_l, test_size=0.2,random_state =↩
    0)

# ## Linear Regression
#Import libraries for LR
import statsmodels.formula.api as sm
from sklearn.metrics import r2_score
from sklearn.metrics import mean_squared_error
# Train the model using the training sets
model = sm.OLS(train_stat.iloc[:,21], train_stat.iloc[:,0:21]).fit()
# Print out the statistics
model.summary()
```

OLS Regression Results

Dep. Variable:	cnt	**R-squared:**	0.834
Model:	OLS	**Adj. R-squared:**	0.827
Method:	Least Squares	**F-statistic:**	137.5
Date:	Wed, 02 Jan 2019	**Prob (F-statistic):**	1.00e-198
Time:	16:54:02	**Log-Likelihood:**	-4599.1
No. Observations:	570	**AIC:**	9240.
Df Residuals:	549	**BIC:**	9331.
Df Model:	20		
Covariance Type:	nonrobust		

Figure 3.1: Model summary for multivariate linear regression.

It can be seen from figure 3.1 that the **Adjusted R-squared is 0.83** which is pretty impressive, as **it means 83% of the variance of the data is explained by the model.** Looking at the F-statistic and combined p-value we can reject the null hypothesis that target variable does not depend on any of the predictor variables. The code to develop the model and its output can be seen in **Appendix** A figure A.20.

3.4.2 Decision Tree Regressor

Now a decision tree regressor model is built on to our data set. At first a model is built on the default parameters of the Decision tree regressor. The default values for the parameters controlling the size of the trees such as max_depth, min_samples_leaf, etc lead to fully grown and unpruned trees which could

require large memory consumption and increase the model complexity. It can also cause overfitting of the model. To reduce the complexity and memory consumption the size of the trees should be controlled by setting those parameter values. In our analysis we are going to tune the max_depth parameter which controls the maximum depth of the tree. The default model without any tuning is taken as the base line model.

After this a grid of intervals of 5 is taken from 5 - 50 for max_depth are given to random search cv. The k fold cross validation parameter given is K is 5 and n_iter is 5, which the cross validation is to be repeated 5 times. The optimal depth from random search cv is 10. Now that we narrowed down our search criteria, grid search cv can be conducted on range 5 - 20 max_depth with an interval of 2. Here it was found that the best parameters for max_depth is 5. The code to develop the models and their output can be seen in **Appendix** A figure A.21, A.22 & A.23.

3.4.3 Random Forest Regressor

Random forest is an ensemble that consists of many decision trees At first a model is built on the default parameters of the Random Forest regressor. To reduce the complexity and memory consumption the max_depth and n_estimators parameters is tuned n_estimators tell the amount of trees to be used in the model. The default model without any tuning is taken as the base line model.

After this a grid of intervals of 2 is taken from 1 - 20 max_depth and a grid of intervals of 2 is taken from 1 - 100 for n_estimators given to random search cv. The k fold cross validation parameter given is K is 5 and n_iter is 5, which the cross validation is to be repeated 5 times. The optimal depth from random search cv is 18 and n_estimators is 15. Now that we narrowed down our search criteria, grid search cv can be conducted on 15 - 25 for max_depth and intervals of 2 is taken from 11 - 17 for n_estimators. Here it was found that the best parameters for max_depth is 12 and n_estimators is 16. The code to develop the models and their output can be seen in **Appendix** A figure A.24, A.25 & A.26.

3.4.4 Gradient Boosting

Gradient boosting is an ensamble boosting method in which a colection of regressors are built in series.It is a method of converting asequence of weak learners to a complex model. Each regressors prediction is based on th previous regressors by adding weights accordingly. We shall now implement the Gradint boosting model on our data. At first a model is built on the default parameters of the Random Forest regressor. To reduce the complexity and memory consumption the max_depth and n_estimators parameters is tuned n_estimators tell the amount of trees to be used in the model. The default model without any tuning is taken as the base line model.

After this a grid of intervals of 2 is taken from 1 - 10 max_depth and a grid of intervals of 10 is taken from 50 - 150 for n_estimators given to random search cv. The k fold cross validation parameter given is K is 5 and n_iter is 5, which the cross validation is to be repeated 5 times. The optimal depth from random search cv is 23 and n_estimators is 15. Now that we narrowed down our search criteria, grid search cv can be conducted on range 1 - 5 for max_depth and intervals of 10 is taken from 120 - 140 for n_estimators. Here it was found that the best parameters for max_depth is 3 and n_estimators is 75. The code to develop the models and their output can be seen in **Appendix** A figure A.27, A.28 & A.29.

3.4.5 Extreme Gradient Boosting

The efficiency and accuracy of the gradient boosting model can further be improved on, by running the xgboost model on the data. Extreme Gradient Boosting, which is an efficient implementation of the gradient boosting frame work. The package xgboost is used to build the xgboost model on the data. The code to develop the model and its output can be seen in **Appendix** A figure A.30.

Chapter 4

Conclusion

4.1 Model Evaluation

Now that a few models have been created on our data set, it is required that a suitable evaluation matrix is selected it compare the different model. As our problem statement is a regression problem. There are a few popular evaluation matrix which are commonly used. These matrix are : MAPE - Mean absolute percentage error, is the measure accuracy as a percentage of error, MSE - Mean squared error, it is the square root of the mean squared errors and RMSE - Root mean squared error, it is the square root of the mean squared errors.

RMSE is mostly used in time series analysis and hence it is not used. There is a choice to choose between MAE and MAPE. As MAPE delivers the values in for of error percentage, it is selected as it is easy to understand. MSE is also selected as it gives us the goodness of fit of the model. The smaller the MSE the better the fit. THe respected code to genereate the models and their respected MAPE and MSE can be seen in **Appendix** A figure(A.20, A.21, A.22, A.23, A.24, A.25, A.26, A.27, A.28, A.29 & A.30).

4.2 Model Selection

From table 4.1 it can be seen that Grid Search CV of Gradient Boost Machine, having the parameters for max_depth- 3 and n_estimator - 100, has the least MAPE of 18.06%. It has a R-Squared value of 0.88 which means it explains 88% of the variance, which is very good. The second best model is the XGBoosting model which has a MAPE of 18.25% and it also has a Rsquared value of 0.88 explaining 88% of the variance. **We see that both the models perform comparitively well while comparing their MSE, R-Squared value and MAPE. Hence either Grid Search CV of Gradient Boost Machine with max_depth- 3 and n_estimator - 100 or XGBoosting model can be selected as the preferred model.**

	Model_name	MSE	MAPE	R^2
0	Multivarient linear regression	5.715082e+05	18.437683	0.866874
1	Decision tree default	1.079306e+06	26.748537	0.748588
2	Decision tree Random Search CV	1.040601e+06	26.572396	0.757604
3	Decision tree Grid Search CV	9.135586e+05	25.073871	0.787197
4	Random Forest Default	6.151233e+05	20.272507	0.856714
5	Random Forest Random Search CV	5.962079e+05	20.508065	0.861120
6	Random Forest Grid Search CV	6.082217e+05	20.578772	0.858322
7	Gradient Boosting Default	5.141599e+05	18.506449	0.880232
8	Gradient Boosting Random Search CV	5.223807e+05	18.406009	0.878317
9	Gradient Boosting Grid Search CV	5.143945e+05	18.063431	0.880178
10	XGBOOST	4.991374e+05	18.254574	0.883732

Figure 4.1: Saving results to a dataframe for easy comprehension.

Appendices

Appendix A

Python code and its corresponding output:

Functions:

In [370]:

```python
#Function to segregate the different data types present in the dataframe.
def dtype_separator(df, col_names):
    '''
    ###########################################################################################
    This function will segregate the different data types present in the dataframe.
    ###########################################################################################
    Input -
    * df - The data frame to be analysed.
    *A list of the column names.
    Output -
    * obj - columns containing object data type.
    * num - columns containing numerical data type (int64/float64).
    * bool_d - columns containing bool data type.
    * unknown = columns containing data type such as datetime64, timedelta[ns] and others.
    ###########################################################################################
    '''
    obj = []
    num = []
    bool_d = []
    unknown = []
    for i in range(len(col_names)):
        if df.iloc[:,i].dtype == 'O':
            obj.append(col_names[i])
        elif df.iloc[:,i].dtype == 'int64' or df.iloc[:,i].dtype == 'float64':
            num.append(col_names[i])
        elif df.iloc[:,i].dtype == 'bool':
            bool_d.append(col_names[i])
        else:
            unknown.append(col_names[i])
    return obj, num, bool_d, unknown
```

Figure A.1: This function will segregate the different data types present in the dataframe.

In [371]:
```
#Function to view the categories present in each categorical feature
def view_feature_cat(obj):
    '''
    ##############################################################################################
    This function will display the categories present in each categorical variable (i.e. Feature).
    ##############################################################################################
    Input -
    *obj - A list of the caegorical column names (i.e. caegorical features).
    Output -
    *Displays on the screen, the different categorical features and the catogeries present in them.
    It also displays the count of each category in the feature.

    This helps us have an easy over view of the categories present in the features and infer on there
    significance
    ##############################################################################################
    '''
    for i in range(len(obj)):
        print('*****************************************')
        print('Feature:',obj[i])
        print('-----------------------')
        print(df[str(obj[i])].value_counts())
        print('*****************************************')
```

In [372]:
```
def outlier_percent(df):
    '''
    ##############################################################################################
    This function will print out the total number of outliers present in data and percentage of
    outliers present in data set.
    ##############################################################################################
    Input -
    * df - The dataframe is take as input.
    Output -
    * print out the total number of outliers and percentage of outliers present in data set
    ##############################################################################################
    '''
    l_q = df.quantile(0.25)
    u_q = df.quantile(0.75)
    iqr = u_q - l_q
    total_outliers = ((df<(l_q - 1.5*iqr))|(df>(u_q + 1.5*iqr))).sum()
    outliers = total_outliers.sum()
    print('Total number of outliers present =',outliers)
    r,c = df.shape
    outlier_percentage = (outliers/r)*100
    print("Outlier percentage :",round(outlier_percentage,3))
```

Figure A.2: First function will display the categories present in each categorical variable (i.e. Feature). The second function will print out the total number of outliers present in data and percentage of outliers present in data set.

```
#Function to perform outlier analysis
def outlier_analysis(df, cnames, method):
    '''
    ##########################################################################################################
    This function will perform outlier analysis on the given data. This function deals with outliers in
    two ways, which are as follows:
    - Delete outliers from data.
    - Detect and replace with NA and then perform KNN KNN Imputation for missing value on it.
    ##########################################################################################################
    Input -
    * df - The dataframe is take as input.
    * cnames - List of column names.
    * method - There are two options (0 or 1), which performs the following operation:
        - Detect and delete outliers from data.
        - Detect and replace with NA and then perform KNN KNN Imputation for missing value on it.
    Output -
    * df - It returns a dataframe after performing outlier analysis.
    ##########################################################################################################
    '''
    if method == 0:
        #Detect and delete outliers from data
        for i in cnames:
            q75, q25 = np.percentile(df.loc[:,i], [75 ,25])
            iqr = q75 - q25

            low_qutr = q25 - (iqr*1.5)
            up_qutr = q75 + (iqr*1.5)

            df = df.drop(df[df.loc[:,i] < low_qutr].index)
            df = df.drop(df[df.loc[:,i] > up_qutr].index)
        return df
    elif method == 1:
        for i in cnames:
            #Detect and replace with NA
            # #Extract quartiles
            q75, q25 = np.percentile(df[i], [75 ,25])

            #Calculate IQR
            iqr = q75 - q25

            #Calculate inner and outer fence
            low_qutr = q25 - (iqr*1.5)
            up_qutr = q75 + (iqr*1.5)

            #Replace with NA
            df.loc[df[i] < low_qutr,:i] = np.nan
            df.loc[df[i] > up_qutr,:i] = np.nan
            #Impute with KNN
            df = pd.DataFrame(KNN(k = 3).fit_transform(df[i]), columns = df.columns)
        return df
    else:
        print('Error: incorrect input')
```

Figure A.3: This function will perform outlier analysis on the given data. This function deals with outliers in two ways, which are as follows: Delete outliers from data or Detect and replace with NA and then perform KNN KNN Imputation for missing value on it.

```
In [374]: def drop_list(columns, rows, treshold):
    '''
    ##########################################################################################################
    This function will select the features whose corelation excides the treshold
    ##########################################################################################################
    Input -
    * columns - Corelation matrix column index.
    * row - Row matrix column index.
    * treshold - * columns - Corelation treshold.
    Output -
    * drop - List of features to be dropped
    ##########################################################################################################
    '''

    drop = []
    drop_val = []
    for i in list(range(0,len(columns))):
        for j in list(range(0, len(rows))):
            if columns[i] != rows[j]:
                if abs(corr.loc[rows[j],columns[i]]) >= abs(treshold):
                    drop.append(rows[j]) if rows[j] and columns[i] not in drop else print['repeat')
                    drop_val.append(corr.loc[rows[j],columns[i]])
    return drop
```

Figure A.4: This function will select the features whose correlation exceeds the threshold.

```
In [375]:
def drop_features(drop_feat, obj_dtype, num_dtype):
    '''
    ##############################################################################################
    This function will remove the dropped features from the variables containing the object feature
    names and numeric feature names, obj_dtype and num_dtype respectively.
    ##############################################################################################
    Input -
    * drop_feat - List containing the names of the dropped features.
    * obj_dtypes - List of all the categorical feature names.
    * num_dtypes - List of all the numeric feature names.
    Output -
    * obj_d - List containing the categorical feature names excluding the dropped features.
    * num_d - List containing the numeric feature names excluding the dropped features.
    ##############################################################################################
    '''
    obj_del = []
    num_del = []
    for i in range(0, len(drop_feat)):
        #print(i)
        if(drop_feat[i] in obj_dtype):
            #print('obj in',i)
            ## Dropping deleted object variable name from obj_dtype variable
            obj_del.append(drop_feat[i])

        elif(drop_feat[i] in num_dtype):
            #print('num in',i)
            ## Dropping deleted numeric variable name from num_dtype variable
            num_del.append(drop_feat[i])

        else:
            print('Data type', i ,'is not an object or numeric.')
    obj_d= list(set(obj_dtype).difference(set(obj_del)))
    num_d= list(set(num_dtype).difference(set(num_del)))
    return obj_d, num_d
```

Figure A.5: This function will remove the dropped features from the variables containing the object feature names and numeric feature names, obj_dtype and num_dtype respectively.

```
In [448]:
#Function to compute MAPE
def MAPE(y_true, y_pred):
    '''
    ##############################################################################################
    This function will Mean Absolute Percent Error (MAPE).
    ##############################################################################################
    Input -
    * y_true - Actual values from y_test.
    * y_pred - Predicted values.
    Output -
    * mape - Mean Absolute Percent Error as error percentage.
    ##############################################################################################
    '''
    mape = np.mean(np.abs((y_true - y_pred) / y_true))*100
    return mape
```

Figure A.6: This function will Mean Absolute Percent Error (MAPE).

Exploratory Data Analysis:

```
In [377]:
#Load data
df = pd.read_csv("day.csv")
```

```
In [378]:
df.head()
```

Out[378]:

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

```
In [379]:
df = df.drop(['instant','dteday','casual','registered'],axis=1)
```

Figure A.7: Loading data and dropping irrelevant features.

```
In [381]: #Calling function 'dtype_separator' to segregate different data types
          obj_dtype, num_dtype, bool_dtype, unknown_dtype = dtype_separator(df, df.columns)

In [382]: df.dtypes

Out[382]: season          int64
          yr              int64
          mnth            int64
          holiday         int64
          weekday         int64
          workingday      int64
          weathersit      int64
          temp          float64
          atemp         float64
          hum           float64
          windspeed     float64
          cnt             int64
          dtype: object

In [383]: ## Note:- while seperating data types it was noticed that several of the categorical variable are stored as int.
          ## Hence these variables are to be converted to object data type.

          ## Create a list of variables which are incorrectly classified as numeric
          convert_obj = ['season', 'yr', 'mnth', 'holiday', 'weekday', 'workingday', 'weathersit']

In [384]: for i in convert_obj:
              df.loc[:,i] = df.loc[:,i].astype("object")

          df.dtypes

Out[384]: season         object
          yr             object
          mnth           object
          holiday        object
          weekday        object
          workingday     object
          weathersit     object
          temp          float64
          atemp         float64
          hum           float64
          windspeed     float64
          cnt             int64
          dtype: object
```

Figure A.8: Separating the features based on data type using function in figure A.1 and converting the features which are incorrectly stored.

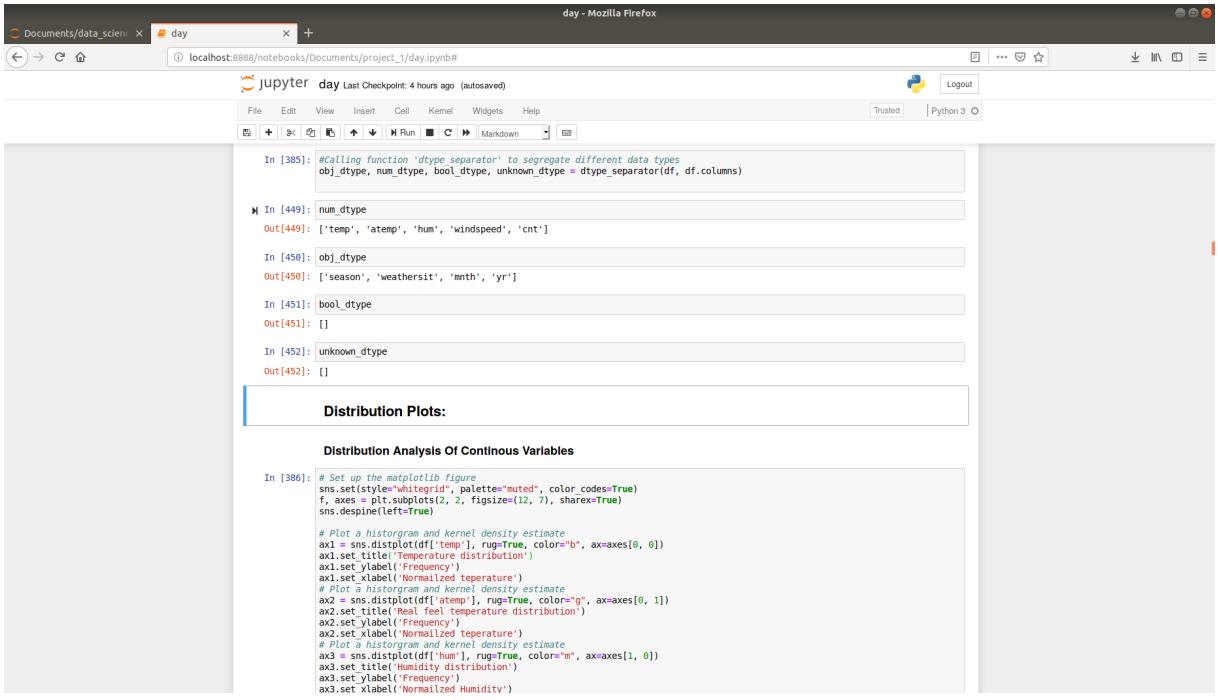


Figure A.9: Output of function in figure A.1.

Distribution Plots:

Distribution Analysis Of Continous Variables

In [306]:
```
# Set up the matplotlib figure
sns.set(style="whitegrid", palette="muted", color_codes=True)
f, axes = plt.subplots(2, 2, figsize=(12, 7), sharex=True)
sns.despine(left=True)

# Plot a historgram and kernel density estimate
ax1 = sns.distplot(df['temp'], rug=True, color="b", ax=axes[0, 0])
ax1.set_title('Temperature distribution')
ax1.set_ylabel('Frequency')
ax1.set_xlabel('Normailzed teperature')
# Plot a historgram and kernel density estimate
ax2 = sns.distplot(df['atemp'], rug=True, color="g", ax=axes[0, 1])
ax2.set_title('Real feel temperature distribution')
ax2.set_ylabel('Frequency')
ax2.set_xlabel('Normailzed teperature')
# Plot a historgram and kernel density estimate
ax3 = sns.distplot(df['hum'], rug=True, color='m', ax=axes[1, 0])
ax3.set_title('Humidity distribution')
ax3.set_ylabel('Frequency')
ax3.set_xlabel('Normailzed Humidity')
# Plot a historgram and kernel density estimate
ax4 = sns.distplot(df['windspeed'], rug=True, color="r", ax=axes[1, 1])
ax4.set_title('Windspeed distribution')
ax4.set_ylabel('Frequency')
ax4.set_xlabel('Normailzed Windspeed')
plt.tight_layout()
plt.savefig('distribution_plot.pdf')
```

```
/home/gid/anaconda3/lib/python3.6/site-packages/scipy/stats/stats.py:1713: FutureWarning: Using a non-tuple sequence fo
r multidimensional indexing is deprecated; use `arr[tuple(seq)]` instead of `arr[seq]`. In the future this will be inte
rpreted as an array index, `arr[np.array(seq)]`, which will result either in an error or a different result.
  return np.add.reduce(sorted[indexer] * weights, axis=axis) / sumval
```

Figure A.10: Code to plot distribution of continuous variables.

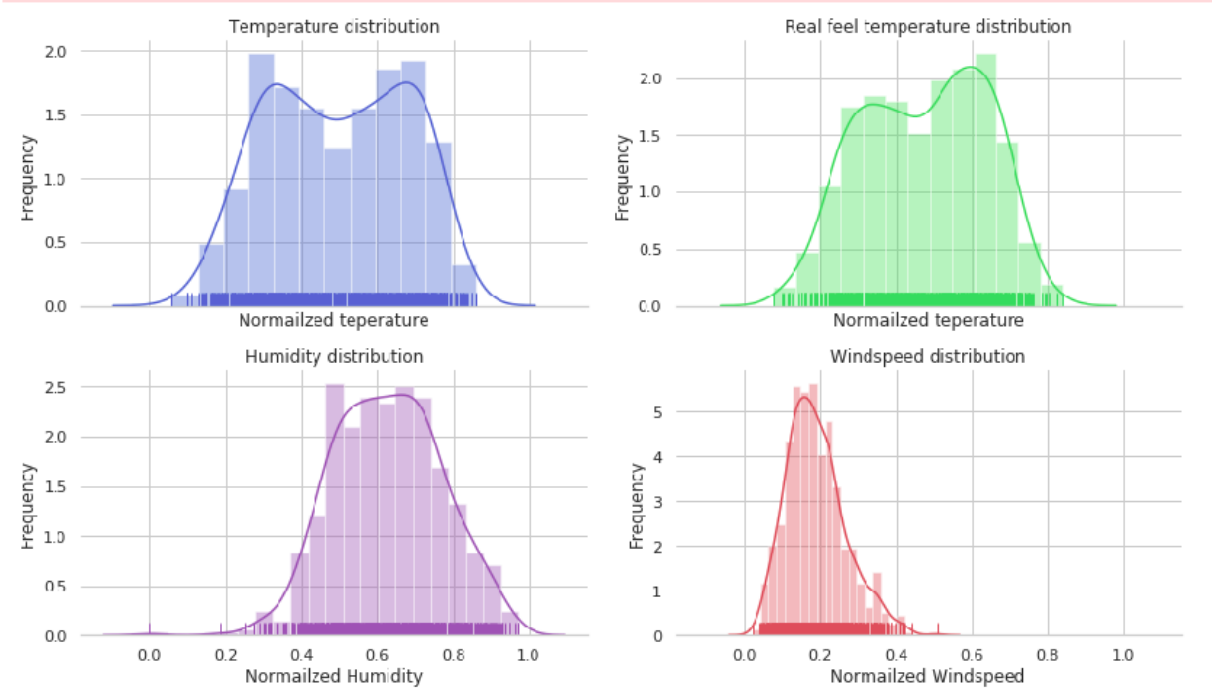


Figure A.11: Distribution plot of continuous variables for code in figure A.10

Check the skewness of the continous variables

In [307]:
```
print("**********************************************************************")
for i in num_ctype:
    from scipy import stats
    skewness = stats.describe(df.loc[:,i])
    print("statistical properities of {0:5s}:".format(i))
    print(skewness)
    print("**********************************************************************")
```

```
**********************************************************************
statistical properities of temp :
DescribeResult(nobs=731, minmax=(0.0591304, 0.861667), mean=0.495384788508892, variance=0.03350766717740828, skewness=-
0.05440902400571610, kurtosis=-1.1194225400473057)
**********************************************************************
statistical properities of atemp:
DescribeResult(nobs=731, minmax=(0.0790696, 0.8408959999999999), mean=0.47435398864569084, variance=0.026556345661055517
4, skewness=-0.1308188980737412, kurtosis=-0.9866019052943136)
**********************************************************************
statistical properities of hum  :
DescribeResult(nobs=731, minmax=(0.0, 0.9725), mean=0.6278940629274967, variance=0.02028604714193028, skewness=-0.06964
015783152368, kurtosis=-0.07228631791987006)
**********************************************************************
statistical properities of windspeed:
DescribeResult(nobs=731, minmax=(0.0223917, 0.507463), mean=0.190486211627907, variance=0.006005919960192755, skewness=
0.6759547264275362, kurtosis=0.39992023832685497)
**********************************************************************
statistical properities of cnt  :
DescribeResult(nobs=731, minmax=(22, 8714), mean=4504.3488372093025, variance=3752788.2082828926, skewness=-0.047255557
55362063, kurtosis=-0.8145762269613592)
**********************************************************************
```

Figure A.12: Skewness analysis of distribution plots in fig A.11

Missing Values Analysis ¶

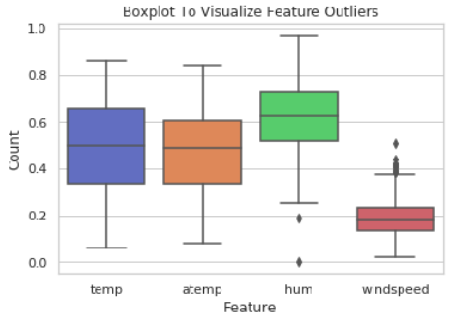
In [389]:
```
## Missing values analysis
pd.DataFrame(df.isnull().sum())
```

Out[389]:

	0
season	0
yr	0
mnth	0
holiday	0
weekday	0
workingday	0
weathersit	0
temp	0
atemp	0
hum	0
windspeed	0
cnt	0

Figure A.13: Missing value analysis, it is noted that there is no values missing.

```
# #Plot boxplot to visualize Feature variable Outliers
ax_o1 = sns.boxplot(data= df.iloc[:,7:11])
ax_o1.set_title('Boxplot To Visualize Feature Outliers')
ax_o1.set_ylabel('Count')
ax_o1.set_xlabel('Feature')
plt.savefig('feature_outlier1.pdf')
```




```
91]: # #Plot boxplot to visualize Target variable Outliers
ax_o2 = sns.boxplot(data= df.iloc[:,11:12])
ax_o2.set_title('Boxplot To Visualize Label Outliers')
ax_o2.set_ylabel('Count')
ax_o2.set_xlabel('Label')
plt.savefig('label_outlier1.pdf')
```

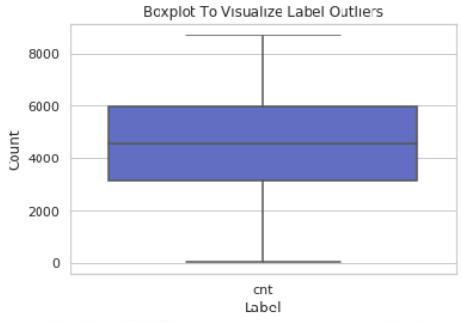


Figure A.14: Boxplot to visualize the outliers for feature and target variables.

```
In [392]: #Calling the outlier function to perform outlier percent
outlier_percent(df)
```

```
Total number of outliers present = 15
Outlier percentage : 2.052
```

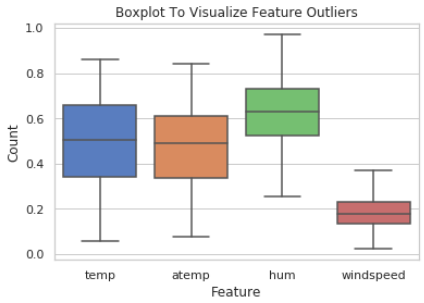
```
In [393]: #Calling the outlier function to perform outlier analysis
method = 0
df = outlier_analysis(df, num_dtype, method)
df.describe()
```

Out[393]:

	temp	atemp	hum	windspeed	cnt
count	717.000000	717.000000	717.000000	717.000000	717.000000
mean	0.497365	0.476252	0.631562	0.186287	4532.843794
std	0.183617	0.163155	0.139222	0.071786	1933.542429
min	0.059130	0.079070	0.254167	0.022392	22.000000
25%	0.337500	0.337939	0.524583	0.134329	3214.000000
50%	0.505833	0.491783	0.630833	0.178496	4570.000000
75%	0.656667	0.611121	0.732917	0.230721	6031.000000
max	0.861667	0.840896	0.972500	0.378108	8714.000000

Figure A.15: Checking the percentage of outliers from function in figure A.2 and removal of outliers using function in fig A.3

In [402]:
```
# #Plot boxplot to visualize Feature variable Outliers
ax_o1 = sns.boxplot(data= df.iloc[:,7:11])
ax_o1.set_title('Boxplot To Visualize Feature Outliers')
ax_o1.set_ylabel('Count')
ax_o1.set_xlabel('Feature')
plt.savefig('feature_outlier4.pdf')
```



In [403]:
```
# #Plot boxplot to visualize Target variable Outliers
ax_o2 = sns.boxplot(data= df.iloc[:,11:12])
ax_o2.set_title('Boxplot To Visualize Label Outliers')
ax_o2.set_ylabel('Count')
ax_o2.set_xlabel('Label')
plt.savefig('label_outlier4.pdf')
```

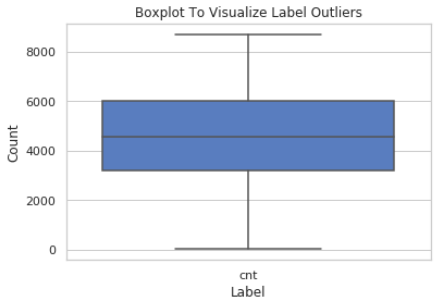


Figure A.16: Boxplot to visualize the outliers for feature and target variables, after outlier analysis.

Correlation Analysis

In [409]:
```
################################################################################
## Correlation plot
#Correlation plot
df_corr = df.loc[:,num_dtype]
#Set the width and hieght of the plot
f, ax = plt.subplots(figsize=(7, 5))
#Generate correlation matrix
corr = df_corr.corr()
#Plot using seaborn library
sns.heatmap(corr, mask=np.zeros_like(corr, dtype=np.bool), cmap=sns.diverging_palette(220, 10, as_cmap=True),
            square=True, annot=True,ax=ax)
plt.savefig('corr_plot.pdf')
```

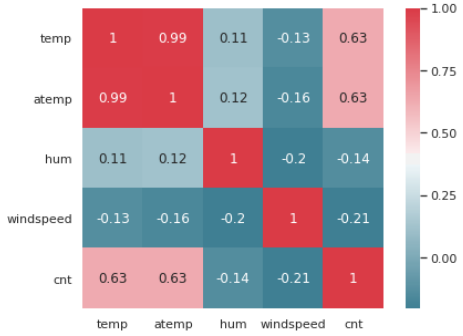


Figure A.17: Correlation plot

ANOVA Test

Less significient variables are added to the drop_feat variable to be removed from the dataframe later.

In [412]:
```
label = 'cnt'
## ANOVA TEST FOR P VALUES
import statsmodels.api as sm
from statsmodels.formula.api import ols

anova_p = []
for  i in obj_dtype:
    buf = label + ' ~ ' + i
    mod = ols(buf,data=df).fit()
    anova_op = sm.stats.anova_lm(mod, typ=2)
    print(anova_op)
    anova_p.append(anova_op.iloc[0:1,3:4])
    p = anova_op.loc[i,'PR(>F)']
    if p >= 0.05:
        drop_feat.append(i)
```

```
                sum_sq     df           F        PR(>F)
season    9.305377e+08    3.0  127.350479  5.482108e-66
Residual  1.726865e+09  709.0         NaN           NaN
                sum_sq     df          F        PR(>F)
yr        8.887398e+08    1.0  357.27215  7.173240e-65
Residual  1.768663e+09  711.0        NaN           NaN
                sum_sq     df          F        PR(>F)
mnth      1.049040e+09   11.0  41.565535  3.197860e-69
Residual  1.608363e+09  701.0        NaN           NaN
                sum_sq     df         F    PR(>F)
holiday   1.415864e+07    1.0  3.808499  0.051385
Residual  2.643244e+09  711.0       NaN       NaN
                sum_sq     df         F    PR(>F)
weekday   1.970414e+07    6.0  0.878994  0.509776
Residual  2.637698e+09  706.0       NaN       NaN
                sum_sq     df         F    PR(>F)
workingday  7.117269e+06    1.0  1.909371  0.167467
Residual    2.650285e+09  711.0       NaN       NaN
                sum_sq     df          F        PR(>F)
weathersit  2.672556e+08    2.0  39.694529  4.553787e-17
Residual    2.390147e+09  710.0        NaN           NaN
```

Figure A.18: ANOVA Test

Multivariant Linear regression

[424]:
```
## Creating dummy variables for multivarient linear regression
X_l = pd.get_dummies(df, columns = obj_dtype, drop_first = True)
X_l["b0"]=1
col_name = X_l.columns.tolist()
col_name = col_name[21:22] + col_name[0:3]+ col_name[4:21]+ col_name[3:4]
X_l = X_l.loc[:, col_name]

#Divide data into train and test
train_stat, test_stat = train_test_split(X_l, test_size=0.2,random_state = 0)

# ## Linear Regression
#Import libraries for LR
import statsmodels.formula.api as sm
from sklearn.metrics import r2_score
from sklearn.metrics import mean_squared_error
# Train the model using the training sets
model = sm.OLS(train_stat.iloc[:,21], train_stat.iloc[:,0:21]).fit()
```

Figure A.19: Multivariate linear regression modeling.

In [148]:
```
# make the predictions by the model
predictions_LR = model.predict(test_stat.iloc[:,0:21])

r2 = r2_score(test_stat.iloc[:,21], predictions_LR)
mse = mean_squared_error(test_stat.iloc[:,21], predictions_LR)

#Calculate MAPE
mape = MAPE(test_stat.iloc[:,21], predictions_LR)

print('Linear Regression Model Performance:')
print('R-squared = {:0.2}.'.format(r2))
print('MSE = ',round(mse))
print('MAPE = {:0.4}%.'.format(mape))
```

```
Linear Regression Model Performance:
R-squared = 0.87.
MSE =  571508.0
MAPE = 18.44%.
```

Figure A.20: Prediction and MAPE for Multivariate linear regression.

```
Decision Regressor Model Performance:
Default Parameters =  {'criterion': 'mse', 'max_depth': None, 'max_features': None, 'max_leaf_nodes': None, 'min_impuri
ty_decrease': 0.0, 'min_impurity_split': None, 'min_samples_leaf': 1, 'min_samples_split': 2, 'min_weight_fraction_leaf
': 0.0, 'presort': False, 'random_state': 0, 'splitter': 'best'}
R-squared = 0.75.
MSE =  1082039.0
MAPE = 26.7%.
****************************************
Random Search CV Decision Regressor Model Performance:
Best Parameters =  {'max_depth': 10}
R-squared = 0.76.
MSE =  1013272.0
MAPE = 25.95%.
****************************************
Grid Search CV Decision Regressor Model Performance:
Best Parameters =  {'max_depth': 5}
R-squared = 0.77.
MSE =  986838.0
MAPE = 25.07%.
****************************************
```

Figure A.21: Decision tree Modeling

Random Search CV - Decision Tree

n [151]:
```
##Random Search CV
from sklearn.model_selection import RandomizedSearchCV
np.random.seed(0)
RDT = DecisionTreeRegressor(random_state = 0)
depth = list(range(5,50,5))
# Create the random grid
randDT_grid = {'max_depth': depth}

randomcv_DT = RandomizedSearchCV(RDT, param_distributions = randDT_grid, n_iter = 5, cv = 5, random_state=0)
randomcv_DT = randomcv_DT.fit(Xlr_train,y_train)

predictions_RDT = randomcv_DT.predict(Xlr_test)
predictions_RDT = np.array(predictions_RDT)

view_best_params_RDT = randomcv_DT.best_params_

best_model = randomcv_DT.best_estimator_

predictions_RDT = best_model.predict(Xlr_test)

#R^2
RDT_r2 = r2_score(y_test, predictions_RDT)
RDT_mse = mean_squared_error(y_test, predictions_RDT)

#Calculate MAPE
RDT_mape = MAPE(y_test, predictions_RDT)

print('Random Search CV Decision Regressor Model Performance:')
print('Best Parameters = ',view_best_params_RDT)
print('R-squared = {:0.2}.'.format(RDT_r2))
print('MSE = ',round(RDT_mse))
print('MAPE = {:0.4}%.'.format(RDT_mape))
print('****************************************')
```

```
Random Search CV Decision Regressor Model Performance:
Best Parameters =  {'max_depth': 10}
R-squared = 0.76.
MSE =  1040601.0
MAPE = 26.57%.
****************************************
```

Grid Search CV - Decision Tree

Figure A.22: Random Search CV Decision Tree Modeling.

Grid Search CV - Decision Tree

n [152]:
```
##Grid Search CV

from sklearn.model_selection import GridSearchCV


Gridregr = DecisionTreeRegressor(random_state = 0)
depth = list(range(1,20,2))

# Create the grid
grid_search = {'max_depth': depth}

## Grid Search Cross-Validation with 5 fold CV
gridcv_GDT = GridSearchCV(Gridregr, param_grid = grid_search, cv = 5)
gridcv_GDT = gridcv_GDT.fit(Xlr_train,y_train)
view_best_params_GDT = gridcv_GDT.best_params_

#Apply model on test data
predictions_GDT = gridcv_GDT.predict(Xlr_test)


GDT_r2 = r2_score(y_test, predictions_GDT)
GDT_mse = mean_squared_error(y_test, predictions_GDT)

#Calculate MAPE
GDT_mape = MAPE(y_test, predictions_GDT)

print('Grid Search CV Decision Regressor Model Performance:')
print('Best Parameters = ',view_best_params_GDT)
print('R-squared = {:0.2}.'.format(GDT_r2))
print('MSE = ',round(GDT_mse))
print('MAPE = {:0.4}%.'.format(GDT_mape))
print('****************************************')
```

```
Grid Search CV Decision Regressor Model Performance:
Best Parameters =  {'max_depth': 5}
R-squared = 0.79.
MSE =  913559.0
MAPE = 25.07%.
****************************************
```

Figure A.23: Grid search CV Decision Tree Modeling

Random Search CV - Random Forest

In [154]:

```
##Random Search CV
from sklearn.model_selection import RandomizedSearchCV

RRF = RandomForestRegressor(random_state = 0)
n_estimator = list(range(1,20,2))
depth = list(range(1,100,2))

# Create the random grid
rand_grid = {'n_estimators': n_estimator,
             'max_depth': depth}

randomcv_rf = RandomizedSearchCV(RRF, param_distributions = rand_grid, n_iter = 5, cv = 5, random_state=0)
randomcv_rf = randomcv_rf.fit(Xlr_train,y_train)
predictions_RRF = randomcv_rf.predict(Xlr_test)
predictions_RRF = np.array(predictions_RRF)

view_best_params_RRF = randomcv_rf.best_params_

best_model = randomcv_rf.best_estimator_

predictions_RRF = best_model.predict(Xlr_test)


#R^2
RRF_r2 = r2_score(y_test, predictions_RRF)
#Calculating MSE
RRF_mse = np.mean(( y_test - predictions_RRF)**2)
#Calculate MAPE
RRF_mape = MAPE(y_test, predictions_RRF)

print('Random Search CV Random Forest Regressor Model Performance:')
print('Best Parameters = ',view_best_params_RRF)
print('R-squared = {:0.2}.'.format(RRF_r2))
print('MSE = ',round(RRF_mse))
print('MAPE = {:0.4}%.'.format(RRF_mape))
print('**********************************************')
```

```
Random Search CV Random Forest Regressor Model Performance:
Best Parameters =  {'n_estimators': 15, 'max_depth': 23}
R-squared = 0.86.
MSE =  596208
MAPE = 20.51%.
**********************************************
```

Figure A.24: Random forest modeling

Grid Search CV - Random Forest

163]:
```
## Grid Search CV
from sklearn.model_selection import GridSearchCV

regr = RandomForestRegressor(random_state = 0)
n_estimator = list(range(11,17,2))
depth = list(range(15,25,1))

# Create the grid
grid_search = {'n_estimators': n_estimator,
               'max_depth': depth}

## Grid Search Cross-Validation with 5 fold CV
gridcv_rf = GridSearchCV(regr, param_grid = grid_search, cv = 5)
gridcv_rf = gridcv_rf.fit(Xlr_train,y_train)
view_best_params_GRF = gridcv_rf.best_params_

#Apply model on test data
predictions_GRF = gridcv_rf.predict(Xlr_test)


#R^2
GRF_r2 = r2_score(y_test, predictions_GRF)
#Calculating MSE
GRF_mse = np.mean(( y_test - predictions_GRF)**2)
#Calculate MAPE
GRF_mape = MAPE(y_test, predictions_GRF)

print('Grid Search CV Random Forest Regressor Model Performance:')
print('Best Parameters = ',view_best_params_GRF)
print('R-squared = {:0.2}.'.format(GRF_r2))
print('MSE = ',round(GRF_mse))
print('MAPE = {:0.4}%.'.format(GRF_mape))
print('**********************************************')
```

```
Grid Search CV Random Forest Regressor Model Performance:
Best Parameters =  {'max_depth': 18, 'n_estimators': 15}
R-squared = 0.86.
MSE =  595450
MAPE = 20.57%.
**********************************************
```

Figure A.25: Random Search CV Random Forest Modeling.

Random Search CV - Random Forest

154]:
```
##Random Search CV
from sklearn.model_selection import RandomizedSearchCV

RRF = RandomForestRegressor(random_state = 0)
n_estimator = list(range(1,20,2))
depth = list(range(1,100,2))

# Create the random grid
rand_grid = {'n_estimators': n_estimator,
             'max_depth': depth}

randomcv_rf = RandomizedSearchCV(RRF, param_distributions = rand_grid, n_iter = 5, cv = 5, random_state=0)
randomcv_rf = randomcv_rf.fit(Xlr_train,y_train)
predictions_RRF = randomcv_rf.predict(Xlr_test)
predictions_RRF = np.array(predictions_RRF)

view_best_params_RRF = randomcv_rf.best_params_

best_model = randomcv_rf.best_estimator_

predictions_RRF = best_model.predict(Xlr_test)


#R^2
RRF_r2 = r2_score(y_test, predictions_RRF)
#Calculating MSE
RRF_mse = np.mean(( y_test - predictions_RRF)**2)
#Calculate MAPE
RRF_mape = MAPE(y_test, predictions_RRF)

print('Random Search CV Random Forest Regressor Model Performance:')
print('Best Parameters = ',view_best_params_RRF)
print('R-squared = {:0.2}.'.format(RRF_r2))
print('MSE = ',round(RRF_mse))
print('MAPE = {:0.4}%.'.format(RRF_mape))
print('**********************************************')
```

```
Random Search CV Random Forest Regressor Model Performance:
Best Parameters =  {'n_estimators': 15, 'max_depth': 23}
R-squared = 0.86.
MSE =  596208
MAPE = 20.51%.
**********************************************
```

Figure A.26: Grid search CV Random Forest Modeling

Gradient Boosting Machine

In [156]:
```
#Gradient Boost
from sklearn.ensemble import GradientBoostingRegressor

gbt = GradientBoostingRegressor(random_state= 0).fit(Xlr_train,y_train)

predictions_gbt = gbt.predict(Xlr_test)


gbt.get_params()

#R^2
GBR_r2 = r2_score(y_test, predictions_gbt)
#Calculate MSE
GBR_mse = mean_squared_error(y_test, predictions_gbt)

#Calculate MAPE
GBR_mape = MAPE(y_test, predictions_gbt)

print('Gradient Boosting Regressor Model Performance:')
print('Default Parameters = ',gbt.get_params())
print('R-squared = {:0.2}.'.format(GBR_r2))
print('MSE = ',round(GBR_mse))
print('MAPE = {:0.4}%.'.format(GBR_mape))
print('************************************************')
```

```
Gradient Boosting Regressor Model Performance:
Default Parameters =  {'alpha': 0.9, 'criterion': 'friedman_mse', 'init': None, 'learning_rate': 0.1, 'loss': 'ls', 'ma
x_depth': 3, 'max_features': None, 'max_leaf_nodes': None, 'min_impurity_decrease': 0.0, 'min_impurity_split': None, 'm
in_samples_leaf': 1, 'min_samples_split': 2, 'min_weight_fraction_leaf': 0.0, 'n_estimators': 100, 'presort': 'auto', '
random_state': 0, 'subsample': 1.0, 'verbose': 0, 'warm_start': False}
R-squared = 0.88.
MSE =  514160.0
MAPE = 18.51%.
************************************************
```

Figure A.27: Gradient Boosting

Random Search CV - Gradient Boosting

ı [157]:
```
##Random Search CV
rGBR = GradientBoostingRegressor(random_state = 0)
#loss = ['ls','lad','huber','quantile']
n_estimator = list(range(50,150,10))
#max_feat = ['auto','sqrt','log2']
depth = list(range(1,10,2))

# Create the random grid
rand_GBT = {#'loss': loss,
            'n_estimators': n_estimator,
            #'max_features': max_feat,
            'max_depth': depth}

randomcv_gbt = RandomizedSearchCV(rGBR, param_distributions = rand_GBT, n_iter = 5, cv = 5, random_state=0)
randomcv_gbt = randomcv_gbt.fit(Xlr_train,y_train)
predictions_GBT = randomcv_gbt.predict(Xlr_test)


view_best_params_GBT = randomcv_gbt.best_params_

#R^2
rGBR_r2 = r2_score(y_test, predictions_GBT)
#Calculate MSE
rGBR_mse = mean_squared_error(y_test, predictions_GBT)

#Calculate MAPE
rGBR_mape = MAPE(y_test, predictions_GBT)

print('Random Search CV Gradient Boosting Regressor Model Performance:')
print('Best Parameters = ',view_best_params_GBT)
print('R-squared = {:0.2}.'.format(rGBR_r2))
print('MSE = ',round(rGBR_mse))
print('MAPE = {:0.4}%.'.format(rGBR_mape))
print('************************************************')
##########################################################################################################################
```

```
Random Search CV Gradient Boosting Regressor Model Performance:
Best Parameters =  {'n_estimators': 60, 'max_depth': 3}
R-squared = 0.88.
MSE =  522381.0
MAPE = 18.41%.
************************************************
```

Figure A.28: Random Search CV Gradient Boosting Modeling.

****

Gride Search CV - Gradient Boosting

In [158]:
```
## Gride Search CV

gGBR = GradientBoostingRegressor(random_state=0)
#loss = ['ls','lad','huber','quantile']
n_estimator = list(range(40,80,5))
#max_feat = ['auto','sqrt','log2']
depth = list(range(1,5,1))

# Create the random grid
grid_GBT = {#'loss': loss,
            'n_estimators': n_estimator,
            #'max_features': max_feat,
            'max_depth': depth}

## Grid Search Cross-Validation with 5 fold CV
gridcv_GBT = GridSearchCV(gGBR, param_grid = grid_GBT, cv = 5)
gridcv_GBT = gridcv_GBT.fit(Xlr_train,y_train)
view_best_params_gridGRF = gridcv_GBT.best_params_

#Apply model on test data
predictions_gridGBT = gridcv_GBT.predict(Xlr_test)

#R^2
gGBR_r2 = r2_score(y_test, predictions_gridGBT)
#Calculate MSE
gGBR_mse = mean_squared_error(y_test, predictions_gridGBT)

#Calculate MAPE
gGBR_mape = MAPE(y_test, predictions_gridGBT)

print('Grid Search CV Gradient Boosting Regressor Model Performance:')
print('Best Parameters = ',view_best_params_gridGRF)
print('R-squared = {:0.2}.'.format(gGBR_r2))
print('MSE = ',round(gGBR_mse))
print('MAPE = {:0.4}%.'.format(gGBR_mape))
print('**********************************************')
```

```
Grid Search CV Gradient Boosting Regressor Model Performance:
Best Parameters =  {'max_depth': 3, 'n_estimators': 75}
R-squared = 0.88.
MSE =  514395.0
MAPE = 18.06%.
**********************************************
```

Figure A.29: Grid search CV Gradient Boosting Modeling

Extra Graident Boosting Machine

In [159]:
```
# Extra Graident Boosting Machine
import xgboost as xgb

xgb_reg = xgb.XGBRegressor(random_state = 0)
xgb_reg.fit(Xlr_train,y_train)


predictions_xgb = xgb_reg.predict(Xlr_test)

#R^2
XBT_r2 = r2_score(y_test, predictions_xgb)
#Calculate MSE
XBT_mse = mean_squared_error(y_test, predictions_xgb)

#Calculate MAPE
XBT_mape = MAPE(y_test, predictions_xgb)

print('XG Boosting Regressor Model Performance:')
print('XG Boosting parameters = ',xgb_reg.get_params())
print('R-squared = {:0.2}.'.format(XBT_r2))
print('MSE = ',round(XBT_mse))
print('MAPE = {:0.4}%.'.format(XBT_mape))
##########################################################################################################################
```

```
XG Boosting Regressor Model Performance:
XG Boosting parameters =  {'base_score': 0.5, 'booster': 'gbtree', 'colsample_bylevel': 1, 'colsample_bytree': 1, 'gamm
a': 0, 'learning_rate': 0.1, 'max_delta_step': 0, 'max_depth': 3, 'min_child_weight': 1, 'missing': None, 'n_estimators
': 100, 'n_jobs': 1, 'nthread': None, 'objective': 'reg:linear', 'random_state': 0, 'reg_alpha': 0, 'reg_lambda': 1, 's
cale_pos_weight': 1, 'seed': None, 'silent': True, 'subsample': 1}
R-squared = 0.88.
MSE =  499137.0
MAPE = 18.25%.
```

Figure A.30: Extra Gradient Boosting Modeling

Appendix B

R program and its corresponding output

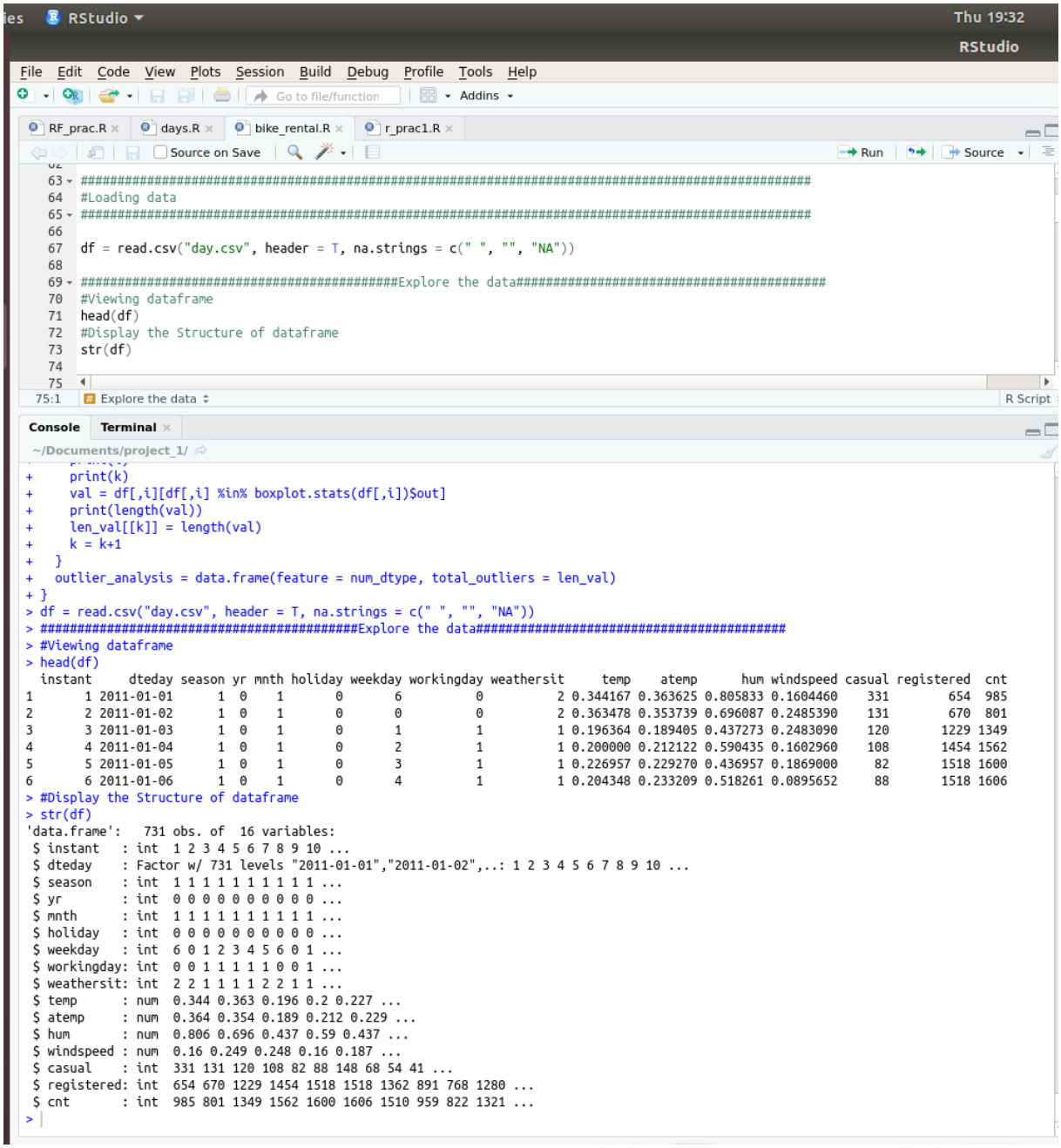


Figure B.1: Loading data and analyzing its properties.

```
ggplot(data = df, aes(x = reorder(weekday,-cnt), y = cnt))+
  geom_bar(stat = "identity",fill = "white")+
  labs(title = "Number of bikes rented with respect to days", x = "Days of the week")+
  theme(panel.background = element_rect("cadetblue"))+
  theme(plot.title = element_text(hjust = 0.5, face = "bold"))
```

Figure B.2: Code to plot number of bikes rented with respect to the day of the week

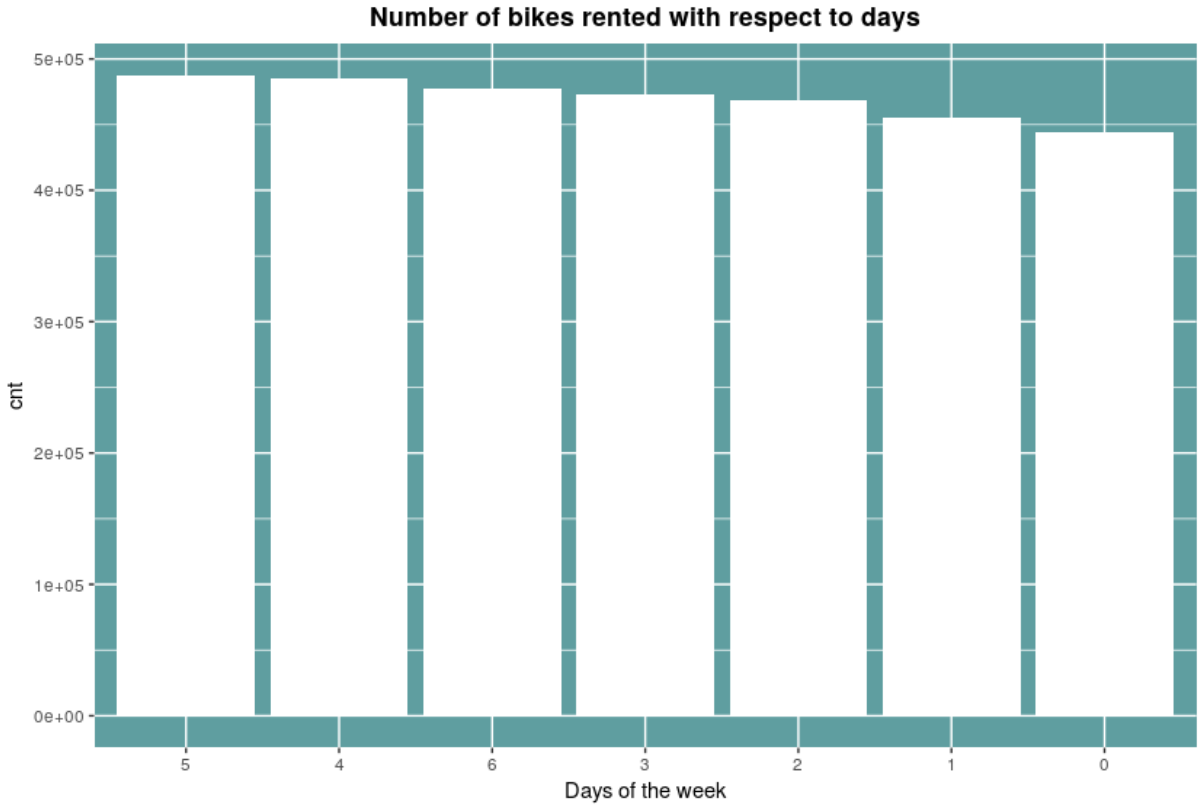


Figure B.3: Number of bikes rented with respect to the day of the week. This is the output for code in figure B.2. It can be seen that the most number of bikes are rented on day 5 and the least on day 0

```
ggplot(df,aes(temp,cnt)) +
  geom_point(aes(color=hum),alpha=0.5) +
  labs(title = "Bikes rented with respect to variation in temperature and hunidity", x = "Normalized temperature")+
  scale_color_gradientn(colors=c('dark blue','blue','light blue','light green','yellow','orange','red')) +
  theme_bw()
```

Figure B.4: Code to visualize bikes rented with respect to variation in temperature and humidity.

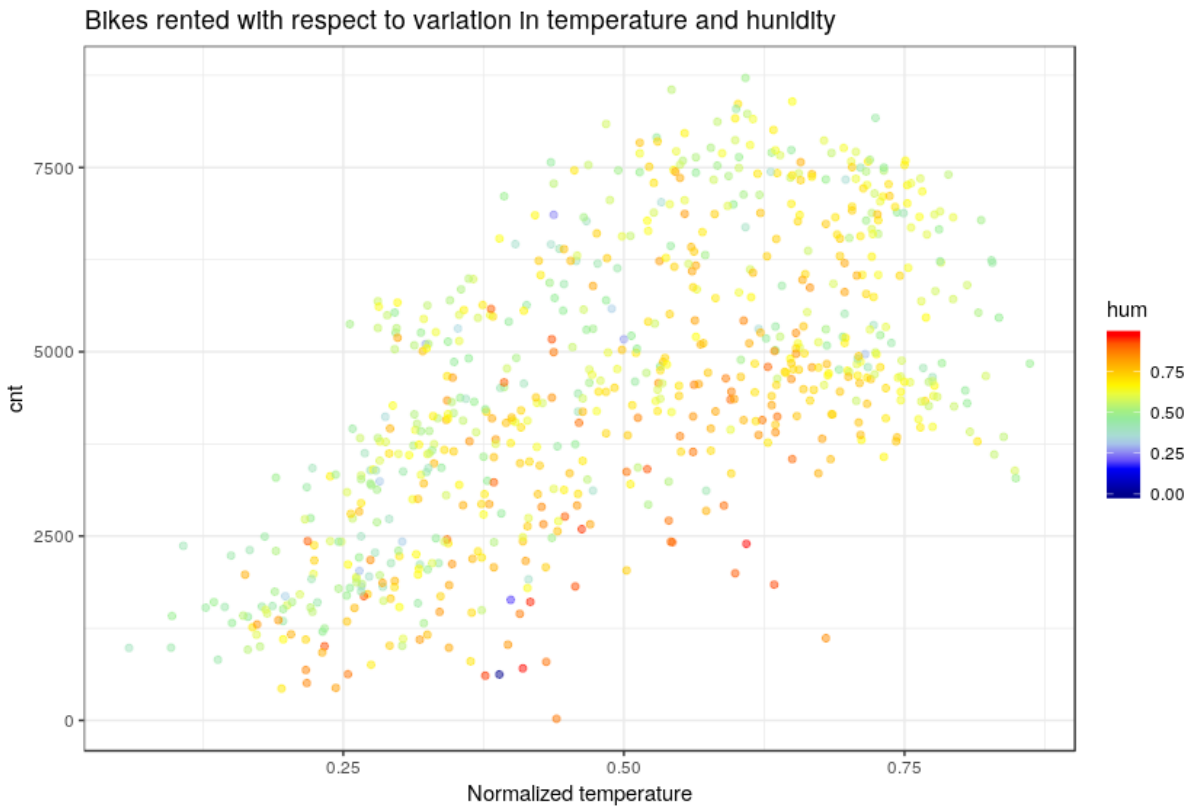


Figure B.5: Bikes rented with respect to variation in temperature and humidity. This is the output for figure B.4. It can be observed that people rent bikes mostly when temperature in between 0.5 and0.75 normalized temperature and between normalized humidity 0.50 and 0.75

```
ggplot(data = df, aes(x = temp, y = cnt))+
  geom_point(aes(color=weathersit))+
  labs(title = "Bikes rented with respect to temperature and weathersite", x = "Normalized temperature")+
  theme(plot.title = element_text(hjust = 0.5, face = "bold"))+
  theme_bw()
```

Figure B.6: Code to visualize bikes rented with respect to variation in temperature and weathersite.

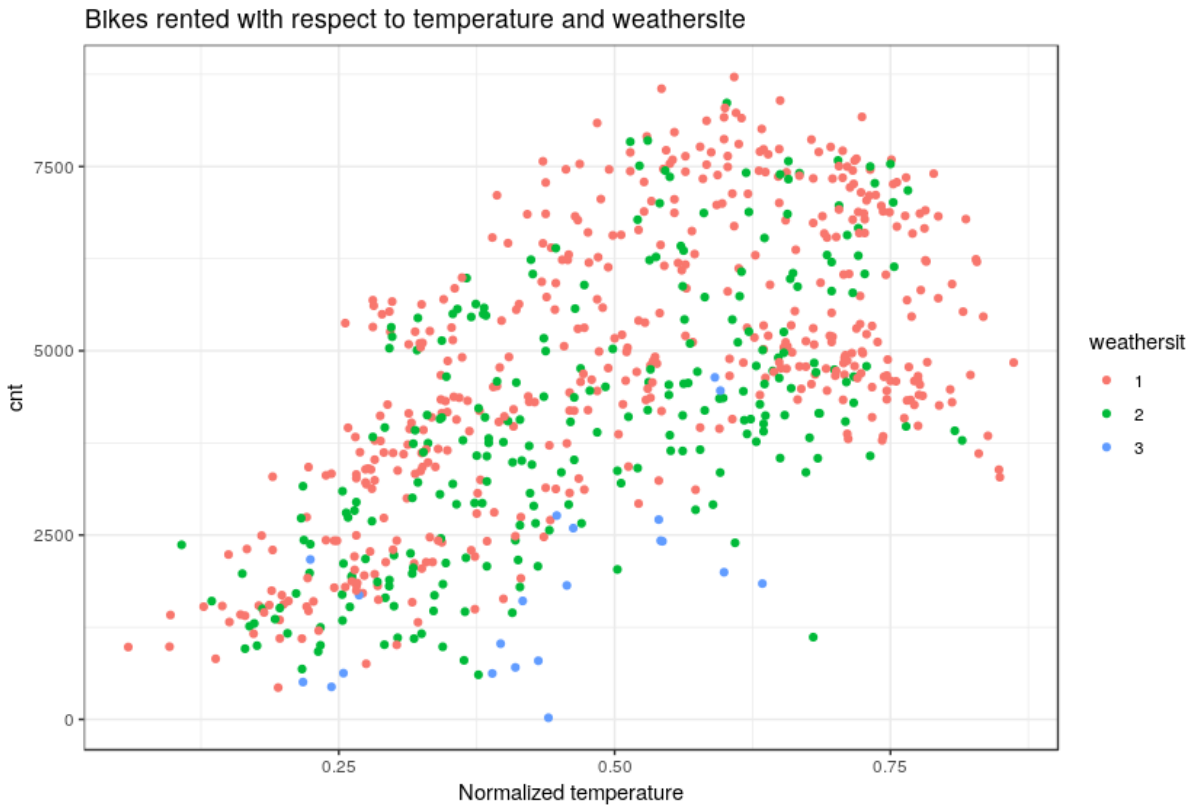


Figure B.7: Bikes rented with respect to variation in temperature and weathersite.This is the output for figure B.6. Most bikes are rented duing weather site forcast 1.

```
ggplot(data = df, aes(x = temp, y = cnt))+
  geom_point(aes(color=workingday))+
  labs(title = "Bikes rented with respect to temperature and workingday", x = "Normalized temperature")+
  #  theme(panel.background = element_rect("white"))+
  theme(plot.title = element_text(hjust = 0.5, face = "bold"))+
  theme_bw()
```

Figure B.8: Code to visualize bikes rented with respect to variation in temperature and workingday.

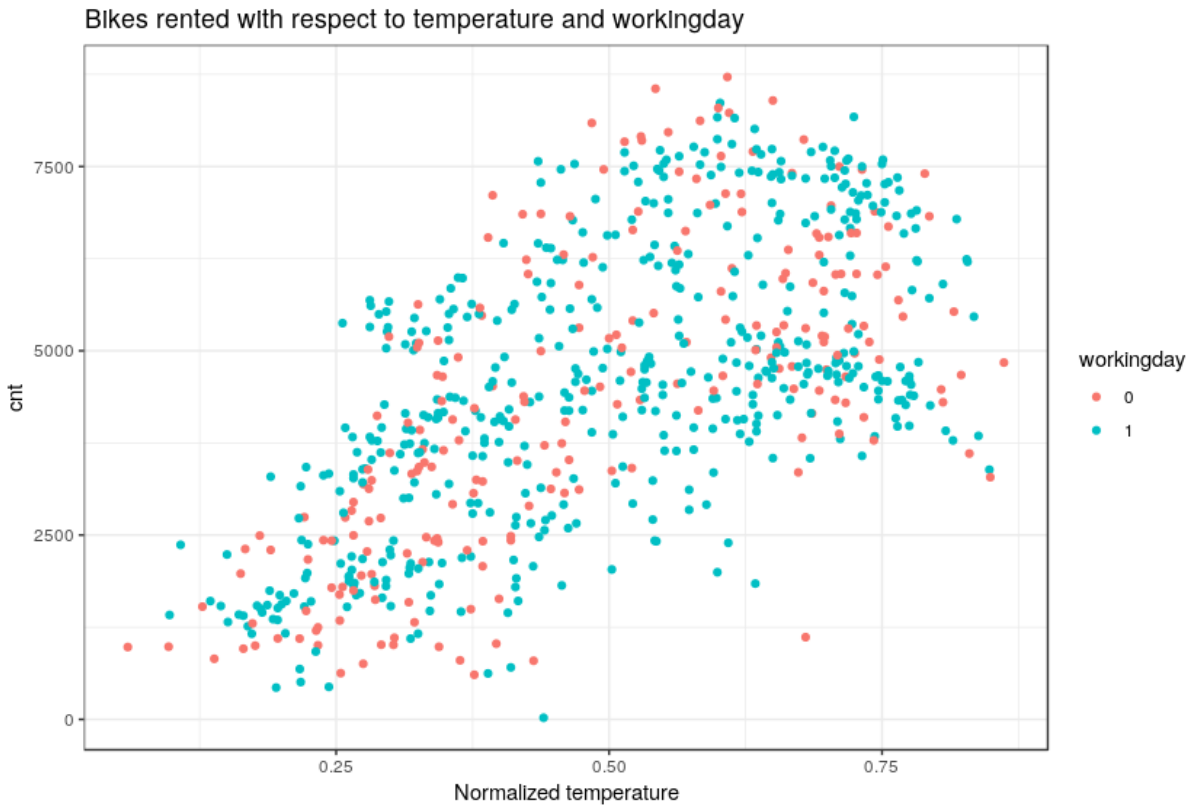


Figure B.9: Bikes rented with respect to variation in temperature and weathersite.This is the output for figure B.8. People rent bikes mostly on working weekdays.

```
########################################################################
# Distribution Plot
########################################################################

#Plot Temperature distribution (temp)
ggplot(data = df, aes(x = temp))+
  geom_histogram(bins = 30, fill = "white", col = "black")+
  labs(title = "Temperature distribution (temp)", x = "Normalized temperature")+
  theme(panel.background = element_rect("cadetblue"))+
  theme(plot.title = element_text(hjust = 0.5, face = "bold"))

#PLot Real feel temperature distribution (atemp)
ggplot(data = df, aes(x = atemp))+
  geom_histogram(bins = 30, fill = "white", col = "black")+
  labs(title = "Real feel temperature distribution (atemp)", x = "Normalized temperature")+
  theme(panel.background = element_rect("cadetblue"))+
  theme(plot.title = element_text(hjust = 0.5, face = "bold"))

#Plot Humidity distribution
ggplot(data = df, aes(x = hum))+
  geom_histogram(bins = 30, fill = "white", col = "black")+
  labs(title = "Humidity distribution", x = "Normalized humidity")+
  theme(panel.background = element_rect("cadetblue"))+
  theme(plot.title = element_text(hjust = 0.5, face = "bold"))

#Plot Windspeed distribution
ggplot(data = df, aes(x = windspeed))+
  geom_histogram(bins = 30, fill = "white", col = "black")+
  labs(title = "Windspeed distribution", x = "Features")+
  theme(panel.background = element_rect("cadetblue"))+
  theme(plot.title = element_text(hjust = 0.5, face = "bold"))
```

Figure B.10: Code to visualize distribution of continous variables.

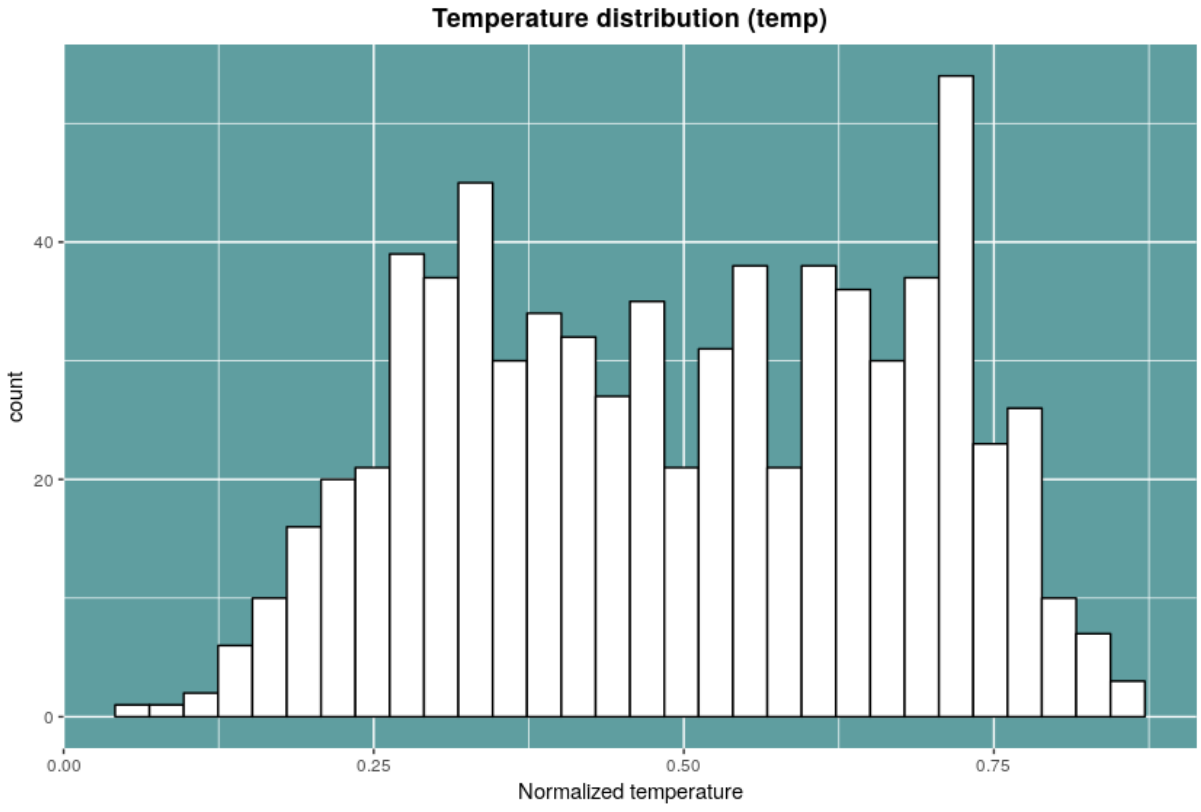


Figure B.11: Distribution of temp variables. This is the output for figure B.10

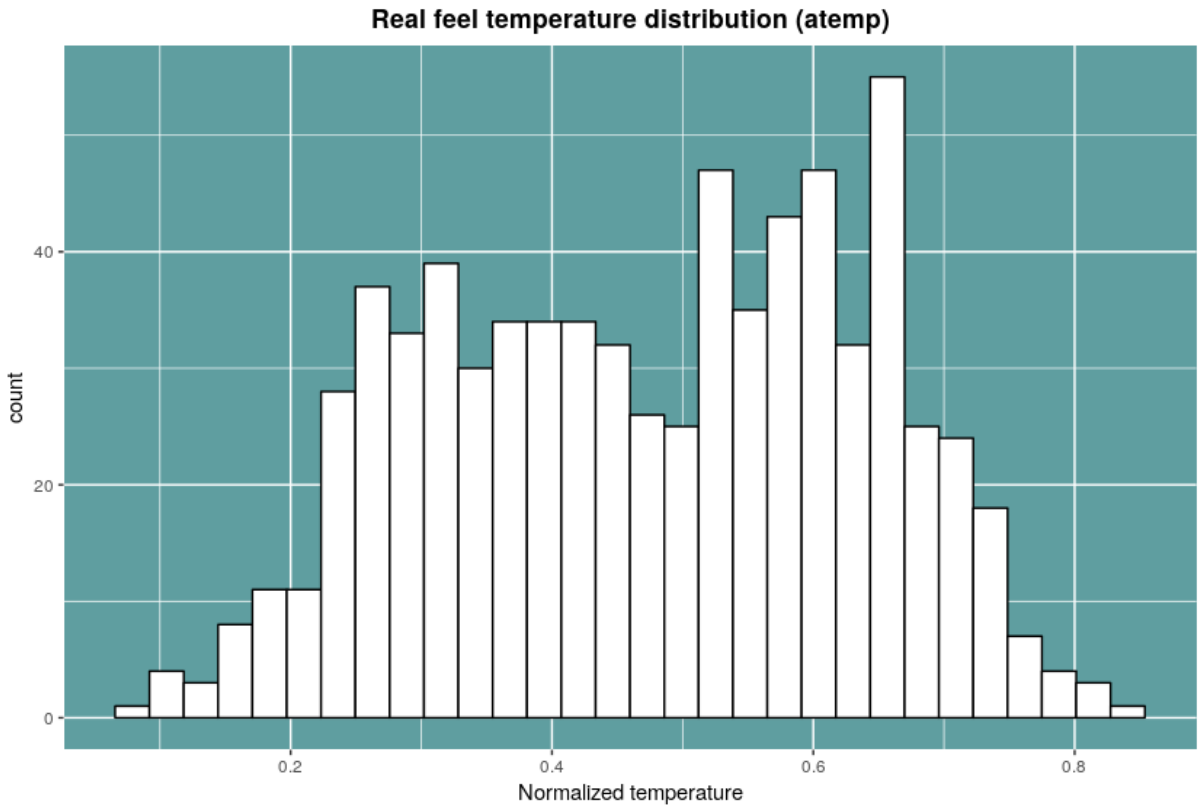


Figure B.12: Distribution of atemp variables. This is the output for figure B.10

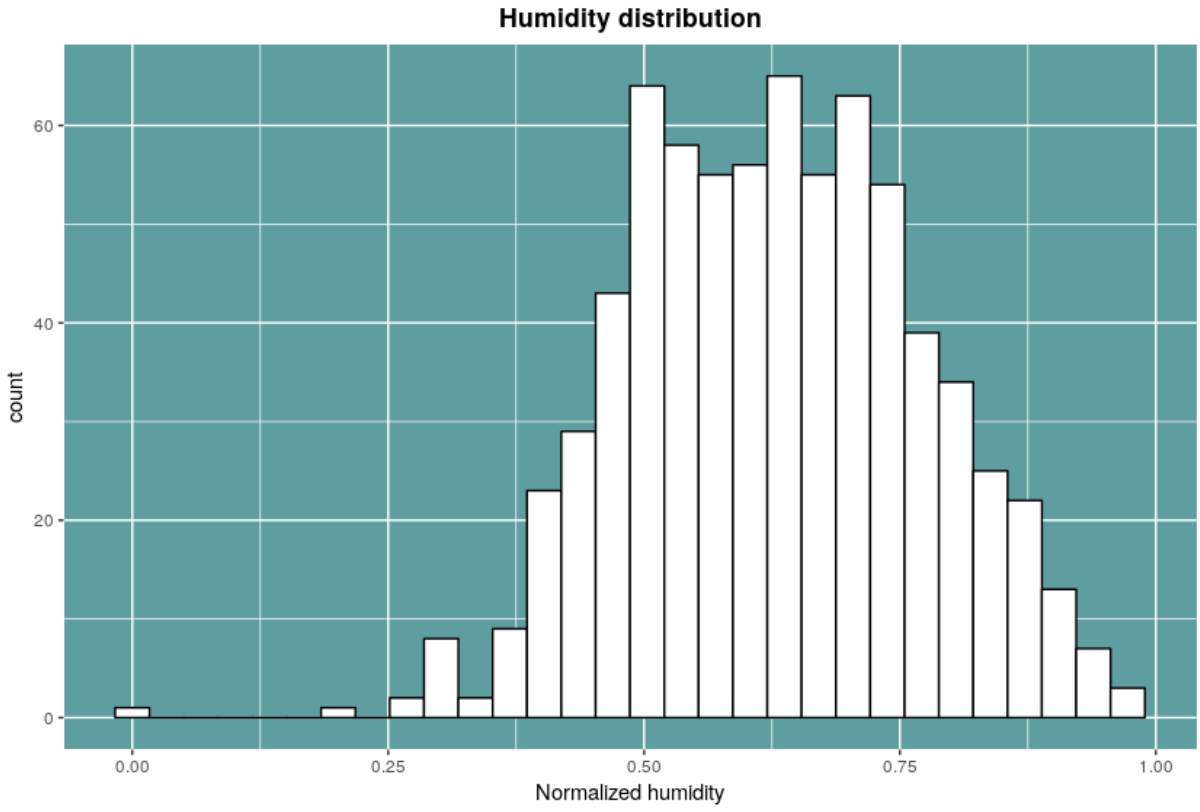


Figure B.13: Distribution of hum variables. This is the output for figure B.10

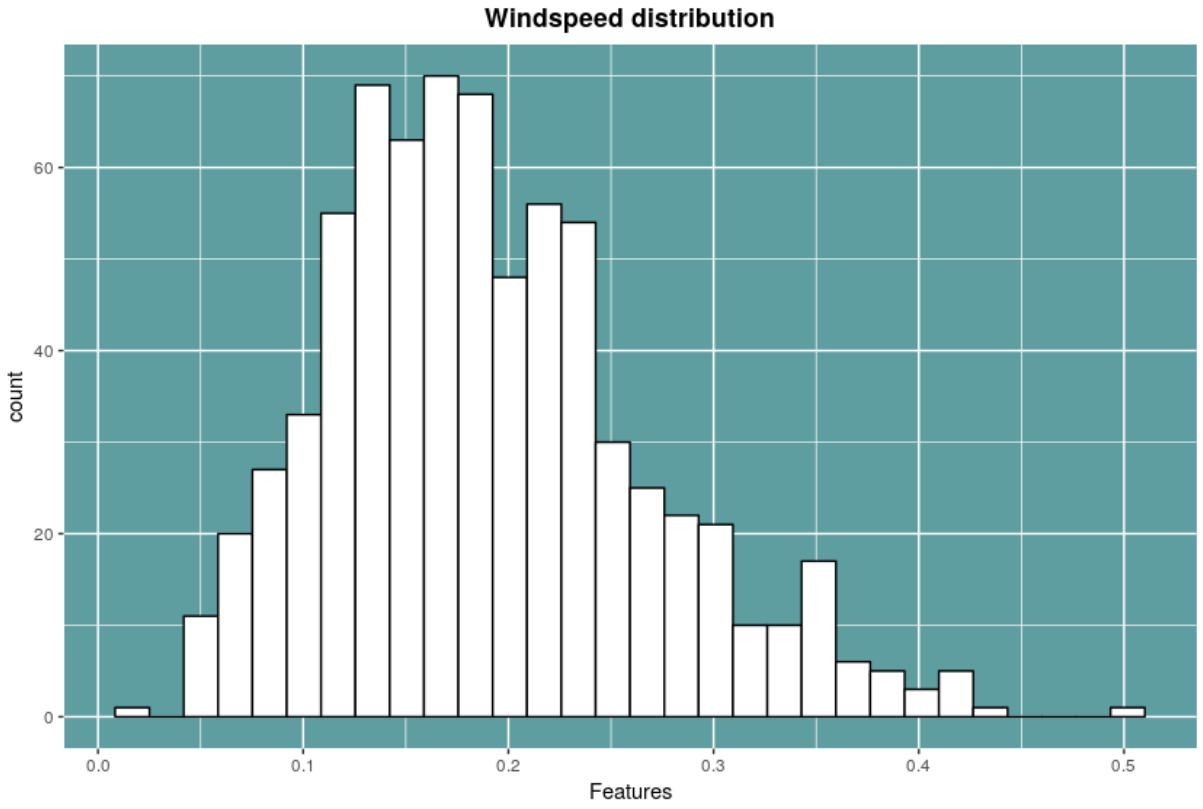


Figure B.14: Distribution of windspeed variables. This is the output for figure B.10

```
> ####################################
> # Missing Values Analysis
> ####################################
> missing_val = sum(is.na(df))
> print(missing_val)
[1] 0
>
```

Figure B.15: Missing value analysis

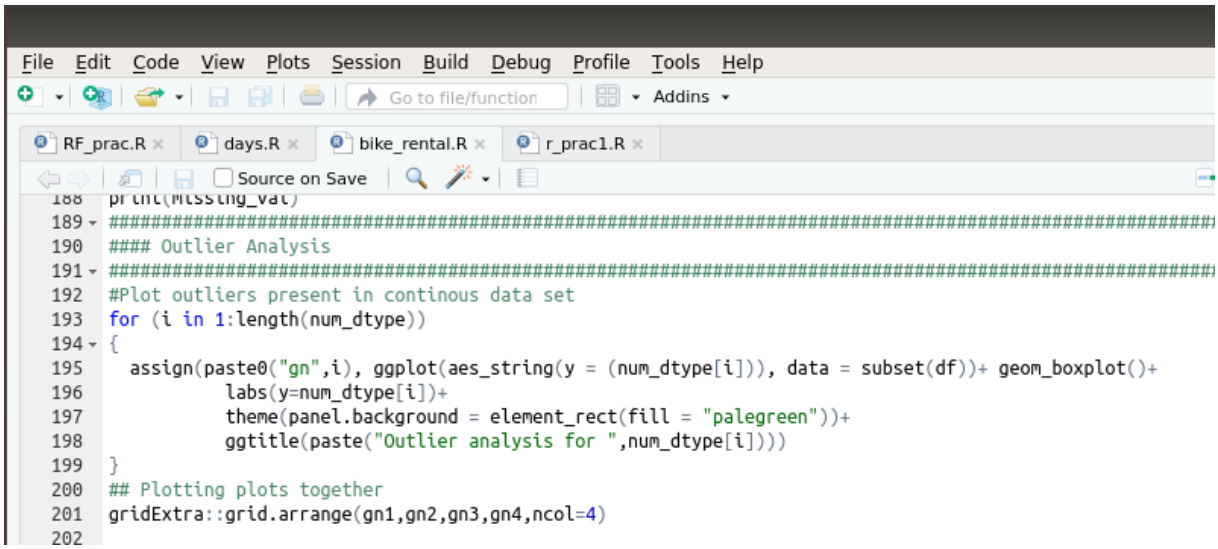


Figure B.16: Code to visualize outliers present in continuous variables.

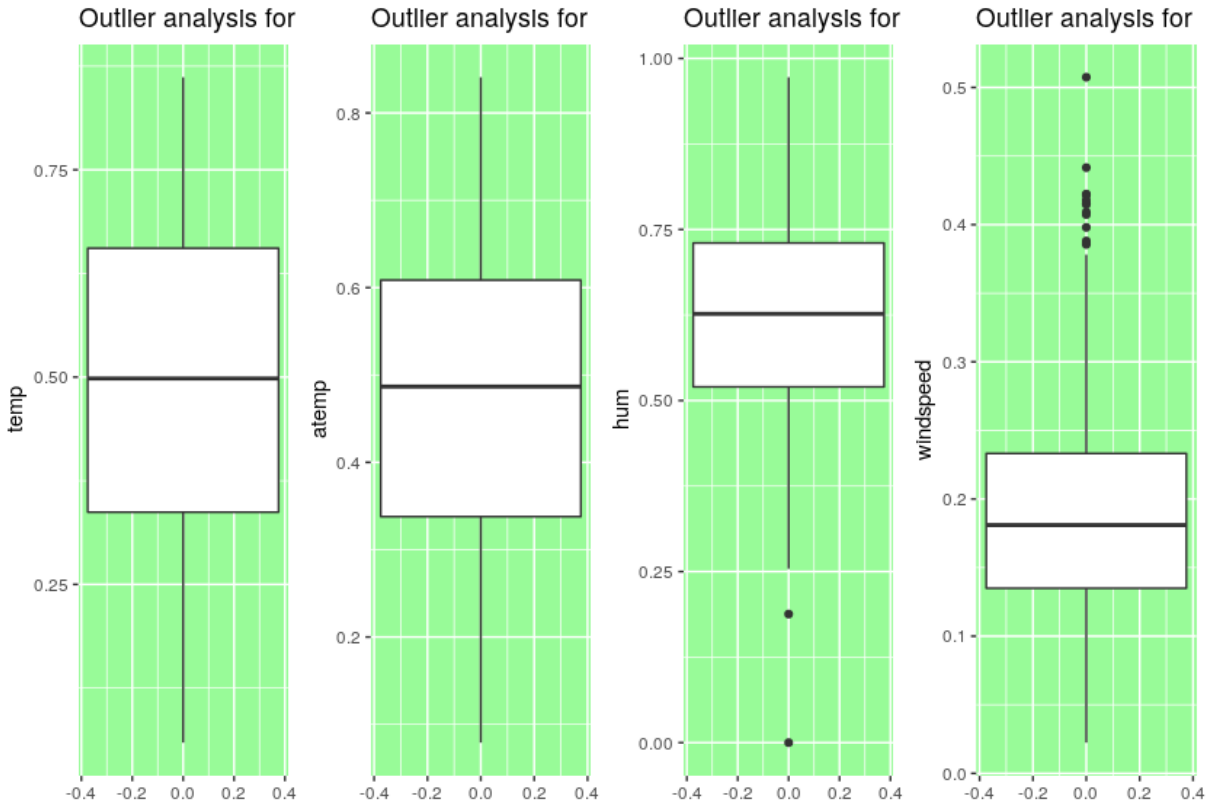


Figure B.17: Boxplot for outlier analysis in figure B.16.

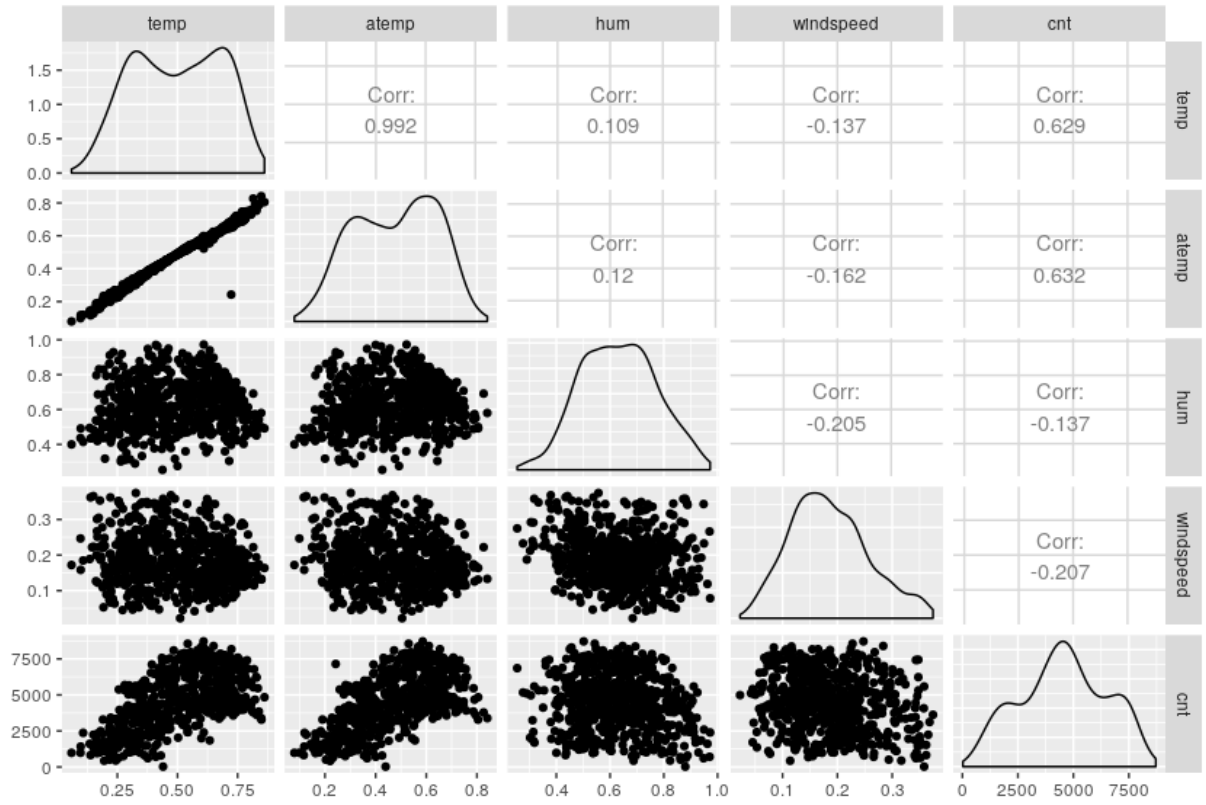


Figure B.18: Correlation plot, we can see that atemp and temp is very strongly correlated with each other.

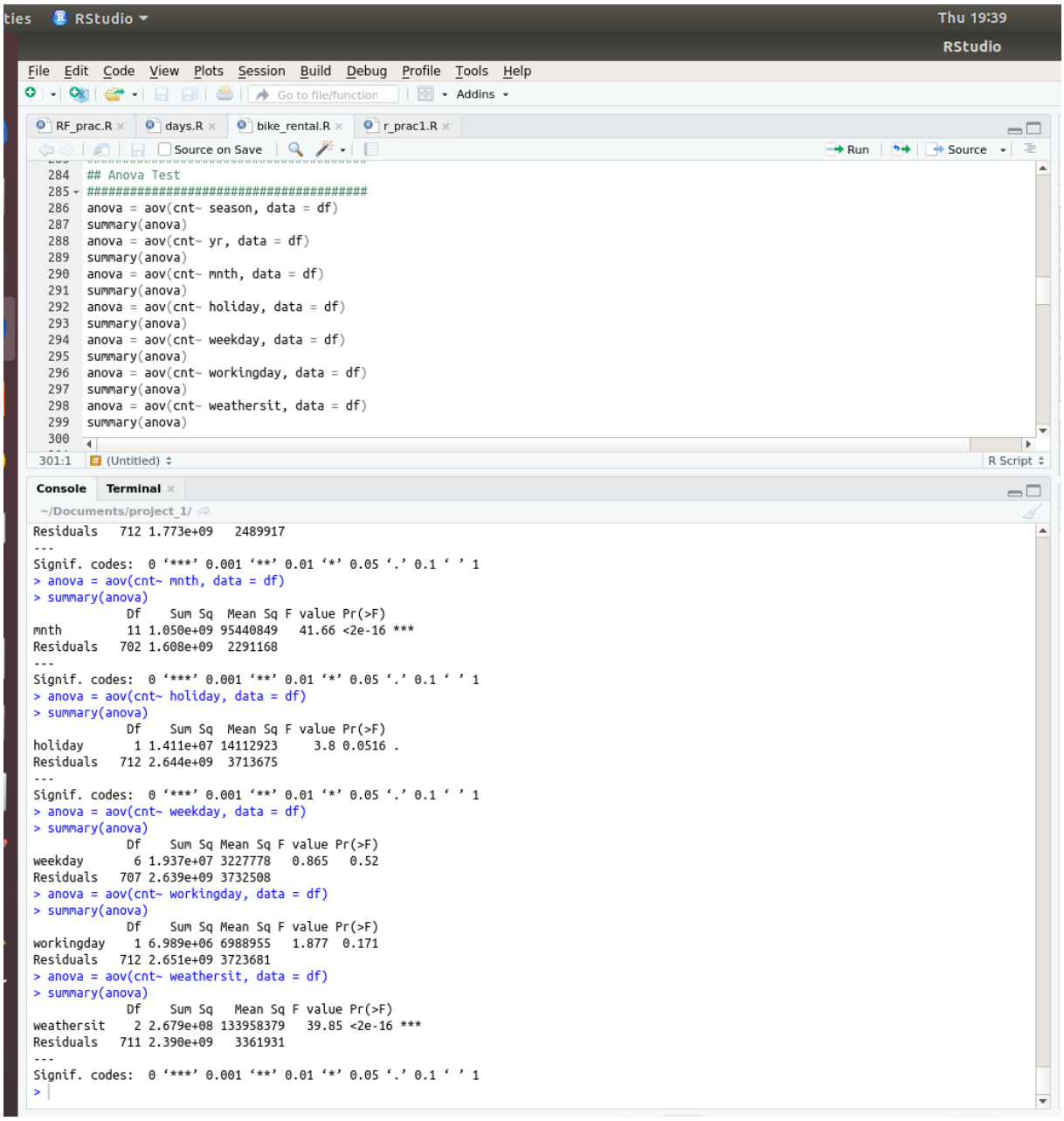


Figure B.19: ANOVA Test

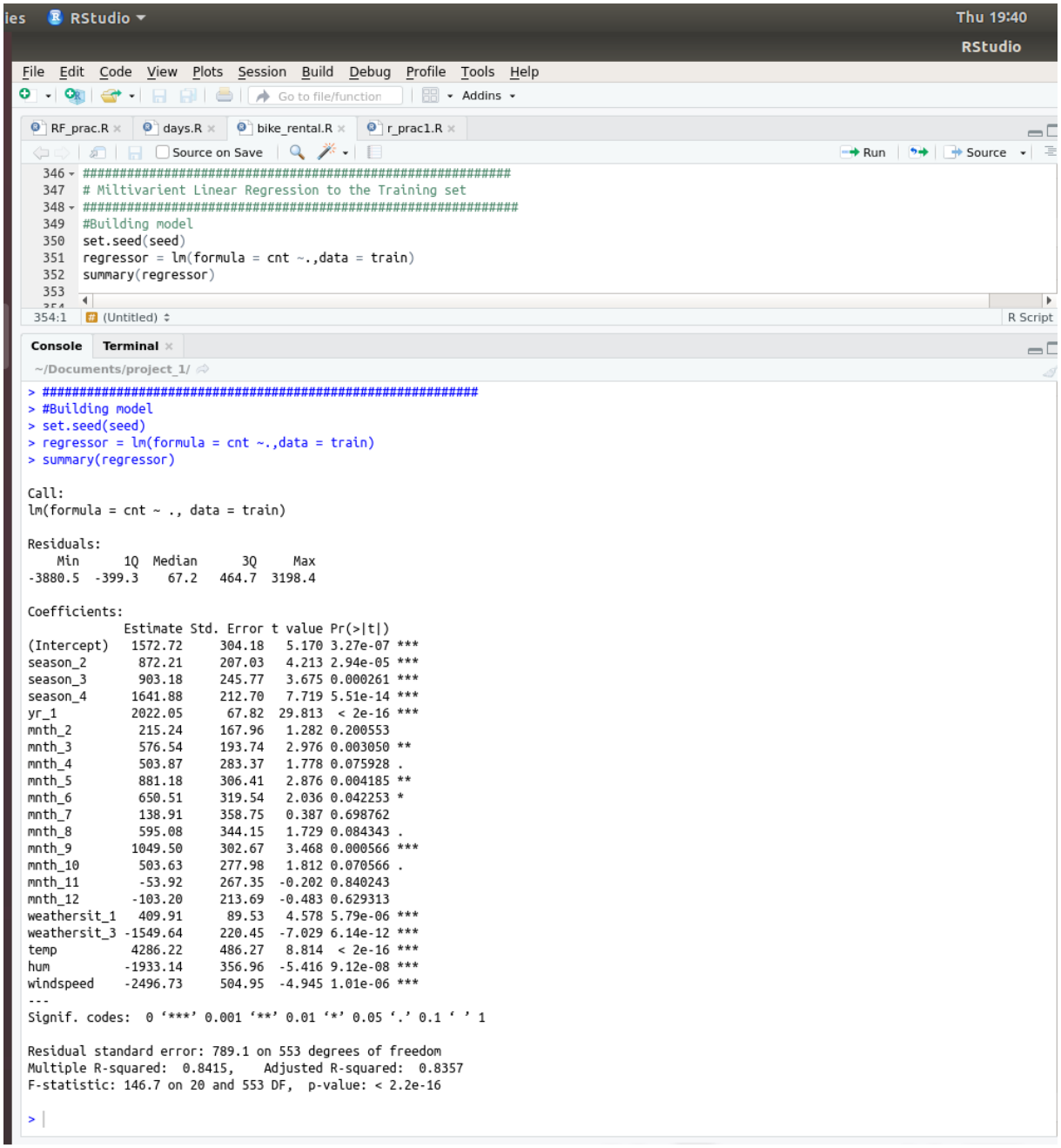


Figure B.20: Multivariate linear regression modeled on the trained data set. It can be seen from the out put of thethis model can explain 83% of the data.

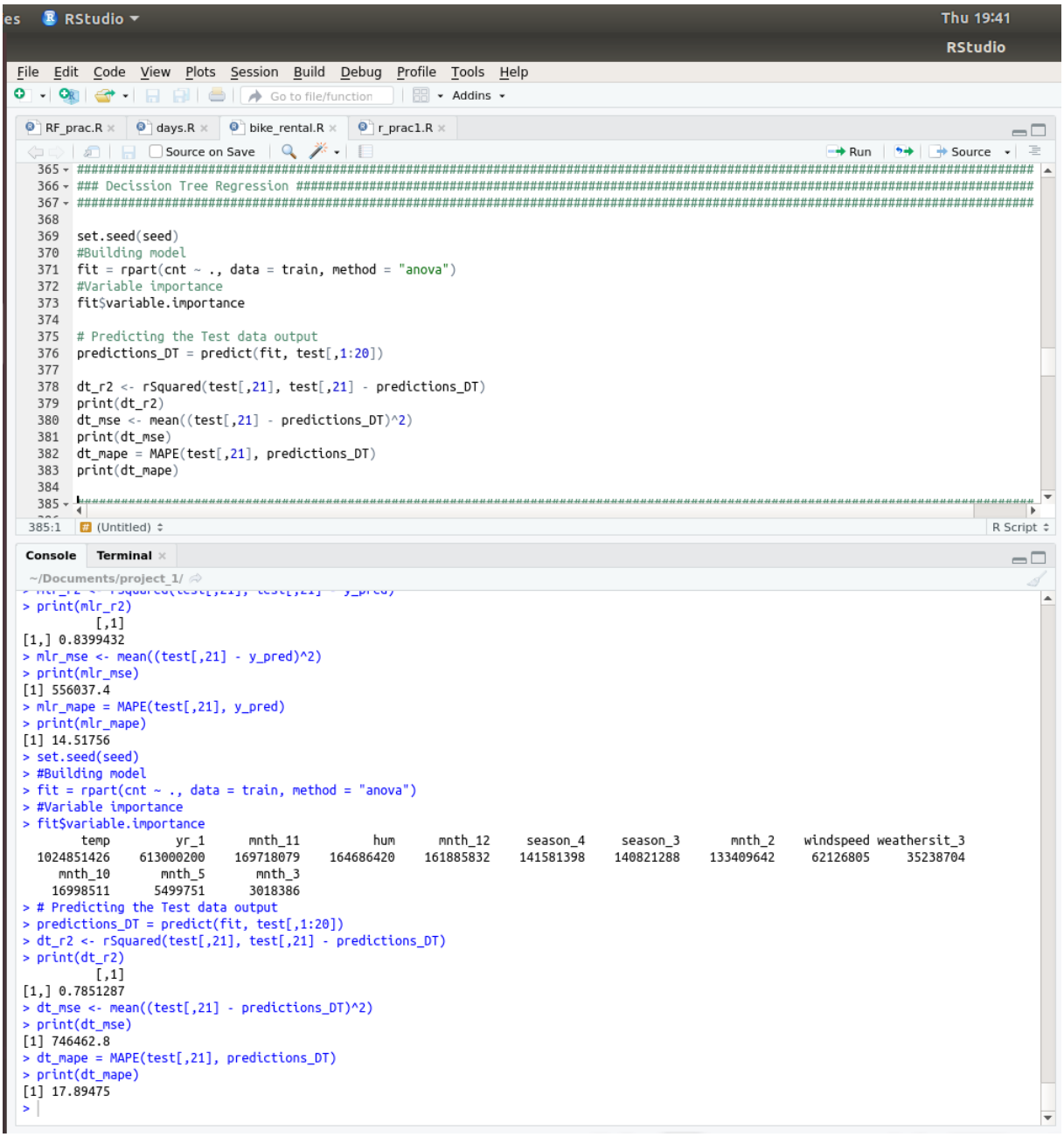


Figure B.21: A Decision tree was modeled on to our train data and the model has an MAPE of 17.89%

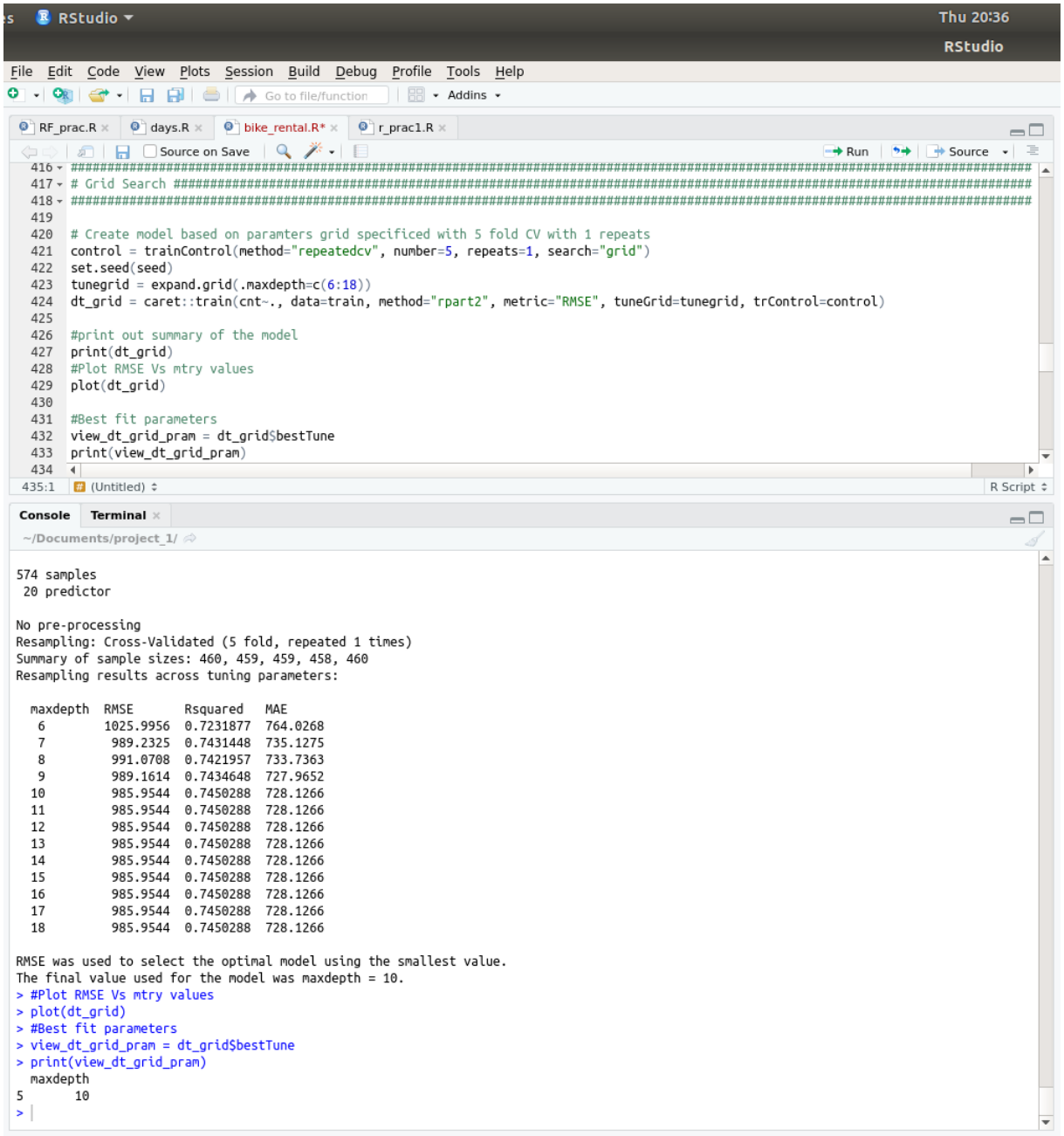


Figure B.22: Grid Search CV Decision tree modeling was performed on the data. Cross validation was conducted 5 folds on the data and the out put of the model is ploted.

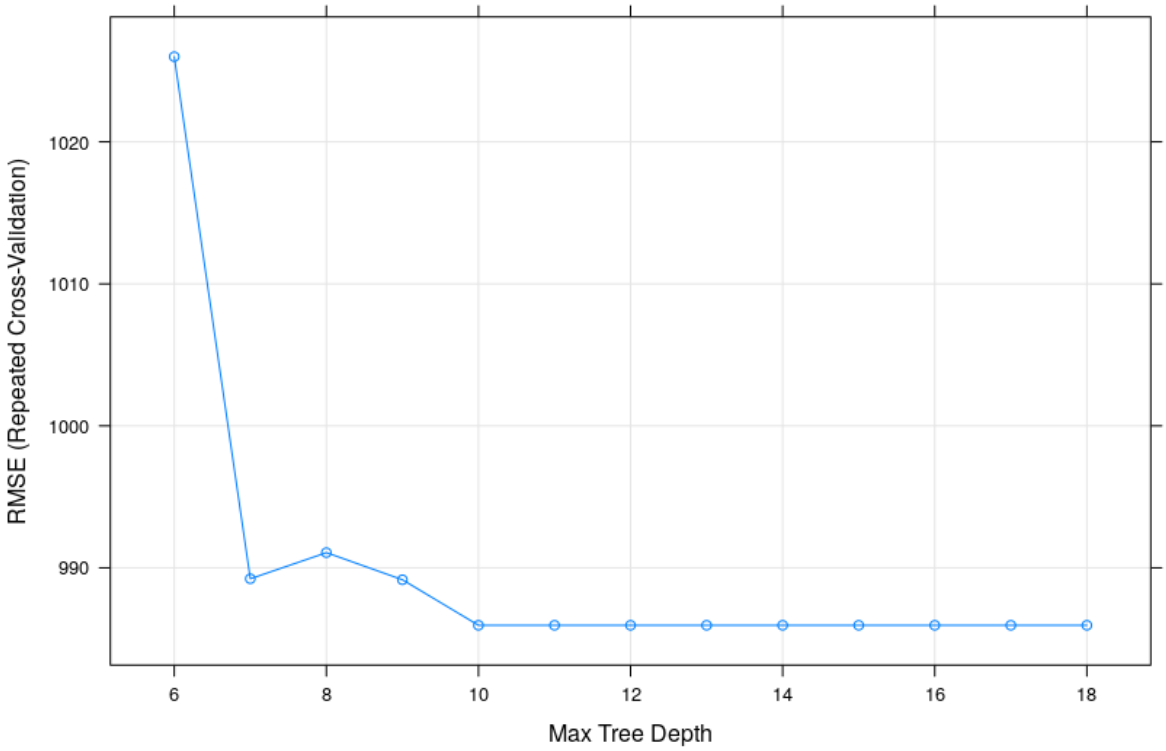


Figure B.23: Maximum tree depth Vs RMSE for repeated cross validation is plotted. It can be noted that the least RMSE is achived at the depth 10. This is the output for figure B.22.

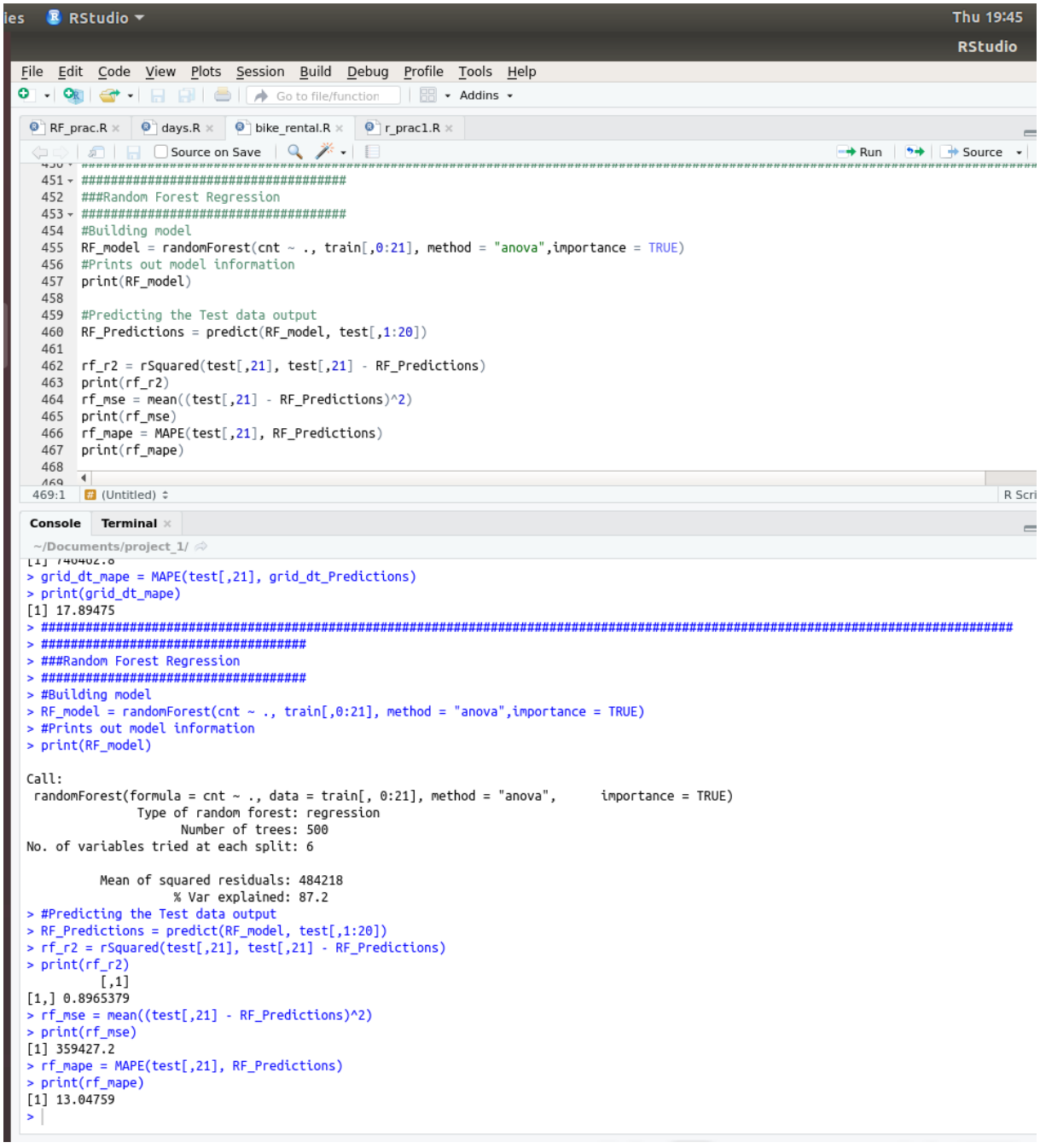


Figure B.24: A random forest modeling was modeled on to our train data and the model has an MAPE of 13.04%

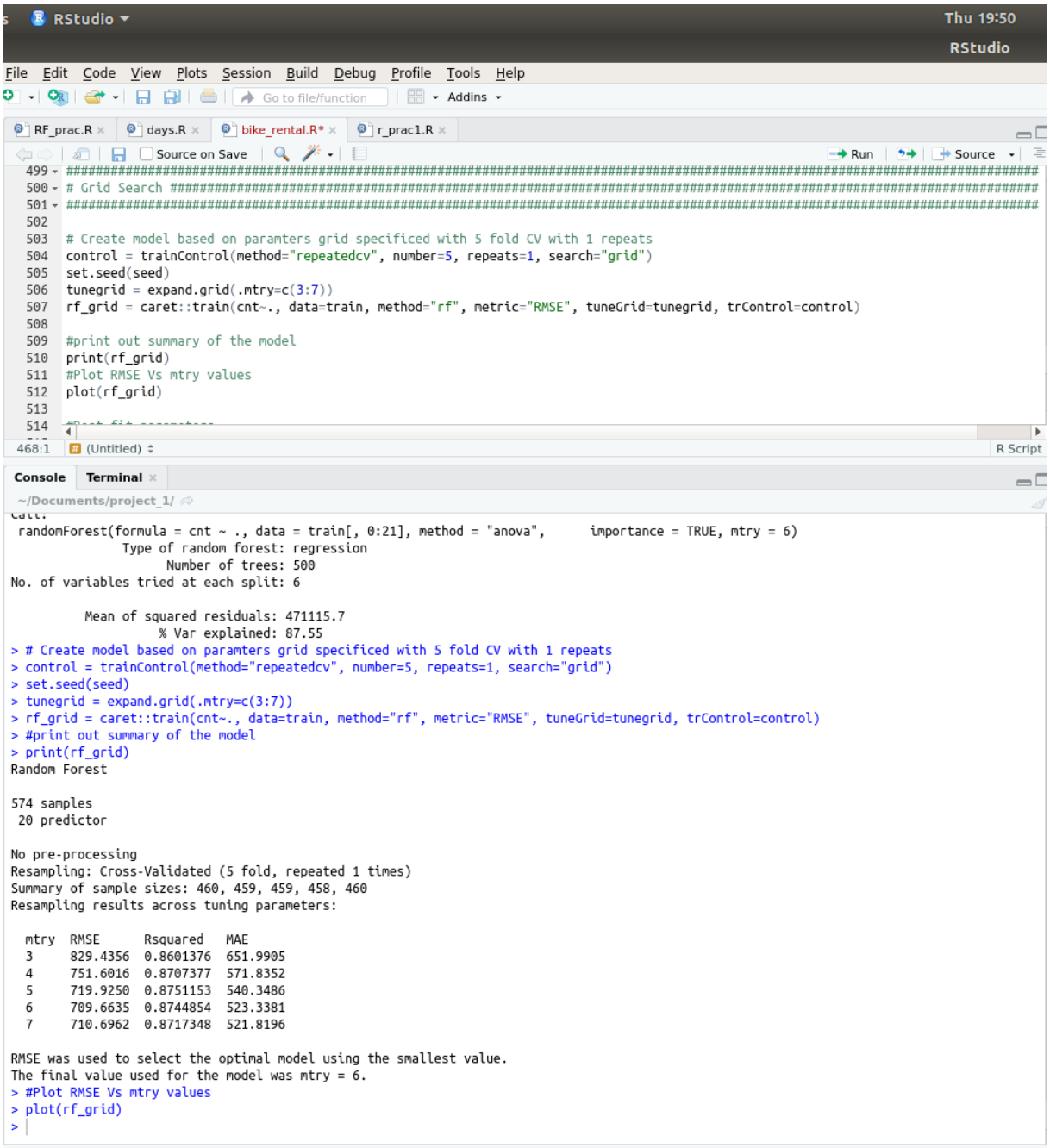


Figure B.25: Grid Search CV Random forest modeling was performed on the data. Cross validation was conducted 5 folds on the data and the out put of the model is ploted.

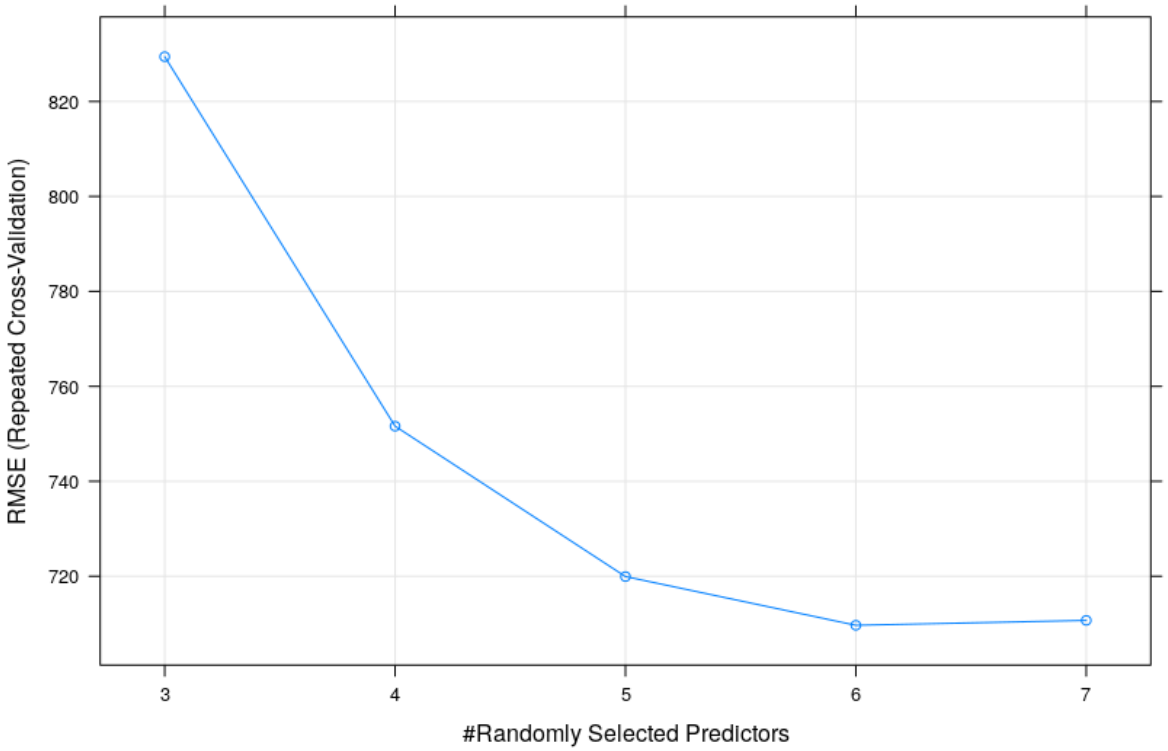


Figure B.26: Randomly selected predictors Vs RMSE for repeated cross validation is plotted. It can be noted that the least RMSE is achived when the randomly selected predictors is 6.This is the output is for figure B.25.

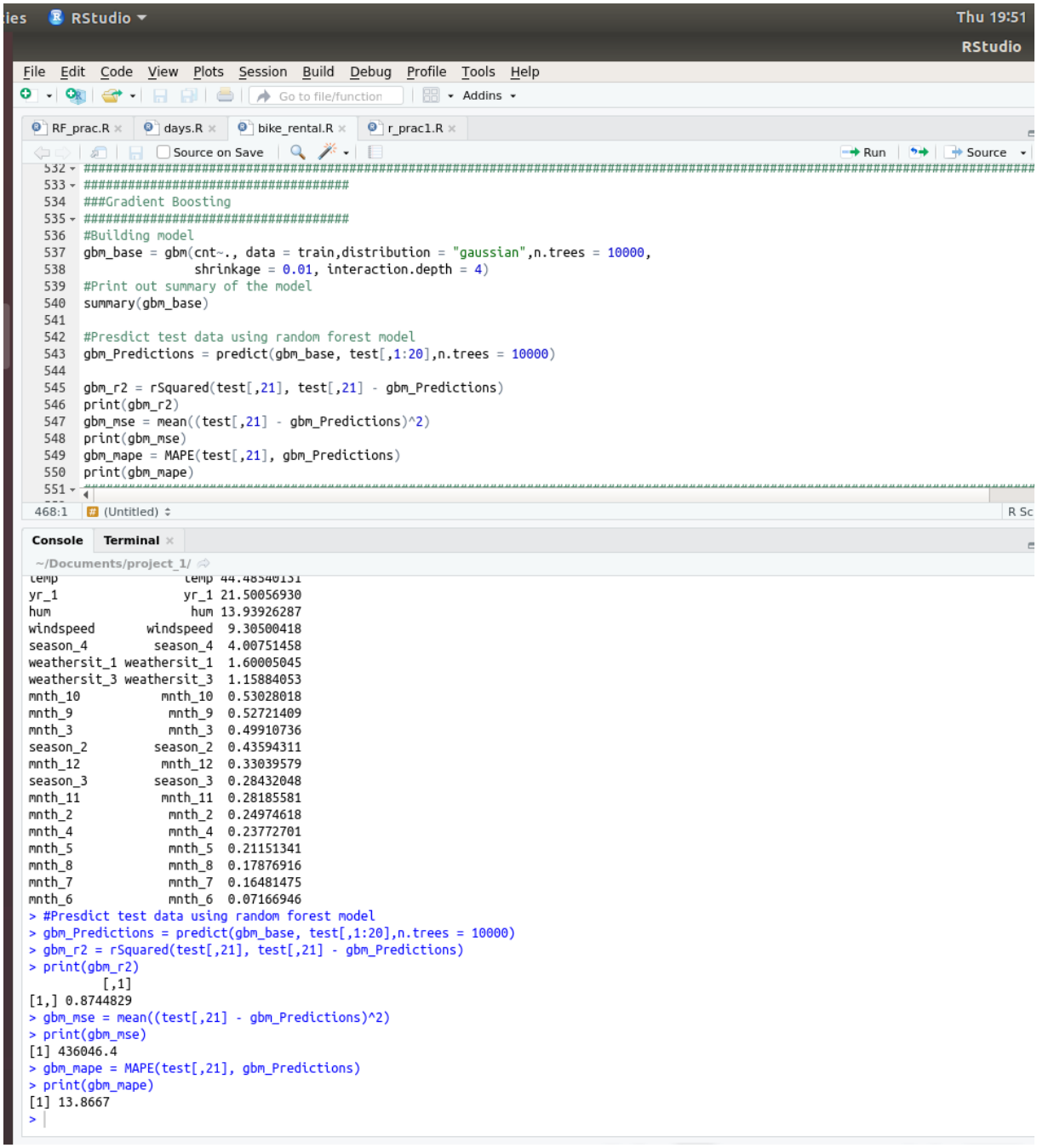


Figure B.27: A Gradient boosting model was modeled on to our train data and the model has an MAPE of 13.86%

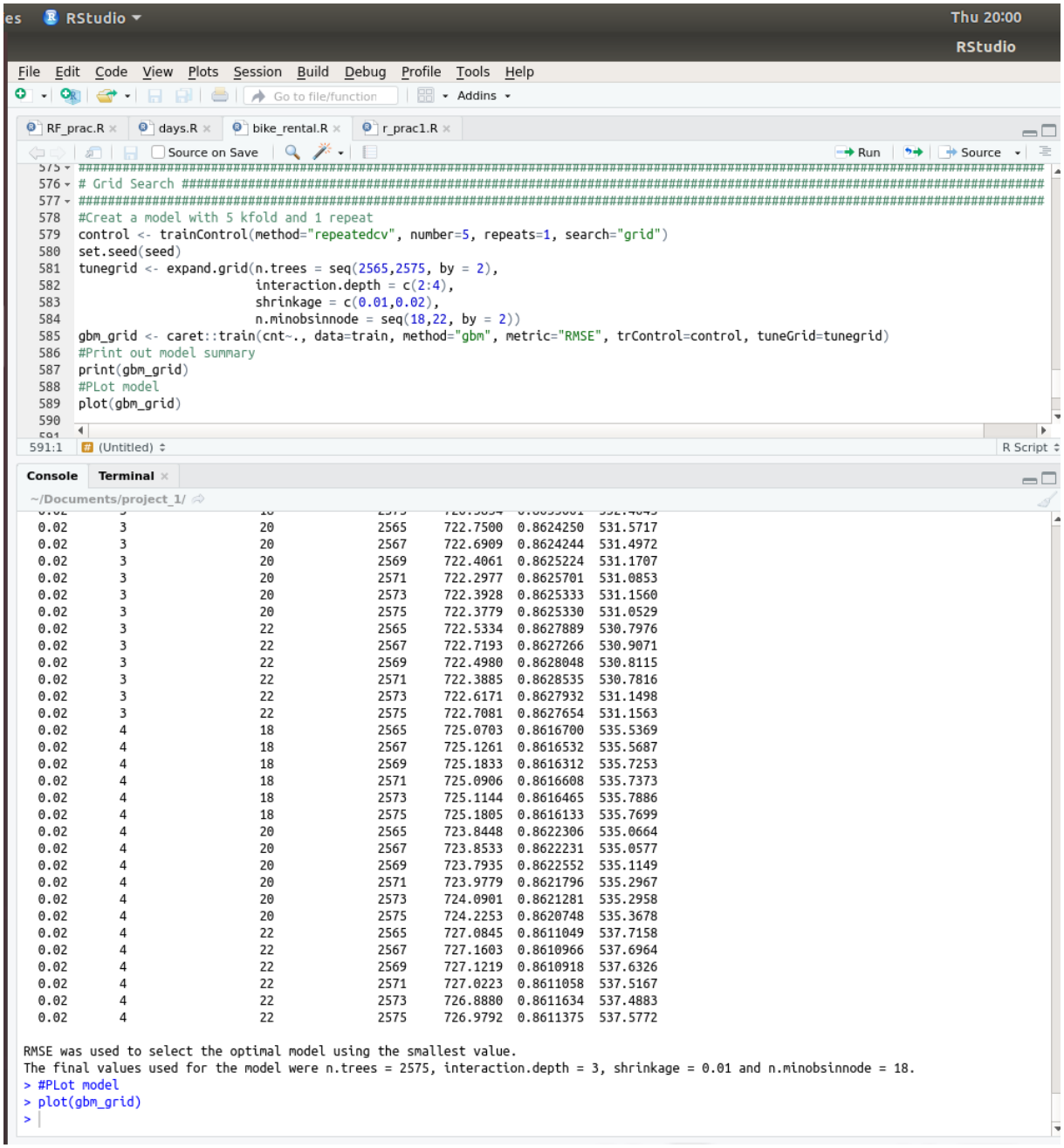


Figure B.28: Grid Search CV Gradient boosting modeling was performed on the data. Cross validation was conducted 5 folds on the data and the out put of the model is ploted.

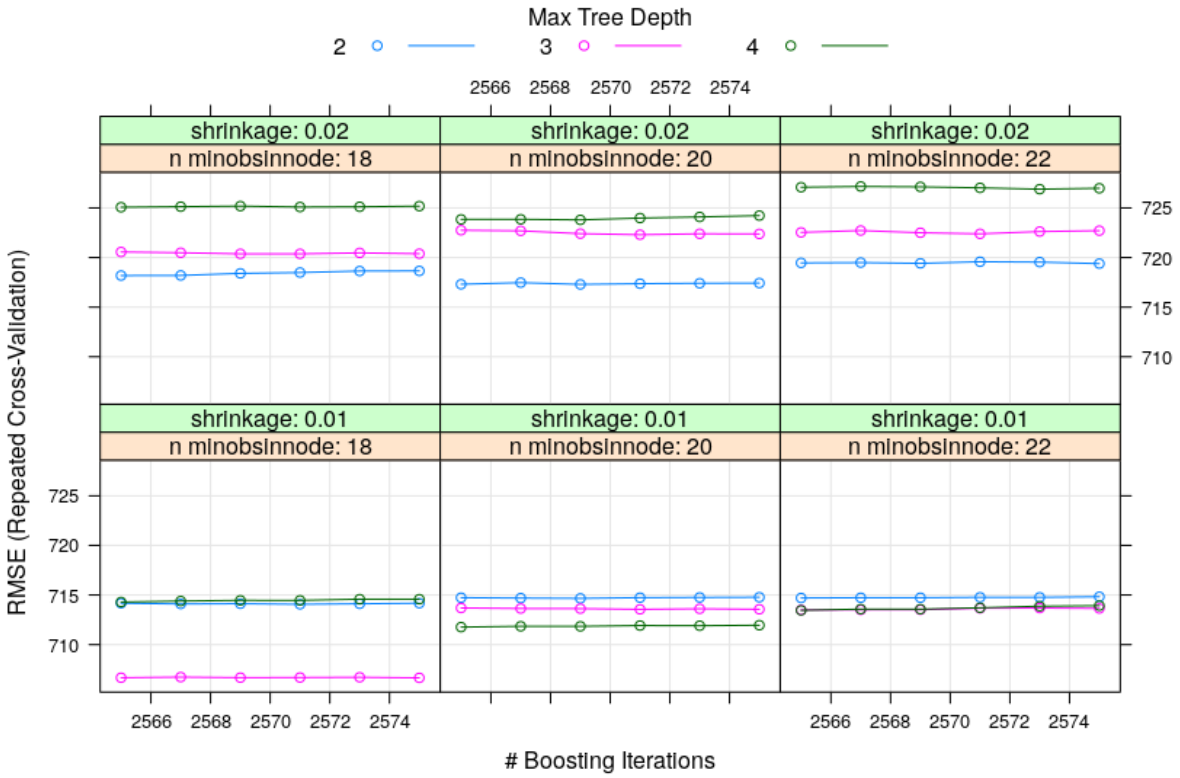


Figure B.29: Boosting Iteration Vs RMSE for repeated cross validation is plotted for every combination of the grid parameter. It can be noted that the least RMSE is achived when the shrinkage is 0.01 n.minobsinnode is 18 max tree depth. This is the output for figure B.28.

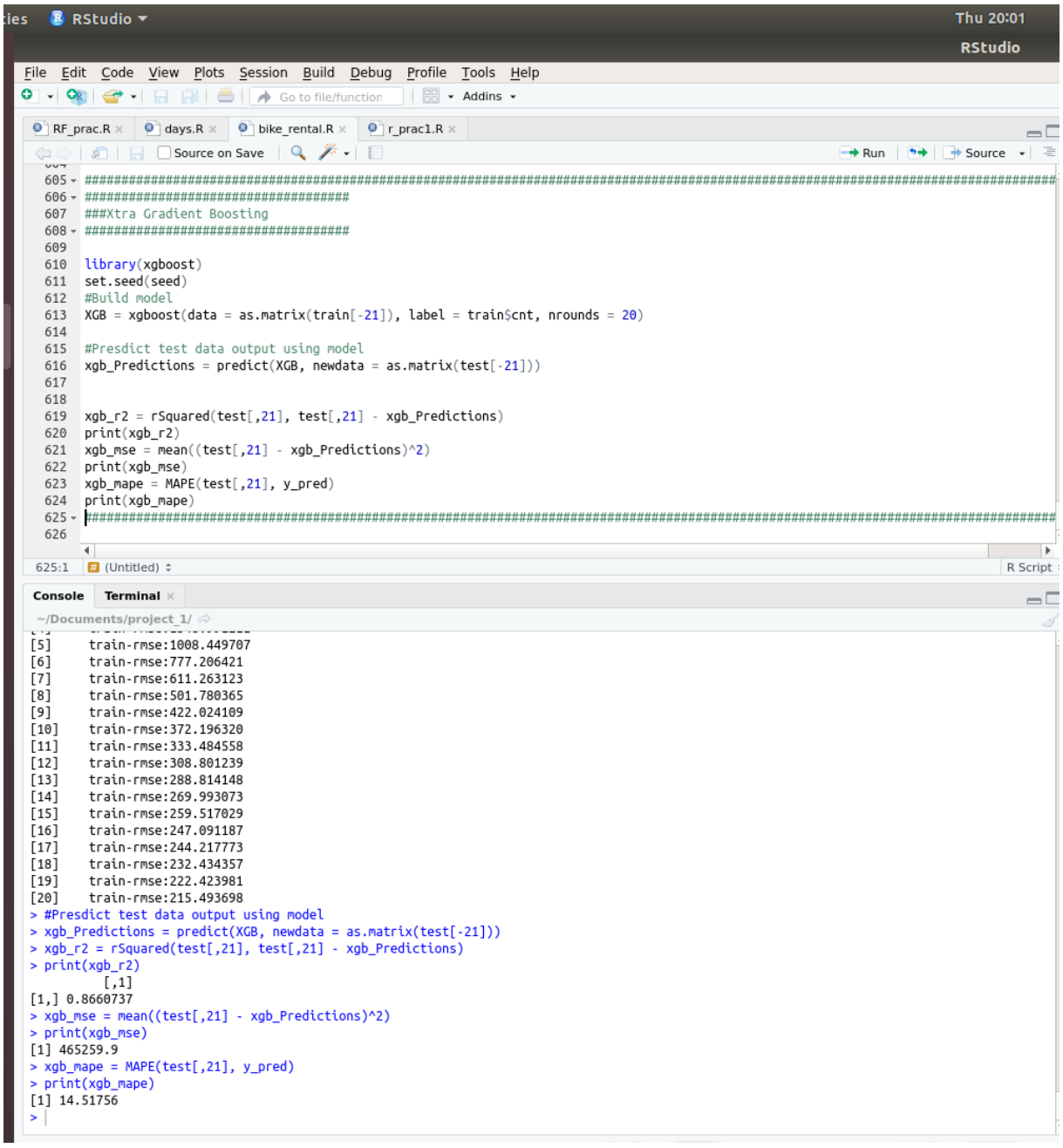


Figure B.30: A Extra Gradient boosting model was modeled on to our train data and the model has an MAPE of 14.51%