ChengM.DA5030.Project

Minxin Cheng

Project Overview

1. General dataset information

The dataset I choose is Bike Sharing Dataset [1] (UCI Machine Learning Repository, URL: http://archive.ics.uci.edu/ml/datasets/Bike+Sharing+Dataset). The dataset is collected from an auto bike rental system called Capital Bike Sharing.

[1] Fanaee-T, Hadi, and Gama, Joao, 'Event labeling combining ensemble detectors and background knowledge', Progress in Artificial Intelligence (2013): pp. 1-15, Springer Berlin Heidelberg.

The original dataset includes 17 columns:

- **instant** – continuous: record index
- **dteday** – interval: date
- **season** – categorical: season (1: winter; 2: spring; 3: summer; 4: fall)
- **yr** – categorical: year (2011 to 2012)
- **mnth** – interval: month (1 to 12)
- **hr** – interval: hour (0 to 23)
- **holiday** – categorical: if a day is holiday or not (1: yes; 0: no)
- **weekday** – categorical: the day of the week (0: Sun, 1-6: Mon - Sat)
- **workingday** – categorical: if a day is working day or not (1: yes; 0: no)
- **weathersit** – categorical: how was the weather of that day (1: clear, few clouds, partly cloudy; 2: mist + cloudy, mist+ broken clouds, mist+ broken clouds; 3: light snow, light rain + thunderstorm + Scattered clouds; 4: heavy rain + ice pallets + thunderstorm + mist,snow + fog)
- **temp** – numeric: normalized temperature in Celsius (min-max)
- **atemp** – numeric: normalized feeling temperature in Celsius (min-max)
- **hum** – numeric: normalized humidity (raw / 100 (max hum))
- **windspeed** – numeric: normalized wind speed (raw / 67 (max windspeed))
- **casual** – numeric: count of casual users
- **registered** – numeric: count of registered users
- **cnt** – numeric: count of total rental bikes including both casual and registered

2. Data Cleaning and transformaion

2.1 Cleaning

In this project, I removed: 1) column "Instant", which is row number 2) column "dteday", which can be represented by year and month 3) column "workingday", which can be represented by weekday

In this project, outliers in the numeric columns (temp, atemp, hum, windspeed, and count) are defined as 2.5 standard deviation from the mean value (z-score > 2.5), which ended up as about 6% of the cases were defined as outliers and were removed.

2.2 Transformation

1) Normalization: all the numeric columns (temp, atemp, hum, windspeed, and count) were kept as what they are in the original dataset after exploring of other possible transformations
2) Feature engineering: hour column was transformed to bins of each 6 hours, four bins were generated as categories 1-4; the amount of casual user and regisitered user were transformed as the percentage of casual user in that day, calculated as casual / (casual + registered).

2.3 Testing and identification

1) Shaperio-Wilk test was used to evaluate distributions of the numeric columns
2) prcomp() was used to identify principal component

3. Algorithms used in this project

1) Multiple linear regression:

- lm() function was used to train multiple linear regression model
- Stepwise backward elimation was used to select potential optimal predictors.
- regsubsets() was used to further select optimal predictors by adjusted-R2, Cp, and BIC
- Adjusted-R2, MAE, RMSE, and RSE were used to select the best lm model
- A cross-validation with 10 folders was also used to further compare the models based on RMSE

2) Neural network regression:

- hidden node was set as 1 given the limitation of computer capability

3) Regression tree:

- rpart() function was used to generate regression trees
- A for loop was used to evaluate different minsplit and maxdepth combination
- MAE and RMSE were used to evaluate the performance

4) Ensemble model based on above three models

- If any two models get a equal prediction in a new case, the final prediction will be this prediction
- Else it will be the prediction from the model with lowest RMSE
- MAE and RMSE were calculated to compare this ensemble model with each individual
- Cross-validation with 10 folders was used to further compare these 4 models based on RMSE

Below are my code for this final project.

```
#install.packages("googledrive")
#install.packages("dplyr")
#install.packages("ggplot2")
#install.packages("psych")
#install.packages("fastDummies")
#install.packages("rsample")
```

```
#install.packages("leaps")
#install.packages("neuralnet")
#install.packages("rpart")
#install.packages("rpart.plot")
library(gsheet)          # for downloading data file from google drive
library(dplyr)           # for data organization
library(ggplot2)         # for data visualization
library(psych)           # for creating pair-panels
library(fastDummies)     # for creating dummy codes
library(rsample)         # for spliting training data and testing data
library(caret)           # for creating folders for cross-validation
library(leaps)           # for generating multiple linear regression models
library(neuralnet)       # for creating neural network models
library(rpart)           # for creating regression tree models
library(rpart.plot)      # for regression tree visualization
```

Step 1. Load the data

1. Read in dataset

```
url <- "https://docs.google.com/spreadsheets/d/1Kp7PnvkEW1ad2kaxp3mJAXlriEDMcEMu0oP_-E85YPQ/edit?usp=sha
bike <- gsheet2tbl(url, sheetid = NULL)
```

Step 2. Get an over overview of the data

1. Check the data structure and get an overview of the dataset

```
# check if there is any missing values
any(is.na(bike))
```

```
## [1] FALSE
```

```
# check the structure of the dataset
str(bike)
```

```
## tibble [17,379 x 17] (S3: spec_tbl_df/tbl_df/tbl/data.frame)
##  $ instant   : num [1:17379] 1 2 3 4 5 6 7 8 9 10 ...
##  $ dteday    : Date[1:17379], format: "2011-01-01" "2011-01-01" ...
##  $ season    : num [1:17379] 1 1 1 1 1 1 1 1 1 1 ...
##  $ yr        : num [1:17379] 0 0 0 0 0 0 0 0 0 0 ...
##  $ mnth      : num [1:17379] 1 1 1 1 1 1 1 1 1 1 ...
##  $ hr        : num [1:17379] 0 1 2 3 4 5 6 7 8 9 ...
##  $ holiday   : num [1:17379] 0 0 0 0 0 0 0 0 0 0 ...
##  $ weekday   : num [1:17379] 6 6 6 6 6 6 6 6 6 6 ...
##  $ workingday: num [1:17379] 0 0 0 0 0 0 0 0 0 0 ...
##  $ weathersit: num [1:17379] 1 1 1 1 1 2 1 1 1 1 ...
```

```
##  $ temp      : num [1:17379] 0.24 0.22 0.22 0.24 0.24 0.24 0.22 0.2 0.24 0.32 ...
##  $ atemp     : num [1:17379] 0.288 0.273 0.273 0.288 0.288 ...
##  $ hum       : num [1:17379] 0.81 0.8 0.8 0.75 0.75 0.75 0.8 0.86 0.75 0.76 ...
##  $ windspeed : num [1:17379] 0 0 0 0 0 0.0896 0 0 0 0 ...
##  $ casual    : num [1:17379] 3 8 5 3 0 0 2 1 1 8 ...
##  $ registered: num [1:17379] 13 32 27 10 1 1 0 2 7 6 ...
##  $ cnt       : num [1:17379] 16 40 32 13 1 1 2 3 8 14 ...
##  - attr(*, "spec")=
##   .. cols(
##   ..   instant = col_double(),
##   ..   dteday = col_date(format = ""),
##   ..   season = col_double(),
##   ..   yr = col_double(),
##   ..   mnth = col_double(),
##   ..   hr = col_double(),
##   ..   holiday = col_double(),
##   ..   weekday = col_double(),
##   ..   workingday = col_double(),
##   ..   weathersit = col_double(),
##   ..   temp = col_double(),
##   ..   atemp = col_double(),
##   ..   hum = col_double(),
##   ..   windspeed = col_double(),
##   ..   casual = col_double(),
##   ..   registered = col_double(),
##   ..   cnt = col_double()
##   .. )
```

Step 3. Clean up the dataset

1. Remove the columns that will not be used for regression

```
# create a percentage column of casual/registered
bike <- bike %>% mutate(percentage = casual / (casual + registered))
# 1) remove dteday column, which can be represented by season & year & month;
# 2) remove casual and registered columns,
# which can be represented by cnt & percentage;
# 3) remove workingday column, which can be represented by weekday
bike <- bike[ , -c(2, 9, 15:16)]
# randomize the rows
bike <- bike[sample(nrow(bike)), ]
```

2. Transfer back the dataset to its original

Some of the numeric columns of the data set has already been transferred (listed as below), for exploration, I will transfer them back first.

The transferred columns in the dataset are:

1) temp column was min-max normalized, calculated as (t - tmin)/(tmax - tmin), tmin = -8, tmax = +39

2) atemp column was min-max normalized, calculated as (at - atmin)/(atmax - atmin), atmin = -16, atmax = +50
3) hum column was normalized by dividing by its maximum value, calculated as hum / 100 (max value of hum)
4) windspeed column was normalized by dividing by its maximum value, calculated as windspeed / 67 (max value of windspeed)

```
# create a copy for transferring back
bike.raw <- bike
# calculate the original temperature
bike.raw$temp <- bike.raw$temp * (39 - (-8)) + (-8)
# calculate the original feeling temperature
bike.raw$atemp <- bike.raw$atemp * (50 - (-16)) + (-16)
# calculate the original humidity
bike.raw$hum <- bike.raw$hum * 100
# calculate the original wind speed
bike.raw$windspeed <- bike.raw$windspeed * 67
```

3. Check the outliers

I will first convert all the numeric columns (tempeture, feeling tempeture, humidity, windspeed, cnt, and percentage) to z-scores, then filter all the data that is 2.5 standard deviations away from the mean.

```
# select all numeric columns from the original dataset
bike.checkoutlier <- bike.raw[ , c(1, 9:14)]
# convert these numeric columns to z-scores
bike.checkoutlier[ , c(2:7)] <-
  as.data.frame(scale(bike.checkoutlier[ , c(2:7)]))
# filter out all outliers (as long as one z-score > 2.5 in a row)
bike.outlier <-
  as.data.frame(bike.checkoutlier[apply(bike.checkoutlier[ , c(2:7)],
                                        1, function(x) any(abs(x) > 2.5)), ])
# check how many rows were considered as outlier
nrow(bike.outlier)
```

```
## [1] 1035
```

```
nrow(bike.outlier) / nrow(bike.raw)
```

```
## [1] 0.05955463
```

From the detection above, about 6% of the days data was considered as outliers, it is an acceptable proporation. I will removem them in the next step.

4. Remove outliers

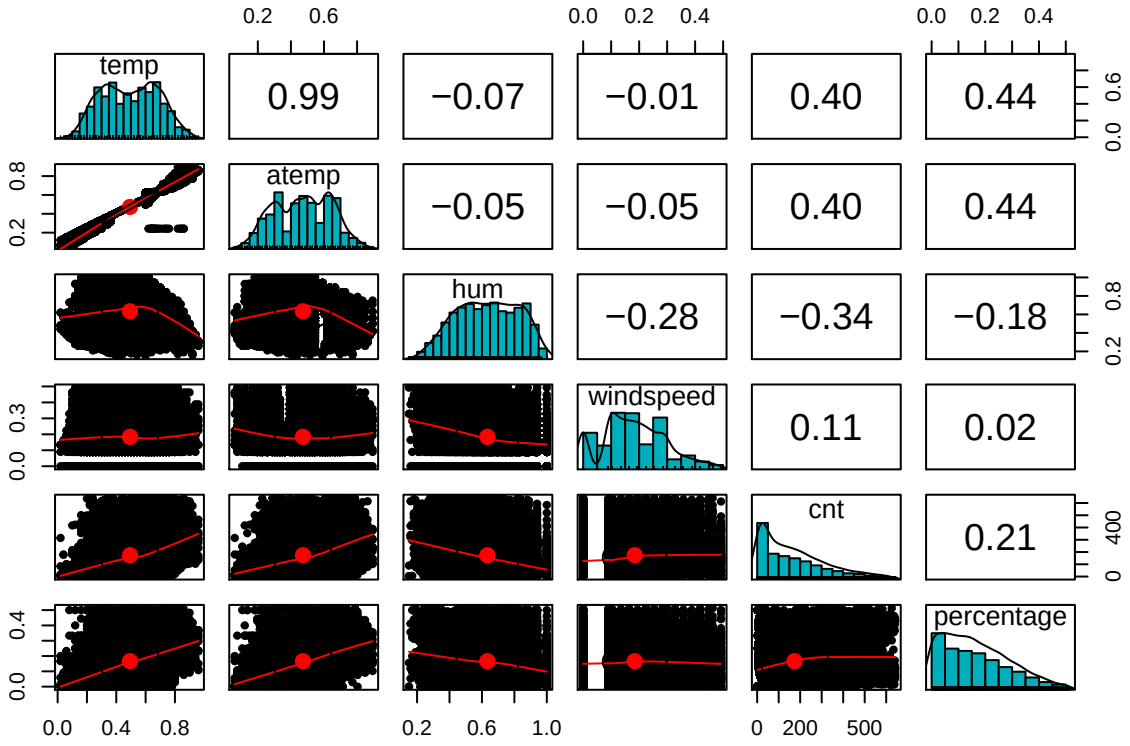
```
# get the row numbers of outliers
bike.outlier <- bike.outlier %>% select(instant)
# remove the outliers from the dataset by row numbers
```

```r
bike.raw.clean <- bike.raw %>%
  anti_join(bike.outlier, by = "instant") %>%
  select(-instant)
# remove the outliers from the original dataset by row numbers
bike.ori.clean <- bike %>%
  anti_join(bike.outlier, by = "instant") %>%
  select(-instant)
```

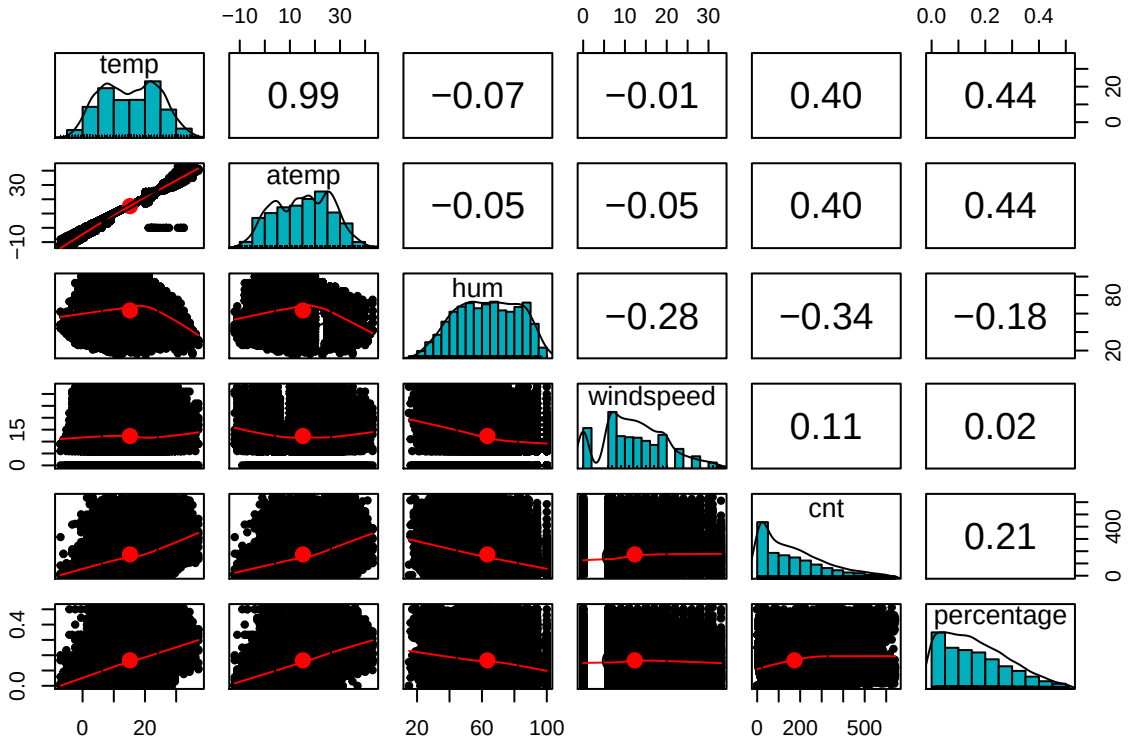
5. Check the distributions and correlations of the dataset

I will plot pair-panels to get the overall distributions and correlations of the dataset. I will make plots for both the outlier-removed original dataset and the outlier-removed raw data I calculated back.

```r
# plot the pair-panels for the outlier-removed original dataset
pairs.panels(bike.ori.clean[, c(8:13)], method = "pearson",
             hist.col = "#00AFBB", density = TRUE, ellipses = TRUE)
```




```r
# plot the pair-panels for the outlier-removed raw dataset
pairs.panels(bike.raw.clean[, c(8:13)], method = "pearson",
             hist.col = "#00AFBB", density = TRUE, ellipses = TRUE)
```



From the panels we can see that tempeature and feeling temperature have strong correlations with the proporation of casual users and the total count as well. Humidity also has relatively strong correlations with the total amount of users. Comparing to the raw data, the original dataset, which did transformation for some columns doesn't seem improve the distribution. I will quickly run a Shapiro test next to check the normality. Since I have already randomized the dataset, I will only select the first 5000 rows.

```
shapiro.test(bike.ori.clean$temp[1:5000])
```

```
##
##  Shapiro-Wilk normality test
##
## data:  bike.ori.clean$temp[1:5000]
## W = 0.97922, p-value < 2.2e-16
```

```
shapiro.test(bike.ori.clean$atemp[1:5000])
```

```
##
##  Shapiro-Wilk normality test
##
## data:  bike.ori.clean$atemp[1:5000]
## W = 0.98143, p-value < 2.2e-16
```

```
shapiro.test(bike.ori.clean$hum[1:5000])
```

```
##
```

```
##  Shapiro-Wilk normality test
##
## data:  bike.ori.clean$hum[1:5000]
## W = 0.97707, p-value < 2.2e-16
```

```
shapiro.test(bike.ori.clean$windspeed[1:5000])
```

```
##
##  Shapiro-Wilk normality test
##
## data:  bike.ori.clean$windspeed[1:5000]
## W = 0.96477, p-value < 2.2e-16
```

As we see here, also from the Shapiro test results, these few transformed columns don't seem to be normally distributed, I will explore if there is any other way to transform the data in the next step.

6. Explore different transformations for the datasetd

Here I will try z-score, squared, and square root transformation for temperature, feeling temperature, humidity, and windspeed see if I can improve them a bit.

```
# try z-score transformation for temp
z.temp <- scale(bike.ori.clean$temp)
# un-commend to check the histogram
# hist(z.temp)
# run Shapiro test
shapiro.test(z.temp[1:5000])
```

```
##
##  Shapiro-Wilk normality test
##
## data:  z.temp[1:5000]
## W = 0.97922, p-value < 2.2e-16
```

```
# try squared the temperature
squared.temp <- (bike.ori.clean$temp) ^2
# un-commend to check the histogram
# hist(squared.temp)
# run Shapiro test
shapiro.test(squared.temp[1:5000])
```

```
##
##  Shapiro-Wilk normality test
##
## data:  squared.temp[1:5000]
## W = 0.94188, p-value < 2.2e-16
```

```
# # try z-score transformation for temp
z.atemp <- scale(bike.ori.clean$atemp)
# un-commend to check the histogram
# hist(z.atemp)
# run Shapiro test
shapiro.test(z.atemp[1:5000])
```

```
##
##  Shapiro-Wilk normality test
##
## data:  z.atemp[1:5000]
## W = 0.98143, p-value < 2.2e-16
```

```
squared.atemp <- (bike.ori.clean$atemp) ^2
#hist(squared.atemp)
shapiro.test(squared.atemp[1:5000])
```

```
##
##  Shapiro-Wilk normality test
##
## data:  squared.atemp[1:5000]
## W = 0.95452, p-value < 2.2e-16
```

```
z.hum <- scale(bike.ori.clean$hum)
#hist(z.hum)
shapiro.test(z.hum [1:5000])
```

```
##
##  Shapiro-Wilk normality test
##
## data:  z.hum[1:5000]
## W = 0.97707, p-value < 2.2e-16
```

```
squared.hum  <- (bike.ori.clean$hum) ^2
#hist(squared.hum)
shapiro.test(squared.hum[1:5000])
```

```
##
##  Shapiro-Wilk normality test
##
## data:  squared.hum[1:5000]
## W = 0.95904, p-value < 2.2e-16
```

```
sqrt.hum <- sqrt(bike.ori.clean$hum)
#hist(sqrt.hum)
shapiro.test(sqrt.hum[1:5000])
```

```
##
##  Shapiro-Wilk normality test
##
## data:  sqrt.hum[1:5000]
## W = 0.97076, p-value < 2.2e-16
```

```
log.hum <- log(bike.ori.clean$hum)
#hist(log.hum)
shapiro.test(log.hum[1:5000])
```

```
##
##  Shapiro-Wilk normality test
##
## data:  log.hum[1:5000]
## W = 0.94929, p-value < 2.2e-16
```

```
z.windspeed <- scale(bike.ori.clean$windspeed )
#hist(z.windspeed)
shapiro.test(z.windspeed[1:5000])
```

```
##
##  Shapiro-Wilk normality test
##
## data:  z.windspeed[1:5000]
## W = 0.96477, p-value < 2.2e-16
```

```
squared.windspeed <- (bike.ori.clean$windspeed) ^2
#hist(squared.windspeed)
shapiro.test(squared.windspeed[1:5000])
```

```
##
##  Shapiro-Wilk normality test
##
## data:  squared.windspeed[1:5000]
## W = 0.82671, p-value < 2.2e-16
```

```
sqrt.windspeed <- sqrt(bike.ori.clean$windspeed)
#hist(sqrt.windspeed)
shapiro.test(sqrt.windspeed[1:5000])
```

```
##
##  Shapiro-Wilk normality test
##
## data:  sqrt.windspeed[1:5000]
## W = 0.87598, p-value < 2.2e-16
```

```
log.windspeed <- log(bike.ori.clean$windspeed )
#hist(log.windspeed)
shapiro.test(log.windspeed[1:5000])
```

```
##
##  Shapiro-Wilk normality test
##
## data:  log.windspeed[1:5000]
## W = NaN, p-value = NA
```

From all the exploration above, it seems nothing has been improved from min-max normalization and divided by max value methods that the dataset already used. Therefore, I will keep using the original data for temp, atemp, hum, and windspeed columns. Next I will try what could be good for casual and register columns.

```r
# create a function for min-max transform
minmax <- function(x){
  return((x - min(x)) / (max(x) - min(x)))
}

# try min-max to percentage column
minmax.percentage <- minmax(bike.ori.clean$percentage)
# check the distribution
# hist(minmax.percentage)
# run Shapiro test
shapiro.test(minmax.percentage[1:5000])
```

```
##
##  Shapiro-Wilk normality test
##
## data:  minmax.percentage[1:5000]
## W = 0.94563, p-value < 2.2e-16
```

```r
# try z-score for percentage column
z.percentage <- scale(bike.ori.clean$percentage)
# check the distribution
# hist(z.percentage)
# run Shapiro test
shapiro.test(z.percentage[1:5000])
```

```
##
##  Shapiro-Wilk normality test
##
## data:  z.percentage[1:5000]
## W = 0.94563, p-value < 2.2e-16
```

```r
# try z-score for percentage column
squared.percentage <- (bike.ori.clean$percentage) ^2
# hist(squared.percentage)
shapiro.test(squared.percentage[1:5000])
```

```
##
##  Shapiro-Wilk normality test
##
## data:  squared.percentage[1:5000]
## W = 0.77876, p-value < 2.2e-16
```

```r
# try z-score for percentage column
sqrt.percentage <- sqrt(bike.ori.clean$percentage)
# hist(sqrt.percentage)
shapiro.test(sqrt.percentage[1:5000])
```

```
##
##  Shapiro-Wilk normality test
##
## data:  sqrt.percentage[1:5000]
## W = 0.96614, p-value < 2.2e-16
```

```r
# try min-max to cnt column
minmax.cnt <- minmax(bike.ori.clean$cnt)
# check the distribution
# hist(minmax.cnt)
# run Shapiro test
shapiro.test(minmax.cnt[1:5000])
```

```
##
##  Shapiro-Wilk normality test
##
## data:  minmax.cnt[1:5000]
## W = 0.89858, p-value < 2.2e-16
```

```r
# try z-score for cnt column
z.cnt <- scale(bike.ori.clean$cnt)
# check the distribution
# hist(z.cnt)
# run Shapiro test
shapiro.test(z.cnt[1:5000])
```

```
##
##  Shapiro-Wilk normality test
##
## data:  z.cnt[1:5000]
## W = 0.89858, p-value < 2.2e-16
```

```r
# try z-score for cnt column
squared.cnt <- (bike.ori.clean$cnt) ^2
# hist(squared.cnt)
shapiro.test(squared.cnt[1:5000])
```

```
##
##  Shapiro-Wilk normality test
##
## data:  squared.cnt[1:5000]
## W = 0.69126, p-value < 2.2e-16
```

```r
# try z-score for cnt column
sqrt.cnt <- sqrt(bike.ori.clean$cnt)
# hist(sqrt.cnt)
shapiro.test(sqrt.cnt[1:5000])
```

```
##
##  Shapiro-Wilk normality test
##
## data:  sqrt.cnt[1:5000]
## W = 0.97173, p-value < 2.2e-16
```

From the Shapiro test results and the distribution figures, no transformation is good enough, I will not transform the cnt column and percentage column.

7. Identify principal components (PCA)

Here I will perform a PCA identifier to check the PCA of the dataset.

```
# perform the PCA identifier for only the numeric columns
bike.pca <- prcomp(bike.ori.clean[ , 8:13], center = TRUE, scale = TRUE)
bike.pca
```

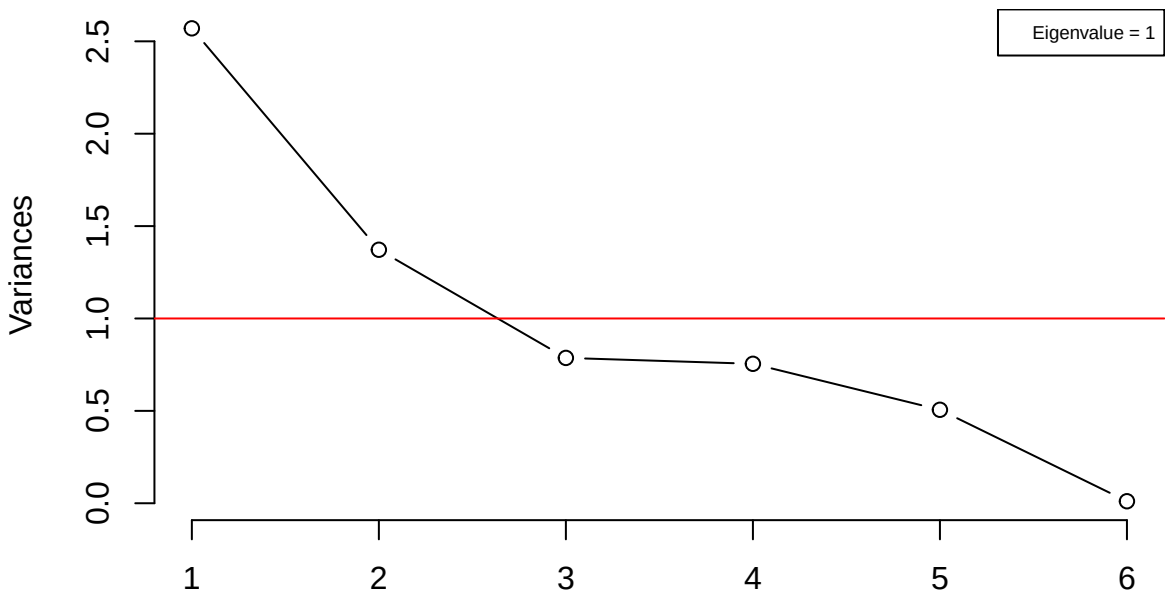
```
## Standard deviations (1, .., p=6):
## [1] 1.6031774 1.1712757 0.8869168 0.8686982 0.7111956 0.1042988
##
## Rotation (n x k) = (6 x 6):
##                   PC1          PC2         PC3        PC4        PC5
## temp       -0.5771331  0.210313034 -0.09616817 -0.2204548  0.2587779
## atemp      -0.5746929  0.236675412 -0.07151177 -0.2062087  0.2543828
## hum         0.1785778  0.656227019 -0.27300147 -0.3722857 -0.5694874
## windspeed  -0.0420120 -0.615321897 -0.64239185 -0.4350898 -0.1301113
## cnt        -0.3839484 -0.300815325  0.59125806 -0.2315391 -0.5990698
## percentage -0.3944289  0.002894761 -0.38581675  0.7262007 -0.4101145
##                    PC6
## temp        0.705599870
## atemp      -0.708072250
## hum         0.005803032
## windspeed  -0.026731400
## cnt         0.003569173
## percentage  0.001237235
```

```
# summary the PCA identifier
summary(bike.pca)
```

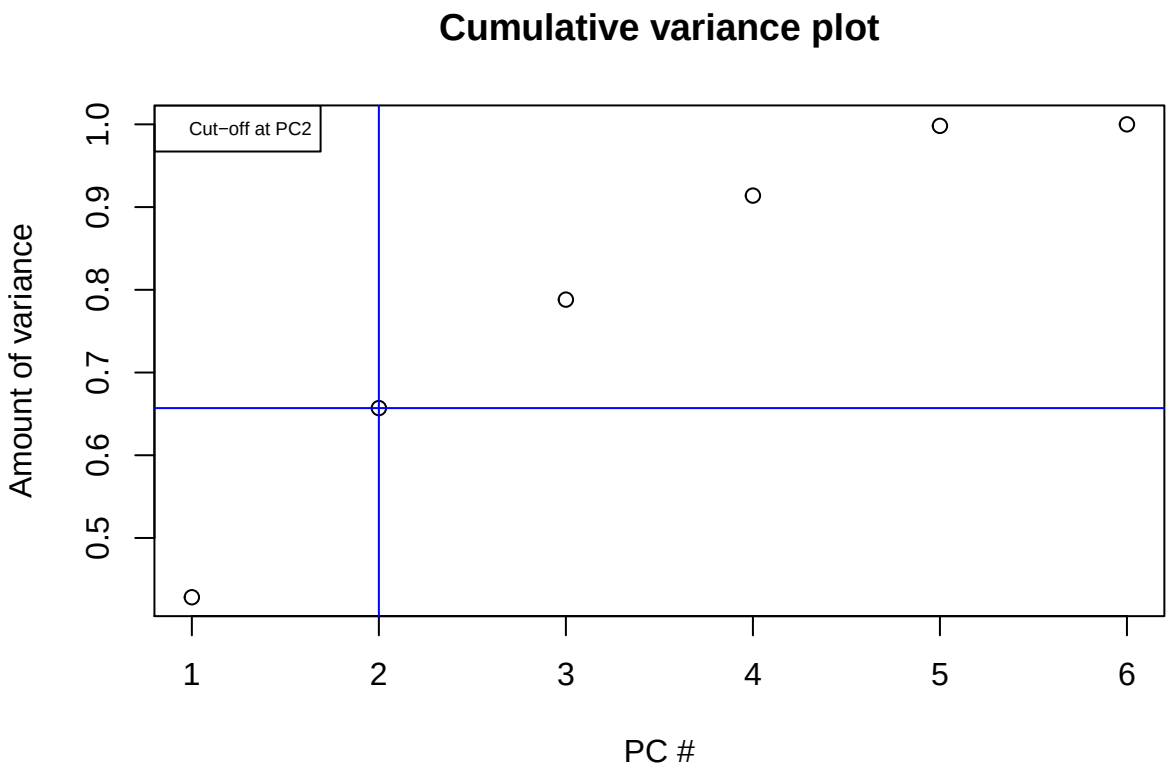
```
## Importance of components:
##                           PC1    PC2    PC3    PC4    PC5     PC6
## Standard deviation     1.6032 1.1713 0.8869 0.8687 0.7112 0.10430
## Proportion of Variance 0.4284 0.2286 0.1311 0.1258 0.0843 0.00181
## Cumulative Proportion  0.4284 0.6570 0.7881 0.9139 0.9982 1.00000
```

In this table, standard deviation are the eigenvalues since the data has been scaled; proportion of variance is the amount variance the component accounts for in the data that PC1 has already accounted for about 43% of total variance; cumulative proportion is the accumulated amount of explained variance. Below I am trying to visualize the relationship between the PCs and variance.

```
# plot the relationship between variance and each pc
screeplot(bike.pca, type = "l", npcs = 6, main = "Plot of all PCs")
abline(h = 1, col ="red")
legend("topright", legend = c("Eigenvalue = 1"), col = c("red"), cex = 0.6)
```




```r
# plot the relationship between cumulative variance and each pc
cumpro <- cumsum(bike.pca$sdev ^ 2 / sum(bike.pca$sdev ^ 2))
plot(cumpro[0:6], xlab = "PC #", ylab = "Amount of variance",
     main = "Cumulative variance plot")
abline(v = 2, col = "blue")
abline(h = 0.6570, col = "blue")
legend("topleft", legend = c("Cut-off at PC2"), col = "blue", cex = 0.6)
```



From the plots above, we can see the first 2 components has an eigenvalue greater than 1 and they explained about 65% of variance, which is fairly well. Since the data also has many categorical variables, it is not necessary to furtherly use principal component regression (PCR).

Step 4. Prepare the dataset

1. Create dummy code for the dataset

```
# create a copy for dummy coding
bike.dummy <- bike.ori.clean
# group the hour column to bins:
# 0-5: bin 1
# 6-11: bin 2
# 12-17: bin 3
# 18-23: bin 4
bike.dummy <- bike.dummy %>%
  mutate(hrbin = case_when(hr <= 5 ~ 1,
                           ((hr >= 6) & (hr <= 11)) ~ 2,
                           ((hr >= 12) & (hr <= 17)) ~ 3,
                           ((hr >= 18) & (hr <= 23)) ~ 4)) %>% select(-hr)
# convert season, month, holiday, weekday, weathersit, and
# the new hour bin column to factor
bike.dummy[, c(1, 3:6, 13)] <- lapply(bike.dummy[, c(1, 3:6, 13)],
                                      as.factor)
```

```r
# quick summary of the data to get an overview
summary(bike.dummy)
```

```
##  season         yr              mnth      holiday   weekday  weathersit
##  1:4004   Min.   :0.0000   12     :1440   0:15878   0:2371   1:10648
##  2:4100   1st Qu.:0.0000   11     :1390   1:  466   1:2320   2: 4357
##  3:4197   Median :0.0000   7      :1389             2:2307   3: 1336
##  4:4043   Mean   :0.4914   5      :1388             3:2335   4:    3
##           3rd Qu.:1.0000   8      :1386             4:2302
##           Max.   :1.0000   1      :1373             5:2353
##                            (Other):7978             6:2356
##       temp             atemp              hum          windspeed
##  Min.   :0.0200   Min.   :0.0606   Min.   :0.1500   Min.   :0.0000
##  1st Qu.:0.3400   1st Qu.:0.3333   1st Qu.:0.4800   1st Qu.:0.1045
##  Median :0.5000   Median :0.4848   Median :0.6400   Median :0.1642
##  Mean   :0.4934   Mean   :0.4728   Mean   :0.6337   Mean   :0.1843
##  3rd Qu.:0.6400   3rd Qu.:0.6212   3rd Qu.:0.7900   3rd Qu.:0.2537
##  Max.   :0.9600   Max.   :0.8939   Max.   :1.0000   Max.   :0.4925
##
##       cnt           percentage       hrbin
##  Min.   :  1.0   Min.   :0.00000   1:4093
##  1st Qu.: 39.0   1st Qu.:0.06278   2:4161
##  Median :137.0   Median :0.14706   3:3948
##  Mean   :173.6   Mean   :0.16635   4:4142
##  3rd Qu.:266.0   3rd Qu.:0.25000
##  Max.   :642.0   Max.   :0.51351
##
```

```r
# dummy code all the categorical columns
bike.dummy <- dummy_cols(bike.dummy,
                         select_columns = c("season", "mnth", "hrbin",
                                            "holiday", "weekday", "weathersit"),
                         remove_first_dummy = TRUE)
# remove the original categorical columns
bike.dummy <- bike.dummy[ , -c(1, 3:6, 13)]
# check the structure
str(bike.dummy)
```

```
## tibble [16,344 x 34] (S3: tbl_df/tbl/data.frame)
##  $ yr          : num [1:16344] 0 0 1 0 1 1 0 0 1 1 ...
##  $ temp        : num [1:16344] 0.62 0.34 0.36 0.64 0.42 0.26 0.16 0.5 0.9 0.32 ...
##  $ atemp       : num [1:16344] 0.561 0.333 0.333 0.576 0.424 ...
##  $ hum         : num [1:16344] 0.88 0.61 0.43 0.83 0.5 0.35 0.55 0.88 0.25 0.66 ...
##  $ windspeed   : num [1:16344] 0.0896 0.1343 0.3881 0.1045 0.2985 ...
##  $ cnt         : num [1:16344] 104 44 61 93 78 134 27 2 283 593 ...
##  $ percentage  : num [1:16344] 0.1923 0.25 0.082 0.2043 0.0641 ...
##  $ season_2    : int [1:16344] 0 0 0 1 0 0 0 0 0 1 ...
##  $ season_3    : int [1:16344] 0 0 0 0 0 0 0 0 1 0 ...
##  $ season_4    : int [1:16344] 1 0 0 0 0 0 0 1 0 0 ...
##  $ mnth_2      : int [1:16344] 0 0 1 0 0 0 0 0 0 0 ...
##  $ mnth_3      : int [1:16344] 0 0 0 0 1 0 0 0 0 1 ...
##  $ mnth_4      : int [1:16344] 0 0 0 0 0 0 0 0 0 0 ...
##  $ mnth_5      : int [1:16344] 0 0 0 0 0 0 0 0 0 0 ...
```

```
##  $ mnth_6      : int [1:16344] 0 0 0 1 0 0 0 0 0 0 ...
##  $ mnth_7      : int [1:16344] 0 0 0 0 0 0 0 0 1 0 ...
##  $ mnth_8      : int [1:16344] 0 0 0 0 0 0 0 0 0 0 ...
##  $ mnth_9      : int [1:16344] 1 0 0 0 0 0 0 0 0 0 ...
##  $ mnth_10     : int [1:16344] 0 0 0 0 0 0 0 0 0 0 ...
##  $ mnth_11     : int [1:16344] 0 0 0 0 0 0 0 1 0 0 ...
##  $ mnth_12     : int [1:16344] 0 1 0 0 0 0 0 0 0 0 ...
##  $ hrbin_2     : int [1:16344] 0 0 0 0 0 0 1 0 0 1 ...
##  $ hrbin_3     : int [1:16344] 0 1 0 0 1 1 0 0 1 0 ...
##  $ hrbin_4     : int [1:16344] 1 0 0 0 0 0 0 0 0 0 ...
##  $ holiday_1   : int [1:16344] 0 0 0 0 0 0 0 0 0 0 ...
##  $ weekday_1   : int [1:16344] 0 0 0 0 0 1 0 0 0 0 ...
##  $ weekday_2   : int [1:16344] 0 0 0 0 0 0 1 1 1 0 ...
##  $ weekday_3   : int [1:16344] 0 0 0 0 0 0 0 0 0 0 ...
##  $ weekday_4   : int [1:16344] 0 0 0 0 0 0 0 0 0 0 ...
##  $ weekday_5   : int [1:16344] 0 0 0 0 1 0 0 0 0 1 ...
##  $ weekday_6   : int [1:16344] 0 0 1 0 0 0 0 0 0 0 ...
##  $ weathersit_2: int [1:16344] 0 0 0 0 1 0 1 1 0 0 ...
##  $ weathersit_3: int [1:16344] 0 0 0 0 0 0 0 0 0 0 ...
##  $ weathersit_4: int [1:16344] 0 0 0 0 0 0 0 0 0 0 ...
##  - attr(*, ".internal.selfref")=<externalptr>
```

2. Split the dataset for training and testing

```
# split the dataset to 80%/20%
set.seed(123)
train.sample <- initial_split(bike.dummy, prop = 0.8, strata = "cnt")
bike.training <- training(train.sample)
bike.testing <- testing(train.sample)
# make a copy of training data and testing data for multiple linear regression
bike.training.lm <- training(train.sample)
bike.testing.lm <- testing(train.sample)
# make a copy of training data and testing data for neural network
bike.training.ann <- training(train.sample)
bike.testing.ann <- testing(train.sample)
# make a copy of training data and testing data for regression tree
bike.training.tree <- training(train.sample)
bike.testing.tree <- testing(train.sample)
```

Step 5. Multiple linear regression

1. Train the first model

I will first train the multiple linear regression with all the columns.

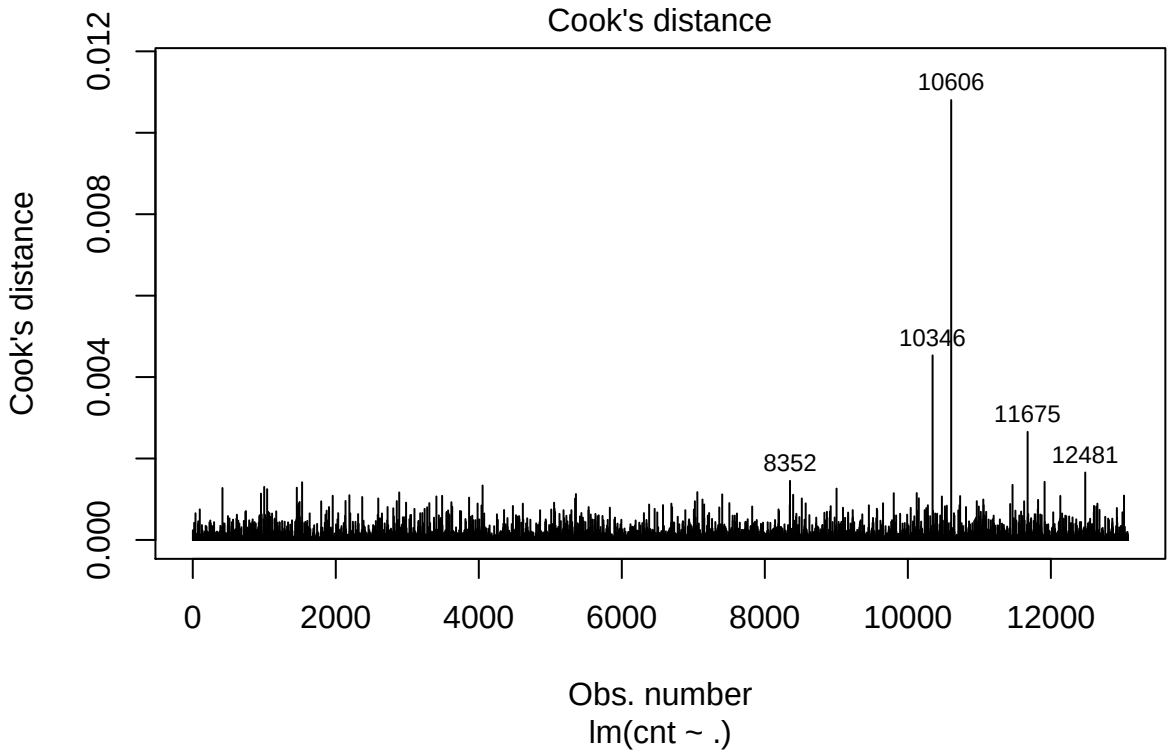
```
set.seed(123)
# train the first model
bike.model.lm1 <- lm(cnt ~ ., data = bike.training.lm)
# get an overview of the model
summary(bike.model.lm1)
```

```
##
## Call:
## lm(formula = cnt ~ ., data = bike.training.lm)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -267.17  -73.50  -20.79   53.59  448.05
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)   -57.0106     7.1438  -7.980 1.58e-15 ***
## yr             56.4271     1.9063  29.600  < 2e-16 ***
## temp          165.0641    37.0131   4.460 8.28e-06 ***
## atemp         144.1490    38.6316   3.731 0.000191 ***
## hum          -125.1455     6.7104 -18.650  < 2e-16 ***
## windspeed       6.8693     9.1171   0.753 0.451192
## percentage    -63.1501    10.2089  -6.186 6.36e-10 ***
## season_2       31.5450     5.9561   5.296 1.20e-07 ***
## season_3       34.0225     7.0022   4.859 1.19e-06 ***
## season_4       64.7896     5.8852  11.009  < 2e-16 ***
## mnth_2          1.6737     4.7298   0.354 0.723447
## mnth_3          0.3582     5.3133   0.067 0.946256
## mnth_4         -9.8952     7.9344  -1.247 0.212370
## mnth_5          4.0630     8.4699   0.480 0.631449
## mnth_6        -26.7444     8.7010  -3.074 0.002119 **
## mnth_7        -54.3532     9.7741  -5.561 2.74e-08 ***
## mnth_8        -34.2032     9.5402  -3.585 0.000338 ***
## mnth_9         -8.6781     8.4957  -1.021 0.307052
## mnth_10       -12.0146     7.8141  -1.538 0.124182
## mnth_11       -14.4898     7.4940  -1.934 0.053195 .
## mnth_12        -9.5434     5.9752  -1.597 0.110253
## hrbin_2       158.6940     2.6775  59.270  < 2e-16 ***
## hrbin_3       197.7113     3.2467  60.896  < 2e-16 ***
## hrbin_4       157.9682     2.7975  56.467  < 2e-16 ***
## holiday_1     -12.1677     5.9276  -2.053 0.040119 *
## weekday_1      -5.3766     3.7448  -1.436 0.151095
## weekday_2      -8.9816     3.7300  -2.408 0.016055 *
## weekday_3      -8.0258     3.7688  -2.130 0.033230 *
## weekday_4      -3.9182     3.7487  -1.045 0.295942
## weekday_5       8.5201     3.6508   2.334 0.019623 *
## weekday_6      12.1776     3.4806   3.499 0.000469 ***
## weathersit_2   -3.5708     2.3150  -1.542 0.122984
## weathersit_3  -41.5816     3.9691 -10.476  < 2e-16 ***
## weathersit_4   29.2347    61.9714   0.472 0.637117
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 107.1 on 13044 degrees of freedom
## Multiple R-squared:  0.5227, Adjusted R-squared:  0.5215
## F-statistic: 432.9 on 33 and 13044 DF,  p-value: < 2.2e-16
```

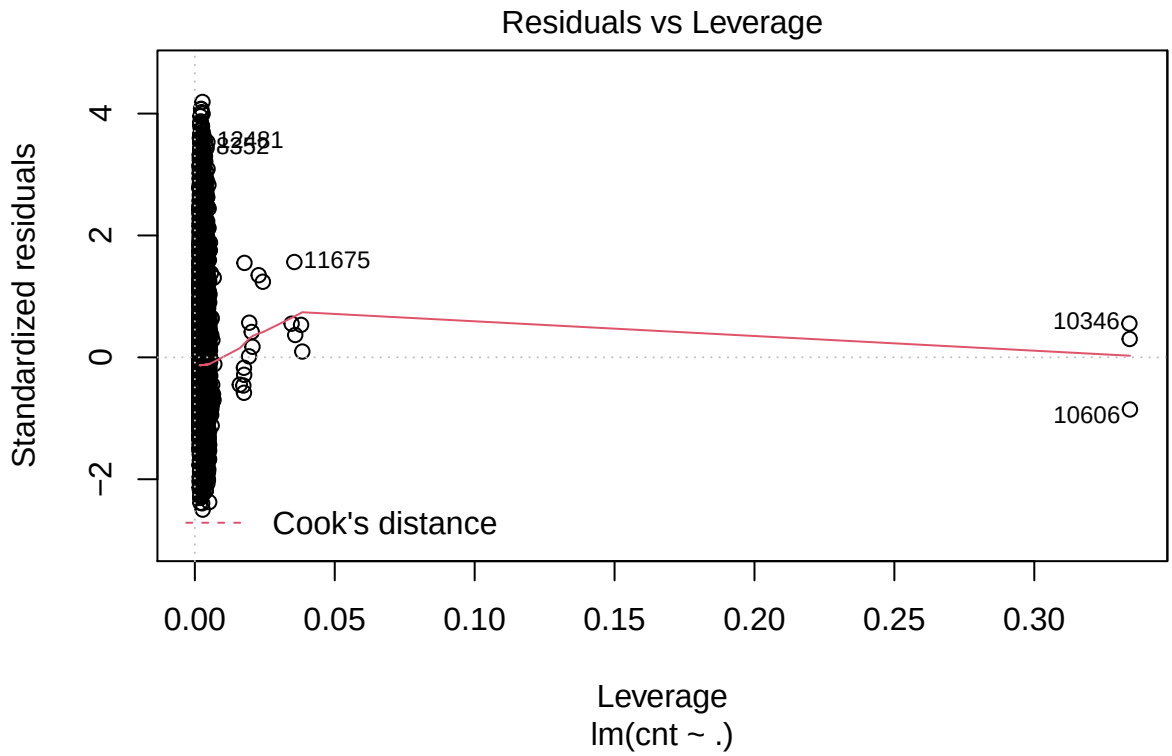
From the summary table above, the overall R squared value is fairly well (R2 = 0.524), year, tempeture, humidity, season, month, hour in a day, holiday, weekday, wheather situation, and the proporation of casual user all showed strong relationship with the total amount of bike rental. I will next do Cook's Distance and

Leverage plots to check if the top influential cases.

```
# Cooks Distance plot
plot(bike.model.lm1, which = 4, id.n = 5)
```




```
# Leverage plot
plot(bike.model.lm1, which = 5, id.n = 5)
```



From the plots, we can see the top 5 influential cases. From Leverage plot we can see that none of these observations is having a huge impact on the model.

2. Perform stepwise backward elimination

Here I will first do a quick backward elimination using AIC to see if I can improve the model.

```
# perform stepwise backward elimination
bike.model.lm2 <- step(bike.model.lm1, direction = "backward")
```

```
## Start:  AIC=122272.8
## cnt ~ yr + temp + atemp + hum + windspeed + percentage + season_2 +
##     season_3 + season_4 + mnth_2 + mnth_3 + mnth_4 + mnth_5 +
##     mnth_6 + mnth_7 + mnth_8 + mnth_9 + mnth_10 + mnth_11 + mnth_12 +
##     hrbin_2 + hrbin_3 + hrbin_4 + holiday_1 + weekday_1 + weekday_2 +
##     weekday_3 + weekday_4 + weekday_5 + weekday_6 + weathersit_2 +
##     weathersit_3 + weathersit_4
##
##                Df Sum of Sq       RSS    AIC
## - mnth_3        1        52 149526882 122271
## - mnth_2        1      1435 149528266 122271
## - weathersit_4  1      2551 149529381 122271
## - mnth_5        1      2638 149529468 122271
## - windspeed     1      6508 149533338 122271
## - mnth_9        1     11961 149538791 122272
```

```
## - weekday_4     1     12523 149539354 122272
## - mnth_4        1     17829 149544660 122272
## <none>                      149526830 122273
## - weekday_1     1     23630 149550461 122273
## - mnth_10       1     27100 149553930 122273
## - weathersit_2  1     27274 149554104 122273
## - mnth_12       1     29242 149556072 122273
## - mnth_11       1     42855 149569685 122274
## - holiday_1     1     48303 149575133 122275
## - weekday_3     1     51984 149578815 122275
## - weekday_5     1     62435 149589265 122276
## - weekday_2     1     66467 149593298 122277
## - mnth_6        1    108301 149635131 122280
## - weekday_6     1    140321 149667151 122283
## - mnth_8        1    147341 149674172 122284
## - atemp         1    159604 149686435 122285
## - temp          1    227983 149754814 122291
## - season_3      1    270624 149797455 122294
## - season_2      1    321542 149848373 122299
## - mnth_7        1    354488 149881318 122302
## - percentage    1    438627 149965458 122309
## - weathersit_3  1   1258113 150784943 122380
## - season_4      1   1389317 150916147 122392
## - hum           1   3986978 153513808 122615
## - yr            1  10043934 159570764 123121
## - hrbin_4       1  36551324 186078154 125131
## - hrbin_2       1  40270140 189796970 125390
## - hrbin_3       1  42508935 192035765 125543
##
## Step:  AIC=122270.8
## cnt ~ yr + temp + atemp + hum + windspeed + percentage + season_2 +
##     season_3 + season_4 + mnth_2 + mnth_4 + mnth_5 + mnth_6 +
##     mnth_7 + mnth_8 + mnth_9 + mnth_10 + mnth_11 + mnth_12 +
##     hrbin_2 + hrbin_3 + hrbin_4 + holiday_1 + weekday_1 + weekday_2 +
##     weekday_3 + weekday_4 + weekday_5 + weekday_6 + weathersit_2 +
##     weathersit_3 + weathersit_4
##
##                Df Sum of Sq       RSS    AIC
## - mnth_2        1      1511 149528393 122269
## - weathersit_4  1      2532 149529414 122269
## - mnth_5        1      3771 149530654 122269
## - windspeed     1      6537 149533419 122269
## - weekday_4     1     12494 149539376 122270
## - mnth_9        1     17562 149544445 122270
## <none>                      149526882 122271
## - weekday_1     1     23592 149550474 122271
## - weathersit_2  1     27355 149554238 122271
## - mnth_4        1     32395 149559278 122272
## - mnth_10       1     34681 149561564 122272
## - mnth_12       1     37675 149564557 122272
## - holiday_1     1     48841 149575724 122273
## - weekday_3     1     51935 149578817 122273
## - mnth_11       1     52305 149579187 122273
## - weekday_5     1     62670 149589552 122274
```

```
## - weekday_2     1     66415 149593298 122275
## - weekday_6     1    140448 149667331 122281
## - atemp         1    159587 149686470 122283
## - mnth_6        1    183935 149710817 122285
## - mnth_8        1    205489 149732371 122287
## - temp          1    230368 149757250 122289
## - season_3      1    283151 149810033 122293
## - season_2      1    382184 149909066 122302
## - percentage    1    440372 149967254 122307
## - mnth_7        1    492258 150019141 122312
## - weathersit_3  1   1258377 150785259 122378
## - season_4      1   1404126 150931008 122391
## - hum           1   3991992 153518874 122613
## - yr            1  10046965 159573847 123119
## - hrbin_4       1  36651351 186178233 125136
## - hrbin_2       1  40279782 189806664 125388
## - hrbin_3       1  43058153 192585035 125578
##
## Step:  AIC=122268.9
## cnt ~ yr + temp + atemp + hum + windspeed + percentage + season_2 +
##     season_3 + season_4 + mnth_4 + mnth_5 + mnth_6 + mnth_7 +
##     mnth_8 + mnth_9 + mnth_10 + mnth_11 + mnth_12 + hrbin_2 +
##     hrbin_3 + hrbin_4 + holiday_1 + weekday_1 + weekday_2 + weekday_3 +
##     weekday_4 + weekday_5 + weekday_6 + weathersit_2 + weathersit_3 +
##     weathersit_4
##
##                Df Sum of Sq       RSS    AIC
## - weathersit_4  1      2445 149530838 122267
## - mnth_5        1      3601 149531994 122267
## - windspeed     1      6410 149534803 122267
## - weekday_4     1     12500 149540893 122268
## - mnth_9        1     18812 149547205 122269
## <none>                      149528393 122269
## - weekday_1     1     23691 149552084 122269
## - weathersit_2  1     27401 149555794 122269
## - mnth_4        1     33038 149561431 122270
## - mnth_10       1     37831 149566224 122270
## - mnth_12       1     43441 149571834 122271
## - holiday_1     1     48568 149576961 122271
## - weekday_3     1     51732 149580125 122271
## - mnth_11       1     56601 149584993 122272
## - weekday_5     1     62610 149591002 122272
## - weekday_2     1     66623 149595016 122273
## - weekday_6     1    140479 149668872 122279
## - atemp         1    160470 149688862 122281
## - mnth_6        1    186477 149714869 122283
## - mnth_8        1    209342 149737735 122285
## - temp          1    229695 149758088 122287
## - season_3      1    282268 149810660 122292
## - season_2      1    390179 149918572 122301
## - percentage    1    443718 149972111 122306
## - mnth_7        1    499189 150027582 122310
## - weathersit_3  1   1256882 150785275 122376
## - season_4      1   1403880 150932272 122389
```

```
## - hum           1   4002174 153530567 122612
## - yr            1  10048221 159576614 123117
## - hrbin_4       1  36649888 186178281 125134
## - hrbin_2       1  40279528 189807921 125386
## - hrbin_3       1  43057137 192585530 125576
##
## Step:  AIC=122267.1
## cnt ~ yr + temp + atemp + hum + windspeed + percentage + season_2 +
##     season_3 + season_4 + mnth_4 + mnth_5 + mnth_6 + mnth_7 +
##     mnth_8 + mnth_9 + mnth_10 + mnth_11 + mnth_12 + hrbin_2 +
##     hrbin_3 + hrbin_4 + holiday_1 + weekday_1 + weekday_2 + weekday_3 +
##     weekday_4 + weekday_5 + weekday_6 + weathersit_2 + weathersit_3
##
##                Df Sum of Sq       RSS    AIC
## - mnth_5        1      3617 149534455 122265
## - windspeed     1      6473 149537311 122266
## - weekday_4     1     12476 149543314 122266
## - mnth_9        1     18835 149549673 122267
## <none>                      149530838 122267
## - weekday_1     1     23533 149554371 122267
## - weathersit_2  1     27791 149558629 122268
## - mnth_4        1     32994 149563832 122268
## - mnth_10       1     37931 149568768 122268
## - mnth_12       1     43710 149574548 122269
## - holiday_1     1     48687 149579524 122269
## - weekday_3     1     51514 149582352 122270
## - mnth_11       1     56768 149587606 122270
## - weekday_5     1     62651 149593488 122271
## - weekday_2     1     66590 149597427 122271
## - weekday_6     1    140883 149671721 122277
## - atemp         1    160134 149690972 122279
## - mnth_6        1    186213 149717051 122281
## - mnth_8        1    209265 149740102 122283
## - temp          1    229830 149760667 122285
## - season_3      1    282112 149812950 122290
## - season_2      1    389683 149920520 122299
## - percentage    1    443845 149974683 122304
## - mnth_7        1    498891 150029728 122309
## - weathersit_3  1   1260483 150791321 122375
## - season_4      1   1403659 150934496 122387
## - hum           1   4000518 153531356 122610
## - yr            1  10052428 159583266 123116
## - hrbin_4       1  36662701 186193539 125133
## - hrbin_2       1  40279111 189809949 125384
## - hrbin_3       1  43093133 192623971 125577
##
## Step:  AIC=122265.4
## cnt ~ yr + temp + atemp + hum + windspeed + percentage + season_2 +
##     season_3 + season_4 + mnth_4 + mnth_6 + mnth_7 + mnth_8 +
##     mnth_9 + mnth_10 + mnth_11 + mnth_12 + hrbin_2 + hrbin_3 +
##     hrbin_4 + holiday_1 + weekday_1 + weekday_2 + weekday_3 +
##     weekday_4 + weekday_5 + weekday_6 + weathersit_2 + weathersit_3
##
##                Df Sum of Sq       RSS    AIC
```

```
## - windspeed     1      6010 149540465 122264
## - weekday_4     1     13058 149547513 122265
## <none>                      149534455 122265
## - weekday_1     1     24109 149558563 122266
## - weathersit_2  1     29059 149563514 122266
## - mnth_9        1     34039 149568494 122266
## - mnth_10       1     47250 149581705 122268
## - holiday_1     1     48124 149582579 122268
## - mnth_12       1     49025 149583479 122268
## - weekday_3     1     52972 149587426 122268
## - weekday_5     1     61907 149596362 122269
## - mnth_11       1     64614 149599069 122269
## - weekday_2     1     68016 149602471 122269
## - mnth_4        1     98558 149633013 122272
## - weekday_6     1    140098 149674553 122276
## - atemp         1    158409 149692864 122277
## - temp          1    244021 149778475 122285
## - mnth_8        1    310320 149844775 122291
## - season_3      1    336142 149870597 122293
## - mnth_6        1    436380 149970835 122302
## - percentage    1    447219 149981674 122302
## - mnth_7        1    718597 150253052 122326
## - season_2      1    774812 150309267 122331
## - weathersit_3  1   1264451 150798906 122374
## - season_4      1   1458223 150992678 122390
## - hum           1   4004365 153538819 122609
## - yr            1  10051468 159585923 123114
## - hrbin_4       1  36756294 186290749 125138
## - hrbin_2       1  40277074 189811529 125383
## - hrbin_3       1  43357362 192891817 125593
##
## Step:  AIC=122263.9
## cnt ~ yr + temp + atemp + hum + percentage + season_2 + season_3 +
##     season_4 + mnth_4 + mnth_6 + mnth_7 + mnth_8 + mnth_9 + mnth_10 +
##     mnth_11 + mnth_12 + hrbin_2 + hrbin_3 + hrbin_4 + holiday_1 +
##     weekday_1 + weekday_2 + weekday_3 + weekday_4 + weekday_5 +
##     weekday_6 + weathersit_2 + weathersit_3
##
##                Df Sum of Sq       RSS    AIC
## - weekday_4     1     13355 149553820 122263
## <none>                      149540465 122264
## - weekday_1     1     24404 149564868 122264
## - weathersit_2  1     28362 149568826 122264
## - mnth_9        1     35039 149575503 122265
## - mnth_10       1     46648 149587113 122266
## - holiday_1     1     48080 149588545 122266
## - mnth_12       1     48407 149588872 122266
## - weekday_3     1     53710 149594174 122267
## - weekday_5     1     60551 149601016 122267
## - mnth_11       1     63213 149603677 122267
## - weekday_2     1     68224 149608689 122268
## - mnth_4        1     95393 149635857 122270
## - weekday_6     1    140356 149680820 122274
## - atemp         1    152804 149693269 122275
```

```
## - temp          1    281597 149822061 122287
## - mnth_8        1    316683 149857148 122290
## - season_3      1    333997 149874462 122291
## - mnth_6        1    444384 149984849 122301
## - percentage    1    451746 149992211 122301
## - mnth_7        1    730019 150270484 122326
## - season_2      1    771127 150311592 122329
## - weathersit_3  1   1263537 150804001 122372
## - season_4      1   1452277 150992741 122388
## - hum           1   4262439 153802904 122629
## - yr            1  10047890 159588355 123112
## - hrbin_4       1  36820350 186360814 125141
## - hrbin_2       1  40494388 190034852 125396
## - hrbin_3       1  43667038 193207502 125612
##
## Step:  AIC=122263.1
## cnt ~ yr + temp + atemp + hum + percentage + season_2 + season_3 +
##     season_4 + mnth_4 + mnth_6 + mnth_7 + mnth_8 + mnth_9 + mnth_10 +
##     mnth_11 + mnth_12 + hrbin_2 + hrbin_3 + hrbin_4 + holiday_1 +
##     weekday_1 + weekday_2 + weekday_3 + weekday_5 + weekday_6 +
##     weathersit_2 + weathersit_3
##
##                Df Sum of Sq       RSS    AIC
## - weekday_1     1     12493 149566312 122262
## <none>                      149553820 122263
## - weathersit_2  1     29579 149583398 122264
## - mnth_9        1     34296 149588115 122264
## - weekday_3     1     40601 149594421 122265
## - mnth_10       1     47381 149601201 122265
## - mnth_12       1     49400 149603220 122265
## - holiday_1     1     52329 149606149 122266
## - weekday_2     1     56013 149609832 122266
## - mnth_11       1     64731 149618551 122267
## - mnth_4        1     95082 149648901 122269
## - weekday_5     1    135157 149688977 122273
## - atemp         1    152553 149706373 122274
## - weekday_6     1    240237 149794056 122282
## - temp          1    277311 149831131 122285
## - mnth_8        1    312350 149866170 122288
## - season_3      1    331550 149885370 122290
## - mnth_6        1    437635 149991455 122299
## - percentage    1    451214 150005034 122301
## - mnth_7        1    721070 150274890 122324
## - season_2      1    767314 150321134 122328
## - weathersit_3  1   1263538 150817357 122371
## - season_4      1   1458948 151012768 122388
## - hum           1   4249393 153803212 122628
## - yr            1  10116458 159670278 123117
## - hrbin_4       1  36936985 186490804 125148
## - hrbin_2       1  40533293 190087112 125398
## - hrbin_3       1  43654630 193208450 125611
##
## Step:  AIC=122262.2
## cnt ~ yr + temp + atemp + hum + percentage + season_2 + season_3 +
```

```
##     season_4 + mnth_4 + mnth_6 + mnth_7 + mnth_8 + mnth_9 + mnth_10 +
##     mnth_11 + mnth_12 + hrbin_2 + hrbin_3 + hrbin_4 + holiday_1 +
##     weekday_2 + weekday_3 + weekday_5 + weekday_6 + weathersit_2 +
##     weathersit_3
##
##                Df Sum of Sq       RSS    AIC
## <none>                      149566312 122262
## - weekday_3     1     30257 149596569 122263
## - weathersit_2  1     30636 149596948 122263
## - mnth_9        1     34245 149600558 122263
## - weekday_2     1     44693 149611005 122264
## - mnth_10       1     46980 149613292 122264
## - mnth_12       1     48405 149614717 122264
## - mnth_11       1     63417 149629730 122266
## - holiday_1     1     72368 149638680 122267
## - mnth_4        1     94463 149660775 122268
## - atemp         1    149745 149716058 122273
## - weekday_5     1    185117 149751430 122276
## - temp          1    280629 149846942 122285
## - weekday_6     1    300386 149866698 122286
## - mnth_8        1    314889 149881202 122288
## - season_3      1    332056 149898369 122289
## - mnth_6        1    439037 150005349 122299
## - percentage    1    440043 150006355 122299
## - mnth_7        1    722598 150288910 122323
## - season_2      1    765180 150331493 122327
## - weathersit_3  1   1264112 150830424 122370
## - season_4      1   1454868 151021180 122387
## - hum           1   4252638 153818951 122627
## - yr            1  10125603 159691915 123117
## - hrbin_4       1  36929688 186496000 125146
## - hrbin_2       1  40526274 190092586 125396
## - hrbin_3       1  43653568 193219880 125609
```

```
# summary the model
summary(bike.model.lm2)
```

```
##
## Call:
## lm(formula = cnt ~ yr + temp + atemp + hum + percentage + season_2 +
##     season_3 + season_4 + mnth_4 + mnth_6 + mnth_7 + mnth_8 +
##     mnth_9 + mnth_10 + mnth_11 + mnth_12 + hrbin_2 + hrbin_3 +
##     hrbin_4 + holiday_1 + weekday_2 + weekday_3 + weekday_5 +
##     weekday_6 + weathersit_2 + weathersit_3, data = bike.training.lm)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -267.16  -73.56  -21.09   53.62  448.74
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)   -58.207      5.825  -9.993  < 2e-16 ***
## yr             56.476      1.900  29.725  < 2e-16 ***
## temp          174.316     35.226   4.948 7.57e-07 ***
```

```
## atemp         135.166     37.393   3.615 0.000302 ***
## hum          -125.434      6.512 -19.263  < 2e-16 ***
## percentage    -59.001      9.522  -6.197 5.94e-10 ***
## season_2       32.990      4.037   8.171 3.33e-16 ***
## season_3       34.784      6.462   5.383 7.46e-08 ***
## season_4       64.976      5.767  11.267  < 2e-16 ***
## mnth_4        -12.415      4.324  -2.871 0.004098 **
## mnth_6        -29.874      4.827  -6.189 6.22e-10 ***
## mnth_7        -57.170      7.200  -7.941 2.18e-15 ***
## mnth_8        -37.148      7.087  -5.242 1.61e-07 ***
## mnth_9        -11.110      6.427  -1.729 0.083896 .
## mnth_10       -13.608      6.721  -2.025 0.042918 *
## mnth_11       -15.662      6.658  -2.352 0.018668 *
## mnth_12       -10.587      5.151  -2.055 0.039882 *
## hrbin_2       158.821      2.671  59.467  < 2e-16 ***
## hrbin_3       197.573      3.201  61.718  < 2e-16 ***
## hrbin_4       158.035      2.784  56.767  < 2e-16 ***
## holiday_1     -14.248      5.670  -2.513 0.011986 *
## weekday_2      -5.783      2.928  -1.975 0.048311 *
## weekday_3      -4.785      2.945  -1.625 0.104217
## weekday_5      11.584      2.882   4.019 5.87e-05 ***
## weekday_6      14.923      2.915   5.120 3.10e-07 ***
## weathersit_2   -3.771      2.307  -1.635 0.102072
## weathersit_3  -41.270      3.929 -10.503  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 107.1 on 13051 degrees of freedom
## Multiple R-squared:  0.5226, Adjusted R-squared:  0.5217
## F-statistic: 549.5 on 26 and 13051 DF,  p-value: < 2.2e-16
```

We can see here that the stepwise backward elimination didn't improve the model a lot. Next I will different ways of selecting predictors.

3. Explore other ways of selecting predictors

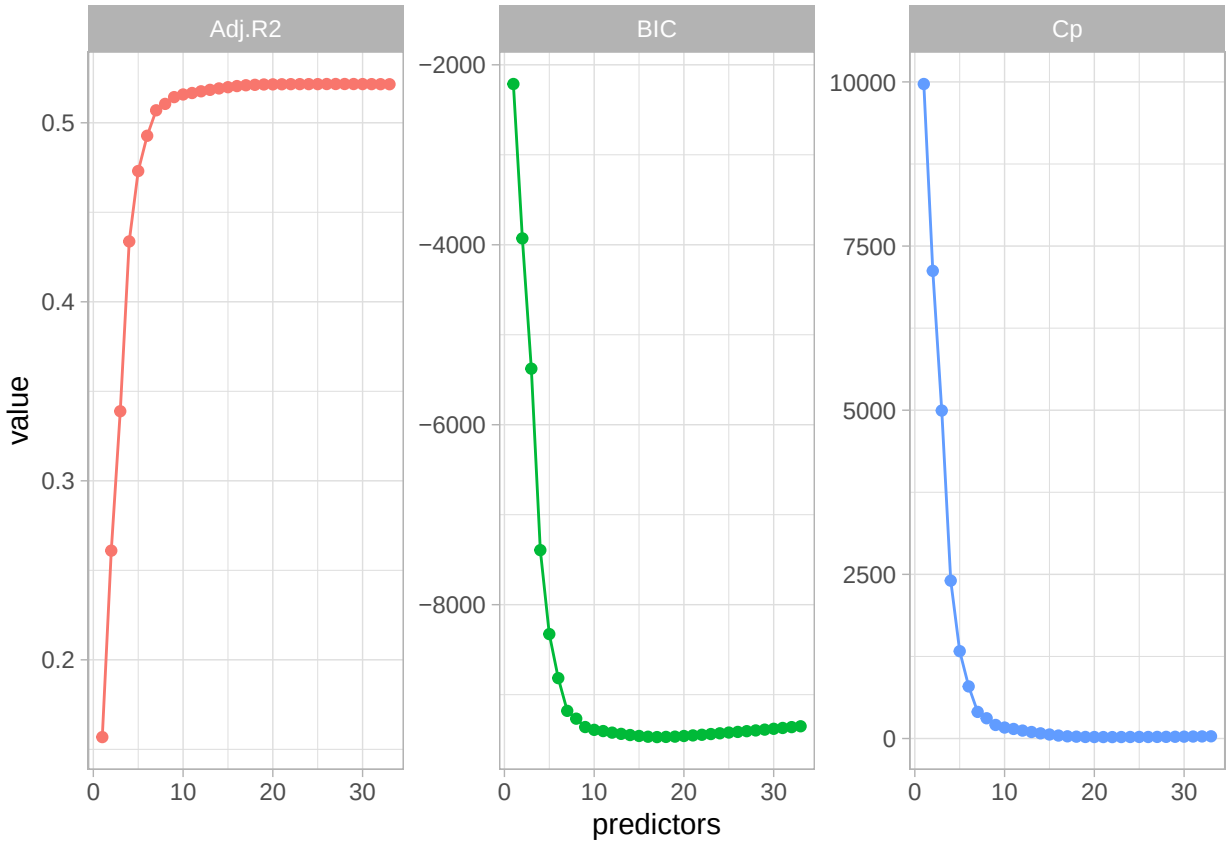
I tried to use AIC as a criteria to select best model in last step. Now I will try to us adjusted R square, Mallows' Cp, and BIC to select predictors.

Mallows' Cp-statistic estimates the size of the bias that is introduced into the predicted responses by having an underspecified model. Subset models with small Cp values have a small estimated total (standardized) variation in predicted responses. If all models, except the full model, yield a large Cp not near k+1, meaning some important predictor(s) are missing from the analysis; If a number of models have Cp near k+1, the model with the smallest Cp value are the best one, which insures that the combination of the bias and the variance is at a minimum.

BIC is an estimate of a function of the posterior probability of a model being true, a lower BIC means that a model is considered to be more likely to be the true model. Comparing to AIC, AIC tries to select the model that most adequately describes an unknown, high dimensional reality, while BIC tries to find the "true" model among the set of potential models.

Here I perform a subset search using regsubsets (package "leaps"), which identifies the best model for a given number of k predictors, where best is quantified using RSS. The nvmax option can be used in order to return as many variables as are desired. I use 33-variable model here.

```
# perform a subset search
subset.lm <- regsubsets(cnt ~ ., data = bike.training.lm, nvmax = 33)
# access to the result
results.lm <- summary(subset.lm)
# create a data frame with all adjusted-R2 values, Cp values, and BIC values,
# then make plots to visualize them
data.frame(predictors = as.integer(c(1:33)),
           Adj.R2 = results.lm$adjr2,
           Cp = results.lm$cp,
           BIC = results.lm$bic) %>%
  gather(statistic, value, -predictors) %>%
  ggplot(aes(predictors, value, color = statistic)) +
  geom_line(show.legend = FALSE) +
  geom_point(show.legend = FALSE) +
  facet_wrap(~ statistic, scales = "free") +
  theme_light()
```




```
# report the maximum adjusted-R2, minimum Cp, and minimum BIC
which.max(results.lm$adjr2)
```

```
## [1] 28
```

```
which.min(results.lm$cp)
```

```
## [1] 22
```

```r
which.min(results.lm$bic)
```

```
## [1] 17
```

From the results, adjusted R2 suggested a 27-variable model, Cp suggestes a 23-variable model, and BIC suggested a 18-variable model. I will further check the coefficients of these suggested models.

```r
# check the coefficients of thest models
coef(subset.lm, 27)
```

```
##  (Intercept)           yr         temp        atemp          hum   percentage
##   -57.235529    56.453945   173.342913   136.508059  -125.389023   -60.127705
##     season_2     season_3     season_4       mnth_4       mnth_6       mnth_7
##    33.038406    34.757612    65.076425   -12.455953   -29.828020   -57.110843
##       mnth_8       mnth_9      mnth_10      mnth_11      mnth_12      hrbin_2
##   -37.004643   -11.117840   -13.666531   -15.827479   -10.697179   158.837525
##      hrbin_3      hrbin_4    holiday_1    weekday_1    weekday_2    weekday_3
##   197.684012   158.053581   -12.584883    -3.302102    -6.873776    -5.889534
##    weekday_5    weekday_6 weathersit_2 weathersit_3
##    10.510064    13.976018    -3.707045   -41.260450
```

```r
coef(subset.lm, 23)
```

```
##  (Intercept)           yr         temp        atemp          hum   percentage
##   -58.161800    56.502735   171.449575   135.499914  -126.379807   -59.658950
##     season_2     season_3     season_4       mnth_4       mnth_5       mnth_6
##    29.320225    27.283319    53.285659    -6.945294     6.806347   -22.535141
##       mnth_7       mnth_8      hrbin_2      hrbin_3      hrbin_4    holiday_1
##   -47.087244   -27.198201   158.828879   197.713873   158.096308   -15.305622
##    weekday_2    weekday_3    weekday_5    weekday_6 weathersit_2 weathersit_3
##    -5.737645    -4.838406    11.414301    14.824754    -3.634426   -41.240759
```

```r
coef(subset.lm, 18)
```

```
##  (Intercept)           yr         temp        atemp          hum   percentage
##    -60.42568     56.38397    166.35082    142.32473   -129.71015    -58.56554
##     season_2     season_3     season_4       mnth_4       mnth_6       mnth_7
##     33.92376     28.24785     53.41230    -11.67209    -25.36895    -47.88089
##       mnth_8      hrbin_2      hrbin_3      hrbin_4    weekday_5    weekday_6
##    -27.45855    158.48972    196.81992    157.86715     13.78459     17.37876
## weathersit_3
##    -39.56086
```

```r
# create corresponding models
# get all the predictors selected by these models respectively
names.r2 <- names(coef(subset.lm, 27))
names.cp <- names(coef(subset.lm, 23))
names.bic <- names(coef(subset.lm, 18))
names.r2 <- names.r2[!names.r2 %in% "(Intercept)"]
names.cp <- names.cp[!names.cp %in% "(Intercept)"]
```

```
names.bic <- names.bic[!names.bic %in% "(Intercept)"]
# create forumla for these models respectively
formula.r2 <- as.formula((paste("cnt",
                                paste(names.r2, collapse = "+"), sep = "~")))
formula.cp <- as.formula((paste("cnt",
                                paste(names.cp, collapse = "+"), sep = "~")))
formula.bic <- as.formula((paste("cnt",
                                 paste(names.bic, collapse = "+"), sep = "~")))
# generate models
bike.model.lm3 <- lm(formula.r2, data = bike.training.lm)
bike.model.lm4 <- lm(formula.cp, data = bike.training.lm)
bike.model.lm5 <- lm(formula.bic, data = bike.training.lm)
```

4. Evaluate models performance with holdout method

In this step, I will generate a table for adjusted-R2, MAE (mean absolute error), RMSE (rooted mean squared error), and RSE (residual standard error) for all the 5 models I created before to compare between them.

```
# make predictions using 5 models
bike.pred.lm1 <- predict(bike.model.lm1, newdata = bike.testing.lm)
bike.pred.lm2 <- predict(bike.model.lm2, newdata = bike.testing.lm)
bike.pred.lm3 <- predict(bike.model.lm3, newdata = bike.testing.lm)
bike.pred.lm4 <- predict(bike.model.lm4, newdata = bike.testing.lm)
bike.pred.lm5 <- predict(bike.model.lm5, newdata = bike.testing.lm)
# create functions to calculate MAE, RMSE, and RSE
# MAE
MAE <- function(actual, predicted){
  mean(abs(actual - predicted))
}
# RMSE
RMSE <- function(actual, predicted){
  sqrt(mean(actual - predicted) ^ 2)
}
# RSE
RSE <- function(model, dataset){
  sigma(model) / mean(dataset$cnt)
}
# generate a table to compare everything together
data.frame(model = c("lm1", "lm2", "lm3", "lm4", "lm5"),
           Adj.R2 = c(summary(bike.model.lm1)$adj.r.squared,
                      summary(bike.model.lm2)$adj.r.squared,
                      summary(bike.model.lm3)$adj.r.squared,
                      summary(bike.model.lm4)$adj.r.squared,
                      summary(bike.model.lm5)$adj.r.squared),
           MAE = c(MAE(bike.testing.lm$cnt, bike.pred.lm1),
                   MAE(bike.testing.lm$cnt, bike.pred.lm2),
                   MAE(bike.testing.lm$cnt, bike.pred.lm3),
                   MAE(bike.testing.lm$cnt, bike.pred.lm4),
                   MAE(bike.testing.lm$cnt, bike.pred.lm5)),
           RMSE = c(RMSE(bike.testing.lm$cnt, bike.pred.lm1),
                    RMSE(bike.testing.lm$cnt, bike.pred.lm2),
```

```
                    RMSE(bike.testing.lm$cnt, bike.pred.lm3),
                    RMSE(bike.testing.lm$cnt, bike.pred.lm4),
                    RMSE(bike.testing.lm$cnt, bike.pred.lm5)),
          RSE = c(RSE(bike.model.lm1, bike.testing.lm),
                  RSE(bike.model.lm2, bike.testing.lm),
                  RSE(bike.model.lm3, bike.testing.lm),
                  RSE(bike.model.lm4, bike.testing.lm),
                  RSE(bike.model.lm5, bike.testing.lm)))
```

```
##   model    Adj.R2      MAE     RMSE       RSE
## 1   lm1 0.5215242 81.50513 1.225421 0.6176416
## 2   lm2 0.5216546 81.60074 1.212886 0.6175575
## 3   lm3 0.5216579 81.57441 1.190980 0.6175553
## 4   lm4 0.5215869 81.60623 1.231821 0.6176011
## 5   lm5 0.5211954 81.68817 1.225008 0.6178538
```

From the table above, there is no single model stands out. Model 2 has the lowest MAE and RMSE values, I would say for now, it is the best model. Next I will try cross-validation to see which one is better than others.

5. Evaluate models performance with k-fold cross-validation

```
set.seed(123)
# create 10 folders from the cleaned dummy dataset
folds.lm <- createFolds(bike.dummy$cnt, k = 10)
# create a list for all 5 models for easier access below
models.lm <- paste0("bike.model.lm", 1:5)
# create a function to calculate RMSE values for all folder
# for specific model i
cvRMSE <- function(i){
  # calculate RMSE for each folder
  lapply(folds.lm, function(x){
    # create a training data
    bike.training.cv <- bike.dummy[-x, ]
    # create a testing data
    bike.testing.cv <- bike.dummy[x, ]
    # get the specific model i
    bike.model <- get(models.lm[i])
    # make prediction
    bike.pred <- predict(bike.model, bike.testing.cv)
    # get the actual value
    bike.actual <- bike.testing.cv$cnt
    # calculate the RMSE value
    RMSE <- RMSE(bike.actual, bike.pred)
    return(RMSE)
  })
}
# apply the function to each model to calculate mean RMSE across 10 folders
mean(unlist(cvRMSE(1)))
```

```
## [1] 2.031265
```

```
mean(unlist(cvRMSE(2)))
```

```
## [1] 2.023346
```

```
mean(unlist(cvRMSE(3)))
```

```
## [1] 2.026373
```

```
mean(unlist(cvRMSE(4)))
```

```
## [1] 2.029059
```

```
mean(unlist(cvRMSE(5)))
```

```
## [1] 1.996073
```

Based on the result, Model 2 is the second best one, conbined with the comparison of adjusted R2, MAE, RMSE, and RSE, I will select Model as the final multiple linear regression model.

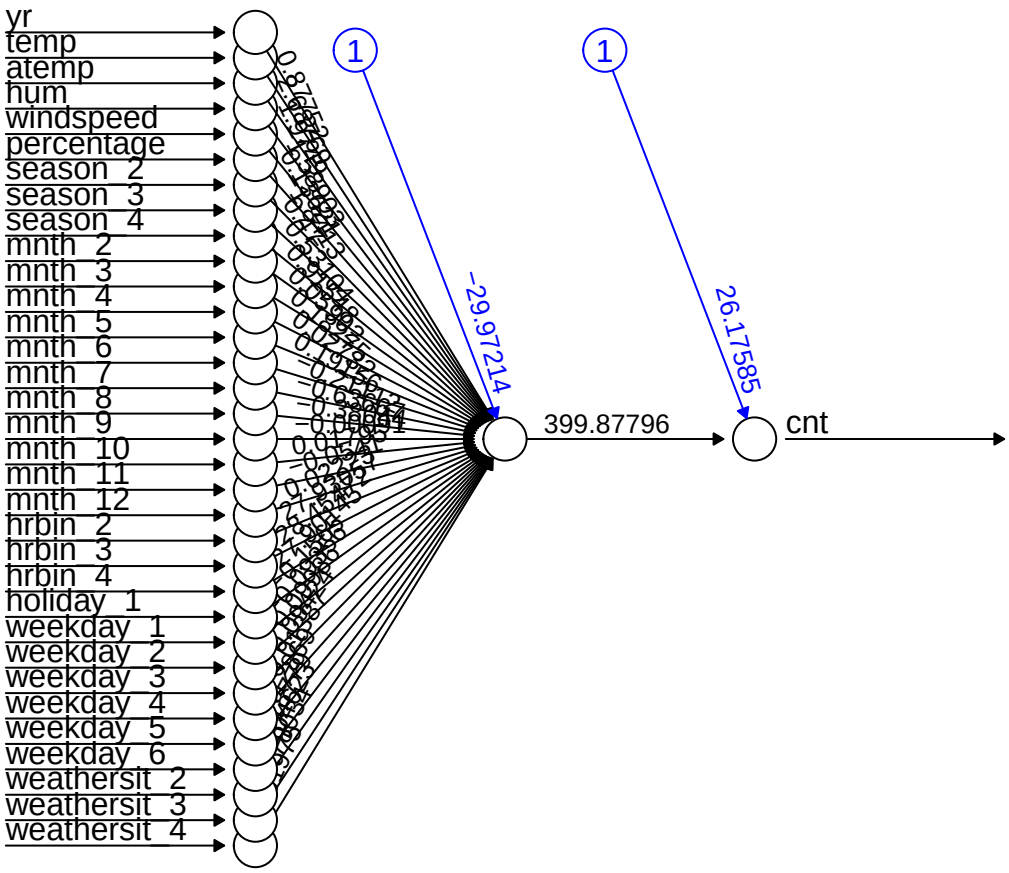
```
bike.model.lm.final <- bike.model.lm2
```

Step 6. Nerual Network

1. Train the first model

I will first train the neural network model with one hidden layer.

```
set.seed(123)
# train the model
bike.model.ann <- neuralnet(cnt ~ ., data = bike.training.ann, threshold = 0.2)
# plot the network
plot(bike.model.ann, rep = "best", information = TRUE, information.pos = 0.1)
```



2. Evaluate the model's performance

Since adding hidden nodes is very time consuming given my local computer's capability. I will keep the first model as the final model of neural network and use it for ensembling and evaluation with all the other models. I will first evaluate its performance here to see if it is good in general.

```
# make predictions
bike.pred.ann <- compute(bike.model.ann, bike.testing.ann[ , c(1:5, 7:34)])
# convert the prediction results to a data frame
bike.pred.ann <- as.data.frame(bike.pred.ann$net.result)
# generate a table with correlation, MAE, and RMSE of the model
data.frame(model = "ann",
           cor = cor(bike.pred.ann$V, bike.testing.ann$cnt),
           MAE = MAE(bike.testing.ann$cnt, bike.pred.ann$V1),
           RMSE = RMSE(bike.testing.ann$cnt, bike.pred.ann$V1))
```

```
##   model       cor      MAE     RMSE
## 1   ann 0.7489745 74.26743 1.428647
```

From the table above, we can see the correlation is 0.72, which is fairly strong linear relationship, and the MAE and RMSE are even lower than all the 5 multiple linear regression models.

```
bike.model.ann.final <- bike.model.ann
```

Step 7. Regression tree

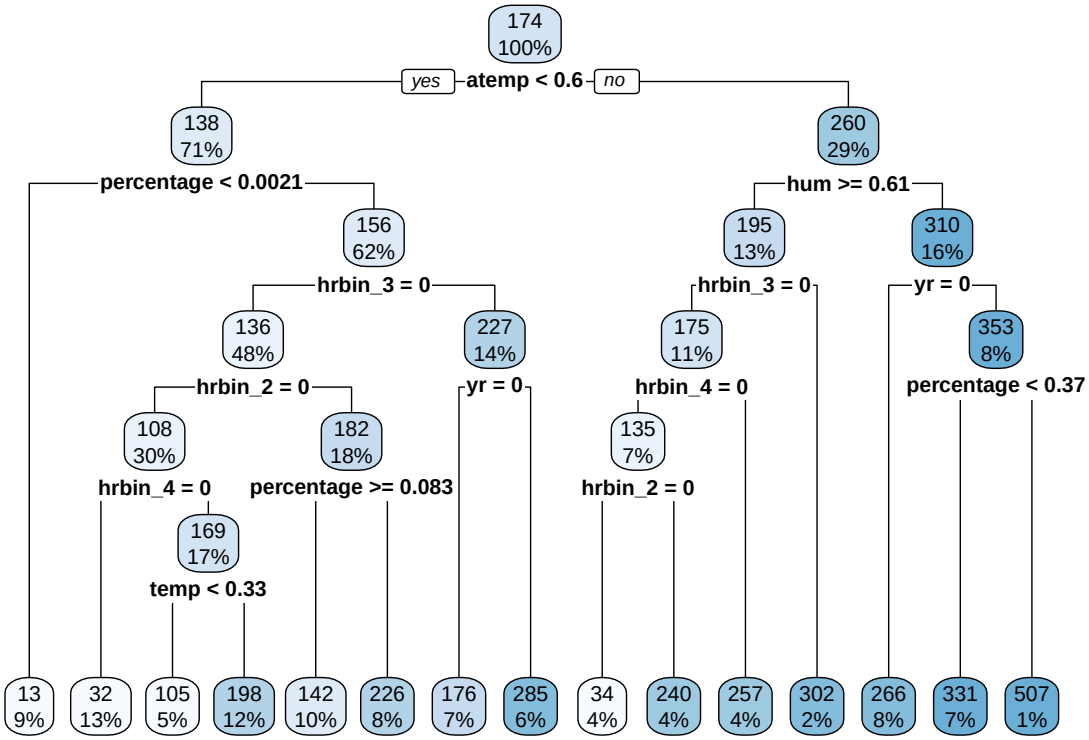
1. Train the first model

Here I will train the first tree and let it select the splits of the nodes.

```
set.seed(123)
# train a tree using all columns
bike.model.tree1 <- rpart(cnt ~ ., data = bike.training.tree, method = "anova")
bike.model.tree1
```

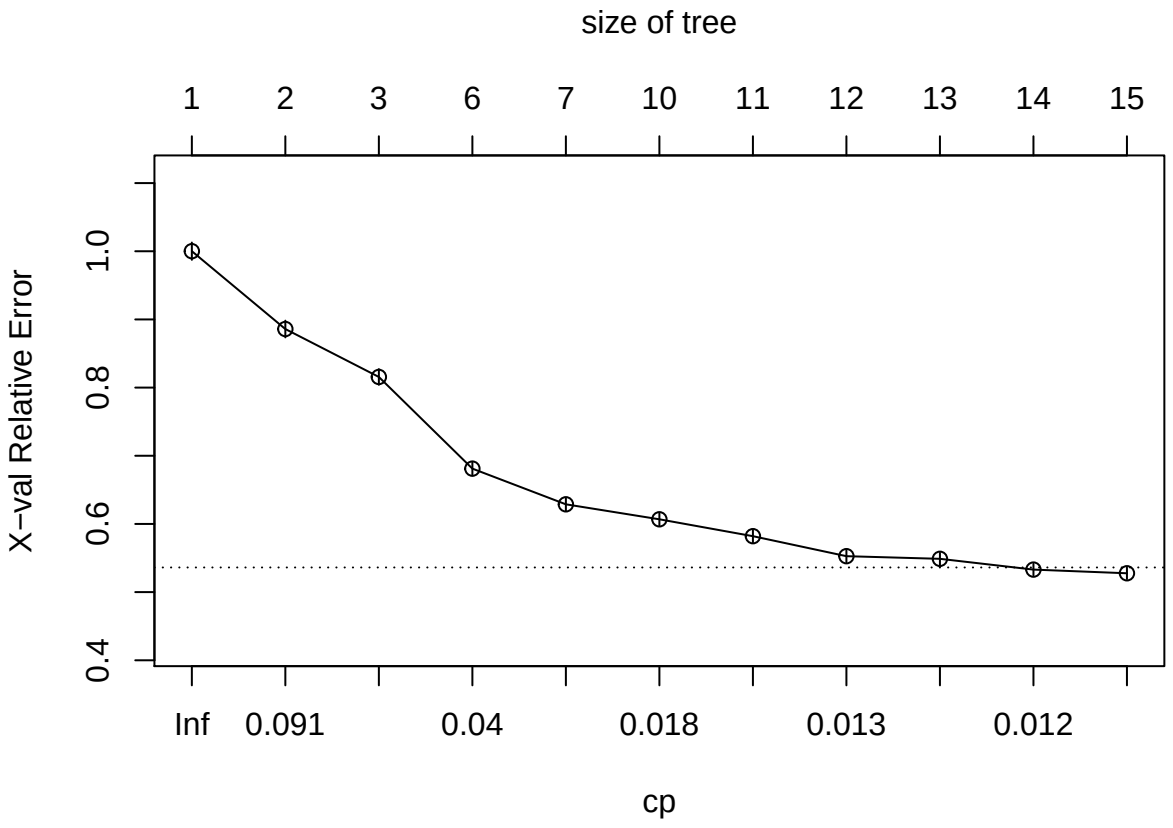
```
## n= 13078
##
## node), split, n, deviance, yval
##       * denotes terminal node
##
##  1) root 13078 313297200.0 173.70470
##    2) atemp< 0.5985 9262 172788000.0 138.25780
##      4) percentage< 0.002083333 1136    392394.7  13.12324 *
##      5) percentage>=0.002083333 8126 152120600.0 155.75140
##       10) hrbin_3< 0.5 6339 106328200.0 135.55290
##         20) hrbin_2< 0.5 3981  53835140.0 108.24090
##           40) hrbin_4< 0.5 1764   1835034.0  31.61338 *
##           41) hrbin_4>=0.5 2217  33400890.0 169.21110
##             82) temp< 0.33 681   5025599.0 105.02500 *
##             83) temp>=0.33 1536  24325770.0 197.66860 *
##         21) hrbin_2>=0.5 2358  44509850.0 181.66370
##           42) percentage>=0.08323436 1256  14517020.0 142.38220 *
##           43) percentage< 0.08323436 1102  25845880.0 226.43470 *
##       11) hrbin_3>=0.5 1787  34032330.0 227.40120
##         22) yr< 0.5 945  11923700.0 175.77990 *
##         23) yr>=0.5 842  16764180.0 285.33730 *
##    3) atemp>=0.5985 3816 100625600.0 259.73950
##      6) hum>=0.605 1659  39359920.0 194.53950
##       12) hrbin_3< 0.5 1408  31241800.0 175.31960
##         24) hrbin_4< 0.5 943  18968980.0 135.22480
##           48) hrbin_2< 0.5 479    587457.9  33.75783 *
##           49) hrbin_2>=0.5 464   8358961.0 239.97200 *
##         25) hrbin_4>=0.5 465   7682556.0 256.63010 *
##       13) hrbin_3>=0.5 251   4680359.0 302.35460 *
##      7) hum< 0.605 2157  48789010.0 309.88640
##       14) yr< 0.5 1067  21842360.0 265.62040 *
##       15) yr>=0.5 1090  22809240.0 353.21830
##         30) percentage< 0.3747124 953  17314820.0 331.07240 *
##         31) percentage>=0.3747124 137   1775753.0 507.27010 *
```

```
# plot to visualize the tree
rpart.plot(bike.model.tree1)
```



We can see without any limitation, the tree used 13 predictors to produce this model. It is the results from the application and selection from different complexity parameter (cp) that rpart did behind, that rpart automatically applyed a range of cost complexity ( ) values to prune the tree. To compare the error for each value, rpart performs a 10-fold cross validation so that the error associated with a given value is computed on the validation data. I will then explore if this predictors selection is the most optimized.

```
# plot the cp, tree size, and error relationship
plotcp(bike.model.tree1)
```




```
# print the table of cp, tree size, and error
bike.model.tree1$cptable
```

```
##            CP nsplit rel error    xerror        xstd
## 1  0.12730251      0 1.0000000 1.0000933 0.012716891
## 2  0.06471494      1 0.8726975 0.8859128 0.012297357
## 3  0.04079461      2 0.8079825 0.8155938 0.011553942
## 4  0.03982390      5 0.6855987 0.6811272 0.010100066
## 5  0.01920497      6 0.6457748 0.6287737 0.009651109
## 6  0.01705871      9 0.5881599 0.6068048 0.009523720
## 7  0.01323647     10 0.5711012 0.5820280 0.009220659
## 8  0.01320602     11 0.5578647 0.5526809 0.008825092
## 9  0.01292551     12 0.5446587 0.5488452 0.008763623
## 10 0.01186946     13 0.5317332 0.5330387 0.008551767
## 11 0.01000000     14 0.5198637 0.5276424 0.008517304
```

We can see from the plot, rpart automatically selected the smallest tree size within 1 standard deviation of the minimum cross validation error, which is 13 in this model with a cross-validated error of 0.54.

2. Explore metohd to improve the model

As I decribed above, rpart performs tuning based on cp value automatically, which returned a tree of 13 splits and 14 terminal nodes, with a cross-validated error of 0.54. Next I will try to perform other types of tuning to see if they can improve the model performance.

In addition to the cost complexity ( ) parameter, it is also common to tune:

minsplit: the minimum number of data points required to attempt a split before it is forced to create a terminal node. The default value is 20. Lowering this value will keep those terminal nodes that may contain only a handful of observations to create the predicted value.

maxdepth: the maximum number of internal nodes between the root node and the terminal nodes. The default value is 30. Changing this values allows different sizes of a tree.

Here I will create a for loop to automatically try different combinations of minsplit and maxdepth.

```
# create a grid for different minsplit and maxdepth values
# for minsplit, I use the range from 5-20 given that the maximum value of
# minsplit can go is 20
# for maxdepth, I use the range from 10-20 since the first model returned a
# tree with an optimal depth of 13
combo.crtl <- expand.grid(minsplit = seq(5, 20, 1), maxdepth = seq(10, 20, 1))
# check the first few rows of the matrix
head(combo.crtl)
```

```
##   minsplit maxdepth
## 1        5       10
## 2        6       10
## 3        7       10
## 4        8       10
## 5        9       10
## 6       10       10
```

```
# create an empty list for models that going to be generated
models.tree <- list()
# create a for loop to run through each combinations of minsplit and maxdepth
for(i in 1:nrow(combo.crtl)){
  # get minsplit and maxdepth values in that row
  minsplit <- combo.crtl$minsplit[i]
  maxdepth <- combo.crtl$maxdepth[i]
  # train the model with this combination and store it in the model list
  models.tree[[i]] <- rpart(cnt ~.,
                        data = bike.training.tree,
                        method = "anova",
                        control = list(minsplit = minsplit, maxdepth = maxdepth))
}
```

Next I will create a function to extract the minimum error associated with the optimal cost complexity value for each model.

```
# function to get optimal cp
get.cp <- function(x){
  min <- which.min(x$cptable[ , "xerror"])
  cp <- x$cptable[min, "CP"]
}
# function to get minimum error
get.minerror <- function(x){
  min <- which.min(x$cptable[ , "xerror"])
  xerror <- x$cptable[min, "xerror"]
}
```

```
# generate a table with the information of minsplit and maxdepth,
# and the optimal cp value and error just calculated
# then only keep the top 5 models with minimum error
combo.crtl <- combo.crtl %>%
  mutate(error = purrr::map_dbl(models.tree, get.minerror)) %>%
  arrange(error) %>%
  top_n(-5, wt = error)
combo.crtl
```

```
##   minsplit maxdepth     error
## 1       19       17 0.5246589
## 2       15       17 0.5249850
## 3       10       18 0.5261724
## 4       12       17 0.5264443
## 5        7       11 0.5265910
```

We can see from the table, then minsplit equals 12 and maxdepth equals 20, the model has the smallest error 0.53. Next I will use this model to compare with the original tree model I generated at the first step.

3. Evaluate models performance

```
# train the model using the optimal minsplit and maxdepth values
bike.model.tree2 <- rpart(cnt ~ .,
                          data = bike.training.tree,
                          method = "anova",
                          control = list(cp = 0, minsplit = 12, maxdepth = 20))
bike.model.tree1$cptable
```

```
##            CP nsplit rel error    xerror        xstd
## 1  0.12730251      0 1.0000000 1.0000933 0.012716891
## 2  0.06471494      1 0.8726975 0.8859128 0.012297357
## 3  0.04079461      2 0.8079825 0.8155938 0.011553942
## 4  0.03982390      5 0.6855987 0.6811272 0.010100066
## 5  0.01920497      6 0.6457748 0.6287737 0.009651109
## 6  0.01705871      9 0.5881599 0.6068048 0.009523720
## 7  0.01323647     10 0.5711012 0.5820280 0.009220659
## 8  0.01320602     11 0.5578647 0.5526809 0.008825092
## 9  0.01292551     12 0.5446587 0.5488452 0.008763623
## 10 0.01186946     13 0.5317332 0.5330387 0.008551767
## 11 0.01000000     14 0.5198637 0.5276424 0.008517304
```

```
# make predictions using these two models
bike.pred.tree1 <- predict(bike.model.tree1, newdata = bike.testing.tree)
bike.pred.tree2 <- predict(bike.model.tree2, newdata = bike.testing.tree)
# generate a table to compare everything together
data.frame(model = c("tree1", "tree2"),
           MAE = c(MAE(bike.testing.tree$cnt, bike.pred.tree1),
                   MAE(bike.testing.tree$cnt, bike.pred.tree2)),
           RMSE = c(RMSE(bike.testing.tree$cnt, bike.pred.tree1),
                    RMSE(bike.testing.tree$cnt, bike.pred.tree2)))
```

```
##   model      MAE      RMSE
## 1 tree1 82.81412 0.3824549
## 2 tree2 67.38911 1.1013999
```

From the table above, the MAE and RMSE are both reduced a lot from setting minsplit and maxdepth (MAE from 80.78 to 64.52, RMSE from 1.08 to 0.50). I will set the second model as the final tree model.

```
bike.model.tree.final <- bike.model.tree2
```

Step 8. Ensemble model

1. Create a ensamble model

In this step, I will create a function to ensemble all three models that were generated before (lm.final, ann.final, and tree.final). If any two models make same prediction on one case, I will keep this prediction. If all three models make different prediction in one case, I will keep the one prediction made by the model with the lowest RMSE.

```
# create the ensemble function
predict.cnt <- function(newdata){
  # copy new data for each model
  data.lm <- newdata
  data.ann <- newdata
  data.tree <- newdata
  # make predictions
  pred.lm <- predict(bike.model.lm.final, newdata[ , c(1:5, 7:34)])
  pred.ann <- compute(bike.model.ann.final, newdata[ , c(1:5, 7:34)])
  pred.tree <- predict(bike.model.tree.final, newdata[ , c(1:5, 7:34)])
  # combine all predictions to one data frame
  pred <- data.frame(pred.lm = pred.lm,
                     pred.ann = pred.ann$net.result,
                     pred.tree = pred.tree)
  # calculate RMSE for each model
  RMSE.lm <- RMSE(newdata$cnt, pred.lm)
  RMSE.ann <- RMSE(newdata$cnt, pred.ann$net.result)
  RMSE.tree <- RMSE(newdata$cnt, pred.tree)
  # return the model index that has the lowest RMSE
  best.model.idx <- which.min(c(RMSE.lm, RMSE.ann, RMSE.tree))
  # add a new column in pred data frame with the final selection for each case
  # if any of two models agree with each other, will keep this prediction
  # else it will select the prediction from the model with lowest RMSE
  pred$pred.final <- ifelse(pred.lm == pred.ann$net.result,
                            pred.lm,
                            ifelse(pred.lm == pred.tree, pred.lm,
                                   ifelse(pred.ann$net.result == pred.tree,
                                          pred.ann$net.result,
                                          pred[ , best.model.idx])))
  # output the final selection
  return(pred)
}
# use this ensemble model to make prediction
bike.pred.ensemble <- predict.cnt(bike.testing)
```

2. Evaluate the ensamble model

Next I will compare this ensemble model's performance with each individual model using MAE and RMSE.

```
# generate a table for comparing each individual model and the ensemble model
data.frame(model = c("lm.final", "ann.final", "tree.final", "model.ens"),
           MAE = c(MAE(bike.testing$cnt, bike.pred.ensemble$pred.lm),
                   MAE(bike.testing$cnt, bike.pred.ensemble$pred.ann),
                   MAE(bike.testing$cnt, bike.pred.ensemble$pred.tree),
                   MAE(bike.testing$cnt, bike.pred.ensemble$pred.final)),
           RMSE = c(RMSE(bike.testing$cnt, bike.pred.ensemble$pred.lm),
                    RMSE(bike.testing$cnt, bike.pred.ensemble$pred.ann),
                    RMSE(bike.testing$cnt, bike.pred.ensemble$pred.tree),
                    RMSE(bike.testing$cnt, bike.pred.ensemble$pred.final)))
```

```
##        model      MAE     RMSE
## 1   lm.final 81.60074 1.212886
## 2  ann.final 74.26743 1.428647
## 3 tree.final 67.38911 1.101400
## 4  model.ens 67.38911 1.101400
```

As we see from the table, the ensemble model seems has no difference from tree model. Next I will run a cross-validation same as the function I created before to check the mean RMSE of each model.

```
set.seed(123)
# create 10 folders from the cleaned dummy dataset
folds <- createFolds(bike.dummy$cnt, k = 10)
# calculate mean RMSE values across all folder for each model
# for lm model
cvRMSE.lm <- lapply(folds, function(x){
  training.cv <- bike.dummy[-x, ]
  testing.cv <- bike.dummy[x, ]
  pred.cv <- predict(bike.model.lm.final, testing.cv)
  rmse.lm <- RMSE(testing.cv$cnt, pred.cv)
})
# for ann model
cvRMSE.ann <- lapply(folds, function(x){
  training.cv <- bike.dummy[-x, ]
  testing.cv <- bike.dummy[x, ]
  pred.cv <- compute(bike.model.ann.final, testing.cv)
  pred.cv <- pred.cv$net.result
  rmse.ann <- RMSE(testing.cv$cnt, pred.cv)
})
# for tree model
cvRMSE.tree <- lapply(folds, function(x){
  training.cv <- bike.dummy[-x, ]
  testing.cv <- bike.dummy[x, ]
  pred.cv <- predict(bike.model.tree.final, testing.cv)
  rmse.tree <- RMSE(testing.cv$cnt, pred.cv)
})
# for ens model
cvRMSE.ens <- lapply(folds, function(x){
  training.cv <- bike.dummy[-x, ]
```

```
  testing.cv <- bike.dummy[x, ]
  pred.cv <- predict.cnt(testing.cv)
  pred.cv <- pred.cv$pred.final
  rmse.ens <- RMSE(testing.cv$cnt, pred.cv)
})

data.frame(model = c("lm", "ann", "tree", "ens"),
           mean.rmse = c(mean(unlist(cvRMSE.lm)),
                         mean(unlist(cvRMSE.ann)),
                         mean(unlist(cvRMSE.tree)),
                         mean(unlist(cvRMSE.ens))))
```

```
##   model mean.rmse
## 1    lm 2.0233464
## 2   ann 1.2169614
## 3  tree 1.0007608
## 4   ens 0.3941865
```

As the table above, the ensemble model has the lowest mean RMSE from the cross validation, meaning it does improved the overall performance.