**Order of files:**

| Filename | Pages | Comments and/or Instructions |
|---|---|---|
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

**Additional Comments:**

Kaggle name: Bike Sharing Demand

Total number of teams on the leaderboard: 3242

Position on the leaderboard at the time of your last submission:  1446

Github: https://github.com/anushangeri/Shangeri_MMA-867_Assignment-1.git

*github contains the Kaggle datasets, R script, predicted value submitted on Kaggle and report*

# 3 Competition Choices

## Option 1: House Prices: Advanced Regression Techniques

Predict sales prices and practice feature engineering, RFs, and gradient boosting. With 79 explanatory variables describing (almost) every aspect of residential homes in Ames, Iowa, this competition challenges you to predict the final price of each home.

Number of entrants: 4751

Status: Ongoing

Link to Kaggle competition: https://www.kaggle.com/c/house-prices-advanced-regression-techniques

## Option 2: New York City Taxi Trip Duration

Predicts the total ride duration of taxi trips in New York City. Your primary dataset is one released by the NYC Taxi and Limousine Commission, which includes pickup time, geo-coordinates, number of passengers, and several other variables.

Number of entrants: 1254

Status: Completed 3 years ago

Link to Kaggle competition: https://www.kaggle.com/c/nyc-taxi-trip-duration/data

## Option 3: Bike Sharing Demand

Participants are asked to combine historical usage patterns with weather data in order to forecast bike rental demand in the Capital Bikeshare program in Washington, D.C.

Number of entrants: 3242

Status: Completed 5 years ago

Link to Kaggle competition: https://www.kaggle.com/c/bike-sharing-demand/data

## Final choice chosen:

Option 3 was chosen - Bike Sharing Demand because it required feature engineering and advanced regression techniques. This competition provided good practice and exposure for predictive modelling, for example, the usage of Extreme Gradient Boosting. This competition requires a predictive regression model to predict the number of bike rental for a given date, season, holiday, working day, weather, temperature, "feels like" temperature, humidity and wind speed. The following section will be a detailed description of how the predictive model was built. Root Mean Squared Log Error (RMSLE) will be used to evaluate the actual versus predicted.

# Building the regression model

## Understanding the data

To begin, the data was first checked for missing data (there was no missing data) and the datetime column in the train and test set was spilt into hour, day and month columns to see if these variables explain the number of bike rentals better.

Before building the regression model, a simple graph was plotted to understand any trends in the data. It could be possible that season affects the number of bike rentals. To be sure, the number of bikes was plotted against each hour of the day for each season as seen in Figure 2.1 below.
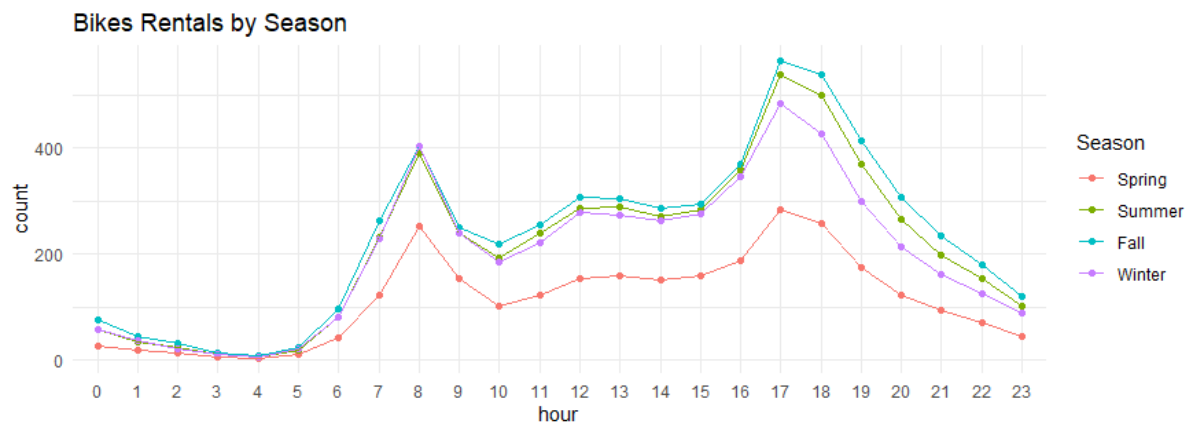
*Figure2.1: Line graph of bike rentals against hour by season.*

From the graph, we understand there are more bike rental in morning, from the 7th hour and in the evening from the 17th to 18th hour. Furthermore, people rent bikes more in fall, summer and winter, and much less in spring. We will bear variable season and hour in mind when building the model later.

## Building the model

Target RMSLE: 0.49644 (position 1625)

The above RMSLE score is the average score on Kaggle for this competition.

## Step 1: separating train dataset to 70% for model building and 30% for testing

The following is a breakdown of the variables and their description:

- datetime - hourly date + timestamp
- season -  1 = spring, 2 = summer, 3 = fall, 4 = winter
- holiday - whether the day is considered a holiday
- workingday - whether the day is neither a weekend nor holiday
- weather
    - 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 - temperature in Celsius
- atemp - "feels like" temperature in Celsius
- humidity - relative humidity
- windspeed - wind speed
- **count - number of total rentals (this is the dependent variable to be predicted)**

## Step 2: Start modelling with all factors

Model: fit<-lm(count~., train_data)
RMSE: 108.6011

```
> fit<-lm(count~., train_data)
> #test model on the training target (30%) to see how good it is
> predicted.rentals.testing<-predict(fit, train_target)
Warning message:
In predict.lm(fit, train_target) :
  prediction from a rank-deficient fit may be misleading
> #score: 108.6011
> rmse(train_target$count,predicted.rentals.testing)
[1] 108.6011
```

*Figure 2.2: RMSLE output for count~.*

The RMSE score seems to be high. In the case of RMSE, the presence of outliers can explode the error term to a very high value. Although, it is better to over specify than under specify the model, outliers may influence bias. As much as possible, all data points should be accounted for in the model if possible. There could be a possibility of outliers in this dataset, however, that can be revisited later on if a good enough RMSLE score cannot be achieve.

## Step 3: Log the dependent variable

Model: logfit<-lm(log(count)~., train_data)

RMSLE: 0.613963

Although an improvement, the RMSLE score is not good enough.

```
> logfit<-lm(log(count)~., train_data)
> #test model on the training target (30%) to see how good it is
> predicted.rentals.testing<-predict(logfit, train_target)
Warning message:
In predict.lm(logfit, train_target) :
  prediction from a rank-deficient fit may be misleading
> predicted.rentals.testing.nonlog <- exp(predicted.rentals.testing)
> #score: 0.613963
> rmsle(train_target$count,predicted.rentals.testing.nonlog)
[1] 0.613963
```

*Figure 2.3: RMSLE output for log(count)~.*

## Step 4: Using Step AIC to get a better model

Model: fit<-lm(log(count)~., train_data)

logfitAIC <- stepAIC(fit, direction = 'both')

RMSLE: 0.6140443

```
Step:  AIC=-6044.21
log(count) ~ holiday + weather + temp + atemp + humidity + windspeed +
    hour + day + month

            Df Sum of Sq      RSS      AIC
- holiday    1       0.1   3403.3  -6046.0
<none>                     3403.2  -6044.2
- atemp      1       2.0   3405.2  -6041.7
- temp       1       7.9   3411.1  -6028.5
- windspeed  1       8.8   3412.0  -6026.5
- humidity   1      30.6   3433.8  -5978.0
- day        6      37.3   3440.5  -5973.0
- weather    3     136.7   3539.9  -5750.0
- month     11     300.4   3703.6  -5421.5
- hour      23    8440.6  11843.8   3412.6

Step:  AIC=-6046.04
log(count) ~ weather + temp + atemp + humidity + windspeed +
    hour + day + month

             Df Sum of Sq      RSS      AIC
<none>                      3403.3  -6046.0
+ workingday  1       0.1   3403.2  -6044.2
+ holiday     1       0.1   3403.2  -6044.2
- atemp       1       2.0   3405.3  -6043.6
- temp        1       8.0   3411.3  -6030.1
- windspeed   1       8.9   3412.1  -6028.2
- humidity    1      30.6   3433.8  -5979.9
- day         6      37.3   3440.6  -5974.9
- weather     3     136.8   3540.1  -5751.7
- month      11     300.4   3703.6  -5423.5
- hour       23    8440.7  11844.0   3410.8
> #test model on the training target (30%) to see how good it is
> predicted.rentals.testing<-predict(logfitAIC, train_target)
> predicted.rentals.testing.nonlog <- exp(predicted.rentals.testing)
> #score: 0.6140443 - ok this model is worst so we use log(count)~. for now
> rmsle(train_target$count,predicted.rentals.testing.nonlog)
[1] 0.6140443
```

*Figure 2.4: RMSLE output for stepAIC() on model log(count)~.*

The stepAIC() function from the MASS package to get a better model. The stepAIC() function performs model selection by starting from a "maximal" model, which is then trimmed down to a model with the independent variables that best explain the dependent variable. However, based on the RMSLE, the "improved" model is worst than log(count) ~. in Step 3. Continue with the model in step 3.

## Step 5: Using train() from Caret package

Model: logfit2 <- train(log(count)~., train_data, method="xgbLinear", trControl = ctrl)
RMSLE: 0.422968
Kaggle Score: 0.52418 [position 1902]

```
> #let's use train in caret package with the log(count)~.
> ctrl <- trainControl(method = "repeatedcv",
+                      number = 3,
+                      repeats = 3,
+                      verboseIter = FALSE,
+                      allowParallel = TRUE)
> #xgbLinear from XGBoost
> logfit2 <- train(log(count)~., train_data,
+                  method="xgbLinear", trControl = ctrl)
> #test model on the training target (30%) to see how good it is
> predicted.rentals.testing<-predict(logfit2, train_target)
> predicted.rentals.testing.nonlog <- exp(predicted.rentals.testing)
> #score: 0.422968 - ok this model IS GOODDD
> rmsle(train_target$count,predicted.rentals.testing.nonlog)
[1] 0.422968
```

*Figure 2.5: RMSLE using train() on log(count)~.*

To aid in the predictive modelling process, the train function in the Caret package was used. The method xbgLinear of XGBoost was used to build a new regression model. Gradient boosting is an approach where instead of training the models in isolation of one another like in the previous steps, each new model is trained to predict and correct the residuals (errors) made by the previous model. The models are then added together to make the final prediction. Furthermore, as tuning parameters were added to get a better average error term. All in all, a good RMSLE score was achieved so far.

So this new model was used to predict the test dataset to be submitted on Kaggle. However, the score is not good enough to reach the target RMSLE: 0.49644 (position 1625).

## Step 6: Remove the outliers

To improve the model, the train dataset was plotted to see if there are any influential outliers to be removed. As shown from the plot below (Figure 2.2), there are some outliers that can potentially be removed to build a better model. These outliers were removed.
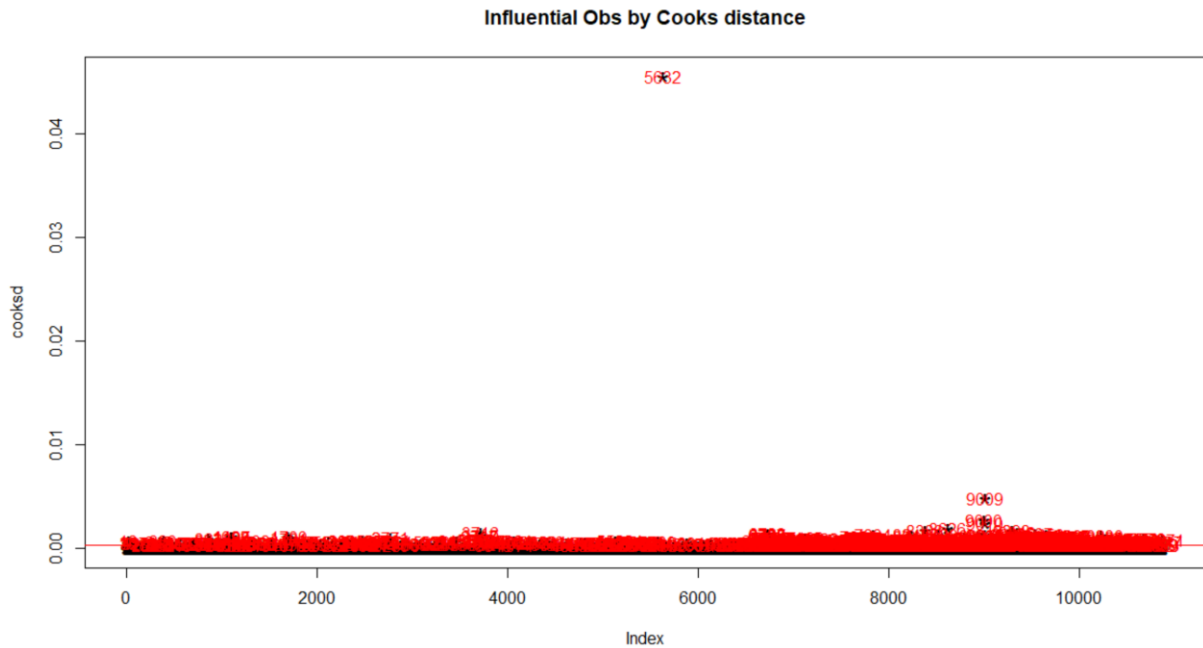
**Influential Obs by Cooks distance**



*Figure 2.6: Plot of outliers in the train dataset*

## Step 7: Using XGBoost (xgblinear) on train dataset with no outliers

Model: logfit2 <- train(log(count)~., train_data, method="xgbLinear", trControl = ctrl)

RMSLE: 0.4237904

Kaggle Score: 0.50459

The RMSLE and Kaggle score improvement significantly but not good enough.

```
> train <- train[-outliers.located, ]
> #separating train dataset to 70% for model building and 30% for testing
> set.seed(1)
> sample <- sample.int(n = nrow(train), size = floor(.7*nrow(train)), replace = F)
> train_data <- train[sample, ]
> train_target  <- train[-sample, ]
> #xgbLinear from XGBoost #new best!
> logfit2 <- train(log(count)~., train_data,
+                  method="xgbLinear", trControl = ctrl)
> #test model on the training target (30%) to see how good it is
> predicted.rentals.testing<-predict(logfit2, train_target)
> predicted.rentals.testing.nonlog <- exp(predicted.rentals.testing)
> #score: 0.4237904 - ok this model looks better
> rmsle(train_target$count,predicted.rentals.testing.nonlog)
[1] 0.4237904
```

*Figure 2.7: RMSLE using train() after removing outliers*

## Step 8: Using stepAIC() to find a better model

fit<-lm(log(count)~., train_data)

logfitAIC <- stepAIC(fit, direction = 'both')

The new model chosen from the stepAIC() function is as follows:

log(count) ~ season + holiday + workingday + weather + temp + humidity + windspeed + hour + day + month

## Step 9: Training the new model [Caret package]

Model: logfit2 <- train(log(count) ~ season + holiday + workingday + weather + temp + humidity + windspeed + hour + day + month, train_data, method="xgbLinear", trControl = ctrl)

RMSLE: 0.4231934

```
> #ok let's train
> logfit2 <- train(log(count) ~ season + holiday + workingday + weather + temp +
+                   humidity + windspeed + hour + day + month, train_data,
+                   method="xgbLinear", trControl = ctrl)
> #test model on the training target (30%) to see how good it is
> predicted.rentals.testing<-predict(logfit2, train_target)
> predicted.rentals.testing.nonlog <- exp(predicted.rentals.testing)
> #score: 0.4231934 - ok this model is a little better but not good enough
> rmsle(train_target$count,predicted.rentals.testing.nonlog)
[1] 0.4231934
```

*Figure 2.8: RMSLE output after using stepAIC() and train()*

While the RMSLE improved, it is not good enough.

## Step 10: Log the season and hour variable

Model: logfit2 <- train(log(count) ~ log(as.numeric(season)) + holiday + workingday + weather + temp + humidity + windspeed + log(as.numeric(hour)) + day + month, train_data, method="xgbLinear", trControl = ctrl)

RMSLE: 0.3432964

```
> #ok let's train again
> logfit2 <- train(log(count) ~ log(as.numeric(season)) + holiday + workingday + weather + temp +
+                   humidity + windspeed + log(as.numeric(hour)) + day + month, train_data,
+                   method="xgbLinear", trControl = ctrl)
> #test model on the training target (30%) to see how good it is
> predicted.rentals.testing<-predict(logfit2, train_target)
> predicted.rentals.testing.nonlog <- exp(predicted.rentals.testing)
> #score: 0.3432964 - ok this model IS GOODDD
> rmsle(train_target$count,predicted.rentals.testing.nonlog)
[1] 0.3432964
```

*Figure 2.9: RMSLE output using train() after logging season and hour*

Kaggle Score: 0.49247 [position 1446 on public and private leaderboard]

Target score achieved with this model. Since the competition is completed, the score does not show up on the private or public leaderboard, but nonetheless, the "would be" position on the Kaggle public and private leaderboard is 1446 (shown in Figure 2.3) which is better that the average target score – 0. 49644.

# Appendix

## Final Kaggle Score



*Appendix 1: Final Kaggle score: 0.49247*



*Appendix 2: Kaggle public leaderboard with "would be" positions highlighted*

## R Script

```r
1  library("readxl")
2  library("lubridate")
3  library("proto")
4  library("gsubfn")
5  library("RSQLite")
6  library("sqldf")
7  library("tidyverse")
8  library("ggplot2")
9  library("mice")
10 library("caret")
11 library("dplyr")
12 library("tidyr")
13 library("MASS")
14 library("car")
15 library("Metrics")
16 library("glmnet")
17 library("xgboost")
18 library("gbm")
19 library("mboost")
20
21 train <- read.csv("./Assignment//Individual/bike-sharing-demand//train.csv")
22 test  <- read.csv("./Assignment//Individual/bike-sharing-demand//test.csv")
23 #Duplicate the test data to save the datetime for later
24 to.predict <- test
25
26 #step 1: lets look at the variables we are dealing with
27 head(train)
28 head(test)
29
30 #columns "casual" and "registered" is not required in the model, so let's remove that from the train
```

```r
28 head(test)
29
30 #columns "casual" and "registered" is not required in the model, so let's remove that from the train
31 train = train[,!(names(train) %in% c("casual"))]
32 train = train[,!(names(train) %in% c("registered"))]
33
34 #Step 2: check for missing values
35 md.pattern(train)
36 md.pattern(test)
37 #no missing data!
38
39 #Step 3: Converting integer to factor
40 # #training set
41 train$season <- as.factor(train$season)
42 train$holiday <- as.factor(train$holiday)
43 train$workingday <- as.factor(train$workingday)
44 train$weather <- as.factor(train$weather)
45
46 #test set
47 test$season <- as.factor(test$season)
48 test$holiday <- as.factor(test$holiday)
49 test$workingday <- as.factor(test$workingday)
50 test$weather <- as.factor(test$weather)
51 #
52 #Step 4: let's work with the train set
53 #Deriving day, hour from datetime field
54 train$datetime <- ymd_hms(train$datetime)
55 train$hour <- hour(train$date)
56 train$day <- wday(train$date)
57 train$month <- month(train$date, label=T)
```

```r
58
59  #Deriving day, hour from datetime field
60  test$datetime <- ymd_hms(test$datetime)
61  test$hour <- hour(test$date)
62  test$day <- wday(test$date)
63  test$month <- month(test$date, label=T)
64
65  str(train)
66  names(train)
67
68  str(test)
69  names(test)
70
71  train[,11:13]<-lapply(train[,11:13], factor) #converting derived variables into factors
72  test[,10:12]<-lapply(test[,10:12], factor) #converting derived variables into factors
73
74  #Step 4: Removing datetime field
75  train$datetime <- NULL
76  colnames(train)
77  str(train)
78
79
80  #Removing the data field for test
81  test$datetime <- NULL
82
83  #Step 5: visualization so we can understand the data
84  season_summary_by_hour <- sqldf('select season, hour, avg(count) as count from train group by season, hour')
85
86  #There are more rental in morning(from 7 hour) and evening(17-18th hour)
87  #People rent bikes more in Fall Summer and Winter, and much less in Spring
```

```r
86  #There are more rental in morning(from 7 hour) and evening(17-18th hour)
87  #People rent bikes more in Fall Summer and Winter, and much less in Spring
88  plot <- ggplot(train, aes(x=hour, y=count, color=season))+
89    geom_point(data = season_summary_by_hour, aes(group = season))+
90    geom_line(data = season_summary_by_hour, aes(group = season))+
91    ggtitle("Bikes Rentals by Season")+ theme_minimal()+
92    scale_colour_hue('Season',breaks = levels(train$season),
93                  labels=c('Spring', 'Summer', 'Fall', 'Winter'))
94  plot
95
96  #Step 6: separating train dataset to 70% for model building and 30% for testing
97  set.seed(1)
98  sample <- sample.int(n = nrow(train), size = floor(.7*nrow(train)), replace = F)
99  train_data <- train[sample, ]
100 train_target  <- train[-sample, ]
101
102 #Step 7: Start modeling with all factors
103 #build a model on training data using all variables (the 70%)
104 fit<-lm(count~., train_data)
105 #test model on the training target (30%) to see how good it is
106 predicted.rentals.testing<-predict(fit, train_target)
107 #score: 108.6011
108 rmse(train_target$count,predicted.rentals.testing)
109
110 #let's log(count)
111 logfit<-lm(log(count)~., train_data)
112 #test model on the training target (30%) to see how good it is
113 predicted.rentals.testing<-predict(logfit, train_target)
114 predicted.rentals.testing.nonlog <- exp(predicted.rentals.testing)
115 #score: 0.613963
```

```r
116  rmsle(train_target$count,predicted.rentals.testing.nonlog)
117
118  #using Step AIC
119  logfitAIC <- stepAIC(logfit, direction = 'both')
120  #test model on the training target (30%) to see how good it is
121  predicted.rentals.testing<-predict(logfitAIC, train_target)
122  predicted.rentals.testing.nonlog <- exp(predicted.rentals.testing)
123  #score: 0.6140443 - ok this model is worst so we use log(count)~. for now
124  rmsle(train_target$count,predicted.rentals.testing.nonlog)
125
126  #let's use train in caret package with the log(count)~.
127  ctrl <- trainControl(method = "repeatedcv",
128                       number = 3,
129                       repeats = 3,
130                       verboseIter = TRUE,
131                       allowParallel = TRUE)
132  #xgbLinear from XGBoost
133  logfit2 <- train(log(count)~., train_data,
134                   method="xgbLinear", trControl = ctrl)
135  #test model on the training target (30%) to see how good it is
136  predicted.rentals.testing<-predict(logfit2, train_target)
137  predicted.rentals.testing.nonlog <- exp(predicted.rentals.testing)
138  #score: 0.422968 - ok this model IS GOODDD
139  rmsle(train_target$count,predicted.rentals.testing.nonlog)
140
141  #LET'S PREDICT
142  predicted.rentals.final<-predict(logfit2, test)
143  predicted.rentals.final.nonlog <- exp(predicted.rentals.final)
144  predicted.rentals = data.frame(datetime = to.predict$datetime, count = predicted.rentals.final.nonlog)
145  colnames(predicted.rentals) <- c("datetime", "count")
```

```r
145  predicted.rentals.final.nonlog <- exp(predicted.rentals.final)
144  predicted.rentals = data.frame(datetime = to.predict$datetime, count = predicted.rentals.final.nonlog)
145  colnames(predicted.rentals) <- c("datetime", "count")
146  write.csv(predicted.rentals, "predicted_rentals.csv", row.names=FALSE)
147  #score on Kaggle 0.52418 [position 1902]
148
149  #attempt 2
150  #let's see restart and see if there is outliers
151  fit<-lm(count~., train)
152  plot(density(resid(fit)))
153  sample_size <- nrow(train)
154  cooksd <- cooks.distance(fit)
155  #plot cook's distance
156  plot(cooksd, pch="*", cex=2, main="Influential Obs by Cooks distance")
157  #add cutoff line
158  abline(h = 4/sample_size, col="red")
159  #add labels
160  text(x=1:length(cooksd)+1, y=cooksd, labels=ifelse(cooksd>4/sample_size, names(cooksd),""), col="red")
161
162  #yes there is, remove them
163  outliers.located <- c(9000, 9009, 9010, 9008, 9011, 8334,
164                        8331, 8332, 8335, 8307, 8333, 8312,5662)
165  #remove the outliers and try modelling again
166  train <- train[-outliers.located, ]
167
168  #separating train dataset to 70% for model building and 30% for testing
169  set.seed(1)
170  sample <- sample.int(n = nrow(train), size = floor(.7*nrow(train)), replace = F)
171  train_data <- train[sample, ]
172  train_target  <- train[-sample, ]
```

```r
#xgbLinear from XGBoost #new best!
logfit2 <- train(log(count)~., train_data,
                 method="xgbLinear", trControl = ctrl)
#test model on the training target (30%) to see how good it is
predicted.rentals.testing<-predict(logfit2, train_target)
predicted.rentals.testing.nonlog <- exp(predicted.rentals.testing)
#score: 0.4237904 - ok this model looks better
rmsle(train_target$count,predicted.rentals.testing.nonlog)
#LET'S PREDICT
predicted.rentals.final<-predict(logfit2, test)
predicted.rentals.final.nonlog <- exp(predicted.rentals.final)
predicted.rentals = data.frame(datetime = to.predict$datetime, count = predicted.rentals.final.nonlog)
colnames(predicted.rentals) <- c("datetime", "count")
write.csv(predicted.rentals, "predicted_rentals.csv", row.names=FALSE)
#score on Kaggle 0.50459 omgggg it got better!

#using stepAIC to find a better model
fit<-lm(log(count)~., train_data)
logfitAIC <- stepAIC(fit, direction = 'both')

#ok let's train
logfit2 <- train(log(count) ~ season + holiday + workingday + weather + temp +
                     humidity + windspeed + hour + day + month, train_data,
                 method="xgbLinear", trControl = ctrl)
#test model on the training target (30%) to see how good it is
predicted.rentals.testing<-predict(logfit2, train_target)
predicted.rentals.testing.nonlog <- exp(predicted.rentals.testing)
#score: 0.4231934 - ok this model is a little better but not good enough
rmsle(train_target$count,predicted.rentals.testing.nonlog)


#lets's log the season and hour
fit<-lm(log(count) ~ log(as.numeric(season)) + holiday + workingday + weather + temp +
            humidity + windspeed + log(as.numeric(hour)) + day + month, train_data)
plot(fit)
#log helps, we get a normal curve
plot(density(resid(fit)))

#ok let's train again
logfit2 <- train(log(count) ~ log(as.numeric(season)) + holiday + workingday + weather + temp +
                     humidity + windspeed + log(as.numeric(hour)) + day + month, train_data,
                 method="xgbLinear", trControl = ctrl)
#test model on the training target (30%) to see how good it is
predicted.rentals.testing<-predict(logfit2, train_target)
predicted.rentals.testing.nonlog <- exp(predicted.rentals.testing)
#score: 0.3432964 - ok this model IS GOODDD
rmsle(train_target$count,predicted.rentals.testing.nonlog)
#LET'S PREDICT
predicted.rentals.final<-predict(logfit2, test)
predicted.rentals.final.nonlog <- exp(predicted.rentals.final)
predicted.rentals = data.frame(datetime = to.predict$datetime, count = predicted.rentals.final.nonlog)
colnames(predicted.rentals) <- c("datetime", "count")
write.csv(predicted.rentals, "predicted_rentals.csv", row.names=FALSE)
#score on Kaggle 0.49247 omgggggggggg [position: 1446]
```