Queen's MASTER OF MANAGEMENT ANALYTICS

**MMA 867**

**Predictive Modelling**

**Anton Ovchinnikov**

**Assignment 1 - Section 2**

**May 3, 2020 11:59 PM**

**Shangeri Sivalingam**

## 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

# 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.

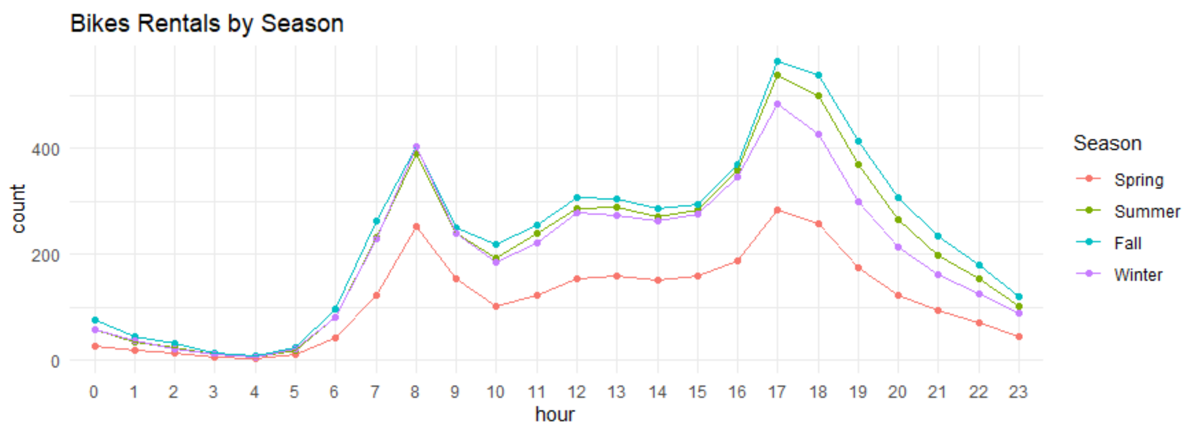



*Figure2.1: Line graph of bike rentals against hour by season.*

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
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.

### 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
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]

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.

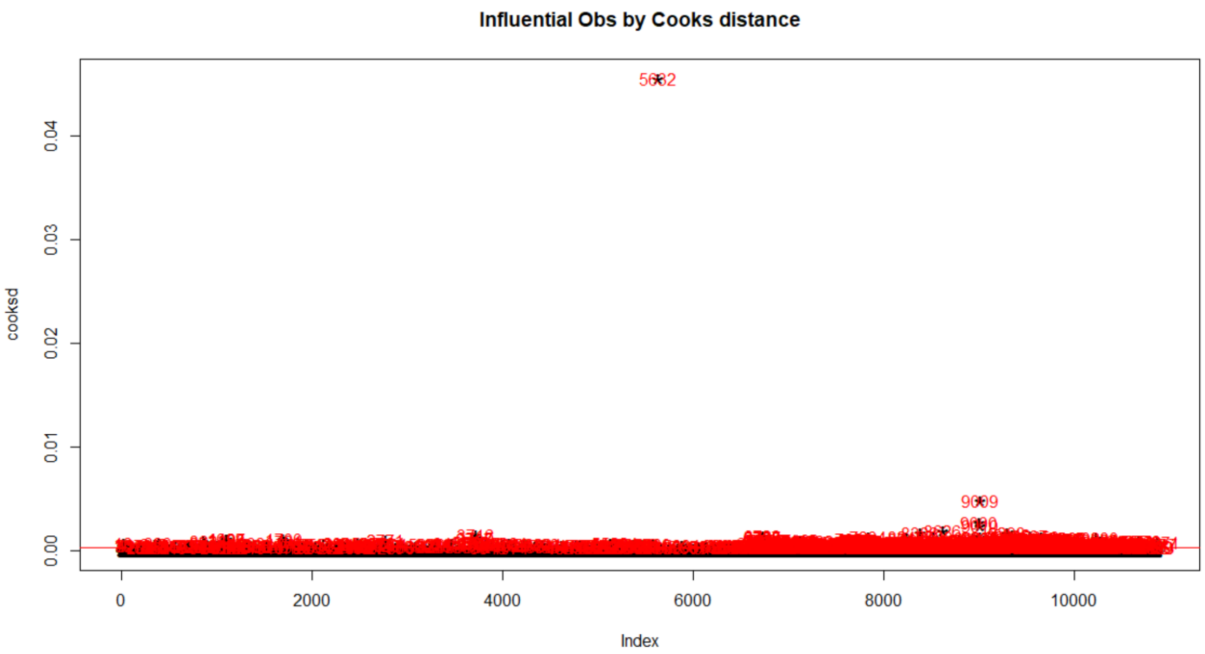



*Figure 2.2: Plot of outliers in the train dataset*

## 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.

## 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
While the RMSLE improved, it is not good enough.

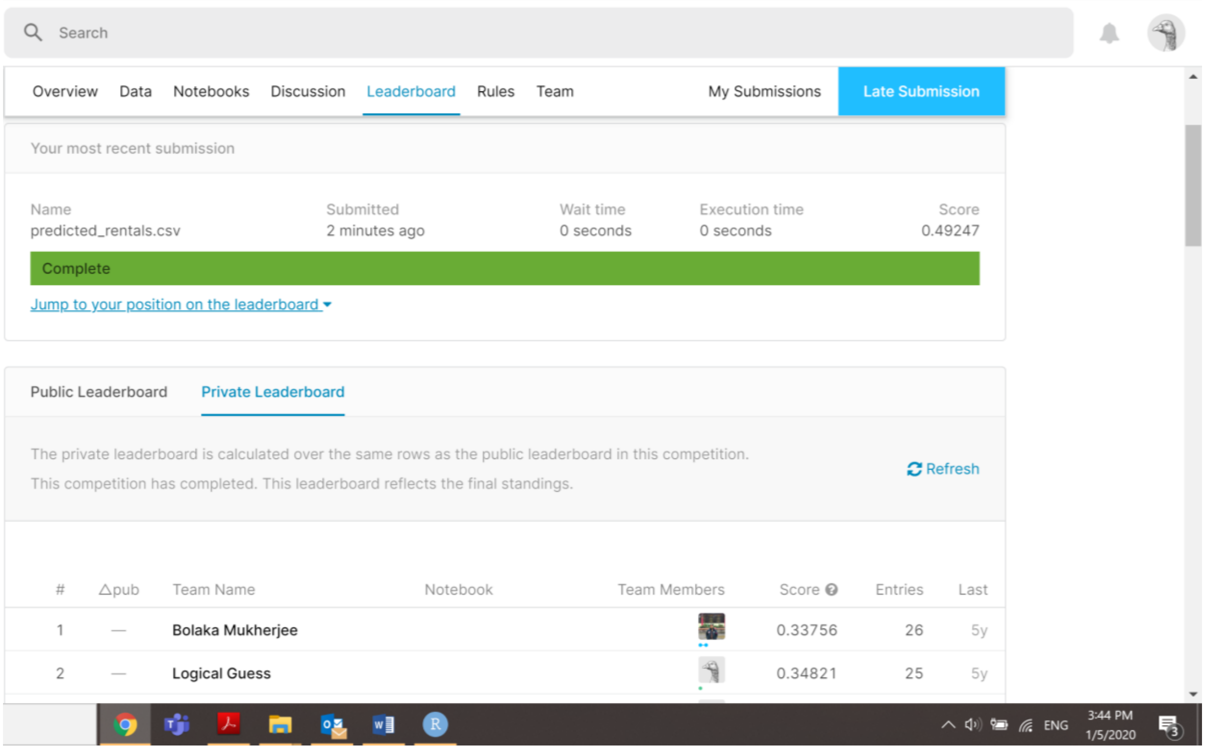
## Step 10: Log the season and hour variable

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
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

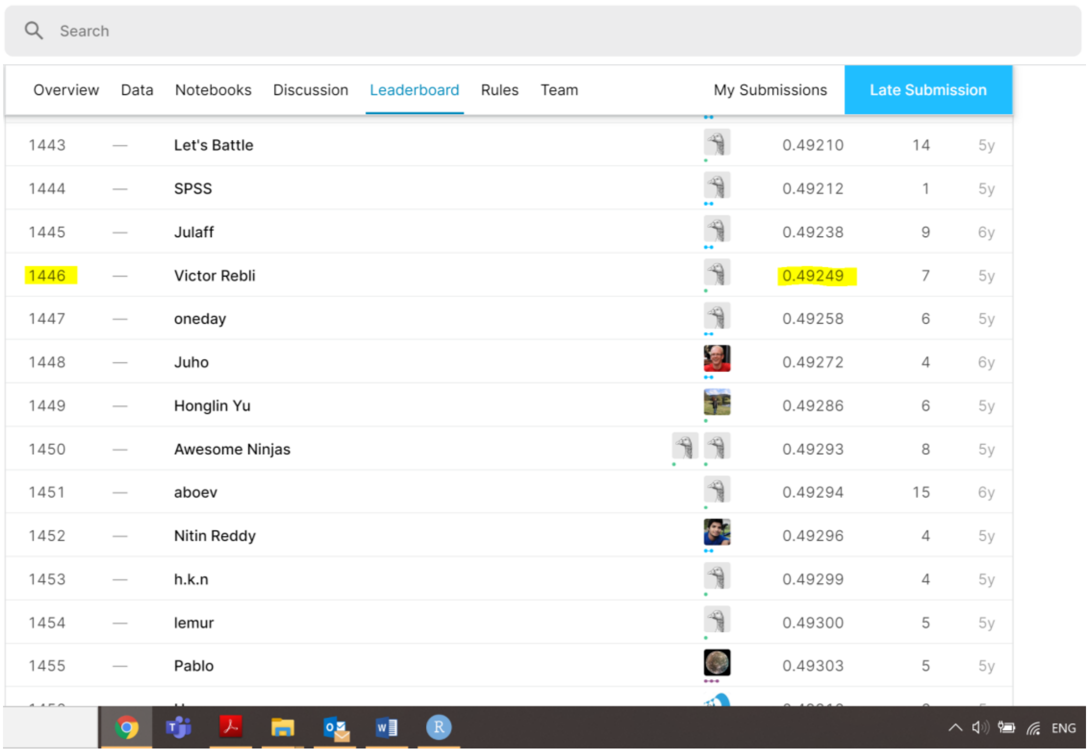
public and private leaderboard is 1446 (shown in Figure 2.3) which is better that the average target score – 0. 49644.

# Appendix

## Final Kaggle Score




*Appendix 1: Final Kaggle score: 0.49247*




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

## R Script

```r
library("readxl")
library("lubridate")
library("proto")
library("gsubfn")
library("RSQLite")
library("sqldf")
library("tidyverse")
library("ggplot2")
library("mice")
library("caret")
library("dplyr")
library("tidyr")
library("MASS")
library("car")
library("Metrics")
library("glmnet")
library("xgboost")
library("gbm")
library("mboost")

train <- read.csv("./Assignment//Individual/bike-sharing-demand//train.csv")
test  <- read.csv("./Assignment//Individual/bike-sharing-demand//test.csv")
#Duplicate the test data to save the datetime for later
to.predict <- test

#step 1: lets look at the variables we are dealing with
head(train)
head(test)

#columns "casual" and "registered" is not required in the model, so let's remove that from the train
train = train[,!(names(train) %in% c("casual"))]
train = train[,!(names(train) %in% c("registered"))]

#Step 2: check for missing values
md.pattern(train)
md.pattern(test)
#no missing data!

#Step 3: Converting integer to factor
# #training set
train$season <- as.factor(train$season)
train$holiday <- as.factor(train$holiday)
train$workingday <- as.factor(train$workingday)
train$weather <- as.factor(train$weather)

#test set
test$season <- as.factor(test$season)
test$holiday <- as.factor(test$holiday)
test$workingday <- as.factor(test$workingday)
test$weather <- as.factor(test$weather)
#
#Step 4: let's work with the train set
#Deriving day, hour from datetime field
train$datetime <- ymd_hms(train$datetime)
train$hour <- hour(train$date)
train$day <- wday(train$date)
train$month <- month(train$date, label=T)
```

```r
#Deriving day, hour from datetime field
test$datetime <- ymd_hms(test$datetime)
test$hour <- hour(test$date)
test$day <- wday(test$date)
test$month <- month(test$date, label=T)

str(train)
names(train)

str(test)
names(test)

train[,11:13]<-lapply(train[,11:13], factor) #converting derived variables into factors
test[,10:12]<-lapply(test[,10:12], factor) #converting derived variables into factors

#Step 4: Removing datetime field
train$datetime <- NULL
colnames(train)
str(train)


#Removing the data field for test
test$datetime <- NULL

#Step 5: visualization so we can understand the data
season_summary_by_hour <- sqldf('select season, hour, avg(count) as count from train group by season, hour')

#There are more rental in morning(from 7 hour) and evening(17-18th hour)
#People rent bikes more in Fall Summer and Winter, and much less in Spring
```


```r
#There are more rental in morning(from 7 hour) and evening(17-18th hour)
#People rent bikes more in Fall Summer and Winter, and much less in Spring
```


```r
plot <- ggplot(train, aes(x=hour, y=count, color=season))+
  geom_point(data = season_summary_by_hour, aes(group = season))+
  geom_line(data = season_summary_by_hour, aes(group = season))+
  ggtitle("Bikes Rentals by Season")+ theme_minimal()+
  scale_colour_hue('Season',breaks = levels(train$season),
                   labels=c('Spring', 'Summer', 'Fall', 'Winter'))
plot

#Step 6: 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, ]

#Step 7: Start modeling with all factors
#build a model on training data using all variables (the 70%)
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)
#score: 108.6011
rmse(train_target$count,predicted.rentals.testing)

#let's log(count)
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)
predicted.rentals.testing.nonlog <- exp(predicted.rentals.testing)
#score: 0.613963
```

```
rmsle(train_target$count,predicted.rentals.testing.nonlog)

#using Step AIC
logfitAIC <- stepAIC(logfit, direction = 'both')
#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)

#let's use train in caret package with the log(count)~.
ctrl <- trainControl(method = "repeatedcv",
                     number = 3,
                     repeats = 3,
                     verboseIter = TRUE,
                     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)

#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")
```

```
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 0.52418 [position 1902]

#attempt 2
#let's see restart and see if there is outliers
fit<-lm(count~., train)
plot(density(resid(fit)))
sample_size <- nrow(train)
cooksd <- cooks.distance(fit)
#plot cook's distance
plot(cooksd, pch="*", cex=2, main="Influential Obs by Cooks distance")
#add cutoff line
abline(h = 4/sample_size, col="red")
#add labels
text(x=1:length(cooksd)+1, y=cooksd, labels=ifelse(cooksd>4/sample_size, names(cooksd),""), col="red")

#yes there is, remove them
outliers.located <- c(9000, 9009, 9010, 9008, 9011, 8334,
              8331, 8332, 8335, 8307, 8333, 8312,5662)
#remove the outliers and try modelling again
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$count,predicted.rentals.testing.nonlog)
#LET'S PREDICT
predicted.rentals.final<-predict(logfit2, test)
predicted.rentals.final.nonlog <- exp(predicted.rentals.final)
predicted.rentals = data.frame(datetime = to.predict$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 omggggggggg [position: 1446]
```