# PREDICTING HOUSEHOLD UTILITY USAGE WITH GENERALIZED BOOSTED REGRESSION MODELING (GBM)

by Serena Alvarez

# DATASET

- I will be using the `econ128` dataset provided.

- The variables are as follows:
  - **hh_id**: unique household id
  - **year**: 2010 and 2011
  - **month**: 4-8
  - **zipcode**: anonymized zip code in which home is located
  - **control**: household-month is part of control group
  - **treatment**: household-month is part of treatment group
  - **children**: household has children
  - **hhsize2-5plus**: household size
  - **income2-9**: income categories <$20k, $20-30k, $30-40k, $40-50k, $50-75k, $75-100k,$100-125k, >$125k
  - **owner**: resident owns home

# CLEANING THE DATA

Dimensions of the raw data:

```
## [1] 234560     28
```

To clean the data, I decided to do three things:

- Remove rows with NA

- Convert dummy variables to factors (`hhsize`, `income`)

- Remove outlier of `luse1`

# REMOVE NA

I decided to remove all rows that contain `NA`:

(with this amount of data, removing it should be fine)

New dimensions:

```
## [1] 221520     28
```

We removed around 5.5% of the observations

# DUMMIES TO FACTORS

The dummy variables were somewhat unclear when looking at, so I converted them to factors

New column headings:

```
##  [1] "hh_id"     "year"      "month"
"zipcode"   "control"   "treatment"

##  [7] "lusage"    "luse1"     "luse2"
"luse3"     "luse4"     "luse5"

## [13] "luse6"     "children"  "owner"
"hhsize"    "income"
```

# REMOVE OUTLIER(S)

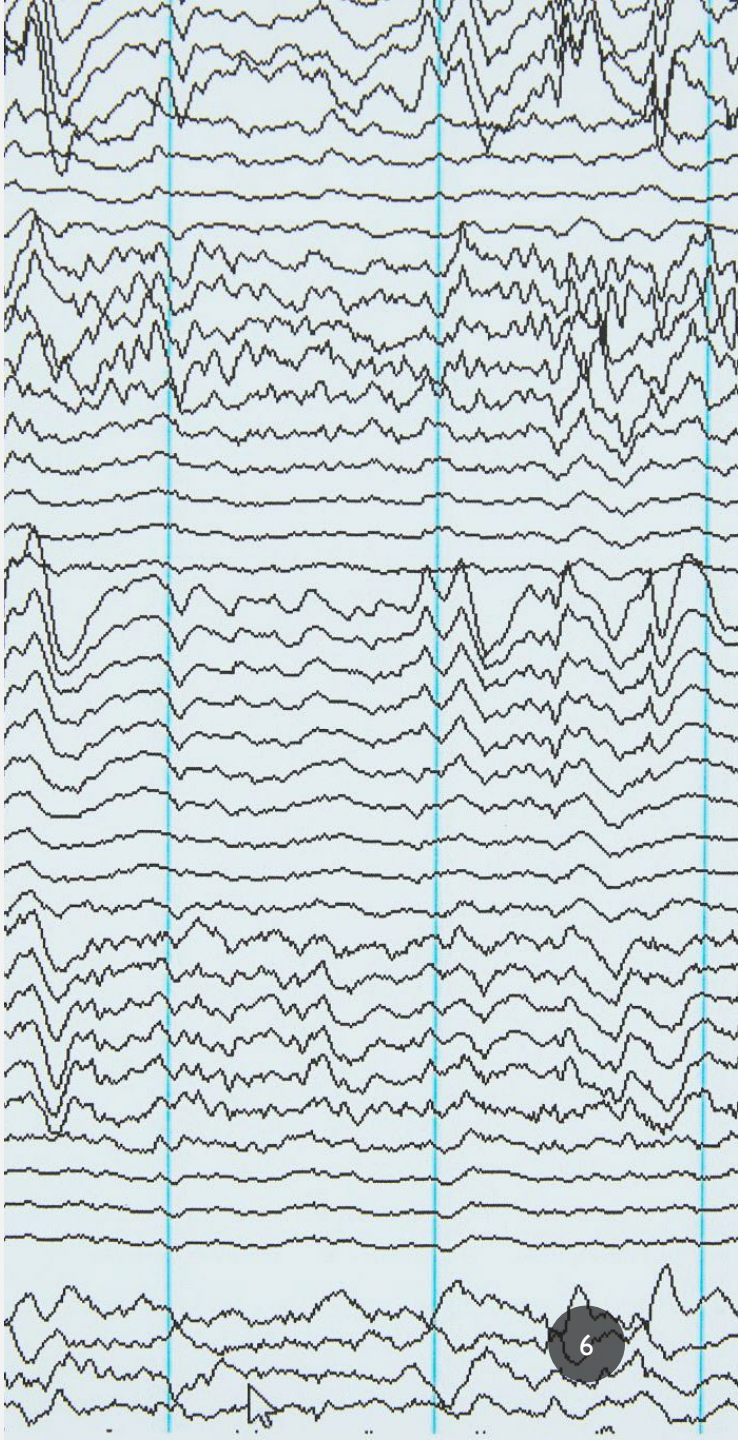
After checking summary stats of the continuous variables, we remove the rows with negative values of `luse1`.

(assume outlier/ entered incorrectly)

New dimensions:

```
## [1] 221510     17
```

We removed a further 10 observations



# ABOUT THE MODEL

- For this project I will be using the gbm() function from the gbm package to produce a Generalized Boosted Regression Modeling (GBM).
- The variable `lusage` will be modeled by using all the variables except for `hh_id`, `year`, and `month`.
- We will use the distribution = "gaussian" option to indicate that we are looking to minimize MSE
- The total number trees to be fit is 5000 and the depth between interactions is set to 3

# MODEL BUILDING

First, I subsetted the training and test data according to year:

```r
#control group
control_2010 <- econ128 %>% filter(control == 1, year == 2010)
control_2011 <- econ128 %>% filter(control ==1, year == 2011)

#treatment group
treatment_2010 <- econ128 %>% filter(control == 0, year == 2010)
treatment_2011 <- econ128 %>% filter(control == 0, year == 2011)
```

# MODEL BUILDING

I decided to use the **Boosting** machine learning method to predict. I found boosting to be better than random forests and bagging thus I found it appropriate to use here. The model trained by **2010 control data** is modeled as follows:

```
```{r, warning=FALSE}
#Be warned: takes forever to load
boost.econ128 <- gbm(lusage ~ hhsize + income + luse1 +luse2 +luse3 +luse4 +luse5 +luse6+ owner + zipcode,
                     data = control_2010,
                     distribution = "gaussian",
                     n.trees = 5000,
                     interaction.depth = 3)
```
```

# PREDICTIONS- CONTROL 2011

I then used the model to predict the **2011 control values** for `lusage`:

```
```{r,warning=FALSE}
yhat.econ128 <- predict(boost.econ128,
    newdata =control_2011, n.trees = 5000)
```
```

Resulting in a low MSE, so we move forward to predicting treatment 2011.

```
## [1] 0.1237311
```

# PREDICTIONS- TREATMENT 2011

Now we use our model to predict values for the **treatment group in 2011**: ( 5 predicted values represented as (cv.fold = 5))

```
```{r}
yhat.econ128_treatment <- predict(boost.econ128,
    newdata =treatment_2011, n.trees = 5000, cv.fold =5)
```
```

# COMPARE

Now we find the MSE

(difference between predicted and mean values squared):

```
## [1] 0.1264041
```

MSE = 0.1264041, it preformed well

# COMPARE

Now we compare the summary statistics between actual vs. predicted values for the 2011 treatment group:

Actual Values

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   3.927   5.948   6.391   6.346   6.781   8.123
```

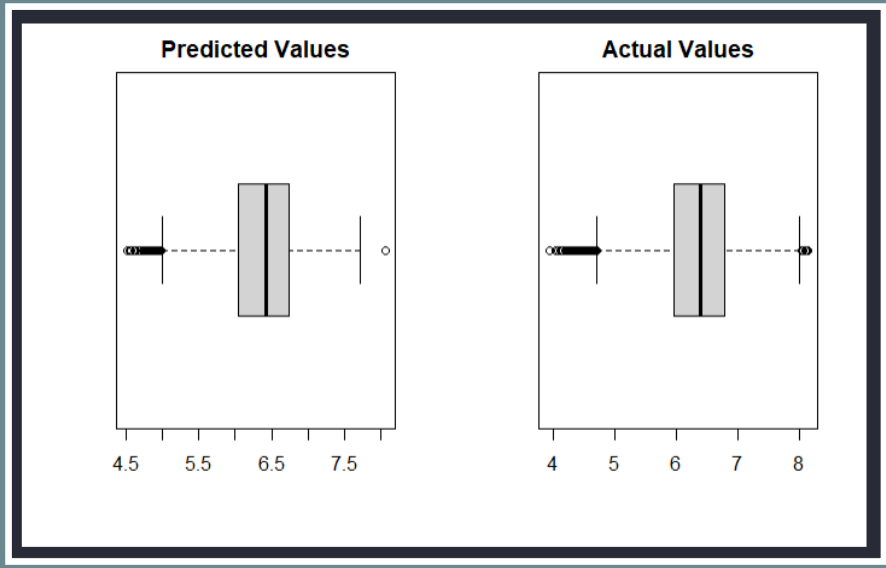
Predicted Values

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   4.407   6.041   6.426   6.373   6.750   7.948
```

We can see that the true values of `lusage` range from [4.516, 8.059], with a mean of 6.373.

The predicted values have a slightly larger range of [3.927, 8.123], and a relatively similar mean of 6.346.

# COMPARE




The boxplots show relatively the same distributions for both groups.

This shows that our model predicted the correct distribution as the actual values had.

# CONCLUSION

- With a MSE of 0.1264041, our model preformed well. While not perfect, our model was able to predict most values relatively closely.
- I can see why boosting is so common in the industry and I felt it was fairly simple to use. The time it took to run was the only downside of this method that I found.

# EXTRA CREDIT

- As we can see by the summary statistics and boxplots, our predicted values are slightly higher than what they actually were. If the households were told to save water in 2011, our model would not have accounted for it because the model was trained using variables `luse1-6' which we can assume did not have the same policy in place when their data was taken.

- Thus, the difference in our predicted and actual values may be due to that effect not being accounted for. If we built our model using treatment 2011 data to train it, we could have possibly accounted for the missing effects and gotten a closer estimate.

# EXTRA CREDIT

- If we only compared the treatment vs. control data from 2011, we would be looking at virtually no difference between the two values. Thus, we can infer that the difference is due to an externality and not poor modeling.

Summary statistics for treatment vs control from 2011:

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   3.927   5.948   6.391   6.346   6.781   8.123
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   3.917   5.957   6.402   6.359   6.810   8.318
```