Lab 9

Brendan Gubbins

11:59PM May 10, 2021

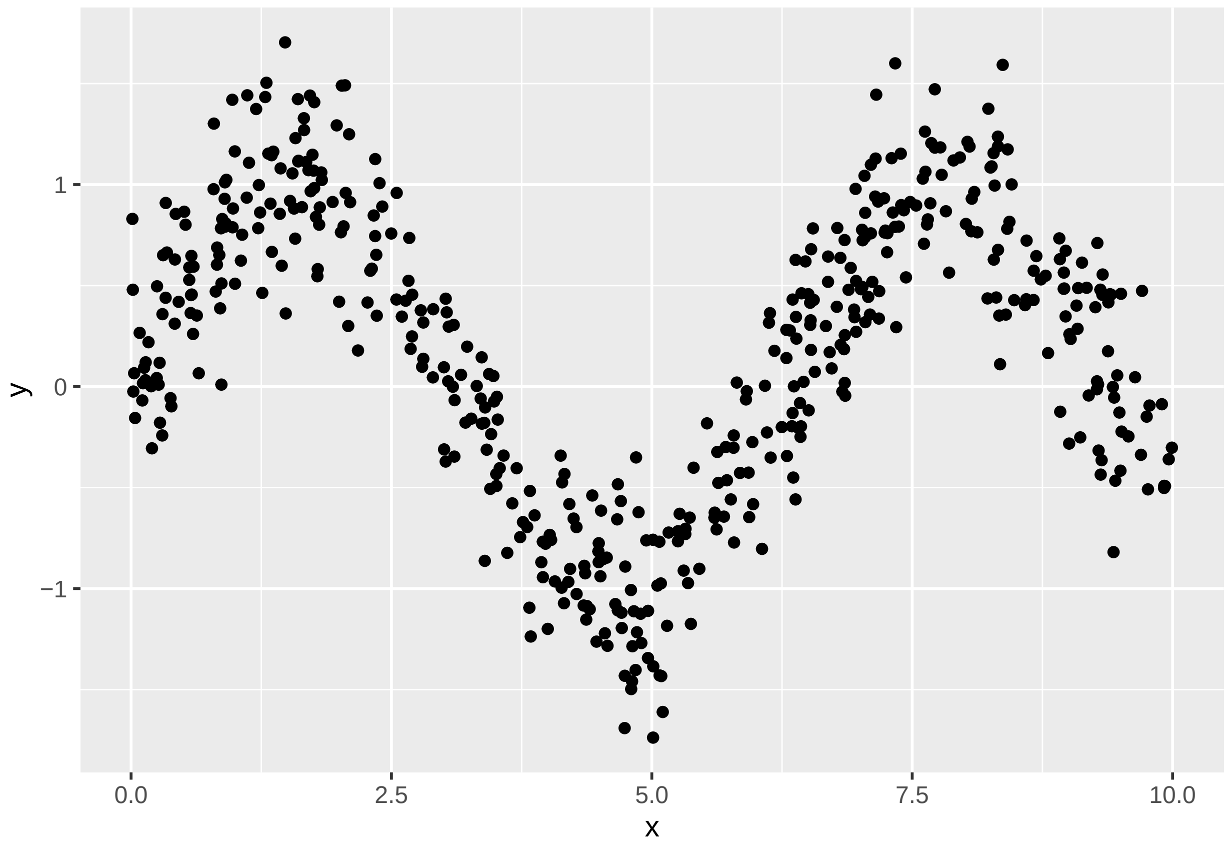
Here we will learn about trees, bagged trees and random forests. You can use the `YARF` package if it works, otherwise, use the `randomForest` package (the standard).

Let's take a look at the simulated sine curve data from practice lecture 12. Below is the code for the data generating process:

```r
rm(list = ls())
n = 500
sigma = 0.3
x_min = 0
x_max = 10
f_x = function(x){sin(x)}
y_x = function(x, sigma){f_x(x) + rnorm(n, 0, sigma)}
x_train = runif(n, x_min, x_max)
y_train = y_x(x_train, sigma)
```

Plot an example dataset of size 500:

```r
pacman::p_load(ggplot2)
ggplot(data.frame(x = x_train, y = y_train)) +
  geom_point(aes(x = x, y = y))
```

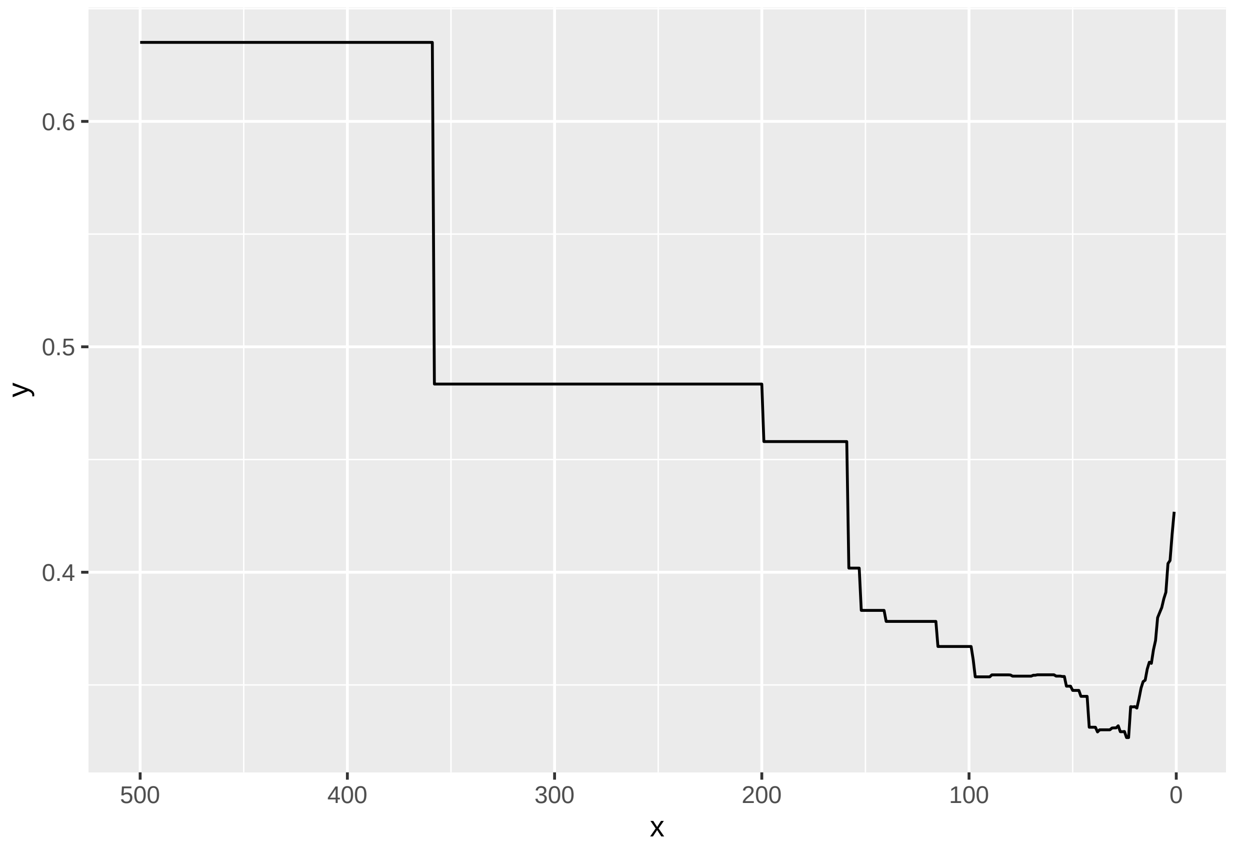


Create a test set of size 500 as well

```
x_test = runif(n, x_min, x_max)
y_test = y_x(x_test, sigma)
```

Locate the optimal node size hyperparameter for the regression tree model. I believe you can use `randomForest` here by setting `ntree = 1`, `replace = FALSE`, `sampsize = n` (`mtry` is already set to be 1 because there is only one feature) and then you can set `nodesize`. Plot `nodesize` by out of sample se.

```
pacman::p_load(randomForest)

node_sizes = 1:n
se_by_node_sizes = array(NA, length(node_sizes))

for (i in 1:length(node_sizes)) {
  rf_mod = randomForest(x = data.frame(x = x_train), y = y_train, ntree = 1, replace = FALSE, sampsize =
  y_hat_test = predict(rf_mod, data.frame(x = x_test))
  se_by_node_sizes[i] = sd(y_test - y_hat_test)
}

ggplot(data.frame(x = node_sizes, y = se_by_node_sizes)) +
  geom_line(aes(x = x, y = y)) +
  scale_x_reverse()
```

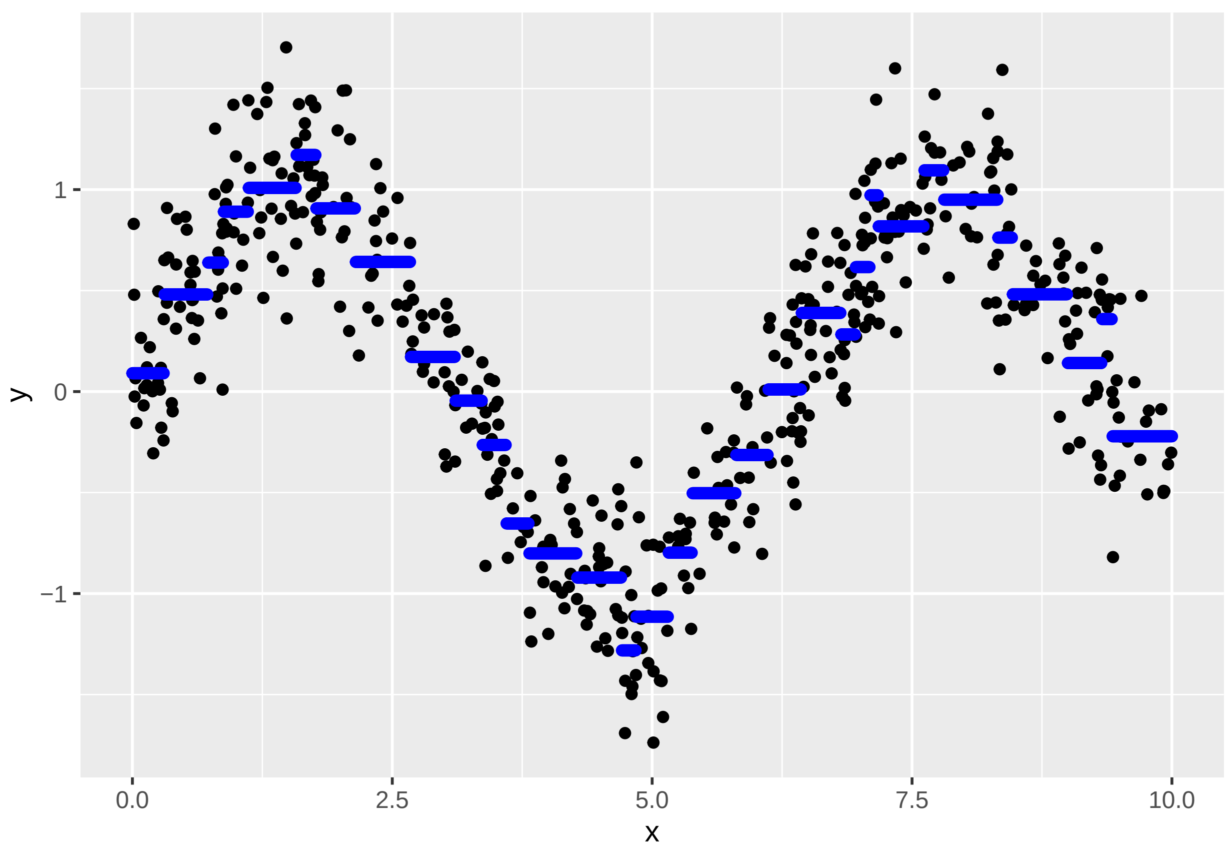


```
which.min(se_by_node_sizes)
```

```
## [1] 23
```

Plot the regression tree model with the optimal node size.

```
rf_mod = randomForest(x = data.frame(x = x_train), y = y_train, ntree = 1, replace = FALSE, sampsize = n

resolution = 0.01
x_grid = seq(from = x_min, to = x_max, by = resolution)
g_x = predict(rf_mod, data.frame(x = x_grid))

ggplot(data.frame(x = x_grid, y = g_x)) +
  aes(x = x, y = y) +
  geom_point(data = data.frame(x = x_train, y = y_train)) +
  geom_point(col = "blue")
```



Provide the bias-variance decomposition of this DGP fit with this model. It is a lot of code, but it is in the practice lectures. If your three numbers don't add up within two significant digits, increase your resolution.

```
#TO-DO ???
```

```
rm(list = ls())
```

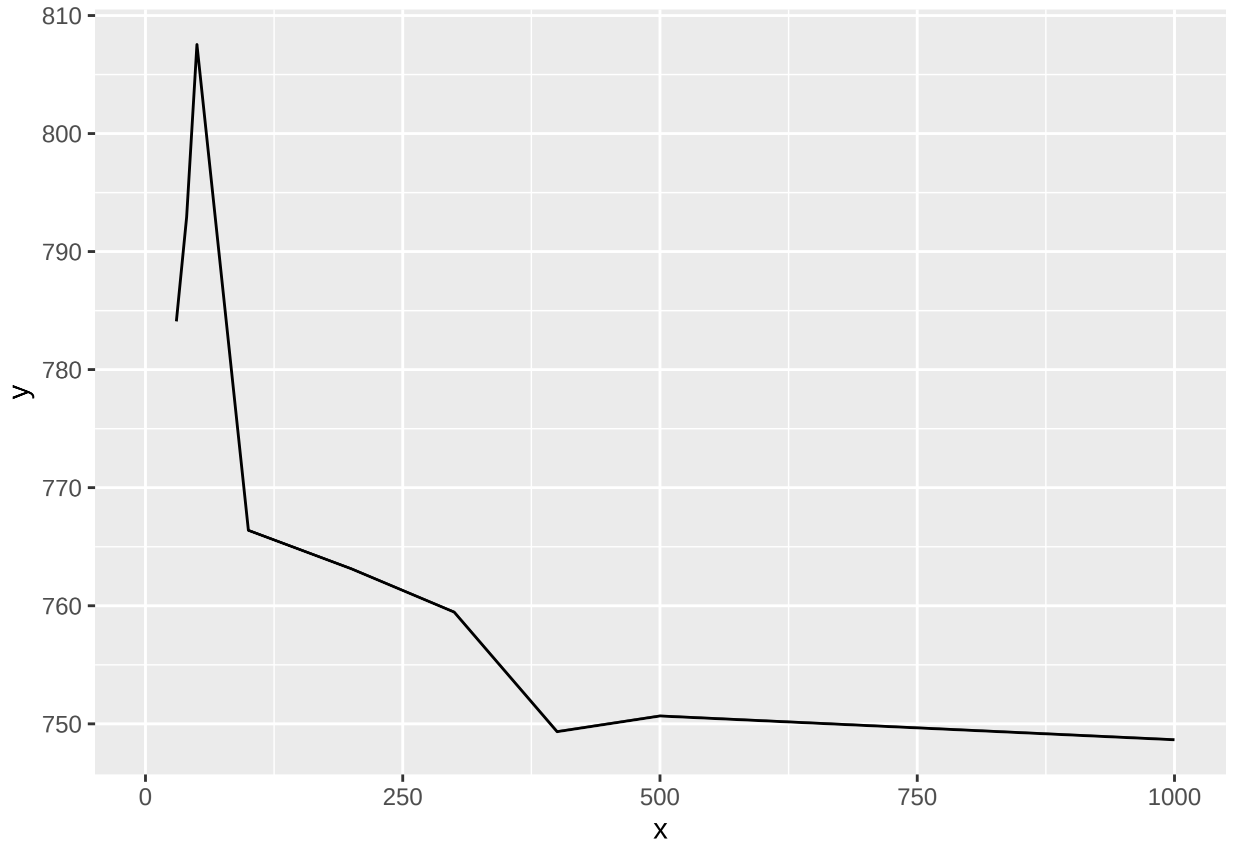
Take a sample of n = 2000 observations from the diamonds data.

```
pacman::p_load(dplyr)
diamond_samp = diamonds %>%
  sample_n(2000)
```

find the oob s_e for a RF model using 1, 2, 5, 10, 20, 30, 40, 50, 100, 200, 300, 400, 500, 1000 trees. If you are using the `randomForest` package, you can calculate oob residuals via `e_oob = y_train - rf_mod$predicted`. Plot it.

```
num_trees = c(1, 2, 5, 10, 20, 30, 40, 50, 100, 200, 300, 400, 500, 1000)
oob_se_by_num_trees = array(NA, length(num_trees))
for(i in 1:length(num_trees)) {
  rf_mod = randomForest(price ~ ., data = diamond_samp, ntree = num_trees[i])
  oob_se_by_num_trees[i] = sd(diamond_samp$price - rf_mod$predicted)
}

ggplot(data.frame(x = num_trees, y = oob_se_by_num_trees)) +
  geom_line(aes(x = x, y = y))
```

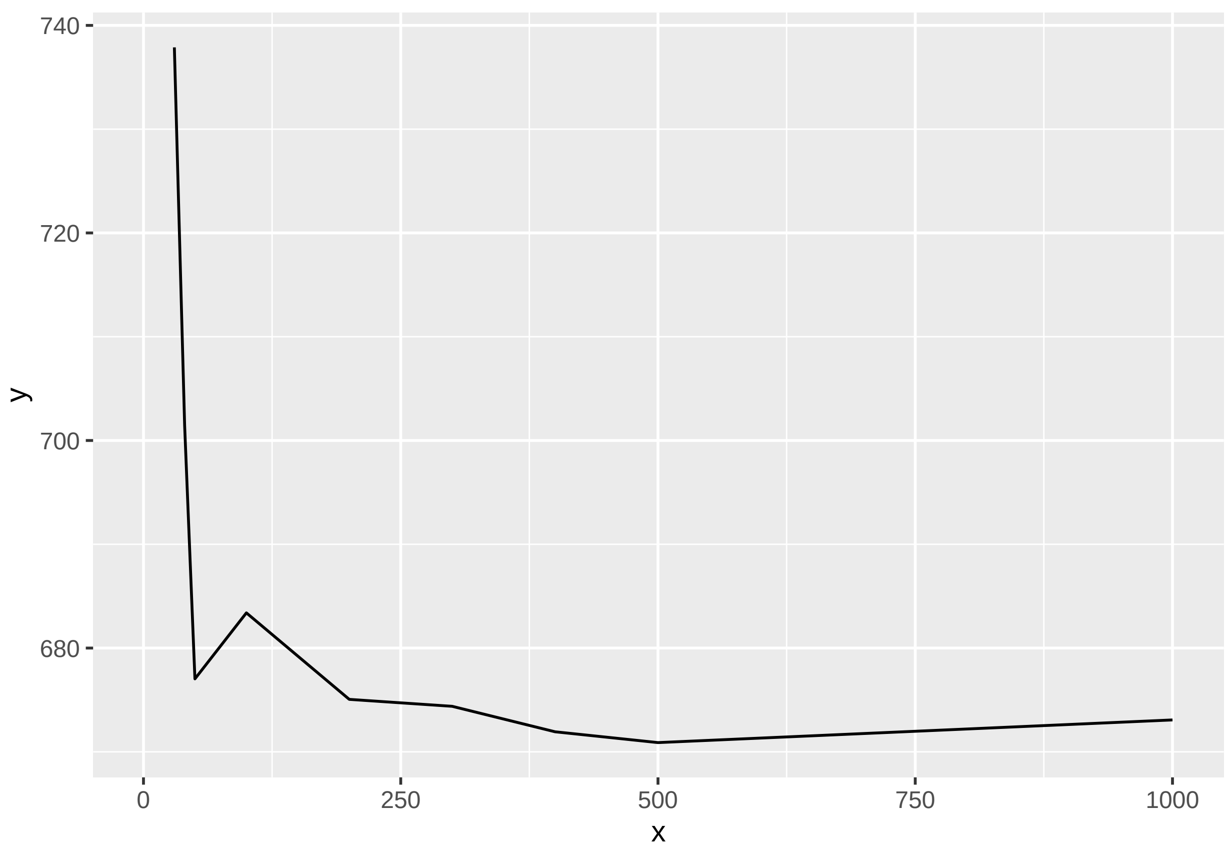
```
## Warning: Removed 5 row(s) containing missing values (geom_path).
```



Using the diamonds data, find the oob s_e for a bagged-tree model using 1, 2, 5, 10, 20, 30, 40, 50, 100, 200, 300, 400, 500, 1000 trees. If you are using the `randomForest` package, you can create the bagged tree model via setting an argument within the RF constructor function.

```
num_trees = c(1, 2, 5, 10, 20, 30, 40, 50, 100, 200, 300, 400, 500, 1000)
oob_se_by_num_trees_bag = array(NA, length(num_trees))
for(i in 1:length(num_trees)) {
  rf_mod = randomForest(price ~ ., data = diamond_samp, ntree = num_trees[i], mtry = ncol(diamond_samp)
  oob_se_by_num_trees_bag[i] = sd(diamond_samp$price - rf_mod$predicted)
}

ggplot(data.frame(x = num_trees, y = oob_se_by_num_trees_bag)) +
  geom_line(aes(x = x, y = y))
```

```
## Warning: Removed 5 row(s) containing missing values (geom_path).
```



What is the percentage gain / loss in performance of the RF model vs bagged trees model?

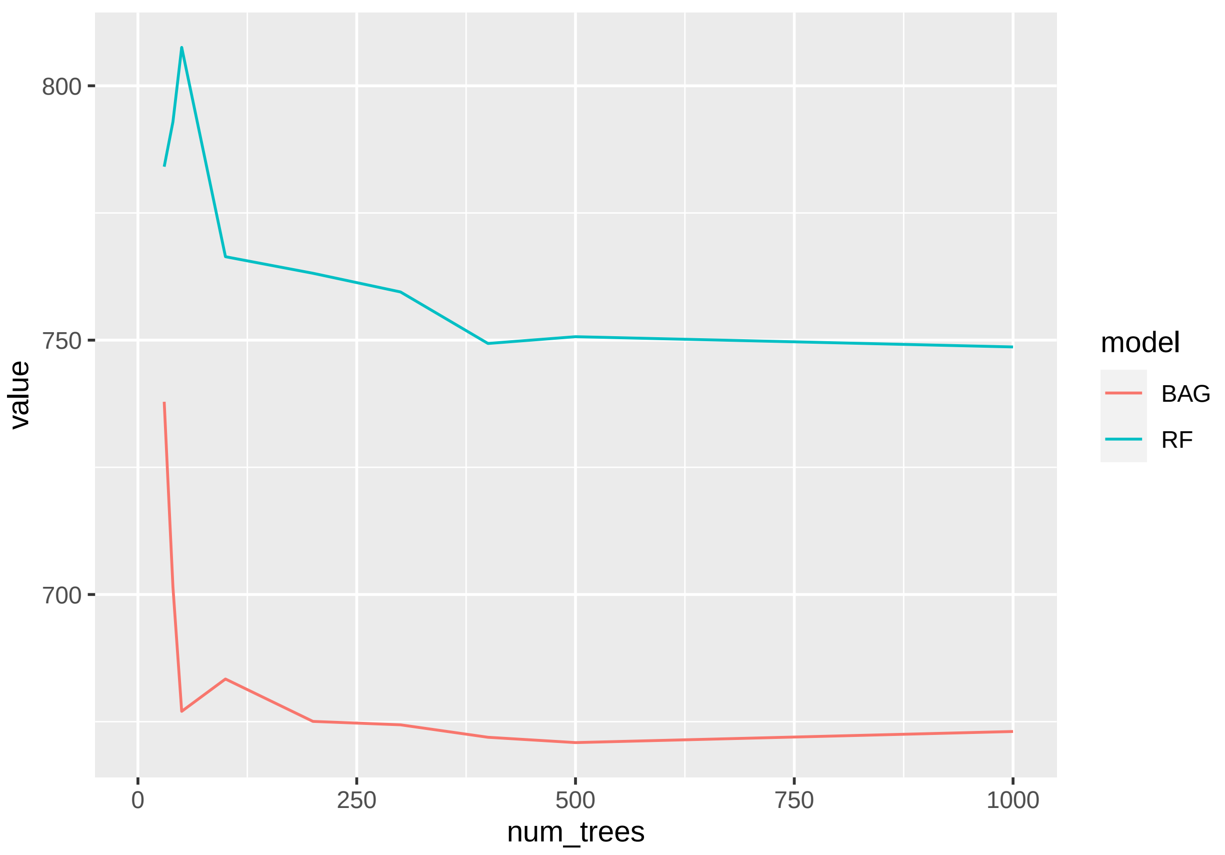
```
(oob_se_by_num_trees - oob_se_by_num_trees_bag) / oob_se_by_num_trees_bag * 100
```

```
##  [1]        NA        NA        NA        NA        NA  6.262792 13.041181
##  [8] 19.277608 12.146230 13.049018 12.616798 11.520882 11.892140 11.229885
```

Plot bootstrap s_e by number of trees for both RF and bagged trees.

```
ggplot(rbind(data.frame(num_trees = num_trees, value = oob_se_by_num_trees, model = "RF"),
      data.frame(num_trees = num_trees, value = oob_se_by_num_trees_bag, model = "BAG"))) +
  geom_line(aes(x = num_trees, y = value, color = model))
```

```
## Warning: Removed 10 row(s) containing missing values (geom_path).
```

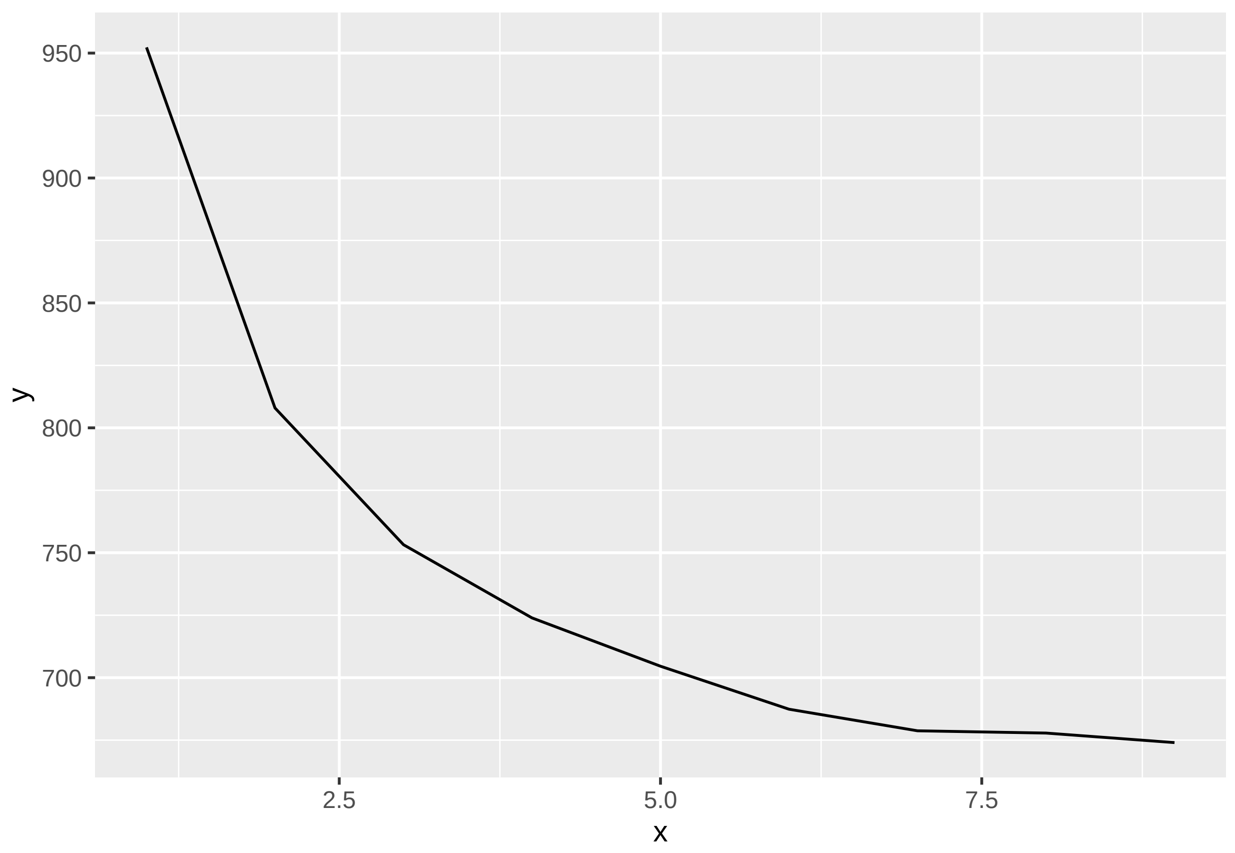


Build RF models for 500 trees using different `mtry` values: 1, 2, . . . the maximum. That maximum will be the number of features assuming that we do not binarize categorical features if you are using `randomForest` or the number of features assuming binarization of the categorical features if you are using `YARF`. Calculate bootstrap s_e for all mtry values.

```
mtrys = 1 : (ncol(diamond_samp) - 1)

oob_se_by_mtrys = array(NA, length(mtrys))
for(i in 1:length(mtrys)) {
  rf_mod = randomForest(price ~ ., data = diamond_samp, mtry = mtrys[i])
  oob_se_by_mtrys[i] = sd(diamond_samp$price - rf_mod$predicted)
}
```

Plot bootstrap s_e by mtry.

```
ggplot(data.frame(x = mtrys, y = oob_se_by_mtrys)) +
  geom_line(aes(x = x, y = y))
```



```r
rm(list = ls())
```

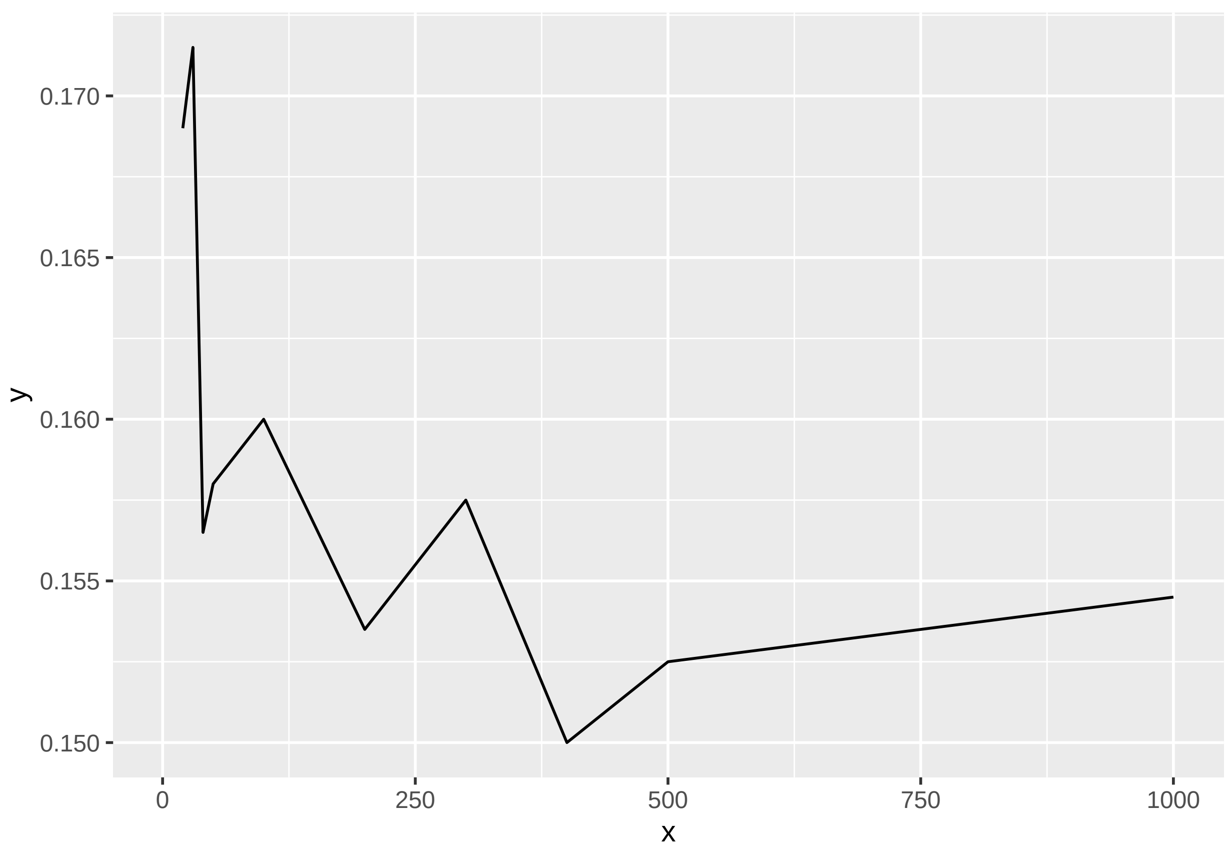
Take a sample of n = 2000 observations from the adult data.

```r
pacman::p_load_gh("coatless/ucidata")
data(adult)
adult = na.omit(adult)

adult_samp = adult %>%
  sample_n(2000)
```

Using the adult data, find the oob misclassification error for an RF model using 1, 2, 5, 10, 20, 30, 40, 50, 100, 200, 300, 400, 500, 1000 trees.

```r
num_trees = c(1, 2, 5, 10, 20, 30, 40, 50, 100, 200, 300, 400, 500, 1000)
oob_me_by_num_trees = array(NA, length(num_trees))
for(i in 1:length(num_trees)) {
  rf_mod = randomForest(income ~ ., data = adult_samp, ntree = num_trees[i])
  oob_me_by_num_trees[i] = mean(adult_samp$income != rf_mod$predicted)
}

ggplot(data.frame(x = num_trees, y = oob_me_by_num_trees)) +
  geom_line(aes(x = x, y = y))
```

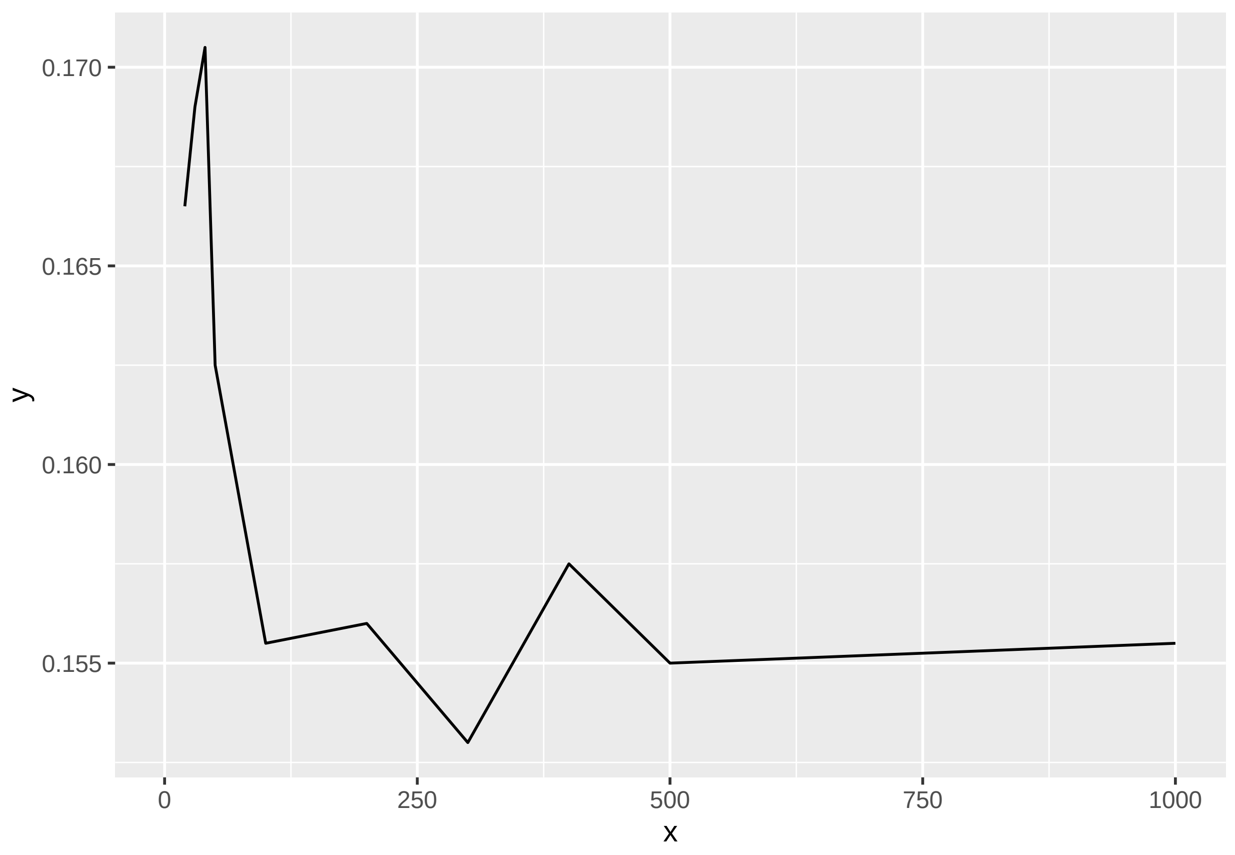
```
## Warning: Removed 4 row(s) containing missing values (geom_path).
```



Using the adult data, find the oob misclassification error for a bagged-tree model using 1, 2, 5, 10, 20, 30, 40, 50, 100, 200, 300, 400, 500, 1000 trees.

```
oob_me_by_num_trees_bag = array(NA, length(num_trees))
for(i in 1:length(num_trees)) {
  rf_mod = randomForest(income ~ ., data = adult_samp, ntree = num_trees[i], mtry = ncol(adult_samp) - 
  oob_me_by_num_trees_bag[i] = mean(adult_samp$income != rf_mod$predicted)
}

ggplot(data.frame(x = num_trees, y = oob_me_by_num_trees_bag)) +
  geom_line(aes(x = x, y = y))
```

```
## Warning: Removed 4 row(s) containing missing values (geom_path).
```



What is the percentage gain / loss in performance of the RF model vs bagged trees model?

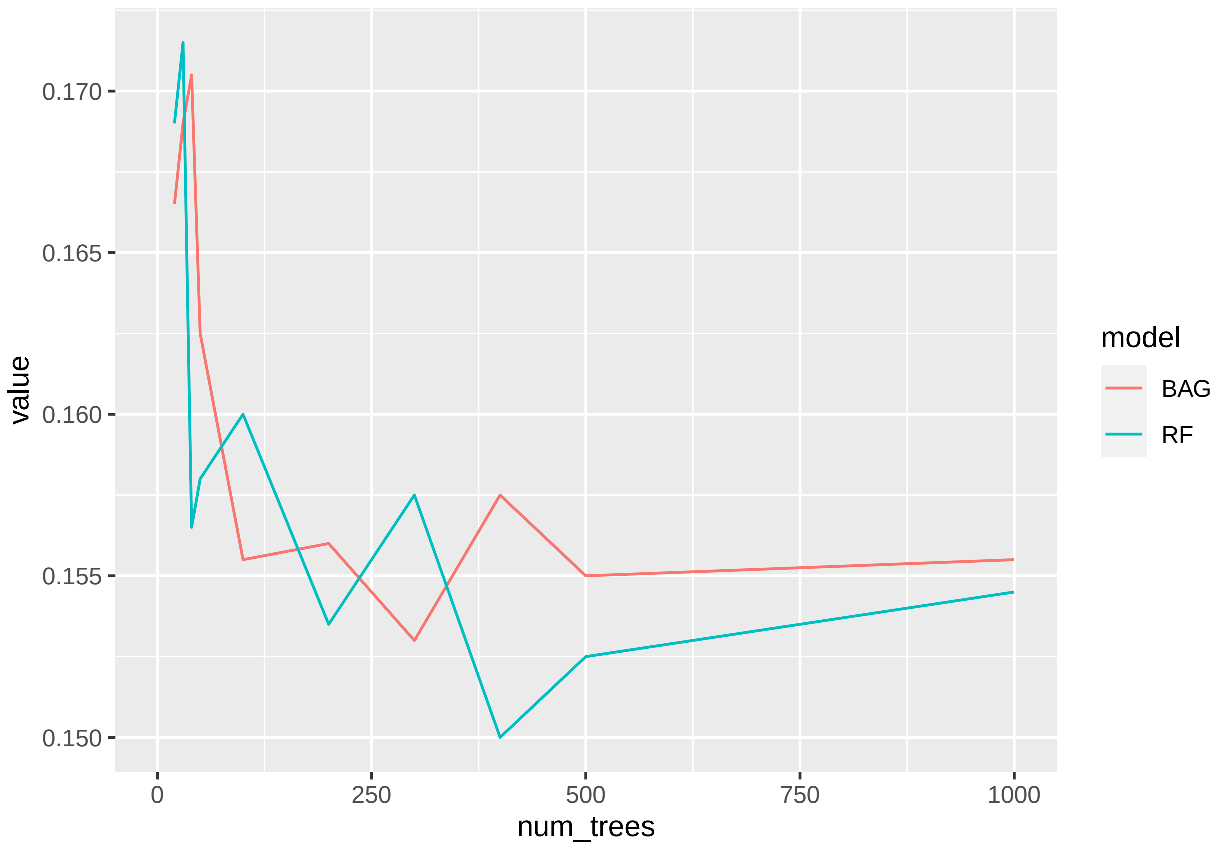
```
(oob_me_by_num_trees - oob_me_by_num_trees_bag) / oob_me_by_num_trees_bag * 100
```

```
##  [1]         NA         NA         NA         NA  1.5015015  1.4792899
##  [7] -8.2111437 -2.7692308  2.8938907 -1.6025641  2.9411765 -4.7619048
## [13] -1.6129032 -0.6430868
```

Plot bootstrap misclassification error by number of trees for both RF and bagged trees.

```
ggplot(rbind(data.frame(num_trees = num_trees, value = oob_me_by_num_trees, model = "RF"),
      data.frame(num_trees = num_trees, value = oob_me_by_num_trees_bag, model = "BAG"))) +
  geom_line(aes(x = num_trees, y = value, color = model))
```

```
## Warning: Removed 8 row(s) containing missing values (geom_path).
```

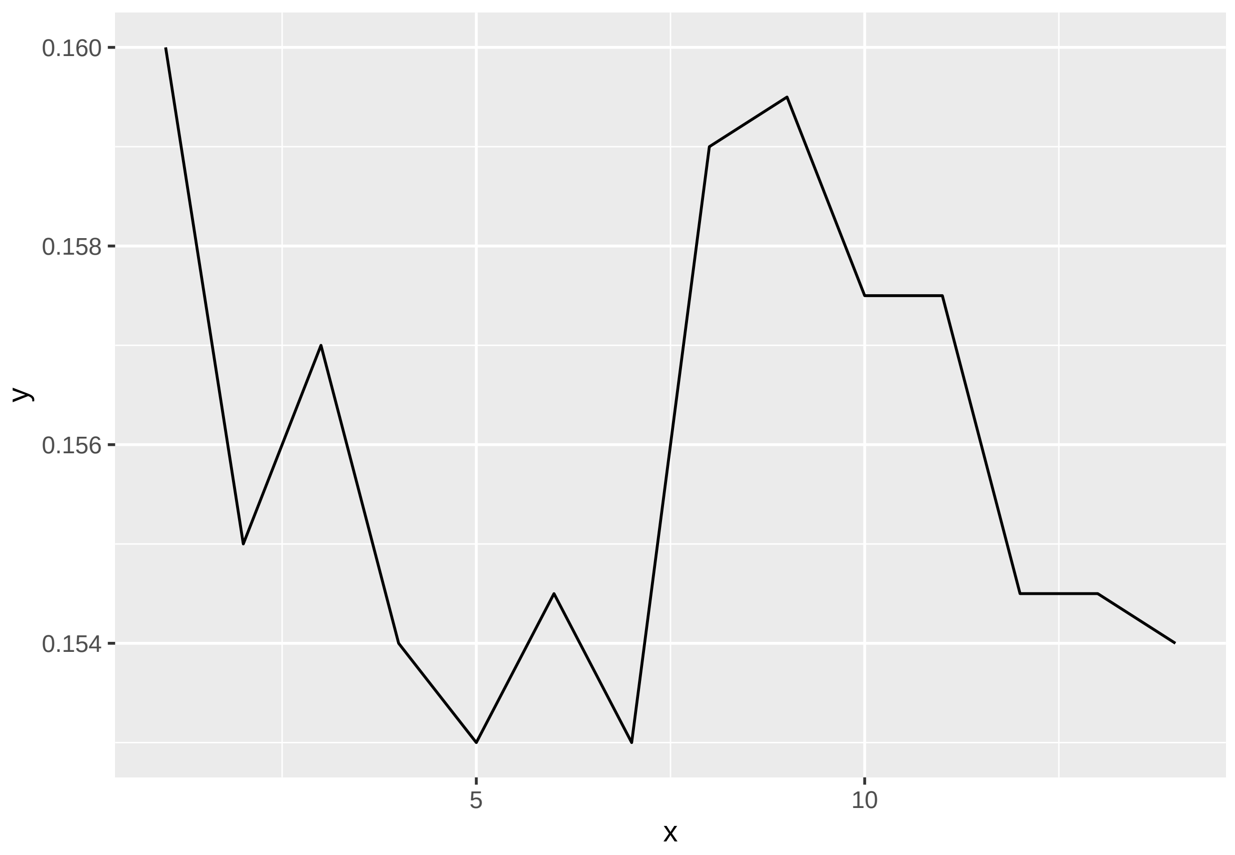


Build RF models for 500 trees using different `mtry` values: 1, 2, ... the maximum (see above as maximum is defined by the specific RF algorithm implementation).

```
mtrys = 1 : (ncol(adult_samp) - 1)

oob_me_by_mtrys = array(NA, length(mtrys))
for(i in 1:length(mtrys)) {
  rf_mod = randomForest(income ~ ., data = adult_samp, mtry = mtrys[i])
  oob_me_by_mtrys[i] = mean(adult_samp$income != rf_mod$predicted)
}
```

Plot bootstrap misclassification error by `mtry`.

```
ggplot(data.frame(x = mtrys, y = oob_me_by_mtrys)) +
  geom_line(aes(x = x, y = y))
```



```
rm(list = ls())
```

Write a function `random_bagged_ols` which takes as its arguments `X` and `y` with further arguments `num_ols_models` defaulted to 100 and `mtry` defaulted to NULL which then gets set within the function to be 50% of available features. This argument builds an OLS on a bootstrap sample of the data and uses only `mtry < p` of the available features. The function then returns all the `lm` models as a list with size `num_ols_models`.

```
random_bagged_ols = function(X, y, num_ols_models = 100, mtry = NULL) {
  bagged_list = list()
  col_names = colnames(X)

  for (i in 1 : num_ols_models) {
    mtry = sample(col_names, floor((ncol(X) - 1) / 2))
    mod = lm(y ~ ., data = X[,mtry])
    bagged_list[[i]] = mod
  }

  bagged_list
}
```

Load up the Boston Housing Data and separate into `X` and `y`.

```r
boston = MASS::Boston
X = boston
y = boston$medv
X$medv = NULL
```

Similar to lab 1, write a function that takes a matrix and punches holes (i.e. sets entries equal to `NA`) randomly with an argument `prob_missing`.

```r
holes = function(X, prob_missing) {
  nr = nrow(X)
  nc = ncol(X)
  M = matrix(rbinom(nr * nc, 1, prob_missing), nrow = nr, ncol = nc)
  X[M == 1] = NA
  X
}
```

Create a matrix `Xmiss` which is `X` but has missingness with probability of 10%.

```r
Xmiss = holes(X, .10)
```

Use a random forest modeling procedure to iteratively fill in the `NA`'s by predicting each feature of X using every other feature of X. You need to start by filling in the holes to use RF. So fill them in with the average of the feature.

```r
# manual way
Ximpnaive = Xmiss

Xj_bars = colMeans(Xmiss, na.rm = TRUE)
Xj_bars
```

```
##         crim           zn        indus         chas          nox           rm
##   3.79054261  10.56818182  11.14581699   0.06802721   0.55556457   6.27600654
##          age          dis          rad          tax      ptratio        black
##  68.46413043   3.77521081   9.54901961 407.72466960  18.51651982 358.12258621
##        lstat
##  12.74503282
```

```r
for(j in 1:ncol(Xmiss)) {
  for (i in 1:nrow(Xmiss)) {
    if (is.na(Ximpnaive[i,j])) {
      Ximpnaive[i,j] = Xj_bars[j]
    }
  }
}

# better way
pacman::p_load(missForest)
Ximp = missForest(data.frame(Xmiss))$ximp
```

```
##   missForest iteration 1 in progress...
## Warning in randomForest.default(x = obsX, y = obsY, ntree = ntree, mtry =
## mtry, : The response has five or fewer unique values. Are you sure you want to
## do regression?

## done!
##   missForest iteration 2 in progress...

## Warning in randomForest.default(x = obsX, y = obsY, ntree = ntree, mtry =
## mtry, : The response has five or fewer unique values. Are you sure you want to
## do regression?

## done!
##   missForest iteration 3 in progress...

## Warning in randomForest.default(x = obsX, y = obsY, ntree = ntree, mtry =
## mtry, : The response has five or fewer unique values. Are you sure you want to
## do regression?

## done!
##   missForest iteration 4 in progress...

## Warning in randomForest.default(x = obsX, y = obsY, ntree = ntree, mtry =
## mtry, : The response has five or fewer unique values. Are you sure you want to
## do regression?

## done!
```