# Lab 9

Elizabeth McHugh

11:59PM May 10, 2021

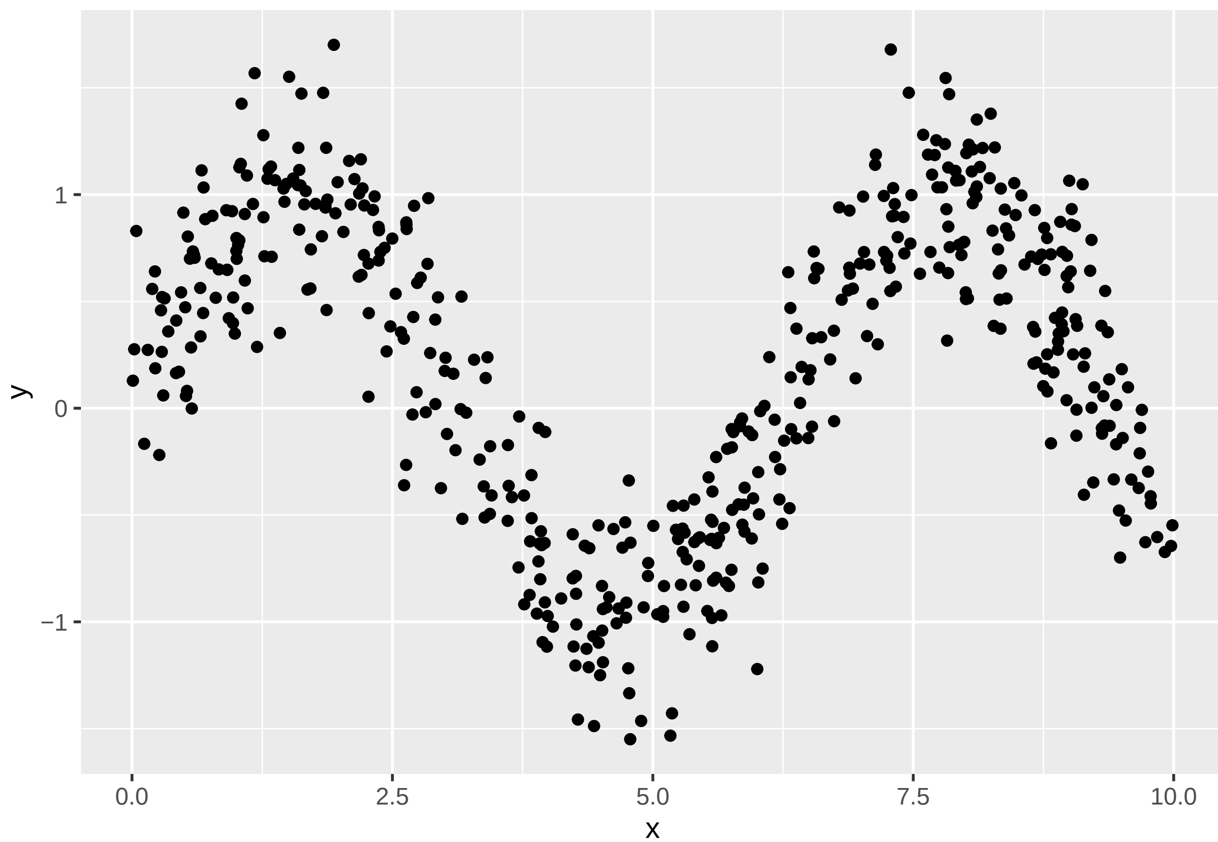
Here we will learn about trees, bagged trees and random forests. You can use the `YARF` package if it works, otherwise, use the `randomForest` package (the standard).

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

```
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:

```
pacman::p_load(ggplot2)

ggplot(data.frame(x = x_train, y = y_train)) +
  geom_point(aes(x = x, y = y))
```

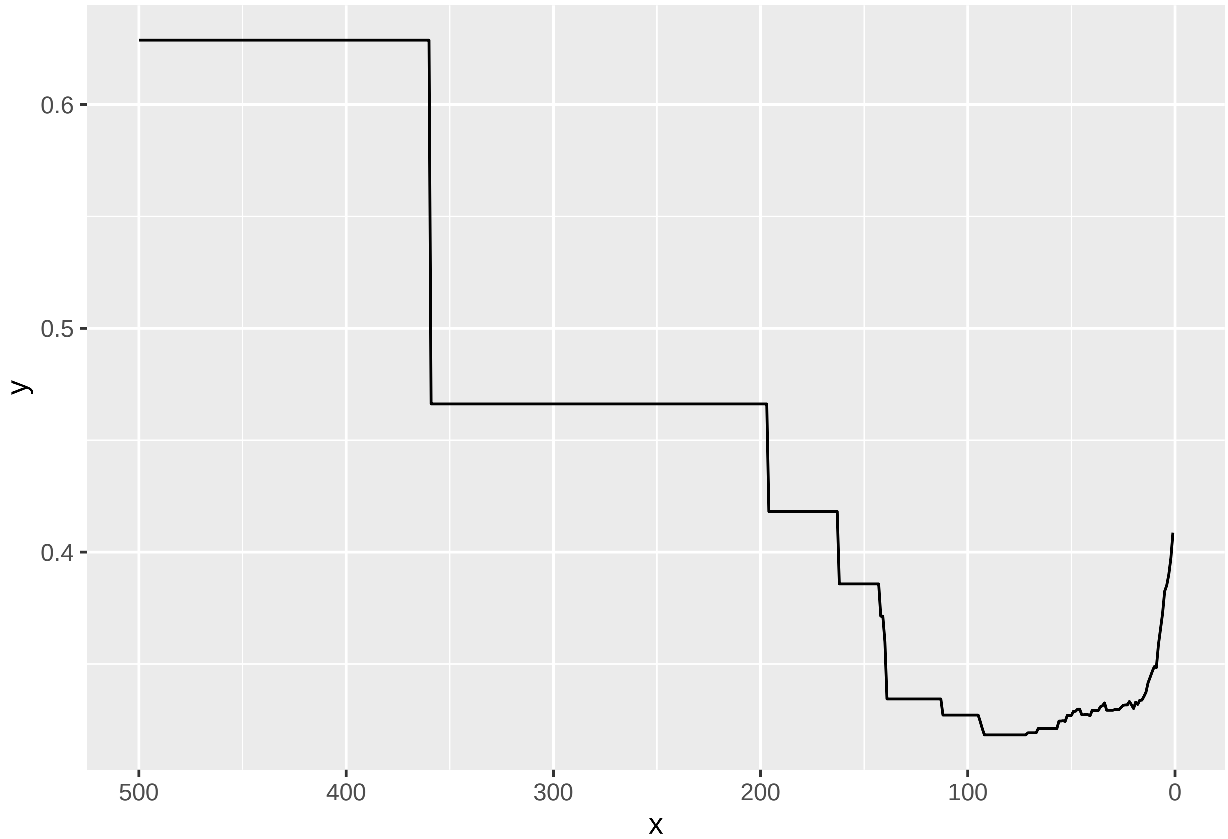



Create a test set of size 500 as well

```
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`.

```
pacman::p_load(randomForest)

nodesizes = 1:n
se_by_nodesizes = array(NA, length(nodesizes))

for (i in 1:length(nodesizes)){
  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_nodesizes[i] = sd(y_test - y_hat_test)
}

ggplot(data.frame(x = nodesizes, y = se_by_nodesizes)) +
  geom_line(aes(x = x, y = y)) +
  scale_x_reverse()
```

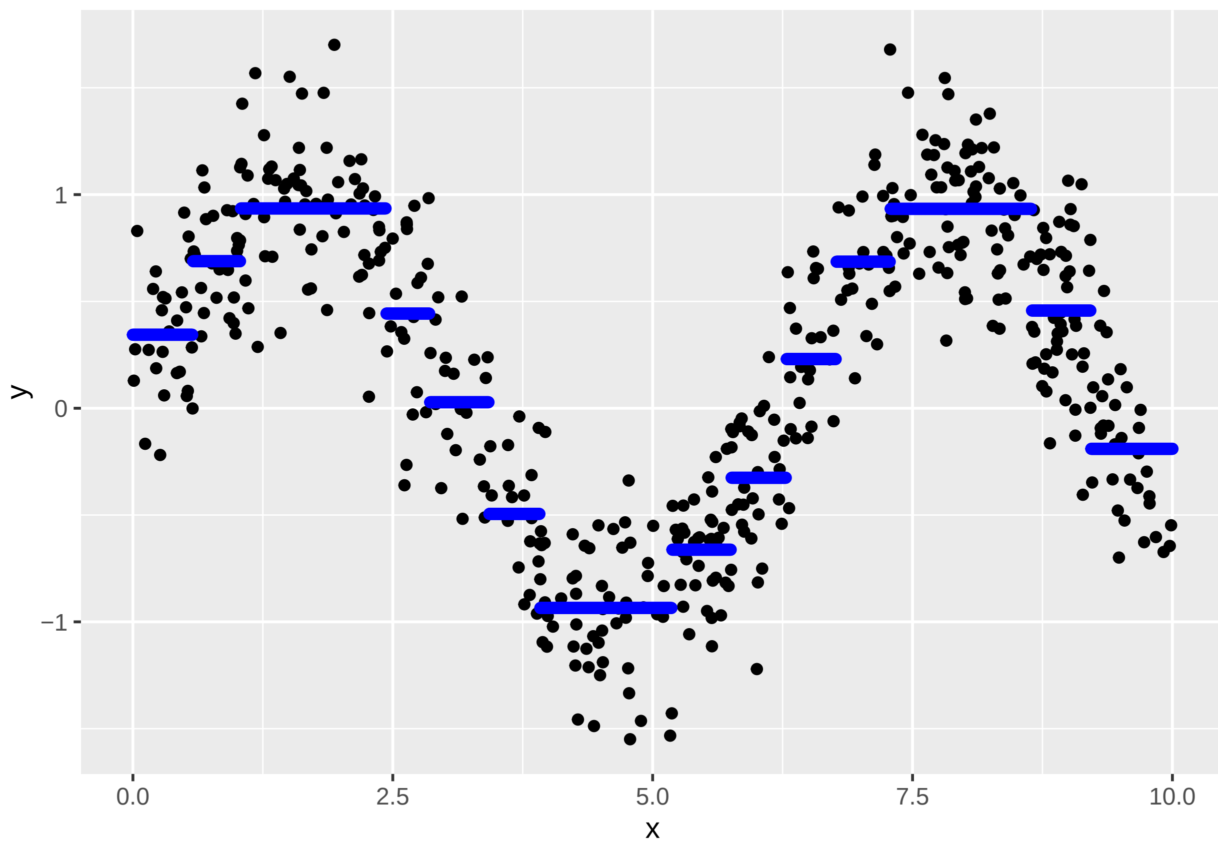



```
which.min(se_by_nodesizes)
```

```
## [1] 72
```

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 = r

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(color = "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.

```
###????
```

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

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

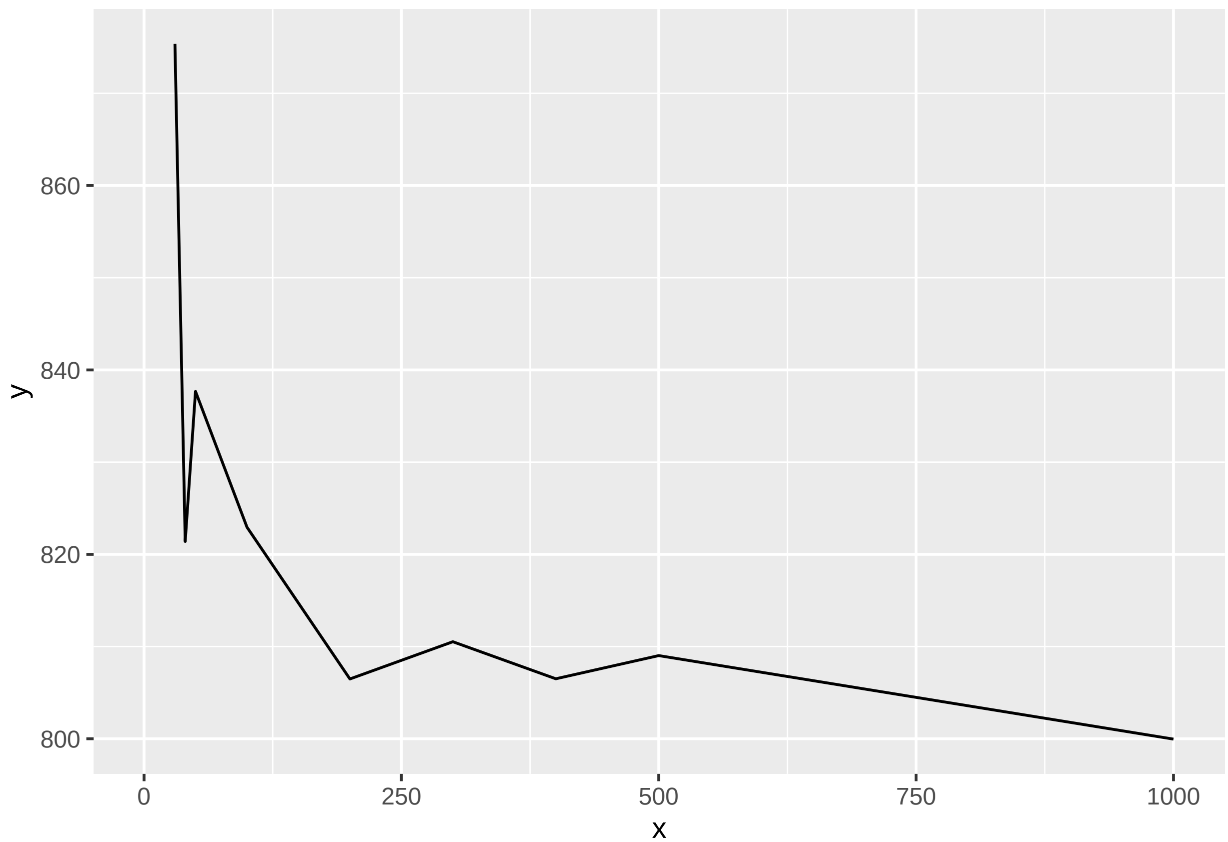
```
pacman::p_load(dplyr)

diamonds_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.

```
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 = diamonds_samp, ntree = num_trees[i])
  oob_se_by_num_trees[i] = sd(diamonds_samp$price - rf_mod$predicted)
```

```
}

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

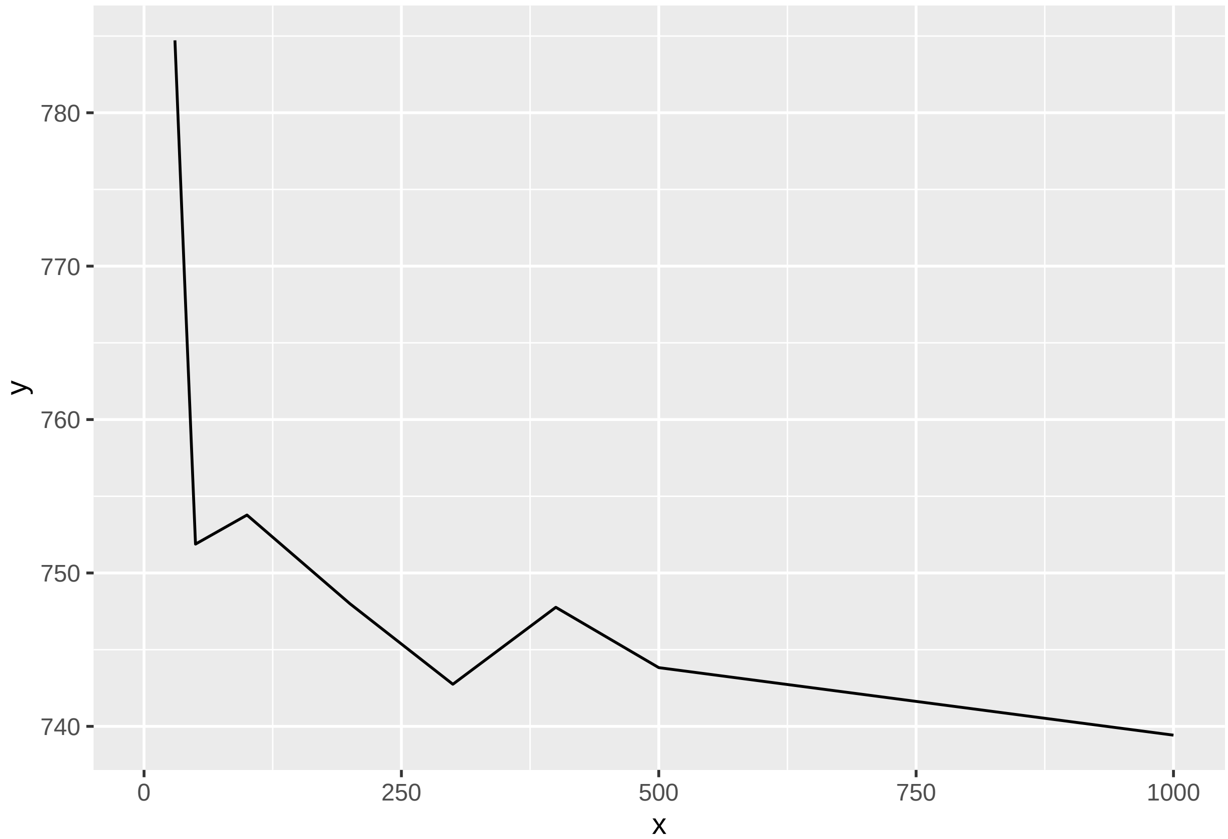
```
## Warning: Removed 5 row(s) containing missing values (geom_path).
```




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 = diamonds_samp, ntree = num_trees[i], mtry = (ncol(diamonds_sa
  oob_se_by_num_trees_bag[i] = sd(diamonds_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?

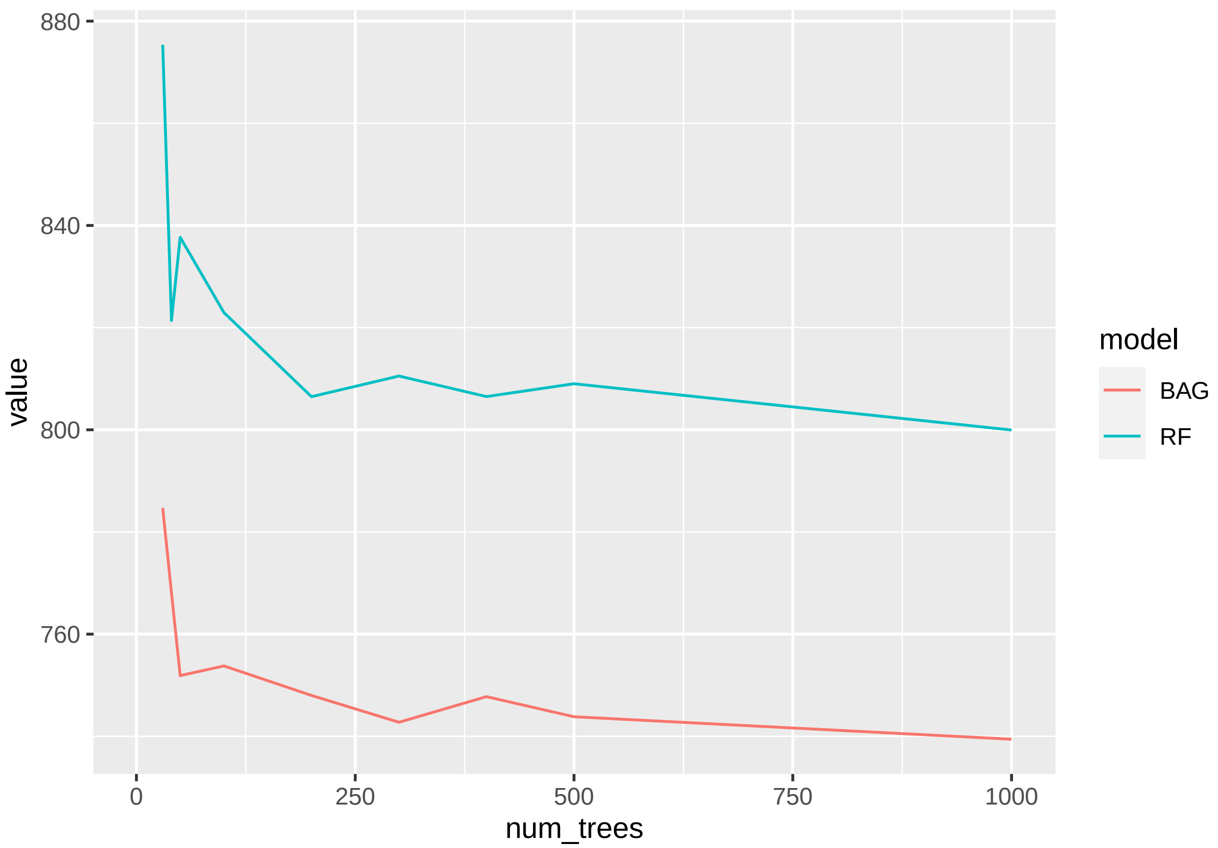
```
(oob_se_by_num_trees - oob_se_by_num_trees_bag) / oob_se_by_num_trees_bag * 100
```

```
##  [1]        NA        NA        NA        NA        NA 11.550811  6.937400
##  [8] 11.410255  9.175883  7.818791  9.124862  7.855409  8.763816  8.186307
```

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(nu
  geom_line(aes(x = num_trees, y = value, color = model))
```

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

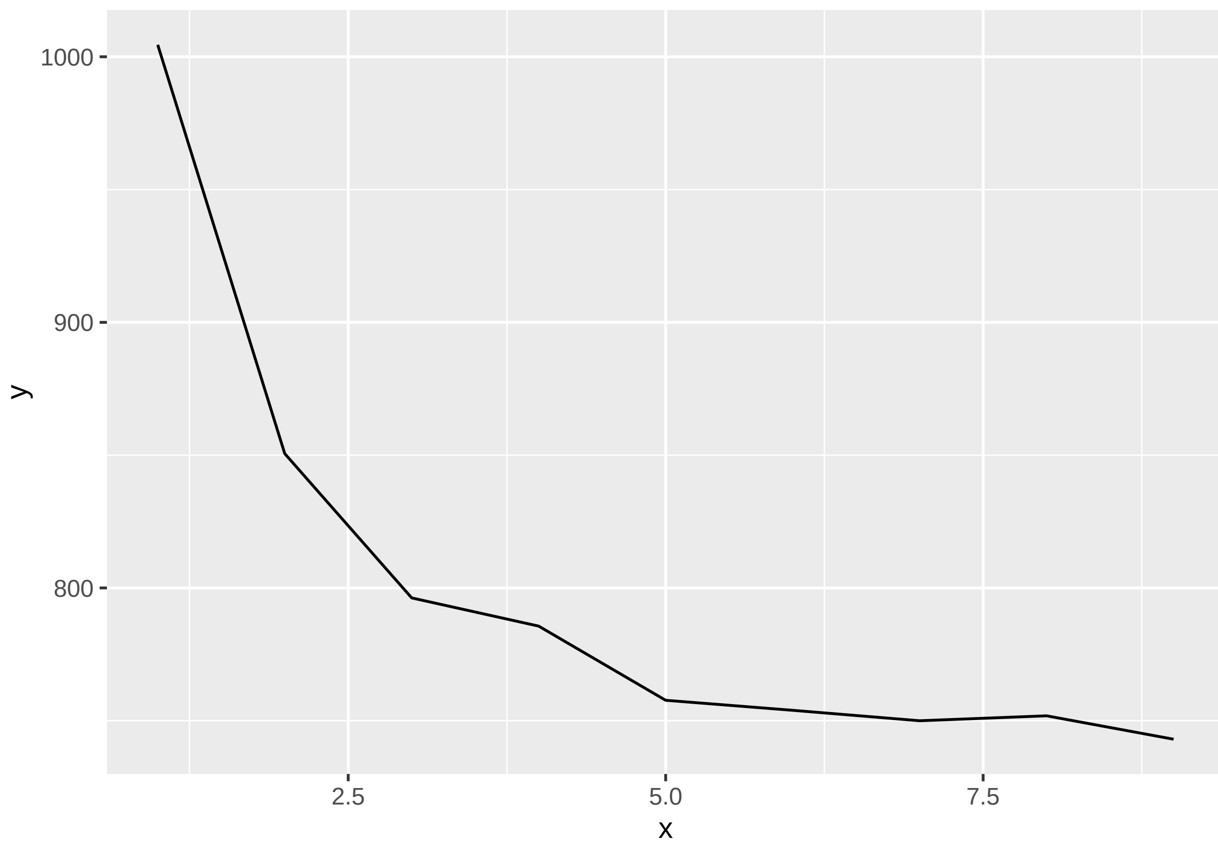



Build RF models for 500 trees using different `mtry` values: 1, 2, … the maximum. That maximum will be the number of features assuming that we do not binarize categorical features if you are using `randomForest` or the number of features assuming binarization of the categorical features if you are using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

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

oob_se_by_mtrys = array(NA, length(mtrys))

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

Plot oob s_e by mtry.

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




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

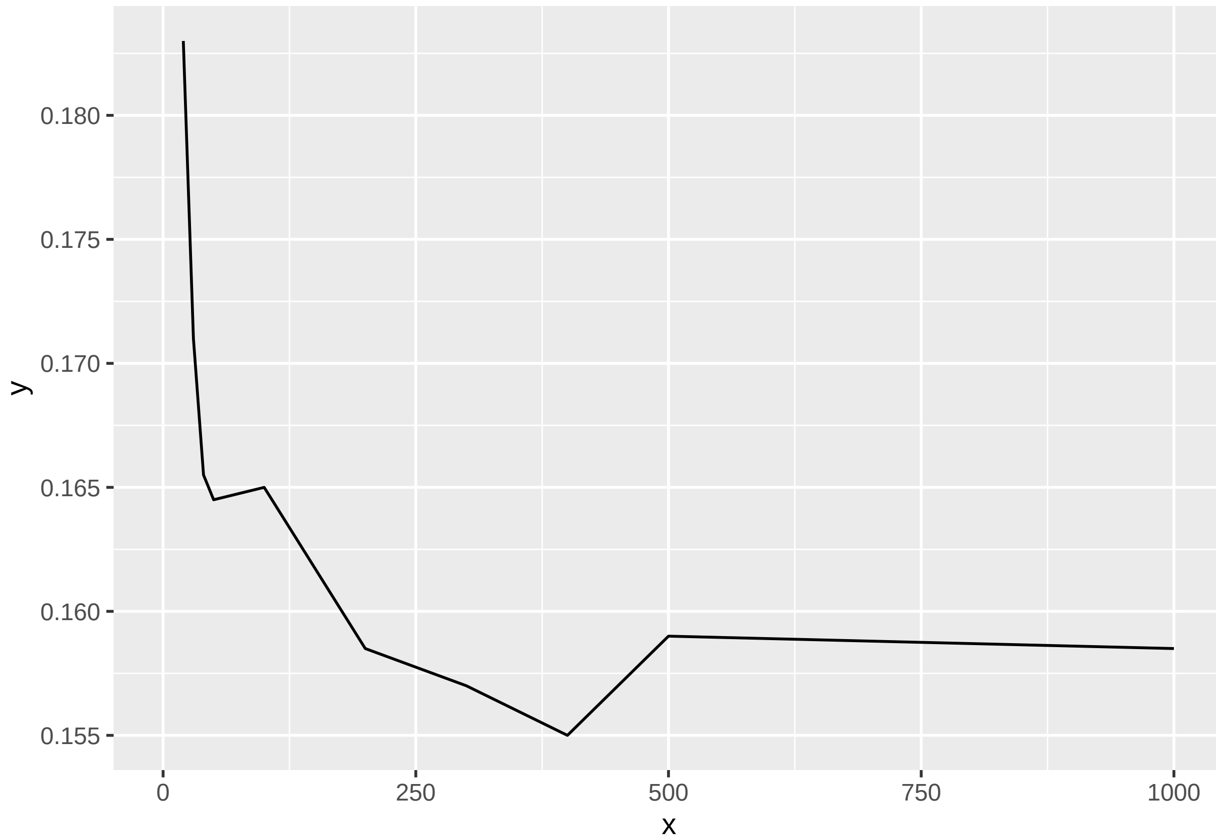
Take a sample of n = 2000 observations from the adult data.

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

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

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

```
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))
```

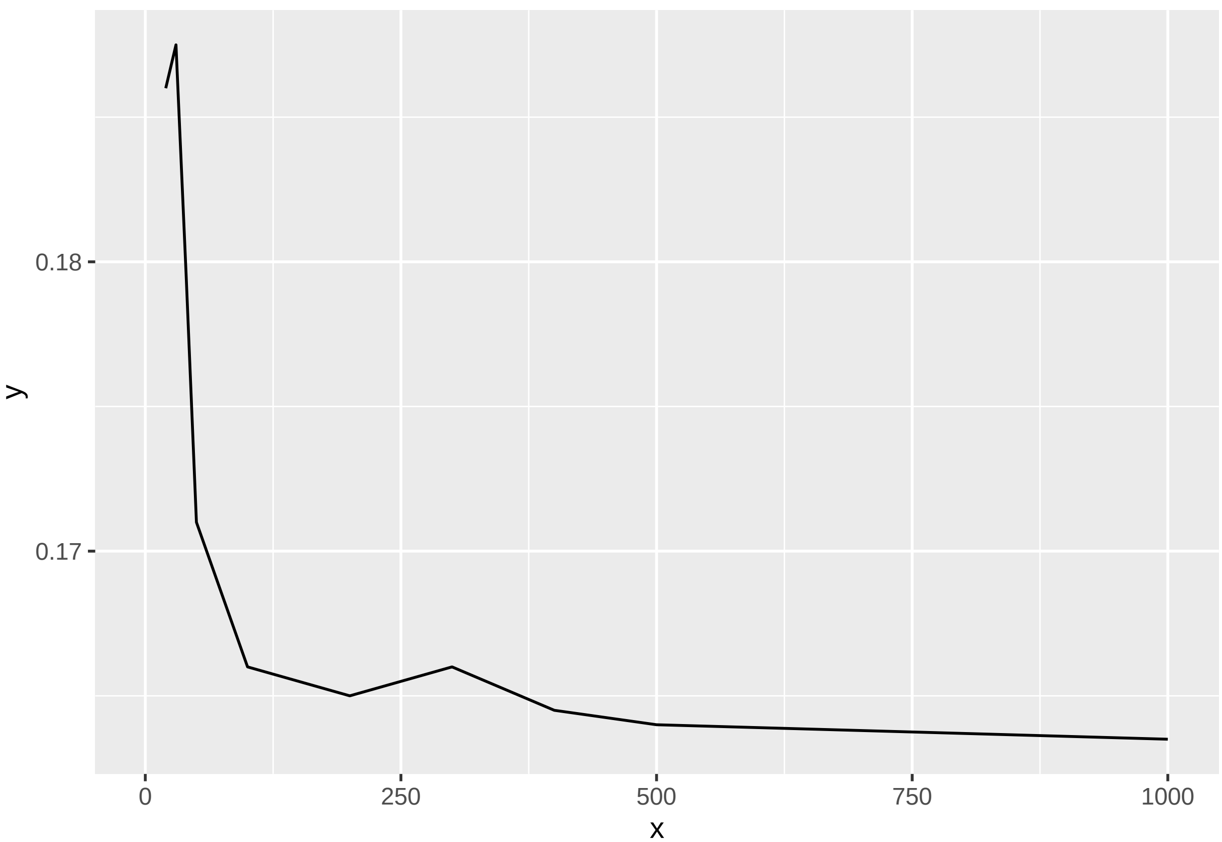
```
## Warning: Removed 4 row(s) containing missing values (geom_path).
```




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

```
num_trees = c(1, 2, 5, 10, 20, 30, 40, 50, 100, 200, 300, 400, 500, 1000)

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?

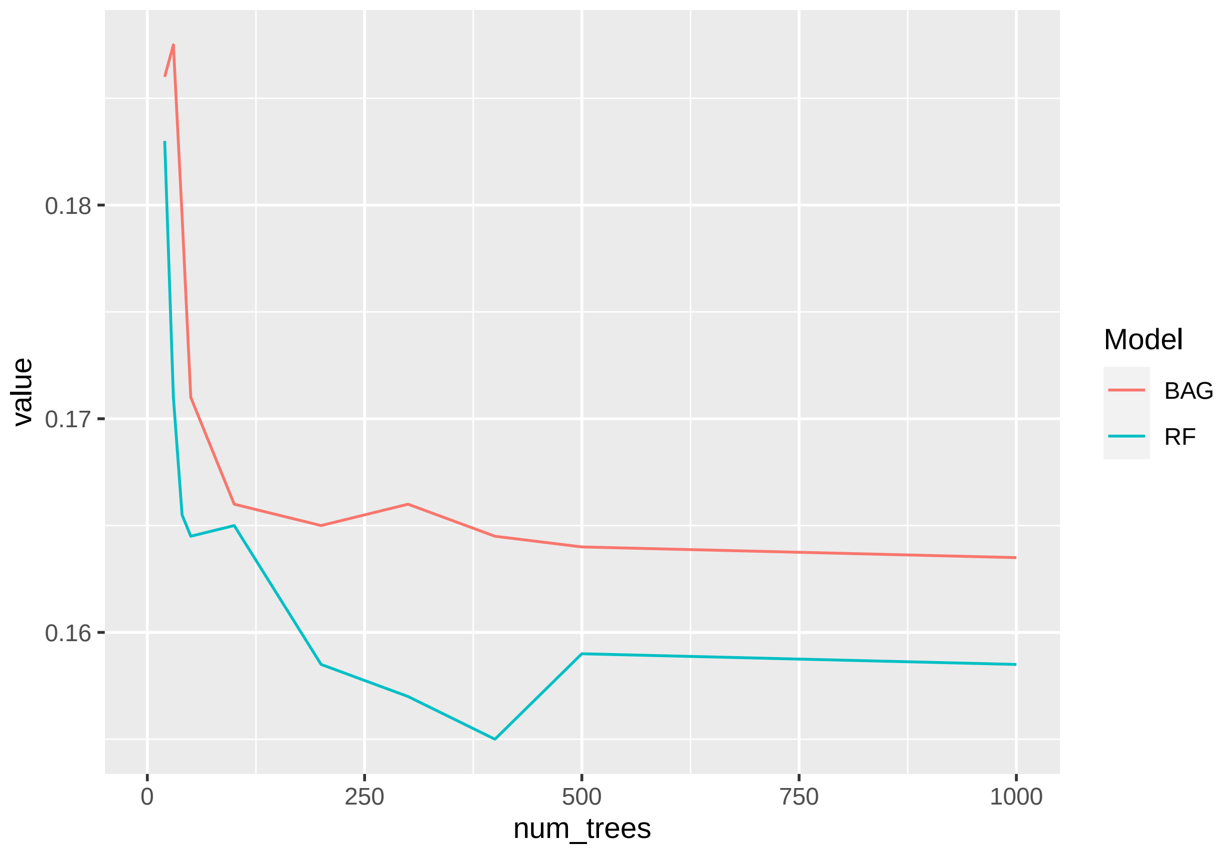
```
((oob_me_by_num_trees - oob_me_by_num_trees_bag) / oob_me_by_num_trees_bag) * 100
```

```
##  [1]         NA         NA         NA         NA -1.6129032 -8.8000000
##  [7] -7.7994429 -3.8011696 -0.6024096 -3.9393939 -5.4216867 -5.7750760
## [13] -3.0487805 -3.0581040
```

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(nu
  geom_line(aes(x = num_trees, y = value, color = Model))
```

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

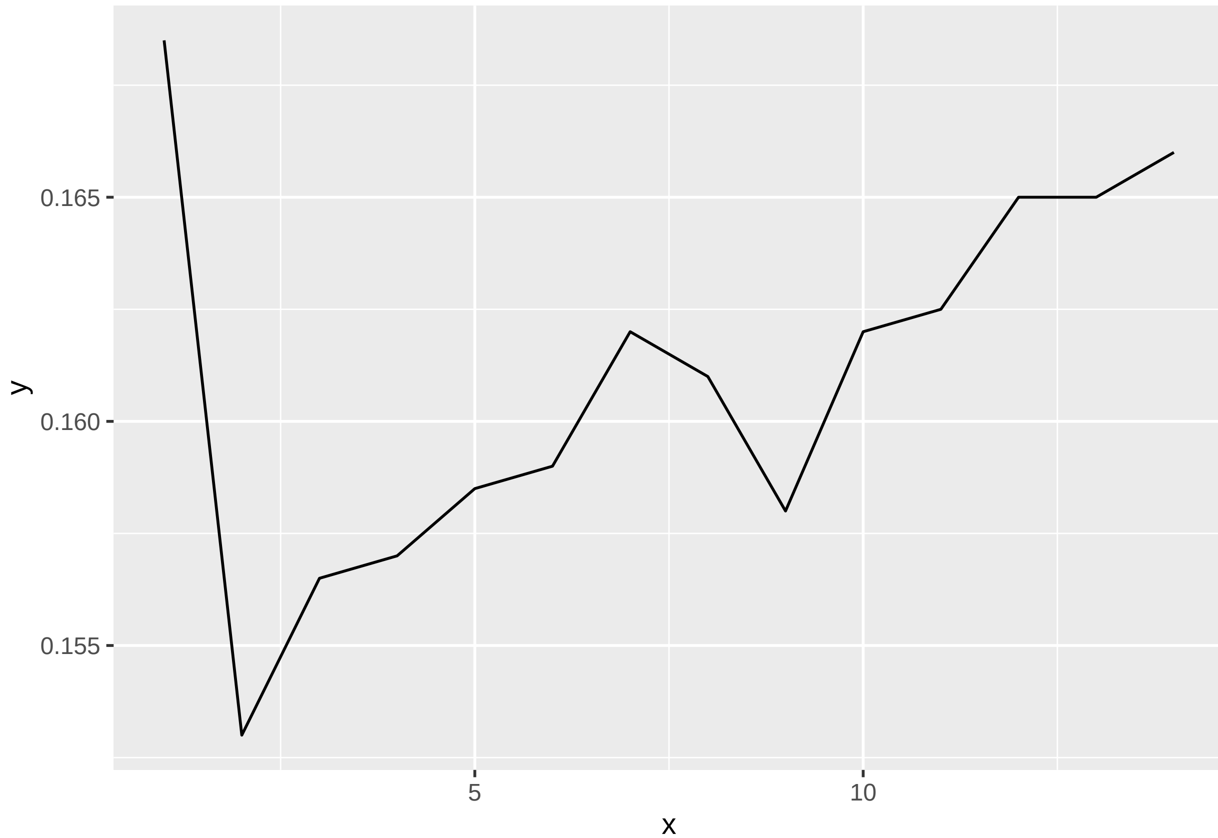



Build RF models for 500 trees using different `mtry` values: 1, 2, ... the maximum (see above as maximum is defined by the specific RF algorithm implementation).

```
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, ntree = 500, 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))
```




```r
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`.

```r
##SEE LECTURE 24??  This is not what I'm trying to do. What in the world am I trying to do again? I need

list_lm_models = list(NA, 100)

random_bagged_ols = function (X, y, num_ols_models = 100, mtry = NULL) {
  mtry = ncol(X) * .5

  list_lm_models = list(NA, num_ols_models)

  for (i in 1:num_ols_models){
    lm_mod = lm(y ~ . , data = X)
    list_lm_models[i] = lm_mod
  }
}
```

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

```r
pacman::p_load(MASS)
data(Boston)

X = Boston[1:13]
y = Boston$medv
```

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

Ximpute_RF = Xmiss

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

head(Ximpute_RF, 40)
```

```
##        crim       zn    indus       chas       nox       rm       age      dis
## 1  3.592145 18.00000  2.31000 0.00000000 0.5380000 6.575000  65.20000 4.090000
## 2  0.027310  0.00000  7.07000 0.00000000 0.4690000 6.421000  78.90000 3.788106
## 3  0.027290  0.00000  7.07000 0.00000000 0.4690000 7.185000  61.10000 4.967100
## 4  0.032370  0.00000  2.18000 0.00000000 0.4580000 6.998000  45.80000 3.788106
## 5  0.069050 10.99143  2.18000 0.00000000 0.4580000 7.147000  54.20000 6.062200
## 6  3.592145  0.00000  2.18000 0.00000000 0.4580000 6.430000  58.70000 6.062200
## 7  0.088290 10.99143  7.87000 0.00000000 0.5540845 6.012000  66.60000 5.560500
## 8  0.144550 12.50000  7.87000 0.00000000 0.5240000 6.172000  96.10000 5.950500
## 9  0.211240 12.50000  7.87000 0.00000000 0.5240000 5.631000 100.00000 6.082100
## 10 0.170040 12.50000  7.87000 0.00000000 0.5240000 6.004000  68.25867 6.592100
## 11 0.224890 12.50000  7.87000 0.00000000 0.5240000 6.377000  94.30000 6.346700
```

```
## 12 0.117470 12.50000  7.87000 0.00000000 0.5240000 6.009000  82.90000 6.226700
## 13 0.093780 12.50000  7.87000 0.00000000 0.5540845 5.889000  39.00000 5.450900
## 14 0.629760  0.00000  8.14000 0.00000000 0.5380000 5.949000  61.80000 4.707500
## 15 0.637960  0.00000  8.14000 0.00000000 0.5380000 6.096000  84.50000 4.461900
## 16 0.627390  0.00000  8.14000 0.00000000 0.5380000 5.834000  56.50000 4.498600
## 17 1.053930  0.00000  8.14000 0.00000000 0.5380000 5.935000  29.30000 4.498600
## 18 3.592145  0.00000  8.14000 0.00000000 0.5380000 6.283719  81.70000 3.788106
## 19 0.802710  0.00000  8.14000 0.00000000 0.5380000 5.456000  36.60000 3.796500
## 20 0.725800  0.00000  8.14000 0.00000000 0.5540845 5.727000  69.50000 3.796500
## 21 1.251790  0.00000  8.14000 0.00000000 0.5380000 5.570000  98.10000 3.797900
## 22 0.852040  0.00000  8.14000 0.00000000 0.5540845 5.965000  89.20000 4.012300
## 23 1.232470  0.00000  8.14000 0.00000000 0.5380000 6.142000  91.70000 3.976900
## 24 0.988430  0.00000  8.14000 0.00000000 0.5380000 5.813000 100.00000 4.095200
## 25 0.750260  0.00000 10.96392 0.00000000 0.5380000 5.924000  94.10000 3.788106
## 26 0.840540  0.00000  8.14000 0.07592191 0.5380000 5.599000  85.70000 4.454600
## 27 0.671910  0.00000  8.14000 0.00000000 0.5380000 5.813000  90.30000 4.682000
## 28 0.955770  0.00000  8.14000 0.00000000 0.5380000 6.047000  88.80000 4.453400
## 29 0.772990  0.00000  8.14000 0.00000000 0.5380000 6.495000  94.40000 4.454700
## 30 1.002450  0.00000  8.14000 0.00000000 0.5380000 6.674000  87.30000 4.239000
## 31 1.130810  0.00000  8.14000 0.00000000 0.5380000 5.713000  94.10000 3.788106
## 32 1.354720  0.00000  8.14000 0.00000000 0.5380000 6.072000 100.00000 4.175000
## 33 1.387990  0.00000  8.14000 0.00000000 0.5380000 5.950000  82.00000 3.990000
## 34 1.151720  0.00000  8.14000 0.00000000 0.5380000 5.701000  95.00000 3.787200
## 35 1.612820  0.00000  8.14000 0.00000000 0.5380000 6.096000  96.90000 3.759800
## 36 3.592145  0.00000  5.96000 0.00000000 0.5540845 5.933000  68.20000 3.788106
## 37 0.097440  0.00000  5.96000 0.00000000 0.4990000 5.841000  61.40000 3.377900
## 38 0.080140  0.00000  5.96000 0.00000000 0.4990000 5.850000  41.50000 3.934200
## 39 0.175050  0.00000  5.96000 0.00000000 0.4990000 6.283719  68.25867 3.847300
## 40 0.027630 75.00000  2.95000 0.00000000 0.4280000 6.595000  21.80000 5.401100
##         rad      tax  ptratio    black    lstat
## 1  1.000000 296.0000 15.30000 396.9000  4.98000
## 2  2.000000 405.6443 17.80000 396.9000  9.14000
## 3  2.000000 242.0000 17.80000 392.8300  4.03000
## 4  3.000000 222.0000 18.70000 394.6300  2.94000
## 5  3.000000 222.0000 18.70000 396.9000  5.33000
## 6  3.000000 222.0000 18.70000 394.1200  5.21000
## 7  5.000000 405.6443 15.20000 395.6000 12.43000
## 8  5.000000 311.0000 15.20000 357.2426 19.15000
## 9  5.000000 311.0000 15.20000 386.6300 29.93000
## 10 5.000000 311.0000 15.20000 386.7100 12.70996
## 11 5.000000 311.0000 15.20000 392.5200 20.45000
## 12 5.000000 311.0000 15.20000 396.9000 13.27000
## 13 9.510965 311.0000 18.40173 390.5000 12.70996
## 14 9.510965 307.0000 21.00000 396.9000 12.70996
## 15 4.000000 307.0000 21.00000 380.0200 12.70996
## 16 4.000000 307.0000 21.00000 395.6200  8.47000
## 17 4.000000 405.6443 21.00000 386.8500  6.58000
## 18 4.000000 307.0000 21.00000 386.7500 14.67000
## 19 4.000000 307.0000 21.00000 288.9900 11.69000
## 20 4.000000 307.0000 21.00000 357.2426 11.28000
## 21 4.000000 307.0000 21.00000 376.5700 12.70996
## 22 4.000000 307.0000 21.00000 392.5300 13.83000
## 23 4.000000 307.0000 21.00000 396.9000 18.72000
## 24 4.000000 307.0000 21.00000 394.5400 19.88000
```

```
## 25 4.000000 307.0000 21.00000 394.3300 16.30000
## 26 4.000000 307.0000 21.00000 357.2426 16.51000
## 27 4.000000 307.0000 21.00000 376.8800 14.81000
## 28 4.000000 307.0000 21.00000 357.2426 17.28000
## 29 4.000000 307.0000 21.00000 387.9400 12.80000
## 30 4.000000 307.0000 21.00000 380.2300 11.98000
## 31 4.000000 307.0000 21.00000 360.1700 22.60000
## 32 4.000000 405.6443 21.00000 376.7300 13.04000
## 33 4.000000 307.0000 21.00000 232.6000 27.71000
## 34 4.000000 307.0000 21.00000 357.2426 12.70996
## 35 4.000000 307.0000 21.00000 248.3100 20.34000
## 36 9.510965 279.0000 19.20000 396.9000  9.68000
## 37 5.000000 279.0000 19.20000 377.5600 12.70996
## 38 9.510965 279.0000 19.20000 357.2426  8.77000
## 39 5.000000 279.0000 19.20000 393.4300 10.13000
## 40 3.000000 252.0000 18.40173 395.6300  4.32000
```