Lab 9

Jacob Minkin

11:59PM May 10, 2021

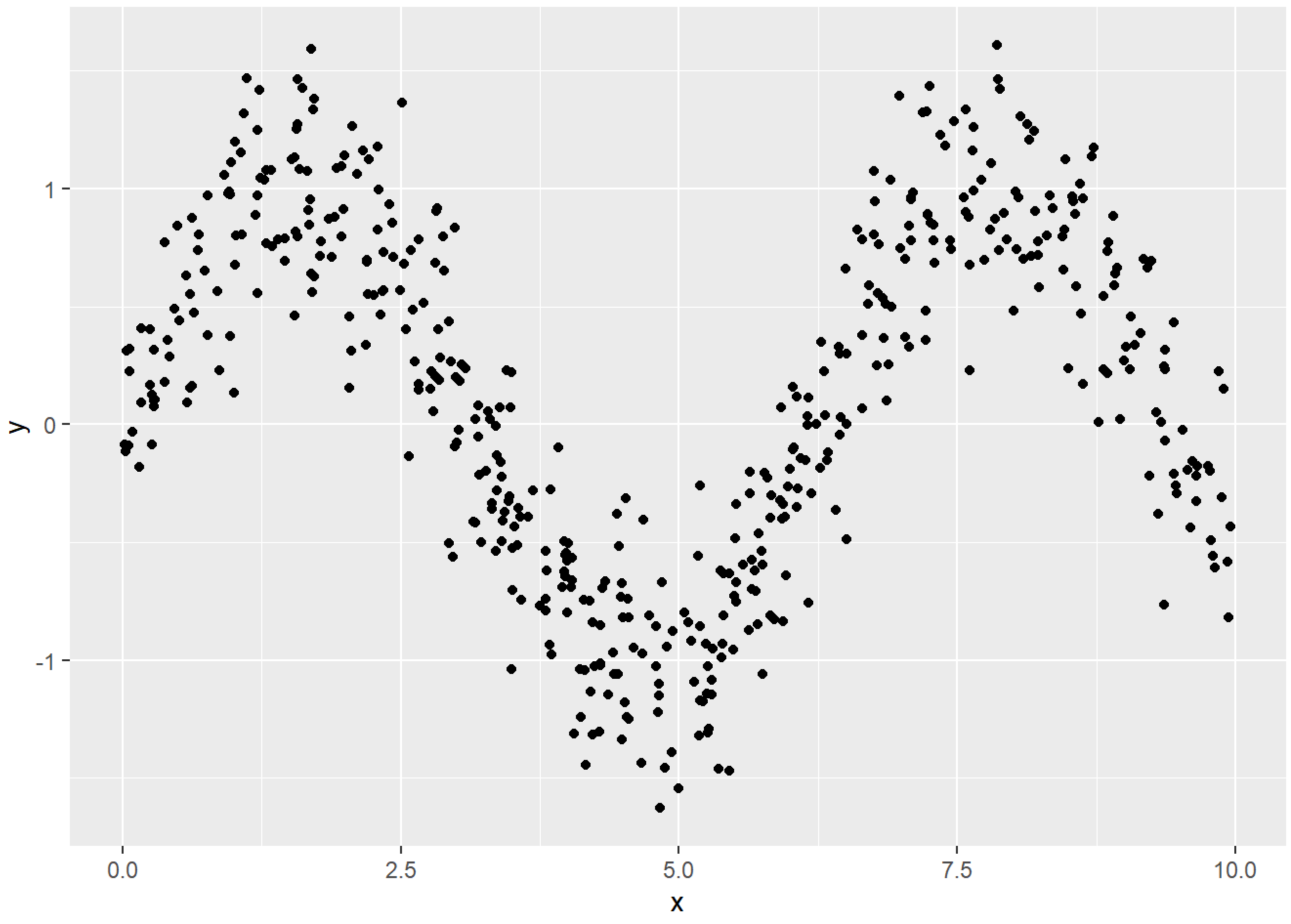
Here we will learn about trees, bagged trees and random forests. You can use the `YARF` package if it works, otherwise, use the `randomForest` package (the standard).

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))
```

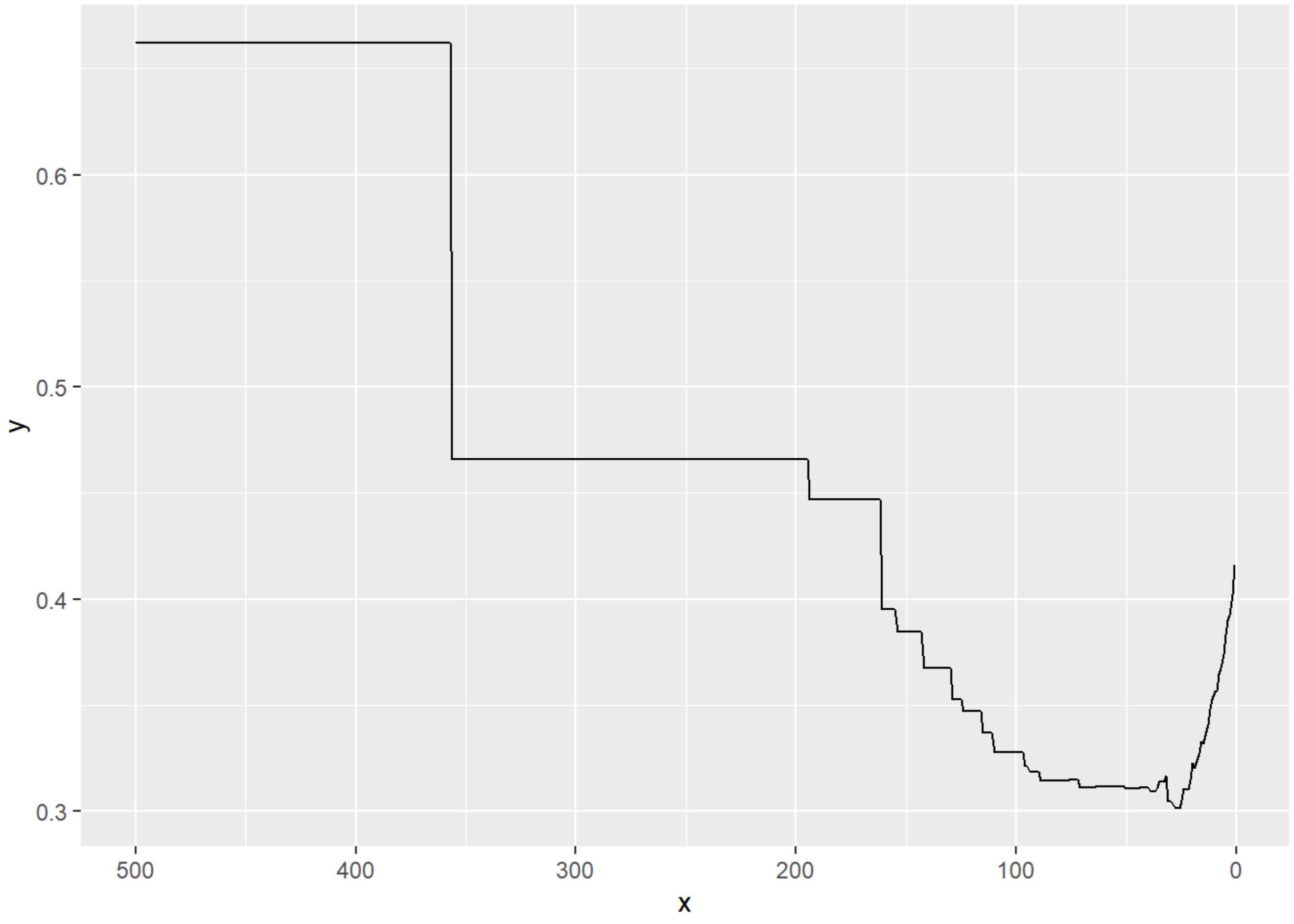


Create a test set of size 500 as well

```
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`. plot node size by OOS 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(data.frame(x = x_train), y_train, ntree = 1, replace = FALSE,
sampsize = n, nodesize = node_sizes[i])

  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()
```

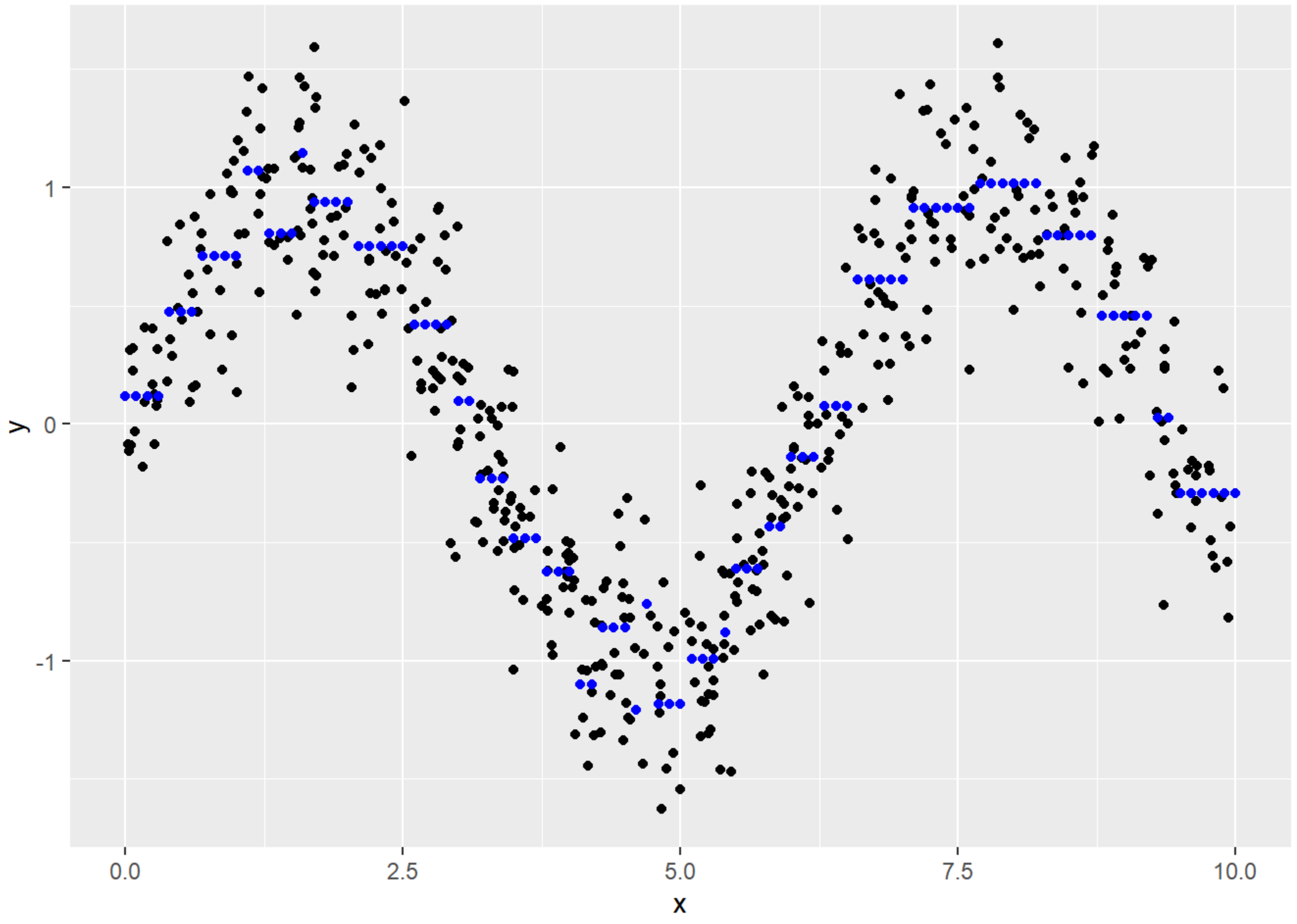


```
which.min(se_by_node_sizes)
## [1] 27
```

Plot the regression tree model with the optimal node size.

```
rf_mod = randomForest(data.frame(x = x_train), y_train, ntree = 1, replace = FALSE, sa
mpsize = n, nodesize = node_sizes[which.min(se_by_node_sizes)])
resolution = 0.1
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.

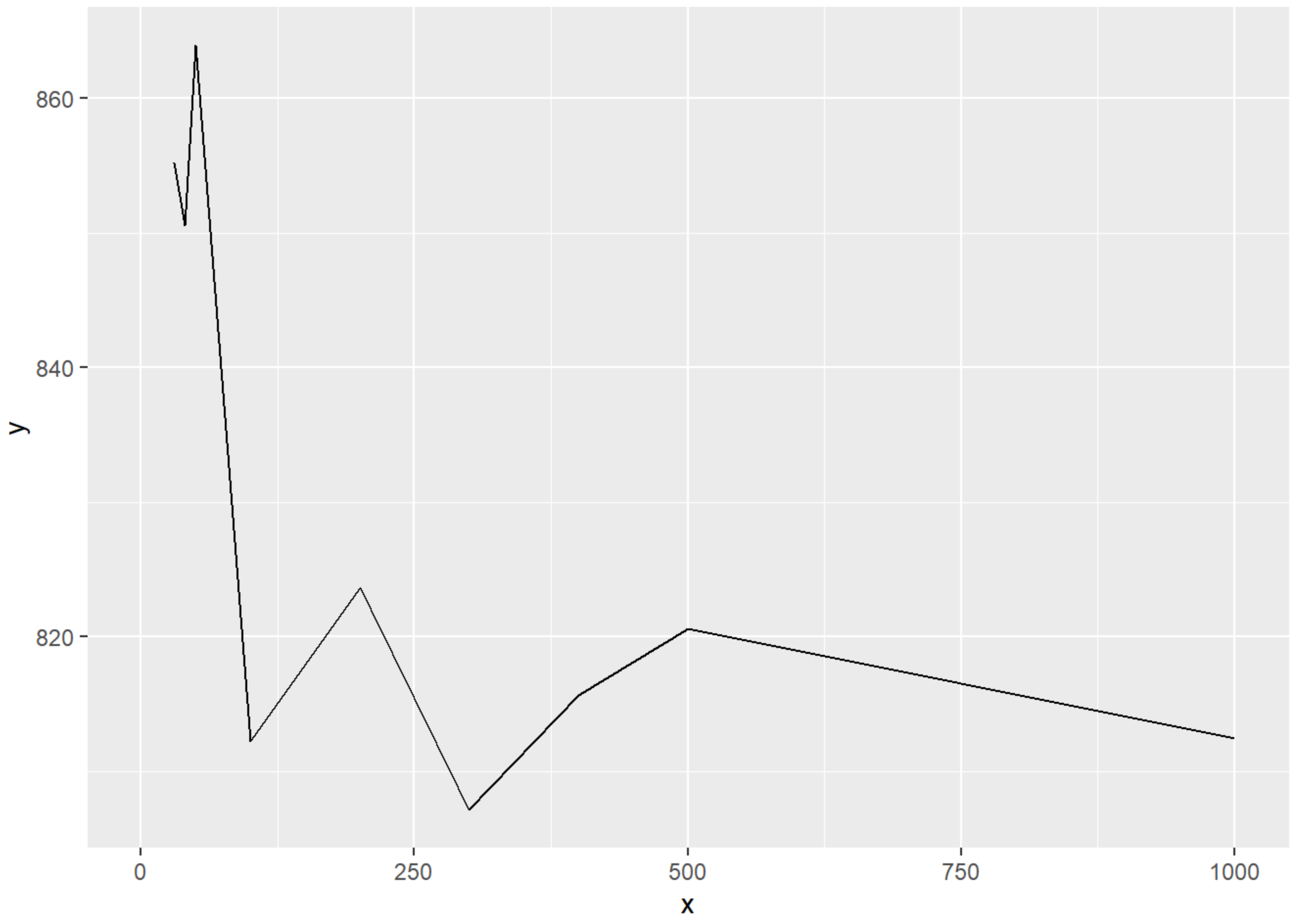
```
# x = seq(xmin, xmax, length.out = resolution)
#
# expe_g_x = g_average[1] + g_average[2] * x + g_average[3] * x^2 + g_average[4] * x^3
+ g_average[5] * x^4 + g_average[6] * x^5
#
# var_x_s = array(NA, Nsim)
# for (nsim in 1 : Nsim){
#   g_x = training_gs[nsim, 1] + training_gs[nsim, 2] * x + training_gs[nsim, 3] * x^2
+ training_gs[nsim, 4] * x^3 + training_gs[nsim, 5] * x^4 + training_gs[nsim, 6] * x^5
#   var_x_s[nsim] = mean((g_x - expe_g_x)^2)
# }
#
# expe_var_g = mean(var_x_s)
# expe_var_g
rm(list = ls())
```

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

```
pacman::p_load(tidyverse)

diamond_samp = diamonds %>%
  sample_n(2000)
```

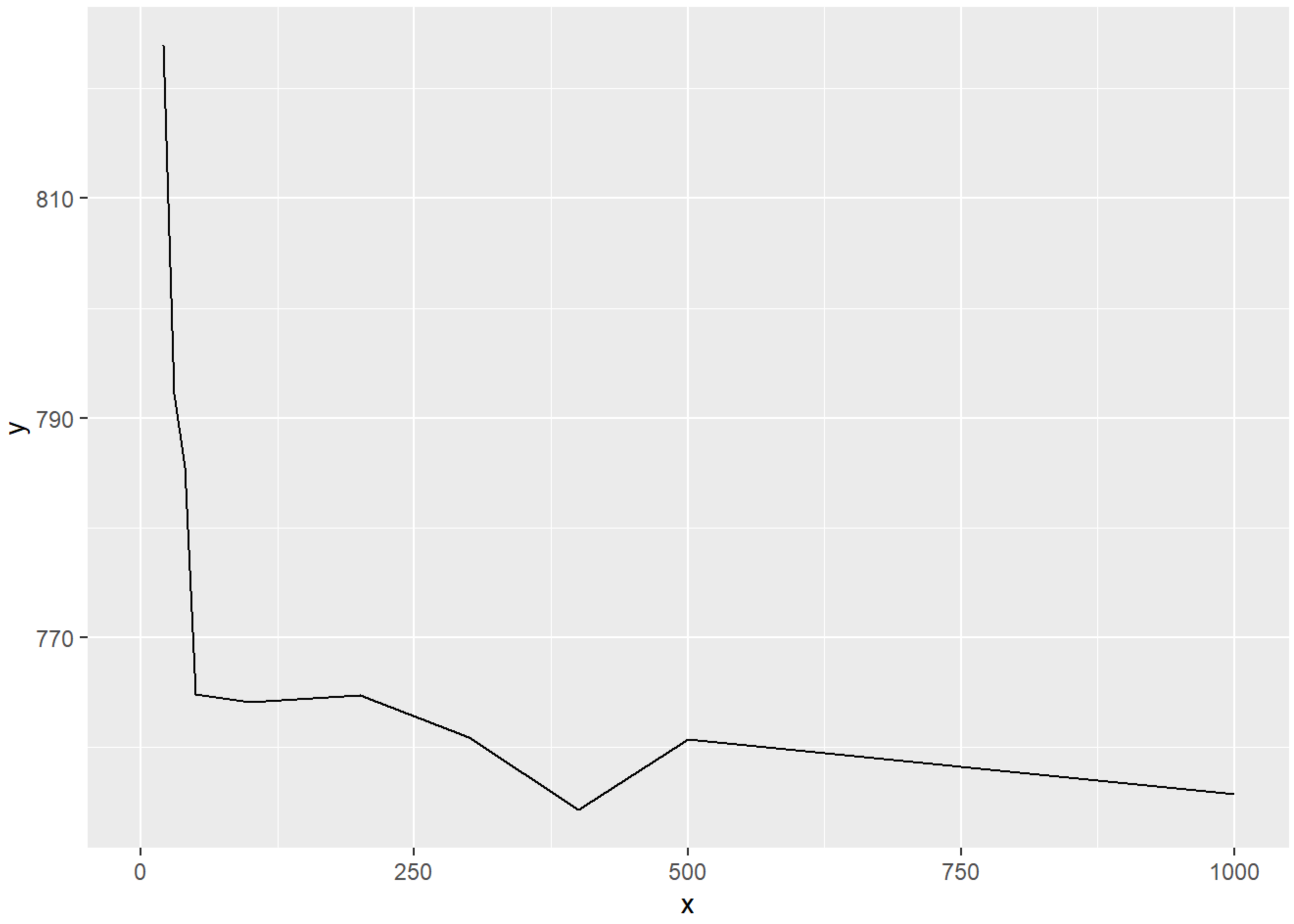
find the oob s_e for a RF model using 1, 2, 5, 10, 20, 30, 40, 50, 100, 200, 300, 400, 500, 1000 trees. If you are using the `randomForest` package, you can calculate oob residuals via `e_oob = y_train - rf_mod$predicted`.

```
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 = nco
l(diamond_samp)- 1)

  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 4 row(s) containing missing values (geom_path).
```

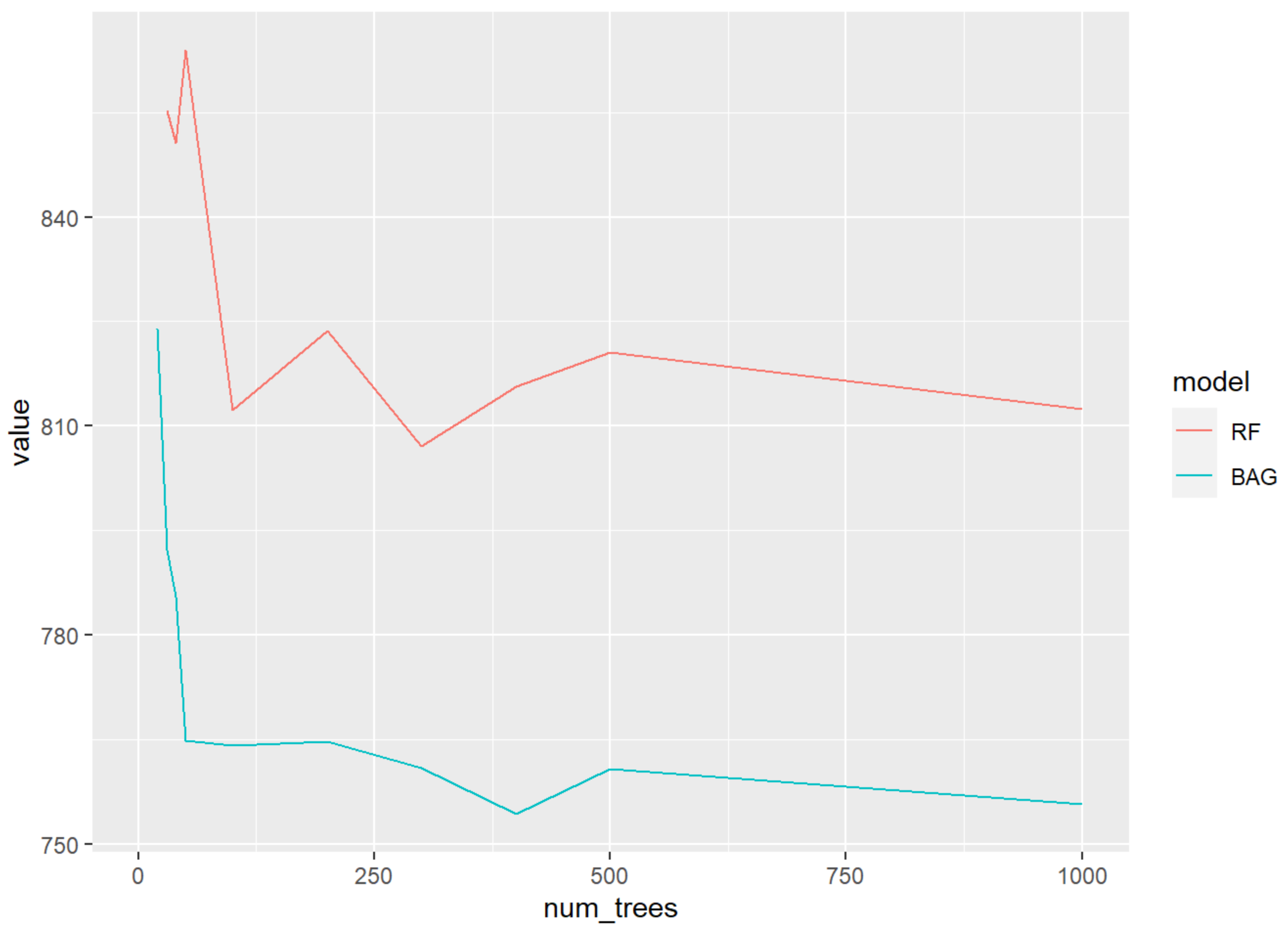


What is the percentage gain / loss in performance of the RF model vs bagged trees model?

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

##  [1]        NA        NA        NA        NA        NA  7.952740  8.277632

##  [8] 12.950894  6.300982  7.709662  6.072274  8.144311  7.873946  7.503979
```

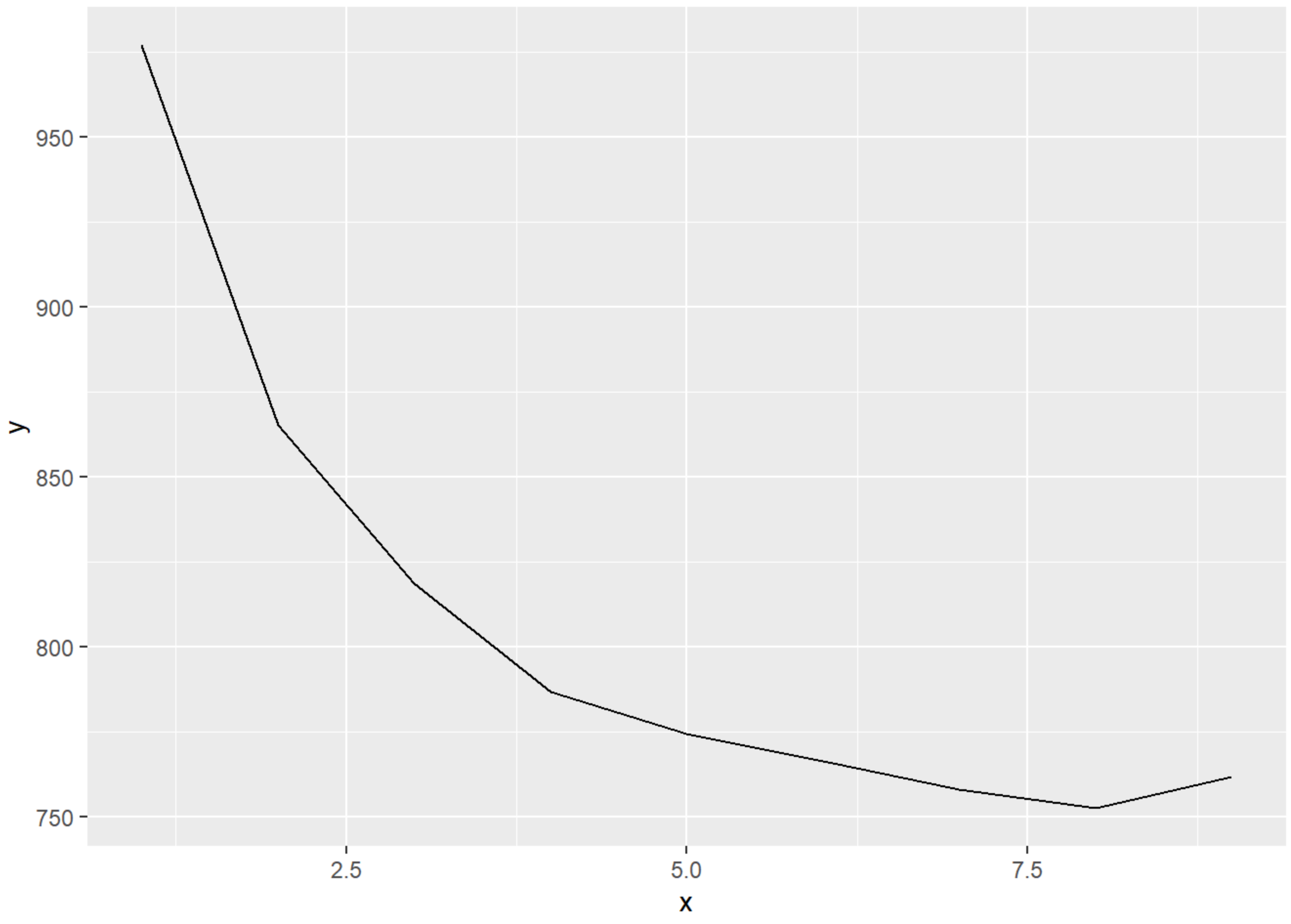
Plot bootstrap s_e by number of trees for both RF and bagged trees.

```
ggplot(rbind(data.frame(num_trees = num_trees, value = oob_se_by_num_trees, model = "R
F"), 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 9 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(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)
}

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



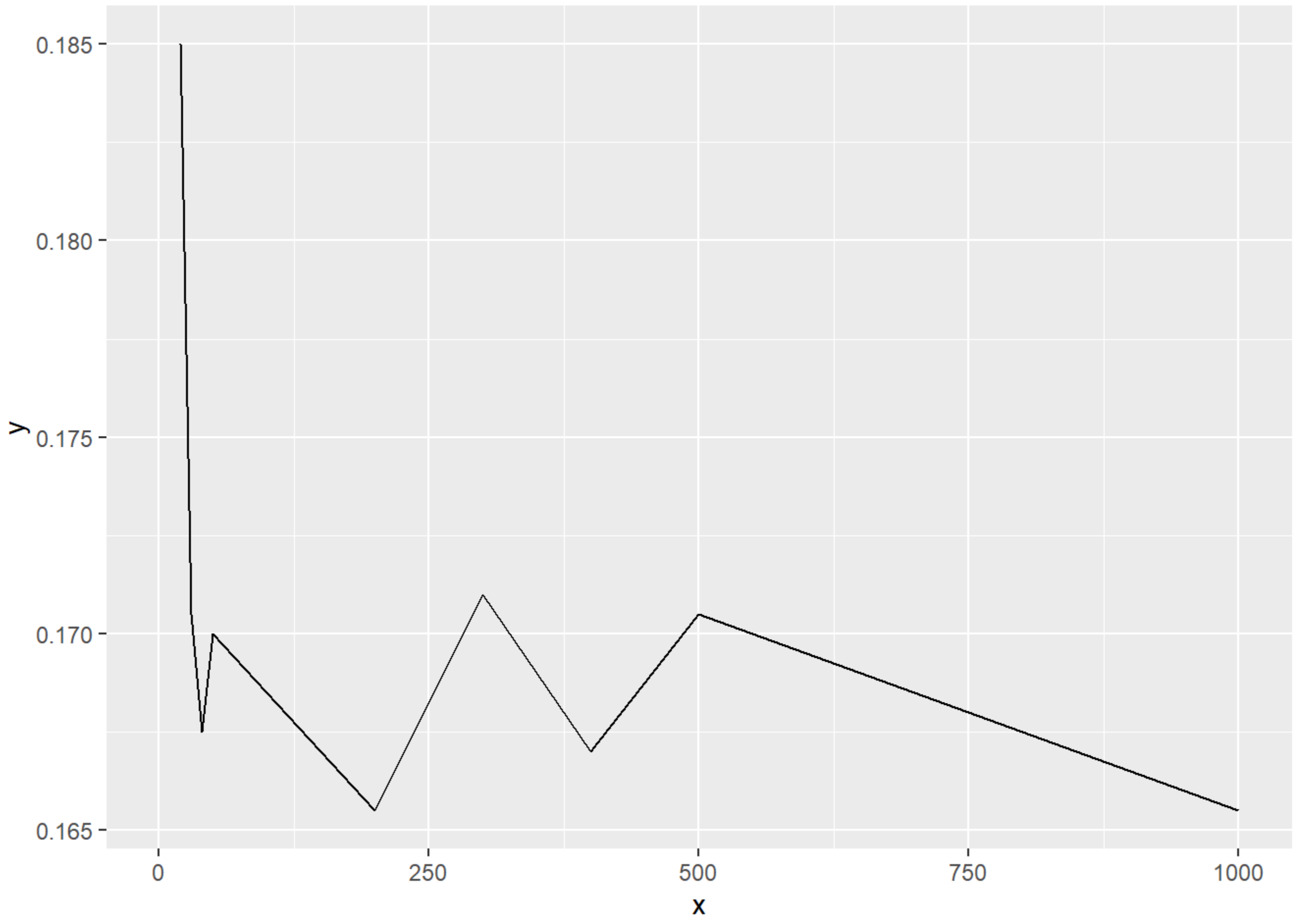
```
rm(list = ls())
```

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

```
pacman::p_load_gh("coatless/ucidata")
data(adult)
adult = na.omit(adult) #kill any observations with missingness

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

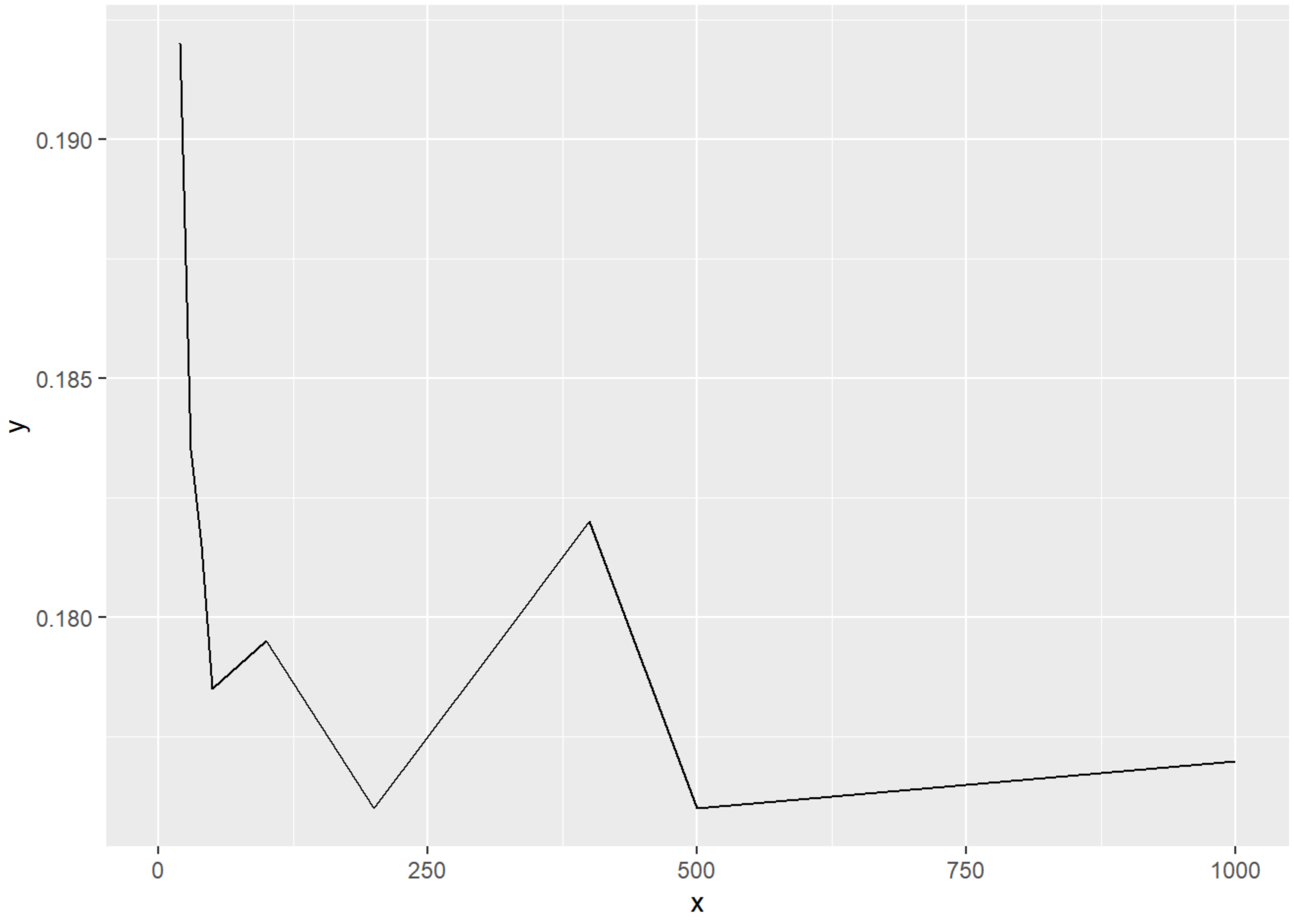
Using the adult data, find the bootstrap misclassification error for an RF model using 1, 2, 5, 10, 20, 30, 40, 50, 100, 200, 300, 400, 500, 1000 trees.

```
num_trees = c(1, 2, 5, 10, 20, 30, 40, 50, 100, 200, 300, 400, 500, 1000)
oob_se_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_se_by_num_trees[i] = mean(adult_samp$income != rf_mod$predicted)
}


ggplot(data.frame(x = num_trees, y = oob_se_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_se_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)- 1)
  oob_se_by_num_trees_bag[i] = mean(adult_samp$income != rf_mod$predicted)
}


ggplot(data.frame(x = num_trees, y = oob_se_by_num_trees_bag)) +
  geom_line(aes(x = x, y = y))
## Warning: Removed 4 row(s) containing missing values (geom_path).
```



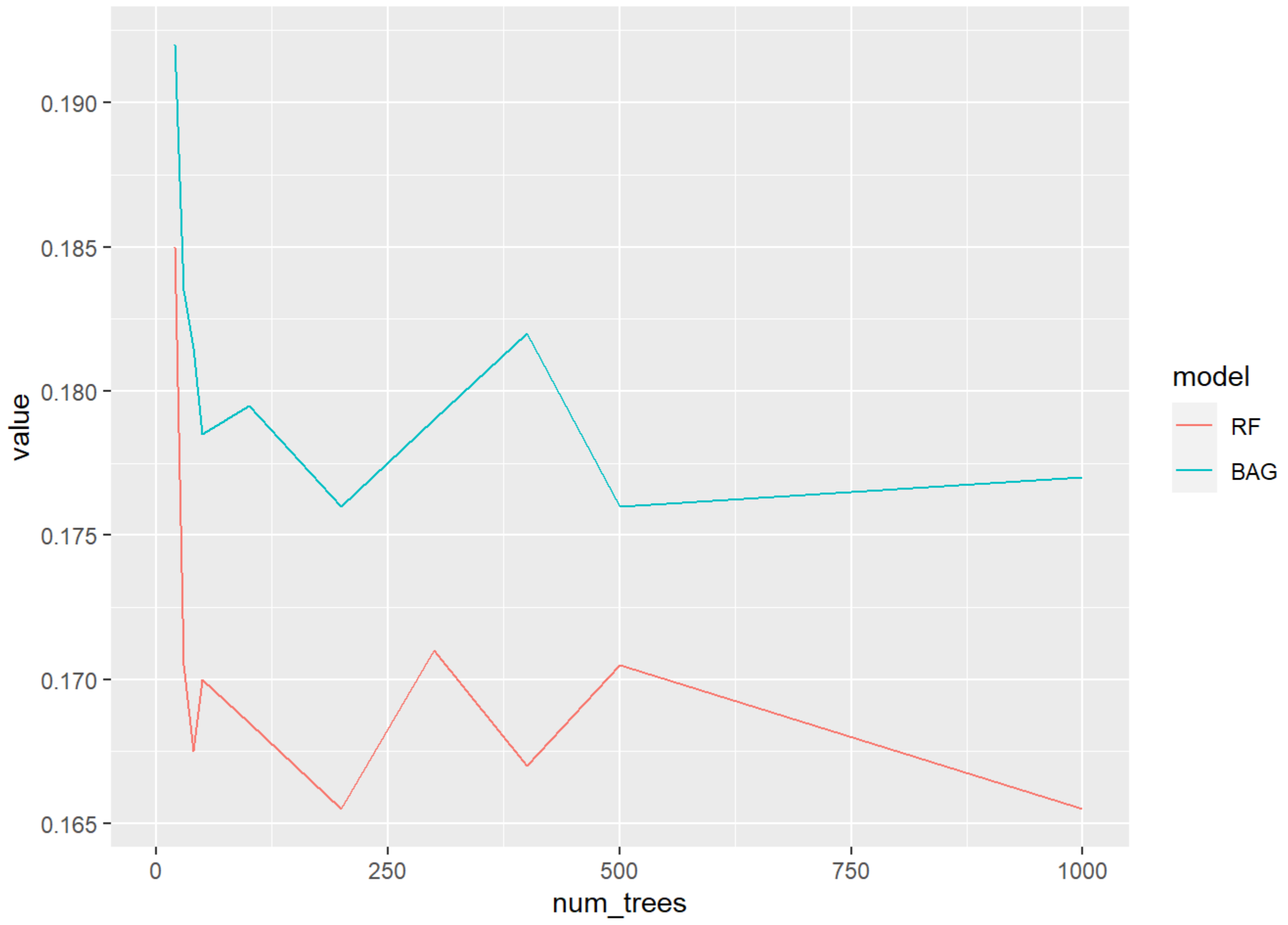
What is the percentage gain / loss in performance of the RF model vs bagged trees model?

```
(oob_se_by_num_trees - oob_se_by_num_trees_bag) /oob_se_by_num_trees_bag *100
##  [1]        NA        NA        NA        NA -3.645833 -7.084469 -7.713499
##  [8] -4.761905 -6.128134 -5.965909 -4.469274 -8.241758 -3.125000 -6.497175
```

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

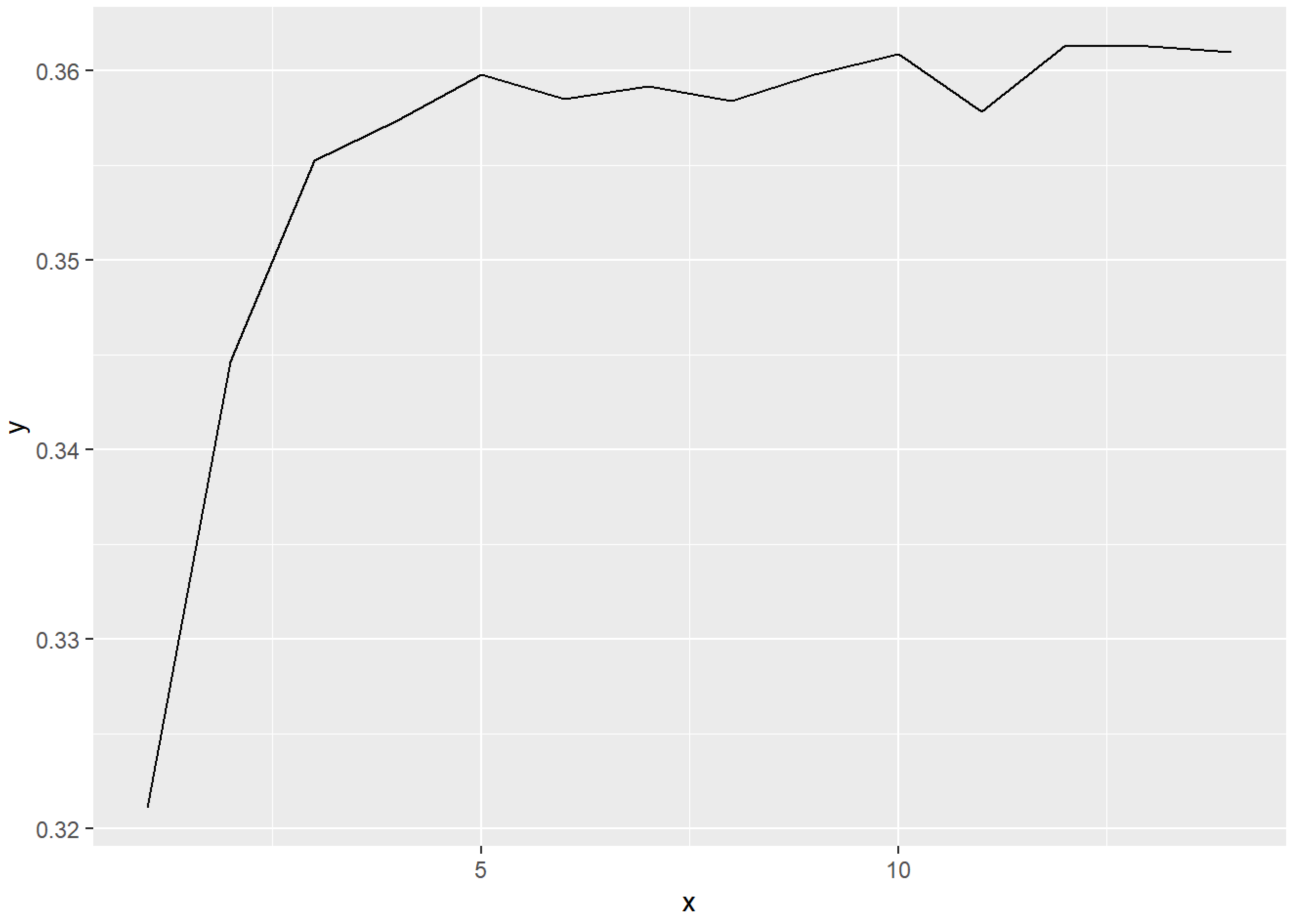
```
ggplot(rbind(data.frame(num_trees = num_trees, value = oob_se_by_num_trees, model = "R
F"), data.frame(num_trees = num_trees, value = oob_se_by_num_trees_bag, model = "BAG")
))+
  geom_line(aes(x = num_trees, y = value, color = model))
```

```
## Warning: Removed 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_se_by_mtrys = array(NA, length(mtrys))

for (i in 1:length(mtrys)){
  rf_mod = randomForest(income~., data = adult_samp, mtry = mtrys[i] )
  oob_se_by_mtrys[i] = mean(adult$income != rf_mod$predicted)
}
## Warning in `!=.default`(adult$income, rf_mod$predicted): longer object length is
## not a multiple of shorter object length
## Warning in is.na(e1) | is.na(e2): longer object length is not a multiple of
## shorter object length
## Warning in `!=.default`(adult$income, rf_mod$predicted): longer object length is
## not a multiple of shorter object length
## Warning in is.na(e1) | is.na(e2): longer object length is not a multiple of
## shorter object length
## Warning in `!=.default`(adult$income, rf_mod$predicted): longer object length is
## not a multiple of shorter object length
## Warning in is.na(e1) | is.na(e2): longer object length is not a multiple of
## shorter object length
## Warning in `!=.default`(adult$income, rf_mod$predicted): longer object length is
## not a multiple of shorter object length
## Warning in is.na(e1) | is.na(e2): longer object length is not a multiple of
## shorter object length
## Warning in `!=.default`(adult$income, rf_mod$predicted): longer object length is
## not a multiple of shorter object length
## Warning in is.na(e1) | is.na(e2): longer object length is not a multiple of
## shorter object length
## Warning in `!=.default`(adult$income, rf_mod$predicted): longer object length is
## not a multiple of shorter object length
## Warning in is.na(e1) | is.na(e2): longer object length is not a multiple of
## shorter object length
## Warning in `!=.default`(adult$income, rf_mod$predicted): longer object length is
## not a multiple of shorter object length
## Warning in is.na(e1) | is.na(e2): longer object length is not a multiple of
## shorter object length
## Warning in `!=.default`(adult$income, rf_mod$predicted): longer object length is
## not a multiple of shorter object length
## Warning in is.na(e1) | is.na(e2): longer object length is not a multiple of
## shorter object length
## Warning in `!=.default`(adult$income, rf_mod$predicted): longer object length is
## not a multiple of shorter object length
## Warning in is.na(e1) | is.na(e2): longer object length is not a multiple of
## shorter object length
## Warning in `!=.default`(adult$income, rf_mod$predicted): longer object length is
## not a multiple of shorter object length
## Warning in is.na(e1) | is.na(e2): longer object length is not a multiple of
## shorter object length
## Warning in `!=.default`(adult$income, rf_mod$predicted): longer object length is
## not a multiple of shorter object length
## Warning in is.na(e1) | is.na(e2): longer object length is not a multiple of
## shorter object length
## Warning in `!=.default`(adult$income, rf_mod$predicted): longer object length is
## not a multiple of shorter object length
## Warning in is.na(e1) | is.na(e2): longer object length is not a multiple of
## shorter object length
## Warning in `!=.default`(adult$income, rf_mod$predicted): longer object length is
## not a multiple of shorter object length
## Warning in is.na(e1) | is.na(e2): longer object length is not a multiple of
## shorter object length
ggplot(data.frame(x = mtrys, y = oob_se_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){
  if(is.null(mtry)){
    mtry = 0.5 * ncol(x)
  }

  pacman::p_load(tidyverse)
  list_ols = list(NA)
  for (i in 1:num_ols_models){
    x_sub = x %>% sample(x, mtry, replace = FALSE)
    list_ols (i) = lm(y ~ ., X_sub)
  }
  list_ols
}
```

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

```
pacman::p_load(data.table, tidyverse)
boston = MASS::Boston %>% data.table
y = boston$medv
X = boston
X$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`.

```
punch_hole = function(x, prob_missing){
  a = matrix(rbinom(ncol(x)*nrow(x), size = 1, prob = prob_missing), nrow = nrow(x), n
col = ncol(x))
  x[which(a==1)] = NA
  a = x
}
```

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

```
Xmiss = punch_hole(as.matrix(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.

```
pacman::p_load(missForest, skimr)
Ximp = missForest(data.frame(Xmiss), sampsize = rep(100, ncol(X)))$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!
skim(Ximp)
```

Data summary

Name	Ximp
Number of rows	506
Number of columns	13
Column type frequency:	
numeric	13
Group variables	None

Variable type: numeric

skim_variable	n_missing	complete_rate	mean	sd	p0	p25	p50	p75	p100	hist
crim	0	1	3.74	8.73	0.01	0.08	0.25	3.70	88.98	▇▁▁▁▁
zn	0	1	11.39	22.97	0.00	0.00	0.00	12.50	100.00	▇▂▁▁▁
indus	0	1	11.16	6.76	0.46	5.19	9.20	18.10	27.74	▇▆▁▇▁
chas	0	1	0.07	0.23	0.00	0.00	0.00	0.00	1.00	▇▁▁▁▁
nox	0	1	0.56	0.12	0.38	0.45	0.54	0.63	0.87	▇▇▅▅▁
rm	0	1	6.29	0.69	3.56	5.92	6.21	6.62	8.78	▁▁▇▂▁
age	0	1	68.61	27.54	2.90	45.83	76.95	93.47	100.00	▁▂▂▃▇
dis	0	1	3.79	2.07	1.13	2.11	3.30	5.12	12.13	▇▅▂▁▁
rad	0	1	9.58	8.66	1.00	4.00	5.00	23.58	24.00	▇▂▁▁▃
tax	0	1	408.72	167.11	187.00	284.00	330.00	663.37	711.00	▇▇▃▁▇
ptratio	0	1	18.48	2.07	12.60	17.40	18.90	20.20	22.00	▁▃▅▅▇
black	0	1	356.42	87.70	0.32	372.18	390.69	395.66	396.90	▁▁▁▁▇
lstat	0	1	12.61	7.04	1.73	7.14	11.30	16.64	37.97	▇▇▅▂▁