

Lab 9

Meihe Liu

11:59PM May 10, 2021

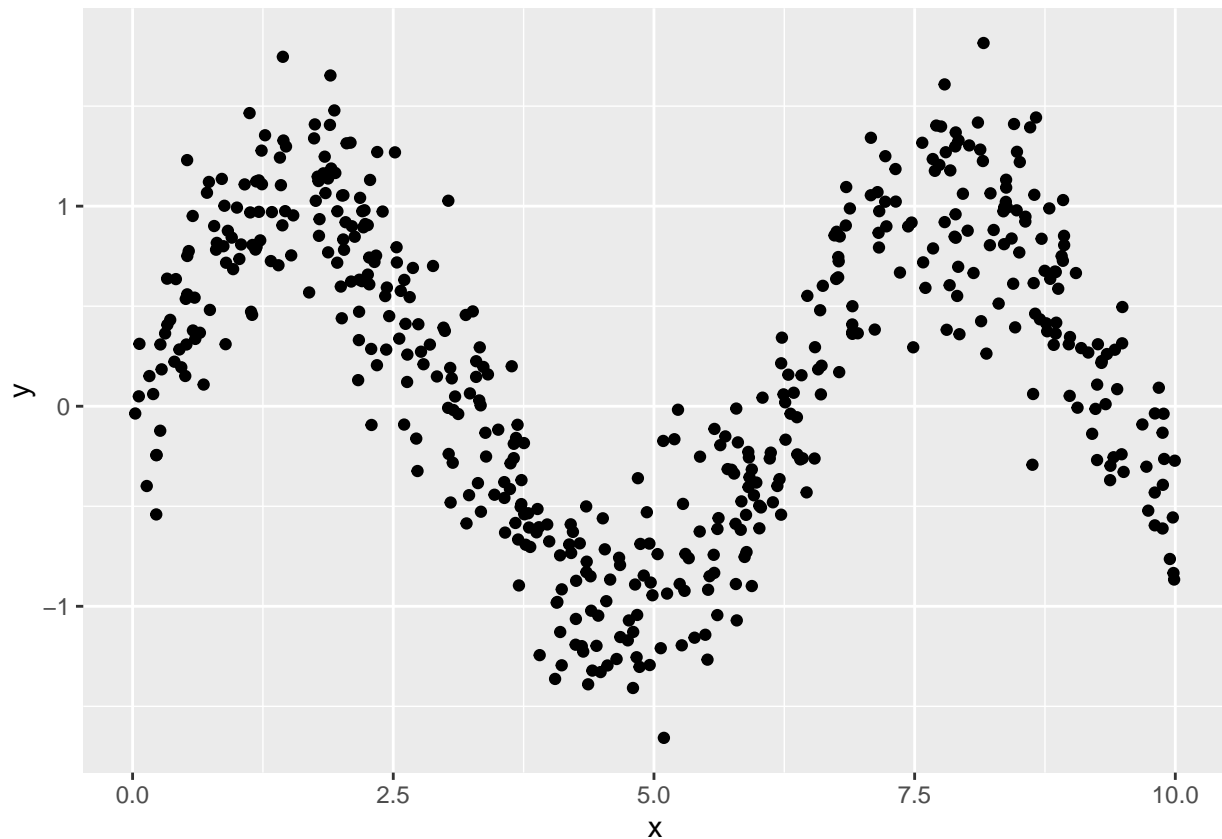
Here we will learn about trees, bagged trees and random forests. You can use the **YARF** package if it works, otherwise, use the **randomForest** package (the standard).

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))
```

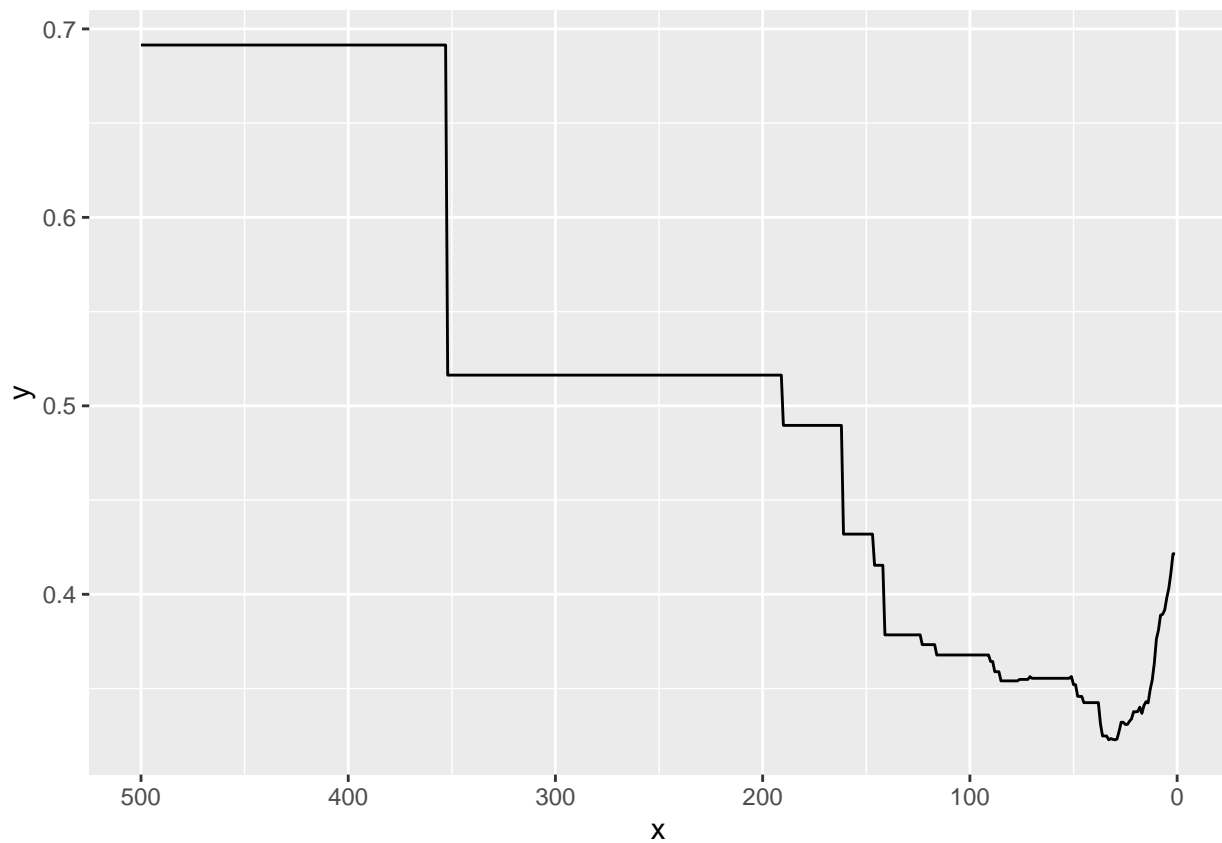


Create a test set of size 500 as well

```
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`, `sampsiz = n` (`mtry` is already set to be 1 because there is only one feature) and then you can set `nodesize`. Plot node size by out of sample Se.

```
pacman::p_load(randomForest)
node_size = 1:n
se_by_node_size = array(NA, length(node_size))
for(i in 1:length(node_size)){
  rf_mod = randomForest(x = data.frame(x=x_train), y=y_train, ntree = 1, replace = FALSE, sampsize = n,
    yhat_test = predict(rf_mod, data.frame(x=x_test))
    se_by_node_size[i] = sd(y_test - yhat_test)
}
ggplot(data.frame(x = node_size, y=se_by_node_size))+
  geom_line(aes(x=x,y=y))+
  scale_x_reverse()
```

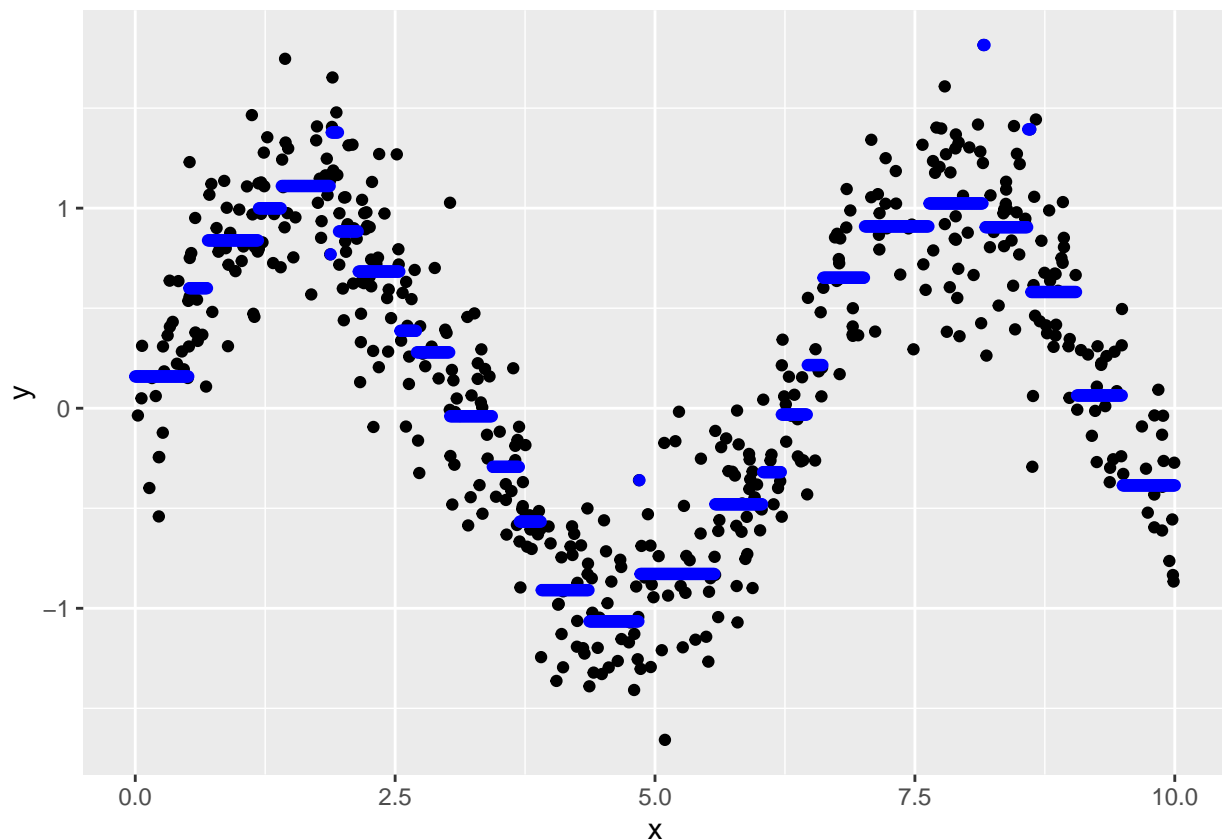


```
which.min(se_by_node_size)
```

```
## [1] 30
```

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())
```

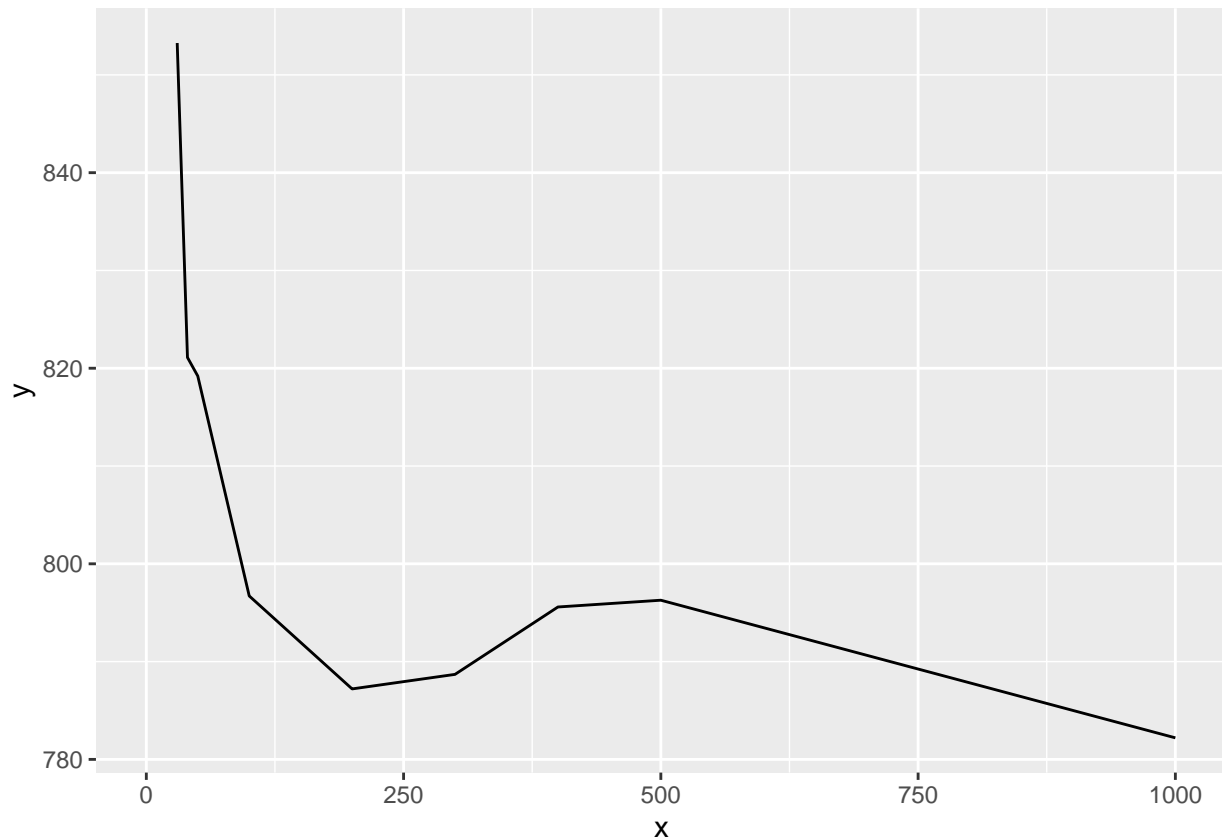
Take a sample of $n = 2000$ observations from the diamonds data.

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

find the bootstrap 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))
```

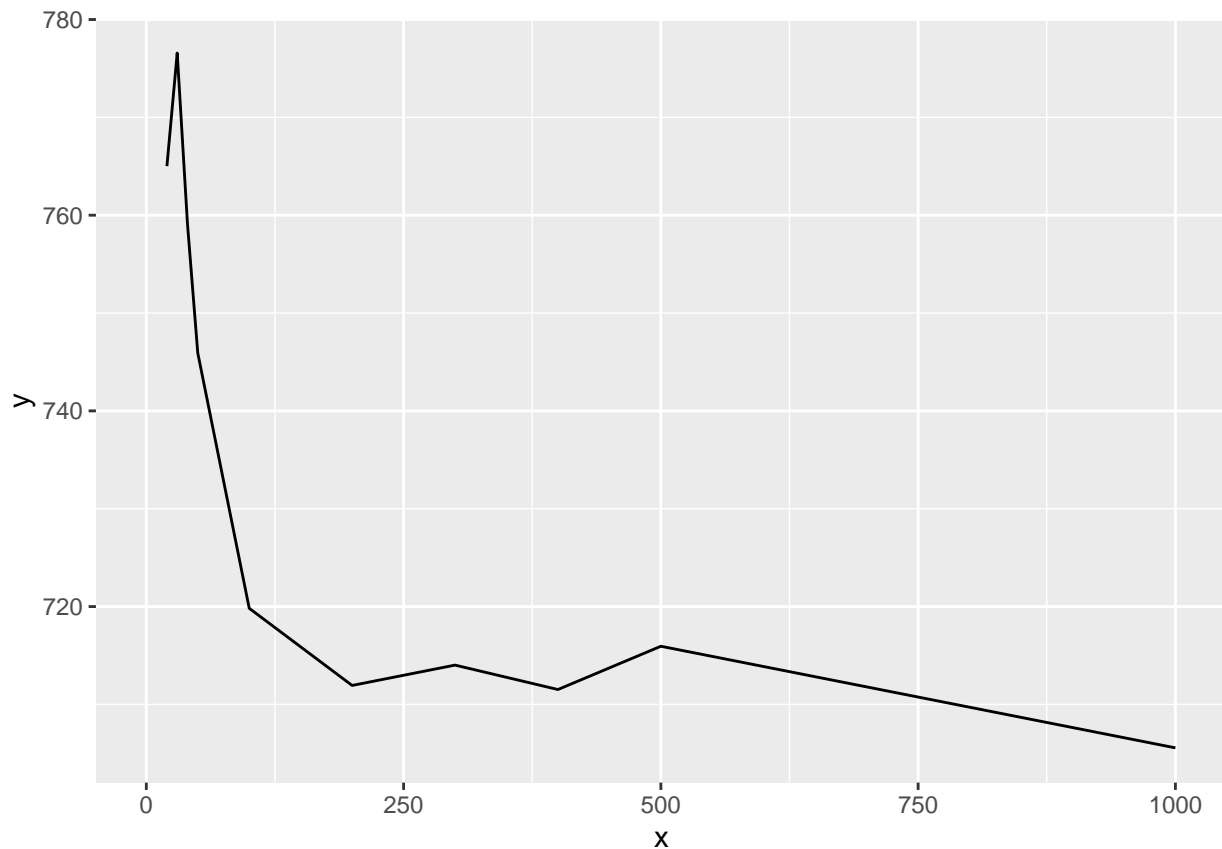
```
## Warning: Removed 5 row(s) containing missing values (geom_path).
```



Using the diamonds data, find the bootstrap oob_se 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)-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?

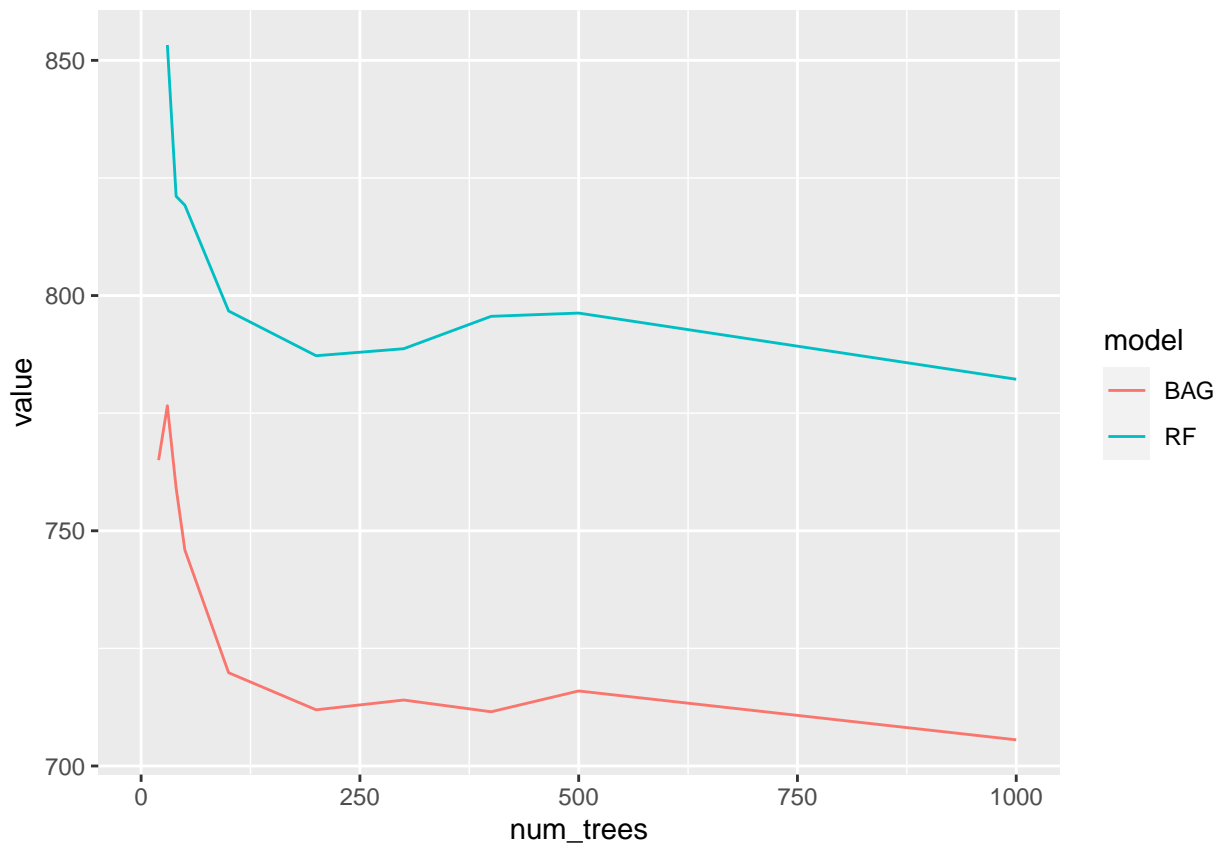
```
(oob_se_by_num_trees - oob_se_by_num_trees_bag)/oob_se_by_num_trees_bag * 100
```

```
## [1]      NA      NA      NA      NA      NA  9.874400  8.166730
## [8]  9.825376 10.682008 10.572396 10.458775 11.814578 11.220597 10.862783
```

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

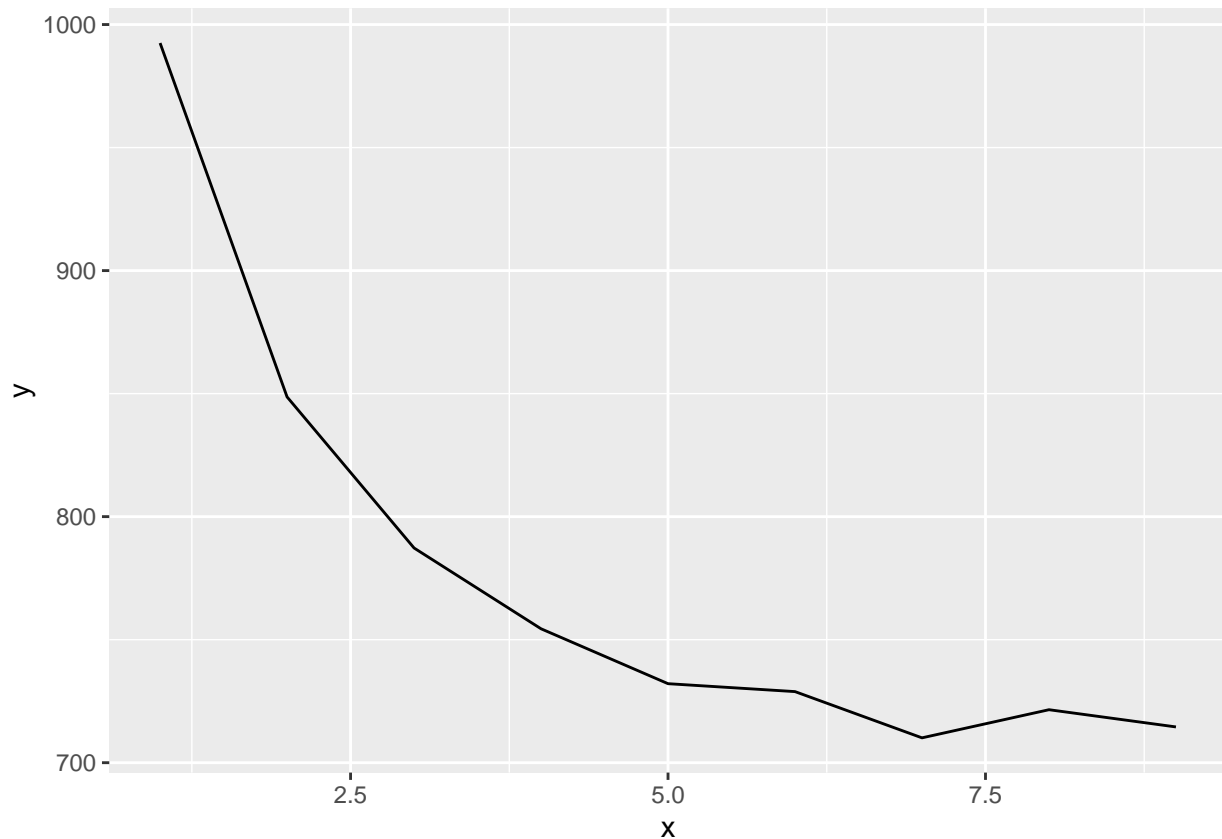
```
ggplot(rbind(data.frame(num_trees = num_trees, value = oob_se_by_num_trees,model = "RF"),data.frame(num_trees = num_trees, value = oob_se_by_num_trees_bag,model = "bagged")),
  geom_line(aes(x=num_trees,y=value, col=model)))
```

```
## 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_se` 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))
```



Plot bootstrap s_e by mtry.

```
#TO-DO
```

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

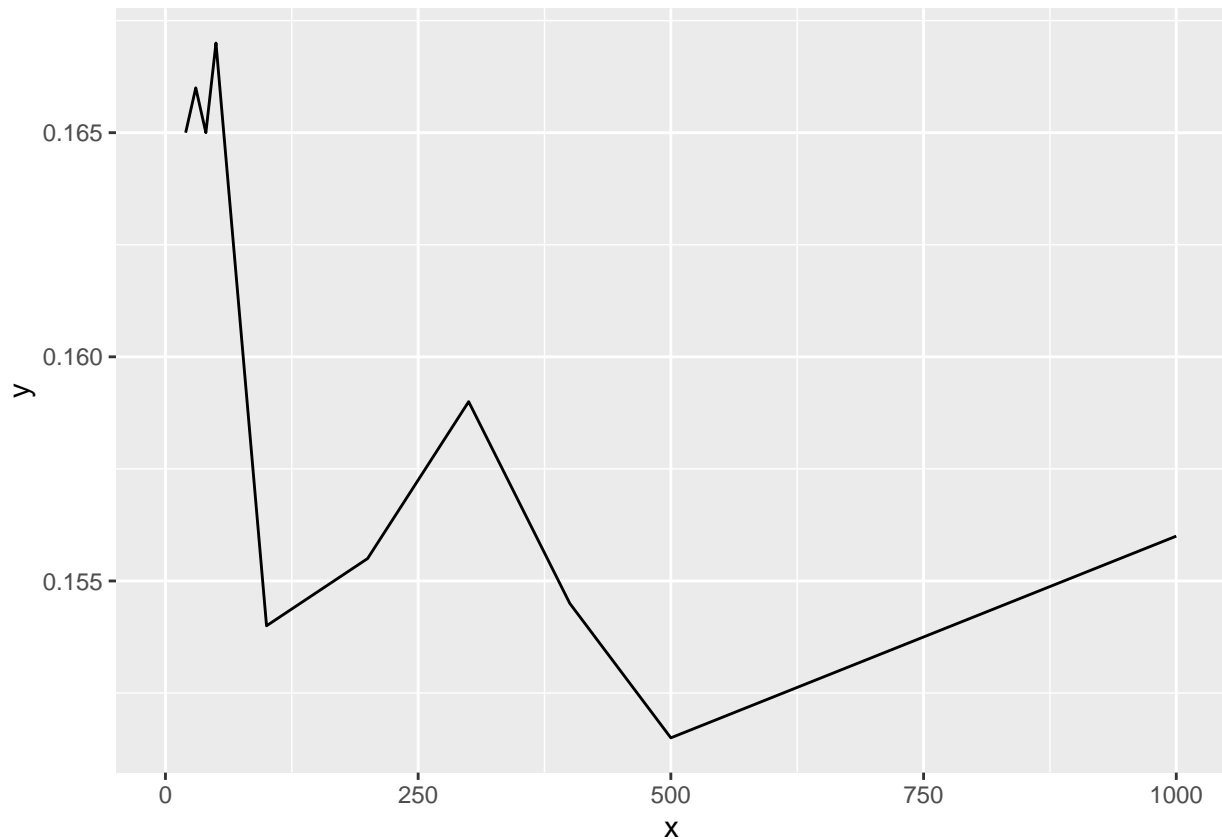
Take a sample of $n = 2000$ observations from the adult data.

```
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))
```

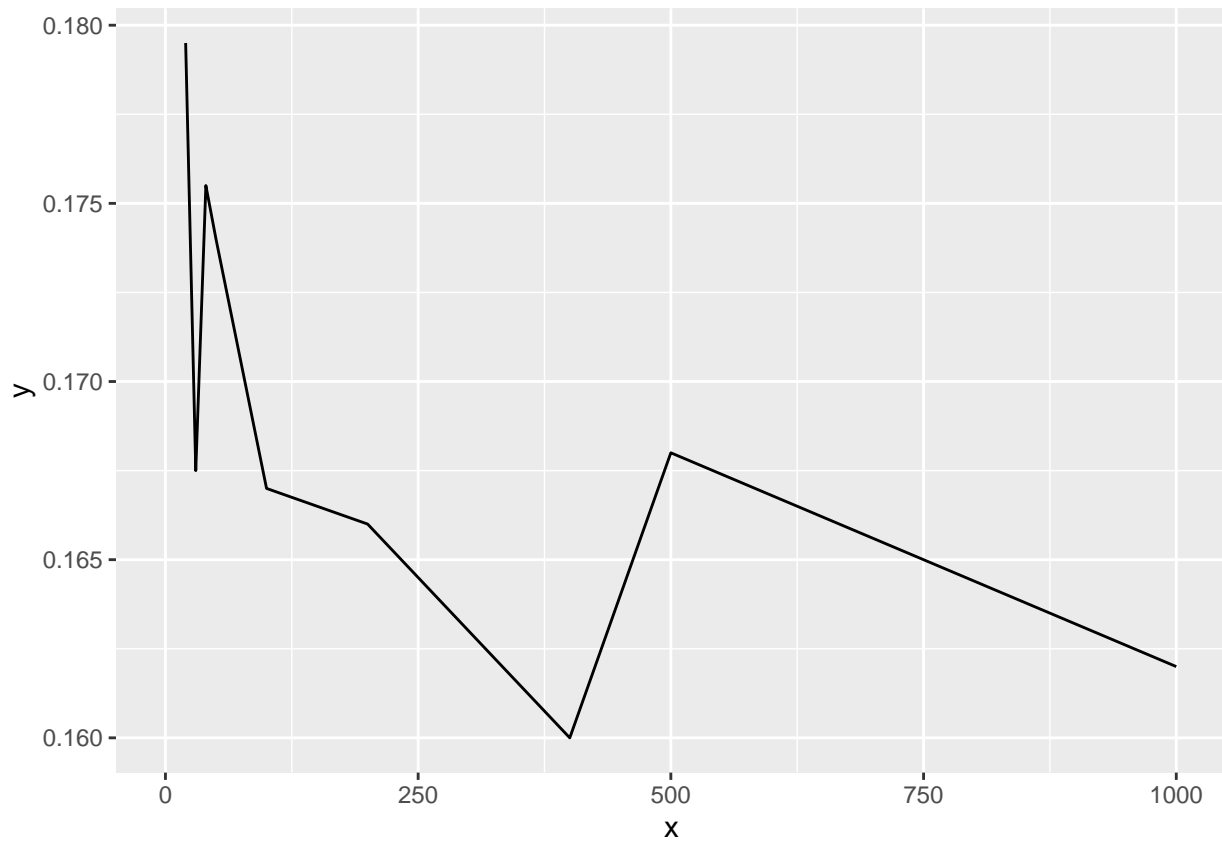
```
## Warning: Removed 4 row(s) containing missing values (geom_path).
```

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.

```
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)-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).



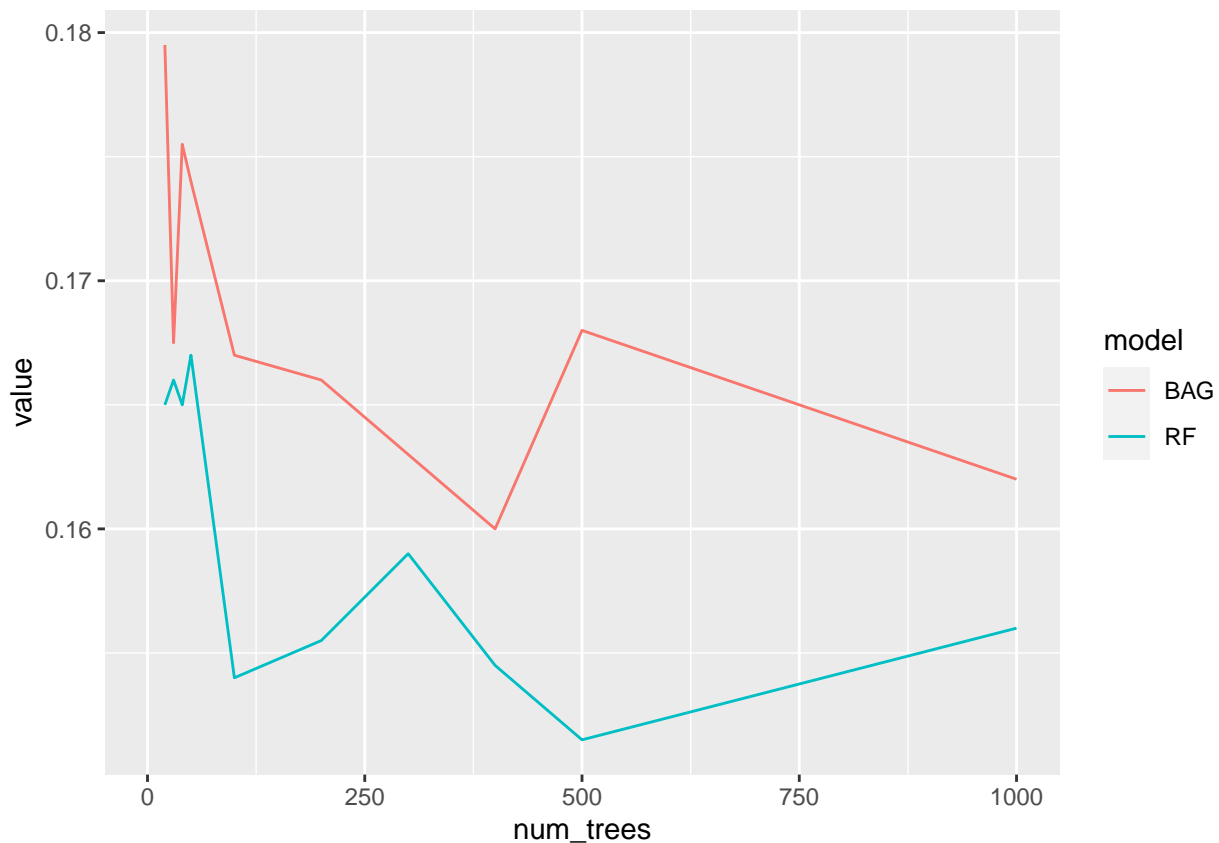
What is the percentage gain / loss in performance of the RF model vs bagged trees model?

#T0-D0

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

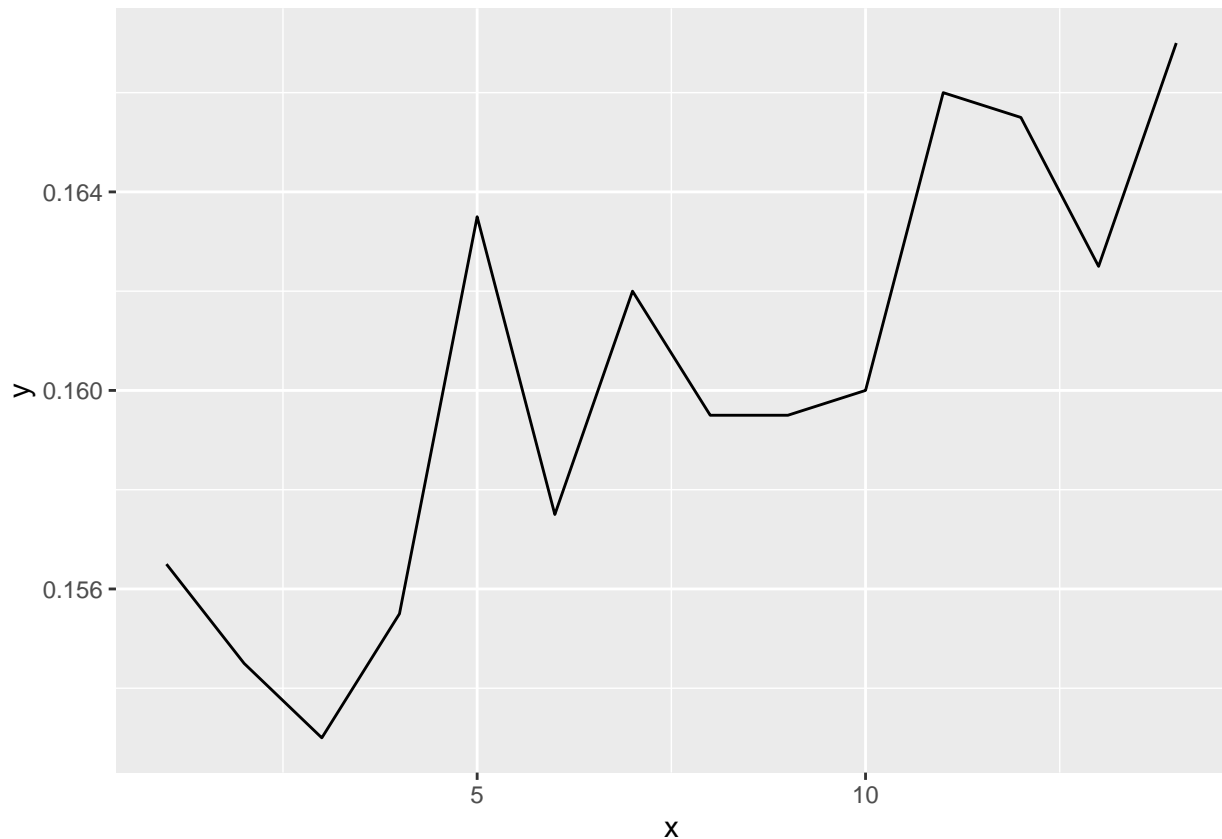
```
ggplot(rbind(data.frame(num_trees = num_trees, value = oob_se_by_num_trees, model = "RF"), data.frame(num_trees = num_trees, value = oob_se_by_num_trees, model = "bagged")),
  geom_line(aes(x=num_trees, y=value, col=model)))
```

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_samp$income != rf_mod$predicted)
}
ggplot(data.frame(x=mtrys, y=oob_se_by_mtrys))+
  geom_line(aes(x=x,y=y))
```



Plot bootstrap misclassification error by `mtry`.

#T0-D0

```
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`.

#T0-D0

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

#T0-D0

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`.

#T0-D0

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

#TO-DO

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.

#TO-DO