Lab 9

Loyd Flores

#YARF

For the next couple of labs, I want you to make some use of a package I wrote that offers convenient and flexible tree-building and random forest-building. Make sure you have a JDK installed first

https://www.oracle.com/java/technologies/downloads/

Then try to install rJava

```
options(java.parameters = "-Xmx4000m")
pacman::p_load(rJava)
.jinit()
```

If you have error, messages, try to google them. Everyone has trouble with rJava!

If that worked, please try to run the following which will install YARF from my github:

```
if (!pacman::p_isinstalled(YARF)){
  pacman::p_install_gh("kapelner/YARF/YARFJARs", ref = "dev")
  pacman::p_install_gh("kapelner/YARF/YARF", ref = "dev", force = TRUE)
}

pacman::p_load(YARF)
```

```
## YARF can now make use of 11 cores.
```

```
print("Package Loaded.")
```

```
## [1] "Package Loaded."
```

Please try to fix the error messages (if they exist) as best as you can. I can help on slack.

#Data Munging: a realistic exercise

This lab exercise may be the most important lab of the semester in terms of real-world experience and "putting it all together". We will be constructing a data frame which will then get passed on to the model-building. So this emulates the pre-steps necessary to get to the point where we assume we're at in this class.

We will be joining three datasets in an effort to make a design matrix that predicts if a bill will be paid on time. Clean up and load up the three files. Then I'll rename a few features and then we can examine the data frames.

Make sure you set the directory of RStudio to the directory where this file lives and make sure you download the bills_dataset folder from github (you can do this via `git pull` and then copying that directory over).

```
#setwd(...)
rm(list = ls())
pacman::p_load(tidyverse, magrittr, data.table, R.utils)
bills = fread("bills_dataset/bills.csv.bz2")
payments = fread("bills_dataset/payments.csv.bz2")
discounts = fread("bills_dataset/discounts.csv.bz2")
setnames(bills, "amount", "tot_amount")
setnames(payments, "amount", "paid_amount")
skimr::skim(bills)
```

Table 1: Data summary

Name	bills
Number of rows	226434
Number of columns	6
Key	NULL
Column type frequency:	
Date	2
numeric	4
Group variables	None

Variable type: Date

skim_variable	n_missing	complete_rate	min	max	median	n_unique
due_date	0	1	2010-03-18	2019-08-15	2017-01-17	1820
invoice_date	0	1	2010-02-16	2019-07-01	2016-12-22	1728

Variable type: numeric

skim_variable	n_missing	complete_rate	mean	sd	p0	p25	p50	p75	p100	hist
id	0	1.00	16400703	1025245.2	5000000.00	16094545.75	16725958.00	17104874.50	17619337	
tot_amount	0	1.00	100000	10000.0	99475.01	99475.13	99476.08	99484.13	1394143	
customer_id	0	1.00	14477367	605253.2	7079442.00	14569622.00	14669157.00	14689861.00	15755432	
discount_id	1313	0.99	6608438	1296965.4	5000000.00	5693147.00	5693147.00	7708050.00	9094345	

```
skimr::skim(payments)
```

Table 4: Data summary

Name	payments
Number of rows	194850
Number of columns	4
Key	NULL
Column type frequency:	
Date	1

numeric	3
Group variables	None

Variable type: Date

skim_variable	n_missing	complete_rate	min	max	median	n_unique
transaction_date	0	1	2012-01-01	2017-07-24	2017-04-29	1316

Variable type: numeric

skim_variable	n_missing	complete_rate	mean	sd	p0	p25	p50	p75	p100	hist
id	0	1	16217248	1007507.1	5000000.00	15845938.00	16514889.00	16911136.25	17471494.0	
paid_amount	0	1	100000	10000.0	99148.07	99148.31	99150.04	99168.01	905403.6	
bill_id	0	1	16510279	960666.4	5000000.00	16283556.00	16826344.50	17088825.00	17619337.0	

```
skimr::skim(discounts)
```

Table 7: Data summary

Name	discounts
Number of rows	60
Number of columns	4
Key	NULL
Column type frequency:	
numeric	4
Group variables	None

Variable type: numeric

skim_variable	n_missing	complete_rate	mean	sd	p0	p25	p50	p75	p100	hist
id	0	1.00	8143802.88	900801.43	5e+06	7756454	8417592	8812157	9094345	
num_days	21	0.65	56.51	80.90	0e+00	15	30	60	365	
pct_off	44	0.27	1.30	0.61	0e+00	1	1	2	2	
days_until_discount	44	0.27	16.69	15.26	5e+00	10	10	15	60	

The unit we care about is the bill. The y metric we care about will be "paid in full" which is 1 if the company paid their total amount (we will generate this y metric later).

Since this is the response, we would like to construct the very best design matrix in order to predict y.

First, join the three datasets in an intelligent way. You will need to examine the datasets beforehand.

```
#TO-DO
bills_and_payments = left_join(bills, payments, by = join_by("id" == "bill_id"))
bills_and_payments_and_discounts = left_join(bills_and_payments, discounts, by = join_by("discount_id" =
```

Now create the binary response metric `paid_in_full` as the last column and create the beginnings of a design matrix `bills_data`. Ensure the unit / observation is bill i.e. each row should be ONE bill ONLY!

```
#TO-DO
bills_and_payments_and_discounts = bills_and_payments_and_discounts %>%
  filter((!is.na(transaction_date) & transaction_date <= due_date) | is.na(transaction_date))
bills_and_payments_and_discounts = bills_and_payments_and_discounts %>%
  group_by(id) %>%
  mutate(payment_total = sum(paid_amount))
bills_and_payments_and_discounts = bills_and_payments_and_discounts %>%
  mutate(payment_total = if_else(is.na(payment_total), 0, payment_total))
# if_else is dypler version of ifelse
bills_and_payments_and_discounts = bills_and_payments_and_discounts %>%
  mutate(paid_in_full = as.numeric(payment_total >= tot_amount))

bills_and_payments_and_discounts = bills_and_payments_and_discounts %>%
  group_by(id) %>%
  mutate(payment_at_least_one_month_before_due_date = as.numeric(as.integer(due_date) - as.integer(trans

bills_and_payments_and_discounts = bills_and_payments_and_discounts %>%
  group_by(id) %>%
  mutate(number_of_payments_made_already = sum(payment_at_least_one_month_before_due_date))

bills_and_payments_and_discounts = bills_and_payments_and_discounts %>%
  mutate(number_of_payments_made_already = if_else(is.na(number_of_payments_made_already), 0, number_of_



bills_and_payments_and_discounts = bills_and_payments_and_discounts %>%
  group_by(id) %>%
  slice(1)
table(bills_and_payments_and_discounts$paid_in_full)
```

```
##
##      0      1
## 125598   8282
```

How should you add features from transformations (called "featurization")? What data type(s) should they be? Make some features below if you think of any useful ones. Name the columns appropriately so another data scientist can easily understand what information is in your variables. Make sure missingness (if in a categorical variable) is treated as a legal level of that variable. Make sure the response variable is there too in the final data frame.

```
#TO-DO
bills_and_payments_and_discounts = bills_and_payments_and_discounts %>%
  mutate(due_date_as_integer = as.integer(due_date))
#do customer id's into factor for common customers > 10 otherwise "other"
#convert discount id to fdactor if number of discount id's is > 10. Drop all other columns from discoun

bills_and_payments_and_discounts = bills_and_payments_and_discounts %>%
  select(-id.y, -id, -due_date, -invoice_date, -customer_id, -discount_id, -paid_amount, -transaction_da
```

```
## Adding missing grouping variables: `id`
```

```
bills_and_payments_and_discounts
```

```
## # A tibble: 133,880 x 5
## # Groups:   id [133,880]
##         id tot_amount paid_in_full number_of_payments_made~1 due_date_as_integer
##      <dbl>      <dbl>        <dbl>                     <dbl>               <int>
##  1 5000000     99480.            0                         1               17013
##  2 5693147     99529.            0                         1               17297
##  3 6386294     99479.            0                         1               17165
##  4 6609438     99477.            0                         1               17293
##  5 6791759     99477.            0                         1               17151
##  6 6945910     99477.            0                         1               17325
##  7 7079442     99477.            0                         1               17298
##  8 7302585     99477.            0                         1               17253
##  9 7484907     99475.            0                         1               17252
## 10 7564949     99481.            0                         1               17137
## # i 133,870 more rows
## # i abbreviated name: 1: number_of_payments_made_already
```

Regression Trees

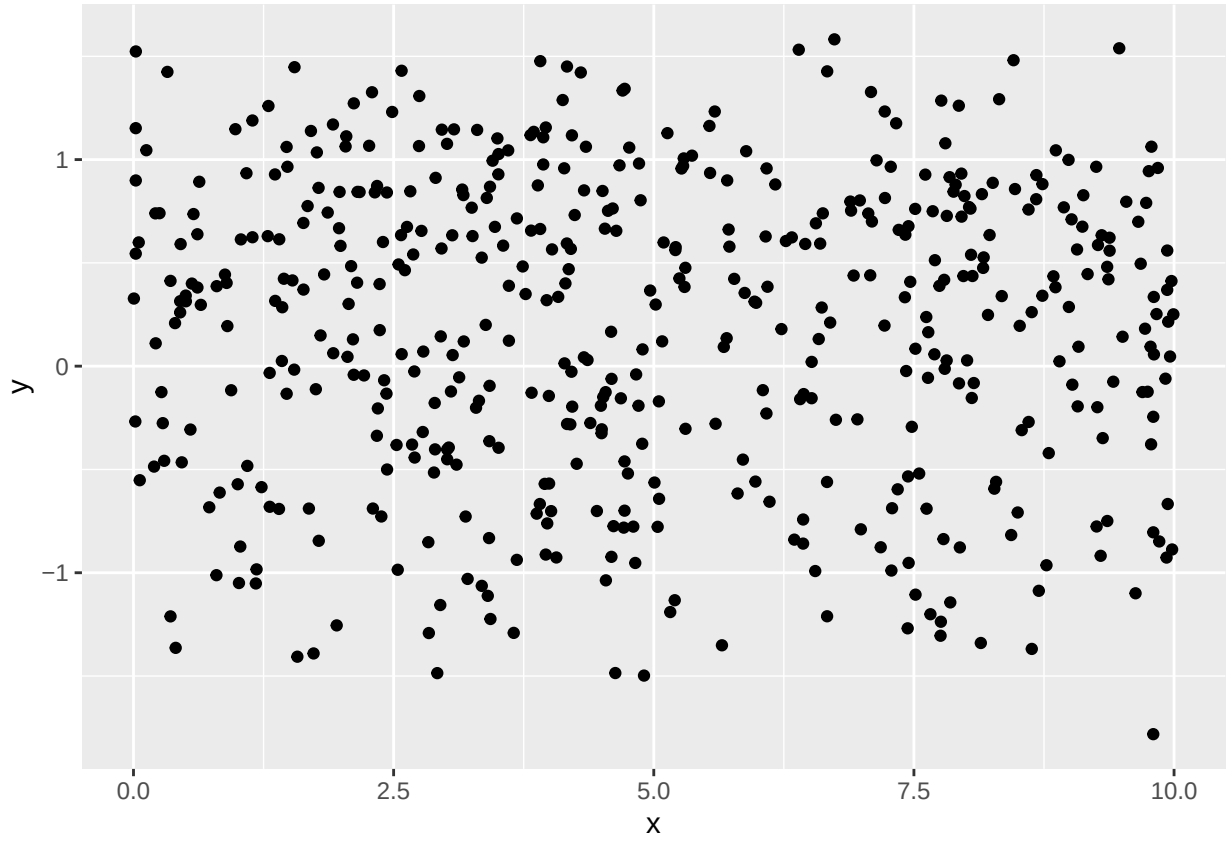
You can use the `YARF` package if it works, otherwise, use the `randomForest` package (the canonical R package for this algorithm).

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

```
rm(list = ls())
n_train = 500
sigma = 0.3
x_min = 0
x_max = 10
x = runif(n_train, x_min, x_max)

f_x = function(x){sin(x)}
x_train = runif(n_train, x_min, x_max)
y_train = f_x(x) + rnorm(n_train, 0, sigma)
```

Plot an example dataset of size 500:

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



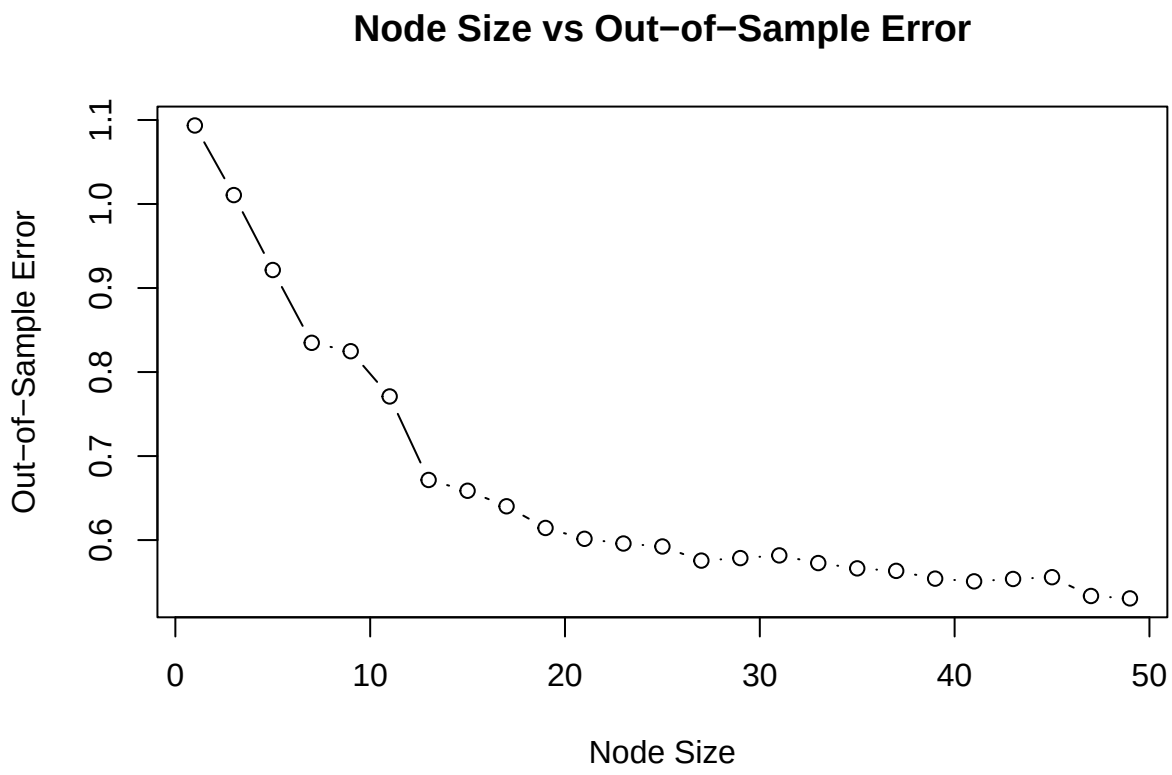
Create a test set of size 500 from this data generating process:

```
#TO-DO
n_test = 500
x_test = runif(n_test, x_min, x_max)
y_test = f_x(x_test) + rnorm(n_test, 0, 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 s_e. Plot.

```
# Load necessary package
pacman::p_load(randomForest)

# Function to calculate out-of-sample error
calculate_oos_error = function(y_true, y_pred) {
  mean((y_true - y_pred)^2)
}

# Define parameters
ntree = 1
replace = FALSE
sampsize = length(y_train)
mtry = 1
nodesizes = seq(1, 50, by = 2) # range of nodesize values to try
```

```r
# Train models with different nodesize values
oos_errors = numeric(length(nodesizes))

for (i in seq_along(nodesizes)) {
  # Create a matrix for x_train and set column name to match test data
  x_train_matrix = matrix(x_train, ncol = 1, dimnames = list(NULL, "x"))

  rf_model = randomForest(x = x_train_matrix, y = y_train, ntree = ntree, replace = replace,
                          sampsize = sampsize, mtry = mtry, nodesize = nodesizes[i])

  # Create a data frame with the test data and explicitly set the column name
  test_data = data.frame(x = x_test)
  names(test_data) <- "x"

  # Predict on test set
  y_pred = predict(rf_model, newdata = test_data)

  # Calculate out-of-sample error
  oos_errors[i] = calculate_oos_error(y_test, y_pred)
}

# Plot nodesize by out of sample s_e
plot(nodesizes, oos_errors, type = "b", xlab = "Node Size", ylab = "Out-of-Sample Error",
     main = "Node Size vs Out-of-Sample Error")
```



Plot the regression tree model g(x) with the optimal node size.

```r
#TO-DO
pacman::p_load(rpart)
optimal_nodesize=49
final_model = rpart(y_train ~ x_train, control=rpart.control(minsplit=optimal_nodesize))

# Plot the decision tree
#rpart.plot(final_model, main=paste("Decision Tree with Optimal Node Size =", optimal_nodesize))
```

Find the oosRMSE of this optimal-node-size model.

```r
#TO-DO
predictions = predict(final_model, newdata = data.frame(x_train = x_test), type = "vector")

# Calculate RMSE
oosRMSE = sqrt(mean((y_test - predictions)^2))

# Print the oosRMSE
print(paste("Out-of-Sample RMSE:", oosRMSE))
```

```
## [1] "Out-of-Sample RMSE: 0.686684135563895"
```

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.

```r
#TO-DO
n_simulations <- 100
predictions <- matrix(nrow = n_simulations, ncol = length(y_test))

generate_data <- function(n = 100, noise_sd = 1) {
  x <- runif(n, 0, 10)  # Uniform distribution of predictor
  y <- 3 + 0.5 * x + rnorm(n, mean = 0, sd = noise_sd)  # Linear relationship with noise
  list(x = x, y = y)
}

# Generate and fit models
for (i in 1:n_simulations) {
  # Generate data
  sim_data <- generate_data()
  x_train_sim <- sim_data$x
  y_train_sim <- sim_data$y

  # Fit model
  model <- rpart(y_train_sim ~ x_train_sim, control=rpart.control(minsplit=optimal_nodesize))

  # Predict on the consistent test set
  predictions[i, ] <- predict(model, newdata = data.frame(x_train_sim = x_test), type = "vector")
}

# Compute Bias
mean_predictions <- rowMeans(predictions)

# Bias squared
```

```r
bias_squared <- mean((mean_predictions - y_test)^2)

# Variance
variance <- mean(apply(predictions, 2, function(p) mean((p - mean_predictions)^2)))

# Mean Squared Error
mse <- mean((predictions - matrix(y_test, nrow = n_simulations, ncol = length(y_test), byrow = TRUE))^2)

# Print results
cat("Bias^2:", bias_squared, "\nVariance:", variance, "\nMSE:", mse, "\n")
```

```
## Bias^2: 29.15056
## Variance: 1.961394
## MSE: 31.03044
```

```r
cat("Check (Bias^2 + Variance):", bias_squared + variance, "\n")
```

```
## Check (Bias^2 + Variance): 31.11196
```

Classification Trees

Let's get the letter recognition data from the `mlbench` package.

```r
rm(list = ls())
pacman::p_load(mlbench)
data(LetterRecognition, package = "mlbench")
n = nrow(LetterRecognition)
skimr::skim(LetterRecognition)
```

Table 9: Data summary

Name	LetterRecognition
Number of rows	20000
Number of columns	17
Column type frequency:	
factor	1
numeric	16
Group variables	None

Variable type: factor

skim_variable	n_missing	complete_rate	ordered	n_unique	top_counts
lettr	0	1	FALSE	26	U: 813, D: 805, P: 803, T: 796

Variable type: numeric

skim_variable	n_missing	complete_rate	mean	sd	p0	p25	p50	p75	p100	hist
x.box	0	1	4.02	1.91	0	3	4	5	15	
y.box	0	1	7.04	3.30	0	5	7	9	15	
width	0	1	5.12	2.01	0	4	5	6	15	
high	0	1	5.37	2.26	0	4	6	7	15	
onpix	0	1	3.51	2.19	0	2	3	5	15	
x.bar	0	1	6.90	2.03	0	6	7	8	15	
y.bar	0	1	7.50	2.33	0	6	7	9	15	
x2bar	0	1	4.63	2.70	0	3	4	6	15	
y2bar	0	1	5.18	2.38	0	4	5	7	15	
xybar	0	1	8.28	2.49	0	7	8	10	15	
x2ybr	0	1	6.45	2.63	0	5	6	8	15	
xy2br	0	1	7.93	2.08	0	7	8	9	15	
x.ege	0	1	3.05	2.33	0	1	3	4	15	
xegvy	0	1	8.34	1.55	0	8	8	9	15	
y.ege	0	1	3.69	2.57	0	2	3	5	15	
yegvx	0	1	7.80	1.62	0	7	8	9	15	

This dataset has 20,000 examples. Create a training-select-test split so that they each have 1,000 observations.

```
train_idx = sample(1 : n, 1000)
select_idx = sample(setdiff(1 : n, train_idx), 1000)
test_idx = sample(setdiff(1 : n, c(train_idx, select_idx)), 1000)
letters_train = LetterRecognition[train_idx, ]
letters_select = LetterRecognition[select_idx, ]
letters_test = LetterRecognition[test_idx, ]
```

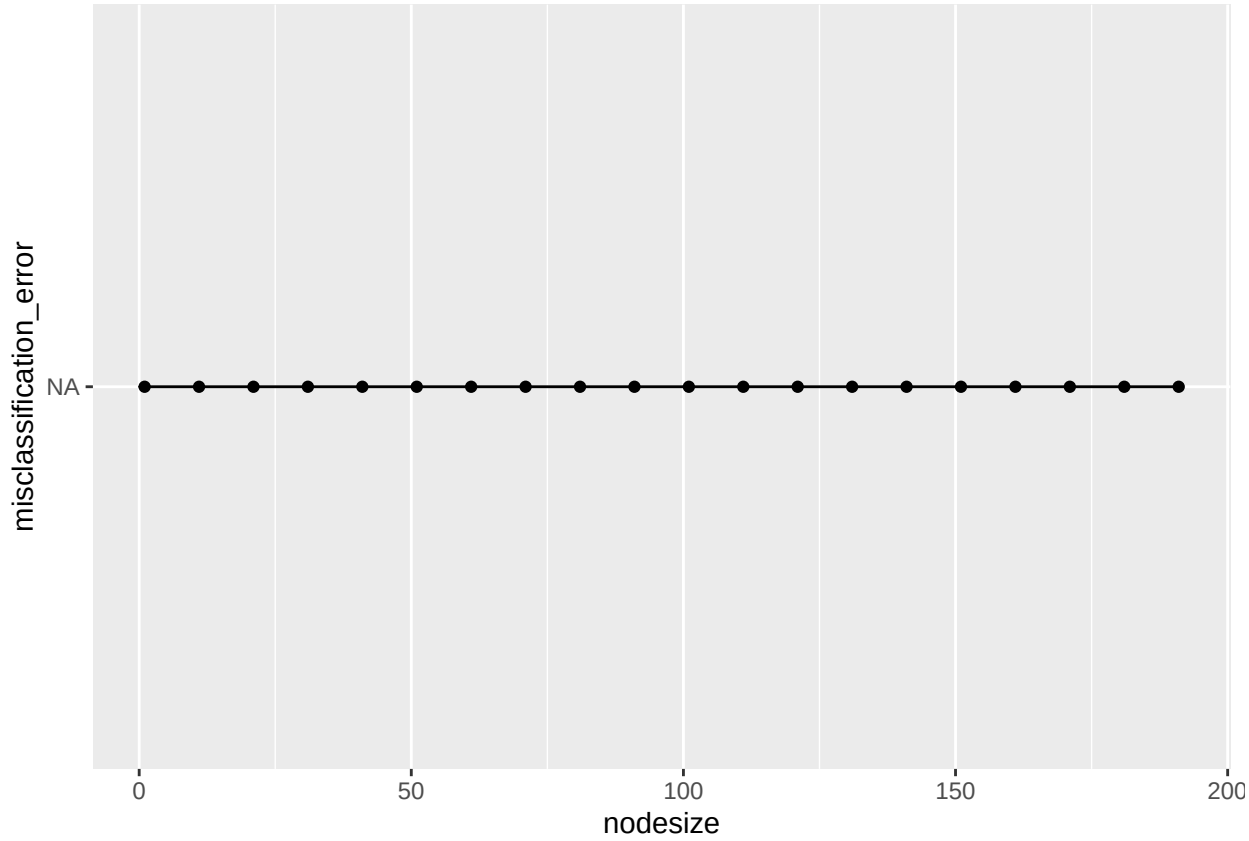
Find the optimal classification tree by using the model selection algorithm to optimize the nodesize hyperparameter. Use misclassification error as the performance metric.

```
nodesizes = seq(1, 200, by = 10)
misclassification_errs = array(NA, length(nodesizes))
#TO-DO
# Find the nodesize with the minimum misclassification error
optimal_nodesize = nodesizes[which.min(misclassification_errs)]
optimal_nodesize
```

```
## numeric(0)
```

Plot the oos misclassification error by nodesize.

```
ggplot(data.frame(nodesize = nodesizes, misclassification_error = misclassification_errs)) +
  aes(x = nodesize, y = misclassification_error) +  # Corrected the y aesthetic
  geom_point() +
  geom_line()
```



Construct the optimal classification tree on train and select sets. Then estimate generalization error. Save `y_hat_test` as we'll need it later.

```
pacman::p_load(rpart)
# Combine train and select datasets if not already combined
#letters_train_select <- rbind(letters_train, letters_select)

# Convert to data frame just to be sure
#letters_train_select <- as.data.frame(letters_train_select)

# Ensure class column is a factor
#letters_train_select$class <- as.factor(letters_train_select$class)

# Build the tree model with the optimal node size
#tree_mod_opt <- rpart(class ~ ., data = letters_train_select, method = "class", control = rpart.contro

# Predict on test data
#y_hat_test <- predict(tree_mod_opt, letters_test, type = "class")

# Calculate generalization error
#generalization_error <- mean(y_hat_test != letters_test$class)
#generalization_error
```

Print out the top of the tree so we can have some level of interpretation to how the model g is predicting.

```
#illustrate_trees(tree_mod_opt, max_depth = 5, length_in_px_per_half_split = 30, open_file = TRUE)
```

Create a "confusion matrix". This means it shows every predicted level (which is a letter in our case) and every actual level. Here you'll see every type of error e.g. "P was predicted but the real letter is H", "M was predicted but the real letter is N" etc. This is really easy: one call to the `table` function is all you need.

```
#TO-DO
```

Which errors are most prominent in this model?

#Tree models theoretically drag bias down to a point where it doesnt affect anything. Variance may be high with the different trees using different predictors. Another error that it can incur is ignorance