# HW6

```
library(readr)
library(ggplot2)
library(dplyr)
```

```
##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
library(tidymodels)
```

```
## -- Attaching packages --------------------------------------- tidymodels 0.2.0 --

## v broom        0.7.12     v rsample      0.1.1
## v dials        0.1.0      v tibble       3.1.6
## v infer        1.0.0      v tidyr        1.2.0
## v modeldata    0.1.1      v tune         0.2.0
## v parsnip      0.2.1      v workflows    0.2.6
## v purrr        0.3.4      v workflowsets 0.2.1
## v recipes      0.2.0      v yardstick    0.0.9

## -- Conflicts ------------------------------------------ tidymodels_conflicts() --
## x purrr::discard()  masks scales::discard()
## x dplyr::filter()   masks stats::filter()
## x dplyr::lag()      masks stats::lag()
## x yardstick::spec() masks readr::spec()
## x recipes::step()   masks stats::step()
## * Use tidymodels_prefer() to resolve common conflicts.
```

```
library(ISLR)
library(tidyverse)
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.1 --

## v stringr 1.4.0     v forcats 0.5.1
```

```
## -- Conflicts ---------------------------------------- tidyverse_conflicts() --
## x scales::col_factor() masks readr::col_factor()
## x purrr::discard()     masks scales::discard()
## x dplyr::filter()      masks stats::filter()
## x stringr::fixed()     masks recipes::fixed()
## x dplyr::lag()         masks stats::lag()
## x yardstick::spec()    masks readr::spec()
```

```
library(glmnet)
```

```
## Loading required package: Matrix
```

```
##
## Attaching package: 'Matrix'
```

```
## The following objects are masked from 'package:tidyr':
##
##     expand, pack, unpack
```

```
## Loaded glmnet 4.1-4
```

```
tidymodels_prefer()
library(rpart.plot)
```

```
## Loading required package: rpart
```

```
##
## Attaching package: 'rpart'
```

```
## The following object is masked from 'package:dials':
##
##     prune
```

```
library(vip)
library(janitor)
library(randomForest)
```

```
## randomForest 4.7-1.1
```

```
## Type rfNews() to see new features/changes/bug fixes.
```

```
library(xgboost)
library(ranger)
```

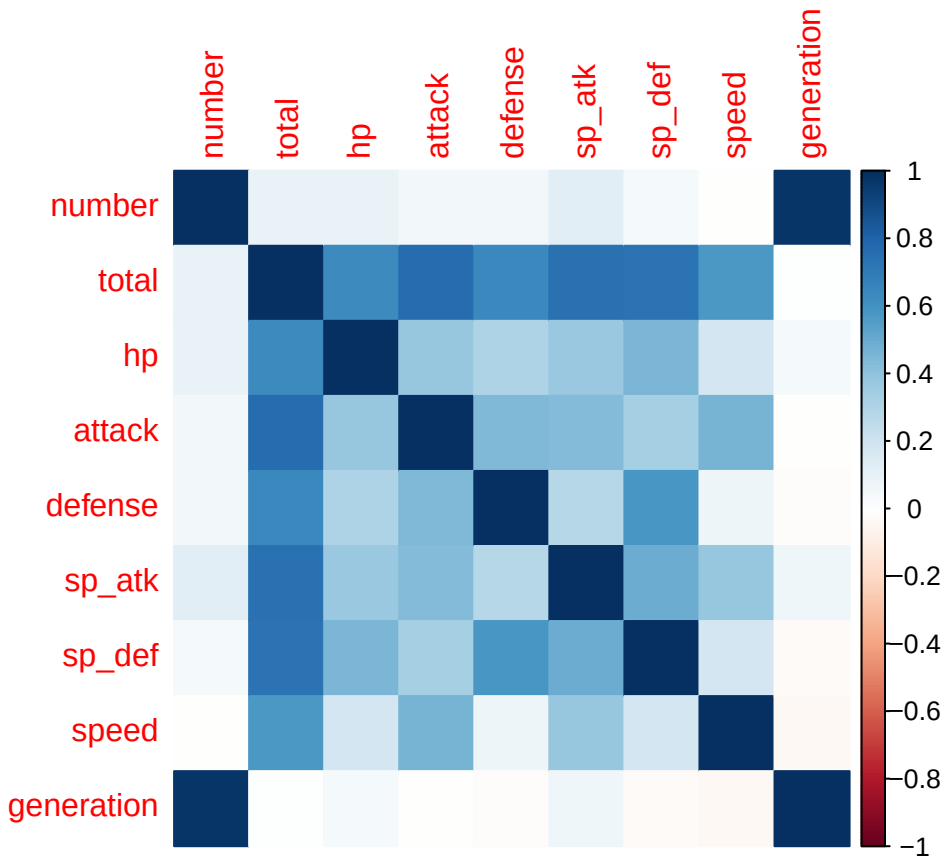
# Question 1

# Question 2

```
library(corrplot)
```

```
## corrplot 0.92 loaded
```

```
Pokemon_train %>%
  select(where(is.numeric)) %>%
  cor() %>%
  corrplot(method = 'color')
```



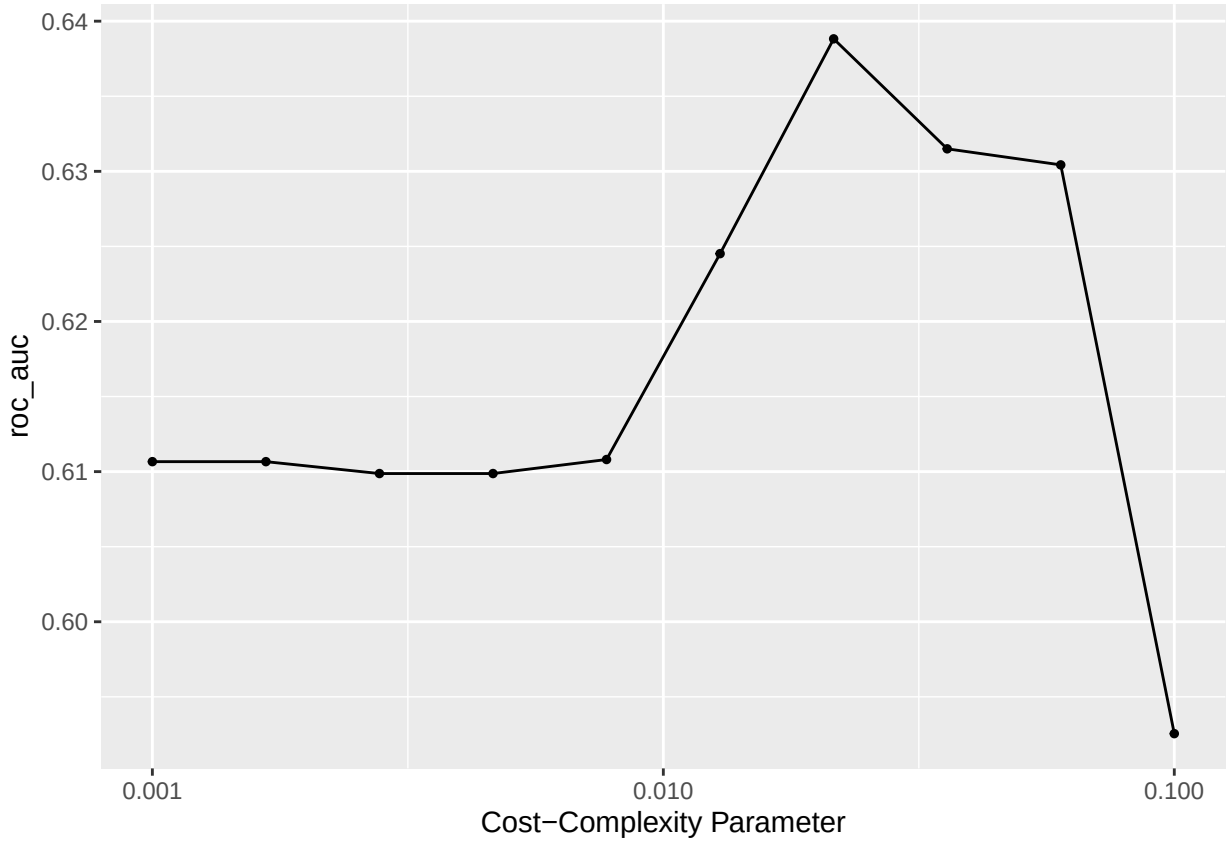

In this plot, I have included all the continuous variables in order to see and study dependences or associations between the variables. This will help determine what makes the primary type of a specific Pokemon.

Looking at the relationships, there seems to be a strong positive relationship between total and all of the battle stats understandably, as the total sums up all the stats. However, it can also be noted that defense & sp_def, as well as attack & sp_attack are also relatively correlated. This again also makes sense, as both groups hold variables performing the same functions, but just towards a specific target.

## Question 3

```
tree_spec <- decision_tree() %>%
  set_engine("rpart")
```

```r
class_tree_spec <- tree_spec %>%
  set_mode("classification")

class_tree_fit <- class_tree_spec %>%
  fit(type_1 ~ legendary + generation + sp_atk + attack + speed + defense + hp + sp_def, data=Pokemon_t

class_tree_wf <- workflow() %>%
  add_model(class_tree_spec %>% set_args(cost_complexity = tune())) %>%
  add_formula(type_1 ~ legendary + generation + sp_atk + attack + speed + defense + hp + sp_def)

poke_grid <- grid_regular(cost_complexity(range = c(-3, -1)), levels = 10)

tune_res <- tune_grid(
  class_tree_wf,
  resamples = pokemon_folds,
  grid = poke_grid,
  metrics = metric_set(roc_auc)
)

autoplot(tune_res)
```



## Question 4

```
arrange(collect_metrics(tune_res))
```

```
## # A tibble: 10 x 7
##    cost_complexity .metric .estimator  mean     n std_err .config
##              <dbl> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
##  1         0.001   roc_auc hand_till  0.611     5  0.0380 Preprocessor1_Model01
##  2         0.00167 roc_auc hand_till  0.611     5  0.0380 Preprocessor1_Model02
##  3         0.00278 roc_auc hand_till  0.610     5  0.0373 Preprocessor1_Model03
##  4         0.00464 roc_auc hand_till  0.610     5  0.0373 Preprocessor1_Model04
##  5         0.00774 roc_auc hand_till  0.611     5  0.0393 Preprocessor1_Model05
##  6         0.0129  roc_auc hand_till  0.625     5  0.0332 Preprocessor1_Model06
##  7         0.0215  roc_auc hand_till  0.639     5  0.0177 Preprocessor1_Model07
##  8         0.0359  roc_auc hand_till  0.631     5  0.0184 Preprocessor1_Model08
##  9         0.0599  roc_auc hand_till  0.630     5  0.0163 Preprocessor1_Model09
## 10         0.1     roc_auc hand_till  0.593     5  0.0241 Preprocessor1_Model10
```

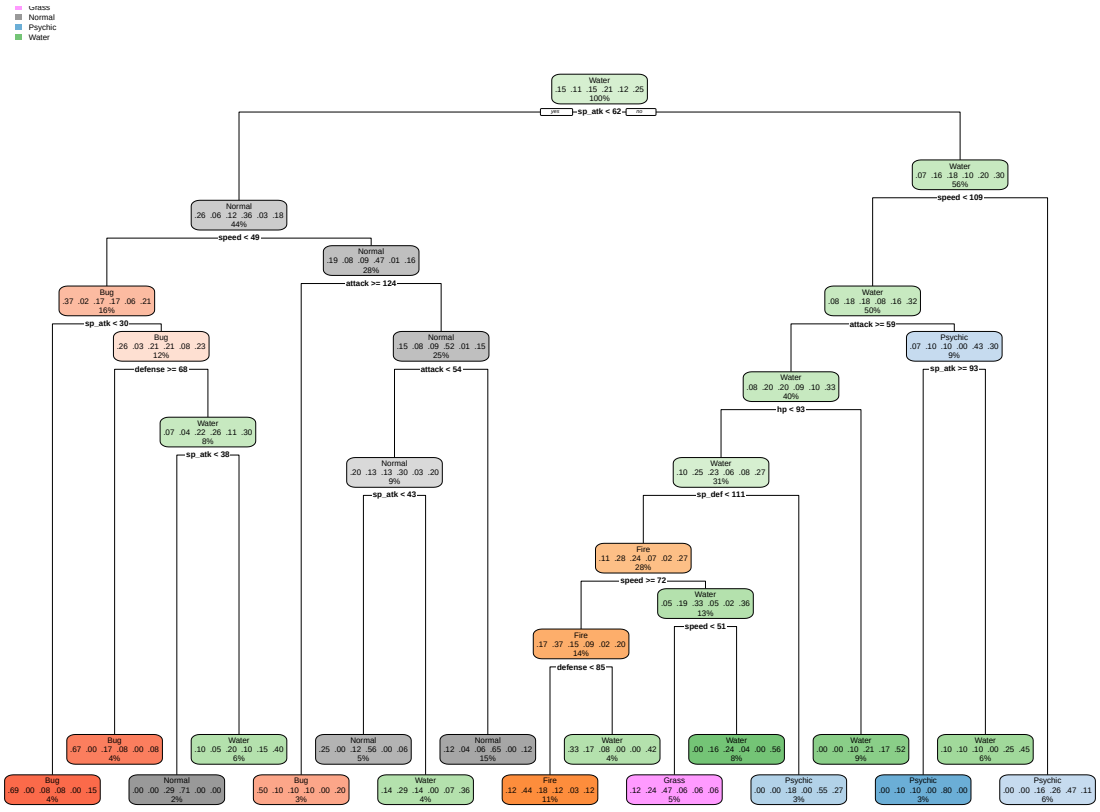
```
best <- select_best(tune_res, metric = "roc_auc")
best
```

```
## # A tibble: 1 x 2
##   cost_complexity .config
##             <dbl> <chr>
## 1          0.0215 Preprocessor1_Model07
```

The roc_auc of the best-performing pruned decision tree on the folds is 0.6388230.

## Question 5

```
tree_spec <- decision_tree() %>%
  set_engine("rpart")

class_tree_fit %>%
  extract_fit_engine() %>%
  rpart.plot(roundint=FALSE)
```




```
rf_spec <- rand_forest(mtry = tune(), trees = tune(), min_n = tune()) %>%
  set_engine("ranger", importance = "impurity") %>%
  set_mode("classification")
```

mtry = An integer for # of predictors that will be randomly sampled at each split when creating the tree models trees = An integer for # of trees contained in the ensemble min_n = An integer for the minimum # of data points in a node that are required for the node to be split further

```
rf_spec2 <- rand_forest(mtry = tune(), trees = tune(), min_n = tune()) %>%
  set_engine("ranger", importance = "impurity") %>%
  set_mode("classification")

class_tree_wf2 <- workflow() %>%
  add_recipe(pokemon_recipe) %>%
  add_model(rf_spec2)

grid2 = grid_regular(mtry(range=c(1,8)), trees(range=c(1,8)), min_n(range=c(1,8)),levels = 8)
```

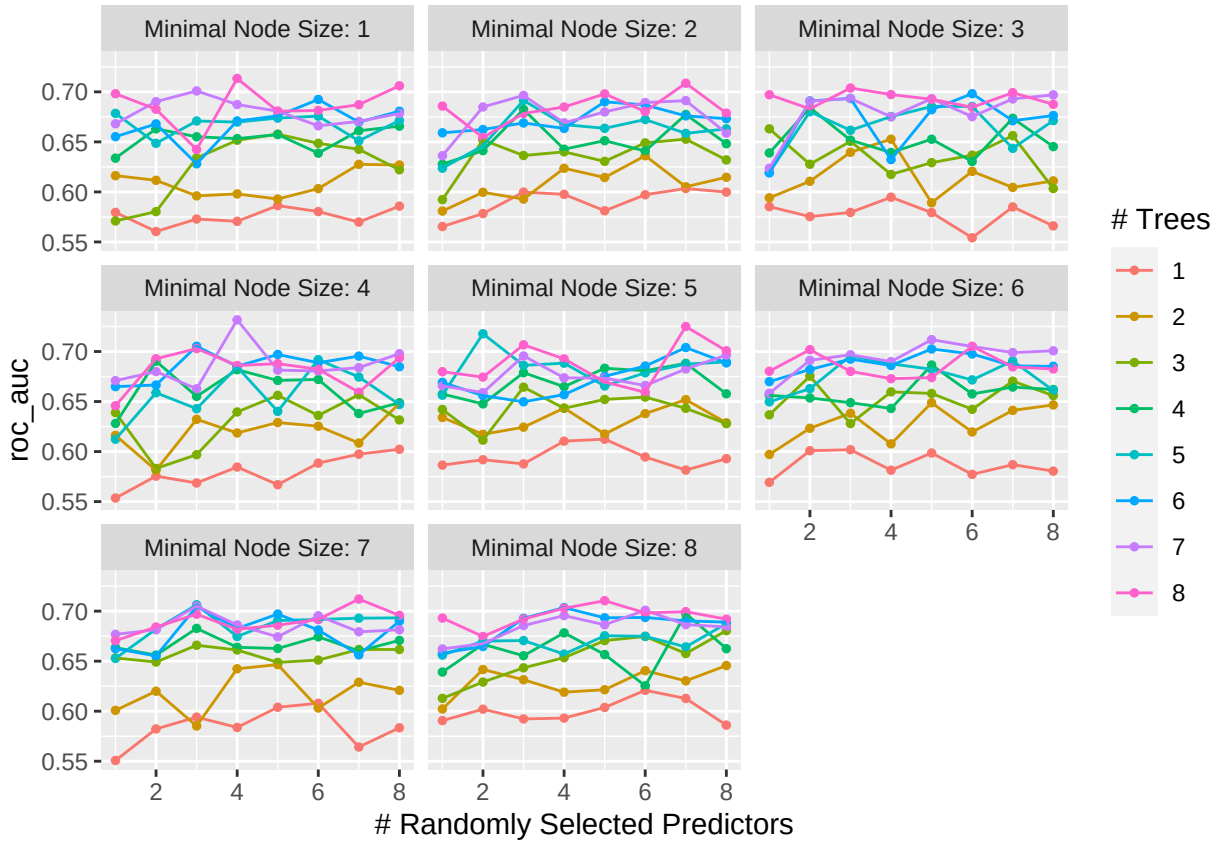
Mtry should not be smaller than 1 or greater than 8 because the RF is based on a subset of the total number of predictors p (p/3). If you set mtry = 8, then you may see a huge variation in RMSEP between the different RF.

## Question 6

```
tune_res2 <- tune_grid(
  class_tree_wf2,
  resamples = pokemon_folds,
  grid = grid2,
  metrics = metric_set(roc_auc)
)
```

```
## ! Fold1: preprocessor 1/1: The following variables are not factor vectors and wil...

## ! Fold2: preprocessor 1/1: The following variables are not factor vectors and wil...

## ! Fold3: preprocessor 1/1: The following variables are not factor vectors and wil...

## ! Fold4: preprocessor 1/1: The following variables are not factor vectors and wil...

## ! Fold5: preprocessor 1/1: The following variables are not factor vectors and wil...
```

```
autoplot(tune_res2)
```




It appears as if the higher number of trees, the better performing the roc_auc. In all of the minimal node sizes, the higher number of trees continue to yield the best performance.

## Question 7

```
arrange(collect_metrics(tune_res2))
```

```
## # A tibble: 512 x 9
##     mtry trees min_n .metric .estimator  mean     n std_err .config
##    <int> <int> <int> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
##  1     1     1     1 roc_auc hand_till  0.580     5  0.0138 Preprocessor1_Model~
##  2     2     1     1 roc_auc hand_till  0.560     5  0.0126 Preprocessor1_Model~
##  3     3     1     1 roc_auc hand_till  0.573     5  0.0158 Preprocessor1_Model~
##  4     4     1     1 roc_auc hand_till  0.571     5  0.0148 Preprocessor1_Model~
##  5     5     1     1 roc_auc hand_till  0.586     5  0.0253 Preprocessor1_Model~
##  6     6     1     1 roc_auc hand_till  0.580     5  0.0139 Preprocessor1_Model~
##  7     7     1     1 roc_auc hand_till  0.570     5  0.0151 Preprocessor1_Model~
##  8     8     1     1 roc_auc hand_till  0.586     5  0.0159 Preprocessor1_Model~
##  9     1     2     1 roc_auc hand_till  0.616     5  0.0211 Preprocessor1_Model~
## 10     2     2     1 roc_auc hand_till  0.612     5  0.0168 Preprocessor1_Model~
## # ... with 502 more rows
```

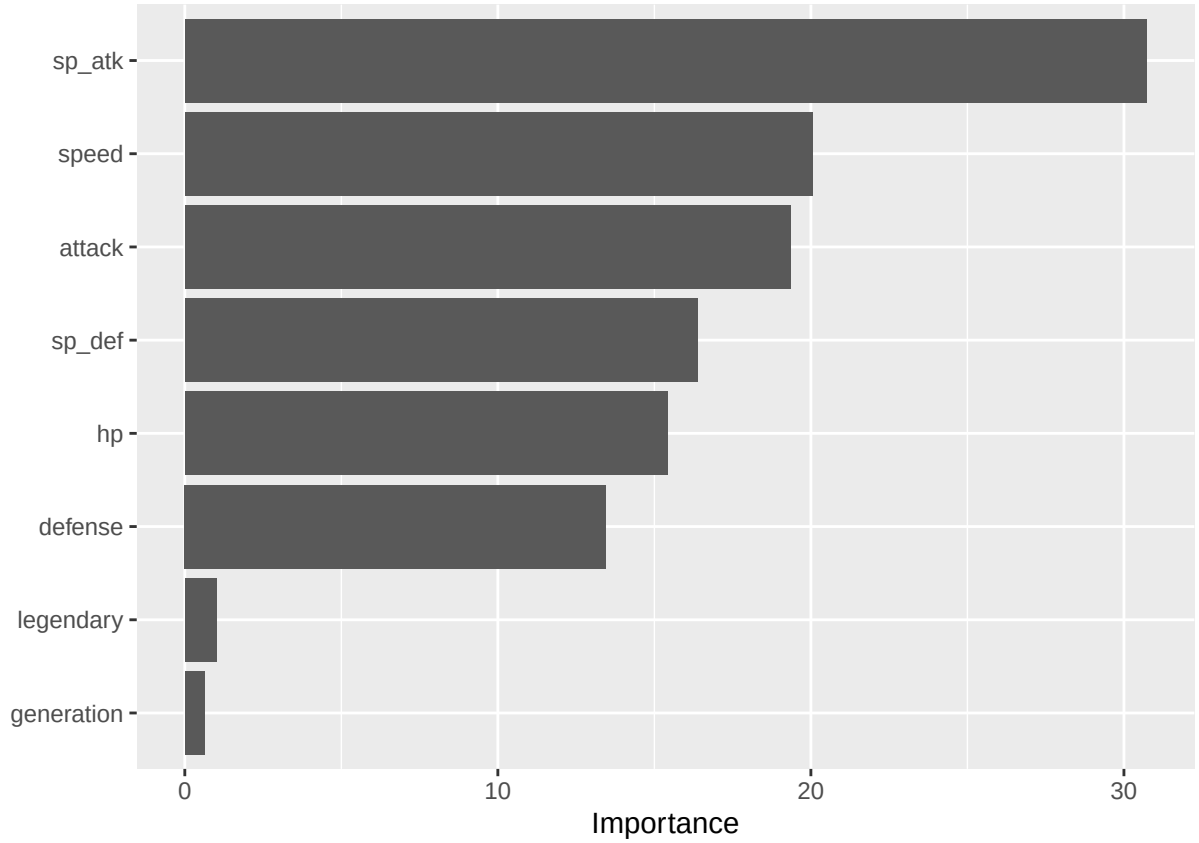
```
best2 <- select_best(tune_res2, metric = "roc_auc")
best2
```

```
## # A tibble: 1 x 4
##    mtry trees min_n .config
##   <int> <int> <int> <chr>
## 1     4     7     4 Preprocessor1_Model244
```

The roc_auc of my best performing random forest model is that of mtry =4, trees = 7, and min_n = 4; 0.7316887

## Question 8

```
vip(class_tree_fit)
```




## Question 9

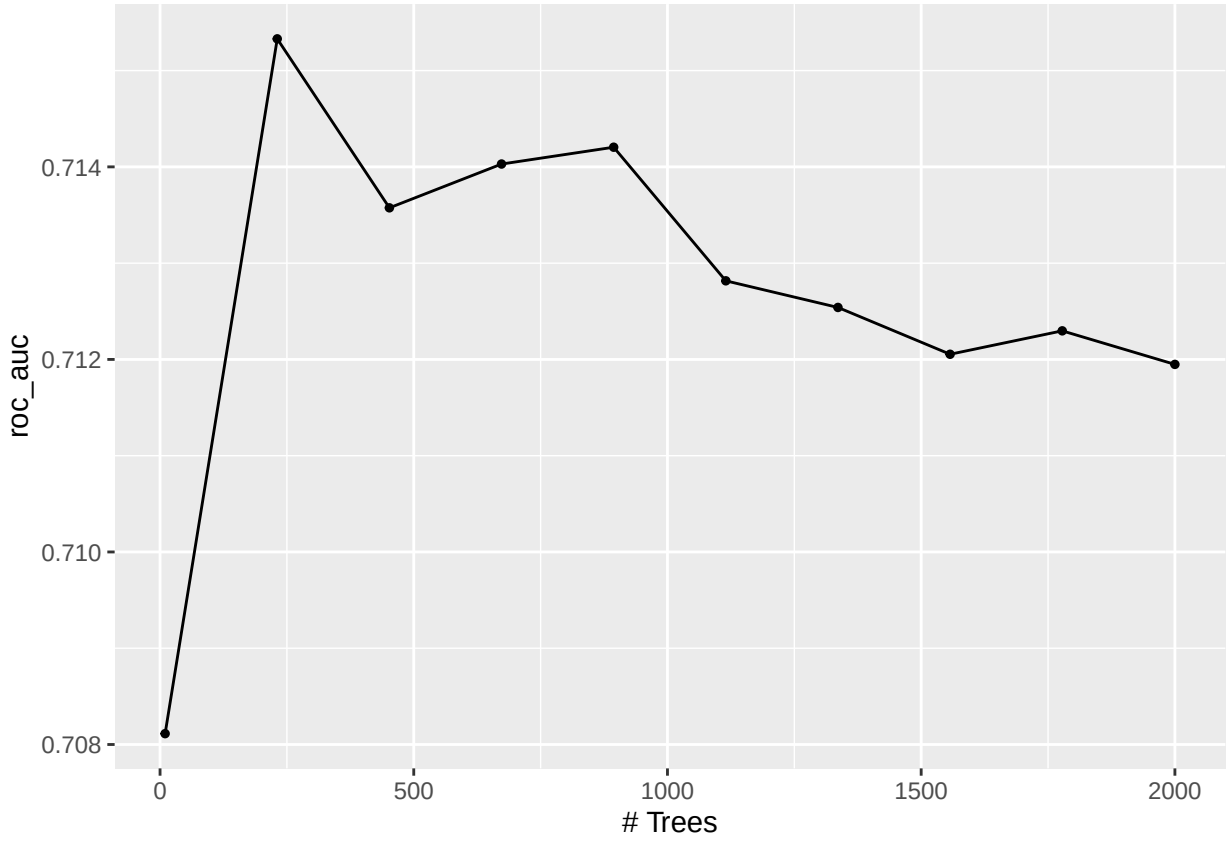
```
boost_spec <- boost_tree(trees= tune(), tree_depth = 4) %>%
  set_engine("xgboost") %>%
  set_mode("classification")

grid3 = grid_regular(trees(range=c(10,2000)), levels = 10)

class_tree_wf3 <- workflow() %>%
  add_recipe(pokemon_recipe) %>%
  add_model(boost_spec)

tune_res3 <- tune_grid(
  class_tree_wf3,
  resamples = pokemon_folds,
  grid = grid3,
  metrics = metric_set(roc_auc)
)
```

```
## ! Fold1: preprocessor 1/1: The following variables are not factor vectors and wil...
```

```
## ! Fold2: preprocessor 1/1: The following variables are not factor vectors and wil...
```

```
## ! Fold3: preprocessor 1/1: The following variables are not factor vectors and wil...

## ! Fold4: preprocessor 1/1: The following variables are not factor vectors and wil...

## ! Fold5: preprocessor 1/1: The following variables are not factor vectors and wil...
```

```
autoplot(tune_res3)
```




```
arrange(collect_metrics(tune_res3))
```

```
## # A tibble: 10 x 7
##    trees .metric .estimator  mean     n std_err .config
##    <int> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
##  1    10 roc_auc hand_till  0.708     5  0.0243 Preprocessor1_Model01
##  2   231 roc_auc hand_till  0.715     5  0.0231 Preprocessor1_Model02
##  3   452 roc_auc hand_till  0.714     5  0.0228 Preprocessor1_Model03
##  4   673 roc_auc hand_till  0.714     5  0.0233 Preprocessor1_Model04
##  5   894 roc_auc hand_till  0.714     5  0.0231 Preprocessor1_Model05
##  6  1115 roc_auc hand_till  0.713     5  0.0226 Preprocessor1_Model06
##  7  1336 roc_auc hand_till  0.713     5  0.0229 Preprocessor1_Model07
##  8  1557 roc_auc hand_till  0.712     5  0.0227 Preprocessor1_Model08
##  9  1778 roc_auc hand_till  0.712     5  0.0227 Preprocessor1_Model09
## 10  2000 roc_auc hand_till  0.712     5  0.0228 Preprocessor1_Model10
```

```
best3 <- select_best(tune_res3, metric = "roc_auc")
best3
```

```
## # A tibble: 1 x 2
##   trees .config
##   <int> <chr>
## 1   231 Preprocessor1_Model02
```

The roc_auc of the best-performing boosted tree model on the folds is that of model 2.

## Question 10

```
models <- c(best,best2,best3)
table(models)
```

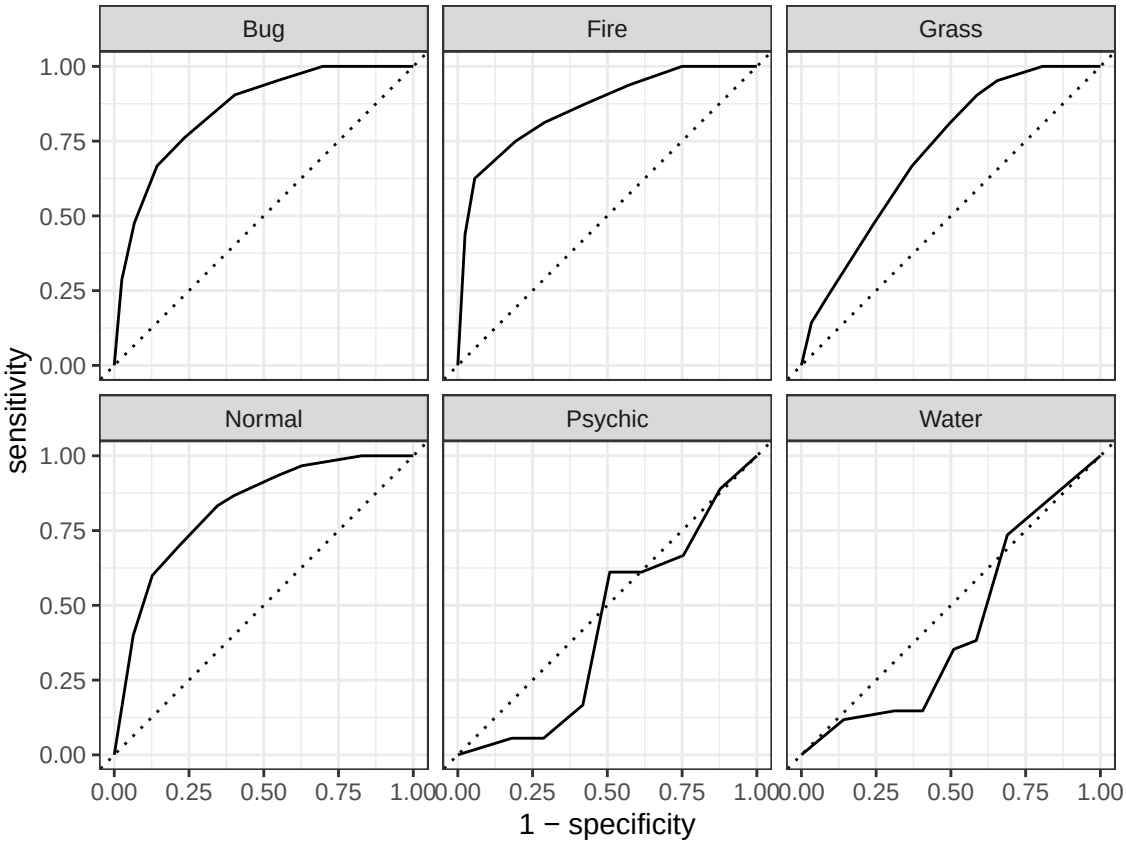
```
## , , mtry = 4, trees = 7, min_n = 4, .config = Preprocessor1_Model244, trees = 231, .config = Preproc
##
##                      .config
## cost_complexity      Preprocessor1_Model07
##   0.0215443469003188                     1
```

```
bestmodel <- select_best(tune_res, tune_res2, tune_res3, metric = "roc_auc")
poke_tree_final <- finalize_workflow(class_tree_wf, bestmodel)
class_tree_final_fit <- fit(poke_tree_final, data = Pokemon_test)
class_tree_final_fit
```

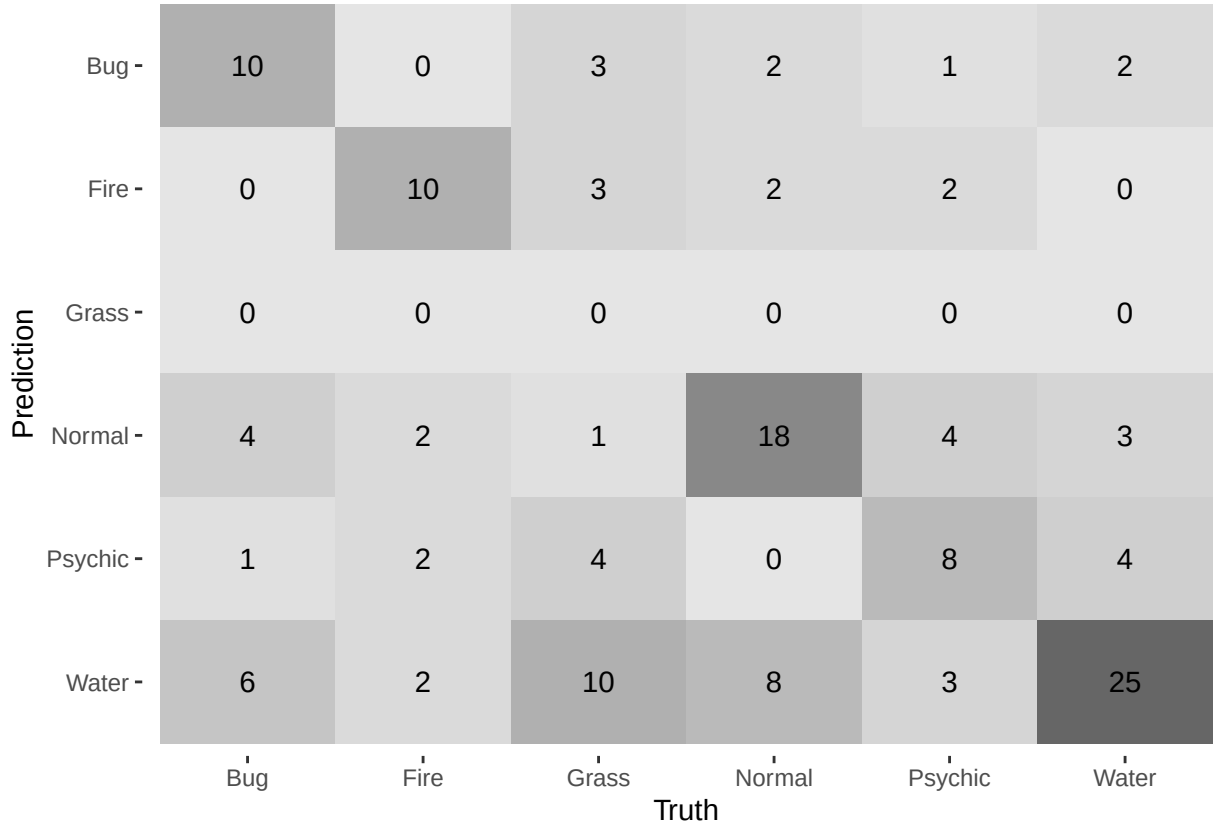
```
## == Workflow [trained] ==========================================================
## Preprocessor: Formula
## Model: decision_tree()
##
## -- Preprocessor ----------------------------------------------------------------
## type_1 ~ legendary + generation + sp_atk + attack + speed + defense +
##     hp + sp_def
##
## -- Model -----------------------------------------------------------------------
## n= 140
##
## node), split, n, loss, yval, (yprob)
##       * denotes terminal node
##
##  1) root 140 106 Water (0.15 0.11 0.15 0.21 0.13 0.24)
##    2) hp< 75.5 93  67 Water (0.22 0.075 0.18 0.14 0.11 0.28)
##      4) sp_atk< 55.5 35  23 Bug (0.34 0.029 0.086 0.29 0 0.26)
##        8) speed< 42.5 9   3 Bug (0.67 0 0.11 0.11 0 0.11) *
##        9) speed>=42.5 26  17 Normal (0.23 0.038 0.077 0.35 0 0.31)
##         18) attack>=52.5 13   7 Normal (0.31 0.077 0 0.46 0 0.15) *
##         19) attack< 52.5 13   7 Water (0.15 0 0.15 0.23 0 0.46) *
##      5) sp_atk>=55.5 58  41 Water (0.14 0.1 0.24 0.052 0.17 0.29)
##       10) attack>=50.5 39  26 Water (0.18 0.1 0.26 0.077 0.051 0.33)
```

```
##         20) generation< 1.5 7   4 Fire (0 0.43 0.43 0.14 0 0) *
##         21) generation>=1.5 32  19 Water (0.22 0.031 0.22 0.062 0.062 0.41)
##           42) sp_def>=78 9   5 Bug (0.44 0 0.22 0.11 0.11 0.11) *
##           43) sp_def< 78 23  11 Water (0.13 0.043 0.22 0.043 0.043 0.52) *
##       11) attack< 50.5 19  11 Psychic (0.053 0.11 0.21 0 0.42 0.21) *
##    3) hp>=75.5 47  30 Normal (0.021 0.19 0.085 0.36 0.17 0.17)
##      6) sp_atk>=78 28  20 Fire (0.036 0.29 0.11 0.18 0.14 0.25)
##       12) attack>=102 10   3 Fire (0 0.7 0 0.1 0.2 0) *
##       13) attack< 102 18  11 Water (0.056 0.056 0.17 0.22 0.11 0.39) *
##      7) sp_atk< 78 19   7 Normal (0 0.053 0.053 0.63 0.21 0.053) *
```

```
augment(class_tree_final_fit, new_data = Pokemon_test) %>%
  roc_curve(truth = type_1, estimate = c(.pred_Bug, .pred_Fire, .pred_Grass, .pred_Normal, .pred_Water,
  autoplot()
```




```
augment(class_tree_final_fit, new_data = Pokemon_test) %>%
  conf_mat(truth = type_1, estimate = .pred_class) %>%
  autoplot(type = "heatmap")
```




The best performing model is the pruned tree model. The water and normal classes had the most accurate prediction results, while grass performed the worst.