131-Homework6

Caleb Mazariegos

2022-05-23

Exercise 1

```
#  Read in the data and set things up as in Homework 5:

# Use clean_names()
pokemon_codebook <- read.csv("/Users/calebmazariegos/Desktop/homework-5/Pokemon.csv")
pokemon_codebook <- clean_names(pokemon_codebook)
head(pokemon_codebook)
```

```
##   x                  name type_1 type_2 total hp attack defense sp_atk sp_def
## 1 1             Bulbasaur  Grass Poison   318 45     49      49     65     65
## 2 2               Ivysaur  Grass Poison   405 60     62      63     80     80
## 3 3              Venusaur  Grass Poison   525 80     82      83    100    100
## 4 3 VenusaurMega Venusaur  Grass Poison   625 80    100     123    122    120
## 5 4            Charmander   Fire          309 39     52      43     60     50
## 6 5            Charmeleon   Fire          405 58     64      58     80     65
##   speed generation legendary
## 1    45          1     False
## 2    60          1     False
## 3    80          1     False
## 4    80          1     False
## 5    65          1     False
## 6    80          1     False
```

```
# Filter out the rarer Pokémon types
pokemon_filter <- pokemon_codebook %>%
  filter((type_1 == "Bug" | type_1 == "Fire" | type_1 == "Grass" | type_1 == "Normal" | type_1 == "Wate

head(pokemon_filter)
```

```
##   x                  name type_1 type_2 total hp attack defense sp_atk sp_def
## 1 1             Bulbasaur  Grass Poison   318 45     49      49     65     65
## 2 2               Ivysaur  Grass Poison   405 60     62      63     80     80
## 3 3              Venusaur  Grass Poison   525 80     82      83    100    100
## 4 3 VenusaurMega Venusaur  Grass Poison   625 80    100     123    122    120
## 5 4            Charmander   Fire          309 39     52      43     60     50
## 6 5            Charmeleon   Fire          405 58     64      58     80     65
##   speed generation legendary
## 1    45          1     False
```

```
## 2    60          1     False
## 3    80          1     False
## 4    80          1     False
## 5    65          1     False
## 6    80          1     False
```

```
# converting type 1 and legendary in factors
pokemon_codebook$type_1 <- as.factor(pokemon_codebook$type_1)
pokemon_codebook$legendary <- as.factor(pokemon_codebook$legendary)
pokemon_codebook$generation <- as.factor(pokemon_codebook$generation)

pokemon_filter$type_1 <- as.factor(pokemon_filter$type_1)
pokemon_filter$legendary <- as.factor(pokemon_filter$legendary)
pokemon_filter$generation <- as.factor(pokemon_filter$generation)
```

```
# Do an initial split of the data; you can choose the percentage for splitting. Stratify on the outcome
set.seed(3465)

pokemon_split <- initial_split(pokemon_filter, prop = 0.70, stata = type_1)

pokemon_train <- training(pokemon_split)

pokemon_test <- testing(pokemon_split)
```

```
# Fold the training set using v-fold cross-validation, with v = 5. Stratify on the outcome variable.

set.seed(234)
pokemon_folds <- vfold_cv(pokemon_train, v=5)
pokemon_folds
```

```
## #  5-fold cross-validation
## # A tibble: 5 x 2
##   splits           id
##   <list>           <chr>
## 1 <split [256/64]> Fold1
## 2 <split [256/64]> Fold2
## 3 <split [256/64]> Fold3
## 4 <split [256/64]> Fold4
## 5 <split [256/64]> Fold5
```

```
# Set up a recipe to predict type_1 with legendary, generation, sp_atk, attack, speed, defense, hp, and
# dummy code legendary and generation; Center and scale all predictors.

pokemon_recipe <- recipe(type_1 ~ legendary + generation + sp_atk + attack + speed + defense + hp + sp_
  step_dummy(legendary) %>%
  step_dummy(generation) %>%
  step_center(all_predictors()) %>%
  step_scale(all_predictors())
```

Exercise 2

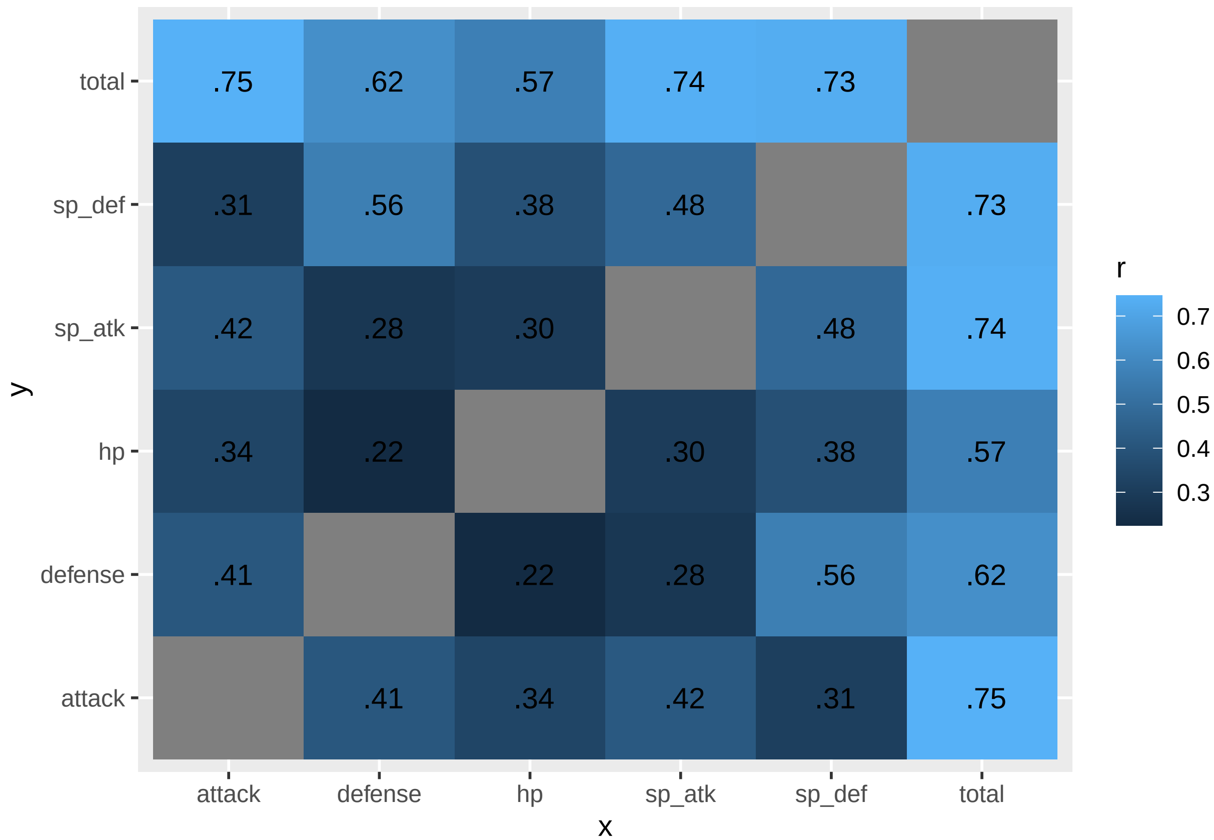
Create a correlation matrix of the training set, using the corrplot package. Note: You can choose how to handle the continuous variables for this plot; justify your decision(s).

What relationships, if any, do you notice? Do these relationships make sense to you?

```
cor_pokemon_train <- pokemon_train %>%
  select(total, hp, attack, defense, sp_atk, sp_def) %>%
  correlate()
```

```
##
## Correlation method: 'pearson'
## Missing treated using: 'pairwise.complete.obs'
```

```
cor_pokemon_train %>%
  stretch() %>%
  ggplot(aes(x,y, fill = r)) +
  geom_tile() +
  geom_text(aes(label = as.character(fashion(r))))
```



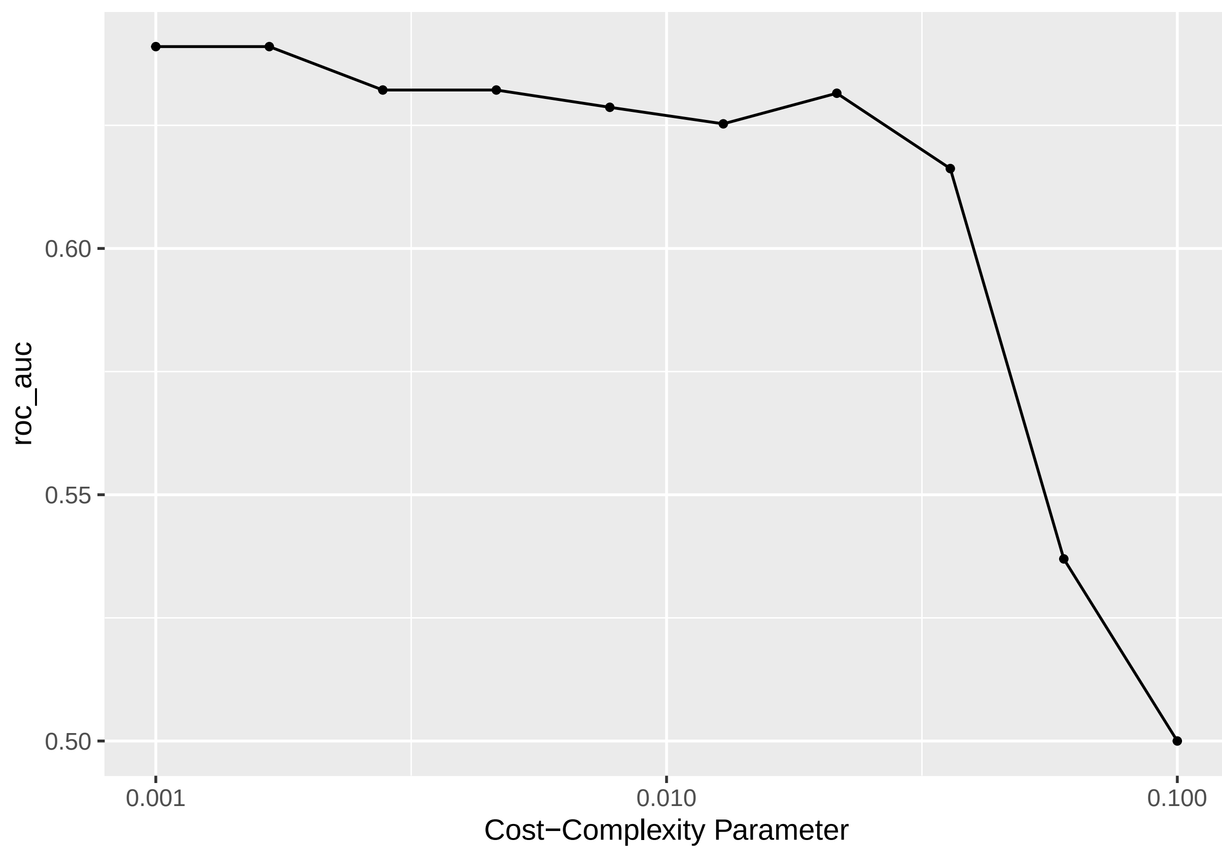
I decided to not include speed, because even though it is a numerical value, I do not consider it to be continuous. I see strong positive relationships between total and attack, total and sp_atk, total and sp_def. I see weak relationships between defense and sp_atk, and hp and defense. The rest of the relationships are moderate.

Exercise 3

First, set up a decision tree model and workflow. Tune the cost_complexity hyperparameter. Use the same levels we used in Lab 7 – that is, range = c(-3, -1). Specify that the metric we want to optimize is roc_auc.

Print an autoplot() of the results. What do you observe? Does a single decision tree perform better with a smaller or larger complexity penalty?

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

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

class_tree_wf <- workflow() %>%
  add_model(class_tree_spec %>% set_args(cost_complexity = tune())) %>%
  add_recipe(pokemon_recipe)

set.seed(5050)

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

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

autoplot(tune_res)
```



The roc_auc seems to be decreasing slightly, then it decreases sharply as the cost-complexity parameter increases.

Exercise 4

What is the `roc_auc` of your best-performing pruned decision tree on the folds? *Hint: Use* `collect_metrics()` *and* `arrange()`*.*

```
tree_roc_auc <- collect_metrics(tune_res) %>%
  arrange(-mean)

head(tree_roc_auc)
```

```
## # A tibble: 6 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  0.641     5  0.0193 Preprocessor1_Model01
## 2         0.00167 roc_auc hand_till  0.641     5  0.0193 Preprocessor1_Model02
## 3         0.00278 roc_auc hand_till  0.632     5  0.0221 Preprocessor1_Model03
## 4         0.00464 roc_auc hand_till  0.632     5  0.0221 Preprocessor1_Model04
## 5         0.0215  roc_auc hand_till  0.632     5  0.0226 Preprocessor1_Model07
## 6         0.00774 roc_auc hand_till  0.629     5  0.0168 Preprocessor1_Model05
```

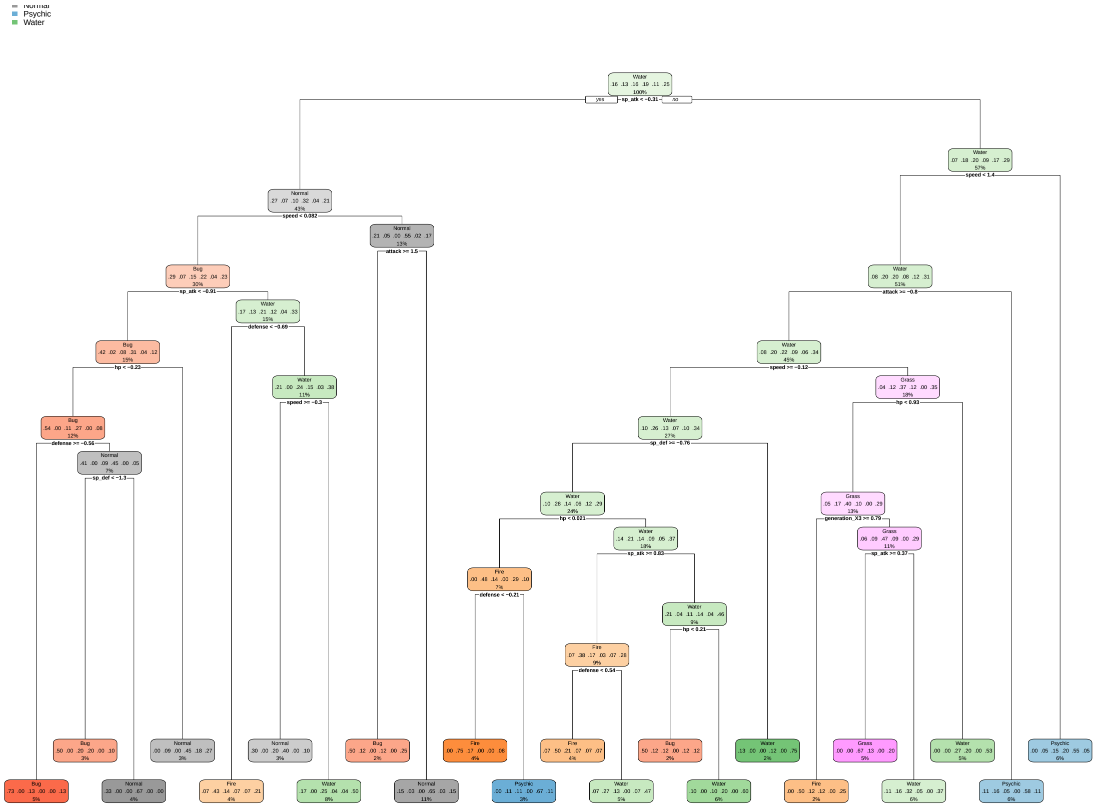
The best roc_auc value is 0.633.

Exercise 5

Using `rpart.plot`, fit and visualize your best-performing pruned decision tree with the *training* set.

```
best_complexity <- select_best(tune_res)
class_tree_final <- finalize_workflow(class_tree_wf, best_complexity)
class_tree_final_fit <- fit(class_tree_final, data = pokemon_train)
class_tree_final_fit %>%
  extract_fit_engine() %>%
  rpart.plot()
```

```
## Warning: Cannot retrieve the data used to build the model (so cannot determine roundint and is.binary
## To silence this warning:
##     Call rpart.plot with roundint=FALSE,
##     or rebuild the rpart model with model=TRUE.
```



Exercise 5

Now set up a random forest model and workflow. Use the `ranger` engine and set `importance = "impurity"`. Tune `mtry`, `trees`, and `min_n`. Using the documentation for `rand_forest()`, explain in your own words what each of these hyperparameters represent.

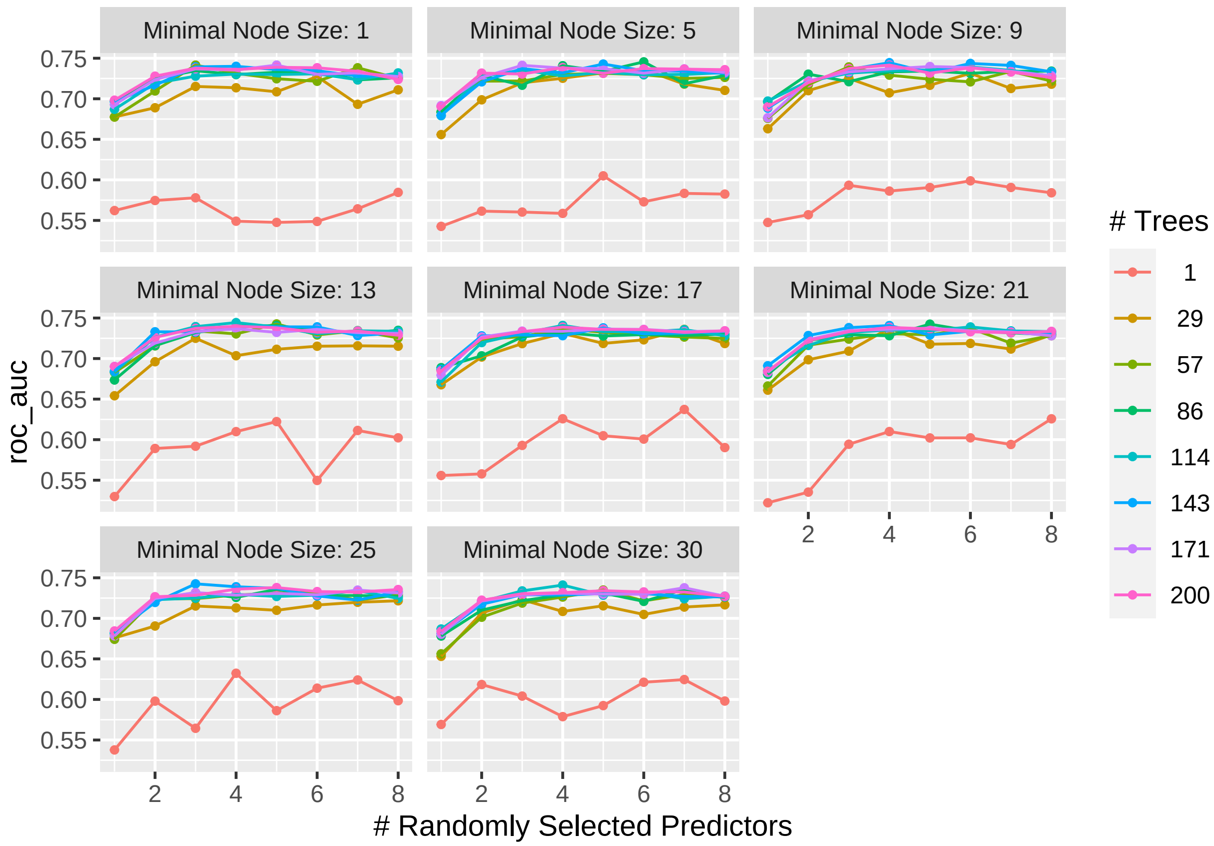
Create a regular grid with 8 levels each. You can choose plausible ranges for each hyperparameter. Note that `mtry` should not be smaller than 1 or larger than 8. **Explain why not. What type of model would `mtry = 8` represent?**

```
forest_spec <- rand_forest() %>%
  set_engine("ranger", importance = "impurity") %>%
  set_mode("classification")
forest_wf <- workflow() %>%
  add_model(forest_spec %>%
  set_args(mtry = tune(), trees = tune(), min_n = tune())) %>%
  add_recipe(pokemon_recipe)
set.seed(3515)

multi_param_grid <- grid_regular(mtry(range = c(1,8)), trees(range(1,200)), min_n(range(1,30)), levels =
multi_tune_res <- tune_grid(
  forest_wf, resamples = pokemon_folds, grid = multi_param_grid, metrics = metric_set(roc_auc)
)
```

Exercise 6

Specify `roc_auc` as a metric. Tune the model and print an `autoplot()` of the results. What do you observe? What values of the hyperparameters seem to yield the best performance?

```
autoplot(multi_tune_res)
```



Exercise 7

What is the `roc_auc` of your best-performing random forest model on the folds? *Hint: Use* `collect_metrics()` *and* `arrange()`.

```
best_metrics <- collect_metrics(multi_tune_res) %>%
  arrange(-mean)

head(best_metrics)
```

```
## # A tibble: 6 x 9
##    mtry trees min_n .metric .estimator  mean     n std_err .config
##   <int> <int> <int> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
## 1     6    86     5 roc_auc hand_till  0.746     5 0.00882 Preprocessor1_Model0~
## 2     4   143     9 roc_auc hand_till  0.744     5 0.0111  Preprocessor1_Model1~
## 3     4   114    13 roc_auc hand_till  0.744     5 0.0114  Preprocessor1_Model2~
## 4     6   143     9 roc_auc hand_till  0.744     5 0.0134  Preprocessor1_Model1~
```

```
## 5     5   143     5 roc_auc hand_till  0.743     5 0.0155  Preprocessor1_Model1~
## 6     5    57    13 roc_auc hand_till  0.743     5 0.00627 Preprocessor1_Model2~
```

```r
best_forest_model <- select_best(multi_tune_res, metric = "roc_auc")

best_forest_model
```

```
## # A tibble: 1 x 4
##    mtry trees min_n .config
##   <int> <int> <int> <chr>
## 1     6    86     5 Preprocessor1_Model094
```

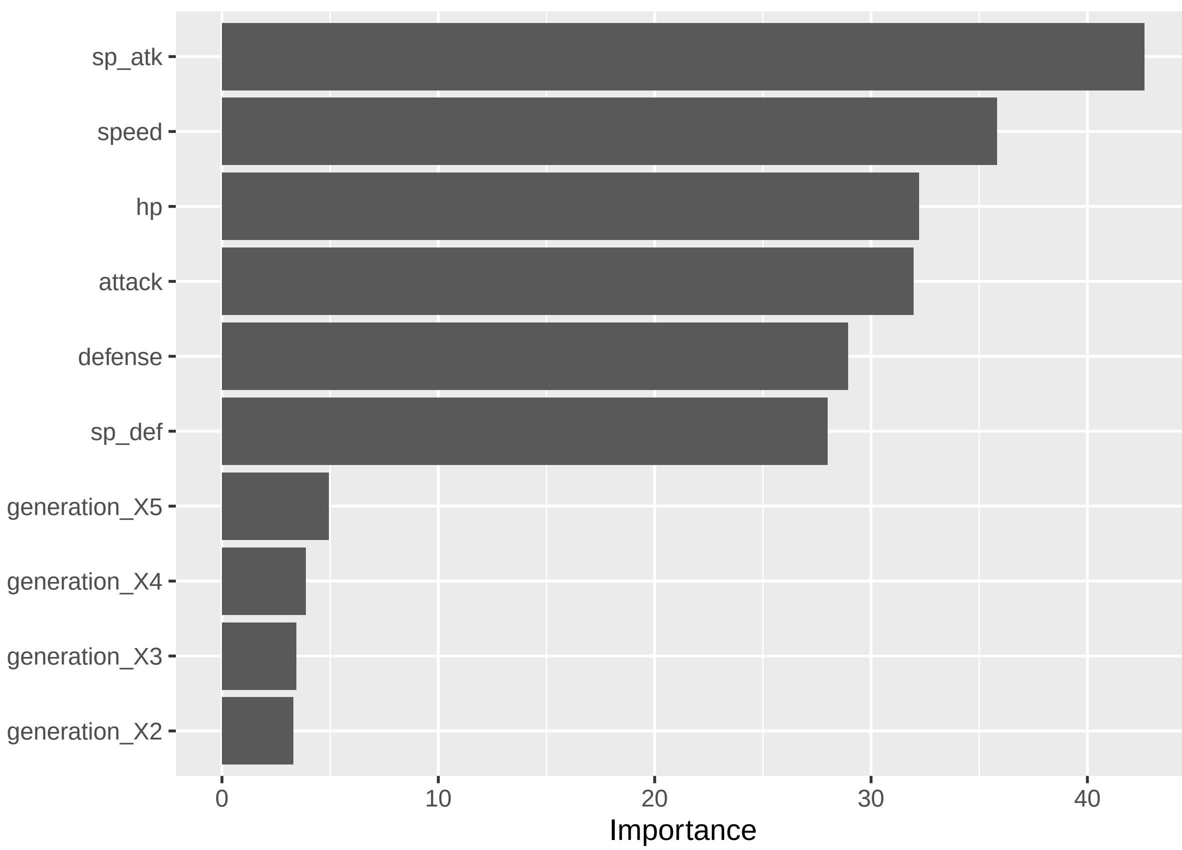
The best model has a roc_auc value of 0.714.

Exercise 8

Create a variable importance plot, using `vip()`, with your best-performing random forest model fit on the *training* set.

Which variables were most useful? Which were least useful? Are these results what you expected, or not?

```r
best_model_final <- finalize_workflow(forest_wf, best_forest_model)
best_model_final_fit <- fit(best_model_final, data = pokemon_train)
best_model_final_fit %>%
  extract_fit_engine() %>%
  vip()
```



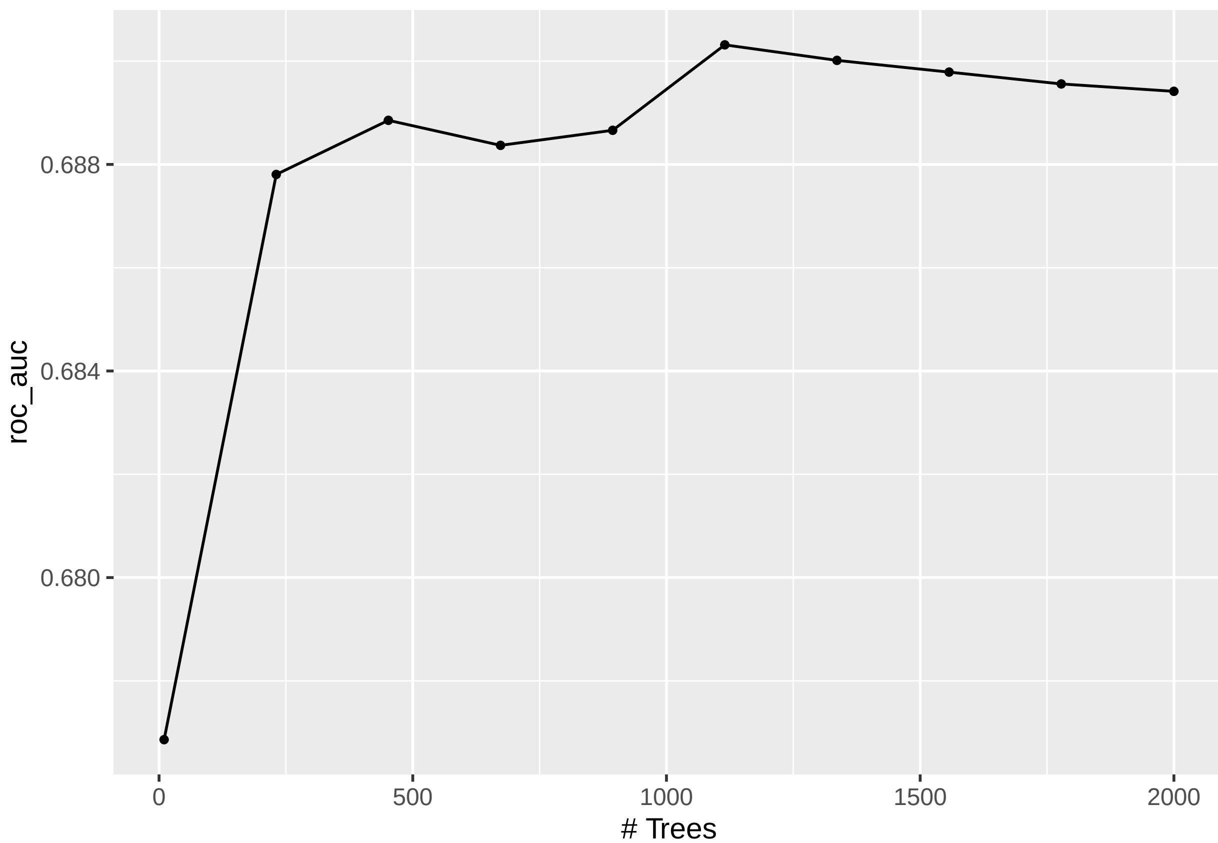
The most useful variables are sp__atk, speed, hp, attack, defense, and sp__def. This is what I expected because generation does not have too much to do with a pokemon's abilities as much as the other variables. ### Exercise 9

Finally, set up a boosted tree model and workflow. Use the `xgboost` engine. Tune `trees`. Create a regular grid with 10 levels; let `trees` range from 10 to 2000. Specify `roc_auc` and again print an `autoplot()` of the results.

What do you observe?

What is the `roc_auc` of your best-performing boosted tree model on the folds? *Hint: Use `collect_metrics()` and `arrange()`.*

```
boost_spec <- boost_tree() %>%
  set_engine("xgboost") %>%
  set_mode("classification")
boost_wf <- workflow() %>%
  add_model(boost_spec %>%
              set_args(trees = tune())) %>%
  add_recipe(pokemon_recipe)
set.seed(500)
boost_grid <- grid_regular(trees(range = c(10,2000)), levels = 10)
boost_tune_res <- tune_grid(
  boost_wf,
  resamples = pokemon_folds,
  grid = boost_grid,
  metrics = metric_set(roc_auc)
)

autoplot(boost_tune_res)
```




```r
collect_metrics(boost_tune_res) %>%
  arrange(-mean) %>%
  head()
```

```
## # A tibble: 6 x 7
##   trees .metric .estimator  mean     n std_err .config
##   <int> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
## 1  1115 roc_auc hand_till  0.690     5  0.0139 Preprocessor1_Model06
## 2  1336 roc_auc hand_till  0.690     5  0.0141 Preprocessor1_Model07
## 3  1557 roc_auc hand_till  0.690     5  0.0140 Preprocessor1_Model08
## 4  1778 roc_auc hand_till  0.690     5  0.0143 Preprocessor1_Model09
## 5  2000 roc_auc hand_till  0.689     5  0.0141 Preprocessor1_Model10
## 6   452 roc_auc hand_till  0.689     5  0.0135 Preprocessor1_Model03
```

```r
best_boost_final <- select_best(boost_tune_res)
best_boost_final_model <- finalize_workflow(boost_wf, best_boost_final)
best_boost_final_model_fit <- fit(best_boost_final_model, data = pokemon_train)
```

The roc_auc sharply increases as the number of trees approaches about 250, and keeps increasing from there. The roc_auc of the best performing boosted tree model is 0.684.

Exercise 10

Display a table of the three ROC AUC values for your best-performing pruned tree, random forest, and boosted tree models. Which performed best on the folds? Select the best of the three and use `select_best()`, `finalize_workflow()`, and `fit()` to fit it to the *testing* set.

Print the AUC value of your best-performing model on the testing set. Print the ROC curves. Finally, create and visualize a confusion matrix heat map.

Which classes was your model most accurate at predicting? Which was it worst at?