# Homework 6

PSTAT 131/231

## Contents

```
library(tidyverse)
library(tidymodels)
library(rpart.plot)
library(ISLR)
library(vip)
library(randomForest)
library(xgboost)
```

**Exercise 1**

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

- Use `clean_names()`
- Filter out the rarer Pokémon types
- Convert `type_1` and `legendary` to factors

```
library(janitor)
pokemon <- read.csv('Pokemon.csv')
pokemon <- janitor:: clean_names(dat = pokemon)
pokemon1 <- pokemon  %>%
  filter(type_1  %in%  c('Bug','Fire', 'Grass', 'Normal', 'Water', 'Psychic'))
```

```
pokemon2 <- pokemon1 %>%
  mutate(type_1 = factor(type_1),legendary = factor(legendary),
         generation = factor(generation))
```

Do an initial split of the data; you can choose the percentage for splitting. Stratify on the outcome variable.

```
set.seed(3435)
pokemon_split <- initial_split(pokemon2, prop = 0.7,
                               strata = 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.

```
pokemon_folds <- vfold_cv(pokemon_train, v = 5, strata = type_1)
```

Set up a recipe to predict `type_1` with `legendary`, `generation`, `sp_atk`, `attack`, `speed`, `defense`, `hp`, and `sp_def`:

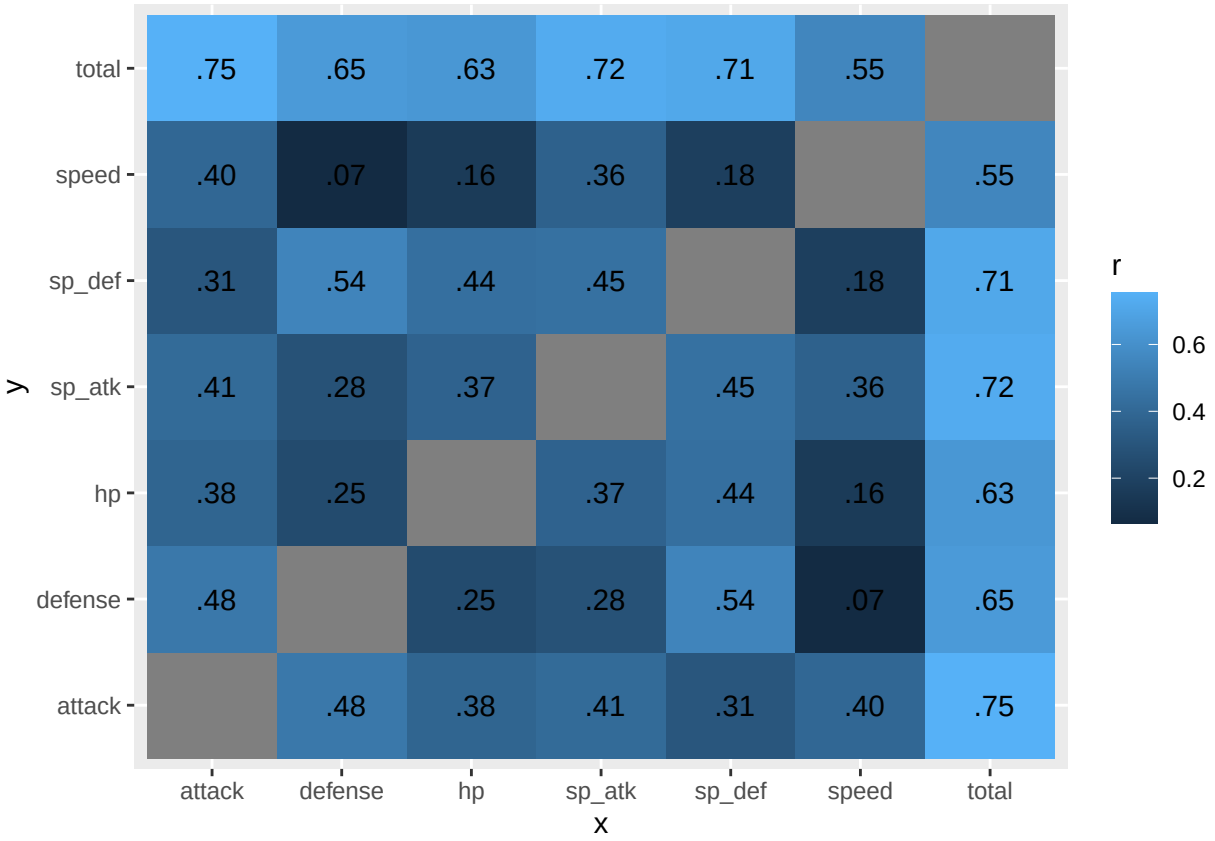
- Dummy-code `legendary` and `generation`;
- Center and scale all predictors.

```
pokemon_recipe <-
  recipe(type_1 ~ legendary + generation + sp_atk + attack + speed + defense + hp + sp_def,
         data = pokemon_train) %>%
  step_dummy(legendary, 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).*

```
library(corrr)
cor_pokemon <- pokemon_train %>%
  select(-x) %>%
  select(is.numeric) %>%
  correlate()

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




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

I choose to remove the variable x, because it is just the ID of each Pokemon. Also, I remove other categorical value. In the plot above, we could notice that total is positive correlate to other variables except for generation. These relationships make sense to me. Total is a general guide to how strong a Pokemon is, so each if each attributes is high and the total would be high.

### Exercise 3

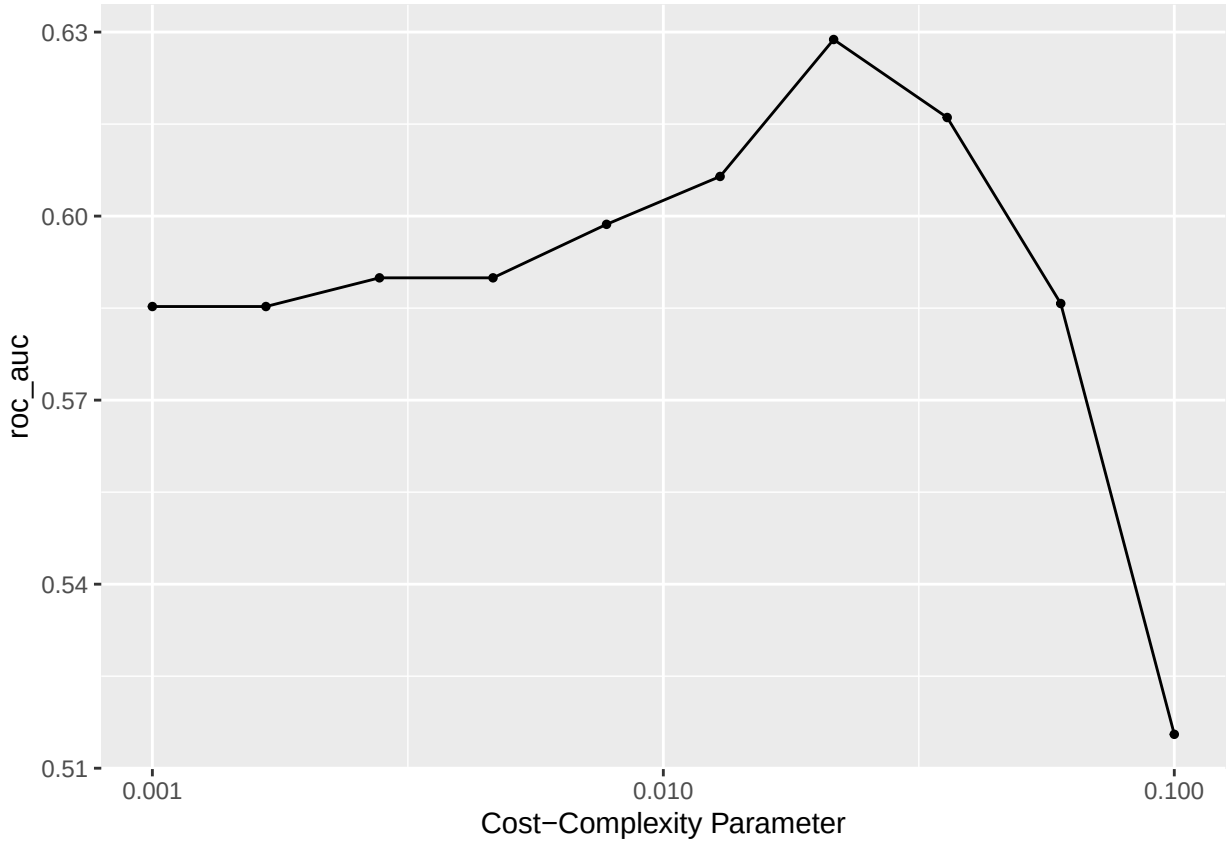
First, set up a decision tree model and workflow. Tune the `cost_complexity` hyperparameter. Use the same levels we used in Lab 7 – that is, `range = c(-3, -1)`. Specify that the metric we want to optimize is `roc_auc`.

```
tree_spec <- decision_tree() %>%
  set_engine("rpart") %>%
  set_mode("classification") %>%
  set_args(cost_complexity = tune())

class_tree_wf <- workflow() %>%
  add_model(tree_spec) %>%
  add_recipe(pokemon_recipe)

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)
)
```

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

```
autoplot(tune_res)
```




I notice when cost-complexity parameter increase, our roc_auc increase.However, when cost-complexity parameter close to 0.07, roc_auc reaches to peak and decrease sharply. A single decision tree perform better with a smaller complexity penalty.

**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)
tree_roc_auc <- tree_roc_auc %>% head(1)
tree_roc_auc
```

```
## # A tibble: 1 x 7
```

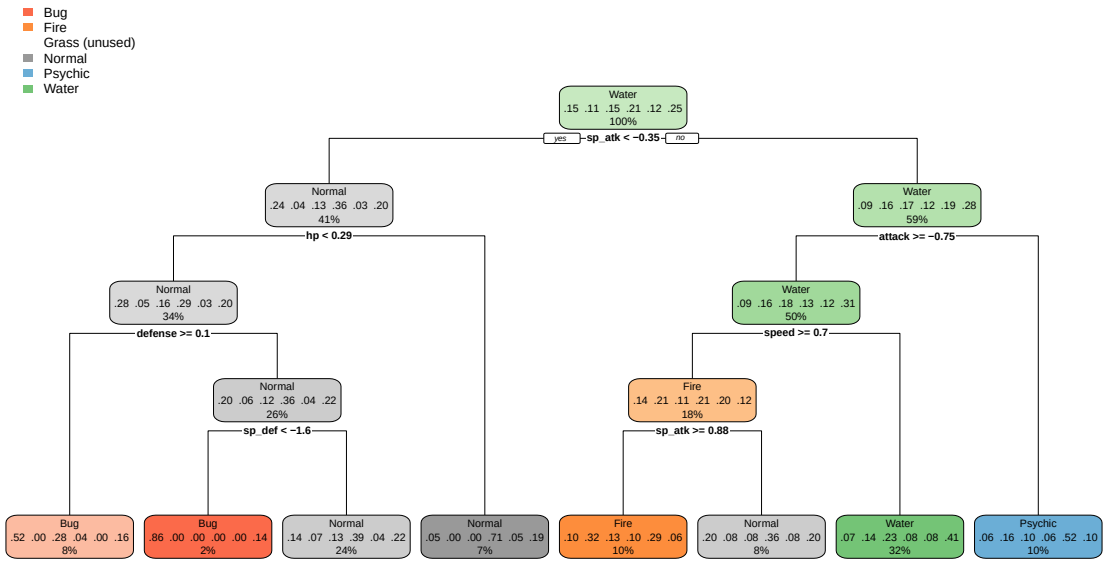
```
##   cost_complexity .metric .estimator  mean     n std_err .config
##             <dbl> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
## 1          0.0215 roc_auc hand_till  0.629     5  0.0191 Preprocessor1_Model07
```

The roc_auc of best-performing pruned decision tree on the folds is 0.628788

### 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()
```




### 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.

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

```
rf_wf <- workflow() %>%
  add_recipe(pokemon_recipe) %>%
  add_model(rf_spec)
```

- mtry means when creating a tree model how many predictors will be randomly sampled at each split
- trees means the number of trees contained in the ensemble
- min_n means the minimum number of data points in a node that are required for the node to be split further.

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?**

```
rp_grid <- grid_regular(mtry(range = c(2,7)), trees(range = c(10,2000)),
                        min_n(range = c(2, 10)), levels = 8)
```

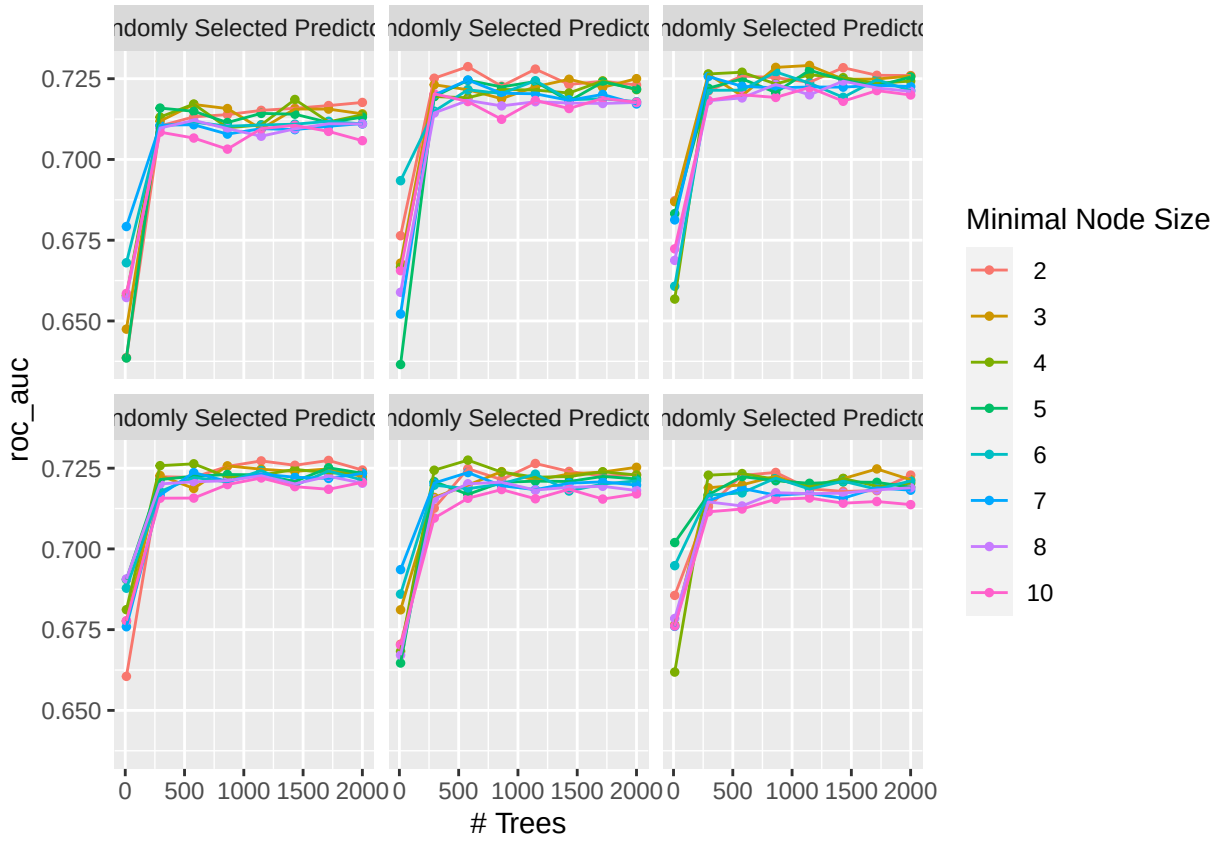
mtry should not be smaller than 1 or larger 8 because we need use at least one predictor in our model and total number of predictors are 8 so mtry could not larger than 8. When mtry = 8, we use all the predictors and the type of model is bagging.

### 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?

```
tune_rf <- tune_grid(
  rf_wf,
  resamples = pokemon_folds,
  grid = rp_grid,
  metrics = metric_set(roc_auc)
)
```

```
autoplot(tune_rf)
```




I observed that when number of trees increases to 250, our roc_auc increase sharply, and then when number of trees increases, our roc_auc does not have a distinct increase. The number of predictors choice has no large influence on our roc_auc, the overall trend seems similar to each other. The choice of minimal node size seems have no large effect on our model accuracy.

**Exercise 7**

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

```
rf_roc_auc <- collect_metrics(tune_rf) %>%
  arrange(-mean)
rf_roc_auc <- rf_roc_auc %>%  head(1)
rf_roc_auc
```

```
## # A tibble: 1 x 9
##    mtry trees min_n .metric .estimator  mean     n std_err .config
##   <int> <int> <int> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
## 1     4  1147     3 roc_auc hand_till  0.729     5  0.0139 Preprocessor1_Model0~
```

The roc_auc of my best-performing model on the folds is 0.729108

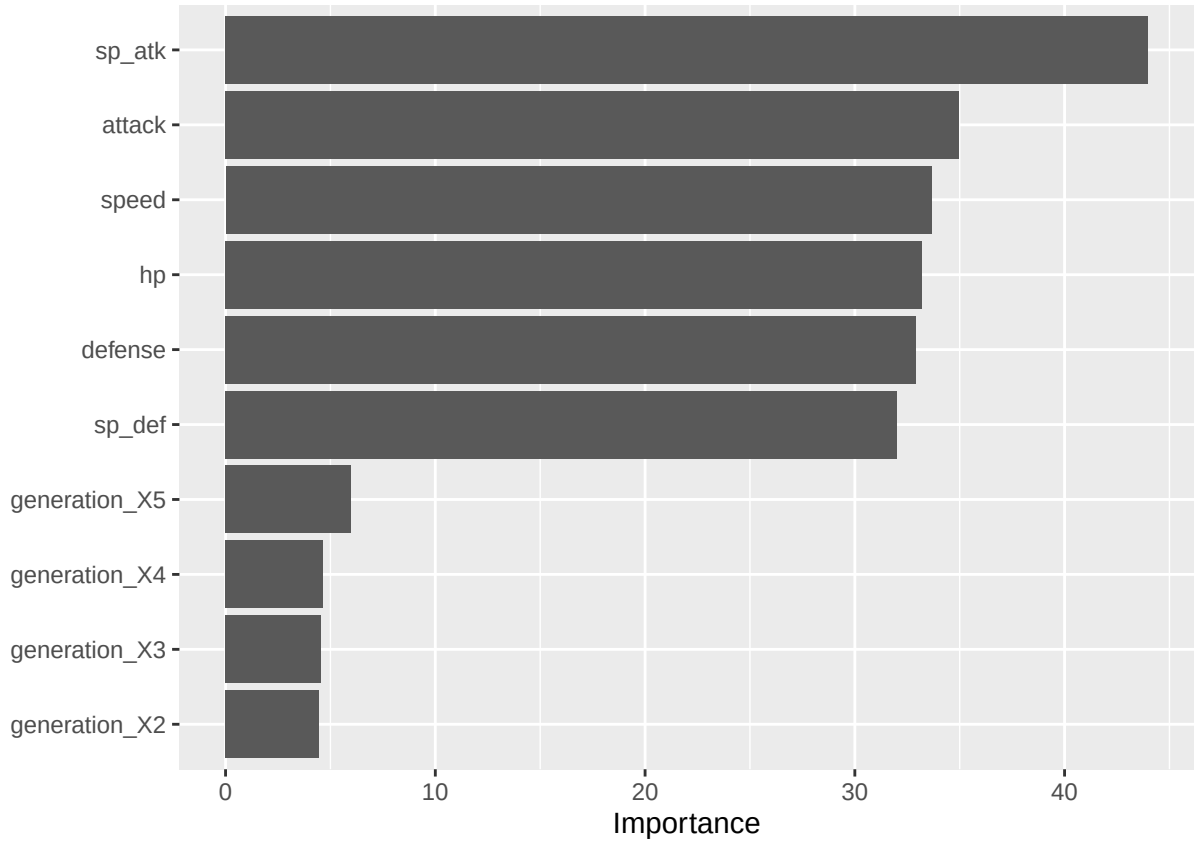
**Exercise 8**

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?

```
best_rf <- select_best(tune_rf, metric = "roc_auc")
rf_final <- finalize_workflow(rf_wf, best_rf)
rf_final_fit <- fit(rf_final, data = pokemon_train)
```

```
rf_final_fit %>%
  extract_fit_engine() %>%
  vip()
```



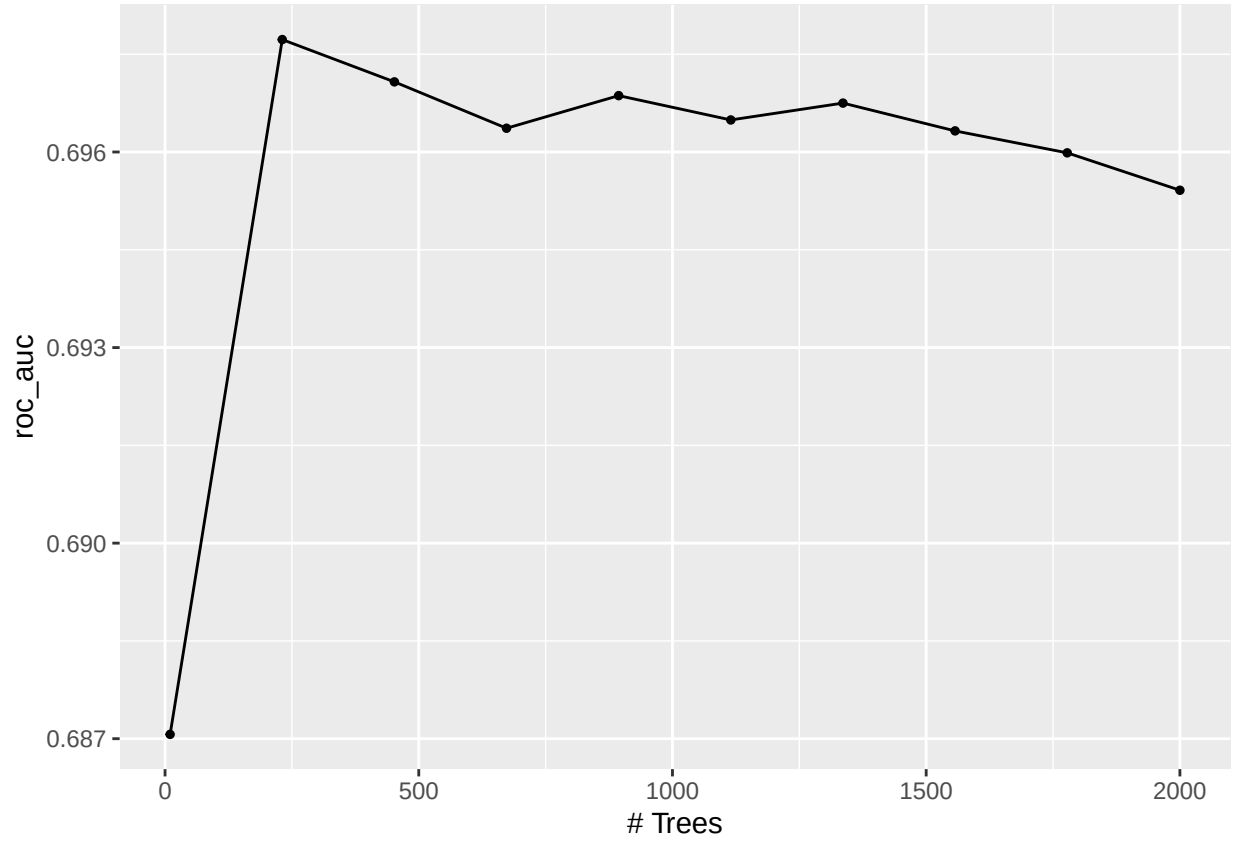

Sp_atk and attack were most useful, and generation and legendary were least useful. Some other attributes like hp speed defense were still important. These results are what I expected because type could determine Pokemon resistent to others.

**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.

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

boost_wf <- workflow() %>%
```

```
  add_recipe(pokemon_recipe) %>%
  add_model(boost_spec)

trees_grid2 <- grid_regular(trees(range = c(10,2000)), levels = 10)


tune_bt <- tune_grid(
  boost_wf,
  resamples = pokemon_folds,
  grid = trees_grid2,
  metrics = metric_set(roc_auc)
)

autoplot(tune_bt)
```




What do you observe?

The roc_auc reach to the peak at #Trees around 250, and then has a decreasing trend.

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

```
boost_roc_auc <- collect_metrics(tune_bt) %>%
  arrange(-mean)
boost_roc_auc <- boost_roc_auc %>% head(1)
boost_roc_auc
```

```
## # A tibble: 1 x 7
##   trees .metric .estimator  mean     n std_err .config
##   <int> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
## 1   231 roc_auc hand_till  0.698     5  0.0139 Preprocessor1_Model02
```

The roc_auc of my best-performing boosted tree model on the fold is 0.6977248

**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.

```
model_roc_auc <- c(tree_roc_auc$mean, rf_roc_auc$mean, boost_roc_auc$mean)
model <- c("pruned tree", "random forest", "boosted tree")
result <- tibble(model_roc_auc, model)
result %>% arrange(-model_roc_auc)
```

```
## # A tibble: 3 x 2
##   model_roc_auc model
##           <dbl> <chr>
## 1         0.729 random forest
## 2         0.698 boosted tree
## 3         0.629 pruned tree
```

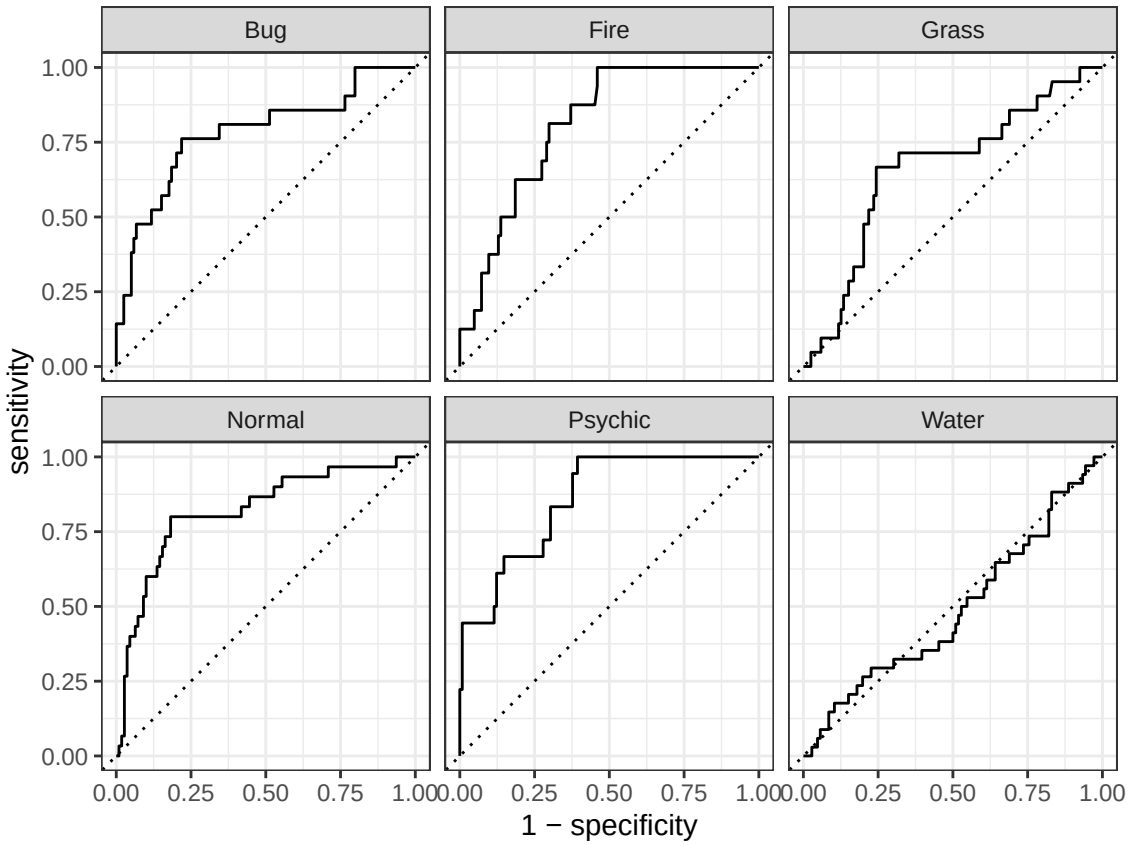
- random forest performed best on the folds

```
rf_parameters <- select_best(tune_rf, metric = "roc_auc")
rf_final_test <- finalize_workflow(rf_wf, rf_parameters)
rf_final_testfit <- fit(rf_final_test, data = pokemon_test)
```

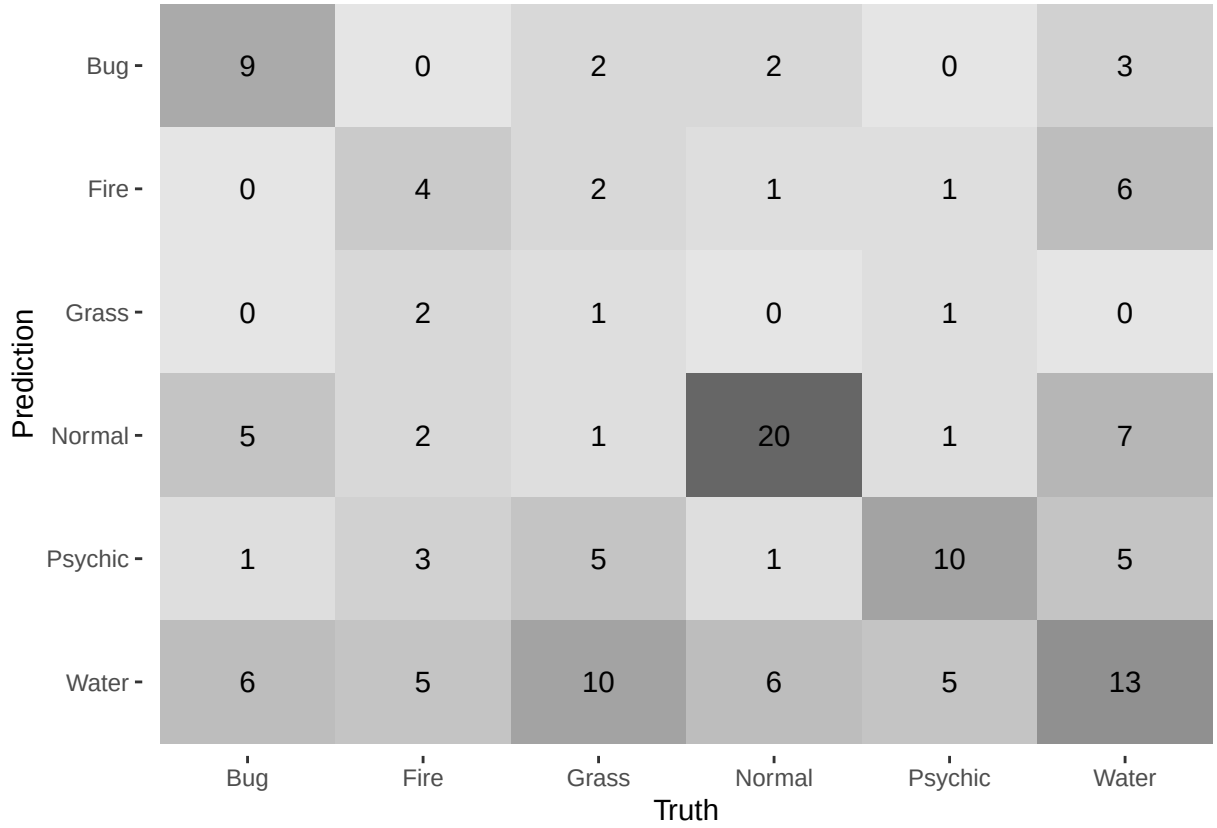
Print the AUC value of your best-performing model on the testing set. Print the ROC curves. Finally, create and visualize a confusion matrix heat map.

```
predictions_result <- augment(rf_final_fit, new_data = pokemon_test) %>%
  select(type_1, .pred_class, .pred_Bug, .pred_Fire, .pred_Grass, .pred_Normal, .pred_Psychic,
         .pred_Water)
```

```
predictions_result %>%
  roc_curve(type_1, .pred_Bug, .pred_Fire, .pred_Grass, .pred_Normal, .pred_Psychic,
            .pred_Water) %>%
  autoplot()
```




```
predictions_result %>%  conf_mat(type_1, .pred_class) %>%
  autoplot(type = "heatmap")
```




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

My model was most accurate at predicting Normal, and it was worst at predicting class Water.