Homework 6

Mason Wong

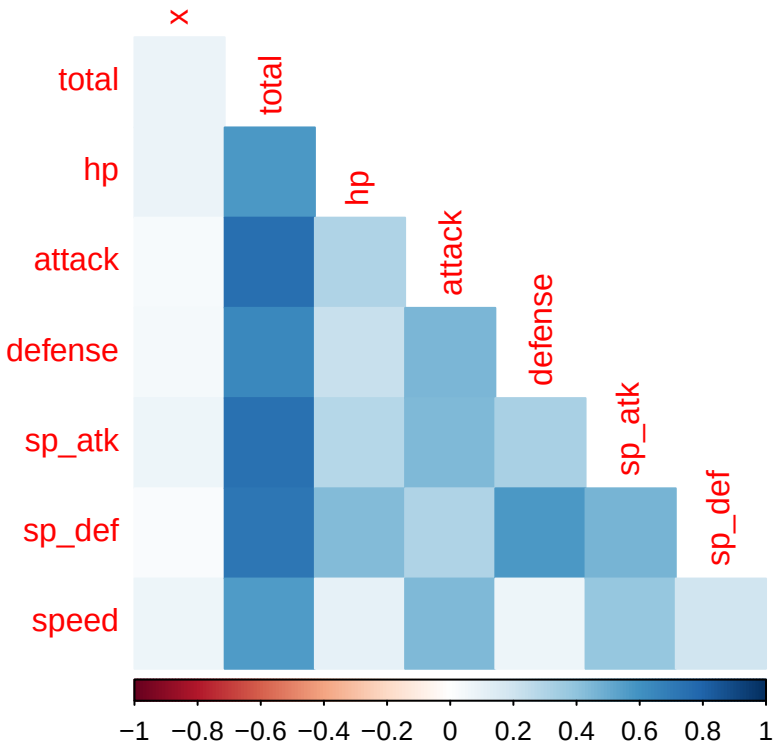
2022-11-28

Exercise 1

```
library(janitor)
library(dplyr)
library(corrplot)
library(tidyverse)
library(rpart.plot)
library(tune)
library(ranger)
library(vip)
library(tidymodels)
library(parsnip)
library(xgboost)

poke <- read.csv("Pokemon.csv")
set.seed(111)

poke_clean <- poke %>%
  clean_names()

poke_filter <- poke_clean %>%
  filter(type_1 == "Bug"|type_1 == "Fire"|type_1 == "Grass"|
           type_1 == "Normal"|type_1 == "Water"|type_1 == "Psychic")

poke_filter$type_1 <- as.factor(poke_filter$type_1)
poke_filter$legendary <- as.factor(poke_filter$legendary)
poke_filter$generation <- as.factor(poke_filter$generation)

poke_split <- initial_split(poke_filter, prop = 0.7, strata = type_1)
poke_train <- training(poke_split)
poke_test <- testing(poke_split)

poke_fold <- vfold_cv(poke_train, v = 5, strata = type_1)

poke_recipe <- recipe(type_1 ~ legendary + generation + sp_atk +  attack +
                        speed + defense + hp + sp_def, data = poke_train) %>%
  step_dummy(all_nominal_predictors())

poke_recipe <- step_center(recipe = poke_recipe, sp_atk, attack, speed, defense, sp_def, hp)
poke_recipe <- step_scale(recipe = poke_recipe, sp_atk, attack, speed, defense, sp_def, hp)
```

Exercise 2

```
poke_train %>%
  select_if(is.numeric) %>%
  cor(use = "complete.obs") %>%
  corrplot(type = 'lower', diag = FALSE, method = 'color')
```



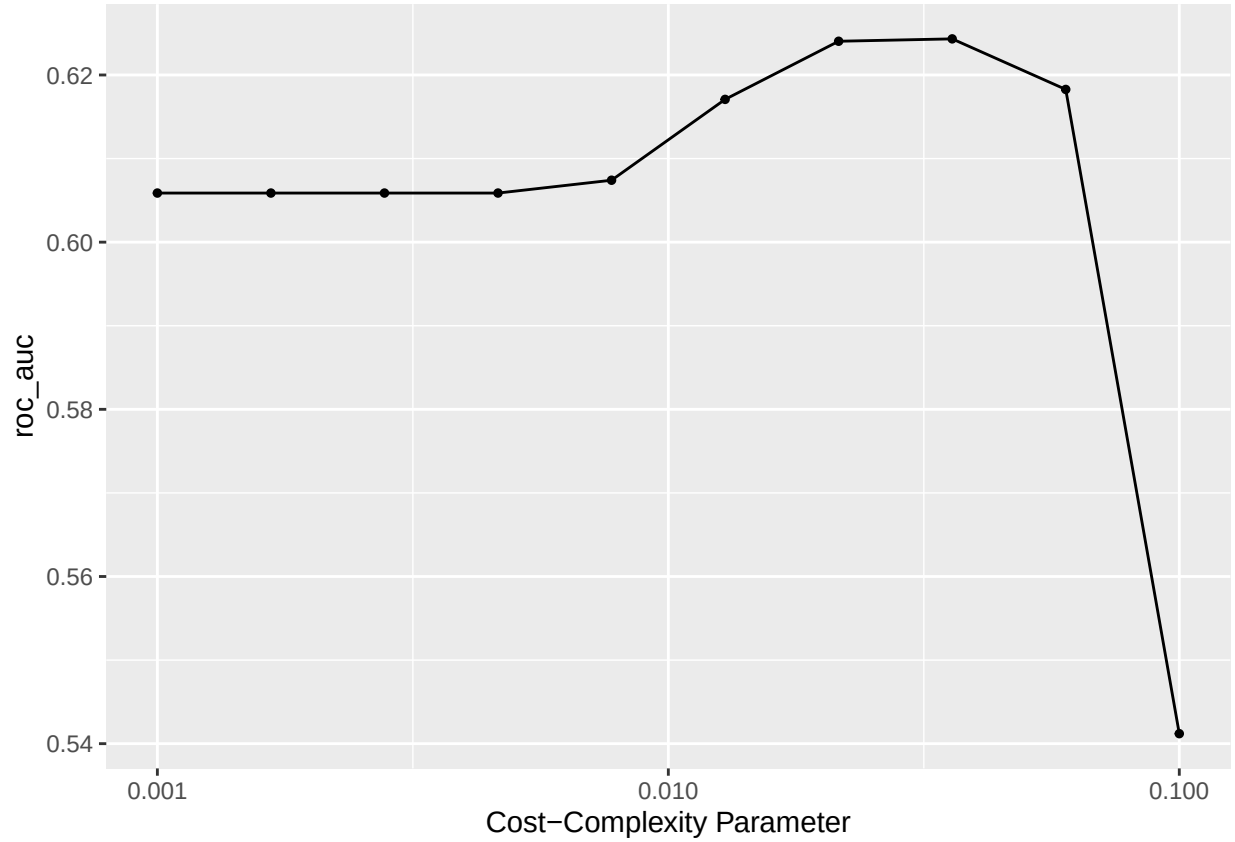
Looking at the corrplot, it makes sense that there is no correlation between the pokemon's number and any of the stats. It also makes sense that all the different stat points have a positive correlation with the stat total as an increase in any of these stats result in the stat total increasing.

It seems that pokemon with higher hp and defense stats had lower speed, possibly because they are heavier and thus bulkier but slower. For the most part though, it makes sense to have positive stat correlation between the 6 different stats because a stronger pokemon would see higher stats than a weaker pokemon, so having a higher stat in attack will most likely lead to having higher stats in other categories when compared to weaker pokemon.

Exercise 3

```
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(poke_recipe)

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

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

autoplot(tune_res)
```



According to our graphs, a single tree performs better when cost complexity is above 0.01 and before 0.075

Exercise 4

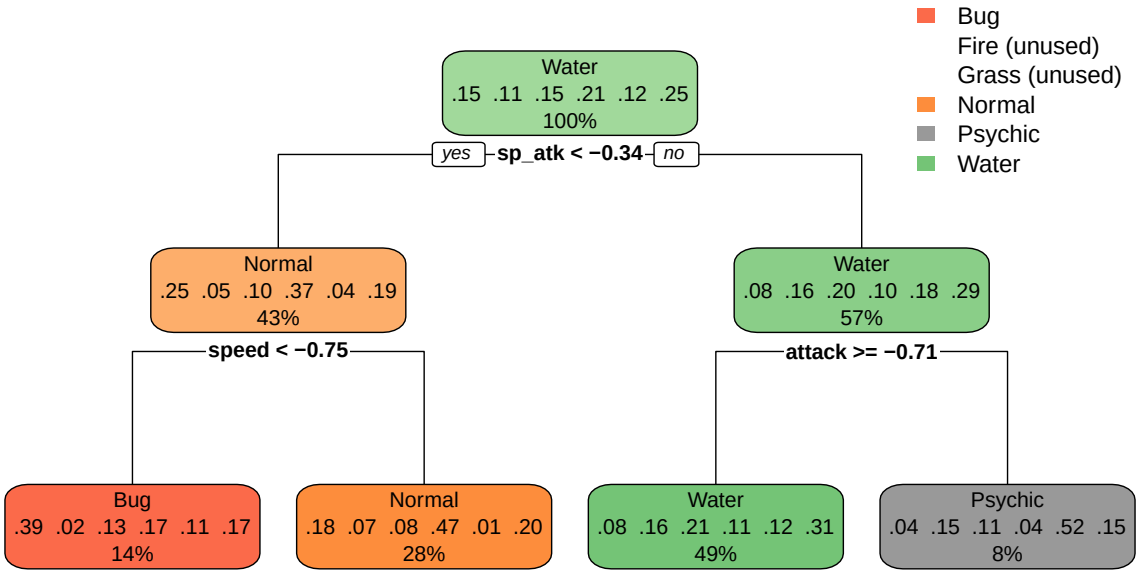
```
tree_best <- tail(collect_metrics(tune_res) %>% arrange(mean), n = 1)
tail(collect_metrics(tune_res) %>% arrange(mean), n = 1)
```

```
## # A tibble: 1 x 7
##   cost_complexity .metric .estimator  mean     n std_err .config
##             <dbl> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
## 1          0.0359 roc_auc hand_till  0.624     5  0.0152 Preprocessor1_Model08
```

Exercise 5

```
best_complexity <- select_best(tune_res)

class_tree_final <- finalize_workflow(class_tree_wf, best_complexity)

class_final_fit <- fit(class_tree_final, data = poke_train)

class_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.
```



```
r_model <- rand_forest() %>%
  set_engine("ranger", importance = "impurity") %>%
  set_mode("classification")

r_forest_wf <- workflow() %>%
  add_model(r_model %>% set_args(mtry = tune(), trees = tune(), min_n = tune())) %>%
  add_recipe(poke_recipe)

r_grid <- grid_regular(mtry(range = c(1,8)), trees(range = c(2,16)), min_n(range = c(2,16)), levels = 8)
r_grid
```

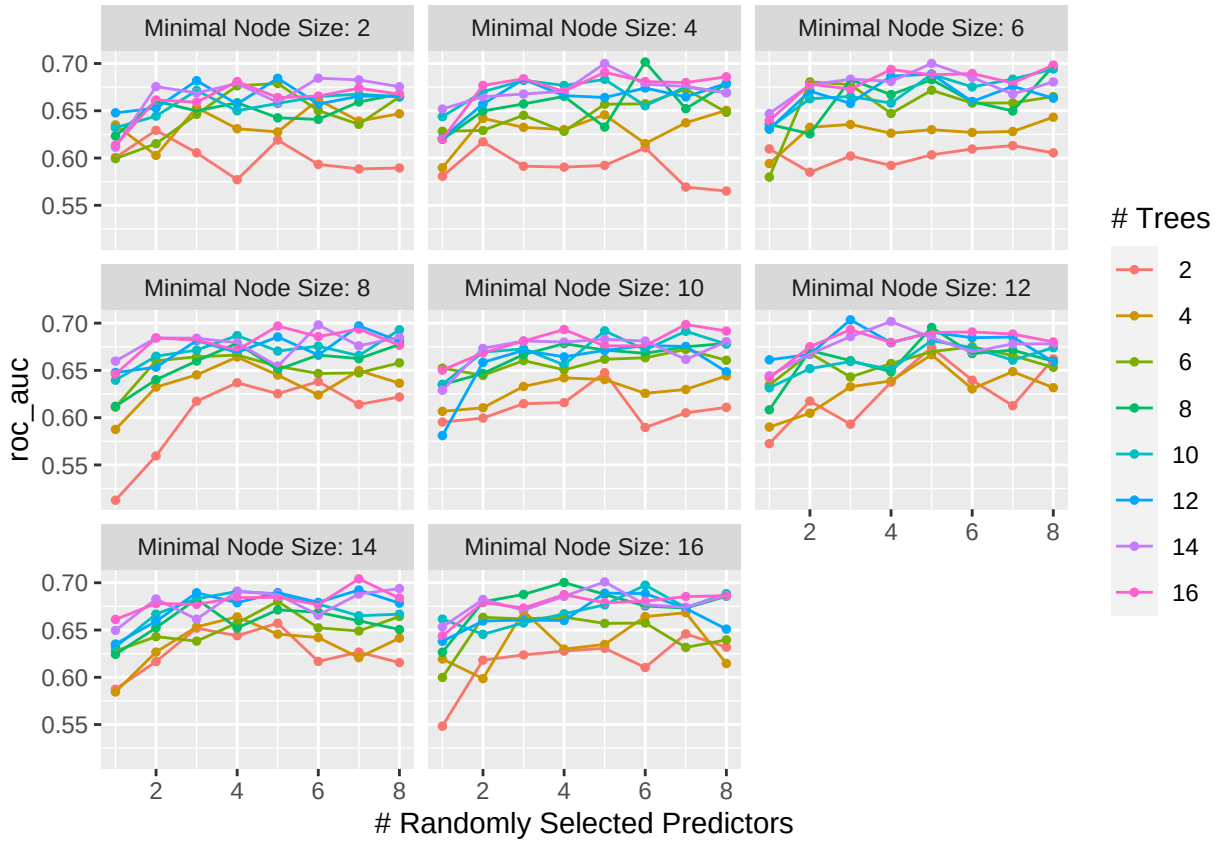
```
## # A tibble: 512 x 3
##     mtry trees min_n
##    <int> <int> <int>
##  1     1     2     2
##  2     2     2     2
##  3     3     2     2
##  4     4     2     2
##  5     5     2     2
##  6     6     2     2
##  7     7     2     2
##  8     8     2     2
##  9     1     4     2
## 10     2     4     2
## # ... with 502 more rows
```

Trees is the number of trees in the forest model. mtry is the number of predictors that are randomly selected for each tree. min_n is the minimum number of observations in each node.

mtry must be between 1 and 8 because there wouldn't be a model if we included 0 predictors, and we only have 8 predictors in our recipe. In the case of mtry being equal to 8, we would have a bagging model.

Exercise 6

```
r_tune_res <- tune_grid(
  r_forest_wf,
  resamples = poke_fold,
  grid = r_grid,
  metrics = metric_set(roc_auc)
)

autoplot(r_tune_res)
```



Exercise 7

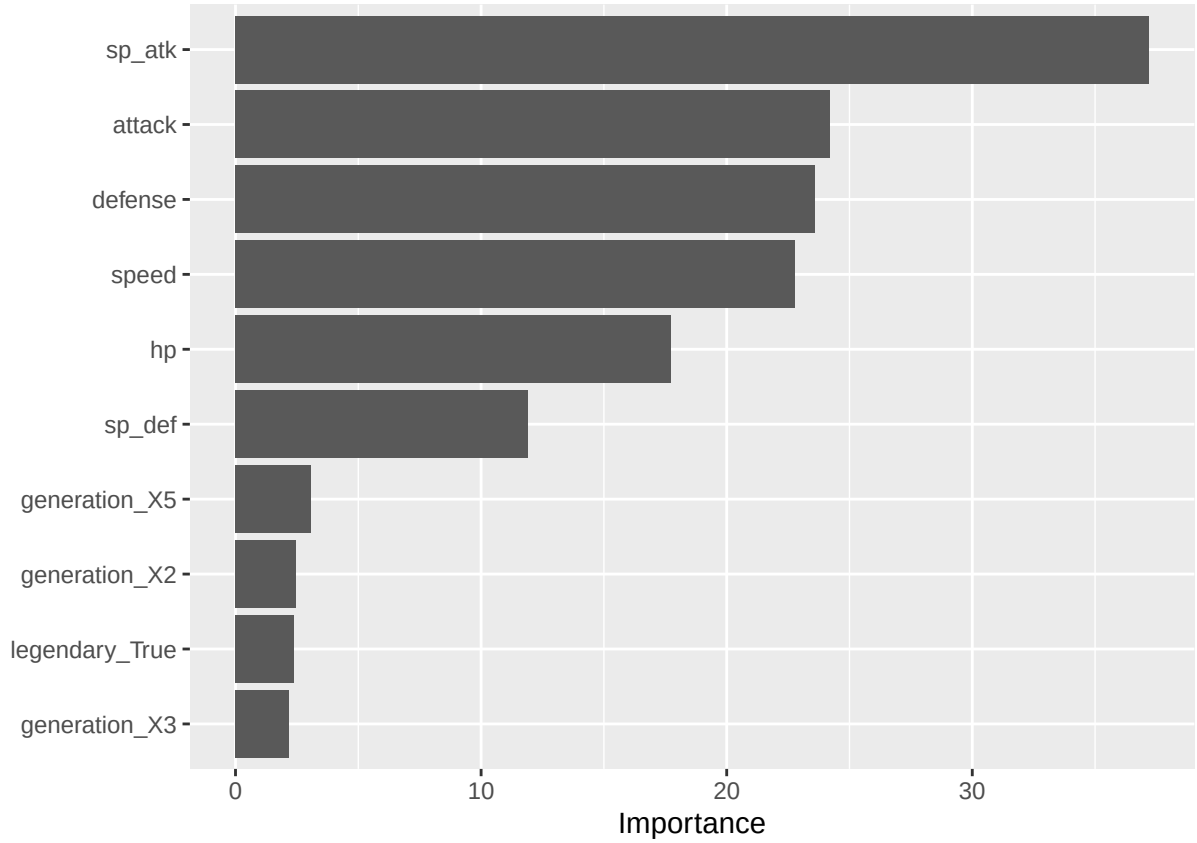
```
forest_best <- tail(collect_metrics(r_tune_res) %>% arrange(mean), n = 1)
tail(collect_metrics(r_tune_res) %>% arrange(mean), n = 1)
```

```
## # 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     7    16    14 roc_auc hand_till  0.704     5 0.00482 Preprocessor1_Model4~
```

My best random-forest model on my folded data has a mean roc area under the curve of 0.7041827

Exercise 8

```
best_r <- select_best(r_tune_res)
r_tree_final <- finalize_workflow(r_forest_wf, best_r)
r_final_fit <- fit(r_tree_final, data = poke_train)

r_final_fit %>%
  extract_fit_parsnip() %>%
  vip()
```



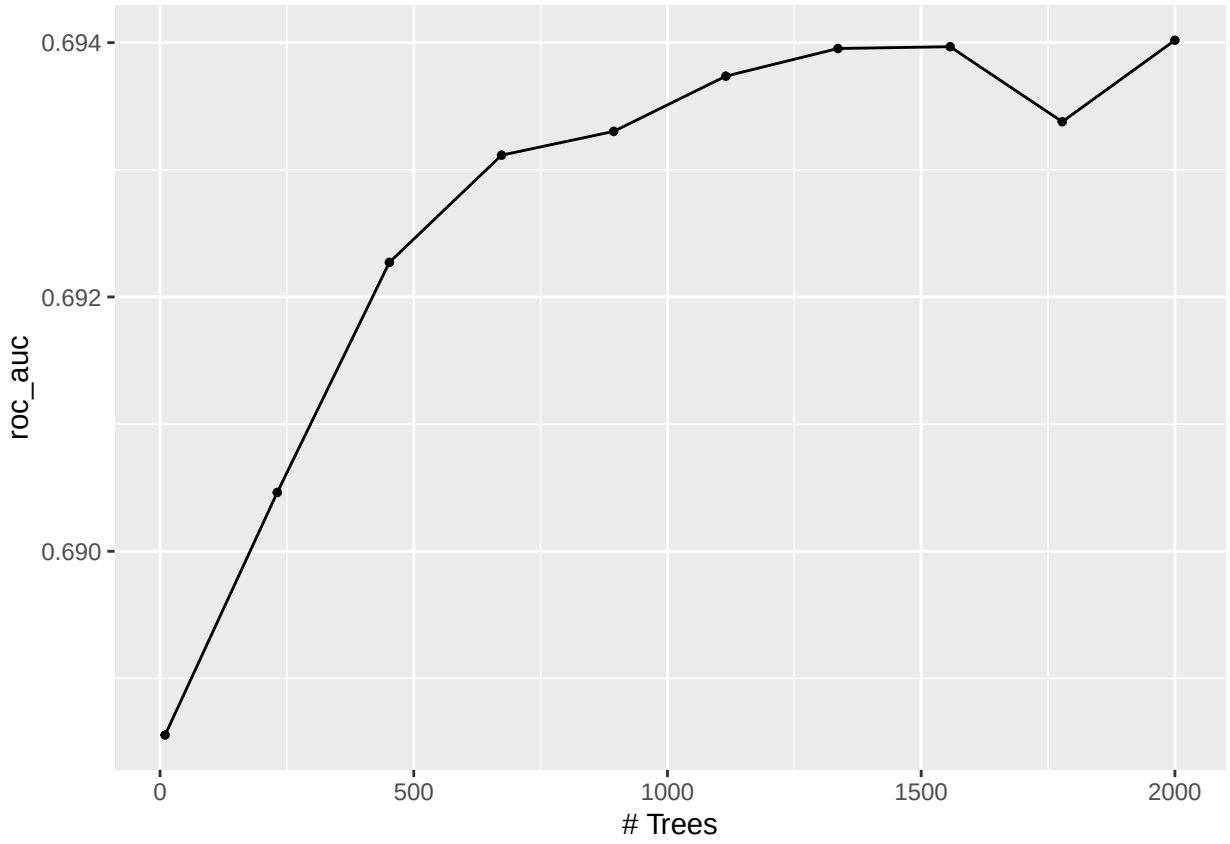
It would seem that special attack and attack were the most useful while the nominal predictors (generation and legendary) were the least useful. This makes sense since pokemon in each generation have varying types so it would be difficult to tell a pokemon's type from the generation alone.

Furthurmore, being a legendary doesn't necessarily specify a type. There are legendaries of different types and their scarcity means it would be difficult for our model to classify a legendary's type just off of the fact that it is a legendary.

Exercise 9

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

boost_wf <- workflow() %>%
  add_model(boost_spec %>% set_args(trees = tune())) %>%
  add_recipe(poke_recipe)

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

boost_tune_res <- tune_grid(
  boost_wf,
  resamples = poke_fold,
  grid = boost_grid,
  metrics = metric_set(roc_auc)
```

```
)

autoplot(boost_tune_res)
```




```
boost_best <- tail(collect_metrics(boost_tune_res) %>% arrange(mean), n = 1)
tail(collect_metrics(boost_tune_res) %>% arrange(mean), n = 1)
```

```
## # A tibble: 1 x 7
##   trees .metric .estimator  mean     n std_err .config
##   <int> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
## 1  2000 roc_auc hand_till  0.694     5  0.0164 Preprocessor1_Model10
```

It seems that my roc area under the curve is highest from 1300-1550 trees as well as at 2000 trees. My autoplot seems to have a constant increase with slight dips at around 900 and 1800.

My best model using the folded data has a mean of 0.6940187.

Exercise 10

```
table <- as.data.frame(bind_rows(tree_best, forest_best, boost_best))

row.names(table) <- c("Pruned Tree", "Random Forest", "Boosted Tree")
```

```
table
```

```
##               cost_complexity .metric .estimator      mean n    std_err
## Pruned Tree        0.03593814 roc_auc  hand_till 0.6243216 5 0.01520951
## Random Forest              NA roc_auc  hand_till 0.7041827 5 0.00481537
## Boosted Tree               NA roc_auc  hand_till 0.6940187 5 0.01635576
##                              .config mtry trees min_n
## Pruned Tree    Preprocessor1_Model08   NA    NA    NA
## Random Forest Preprocessor1_Model447    7    16    14
## Boosted Tree   Preprocessor1_Model10   NA  2000    NA
```

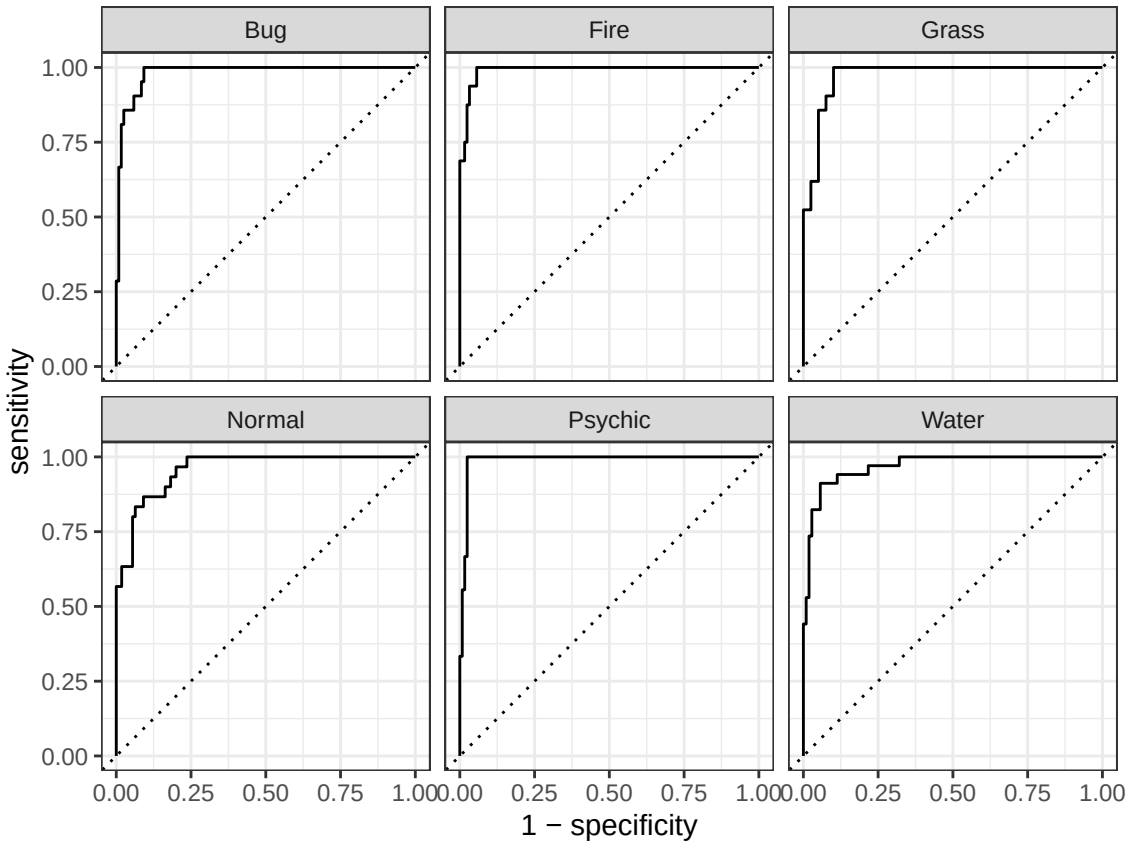
It seems that my random forest model was the best model

```
best_model <- select_best(r_tune_res)
best_final <- finalize_workflow(r_forest_wf, best_model)
best_final_fit <- fit(best_final, data = poke_test)

augment(best_final_fit, new_data = poke_test) %>%
  roc_auc(truth = type_1, estimate = .pred_Bug:.pred_Water)
```

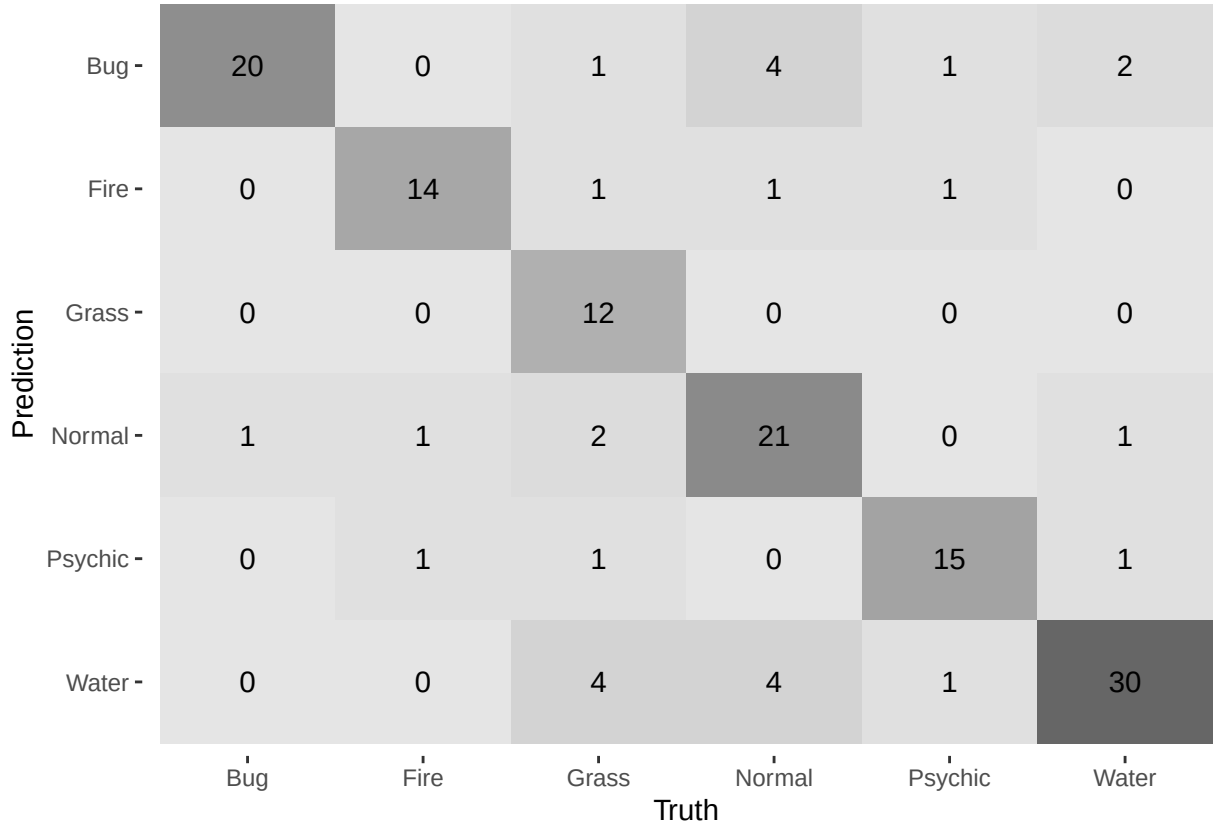
```
## # A tibble: 1 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 roc_auc hand_till      0.978
```

The roc area under the curve for my best model had an area of 0.9773773 on the testing set

```
augment(best_final_fit, new_data = poke_test) %>%
  roc_curve(truth = type_1, estimate = .pred_Bug:.pred_Water) %>%
  autoplot()
```




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



It seems that water normal and bug had the most accurate predictions. Meanwhile Fire and Grass were by far the biggest underpreformers.