PSTAT 131 Final Project

Kenneth Villatoro

2023-12-11

```
#libraries that will be used
library(tidyverse)
library(tidymodels)
library(ggplot2)
library(corrplot)
library(corrr)
library(MASS)
library(ISLR)
library(ISLR2)
library(ggthemes)
library(parsnip)
library(readxl)
library(ranger)
library(glmnet)
library(modeldata)
library(janitor)
library(naniar)
library(xgboost)
library(ranger)
library(vip)
library(corrplot)
tidymodels_prefer()
```

Table of Contents

Introduction

What I will be predicting in this final Project are the outcome of species in terms of the numeric predictors that the data set provides. The data set I will be using in this final project is the Palmer Penguins data set from Kaggle. The website of this data set is located below along with the code book in order to understand the values used during the prediction. :

Website: https://www.kaggle.com/datasets/parulpandey/palmer-archipelago-antarctica-penguin-data

Codebook: *sent separately as well*

- species: penguin species (Chinstrap, Adélie, or Gentoo)
- culmen_length_mm: culmen length (mm)
- culmen_depth_mm: culmen depth (mm)

- flipper_length_mm: flipper length (mm)
- body_mass_g: body mass (g)
- island: island name (Dream, Torgersen, or Biscoe) in the Palmer Archipelago (Antarctica)
- sex: penguin sex

Explatory Data Analysis

First off, we read in the data set while making the variables species, island, and sex all factors. The reason for this is because these three variables are nominal variables, or in other words, categorical variables. These variables must be converted in order to use these variables for the prediction. We also took off any data with any missing values as it can alter the prediction by giving null values.

```
penguins<-read_excel("/Users/kennethvillatoro/Desktop/pstat 131/penguins_size.xlsx",
                     na = "NA")

penguins$species <- as.factor(penguins$species)
penguins$island <- as.factor(penguins$island)
penguins$sex <- as.factor(penguins$sex)

penguins <- na.omit(penguins)

head(penguins)
```

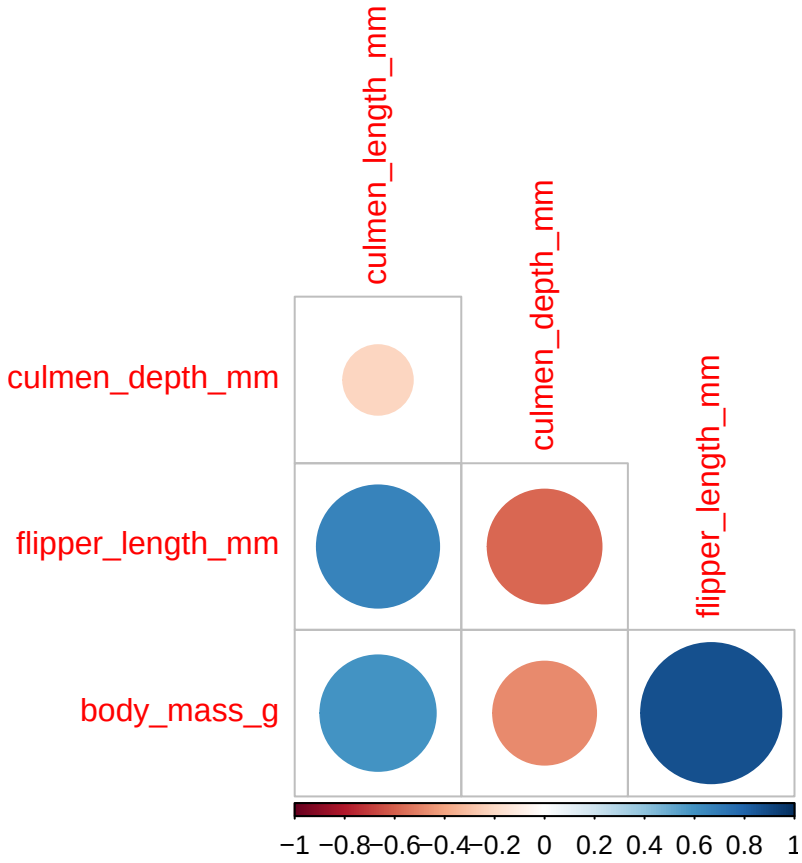
```
## # A tibble: 6 x 7
##   species island  culmen_length_mm culmen_depth_mm flipper_length_mm body_mass_g
##   <fct>   <fct>              <dbl>           <dbl>             <dbl>       <dbl>
## 1 Adelie  Torger~             39.1            18.7               181        3750
## 2 Adelie  Torger~             39.5            17.4               186        3800
## 3 Adelie  Torger~             40.3            18                 195        3250
## 4 Adelie  Torger~             36.7            19.3               193        3450
## 5 Adelie  Torger~             39.3            20.6               190        3650
## 6 Adelie  Torger~             38.9            17.8               181        3625
## # i 1 more variable: sex <fct>
```

We then split the data into training and testing set. The proportion of data to be split will be 75 percent for the training set and 25 percent for the testing set. Stratified sampling based on the species variable in order to help predict based on that variable. The cross validation that will be used will be 5 folds that are stratified on the species variable as well.

```
#splitting into testing and training set using strata

set.seed(123)

penguins_split <- initial_split(penguins, prop = 0.75, strata = "species")

penguins_train <- training(penguins_split)
penguins_test <- testing(penguins_split)

penguins_folds <- vfold_cv(penguins_train, v = 5, strata = "species")
```

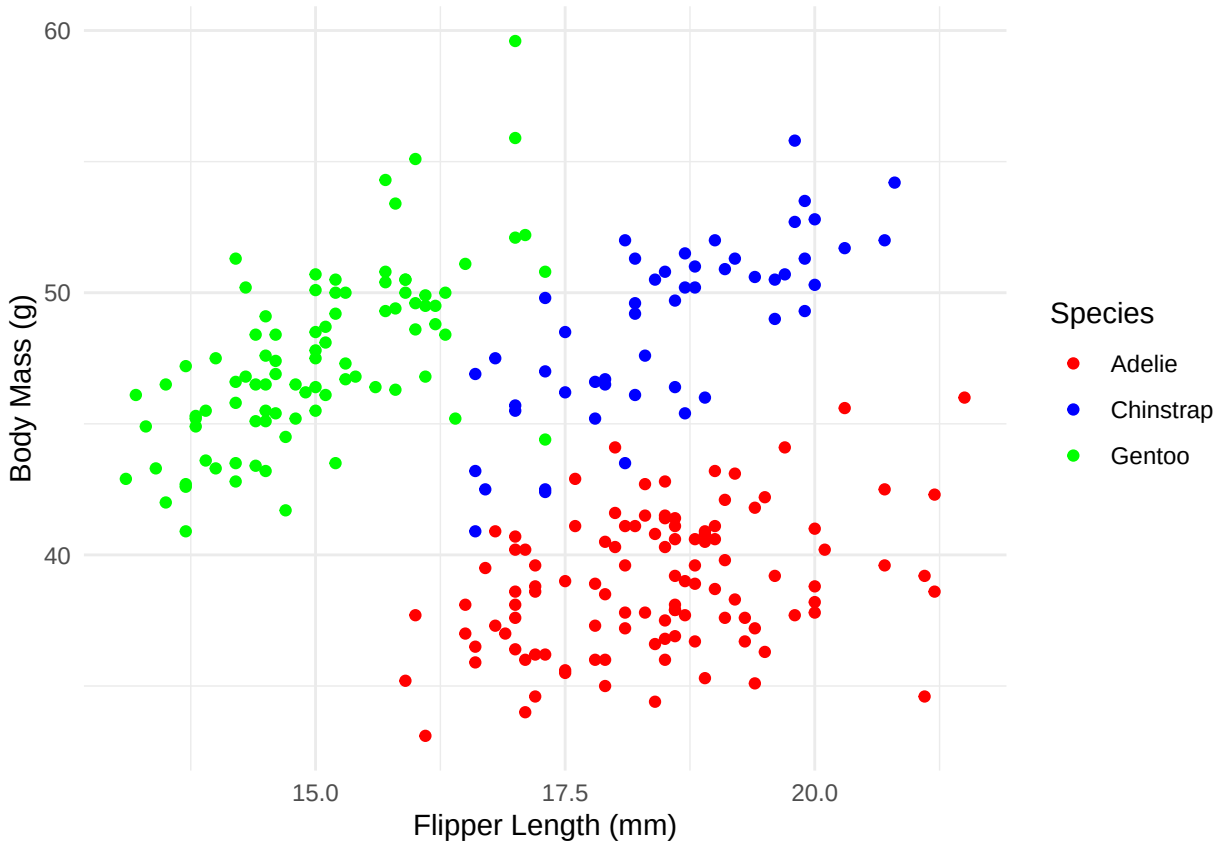
Below is a correlation matrix that will help determine any postive or negative correlations among the numeric variables. We see that there are postive correlations between flipper length and culmen length, body mass and culmen length, and flipper length and body mass. On the other hand, we see negative correlations between culmen length and culmen depth, flipper length and culmen depth, and body mass and culmen depth.

```r
#correlation of penguins to determine relationships

penguins_train %>%
  select(where(is.numeric)) %>%
  cor(use = "pairwise.complete.obs") %>%
  corrplot(type = "lower", diag = FALSE)
```



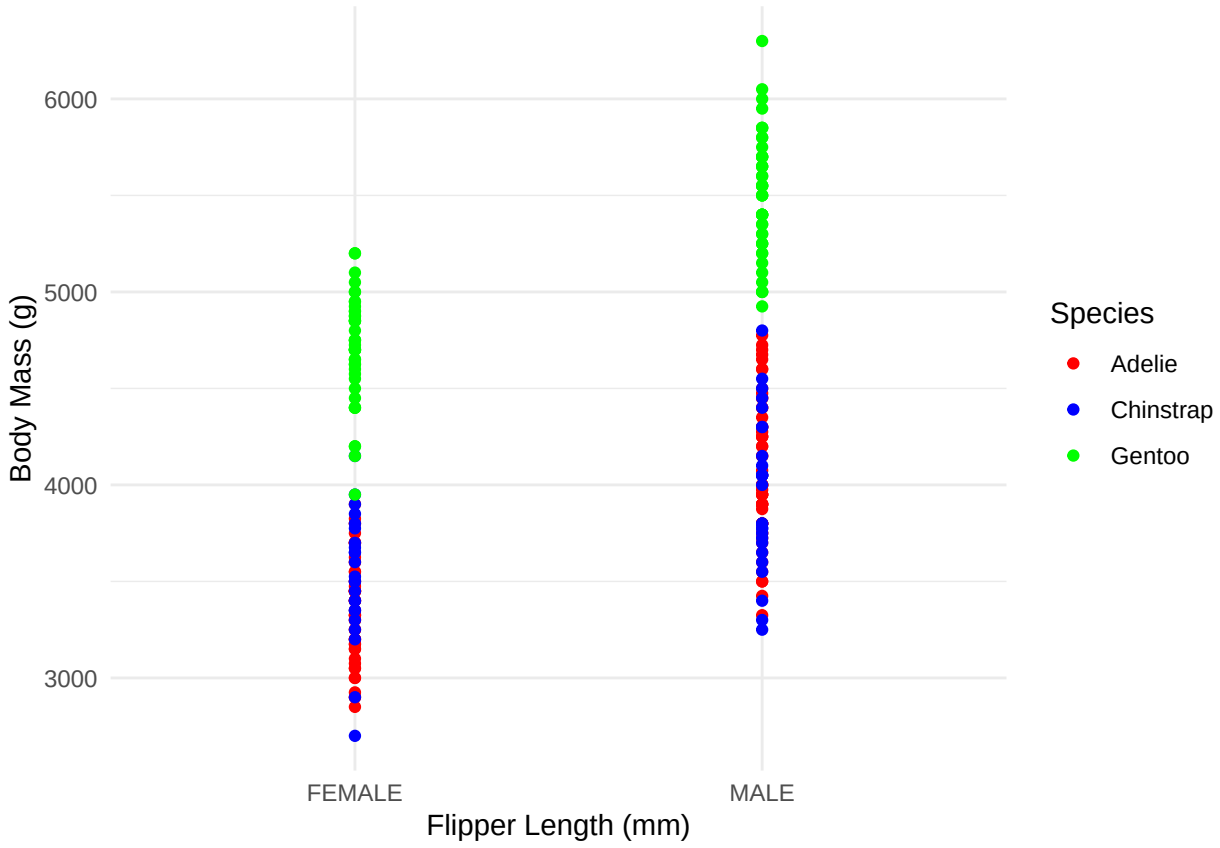
Going more in depth with the positive correlation, below is a plot analyzing the interaction between and flipper length and body mass based on the type of species. The observations on this plot is that the Chinstrap species contain more flipper length than the Gentoo species while having more body mass than the Adelie species. The body mass between the Gentoo relatively equal with the Chinstrap species while the flipper length between the Adelie and Chinstrap species are also relatively equal.

```r
#plot to correlate differences with culmen depth and length based on species


palette <- c("red", "blue", "green")

penguins_train %>%
  ggplot(aes(x = culmen_depth_mm, y = culmen_length_mm,
             color = species)) +
  geom_point() +
  scale_color_manual(values = palette) +
  theme_minimal() +
  xlab("Flipper Length (mm)") +
  ylab("Body Mass (g)") +
  labs(color = "Species")
```



In addition, sex must also be accounted for in order to predict the type of species the penguins are. Below is a plot that helps determine the relationship between Sex and a numeric variable, in this case will be body mass. We analyze that the Gentoo species between Male and Female contain higher body mass than either Adelie and Chinstrap species. Adelie and Chinstrap species are more equal to each other, with the Chinstrap female penguins having more penguins with higher body mass than Adelie female penguins. It is more even in the male plot with observations spread out.

```
#plot to correlate differences between sex and body mass based on species


penguins_train %>%
  ggplot(aes(x = sex, y = body_mass_g,
             color = species)) +
  geom_point() +
  scale_color_manual(values = palette) +
  theme_minimal() +
  xlab("Flipper Length (mm)") +
  ylab("Body Mass (g)") +
  labs(color = "Species")
```



Model Predictions

Having all of these positive and negative correlations taken into account, the recipe that will be used will be predicting species using all numeric variables. Categorical variables will be dummy variables and the recipe will have interactions between culmen length (mm) and culmen depth (mm), and flipper length (mm) and body mass (grams). All predictors will also be centered and scaled.

```
#recipe for prediction

penguins_recipe <- recipe(species ~ island + culmen_length_mm + culmen_depth_mm
                          + flipper_length_mm + body_mass_g + sex,
                          data = penguins_train) %>%
  step_dummy(all_nominal_predictors()) %>%
  step_interact(terms = ~starts_with("culmen_length_mm"):culmen_depth_mm) %>%
  step_interact(terms = ~flipper_length_mm:body_mass_g) %>%
  step_center(all_predictors()) %>%
  step_scale(all_predictors())
```

The four models that will be used to conduct the predictions are random forest, elastic net **logistic** regression, knn, and gradient boost trees. For the random forest, mtry, trees, and min_n will be tuned. For Elastic net logistic regression, mixture and penalty will be tuned. For knn, only neighbors will be tuned. Finally, for Gradient Boost Trees, mtry, trees, and the learn rate will be tuned. Below is the set up for the engines, mods, and workflows for all four models.

```
#models to be used to predict species: random forest,
#elastic net **logistic** regression, knn, and gradient boost trees

#random forest mod and workflow
```

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

rf_class_wf <- workflow() %>%
  add_model(rf_class) %>%
  add_recipe(penguins_recipe)

#elastic mod and workflow
elastic_mod_class <- multinom_reg(mixture = tune(),penalty = tune()) %>%
  set_mode("classification") %>%
  set_engine("glmnet")

elastic_wkflow_class <- workflow() %>%
  add_model(elastic_mod_class) %>%
  add_recipe(penguins_recipe)


#knn mod and workflow
knn_mod_class <- nearest_neighbor(neighbors = tune()) %>%
  set_mode("classification") %>%
  set_engine("kknn")

knn_wkflow_class <- workflow() %>%
  add_model(knn_mod_class) %>%
  add_recipe(penguins_recipe)


#gradient boost trees
bt_class_spec <- boost_tree(mtry = tune(),
                        trees = tune(),
                        learn_rate = tune()) %>%
  set_engine("xgboost") %>%
  set_mode("classification")

bt_class_wf <- workflow() %>%
  add_model(bt_class_spec) %>%
  add_recipe(penguins_recipe)
```

Below are the grid setups. For knn, the neighbors will be from a range of 1 to 10, conducting 10 levels of these grid. For Elastic Net, the penalty will be set to default, mixture will be from range 0 to 1, while conducting 10 levels in the grid. For random forest, mtry will be from 1 to 6 as that is the maximum range of the variables being analyzed, a range of 100 to 400 trees, min_n from a range of 10 to 20, while conducting 5 levels of these predictions. Finally, for Gradient Boost Trees, mtry will also be from 1 to 6 and 50 to 300 trees. The learn rate will be from a range of -10 to -1, with 5 levels of these predictions in the grid.

```
#grid and tuning

#grids
knn_grid <- grid_regular(neighbors(range= c(1,10)), levels = 10)

elastic_grid <- grid_regular(penalty(), mixture(range = c(0,1)), levels = 10)
```

```
rf_grid <- grid_regular(mtry(range = c(1, 6)),
                        trees(range = c(100, 400)),
                        min_n(range = c(10, 20)),
                        levels = 5)

bt_grid <- grid_regular(mtry(range = c(1, 6)),
                        trees(range = c(50, 300)),
                        learn_rate(range = c(-10, -1)),
                        levels = 5)
```

After creating the grids for each model, each model must be tuned using the workflows created prior, while adding the grids to each workflow and resamples from the cross validation in the beginning of the prediction process.

```
#tuning and collecting the metrics for each model and fitting to final workflow

#tuning

tune_knn_class <- tune_grid(object = knn_wkflow_class,
                      grid = knn_grid,
                      resamples = penguins_folds,
                      )

tune_elastic_class <- tune_grid(object = elastic_wkflow_class,
                          grid = elastic_grid,
                          resamples = penguins_folds,
                          )

tune_rf_class <- tune_grid(rf_class_wf,
                           resamples = penguins_folds,
                           grid = rf_grid)

tune_bt_class <- tune_grid(bt_class_wf,
                           resamples = penguins_folds,
                           grid = bt_grid)
save(tune_knn_class, file = "knn_class.rda")
save(tune_elastic_class, file = "elastic_class.rda")
save(tune_rf_class, file = "rf_class.rda")
save(tune_bt_class, file ="bt_class.rda")
```

Below are the roc-auc and accuracy metrics that are collected from each model after tuning the workflows:

```
load("knn_class.rda")
load("elastic_class.rda")
load("rf_class.rda")
load("bt_class.rda")

#metrics for each model
collect_metrics(tune_knn_class)
```

```
## # A tibble: 20 x 7
##    neighbors .metric  .estimator  mean     n std_err .config
##        <int> <chr>    <chr>      <dbl> <int>   <dbl> <chr>
##  1         1 accuracy multiclass 0.996     5 0.00408 Preprocessor1_Model01
##  2         1 roc_auc  hand_till  0.998     5 0.00238 Preprocessor1_Model01
```

```
##  3         2 accuracy multiclass 0.996     5 0.00408 Preprocessor1_Model02
##  4         2 roc_auc  hand_till  0.998     5 0.00238 Preprocessor1_Model02
##  5         3 accuracy multiclass 0.996     5 0.00408 Preprocessor1_Model03
##  6         3 roc_auc  hand_till  1         5 0       Preprocessor1_Model03
##  7         4 accuracy multiclass 0.996     5 0.00408 Preprocessor1_Model04
##  8         4 roc_auc  hand_till  1         5 0       Preprocessor1_Model04
##  9         5 accuracy multiclass 0.996     5 0.00408 Preprocessor1_Model05
## 10         5 roc_auc  hand_till  1         5 0       Preprocessor1_Model05
## 11         6 accuracy multiclass 0.996     5 0.00408 Preprocessor1_Model06
## 12         6 roc_auc  hand_till  1         5 0       Preprocessor1_Model06
## 13         7 accuracy multiclass 0.996     5 0.00408 Preprocessor1_Model07
## 14         7 roc_auc  hand_till  1         5 0       Preprocessor1_Model07
## 15         8 accuracy multiclass 0.996     5 0.00408 Preprocessor1_Model08
## 16         8 roc_auc  hand_till  1         5 0       Preprocessor1_Model08
## 17         9 accuracy multiclass 0.996     5 0.00408 Preprocessor1_Model09
## 18         9 roc_auc  hand_till  1         5 0       Preprocessor1_Model09
## 19        10 accuracy multiclass 0.992     5 0.00495 Preprocessor1_Model10
## 20        10 roc_auc  hand_till  1         5 0       Preprocessor1_Model10
```

```
collect_metrics(tune_elastic_class)
```

```
## # A tibble: 200 x 8
##          penalty mixture .metric  .estimator  mean     n std_err .config
##            <dbl>   <dbl> <chr>    <chr>      <dbl> <int>   <dbl> <chr>
##  1 0.0000000001        0 accuracy multiclass 0.996     5 0.00400 Preprocessor1_~
##  2 0.0000000001        0 roc_auc  hand_till  1         5 0       Preprocessor1_~
##  3 0.00000000129       0 accuracy multiclass 0.996     5 0.00400 Preprocessor1_~
##  4 0.00000000129       0 roc_auc  hand_till  1         5 0       Preprocessor1_~
##  5 0.0000000167        0 accuracy multiclass 0.996     5 0.00400 Preprocessor1_~
##  6 0.0000000167        0 roc_auc  hand_till  1         5 0       Preprocessor1_~
##  7 0.000000215         0 accuracy multiclass 0.996     5 0.00400 Preprocessor1_~
##  8 0.000000215         0 roc_auc  hand_till  1         5 0       Preprocessor1_~
##  9 0.00000278          0 accuracy multiclass 0.996     5 0.00400 Preprocessor1_~
## 10 0.00000278          0 roc_auc  hand_till  1         5 0       Preprocessor1_~
## # i 190 more rows
```

```
collect_metrics(tune_rf_class)
```

```
## # A tibble: 250 x 9
##     mtry trees min_n .metric  .estimator  mean     n  std_err .config
##    <int> <int> <int> <chr>    <chr>      <dbl> <int>    <dbl> <chr>
##  1     1   100    10 accuracy multiclass 0.996     5 0.00400  Preprocessor1_Mod~
##  2     1   100    10 roc_auc  hand_till  0.999     5 0.000909 Preprocessor1_Mod~
##  3     2   100    10 accuracy multiclass 0.992     5 0.00490  Preprocessor1_Mod~
##  4     2   100    10 roc_auc  hand_till  0.999     5 0.00121  Preprocessor1_Mod~
##  5     3   100    10 accuracy multiclass 0.980     5 0.00904  Preprocessor1_Mod~
##  6     3   100    10 roc_auc  hand_till  0.998     5 0.00194  Preprocessor1_Mod~
##  7     4   100    10 accuracy multiclass 0.984     5 0.00749  Preprocessor1_Mod~
##  8     4   100    10 roc_auc  hand_till  0.999     5 0.00106  Preprocessor1_Mod~
##  9     6   100    10 accuracy multiclass 0.980     5 0.0110   Preprocessor1_Mod~
## 10     6   100    10 roc_auc  hand_till  0.998     5 0.00179  Preprocessor1_Mod~
## # i 240 more rows
```

```
collect_metrics(tune_bt_class)
```

```
## # A tibble: 250 x 9
```

```
##     mtry trees   learn_rate .metric  .estimator  mean     n std_err .config
##    <int> <int>        <dbl> <chr>    <chr>      <dbl> <int>   <dbl> <chr>
##  1     1    50 0.0000000001 accuracy multiclass 0.436     5 0.00250 Preprocesso~
##  2     1    50 0.0000000001 roc_auc  hand_till  0.5       5 0       Preprocesso~
##  3     1   112 0.0000000001 accuracy multiclass 0.436     5 0.00250 Preprocesso~
##  4     1   112 0.0000000001 roc_auc  hand_till  0.5       5 0       Preprocesso~
##  5     1   175 0.0000000001 accuracy multiclass 0.436     5 0.00250 Preprocesso~
##  6     1   175 0.0000000001 roc_auc  hand_till  0.5       5 0       Preprocesso~
##  7     1   237 0.0000000001 accuracy multiclass 0.436     5 0.00250 Preprocesso~
##  8     1   237 0.0000000001 roc_auc  hand_till  0.5       5 0       Preprocesso~
##  9     1   300 0.0000000001 accuracy multiclass 0.436     5 0.00250 Preprocesso~
## 10     1   300 0.0000000001 roc_auc  hand_till  0.5       5 0       Preprocesso~
## # i 240 more rows
```

Below are the best models that were conducting after tuning the workflows using the grids and resamplings from the cross validation. These best values will be used in order to finalize the workflow for each model.

```
show_best(tune_knn_class)
```

```
## # A tibble: 5 x 7
##   neighbors .metric .estimator  mean     n std_err .config
##       <int> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
## 1         3 roc_auc hand_till      1     5       0 Preprocessor1_Model03
## 2         4 roc_auc hand_till      1     5       0 Preprocessor1_Model04
## 3         5 roc_auc hand_till      1     5       0 Preprocessor1_Model05
## 4         6 roc_auc hand_till      1     5       0 Preprocessor1_Model06
## 5         7 roc_auc hand_till      1     5       0 Preprocessor1_Model07
```

```
show_best(tune_elastic_class)
```

```
## # A tibble: 5 x 8
##         penalty mixture .metric .estimator  mean     n std_err .config
##           <dbl>   <dbl> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
## 1 0.0000000001        0 roc_auc hand_till      1     5       0 Preprocessor1_Mo~
## 2 0.00000000129       0 roc_auc hand_till      1     5       0 Preprocessor1_Mo~
## 3 0.0000000167        0 roc_auc hand_till      1     5       0 Preprocessor1_Mo~
## 4 0.000000215         0 roc_auc hand_till      1     5       0 Preprocessor1_Mo~
## 5 0.00000278          0 roc_auc hand_till      1     5       0 Preprocessor1_Mo~
```

```
show_best(tune_rf_class)
```

```
## # A tibble: 5 x 9
##    mtry trees min_n .metric .estimator  mean     n  std_err .config
##   <int> <int> <int> <chr>   <chr>      <dbl> <int>    <dbl> <chr>
## 1     1   400    10 roc_auc hand_till  0.999     5 0.000758 Preprocessor1_Model~
## 2     1   100    10 roc_auc hand_till  0.999     5 0.000909 Preprocessor1_Model~
## 3     1   175    10 roc_auc hand_till  0.999     5 0.000909 Preprocessor1_Model~
## 4     1   250    10 roc_auc hand_till  0.999     5 0.000909 Preprocessor1_Model~
## 5     4   250    10 roc_auc hand_till  0.999     5 0.000909 Preprocessor1_Model~
```

```
show_best(tune_bt_class)
```

```
## # A tibble: 5 x 9
##    mtry trees learn_rate .metric .estimator  mean     n  std_err .config
##   <int> <int>      <dbl> <chr>   <chr>      <dbl> <int>    <dbl> <chr>
## 1     1   112        0.1 roc_auc hand_till  0.999     5 0.000909 Preprocessor1_~
## 2     1   175        0.1 roc_auc hand_till  0.999     5 0.000909 Preprocessor1_~
```

```
## 3     1   237        0.1 roc_auc hand_till  0.999     5 0.000909 Preprocessor1_~
## 4     1   300        0.1 roc_auc hand_till  0.999     5 0.000909 Preprocessor1_~
## 5     2   112        0.1 roc_auc hand_till  0.999     5 0.000909 Preprocessor1_~
```

After storing each best models in its each indivudal assignment, it is imperative we make sure that the metric being used is only roc-auc. Otherwise, R will be confused in trying to figure out with metric to use in order to finalize the workflow. We then finalize out workflows for each model shown below:

```
#best model of each tuning

best_knn <- select_best(tune_knn_class, metric = "roc_auc")
best_elastic<- select_best(tune_elastic_class, metric = "roc_auc")
best_rf <- select_best(tune_rf_class, metric = "roc_auc")
best_bt <- select_best(tune_bt_class, metric = "roc_auc")

#finalizing workflows
final_wkflow_knn <- finalize_workflow(knn_wkflow_class, best_knn)
final_wkflow_elastic <- finalize_workflow(elastic_wkflow_class, best_elastic)
final_wkflow_rf <- finalize_workflow(rf_class_wf, best_rf)
final_wkflow_bt <- finalize_workflow(bt_class_wf, best_bt)
```

After finalizing the workflows, the workflows are then fitted into the training dataset:

```
#fitting workflows

fitted_final_wkflow_knn <- fit(final_wkflow_knn, data = penguins_train)
fitted_final_wkflow_elastic <- fit(final_wkflow_elastic, data = penguins_train)

fitted_final_wkflow_rf <- fit(final_wkflow_rf, data = penguins_train)
fitted_final_wkflow_bt <- fit(final_wkflow_bt, data = penguins_train)
```

After fitting the finalized workflows, they will finally be fitted now using the testing dataset in order to determine which model or two models predicting the outcome of species the best.

```
#fitting all models into testing set and choosing best one or two models

testing_bt <- augment(fitted_final_wkflow_bt, new_data = penguins_test) %>%
    roc_auc(truth = species, .pred_Adelie:.pred_Gentoo)

testing_elastic<-augment(fitted_final_wkflow_elastic, new_data = penguins_test) %>%
  roc_auc(truth = species, .pred_Adelie:.pred_Gentoo)

testing_knn <- augment(fitted_final_wkflow_knn, new_data = penguins_test) %>%
  roc_auc(truth = species, .pred_Adelie:.pred_Gentoo)

testing_rf <- augment(fitted_final_wkflow_rf, new_data = penguins_test) %>%
  roc_auc(truth = species, .pred_Adelie:.pred_Gentoo)

testing_bt
```

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

```
testing_elastic
```

```
## # A tibble: 1 x 3
```

```
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 roc_auc hand_till          1
```

```
testing_knn
```

```
## # A tibble: 1 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 roc_auc hand_till      0.993
```

```
testing_rf
```

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

Conclusion

- We see that the the ROC-AUC for Random Forest, Elastic Net Logistic Regression, and Boost Trees had no problem in predicting the species outcome of the penguins as it was 1. KNN also had a great score but was the least with a score of 0.993. I am honestly not surpised that the models did really well as the postive interactions played a significant role in determining which observations were part of an specific species. I would suggest doing Neural Networks and SVMs in order to further evaluate the data set and whether or not having different interactions of the data set would effect the outcome of the machine learning algorithms. In addition, this data set can also be analyzed through other aspects such as body mass, type_1, and other variables that the data set can offered, which is also another alternative in helping further evaluate machine learning algorithms.