Problem Set 9

Classification

[YOUR NAME]

Due Date: 2024-03-29

Getting Set Up

Open `RStudio` and create a new RMarkDown file (`.Rmd`) by going to `File -> New File -> R Markdown...` . Accept defaults and save this file as `[LAST NAME]_ps9.Rmd` to your `code` folder.

Copy and paste the contents of this `.Rmd` file into your `[LAST NAME]_ps9.Rmd` file. Then change the `author: [Your Name]` to your name.

We will be using the `fn_cleaned_final.Rds` file from the course github page (https://github.com/jbisbee1/DS1000_S2024/raw/main/data/fn_cleaned_final.rds).

All of the following questions should be answered in this `.Rmd` file. There are code chunks with incomplete code that need to be filled in.

This problem set is worth 8 total points, plus two extra credit points. The point values for each question are indicated in brackets below. To receive full credit, you must have the correct code. In addition, some questions ask you to provide a written response in addition to the code.

You are free to rely on whatever resources you need to complete this problem set, including lecture notes, lecture presentations, Google, your classmates…you name it. However, the final submission must be complete by you. There are no group assignments. To submit, compiled the completed problem set and upload the PDF file to Brightspace on Friday by midnight. Also note that the TAs and professors will not respond to Campuswire posts after 5PM on Friday, so don't wait until the last minute to get started!

Good luck!

*Copy the link to ChatGPT you used here: _____

Question 0

Require `tidyverse` and load the `fn_cleaned_final.Rds` (https://github.com/jbisbee1/DS1000_S2024/raw/main/data/fn_cleaned_final.rds) data to an object called `fn` .

```
require(tidyverse)
```

```
## Loading required package: tidyverse
```

```
## Warning: package 'ggplot2' was built under R version 4.3.3
```

```
## Warning: package 'purrr' was built under R version 4.3.3
```

```
## Warning: package 'stringr' was built under R version 4.3.3
```

```
## ── Attaching core tidyverse packages ──────────────────────── tidyverse 2.0.0 ──
## ✓ dplyr     1.1.2     ✓ readr     2.1.4
## ✓ forcats   1.0.0     ✓ stringr   1.5.1
## ✓ ggplot2   3.5.0     ✓ tibble    3.2.1
## ✓ lubridate 1.9.2     ✓ tidyr     1.3.0
## ✓ purrr     1.0.2
```

```
## ── Conflicts ────────────────────────────────────────── tidyverse_conflicts() ──
## ✖ dplyr::filter() masks stats::filter()
## ✖ dplyr::lag()    masks stats::lag()
## ℹ Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to becom
e errors
```

```
fn <- read_rds('https://github.com/jbisbee1/DS1000_S2024/raw/main/data/fn_cleaned_final.rds')
```

Question 1 [2 points]

Let's consider two possible X variables which might help us predict whether a player wins a Fortnite match: `revives` and `eliminations`. `revives` counts the total number of times a player is brought back to life by a teammate. `eliminations` is a measure of how many times the player killed an opponent. Which variable do you think is more helpful for predicting whether a player wins a game of Fortnite? Why?

> I think revives should be more helpful than eliminations I think this because revives mean you stay alive longer, whereas eliminations can be high even if the player gets lucky. Since winning the game is a matter of surviving until the end, I think revives should be a stronger predictor.

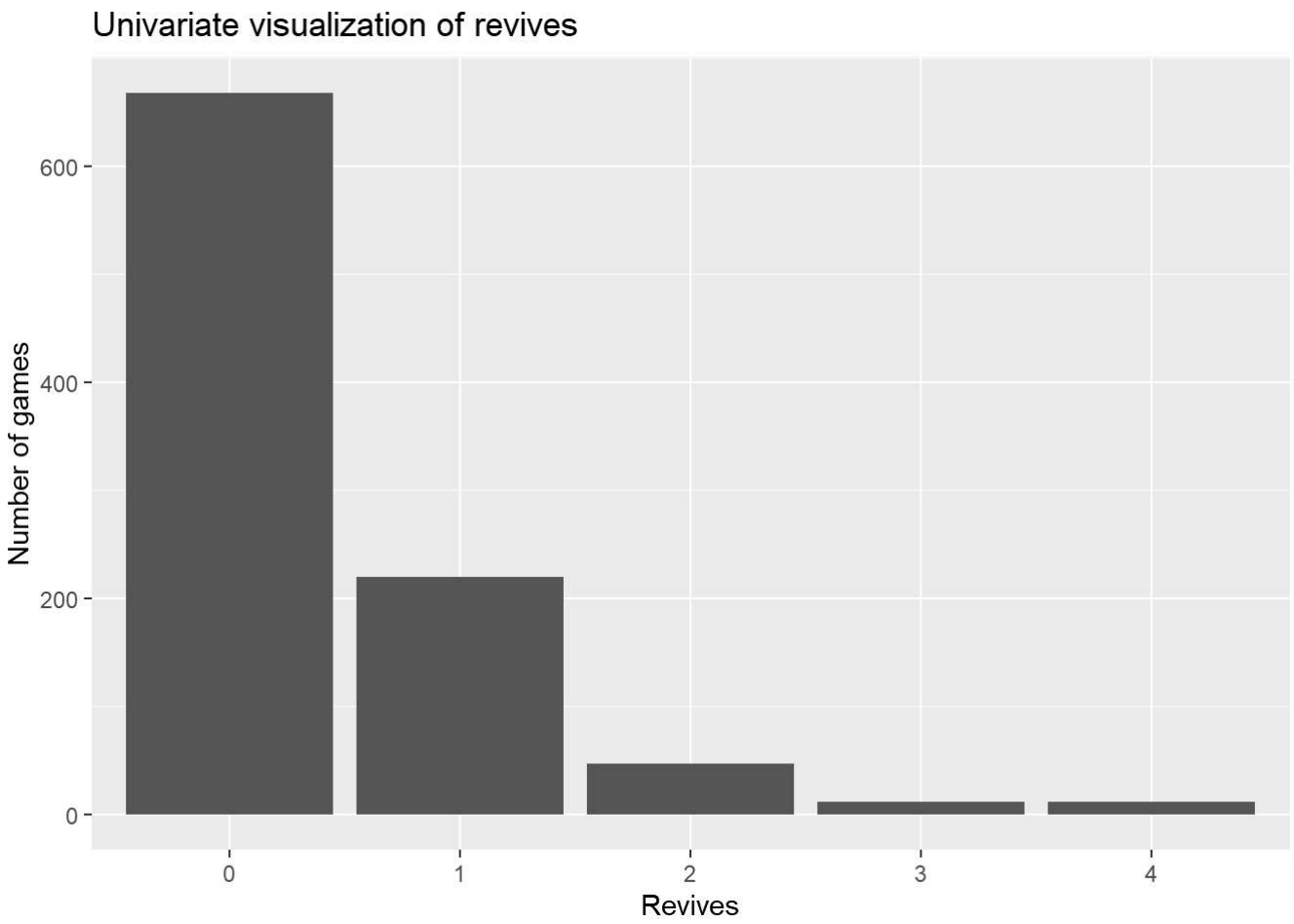
Question 2 [2 points]

Look at the data and provide univariate and multivariate visualizations of both variables. Make sure to think carefully about what types of variables these are, and justify your visualization choices accordingly!

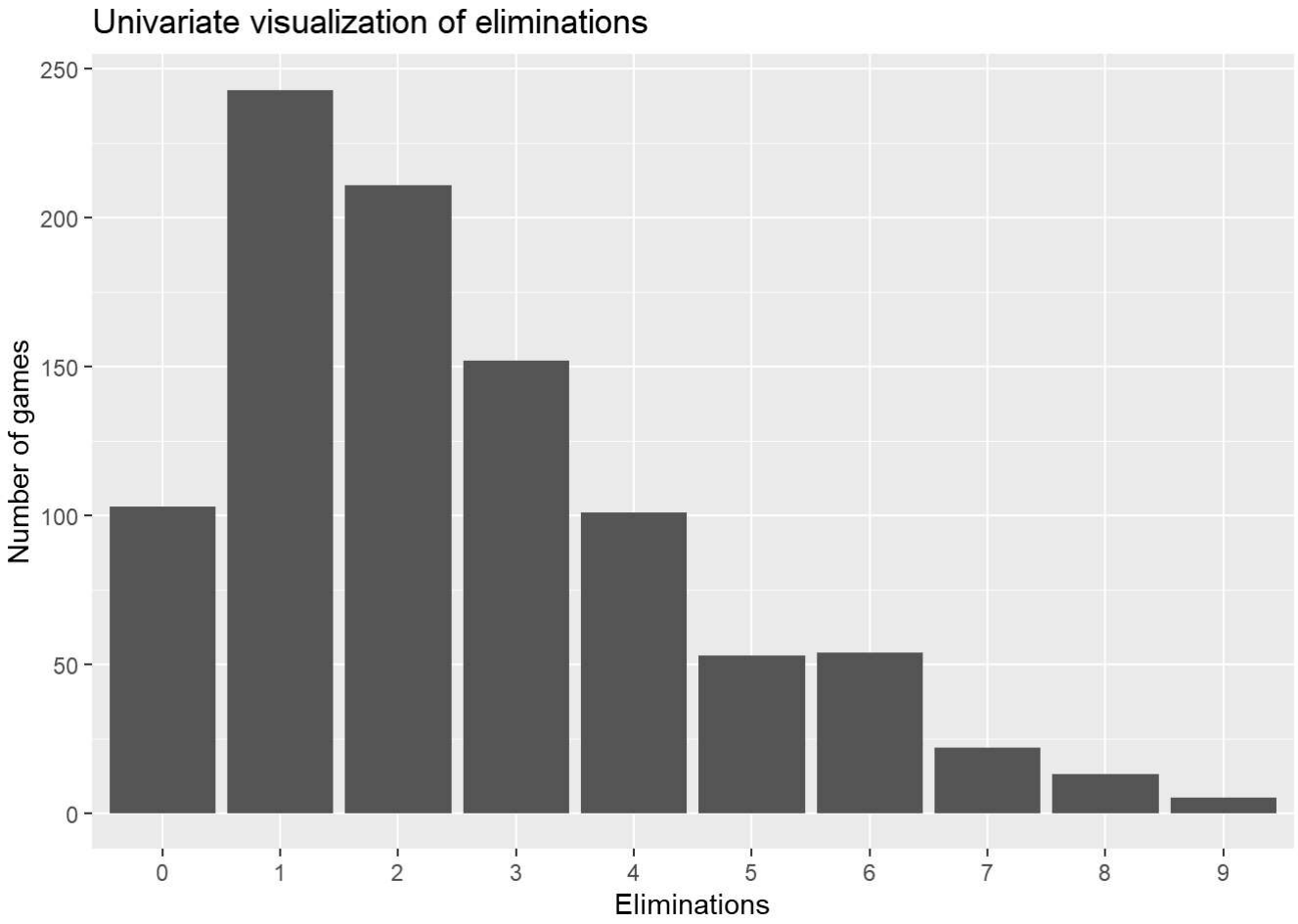
```
fn %>%
  select(revives,eliminations)
```

```
## # A tibble: 957 × 2
##    revives eliminations
##      <dbl>        <dbl>
##  1       0            2
##  2       0            0
##  3       0            3
##  4       0            1
##  5       1            3
##  6       0            0
##  7       0            2
##  8       0            3
##  9       1            4
## 10       0            1
## # ℹ 947 more rows
```

```
# Univariate
fn %>%
  ggplot(aes(x = factor(revives))) +
  geom_bar() +
  labs(x = 'Revives',y = 'Number of games',
       title = 'Univariate visualization of revives')
```

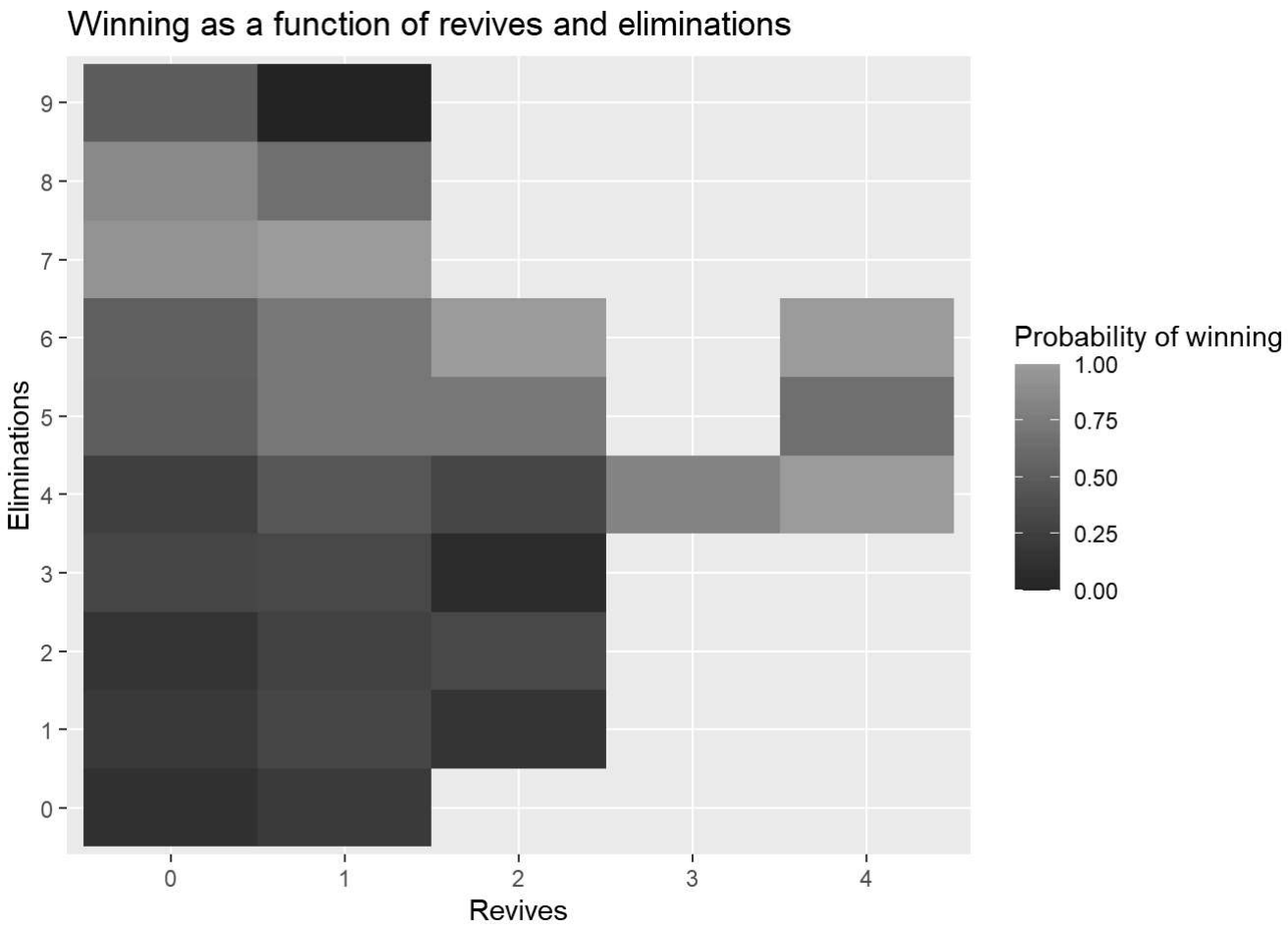


```
fn %>%
  ggplot(aes(x = factor(eliminations))) +
  geom_bar() +
  labs(x = 'Eliminations',y = 'Number of games',
       title = 'Univariate visualization of eliminations')
```

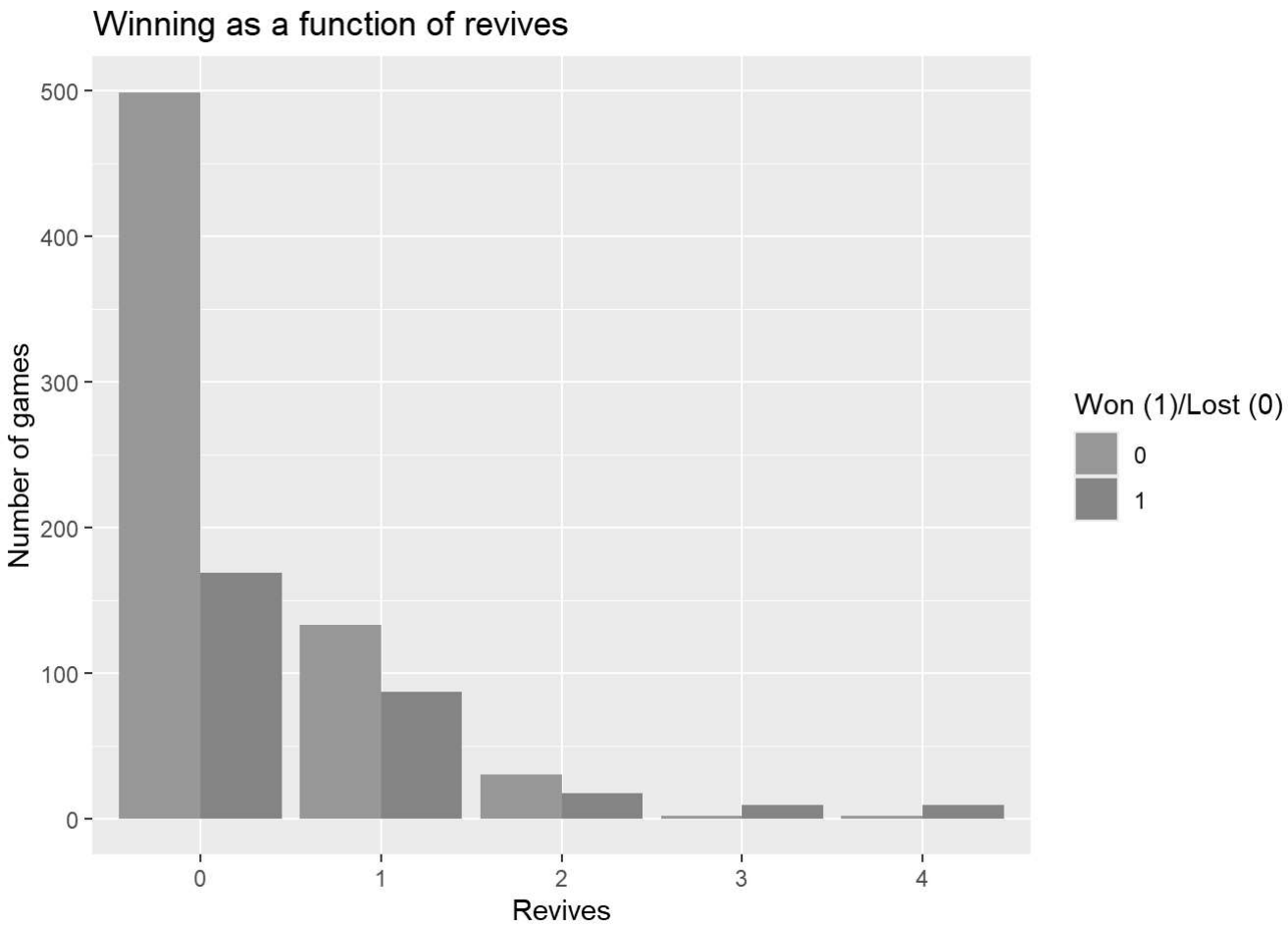



```
# Multivariate options (all of these are good)
fn %>%
  group_by(revives,eliminations) %>%
  summarise(prob_win = mean(won)) %>%
  ggplot(aes(x = factor(revives),y = factor(eliminations),fill = prob_win)) +
  geom_tile() +
  labs(x = 'Revives',
       y = 'Eliminations',
       fill = 'Probability of winning',
       title = 'Winning as a function of revives and eliminations')
```

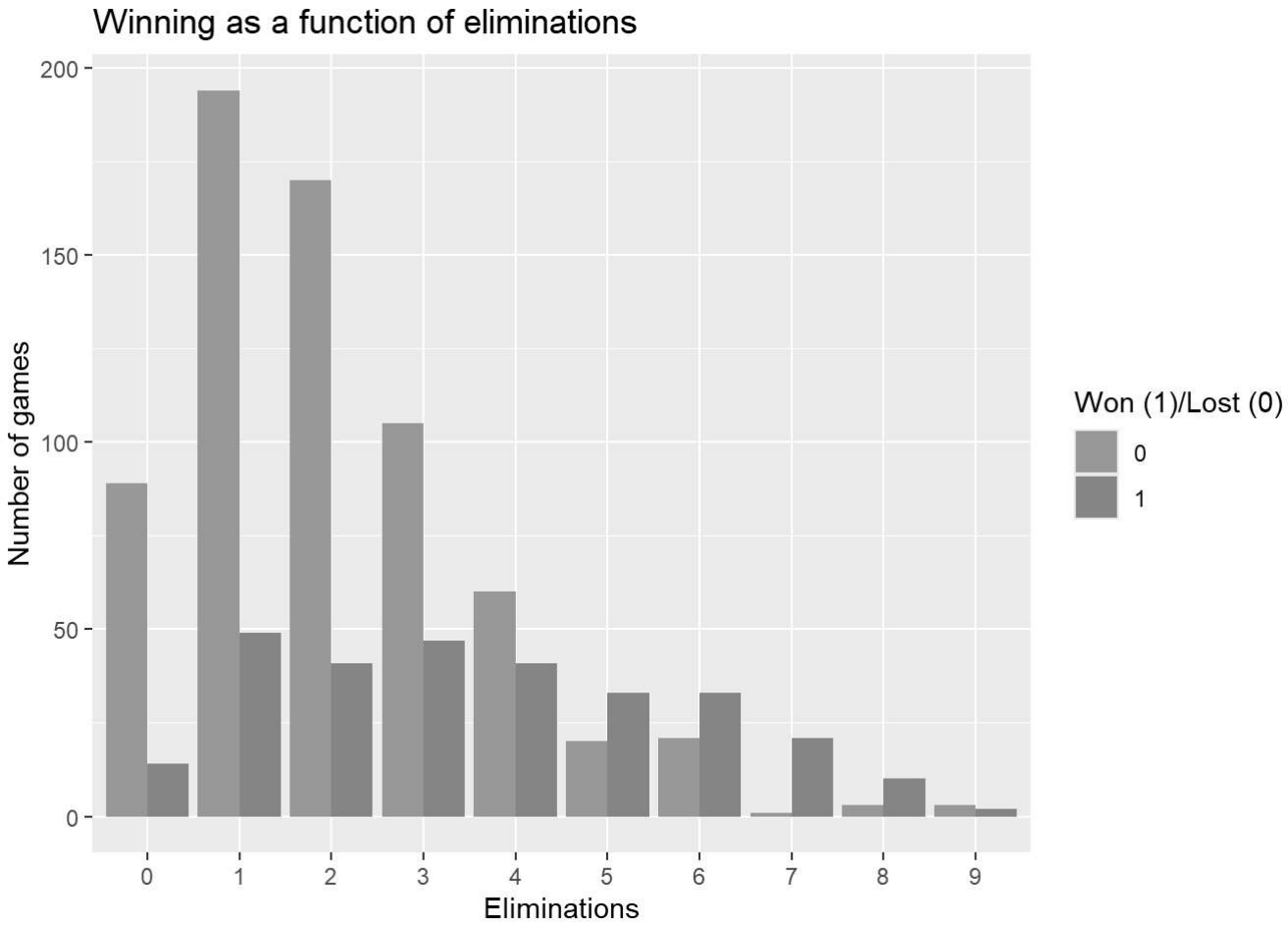
```
## `summarise()` has grouped output by 'revives'. You can override using the
## `.groups` argument.
```



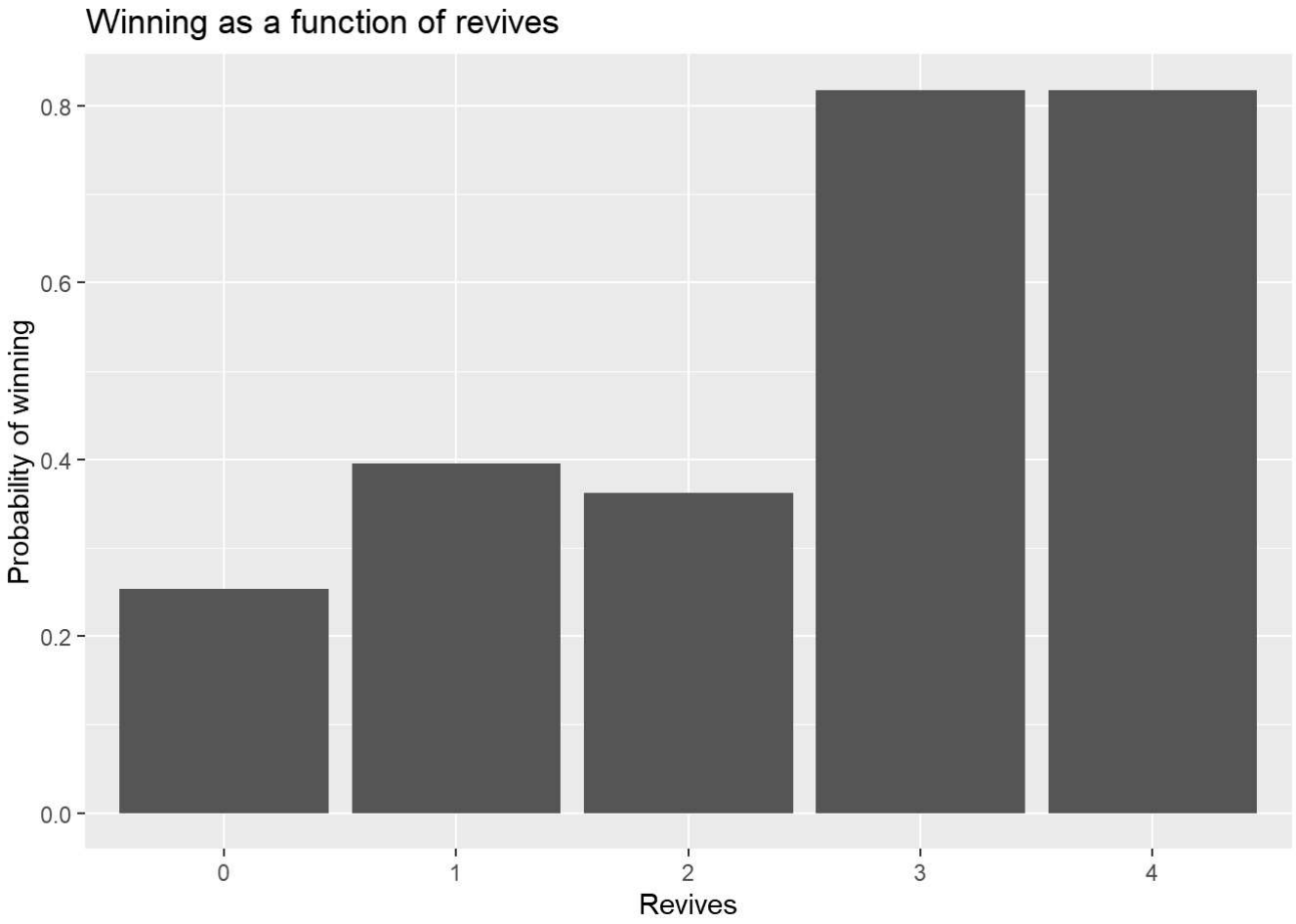

```
fn %>%
  ggplot(aes(x = factor(revives),
             fill = factor(won))) +
  geom_bar(position = 'dodge') +
  labs(x = 'Revives',
       y = 'Number of games',
       fill = 'Won (1)/Lost (0)',
       title = 'Winning as a function of revives')
```



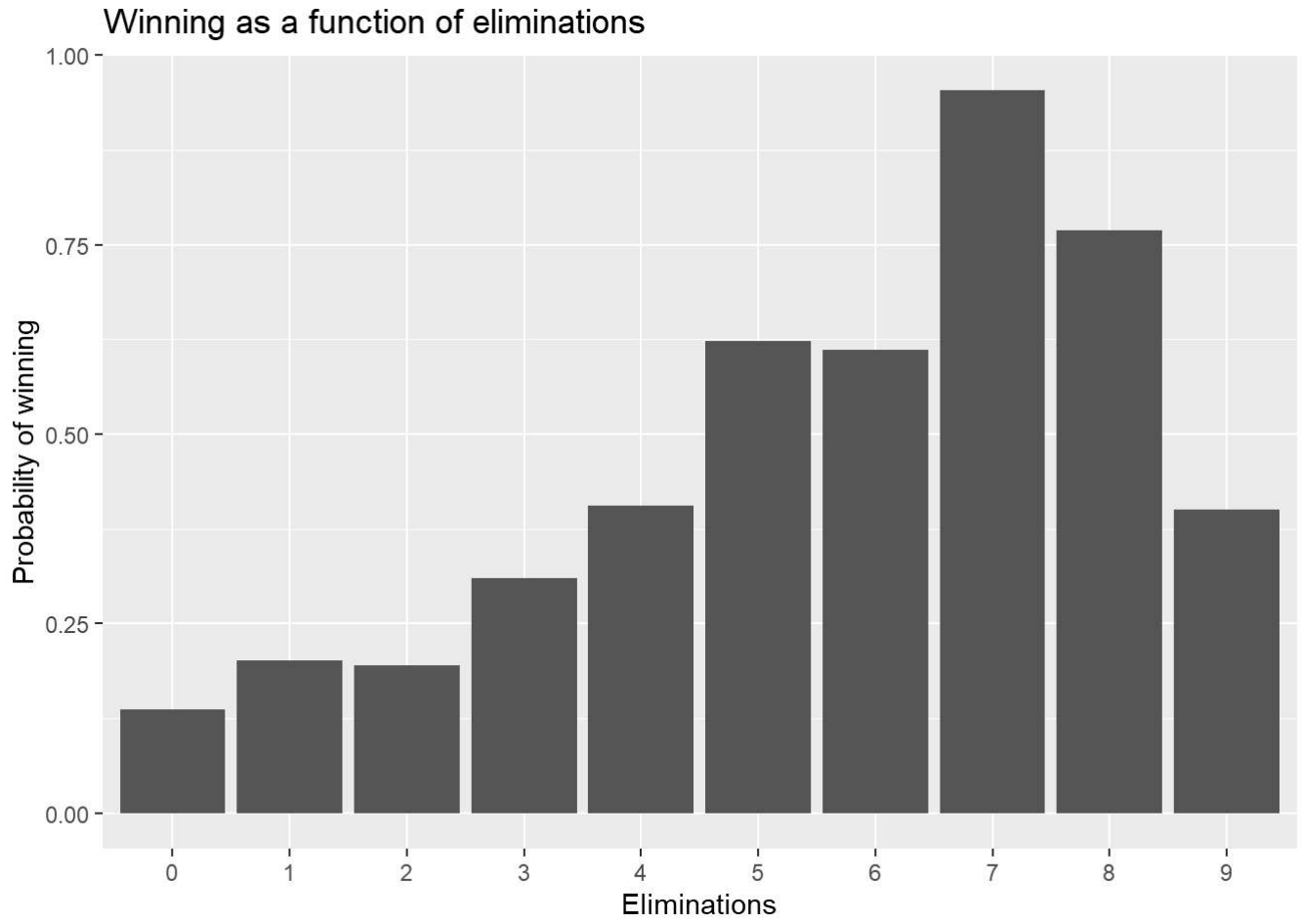

```
fn %>%
  ggplot(aes(x = factor(eliminations),
             fill = factor(won))) +
  geom_bar(position = 'dodge') +
  labs(x = 'Eliminations',
       y = 'Number of games',
       fill = 'Won (1)/Lost (0)',
       title = 'Winning as a function of eliminations')
```




```
fn %>%
  group_by(revives) %>%
  summarise(prob_win = mean(won)) %>%
  ggplot(aes(x = factor(revives),y = prob_win)) +
  geom_bar(stat = 'identity') +
  labs(x = 'Revives',
       y = 'Probability of winning',
       title = 'Winning as a function of revives')
```




```
fn %>%
  group_by(eliminations) %>%
  summarise(prob_win = mean(won)) %>%
  ggplot(aes(x = factor(eliminations),y = prob_win)) +
  geom_bar(stat = 'identity') +
  labs(x = 'Eliminations',
       y = 'Probability of winning',
       title = 'Winning as a function of eliminations')
```



Question 3 [2 points]

Let's test your intuition. Using 100 cross validation with a logit model and a 60-40 split, calculate the AUC for the model which uses the variable you think is best, compared to the model you think is the worst. Pay attention to the things you need to change to use a logit model! Is your assumption from Q1 supported in the data?

```
require(tidymodels) # Require the tidymodels package
```

```
## Loading required package: tidymodels
```

```
## Warning: package 'tidymodels' was built under R version 4.3.3
```

```
## ── Attaching packages ─────────────────────────────────────── tidymodels 1.1.1 ──
```

```
## ✓ broom        1.0.5      ✓ rsample      1.2.0
## ✓ dials        1.2.1      ✓ tune         1.1.2
## ✓ infer        1.0.6      ✓ workflows    1.1.4
## ✓ modeldata    1.3.0      ✓ workflowsets 1.0.1
## ✓ parsnip      1.2.0      ✓ yardstick    1.3.0
## ✓ recipes      1.0.10
```

```
## Warning: package 'dials' was built under R version 4.3.3
```

```
## Warning: package 'scales' was built under R version 4.3.3
```

```
## Warning: package 'infer' was built under R version 4.3.3
```

```
## Warning: package 'modeldata' was built under R version 4.3.3
```

```
## Warning: package 'parsnip' was built under R version 4.3.3
```

```
## Warning: package 'recipes' was built under R version 4.3.3
```

```
## Warning: package 'rsample' was built under R version 4.3.3
```

```
## Warning: package 'tune' was built under R version 4.3.3
```

```
## Warning: package 'workflows' was built under R version 4.3.3
```

```
## Warning: package 'workflowsets' was built under R version 4.3.3
```

```
## Warning: package 'yardstick' was built under R version 4.3.3
```

```
## ── Conflicts ──────────────────────────────────────── tidymodels_conflicts() ──
## ✖ scales::discard() masks purrr::discard()
## ✖ dplyr::filter()   masks stats::filter()
## ✖ recipes::fixed()  masks stringr::fixed()
## ✖ dplyr::lag()      masks stats::lag()
## ✖ yardstick::spec() masks readr::spec()
## ✖ recipes::step()   masks stats::step()
## • Use tidymodels_prefer() to resolve common conflicts.
```

```
# Running logit model
model_r <- glm(formula = won ~ revives, # Define the regression equation
               data = fn,family = binomial(link = 'logit')) # Provide the dataset

model_e <- glm(formula = won ~ eliminations, # Define the regression equation
               data = fn,family = binomial(link = 'logit')) # Provide the dataset

# Calculate the AUC
forAUCBest <- fn %>%
  mutate(prob_win = predict(model_r,type = 'response'), # Generate predicted probabilities of wi
nning from our model
         truth = factor(won,levels = c('1','0'))) %>% # Convert the outcome to a factor with lev
els c('1','0')
  select(truth,prob_win) # Select only the probability and true outcome columns

roc_auc(data = forAUCBest, # Run the roc_auc() function on the dataset we just created
        truth, # Tell it which column contains the true outcomes
        prob_win) # Tell it which column contains our model's predicted probabilities
```

```
## # A tibble: 1 × 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 roc_auc binary         0.590
```

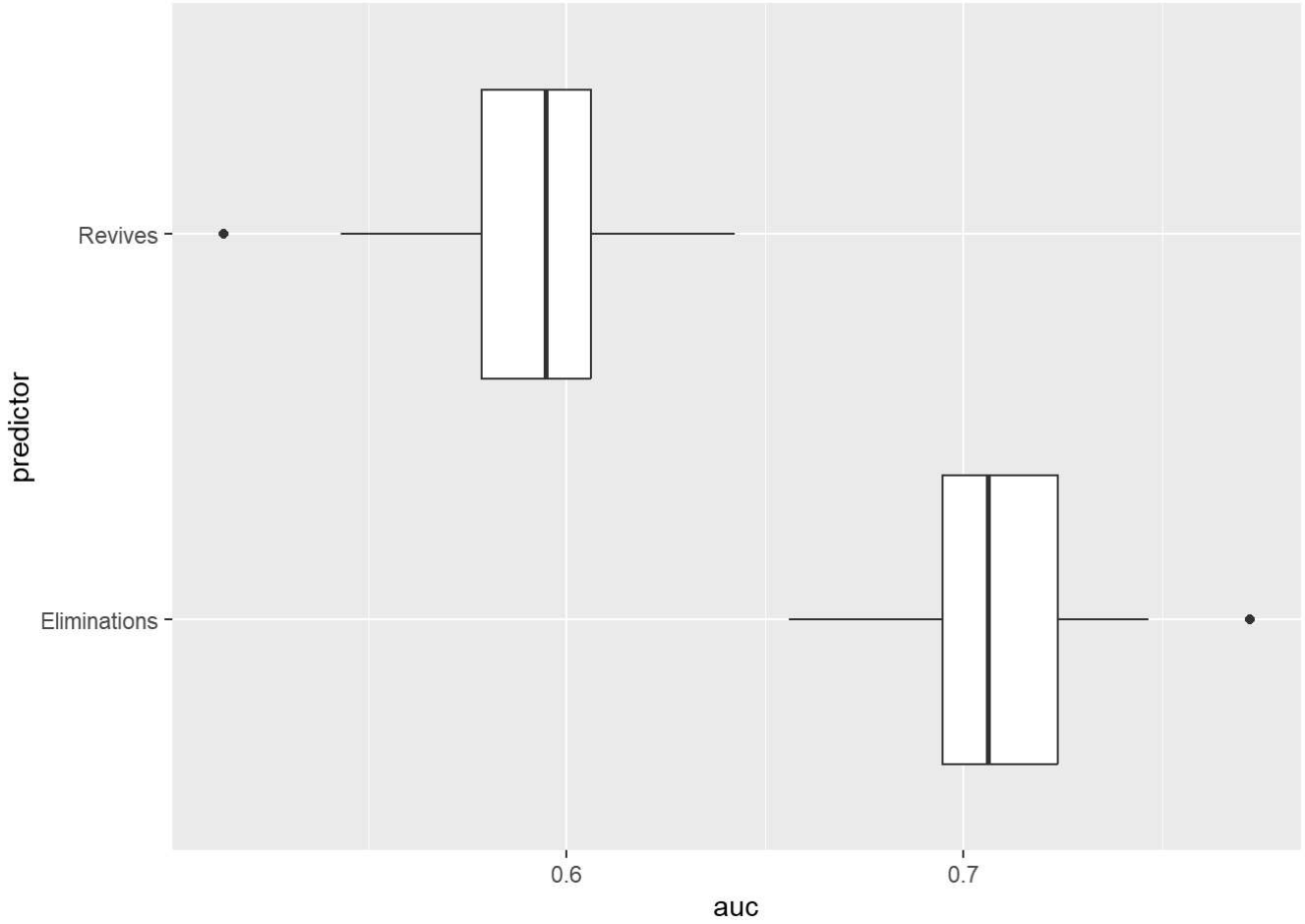
```
forAUCWorst <- fn %>%
  mutate(prob_win = predict(model_e,type = 'response'), # Generate predicted probabilities of wi
nning from our model
         truth = factor(won,levels = c('1','0'))) %>% # Convert the outcome to a factor with lev
els c('1','0')
  select(truth,prob_win) # Select only the probability and true outcome columns

roc_auc(data = forAUCWorst, # Run the roc_auc() function on the dataset we just created
        truth, # Tell it which column contains the true outcomes
        prob_win) # Tell it which column contains our model's predicted probabilities
```

```
## # A tibble: 1 × 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 roc_auc binary         0.705
```

```
set.seed(123)
cvRes <- NULL
for(i in 1:100) {
  # Cross validation prep
  inds <- sample(1:nrow(fn),size = round(nrow(fn)*.6),replace = F)
  train <- fn %>% slice(inds)
  test <- fn %>% slice(-inds)

  # Training models
  mR <- glm(won ~ revives,train,family = binomial(link = 'logit'))
  mE <- glm(won ~ eliminations,train,family = binomial(link = 'logit'))

  # Predicting models
  toEval <- test %>%
    mutate(mRPreds = predict(mR,newdata = test,type = 'response'), # Calculate the probability o
f winning from the revives predictor
           mEPreds = predict(mE,newdata = test,type = 'response'), # Calculate the probability o
f winning from the eliminations predictor
           truth = factor(won,levels = c('1','0'))) # Convert the outcome to a factor with level
s c('1','0')

  # Evaluating models
  rocB <- roc_auc(toEval,truth,mRPreds) %>% # Calculate the AUC for the best predictor
    mutate(predictor = "Revives") %>% # Record which predictor this is
    rename(auc = .estimate) # Rename to 'auc'

  rocW <- roc_auc(toEval,truth,mEPreds) %>% # Calculate the AUC for the worst predictor
    mutate(predictor = "Eliminations") %>% # Record which predictor this is
    rename(auc = .estimate) # Rename to 'auc'

  cvRes <- rocB %>%
    bind_rows(rocW) %>%
    mutate(cvInd = i) %>%
    bind_rows(cvRes)
}

cvRes %>%
  group_by(predictor) %>%
  summarise(mean_auc = mean(auc))
```

```
## # A tibble: 2 × 2
##   predictor    mean_auc
##   <chr>           <dbl>
## 1 Eliminations    0.707
## 2 Revives         0.591
```

```
cvRes %>%
  ggplot(aes(x = auc,y = predictor)) +
  geom_boxplot()
```




```
# Proportion of time the "best" model is better than the "worst"
cvRes %>%
  spread(predictor,auc) %>%
  summarise(mean(Revives > Eliminations))
```

```
## # A tibble: 1 × 1
##   `mean(Revives > Eliminations)`
##                            <dbl>
## 1                              0
```

No my assumption is not supported in the data. Eliminations are in fact way better at predicting winning or losing than revives.

Question 4 [2 points]

Now let's run a kitchen sink model using a random forest. Use the following X variables: - `hits` - `assists` - `accuracy` - `head_shots` - `damage_to_players` - `eliminations` - `revives` - `distance_traveled` - `materials_gathered` - `mental_state` - `startTime` - `gameIdSession`

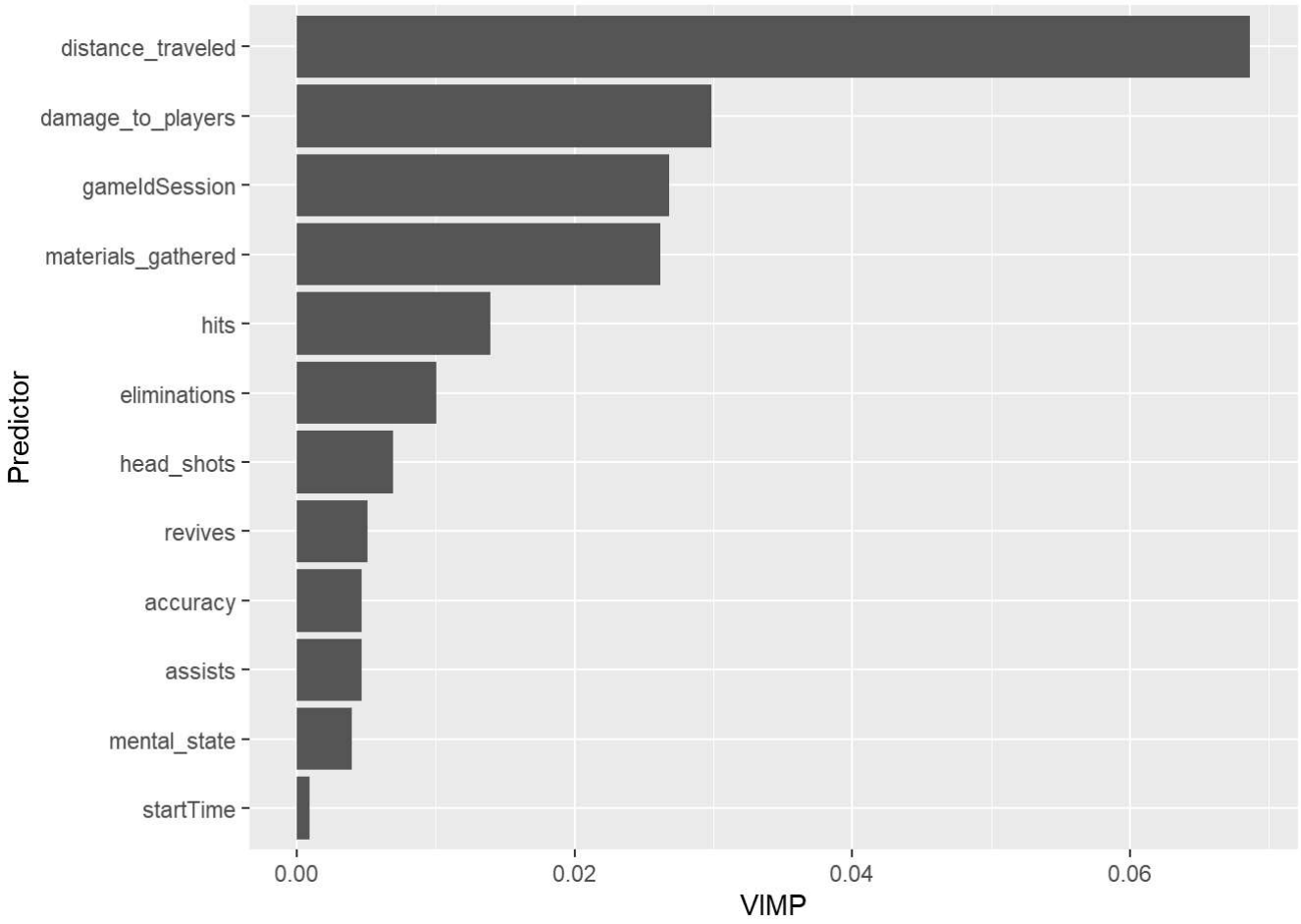
Run it on the full data and use `importance = 'permutation'` to see which variables the random forest thinks are most important. Visualize these results with a barplot. Where do the variables you thought would be best and worst appear?

```
require(ranger)
```

```
## Loading required package: ranger
```

```
## Warning: package 'ranger' was built under R version 4.3.3
```

```
model_rf  <- ranger(won ~ hits + assists + accuracy + head_shots + damage_to_players + eliminati
ons + revives + distance_traveled + materials_gathered + mental_state + startTime + gameIdSessio
n,fn,importance = 'permutation')

toplot <- data.frame(vimp = model_rf$variable.importance,
                     vars = names(model_rf$variable.importance))
toplot %>%
  ggplot(aes(x = vimp,y = reorder(vars,vimp))) +
  geom_bar(stat = 'identity') + labs(x = 'VIMP',y = 'Predictor')
```



While eliminations are better than revives according to this model, they are not nearly as good as distance_traveled and damage_to_players.

Extra Credit [2 Points]

What is the overall AUC from your random forest model? Then use 100 cross validation with a 60-40 split to get a better measure of the true model performance. Is the AUC from running the model on the full data different from the cross validation answer? Calculate the proportion of cross validation splits where the AUC is worse than the value calculated on the full data. Do you think there is evidence of overfitting?

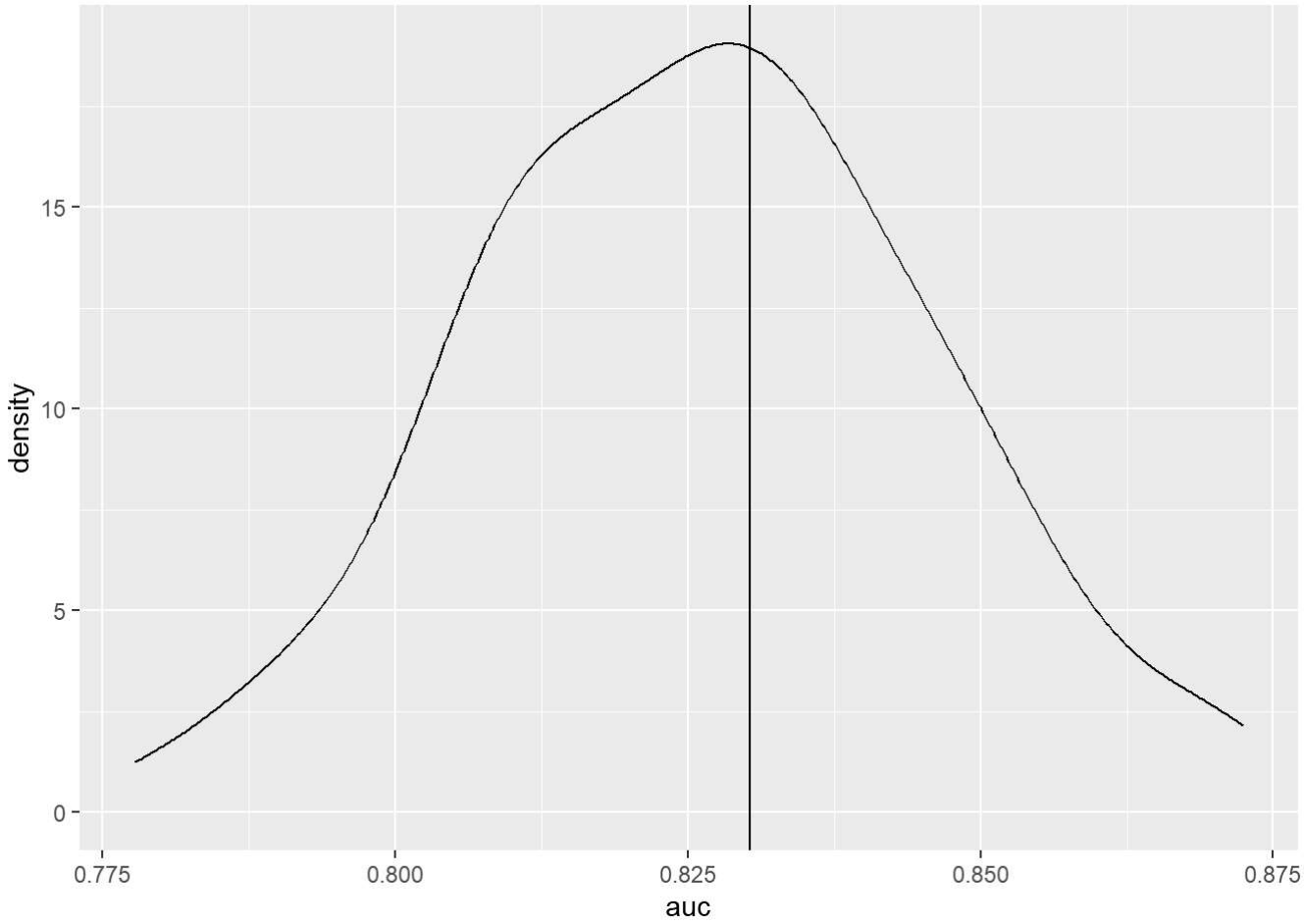
```
toEval <- fn %>%
  mutate(prob_win = model_rf$predictions,
         truth = factor(won,levels = c('1','0')))

fullAUC <- roc_auc(toEval,truth,prob_win)
fullAUC
```

```
## # A tibble: 1 × 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 roc_auc binary         0.830
```

```
set.seed(123)
cvRes <- NULL
for(i in 1:100) {
  # Cross validation prep
  inds <- sample(1:nrow(fn),size = round(nrow(fn)*.6),replace = F)
  train <- fn %>% slice(inds)
  test <- fn %>% slice(-inds)

  # Training models
  mTmp <- ranger(won ~ hits + assists + accuracy + head_shots + damage_to_players + eliminations
+ revives + distance_traveled + materials_gathered + mental_state + startTime + gameIdSession,tr
ain,importance = 'permutation')

  preds <- predict(mTmp,data = test)

  # Predicting models
  toEval <- test %>%
    mutate(prob_win = preds$predictions, # Calculate the probability of winning from the best pr
edictor
           truth = factor(won,levels = c('1','0'))) # Convert the outcome to a factor with level
s c('1','0')

  # Evaluating models
  roc <- roc_auc(toEval,truth,prob_win) %>% # Calculate the AUC for the best predictor
    rename(auc = .estimate) # Rename to 'auc'

  cvRes <- roc %>%
    mutate(cvInd = i) %>%
    bind_rows(cvRes)
}

cvRes %>%
  summarise(mean_auc = mean(auc))
```

```
## # A tibble: 1 × 1
##   mean_auc
##      <dbl>
## 1    0.826
```

```
cvRes %>%
  ggplot(aes(x = auc)) +
  geom_density() +
  geom_vline(xintercept = fullAUC$.estimate)
```




```
cvRes %>%
  summarise(mean(auc < fullAUC$.estimate))
```

```
## # A tibble: 1 × 1
##   `mean(auc < fullAUC$.estimate)`
##                             <dbl>
## 1                            0.57
```

The overall AUC is roughly 0.83. The average across 100 60-40 cross validated splits is 0.826. There is no evidence of overfitting since the distribution of cross-validated AUCs is basically centered on the AUC calculated in the full data.