# Problem Set 8

## Classification

[YOUR NAME]

Due Date: 2024-03-22

# 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]_ps8.Rmd` to your `code` folder.

Copy and paste the contents of this `.Rmd` file into your `[LAST NAME]_ps8.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/blob/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_S2023/blob/main/Lectures/5_Regression/data/fn_cleaned_final.Rds?raw=true') 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]

In this problem set, we are interested in developing a classifier that maximizes our accuracy for predicting Fortnite victories. To do so we will use both a linear probability model and a logit, and then compare their predictive accuracy. We will use two $X$ variables to predict the probability of winning: accuracy ( `accuracy` ), and head shots ( `head_shots` ). Our outcome variable of interest $Y$ is whether the player won the game ( `won` ).

Start by **looking** at these variables. Why types of variables are they? How much missingness do they have? What do their univariate visualizations look like? Then create two multivariate visualizations of the relationship between `won` and each of the two $X$ variables one-by-one. Finally, use `geom_tile()` to create a heatmap of the three-way relationship, where quintiles of `accuracy` is on the x-axis, quintiles of `head_shots` is on the y-axis, and tiles are filled according to the average winning probability. (NB: look up what "quintile" means if you are not sure.) Is there anything surprising about this result?

```
# What types?
glimpse(fn %>% select(accuracy,head_shots,won))
```

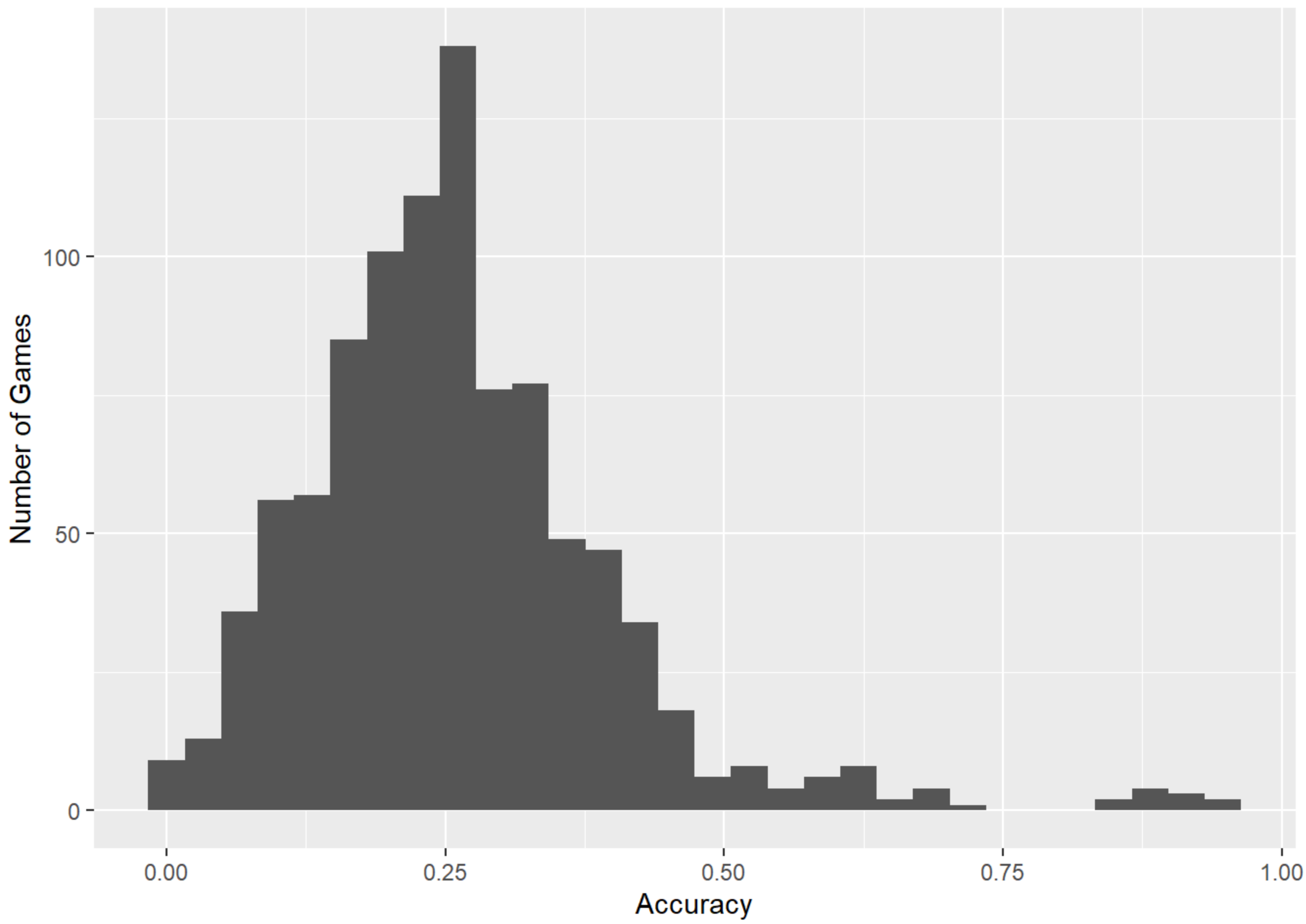
```
## Rows: 957
## Columns: 3
## $ accuracy   <dbl> 0.19371429, 0.32400265, 0.33653340, 0.10506617, 0.62161607,…
## $ head_shots <dbl> 1, 0, 0, 3, 18, 3, 2, 3, 13, 0, 2, 1, 3, 2, 11, 2, 12, 2, 5…
## $ won        <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 1, 0, 0,…
```

```
# How much missingness?
summary(fn %>% select(accuracy,head_shots,won))
```

```
##     accuracy        head_shots          won        
##  Min.   :0.0000   Min.   : 0.000   Min.   :0.0000  
##  1st Qu.:0.1736   1st Qu.: 1.000   1st Qu.:0.0000  
##  Median :0.2469   Median : 3.000   Median :0.0000  
##  Mean   :0.2605   Mean   : 4.829   Mean   :0.3041  
##  3rd Qu.:0.3256   3rd Qu.: 6.000   3rd Qu.:1.0000  
##  Max.   :0.9472   Max.   :36.000   Max.   :1.0000
```

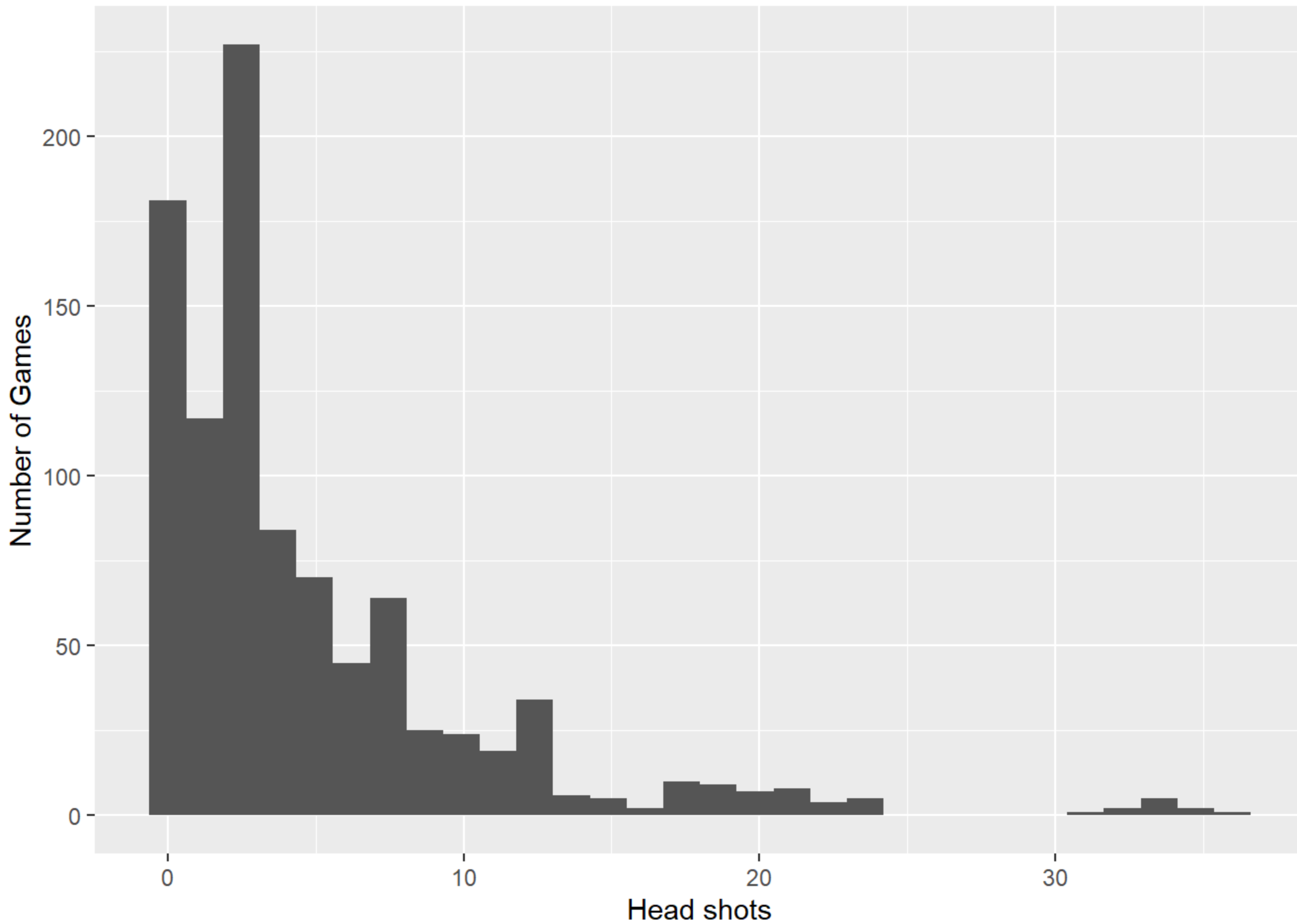
```
# Univariate
fn %>%
  ggplot(aes(x = accuracy)) +
  geom_histogram() +
  labs(x = 'Accuracy',
       y = 'Number of Games')
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```

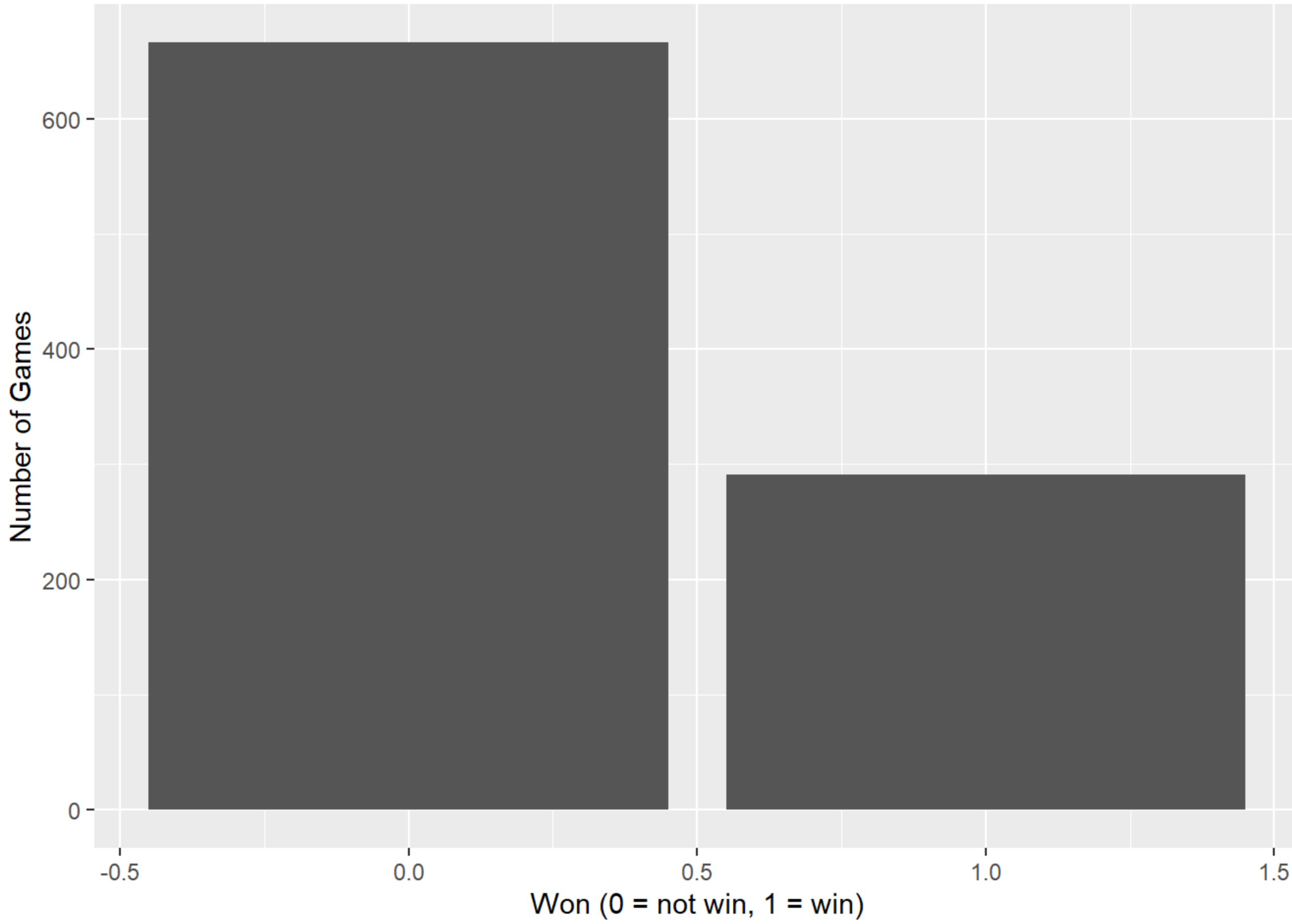



```
fn %>%
  ggplot(aes(x = head_shots)) +
  geom_histogram() +
  labs(x = 'Head shots',
       y = 'Number of Games')
```

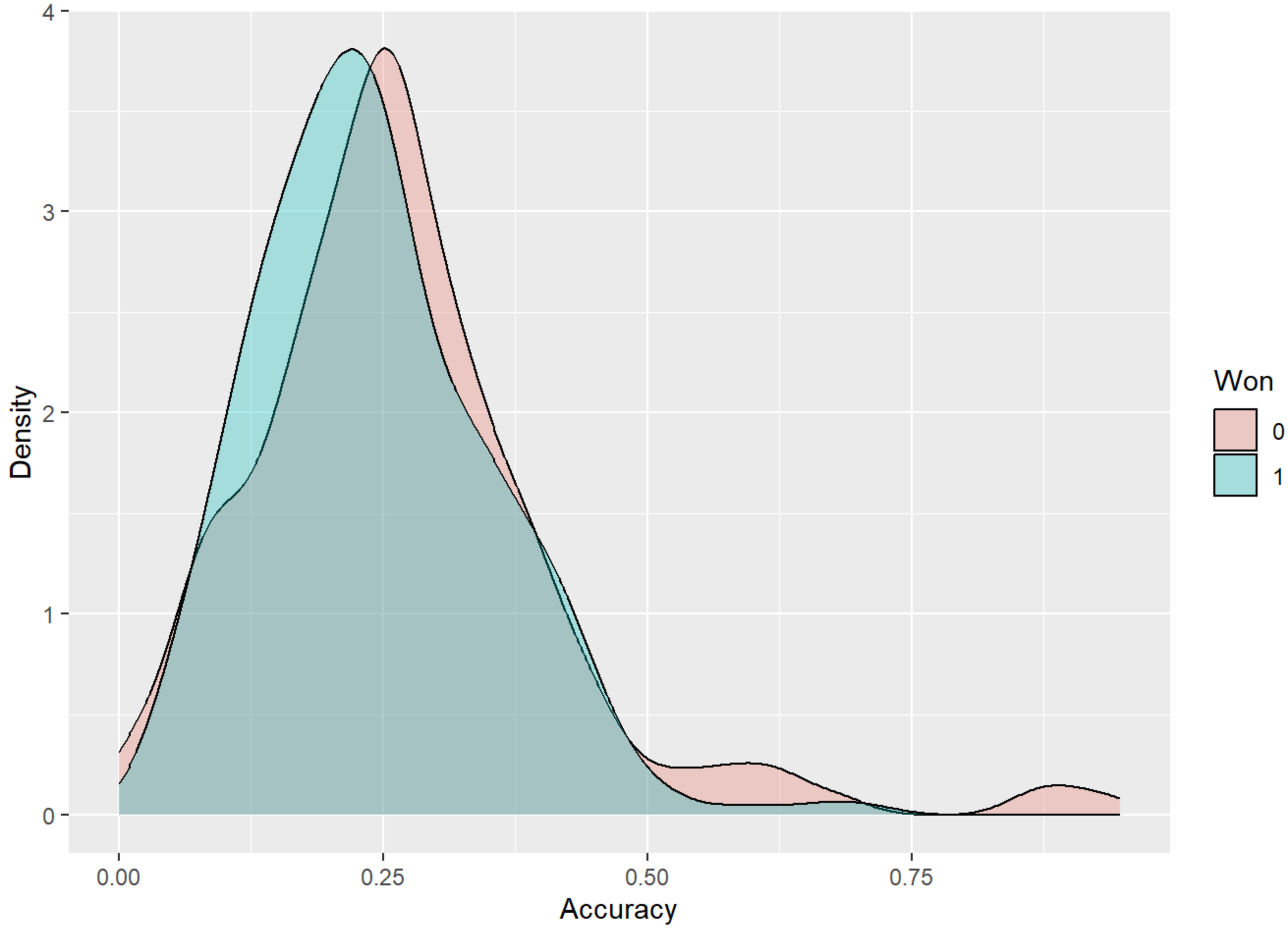
```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



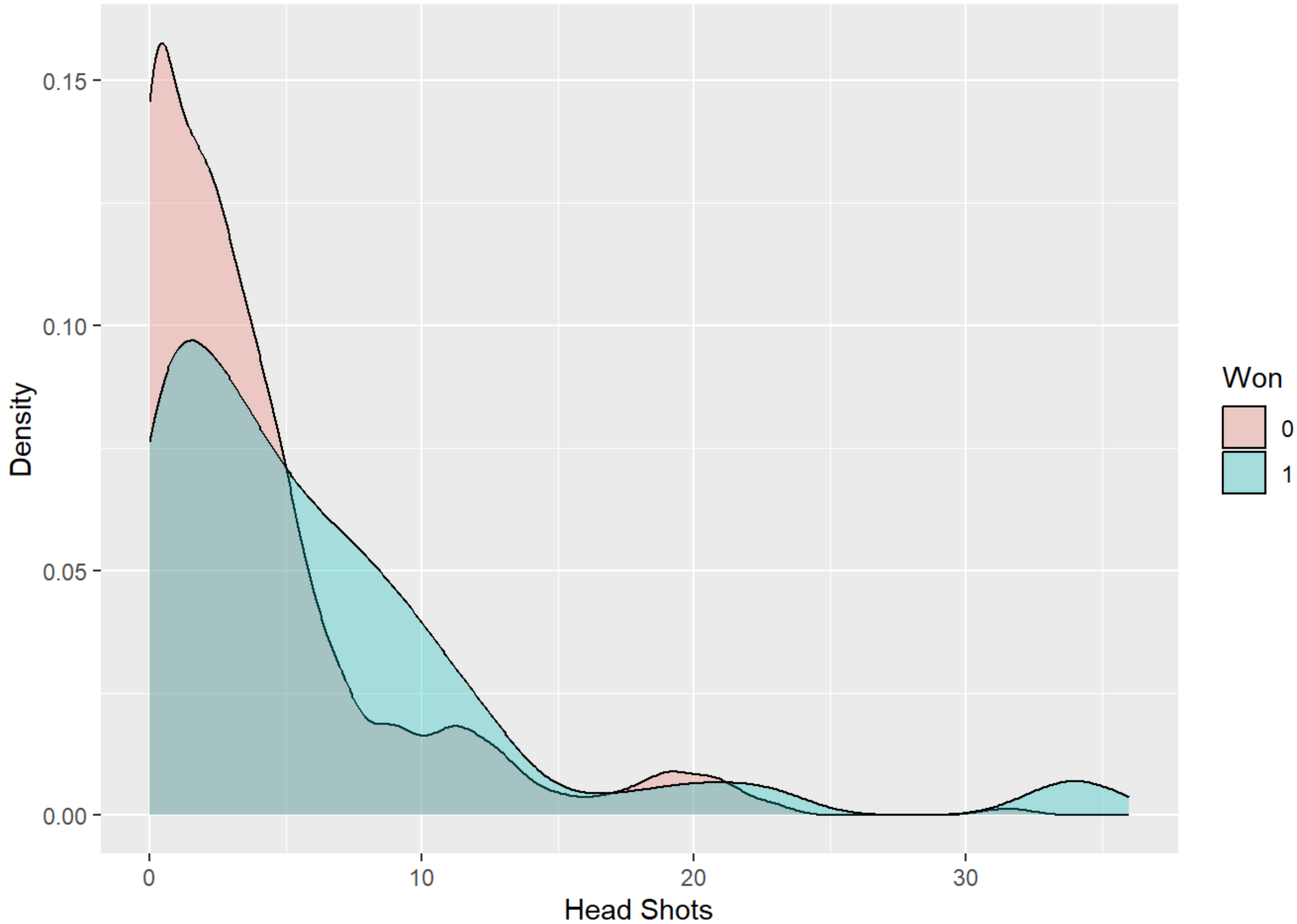

```
fn %>%
  ggplot(aes(x = won)) +
  geom_bar()  +
  labs(x = 'Won (0 = not win, 1 = win)',
       y = 'Number of Games')
```




```
# Multivariate: one-by-one
fn %>%
  ggplot(aes(x = accuracy,fill = factor(won))) +
  geom_density(alpha = .3) +
  labs(x = 'Accuracy',
       y = 'Density',
       fill = 'Won')
```

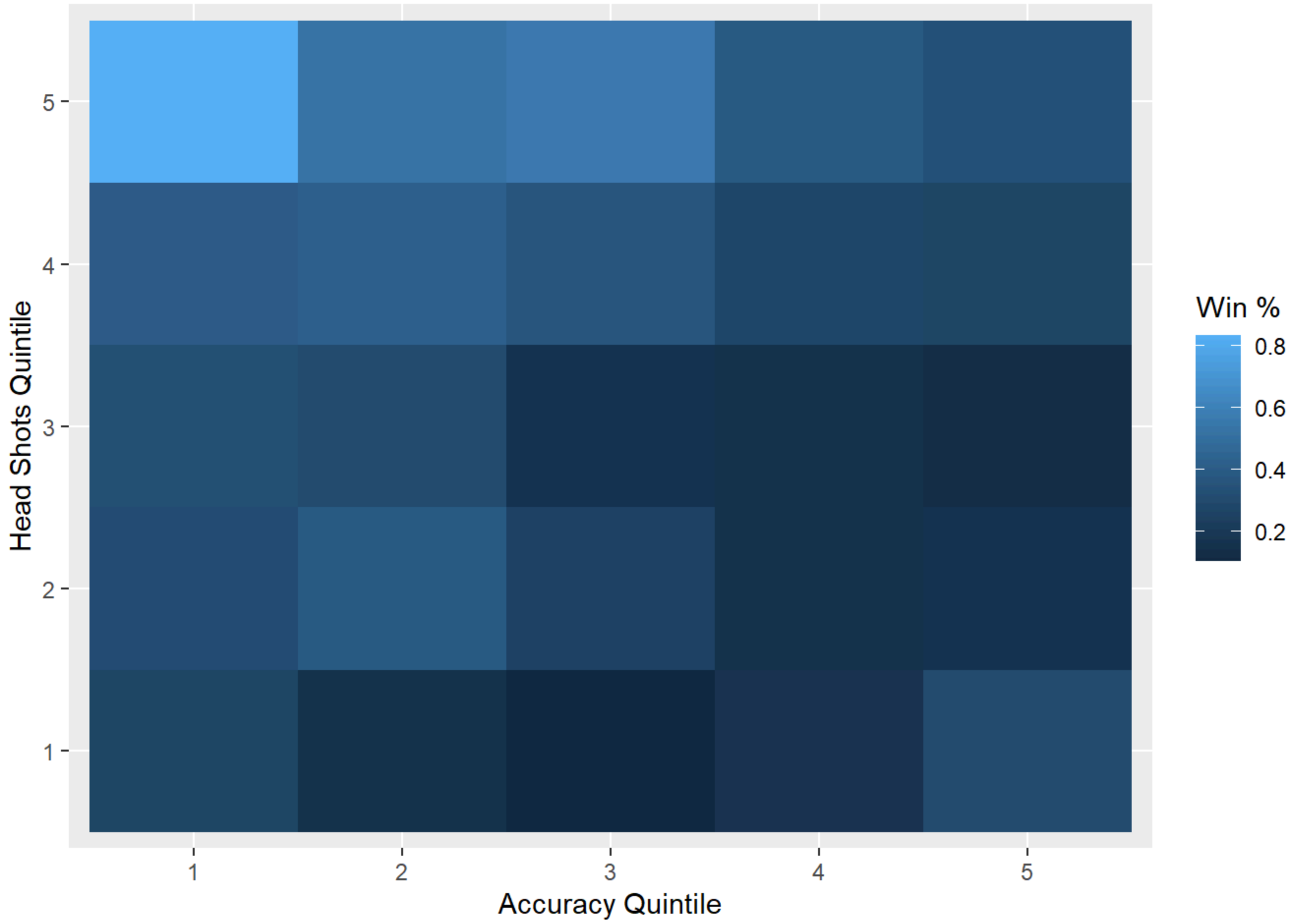



```
fn %>%
  ggplot(aes(x = head_shots,fill = factor(won))) +
  geom_density(alpha = .3) +
  labs(x = 'Head Shots',
       y = 'Density',
       fill = 'Won')
```




```
# Multivariate: 3-dimensions
fn %>%
  mutate(accuracy_quintile = ntile(accuracy,n= 5),
         head_quintile = ntile(head_shots,n = 5)) %>%
  group_by(accuracy_quintile,head_quintile) %>%
  summarise(prob_win = mean(won),nGames = n()) %>%
  # ungroup() %>%
  # mutate(wdth = scales::rescale(nGames,to = c(.15,.95))) %>%
  ggplot(aes(x = factor(accuracy_quintile),y = factor(head_quintile),fill = prob_win)) +
  geom_tile() +
  labs(x = 'Accuracy Quintile',
       y = 'Head Shots Quintile',
       fill = 'Win %')
```

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




> `accuracy` appears to be a continuous measure bounded between 0 and 1. `head_shots` is also continuous, but more of a non-negative count. Finally, won appears to be a binary variable. None of them have missing data. The univariate visualizations suggest that accuracy is unevenly distributed, with some games having a very high accuracy but most having only around 25% accuracy. This might reflect the games in which the player only takes a few, very lucky, shots. Somewhat surprisingly, the heat map suggests that the probability of winning is highest at lower accuracy, but where there are more headshots. This might reflect better players who aim for headshots, sacrificing accuracy in exchange for better damage.

# Question 2 [2 points]

Now let's run a linear model and evaluate it in terms of overall accuracy, sensitivity and specificity using a threshold of 0.5. Then, determine the threshold that maximizes both specificity and sensitivity. Finally, calculate the area under the curve (AUC).

```
require(scales)
```

```
## Loading required package: scales
```

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

```
##
## Attaching package: 'scales'
```

```
## The following object is masked from 'package:purrr':
##
##     discard
```

```
## The following object is masked from 'package:readr':
##
##     col_factor
```

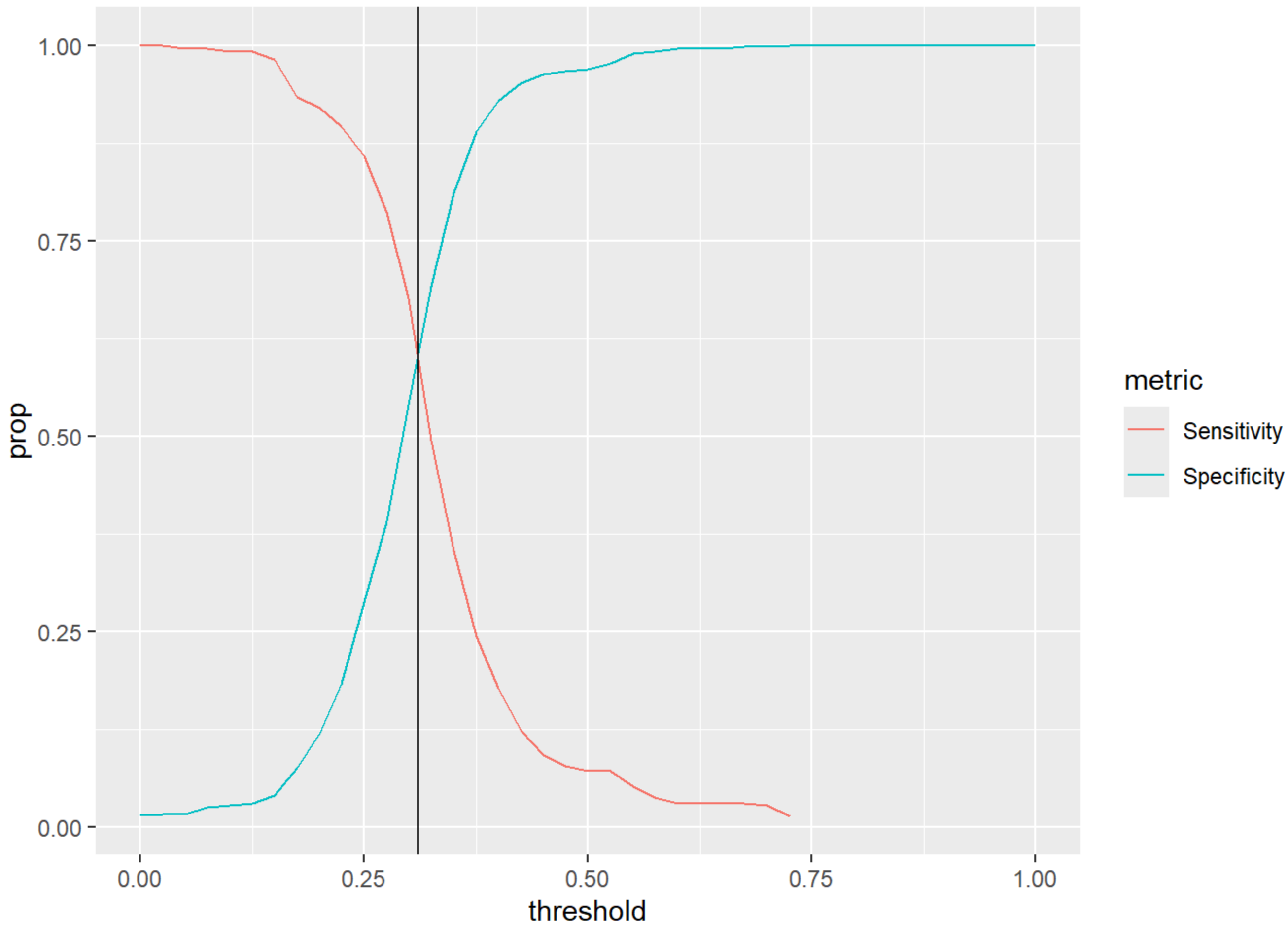
```
# Running linear model
model_lm <- lm(formula = won ~ accuracy + head_shots, # Define the regression equation
               data = fn) # Provide the dataset

# Calculating accuracy, sensitivity, and specificity
fn %>%
  mutate(prob_win = predict(model_lm)) %>% # Calculate the probability of winning
  mutate(pred_win = ifelse(prob_win > .5,1,0)) %>% # Convert the probability to a 1 if the proba
bility is greater than 0.5, or zero otherwise
  group_by(won) %>% # Calculate the total games by whether they were actually won or lost
  mutate(total_games = n()) %>%
  group_by(won,pred_win,total_games) %>% # Calculate the number of games by whether they were ac
tually won or lost, and by whether they were predicted to be won or lost
  summarise(nGames=n(),.groups = 'drop') %>%
  mutate(prop = nGames / total_games) %>% # Calculate the proportion of game by the total games
  ungroup() %>%
  mutate(accuracy = percent(sum((won == pred_win)*nGames) / sum(nGames))) # Calculate the overal
l accuracy
```

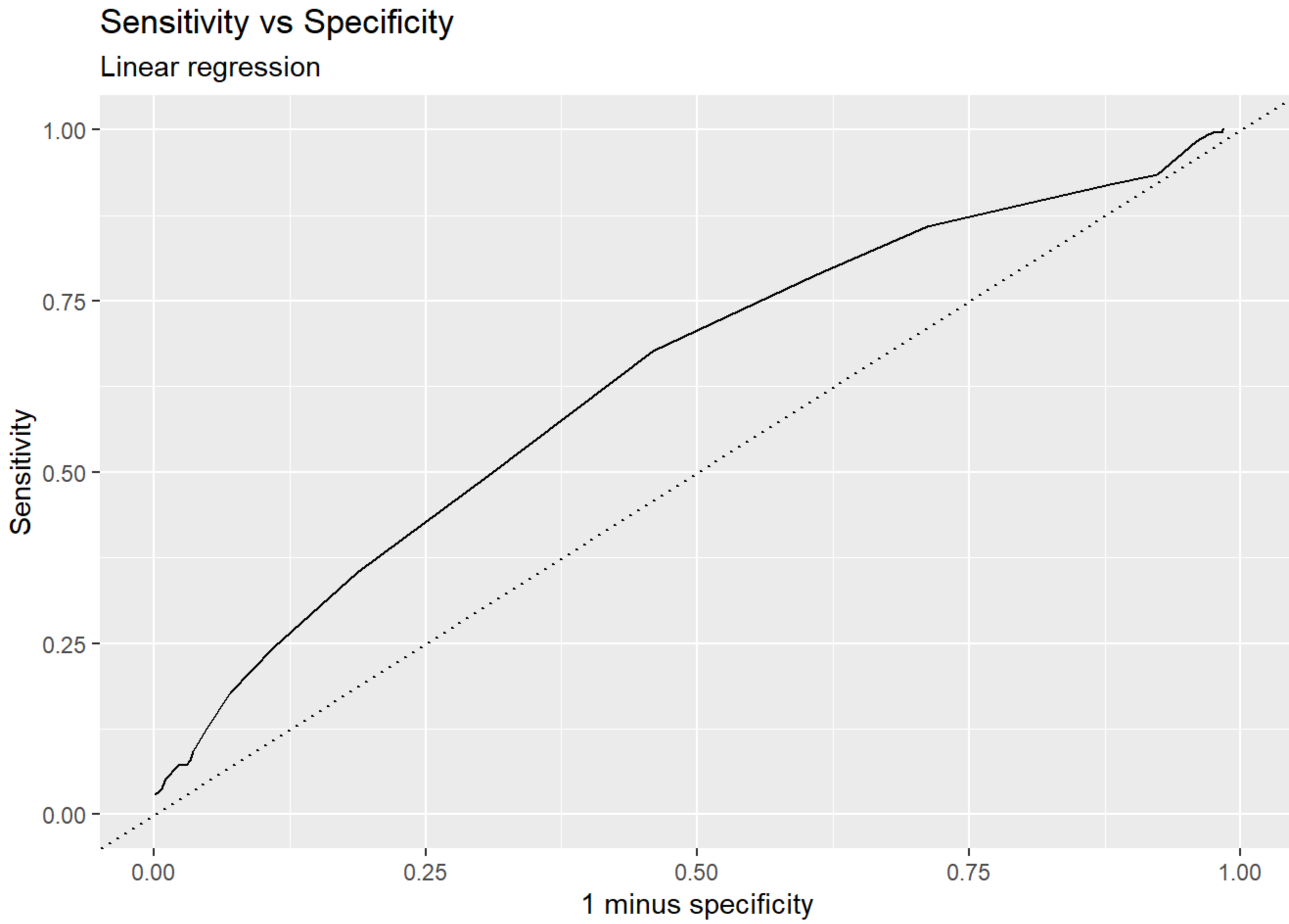
```
## # A tibble: 4 × 6
##     won pred_win total_games nGames   prop accuracy
##   <dbl>    <dbl>       <int>  <int>  <dbl> <chr>
## 1     0        0         666    646 0.970  70%
## 2     0        1         666     20 0.0300 70%
## 3     1        0         291    270 0.928  70%
## 4     1        1         291     21 0.0722 70%
```

```
# Create the sensitivity vs specificity plot
toplot <- NULL # Instantiate an empty object
for(thresh in seq(0,1,by = .025)) {
  toplot <- fn %>%
  mutate(prob_win = predict(model_lm)) %>% # Calculate the probability of winning
  mutate(pred_win = ifelse(prob_win > thresh,1,0)) %>% # Convert the probability to a 1 if the p
robability is greater than the given threshold, or zero otherwise
  group_by(won) %>% # Calculate the total games by whether they were actually won or lost
  mutate(total_games = n()) %>%
  group_by(won,pred_win,total_games) %>% # Calculate the number of games by whether they were ac
tually won or lost, and by whether they were predicted to be won or lost
  summarise(nGames=n(),.groups = 'drop') %>%
  mutate(prop = nGames / total_games) %>% # Calculate the proportion of game by the total games
  ungroup() %>%
  mutate(accuracy = percent(sum((won == pred_win)*nGames) / sum(nGames))) %>% # Calculate the ov
erall accuracy
  mutate(threshold = thresh) %>% # Record the threshold level
    bind_rows(toplot) # Add it to the toplot object
}

toplot %>%
  mutate(metric = ifelse(won == 1 & pred_win == 1,'Sensitivity',
                         ifelse(won == 0 & pred_win == 0,'Specificity',NA))) %>% # Using an ifel
se() function, label each row as either Sensitivity (if the predicted win is 1 and the true win
is 1), Specificity (if the predicted win is 0 and the true win is 0), or NA
  drop_na(metric) %>% # Drop rows that are neither sensitivity nor specificity measures
  ggplot(aes(x = threshold,y = prop,color = metric)) + # Visualize the Sensitivity and Specifici
ty curves by putting the threshold on the x-axis, the proportion of all games on the y-axis, and
coloring by Sensitivity or Specificity
  geom_line() +
  geom_vline(xintercept = .31)
```




```
# Plot the AUC
toplot %>%
  mutate(metric = ifelse(won == 1 & pred_win == 1,'Sensitivity',
                         ifelse(won == 0 & pred_win == 0,'Specificity',NA))) %>% # Using an ifel
se() function, label each row as either Sensitivity (if the predicted win is 1 and the true win
is 1), Specificity (if the predicted win is 0 and the true win is 0), or NA
  drop_na(metric) %>% # Drop rows that are neither sensitivity nor specificity measures
  select(prop,metric,threshold) %>% # Select only the prop, metric, and threshold columns
  spread(metric,prop) %>% # Pivot the data to wide format using either spread() or pivot_wider
(), where the new columns should be the metric
  arrange(desc(Specificity),Sensitivity) %>% # Arrange by descending specificity, and then by se
nsitivity
  ggplot(aes(x = 1-Specificity, # Plot 1 minus the Specificity on the x-axis
             y = Sensitivity)) +  # Plot the Sensitivity on the y-axis
  geom_line() +
  xlim(c(0,1)) + ylim(c(0,1)) + # Expand the x and y-axis limits to be between 0 and 1
  geom_abline(slope = 1,intercept = 0,linetype = 'dotted') + # Add a 45-degree line using geom_a
bline()
  labs(x = '1 minus specificity', # Add clear labels! (Make sure to indicate that this is the re
sult of a linear regression model)
       y = 'Sensitivity',
       title = 'Sensitivity vs Specificity',
       subtitle = 'Linear regression')
```




```
# Calculate the AUC
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 '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.
```

```
forAUC <- fn %>%
  mutate(prob_win = predict(model_lm), # Generate predicted probabilities of winning from our mo
del
         truth = factor(won,levels = c('1','0'))) %>% # Conver the outcome to a factor with leve
ls c('1','0')
  select(truth,prob_win) # Select only the probability and true outcome columns

roc_auc(data = forAUC, # 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         0.639
```

> The threshold that maximizes the sensitivity and specificity is about 0.31.

# Question 3 [2 points]

Now let’s re-do the exact same work, except use a logit model instead of a linear model. Based on your analysis, which mdoel has a larger AUC?

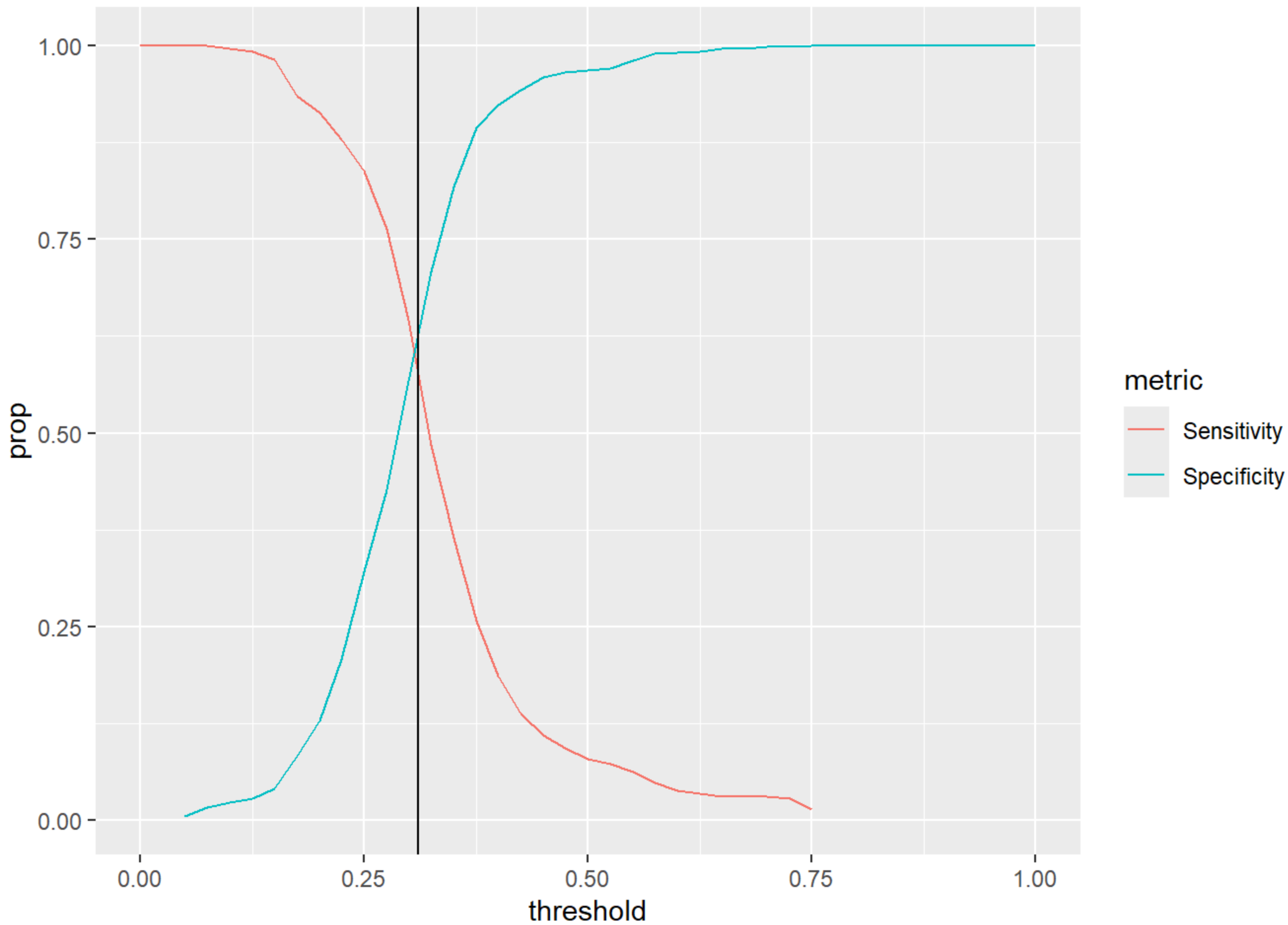
```
# Running linear model
model_glm <- glm(formula = won ~ accuracy + head_shots, # Define the regression equation
               data = fn, # Provide the dataset
               family = binomial(link = 'logit')) # Specify the link function

# Calculating accuracy, sensitivity, and specificity
fn %>%
  mutate(prob_win = predict(model_glm,type = 'response')) %>% # Calculate the probability of win
ning (don't forget to set type = 'response' for the glm()!)
  mutate(pred_win = ifelse(prob_win > .5,1,0)) %>% # Convert the probability to a 1 if the proba
bility is greater than 0.5, or zero otherwise
  group_by(won) %>% # Calculate the total games by whether they were actually won or lost
  mutate(total_games = n()) %>%
  group_by(won,pred_win,total_games) %>% # Calculate the number of games by whether they were ac
tually won or lost, and by whether they were predicted to be won or lost
  summarise(nGames=n(),.groups = 'drop') %>%
  mutate(prop = nGames / total_games) %>% # Calculate the proportion of game by the total games
  ungroup() %>%
  mutate(accuracy = percent(sum((won == pred_win)*nGames) / sum(nGames))) # Calculate the overal
l accuracy
```

```
## # A tibble: 4 × 6
##     won pred_win total_games nGames   prop accuracy
##   <dbl>    <dbl>       <int>  <int>  <dbl> <chr>
## 1     0        0         666    645 0.968  70%
## 2     0        1         666     21 0.0315 70%
## 3     1        0         291    268 0.921  70%
## 4     1        1         291     23 0.0790 70%
```

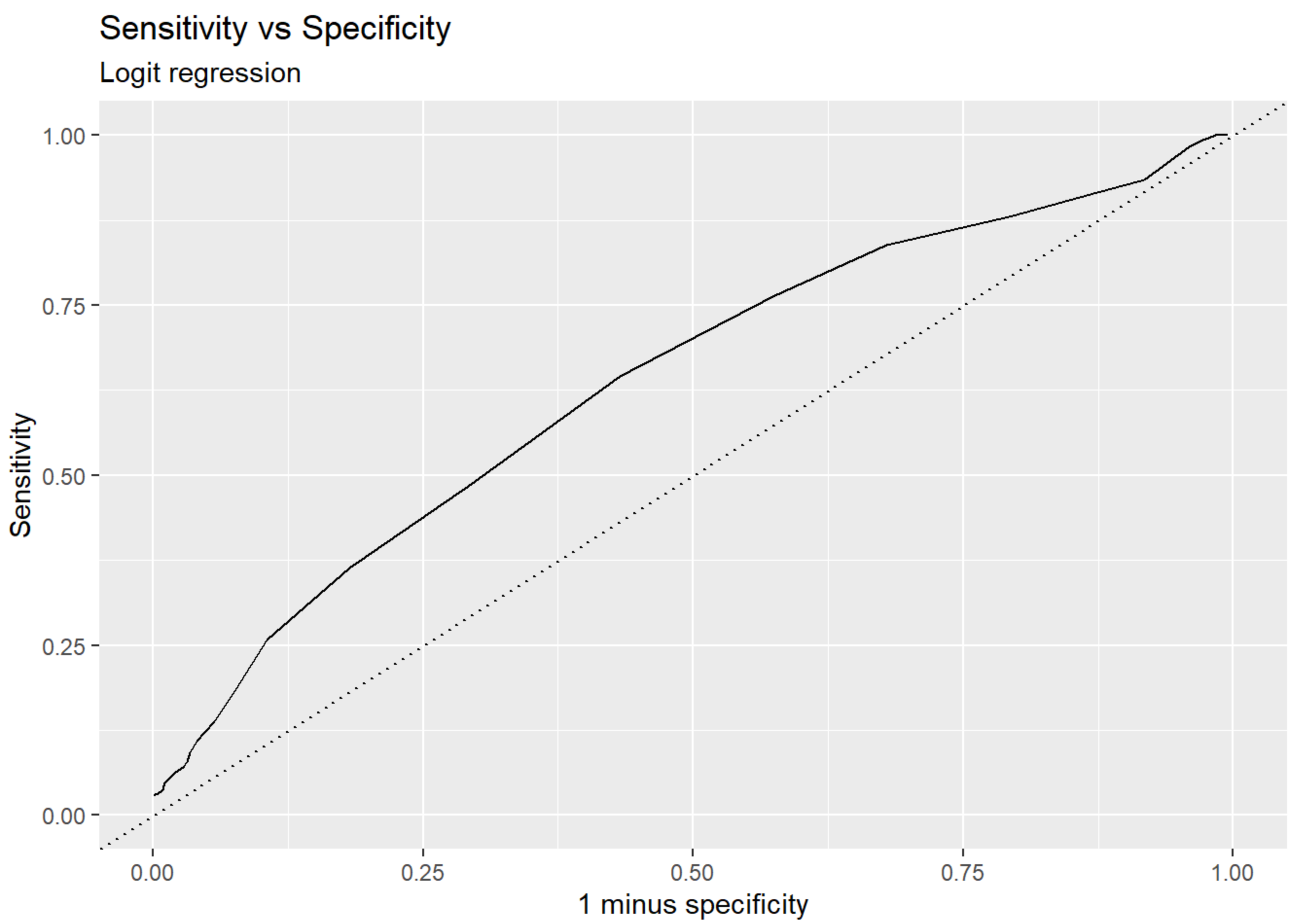
```
# Create the sensitivity vs specificity plot
toplot <- NULL # Instantiate an empty object
for(thresh in seq(0,1,by = .025)) {
  toplot <- fn %>%
  mutate(prob_win = predict(model_glm,type = 'response')) %>% # Calculate the probability of winning (don't forget to set type = 'response' for the glm()!)
  mutate(pred_win = ifelse(prob_win > thresh,1,0)) %>% # Convert the probability to a 1 if the probability is greater than the given threshold, or zero otherwise
  group_by(won) %>% # Calculate the total games by whether they were actually won or lost
  mutate(total_games = n()) %>%
  group_by(won,pred_win,total_games) %>% # Calculate the number of games by whether they were actually won or lost, and by whether they were predicted to be won or lost
  summarise(nGames=n(),.groups = 'drop') %>%
  mutate(prop = nGames / total_games) %>% # Calculate the proportion of game by the total games
  ungroup() %>%
  mutate(accuracy = percent(sum((won == pred_win)*nGames) / sum(nGames))) %>% # Calculate the overall accuracy
  mutate(threshold = thresh) %>% # Record the threshold level
    bind_rows(toplot) # Add it to the toplot object
}

toplot %>%
  mutate(metric = ifelse(won == 1 & pred_win == 1,'Sensitivity',
                         ifelse(won == 0 & pred_win == 0,'Specificity',NA))) %>% # Using an ifelse() function, label each row as either Sensitivity (if the predicted win is 1 and the true win is 1), Specificity (if the predicted win is 0 and the true win is 0), or NA
  drop_na(metric) %>% # Drop rows that are neither sensitivity nor specificity measures
  ggplot(aes(x = threshold,y = prop,color = metric)) + # Visualize the Sensitivity and Specificity curves by putting the threshold on the x-axis, the proportion of all games on the y-axis, and coloring by Sensitivity or Specificity
  geom_line() +
  geom_vline(xintercept = .31)
```




```
# Plot the AUC
toplot %>%
  mutate(metric = ifelse(won == 1 & pred_win == 1,'Sensitivity',
                         ifelse(won == 0 & pred_win == 0,'Specificity',NA))) %>% # Using an ifel
se() function, label each row as either Sensitivity (if the predicted win is 1 and the true win
is 1), Specificity (if the predicted win is 0 and the true win is 0), or NA
  drop_na(metric) %>% # Drop rows that are neither sensitivity nor specificity measures
  select(prop,metric,threshold) %>% # Select only the prop, metric, and threshold columns
  spread(metric,prop) %>% # Pivot the data to wide format using either spread() or pivot_wider
(), where the new columns should be the metric
  arrange(desc(Specificity),Sensitivity) %>% # Arrange by descending specificity, and then by se
nsitivity
  ggplot(aes(x = 1-Specificity, # Plot 1 minus the Specificity on the x-axis
             y = Sensitivity)) +  # Plot the Sensitivity on the y-axis
  geom_line() +
  xlim(c(0,1)) + ylim(c(0,1)) + # Expand the x and y-axis limits to be between 0 and 1
  geom_abline(slope = 1,intercept = 0,linetype = 'dotted') + # Add a 45-degree line using geom_a
bline()
  labs(x = '1 minus specificity',# Add clear labels! (Make sure to indicate that this is the res
ult of a logit regression model)
       y = 'Sensitivity',
       title = 'Sensitivity vs Specificity',
       subtitle = 'Logit regression')
```

```
## Warning: Removed 2 rows containing missing values or values outside the scale range
## (`geom_line()`).
```




```
# Calculate the AUC
forAUC <- fn %>%
  mutate(prob_win = predict(model_glm,type = 'response'), # Generate predicted probabilities of winning from our model
         truth = factor(won,levels = c('1','0'))) %>% # Conver the outcome to a factor with levels c('1','0')
  select(truth,prob_win) # Select only the probability and true outcome columns

roc_auc(data = forAUC, # 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         0.639
```

> As before, the threshold that maximizes the sensitivity and specificity is about 0.31. There is no difference between the linear and logit models in this example.

# Question 4 [2 points]

Use 100-fold cross validation with a 60-40 split to calculate the average AUC for both the linear and logit models. Which is better?

```
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
  mLM <- lm(formula = won ~ accuracy + head_shots,
            data = train)
  mGLM <- glm(formula = won ~ accuracy + head_shots,
            data = train,
            family = binomial(link = 'logit'))

  # Predicting models
  toEval <- test %>%
    mutate(mLMPreds = predict(mLM,newdata = test),
           mGLMPreds = predict(mGLM,newdata = test,type = 'response'),
           truth = factor(won,levels = c('1','0')))

  # Evaluating models
  rocLM <- roc_auc(toEval,truth,mLMPreds) %>%
    mutate(model = 'linear') %>%
    rename(auc = .estimate)

  rocGLM <- roc_auc(toEval,truth,mGLMPreds) %>%
    mutate(model = 'logit') %>%
    rename(auc = .estimate)

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

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

```
## # A tibble: 2 × 2
##   model  mean_auc
##   <chr>     <dbl>
## 1 linear    0.639
## 2 logit     0.639
```

> There is no difference between these models across cross validated calculations of the AUC.

# Extra Credit [2 Points + 2 points for winner]

Can you improve on the best model identified above? You will receive two extra credit points for executing the analysis correctly. The student(s) who achieve the best cross-validated AUC in class will receive an additional 2 extra points on top of the EC.

> Who can beat an AUC of 0.84?

```
# INSERT CODE HERE
require(ranger)
```

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

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

```
require(glmnet)
```

```
## Loading required package: glmnet
```

```
## Loading required package: Matrix
```

```
##
## Attaching package: 'Matrix'
```

```
## The following objects are masked from 'package:tidyr':
##
##     expand, pack, unpack
```

```
## Loaded glmnet 4.1-8
```

```
Xs <- colnames(fn %>%
                 select(-placed,-won,-player,-sessionId,-lagSess))

fn <- fn %>%
  drop_na(all_of(Xs))

form.full <- paste0('won ~ ',paste(Xs,collapse = ' + '))

rf.full <- ranger(formula = as.formula(form.full),
                  data = fn,
                  num.trees = 5000,mtry = 19)

(fullCV <- roc_auc(fn %>%
  mutate(prob_win = rf.full$predictions,
         truth = factor(won,levels = c('1','0'))),
  truth,prob_win))
```

```
## # A tibble: 1 × 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 roc_auc binary         0.827
```

```
set.seed(123)
cvRes <- NULL
for(i in 1:30) {
  # Cross validation prep
  inds <- sample(1:nrow(fn),size = round(nrow(fn)*.8),replace = F)
  train <- fn %>% slice(inds)
  test <- fn %>% slice(-inds)

  # Training models
  rf.full <- ranger(formula = as.formula(form.full),data = train,
                  num.trees = 2000,mtry = 5)

  # Predicting models
  preds <- predict(rf.full,data = test)
  cvRes <- roc_auc(test %>%
  mutate(prob_win = preds$predictions,
         truth = factor(won,levels = c('1','0'))),
  truth,prob_win) %>%
    mutate(cvInd = i) %>%
    bind_rows(cvRes)
}

cvRes %>%
  summarise(mean_auc = mean(.estimate))
```

```
## # A tibble: 1 × 1
##   mean_auc
##      <dbl>
## 1    0.840
```

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