Problem Set 8

Classification

[YOUR NAME]

Due Date: 2024-11-08

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 `[YOUR NAME]_ps8.Rmd` to your `code` folder.

Copy and paste the contents of this `.Rmd` file into your `[YOUR NAME]_ps8.Rmd` file. Then change the `author: [Your Name]` on line 2 to your name.

We will be using the fn_cleaned_final.Rds file from the course github page (https://github.com/jbisbee1/DS1000_F2024/blob/main/data/fn_cleaned_final.Rds). The codebook for this dataset is produced below.

Name	Description
player	A unique ID for each player
gameId	A unique ID for each game
startTime	When the game started
sessionId	A unique ID for each gaming session
gameIdSession	A unique ID for each game within a session
won	A binary variable indicating whether the game was won or not
mental_state	Whether the player was drunk or sober when they played
eliminations	How many times they killed an enemy player
assists	How many times they helped a teammate kill an enemy player
revives	How many times was the player revived (brought back to life)?
accuracy	The proportion of total shots that hit an enemy player
hits	The number of shots that hit an enemy player
head_shots	The number of shots that hit an enemy player in the head
distance_traveled	How far did the player move in the game?
materials_gathered	How many materials did the player gather in the game?
materials_used	How many materials did the player use in the game?
damage_taken	The amount of damage the player received from enemy players

Name	Description
damage_to_players	How much damage did the player commit to other players in the game?
damage_to_structures	How much damage did the player commit to structures in the game?

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.

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 5 total points, plus **one** extra credit question. However, there are also additional opportunities for additional points in problem 5 and the extra credit question itself. Note that these additional points are **very hard**. They are designed for the students who claimed that the course is easy on the midterm evals. I encourage you all to attempt all extra credit, but don't worry if you don't get the super hard ones.

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. 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. To submit, compile (i.e., `knit`) the completed problem set and upload the PDF file to Brightspace on Friday by midnight. Instructions for how to compile the output as a PDF can be found in Problem Set 0 (https://github.com/jbisbee1/DS1000_F2024/blob/main/Psets/ds1000_pset_0.pdf) and in this gif tutorial (https://github.com/jbisbee1/DS1000_F2024/blob/main/Psets/save_as_pdf.gif).

This problem set is worth 5 total points, plus 1 extra credit point. 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 will be deducted 1 point for each day late the problem set is submitted, and 1 point for failing to submit in the correct format.

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.

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`, `tidymodels` and `ranger`, and then load the `fn_cleaned_final.Rds` (https://github.com/jbisbee1/DS1000_F2024/raw/main/data/fn_cleaned_final.rd') data to an object called `fn`.

```
require(tidyverse)
require(tidymodels)
```

```
## Warning: package 'tidymodels' was built under R version 4.4.1
## Warning: package 'dials' was built under R version 4.4.1
```

```
## Warning: package 'infer' was built under R version 4.4.1
```

```
## Warning: package 'modeldata' was built under R version 4.4.1
```

```
## Warning: package 'parsnip' was built under R version 4.4.1
```

```
## Warning: package 'recipes' was built under R version 4.4.1
```

```
## Warning: package 'rsample' was built under R version 4.4.1
```

```
## Warning: package 'tune' was built under R version 4.4.1
```

```
## Warning: package 'workflows' was built under R version 4.4.1
```

```
## Warning: package 'workflowsets' was built under R version 4.4.1
```

```
## Warning: package 'yardstick' was built under R version 4.4.1
```

```
require(ranger)
```

```
## Warning: package 'ranger' was built under R version 4.4.1
```

```
fn <- read_rds('https://github.com/jbisbee1/DS1000_F2024/raw/main/data/fn_cleaned_final.rds')
```

Question 1 [1 point]

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 a linear probability model, a logit, and a random forest. We will start by using 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?*

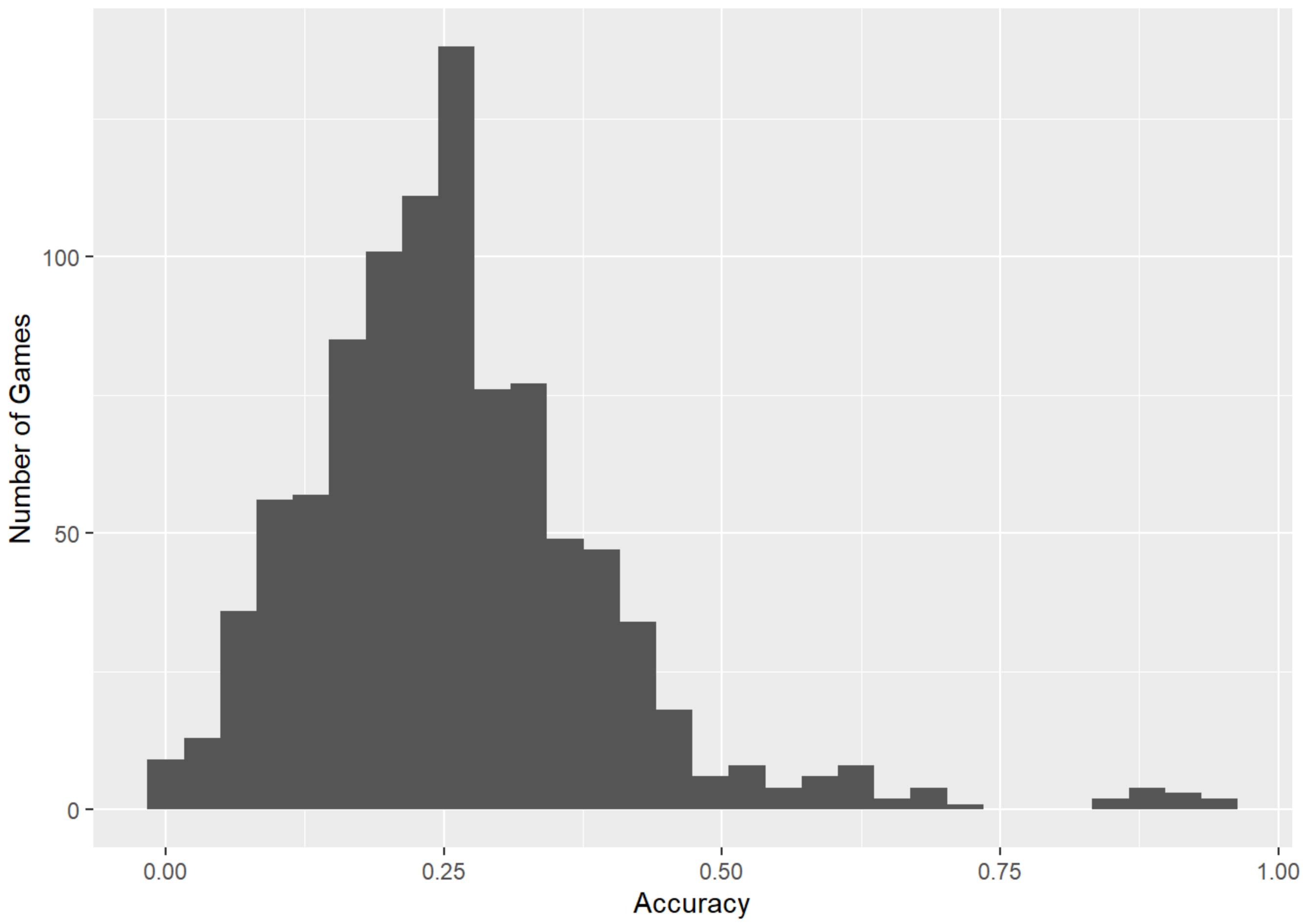
```
# What types?
glimpse(fn %>% select(accuracy,head_shots,won))
```

```
## 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,…
```

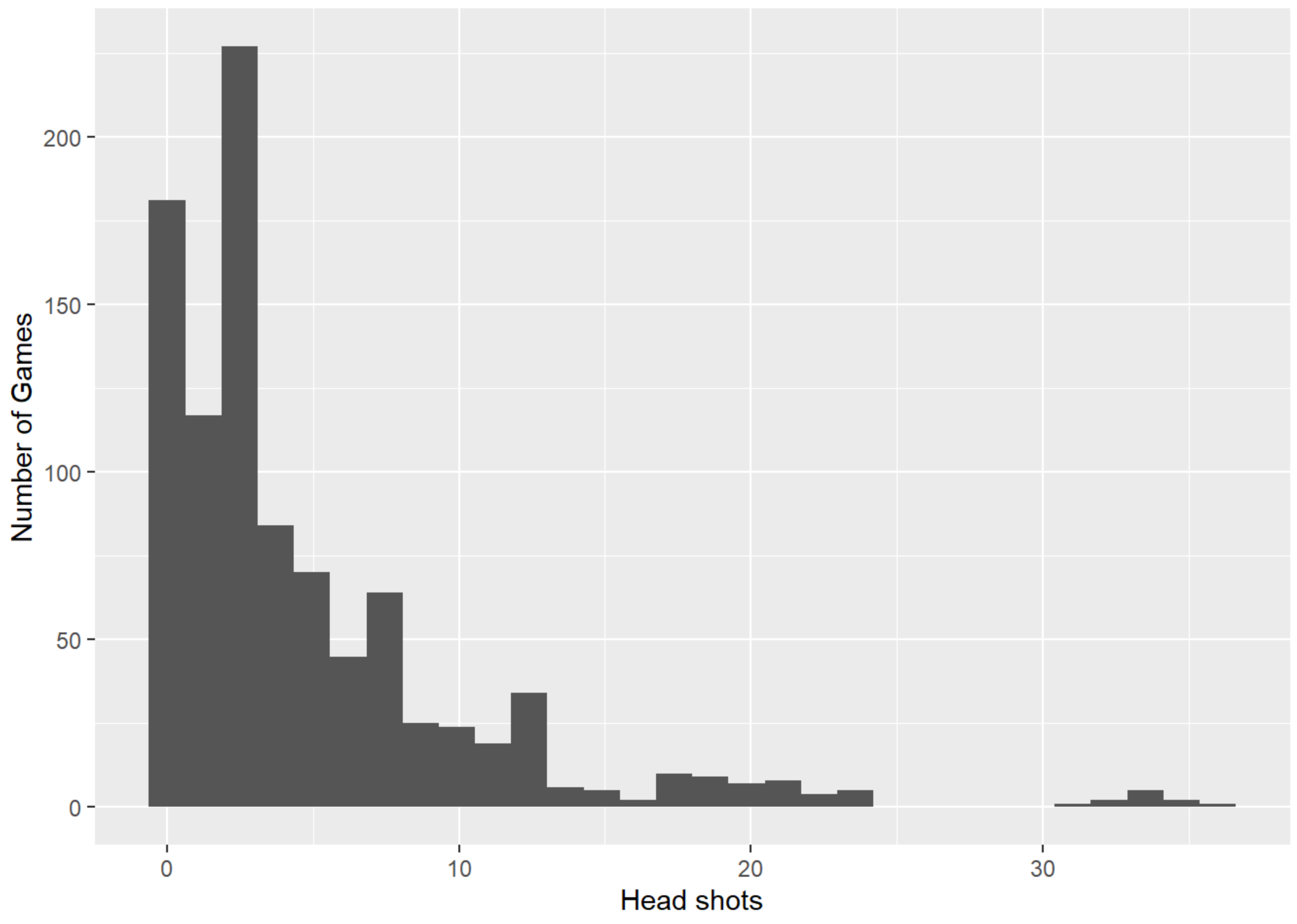
```
# How much missingness?
summary(fn %>% select(accuracy,head_shots,won))
```

```
##     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
```

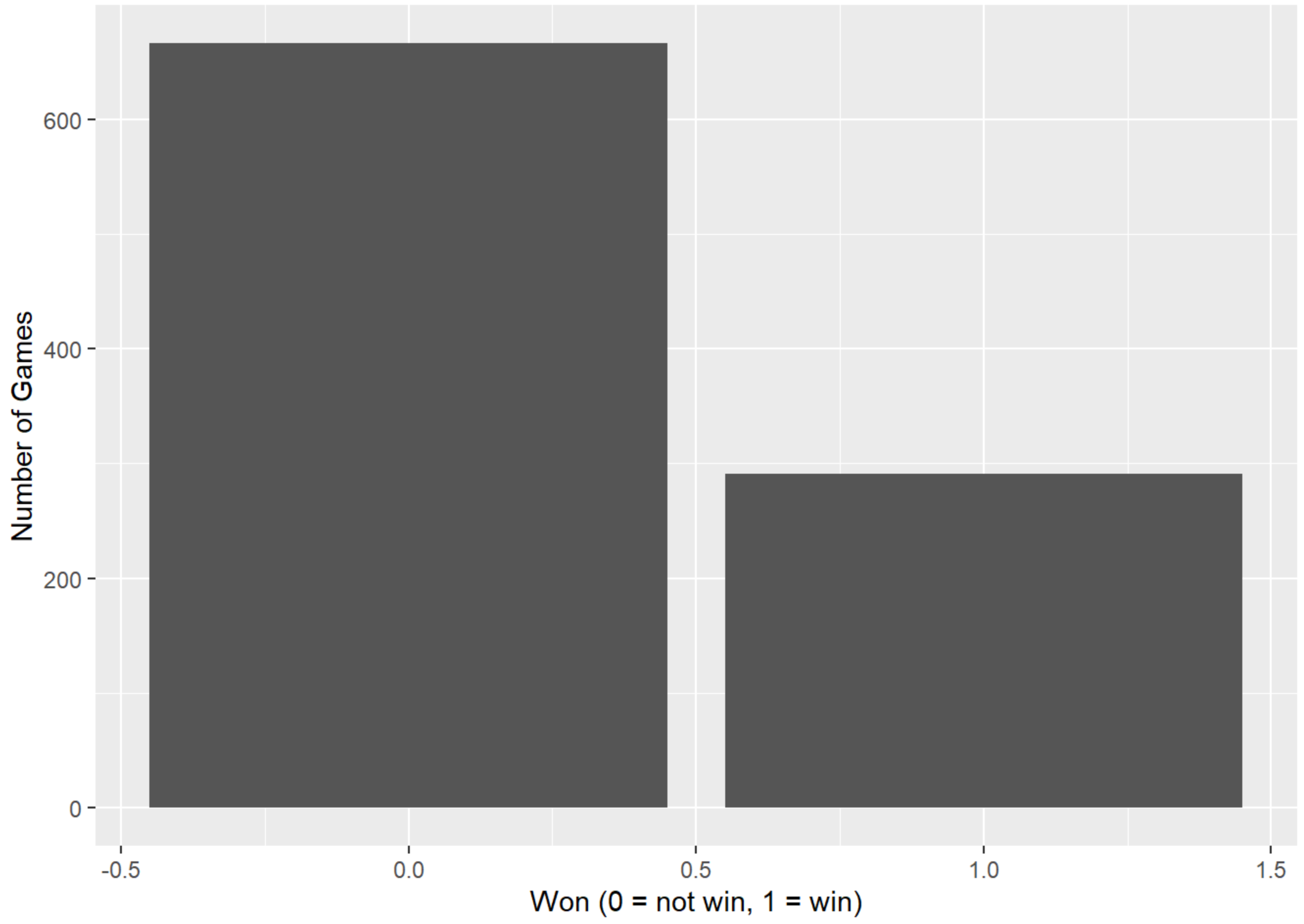
```
# Univariate
fn %>%
  ggplot(aes(x = accuracy)) +
  geom_histogram() +
  labs(x = 'Accuracy',
       y = 'Number of Games')
```



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



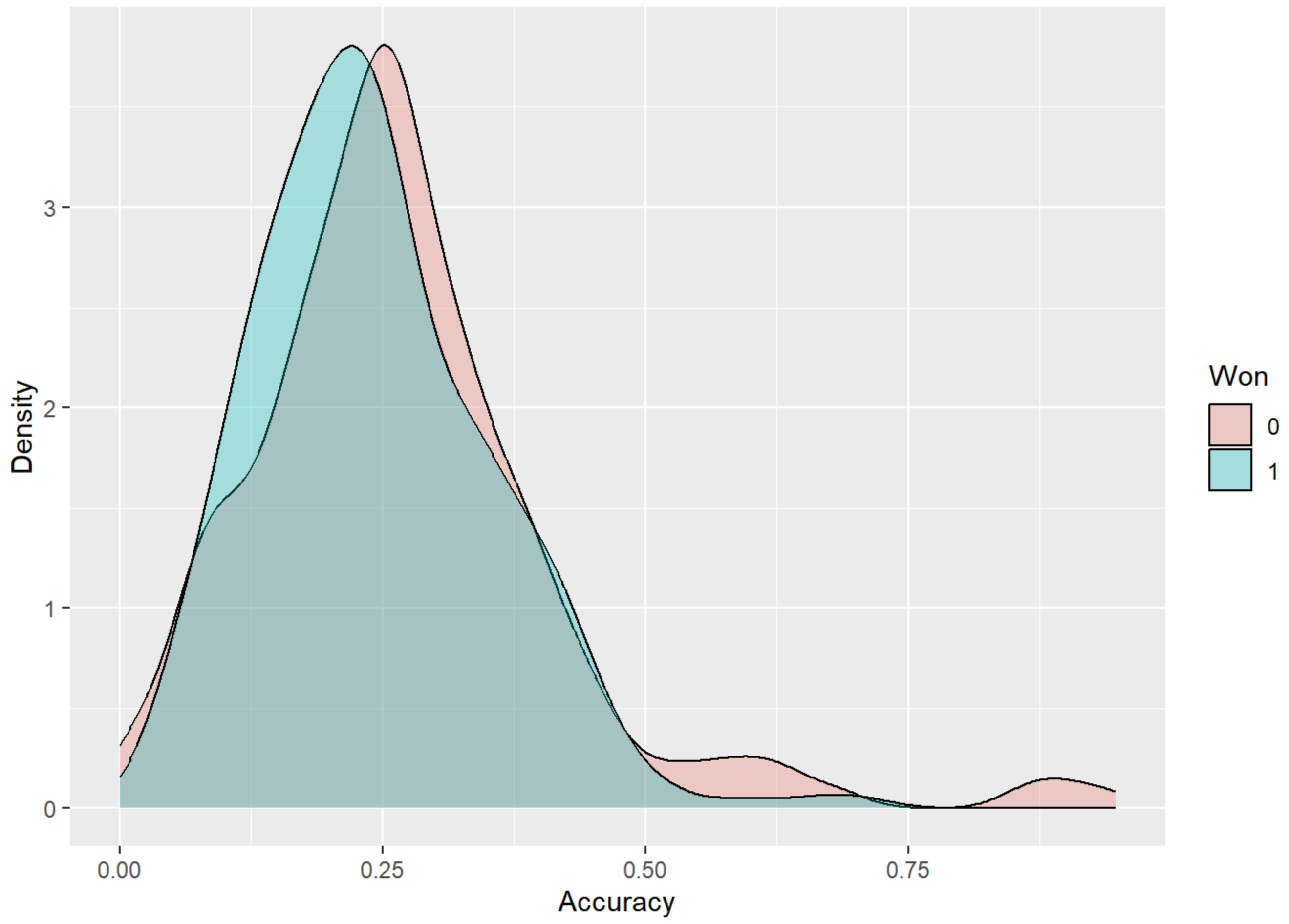
```
fn %>%
  ggplot(aes(x = won)) +
  geom_bar()  +
  labs(x = 'Won (0 = not win, 1 = win)',
       y = 'Number of Games')
```



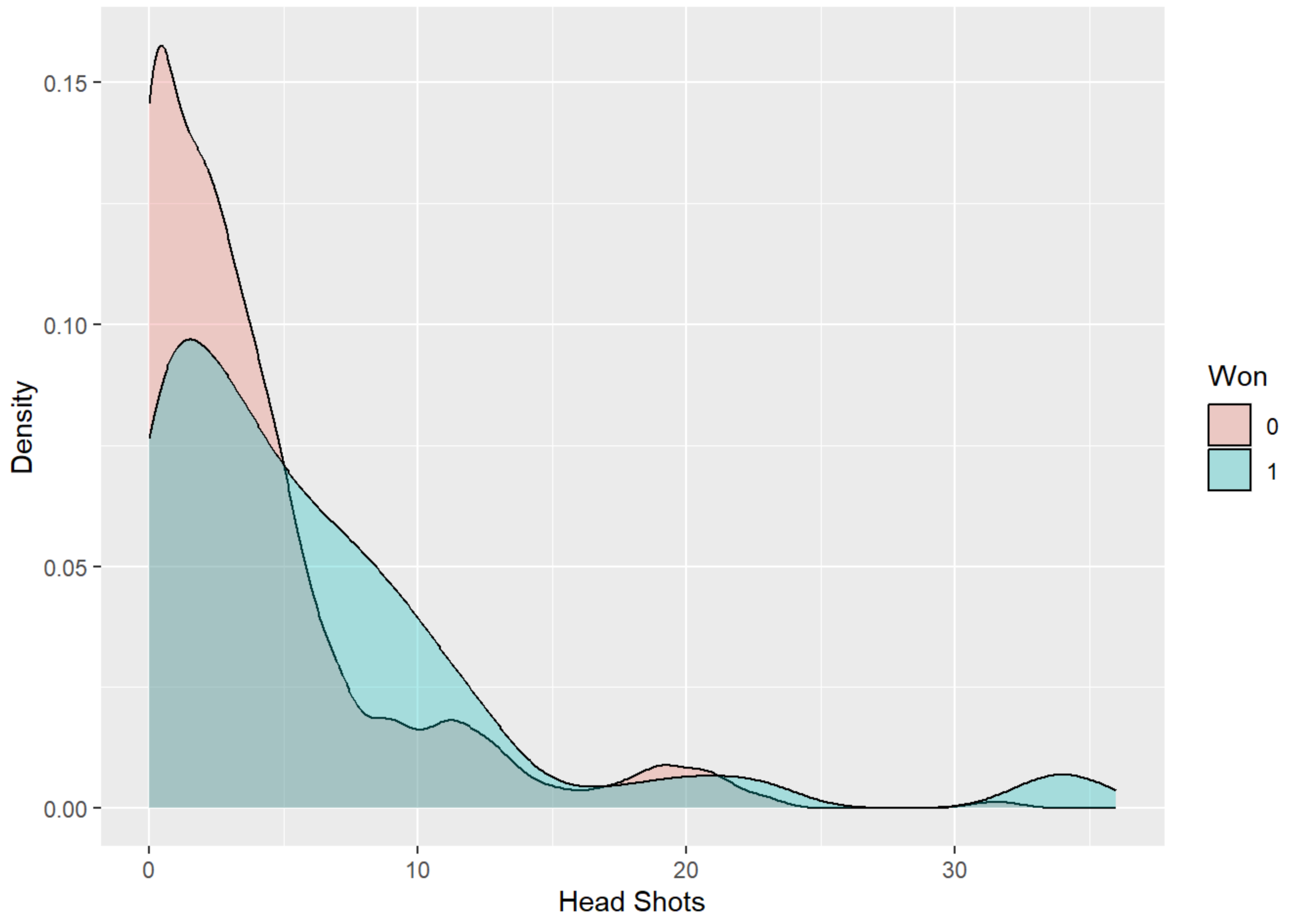
> `accuracy` appears to be a continuous measure bounded between 0 and 1. `head_shots` is also continuous, but more of a non-negative count. Finally, won appears to be a binary variable. None of them have missing data. The univariate visualizations suggest that accuracy is unevenly distributed, with some 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. [RUBRIC: +.1 for correctly identifying variable type; +.1 for noting that there is no missingness; +.3 points for correct univariate visualizations for each.]

Then create two multivariate visualizations of the relationship between `won` *and each of the two X variables one-by-one.*

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



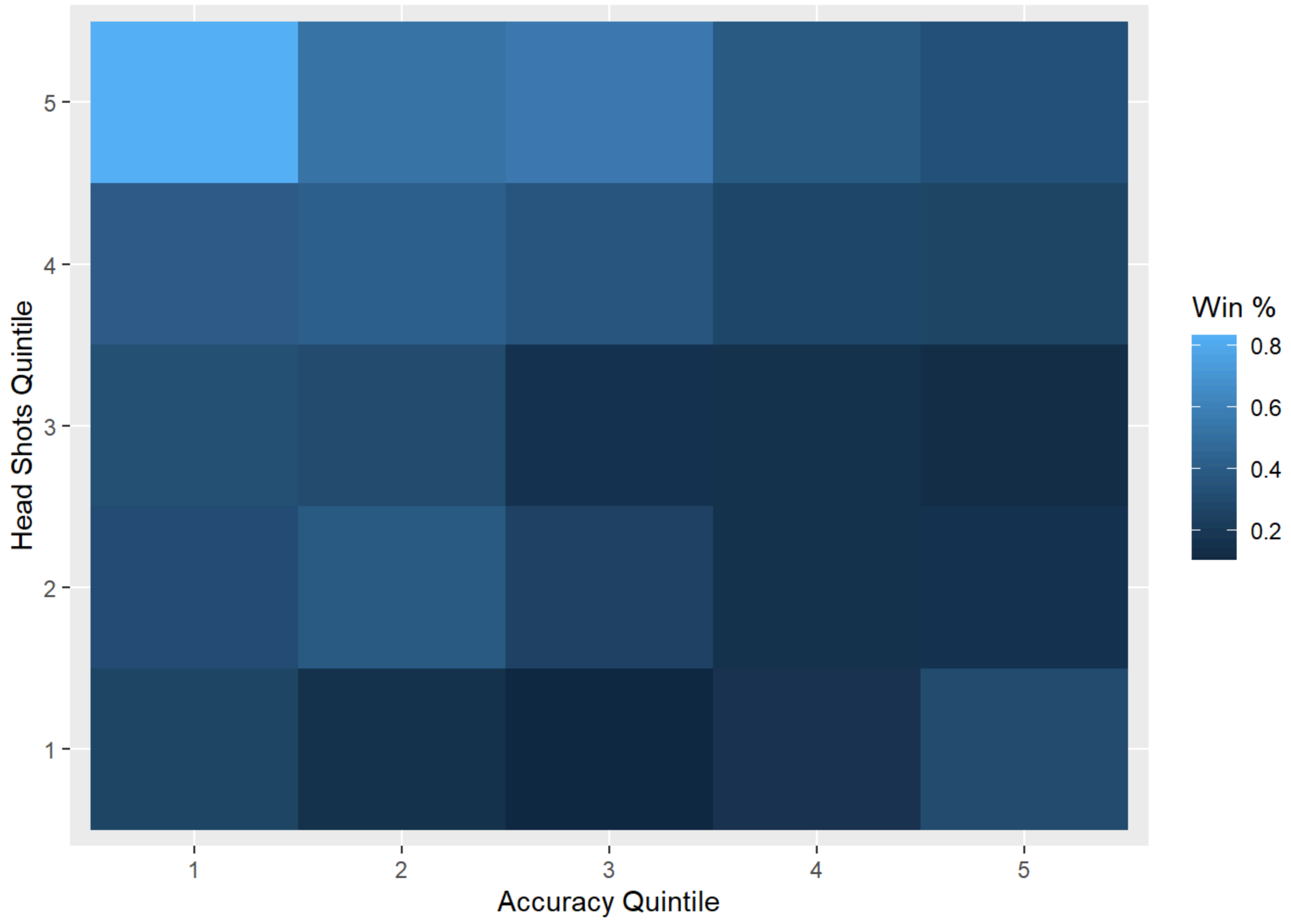
```
# Multivariate: head_shots
fn %>%
  ggplot(aes(x = head_shots,fill = factor(won))) +
  geom_density(alpha = .3) +
  labs(x = 'Head Shots',
       y = 'Density',
       fill = 'Won')
```



[RUBRIC: +.2 for correct plots.]

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?

```
# 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()) %>%
  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 %')
```



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. [RUBRIC: +.2 for correct plot. +.1 for correct written answer.]

Question 2 [1 point]

Now let's run a linear model and evaluate it in terms of overall accuracy, sensitivity and specificity using a threshold of 0.5.

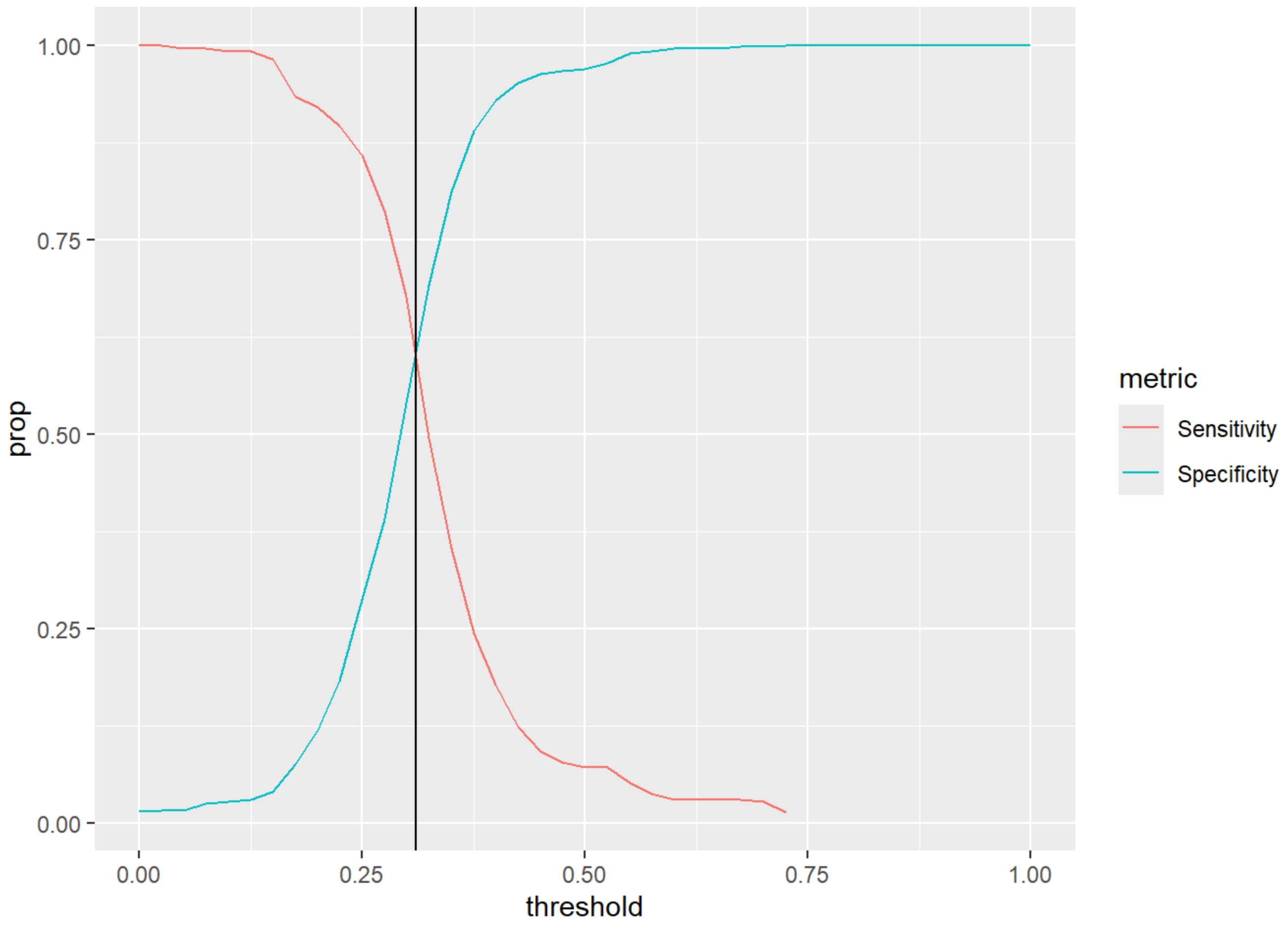
```
# Running linear model
model_lm <- lm(formula = won ~ accuracy + head_shots, # Define the regression equation
               data = fn) # Provide the dataset

# 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()) %>%
  mutate(prop = nGames / total_games) %>% # Calculate the proportion of game by the total games
  ungroup() %>%
  mutate(accuracy = sum((won == pred_win)*nGames) / sum(nGames)) # Calculate the overall accurac
y
```

```
## # A tibble: 4 × 6
##     won pred_win total_games nGames   prop accuracy
##   <dbl>    <dbl>       <int>  <int>  <dbl>    <dbl>
## 1     0        0         666    646 0.970     0.697
## 2     0        1         666     20 0.0300    0.697
## 3     1        0         291    270 0.928     0.697
## 4     1        1         291     21 0.0722    0.697
```

[RUBRIC: +.1 running correct model. +.2 points for calculating accuracy, sensitivity, and specificity table.]

Then, determine the threshold that maximizes both specificity and sensitivity by creating a sentivity-vs-specificity plot and adding a vertical line to pinpoint where the two measures intersect.

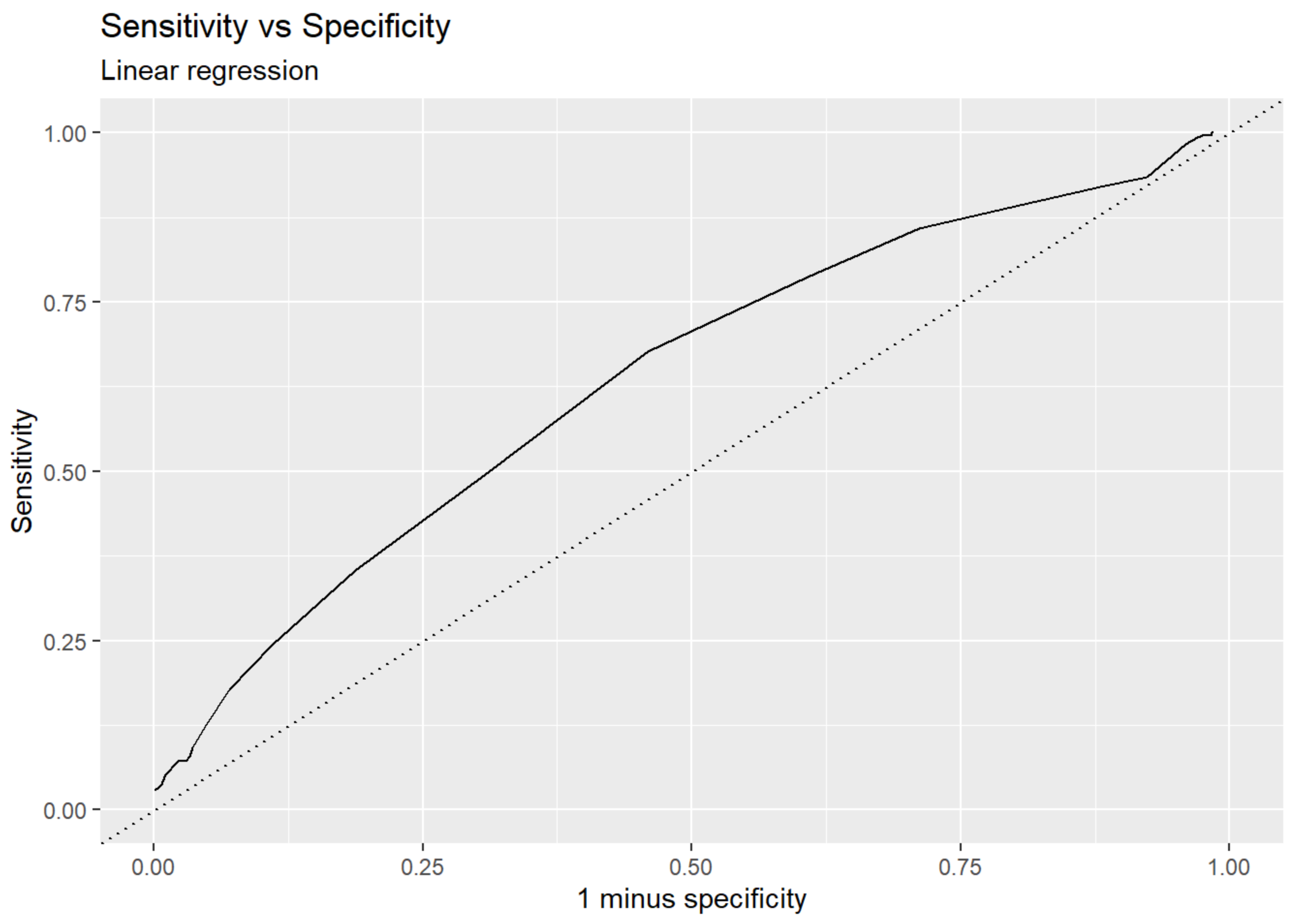
```
# Create the sensitivity vs specificity plot
toplot <- NULL # Instantiate an empty object
for(thresh in seq(0,1,by = .025)) {
  toplot <- fn %>%
  mutate(prob_win = predict(model_lm)) %>% # Calculate the probability of winning
  mutate(pred_win = ifelse(prob_win > thresh,1,0)) %>% # Convert the probability 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()) %>%
  mutate(prop = nGames / total_games) %>% # Calculate the proportion of game by the total games
  ungroup() %>%
  mutate(accuracy = sum((won == pred_win)*nGames) / sum(nGames)) %>% # Calculate the overall acc
uracy
  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)
```



The threshold that maximizes the sensitivity and specificity is about 0.31. [RUBRIC: +.2 point for creating correct plot. +.1 point for identifying threshold that minimizes trade-off.]

Finally, plot the area under the curve (AUC) and then calculate it. What grade would you give this model?

```
# 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
  pivot_wider(names_from = "metric",
              values_from = "prop") %>% # Pivot the data to wide format, where the new columns s
hould 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
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
```

> I would give this model a bad grade, since the AUC is only about 0.64. This is equivalent to a D grade. [RUBRIC: +.2 points for correct visualization of the AUC plot. +2 points for correct interpretation of AUC score.]

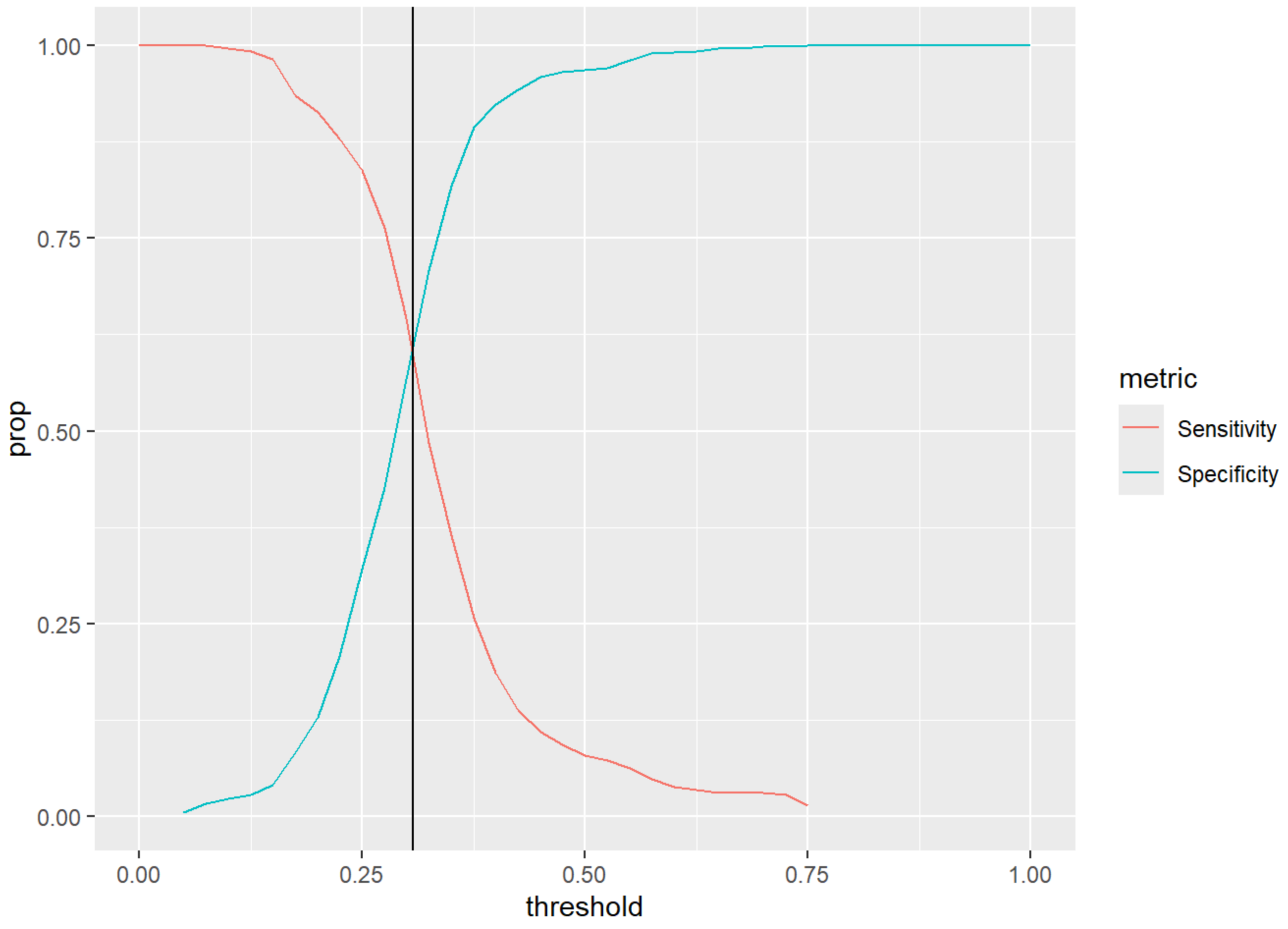
Question 3 [1 point]

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

```
# Running logit 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()) %>%
  mutate(prop = nGames / total_games) %>% # Calculate the proportion of game by the total games
  ungroup() %>%
  mutate(accuracy = sum((won == pred_win)*nGames) / sum(nGames)) # Calculate the overall accurac
y
```

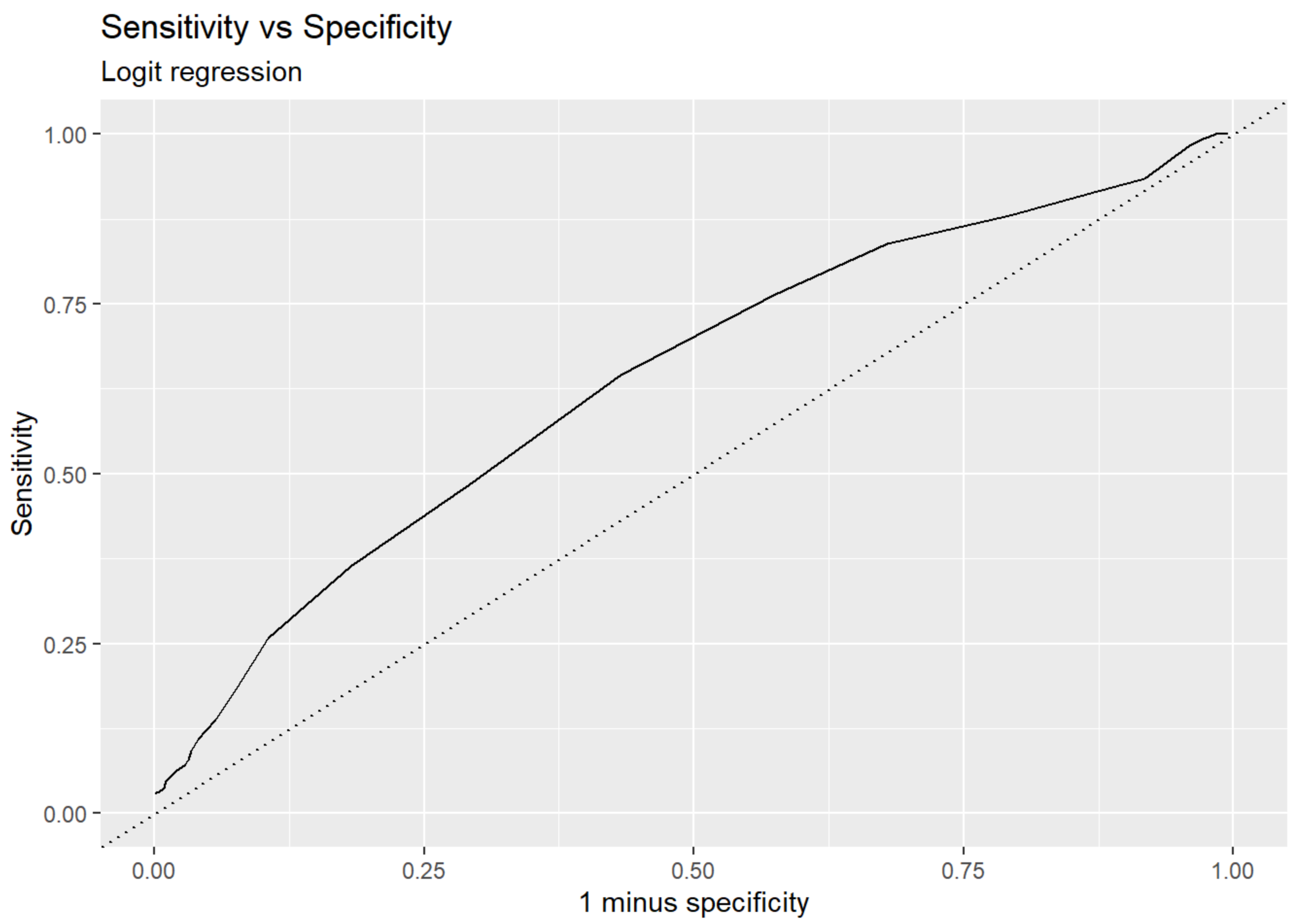
```
## # A tibble: 4 × 6
##     won pred_win total_games nGames   prop accuracy
##   <dbl>    <dbl>       <int>  <int>  <dbl>    <dbl>
## 1     0        0         666    645 0.968     0.698
## 2     0        1         666     21 0.0315    0.698
## 3     1        0         291    268 0.921     0.698
## 4     1        1         291     23 0.0790    0.698
```

```
# 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 win
ning (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 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()) %>%
  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 = .307)
```



```
# 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
  pivot_wider(names_from = "metric",
              values_from = "prop") %>% # Pivot the data to wide format, where the new columns s
hould 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 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
```

> 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. [RUBRIC: deduct -0.1 points for each mistake.]

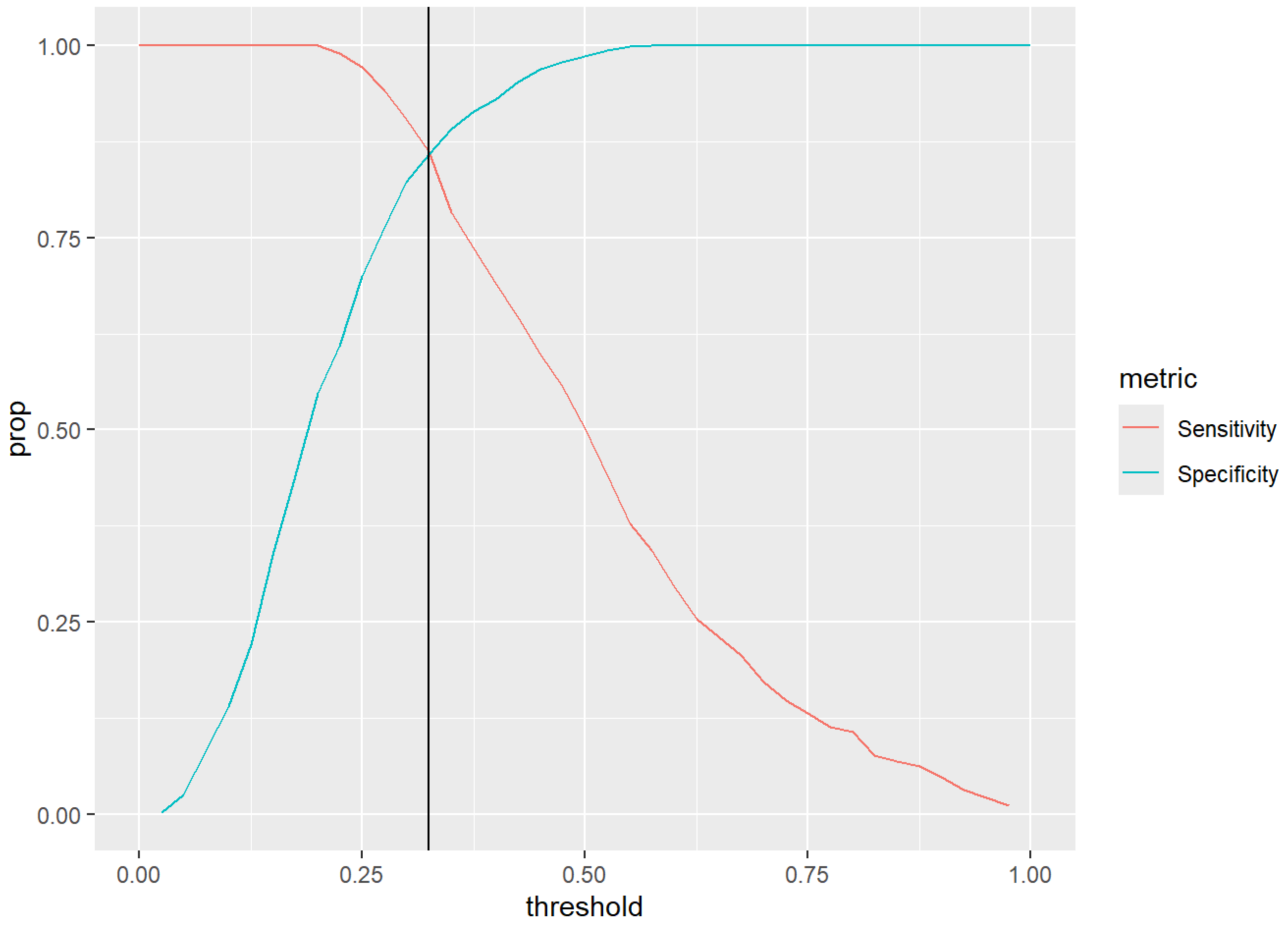
Question 4 [1 points]

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

```
# Running random forest model
model_rf <- ranger(formula = won ~ accuracy + head_shots, # Define the regression equation
               data = fn) # Specify the link function

# Calculating accuracy, sensitivity, and specificity
fn %>%
  mutate(prob_win = predict(model_rf,data = fn)$predictions) %>% # Calculate the probability of
winning (don't forget to use the $ sign to get the predictions for random forests!)
  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()) %>%
  mutate(prop = nGames / total_games) %>% # Calculate the proportion of game by the total games
  ungroup() %>%
  mutate(accuracy = sum((won == pred_win)*nGames) / sum(nGames)) # Calculate the overall accurac
y
```

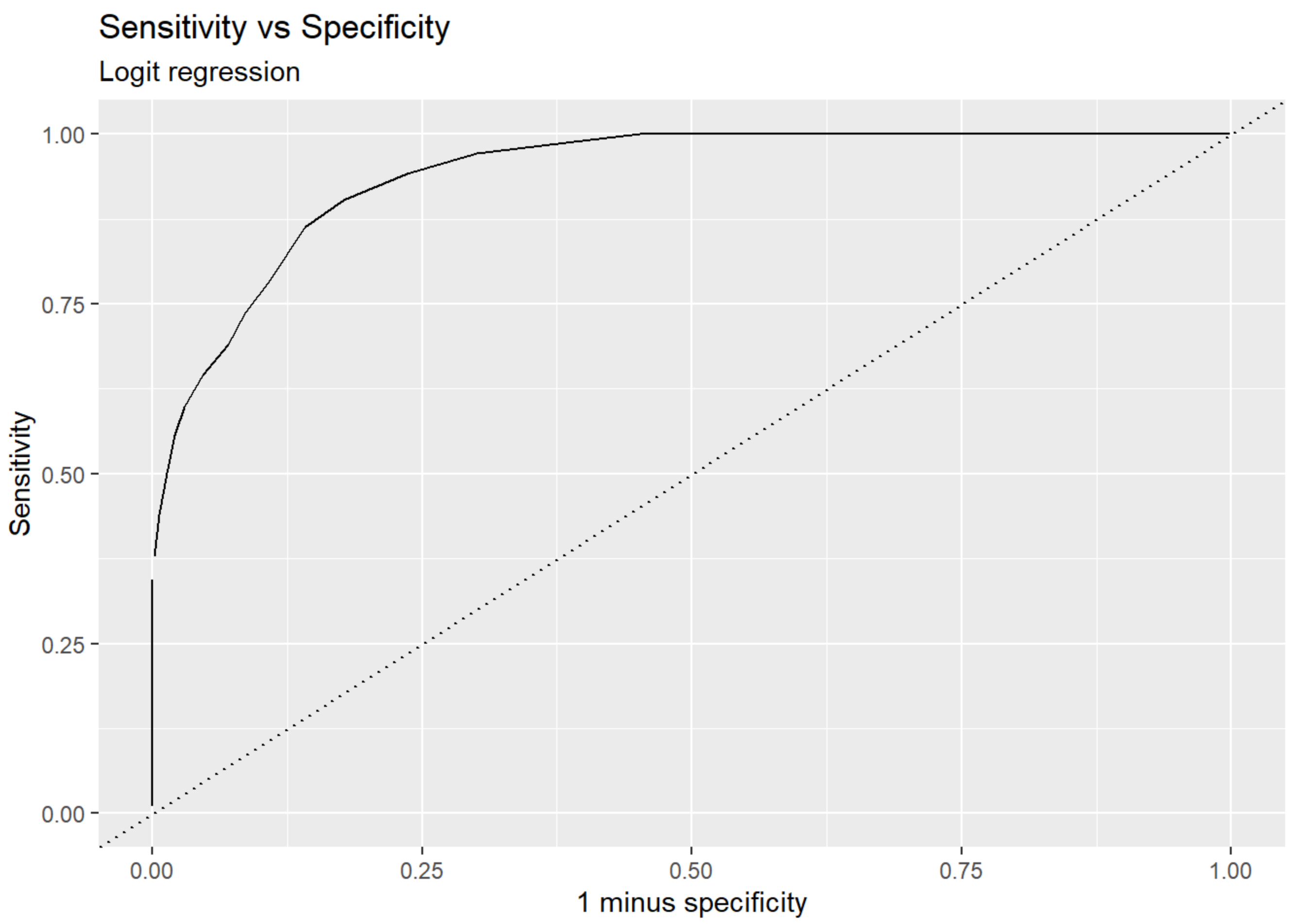
```
## # A tibble: 4 × 6
##     won pred_win total_games nGames   prop accuracy
##   <dbl>    <dbl>       <int>  <int>  <dbl>    <dbl>
## 1     0        0         666    657 0.986     0.839
## 2     0        1         666      9 0.0135    0.839
## 3     1        0         291    145 0.498     0.839
## 4     1        1         291    146 0.502     0.839
```

```
# 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_rf,data = fn)$predictions) %>% # Calculate the probability of
winning (don't forget to use the $ sign to get the predictions for random forests!)
  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()) %>%
  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 = .325)
```



```
# 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
  pivot_wider(names_from = "metric",
              values_from = "prop") %>% # Pivot the data to wide format, where the new columns s
hould 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 1 row containing missing values or values outside the scale range
## (`geom_line()`).
```



```
# Calculate the AUC
forAUC <- fn %>%
  mutate(prob_win = predict(model_rf,data = fn)$predictions, # Generate predicted probabilities
of winning from our model (don't forget to use the $ sign to get the predictions for random fore
sts!)
         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 = 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         0.943
```

> Here, the threshold that maximizes the sensitivity and specificity is about 0.33. The random forest model dramatically improves on the linear and logit models with an AUC of 0.942. [RUBRIC: Deduct -0.1 points for each mistake.]

Question 5 [1 point]

Use 100-fold cross validation with a 60-40 split to calculate the average AUC for all three models. Which is better?

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

  # Training models
  mLM <- lm(formula = won ~ accuracy + head_shots,
            data = train)
  mGLM <- glm(formula = won ~ accuracy + head_shots,
            data = train,
            family = binomial(link = 'logit'))
  mRF <- ranger(formula = won ~ accuracy + head_shots,
                data = train)

  # Predicting models
  toEval <- test %>%
    mutate(mLMPreds = predict(mLM,newdata = test),
           mGLMPreds = predict(mGLM,newdata = test,type = 'response'),
           mRFPreds = predict(mRF,data = test)$predictions,
           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)

  rocRF <- roc_auc(toEval,truth,mRFPreds) %>%
    mutate(model = 'rf') %>%
    rename(auc = .estimate)

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

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

```
## # A tibble: 3 × 2
##   model  mean_auc
##   <chr>     <dbl>
## 1 linear    0.636
## 2 logit     0.636
## 3 rf        0.602
```

> It turns out that the random forest model's AUC is actually lower than the other two models when calculating via cross validation. [RUBRIC: +0.1 for correct set-up (i.e., set.seed, instantiate empty object, and correctly splitting the sample by 60-40%). +0.1 for correct training stage (i.e., correct functions, correct data). +0.2 for correct prediction stage (i.e., correctly specifying how to predict an lm(), a glm(), and a ranger() model). +0.3 for correct results from cross validation code. +0.3 points for correct written interpretation.]

Extra Credit [2 points]

Now let's re-do EVERYTHING from the previous questions using a "kitchen sink" model with many different predictors. We will use the following X variables:

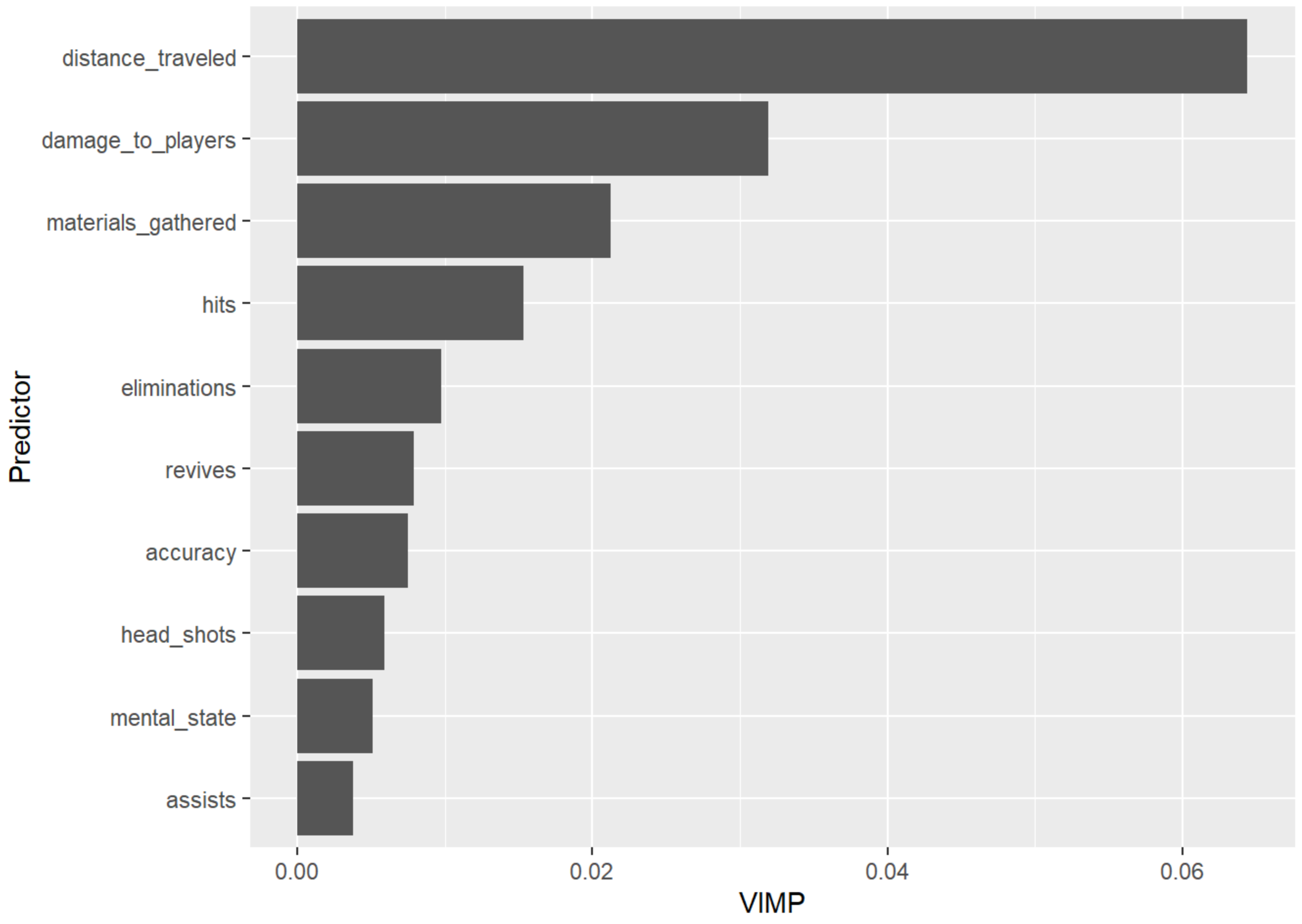
- `hits`
- `assists`
- `accuracy`
- `head_shots`
- `damage_to_players`
- `eliminations`
- `revives`
- `distance_traveled`
- `materials_gathered`
- `mental_state`

Which of these variables do you think will be **most important** *for predicting whether the player wins the game? Why? What about the variables that are* **least important***? Why?*

> I think that `eliminations` should be the most important for predicting whether the player wins the game because these mean that the player kills other players, which is the point of the game. I think the `distance_traveled` should be least important, since this doesn't seem to have anything to do with how to win the game. [RUBRIC: +0.5 for a coherent written explanation about which variables will be most and least important.]

Now run a random forest that uses all of these variables as predictors 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?

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

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')
```



Interestingly, `distance_traveled` is the **most** important predictor, perhaps because the more distance you travel, the longer you have stayed alive? Meanwhile `eliminations is in fifth place, and well below the variable importance of the other predictors. [RUBRIC: +0.5 for a correctly calculated and visualize variable importance result and written interpretation (although the explanation for why distance_traveled is most important is not necessary).]

What is the overall AUC from your random forest model using all these variables?

```
toEval <- fn %>%
  mutate(prob_win = predict(model_rf,data = fn)$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          1.00
```

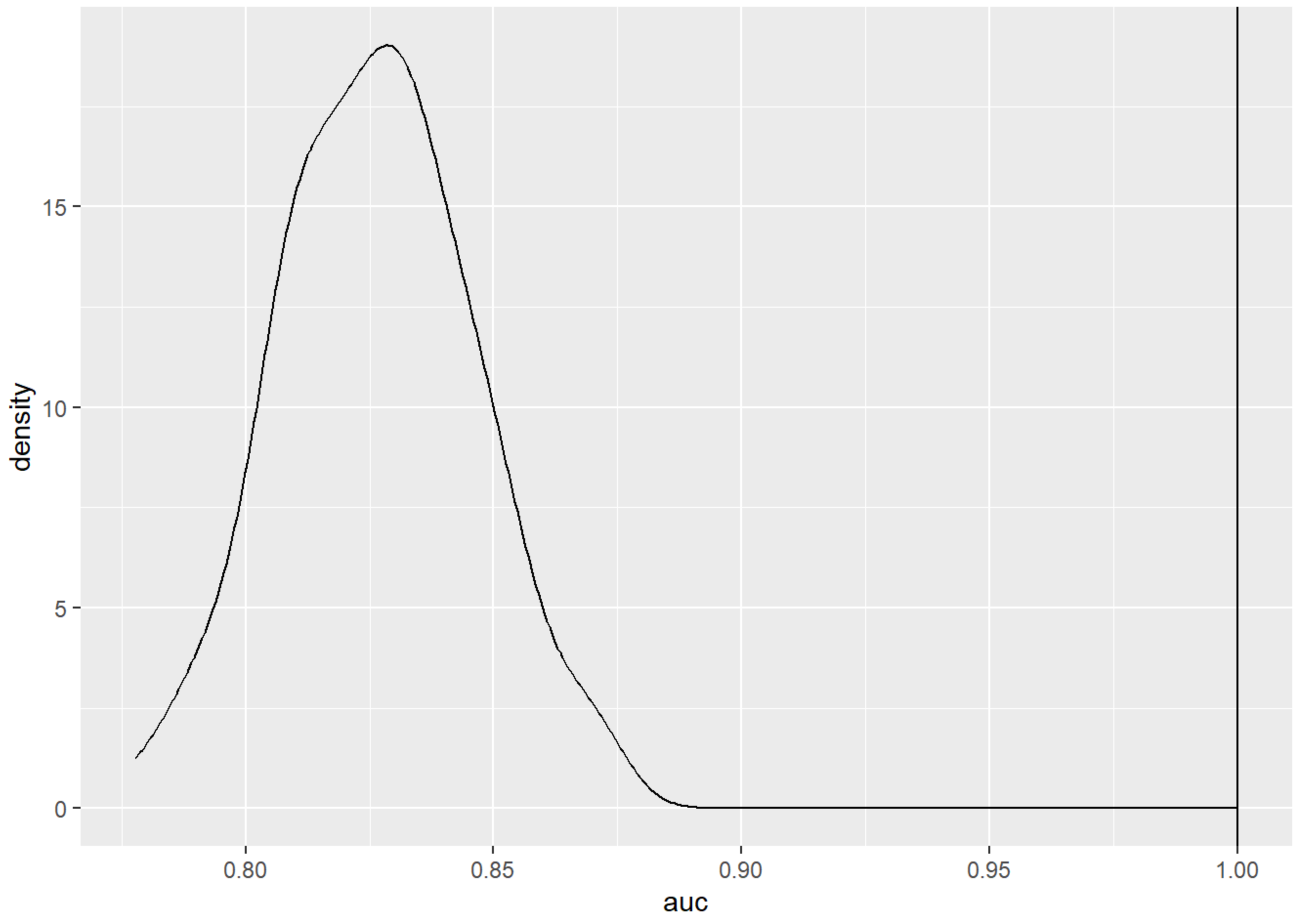
> The AUC is 1, meaning that our model perfectly predicts every observation. [RUBRIC: +0.3 for correctly calculated AUC for the random forest model.]

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?

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

  # 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                               1
```

The average across 100 60-40 cross validated splits is 0.826. This is lower than the AUC for the full model in 100 out of 100 cross validated splits. There is thus substantial evidence that the random forest model overfits to the data, highlighting the importance of cross validation. [RUBRIC: +0.5 for correctly executed cross validation loop. +0.2 for correctly calculated proportion of cross validated runs that are less than the AUC for the full model, and correct written interpretation of the result.]