Lecture 10 Notes

2024-07-17

On Zoom!

To get started:

1. Require `tidyverse`
2. Load the Fortnite data `fn_cleaned_final.rds` from Github

```
require(tidyverse)
```

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

```
## ── Attaching core tidyverse packages ──────────────────────── tidyverse 2.0.0 ──
## ✓ dplyr     1.1.4     ✓ readr     2.1.5
## ✓ forcats   1.0.0     ✓ stringr   1.5.1
## ✓ ggplot2   3.5.1     ✓ tibble    3.2.1
## ✓ lubridate 1.9.3     ✓ tidyr     1.3.1
## ✓ 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 t
o become errors
```

```
fn <- read_rds("https://github.com/jbisbee1/ISP_Data_Science_2024/raw/main/data/fn_clean
ed_final.rds")
```

What is a binary variable?

- Takes on only one of two values: i.e., 0 or 1/ pass or fail / won or lost / rookie or not rookie

```
summary(fn %>%
  select(won))
```

```
##       won
##  Min.   :0.0000
##  1st Qu.:0.0000
##  Median :0.0000
##  Mean   :0.3041
##  3rd Qu.:1.0000
##  Max.   :1.0000
```

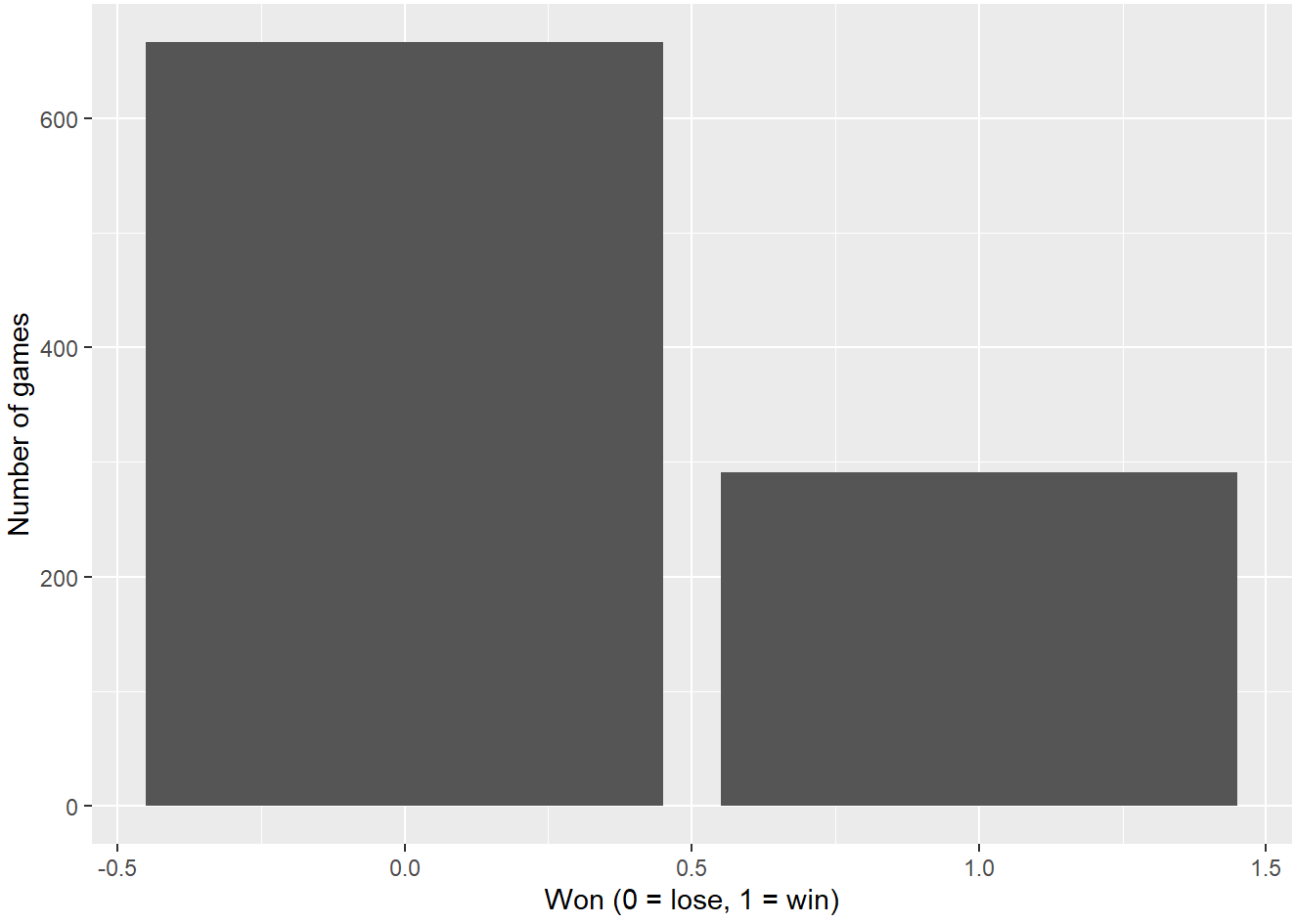
```
fn %>%
  summarise(prob_win = mean(won))
```

```
## # A tibble: 1 × 1
##   prob_win
##      <dbl>
## 1    0.304
```

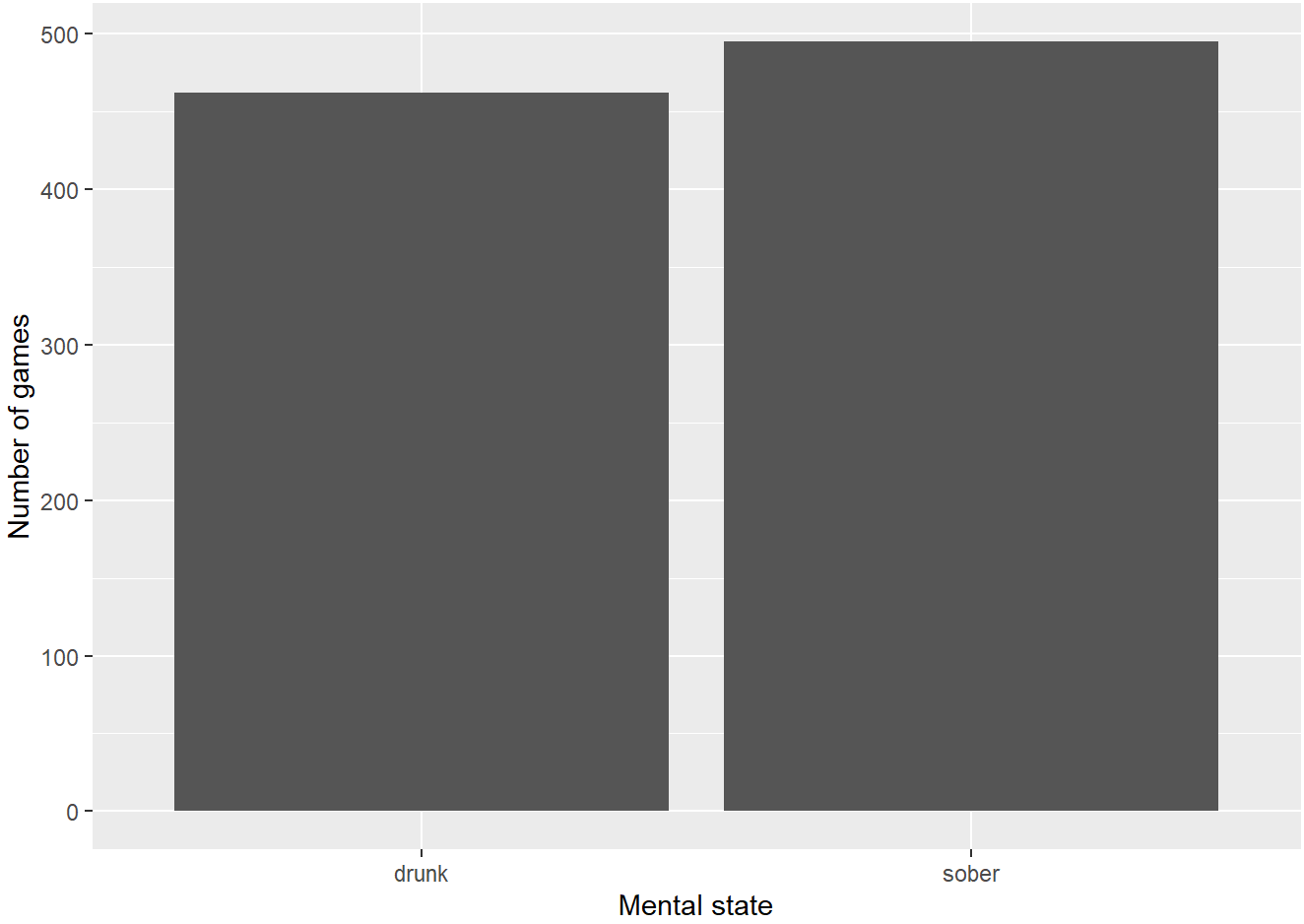
RQ: does the mental state predicting the probability of winning?

- Univariate visualization of both X and Y variable
    - X: mental state
    - Y: winning (0,1)

```
# Univariate visualization of the Y
fn %>%
  ggplot(aes(x = won)) +
  geom_bar() +
  labs(x = 'Won (0 = lose, 1 = win)',
       y = 'Number of games')
```



```
# Univariate visualization of the X
fn %>%
  ggplot(aes(x = mental_state)) +
  geom_bar() +
  labs(x = "Mental state",
       y = "Number of games")
```



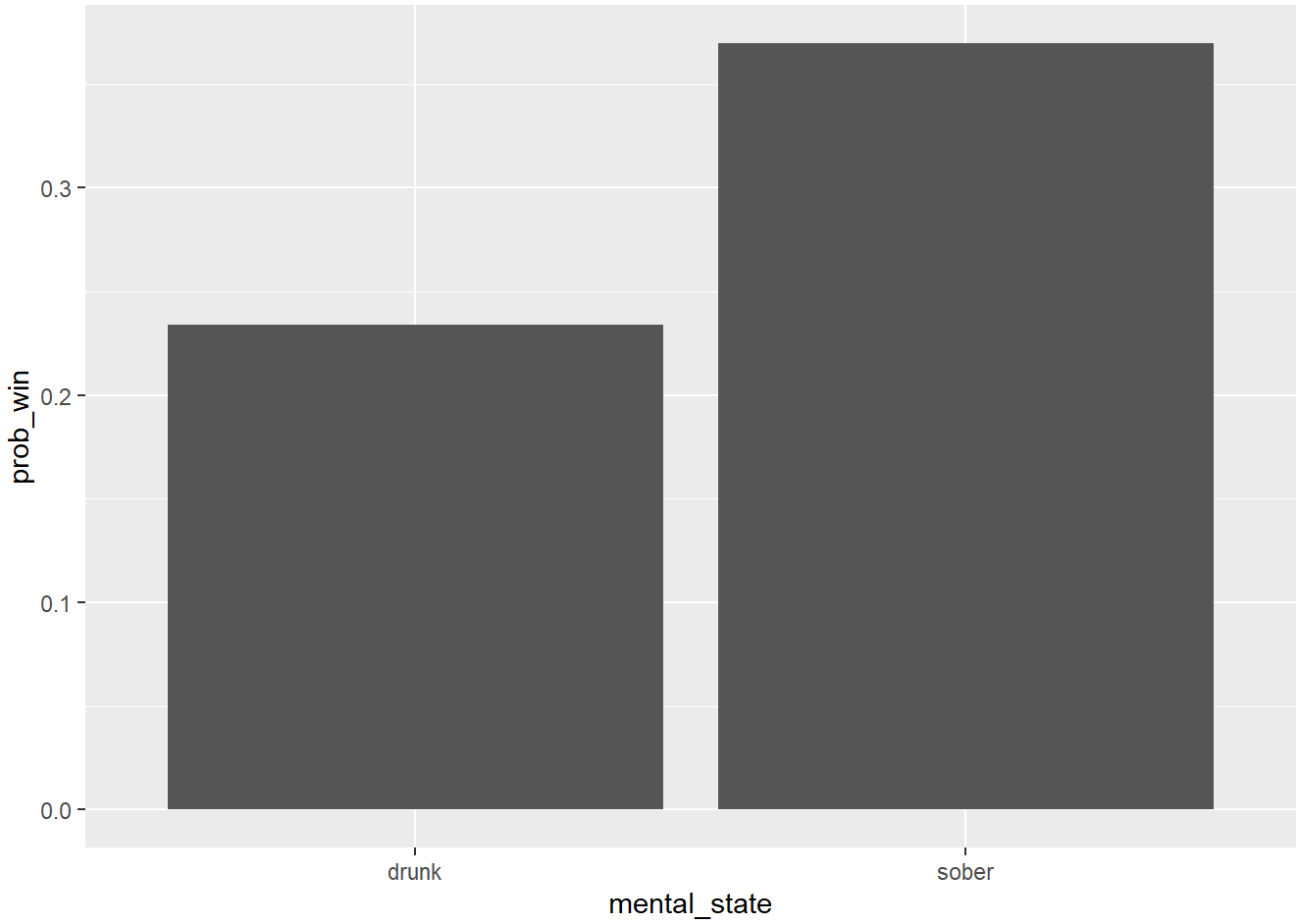
Multivariate analysis

- Calculate probability of winning **by** the mental state

```
fn %>%
  group_by(mental_state) %>%
  summarise(prob_win = mean(won))
```

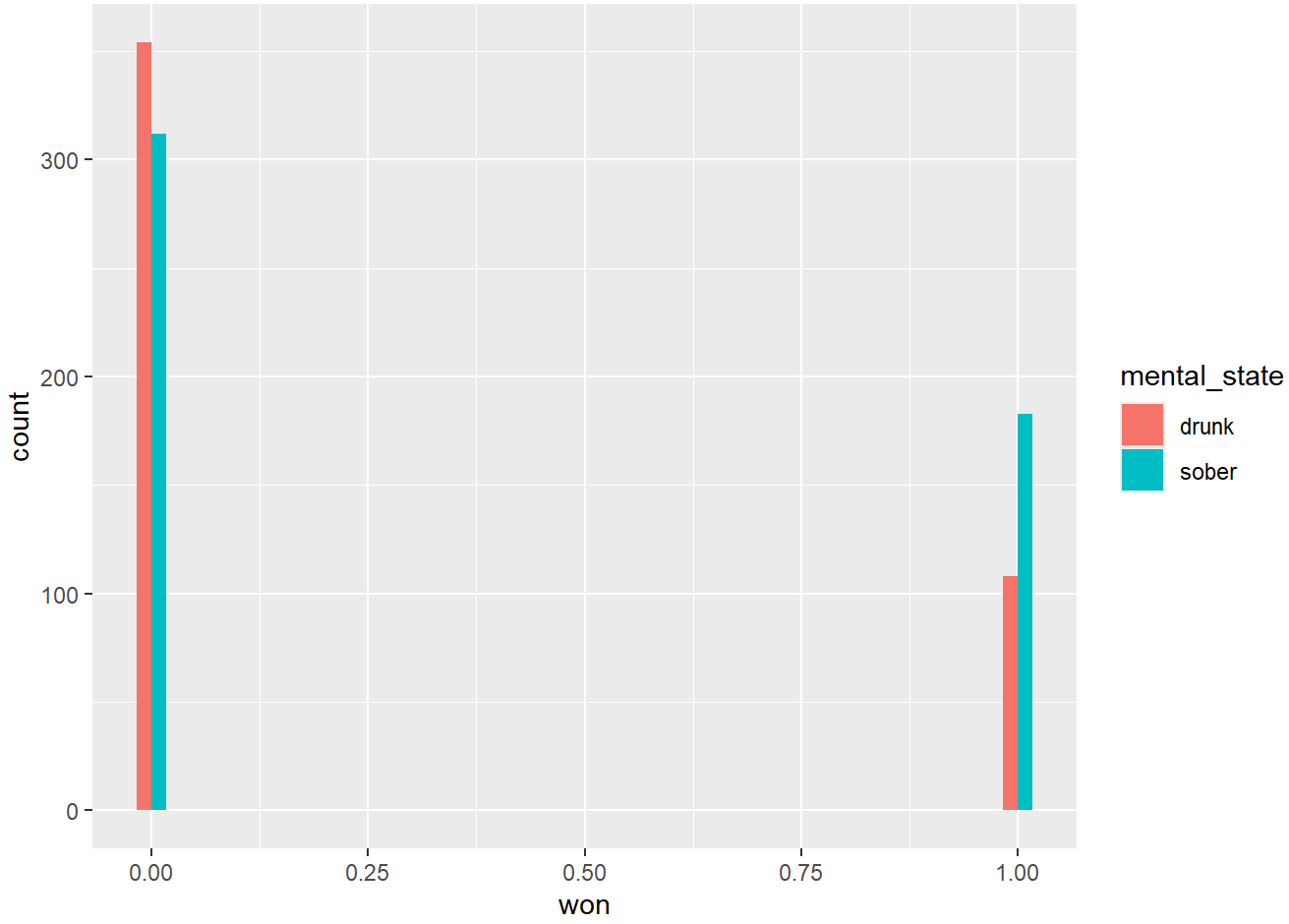
```
## # A tibble: 2 × 2
##   mental_state prob_win
##   <chr>           <dbl>
## 1 drunk           0.234
## 2 sober           0.370
```

```
# Multivariate visualization
# Option 1: geom_bar()
fn %>%
  group_by(mental_state) %>%
  summarise(prob_win = mean(won)) %>%
  ggplot(aes(x = mental_state,
             y = prob_win)) +
  geom_bar(stat = 'identity')
```

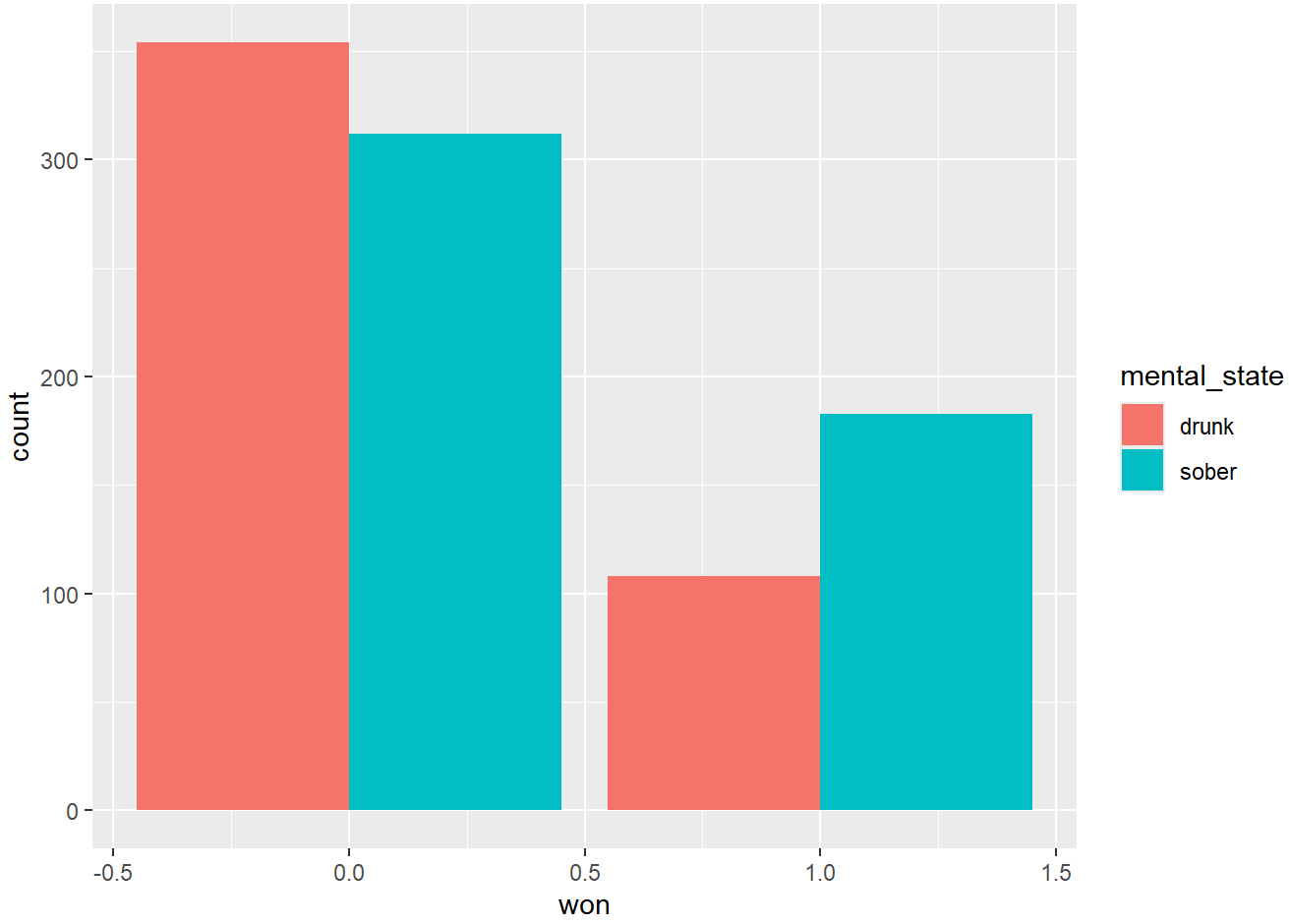



```
# Option 2: geom_density() / geom_histogram() + fill
fn %>%
  ggplot(aes(x = won,
             fill = mental_state)) +
  geom_histogram(position = 'dodge')
```

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




```
# Option 3: geom_bar() + fill
fn %>%
  ggplot(aes(x = won,
             fill = mental_state)) +
  geom_bar(position = 'dodge')
```



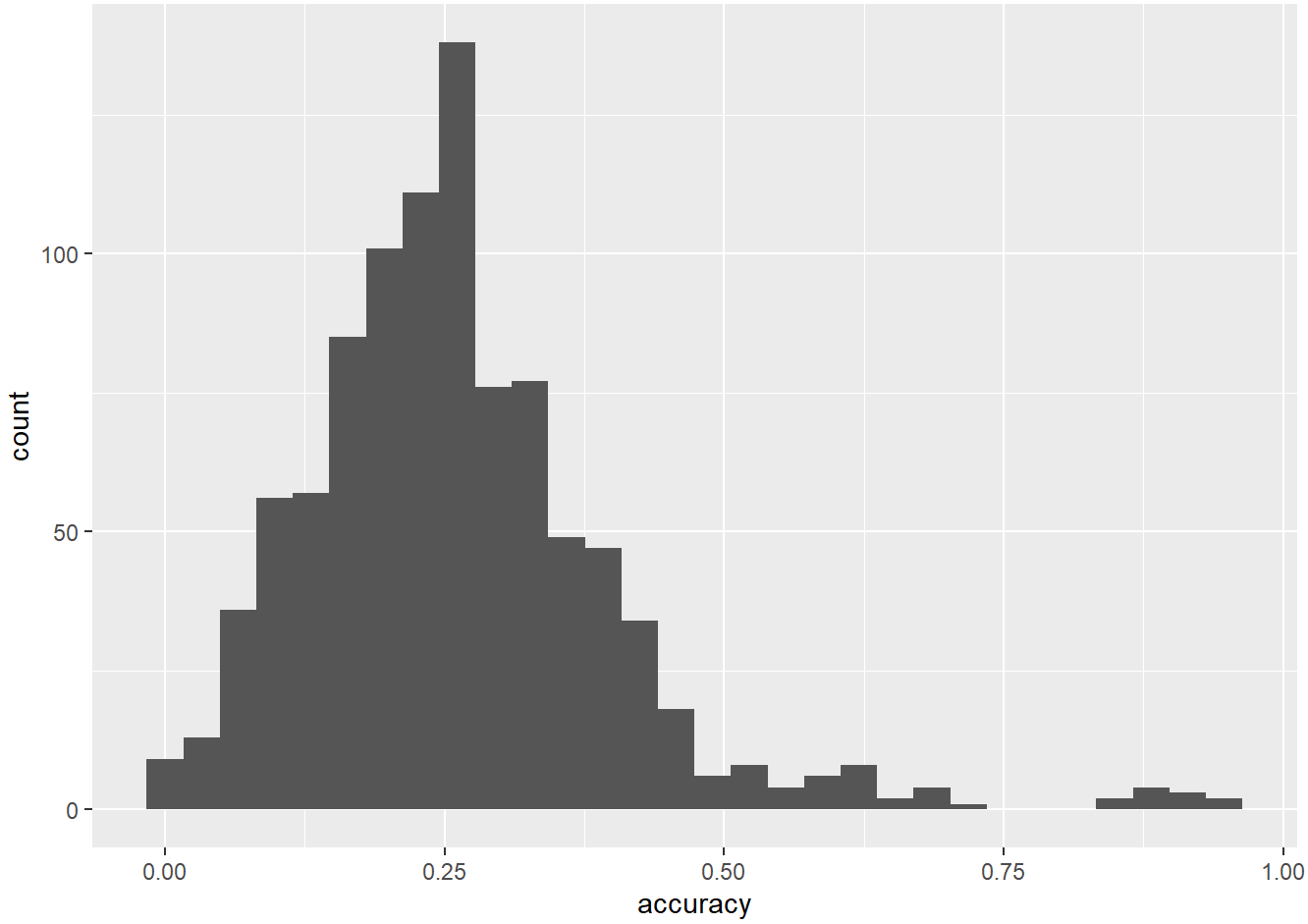
Add one more X variable: accuracy

```
fn %>%
  select(accuracy)
```

```
## # A tibble: 957 × 1
##    accuracy
##       <dbl>
##  1    0.194
##  2    0.324
##  3    0.337
##  4    0.105
##  5    0.622
##  6    0.0582
##  7    0.265
##  8    0.272
##  9    0.383
## 10    0.328
## # ℹ 947 more rows
```

```
fn %>%
  ggplot(aes(x = accuracy)) +
  geom_histogram()
```

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



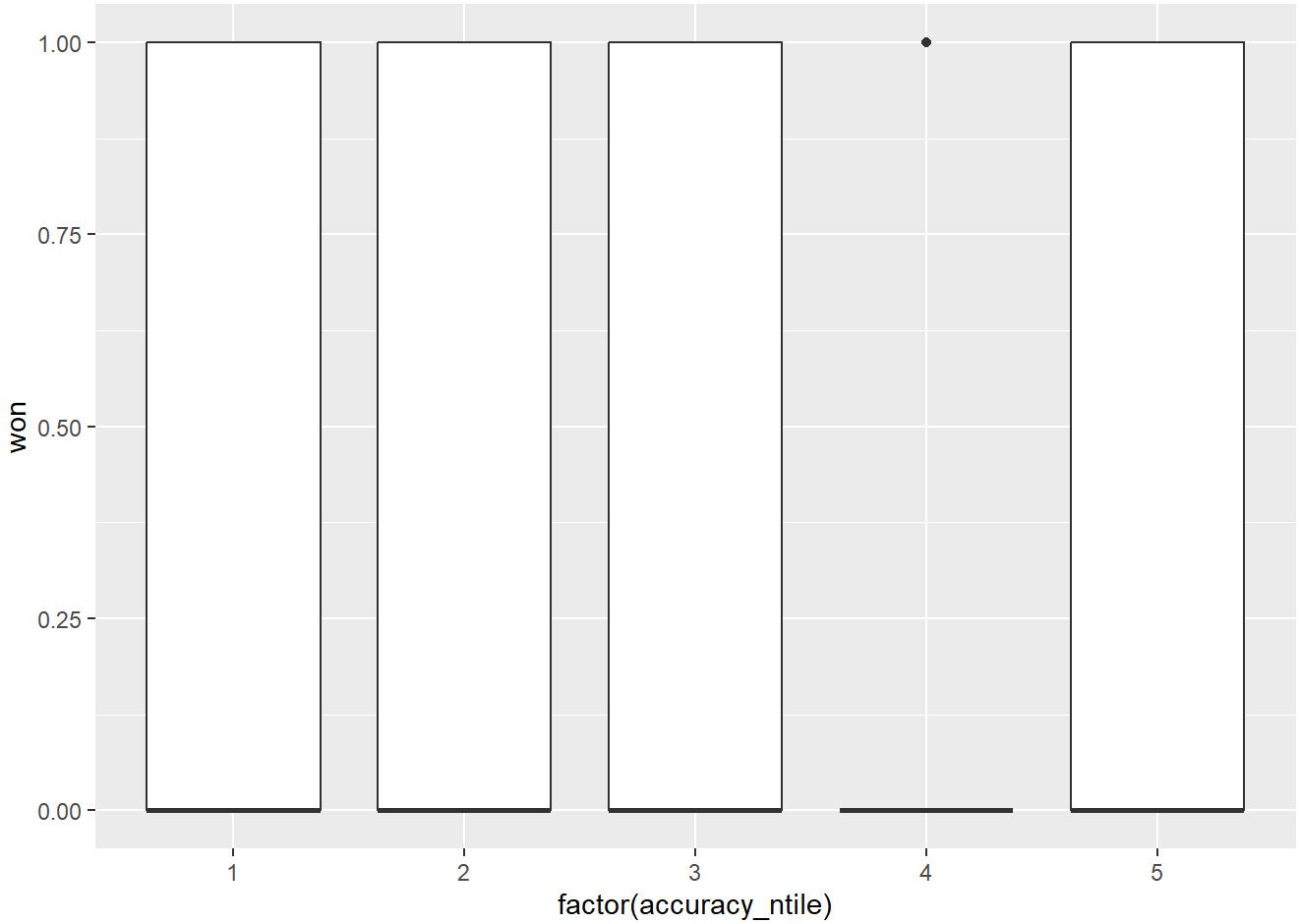
Trivariate Visualization

- Y: `won`
- X_1: `mental_state`
- X_2: `accuracy`
- `geom_tile()`

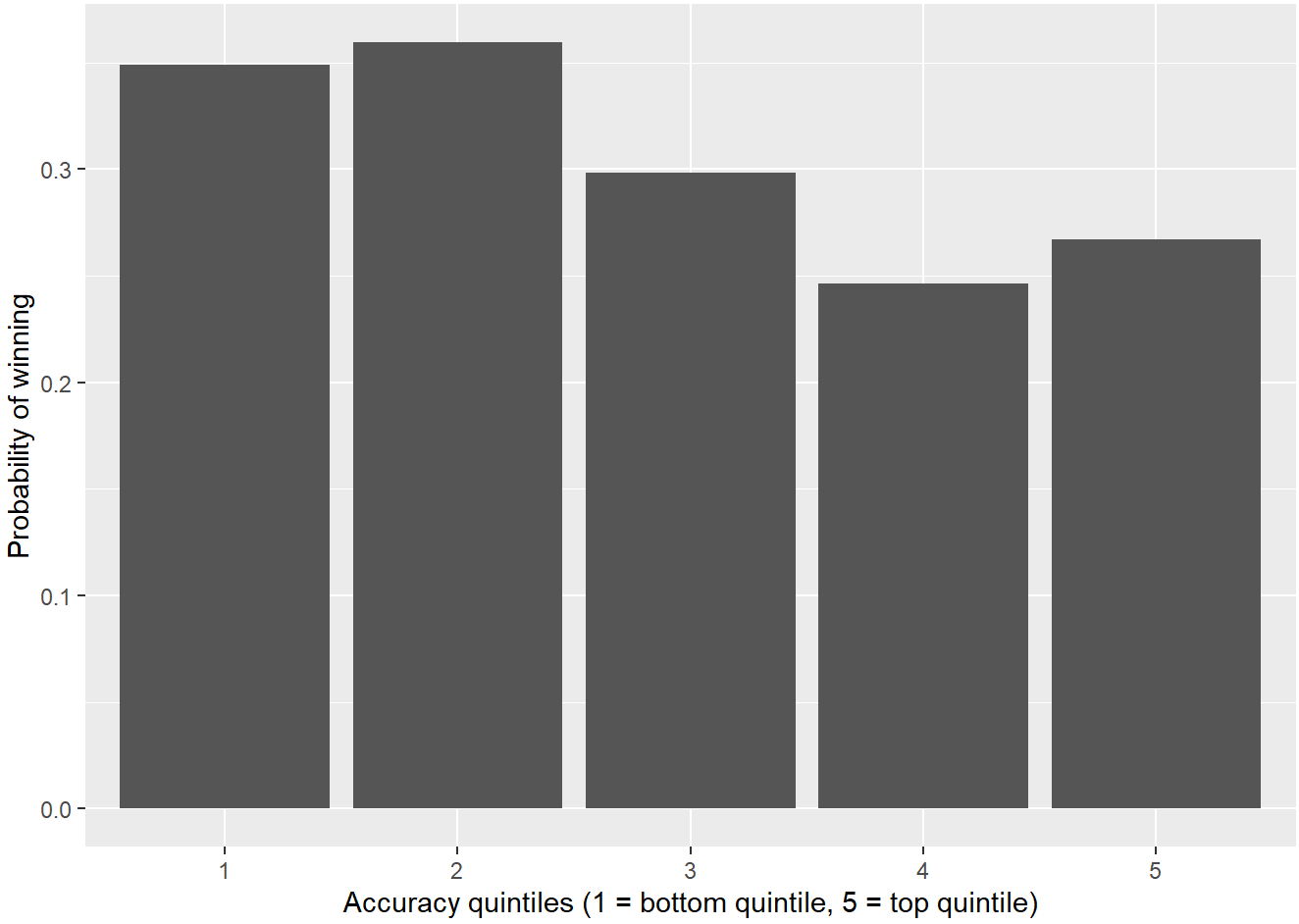
```
# Step 1: convert numeric to categorical
# Child function: ntile()
fn <- fn %>%
  mutate(accuracy_ntile = ntile(accuracy, n = 5))

# Calculate probability of winning by categorical accuracy
fn %>%
  group_by(accuracy_ntile) %>%
  summarise(prob_win = mean(won))
```

```
## # A tibble: 5 × 2
##   accuracy_ntile prob_win
##            <int>    <dbl>
## 1              1    0.349
## 2              2    0.359
## 3              3    0.298
## 4              4    0.246
## 5              5    0.267
```

```
# Multivariate visualization first:
# Bad approach
fn %>%
  ggplot(aes(x = factor(accuracy_ntile),
             y = won)) +
  geom_boxplot()
```




```
# Good approach
fn %>%
  group_by(accuracy_ntile) %>%
  summarise(prob_win = mean(won)) %>%
  ggplot(aes(x = factor(accuracy_ntile),
             y = prob_win)) +
  geom_bar(stat = 'identity') +
  labs(x = 'Accuracy quintiles (1 = bottom quintile, 5 = top quintile)',
       y = 'Probability of winning')
```

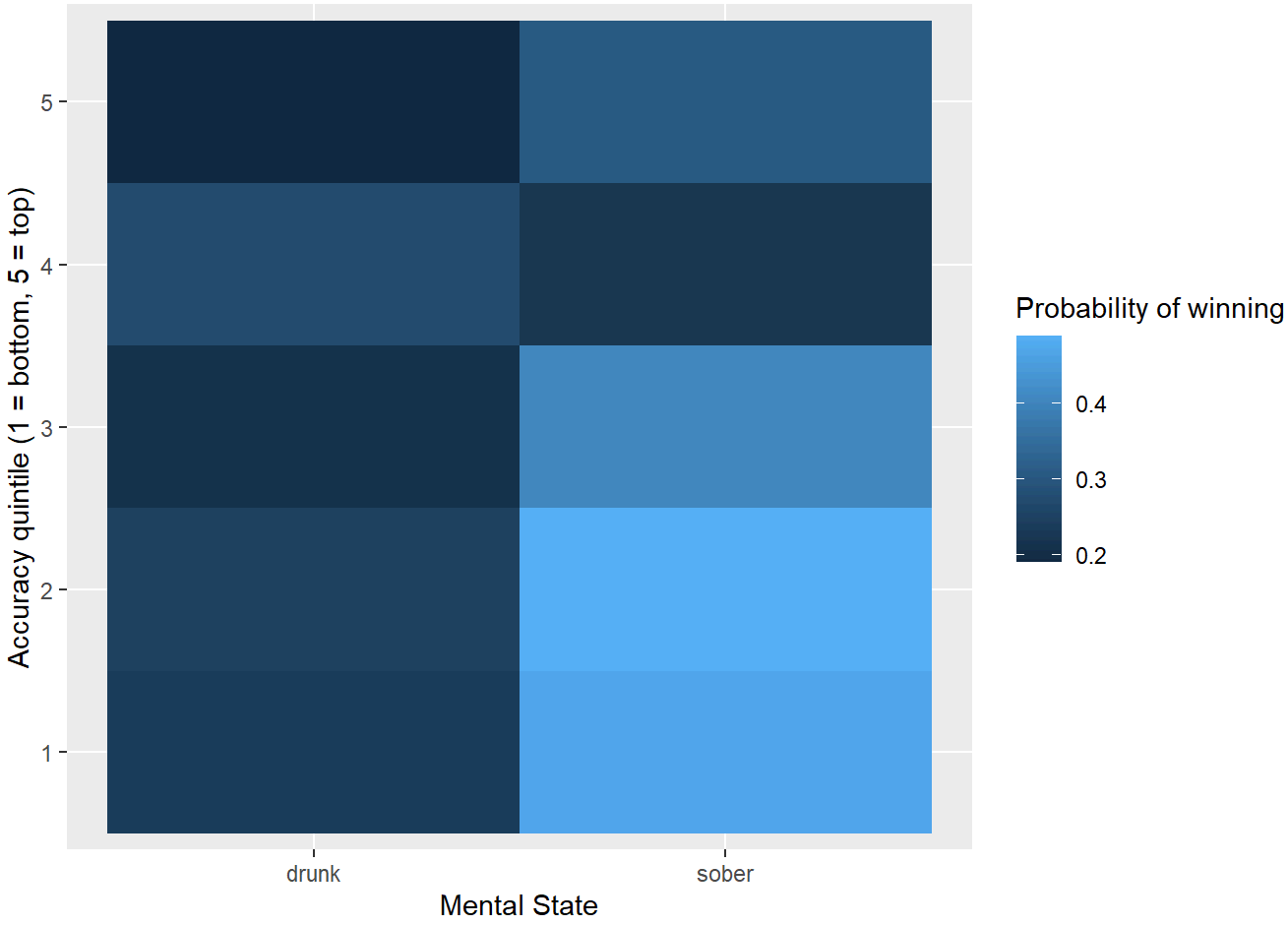


Trivariate visualization with geom_tile()

- Calculate probability of winning **by** mental state **and** accuracy quintile

```
fn %>%
  group_by(mental_state,
           accuracy_ntile) %>%
  summarise(prob_win = mean(won)) %>%
  ggplot(aes(x = mental_state,
             y = factor(accuracy_ntile),
             fill = prob_win)) +
  geom_tile() +
  labs(x = "Mental State",
       y = "Accuracy quintile (1 = bottom, 5 = top)",
       fill = "Probability of winning")
```

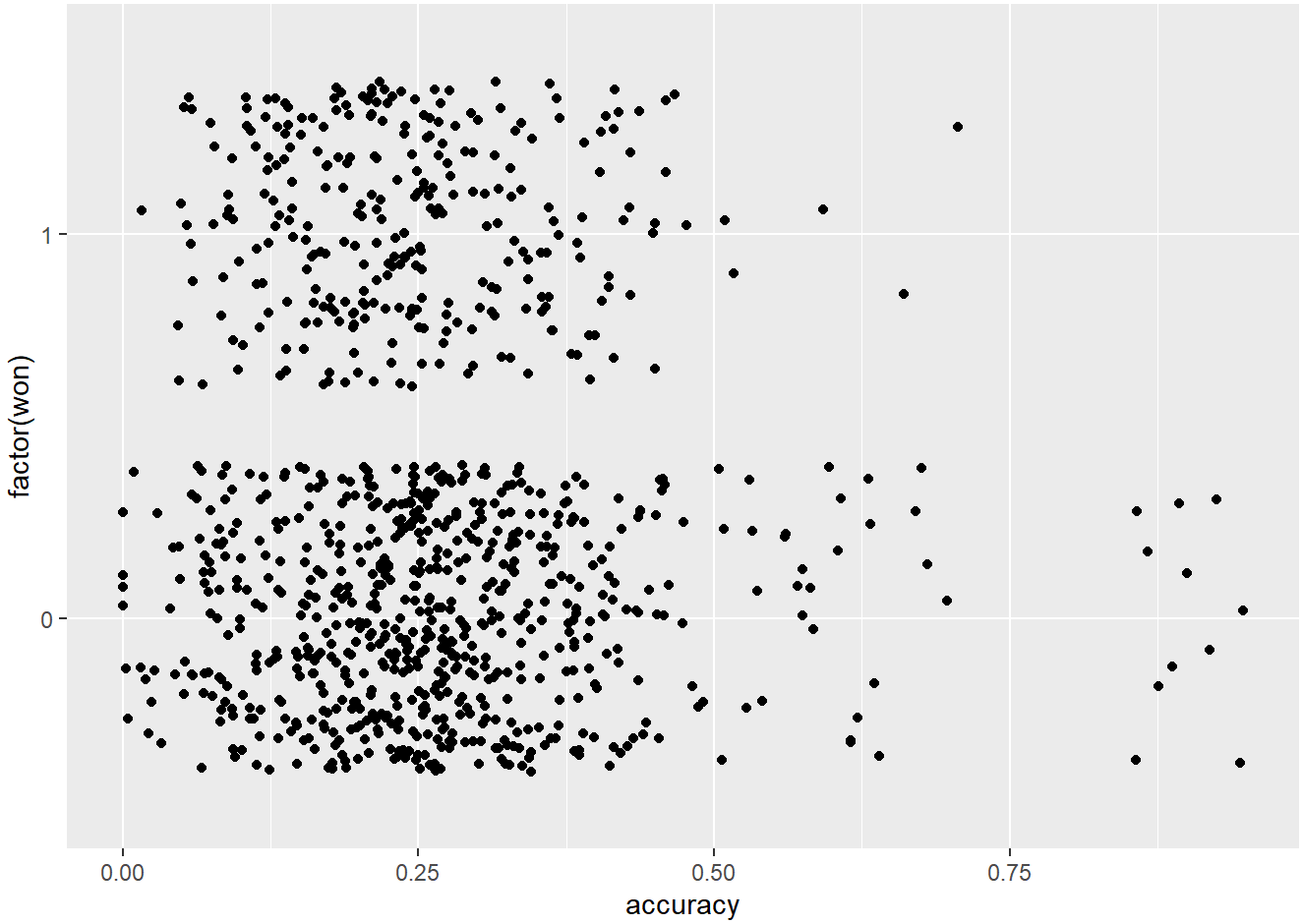
```
## `summarise()` has grouped output by 'mental_state'. You can override using the
## `.groups` argument.
```



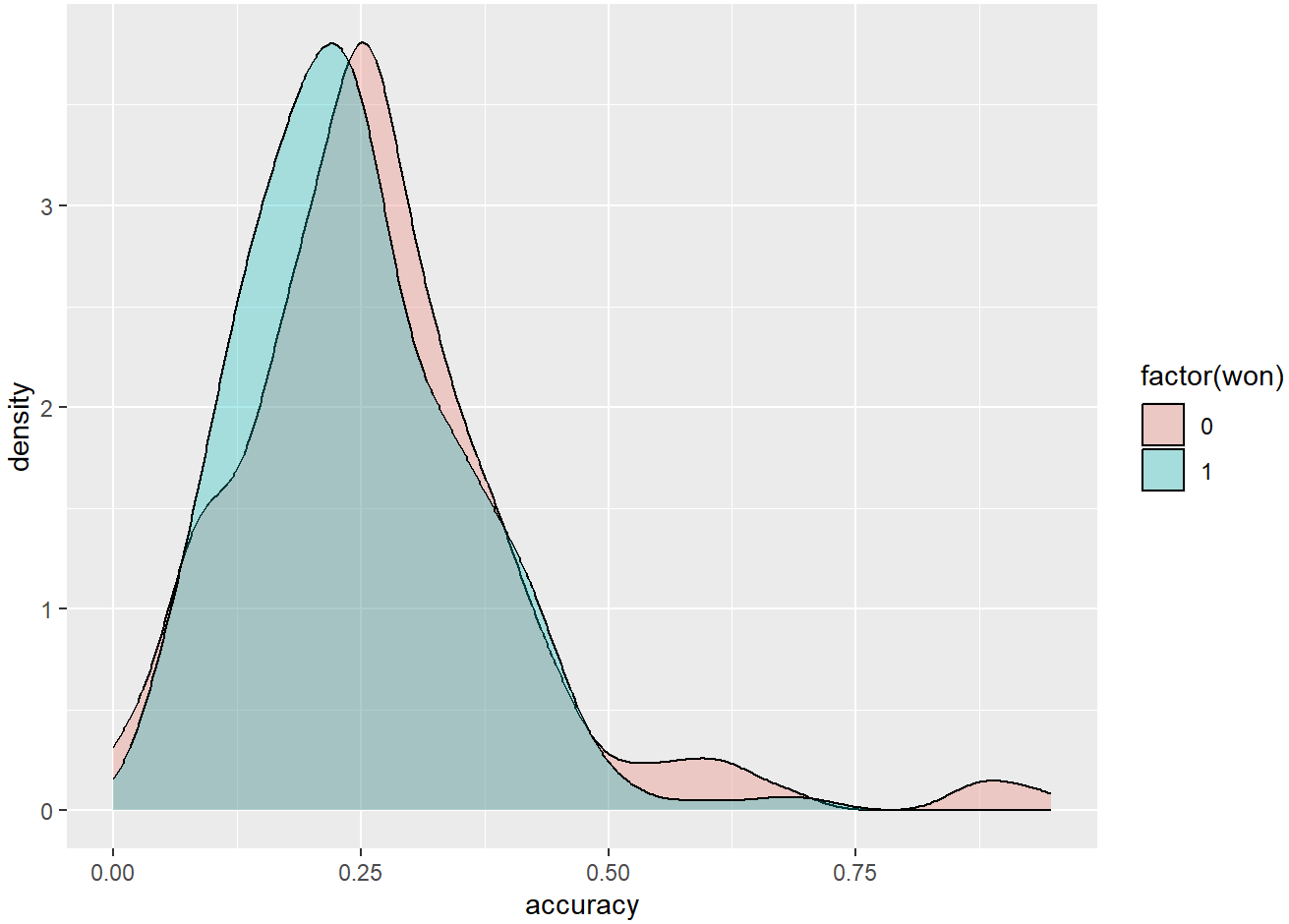
Quick aside: What about continuous X?

- Multivariate visualization of `accuracy` versus `won`

```
fn %>%
  ggplot(aes(y = factor(won),
             x = accuracy)) +
  # geom_boxplot()
  geom_jitter()
```




```
fn %>%
  ggplot(aes(x = accuracy,
             fill = factor(won))) +
  geom_density(alpha = .3)
```

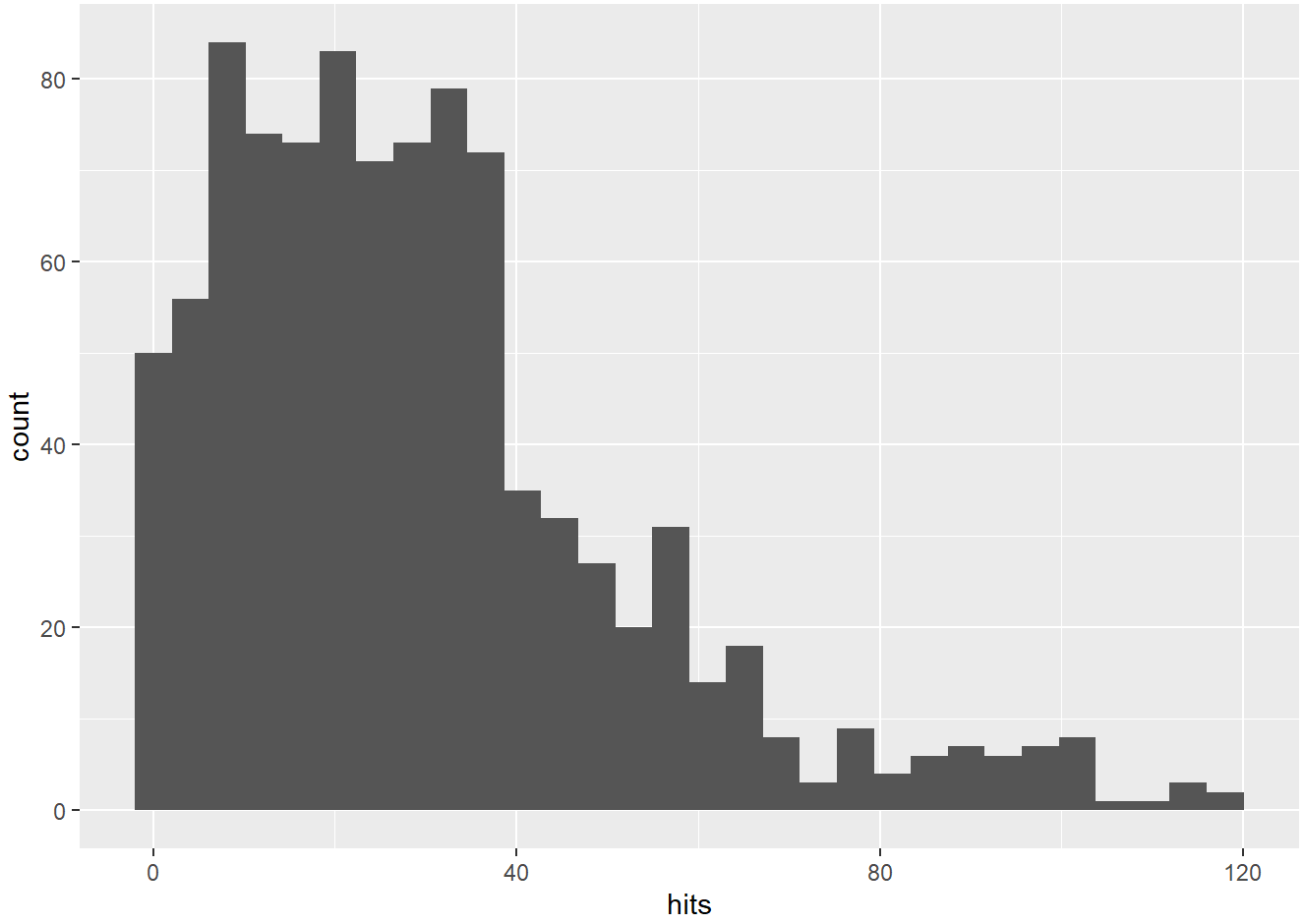


Accuracy, sensitivity and specificity

- Looking at `hits` instead of `accuracy` so I don't confuse students

```
fn %>%
  ggplot(aes(x = hits)) +
  geom_histogram()
```

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




```
fn <- fn %>%
  mutate(hits_quintile = ntile(hits,n = 5))

fn %>%
  group_by(hits_quintile,mental_state) %>%
  summarise(prob_win = mean(won))
```

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

```
## # A tibble: 10 × 3
## # Groups:   hits_quintile [5]
##    hits_quintile mental_state prob_win
##            <int> <chr>           <dbl>
##  1             1 drunk           0.16
##  2             1 sober           0.174
##  3             2 drunk           0.139
##  4             2 sober           0.274
##  5             3 drunk           0.215
##  6             3 sober           0.348
##  7             4 drunk           0.293
##  8             4 sober           0.474
##  9             5 drunk           0.38
## 10             5 sober           0.549
```

```
# Use mutate() instead of summarise() to ADD prob_win to the data
fn <- fn %>%
  group_by(hits_quintile,mental_state) %>%
  mutate(prob_win = mean(won)) %>%
  ungroup() # Best practices for clean coding

fn %>%
  select(hits_quintile,mental_state,won,prob_win)
```

```
## # A tibble: 957 × 4
##    hits_quintile mental_state   won prob_win
##            <int> <chr>        <dbl>    <dbl>
##  1             1 sober            0    0.174
##  2             2 sober            0    0.274
##  3             4 drunk            0    0.293
##  4             3 drunk            0    0.215
##  5             5 drunk            0    0.38
##  6             1 drunk            0    0.16
##  7             4 drunk            0    0.293
##  8             2 drunk            0    0.139
##  9             5 drunk            0    0.38
## 10             3 drunk            1    0.215
## # ℹ 947 more rows
```

Thresholds for prediction

- Goal: convert `prob_win` into 0 or 1
- Solution: choose a threshold value
    - Above this threshold, convert to 1
    - Below this threshold, convert to 0

```
fn <- fn %>%
  mutate(pred_win = ifelse(prob_win > .5,
                           1,
                           0))

fn %>%
  select(mental_state,hits_quintile,
         won,prob_win,pred_win)
```

```
## # A tibble: 957 × 5
##    mental_state hits_quintile   won prob_win pred_win
##    <chr>                <int> <dbl>    <dbl>    <dbl>
##  1 sober                    1     0    0.174        0
##  2 sober                    2     0    0.274        0
##  3 drunk                    4     0    0.293        0
##  4 drunk                    3     0    0.215        0
##  5 drunk                    5     0    0.38         0
##  6 drunk                    1     0    0.16         0
##  7 drunk                    4     0    0.293        0
##  8 drunk                    2     0    0.139        0
##  9 drunk                    5     0    0.38         0
## 10 drunk                    3     1    0.215        0
## # ℹ 947 more rows
```

- Calculate model accuracy

```
fn %>%
  group_by(won,pred_win) %>%
  summarise(nGames = n())
```

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

```
## # A tibble: 4 × 3
## # Groups:   won [2]
##     won pred_win nGames
##   <dbl>    <dbl>  <int>
## 1     0        0    625
## 2     0        1     41
## 3     1        0    241
## 4     1        1     50
```

```
# fn %>%
#   count(won,pred_win)

# Manually calculate accuracy
(625 + 50) / (625 + 41 + 241 + 50)
```

```
## [1] 0.7053292
```

```
# Programmatically calculate accuracy
fn %>%
  group_by(won,pred_win) %>%
  summarise(nGames = n()) %>%
  ungroup() %>% # BEST PRACTICES!
  mutate(nCorrect = (won == pred_win)*nGames) %>%
  mutate(accuracy = sum(nCorrect) / sum(nGames))
```

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

```
## # A tibble: 4 × 5
##     won pred_win nGames nCorrect accuracy
##   <dbl>    <dbl>  <int>    <int>    <dbl>
## 1     0        0    625      625    0.705
## 2     0        1     41        0    0.705
## 3     1        0    241        0    0.705
## 4     1        1     50       50    0.705
```

```
# Overall proportion of wins
mean(fn$won)
```

```
## [1] 0.3040752
```

```
fn %>%
  # mutate(pred_win = 0) %>%
  group_by(won,pred_win) %>%
  summarise(nGames = n()) %>%
  ungroup() %>% # BEST PRACTICES!
  mutate(nCorrect = (won == pred_win)*nGames) %>%
  mutate(accuracy = sum(nCorrect) / sum(nGames))
```

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

```
## # A tibble: 4 × 5
##     won pred_win nGames nCorrect accuracy
##   <dbl>    <dbl>  <int>    <int>    <dbl>
## 1     0        0    625      625    0.705
## 2     0        1     41        0    0.705
## 3     1        0    241        0    0.705
## 4     1        1     50       50    0.705
```

Sensitivity and Specificity

- **Sensitivity**: Accuracy in predicting 1s
- **Specificity**: Accuracy in predicting 0s

```
# Calculate the proportion for each row
fn %>%
  group_by(won,pred_win) %>%
  summarise(nGames = n()) %>%
  ungroup() %>%
  group_by(won) %>%
  mutate(total_games = sum(nGames)) %>%
  ungroup() %>%
  mutate(proportion = nGames / total_games) %>%
  mutate(nCorrect = (won == pred_win)*nGames) %>%
  mutate(accuracy = sum(nCorrect) / sum(nGames))
```

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

```
## # A tibble: 4 × 7
##     won pred_win nGames total_games proportion nCorrect accuracy
##   <dbl>    <dbl>  <int>       <int>      <dbl>    <int>    <dbl>
## 1     0        0    625         666     0.938       625    0.705
## 2     0        1     41         666     0.0616        0    0.705
## 3     1        0    241         291     0.828         0    0.705
## 4     1        1     50         291     0.172        50    0.705
```

Varying the threshold for predicted wins

```
fn %>%
  mutate(pred_win = ifelse(prob_win > .35, # Generate predictions
                           1,
                           0)) %>%
  group_by(won,pred_win) %>%   # Create results table
  summarise(nGames = n()) %>%
  ungroup() %>%
  group_by(won) %>%
  mutate(total_games = sum(nGames)) %>%
  ungroup() %>%
  mutate(proportion = nGames / total_games) %>%
  mutate(nCorrect = (won == pred_win)*nGames) %>%
  mutate(accuracy = sum(nCorrect) / sum(nGames))
```

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

```
## # A tibble: 4 × 7
##     won pred_win nGames total_games proportion nCorrect accuracy
##   <dbl>    <dbl>  <int>       <int>      <dbl>    <int>    <dbl>
## 1     0        0    502         666      0.754      502    0.674
## 2     0        1    164         666      0.246        0    0.674
## 3     1        0    148         291      0.509        0    0.674
## 4     1        1    143         291      0.491      143    0.674
```

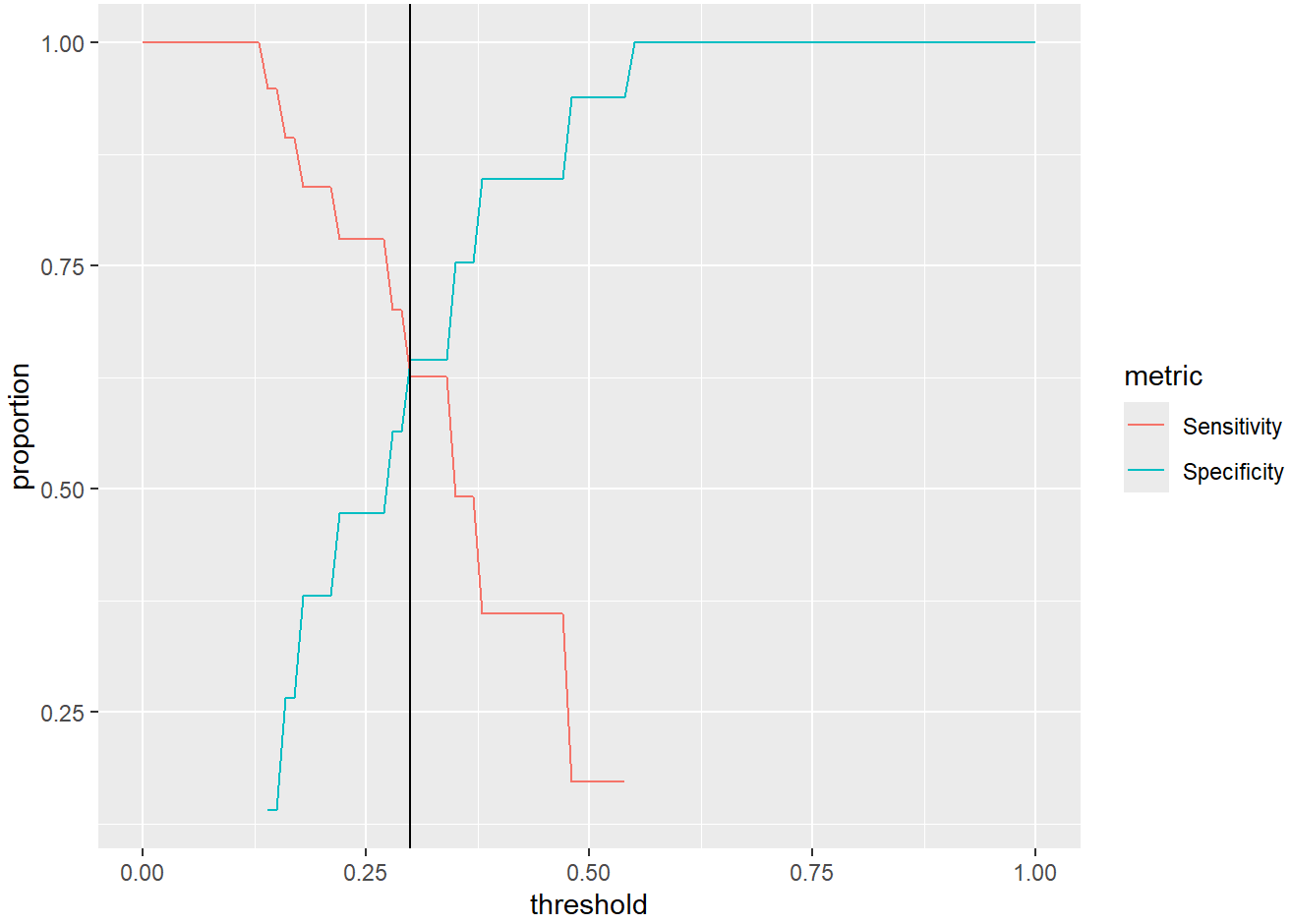
- Use a `for()` loop

```
list_of_thresholds <- seq(0,1,by = .01)
threshRes <- NULL
for(threshold in list_of_thresholds) {
  answer <- fn %>%
  mutate(pred_win = ifelse(prob_win > threshold, # Generate predictions
                           1,
                           0)) %>%
  group_by(won,pred_win) %>%   # Create results table
  summarise(nGames = n()) %>%
  ungroup() %>%
  group_by(won) %>%
  mutate(total_games = sum(nGames)) %>%
  ungroup() %>%
  mutate(proportion = nGames / total_games) %>%
  mutate(nCorrect = (won == pred_win)*nGames) %>%
  mutate(accuracy = sum(nCorrect) / sum(nGames)) %>%
    mutate(threshold = threshold)

  threshRes <- threshRes %>%
    bind_rows(answer)
}

threshRes
```

```
## # A tibble: 284 × 8
##     won pred_win nGames total_games proportion nCorrect accuracy threshold
##   <dbl>    <dbl>  <int>       <int>      <dbl>    <int>    <dbl>     <dbl>
##  1     0        1    666         666          1        0    0.304      0
##  2     1        1    291         291          1      291    0.304      0
##  3     0        1    666         666          1        0    0.304      0.01
##  4     1        1    291         291          1      291    0.304      0.01
##  5     0        1    666         666          1        0    0.304      0.02
##  6     1        1    291         291          1      291    0.304      0.02
##  7     0        1    666         666          1        0    0.304      0.03
##  8     1        1    291         291          1      291    0.304      0.03
##  9     0        1    666         666          1        0    0.304      0.04
## 10     1        1    291         291          1      291    0.304      0.04
## # ℹ 274 more rows
```

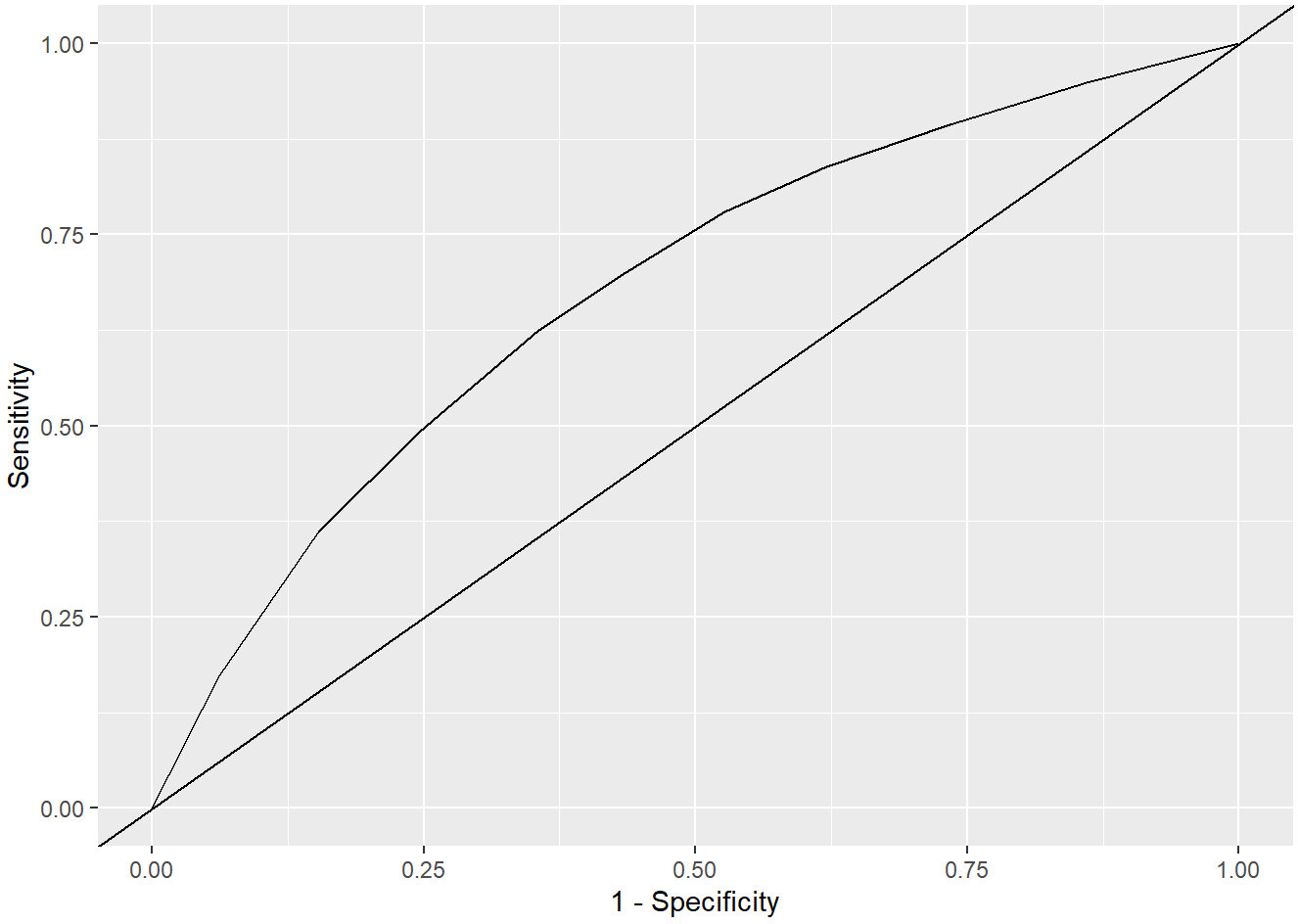
Let’s analyze the threshold results!

```
threshRes %>%
  mutate(metric = ifelse(won == 1 & pred_win == 1,"Sensitivity",
                         ifelse(won == 0 & pred_win == 0,'Specificity',
                                NA))) %>%
  drop_na(metric) %>%
  ggplot(aes(x = threshold,
             y = proportion,
             color = metric)) +
  geom_line() +
  geom_vline(xintercept = .3)
```



Area Under the Curve (AUC)

```
View(threshRes)

threshRes %>%
  mutate(metric = ifelse(won == 1 & pred_win == 1,"Sensitivity",
                         ifelse(won == 0 & pred_win == 0,'Specificity',
                                NA))) %>%
  drop_na(metric) %>%
  select(metric,proportion,threshold) %>%
  pivot_wider(names_from = 'metric',
              values_from = 'proportion',
              values_fill = 0) %>%
  ggplot(aes(x = 1-Specificity,
             y = Sensitivity)) +
  geom_line() +
  geom_abline(intercept = 0,
              slope = 1)
```



Calculating the AUC

- R does this for us!
    - Need the `roc_auc()` function from the `tidymodels` package

```
require(tidymodels)
```

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

```
## ── Attaching packages ─────────────────────────────────────── tidymodels 1.2.0 ──
```

```
## ✓ broom        1.0.6     ✓ rsample      1.2.1
## ✓ dials        1.2.1     ✓ tune         1.2.1
## ✓ infer        1.0.7     ✓ workflows    1.1.4
## ✓ modeldata    1.4.0     ✓ workflowsets 1.1.0
## ✓ parsnip      1.2.1     ✓ yardstick    1.3.1
## ✓ recipes      1.1.0
```

```
## ── 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()
## • Search for functions across packages at https://www.tidymodels.org/find/
```

```
# roc_auc()

toEval <- fn %>%
  select(won,prob_win) %>%
  mutate(won = factor(won,    # Need to convert numeric binary Y to factor
                      levels = c('1','0'))) # NEED TO MAKE THE 1 VALUE COME FIRST

roc_auc(toEval,won,prob_win)
```

```
## # A tibble: 1 × 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 roc_auc binary         0.678
```