Stat 301-2 Final Project

Jeremy OFlynn

March 5, 2019

Loading Packages

```
library(tidyverse)
```

```
## ── Attaching packages ─────────────────────────────────── tidyverse 1.2.1 ──
```

```
## ✔ ggplot2 3.1.0     ✔ purrr   0.3.1
## ✔ tibble  2.0.1     ✔ dplyr   0.8.0.1
## ✔ tidyr   0.8.3     ✔ stringr 1.4.0
## ✔ readr   1.3.1     ✔ forcats 0.4.0
```

```
## ── Conflicts ────────────────────────────────────── tidyverse_conflicts() ──
## ✖ dplyr::filter() masks stats::filter()
## ✖ dplyr::lag()    masks stats::lag()
```

```
library(modelr)
library(janitor)
library(skimr)
```

```
##
## Attaching package: 'skimr'
```

```
## The following object is masked from 'package:stats':
##
##     filter
```

```
library(broom)
```

```
##
## Attaching package: 'broom'
```

```
## The following object is masked from 'package:modelr':
##
##     bootstrap
```

```r
library(corrplot)
```

```
## corrplot 0.84 loaded
```

```r
library(ggfortify)
library(glmnet)
```

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

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

```
## The following object is masked from 'package:tidyr':
##
##     expand
```

```
## Loading required package: foreach
```

```
##
## Attaching package: 'foreach'
```

```
## The following objects are masked from 'package:purrr':
##
##     accumulate, when
```

```
## Loaded glmnet 2.0-16
```

```r
library(glmnetUtils)
```

```
##
## Attaching package: 'glmnetUtils'
```

```
## The following objects are masked from 'package:glmnet':
##
##     cv.glmnet, glmnet
```

```r
library(pls)
```

```
##
## Attaching package: 'pls'
```

```
## The following object is masked from 'package:corrplot':
##
##     corrplot
```

```
## The following object is masked from 'package:stats':
##
##     loadings
```

```
library(class)
library(ROCR)
```

```
## Loading required package: gplots
```

```
##
## Attaching package: 'gplots'
```

```
## The following object is masked from 'package:stats':
##
##     lowess
```

Data Processing

```
# reading in data
data <- read.csv("data/horses.csv") %>%
  filter(position_two > 0)

# getting rid of predictors that are impractical or inconsistent
data <- data %>%
  select(-previous_margin, -position, -position_again, -margin, -bf_odds_all,
         -vic_tote_all, -nsw_tote_all, -betfair_slope, -vic_tote_slope, -nsw_tote_slo
pe,
         -nsw_odds_slope, -country_code, -venue_name, -date, -market_name, -condition
,
         -name, -runner_name_uuid, -last_five_starts, -penalty, -sire,
         -dam, -colour, -jockey, -jockey_sex, -trainer, -form_comment, -form_comment_
sentiment,
         -last_twenty_starts, -class_level, -field_strength, -emergency, -blinkers,
         -favourite_odds_win, -favourite_tote_win, -tip_12_months_win,
         -tip_distance_win, -tip_class_win, -tip_time_win, -tip_overall_win,
         -sex, -runs_since_spell, -weather, -runner_id, -dfs_form_rating,
         -tip_rating_win)
```

The first step in my data processing was to filter out horses who did not finish their races. This occurs when horses get injuries or disqualifications, which are unpredictable occurences that can skew the results of our analysis.

Next, I got rid of approximately 30 variables which I was not going to use for any analysis. Most of these variables were character strings that were different for each and every horse, were completely random, or variables that contained information that was unattainable.

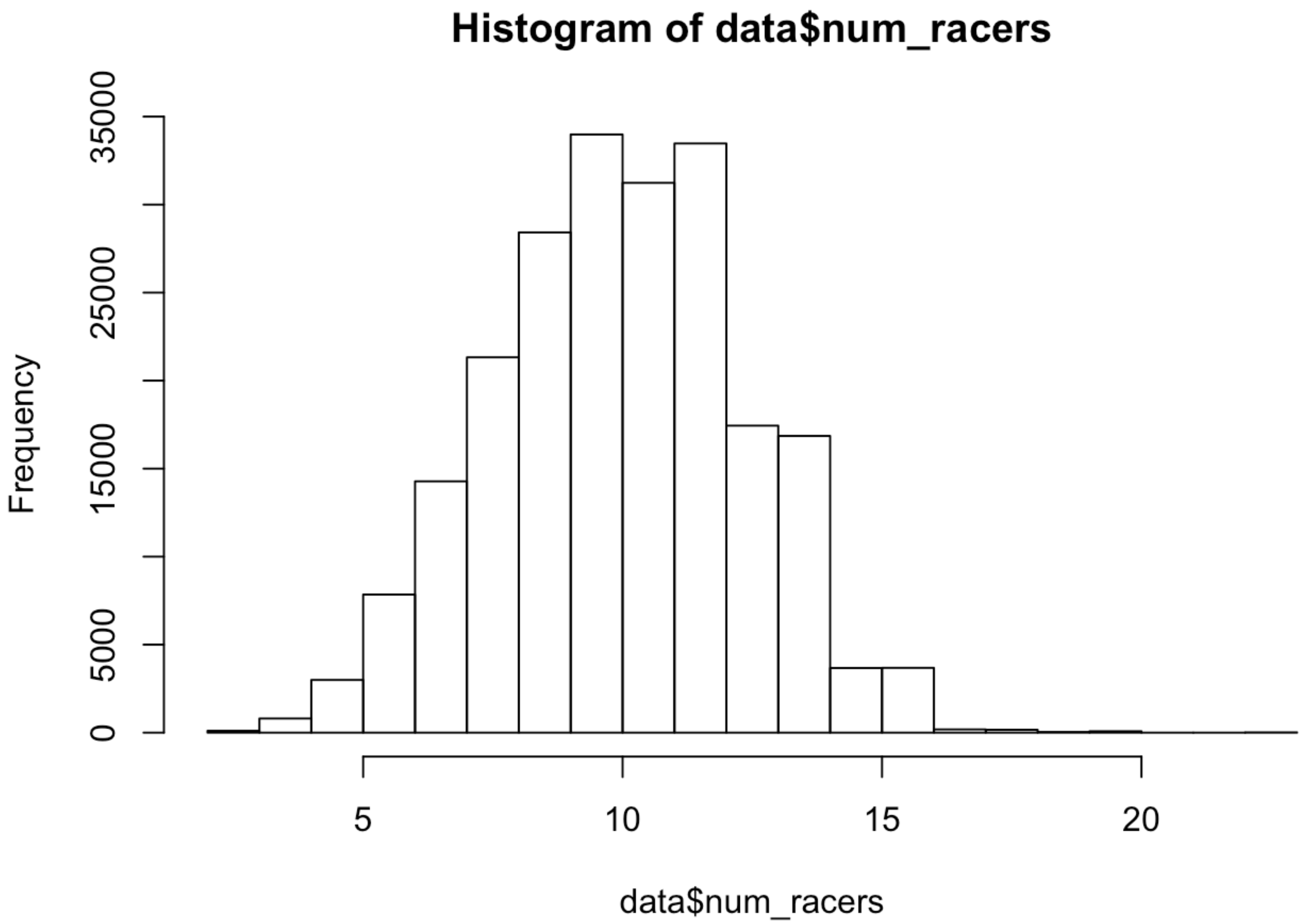
```
data <- data %>%
  mutate(tip_pundit_win = ifelse(tip_pundit_win == TRUE, 1, 0),
         tip_recent_win = ifelse(tip_recent_win == TRUE, 1, 0))
```

I converted the remaining binary variables to be of the form 0/1 instead of TRUE/FALSE.

```
# adding cumulative variables
data <- data %>%
  group_by(market_id) %>%
  # variable to track total number of horses in each race
  mutate(num_racers = max(position_two),
         # variable to track total cumulative winnings of all horses in
         # in a race
         tot_prize_money = sum(prize_money))
```

I added variables to track cumulative metrics for individual races. Creating a variable that tracked the number of racers allowed me to calculate `finish_percentile`. Calculating the total prize money in a race allowed me to create predictors based on a horse's prize money relative to the other horses in their races.

```
hist(data$num_racers)
```



```r
# adding variables for analysis
data <- data %>%
  # restricting the dataset to races with 15 horses or less
  filter(num_racers < 16 & num_racers > 4) %>%
  # percentile of final position
  mutate(finish_percentile = 1 - position_two / num_racers,
         # individual horse share of total cumulative winnings of horses in race
         prize_money_share = prize_money / tot_prize_money,
         # track if the horse covered a winning bet
         win = ifelse(position_two == 1, 1, 0),
         # track if the horse covered a place (top 2 finish) bet
         place = if_else(position_two <= 2, 1, 0),
         # track if the horse covered a show (top 3 finish) bet
         show = if_else(position_two <= 3, 1, 0),
         # average final odds of 3 sources
         mean_final_odds = (bf_odds_two_mins_out + vic_tote_two_mins_out +
                              nsw_odds)/3,
         # winnings per run
         prize_money_per_run = ifelse(overall_starts > 0,
                                      prize_money / overall_starts, 0),
         # win rate
         win_rate = overall_wins / overall_starts,
         # place rate
         place_rate = overall_places / overall_starts)
```

I decided to filter out races with more than 15 horses, because these are outliers that can skew the scale of our response variables.

I created many other variables to indicate performance in the race, performance in historical races, and predictors to compare horses to each other before the race.

```r
# create variable to track prize money per run share
data <- data %>%
  group_by(market_id) %>%
  mutate(tot_prize_money_per_run = sum(prize_money_per_run),
         prize_money_per_run_share = prize_money_per_run / tot_prize_money_per_run)
```

I created another variable to track historical prize money by horses, standardized by how many races horses participated in.

```r
# creating variable to track relative odds rank, and relative prize money
# per run rank
data <- data %>%
  group_by(market_id) %>%
  mutate(odds_rank = min_rank(mean_final_odds),
         prize_money_per_run_rank = abs(num_racers - min_rank(prize_money_per_run)))
```

The last variables I created were the ranks of some statistics of the horses.

```
# writing the data to an RDS to make it easier to read in
write_rds(data, "data/horse_processed.rds")
```

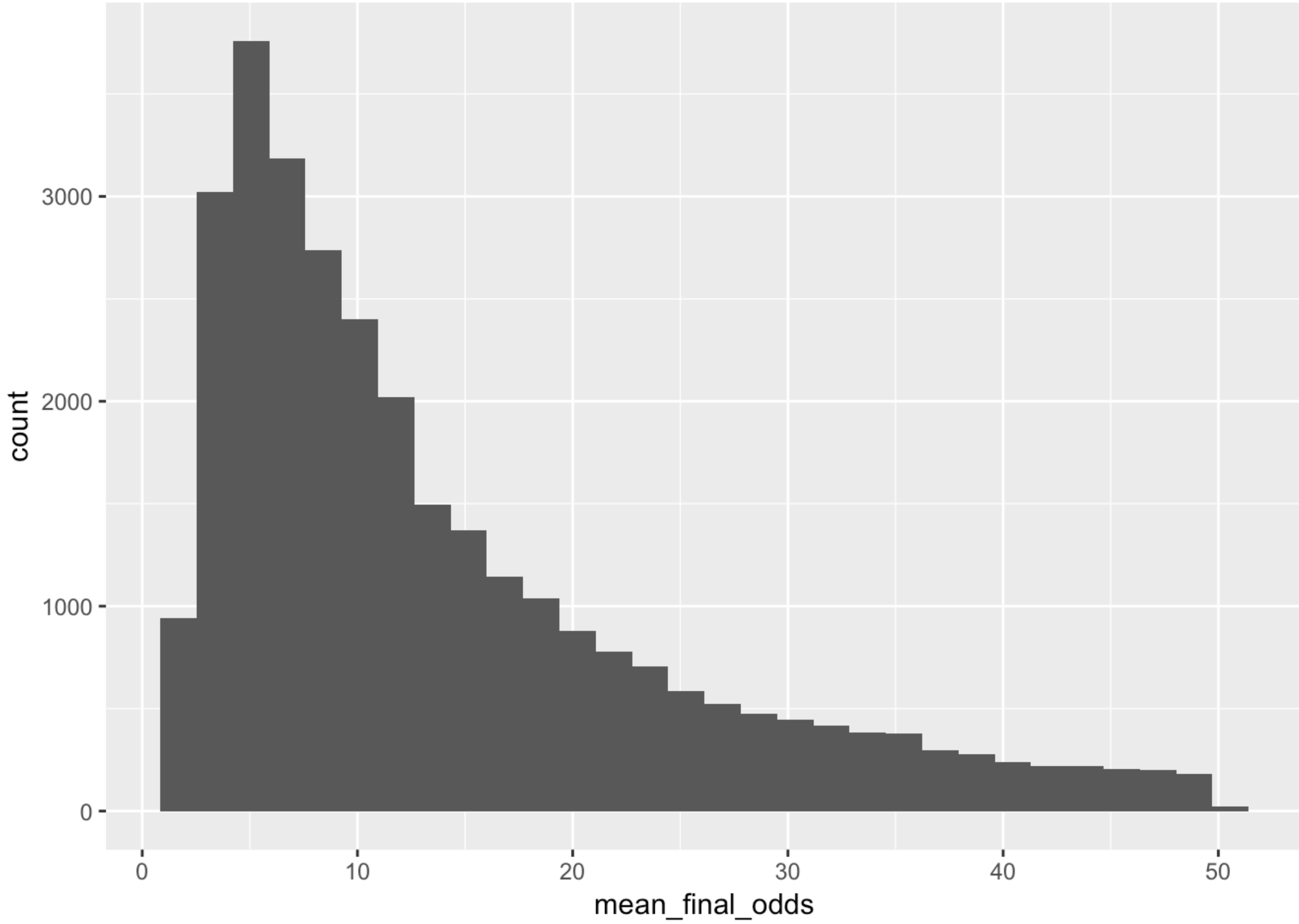
EDA

```
# setting seed for splitting datasets
set.seed(1)

# splitting data into eda and non eda
eda_data <- data %>%
  drop_na() %>%
  sample_frac(0.2)

non_eda_data <- data %>%
  drop_na() %>%
  setdiff(eda_data)
```

I split the data into an EDA set and non-EDA set, using fractions of 20% and 80%.

```
# histogram of odds frequencies
eda_data %>%
  filter(mean_final_odds < 50) %>%
  ggplot(aes(x = mean_final_odds)) +
  geom_histogram()
```

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



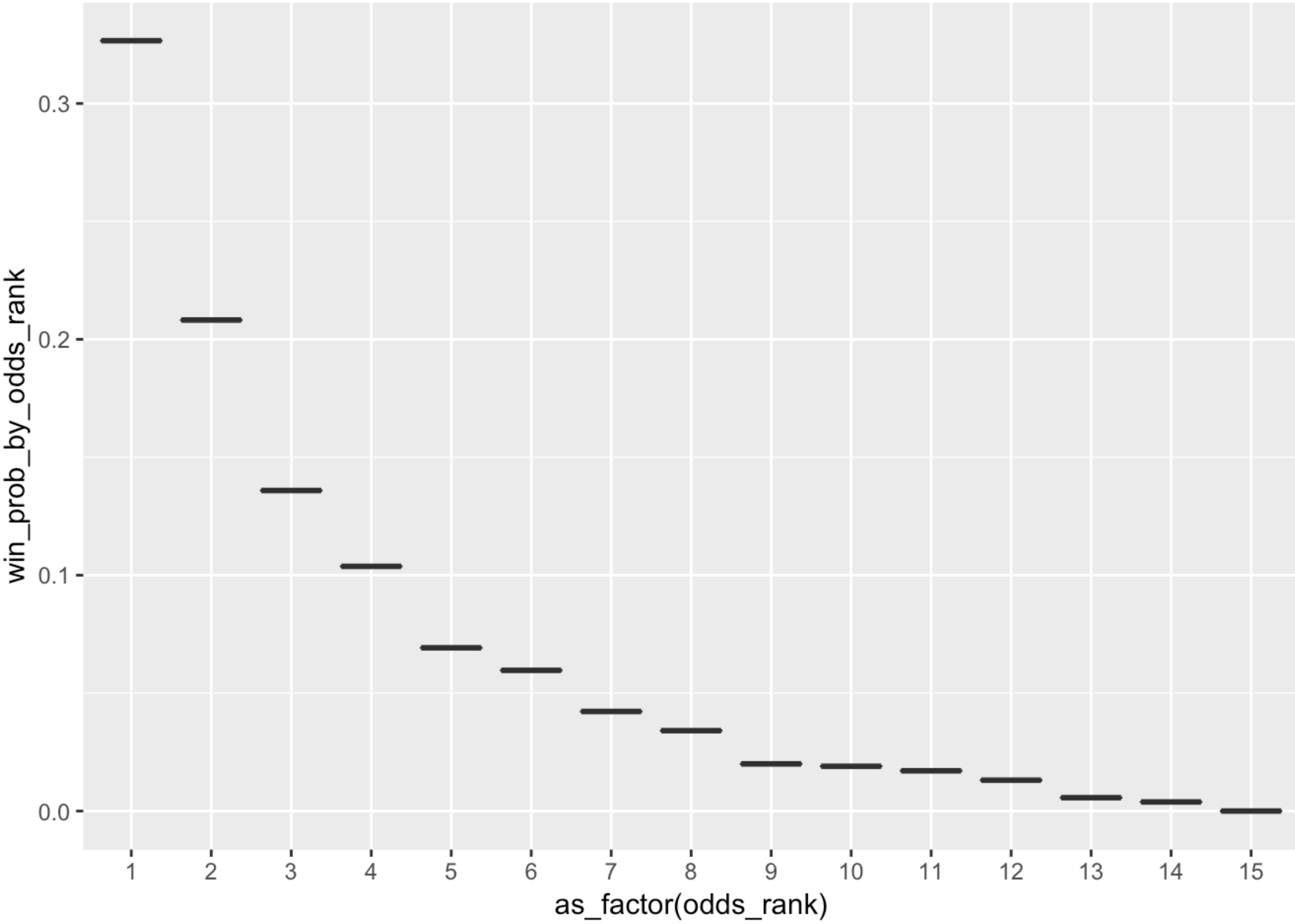
We see that the most frequent odds are between 0 and 10, which is not surprising.

```
eda_data %>%
  drop_na() %>%
  group_by(odds_rank) %>%
  mutate(win_prob_by_odds_rank = mean(win)) %>%
  select(odds_rank, win_prob_by_odds_rank) %>%
  unique() %>%
  arrange(desc(win_prob_by_odds_rank))
```

```
## # A tibble: 15 x 2
## # Groups:   odds_rank [15]
##    odds_rank win_prob_by_odds_rank
##        <int>                 <dbl>
##  1         1               0.327
##  2         2               0.208
##  3         3               0.136
##  4         4               0.104
##  5         5               0.0692
##  6         6               0.0597
##  7         7               0.0422
##  8         8               0.0341
##  9         9               0.0200
## 10        10               0.0190
## 11        11               0.0170
## 12        12               0.0131
## 13        13               0.00566
## 14        14               0.00386
## 15        15               0
```

These are the probabilites of winning by odds rank. The probabilites level off in a diminshing manner, suggesting that there are few inconsistencies in the odds being set.

```
eda_data %>%
  drop_na() %>%
  group_by(odds_rank) %>%
  mutate(win_prob_by_odds_rank = mean(win)) %>%
  select(odds_rank, win_prob_by_odds_rank) %>%
  unique() %>%
  arrange(desc(win_prob_by_odds_rank)) %>%
  ggplot(aes(x = as_factor(odds_rank), y = win_prob_by_odds_rank)) +
    geom_boxplot()
```

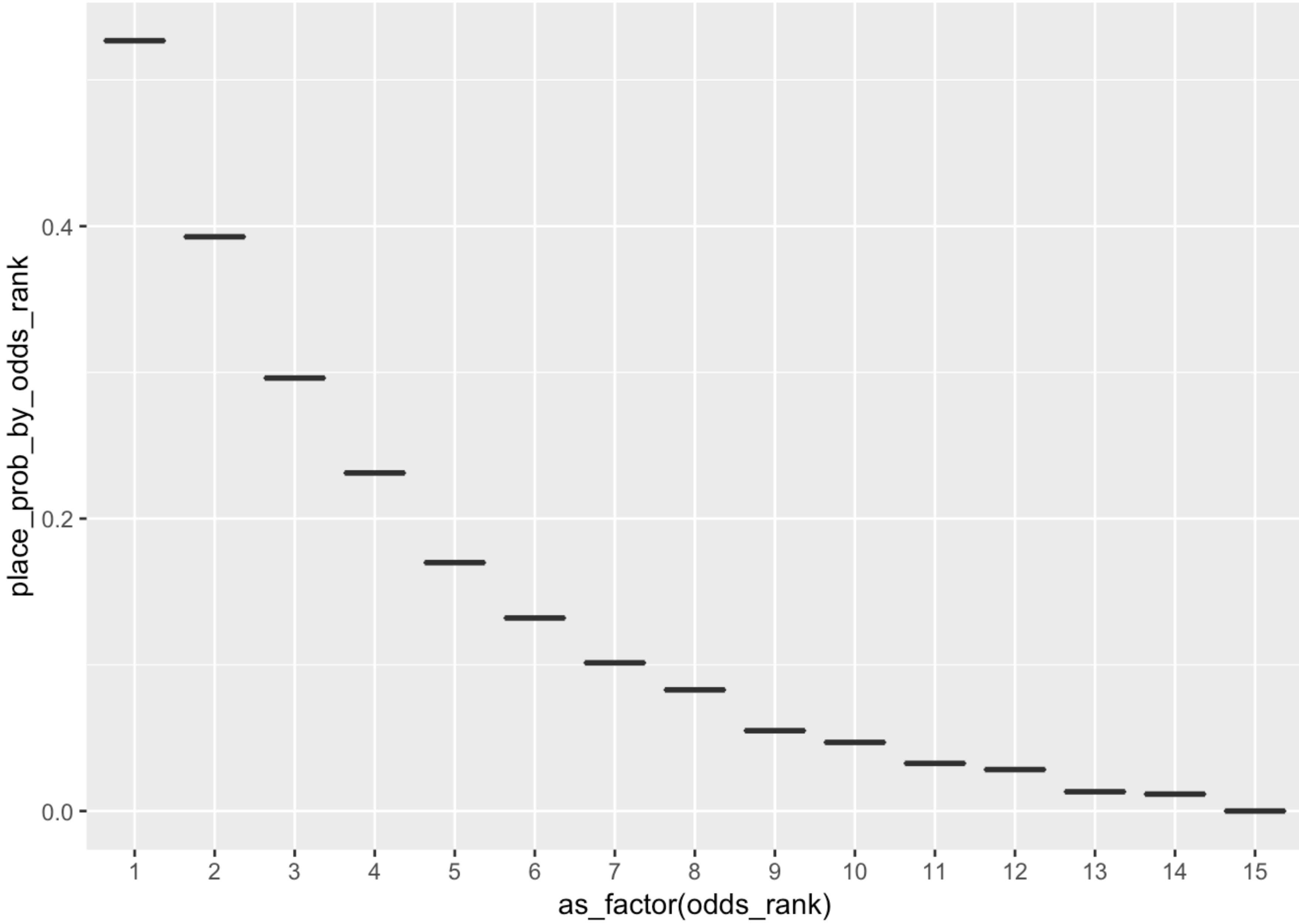


This plot shows how much the probabilities level off. It suggests that poorly ranked horses have similar probabilites of winning, so we can potentially consider betting on low-ranked horses if we are not going to bet one of the top 4 favorites.

```
eda_data %>%
  drop_na() %>%
  group_by(odds_rank) %>%
  mutate(place_prob_by_odds_rank = mean(place)) %>%
  select(odds_rank, place_prob_by_odds_rank) %>%
  unique() %>%
  arrange(desc(place_prob_by_odds_rank))
```

```
## # A tibble: 15 x 2
## # Groups:   odds_rank [15]
##    odds_rank place_prob_by_odds_rank
##        <int>                   <dbl>
##  1         1                  0.527
##  2         2                  0.393
##  3         3                  0.296
##  4         4                  0.231
##  5         5                  0.170
##  6         6                  0.132
##  7         7                  0.101
##  8         8                  0.0829
##  9         9                  0.0550
## 10        10                  0.0470
## 11        11                  0.0326
## 12        12                  0.0284
## 13        13                  0.0132
## 14        14                  0.0116
## 15        15                  0
```

```
# probabilites of placing by odds rank

eda_data %>%
  drop_na() %>%
  group_by(odds_rank) %>%
  mutate(place_prob_by_odds_rank = mean(place)) %>%
  select(odds_rank, place_prob_by_odds_rank) %>%
  unique() %>%
  arrange(desc(place_prob_by_odds_rank)) %>%
  ggplot(aes(x = as_factor(odds_rank), y = place_prob_by_odds_rank)) +
  geom_boxplot()
```

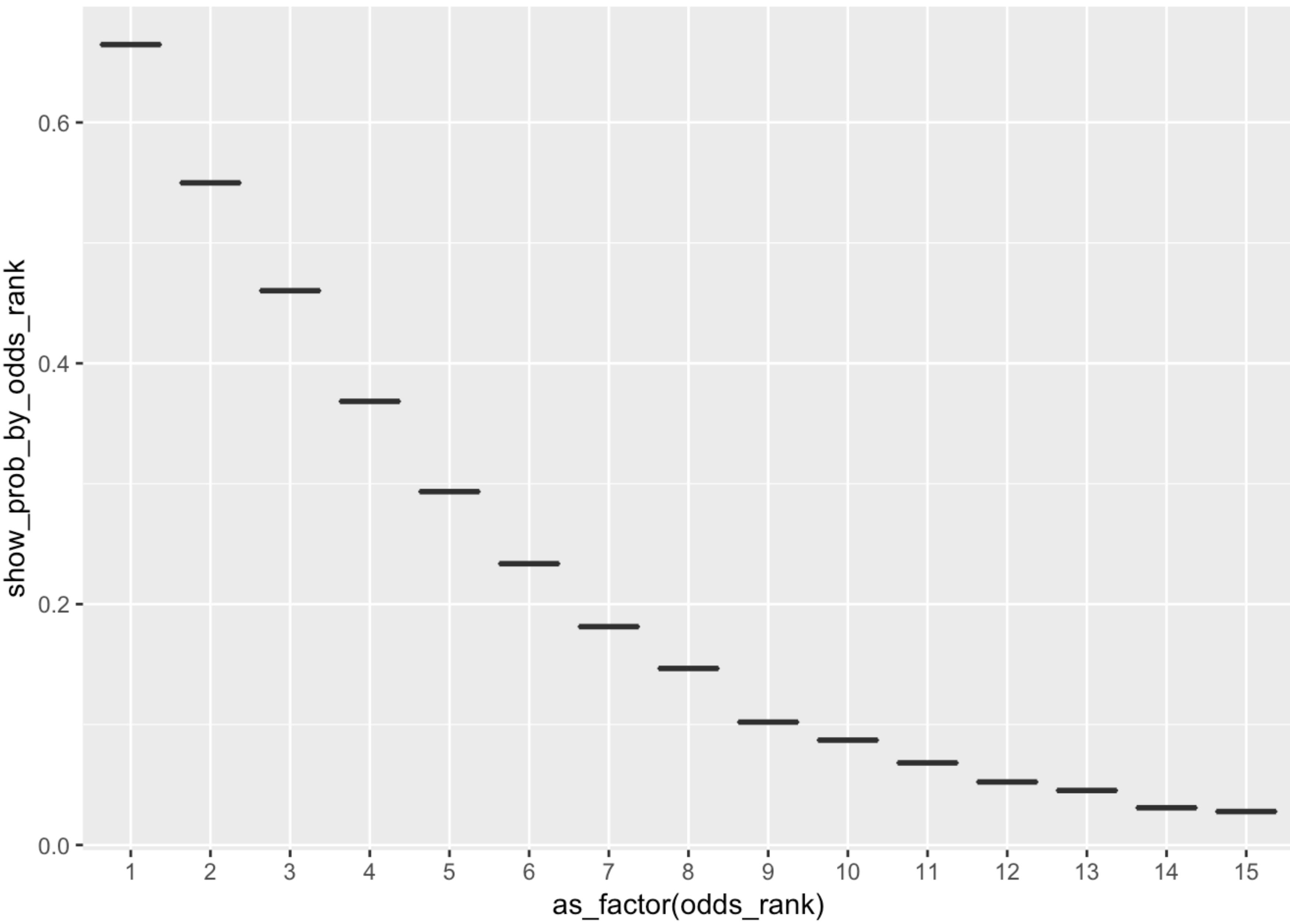


We see a more pronounced curve for the probability of placing, in which the probabilites fall off at a rather steep rate until the horse with roughly the 9th best odds or so.

```
eda_data %>%
  drop_na() %>%
  filter(num_racers < 16) %>%
  group_by(odds_rank) %>%
  mutate(show_prob_by_odds_rank = mean(show)) %>%
  select(odds_rank, show_prob_by_odds_rank) %>%
  unique() %>%
  arrange(desc(show_prob_by_odds_rank))
```

```
## # A tibble: 15 x 2
## # Groups:   odds_rank [15]
##    odds_rank show_prob_by_odds_rank
##        <int>                  <dbl>
##  1         1                 0.665
##  2         2                 0.550
##  3         3                 0.460
##  4         4                 0.368
##  5         5                 0.293
##  6         6                 0.234
##  7         7                 0.181
##  8         8                 0.147
##  9         9                 0.102
## 10        10                 0.0871
## 11        11                 0.0682
## 12        12                 0.0523
## 13        13                 0.0453
## 14        14                 0.0309
## 15        15                 0.0278
```

```
eda_data %>%
  drop_na() %>%
  filter(num_racers < 16) %>%
  group_by(odds_rank) %>%
  mutate(show_prob_by_odds_rank = mean(show)) %>%
  select(odds_rank, show_prob_by_odds_rank) %>%
  unique() %>%
  arrange(desc(show_prob_by_odds_rank)) %>%
  ggplot(aes(x = as_factor(odds_rank), y = show_prob_by_odds_rank)) +
  geom_boxplot()
```



We see a similar distribution for showing as we do for placing. The curve again levels off around the 9th best horse. This suggests that if one is betting on the 9th best horse, they may want to consider betting on a place instead of a show, since the patterns are very similar.

```
eda_data %>%
  drop_na() %>%
  select(market_id, win, place, show, mean_final_odds, prize_money_share, prize_money
,
         finish_percentile, bf_odds_two_mins_out, vic_tote_two_mins_out,
         nsw_odds, number, tech_form_rating, age, position_two) %>%
  cor()
```

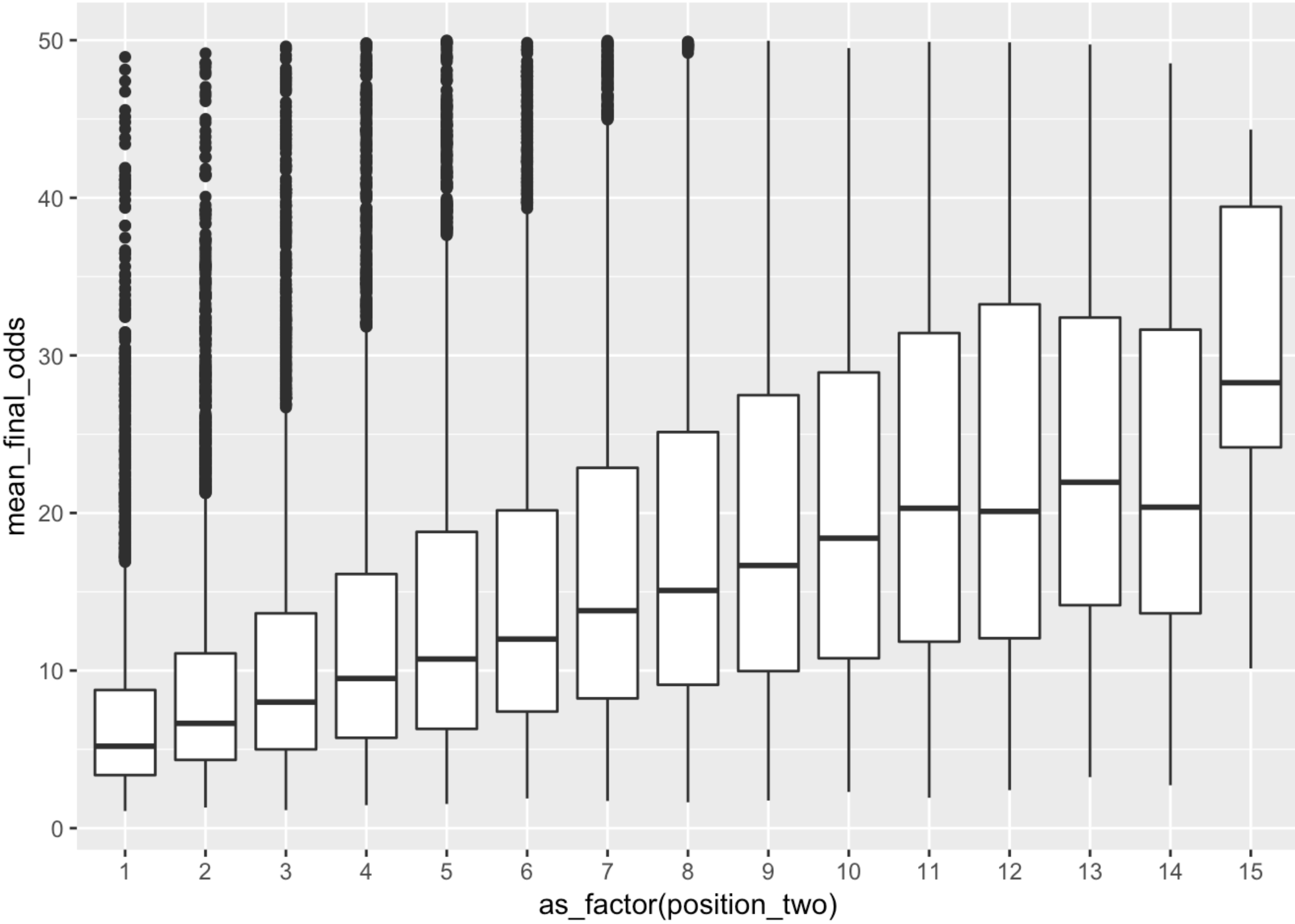
```
##                          market_id          win        place         show
## market_id              1.000000000 -0.007036745 -0.001070225  0.004216747
## win                   -0.007036745  1.000000000  0.669225553  0.507267541
## place                 -0.001070225  0.669225553  1.000000000  0.757991889
## show                   0.004216747  0.507267541  0.757991889  1.000000000
## mean_final_odds        0.062229068 -0.176481316 -0.244241461 -0.290723544
## prize_money_share      0.017050272  0.157636634  0.193693928  0.213116345
## prize_money           -0.020053418  0.037378769  0.035985477  0.035917655
## finish_percentile     -0.003931794  0.520394008  0.682153018  0.771990260
## bf_odds_two_mins_out   0.064455945 -0.159336382 -0.221188927 -0.264253387
## vic_tote_two_mins_out  0.046543351 -0.188735633 -0.257349905 -0.302401851
```

```
## nsw_odds                0.059277892 -0.172398097 -0.239623649 -0.285873634
## number                 -0.013323131 -0.105133332 -0.139662701 -0.162697534
## tech_form_rating        0.003719914  0.203396381  0.264960680  0.298732083
## age                    -0.148466552 -0.054632584 -0.070147990 -0.075838620
## position_two           -0.006707294 -0.486908856 -0.650177246 -0.752768131
##                        mean_final_odds prize_money_share prize_money
## market_id                   0.06222907        0.01705027 -0.02005342
## win                        -0.17648132        0.15763663  0.03737877
## place                      -0.24424146        0.19369393  0.03598548
## show                       -0.29072354        0.21311635  0.03591765
## mean_final_odds             1.00000000       -0.23218615 -0.06282866
## prize_money_share          -0.23218615        1.00000000  0.20776949
## prize_money                -0.06282866        0.20776949  1.00000000
## finish_percentile          -0.37559964        0.17995001  0.03957565
## bf_odds_two_mins_out        0.97367435       -0.20975860 -0.05882246
## vic_tote_two_mins_out       0.89484871       -0.24686534 -0.05015869
## nsw_odds                    0.96564549       -0.22738027 -0.06722814
## number                      0.29278684       -0.27334086 -0.11641654
## tech_form_rating           -0.53094422        0.26045467  0.03362305
## age                         0.10124140        0.14241008  0.26597524
## position_two                0.43163646       -0.26908198 -0.03955283
##                        finish_percentile bf_odds_two_mins_out
## market_id                   -0.003931794           0.06445594
## win                          0.520394008          -0.15933638
## place                        0.682153018          -0.22118893
## show                         0.771990260          -0.26425339
## mean_final_odds             -0.375599641           0.97367435
## prize_money_share            0.179950011          -0.20975860
## prize_money                  0.039575653          -0.05882246
## finish_percentile            1.000000000          -0.35017132
## bf_odds_two_mins_out        -0.350171320           1.00000000
## vic_tote_two_mins_out       -0.363420444           0.80822594
## nsw_odds                    -0.370890100           0.89809089
## number                      -0.128492996           0.26932782
## tech_form_rating             0.346027202          -0.47953760
## age                         -0.095048753           0.09018474
## position_two                -0.900359231           0.39100058
##                        vic_tote_two_mins_out    nsw_odds      number
## market_id                         0.04654335  0.05927789 -0.01332313
## win                              -0.18873563 -0.17239810 -0.10513333
## place                            -0.25734991 -0.23962365 -0.13966270
## show                             -0.30240185 -0.28587363 -0.16269753
## mean_final_odds                   0.89484871  0.96564549  0.29278684
## prize_money_share                -0.24686534 -0.22738027 -0.27334086
## prize_money                      -0.05015869 -0.06722814 -0.11641654
## finish_percentile                -0.36342044 -0.37089010 -0.12849300
## bf_odds_two_mins_out              0.80822594  0.89809089  0.26932782
## vic_tote_two_mins_out             1.00000000  0.85126587  0.30152006
## nsw_odds                          0.85126587  1.00000000  0.28486264
## number                            0.30152006  0.28486264  1.00000000
```

```
## tech_form_rating                -0.57190291 -0.51627520 -0.35545966
## age                              0.11378856  0.09778520 -0.02463196
## position_two                     0.44832099  0.42670549  0.24025128
##                       tech_form_rating         age position_two
## market_id                  0.003719914 -0.14846655 -0.006707294
## win                        0.203396381 -0.05463258 -0.486908856
## place                      0.264960680 -0.07014799 -0.650177246
## show                       0.298732083 -0.07583862 -0.752768131
## mean_final_odds           -0.530944224  0.10124140  0.431636461
## prize_money_share          0.260454670  0.14241008 -0.269081984
## prize_money                0.033623050  0.26597524 -0.039552828
## finish_percentile          0.346027202 -0.09504875 -0.900359231
## bf_odds_two_mins_out      -0.479537602  0.09018474  0.391000578
## vic_tote_two_mins_out     -0.571902913  0.11378856  0.448320986
## nsw_odds                  -0.516275201  0.09778520  0.426705486
## number                    -0.355459663 -0.02463196  0.240251278
## tech_form_rating           1.000000000 -0.22364257 -0.361617522
## age                       -0.223642568  1.00000000  0.081172525
## position_two              -0.361617522  0.08117252  1.000000000
```

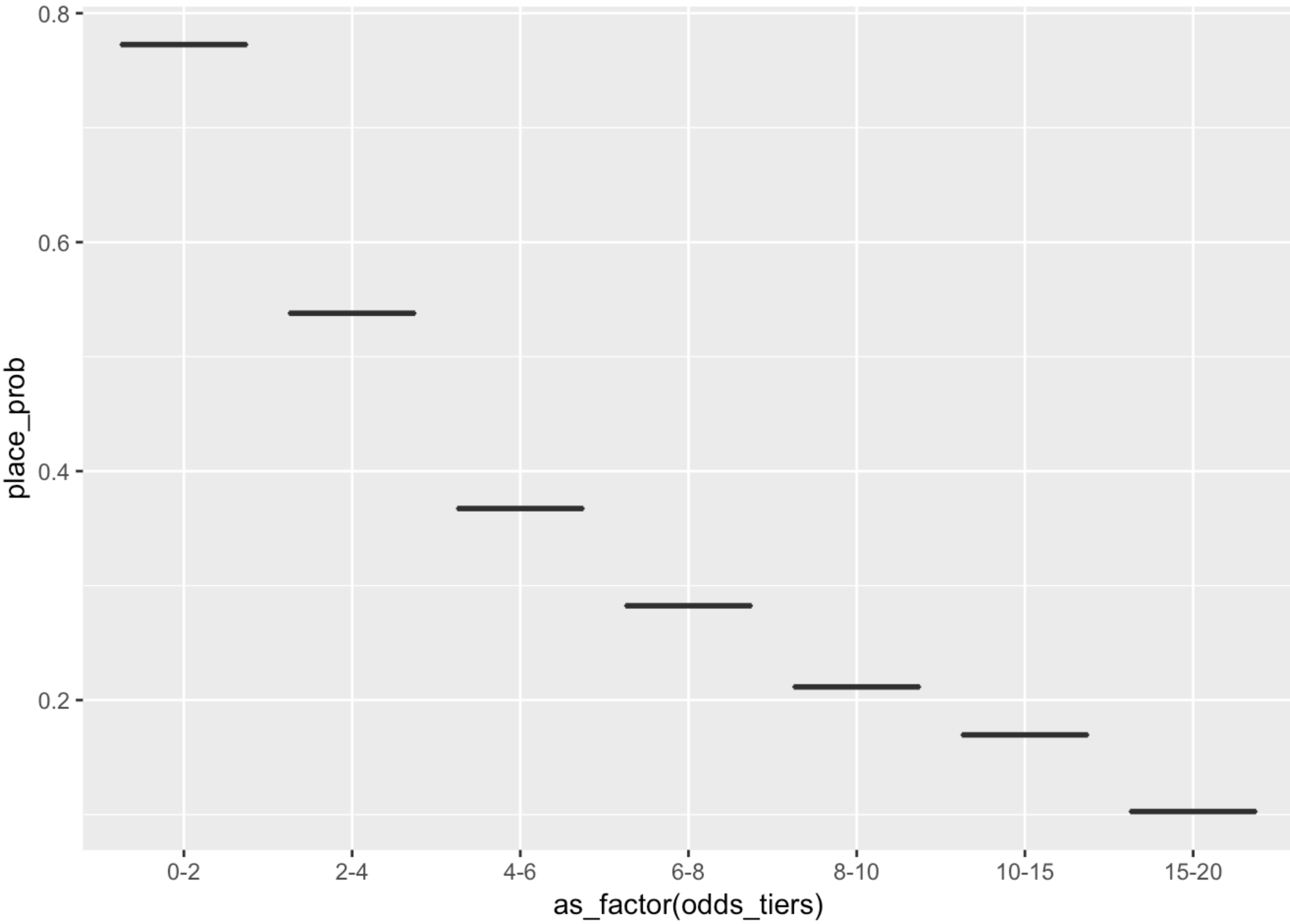
We see that `win` `place` `show` and `finish_percentile` are strongly correlated as expected.
`prize_money_share` has a much stronger correlation with results than `prize_money`, which shows that we may be on the right track with some of the predictors we created.
`nsw_odds` has a stronger correlation with finish percentile than the other sources. Perhaps nsw is a more accurate provider of odds.
strongest correlations are `mean_final_odds` and `prize_money_share`, which makes sense.

```
eda_data %>%
  drop_na() %>%
  filter(mean_final_odds < 50, position_two > 0) %>%
  ggplot(aes(x = as_factor(position_two), y = mean_final_odds)) +
  geom_boxplot()
```



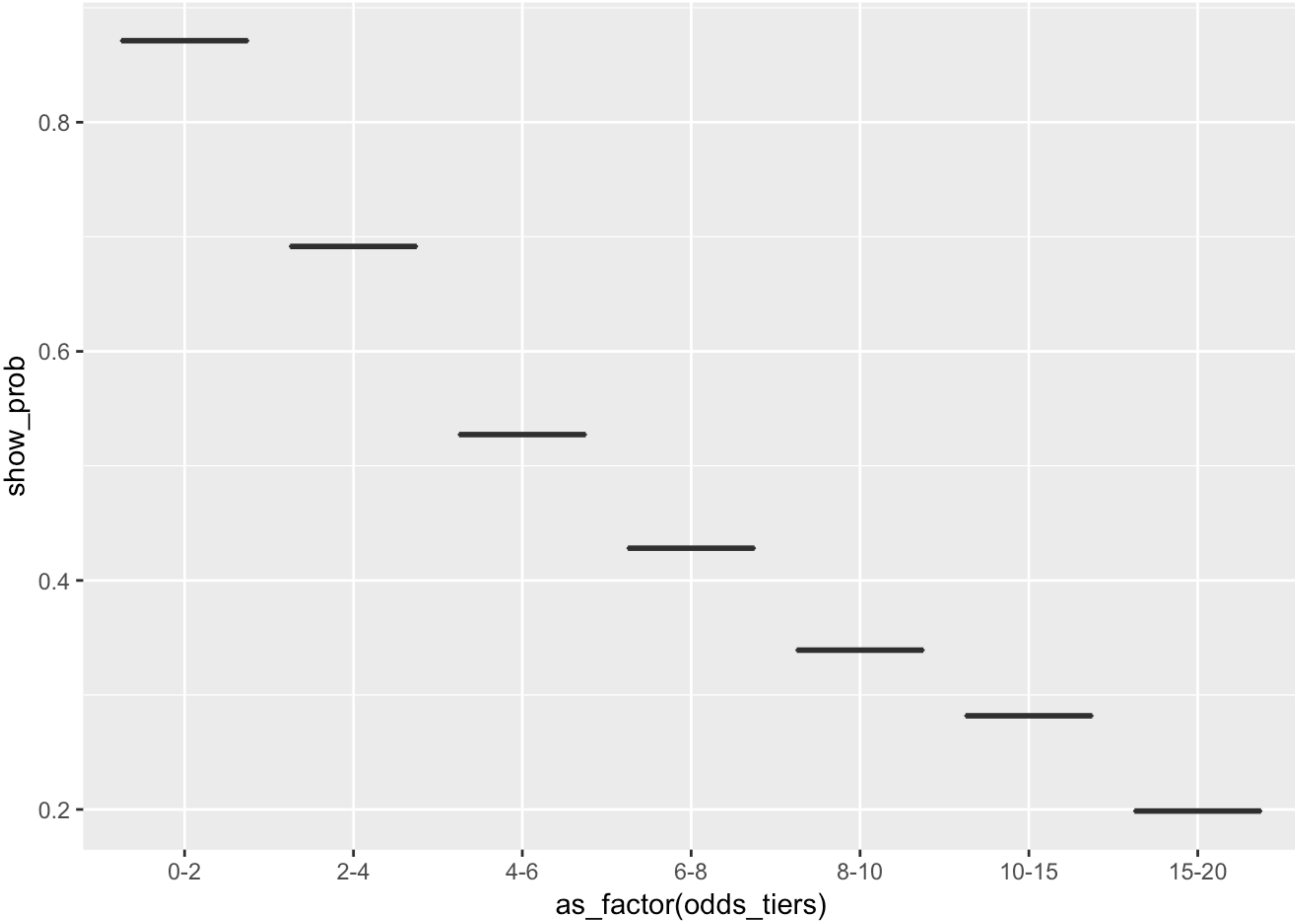
This plot shows that horses with better odds definitely tend to finish in better positions. However, the average odds of horses that finish in 1st is somewhere between 5:1 and 6:1. These may be skewed by rare occurences in which low ranked horses manage to win.

```
eda_data %>%
  drop_na() %>%
  filter(mean_final_odds <= 20) %>%
  mutate(odds_tiers = case_when(mean_final_odds <= 2 ~ "0-2",
                                mean_final_odds <= 4 ~ "2-4",
                                mean_final_odds <= 6 ~ "4-6",
                                mean_final_odds <= 8 ~ "6-8",
                                mean_final_odds <= 10 ~ "8-10",
                                mean_final_odds <= 15 ~ "10-15",
                                mean_final_odds <= 20 ~ "15-20")) %>%
  group_by(odds_tiers) %>%
  mutate(place_prob = mean(place)) %>%
  arrange(desc(place_prob)) %>%
  ggplot(aes(x = as_factor(odds_tiers), y = place_prob)) +
  geom_boxplot()
```



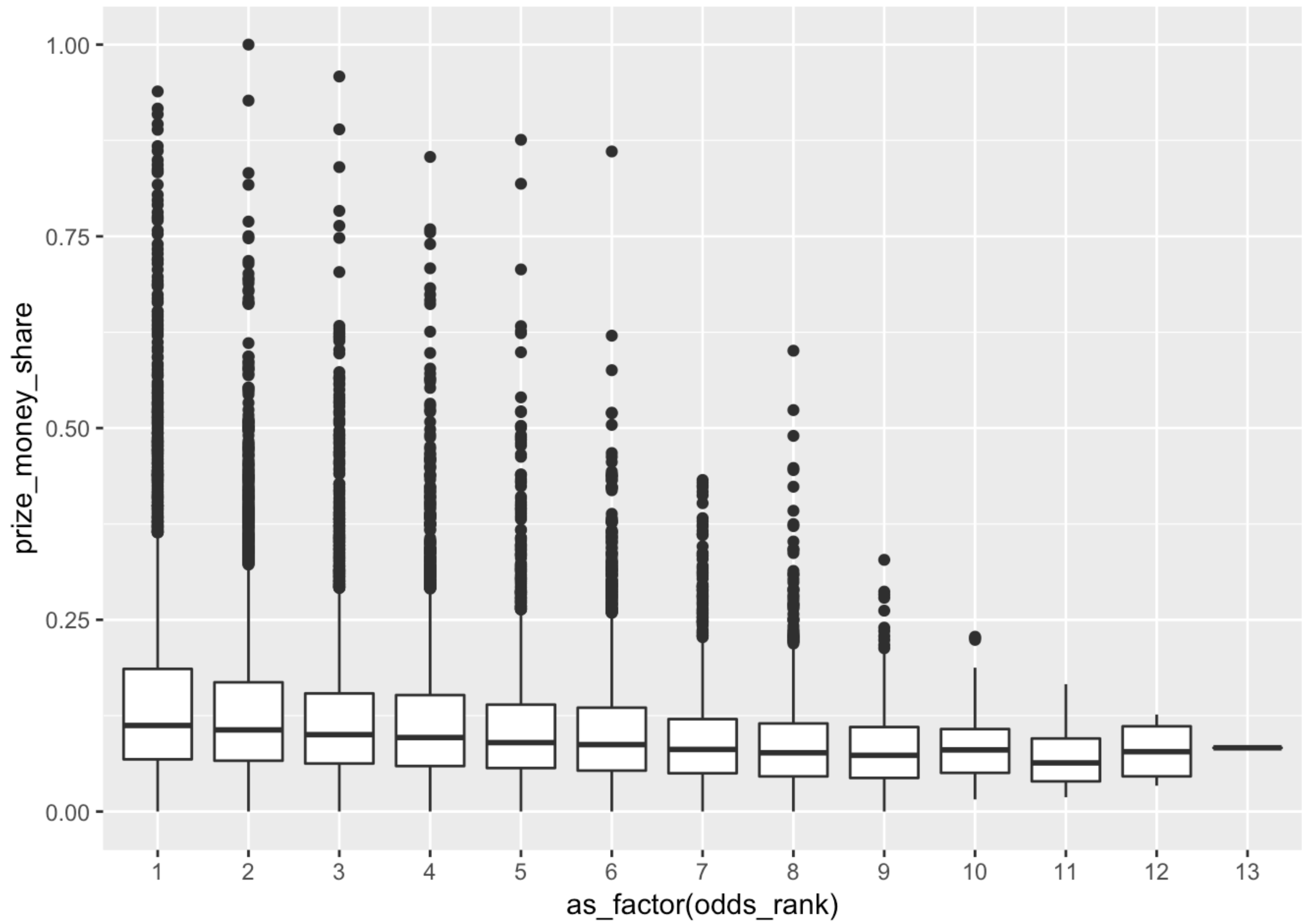
Treating horse odds as tiers as opposed to obsolute ranks gives us a similar result to what was observed earlier. The favorites appear to be very likely to finish in the top 2.

```
eda_data %>%
  drop_na() %>%
  filter(mean_final_odds <= 20) %>%
  mutate(odds_tiers = case_when(mean_final_odds <= 2 ~ "0-2",
                                mean_final_odds <= 4 ~ "2-4",
                                mean_final_odds <= 6 ~ "4-6",
                                mean_final_odds <= 8 ~ "6-8",
                                mean_final_odds <= 10 ~ "8-10",
                                mean_final_odds <= 15 ~ "10-15",
                                mean_final_odds <= 20 ~ "15-20")) %>%
  group_by(odds_tiers) %>%
  mutate(show_prob = mean(show)) %>%
  arrange(desc(show_prob)) %>%
         ggplot(aes(x = as_factor(odds_tiers), y = show_prob)) +
         geom_boxplot()
```



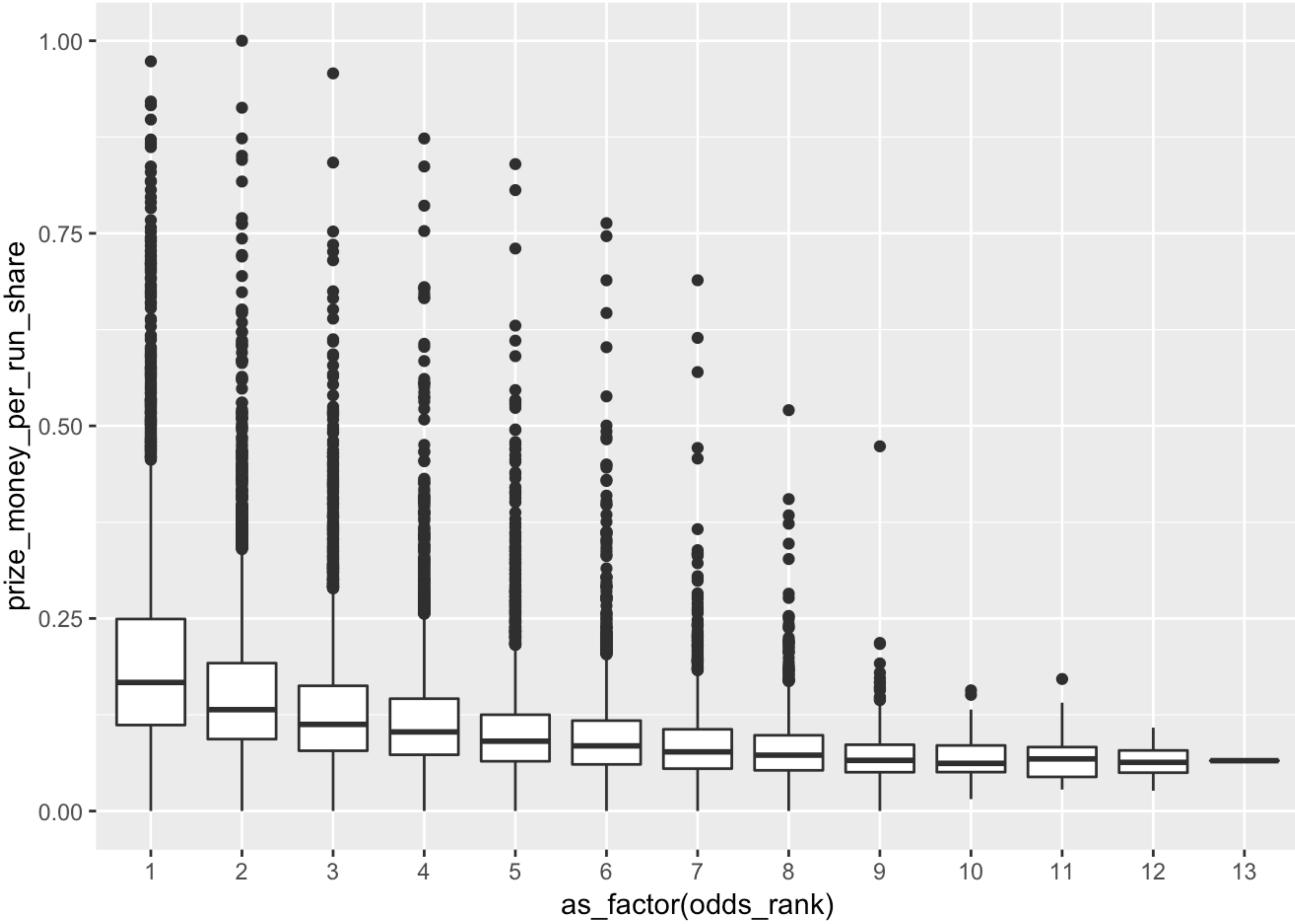
The probability of showing follows very similarly.

```
eda_data %>%
  drop_na() %>%
  # experimenting with removing low ranked horses
  filter(mean_final_odds <= 20) %>%
  mutate(show_prob = mean(show)) %>%
  arrange(desc(show_prob)) %>%
  ggplot(aes(x = as_factor(odds_rank), y = prize_money_share)) +
  geom_boxplot()
```



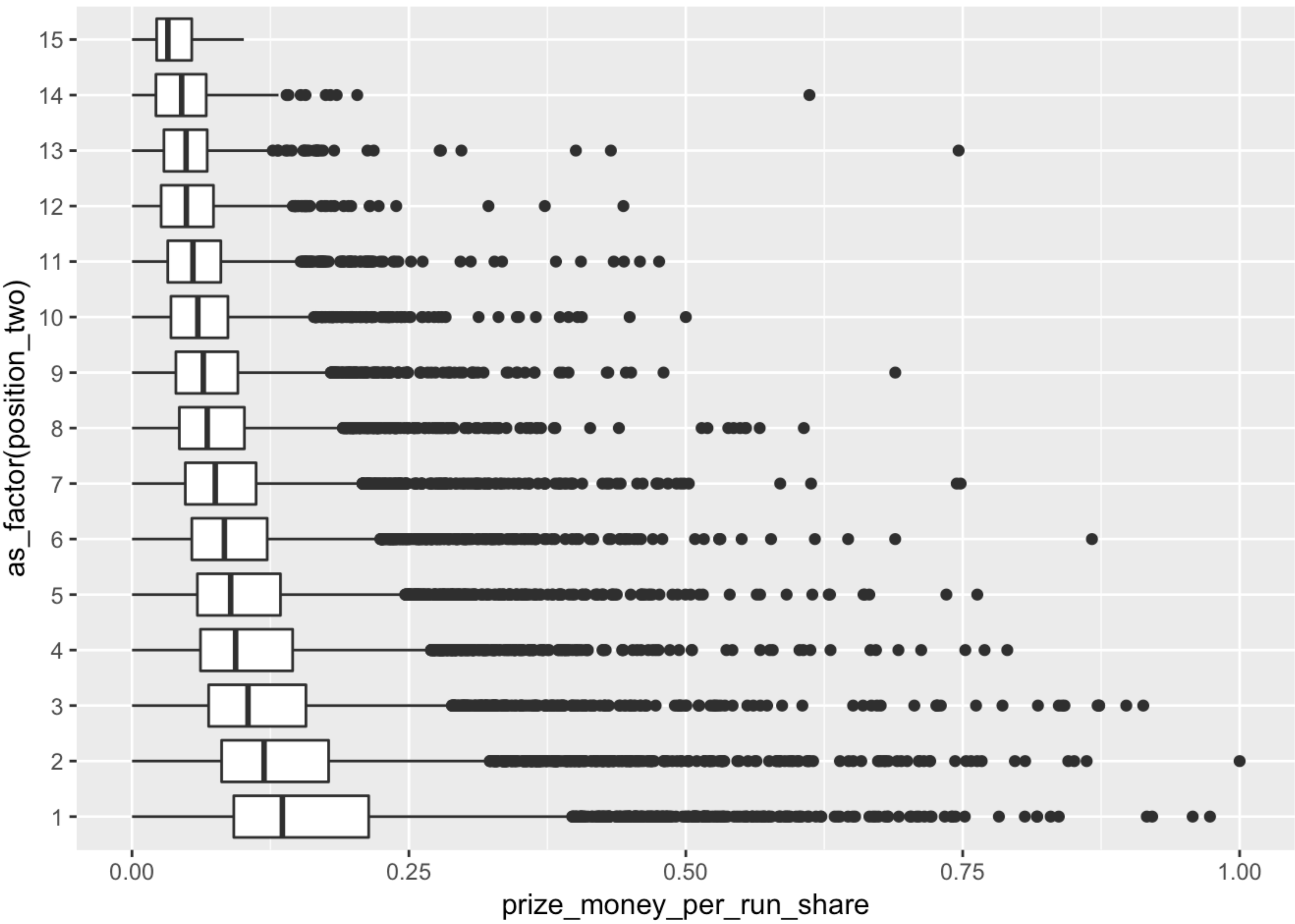
The relationship between `prize_money_share` and `odds_rank` does not seem very strong. Perhaps `prize_money_share` is not being taken into account as much as it should be.

```
eda_data %>%
  drop_na() %>%
  filter(mean_final_odds <= 20) %>%
  mutate(show_prob = mean(show)) %>%
  arrange(desc(show_prob)) %>%
  ggplot(aes(x = as_factor(odds_rank), y = prize_money_per_run_share)) +
  geom_boxplot()
```



Compared to the previous plot, it seems that `prize_money_per_run_share` may be a better predictor for `odds_rank` than just `prize_money_share`. This makes sense, since we are correcting for a horse's total number of starts.

```
eda_data %>%
  drop_na() %>%
  filter(overall_starts > 0, position_two < 16, position_two > 0) %>%
  arrange(desc(prize_money_per_run_share)) %>%
  ggplot(aes(x = as_factor(position_two), y = prize_money_per_run_share)) +
  geom_boxplot() +
  coord_flip()
```



`prize_money_per_run_share` appears to be just as closely related to final position as it is to final odds.

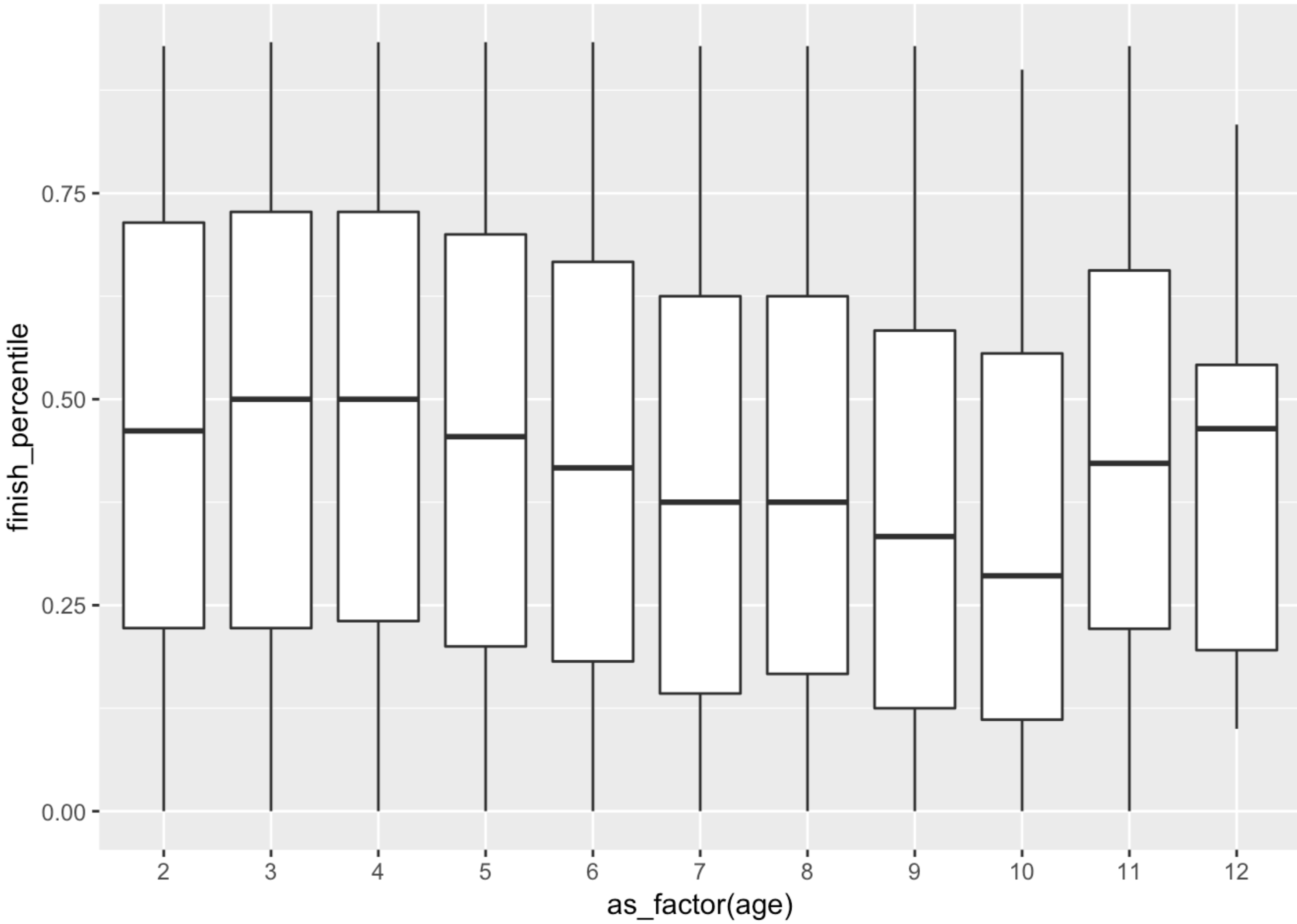
```
eda_data %>%
  drop_na() %>%
  select(tip_pundit_win, tip_recent_win, finish_percentile) %>%
  cor()
```

```
## Adding missing grouping variables: `market_id`
```

```
##                       market_id tip_pundit_win tip_recent_win
## market_id          1.0000000000   0.0008697498   -0.004418709
## tip_pundit_win     0.0008697498   1.0000000000    0.168494287
## tip_recent_win    -0.0044187091   0.1684942871    1.000000000
## finish_percentile -0.0039317944   0.1972224186    0.141593072
##                   finish_percentile
## market_id              -0.003931794
## tip_pundit_win          0.197222419
## tip_recent_win          0.141593072
## finish_percentile       1.000000000
```

The presence of a horse recommendation from a pundit is more correlated with `finish_percentile` than whether or not the horse has a recent win.

```
eda_data %>%
  drop_na() %>%
  ggplot(aes(x = as_factor(age), y = finish_percentile)) +
  geom_boxplot()
```



A

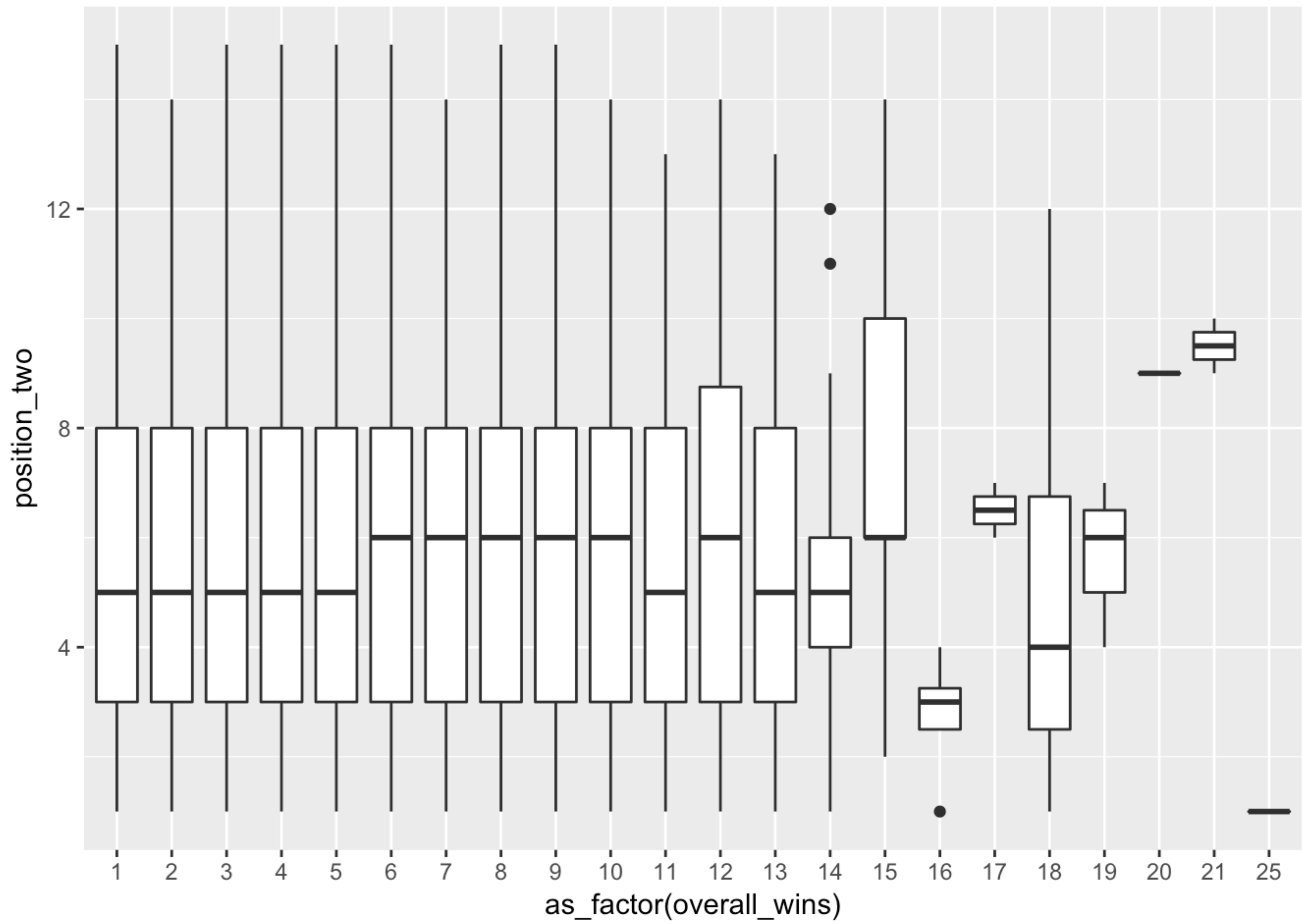
horse's age seems to be negatively correlated with it's finish percentile. Although there seems to be an increase for horses that are extremely old. This is surprising, we will examine the sample size of these old horses.

```
eda_data %>%
  drop_na() %>%
  filter(age > 10)
```

```
## # A tibble: 46 x 77
## # Groups:   market_id [46]
##    position_two bf_odds bf_odds_two_min… vic_tote vic_tote_two_mi… nsw_tote
##           <int>   <dbl>            <dbl>    <dbl>            <dbl>    <dbl>
##  1           14    18               12       23               19.1     19.8
##  2           13    65               70       27.6             26.1     28.3
##  3            2    22               21       22.1             22.7     25.3
##  4            1    14               13       18.3             11       14.2
##  5            1     6.8              6.8      6.4              5.9      6.1
##  6           10   140               70       43.7             24.9     77.4
##  7            4    40               44       30.7             29.5     27.2
##  8            4     7.6              8.4      6.1              5.3      5.7
##  9            9     8.2              7.6      9.1              9.8      8.9
## 10            4    10.5              9.4      8.2              7.2      8
## # … with 36 more rows, and 71 more variables: nsw_tote_two_mins_out <dbl>,
## #   nsw_odds <dbl>, market_id <int>, race_number <int>, number <int>,
## #   barrier <int>, tech_form_rating <int>, total_rating_points <int>,
## #   handicap_weight <dbl>, tip_pundit_win <dbl>, tip_recent_win <dbl>,
## #   prize_money <dbl>, age <int>, days_since_last_run <int>,
## #   overall_starts <int>, overall_wins <int>, overall_places <int>,
## #   track_starts <int>, track_wins <int>, track_places <int>,
## #   firm_starts <int>, firm_wins <int>, firm_places <int>,
## #   good_starts <int>, good_wins <int>, good_places <int>,
## #   dead_starts <int>, dead_wins <int>, dead_places <int>,
## #   slow_starts <int>, slow_wins <int>, slow_places <int>,
## #   soft_starts <int>, soft_wins <int>, soft_places <int>,
## #   heavy_starts <int>, heavy_wins <int>, heavy_places <int>,
## #   distance_starts <int>, distance_wins <int>, distance_places <int>,
## #   class_same_starts <int>, class_same_wins <int>,
## #   class_same_places <int>, class_stronger_starts <int>,
## #   class_stronger_wins <int>, class_stronger_places <int>,
## #   first_up_starts <int>, first_up_wins <int>, first_up_places <int>,
## #   second_up_starts <int>, second_up_wins <int>, second_up_places <int>,
## #   track_distance_starts <int>, track_distance_wins <int>,
## #   track_distance_places <int>, num_racers <dbl>, tot_prize_money <dbl>,
## #   finish_percentile <dbl>, prize_money_share <dbl>, win <dbl>,
## #   place <dbl>, show <dbl>, mean_final_odds <dbl>,
## #   prize_money_per_run <dbl>, win_rate <dbl>, place_rate <dbl>,
## #   tot_prize_money_per_run <dbl>, prize_money_per_run_share <dbl>,
## #   odds_rank <int>, prize_money_per_run_rank <dbl>
```

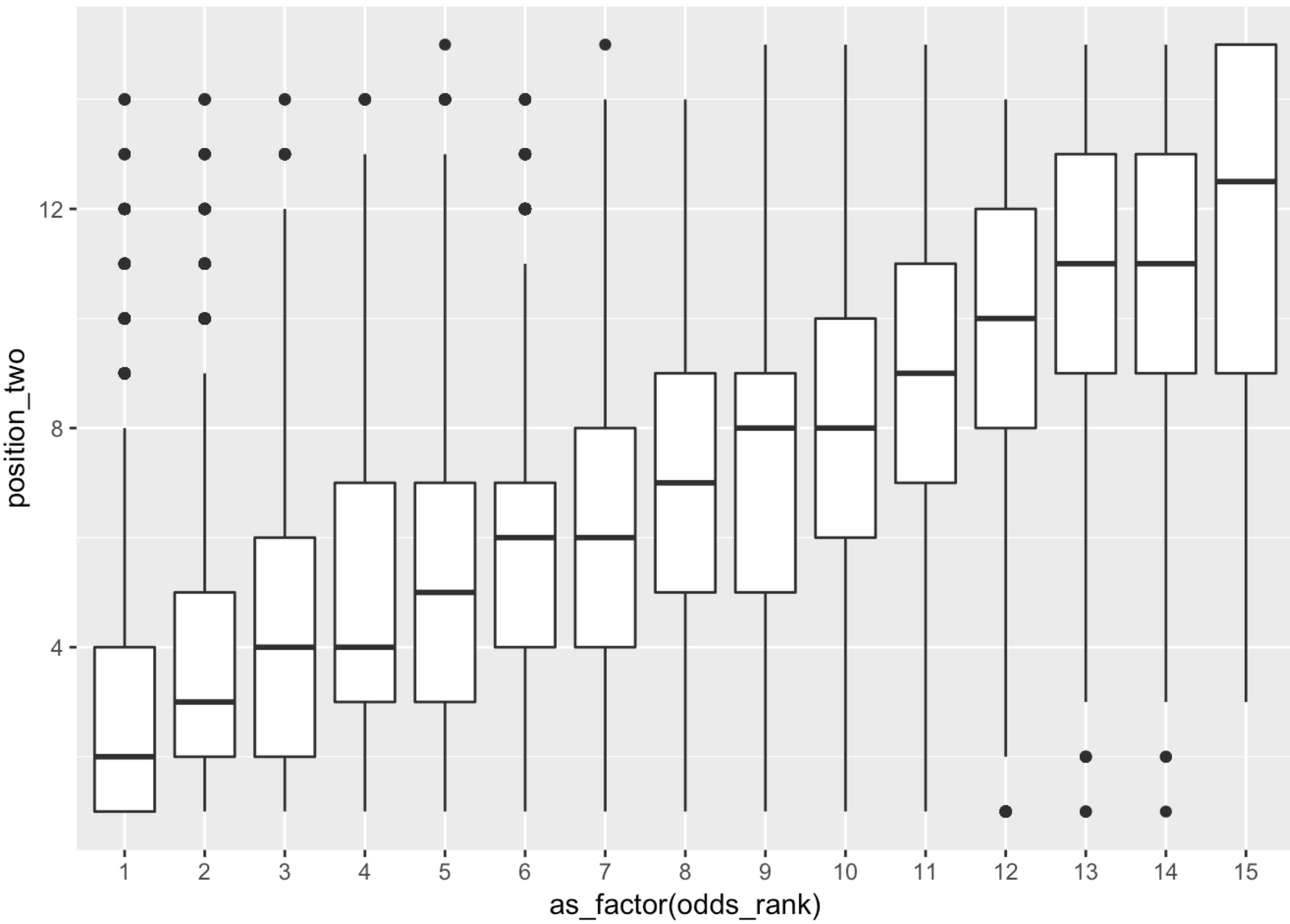
There are only 46 occurrences of horses competing that were older than 10 years old. Therefore, we can not make any confident conclusions of this peculiar relationships.

```
eda_data %>%
  filter(overall_wins >= 1) %>%
  ggplot(aes(x = as_factor(overall_wins), y = position_two)) +
  geom_boxplot()
```



Overall wins on it’s own does not appear to be a good predictor for final position of a horse.

```
eda_data %>%
  drop_na() %>%
  filter(num_racers <= 15) %>%
  ggplot(aes(x = as_factor(odds_rank), y = position_two)) +
  geom_boxplot()
```



We previously examined this relationship with regard to probabilites of win/place/show compared to odds rank, but now we compare the odds rank to the final position. The trend is clearly increasing. However, horses ranked 3rd and 4th appear to have the same average finish. This is surprising, and suggests that it is not worth it to bet on the 3rd ranked horse, since similar finishes can be acheived by the 4th ranked horse.

```
eda_data %>%
  drop_na() %>%
  filter(odds_rank == 3 | odds_rank == 4) %>%
  group_by(odds_rank) %>%
  mutate(mean_odds_by_rank = mean(mean_final_odds),
         sd_odds_by_rank = sd(mean_final_odds)) %>%
  select(odds_rank, mean_odds_by_rank, sd_odds_by_rank) %>%
  unique()
```

```
## # A tibble: 2 x 3
## # Groups:   odds_rank [2]
##   odds_rank mean_odds_by_rank sd_odds_by_rank
##       <int>             <dbl>           <dbl>
## 1         4              9.30            3.35
## 2         3              6.98            2.03
```

We dig in further to this observation. Since the odds between the 3rd and 4th ranked horses appear to be substantially different, it looks even moreso that betting on the 4th ranked horse makes more sense than betting on the 3rd ranked horse.

Model Building

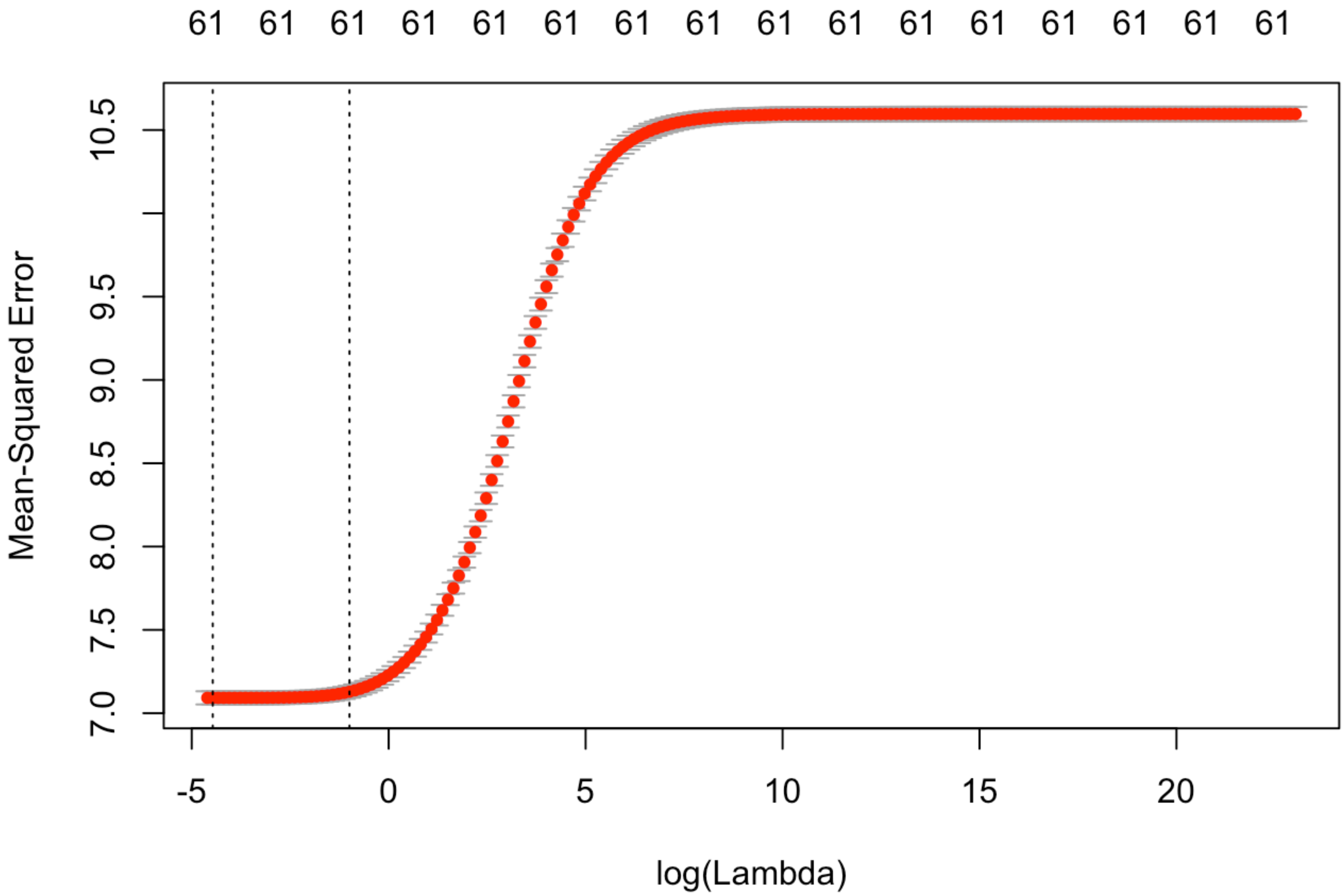
```
train_dat <- non_eda_data %>%
  sample_frac(0.70)

test_dat <- non_eda_data %>%
  setdiff(train_dat)
```

For our model developing and testing split, we use 70% of our non-eda data for model developing, and 30% of the non-eda data for testing our models.

Ridge

We will use `position_two` as the response variable

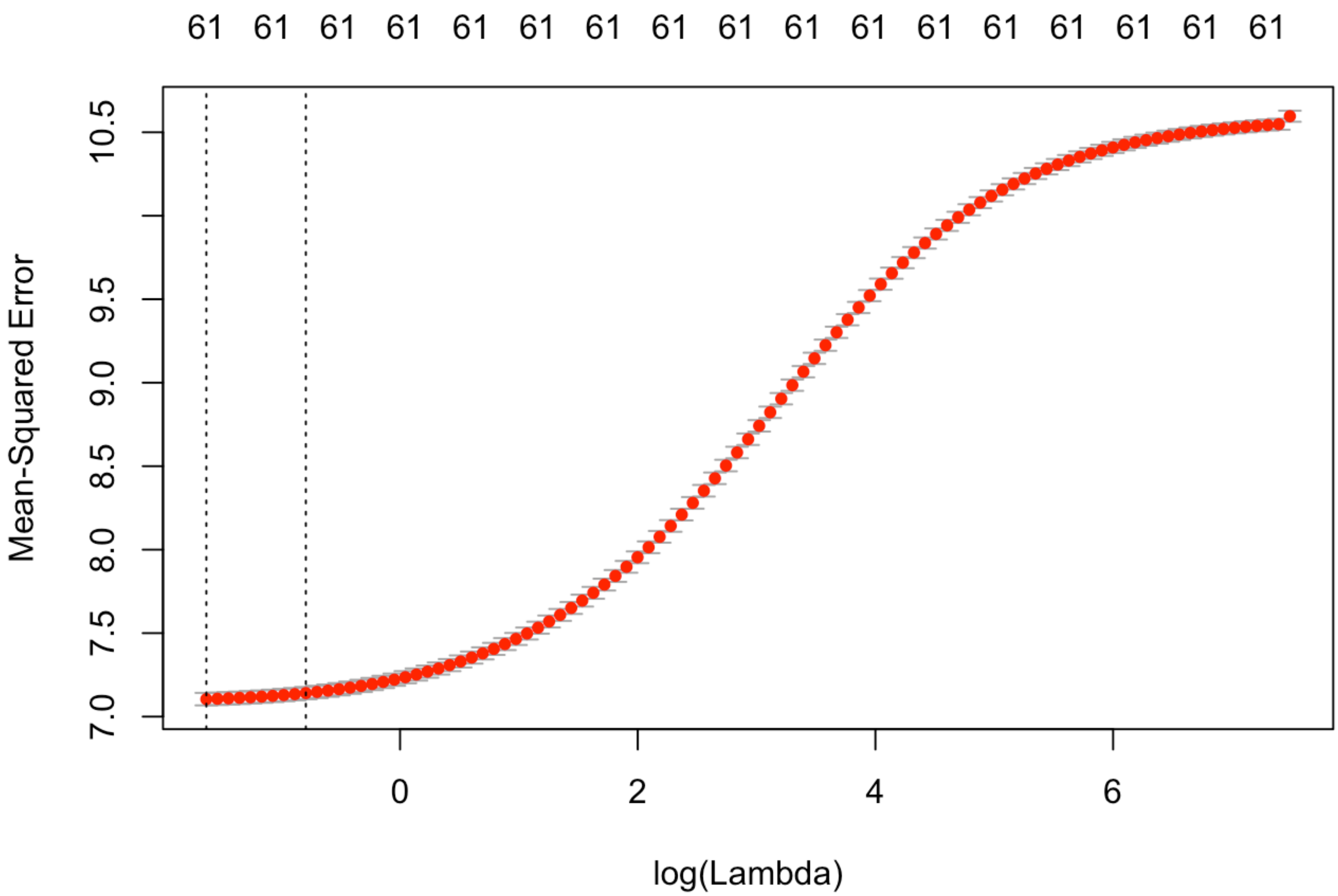
```
lambda_grid <- 10^seq(-2, 10, length = 200)

position_two_train_dat <- train_dat %>%
  dplyr::select(-bf_odds,-bf_odds_two_mins_out,-vic_tote, -vic_tote_two_mins_out,
                -vic_tote_two_mins_out, -nsw_tote, -nsw_tote_two_mins_out, -nsw_odds,
                -win, -show, -place, -finish_percentile, -num_racers,
                -prize_money_share,-prize_money_per_run, -prize_money_per_run_rank)

ridge_cv <- position_two_train_dat %>%
  cv.glmnet(formula = position_two ~ .,
            data = ., alpha = 0, nfolds = 10, lambda = lambda_grid)

plot(ridge_cv)
```




```
ridge_lambda_min <- ridge_cv$lambda.min
ridge_lambda_1se <- ridge_cv$lambda.1se
```

Lasso

```
lasso_cv <- position_two_train_dat %>%
  cv.glmnet(formula = position_two ~ .,
            data = ., alpha = 0, nfolds = 10)

plot(lasso_cv)
```




```
lasso_lambda_1se <- lasso_cv$lambda.1se
lasso_lambda_min <- lasso_cv$lambda.min


data_glmnet <- tibble(train = position_two_train_dat %>% list(),
                      test  = test_dat %>% list()) %>%
  mutate(ridge_min = map(train, ~ glmnet(position_two ~ ., data = .x,
                                         alpha = 0, lambda = ridge_lambda_min)),
         ridge_1se = map(train, ~ glmnet(position_two ~ ., data = .x,
                                         alpha = 0, lambda = ridge_lambda_1se)),
         lasso_min = map(train, ~ glmnet(position_two ~ ., data = .x,
                                         alpha = 1, lambda = lasso_lambda_min)),
         lasso_1se = map(train, ~ glmnet(position_two ~ ., data = .x,
                                         alpha = 1, lambda = lasso_lambda_1se))) %>%
  gather(key = method, value = fit, -test, -train)
```

We have built the models, and can now examine their compositions.

```
data_glmnet %>%
  pluck("fit") %>%
  map( ~ coef(.x) %>%
         as.matrix() %>%
         as.data.frame() %>%
         rownames_to_column("name")) %>%
  reduce(full_join, by = "name") %>%
  mutate_if(is.double, ~ if_else(. == 0, NA_real_, .)) %>%
  rename(ridge_min = s0.x,
         ridge_1se = s0.y,
         lasso_min = s0.x.x,
         lasso_1se = s0.y.y) %>%
  knitr::kable(digits = 3)
```

name	ridge_min	ridge_1se	lasso_min	lasso_1se
(Intercept)	-1.559	0.110	3.427	3.545
market_id	0.000	0.000	NA	NA
race_number	0.050	0.053	NA	NA
number	0.032	0.033	NA	NA
barrier	0.053	0.057	0.016	NA
tech_form_rating	0.019	0.007	NA	NA
total_rating_points	-0.015	-0.017	NA	NA
handicap_weight	0.053	0.047	NA	NA
tip_pundit_win	-0.110	-0.211	NA	NA
tip_recent_win	0.045	0.005	NA	NA
prize_money	0.000	0.000	NA	NA
age	-0.052	-0.022	NA	NA
days_since_last_run	0.000	0.000	NA	NA
overall_starts	-0.005	0.000	NA	NA
overall_wins	-0.001	0.007	NA	NA
overall_places	-0.008	-0.004	NA	NA
track_starts	-0.015	-0.006	NA	NA
track_wins	0.020	-0.003	NA	NA
track_places	-0.010	-0.019	NA	NA
firm_starts	0.011	0.007	NA	NA

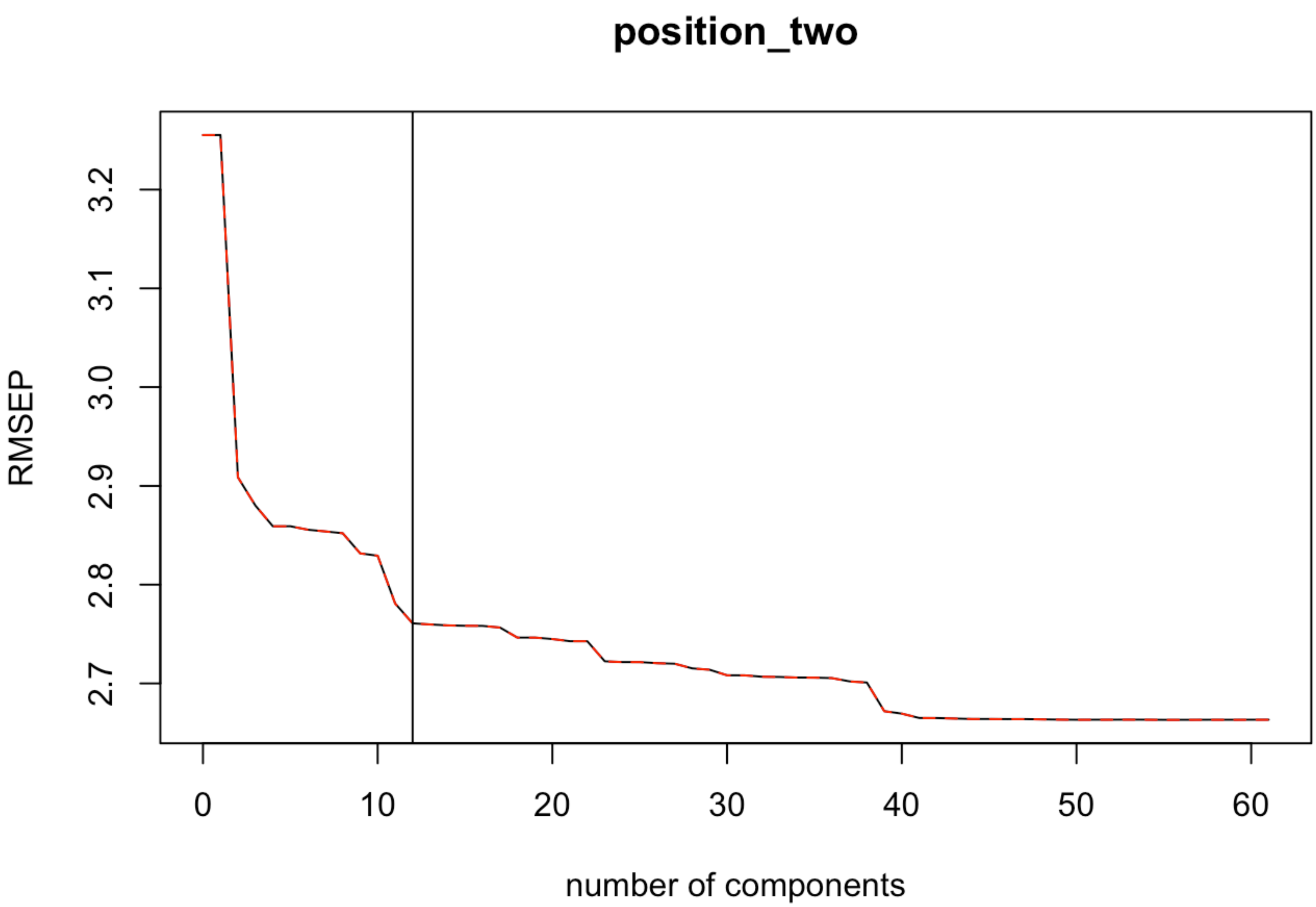
firm_wins	0.003	0.013	NA	NA
firm_places	0.034	0.037	NA	NA
good_starts	0.006	-0.001	NA	NA
good_wins	0.007	0.005	NA	NA
good_places	-0.006	-0.004	NA	NA
dead_starts	0.002	-0.001	NA	NA
dead_wins	-0.044	-0.026	NA	NA
dead_places	-0.013	-0.013	NA	NA
slow_starts	-0.009	-0.008	NA	NA
slow_wins	-0.039	-0.014	NA	NA
slow_places	0.009	-0.003	NA	NA
soft_starts	0.002	-0.005	NA	NA
soft_wins	0.042	0.017	NA	NA
soft_places	-0.027	-0.020	NA	NA
heavy_starts	-0.003	-0.006	NA	NA
heavy_wins	0.006	0.002	NA	NA
heavy_places	-0.002	-0.009	NA	NA
distance_starts	-0.001	-0.002	NA	NA
distance_wins	-0.008	-0.006	NA	NA
distance_places	-0.021	-0.017	NA	NA
class_same_starts	0.005	0.005	NA	NA
class_same_wins	0.033	0.033	NA	NA
class_same_places	0.009	0.006	NA	NA
class_stronger_starts	0.006	0.003	NA	NA
class_stronger_wins	0.051	0.054	NA	NA
class_stronger_places	0.007	0.008	NA	NA
first_up_starts	0.046	0.017	NA	NA
first_up_wins	-0.034	-0.032	NA	NA
first_up_places	0.011	0.010	NA	NA

second_up_starts	-0.041	-0.013	NA	NA
second_up_wins	0.020	0.012	NA	NA
second_up_places	0.013	0.005	NA	NA
track_distance_starts	0.009	0.001	NA	NA
track_distance_wins	-0.021	-0.012	NA	NA
track_distance_places	-0.020	-0.016	NA	NA
tot_prize_money	0.000	0.000	NA	NA
mean_final_odds	0.012	0.014	0.006	0.002
win_rate	0.709	0.459	NA	NA
place_rate	0.400	0.231	NA	NA
tot_prize_money_per_run	0.000	0.000	NA	NA
prize_money_per_run_share	-4.785	-4.352	-2.896	-1.038
odds_rank	0.424	0.336	0.404	0.384

It is clear that the Lasso m	odel pushes	almost all o	f the predic	tors to 0, while the Ridge model keeps them just close to 0.

PCR

```
pcr_cv <- position_two_train_dat %>%
  pcr(position_two ~ ., data = ., scale = TRUE, validation = "CV")

validationplot(pcr_cv)
abline(v=12)
```




```
pcr_cv %>%
  summary()
```

```
## Data:    X dimension: 98861 61
##  Y dimension: 98861 1
## Fit method: svdpc
## Number of components considered: 61
##
## VALIDATION: RMSEP
## Cross-validated using 10 random segments.
##        (Intercept)  1 comps  2 comps  3 comps  4 comps  5 comps  6 comps
## CV           3.255    3.255    2.909     2.88    2.859    2.859    2.856
## adjCV        3.255    3.255    2.909     2.88    2.859    2.859    2.856
##        7 comps  8 comps  9 comps  10 comps  11 comps  12 comps  13 comps
## CV       2.854    2.852    2.832     2.829     2.781     2.761      2.76
## adjCV    2.854    2.852    2.832     2.829     2.781     2.761      2.76
##        14 comps  15 comps  16 comps  17 comps  18 comps  19 comps
## CV        2.759     2.758     2.758     2.757     2.746     2.746
## adjCV     2.759     2.758     2.758     2.757     2.746     2.746
##        20 comps  21 comps  22 comps  23 comps  24 comps  25 comps
## CV        2.745     2.743     2.743     2.722     2.722     2.722
## adjCV     2.745     2.743     2.743     2.722     2.722     2.722
```

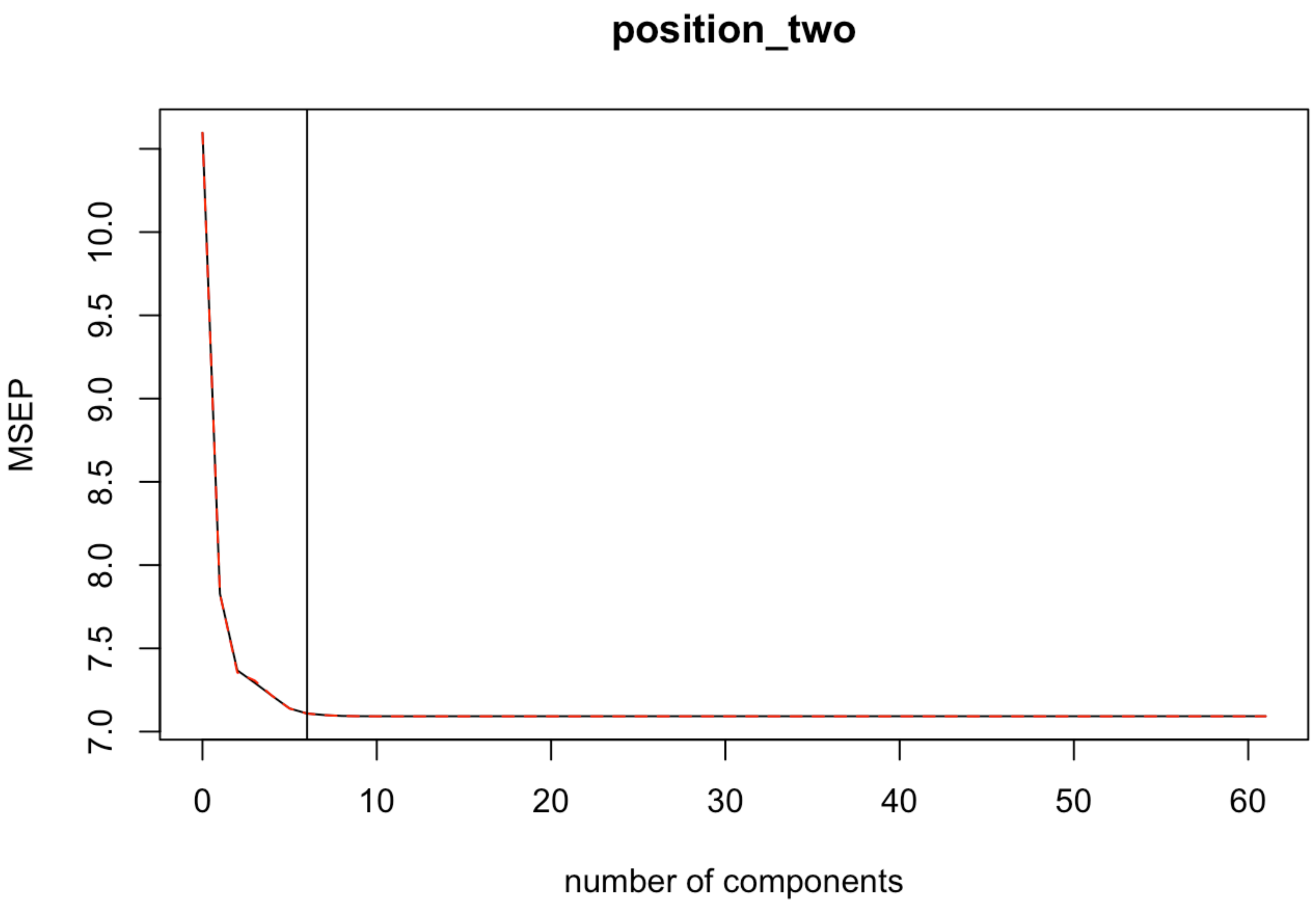
```
##        26 comps  27 comps  28 comps  29 comps  30 comps  31 comps
## CV         2.72      2.72     2.715     2.714     2.708     2.708
## adjCV      2.72      2.72     2.715     2.714     2.708     2.708
##        32 comps  33 comps  34 comps  35 comps  36 comps  37 comps
## CV        2.707     2.707     2.706     2.706     2.705     2.702
## adjCV     2.707     2.707     2.706     2.706     2.705     2.702
##        38 comps  39 comps  40 comps  41 comps  42 comps  43 comps
## CV        2.701     2.672     2.669     2.665     2.665     2.664
## adjCV     2.701     2.672     2.669     2.665     2.665     2.664
##        44 comps  45 comps  46 comps  47 comps  48 comps  49 comps
## CV        2.664     2.664     2.664     2.664     2.664     2.663
## adjCV     2.664     2.664     2.664     2.664     2.664     2.663
##        50 comps  51 comps  52 comps  53 comps  54 comps  55 comps
## CV        2.663     2.663     2.663     2.663     2.663     2.663
## adjCV     2.663     2.663     2.663     2.663     2.663     2.663
##        56 comps  57 comps  58 comps  59 comps  60 comps  61 comps
## CV        2.663     2.663     2.663     2.663     2.663     2.663
## adjCV     2.663     2.663     2.663     2.663     2.663     2.663
## 
## TRAINING: % variance explained
##                 1 comps  2 comps  3 comps  4 comps  5 comps  6 comps
## X             2.907e+01    36.94    42.89    48.66    52.26    55.64
## position_two  1.051e-04    20.16    21.72    22.86    22.86    23.05
##               7 comps  8 comps  9 comps  10 comps  11 comps  12 comps
## X               58.59    61.25    63.50     65.43     67.25     69.05
## position_two    23.15    23.25    24.35     24.48     27.04     28.10
##               13 comps  14 comps  15 comps  16 comps  17 comps  18 comps
## X                70.79     72.48     74.13     75.76     77.25     78.68
## position_two     28.15     28.20     28.22     28.22     28.31     28.85
##               19 comps  20 comps  21 comps  22 comps  23 comps  24 comps
## X                80.04     81.36     82.63     83.88     85.08     86.22
## position_two     28.85     28.93     29.04     29.04     30.09     30.13
##               25 comps  26 comps  27 comps  28 comps  29 comps  30 comps
## X                87.29     88.32     89.15     89.96     90.75     91.50
## position_two     30.13     30.20     30.22     30.46     30.54     30.83
##               31 comps  32 comps  33 comps  34 comps  35 comps  36 comps
## X                92.21     92.87     93.47     94.06     94.55     95.01
## position_two     30.83     30.91     30.92     30.95     30.96     30.98
##               37 comps  38 comps  39 comps  40 comps  41 comps  42 comps
## X                95.44     95.87     96.24     96.61     96.96     97.29
## position_two     31.16     31.21     32.69     32.81     33.03     33.04
##               43 comps  44 comps  45 comps  46 comps  47 comps  48 comps
## X                97.62     97.92     98.19     98.45     98.69     98.91
## position_two     33.06     33.09     33.09     33.10     33.10     33.11
##               49 comps  50 comps  51 comps  52 comps  53 comps  54 comps
## X                99.12     99.30     99.46     99.57     99.68     99.77
## position_two     33.13     33.14     33.14     33.14     33.14     33.15
##               55 comps  56 comps  57 comps  58 comps  59 comps  60 comps
## X                99.86     99.93     99.97    100.00    100.00    100.00
## position_two     33.15     33.15     33.15     33.15     33.16     33.16
```

```
##                   61 comps
## X                  100.00
## position_two        33.16
```

We use 12 components because this a good balance between CV and model complication.

PLS

```
pls_cv <- position_two_train_dat %>%
  plsr(position_two ~., data = ., scale = TRUE, validation = "CV")

validationplot(pls_cv, val.type = "MSEP")
abline(v=6)
```




```
pls_cv %>%
  summary()
```

```
## Data:    X dimension: 98861 61
##  Y dimension: 98861 1
```

```
## Fit method: kernelpls
## Number of components considered: 61
##
## VALIDATION: RMSEP
## Cross-validated using 10 random segments.
##        (Intercept)  1 comps  2 comps  3 comps  4 comps  5 comps  6 comps
## CV           3.255    2.798    2.714    2.701    2.686    2.672    2.666
## adjCV        3.255    2.798    2.712    2.703    2.686    2.672    2.666
##        7 comps  8 comps  9 comps  10 comps  11 comps  12 comps  13 comps
## CV       2.664    2.664    2.663     2.663     2.663     2.663     2.663
## adjCV    2.664    2.663    2.663     2.663     2.663     2.663     2.663
##        14 comps  15 comps  16 comps  17 comps  18 comps  19 comps
## CV        2.663     2.663     2.663     2.663     2.663     2.663
## adjCV     2.663     2.663     2.663     2.663     2.663     2.663
##        20 comps  21 comps  22 comps  23 comps  24 comps  25 comps
## CV        2.663     2.663     2.663     2.663     2.663     2.663
## adjCV     2.663     2.663     2.663     2.663     2.663     2.663
##        26 comps  27 comps  28 comps  29 comps  30 comps  31 comps
## CV        2.663     2.663     2.663     2.663     2.663     2.663
## adjCV     2.663     2.663     2.663     2.663     2.663     2.663
##        32 comps  33 comps  34 comps  35 comps  36 comps  37 comps
## CV        2.663     2.663     2.663     2.663     2.663     2.663
## adjCV     2.663     2.663     2.663     2.663     2.663     2.663
##        38 comps  39 comps  40 comps  41 comps  42 comps  43 comps
## CV        2.663     2.663     2.663     2.663     2.663     2.663
## adjCV     2.663     2.663     2.663     2.663     2.663     2.663
##        44 comps  45 comps  46 comps  47 comps  48 comps  49 comps
## CV        2.663     2.663     2.663     2.663     2.663     2.663
## adjCV     2.663     2.663     2.663     2.663     2.663     2.663
##        50 comps  51 comps  52 comps  53 comps  54 comps  55 comps
## CV        2.663     2.663     2.663     2.663     2.663     2.663
## adjCV     2.663     2.663     2.663     2.663     2.663     2.663
##        56 comps  57 comps  58 comps  59 comps  60 comps  61 comps
## CV        2.663     2.663     2.663     2.663     2.663     2.663
## adjCV     2.663     2.663     2.663     2.663     2.663     2.663
##
## TRAINING: % variance explained
##               1 comps  2 comps  3 comps  4 comps  5 comps  6 comps
## X               7.622    11.69    36.30    44.31    46.19    48.70
## position_two   26.155    30.58    31.14    31.97    32.70    32.99
##               7 comps  8 comps  9 comps  10 comps  11 comps  12 comps
## X               52.63    55.58    57.55     59.28     61.22     62.67
## position_two    33.08    33.13    33.14     33.15     33.15     33.15
##               13 comps  14 comps  15 comps  16 comps  17 comps  18 comps
## X                64.95     66.76     68.68     69.69     70.76     71.87
## position_two     33.15     33.15     33.15     33.15     33.15     33.15
##               19 comps  20 comps  21 comps  22 comps  23 comps  24 comps
## X                72.80     73.91     74.94     75.70     76.70     77.64
## position_two     33.15     33.15     33.15     33.15     33.15     33.15
##               25 comps  26 comps  27 comps  28 comps  29 comps  30 comps
```

```
## X                78.60     79.40     80.46     81.18     82.33     83.09
## position_two     33.15     33.15     33.15     33.15     33.15     33.15
##               31 comps  32 comps  33 comps  34 comps  35 comps  36 comps
## X                83.90     84.75     85.78     86.52     87.33     88.00
## position_two     33.15     33.15     33.15     33.16     33.16     33.16
##               37 comps  38 comps  39 comps  40 comps  41 comps  42 comps
## X                88.56     89.29     90.35     91.01     91.93     92.58
## position_two     33.16     33.16     33.16     33.16     33.16     33.16
##               43 comps  44 comps  45 comps  46 comps  47 comps  48 comps
## X                93.40     93.89     94.38     95.10     95.85     96.41
## position_two     33.16     33.16     33.16     33.16     33.16     33.16
##               49 comps  50 comps  51 comps  52 comps  53 comps  54 comps
## X                96.91     97.31     97.68     98.00     98.30     98.62
## position_two     33.16     33.16     33.16     33.16     33.16     33.16
##               55 comps  56 comps  57 comps  58 comps  59 comps  60 comps
## X                98.72     98.97     99.16     99.42     99.67    100.00
## position_two     33.16     33.16     33.16     33.16     33.16     33.16
##               61 comps
## X               100.00
## position_two     33.16
```

We use 6 components because this a good balance between CV and model complication.

```
glmnet_error <- data_glmnet %>%
  mutate(pred = map2(fit, test, predict),
         test_mse = map2_dbl(test, pred, ~ mean((.x$position_two - .y)^2)),
         test_rmse = map2_dbl(test, pred, ~ sqrt(mean((.x$position_two - .y)^2)))) %>
%
  unnest(test_mse, .drop = TRUE)


data_dim_reduct <- tibble(train = position_two_train_dat %>% list(),
                          test  = test_dat %>% list()) %>%
  mutate(pcr_12m = map(train, ~ pcr(position_two ~ ., data = .x, ncomp = 12)),
         pls_6m = map(train, ~ plsr(position_two ~ ., data = .x, ncomp = 6))) %>%
  gather(key = method, value = fit, -test, -train)


dim_reduce_error <- data_dim_reduct %>%
  mutate(pred = pmap(list(fit, test, c(12,6)), predict),
         test_mse = map2_dbl(test, pred, ~ mean((.x$position_two - .y)^2)),
         test_rmse = map2_dbl(test, pred, ~ sqrt(mean((.x$position_two - .y)^2)))) %>
%
  unnest(test_mse, .drop = TRUE)

dim_reduce_error %>%
  bind_rows(glmnet_error) %>%
  arrange(test_mse) %>%
  knitr::kable(digits = 2)
```

method	test_rmse	test_mse
ridge_min	2.66	7.07
ridge_1se	2.66	7.10
lasso_min	2.69	7.23
lasso_1se	2.74	7.50
pcr_12m	2.85	8.12
pls_6m	2.87	8.24

```
loadings(data_dim_reduct$fit[[1]])
```

```
##
## Loadings:
##                              Comp 1 Comp 2 Comp 3 Comp 4 Comp 5 Comp 6 Comp 7
```

```
## market_id                                              -0.999
## race_number
## number
## barrier
## tech_form_rating                                                      0.230
## total_rating_points
## handicap_weight
## tip_pundit_win
## tip_recent_win
## prize_money                                 0.995
## age
## days_since_last_run                                          -0.998
## overall_starts                                                              -0.649
## overall_wins
## overall_places                                                              -0.163
## track_starts
## track_wins
## track_places
## firm_starts
## firm_wins
## firm_places
## good_starts                                                                 -0.456
## good_wins
## good_places                                                                 -0.115
## dead_starts                                                                 -0.232
## dead_wins
## dead_places
## slow_starts
## slow_wins
## slow_places
## soft_starts                                                                 -0.133
## soft_wins
## soft_places
## heavy_starts
## heavy_wins
## heavy_places
## distance_starts                                                             -0.294
## distance_wins
## distance_places
## class_same_starts                                                           -0.166
## class_same_wins
## class_same_places
## class_stronger_starts                                                       -0.237
## class_stronger_wins
## class_stronger_places
## first_up_starts
## first_up_wins
## first_up_places
## second_up_starts
## second_up_wins
```

```
## second_up_places
## track_distance_starts
## track_distance_wins
## track_distance_places
## tot_prize_money           -0.994
## mean_final_odds                                                -0.956  0.146
## win_rate
## place_rate
## tot_prize_money_per_run                  0.998
## prize_money_per_run_share
## odds_rank
##                           Comp 8 Comp 9 Comp 10 Comp 11 Comp 12
## market_id
## race_number
## number                    -0.115                        -0.228
## barrier
## tech_form_rating           0.862                        -0.162
## total_rating_points        0.389                         0.206
## handicap_weight
## tip_pundit_win
## tip_recent_win
## prize_money
## age
## days_since_last_run
## overall_starts                          -0.208          -0.249
## overall_wins
## overall_places
## track_starts                      0.169        -0.369   0.539
## track_wins
## track_places                                    -0.108   0.151
## firm_starts
## firm_wins
## firm_places
## good_starts                                     -0.333   0.120
## good_wins
## good_places                                     -0.137
## dead_starts                                     -0.147  -0.150
## dead_wins
## dead_places
## slow_starts                                      0.118  -0.146
## slow_wins
## slow_places
## soft_starts                                      0.143  -0.245
## soft_wins
## soft_places
## heavy_starts                                     0.109  -0.124
## heavy_wins
## heavy_places
## distance_starts                   0.154  0.805   0.327
## distance_wins                            0.141
```

```
## distance_places                              0.241
## class_same_starts                     0.528 -0.375   0.591   0.289
## class_same_wins
## class_same_places                     0.132          0.133
## class_stronger_starts                -0.755          0.311   0.376
## class_stronger_wins
## class_stronger_places                -0.172
## first_up_starts
## first_up_wins
## first_up_places
## second_up_starts
## second_up_wins
## second_up_places
## track_distance_starts                 0.107  0.190  -0.123   0.269
## track_distance_wins
## track_distance_places
## tot_prize_money
## mean_final_odds               0.248
## win_rate
## place_rate
## tot_prize_money_per_run
## prize_money_per_run_share
## odds_rank
##
##                Comp 1 Comp 2 Comp 3 Comp 4 Comp 5 Comp 6 Comp 7 Comp 8
## SS loadings     1.000  1.000  1.000  1.000  1.000  1.000  1.000  1.000
## Proportion Var  0.016  0.016  0.016  0.016  0.016  0.016  0.016  0.016
## Cumulative Var  0.016  0.033  0.049  0.066  0.082  0.098  0.115  0.131
##                Comp 9 Comp 10 Comp 11 Comp 12
## SS loadings     1.000   1.000   1.000   1.000
## Proportion Var  0.016   0.016   0.016   0.016
## Cumulative Var  0.148   0.164   0.180   0.197
```

```
loadings(data_dim_reduct$fit[[2]])
```

```
##
## Loadings:
##                           Comp 1 Comp 2 Comp 3 Comp 4 Comp 5 Comp 6
## market_id                                      -1.024  0.235
## race_number
## number
## barrier
## tech_form_rating                                      -0.228
## total_rating_points
## handicap_weight
## tip_pundit_win
## tip_recent_win
## prize_money                      -0.994
## age
```

```
## days_since_last_run                                           0.233 -1.182
## overall_starts
## overall_wins
## overall_places
## track_starts
## track_wins
## track_places
## firm_starts
## firm_wins
## firm_places
## good_starts
## good_wins
## good_places
## dead_starts
## dead_wins
## dead_places
## slow_starts
## slow_wins
## slow_places
## soft_starts
## soft_wins
## soft_places
## heavy_starts
## heavy_wins
## heavy_places
## distance_starts
## distance_wins
## distance_places
## class_same_starts
## class_same_wins
## class_same_places
## class_stronger_starts
## class_stronger_wins
## class_stronger_places
## first_up_starts
## first_up_wins
## first_up_places
## second_up_starts
## second_up_wins
## second_up_places
## track_distance_starts
## track_distance_wins
## track_distance_places
## tot_prize_money             1.021 -0.132
## mean_final_odds                                               0.917
## win_rate
## place_rate
## tot_prize_money_per_run                   0.994 -0.105
## prize_money_per_run_share
## odds_rank
```

```
##
##                Comp 1 Comp 2 Comp 3 Comp 4 Comp 5 Comp 6
## SS loadings     1.054  1.006  1.004  1.059  1.031  1.413
## Proportion Var  0.017  0.016  0.016  0.017  0.017  0.023
## Cumulative Var  0.017  0.034  0.050  0.068  0.084  0.108
```

We compare the loadings on our PCR and PLS models. The PCR includes more predictors since we fed it 12 components instead of 6.

Now it ss time to compare all of the models we have run with final position as the response variable.

```
glmnet_error <- data_glmnet %>%
  mutate(pred = map2(fit, test, predict),
         test_mse = map2_dbl(test, pred, ~ mean((.x$position_two - .y)^2)),
         test_rmse = map2_dbl(test, pred, ~ sqrt(mean((.x$position_two - .y)^2)))) %>
%
  unnest(test_mse, .drop = TRUE)


data_dim_reduct <- tibble(train = position_two_train_dat %>% list(),
                          test  = test_dat %>% list()) %>%
  mutate(pcr_12m = map(train, ~ pcr(position_two ~ ., data = .x, ncomp = 12)),
         pls_6m = map(train, ~ plsr(position_two ~ ., data = .x, ncomp = 6))) %>%
  gather(key = method, value = fit, -test, -train)


dim_reduce_error <- data_dim_reduct %>%
  mutate(pred = pmap(list(fit, test, c(12,6)), predict),
         test_mse = map2_dbl(test, pred, ~ mean((.x$position_two - .y)^2)),
         test_rmse = map2_dbl(test, pred, ~ sqrt(mean((.x$position_two - .y)^2)))) %>
%
  unnest(test_mse, .drop = TRUE)

dim_reduce_error %>%
  bind_rows(glmnet_error) %>%
  arrange(test_mse) %>%
  knitr::kable(digits = 2)
```

method	test_rmse	test_mse
ridge_min	2.66	7.07
ridge_1se	2.66	7.10
lasso_min	2.69	7.23
lasso_1se	2.74	7.50
pcr_12m	2.85	8.12

pls_6m	2.87	8.24
We find that	the ridge r	egression with the minimum value of lambda has the lowest test_mse. It is surprising that the ridge models performed better than the lasso models. Perhaps the lasso models should not have pushed such a high number of coefficients to 0, as this may have caused the model to lose some predictive ability.
We find that	the PCR mod	el has a slightly lower test error than the PLS model. This makes sense, because the PCR model included 6 more predictors. However, I would argue that the PLS model is more useful. It has half as many predictors, so it is less complicated and more interpretable than the PCR. For this increased interpretability, we are only giving up a marginal amount of test error. The models have similar predictive power.
Broadly, the	models are	decent, but not great, predictors of final position. A standard error of roughly 2.7 positions does give us the ability to predict where a horse will finish with some confidence. However, the models do not appear to be strong enough to consistently make these predictions when money is on the line.
# Logistic		
Now we move	into the cla	ssification portion of the models, where we use win/place/show as our response variables. We end up focusing on `place` , since it is a good balance between bet payoff and predictability.

```r
data_log_db <- tibble(train = list(train_dat),
                      test = list(test_dat))

# fitting logistic models
glm_fits <- data_log_db %>%
  mutate(win_mod = map(train, glm,
                       formula = win ~ mean_final_odds + prize_money_per_run_share +
                         overall_starts + days_since_last_run + win_rate + place_rate
,
                       family = binomial),
         place_mod = map(train, glm,
                         formula = place ~ mean_final_odds + prize_money_per_run_shar
e +
                           overall_starts + days_since_last_run + win_rate + place_ra
te,
                         family = binomial),
         place_mod_no_odds = map(train, glm,
                         formula = place ~ prize_money_per_run_share +
                           overall_starts + days_since_last_run + win_rate + place_ra
te,
                         family = binomial),
         show_mod = map(train, glm,
                         formula = show ~ mean_final_odds + prize_money_per_run_share
+
                           overall_starts + days_since_last_run + win_rate + place_ra
te,
                         family = binomial))
```

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```r
glm_fits %>%
  pluck("win_mod", 1) %>%
  tidy()
```

```
## # A tibble: 7 x 5
##   term                        estimate std.error statistic   p.value
##   <chr>                          <dbl>     <dbl>     <dbl>     <dbl>
## 1 (Intercept)                -1.39      0.0391     -35.4   1.63e-274
## 2 mean_final_odds            -0.0972    0.00202    -48.1   0.
## 3 prize_money_per_run_share   3.17      0.104       30.5   3.51e-204
## 4 overall_starts              0.000115  0.000788     0.146 8.84e-  1
## 5 days_since_last_run        -0.000206  0.000244    -0.843 3.99e-  1
## 6 win_rate                   -0.0758    0.0701      -1.08  2.80e-  1
## 7 place_rate                 -0.108     0.0574      -1.88  5.97e-  2
```

```
glm_fits %>%
  pluck("place_mod", 1) %>%
  tidy()
```

```
## # A tibble: 7 x 5
##   term                         estimate std.error statistic   p.value
##   <chr>                           <dbl>     <dbl>     <dbl>     <dbl>
## 1 (Intercept)                -0.805       0.0295     -27.3   1.93e-164
## 2 mean_final_odds            -0.0697      0.00117    -59.8   0.
## 3 prize_money_per_run_share   3.79        0.0971      39.0   0.
## 4 overall_starts             -0.0000838   0.000580    -0.144 8.85e-  1
## 5 days_since_last_run        -0.000796    0.000190    -4.18  2.91e-  5
## 6 win_rate                   -0.149       0.0580      -2.56  1.04e-  2
## 7 place_rate                  0.0474      0.0463       1.03  3.05e-  1
```

```
glm_fits %>%
  pluck("show_mod", 1) %>%
  tidy()
```

```
## # A tibble: 7 x 5
##   term                        estimate std.error statistic  p.value
##   <chr>                          <dbl>     <dbl>     <dbl>    <dbl>
## 1 (Intercept)                -0.387     0.0256     -15.1   1.29e-51
## 2 mean_final_odds            -0.0546    0.000817   -66.8   0.
## 3 prize_money_per_run_share   4.38      0.0999      43.8   0.
## 4 overall_starts             -0.000212  0.000502    -0.422 6.73e- 1
## 5 days_since_last_run        -0.00106   0.000168    -6.32  2.56e-10
## 6 win_rate                   -0.288     0.0545      -5.28  1.29e- 7
## 7 place_rate                  0.0542    0.0429       1.26  2.06e- 1
```

Comparing our models for winning vs. placing vs. showing gives us some interesting results. `days_since_last_run` and `win_rate` are not significant in the `win` regression, but become significant in the `place` regression. They become even more signifcant in the `show` regression. This may tell the story of which variables to put more stock into when predicting wins vs places vs shows.

```r
glm_fits %>%
  pluck("win_mod", 1) %>%
  predict(type = "response") %>%
  skim()
```

```
##
## Skim summary statistics
##
## ── Variable type:numeric ──────────────────────────────────────────────────
##  variable missing complete     n mean    sd      p0   p25  p50  p75 p100
##         .       0    98861 98861  0.1 0.094 2.2e-16 0.024 0.09 0.16 0.83
##      hist
##  ▇▃▁▁▁▁▁▁
```

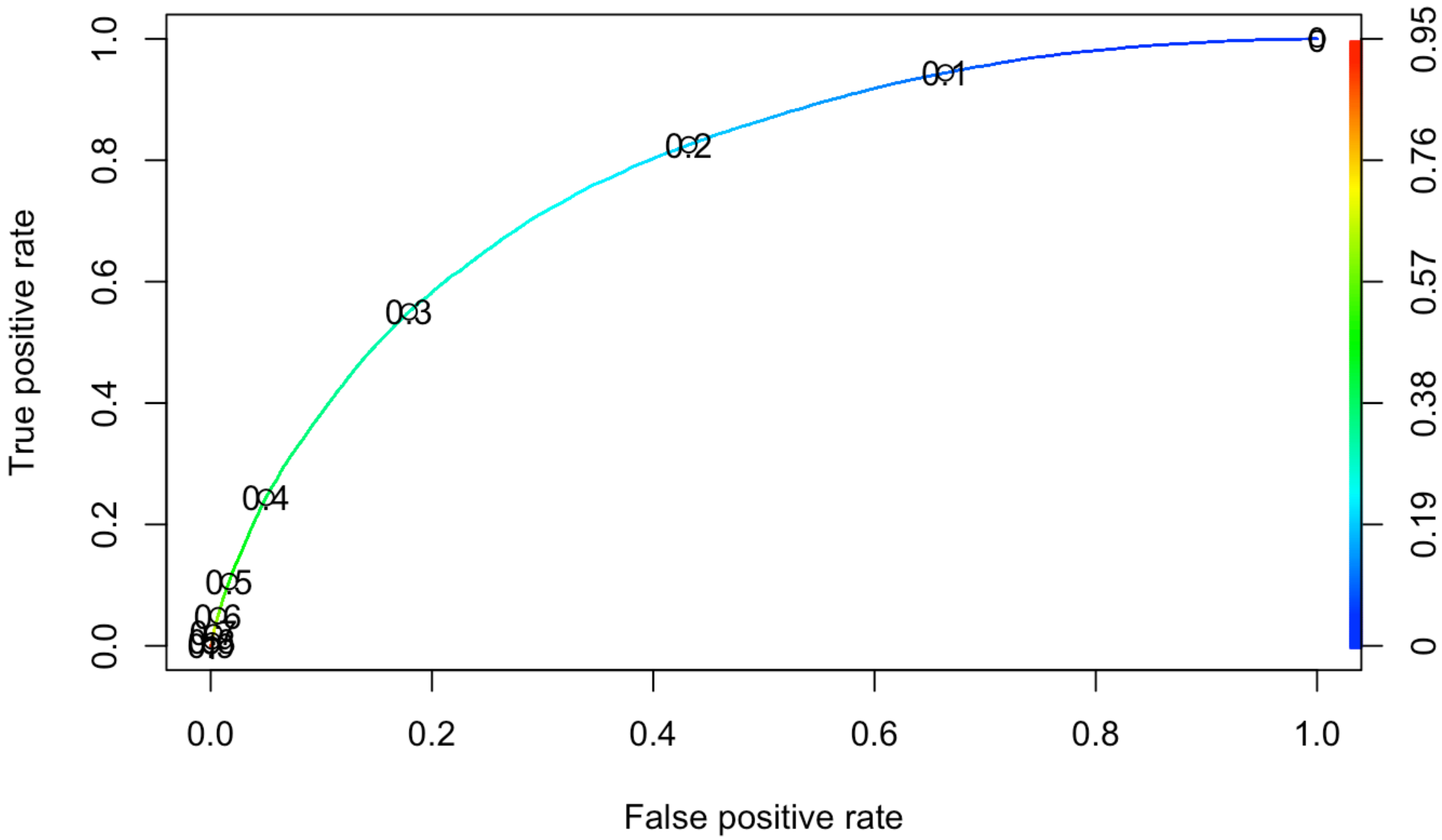
```r
# 50th percentile is 0.09, we will use this as the benchmark

glm_fits %>%
  pluck("place_mod", 1) %>%
  predict(type = "response") %>%
  skim()
```

```
##
## Skim summary statistics
##
## ── Variable type:numeric ──────────────────────────────────────────────────
##  variable missing complete     n mean   sd      p0   p25  p50 p75 p100
##         .       0    98861 98861 0.21 0.15 2.2e-16 0.084 0.21 0.3 0.95
##      hist
##  ▇▇▇▁▁▁▁▁
```

```r
# 50th percentile is .21, we will use this as the benchmark
```

We spot an issue with our model. The `win` model is only predicting a win 9% of the time. This is intuitive, but means that are model will predict very few wins. A similar pattern is true for our `place` model, which only predicts a place 21% of the time.

```
rocr_mod <- glm_fits %>%
  pluck("place_mod", 1)

rocr_train <- data_log_db %>% pluck("train", 1)

predicts <- predict(rocr_mod, rocr_train, type = "response")

rocr_preds <- prediction(predicts, rocr_train$place)

rocr_pref <- performance(rocr_preds, "tpr", "fpr")

plot(rocr_pref, colorize = TRUE, print.cutoffs.at = seq(0,1,by = 0.1))
```



We use an ROCR curve to see if we can find a better threshold to predict a place than 0.21. The curve shows us that 0.28 may be a better option. The proportion of false positives is high. However, in the context of betting, this may not be the case. It appears that this threshold value will still allow us to be accurate roughly 75% of the time, which is a good number by the standards of horse betting.

```
place_tib <- glm_fits %>%
  mutate(train_prob = map(place_mod, predict, type = "response"),
         train_place = map(train_prob, ~ if_else(.x > 0.28, "1", "0")))

place_tib %>%
  unnest(train, train_place) %>%
  mutate(correct = if_else(train_place == place, 1, 0)) %>%
  summarise(train_accuracy = mean(correct),
            train_error    = 1 - train_accuracy)
```

```
## # A tibble: 1 x 2
##   train_accuracy train_error
##            <dbl>       <dbl>
## 1          0.742       0.258
```

```
place_tib %>%
  unnest(train, train_place) %>%
  count(train_place) %>%
  mutate(prop = n / sum(n))
```

```
## # A tibble: 2 x 3
##   train_place     n  prop
##   <chr>       <int> <dbl>
## 1 0           68481 0.693
## 2 1           30380 0.307
```

```
place_tib <- glm_fits %>%
  mutate(test_prob = map2(place_mod, test, predict, type = "response"),
         test_place = map(test_prob, ~ if_else(.x > 0.28, "1", "0")))

place_tib %>%
  unnest(test, test_place) %>%
  mutate(correct = if_else(test_place == place, 1, 0)) %>%
  summarise(test_accuracy = mean(correct),
            test_error    = 1 - test_accuracy)
```

```
## # A tibble: 1 x 2
##   test_accuracy test_error
##           <dbl>      <dbl>
## 1         0.746      0.254
```

```
place_tib %>%
  unnest(test, test_place) %>%
  count(test_place) %>%
  mutate(prop = n / sum(n))
```

```
## # A tibble: 2 x 3
##   test_place     n  prop
##   <chr>      <int> <dbl>
## 1 0          28644 0.694
## 2 1          12644 0.306
```

We see that our place model is accurate 74.6% of the time. However, this test accuracy may be inflated by the true negative predictions on horses with very low odds to win. But, our model is predicting a place 30.6% of the time, which seems to be a repectable clip.

Upon dissecting these results, I realized that our model may not be accomplishing what I want it to be accomplishing. Oour model is estimating probabilities that a horse places, and then predicting that it places if the probability is greater than 0.28. This is a bad betting strategy. For example, a horse with 1:1 odds to finish in the top 2 may be rather likely to finish in the top 2. However, we should not be betting on this horse unless the probability of a top 2 finish is greater than 50%, since we will not be profitbale unless this is the case. So I decide to set the threshold to the required payoff of a horse with given odds to make us profitable in the long run. I then move from test_accuracy to returns, which measure our average profit per bet we make, as a function of the odds of the horse and it’s respective payoff if it finishes in the top 2.

```r
returns_log <- function(data, probs){

  tib <- probs %>% as.data.frame() %>% as_tibble()

  names(tib)[1] <- "pred_prob"

  tib <- tib %>%
    mutate(pred_place = case_when(data$mean_final_odds < 1 ~
                                    if_else(pred_prob > 2*data$mean_final_odds, 1,
0),
                                  data$mean_final_odds >= 1 ~
                                    if_else(pred_prob > 2*1/data$mean_final_odds, 1
, 0)),
           potential_payoff = if_else(data$mean_final_odds < 1, data$mean_final_odds/
2,
                                      (data$mean_final_odds/2 - 1)),
           result = data$place,
           real_payoff = if_else(result == 1 & pred_place == 1, potential_payoff,
                                 ifelse(result == 0 & pred_place == 1, -1, 0)))


  mean(tib$real_payoff)
}

returns_log_2 <- function(data, probs){

  tib <- probs %>% as.data.frame() %>% as_tibble()

  names(tib)[1] <- "pred_prob"

  tib <- tib %>%
    mutate(pred_place = case_when(data$mean_final_odds < 1 ~
                                  if_else(pred_prob > 1.5*2*data$mean_final_odds, 1
, 0),
                                  data$mean_final_odds >= 1 ~
                                  if_else(pred_prob > 1.5*2*1/data$mean_final_odds,
1, 0)),
           potential_payoff = if_else(data$mean_final_odds < 1, data$mean_final_odds/
2,
                                      (data$mean_final_odds/2 - 1)),
           result = data$place,
           real_payoff = if_else(result == 1 & pred_place == 1, potential_payoff,
                                 ifelse(result == 0 & pred_place == 1, -1, 0)))


  mean(tib$real_payoff)
}
```

```
returns_log(test_dat, place_tib$test_prob)
```

```
## [1] 0.005404436
```

Our model gives us returns of 0.54%. We will see what happens when we increase the threshold for betting by a factor of 1.5.

```
returns_log_2(test_dat, place_tib$test_prob)
```

```
## [1] 0.0200809
```

We obtain returns of roughly 2.0%. This is rather impressive, as all of the information in our model is gleaned before the race, and 2.0% in the long run with high volume can be a substantial return.

LDA

We will run LDA and QDA models using the regressions with `place` as the response variable, since placing is the best balance between payoff and predictability.

```
library(MASS)
```

```
##
## Attaching package: 'MASS'
```

```
## The following object is masked from 'package:dplyr':
##
##     select
```

```
lda_fits <- data_log_db %>%
  mutate(lda_place_mod = map(train, ~ lda(formula = place ~ mean_final_odds + prize_m
oney_per_run_share +
                                    overall_starts + days_since_last_run + win_rate
+ place_rate,
                                   data=.x)),
         lda_place_mod_no_odds = map(train, ~ lda(formula = place ~ prize_money_per_r
un_share +
                                          overall_starts + days_since_last_run + wi
n_rate + place_rate,
                                          data=.x))) %>%
  gather(key = model_name, value = model_fit, contains("lda_"))
```

We create the models using the same set of predictors from the logistic fits.

```
pred_error_lda_qda <- function(data, model, threshold = 0.5){

  pred_prob <- predict(model, data) %>%
    pluck("posterior") %>%
    as.data.frame() %>%
    as_tibble() %>%
    dplyr::select(`1`)

  pred_place <- if_else(pred_prob > threshold, 1, 0)

  mean(pred_place != data$place)
}

lda_fits <- lda_fits %>%
  mutate(test_error  = map2_dbl(test, model_fit, pred_error_lda_qda, threshold = 0.28
))

lda_fits
```

```
## # A tibble: 2 x 5
##   train                 test                 model_name          model_fit test_error
##   <list>                <list>               <chr>               <list>         <dbl>
## 1 <tibble [98,861 … <tibble [41,288… lda_place_mod       <S3: lda>      0.224
## 2 <tibble [98,861 … <tibble [41,288… lda_place_mod_no… <S3: lda>      0.225
```

We create a function to measure the error of our LDA models, and find the test errors to be 22.4% and 22.5%.

```
returns_lda_qda <- function(data, model){

  pred_prob <- predict(model, data) %>%
    pluck("posterior") %>%
    as.data.frame() %>%
    as_tibble() %>%
    dplyr::select(`1`)

  pred_place <-
    case_when(data$mean_final_odds < 1 ~
                if_else(pred_prob > 2*data$mean_final_odds, 1, 0),
              data$mean_final_odds >= 1 ~
                if_else(pred_prob > 2*1/data$mean_final_odds, 1, 0))

  payoff_vector <- (data$mean_final_odds/2 -1)

  results <- data$place
```

```
  analysis_matrix <- cbind(pred_prob, pred_place, payoff_vector, results)

  analysis_matrix <- analysis_matrix %>%
    mutate(payoff = if_else(results == 1 & pred_place == 1, payoff_vector,
                            ifelse(results == 0 & pred_place == 1, -1, 0)))

  analysis_matrix

  mean(analysis_matrix$payoff)
}

returns_lda_qda_2 <- function(data, model){

  pred_prob <- predict(model, data) %>%
    pluck("posterior") %>%
    as.data.frame() %>%
    as_tibble() %>%
    dplyr::select(`1`)

  pred_place <-
    case_when(data$mean_final_odds < 1 ~
                if_else(pred_prob > 1.5*2*data$mean_final_odds, 1, 0),
              data$mean_final_odds >= 1 ~
                if_else(pred_prob > 1.5*2*1/data$mean_final_odds, 1, 0))

  payoff_vector <- (data$mean_final_odds/2 -1)

  results <- data$place

  analysis_matrix <- cbind(pred_prob, pred_place, payoff_vector, results)

  analysis_matrix <- analysis_matrix %>%
    mutate(payoff = if_else(results == 1 & pred_place == 1, payoff_vector,
                            ifelse(results == 0 & pred_place == 1, -1, 0)))

  analysis_matrix

  mean(analysis_matrix$payoff)
}
```

We create a similar function to measure to returns acheived by our LDA model.

```
returns_lda_qda_2(test_dat, lda_fits %>%
                  pluck("model_fit", 1))
```

```
## [1] -0.01346929
```

```
returns_lda_qda(test_dat, lda_fits %>%
              pluck("model_fit", 1))
```

```
## [1] -0.02937544
```

Unfortunately, our LDA model gives us negative returns. We earn -2.9% using the original threshold calculation, and -1.3% if we artificially increase the relative threshold.

QDA

```
qda_fits <- data_log_db %>%
  mutate(qda_place_mod = map(train, ~ qda(formula = place ~ mean_final_odds + prize_m
oney_per_run_share +
                                            overall_starts + days_since_last_run + wi
n_rate + place_rate,
                                          data=.x)),
         qda_place_mod_no_odds = map(train, ~ qda(formula = place ~ prize_money_per_r
un_share +
                                                    overall_starts + days_since_last_
run + win_rate + place_rate,
                                                  data=.x))) %>%
  gather(key = model_name, value = model_fit, contains("qda_"))


qda_fits <- qda_fits %>%
  mutate(test_error  = map2_dbl(test, model_fit, pred_error_lda_qda, threshold = 0.28
))

qda_fits
```

```
## # A tibble: 2 x 5
##   train                 test                  model_name         model_fit test_error
##   <list>                <list>                <chr>              <list>         <dbl>
## 1 <tibble [98,861 … <tibble [41,288… qda_place_mod      <S3: qda>      0.384
## 2 <tibble [98,861 … <tibble [41,288… qda_place_mod_no… <S3: qda>      0.236
```

Our QDA models end up with a test error of 38.4% and 23.6%.

```
returns_lda_qda(test_dat, qda_fits %>%
              pluck("model_fit", 1))
```

```
## [1] -0.0253363
```

```
returns_lda_qda_2(test_dat, qda_fits %>%
              pluck("model_fit", 1))
```

```
## [1] -0.002144166
```

Our QDA model earns us a return of -2.5% using the original threshold, and -0.2% using the modified threshold.

The logistic model clearly earns the best returns, and the only positive returns, from betting on horses based on their potential payoff and their probability of placing.