```r
library(tidyverse)
```

```
## -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
## v dplyr     1.1.4     v readr     2.1.5
## v forcats   1.0.0     v stringr   1.5.1
## v ggplot2   3.5.1     v tibble    3.2.1
## v lubridate 1.9.4     v tidyr     1.3.1
## v purrr     1.0.4
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become errors
```

```r
library(PlayerRatings)
library(BradleyTerry2)
library(dplyr)
library(tidyr)
library(ggplot2)
```

```r
matches <- read.csv("atp_matches_with_features.csv")

# remove matches with carpet surface
matches <- matches %>% filter(surface != "Carpet")

# sort by date
matches$tourney_date <- as.Date(matches$tourney_date)
matches <- matches[order(matches$tourney_date), ]

# identify all winner rolling columns
w_rolling_cols <- grep("^w_rolling_10_", colnames(matches), value = TRUE)

# create difference columns dynamically
for (col in w_rolling_cols) {
  base_stat <- sub("^w_rolling_10_", "", col)
  l_col <- paste0("l_rolling_10_", base_stat)

  if (l_col %in% colnames(matches)) {
    diff_col <- paste0("diff_rolling_10_", base_stat)
    matches[[diff_col]] <- matches[[col]] - matches[[l_col]]
  }
}

# diff feature
matches$diff_h2h = matches$w_previous_wins - matches$l_previous_wins

# win loss for BT
matches$win  <- 1L
matches$loss <- 0L
```

```r
### ADD ELO ###

# 0. Make sure your full `matches` is sorted by date
```

```r
matches <- matches[order(matches$tourney_date), ]

# 1. Initialize
initial_rating  <- 1500
k_factor        <- 20
all_players     <- unique(c(matches$winner_name, matches$loser_name))
# start everybody at 1500
current_ratings <- setNames(
  rep(initial_rating, length(all_players)),
  all_players
)

# 2. Pre-allocate three new columns
matches$elo_w     <- NA_real_   # winner's rating *before* the match
matches$elo_l     <- NA_real_   # loser's rating  *before* the match
matches$elo_prob  <- NA_real_   # predicted win prob from Elo
matches$diff_elo  <- NA_real_   # to enrich BT


# 3. helper for win-prob
calculate_win_prob <- function(diff) 1 / (1 + 10^(-diff/400))

# 4. loop through every match in chronological order
for (i in seq_len(nrow(matches))) {
  w <- matches$winner_name[i]
  l <- matches$loser_name[i]

  # 4a. get pre-match ratings (default 1500 if brand-new)
  r_w <- current_ratings[w]
  r_l <- current_ratings[l]

  # 4b. record them
  matches$elo_w[i]    <- r_w
  matches$elo_l[i]    <- r_l

  # 4c. compute predicted win-prob
  p_win              <- calculate_win_prob(r_w - r_l)
  matches$elo_prob[i] <- p_win
  matches$diff_elo[i] <- r_w - r_l

  # 4d. update with outcome
  current_ratings[w] <- r_w + k_factor * (1 - p_win)
  current_ratings[l] <- r_l + k_factor * (0 - (1 - p_win))
}
```

```r
### TRAN TEST SPLIT ###
matches$season <- as.numeric(format(matches$tourney_date, "%Y"))
train_matches <- matches %>% filter(season < 2024) %>% filter(season >= 2019)
test_matches  <- matches %>% filter(season == 2024)
cat("Training set matches:", nrow(train_matches), "\n")
```

```
## Training set matches: 11606
```

```r
cat("Testing set matches:", nrow(test_matches), "\n")
```

```
## Testing set matches: 2839
```

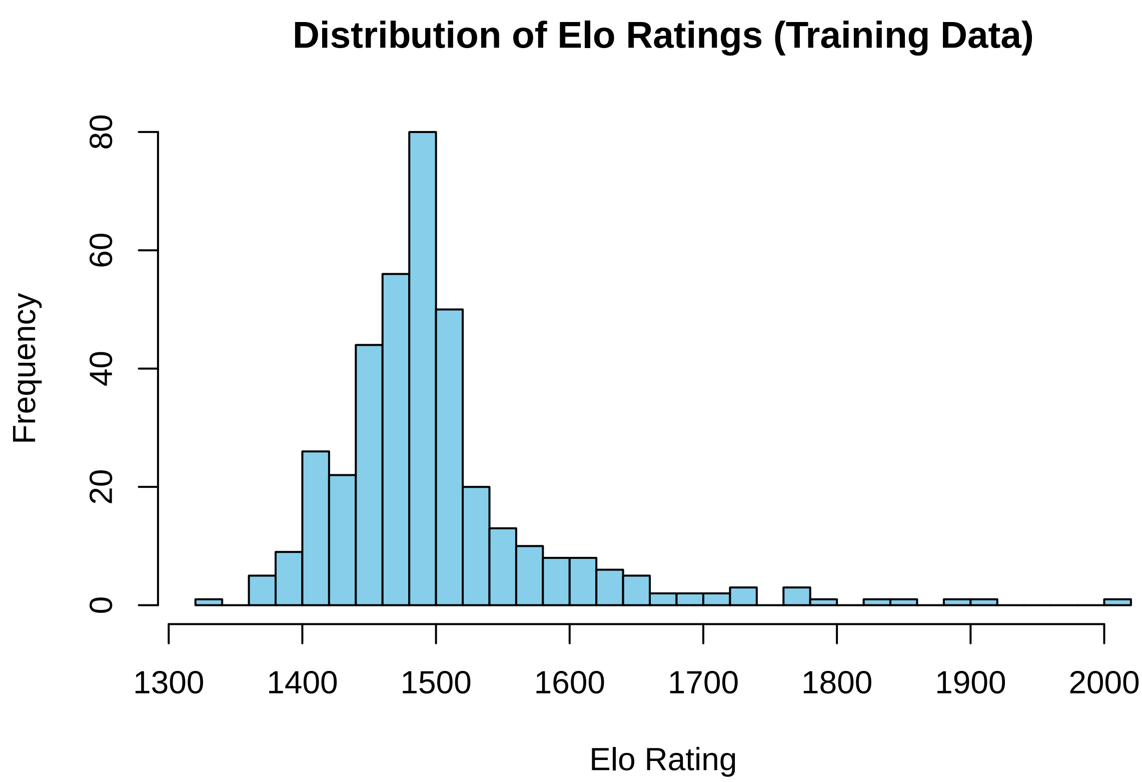
```r
# filter and drop anyone who has only won or only lost in dataset
filter_pure_players <- function(df) {
  wins <- df %>%
    count(player = winner_name, name = "n_wins")

  losses <- df %>%
    count(player = loser_name,  name = "n_losses")

  counts <- full_join(wins, losses, by = "player") %>%
    replace_na(list(n_wins = 0, n_losses = 0)) %>%
    mutate(n_total = n_wins + n_losses)

  bad_players <- counts %>%
    filter(n_wins == 0 | n_losses == 0) %>%
    filter(n_total <= 10) %>%
    pull(player)

  df %>%
    filter(
      !as.character(winner_name) %in% bad_players,
      !as.character(loser_name)  %in% bad_players
    )
}

train_matches <- filter_pure_players(train_matches)
test_matches <- filter_pure_players(test_matches)
```

```r
### ELO ###

# elo
train_games <- data.frame(
  Date  = as.numeric(format(train_matches$tourney_date, "%Y%m%d")),
  Team1 = train_matches$winner_name,
  Team2 = train_matches$loser_name,
  Score = 1,
  stringsAsFactors = FALSE
)
elo_model_train <- elo(train_games, init = 1500, kfac = 20, gamma = 0)

# top 10 players
elo_ratings_train <- elo_model_train$ratings
elo_ratings_train <- elo_ratings_train[order(elo_ratings_train$Rating, decreasing = TRUE), ]
top10_elo_train <- head(elo_ratings_train, 10)
print(top10_elo_train)
```

```
##               Player   Rating Games Win Draw Loss Lag
## 1     Novak Djokovic 2017.002   267 234    0   33   2
## 2      Jannik Sinner 1916.464   255 185    0   70   1
```

```
## 3     Carlos Alcaraz 1894.688   192 150    0   42   7
## 4    Daniil Medvedev 1849.723   326 247    0   79   7
## 5       Rafael Nadal 1820.167   175 145    0   30  47
## 6   Alexander Zverev 1795.039   289 203    0   86   7
## 7      Andrey Rublev 1769.745   327 234    0   93   7
## 8    Grigor Dimitrov 1765.806   216 124    0   92   9
## 9       Nick Kyrgios 1760.050   101  69    0   32  32
## 10    Hubert Hurkacz 1736.541   258 157    0  101   7
```

```r
# plot distribution of Elo ratings based on training data
hist(elo_ratings_train$Rating,
     main = "Distribution of Elo Ratings (Training Data)",
     xlab = "Elo Rating",
     col = "skyblue",
     breaks = 30)
```




```r
# evaluation metrics
correct_prediction <- test_matches$elo_prob > 0.5
accuracy <- mean(correct_prediction, na.rm = TRUE)
misclassification_rate <- 1 - accuracy
cat("Misclassification Rate:", misclassification_rate, "\n")
```

```
## Misclassification Rate: 0.3690078
```

```r
brier_score <- (1 - test_matches$elo_prob)^2
mean_brier_score <- mean(brier_score, na.rm = TRUE)
cat("Mean Brier Score:", mean_brier_score, "\n")
```

```
## Mean Brier Score: 0.2204229
```

```r
# look at elo's biggest fails
worst_elo <- matches %>%
  # only consider rows where we actually computed elo_prob
  filter(!is.na(elo_prob)) %>%
  # sort by ascending elo_prob
  arrange(elo_prob) %>%
  # take the 10 lowest
  slice_head(n = 10)

print(worst_elo)
```

```
##    tourney_id          tourney_name surface draw_size tourney_level tourney_date
## 1   2016-0410  Monte Carlo Masters    Clay        64             M   2016-04-11
## 2    2017-580       Australian Open    Hard       128             G   2017-01-16
## 3    2014-500                 Halle   Grass        28             A   2014-06-09
## 4   2018-M006 Indian Wells Masters    Hard       128             M   2018-03-05
## 5    2014-540             Wimbledon   Grass       128             G   2014-06-23
## 6   2017-0495                 Dubai    Hard        32             A   2017-02-27
## 7   2024-0404 Indian Wells Masters    Hard       128             M   2024-03-04
## 8    2016-540             Wimbledon   Grass       128             G   2016-06-27
## 9   2016-0495                 Dubai    Hard        32             A   2016-02-22
## 10   2013-505              Santiago    Clay        28             A   2013-02-04
##    match_num winner_id winner_seed winner_entry      winner_name winner_hand
## 1        285    106210          NA                    Jiri Vesely           L
## 2        195    104797          NA                 Denis Istomin           R
## 3         13    104460          NA           WC     Dustin Brown           R
## 4        242    106121          NA            Q      Taro Daniel           R
## 5        120    106401          NA           WC     Nick Kyrgios           R
## 6        291    105539          NA            Q   Evgeny Donskoy           R
## 7        285    208134          NA           LL       Luca Nardi           R
## 8        196    105023          28                   Sam Querrey           R
## 9        297    103852           6               Feliciano Lopez           L
## 10        27    104547          NA              Horacio Zeballos           L
##    winner_ioc loser_id loser_seed loser_entry      loser_name loser_hand
## 1         CZE   104925          1              Novak Djokovic          R
## 2         UZB   104925          2              Novak Djokovic          R
## 3         JAM   104745          1                Rafael Nadal          L
## 4         JPN   104925         10              Novak Djokovic          R
## 5         AUS   104745          2                Rafael Nadal          L
## 6         RUS   103819          3               Roger Federer          R
## 7         ITA   104925          1              Novak Djokovic          R
## 8         USA   104925          1              Novak Djokovic          R
## 9         ESP   104925          1              Novak Djokovic          R
## 10        ARG   104745          1                Rafael Nadal          L
##    loser_ioc                         score best_of round minutes w_ace w_df w_svpt
## 1        SRB               6-4 2-6 6-4       3   R32     126     4    4     76
## 2        SRB 7-6(8) 5-7 2-6 7-6(5) 6-4       5   R64     288    17    3    194
```

```
## 3        ESP                   6-4 6-1       3   R16     60    11    5     51
## 4        SRB            7-6(3) 4-6 6-1       3   R64    150     1    8    107
## 5        ESP     7-6(5) 5-7 7-6(5) 6-3       5   R16    178    37    4    144
## 6        SUI         3-6 7-6(7) 7-6(5)       3   R16    122     6    3     95
## 7        SRB               6-4 3-6 6-3       3   R32    140     6    7     90
## 8        SRB     7-6(6) 6-1 3-6 7-6(5)       5   R32    176    31    2    160
## 9        SRB               6-3 0-0 RET       3    QF     33     2    2     22
## 10       ESP         6-7(2) 7-6(6) 6-4       3     F    167    12    0     96
##    w_1stIn w_1stWon w_2ndWon w_SvGms w_bpSaved w_bpFaced l_ace l_df l_svpt
## 1       45       31       19      14         1         4     3    3     92
## 2      134       94       27      27         9        15    14    9    185
## 3       31       26       12       9         1         1     2    0     47
## 4       52       37       28      15         5         7     4    4     94
## 5       98       81       25      23         2         3    11    3    137
## 6       46       35       29      16         1         5    12    2    109
## 7       64       44       13      14         2         4     4    1     95
## 8      100       79       25      20        14        17     7    2    146
## 9       10        9        8       4         0         0     1    1     29
## 10      58       45       27      17         2         3     4    1    106
##    l_1stIn l_1stWon l_2ndWon l_SvGms l_bpSaved l_bpFaced winner_rank_points
## 1       61       44       14      14         5         8                840
## 2      107       77       43      27        10        14                498
## 3       32       22        6       8         6         9                633
## 4       60       39       18      14         8        11                539
## 5      100       77       22      22         3         4                400
## 6       80       59       15      17         1         4                493
## 7       57       44       16      14         8        11                513
## 8       97       63       27      20        11        15               1075
## 9       16       11        6       5         4         6               1495
## 10      77       57       18      17         5         7                708
##    loser_rank_points w_rank l_rank w_ht l_ht w_age l_age
## 1              16540     55      1  198  188  22.7  28.8
## 2              11780    117      2  185  188  30.3  29.6
## 3              12500     85      1  196  185  29.5  28.0
## 4               2380    109     13  191  188  25.1  30.7
## 5              12500    144      1  193  185  19.1  28.0
## 6               3260    116     10  185  185  26.8  35.5
## 7               9675    123      1  185  188  20.5  36.7
## 8              16950     41      1  198  188  28.7  29.1
## 9              16790     24      1  188  188  34.4  28.7
## 10              5400     73      5  188  185  27.7  26.6
##    w_rolling_10_ace_per_svgm w_rolling_10_df_per_svgm
## 1                  0.4511491                0.2078711
## 2                  0.5833922                0.1660722
## 3                  0.6999999                0.3799950
## 4                  0.2976221                0.4324123
## 5                  0.8449815                0.2864957
## 6                  0.3989322                0.1825722
## 7                  0.2319444                0.1845635
## 8                  1.1158604                0.1560408
## 9                  1.0280240                0.3345285
## 10                 0.4224870                0.1723922
##    w_rolling_10_svpt_per_svgm w_rolling_10_1stIn_per_svgm
## 1                    6.660308                    4.318270
```

```
## 2                  6.863417                  4.349540
## 3                  6.493274                  4.030144
## 4                  6.852013                  3.966637
## 5                  6.696712                  4.315922
## 6                  6.757023                  4.066698
## 7                  6.253968                  4.262897
## 8                  5.920988                  3.569351
## 9                  6.394547                  3.462069
## 10                 6.441168                  3.999141
##    w_rolling_10_1stWon_per_svgm w_rolling_10_2ndWon_per_svgm w_rolling_10_SvGms
## 1                      2.982642                    1.1922912               12.4
## 2                      2.997867                    1.0893146               15.7
## 3                      3.065017                    1.1948563               12.0
## 4                      2.579143                    1.2825278               14.9
## 5                      3.153023                    1.1751100               20.0
## 6                      2.802148                    1.3225900               14.0
## 7                      2.752540                    0.8684921               11.7
## 8                      3.053470                    1.2897872               14.5
## 9                      2.754359                    1.5197981               15.1
## 10                     2.750759                    1.2496695               12.3
##    w_rolling_10_bpSaved_per_svgm w_rolling_10_bpSaved_per_bpFaced_per_svgm
## 1                      0.4206411                                0.05825708
## 2                      0.4697678                                0.06386140
## 3                      0.2225720                                0.05620622
## 4                      0.4987574                                0.04407812
## 5                      0.3130449                                0.03015992
## 6                      0.4700900                                0.04824335
## 7                      0.2821429                                0.04087693
## 8                      0.1862734                                0.04731727
## 9                      0.2373069                                0.04352061
## 10                     0.3053836                                0.04631534
##    w_rolling_10_bpFaced_per_svgm w_rolling_10_rank w_rolling_10_ht
## 1                      0.6420977              54.7             198
## 2                      0.7603549             107.8             185
## 3                      0.4000875              90.1             196
## 4                      0.8251344             108.2             191
## 5                      0.4938304             174.7             193
## 6                      0.7230115             107.6             185
## 7                      0.6248413             135.4             185
## 8                      0.2471916              37.3             198
## 9                      0.3702483              21.5             188
## 10                     0.5366593              90.6             188
##    w_rolling_10_age l_rolling_10_ace_per_svgm l_rolling_10_df_per_svgm
## 1             22.65                0.34693001               0.26968975
## 2             30.01                0.24847375               0.18970238
## 3             29.37                0.08620614               0.08698878
## 4             24.86                0.38599512               0.23354701
## 5             18.78                0.21323880               0.06707251
## 6             26.54                0.76088593               0.12058827
## 7             20.05                0.63787879               0.13021645
## 8             28.66                0.22706855               0.16498849
## 9             34.33                0.49201034               0.13419279
## 10            27.46                0.25666616               0.12407466
##    l_rolling_10_svpt_per_svgm l_rolling_10_1stIn_per_svgm
```

```
## 1                  6.491068                  4.234091
## 2                  6.376813                  4.222031
## 3                  6.050694                  4.278675
## 4                  6.153907                  4.198639
## 5                  5.729080                  3.930226
## 6                  5.914489                  3.781874
## 7                  6.087828                  3.863506
## 8                  6.297842                  4.318990
## 9                  5.879572                  3.756680
## 10                 6.097670                  4.094065
##    l_rolling_10_1stWon_per_svgm l_rolling_10_2ndWon_per_svgm l_rolling_10_SvGms
## 1                      3.083806                     1.259513                9.6
## 2                      3.162063                     1.151723               11.6
## 3                      3.006298                     1.066767               13.3
## 4                      3.150263                     1.076288               12.7
## 5                      2.975039                     1.111718               14.4
## 6                      2.988445                     1.193301               19.4
## 7                      2.963752                     1.232626               15.1
## 8                      3.015420                     1.138755               15.6
## 9                      2.853129                     1.325123               13.8
## 10                     3.086653                     1.210515               12.8
##    l_rolling_10_bpSaved_per_svgm l_rolling_10_bpSaved_per_bpFaced_per_svgm
## 1                      0.2527525                                0.04460017
## 2                      0.2712042                                0.06716106
## 3                      0.3279863                                0.06085844
## 4                      0.1688462                                0.04599953
## 5                      0.2789513                                0.05712679
## 6                      0.1885218                                0.02854249
## 7                      0.2391775                                0.04660534
## 8                      0.3065307                                0.03873206
## 9                      0.2631004                                0.06151036
## 10                     0.3097921                                0.06417177
##    l_rolling_10_bpFaced_per_svgm l_rolling_10_rank l_rolling_10_ht
## 1                      0.3746212               1.0             188
## 2                      0.3991010               2.0             188
## 3                      0.4691441               1.0             185
## 4                      0.2641575               8.0             188
## 5                      0.3880318               1.0             185
## 6                      0.2938705              13.5             185
## 7                      0.3626696               1.0             188
## 8                      0.4592503               1.0             188
## 9                      0.3654388               1.0             188
## 10                     0.3981124               2.9             185
##    l_rolling_10_age w_previous_wins l_previous_wins
## 1             28.76               0               0
## 2             29.52               0               5
## 3             27.90               0               0
## 4             30.29               0               0
## 5             27.94               0               0
## 6             35.29               0               0
## 7             36.61               0               0
## 8             29.01               1               6
## 9             28.62               0               7
## 10            26.15               0               1
```

```
##    diff_rolling_10_ace_per_svgm diff_rolling_10_df_per_svgm
## 1                     0.1042191                -0.061818606
## 2                     0.3349185                -0.023630151
## 3                     0.6137938                 0.293006248
## 4                    -0.0883730                 0.198865341
## 5                     0.6317427                 0.219423221
## 6                    -0.3619537                 0.061983885
## 7                    -0.4059343                 0.054347042
## 8                     0.8887918                -0.008947725
## 9                     0.5360136                 0.200335699
## 10                    0.1658209                 0.048317531
##    diff_rolling_10_svpt_per_svgm diff_rolling_10_1stIn_per_svgm
## 1                      0.1692403                     0.08417876
## 2                      0.4866037                     0.12750809
## 3                      0.4425795                    -0.24853144
## 4                      0.6981059                    -0.23200129
## 5                      0.9676317                     0.38569576
## 6                      0.8425337                     0.28482441
## 7                      0.1661400                     0.39939033
## 8                     -0.3768531                    -0.74963969
## 9                      0.5149756                    -0.29461065
## 10                     0.3434972                    -0.09492438
##    diff_rolling_10_1stWon_per_svgm diff_rolling_10_2ndWon_per_svgm
## 1                      -0.10116395                     -0.06722180
## 2                      -0.16419622                     -0.06240850
## 3                       0.05871903                      0.12808939
## 4                      -0.57111932                      0.20623962
## 5                       0.17798370                      0.06339159
## 6                      -0.18629632                      0.12928936
## 7                      -0.21121212                     -0.36413420
## 8                       0.03805058                      0.15103180
## 9                      -0.09876932                      0.19467527
## 10                     -0.33589437                      0.03915466
##    diff_rolling_10_SvGms diff_rolling_10_bpSaved_per_svgm
## 1                    2.8                       0.16788855
## 2                    4.1                       0.19856364
## 3                   -1.3                      -0.10541433
## 4                    2.2                       0.32991124
## 5                    5.6                       0.03409359
## 6                   -5.4                       0.28156816
## 7                   -3.4                       0.04296537
## 8                   -1.1                      -0.12025727
## 9                    1.3                      -0.02579346
## 10                  -0.5                      -0.00440852
##    diff_rolling_10_bpSaved_per_bpFaced_per_svgm
## 1                                   0.013656907
## 2                                  -0.003299658
## 3                                  -0.004652223
## 4                                  -0.001921409
## 5                                  -0.026966869
## 6                                   0.019700855
## 7                                  -0.005728407
## 8                                   0.008585212
## 9                                  -0.017989751
```

```
## 10                                 -0.017856426
##    diff_rolling_10_bpFaced_per_svgm diff_rolling_10_rank diff_rolling_10_ht
## 1                        0.26747645                 53.7                 10
## 2                        0.36125381                105.8                 -3
## 3                       -0.06905666                 89.1                 11
## 4                        0.56097688                100.2                  3
## 5                        0.10579867                173.7                  8
## 6                        0.42914101                 94.1                  0
## 7                        0.26217172                134.4                 -3
## 8                       -0.21205876                 36.3                 10
## 9                        0.00480957                 20.5                  0
## 10                       0.13854693                 87.7                  3
##    diff_rolling_10_age diff_h2h win loss    elo_w    elo_l   elo_prob  diff_elo
## 1                -6.11        0   1    0 1541.692 2289.451 0.01332828 -747.7595
## 2                 0.49       -5   1    0 1551.530 2221.740 0.02067291 -670.2104
## 3                 1.47        0   1    0 1527.063 2181.300 0.02261885 -654.2374
## 4                -5.43        0   1    0 1459.490 2108.049 0.02335290 -648.5587
## 5                -9.16        0   1    0 1515.022 2162.449 0.02350193 -647.4271
## 6                -8.75        0   1    0 1449.363 2095.363 0.02369118 -646.0002
## 7               -16.56        0   1    0 1460.460 2102.475 0.02422768 -642.0146
## 8                -0.35       -5   1    0 1664.010 2284.417 0.02734879 -620.4076
## 9                 5.71       -7   1    0 1684.826 2291.480 0.02953551 -606.6541
## 10                1.31       -1   1    0 1491.265 2097.051 0.02967899 -605.7865
##    season
## 1    2016
## 2    2017
## 3    2014
## 4    2018
## 5    2014
## 6    2017
## 7    2024
## 8    2016
## 9    2016
## 10   2013
```

```r
extract_diff_coefficients <- function(model, pattern = "^diff", sort_by = "p.value", sort = TRUE) {
  coef_table <- summary(model)$coefficients
  matching_rows <- grep(pattern, rownames(coef_table))
    results <- data.frame(
    term      = rownames(coef_table)[matching_rows],
    estimate  = coef_table[matching_rows, "Estimate"],
    p.value   = coef_table[matching_rows, "Pr(>|z|)"],
    row.names = NULL
  )
  if (sort) {
    results <- results[order(results[[sort_by]], decreasing = FALSE), ]
  }
  results
}

evaluate_glm_model <- function(model, test_data, newdata, valid_idx, prediction_col = "pred_prob", thres
  preds <- predict(
    model,
    newdata = newdata,
```

```r
    type = "response"
  )

  test_data[[prediction_col]] <- NA_real_
  test_data[[prediction_col]][valid_idx] <- preds

  correct_prediction <- test_data[[prediction_col]] > threshold
  accuracy <- mean(correct_prediction, na.rm = TRUE)
  misclassification_rate <- 1 - accuracy

  brier_score <- (1 - test_data[[prediction_col]])^2
  mean_brier_score <- mean(brier_score, na.rm = TRUE)

  cat("Evaluation Results:\n")
  cat("-------------------\n")
  cat("Misclassification Rate:", round(misclassification_rate, 4), "\n")
  cat("Mean Brier Score:", round(mean_brier_score, 4), "\n")

  # Return results as a list
  results <- list(
    predictions = test_data[[prediction_col]],
    accuracy = accuracy,
    misclassification_rate = misclassification_rate,
    mean_brier_score = mean_brier_score,
    test_data = test_data
  )

  return(results)
}
```

```r
# train levels
all_players <- (unique(c(train_matches$winner_name,
                             train_matches$loser_name)))

train_matches$winner_name = factor(train_matches$winner_name, levels = all_players)
train_matches$loser_name = factor(train_matches$loser_name, levels = all_players)

# re-level the test set to exactly those levels used in train
test_matches$winner_name = factor(test_matches$winner_name, levels = all_players)
test_matches$loser_name = factor(test_matches$loser_name, levels = all_players)

# drop any rows that now have NA (i.e. brand-new players unseen in train)
valid_idx <- which(
  !is.na(test_matches$winner_name) &
  !is.na(test_matches$loser_name)
)

test_valid <- test_matches[valid_idx, ]
nrow(test_matches)
```

```
## [1] 2691
```

```r
nrow(test_valid)
```

```
## [1] 2459
```

```r
#### VANILLA BT ###
X <- model.matrix(~ train_matches$winner_name - 1) - model.matrix(~ train_matches$loser_name - 1)

model <- glm(train_matches$win ~ X[, -1] - 1, family = binomial, data = train_matches)
```

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

```r
ratings <- coef(model)
ratings_df <- data.frame(
  player = names(ratings),
  rating = ratings
)
ratings_df$player <- sub("^.*winner_name", "", ratings_df$player)

sorted_ratings_df <- ratings_df[order(ratings_df$rating, decreasing = TRUE), ]
top10 <- sorted_ratings_df[1:10, ]

# Print the top 10 rated players.
cat("Top 10 Rated Players (Higher rating = stronger):\n")
```

```
## Top 10 Rated Players (Higher rating = stronger):
```

```r
print(top10)
```

```
##                                                         player    rating
## X[, -1]train_matches$winner_nameNick Hardt              Nick Hardt 13.913203
## X[, -1]train_matches$winner_nameNovak Djokovic      Novak Djokovic  2.656996
## X[, -1]train_matches$winner_nameRafael Nadal          Rafael Nadal  2.248558
## X[, -1]train_matches$winner_nameRoger Federer        Roger Federer  1.950682
## X[, -1]train_matches$winner_nameCarlos Alcaraz      Carlos Alcaraz  1.800364
## X[, -1]train_matches$winner_nameDaniil Medvedev    Daniil Medvedev  1.720706
## X[, -1]train_matches$winner_nameFranco Agamenone  Franco Agamenone  1.445257
## X[, -1]train_matches$winner_nameJannik Sinner        Jannik Sinner  1.426718
## X[, -1]train_matches$winner_nameStefanos Tsitsipas Stefanos Tsitsipas  1.401950
## X[, -1]train_matches$winner_nameAlexander Zverev  Alexander Zverev  1.377390
```

```r
# test
predict_probs <- function(tourney_df, ratings_df) {
  tourney_df %>%
    left_join(ratings_df, by = c("winner_name" = "player")) %>%
    rename(W_rating = rating) %>%
    left_join(ratings_df, by = c("loser_name" = "player")) %>%
    rename(L_rating = rating) %>%
    mutate(
      pred_prob = exp(W_rating) / (exp(W_rating) + exp(L_rating))
    )
```

```r
}

pred_prob <- predict_probs(test_matches, ratings_df)$pred_prob

# evaluation metrics
correct_prediction <- pred_prob > 0.5
accuracy <- mean(correct_prediction, na.rm = TRUE)
misclassification_rate <- 1 - accuracy
cat("Misclassification Rate:", misclassification_rate, "\n")
```

```
## Misclassification Rate: 0.3926658
```

```r
brier_score <- (1 - pred_prob)^2
mean_brier_score <- mean(brier_score, na.rm = TRUE)
cat("Mean Brier Score:", mean_brier_score, "\n")
```

```
## Mean Brier Score: 0.231361
```

```r
### BT WITH COVARIATES ####
covariates <- c(
  "diff_rolling_10_ace_per_svgm",
  "diff_rolling_10_df_per_svgm",
  "diff_rolling_10_svpt_per_svgm",
  "diff_rolling_10_1stIn_per_svgm",
  "diff_rolling_10_1stWon_per_svgm",
  "diff_rolling_10_2ndWon_per_svgm",
  "diff_rolling_10_SvGms",
  "diff_rolling_10_bpSaved_per_svgm",
  "diff_rolling_10_bpSaved_per_bpFaced_per_svgm",
  "diff_rolling_10_bpFaced_per_svgm",
  "diff_rolling_10_rank",
  "diff_rolling_10_ht",
  "diff_rolling_10_age",
  "diff_h2h"
)
cov_str <- paste(covariates, collapse = " + ")

model_w_diffs <- glm(
  as.formula(
    paste0(
      "win ~ X[, -1] + ",
      cov_str,
      " - 1"
    )
  ),
  data   = train_matches,
  family = binomial(link = "logit")
)
```

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

```r
diff_coefs <- extract_diff_coefficients(model_w_diffs)
diff_coefs
```

```
##                                             term     estimate      p.value
## 12                          diff_rolling_10_age  1.343533097 6.529092e-51
## 11                         diff_rolling_10_rank  0.001749588 1.910604e-06
## 2                  diff_rolling_10_df_per_svgm -1.322543348 5.586404e-06
## 5                diff_rolling_10_1stWon_per_svgm  0.965788475 9.297474e-03
## 9  diff_rolling_10_bpSaved_per_bpFaced_per_svgm  4.908493101 5.818879e-02
## 10              diff_rolling_10_bpFaced_per_svgm -1.561801088 1.333007e-01
## 6                diff_rolling_10_2ndWon_per_svgm  0.576142635 1.446710e-01
## 3                  diff_rolling_10_svpt_per_svgm -0.275939123 1.870595e-01
## 4                 diff_rolling_10_1stIn_per_svgm -0.237914306 2.105609e-01
## 13                                     diff_h2h  0.018013770 2.894171e-01
## 8               diff_rolling_10_bpSaved_per_svgm  0.993491910 3.878285e-01
## 7                         diff_rolling_10_SvGms  0.010429035 4.835506e-01
## 1                   diff_rolling_10_ace_per_svgm -0.064999450 6.985835e-01
```

```r
# combine into one newdata frame
X <- model.matrix(~ test_valid$winner_name - 1) - model.matrix(~ test_valid$loser_name - 1)
newdata <- cbind(
  as.data.frame(X),
  test_valid[, covariates, drop = FALSE]
)

results <- evaluate_glm_model(
    model = model_w_diffs,
    test_data = test_matches,
    newdata = newdata,
    valid_idx = valid_idx
)
```

```
## Evaluation Results:
## ------------------
## Misclassification Rate: 0.342
## Mean Brier Score: 0.2168
```

```r
### BT WITH COVARIATES REDUCED ####
covariates <- c(
  #"diff_rolling_10_ace_per_svgm",
  "diff_rolling_10_df_per_svgm",
  "diff_rolling_10_svpt_per_svgm",
  #"diff_rolling_10_1stIn_per_svgm",
  "diff_rolling_10_1stWon_per_svgm",
  "diff_rolling_10_2ndWon_per_svgm",
  #"diff_rolling_10_SvGms",
  #"diff_rolling_10_bpSaved_per_svgm",
  "diff_rolling_10_bpSaved_per_bpFaced_per_svgm",
  #"diff_rolling_10_bpFaced_per_svgm",
  "diff_rolling_10_rank",
  "diff_rolling_10_ht",
  "diff_rolling_10_age",
```

```r
  "diff_h2h"
)
cov_str <- paste(covariates, collapse = " + ")

X <- model.matrix(~ train_matches$winner_name - 1) - model.matrix(~ train_matches$loser_name - 1)
model_w_diffs_red <- glm(
  as.formula(
    paste0(
      "win ~ X[, -1] + ",
      cov_str,
      " - 1"
    )
  ),
  data   = train_matches,
  family = binomial(link = "logit")
)
```

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

```r
diff_coefs <- extract_diff_coefficients(model_w_diffs_red, sort=TRUE)
diff_coefs
```

```
##                                          term     estimate      p.value
## 7                        diff_rolling_10_age  1.340788829 7.240926e-51
## 3              diff_rolling_10_1stWon_per_svgm  1.458405970 2.889350e-31
## 2                diff_rolling_10_svpt_per_svgm -0.780156579 1.533684e-25
## 4              diff_rolling_10_2ndWon_per_svgm  1.396246964 1.353100e-18
## 6                         diff_rolling_10_rank  0.001762834 1.585258e-06
## 1                  diff_rolling_10_df_per_svgm -1.170818524 1.140837e-05
## 5 diff_rolling_10_bpSaved_per_bpFaced_per_svgm  2.777288267 1.109569e-01
## 8                                    diff_h2h  0.018105993 2.865224e-01
```

```r
# combine into one newdata frame
X <- model.matrix(~ test_valid$winner_name - 1) - model.matrix(~ test_valid$loser_name - 1)
newdata <- cbind(
  as.data.frame(X),
  test_valid[, covariates, drop = FALSE]
)

results <- evaluate_glm_model(
    model = model_w_diffs_red,
    test_data = test_matches,
    newdata = newdata,
    valid_idx = valid_idx
)
```

```
## Evaluation Results:
## -------------------
## Misclassification Rate: 0.34
## Mean Brier Score: 0.217
```

```r
### BT WITH COVARIATES REDUCED BY SURFACE ####
X <- model.matrix(~ train_matches$winner_name - 1) - model.matrix(~ train_matches$loser_name - 1)
model_w_diffs_surface <- glm(
  as.formula(
    paste0(
      "win ~ X[, -1] + (",
      cov_str,
      ") : surface - 1"
    )
  ),
  data   = train_matches,
  family = binomial(link = "logit")
)
```

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

```r
diff_coefs <- extract_diff_coefficients(model_w_diffs_surface, sort=FALSE)
diff_coefs
```

```
##                                                          term     estimate
## 1                     diff_rolling_10_df_per_svgm:surfaceClay -1.877282046
## 2                    diff_rolling_10_df_per_svgm:surfaceGrass -0.143376301
## 3                     diff_rolling_10_df_per_svgm:surfaceHard -1.031402739
## 4                   diff_rolling_10_svpt_per_svgm:surfaceClay -0.392713263
## 5                  diff_rolling_10_svpt_per_svgm:surfaceGrass -1.728070922
## 6                   diff_rolling_10_svpt_per_svgm:surfaceHard -0.809332382
## 7                 diff_rolling_10_1stWon_per_svgm:surfaceClay  0.681850898
## 8                diff_rolling_10_1stWon_per_svgm:surfaceGrass  3.176587214
## 9                 diff_rolling_10_1stWon_per_svgm:surfaceHard  1.527768419
## 10                diff_rolling_10_2ndWon_per_svgm:surfaceClay  0.599140796
## 11               diff_rolling_10_2ndWon_per_svgm:surfaceGrass  3.137077719
## 12                diff_rolling_10_2ndWon_per_svgm:surfaceHard  1.521025162
## 13  diff_rolling_10_bpSaved_per_bpFaced_per_svgm:surfaceClay  1.030871487
## 14 diff_rolling_10_bpSaved_per_bpFaced_per_svgm:surfaceGrass -8.214851229
## 15  diff_rolling_10_bpSaved_per_bpFaced_per_svgm:surfaceHard  5.465138868
## 16                          diff_rolling_10_rank:surfaceClay  0.001252076
## 17                         diff_rolling_10_rank:surfaceGrass  0.002943580
## 18                          diff_rolling_10_rank:surfaceHard  0.001658732
## 19                            diff_rolling_10_ht:surfaceClay  0.003018123
## 20                           diff_rolling_10_ht:surfaceGrass -0.011178967
## 21                           diff_rolling_10_age:surfaceClay  1.341453092
## 22                          diff_rolling_10_age:surfaceGrass  1.374128007
## 23                           diff_rolling_10_age:surfaceHard  1.347935711
## 24                                      diff_h2h:surfaceClay  0.032464789
## 25                                     diff_h2h:surfaceGrass -0.010039361
## 26                                      diff_h2h:surfaceHard  0.016768835
##          p.value
## 1   7.522918e-07
## 2   7.855126e-01
## 3   6.161771e-04
## 4   9.507736e-04
## 5   2.530505e-15
## 6   2.808953e-18
```

```
## 7  9.027279e-04
## 8  1.844799e-19
## 9  6.307114e-23
## 10 1.221952e-02
## 11 6.063637e-14
## 12 4.767100e-15
## 13 7.385806e-01
## 14 1.322305e-01
## 15 1.229408e-02
## 16 1.739045e-02
## 17 6.235364e-04
## 18 1.400067e-04
## 19 6.245040e-01
## 20 2.362331e-01
## 21 4.158398e-50
## 22 2.769495e-52
## 23 8.094602e-51
## 24 2.912763e-01
## 25 8.504903e-01
## 26 4.254454e-01
```

```r
# combine into one newdata frame
X <- model.matrix(~ test_valid$winner_name - 1) - model.matrix(~ test_valid$loser_name - 1)
newdata <- cbind(
  as.data.frame(model.matrix(~ test_valid$winner_name - 1) - model.matrix(~ test_valid$loser_name - 1))
  test_valid[, covariates, drop = FALSE],
  surface = test_valid$surface
)

results <- evaluate_glm_model(
    model = model_w_diffs_surface,
    test_data = test_matches,
    newdata = newdata,
    valid_idx = valid_idx
)
```

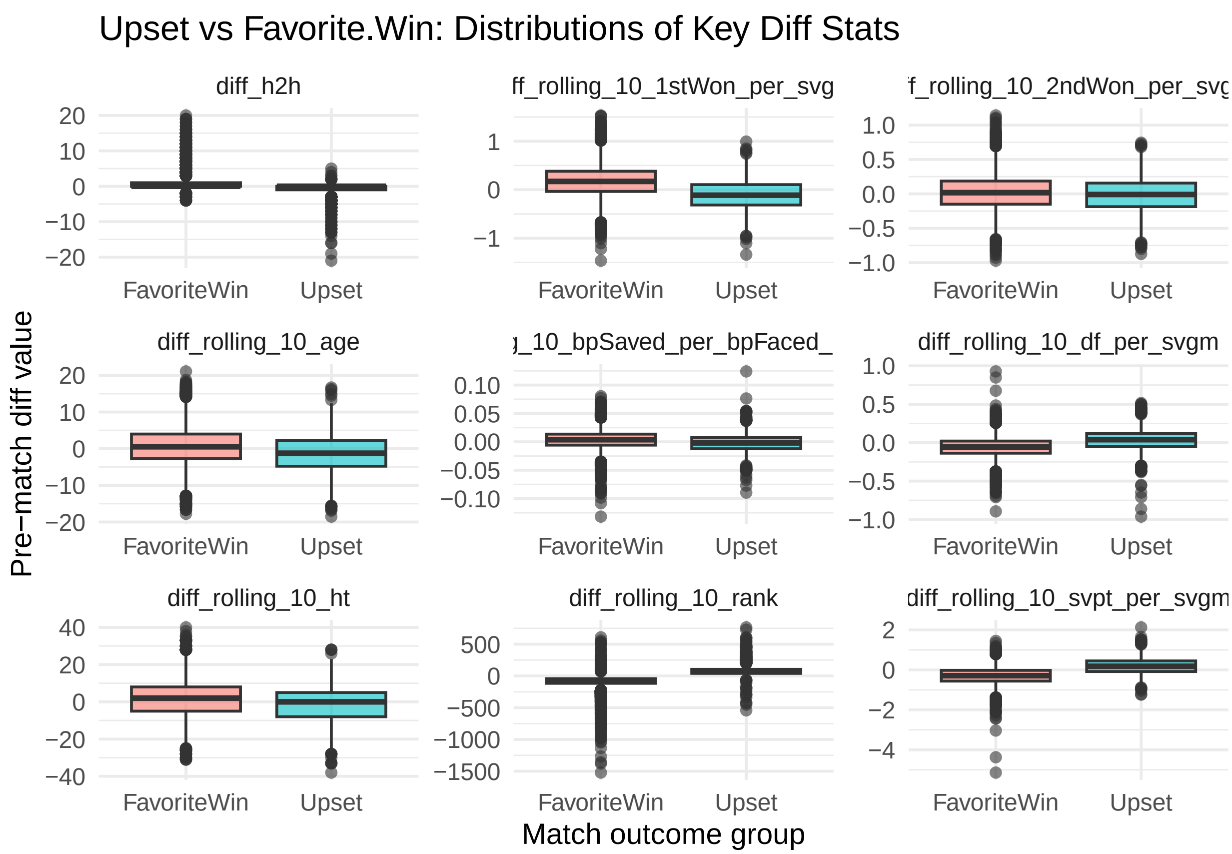
```
## Evaluation Results:
## -------------------
## Misclassification Rate: 0.3449
## Mean Brier Score: 0.2163
```

```r
### EXAMINE STAT DIFFERENCES BETWEEN TOP GROUP VS WORST GROUP  ###
# 1. pick the vars to test
test_vars <- covariates

# 2. label each match as 'Upset' or 'FavoriteWin'
cmp_df <- matches %>%
  filter(!is.na(elo_prob)) %>%
  mutate(
    result_group = case_when(
      elo_prob <  0.2 ~ "Upset",
      elo_prob >  0.8 ~ "FavoriteWin",
      TRUE            ~ NA_character_
```

```r
    )
  ) %>%
  filter(!is.na(result_group))

# 3. run t-tests for each variable
ttests <- map_dfr(test_vars, function(var){
  x <- pull(cmp_df, var)[cmp_df$result_group == "Upset"]
  y <- pull(cmp_df, var)[cmp_df$result_group == "FavoriteWin"]
  t  <- t.test(y, x)
  tibble(
    stat          = var,
    mean_fav      = mean(y, na.rm=TRUE),
    mean_upset    = mean(x, na.rm=TRUE),
    diff_means    = mean(y, na.rm=TRUE) - mean(x, na.rm=TRUE),
    t_statistic   = t$statistic,
    p_value       = t$p.value
  )
}) %>%
  arrange(diff_means)

ttests <- ttests[order(ttests$p_value), ]
ttests
```

```
## # A tibble: 9 x 6
##   stat                        mean_fav mean_upset diff_means t_statistic   p_value
##   <chr>                          <dbl>      <dbl>      <dbl>       <dbl>     <dbl>
## 1 diff_rolling_10_rank      -9.80e+1   83.5     -181.            -80.9   0
## 2 diff_rolling_10_svpt_per~ -2.93e-1    0.188     -0.481         -46.2   0
## 3 diff_h2h                   1.06e+0   -0.924      1.99           38.4   5.44e-256
## 4 diff_rolling_10_1stWon_p~  1.74e-1   -0.109      0.284          35.7   1.92e-226
## 5 diff_rolling_10_df_per_s~ -5.96e-2    0.0336    -0.0933        -26.6   2.47e-137
## 6 diff_rolling_10_bpSaved_~  3.94e-3   -0.00229    0.00624        15.3   1.63e- 50
## 7 diff_rolling_10_age        7.47e-1   -1.30       2.05           15.2   7.57e- 50
## 8 diff_rolling_10_ht         1.65e+0   -1.09       2.74           11.3   6.77e- 29
## 9 diff_rolling_10_2ndWon_p~  2.41e-2   -0.0155     0.0397          6.06 1.52e-   9
```

```r
# 4. visualize distributions side-by-side
cmp_df %>%
  select(result_group, all_of(test_vars)) %>%
  pivot_longer(-result_group, names_to="stat", values_to="value") %>%
  ggplot(aes(x=result_group, y=value, fill=result_group)) +
    geom_boxplot(alpha=0.6) +
    facet_wrap(~stat, scales="free") +
    labs(
      x = "Match outcome group",
      y = "Pre-match diff value",
      title = "Upset vs Favorite-Win: Distributions of Key Diff Stats"
    ) +
    theme_minimal() +
    theme(legend.position="none")
```




```r
### DEPRECATED: TOO MUCH SPARSITY IN PLAYER-SURFACE BT ###
# 1. build the universe of all player×surface combos
all_players  <- sort(unique(c(as.character(train_matches$winner_name),
                             as.character(train_matches$loser_name))))
surfaces <- sort(unique(as.character(train_matches$surface)))

all_surf_lvls <- as.vector(outer(all_players, surfaces,
                                 FUN = function(p, s) paste(p, s, sep = ".")))

train_df <- train_matches %>%
  mutate(
    w_surf = factor(paste(winner_name, surface, sep = "."), levels = all_surf_lvls),
    l_surf = factor(paste(loser_name,  surface, sep = "."), levels = all_surf_lvls)
  )


# 2. create winner×surface and loser×surface factors with those levels
train_df <- train_df %>%
  mutate(
    w_surf = factor(paste(winner_name, surface, sep = "."), levels = all_surf_lvls),
    l_surf = factor(paste(loser_name,  surface, sep = "."), levels = all_surf_lvls)
  )

# 3. now build BT design matrices
Xw_surf   <- model.matrix(~ w_surf - 1, data = train_df)
Xl_surf   <- model.matrix(~ l_surf - 1, data = train_df)
```

```r
X_bt_surf <- Xw_surf - Xl_surf   # same columns in same order—you can subtract

# 4. bind with your diff features and fit
glm_input <- cbind(
  win = train_df$win,
  as.data.frame(X_bt_surf)
)

glm_surf <- glm(win ~ . - 1, data = glm_input, family = binomial)
```

```
## Warning: glm.fit: algorithm did not converge
```

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

```r
# --- 1. Re-level test set to your training factors
test_matches <- test_matches %>%
  mutate(
    winner_name = factor(winner_name, levels = all_players),
    loser_name  = factor(loser_name,  levels = all_players),
    surface     = factor(surface,     levels = surfaces)
  )

# 2. Drop any matches with unseen players
valid_idx <- which(
  !is.na(test_matches$winner_name) &
  !is.na(test_matches$loser_name)  &
  !is.na(test_matches$surface)      # <- drop matches on unseen surfaces
)
test_valid <- test_matches[valid_idx, ]

# 3. Recreate the player×surface interaction on test
test_valid <- test_valid %>%
  mutate(
    w_surf = factor(paste(winner_name, surface, sep="."), levels = all_surf_lvls),
    l_surf = factor(paste(loser_name,  surface, sep="."), levels = all_surf_lvls)
  )

# 4. Build the BT design matrix for player×surface
Xw_surf_test <- model.matrix(~ w_surf - 1, data = test_valid)
Xl_surf_test <- model.matrix(~ l_surf - 1, data = test_valid)
X_bt_surf_test <- Xw_surf_test - Xl_surf_test

# 5. Combine with your rolling-diff covariates
newdata <- cbind(
  as.data.frame(X_bt_surf_test)#,
  #test_valid[, covariates]
)

# 6. Predict win-probabilities
preds <- predict(
  glm_surf,
  newdata = newdata,
```

```r
  type    = "response"
)
```

```
## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :
## prediction from rank-deficient fit; attr(*, "non-estim") has doubtful cases
```

```r
test_matches$pred_prob_bt_w_diffs_and_surf <- NA_real_
test_matches$pred_prob_bt_w_diffs_and_surf[valid_idx] <- preds

# evaluation metrics
test_matches$correct_prediction <- test_matches$pred_prob_bt_w_diffs_and_surf > 0.5
accuracy <- mean(test_matches$correct_prediction, na.rm = TRUE)
misclassification_rate <- 1 - accuracy
cat("Misclassification Rate:", misclassification_rate, "\n")
```

```
## Misclassification Rate: 0.4343229
```

```r
test_matches$brier_score <- (1 - test_matches$pred_prob_bt_w_diffs_and_surf)^2
mean_brier_score <- mean(test_matches$brier_score, na.rm = TRUE)
cat("Mean Brier Score:", mean_brier_score, "\n")
```

```
## Mean Brier Score: 0.4337129
```

```r
### RANDOM FOREST ###
library(randomForest)
```

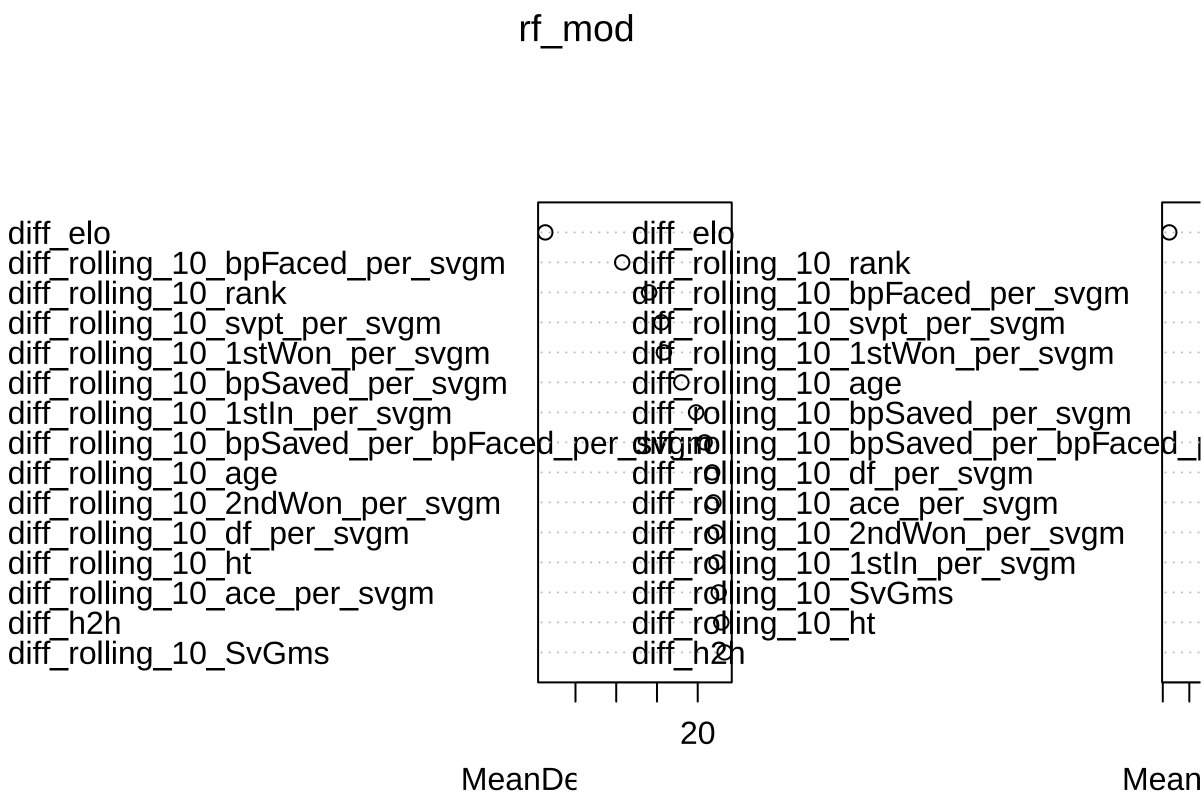
```
## randomForest 4.7-1.2

## Type rfNews() to see new features/changes/bug fixes.

##
## Attaching package: 'randomForest'

## The following object is masked from 'package:dplyr':
##
##     combine

## The following object is masked from 'package:ggplot2':
##
##     margin
```

```r
# 1. Identify your diff columns
diff_cols <- grep("^diff", names(train_matches), value = TRUE)

# 2. Build the RF training set
rf_pos <- train_matches %>%
  dplyr::select(all_of(diff_cols)) %>%
  mutate(label = 1)
```

```r
rf_neg <- train_matches %>%
  dplyr::select(all_of(diff_cols)) %>%
  mutate(across(all_of(diff_cols), ~ -.),  # flip the diffs
         label = 0)

rf_train <- bind_rows(rf_pos, rf_neg)

# 3. Fit a random forest
set.seed(42)
rf_mod <- randomForest(
  x           = rf_train %>% dplyr::select(all_of(diff_cols)),
  y           = factor(rf_train$label),   # as a factor for classification
  ntree       = 500,
  importance = TRUE
)

print(rf_mod)
```

```
##
## Call:
##  randomForest(x = rf_train %>% dplyr::select(all_of(diff_cols)),      y = factor(rf_train$label), nt
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 3
##
##         OOB estimate of  error rate: 35.27%
## Confusion matrix:
##      0    1 class.error
## 0 7365 4034   0.3538907
## 1 4007 7392   0.3515221
```

```r
varImpPlot(rf_mod)  # see which diffs matter most
```




```r
# 4. Predict on your test set
rf_test <- test_matches %>%
  filter(season > 2014) %>%
  dplyr::select(all_of(diff_cols))

# P(label=1) = probability that the "first-player" (i.e. winner_name) wins
test_matches$rf_prob <- predict(rf_mod, newdata = rf_test, type = "prob")[, "1"]

# 5. Evaluate
test_matches <- test_matches %>%
  mutate(
    rf_pred_win = rf_prob > 0.5,
    rf_brier    = (1 - rf_prob)^2  # since actual label is 1 for "winner_name"
  )

misclass_rf <- mean(!test_matches$rf_pred_win[test_matches$season > 2014])
brier_rf     <- mean(test_matches$rf_brier[test_matches$season > 2014])

cat("RF misclassification rate:", round(misclass_rf,4), "\n")
```

```
## RF misclassification rate: 0.3512
```

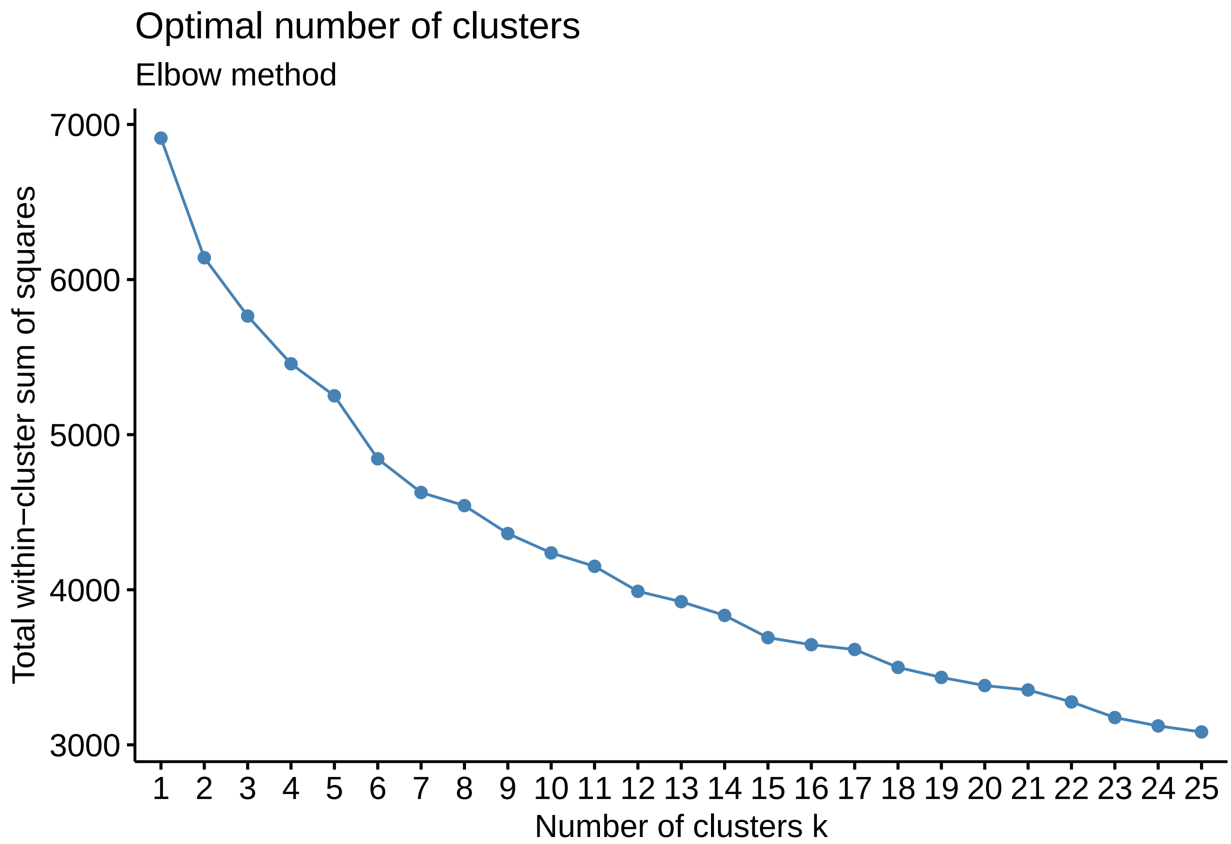
```r
cat("RF mean Brier score:       ", round(brier_rf,4), "\n")
```

```
## RF mean Brier score:        0.2149
```

```r
### CLUSTERING ###
library(factoextra)   # for fviz_nbclust() & fviz_cluster()
```

```
## Welcome! Want to learn more? See two factoextra-related books at https://goo.gl/ve3WBa
```

```r
train_test_matches <- matches %>% filter(season >= 2019)

# 1 & 2) stack winners and losers, flipping sign for losers
player_surface <- bind_rows(
  train_test_matches %>%
    select(surface, player = winner_name, all_of(covariates)),
  train_test_matches %>%
    select(surface, player = loser_name, all_of(covariates)) %>%
    mutate(across(all_of(covariates), ~ - .x))
)

# 3) average per player times surface
player_surface <- player_surface %>%
  group_by(player, surface) %>%
  summarise(
    across(all_of(covariates), ~ mean(.x, na.rm = TRUE)),
    n_matches = n(),
    .groups = "drop"
  ) %>%
  select(-n_matches)

# 4) pivot to wide: one row per player, cols = covariate_surface
player_wide <- player_surface %>%
  pivot_wider(
    id_cols     = player,
    names_from  = surface,
    values_from = all_of(covariates),
    names_sep   = "_"
  ) %>%
  drop_na()   # drop players missing any surface

# 5) scale features
feat_mat <- player_wide %>% select(-player)
feat_scaled <- scale(feat_mat)

# 6) elbow plot
fviz_nbclust(
  feat_scaled,
  kmeans,
  method = "wss",
  k.max  = 25
) +
  labs(
    subtitle = "Elbow method",
    x = "Number of clusters k",
    y = "Total within-cluster sum of squares"
  )
```




```
# 7) run k-means
set.seed(123)
km <- kmeans(feat_scaled, centers = 10, nstart = 25)

# 8) attach cluster back
player_clusters <- player_wide %>%
  mutate(cluster = km$cluster)

# now `player_clusters` has columns:
#   player | diff_rolling_10_df_per_svgm_Clay | ... | diff_h2h_Hard | cluster
#
# can inspect each cluster's "specialty" by:
cluster_summary <- player_clusters %>%
  bind_cols(as_tibble(feat_scaled)) %>%
  group_by(cluster) %>%
  summarize(across(
    contains("svpt"),     mean, na.rm = TRUE
  ), across(
    contains("df_per_svgm"), mean, na.rm = TRUE
  ), across(
    contains("1stWon"),    mean, na.rm = TRUE
  ), across(
    contains("2ndWon"),    mean, na.rm = TRUE
  ))
```

```
## New names:
```

```
## * ‘diff_rolling_10_df_per_svgm_Clay‘ -> ‘diff_rolling_10_df_per_svgm_Clay...2‘
## * ‘diff_rolling_10_df_per_svgm_Grass‘ ->
##   ‘diff_rolling_10_df_per_svgm_Grass...3‘
## * ‘diff_rolling_10_df_per_svgm_Hard‘ -> ‘diff_rolling_10_df_per_svgm_Hard...4‘
## * ‘diff_rolling_10_svpt_per_svgm_Clay‘ ->
##   ‘diff_rolling_10_svpt_per_svgm_Clay...5‘
## * ‘diff_rolling_10_svpt_per_svgm_Grass‘ ->
##   ‘diff_rolling_10_svpt_per_svgm_Grass...6‘
## * ‘diff_rolling_10_svpt_per_svgm_Hard‘ ->
##   ‘diff_rolling_10_svpt_per_svgm_Hard...7‘
## * ‘diff_rolling_10_1stWon_per_svgm_Clay‘ ->
##   ‘diff_rolling_10_1stWon_per_svgm_Clay...8‘
## * ‘diff_rolling_10_1stWon_per_svgm_Grass‘ ->
##   ‘diff_rolling_10_1stWon_per_svgm_Grass...9‘
## * ‘diff_rolling_10_1stWon_per_svgm_Hard‘ ->
##   ‘diff_rolling_10_1stWon_per_svgm_Hard...10‘
## * ‘diff_rolling_10_2ndWon_per_svgm_Clay‘ ->
##   ‘diff_rolling_10_2ndWon_per_svgm_Clay...11‘
## * ‘diff_rolling_10_2ndWon_per_svgm_Grass‘ ->
##   ‘diff_rolling_10_2ndWon_per_svgm_Grass...12‘
## * ‘diff_rolling_10_2ndWon_per_svgm_Hard‘ ->
##   ‘diff_rolling_10_2ndWon_per_svgm_Hard...13‘
## * ‘diff_rolling_10_bpSaved_per_bpFaced_per_svgm_Clay‘ ->
##   ‘diff_rolling_10_bpSaved_per_bpFaced_per_svgm_Clay...14‘
## * ‘diff_rolling_10_bpSaved_per_bpFaced_per_svgm_Grass‘ ->
##   ‘diff_rolling_10_bpSaved_per_bpFaced_per_svgm_Grass...15‘
## * ‘diff_rolling_10_bpSaved_per_bpFaced_per_svgm_Hard‘ ->
##   ‘diff_rolling_10_bpSaved_per_bpFaced_per_svgm_Hard...16‘
## * ‘diff_rolling_10_rank_Clay‘ -> ‘diff_rolling_10_rank_Clay...17‘
## * ‘diff_rolling_10_rank_Grass‘ -> ‘diff_rolling_10_rank_Grass...18‘
## * ‘diff_rolling_10_rank_Hard‘ -> ‘diff_rolling_10_rank_Hard...19‘
## * ‘diff_rolling_10_ht_Clay‘ -> ‘diff_rolling_10_ht_Clay...20‘
## * ‘diff_rolling_10_ht_Grass‘ -> ‘diff_rolling_10_ht_Grass...21‘
## * ‘diff_rolling_10_ht_Hard‘ -> ‘diff_rolling_10_ht_Hard...22‘
## * ‘diff_rolling_10_age_Clay‘ -> ‘diff_rolling_10_age_Clay...23‘
## * ‘diff_rolling_10_age_Grass‘ -> ‘diff_rolling_10_age_Grass...24‘
## * ‘diff_rolling_10_age_Hard‘ -> ‘diff_rolling_10_age_Hard...25‘
## * ‘diff_h2h_Clay‘ -> ‘diff_h2h_Clay...26‘
## * ‘diff_h2h_Grass‘ -> ‘diff_h2h_Grass...27‘
## * ‘diff_h2h_Hard‘ -> ‘diff_h2h_Hard...28‘
## * ‘diff_rolling_10_df_per_svgm_Clay‘ -> ‘diff_rolling_10_df_per_svgm_Clay...30‘
## * ‘diff_rolling_10_df_per_svgm_Grass‘ ->
##   ‘diff_rolling_10_df_per_svgm_Grass...31‘
## * ‘diff_rolling_10_df_per_svgm_Hard‘ -> ‘diff_rolling_10_df_per_svgm_Hard...32‘
## * ‘diff_rolling_10_svpt_per_svgm_Clay‘ ->
##   ‘diff_rolling_10_svpt_per_svgm_Clay...33‘
## * ‘diff_rolling_10_svpt_per_svgm_Grass‘ ->
##   ‘diff_rolling_10_svpt_per_svgm_Grass...34‘
## * ‘diff_rolling_10_svpt_per_svgm_Hard‘ ->
##   ‘diff_rolling_10_svpt_per_svgm_Hard...35‘
## * ‘diff_rolling_10_1stWon_per_svgm_Clay‘ ->
##   ‘diff_rolling_10_1stWon_per_svgm_Clay...36‘
## * ‘diff_rolling_10_1stWon_per_svgm_Grass‘ ->
##   ‘diff_rolling_10_1stWon_per_svgm_Grass...37‘
```

```
## * `diff_rolling_10_1stWon_per_svgm_Hard` ->
##   `diff_rolling_10_1stWon_per_svgm_Hard...38`
## * `diff_rolling_10_2ndWon_per_svgm_Clay` ->
##   `diff_rolling_10_2ndWon_per_svgm_Clay...39`
## * `diff_rolling_10_2ndWon_per_svgm_Grass` ->
##   `diff_rolling_10_2ndWon_per_svgm_Grass...40`
## * `diff_rolling_10_2ndWon_per_svgm_Hard` ->
##   `diff_rolling_10_2ndWon_per_svgm_Hard...41`
## * `diff_rolling_10_bpSaved_per_bpFaced_per_svgm_Clay` ->
##   `diff_rolling_10_bpSaved_per_bpFaced_per_svgm_Clay...42`
## * `diff_rolling_10_bpSaved_per_bpFaced_per_svgm_Grass` ->
##   `diff_rolling_10_bpSaved_per_bpFaced_per_svgm_Grass...43`
## * `diff_rolling_10_bpSaved_per_bpFaced_per_svgm_Hard` ->
##   `diff_rolling_10_bpSaved_per_bpFaced_per_svgm_Hard...44`
## * `diff_rolling_10_rank_Clay` -> `diff_rolling_10_rank_Clay...45`
## * `diff_rolling_10_rank_Grass` -> `diff_rolling_10_rank_Grass...46`
## * `diff_rolling_10_rank_Hard` -> `diff_rolling_10_rank_Hard...47`
## * `diff_rolling_10_ht_Clay` -> `diff_rolling_10_ht_Clay...48`
## * `diff_rolling_10_ht_Grass` -> `diff_rolling_10_ht_Grass...49`
## * `diff_rolling_10_ht_Hard` -> `diff_rolling_10_ht_Hard...50`
## * `diff_rolling_10_age_Clay` -> `diff_rolling_10_age_Clay...51`
## * `diff_rolling_10_age_Grass` -> `diff_rolling_10_age_Grass...52`
## * `diff_rolling_10_age_Hard` -> `diff_rolling_10_age_Hard...53`
## * `diff_h2h_Clay` -> `diff_h2h_Clay...54`
## * `diff_h2h_Grass` -> `diff_h2h_Grass...55`
## * `diff_h2h_Hard` -> `diff_h2h_Hard...56`
```

```
## Warning: There was 1 warning in `summarize()`.
## i In argument: `across(contains("svpt"), mean, na.rm = TRUE)`.
## i In group 1: `cluster = 1`.
## Caused by warning:
## ! The `...` argument of `across()` is deprecated as of dplyr 1.1.0.
## Supply arguments directly to `.fns` through an anonymous function instead.
##
##   # Previously
##   across(a:b, mean, na.rm = TRUE)
##
##   # Now
##   across(a:b, \(x) mean(x, na.rm = TRUE))
```

```r
library(ggplot2)
library(ggrepel)
library(dplyr)
library(colorspace)

# PCA projection of your scaled features
pca_res <- prcomp(feat_scaled, center = TRUE, scale. = FALSE)

# build the scores data.frame
scores <- as.data.frame(pca_res$x[, 1:2])
colnames(scores) <- c("PC1", "PC2")
scores$player  <- player_clusters$player
scores$cluster <- factor(player_clusters$cluster)
```

```r
# generate distinct HCL hues
pal <- qualitative_hcl(10, palette = "Dark 3")

ggplot(scores, aes(PC1, PC2, color = cluster, label = player)) +
  geom_point(size = 2, alpha = 0.8) +
  geom_text_repel(
    max.overlaps  = 25,
    size          = 3,
    box.padding   = 0.4,
    point.padding = 0.3
  ) +
  scale_color_manual(values = pal) +
  theme_minimal(base_size = 14) +
  labs(
    title    = "Serve-Specialist Clusters on PC1 vs PC2",
    x        = "PC1",
    y        = "PC2",
    color    = "Cluster"
  )
```

```
## Warning: ggrepel: 214 unlabeled data points (too many overlaps). Consider
## increasing max.overlaps
```