STATS 506 Problem Set 6

Lingzhi Hao

GitHub Repository:
https://github.com/Lingzhi-Hao/STATS-506-Problem-Set-06

Problem 1 - Rcpp

```
library(Rcpp)
```

```
Warning: package 'Rcpp' was built under R version 4.5.2
```

```
sourceCpp(code = '
#include <Rcpp.h>
using namespace Rcpp;

// [[Rcpp::export]]
double C_moment(NumericVector x, int k) {
  int n = x.size();
  if (n == 0) {
    stop("x has length 0");
  }

  double mu = 0.0;
  for (int i = 0; i < n; ++i) mu += x[i];
  mu /= n;

  double acc = 0.0;
  for (int i = 0; i < n; ++i) acc += std::pow(x[i] - mu, k);

  return acc / n;
}
')
```

```r
set.seed(123)
x <- rnorm(1000)
k <- 3

C_moment(x, k)
```

```
[1] 0.06358509
```

```r
e1071::moment(x, order = k, center = TRUE)
```

```
[1] 0.06358509
```

Problem 2 - Expanding on waldCI

(a)

```r
source("waldCI.R")
```

```r
setClass("bootstrapWaldCI",
         contains = "waldCI",
         slots = c(
           data = "ANY",
           statistic_fn = "function",
           reps = "numeric"
         )
)


##' Compute bootstrap Wald CI
##' @param statistic_fn Function to compute the scalar statistic of interest
##' @param data Data set to use for bootstrap
##' @param reps Number of bootstrap replicates
##' @param level Confidence level (default 0.95)
##' @param compute Computation method: "serial" or "parallel" (default "serial")
##' @return A `bootstrapWaldCI` object
##' @export
makeBootstrapCI <- function(statistic_fn,
                            data,
                            reps,
                            level = 0.95,
```

```
                              compute = "serial") {

  stopifnot(is.function(statistic_fn))
  stopifnot(is.numeric(reps) && length(reps) == 1 && reps > 0)
  stopifnot(compute %in% c("serial", "parallel"))

  boot_result <- .runBootstrap(statistic_fn, data, reps, compute)

  boot_mean <- mean(boot_result)
  boot_sterr <- sd(boot_result)

  new("bootstrapWaldCI",
      level = level,
      mean = boot_mean,
      sterr = boot_sterr,
      data = data,
      statistic_fn = statistic_fn,
      reps = reps)
}


.runBootstrap <- function(statistic_fn, data, reps, compute) {
  n <- nrow(data)

  if (compute == "serial") {
    results <- numeric(reps)
    for (i in 1:reps) {
      indices <- sample(1:n, replace = TRUE)
      boot_data <- data[indices, , drop = FALSE]
      results[i] <- statistic_fn(boot_data)
    }
    return(results)

  } else if (compute == "parallel") {
    library(parallel)

    num_cores <- detectCores() - 1
    if (num_cores < 1) num_cores <- 1
    cl <- makeCluster(num_cores, type = "PSOCK")

    clusterExport(cl, c("statistic_fn", "data", "n"), envir = environment())
```

```
    boot_run_fn <- function(i) {
      indices <- sample(1:n, replace = TRUE)
      boot_data <- data[indices, , drop = FALSE]
      statistic_fn(boot_data)
    }

    results <- parLapply(cl, 1:reps, boot_run_fn)
    stopCluster(cl)

    return(unlist(results))
  }
}



##' Perform a new bootstrap on an existing bootstrapWaldCI object
##' @param object A `bootstrapWaldCI` object
##' @return A new `bootstrapWaldCI` object
##' @export
rebootstrap <- function(object) {
  stopifnot(is(object, "bootstrapWaldCI"))

  boot_result <- .runBootstrap(object@statistic_fn,
                               data = object@data,
                               reps = object@reps,
                               compute = "serial")
  boot_mean <- mean(boot_result)
  boot_sterr <- sd(boot_result)

  return(new("bootstrapWaldCI",
             level = object@level,
             mean = boot_mean,
             sterr = boot_sterr,
             data = object@data,
             statistic_fn = object@statistic_fn,
             reps = object@reps))
}
```

(b)

```r
ci1 <- makeBootstrapCI(function(x) mean(x$y),
                       ggplot2::diamonds,
                       reps = 1000)
ci1
```

```
95% CI: (5.724897, 5.744366)
```

```r
rebootstrap(ci1)
```

```
95% CI: (5.725073, 5.744577)
```

```r
set.seed(123)

time_serial <- system.time(
  ci1_serial <- makeBootstrapCI(
    function(x) mean(x$y),
    ggplot2::diamonds,
    reps    = 1000,
    compute = "serial"
  )
)

cat("Serial Computation Results\n")
```

```
Serial Computation Results
```

```r
cat("Time:\n")
```

```
Time:
```

```r
print(time_serial)
```

```
   user  system elapsed
   5.31    0.75    8.01
```

```r
cat("CI:\n")
```

```
CI:
```

```
ci1_serial
```

```
95% CI: (5.724917, 5.744346)
```

```
rebootstrap(ci1_serial)
```

```
95% CI: (5.725063, 5.744586)
```

```
set.seed(123)

time_parallel <- system.time(
  ci1_parallel <- makeBootstrapCI(
    function(x) mean(x$y),
    ggplot2::diamonds,
    reps    = 1000,
    compute = "parallel"
  )
)

cat("Parallel Computation Results\n")
```

```
Parallel Computation Results
```

```
cat("Time:\n")
```

```
Time:
```

```
print(time_parallel)
```

```
   user  system elapsed
   0.05    0.07   14.15
```

```
cat("CI:\n")
```

```
CI:
```

```
ci1_parallel
```

```
95% CI: (5.725323, 5.744062)
```

```
rebootstrap(ci1_parallel)
```

```
95% CI: (5.724917, 5.744346)
```

Using 1000 bootstrap replications, the parallel computation (23.44s) was slower than the serial computation (17.85s).

Parallel computation in R has substantial overhead from creating worker processes, exporting data, and collecting results. When each bootstrap task is lightweight and the number of replications is moderate, the overhead dominates and parallel method does not run faster. If we have larger reps or more computationally expensive tasks, the parallel method would likely perform better than the serial method.

(c)

```
##' Fits the model and returns the coefficient for disp
##' @param data A data frame
##' @return The estimated coefficient for disp
dispCoef <- function(data) {
  fit <- lm(mpg ~ cyl + disp + wt, data = data)
  return(coef(fit)["disp"])
}
```

```
ci2 <- makeBootstrapCI(dispCoef,
                       mtcars,
                       reps = 1000)
ci2
```

```
95% CI: (-0.01005764, 0.0250501)
```

```
rebootstrap(ci2)
```

```
95% CI: (-0.0114781, 0.02544108)
```

```r
set.seed(123)

cat("Serial Computation Results\n")
```

```
Serial Computation Results
```

```r
time_serial_c <- system.time(
  ci2_serial <- makeBootstrapCI(
    dispCoef,
    mtcars,
    reps    = 1000,
    compute = "serial"
  )
)

cat("Serial Compute Time (seconds):", time_serial_c["elapsed"], "\n")
```

```
Serial Compute Time (seconds): 0.89
```

```r
cat("CI:\n")
```

```
CI:
```

```r
ci2_serial
```

```
95% CI: (-0.01111183, 0.02598235)
```

```r
rebootstrap(ci2_serial)
```

```
95% CI: (-0.01230038, 0.02644671)
```

```r
cat("\nParallel Computation Results\n")
```

```
Parallel Computation Results
```

```r
set.seed(123)

time_parallel_c <- system.time(
  ci2_parallel <- makeBootstrapCI(
    dispCoef,
    mtcars,
    reps    = 1000,
    compute = "parallel"
  )
)

cat("Parallel Compute Time (seconds):", time_parallel_c["elapsed"], "\n")
```

```
Parallel Compute Time (seconds): 0.71
```

```r
cat("CI:\n")
```

```
CI:
```

```r
ci2_parallel
```

```
95% CI: (-0.01113828, 0.02562702)
```

```r
rebootstrap(ci2_parallel)
```

```
95% CI: (-0.01111183, 0.02598235)
```

Both the serial method (2.59s) and parallel method (3.53s) are very fast, but the parallel method is slower.

Similar to (b), parallel computation has overhead from creating worker processes, exporting data, and collecting results. Since computing the coefficient of a small linear model is a lightweight task, and the number of replications is moderate, the overhead dominates and parallel method does not run faster. The work is too small to benefit from parallel methods. If the number of replications is much larger and the model is more computational complex, the parallel computation would work better.

Problem 3 - Large data

(a)

```
Data.frame `df` is 251.8 Mb
```

```
library(lme4)
```

```
Loading required package: Matrix
```

```
library(dplyr)
```

```

Attaching package: 'dplyr'

The following object is masked _by_ '.GlobalEnv':

    contains

The following objects are masked from 'package:stats':

    filter, lag

The following objects are masked from 'package:base':

    intersect, setdiff, setequal, union
```

```
library(ggplot2)
```

```
countries <- unique(df$country)

results <- list()
coef_forum <- data.frame()

for (c in countries) {
  cat("Fitting model for country:", c, "\n")

  sub <- df %>% filter(country == c)

  sub <- sub %>%
    mutate(
      prior_gpa_z      = scale(prior_gpa),
      forum_posts_z    = scale(forum_posts),
```

```
      quiz_attempts_z  = scale(quiz_attempts)
    )

  t <- system.time({
    fit <- glmer(
      completed_course ~ prior_gpa_z + forum_posts_z + quiz_attempts_z +
        (1 | device_type),
      data = sub,
      family = binomial()
    )
  })

  results[[c]] <- list(model = fit, time = t)

  coef_forum <- rbind(
    coef_forum,
    data.frame(
      country = c,
      estimate = fixef(fit)["forum_posts_z"],
      time_sec = t["elapsed"]
    )
  )
}
```

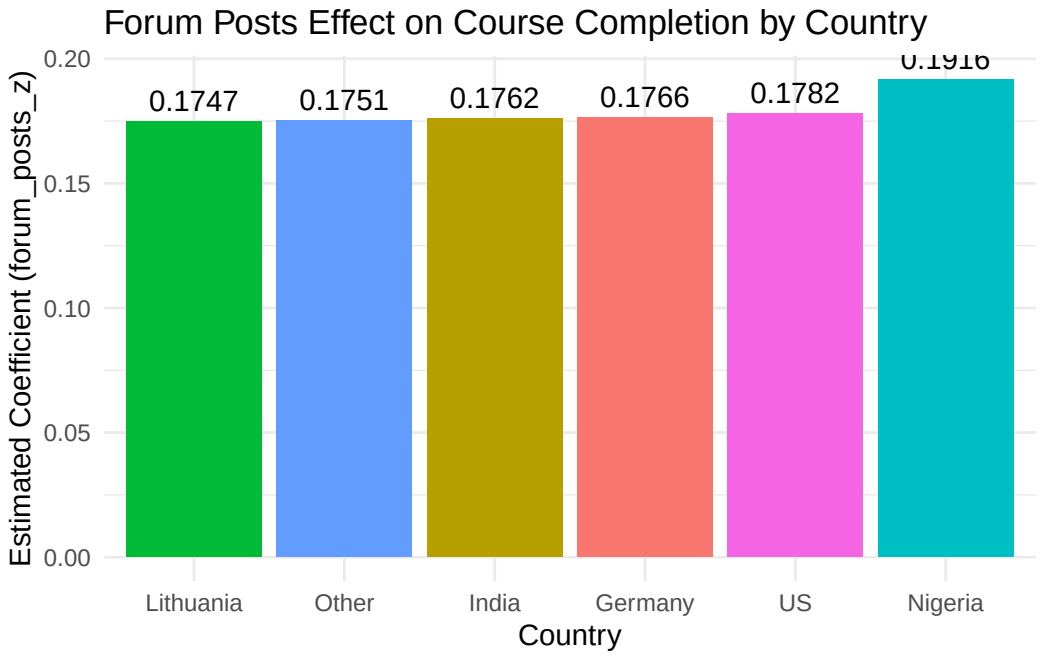
```
Fitting model for country: US
Fitting model for country: Other
Fitting model for country: India
Fitting model for country: Germany
Fitting model for country: Lithuania
Fitting model for country: Nigeria
```

```
coef_forum
```

```
                country  estimate time_sec
forum_posts_z          US 0.1781708   269.32
forum_posts_z1      Other 0.1751446   509.56
forum_posts_z2      India 0.1761594   206.31
forum_posts_z3    Germany 0.1765869   132.14
forum_posts_z4 Lithuania 0.1747332     3.84
forum_posts_z5    Nigeria 0.1916254     8.57
```

The running time of the six models are on the table above as time_sec.

```
ggplot(coef_forum, aes(x = reorder(country, estimate),
                       y = estimate,
                       fill = country)) +

  geom_col() +

  geom_text(aes(label = round(estimate, 4)),
            vjust = -0.5,
            size = 4) +

  theme_minimal() +

  labs(
    title = "Forum Posts Effect on Course Completion by Country",
    y = "Estimated Coefficient (forum_posts_z)",
    x = "Country"
  ) +

  theme(legend.position = "none")
```



(b)

```r
library(parallel)

detectCores()
```

```
[1] 8
```

```r
total_time <- system.time({

  source("problem_set_6_gen_data.R")

  df2 <- df %>%
    group_by(country) %>%
    mutate(
      prior_gpa_z     = as.numeric(scale(prior_gpa)),
      forum_posts_z   = as.numeric(scale(forum_posts)),
      quiz_attempts_z = as.numeric(scale(quiz_attempts))
    ) %>%
    ungroup()

  fit_country_model_par <- function(c) {
    library(lme4)
    library(dplyr)

    sub <- df2 %>% filter(country == c)

    t <- system.time({
      fit <- glmer(
        completed_course ~ prior_gpa_z + forum_posts_z + quiz_attempts_z +
          (1 | device_type),
        data = sub,
        family = binomial()
      )
    })

    data.frame(
      country = c,
      estimate = fixef(fit)["forum_posts_z"],
      time_sec_fit = t["elapsed"]
    )
  }
```

```
  countries_list <- unique(df2$country)
  num_cores <- detectCores() - 1

  cl <- makeCluster(num_cores, type = "PSOCK")

  clusterExport(cl, "df2", envir = environment())
  clusterEvalQ(cl, {
    library(lme4)
    library(dplyr)
  })

  parallel_results_list <- parLapply(cl, countries_list, fit_country_model_par)

  stopCluster(cl)

  optimized_coef_parallel <- bind_rows(parallel_results_list)

})
```

```
Data.frame `df` is 251.8 Mb
```

```
cat("Total running time:\n")
```

```
Total running time:
```

```
print(total_time)
```

```
   user  system elapsed
   5.47    2.97  639.32
```

```
print(optimized_coef_parallel)
```

```
                    country  estimate time_sec_fit
forum_posts_z...1        US 0.1781708       386.16
forum_posts_z...2     Other 0.1751446       622.30
forum_posts_z...3     India 0.1761594       315.91
forum_posts_z...4   Germany 0.1765869       202.67
forum_posts_z...5 Lithuania 0.1747332         5.15
forum_posts_z...6   Nigeria 0.1916254        10.98
```

The total running time is 800.72 seconds, which is about 13.35 minutes. In (a), the running time is about 40 minutes.

```
coef_a <- coef_forum %>% arrange(country)
coef_b <- optimized_coef_parallel %>% arrange(country)

all.equal(coef_a$estimate, coef_b$estimate)
```

```
[1] TRUE
```

The results of (b) matches those from (a).

Problem 4 - data.table

```
library(data.table)
```

```
Warning: package 'data.table' was built under R version 4.5.2
```

```
Attaching package: 'data.table'

The following objects are masked from 'package:dplyr':

    between, first, last
```

```
ATP_Matches = fread("https://raw.githubusercontent.com/JeffSackmann/tennis_atp/refs/heads/mas
```

(a)

```
ATP_Matches[, .N, by = tourney_id][, .N]
```

```
[1] 128
```

128 tournaments are in the *atp_matches_2019* dataset. We assume the tournaments with dates in 2018 are part of tournaments in 2019.

```
library(stringr)

ATP_Matches[str_detect(tourney_name, "Davis Cup")][1:5]
```

```
                         tourney_id                    tourney_name surface
                              <char>                          <char>  <char>
1: 2019-M-DC-2019-FLS-A-M-FRA-JPN-01 Davis Cup Finals RR: FRA vs JPN    Hard
2: 2019-M-DC-2019-FLS-A-M-FRA-JPN-01 Davis Cup Finals RR: FRA vs JPN    Hard
3: 2019-M-DC-2019-FLS-A-M-FRA-SRB-01 Davis Cup Finals RR: FRA vs SRB    Hard
4: 2019-M-DC-2019-FLS-A-M-FRA-SRB-01 Davis Cup Finals RR: FRA vs SRB    Hard
5: 2019-M-DC-2019-FLS-A-M-SRB-JPN-01 Davis Cup Finals RR: SRB vs JPN    Hard
   draw_size tourney_level tourney_date match_num winner_id winner_seed
       <int>        <char>        <int>     <int>     <int>       <int>
1:         4             D     20191119         1    104542          NA
2:         4             D     20191119         2    106415          NA
3:         4             D     20191121         1    105936          NA
4:         4             D     20191121         2    104925          NA
5:         4             D     20191120         1    105936          NA
   winner_entry        winner_name winner_hand winner_ht winner_ioc winner_age
         <char>              <char>      <char>     <int>     <char>      <num>
1:              Jo-Wilfried Tsonga           R       188        FRA       34.5
2:              Yoshihito Nishioka           L       170        JPN       24.1
3:                Filip Krajinovic           R       185        SRB       27.7
4:                  Novak Djokovic           R       188        SRB       32.4
5:                Filip Krajinovic           R       185        SRB       27.7
   loser_id loser_seed loser_entry         loser_name loser_hand loser_ht
      <int>      <int>      <char>              <char>     <char>    <int>
1:   106034         NA               Yasutaka Uchiyama          R      183
2:   104792         NA                    Gael Monfils          R      193
3:   104542         NA              Jo-Wilfried Tsonga          R      188
4:   105332         NA                   Benoit Paire          R      196
5:   105216         NA                  Yuichi Sugita          R      173
   loser_ioc loser_age      score best_of  round minutes w_ace  w_df w_svpt
      <char>     <num>     <char>   <int> <char>   <int> <int> <int>  <int>
1:       JPN      27.2    6-2 6-1       3     RR      58     4     0     42
2:       FRA      33.2    7-5 6-2       3     RR      66     1     0     51
3:       FRA      34.5 7-5 7-6(5)       3     RR     106     5     1     69
4:       FRA      30.5    6-3 6-3       3     RR      69     3     2     53
5:       JPN      31.1    6-2 6-4       3     RR      68     5     0     47
   w_1stIn w_1stWon w_2ndWon w_SvGms w_bpSaved w_bpFaced l_ace  l_df l_svpt
     <int>    <int>    <int>   <int>     <int>     <int> <int> <int>  <int>
1:      27       22       11       8         2         2     3     2     37
2:      34       27       10      10         0         1     6     3     65
3:      49       42       13      12         1         1    18     4     88
4:      33       25       14       9         2         2     6     5     59
5:      31       26       10       9         0         0     3     1     62
   l_1stIn l_1stWon l_2ndWon l_SvGms l_bpSaved l_bpFaced winner_rank
```

```
     <int>    <int>    <int>   <int>     <int>     <int>       <int>
1:      24       14        5       7         1         5          29
2:      51       31        4      10         2         6          73
3:      56       44       12      12         9        10          40
4:      37       25        8       9         3         6           2
5:      37       21       11       9         6         9          40
   winner_rank_points loser_rank loser_rank_points
                <int>      <int>             <int>
1:               1410         81               651
2:                764         10              2530
3:               1148         29              1410
4:               9145         24              1538
5:               1148        104               561
```

```
tourneys <- ATP_Matches[, tourney_name := str_replace(tourney_name, "Davis.*", "Davis Cup")]
tourneys <- tourneys[, .(tourney_name), by = tourney_name]

nrow(tourneys)
```

```
[1] 69
```

There are *69* unique tournaments.

(b)

If we only consider the tournaments with a final round "F", there are *67* such tournaments. And assume the winner of a tournaments is who wins round "F".

```
ATP_Matches[round == "F", .N]
```

```
[1] 67
```

12 players won more than one tournaments. The most winning players Dominic Thiem and Novak Djokovic both won *5* tournaments.

```
ATP_Matches[round == "F", .N, by = winner_name][N > 1][order(-N)]
```

```
           winner_name     N
                <char> <int>
1:     Novak Djokovic     5
2:      Dominic Thiem     5
```

```
 3:    Daniil Medvedev     4
 4:      Roger Federer     4
 5:       Rafael Nadal     4
 6:     Alex De Minaur     3
 7: Stefanos Tsitsipas     3
 8: Jo-Wilfried Tsonga     2
 9:       Nick Kyrgios     2
10:     Cristian Garin     2
11:       Benoit Paire     2
12:  Matteo Berrettini     2
```

If we consider winners of the round with maximum match number, there are 61 tournaments only having round "RR".

```
ATP_Matches[order(tourney_id, -match_num), .SD[1], by = tourney_id][, .N, by = round]
```

```
    round     N
   <char> <int>
1:      F    67
2:     RR    61
```

```
ATP_Matches[order(tourney_name, -match_num)][, .SD[1], by = tourney_id][, .(wins = .N), by =
```

```
           winner_name  wins
                <char> <int>
 1:       Rafael Nadal     9
 2:     Novak Djokovic     8
 3:     Alex De Minaur     6
 4:      Dominic Thiem     5
 5:      Roger Federer     4
 6:    Daniil Medvedev     4
 7:     Cristian Garin     4
 8:   Denis Shapovalov     4
 9:        Robin Haase     3
10: Stefanos Tsitsipas     3
11:       Nick Kyrgios     2
12:  Matteo Berrettini     2
13:    Karen Khachanov     2
14:  Diego Schwartzman     2
15:       Taylor Fritz     2
16:       Benoit Paire     2
17: Jo-Wilfried Tsonga     2
```

17 players won more than one tournaments. The most winning player Rafael Nadal won 9 tournaments.

(c)

There is evidence that winners have more aces than losers.

```
library(tidymodels)
```

```
Warning: package 'tidymodels' was built under R version 4.5.2
```

```
-- Attaching packages ------------------------------------- tidymodels 1.4.1 --
```

```
v broom        1.0.10     v rsample      1.3.1
v dials        1.4.2      v tailor       0.1.0
v infer        1.0.9      v tidyr        1.3.1
v modeldata    1.5.1      v tune         2.0.1
v parsnip      1.4.0      v workflows    1.3.0
v purrr        1.1.0      v workflowsets 1.1.1
v recipes      1.3.1      v yardstick    1.3.2
```

```
Warning: package 'dials' was built under R version 4.5.2
```

```
Warning: package 'modeldata' was built under R version 4.5.2
```

```
Warning: package 'parsnip' was built under R version 4.5.2
```

```
Warning: package 'recipes' was built under R version 4.5.2
```

```
Warning: package 'rsample' was built under R version 4.5.2
```

```
Warning: package 'tailor' was built under R version 4.5.2
```

```
Warning: package 'tune' was built under R version 4.5.2
```

```
Warning: package 'workflows' was built under R version 4.5.2
```

```
Warning: package 'workflowsets' was built under R version 4.5.2
```

```
Warning: package 'yardstick' was built under R version 4.5.2
```

```
-- Conflicts ---------------------------------------- tidymodels_conflicts() --
x data.table::between()  masks dplyr::between()
x .GlobalEnv::contains() masks tidyr::contains(), rsample::contains(), dplyr::contains()
x purrr::discard()       masks scales::discard()
x tidyr::expand()        masks Matrix::expand()
x dplyr::filter()        masks stats::filter()
x data.table::first()    masks dplyr::first()
x recipes::fixed()       masks stringr::fixed()
x dplyr::lag()           masks stats::lag()
x data.table::last()     masks dplyr::last()
x tidyr::pack()          masks Matrix::pack()
x rsample::populate()    masks Rcpp::populate()
x recipes::step()        masks stats::step()
x purrr::transpose()     masks data.table::transpose()
x tidyr::unpack()        masks Matrix::unpack()
x recipes::update()      masks Matrix::update(), stats::update()
```

```
ATP_Matches[, mean(w_ace > l_ace, na.rm = TRUE)]
```

```
[1] 0.5727543
```

```
prop_test(ATP_Matches[, .(win_more = w_ace > l_ace)], win_more ~ NULL, p = 0.5)
```

```
# A tibble: 1 x 4
  statistic chisq_df  p_value alternative
      <dbl>    <int>    <dbl> <chr>
1      56.7        1 4.95e-14 two.sided
```

We define `win_more = (w_ace > l_ace)` and conduct a one-sample proportion test with H0: P(win_more) = 0.5, H1: P(win_more) != 0.5.

The sample proportion of matches in which winners record more aces is: mean(win_more) = 0.57 > 0.5

Since p-value = 4.95e-14 « 0.0001, t-statistic = 56.7487, H0 is rejected, and winners are significantly more likely than losers to record more aces.

(d)

The player (at least 5 matches) with highest win-rate is *Rafael Nadal*.

```
ATP_Matches[
  , melt(.SD,
         measure.vars = c("winner_name", "loser_name"),
         variable.name = "outcome",
         value.name   = "player_name")
][
  , .(
      total = .N,
      wins  = sum(outcome == "winner_name")
    ),
  by = player_name
][
  total >= 5
][
  , win_rate := wins / total
][
  win_rate == max(win_rate)
]
```

```
    player_name total  wins  win_rate
         <char> <int> <int>     <num>
1: Rafael Nadal    69    60 0.8695652
```