Assignment 04

Statistical Computing and Empirical Methods

A word of advice

Think of the SCEM labs like going to the gym: if you pay for gym membership, but instead of working out you use a machine to lift the weights for you, you won't grow any muscle.

ChatGPT, DeepSeek, Claude and other GenAI tools can provide answers to most of the questions below. Before you try that, please consider the following: answering the specific questions below is not the point of this assignment. Instead, the questions are designed to give you the chance to develop a better understanding of estimation concepts and a certain level of **statistical thinking**. These are essential skills for any data scientist, even if they end up using generative AI - to write an effective prompt and to catch the common (often subtle) errors that AI produces when trying to solve anything non-trivial.

A very important part of this learning involves not having the answers ready-made for you but instead taking the time to actually search for the answer, trying something, getting it wrong, and trying again.

So, make the best use of this session. The assignments are not marked, so it is much better to try them yourself even if you get incorrect answers (you'll be able to correct yourself later when you receive feedback) than to submit a perfect, but GPT'd solution.

Introduction

Before starting, make sure you have reviewed the Week 4 lecture slides and watched all videos for the week. Additionally, download the following data files from Blackboard and save them in the same folder as your solutions Rmarkdown. We will use these files for some of the questions in this assignment.

- `HockeyLeague.xlsx`
- `pharma_drug_spending.csv`

This assignment will be done in **RStudio**. To enter your solutions, please use the pre-structured R markdown template provided for this week. This is a similar format to what you will use in the final coursework.

You can **optionally** submit your .Rmd solutions file by 15:00h Friday 10 October. This will help us understand your work but will not count towards your final grade.

If you want to upload your .Rmd file, use the "Assignment 04" link under the *Assessments, Submission and Feedback* tab in the course Blackboard page.

Part I: Tidy data and iteration

This part is mainly about tidy data and iteration in R. The main package you'll use for data reshaping is `tidyr`. You'll also use package `purrr` for some vectorisation operations. Both of these packages are part of the tidyverse meta-package (tidyverse also includes important packages such as `dplyr`, `ggplot2`, and others). The `tidyverse` packages will be auto-loaded for you at the start of the solutions sheet.

Q1: Missing data and iteration

a. Create a function called `impute_by_median` which imputes missing values based on the median of the sample, rather than the mean. Test your function on the sample vector provided in the solutions file.

b. Next, generate a data frame with two variables, x and y. Define variable x as a sequence of n values with $x_1 = 0,\ x_n = 10$, and $x_{i+1} = x_i + 0.1$, for $i = 1, \ldots, n - 1$. Variable y should be set such that $y_i = 5x_i + 1$. Define your data frame with those two columns and store it as variable `df_xy`.

c. Function `map2()` and its variations are similar to `map()`, but iterating over two variables in parallel rather than one. You can learn more at https://purrr.tidyverse.org/reference/map2.html (or by typing `?map2` on the R prompt). The following simple example shows you how `map2_dbl()` can be combined with `mutate()` (from package dplyr).

```r
library(dplyr)
df_xy %>%
  mutate(z = map2_dbl(x, y, .f = ~.x+.y)) %>%
  head(n=5)
```

You will use `map2_dbl()` to generate a new data frame with missing data. First, create a function called `sometimes_missing` with two input arguments: `index` and `value`. The function should return `NA` if index is divisible by 5, and return `value` otherwise. For instance, the function should return the following outputs:

```r
sometimes_missing(12, 3)

## [1] 3

sometimes_missing(15, 3)

## [1] NA
```

Next, generate a new data frame called `df_xy_missing` with two variables, x and y, but some missing data. Variable x should be the same one you defined for `df_xy`. Variable y should contain a sequence of n values, $y_1, \ldots, y_n$, in which $y_i = NA$ if i is divisible by 5, and $y_i = 5x_i + 1$ otherwise. You may want to explore functions `row_number()`, `map2_dbl()` and `mutate()` (but there are many ways of achieving the same result, so feel free to explore).

d. Create a new data frame `df_xy_imputed` by imputing all the missing values in `df_xy_missing$y` with the median, using your function `impute_by_median()`.

Q2: Tidying data

In this question you will read in data from a spreadsheet and apply some data wrangling tasks to tidy that data. First, make sure you have downloaded the excel spreadsheet `HockeyLeague.xlsx`. This file contains two worksheets: one with the wins for each team and one with the losses for each team. To read this spreadsheet into R, you can use package `readxl` (this is already already installed and loaded automatically in the answers file). To read sheets within the excel file, you can use:

```
library(readxl)
## note: you can see your current working folder using getwd()
## and set it using setwd().

folder_path <- "."  # current working folder. Change this if needed.
file_path   <- paste0(folder_path, "/HockeyLeague.xlsx") # set the file path

wins_data_frame <- read_excel(file_path, sheet = "Wins")
losses_data_frame <- read_excel(file_path, sheet = "Losses")

# Inspect the first 3 rows of the first five columns of the wins dataframe:
wins_data_frame[1:3, 1:5]
```

```
## # A tibble: 3 × 5
##   ...1   `1990`   `1991`   `1992`   `1993`
##   <chr>  <chr>    <chr>    <chr>    <chr>
## 1 Ducks  30 of 50 11 of 50 30 of 50 12 of 50
## 2 Eagles 24 of 50 12 of 50 37 of 50 14 of 50
## 3 Hawks  20 of 50 22 of 50 33 of 50 11 of 50
```

A cell value of the form *"a of b"* means that *a* games were won out of a total of *b* for that season.

Is this tidy data?

a. Now apply your data wrangling skills to transform the `wins_data_frame` data frame object into a data frame called `wins_tidy`, which must contain the same information but in a tidy format. In this case, `wins_tidy` should have just four columns:

- `Team`, with the team name (character)
- `Year`, with the year (integer)
- `Wins`, with the number of wins (integer)
- `Total`, with the number of games (integer)

You can do this task combining functions `pivot_longer()`, `mutate()`, `rename()` and `separate()`. You may also need to use `as.integer()`

b. `HockeyLeague.xlsx` also contains a sheet with the losses for each team by season. Apply a similar procedure to read the data from this sheet and transform that data into a data frame called `losses_tidy` with four columns: `Team`, `Year`, `Losses`, and `Total`, similar to their counterparts in the `wins_tidy` data frame.

c. Now combine your two data frames, `wins_tidy` and `losses_tidy`, into a single data frame entitled `hockey_df`, which will have 248 rows and 9 columns (in the following order)): `Team`, `Year`, `Wins`, `Losses`, `Total`, plus the following:

- `Draws`, with the number of draws, which can be calculated for each team on each season as `Total`-(`Wins`+`Losses`). The `Total` column is available in both `wins_tidy` and `losses_tidy`.
- `Wins_rt`, with the proportion of wins (i.e. `Wins`/`Total`)
- `Losses_rt`, with the proportion of losses (i.e. `Losses`/`Total`)
- `Draws_rt`, with the proportion of draws (i.e. `Draws`/`Total`).

To do this you can make use of `mutate()` and `select()`. You may also want to use the `across()` function (not necessary, but can result in a slightly neater solution).

The top five rows of your final data frame should look as follows:

```
hockey_df %>% head(5)

## # A tibble: 5 × 9
##   Team   Year  Wins Losses Draws Total Wins_rt Losses_rt Draws_rt
##   <chr> <int> <int>  <int> <int> <int>   <dbl>     <dbl>    <dbl>
## 1 Ducks  1990    30     20     0    50    0.6       0.4      0
## 2 Ducks  1991    11     37     2    50    0.22      0.74     0.04
## 3 Ducks  1992    30      1    19    50    0.6       0.02     0.38
## 4 Ducks  1993    12     30     8    50    0.24      0.6      0.16
## 5 Ducks  1994    24      7    19    50    0.48      0.14     0.38
```

d. To conclude this task, generate a summary data frame called `summary_teams`, which contains, for each team, the *median win rate*, the *mean win rate*, the *median loss rate* and the *mean loss rate*. The number of rows in your summary should equal the number of teams. Your rows should be sorted in *descending order of median win rate*.

The following functions may be useful: `select()`, `group_by()`, `across()` and `arrange()`.

Part II: confidence intervals

This part relates to this week's lecture on calculating confidence intervals. Notice that there are new concepts here that were not covered in the lectures - read those carefully, as they are important and can be helpful later in the course.

Q3. CIs for the Mean and Variance of a normal variable

a. Write a function `ci_mean_known_var` that calculates a $(1-\alpha)$ confidence interval for the **mean** of a normal distribution when the variance is known, using the quantiles of the standard normal distribution (function `qnorm()`). The function should take as inputs:

- a numeric vector x
- the known population variance `var`
- the confidence level `conf.level`

and return the lower and upper bounds of the confidence interval as a numeric vector.

b. Write a function `ci_variance` that calculates a $(1-\alpha)$ confidence interval for the **variance** of a normal distribution. The function should take as input a numeric vector x and a confidence level, and return the CI bounds as a numeric vector.

c. Adapt your function `ci_mean_known_var` to calculate a $(1-\alpha)$ confidence interval when the variance is not known - this can be done by simply replacing the standard normal quantiles by the quantiles of the t distribution with the appropriate number of degrees-of-freedom (check function `qt()`). Save this new function as `ci_mean_t()`. This new function should take as inputs:

- a numeric vector x
- the confidence level `conf.level`

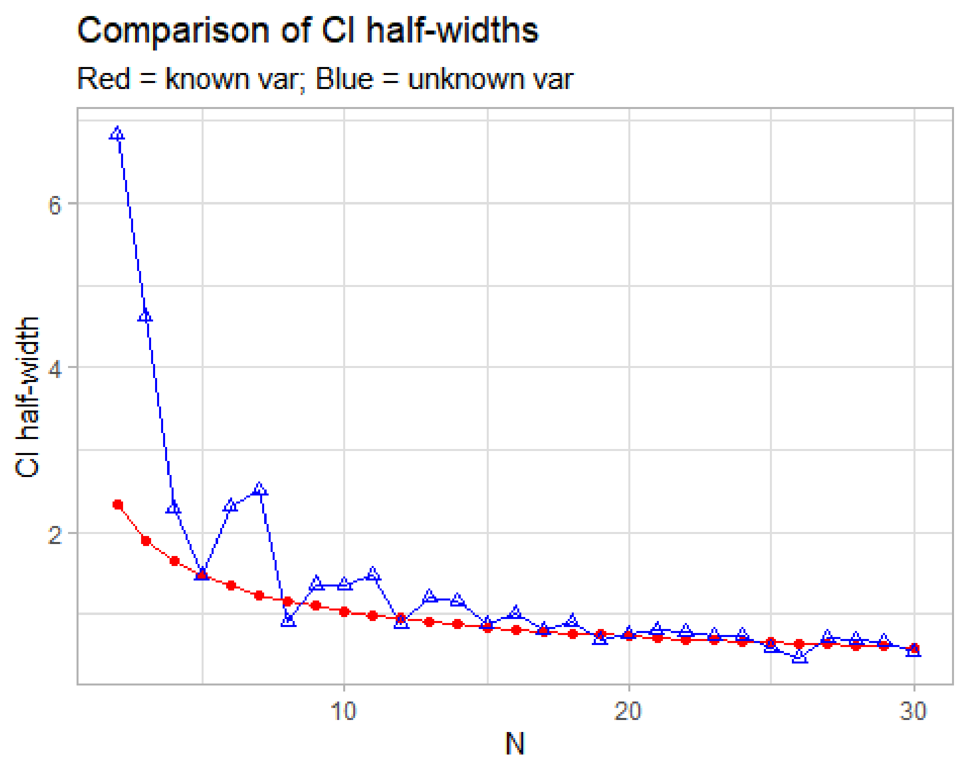
and return the lower and upper bounds of the confidence interval as a numeric vector.

d. Given the properties of the t distribution, we expect that the intervals derived when the variance is unknown (i.e., using the t-quantiles) should approach the case of the known variance (using the z-quantiles). You will now build a visualisation of this phenomenon. For that, you will build a dataframe with the following columns:

- `N`: the sample size
- `CI.hw.norm`: the half-width of a CI derived using `ci_mean_known_var`
- `CI.hw.t`: the half-width of a CI derived using `ci_mean_t`

You should vary `N` from 2 to 30. For each value of `N`, generate a single vector of observations from a standard normal distribution (using `rnorm()`), calculate two $CI_{.95}$ for that sample (using your two functions) and calculate the half-width of the CIs derived for each case.

Once you have your dataframe, use ggplot2 to plot the half widths of each case vs. the sample size. Your plot should look somewhat like this:



Can you explain why the known variance line is smooth while the unknown variance one is jagged?

Discussion: Bootstrap confidence intervals

The **bootstrap** is a resampling-based method for approximating the sampling distribution of a statistic. It works by resampling (with replacement) from the observed data and recalculating the statistic of interest many times.

Quoting Jeremy Orloff and Jonathan Bloom's text on Bootstrap confidence intervals,

The empirical bootstrap is a statistical technique popularized by Bradley Efron in 1979. Though remarkably simple to implement, the bootstrap would not be feasible without modern computing power. The key idea is to perform computations on the data itself to estimate the variation of statistics that are themselves computed from the same data. Such techniques existed before 1979, but Efron widened their applicability and demonstrated how to implement the bootstrap effectively using computers. He also coined the term 'bootstrap'.

The empirical bootstrap is also known as the ***nonparametric bootstrap****.*

The original text quoted above is a good introduction to the bootstrap. Consider reading it after you finish this lab.

One of the most useful applications of the bootstrap lies in estimating the variation of point estimates. This can be used either to compute their standard errors - which can then be used in the large-sample approximation of general CIs, $\hat{\theta} \pm z_{\alpha/2} se_{\hat{\theta}}$; or even directly to estimate the CI itself.

One of the ways to estimate a $(1 - \alpha)$ CI using the bootstrap is to build a *bootstrap sample*, which approximates the sampling distribution of the statistic, and then take the $\alpha/2$ and $1 - \alpha/2$ quantiles of the bootstrap sample. A short example code for a simple bootstrap CI on the mean is shown below:

```
set.seed(123)

# simulate some data
x <- rnorm(50, mean = 5, sd = 2)

R <- 2000 # 2000 bootstrap resamples

# Generate R resamplings of the data in x, and calculate the mean for each
case
boot_means <- replicate(R, mean(sample(x, size = length(x), replace = TRUE)))

# Return the 95% confidence interval on the mean
quantile(boot_means, c(0.025, 0.975))

##     2.5%    97.5%
## 4.576968 5.563031
```

Notice a few specific aspects of this method:

- The size of each resample is the same as the original sample. This is important since we are trying to estimate the variability of the sample statistic, which is conditional on the specific sample size (just think of the sample standard error of the mean, $s/\sqrt{N}$.

- The sample is done **with replacement**. This is done because one of the underlying ideas in the bootstrap is that the *empirical distribution* estimated from your available data is a good enough representation of your actual *populational distribution*. Sampling with replacement from your available data is equivalent to generating new samples based on the *empirical distribution*.

Again paraphrasing from Orloff and Bloom, we can summarise the bootstrap principle as:

- Let $S = x_1, x_2, \ldots, x_n$ be a sample drawn from a distribution F.
- Let u be a statistic computed from the sample.

- Let $F^*(S)$ be the empirical distribution of the sada, computed from the sample.
- Suppose a bootstrap sample $S^* = x_1^*, x_2^*, \ldots, x_n^*$, composed by randomly drawing (with replacement) n items from S.
- Let u^* be a statistic computed from the bootstrap sample.

The bootstrap principle states that:

- F^* is approximately equal to F
- u is approximated by u^*
- The variability of u is approximated by the variability of $u\ *$.

The Bootstrap method has several advantages over parametric methods:

- It is *non-parametric*, i.e. does not require strong distributional assumptions e.g. Gaussian data.
- It applies to any statistical estimator e.g. median, trimmed mean etc.

The Bootstrap method also some has drawbacks:

- It is *computationally expensive* (less of a concern with modern hardware, but still relevant if real-time estimates are needed).

- Parametric methods typically outperform the Bootstrap methods when the assumptions hold (especially with small sample sizes). This is a common aspect of statistical methods (and machine learning) - commonly, you gain flexibility at the price of precision. The more problem-specific information you can add to your analysis / inference / modelling, the more precise these tend to be.

Finally, it is important to be conscious of what resampling **cannot** do: it cannot improve your point estimate. The bootstrap resamples are estimating the variability of a statistic from a sample - not the real value of the parameter.

Q4: Bootstrap confidence intervals

For this question, you will work with data related to pharmaceutical drug spending in different countries. This dataset comes from a combination of economic data from the OECD and populational data from DataHub. You can download it from the Blackboard page, as file - `pharma_drug_spending.csv`.

The data consists of useful information about percent of health spending, percent of GDP and US dollars per capita for specific countries. The *data schema* is as follows:

name	type	description
LOCATION	string	Country code
TIME	year	Year

name	type	description
PC_HEALTHXP	number	% of Health spending used for pharma drugs
PC_GDP	number	% of GDP used for pharma drugs
USD_CAP	number	in USD per capita (using economy-wide PPPs)
TOTAL_SPEND	number	Total spending in millions

Read the data from file `pharma_drug_spending.csv` into a data frame called `oecd.pharma` (you can use base R's `read.csv()` or `read_csv()` from package `readr`).

a. Write a function called `median_boot_ci` to estimate the nonparametric bootstrap confidence interval on the median, for an arbitrary confidence level. The function should receive as input parameters:

- `x`, a vector of data
- `R`, the number of bootstrap resamples
- `conf`, the confidence level

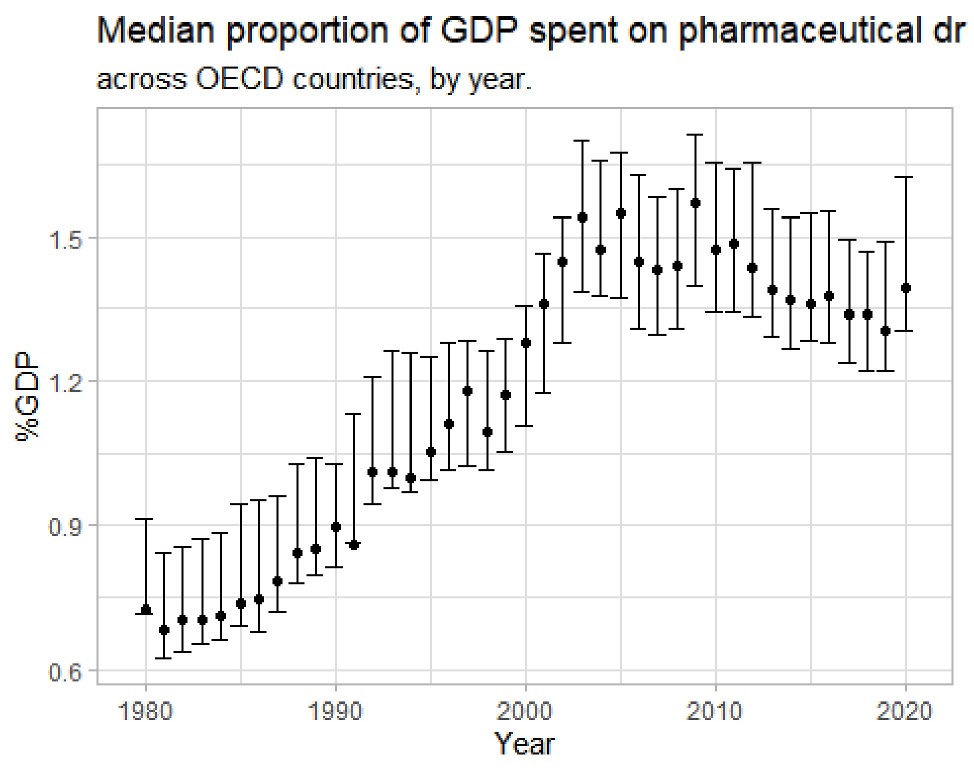
And return the estimated CI as a numeric vector.

Note: You can adapt the example code provided earlier for the bootstrap CI on the mean.

b. Isolate the values of `USD_CAP` for the year 2022 from the `oecd.pharma` dataset, and use your function to calculate the $CI_{.95}$ on the median. Store the confidence interval as a vector variable called `med_usd_cap_ci`

c. *Challenge*: You will now use your function `median_boot_ci` to calculate and plot the point estimate and the 95% confidence interval on the median % of GDP used for pharma drugs in OECD countries by year, between 1980 and 2020. Use `R = 2000` bootstrap simulations for each case.

To calculate that, you may need functions `filter()`, `group_by()`, `summarise()` and `unnest_wider()` (from package `tidyr`). Your final plot should look somewhat like this:


Median proportion of GDP spent on pharmaceutical dr
across OECD countries, by year.
%GDP
1.5
1.2
0.9
0.6
1980
1990
2000
2010
2020
Year