Lab1 Start with R

GHP 501 Modeling for Health System Analysis & Priority Setting (Spring1 2024)

Yizhi (Diego) Liang

2024-01-21

The introduction of R and some common techniques that will be used in the GHP 501. Major materials comes from the R tutorial by Stephen Pettigrew, and further developed by Boshen Jiao, Dorit Stein, Sarah Bolongaita, Mayya Komisarchik, Anton Strezhnev, Stéphane Verguet, Allison Portnoy, Xiaoxiao Jiang Kwete, and Jimmy Potter.

Contents

1 R learning resources

- The Bible for R: R for Data Science (2nd edition)
- Useful cheat sheets: Posit (RStudio) Cheat Sheets
- Practical tutorial online, such as R for applied epidemiology and public health | The Epidemiologist R Handbook (epirhandbook.com)
    - It provides a free interactive platform: Interactive R Tutorials (appliedepi.org)
- `learnr` package

2 Package loaded

```r
knitr::opts_chunk$set(collapse = TRUE, # the code and its output are shown together
                      warning = FALSE, # suppresses warning messages
                      error = FALSE, # suppresses error messages
                      message = FALSE # suppresses messages generated by the code
                      )

# install "pacman" package if not already installed
if (!require("pacman")) install.packages("pacman")
```

```
Loading required package: pacman
```

```r
# "library" all packages needed at one time
p_load(tidyverse, rio, here)
```

3 R and R studio

R is a **programming language** used for statistical computing, while **R Studio is an environment** that uses the R language to develop statistical programs. In R, you can write a program and run the code independently of any other computer program through your computer's terminal. R Studio, however, must be used alongside R in order to properly function. Often referred to as an IDE, or an integrated development environment, R Studio allows users to develop and edit programs in R by supporting a large number of statistical packages, higher quality graphics, and the ability to manage your work space.

R and R Studio are not separate versions of the same program, and cannot be substituted for one another. R may be used without R Studio, but R Studio may not be used without R.

Other common R IDE is visual studio code, DataSpell by Jetbrains.

4 Getting help in R

- get the documentation within the R console

```
?mean
??mean
```

- Cheatsheet in Rstudio
- Vignettes of R packages / functions
- Google it!

5 R studio projects

R studio projects are very powerful to create a specific work space for your specific work, each with their own working directory, workspace, history, and source documents. It is highly recommended to create one for each project or coursework you will use R, also connecting with Git/Github for version control.

Let's create one for GHP501!

6 Basic R operations

6.1 Basic calculator

```r
2 + 2 # addition
## [1] 4
9 - 4 # subtraction
## [1] 5
6 * 12 # multiplication
## [1] 72
12 / 3 # division
## [1] 4
2^8 # exponentiation
## [1] 256
log(12) # the log function with the base e, "natural log"
## [1] 2.484907
exp(2) # e^2
## [1] 7.389056
```

6.2 Functions

Functions are operations that are more complex than your basic math operations. We've already seen some of R's basic functions, such as `mean()` and `log()`, but there are many more built-in functions in R. Trying searching for these using the `?` and `??` tools or Google. In addition, we can write our own functions in R very easily.

The general syntax to use functions in R is: `function.name(argument)`, where the function name varies according to what the function is, and the argument contains the values or "object" to which we want to apply the function. In many cases, you can create the object right inside your function statement.

```r
a_vector <- c(2, 3, 10, 12)
mean(a_vector)
## [1] 6.75
```

Let's create a function by ourselves!

```r
add_function <- function(number_1, number_2) {
  sum = number_1 + number_2
  return(sum)
}

add_function(1, 2)
## [1] 3
```

6.3 Useful data management skills

6.3.1 working directory and "here" package

```r
# in the base R way
getwd() # check the current working directory
# change my working directory
setwd("/Users/yizhi_liang/Library/CloudStorage/OneDrive-HarvardUniversity/OB/hsph/A-Course
getwd() # the new working directory

# use "here" package
here() # check the current working directory
```

```r
dr_here()
```

`here("folder1", "folder2",…,"filename.csv")` it searches your file starting from the "root" directory (NOT WORKING DIRECTORY).

6.3.2 Import data using "rio" package

We want to load data `lab1.csv` using a powerful package `rio` with `here` package locating. It is in the folder "01_data".

```r
# "import" package and "here" package
lab1 <- import(here("01_data", "lab1.csv"))
```

```r
## check the first 5 rows
head(lab1)
```

country	country_code	gdp_2016	u5_2016	le_2016	gdp_2018	u5_2018	le_2018
Afghanistan	AFG	547.2281	67.5	63.763	520.8966	62.3	NA
Albania	ALB	4124.1089	9.3	78.194	5268.8485	8.8	NA
Algeria	DZA	3946.4214	24.5	76.298	4114.7151	23.5	NA
American Samoa	ASM	11696.9556	NA	NA	11466.6907	NA	NA
Andorra	AND	37224.1089	3.1	NA	42029.7627	2.9	NA
Angola	AGO	3506.0729	84.0	59.925	3432.3857	77.2	NA

The `lab1` data is extracted from the World Bank's World Development Indicators website and contains indicators of:

- country: Country name
- country_code: Country codes used by the World Bank
- gdp_2016: GDP per capita (current US$) (2016)
- u5_2016: Under-five mortality (per 1,000 live births) (2016)
- le_2016: Life expectancy at birth (2016)
- gdp_2018: GDP per capita (current US$) (2018)
- u5_2018: Under-five mortality (per 1,000 live births) (2018)
- le_2018: Life expectancy at birth (2018)

6.3.3 Glance at the data

```r
# base R: "str" function
str(lab1)
## 'data.frame':    217 obs. of  8 variables:
##  $ country     : chr  "Afghanistan" "Albania" "Algeria" "American Samoa" ...
##  $ country_code: chr  "AFG" "ALB" "DZA" "ASM" ...
##  $ gdp_2016    : num  547 4124 3946 11697 37224 ...
##  $ u5_2016     : num  67.5 9.3 24.5 NA 3.1 84 7 10.9 13.7 NA ...
##  $ le_2016     : num  63.8 78.2 76.3 NA NA ...
##  $ gdp_2018    : num  521 5269 4115 11467 42030 ...
##  $ u5_2018     : num  62.3 8.8 23.5 NA 2.9 77.2 6.4 9.9 12.4 NA ...
## $ le_2018     : logi  NA NA NA NA NA NA ...

# tidyverse: "glimpse" function
glimpse(lab1)
## Rows: 217
## Columns: 8
## $ country      <chr> "Afghanistan", "Albania", "Algeria", "American Samoa", "A~
## $ country_code <chr> "AFG", "ALB", "DZA", "ASM", "AND", "AGO", "ATG", "ARG", "~
## $ gdp_2016     <dbl> 547.2281, 4124.1089, 3946.4214, 11696.9556, 37224.1089, 3~
## $ u5_2016      <dbl> 67.5, 9.3, 24.5, NA, 3.1, 84.0, 7.0, 10.9, 13.7, NA, 3.8,~
## $ le_2016      <dbl> 63.76300, 78.19400, 76.29800, NA, NA, 59.92500, 76.61700,~
## $ gdp_2018     <dbl> 520.8966, 5268.8485, 4114.7151, 11466.6907, 42029.7627, 3~
## $ u5_2018      <dbl> 62.3, 8.8, 23.5, NA, 2.9, 77.2, 6.4, 9.9, 12.4, NA, 3.7, ~
## $ le_2018      <lgl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, N~

# other packages
## "skimr" package
if (!require("skimr")) install.packages("skimr")
skimr::skim(lab1)
```

Table 2: Data summary

Name	lab1
Number of rows	217
Number of columns	8
Column type frequency:	
character	2
logical	1
numeric	5
Group variables	None

Variable type: character

skim_variable	n_missing	complete_rate	min	max	empty	n_unique	whitespace
country	0	1	4	30	0	217	0
country_code	0	1	3	3	0	217	0

Variable type: logical

skim_variable	n_missing	complete_rate	mean	count
le_2018	217	0	NaN	:

Variable type: numeric

skim_variable	n_missing	complete_rate	mean	sd	p0	p25	p50	p75	p100	hist
gdp_2016	13	0.94	16202.29	24441.87	282.19	2191.09	5813.80	19636.35	169915.80	
u5_2016	24	0.89	30.11	30.30	2.00	7.70	17.40	47.70	129.40	
le_2016	18	0.92	72.30	7.78	51.59	67.13	73.84	77.91	84.23	
gdp_2018	22	0.90	15898.80	23585.24	271.75	2142.34	6289.94	19214.41	185741.28	
u5_2018	24	0.89	28.17	28.42	1.70	7.10	16.60	43.60	121.50	

```r
## "summarytools" package
if (!require("summarytools")) install.packages("summarytools")
### Note: You may find we cannot use this function (dfSummary) to glance at this data
### Why? Try to debug it!
# dfSummary(lab1)
```

7 Tidyverse!

```
# p_load(tidyverse)
# library(tidyverse)
?tidyverse
```

7.1 Data Frames or "tibble"

The "standard" format of a data frame or a tibble should be: each columns is a variable, each rows is a observation, and each cell is the value.

```
class(lab1)
## [1] "data.frame"
head(lab1)
```

country	country_code	gdp_2016	u5_2016	le_2016	gdp_2018	u5_2018	le_2018
Afghanistan	AFG	547.2281	67.5	63.763	520.8966	62.3	NA
Albania	ALB	4124.1089	9.3	78.194	5268.8485	8.8	NA
Algeria	DZA	3946.4214	24.5	76.298	4114.7151	23.5	NA
American Samoa	ASM	11696.9556	NA	NA	11466.6907	NA	NA
Andorra	AND	37224.1089	3.1	NA	42029.7627	2.9	NA
Angola	AGO	3506.0729	84.0	59.925	3432.3857	77.2	NA

```
lab1_tibble = lab1 |> as_tibble()
class(lab1_tibble)
## [1] "tbl_df"     "tbl"        "data.frame"
head(lab1_tibble)
```

country	country_code	gdp_2016	u5_2016	le_2016	gdp_2018	u5_2018	le_2018
Afghanistan	AFG	547.2281	67.5	63.763	520.8966	62.3	NA
Albania	ALB	4124.1089	9.3	78.194	5268.8485	8.8	NA
Algeria	DZA	3946.4214	24.5	76.298	4114.7151	23.5	NA
American Samoa	ASM	11696.9556	NA	NA	11466.6907	NA	NA
Andorra	AND	37224.1089	3.1	NA	42029.7627	2.9	NA
Angola	AGO	3506.0729	84.0	59.925	3432.3857	77.2	NA

7.2 Loop, "Apply" family, "Map" family from "Purrr" package

```
df = tibble(
  q = 1:5, # five quintiles
  prevalence = seq(0.1, 0.5, by = 0.1) # a vector from 0.1 to 0.5, each gap with 0.2
  )
df
```

q	prevalence
1	0.1
2	0.2
3	0.3
4	0.4
5	0.5

For example, for each quinitle, there are 1,000 people, and the prevalence of a disease is 0.1. How to simulate a 1,000 people to reflect this disease burden?

```
# to keep replicatable, set a random seed
set.seed(123)

# a Binomial distribution, 1000 population, each person has 1 trial, the "success" probabi
df_q1 = rbinom(1000, 1, 0.5)
length(df_q1)
## [1] 1000
sum(df_q1)
## [1] 493
```

We can do this process for each quintile one by one, but what could we do to be more efficient?

1. "FOR LOOP"

```
# Loop
## a container to store the 1,000 population
pop_list_loop = vector(mode = "list", length = 5) # list()
## a vector to store the number of patients
patients_num_loop = vector(length = 5)

for (i in 1:5) {
  prevalence = df$prevalence
  pop_list_loop[[i]] = rbinom(1000, 1, prevalence[i])
  patients_num_loop[i] = sum(pop_list_loop[[i]])
}

str(pop_list_loop)
## List of 5
##  $ : int [1:1000] 0 0 0 0 0 0 0 0 0 0 ...
##  $ : int [1:1000] 0 0 0 0 0 0 0 0 0 0 ...
##  $ : int [1:1000] 0 1 0 0 0 0 1 0 1 1 ...
##  $ : int [1:1000] 0 1 0 1 0 0 1 0 0 1 ...
##  $ : int [1:1000] 0 1 1 1 0 0 0 1 1 0 ...
patients_num_loop
## [1]  85 197 306 384 493
```

2. "APPLY FAMILY"

```
# ?apply

pop_list_app = lapply(df$prevalence, function(x) rbinom(1000, 1, prob = x))
patients_num_app = sapply(pop_list_app, \(x) sum(x))

str(pop_list_app)
## List of 5
##  $ : int [1:1000] 0 0 0 0 0 0 0 0 1 0 ...
##  $ : int [1:1000] 1 0 0 1 0 1 0 0 0 0 ...
##  $ : int [1:1000] 0 0 1 0 1 1 0 0 0 0 ...
##  $ : int [1:1000] 1 1 0 0 1 0 0 0 1 0 ...
##  $ : int [1:1000] 0 0 1 0 0 0 1 1 1 0 ...
patients_num_app
## [1]  88 198 278 402 496
```

3. (Recommend!) "MAP" from "Purrr" package

```
if (!require("purrr")) install.packages("purrr")
library(purrr)
df |>
  mutate(
    # to create a new column, a list column to store 1000 population
    pop = map(prevalence, ~ {set.seed(123); rbinom(1000, 1, prob = .x)}),
    # create a new column to store the patients number
    patient_num = map_dbl(pop, ~ sum(.x))
    ) |>
  str()
## tibble [5 x 4] (S3: tbl_df/tbl/data.frame)
##  $ q          : int [1:5] 1 2 3 4 5
##  $ prevalence : num [1:5] 0.1 0.2 0.3 0.4 0.5
##  $ pop        :List of 5
##   ..$ : int [1:1000] 0 0 0 0 1 0 0 0 0 0 ...
##   ..$ : int [1:1000] 0 0 0 1 1 0 0 1 0 0 ...
##   ..$ : int [1:1000] 0 1 0 1 1 0 0 1 0 0 ...
##   ..$ : int [1:1000] 0 1 0 1 1 0 0 1 0 0 ...
##   ..$ : int [1:1000] 0 1 0 1 1 0 1 1 1 0 ...
##  $ patient_num: num [1:5] 92 198 295 393 493
```

7.3 Data manipulation in R

Let's back to the loaded data set, `lab1`.

```
glimpse(lab1)
## Rows: 217
## Columns: 8
## $ country      <chr> "Afghanistan", "Albania", "Algeria", "American Samoa", "A~
## $ country_code <chr> "AFG", "ALB", "DZA", "ASM", "AND", "AGO", "ATG", "ARG", "~
## $ gdp_2016     <dbl> 547.2281, 4124.1089, 3946.4214, 11696.9556, 37224.1089, 3~
## $ u5_2016      <dbl> 67.5, 9.3, 24.5, NA, 3.1, 84.0, 7.0, 10.9, 13.7, NA, 3.8,~
## $ le_2016      <dbl> 63.76300, 78.19400, 76.29800, NA, NA, 59.92500, 76.61700,~
## $ gdp_2018     <dbl> 520.8966, 5268.8485, 4114.7151, 11466.6907, 42029.7627, 3~
## $ u5_2018      <dbl> 62.3, 8.8, 23.5, NA, 2.9, 77.2, 6.4, 9.9, 12.4, NA, 3.7, ~
## $ le_2018      <lgl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, N~
```

7.3.1 Extracting a variable (column), an observation (row), or a cell value

- A variable (column)

```
# base R
head(lab1["country_code"])
```

country_code
AFG
ALB
DZA

country_code
ASM
AND
AGO

```r
class(lab1["country_code"]) ## return a data frame with only ONE column
## [1] "data.frame"

# dplyr(tidyverse)
lab1 |> select(country_code) |> head() ## return a data frame with only ONE column
```

country_code
AFG
ALB
DZA
ASM
AND
AGO

```r
# base R
head(lab1$country_code)
## [1] "AFG" "ALB" "DZA" "ASM" "AND" "AGO"
class(lab1$country_code) ## reutn a "character" vector
## [1] "character"

# dplyr (tidyverse)
lab1 |> pull(country_code) |> head()
## [1] "AFG" "ALB" "DZA" "ASM" "AND" "AGO"
class(lab1 |> pull(country_code)) ## identical to lab1$country
## [1] "character"

# check if they are identical
identical(lab1$country_code, lab1 |> pull(country_code))
## [1] TRUE
```

- An observation

```r
# get observations with gdp_2016 greater than 10,000

## base R
head(lab1[lab1$gdp_2016 > 10000, ])
```

	country	country_code	gdp_2016	u5_2016	le_2016	gdp_2018	u5_2018	le_2018
4	American Samoa	ASM	11696.96	NA	NA	11466.69	NA	NA
5	Andorra	AND	37224.11	3.1	NA	42029.76	2.9	NA
7	Antigua and Barbuda	ATG	15197.62	7.0	76.61700	16726.98	6.4	NA
8	Argentina	ARG	12790.24	10.9	76.22100	11683.95	9.9	NA
10	Aruba	ABW	25239.60	NA	75.86800	NA	NA	NA
11	Australia	AUS	49971.13	3.8	82.44878	57373.69	3.7	NA

```r
## dplyr (tidyverse)
lab1 |> filter(gdp_2016 > 10000) |> head()
```

country	country_code	gdp_2016	u5_2016	le_2016	gdp_2018	u5_2018	le_2018
American Samoa	ASM	11696.96	NA	NA	11466.69	NA	NA
Andorra	AND	37224.11	3.1	NA	42029.76	2.9	NA
Antigua and Barbuda	ATG	15197.62	7.0	76.61700	16726.98	6.4	NA
Argentina	ARG	12790.24	10.9	76.22100	11683.95	9.9	NA
Aruba	ABW	25239.60	NA	75.86800	NA	NA	NA
Australia	AUS	49971.13	3.8	82.44878	57373.69	3.7	NA

```r
## check if they are identical
identical(lab1[lab1$gdp_2016 > 10000, ], lab1 |> filter(gdp_2016 > 10000))
## [1] FALSE
```

You may find they did not return the identical result - why?

```r
# check the gdp_2016
summary(lab1$gdp_2016) # 13 Missing values!
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.     NA's
##    282.2   2191.1   5813.8  16202.3  19636.3 169915.8       13
## using base R, NAs were kept
summary(lab1[lab1$gdp_2016 > 10000, ]$gdp_2016)
##     Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's 
##    10821   17527   30629   37901   46350  169916      13 

## using tidyverse, NAs were exlcuded
summary(lab1 |> filter(gdp_2016 > 10000) |> pull(gdp_2016))
##     Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
##    10821   17527   30629   37901   46350  169916 
```

- A cell value

```r
# get the US's gdp in 2016

## base R
lab1[lab1$country == "United States", "gdp_2016"] # return a cell value
## [1] 57904.2

## tidyverse
lab1 |>
  filter(country == "United States") |>
  pull(gdp_2016)
## [1] 57904.2
```

7.3.2 Reorder the rows

```r
# remain as a data frame (or tibble), sort by gdp_2016 from min to max
lab1 |>
  arrange(gdp_2016) |>
  head()
```

country	country_code	gdp_2016	u5_2016	le_2016	gdp_2018	u5_2018	le_2018
Burundi	BDI	282.1931	63.8	60.528	271.7520	58.5	NA
Somalia	SOM	295.9679	129.4	56.324	314.5442	121.5	NA
Malawi	MWI	315.7773	55.7	62.681	389.3980	49.7	NA
Niger	NER	362.1311	90.3	61.137	413.9803	83.7	NA
Central African Republic	CAF	401.9182	123.9	51.593	475.7213	116.5	NA
Mozambique	MOZ	428.9265	78.1	58.309	498.9572	73.2	NA

7.3.3 Create new variables (columns) or change existing ones

```r
# create a new variable as the average of gdp_2016 and gdp_2018
lab1 |>
  mutate(gdp_avg = (gdp_2016 + gdp_2018)/2) |>
  # keep country, gdp information only
  select(country_code, gdp_2016, gdp_2018,
         # change the column name by the way in the select()
         ## You can also use rename() outside select()
         gdp_avg_16_18 = gdp_avg) |>
  # calculate the cumulative sum of a the average gdp
  arrange(gdp_avg_16_18) |>
  mutate(cum_gdp = cumsum(gdp_avg_16_18)) |>
  head()
```

country_code	gdp_2016	gdp_2018	gdp_avg_16_18	cum_gdp
BDI	282.1931	271.7520	276.9726	276.9726
SOM	295.9679	314.5442	305.2560	582.2286
MWI	315.7773	389.3980	352.5877	934.8163
NER	362.1311	413.9803	388.0557	1322.8720
CAF	401.9182	475.7213	438.8197	1761.6917
MOZ	428.9265	498.9572	463.9419	2225.6336

```r
# let gdp_2016 and gdp_2018 as integers
lab1 |>
  mutate(
    gdp_2016 = round(gdp_2016),
    gdp_2018 = round(gdp_2018)
  ) |>
  select(country_code,
         # select variables staring with "gdp_"
         starts_with("gdp_")
         ) |>
  head()
```

country_code	gdp_2016	gdp_2018
AFG	547	521
ALB	4124	5269
DZA	3946	4115

country_code	gdp_2016	gdp_2018
ASM	11697	11467
AND	37224	42030
AGO	3506	3432

```r
# or, we can use across()
lab1 |>
  mutate(across(
    .cols = starts_with("gdp_"), # .cols = where(is.numeric)
    .fns = ~ round(.x)
  )) |>
  select(country_code, starts_with("gdp_")) |>
  head()
```

country_code	gdp_2016	gdp_2018
AFG	547	521
ALB	4124	5269
DZA	3946	4115
ASM	11697	11467
AND	37224	42030
AGO	3506	3432

7.3.4 Collapse values down to the summary statistics

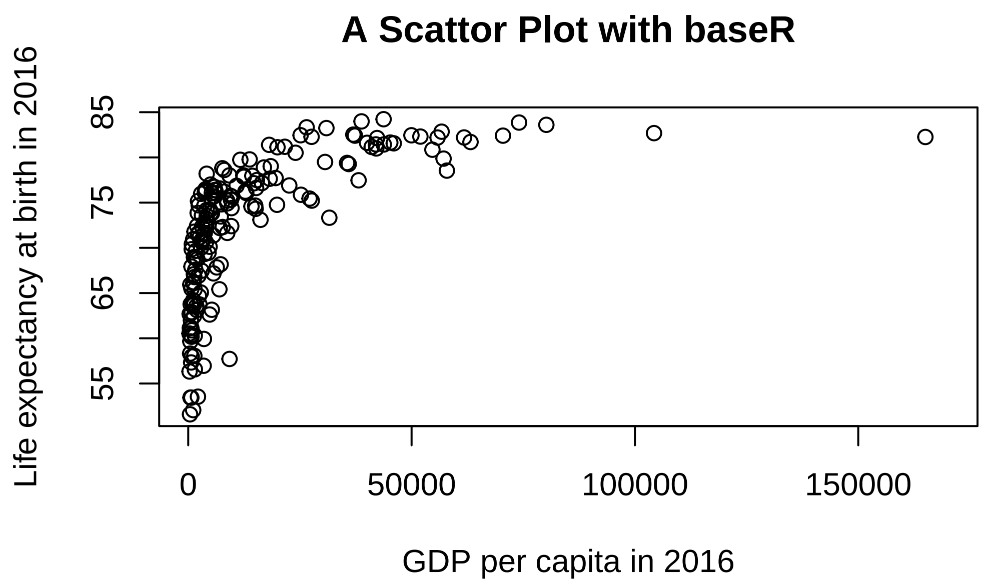
```r
# first, let's mcreate the groups by five-quintiles of "gdp_2016"
# store as a new tibble (data frame)
lab1_group = lab1 |>
  mutate(group = ntile(gdp_2016, 5))

# calculate the mean, 95% range of the gdp_2018, by group
lab1_group |>
  group_by(group) |>
  summarise(
    # na.rm = TRUE => ignore the NAs
    mean_gdp_2018 = mean(gdp_2018, na.rm = TRUE),
    lower_gdp_2018 = quantile(gdp_2018, probs = 0.025, na.rm = TRUE),
    upper_gdp_2018 = quantile(gdp_2018, probs = 0.975, na.rm = TRUE)
  )
```

group	mean_gdp_2018	lower_gdp_2018	upper_gdp_2018
1	969.8693	314.5442	1715.531
2	3078.0741	1532.3717	4364.016
3	7149.2893	4240.3293	10354.572
4	18406.5454	10239.4704	30402.328
5	57941.4274	31593.6288	130460.166
NA	NaN	NA	NA

8 Basic Plotting

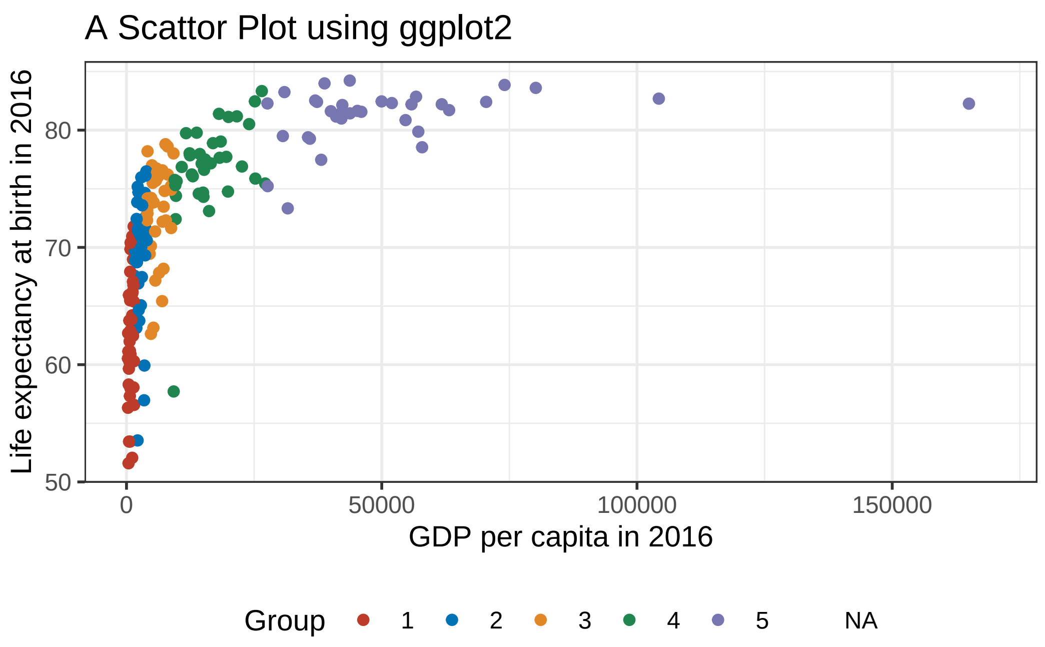
Let's plot a scatter plot to show the life expectancy in 2016 on the y-axis and the GDP per capita in 2016 on the x-axis.

```r
plot(
  x = lab1_group$gdp_2016,
  y = lab1_group$le_2016,
  xlab = "GDP per capita in 2016",
  ylab = "Life expectancy at birth in 2016",
  main = "A Scattor Plot with baseR"
)
```



Let's use "ggplot2" package from tidyverse!

```r
lab1_group |>
  ggplot(aes(x = gdp_2016, y = le_2016)) +
  # color by group
  geom_point(aes(color = factor(group))) +
  # set a good-looking theme
  theme_bw() +
  # choose a nice palettee for colors
  ggsci::scale_color_nejm() +
  labs(
    x = "GDP per capita in 2016",
    y = "Life expectancy at birth in 2016",
    title = "A Scattor Plot using ggplot2",
    color = "Group"
  ) +
  # put the color legend at the bottom
  theme(legend.position = "bottom") +
  # let the color legend to be in a row
  guides(color = guide_legend(nrow = 1, byrow = TRUE))
```



9 Fun topic: R with AI tools

Github Copilot is an “AI pair programmer that offers autocomplete-style suggestions as you code”. GitHub Copilot is available as an opt-in integration with RStudio.

You can find the information about prerequisites, setup from the Github Copilot user guide.