BTS 510 Lab 1

```r
set.seed(12345)
```

> **Seed**
>
> This first code chunk usually has more in it. See notes below. This code sets a random number seed so that if we do anything that requires a random number (e.g., create data with a random number generating function), it will produce the **same values** every time.

> **Where is the code???**
>
> - At the top of the page, in the upper right, there is a button that reads "</> Code"
>   - Click on that and a window with the code that made this page will pop up
>   - Click on the little notepad icon in the upper right to copy the entire document
>   - Sometimes this doesn't work so you need to select and copy the text

Learning objectives

- Start using **R** and **Rstudio**
- Start using **Quarto** documents as a **reproducible** method to present analyses along with **narrative text**

A few miscellaneous things about R and Quarto

About R

- R is **case sensitive**
  - `mean` is not the same as `Mean`
  - The object `b` is not the same as the object `B`

- Extra returns don't matter
  - Extra spaces generally don't matter either
  - I often add extra spaces, tabs, and returns to make the code more readable
- The assignment operator is `<-`
  - "Put whatever is on the right side in the object named on the left side"
- Sometimes, equal isn't equal
  - If you're talking about *assigning* something to a value, you can use `=`
  - If you want to *compare* two values and make a decision based on it, you use `==`
- There isn't a standard, widely-used naming convention for variables, so you should pick something and try to be consistent
  - In general, variable names must start with a letter – no numbers or special characters
  - There are some options for multi-word variable naming
    - `variablename`: Just smush it all together
    - `variableName` or `VariableName`: Camel case
    - `variable_name` or `Variable_Name`: Snake case
    - `variable-name` or `Variable-Name`: Kebab case
  - In general, make the variable names **as short as possible** while keeping each name unique

About Quarto

- Look at the top of the Quarto file. The section with `---` at the top and bottom. This is the **YAML** (pronounced "yam-ull"). It provides options for the overall document.
  - `title:` is pretty obvious
  - `format:` tells R which output format to produce when you knit the document together
  - This is an `html` document, but you can specify *different* or *multiple* output formats, like both HTML and PDF
    - There are additional options that are specific to this format
    - `embed-resources: true` (older version: `self-contained: true`) and `self-contained-math: true` make sure that any additional files that are created, like figures, are part of the HTML file. If I didn't include this and sent you to the HTML file, the figures etc. would be missing.
    - `html-math-method: katex` is an option about the equations and math text. It's probably not necessary, but I include it anyway.
    - `number-sections: true` and `toc: true` produce the table of contents on the right side of this page and number the sections and sub-sections (remember headings and organization?) to make it easier to navigate

    * `code-tools: true` adds the `</> Code` button at the top that shows the code that made this page. You can copy the code (using the little "Copy to clipboard" icon in the upper right) and paste it right into R
    * `code-block-bg: true` and `code-block-border-left: "#31BAE9"` make the code chunks a little more noticeable – they blended into the background too much otherwise.
    * The YAML is one of the few places that **spacing matters**. Note how each sub-option is indented more than it's over-option. If you don't get this right, the file won't run
    * Read more about **YAML** options here

- You can **label** each code chunk in the document
  - This is a very smart thing to do
    * I don't do it consistently :(but I'm trying
  - If you have an error, R will tell you which chunk has the problem
    * If the chunks are named, **it will use the name** so it's easy to find
    * If they're not named, it will tell you something like "Unnamed chunk 52", which is *way less helpful*
  - Name your chunks with `#| label: name-of-this-chunk`
    * See the setup chunk above – that's just its name, "setup"

- There are other options for code chunks. Here are some useful ones
  - `echo: true`: Repeats the code in the output
  - `warning: false`: Suppress any warnings from R (**not recommended**)
  - `eval: false`: Don't run this code, just print it (useful for teaching)
  - For any of these, you can swap from `true` to `false` or vice versa and get the opposite effect
  - `fig-cap: "My figure caption is cool"`: Adds a caption to the figure produced in that chunk – useful for accessibility
  - More info here and here

- You may be used to running small chunks of code individually, even in a large file of R code
  - Quarto (and markdown) are more about **rendering** the **whole document** into an external (html or pdf or doc) file
  - Click the "Render" button at the top of the window to render
    * Depending on the size of the file and what it needs to do, this can take a few minutes
    * The console shows a progress bar with percentage complete

Installing packages

You must **install** a new package the first time you use it

```
#install.packages(gapminder)
```

You only need to do this once. I have **commented out** this part of the code by putting `#` at the start of the line because I already have the package installed. If you try to install a package that is already installed, *you will probably get an error*.

> **i** Note
>
> In my own work, I usually install packages via the Rstudio graphical user interface (GUI). I'm doing it this way for pedagogical reasons.

Alternatively, if you're using Rstudio, you can quickly install packages in the "Packages" pane using the "Install" button, and *exclude this code*.

Loading packages

You must **load** or `library()` a package each time you use it. This means that you need to load it at the top of each markdown or quarto file you write.

```
library(gapminder)
```

This allows us to use the functions in these packages. If you're loading multiple packages, just list each on it's own line.

> **i** Note
>
> In my own work, I usually load all packages in the "setup" chunk at the top of the document. I'm doing it this way for pedagogical reasons.

The **gapminder** package is a *data package*. It provides a subset of the data available at gapminder.org. You may have heard about this data from Hans Rosling's very famous TED talk.

Reading in data

External data

You can read in data from almost any source.

- Comma-separated values (CSV) file: Use `read.csv()` (built-in)
- Excel: Use `read_excel()` function from **readxl** package
- SAS, SPSS, Stata: Use the appropriate function from the **haven** package
- SAS, SPSS, S, Stata, Systat, Epi Info, Minitab: Use the appropriate function from the **foreign** package

If you're using Rstudio, you can read in most types of data using "Import Dataset" in the "Environment" tab in the upper right pane.

We'll focus on this more next week. You'll almost always be using external data (not internal R data) in your own research, so this is really important.

Data from a package

Some packages contain data. We will use the `gapminder` dataset from the **gapminder** package. Another great source of dataset is the **datasets** package, which includes commonly used datasets such as `iris` and `mtcars`. (Note that if you want to use `iris` or `mtcars`, you'll need to **load** it – it is installed with R by default, so you don't need to install it.)

To load a *dataset*, use the `data()` function

```
data(gapminder)
```

Looking at your data

There are a number of functions to help you examine your data. You should **always** look at your data to make sure it was read in correctly.

> **i** Note
>
> What **should** your data look like? If you're doing a typical experimental study, the organization of your dataset should follow a pretty standard format:
>
> - Each row is a unit of study (i.e., person, animal, petri dish)
> - Each column is a variable
>
> There are some deviations from this

- **Longitudinal studies** have *multiple observations per unit* and can be organized 2 different ways
  - *Wide* or *multivariate* format: Each repeated measure is a new variable (e.g., X1, X2, X3 for the X variable at 3 different time points)
  - *Tall* or *stacked* or *univariate* format: Each repeated measure is a new row, so each unit will have multiple rows of data (with an additional variable that indicates which time point is which)
- Data from **non-experiments** is often structured differently or even *unstructured*
  - Web scraping
  - Data from devices or automated processes
  - Data from someone who doesn't know what they're doing…

List the dataset

If you want to view the dataset, you can just name the dataset. (You can also `print()` the dataset for the same result.)

```
gapminder
```

```
# A tibble: 1,704 x 6
   country     continent  year lifeExp      pop gdpPercap
   <fct>       <fct>     <int>   <dbl>    <int>     <dbl>
 1 Afghanistan Asia       1952    28.8  8425333      779.
 2 Afghanistan Asia       1957    30.3  9240934      821.
 3 Afghanistan Asia       1962    32.0 10267083      853.
 4 Afghanistan Asia       1967    34.0 11537966      836.
 5 Afghanistan Asia       1972    36.1 13079460      740.
 6 Afghanistan Asia       1977    38.4 14880372      786.
 7 Afghanistan Asia       1982    39.9 12881816      978.
 8 Afghanistan Asia       1987    40.8 13867957      852.
 9 Afghanistan Asia       1992    41.7 16317921      649.
10 Afghanistan Asia       1997    41.8 22227415      635.
# i 1,694 more rows
```

`head()` function

The `head()` function will show the head of the dataset. By default, this shows the first 6 rows of the dataset

```
head(gapminder)
```

```
# A tibble: 6 x 6
  country     continent  year lifeExp      pop gdpPercap
  <fct>       <fct>     <int>   <dbl>    <int>     <dbl>
1 Afghanistan Asia       1952    28.8  8425333      779.
2 Afghanistan Asia       1957    30.3  9240934      821.
3 Afghanistan Asia       1962    32.0 10267083      853.
4 Afghanistan Asia       1967    34.0 11537966      836.
5 Afghanistan Asia       1972    36.1 13079460      740.
6 Afghanistan Asia       1977    38.4 14880372      786.
```

You can change the number of rows it shows using the `n` argument. Here, I've asked for the first 15 rows instead.

```
head(gapminder, n = 15)
```

```
# A tibble: 15 x 6
   country     continent  year lifeExp      pop gdpPercap
   <fct>       <fct>     <int>   <dbl>    <int>     <dbl>
 1 Afghanistan Asia       1952    28.8  8425333      779.
 2 Afghanistan Asia       1957    30.3  9240934      821.
 3 Afghanistan Asia       1962    32.0 10267083      853.
 4 Afghanistan Asia       1967    34.0 11537966      836.
 5 Afghanistan Asia       1972    36.1 13079460      740.
 6 Afghanistan Asia       1977    38.4 14880372      786.
 7 Afghanistan Asia       1982    39.9 12881816      978.
 8 Afghanistan Asia       1987    40.8 13867957      852.
 9 Afghanistan Asia       1992    41.7 16317921      649.
10 Afghanistan Asia       1997    41.8 22227415      635.
11 Afghanistan Asia       2002    42.1 25268405      727.
12 Afghanistan Asia       2007    43.8 31889923      975.
13 Albania     Europe     1952    55.2  1282697     1601.
14 Albania     Europe     1957    59.3  1476505     1942.
15 Albania     Europe     1962    64.8  1728137     2313.
```

> **Note**
>
> There is also a `tail()` function that – guess what – prints the last 6 rows of the dataset. If you wanted to check the end for some reason.

`str()` function

The `str()` function ("`str`" = "structure") tells you about the structure of the dataset. It shows the number of rows and columns, and summarizes some basic info about each variable (like whether it's a number or character variable and the possible values).

```
#|label: str
str(gapminder)
```

```
tibble [1,704 x 6] (S3: tbl_df/tbl/data.frame)
 $ country  : Factor w/ 142 levels "Afghanistan",..: 1 1 1 1 1 1 1 1 1 1 ...
 $ continent: Factor w/ 5 levels "Africa","Americas",..: 3 3 3 3 3 3 3 3 3 3 ...
 $ year     : int [1:1704] 1952 1957 1962 1967 1972 1977 1982 1987 1992 1997 ...
 $ lifeExp  : num [1:1704] 28.8 30.3 32 34 36.1 ...
 $ pop      : int [1:1704] 8425333 9240934 10267083 11537966 13079460 14880372 12881816 1386
 $ gdpPercap: num [1:1704] 779 821 853 836 740 ...
```

`glimpse()` function

The `glimpse()` function is in the **dplyr** package in the **tidyverse**. We're going to talk about **dplyr** more next week, but I wanted to include this function here. This function shows you the number of rows and columns in the dataset. It also flips your dataset so that the variables are rows and prints the dataset out.

```
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   1.5.2
v ggplot2   4.0.0     v tibble    3.3.0
v lubridate 1.9.4     v tidyr     1.3.1
v purrr     1.1.0
-- 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 becor
```

```
glimpse(gapminder)
```

```
Rows: 1,704
Columns: 6
$ country   <fct> "Afghanistan", "Afghanistan", "Afghanistan", "Afghanistan", ~
$ continent <fct> Asia, Asia, Asia, Asia, Asia, Asia, Asia, Asia, Asia, Asia, ~
$ year      <int> 1952, 1957, 1962, 1967, 1972, 1977, 1982, 1987, 1992, 1997, ~
$ lifeExp   <dbl> 28.801, 30.332, 31.997, 34.020, 36.088, 38.438, 39.854, 40.8~
$ pop       <int> 8425333, 9240934, 10267083, 11537966, 13079460, 14880372, 12~
$ gdpPercap <dbl> 779.4453, 820.8530, 853.1007, 836.1971, 739.9811, 786.1134, ~
```

> **Note**
>
> Don't worry about all those warnings that show up about tidyverse and such after `library(tidyverse)`. That's normal.

Activities

Click on the `</> Code` button at the top of the screen, select and copy the text into a new quarto (.qmd) file in Rstudio, and complete these tasks.

1. Render / knit the file together into an HTML file.
2. Render / knit the file together into a PDF file. You'll need to install the **tinytex** package for this to work (or have LaTeX installed on your computer).
3. How many rows are in the `gapminder` dataset? Where did you find that out?
4. How many columns are in the `gapminder` dataset? Where did you find that out?
5. How many **countries** are represented in the `gapminder` dataset? Where did you find that out?
6. The `gapminder` dataset is longitudinal. Is it organized as tall or wide?