Gov 50 Lab 1: Getting Started with R & Data Visualization

Note: This lab was adapted from materials from Matt Blackwell, a previous instructor of this course, as well as a previous problem set from Sam Fuller.

Part 1: Software Setup

Installing R and RStudio

In this lab, we're going to get R, RStudio, and R Markdown set up on your computer. To get started, follow these steps:

1. Download and install the most recent version of R here. There are versions available for the Windows, Mac, and Linux operating systems. On a Windows machine, you will want to install using the `R-x.y.z-win.exe` file where `x.y.z` is a version number. On a Mac, there are two files to choose from: `R-x.y.z-arm64.pkg` is for Macs with an Apple silicon chip (M1, M2, M3, or later, which is every Mac sold since 2021), and `R-x.y.z-x86_64.pkg` is for older Intel Macs. To check which you have, click the Apple menu -> About This Mac and look at the Chip or Processor line.
2. With R installed, download and install RStudio here. RStudio is a type of "integrated development environment" or IDE designed for R. It makes working with R considerably easier and is available for most platforms. It is also free.
3. Install the packages we will use throughout the semester. To do this, either type or copy and paste each of the two code blocks below into the "Console" in RStudio (lower left panel by default) and press Enter. Run the first block before the second. If you are asked if you want to install any packages from source, type "no". Note that the symbols next to `my_packages` are a less than sign `<` followed by a minus sign `-` with no space between them. (Don't be worried if you see some red text here. Those are usually just messages telling you information about the packages you are installing. Unless you see the word `Error` you should be fine.)

```
my_packages <- c("tidyverse", "rmarkdown", "knitr", "usethis", "devtools",
                 "learnr", "tinytex", "gapminder", "pak")
install.packages(my_packages)
```

You can also install packages from other sources, like Github, which we'll be using for some datasets that we'll use in class:

```
pak::pak("samjfuller/gov50data")
```

> **!** Important
>
> We will be using other packages in class—and in section depending on your TF's preferences—so we will be installing more over the course of class. Installing a package is a one-time step, so `install.packages()` always goes in the Console, never in a code chunk of a document you plan to knit.

4. You will submit your labs as PDFs, and producing a PDF from R requires a program called LaTeX. If you have never heard of that, it is completely fine: the `tinytex` package you just installed can set up a small version for you. Run the following line in the Console. It downloads a few hundred megabytes, so give it a few minutes and do not close RStudio while it runs. When it finishes, quit and reopen RStudio so it notices the new installation:

```
tinytex::install_tinytex()  # install TinyTeX
```

5. Go to File -> New File -> R Markdown. Title it "Gov 50 Lab 1", put your name as the author, and select PDF as the output format. This will create a new file with some example content which allows you to both type and run code. Click "Knit" (the button with the ball of yarn at the top of the editor pane) to see how the file renders. If a PDF opens, your LaTeX installation from step 4 works. If instead you see an error mentioning `pdflatex`, `xelatex`, or `LaTeX`, redo step 4 and try again. You can delete the example content below the header once you have seen it knit.

6. Installing a package downloads it to your computer once. To actually use it, you have to *load* it with the `library()` function every time you start a new R session or knit a document. Go to Code -> Insert Chunk to insert a code chunk near the top of your file and put the following in it:

```
library(tidyverse)
library(gapminder)
```

This chunk should be the first one in your file so that the packages are available to everything below it. Run it now by clicking the green arrow on the top right of the chunk. If you see `Error in library(tidyverse) : there is no package called 'tidyverse'`, go back to step 3.

7. You can create section headings by using hash marks (##). At the bottom of the page, create a new heading called "Question 1" by typing `## Question 1` on a new line.

Then, go to Code -> Insert Chunk to insert a workspace to run your code. You are ready to begin the first question!

> **Tip**
>
> For the rest of this lab, do all of your work in code chunks in your R Markdown file, not in the Console. Anything you type in the Console disappears when you knit; only what is in the file ends up in your submission.

Part 2: Introduction to Objects and Basic Operations

Question 1

First, we'll learn how to use R as a calculator. Use the + sign to add 5 and 3. You can run a whole code chunk by clicking on the green arrow on the top right of the chunk, or with the keyboard shortcut Cmd + Shift + Enter (Mac) or Ctrl + Shift + Enter (Windows). To run just the single line your cursor is on, use Cmd + Enter (Mac) or Ctrl + Enter (Windows).

```
5 + 3
```

Use the / symbol to divide 6 by 2.

```
6 / 2
```

Now divide 7 by 2, and then try the following two operators on the same numbers:

```
7 / 2
7 %/% 2
7 %% 2
```

In the main text, tell us in one sentence each what `/`, `%/%`, and `%%` do. Note that in R, `/` always gives you the decimal answer, even when both numbers are whole numbers. This is not true in every programming language, so it is worth remembering.

Use the sqrt() function to take the square root of 16:

```
sqrt(16)
```

Question 2

R stores things in named *objects*: you can create an object and manipulate it with the assignment arrow `<-`. For instance:

```
my_number <- 1           # Assign my_number the value of 1
my_number + 2            # Display my_number + 2 without assigning it
smaller <- my_number / 3 # Assign smaller the value of my_number / 3
```

Descriptive names like these make your code much easier to read later. Create an object called `apples` with the value of 7, an object called `oranges` with the value of 8, and then multiply `apples` and `oranges` together.

Part 3: Vectors, Dataframes and Data Visualization

Data isn't just simply stored in single-value, separate objects, however. In the real world, we would want to aggregate multiple values into a single object. For example, if we have the sociodemographic characteristics of a set of individuals (e.g. income, race, gender) we would want to store each characteristic (variable) in its own aggregated object. This leads us to vectors. Simply put, vectors aggregate data into a single object:

```
two_numbers <- c(1, 2)   # c() "concatenates" 1 and 2 into a single object.
two_numbers              # Displays two_numbers.
```

You could also do a similar thing with text-based values, instead of numbers:

```
lyrics <- c("Slip Like Freudian",
            "Your first and last step to playing yourself like accordion")
```

But what if we want to aggregate data in an even larger, more organized format? What if we want to have both the aggregated information of the different values of a vector (e.g. all the different incomes in our sample) and the combinations of values from all of the vectors each observation has (e.g. the combination of income, race, and gender for each individual)? This is where matrices/dataframes come in.

You can think of dataframes as organizing different variables (column vectors) by observations (row vectors). Specifically, a row denotes an observation/individual whereas a column denotes a variable. The intersection of the two, let's say row 3, column 2 (notated [3,2]), is the value of variable 2 for observation 3 (e.g. Individual 3 has an income of $30,000). So we can either look at the dataframe by column (variable) or row (individual).

In this section, you will get your bearings on how to work with a dataframe and how to produce data visualizations using `ggplot`. The data we will use is the Gapminder dataset, which gives some economic and demographic information about countries over time. The variables in this data are described below.

Name	Description
`country`	name of the country
`continent`	name of the country's continent
`year`	year of the measurement, ranging from 1952 to 2007 in 5-year increments
`lifeExp`	life expectancy at birth, in years
`pop`	population
`gdpPercap`	GDP per capita (US dollars, inflation-adjusted)

Question 3

The `gapminder` dataset comes with the `gapminder` package, and `glimpse()` comes with the `tidyverse`. Both are already loaded by the `library()` chunk you wrote at the top of your file in Part 1, so you can use them directly. Use the `glimpse()` function to get a compact summary of the dataset:

```
glimpse(gapminder)
```

Each line of the output is one variable (column), followed by its type in angle brackets and its first few values. A type of `<fct>` means the variable is a *factor*, R's way of storing categorical variables.

In the main text (that is, outside of a code chunk), tell us how many rows and columns there are in the data set and which of the variables are factors.

> **Note**
>
> You may also see people use `View(gapminder)`, which opens the data in a spreadsheet-like tab in RStudio. It is handy for browsing, but it only works in the Console: a chunk containing `View()` will cause an error when you knit. Stick to `glimpse()` in your file.

Question 4

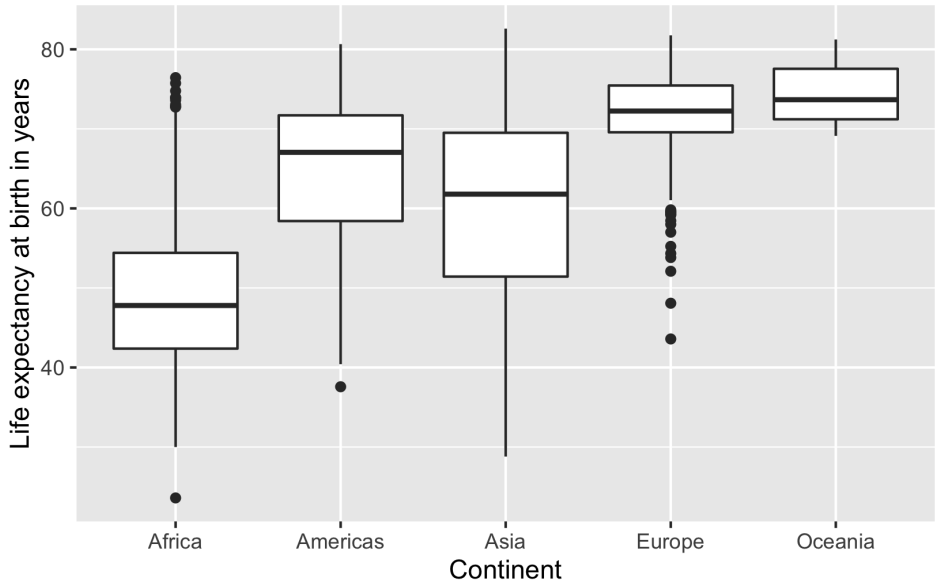
To work with a particular variable (column) in `gapminder`, you can use the `$` sign. For example, you could find the average life expectancy in country-years in the dataset using:

```
mean(gapminder$lifeExp)
```

Find the average population across all the country-years in the dataset. Then, in the main text, report that number in millions of people, rounded to one decimal place.

Question 5

Now it's time to increase the difficulty! Let's investigate how life expectancy varies across the continents. Using `ggplot`, we want you to recreate the following figure:



These are boxplots of the distribution of life expectancy in each continent. Please make sure that you include the labels as shown in this figure. Your code should look something like this:

```
plot_q5 <- ggplot(<arguments>) +
  geom_<type>(<arguments>) +
  ...

plot_q5
```

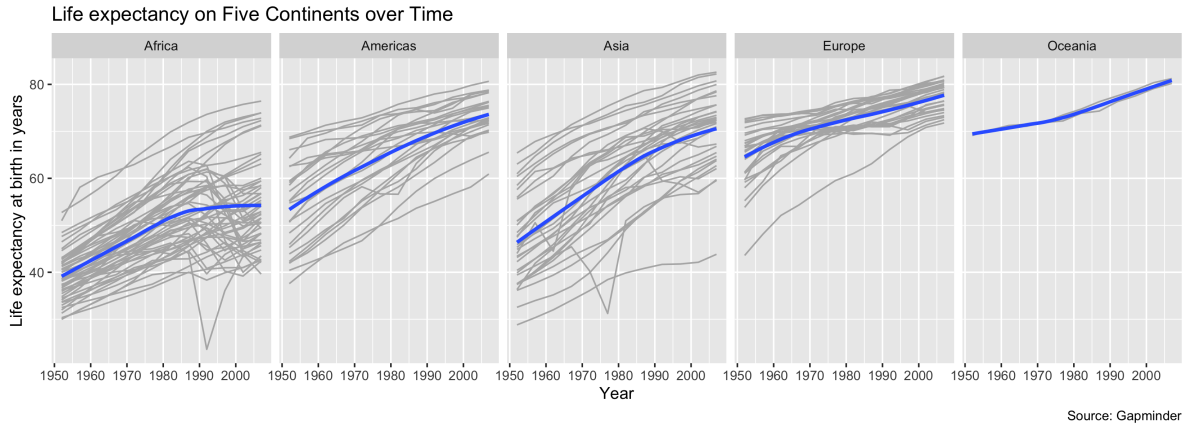
We haven't given you all the steps here, but the goal is to make you more independent. Please feel free to look up the documentation of ggplot to help.

Question 6

Looking at the previous plot, which continent has the highest median life expectancy? Which part of the boxplot can we determine this from?

Question 7 [Bonus Question]

The previous boxplot groups all the years together into one boxplot, but what if we want to understand how life expectancy is changing over time? Next, we will recreate the following plot:



The plot shows each country's life expectancy trajectory over time, broken out by continent with smoothed average lines overlayed for each continent. To get started, we'll give you a few clues about what we've done here:

- The lines for each country use the color `"gray70"`.
- The `linewidth` of the smoothed line is 1.1 and the method used is the loess smoother. We also have turned off the standard errors.
- Make sure that the facets are all on one row. Look at the `facet_wrap` documentation if you need help with this.
- Make sure that the labels are correctly specified, including the title and the "Source: Gapminder" caption. Look up `labs()` if you have not used a caption before.
- Use the chunk options `fig.width = 11` and `fig.height = 4` to shrink the font size so the year labels will not overlap. That is, start your chunk with ```` ```{r, fig.width = 11, fig.height = 4} ```` instead of ```` ```{r} ````.

> **Note**
>
> If you use older ggplot code, you may see `size = 1.1` used for line thickness. That still works but produces a *warning* telling you to use `linewidth` instead. A warning means R ran your code and is flagging something to be aware of; an *error* means R stopped and did not run it. Read both, but only errors will stop your document from knitting.

Submitting

> **Note**
>
> **Where:** The Lab 1 assignment on Canvas.
> **What:** The knitted PDF. Open the PDF before uploading and check that your code, output, plots, and written answers all appear. Keep your `.Rmd` file; your TF may ask to see it.
> **When:** At the end of section. (If you run into issues with installing R, please feel free to ask your TF for more time).