Lab 01 Basic of R, RStudio, and GitHub

RE 519 Data Analytics and Visualization | Autumn 2025

Welcome to start the journey in data science! Through this quarter, we will learn how to conduct data analysis and visualization using R, RStudio, and Tableau. There are tons of resources, both online and within University of Washington, for you to start learning data science, check resource page.

We will have **8** labs in total (2~3 classes for each lab) and you only need to submit **once** after each lab. The due day for each lab can be found on the course wesbite. The submission should include Rmd, html, and any other files required to rerun the codes.

Please check grading policy of this class in advance, including grade curving, late date policy, etc. **For Lab 1–3, the use of any generative AI tool (ChatGPT, Copilot, etc.) is prohibited**. We will run AI detection on each submission. More information about Academic Integrity and the Use of AI.

Recommended Readings:
Appendix B and D, Modern Data Science with R. Baumer et al. 2024.
Chapter 2, Data Visualization. Kieran Healy. 2019.

Lab 01-A: Setup

Why R and RStudio?

R is a programming language built for statistical computing. But, if one already knows Stata or similar software, why use R?

- R is *free*, so you don't need a terminal server.
- R has a *very* large community.
- R can handle virtually any data format.
- R makes replication easy.
- R is a *language* so it can do *everything*.
- R skills transfer to other languages like Python and Julia.

R Studio is a "front-end" or integrated development environment (IDE) for R that can make your life *easier*. We'll show RStudio can…

- Organize your code, output, and plots
- Auto-complete code and highlight syntax
- Help view data and objects
- Enable easy integration of R code into documents with *R Markdown*
- Manage `git` repositories
- Run interactive tutorials

Selling you on R Markdown

The ability to create R Markdown files is a powerful advantage of R:

- Document analyses by combining text, code, and output
  - No copying and pasting into Word
  - Easy for collaborators to understand
  - Show as little or as much code as you want
- Produce many different document types as output
  - PDF documents
  - HTML webpages and reports
  - Word and PowerPoint documents
  - Presentations
- Works with $\LaTeX$ and HTML for math and more formatting control

R and RStudio Installation

Try to complete the setup before the first class. The goal of this assignment is simply to make sure that you have R up and running and can open the source document of this page, a `rmd` (R Markdown) file.

If R & RStudio aren't running and installing properly you may need to update your operating system. Don't worry too much if you can't get them installed and running, I just want everyone to make an attempt which should save you time later on.

1. Download and install R, see here: https://cran.r-project.org/
2. Download and install RStudio Desktop, see here: https://posit.co/download/rstudio-desktop/
3. Windows video guide and Mac video guide.

The latest version of R is **4.5.1 (Great Square Root).** You can check by typing `R.version` into your RStudio Console or run it in one chunk.

Getting Started

Open up RStudio now and try to download the source file, a `rmd` (R Markdown) file here and open it on your local machine.

Use `install.packages("[package name]")` to install any required packages, or R Studio may automatically recognize that there are some packages you need to install and will install them for you.

Then, let's get oriented with the interface:

- *Top Left*: Code **editor** pane, data viewer (browse with tabs)
- *Bottom Left*: **Console** for running code (`>` prompt)
- *Top Right*: List of objects in **environment**, code **history** tab.
- *Bottom Right*: Tabs for browsing files, viewing plots, managing packages, and viewing help files.

You can change the layout and theme in *Preferences > Pane Layout/Appearance*.

Lab 01-B: Basic of R/RStudio and Markdown

R Markdown

Markdown is a simple formatting syntax for authoring HTML, PDF, and MS Word documents. For more details on using R Markdown see http://rmarkdown.rstudio.com. In the lab, we will review some basic Markdown syntax: Markdown Live Preview.

You can put some math $y = \left(\frac{2}{3}\right)^2$ right up in there using $\LaTeX$.

$$\frac{1}{n}\sum_{i=1}^{n} x_i = \bar{x}_n$$

Note: $\LaTeX$ is not required in this class. If you are curious about how mathematicians compile their books, you may refer to the Learn LaTeX in 30 minutes from Overleaf. Overleaf is the most popular collaborative cloud-based $\LaTeX$ editor and UW students can get Overleaf Premium account for free.

R Markdown Documents

This document is a R Markdown file, you can create a new one by:

1. Choose *File > New File > R Markdown…*
2. Make sure *HTML Output* is selected and click OK (select PDF if you'd like PDF as outputs)
3. Save the file somewhere, call it `any_name.Rmd`
4. Click the *Knit HTML* button
5. Watch the progress in the R Markdown pane, then gaze upon your result!

You may also open up the HTML file in your computer's browser if you so desire, using the *Open in Browser* button at the top of the preview window.

R Markdown Headers

The header of an .Rmd file is a YAML code block, and everything else is part of the main document. To mess with global formatting, you can modify the header.

```
---
title: "Untitled"
author: "Name of a person"
date: "March 28, 2025"
output:
  html_document:
    toc: true # including a table of contents
    theme: readable # using a theme called readable
---
```

R Markdown Tinkering

R Markdown docs can be modified in many ways. Visit these links for more information.

- Ways to modify the overall document appearance
- Ways to format parts of your document
- R Markdown: The Definitive Guide

📚 TODO:Using Markdown for your Resume / CV

4 points

Create a brief Resume / CV using R Markdown with your name as level 4 headings, headings for (at least) education or employment. Each of the sections should include a bullet list of *jobs/degrees. Highlight the year in bold. Try and incorporate as many R Markdown features as possible. You can work on this document or create a new markdown file.*

R Code in R Markdown

Inside RMarkdown, lines of R code are called **chunks**. Code is sandwiched between sets of three backticks and `{r}`. This chunk of code…

~~~
```{r}
mean(c(1,2,3,4,5,6))
```
~~~

Produces this output in your document:

```
mean(c(1,2,3,4,5,6)) # calculate the mean value of a list of numbers
```

```
## [1] 3.5
```

Most commonly used two shortcuts for chunks:

Action	Mac Shortcut	Windows Shortcut
Insert R code chunk	⌘ + Option + I	Ctrl + Alt + I
Run current chunk	⌘ + Option + C	Ctrl + Alt + C

Chunk Options

Chunks have options that control what happens with their code, such as:

- `echo=FALSE` : Keeps R code from being shown in the document
- `eval=FALSE` : Shows R code in the document without running it
- `include=FALSE` : Hides all output but still runs code (good for `setup` chunks where you load packages!)
- `results='hide'` : Hides R's (non-plot) output from the document
- `cache=TRUE` : Saves results of running that chunk so if it takes a while, you won't have to re-run it each time you re-knit the document
- `fig.height=5, fig.width=5` : modify the dimensions of any plots that are generated in the chunk (units are in inches)

Some of these can be modified using the gear-shaped *Modify Chunk Options* button in each chunk. There are a *lot* of other options, however.

Editing and Running Code

In R Markdown documents, click within a code chunk and click the green arrow to run the chunk. The button beside that runs *all prior chunks*. The console will show the lines you ran followed by any printed output. There are more options on the top right of this window (green arrow with `Run`).

```
pi # output the value of pi
```

```
## [1] 3.141593
```

Sometimes we want to insert a value directly into our text. We do that using code in single backticks starting off with `r`. For example, the value of `pi` rounded to the nearest three decimals is 3.142.

Math and Logical Operation in R

```
# Math operations
2 + 3       # addition → 5
5 - 2       # subtraction → 3
4 * 6       # multiplication → 24
10 / 4      # division → 2.5
2^3         # exponentiation → 8
10 %% 3     # modulo (remainder) → 1
sqrt(400)   # square root of 400 → 20

x <- c(2, 4, 6, 8, 10) # Create those numbers to calculate

mean(x)       # average → 6
var(x)        # variance
sd(x)         # standard deviation
sum(x)        # sum of all elements
min(x)        # smallest value
max(x)        # largest value
range(x)      # min and max together
length(x)     # number of elements


# Logical operations
```

```r
3 == 3      # equal → TRUE
3 != 4      # not equal → TRUE
5 > 2       # greater than → TRUE
2 < 1       # less than → FALSE
5 >= 5      # greater or equal → TRUE
2 <= 3      # less or equal → TRUE
TRUE & FALSE    # AND → FALSE
TRUE | FALSE    # OR → TRUE
!TRUE           # NOT → FALSE
```

Functions and Help

`sqrt()` is an example of a **function** in R. If we didn't have a good guess as to what `sqrt()` will do, we can type `?sqrt` in the console and look at the **Help** panel on the right.

```r
?sqrt # what is sqrt()?
```

Arguments are the *inputs* to a function. In this case, the only argument to `sqrt()` is `x` which can be a number or a vector of numbers. *Help* files provide documentation on how to use functions and what functions produce.

Common Data Types in R

R has several *basic data types* that are used to build larger data structures.

Data Type	Example	Description	Check with
Numeric	`x <- 3.142`	Decimal numbers (floating point)	`is.numeric(x)`
Integer	`y <- 5L`	Whole numbers (the **L** forces integer type)	`is.integer(y)`
Character	`z <- "Hello"`	Text or string values	`is.character(z)`
Logical	`flag <- TRUE`	Boolean values: **TRUE** or **FALSE**	`is.logical(flag)`
Factor	`f <- factor(c("A", "B", "C"))`	Categorical values with fixed levels	`is.factor(f)`

You can also check type with `class()` or `is.*()` functions.

Data Structures in R

Objects

R stores everything as an **object**, including data, functions, models, and output. Creating an object can be done using the assignment operator: `<-`:

```
new_object_preferred <- 144
new_object_not_preferred = 144 # using `=` is OK in R but not recommended; `=` is mainly for function a
```

The *Environment Pane* (right top) shows all active objects once you created in your current R session. Each assignment (`<-`) updates this list. Then. you can display or 'use' an object simply by using its name.

```
new_object_preferred
```

```
## [1] 144
```

```
sqrt(new_object_preferred)
```

```
## [1] 12
```

Object names can contain `_` (preferred) and `.` in them but CANNOT begin with numbers. Try to be consistent and clear in naming objects. Clear naming will help you save lots of time in the future!

Vectors

Vector is a sequence of elements of the same type.

```
numbers <- c(1, 2, 3, 4)       # numeric vector; `c()` means combine
names <- c("Alice", "Bob")     # character vector
```

After creating them, we often want to inspect what they contain and their structure.

```
print(numbers)           # show the values
```

```
## [1] 1 2 3 4
```

```
length(numbers)          # check the number of elements
```

```
## [1] 4
```

```
class(numbers)           # check the type
```

```
## [1] "numeric"
```

```
numbers > 2              # check whether each element is larger than 2
```

```
## [1] FALSE FALSE  TRUE  TRUE
```

We may also want to select some elements in one vector.

```
numbers[1]            # show the first element
```

```
## [1] 1
```

```
numbers[-1]           # show the vector without the first element
```

```
## [1] 2 3 4
```

```
numbers[1:3]          # show the elements with index 1 to 3
```

```
## [1] 1 2 3
```

```
numbers[c(1,3)]       # show the elements with index 1 and 3
```

```
## [1] 1 3
```

```
numbers[c(TRUE, TRUE, FALSE)]       # show the elements with `TRUE`
```

```
## [1] 1 2 4
```

Martix

A 2D collection of elements of the same type (optional for this class).

```r
mat <- matrix(1:6, nrow = 2, ncol = 3)
dim(mat)             # dimensions (# of rows, # of columns)
```

```
## [1] 2 3
```

Lists

A collection of objects (can be of different types). List members can be named, referenced using the `[[` operator or using the names. *Note*: `list[ ]` will keep the result as a list (still wrapped).

```r
my_list <- list(name = "Alice", age = 25, scores = c(90, 85, 88))
```

```r
my_list$age     # get element by name ("age" = 25)
```

```
## [1] 25
```

```r
my_list[[2]]    # get element by position (2nd element = 25)
```

```
## [1] 25
```

```r
my_list[2]      # still a list with "age = 25"
```

```
## $age
## [1] 25
```

Dataframes

Most data sets you will work with will be read into R and stored as a **dataframe**, so this course will mainly focus on manipulating and visualizing these objects. We will leave dataframes to next lab session.

📚 TODO: R Basic Review Tasks

5 points

What's your birthday (or any special day for you)? Please print it out using the format

`YYYY-MM-DD` without showing the R code in the output HTML document. What's the data type of your `YYYY-MM-DD` object?

```
# TODO
```

In R, a date can be defined as a class called `date` using the function `as.Date()`. Loop up the documentation of this function and tell me what's the simplest argument this function needs. Then, assign your birthday to an object.

```
# TODO
```

Let's find the date for today! Sorry for the stupid request, but we need to use a function called `Sys.Date()` to always get "today". Check the documentation, what argument does we need?

```
# TODO
```

Create a vector with those two dates.

```
# TODO
```

Let's use indexing `[]` to select today's date, then subtract your birthday. What's happened?

```
# TODO
```

Can you calculate your age by simply dividing the difference in days by `365`? Note: The calculation doesn't account for the exact month/day alignment — it just assumes each year is a flat 365 days. You can install package `lubridate` to calculate a more precise age (optional).

```
# TODO
```

Then, finish the sentence using inline R code:

Today is `today's date`, it has been `number of year` years since {what did happen on the big day, such as my birthday}.

Lab 01-C: Dataframes and Accessing Census Data

Working Directory

```
# show the current working directory
getwd()
```

```
## [1] "/Users/yohaoyu/Repo/data-analytics-visualization/labs"
```

```
# TODO: modify the folder stored this `.Rmd` as the working directory
setwd("/Users/yohaoyu/Repo/data-analytics-visualization/labs")
```

- `.Rmd` files use their current directory as a working directory.
- For larger projects, instead of setting a working directory, it is usually better to use RStudio projects to manage working directories.
- Windows users: If you copy a path from Explorer, make sure to change back slashes (`\`) to forward slashes (`/`) for the file paths

Access Census Data

In this lab, we will explore datasets from American Community Survey (ACS) via DATA.CENSUS.GOV and R package `tidycensus`.

We will use 2023 ACS 5-year estimate for B25075 Value, which means the estimated selling prices for occupied housing units, as an example. Some intro to this data from Census Reporter: https://censusreporter.org/tables/B25075/.

To use `tidycensus`, we need load the package and have a Census API key. A key can be obtained from http://api.census.gov/data/key_signup.html.

```
# remove the following '#' to install the `tidycensus` and `tidyverse` package; `tidyverse` will be mos
# install.packages("tidycensus")
# install.packages("tidyverse")
# This loads packages into the current session; however, it is best practice to load all packages at th
library(tidycensus)
library(tidyverse)
```

```
census_api_key("96f943fd5b38142362574386830d3c855e82dd28")
```

```
## To install your API key for use in future sessions, run this function with `install = TRUE`.
```

```
king_value <- get_acs(geography = "tract", # extract all tract-level data
            table = "B25075",
            state = "53", # 53 means Washington State, you can use 'WA'
            county = "033", # 033 means King County, you can use 'king'
            year = 2023)
```

```
## Getting data from the 2019-2023 5-year ACS
```

We can save this `king_value` tibble (just like a table) on our local machine as a `.csv` file. `.csv` is one of the most common-used file types in the world of data science. But, what is a CSV file?

```
# The CSV file will be saved in your current working directory
# Here, we use the function `write_csv` from package `readr`, which is built within `tidyverse`
write_csv(king_value, "king_value.csv")
```

Once you've set the working directory—or you're in an RStudio project—you can refer to folders and files within the working directory using relative paths, like we've done in previous codes - saving the file in the current working directory. Then, we can also import this `.csv` again using `read_csv()`:

```
another_king_value <- read_csv("king_value.csv")
```

```
## Rows: 13365 Columns: 5
## ── Column specification ────────────────────────────────────────────────────────
## Delimiter: ","
## chr (2): NAME, variable
## dbl (3): GEOID, estimate, moe
##
## ℹ Use `spec()` to retrieve the full column specification for this data.
## ℹ Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

Dataframes

Most data sets you will work with will be read into R and stored as a **dataframe**, just like the `king_value` data. So this course will mainly focus on manipulating and visualizing these objects.

```
# check whether an object is a dataframe
is.data.frame(king_value)
```

```
## [1] TRUE
```

You can check the dataframe by clicking the object name in the *Environment Panel* (right top) or using `view()` function.

```
# check the dataframe
view(king_value)

# check the first three rows, default as five rows
head(king_value, 3)
```

```
## # A tibble: 3 × 5
##   GEOID       NAME                                      variable estimate   moe
##   <chr>       <chr>                                     <chr>       <dbl> <dbl>
## 1 53033000101 Census Tract 1.01; King County; Washington B25075_…      159    96
## 2 53033000101 Census Tract 1.01; King County; Washington B25075_…        0    13
## 3 53033000101 Census Tract 1.01; King County; Washington B25075_…        0    13
```

```
# check the column names
colnames(king_value)
```

```
## [1] "GEOID"    "NAME"     "variable" "estimate" "moe"
```

```
# number of rows and columns
dim(king_value)
```

```
## [1] 13365     5
```

```
# select GEOID column and elements with index 1 to 10
king_value$GEOID[1:10]
```

```
##  [1] "53033000101" "53033000101" "53033000101" "53033000101" "53033000101"
##  [6] "53033000101" "53033000101" "53033000101" "53033000101" "53033000101"
```

```
# calculate the average estimate
# na.rm = TRUE helps to ignore missing values
mean(king_value$estimate, na.rm = TRUE)
```

```
## [1] 77.8737
```

When you click the **Knit** button (top of this panel) a document will be generated that includes both content as well as the output of any embedded R code chunks within the document.

📚 TODO: Explore Census Demographic Data

7 points

Let's do the similar step for other variables, select 1~2 interesting datasets from ACS, download them via API or via DATA.CENSUS.GOV. You may look up variables you're interested in from this Census Reporter site or ACS Technical Documentation.

Using a dataframe to store the data and calculate basic summary statistics such as mean, median, min/max, 25th/75th percentiles for your data; and briefly introduce why you're interested in this data and what did you find through the simple exploration.

After finish this lab, you need to submit everything via Canvas assignment page.

```
# TODO
```

Acknowledgement

The materials are developed by Haoyu Yue based materials from Dr. Feiyang Sun at UC San Diego, Siman Ning and Christian Phillips at University of Washington, Dr. Charles Lanfear at University of Cambridge.