Section 1: Introduction to R and RStudio

Contents

1 Installing R and RStudio

Before Section

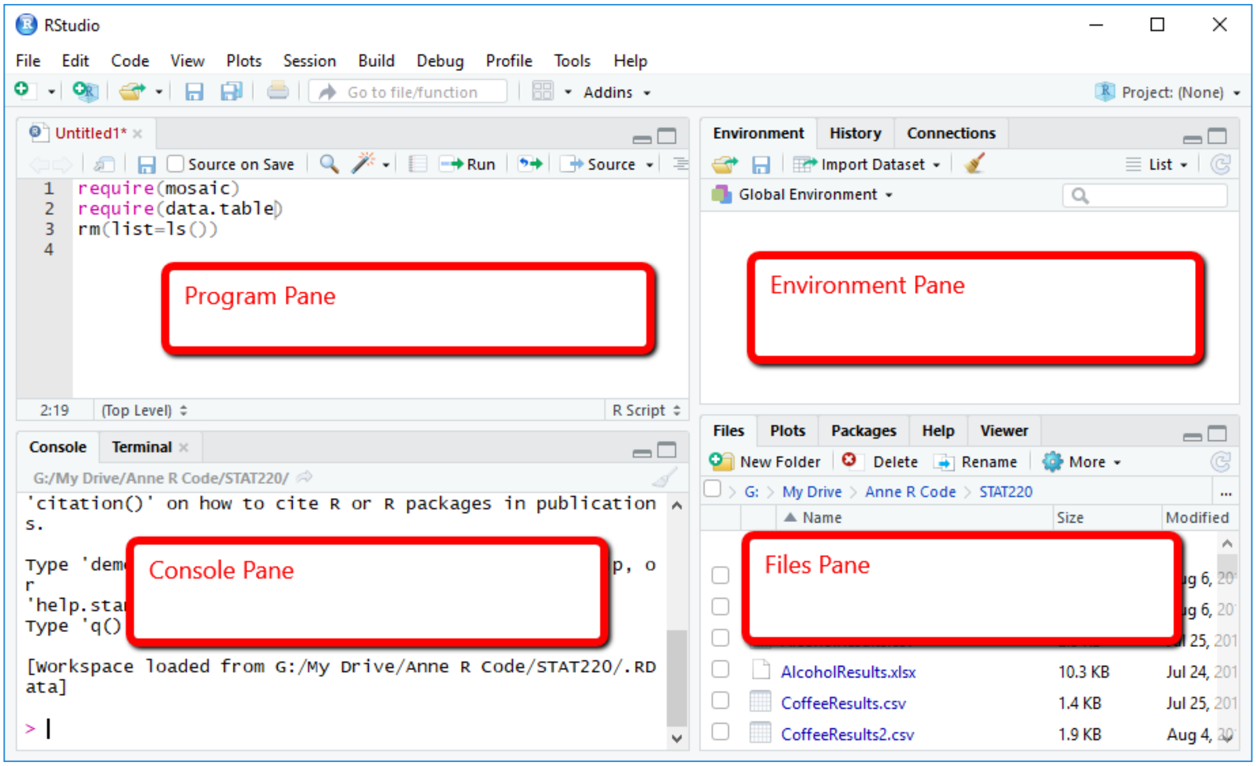
Download and install **BOTH** R and RStudio. Visit the following links and download the latest version of R and RStudio suitable for your operating system.

- To download R, visit https://cloud.r-project.org/
- Install R
- To download RStudio, visit https://posit.co/download/rstudio-desktop/#download
- Install RStudio
- Screenshots are available at https://bcourses.berkeley.edu/files/90752389/download?download_frd=1
- To learn more about R and RStudio, visit https://rstudio-education.github.io/hopr/starting.html

2 Rstudio

Before Section

- Watch video (5min): https://www.youtube.com/watch?v=FIrsOBy5k58



```
require(mosaic)
require(data.table)
rm(list=ls())
```

```
'citation()' on how to cite R or R packages in publication
s.

Type 'dem                                                 p, o
r
'help.sta
Type 'q()

[workspace loaded from G:/My Drive/Anne R Code/STAT220/.RD
ata]

>
```

Name	Size	Modified
AlcoholResults.xlsx	10.3 KB	Jul 24, 201
CoffeeResults.csv	1.4 KB	Jul 25, 201
CoffeeResults2.csv	1.9 KB	Aug 4, 20

Figure 1: Source: https://bookdown.org/ageraci/STAT160Companion

Check after section

- □ know what each pane in RStudio is for
  - (top-left) program pane where you write codes
  - (bottom-left) console pane where you see the codes you executed and their corresponding results
  - (top-right) environment pane that shows objects you created
  - (bottom-right) files pane that shows files, plots, help, and packages
- □ know how to create and save an R script
- □ know how to execute an R code

3 R

3.1 R Objects

You can create an R object by using <-.

```
a <- 1
print(a)
```

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

Do you see the object `a` in your environment pane?

You can remove object(s) by using `rm`.

```
rm(a)
```

If you would like to clear your environment (i.e., remove all objects you created), type `rm(list=ls())`

3.2 Working Directory

You can check your current working directory by typing `getwd()` and set your working directory by using `setwd(<YOUR PATH>)`

```
getwd()
```

```
## [1] "C:/Users/shuoy/Dropbox/161/Sections/Section1"
```

```
setwd("C:/Users/shuoy/Dropbox/EEP161/Sections/Section1")
```

3.3 Comments

You can write comments in your R script by typing # before your comment. R will skip executing the lines starting with #. For example,

```
a <- 2
# a <- 1
print(a)
```

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

3.4 Variable Types

R has several basic variable types (or classes) that are commonly used in programming and data analysis. Understanding these types is crucial for working with data effectively. - Numeric: Represents numbers (e.g., 42, 3.14). - Character: Represents text or strings (e.g., "hello"). - Logical: Represents `TRUE` or `FALSE` values. - Integer: Whole numbers specified with `L` (e.g., 10L). - Factor: Categorical data stored as levels. - Complex: Numbers with imaginary parts (e.g., 1 + 2i).

```
# Create variables
num <- 3.14          # Numeric
txt <- "hello"       # Character
flag <- TRUE         # Logical
int <- 10L           # Integer
category <- factor(c("A", "B", "A"))  # Factor

# Check their types
class(num)           # "numeric"
```

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

```r
class(txt)          # "character"
```

```
## [1] "character"
```

```r
class(flag)         # "logical"
```

```
## [1] "logical"
```

```r
class(int)          # "integer"
```

```
## [1] "integer"
```

```r
class(category)     # "factor"
```

```
## [1] "factor"
```

```r
# Use is.* functions
is.numeric(num)     # TRUE
```

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

```r
is.character(txt)   # TRUE
```

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

```r
is.logical(flag)    # TRUE
```

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

Convert between types as needed.

```r
num <- as.numeric("42")   # Convert character to numeric
text <- as.character(42)  # Convert numeric to character
```

3.5 Simple Calculations

R can be used like a calculator to perform basic arithmetic operations. Simply type the operation into the console and press Enter to see the result.

```r
# Basic arithmetic operations
2 + 3       # Addition
```

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

```
10 - 4       # Subtraction
```

```
## [1] 6
```

```
5 * 6        # Multiplication
```

```
## [1] 30
```

```
20 / 4       # Division
```

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

```
2^3          # Exponentiation (2 raised to the power of 3)
```

```
## [1] 8
```

```
sqrt(16)     # Square root
```

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

R follows the standard order of operations (parentheses, exponents, multiplication/division, and addition/subtraction). You can use parentheses to make your calculations clear and precise.

```
# Using parentheses for clarity
(2 + 3) * 4
```

```
## [1] 20
```

Certain operations only work on specific types.

```
"text" + 1  # Error: non-numeric argument
```

3.6 Vectors

A **vector** is a collection of elements of the same type (e.g., numbers, characters, or logical values). You can create a vector using the `c()` function, which stands for “combine” or “concatenate.”

```
# Numeric vector
yields <- c(7.5, 4.2, 3.8)

# Character vector
crops <- c("corn", "wheat", "soybean")

# Logical vector
is_large_field <- c(TRUE, FALSE, TRUE)

# Print vectors
print(yields)
```

```
## [1] 7.5 4.2 3.8
```

```
print(crops)
```

```
## [1] "corn"    "wheat"   "soybean"
```

```
print(is_large_field)
```

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

You can access individual elements in a vector using square brackets (`[]`). R uses 1-based indexing, meaning the first element is at position 1.

```
# Accessing vector elements
yields[1]   # First element of yields
```

```
## [1] 7.5
```

```
crops[3]    # Third element of crops
```

```
## [1] "soybean"
```

You can perform operations on entire vectors, and R will apply the operation to each element.

```
# Vectorized operations
yields + 1              # Add 1 to each yield
```

```
## [1] 8.5 5.2 4.8
```

```
yields * 2              # Multiply each yield by 2
```

```
## [1] 15.0  8.4  7.6
```

```
yields > 4              # Check which yields are greater than 4
```

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

3.7 Data Frames

A **data frame** is a table where each column is a vector, and all columns have the same number of elements. It's one of the most common data structures in R for organizing and analyzing data.

You can create a data frame using the `data.frame()` function.

```
# Creating a data frame
data <- data.frame(
  Crop = crops,
  Yield = yields,
  LargeField = is_large_field
)

# Print the data frame
print(data)
```

```
##      Crop Yield LargeField
## 1    corn   7.5       TRUE
## 2   wheat   4.2      FALSE
## 3 soybean   3.8       TRUE
```

You can access specific rows, columns, or individual elements in a data frame using the $ operator or square brackets ([,]).

```r
# Access a column
data$Yield
```

```
## [1] 7.5 4.2 3.8
```

```r
# Access specific rows and columns
data[1, ]    # First row
```

```
##   Crop Yield LargeField
## 1 corn   7.5       TRUE
```

```r
data[, 2]    # Second column
```

```
## [1] 7.5 4.2 3.8
```

```r
data[1, 2]   # Element in the first row and second column
```

```
## [1] 7.5
```

You can also view the structure and summary of a data frame.

```r
# View structure and summary
str(data)
```

```
## 'data.frame':    3 obs. of  3 variables:
##  $ Crop      : chr  "corn" "wheat" "soybean"
##  $ Yield     : num  7.5 4.2 3.8
##  $ LargeField: logi  TRUE FALSE TRUE
```

```r
summary(data)
```

```
##      Crop               Yield       LargeField
##  Length:3           Min.   :3.800   Mode :logical
##  Class :character   1st Qu.:4.000   FALSE:1
##  Mode  :character   Median :4.200   TRUE :2
##                     Mean   :5.167
##                     3rd Qu.:5.850
##                     Max.   :7.500
```

Check after section

- □ know how to create and remove an object
- □ know how to check and set a working directory
- □ know how to comment by using # in R script
- □ know how to distinguish between different variable type
- □ know how to conduct basic calculations
- □ know how to work with vectors and data frames

4 Package

A package bundles together code, data, documentation, and tests, and is easy to share with others. You can install package using `install.package("some_pacakage")`. Note that you need double quote the package name you want to install. Once installed, you can load package by typing `library(some_package)`. In this case, you do not need to double quote. See

```
install.packages("tidyverse")
library(tidyverse)
```

4.1 tidyverse package

One package that we are going to frequently use is `tidyverse`. `install.packages("tidyverse")` will install the following packages:

```
##  [1] "broom"         "conflicted"    "cli"           "dbplyr"       
##  [5] "dplyr"         "dtplyr"        "forcats"       "ggplot2"      
##  [9] "googledrive"   "googlesheets4" "haven"         "hms"          
## [13] "httr"          "jsonlite"      "lubridate"     "magrittr"     
## [17] "modelr"        "pillar"        "purrr"         "ragg"         
## [21] "readr"         "readxl"        "reprex"        "rlang"        
## [25] "rstudioapi"    "rvest"         "stringr"       "tibble"       
## [29] "tidyr"         "xml2"
```

The `tidyverse` is a collection of R packages designed for data science. It provides a cohesive and consistent approach to importing, tidying, transforming, visualizing, and modeling data. These packages share an underlying philosophy and grammar of data manipulation and visualization. Key features include:

- Consistent syntax and integrated packages: Functions in the tidyverse use a consistent structure, making it easy to learn and use.
- Pipe operator (%>%): Allows chaining commands together in a readable and efficient way.
- Core `tidyverse` packages: `ggplot2` for data visualization based on the grammar of graphics. `dplyr` for data manipulation, such as filtering, summarizing, and transforming. `tidyr` for tidying messy data, such as reshaping data frames. `readr` for reading rectangular data (like CSV files) into R. `stringr` for string manipulation. Additional tools like `purrr` for functional programming and `forcats` for handling categorical data enhance its versatility.

4.2 Pipe Operator (%>%)

The pipe operator takes the output of one function and uses it as the input for the next function. This eliminates the need to write intermediate variables or nested function calls, simplifying your workflow.

```
# Without pipes (base R):
filtered <- data[data$Yield > 4, ]
AvgYield <- mean(filtered$Yield)
print(data.frame(AvgYield = AvgYield))
```

```
##   AvgYield
## 1     5.85
```

```
# With pipes:
data %>%
  filter(Yield > 4) %>%
  summarize(AvgYield = mean(Yield)) %>%
  print()
```

```
##   AvgYield
## 1     5.85
```

Check after section

- □ know how to install by using `install.package` and load package `library`
- □ know what `tidyverse` package is.
- □ know how to use pipe operator.

5 Working with Data

Let's load a dataset.

```
Daily_price <- read.csv("DailyPrices.csv")
```

You can check first several rows by using `head`.

```
head(Daily_price)
```

```
##         Date   Egg.Class Market.Name Delivery.Name      Price.Unit Low.Price
## 1 2013-02-04       LARGE  CALIFORNIA       INVOICE CENTS PER DOZEN       171
## 2 2013-02-04       JUMBO  CALIFORNIA       INVOICE CENTS PER DOZEN       182
## 3 2013-02-04 EXTRA LARGE  CALIFORNIA       INVOICE CENTS PER DOZEN       175
## 4 2013-02-04       LARGE  CALIFORNIA       INVOICE CENTS PER DOZEN       171
## 5 2013-02-04      MEDIUM  CALIFORNIA       INVOICE CENTS PER DOZEN       139
## 6 2013-02-04      MEDIUM  CALIFORNIA       INVOICE CENTS PER DOZEN       139
##   High.Price Grade Mostly.Low Mostly.High
## 1        171  <NA>         NA          NA
## 2        182  <NA>         NA          NA
## 3        175  <NA>         NA          NA
## 4        171  <NA>         NA          NA
## 5        139  <NA>         NA          NA
## 6        139  <NA>         NA          NA
```

If you want to see all rows and columns, use `View()` or click the object in the environment pane.

```
View(Daily_price)
```

We can do a quick summary of data by using `summary`.

```
summary(Daily_price)
```

```
##      Date            Egg.Class         Market.Name        Delivery.Name
##  Length:42702       Length:42702       Length:42702       Length:42702
##  Class :character   Class :character   Class :character   Class :character
##  Mode  :character   Mode  :character   Mode  :character   Mode  :character
##
##
##
##
##   Price.Unit          Low.Price       High.Price       Grade
##  Length:42702       Min.   :  0.0   Min.   :  5.0   Length:42702
##  Class :character   1st Qu.: 81.0   1st Qu.: 88.0   Class :character
##  Mode  :character   Median :128.0   Median :133.0   Mode  :character
##                     Mean   :154.7   Mean   :160.3
##                     3rd Qu.:193.0   3rd Qu.:196.0
##                     Max.   :899.0   Max.   :910.0
##                     NA’s   :12      NA’s   :12
##    Mostly.Low     Mostly.High
##  Min.   :  3.0   Min.   : 24.0
##  1st Qu.: 73.0   1st Qu.: 81.0
##  Median :104.0   Median :111.0
##  Mean   :130.1   Mean   :138.9
##  3rd Qu.:157.0   3rd Qu.:165.0
##  Max.   :620.0   Max.   :612.0
##  NA’s   :18726   NA’s   :24722
```