Introduction to R Session 1

Department of Sociology | University of Oxford

Casey Breen

2024-10-15

Welcome to “Intro to R”

- Two sessions:
  - Thursday, October 10th, 1pm - 4pm
  - Friday, October 11th, 9:30am - 12:30pm
- Course website:
  - www.github.com/caseybreen/intro_r
  - Slides, exercises, and solutions

Course goals

- Overview: why R is a powerful tool for social science research
- Install R and `RStudio`
- Introduction to R syntax, data types, and data structures
- Basic understanding of data manipulation and visualization

Course agenda

- **Session 1**
  - Module 1: Introduction to R, `RStudio,` and code formats
  - Module 2: R programming fundamentals (syntax, operators, data types, data structures, sequencing)
  - Module 3: Working with data (indexing vectors / matrices, importing data)
- **Session 2**
  - Module 4: Importing and exporting data
  - Module 5: Data manipulation (`dplyr`) and data visualization (`ggplot2`)
  - Module 6: Best practices and resources for self-study

Module 1

Introduction to R, `RStudio`, and code formats

Learning objectives:

- Installing R and `RStudio`
- Why R?
- Understanding R Scripts, R notebooks, Quarto documents

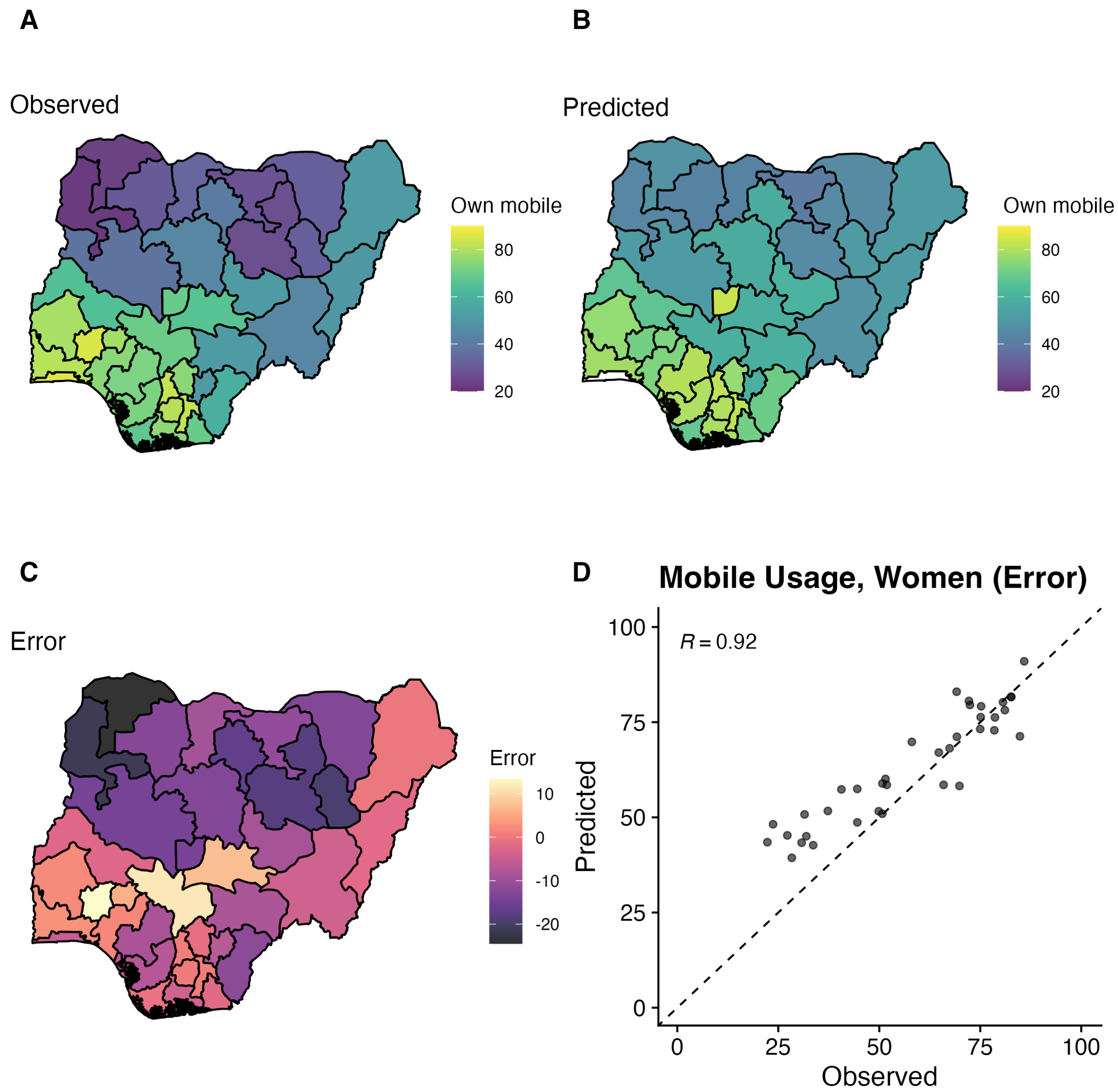
R and `RStudio`

- R is a statistical programming language
  - Download: https://cloud.r-project.org
- `RStudio` is an integrated development environment (IDE) for R programming
  - Download: http://www.rstudio.com/download

Why R?

- Free, open source — great for reproducibility and open science
- Powerful language for data manipulation, statistical analysis, and publication-ready data visualizations
- Excellent community, lots of free resources

Data visualization

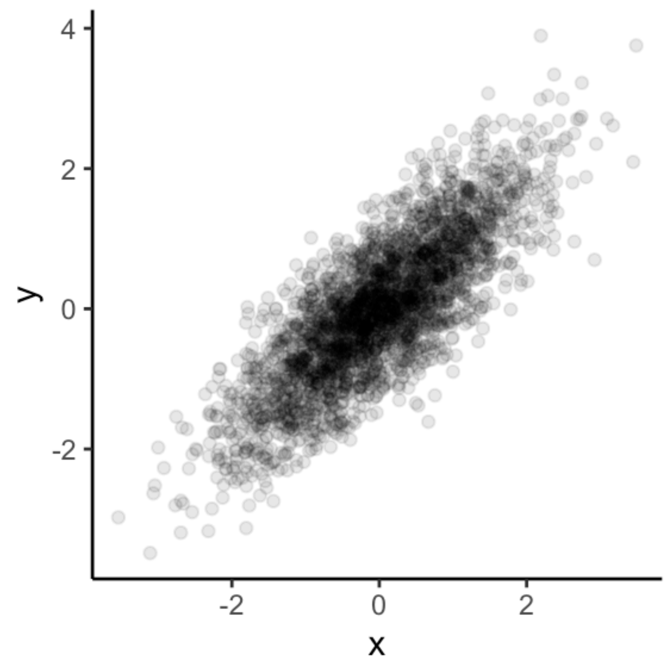


Easy to simulate + plot data

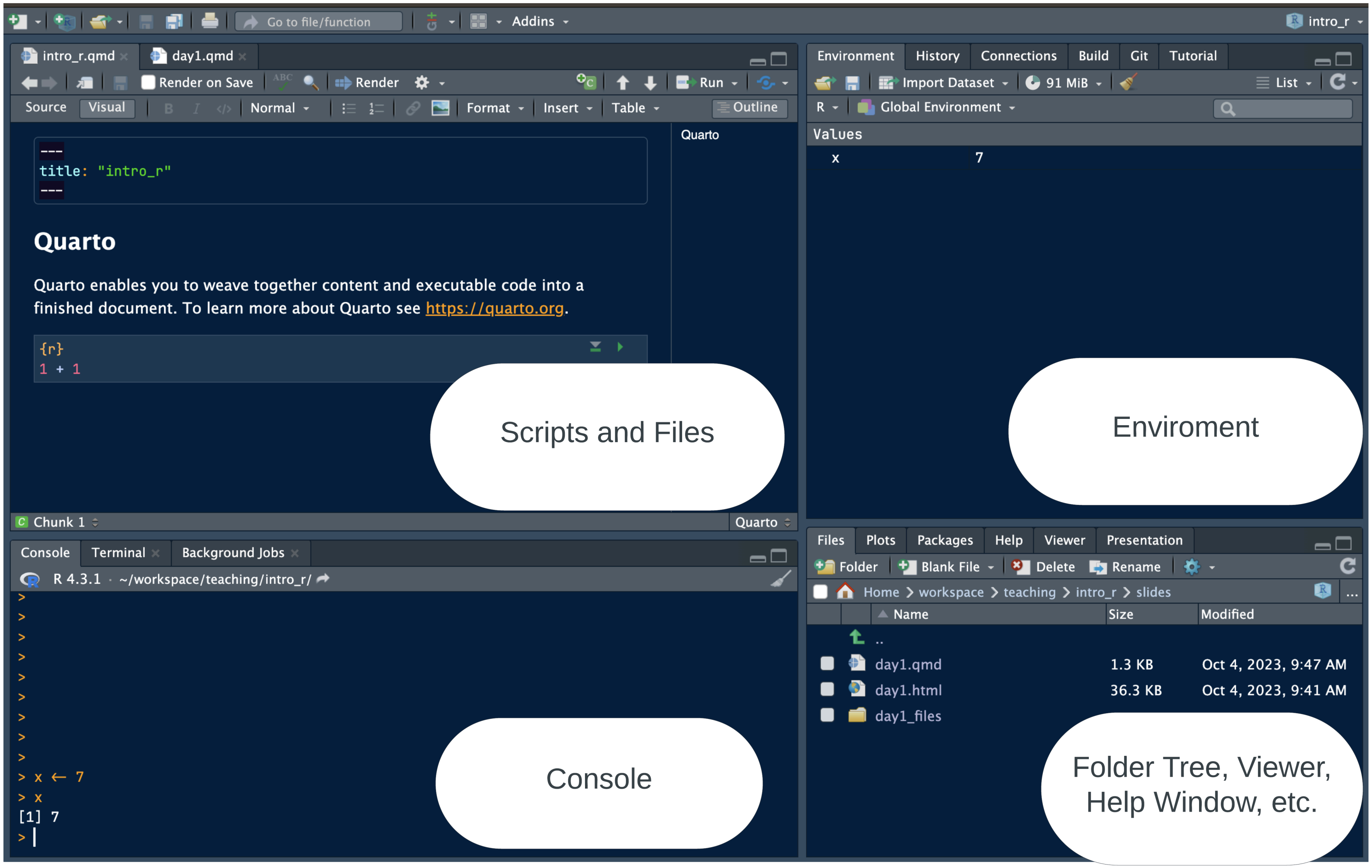
```r
# Generate random data for x
x <- rnorm(n = 3000)
y <- 0.8 * x + rnorm(3000, 0, sqrt(1 - 0.8^2))

# Create data.frame
data_df <- data.frame(x = x, y = y)

# Generate visualization
data_df %>%
  ggplot(aes(x = x, y = y)) +
  geom_point(alpha = 0.1) +
  theme_classic()
```



RStudio panes



Why `RStudio`?

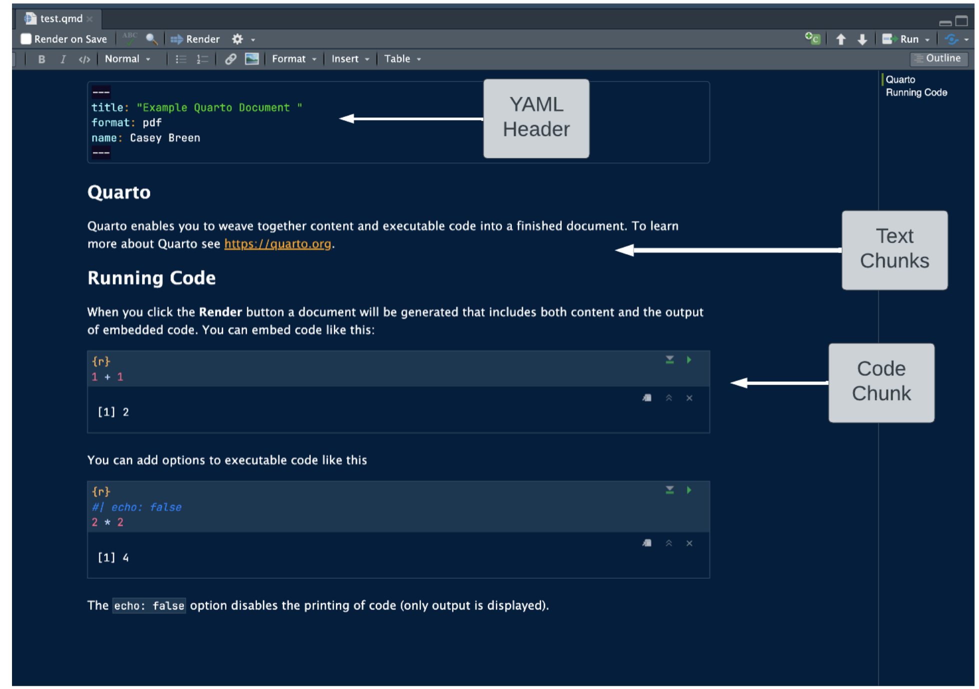
- **All-in-one development environment**: streamlines coding, data visualization, and workflow
- **Extensible**: supports R — but also Python, SQL, and Git
- **Rich community**: eases learning and problem-solving

Code formats: R Scripts vs. R Notebooks

- **R Scripts**
  - Simple: just code
  - Best for simple tasks (and multi-script pipelines)
- **R Notebooks (Quarto, R Notebook)**
  - Integrated: Mix of code, text, and outputs for easy documentation
  - Interactive: real-time code execution and output display

Quarto documents

- “Notebook” Style: supports interactive code and text
  - Code cells: segments for code execution
  - Text chunks: annotations or explanations in Markdown format.
- Inline output: figures and code output display directly below the corresponding code cell



Installing packages

- Packages: pre-built code and functions.
- Packages are generally installed from the Comprehensive R Archive Network (CRAN)

Install: new packages

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

Library: load installed packages

```
library(tidyverse)
```

YaRrr! The Pirates Guide to R. Nathaniel D. Phillips, 2018.

Running code

- Run all code in a quarto document (or R script, or R notebook)
  - Exception: install packages, quick checks in console
- To run a single line of code in a code cell
  - Cursor over line, **`Ctrl + Enter`** (Windows/Linux) or **`Cmd + Enter`** (Mac).
- To run a full code cell (or script)
  - **`Ctrl + Shift + Enter`** (Windows/Linux) or **`Cmd + Shift + Enter`** (Mac).

Live coding demo

- Demo of creating a new Quarto document and running code in a code cell
- Your turn next…

In-class exercise 0

- Create a new quarto document
  - `File -> New File -> Quarto Document -> Create`
- Create a new code cell
  - `Insert -> Executable cell -> R`
- Practice running code below

```r
3+3
```

```
[1] 6
```

```r
print("Thank you for attending the intro to R session!")
```

```
[1] "Thank you for attending the intro to R session!"
```

Module 2

R programming fundamentals

Learning objectives:

- Comprehend R objects and functions
- Master basic syntax, including comments, assignment, and operators
- Understand data structures and types in R

Objects

- Everything in R is an object
  - **Vectors**: Ordered collection of same type
  - **Data Frames**: Table of columns and rows
  - **Function**: Reusable code block
  - **List**: Ordered collection of objects

```r
## Objects in R

## Numeric like `1`, `2.5`
x <- 2.5

## Character: Text strings like `"hello"`
y <- "hello"

## Boolean: `TRUE`, `FALSE`
z <- TRUE

## Vectors
vec1 <- c(1, 2, 3)
vec2 <- c("a", "b", "c")

## data.frames
df <- data.frame(vec1, vec2)
```

Functions

- Built-in “base” functions

```
## Functions in R
result_sqrt <- sqrt(25)
result_sqrt
```

```
[1] 5
```

- Custom, user-defined functions

```
# User-Defined Functions: Custom functions
my_function <- function(a, b) {
  return(a^2 + b)
}

my_function(2, 3)
```

```
[1] 7
```

- Functions from packages

```
# User-Defined Functions: Custom functions

library(here) ## library package here
here() ## run custom "here" function to print out working directory
```

```
[1] "/Users/caseybreen/workspace/teaching/intro_r"
```

Comments

- Use # to start a single-line comment
- Comments are an important way to document code

```
## Add comments

x <- 7 # assigns 1 to x

## the line below won't assign 12 to x because it's commented out
# x <- 12

x
```

```
[1] 7
```

Assignment operators

- Use `<-` or `=` for assignment
  - `<-` is preferred and advised for readability
- Formally, assignment means "assign the result of the operation on the right to object on the left"

```
## Add comments

x <- 7 # assigns 7 to x

## Question: what does this do?
y <- x
```

Arithmetic operators

- Addition / Subtraction

```
## R as a calculator (# adds a comment)
## Addition
10 + 3
```

```
[1] 13
```

```
## Subtraction
4 - 2
```

```
[1] 2
```

- Multiplication / division

```
## Multiplication
4 * 3
```

```
[1] 12
```

```
## Division
12 / 6
```

```
[1] 2
```

- Exponents

```
## exponents
10^2 ## or 10 ** 2
```

```
[1] 100
```

Comparison and logical operators

Operators

Operator	Symbol
AND	&
OR	|
NOT	!
Equal	==
Not Equal	!=
Greater/Less Than	> or <
Greater/Less Than or Equal	>= or <=
Element-wise In	%in%

Examples

```r
## Logical operators

10 == 10
```

```
[1] TRUE
```

```r
9 == 10
```

```
[1] FALSE
```

```r
9 < 10
```

```
[1] TRUE
```

```r
"apple" %in% c("bananas", "oranges")
```

```
[1] FALSE
```

```r
"apple" %in% "bananas" | "apple" %in% "apple"
```

```
[1] TRUE
```

```r
"apple" %in% "bananas" & "apple" %in% "apple"
```

```
[1] FALSE
```

Data structures

- There are lots of data structures; we'll focus on `vectors` and `data frames`.
    - `Vectors`: One-dimensional arrays that hold elements of a single data type (e.g., all numeric or all character).
    - `Data frames`: Two-dimensional tables where each column can have a different data type; essentially a list of vectors of equal length.

`Vectors` and `data frames`

- `Vector` example

```r
## Vector Example
vec_example <- c(1, 2, 3, 4, 5)

vec_example ## prints out vec_example
```

```
[1] 1 2 3 4 5
```

- `Data frame` example

```r
# Data.frame example
example_df <- data.frame(
  ID = c(1, 2, 3, 4),
  Name = c("Alice", "Bob", "Charlie", "David"),
  Age = c(25, 30, 35, 40),
  Score = c(90, 85, 88, 76)
)

example_df ## prints out df_example
```

```
  ID    Name Age Score
1  1   Alice  25    90
2  2     Bob  30    85
3  3 Charlie  35    88
4  4   David  40    76
```

Data types

- Each `vector` or `data frame` column can only contain one data type:
  - `Numeric`: Used for numerical values like integers or decimals.
  - `Character`: Holds text and alphanumeric characters.
  - `Logical`: Represents binary values - TRUE or FALSE.
  - `Factor`: Categorical data, either ordered or unordered, stored as levels.

```
## generate vectors
vec1 <- c(1, 2, 3)
vec2 <- c("a", "b", "c")

## check type
class(vec1)
```

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

```
class(vec2)
```

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

NA (missing) values in R

- `NA` represents missing or undefined data.
  - Can vary by data type (e.g., `NA_character_` and `NA_integer_`)
- **NA** values can affect summary statistics and data visualization.
- What happens when you run the code below?

```r
vec <- c(1, 2, 3, NA)
mean(vec)
```

Generating sequences in R

- Method 1: Manually write out sequence using `c()`

```
## Basic
c(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
```

```
 [1]  1  2  3  4  5  6  7  8  9 10
```

- Method 2: Colon operator (`:`), creates sequences with increments of 1

```
c(1:10)
```

```
 [1]  1  2  3  4  5  6  7  8  9 10
```

- Method 3: **`seq()`** Function: More flexible and allows you to specify the **`start`**, **`end`**, and **`by`** parameters.

```
## seq 1-10, by = 2
seq(1, 10, by = 2)
```

```
[1] 1 3 5 7 9
```

Functions

- Function: Input arguments, performs operations on them, and returns a result
- For each of the below functions, what are the:
  - Input arguments?
  - Operations performed?
  - Results?

```r
## hint: rnorm simulates random draws from a standard normal distribution
random_draws <- rnorm(n = 5,
      mean = 0,
      sd = 1)

## find the mean
mean(random_draws)
```

```
[1] 0.2525844
```

```r
## find the median
median(random_draws)
```

```
[1] 0.1229782
```

```r
## find the standard deviation
sd(random_draws)
```

```
[1] 1.238935
```

Keyboard shortcuts

Insert new code cell

- macOS: `Cmd` + `Option` + `I`
- Windows/Linux: `Ctrl` + `Alt` + `I`

Run full code cell or script

- macOS: `Cmd` + `Shift` + `Enter`
- Windows/Linux: `Ctrl` + `Shift` + `enter`

Assignment operator (creates `<-`)

- macOS: `option` + `-`
- Windows/Linux: `option` + `-`

Live coding demo

- Assignment (e.g., `x <- 4`)
- Logical expressions (e.g., `x > 10`)
- Creating a basic sequence
- Your turn next…

In-class exercise 1

1. Assign `x` and `y` to take values 3 and 4.
2. Assign `z` as the product of `x` and `y`.
3. Write code to calculate the square of 3. Assign this to a variable `three_squared`.
4. Write a logical expression to check if `three_squared` is greater than 10.
5. Write a logical expression testing whether `x` is *not* greater than 10. Use the `negate` symbol (`!`).

Exercise 1 solutions

1. Assign `x` and `y` to take values 3 and 4.

```
x <- 3
y <- 4
```

2. Assign `z` as the product of `x` and `y`.

```
z <- x * y
```

3. Calculate the square of 3 and assign it to a variable called `three_squared`.

```
three_squared <- 3^2
```

4. Write a logical expression to check if `three_squared` is greater than 10.

```
three_squared > 10
```

```
[1] FALSE
```

5. Write a logical expression to test whether `three_squared` is *not* greater than 10. Use the `negate` symbol (`!`).

```
!three_squared > 10
```

```
[1] TRUE
```

In-class exercise 2

1. Generate vectors containing the numbers 100, 101, 102, 103, 104, and 105 using 3 different methods (e.g., **`c()`**, **`seq()`**, **`:`**). In what scenarios might each method be most convenient?
2. Generate a sequences of all **even** numbers between 0 and 100. Use the `seq()` function.
3. Create a descending sequence of numbers from 100 to 1, and assign it to a variable. Use the `seq()` function.

Exercise 2 solutions

1. Generate vectors containing the numbers 100 to 105 using three different methods (`c()`, `seq()`, `:`). Discuss the convenience of each method.

```r
# Generate a vector using c() method
vector_c <- c(100, 101, 102, 103, 104, 105)

# Generate a vector using seq() method
vector_seq <- seq(100, 105, by = 1)

# Generate a vector using : operator
vector_colon <- c(100:105)
```

2. Generate a sequence of all even numbers between 0 and 100. Use the `seq()` function.

```r
# Generate a sequence of all even numbers between 0 and 100
even_seq <- seq(0, 100, by = 2)
```

3. Create a descending sequence of numbers from 100 to 1, and assign it to a variable. Use the `seq()` function.

```r
# Create a descending sequence of numbers from 100 to 1
desc_seq <- seq(100, 1, by = -1)
```

Module 3

Working with `vectors` and `data frames`

Learning objectives

- Select elements from `vectors` and columns from `data frames`
- Subset `data frames`
- Investigate characteristics of `data frames`

Indexing vectors

- Basic indexing

```r
vec <- c(1, 2, 3, 4, 5)
first_element <- vec[1]
first_element
```

```
[1] 1
```

```r
third_element <- vec[3]
third_element
```

```
[1] 3
```

- Conditional indexing

```r
vec <- seq(5, 33, by = 2)
vec[vec > 25]
```

```
[1] 27 29 31 33
```

Working with `data frames`

- `Data frames` are the most common and versatile data structure in `R`
- Structured as rows (observations) and columns (variables)

```r
test_scores <- data.frame(
  id = c(1, 2, 3, 4, 5),
  name = c("Alice", "Bob", "Carol", "Dave", "Emily"),
  age = c(25, 30, 22, 28, 24),
  gender = c("F", "M", "F", "M", "F"),
  score = c(90, 85, 88, 92, 89)
)

knitr::kable(test_scores)
```

id	name	age	gender	score
1	Alice	25	F	90
2	Bob	30	M	85
3	Carol	22	F	88
4	Dave	28	M	92
5	Emily	24	F	89

Working with `data frames`

- `head()` - looks at top rows of the `data frame`
- `$` operator - access a column as a `vector`

```
## print first two rows  first row
head(test_scores, 2)
```

```
  id  name age gender score
1  1 Alice  25      F    90
2  2   Bob  30      M    85
```

```
## access name column
test_scores$name
```

```
[1] "Alice" "Bob"   "Carol" "Dave"  "Emily"
```

Subsetting `data frames`

- **Methods:**
  - `$`: Single column by name.
  - `df[i, j]`: Row `i` and column `j`.
  - `df[i:j, k:l]`: Rows `i` to `j` and columns `k` to `l`.
- **Conditional Subsetting:** `df[df$age > 25, ]`.

```
## all rows, columns 1-3
test_scores[,1:3]
```

```
  id  name age
1  1 Alice  25
2  2   Bob  30
3  3 Carol  22
4  4  Dave  28
5  5 Emily  24
```

```
## all columns, rows 4-5
test_scores[4:5,]
```

```
  id  name age gender score
4  4  Dave  28      M    92
5  5 Emily  24      F    89
```

Quiz

Which rows and will this return?

```
test_scores[1:3,]
```

- Which rows and which columns will this return?

```
test_scores[test_scores$score >= 90, ]
```

Answers

```r
test_scores[1:3,]
```

```
  id  name age gender score
1  1 Alice  25      F    90
2  2   Bob  30      M    85
3  3 Carol  22      F    88
```

```r
test_scores[test_scores$score >= 90, ]
```

```
  id  name age gender score
1  1 Alice  25      F    90
4  4  Dave  28      M    92
```

Explore `data frame` characteristics

Check number of rows

```r
## check number of rows (observations)
nrow(test_scores)
```

```
[1] 5
```

Check number of columns

```r
## check number of columns (variables)
ncol(test_scores)
```

```
[1] 5
```

Check column names

```r
names(test_scores)
```

```
[1] "id"     "name"   "age"    "gender" "score"
```

Live coding demo

- Generate random draws from a normal distribution using the `rnorm` function
- Subset the vector of random draws to only include certain observations
- Look at basic summary statistics

In-class exercise 3

1. Generate a `vector` of 100 observations drawn from a normal distribution with a mean of 10 and a standard deviation of 2. Use the `rnorm` function.
2. What are the 1st, 10th, and 100th elements of this `vector`?
3. Calculate the mean of this `vector`. How does this `sample` mean relate to the `population` mean (hint: population mean = 10) of the distribution?
4. Calculate the difference between the sample mean and the population mean. Discuss the reason for the discrepancy.
5. Repeat steps 1-4 with a new sample size of 10,000. Did the difference between the sample mean and the population mean decrease? Why?

Exercise 3 solutions

```r
# Generate a sample of 1,000 draws from a normal distribution with mean = 10 and sd = 2
sample_data_100 <- rnorm(100,
                     mean = 10,
                     sd = 2)

## look at 1st, 10th, and 100th element
sample_data_100[c(1, 10, 100)]
```

```
[1]  8.394597  9.526573 10.031163
```

```r
# Calculate the mean of this sample
sample_mean_100 <- mean(sample_data_100)

# Calculate the difference between the mean of the sample and the expected value of the mean
difference_100 <- abs(sample_mean_100 - 10)

difference_100
```

```
[1] 0.07207561
```

```r
# Calculate the Z-score for the sample mean
sample_data_10000 <- rnorm(10000,
                     mean = 10,
                     sd = 2)

# Calculate the mean of this sample
sample_data_10000 <- mean(sample_data_10000)

# Calculate the difference between the mean of the sample and the expected value of the mean
sample_data_10000 <- abs(sample_data_10000 - 10)

sample_data_10000
```

```
[1] 0.01031513
```

Questions?

- Thanks for your attendance and participation
- Please independently complete all exercises in problem set 1 (and review solutions)
- Questions: casey.breen@demography.ox.ac.uk