Introduction to R Session 1

Department of Sociology | University of Oxford

Casey Breen

2023-10-08

Welcome to "Intro to R"

- Two sessions:
  - Thursday, October 5th, 1pm - 4pm
  - Friday, October 6th, 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
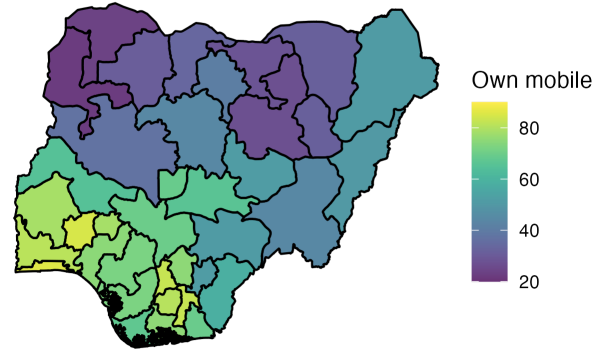
  - Download: http://www.rstudio.com/download

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
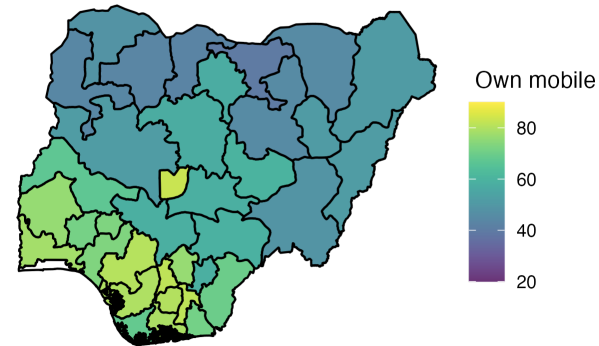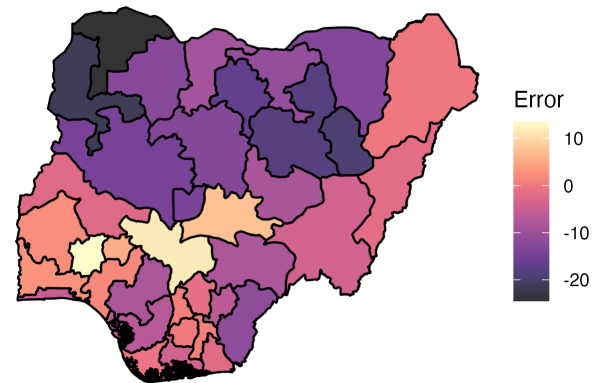
Data visualization
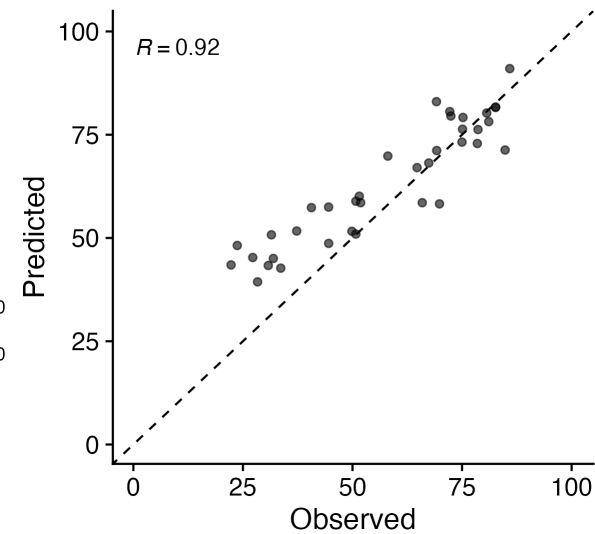
Easy to simulate + plot data

```r
1  # Generate random data for x
2  x <- rnorm(n = 3000)
3  y <- 0.8 * x + rnorm(3000, 0, sqrt(1 - 0.8^2))
4
5  # Create data.frame
6  data_df <- data.frame(x = x, y = y)
7
8  # Generate visualization
9  data_df %>%
10   ggplot(aes(x = x, y = y)) +
11   geom_point(alpha = 0.1) +
12   theme_classic()
```
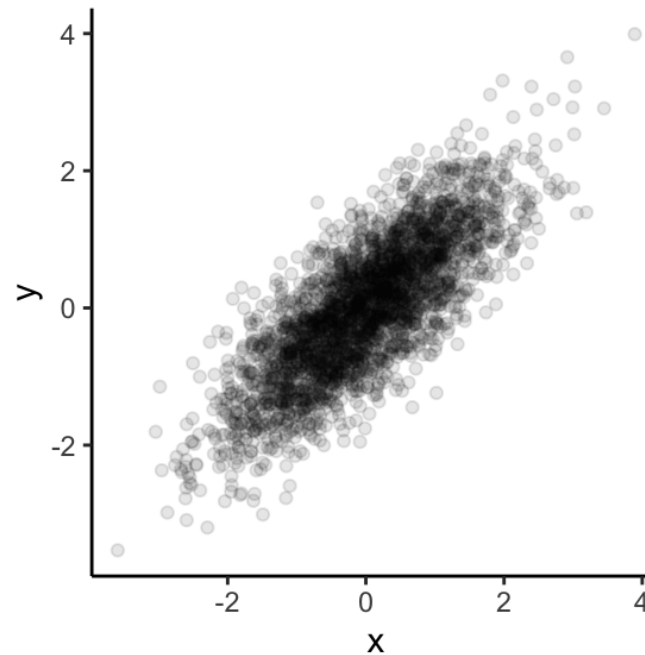
RStudio panes
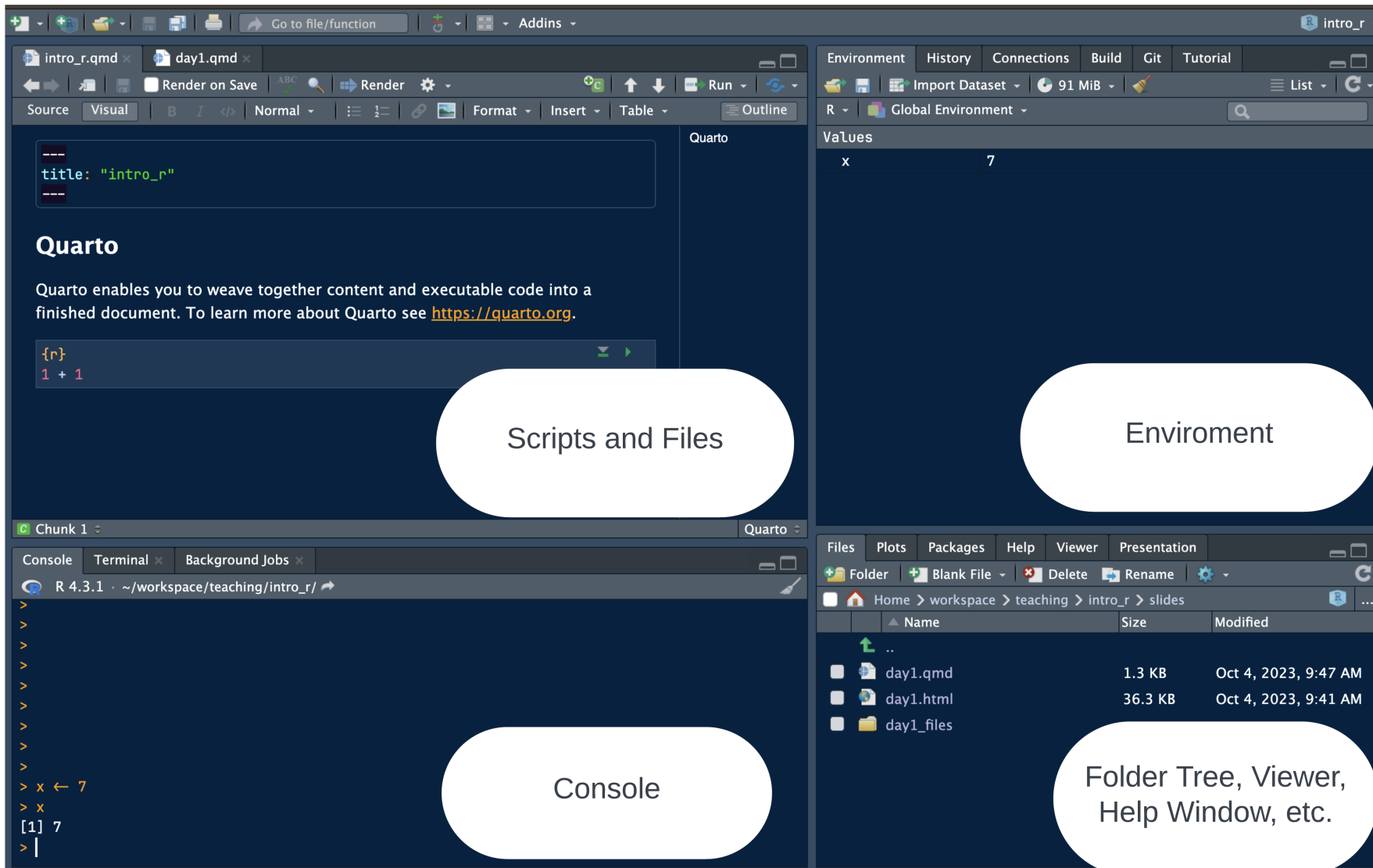
Why `RStudio`?

- **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
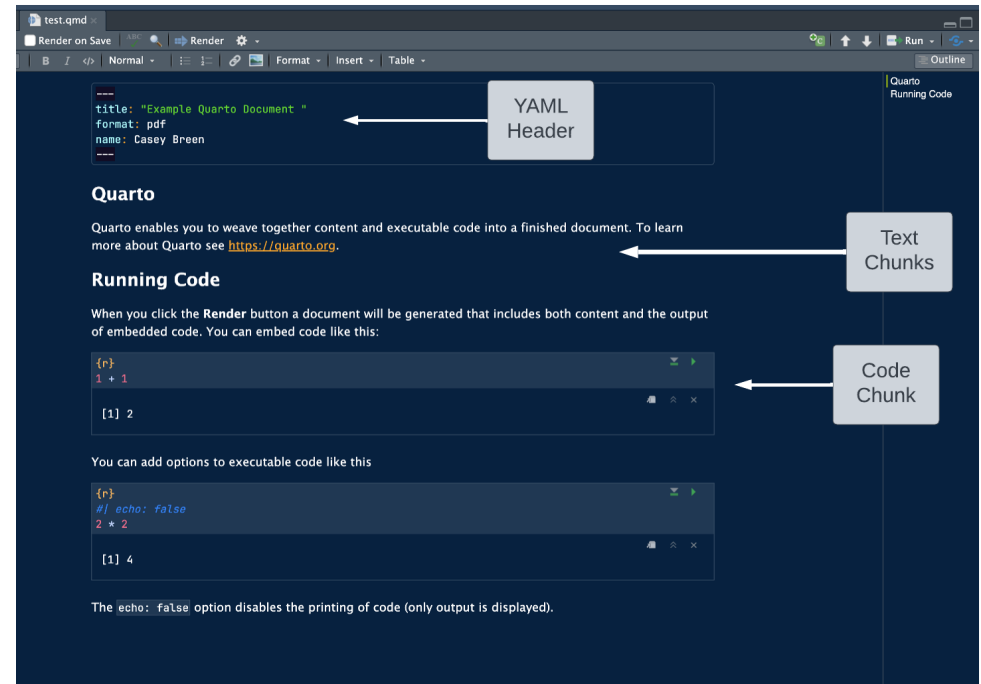
Installing packages

- Packages: pre-built code and functions.

- Packages are generally installed from the Comprehensive R Archive Network (CRAN)

Install: new packages

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

Library: load installed packages

```
1  library(tidyverse)
```


Installing a package
```
install.packages('my.package')
```

Loading a package
```
library('mypackage')
```
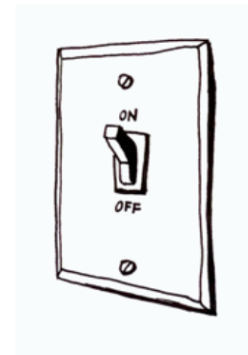
YaRrr! The Pirates Guide to R. Nathaniel D. Phillips, 2018.

16

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

```
1   3+3
```
```
[1] 6
```
```
1   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
1  ## Objects in R
2
3  ## Numeric like `1`, `2.5`
4  x <- 2.5
5
6  ## Character: Text strings like `"hello"`
7  y <- "hello"
8
9  ## Boolean: `TRUE`, `FALSE`
10 z <- TRUE
11
12 ## Vectors
13 vec1 <- c(1, 2, 3)
14 vec2 <- c("a", "b", "c")
15
16 ## data.frames
17 df <- data.frame(vec1, vec2)
```

Functions

- Built-in "base" functions

```r
1  ## Functions in R
2  result_sqrt <- sqrt(25)
3  result_sqrt
```

```
[1] 5
```

- Custom, user-defined functions

```r
1  # User-Defined Functions: Custom functions
2  my_function <- function(a, b) {
3    return(a^2 + b)
4  }
5
6  my_function(2, 3)
```

```
[1] 7
```

- Functions from packages

```r
1  # User-Defined Functions: Custom functions
2
3  library(here) ## library package here
4  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

```
1  ## Add comments
2
3  x <- 7 # assigns 1 to x
4
5  ## the line below won't assign 12 to x because it's commented out
6  # x <- 12
7
8  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"

```
1  ## Add comments
2
3  x <- 7 # assigns 7 to x
4
5  ## Question: what does this do?
6  y <- x
```

Arithmetic operators

- Addition / Subtraction

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

```
[1] 13
```

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

```
[1] 2
```

- Multiplication / division

```
1  ## Multiplication
2  4 * 3
```

```
[1] 12
```

```
1  ## Division
2  12 / 6
```

```
[1] 2
```

- Exponents

```
1  ## exponents
2  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

```
1  ## Logical operators
2
3  10 == 10
```
```
[1] TRUE
```
```
1  9 == 10
```
```
[1] FALSE
```
```
1  9 < 10
```
```
[1] TRUE
```
```
1  "apple" %in% c("bananas", "oranges")
```
```
[1] FALSE
```
```
1  "apple" %in% "bananas" | "apple" %in% "apple"
```
```
[1] TRUE
```
```
1  "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

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

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

- **Data frame** example

```
1  # Data.frame example
2  example_df <- data.frame(
3    ID = c(1, 2, 3, 4),
4    Name = c("Alice", "Bob", "Charlie", "David"),
5    Age = c(25, 30, 35, 40),
6    Score = c(90, 85, 88, 76)
7  )
8
9  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.

```r
1  ## generate vectors
2  vec <- c(1, 2, 3)
3  vec1 <- c("a", "b", "c")
4
5  ## check type
6  class(vec1)
```

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

```r
1  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()`

```
1  ## Basic
2  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

```
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.

```
1  ## seq 1-10, by = 2
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?

```
1  ## hint: rnorm simulates random draws from a standard normal distribution
2  random_draws <- rnorm(n = 5,
3         mean = 0,
4         sd = 1)
5
6  ## find the mean
7  mean(random_draws)
```

```
[1] 0.3084989
```

```
1  ## find the median
2  median(random_draws)
```

```
[1] 0.3596781
```

```
1  ## find the standard deviation
2  sd(random_draws)
```

```
[1] 0.8140028
```

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.

```
1   x <- 3
2   y <- 4
```

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

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

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

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

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

```
1   three_squared > 10
```
```
[1] FALSE
```

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

```
1   !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
1  # Generate a vector using c() method
2  vector_c <- c(100, 101, 102, 103, 104, 105)
3
4  # Generate a vector using seq() method
5  vector_seq <- seq(100, 105, by = 1)
6
7  # Generate a vector using : operator
8  vector_colon <- c(100:105)
```

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

```r
1  # Generate a sequence of all even numbers between 0 and 100
2  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
1  # Create a descending sequence of numbers from 100 to 1
2  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
1  vec <- c(1, 2, 3, 4, 5)
2  first_element <- vec[1]
3  first_element
```

```
[1] 1
```

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

```
[1] 3
```

- Conditional indexing

```r
1  vec <- seq(5, 33, by = 2)
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`

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

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

```
1  ## access name column
2  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, ]`**.

```
1  ## all rows, columns 1-3
2  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
```

```
1  ## all columns, rows 4-5
2  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?

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

- Which rows and which columns will this return?

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

Answers

```
1  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
```

```
1  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

```
1  ## check number of rows (observations)
2  nrow(test_scores)
```

```
[1] 5
```

Check number of columns

```
1  ## check number of columns (variables)
2  ncol(test_scores)
```

```
[1] 5
```

Check column names

```
1  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
1  # Generate a sample of 1,000 draws from a normal distribution with mean = 10 and sd = 2
2  sample_data_100 <- rnorm(100,
3                           mean = 10,
4                           sd = 2)
5
6  ## look at 1st, 10th, and 100th element
7  sample_data_100[c(1, 10, 100)]
```
```
[1] 10.14126 10.86688 13.95133
```

```r
1  # Calculate the mean of this sample
2  sample_mean_100 <- mean(sample_data_100)
3
4  # Calculate the difference between the mean of the sample and the expected value of the mean
5  difference_100 <- abs(sample_mean_100 - 10)
6
7  difference_100
```
```
[1] 0.02145836
```

```r
1   # Calculate the Z-score for the sample mean
2   sample_data_10000 <- rnorm(10000,
3                              mean = 10,
4                              sd = 2)
5
6   # Calculate the mean of this sample
7   sample_data_10000 <- mean(sample_data_10000)
8
9   # Calculate the difference between the mean of the sample and the expected value of the mean
10  sample_data_10000 <- abs(sample_data_10000 - 10)
11
12  sample_data_10000
```
```
[1] 0.01288612
```

Questions?

- Thanks for your attendance and participation

- Please independently complete all exercises in problem set 2 (and review solutions)

- Questions: casey.breen@sociology.ox.ac.uk