Introduction to R

Department of Sociology, University of Oxford

Casey Breen

2023-10-05

Welcome to “Intro to R”

- Two sessions:
  - Thursday, 1pm - 4pm
  - Friday, 9:30am - 12:30pm
- Course materials available from:
  - www.github.com/caseybreen/intro_r

Course goals

- Why R is a powerful tool for social science research
- Install R and `RStudio`
- Introduction to R syntax and data types
- Basic understanding of data manipulation + visualization

Course agenda

- **Session 1**
  - Introduction + installing R and `RStudio`
  - Overview of `RStudio` interface + R scripts, notebooks, quarto
  - Basic syntax and data types
  - Data import and export
- **Session 2**
  - Data manipulation (`dplyr`)
  - Data visualization (`ggplot2`)
  - Best practices: coding style, commenting, and documentation
  - Resources for self-teaching

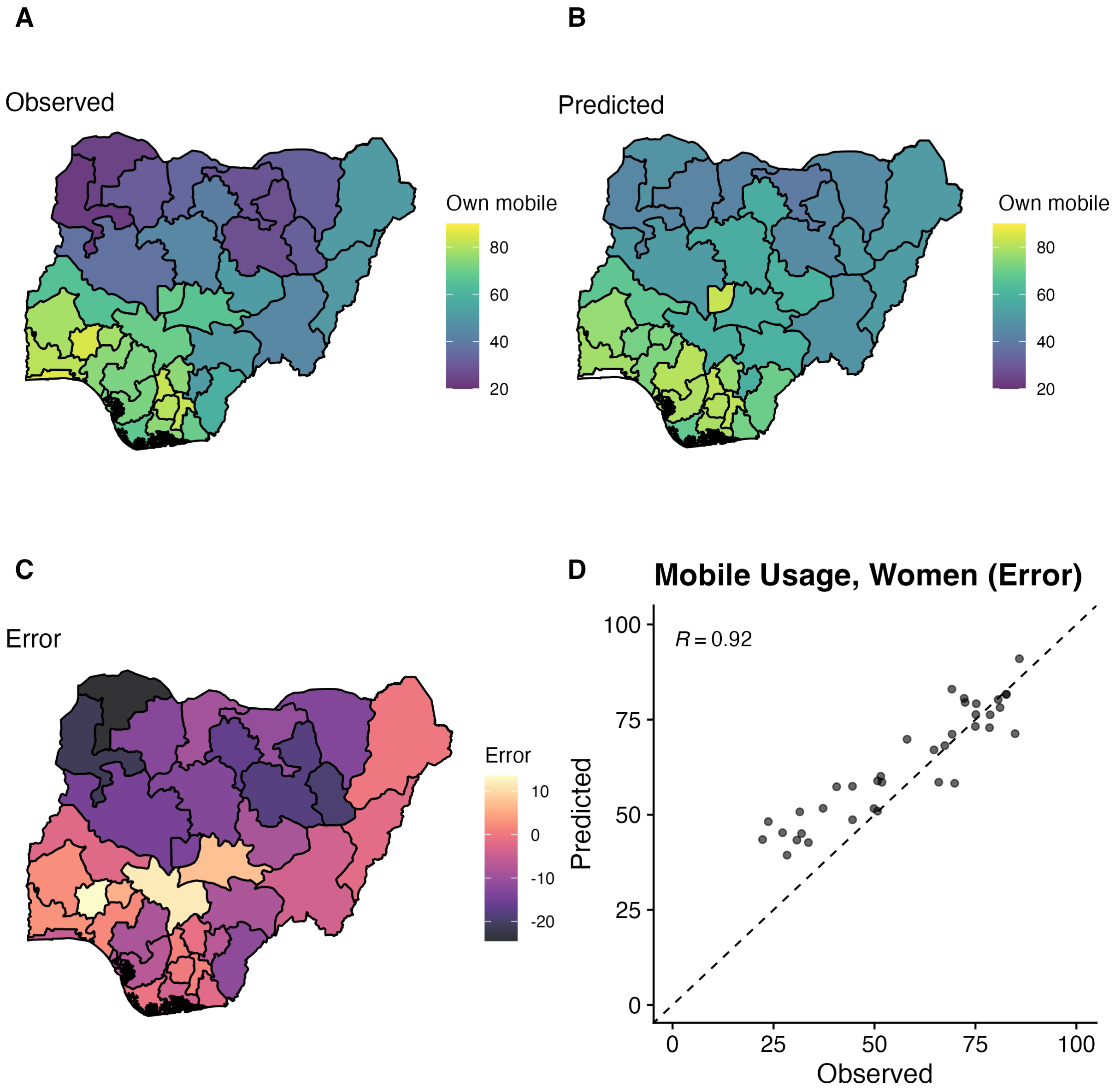
R and RStudio

- 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?

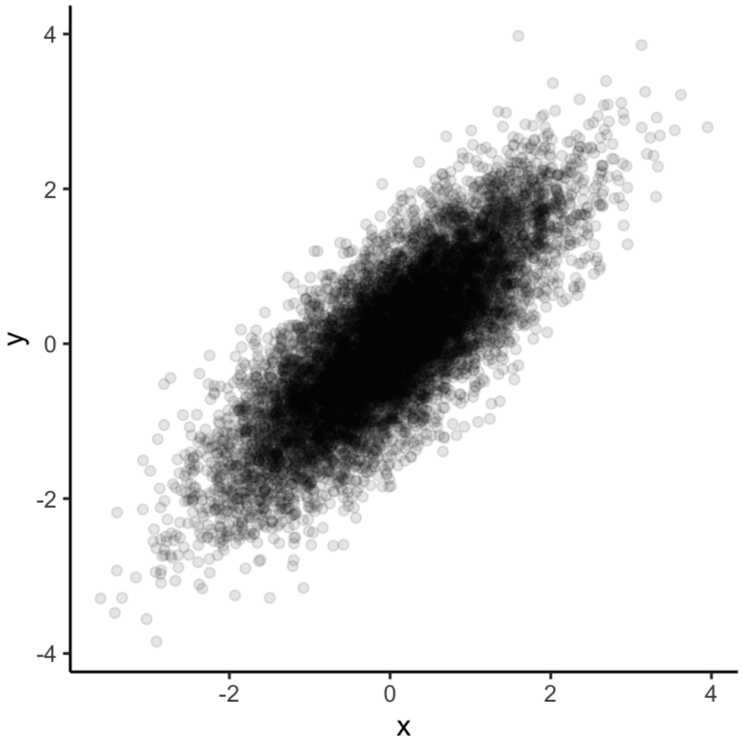
- Free, open source — great for reproducibility and open science
- Powerful language for data manipulation, statistical analysis
- Publication-ready data visualizations
- Well supported, excellent community

Data visualization

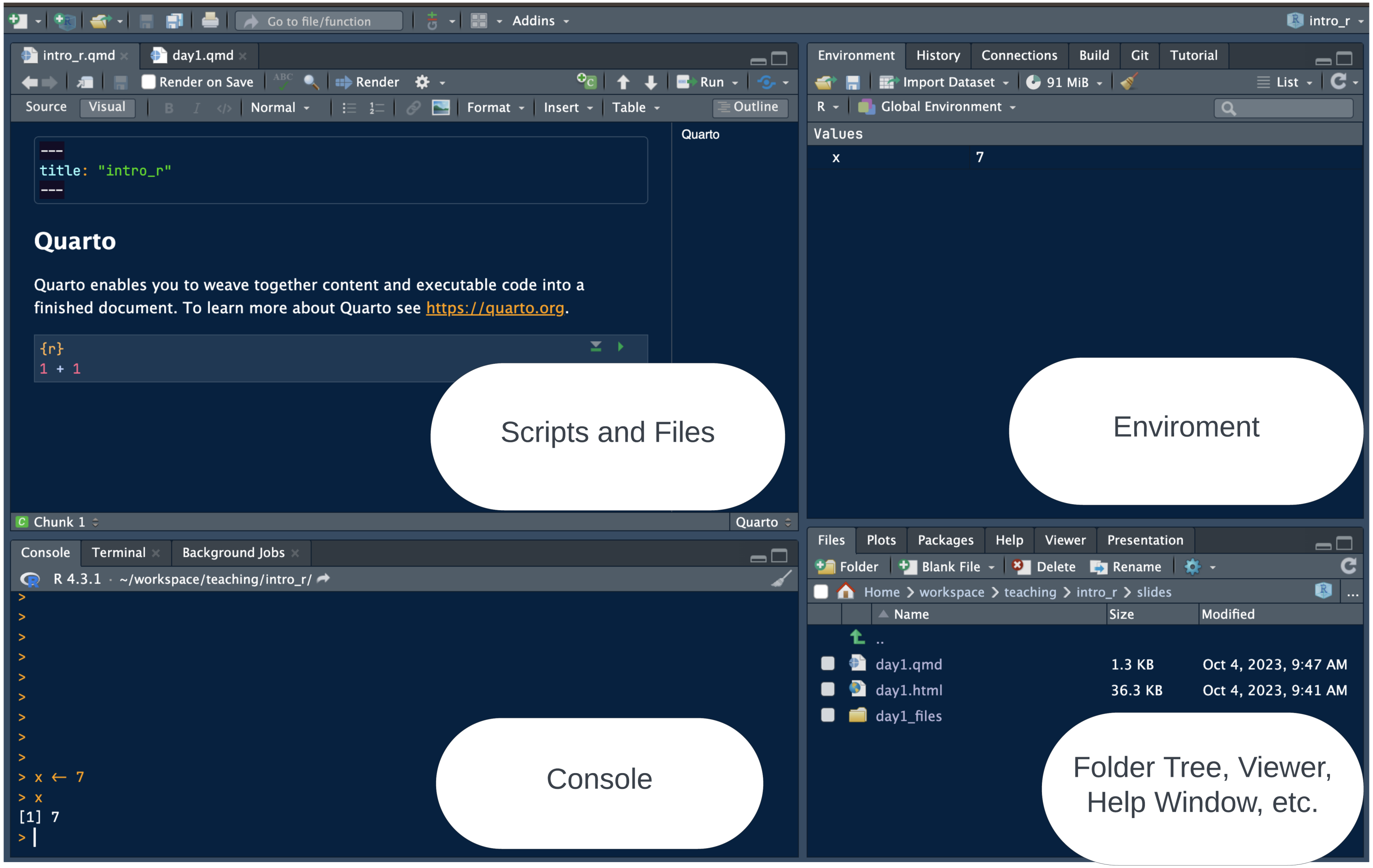


Easy to simulate + plot data

```r
# Generate random data for x
x <- rnorm(n = 10000)
y <- 0.8 * x + rnorm(10000, 0, sqrt(1 - 0.8^2))
# Create data.frame
data_df <- data.frame(x = x, y = y)
# Generate df
data_df %>%
  ggplot(aes(x = x, y = y)) +
  geom_point(alpha = 0.1) +
  theme_classic()
```



RStudio Panes



```
---
title: "intro_r"
---
```

Quarto

Quarto enables you to weave together content and executable code into a finished document. To learn more about Quarto see https://quarto.org.

```
{r}
1 + 1
```

Values

```
x        7
```

```
>
>
>
>
>
>
>
>
>
> x ← 7
> x
[1] 7
>
```

	Name	Size	Modified
	..		
□	day1.qmd	1.3 KB	Oct 4, 2023, 9:47 AM
□	day1.html	36.3 KB	Oct 4, 2023, 9:41 AM
□	day1_files		

Scripts and Files

Enviroment

Console

Folder Tree, Viewer, Help Window, etc.

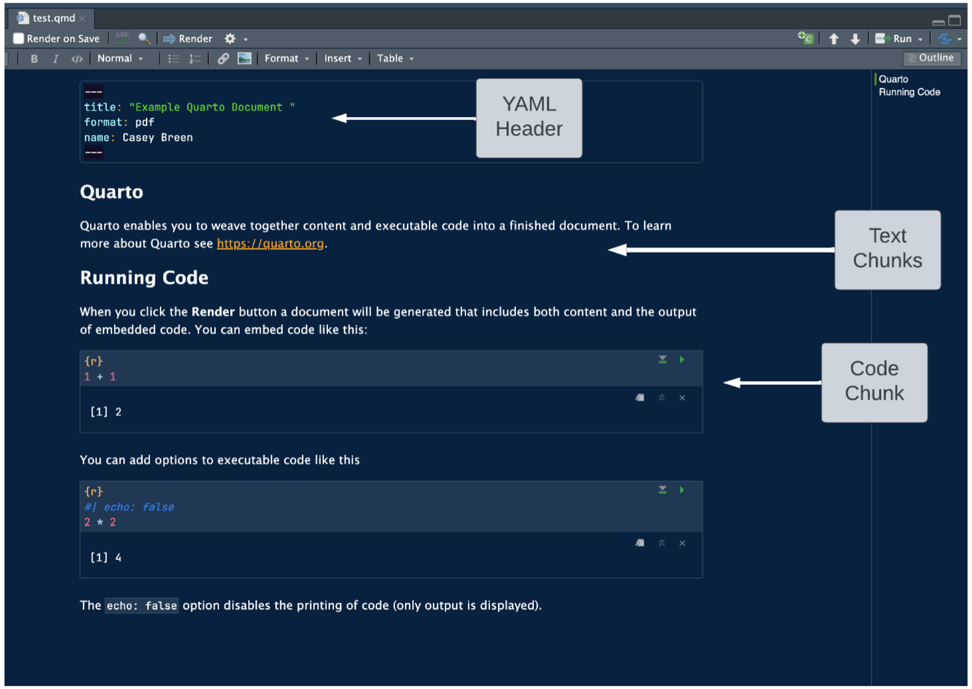
R Scripts, R-Notebooks

- **Scripts**:
  - Just code
  - Ideal for simple tasks (and multi-script pipelines)
- **Notebooks** (Quarto, R Notebook):
  - Integrated code, text, and outputs (great for documentation!)
  - Interactive
  - We will focus on notebooks

Quarto Document

- Notebook-Style Layout: Supports interactive code and text chunks.
    - Code Chunks: Segments for code execution
    - Text Chunks: Annotations or explanations in Markdown format.

- Inline Output: Figures and code output display directly below the corresponding code chunk



```
---
title: "Example Quarto Document "
format: pdf
name: Casey Breen
---
```

Quarto

Quarto enables you to weave together content and executable code into a finished document. To learn more about Quarto see https://quarto.org.

Running Code

When you click the **Render** button a document will be generated that includes both content and the output of embedded code. You can embed code like this:

```
{r}
1 + 1
```

```
 [1] 2
```

You can add options to executable code like this

```
{r}
#| echo: false
2 * 2
```

```
 [1] 4
```

The `echo: false` option disables the printing of code (only output is displayed).

Live Coding Session 1: Creating new Quarto file

- I’ll demo first; please pay attention

You turn

- Please create a quarto document
- You can use this document for the rest of this session
- Add a new code chunk
    - click + point: `Insert -> Executable cell -> R`
    - macOS: `Cmd` + `Option` + `I`
    - Windows/Linux: `Ctrl` + `Alt` + `I`

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

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

```
[1] 5
```

- Custom, user-defined functions

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

my_function(2, 3)
```

```
[1] 7
```

- Functions from packages

Installing packages

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

Install: new packages

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

Library: load installed packages

```r
library(tidyverse)
```

Installing a package

```
install.packages(‘my.package’)
```

Loading a package

```
library(‘mypackage’)
```

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

Running code

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

Your Turn

- Create a new code cell in `quarto` document
    - (`insert -> executable cell -> R`)
- Run each line one at a time
- Run the full code chunk

```r
x <- 1

y <- (x + 1)^3

cat("Thank you for attending R session number", x, "!")
```

```
Thank you for attending R session number 1 !
```

Break

10 minute break

Basic Syntax: assignment

- 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”

```r
## Add comments

x <- 7 # assigns 7 to x

## Quesiton: what does this do?
y <- x <- 25
```

Basic Syntax: comments

- Use # to start a single-line comment
- Include lots of comments when you’re writing code

```r
## Add comments

x <- 7 # assigns 1 to x
x <- 12
```

Basic syntax: operators

```r
## R as a calculator (# adds a comment)
3 * 3
```

```
[1] 9
```

```r
## Division
12/4
```

```
[1] 3
```

```r
## Subtraction
100-12
```

```
[1] 88
```

```r
## Exponents (10^2)
10 ** 2
```

```
[1] 100
```

Basic syntax: comparisons

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

```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)

print(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)
)
```

ID	Name	Age	Score
1	Alice	25	90
2	Bob	30	85
3	Charlie	35	88

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.

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

## check type
class(vec1)
```

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

```r
class(vec2)
```

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

Generating Sequences in R

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

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

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

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

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

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

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

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

Basic 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
## generate random draws from a standard normal distribution
random_draws <- rnorm(n = 5,
      mean = 0,
      sd = 1)

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

```
[1] -0.0343441
```

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

```
[1] -0.1551556
```

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

```
[1] 0.6595093
```

In-class exercise 1

1. Assign `x` and `y` to take values 3 and 4.
2. Create a new variable `z` as the product of variables `x` and `y.`
3. Write code to calculate the square of 3. Assign this to a variable `three_squared`.
4. Write a logical expression on whether `x` is greater than 10. When might you need to filter data based on a condition?
5. Write a logical expression testing whether `x` is *not* greater than 10.

Exercise 1 solutions

```r
# Assign x and y to take values 3 and 4
x <- 3
y <- 4

# Create a new variable z as the product of variables x and y
z <- x * y

# Write code to calculate the square of 3 and assign it to a variable three_squared
3^2
```

```
[1] 9
```

```r
# Write a logical expression to check if x is greater than 10
x > 10
```

```
[1] FALSE
```

```r
# Write a logical expression to check if x is not greater than 10
x <= 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.

3. Create a descending sequence of numbers from 100 to 1, and assign it to a variable.

Exercise 2 solutions

```r
# Generate a vector using c() method
vector_c <- c(100, 101, 102, 103, 104, 105)
# Use when numbers are not in a simple sequence or pattern

# Generate a vector using seq() method
vector_seq <- seq(100, 105, by = 1)
# Use when numbers follow a pattern but not necessarily just increment by 1

# Generate a vector using : operator
vector_colon <- c(100:105)
# Use when numbers increment by 1

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

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

In-class exercise 3

1. Generate a sample of 100 observations drawn from a normal distribution with a mean of 10 and a standard deviation of 2. How is this type of random sampling useful in statistical analysis?
2. Calculate the mean of this generated sample. How does this sample mean relate to the population mean of the distribution?
3. Calculate the difference between the sample mean and the population mean. Why the discrepancy?
4. Repeat steps 1--3 with a sample of 10,000. Did the difference between the sample mean and the population mean decrease? Will this always be the case?

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)

# 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] 0.2247292
```

```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] 0.01523271
```

Break

- 10 minutes
- Tea + cake

Indexing vectors

- Basic indexing, specify position

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

- Conditional indexing, specify position

```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`
- `Data frames` are 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`

```r
## 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
```

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

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

```r
## 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
```

```r
## 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
```

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, ]`.

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

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

```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"
```

Reading in data

Common Formats

- CSV, Excel, TXT

Key Functions

- **`read.csv()`**: Read CSV files
  - Faster alternatives: `read_csv` from `tidyverse` and `fread()` from `data.table`
- **`read.table()`**: Read text files
- **`readxl::read_excel()`**: Read Excel files

```r
## read in CSV file
df <- read.csv("/path/to/your/data.csv")
df <- read_csv("/path/to/your/data.csv") ## faster


## read in stata file
library(haven)
data <- read_dta("path/to/file.dta")
```

In-class Exercise 4

- Let’s work with a real-world, social science dataset
  - CenSoc-Numident, individual-level mortality dataset
  - https://shorturl.at/gnBQS
- Please download the CenSoc-Numident Demo file (.csv) and code (pdf) bookfrom the Harvard DataVerse

In-class Exercise 4 (cont.)

1. Install and library the `tidyverse` package.
2. Read in the dataset using `read_csv()` from the `tidyverse` package.
3. How many columns does that dataset have?
4. How many rows the dataset have?
5. What are the column names? What type of research question could we use this dataset for?

Exercise 4 Solutions

```r
1. ## install packages
install.packages(tidyverse)

## library tidyverse
library(tidyverse)

2. ## read in data
censoc_numident <- read_csv("/path/to/censoc_numident_demo_dataset.csv")

3. ## nrows
nrow(censoc_numident)

4. ## ncols
ncols(censoc_numident)

5. ## column names
names(censoc_numident)
```

Thank you

- Session tomorrow: 9:30am – 12:30pm
- Please try to finish exercises in advance
- Questions: casey.breen@sociology.ox.ac.uk