THE LONDON SCHOOL OF ECONOMICS AND POLITICAL SCIENCE

Week 1: The Computational Core

MY472: Data for Data Scientists
https://lse-my472.github.io/

Autumn Term 2025

Ryan Hübert
Associate Professor of Methodology

Slides last updated: 1 October 2025

Outline

1. What is this course about?
2. Administration and logistics
3. Computational landscape
4. Programming for data science
5. Directory management in R
6. Version control with Git

80/20 rule of data science:
80% data preparation, 20% data analysis

MY472 is about the 80%

Course philosophy

This course will teach you how to *prepare data* using R

- "Prepare" = collect and wrangle/munge

Not many classes like this—I think of it as the *missing semester of your data science education* (inspiration from here)

Who is this course for?

- Two audiences: research-oriented and "industry" bound
- Wide range of skill-sets are welcome

What this course is not:

- Introductory R (or programming) course
- Course on *data analysis*

Course philosophy

How to learn in this course?

- Lectures: conceptual foundations and simple examples
- Seminars: review and write code
- Formative assessments: build your own code

This is a graduate course so **we will move fast**

Supplement your learning with:

- Readings and web searches
- Generative AI assistants (MY472 takes position 3)
- Office hours and Slack

Course philosophy

How does this course fit into the new AI landscape?

- Many people say “vibe coding” will make coding obsolete
- It is true that LLMs are amazing resources for data scientists
  - We have added a new unit on LLMs for this reason!
- But learning to code isn’t just rote memorisation!
  - Learning to code is about learning how to (1) communicate with computers, and (2) think in a structured way
  - This is as true now as it was 10 years ago
- Faculty members in this department talk to people in industry, and they do not want to hire vibecoders!

Course outline

1. The Computational Core
2. Fundamentals of Digital Data
3. Tabular Data
4. Visualisation
5. Accessing Data in the Cloud
6. Reading week (no lecture or seminars)
7. Scraping Static Webpages
8. Scraping Dynamic Webpages
9. Audio and Image Processing
10. Text as Data
11. Large Language Models

Outline

Teaching team

Ryan Hübert
Associate Professor
Course convenor, lecturer
& seminar teacher

Yuanmo He
LSE Fellow
Seminar teacher

Thomas Robinson
Assistant Professor
Seminar teacher

Salvatore Finizio
PhD Student
GTA & seminar teacher

Charlotte Kuberka
PhD Student
GTA & seminar teacher

Asami Narita
PhD Student
GTA

Teaching team

Ryan Hübert
Associate Professor
Course convenor, lecturer
& seminar teacher

Yuanmo He
LSE Fellow
Seminar teacher

Thomas Robinson
Assistant Professor
Seminar teacher

Salvatore Finizio
PhD Student
GTA & seminar teacher

Charlotte Kuberka
PhD Student
GTA & seminar teacher

Asami Narita
PhD Student
GTA

Requirements

- You should bring your laptop to all lectures and seminars
  - Please be sure your laptop is suitable for this course
  - We're very helpful, but there are limits—we can't be tech support (your computer builds are all different!)
- You must have completed the R preparatory course
  - `https://moodle.lse.ac.uk/course/view.php?id=8669`
- Required software to be installed before first seminar:
  - R – Install from `https://www.r-project.org/`
  - Positron – Install from `https://positron.posit.co/`
  - Git – Install from here or here
  - Also create a GitHub account with education benefits

Drop-in tech help sessions

The course GTAs will hold six hours of **drop-in tech help sessions**, which you can attend if you:

- are having trouble installing required software
- have other (course-related) computer issues/questions

Sessions:

- GTA **Salvo Finizio**:
  - Thursday, 2 October 12:00-13:30 in CON.2.02
  - Friday, 3 October 12:00-13:30 in CON.2.02
- GTA **Charlotte Kuberka**:
  - Tuesday, 7 October 16:00-17:30 in CON.2.02
  - Wednesday, 8 October 16:00-17:30 in CON.2.02

Onboarding survey—complete today or tomorrow!

Full link: https://forms.office.com/e/vyaBW1D6mu

Course structure

Meetings:

- Weekly lectures (2 hours), except week 6
- Weekly seminars/labs (1.5 hours), except week 6
- You must attend your assigned seminar every week
- **If you become ill**: please limit your exposure to others as much as possible & take precautions (mask, wash hands, etc.)

Formative assessments:

- Exercises completed in seminars, with solutions provided
- Independent data science project

Course structure

Summative assessment (100%): in-person practical test resembling coding test for data science job interview

- Takes place during week 11 seminars (11th December)
- Lasts two hours: week 11 seminars are 30 minutes longer
- You will take the exam on a School-issued MacBook computer with a standardised build and limited internet access
- No extensions or alternative dates will be allowed
- If you have a School disability accommodation, please contact me by end of week 2

More assessment details will be announced in due course

Materials and resources

- Course website: https://lse-my472.github.io/
  - Review very carefully!
  - Will post slides, code, data and exercises on website, plus other resources and readings
- Slack: for announcements and communications
  - Complete onboarding survey ASAP for access
  - Please ensure you get notifications
  - Please be professional (and sparing) with your messages
- Moodle: for miscellaneous
- GitHub Classroom: for submitting work
- Office hours: book at https://studenthub.lse.ac.uk/.

Take your learning seriously

You have a unique opportunity to learn new skills and ways of thinking from experts in a specially curated learning environment

But **learning is difficult**, so please take the process seriously

- Use generative AI to aide your learning, not bypass it
  - MY472 takes position 3 on generative AI
  - But, using AI to do your work for you is *always* academic misconduct unless you were explicitly asked to do it
  - It's noticeable when someone relies too heavily on AI
- Review School guidelines on academic conduct
- Smartphones are addictive and not conducive to learning—please do not feed your addiction during class

Outline


Boomers
Gen Z
Gen Alpha
Millennials
coming to fix the
computer

Directories and files

A **directory** is a location on a computer

- Directories are containers for computer **files**
- Often referred to as “folders” (because of folder icons)
- A **working directory** contains files you’re currently using—it’s where you are currently working
- Sometimes tricky to know what your working directory is

Directories can be nested

- Often refer to the “top” directory as the **root directory**
- Directories inside of other directories are called **subdirectories** (or **subfolders**)
- A **parent directory** is one level up from a file or directory

Directories and files

Every directory on a computer as an “address” called a **(file) path**

- macOS example: `/Users/r.hubert/Documents/`
- Windows example: `C:\Users\HUBERTR\Documents\`

Note: on many machines, `~` is a shortcut for the “home” directory

- A home directory is a user’s root directory
- For me: `~/Documents/` ⇔ `/Users/r.hubert/Documents/`

File paths can be **absolute** or **relative** to working directory

- `/Users/r.hubert/Documents/` is an absolute file path
- `Documents/` is an relative file path if my current working directory is `/Users/r.hubert/`

Directories and files

Files also have file paths like `/Users/r.hubert/Example.txt`

- Files always end with a **file extension** like `.txt` or `.R`, although some operating systems hide them by default
- In file paths, files always end with a file extension, directories will end with a slash or not: `~/Documents` ⇔ `~/Documents/`
- Some files are marked as **hidden files**, but data scientists usually need (and want) to see them
- Get in habit of showing extensions (macOS/Windows) and showing hidden files (Windows)
  - Show hidden files in macOS Finder: `Cmd+Shift+.`
- Copy absolute path of a file/folder with `Option+Right-click` (macOS) or `Shift+Right-click` (Windows)

Directories and files

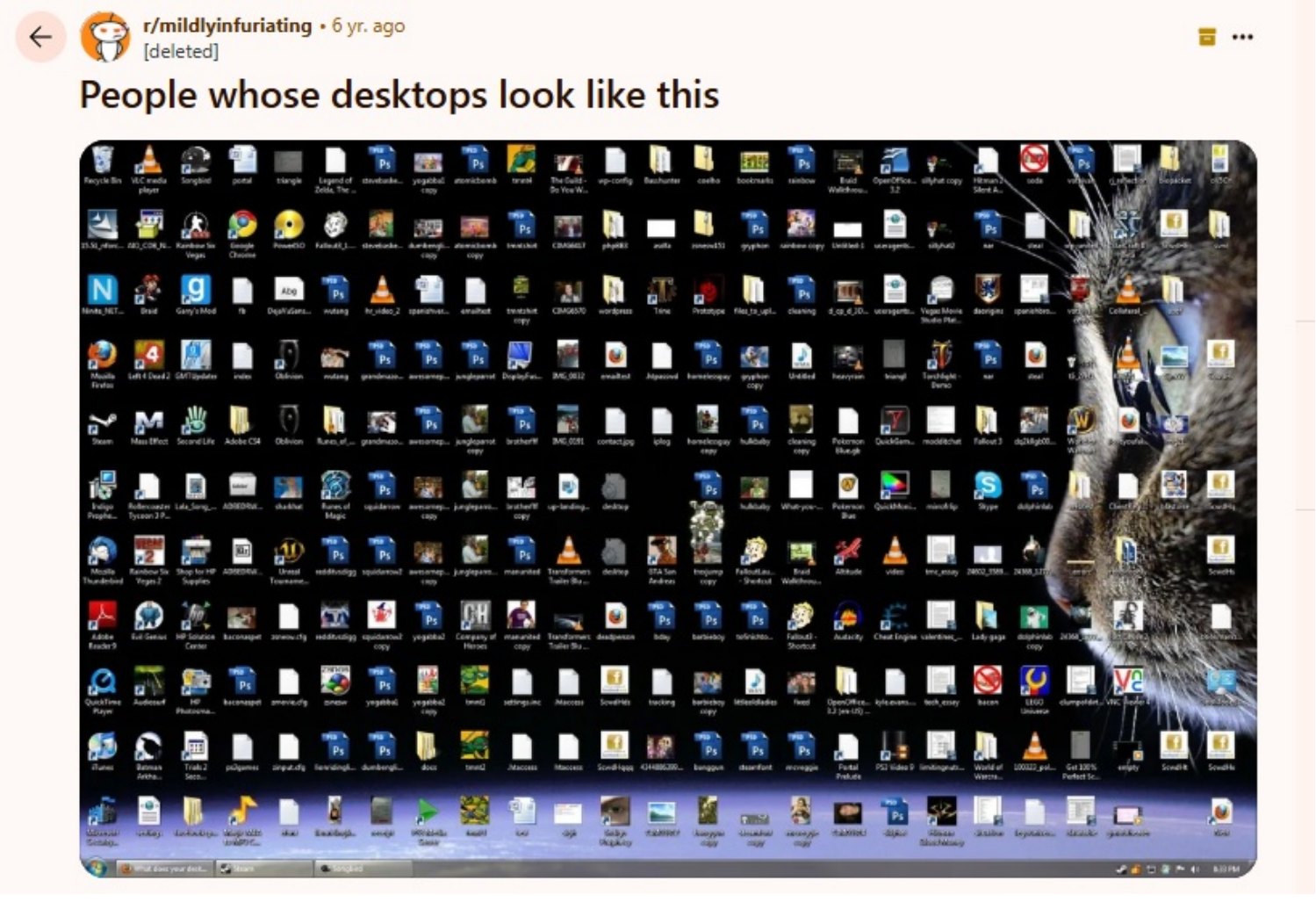
There are many naming conventions you should learn/use

- Only use alphanumeric characters, underscores and hyphens
- Do not use spaces or other symbols
- Especially important: do not use slashes or periods!
- Use concise but descriptive names for folders and files:
  - Folders: `data`, `results`, `figures`, `code`
  - Files: `data_raw.csv`, `clean_data.R`, `data_clean.csv`

Get in the habit of building organised directory structures that others can understand and that will stand the test of time

- We'll evaluate you on this in MY472—use *good judgement*

Don’t let this be you



The command line

There are two primary ways users “talk to” their computers

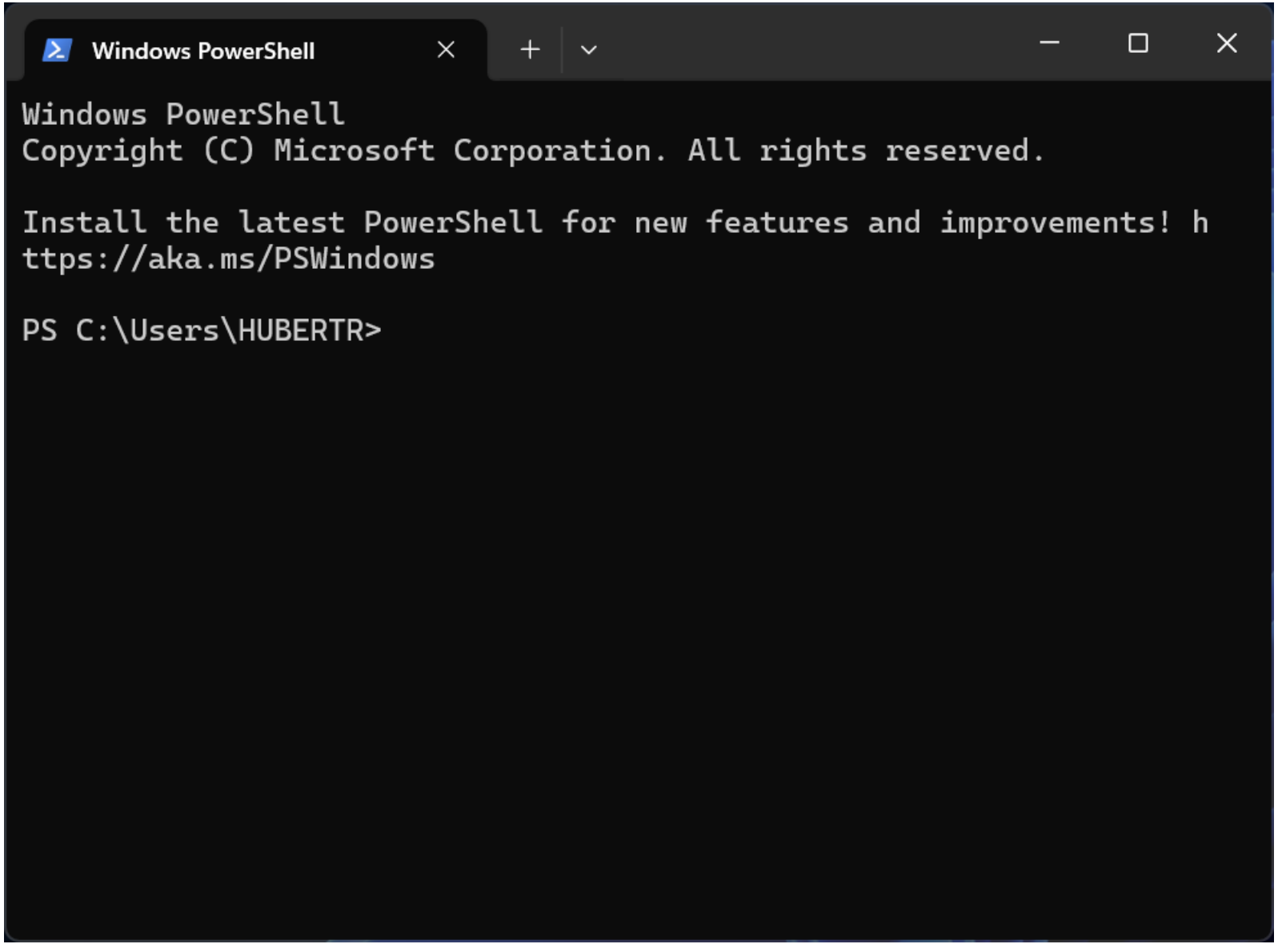
- **Graphical user interface (GUI)**: “point-and-click” interface—this is how most users interact with computers
- **Command line interface (CLI)**: text-based interface where a user types instructions to computer via a **command line**
  - Mac/Linux: use command line in the **Terminal** app
  - Windows: use command line in **PowerShell**

Can do basic computer functions in both, such as moving files around, editing text files, etc.

Many apps have a GUI *and* a CLI, but some only have one

The command line

The command line



The command line

The command line is always in a particular directory

- When you open your command line, it usually starts in your home directory

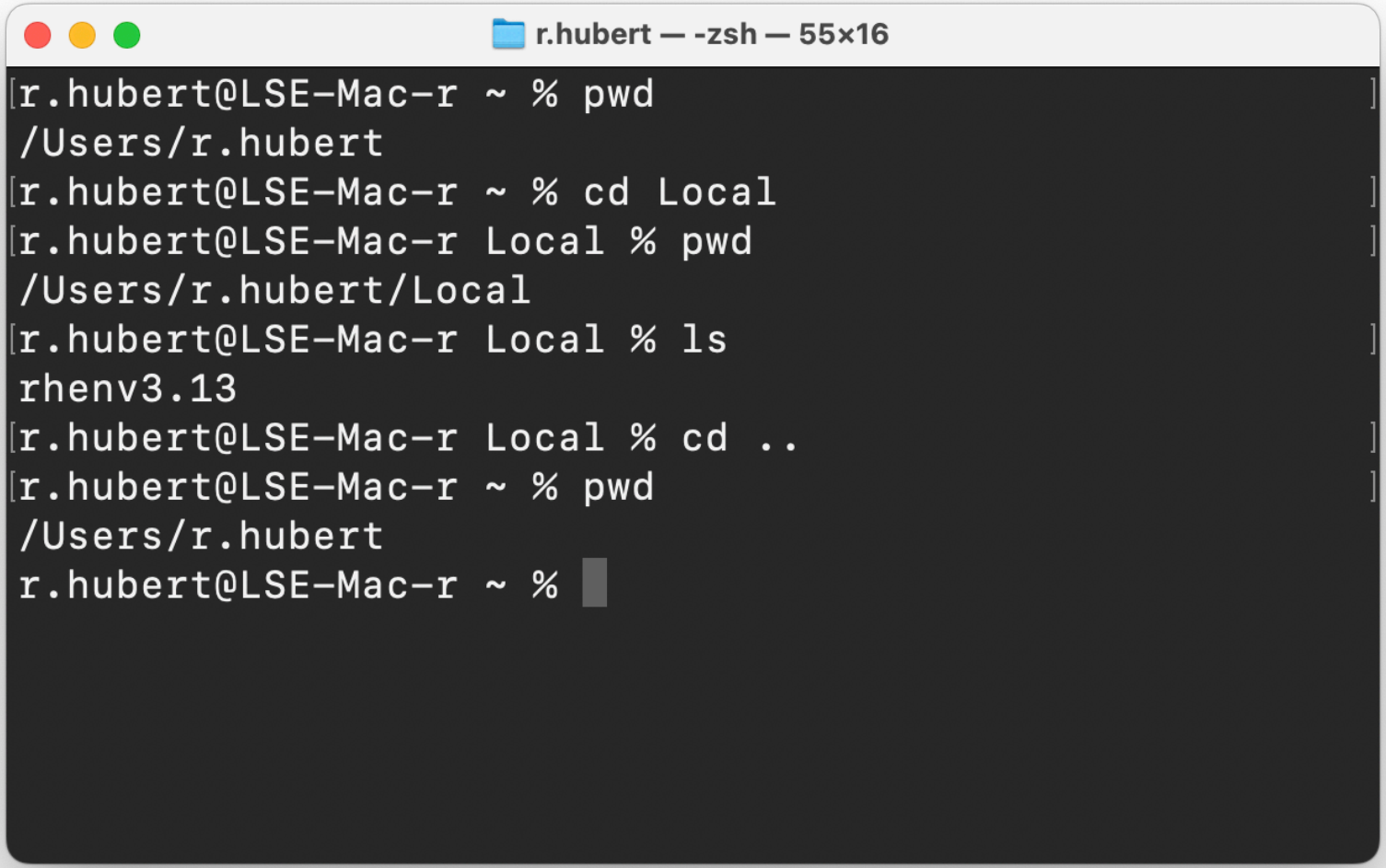
You'll often need to navigate to other directories to do stuff

Some useful commands (Linux/Unix/macOS/PowerShell):

- `pwd`: Print working directory
- `ls`: list folders and files in the working directory
- `cd`: Change directory using absolute/relative filepaths
  - `cd ..` goes up one folder level

Some other related commands include `mkdir`, `rmdir`, `rm`, `touch`

The command line



Operating system types

We won't go deep into operating systems, but it helps to know there are two main kinds:

- **Unix-like** operating systems (Linux, macOS)
  - Built around the command line and the idea of combining small tools to do complex tasks
  - Programmers often prefer them because most modern programming tools are designed to "just work" on Unix
- **Windows** (non-Unix)
  - Built around a graphical interface first, heavy GUI vibes
  - Can run modern programming tools, but very often needs extra setup or compatibility tools

Operating system types

At some risk of getting cancelled by OS warriors…

Windows sometimes breaks in data science applications because of coordination on Unix and limits of Windows GUI emphasis

- Most problems occur during installation and start up

Reality in MY472 as well:

- Instructors are all macOS (Unix) users and the materials are developed on macOS computers
- In past, some Windows users have struggled with some topics
- The in-person practical test will be on macOS
- Blunt truth: you must be comfortable with Unix-like systems

Outline

Programming languages: overview

A **computer program** is a sequence of instructions that a computer is asked to perform

Computer programs are written in **programming languages** that control how the computer interprets instructions

Programming languages can be *low-level* (e.g., C++) or *high-level* (e.g., R or Python)

Data scientists typically work with high-level languages that are built for **object oriented programming**

- OOP organises code around **objects** that are defined in code

R

High level OOP language targeted at *statistical* computing

Why use R?

- Free and open-source (available at r-project.org)
- Quite accessible even to novice coders
- Frequently used in academia and the private sector
- Flexible and extensible through many **packages** or **libraries**
- Excellent online documentation and troubleshooting resources
- A fully-fledged programming language, making it easier to transition to/from other languages

It is well suited to one-off analyses, but unfortunately it is often too unstable and too slow for production

What about Python?

Python is a “similar” high-level OOP language

- Also free and open-source
- Also a robust set of **modules**
- Quite popular in industry, gaining traction in academia

It has some advantages and disadvantages relative to R

- Better for production
- But a bit clunkier for statistical analysis and plotting

As a data scientist, you’ll probably need to be bilingual

- But if you learn one, you can learn the other quickly

SQL and database languages

Data stored in databases is often accessed, queried and manipulated using **Structured Query Language (SQL)**

SQL comes in many flavours

Reasonably easy to learn, but very powerful

You (might) see some SQL later in the term, but not much

You will also learn how to use the `{dplyr}` package in R

- The functions in this package often resemble SQL commands (by design)

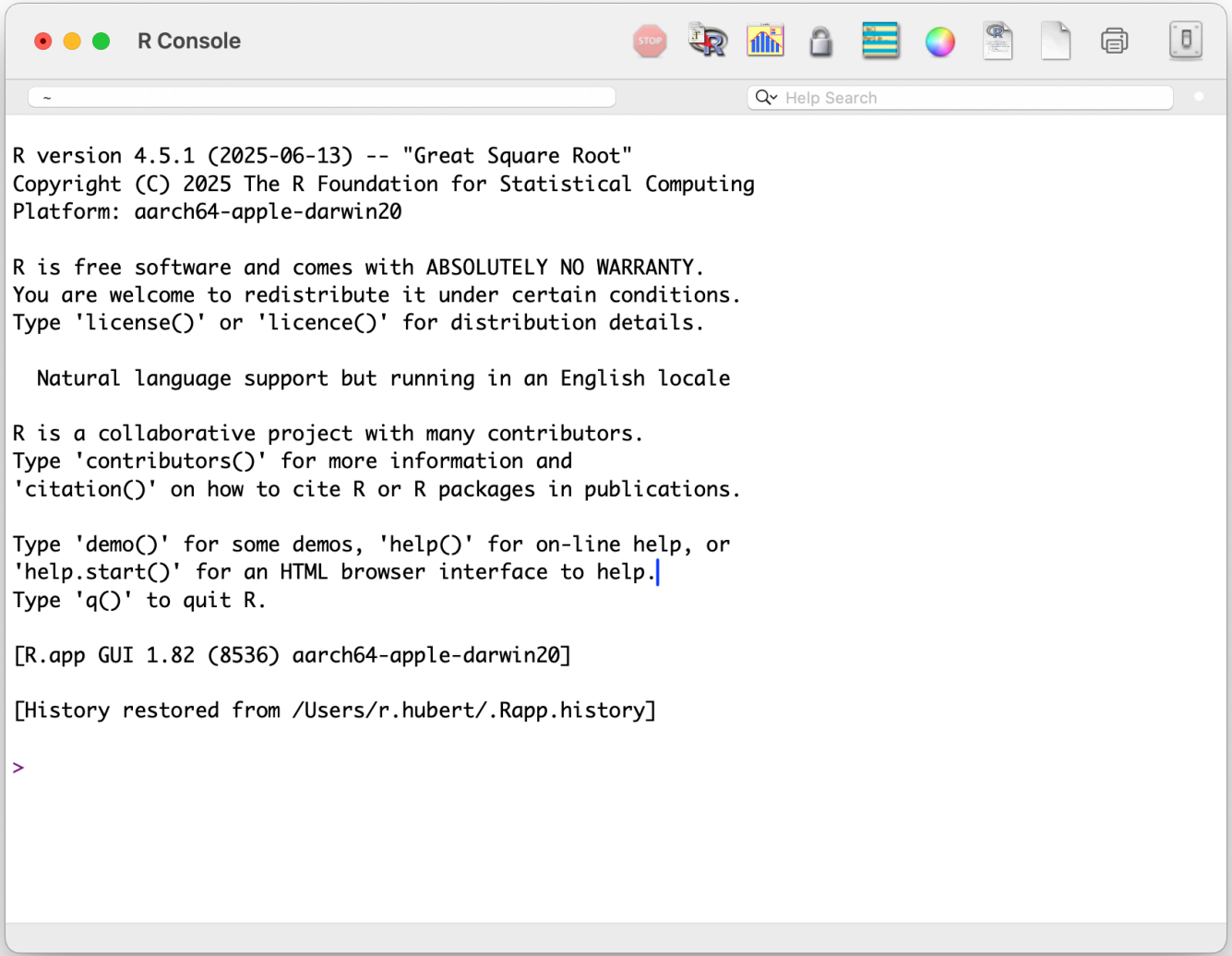
Coding

You can write lines of R code directly into a **console** and get an immediate response

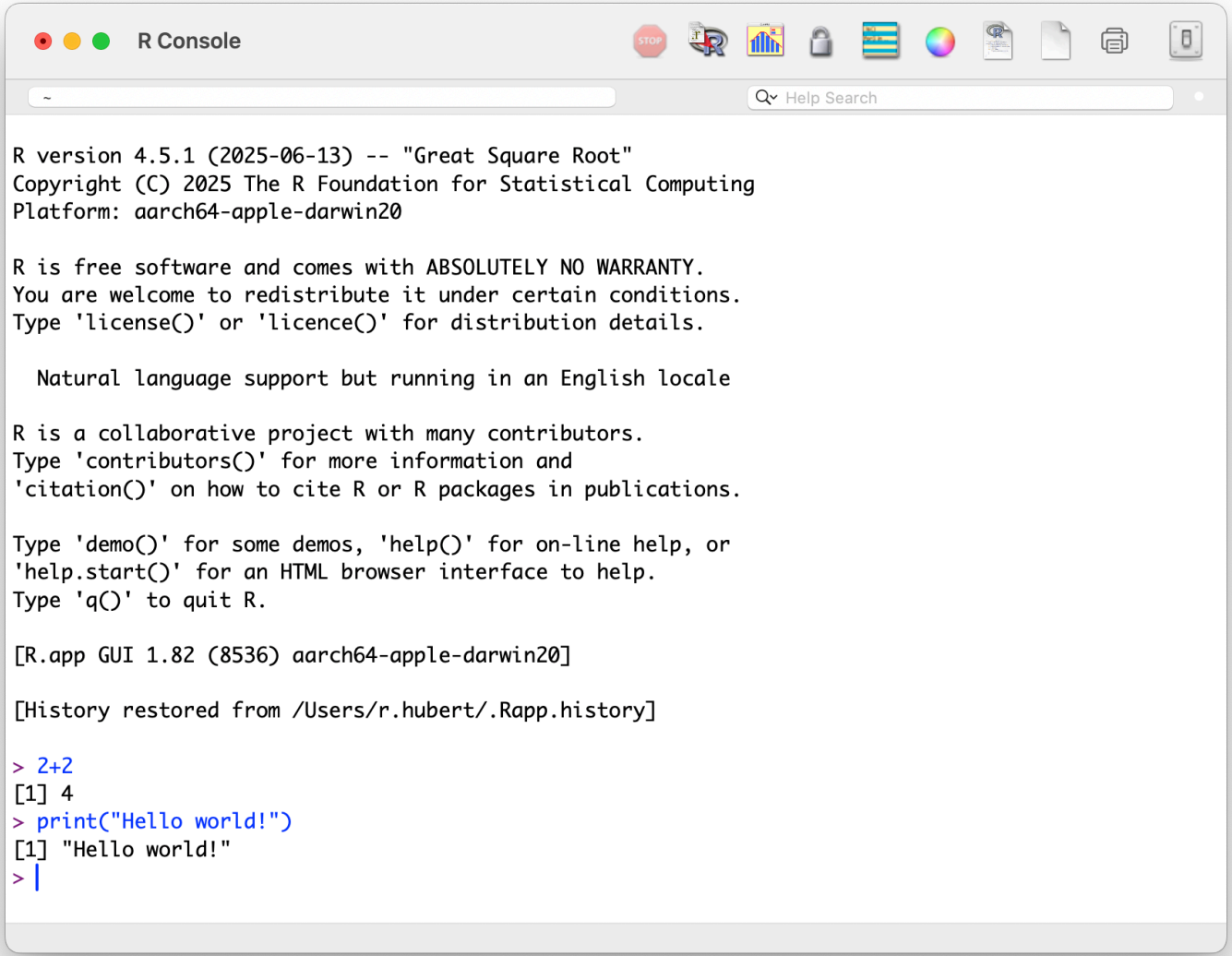
- When you ask the console to process your code, you are **running** or **executing** your code
- The response is printed in the console and it's usually referred to as the **output**

An R console can be launched from inside many applications on a computer, including the official R app and the Terminal (macOS)

Coding


R Console
~
Help Search
R version 4.5.1 (2025-06-13) -- "Great Square Root"
Copyright (C) 2025 The R Foundation for Statistical Computing
Platform: aarch64-apple-darwin20
R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.
Natural language support but running in an English locale
R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.
Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.
[R.app GUI 1.82 (8536) aarch64-apple-darwin20]
[History restored from /Users/r.hubert/.Rapp.history]
>

Coding


R Console
~
Help Search
R version 4.5.1 (2025-06-13) -- "Great Square Root"
Copyright (C) 2025 The R Foundation for Statistical Computing
Platform: aarch64-apple-darwin20
R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.
Natural language support but running in an English locale
R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.
Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.
[R.app GUI 1.82 (8536) aarch64-apple-darwin20]
[History restored from /Users/r.hubert/.Rapp.history]
> 2+2
[1] 4
> print("Hello world!")
[1] "Hello world!"
>

Coding

You can also write lines of code into a plain text file, which you can execute whenever you choose

- You can run lines of code as you are drafting
- You can include **comments**, which are lines in a script that will not be interpreted as instructions by the console
- In R, comments are preceded by a hash: #
- Can “comment out” code that is broken or you no longer want to run (but that you want to keep for posterity)

You can write code in a text editor (e.g. Notepad or TextEdit).

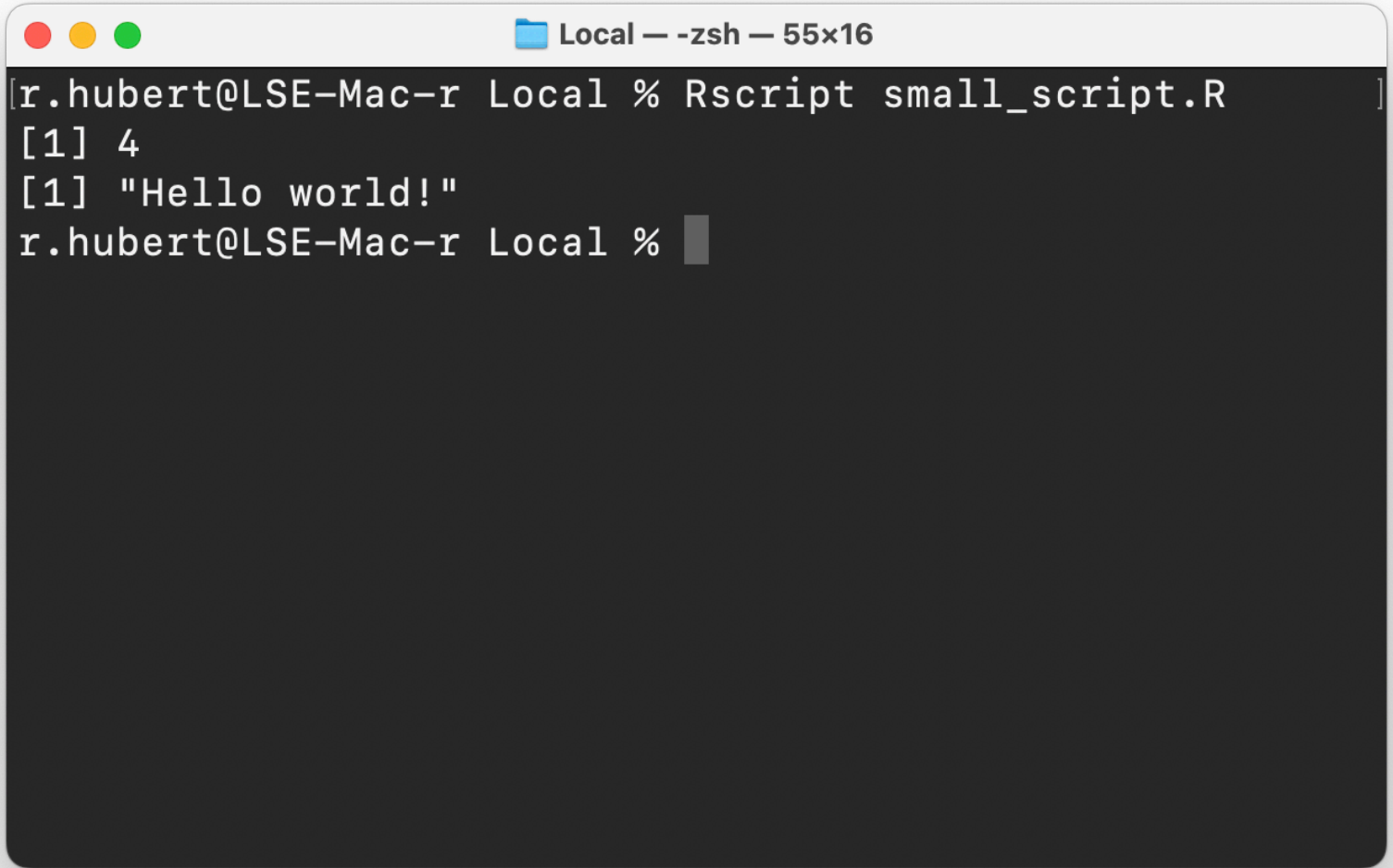
- But then you need a separate tool to run the code
- For example, you can use the command line

Coding

small_script.R

```
2+2
print("Hello world!")
# print("This won't print!")
```

Coding


Local — -zsh — 55×16
r.hubert@LSE-Mac-r Local % Rscript small_script.R
[1] 4
[1] "Hello world!"
r.hubert@LSE-Mac-r Local %

Integrated development environments (IDEs)

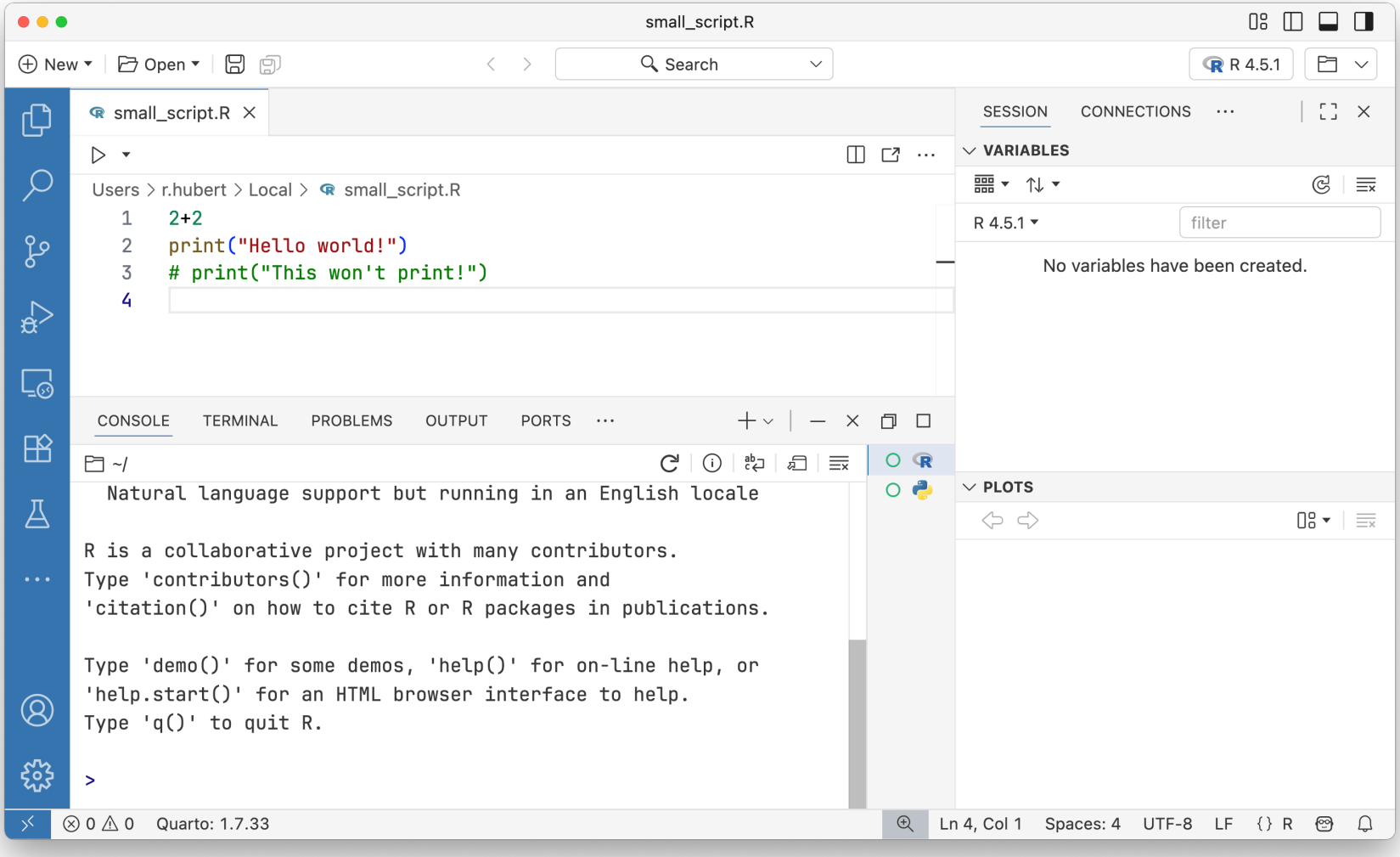
Now more common to use an **integrated development environment (IDE)** instead of text file + command line/R app

- Provides a suite of tools for developing and running code
- Some examples: RStudio, PyCharm, VS Code, Positron

In this class, we will teach you using Positron

- Feel free to use RStudio or VS Code if you prefer

Positron


small_script.R
New
Open
Search
R 4.5.1
small_script.R
Users > r.hubert > Local > small_script.R
1 2+2
2 print("Hello world!")
3 # print("This won't print!")
4
SESSION
CONNECTIONS
VARIABLES
R 4.5.1
filter
No variables have been created.
CONSOLE
TERMINAL
PROBLEMS
OUTPUT
PORTS
~/
Natural language support but running in an English locale
R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.
Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.
>
PLOTS
0 0
Quarto: 1.7.33
Ln 4, Col 1
Spaces: 4
UTF-8
LF
{} R

Scripts versus notebooks

With R (and some other languages) there are actually two kinds of plain text files you can write code into:

- A **script** contains only lines of code and comments (again, preceded by a `#`)
  - With the R language, scripts are saved in `.R` files
- A **notebook** contains both written text plus code, where the code is contained within demarcated **chunks**
  - The written text can be formatted using **markdown**
  - You can still run lines of code from within chunks
  - You can also **compile** (or **knit** or **render**) the notebook to a nicely formatted `.html` or `.pdf` file
  - The compiled document will contain formatted text, chunks of code, and the code output

R notebooks

With R, the two most common kind of notebooks are R Markdown and Quarto

- R Markdown notebooks are saved as `.rmd` files
- Quarto notebooks are saved as `.qmd` files

In this course, we will use Quarto `.qmd` files

- We will write our code (and text) in `.qmd` files and then compile the end product to nicely formatted `.html` files
- For assessments, you will need to submit `.qmd` and `.html` files
- If you are familiar with R Markdown, Quarto is very similar

Outline

Directory management in R

Just like the command line, R always starts in a directory

- So, when you run R code, you need to think about your working directory—but why?

```
read.csv("fake_dataset.csv")
```

```
Error in file(file, "rt"): cannot open the connection
```

File `fake_dataset.csv` isn't in R's (current) working directory!

- Can see this with `list.files()`

```
list.files(getwd())
```

```
[1] "fake_doc.pdf"  "fake_text.txt"
```

Sidebar on error handling

Previous error is ***by far*** most common error in MY472

Huge part of data science (& coding): dealing with errors in code

Computers are very literal—they cannot interpret subtext

- Your *intended* instructions might not be what computer understands your instructions to be
- This is why vibecoding is dangerous

More than 95% of errors in MY472 are human errors

A key skill: learning how to read and interpret error messages

- **I want to emphasise: please read error messages!**
- Learn to love Stack Overflow, help docs & AI assistants

Directory management in R

How to deal with this?

- Specify the *full* path: `read.csv("~/FakeDir/fake_dataset.csv")`
- Try to change the working directory: `setwd("~/FakeDir")`

This is fine if you never expect anything to move

Our approach in MY472: include a line where user inserts their working directory path, then use it in later lines

```
wdir <- "" # <- paste your path here
```

Then when opening a file:

```
read.csv(file.path(wdir, "fake_dataset.csv"))
```

Directory management in R

You can also check if directories exist, and make them if not

```
new.folder <- "~/TestFolder"
file.exists(new.folder)
```

```
[1] FALSE
```

```
if(!file.exists(new.folder)){
    dir.create(new.folder)
}
```

Helpful tip: make an object with the folder name to avoid copy/paste errors in subsequent lines

Local versus remote

So far, we just saw examples of **local** directories

- "Local" just means *on your computer*

Files can also be stored in **remote** directories

- "Remote" means *on another computer* (a **server**)

To use a file from a remote directory on your own computer, you have to **download** it first

```
remote.file <- "https://lse-my472.github.io/robots.txt"
local.file <- file.path(wdir, "robots.txt")
download.file(remote.file, local.file)
raw.text <- readLines(local.file)
print(raw.text)
```

```
[1] "Sitemap: https://lse-my472.github.io/sitemap.xml"
```

Outline

Version control

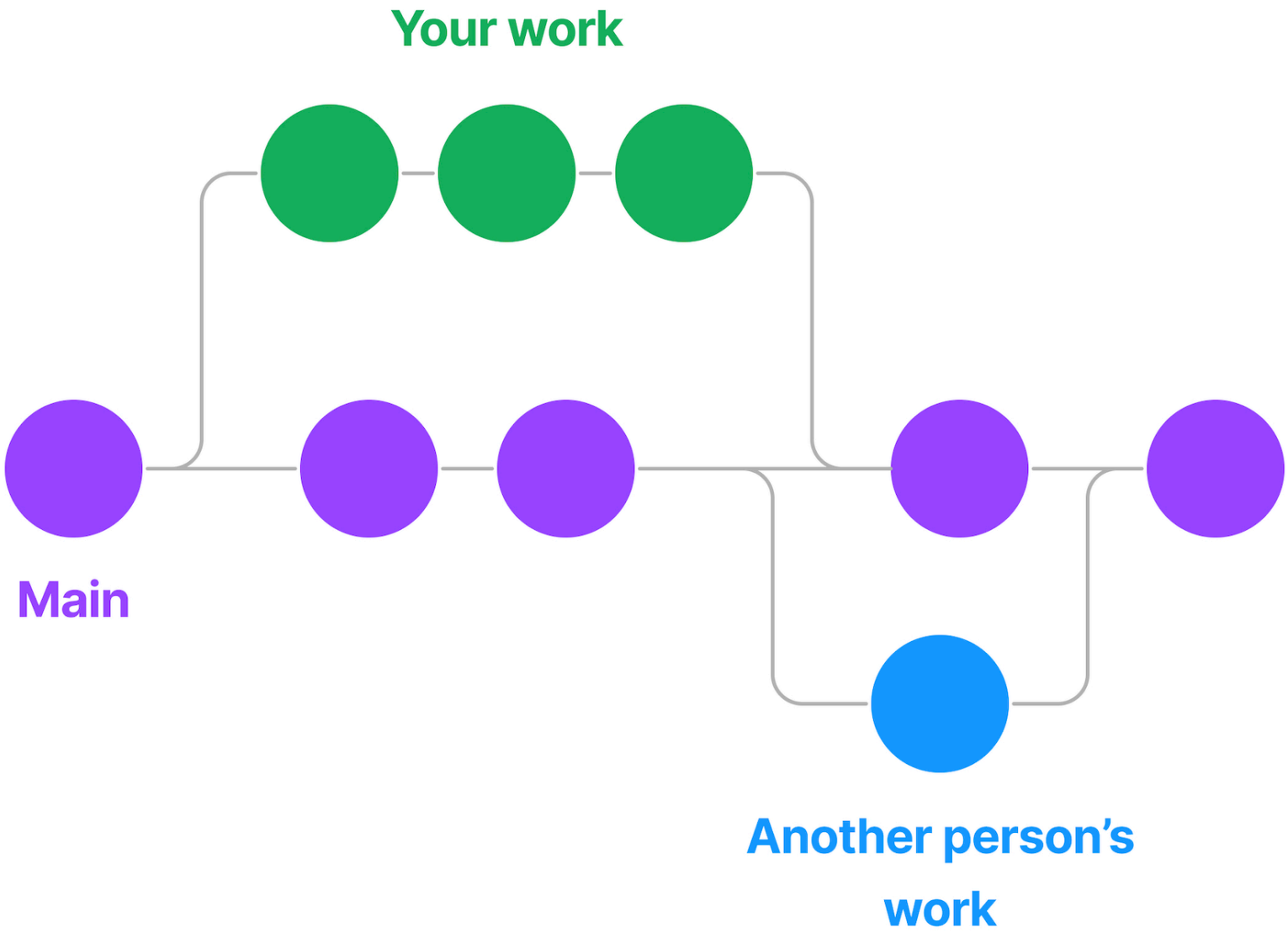
A **version control system (VCS)** is key when working on code, particularly when collaborating

It keeps records of changes in files: *who* made *which* changes *when*

Allows for possibility of reverting changes and going back to previous states

When a VCS keeps the entire code and history on each collaborator's computer, it is called **distributed**

The main idea(s)



Git and GitHub

Git is a very popular distributed version control system

- It's software you install on your computer, which tracks files

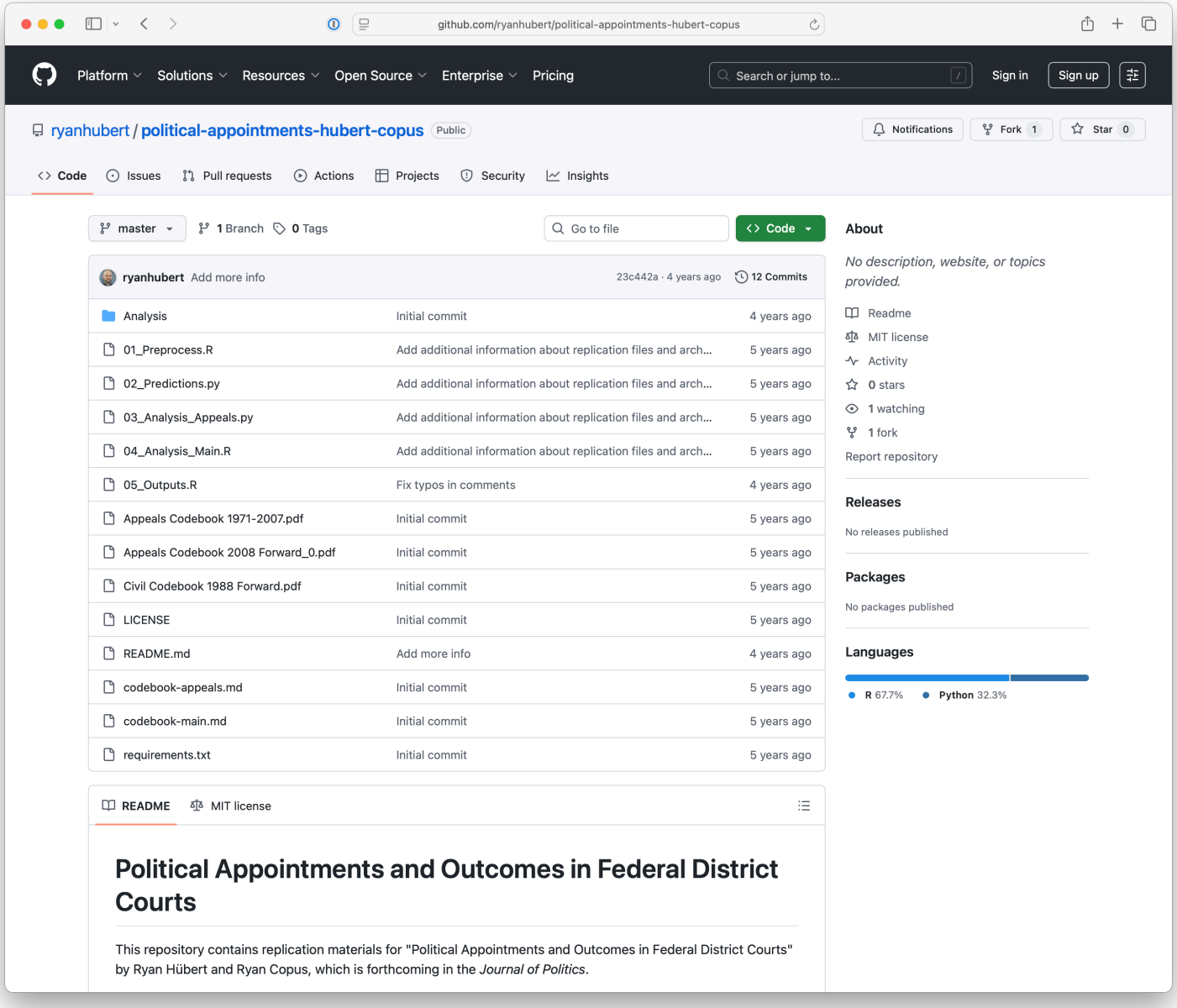
A **repository** (or **repo**) is a collection of files (code, data, etc.) that are tracked by Git

- A repo is stored in a directory (recall: folder) on your computer with some special (hidden) files in them
- You can instruct Git to ignore and not track some files

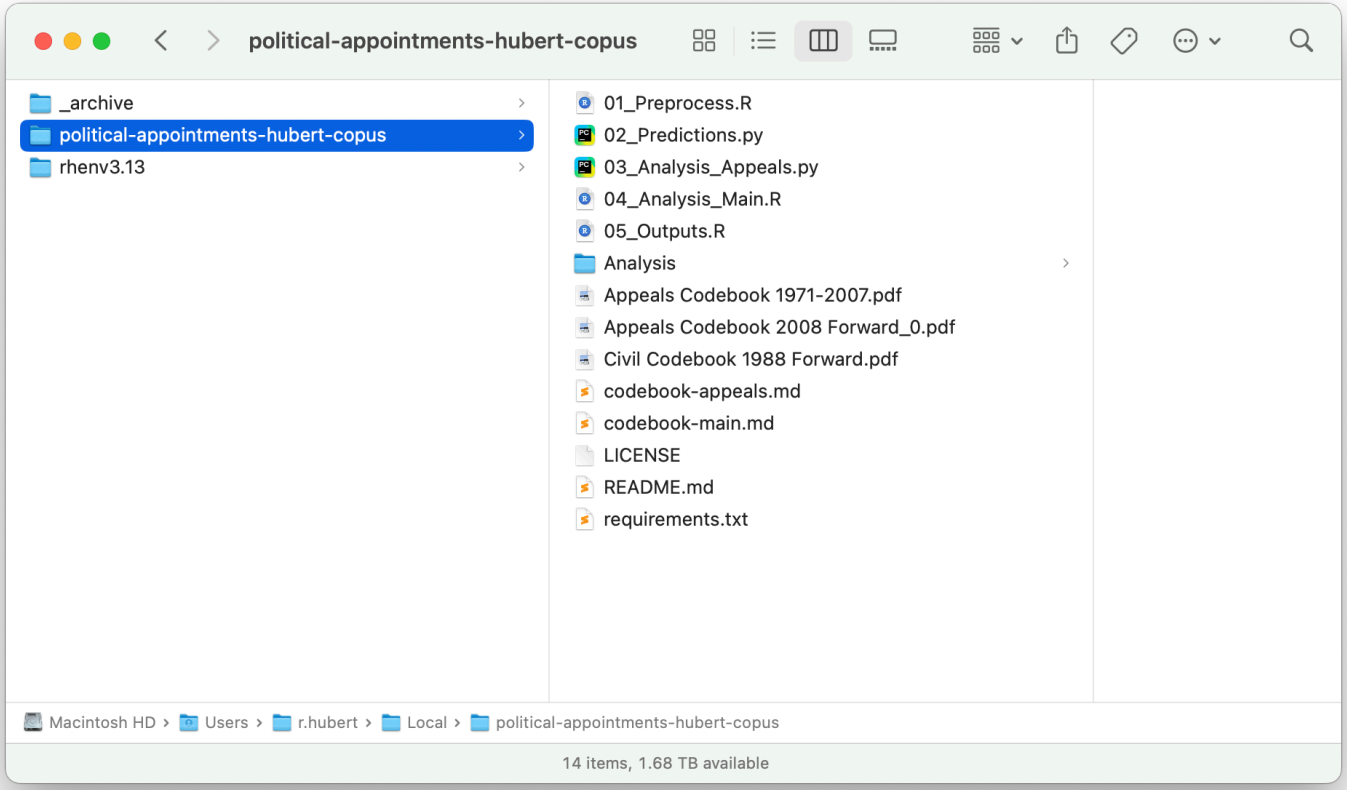
GitHub is a web-based service that hosts collections of code online (with many extra functionalities)

- GitHub is a cloud backup/distribution system for local repos

Git repositories



Git repositories



Git workflow basics

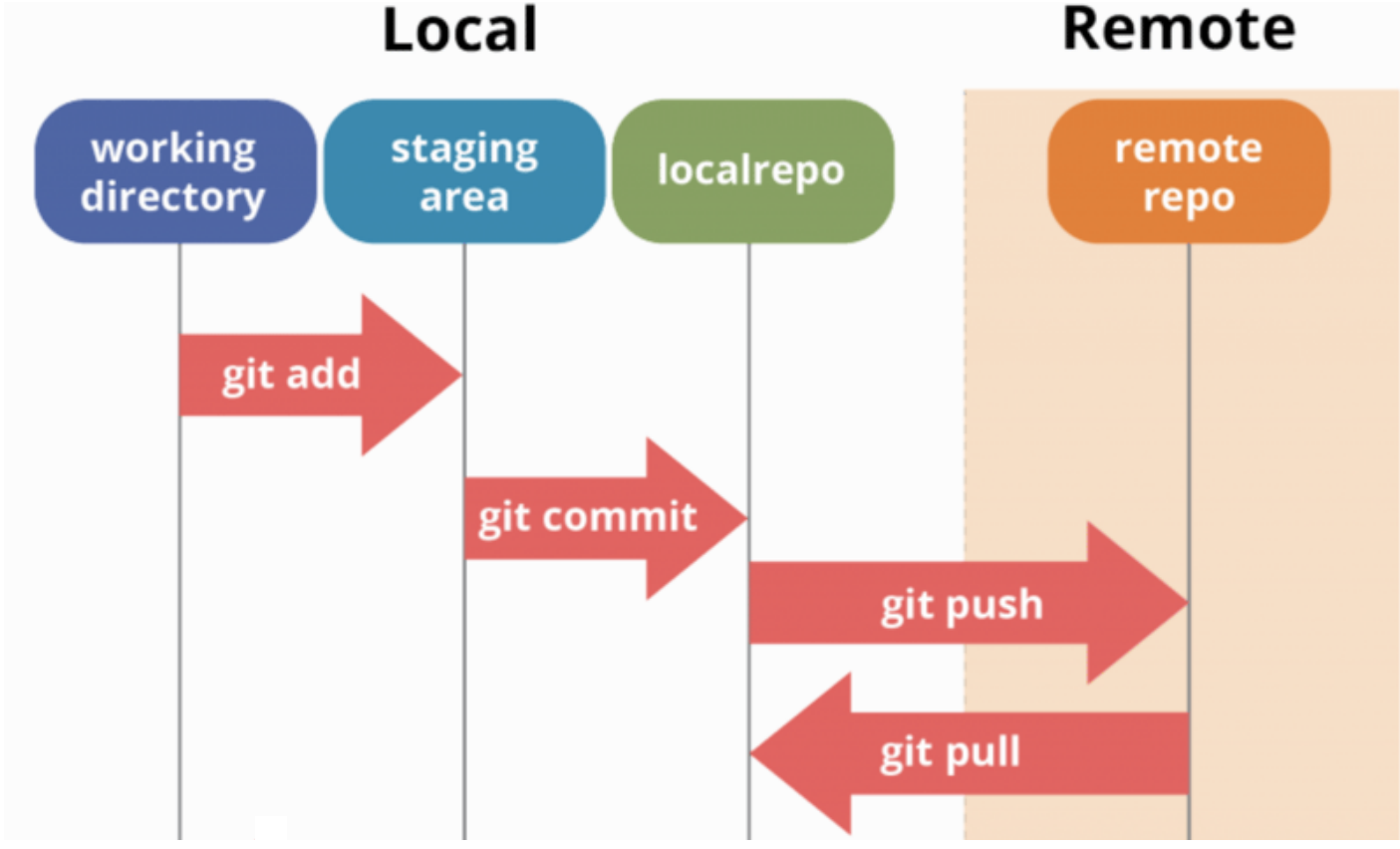
Most common way to work with Git on your computer ("locally") is via the command line

- You *can* use the GitHub app or Git/GitHub integration in your IDE, but we will teach you on command line

These are the main commands we will use:

1. `git clone`: save a repo from a remote server to a local folder
2. `git pull`: pull files from remote to local
3. `git status`: check the status of the local repo
4. `git add`: stage files, preparing them for commit
5. `git commit`: commit changes to the local repo (always need to add a message)
6. `git push`: push changes from local to remote

Git workflow basics



Adapted from: Molly Nemerever, Git, GitHub, & Workflow Fundamentals

Advanced usage of Git

In most real-world settings, you will need to know:

- **Fork**: make your own version of a repository
  - Note: you're *forking* off, not *cloning*
- **Branch**: a parallel version of a repo originating at a particular point
- **Merge**: combining branches together to maintain single "codebase"
- **Pull request**: GitHub based request to merge a branch or a fork into other code

You won't use these in this course.

- For practice: `https://learngitbranching.js.org/`

Homework before seminar

1. Install R and Positron
2. Ensure that your GitHub account has been set up
3. Ensure that Git is installed on your computer

Attend GTA drop-in tech help session (if needed)