

Advanced Public Economics — Applied Tutorial

Session I: Course Overview / Introduction to Tools for Empirical Research

Daniel Weishaar

Welcome!

Welcome to the Course!

About me:

- ▶ Since July 2025: Postdoctoral researcher in Cologne.
- ▶ PhD in Economics from LMU Munich, research stays in US (Columbia, Berkeley).
- ▶ **Research fields:** Public economics, political economy, normative economics.
- ▶ **Research topics:** Inequality Measurement, (Optimal) Taxation, Redistributive Preferences, Tax Evasion. → see [website](#) if you want to know more.

Welcome to the Course!

About me:

- ▶ Since July 2025: Postdoctoral researcher in Cologne.
- ▶ PhD in Economics from LMU Munich, research stays in US (Columbia, Berkeley).
- ▶ **Research fields:** Public economics, political economy, normative economics.
- ▶ **Research topics:** Inequality Measurement, (Optimal) Taxation, Redistributive Preferences, Tax Evasion. → see [website](#) if you want to know more.

About you?

- ① Your name and background (study program, semester).
- ② Your expectations about the course (if any).
- ③ Experience in coding? Which software/tools (if any)?
- ④ One topic in economics you find particularly interesting.

Course Overview

Course Overview

Core Idea: Provide some empirical complement to the theory-focused lecture.

Course Overview

Core Idea: Provide some empirical complement to the theory-focused lecture.

Objectives:

- ▶ Show connection between theory and empirics using **hands-on exercises**.
- ▶ Not a classic programming course! Pragmatic “learning by doing” .
- ▶ **Reveal challenges** when moving from clean theory to limited nature of data.
- ▶ Show how connection between theory and empirics can **inform policy makers**.
- ▶ Stimulate thoughts about your own research ideas.

Course Overview

Core Idea: Provide some empirical complement to the theory-focused lecture.

Objectives:

- ▶ Show connection between theory and empirics using **hands-on exercises**.
- ▶ Not a classic programming course! Pragmatic “learning by doing”.
- ▶ **Reveal challenges** when moving from clean theory to limited nature of data.
- ▶ Show how connection between theory and empirics can **inform policy makers**.
- ▶ Stimulate thoughts about your own research ideas.

Examples:

- ① What is the optimal level of top tax rates in the US?
- ② Is the German tax and transfer system Pareto-efficient?
- ③ Under which welfare weights are tax reforms welfare improving?

Logistics and Organization

Meeting time: Tuesdays, 17:45–19:15

Date range: 25 November 2025 – 03 February 2026

Location: 106 Seminarraum S12 (106/01/1.06)

Please bring your Laptop to all sessions (if possible)!

Material: Available via [GitHub repository](#), see also overview document there.

Assessment: Some exercises in problem set may refer to the course material.

Contact: d.weishaar@wiso.uni-koeln.de

Schedule (tentative)

Date	Topic
25.11.2025	Course Overview / Introduction to Tools for Empirical Research
02.12.2025	Optimal Tax Rates, Estimating Optimal Top Tax Rates
09.12.2025	Optimal Tax Rates, Estimating Shape of Optimal Tax Rates
16.12.2025	Optimal Tax Rates, Inverse Optimum Approach
<hr/> <i>Christmas break</i> <hr/>	
13.01.2026	Tax Reforms, Microsimulation Models I
20.01.2026	Tax Reforms, Microsimulation Models II
27.01.2026	Tax Reforms, Estimating Revenue and Welfare Effects
03.02.2026	Estimating Behavioral Responses to Taxation

Schedule (tentative)

Date	Topic
25.11.2025	Course Overview / Introduction to Tools for Empirical Research
02.12.2025	Optimal Tax Rates, Estimating Optimal Top Tax Rates
09.12.2025	Optimal Tax Rates, Estimating Shape of Optimal Tax Rates
16.12.2025	Optimal Tax Rates, Inverse Optimum Approach
<hr/> <i>Christmas break</i> <hr/>	
13.01.2026	Tax Reforms, Microsimulation Models I
20.01.2026	Tax Reforms, Microsimulation Models II
27.01.2026	Tax Reforms, Estimating Revenue and Welfare Effects
03.02.2026	Estimating Behavioral Responses to Taxation

⇒ Any questions?

Introduction to Tools for Empirical Research

Getting started

We will (mainly) use **R** with **RStudio**, and **git** in this course.

Getting started

We will (mainly) use **R** with **RStudio**, and **git** in this course.

① **R** is an open source programming language.

- ▶ In my view, R provides the best mix of *econometrics*, *data handling*, and *reproducible research tools* for this course.
- ▶ But! Everything can also be done in Stata, MATLAB, python, or a combination.

Getting started

We will (mainly) use **R** with **RStudio**, and **git** in this course.

① **R** is an open source programming language.

- ▶ In my view, R provides the best mix of *econometrics*, *data handling*, and *reproducible research tools* for this course.
- ▶ But! Everything can also be done in Stata, MATLAB, python, or a combination.

② **RStudio** is a development environment that handles programming with R, and contains scripts, console, environment, plots, help files, file browser.

Getting started

We will (mainly) use **R** with **RStudio**, and **git** in this course.

- ① **R** is an open source programming language.
 - ▶ In my view, R provides the best mix of *econometrics*, *data handling*, and *reproducible research tools* for this course.
 - ▶ But! Everything can also be done in Stata, MATLAB, python, or a combination.
- ② **RStudio** is a development environment that handles programming with R, and contains scripts, console, environment, plots, help files, file browser.
- ③ **git** is a version control system. It keeps track of changes in your code, it is like a time-machine for your research project.

Getting started

We will (mainly) use **R** with **RStudio**, and **git** in this course.

- ① **R** is an open source programming language.
 - ▶ In my view, R provides the best mix of *econometrics*, *data handling*, and *reproducible research tools* for this course.
 - ▶ But! Everything can also be done in Stata, MATLAB, python, or a combination.
- ② **RStudio** is a development environment that handles programming with R, and contains scripts, console, environment, plots, help files, file browser.
- ③ **git** is a version control system. It keeps track of changes in your code, it is like a time-machine for your research project.

⇒ Installation instructions can be found in overview document in [GitHub repository](#).

Roadmap for today

Today, we focus on the empirical tools that we will use in the course.

- ① Short introduction into git.
- ② Short introduction into RStudio.
- ③ Basic commands in R for data analysis using (hypothetical) data set.

Short introduction into git

Overview

What is git? Version control system that tracks changes in your project over time.

Overview

What is git? Version control system that tracks changes in your project over time.

Why do we use it?

- **Project History:** Every change you make is stored as a “snapshot”.

Overview

What is git? Version control system that tracks changes in your project over time.

Why do we use it?

- **Project History:** Every change you make is stored as a “snapshot”.
 - You can always go back to an earlier version.

Overview

What is git? Version control system that tracks changes in your project over time.

Why do we use it?

- **Project History:** Every change you make is stored as a “snapshot”.
 - You can always go back to an earlier version.
 - In contrast to normal backup software, you specify when to make snapshots, what changes to include, and you add a description of the snapshot.

Overview

What is git? Version control system that tracks changes in your project over time.

Why do we use it?

- ▶ **Project History:** Every change you make is stored as a “snapshot”.
 - You can always go back to an earlier version.
 - In contrast to normal backup software, you specify when to make snapshots, what changes to include, and you add a description of the snapshot.
- ▶ **Collaboration:** Work on one version of the file jointly, but prevent overwriting each other's work through structured highlighting of conflicts.

Overview

What is git? Version control system that tracks changes in your project over time.

Why do we use it?

- ▶ **Project History:** Every change you make is stored as a “snapshot”.
 - You can always go back to an earlier version.
 - In contrast to normal backup software, you specify when to make snapshots, what changes to include, and you add a description of the snapshot.
- ▶ **Collaboration:** Work on one version of the file jointly, but prevent overwriting each other's work through structured highlighting of conflicts.
 - $\text{git} \neq \text{GitHub}$. Git is the software for version control, GitHub/GitLab are online platforms that allow working jointly on projects tracked with git.

Overview

What is git? Version control system that tracks changes in your project over time.

Why do we use it?

- ▶ **Project History:** Every change you make is stored as a “snapshot”.
 - You can always go back to an earlier version.
 - In contrast to normal backup software, you specify when to make snapshots, what changes to include, and you add a description of the snapshot.
- ▶ **Collaboration:** Work on one version of the file jointly, but prevent overwriting each other's work through structured highlighting of conflicts.
 - $\text{git} \neq \text{GitHub}$. Git is the software for version control, GitHub/GitLab are online platforms that allow working jointly on projects tracked with git.
- ▶ **Exploration:** Work on different ideas using branches without breaking anything.

Overview

What is git? Version control system that tracks changes in your project over time.

Why do we use it?

- ▶ **Project History:** Every change you make is stored as a “snapshot”.
 - You can always go back to an earlier version.
 - In contrast to normal backup software, you specify when to make snapshots, what changes to include, and you add a description of the snapshot.
- ▶ **Collaboration:** Work on one version of the file jointly, but prevent overwriting each other's work through structured highlighting of conflicts.
 - $\text{git} \neq \text{GitHub}$. Git is the software for version control, GitHub/GitLab are online platforms that allow working jointly on projects tracked with git.
- ▶ **Exploration:** Work on different ideas using branches without breaking anything.

Overview

What is git? Version control system that tracks changes in your project over time.

Why do we use it?

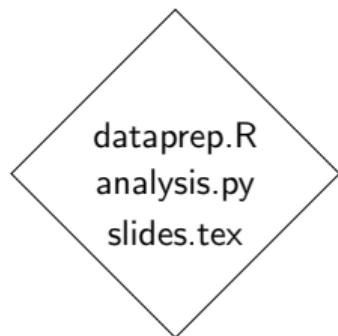
- ▶ **Project History:** Every change you make is stored as a “snapshot”.
 - You can always go back to an earlier version.
 - In contrast to normal backup software, you specify when to make snapshots, what changes to include, and you add a description of the snapshot.
- ▶ **Collaboration:** Work on one version of the file jointly, but prevent overwriting each other's work through structured highlighting of conflicts.
 - $\text{git} \neq \text{GitHub}$. Git is the software for version control, GitHub/GitLab are online platforms that allow working jointly on projects tracked with git.
- ▶ **Exploration:** Work on different ideas using branches without breaking anything.

How do we use it?

- ▶ Can be used independently of all other software via command line.
- ▶ Some software integrate git commands into the user interface (e.g. RStudio).

Key git Concepts and Workflow

Working Directory

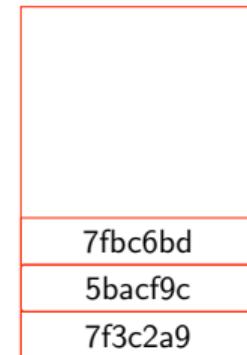


Staging Area



Prepare Snapshot
Stage Changes

Local Repository

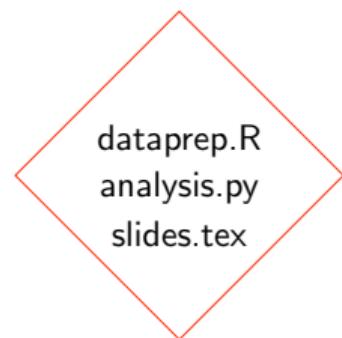


Make Snapshot
Commit Changes

Repository: A project folder that git tracks. Contains snapshots of all code, e.g. data cleaning scripts, tex files. Each snapshot has an identifier.

Key git Concepts and Workflow

Working Directory



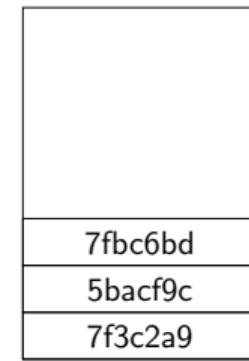
Prepare Snapshot
→
Stage Changes

Staging Area



Make Snapshot
→
Commit Changes

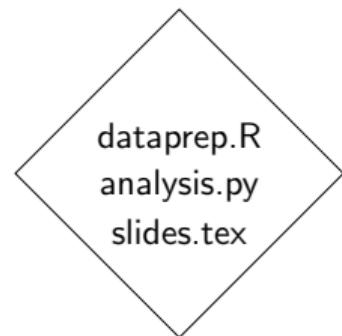
Local Repository



Working Directory: Your current files as you edit them.

Key git Concepts and Workflow

Working Directory

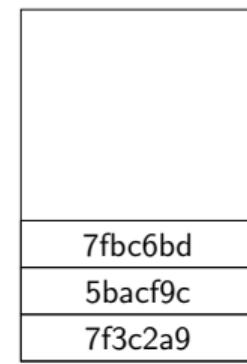


Staging Area



Prepare Snapshot
→
Stage Changes

Local Repository



Make Snapshot
→
Commit Changes

Staging Area: A “waiting room” where you select changes for next commit.

Key git Concepts and Workflow

Working Directory

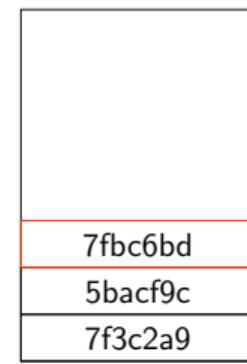


Staging Area



Prepare Snapshot
→
Stage Changes

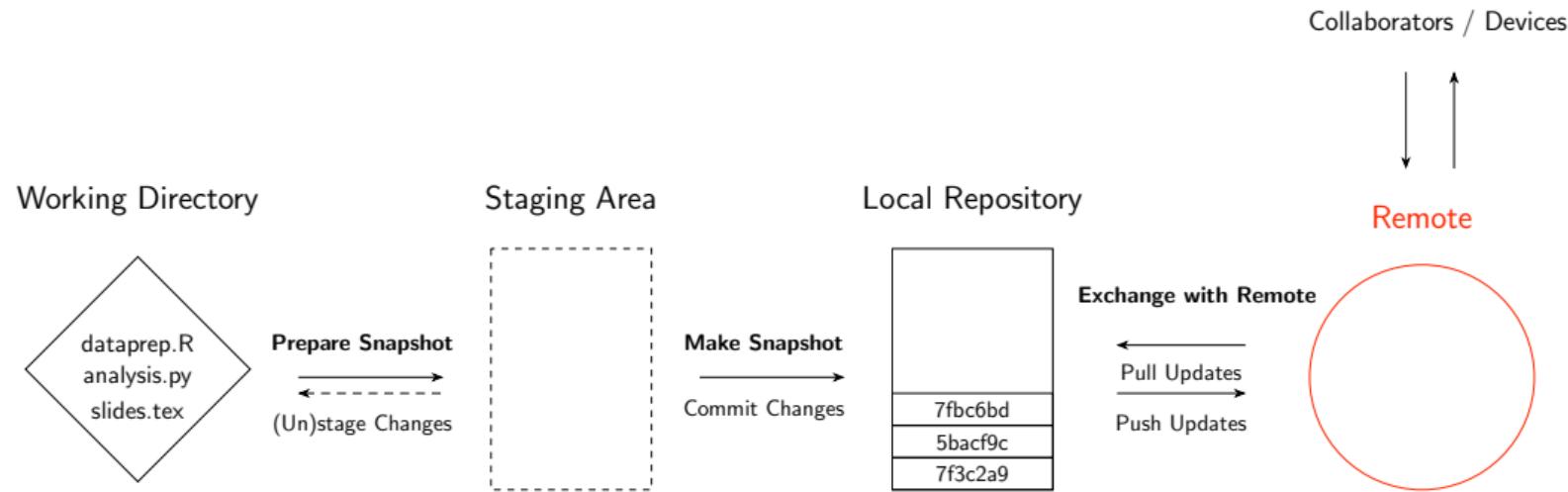
Local Repository



Make Snapshot
→
Commit Changes

Commit: A snapshot of your project, with a message describing what changed.

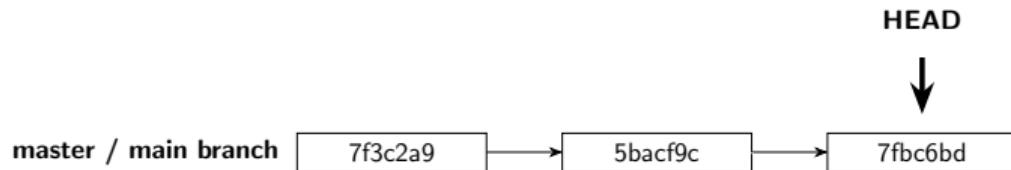
Key git Concepts and Workflow



Remote: An online copy of your repository to back up and collaborate with others.

Examples of online platforms for repositories are [GitHub](#) or [GitLab](#).

git History and Branches



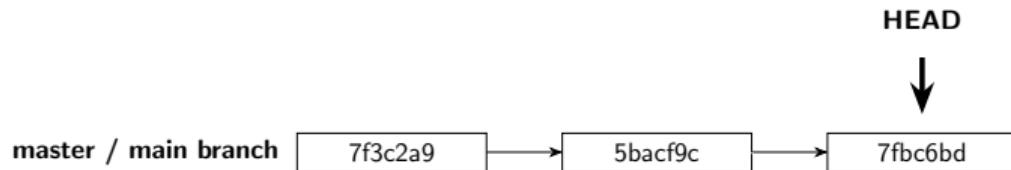
HEAD: Where you are currently looking at in your working directory.

git History and Branches



We can **checkout** different points in history to see a previous snapshot.

git History and Branches



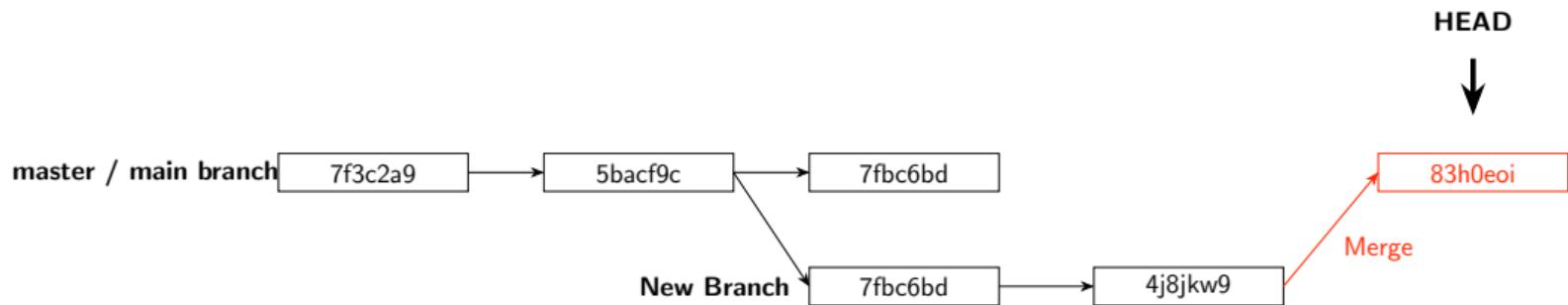
We can **checkout** different points in history to see a previous snapshot.

git History and Branches



Branch: A parallel line of development to test ideas.

git History and Branches



Branch can be **merged** back into the master / main branch to take over changes.

Using git via Command Line (git bash, Terminal,...)

Repository setup

- ▶ `git init` — start a new repository
- ▶ `git clone <URL>` — download an existing GitHub repository

Using git via Command Line (git bash, Terminal,...)

Repository setup

- ▶ `git init` — start a new repository
- ▶ `git clone <URL>` — download an existing GitHub repository

Daily workflow

- ▶ `git status` — see what changed
- ▶ `git add (restore) <file>` — (un)stage changes
- ▶ `git diff (--staged)` — show unstaged (staged) changes
- ▶ `git commit -m "message"` — save a snapshot
- ▶ `git push` — upload commits to GitHub
- ▶ `git pull` — download updates from GitHub
- ▶ `git log (--oneline)` — list history of repository

Using git via Command Line (git bash, Terminal,...)

Repository setup

- ▶ `git init` — start a new repository
- ▶ `git clone <URL>` — download an existing GitHub repository

Daily workflow

- ▶ `git status` — see what changed
- ▶ `git add (restore) <file>` — (un)stage changes
- ▶ `git diff (--staged)` — show unstaged (staged) changes
- ▶ `git commit -m "message"` — save a snapshot
- ▶ `git push` — upload commits to GitHub
- ▶ `git pull` — download updates from GitHub
- ▶ `git log (--oneline)` — list history of repository

Branches

- ▶ `git branch` — list branches
- ▶ `git branch <name>` — create new branch
- ▶ `git checkout <name>` — switch branches / access specific snapshot
- ▶ `git merge <name>` — merge a branch into the current one

Local git Repository: Hands-on Exercise

Additional Considerations (I)

What (not) to track?

- ▶ In principle, all files and documents can be tracked with git.
- ▶ However, git works efficiently only with plain text files (.R, .py, .tex, .txt).
 - ▶ When making a commit, git stores internally only the difference.
 - ▶ This is not possible with binary files (.png, .pdf, .docx, .RData). For those file types, git internally needs to store a full copy. ⇒ Avoid to track data and graphs.

Additional Considerations (I)

What (not) to track?

- ▶ In principle, all files and documents can be tracked with git.
- ▶ However, git works efficiently only with plain text files (.R, .py, .tex, .txt).
 - ▶ When making a commit, git stores internally only the difference.
 - ▶ This is not possible with binary files (.png, .pdf, .docx, .RData). For those file types, git internally needs to store a full copy. ⇒ Avoid to track data and graphs.

How can we tell git what we don't want to track?

- ▶ List folders, files, or file types in a .gitignore file in the repository.
- ▶ Add .gitignore file via command touch .gitignore ⇒ [Hands-on exercise](#).
- ▶ Make it visible with "Command + Shift + ." (Mac) or via "File Explorer - View tab - hidden items" (Windows)

Additional Considerations (II)

Local repositories vs. online repositories

- ▶ So far, the hands-on exercise dealt with a repository that is stored locally.
- ▶ When you collaborate with others, you use repositories that are hosted online.
 - ▶ Can be public or private. You can specify who is allowed to contribute.
 - ▶ Also, when you work on different computers yourself, having the online repository is useful to synchronize between different devices.
- ▶ **Example:** the course material is hosted in a public repository on GitHub. Only I can contribute/upload material (push), but you can clone the repository to your local machine and pull updates. ⇒ [Hands-on exercise](#).

Additional Considerations (II)

Local repositories vs. online repositories

- ▶ So far, the hands-on exercise dealt with a repository that is stored locally.
- ▶ When you collaborate with others, you use repositories that are hosted online.
 - ▶ Can be public or private. You can specify who is allowed to contribute.
 - ▶ Also, when you work on different computers yourself, having the online repository is useful to synchronize between different devices.
- ▶ **Example:** the course material is hosted in a public repository on GitHub. Only I can contribute/upload material (push), but you can clone the repository to your local machine and pull updates. ⇒ [Hands-on exercise](#).

Using git outside of command line

- ▶ Working with git in the command line has the benefit that we understand very clearly the different steps involved in git. But very tedious!
- ▶ There are general [git GUI](#) tools for all operating systems.
- ▶ Later, we will see that git is also integrated into the user interface of RStudio.

Short introduction to RStudio

What is RStudio?

RStudio is an Integrated Development Environment (IDE) for R.

What it provides:

- ▶ Script editor + console in one interface.
- ▶ File browser, environment viewer, help viewer.
- ▶ Data viewer (spreadsheet-like).
- ▶ Plot pane for graphics.
- ▶ Integrated support for Git version control.

Objective: Make coding, debugging, and project management smoother.

The RStudio Interface and Projects

Interface:

- ▶ **Top-left:** Script editor (write code).
- ▶ **Bottom-left:** Console (run code interactively).
- ▶ **Top-right:** Environment (see characteristics of objects), history, git.
- ▶ **Bottom-right:** Files, plots, help, packages.

The RStudio Interface and Projects

Interface:

- ▶ **Top-left:** Script editor (write code).
- ▶ **Bottom-left:** Console (run code interactively).
- ▶ **Top-right:** Environment (see characteristics of objects), history, git.
- ▶ **Bottom-right:** Files, plots, help, packages.

RStudio Projects:

- ▶ Each project has its own working directory.
- ▶ Relative paths work automatically.
- ▶ Keeps scripts, data, outputs organized.
- ▶ Integrates smoothly with git.

Project Folder Structure (Recommended)

Example of our folder structure

```
tutorial-publicecon/
  - 01-slides/
  - 02-code/
  - 03-output/
  - 04-data/
    - raw/
    - processed/
  - overview.pdf
  - README.md
  - tutorial-publicecon.Rproj
```

Key idea: Always use paths relative to the .Rproj file. No absolute file paths.

Running Code in RStudio

Console: non-reproducible quick experimentation.

- ▶ Commands are not saved.
- ▶ History stored in `.Rhistory` (should not be tracked via git).

Running Code in RStudio

Console: non-reproducible quick experimentation.

- ▶ Commands are not saved.
- ▶ History stored in `.Rhistory` (should not be tracked via git).

Script: Reproducible recipe for your analysis.

- ▶ Run a single line: `Ctrl + Enter / Cmd + Enter`.
- ▶ Run a selection: highlight + `Ctrl/Cmd + Enter`.
- ▶ Run an entire script: Source button.
- ▶ Scripts can be version-controlled with git. Direct handling without command line in right top panel.

Running Code in RStudio

Console: non-reproducible quick experimentation.

- ▶ Commands are not saved.
- ▶ History stored in `.Rhistory` (should not be tracked via git).

Script: Reproducible recipe for your analysis.

- ▶ Run a single line: `Ctrl + Enter / Cmd + Enter`.
- ▶ Run a selection: highlight + `Ctrl/Cmd + Enter`.
- ▶ Run an entire script: Source button.
- ▶ Scripts can be version-controlled with git. Direct handling without command line in right top panel.

⇒ [Hands-on exercise](#).

- ① Set up R project in the course folder, add folder structure and check `.gitignore`.
- ② Generate new script `<lastname>-session-01.R`.
- ③ Save snapshot via git.

Packages in R

Why packages exist:

- ▶ Provide tools for specific tasks (e.g. econometrics, plotting, data import).
- ▶ Allow researchers to share methods and workflows.

How to use a package:

- ▶ **Install once:** via console `install.packages("<NAME>")` or via “Packages” panel in RStudio on the bottom right.
- ▶ **Load in every session:** via `library(<NAME>)` in script.

Important packages for today:

- ▶ tidyverse — collection of packages for data wrangling (`dplyr`), plots (`ggplot2`), and more. ⇒ Nice for learning, because consistent syntax.
- ▶ haven — import Stata/SPSS data files.
- ▶ ... later we will add more.

Basic commands for data analysis

R Basics — Creating Objects

Assignment operator:

- ▶ `x <- 5` assigns 5 to object `x`.
- ▶ Preferred over `=` for clarity and consistency.

Basic object types:

- ▶ vectors (numeric, character, logical)
- ▶ data frames / tibbles (tables)
- ▶ lists (mixed objects)

Example

```
x <- 1:5
df <- tibble(id = x, income = x*1000)
```

R Basics — Loading Data in R

Typical workflow: Import data into R using functions from tidyverse and haven.

Load a .csv file (using `readr`)

```
library(readr)
data <- read_csv("04-data/raw/xxx.csv")
```

Load a Stata (.dta) file (using `haven`)

```
library(haven)
data <- read_dta("04-data/raw/xxx.dta")
```

⇒ Today, we will use [training data](#) from the [German Socioeconomic Panel \(GSOEP\)](#).

Data Analysis — The Pipe Operator %>%

Reads as “and then”. It sends the output of one function into the next.

Example

```
data %>%
  filter(age > 30) %>%
  summarise(mean_income = mean(income_tot_y, na.rm = TRUE))
```

Why useful? Linear, readable, avoids intermediate objects.

Data Analysis — Core Verbs in dplyr

- ▶ `filter()` – select rows
- ▶ `select()` – pick columns
- ▶ `mutate()` – create new variables
- ▶ `arrange()` – reorder rows
- ▶ `group_by()` – define groups
- ▶ `summarise()` – summary stats

Example

```
data %>%
  group_by(sex) %>%
  summarise(mean_income = mean(income_tot_y))
```

Data Analysis — Reshaping Data with `tidyR`

Key functions:

- ▶ `pivot_longer()` – wide → long
- ▶ `pivot_wider()` – long → wide

Example

```
data_long <- data_wide %>%  
pivot_longer(cols = starts_with("income_"),  
names_to = "type",  
values_to = "amount")
```

Data Analysis — Plotting with ggplot2

Graphics: build plots in layers (similar to data analysis with dplyr)

Example

```
ggplot(data, aes(x = age, y = income_tot_y)) +  
  geom_point(alpha = 0.3) +  
  geom_smooth() +  
  labs(x = "Age", y = "Total Income")
```

Finding Help in R

Learning to look up documentation is essential.

Help pages:

- ▶ `?filter`
- ▶ `?pivot_longer`

Vignettes (tutorial-style docs):

- ▶ `vignette("dplyr")`
- ▶ `vignette("ggplot2")`

Search:

- ▶ `help.search("regression")`

Good practice: When you forget a function, look up its help page.

A note on LLM tools

- LLM tools (ChatGPT, Claude AI, ...) are extremely useful for learning, debugging, and getting help. Some integrate directly into RStudio:

A note on LLM tools

- ▶ LLM tools (ChatGPT, Claude AI, ...) are extremely useful for learning, debugging, and getting help. Some integrate directly into RStudio:
 - ▶ **Ask questions** inside RStudio (e.g. [gptstudio](#), [chattr](#)).

A note on LLM tools

- ▶ LLM tools (ChatGPT, Claude AI, ...) are extremely useful for learning, debugging, and getting help. Some integrate directly into RStudio:
 - ▶ **Ask questions** inside RStudio (e.g. [gptstudio](#), [chattr](#)).
 - ▶ **Rewrite / auto-complete code** in scripts (e.g. [gander](#) package, [GitHub Copilot](#)).

A note on LLM tools

- ▶ LLM tools (ChatGPT, Claude AI, ...) are extremely useful for learning, debugging, and getting help. Some integrate directly into RStudio:
 - ▶ **Ask questions** inside RStudio (e.g. [gptstudio](#), [chattr](#)).
 - ▶ **Rewrite / auto-complete code** in scripts (e.g. [gander](#) package, [GitHub Copilot](#)).
 - ▶ **Get code and project consulting** via Terminal (e.g. [Claude Code](#)).

A note on LLM tools

- ▶ LLM tools (ChatGPT, Claude AI, ...) are extremely useful for learning, debugging, and getting help. Some integrate directly into RStudio:
 - ▶ **Ask questions** inside RStudio (e.g. [gptstudio](#), [chattr](#)).
 - ▶ **Rewrite / auto-complete code** in scripts (e.g. [gander](#) package, [GitHub Copilot](#)).
 - ▶ **Get code and project consulting** via Terminal (e.g. [Claude Code](#)).

A note on LLM tools

- ▶ LLM tools (ChatGPT, Claude AI, ...) are extremely useful for learning, debugging, and getting help. Some integrate directly into RStudio:
 - ▶ **Ask questions** inside RStudio (e.g. [gptstudio](#), [chattr](#)).
 - ▶ **Rewrite / auto-complete code** in scripts (e.g. [gander](#) package, [GitHub Copilot](#)).
 - ▶ **Get code and project consulting** via Terminal (e.g. [Claude Code](#)).
- ⇒ **LLMs are great for learning, explanations, and debugging.**

A note on LLM tools

- ▶ LLM tools (ChatGPT, Claude AI, ...) are extremely useful for learning, debugging, and getting help. Some integrate directly into RStudio:
 - ▶ **Ask questions** inside RStudio (e.g. [gptstudio](#), [chattr](#)).
 - ▶ **Rewrite / auto-complete code** in scripts (e.g. [gander](#) package, [GitHub Copilot](#)).
 - ▶ **Get code and project consulting** via Terminal (e.g. [Claude Code](#)).
- ⇒ **LLMs are great for learning, explanations, and debugging.**
- ▶ BUT:

A note on LLM tools

- ▶ LLM tools (ChatGPT, Claude AI, ...) are extremely useful for learning, debugging, and getting help. Some integrate directly into RStudio:
 - ▶ Ask questions inside RStudio (e.g. [gptstudio](#), [chattr](#)).
 - ▶ Rewrite / auto-complete code in scripts (e.g. [gander](#) package, [GitHub Copilot](#)).
 - ▶ Get code and project consulting via Terminal (e.g. [Claude Code](#)).
- ⇒ **LLMs are great for learning, explanations, and debugging.**
- ▶ BUT:
 - ▶ Tools can be **incorrect** even with moderately complex tasks.

A note on LLM tools

- ▶ LLM tools (ChatGPT, Claude AI, ...) are extremely useful for learning, debugging, and getting help. Some integrate directly into RStudio:
 - ▶ Ask questions inside RStudio (e.g. [gptstudio](#), [chattr](#)).
 - ▶ Rewrite / auto-complete code in scripts (e.g. [gander](#) package, [GitHub Copilot](#)).
 - ▶ Get code and project consulting via Terminal (e.g. [Claude Code](#)).
- ⇒ **LLMs are great for learning, explanations, and debugging.**
- ▶ BUT:
 - ▶ Tools can be **incorrect** even with moderately complex tasks.
 - ▶ They handle sub-tasks well, but do **not reliably organize a full research workflow**.

A note on LLM tools

- ▶ LLM tools (ChatGPT, Claude AI, ...) are extremely useful for learning, debugging, and getting help. Some integrate directly into RStudio:
 - ▶ Ask questions inside RStudio (e.g. [gptstudio](#), [chattr](#)).
 - ▶ Rewrite / auto-complete code in scripts (e.g. [gander](#) package, [GitHub Copilot](#)).
 - ▶ Get code and project consulting via Terminal (e.g. [Claude Code](#)).
- ⇒ **LLMs are great for learning, explanations, and debugging.**
- ▶ BUT:
 - ▶ Tools can be **incorrect** even with moderately complex tasks.
 - ▶ They handle sub-tasks well, but do **not reliably organize a full research workflow**.
 - ▶ Over-reliance makes you forget core skills.

A note on LLM tools

- ▶ LLM tools (ChatGPT, Claude AI, ...) are extremely useful for learning, debugging, and getting help. Some integrate directly into RStudio:
 - ▶ Ask questions inside RStudio (e.g. [gptstudio](#), [chattr](#)).
 - ▶ Rewrite / auto-complete code in scripts (e.g. [gander](#) package, [GitHub Copilot](#)).
 - ▶ Get code and project consulting via Terminal (e.g. [Claude Code](#)).
- ⇒ **LLMs are great for learning, explanations, and debugging.**
- ▶ BUT:
 - ▶ Tools can be **incorrect** even with moderately complex tasks.
 - ▶ They handle sub-tasks well, but do **not reliably organize a full research workflow**.
 - ▶ Over-reliance makes you forget core skills.
 - ▶ Never paste **sensitive data** into online tools.

A note on LLM tools

- ▶ LLM tools (ChatGPT, Claude AI, ...) are extremely useful for learning, debugging, and getting help. Some integrate directly into RStudio:
 - ▶ Ask questions inside RStudio (e.g. [gptstudio](#), [chattr](#)).
 - ▶ Rewrite / auto-complete code in scripts (e.g. [gander](#) package, [GitHub Copilot](#)).
 - ▶ Get code and project consulting via Terminal (e.g. [Claude Code](#)).
- ⇒ **LLMs are great for learning, explanations, and debugging.**
- ▶ BUT:
 - ▶ Tools can be **incorrect** even with moderately complex tasks.
 - ▶ They handle sub-tasks well, but do **not reliably organize a full research workflow**.
 - ▶ Over-reliance makes you forget core skills.
 - ▶ Never paste **sensitive data** into online tools.

A note on LLM tools

- ▶ LLM tools (ChatGPT, Claude AI, ...) are extremely useful for learning, debugging, and getting help. Some integrate directly into RStudio:
 - ▶ Ask questions inside RStudio (e.g. [gptstudio](#), [chattr](#)).
 - ▶ Rewrite / auto-complete code in scripts (e.g. [gander](#) package, [GitHub Copilot](#)).
 - ▶ Get code and project consulting via Terminal (e.g. [Claude Code](#)).
- ⇒ **LLMs are great for learning, explanations, and debugging.**

- ▶ BUT:
 - ▶ Tools can be **incorrect** even with moderately complex tasks.
 - ▶ They handle sub-tasks well, but do **not reliably organize a full research workflow**.
 - ▶ Over-reliance makes you forget core skills.
 - ▶ Never paste **sensitive data** into online tools.
- ⇒ **Use LLM tools with caution and critical thinking.**

That's it! See you next time!