

Lesson 2: Reproducible workflows with Quarto & GitHub

Course website: https://hannahmetzler.eu/R_intro/

Hannah Metzler

2025-02-06

Table of contents

0.1	Quick repetition of lesson 1	2
0.2	Goals for lesson 2	3
0.3	File names and system	3
0.3.1	Naming things	3
0.3.2	Navigating the file system: Terminal & bash	4
0.3.3	Working Directory	6
0.4	Setup Git & GitHub repository for your project	7
0.4.1	What for?	7
0.4.2	How Git works	7
0.4.3	Create a remote git repository	8
0.4.4	Clone the remote Git Repository to your laptop	9
0.5	Organize your project	10
0.5.1	R-projects	10
0.5.2	Folder structure	11
0.6	Backup your project on GitHub	12
0.6.1	Commit to Git	12
0.6.2	Upload to GitHub	13
0.7	Read in and manipulate data	13
0.7.1	Look at the data	14
0.7.2	Select a part of the data	16
0.8	Make a figure with ggplot2	17
0.9	Commit your work to Git	19
0.10	Create a report using Quarto	20
0.10.1	What is Quarto?	20
0.10.2	Let's make a Quarto Report of our results!	20

html: theme: flatly

0.1 Quick repetition of lesson 1

- Assigning numeric values or strings to objects, then using the objects:

```
x <- 5
y <- "string"
x*2
```

```
[1] 10
```

```
print(y)
```

```
[1] "string"
```

- Using whitespaces to organize your code
- Removing objects in the environment

```
rm(list = ls())
```

- Functions & arguments

```
rnorm(20, mean = 100, sd = 10)
```

```
[1] 109.71562  91.34933 116.70069  84.79243 100.19491  86.72420 121.43935
[8] 100.90289 114.26247 116.54353 109.84438 103.11011  96.33643 104.70419
[15] 101.12897 110.27493  89.61853  94.29819  91.20694  94.97448
```

- Function help

```
?rnorm
```

- Installing & loading packages

```
# install.packages("tidyverse")
library(ggplot2)
library(dplyr)
library(readr)
```

0.2 Goals for lesson 2

- Commit to Git
- Push to GitHub
- Organize a project (folder, files, R-project)
- Read in & manipulate data
- Make a figure & save it to PDF
- Create a Quarto document

0.3 File names and system

Before we create our first R-project, we need to talk about how to name files and folders, and how the file system and file paths on your PC work, and how you navigate your way through it.

0.3.1 Naming things

- Use names that work well for computers & people
- No spaces
- Be consistent with capitalisation (set a rule to make it easy to remember, like always use lowercase)
- A full stop only before the file extension `data_file.csv`
- Use underscores (`_`) to separate parts of the file name, and dashes (`-`) to separate words in a section "`data_questionnaire_2021-11-15.xls`"
- Prefix a file name with an underscore to move it to the top of the list, or prefix all files with numbers to control their order
- Use YYYY-MM-DD format for dates so files sort in chronological order.

0.3.1.1 Bad examples

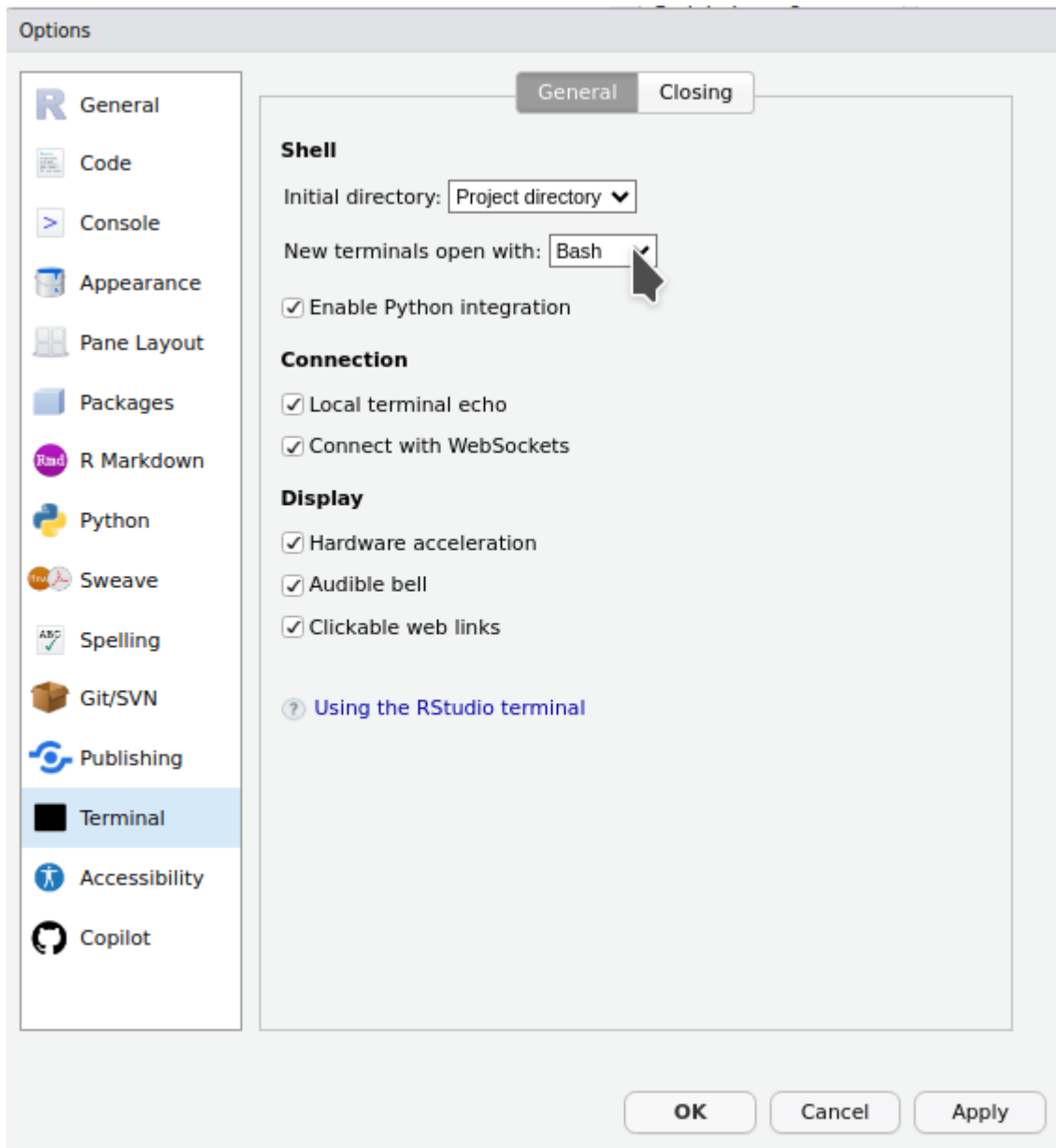
- Data (Participants) 11-15.xls
- final report2.doc
- Participants Data Nov 12.xls
- project notes.txt
- Questionnaire Data November 15.xls
- report final.doc
- report final 2.doc

0.3.1.2 Good examples

- `_project-notes.txt`
- `data_participants_2021-11-12.xls`
- `data_participants_2021-11-15.xls`
- `report_v1.doc`
- `report_v2.doc`
- `report_v3.doc`

0.3.2 Navigating the file system: Terminal & bash

- File paths: forward or backward slash to divide sub folders: `directory/subdirectory/`
 - `/` on Linux and Mac
 - `\` on Windows
- Your user accounts' main directory: `/Users/UserName/`, or on Windows: `C:\Users\Username\`
- A file on your Desktop: `/Users/UserName/Desktop/datafile.csv`
- You can navigate your file system via the **Terminal**.
 - On Mac & Linux (Unix-based systems) this software is called Terminal or Console outside of Rstudio
 - On Windows, the software we need is called: PowerShell or Git Bash.
 - Try to find the program in the same way you would find any other program, then close it again.
- You can access the Terminal from within R-studio (next to the Console at the bottom left in Rstudio).
 - On Windows, make sure you use the right Terminal in the Global Options:
 - Choose Git Bash or PowerShell here.
 - The default Terminal of some older Windows systems is **Command Prompt** - it won't work in the same way.



- In the Terminal, we will use the coding language **bash**.
- Unlike in R, white spaces have a meaning in **bash**, so make sure you get them all right!
- In **bash**, you can use these commands to navigate between folders:
 - `cd` change directory, e.g., to move to my user directory: `cd /Users/Hannah/`
 - `~` shortcut for the user account directory `/Users/UserName/`, e.g., to move to the user directory: `cd ~`
 - `..` move one directory upwards: `cd ..`

- Note how there is always a white space after the function `cd`. In R, we would have `()` instead: `function()`

💡 Tip

I find it useful to have all my analysis projects in a common folder directly in my user directory: `~/AnalysisProjects/`. In it, you could then have a folder `R_introduction` for this course.

0.3.3 Working Directory

- Folder of the project you are currently working on
- Never change the working directory in a script
- Your scripts should only reference three types of locations:

Where	Example
on the web	“ <code>https://hannahmetzler.eu/R_intro/Lesson_2/data/datafile.csv</code> ”
in the working directory	“ <code>datafile.csv</code> ”
in a subdirectory	“ <code>data/datafile.csv</code> ”

- Typing `pwd` (print working directory) in the Terminal shows you the directory you are currently in.

🔥 Exercise

- In Rstudio, go to the **Terminal**
- Check which directory you are in
- Navigate to your user directory
- Move back up one folder
- Move back to your user directory using the shortcut

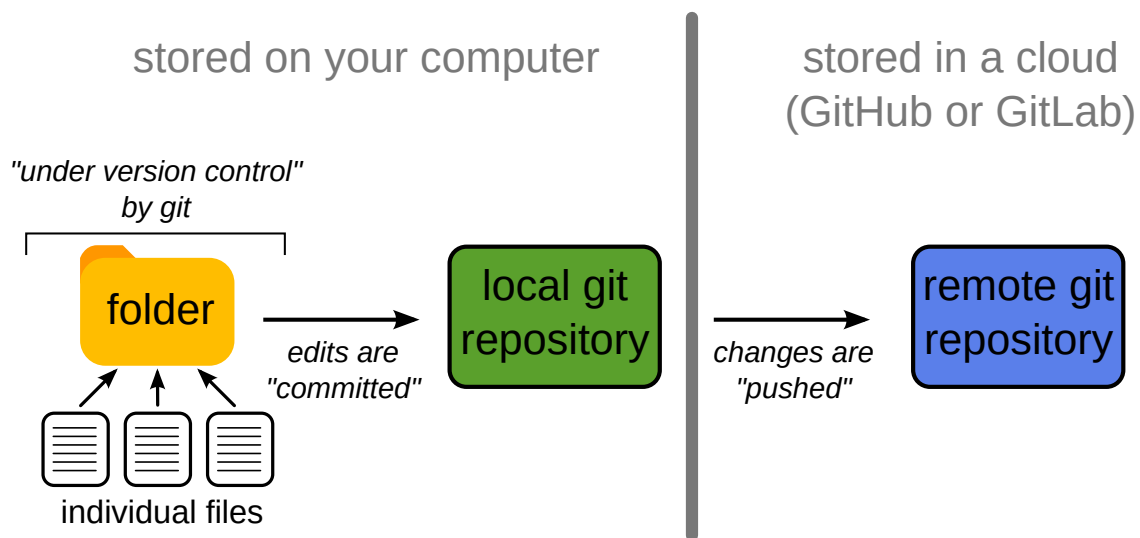
0.4 Setup Git & GitHub repository for your project

- We cover the absolute basics here, for more: <http://happygitwithr.com/>
- Do you all have Git Setup & a GitHub account?

0.4.1 What for?

- Keeping your coding projects organized in one place
- Backup your code & data
- Version control: edits over time
- Sharing & collaborating on code

0.4.2 How Git works



- Committing: creating a named/time stamped version on your PC
- Pushing: Uploading to GitHub
- Repository: directory/folder registered by Git

💡 Tip

You don't technically need to use Git and GitHub to code your data analysis. You could just store your files locally. BUT: I strongly recommend you make a Git repository for every data analysis project. It really helps to keep things organized in one place, not lose your code, go back to earlier versions, share it, collaborate on it, and so on. To practice, we will create a repository for every lesson.


0.4.3 Create a remote git repository

We will now create a remote repository on Github, copy it to our local PCs, and then continue Lesson 2 in that local repository.

- On <https://github.com/> click on **New** on the top left.
- For later: you can navigate to your repositories via clicking on your profile and then choose **Your repositories** or **Repositories**.

As shown in the screenshot below:


- Name the repository
- Make it private
- Add a README file
- Choose the template R for the .gitignore file
- (No License needed)
- Click **Create repository**:


Owner *  hannahmetzler / **Repository name ***

✓ **R_intro_lesson2 is available.**

Great repository names are short and memorable. Need inspiration? How about **musical-octo-disco** ?

Description (optional)

☐  **Public**
Anyone on the internet can see this repository. You choose who can commit.

☒  **Private**
You choose who can see and commit to this repository.

Initialize this repository with:


☒ **Add a README file**
This is where you can write a long description for your project. [Learn more about READMEs.](#)


Add .gitignore

Choose which files not to track from a list of templates. [Learn more about ignoring files.](#)

Choose a license

A license tells others what they can and can't do with your code. [Learn more about licenses.](#)

This will set  **main** as the default branch. Change the default name in your [settings](#).

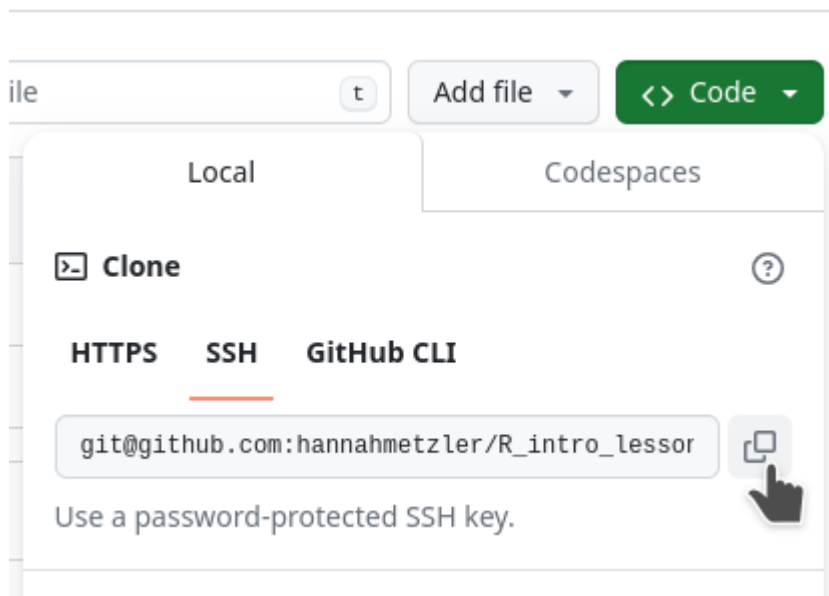
 You are creating a private repository in your personal account.

[Create repository](#)

- **README.md**: where you describe the project (folder structure, content, etc.)
- **.gitignore**: Hidden file that tells Git what not to upload to GitHub (e.g., figures)

0.4.4 Clone the remote Git Repository to your laptop

1. Copy the SSH link of your repository (for connecting it to Rstudio):



2. In Rstudio, in the Terminal:

- Navigate to the directory where you put the R_introduction folder using `cd`. (I want it in my folder `AnalysisProjects`)
- `git clone` and paste your repository link:

```
hannah@TP-X1-Ext:~$ pwd
/home/hannah
hannah@TP-X1-Ext:~$ cd AnalysisProjects/
hannah@TP-X1-Ext:~/AnalysisProjects$ git clone git@github.com:hannahmetzler/R_intro_lesson2.git
Cloning into 'R_intro_lesson2'...
remote: Enumerating objects: 4, done.
remote: Counting objects: 100% (4/4), done.
remote: Compressing objects: 100% (4/4), done.
remote: Total 4 (delta 0), reused 0 (delta 0), pack-reused 0
Receiving objects: 100% (4/4), done.
hannah@TP-X1-Ext:~/AnalysisProjects$
```

There is now a folder with the name of the remote repository in the directory you “cloned” it to. (For me, in the `AnalysisProjects/` folder.) In that folder, let’s start our first R-project!

0.5 Organize your project

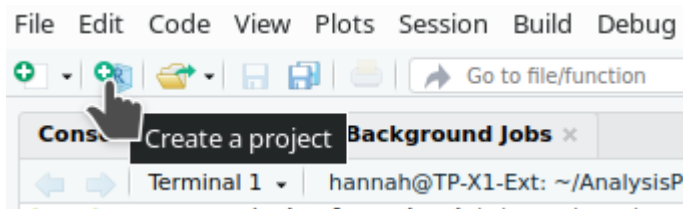
0.5.1 R-projects

- Useful to organize your coding projects
- Group all files of a project in a directory
- Allows using Git for version control

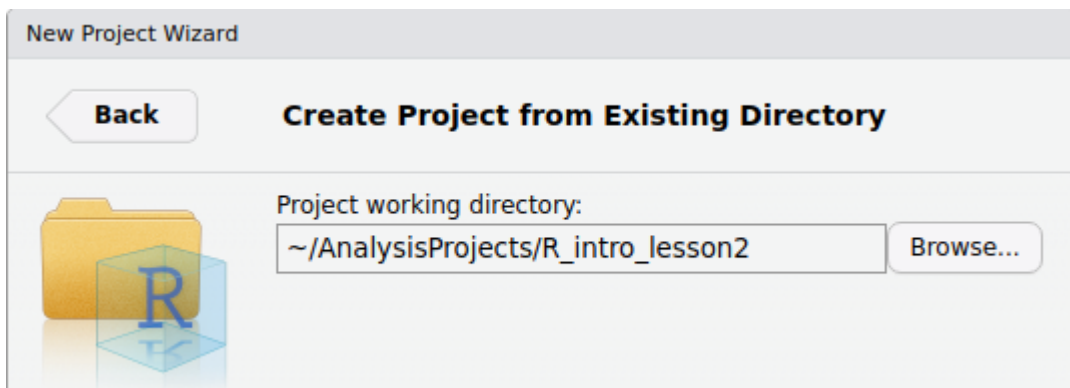
- R-project automatically sets working directory to the R-project folder
- In the top right corner in Rstudio, you can see that we are not currently in any project.
- Let's create a project for this lesson:

🔥 Exercise

1. Create a project in Rstudio: Choose **Create a project** (or **New Project** in the **File** menu).



2. Choose Existing Directory
3. Use **Browse** to navigate to our folder `R_intro_lesson_2` and click **Create Project**



From now on, you can double click this Rproject file in your `R_intro_lesson2` folder to directly open Rstudio from this folder.

0.5.2 Folder structure

🔥 To do

Create these folders inside the main project directory `R_intro_lesson2`:

- data
- code
- figures

- write_up

Download this [data file](#) from the course website and put it into the data folder. We'll be writing a script to analyse it during this session. If needed, you can also download it [here](#).

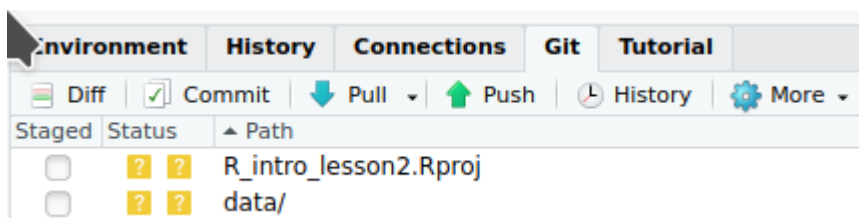
- Alternative/additional folder names in future projects could be:
 - tables
 - reports
 - output (tables, reports, ...)
 - raw_data: I recommend having a separate folder for the raw data
 - code = scripts
- Plus any other project specific folders

0.6 Backup your project on GitHub

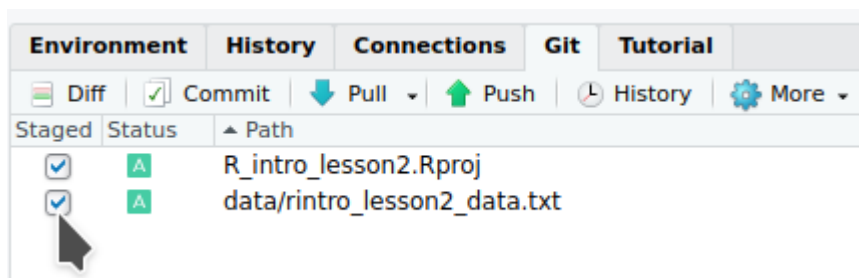
0.6.1 Commit to Git

We can now back up the project and data to a “remote repository” on GitHub.

1. Click on **Git** in the upper right pane.
2. You see 2 files to commit:

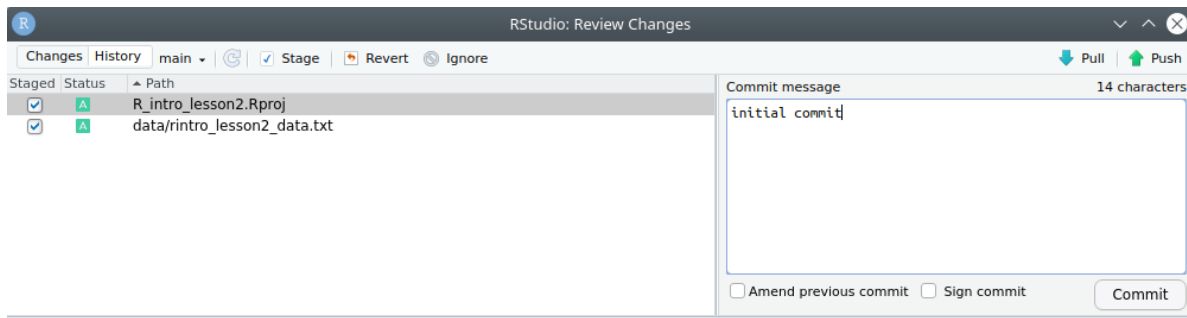


3. Add them to git by checking the box



4. Click commit

5. Write a commit message (describe what the commit contains)



6. Click Commit again, then Close.

0.6.2 Upload to GitHub

- Click **Push**.
- Go to your remote repository on GitHub. Do you see the folder and files you added?

0.7 Read in and manipulate data

We will finally start coding!

To do

- Open an R-script by clicking on the green plus below **File** (top left)
- Save it in the code folder, e.g. with the name `rintro_lesson2`
- At the top of the script, we load all required packages.
- Lines with `#` are comments, so R does not read them as code.
 - Keyboard shortcut to (un)comment a line or selection: `Ctrl+Shift+C` / `Cmd+Shift+C`
- Lines ending with `####` are code blocks, that can be opened and closed. Use them to organize your script!
- To run a line in a script, press `Ctrl + Enter`
- To run an entire R-script: `Ctrl + Shift + Enter`

To do

Write and run the code below:

```
## LOAD PACKAGES ####  
library(tidyverse)
```

- To read in the data, we use `read_csv()` from the package `readr`.

```
## READ IN DATA AND ORGANIZE ####  
# Read in data  
data = readr::read_csv("data/pets.csv", col_types = "cffiid")
```

- The working directory is in the root folder, so we need to first indicate that the datafile is in the data folder: `"data/pets.csv"`
- The last part `col_types = "cffiid"` defines the type of variable - we will get to this below.

0.7.1 Look at the data

The first and last rows of the data frame:

id	pet	country	score	age	weight
S001	dog	UK	90	6	19.78932
S002	dog	UK	107	8	20.01422
S003	dog	UK	94	2	19.14863
S004	dog	UK	120	10	19.56953
S005	dog	UK	111	4	21.39259
S006	dog	UK	110	8	21.31880

id	pet	country	score	age	weight
S795	ferret	NL	116	1	3.0276944
S796	ferret	NL	118	7	3.2127433
S797	ferret	NL	98	10	0.7909462
S798	ferret	NL	123	3	3.9612231
S799	ferret	NL	103	8	6.9195795
S800	ferret	NL	120	5	0.5048159

- `data` is our data frame (the whole data table with variables as columns and observations as rows)
- `id`, `pet`, `score`, `age`, `weight` are variables
- `dog` and `ferret` are levels of the variable `pet`

- UK and NL are levels of the variable `country`

Now, we'll have a look at the data using these functions (from basic R):

- `dim()`: number of rows and columns
- `head()`: print the first 6 rows
- `tail()`: print the last 6 rows
- `xtabs()`: how many data points exist for each level of a variable

To do

Write and then run the code below. After each written line, use **Ctrl + Enter** to run it.

```
# Look at data
dim(data)
head(data)
tail(data)
xtabs(~pet, data)
```

All of this is basic R. In addition, we can use functions from packages. Using `glimpse()` from `dplyr` gives you an overview of the data:

```
dplyr::glimpse(data)
```

```
Rows: 800
Columns: 6
$ id      <chr> "S001", "S002", "S003", "S004", "S005", "S006", "S007", "S008"~
$ pet     <fct> dog, dog, dog, dog, dog, dog, dog, dog, dog, dog, dog, dog, do~
$ country <fct> UK, UK, UK, UK, UK, UK, UK, UK, UK, UK, UK, UK, UK, UK, UK~
$ score   <int> 90, 107, 94, 120, 111, 110, 100, 107, 106, 109, 85, 110, 102, ~
$ age     <int> 6, 8, 2, 10, 4, 8, 9, 8, 6, 11, 5, 9, 1, 10, 7, 8, 1, 8, 5, 13~
$ weight  <dbl> 19.78932, 20.01422, 19.14863, 19.56953, 21.39259, 21.31880, 19~
```

Here, you can see that the data frame contains different types of variables:

- `chr` (character): for variables with text (can contain any character: letter, numbers, symbols, spaces)
- `fct` (factor): for categorical variables
- `int` (integer): numeric, stores whole numbers
- `dbl` (double): numeric, numbers with decimals

0.7.2 Select a part of the data

There are always lots of ways to do the same thing in R.

Let's say we want to create a new data frame that contains only cats data. Here are some ways to do that, which all produce exactly the same result. (Just examples, no need to understand this yet):

```
# basic R to extract specific rows from a dataframe
data_cats = data[data$pet == 'cat',]

# the function subset
data_cats = subset(data, pet == 'cat')

# using the function filter from the package dplyr
data_cats = data %>%
  dplyr::filter(pet == 'cat')
```

We'll use dplyr (the third way). dplyr is a package from the Tidyverse for

```
# Select only cats
data_cats <- data %>%
  dplyr::filter(pet == "cat")
```

Here is a step by step explanation:

- `data_cats`: new data frame
- `<-` (or `=`) is used to assign
- `data`: original data frame
- `%>%` (or `\|\>\|`): a “pipe” in dplyr - let's R know you are not done writing code, it waits until it gets a line not ending with `%>%` before executing the code.
- `filter()`: verb
- `pet`: variable
- `cat`: level of the variable
- `==` is a marker of relationship (like `<`, `>`)



Write down and run the code for selecting only cats data.

You can use these keyboard shortcuts:

- `<-` for assigning: `Alt + -` or `Option + -`
- Keyboard shortcut: `%>%` pipe: `Shift + Ctrl + M` or `Cmd + Shift + M`

Let's look at the top rows:


```
head(data_cats)
```

id	pet	country	score	age	weight
S401	cat	UK	95	2	8.571053
S402	cat	UK	96	7	11.196836
S403	cat	UK	72	4	10.307364
S404	cat	UK	99	6	7.243529
S405	cat	UK	85	3	6.650295
S406	cat	UK	115	2	3.616348

There are only rows with cats left.

Exercise

Look at the new data frame `data_cats` with all the functions you used above for `data`.

0.8 Make a figure with ggplot2

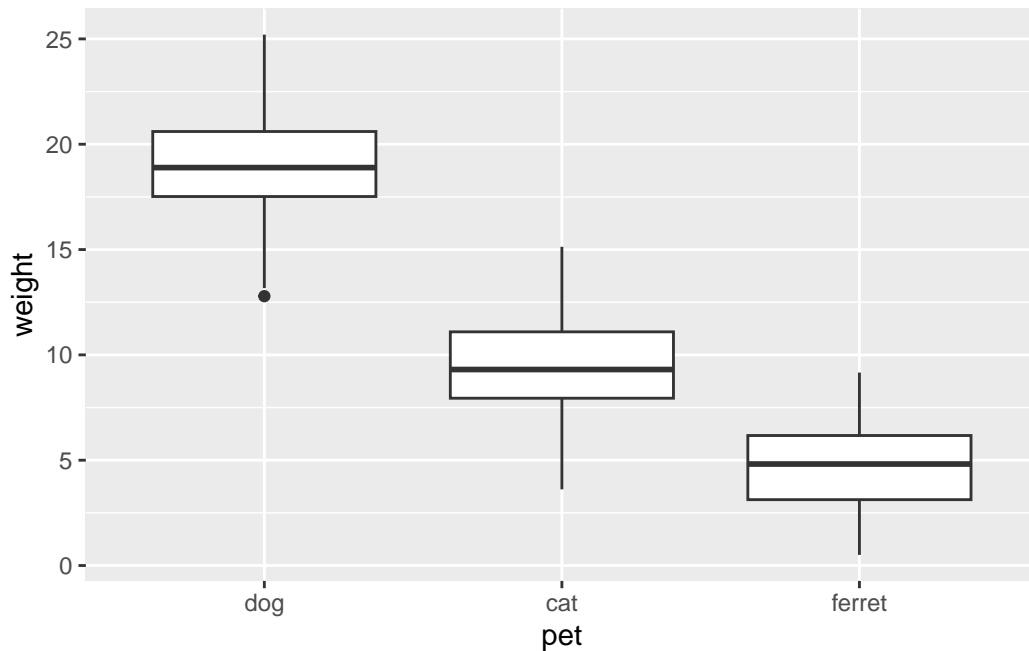
We'll make a box plot showing the weight of the different pets. `ggplot2` is a package from the Tidyverse that is very flexible for making pretty plots.

Plots in `ggplot` are created by adding elements on top of each other. See the vignette: We will use data, mapping and a layer for a box plot for now.

```
vignette("ggplot2")
```

Start again with a section header (####).

```
## MAKE FIGURES ####  
# Weight by pet  
data.plot <- ggplot(data, aes(x = pet, y = weight)) +  
  geom_boxplot()  
data.plot
```



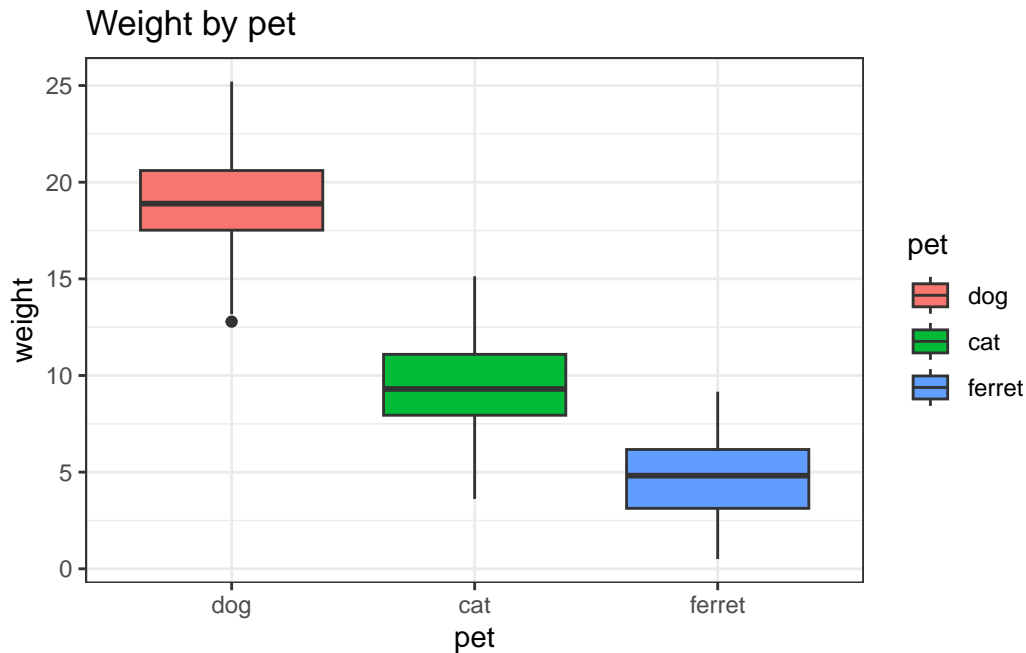
Step by step explanations:

- Every plot in ggplot2 is initiated with the call `ggplot()`
- Next comes the data frame (here `data`)
- Then we map the data to the aesthetics (`aes()`) of the figure: here only which variable onto the x- and y-axis.
- For ggplot2, we use `+` to connect lines (like `%>%` above for dplyr)
- In line 2 comes the type of plot (box plot). Most plots in ggplot2 with `geom_` plus the type of plot, so here `geom_boxplot`.
- We store the plot in the variable `data.plot`.
- We print out `data.plot` to see the plot.

It is very easy to make this plot pretty. With the code below:

- We fill each box with a colour by including `fill = pet` in the aesthetics.
- We change the design of the entire plot using one of the default themes: `theme_bw()`
- We add a title using `ggtitle()`

```
# Pretty plot
data.plot <- ggplot(data, aes(x = pet, y = weight, fill = pet)) +
  geom_boxplot() +
  theme_bw() +
  ggtitle("Weight by pet")
data.plot
```



To save the plot to a file to use in your paper, use `ggsave` to save it in our subfolder for figures:

```
# save the plot
ggsave('figures/boxplot_pets_weight.png', plot = data.plot, width = 6, height = 4, dpi = 300)
```

- Arguments in order:
 1. file name including the paths to the file
 2. plot you want to save
 3. width
 4. height
 5. resolution (300 dpi is often required for papers)
- Instead of .png, you could use other formats: .pdf, .jpg, .tiff and more.
- Check the figure in your `figures/` folder.

0.9 Commit your work to Git

- Go back to the **Git** menu at the top right and choose **Commit**
- Once again check all of the boxes of changed files (Select all files at once using Shift)
 - Write a message (e.g., “Looked at data, made boxplot”).
- Click **Commit** to store a new version of your project, then **Close**.

- To upload to GitHub, click **Push**.
- To confirm that it worked, go to your remote repository on github.com and refresh the page. Do you see the new files?

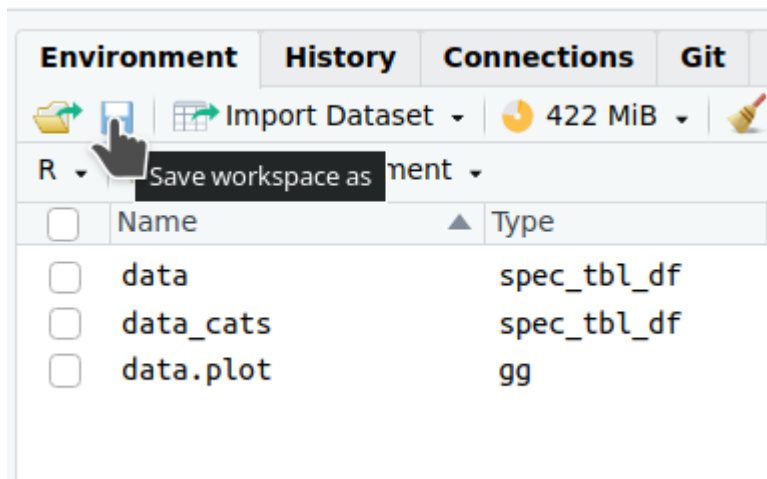
0.10 Create a report using Quarto

0.10.1 What is Quarto?

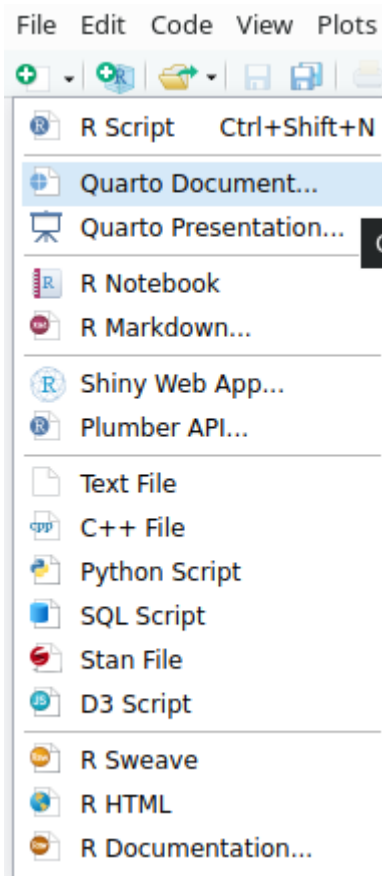
- Quarto enables you to create reproducible reports, presentations, publications & websites
- Output in various formats: HTML (websites & slides), PDF, Word
- From a variety of coding languages: R, Python, Julia, Observable Java Script
 - Quarto builds on R Markdown, which did the same but only for R.
- R script: only code (comments in lines starting with #)
- Quarto Document allows to combine text and code
- Useful to have automated reports about your results
- You could write entire papers (see [Data Skills for Reproducible Research](#))

0.10.2 Let's make a Quarto Report of our results!

- Save the objects in your **Environment** to the `write_up` folder, naming them `lesson2_environment`.





- Open a Quarto Document




- Enter a title and your name
- Choose HTML as output
- Click Create

New Quarto Document

 Document

 Presentation

 Interactive

Title: Lesson 2 Report

Author: Hannah Metzler

☒ **HTML**
Recommended format for authoring (you can switch to PDF or Word output anytime)

☐ **PDF**
PDF output requires a LaTeX installation (e.g. <https://yihui.org/tinytex/>)

☐ **Word**
Previewing Word documents requires an installation of MS Word (or Libre/Open Office on Linux)

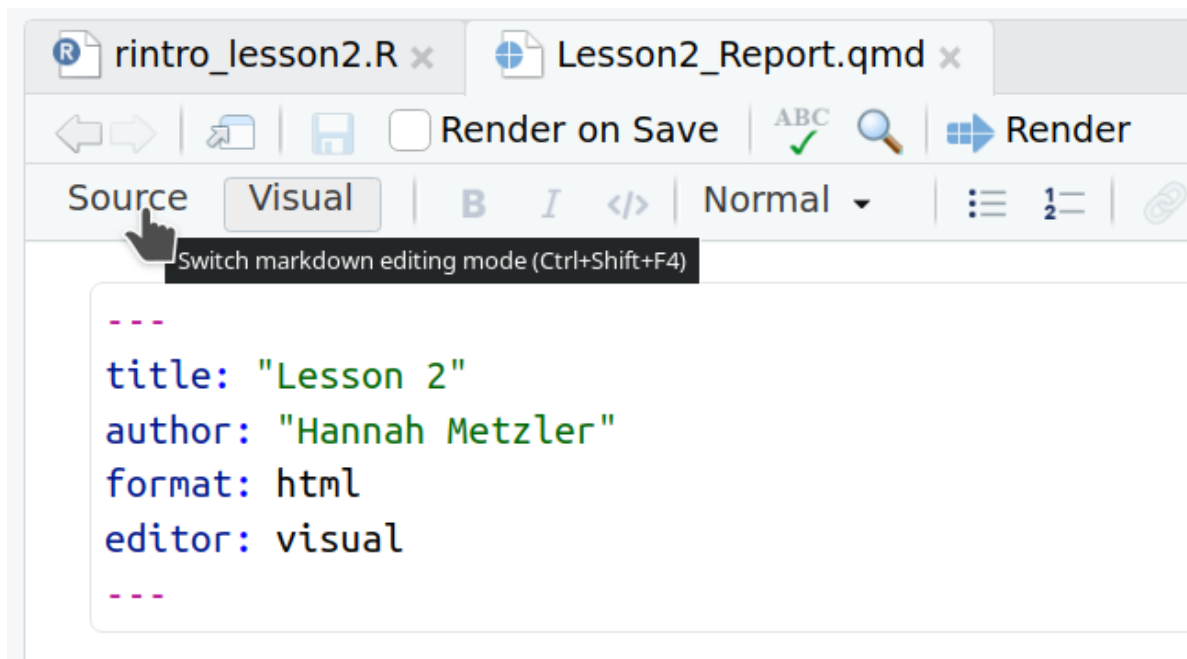
Engine: Knitr

Editor: ☒ Use visual markdown editor ?

? [Learn more about Quarto](#)

Create Empty DocumentCreateCancel

- Save the Quarto document in your write_up folder Lesson2_Report
- You can see that your R-script ends with .R, the Quarto file with .qmd



- By clicking on **Source**, you will see the content written in “Markdown” language.
- **Markdown** is
 - (1) a plain text formatting syntax, and
 - (2) a software tool that converts text to HTML.

Let's start writing in our Quarto Document:

- Delete the example text after the heading **Quarto** (including the heading, but not everything above)
- Use **Ctrl+Alt+I** to insert a code chunk. “`{r}`” lets R know that the following is code.
- In this code chunk, we load the data we have saved before.
- Quarto files directly run in the folder where they are saved, so here directly in the `write_up` folder.

```
load("lesson2_environment.RData")
```

- Start writing normal text (as below) in your document. Using `#` turns text into a title in Markdown.

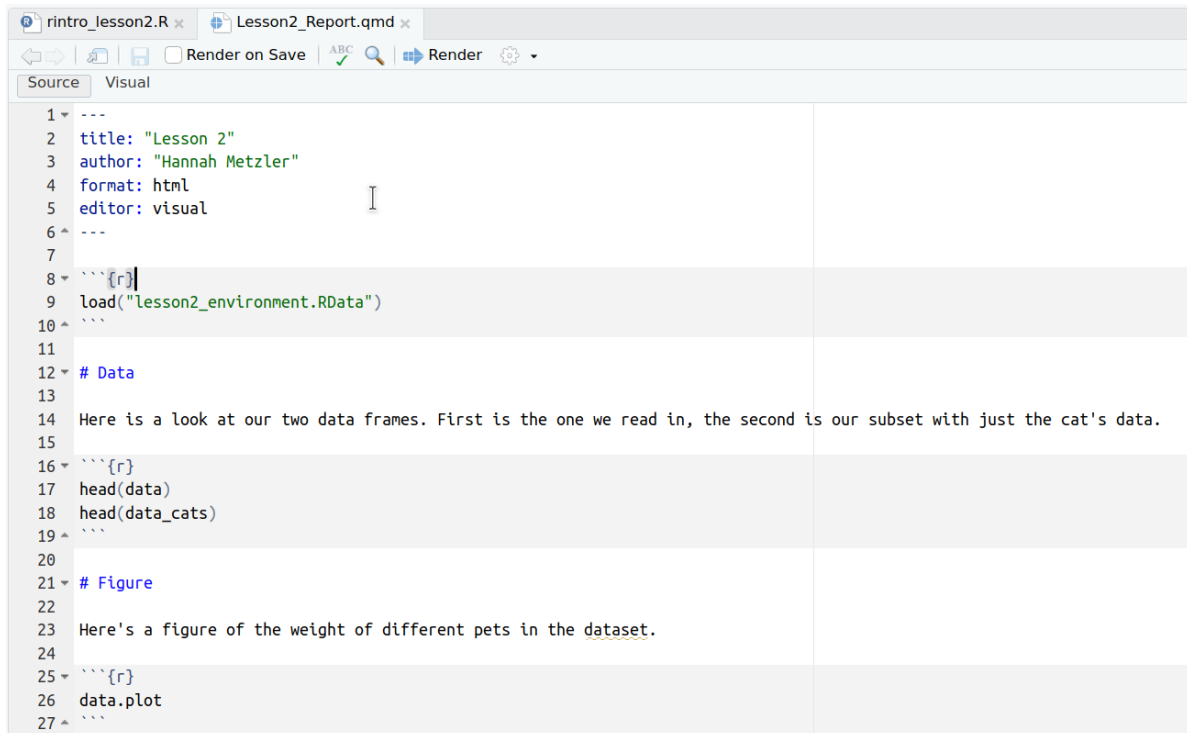
```
# Data
```

Here is a look at our two data frames. First is the one we read in, the second is our subset

```
# Figure
```

Here's a figure of the weight of different pets in the dataset.

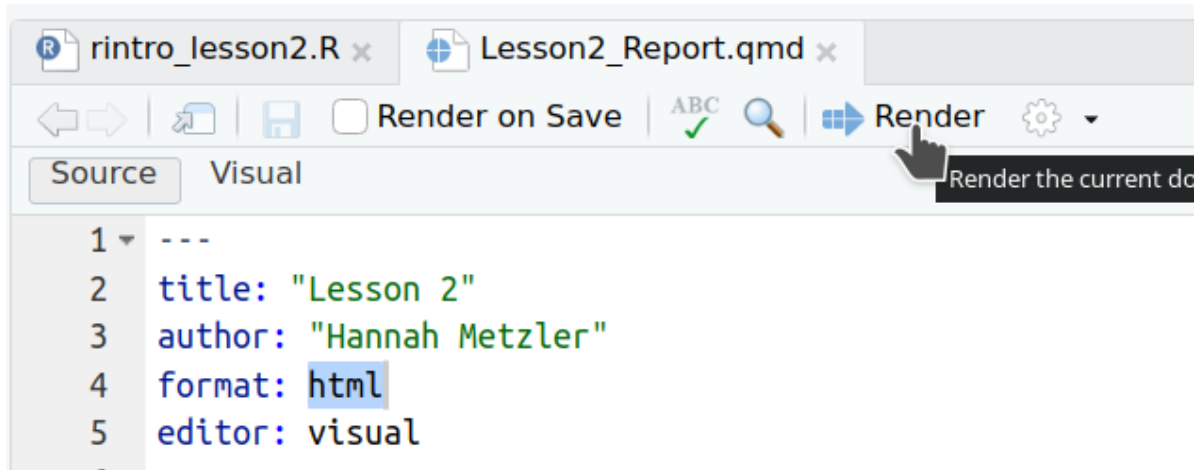
- Now, we will fill in code chunks to present our data and figure. Use **Ctrl+Alt+I** to insert them in the right places.
- Use `head(data)` and `head(data_cats)` to present your data sets under `# Data`.
- Use the object in which you stored the plot for the figure (`data.plot`) under `# Figure`.
- This is how it should look like:



The screenshot shows a web-based editor for a Quarto document. The top bar includes tabs for 'rintro_lesson2.R' and 'Lesson2_Report.qmd', along with icons for navigation, saving, and rendering. The main editor area is split into 'Source' and 'Visual' views. The 'Source' view shows a document template with the following content:

```
1 ---
2 title: "Lesson 2"
3 author: "Hannah Metzler"
4 format: html
5 editor: visual
6 ---
7
8 ```{r}
9 load("lesson2_environment.RData")
10 ```
11
12 # Data
13
14 Here is a look at our two data frames. First is the one we read in, the second is our subset with just the cat's data.
15
16 ```{r}
17 head(data)
18 head(data_cats)
19 ```
20
21 # Figure
22
23 Here's a figure of the weight of different pets in the dataset.
24
25 ```{r}
26 data.plot
27 ```
```

- If you click on **Render** at the top, you will get a HTML file with your report. Open it from the folder.



- You can simply switch the format to `format: docx` at the top, then render again as a Word document.
- `format: pdf` would also work, but takes a bit longer the first time because of installations.

💡 Tip

You can write a lot of your code directly in Quarto Documents, allowing to combine your results (tables, figures) with interpretations directly in the same document. Lead through your entire analyses step by step. If a step needs a lot of code, move that code to a script in the code folder, and call the script from your Quarto document using the function `source()`. For example, you could first call a script that cleans the data, then make figures and run statistical tests directly in the Quarto Document.

0.11 References

- The completed code for the lesson can be found [here](#).

This lesson is based on:

- Page Piccinini, R-Course, [Lesson 1: R Basics](#).
- Chapter 2 of: Lisa DeBruine & Dale Barr. (2022). [Data Skills for Reproducible Research](#): (3.0) Zenodo. doi:10.5281/zenodo.6527194.