

Live coding: your first Quarto report

Session 1 — Applied Data Science 2026

Claudius Gräbner-Radkowsch

2026-09-14

Table of contents

1	Step 0: set up a project	2
2	Step 1: the console vs. the script	3
3	Step 2: create the Quarto document	4
3.1	The YAML header	4
3.2	Writing text	4
4	Step 3: your first code chunk	5
5	Step 4: load the data	5
6	Step 5: look before you leap	5
7	Step 6: your first chart	6
7.1	Interrogating the chart	7
8	Step 7: adding a second variable	10
9	Step 8: render	11
10	Step 9: useful chunk options	12
11	Recap	13

This is the document we build together in the session, written from an empty file step by step. Use it afterwards as a model for your own reports.

1 Step 0: set up a project

Before writing a single line of code, give your work a home.

! Do this now — this should be the first step for each new project of yours

1. In RStudio: **File** → **New Project** → **New Directory** → **New Project**
2. Name it **applied-data-science**
3. Put it somewhere you will find it again — *not* in Downloads or Desktop
4. Click **Create Project**
5. Using **New Folder** in the Files pane, create **data**, **R** and **Output**
6. Download **hotels-vienna.csv** from the session page and save it into **data/**

You should end up with this:

```
applied-data-science/  
  applied-data-science.Rproj  ← created for you; never edit it  
  data/  
    hotels-vienna.csv          ← raw data, read-only. Never edit by hand.  
  R/                          ← .R scripts (we start using this in session 2)  
  Output/                     ← rendered reports, exported figures and tables  
  session01.qmd               ← the document we are about to create
```

The rule for each folder: **data/** goes in and never changes; **R/** holds reusable code; **Output/** holds anything produced *by* your code, and should be safe to delete because you can always regenerate it.

i How you organise the semester is up to you

I keep **one project for the whole course** and you are welcome to copy that. One project *per session* works just as well, and nothing in this course depends on which you pick. Reports are the exception — each report gets its own project, for reasons we come to in session 3.

Two things matter whichever way you go:

- **The .qmd you are working on sits at the top of the project, with data/ beside it.** Every session reads its data as **data/something.csv**, so if you bury the document one folder deeper, the path stops working. Session 3 shows the proper fix for that; until then, keep it flat.
- **If you keep one project, name the files so that future-you can tell them apart:** **session01.qmd**, **session02.qmd**, and so on. By December there will be a dozen of them, and three files called **Untitled1.qmd** will tell you nothing.

RStudio restarts and you are now *inside* the project. Two things changed: the top-right corner shows the project name, and R's working directory is the project folder.

That second one matters more than it looks. It means you can write `data/hotels-vienna.csv` instead of `C:/Users/yourname/Desktop/stuff/final_v3/data/hotels-vienna.csv` — and your code will still work on someone else's computer.

The rule

Never write a file path that starts with `C:/` or `/Users/`. If you do, your analysis only runs on your machine, which means it is not reproducible, which means it is not evidence.

2 Step 1: the console vs. the script

Type this in the **Console** (bottom left) and press Enter:

```
2 + 2
```

```
[1] 4
```

R answers. Now something more useful — store a value:

```
revenue <- 12500  
costs <- 9800
```

Nothing is printed. Look at the **Environment** pane (top right): the two objects are now there. Ask for them:

```
revenue - costs
```

```
[1] 2700
```

The arrow `<-` is how you name things. Read it as “*gets*”: `revenue` gets 12500.

Console work disappears

Everything you type in the console is gone when you close RStudio. If you want to keep it, write in a file instead!

3 Step 2: create the Quarto document

File → **New File** → **Quarto Document**. Give it a title, choose HTML, click **Create**. Save it as `session01.qmd`.

RStudio gives you a file with three kinds of content:

Part	What it is	How to spot it
YAML header	Settings for the whole document	Between the two <code>---</code> lines at the top
Text	Your prose, in Markdown	Plain writing
Code chunks	R code that gets run	Between <code>```{r}</code> and <code>```</code>

3.1 The YAML header

Ours looks like this:

```
---
title: "What does a hotel room cost in Vienna?"
author: "Your Name"
date: today
format: html
---
```

That is the minimum. `format: pdf` would produce a PDF instead.

3.2 Writing text

Markdown is plain text with a few marks:

You write	You get
<code># Heading</code>	A big heading
<code>## Subheading</code>	A smaller one
<code>**bold**</code>	bold
<code>*italic*</code>	<i>italic</i>
<code>- item</code>	a bullet point

That is genuinely most of it.

4 Step 3: your first code chunk

Insert a chunk with **Ctrl/Cmd + Alt + I**, or type it out:

```
library(tidyverse)
library(flextable)
```

This loads the **tidyverse**, a collection of packages we use all semester. It also loads **flextable**, which makes tables look nicer in a report.

i Package vs. library

A *package* is installed once, with `install.packages("tidyverse")`. It is loaded *every session*, with `library(tidyverse)`. Installing is buying the book; loading is taking it off the shelf.

5 Step 4: load the data

```
hotels <- read_csv("data/hotels-vienna.csv")
```

We will take `read_csv()` apart properly next week. For today: it reads a comma-separated file and gives you a table.

i Where this data comes from

428 hotels in Vienna, price for **one night** in November 2017. From Békés & Kézdi (2021), *Data Analysis for Business, Economics and Policy*, licensed CC BY-NC-SA 4.0. Each row is one hotel.

6 Step 5: look before you leap

Three questions to ask of any new dataset.

How big is it?

```
nrow(hotels)
```

```
[1] 428
```

```
ncol(hotels)
```

```
[1] 24
```

What is in it?

```
names(hotels)
```

```
[1] "country"          "city_actual"      "rating_count"
[4] "center1label"     "center2label"     "neighbourhood"
[7] "price"            "city"             "stars"
[10] "ratingta"         "ratingta_count"   "scarce_room"
[13] "hotel_id"         "offer"            "offer_cat"
[16] "year"             "month"            "weekend"
[19] "holiday"          "distance"         "distance_alter"
[22] "accommodation_type" "nnights"          "rating"
```

What do the numbers look like?

```
summary(hotels$price)
```

Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
27.0	83.0	109.5	131.4	146.0	1012.0

The `$` picks one column out of the table, so `hotels$price` is the price column on its own.

💡 Read that summary properly

Median €109.50, mean €131.40. The mean sits well above the median, so the distribution is **right-skewed** — a few very expensive hotels pull the average up. For prices, incomes and firm sizes this is the norm, and the median is the more honest summary. Also note the maximum: **€1012**. Hold that thought.

7 Step 6: your first chart

```
ggplot(
  data = hotels,
  mapping = aes(x = distance, y = price)
) +
  geom_point(alpha = 0.5) +
  theme_classic()
```

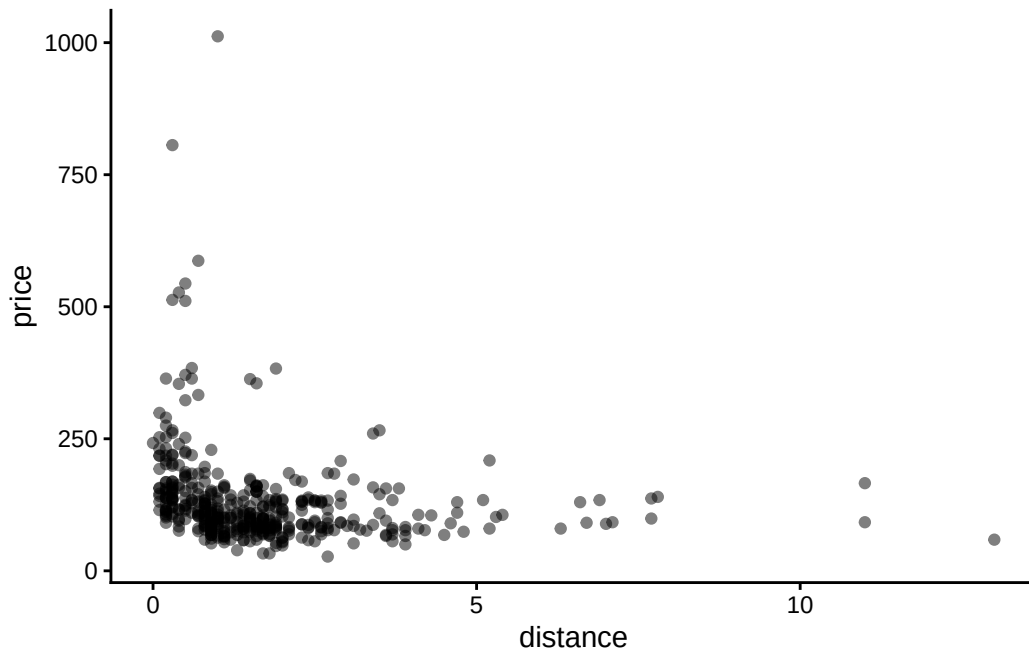


Figure 1: Price against distance from the city centre, 428 Vienna hotels.

Three things are going on, and they are the same three things in every `ggplot`:

1. `data =` — which table
2. `mapping = aes(...)` — which column goes on which axis
3. `geom_*()` — how to draw it

`alpha = 0.5` makes points semi-transparent so we can see where they overlap.

`theme_classic()` is one line making the plot look more decent.

7.1 Interrogating the chart

The headline is unsurprising: closer to the centre, more expensive. The interesting part is everything else.

Who is that hotel at €1012?

```
hotels |>
  filter(price > 500) |>
  select(price, stars, distance, accommodation_type, neighbourhood) |>
  flextable() |>
  theme_vanilla()
```

price	stars	distance	accom- moda- tion_type	neigh- bourhood
513	5.0	0.3	Hotel	Innere Stadt
806	5.0	0.3	Hotel	Innere Stadt
544	5.0	0.5	Hotel	Innere Stadt
511	3.5	0.5	Apartment	Innere Stadt
527	5.0	0.4	Hotel	Kaerntner Strasse
587	5.0	0.7	Hotel	Leopold- stadt
1,012	3.0	1.0	Hotel	Neubau

A **three-star** hotel at €1012, when the most expensive five-stars are around €800. That is almost certainly a data-entry error, and we found it by looking at a picture rather than by running a test.

And who is 13 km out?

```
hotels |>
  filter(distance > 6) |>
  select(city_actual, price, distance, stars) |>
  flextable() |>
  theme_vanilla()
```

city_actual	price	distance	stars
Fischamend	59	13.0	3
Schwechat	92	11.0	3
Schwechat	166	11.0	4
Schwechat	137	7.7	4
Schwechat	99	7.7	3
Schwechat	140	7.8	4
Schwechat	80	6.3	3
Schwechat	91	6.7	3
Vienna	134	6.9	3
Vienna	130	6.6	3
Voesendorf	89	7.0	4
Voesendorf	92	7.1	4

Twelve properties sit more than 6 km out — and for **ten** of them the `city_actual` column does not say Vienna. Schwechat is the airport town, Fischamend and Voesendorf are neighbours. “Distance to the city centre” means something quite different for an airport hotel, so those ten should come out of most analyses.

The other two genuinely are in Vienna, just far from the middle. They stay.

💡 Filter on the reason, not on the symptom

It would be easy to write `filter(distance < 6)` and move on. That would also throw away two perfectly good Vienna hotels, and it would silently break the moment someone adds a hotel 6.5 km out in the 22nd district.

The thing we actually object to is that ten properties are **not in the city**, so that is what we should filter on:

```
hotels |>
  filter(city_actual != "Vienna") |>
  nrow()
```

[1] 10

! This is the actual job

Nobody told us the data had these problems. We found them in the first five minutes by plotting it and asking who the odd points were. Do this every time, before you model anything.

8 Step 7: adding a second variable

Colour by star rating:

```
ggplot(  
  data = hotels,  
  mapping = aes(x = distance, y = price, colour = factor(stars))  
) +  
  geom_point(alpha = 0.7) +  
  labs(  
    x = "Distance to city centre (km)",  
    y = "Price per night (EUR)",  
    colour = "Stars",  
    title = "Central hotels cost more - but they are also fancier"  
  ) +  
  theme_classic()
```

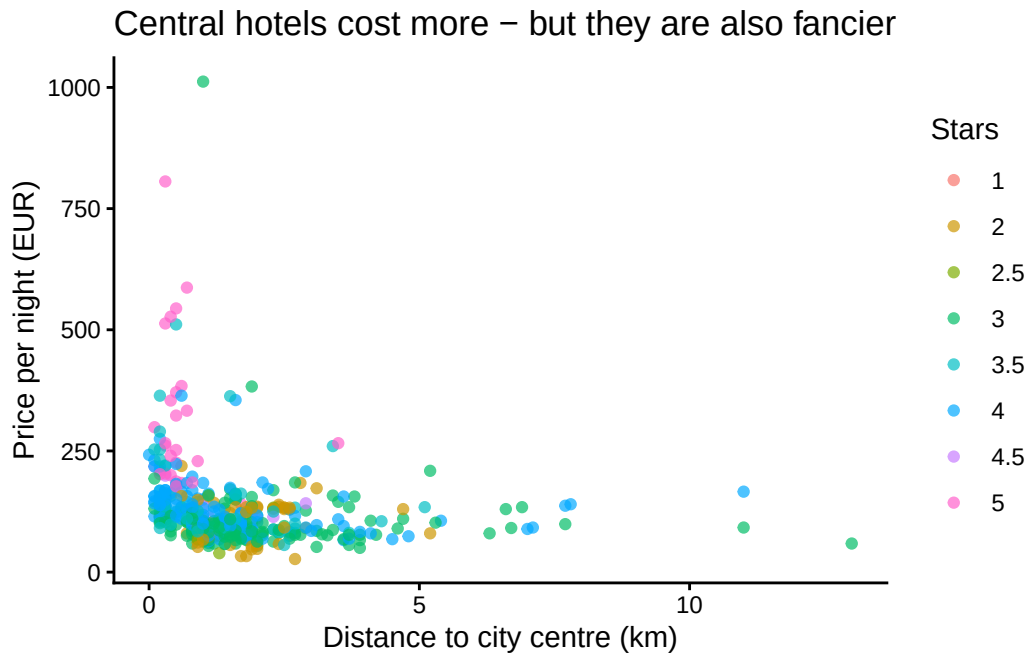


Figure 2: The same data, coloured by star rating.

Look at the colours near the left edge. The expensive central hotels are also the high-star ones.

i The question for next week

Is the centre expensive **because** it is central, or because that is where the good hotels happen to be?

The chart cannot tell us. Session 2 gets us most of the way, and session 8 finishes the job.

9 Step 8: render

Click **Render** (or Ctrl/Cmd + Shift + K).

Quarto runs every chunk from top to bottom in a **fresh R session**, and writes an HTML file next to your `.qmd`.

⚠ Why rendering fails

Rendering starts from nothing. If your code only works because you typed something into the console an hour ago, it will fail here.

This is a feature. It is the machine checking that your analysis actually reproduces. It is also exactly what the automated check on your reports will do.

10 Step 9: useful chunk options

Options go at the top of a chunk, after `#|`:

Option	Effect
<code># echo: false</code>	Run the code, hide it — show only the output
<code># eval: false</code>	Show the code, do not run it
<code># warning: false</code>	Suppress warning messages
<code># fig-cap: "..."</code>	Add a caption to a figure
<code># label: fig-name</code>	Name the figure so you can refer to it

Example — a chart with no code shown:

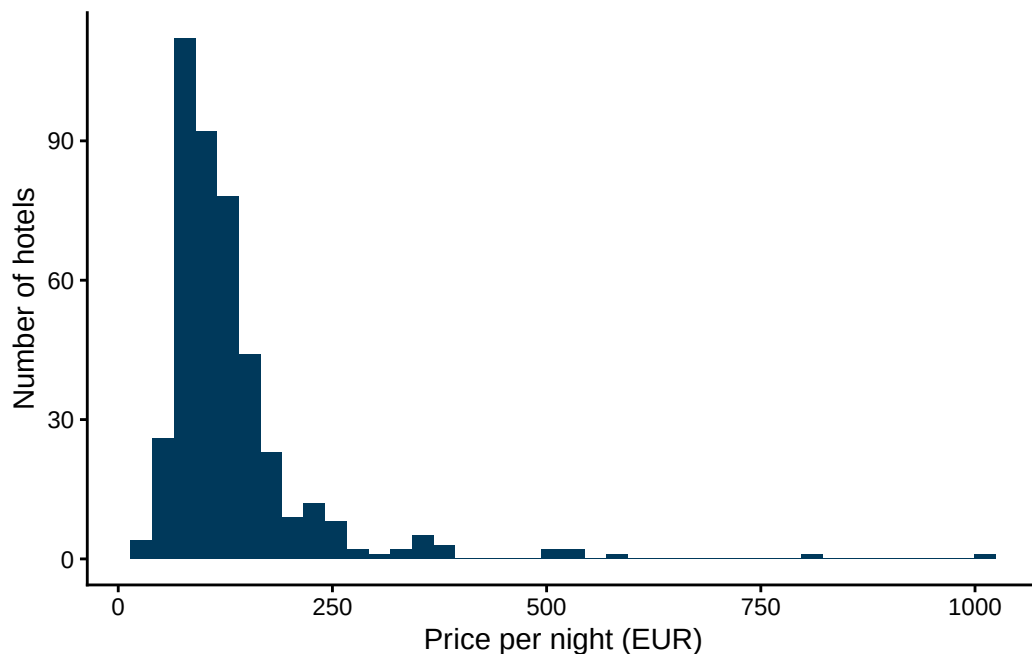


Figure 3: Where the prices sit.

 That long tail to the right is a real problem, and it has a standard fix

Almost every hotel sits on the left. A handful stretch the axis out to €1012, and they squash everybody else into a narrow band — the same reason the scatter earlier looked like a dense clump with a few strays.

Prices behave like this almost everywhere: a hotel can be ten times the average, but never a tenth of it. The usual fix is to plot price on a **logarithmic** scale, which spreads the crowded part out and pulls the tail in.

We are not doing it today, deliberately. On a log scale the €1012 hotel stops looking like a mistake and starts looking like an ordinary expensive hotel — and finding that mistake was the whole point of this session. **Session 5** picks this up properly by discussing the cases where you must transform your data using logarithms.

11 Recap

You now have all of it:

- A **project** so paths work everywhere
- A **.qmd file** mixing text and code
- **Chunks** that run R
- A **rendered HTML report** you could send to someone
- And two data problems nobody warned you about

 If you remember three things

1. Write in the Source pane by default, not the console
2. Never use absolute file paths
3. Plot the data before you believe anything about it