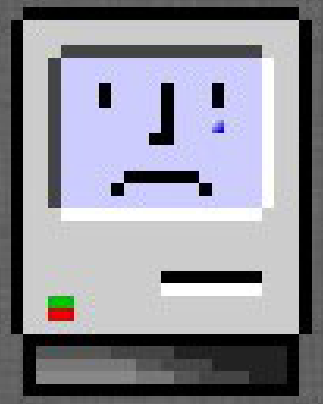
02 — Getting Oriented

Kieran Healy

January 23, 2024

Motivation

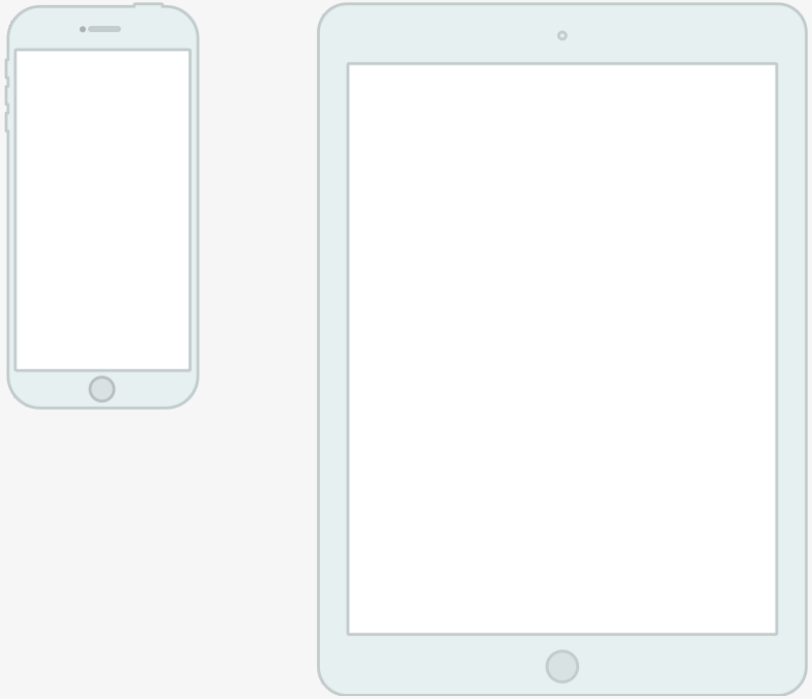
Technical computing is often frustrating



What is this?

Two Revolutions in Computing

What everyday computing is now



Touch-based user interface

Foregrounds a single application

Dislikes multi-tasking

Hides the *file system*

“Laundry Pile” user model of where things are stored

Where technical computing lives

Windows and pointers.

Multi-tasking, multiple windows.

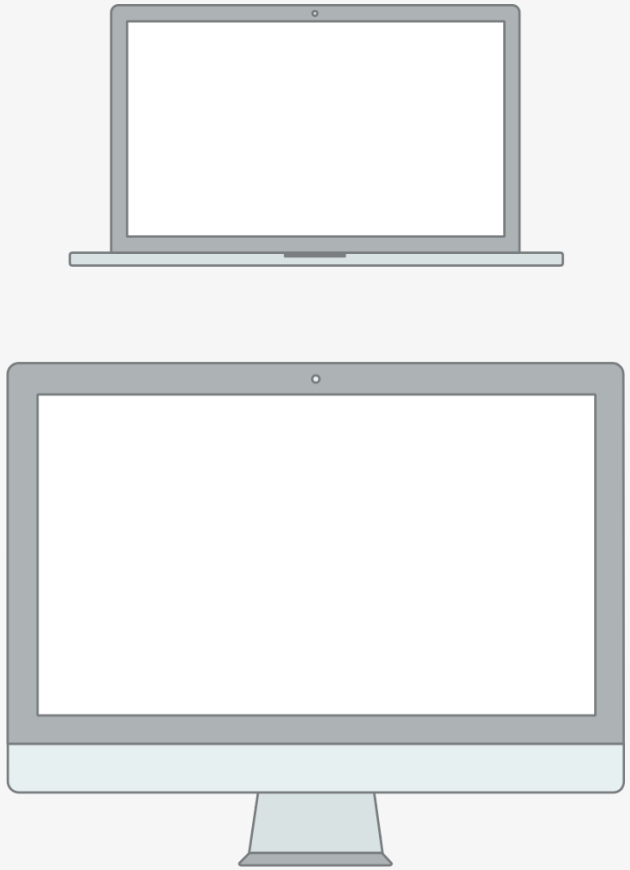
Exposes and leverages the *file system*.

Many specialized tools in concert.

Underneath, it's the 1970s, *UNIX*, and the command-line.

Cabinets, drawers, and files model of where things are stored

Where technical computing lives



This toolset is by now really good!

Free! Open! Powerful!

Friendly communities! Lots of information! Many resources!

But: grounded in a UI paradigm that is increasingly far away from the everyday use of computing devices

So why do we use this stuff?


CONTROL

“Office” vs “Engineering” approaches

What is “real” in your project?

What is the final output?

How is it produced?

How are changes managed?

Different Answers

Office model

Formatted documents are real.

Intermediate outputs are cut and pasted into documents.

Changes are tracked inside files.

Final output is often in the same format you've been working in, e.g. a Word file, or a PDF.

Engineering model

Plain-text files are real.

Intermediate outputs are produced via code, often inside documents.

Changes are tracked outside files, at the level of a project.

Final outputs are assembled programmatically and converted to some desired format.

Different strengths and weaknesses

Office model

Everyone knows Word, Excel, or Google Docs.

“Track changes” is powerful and easy.

Hm, I can’t remember how I made this figure

Where did this table of results come from?

`Paper_edits_FINAL_kh-1.docx`

Engineering model

Plain text is highly portable.

Push button, recreate analysis.

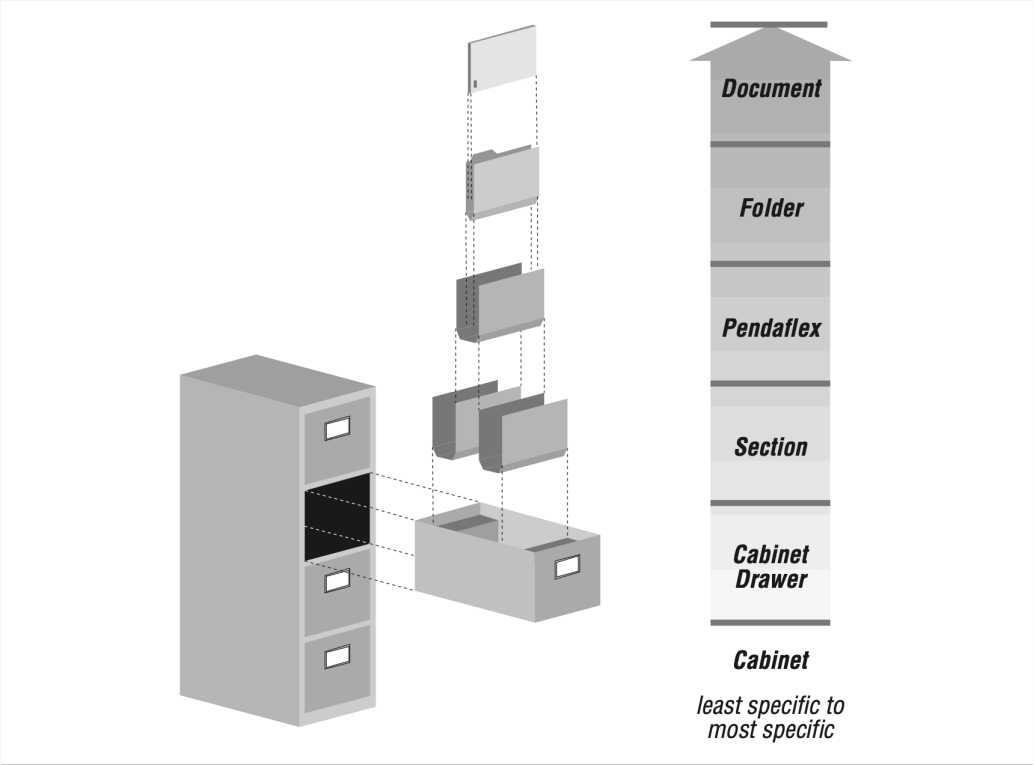
JFC Why can’t I do this simple thing?

`Object of type 'closure' is not subsettable`

Each approach generates solutions to its own problems

The File System

The traditional analog

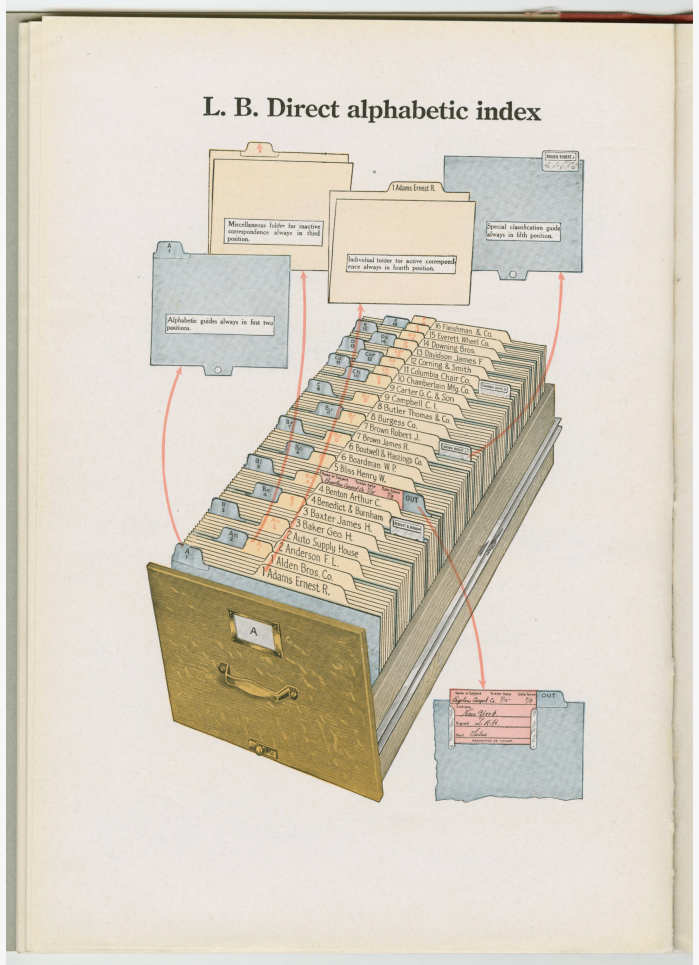


The problem is, you probably have never have actually used one of these!

Illustration: Shelley Powers et al. *Unix Power Tools*, 3rd ed. (Sebastopol, CA: O'Reilly Media, 2002), 21.

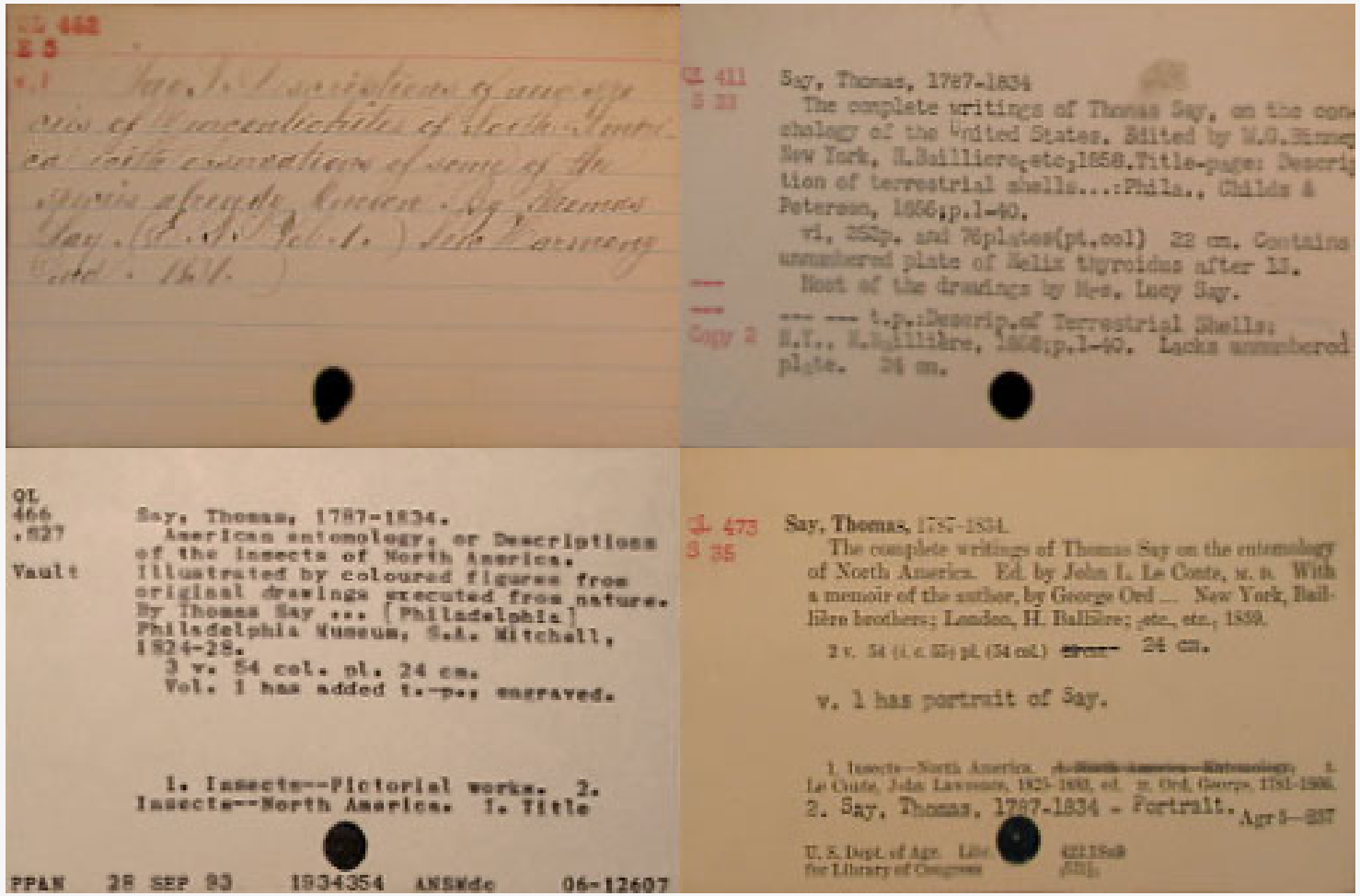
The file cabinet!

The file cabinet!

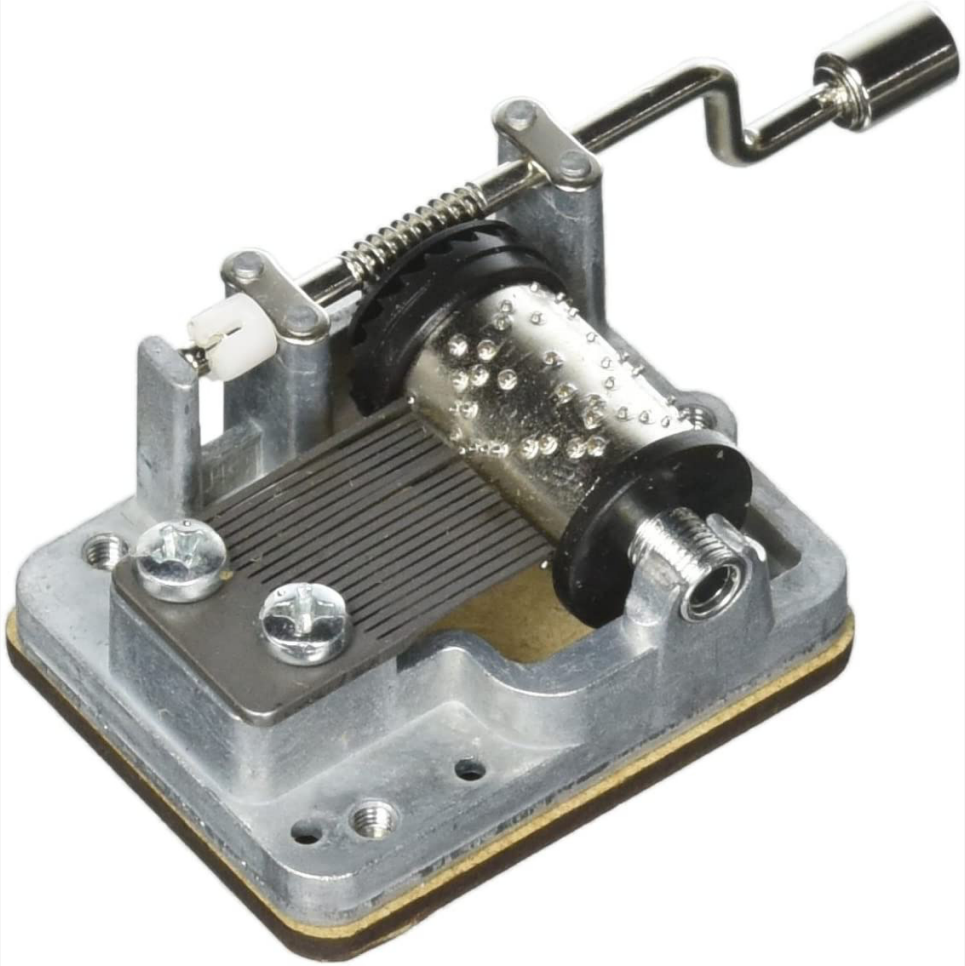


Index cards

Index cards


Say, Thomas, 1787-1834
The complete writings of Thomas Say, on the conchology of the United States. Edited by W.G. Binney
New York, H.Bailliere, etc, 1858. Title-page: Description of terrestrial shells...: Phila., Childs & Peterson, 1856; p.1-40.
Most of the drawings by Mrs. Lucy Say.
Say, Thomas, 1787-1834.
American entomology, or Descriptions of the insects of North America.
Illustrated by coloured figures from original drawings executed from nature.
By Thomas Say ... [Philadelphia]
Philadelphia Museum, S.A. Mitchell, 1824-28.
3 v. 54 col. pl. 24 cm.
Vol. 1 has added t.-p., engraved.
1. Insects--Pictorial works. 2. Insects--North America. I. Title
Say, Thomas, 1787-1834.
The complete writings of Thomas Say on the entomology of North America. Ed. by John L. Le Conte, M. D. With a memoir of the author, by George Ord ... New York, Bailliere brothers; London, H. Bailliere; etc., etc., 1859.
v. 1 has portrait of Say.
2. Say, Thomas, 1787-1834 - Portrait.

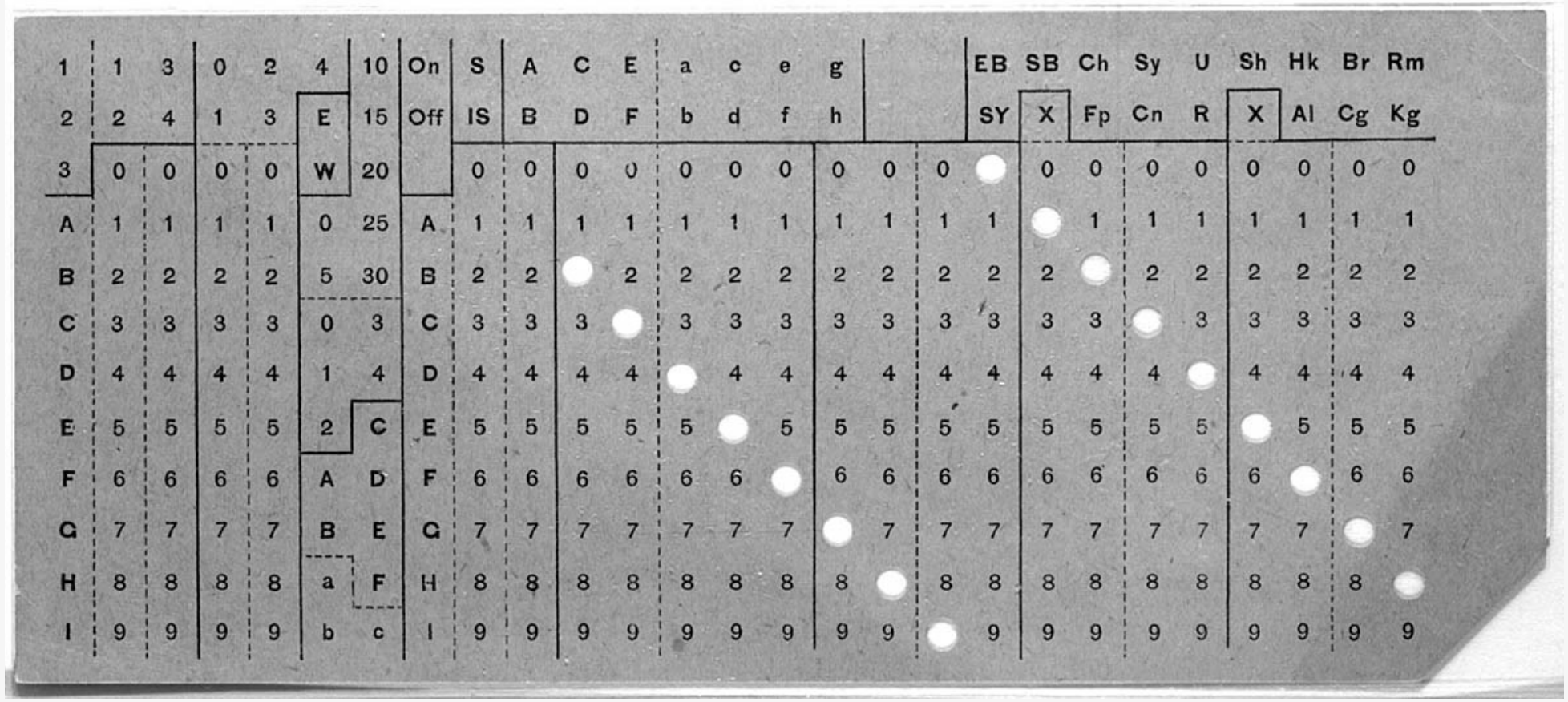
Automating information processing



Automating information processing

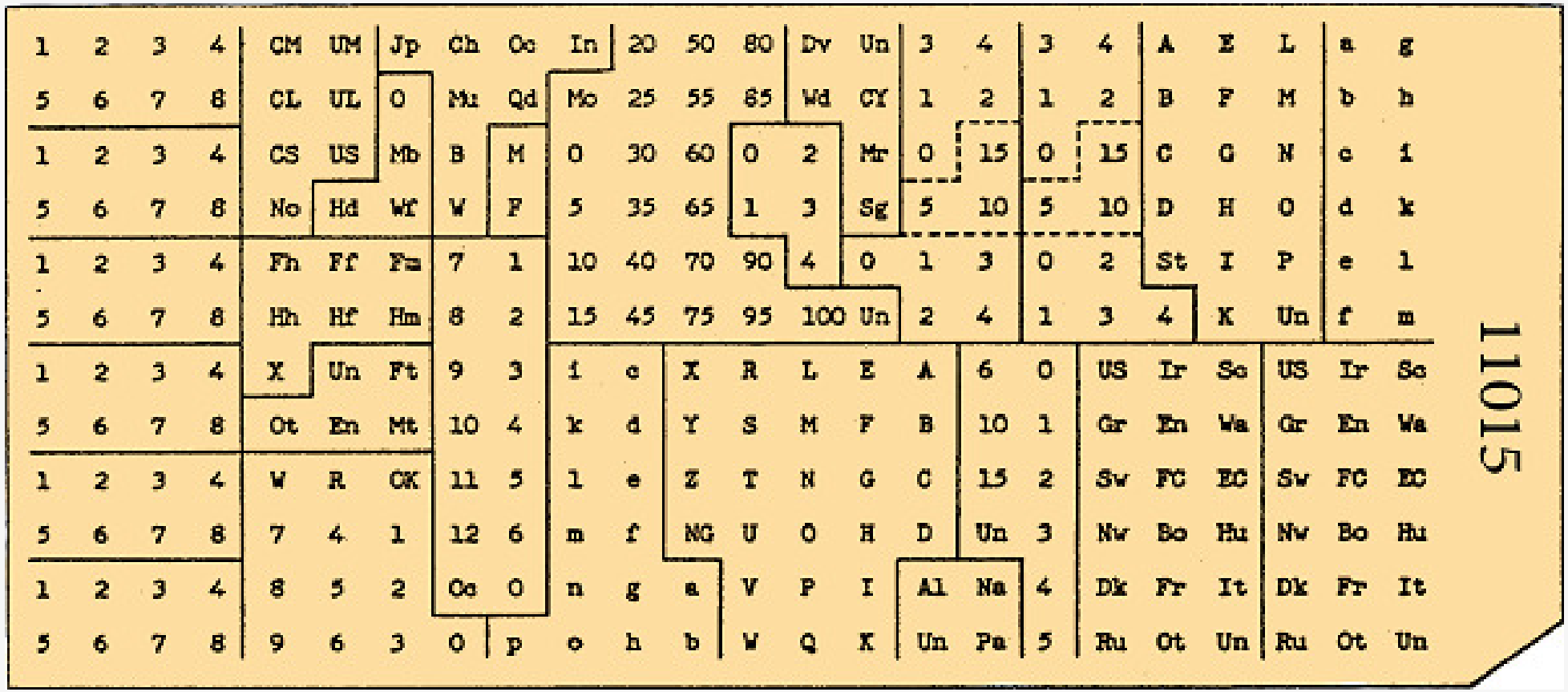
Automating information processing

Hollerith machines

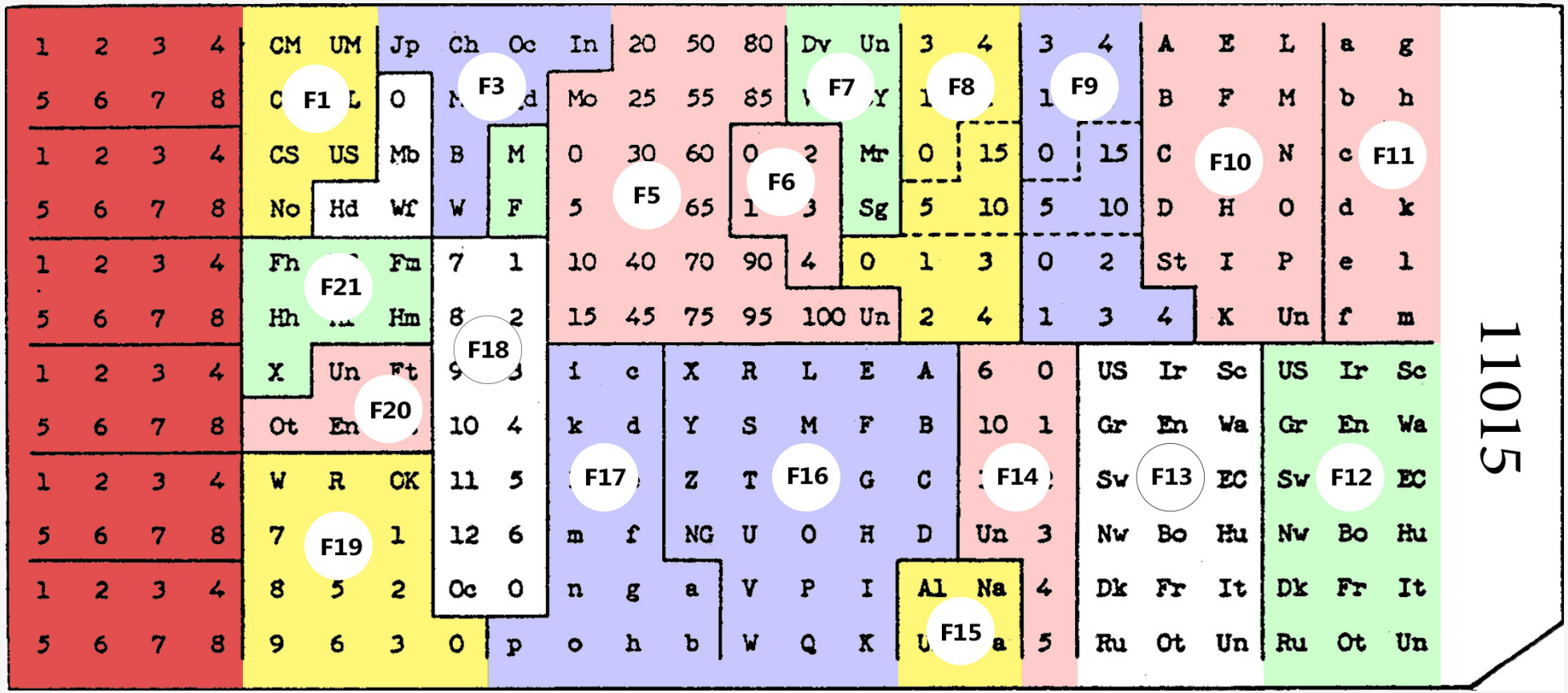


Hollerith Machines

Hollerith machines



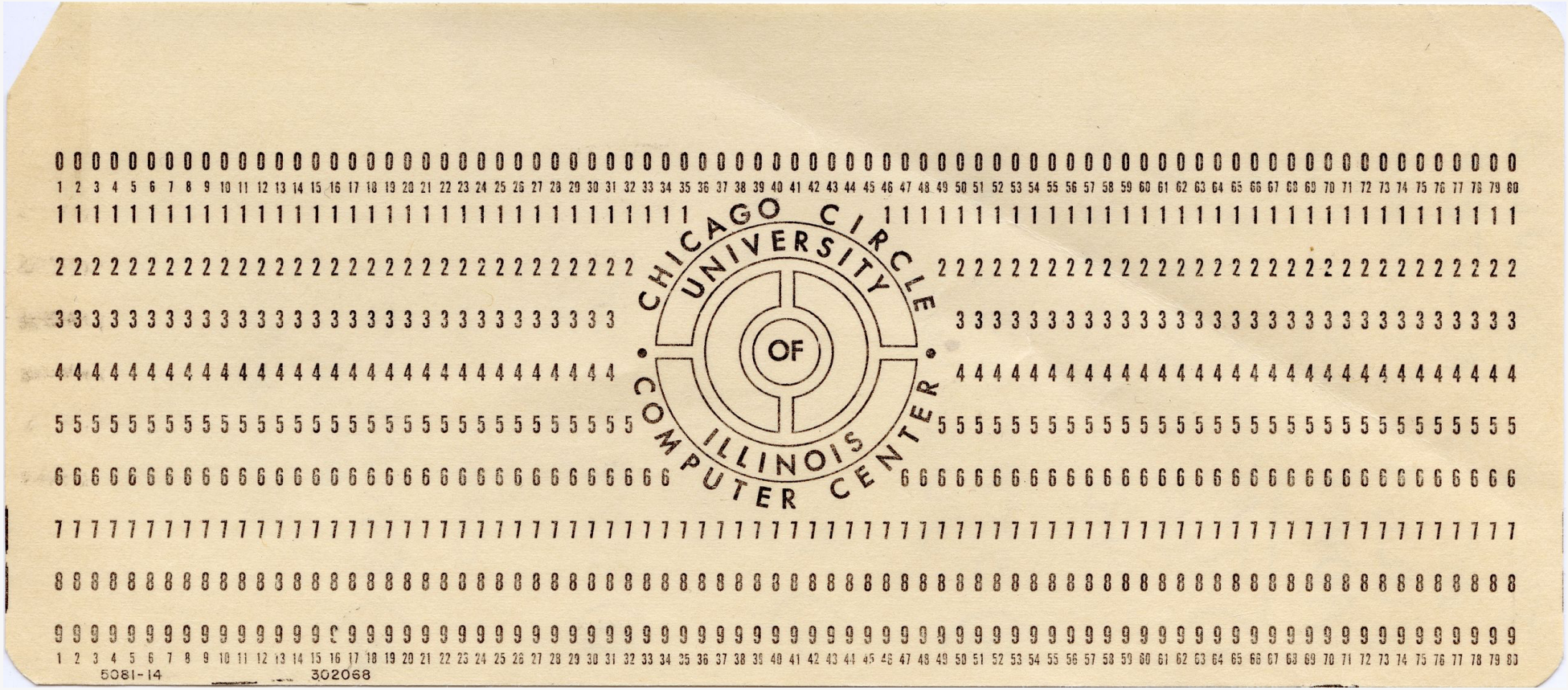
Hollerith machines



Hollerith Operators

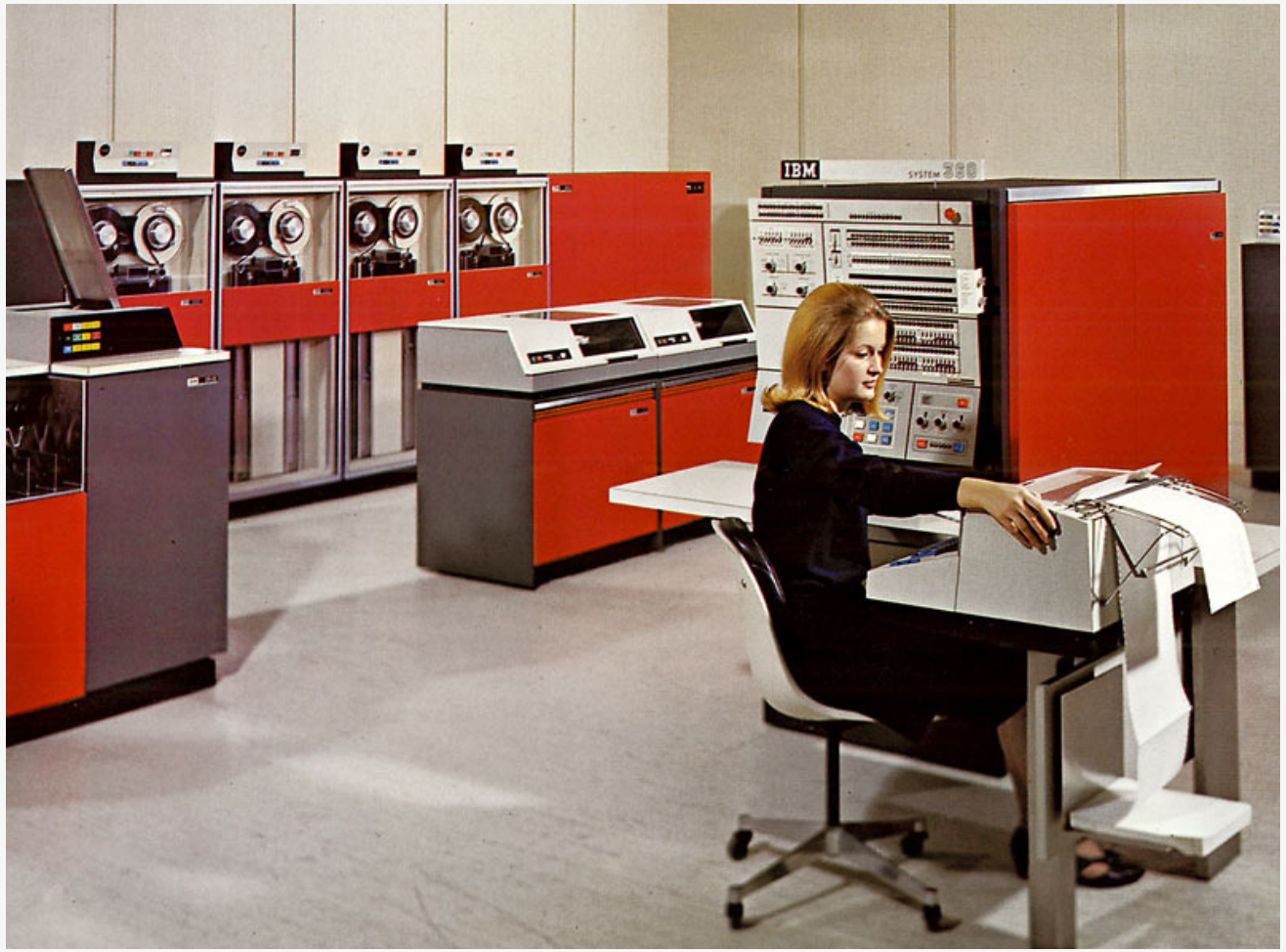
Hollerith Operators

IBM punch cards



IBM punch cards

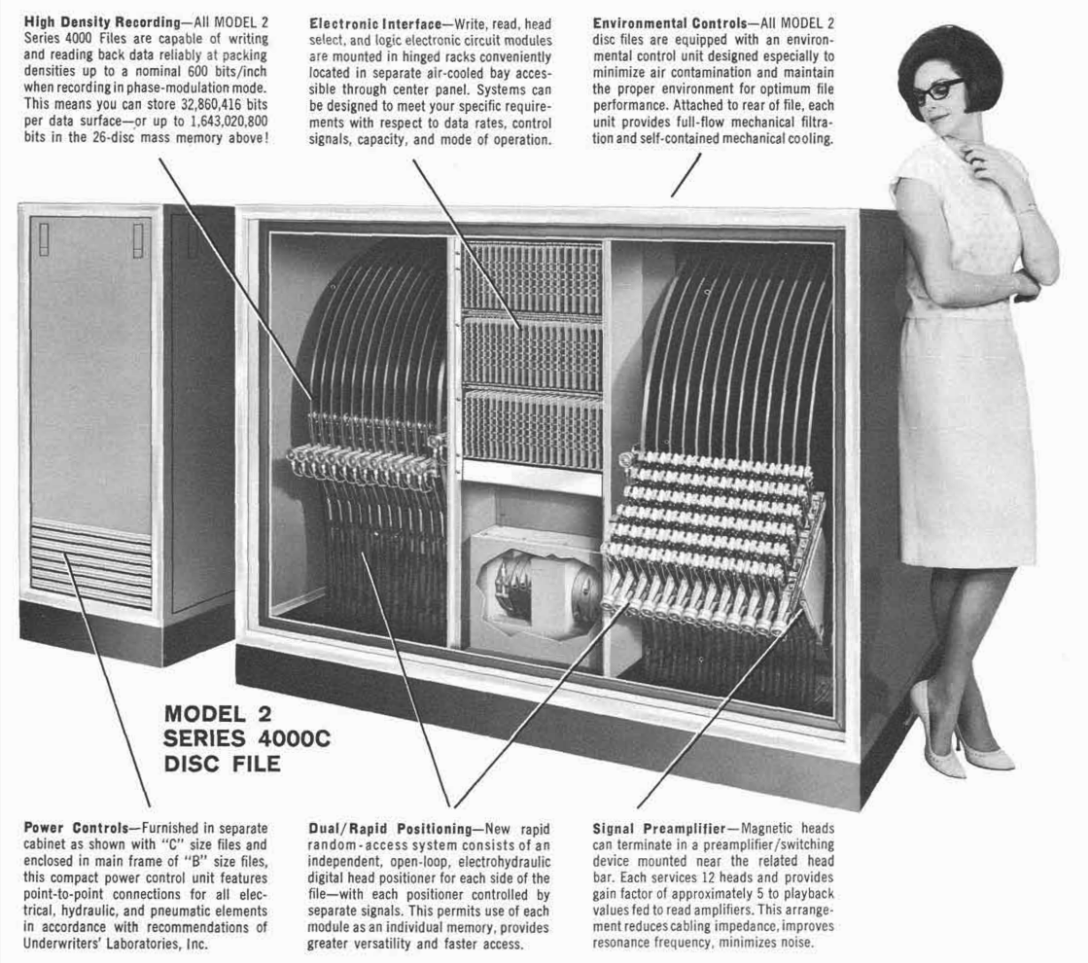
Big Iron



Storage


more of everything
you need and want
in a random-access
mass memory . . .
BRYANT Model-2 DISC FILES
SERIES 4000
Copyright 1965—Ex-Cell-O Corporation • All rights reserved

Storage


High Density Recording—All MODEL 2 Series 4000 Files are capable of writing and reading back data reliably at packing densities up to a nominal 600 bits/inch when recording in phase-modulation mode. This means you can store 32,860,416 bits per data surface—or up to 1,643,020,800 bits in the 26-disc mass memory above!
Electronic Interface—Write, read, head select, and logic electronic circuit modules are mounted in hinged racks conveniently located in separate air-cooled bay accessible through center panel. Systems can be designed to meet your specific requirements with respect to data rates, control signals, capacity, and mode of operation.
Environmental Controls—All MODEL 2 disc files are equipped with an environmental control unit designed especially to minimize air contamination and maintain the proper environment for optimum file performance. Attached to rear of file, each unit provides full-flow mechanical filtration and self-contained mechanical cooling.
MODEL 2
SERIES 4000C
DISC FILE
Power Controls—Furnished in separate cabinet as shown with "C" size files and enclosed in main frame of "B" size files, this compact power control unit features point-to-point connections for all electrical, hydraulic, and pneumatic elements in accordance with recommendations of Underwriters' Laboratories, Inc.
Dual/Rapid Positioning—New rapid random-access system consists of an independent, open-loop, electrohydraulic digital head positioner for each side of the file—with each positioner controlled by separate signals. This permits use of each module as an individual memory, provides greater versatility and faster access.
Signal Preamplifier—Magnetic heads can terminate in a preamplifier/switching device mounted near the related head bar. Each services 12 heads and provides gain factor of approximately 5 to playback values fed to read amplifiers. This arrangement reduces cabling impedance, improves resonance frequency, minimizes noise.

Input/Output

A late-model teletype (TTY) machine

Input/Output

The DEC VT-100 Terminal

Input/Output

Back to the file system

File system hierarchy

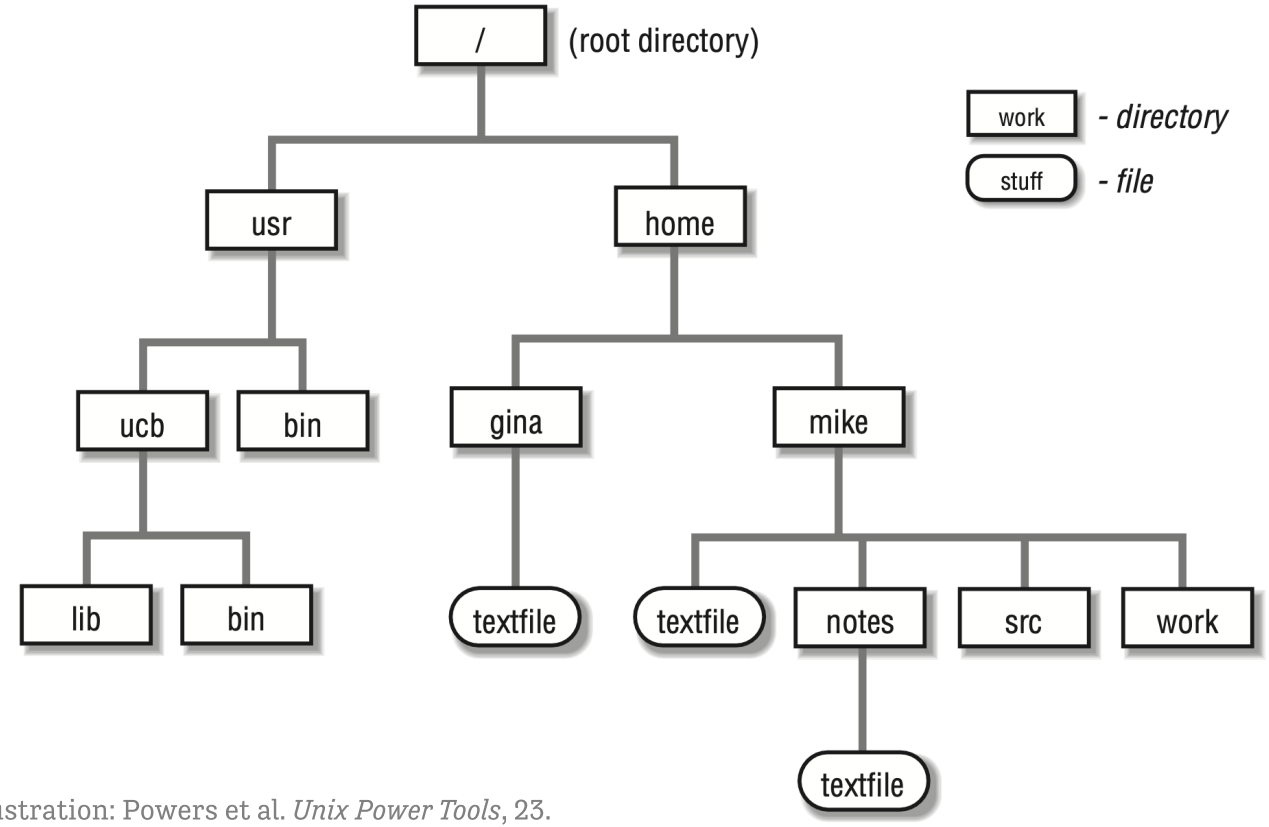


Illustration: Powers et al. *Unix Power Tools*, 23.

Stepping back

Your computer stores files and does stuff, or “runs commands”

Files are stored in a large hierarchy of folders

The Finder or Window Manager or File Manager is a visual metaphor for representing this hierarchy of files and for running commands on them. But you can also do these things via text-based commands delivered from a prompt, console, or “command line”.

Software like RStudio has a lot of these “old school” computing elements

Getting to know R and RStudio

We want to draw graphs reproducibly

R

RStudio®

Abstraction in software

Less

Easy things are awkward

Hard things are straightforward

Really hard things are possible

Abstraction in software

Less

Easy things are awkward

Hard things are straightforward

Really hard things are possible

More

Easy things are trivial

Hard things are awkward

Really hard things are impossible

Compare

D3

Grid

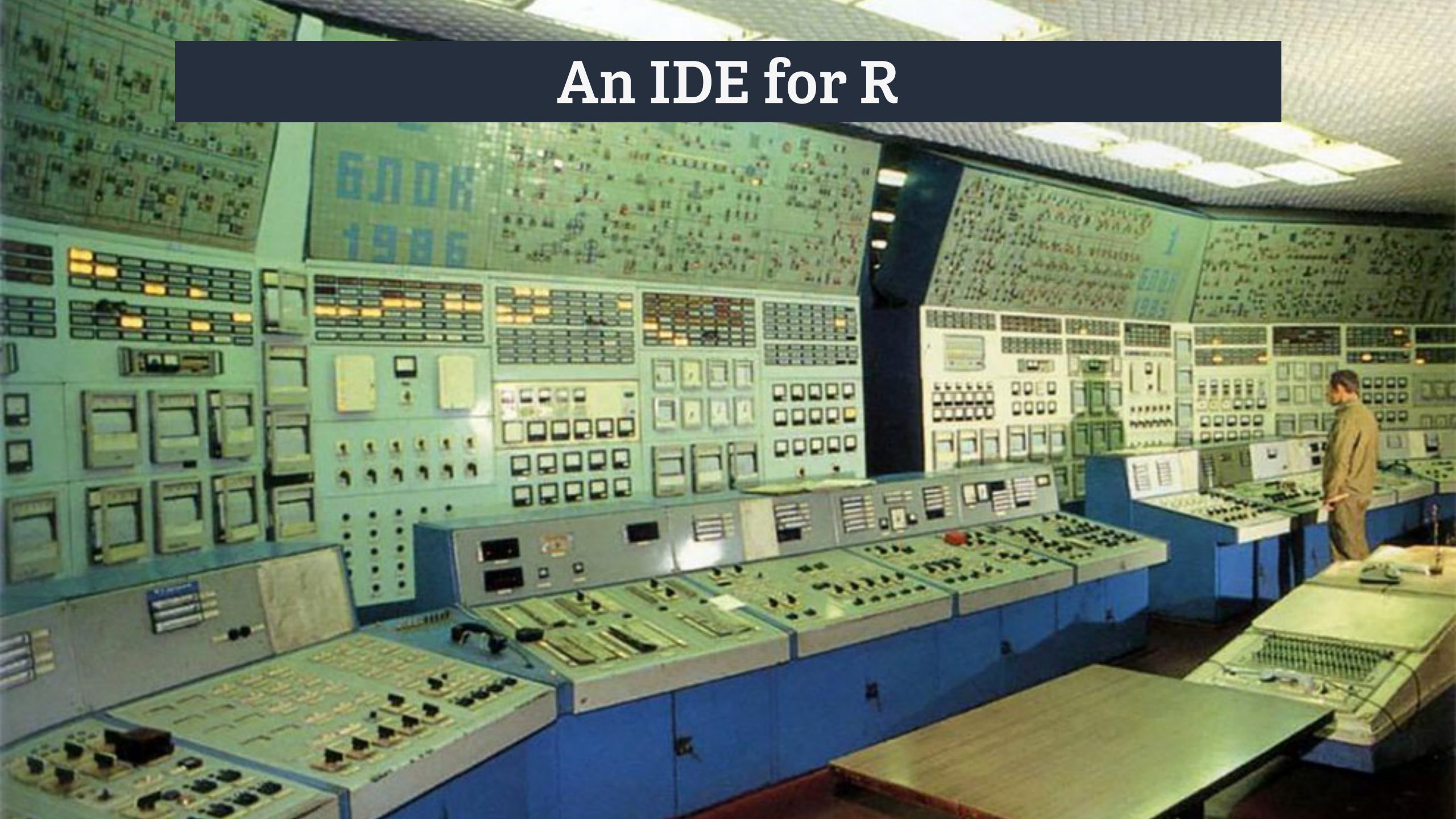
ggplot

Stata

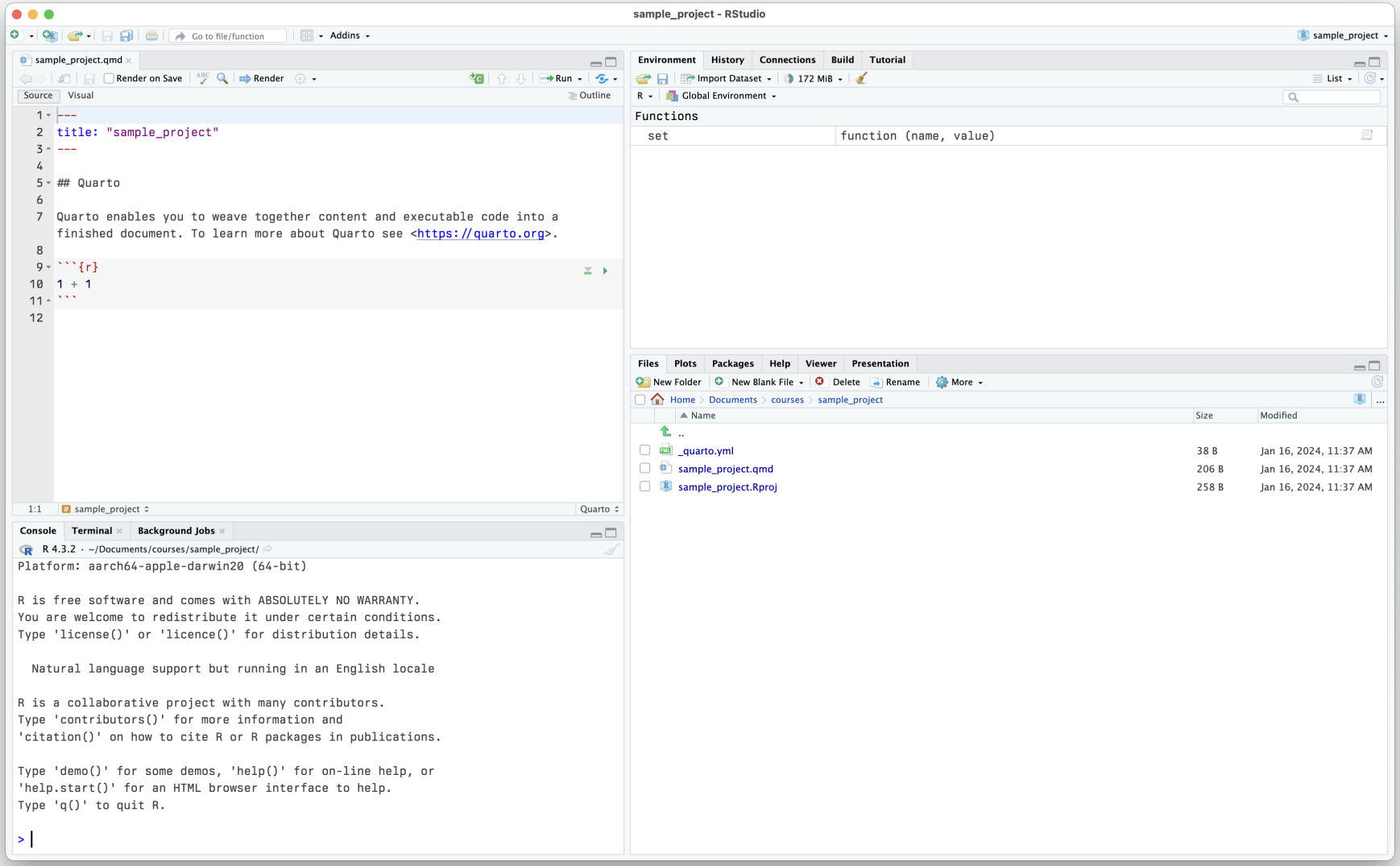
Excel

The RStudio IDE

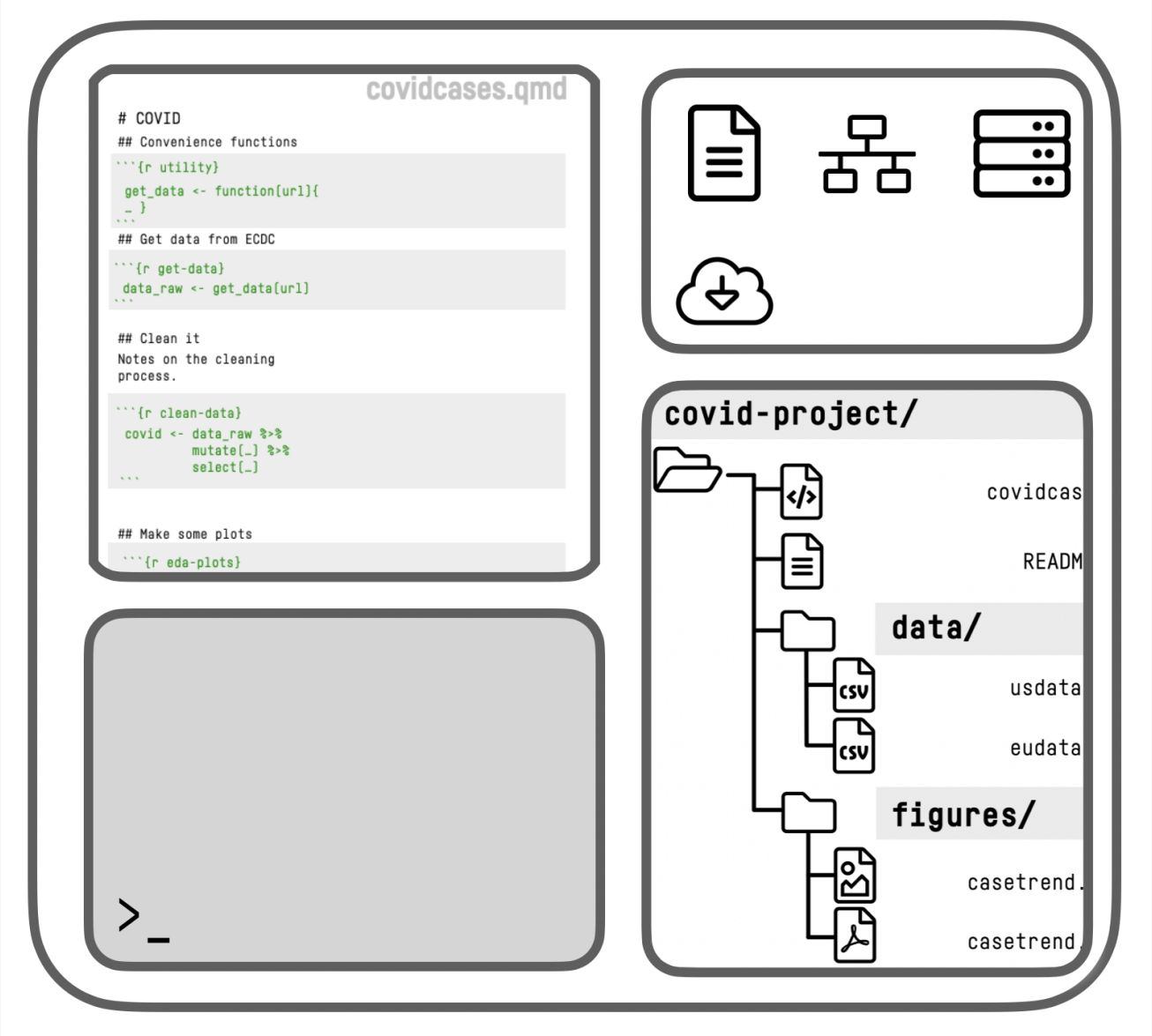
An IDE for R



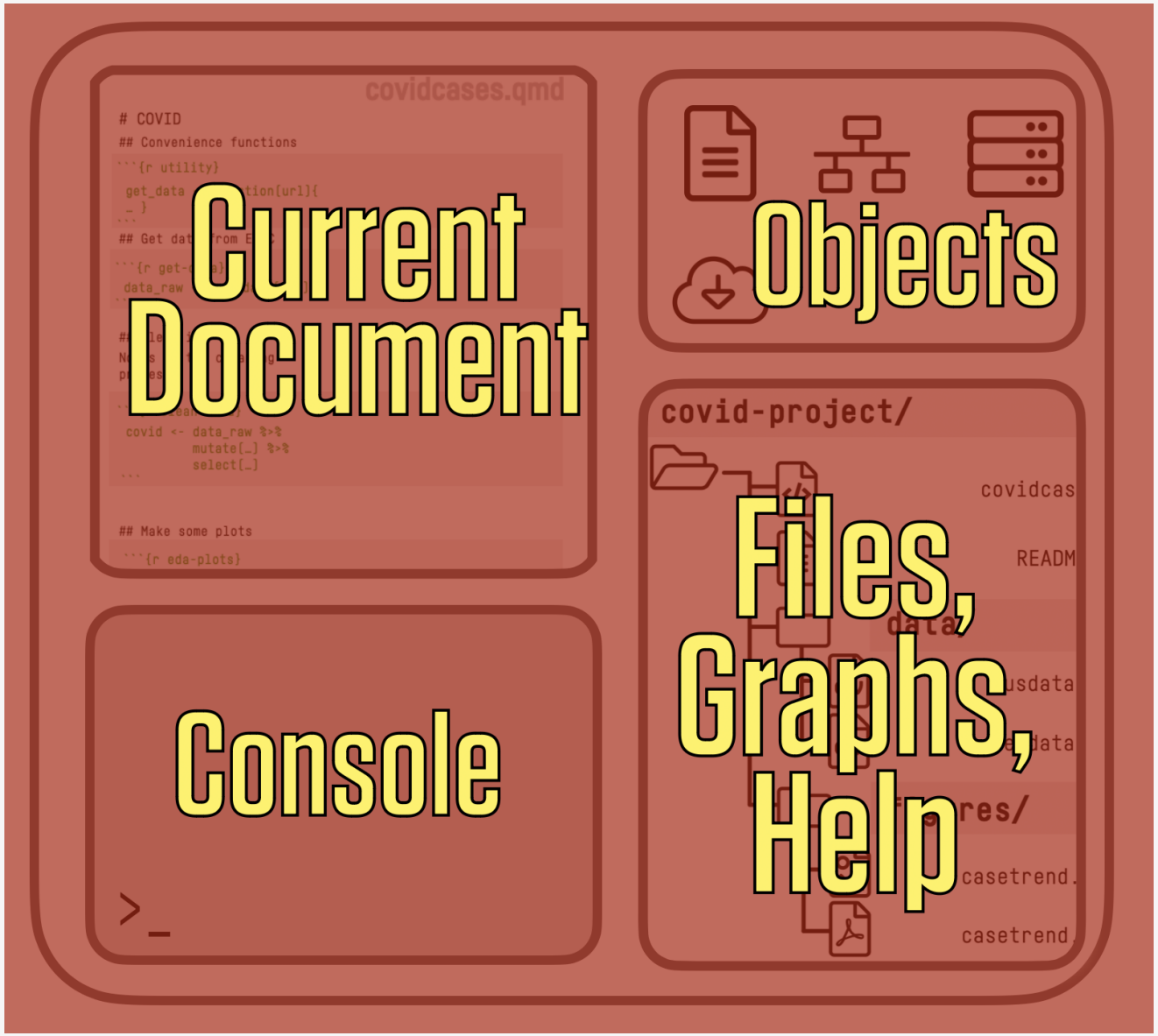
An IDE for Meals



RStudio at startup



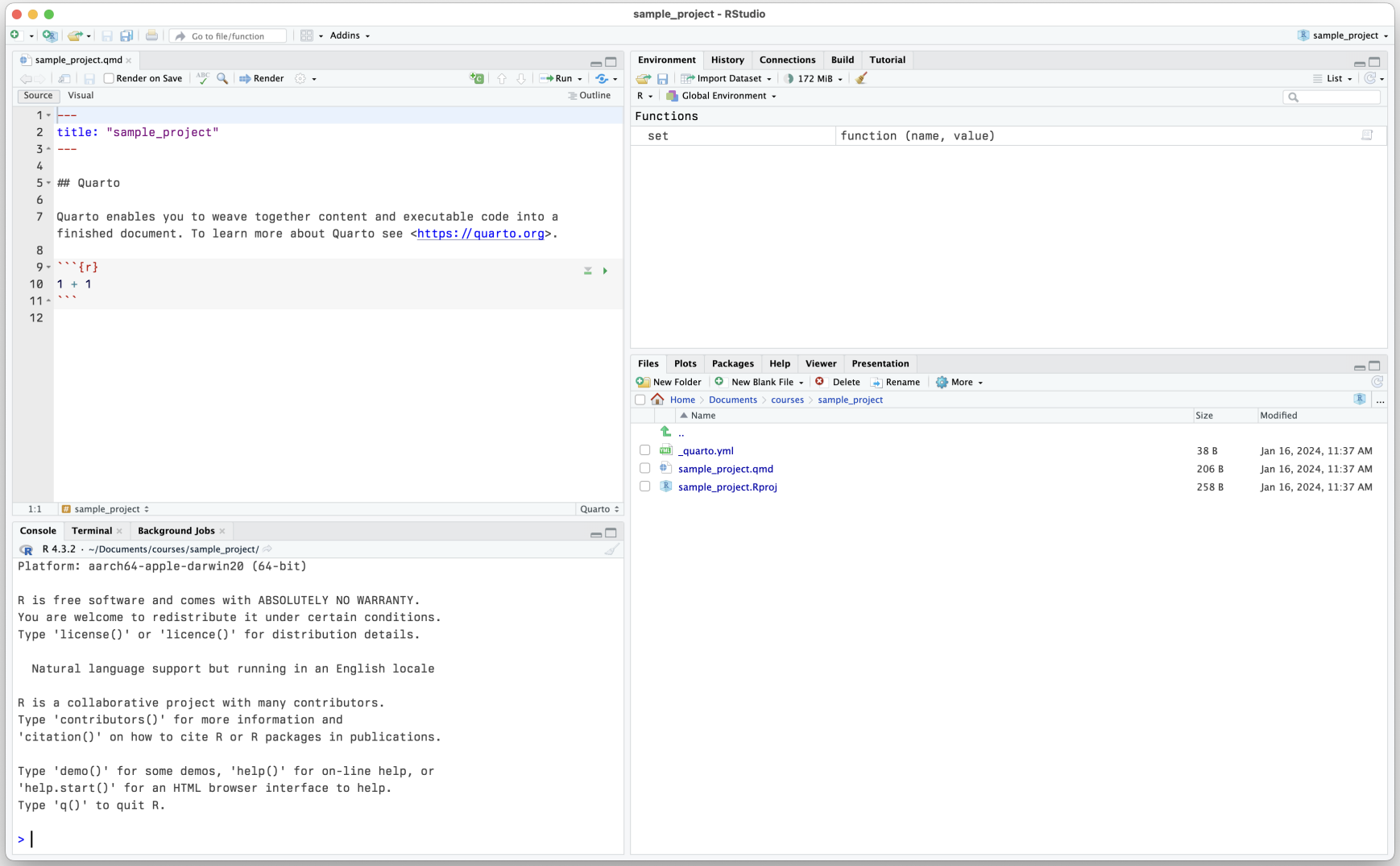
RStudio schematic overview



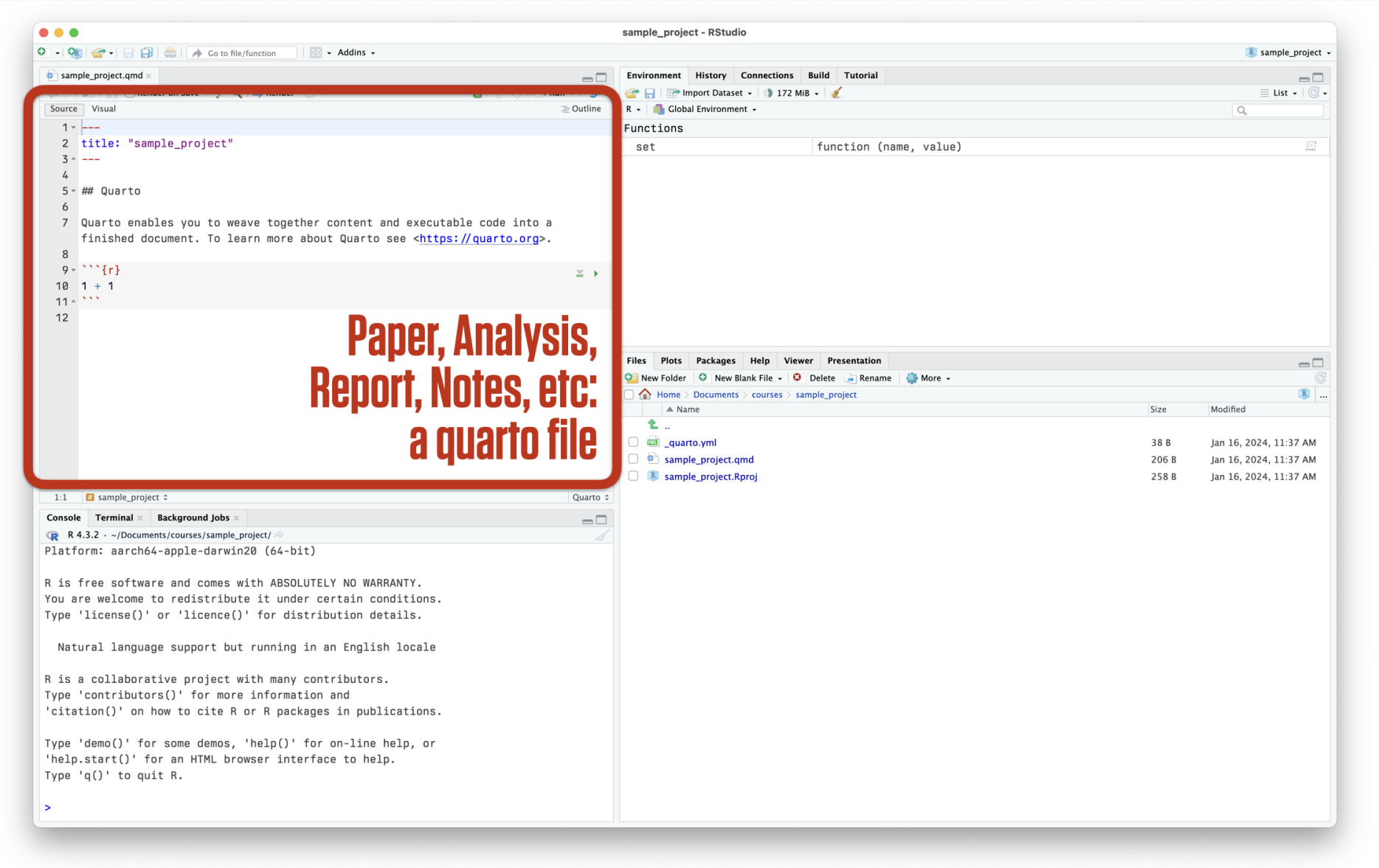
RStudio schematic overview

Think in terms of Data + Transformations, written out as code, rather than a series of point-and-click steps

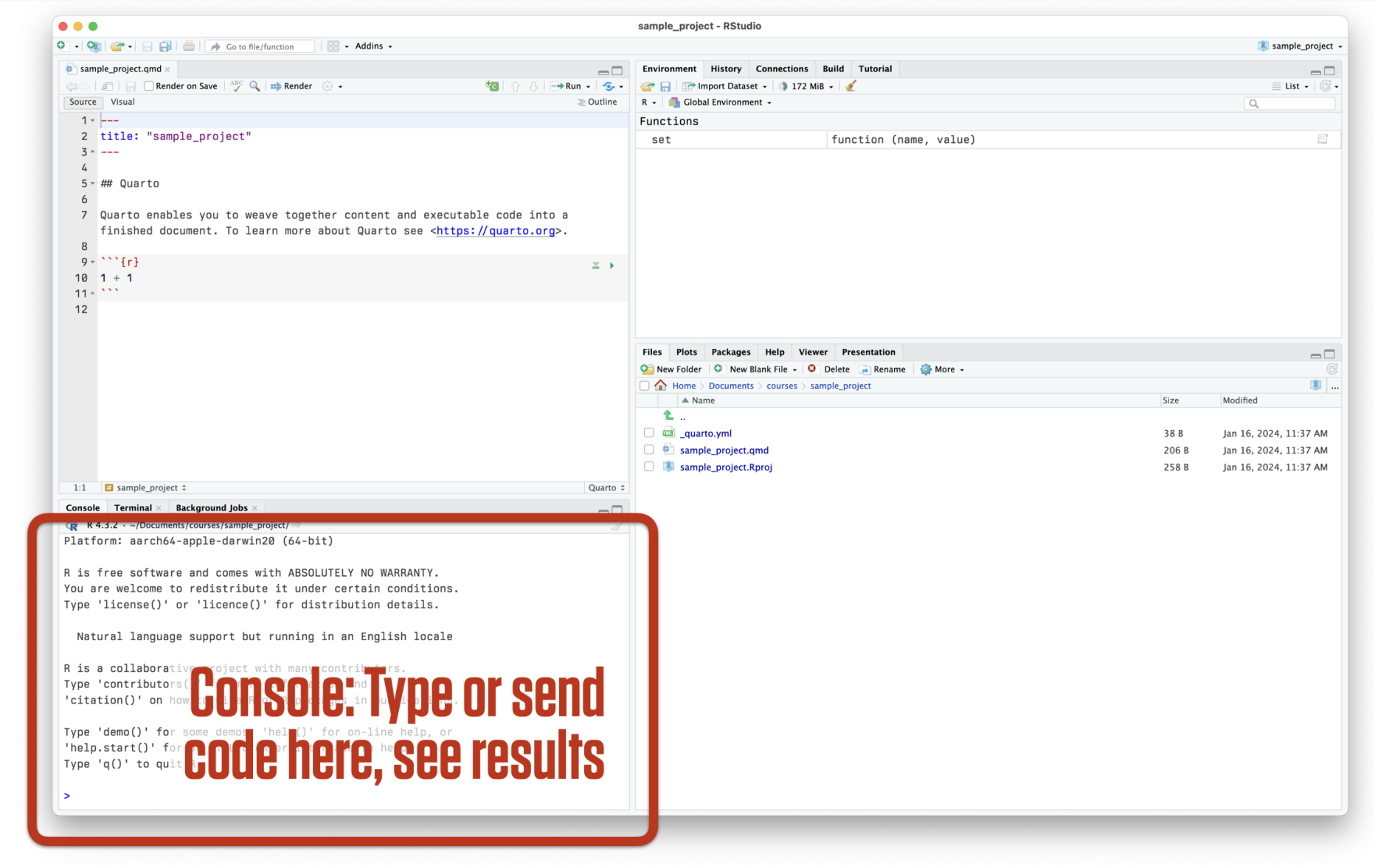
Our starting data + our code is what's "real" in our projects, not the final output or any intermediate objects



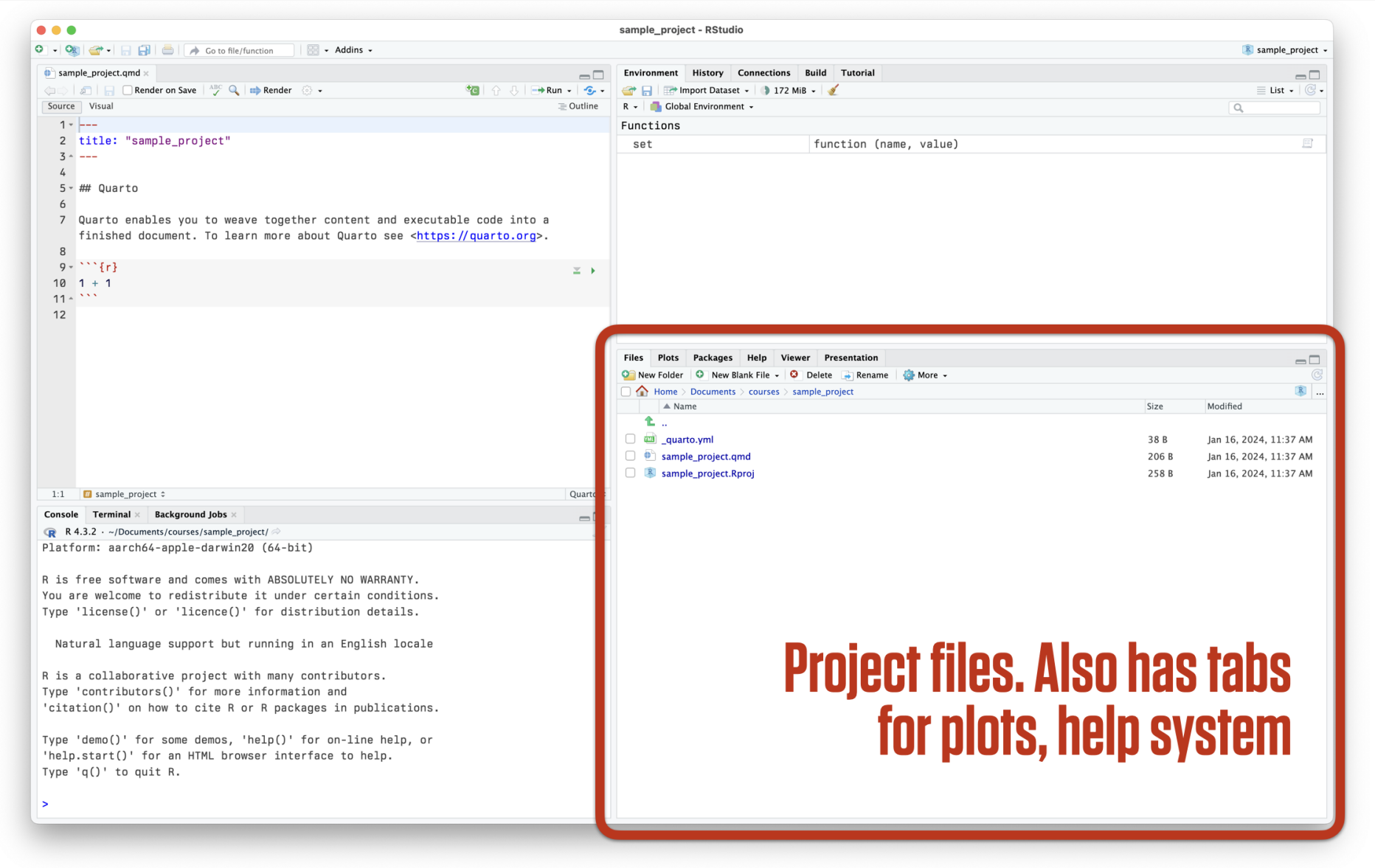
RStudio at startup



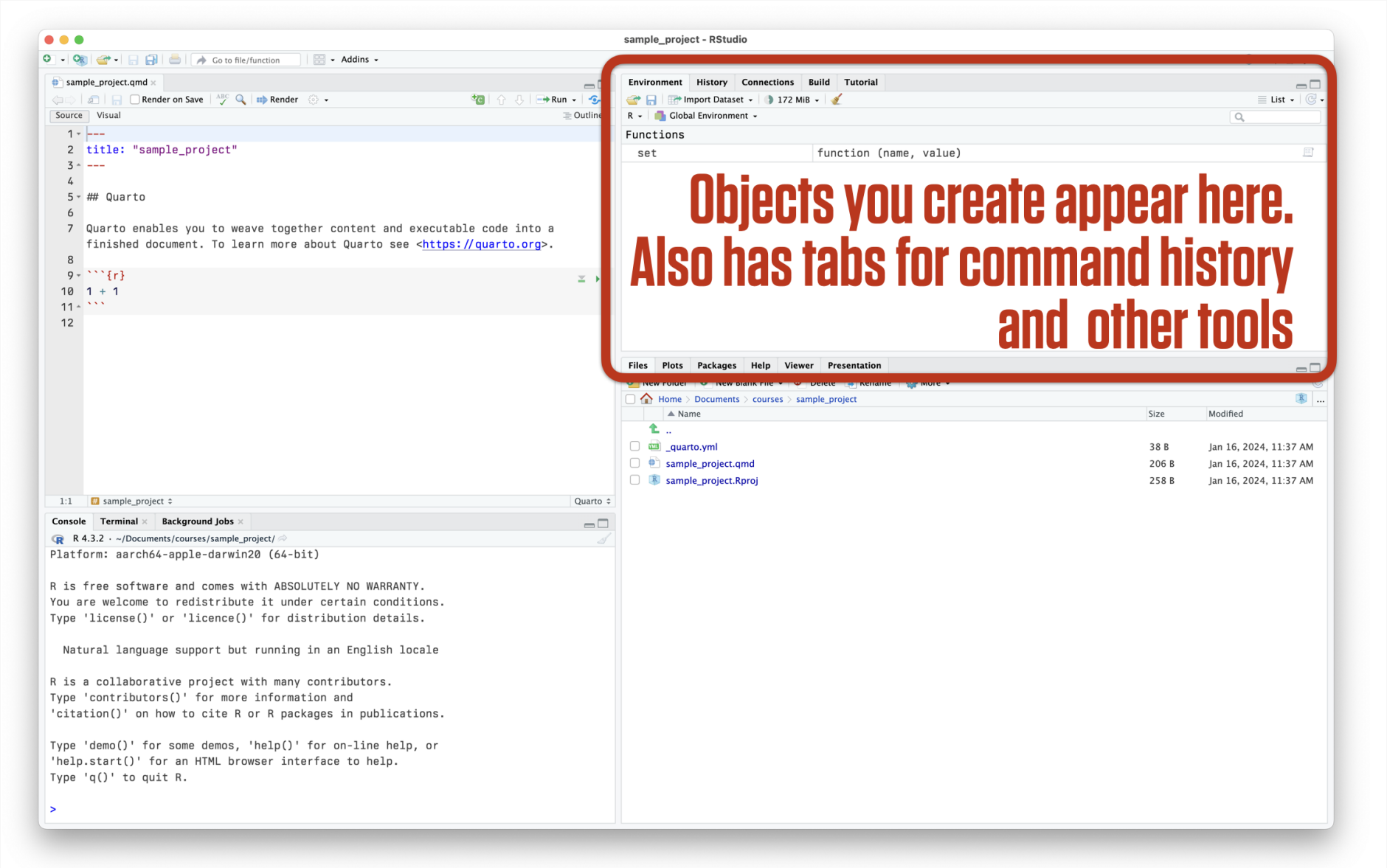
RStudio at startup



RStudio at startup



RStudio at startup



RStudio at startup

Use RMarkdown to produce and reproduce work

Where we want to end up

Covid Cases

Kieran Healy

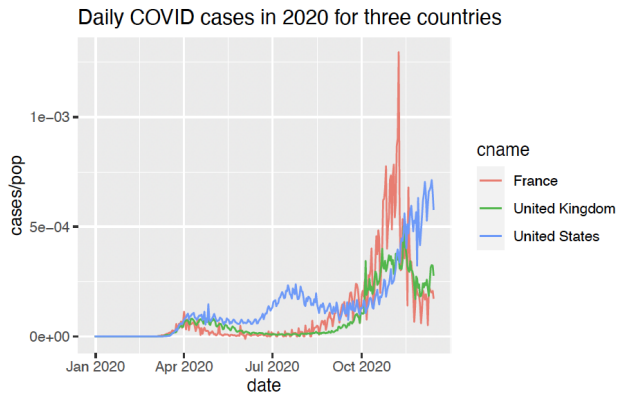
Introduction

We will look at some data from the `covdata` package.

cname	cases
France	2376852
United Kingdom	1849403
United States	16256754

A little graph

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.



PDF out

Where we want to end up

Covid Cases

AUTHOR
Kieran Healy

Introduction

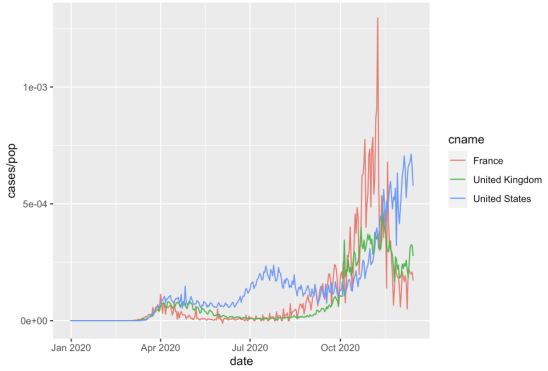
We will look at some data from the `covdata` package.

cname	cases
France	2376852
United Kingdom	1849403
United States	16256754

A little graph

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Daily COVID cases in 2020 for three countries



HTML out

Where we want to end up

Covid Cases

Kieran Healy

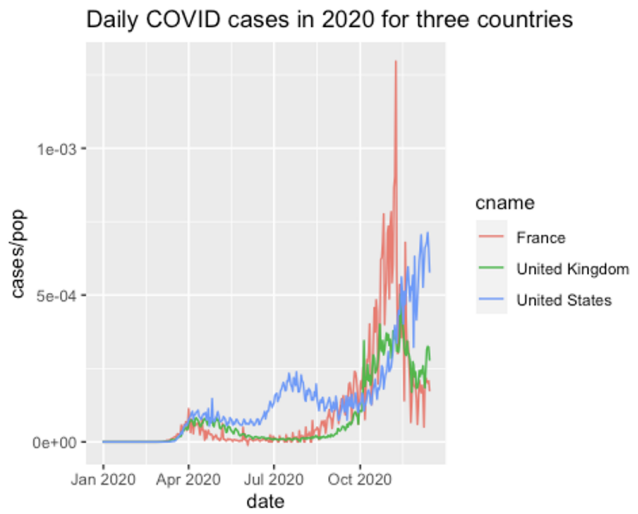
Introduction

We will look at some data from the `covdata` package.

cname	cases
France	2376852
United Kingdom	1849403
United States	16256754

A little graph

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.



Word out

How to get there?

```
# COVID                covidcases.R
# Get data from ECDC
# FIXME Write a fn to
# do this
 data_raw <- read_csv(url)

# Clean it
# Notes on the cleaning
# process.

 covid <- data_raw %>%
          mutate(…) %>%
          select(…)


# Make some plots
 covid %>%
      ggplot(…) +
      geom_line(…)
```

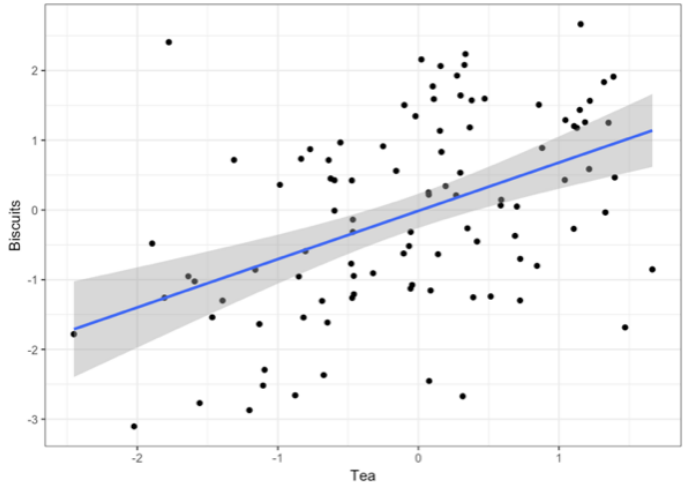
We could write an **R script** with some notes inside, using it to create some figures and tables, paste them into our document.

This will work, but we can do better.

We can make this ...

1. Lorem Ipsum

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do **eiusmod tempor** incididunt ut labore et dolore magna aliqua. Ut enimad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.



Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

... by writing this

Lorem Ipsum

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do *eiusmod tempor* incididunt ut labore et dolore magna aliqua. Ut enimad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.

```
library(ggplot2)
tea <- rnorm(100)
biscuits <- tea + rnorm(100, 0, 1.3)
data <- data.frame(tea, biscuits)
p <- ggplot(data, aes(x = tea, y = biscuits)) +
    geom_point() +
    geom_smooth(method = "lm") +
    labs(x = "Tea", y = "Biscuits") + theme_bw()
print(p)
```

Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

The code gets replaced by its output

Lorem Ipsum

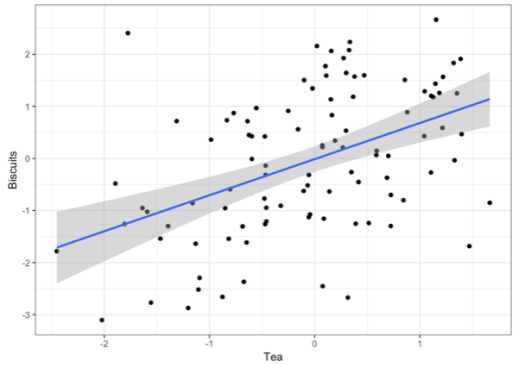
Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do *eiusmod tempor* incididunt ut labore et dolore magna aliqua. Ut enimad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.

```
library(ggplot2)
tea <- rnorm(100)
biscuits <- tea + rnorm(100, 0, 1.3)
data <- data.frame(tea, biscuits)
p <- ggplot(data, aes(x = tea, y = biscuits)) +
    geom_point() +
    geom_smooth(method = "lm") +
    labs(x = "Tea", y = "Biscuits") + theme_bw()
print(p)
```

Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

1. Lorem Ipsum

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do **eiusmod tempor** incididunt ut labore et dolore magna aliqua. Ut enimad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.



Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

```
---
title: "Covid Cases"
author: "Kieran Healy"
format: html
---


```{r}
#| label: setup
#| echo: false
#| message: false

# Don't include code chunks in the document
knitr::opts_chunk$set (echo = FALSE)

library(tidyverse)
library(covdata)

```

Introduction

We will look at some data from the `covdata` package.

```{r}
#| label: data

covnat_daily ▷
  filter(iso3 %in% c("USA", "GBR", "FRA")) ▷
  group_by(cname) ▷
  summarize(cases = sum(cases)) ▷
  knitr::kable()
```
```

Markdown document

```
---
title: "Covid Cases"
author: "Kieran Healy"
format: html
---


```{r}
#| label: setup
#| echo: false
#| message: false

# Don't include code chunks in the document
knitr::opts_chunk$set (echo = FALSE)

library(tidyverse)
library(covdata)

```

Introduction

We will look at some data from the `covdata` package.

```{r}
#| label: data

covnat_daily ▷
  filter(iso3 %in% c("USA", "GBR", "FRA")) ▷
  group_by(cname) ▷
  summarize(cases = sum(cases)) ▷
  knitr::kable()
```
```
```

Header section with metadata

Chunks can have labels or options

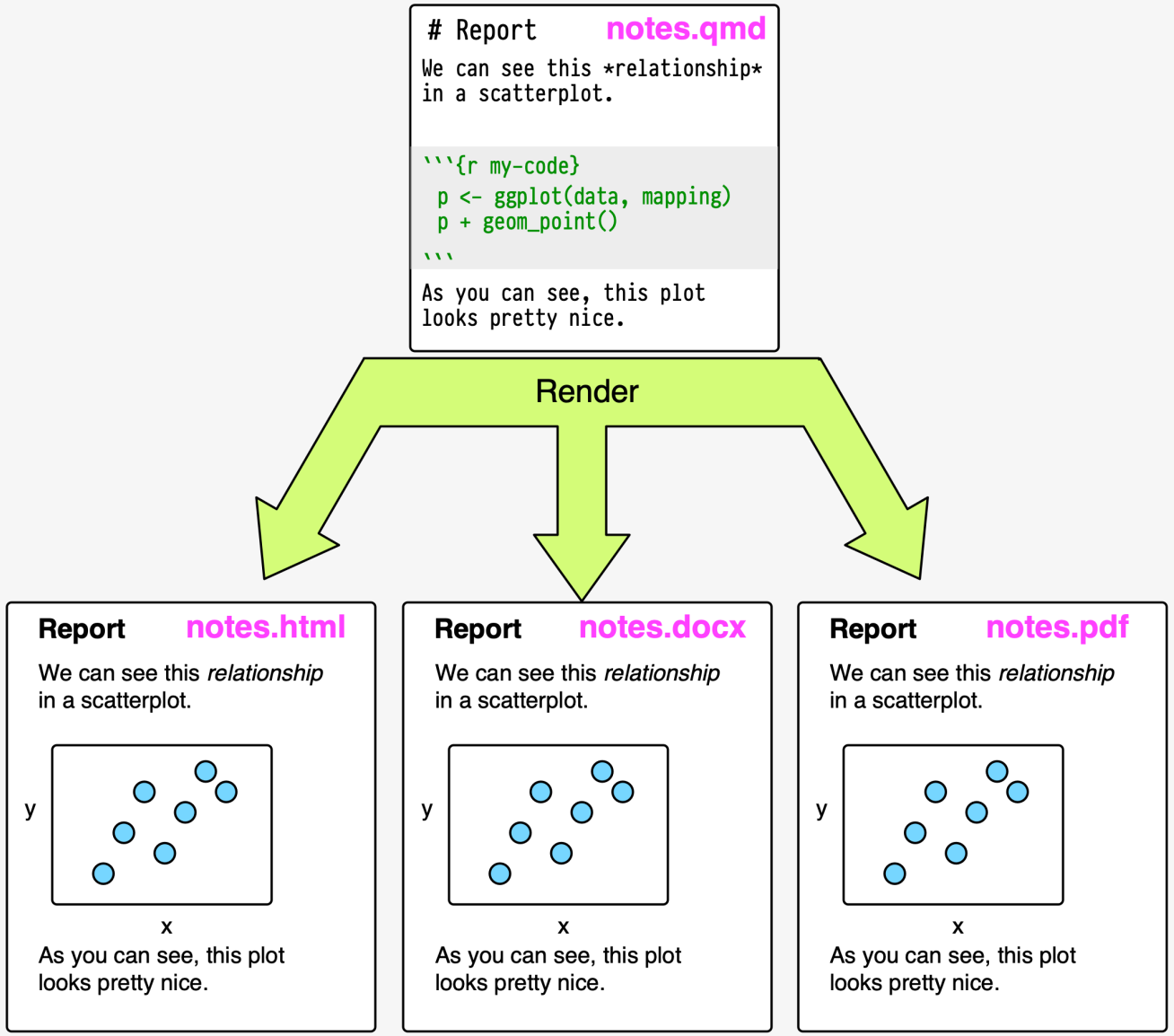
Code chunk (or cell)

Text with markdown formatting

Code chunks can be "played" one at a time

When rendered, code chunks are replaced by their output

Markdown document annotated
```



This approach has its limitations, but it's *very* useful and has many benefits.

Basic markdown summary

Desired style	Use the following Markdown annotation
Heading 1	`# Heading 1`
Heading 2	`## Heading 2`
Heading 3	`### Heading 3` (Actual heading styles will vary.)
Paragraph	Just start typing
Bold	`**Bold**`
Italic	`*Italic*`
Images	`[Alternate text for image](path/image.jpg)`
Hyperlinks	`[Link text](https://www.visualizingsociety.com/)`
Unordered Lists	
- First	`- First`
- Second.	`- Second`
- Third	`- Third`
Ordered Lists	
1. First	`1. First`
2. Second.	`2. Second`
3. Third	`3. Third`
Footnote.[1]	`Footnote[^notelabel]`
[1]The note's content.	`[^notelabel] The note's content.`

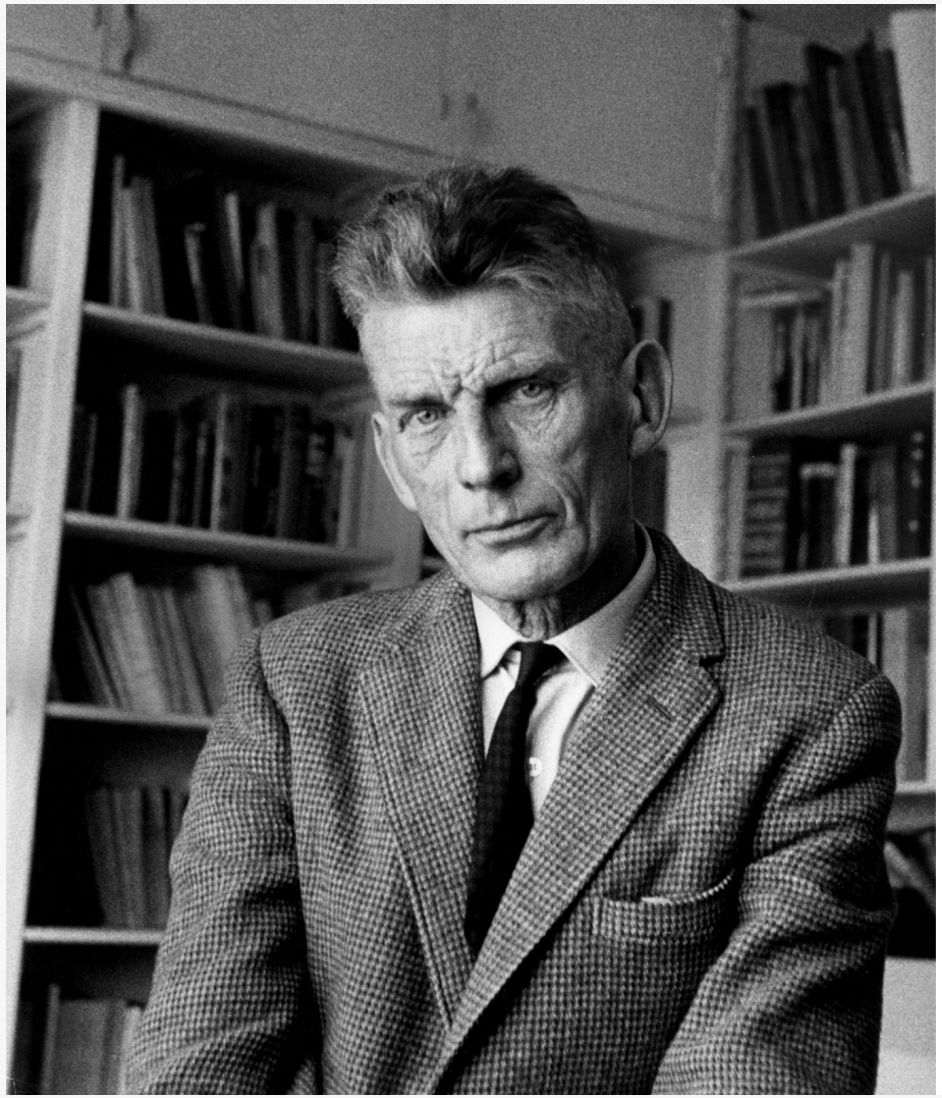
The right frame of mind

This is like learning how to drive a car, or how to cook in a kitchen … or learning to speak a language.

After some orientation to what's where, you will learn best by *doing*.

Software is a pain, but you won't crash the car or burn your house down.

TYPE OUT YOUR CODE BY HAND



Ever tried.
Ever failed.
No matter.
Try again.
Fail again.
Fail better.

Samuel Beckett,
early data analyst

Samuel Beckett

GETTING ORIENTED

Loading the tidyverse libraries

```
library(tidyverse)
```

The tidyverse has several components.

We'll return to this message about Conflicts later.

Again, the code and messages you see here is actual R output, produced at the same time as the slide.

Tidyverse components

```
library(tidyverse)
Loading tidyverse: ggplot2
Loading tidyverse: tibble
Loading tidyverse: tidyr
Loading tidyverse: readr
Loading tidyverse: purrr
Loading tidyverse: dplyr
```

Call the package and ...

- **Draw graphs**
- **Nicer data tables**
- **Tidy your data**
- **Get data into R**
- **Fancy Iteration**
- **Action verbs for tables**

What R looks like

Code you can type and run:

```
## Inside code chunks, lines beginning with a # character are comments
## Comments are ignored by R

my_numbers <- c(1, 1, 2, 4, 1, 3, 1, 5) # Anything after a # character is ignored as well
```

Output:

```
my_numbers
```

```
[1] 1 1 2 4 1 3 1 5
```

This is equivalent to running the code above, typing `my_numbers` at the console, and hitting enter.

What R looks like

By convention, code output in documents is prefixed by ##

Also by convention, outputting vectors, etc, gets a counter keeping track of the number of elements. For example,

```
letters
```

```
 [1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l" "m" "n" "o" "p" "q" "r" "s"
[20] "t" "u" "v" "w" "x" "y" "z"
```

Some things to know about R

0. It’s a calculator

Arithmetic

```
(31 * 12) / 2^4
```

```
[1] 23.25
```

```
sqrt(25)
```

```
[1] 5
```

```
log(100)
```

```
[1] 4.60517
```

```
log10(100)
```

```
[1] 2
```

0. It's a calculator

Arithmetic

```
(31 * 12) / 2^4
```

```
[1] 23.25
```

```
sqrt(25)
```

```
[1] 5
```

```
log(100)
```

```
[1] 4.60517
```

```
log10(100)
```

```
[1] 2
```

Logic

```
4 < 10
```

```
[1] TRUE
```

```
4 > 2 & 1 > 0.5 # The "&" means "and"
```

```
[1] TRUE
```

```
4 < 2 | 1 > 0.5 # The "|" means "or"
```

```
[1] TRUE
```

```
4 < 2 | 1 < 0.5
```

```
[1] FALSE
```

Boolean and Logical operators

Logical equality and inequality (yielding a `TRUE` or `FALSE` result) is done with `==` and `!=`. Other logical operators include `<`, `>`, `<=`, `>=`, and `!` for negation.

```
## A logical test
2 == 2 # Write `=` twice
```

```
[1] TRUE
```

```
## This will cause an error, because R will think you are trying to assign a value
2 = 2

## Error in 2 = 2 : invalid (do_set) left-hand side to assignment
```

```
3 != 7 # Write `!` and then `=` to make `!=`
```

```
[1] TRUE
```

1. Everything in R has a name

```r
my_numbers # We created this a few minutes ago
```

```
[1] 1 1 2 4 1 3 1 5
```

```r
letters  # This one is built-in
```

```
 [1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l" "m" "n" "o" "p" "q" "r" "s"
[20] "t" "u" "v" "w" "x" "y" "z"
```

```r
pi  # Also built-in
```

```
[1] 3.141593
```

Some names are forbidden

Or it's a *really* bad idea to try to use them

```
TRUE
FALSE
Inf
NaN
NA
NULL

for
if
while
break
function
```

2. Everything is an object

There are a few built-in objects:

```
letters
```

```
 [1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l" "m" "n" "o" "p" "q" "r" "s"
[20] "t" "u" "v" "w" "x" "y" "z"
```

```
pi
```

```
[1] 3.141593
```

```
LETTERS
```

```
 [1] "A" "B" "C" "D" "E" "F" "G" "H" "I" "J" "K" "L" "M" "N" "O" "P" "Q" "R" "S"
[20] "T" "U" "V" "W" "X" "Y" "Z"
```

3. You can create objects

In fact, this is mostly what we will be doing.

Objects are created by assigning a thing to a name:

```
## name ... gets ... this stuff
my_numbers <- c(1, 2, 3, 1, 3, 5, 25, 10)

## name ... gets ... the output of the function `c()`
your_numbers <- c(5, 31, 71, 1, 3, 21, 6, 52)
```

The `c()` function *combines* or *concatenates* things

The assignment operator

The assignment operator performs the action of creating objects

Use a keyboard shortcut to write it:

Press `option` *and* `-` on a Mac

Press `alt` *and* `-` on Windows

Assignment with =

You can use = as well as ← for assignment.

```
my_numbers = c(1, 2, 3, 1, 3, 5, 25)

my_numbers
```

```
[1]  1  2  3  1  3  5 25
```

On the other hand, = has a different meaning when used in functions.

I'm going to use ← for assignment throughout.

Be consistent either way.

4. You do things with functions

```r
## this object ... gets ... the output of this function
my_numbers <- c(1, 2, 3, 1, 3, 5, 25, 10)

your_numbers <- c(5, 31, 71, 1, 3, 21, 6, 52)
```

```r
my_numbers
```

```
[1]  1  2  3  1  3  5 25 10
```

4. You do things with functions

Functions can be identified by the parentheses after their names.

```
my_numbers
```

```
[1]  1  2  3  1  3  5 25 10
```

```
## If you run this you'll get an error
mean()
```

What functions usually do

They take **inputs** to **arguments**

They perform **actions**

They produce, or return, **outputs**

```
mean(x = my_numbers)
```

What functions usually do

They take **inputs** to **arguments**

They perform **actions**

They produce, or return, **outputs**

```
mean(x = my_numbers)
```

```
[1] 6.25
```

What functions usually do

```r
## Get the mean of what? Of x.
## You need to tell the function what x is
mean(x = my_numbers)
```

```
[1] 6.25
```

```r
mean(x = your_numbers)
```

```
[1] 23.75
```

What functions usually do

If you don't *name* the arguments, R assumes you are providing them in the order the function expects.

```
mean(your_numbers)
```

```
[1] 23.75
```

What functions usually do

What arguments? Which order? Read the function's help page

```
help(mean)
```

```
## quicker
?mean
```

How to read an R help page?

What functions usually do

Arguments often tell the function what to do in specific circumstances

```
missing_numbers <- c(1:10, NA, 20, 32, 50, 104, 32, 147, 99, NA, 45)

mean(missing_numbers)
```

```
[1] NA
```

```
mean(missing_numbers, na.rm = TRUE)
```

```
[1] 32.44444
```

Or select from one of several options

```
## Look at ?mean to see what `trim` does
mean(missing_numbers, na.rm = TRUE, trim = 0.1)
```

```
[1] 27.25
```

What functions usually do

There are all kinds of functions. They return different things.

```r
summary(my_numbers)
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
   1.00    1.75    3.00    6.25    6.25   25.00 
```

What functions usually do

You can assign the output of a function to a name, which turns it into an object. (Otherwise it'll send its output to the console.)

```
my_summary <- summary(my_numbers)

my_summary
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
   1.00    1.75    3.00    6.25    6.25   25.00 
```

What functions usually do

Objects hang around in your work environment until they are overwritten by you, or are deleted.

```
## rm() function removes objects
rm(my_summary)

my_summary

## Error: object 'my_summary' not found
```

Functions can be nested

```r
c(1:20)
```

```
 [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20
```

```r
mean(c(1:20))
```

```
[1] 10.5
```

```r
summary(mean(c(1:20)))
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
   10.5    10.5    10.5    10.5    10.5    10.5 
```

```r
names(summary(mean(c(1:20))))
```

```
[1] "Min."    "1st Qu." "Median"  "Mean"    "3rd Qu." "Max."
```

```r
length(names(summary(mean(c(1:20)))))
```

```
[1] 6
```

Nested functions are evaluated from the inside out.

Use the pipe operator: |>

Instead of deeply nesting functions in parentheses, we can use the *pipe operator*:

```
c(1:20) |> mean() |> summary() |> names() |>  length()
```

```
[1] 6
```

Read this operator as "***and then***"

Use the pipe operator: |>

Better, vertical space is free in R:

```
c(1:20) |>
  mean() |>
  summary() |>
  names() |>
  length()
```

```
[1] 6
```

Pipelines make code more readable

Not great, Bob:

```
serve(stir(pour_in_pan(whisk(crack_eggs(get_from_fridge(eggs), into = "bowl"), len = 40), temp = "med-high")))
```

Notice how the first thing you read is the last operation performed.

Pipelines make code more readable

We can use vertical space and indents, but it's really not much better:

```
serve(
  stir(
    pour_in_pan(
      whisk(
        crack_eggs(
          get_from_fridge(eggs),
        into = "bowl"),
      len = 40),
    temp = "med-high")
  )
)
```

Pipelines make code more readable

Much nicer:

```
eggs ▷
  get_from_fridge() ▷
  crack_eggs(into = "bowl") ▷
  whisk(len = 40) ▷
  pour_in_pan(temp = "med-high") ▷
  stir() ▷
  serve()
```

We'll still use nested parentheses quite a bit, often in the context of a function working inside a pipeline. But it's good not to have too many levels of nesting.

The other pipe: %>%

The Base R pipe operator, ▷ is a relatively recent addition to R.

Piping operations were originally introduced in a package called called `magrittr`, where it took the form **%>%**

The other pipe: `%>%`

The Base R pipe operator, `|>` is a relatively recent addition to R.

Piping operations were originally introduced in a package called called `magrittr`, where it took the form `%>%`

It's been so successful, a version of it has been incorporated into Base R. For our puposes, they're the same.

Functions are bundled into packages

Packages are loaded into your working environment using the `library()` function:

```r
## A package containing a dataset rather than functions
library(gapminder)

gapminder
```

```
# A tibble: 1,704 × 6
   country     continent  year lifeExp      pop gdpPercap
   <fct>       <fct>     <int>   <dbl>    <int>     <dbl>
 1 Afghanistan Asia       1952    28.8  8425333      779.
 2 Afghanistan Asia       1957    30.3  9240934      821.
 3 Afghanistan Asia       1962    32.0 10267083      853.
 4 Afghanistan Asia       1967    34.0 11537966      836.
 5 Afghanistan Asia       1972    36.1 13079460      740.
 6 Afghanistan Asia       1977    38.4 14880372      786.
 7 Afghanistan Asia       1982    39.9 12881816      978.
 8 Afghanistan Asia       1987    40.8 13867957      852.
 9 Afghanistan Asia       1992    41.7 16317921      649.
10 Afghanistan Asia       1997    41.8 22227415      635.
# ℹ 1,694 more rows
```

Functions are bundled into packages

You need only *install* a package once (and occasionally update it):

```
## Do at least once for each package. Once done, not needed each time.
install.packages("palmerpenguins", repos = "http://cran.rstudio.com")

## Needed sometimes, especially after an R major version upgrade.
update.packages(repos = "http://cran.rstudio.com")
```

Functions are bundled into packages

But you must *load* the package in each R session before you can access its contents:

```r
## To load a package, usually at the start of your RMarkdown document or script file
library(palmerpenguins)
penguins
```

```
# A tibble: 344 × 8
   species island    bill_length_mm bill_depth_mm flipper_length_mm body_mass_g
   <fct>   <fct>              <dbl>         <dbl>             <int>       <int>
 1 Adelie  Torgersen           39.1          18.7               181        3750
 2 Adelie  Torgersen           39.5          17.4               186        3800
 3 Adelie  Torgersen           40.3          18                 195        3250
 4 Adelie  Torgersen           NA            NA                  NA          NA
 5 Adelie  Torgersen           36.7          19.3               193        3450
 6 Adelie  Torgersen           39.3          20.6               190        3650
 7 Adelie  Torgersen           38.9          17.8               181        3625
 8 Adelie  Torgersen           39.2          19.6               195        4675
 9 Adelie  Torgersen           34.1          18.1               193        3475
10 Adelie  Torgersen           42            20.2               190        4250
# ℹ 334 more rows
# ℹ 2 more variables: sex <fct>, year <int>
```

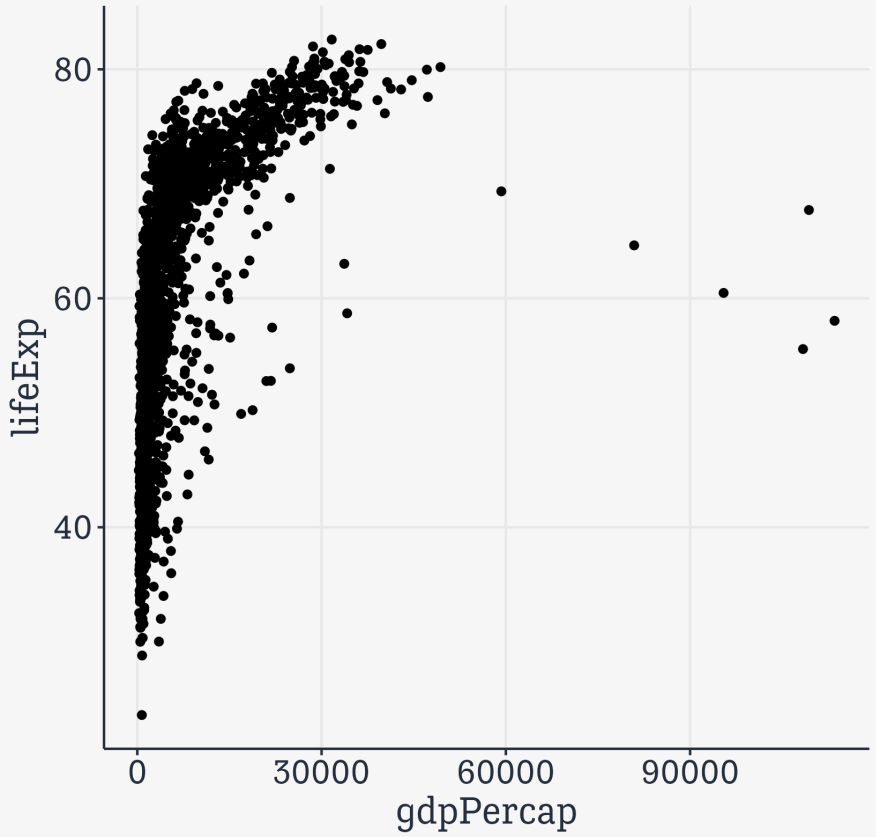
Let’s Go!

Like before

```r
library(tidyverse)
library(gapminder)

p <- ggplot(data = gapminder,
            mapping = aes(x = gdpPercap,
                          y = lifeExp))

p + geom_point()
```



What we did

```r
library(tidyverse)

library(gapminder)
```

Load the packages we need: `tidyverse` and `gapminder`

What we did

```
p <- ggplot(data = gapminder,
            mapping = aes(x = gdpPercap,
                          y = lifeExp))
```

New object named **`p`** `gets` the output of the `ggplot()` *function*, given these *arguments*

Notice how one of the arguments, `mapping`, is itself taking the output of a function named `aes()`

What we did

```
p + geom_point()
```

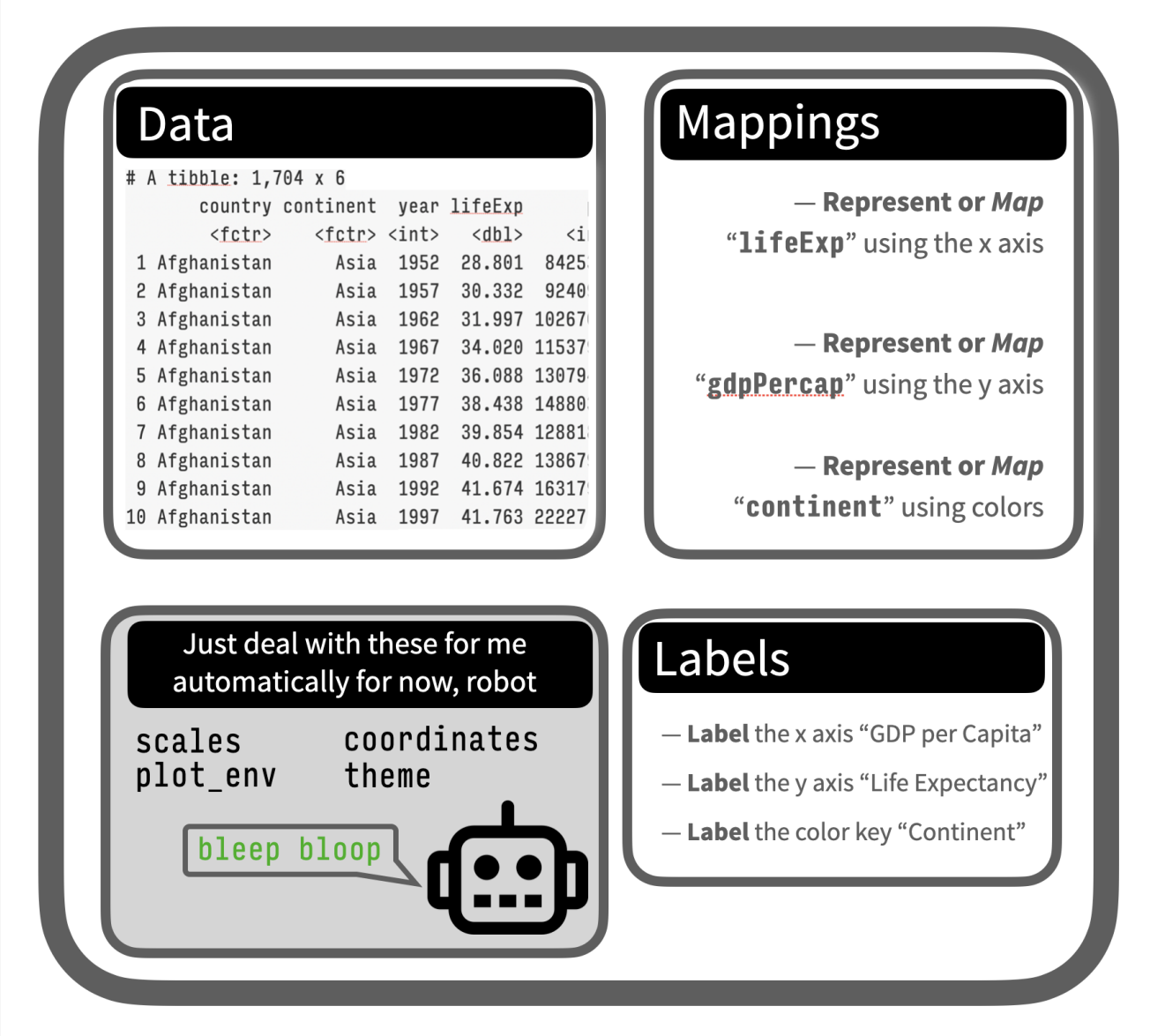
Show me the output of the **p** object and the `geom_point()` function.

The + here acts just like the ▷ pipe, but for ggplot functions only. (This is an accident of history.)

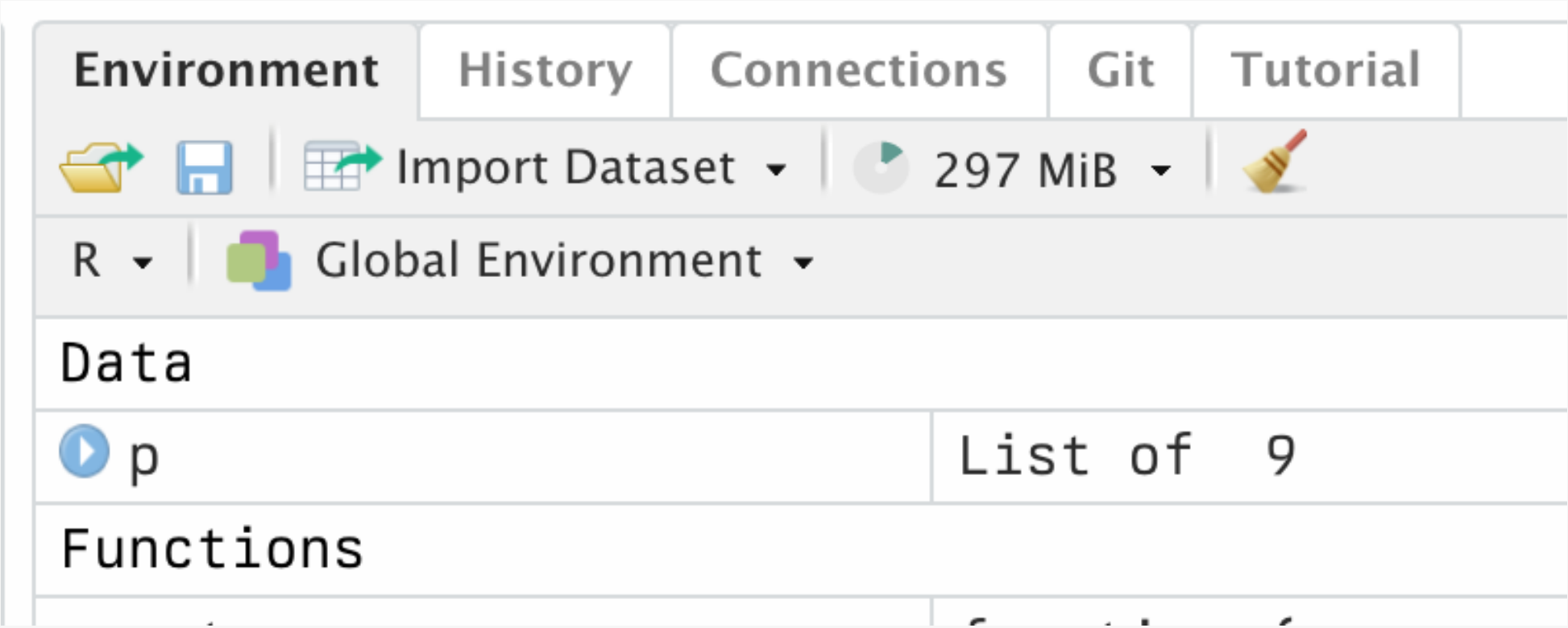
And what is R doing?

R objects are just lists of stuff to use or things to do

Objects are like Bento Boxes


Data
A tibble: 1,704 x 6
country continent year lifeExp
<fctr> <fctr> <int> <dbl> <i
1 Afghanistan Asia 1952 28.801 8425
2 Afghanistan Asia 1957 30.332 9240
3 Afghanistan Asia 1962 31.997 10267
4 Afghanistan Asia 1967 34.020 11537
5 Afghanistan Asia 1972 36.088 13079
6 Afghanistan Asia 1977 38.438 14880
7 Afghanistan Asia 1982 39.854 12881
8 Afghanistan Asia 1987 40.822 13867
9 Afghanistan Asia 1992 41.674 16317
10 Afghanistan Asia 1997 41.763 22227
Mappings
— Represent or Map
"lifeExp" using the x axis
— Represent or Map
"gdpPercap" using the y axis
— Represent or Map
"continent" using colors
Just deal with these for me automatically for now, robot
scales
coordinates
plot_env
theme
bleep bloop
Labels
— Label the x axis "GDP per Capita"
— Label the y axis "Life Expectancy"
— Label the color key "Continent"

The p object



Peek in with the object inspector

p

Show Attributes

Name	Type	Value
p	list [9] (S3: gg, ggplot)	List of length 9
data	list [1704 x 6] (S3: tbl_df, tbl,	A tibble with 1704 rows and 6 columns
layers	list [0]	List of length 0
scales	environment [1] (S3: ScalesLis	<environment: 0x11f8106b0>
mapping	list [2] (S3: uneval)	List of length 2
theme	list [0]	List of length 0
coordinates	environment [5] (S3: CoordCa	<environment: 0x11f8150c8>
facet	environment [2] (S3: FacetNul	<environment: 0x12a81ab00>
plot_env	environment [6]	<environment: R_GlobalEnv>
labels	list [2]	List of length 2

Peek in with the object inspector

Core concepts: mappings + geoms

The core idea, which we'll focus on more formally next week, is that we have *data*, arranged in columns, that we want to represent visually on some sort of plot.

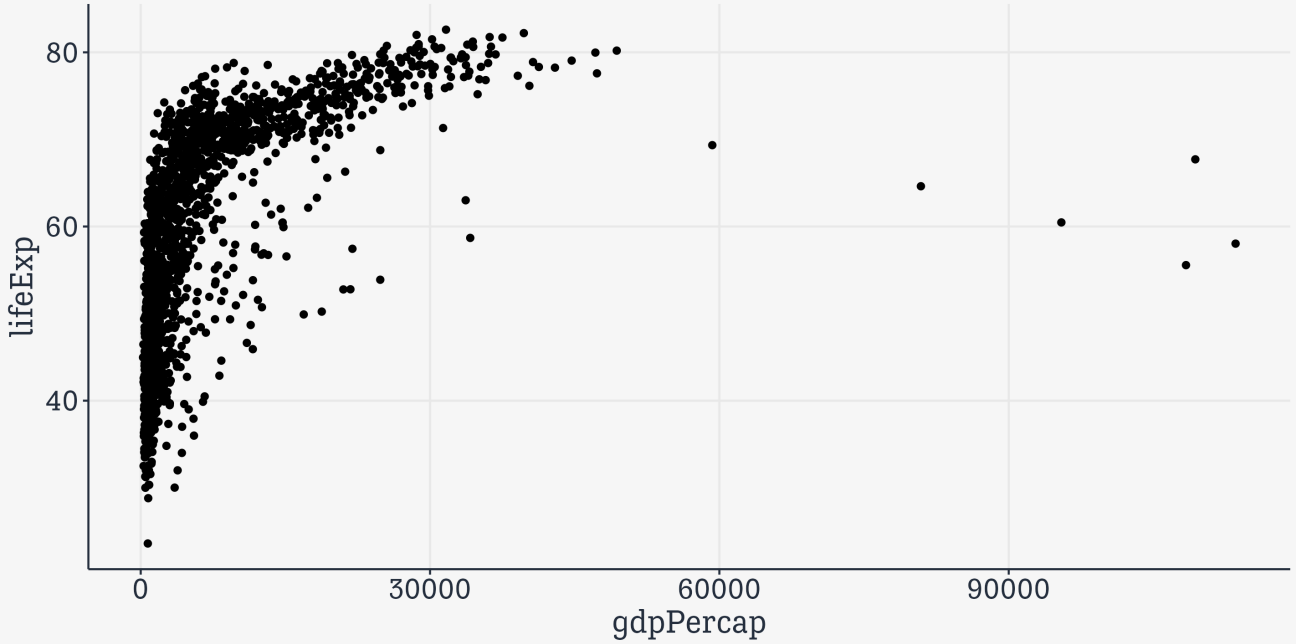
That means we need a *mapping* — a link, a connection, a representation — between things in our table and stuff we can draw. That is what the *mapping* argument is for.

And we need a *geom* — a kind of plot, a particular sort of graph — that we draw with that.

Practical examples

Let's try some live examples ...How might we improve or extend this graph based on the data we have? Or how might we look at it differently?

```
p + geom_point()
```




```
```