

R in production

Code is run repeatedly

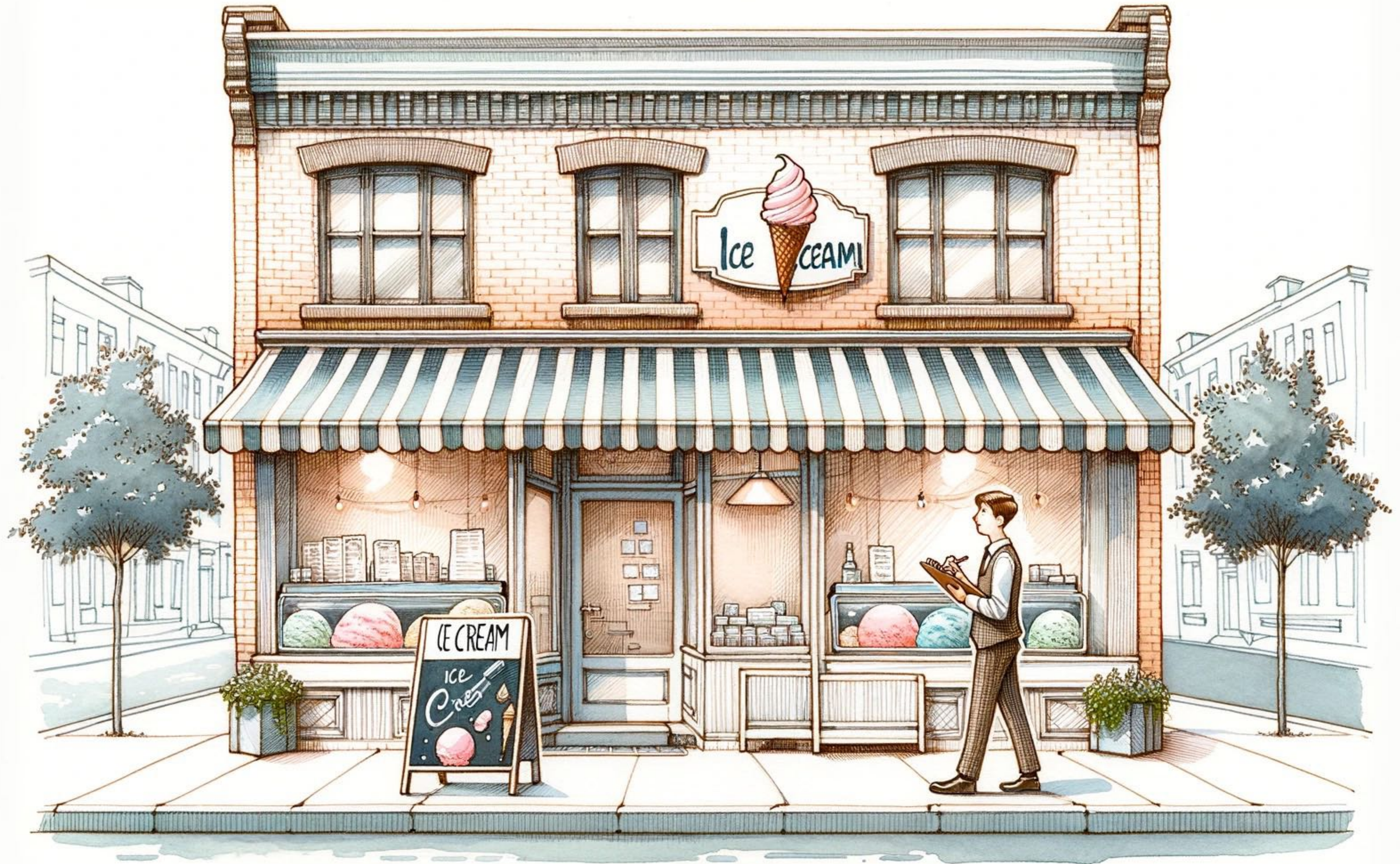
Hadley Wickham

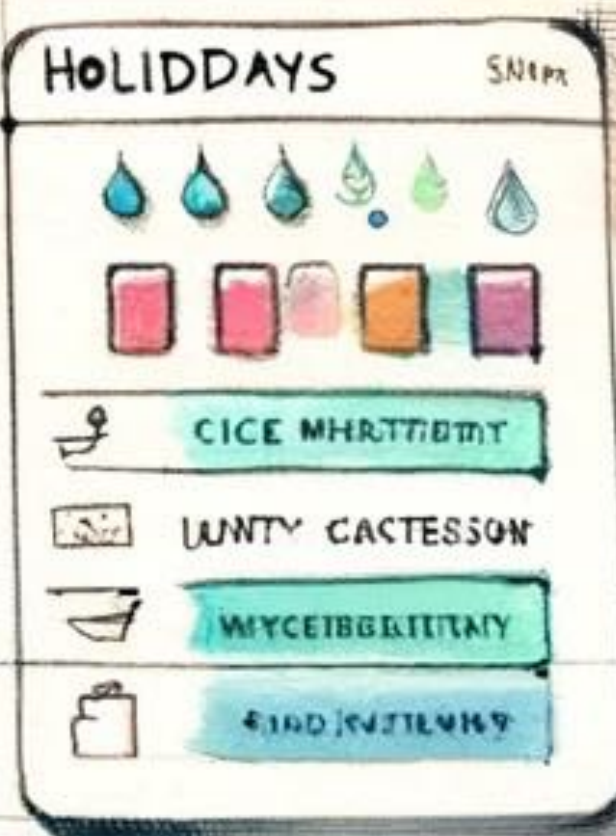
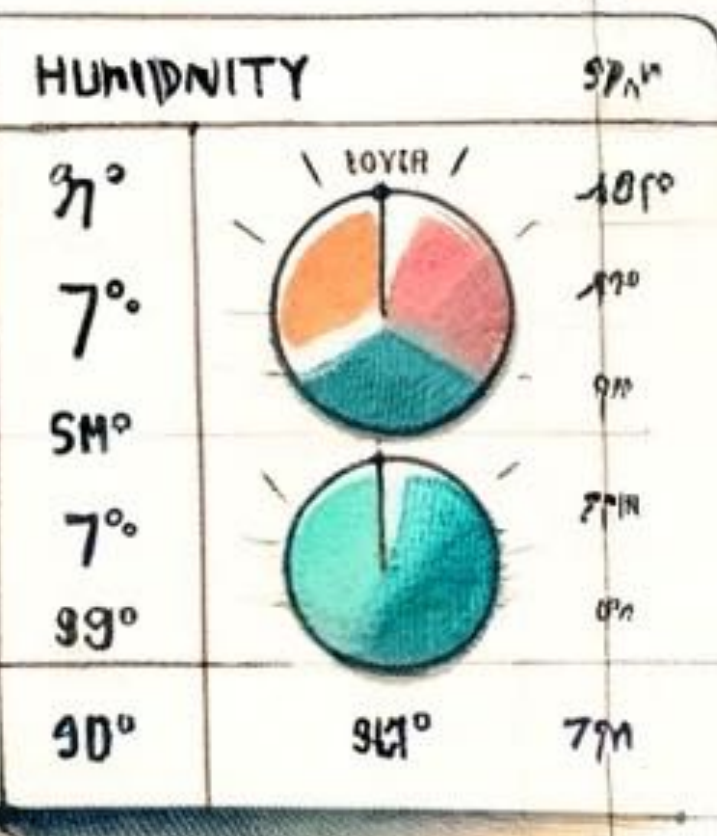
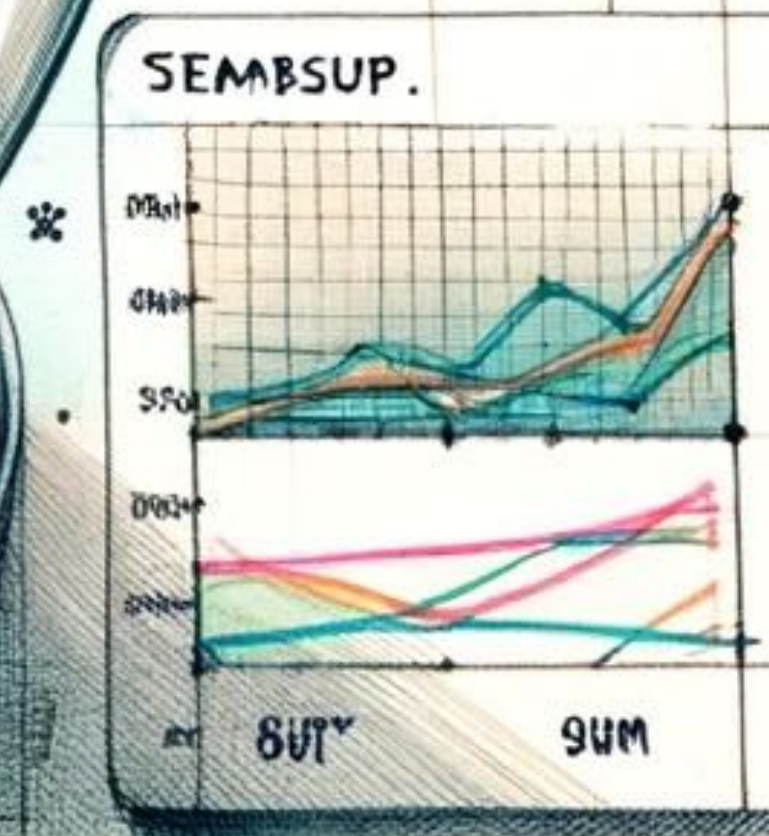
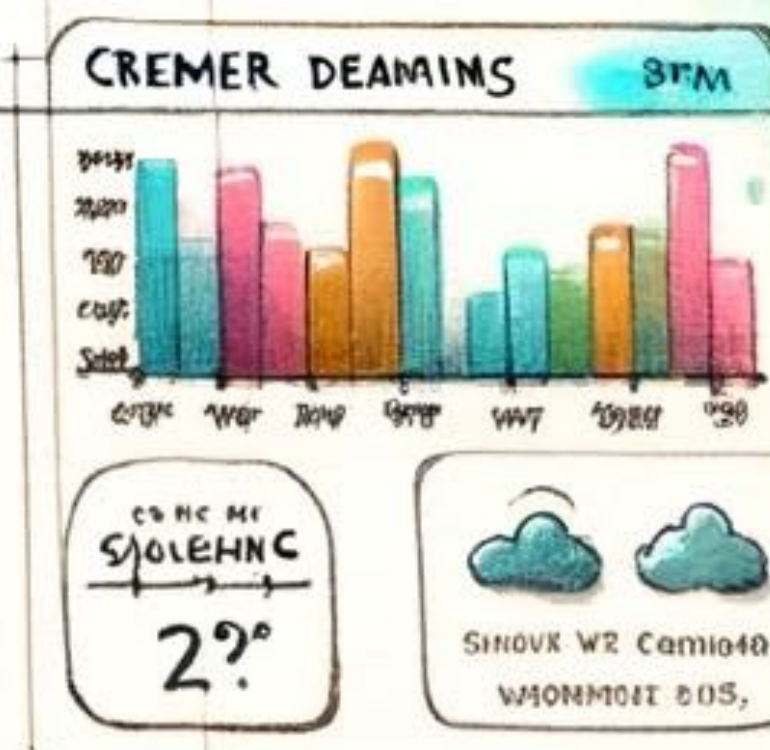
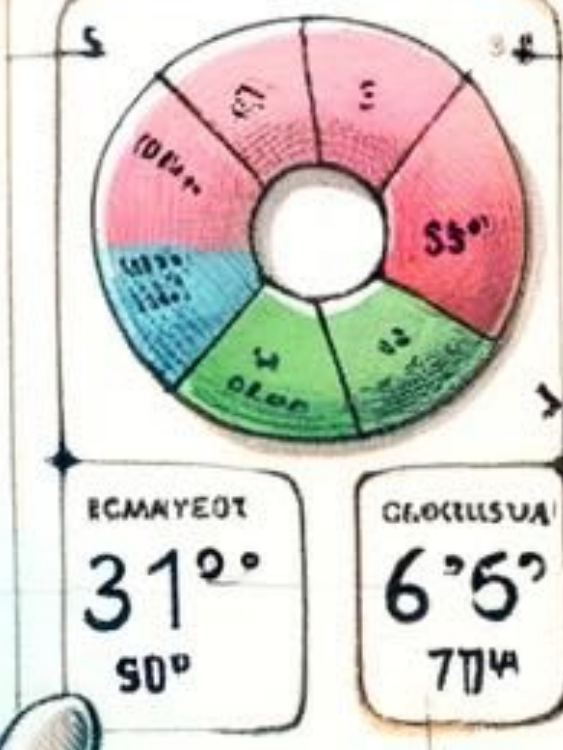
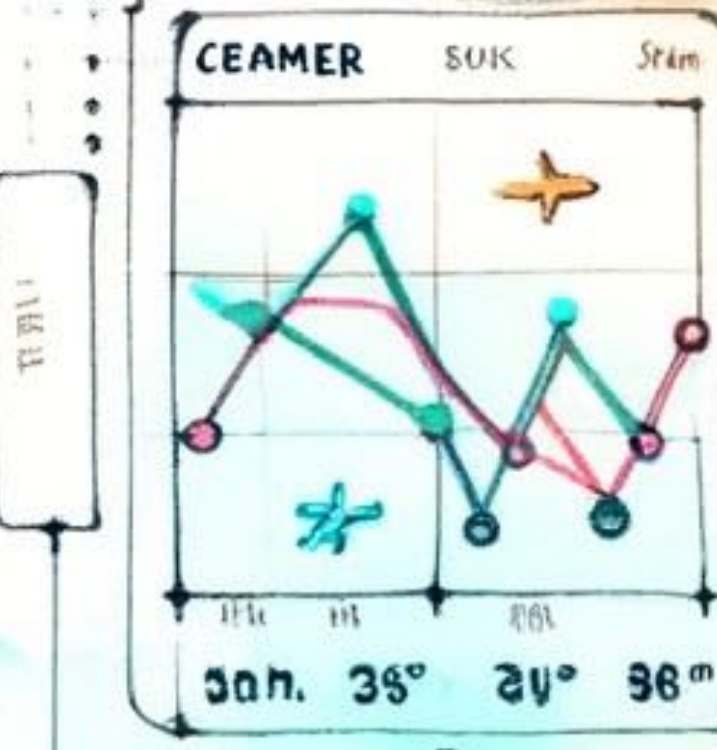
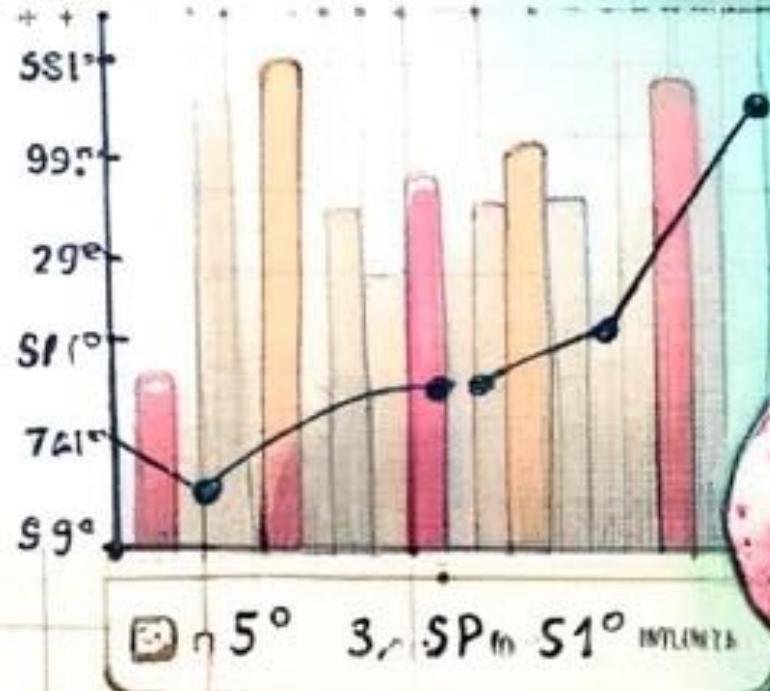
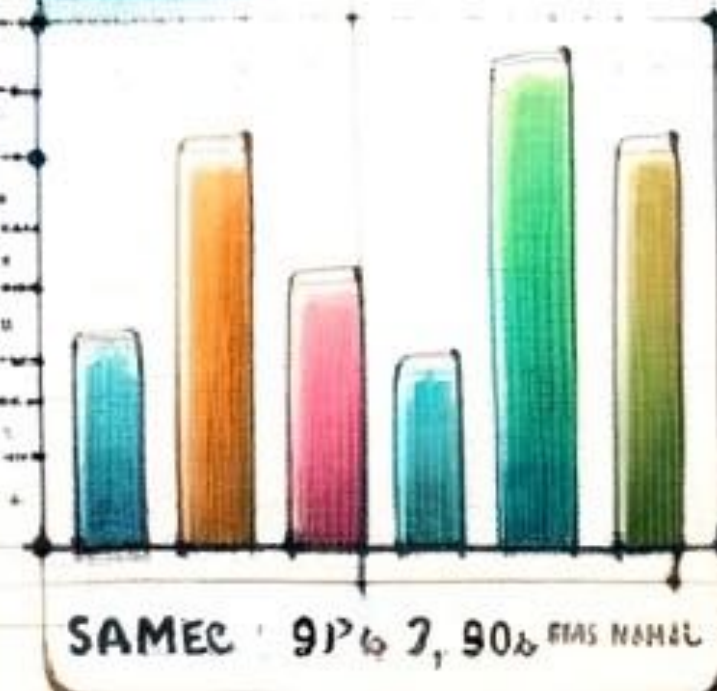
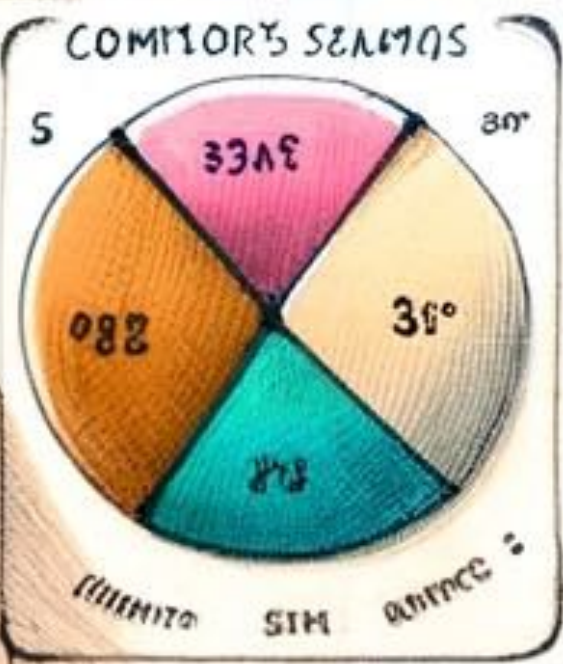
Chief Scientist, Posit

September 2025

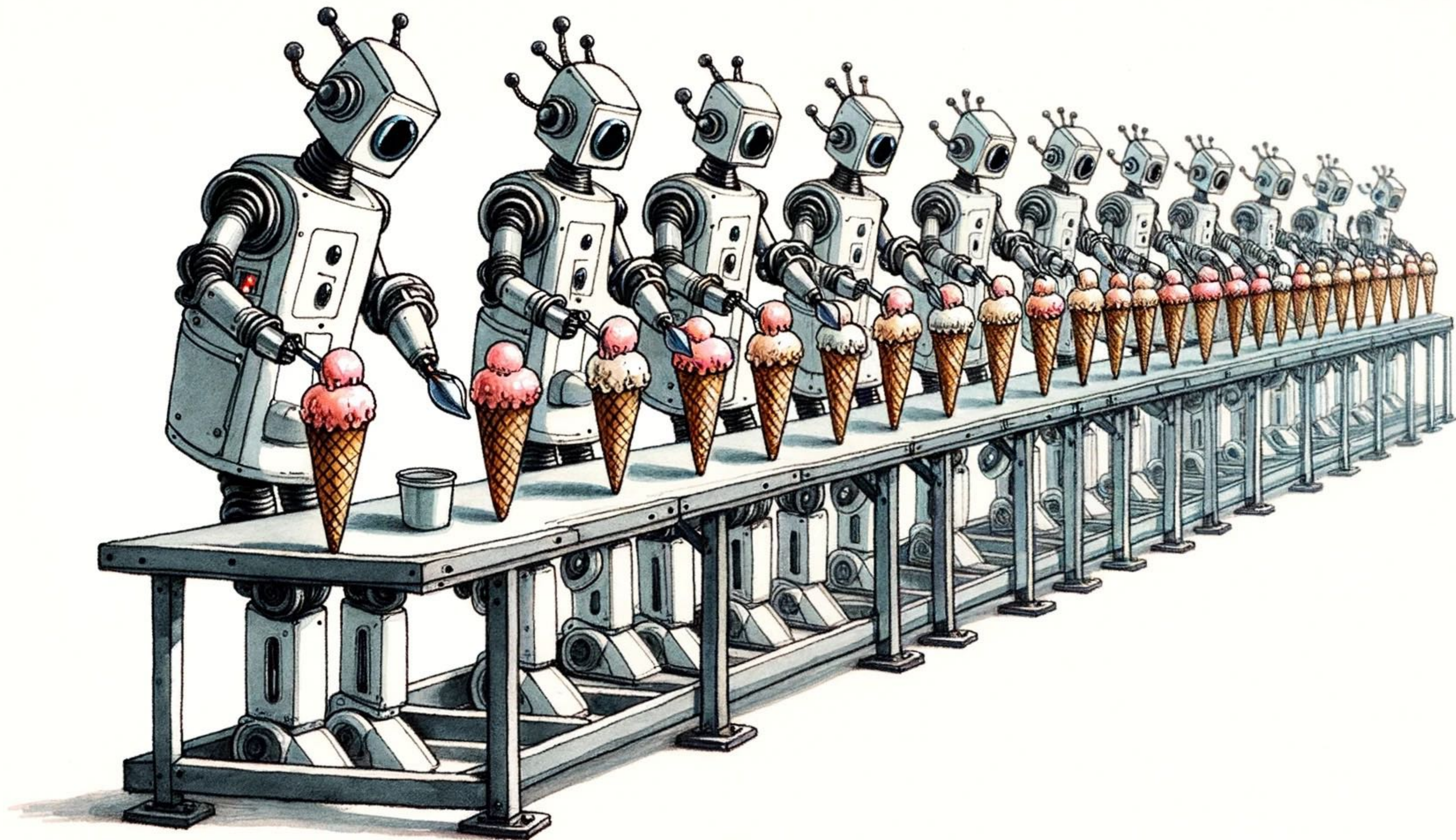


The challenges





IDE ICREREM



The
data
changes



The
schema
changes

date	temperature
05/01/2024	64.4
05/02/2024	68.0
05/03/2024	71.6
05/04/2024	66.2
05/05/2024	69.8
05/06/2024	73.4
05/07/2024	68.0
05/08/2024	71.6

The
schema
changes

date	temp
2024-05-01	18
2024-05-02	20
2024-05-03	22
2024-05-04	19
2024-05-05	21
2024-05-06	23
2024-05-07	20
2024-05-08	22

What changed? How is it likely to affect your code?

date	temperature
05/01/2024	64.4
05/02/2024	68.0
05/03/2024	71.6
05/04/2024	66.2
05/05/2024	69.8
05/06/2024	73.4
05/07/2024	68.0
05/08/2024	71.6

date	temp
2024-05-01	18
2024-05-02	20
2024-05-03	22
2024-05-04	19
2024-05-05	21
2024-05-06	23
2024-05-07	20
2024-05-08	22

Change	Impact
column name	probably errors
date format	might just work might error
temperature units	garbage predictions

A
package
changes



The platform changes

R/Python
System libraries
Operating system
Architecture



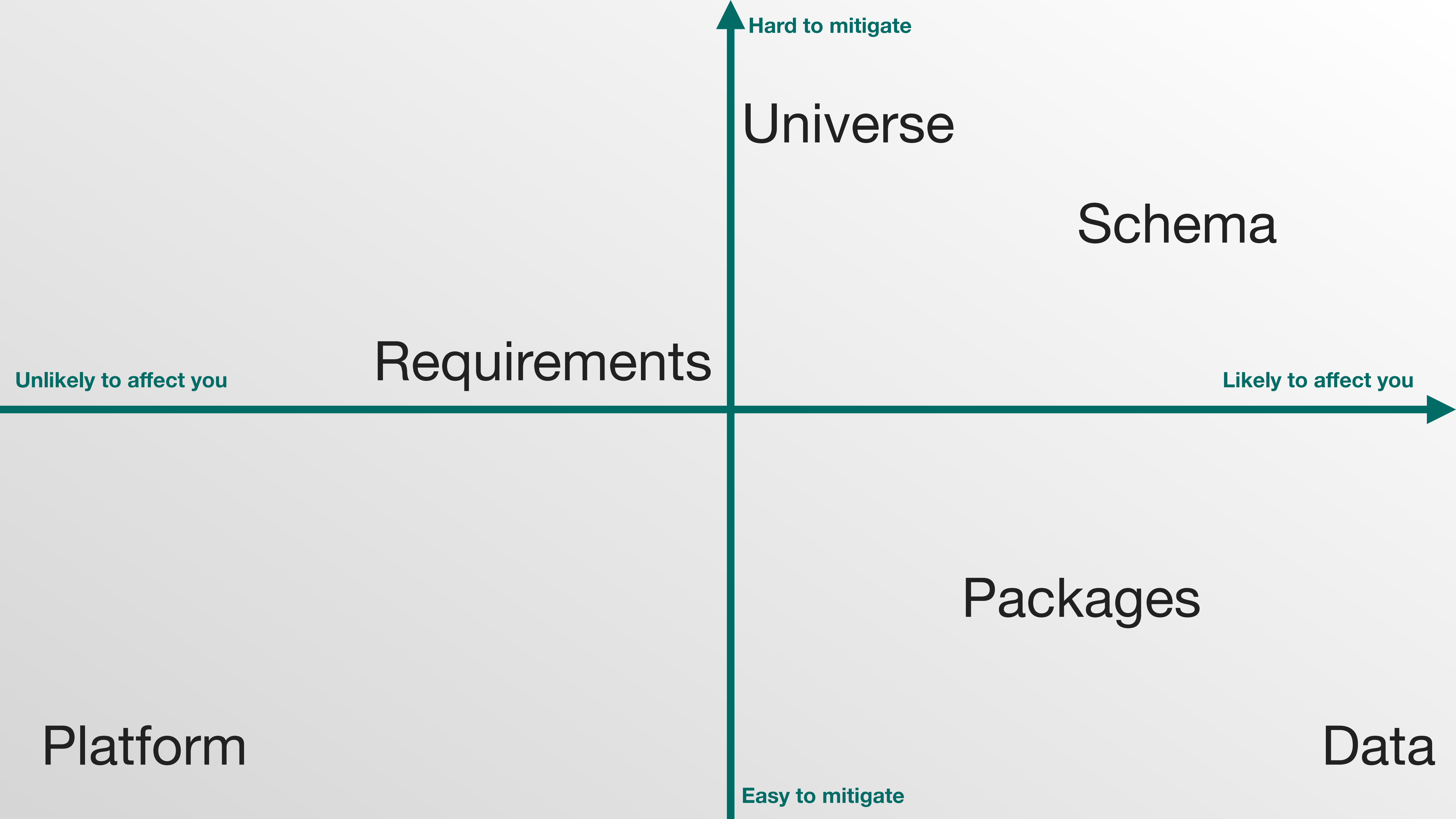
The universe changes

Concept drift
Model drift
Data drift



A requirement changes





1. Platform

2. Packages

3. Schema

4. Requirements

Platform

Insulate yourself from platform changes with a container

- Defines operating system + system dependencies
- Isolated
- Portable
- Immutable
- Scalable

Useful containers to know about

- GitHub action containers:
<https://docs.github.com/en/actions/writing-workflows/choosing-where-your-workflow-runs/choosing-the-runner-for-a-job>
- <https://github.com/rocker-org/rocker>
- <https://github.com/r-hub/rhub>
- <https://github.com/r-hub/evercran>

We have never experienced a problem caused by using ubuntu-latest on GitHub

Packages

Reducing version dependency

Right-sizing dependencies

- I strongly believe you shouldn't care at all about dependencies when you're prototyping. It's fine to take a dependency even if it's one function that saves you 5 minutes.
- But more dependencies can make deployment more challenging, and can make your code more fragile.
- So when you've got to the point of having something to deploy it's worth taking a look at your dependencies to see if there any that you can now shed.
- <https://www.tidyverse.org/blog/2019/05/itdepends/>

How could you reduce dependencies here?

```
library(tidyverse)

create_silly_story ← function(name, animal, color, food, place) {
  str_c(
    "Once upon a time, there was a magical ", animal, " named ", name, ".\n",
    name, " had a beautiful ", color, " mane that shimmered in the sunlight.\n",
    "Every day, ", name, " would prance through the fields of ", food, " near ", place,
    ".\n",
    "One day, ", name, " discovered a secret portal that led to a world made entirely
of ", food, "!\n",
    name, " lived happily ever after, munching on ", food, " and spreading ", color, "
joy wherever ", name, " went.\n"
  )
}
```


R's condition hierarchy



Error

You must fix this before continuing

Warning

We'll let it go this time, but you need to fix this

Message

FYI; no action needed

But still worth eliminating to
make logs easier to read

Eliminate all warnings and messages

```
options(warn = 1)
```

```
options(warn = 2)
```

```
# https://lifecycle.r-lib.org/
```

```
options(lifecycle_verbosity = "warning")
```

```
options(lifecycle_verbosity = "error")
```

```
# Tools of last resort
```

```
suppressWarnings()
```

```
suppressMessages()
```


Eliminate all messages and warnings in this code

```
library(dplyr)
```

```
library(readr)
```

```
df1 <- data.frame(x = c(1, 1, 2), y = 1:3)
```

```
df2 <- data.frame(x = c(1, 2, 2), z = letters[1:3])
```

```
left_join(df1, df2)
```

```
path <- tempfile()
```

```
write_csv(df1, path)
```

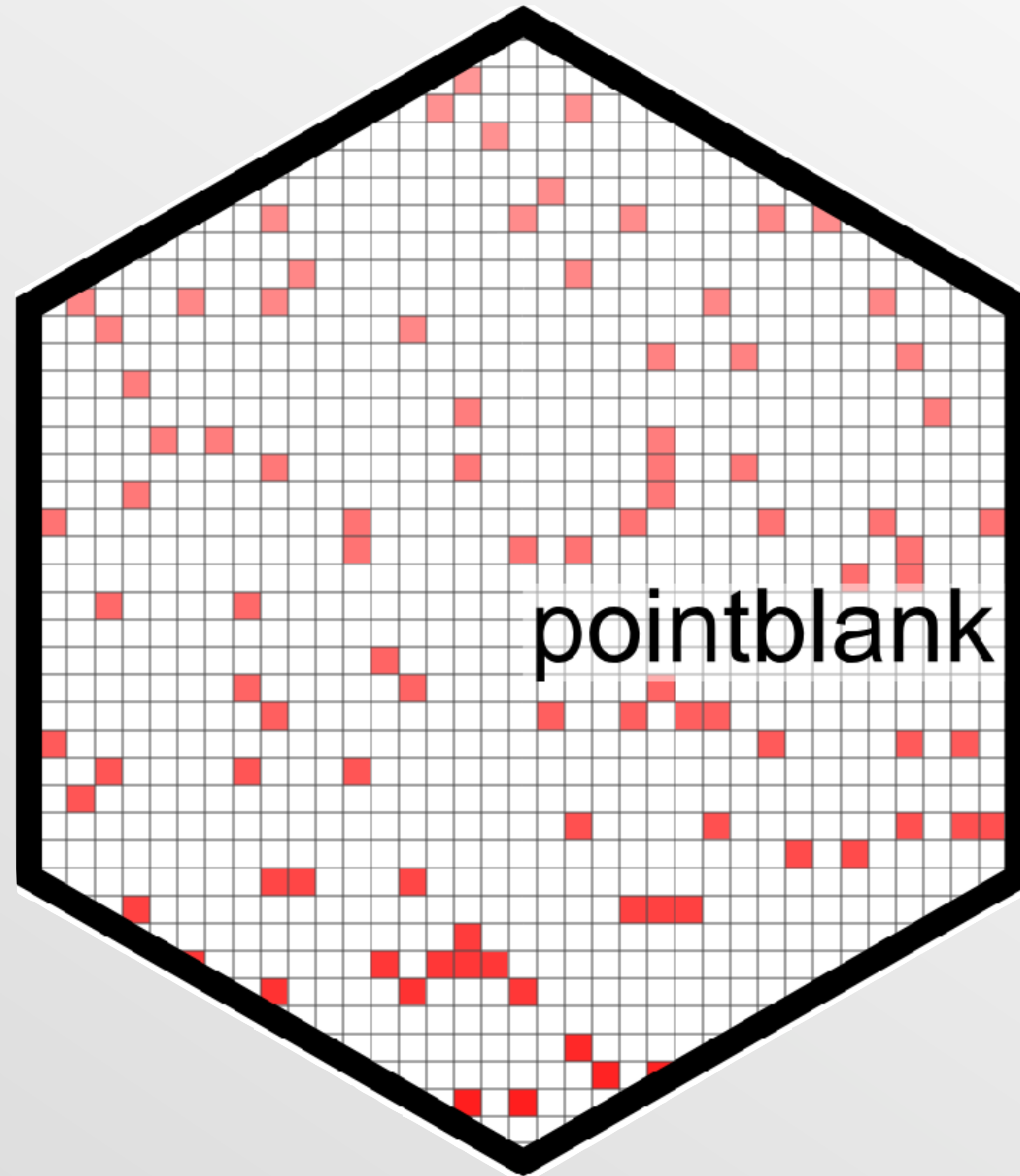
```
read_csv(path)
```


Schema

Mitigation strategy

- Social
 - Make friends with the folks who provide your data!
 - Establish data contracts
- Technical
 - Aggressively check that your data looks as expected

Pointblank provides a flexible tool for validating data



There are six ways to use it

- Canned data report
- Data quality reporting
- Pipeline data validation
- Expectations in unit tests
- Custom control flow
- Rmd integration

We'll focus on two

- **Canned data report**
- Data quality reporting
- **Pipeline data validation**
- Expectations in unit tests
- Custom control flow
- Rmd integration

scan_data() produces a handy report

```
report ← pointblank::scan_data(mtcars)
```

```
report
```

```
pointblank::export_report(mtcars, "mtcars-report.html")
```

```
# Interactions/correlations sections are slow for large datasets
```

```
# so you can drop them with this code
```

```
pointblank::scan_data(ggplot2::diamonds, sections = "OVMS")
```


Pipeline validation throws an error if data isn't as you expect

- Check variable types:
col_is_numeric(), col_is_character(), ...
- Check missingness (if you don't expect any):
col_vals_not_null()
- Check ranges/valid values:
col_is_between(), col_vals_in_set()
- Special purpose:
col_vals_expr(), col_vals_regex()
- Custom: **specially()**
e.g. `specially(\(df) nrow(anti_join(df, ref) == 0)`

What would you validate for the ice cream data?

```
library(dplyr)
```

```
data <- tribble(
  ~date,      ~temperature,
  "2024-05-01", 64.4,
  "2024-05-02", 68.0,
  "2024-05-03", 71.6,
  "2024-05-04", 66.2,
  "2024-05-05", 69.8,
  "2024-05-06", 73.4,
  "2024-05-07", 68.0,
  "2024-05-08", 71.6
)
```

```
data <- data > mutate(date = as.Date(date))
```


Some ideas

```
library(pointblank)
```

```
data ▷
```

```
  col_is_date(date) ▷
```

```
  col_is_numeric(temperature) ▷
```

```
  col_vals_not_null(date) ▷
```

```
  col_vals_between(temperature, 30, 120)
```


Your turn

```
# You can download USD ↔ EUR exchange range data from this API
url ← "https://data-api.ecb.europa.eu/service/data/EXR/
D.USD.EUR.SP00.A?
format=csvdata&detail=dataonly&startPeriod=2024-08-08"

# Assume you want to download this data daily.
# Write some pointblank code to ensure that you get an
# error if the data format changes
```


Requirements

No great insights, but...

- Adopt the mindset that a successful project will attract changes and will require upkeep.
- If you are using GitHub (or similar) internally, issues and projects are a great way to track desired changes.
- Do your best to **batch** and **time box** projects.
- Plan to regularly invest time in refactoring.

What is refactoring?

- Rewriting code so the reduces are the same but the internals are better: easier to read or easier to maintain.
- Second order benefit is improving your programming skills.
- Common tasks:
 - Enforce common style
 - Fix any kludges you added in the heat of the moment
 - Reduce redundant packages
 - Reduce code by using packages

Your turn

What other techniques for handling changing requirements have you found useful in your career?