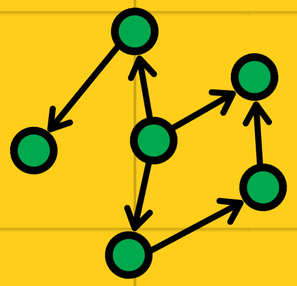
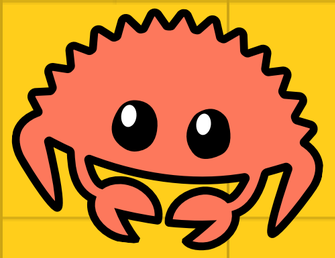
Rust for Good: Systems Programming in Science and Public Health

Dr. Caroline Morton

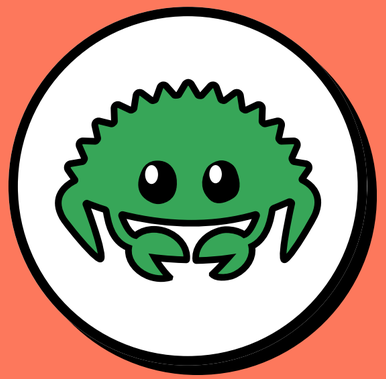
Software Engineer, Epidemiologist and Former GP



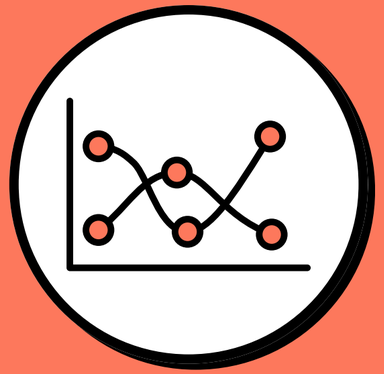


What we are going to talk about

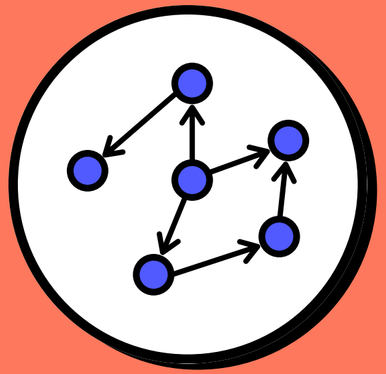
Example from my own work



Great Rust tools



How to build great tools

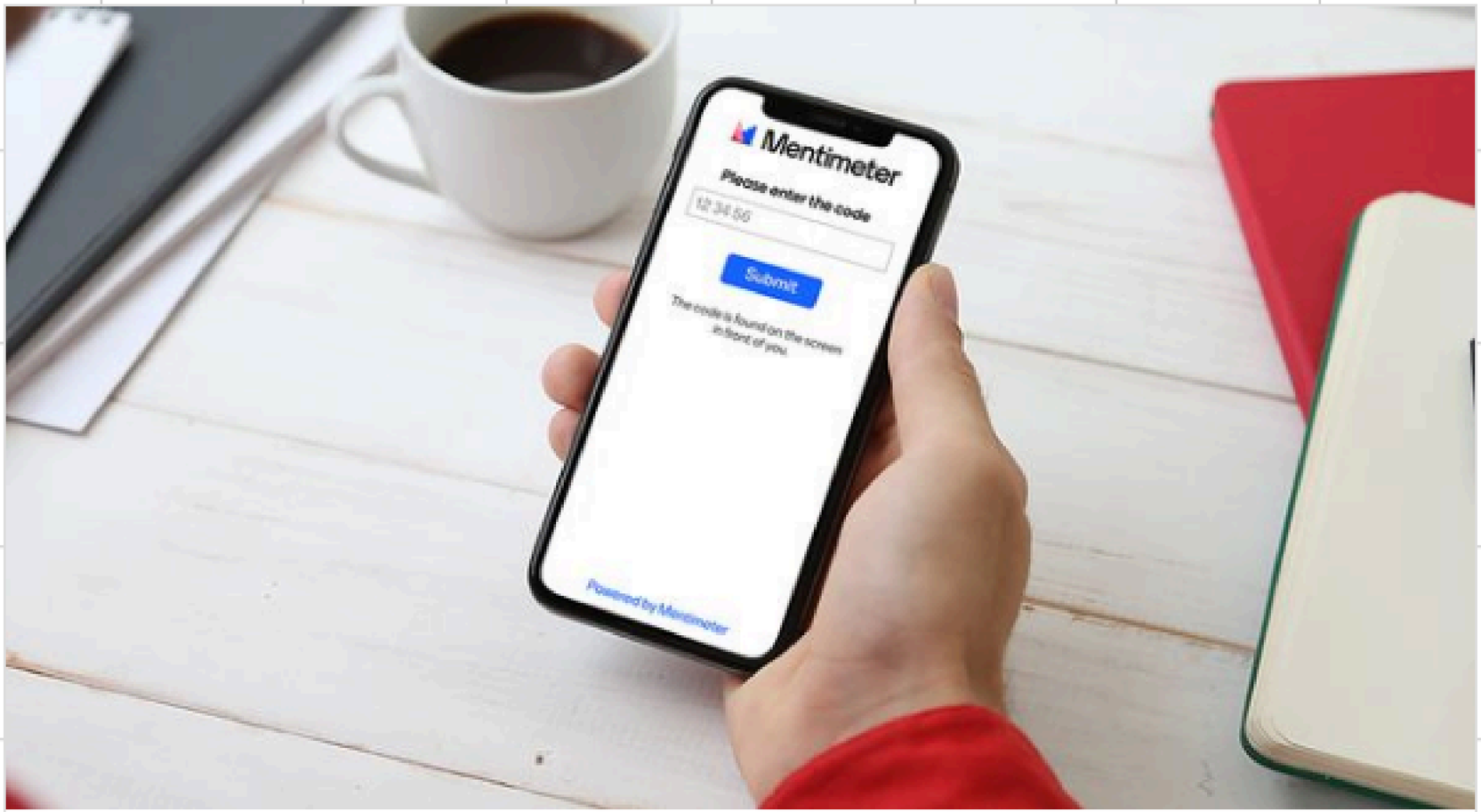


What next?

What we want from great science?

- **Reproducibility:** Research must produce identical results across runs, teams, and datasets
- **Transparency**: Methods and logic must be explicit and auditable
- **Trust**: Results that scientists, reviewers, and policymakers can believe
- **Accessibility**: Findings and methods understandable to both domain experts and the broader research community

Quiz Time



5260 5529

Rust in Science Quiz

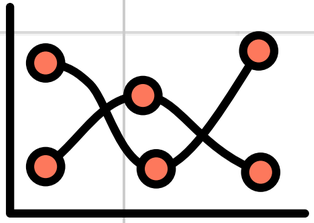
What we want from great science?

- **Reproducibility:** Research must produce identical results across runs, teams, and datasets
- **Transparency**: Methods and logic must be explicit and auditable
- **Trust**: Results that scientists, reviewers, and policymakers can believe
- **Accessibility**: Findings and methods understandable to both domain experts and the broader research community

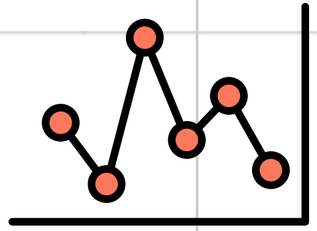
What we want from great tools?

- **Reproducibility is foundational:** Research pipelines must produce identical results across runs, teams, and datasets - the same input always yields the same output
- Code must be **trustworthy:**
  - Errors surface **clearly**, not silently
  - Logic is **explicit and auditable**
  - Results can be **believed** by scientists, reviewers, and policymakers
- Documentation speaks to both domain experts and the broader research community

Codelist Tools

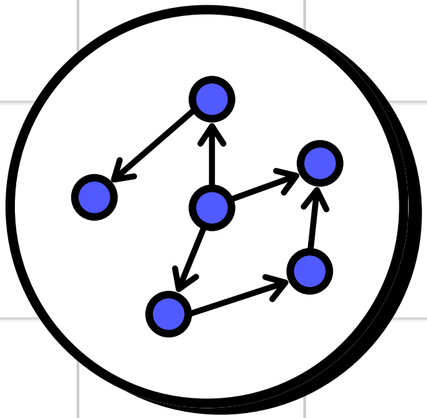


The Problem

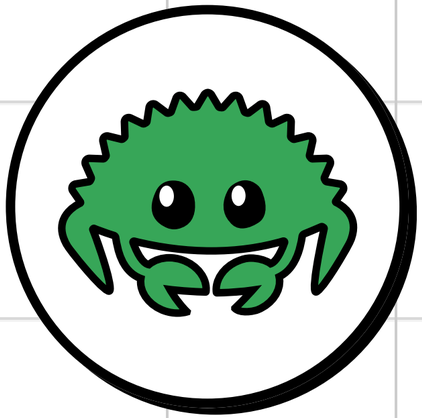


- Epidemiology relies on **reproducible** pipelines: data cleaning, coding, transformation, analysis.
- Code must be **repeated**: the same input produces the same output, across runs, teams, and datasets
- Code must be **believable**: errors are surfaced, logic is explicit, results can be trusted by scientists and policymakers.

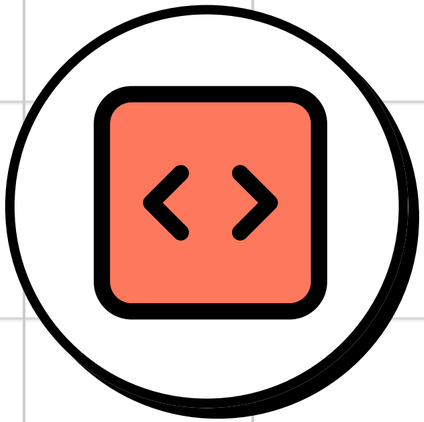
Big Data Pipeline



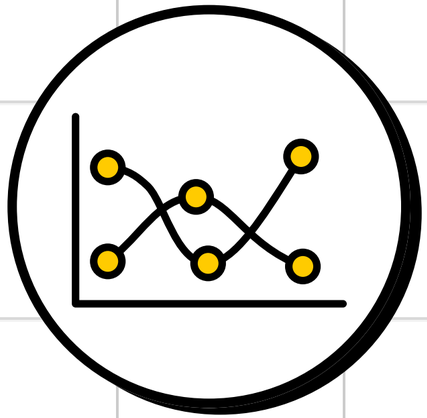
Messy Data



Data Cleaning

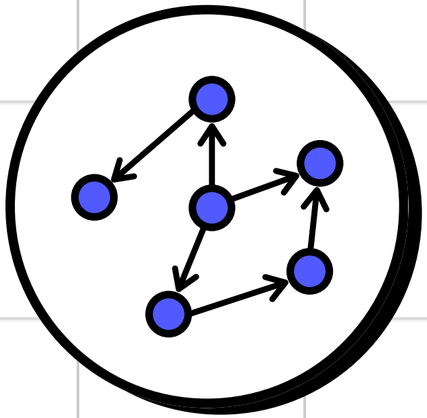


Analysis

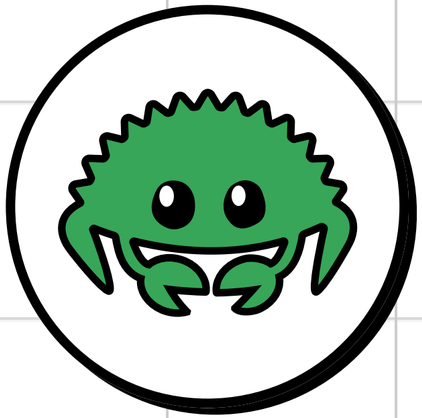


Results

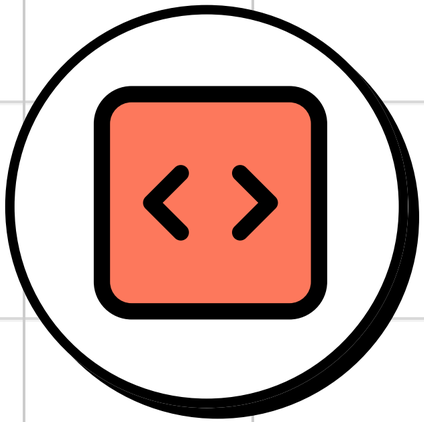
Big Data Pipeline



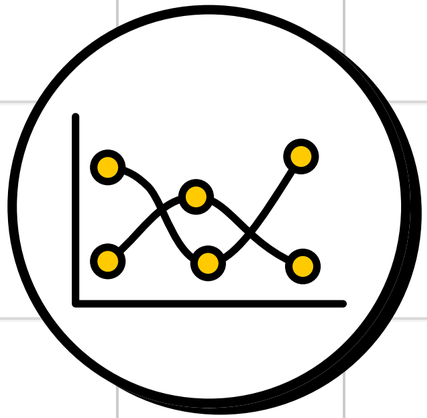
Load GP Records



Data Cleaning

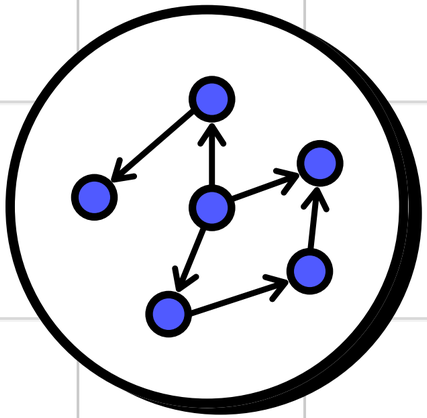


R or Stata or Python

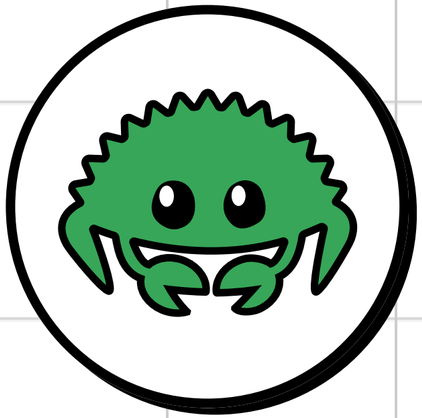


Results & published code

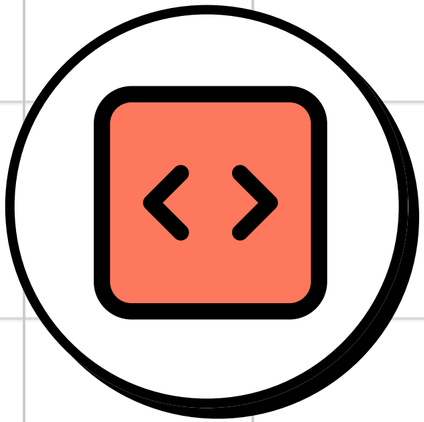
Big Data Pipeline



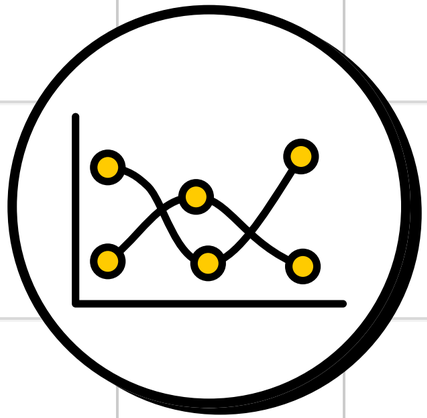
Load GP Records



Data Cleaning

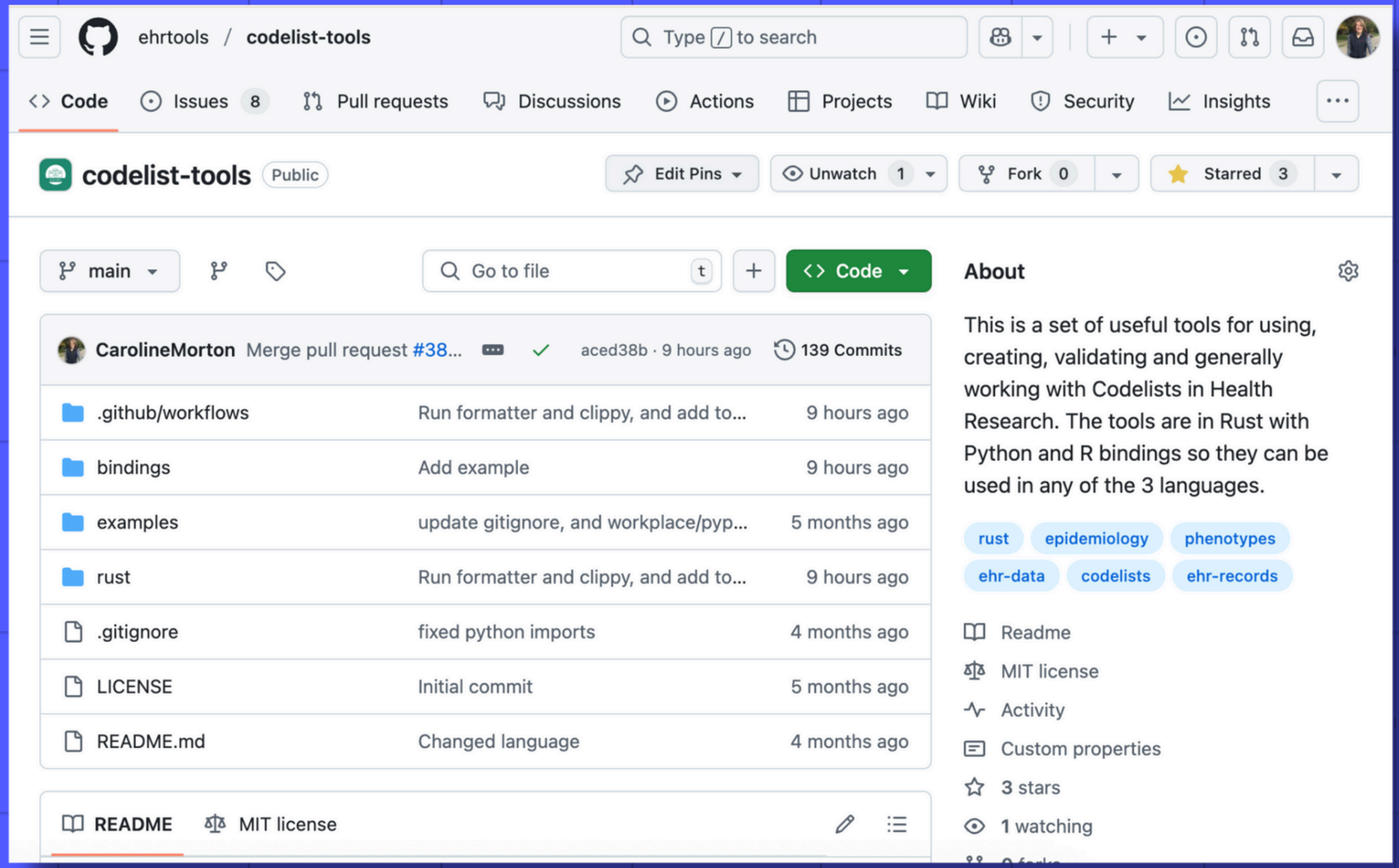


R or Stata or Python



Results & published code

Codelist-Tools



What are coded events?

Imagine this scenario

COUGH

Few days of feeling unwell.

Visit your GP.

VISITING YOUR GP

When visiting your general practitioner, you may see them writing notes into their computer; this turns into coding events for future research and service monitoring.

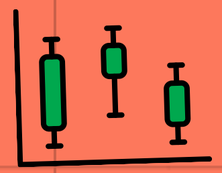
Your appointment becomes this

SNOMED Code	SNOMED Term	Measurement
49727002	Cough (finding)	
271649006	Systolic blood pressure (observable entity)	110
271650006	Diastolic blood pressure (observable entity)	75

Codes represent your appointment

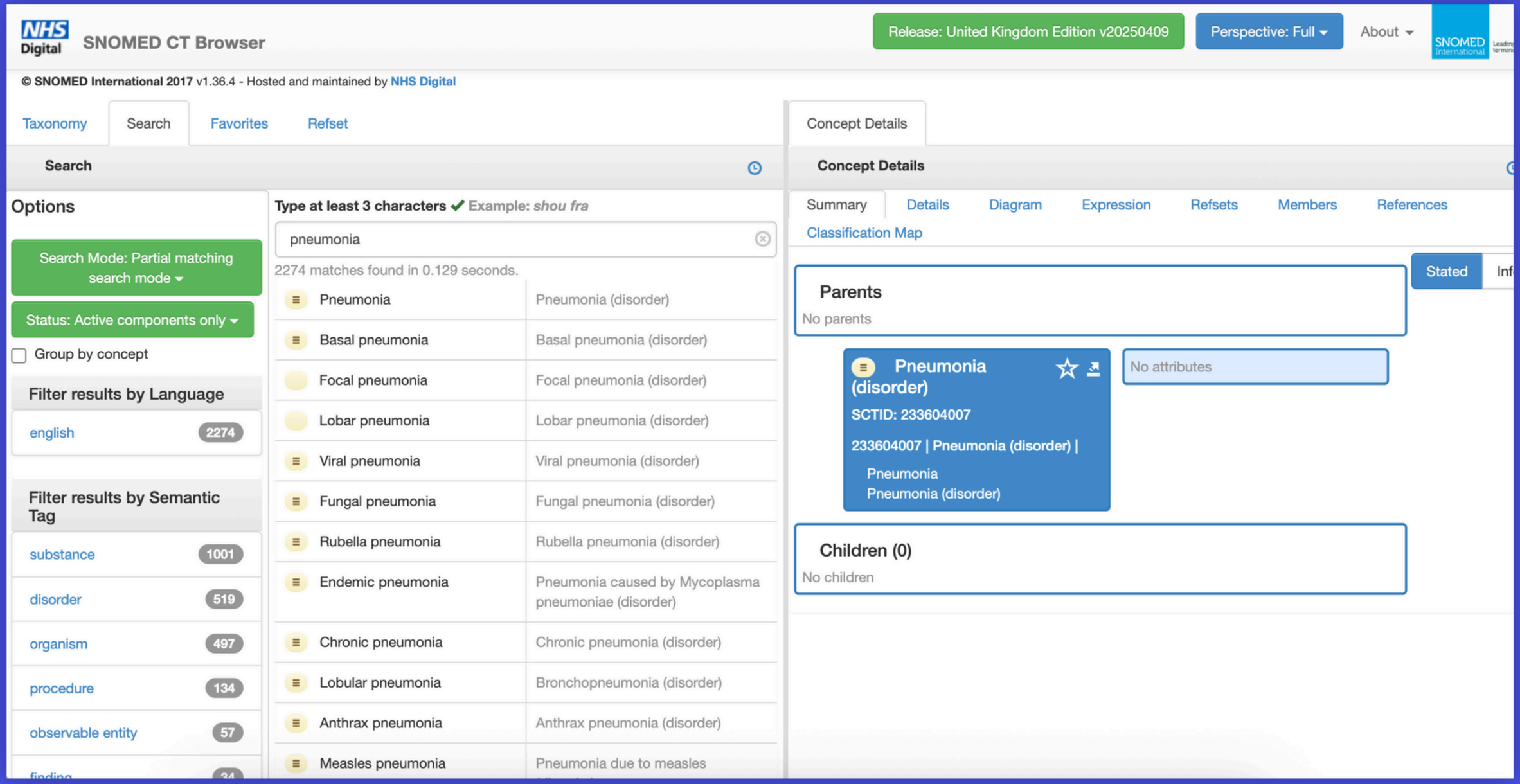
SNOMED Code	SNOMED Term	Measurement
49727002	Cough (finding)	
271649006	Systolic blood pressure (observable entity)	110
271650006	Diastolic blood pressure (observable entity)	75
386661006	Fever (finding)	
233604007	Pneumonia (disorder)	
372687004	Amoxicillin (substance)	

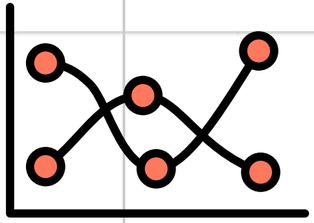
Codes are in a tree



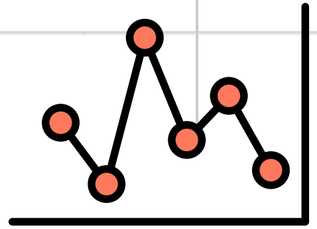
- Clinical Finding
  - Respiratory Symptom
    - Cough
      - Chesty Cough
      - Allergic Cough
  - Cardiovascular Disorder
    - Hypertensive Disorder

But there are 10,000s of codes


NHS Digital
SNOMED CT Browser
Release: United Kingdom Edition v20250409
Perspective: Full
About
SNOMED International
© SNOMED International 2017 v1.36.4 - Hosted and maintained by NHS Digital
Taxonomy
Search
Favorites
Refset
Concept Details
Search
Concept Details
Options
Search Mode: Partial matching search mode
Status: Active components only
Group by concept
Filter results by Language
english 2274
Filter results by Semantic Tag
substance 1001
disorder 519
organism 497
procedure 134
observable entity 57
finding 34
Type at least 3 characters ✔ Example: shou fra
pneumonia
2274 matches found in 0.129 seconds.
Pneumonia | Pneumonia (disorder)
Basal pneumonia | Basal pneumonia (disorder)
Focal pneumonia | Focal pneumonia (disorder)
Lobar pneumonia | Lobar pneumonia (disorder)
Viral pneumonia | Viral pneumonia (disorder)
Fungal pneumonia | Fungal pneumonia (disorder)
Rubella pneumonia | Rubella pneumonia (disorder)
Endemic pneumonia | Pneumonia caused by Mycoplasma pneumoniae (disorder)
Chronic pneumonia | Chronic pneumonia (disorder)
Lobular pneumonia | Bronchopneumonia (disorder)
Anthrax pneumonia | Anthrax pneumonia (disorder)
Measles pneumonia | Pneumonia due to measles
Summary
Details
Diagram
Expression
Refsets
Members
References
Classification Map
Stated
Parents
No parents
Pneumonia (disorder)
SCTID: 233604007
233604007 | Pneumonia (disorder) |
Pneumonia
Pneumonia (disorder)
No attributes
Children (0)
No children



What I’m interested in

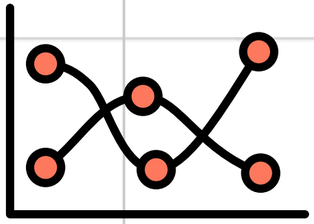


Codelists

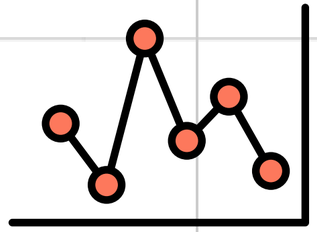
- Clinical data contains SNOMED or ICD 10 codes
- 45 different codes for “Cough”
- Researchers must decide:
  - **What codes count as a disease state?**
  - **What codes to exclude?**

Challenges

- Challenge is not only selecting codes but also:
- Managing them over time
- Sharing them across teams
- Recording decisions for transparency and reproducibility



Structure

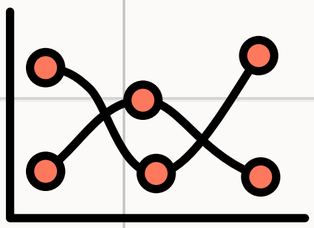


Rust Base Layer
Small well-tested packages that do one thing well

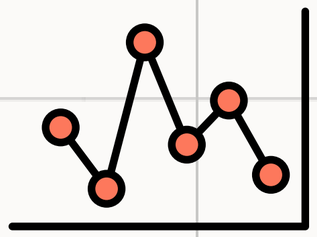
Bindings

Python Package
Callable in scripts

R Package
Callable in scripts



Codelist Management



Core Structure: CodeList
Central data structure representing a medical codelist.

Holds

- Entries (codes + terms)
- Type (e.g. ICD10, SNOMED)
- Metadata (provenance, usage, validation)
- Logs and custom options

Key Features

- Add/remove codes with validation
- Export to CSV and JSON
- Fully test-covered: duplicate handling, metadata integrity, reproducibility

```rust
let mut cl = CodeList::new("Sepsis", CodeListType::ICD10, metadata, None);
cl.add_entry("R65.2", "Severe sepsis", None)?;
cl.save_to_csv("sepsis.csv")?;
```

Codelist Management

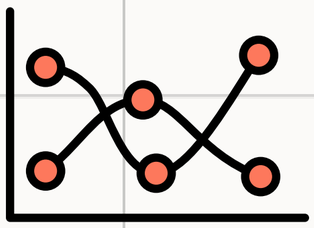
```rust
    let mut contributors = HashSet::new();
    contributors.insert("Dr. Caroline Morton".to_string());
    let provenance = Provenance::new(Source::ManuallyCreated, Some(contributors));

    let mut categorisation = CategorisationAndUsage::new(None, None, Some("ODC-BY-1.0".to_string()));
    categorisation.add_usage("Epidemiological study on sepsis".to_string());

    let context = PurposeAndContext::new(
        Some("Case identification".to_string()),
        Some("Clinical researchers".to_string()),
        Some("UK hospital data".to_string()),
    );

    let mut validation = ValidationAndReview::new(None, None, None, None, None);
    validation.add_reviewer("Prof. Jane Doe".to_string());
    validation.add_validation_notes("Reviewed by clinical team".to_string())?;

    let metadata = Metadata {
        provenance,
        categorisation_and_usage: categorisation,
        purpose_and_context: context,
        validation_and_review: validation,
    };

    let mut cl = CodeList::new("Sepsis".to_string(), CodeListType::ICD10, metadata,
Some(codelist_opts));
    cl.add_entry("R65.2".to_string(), "Severe sepsis".to_string(), None)?;
    cl.add_entry("A41.9".to_string(), "Sepsis, unspecified".to_string(), Some("Common generic
code".to_string()))?;

    // Step 7: Save it
    cl.save_to_csv("sepsis_codelist.csv")?;
```

Python Bindings

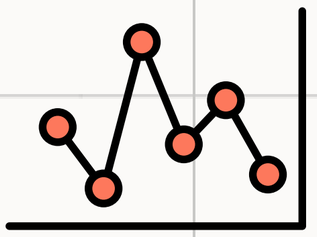
```rust
// Internal imports
use codelist_rs::codelist::CodeList;
use codelist_rs::codelist_options::CodeListOptions;
use codelist_rs::metadata::{
    CategorisationAndUsage, Metadata, Provenance, PurposeAndContext, Source, ValidationAndReview,
};
use codelist_rs::types::CodeListType;
use codelist_validator_rs::validator::Validator;

/// Python wrapper for the CodeList struct
///
/// This struct is a python wrapper for the CodeList struct in the codelist-rs library.
/// It allows us to create a new CodeList object from python and interact with it.
#[pyclass(name = "CodeList")]
pub struct PyCodeList {
    pub inner: CodeList,
}

/// Python methods for the PyCodeList struct
#[pymethods]
impl PyCodeList {
    #[new]
    #[pyo3(signature = (name, codelist_type, source, authors=None))]
    fn new(
        name: String,
        codelist_type: &str,
        source: &str,
        authors: Option<Vec<String>>,
    ) -> PyResult<Self> {
```

```python
from codelists_rs.codelist import CodeList

# Create a new codelist object
c = CodeList("Pneumonia", "ICD10", "Manually created")

# Add a code to the list
c.add_entry("A119", "pneumonia")

# Add a reviewer
c.add_contributor("John Doe")
c.add_usage("Identify pneumonia in patients")
c.add_tag("pneumonia")
c.add_validation_info(
    reviewer="Jane Smith",
    status="in review",
    notes="Looking good"
)

entries = c.entries()
print(f"Codelist entries for {c.name}:")
for code, label in entries:
    print(f"{code}: {label}")
print(f"Validation Notes: {c.get_validation_notes()}")
```

```
Codelist entries for Pneumonia:
A119: pneumonia
Validation Notes: Looking good
```



Codelist Validation



Check codes are even real

Codes have a certain format.

ICD10 Rules

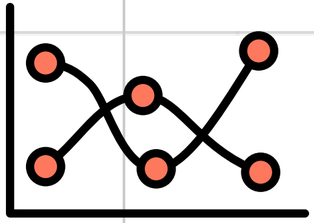
- The code must be 7 characters or less
- The first character must be a letter
- The second and third characters must be numbers
- The fourth character must be a dot, or a number or X
- If the fourth character is a dot, there must be at least 1 number after the dot
- If the fourth character is a X, there are no further characters
- The fifth to seventh characters must be numbers if present

```rust
use crate::{errors::CodeListValidatorError, validator::CodeValidator};

pub struct IcdValidator<'a>(pub &'a CodeList);

static REGEX: LazyLock<Regex> = LazyLock::new(|| {
    Regex::new(r"^[A-Z]\d{2}(X|(\.\d{1,3})?|\d{1,4})?$").expect("Unable to create regex")
});

impl CodeValidator for IcdValidator<'_> {
    fn validate_code(&self, code: &str) -> Result<(), CodeListValidatorError> {
        ...

    fn validate_all_code(&self) -> Result<(), CodeListValidatorError> {
        ...
        }
    }
}
```

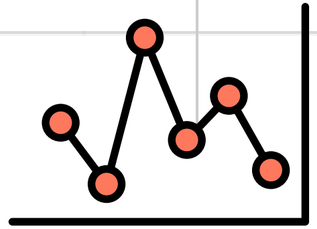
Synthetic Data Generation

Why do we need Synthetic Data?

- **Enable open science:** Release fake data with code - make research reproducible without compromising privacy
- **Accelerate research:** Skip months of data access approvals
- **Foster collaboration:** Share datasets across institutions without complex governance barriers
- **Scale quickly:** Generate large, realistic datasets for testing when real data is scarce or help med tech companies test their products without the data



What do we want from our synthetic data?



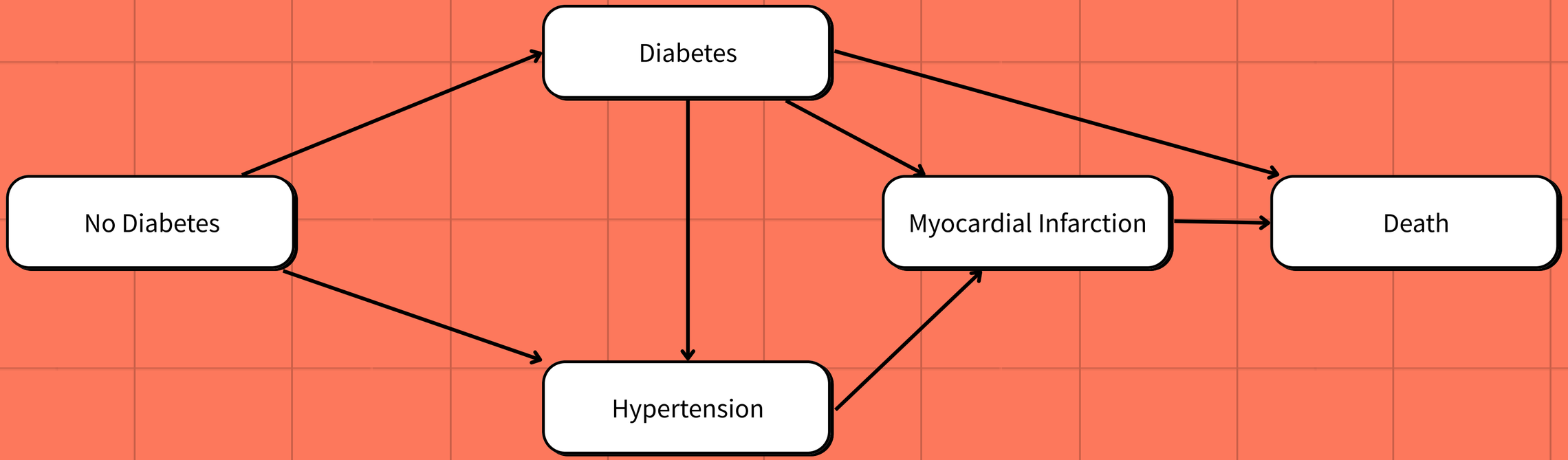
Wishlist

- **Realistic complexity:**
  - Capture causal relationships (DAG structure)
  - Temporal disease progression
  - Missing data patterns
- **Privacy guarantee**: Zero risk of re-identification—no real patient information leaked

Challenges

- **Scale**: Generating millions of rows for realistic
- **Statistical reproducibility:** Same configuration should produce statistically consistent datasets
- **Temporal constraints:** Must enforce logical ordering - 2nd COVID vaccine after 1st, diagnosis before treatment, death ends all events

Directed Acyclic Graphs



DAG Problem

- **Challenge:** Must preserve both statistical properties AND temporal logic
  - Right prevalence at each state
  - Right transition probabilities between states
- **Scale**: 50,000 patients × multiple events each = millions of rows to generate efficiently
- **Why Rust:** Patients are independent - perfect for parallel generation. Async/multi-threading gives massive speedup

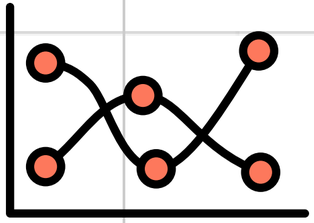
Forward Simulation

- **Approach**: Markov transition matrices - sample states step-by-step through time
- **Problem**: Timing constraints broke prevalence targets
- Forcing age constraints prevented multi-step pathways from completing
- Ended with too many "None" states, too few deaths
- **Rust advantage:** Fast iteration - could test hundreds of parameter combinations in minutes

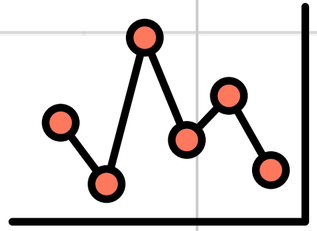
Backward simulation

- Two-phase approach:
  - **Sample the path:** Use DAG as distribution to pick final outcome
  - **Sample the timing:** Work backwards from end state, sampling ages for each transition
- Why this works: Decouples "what happens" from "when it happens"
  - Guarantees correct prevalence & temporal logic
- Rust generate 50K patients in seconds rather than minutes with Python

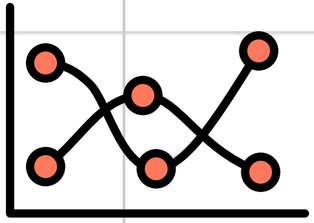
Common Uses of Rust



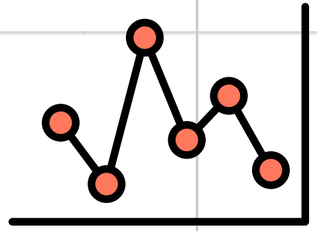
Common Use Cases



- Binding for other languages like R, Python
  - Py03, extendR
- High performance computing
- Reproducibility and correctness
- Simulation and visual modeling



Talk to your neighbour



Which of these resonates most with YOUR work or interests?

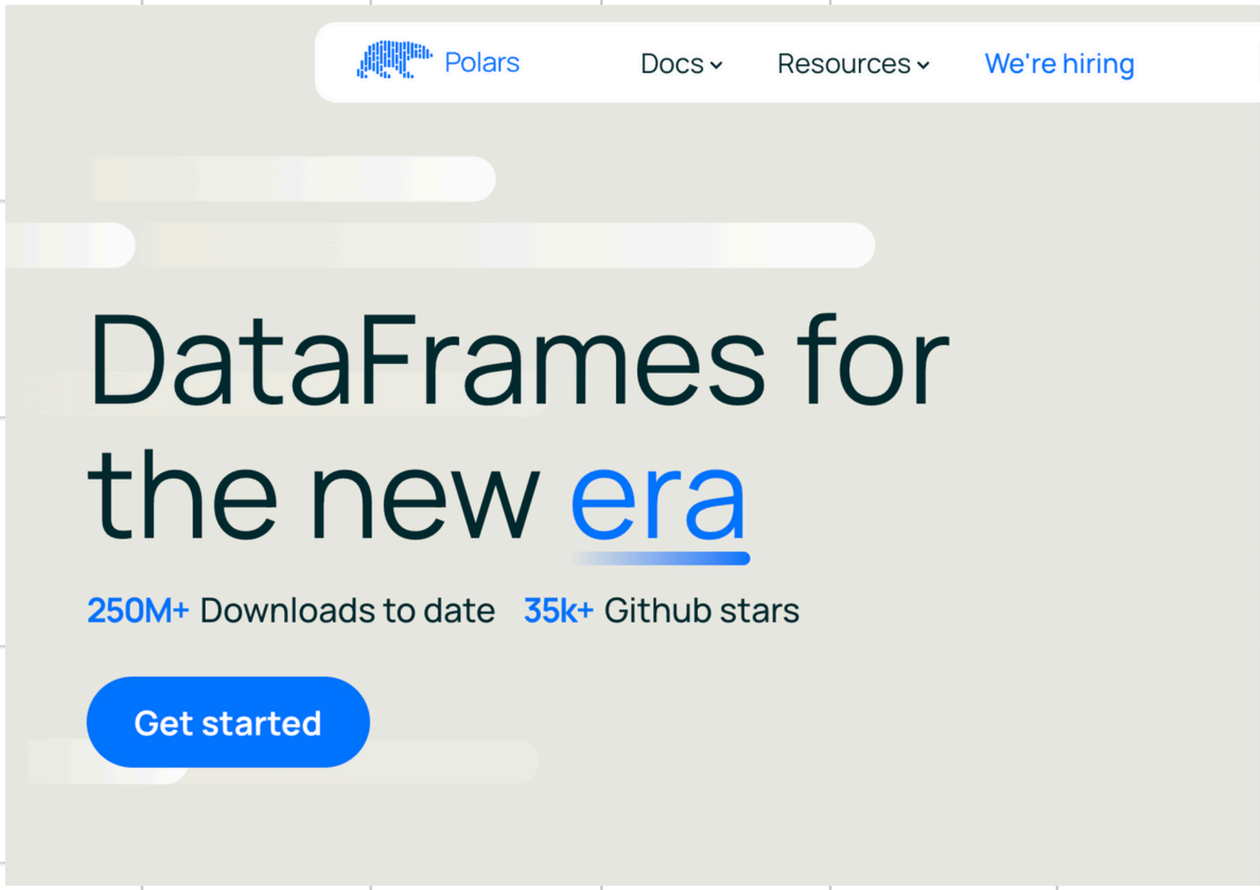
- Binding for other languages like R, Python
  - Py03, extendR
- High performance computing
- Reproducibility and correctness
- Simulation and visual modeling

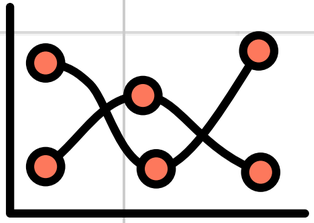
Amazing Rusty Tools

Polars

Dataframe library

- Open-source library for data manipulation using data frames
- Implemented in Rust but callable from other languages like Python
  - User might not even know it is using Rust under the hood
  - API similar to Pandas





ExtendR

R packages

- R Extendr allows you to write fast rust packages that are callable in R
- Meeting researchers in the environment and language that they are using
- Very friendly community

extendr Get Started Documentation Blog

extendr - extending R with Rust

Tests failing | crates.io v0.8.1 | docs passing | License MIT

Build blazingly fast R packages 📦 with ease. The extendr ecosystem is **developer-friendly** and **CRAN**-ready. extendr let's you write ergonomic and idomatic Rust without having to worry about R's internals (much 😅).

Getting Started

- Don't have Rust installed yet? Start here.
- Familiar with Rust and want to get your hands dirty? Follow along with a complete example.

The ecosystem

Rust crates 🦀:

- `extendr-api` —ergonomic, opinionated, and safe interface between R and Rust
- `extendr-engine` —embed and use R in Rust
- `extendr-ffi` —hand-crafted bindings to R's C-API

Linear Algebra

Faer

- Project from Sarah Quiñones El Kazdadi, software dev/phd student at Inria
    - who describes that write linear algebra code in exchange for the praise of strangers on the internet!

faer is a modern linear algebra library for the rust programming language

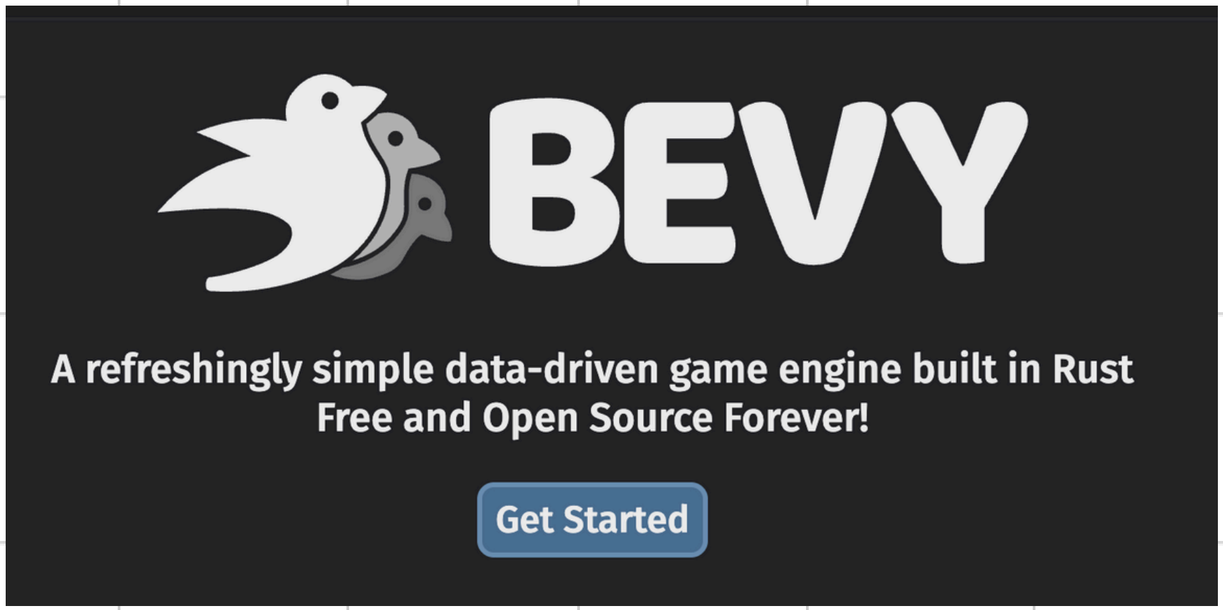
get started

MIT license

Bevy

Simulation

- Bevy is a game engine built in rust
  - But used a lot in simulations, not just gaming!
- Uses data to drive the construction which makes it perfect for scientific simulation
- 2D and 3D simulations
- Interactive graphing



PetGraph

Graphing

- Graph data structure library

petgraph

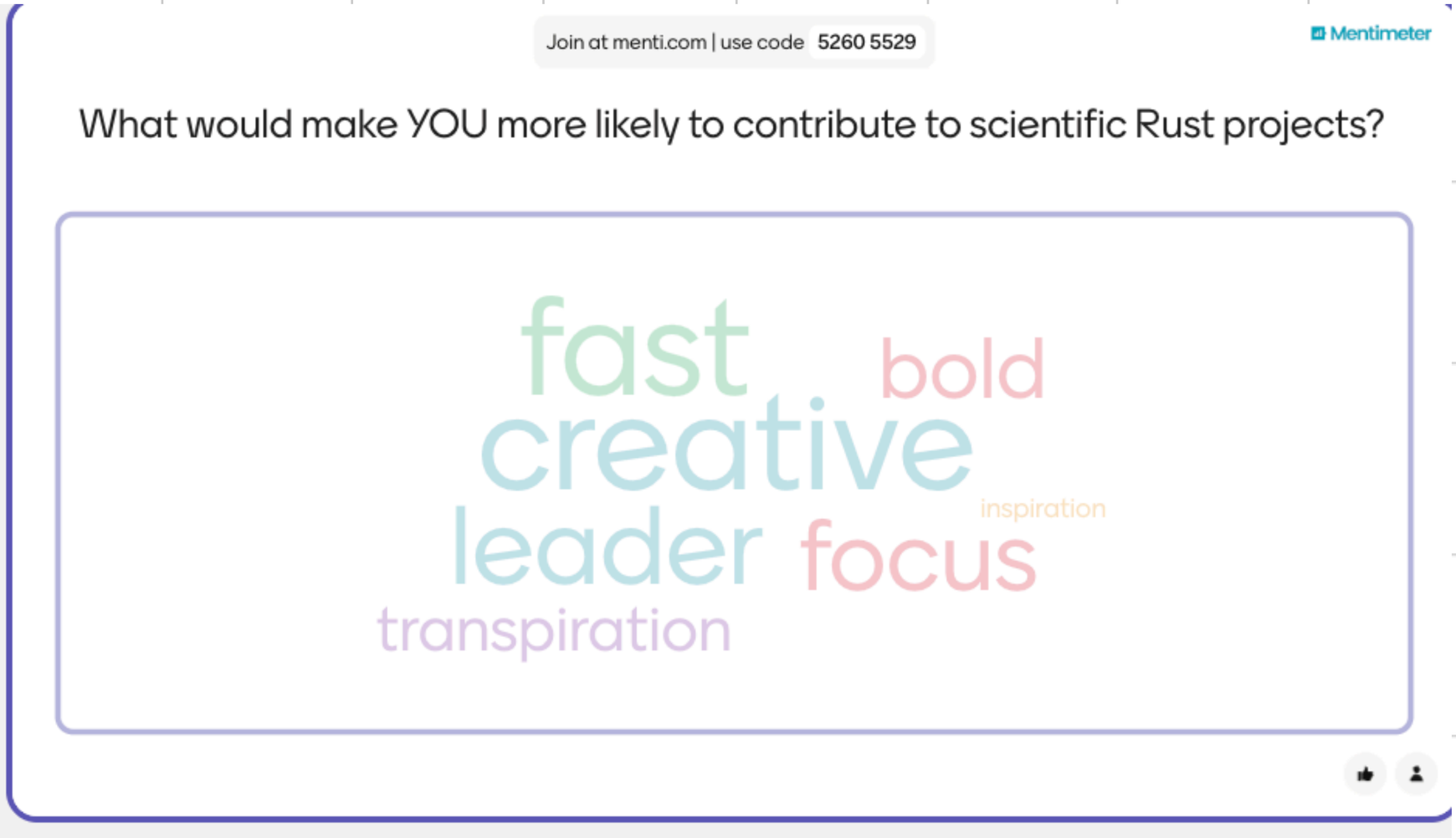
Petgraph provides fast, flexible graph data structures and algorithms in Rust. Supp
both directed and undirected graphs with arbitrary node and edge data. It comes v

- **Multiple Graph Types**: Graph, StableGraph, GraphMap, and MatrixGraph to su
use cases.
- **Algorithms Included & Extensible**: For tasks like path-finding, minimum spani
trees, graph isomorphisms, and more - with traits exposed for implementing c
algorithms.
- **Graph Visualization support**: Export/import graphs to/from DOT format for
visualization with Graphviz.

Supports Rust 1.64 and later. This will only change on major releases.

crates.io v0.8.3 | docs passing | rustc 1.64+ | chat 5 online | Continuous integration passing

Quiz Time



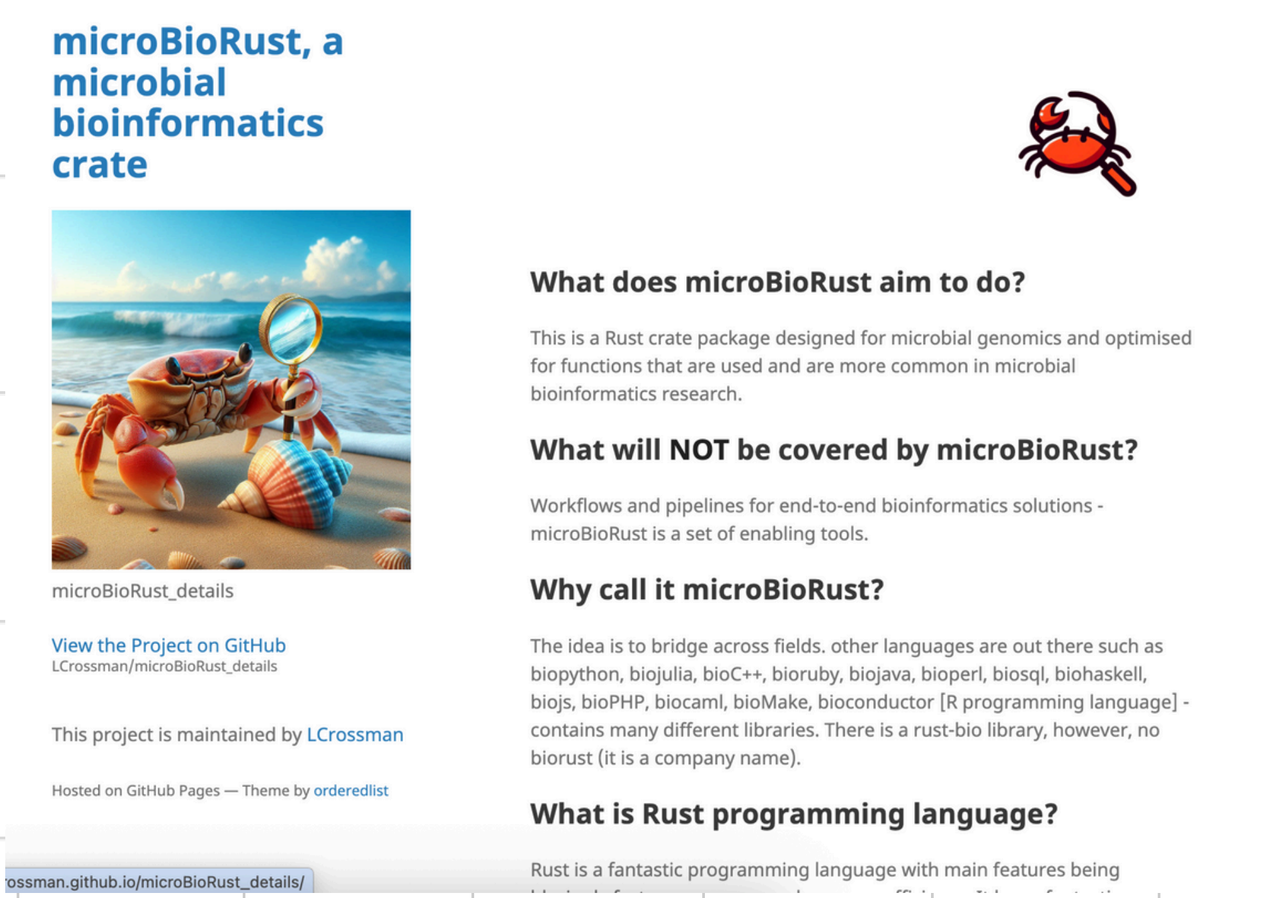
5260 5529

Rust Science Projects

MicroBioRust

Aims

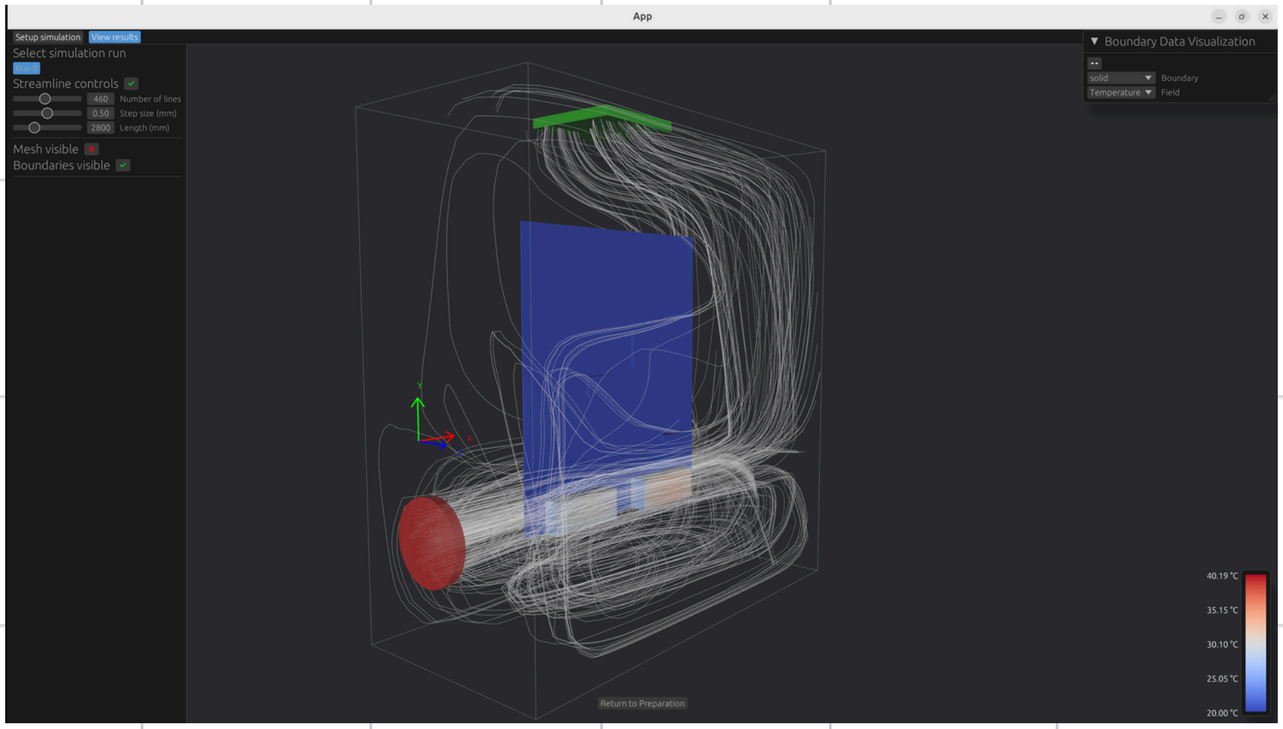
- MicroBioRust crate - a high-performance, modular bioinformatics toolkit written in Rust
  - Fast parsers for GenBank and EMBL formats
  - Output to GFF, FAA, and FFN
- microbiorust-py - Python bindings



Computer Chip Cooling

Vanellus

- Developed a thermal analysis solution for electronics/PCBs using a combination of Python and Rust
  - Implemented mesh generation in Rust
  - Well-suited for file parsing of specialized scientific formats
- Uses Bevy
- Python/Rust interop with Py03 and calling Python code from rust server



Computational Neuroscience

Modelling neural networks

- Neural networks in brains and computers use back propagation
  - Machines use same network to go back/forward
  - Brains do not!
- Using Bevy to visualise 3 dimensional data across time
  - Simulate populations of neurons clustered together. How info flows between them.

Questions

Open Science Principles

What we want from great tools?

- **Reproducibility is foundational:** Research pipelines must produce identical results across runs, teams, and datasets - the same input always yields the same output
- Code must be **trustworthy:**
  - Errors surface **clearly**, not silently
  - Logic is **explicit and auditable**
  - Results can be **believed** by scientists, reviewers, and policymakers
- Documentation speaks to both domain experts and the broader research community

Why Code Sharing Matters

- **Reproducibility**: Other researchers can verify findings and replicate analyses
- **Transparency**: Methods become auditable - reviewers and readers can see exactly what was done, not just what was reported
- **Accelerates science:** Build on existing work rather than reinventing the wheel
- **Trust**: Open code demonstrates confidence in methods and invites scrutiny

Editorials

Mandatory data and code sharing for research published by *The BMJ*

BMJ 2024 ; 384 doi: https://doi.org/10.1136/bmj.q324 (Published 05 March 2024)
Cite this as: *BMJ* 2024;384:q324

Article | Related content | Metrics | Responses

Elizabeth Loder, head of research, Helen Macdonald, publication ethics and integrity editor, Theodora Bloom, executive editor, Kamran Abbasi, editor in chief

Author affiliations

Correspondence to: E Loder eloder@bmj.com

New policy requires authors to share analytic codes from all studies and data from all trials

The case for sharing data from clinical research is strong.[1] [2] Clinical study data include all information collected during a study, which is then analysed using computer codes to generate results. Unimpeded access to both data and code across the research community maximises the value of each research project. It shows respect for the efforts of research participants and the economic contributions of the public. It enables data scrutiny and re-analyses, which are essential for the self-correcting activities that contribute to good science and better patient and population outcomes.

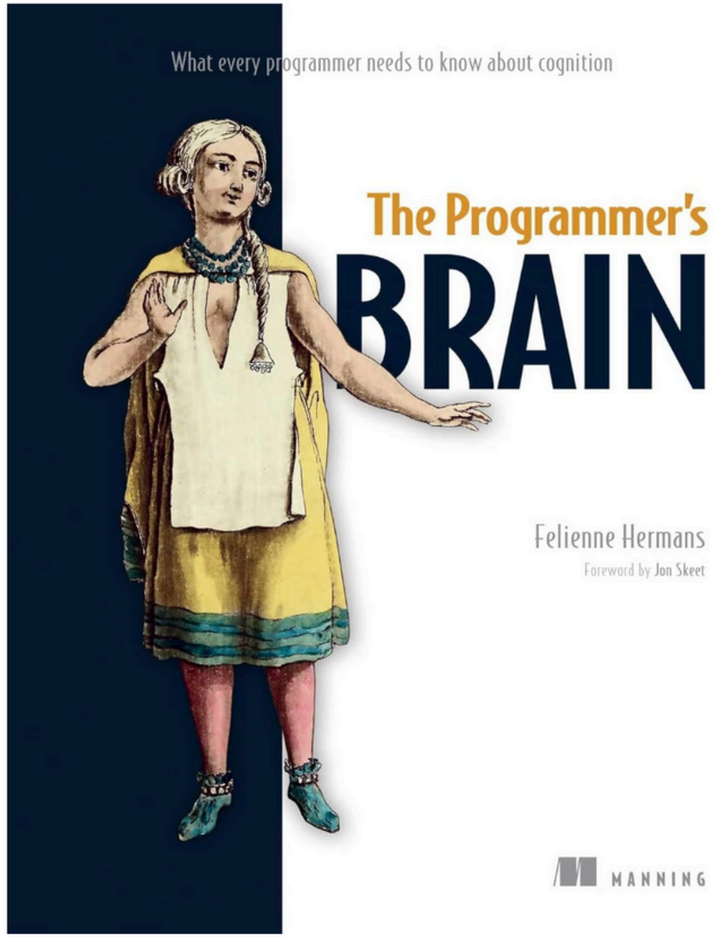
Code sharing, in particular, makes it possible to evaluate analytical decisions that cannot easily be described in the methods section of a paper but have an important effect on results. It is plausible, for example, that requirements for code sharing might have deterred submission, or prevented publication of, the fraudulent Surgisphere papers published during the pandemic.[3] Although there are practical challenges and potential harms associated with data

How to make Rust tools usable for non-devs

- Reduce cognitive Load
- Good documentation

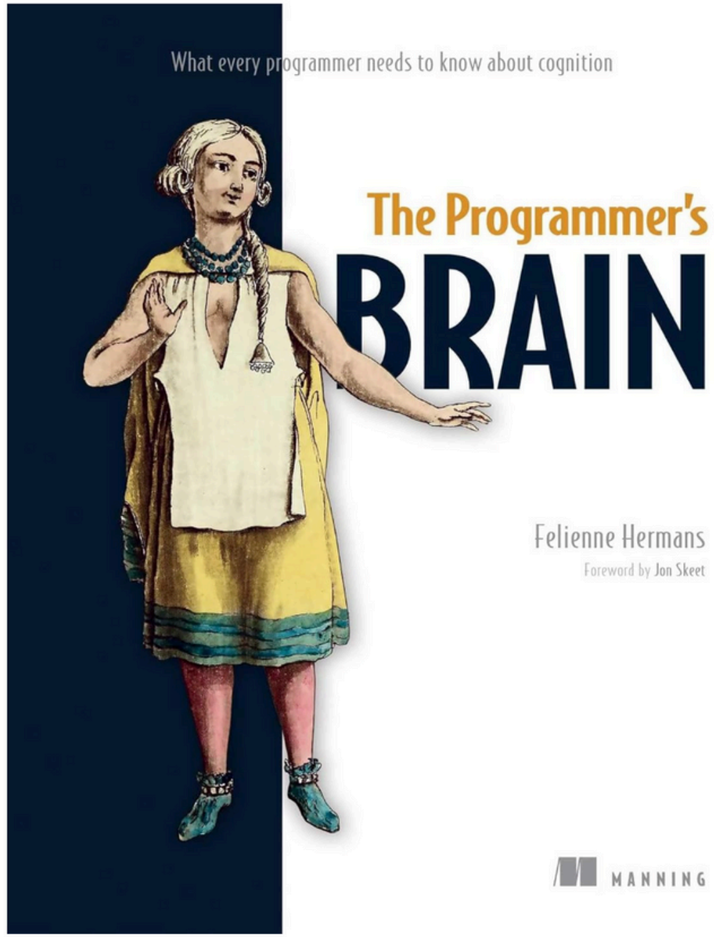
Reducing Cognitive Load

- **Code Comments**
- **Minimise working memory demands:**
  - clear function names
  - consistent patterns
  - avoid deep nesting
- **Beacons**
  - Like Node, Adaptors



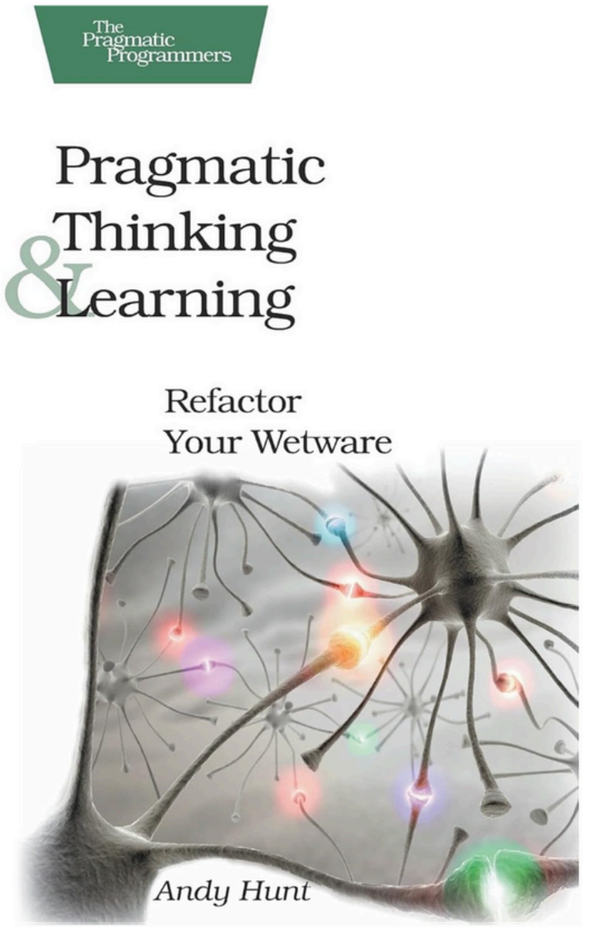
Reducing Cognitive Load

- **Leverage long-term memory:**
  - use domain terminology researchers already know (e.g., "prevalence" not "final_state_probability")
- **Design Patterns**
  - Activate chunking



Reducing Cognitive Load

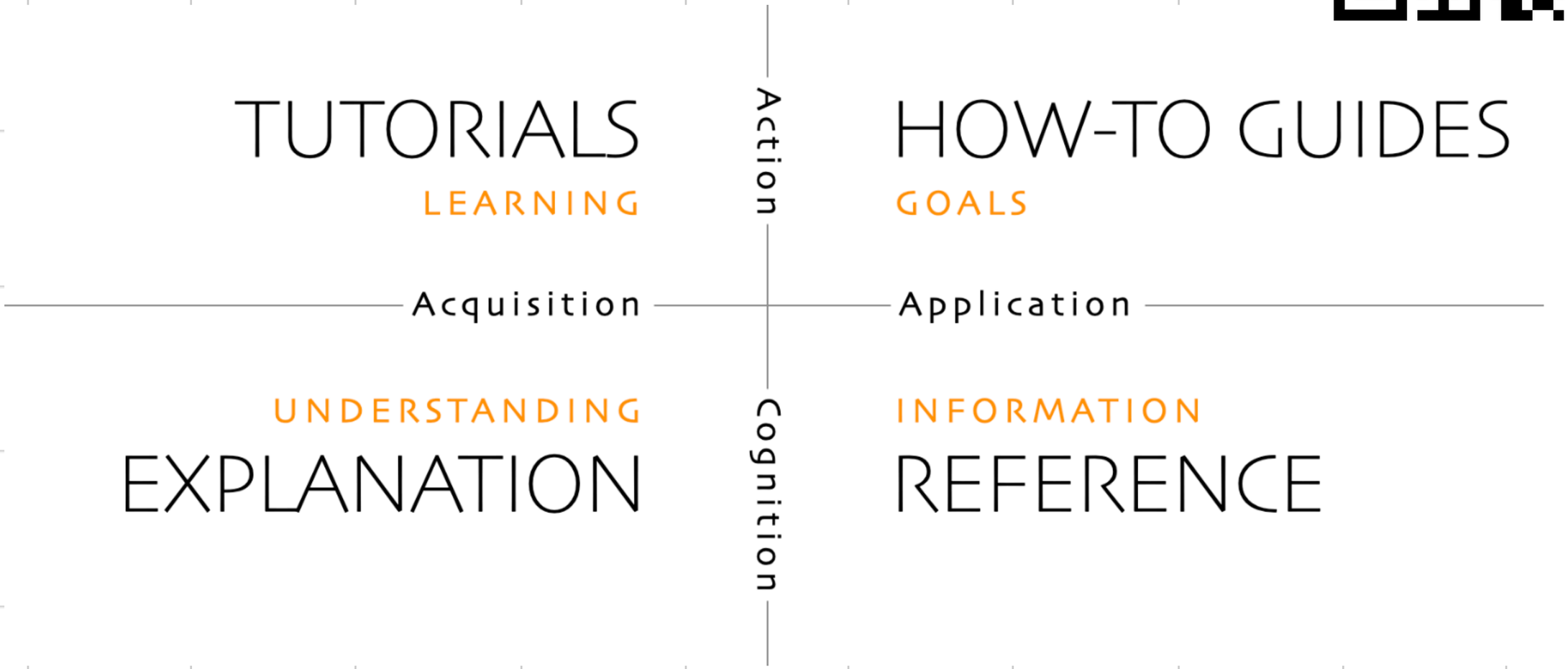
- **Type system as documentation:**
  - Rich types that make invalid states unrepresentable (e.g., Age type that can't be negative)
- **Error messages that teach:**
  - Not just "invalid input" but "expected age between 0-110, got -5"
- **Sensible defaults:** Minimize configuration burden for common use case



How to make Rust code readable for non-devs

- **Clear documentation:** Explain what the code does and why, not just how - write for domain experts, not just Rustaceans
- **Familiar interfaces:** Provide R/Python bindings so scientists use the tool in their native language
- **Examples over syntax:** Show working examples with real data - let researchers learn by doing, not by reading Rust docs
- **Explicit over clever:** Prioritize readable code over idiomatic Rust- avoid advanced features when simpler patterns work

How to make Rust code readable for non-devs



What can you do?

Join any of the projects mentioned

Speak to Scientists

Scientific Computing in Rust Conf

- Attend this conference
- Watch youtube channel

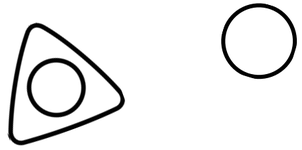
Scientific Computing in Rust

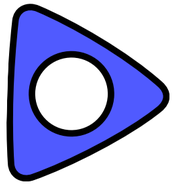
Welcome to Scientific Computing in Rust. On this website, you'll find information about the Scientific Computing in Rust annual workshop and monthly newsletter.

Annual workshop

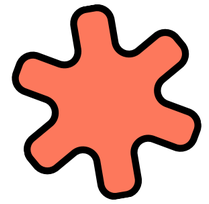
The Scientific Computing in Rust annual workshop is a virtual event focussing on the use of the Rust programming language within scientific computing. Topics covered in past workshops have included linear algebra libraries, running code on GPUs, and a huge range of application-specific libraries. It was first held in 2023. Recordings of the talks for every workshop are available on the Scientific Computing in Rust YouTube channel.

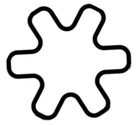
Information about the most recent 2025 workshop can be found on the Scientific Computing in Rust 2025 workshop page.

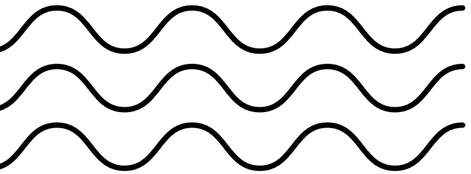




Let’s build RustinScience.com







Questions

Explore more

- GitHub Repo
- Add me on Linkedin
- Get involved in building rustinscience.com
- Rust in Scientific Computing conference

Rust for Good: Systems Programming in Science and Public Health

Dr. Caroline Morton

Software Engineer, Epidemiologist and Former GP