A different type of DAG: Data pipelines principles for epidemiology

Emily Riederer

Data Science Lead at Capital One

Disclosures

I am an employee of Capital One Financial Corporation

I have no relevant financial relationships to disclose

Industry data science can learn a lot from epidemiology

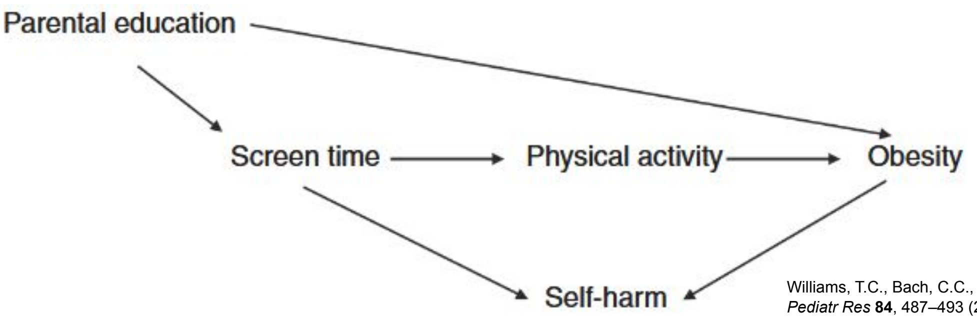
Epi

Data	Analytical Approach	Precise Communication
Imperfection expected	Prototypical study designs	Estimands
Data Management	Packaged Code	Deployed Outputs

DS

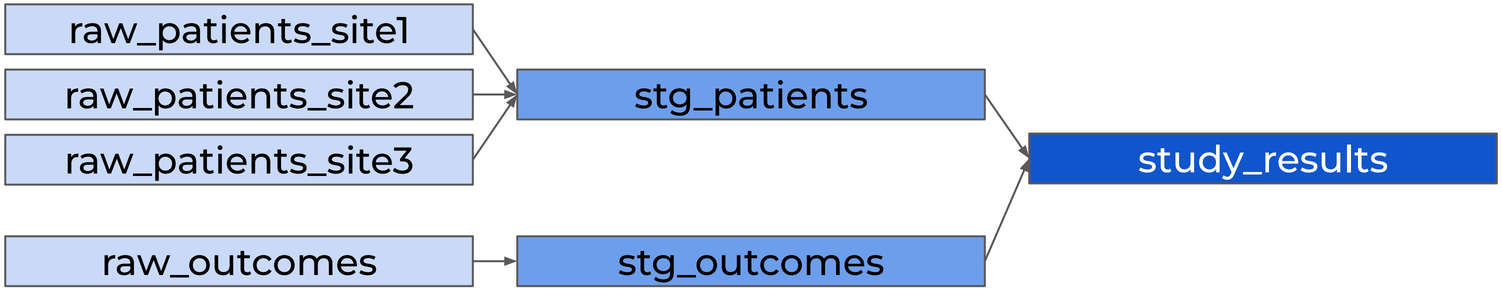
Data pipelines are a different sort of DAG that encode the computational versus conceptual data generating process

Epi



Williams, T.C., Bach, C.C., Matthiesen, N.B. *et al.* Directed acyclic graphs: a tool for causal studies in paediatrics. *Pediatr Res* **84**, 487–493 (2018). https://doi.org/10.1038/s41390-018-0071-3

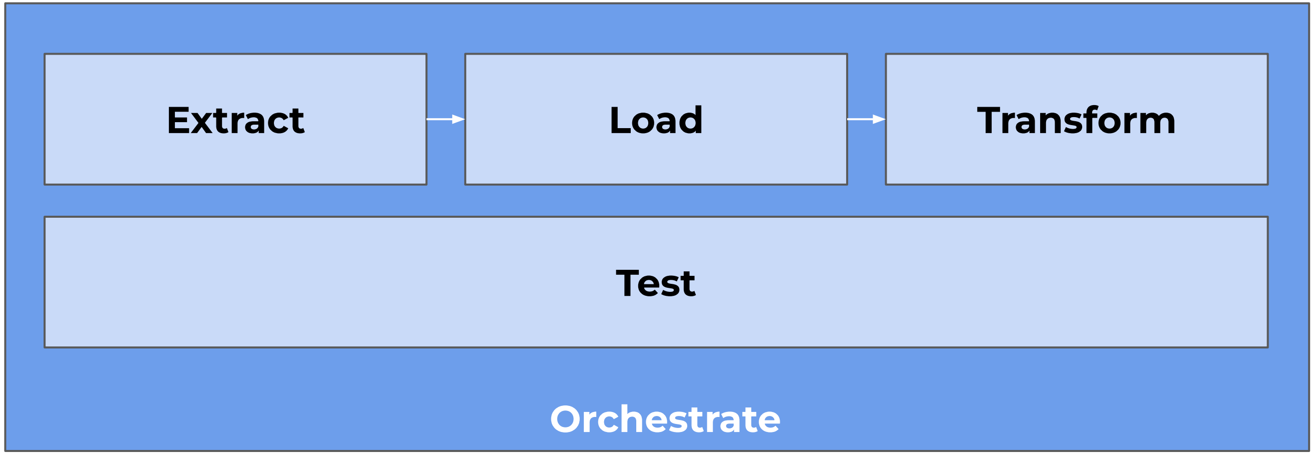
DS



Data pipelines and databases serve many needs in industry with parallels for individual analytical projects

Data Pipelines	Project Benefit
Data lineage and quality control	Reproducibility
New data ingestion, integration, and updates	Harmonizing disparate sources
Sharing data artifacts across many users	Collaboration
Sharing data artifacts across many use cases	Reuse across projects

Today, we will discuss some useful tools and ideas throughout the data pipeline ecosystem

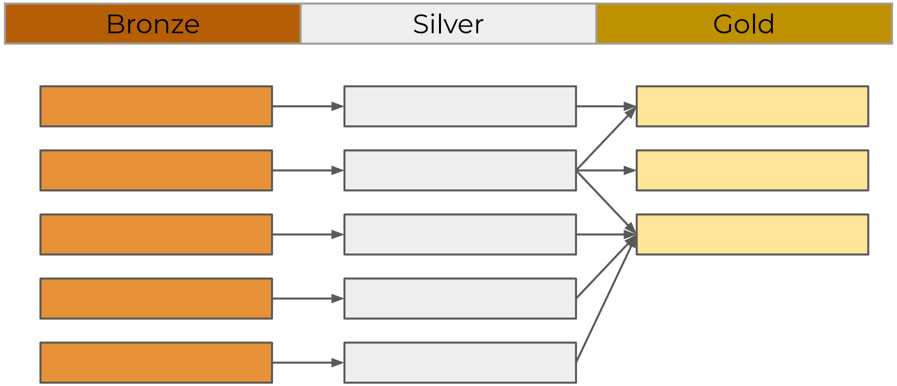


The same principles apply across platforms

	Local	Shared	Cloud
Storage	📁 Local Drive	📁 Network Drive 🗄 Google Cloud Storage 🗄 Amazon S3	🗄 Google BigQuery
Compute	🗄 duckdb + dbt R, python, or any scripting language		🗄 Google BigQuery SQL + dbt (data pipeline) Dagster (orchestration)

ARCHITECTURE: Pipelines preserve data artifacts at raw, standardized, and transformed stages

Medallion Architecture



Bronze (= Raw)
- Unmodified source data

Silver (= Structure)
- Standardized file structure
- Cleaned column name
- Casted data types
- Light re-encoding

Gold (= Prepared)
- Transformed and merged data
- Derived fields
- Ready for analytical use

ARCHITECTURE: Preserving artifacts at multiple stages balances reproducibility with analytical efficiency

In Database Schema

- 🗏 `data.raw`
- 🗏 `data.structured`
- 🗏 `data.clean`

In File System

- 📁 `data/raw/`
- 📁 `data/structured/`
- 📁 `data/clean/`

Raw

- Preserves full reproducibility
- Enables testing for upstream errors pre-transformation

Structured

- Preserves data flexible enough to transform in different ways

Prepared

- Ready for use without rerunning expensive computations
- Ensures collaborators made same design decisions

ARCHITECTURE: Pure reading and writing is what makes pipeline acyclic and idempotent, resilient to re-runs

```
df = readr::read_csv(‘data/raw/patients.csv’)

df_cleaned = << transformations >>

readr::write_csv(df_cleaned, ‘data/structured/patients.csv’)
```

```
df = readr::read_csv(‘data/raw/patients.csv’)

df = << transformations >>

readr::write_csv(df, ‘data/raw/patients.csv’)
```

```
df = readr::read_csv(‘data/raw/patients.csv’)

df_cleaned = << transformations >>

readr::write_csv(df_cleaned, ‘data/raw/patients.csv’)
```

ELT
TO

EXTRACT: Many different techniques can be used to collect data across disparate and sometimes inconvenient sources

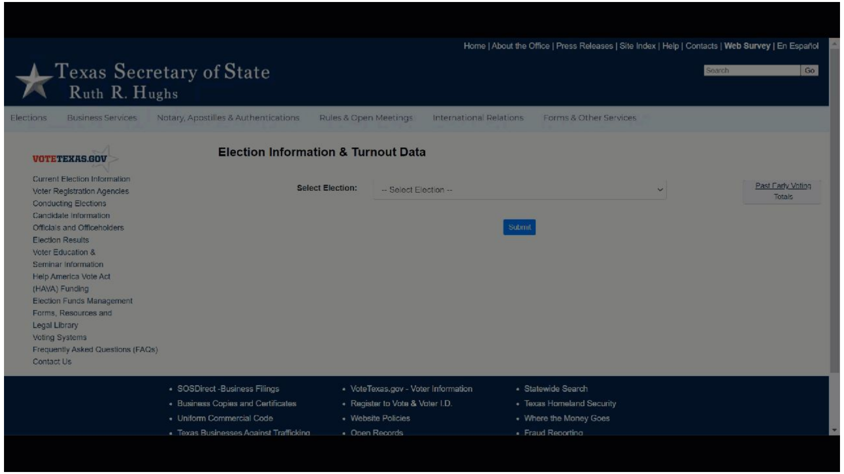
When data is available as...	Method
Link to a standalone file, e.g. in an S3 bucket	**Download:** Download with R {readr} package or python {urllib.requests}
Exposed through a REST API	**API Call:** R {httr2} package or python {requests}
Rendered server-side in the webpage's source, e.g. a static HTML table	**Scraping:** Extract content from underlying webpage structure
Rendered client-side or requires navigating through a UI	**Headless Browsing:** Use code to navigate to URLs, click buttons, etc.
Trapped within a complex UI or dashboard with no access to the source	**Computer Vision:** Parse characters from an image or webpage based on visual characteristics

Increasing Complexity

ELT
TO

EXTRACT: Automation can save manual data collection effort

Example:
Headless Browsing with Playwright




```
from playwright.sync_api import sync_playwright

with sync_playwright() as p:

  browser = p.firefox.launch()
  context = browser.new_context(accept_downloads = True)
  page = context.new_page()

# pick selections from menu
  page.goto(url)
  page.select_option('#idElection',
    label = "2020 NOVEMBER 3RD GENERAL ELECTION")
  page.click('#electionsInfoForm button')
  page.wait_for_selector('#selectedDate')

  page.select_option('#selectedDate', value = target_date)
  page.click('#electionsInfoForm button:nth-child(2)')
  page.wait_for_selector('"Generate Statewide Report"')

# download report
  with page.expect_download() as download_info:
    page.click('"Generate Statewide Report"')
  download = download_info.value
  download.save_as(target_file)
```

LOAD: Despite being the most common data storage formats, CSVs and Excel files can be brittle for computing

id_patient <str>	treated <int>	name <str>
‘01’	1	Emily
‘02’	1	Malcolm,
‘03’	1	Travis

write_csv(‘patients.csv’)
read_csv(‘patients.csv’)

Wrong type inferred,
leading zeros ignored

id_patient <int>	treated <int>	name <str>
1	1	Emily
2	1	Malcolm,
3	1	Travis

Could break file or skip row if not escaped

BONUS:
DuckDB’s CSV Sniffer can help identify and quarantine bad rows that keep you from opening a file

LOAD: Parquet has numerous advantages over CSV as a standard data format

1	**Similar scripting read/write**	Easily handled by {readr} or {arrow}
2	**Lower risk of accidental editing**	Not easily opened in UI or modified without explicit read/write
3	**Preserves data types**	Remembers column types to avoid data corruption
4	**Storing strategies allow efficient reads**	Allows for partitioned read/writes for pre-filtering of large files
5	**Smaller file sizes**	Compressed on disk

LOAD: Partitioning improves data organization and aids in efficient reading of large files

```
arrow::write_dataset(
  data,
  path = 'data/',
  format = "parquet",
  partitioning = "year"
)
```

```
arrow::open_dataset(
  'data'
)
```

```
arrow::open_dataset(
  'data/year=2022/'
)
```

📁data/year=2021/
📁data/year=2022/
📁data/year=2023/

📁data/year=2021/
📁data/year=2022/
📁data/year=2023/

📁data/year=2021/
📁data/year=2022/
📁data/year=2023/

E L T
T O

TRANSFORM: Data transformation choices help us organize our data for maximum success across projects

Data Modeling

Organizing the relationship between types of information in your warehouse

Table Design

Designing table contents, column names, documentation, etc. for maximum usability

Materialization

Deciding which intermediates you save as tables versus what exists only in transient code

TRANSFORM: Keep the simplest data structures as long as possible ("shift right") to preserve flexibility

id_patient	date	event
1	2025-01-05	treatment
1	2025-01-10	ice
2	2025-01-05	treatment
3	2025-01-05	treatment
3	2025-03-14	death
4	2025-01-05	treatment
4	2025-02-10	withdraw

id_patient	treated	withdrew	ice	death
1	1		1	
2	1			
3	1			1
4	1	1		

TRANSFORM: Keep the simplest data structures as long as possible ("shift right") to preserve flexibility

id_patient	date	event
1	2025-01-05	treatment
1	2025-01-10	ice
1	2025-01-12	death
2	2025-01-05	treatment
3	2025-01-05	treatment
4	2025-01-05	treatment
4	2025-02-10	withdraw

Flexible to query different inclusion/exclusion criteria

Easily updated with new records
(favor appends to joins)

TRANSFORM: Data modeling principles can help standardize disparate sources and make wrangling more manageable

Separate tables by different entity types and sources

patients
(1 row / patient)

patients_pii
(1 row / patient, limited acces)

patient_visits
(1 row / patient x visit)

Create separate tables for values applying to multiple records

id_patient	date	cd_diagnosis
1	2025-01-05	13
1	2025-01-10	1
1	2025-01-12	2
2	2025-01-05	14

cd_diagnosis	desc_diagnosis
1	A
2	B
3	C
4	D

TRANSFORM: Consistent column naming conventions across tables aids comprehension and downstream analysis code

Stub	Semantics
ID	Unique entity identifier
IND / IS	Binary 0/1 indicator; rest of name describes 1 condition
BIN	Binary 0/1 indicator; rest of name describes 1 condition
N	Count of quantity or event occurrences
AMT	Sum-able real number amount (“denominator free”)
…	…

```
library(dplyr)

data %>%

  group_by(NM_STATE) %>%

  summarize(

    across(starts_with("IND"), mean),

    across(contains("_ACTL_"), sum)

  )

#> # A tibble: 51 x 4
#>   NM_STATE     IND_COUNTY_HPSA  N_CASE_ACTL  N_DEATH_ACTL
#>   <chr>                  <dbl>        <dbl>         <dbl>
#> 1 Alabama                0.149       455582          7566
#> 2 Alaska                 0.235        51338           250
#> 3 Arizona                0           753379         13098
```

TEST: A multistage pipeline allows us to conduct hypothesis-drive upstream data quality checks (“shift left”)

stg_patients

ID	Age	Arm
1	57	1
2	45	0

stg_outcomes

ID	Score
1	50
2	0
3	20
4	10

LEFT JOIN

study_results

ID	Age	Arm	Score
1	57	1	50
2	45	0	0
3	NA	NA	20
4	NA	NA	10

Seen

Silent Failure

TEST: A multistage pipeline allows us to conduct hypothesis-drive upstream data quality checks (“shift left”)

stg_patients

ID	Age	Arm
1	57	1
2	45	0

Test Here:
Same cardinality in
stg_patient and stg_outcomes

stg_outcomes

ID	Score
1	50
2	0
3	20
4	10

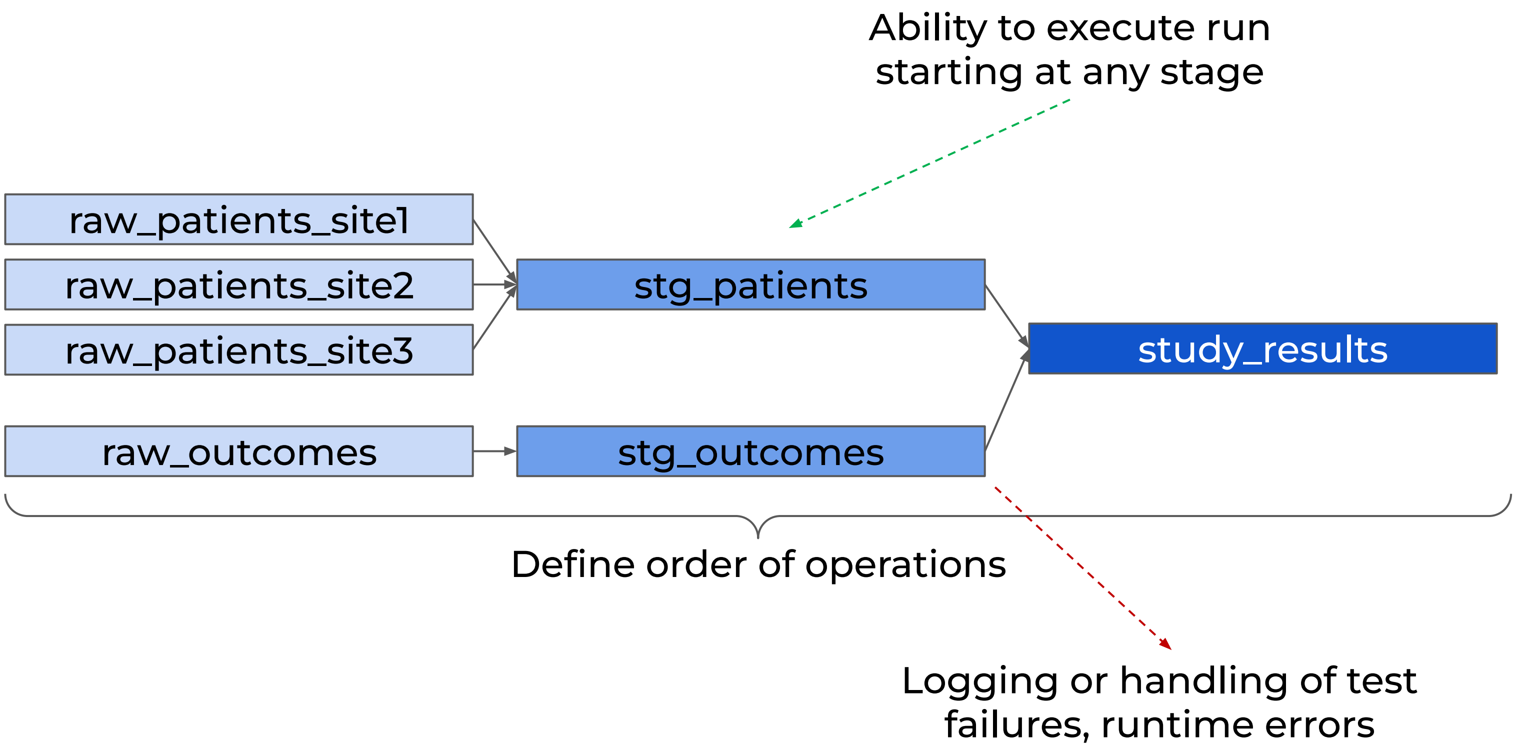
LEFT JOIN

study_results

ID	Age	Arm	Score
1	57	1	50
2	45	0	0

Not Here

ORCHESTRATE: Orchestration tools help us run many steps of an analysis pipeline in order to keep dependencies consistent



ORCHESTRATE: Many different tools exist for orchestration

makefile

- General purpose config file
- Runs any CLI commands based on user-defined dependencies
- Higher learning curve

```
all: ./data/clean.json

./data/raw.json: ./data/raw.json ./ingest.py
      mkdir -p data
      ./ingest.py

./data/clean.json: ./data/raw.json ./clean.py
      ./clean.py
```

dbt

- SQL framework that infers dependencies
- Compatible with any database including local compute (duckdb)
- Lower learning curve but less flexible

patients.sql

```
select id, name
from raw.patients
where dt_enroll = '2025-01-01'
```

results.sql

```
select id, name, outcome
from {{ref('patients')}}
  left join
  {{ref('outcomes')}}
  using (id)
```

Makefile example from https://datasciencesouth.com/blog/make/#a-make-data-pipeline-example
See also R package {targets} and Dagster

ELT
TO

Big ideas from data pipelining can help improve data handling in projects of any size of complexity

Architecture	Store data at different stages for reproducibility and efficiency
Extract	Use automated tools to accelerate tedious extraction processes
Load	Consider Parquet files for resilient type-safe file storage and sharing
Transform	Apply data modeling techniques to increase flexibility and clarity
Test	Test intermediate artifacts to detect hidden failures at their root cause
Orchestrate	Adopt automated tooling to ensure all steps of processing stay in sync

Thank you!