

# **A different type of DAG: Data pipelines principles for epidemiology**

Emily Riederer  
*Data Science Lead at Capital One*

# Disclosures

No relevant financial relationships exist.

I am an employee of Capital One Financial Corporation.

# Industry data science can learn a lot from epidemiology

Epi

**Data**

**Analytical Approach**

**Precise Communication**

Imperfection expected

Prototypical study designs

Estimands

Data Management

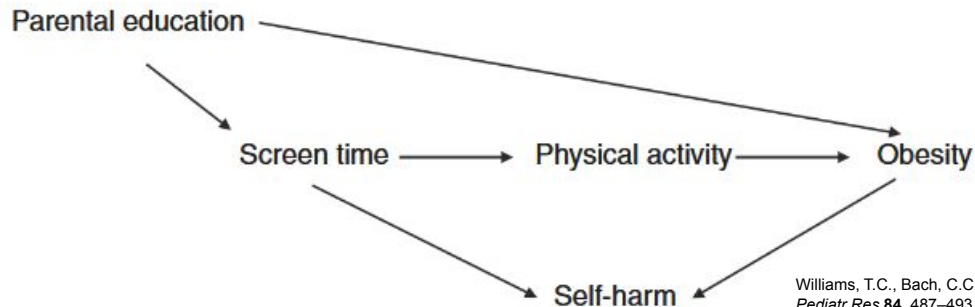
Packaged Code

Deployed Outputs

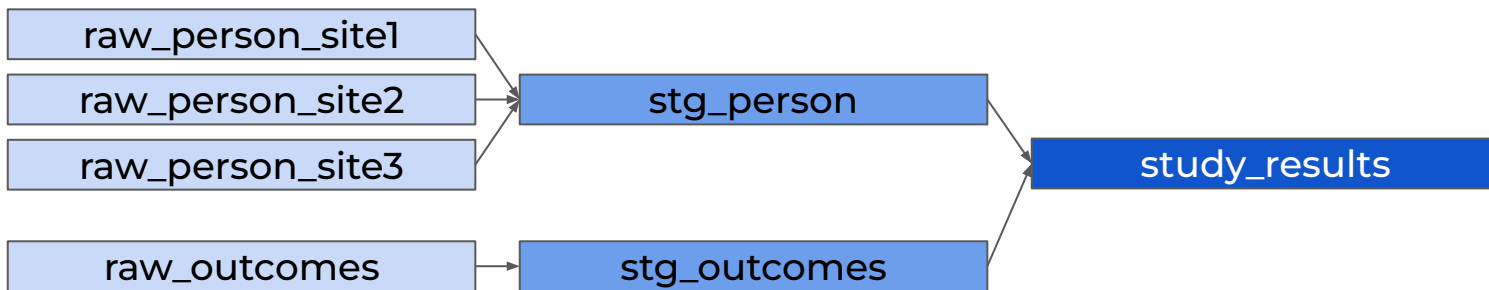
DS

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

Epi



DS



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

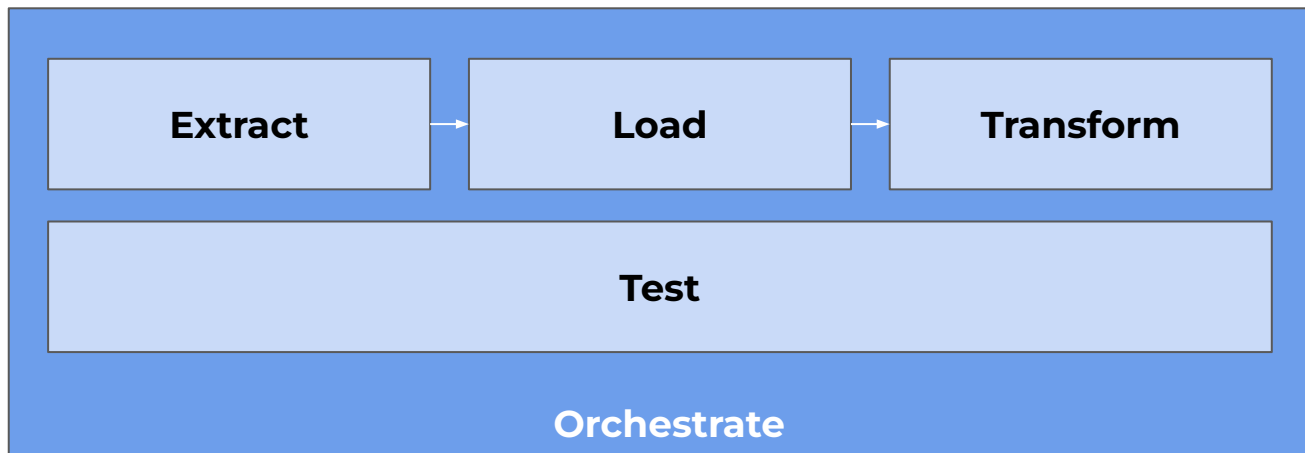
## Data Pipelines

- Data lineage and quality control
- New data ingestion, integration, and updates
- Sharing data artifacts across many users
- Sharing data artifacts across many use cases






## Project Benefit

- Reproducibility
- Harmonizing disparate sources
- Collaboration
- Reuse across projects

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

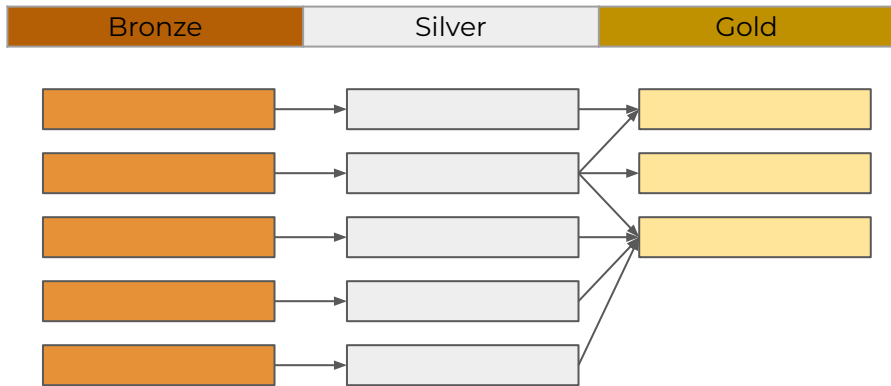


# The same principles apply across platforms – all you need to get started is a hard drive

	Local	Shared	Cloud
<b>Storage</b>	 Local Drive	 Network Drive <ul style="list-style-type: none"><li>Google Cloud Storage</li><li>Amazon S3</li></ul>	 Google BigQuery
<b>Compute</b>	 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/persons.csv')  
  
df_cleaned = dplyr::filter(df, dt_enroll < cutoff)  
  
readr::write_csv(df_cleaned, 'data/structured/persons.csv')
```



```
df = readr::read_csv('data/raw/persons.csv')  
  
df = dplyr::filter(df, dt_enroll < cutoff)  
  
readr::write_csv(df, 'data/raw/persons.csv')
```



```
df = readr::read_csv('data/raw/persons.csv')  
  
df_cleaned = dplyr::filter(df, dt_enroll < cutoff)  
  
readr::write_csv(df_cleaned, 'data/raw/persons.csv')
```



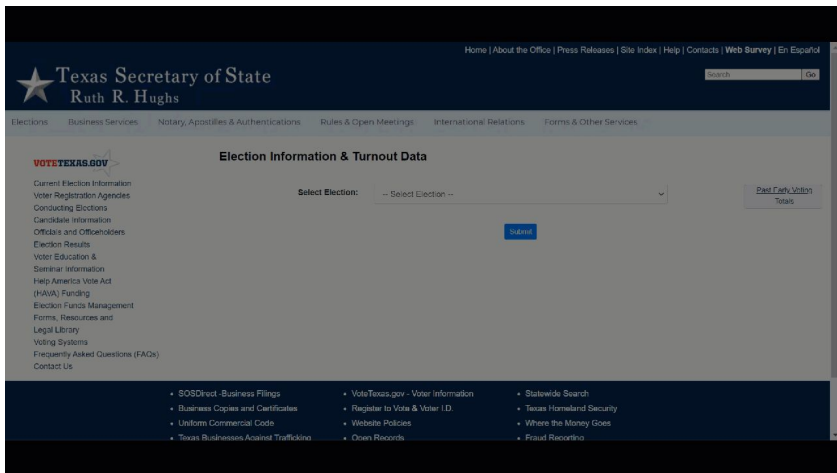
# EXTRACT: Many different techniques can be used to collect data across disparate and sometimes inconvenient sources

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

*Increasing Complexity*

# 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_person <str>	treated <int>	name <str>
'01'	1	Emily
'02'	1	Malcolm,
'03'	0	Travis
'04'	0	Konrad

→  
`write_csv('persons.csv')`  
`read_csv('persons.csv')`

Wrong type inferred,  
leading zeros ignored

id_person <int>	treated <int>	name <str>
1	1	Emily
2	1	Malcolm,
3	0	Travis
4	0	Konrad

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 files are like a zipfile of data and metadata

## CSV

- Human readable and editable
- All data in single text file
- Row-based

```
id_person, treated, name
1, 1, Emily
2, 1, Malcolm
3, 0, Travis
4, 0, Konrad
```

## Parquet

- Compressed and machine readable
- Bundles data and metadata
- Column-based

Metadata

id\_person:1,2,3,4

treated:1,1,0,0




name:Emily,Malcolm,Travis,Konrad

# 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"  
)
```

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

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

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

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

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



# 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_person	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_person	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_person	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

person (1 row / person)
----------------------------

persons_pii (1 row / person, limited access)
---

persons_visits (1 row / person x visit)
--

Create separate tables for values applying to multiple records

id_person	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(

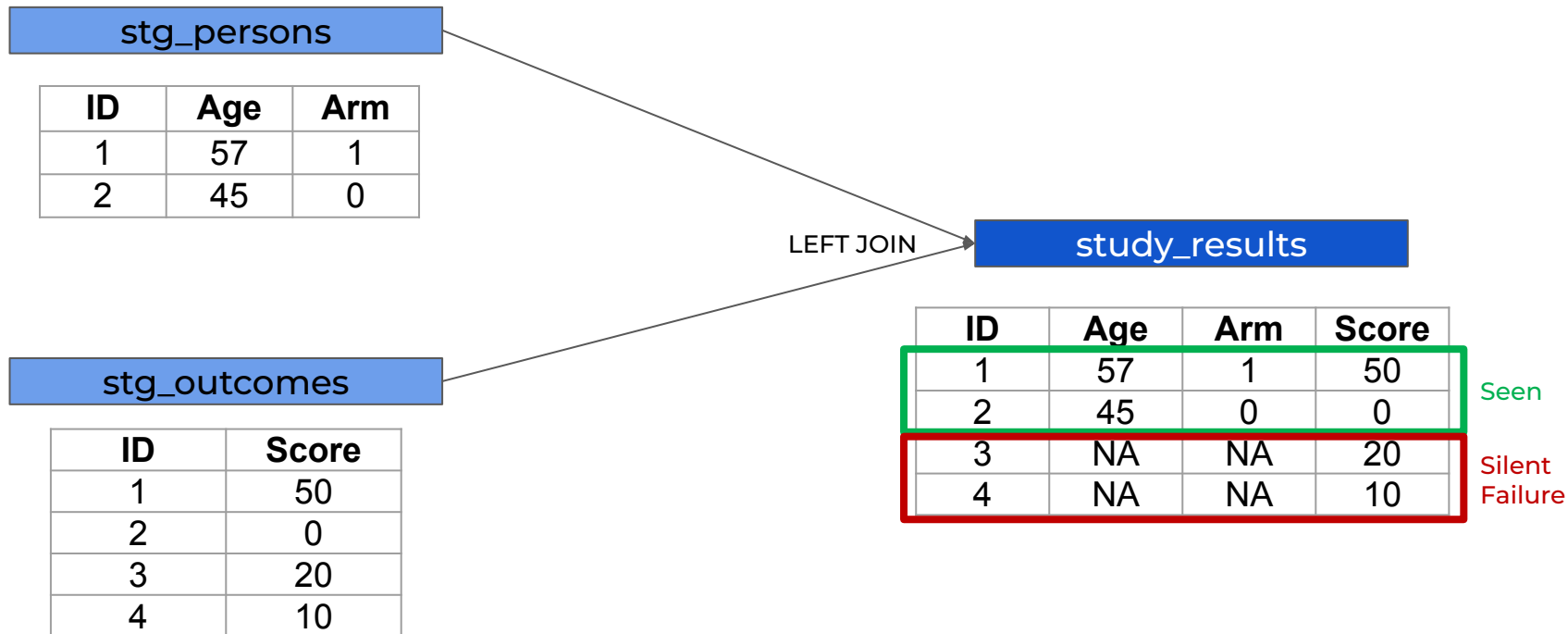
    across(starts_with("IND"), mean),

    across(contains("_ACTL_"), sum)

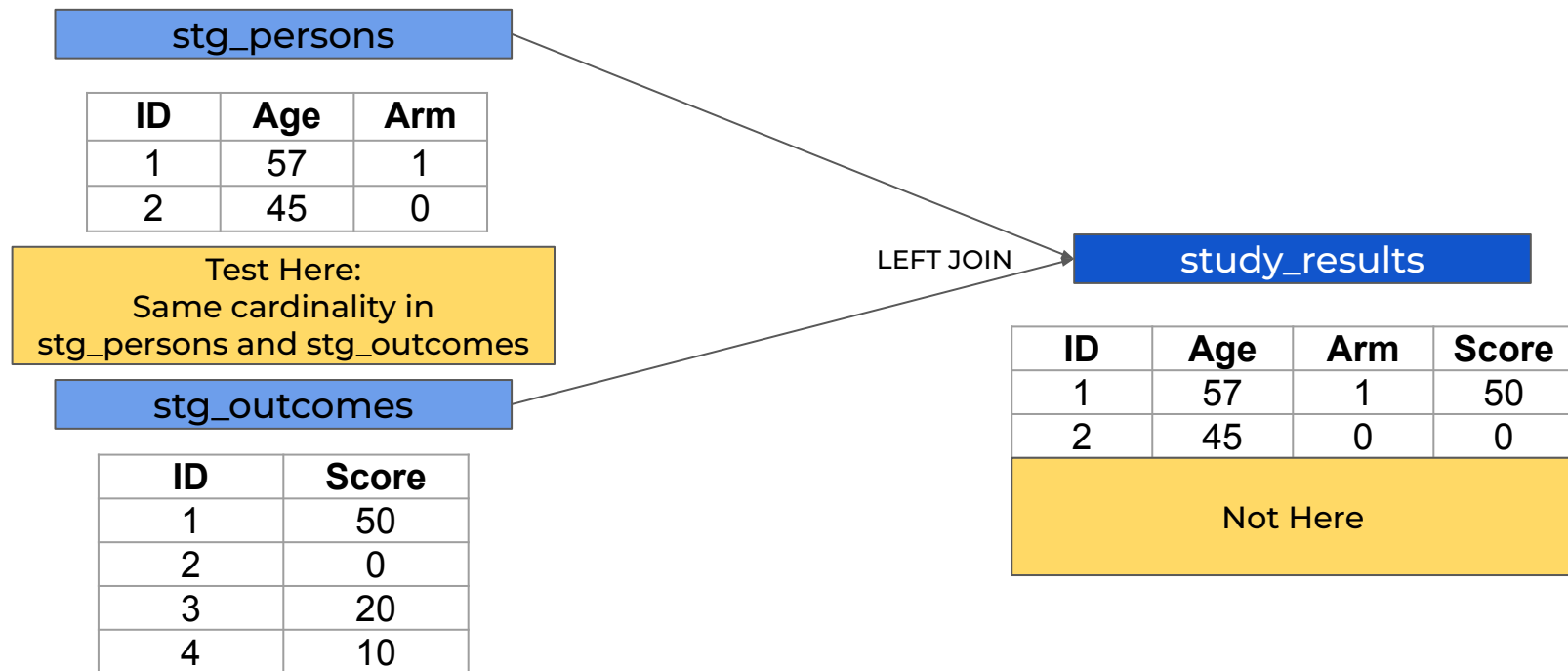
  )

#> # 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
```

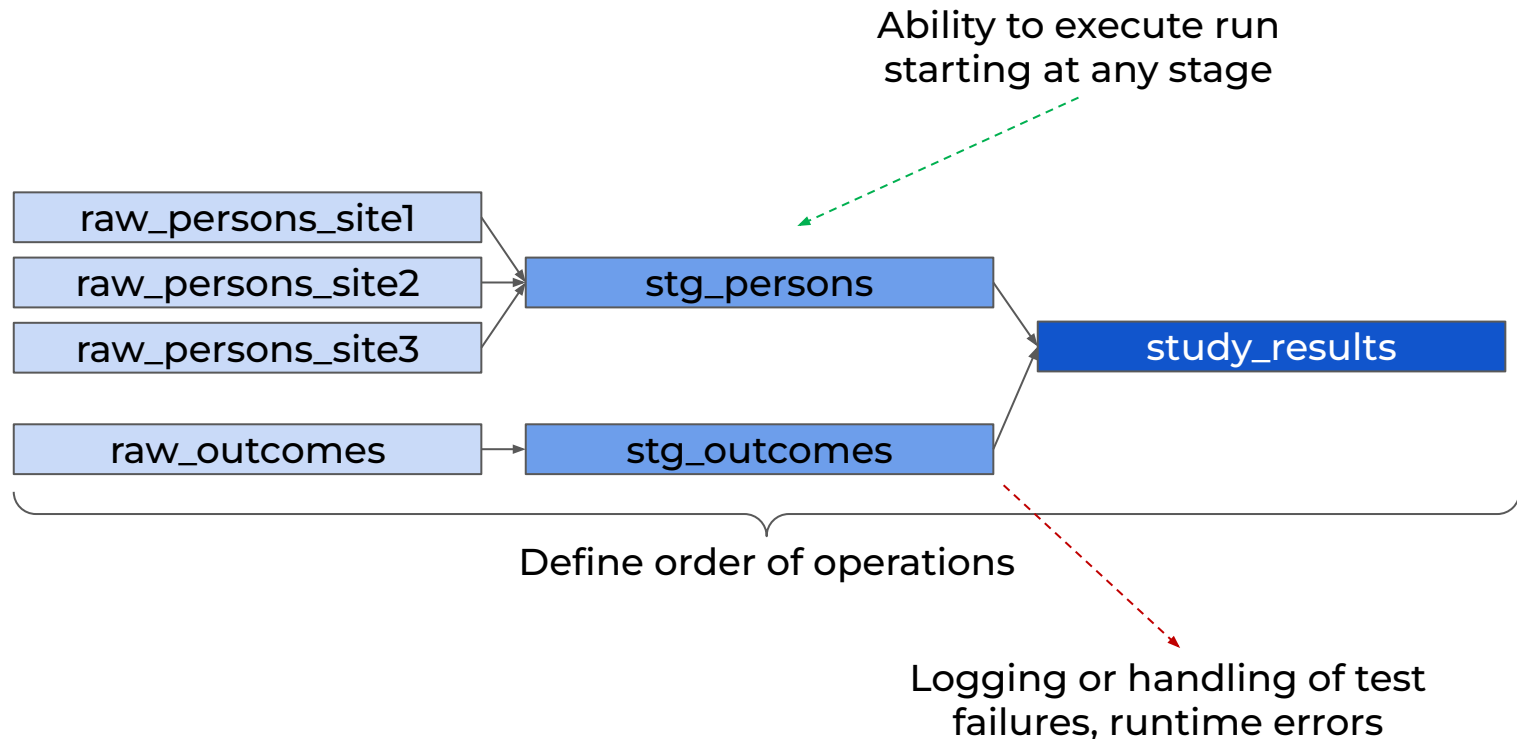
# TEST: A multistage pipeline allows us to conduct hypothesis-drive upstream data quality checks (“shift left”)



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



# 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 shell 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
```

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

## dbt

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

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

persons.sql

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

results.sql

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

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

**Thank you!**