dagster

Rising Tides with Radical Transparency:

# Why and How to Open Source Your Data Platform

Tim Castillo
Data Engineer + Developer Advocate - Dagster Labs

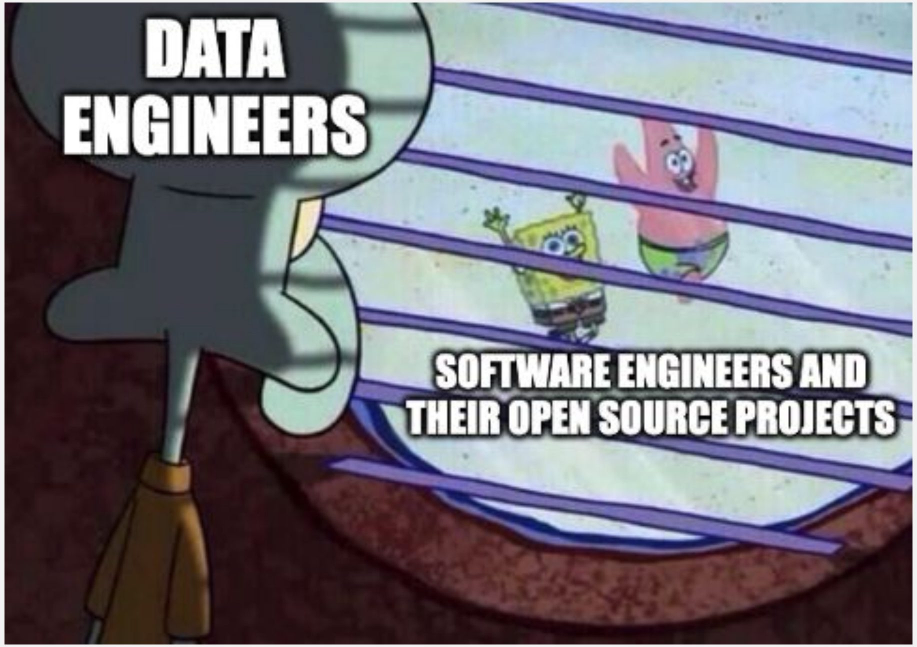
## A quick overview

- Who am I?
- Finding realistic, good quality projects to learn from is hard
- What is Dagster Open Platform (DOP)?
- What lessons can you learn from DOP?
- How else can you learn?
- How can you open source?

## Who am I?

- Tim Castillo
- Fun Fact
  - Every morning, I share a banana with one of my cats
- Data Engineer + Developer Advocate for Dagster Labs

## Finding **realistic**, good quality projects to learn from is hard



dagster

**Dagster Open Platform (DOP)** is a public repository of Dagster Lab's own data pipelines.

It's what a fast-growing SaaS startup does, and includes parts of our cloud product analytics, marketing attribution, and KPIs

dagster

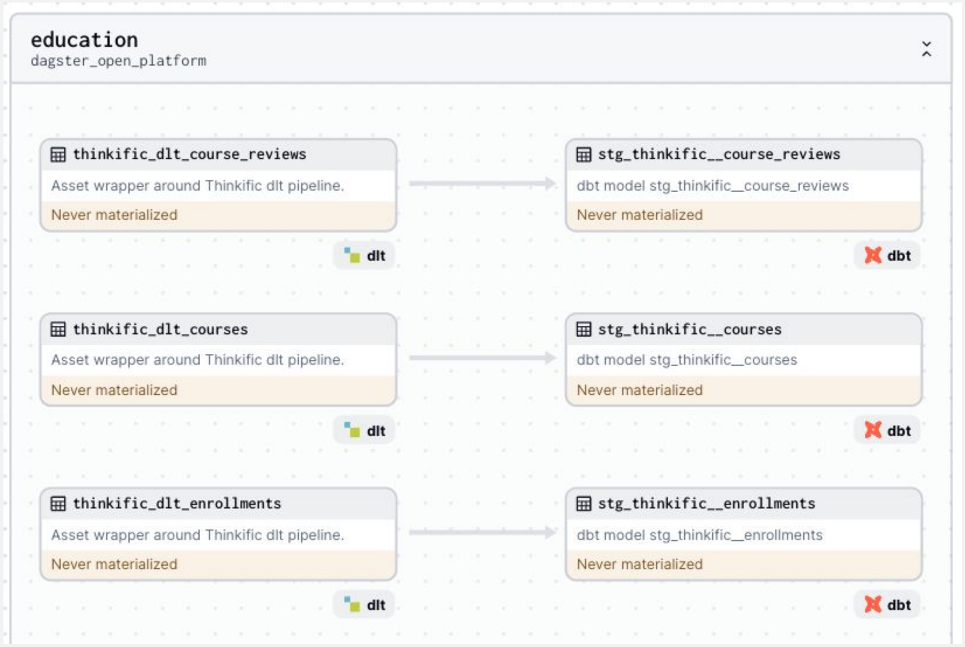
What can I do with **Dagster Open Platform?**

What is **Dagster?**

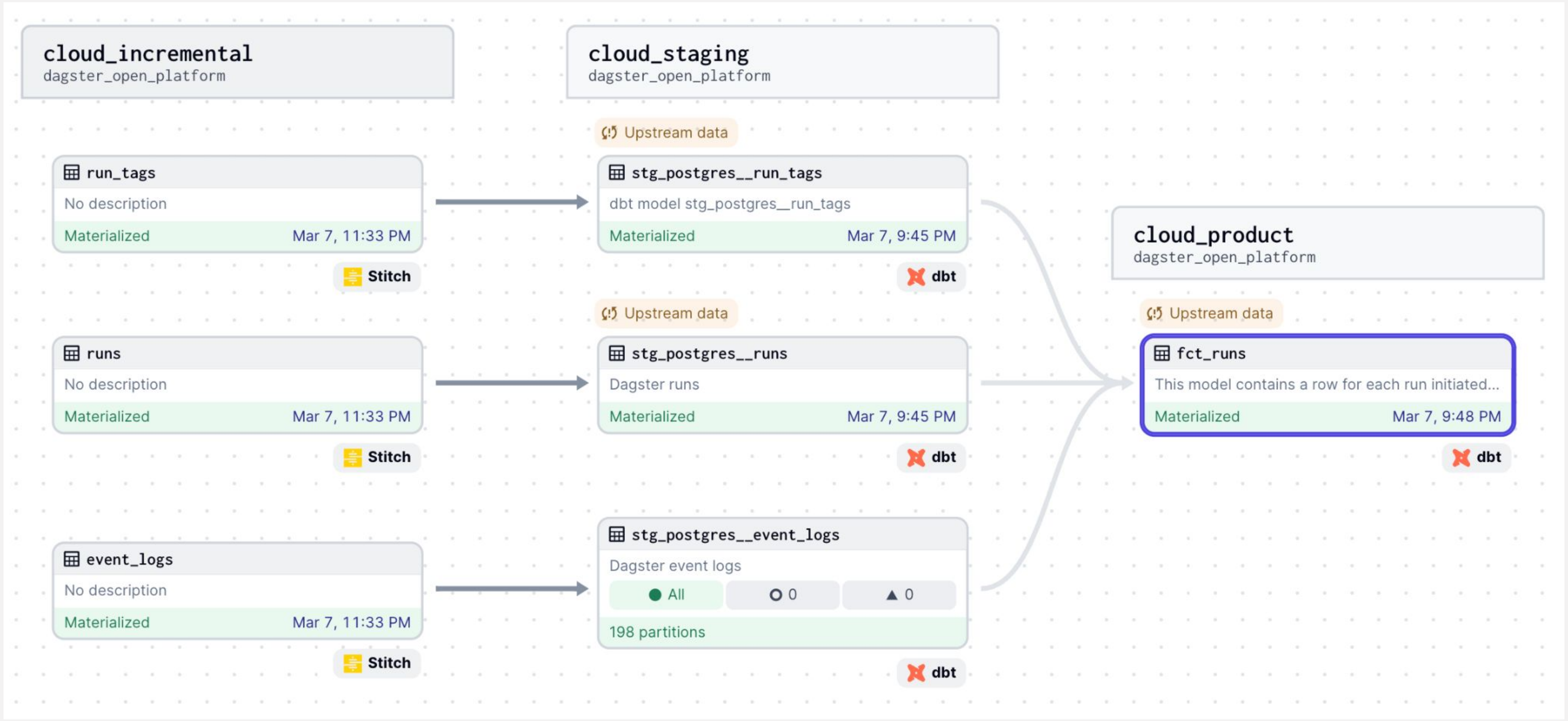
What can you **learn about data engineering** from DOP?

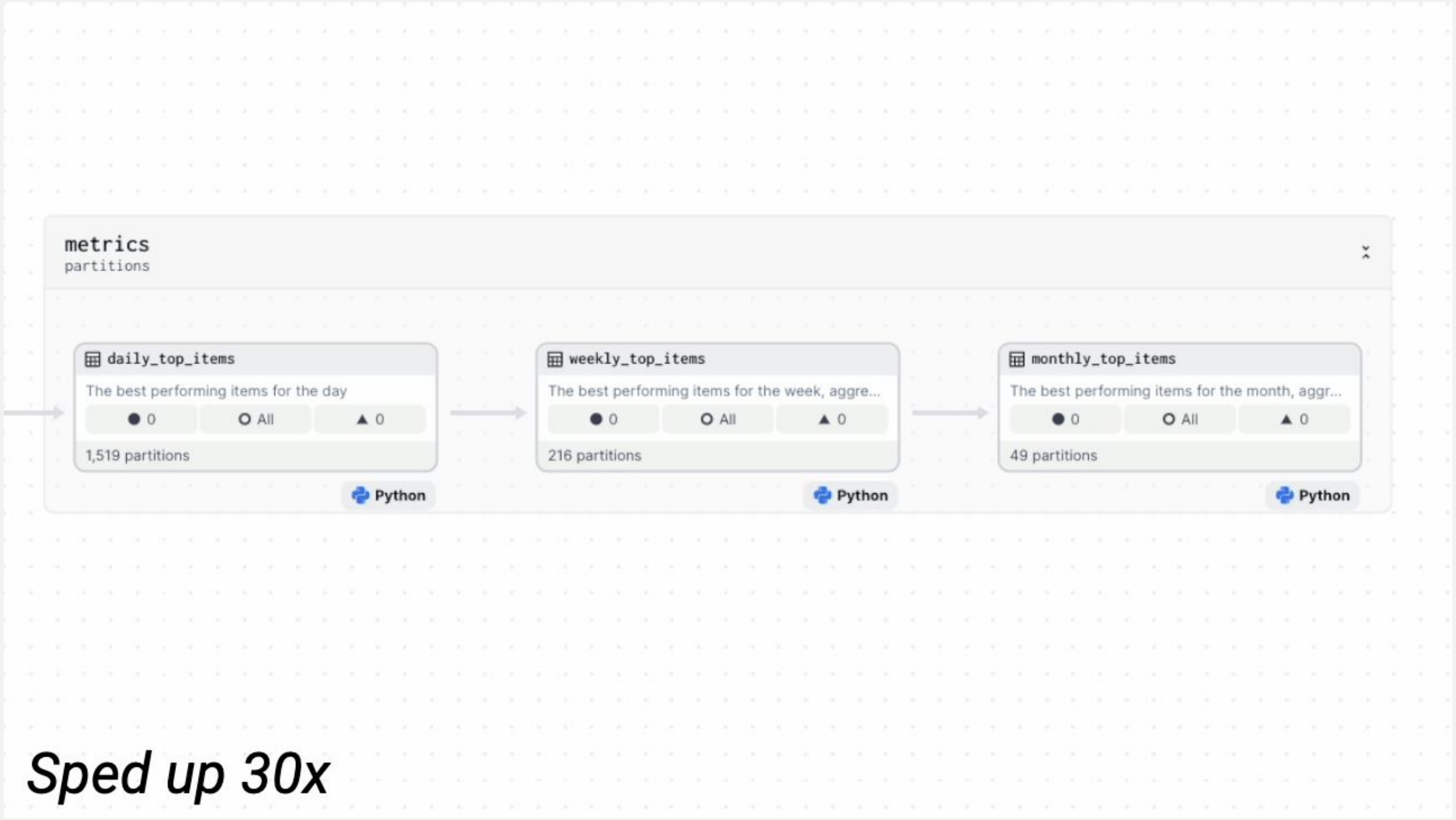
# What is Dagster?

- Dagster is a Python-based framework for **orchestrating data pipelines**
- Users focus on building their data **assets**, rather than the tasks to make them
  - Assets are tables, files, ML models, etc.
- This asset-centric approach comes with features and buildings blocks like:
  - Partitioning your data
  - Rebuilding data automatically
  - Cost observability



education
dagster_open_platform
thinkific_dlt_course_reviews
Asset wrapper around Thinkific dlt pipeline.
Never materialized
dlt
stg_thinkific__course_reviews
dbt model stg_thinkific__course_reviews
Never materialized
dbt
thinkific_dlt_courses
Asset wrapper around Thinkific dlt pipeline.
Never materialized
dlt
stg_thinkific__courses
dbt model stg_thinkific__courses
Never materialized
dbt
thinkific_dlt_enrollments
Asset wrapper around Thinkific dlt pipeline.
Never materialized
dlt
stg_thinkific__enrollments
dbt model stg_thinkific__enrollments
Never materialized
dbt


cloud_incremental
dagster_open_platform
cloud_staging
dagster_open_platform
cloud_product
dagster_open_platform
Upstream data
run_tags
No description
Materialized
Mar 7, 11:33 PM
Stitch
stg_postgres__run_tags
dbt model stg_postgres__run_tags
Materialized
Mar 7, 9:45 PM
dbt
Upstream data
runs
No description
Materialized
Mar 7, 11:33 PM
Stitch
stg_postgres__runs
Dagster runs
Materialized
Mar 7, 9:45 PM
dbt
Upstream data
fct_runs
This model contains a row for each run initiated...
Materialized
Mar 7, 9:48 PM
dbt
event_logs
No description
Materialized
Mar 7, 11:33 PM
Stitch
stg_postgres__event_logs
Dagster event logs
All
0
0
198 partitions
dbt


metrics
partitions
daily_top_items
The best performing items for the day
● 0
O All
▲ 0
1,519 partitions
Python
weekly_top_items
The best performing items for the week, aggre...
● 0
O All
▲ 0
216 partitions
Python
monthly_top_items
The best performing items for the month, aggr...
● 0
O All
▲ 0
49 partitions
Python

*Sped up 30x*

dagster

Dagster is a framework for orchestrating **assets** in data pipelines

Dagster Open Platform is written in Dagster, but you don’t need to use Dagster to learn from it

# What lessons can you learn from DOP?

- Common heuristics for data quality tests
- Pythonic best practices
- **Partitioning data to improve performance**
- Documentation standards
- Python type hinting and annotations
- **Software design patterns**
- dbt best practices
- SQL query optimization techniques
- Ingesting data from various sources with different tools
- **Discovering new tools and utilities**
- Finetune a RAG model
- Syncing data to Salesforce
- Testing externally validated data
- CI/CD
- **Local, staging, and production environments**
- *And more!*

*None of this is Dagster-exclusive*

dagster

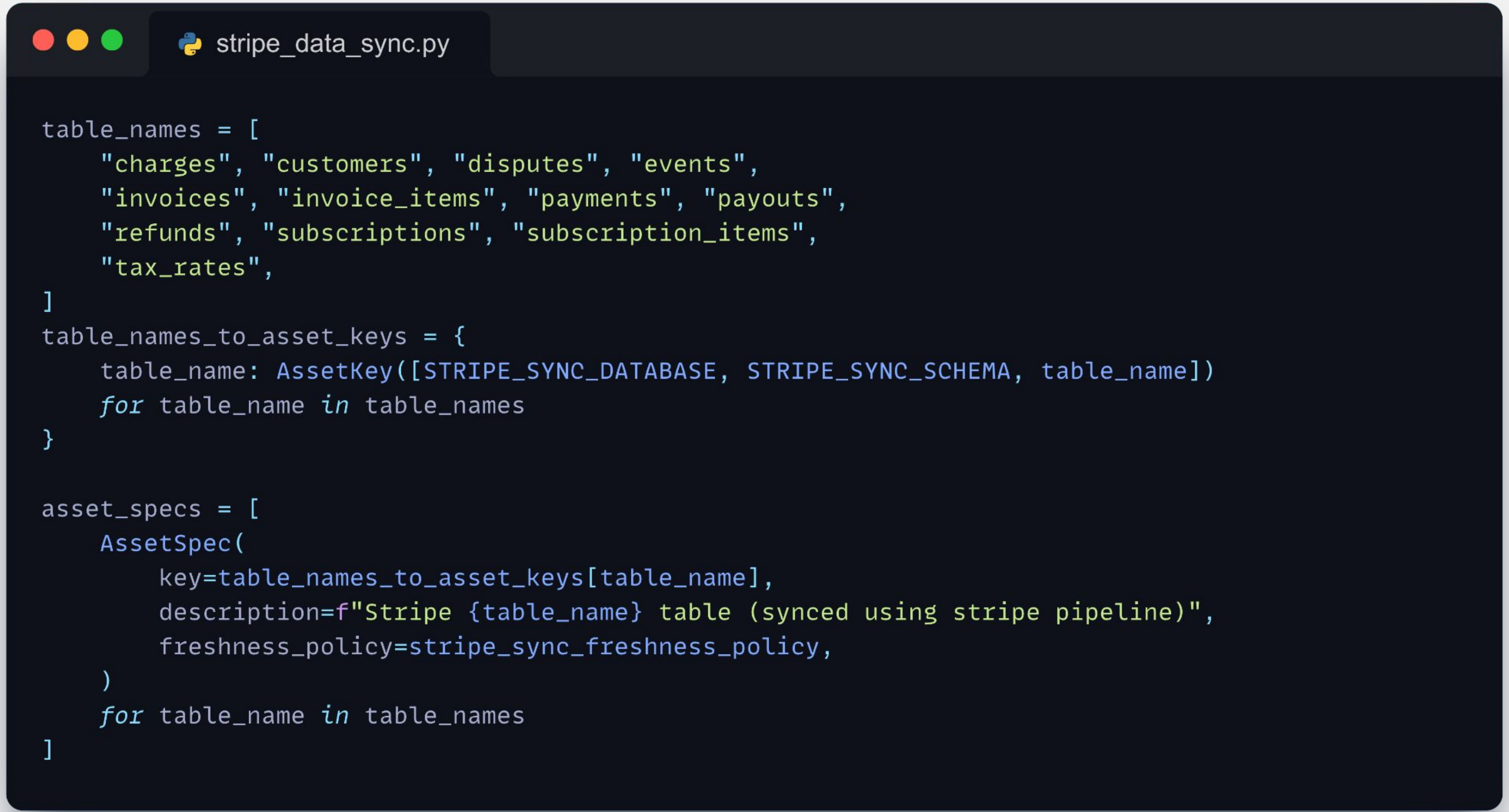
Use software **design patterns**

Invest in **developer experience**

Be **environment-aware**

Thinking of your data in **partitions**

# Use design patterns

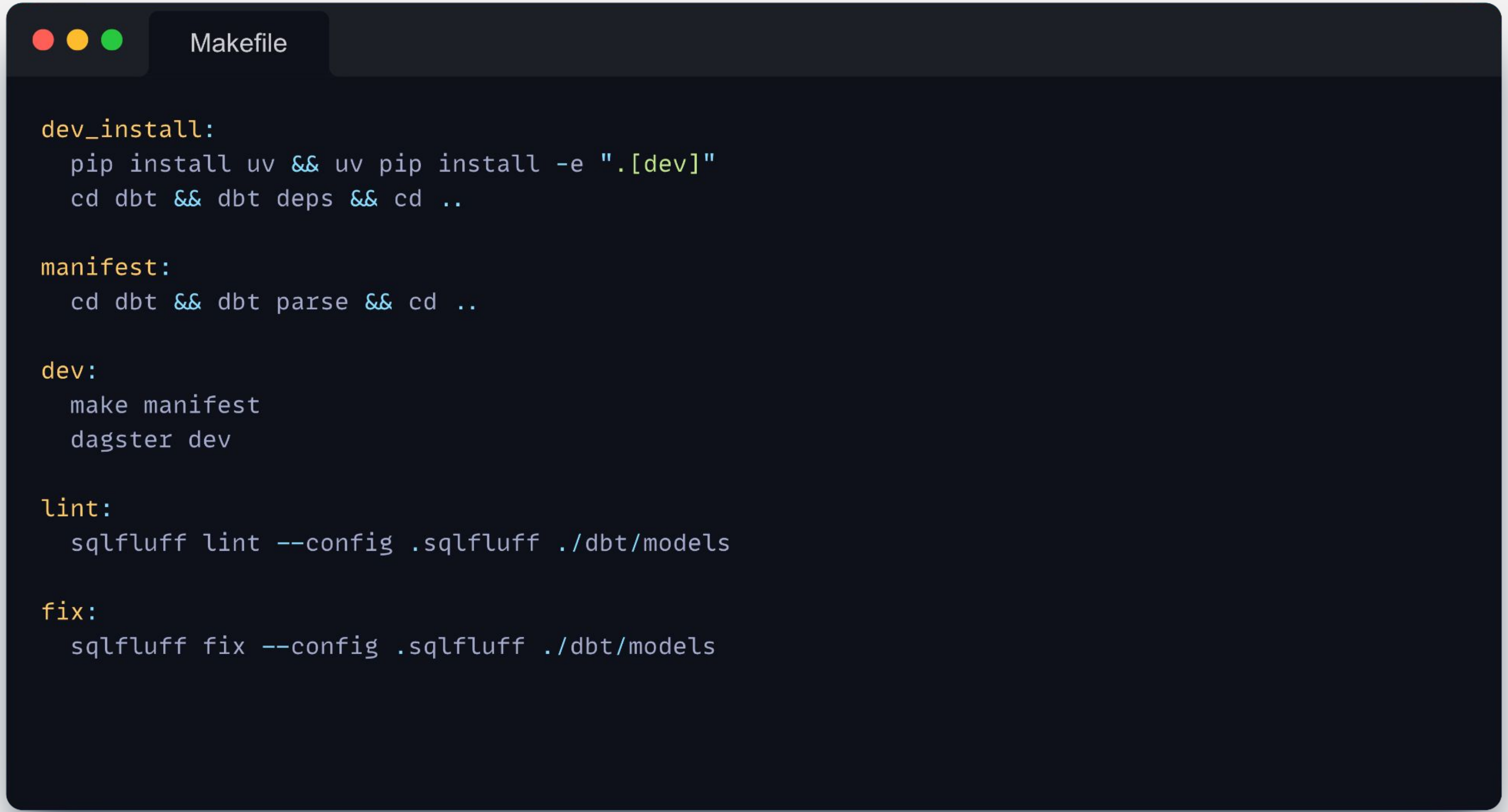


stripe_data_sync.py

```python
table_names = [
    "charges", "customers", "disputes", "events",
    "invoices", "invoice_items", "payments", "payouts",
    "refunds", "subscriptions", "subscription_items",
    "tax_rates",
]
table_names_to_asset_keys = {
    table_name: AssetKey([STRIPE_SYNC_DATABASE, STRIPE_SYNC_SCHEMA, table_name])
    for table_name in table_names
}

asset_specs = [
    AssetSpec(
        key=table_names_to_asset_keys[table_name],
        description=f"Stripe {table_name} table (synced using stripe pipeline)",
        freshness_policy=stripe_sync_freshness_policy,
    )
    for table_name in table_names
]
```

Design patterns help you make sustainable and scalable code

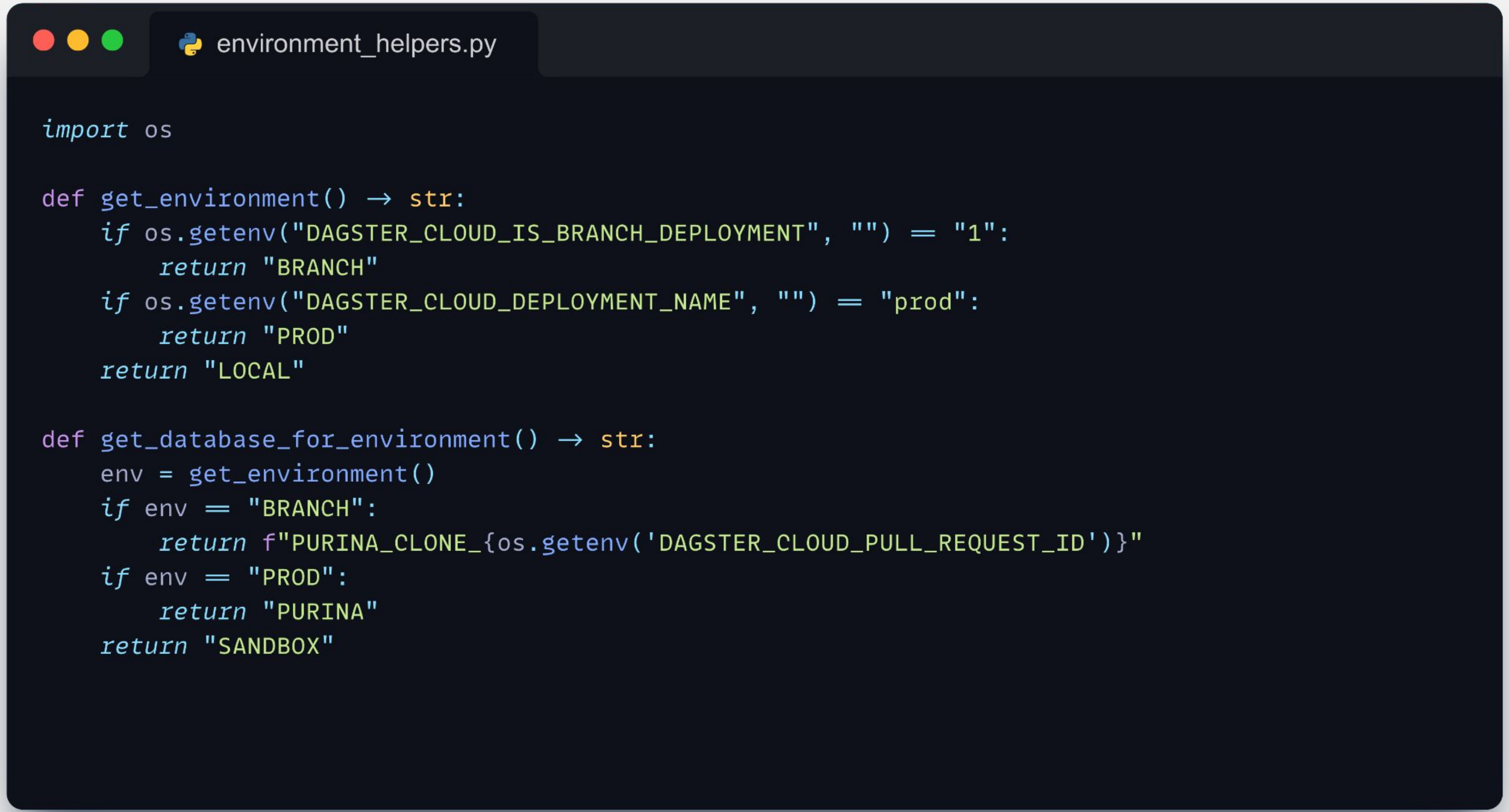
# Invest in developer experience



```
dev_install:
  pip install uv && uv pip install -e ".[dev]"
  cd dbt && dbt deps && cd ..

manifest:
  cd dbt && dbt parse && cd ..

dev:
  make manifest
  dagster dev

lint:
  sqlfluff lint --config .sqlfluff ./dbt/models

fix:
  sqlfluff fix --config .sqlfluff ./dbt/models
```

Make it easy to run and contribute

# Be environment-aware

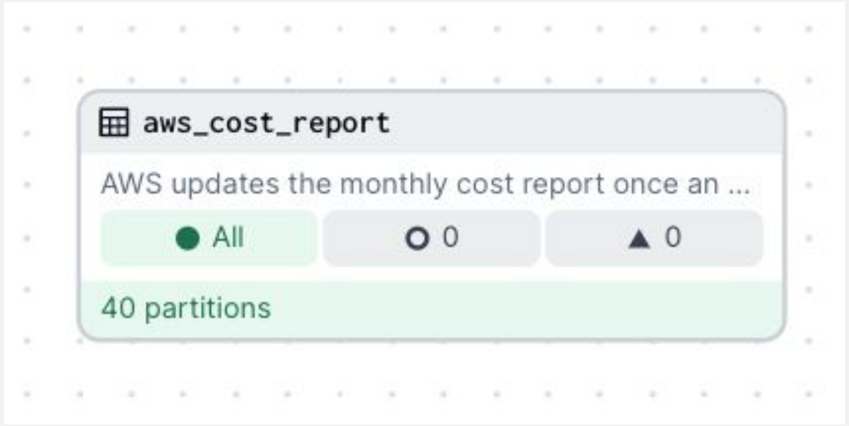


environment_helpers.py

```python
import os

def get_environment() -> str:
    if os.getenv("DAGSTER_CLOUD_IS_BRANCH_DEPLOYMENT", "") == "1":
        return "BRANCH"
    if os.getenv("DAGSTER_CLOUD_DEPLOYMENT_NAME", "") == "prod":
        return "PROD"
    return "LOCAL"

def get_database_for_environment() -> str:
    env = get_environment()
    if env == "BRANCH":
        return f"PURINA_CLONE_{os.getenv('DAGSTER_CLOUD_PULL_REQUEST_ID')}"
    if env == "PROD":
        return "PURINA"
    return "SANDBOX"
```

## Optimize your workflows per environment

# Think in partitions



aws_cost_report

AWS updates the monthly cost report once an ...

40 partitions

Link to File

## Save time and money by partitioning your data

dagster

Use software **design patterns**

Invest in **developer experience**

Be **environment-aware**

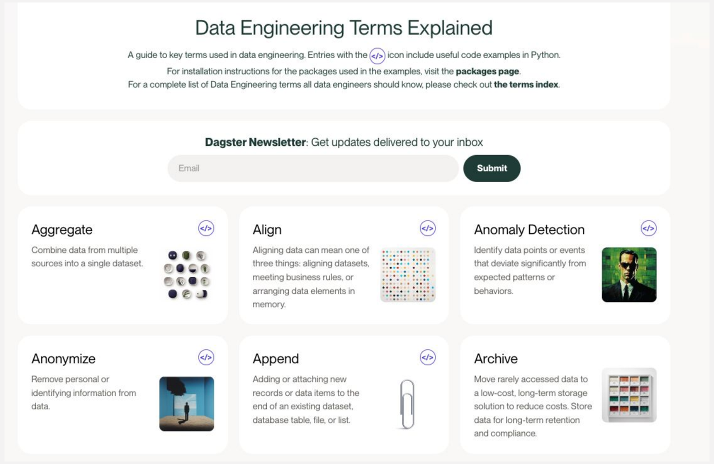
Thinking of your data in **partitions**

dagster

## How else can you learn?

# Other resources

- MIT Open Learning
- Ibis Project
- GitLab Analytics
- Mattermost's Data Warehouse



Data Engineering Terms Explained

A guide to key terms used in data engineering. Entries with the </> icon include useful code examples in Python.
For installation instructions for the packages used in the examples, visit the **packages page**.
For a complete list of Data Engineering terms all data engineers should know, please check out **the terms index**.

**Dagster Newsletter**: Get updates delivered to your inbox

Aggregate
Combine data from multiple sources into a single dataset.

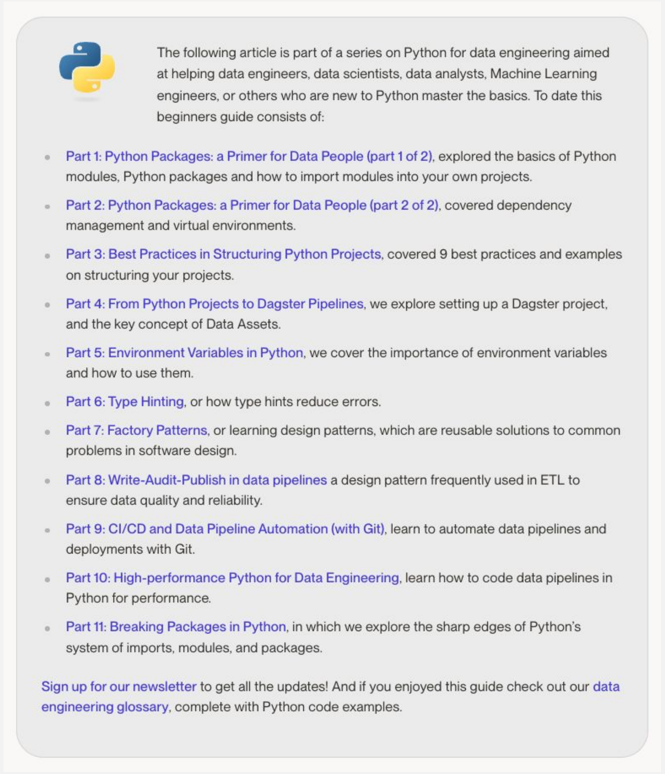
Align
Aligning data can mean one of three things: aligning datasets, meeting business rules, or arranging data elements in memory.

Anomaly Detection
Identify data points or events that deviate significantly from expected patterns or behaviors.

Anonymize
Remove personal or identifying information from data.

Append
Adding or attaching new records or data items to the end of an existing dataset, database table, file, or list.

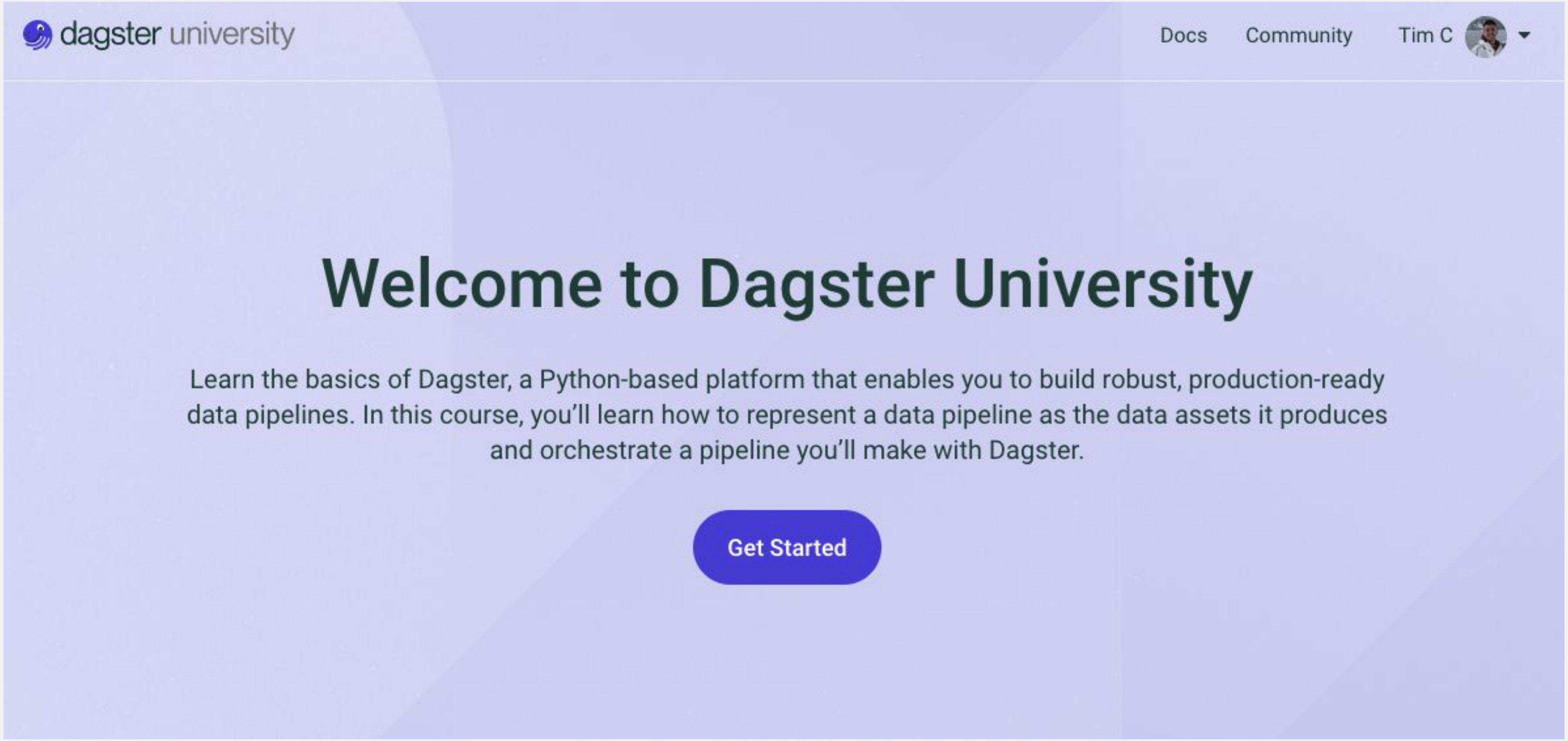
Archive
Move rarely accessed data to a low-cost, long-term storage solution to reduce costs. Store data for long-term retention and compliance.



The following article is part of a series on Python for data engineering aimed at helping data engineers, data scientists, data analysts, Machine Learning engineers, or others who are new to Python master the basics. To date this beginners guide consists of:

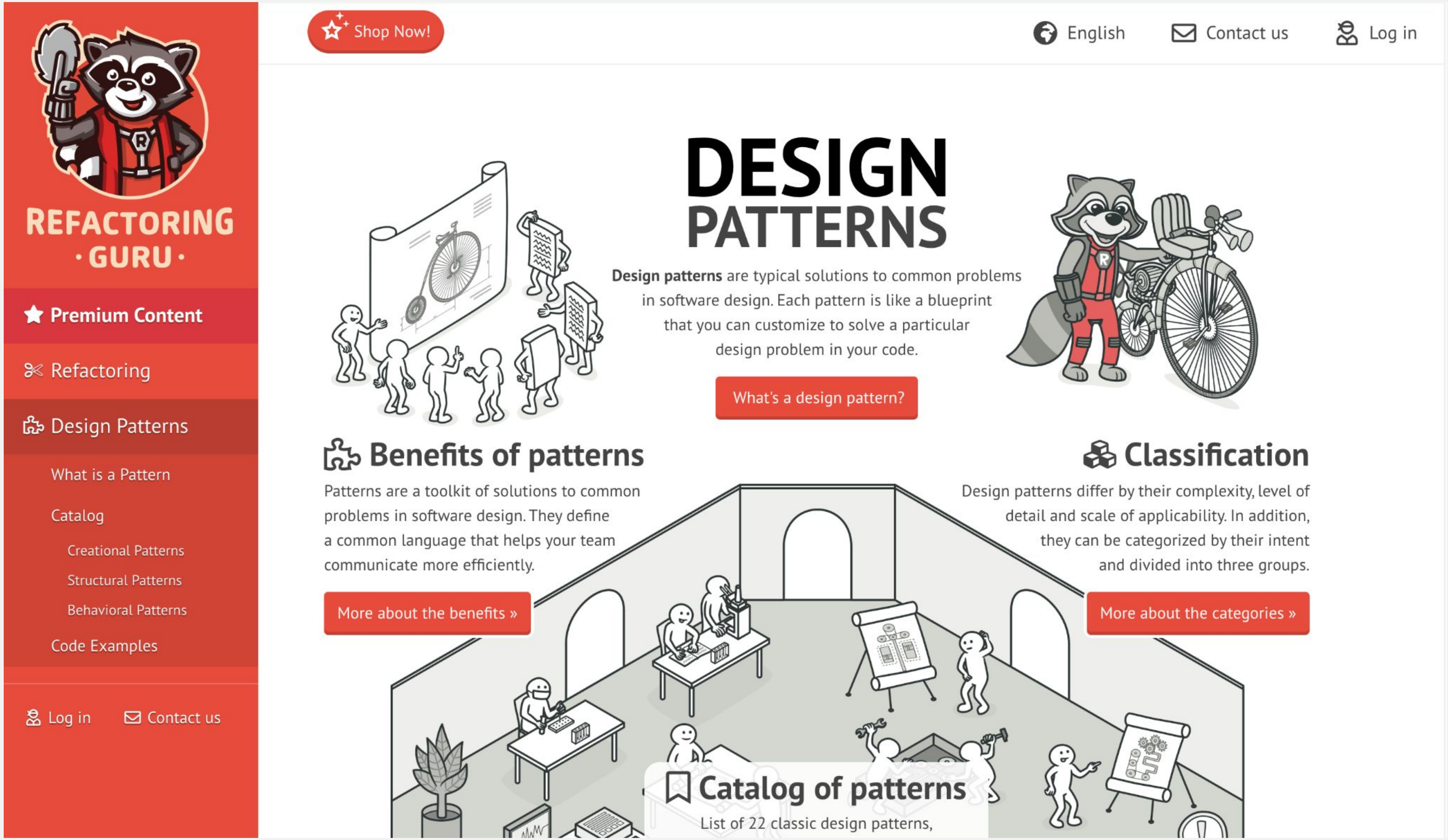
- Part 1: Python Packages: a Primer for Data People (part 1 of 2), explored the basics of Python modules, Python packages and how to import modules into your own projects.
- Part 2: Python Packages: a Primer for Data People (part 2 of 2), covered dependency management and virtual environments.
- Part 3: Best Practices in Structuring Python Projects, covered 9 best practices and examples on structuring your projects.
- Part 4: From Python Projects to Dagster Pipelines, we explore setting up a Dagster project, and the key concept of Data Assets.
- Part 5: Environment Variables in Python, we cover the importance of environment variables and how to use them.
- Part 6: Type Hinting, or how type hints reduce errors.
- Part 7: Factory Patterns, or learning design patterns, which are reusable solutions to common problems in software design.
- Part 8: Write-Audit-Publish in data pipelines a design pattern frequently used in ETL to ensure data quality and reliability.
- Part 9: CI/CD and Data Pipeline Automation (with Git), learn to automate data pipelines and deployments with Git.
- Part 10: High-performance Python for Data Engineering, learn how to code data pipelines in Python for performance.
- Part 11: Breaking Packages in Python, in which we explore the sharp edges of Python's system of imports, modules, and packages.

Sign up for our newsletter to get all the updates! And if you enjoyed this guide check out our data engineering glossary, complete with Python code examples.



Welcome to Dagster University

Learn the basics of Dagster, a Python-based platform that enables you to build robust, production-ready data pipelines. In this course, you'll learn how to represent a data pipeline as the data assets it produces and orchestrate a pipeline you'll make with Dagster.



# DESIGN PATTERNS

**Design patterns** are typical solutions to common problems in software design. Each pattern is like a blueprint that you can customize to solve a particular design problem in your code.

What's a design pattern?

## Benefits of patterns

Patterns are a toolkit of solutions to common problems in software design. They define a common language that helps your team communicate more efficiently.

More about the benefits »

## Classification

Design patterns differ by their complexity, level of detail and scale of applicability. In addition, they can be categorized by their intent and divided into three groups.

More about the categories »

## Catalog of patterns

List of 22 classic design patterns,

dagster

## How to navigate the open source process

## So you want to show off your work?

1. Get leadership buy-in
2. Navigate their priorities and needs
3. Understand the responsibility involved
4. Build processes and safeguards
5. Make the culture shift across the company

dagster

## Get leadership buy-in

## Navigate their priorities and needs

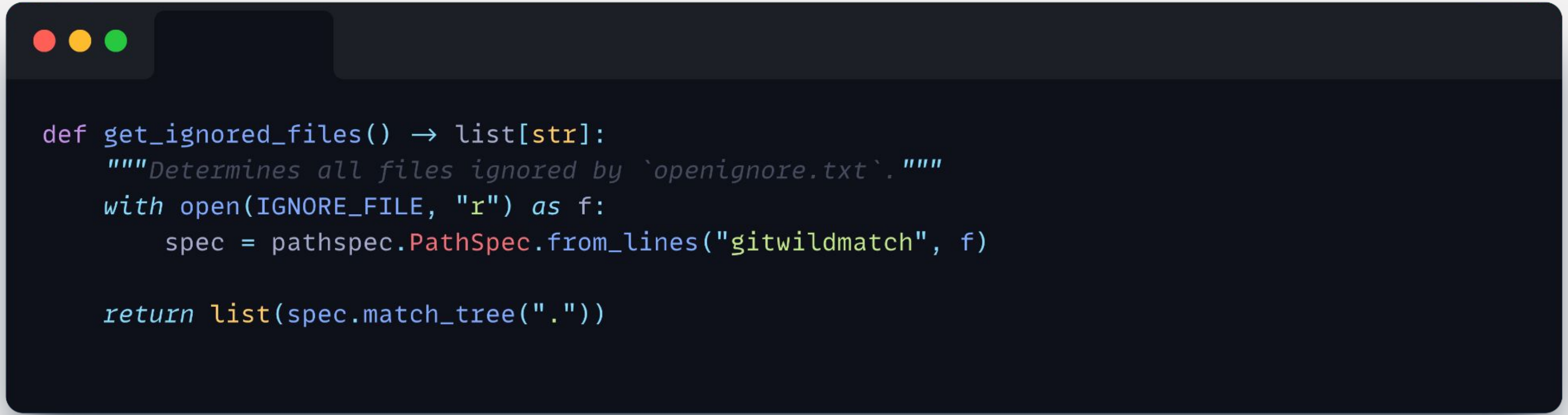
1. **Be Compliant** - Our data or sensitive business logic must not be open-sourced as part of this process
2. **Don’t Sacrifice Productivity** - We as data engineers cannot take a significant productivity hit by doing this. These are our data pipelines first, and education second, so don’t let it in our way.
3. **Use Best Practices** - Instead, use it as a forcing function to push for us writing better quality code knowing that the entire world could look at us

dagster

## Understand the responsibility involved

## Build processes and safeguards

1. All pull requests/code review happens in our internal repository
2. Merged pull requests push code changes to the public repository
3. Things we want to keep private are defined by a .gitignore-like file



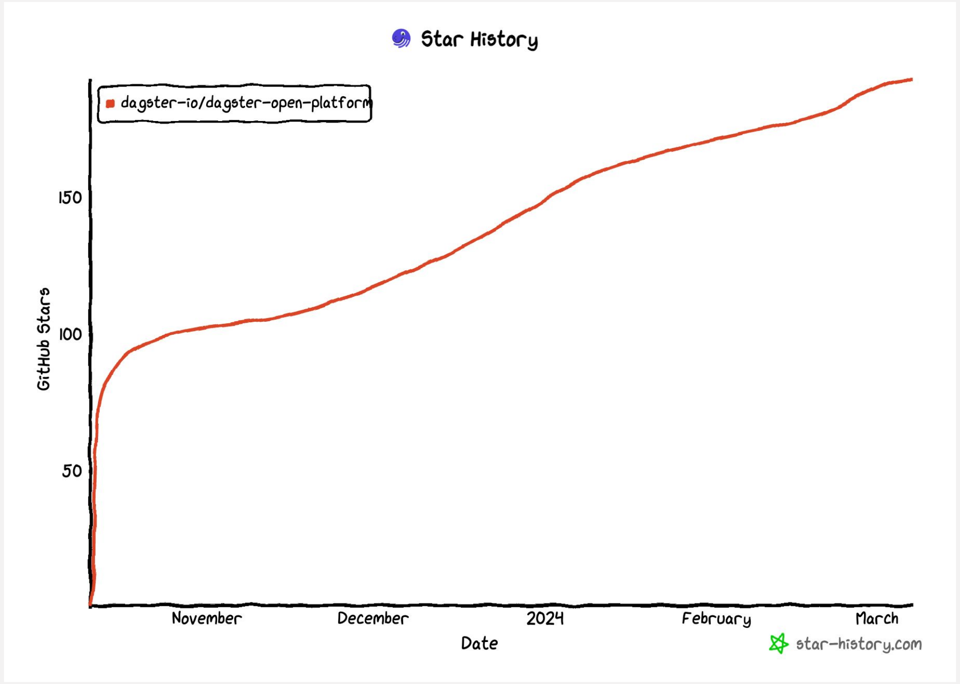
```python
def get_ignored_files() → list[str]:
    """Determines all files ignored by `openignore.txt`."""
    with open(IGNORE_FILE, "r") as f:
        spec = pathspec.PathSpec.from_lines("gitwildmatch", f)

    return list(spec.match_tree("."))
```

dagster

Make the culture shift across the company

# The reception to DOP has been **great**

- A stronger and more active community of data engineers
- Easy to link to these real-life examples during support
- More self-sufficient users
- Engineers can still easily contribute



## Summary

1. Get leadership buy-in
2. Navigate their priorities and needs
3. Understand the responsibility involved
4. Build processes and safeguards
5. Make the culture shift across the company

dagster

Data engineering is a hard and lonely journey

Dagster Open Platform is a public project for you to learn software engineering best practices from

Navigating the open source process is like building trusted analytics

# Next steps & resources

## Star the Dagster Open Platform Repository

Keep up the changes we roll out for ourselves

## Try Dagster

Python-based framework for orchestrating data pipelines

dagster.io