Python data processing libraries

and how to stitch them into a data platform

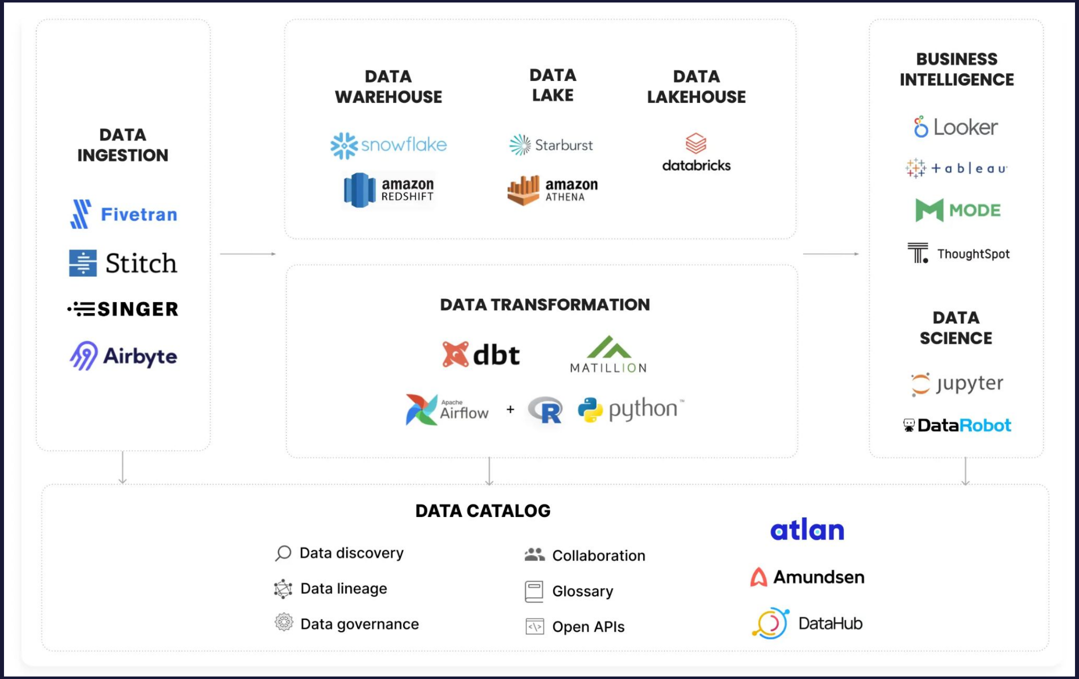
Modern Data Stack

Layers

- Data Ingestion
- Data Storage
- Data Transformation
- Data Use
- Data Governance

Language: SQL

It is not really a stack. No open standards and vendors taking it all.



New Type of User and early ML stack

New User: Data Scientist

“Stack”:

- Ingestion: Python
- Storage: csv files, blobs
- Transformations: pandas, numpy, all ML libraries
- Data visualization: notebooks, matplotlib etc.

A set of libraries glued together with Python

A set of “de facto” standards to make it a little bit easier: pandas, numpy, iPython...

dltHub

Composable data stack: same concept - for data

- Composability
- Portability (pip install data platform)
- Open “standards” (mostly de facto)
- Breaking silos
- Tool Specialization
- UI is Code (Python)

https://wesmckinney.com/blog/looking-back-15-years/

dltHub

What's there? What's missing?

- Data Representation & Storage: arrow, parquet, avro
- Table Storage: Delta, Iceberg
- Query Engine: duckdb, polars, datafusion, ibis, sqlglot …
- Data Ingestion: Python, singer, dlt
- Data Transformation: ML libs, dataframes, sqlmesh, ibis, hamilton …
- Runners, orchestrators: Airflow, temporal, modal
- Data (Power) Use: Evidence, Observable, Notebooks
- Data Governance: Nessie, Hive (catalogs), SODA (data quality), data contracts (?)

Enablers: columnar data in-memory and at rest

PARQUET

Released in 2013, Apache Parquet is an open-source **columnar storage format** designed for efficient data storage and retrieval in large-scale data processing frameworks.

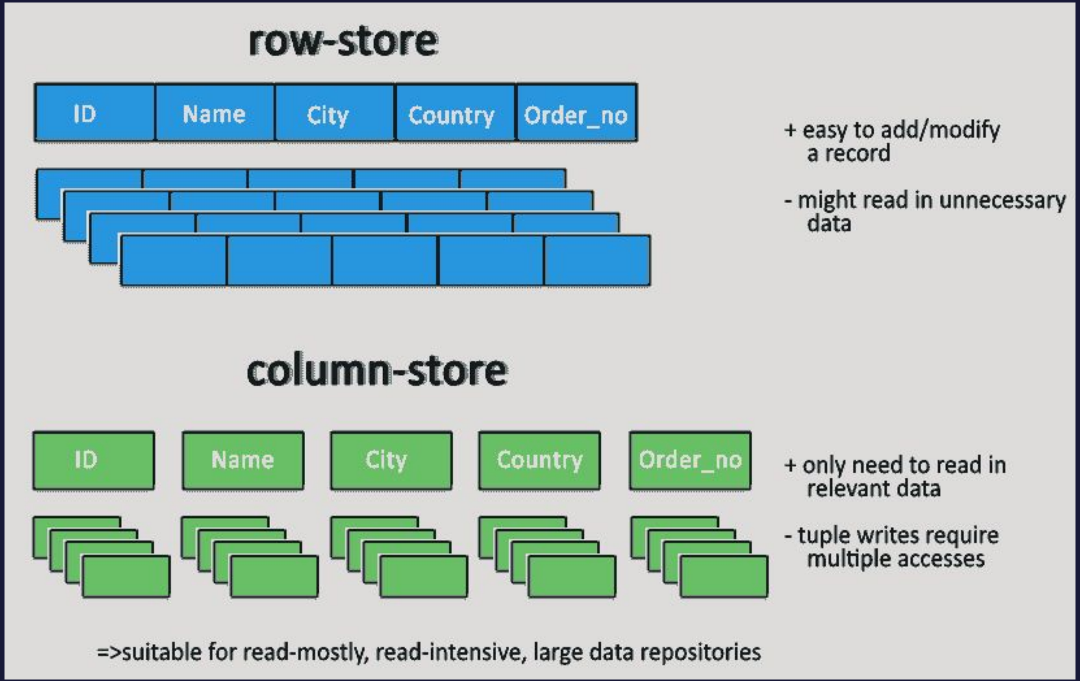
Efficient compression, Optimized IO

ARROW

Launched in 2016, Apache Arrow is an open-source **in-memory columnar data format** that facilitates high-performance analytics.

Standardized Memory Representation,

Columnar storage



dltHub

Python bindings for arrow: pyarrow

- Tons of things combined in one lib: in-memory tables, compute, parquet storage, parsers, writers, datasets, query engines, remote filesystems.
- Very well integrated with Python native objects and Pandas

```python
import pyarrow as pa
import pyarrow.parquet as pq
import pyarrow.compute as pc

data = {
    'id': [1, 2, 3, 4, 5],
    'value': [10, 20, 30, 40, 50]
}

# create a PyArrow Table from the data
table = pa.Table.from_pydict(data)
print(table.schema)

# transform the 'value' column (e.g., multiply each value by 2)
transformed_value = pc.multiply(table['value'], 2)

# replace the 'value' column with the transformed data
table = table.set_column(
    table.schema.get_field_index('value'),
    'value',
    transformed_value
)

# save the transformed table to a Parquet file
pq.write_table(table, 'transformed_data.parquet')
```

Open table formats: new industry standards

What if we want to update, delete data in parquet? Manage many tables? Evolve the schema? ACID Transactions? Petabytes of data?

ICEBERG

Created by Netflix to manage their massive data lakes, Iceberg was contributed to the Apache Software Foundation in 2018.
It is the new storage standard for warehouses. Industry adopting it.
Complicated ecosystem of vendors: query engines, catalogs.

DELTALAKE

Developed by Databricks to address the challenges of data lakes, Delta Lake was open-sourced in 2019.

Standardized Memory Representation

Open table formats: delta-rs

DELTA-RS

Python binding for rust library. Linux Foundation. Started independently from Databricks. Pretty feature complete.

- Append, replace, merge
- Schema evolution
- Table maintenance
- Seamless arrow integration

```
from deltalake.writer import write_deltalake

write_deltalake("s3://my-bucket/dataset/table", arrow_table)
```

Open table formats: pyiceberg

PYICEBERG

Supports table storage and catalogs. Apache Foundation. Lacks several fundamental features. Low level interfaces. Mandatory catalog.

Gaining a lot of momentum.

```python
from pyiceberg.io.pyarrow import PyArrowFileIO
from pyiceberg.schema import Schema
from pyiceberg.types import StructType, IntegerType, StringType
from pyiceberg.catalog import load_catalog

# Define the Iceberg schema
iceberg_schema = Schema(
    StructType()
    .add_field('id', IntegerType(), required=True)
    .add_field('value', StringType(), required=True)
)

# Load or create a catalog
catalog = load_catalog("default", uri="warehouse/path")

# Define table identifier
table_identifier = "my_namespace.my_table"

# Create a new Iceberg table
catalog.create_table(
    identifier=table_identifier,
    schema=iceberg_schema,
    partition_spec=None,
    properties={"format-version": "2"},
)

# Load the Iceberg table
table = catalog.load_table(table_identifier)

# Write the PyArrow Table to Iceberg format
file_io = PyArrowFileIO()
with table.new_write() as writer:
    writer.append(arrow_table)
```

Query engines: separated from data

A kind of innovation in data:

- Open storage and table formats enable query engines independent from data
- Move query engine where your data is (vs. data to the query engine)
- Simple, fast, portable, in-memory and hybrid. No backend

Ecosystem of interfaces and optimizers:

- Data frame expressions, to-sql compilers: ibis
- SQL parsers, optimizers, lineage: sqlglot

Query engines: duckdb and datafusion

- Fast, portable analytical database
- Scanners for parquet, iceberg, delta, postgres, json, csv...
- Also own optimized storage.
- Very good Python bindings with variety of interfaces.
- Can query arrow.
- Thriving community

```python
import duckdb

con = duckdb.connect()
con.register('my_table', arrow_table)


query = """
SELECT *, encode(binary_col, 'hex') AS hex_string
FROM my_table
"""
arrow_table = con.execute(query).arrow()
```

dltHub

Query interfaces: ibis, sqlglot

Are we back to SQL? No

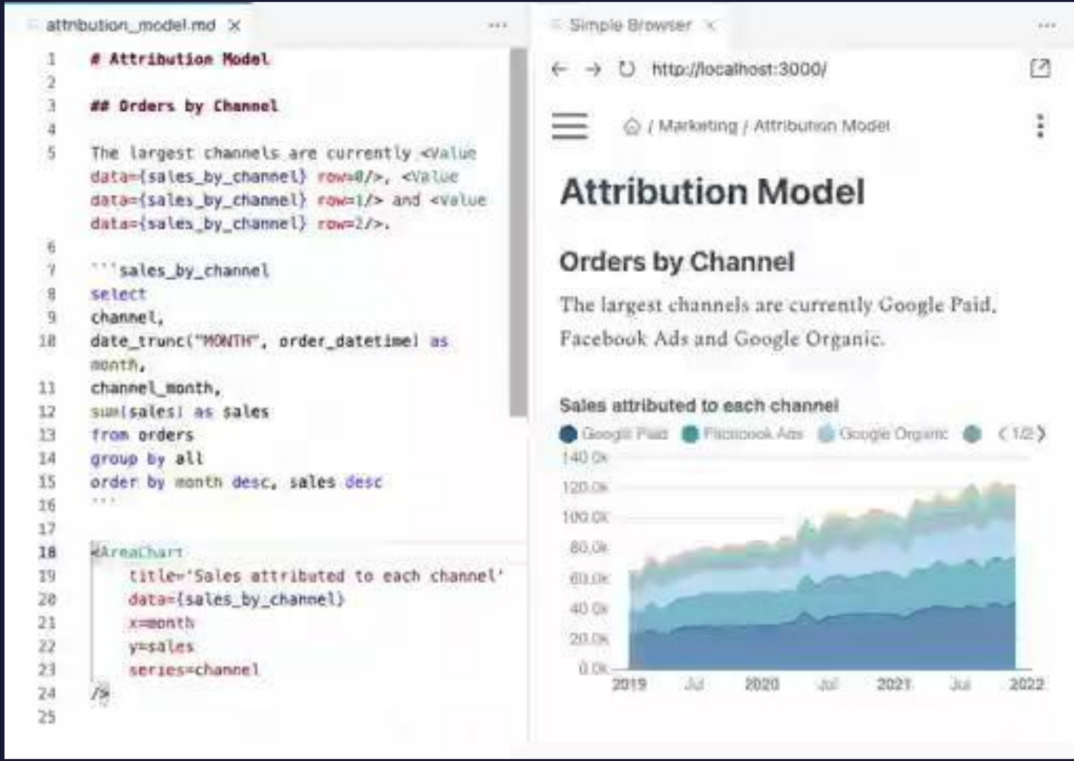
- Ibis converts data frame expressions into SQL and talks to tons of backends.
- Sqlglot and duckdb inside (query files, arrow tables)
- Lazy execution (only when materialized)
- Very composable
- Also https://github.com/data-apis

```python
expr = table.filter(table.value > 50).select('id', 'value')

expr = table.order_by(ibis.desc(table.timestamp))

expr = table.group_by('category').aggregate(
    total_amount=table.amount.sum(),
    average_amount=table.amount.mean()
)

expr = orders.join(customers, orders.customer_id == customers.id).select(
    orders.order_id,
    customers.name,
    orders.total_amount
)
```

dltHub

Shift left: data power user

- BI as code
- Share data back
- Real time updates
- Duckdb inside (WASM)
- pip install

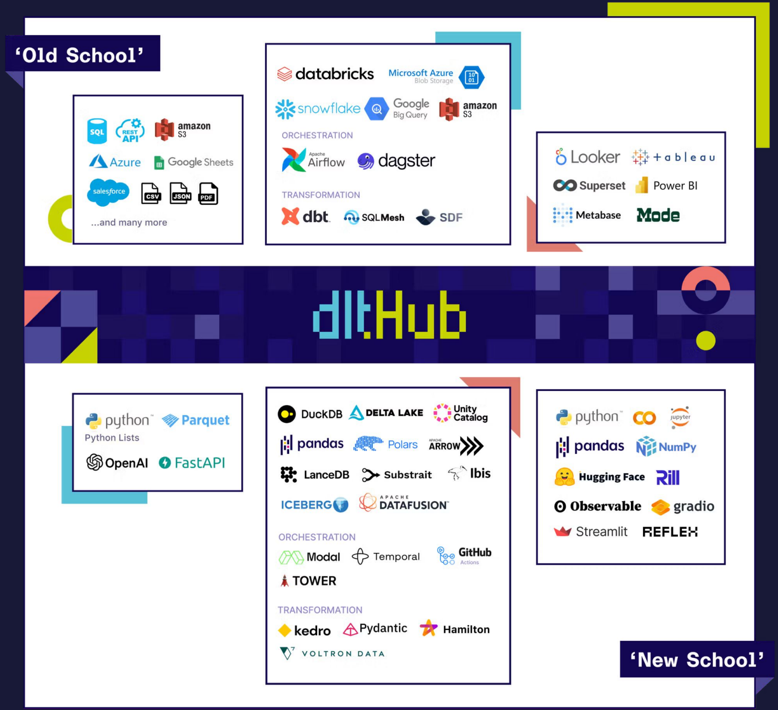


https://evidence.dev/

Stitching together data platforms: dlt

dlt is Python OS library for moving data

- Talks to modern and composable data stacks.
- Lightweight, no backend
- Automates data loading, schema inference, incrementals
-



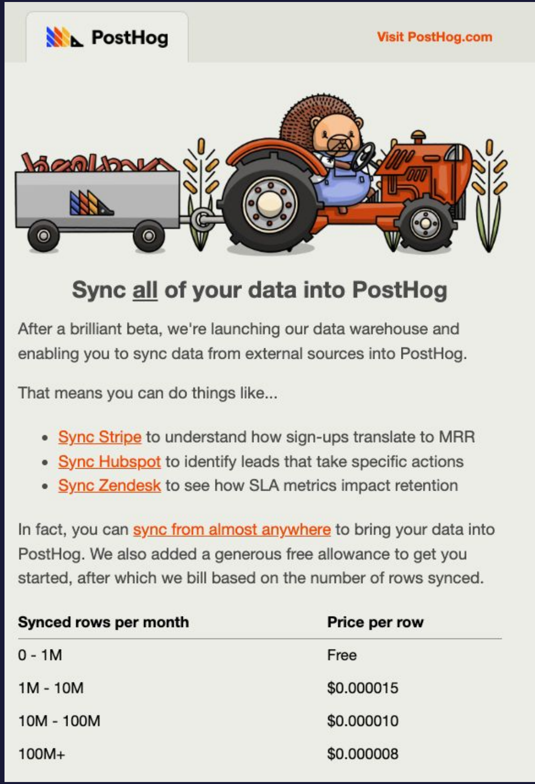
Rest api → json → parquet → iceberg@s3

- Partitioned, incremental dlt resource

```python
import dlt
from dlt.sources.helpers.rest_client import paginate as requests

@dlt.resource(
    primary_key="id",
    table_name=lambda i: i["type"],
    table_format="iceberg",
    write_disposition="append",
    columns={"_dlt_load_id": {"partition": True}},
    incremental=dlt.sources.incremental("created_at", row_order="desc"),
)
def events():
    for page in requests(
        "https://api.github.com/repos/dlt-hub/dlt/events",
        params={"per_page": 100},
    ):
        yield page
```

An example data platform: PostHog

- PostHog: all-in-one open source platform that helps +200,000 developers build successful products
- dlt & temporal are the main OSS tools that power the recently launched data warehouse product in Posthog's storage in S3
- The stack:
    - dlt to move data
    - pyarrow + delta-rs for storage and table maintenance
    - duckb and clickhouse as query engines
    - temporal to run
    - Posthog platform to explore data



PostHog

Visit PostHog.com

Sync all of your data into PostHog

After a brilliant beta, we're launching our data warehouse and enabling you to sync data from external sources into PostHog.

That means you can do things like...

- Sync Stripe to understand how sign-ups translate to MRR
- Sync Hubspot to identify leads that take specific actions
- Sync Zendesk to see how SLA metrics impact retention

In fact, you can sync from almost anywhere to bring your data into PostHog. We also added a generous free allowance to get you started, after which we bill based on the number of rows synced.

Synced rows per month	Price per row
0 - 1M	Free
1M - 10M	$0.000015
10M - 100M	$0.000010
100M+	$0.000008

dlt+ demo

dlt+ is dlt for data platform teams