

Data Engineering and MLOps

- [DVC](#)
- [Model Registry](#)
- [Data Wrangling](#)
- [DBT](#)
- [Data Loading with Airbyte](#)

DVC

DVC or [Data Version Control](#) is an open source tool for managing data assets. It is very useful but also can be quite overwhelming to use.

The main use cases I've found for DVC are:

1. Keeping large data assets (e.g. machine learning datasets) version controlled alongside code so that you always know where the latest version of some project specific training data can be found.
2. Making sure that all intermediate and final outputs from experiments are reproducible. This is always good to know when a client inevitably asks you "ok how did you get **Y** surprising result?" or "can you just confirm that you included **X** features in your model?" DVC helps by:
 1. Keeping track of file hashes used at each stage in a pipeline
 2. Keeping a copy of the file content at each stage in the pipeline.

Model Registry

A model registry is a service that provides version-control-like behaviour for ML models. There are a number of open source and commercial model registries.

MLFlow

[MLFlow](#) is an open source model registry that provides a bunch of features including model version registration and result storage.

SnowPark Registry

SnowPark is a container runtime environment inside Snowflake and it provides Model Registry functionality which is documented [here](#).

Data Wrangling

DuckDB

[DuckDB](#) is a lightweight OLAP type database system written in C++ and designed to be used for EDA style activities:

When to use DuckDB



- Processing and storing tabular datasets, e.g. from CSV or Parquet files
- Interactive data analysis, e.g. Joining & aggregate multiple large tables
- Concurrent large changes, to multiple large tables, e.g. appending rows, adding/removing/updating columns
- Large result set transfer to client

When to not use DuckDB



- High-volume transactional use cases (e.g. tracking orders in a webshop)
- Large client/server installations for centralized enterprise data warehousing
- Writing to a single database from multiple concurrent processes

From their website: advice on when to use and not to use DuckDB

Polars

[Polars](#) is a rust-based data frames library with Python bindings



Today I discovered, to my surprise, that vectorized string operations in #pandas are slower than raw #python. I thought it was the opposite!

You know what's faster than both?
#polars!

```

100 report pandas as pd
101
102 report pandas as pl
103
104 report random
105
106 In [41]: list_of_strings = [
107     ...:     f"{20*random.randrange(8, 1000000)}-{random.randrange(1, 1000000)}-{random.randrange(1,
108     ...:     2000000)}"
109     ...:     for _ in range(1, 800, 800)]
110     ...: ]
111
112 In [51]: Memory
113     ...: s = pd.Series(list_of_strings).str.split("@").str[0].astype(int)
114     ...:
115 400 ms ± 3.89 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
116
117 In [52]: Memory
118     ...: s = pd.Series(list_of_strings).str.split("@").str[0].astype(int)
119     ...:
120 400 ms ± 3.97 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
121
122 In [53]: Memory
123     ...: s = pd.Series(list_of_strings).str.split("@").str.get(2).astype(int)
124     ...:
125 125 ms ± 519 μs per loop (mean ± std. dev. of 7 runs, 10 loops each)

```

0 0 0 0

[Here is a talk](#) that Juan Luis gave about the library

DBT

[DBT](#) is a data transformation tool with a SaaS platform and an open-core command line tool.

The tool is [widely used](#) to put the T in ELT.

Robin Moffat has written [a walkthrough/guide](#) on how he used DBT with [DuckDB](#)

Data Loading with Airbyte

Airbyte is a FOSS tool for mass data import and export when working with common flavours of SQL and OLAP databases.