



Tabular Feature Pipelines at Scale

Tabular and time series 01 | Fit/transform discipline, point-in-time correct training sets (as-of joins), aggregation features on tens of millions of rows with Polars, DuckDB and Spark, memory budgets, train/serve skew and data contracts (feature ideas themselves: data 03; feature-store products: mlops 06)

#Tabular #FeaturePipelines #PointInTime #Polars #DuckDB

Why pipelines, not notebooks

A tabular feature pipeline

In one sentence

●○○○

Code that turns raw tables into one row per prediction, the same way for training and for live use.

Practitioner

●●○○

Every learned step (imputation, scaling, encoding) is fitted on training rows only and replayed frozen on validation, test and production.

- Pipeline + ColumnTransformer
- fit inside each CV fold

Expert

●●●○

Each training row is anchored at a prediction time; features come from data stamped before it, labels from the window after it.

- as-of joins
- label window
- data latency

Under the hood

●●●●

At scale the pipeline is a versioned query plan over Parquet with a schema contract, run in batch for training and backfills, mirrored online for serving.

- lazy engines
- contracts and freshness checks

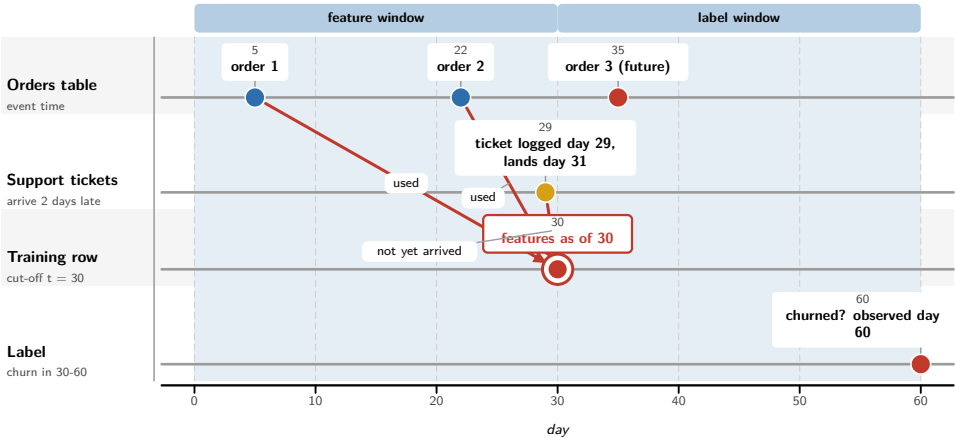
Analogy

A pipeline is a recipe card; a notebook is a cook's memory of what they did last Tuesday.

Before Feature pipelines at scale, can you explain...

	Not yet	Partly	Yes
MUST KNOW Train / validation / test and leakage catalogue Learn: machine-learning 08	☐	☐	☐
MUST KNOW pandas or Polars groupby and joins Learn: programming 03	☐	☐	☐
MUST KNOW Feature ideas: scaling, binning, dates, interactions Learn: data 03	☐	☐	☐
HELPFUL Parquet, Arrow, partitioning Learn: data 07	☐	☐	☐
HELPFUL SQL window functions Learn: programming 10	☐	☐	☐
ADVANCED Spark or Ray Data Learn: tools-and-platforms 02-04	☐	☐	☐

Point-in-time correct training sets



One training row for customer 42 with prediction time day 30: features read only events before day 30 (and only what had arrived by then), the label reads days 30-60

Leak	How it happens	Rule
Snapshot leak	joining today's customer table (current plan, current balance) to last year's rows	join history tables by validity interval, never current snapshots
Future events	aggregating all orders of a customer, including orders after the cut-off	filter event_time < cutoff before aggregating
Late-arriving data	using rows by event time that only landed in the warehouse later	filter on ingestion time ≤ cutoff, or lag by the known delay
Same-timestamp leak	as-of join with ties lets the label event join itself	strict inequality (allow_exact_matches=False)
Label in a feature	status column updated when the outcome happens (account_closed)	drop columns written by the outcome process
Duplicate entities across splits	same customer in train and test through several rows	group split by entity id (ML 08)

Leaks that only appear when tables are joined, and the rule that blocks each

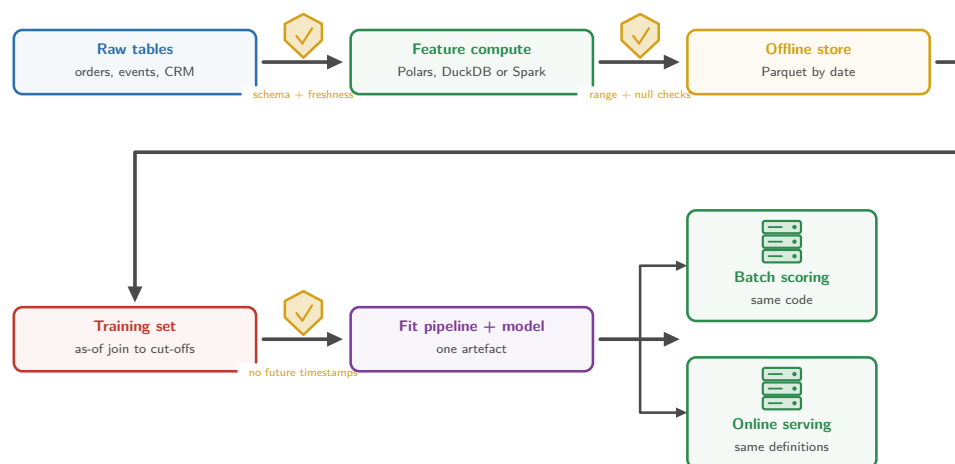
As-of join in three engines (feature value valid at each row's cut-off)

```
# pandas: both frames sorted by the time key
train = pd.merge_asof(rows.sort_values("cutoff"),
    feats.sort_values("ts"),
    left_on="cutoff", right_on="ts", by="customer_id",
    direction="backward", allow_exact_matches=False)

# Polars (sorted by time within each group)
train = rows.join_asof(feats, left_on="cutoff", right_on="ts",
    by="customer_id", strategy="backward")

# DuckDB SQL
SELECT r.*, f.balance FROM rows r
ASOF JOIN feats f ON r.customer_id = f.customer_id AND r.cutoff > f.ts;
```

⚡ Pipelines that scale



Batch feature pipeline: one definition feeds training (point-in-time) and serving; the shields are the checks that stop bad data

scikit-learn: every learned step inside one Pipeline, so CV refits it per fold

```
from sklearn.compose import ColumnTransformer, make_column_selector as sel
from sklearn.pipeline import make_pipeline
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import OneHotEncoder, TargetEncoder
from sklearn.ensemble import HistGradientBoostingClassifier

pre = ColumnTransformer([
    ("num", SimpleImputer(strategy="median"), sel(dtype_include="number")),
    ("low", OneHotEncoder(handle_unknown="infrequent_if_exist",
        min_frequency=20, sparse_output=False),
        ["plan", "device"]),
    ("high", TargetEncoder(), ["city", "merchant_id"]), # sheet 02
], verbose_feature_names_out=False).set_output(transform="pandas")
model = make_pipeline(pre, HistGradientBoostingClassifier())
# cross_val_score(model, X, y, cv=GroupKFold(5), groups=X.customer_id)
```

Polars lazy: aggregation features per entity before a cut-off, streamed from Parquet

```
import polars as pl
cutoff = pl.datetime(2026, 6, 30)
feats = (pl.scan_parquet("orders/*.parquet")
    .filter(pl.col("ts") < cutoff) # nothing read yet
    .group_by("customer_id") # pushed down to the scan
    .agg(n_orders=pl.len(),
        spend_90d=pl.col("amount")
            .filter(pl.col("ts") >= cutoff - pl.duration(days=90)).sum(),
        days_since=cutoff - pl.col("ts").max()).dt.total_days()))
feats.collect(engine="streaming").write_parquet("feats_2026-06-30.parquet")
```

Rows of raw events	Engine	Why
< ~10 M	pandas or Polars eager	fits in memory; simplest to debug
~10 M - 1 B	Polars lazy / streaming, DuckDB	column pruning, predicate push-down, out-of-core execution on one node
> ~1 B or many joins	Spark, Ray Data, warehouse SQL	distributed shuffles; watch key skew (salt hot keys, broadcast small tables)
Any size, many cut-offs	one aggregation per cut-off date, partitioned output	backfills become embarrassingly parallel by date

Engine by data size (single machine with approx. 64 GB RAM; orders of magnitude, measure on your data)

🛡️ Memory, skew and contracts

Will the training matrix fit?

PROBLEM

50 million rows, 200 numeric features. Memory of the dense matrix in float64, in float32, and as the 1-byte histogram bins a GBM builds ($\text{max_bin} \leq 255$)?

1. Cells

$\Rightarrow R \times F = 5.0e7 \times 200 = 1.0e10$ values
why: one value per row and feature

2. float64

$\Rightarrow 1.0e10 \times 8 = 8.0e10 \text{ B} = 80 \text{ GB}$
why: pandas default for floats

3. float32

$\Rightarrow 1.0e10 \times 4 = 40 \text{ GB}$
why: enough precision for features; halve memory

4. Binned

$\Rightarrow 1.0e10 \times 1 = 10 \text{ GB}$
why: LightGBM/XGBoost hist keep bin indices, not raw floats

GIVEN

- $M = R \times F \times b$, where M = memory in bytes, R = rows, F = features, b = bytes per value
- R = 50,000,000; F = 200; b = 8 (float64), 4 (float32), 1 (uint8 bin index)

ANSWER

80 GB in float64, 40 GB in float32, about 10 GB once binned: cast to float32 and build the booster's dataset straight from Parquet in chunks

Check: A 64 GB machine cannot hold the float64 copy plus the bins plus a pandas copy; peak memory is often 2-3 $\times$ the final matrix

churn_training_2026-06-30						
4.2 M rows 8 columns						
Column	Type	Missing	Distinct	Distribution	Issues	
customer_id	id	0%	4200.0k		● group key for CV	
cutoff	datetime	0%	12			
plan	categorical	0%	4	basic pro team legacy		
city	categorical	3.0%	18.5k		● high cardinality: sheet 02	
spend_90d	float	0%	950.0k		● right-skewed	
last_ticket_days	float	61%	365		● mostly missing	
account_closed	bool	0%	2		● written by the outcome: leak	
churn_30d	bool	0%	2	0 1		

Profile the training table before the first fit: the flags point at the next pipeline step

✓ baseline / mandatory □ recommended – verify in your context

- ✓ Schema: column names and dtypes match the contract `pandera`, Great Expectations or dbt tests
- ✓ Freshness: newest partition within the allowed delay
- ✓ No event timestamp after the cut-off in any feature table
- ✓ Row count within $\pm \times \%$ of the previous run
- **SHOULD** Null share and value ranges per feature within bounds
- **SHOULD** Category sets: new levels counted, not silently dropped
- **SHOULD** Online vs offline feature values compared on a sample (skew check) `log served features`, `recompute offline`, `diff`

⚠ Train/serve skew is the silent failure

Two implementations of the same feature (SQL for training, Python in the service) drift apart: different time zones, different window ends, nulls filled differently. Keep one definition, log the features the service actually used, and compare them daily with the offline recomputation.

● Myths and recap

YOU MAY HEAR

"A random train/test split is fine if I drop the ID column"

BUSTED

✓ ACTUALLY

Rows of the same entity still leak across the split

Split by entity (GroupKFold) or by time.

YOU MAY HEAR

"Joining the latest customer table is good enough for training"

BUSTED

✓ ACTUALLY

It leaks values from after the prediction time

Use history tables and as-of joins.

YOU MAY HEAR

"Big data needs Spark from day one"

BUSTED

✓ ACTUALLY

Polars or DuckDB handle hundreds of millions of rows on one node

Lazy scans prune columns and stream; distribute only when one node is not enough.

YOU MAY HEAR

"float64 is required for accurate features"

BUSTED

✓ ACTUALLY

float32 is enough for model inputs

Trees bin features into at most a few hundred values anyway.

Fit on train, replay frozen
Pipeline + ColumnTransformer

Anchor every row at a cut-off
as-of joins, strict <

Filter by arrival time
late data is future data

Lazy engines first
Polars, DuckDB; Spark when needed

Budget memory
rows × features × bytes; float32

Contracts and skew checks
schema, freshness, online vs offline

Q1 Why must a scaler be fitted inside each CV fold?

A Its statistics would otherwise include the validation fold, leaking information and inflating scores.

Q2 What does `allow_exact_matches=False` do in `pd.merge_asof`?

A Excludes a right row with the same timestamp, so an event cannot join to itself at the cut-off.

Q3 Memory of 10 M rows × 100 float32 features?

A $10e6 \times 100 \times 4 \text{ B} = 4 \text{ GB}$.

Q4 What is a label window?

A The period after the prediction time where the outcome is observed (e.g. churn in the next 30 days).

Q5 How do you avoid late-arriving-data leakage?

A Filter on ingestion time $\leq$ cut-off, or lag features by the known delivery delay.

Q6 What does Polars' `scan_parquet + collect(engine='streaming')` give you?

A A lazy plan with predicate and column push-down, executed in batches that need not fit in RAM.

Q7 How do you fix a skewed Spark join?

A Broadcast the small table, or salt the hot keys (add a random suffix, replicate the other side).

Q8 What is train/serve skew?

A A feature computed differently in training and serving; detected by logging served features and comparing them offline.

Q9 Why does a GBM reduce memory after binning?

A It stores one small bin index per value (≤ 255 bins $\rightarrow$ 1 byte) instead of the raw float.