Sri Sathya Sai Institute of Higher Learning
Department of Mathematics and Computing
B.S. Honours Mathematics

Suvam Pradhan

Registration No: _24010107010_
3rd Year BS Mathematics
COURSE- Data Visualization

Data pipeline: raw data → transformation → enrichment

1. Raw data

Definition. Raw data is whatever arrives at the first point you touch it — before you compute, reshape, retype, or join anything. It's relative to your pipeline's boundary, not an absolute property of a file: the output of one stage is the raw input of the next stage downstream.

Note. Manual cleanup before a file ever reaches your script (fixing a typo in Excel, deleting a few rows by hand) is still a transformation — it just has no audit trail, which breaks reproducibility. If a value's origin requires you to recall a manual edit, it isn't raw anymore.

```
import pandas as pd

df = pd.read_csv("orders_raw.csv")   # <- this is the raw boundary
for this pipeline
```

2. Transformation stage

Definition. Operations that rearrange or summarize data already present in the dataset. No new information enters — you're making existing information usable.

Note. Type conversion runs first, always — every later step (dates, aggregation, pivoting) depends on correctly-typed columns. A "date" stored as text won't sort or filter correctly.

2.1 Type conversion

```
df = df.astype({"amount": "float64"})
df["order_date"] = pd.to_datetime(df["order_date"], format="%Y-%m-
%d")
df["region"] = df["region"].astype("category")
```

2.2 Aggregation

Note. Aggregation reduces row count by grouping. It summarizes existing values (sum, mean, count) — it doesn't introduce anything the dataset didn't already imply.

```
df_agg = (
    df.groupby(["region", "month"])
      .agg(revenue=("amount", "sum"),
           order_ct=("amount", "count"),
           avg_order=("amount", "mean"))
```

```
      .reset_index()
)
```

2.3 Pivoting (long ↔ wide)

Note. Pivoting reshapes columns into rows or vice versa. Same information, different layout — nothing is added or lost.

```
# long -> wide
df_wide = df.pivot(index="region", columns="month",
values="revenue")

# wide -> long
df_long = df_wide.reset_index().melt(id_vars="region",
var_name="month", value_name="revenue")
```

3. Enrichment stage

Definition. Actions that net new information into the dataset: new fields the original data didn't contain, or new records from another source. Three primary types: unions and joins, metadata insertion, and derivation of values.

Note. The join or derivation itself is never the risk — the risk is leakage: joining or deriving a value that reflects information not actually knowable as of the date/event on the row it's attached to. Flagged inline below wherever it applies.

3.1 Unions — stacking records vertically

Definition. Appending records from multiple sources with the same schema (e.g. monthly reports into one quarterly table). No key matching — just vertical stacking.

Note. Unions don't error on mismatched column names — pd.concat silently unions the column sets and fills the gap with NaN. Standardize names/types before the union.

```
import glob, os

quarterly = pd.concat(
    [pd.read_csv(f) for f in glob.glob("orders/*.csv")],
    ignore_index=True
)
```

3.2 Joins — linking records via a shared key

Definition. The most common enrichment action: linking two datasets on a key field (e.g. customer_id) so an analyst can relate one dataset's fields to another's.

Note — static lookup joins are safe. A join to a table that doesn't change over time (region metadata, product category) carries no leakage risk.

```
df_enriched = df.merge(region_metadata, on="region", how="left")
```

Note — time-varying joins are where leakage lives. Joining a customer's current status onto historical transactions makes every past row look like it belongs to whatever tier the customer holds today, even transactions from before they reached that tier.

```
# LEAKS: current_tier reflects information from the future relative
to order_date
df_leaky = transactions.merge(
```

```
    customers[["customer_id", "current_tier"]], on="customer_id",
how="left"
)

# FIX: as-of join -- only match status effective on or before the
transaction date
transactions_sorted = transactions.sort_values("order_date")
status_sorted =
customer_status_history.sort_values("effective_date")

df_correct = pd.merge_asof(
    transactions_sorted,
    status_sorted,
    left_on="order_date",
    right_on="effective_date",
    by="customer_id",
    direction="backward"
)
```

3.3 Metadata insertion

Definition. Adding fields that describe a record's origin rather than its content: source filename, load timestamp, row number, lineage.

Note. Skipped most often because it doesn't show up in the visualization — but it's what lets you trace a wrong number back to its source file six months later.

```
quarterly = pd.concat(
    [pd.read_csv(f).assign(source_file=os.path.basename(f),
                           load_timestamp=pd.Timestamp.now())
     for f in glob.glob("orders/*.csv")],
    ignore_index=True
)
```

3.4 Derivation of values

Definition. Computing new fields from data already present, to give a more direct perspective for modeling or visualization. Two groups: generic derivations (broadly applicable) and proprietary derivations (organization-specific).

Note. This is the category that gets misfiled as "transformation" most often — date-part extraction, binning, and normalization all create a field, which by definition is enrichment, not structuring, even though nothing external was joined in.

Generic — time

```
df["weekday"] = df["order_date"].dt.day_name()
df["quarter"] = df["order_date"].dt.quarter
df["utc_timestamp"] = df["order_date"].dt.tz_localize("UTC")
```

Generic — geography

```
# cache external lookups -- don't hit an API on every refresh
import pickle
if not os.path.exists("geocode_cache.pkl"):
```

```
    geocodes = {city: geocode_api_call(city) for city in
df["city"].unique()}
    pickle.dump(geocodes, open("geocode_cache.pkl", "wb"))
geocode_cache = pickle.load(open("geocode_cache.pkl", "rb"))
df["lat_lon"] = df["city"].map(geocode_cache)
```

Generic — text (NLP)

```
from textblob import TextBlob
df["sentiment_score"] = df["review_text"].apply(lambda t:
TextBlob(t).sentiment.polarity)
```

Generic — numeric

```
df["order_size_bucket"] = pd.cut(
    df["amount"],
    bins=[-float("inf"), 50, 200, 500, float("inf")],
    labels=["small", "medium", "large", "enterprise"]
)
df["revenue_z"] = (df["revenue"] - df["revenue"].mean()) /
df["revenue"].std()
```

Note — aggregate derivations leak the same way time-varying joins do. A lifetime-value figure computed across all a customer's transactions and joined back onto each transaction row leaks future information into every past row.

```
# LEAKS
customer_ltv = transactions.groupby("customer_id")
["amount"].sum().rename("ltv")
df_leaky = transactions.merge(customer_ltv, on="customer_id",
how="left")

# FIX -- running total using only prior rows per customer
transactions = transactions.sort_values(["customer_id",
"order_date"])
transactions["ltv_to_date"] = (
    transactions.groupby("customer_id")["amount"]
                .apply(lambda s: s.shift().cumsum().fillna(0))
                .reset_index(level=0, drop=True)
)
```

Proprietary — brief note only. Organization-specific models (e.g. churn-likelihood scores), usually implemented as user-defined functions in a big-data system. No generic example — by definition these are domain-specific and not broadly reusable.

3.5 ML-specific numeric/categorical encodings

Definition. The scikit-learn preprocessing taxonomy (discretization, ordinal, one-hot, normalization, standardization, power, quantile transforms) answers a different question than the sections above: how do you encode a variable so a model can consume it, not how do you get information into a dataset for a chart. It overlaps with derivation of values above (binning = discretization, min-max = normalization, z-score = standardization) but adds four transforms not yet covered.

Note. All four still count as enrichment under this document's taxonomy — each creates a new field. Whether you need them depends on whether the downstream consumer is a model or a chart; one-hot encoding a region into a dozen dummy columns is standard before a model, but a bar chart just uses the categorical column directly.

```
from sklearn.preprocessing import OrdinalEncoder, OneHotEncoder,
PowerTransformer, QuantileTransformer

# Ordinal -- categorical to integer codes. Only use when the
category has a real order
# (small < medium < large) -- otherwise this imposes a false
ranking a model can pick up on.
ordinal_enc = OrdinalEncoder()
df["size_ordinal"] =
ordinal_enc.fit_transform(df[["order_size_bucket"]])

# One-hot -- categorical to binary dummy columns. Safe default for
unordered categories.
onehot_enc = OneHotEncoder(sparse_output=False)
onehot_cols = onehot_enc.fit_transform(df[["region"]])
df_onehot = pd.DataFrame(onehot_cols,
columns=onehot_enc.get_feature_names_out(["region"]))
df = pd.concat([df, df_onehot], axis=1)

# Power transform -- reshapes a skewed distribution toward
Gaussian.
# Yeo-Johnson handles negative values; Box-Cox requires strictly
positive data.
pt = PowerTransformer(method="yeo-johnson")
df["revenue_power"] = pt.fit_transform(df[["revenue"]])

# Quantile transform -- forces a uniform or Gaussian distribution
regardless of the
# original shape. More aggressive than power transform; distorts
relationships between
# values more, so use when you specifically need the target
distribution, not by default.
qt = QuantileTransformer(output_distribution="normal",
n_quantiles=100)
df["revenue_quantile"] = qt.fit_transform(df[["revenue"]])
```

Rule of thumb

Before any enrichment step — join or derivation — ask: could this value have changed between the event on this row and now? If yes, it needs a dated/history table and an as-of filter (or a running aggregate), not a join to current state or a full-history aggregate.

Data Pipeline Flow: From Raw to Enriched

RAW DATA

This pipeline tracks data from its "Raw" entry point through two distinct phases:

Transformation (structuring what you have) and **Enrichment** (adding what you don't). The goal is to ensure data reproducibility while avoiding "leakage"—the accidental inclusion of future information in historical records.

TRANSFORMATION STAGE (STRUCTURING)

Type Conversion: The Critical First Step

Every later step—dates, aggregation, and piccting—depends on columns being correctly typed to function.

Reshape Without Adding New Data

Transformations like plvoting or aggregation rearrange or summarize existing info without introducing new fields.

Maintain the Audit Trail

Manual edits (like fixing types in Excel) break reproducibility; use scripted transformations instead.

ENRICHMENT STAGE (ADDING VALUE)

Net New Information into the Dataset

Enrichment adds new fields or records via esternat joins, metadata insertion, or derived values.

Derivations Create New Fields

Computing date-parts or tims counts as enrichment besouse it creates fields not present in rew date.

The Rule of Thumb: Avoid Leakage

Do not join current status onto historical rows, it creates "impossible" views of past events.

ENRICHMENT METHODS

Unions
(Stacking records vertically)

Use for combining periodic reports.

Joins
(Linking records via a shared key)

Use for connecting customers to transactions.

Metadata
(Adding origin info / lineage)

Essential for tracing errers back to source files.

ENRICHED DATA