Data Preprocessing – 2

Data Integration

Data Integration is the process of combining data from different sources to provide a unified view.

Data Integration Challenges

Challenge	Explanation
Schema mismatch	Different datasets may have inconsistent structures (e.g., cust_id vs customer_id)
Data format inconsistency	One source may use DD/MM/YYYY and another MM-DD-YYYY for dates
Duplicate entries	Same entity appears in multiple datasets, e.g., same customer listed twice
Conflicting data	Same attribute has different values across sources (e.g., age = 25 vs 26)
Missing keys	No common attribute to join on (foreign/primary key mismatch)
Data redundancy	Repetitive or duplicate information inflates storage and reduces accuracy
Semantic heterogeneity	Same word used differently or different words used for the same thing

Adding Attributes (Horizontal Integration)

- **Definition**: Enriching a dataset by adding new **features (columns)** from another source.

- **Example**:
  - Dataset A: student_id, name
  - Dataset B: student_id, GPA
  - Merged: student_id, name, GPA

✅ *Useful when datasets share a common key.*

Adding Data Objects (Vertical Integration)

- **Definition**: Appending **rows (records)** from another dataset with the same schema.
- **Example**:
  - Dataset A: Data for Branch X students
  - Dataset B: Data for Branch Y students
  - Combined: Data for all students

✅ *Ensure column structure matches.*

List of essential pandas functions for data integration

1. **pd.merge()** – SQL-style JOIN

   pd.merge(df1, df2, on='key', how='inner')

how= option	Meaning
'inner'	Only matching records
'left'	All from df1, matching from df2
'right'	All from df2, matching from df1
'outer'	All records from both

2. **pd.concat()** – Stack Datasets Vertically/Side-by-side

   pd.concat([df1, df2], axis=0) # row-wise

   pd.concat([df1, df2], axis=1) # column-wise
3. **df.join()** – Merge by Index

   df1.join(df2, how='left')

4. **df.drop_duplicates()** – Remove Duplicate Rows

   df.drop_duplicates(subset=['id'], keep='first')

5. **df.duplicated() - Detect duplicate rows**

   df[df.duplicated()]

6. **df.fillna()** – Fill Missing Values

   df['score'].fillna(0)

   df['gender'].fillna(df['gender'].mode()[0])

7. **df.astype()** – Convert Data Types

   df['score'] = df['score'].astype(int)

8. **pd.to_datetime()** – Convert to DateTime

   df['date'] = pd.to_datetime(df['date'], errors='coerce')

9. **df.rename()** – Rename Columns

   df.rename(columns={'EmpID': 'employee_id'})

10. **df.groupby()** – Aggregate Before Integration

    df.groupby('department')['salary'].mean()

11. **df.set_index() / reset_index()** – Manage Indexes

    df.set_index('id', inplace=True)

    df.reset_index(inplace=True)

12. **df.replace()** – Fix Inconsistent Values Before Merge

    df['gender'] = df['gender'].replace({'M': 'Male', 'F': 'Female'})

Function	Purpose	Common Use Case
pd.merge()	Horizontal integration	Combine customer and order data
pd.concat()	Vertical/column stacking	Combine monthly files
join()	Index-based merging	Combine time series data
drop_duplicates()	Remove repeated records	Student registration cleanup
fillna()	Fill missing values	Replace missing marks or genders

astype()	Fix data types	Convert strings to numbers
to_datetime()	Convert strings to datetime	Parse inconsistent date formats
rename()	Rename columns	Fix mismatched column names
groupby()	Summarize/aggregate	Departmental salary summary
set_index()	Index management	Prepare for index-based joins
replace()	Fix inconsistent data values	Normalize categorical entries before merge
align()	Align structure	Match shapes of two datasets
isin()	Filter rows	Keep only overlapping students

Types of Schema Matching in Integration

Type	Description	Example
Name mismatch	Same column has different names in each dataset	Emp_ID vs EmployeeID
Data type mismatch	Same column but different data types	salary as string vs numeric
Format mismatch	Values in different formats	01-01-2023 vs 2023/01/01
Structure mismatch	Column exists in one table but not the other	address only in one dataset
Value inconsistency	Same field with different case/abbreviations	'M', 'male', 'MALE'

🎓 Classroom Activity Prompt

Task: You are given two student datasets — one from the **admission portal** and one from the **exam department**.

Admission Dataset:

- student_id, student_name, branch

Exam Dataset:

- StudentID, name, Branch, GPA

Student Tasks:

1. Identify and resolve schema mismatches.
2. Standardize column names and formats.
3. Merge both datasets into one master record.
4. Handle missing or inconsistent GPA values.

📝 Scenario: Schema Matching in Integration

🗂 You have two datasets to integrate:

◆ **employees.csv**

Emp_ID	Name	Department	Salary
101	Alice	HR	55000
102	Bob	Sales	60000

◆ **payroll.csv**

EmployeeID	Emp_Name	Dept	Salary ($)
101	Alice	HR	55000.00
102	Bob	Sales	60000.00

⚠ Issues You'll Encounter:

Issue Type	Explanation	Resolution
Name mismatch	Emp_ID ≠ EmployeeID	Rename column before merge
Column name mismatch	Name ≠ Emp_Name	Rename column to match
Salary format mismatch	Salary vs Salary ($) (float/format)	Strip $, convert to float
Extra spaces	Might be hidden in headers	Use .strip() or clean headers

Integration Challenge (Mini Project)

📝 Scenario:

You're building a single dataset from these files:

- users.csv: user_id, name, email
- transactions.csv: transaction_id, user_id, amount
- logins.csv: user_id, last_login, login_count

🔧 Tasks:

1. Merge all datasets using user_id.
2. Drop users with no transaction history.
3. Fill missing login_count with the mean value.
4. Convert last_login to datetime.
5. Create a flag is_active → if last_login within last 30 days.

Combine Orders with Products

📝 Scenario:

You have:

- orders.csv: order_id, product_id, quantity
- products.csv: product_id, product_name, price

🔧 Tasks:

1. Merge datasets using product_id.
2. Calculate a new column total_price = quantity × price.
3. Find orders with missing product information.
4. Export final order summary to CSV.

Data Reduction

Data reduction is the process of reducing the **volume** but producing the **same or nearly the same analytical results**.

Objectives of Data Reduction

1. ✅ Reduce storage space
2. ⚡ Improve algorithm efficiency
3. □ Simplify data visualization & interpretation
4. 🔍 Highlight important patterns or variables
5. 🚀 Improve model generalization (less overfitting)

Data Reduction Techniques

1. Numerosity Reduction

Reduces data volume **by replacing or summarizing data** with smaller representations.

✅ Methods:

- **Parametric models**: Fit the data into a model (e.g., regression)
- **Clustering**: Group similar data points; store only cluster centers
- **Sampling**: Use a representative subset of the data
- **Aggregation**: Use summaries like mean, count, min, max
- **Histograms/Binning**: Represent data distribution in ranges

📘 Example:

- Instead of storing all sensor data from every second, keep **5-minute averages**

2. Dimensionality Reduction

Reduces the **number of features (columns)** used to represent data.

✅ *Methods:*

- **Principal Component Analysis (PCA)**: Transforms correlated variables into uncorrelated components
- **Feature selection**: Select only important features
- **Autoencoders**: Neural network-based reduction

📘 *Example:*

- In a medical dataset with 200 lab features, PCA reduces them to **10 principal components** while retaining 95% variance

Classroom Activity Suggestion

Title: "Data Shrinking Detective"

- Dataset: Titanic or any sales dataset
- Tasks:
  - Reduce dataset to 30% using sampling
  - Aggregate survival by class and gender
  - Drop features with >95% missing or zero variance

Pandas functions used for **data reduction**

A. Sampling – Reduce Rows

☐ **1. df.sample()**

- **Use**: Randomly sample a fraction or fixed number of rows

- **Example**:

```
df_sample = df.sample(frac=0.1, random_state=42)  # 10% of data
```

- Useful for training models on a subset of data

Aggregation – Summarize Data

☐ 2. df.groupby()

- **Use**: Group rows by one or more columns and summarize
- **Example**:

```
df.groupby('department')['salary'].mean()
```

✅ Useful to reduce details into group-level summaries

Resampling (for Time Series)

☐ 3. df.resample()- Aggregate over time intervals (e.g., monthly sales)

```
df.set_index('date').resample('M').sum()
```

Data Transformation

Data transformation is the process of converting data into a format that is more suitable for:

- Modeling and analysis
- Improved accuracy
- Faster processing

Need for Data Transformation

Reason	**Explanation**
📊 Scale Alignment	Features with different units (e.g., age vs income) can skew models
Algorithm Requirements	Some models assume data is normally distributed or scaled (e.g., KNN, SVM)
✸ Outlier Reduction	Reduces the influence of extreme values
Clean Input Format	Transforms categories, dates, etc., into usable numeric formats
🔄 Uniformity	Makes data from different sources consistent

Transformation Techniques

1. 📏 Normalization (Min-Max Scaling)

- Scales all values to a **range [0, 1]**
- **Formula:**

$$x_{\text{norm}} = \frac{x - \min(x)}{\max(x) - \min(x)}$$

```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
df[['age', 'salary']] = scaler.fit_transform(df[['age', 'salary']])
```

Use when:

- Features have **different units or scales**
- Algorithm is **distance-based** (KNN, K-means)
- when your algorithm cares about **scale**

2. 📊 Standardization (Z-score Scaling)

- Converts values into standard normal distribution: **mean = 0, std = 1**
- **Formula:**

$$z = \frac{x - \mu}{\sigma}$$

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
df[['age', 'income']] = scaler.fit_transform(df[['age', 'income']])
```

Use when:

- You need **Gaussian distribution**
- Algorithm is **sensitive to scale**
- when your algorithm cares about **distribution**

Binary Coding

Binary coding is a method of converting categorical variables (especially those with multiple categories) into binary (0/1) form for use in machine learning models.

It's part of data transformation to make categorical data numerically usable.

Label Encoding : Simple binary encoding — works when only two categories are present.

Multiple Categories → One-Hot encoding

df = pd.DataFrame({'color': ['Red', 'Green', 'Blue', 'Red']})

df_encoded = pd.get_dummies(df, columns=['color'])

print(df_encoded)

Ranking Transformation is a data transformation technique where **numerical or categorical data** is replaced by its **rank** in the dataset, based on sorting.

Different Types of Ranks

Method	Description
Average	Average rank for tied values (default in pandas)
Min	All tied values get the lowest rank (dense ranking)
Max	All tied values get the highest rank
First	Assign ranks in order they appear
Dense	No skipped ranks for ties

```
import pandas as pd

df = pd.DataFrame({
    'Student': ['A', 'B', 'C', 'D', 'E'],
    'Score': [95, 88, 76, 65, 72]
})

# Apply ranking
df['Rank'] = df['Score'].rank(method='min')  # or 'average', 'dense', 'first', 'm
print(df.sort_values('Rank'))
```