Data Wrangling II

- Wrangling = data surgery → no real strict definition, but to me is getting the data into an analyzable state
  - Cleaning/EDA
  - Joining/Filtering
  - Transforming/Normalizing/Scaling if necessary
- Today we are going to cover:
  - restructuring, merging, encoding, time series, text, dimensionality, dimensionality reduction

Reshaping: Wide vs. Long Format

- Wide = multiple variables in columns
- Long/tidy = one observation per row
- Data typically come in wide because it's more human readable and easier to make comparisons

Wide vs Long Format

Wide vs Long Format Example

Wide Format

Student	Math	English	Science
A	90	85	95
B	80	88	78

Long Format

Student	Subject	Score
A	Math	90
A	English	85
A	Science	95
B	Math	80
B	English	88
B	Science	78

Utility of Long Format

- Long format makes analysis and plotting easier
- Most packages these days allow grouping by columns and so a single function call can handle multiple categories in a column – avoids having to do multiple calls or create loops

Utility of Wide Format

- Human readability & reports
  - Easy to scan in Excel or as a summary table.
  - Example: monthly sales across columns → managers can glance at trends.
- Matrix-style data for algorithms
  - Some methods (e.g., classical ML libraries, PCA) expect a feature matrix: each row = observation, each column = feature.
  - This is a kind of “wide” view where columns are the input variables.

Melting: Wide→Long

- "Melt" variables into a single column.
- Use when variables are stored as columns but really represent categories.

```
import pandas as pd

# Example wide-format dataframe
df = pd.DataFrame({
    "Student": ["A", "B"],
    "Math": [90, 80],
    "English": [85, 88],
    "Science": [95, 78]
})

print("Wide format:")
print(df)

# Melt into long format
df_long = df.melt(id_vars="Student",
                  var_name="Subject",
                  value_name="Score")

print("\nLong format:")
print(df_long)
```

	Student	Math	English	Science
0	A	90	85	95
1	B	80	88	78

	Student	Subject	Score
0	A	Math	90
1	B	Math	80
2	A	English	85
3	B	English	88
4	A	Science	95
5	B	Science	78

Code for Plotting

```python
import pandas as pd
import matplotlib.pyplot as plt

# Wide data
df_wide = pd.DataFrame({
    "Student": ["A", "B"],
    "Math": [90, 80],
    "English": [85, 88],
    "Science": [95, 78]
})

# Plotting requires separate calls for each subject
plt.figure(figsize=(6,4))
plt.bar(["A","B"], df_wide["Math"], label="Math")
plt.bar(["A","B"], df_wide["English"], bottom=df_wide["Math"], label="English")
# Hard to add Science here — need more manual handling
plt.title("Student Scores (Wide Format Plotting)")
plt.legend()
plt.show()
```

```python
import seaborn as sns

# Melt to long format
df_long = df_wide.melt(id_vars="Student",
                       var_name="Subject",
                       value_name="Score")

# One call does it all
plt.figure(figsize=(6,4))
sns.barplot(x="Subject", y="Score", hue="Student", data=df_long)
plt.title("Student Scores (Long Format Plotting)")
plt.show()
```

Pivoting: Long → Wide

```python
import pandas as pd

# Long data
df_long = pd.DataFrame({
    "Student": ["A","A","A","B","B","B"],
    "Subject": ["Math","English","Science","Math","English","Science"],
    "Score": [90, 85, 95, 80, 88, 78]
})

print(df_long)
```

```
  Student  Subject  Score
0       A     Math     90
1       A  English     85
2       A  Science     95
3       B     Math     80
4       B  English     88
5       B  Science     78
```

```python
# Pivot: Subjects become columns
df_wide = df_long.pivot(index="Student",
                        columns="Subject",
                        values="Score")

print(df_wide)
```

```
Subject  English  Math  Science
Student
A             85    90       95
B             88    80       78
```

Pitfalls of Reshaping

- Forgetting the unit of analysis (row meaning changes).
- Duplicates or missing values can cause expansion/shrinkage.
- Always check counts before and after.

Merging and Joining

- Why merge/join?
  - Data is rarely all in one file.
  - Example: student roster + grades + attendance.
  - Need structured merging to combine meaningfully.

📘 Table 1: Student Roster

StudentID	Name	GradeLevel
101	Alice	10
102	Bob	11
103	Carol	10
104	David	12

📘 Table 2: Exam Scores

StudentID	Exam	Score
101	Math	90
102	Math	80
105	Math	85

Joining Basics - Keys

- Joins use keys (IDs).
- Primary key
  - A **unique identifier** for rows in a table
  - Each value appears **only once** in that table.
- Foreign key
  - A column in one table that **refers to a primary key in another table**.
  - Values can repeat (many students can take the same exam)

StudentID (PK)	Name	GradeLevel
101	Alice	10
102	Bob	11
103	Carol	10

ExamID	StudentID (FK)	Exam	Score
1	101	Math	90
2	102	Math	80
3	101	English	85

Joining Basics – Types of Joins

- Inner Join
  - Returns only rows with keys present in both tables.
- Left Join
  - Returns all rows from the left table, with matching rows from the right.
  - Missing matches in the right table become NULL.
- Right Join
  - Returns all rows from the right table, with matching rows from the left.
  - Missing matches in the left table become NULL.
- Full Outer Join
  - Returns all rows from both tables.
  - Where no match exists, fills with NULL.
- Cross Join (Cartesian Product)
  - Every row in the left table combines with every row in the right.
  - Rarely used, but useful for generating combinations
- **If keys don't align → missing values or extra rows.**

📘 Table 1: Student Roster

StudentID	Name	GradeLevel
101	Alice	10
102	Bob	11
103	Carol	10
104	David	12

📘 Table 2: Exam Scores

StudentID	Exam	Score
101	Math	90
102	Math	80
105	Math	85

Inner Join

Table 1: Student Roster

StudentID	Name	GradeLevel
101	Alice	10
102	Bob	11
103	Carol	10
104	David	12

Table 2: Exam Scores

StudentID	Exam	Score
101	Math	90
102	Math	80
105	Math	85

Inner Join Result

StudentID	Name	GradeLevel	Exam	Score
101	Alice	10	Math	90
102	Bob	11	Math	80

- Inner Join
  - Returns only rows with keys present in both tables.

```python
# Inner join on StudentID
inner = pd.merge(roster, scores, on="StudentID", how="inner")
```

```sql
SELECT
    r.StudentID,
    r.Name,
    r.GradeLevel,
    s.Exam,
    s.Score
FROM StudentRoster r
INNER JOIN ExamScores s
    ON r.StudentID = s.StudentID;
```

Left Join

Table 1: Student Roster

StudentID	Name	GradeLevel
101	Alice	10
102	Bob	11
103	Carol	10
104	David	12

Table 2: Exam Scores

StudentID	Exam	Score
101	Math	90
102	Math	80
105	Math	85

Left Join Result

StudentID	Name	GradeLevel	Exam	Score
101	Alice	10	Math	90.0
102	Bob	11	Math	80.0
103	Carol	10	nan	nan
104	David	12	nan	nan

- Left Join
  - Returns all rows from the left table, with matching rows from the right.
  - Missing matches in the right table become NULL.

```python
# Left join on StudentID
left = pd.merge(roster, scores, on="StudentID", how="left")
```

```sql
SELECT
    r.StudentID,
    r.Name,
    r.GradeLevel,
    s.Exam,
    s.Score
FROM StudentRoster r
LEFT JOIN ExamScores s
    ON r.StudentID = s.StudentID;
```

Right Join

Table 1: Student Roster

StudentID	Name	GradeLevel
101	Alice	10
102	Bob	11
103	Carol	10
104	David	12

Table 2: Exam Scores

StudentID	Exam	Score
101	Math	90
102	Math	80
105	Math	85

Right Join Result

StudentID	Name	GradeLevel	Exam	Score
101	Alice	10.0	Math	90
102	Bob	11.0	Math	80
105	nan	nan	Math	85

- Right Join
  - Returns all rows from the right table, with matching rows from the left.
  - Missing matches in the left table become NULL.

```python
# Right join on StudentID
right = pd.merge(roster, scores, on="StudentID", how="right")
```

```sql
SELECT
    r.StudentID,
    r.Name,
    r.GradeLevel,
    s.Exam,
    s.Score
FROM StudentRoster r
RIGHT JOIN ExamScores s
    ON r.StudentID = s.StudentID;
```

Full Outer Join

Table 1: Student Roster

StudentID	Name	GradeLevel
101	Alice	10
102	Bob	11
103	Carol	10
104	David	12

Table 2: Exam Scores

StudentID	Exam	Score
101	Math	90
102	Math	80
105	Math	85

Full Outer Join Result

StudentID	Name	GradeLevel	Exam	Score
101	Alice	10.0	Math	90.0
102	Bob	11.0	Math	80.0
103	Carol	10.0	nan	nan
104	David	12.0	nan	nan
105	nan	nan	Math	85.0

- Full Outer Join
  - Returns all rows from both tables.
  - Where no match exists, fills with NULL.

```python
# Full outer join on StudentID
outer = pd.merge(roster, scores, on="StudentID", how="outer")
```

```sql
SELECT
    r.StudentID,
    r.Name,
    r.GradeLevel,
    s.Exam,
    s.Score
FROM StudentRoster r
FULL OUTER JOIN ExamScores s
    ON r.StudentID = s.StudentID;
```

Cross Join (Cartesian)

Table 1: Student Roster

StudentID	Name	GradeLevel
101	Alice	10
102	Bob	11
103	Carol	10
104	David	12

Table 2: Exam Scores

StudentID	Exam	Score
101	Math	90
102	Math	80
105	Math	85

Cross Join Result (First 10 Rows)

StudentID_x	Name	GradeLevel	StudentID_y	Exam	Score
101	Alice	10	101	Math	90
101	Alice	10	102	Math	80
101	Alice	10	105	Math	85
102	Bob	11	101	Math	90
102	Bob	11	102	Math	80
102	Bob	11	105	Math	85
103	Carol	10	101	Math	90
103	Carol	10	102	Math	80
103	Carol	10	105	Math	85
104	David	12	101	Math	90

- Cross Join (Cartesian Product)
  - Every row in the left table combines with every row in the right.
  - Rarely used, but useful for generating combinations

```python
import pandas as pd

# Example data
roster = pd.DataFrame({
    "StudentID": [101, 102],
    "Name": ["Alice", "Bob"]
})

exams = pd.DataFrame({
    "Exam": ["Math", "English"]
})

# Add dummy key to both
roster["key"] = 1
exams["key"] = 1

# Cross join by merging on dummy key
cross = pd.merge(roster, exams, on="key").drop("key", axis=1)

print(cross)
```

```sql
SELECT
    r.StudentID,
    r.Name,
    e.Exam
FROM StudentRoster r
CROSS JOIN Exams e;
```

Notes on Cross Join

- Generate all possible combinations
  - Example:
    - Table A = list of students
    - Table B = list of exam dates
    - Cross join → full exam schedule (each student × each exam date).
- Parameter sweeps / grid searches
  - Example: you have hyperparameters alpha=[0.1,0.2] and beta=[1,2,3].
  - Cross join them to test all 2×3 = 6 combinations.
- Simulation setups
  - Example: cross join weather conditions × soil types × crop types → all possible scenarios for testing.
- Filling in missing observations
  - Example: create a skeleton of all (store, day) pairs via cross join, then left join sales → reveals which store/day combos are missing sales.

Pitfalls Joins/Merges

- Duplicate Keys → Row Explosion
  - If both tables have repeated keys, join multiplies rows.
  - Example: Student 101 has 2 grade records × 2 roster entries = 4 rows.
- Mismatched Key Formats"00123" vs 123 → no match.
  - Different data types (string vs integer) silently fail.
- Missing Keys → Unexpected NULLs
  - Rows without matches show up with missing values.
  - Can distort counts and averages if not handled.
- Unintended Cross Join
  - Forgetting join keys results in Cartesian product.
  - Data size explodes: 1,000 × 1,000 = 1,000,000 rows.
- Column Name Collisions
  - Columns with the same name (not join keys) may get suffixes (_x, _y).
  - Can cause confusion in downstream analysis.

Encoding Categorical Data

- Why Encoding?
  - Many datasets contain categorical variables (e.g., gender, color, shirt size).
  - Computers and models can't work directly with text labels.
  - We need to convert categories into numbers while preserving meaning.
- Key Point
  - Encoding = turning categories into a format that models can use.
  - The trick is:
    - Don't accidentally add false order where none exists.
    - Don't lose important structure (e.g., ordinal rankings).

```
"red" → 0, "blue" → 1, "green" → 2
```

```
"Small" → 1, "Medium" → 2, "Large" → 3
```

Why Teach Encoding?

- Packages do it for you:
  - pandas.get_dummies() for one-hot encoding.
  - scikit-learn’s OneHotEncoder, OrdinalEncoder, etc.
  - Even tree-based models (like XGBoost, CatBoost, LightGBM) can handle categorical features directly in some cases.
- But they don’t decide for you:
  - You must choose which encoding is appropriate.
  - Example: Small/Medium/Large → Ordinal (ordering matters)
  - Example: Red/Blue/Green → One-hot (no order).
- And they don’t prevent mistakes:
  - If you label encode Red=0, Blue=1, Green=2 and feed it into a linear regression, the model thinks Green > Blue > Red.
  - Packages won’t stop you — you need to know why that’s wrong.

Label Encoding

- Assigns **arbitrary** integers to categories.
  - Example: red=0, blue=1, green=2.
  - Fast and simple.
  - Dangerous if the model interprets the numbers as ordered.
  - Use only when categories are nominal and the model doesn’t assume order (tree-based models are okay).

Ordinal Encoding

- Numbers are assigned to respect a real, meaningful order.
  - Example: shirt sizes Small=1, Medium=2, Large=3.
  - Encodes the rank information.
  - Doesn’t capture distances (Medium isn’t necessarily “twice” Small).
  - Use when categories are truly ordinal.

Label vs Ordinal Summary

- Label = arbitrary codes (risk of fake order).
- Ordinal = deliberate order (when order matters).

One Hot Encoding

- What It Is
  - Creates a new binary column for each category.
  - Each row has exactly one “1” (hot) and the rest “0”.
- Why Use It
  - Prevents fake ordering.
  - Works for nominal categories (unordered).Most common and safest encoding method.

One Hot Example

Student	Color	→	Red	Blue	Green
A	Red		1	0	0
B	Blue		0	1	0
C	Green		0	0	1

Pitfall:

•High dimensionality when there are many categories (e.g., thousands of ZIP codes).

Dummy Encoding

- Sometimes used interchangeably with One Hot Encoding but slightly different
- What It Is
  - Same as one-hot encoding, but drops one category.
  - The dropped column is the reference (baseline).
- Why Use It
  - Prevents multicollinearity in linear models.
  - Keeps the same information with fewer columns.

Dummy Encoding Example

Student	Color	Blue	Green
A	Red	0	0
B	Blue	1	0
C	Green	0	1

When both dummy columns = 0 → means the dropped category (Red).

One Hot vs Dummy Encoding

One-Hot Encoding (All Columns)
→ Multicollinearity Risk

Student	Color	Red	Blue	Green
A	Red	1	0	0
B	Blue	0	1	0
C	Green	0	0	1

Dummy Encoding (Drop Red)
→ No Redundancy

Student	Color	Blue	Green
A	Red	0	0
B	Blue	1	0
C	Green	0	1

- One-Hot Encoding (Red, Blue, Green all included) → multicollinearity risk because one column can be perfectly predicted from the others.
  - If you know Red and Blue, you can always infer Green.
  - That's perfect multicollinearity.
  - In linear models (regression, logistic regression), this makes the design matrix singular (can't invert), so the model parameters can't be estimated uniquely.
- Dummy Encoding (drop Red as baseline) → removes redundancy, no multicollinearity.

There are many other types

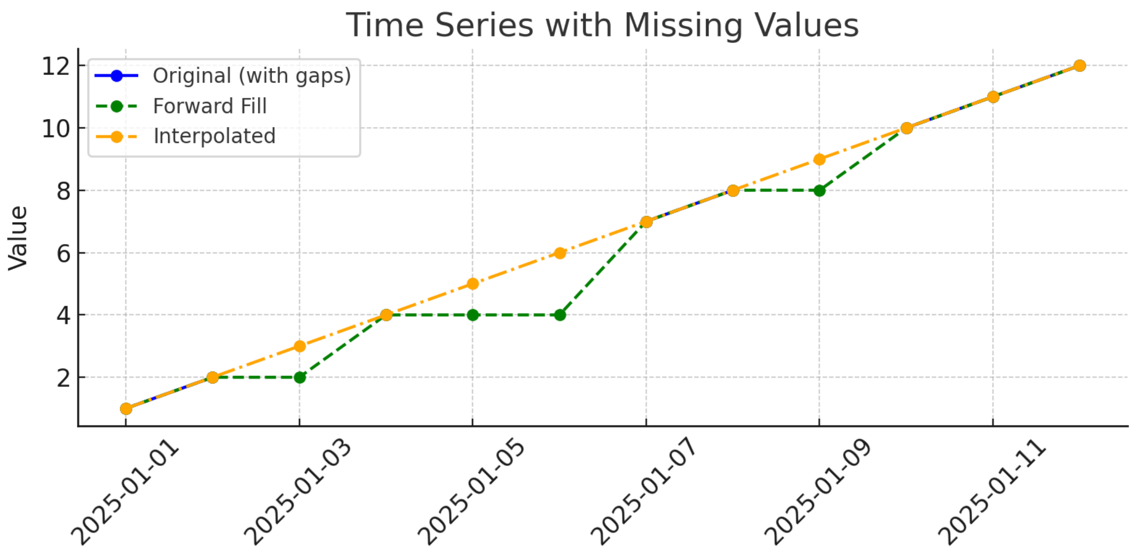
- Binary Encoding
  - Convert category ID → integer → binary digits.
  - Fewer columns than one-hot.
  - Example: 5 categories → 001, 010, 011, 100, 101.
- Hash Encoding
  - Hash categories into fixed number of buckets.
  - Handles huge categorical spaces.
  - Risk: collisions.
- Target (Mean) Encoding
  - Replace category with average of target variable.
  - Example: City → average house price.
  - Risk: data leakage if not cross-validated.
- Frequency / Count Encoding
  - Replace category with how often it appears.
  - Captures rarity/popularity.
- Embedding Encoding
  - Neural networks learn dense vectors for categories.
  - Used in recommendation systems & NLP.

Time Series Wrangling

- Why are time series special?
  - Observations are ordered in time
  - Wrangling errors can break sequence

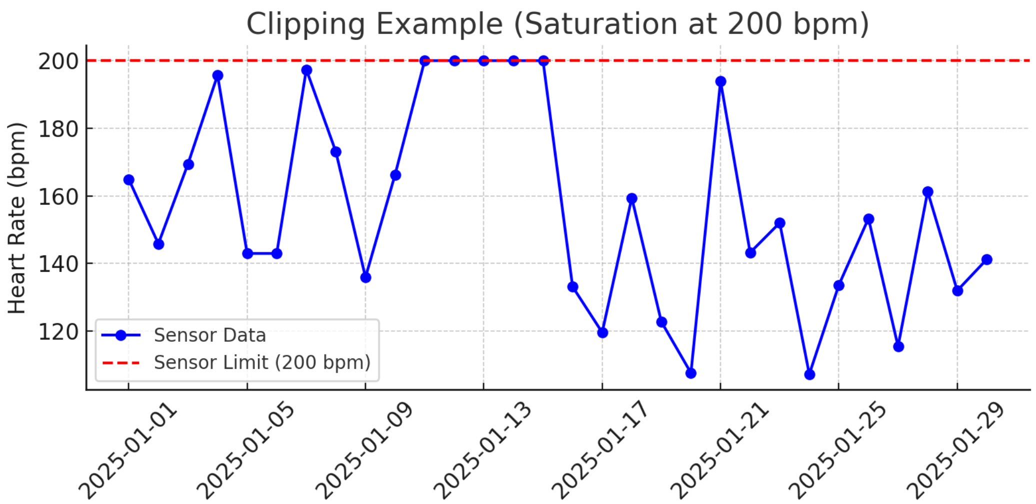
Time Based Missingness

- Irregular timestamps, dropped sensor packets
- Fixes: forward fill, interpolation, or dropping
  - The type of fix will depend on the analysis you are doing



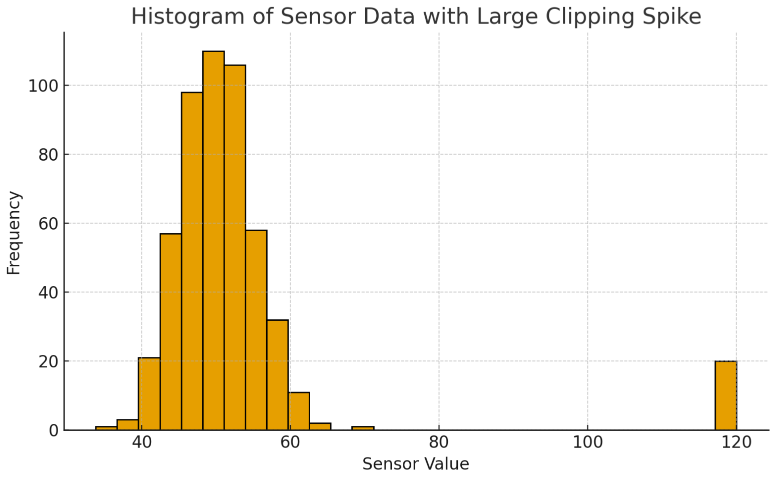
Clipping and Saturation

- Sensors capped at max/min (usually max)
- Fixes:
  - Flag clipped values
  - Replace with NA



How to Spot Clipping vs. Real Values

- Spotting:
  - Generally by visualization (time series or histograms)
- At the sensor limit
  - If values hit the device's documented maximum or minimum (e.g., HR monitor max = 200 bpm, accelerometer range = ±16g), and sit there, it's probably clipping.
- Flatlined at boundary
  - True data is rarely exactly flat at an instrument's extreme.
  - Long sequences of identical max values → suspicious.
- Distribution check
  - Plot histogram of values
  - Sharp spike at the sensor limit suggests clipping

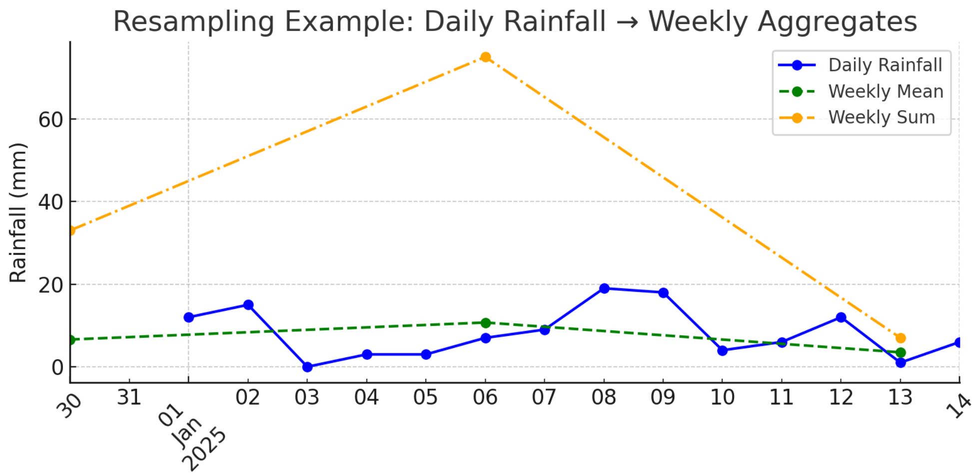


Resampling

- **What It Is**
  - Downsampling → coarser (daily → weekly).
  - Upsampling → finer (monthly → daily, often with interpolation).
- **Why Do It**
  - Align datasets with different sampling rates.
  - Smooth out noisy data.
  - Match scale to research question (daily sales vs. monthly sales).
- **Pitfalls**
  - Downsampling → loses detail.
  - Upsampling → invents values (risk of artifacts).
  - Aggregation choice (mean, sum, max) changes conclusions.

Resampling Example

- Plot showing 14 days of daily rainfall totals.
- Two resampled versions overlaid:
  - Weekly Mean (green dashed) → suggests light, steady rainfall.
  - Weekly Sum (orange dash-dot) → shows big storm weeks.
- Same data → very different story depending on aggregation.



Preprocessing Text

- Free-form, unstructured (not neat rows & columns).
- Computers don't understand words directly.
- Need to convert text → numbers

- Goal
  - Clean and normalize raw text before analysis.
  - Reduce noise, standardize representation.
- Common Steps:
  - Text cleaning
    - Lowercasing → "Data" → "data“
    - Remove punctuation/symbols → "fun!!!" → "fun“
    - Remove stopwords → "the, is, and“
    - Stemming → "running" → "run“
    - Lemmatization → "better" → "good“
  - Tokenization
    - Sentence level
    - Word level
    - Subword level

Tokenization – the core of NLP

- Varies per language and project
- What It Is
  - Breaking text into smaller units (tokens).
  - Tokens are the building blocks for analysis.
  - Not always just words.
- Types of Tokens
  - Words → "Data wrangling is fun" → [Data, wrangling, is, fun].
  - Subwords → "wrangling" → [wrangl, ing].
  - Characters → [D, a, t, a].
  - Sentences → "Data wrangling is fun.", "Text processing is important.".
- Why It Matters
  - Defines what “counts” as a unit of meaning.
  - Impacts vocabulary size, interpretability, and downstream models.

- Pre-neural era (before ~2013)
  - Training was **mostly supervised or rule-based**.
  - Labeled “spam vs not spam”
- Word embeddings era (2013–2016)
  - Training objective: make word vectors that capture meaning through co-occurrence.
    - “King – Man + Woman ≈ Queen”.
- Subword models & Transformers (2017 onward)
  - Training is still self-supervised, just with different objectives:
    - Masked language modeling (BERT): Hide some tokens and train the model to guess them.
    - Causal language modeling (GPT): Predict the next token in a sequence.
  - Tokens here are subwords (Byte Pair Encoding, WordPiece, SentencePiece).

TF-IDF

- TF-IDF = Term Frequency × Inverse Document Frequency
  - Term Frequency (TF)
  - How often a word appears in a single document.
    - Example: in sentence “Data wrangling is fun, data is powerful”
      - data appears 2 times, wrangling 1, fun 1.
- Inverse Document Frequency (IDF)
  - Downweights words that appear in many documents.
  - Idea: if a word is everywhere (like the, is), it doesn’t help distinguish documents.
  - Rare words (like wrangling) get higher weight.
- TF × IDF = TF-IDF
  - Word importance = how often it appears in this document × how rare it is across the collection.
  - So:
    - data might have a medium score (common, but relevant).
    - wrangling gets a high score (rare, distinctive).
    - the gets near zero (too common to be useful).

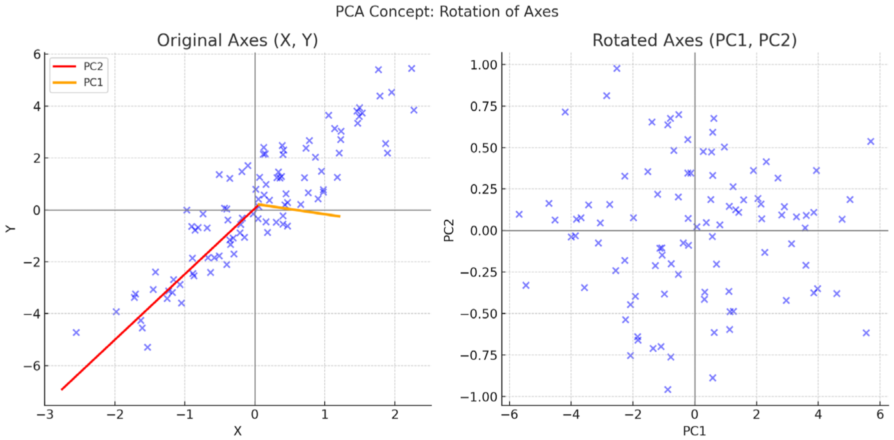
- Information retrieval (search engines): Used to rank documents by query relevance.
- Text classification: Helps highlight discriminative words.
- Clustering: Makes documents more comparable by their unique vocabulary.

Dimensionality Reduction

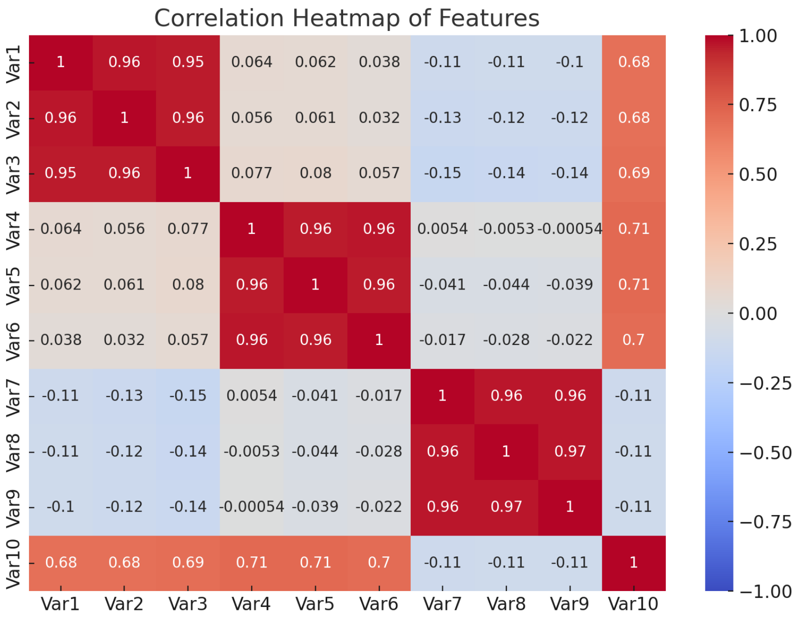
- Could be part of EDA (but that's also part of Data Wrangling)
- Why Reduce Dimensions?
  - Too many features = redundancy, inefficiency.
  - Helps with collinearity.
- There are many, many methods, the one we will talk about is Principal Components Analysis (PCA)

Principal Component Analysis (PCA)

- What Is PCA?
  - Principal Component Analysis (PCA) is a method to reduce dimensions.
  - Finds new “axes” (directions) that capture the most variance in the data.
  - Think: rotating the coordinate system to simplify data.
- Why Use It?
  - Too many features → redundancy & inefficiency.
  - PCA keeps the most informative parts, drops the rest.
  - Helps with collinearity and visualization.
- Key Idea
  - Replace original correlated variables with a smaller set of uncorrelated components.
  - Each principal component = weighted combo of original features.

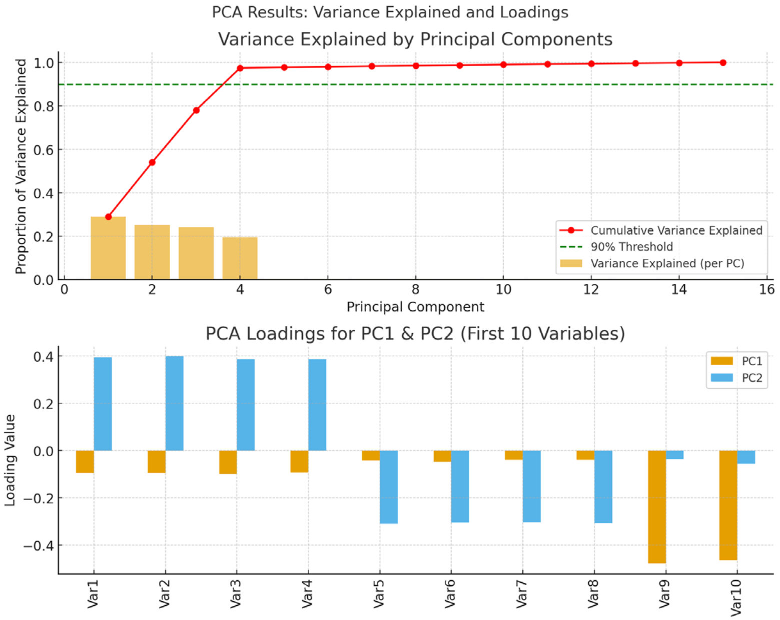


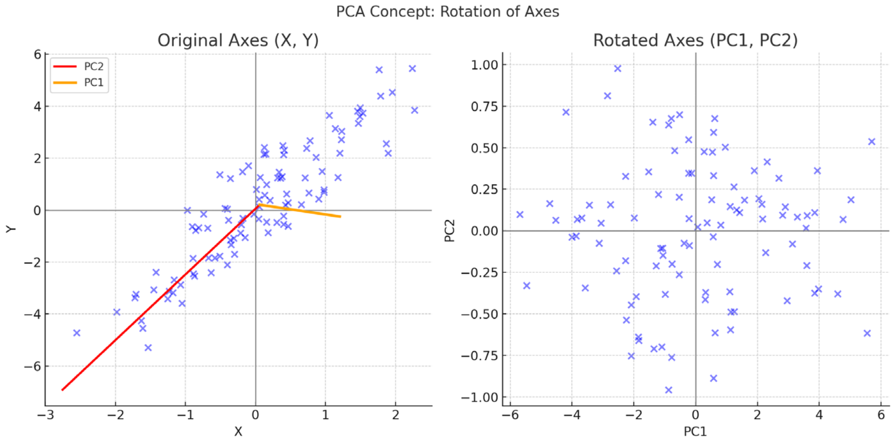
Toy Example



- A correlation heatmap of 10 synthetic variables.
- Clear clusters of strong correlations (Var1–Var3, Var4–Var6, Var7–Var9), plus a mixed variable (Var10).
- We have redundancy. PCA can simplify this

PCA Interpretation





Reduce the feature space from 15 to 4 variables and capture almost 100% of the variance