Session 2 Manual

Topic: *Binning, Outlier Treatment, & Fuzzy Matching*

Focus: Learn how to detect and treat outliers, group continuous data into bins, and handle messy text data using fuzzy logic.

1. Outlier Detection

Outliers are data points that differ significantly from the rest of the dataset. They can mislead averages, skew models, and cause wrong insights.

For example, in a salary dataset, most employees earn between 30,000–60,000. If one wrongly recorded value is 3,000,000, the average salary will be way higher than reality.

Methods for Outlier Detection

1. **Z-Score Method –** Measures how many standard deviations a value is away from the mean.

Formula: $Z = (x - \mu) / \sigma$ where:

- o x = data point
- o μ = mean
- o σ = standard deviation

Rule of Thumb: If $|Z| > 3 \rightarrow$ possible outlier

Example:

```
import numpy as np
import pandas as pd

df = pd.DataFrame({"Salary": [30000, 35000, 40000, 45000, 1000000]})
mean = df["Salary"].mean()
std = df["Salary"].std()
df["Zscore"] = (df["Salary"] - mean) / std
print(df)
```

Note: *Use when data is normally distributed.*

2. **IQR (Interquartile Range) Method –** Measures spread of the middle 50% of data.

Steps:

1. Calculate Q1 (25th percentile) and Q3 (75th percentile).
2. Compute IQR = Q3 – Q1.
3. Define bounds:
   - Lower bound = Q1 – 1.5 × IQR
   - Upper bound = Q3 + 1.5 × IQR
4. Anything outside bounds is an outlier.

Example:

```
Q1 = df["Salary"].quantile(0.25)
Q3 = df["Salary"].quantile(0.75)
IQR = Q3 - Q1
lower = Q1 - 1.5 * IQR
upper = Q3 + 1.5 * IQR
outliers = df[(df["Salary"] < lower) | (df["Salary"] > upper)]
print(outliers)
```

Note: *Good for skewed/non-normal distributions.*

Outlier Treatment

1. **Remove** them by dropping the rows.

```
df_clean = df[(df["Salary"] >= lower) & (df["Salary"] <= upper)]
```

2. **Cap** them by setting extreme values to boundary.

```
df["Salary"] = np.where(df["Salary"] > upper, upper, df["Salary"])
```

3. **Transform** by applying log/square root to reduce effect.

```
df["Salary"] = np.log(df["Salary"])
```

2. Data Binning

Binning means converting continuous values into categories/groups. It helps simplify analysis and spot patterns.

Binning makes it easier to interpret (e.g., “Age Groups” instead of raw ages), is very useful for visualization, and helps when models need categorical features.

b) Methods

1. **Fixed-width Bins (`cut`)** – Divide values into equal ranges.

```
ages = [5, 12, 18, 24, 37, 45, 52, 67, 80]
df = pd.DataFrame({"Age": ages})
df["AgeGroup"] = pd.cut(df["Age"], bins=[0, 18, 35, 60, 100],
labels=["Child", "Young Adult", "Adult", "Senior"])
print(df)
```

2. **Quantile Bins (`qcut`)** – Divide data so each bin has approximately equal number of observations.

```
df["SalaryBin"] = pd.qcut(df["Salary"], q=4, labels=["Low", "Medium", "High",
"Very High"])
print(df)
```

Types of Binning

1. **Equal-width binning** – Divides range into equal intervals.
2. **Equal-frequency binning** – Each bin has roughly the same number of values.
3. **Custom bins** – You define ranges manually.

Note: *Use `cut` for fixed intervals, `qcut` for distribution-based groups.*

3. Introduction to Fuzzy Logic & Fuzzy Matching

Real-world data is messy; it contains spelling mistakes, different formats, and near-matches. Fuzzy matching helps identify records that are “close enough.”

What is Fuzzy Logic?

Traditional logic consists of just either 0 or 1 (true/false). Fuzzy logic, on the other hand, allows degrees of truth (0.0 → 1.0).

For example, instead of saying “Age = Young (True/False),” fuzzy logic says “Age = 0.8 Young, 0.2 Middle-aged.”

What is Fuzzy Matching?

Matching strings even if they are not exactly the same.

For example:
- “Jon” vs “John” → Similar.
- “Colour” vs “Color” → Similar.

Using FuzzyWuzzy in Python

Install:

```
pip install fuzzywuzzy
pip install python-Levenshtein
```

Import: `from fuzzywuzzy import fuzz, process`

Examples:

```
fuzz.ratio("apple", "aple")                          # 80
fuzz.partial_ratio("new york", "york")               # 100
fuzz.token_sort_ratio("hello world", "world hello")  # 100
```

Practical Example

Suppose you have customer names:

```
names = ["Jon", "John", "Jonathan", "Jonny", "Johan"]
query = "John"
matches = process.extract(query, names, limit=3)
print(matches)
```

Output might be: `[('John', 100), ('Jon', 80), ('Jonathan', 73)]`

Note: *Helps when merging datasets with inconsistent spellings.*

4. Practical Mini-Workflow

1. Detect outliers → Z-score / IQR
2. Decide treatment → Remove, Cap, or Transform
3. Apply binning → Use `cut` or `qcut` to group continuous variables
4. Handle messy text → Use FuzzyWuzzy for name/address/email cleaning