COMPLETE DATA PREPROCESSING WORKBOOK

From Raw Numbers to Intelligent Models

Name: ____________________ Branch/Year: ____________________ Date: ____________________

> ***The Golden Rule:*** *"A simple model with clean data is better than an advanced model with messy & unclean data."*

WHY PREPROCESSING MATTERS?

Machine Learning is based on data. Think of a model like a car engine. Data is the fuel. If you put dirty, contaminated fuel into a Ferrari, the engine will break. Similarly, if you feed missing values or bad data into a powerful AI model, it will fail.

MODULE 1: LOADING THE DATA

Before we can clean anything, we must import our toolkit. `pandas` is used for data manipulation (tables), and `numpy` handles the math.

Task 01: Type the following code to import libraries and load the dataset.

```python
import numpy as np
import pandas as pd

# Load the dataset
df = pd.read_csv('loan_data.csv')

# Print the first few rows
df
```

**The output shows a table with columns like Name, Income, Education, etc.*

MODULE 2: HANDLING MISSING VALUES

PART 1: IDENTIFYING THE HOLES

We can't visually scan thousands of rows to find errors. We use code to find them.

Step 1: Get a Summary of Data

The `info()` function tells us how many non-null values exist in every column.

```
# Get summary of data types and non-null counts
df.info()
```

Step 2: Count Specific Missing Values

The `isnull().sum()` function gives us the exact count of missing cells per column.

```
# Count missing values in each column
df.isnull().sum()
```

OBSERVATION & REFLECTION

1. Look at the output. Which column has missing values?

   Answer: ____________

2. Which command is better for quickly identifying which column has missing data?

   - ☐ **df.info()**
   - ☐ **df.isnull().sum()**

3. Explain your choice. Why is that command better?

____

____

PART 2: STRATEGIES TO FIX MISSING VALUES

Now that we have identified the problem, we must fix it. We have 3 common strategies.

Strategy A: Dropping Rows (The "Nuclear" Option)

If we have thousands of records and only a few are empty, we can simply delete the rows that have missing info.

```
# Drop any row that contains missing values
df_dropped = df.dropna()

# View result
df_dropped
```

Strategy B: Filling with Mean (The "Average")

If the data is normally distributed (no crazy rich people like Elon Musk), we fill the blank with the average.

```
# Calculate the Mean (Average)
mean_val = df['Income'].mean()

# Fill NaN values with the Mean
df['Income'] = df['Income'].fillna(mean_val)

# View result
df
```

Strategy C: Filling with Median (The "Outlier-Proof" Option)

Because our dataset includes **Elon Musk** (Income = 10,000,000), the Mean is skewed very high! In this case, the Median (the middle value) is a safer choice.

```
# Load the dataset again,
# since filling with mean was a bad choice
df = pd.read_csv('loan_data.csv')
```

```python
# Calculate the Median (Middle Value)
median_val = df['Income'].median()

# Fill NaN values with the Median
df['Income'] = df['Income'].fillna(median_val)

# View result
df
```

SUMMARY: WHEN TO USE WHICH STRATEGY?

1. Drop Rows: Use when you have a massive dataset and very few missing values.

2. Fill with Mean: Use when data is normally distributed (symmetric) with **no outliers**.

3. Fill with Median: Use when data is skewed or contains **outliers** (like Elon Musk).

MODULE 3: REMOVING OUTLIERS

1. What is an Outlier?

> **Definition:** An outlier is a data point that differs significantly from other observations. It is an anomaly—like a student scoring 1000% on a test, or someone earning $10 Million in a group of students.

Question: In our dataset, we have incomes like 3200, 4100, 5200... and 10,000,000 (Elon Musk). Why should we remove Elon Musk before training our model?

2. The IQR Method

How do we decide mathematically what counts as "too big" or "too small"? The mathematician **John Tukey** created the **Interquartile Range (IQR)** method.

3. Exercise: Calculate by Hand

Let's play computer. Look at the **Income** column in your dataset.

Step A: Arrange all 8 income values in Ascending Order (Smallest to Largest).

Step B: Count the total number of values (N).

N = ________

Step C: Find the 25th Percentile (Q1).

Hint: 25% of 8 is 2. So, take the 2nd value in your sorted list.

Q1 = ________

Step D: Find the 75th Percentile (Q3).

Hint: 75% of 8 is 6. So, take the 6th value in your sorted list.

Q3 = ________

Step E: Calculate IQR (The spread of the middle data).

Formula: `IQR = Q3 - Q1`

IQR = ________ - ________ = ________

Step F: Calculate the Allowed Range (The Fences).

Anything below **Lower Range** or above **Upper Range** is an outlier.

`Lower Range = Q1 - (1.5 * IQR)`

Lower = ________ - (1.5 * ________) = ________

`Upper Range = Q3 + (1.5 * IQR)`

Upper = ________ + (1.5 * ________) = ________

Conclusion:

Is Elon Musk (10,000,000) greater than your Upper Range?

☐ YES (Remove him) ☐ NO (Keep him)

4. Code Implementation

Task: Type the code to remove the outliers.

```
# 1. Calculate Q1 (25%) and Q3 (75%)
Q1 = df['Income'].quantile(0.25)
Q3 = df['Income'].quantile(0.75)

# 2. Calculate IQR
IQR = Q3 - Q1

# 3. Define the Upper and Lower Limits
lower_limit = Q1 - 1.5 * IQR
upper_limit = Q3 + 1.5 * IQR

# 4. Filter the data (Keep only valid rows)
df_clean = df[ (df['Income'] >= lower_limit) &
               (df['Income'] <= upper_limit) ]

# View the clean data
df_clean
```

MODULE 4: ENCODING CATEGORICAL DATA

1. The Language Barrier

> **The Problem:** Machine Learning models are mathematical equations. They can multiply, divide, and subtract numbers.
>
> They **cannot** understand text. You cannot calculate `"Male" * 5` or `"PhD" / 2`.

The Solution: Encoding. We must translate text (Categorical Data) into Numbers (Numerical Data).

2. Exercise: Education (Ranked Data)

Convert Education to Numbers

Task: Assign a number to each degree to represent its rank.

```
B.Tech (Lowest)  →  1
M.Tech (Medium)  →  2
PhD (Highest)    →  3
```

> ***Success!** You have converted text to numbers. Let's see how to do this automatically.*

3. Code: Label Encoding

We use the `LabelEncoder` library. It automatically assigns numbers (0, 1, 2...) to ranks.

Task: Type the code to encode Education.

```
from sklearn.preprocessing import LabelEncoder

# Initialize the Encoder
le = LabelEncoder()

# Apply it to the Education column
df['Education'] = le.fit_transform(df['Education'])

# View the data
df.head()
```

4. Exercise: The Gender Trap

Part A: Assigning Numbers

Now let's try the same strategy for the **Gender** column.

```
Male → 0
Female → 1
```

Part B: The Problem

Critical Thinking Question: We assigned numbers 0 and 1. In math, 1 > 0. Does this mathematical ranking make sense for Gender? Why or why not?

5. The Solution: One-Hot Encoding

> **One-Hot Encoding:** Instead of ranking them (0 vs 1), we create a separate column for each category.
>
> - Is the person Male? (Yes/No)
> - Is the person Female? (Yes/No)

6. Code: One-Hot Encoding

We use the `get_dummies()` function from pandas to create these new columns automatically.

Task: Type the code to convert Gender using One-Hot Encoding.

```python
# Apply One-Hot Encoding to Gender
# columns=['Gender'] tells it which column to split
df = pd.get_dummies(df, columns=['Gender'])

# View the new columns (Gender_Male, Gender_Female)
df.head()
```

MODULE 5: FEATURE SCALING

1. The Size Problem

The Bias Problem: Machine Learning models become **biased towards bigger numbers**.

The computer does not understand the importance of features (e.g., that 'Job Stability' is just as important as 'Income'). It only understands numbers.

- **Income:** 5000
- **Job Stability:** 2.5

Because 5000 is much bigger than 2.5, the model thinks Income is **2000 times more important** than Job Stability.

The Solution: Min-Max Scaling. We shrink all columns so they fit between 0 and 1.

2. Exercise: Normalization (Min-Max)

We use a technique called **Min-Max Scaling** to squash data between 0 and 1.

```
New Value = (Value - Min) / (Max - Min)
```

Scenario: Imagine in our dataset:
The **Minimum Income** is 3000.
The **Maximum Income** is 7000.

Task A: Scale a Middle Class Income (5000).

New Value = (**5000** - **3000**) / (**7000** - **3000**)

New Value = **2000** / **4000**

Answer = ________

Task B: Scale the Poorest Income (3000).

New Value = (**3000** - **3000**) / **4000**

Answer = ________

Task C: Scale the Richest Income (7000).

New Value = (**7000** - **3000**) / **4000**

Answer = ________

**Notice how all numbers (3000 to 7000) transformed into decimals between 0 and 1.*

3. Code Implementation

We use the `MinMaxScaler` from the sklearn library.

Task: Type the code to scale Income and JobStability.

```python
from sklearn.preprocessing import MinMaxScaler

# Initialize the Scaler
scaler = MinMaxScaler()

# Scale Income individually
# Note: We use [['double brackets']] to make it a 2D table for the tool
df['Income'] = scaler.fit_transform(df[['Income']])

# Scale JobStability individually
df['JobStability'] = scaler.fit_transform(df[['JobStability']])

# View the final result
df.head()
```

Going Further: Standard Scaling

We focused on Min-Max Scaling today. However, in the industry, **Standard Scaling (Z-Score)** is also very popular. It centers data around 0 (using Mean and Standard Deviation) and handles outliers slightly better than Min-Max.

Recommendation: You are encouraged to explore the code for Standard Scaling on your own!

— End of Workshop —