TP4: Debugging & Profiling

Exercise 1 : Debug a Logistic Regression Model

Task:
Given a Python file containing a Logistic Regression implementation using scikit-learn.
The model trains but produces very low accuracy and several warnings.

Issues include:

- Incorrect data preprocessing
- Scaling applied after train-test split
- Wrong label encoding
- A bug in model fitting section
- Misaligned features and labels

Goal:
Find all bugs, correct the data preprocessing steps, and achieve at least 80% accuracy.

***Deliverable*:**

- Debugged .py file
- Explanation of each bug and how you fixed it
- Final accuracy score

Exercise 2 : Profile a Slow Pandas Workflow

***Task*:**
Receive a script that loads a CSV data (ex. 100k rows) and performs many slow operations:

- Multiple .apply() functions
- Inefficient loops
- Repeated merges
- Re-reading the file in a loop

Use Python's built-in profiling tools:

- cProfile
- line_profiler
- time.perf_counter()

***Goal*:**
Identify the top 3 slowest parts of the program and optimize them using:

- Vectorization
- map() / replace()
- Efficient joins
- Caching

Deliverable:

- Profiling output screenshot or text
- Optimized code
- A short report: Before vs After (time-improvement table)

Exercise 3 : Debug a Neural Network Training Script (PyTorch or Keras)

Task:
A simple neural network fails to train properly:

Symptoms:

- Loss not decreasing
- Accuracy stuck
- Model predicting same class repeatedly
- Possible gradient or shape mismatch error

Potential hidden bugs:

- Wrong activation function
- Incorrect loss function
- Incorrect input tensor shape
- Learning rate too high/too low
- Data not normalized
- Batch dimension missing

Goal:
Fix all issues and get the model to learn with a decreasing loss over epochs.

Deliverable:

- Debugged training script
- Training curve screenshot (loss & accuracy)
- 3-5 sentences summarizing the fixes

Exercise 4 : Profile & Optimize a NumPy-Based ETL Pipeline

***Task*:**
A NumPy ETL pipeline does:

1. Load raw arrays
2. Clean missing values using loops
3. Perform normalization inside Python for loops
4. Compute statistics row by row
5. Save output to disk

The script is extremely slow.

Goal:
Use:

- np.nanmean, np.where, vectorized arithmetic
- Avoid Python loops
- Measure speed before and after using timeit

Deliverable:

- Optimized ETL script
- Short comparison table:
  | Task | Original Time | Optimized Time | Improvement % |
- Explanation of what vectorization changed

Exercise 5 : Build a Debugged & Optimized Full ETL Pipeline (Final Deliverable)

Task:
Combine what you learned. Receive an ETL dataset + buggy code that:

- Fails at some steps (missing column, wrong dtype, key error)
- Contains slow transformations
- Contains silent logical errors (dropping wrong columns, wrong groupby logic)
- Produces incorrect output

Goal:
Debug the entire pipeline AND optimize it.

You must fix:

- Data loading issues
- Data cleaning bugs
- Incorrect transformations
- Slow loops

- Output formatting

Deliverable:
A final working ETL pipeline containing:

1. Debugged Code
2. Profile Report (using cProfile or line_profiler)
3. Optimized Code
4. Before vs After performance comparison
5. Final clean dataset (CSV)