# Exercise 1 - Phase 1: Security, Privacy and Explainability in Machine Learning

**Student**: Marko Georgiev 12442103

**Mentor**: Prof. Rudolf Mayer

**Notice**: All supporting documentation (e.g., notebooks, code, datasets, sources, etc.), referenced are included in the .zip file submitted to TUWEL. Any other (external) sources or references can be located at the end of this document.

## Overview

This report serves as an overview of my activities during Phase 1 (P1) of Exercise 1 (E1) from the SPEiML (194.055) course at TU Wien. The exercise puts us in a scenario where we are given a fingerprinted dataset titled ‘Financial\_Records.csv’. This is a copy of the original data which has a fingerprint embedded, allowing the original owner to trace it back to us. As per the P1 description, we have gone rouge, and we now want to redistribute this dataset while ensuring the original owner cannot confirm this data came from us. In Section 1 (Benchmarking), this report contains the approach to assessing data utility and fidelity, both for the fingerprinted and attacked dataset. While Section 2 (Fingerprint Attack), contains the full attack methodology. Throughout the report there are sections dedicated to encountered issues and how I solved them. In the spirit of transparency, during the executing of P1 I used LLMs, and their uses will be explicitly stated where relevant.

## Benchmarking

After taking the time to fully comprehend the fingerprinting process covered in SBA research website, it was clear that I needed to set some sort of benchmark on the original dataset. Meaning, I had to have some way to compare the original and attacked dataset, to ensure the usability of the data, after my attack.

### Data Utility

Initially I wanted to use data utility in the context of model performance, but after many attempts I could not get a model to reliably produce accuracy >51% (mainly due to the data being limited inherently). Unfortunately, describing my attempts is beyond the scope of this report, but it was an interesting experience. Following the discussion on the TUWEL forum should give a clear idea why this approach was not potent.

### Data Fidelity

So, instead I decided to use a fidelity centric approach for benchmarking the data.

#### Choosing Fidelity Metrics

First, I used a python script at [[1]](#footnote-1)‘dataset-analysis.py’ which gave me a rough idea of the data. The script python code reads each column a .csv and tells me its range, average, standard deviation, and type (numeric, categorical, or mixed). This information is extremely useful as it allows me to choose specific fidelity metrics depending on the column characteristics. After researching online, I decided on these metrics:

* Mean Squared Error (MSE) for numeric columns
* Categorical agreement rate and Jensen-Shannon Divergence (JSD) for categorical columns

In our case the checking\_account is another mixed attribute, for which we will use MSE for the numeric values and apply Exact Match Rate and JSD for distributional comparison.

#### Preparing Data for Fidelity Checks

When assessing fidelity metrics, data still needs to be prepared. In my context, this was straight forward, as the different fidelity metrics clearly give their requirements for input data. Firstly, all numeric values should be converted to integer. This has to be done for all numeric columns (age, liable\_people, existing\_credits, duration, installment\_rate, credit\_amount, monthly\_rent\_or\_mortgage, residence\_since). For all categorical values (sex, marital\_status, job, credit\_hist, purpose, debtors, property, installment\_other, foreign, housing, tel, online\_banking,) we ensure that there are no trailing values, all are lowercase, and there are no missing values. Lastly, for the mixed attribute employment\_since containing two string options ‘unemployed’ and ‘1<year’. For this attribute we will define a custom mapping ‘unemployed’ to -1, and ‘1<year’ to 0, the rest of the values are converted to integers. The full preparation process can be found in the [[2]](#footnote-2)‘comparison-preparation-data.ipynb’ file.

## Fingerprint Attack

With that we finally get to the attack. My attack on the dataset, doesn’t attempt to remove the fingerprint, which would revert the data to the original state.

### Attack Plan

The first noticeable problem is that PID is an attribute not mentioned in the dataset description. This leads me to believe that it is no a mission critical attribute, and therefore likely a record identifier that helps with the fingerprint’s PRNG synchronization. Another assumption I am making is that fingerprint embedding follows the process described in the forum.

Therefore, my attack plan will target the PID in a way that makes it not easily recoverable. After this I will systematically perturb both categorical and numerical attributes across the dataset to introduce fingerprint bit collisions. I will introduce controlled modifications across the dataset to disrupt the fingerprint detection process, which will alter value distributions, introducing ambiguity in frequently marked attributes, and degrading the reliability of fingerprint bit reconstruction without significantly affecting overall data utility.

### Attack Execution

This was the fun part of the whole exercise. The first thing I did was create a script with the file name ‘dataset-shuffler.py’. The file takes a .csv file and shuffles the rows inside based purely on a key (passcode). I need this as I must shuffle the entries in the dataset, so that the attack on the PID makes sense. If I don’t shuffle the records, the owner can simply recompute the PID and find places where there has been tampering (or at least make it more difficult). Next, I implemented a full-scale fingerprint removal attack implemented in a file called ‘dataset-attack.py’. The next phase of the attack was chosen with the goal of exploiting the fingerprint generation structure described in the instructions of the SBA research website. I chose bit collision engineering. For this part of the attack, I aimed to confuse fingerprint detection by tweaking the statistical patterns in the data. I randomly swapped categorical values like marital\_status, job, and property with other common ones. For numeric fields such as credit\_amount and duration, I applied small random noise (around ±15%) and rounded the results to keep them as integers. I also flipped binary values like default with a 20% chance. This disrupted local distributions without breaking the dataset structure, to much.

## Results

1. code generated with DeepSeek LLM. [↑](#footnote-ref-1)
2. Comments and code sections generated by OpenAI LLM ChatGPT 4o [↑](#footnote-ref-2)