Homework 8

Exercise 1: [attack] What are Donald's Favourite Movies?

You have learned in class that it is possible to deanonymize most kinds of "anonymized" datasets. It just happens so that you have obtained several datasets containing "anonymized" movie ratings. They are databases stored as csv-files in the following format:

```
sha256(salt | email), sha256(salt | movie), date, rating
```

The salt is the same within each database. We call these the "COM-402 databases".

The emails and the movies in the databases are hashed, so they are perfectly private, right? Let's see if you can circumvent this protection mechanism. To de-anonymize a database, you also get csv-files with publicly available ratings from IMDb (we call them "IMDb databases") in the following format:

```
email, movie, date, rating
```

The entries in an IMDb database are a strict subset of the entries in a corresponding COM-402 database. Hence, there is some overlap between the ratings in an anonymized dataset and the public dataset.

You goal is to find out what movies a user with email `donald.trump@whitehouse.gov` has rated. **Disclaimer:** Any coincidences with real people are accidental.

You have three sub-exercises of increasing difficulty, the third being optional (we will not ask questions about it). Each exercise comes with its own pair of anonymized and public databases. You can solve each exercise in any language you like, even paper and pencil, if you're not afraid of searching for overlaps in 200 emails.

1. Dates are giving it away

Each user rated the movie at the same date in both the anonymized COM-402 and public IMDb database. Database files: `com402-1.csv`, `imdb-1.csv`.

Hint: some dates might have more than one rating, so you need to make sure you remove these duplicates.

2. More realistic

Here the dates are random, reflecting the fact that you might not rate a movie on Netflix and on IMDb on the same day. However, a simple frequency attack on the movies is enough to map the movies to the hashes. Database files: `com402-2.csv`, `imdb-2.csv`.

Hint: First match the movie hashes to their plaintext names by frequencies in the anonymized and private dataset.

Hint: Once you have mapped the hashes of the movies to the plaintext names of the movies, search for any user who rated all films you find in their public ratings.

3. Even more realistic (optional)

Here the dates of ratings in COM-402 database and IMDb are not the same, but similar. Dates are within 14 days, and are following a triangular distribution using Python's `random.choices` and weights: `[1, 2, ..., 14, 13, ..., 1]`.

Database files: `com402-3.csv`, `imdb-3.csv`.

Hint: First search for email hash/plaintext overlap and fit those to find the hash of the victim's email. Then you can search for the closest overlap of the public ratings and the anonymous ratings of your email.

Getting the database files and solutions

To get the database files, create a Docker container from `com402/hw8` image:

```
docker create com402/hw8
```

Noting the returned container ID as `id`, you can now copy all the databases from the `/anon_data` folder on the container:

```
docker cp "<id>:/anon_data/" .
```

To check your solutions, get the lists of movies and hashes from the `/real_data` folder:

```
docker cp "<id>:/real_data/" .
```

Exercise 2: [defense] Differentially Private Queries

This time, you play the role of a company that collects movie ratings, e.g., IMDb, and you want to securely disclose them. You have a rating database in clear, where each row has the format `[email, movie, date, rating]`. By now you know well that creating anonymized datasets is very hard, so releasing a database with pseudonymized (e.g., hashed) identifiers is not an option. You still want to enable researchers to learn some useful information from your database, but not at the cost of the privacy of your users. Hence, you decide to allow the researchers to send you queries that you can respond to in a differentially private way. This should help preserve the privacy of your users while still allowing the researchers to learn some insights about the data.

You allow counting queries of a certain format. The researchers should be able to get the number of people who have rated a given movie greater or equal than a certain level. If you were to express this in terms of SQL:

```
SELECT COUNT(*)
FROM db
WHERE movie = [queried movie name]
      AND rating >= [queried rating level]
```

This will enable them to learn which movies are rated well and which ones are not. As this is a typical example of a count query, you use the Laplace mechanism to ensure differential privacy (DP). Recall that the Laplace mechanism works as follows: to satisfy epsilon-DP, to each response you will add some noise drawn from Laplace distribution with the scale parameter that depends on the sensitivity of the count query and the epsilon. Refer to the lecture material for more detail.

The researchers pass a thorough vetting process, so you can safely assume they do not collude: will not share outputs they receive with each other. Each researcher gets their own **privacy budget**: the total epsilon for all the queries of this researcher. With each individual query the researcher specifies what value of epsilon should be used for the query. The lower the epsilon, the more noise will have to be added to the response (and more privacy the users will have). The higher the epsilon, the more accurate will the response be. Each query will always have to be within the total epsilon budget of the researcher.

Hint: Use sequential composition property of differential privacy to figure out how to keep account of the privacy budget and ensure that the queries do not exceed it.

Hint: In your database, a user can only rate one movie once. This assumption allows you to compute the sensitivity of a single query.

To implement this querying system, you have to code up a a class `DpQuerySession` with a property `remaining_budget` and a method `get_count`. Start with the following annotated template (best to copy from the online/Markdown version):

```python
# dp.py

import csv
import attr


class BudgetDepletedError(Exception):
    pass


@attr.s
class Rating:
    """Movie rating."""
    user = attr.ib()
    movie = attr.ib()
    date = attr.ib()
    stars = attr.ib()

# Unsure what Rating this? It is a convenient alternative to namedtuple.

class DpQuerySession:
    """
    Respond to database queries with differential privacy.

    Args:
        db (str): Path to the ratings database csv-file.
        privacy_budget (float): Total differential privacy epsilon for the session.
    """

    def __init__(self, db, privacy_budget):
        self.db = db
        self.privacy_budget = privacy_budget
        self._load_db()

    def _load_db(self):
        """Load the rating database from a csv-file."""
        self._entries = []
        with open(self.db) as f:
            reader = csv.reader(f, quotechar='"', delimiter=",")
            for email, movie, date, stars in reader:
                self._entries.append(
                    Rating(user=email, movie=movie, date=date, stars=int(stars))
                )

    @property
    def remaining_budget(self):
        """
        Calculate the remaining privacy budget.

        Returns:
            float: The remaining privacy budget.
        """
```

```
        return 0

    def get_count(self, movie_name, rating_threshold, epsilon):
        """
        Get the number of ratings where a given movie is rated at least as high as threshold.

        Args:
            movie_name (str): Movie name.
            rating_threshold (int): Rating threshold (number between 1 and 5).
            epsilon: Differential privacy epsilon to use for this query.

        Returns:
            float: The count with differentially private noise added.

        Raises:
            BudgetDepletedError: When query would exceed the total privacy budget.
        """
        # WARNING: Do not convert the response to positive integers. Leave as a
        # possibly negative float. This is a requirement for our verification.
        #
        # Question: Converting to a positive integer does not affect privacy. Why?

        return 0
```

This is an example using the `DpQuerySession`:

```
querier = DpQuerier("imdb-dp.csv", privacy_budget=1)
count = querier.get_count("Seven Samurai", rating_threshold=3, epsilon=0.25)
```

You can get the `imdb-dp.csv` database from the hw8 container's `/dp` folder:

```
docker cp <id>:/dp/imdb-dp.csv .
```

Verifying your solution

From your current folder where your `dp.py` file resides, run the following to verify your solution:

```
docker run -v $(pwd):/dp/solution com402/hw8
```

Our verification script will do its best to guide you to the right implementation. **We recommend that you do not inspect the exact content of our verification script and try to figure out on your own how to translate the theory of differential privacy into practice.**

If you want to run `dp.py` outside of our Docker container, make sure you have installed the `attrs` package in your Python environment with `pip install attrs`, and you are using Python 3.