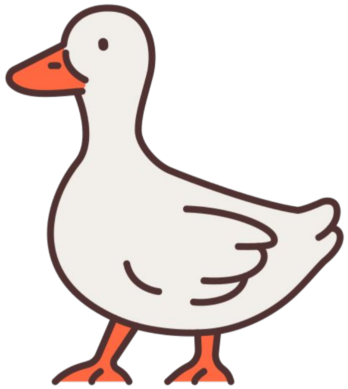
0x27 Privacy
Exercise Session

Data de-anonymization

- **Netflix Prize**
  - An open competition for the best algorithm to predict user ratings for films
  - Netflix Prize Dataset: Netflix released anonymous ratings of 500,000 Netflix users
- **Netflix data de-anonymization**
  - “How to Break Anonymity of the Netflix Prize Dataset” Narayanan and Shmatikov
  - correlated rankings with IMDB records to de-anonymize users

Ex1. What are Donald’s favorite movies



Setting

- Goal: de-anonymize “anonymized” datasets
- Download zip file from Moodle
- “anonymized” COM-402 databases - similar to Netflix DB
  - `com402-1.csv`
  - `com402-2.csv`
  - `com402-3.csv`

  `sha256(salt | email), sha256(salt | movie), date, rating`
- Public IMDB databases
  - `imdb-1.csv`
  - `imdb-2.csv`
  - `imdb-3.csv`

  `email, movie, date, rating`
- Task: find out what movies a user with email ``donald.trump@whitehouse.gov`` has rated.
- The answer for true movies are given in files:
  - `user-1.csv`
  - `user-2.csv`
  - `user-3.csv`

3 parts

Ex1-1: Dates are giving it away

- Each user rated the movie at the same date in both datasets

Ex1-2: More realistic

- Dates are randomized, reflecting the fact that you might not rate a movie on Netflix and on IMDb on the same day.

Ex1-3: Even more realistic (optional)

- Dates are within 14 days, and are following a triangular distribution using Python's `random.choices` and weights: `[1, 2, …, 14, 13, …, 1]`.

Ex. 1.1)

$$\boxed{\mathrm{IMDB}_1 \subset \mathrm{COM\text{-}402}_1}$$

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

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

- You can test with the user-*.csv file if the guesses are true
  - `assert movie_guesses == true_movies`
  - `print(movie_guesses)`

Ex. 1.1) - solution idea

- If the date and stars match in public and anonymized dataset, then record the candidate plaintext email and movie
  - `hash2movie[anon_entry.movie].append(pub_entry.movie)`
  - `user2hash[pub_entry.user].append(anon_entry.user)`
- Guess the victim's email hash as the most common candidate.
  - `Get_most_common helper method`
- Filter the ratings made by this victim’s email hash

Ex. 1.2)

$$\boxed{\text{IMDB}_2 \subset \text{COM-402}_2}$$

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

Where $\text{randomize} : \mathcal{D} \to \mathcal{D}$ maps each date (uniquely) to some random date

Ex. 1.2) - solution idea

- Match movie hashes to plaintexts by frequency.
  - `Sort_by_freq method in helpers.py`
  - `movie2hash[movie_name] = movie_hash`
  - `hash2movie[movie_hash] = movie_name`
- Get the victim's movie hashes from the public data.
  - `filter_ratings_by_user(public, victim_email)`
- Identify the victim's email hash:
  - Map the email hashes to the corresponding movie hashes from the anonymized data.
  - find the hash that has all the movie hashes from above.
    - If multiple possibilities, then the generated datasets are not good.

Ex. 1.3) [OPTIONAL]

$$\boxed{\mathrm{IMDB}_3 \subset \mathrm{COM\text{-}402}_3}$$

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

Where $\mathrm{real\_randomize} : \mathcal{D} \rightarrow \mathcal{D}$ maps randomly maps each date according to a triangular distribution within 14 days of the initial date

Ex2. Differentially Private Queries

[optional]

Ex2

- Playing the role of the company: IMDB
- Researchers send you queries, you respond in a differentially private way
- Query : “For a given movie, how many reviews are above a threshold?”
    - `get_count(movie_name, rating_threshold, epsilon)`
- Each researcher gets their own **privacy budget** $\varepsilon_{\text{total}}$: the total epsilon a researcher can use across queries.
- for each query, an epsilon value can be specified
    - the higher epsilon is, the higher the accuracy of the response will be, but will allow for less queries. and vice versa, a lower epsilon will allow for more queries.

Laplace mechanism for adding noise

Assume f is a scalar function, i.e., $f : \mathcal{D} \rightarrow \mathbb{R}$

Return $\boldsymbol{A(D) = f(D) + \mathrm{Lap}\left(\frac{\Delta f}{\epsilon}\right)}$

The ground truth answer to the query

Noise!

Privacy guarantee

Selected by querier,
Higher⇔more accurate results, but less queries

$\mathrm{Lap}\left(\frac{\Delta f}{\epsilon}\right)$ is **noise** drawn from a **Laplacian** distribution of parameter $\frac{\Delta f}{\epsilon}$

Δf is the **sensitivity** of function f:
$$\Delta f = \max_{r} |f(D) - f(D_{-r})|$$

- In your database, a user can only rate one movie once. This means sensitivity Δf =1
- `np.random.laplace(loc=0, scale=1. / epsilon)`

Sequential composition

- Sequential composition property: **If algorithms A_1, A_2, …, A_k** use [independent randomness]() and each **A_i** satisfies ε_i-differential privacy, respectively. Then (A_1, A_2, …, A_k) is **($\varepsilon_1 + \varepsilon_2 + \ldots + \varepsilon_k$)**-differentially private
    - can keep account of spent privacy budget and ensure that the queries do not exceed it.

Testing with verify.py

```
(base) ➜  solution git:(main) ✗ /usr/local/anaconda3/bin/python

Running tests...

> Basic privacy budget accounting
PASSED 👌

> Privacy budget depletion control
PASSED 👌

> DP noise distribution
PASSED 👌

> Multiple query behavior (I)
PASSED 👌

> Multiple query behavior (II)
PASSED 👌
```