Problem 1

Task 1 Sensitivity

Given a dataset of salaries, assume that all numbers are in the range (0,900K]. What is the sensitivity of each of the following queries?

Query 1. The number of people with a salary above 100K.
Query 2. The histogram of the number of people with salary in each of (0, 100K], (100K, 200K], $\cdots$, (800K, 900K].
Query 3. The histogram of the number of people with salary in each of (0, 10K], (10K, 20K], $\cdots$, (890K, 900K].
Query 4. The sum of the total salary.
Query 5. The medium salary.
Query 6. The mode, i.e., the number which appears most often in the set.

Note:

1. We discussed with the professor about the interval problems that he mentioned in the email on Feb 15. I will solve problems of hw1 based on given the interval on both sides are closed but I would not update the description and will solve problems based on it for convenience. This would not influence the solutions much.
2. My solutions are based on bounded differential privacy and unbounded differential privacy. For certain queries in hw1, there are two ways to get sensitivity based on bounded differential privacy and unbounded differential privacy. Reference of bounded differential privacy and unbounded differential privacy: https://programming-dp.com/ch9.html
3. Way 1 in my solutions means bounded differential privacy, Way 2 in my solutions means unbounded differential privacy

My Answer

Query 1. The number of people with a salary above 100K.

The sensitivity of "Query 1" is 1. Because a change in one record in the dataset would change the query result at most by 1. For example, the ith person's salary in this dataset is greater than 100k originally. If we change the ith person's salary to 50k, the Query 1 result that counts the number of people whose salary is above 100k will be changed by at most 1 person. So, the maximum difference between D and D' is 1.

Query 2. The histogram of the number of people with salary in each of (0, 100K], (100K, 200K], ···, (800K, 900K].

Way 1

The sensitivity of "Query 2." is 2. Here is my analysis based on modifying one record:

(1) If we change one record in a certain bin on the histogram, the new salary of this record could be still in the range of this bin. It does not change the frequency of this bin. In this case, the sensitivity is 0;
(2) If we change one record in a certain bin on the histogram, the new salary of this record could be located in a range

of a different bin. The frequency of the two bins will be both changed. The frequency of one bin should be minus 1, and the frequency of the other bin will be added by 1. In this case, the sensitivity is 2.

Hence, the sensitivity of "Query 2." is 2

Way 2

The sensitivity of "Query 2." is 1. Here is my analysis based on deletion and addition: It will change the frequency of this bin at most by 1. In this case, the sensitivity is 1;

Query 3. The histogram of the number of people with salary in each of (0, 10K], (10K, 20K], ···, (890K, 900K].

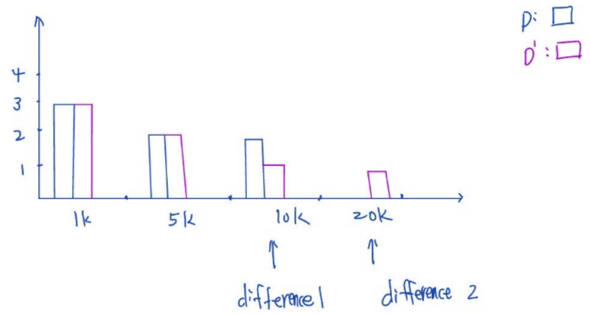
Way 1

The sensitivity of "Query 3" is 2**.** Here is my analysis: the solution shall be similar to Query 2. A modification in any single salary will change the query result of the corresponding bin at most by 1 and it also changed one other bin that is influenced by the changed record. Because we change the salary of a single record, it's still in the (0, 900K]. So the frequency of the influenced bins will add by 1 or reduce by 1. Hence, the maximum difference between D and D' is 2.

$D = \{c_1, c_2, \cdots, c_n\} = \{1k, 1k, 1k, 5k, 5k, 10k, 10k\}$ belongs to (0, 10k]

$D' = \{c_1, c_2, \cdots c_n\} = \{1k, 1k, 1k, 5k, 5k, 10k, 20k\}$ belongs to (0, 10k] and (10k, 20k]

There's only one change. Let's see the distribution of the histogram.



We see changing one record will influence the frequency of two bins.

Hence, the sensitivity is 2

Way 2

The sensitivity of "Query 3." is 1. Here is my analysis based on deletion and addition:

If we add or delete one record in a certain bin on the histogram that the salary of the new record shall be in the range of a certain bin. It will change the frequency of this bin at most by 1. In this case, the sensitivity is 1;

Query 4. The sum of the total salary.

Given the sum of the total salary is m on the original dataset. If we change the salary of one record, the query result that the sum of the total salary will be changed to at most 900k. Given the new sum of the total salary is n, $S(q) = \max_{\{D, D'\}} |q(D) - q(D')| = |m-n|$ and given an extreme case: D={0, 900}, D' = {0, 0}, $S(q) = \max_{\{D, D'\}} |q(D) - q(D')| = |m-n| = 900$; Hence, the sensitivity is very close to 900k. Since the professor update the description, the interval is closed, then the sensitivity is 900k.

Query 5. The medium salary.

The sensitivity of Query 5 is 450k. Here is my analysis: since the salary of all members is in the range (0, 900K]. If a single record is changed in the dataset, its salary should be still in the range (0, 900K]. Given that an extreme case is {0k, 900k}, the medium salary here is 450k. If we change one record to get D'={0K, 0K}, in this extreme case, the medium salary is 0k. In this extreme case, sensitivity of Query 5 is very close to 450. Hence, in the range [0, 900k], the sensitivity of Query 5 is 450k.

Query 6. The mode, i.e., the number which appears most often in the set.

The sensitivity of Query 6 is 900k. Given the extreme case D={0k, 900k, 900k}, mode of D is 900k. Then change one record in dataset D, we get D' = {0k, 0k, 900k}, the mode of D' is 0; Hence, the sensitivity of Query 6 is 900k.

Task 2 Error and variance for Randomized Response

Derive the expected error and variance of the best estimate πhat shown in Slide 38 (each 4 points).

My answer:
The formula on Slide 38 is based on the experiment of flipping a coin.

π: True fraction of uers answering "yes"
p: Probability coin falls heads

$Y_i = 1$, if the i^{th} user says "yes"
$\quad = 0$, if the i^{th} user says "no"

$p_{yes} = p(Y_i = 1)$
= (True answer = yes AND coin = heads) OR
(True answer = no AND coin = tails)
$= \pi p + (1-\pi)(1-p)$

$p_{no} = P(Y_i = 0) = \pi(1-p) + (1-\pi)p = 1 - p_{yes}$

Likelihood: $L = C^n_{n1}\, p_{yes}^{n1}\, p_{no}^{(n-n1)}$

Most likely value of π: (by setting $dL/d\pi = 0$)
=> $\boldsymbol{\pi_{hat} = \{n1/n - (1-p)\}/(2p-1)}$

(1)
Since the experiment is a binomial distribution,

The expected value of a random variable X is denoted by E(X).
E(X) is calculated as the sum of the product of each possible outcome and its corresponding probability. Mathematically,

$E(X) = \sum_1^n x(P(x)) = x_1 P_1 + x_2 P_2 + \cdots + x_n P_n$

In the experiment, we only have two possibilities: $P_1 = P_{Yes}$, $P_2 = P_{No}$

Hence, $E(X) = P_1 x_1 + P_2 x_2 = P_{Yes} X_{yes} + P_{No} X_{no}$. π_{hat} is the best estimate.

we get $P_{No} = 0$.

Hence the expected value for π_{ha} is $E(\pi_{hat}) = P_1 X_1 = P_{Yes} X_{Yes} = \pi$

Note: π is the fraction of users answering 'Yes'

(2)

(1) The formula we get is based on an experiment flipping a coin that we can get a randomized response schema on a binary distribution. It's also called binomial distribution. The variance of binomial distribution is $Var(X) = np(1-p)$

X is the number of "Yes" responses, n is the total number of responses

p is the probability of a "Yes" response. Then we can rewrite the variance formula like this way:

$$Var(X) = np(1-p) \Rightarrow Var\left(\frac{X}{n}\right) = \frac{p(1-p)}{n} = \frac{\pi(1-\pi)}{n}$$

where $\frac{X}{n}$ is the proportion of "Yes" responses.

(2) The delta method is a statistical technique used to estimate the variance of a function of a random variable. Use the delta method to find the best estimate π hat:

the derivative of π hat with respect to π: $\frac{d\pi_{hat}}{d\pi} = 1 - \left(\frac{1-p}{2\pi p-p-1}\right)^2$

(3) Find the variance of the best estimate π_{hat}:

$$Var(\pi hat) = \left(\frac{d\pi_{hat}}{d\pi}\right)^2 \times Var(\pi)$$

substituting the expressions in (1), (2):

$$Var(\pi hat) = \left[1 - \left(\frac{1-p}{2\pi p-p-1}\right)^2\right]^2 \times \frac{\pi(1-\pi)}{n}$$

$$= \frac{\pi(1-\pi)}{n} \times \left[1 - \frac{2(1-p)}{(2\pi p-p-1)^2} + \frac{(1-p)^2}{(2\pi p-p-1)^4}\right]$$

$$= \frac{\pi(1-\pi)}{n} - \frac{2(1-p)\pi(1-\pi)}{(n(2\pi p-p-1)^2)} + \frac{(1-p)^2\pi(1-\pi)}{n(2\pi p-p-1)^4}$$

To simplify, use: $2\pi p - p - 1 = (2\pi-1)(2p-1)$

$$Var(\pi_{hat}) = \frac{\pi(1-\pi)}{n} - \frac{2(1-p)\pi(1-\pi)}{n(2\pi-1)^2(2p-1)^2} + \frac{(1-p)^2\pi(1-\pi)}{n(2\pi-1)^4(2p-1)^4}$$

combine the terms with a common denominator:

$$\pi(hat) = \frac{\pi(1-\pi)}{n} - \left[\frac{1}{(2\pi-1)^2(2p-1)^2} - \frac{2(1-p)}{(2\pi-1)^2(2p-1)^2} + \frac{(1-p)^2}{(2\pi-1)^2(2p-1)^2}\right] \times \frac{\pi(1-\pi)}{n}$$

Simply the middle term in the squre brackets,

$$\frac{1}{(2\pi-1)^2(2p-1)^2} - \frac{2(1-p)}{(2\pi-1)^2(2p-1)^2} + \frac{(1-p)^2}{(2\pi-1)^2(2p-1)^2}$$

$$= \frac{1}{(2\pi-1)^2(2p-1)^2} - \frac{2}{(2\pi-1)(2p-1)} + \frac{1}{(2\pi-1)^2(2p-1)^2}$$

Based on above, further simplify, we get

$$Var(\pi_{hat}) = \frac{\pi(1-\pi)}{n} + \frac{1}{n(16(p-0.5)^2 - 0.25)}$$

Task 3 Composition Theorems Proofs

In the class, we have shown the proof of ϵ-Differential Privacy for the Sequential Composition theorem. Following that, prove the Parallel Composition and Postprocessing theorems

Show the proof of ϵ-Differential Privacy for parallel Composition theorem

Considering the definition of ϵ-differential privacy for a single mechanism M: Pr[M(x) ∈ S] ≤ e^ϵ Pr[M(x') ∈ S] + δ where ϵ is the privacy parameter and δ is the privacy budget. Suppose we have k mechanisms M_1, M_2, ..., M_k, and we apply each mechanism to the corresponding user's data to obtain the output for that user. Let y = (y_1, y_2, ..., y_k) be the joint output of all the mechanisms for the data set x, and y' = (y'_1, y'_2, ..., y'_k) be the joint output of all the mechanisms for the data set x'. Since the mechanisms are independent, we can write the joint probabilities as products of the individual probabilities:
Pr[M(y) ∈ S] = Pr[M_1(y_1) ∈ S] * Pr[M_2(y_2) ∈ S] * ... * Pr[M_k(y_k) ∈ S]
Pr[M(y') ∈ S] = Pr[M_1(y'_1) ∈ S] * Pr[M_2(y'_2) ∈ S] * ... * Pr[M_k(y'_k) ∈ S]
By the definition of differential privacy for each mechanism M_i, we know that: Pr[M_i(y_i) ∈ S] ≤ e^$ε_i$ Pr[M_i(y'_i) ∈ S] + δ/k
Substituting this into the above ratio, we get: Pr[M(y) ∈ S] / Pr[M(y') ∈ S] ≤ e^$ε_1$ * e^$ε_2$ * ... * e^$ε_k$ = e^($ε_1$+$ε_2$+...+$ε_k$)
Use the fact that e^a + e^b ≤ e^max(a,b) for any real numbers a and b, to obtain:
e^($ε_1$+$ε_2$+...+$ε_k$) = e^max($ε_1$, $ε_2$, ..., $ε_k$) + e^($ε_1$+$ε_2$+...+$ε_k$) - e^max($ε_1$, $ε_2$, ..., $ε_k$) ≤ 2 * e^max($ε_1$, $ε_2$, ..., $ε_k$)
Substituting this back into the ratio, we get: Pr[M(y) ∈ S] / Pr[M(y') ∈ S] ≤ 2 * e^max($ε_1$, $ε_2$, ..., $ε_k$)
Choosing δ = 0, we can apply the definition of ϵ-differential privacy to get: Pr[M(y) ∈ S] ≤ e^max($ε_1$, $ε_2$, ..., $ε_k$)
Hence, ε = max(ε1 , ... , εk)

Show the proof of ϵ-Differential Privacy for Postprocessing Composition theorem

If M is an ε-differentially private algorithm that accesses a private database D, then any additional post-processing A(M(D)) also satisfies e-differential privacy. Here is my proof:
Let M be a randomized algorithm that satisfies ϵ-differential privacy, and let f be any post-processing function that maps the output of M to some other value. Then, the composition of M and f, denoted by M ∘ f, also satisfies ϵ-differential privacy. For any two adjacent datasets x and x' that differ in the data of a single individual i, and any subset of outputs S, we have: Pr[M(x) ∈ S] ⩽ e^ε Pr[M(x') ∈ S]. Let's consider the output of M composed with the post-processing function f, denoted by M ∘ f. Let y and y' be the outputs of M ∘ f on datasets x and x', respectively. Then, we have: y = f(M(x)), y' = f(M(x')).

Since f is a deterministic function, we can write: Pr[M ∘ f(x) ∈ S] = Pr[y ∈ S | M(x) ∈ f^{-1}(y)].
Similarly: Pr[M ∘ f(x') ∈ S] = Pr[y' ∈ S | M(x') ∈ f^{-1}(y')].

Since they are the possible outputs of M that get mapped to y and y', respectively. We can use the fact that f is a post-processing function to write: Pr[y ∈ S | M(x) ∈ f^{-1}(y)] = Pr[f(M(x)) ∈ S | M(x) ∈ f^{-1}(y)].
Similarly: Pr[y' ∈ S | M(x') ∈ f^{-1}(y')] = Pr[f(M(x')) ∈ S | M(x') ∈ f^{-1}(y')].

Since M satisfies ϵ-differential privacy, we can use the definition of differential privacy to write:
Pr[M(x) ∈ f^{-1}(y)] ⩽ e^ε Pr[M(x') ∈ f^{-1}(y')]

Multiplying both sides of the above inequality with Pr[f(M(x)) ∈ S] and using the above expressions for Pr[y ∈ S | M(x) ∈ f^{-1}(y)] and Pr[y' ∈ S | M(x') ∈ f^{-1}(y')], we obtain: Pr[f(M(x)) ∈ S | M(x) ∈ f^{-1}(y)] * Pr[M(x) ∈ f^{-1}(y)] ⩽ e^ε Pr[f(M(x')) ∈ S | M(x') ∈ f^{-1}(y')] * Pr[M(x') ∈ f^{-1}(y')]

Since y and y' are outputs of M ∘ f on adjacent datasets, we have: M(x) ∈ f^{-1}(y) if and only if M ∘ f(x) = y
Similarly: M(x') ∈ f^{-1}(y') if and only if M ∘ f(x') = y' ➔ Pr[M ∘ f(x) = y, f(M(x)) ∈ S] ⩽ e^ε Pr[M ∘ f(x') = y', f(M(x')) ∈ S]

Problem 2: Local Differential Privacy

Task 1: Frequency Oracle:

We want each user to report a value that has a domain of d = 100 values, in a way that satisfies ϵ-local differential privacy for ϵ = ln 4.

1

When using generalized randomized response, what probability should one report the value without change? What probability should one report the value with a change? Please answer using a common fraction.

a domain : $d = 100$

$\varepsilon = \ln 4$

1. For one report the value without change :

$$P_{1\to1} = P_{0\to0} = p = \frac{e^{\varepsilon/2}}{e^{\varepsilon/2}+1} = \frac{e^{\frac{\ln 4}{2}}}{e^{\frac{\ln 4}{2}}+1} = \frac{\left(e^{\ln 4}\right)^{\frac{1}{2}}}{\left(e^{\ln 4}\right)^{\frac{1}{2}}+1} = \frac{(4)^{\frac{1}{2}}}{(4)^{\frac{1}{2}}+1} = \frac{2}{2+1}$$

$$= \frac{2}{3}$$

For one report the value with a change :

$$P_{1\to0} = P_{0\to1} = q = \frac{1}{e^{\varepsilon/2}+1} = \frac{1}{e^{\ln 4/2}+1} = \frac{1}{\left(e^{\ln 4}\right)^{\frac{1}{2}}+1} = \frac{1}{2+1}$$

$$= \frac{1}{3}$$

2

When using generalized randomized response, suppose each value is preserved with probability p. If a server collects 100,000 responses, among which 3,000 has a particular value. What is the expected number of responses on that value? What is the best estimate of the number of respondents who actually have that value by the server? Please answer with a formula involving p.

2. $d = 100$, $\varepsilon = \ln 4$; Suppose there are n people which are same as the number of responses a sever collected, then $n = n_v = 100\ 000$,

I_v = the number of reports on v = the number has a partialar value $= 3000$

The expected number is $E[I_v] = n_v \cdot p + (n - n_v) \cdot q = 100\ 000p$

The best estimate of the number of respondents:

$$C(v) = \frac{I_v - n \cdot q}{p - q} = \frac{3000 - 100000\,\varepsilon}{p - q}$$

3

When using unary encoding, each value is encoded using a 100-bit string with one bit being 1 and the other bits being 0. Every bit is randomly perturbed independently before being transmitted. When using the basic Rappor protocol, what is the probability that a 1-bit is not changed? What is the probability that a 0-bit is not changed?

3. Since every bit is randomly perturbed, we can just apply the probability formula for unchange bit: $P_{1\to1} = P_{0\to0}$

$$P_{1\to1} = P_{0\to0} = p = \frac{e^{\varepsilon/2}}{e^{\varepsilon/2} + 1} = \frac{(e^{\ln 4})^{\frac{1}{2}}}{(e^{\ln 4})^{\frac{1}{2}} + 1} = \frac{\sqrt{4}}{\sqrt{4} + 1} = \frac{2}{3}$$

Thus, the probability that 1-bit or 0-bit is not changed are both $\frac{2}{3}$

4

When using the Optimized Unary Encoding protocol, what is the probability that a 1-bit is not changed? What is the probability that a 0-bit is not changed?

4. For the probability that a 1-bit is not changed: $P_{1\to1} = \frac{1}{2}$

For the probability that 0-bit is not changed: $P_{0\to0} = \frac{e^{\varepsilon}}{e^{\varepsilon} + 1} = \frac{e^{\ln 4}}{e^{\ln 4} + 1}$

$$= \frac{4}{4+1} = \frac{4}{5}$$

Task 2: Heavy Hitter Discovery

Given a set of n values V = {v1, v2, · · · , vn} from n users, where the values are from a bounded domain D. Suppose each value vi is represented as a binary string with length m (e.g., when m = 4, vi's value is 7, then vi = 0111; vi's value is 8, then vi = 1000). The naïve approach of querying the frequency of each string requires 2m oracle queries and is infeasible when m is large. Now your goal is to design a Local Differential Privacy protocol to identify the top-k heavy hitter, i.e., the k most frequent values in V , such that it is computationally feasible to query the frequency oracle.

Straw man protocol

A length-m value v is divided into g equal-size segments, each of length s = m/g. In this protocol, each user randomly chooses a segment to report, and the aggregator first queries the frequency of each length-s binary string in each of the g segments, and then identifies the frequent patterns in each segment, which are denoted as C1, C2, · · · , Cg. The candidate set C is the Cartesian product of {Ci}'s, i.e., C = C1 × C2 × · · · × Cg, where Cartesian product operation × is defined as C1 × C2 = {ci||cj : ci ∈ C1 and cj ∈ C2}, and || is the string concatenation operation. Finally, the aggregator queries the frequencies of the strings in candidate C. Answer the following questions:

1. What is the number of total frequency oracle queries using this protocol?
2. What is the size of the candidate set C for top-k hevay hitter discovery?

1. ① Straw man protocol (Find one most frequent value from D)

m: length, g: size of each equal-size segment

s: each of lengths : $s = \frac{m}{g}$

Because V_i is a binary string, for each segment has 2^m possible queries

For g segments, the total oracle queries is $2^m \times g$

② Because $C = C_1 \times C_2 \times \cdots \times C_g = k_1 \times k_2 \times \cdots \times k_g$

k most frequent values in $V_i \in D$,

the size of candidate set $C = \sum_1^g k_i \Rightarrow k^g$

① The number of total frequency orcale queries for straw man protocal is $2^m \times g$

② The size of the candidate set C for top-k heavy hitter discovery is k^g

Segment pair protocol:

This protocol improves upon the Straw man protocol. The key difference is that, instead of reporting only one segment from g segments, each user reports a pair of two randomly chosen segments. The detailed protocol is as follows: First, the aggregator identifies the frequent patterns in each of the g segments. Then, it queries, for each pair i, j of segments, the frequency for the values in Ci × Cj and identifies the value pairs that are frequent in segments i, j. From the frequent value pairs for each pair of segments, the aggregator recovers candidates for frequent values for the whole domain, using the a priori principle that if a value v ∈ D is frequent, every pair of its segments must also be frequent.

Answer the following questions:

1. What is the number of total frequency oracle queries using this protocol?
2. What is the expected number of user reports on each pair of segments?

2. Segment pair protocol

There are n users, each user can report a pair of 2 randomly segments from g segment $\Rightarrow$ the population is divide into C_2^g groups

If n users are reporting, the $\frac{n}{\frac{g}{2}}$ users will be expected to report on each segment. Hence, the number of total frequency oracle queries using this protocol is equal to the number of pairs of segments:

$$C_2^g = \frac{g!}{(g-2)!\,2!} = \frac{g\cdot(g-1)\,(g-2)!}{(g-2)!\;2!} = \frac{g\,(g-1)}{2}$$

The sample size here is C_2^g, the expected value $= \frac{\cancel{C_2^g}}{\frac{g(g-1)}{2}} \times \frac{n}{\cancel{C_2^g}}$

$= \frac{n}{g(g-1)/2}$

① The number of total frequency oracle queries is $\frac{g(g-1)}{2}$

② The expected number of user reports on each pair of segments is $\frac{n}{g(g-1)/2}$

Prefix Extending protocol

Assume you want to identify k = 150 most frequent values using the Prefix Extending protocol. The input domain D is 15 bytes (i.e., 120 bits), and you want to limit the total number of frequency oracle queries to no more than 228.

1. How to design the Prefix Extending protocol?

2. Which frequency oracles can be used to achieve high accuracy?

1. How to design the Prefix Extending protocol?

1) Divide the input domain D into a fixed number of equal-sized segments. For example, we can divide D into 2^15 segments, each containing 2^9 possible values.
2) Initialize a dictionary for each segment. Use a separate dictionary for each segment to reduce the number of frequency oracle queries needed for each segment.
3) Read in the input data and divide each value into segments using a simple bit-wise operation. For example, if each segment contains 2^9 possible values, we can use the upper 9 bits of each input value to determine the segment.
4) For each input value, use the dictionary corresponding to its segment to look up its frequency count. If the value is not in the dictionary, add it with a count of 1. If the value is already in the dictionary, increment its count.
5) Once all input values have been processed, combine the dictionaries for all segments and sort them by count to identify the most frequent values.
6) Use the codes assigned to these most frequent values to create an encoded representation of the input data that is smaller in size than the original.

2. Which frequency oracles can be used to achieve high accuracy?

It's not necessary to be designed to hash into one bit, hashing into a larger range, the result might be better. Hence, optimized local hash can be used to achieve high accuracy. Here is the reason: when g = e^ε + 1, can achieve better accuracy.

Problem 3: Implementing (Centralized) Differential Privacy

1 Laplace Mechanism

Step 0. Using the same dataset (UCI Machine Learning Adult data) as in Assignment 1 to study differential privacy.

Here's my code to use the dataset

In [99]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

adult_data_url = 'https://archive.ics.uci.edu/ml/machine-learning-databases/adult/adult.data'
adult_data = pd.read_csv(adult_data_url, header=None)

column_names = ['age', 'workclass', 'fnlwgt', 'education', 'education_num', 'marital_status',
                'occupation', 'relationship', 'race', 'sex', 'capital_gain', 'capital_loss',
                'hours_per_week', 'native_country', 'income']

adult_data.columns = column_names
# adult_data.head()
len(adult_data.index)
```
Out[99]:
```
32561
```

Laplace Mechanism: Query the average age of the records (each record is an individual) with age > 25.
Inject Laplacian noise to the query result (i.e., average age) to ensure ϵ-differential privacy with $\epsilon = 0.5, 1.0$.

Step 1. In case of $\epsilon = 0.5$, generate 1,000 results for the query over the original dataset and generate 1,000 results for the query over each of three other datasets: removing a record with the oldest age; removing any record with age = 26; and removing any record with the youngest age.

For generating the 4 datasets, I need to do data processing as the description requires and then generate 1000 results for each data set. After that, I get the query result (the average age of the records) based on the 4 data sets with the size of each data set is 1000 results.

Here are the ways and related code I did data processing.

(1) Query the average age of the records (each record is an individual) with age > 25 and get a new dataset that is with age > 25

In [100]:
```
# Query the average age of the records (each record is an individual) with age > 25
# Filter the data to only include records with age > 25 and calculate the average age
adult_data_age_greater25 = adult_data[adult_data['age'] > 25]
# The average age of data set with age > 25
original_average_age = adult_data_age_greater25['age'].mean()

# This is the initial query result on data set with age > 25
original_average_age = round(original_average_age, 2)
print('This is the initial query result on data set with age > 25 that average age is', original_average_age)
```
```
This is the initial query result on data set with age > 25 that average age is 42.78
```

(2) I get a new data set as shown above which is called adult_data_age_greater25. And then I generate 1,000 results and 4000 results separately for the query over the original dataset and get the query results that I will use in the following code

```
In [118]: # Generate 1,000 results for the query over the original dataset
df_1000 = adult_data_age_greater25.sample(n=1000, random_state=42)
# Query the average age of the records
df_average_age_1000 = df_1000['age'].mean()
df_average_age_1000 = round(df_average_age_1000, 2)
print('This is the query result by generating 1,000 results that average age is', df_average_age_1000)

# Generate 4,000 results for the query over the original dataset
df_4000 = adult_data_age_greater25.sample(n=4000, random_state=42)

# Query the average age of the records
df_mean_age_4000 = df_4000['age'].mean()
df_mean_age_4000 = round(df_mean_age_4000, 2)
print('This is the query result by generating 4,000 results that average age is', df_mean_age_4000)

This is the query result by generating 1,000 results that average age is 43.09
This is the query result by generating 4,000 results that average age is 42.87
```

(3) removing a record with the oldest age and then I generate 1,000 results and 4 000 results for this dataset and getting the query results that I will use in the following code

```
In [121]: # Data proccessing, remove a record with oldest age
oldest_age_record = adult_data_age_greater25['age'].max() # 90
df_without_a_oldest_age = adult_data_age_greater25[adult_data_age_greater25['age'] != oldest_age_record]

# Generate 1,000 results based on adult_data_age_greater25
df_without_a_oldest_age_1000 = df_without_a_oldest_age.sample(n=1000, random_state=42)
average_age_df_without_a_oldest_age = df_without_a_oldest_age_1000['age'].mean()
average_age_df_without_a_oldest_age = round(average_age_df_without_a_oldest_age, 2)
average_age_df_without_a_oldest_age
print('This is the query result by generating 1,000 results without_a_oldest_age that average age is', average_age_df_without_a_o

# Generate 4,000 results for the query over the dataset that removing a record with the oldest age
df_without_a_oldest_age_4000 = df_without_a_oldest_age.sample(n=4000, random_state=42)
mean_age_df_without_a_oldest_age = df_without_a_oldest_age_4000['age'].mean()
mean_age_df_without_a_oldest_age = round(mean_age_df_without_a_oldest_age, 2)
mean_age_df_without_a_oldest_age
print('This is the query result by generating 4,000 results without_a_oldest_age that average age is', mean_age_df_without_a_olde

This is the query result by generating 1,000 results without_a_oldest_age that average age is 42.58
This is the query result by generating 4,000 results without_a_oldest_age that average age is 42.46
```

(4) Generate 1,000 results and 4,000 results separately for the query over the dataset that removing any record with age = 26 and get the query results that I will use in the following code

```
In [122]: # Generate 1,000 results for the query over the dataset that removing any record with age = 26
df_without_26 = adult_data_age_greater25[adult_data_age_greater25['age'] != 26]
df_without_26_1000 = df_without_26.sample(n=1000, random_state=42)

average_age_df_without_26 = df_without_26_1000['age'].mean()

# Here is the query result based on the new data set
average_age_df_without_26 = round(average_age_df_without_26, 2)
print('This is the query result by generating 1,000 results average_age_df_without_26 that average age is', average_age_df_witho

# Generate 4,000 results for the query over the dataset that removing any record with age = 26
df_without_26_4000 = df_without_26.sample(n=4000, random_state=42)

mean_age_df_without_26 = df_without_26_4000['age'].mean()

# Here is the query result based on the new data set
mean_age_df_without_26 = round(mean_age_df_without_26, 2)
print('This is the query result by generating 4,000 results average_age_df_without_26 that average age is', mean_age_df_without_

This is the query result by generating 1,000 results average_age_df_without_26 that average age is 42.93
This is the query result by generating 4,000 results average_age_df_without_26 that average age is 43.14
```

(5) Generate 1,000 results, 4,000 results separately for the query over the dataset that removing any record with the youngest age and get the query results that I will use in the following code

```
In [123]: # Generate 1,000 results for the query over the dataset that removing any record with the youngest age
youngest_age = adult_data_age_greater25['age'].min()
df_without_youngest_age = adult_data_age_greater25[adult_data_age_greater25['age'] != youngest_age]

df_without_youngest_age_1000 = df_without_youngest_age.sample(n=1000, random_state=42)
average_age_df_without_youngest_age = df_without_youngest_age_1000['age'].mean()

# Here is the query result based on new data set
average_age_df_without_youngest_age = round(average_age_df_without_youngest_age, 2)
average_age_df_without_youngest_age
print('This is the query result by generating 1,000 results average_age_df_without_youngest_age that average age is', average_age

# Generate 4,000 results for the query over the dataset that removing any record with the youngest age
df_without_youngest_age_4000 = df_without_youngest_age.sample(n=4000, random_state=42)
mean_age_df_without_youngest_age = df_without_youngest_age_4000['age'].mean()

# Here is the query result based on new data set
mean_age_df_without_youngest_age = round(mean_age_df_without_youngest_age, 2)
mean_age_df_without_youngest_age
print('This is the query result by generating 4,000 results average_age_df_without_youngest_age that average age is', mean_age_d

This is the query result by generating 1,000 results average_age_df_without_youngest_age that average age is 42.93
This is the query result by generating 4,000 results average_age_df_without_youngest_age that average age is 43.14
```

Step 2. In each of the above 4 groups of 1,000 results, round each number to two decimal places, define a measure, and utilize it to validate that each of the last 3 groups of results and the original results is 0.5-indistinguishable.

(1) To round each number to two decimal places: I use round() function e.g. round(data, 2)

(2) Define a measure with Laplace Mechanism:

$A(D) = q(D) + Lap(S/\varepsilon)$ is ε-DP

Noise answer = A(D) = query result, true answer = q(D) = new_average_age

```
In [124]: '''
Define a measure with Laplace Mechanism
A(D) = q(D) + Lap(S/ε) is ε-DP
output = A(D), q(D) = new_average_age
'''
def laplace_mechanism(new_average_age, original_average_age, epsilon):
    s = abs(new_average_age - original_average_age)
    beta = s/epsilon
    noise = np.random.laplace(loc=0.0, scale=beta)
    output = new_average_age + noise
    output = round(output, 2)
    return output

'''
```

(3) Here is my way to validate that each of the last 3 groups of results and the original results are 0.5-indistinguishable.

a) Calculate the average age of the original data set D and the data set D'.

b) Calculate the absolute difference between the two average ages.

c) If the absolute difference between the two average ages is less than or equal to 0.5, then we can say that the two data sets are 0.5-indistinguishable in terms of the average age. $|avg(D) - avg(D')| <= 0.5$

```
def cal_difference(original_ave_age, new_average_age):
    return abs(original_ave_age - new_average_age)
```

I will apply this function in the next step to validate that each of the last 3 groups of results and the original results are 0.5-indistinguishable.

(4) Utilize it to validate that each of the last 3 groups of results and the original results is 0.5-indistinguishable. And then calculate the difference and variance to figure out the distortion. Here I show two ways of distortion that I will explain in the next step. I applied the two ways to compare the true answer and the noise answer here.

```python
print('==================== dataset size = 1000, epsilon = 0.5 ========================')
# dataset size = 1000, noise answer = A(D) of the dataset that removing a record with the oldest age
# noise answer = A_D_1_e05

epsilon05 = 0.5

A_D_1_e05 = average_age_df_without_a_oldest_age + laplace_mechanism(average_age_df_without_a_oldest_age, df_average_age_1000, ep
A_D_1_e05 = round(A_D_1_e05, 2)
print('* Noise answer == A_D_1_e05 ==', A_D_1_e05)

# The difference between true answer and noise answer that removing a record with the oldest age
distortion_A_D_1_e05 = cal_distortion(df_average_age_1000, A_D_1_e05)
distortion_A_D_1_e05 = round(distortion_A_D_1_e05, 2)
print('* The difference between true answer and noise answer == distortion_A_D_1_e05 ==', distortion_A_D_1_e05)

# variance of the dataset that removing a record with the oldest age
variance_A_D_1_e05 = cal_variance(df_average_age_1000, A_D_1_e05, epsilon05)
variance_A_D_1_e05 = round(variance_A_D_1_e05, 2)
print('* The variance between true answer and noise answer == variance_A_D_1_e05 ==', variance_A_D_1_e05)
print('-----------------------------------------------------------------------')
```

```python
# dataset size = 1000, noise answer = A(D) of the dataset that removing any record with age = 26
A_D_2_e05 = average_age_df_without_26 + laplace_mechanism(average_age_df_without_26, df_average_age_1000, epsilon05)
A_D_2_e05 = round(A_D_2_e05, 2)
print('* Noise answer == A_D_2_e05 ==', A_D_2_e05)

# The difference between true answer and noise answer that removing any record with age = 26
distortion_A_D_2_e05 = cal_distortion(df_average_age_1000, A_D_2_e05)
distortion_A_D_2_e05 = round(distortion_A_D_2_e05, 2)
print('* The difference between true answer and noise answer == distortion_A_D_2_e05 ==', distortion_A_D_2_e05)

# variance of the dataset that removing any record with age = 26
variance_A_D_2_e05 = cal_variance(df_average_age_1000, A_D_2_e05, epsilon05)
variance_A_D_2_e05 = round(variance_A_D_2_e05, 2)
print('* The variance between true answer and noise answer == variance_A_D_1_e05 ==', variance_A_D_2_e05)
print('-----------------------------------------------------------------------')
```

```python
# dataset size = 1000, noise answer = A(D) of the dataset that removing any record with the youngest age
A_D_3_e05 = average_age_df_without_youngest_age + laplace_mechanism(average_age_df_without_youngest_age, df_average_age_1000, eps
A_D_3_e05 = round(A_D_3_e05, 2)
print('* Noise answer == A_D_3_e05 == ', A_D_3_e05)

# The difference between true answer and noise answer that removing any record with the youngest age
distortion_A_D_3_e05  = cal_distortion(df_average_age_1000, A_D_3_e05)
distortion_A_D_3_e05 = round(distortion_A_D_3_e05, 2)
print('* The difference between true answer and noise answer == distortion_A_D_3_e05 ==', distortion_A_D_3_e05)

# variance of the dataset that removing any record with the youngest age
variance_A_D_3_e05 = cal_variance(df_average_age_1000, A_D_3_e05, epsilon05)
variance_A_D_3_e05 = round(variance_A_D_3_e05, 2)
print('* The variance between true answer and noise answer == variance_A_D_1_e05 ==', variance_A_D_3_e05)
```

Output and my conclusion based above code:

```
==================== dataset size = 1000, epsilon = 0.5 ========================
* Noise answer == A_D_1_e05 == 85.25
* The difference between true answer and noise answer == distortion_A_D_1_e05 == 42.16
* The variance between true answer and noise answer == variance_A_D_1_e05 == 14219.72
-----------------------------------------------------------------------
* Noise answer == A_D_2_e05 == 85.89
* The difference between true answer and noise answer == distortion_A_D_2_e05 == 42.8
* The variance between true answer and noise answer == variance_A_D_1_e05 == 14654.72
-----------------------------------------------------------------------
* Noise answer == A_D_3_e05 ==  85.76
* The difference between true answer and noise answer == distortion_A_D_3_e05 == 42.67
* The variance between true answer and noise answer == variance A D 1 e05 == 14565.83
```

We see |avg(D) - avg(D')| > 0.5.

The distinguish among the differences of the last 3 groups are:

|42.16 – 42.8| = 0.64

|42.16 – 42.67| = 0.51

|42.8 – 42.67| = 0.13

Step 3 Repeat all the above for ϵ = 1.0, utilize the above measure to validate that each of the last 3 groups of results and the original results are 1.0-indistinguishable.

The only change here is epsilon = 1. The code design is similar to the above that I applied when epsilon = 0.5. For save some space I would not paste all code here.

Part of the code:

```
# size=1000, noise answer = A(D) of the dataset that removing any record with the youngest age
A_D_3_e1 = average_age_df_without_youngest_age + laplace_mechanism(average_age_df_without_youngest_age, df_average_age_1000, eps:
A_D_3_e1 = round(A_D_3_e1)
print('* Noise answer == A_D_3_e1 ==', A_D_3_e1)

# The difference between true answer and noise answer that removing any record with the youngest age
distortion_A_D_3_e1 = cal_distortion(df_average_age_1000, A_D_3_e1)
distortion_A_D_3_e1 = round(distortion_A_D_3_e1, 2)
print('* The difference between true answer and noise answer == distortion_A_D_3_e1 ==', distortion_A_D_3_e1)

# variance of the dataset that removing any record with the youngest age
variance_A_D_3_e1 = cal_variance(df_average_age_1000, A_D_3_e1, epsilon_1)
variance_A_D_3_e1 = round(variance_A_D_3_e1, 2)
print('* The variance between true answer and noise answer == variance_A_D_1_e1 ==', variance_A_D_3_e1)
print('-----------------------------------------------------------------------')
```

Output and my thoughts:

```
==================== dataset size = 1000, epsilon = 1 =========================
* Noise answer == A_D_1_e1 == 85.85
* The difference between true answer and noise answer == distortion_A_D_1_e1 == 42.76
* The variance between true answer and noise answer == variance_A_D_1_e1 == 3656.84
-----------------------------------------------------------------------
* Noise answer == A_D_2_e1 == 86.08
* The difference between true answer and noise answer == distortion_A_D_2_e1 == 42.99
* The variance between true answer and noise answer == variance_A_D_1_e1 == 3696.28
-----------------------------------------------------------------------
* Noise answer == A_D_3_e1 == 86
* The difference between true answer and noise answer == distortion_A_D_3_e1 == 42.91
* The variance between true answer and noise answer == variance_A_D_1_e1 == 3682.54
```

We see |avg(D) - avg(D')| > 1.

The distinguish among the differences of the last 3 groups are:

|42.76 – 42.99| = 0.23

|42.76 – 42.91| = 0.15

|42.99 – 42.91| = 0.08

Step 4 Define another measure and utilize it to justify that the distortion of the 4,000 results for ϵ = 1.0 is less than that of ϵ = 0.5.

There are two ways we can compare the distortion

way 1: calculate the difference between the true answer and the noise answer

way 2: calculate the variance between the true answer and the noise answer

```python
'''
There are two ways we can compare the distortion
way 1: calculate the difference between true answer and noise answer
way 2: calculate the variance between true answer and noise answer
'''
# way 1
def cal_distortion(original_ave_age, new_average_age):
    return abs(original_ave_age - new_average_age)

# way 2
def cal_variance(original_ave_age, new_average_age, epsilon):
    '''Error: E(true answer - noisy answer)2 = Var(Lap(S(q)/e)) = 2*S(q))2/e2'''
    variance = 2*(original_ave_age - new_average_age)**2/epsilon**2
    return variance
```

If we want to compare the distortion, we firstly need to get the query results based on 4000 results with two epsilon, and then apply the two ways to see distortion. Since the only difference here is epsilon. I paste part of the code here which is when the epsilon is 0.5.

```python
print('==================== dataset size = 4000, epsilon = 0.5 =========================')

# dataset size = 4000, noise answer = A(D) of the dataset that removing a record with the oldest age
# noise answer = A_D_1_e05_four
A_D_1_e05_four = mean_age_df_without_a_oldest_age + laplace_mechanism(mean_age_df_without_a_oldest_age, df_mean_age_4000 , epsil
A_D_1_e05_four = round(A_D_1_e05_four, 2)
print('* Noise answer == A_D_1_e05_four ==', A_D_1_e05_four)

# The difference between true answer and noise answer that removing a record with the oldest age
distortion_A_D_1_e05_four = cal_distortion(df_mean_age_4000, A_D_1_e05_four)
distortion_A_D_1_e05_four = round(distortion_A_D_1_e05_four, 2)
print('* The difference between true answer and noise answer == distortion_A_D_1_e05_four ==', distortion_A_D_1_e05_four)

# variance of the dataset that removing a record with the oldest age
variance_A_D_1_e05_four = cal_variance(df_mean_age_4000, A_D_1_e05_four, epsilon05)
variance_A_D_1_e05_four = round(variance_A_D_1_e05_four, 2)
print('* The variance between true answer and noise answer == variance_A_D_1_e05_four ==', variance_A_D_1_e05_four)
print('---------------------------------------------------------------------')
```

```python
# dataset size = 4000, noise answer = A(D) of the dataset that removing any record with age = 26
A_D_2_e05_four = mean_age_df_without_26 + laplace_mechanism(mean_age_df_without_26, df_mean_age_4000 , epsilon05)
A_D_2_e05_four = round(A_D_2_e05_four, 2)
print('* Noise answer == A_D_2_e05_four ==', A_D_2_e05_four)

# The difference between true answer and noise answer that removing any record with age = 26
distortion_A_D_2_e05_four = cal_distortion(df_mean_age_4000, A_D_2_e05_four)
distortion_A_D_2_e05_four = round(distortion_A_D_2_e05_four, 2)
print('* The difference between true answer and noise answer == distortion_A_D_2_e05_four ==', distortion_A_D_2_e05_four)

# variance of the dataset that removing any record with age = 26
variance_A_D_2_e05_four = cal_variance(df_mean_age_4000, A_D_2_e05_four, epsilon05)
variance_A_D_2_e05_four = round(variance_A_D_2_e05_four, 2)
print('* The variance between true answer and noise answer == variance_A_D_2_e05_four ==', variance_A_D_2_e05_four)
print('-----------------------------------------------------------------------')
```

```python
# dataset size = 4000, noise answer = A(D) of the dataset that removing any record with the youngest age
A_D_3_e05_four = mean_age_df_without_youngest_age + laplace_mechanism(mean_age_df_without_youngest_age, df_mean_age_4000, epsilor
A_D_3_e05_four = round(A_D_3_e05_four, 2)
print('* Noise answer == A_D_3_e05_four ==', A_D_3_e05_four)

# The difference between true answer and noise answer that removing any record with the youngest age
distortion_A_D_3_e05_four = cal_distortion(df_mean_age_4000, A_D_3_e05_four)
distortion_A_D_3_e05_four = round(distortion_A_D_3_e05_four, 2)
print('* The difference between true answer and noise answer == distortion_A_D_3_e05_four ==', distortion_A_D_3_e05_four)

# variance of the dataset that removing any record with the youngest age
variance_A_D_3_e05_four = cal_variance(df_mean_age_4000, A_D_3_e05_four, epsilon)
variance_A_D_3_e05_four = round(variance_A_D_3_e05_four, 2)
print('* The variance between true answer and noise answer == variance_A_D_3_e05_four ==', variance_A_D_3_e05_four)
```

Output:

```
==================== dataset size = 4000, epsilon = 0.5 ==========================
* Noise answer == A_D_1_e05_four == 86.12
* The difference between true answer and noise answer == distortion_A_D_1_e05_four == 43.25
* The variance between true answer and noise answer == variance_A_D_1_e05_four == 14964.5
------------------------------------------------------------------------
* Noise answer == A_D_2_e05_four == 85.93
* The difference between true answer and noise answer == distortion_A_D_2_e05_four == 43.06
* The variance between true answer and noise answer == variance_A_D_2_e05_four == 14833.31
------------------------------------------------------------------------
* Noise answer == A_D_3_e05_four == 85.58
* The difference between true answer and noise answer == distortion_A_D_3_e05_four == 42.71
* The variance between true answer and noise answer == variance_A_D_3_e05_four == 14593.15


==================== dataset size = 4000, epsilon = 1 ==========================
* Noise answer == A_D_1_e1_four == 84.86
* The difference between true answer and noise answer == distortion_A_D_1_e05_four == 41.99
* The variance between true answer and noise answer == variance_A_D_1_e05_four == 3526.32
------------------------------------------------------------------------
* Noise answer == A_D_2_e1_four == 86.67
* The difference between true answer and noise answer == distortion_A_D_2_e1_four == 43.8
* The variance between true answer and noise answer == variance_A_D_2_e1_four == 3836.88
------------------------------------------------------------------------
* Noise answer == A_D_3_e1_four == 86.03
* The difference between true answer and noise answer == distortion_A_D_3_e1_four == 43.16
* The variance between true answer and noise answer == variance A D 3 e1 four == 3725.57
```

Compare distortion between epsilon = 0.5 and epsilon =1

Way 1 Compare the difference

```python
# When e = 0.5, dataset size = 4000, The differences between true answers and noise answers are:
arr_distortion_e05_four = [distortion_A_D_1_e05_four, distortion_A_D_2_e05_four, distortion_A_D_3_e05_four]
print('When e = 0.5, dataset size = 4000, The distortion values are:', )
print('[distortion_A_D_1_e05_four, distortion_A_D_2_e05_four, distortion_A_D_3_e05_four] =', arr_distortion_e05_four, '\n')

# When e = 1, dataset size = 4000, The differences between true answers and noise answers are:
arr_distortion_e1_four = [distortion_A_D_1_e1_four, distortion_A_D_2_e1_four, distortion_A_D_3_e1_four]
print('When e = 1, dataset size = 4000, The distortion values are:', )
print('[distortion_A_D_1_e1_four, distortion_A_D_2_e1_four, distortion_A_D_3_e1_four] =', arr_distortion_e1_four, '\n')

# Let's check: when epsilon increases, how it influence the difference between true answer and noise anser
diff_btw_truth_lie = [arr_distortion_e05_four[i] - arr_distortion_e1_four[i] for i in range(min(len(arr_distortion_e05_four), ler
print('The difference between true answer and noise anser ==', diff_btw_truth_lie )

'''
```

Output:

```
When e = 0.5, dataset size = 4000, The distortion values are:
[distortion_A_D_1_e05_four, distortion_A_D_2_e05_four, distortion_A_D_3_e05_four] = [43.25, 43.06, 42.71]

When e = 1, dataset size = 4000, The distortion values are:
[distortion_A_D_1_e1_four, distortion_A_D_2_e1_four, distortion_A_D_3_e1_four] = [41.99, 43.8, 43.16]

The difference between true answer and noise anser == [1.2599999999999998, -0.7399999999999949, -0.44999999999999574]
```

We see the difference when e=0.5 and e=1 is not that huge.

Way 2 compare the variance

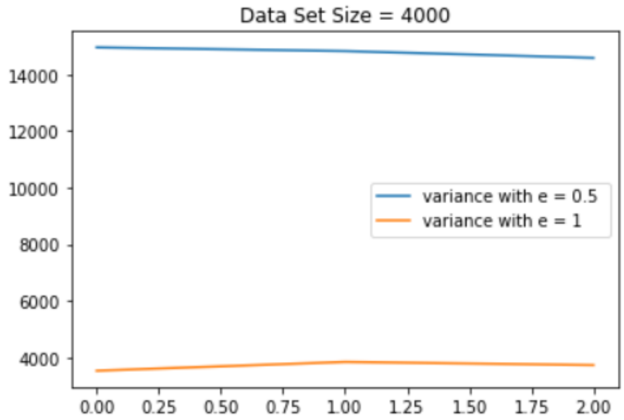
```
# When e = 0.5, dataset size = 4000, The variance values are:
print('When e = 0.5, dataset size = 4000, The variance values are:', )
varr_e05_four = [variance_A_D_1_e05_four, variance_A_D_2_e05_four, variance_A_D_3_e05_four]
print('[variance_A_D_1_e05_four, variance_A_D_2_e05_four, variance_A_D_3_e05_four] =', varr_e05_four, '\n')

# When e = 1, dataset size = 4000, The variance values are:
print('When e = 1, dataset size = 4000, The variance values are:', )
varr_e1_four = [variance_A_D_1_e1_four, variance_A_D_2_e1_four, variance_A_D_3_e1_four]
print('[variance_A_D_1_e1_four, variance_A_D_2_e1_four, variance_A_D_3_e1_four] =', varr_e1_four, '\n')
```

```
When e = 0.5, dataset size = 4000, The variance values are:
[variance_A_D_1_e05_four, variance_A_D_2_e05_four, variance_A_D_3_e05_four] = [14964.5, 14833.31, 14593.15]

When e = 1, dataset size = 4000, The variance values are:
[variance_A_D_1_e1_four, variance_A_D_2_e1_four, variance_A_D_3_e1_four] = [3526.32, 3836.88, 3725.57]
```

```
plt.plot(varr_e05_four, label='variance with e = 0.5 ')
plt.plot(varr_e1_four, label='variance with e = 1')
plt.title('Data Set Size = 4000')
plt.legend()
plt.show()
```



According the plot above, we clearly see that: as epsilon increases, variance decreases. The variance of the Laplace mechanism is related to the amount of noise added to the output. When epsilon is larger, more noise is added to the output, which reduces the variance of the mechanism such that Error: E(true answer – noisy answer)^2 decreases.. Hence, the distortion of the 4,000 results for $\epsilon = 1.0$ is less than that of $\epsilon = 0.5$.

2 Exponential Mechanism

2.1 Introduction to Data sets

1. Initial datasets after data processing and **generating data** sets

df01: removing a record with the most frequent "Education"
df02: removing any record with the second most frequent "Education"
df03: removing any record with the least frequent "Education"

Here is the major code:

generate 1,000 results, 4,000 results separately for the query over the original dataset.
df00_1k: original data set

```
# Generate required results for the query over the original dataset.

'''
================================== 1 ===============================================
note: the description does not mention how we can generate the required results. There are at least two ways:
way 1. get the required results by head()
way 2. get the required results by sample()
Here, I choose to generate data by way2
'''

# Generate 1,000 results for the query(the most frequent “Education” result) over the original dataset
df00_1k = df.sample(n=1000, random_state=40)
print('Length of the original data set with 1000 results, df00_1k size ==', len(df00_1k))

# Generate 4,000 results for the query(the most frequent “Education” result) over the original dataset
df00_4k = df.sample(n=4000, random_state=40)
print('Length of the original data set with 1000 results, df00_4k size ==', len(df00_4k))
```

```
Length of the original data set with 1000 results, df00_1k size == 1000
Length of the original data set with 1000 results, df00_4k size == 4000
```

Here are data processing for 3 other data sets code.

Note: the description does not mention whether or not we do data processing based on the original data set, by default, I do data processing based on the original data set

```
******************************** Data Proccessing ********************************
df01: removing a record with the most frequent “Education”

The condition means we need to remove one record with the most frequent “Education”. Since we clearly know that the most
frequent “Education” is with many records, then we can just pick one record with the most frequent “Education”.

Simply, I get the first index corresponding to the most most frequent “Education” level, then remove a record with the most
frequent “Education” as the question description requires.
'''

# ************** df01: Removing a record with the most frequent “Education”  ************
most_fre_level = df['education'].value_counts().idxmax()
most_fre_num = df['education'].value_counts().max()
print('* The value corresponding to most frequency in the original dataset is', most_fre_level)
print('* The amount corresponding to most frequency in the original dataset is', most_fre_num)

# get the first index corresponding to the most frequent education level
first_index = df.index[df['education'] == most_fre_level].min()
first_index
print('* The first index corresponding to the most most frequent “Education” level == ', first_index)
print('* Check the length of the original data frame: the length should be 32561 ==', len(df))

# remove a record with the most frequent “Education”
df01 = df.drop(index = first_index)
print('* After remove a record with the most frequent “Education”, check the length of df01: the length should be 32560 ==', len(
# Now, I get the data set that removing a record with the most frequent “Education”
print('* Now, I get the data set that removing a record with the most frequent “Education which is df01”')
```

Output:

```
* The value corresponding to most frequency in the original dataset is  HS-grad
* The amount corresponding to most frequency in the original dataset is 10501
* The first index corresponding to the most most frequent “Education” level ==  2
* Check the length of the original data frame: the length should be 32561 == 32561
* After remove a record with the most frequent “Education”, check the length of df01: the length should be 32560 == 32560
* Now, I get the data set that removing a record with the most frequent “Education which is df01”
```

df02: removing any record with the second most frequent "Education"

```
# ************** df02: removing any record with the second most frequent “Education”  ************

# get the levol of the second most frequent “Education” in the original data set
sec_most_fre_level =  df['education'].value_counts().index[1]
print('* The second most frequent Education level in the original dataset is', sec_most_fre_level)

df02 = df[df['education'] != sec_most_fre_level]
print('* Now, I get the data set that removing any record with the second most frequent “Education” which is df02”')
```

```
* The second most frequent Education level in the original dataset is  Some-college
* Now, I get the data set that removing any record with the second most frequent “Education” which is df02”
```

df03: removing any record with the least frequent “Education”

```
# ************** df03: removing any record with the least frequent “Education”  ************

# find the least frequent education level
least_frequent_level = df['education'].value_counts().index[-1]
print('* The least frequent education level is', least_frequent_level)

df03 = df[df['education'] != least_frequent_level]
print('* Now, I get the data set that removing any record with the least frequent “Education” which is df03”')
```

```
* The least frequent education level is  Preschool
* Now, I get the data set that removing any record with the least frequent “Education” which is df03”
```

2. Generate data with the required conditions

2.1 Generate data with date size = 1000 results for the four data sets

df01_1k: removing a record with the most frequent “Education”

df02_1k: removing any record with the second most frequent “Education”

df03_1k: removing any record with the least frequent “Education”

```
In [130]: # generate 1,000 results for the query over each of three datasets
df01_1k = df01.sample(n=1000, random_state=40)
print('Length of df01 ==', len(df01_1k))

df02_1k = df02.sample(n=1000, random_state=40)
print('Length of df02 ==', len(df02_1k))

df03_1k = df03.sample(n=1000, random_state=40)
print('Length of df03 ==', len(df03_1k))
```

```
Length of df01 == 1000
Length of df02 == 1000
Length of df03 == 1000
```

2.2 Generate data with date size = 4000 results for the four data sets

df00_4k: original data set

df01_4k: removing a record with the most frequent “Education”

df02_4k: removing any record with the second most frequent “Education”

df03_4k: removing any record with the least frequent “Education”

Here is the major code:

```
In [131]: # generate 1,000 results for the query over each of three datasets
df01_4k = df01.sample(n=4000, random_state=40)
print('Length of df01 ==', len(df01_4k))

df02_4k = df02.sample(n=4000, random_state=40)
print('Length of df02 ==', len(df02_4k))

df03_4k = df03.sample(n=4000, random_state=40)
print('Length of df03 ==', len(df03_4k))
```

```
Length of df01 == 4000
Length of df02 == 4000
Length of df03 == 4000
```

2.2 Implement of Exponential Mechanism

Note: the code that was implemented on 1000 results or e = 0.5 is similar to 4000 results or e =1. Hence, I just introduce my solutions with part code which is more related to 1000 results or e = 0.5.

Exponential Mechanism is related to score/utility function w, here is the sensitivity of w: $\Delta w = \max_{O \& D, D'} |w(D,O) - w(D',O)|$.

The exponential mechanism, which allows selecting the "best" element from a set while preserving differential privacy. The analyst defines which element is the "best" by specifying a scoring function that outputs a score for each element in the set, and also defines the set of things to pick from.

```
def score_1k(data, most_freq_level):
    '''
        data: values of a level, here means: values of df['education']
        option: item of data
    '''
    return data.value_counts()[most_freq_level]/1000

def score_4k(data, most_freq_level):
    '''
        data: values of a level, here means: values of df['education']
        option: item of data
    '''
    return data.value_counts()[most_freq_level]/4000
```

Check the score on 4 data sets with different data size (no noise added)

```
# ****** score the most frequent level on each data set with size 1000 ******
score_df00_1k = score_1k(df00_1k['education'], most_fre_level_df00_1k)
score_df01_1k = score_1k(df01_1k['education'], most_fre_level_df01_1k)
score_df02_1k = score_1k(df02_1k['education'], most_fre_level_df02_1k)
score_df03_1k = score_1k(df03_1k['education'], most_fre_level_df03_1k)
# Get the score of most frequen level of each data sets with size 1000
scores_list_4datasets = [score_df00_1k, score_df01_1k, score_df02_1k, score_df03_1k]
print('* With size 1000, get the score of most frequen level of each data sets', most_fre_levels_list_1k, '==', scores_list_4data

# ****** score the most frequent level on each data set with size 4000 ******
score_df00_4k = score_4k(df00_4k['education'], most_fre_level_df00_4k)
score_df01_4k = score_4k(df01_4k['education'], most_fre_level_df01_4k)
score_df02_4k = score_4k(df02_4k['education'], most_fre_level_df02_4k)
score_df03_4k = score_4k(df03_4k['education'], most_fre_level_df03_4k)
# Get the score of most frequen level of each data sets with size 4000
scores_list_4datasets = [score_df00_4k, score_df01_4k, score_df02_4k, score_df03_4k]
print('* With size 4000, get the score of most frequen level of each data sets', most_fre_levels_list_4k, '==', scores_list_4data
```

Output:

```
* With size 1000, get the score of most frequen level of each data sets [' HS-grad', ' HS-grad', ' HS-grad', ' HS-grad'] == [0.
315, 0.362, 0.427, 0.323]

* With size 4000, get the score of most frequen level of each data sets [' HS-grad', ' HS-grad', ' HS-grad', ' HS-grad'] == [0.
31975, 0.32675, 0.43375, 0.3185]
```

We see in these data sets with different data size and without noise, the most frequent level are all HS-grad.

define a measure: based on Exponential Mechanism we learned.

```
In [19]: def exponential(x, R, u, new_most_fre_num, old_most_fre_num, epsilon):
    '''
    x: a value representing the sensitive attribute being released (in this case, it appears to be the 'education' column of a
    R: a list of potential values that x can be replaced with in the released data (the 'options'/levels variable)
    u: a scoring function that takes in x and one of the elements in R and returns a score for how well that replacement value
    epsilon: a parameter that controls the privacy guarantee provided by the mechanism (a larger value of epsilon provides less
    '''
    sensitivity = abs(new_most_fre_num - old_most_fre_num)
    # Calculate the score for each element of R
    scores = [u(x, r) for r in R]

    # Calculate the probability for each element, based on its score
    probabilities = [np.exp(epsilon * score / (2 * sensitivity)) for score in scores]

    # Normalize the probabilties so they sum to 1
    probabilities = probabilities / np.linalg.norm(probabilities, ord=1)

    # Choose an element from R based on the probabilities
    return np.random.choice(R, 1, p=probabilities)[0]
```

Utilize it to validate that each of the last 3 groups of results and the original results are 0.5-indistinguishable. Repeat all the above for ϵ = 1.0, and utilize the above measure to validate that each of the last 3 groups of results and the original results are 1.0-indistinguishable (on different data size and different e, some code is similar, I just paste the key implement of my ideas):

Here, I apply the Exponential Mechanism function I designed above and get the noise answers, in the next step, I will apply the score function to get the most frequency and its corresponding levels.

```
epsilon05 = 0.5
epsilon_1 = 1
print('========== Date size = 1000, the noise answer we get by applying Exponential Mechanism ============', '\n')
noise_answer_df01_1k_e05 = exponential(df01_1k['education'], options_df01_1k, score_1k, most_fre_num_df01_1k, most_fre_num_df00_1
noise_answer_df02_1k_e05 = exponential(df02_1k['education'], options_df02_1k, score_1k, most_fre_num_df02_1k, most_fre_num_df00_1
noise_answer_df03_1k_e05 = exponential(df03_1k['education'], options_df03_1k, score_1k, most_fre_num_df03_1k, most_fre_num_df00_1
noise_answer_list_df_1k_e05 = [noise_answer_df01_1k_e05, noise_answer_df02_1k_e05, noise_answer_df03_1k_e05]
print('When epsilon = 0.5, data size = 1000, The noise answer of 3 data sets are:', noise_answer_list_df_1k_e05, '\n')


noise_answer_df01_1k_e1 = exponential(df01_1k['education'], options_df01_1k, score_1k, most_fre_num_df01_1k, most_fre_num_df00_1k
noise_answer_df02_1k_e1 = exponential(df02_1k['education'], options_df02_1k, score_1k, most_fre_num_df02_1k, most_fre_num_df00_1k
noise_answer_df03_1k_e1 = exponential(df03_1k['education'], options_df03_1k, score_1k, most_fre_num_df03_1k, most_fre_num_df00_1k
noise_answer_list_df_1k_e1 = [noise_answer_df01_1k_e1, noise_answer_df02_1k_e1, noise_answer_df03_1k_e1]
print('When epsilon = 1, data size = 1000, The noise answer of 3 data sets are:', noise_answer_list_df_1k_e1, '\n')
```

Output:

```
========== Date size = 1000, the noise answer we get by applying Exponential Mechanism ============

When epsilon = 0.5, data size = 1000, The noise answer of 3 data sets are: [' Masters', ' Prof-school', ' Assoc-voc']

When epsilon = 1, data size = 1000, The noise answer of 3 data sets are: [' Prof-school', ' Prof-school', ' Doctorate']

========== Date size = 4000, the noise answer we get by applying Exponential Mechanism ============

When epsilon = 0.5, data size = 1000, The noise answer of 3 data sets are: [' 10th', ' 1st-4th', ' 5th-6th']

When epsilon = 1, data size = 1000, The noise answer of 3 data sets are: [' 10th', ' Assoc-acdm', ' Bachelors']
```

We see the output of the most frequent levels changed after adding noise. If we publish these data, the attacker could not get some sensitive information from the query results and the query results are still in the data sets.

Get the most frequency and its corresponding levels after adding noise by the Exponential Mechanism. Here is the code about most frequency after applying score/utility function w on the data sets with different sizes and epsilon.

Note: since the code is so clear and the main function works in this step is exponential() I implemented and pd.Series() which can help me get the counts of each level. Hence, I would not explain much for the clear code and corresponding idea.

```
print("============ Data size = 1000, e = 0.5 ==============")

r_df01_1k_e05 = [exponential(df01_1k['education'], options_df01_1k, score_1k,
              most_fre_num_df01_1k, most_fre_num_df00_1k, epsilon05) for i in range(data_size)]
r_df01_1k_e05_counts = pd.Series(r_df01_1k_e05).value_counts()
#print('* noise answers in df01_1k ==', r_df01_1k_e05_counts, '\n')
most_fre_level_noise_df01_1k_e05 = r_df01_1k_e05_counts.idxmax()
print("In df01_1k, the most frequency level with noise is:", most_fre_level_noise_df01_1k_e05)
most_fre_noise_df01_1k_e05 = r_df01_1k_e05_counts.max()
print("In df01_1k, the most frequency with noise is", most_fre_noise_df01_1k_e05, '\n')

r_df02_1k_e05 = [exponential(df02_1k['education'], options_df02_1k, score_1k,
              most_fre_num_df02_1k, most_fre_num_df00_1k, epsilon05) for i in range(data_size)]
r_df02_1k_e05_counts = pd.Series(r_df02_1k_e05).value_counts()
#print('* noise answers in df02_1k ==', r_df02_1k_e05_counts, '\n')
most_fre_level_noise_df02_1k_e05 = r_df02_1k_e05_counts.idxmax()
print("In df02_1k, the most frequency level with noise is:", most_fre_level_noise_df02_1k_e05)
most_fre_noise_df02_1k_e05 = r_df02_1k_e05_counts.max()
print("In df02_1k, the most frequency with noise is", most_fre_noise_df02_1k_e05, '\n')

r_df03_1k_e05 = [exponential(df03_1k['education'], options_df03_1k, score_1k,
              most_fre_num_df03_1k, most_fre_num_df00_1k, epsilon05) for i in range(data_size)]
r_df03_1k_e05_counts = pd.Series(r_df03_1k_e05).value_counts()
#print('* noise answers in df03_1k ==', r_df03_1k_e05_counts, '\n')
most_fre_level_noise_df03_1k_e05 = r_df03_1k_e05_counts.idxmax()
print("In df03_1k, the most frequency level with noise is:", most_fre_level_noise_df03_1k_e05)
most_fre_noise_df03_1k_e05 = r_df03_1k_e05_counts.max()
print("In df03_1k, the most frequency with noise is", most_fre_noise_df03_1k_e05, '\n')
```

Output:

```
============ Data size = 1000, e = 0.5 ==============
In df01_1k, the most frequency level with noise is:  HS-grad
In df01_1k, the most frequency with noise is 77

In df02_1k, the most frequency level with noise is:  Masters
In df02_1k, the most frequency with noise is 83

In df03_1k, the most frequency level with noise is:  10th
In df03_1k, the most frequency with noise is 78

============ Data size = 1000, e = 1 ==============
In df01_1k, the most frequency level with noise is:  11th
In df01_1k, the most frequency with noise is 73

In df02_1k, the most frequency level with noise is:  9th
In df02_1k, the most frequency with noise is 82

In df03_1k, the most frequency level with noise is:  7th-8th
In df03_1k, the most frequency with noise is 72
```

```
============ Data size = 4000, e = 0.5 ==============
In df01_4k, the most frequency level with noise is:  11th
In df01_4k, the most frequency with noise is 76

In df02_4k, the most frequency level with noise is:  Prof-school
In df02_4k, the most frequency with noise is 88

In df03_4k, the most frequency level with noise is:  Assoc-voc
In df03_4k, the most frequency with noise is 85

============ Data size = 4000, e = 1 ==============
In df01_4k, the most frequency level with noise is:  Masters
In df01_4k, the most frequency with noise is 77

In df02_4k, the most frequency level with noise is:  5th-6th
In df02_4k, the most frequency with noise is 80

In df03_4k, the most frequency level with noise is:  Doctorate
In df03_4k, the most frequency with noise is 78
```

Since I get the true answers that are the most frequency in the data sets without applying the exponential mechanism and noise answers that are the most frequency after adding the noise by exponential mechanism. According to the code implement above, here are the trues answers and noises answers I got:

- The true answers with data size is 1000:

most_fre_levels_list_1k = [most_fre_level_df00_1k, most_fre_level_df01_1k, most_fre_level_df02_1k, most_fre_level_df03_1k]

most_fre_num_list_1k = [most_fre_num_df00_1k, most_fre_num_df01_1k, most_fre_num_df02_1k, most_fre_num_df03_1k]

- The true answers with data size is 4000:

most_fre_levels_list_4k = [most_fre_level_df00_4k, most_fre_level_df01_4k, most_fre_level_df02_4k, most_fre_level_df03_4k]

most_fre_num_list_4k = [most_fre_num_df00_4k, most_fre_num_df01_4k, most_fre_num_df02_4k, most_fre_num_df03_4k]

Note:

here I only list the noise answer that I need to calculate for the distortion. For the noise answer in data sets with 1000 size, please run my code and read the outputs for detailed information.

- The noise answers with data size are 4000, e = 0.5:

most_fre_levels_list_4k_noise_e05 = [most_fre_level_noise_df01_4k_e05, most_fre_level_noise_df02_4k_e05, most_fre_level_noise_df03_4k_e05]

most_fre_num_list_4k_noise_e05 =[most_fre_noise_df01_4k_e05, most_fre_noise_df02_4k_e05, most_fre_noise_df03_4k_e05]

- The noise answers with data size are 4000, e = 1:

most_fre_levels_list_4k_noise_e1 = [most_fre_level_noise_df01_4k_e1, most_fre_level_noise_df02_4k_e1, most_fre_level_noise_df03_4k_e1]

most_fre_num_list_4k_noise_e1 = [most_fre_noise_df01_4k_e1, most_fre_noise_df02_4k_e1, most_fre_noise_df03_4k_e1]

Define another measure and utilize it to justify that the distortion of the 4,000 results for ϵ = 1.0 is less than that of ϵ = 0.5.

There are two ways we can compare the distortion

way 1: calculate the difference between the true answer and the noise answer

way 2: calculate the variance between the true answer and the noise answer

```
'''
There are two ways we can compare the distortion
way 1: calculate the difference between true answer and noise answer
way 2: calculate the variance between true answer and noise answer
'''
# way 1
def cal_difference(true_answer, noise_answer):
    return abs(true_answer - true_answer)

# way 2
def cal_variance(true_answer, noise_answer, epsilon):
    '''Error: E(true answer - noisy answer)2 = Var(Lap(S(q)/e)) = 2*S(q))2/e2'''
    variance = 2*(true_answer - noise_answer)**2/(epsilon**2)
    return variance
```

We actually can just compare the variance in each data set with e =0.5 and the variance in each data set with e=1 to check the distortion.

```
In [126]: variance_4k_e05 = []
for true_answer, noise_answer in zip(most_fre_num_list_4k, most_fre_num_list_4k_noise_e05):
    variance = cal_variance(true_answer, noise_answer, epsilon05)
    variance_4k_e05.append(variance)

print(f"Variance with data size is 4000, e =0.5: {variance_4k_e05}")
```

```
Variance with data size is 4000, e =0.5: [11520000.0, 12005000.0, 21806408.0]
```

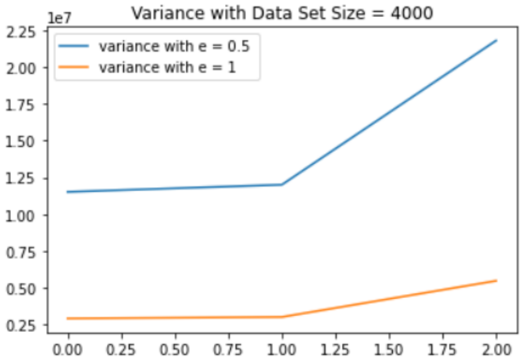
```
In [128]: variance_4k_e1 = []
for true_answer, noise_answer in zip(most_fre_num_list_4k, most_fre_num_list_4k_noise_e1):
    variance = cal_variance(true_answer, noise_answer, epsilon_1)
    variance_4k_e1.append(variance)

print(f"Variance with data size is 4000, e = 1: {variance_4k_e1}")
```

```
Variance with data size is 4000, e = 1: [2913698.0, 3011058.0, 5471432.0]
```

It would be easier to compare by plot:

```
In [129]: plt.plot(variance_4k_e05, label='variance with e = 0.5 ')
plt.plot(variance_4k_e1, label='variance with e = 1')
plt.title('Variance with Data Set Size = 4000')
plt.legend()
plt.show()
```



We see when the epsilon increases, the variance decreases. When the epsilon is larger, more noise is added to the output, which reduces the variance of the mechanism such that Error: E(true answer – noisy answer)^2 decreases. Hence, the distortion of the 4,000 results for ϵ = 1.0 is less than that of ϵ = 0.5.

Reference

Programming differential privacy. (n.d.). Retrieved 19 February 2023, from https://programming-dp.com/index.html