LECTURE 15

$$F(x) = f(x) + Lap\left(\frac{\Delta}{\epsilon}\right) \left[\text{where } \Delta = S = \text{sensitivity}\right]$$

$\downarrow = b$

The above eqn = laplace distribution centred at '0'

→ SENSITIVITY IS THE PROPERTY OF THE QUERY [THAT is for COUNT QUERY THE sensitivity was L, sensitivity (1)],

Q) Now let's imagine we have various queries and the privacy budget (ϵ) is fixed, so which query will involve more noise? Will it be the one with higher Δ or the one with lower Δ?

Ans FOR HIGH VALUE Δ MORE NOISE
FOR LOW VALUE Δ LESS NOISE

So the conclusion is if we have query with High sensitivity (Δ), the Laplace privacy scheme will inject more noise, and give us privacy at value of ϵ we have set.

query
[This is logical, because see if a ~~database~~ has a High sensitivity that means it is able to ~~mask~~/hide the records of the individual user and inorder to hide the record we need more noise, because high sensitivity also means that the presence or absence of that user in the database will make a huge difference to the database, so we have to ensure High privacy of that user. Hence we have ~~to~~ ~~hide~~ add more noise to hide the record]

Q) Now if for a query Δ is fixed and ε is varied, what will happen?

HIGH VALUE OF ε = LESS IS THE VALUE OF b = LESS IS NOISE

LOW VALUE OF ε = MORE IS THE VALUE OF b = MORE IS THE NOISE

[If we have to understand this logical if ε / privacy budget is High, that means we Have a Higher tolerance for privacy loss hence we can make do with a little bit of noise

Q EXAMPLE WAY OF UNDERSTANDING THE EQN

Suppose you do a count query on a Hospital database for certain database

$$F(x) = f(x) + lap(\Delta/\varepsilon)$$

so [o/p of count] s [the noise added] [this noise is properly generated and not randomly generated to ensure the DP bounds]

$$f(x) = \underline{55}$$

Q IN A CODING FORMAT HOW CAN LAPLACE MECHANISM BE WRITTEN

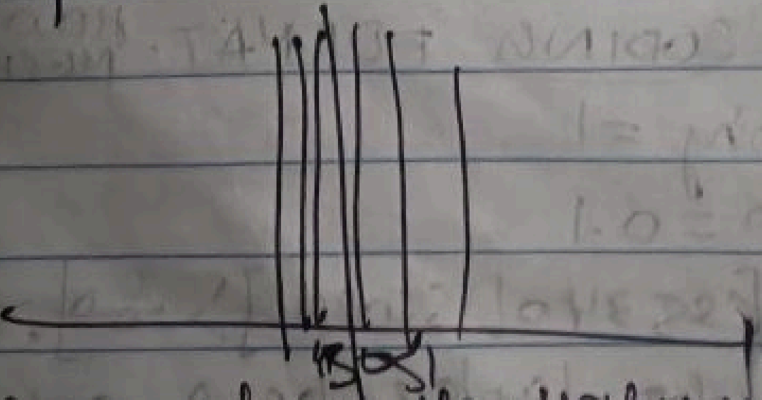
Ans

```
sensitivity = 1
epsilon = 0.1
cs5340[cs5340['score'] >= 80].shape[0] +
np.random.laplace(loc=0, scale= sensitivity/ε
```

In the code sensitivity is set to '1' as it is a counting query and ε is set to 0.1, ~~we thereby sets it to~~
which is set randomly here
we can set different values of ε depending on our privacy tolerance

Q In the previous example way of understanding the eqn, we got the actual ans total ans as 55 (were the actual ans was 50), so, How will the user know what is the correct ans for the query

Ans → We will simply run the query F(x) multiple time, on running it multiple times, we might get the answers as 48, 49, 50, 51, 52 — 55, so on so if we plot this in the graph



we know where the values are centered around.

On simply taking the average of the query results [that is 48, 49, 50·53] we get the ans = 50

Q BUT WHAT PROBLEM DOES THE ABOVE SOLN CAUSE ?

So suppose we query 'n' times in the above scenario, the privacy budget (ε) also increases by $\underline{\underline{n\varepsilon}}$, this means we are compromising <u>on our privacy</u>

Q SO HOW TO FIX THIS PROBLEM? [i.e the threat of the person doing repeated queries and finding out the ans]

Ans we will simply put a <u>limit</u> on how much a person can query. i.e if we have a privacy budget ε we can design a system ~~that~~ such that if a person does a query the privacy budget as ε_1 and he does query 'n' times

$$\boxed{n\varepsilon_1 = \varepsilon}$$

[The limit here is that the person cannot do more than n queries]

→ Another way of saying is that each query should be $\boxed{\frac{\varepsilon}{n}}$

PROPERTIES OF DP

(1) Sequential Composition:

$F_1(x)$ satisfies ε_1 – DP

$F_2(x)$ satisfies ε_2 – DP

The mechanism $G = (F_1(x), F_2(x))$ which releases both the results satisfies $(\varepsilon_1 + \varepsilon_2)$ DP

[Above eg can be related to query multiple times. i.e $F_1(x) = 50$ $F_2(x) = 52$

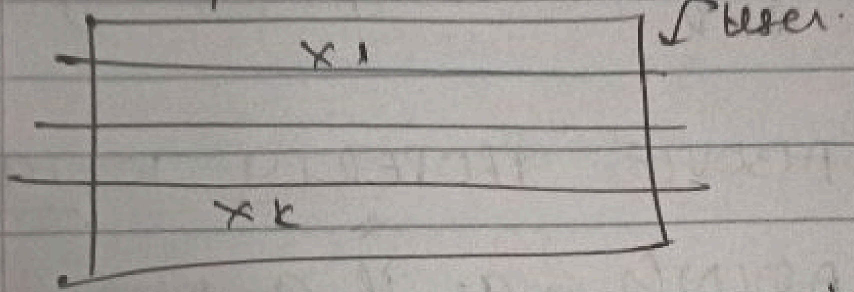
So using the ABOVE SCENARIO IT HELPS US IN KNOWING HOW TO DESIGN OUR SYSTEM [Eg in the previous case we need a limit on the no of queries]

(2) Parallel composition

If $f(x)$ satisfies ε-DP and we split a dataset x into k disjoint chunks $x_1 \cup x_2 \cdots \cup x_2 = x$

→ The mechanism which releases $F(x_1), F(x_2) \ldots F(x_k)$ satisfies ε-DP

[Eg: There is a dataset, and its rows are split into chunks



In ABOVE CASE, THE USER is not present in each chunk but is present in one of the chunk maybe $x_1 / x_2 \ldots / x_k$, so if the user does query on x_1 then query on x_2 -- till x_k then only a particular user is affected, as if only one query [Because user's data is only in one chunk] run on the dataset, this eliminates the ε' adding up [Hence ε does not change here]

THUS PARALLEL COMPOSITION IS BETTER THAN SEQUENTIAL COMPOSITION

(3) POST PROCESSING

→ we can't reverse DP through post processing.

→ If $\{ C(x)$ satisfies ε-DP

→ For any function h, $h(f(x))\}$ satisfied ε-DP

[THE ABOVE PROPERTY IS HIGHLY ENCOURAGING, as if a person wants to release His database, He can rest assured that his database will still have ε-DP privacy guarantee despite of any post processing done his data or any kind of manipula[tion] done to his data such as h]