


## Differential Privacy — Noise adding Mechanisms

Laplace Mechanism, Exponential Mechanism and Gaussian Mechanism

**Summary:** Sometimes knowing the basics is important! This the fifth blog post of “Differential Privacy Basics Series” covering the general noise adding mechanism used in Differential Privacy. For more posts like these on Differential Privacy follow [Shaistha Fathima](#) on twitter.

 Noise Adding Mechanisms		
Laplace Mechanism	Exponential Mechanism	Gaussian Mechanism
<ol style="list-style-type: none"><li>1. <math>\epsilon</math>-Differential Privacy</li><li>2. Computes <math>f(\text{query})</math> and perturbs EACH co-ordinate with noise drawn from Laplace Distribution.</li><li>3. The scale of the noise will be calibrated to the <math>\lceil \text{sensitivity of } f(\text{query}) \rceil / \epsilon</math>, where <math>\delta</math> is always equal to 0.</li><li>4. Works best with neumeric query with low sensitivity.</li></ol>	<ol style="list-style-type: none"><li>1. <math>\epsilon</math>-Differential Privacy.</li><li>2. Applicable to both neumeric and categorical functional query outputs.</li><li>3. Allows Selecting the "best" element from a set while preserving its differential privacy.</li><li>4. Releases only the identity of the element with MAX noisy score and not the score itself.</li></ol>	<ol style="list-style-type: none"><li>1. <math>(\epsilon, \delta)</math>-differential privacy.</li><li>2. Works almost exactly as Laplace Mechanism, given that <math>\delta</math> is also taken into account.</li><li>3. L1 sensitivity is required for Laplace Mechanism whereas for Gaussian, either L1 or L2 sesnitvity can work.</li><li>4. L2 sensitivity is lower than L1 sensitivity in value. Thus, allows addition of much less noise in comparison to Laplace Mechanism.</li></ol>

Quick recap.

### (1) Laplace Mechanism

Numeric queries, functions such as,

$$f : \mathbb{N}^{|\mathcal{X}|} \rightarrow \mathbb{R}^k;$$

are one of the most fundamental types of database queries. These queries map [databases](#) to  $k$  real numbers. One of the important parameters that will determine just how accurately we can answer such queries is their  $\ell_1$  sensitivity.

**Definition 3.1** ( $\ell_1$ -sensitivity). The  $\ell_1$ -sensitivity of a function  $f : \mathbb{N}^{|\mathcal{X}|} \rightarrow \mathbb{R}^k$  is:

$$\Delta f = \max_{\substack{x, y \in \mathbb{N}^{|\mathcal{X}|} \\ \|x - y\|_1 = 1}} \|f(x) - f(y)\|_1.$$

*The  $\ell_1$  sensitivity of a function gives an upper bound on how much we must perturb its output to preserve privacy*, i.e., The  $\ell_1$  sensitivity of a function  $f$  *captures the magnitude by which a single individual's data can change the function  $f$  in the worst case*, and therefore, intuitively, the uncertainty in the response that we must introduce in order to hide the participation of a single individual.

Artificial Intelligence Jobs

**Definition 3.2** (The Laplace Distribution). The Laplace Distribution (centered at 0) with scale  $b$  is the distribution with probability density function:

$$\text{Lap}(x|b) = \frac{1}{2b} \exp\left(-\frac{|x|}{b}\right).$$

*The Laplace distribution is a symmetric version of the exponential distribution. Adds noise from a symmetric continuous distribution to true answer.*

The **Laplace mechanism** will simply compute  $f$ , and perturb **each coordinate** with noise drawn from the Laplace distribution. *The scale of the noise will be calibrated to the  $[(\text{sensitivity of } f(\text{query}))/\epsilon]$ , where  $\delta$  is always equal to 0.*

Noise is scaled to  $1/\epsilon$ , that is, by adding noise drawn from  $\text{Lap}(1/\epsilon)$ . The expected distortion, or error, is  $1/\epsilon$ , independent of the size of the database.

The Laplace mechanism preserves  $(\epsilon, 0)$ -differential privacy or  $\epsilon$ -differentially private.

This corresponds to the intuition that the **more sensitive** the query, and the **stronger** the **desired guarantee**, the **more “noise” is needed** to achieve that guarantee.

**Definition 3.3** (The Laplace Mechanism). Given any function  $f : \mathbb{N}^{|\mathcal{X}|} \rightarrow \mathbb{R}^k$ , the Laplace mechanism is defined as:

$$\mathcal{M}_L(x, f(\cdot), \varepsilon) = f(x) + (Y_1, \dots, Y_k)$$

where  $Y_i$  are i.i.d. random variables drawn from  $\text{Lap}(\Delta f / \varepsilon)$ .

Another way to look at it is,

According to the Laplace mechanism, for a function  $f(x)$  which returns a number, the following definition of  $F(x)$  satisfies  $\epsilon$ -differential privacy:

$$F(x) = f(x) + \text{Lap}\left(\frac{s}{\epsilon}\right) \quad (1)$$

where  $s$  is the **sensitivity** of  $f$ , and  $\text{Lap}(S)$  denotes sampling from the Laplace distribution with center 0 and scale  $S$ .

**The sensitivity of a function  $f$  is the amount  $f$ 's output changes when its input changes by 1.**

**Example: Counting queries always have a sensitivity of 1:** if a query counts the number of rows in the [dataset](#) with a particular property, and then we modify exactly one row of the dataset, then the query's output can change by at most 1.

Thus we can achieve differential privacy for our example query by using the Laplace mechanism with sensitivity 1 and an  $\epsilon$  of our choosing. For now, let's pick  $\epsilon = 0.1$ . We can sample from the Laplace distribution using Numpy's `random.laplace`.

```
sensitivity = 1
epsilon = 0.1

adult[adult['Age'] >= 40].shape[0] + np.random.laplace(loc=0, scale=sensitivity/epsilon)

14238.147613610243
```

You can see the effect of the noise by running the proceeding cell multiple times. Each time, the output changes, but most of the time, the answer is close enough to the true answer (14,235) to be useful.

Considerable care must be taken when programming real-valued mechanisms, such as the Laplace mechanism, due to subtleties in the implementation of floating point numbers. Otherwise differential privacy can be destroyed, as outputs with non-zero probability on a database  $x$ , may, because of rounding, have zero probability on adjacent databases  $y$ . This is just one way in which the implementation of floating point requires scrutiny in the context of differential privacy, and it is not unique.

### **Drawbacks :**

- (i) It is only really good for low sensitivity queries. (usually with L1 sensitivity)
- (ii) Need large epsilon values (aka privacy budget) if you are going to fire off a bunch of queries. (Large epsilon values produce results that are less accurate in order to achieve the privacy guarantee).
- (iii) Provides solution to handle numeric queries, but cannot be applied to the non-numeric valued queries like “what is the most common nationality in this room”: Chinese/Indian/American...

### **Need for Exponential Mechanism when Laplace Mechanism already existed:**

- Laplace could be used for numeric queries only, while, exponential can be applied for numerical or categorical functional query output.
- Most of the differential privacy revolved around **real-valued functions** which have **relatively low sensitivity** to change in the [data](#) of a single individual and whose usefulness is not hampered by small additive perturbations. *A natural question is what happens in the situation when one wants to preserve more general sets of properties?* The exponential mechanism helps to extend the notion of differential privacy to address these issues.
- *(Laplace and Gaussian) are focused on numerical answers, and add noise directly to the answer itself.* What if we want to return a precise answer (i.e. no added noise), but still preserve differential privacy? One solution is the exponential mechanism, which allows **selecting the “best” element** from a set while preserving differential privacy.

### **Example:**

- The Laplace Mechanism gives a **general purpose way of adding noise** to satisfy differential privacy *assuming that computing  $f$  accurately is the best measure of what we want to extract from our data.*
- BUT, if **input:  $x$** = A database of training

**dataGoal/Output:  $y$** = A neural network that minimizes the training error on  $x$

- The neural network **returned** is defined by a series of weights.
- If we were to apply the Laplace Mechanism to this function, *Laplace noise would be added to the weights before returning the network.*
- However, *even small fluctuations* in weights in a neural network may *severely impact the performance* of that network.
- Therefore, the *returned network (with added noise)* will likely **behave very differently** than the *initial network found (before adding noise)* that minimized error, and thus would have a **unpredictably higher error** than the minimal error network we desired.
- Thus, this presents us with motivation to create another mechanism to satisfy Differential Privacy in such cases where the Laplace Mechanism fails.

## (2) Exponential Mechanism

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.

The mechanism provides differential privacy by approximately maximizing the score of the element it returns — in other words, to satisfy differential privacy, the exponential mechanism sometimes returns an element from the set which does not have the highest score.

The exponential mechanism satisfies  $\epsilon$ -differential privacy:

1. The analyst selects a set  $\mathcal{R}$  of possible outputs
2. The analyst specifies a scoring function  $u : \mathcal{D} \times \mathcal{R} \rightarrow \mathbb{R}$  with global sensitivity  $\Delta u$
3. The exponential mechanism outputs  $r \in \mathcal{R}$  with probability proportional to:

$$\exp\left(\frac{\epsilon u(x, r)}{2\Delta u}\right) \quad (1)$$

Rather than adding noise to the output of a function  $f$ , the exponential mechanism draws an output  $o$  from a probability distribution. Given a parameter  $\epsilon$ , an input  $x$ , and a utility function  $u$  with generalized sensitivity  $\Delta u$ , we draw an output  $o$  from the following distribution:

$$Pr[o] = e^{\frac{\epsilon u(x, o)}{2\Delta u}}$$

The biggest practical difference between the exponential mechanism and the Laplace mechanism is that the **output of the exponential mechanism is always a member of the set  $R$** . This is extremely useful when *selecting an item from a finite set, when a noisy answer would not make sense*.

**For example:**

We might want to **pick a date** for a big meeting, which uses each participant's personal calendar to maximize the number of participants without a conflict, while providing differential privacy for the calendars.

*Adding noise to a date doesn't make much sense*: it might turn a Friday into a Saturday, and increase the number of conflicts significantly. **The exponential mechanism is perfect for problems like this one: it selects a date without a noise.**

**The exponential mechanism is interesting for several reasons:**

- The **privacy cost** of the mechanism is *just epsilon*, regardless of the size of  $R$ .
- It theoretically works for **both finite and infinite** sets  $R$ , but it can be really challenging to build a practical implementation which samples from the appropriate probability distribution when  $R$  is infinite.
- It represents a "*fundamental mechanism*" of  $\epsilon$ -differential privacy: all other  $\epsilon$ -differentially private mechanisms can be defined in terms of the exponential mechanism with the appropriate definition of the scoring function  $u$ .

**Why is the exponential mechanism so much better?** Because *it releases less information*.

The exponential mechanism releases *only* the identity of the element with the maximum noisy score — *not* the score itself, or the scores of any other element.

**Drawbacks:**

- **The exponential mechanism is extremely general** — it's generally possible to redefine any  $\epsilon$ -differentially private mechanism in terms of a carefully chosen definition of the scoring function  $u$ . If we can analyze the sensitivity of this scoring function, then the proof of differential privacy comes for free.

- On the other hand, applying the general analysis of the exponential mechanism sometimes ***comes at the cost of looser bounds***, and mechanisms defined in terms of the exponential mechanism are ***often very difficult to implement***.
- The exponential mechanism is ***often used to prove theoretical lower bounds*** (by showing that a differentially private algorithm *exists*), but practical algorithms often replicate the same behavior using some other approach (such as noisy max approach using laplace mechanism).

### (3) Gaussian Mechanism

The Gaussian mechanism is an *alternative to the Laplace mechanism*, which adds Gaussian noise instead of Laplacian noise. The Gaussian mechanism does *not* satisfy pure  $\epsilon$ -differential privacy, but does *satisfy  $(\epsilon, \delta)$ -differential privacy*.

According to the Gaussian mechanism, for a function  $f(x)$  which returns a number, the following definition of  $F(x)$  satisfies  $(\epsilon, \delta)$ -differential privacy:

$$F(x) = f(x) + \mathcal{N}(\sigma^2) \quad (1)$$

$$\text{where } \sigma^2 = \frac{2s^2 \log(1.25 / \delta)}{\epsilon^2} \quad (2)$$

where  $s$  is the sensitivity of  $f$ , and  $\mathcal{N}(\sigma^2)$  denotes sampling from the Gaussian (normal) distribution with center 0 and variance  $\sigma^2$ . Note that here (and elsewhere in these notes),  $\log$  denotes the natural logarithm.

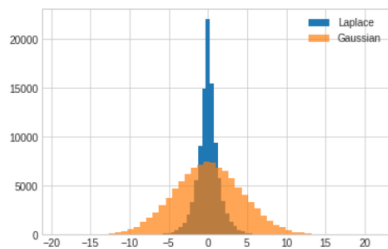
For real-valued functions

$$f : D \rightarrow \mathbb{R}$$

we can use the Gaussian mechanism in exactly the same way as we do the Laplace mechanism, and it's easy to compare what happens under both mechanisms for a given value of  $\epsilon$ .

**For example:**

```
%matplotlib inline
import matplotlib.pyplot as plt
plt.style.use('seaborn-whitegrid')
import pandas as pd
import numpy as np
epsilon = 1
delta = 10e-5
vals_laplace = [np.random.laplace(loc=0, scale=1/epsilon) for x in range(100000)]
sigma = np.sqrt(2 * np.log(1.25 / delta)) * 1 / epsilon
vals_gauss = [np.random.normal(loc=0, scale=sigma) for x in range(100000)]
plt.hist(vals_laplace, bins=50, label='Laplace')
plt.hist(vals_gauss, bins=50, alpha=.7, label='Gaussian');
plt.legend();
```



Here, we graph the empirical probability density function of the Laplace and Gaussian mechanisms for  $\epsilon = 1$ , with  $\delta = 10^{-5}$  for the Gaussian mechanism.

Compared to the Laplace mechanism, the *plot for the Gaussian mechanism looks*

*“squished.” Differentially private outputs which are far from the true answer are much more likely using the Gaussian mechanism than they are under the Laplace mechanism (which, by comparison, looks extremely “pointy”).*

**Gaussian mechanism also has *two major drawbacks* :**

- (i) It requires the use of the relaxed  $(\epsilon, \delta)$ -differential privacy definition,
- (ii) It's less accurate than the Laplace mechanism.

**So, Why would we want to use it?**

In the above example, we have only considered real-valued functions (i.e. the function's output is always a single real number).

Such functions are of the form

$$f : D \rightarrow \mathbb{R}$$

Both the Laplace and Gaussian mechanism, however, can be extended to *vector-valued* functions of the form

$$f : D \rightarrow \mathbb{R}^k$$



This return vectors of real numbers. *For example*, we can think of histograms as vector-valued functions, which return a vector whose elements consist of histogram bin counts.

Both the Laplace and Gaussian mechanisms can be extended to vector-valued functions. However, there's a **key difference** between these two extensions:

The **vector-valued Laplace mechanism** releases  $f(x) + (Y_1, \dots, Y_k)$ , where  $Y_i$  are drawn i.i.d. from the Laplace distribution with scale  $\frac{s}{\epsilon}$  and  $s$  is the *L1 sensitivity* of  $f$

The **vector-valued Gaussian mechanism** releases  $f(x) + (Y_1, \dots, Y_k)$ , where  $Y_i$  are drawn i.i.d. from the Gaussian distribution with  $\sigma^2 = \frac{2s^2 \log(1.25 / \delta)}{\epsilon^2}$  and  $s$  is the *L2 sensitivity* of  $f$

(i) The vector-valued **Laplace** mechanism **requires** the **use of L1 sensitivity**, while the vector-valued **Gaussian** mechanism allows the use of **either L1 or L2 sensitivity**.

(ii) This is a major strength of the Gaussian mechanism. For applications in which **L2 sensitivity is much lower than L1 sensitivity**, the Gaussian mechanism allows adding much less noise.

We have seen that Differential Privacy(DP) adds statistical noise to the [data](#) for better privacy, but, **How much noise should we add?** This depends on the below 4 factors:

- Sensitivity of the Query.
- Desired Epsilon.
- Desired delta.
- Type of Noise to be added. (most commonly used — Gaussian Noise or Laplacian Noise, among which Laplacian noise is easy to use)

In this blog post we will focus on the SENSITIVITY. To know about the Epsilon and Delta, please look at the (Part-3) Differential Privacy Definition of this series.

## Sensitivity

Sensitivity parametrizes the amount i.e., how much **noise** perturbation is required in the DP mechanism. To determine the sensitivity, **the maximum of possible change in the result** needs to be calculated.

**Generally sensitivity refers to the impact a change in the underlying data set can have on the result of the query.** Let  $x_A, x_B$  be any data set from all possible data set of  $X$  differing in *at most one element*. Then the sensitivity is calculated with the following equation:

$$\text{Sensitivity} = \max_{x_A, x_B \subseteq X} \|q(x_A) - q(x_B)\|_1$$

**where  $\|\cdot\|_1$  is the  $L_1$ -norm distance between data sets differing at most one element.**

The sensitivity of a query can be stated both in the global and local context:

- **Global sensitivity** refers to the consideration of **all possible data sets** differing in at most one element.
- Whereas, **local sensitivity** is the change in **one data set** with differing at most one element.

## Global Sensitivity

Global sensitivity refers to the **maximum difference of the output** a query function can result in **when one change is made to any data set**.

Global sensitivity determines the **magnitude of the noise** needed in order to meet the  $\epsilon$ -DP requirements.

Given a query function  $f$  that is operating on a dataset  $D$  and producing the maximum result difference for all data sets  $(D_1, D_2)$  with at most one different entry will be:

$$\Delta f_{GS} = \max_{D_1, D_2} \|f(D_1) - f(D_2)\|_1$$

where  $\| \cdot \|_1$  is the  $L_1$ -norm distance between datasets differing at most one element,  $\max$  is the maximum result of  $f(D_1) - f(D_2)$  for all datasets  $D_1, D_2$ .

**Note:** Global sensitivity is the maximum differences in output with consideration of all possible datasets and is therefore only **dependent on the query and not the dataset**.

This has wide consequences for the utility of certain queries. This can be demonstrated by a simple query that takes the **sum** of any dataset where every entry can in theory contribute any arbitrary amount. *The largest difference an output can result in from any query is infinite* due to the fact that there is no upper bound on any single entry, *thus the global sensitivity for the sum mechanism is infinite*.

queries. **Bounds** can be introduced into the query mechanism for the dataset, that **effectively limits the datasets ability to store values** greater than a predetermined threshold.

The dataset must continuously be modified in order to guarantee no value exceeds the threshold, the global sensitivity would not be infinite and only depend on the query function and the threshold. Thus, it can be said that **Global sensitivity is the minimum sensitivity needed for a query to cover all possible datasets**.

In the process of releasing data while using queries such as **count** or **sum** that has **low sensitivity work well with global sensitivity**. We could take the case of count query as an example which has  $GS(f) = 1$  that is smaller than the true answer. However, when it comes to queries like median, average the global sensitivity is much higher.

**Another way of looking at it is,**

For a function  $f : \mathcal{D} \rightarrow \mathbb{R}$  mapping datasets ( $\mathcal{D}$ ) to real numbers, the *global sensitivity* of  $f$  is defined as follows:

$$GS(f) = \max_{x, y : d(x, x') \leq 1} |f(x) - f(x')| \quad (1)$$

Here,  $d(x, x')$  represents the *distance* between two datasets  $x$  and  $x'$  and we say that two datasets are *neighbors* if their distance is 1 or less. How this distance is defined has a huge effect on the definition of privacy we obtain.

- The distance between two datasets should be equal to 1 (i.e. the datasets are neighbors) if they differ in the data of exactly one individual.
- This idea is easy to formalize in some contexts (e.g. in the US Census, each individual submits a single response containing their data) but extremely challenging in others (e.g. location trajectories, social networks, and time-series data).

Formally, the distance is encoded as a *symmetric difference* between the two datasets:

$$d(x, x') = |x - x' \cup x' - x|$$

***This particular definition has several interesting and important implications, for example:***

- If  $x'$  is constructed from  $x$  by *adding one row*, then  $d(x, x') = 1$
- If  $x'$  is constructed from  $x$  by *removing one row*, then  $d(x, x') = 1$
- If  $x'$  is constructed from  $x$  by *modifying one row*, then  $d(x, x') = 2$

In other words, adding or removing a row results in a neighboring dataset at distance 1, whereas modifying a row results in a dataset at a distance 2.

This particular definition of distance results in what is typically called *unbounded differential privacy*. Many other definitions are possible, including one called *bounded differential privacy* in which modifying a single row in a dataset *does* result in a neighboring dataset.

The definition of global sensitivity says that for *any two* neighboring datasets  $x$  and  $x'$ , the difference between  $f(x)$  and  $f(x')$  is at most  $GS(f)$ . **This measure of sensitivity is called “global” because it is independent of the actual dataset being queried, it holds true for *any* choice of neighboring  $x$  and  $x'$ .**

## Other Important Concepts

## L1 and L2 Norms

The **L1 norm** of a vector  $V$  of length  $k$  is defined as (i.e. it's the sum of the vector's elements).

$$\|V\|_1 = \sum_{i=1}^k |V_i|$$

**Example:** In 2-dimensional space, the L1 norm of the difference between two vectors yields the “Manhattan distance” between them.

The **L2 norm** of a vector  $V$  of length  $k$  is defined as (i.e. the square root of the sum of the squares).

$$\|V\|_2 = \sqrt{\sum_{i=1}^k V_i^2}$$

**Example:** In 2-dimensional space, this is the “Euclidian distance,” and it's always less than or equal to the L1 distance.

## L1 VS L2 Sensitivities

The L1 sensitivity of a vector-valued function is equal to the **sum of the element wise sensitivities**. For example, if we define a vector-valued function  $f$  that returns a length- $k$  vector of 1-sensitive results, then the L1 sensitivity of  $f$  is  $k$ .

The L2 sensitivity of a vector-valued function is a vector-valued function  $f$  returning a length- $k$  vector of 1-sensitive results has L2 sensitivity of  $\sqrt{k}$ . **For long vectors, the L2 sensitivity will obviously be much lower than the L1 sensitivity! For some applications, like [machine learning](#) algorithms (which sometimes return vectors with thousands of elements), L2 sensitivity is *significantly* lower than L1 sensitivity.**

## Local Sensitivity

Local sensitivity attempts to calculate the sensitivity for a local dataset, where the possible **changes are bound by the local data set and not the universe of all [data sets](#)**.

Given a query function  $f$  that is operating on a data set  $D_1$ , the local sensitivity is then the maximum differences that one change in  $D_1$  can produce:

$$\Delta f_{LS} = \max_{D_2} \|f(D_1) - f(D_2)\|_1$$

$D_1$  is the known dataset and  $D_2$  is another dataset with at most one different element (relative to the data set  $D_1$ ).

For queries such as **count** or **range**, the local sensitivity is identical to the global sensitivity.

**Note:** The feasibility of replacing the global with the local sensitivity have been proven to be **problematic and additional limitations are needed** in order to satisfy the conditions of  $\epsilon$ -Differential Privacy. It is seen that the magnitude of the noise could leak information due to the fact the *amount of noise reveals information about the data set to adversaries* as from the formula, every differentially private algorithm must add a noise at least as large as the local sensitivity. **Local sensitivity is the minimum sensitivity needed for a query to cover one specific data set.**

You may also check this — [A Case Study on Differential Privacy — Sensitivity \(pg-19\)](#)

Similarly, **another way of looking at it is:**

Formally, the local sensitivity of a function  $f : \mathcal{D} \rightarrow \mathbb{R}$  at  $x : \mathcal{D}$  is defined as:

$$LS(f, x) = \max_{x' : d(x, x') \leq 1} |f(x) - f(x')|$$

Global sensitivity considers *any* two neighboring datasets, but since we're going to run our differentially private mechanisms on an *actual* dataset — shouldn't we consider neighbors of *that* dataset?

The intuition behind *local sensitivity* is to **fix one of the two datasets** to be the **actual dataset being queried**, and consider all of its neighbors. Global sensitivity considers *any* two neighboring datasets, but since we're going to run our differentially private mechanisms on an *actual* dataset — shouldn't we consider neighbors of *that* dataset?

Local sensitivity is a function of both the query ( $f$ ) and the *actual* dataset ( $x$ ). Unlike in the case of global sensitivity, we **can't** talk about the local sensitivity of a function ***without also considering the dataset at which that local sensitivity occurs.***

Local sensitivity allows us to place **finite bounds** on the sensitivity of some functions whose global sensitivity is difficult to bound. This is because ***local sensitivity measure is defined in terms of the actual dataset's size, which is not possible under global sensitivity.***

**But, the issue with local sensitivity** is, because it itself depends on the dataset, if the analyst knows the local sensitivity of a query *at a particular dataset*, then the analyst may be able to infer some information about the dataset. It's therefore *not possible* to use local sensitivity directly to achieve differential privacy.

Moreover, keeping the local sensitivity secret from the analyst *doesn't help either*. It's possible to determine the scale of the noise from just a few query answers, and the analyst can use this value to infer the local sensitivity. ***Differential privacy is designed to protect the output of  $f(x)$  — not of the sensitivity measure used in its definition.***

To solve this, ***Propose-test-release and Smooth Sensitivity like approaches*** have been proposed for *safely using local sensitivity*, which is beyond the scope of this blog post, but if you are interested to know more about it — [check these notes shared by Professor Joseph Near](#).

**Now, you might be wondering why not just go with Global sensitivity? Why waste time and effort on local sensitivity and on creating the better safer approaches of it?**

Global sensitivity considers *any* two neighboring datasets, i.e., *Global sensitivity is the minimum sensitivity needed for a query to cover all possible datasets*. But since we will run our differentially private mechanisms on an *actual* dataset we might want to consider neighbors of *that* dataset! Thus, *Local sensitivity is the minimum sensitivity needed for a query to cover one specific data set*.