1. **What is Row Context in DAX?**
    - **Row Context** exists when DAX evaluates an expression **row by row** in a table.
    - Each row has its own "context", meaning DAX knows *which row* it is currently calculating.
    - You usually see **Row Context** in **calculated columns**, because they are evaluated per row.

🔹 **Example:**

Suppose we have a table Sales:

ProductID	SalesAmount	Cost
1	1000	700
2	2000	1200
3	1500	1000

Now, we add a **calculated column**:

Profit = Sales[SalesAmount] - Sales[Cost]

👉 Here, DAX evaluates the formula for **each row**:

- Row 1: 1000 - 700 = 300
- Row 2: 2000 - 1200 = 800
- Row 3: 1500 - 1000 = 500

🔹 **Why is this Row Context?**

Because the formula automatically applies **to the current row only** (no need to specify filters).
Each row has its own context — SalesAmount and Cost are picked from that row.

✅ **Summary**:

- **Row Context = row-by-row evaluation.**
- Mostly applies to **calculated columns** (and iterators like SUMX, FILTER).
- In the example, Profit column used row context to calculate values.

5. 🔹 **VAR**

- VAR is used to **define a variable** inside a measure or calculated column.
- You can assign an intermediate calculation to a variable so that it can be reused multiple times in your formula.
- This makes your DAX code **cleaner, easier to read, and faster** (because the calculation is performed once).

Example:

```
TotalProfit =
VAR SalesAmount = SUM(Sales[Amount])
VAR CostAmount  = SUM(Sales[Cost])
RETURN
    SalesAmount - CostAmount
```

Here:

- SalesAmount and CostAmount are **variables** defined using VAR.

🔹 **RETURN**

- RETURN tells DAX **what to output** after defining variables.
- It is the **final expression** that gets evaluated and returned as the result of the measure.
- Without RETURN, your VAR values don't mean anything — they need to be combined or used in RETURN.

In the example above:

```
RETURN
    SalesAmount - CostAmount
```

means the result of TotalProfit = difference between the two variables.

✅ **Summary**

- **VAR** → defines temporary storage (like a note where you write intermediate results).
- **RETURN** → tells Power BI what to finally show from those stored values.

10. The **reason CALCULATE overrides filters** is because of how it works with **filter context** in DAX. Let's break it down clearly:

🔑 1. CALCULATE = Evaluate in a Modified Filter Context

- Normally, a measure like:
- SUM(Sales[Amount])

runs inside whatever filters are applied (like slicers, visuals, or page filters).

- But when you wrap it in **CALCULATE**:
- CALCULATE(SUM(Sales[Amount]), Sales[Category] = "Electronics")

you are saying:

"Take the current filter context, **but replace/add these filters** before doing the calculation."

🔑 2. Override happens because CALCULATE rewrites the filter

If your report already has a filter like Sales[Category] = "Clothing",
then CALCULATE(..., Sales[Category] = "Electronics") will **ignore Clothing** and force Electronics.

👉 That's why it *overrides*.

🔑 3. If you want to keep existing filters instead of overriding

You can use functions like:

- FILTER (for more flexible conditions)
- KEEPFILTERS (to add instead of replace)
  Example:

```
CALCULATE(
   SUM(Sales[Amount]),
   KEEPFILTERS(Sales[Category] = "Electronics")
)
```

This will respect current filters and only keep rows where Category = Electronics **within them**, instead of replacing.

✅ In short:

- **CALCULATE** changes the filter context.
- If you pass conditions, they **replace existing filters** on the same column.
- That’s why it looks like CALCULATE "ignores" previous filters.

14. **Example:**

Suppose you have a Sales table with Category and SalesAmount.

```
Total Sales (Ignore Filters) =
CALCULATE (
   SUM ( Sales[SalesAmount] ),
   ALL ( Sales[Category] )
)
```

🔎 **How this works:**

- SUM(Sales[SalesAmount]) → normal total under current filters.
- ALL(Sales[Category]) → removes **Category filter**, so even if the user clicks only *Electronics*, this measure still shows the **total sales across all categories**.

💡 **Simulating a “Remove Filters” button:**

1. Create two measures:
   - Total Sales → respects slicers/filters.
   - Total Sales (Ignore Filters) → uses ALL() to ignore.
2. Place both in a **card visual** or chart.
3. Now, when the user applies a slicer (like selecting "Electronics"),
   - the normal measure changes
   - the “Ignore Filters” measure behaves like a *reset/remove filters button*.

If a **CALCULATE** measure is **ignoring a slicer**, the **most likely cause** is that inside the CALCULATE() you have used a function like **ALL()**, **REMOVEFILTERS()**, or something similar that **overrides or removes slicer filters**.

🔎 **Common causes:**

1. **Using ALL() or REMOVEFILTERS()**
2. Sales Ignore Category =
3. CALCULATE (
4.   SUM (Sales[SalesAmount]),
5.   ALL (Sales[Category])   -- removes slicer on Category
6.)

→ Any slicer on *Category* will be ignored.

7. **Filter in CALCULATE conflicts with slicer**
8. Electronics Sales =
9. CALCULATE (
10.   SUM (Sales[SalesAmount]),
11.   Sales[Category] = "Electronics"
12.)

→ Even if slicer picks *Furniture*, this measure will **force Electronics only**.

13. **Wrong relationship / inactive relationship**
    - If slicer is based on a table not related (or incorrectly related) to Sales, the slicer won't filter.
    - Sometimes you need USERELATIONSHIP() in CALCULATE.

✅ **Rule of thumb:**

- If you see slicers being ignored → check if your measure uses ALL(), REMOVEFILTERS(), or a **hardcoded filter inside CALCULATE**.