Parameterizing data pipelines in **Palantir Foundry** is a powerful way to make your pipelines more **flexible**, **reusable**, and **dynamic**. It allows the same pipeline logic to run under different configurations (e.g., for different regions, time ranges, or customers) without duplicating code or logic.

Here's a breakdown of how to **parameterize pipelines** in Foundry:

🔧 1. Using Pipeline Templates with Parameters

Foundry supports pipeline **templates**, where you define parameters that are passed in at runtime.

💡 Example Parameters:

- start_date: Date from which to extract records
- region_code: Region or country to filter data
- schema_version: Version of the schema to apply

How to Define Parameters:

In a **Code Workbook** or **Code Repository (e.g., PySpark or SQL)**:

```
params = context.params


start_date = params.get("start_date", "2023-01-01")

region = params.get("region_code", "US")
```

In a **SQL Transformation**:

```
SELECT *

FROM transactions

WHERE region = '${region_code}'

  AND transaction_date >= '${start_date}'
```

${parameter_name} is the syntax for referencing parameters in SQL blocks.

🛠️ 2. Declaring Parameters in the Transformation Settings

In **Foundry UI (Code Workbook or Transformation Graph)**:

1. Click on a **transformation node**.
2. Go to the **"Parameters" tab**.

3. Define the parameters and their **default values**.
4. Optionally, link these parameters to **global variables** or **upstream datasets**.

🔁 3. Using Parameterized Pipelines in Scheduled Jobs

You can set pipeline parameters when scheduling jobs (for example, to run the pipeline every day with today's date):

- In the **Job Scheduler**, set parameters dynamically using expressions like:

  ```
  {
    "start_date": "${TODAY.minusDays(1)}",
    "region_code": "EU"
  }
  ```

This lets your pipeline adapt to each scheduled run.

📦 4. Templated Datasets with Parameterized Paths

If you store outputs per parameter (e.g., per region):

```
output_path = f"/pipeline_output/region={region}/date={start_date}/"
df.write.format("parquet").save(output_path)
```

Foundry can interpret these as **templated datasets**, enabling efficient storage and querying via partitioning.

🔍 5. Validation and Defaults

Always define **fallback/defaults** to ensure your pipeline doesn't break if parameters are missing:

```
region = params.get("region", "global")
```

Also consider adding **assertions** to validate inputs:

```
assert region in ["EU", "US", "APAC"], "Invalid region parameter!"
```

🧠 Best Practices

- **Use consistent parameter naming** across your pipelines.
- **Avoid hardcoding values** when they can be parameterized.

- Use parameters for:
    - Dates or time windows
    - Region, market, or customer filters
    - Feature toggles (e.g., enable/disable enrichment logic)
- Document parameter use at each transformation node for clarity.
