UDF(): {user defined function}

→ lets us to write custom logic in python & apply it to spark Dataframe columns row by row when built-in functions aren't enough.

```
def lower_name(x):
    return x.lower()
```

```
udf_fun_lower = udf(lower_name, StringType())
```

```
df.withColumn(
        "lower_names",
        udf_fun_lower( col("name"))
)
```

Note:

> UDFs are slow compared to built-in functions becau

- They bypass catalyst optimizer
- Data crosses JVM ↔ python boundary
- No vectorization (unless using pandas udf)

transform()

pyspark.sql.DataFrame.transform() is used to chain the custom transformation & this function returns new DataFrame after applying the specified transformation.

→ transform() applies a function to a DataFrame & returns a new DataFrame

→ Its not row-wide like UDF
It works on the whole DataFrame

Example:

```
def select_basic(df):
    return df.select("id", "sales")

df1 = df.transform(select_basic)
```

⇒ ① clean nulls:

```
def clean_nulls(df):
    return df.fillna({"sales": 0, "name": "NA"
                     })

df_clean = df.transform(clean_~~sales~~nulls)
```

chaining multiple transforms:

```
def filter_sales (df):
    return df.filter (col("sales") > 0)

def add_margin (df):
    return df.withColumn ("margin", col("profit")/
                                                   col("sales")

df_final = (
        df
        .transform (clean_nulls)
        .transform(filter_sales)
        .transform(add_margin)
)
```

passing parameters into transform():

```
def filter_by_threshold(th):
    return lambda df : df.filter(col("sales") > th

df.transform(filter_by_threshold (1000))
```

transform() vs udf()

Aspect	transform()	udf()
level	Dataframe	Row
performance	Fast	slow
uses catalyst	Yes	No
Best for	Pipelines	custom row logic

Temp view

- A temp view is a temporary SQL Table name given to a Dataframe so you can query it using SQL.

```
Dataframe ⟶ Temp view(name) ⟶ spark.sql("select...")
```

→ temp view ⇒ name only, not data copy
→ exists only for this spark session

Example:

```
data = [
    (1, 'A', 500), (2, 'B', 700), (3, 'C', 800)
]

df = spark.createDataframe(data, ["id", "name", "salary"])
```

#create a Temp view

```
df.createOrReplaceTempView("employee")
```

↓

Gives this Dataframe; a SQL table name

Query using SQL

```
spark.sql("""
        select  name, salary
        from  employee
        where  salary > 500
    """
)
```

Therefore, createOrReplaceTempView() → will helps us to create a temporary sql table on the dataframe; & using the view we can query the data by writing SQL queries.

Global Temp View:

> A global temp view is a temporary SQL view that is accessible across All spark sessions within the same spark application

> same cluster | spark application
> Different notebooks | sessions
> shared temporary view

```
df.createOrReplaceGlobalTempView("sales")
```

Note:
spark automatically stores it in a special database called global_temp

```
⇒ spark.sql("""
        select *
        from global_temp.sales
        """)
```

(Must use).

Temp view vs Global view

Feature	Temp view	Global temp view
scope	single session	All session
Database	none	global_temp
cluster restart	X gone	X gone
SQL access	Direct	global_temp.view_name
used in production	rare	X Almost never