collect()

- `collect()` is an Action that brings all rows of a DataFrame/RDD to the driver as a python list

- Executors ⟶ Driver (everything at once)

- Spark runs distributed, but `collect()` pulls all data into one machine's memory

Syntax:

```
rows = df.collect()
```

→ returns: List[row]

→ triggers execution immediately [ACTION]

Ex:

```
rows = df.select('id', 'name').collect()
rows[0]['name']
```

Why collect() is dangerous?

- OOM risk (driver runs out of memory)
- slow for large datasets
- crashes notebooks/jobs in Databricks.
- Never use on large tables

When is collect() acceptable?

- use only when you're sure the date is tiny:
  - > Debugging a few rows
  - > unit tests
  - > fetching a small lookup list

Alternatives of collect(): show(20), take(), head(), limit()

StructType & StructField

StructType : defines the entire schema (table structure)

StructField : defines one column inside that schema

- Together, they let us explicitly control column names, data types & nullability.

Why we should care?

↳ If we don't define a schema, then Spark infers it
↳ Inference can be slow, wrong, inconsistent
↳ Explicit schema ⟶ faster + safer + production-ready

```
from pyspark.sql.types import (
    StructType, StructField,
    IntegerType, StringType, DoubleType
)
```

```
StructField("id", IntegerType(), nullable=True)
```

```
schema = StructType([
    StructField("id", IntegerType(), True),
    StructField("name", StringType(), True)
])
```

⟶ creating DataFrame with explicit schema

```
data = [
    (1, 'A'), (2, 'B')
]
```

df= spark.createDataframe (data, schema)

---

* Reading files with schema :

```
df = spark.read\
        .schema(schema)\
        .option ('header', True)\
        .csv (——)
```

. faster than inference
. prevents wrong types

Nested schema :

```
nested_schema = StructType ([
        StructField ("id", IntegerType(), True),
        StructField ("address", StructType ([
                StructField("city", StringType(), True),
                StructField("zip", StringType(), True)
        ]), True)
])
```

{⇒ RDD = Dataframe with StructType}

pivot() & unpivot()

pivot() → turns row values into columns

unpivot() → turns columns back into rows

Pivot() { row → columns }

When to use :

- > reports/ dashboards
- > Category wise metrics as columns
- > SQL-style crosstabs

Basic Syntax :

```
df.groupBy(group_cols).pivot(pivot_col).agg(agg_f
```

Ex:

State	Product	Sales
AP	Shoes	100
AP	Shirts	80
TS	Shoes	120

pivot product-wise sales per state :

```
df.groupBy("state")\
  .pivot("product")
  .agg(sum('sales'))\
```

Op

State	Shoes	Shirts
AP	100	80
TS	120	NULL

→ If we know pivot values, specify them to avoid an extra pass:

```
df.groupBy("state")\
  .pivot("product", ["shoes", "shirts"])\
  .agg(sum("sales"))
```

→ Common rules:

① `pivot()` must be used with `groupBy()`

② Always needs an aggregation

③ Too many pivot values ⇒ many columns ⇒ (memory risk)

---

unpivot(): { Columns → Rows }

→ pyspark doesn't have a single `unpivot()` function everywhere, but you can do it ~~directly~~ cleanly using `stack()` in SQL

⇒ When to use:

- > Normalize wide data
- > ML features
- > Schema standardization

```
df_unpivot = df.select(
    "state",
    expr("stack(2, 'shoes', shoes, 'shirts', shirts) as (product, sales)")
)
```

⇒ we have;

State	Shoes	Shirts
AP	100	80
TS	120	NULL

we want → | State | product | sales |

That is; { column names become values
column values Become raw values }

How unpivot works?

↓

for each row;

Take column "Shoes" → create a row (product: shoes, sales = value)

Take column "Shirts" → create another row.

```
df_unpivot = df.select(
        "state",
        expr(
            "stack( 2, "Shoes", shoes,
                       "Shirts", shirts)
             as (product, sales)"
        )
)
```

after "State"; how many columns we need to unpivot

2 → number of columns to unpivot

'shoes' → new value for product

shoes → value goes into sales

'shirts' → next product

shirts → next sales value

=> Why unpivot() is useful?

1. ML & analysis
2. schema consistency
3. reverse of pivot