Operators

ETL IN PYTHON

Stefano Francavilla
CEO - Geowox

Operators

- **In** / **not in** comparison
- **Basic** comparison
- **Identity** comparison
- **String** comparison
- **Conjunctions** and **negations** comparison

```
in_, ~in_
==, !=, >, >=, <, <=
is_, is_not
like, notlike
and_, or_
```

and_() operator

- **Conjunction** of expressions in `WHERE` clause
- `from sqlalchemy import and_`
- Use:
  - `and_(expression1, expression2, ..., expressionN)`
  - `(expression1 & expression2 & ... & expressionN)`

and_() operator: an example

SQL

```sql
SELECT * FROM ppr_clean_all
WHERE date_of_sale >= '2021-01-01' AND date_of_sale <= '2021-01-10'
```

Python

```python
from sqlalchemy import and_

result = session.query(PprCleanAll)
                .filter(and_(PprCleanAll.date_of_sale >= '2021-01-01',
                             PprCleanAll.date_of_sale <= '2021-01-10'))
                .all()
```

or_() operator

- **Disjunction** of expressions in the `WHERE` clause
- `from sqlalchemy import or_`
- Use:
  - `or_(expression1, expression2, ..., expressionN)`
  - `(expression1 | expression2 ... | expressionN)`

or_() operator: an example

SQL

```sql
SELECT * FROM ppr_clean_all
WHERE price <= 50000 OR price >= 5000000
```

Python

```python
from sqlalchemy import or_

result = session.query(PprCleanAll)
    .filter(or_(PprCleanAll.price <= 50000,
                PprCleanAll.price >= 5000000))
    .all()
```

Let's practice!

ETL IN PYTHON

Sqlalchemy func

ETL IN PYTHON

Stefano Francavilla
CEO - Geowox

SQL aggregate functions

- Aggregate functions perform calculation on rows
- `COUNT()` , number of rows
- `SUM()` , sum of all or distinct values
- `MAX()` , maximum value
- `MIN()` , minimum value
- `AVG()` , average value

The func attribute

- `from sqlalchemy import func`
- Generates SQL functions (like `COUNT(*)`)
  - `COUNT` --> `func.count()`
  - `SUM` --> `func.sum()`
  - `MAX` --> `func.max()`
  - `MIN` --> `func.min()`
  - `AVG` --> `func.avg()`

The func attribute: an example

Table name: **Products**

id	category	name	price
1	books	Sapiens	12
2	electronics	iPhone 12	900
3	books	Measure What Matters	10
4	books	Greenlights	14
5	electronics	Macbook Pro 13	1500

```
class Products(Base):
    __tablename__ = "products"

    id = Column(Integer,
                primary_key=True)
    category = Column(String(55))
    name = Column(String(55))
    price = Column(Integer)
```

The func attribute: COUNT(*)

SQL

```sql
SELECT COUNT(*) FROM products
```

- `COUNT(*)`
  - **counts** null values
- `COUNT([Column name])`
  - **doesn't count** null values

Python

```python
result = session.query(func.count(Products.id))
                .all()

print("Result:", result)
```

```
Result: [(5,)]
```

- `id` primary key and not null
- `.all()` retrieves the query result

The func attribute: SUM()

SQL

```sql
SELECT SUM(price)
FROM products
```

Python

```python
result = session.query(func.sum(Products.price)).all()

print("Result:", result)
```

- `func.sum([Column name])`

```
Result: [(2436,)]
```

The func attribute: MAX() and MIN()

SQL

```sql
SELECT MAX(price), MIN(price)
FROM products
```

- `func.max([Column name])`
- `func.min([Column name])`
- `session.query([Arg1], [Arg2])`

Python

```python
result = session.query(func.max(Products.price),
                       func.min(Products.price)).all()

print("Result:", result)
```

```
Result: [(1500, 10)]
```

The func attribute: AVG()

SQL

```
SELECT AVG(price) FROM products
```

- `func.avg([Column name])`

Python

```
result = session.query(func.avg(Products.price)).all

print("Result:", result)
print("Result (int):", int(result[0][0]))
```

```
Result: [(Decimal('487.2'),)]

Result (int): 487
```

Group by

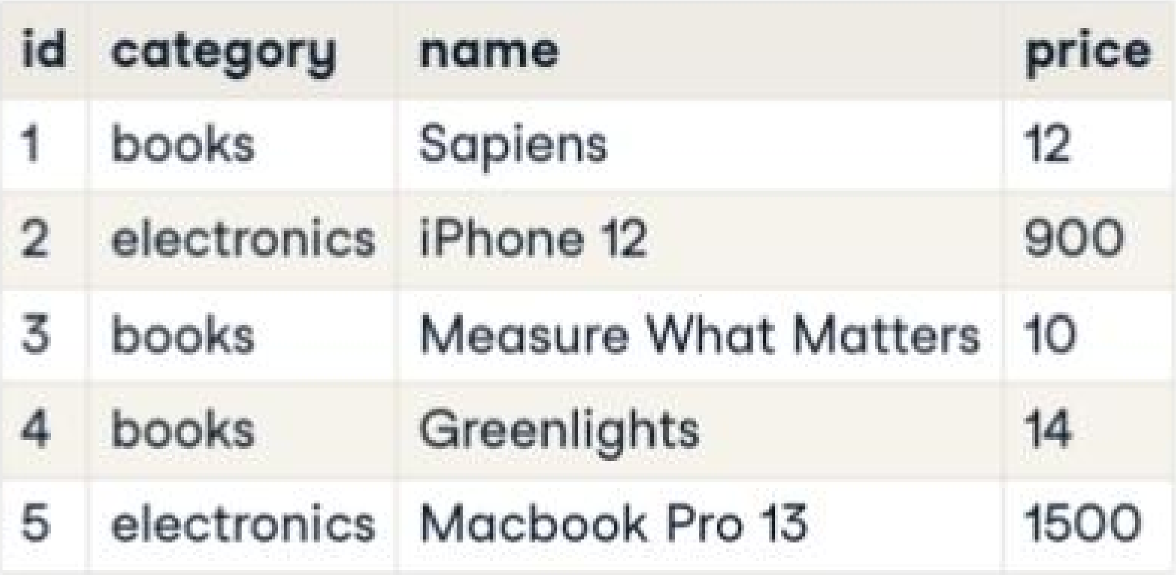
- `SELECT column FROM table GROUP BY <column>`
- Divides results into groups
- Then applies the aggregate function
- `session.query().group_by([Column name]).all()`

Group by: an example

id	category	name	price
1	books	Sapiens	12
2	electronics	iPhone 12	900
3	books	Measure What Matters	10
4	books	Greenlights	14
5	electronics	Macbook Pro 13	1500

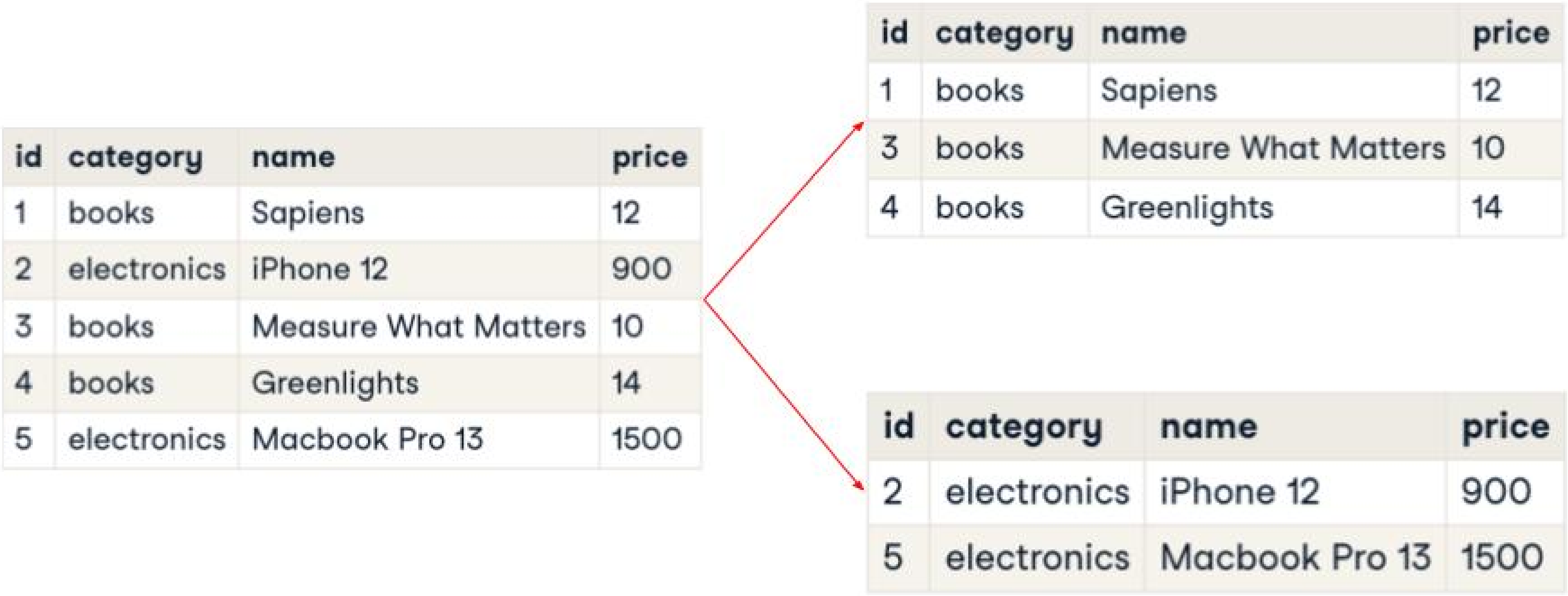
```sql
SELECT category, SUM(price)
FROM products
GROUP BY category
```

Group by: an example



id	category	name	price
1	books	Sapiens	12
2	electronics	iPhone 12	900
3	books	Measure What Matters	10
4	books	Greenlights	14
5	electronics	Macbook Pro 13	1500

Group by: an example

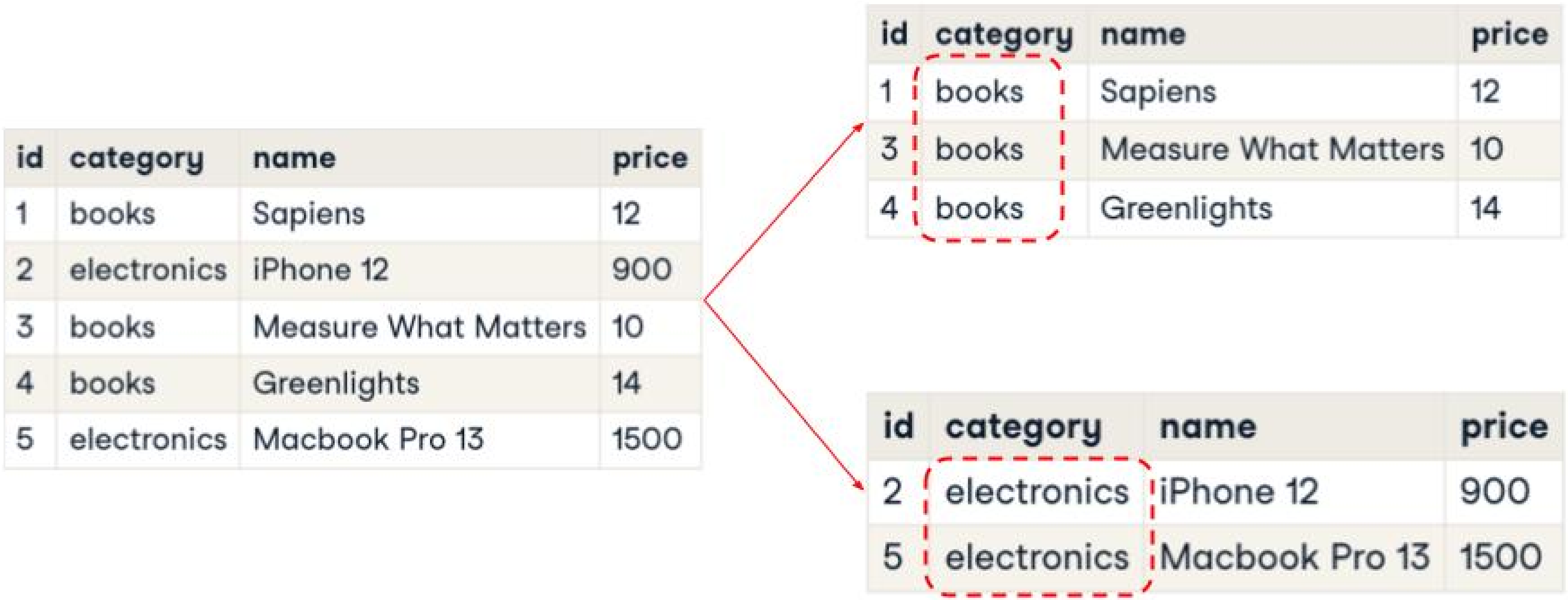


id	category	name	price
1	books	Sapiens	12
2	electronics	iPhone 12	900
3	books	Measure What Matters	10
4	books	Greenlights	14
5	electronics	Macbook Pro 13	1500

id	category	name	price
1	books	Sapiens	12
3	books	Measure What Matters	10
4	books	Greenlights	14

id	category	name	price
2	electronics	iPhone 12	900
5	electronics	Macbook Pro 13	1500

Group by: an example

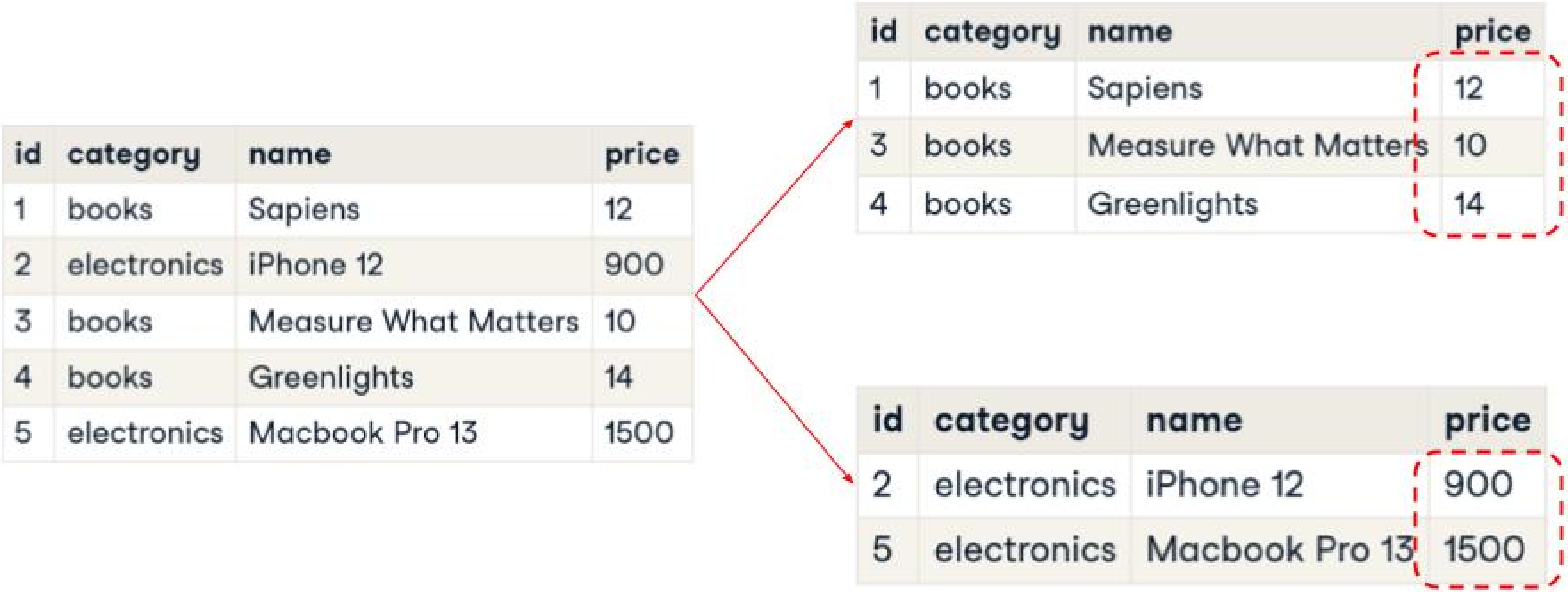


id	category	name	price
1	books	Sapiens	12
2	electronics	iPhone 12	900
3	books	Measure What Matters	10
4	books	Greenlights	14
5	electronics	Macbook Pro 13	1500

id	category	name	price
1	books	Sapiens	12
3	books	Measure What Matters	10
4	books	Greenlights	14

id	category	name	price
2	electronics	iPhone 12	900
5	electronics	Macbook Pro 13	1500

Group by: an example

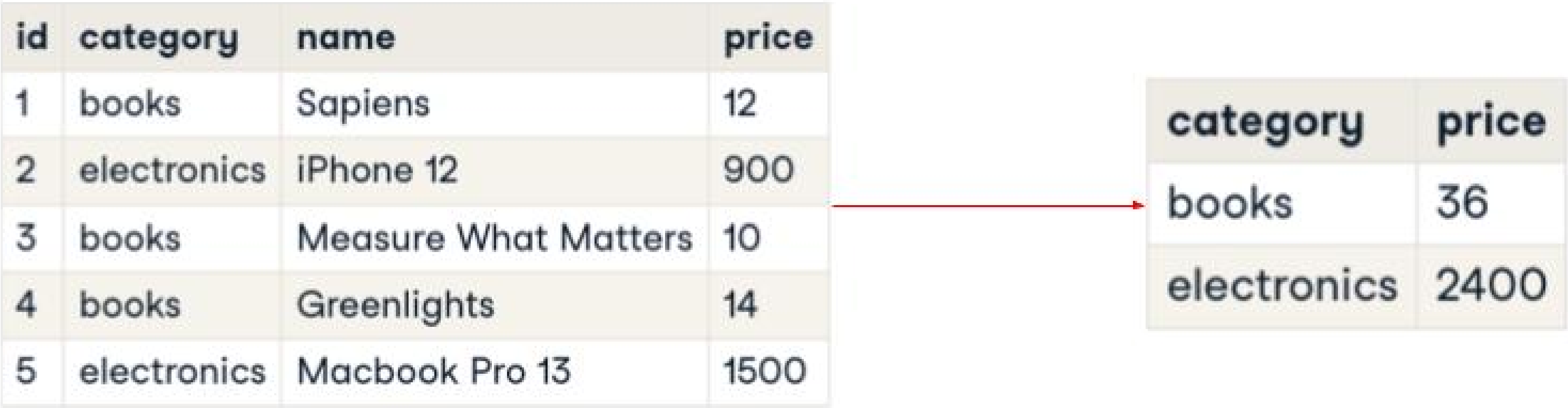


id	category	name	price
1	books	Sapiens	12
2	electronics	iPhone 12	900
3	books	Measure What Matters	10
4	books	Greenlights	14
5	electronics	Macbook Pro 13	1500

id	category	name	price
1	books	Sapiens	12
3	books	Measure What Matters	10
4	books	Greenlights	14

id	category	name	price
2	electronics	iPhone 12	900
5	electronics	Macbook Pro 13	1500

Group by: an example



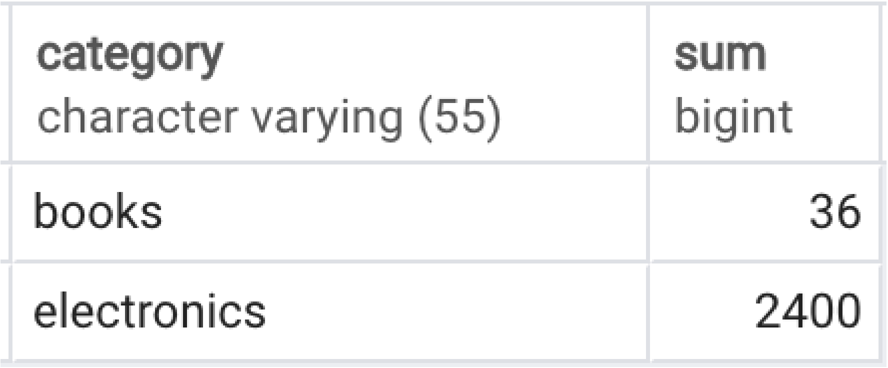
id	category	name	price
1	books	Sapiens	12
2	electronics	iPhone 12	900
3	books	Measure What Matters	10
4	books	Greenlights	14
5	electronics	Macbook Pro 13	1500

category	price
books	36
electronics	2400

Group by: an example

SQL

```sql
SELECT category, SUM(price)
FROM products
GROUP BY category
```



category character varying (55)	sum bigint
books	36
electronics	2400

Python

```python
result = session.query(Products.category,
                         func.sum(Products.price))
                       .group_by(Products.category).all()
print("Result:", result)
```

```
Result: [('books', 36), ('electronics', 2400)]
```

Let's practice!

ETL IN PYTHON

Create the insights

ETL IN PYTHON

Stefano Francavilla
CEO - Geowox

Where we left

ppr_clean_all

Columns
id
date_of_sale
address
postal_code
county
price
description

- Derive insights for shareholders
- Work with `date_of_sale` , `county` and `price`
  - `county` is a string
  - `price` is numeric
  - `date_of_sale` is a date

What insights do we need?

- **Total** number sales
- **Sum** of all sales in euro
- **Highest** sold price
- **Lowest** sold price
- **Average** sold price
- Specific **date** range (e.g. first quarter, from 2021-01-01 to 2021-03-30)
- For each **county** (e.g. Dublin, Galway, Cork, etc.)

What insights do we need?

SQL

```
SELECT county,
       COUNT(*),
       SUM(*),
       MAX(price),
       MIN(price),
       AVG(price)
FROM ppr_clean_all
WHERE data_of_sale >= "2021-01-01" AND data_of_sale <= "2021-03-30"
GROUP BY county
```

Views

- Views are **not physically materialized**
- Create view with **pure SQL**
- Defined with `CREATE OR REPLACE VIEW <table_name> AS` query

```sql
CREATE OR REPLACE VIEW insights AS
SELECT * FROM ppr_clean_all
WHERE county='dublin'
```

Raw SQL on SQLAlchemy

- `session.execute("CREATE OR REPLACE VIEW AS ...")`
- `session.commit()`

Let's practice!

ETL IN PYTHON

Working with Excel files

ETL IN PYTHON

Stefano Francavilla
CEO - Geowox

xlsxwriter library

- `import xlsxwriter`
- `.xlsx` file format
- Can write:
  - text
  - numbers
  - formulas
  - images
  - charts
  - others

Workbook

```python
import xlsxwriter
workbook = xlsxwriter.Workbook("Insights.xlsx")
# Do things with the workbook
workbook.close()
```

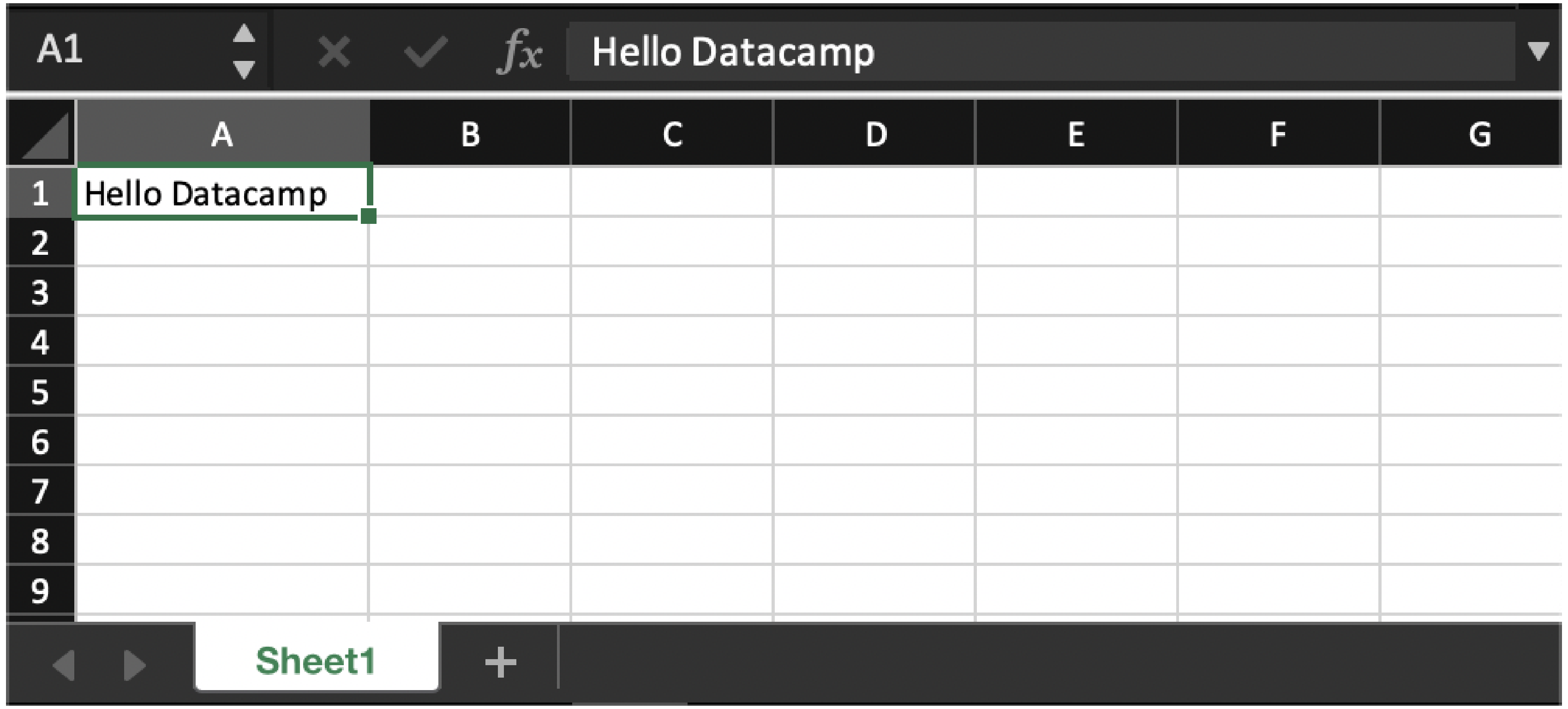
- `xlsxwriter.Workbook()` creates `.xlsx` files
- `workbook.close()` closes the file

Worksheet

- Manages Excel file sheets:
  - formulas
  - data
  - graph
- `worksheet = workbook.add_worksheet()`
- Default names: `Sheet1` , `Sheet2` , `...` , `SheetN`
- `worksheet = workbook.add_worksheet("Results")`
- `worksheet.write(row, column, data)`
  - A1 is `(0, 0)`

Worksheet: an example

```
import xlsxwriter
workbook = xlsxwriter.Workbook("Greetings.xlsx")
worksheet = workbook.add_worksheet()
worksheet.write(0, 0, "Hello Datacamp")
workbook.close()
```



A1 *fx* Hello Datacamp

	A	B	C	D	E	F	G
1	Hello Datacamp						
2							
3							
4							
5							
6							
7							
8							
9							

Worksheet: add_table()

- `add_table()` adds table into the Excel file
- `add_table(<range (e.g. "B3:C5")>, {options})`
- `options` :
  - `data` specifies data to insert in cells
  - `columns` sets specific properties
- `"columns": [{"header": "Header 1"}, ..., {"header": "Header N"}]` overrides default column names (e.g. `Column1` , `...` , `ColumnN`)

```
{"columns": [
              {"header": "id"},
              {"header": "name"}
           ]
}
```

add_table(): an example

Books.xlsx

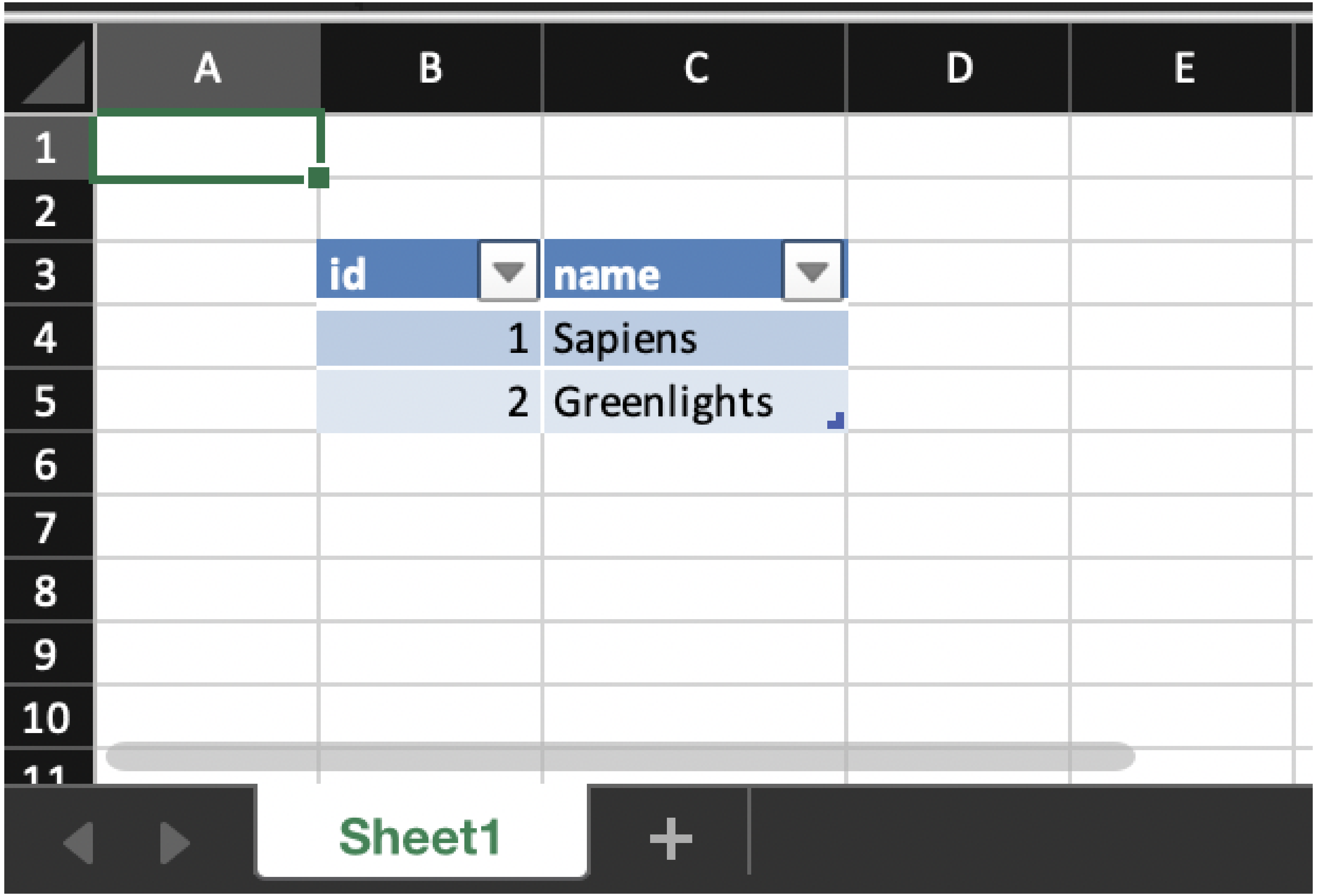
id	name
1	Sapiens
2	Greenlights

add_table(): an example

```python
import xlsxwriter

workbook = xlsxwriter.Workbook("Books.xlsx")
worksheet = workbook.add_worksheet()
data = [[1, "Sapiens"],
        [2, "Greenlights"]]
worksheet.add_table(
    "B3:E6", {"data": data,
              "columns": [
                 {"header": "id"},
                 {"header": "name"}
              ]})
workbook.close()
```

add_table(): an example



id	name
1	Sapiens
2	Greenlights

Let's practice!

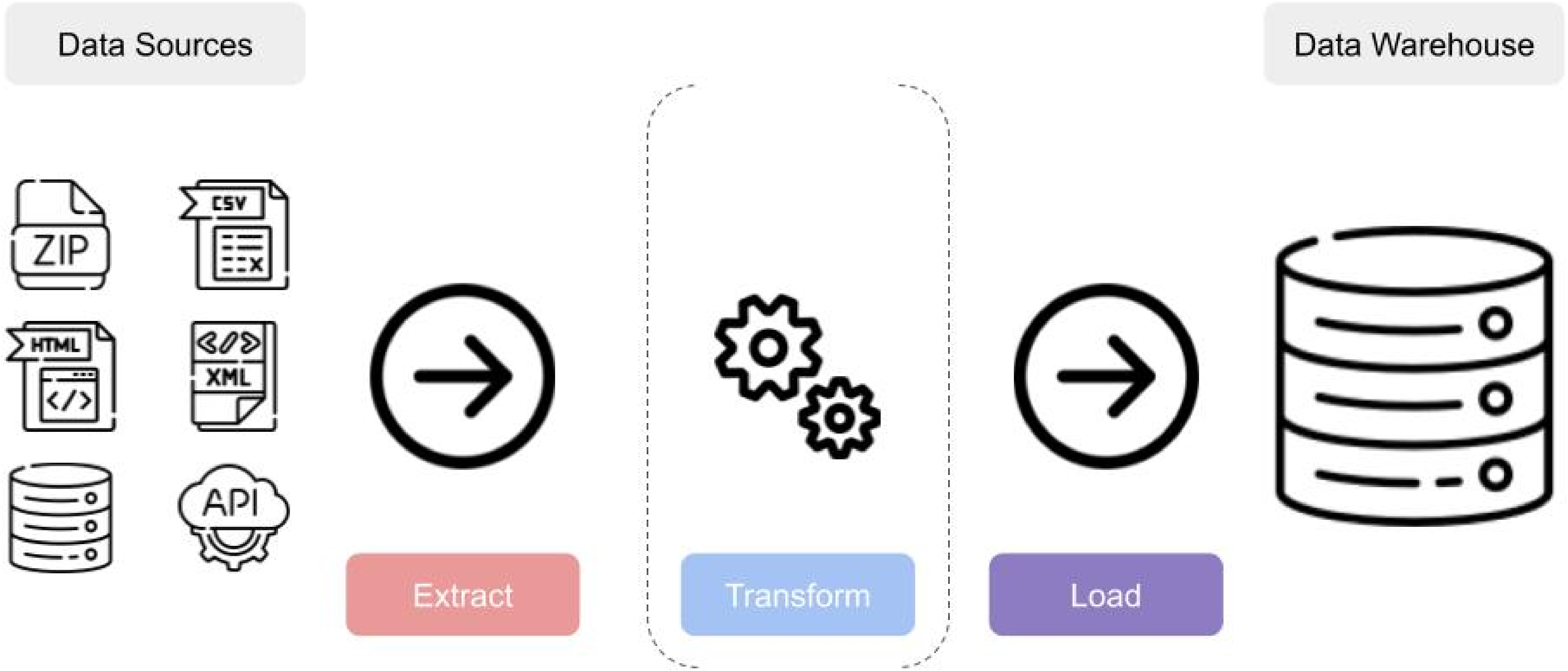
ETL IN PYTHON

Wrap-up

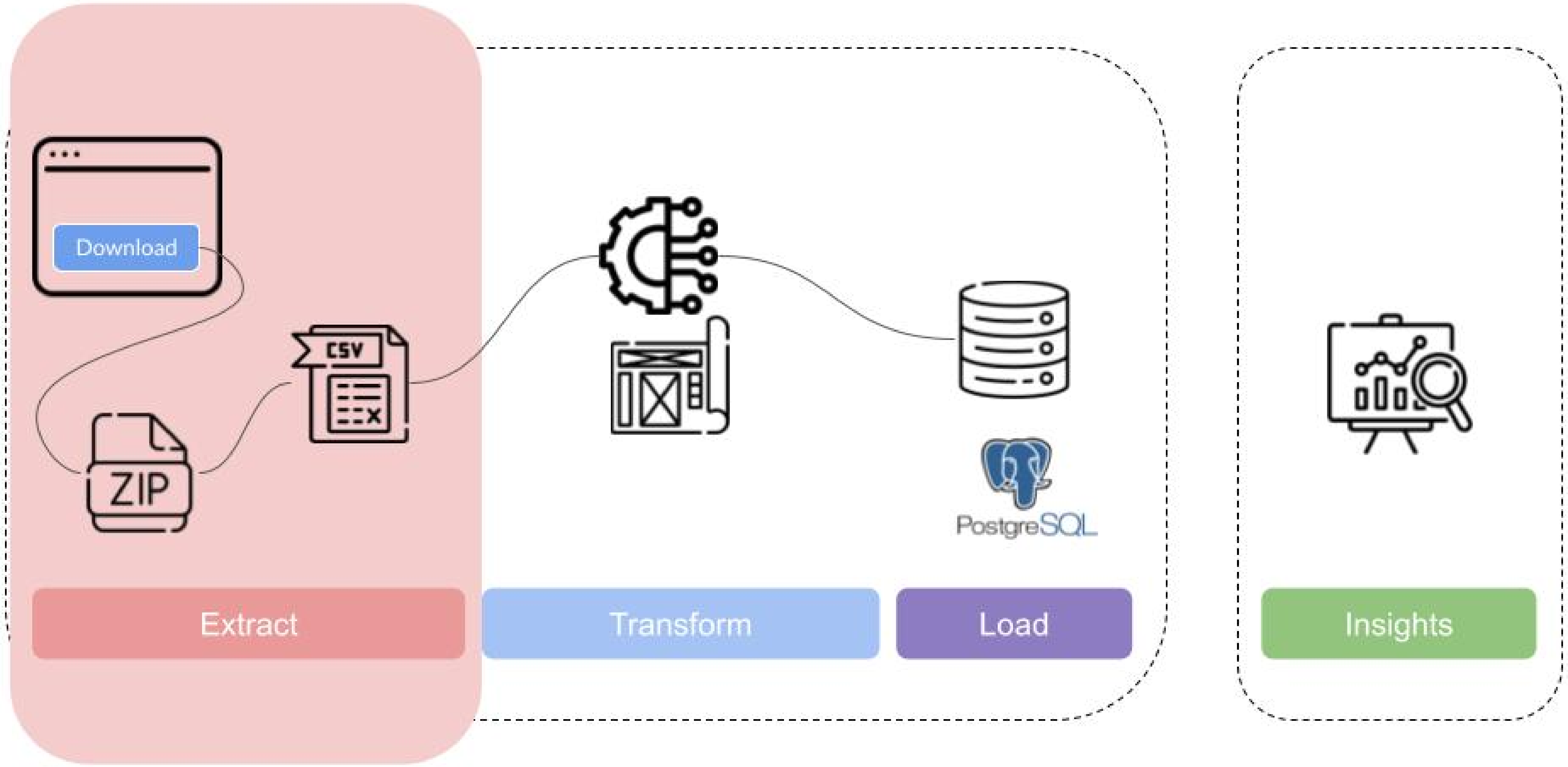
ETL IN PYTHON

Stefano Francavilla
CEO - Geowox

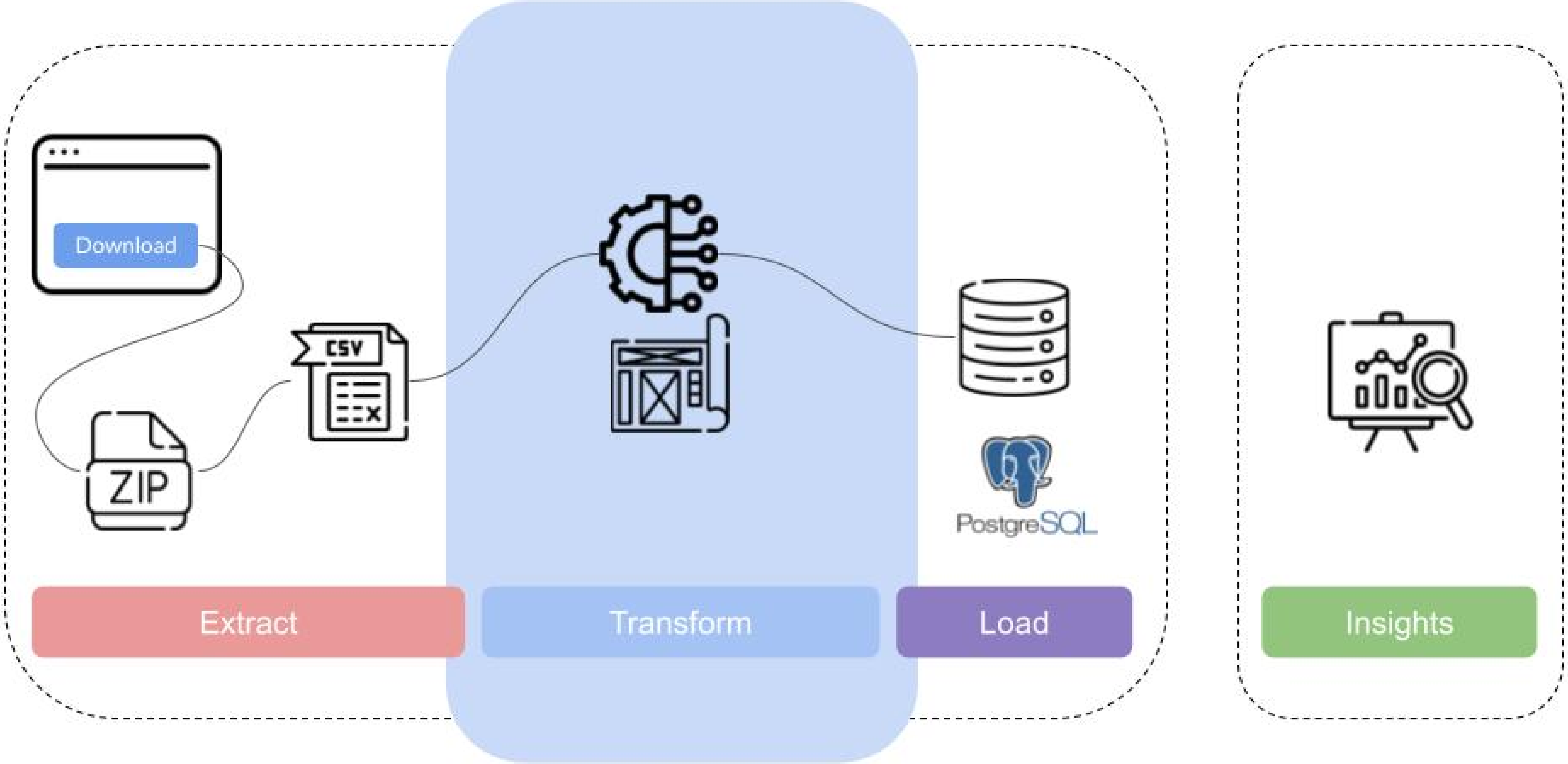
Explore the data and extract



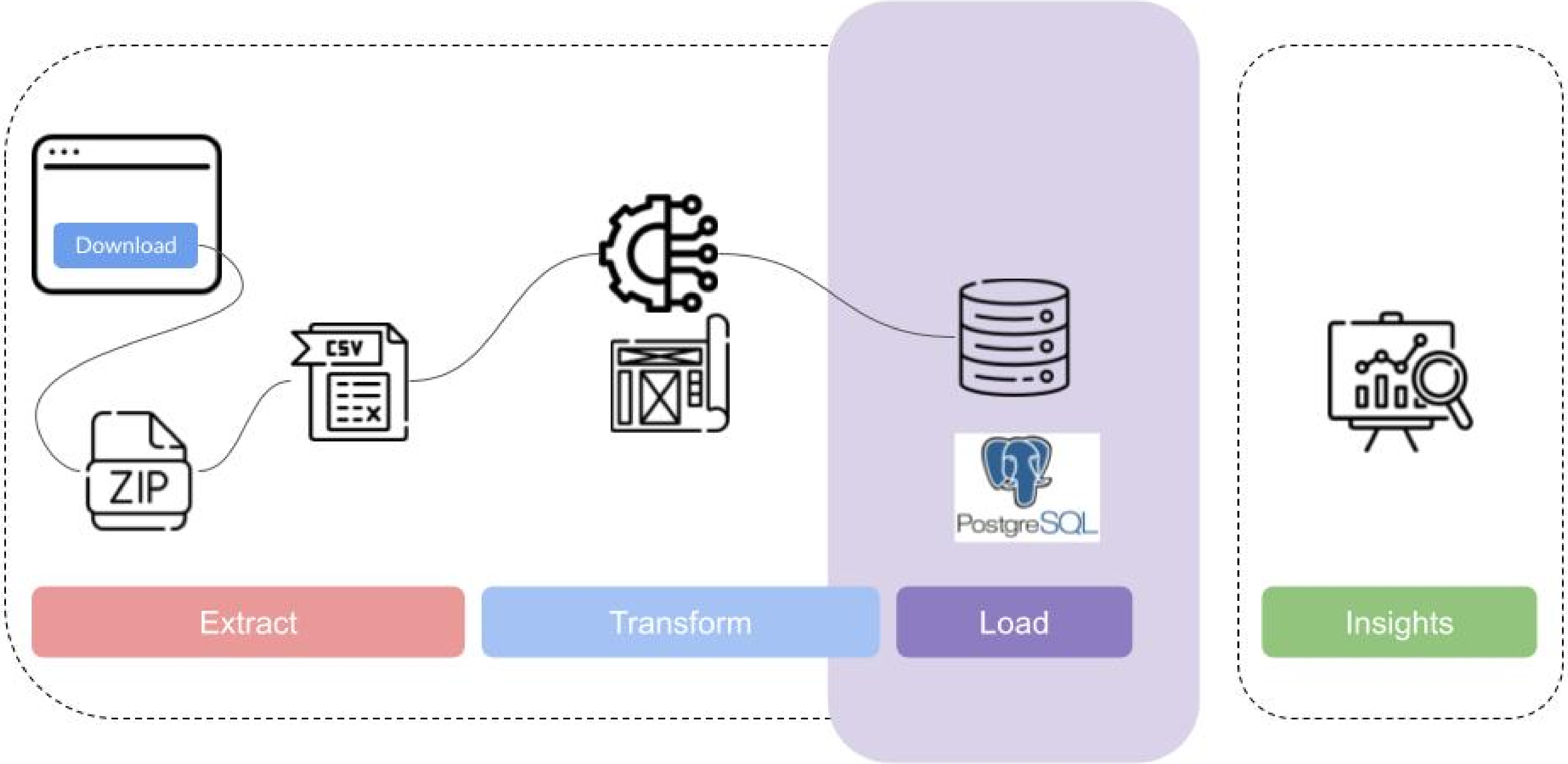
Explore the data and extract



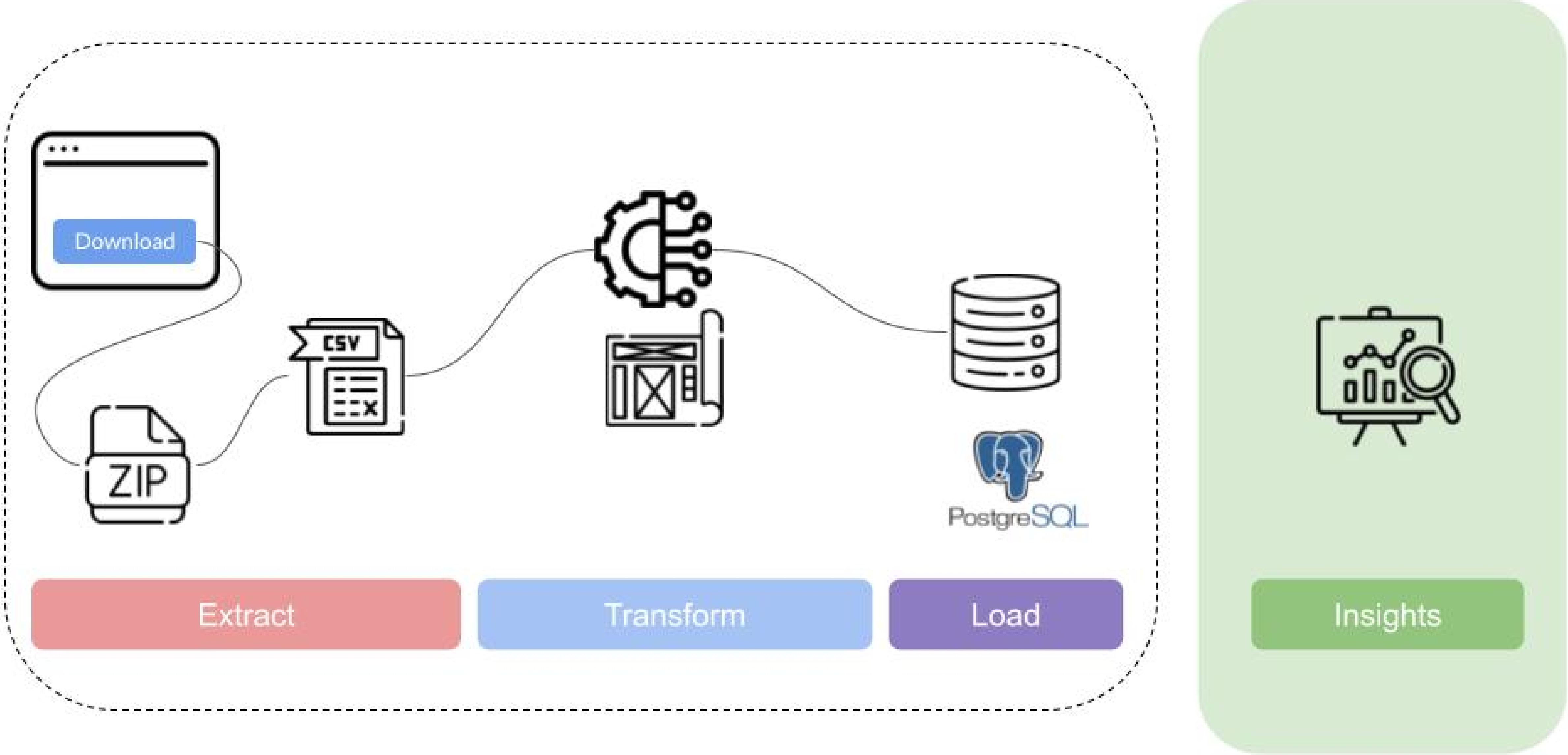
ETL foundations and tranform



From raw to clean: load



From clean data to meaningful insights



Thank you!

ETL IN PYTHON