Unit testing with Databricks

Jonathan Neo

Senior Data Engineer, Cuusoo

Who is Cuusoo

Reimagine data without the limits of the status quo, make it happen with databricks

Roadmap, design and strategy

We help you get started linking your overall strategy to design and use cases

Databricks deployment

Help you re-imagine your data environment to be powered by databricks and not constrained by the messy status quo

Databricks tune up

We assess your Databricks platform and usage within your business context and recommend and implement ways to reduce consumption, strengthen security and optimise performance, so you get more from your investment.

Custom use cases

We flesh that out into well-defined use cases and then make them happen on Databricks. It can be any combination of data engineering, analytics, data science and machine learning.

Rapid value accelerators

Helping you to implement the databricks accelerators that already exist across verticals including, but not limited to, financial services, healthcare, retail and consumer goods and manufacturing.

Machine learning at scale

Standing up your scaled machine learning environment and helping you team to implement the best practices and systems to get the most from databricks

Cuusoo

Koo-Soh

Cuusoo, pronounced 'koo-soh', means imagination, vision and clean-slate thinking. Cuusoo will focus on helping businesses imagine data and its applications using a first principles approach.

CUUSOO

We’re are member of the Mantel group

We’re an Australian-owned, technology-led consulting with capabilities from strategy to managed services

Established in November 2017, we’re a dynamic and growing business currently comprised of seven brands. We’ve been recognised in the AFR’s 2020 fastest growing new companies and LinkedIn’s Top Australian Startups. Our plan is to go IPO in 2023. We have hubs in Melbourne, Sydney, Brisbane, Perth, Auckland, Queenstown and Magnetic Island, supporting a team of 450 that will grow to 550 over the next year.

Advisory
CTO Advisory
Security Advisory
Design Advisory

Design
UX and CX Design
Service Design
Customer Research

Data / AI / ML
Data Engineering
AI/Machine Learning
Data Science

Engineering
Software Engineering (Web, Mobile, API)
Test Automation

Cloud
Platform Engineering (AWS, Google, Azure)
Modern Workplace & Devices (G-suite, AWS EUC)

Delivery & Method
Method Coaching
Delivery Leadership
Product Ownership
Business Analysis

Managed Services
Customer Software & Data Security & Access Management Operating System
Network
Cloud Services
Hardware & Global Infrastructure

EUUSOO

Our Customers

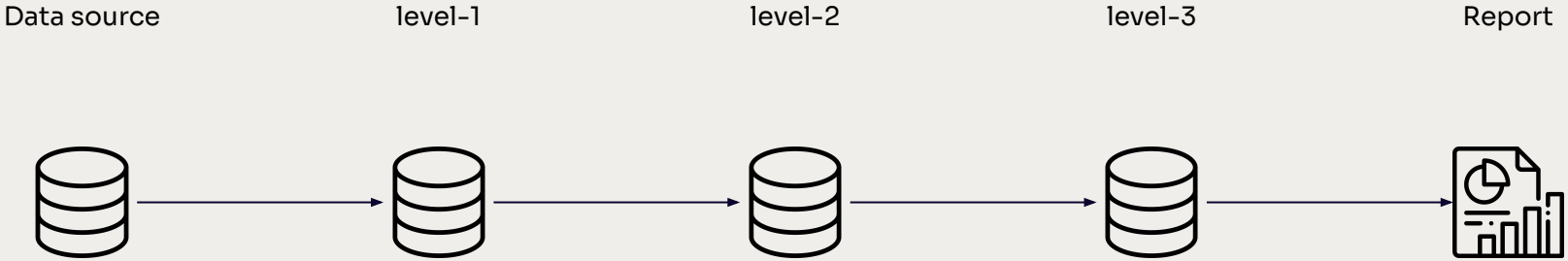
Diverse experience across a range of industries

Agenda

1. Why test?
2. Types of testing
3. How to write unit tests + demo
4. How to automate unit tests + demo

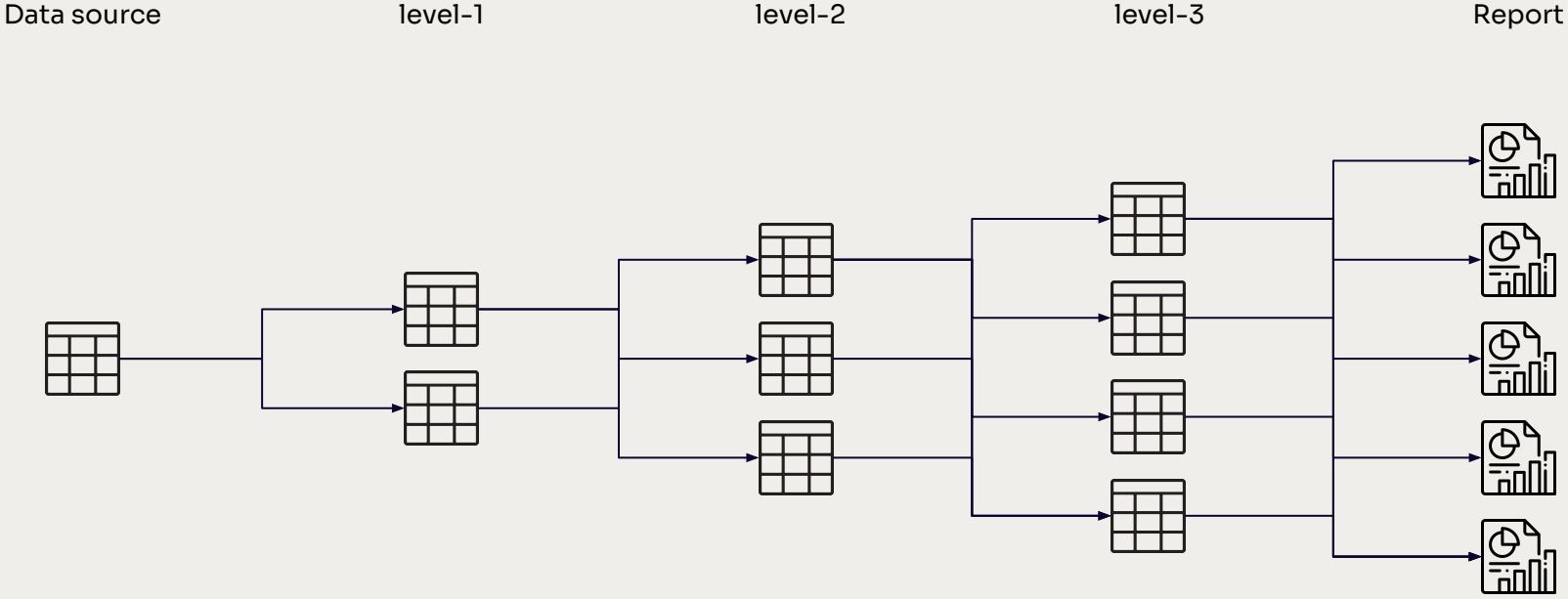
Why bother with testing?

An everyday ingestion pipeline



Why bother with testing?

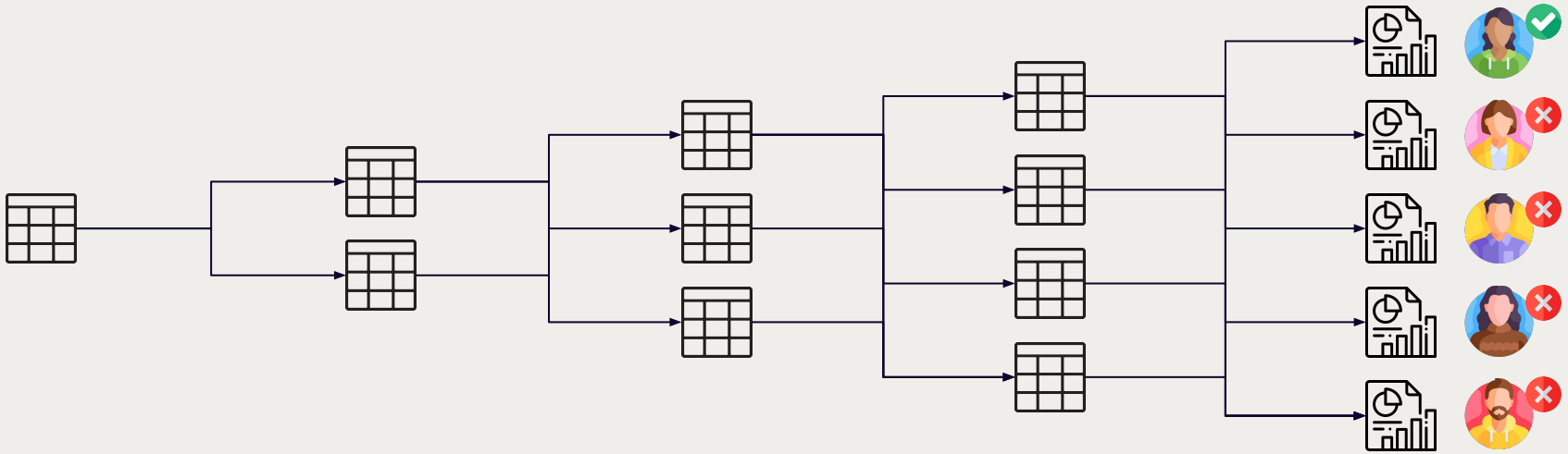
An everyday ingestion pipeline... looks more like this



Why bother with testing?

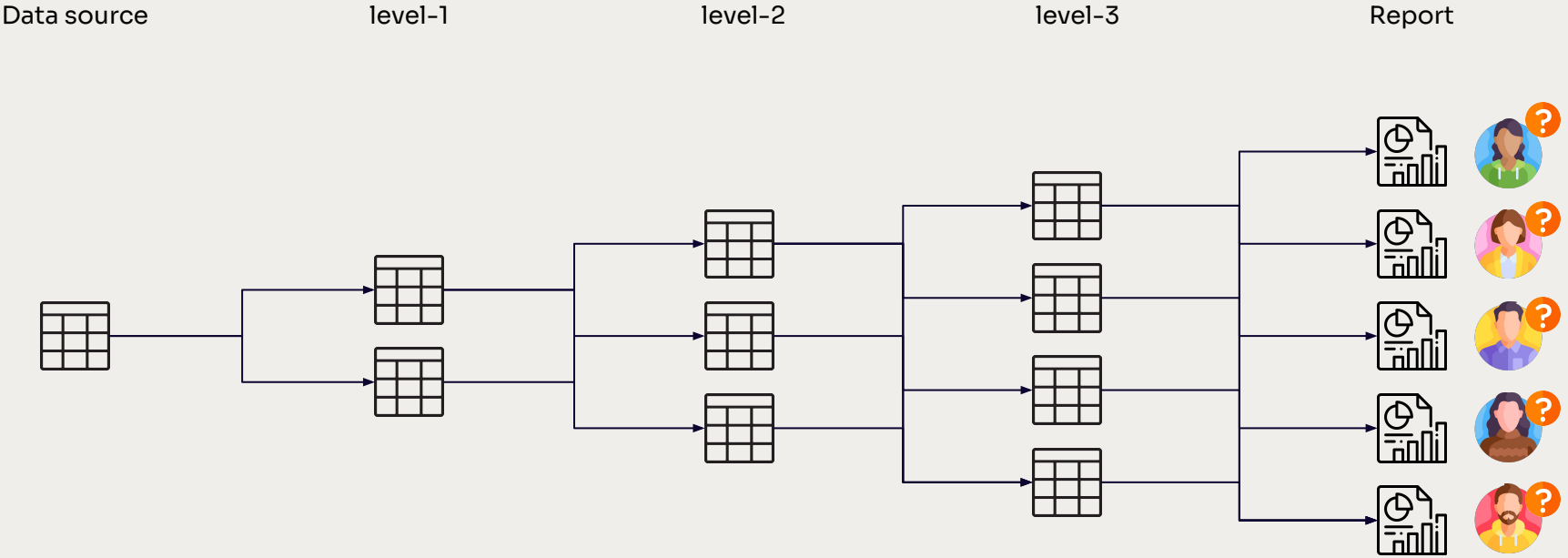
One fine day, your team made changes across the different levels, when suddenly...

Data source | level-1 | level-2 | level-3 | Report



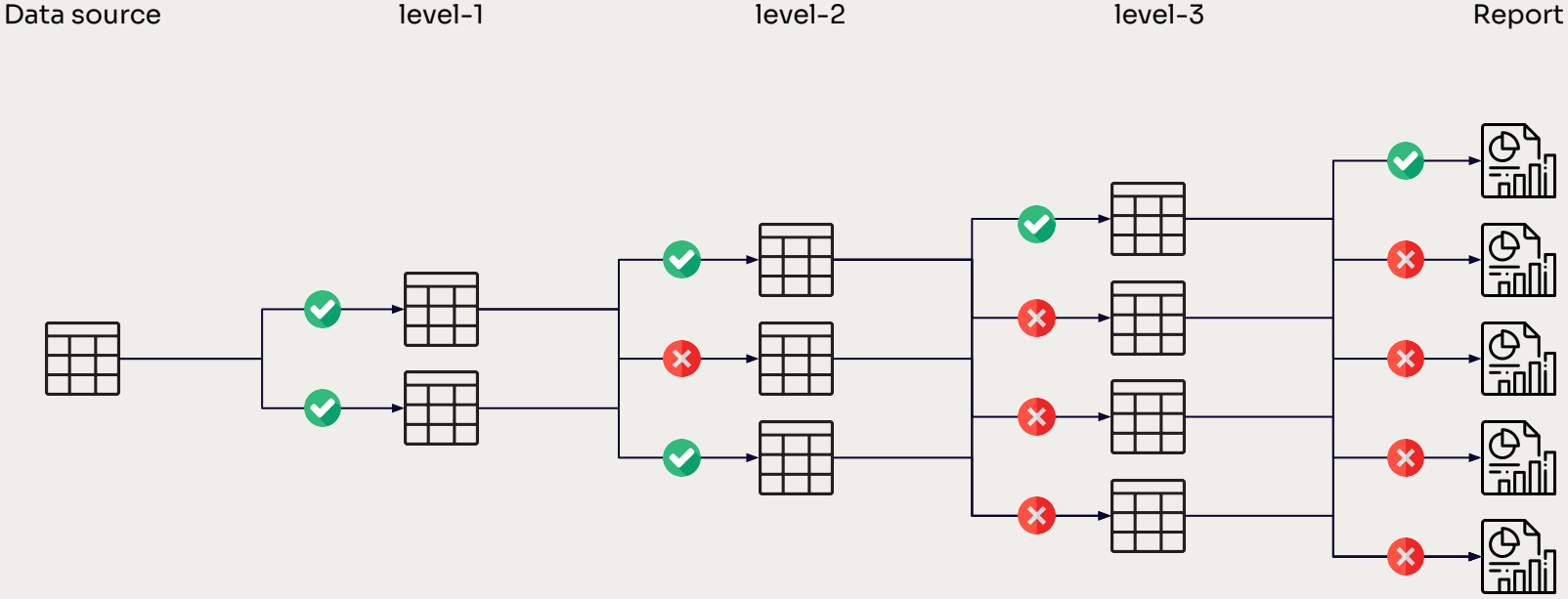
Why bother with testing?

What's worse: people made decisions without knowing there was an error with the data



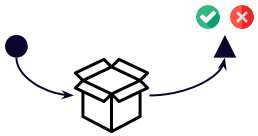
How do we fix this problem?

Testing!!!



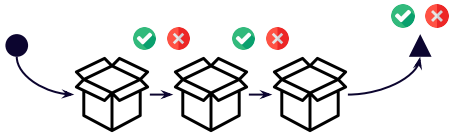
Types of testing

Unit testing



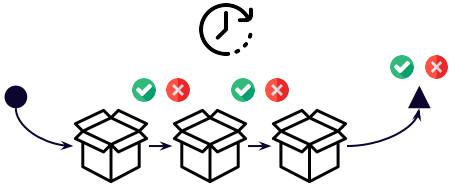
- ***PyTest*** on functions
- Triggered via CI pipeline

Integration testing



- ***PyTest*** on pipeline steps
- Triggered via CI pipeline

Quality testing

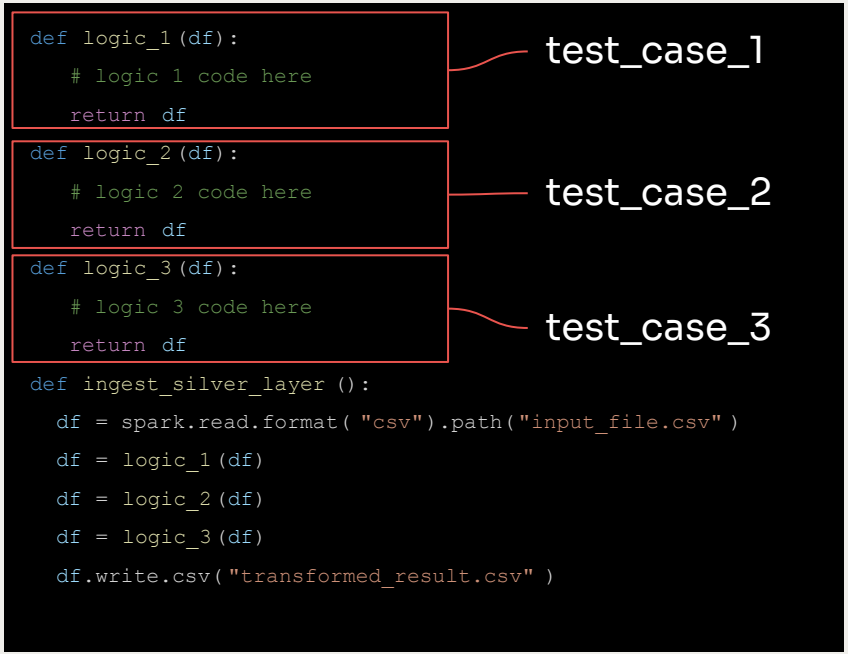


- ***Great_expectations*** on pipeline steps
- Triggered via data pipeline

EUUSOO

Types of testing

Unit testing: *“Test individual units of software”*



Trigger

Manual
Testing during development

Pull request (automated)
CI pipeline runs, test must pass before code can be merged

Tools

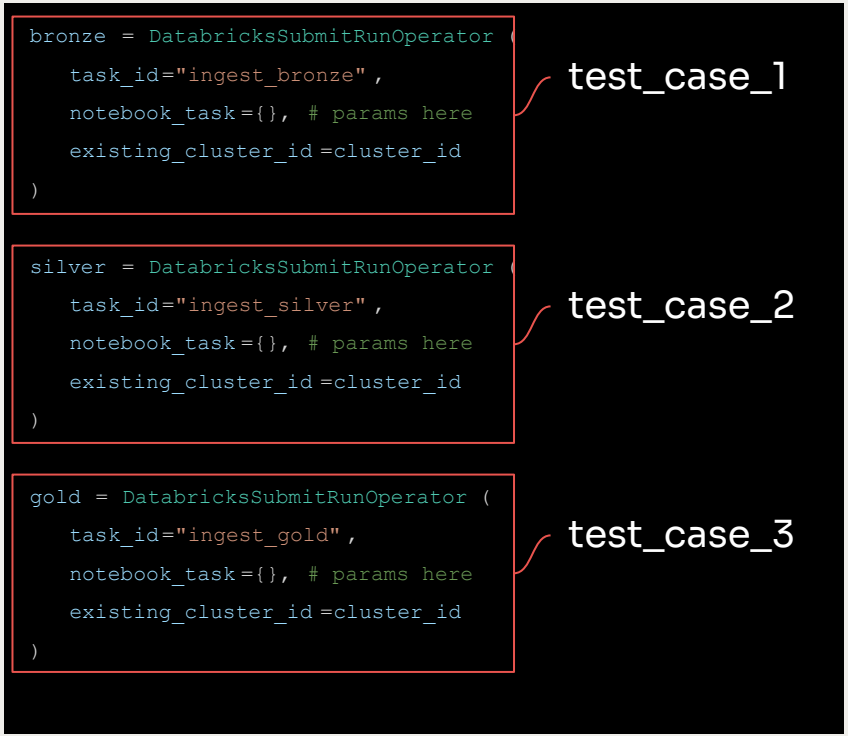
PyTest

Unittest

EUUSOO

Types of testing

Integration testing: *"individual software modules are combined and tested as a group"*



Trigger

Manual
Testing during development

Pull request (automated)
CI pipeline runs, test must pass before code can be merged

Tools

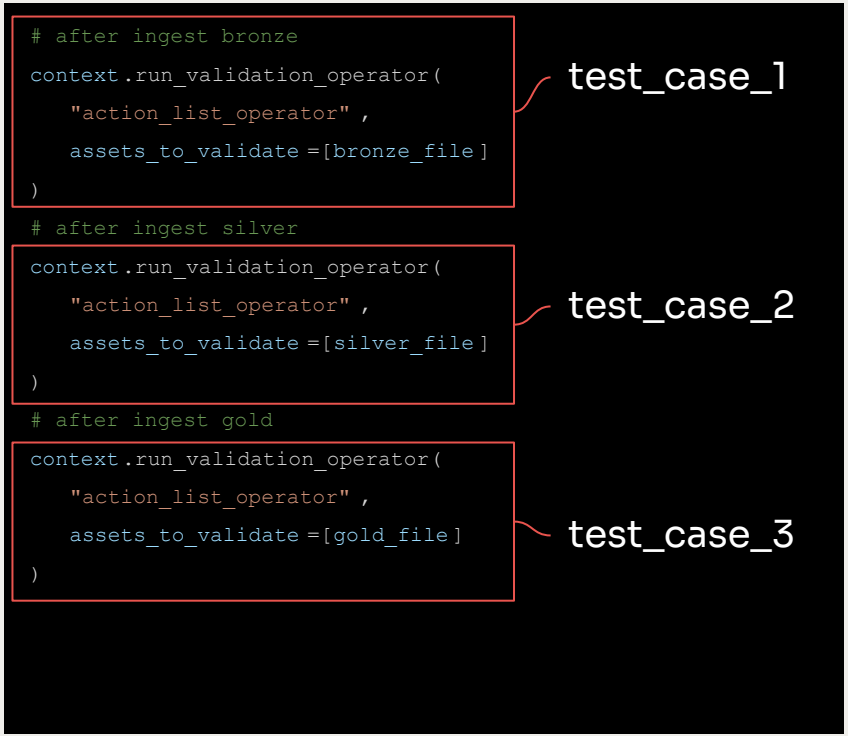
PyTest

Unittest

EUUSOO

Types of testing

Quality testing: *"Validate that actual data flowing through your end-to-end pipeline meets your expectations"*




```
# after ingest bronze
context.run_validation_operator(
    "action_list_operator",
    assets_to_validate=[bronze_file]
)
# after ingest silver
context.run_validation_operator(
    "action_list_operator",
    assets_to_validate=[silver_file]
)
# after ingest gold
context.run_validation_operator(
    "action_list_operator",
    assets_to_validate=[gold_file]
)
```

Trigger

Manual
Testing during development

When pipeline runs
Validate data flowing through each step of the pipeline against a profile as it runs

Tools

Great_expectations

Soda SQL

How to unit test

Step 1: break your code down into testable functions

```
def application_code():
    df = spark.read.format("csv").path("input_file.csv")

    # logic_1 code here
    # logic_2 code here
    # logic_3 code here

    df.write.csv("transformed_result.csv")
```

```
def logic_1(df):
    # logic 1 code here
    return df
def logic_2(df):
    # logic 2 code here
    return df
def logic_3(df):
    # logic 3 code here
    return df
def application_code():
    df = spark.read.format("csv").path("input_file.csv")
    df = logic_1(df)
    df = logic_2(df)
    df = logic_3(df)
    df.write.csv("transformed_result.csv")
```

How to unit test

Step 2: Choose a testing framework

Unittest	Pytest
`assertEqual(a, b)`	`assert a == b`
`assertNotEqual(a, b)`	`assert a != b`
`assertTrue(x)`	`assert x is True`
`assertFalse(x)`	`assert x is False`
`assertIs(a, b)`	`assert a is b`
`assertIsNot(a, b)`	`assert a is not b`
`assertIsNone(x)`	`assert x is None`
`assertIsNotNone(x)`	`assert x is not None`
`assertIn(a, b)`	`assert a in b`
`assertNotIn(a, b)`	`assert a not in b`
`assertIsInstance(a, b)`	`assert isinstance(a, b)`
`assertNotIsInstance(a, b)`	`assert not isinstance(a, b)`

PyTest

- Syntactically, more natural to write
- Supports junit xml test output format, which can be nicely rendered by your CI provider
- Can also execute python unittest

How to unit test

Step 3: write your tests - Arrange, Act, Assert

```
# ARRANGE
test_data = [
    # pump_id, start_time, end_time, litres_pumped
    (1, '2021-02-01 01:05:32', '2021-02-01 01:09:13', 24),
    (1, '2021-02-01 01:14:18', '2021-02-01 01:15:58', 11)
]
test_data = [
    {
        'pump_id': row[0],
        'start_time': row[1],
        'end_time': row[2],
        'litres_pumped': row[3]
    } for row in test_data
]
spark = SparkSession.builder.getOrCreate()
test_df = spark.createDataFrame(map(lambda x: Row(**x), test_data))
```

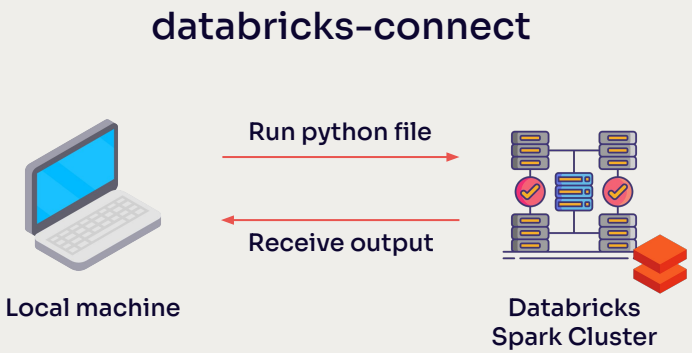
```
    # ACT
    output_df = get_litres_per_second(test_df)
    output_df_as_pd = output_df.sort('pump_id').toPandas()

    expected_output_df = pd.DataFrame([
        {
            'pump_id': 1,
            'total_duration_seconds': 800,
            'total_litres_pumped': 80,
            'avg_litres_per_second': 0.1
        }
    ])
    # ASSERT

pd.testing.assert_frame_equal(left=expected_output_df, right=output_df_as_p
d, check_exact=True)
```

How to unit test

Step 4: configure your ‘local’ environment for test execution



How to unit test

Hands-on demonstration

How to automate unit test

Continuous integration pipelines

Choose a CI provider

GitHub
GitLab
Azure DevOps
Octopus Deploy
Jenkins
circleci

Set up an agent/runner with required dependencies

Host on:

aws
Azure
Google Cloud

Manage dependencies:

docker

Write CI pipeline with trigger on PR

Write using:

Trigger on:

Pull request,
test must pass to
merge

EUUSOO

How to automate unit test

Hands-on demonstration

Thank you