

Manually testing a data pipeline

ETL AND ELT IN PYTHON



Jake Roach
Data Engineer

Testing data pipelines

Data pipelines should be thoroughly tested

- Validate that data is extracted, transformed, and loaded as expected

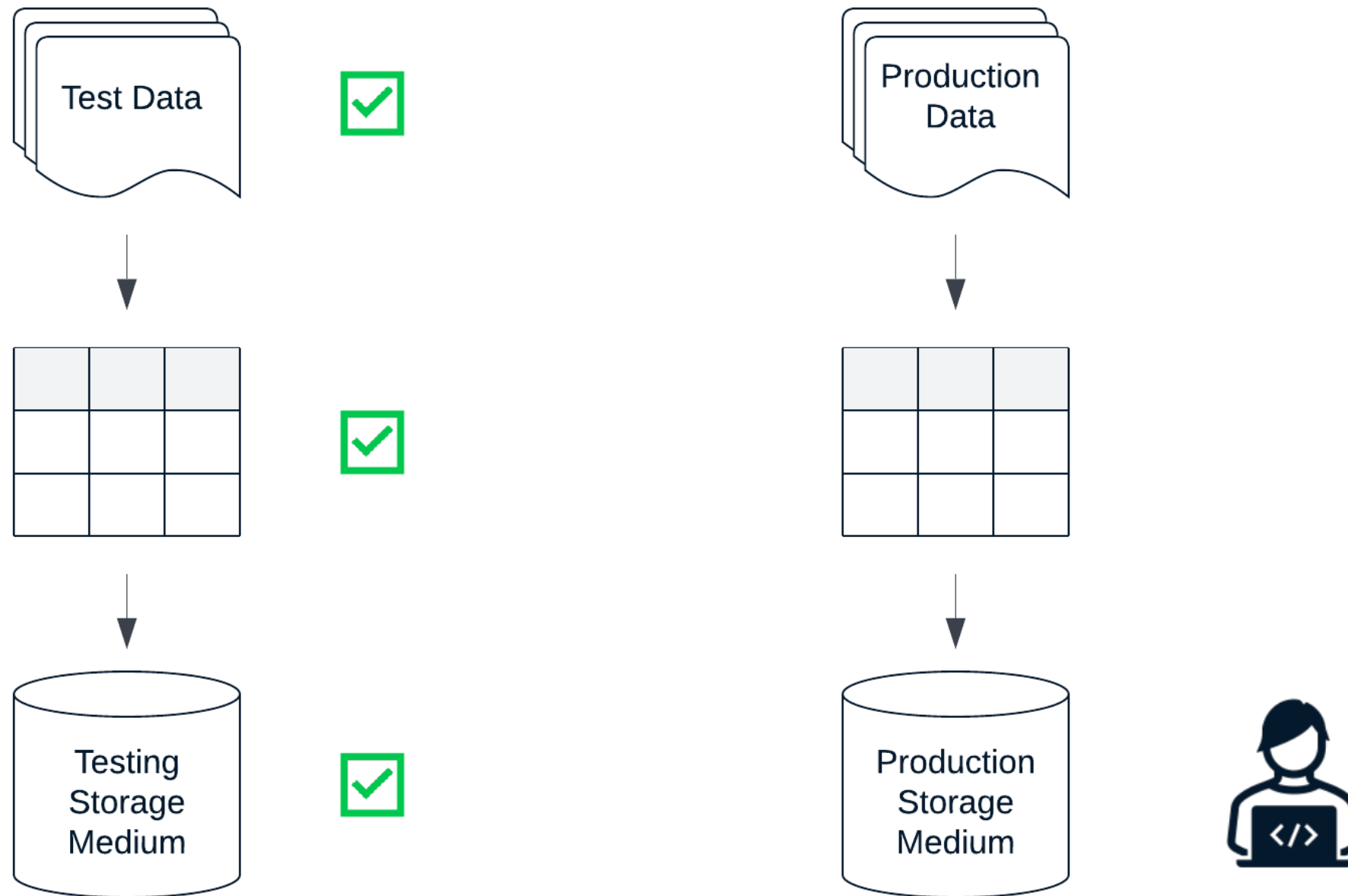
Validating pipelines' limits maintenance efforts after deployment

- Identify and fix data quality issues
- Improves data reliability

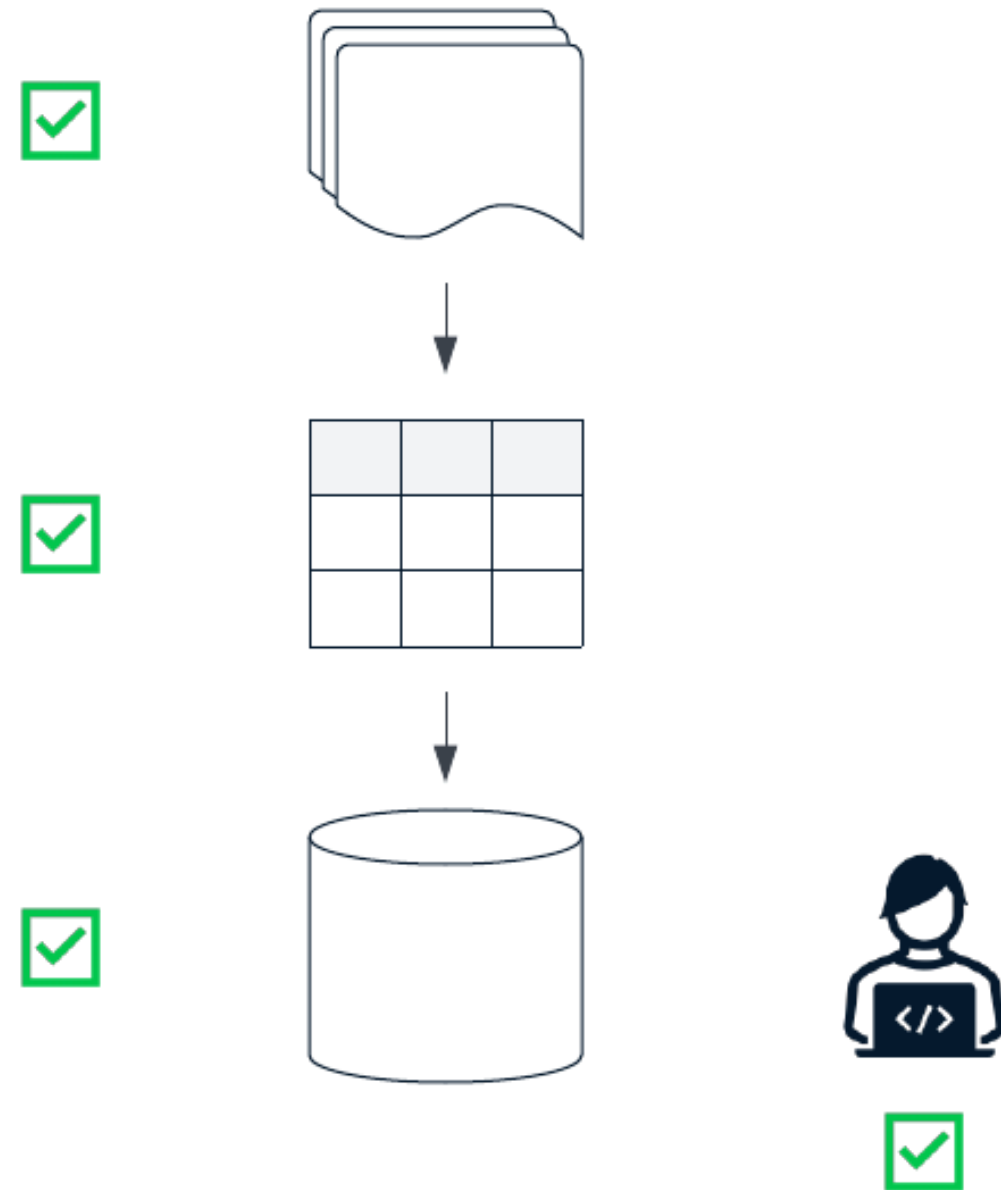
Tools and techniques to test data pipelines

- End-to-end testing
- Validating data at "checkpoints"
- Unit testing

Testing and production environments



Testing a pipeline end-to-end



End-to-end testing

- Confirm that pipeline runs on repeated attempts
- Validate data at pipeline checkpoints
- Engage in peer review, incorporate feedback
- Ensure consumer access and satisfaction with solution

Validating pipeline checkpoints

```
# Extract, transform, and load data as part of a pipeline
...

# Take a look at the data made available in a Postgres database
loaded_data = pd.read_sql("SELECT * FROM clean_stock_data", con=db_engine)
print(loaded_data.shape)
```

```
(6438, 4)
```

```
print(loaded_data.head())
```

	timestamps	volume	open	close
1997-05-15	13:30:00	1443120000	0.121875	0.097917
1997-05-16	13:30:00	294000000	0.098438	0.086458
1997-05-19	13:30:00	122136000	0.088021	0.085417

Validating DataFrames

```
# Extract, transform, and load data, as part of a pipeline
...

# Take a look at the data made available in a Postgres database
loaded_data = pd.read_sql("SELECT * FROM clean_stock_data", con=db_engine)

# Compare the two DataFrames.
print(clean_stock_data.equals(loaded_data))
```

True

Let's practice!

ETL AND ELT IN PYTHON

Unit-testing a data pipeline

ETL AND ELT IN PYTHON

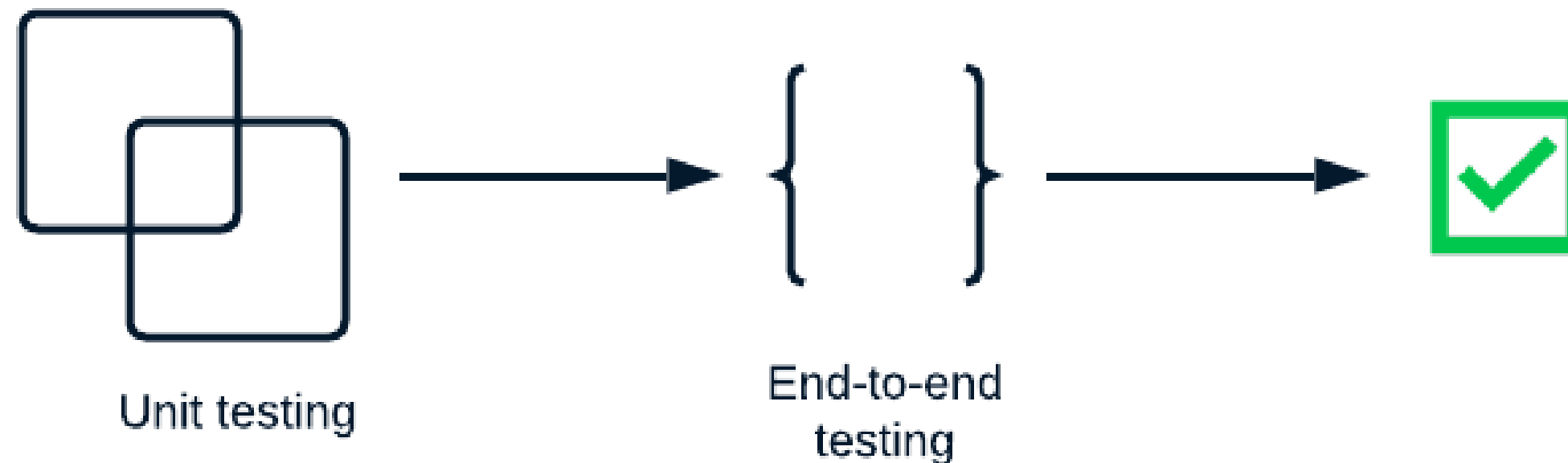


Jake Roach
Data Engineer

Validating a data pipeline with unit tests

Unit tests:

- Commonly used in software engineering workflows
- Ensure code works as expected
- Help to validate data



pytest for unit testing

```
from pipeline import extract, transform, load

# Build a unit test, asserting the type of clean_stock_data
def test_transformed_data():
    raw_stock_data = extract("raw_stock_data.csv")
    clean_stock_data = transform(raw_data)
    assert isinstance(clean_stock_data, pd.DataFrame)
```

```
> python -m pytest
```

```
test_transformed_data . [100%]
===== 1 passed in 1.17s =====
```

assert and isinstance

```
pipeline_type = "ETL"

# Check if pipeline_type is an instance of a str
isinstance(pipeline_type, str)
```

True

```
# Assert that the pipeline does indeed take value "ETL"
assert pipeline_type == "ETL"
```

```
# Combine assert and isinstance
assert isinstance(pipeline_type, str)
```

AssertionError

```
pipeline_type = "ETL"  
  
# Create an AssertionError  
assert isinstance(pipeline_type, float)
```

```
Traceback (most recent call last):  
  File "<stdin>", line 4, in <module>  
AssertionError
```

Mocking data pipeline components with fixtures

```
import pytest

@pytest.fixture()
def clean_data():
    raw_stock_data = extract("raw_stock_data.csv")
    clean_stock_data = transform(raw_data)
    return clean_stock_data
```

```
def test_transformed_data(clean_data):
    assert isinstance(clean_data, pd.DataFrame)
```

Unit testing DataFrames

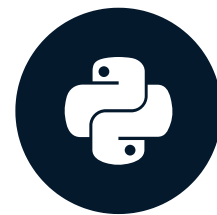
```
def test_transformed_data(clean_data):  
    # Include other assert statements here  
    ...  
  
    # Check number of columns  
    assert len(clean_data.columns) == 4  
  
    # Check the lower bound of a column  
    assert clean_data["open"].min() >= 0  
  
    # Check the range of a column by chaining statements with "and"  
    assert clean_data["open"].min() >= 0 and clean_data["open"].max() <= 1000
```

Let's practice!

ETL AND ELT IN PYTHON

Running a data pipeline in production

ETL AND ELT IN PYTHON



Jake Roach
Data Engineer

Data pipeline architecture patterns

```
# Define ETL function
...
def load(clean_data):
...

# Run the data pipeline
raw_stock_data = extract("raw_stock_data.csv")
clean_stock_data = transform(raw_stock_data)
load(clean_stock_data)
```

```
> ls
etl_pipeline.py
```

```
# Import extract, transform, and load functions
from pipeline_utils import extract, transform, load

# Run the data pipeline
raw_stock_data = extract("raw_stock_data.csv")
clean_stock_data = transform(raw_stock_data)
load(clean_stock_data)
```

```
> ls
etl_pipeline.py
pipeline_utils.py
```

Running a data pipeline end-to-end

```
import logging
from pipeline_utils import extract, transform, load

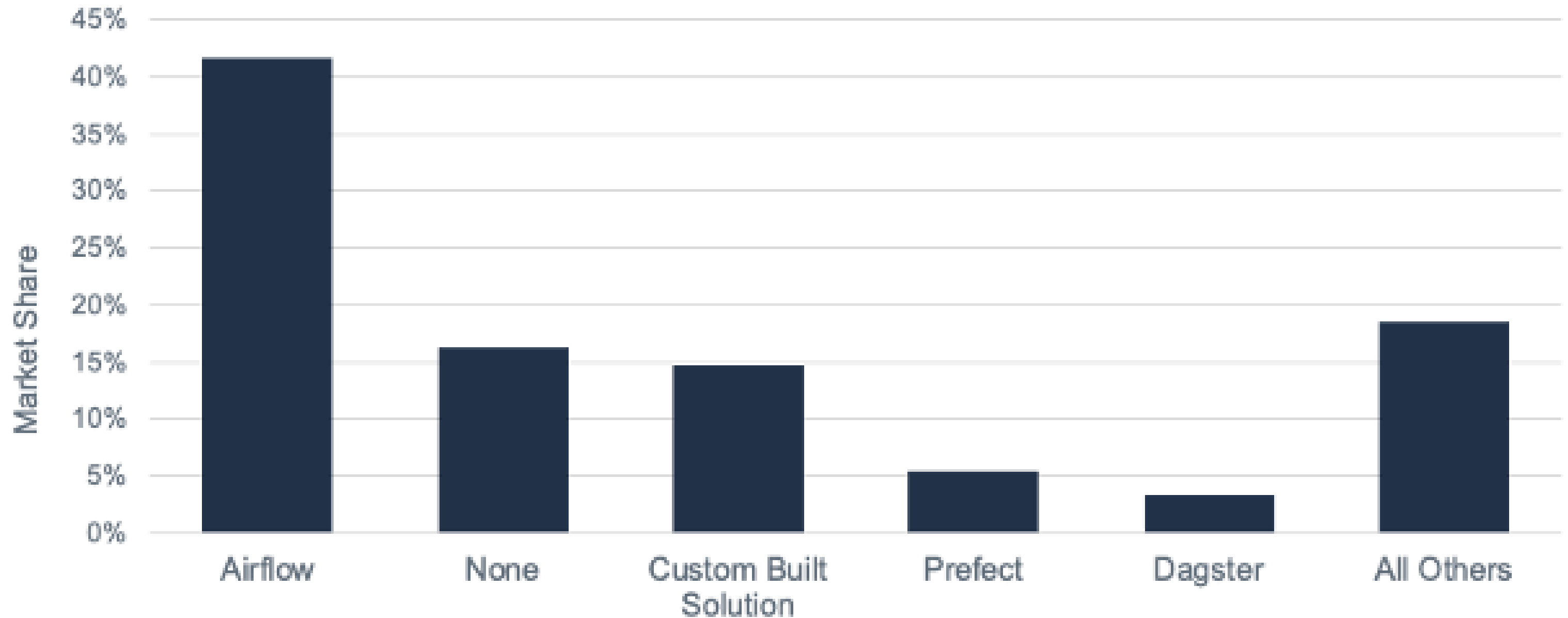
logging.basicConfig(format='%(levelname)s: %(message)s', level=logging.DEBUG)

try:
    # Extract, transform, and load data
    raw_stock_data = extract("raw_stock_data.csv")
    clean_stock_data = transform(raw_stock_data)
    load(clean_stock_data)

    logging.info("Successfully extracted, transformed and loaded data.") # Log success message

# Handle exceptions, log messages
except Exception as e:
    logging.error(f"Pipeline failed with error: {e}")
```

Orchestrating data pipelines in production



¹ https://open.substack.com/pub/seattledataguy/p/the-state-of-data-engineering-part?r=1po78c&utm_campaign=post&utm_medium=web

Let's practice!

ETL AND ELT IN PYTHON

Congratulations!

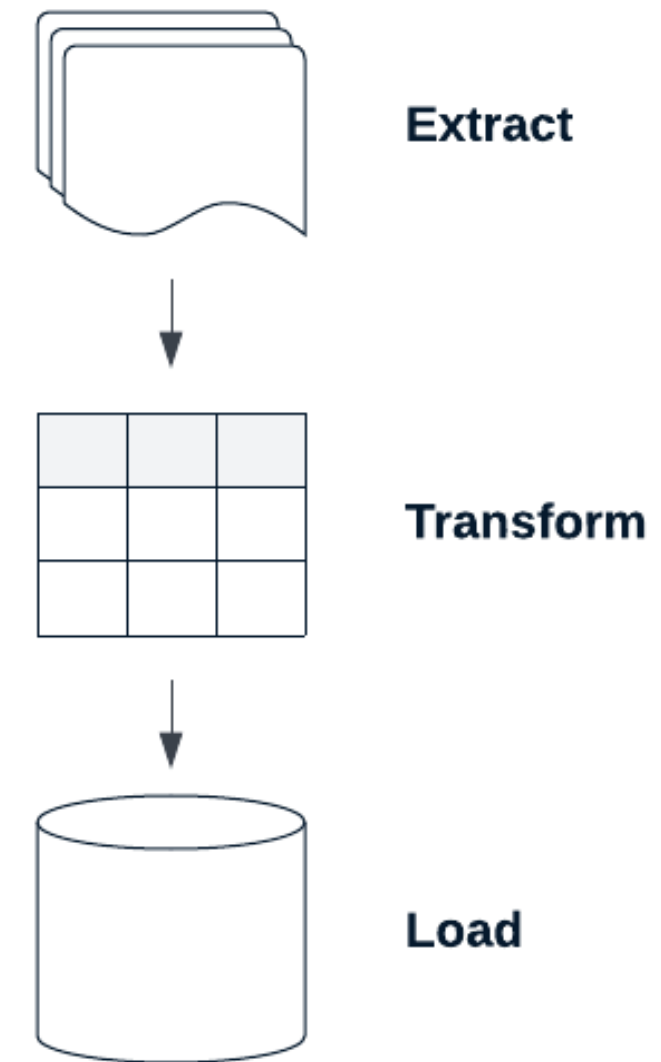
ETL AND ELT IN PYTHON



Jake Roach
Data Engineer

Designing and building data pipelines

- Designing sound data pipelines
- Extract, transform, and load architecture
- Exception handling and logging



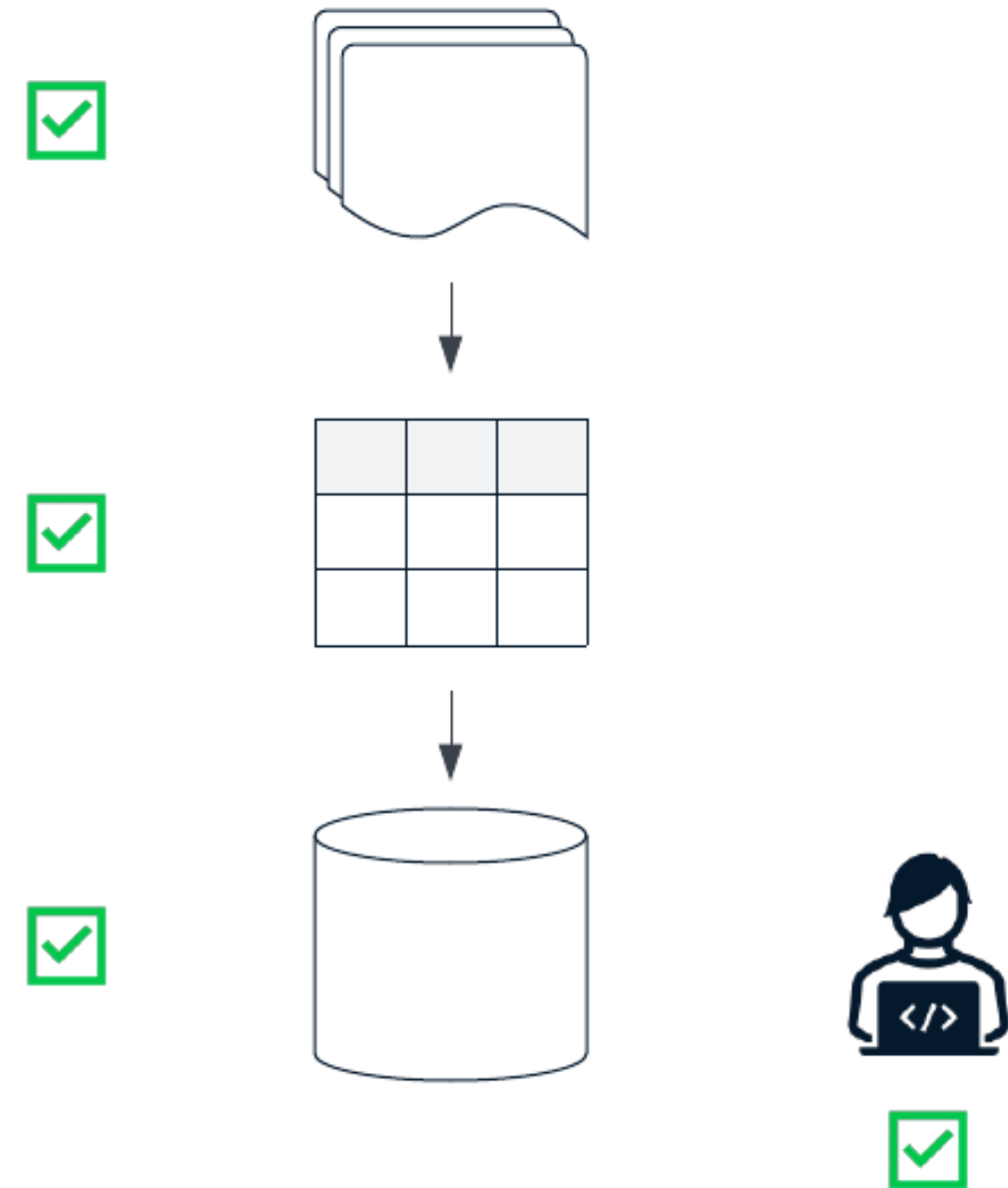
Advanced ETL techniques

- Handling nested JSON
- Advanced transformation logic
- Persisting data to SQL databases

```
{  
  "863703000": {  
    "volume": 1443120000,  
    "price": {  
      "close": 0.09791,  
      "open": 0.12187  
    }  
  },  
  ...  
}
```

Deploying and maintaining data pipelines

- Validate and test data pipelines
- Running a pipeline in a production setting
- Orchestration tools



Next steps



- *Introduction to Airflow in Python Course*
- **Data Engineer Career Track**
- **Associate Data Engineer Certification**



- Apache Airflow
- Astronomer
- Snowflake

Thank you!
ETL AND ELT IN PYTHON