Test-Driven Data Engineering

Using VS Code, Docker, python, PySpark, and AWS Glue

Presented May 24, 2022 at Open Source North

Which of you can describe the **quality** of your data pipelines using the results of an **automated testing** process?

Today's Problem Statement

Specific examples to demonstrate multiple TDDE concepts.

Data Engineers need the ability to **automate unit testing** for **data engineering pipelines** so that **test-driven** methodologies can be used.

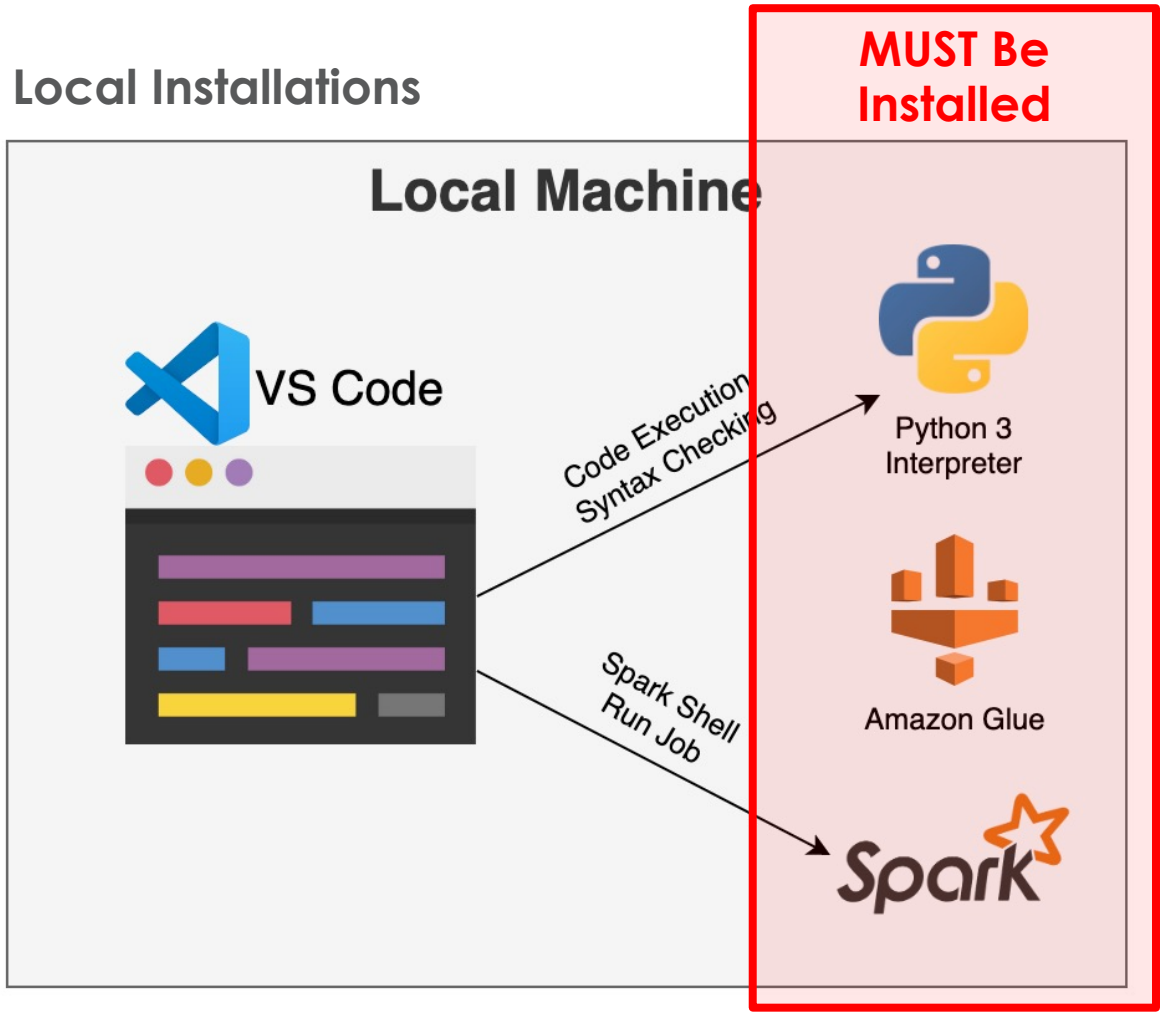
Issues this specific demo addresses:

1. Local development requiring multiple frameworks (python, PySpark, Glue)
2. Leveraging Docker for testing & development for automation
3. Using a free tool (VS Code) instead of paid (PyCharm)
4. Designing code to enable **unit** testing, reducing integration tests

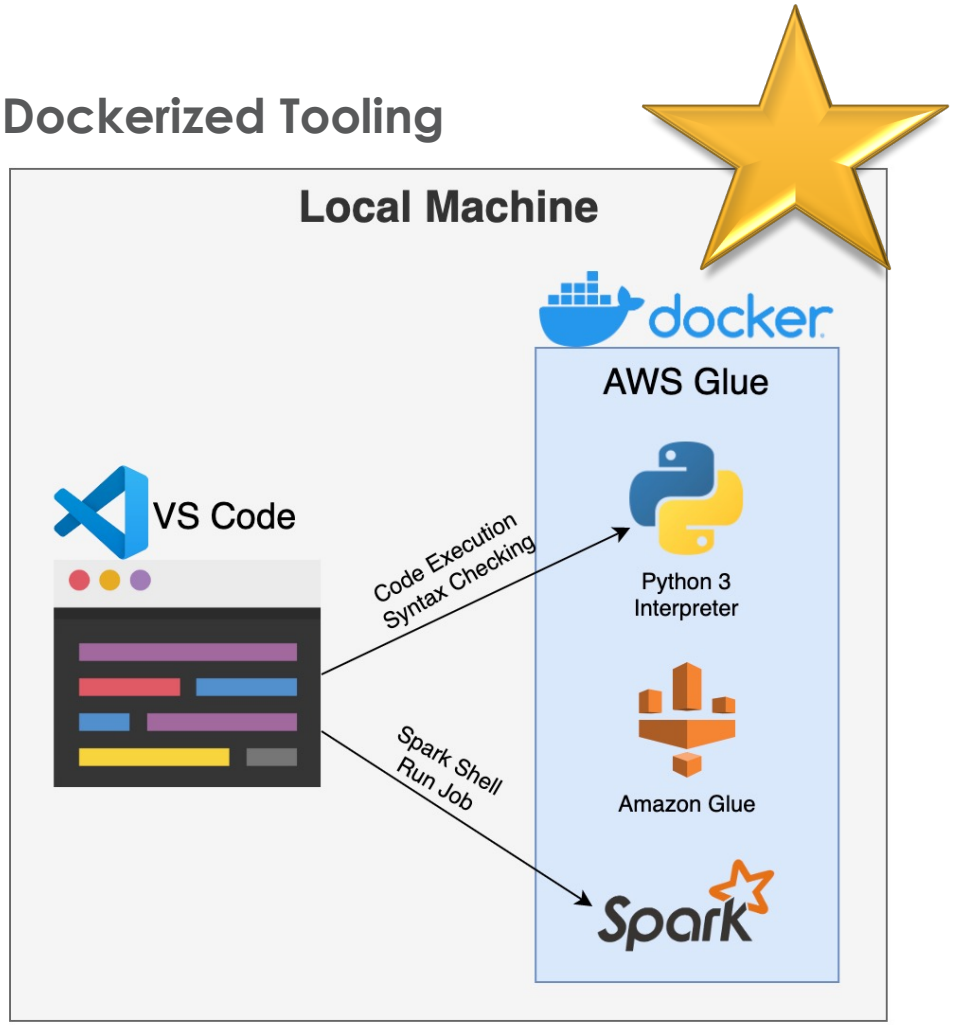
Develop Local

Pytest is used as the python testing framework for this demo.

Local Installations



Dockerized Tooling



Today's Test Automation Journey

Context

Background info and overview of the demo

TDD + python

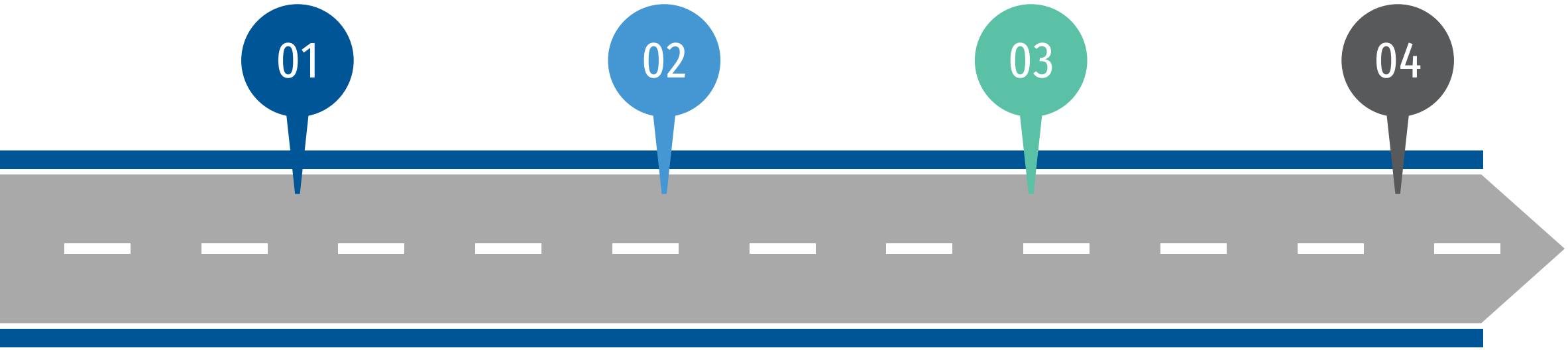
Using VS Code + Docker for python TDD

TDDE + PySpark

Test-Driven Spark UDF and DataFrame ops

TDDE + AWS Glue

Automated testing with AWS Glue



improving
It's what we do.™

Overview and Context

Background Information

Test-Driven Data Engineering
A test-first approach to building data pipelines that sometimes breaks the rules of TDD because sometimes integration tests must be used instead of unit test.

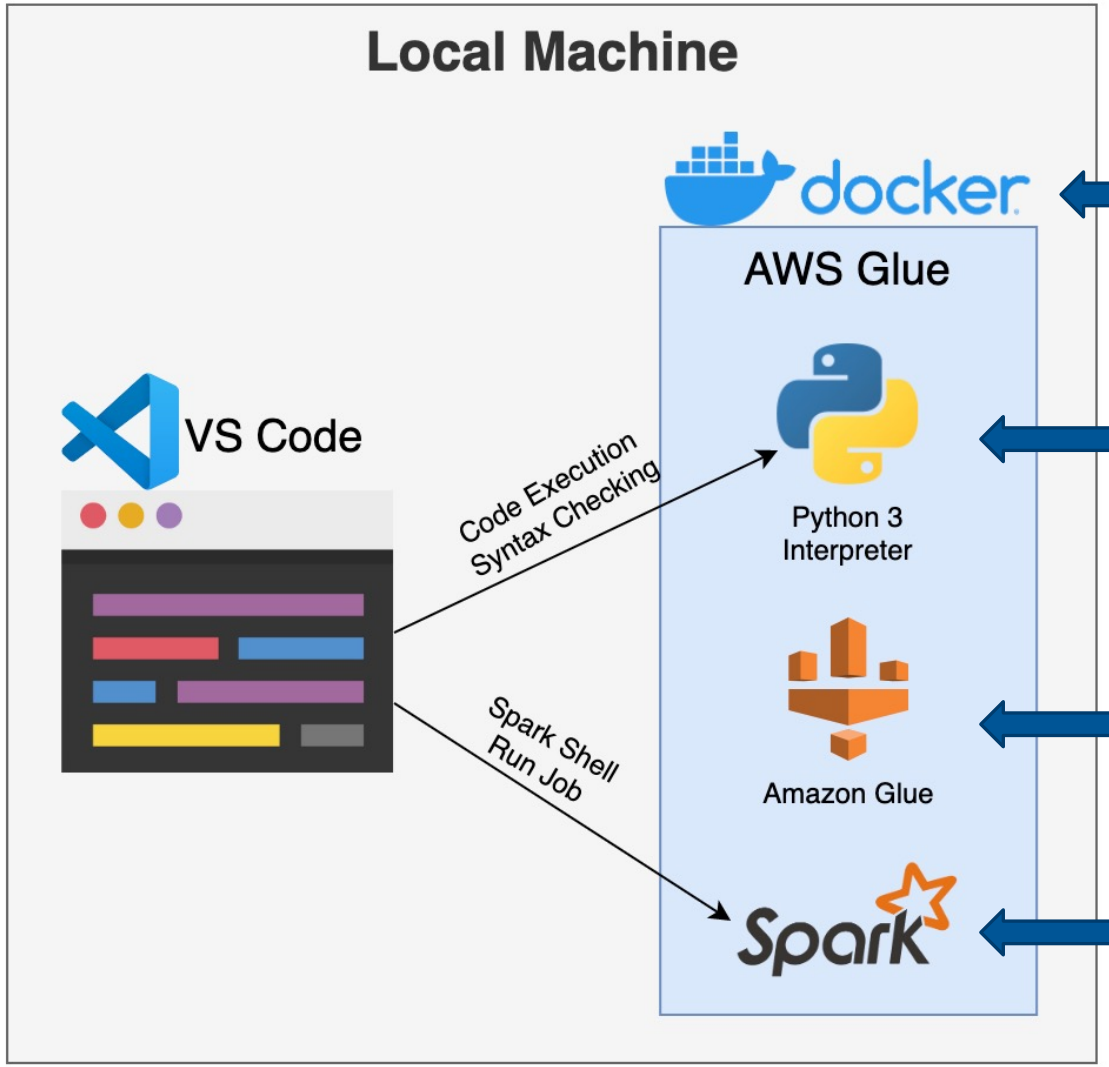
Docker
Docker provides the ability to package and run an application in a loosely isolated environment called a container.

VS Code
Visual Studio Code is a lightweight but powerful source code editor which runs on your desktop and is available for Windows, macOS and Linux.

Apache Spark
Apache Spark™ is a multi-language engine for executing data engineering, data science, and machine learning on single-node machines or clusters.

Amazon Glue
AWS Glue is a fully managed ETL (extract, transform, and load) service that makes it simple and cost-effective to categorize your data, clean it, enrich it, and move it reliably between various data stores and data streams.

Demos Used in this Presentation



Integrate VS Code and Docker to use the container as the execution environment.

TDD of a transformation function implemented in python.

TDDE of function and DynamicFrame manipulation with Amazon Glue (using the python function).

TDDE of Spark UDF and DataFrame manipulation with PySpark (using the python function).

Use Case for this Presentation

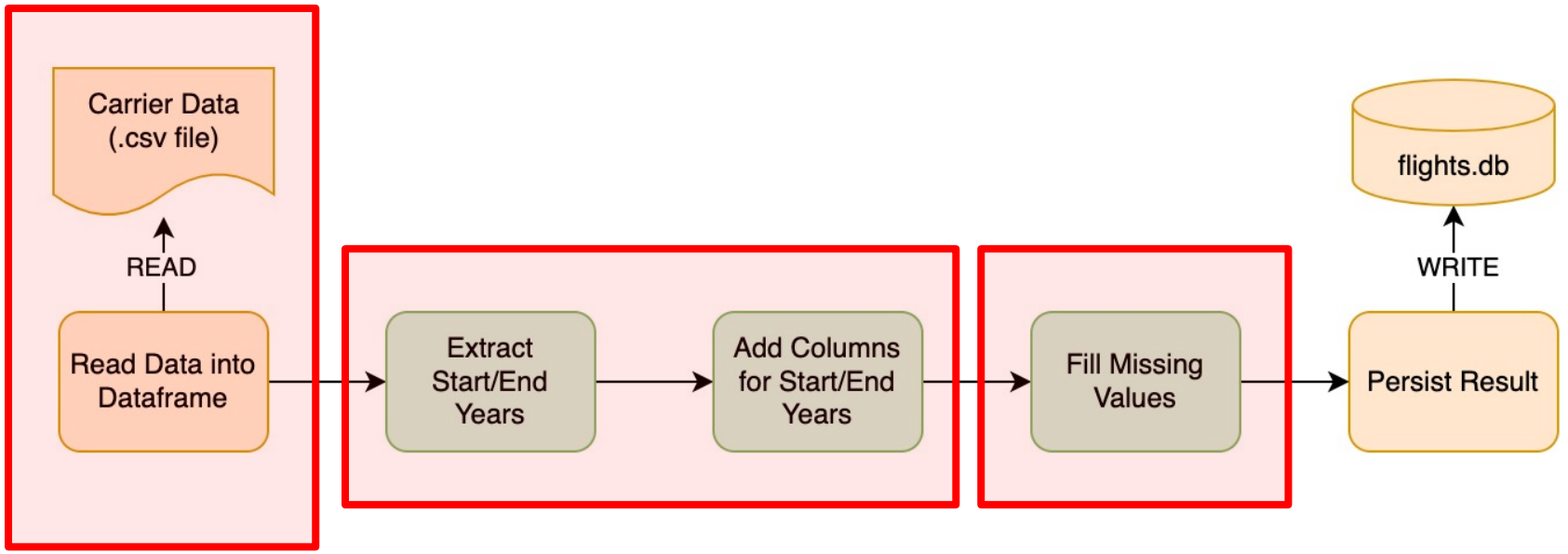
Focus on the unit testable code.

Given a csv file of carrier information:

1. Read the carriers and parse the start/end effective years
2. Add start/end years as new columns
3. Replace null start years with the value 1900
4. Replace null end years with the value 9999
5. Invalid ranges should use [-1, -1] for [start, end]

Sample Carrier String Values:

- Trans-American Airlines(- 2010)
- American Airlines Intl (2011 - 2016)
- American Airlines (2016 -)
- Sun Country (-)



Use Case for this Presentation

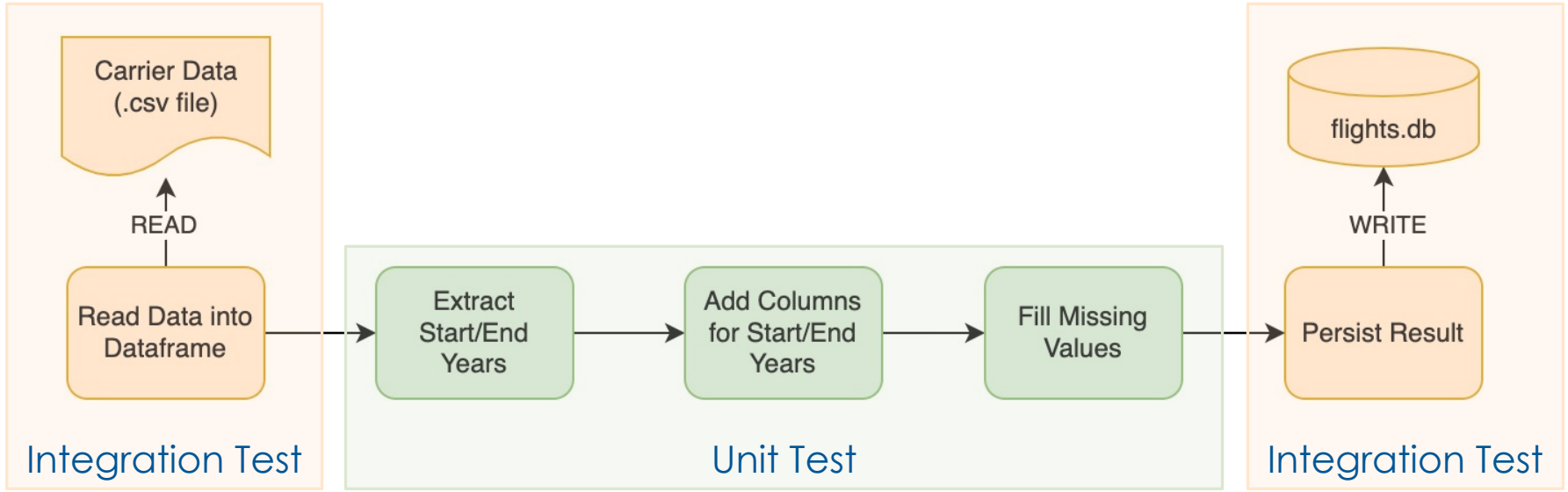
Focus on the unit testable code.

Given a csv file of carrier information:

1. Read the carriers and parse the start/end effective years
2. Add start/end years as new columns
3. Replace null start years with the value 1900
4. Replace null end years with the value 9999
5. Invalid ranges should use [-1, -1] for [start, end]

Sample Carrier String Values:

- Trans-American Airlines(- 2010)
- American Airlines Intl (2011 - 2016)
- American Airlines (2016 -)
- Sun Country (-)



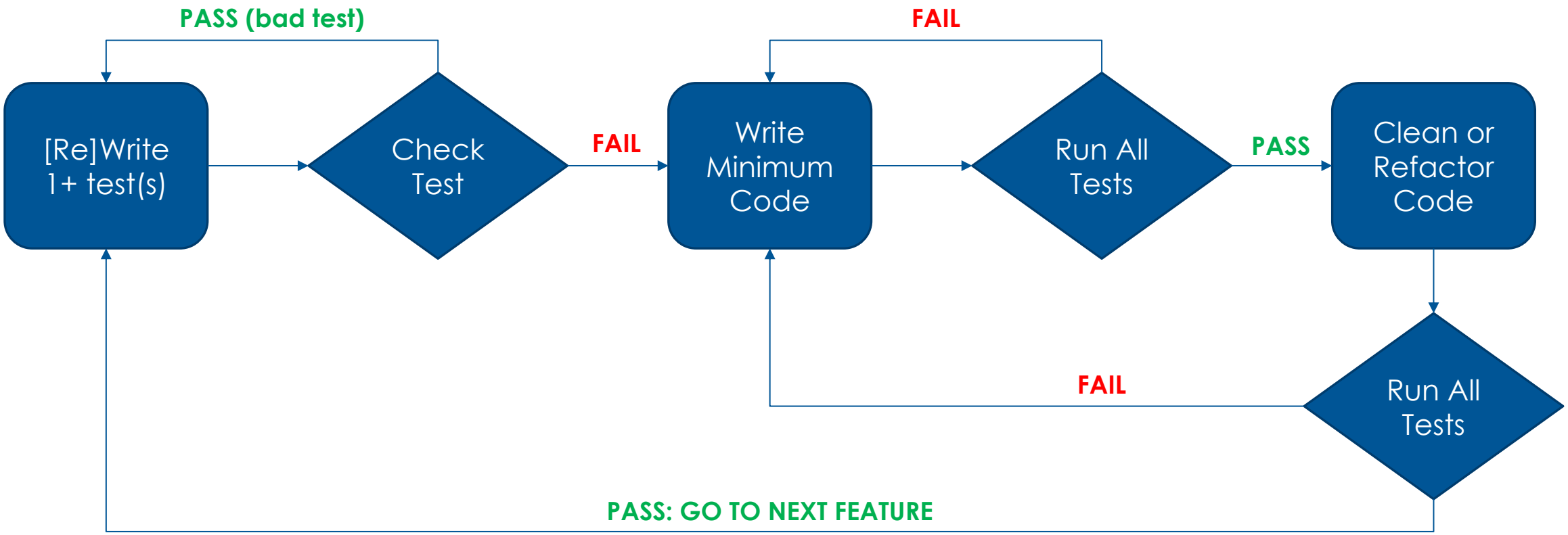
improving
It's what we do.™

Test Cases

Test planning requires creation of test scenarios and expected output given an input value (input comes from a column)

Id	Test Case	Sample	Validity	Start	End
1	Two years	Carrier (2016 - 2020)	Valid	2016	2020
2	Missing end year	Carrier (2016 -)	Valid	2016	9999
3	Missing start year	Carrier (- 2016)	Valid	1900	2016
4	Two years, extra parenthesis	Carrier (carrier) (2010 - 2016)	Valid	2010	2016
5	Missing both years, but valid string	Carrier (-)	Valid	1900	9999
6	Missing parenthesis	Carrier	Invalid	-1	-1
7	Missing right parenthesis	Carrier (2016 – 2022	Invalid	-1	-1
8	Missing left parenthesis	Carrier 2016 - 2022)	Invalid	-1	-1

TDD Methodology

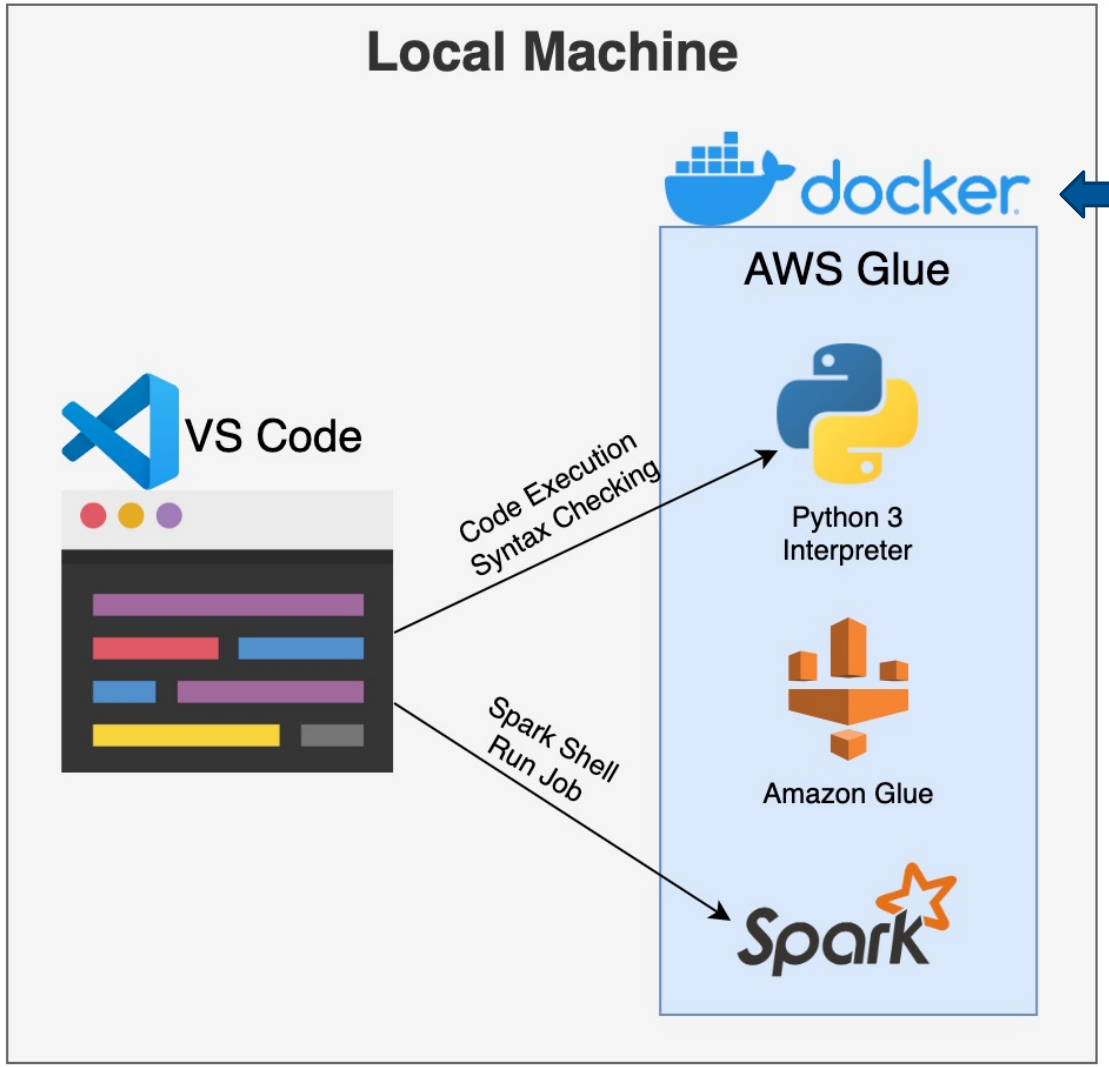


improving
It's what we do.™

Test-Driven Development with python

Isolate python-only logic into python functions

Demos Used in this Presentation

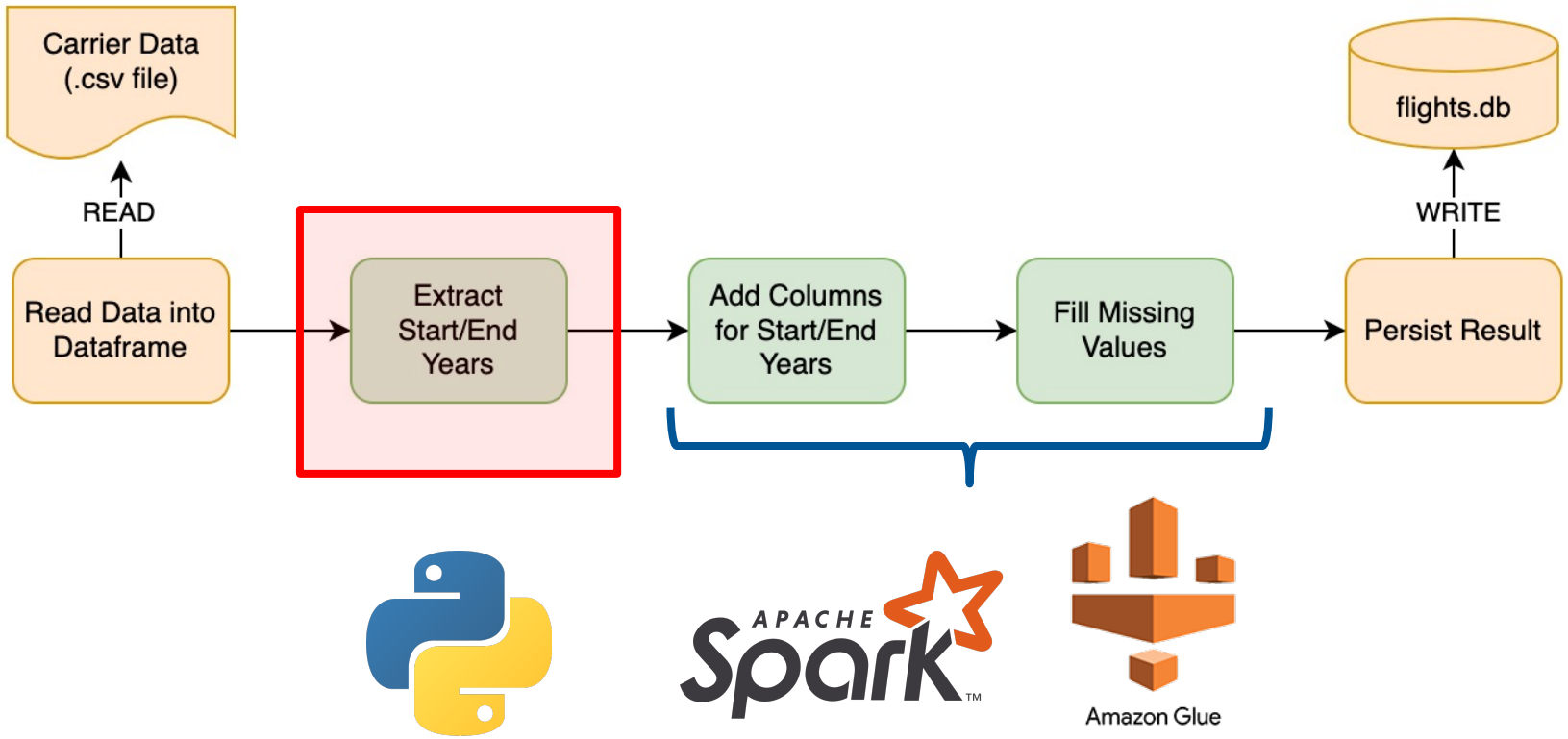


Integrate VS Code and Docker to use the container as the execution environment.

improving
It's what we do.™

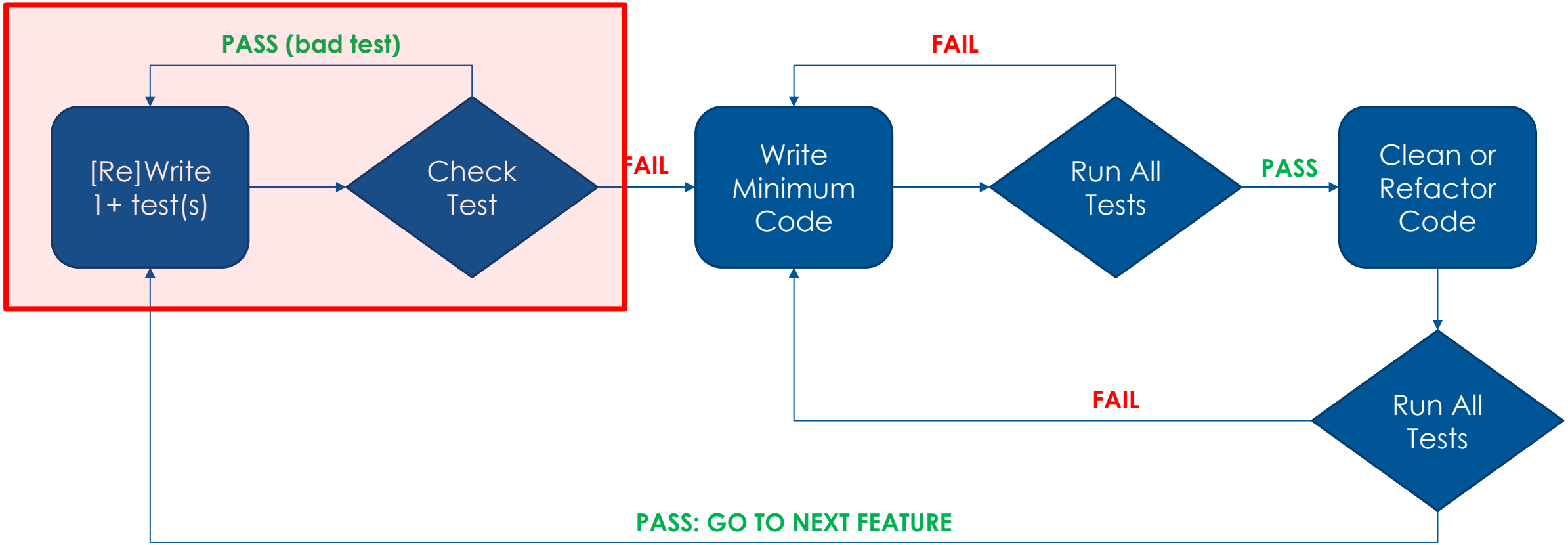
Implement the Python Logic

Test Code File: tests/test_flight_utilities.py **Code file:** flight_utilities.py



TDD: Write the Tests

Code stubs are required so the test can run dummy code.



improving
It's what we do.™

Write the Stub(s) and Test(s)

Test Methodology: One coded test per test case. **Tests Code File:** test_flight_utilities.py, **Code File:** flight_utilities.py

Function Stub

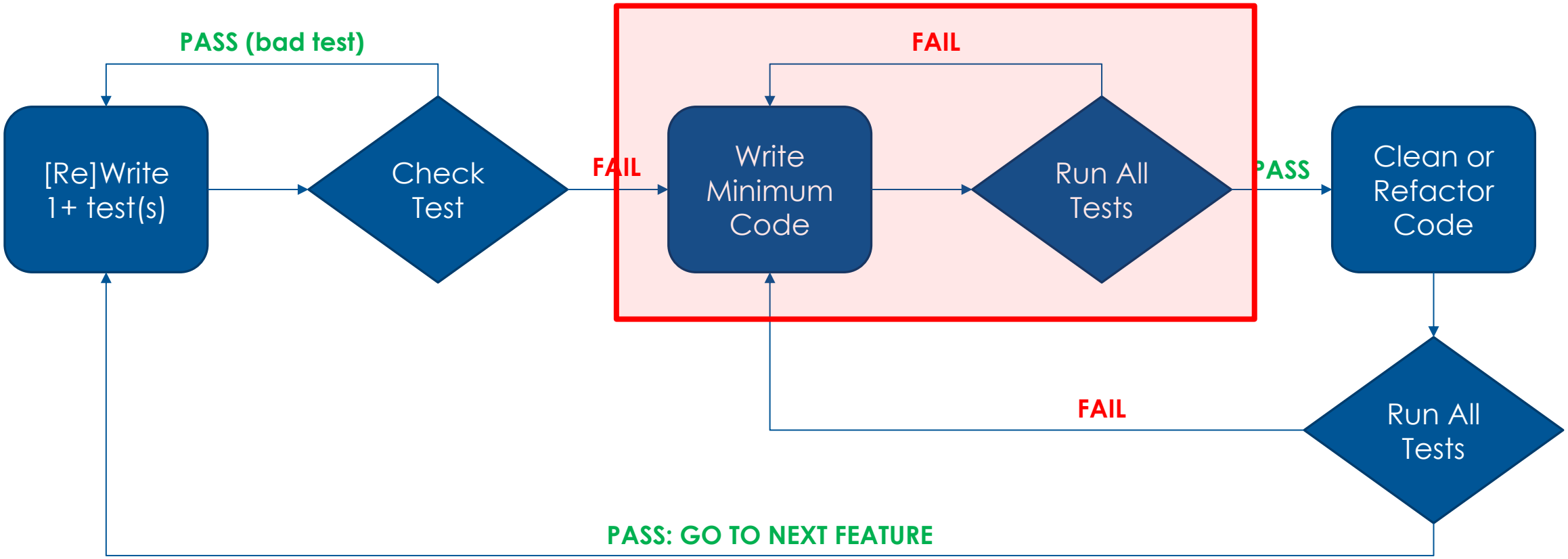
```
def get_year_range(desc_str: object):
     return None
```

Sample Test Function

```
def test_get_year_range_2016_2020():
  # given
  sut_input = 'Carrier (2016 - 2020)'
  expected = [2016, 2020]

  # when
  actual = sut.get_year_range(sut_input)

  # then
  assert expected == actual
```

TDD: Write the Implementation Code

Demo: Implement the function code and re-run the tests.



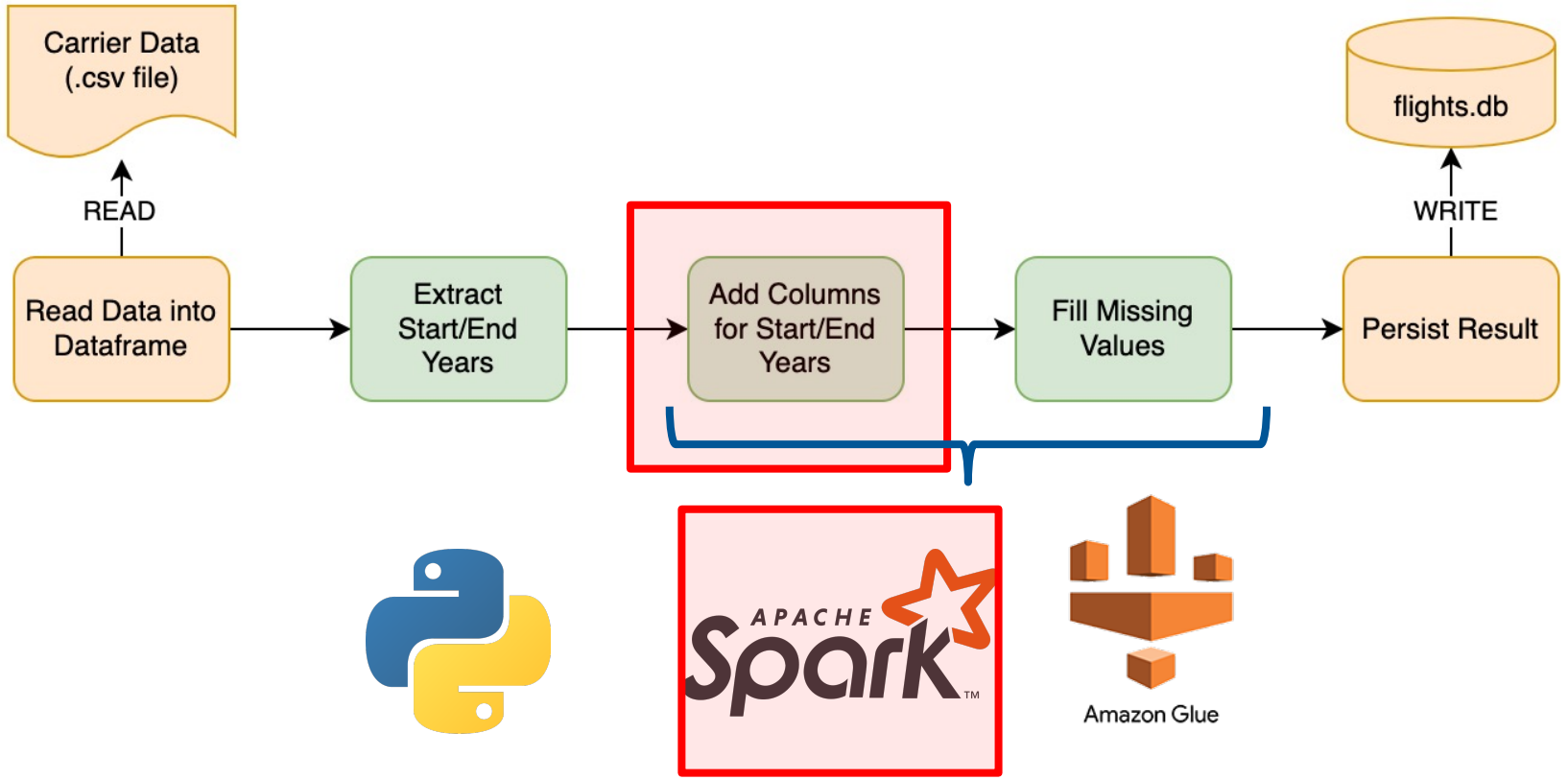
improving
It's what we do.™

Test-Driven Data Engineering with PySpark

Implement the DataFrame manipulations with PySpark

Implement the PySpark Transformations (UDF)

Tests Code File: tests/test_get_effective_year_udfs.py, **Code file:** carrier_transforms.py



UDF Test: Test Cases with Similar Output

Group the valid and invalid test cases together.

What is a UDF?

User-Defined Functions (UDFs) are user-programmable routines that act on one row.

Input DataFrame

expected	input_str
2016	Carrier (2016 - 2020)
2016	Carrier (2016 -)
1900	Carrier (- 2016)
2010	Carrier (carrier) (2010 - 2016)
1900	Carrier (-)

Output DataFrame

expected	input_str	actual
2016	Carrier (2016 - 2020)	???
2016	Carrier (2016 -)	???
1900	Carrier (- 2016)	???
2010	Carrier (carrier) (2010 - 2016)	???
1900	Carrier (-)	???

UDF Test: All Test Cases at Once

Group all test cases together.

Input DataFrame

expected	Sample
2016	Carrier (2016 - 2020)
2016	Carrier (2016 -)
1900	Carrier (- 2016)
2010	Carrier (carrier) (2010 - 2016)
1900	Carrier (-)
-1	Carrier
-1	Carrier (2016 – 2022
-1	Carrier 2016 - 2022)

Output DataFrame

expected	Sample	actual
2020	Carrier (2016 - 2020)	???
9999	Carrier (2016 -)	???
2016	Carrier (- 2016)	???
2016	Carrier (carrier) (2010 - 2016)	???
9999	Carrier (-)	???
-1	Carrier	???
-1	Carrier (2016 – 2022	???
-1	Carrier 2016 - 2022)	???

Two Sample Test Methods

By Similarity of Output Result

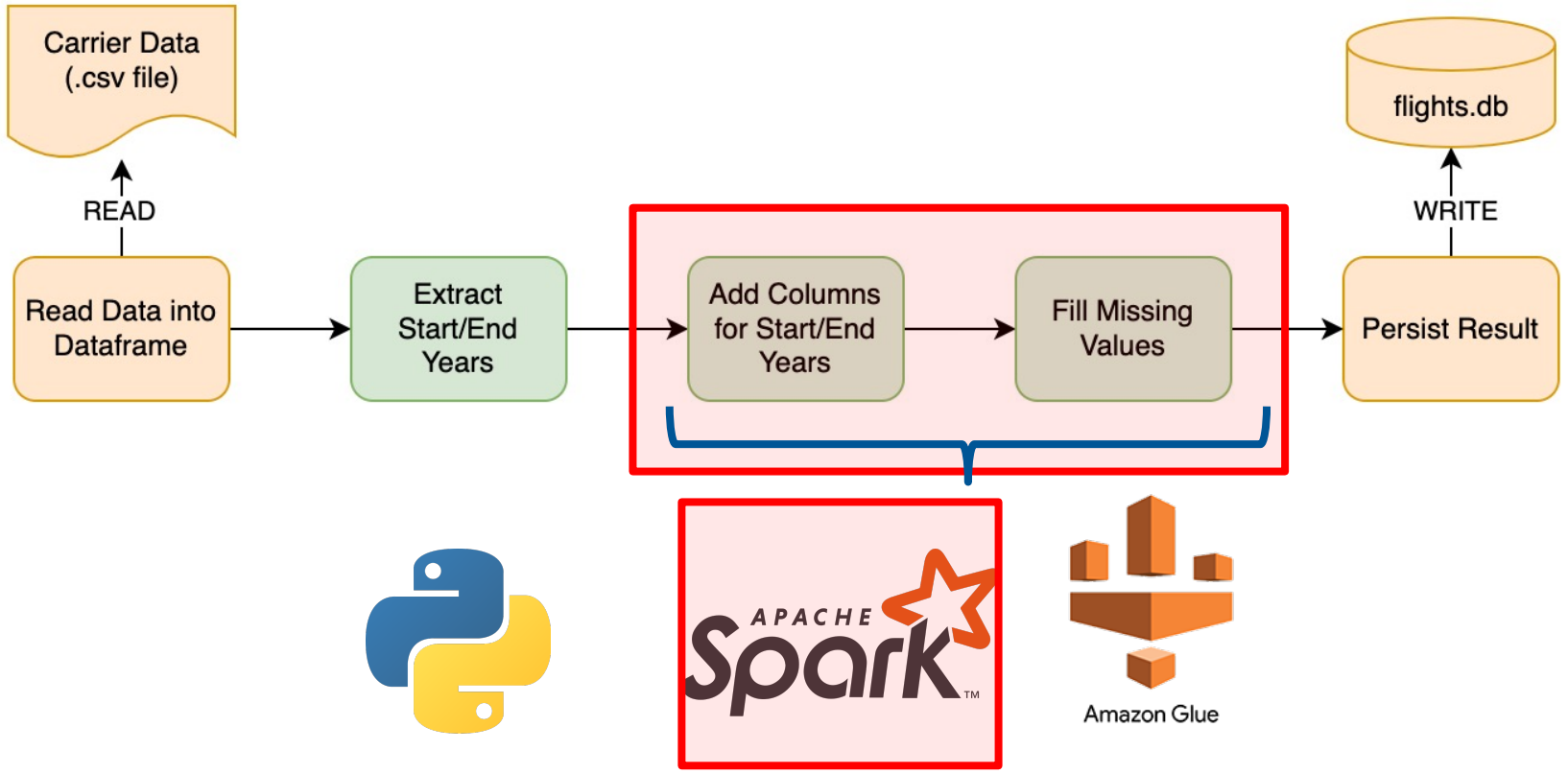
```
def test_get_start_year_valid_ranges(spark_session):
  # given
  sut_df: DataFrame = spark_session.createDataFrame(
    [
      (2016, 'Carrier (2016 – 2020)'),
      (2016,'Carrier (2016 – )'),
      (None, 'Carrier ( – 2016)'),
      (2010, 'Carrier (carrier) (2010 – 2016)'),
      (None, 'Carrier ( – )')
    ],
    ["expected", "input_str"])

  # when
  result_df = sut_df.\
    withColumn("actual", ct.get_start_year(col("input_str"))).\
      select('expected', 'actual')

  # then
  failure_df = result_df.where(col("actual") != col("expected"))
  failure_df.show()
  assert(failure_df.rdd.isEmpty())
```

All Results at Once

```
def test_get_end_year_all_formats(spark_session):
  # given
  sut_df: DataFrame = spark_session.createDataFrame(
    [
      (2020, 'Carrier (2016 – 2020)'),
      (None,'Carrier (2016 – )'),
      (2016, 'Carrier ( – 2016)'),
      (2016, 'Carrier (carrier) (2010 – 2016)'),
      (None, 'Carrier ( – )'),
      (-1, 'Carrier'),
      (-1, 'Carrier (2016 – 2022'),
      (-1, 'Carrier 2016 – 2022)')
    ],
    ["expected", "input_str"])

  # when
  result_df = sut_df.\
    withColumn("actual", ct.get_end_year(col("input_str"))).\
      select('expected', 'actual')

  # then
  failure_df = result_df.where(col("actual") != col("expected"))
  failure_df.show()
  assert(failure_df.rdd.isEmpty())
```

Implement the Spark DataFrame Transformations

Tests Code File: tests/test_carrier_transforms_spark, **Code file:** carrier_transforms.py



DataFrame Test: Test Cases with Similar Input

Group the test cases by inputs that have similar attributes.

Input DataFrame

code	description
C01	Carrier (2016 - 2020)
C04	Carrier (carrier) (2010 - 2016)

processYearRange()

Actual Output DataFrame

code	description	effective_ start_ year	effective_ end_ year
C01	Carrier (2016 - 2020)	???	???
C04	Carrier (carrier) (2010 - 2016)	???	???

Expected Output DataFrame

code	description	effective_ start_ year	effective_ end_ year
C01	Carrier (2016 - 2020)	2016	2020
C04	Carrier (carrier) (2010 - 2016)	2010	2016

Compare for Equality

Reusable Test Resources: Pytest Fixtures

Reusable resources available for tests -- https://docs.pytest.org/en/6.2.x/fixture.html

Session Fixture – for all tests

Used for resources that need to be used across all tests in a test run. e.g. Spark Session

```
@pytest.fixture(scope="session")
def spark_session(request) -> SparkSession:
  """Fixture for creating a spark context."""

  spark = (SparkSession
    .builder
    .master('local[*]')
    .appName("pytest spark_session")
    .enableHiveSupport()
    .getOrCreate())
  request.addfinalizer(lambda: spark.stop())

  quiet_py4j()

  return spark
```

Module Fixture – for a specific test module

Used for functions and resources that can be reused within a module. e.g. schema definitions

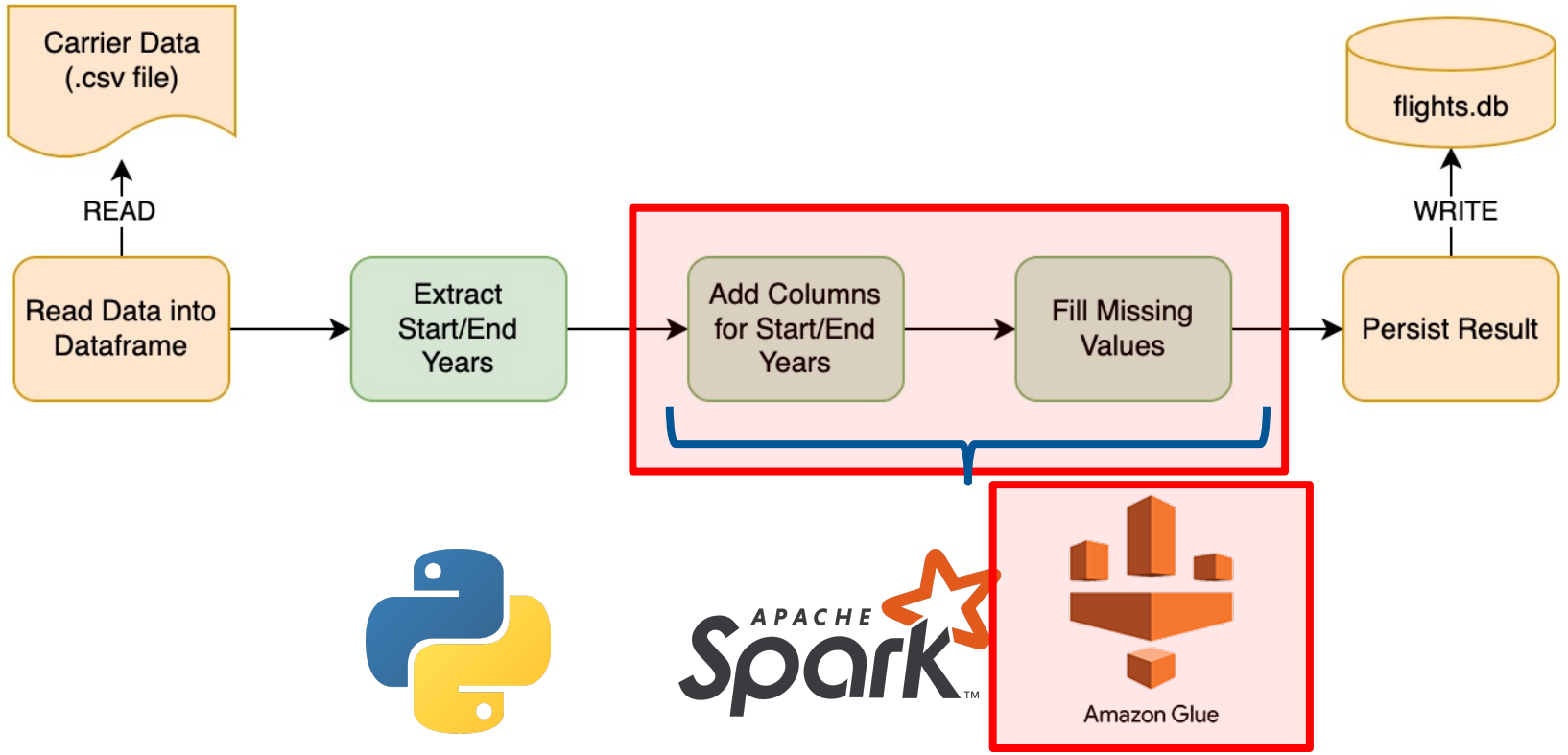
```
@pytest.fixture(scope="module")
def carrier_input_schema() -> StructType:
  return StructType([
    StructField("code", StringType(), False),
    StructField("description", StringType(), False)
  ])


@pytest.fixture(scope="module")
def carrier_output_schema() -> StructType:
  return StructType([
    StructField("code", StringType(), False),
    StructField("description", StringType(), False),
    StructField("effective_start_year", IntegerType(), True),
    StructField("effective_end_year", IntegerType(), True)
    ])
```

Test-Driven Data Engineering with AWS Glue

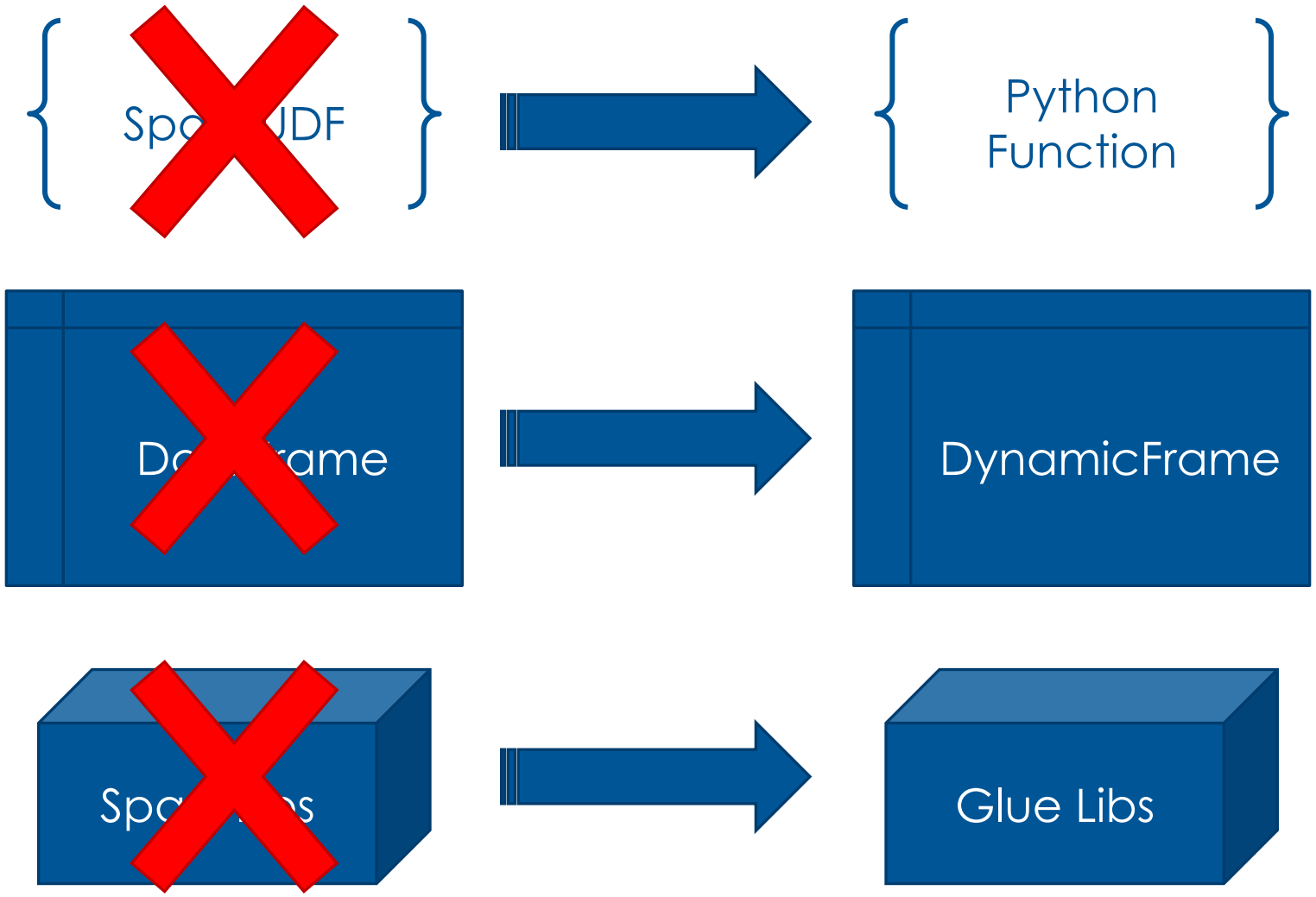
Implement the DynamicFrame manipulations with AWS Glue

Implement the Spark DataFrame Transformations

Tests Code File: tests/test_carrier_transforms_spark, **Code file:** carrier_transforms.py



AWS Glue Differences



AWS Glue Test

Sample Test (using DataFrame & Glue)

```
def test_processYearRange_valid_twoYears(
  spark_session, glue_context, carrier_input_schema,
  carrier_output_schema):

  # given
  input_data = [
    ('C01', 'Carrier (2016 - 2020)'),
    ('C04', 'Carrier (carrier) (2010 - 2016)')]
    input_df = spark_session.createDataFrame(
      data=input_data, schema=carrier_input_schema)

  expected_df = spark_session.createDataFrame(
    data=[
      ('C01', 'Carrier (2016 - 2020)', 2016, 2020),
      ('C04 ', 'Carrier (carrier) (2010 - 2016)', 2010, 2016)
      ],
      schema=carrier_output_schema)

  # when
  actual_df = processYearRange(input_df, glue_context)

  # then
  assert_dataframes_equal(expected_df, actual_df)
```

A Few Notes

- This test starts with a DataFrame and not a DynamicFrame to show the similarity in test writing.
- Normally DynamicFrames would be used exclusively.
- Notice that the change is that processYearRange() takes in glue_context
- GlueContext will be used in the Docker container to leverage AWS Glue

Review

- Docker is a great way to isolate development environments
- Dev IDEs can use the Docker containers as the python interpreter
- Design your pipelines by isolating transformations from integrations (I/O)
- Create testable scalar functions for business logic
- For Spark/Glue, define test data as input DataFrames/DynamicFrames
- Use fixtures for reusable resources like the SparkSession or GlueContext

Next Steps & Challenges

- The Docker setup was not as easy as hoped (see Readme.md)
- Use more efficient code organization for tests (no copy/paste)
- Explore other Glue APIs
- Implement tests using S3 and other integrations
- Configure the IDE to be able to debug tests
- As ELT/ETL gets more complex, the test automation gets more difficult
- Testing is a very complex skill that requires practice

Parting Words for Action

1 Test > No Tests.

Always.

Be quantifiably and visibly **trustworthy**.

If you can “find time” to support defects and production issues, you can **plan time** to start testing.

Thank You!

Github → https://github.com/donaldsawyer/tdde-py-pyspark

LinkedIn → https://www.linkedin.com/my/donaldsawyer