ACE - Technical Case Study:

SAP Data Modeling and PySpark Coding

Exercise Overview

We are working with data from two SAP systems that have similar data sources. Task is to implement and integrate this data to provide a unified view for supply chain insights.

The exercise involves:

- Processing Local Material data.
- Processing Process Order data.
- Ensuring both datasets have the same schema for harmonization across systems.
- Writing modular, reusable code with proper documentation.
- Following best-in-class principles for flexibility and maintainability.

Development Overview

This document outlines the development and setup process for the project, which was built and managed across two environments: the local system and Palantir Foundry. It highlights version control, CI/CD practices, coding standards, and execution workflows to ensure the project's reliability, maintainability, and ease of use for developers and reviewers.

About Data

System 1: Pre-Production Environment (PRE)

1. PRE_AFKO: Production Orders Header Data (Pre-Production Environment)
2. PRE_MARC: Plant-Specific Material Master Data (Pre-Production Environment)
3. PRE_MBEW: Material Valuation Data (Pre-Production Environment)
4. PRE_AUFK: Order Master Data (Pre-Production Environment)
5. PRE_T001K: Controlling Area Settings (Pre-Production Environment)
6. PRE_T001: Company Codes Data (Pre-Production Environment)
7. PRE_MARA: General Material Master Data (Pre-Production Environment)
8. PRE_T001W: Plants Data (Pre-Production Environment)
9. PRE_AFPO: Production Order Item Data (Pre-Production Environment)

System 2: Production Environment (PRD)

1. PRD_AFKO: Production Orders Header Data (Production Environment)
2. PRD_MARC: Plant-Specific Material Master Data (Production Environment)
3. PRD_MBEW: Material Valuation Data (Production Environment)
4. PRD_AUFK: Order Master Data (Production Environment)
5. PRD_T001K: Controlling Area Settings (Production Environment)
6. PRD_T001: Company Codes Data (Production Environment)
7. PRD_MARA: General Material Master Data (Production Environment)
8. PRD_T001W: Plants Data (Production Environment)
9. PRD_AFPO: Production Order Item Data (Production Environment)

Version Control

- Local System:
    - Used GitHub for version control, where branches were structured to maintain a clean and organized workflow for development, testing, and deployment.
    - Ensured consistent version tracking for all project changes.
- Foundry:
    - Leveraged Foundry's built-in Git setup for version control, seamlessly integrating with Foundry's platform while adhering to its repository and branching standards.

Project Setup

1. Folder Structure:
    - The project adheres to a well-structured folder organization that ensures clear separation of concerns (e.g., code, tests, configurations, datasets).
    - The folder structure aligns with best practices to streamline collaboration and CI/CD processes.
2. Coding Standards:
    - Enforced consistent coding standards through automated linting and style checks.
    - This ensures that all code adheres to established guidelines, improving readability, and reducing technical debt.
3. Continuous Integration (CI):
    - Advanced CI Pipeline: Maintained an advanced CI setup that includes:

        - Automated code quality checks.
        - Test execution for every pull request or commit.
        - Coverage checks to ensure thorough testing.
    - Designed for both local development and Foundry environments, maintaining consistency across platforms.

Code Coverage

- Introduced code coverage to ensure that all code, including edge cases, is tested.
- Features:
    - Automated Coverage Checks: Prevent untested functions from being introduced into the codebase.
    - Simple Developer Commands: Developers can easily identify missed test cases using straightforward commands.
    - Lead Review Support: Leads can efficiently review code changes and identify gaps in testing, simplifying the code review process.

Execution Setup

1. Local System:
    - Designed the project to be executed as a library, enabling users to run it via simple commands without requiring manual code modifications or imports.
    - Process:
        - Pass input datasets and instructions as parameters.
        - Output results are automatically saved to the desired location.
    - This setup eliminates the need for a cloud platform or automation during local development.
2. Foundry:
    - Maintained the same setup principles, while adapting to Foundry's pipeline-based execution model:
        - Ensured seamless integration with Foundry's tools and workflows.
        - Automated execution aligned with Foundry's principles, including data lineage and dependency tracking.

Key Benefits

- Security and Automation: The project ensures a secure workflow with automated checks and test execution, reducing human errors.
- Developer-Friendly:
    - Developers can quickly identify and fix issues using simple commands.
    - The library-based execution eliminates complexity for end-users.
- Reviewer Efficiency: Code review is streamlined with coverage reports and clear testing feedback.
- Platform Consistency: The setup and principles are consistent across local and Foundry environments, reducing the learning curve for developers.

Steps to run project.

- Step 1: Obtain the Project Files
    - clone the project from the Git repository:
        - git clone https://github.com/vinay07o/ace-use-case.git
    - OR 2. If provided as a .zip file in an email, extract the files:
        - Right-click the .zip file.
        - Choose Extract All and select a destination folder.

```
C:\Users\M334161\Documents\SDE-Interview\test_git_version>git clone https://github.com/vinay07o/ace-use-case.git
Cloning into 'ace-use-case'...
remote: Enumerating objects: 162, done.
remote: Counting objects: 100% (162/162), done.
remote: Compressing objects: 100% (116/116), done.
remote: Total 162 (delta 61), reused 138 (delta 40), pack-reused 0 (from 0)
Receiving objects: 100% (162/162), 6.69 MiB | 4.82 MiB/s, done.
Resolving deltas: 100% (61/61), done.

C:\Users\M334161\Documents\SDE-Interview\test_git_version>
```

- Step 2: Set Up the Environment
    - Pre-requisites: Ensure the following are installed:
        - Java 17:
            - java --version
        - Python (compatible with PySpark).
            - PySpark 3.5.3 installed via pip:
            - pip install pyspark==3.3.2
        - Hadoop winutils (Windows users only):
            - Download winutils.exe.
            - Place it in C:\hadoop\bin.

        - Make command.
            - powershell -Command "Set-ExecutionPolicy RemoteSigned -Scope CurrentUser -Force; iwr -useb get.scoop.sh | iex"
            - scoop install make
- Step 3: Navigate to the project folder:
    - cd ace-use-case

```
C:\Users\M334161\Documents\SDE-Interview\test_git_version>cd ace-use-case

C:\Users\M334161\Documents\SDE-Interview\test_git_version\ace-use-case>
```

- Step 4: Setup Virtual Environment
    - Create a vertual Environment
    - make venv

```
C:\Users\M334161\Documents\SDE-Interview\test_git_version\ace-use-case>make venv
"Creating a new virtual environment in venv..."
python -m venv venv
"Installing dependencies..."
venv\Scripts\pip.exe install -r requirements.txt
Collecting black (from -r requirements.txt (line 2))
  Using cached black-24.10.0-cp312-cp312-win_amd64.whl.metadata (79 kB)
```

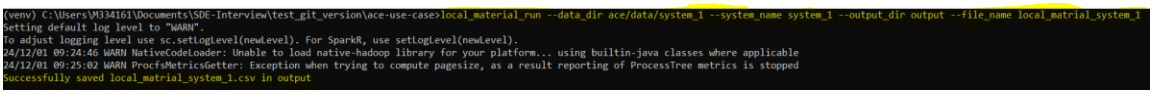
    - Activate the Virtual Environment
    - venv\Scripts\activate

```
C:\Users\M334161\Documents\SDE-Interview\test_git_version\ace-use-case>venv\Scripts\activate

(venv) C:\Users\M334161\Documents\SDE-Interview\test_git_version\ace-use-case>
```

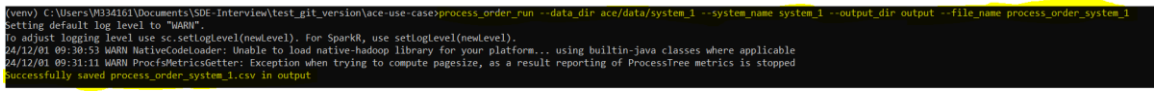
- Step 5: Execute Local material dataset from any system.
    - Get information on input parameter.
    - local_material_run –help

```
(venv) C:\Users\M334161\Documents\SDE-Interview\test_git_version\ace-use-case>local_material_run --help
usage: local_material_run [-h] -d DATA_DIR -s SYSTEM_NAME -o OUTPUT_DIR [-f FILE_NAME]

options:
  -h, --help            show this help message and exit
  -d DATA_DIR, --data_dir DATA_DIR
                        data set folder path where csv files located.
  -s SYSTEM_NAME, --system_name SYSTEM_NAME
                        specify the system name where source data came.
  -o OUTPUT_DIR, --output_dir OUTPUT_DIR
                        path to save output dataset.
  -f FILE_NAME, --file_name FILE_NAME
                        path to save output dataset.
```

    - Pass input and output paths with master function.
    - local_material_run --data_dir ace/data/system_1 --system_name system_1 --output_dir output --file_name local_matrial_system_1



    - local_material_run --data_dir ace/data/system_2 --system_name system_2 --output_dir output --file_name local_matrial_system_2
- Step 6: Execute process order dataset from any system.
    - Pass input and output paths with master function.
    - process_order_run --data_dir ace/data/system_1 --system_name system_1 --output_dir output --file_name process_order_system_1



    - process_order_run --data_dir ace/data/system_2 --system_name system_2 --output_dir output --file_name process_order_system_2
- Step 7: If you want to union multiple system dataset use below command (optional).
    - Pass any model different systems dataset.
    - union_datasets --data_path output/local_matrial_system_2.csv output/local_matrial_system_1.csv --output_dir output --file_name local_material

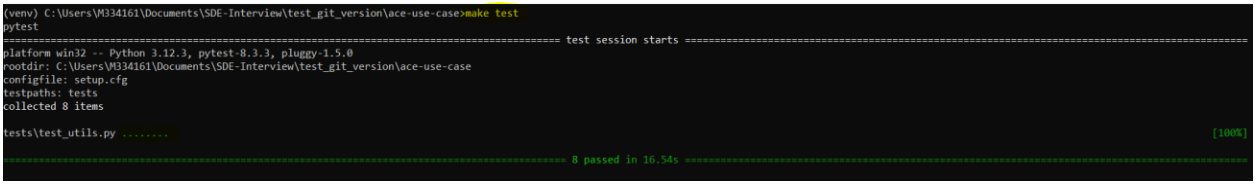
    - union_datasets --data_path output/process_order_system_2.csv output/process_order_system_1.csv --output_dir output --file_name process_order

Steps to run execute test cases and code coverage.

These acceptance tests ensure that our codebase is thoroughly tested, with 100% test coverage. They act as a quality gate to verify that all implemented functionality is covered by test cases. This approach helps both developers and lead developers streamline the review process by adhering to coding standards. It facilitates smooth self-reviews and lead reviews, ensuring a robust and maintainable codebase.

Step 1: Run make test to validate the project and confirm that all utilities execute correctly without any errors (Optional).

make test



Step 2: Run a code coverage analysis to verify if all logic branches in the code have been tested effectively (Optional)

make coverage

```
(venv) C:\Users\M334161\Documents\SDE-Interview\test_git_version\ace-use-case>make coverage
coverage run -m pytest -m 'not compare'
============================================================================================= test session starts =
platform win32 -- Python 3.12.3, pytest-8.3.3, pluggy-1.5.0
rootdir: C:\Users\M334161\Documents\SDE-Interview\test_git_version\ace-use-case
configfile: setup.cfg
testpaths: tests
collected 8 items

tests\test_utils.py ........

============================================================================================== 8 passed in 15.60s =
coverage report
SUCCESS: The process with PID 33624 (child process of PID 35684) has been terminated.
SUCCESS: The process with PID 35684 (child process of PID 15116) has been terminated.
SUCCESS: The process with PID 15116 (child process of PID 31196) has been terminated.
Name                          Stmts   Miss Branch BrPart  Cover
-------------------------------------------------------------
ace\utils\__init__.py             3      0      0      0   100%
ace\utils\_business_utils.py     98     79     16      0    17%
ace\utils\_use_case_utils.py     87     53     50      0    35%
-------------------------------------------------------------
TOTAL                           188    132     66      0    28%
```

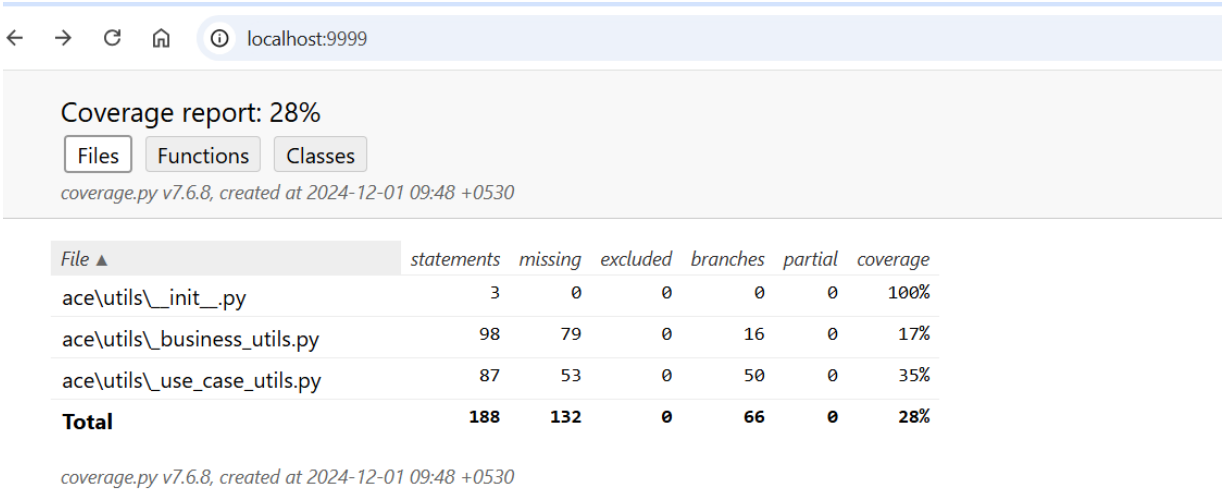
Step 3: Run below command to get coverage report

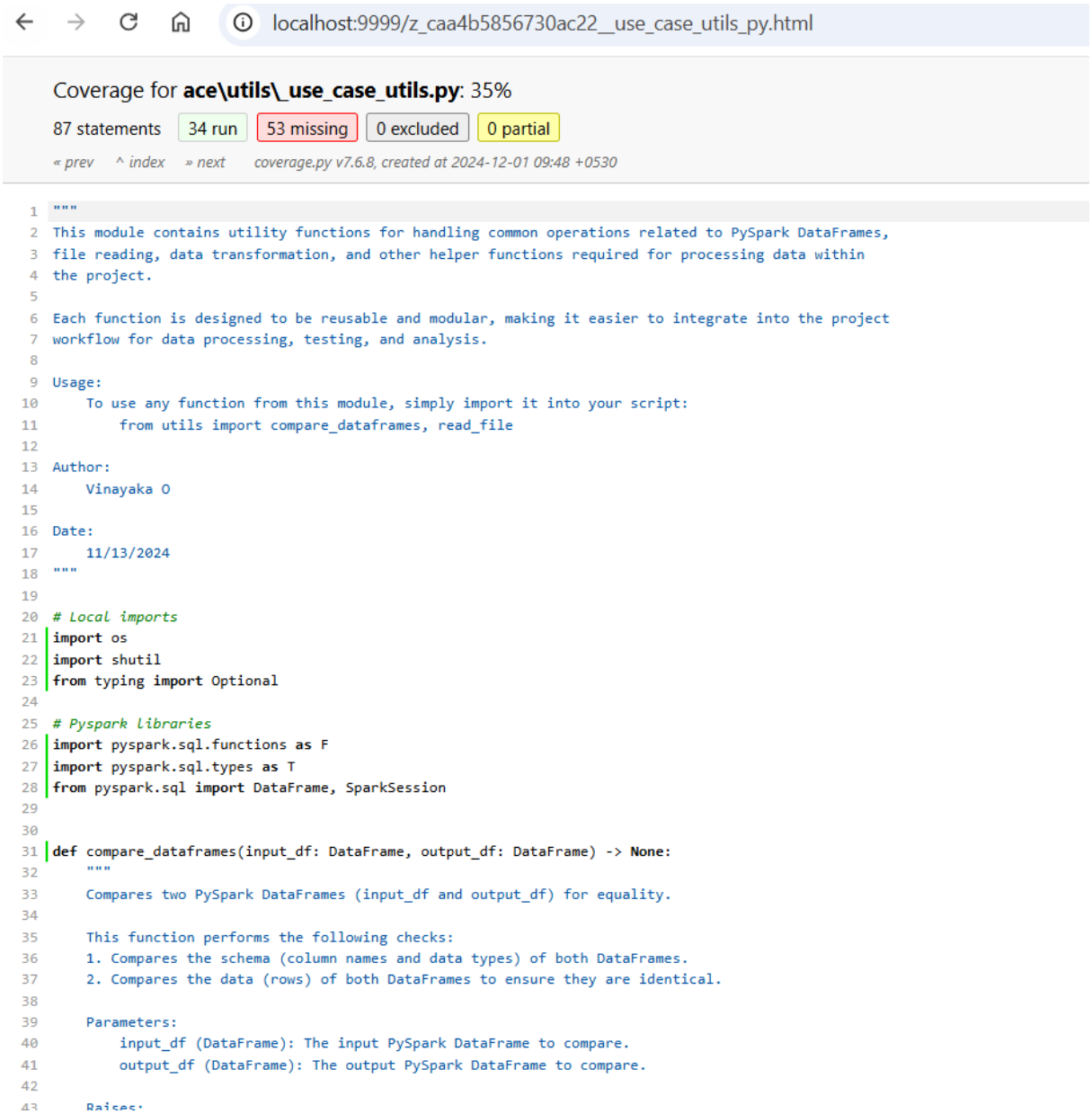
coverage report

Step 4: To check for missing lines in code coverage, run the following command

make coverage-html

The server starts at localhost:9999.



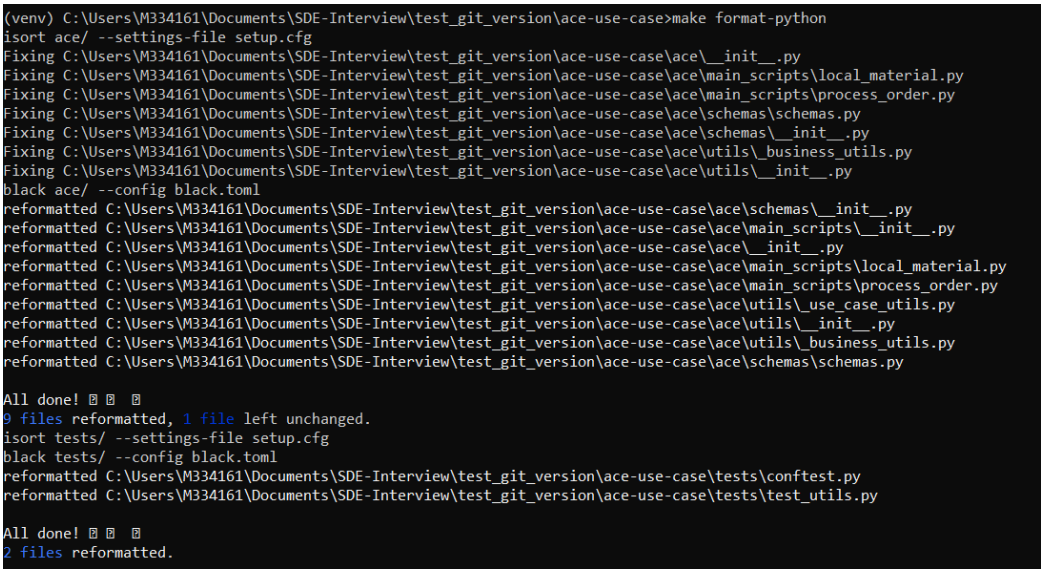


Use this address to review each file and gather details about functions or logic segments that have not been tested. This step is crucial for identifying areas that require additional test coverage to ensure the robustness of the project.

Steps to Run Python Formatters

I am also providing a simple command to auto-format all Python files in the project to align with PEP8 standards. This automation ensures that developers no longer need to worry about code formatting manually. By running this command, the entire project will be formatted automatically, saving time and maintaining consistency across the codebase.

- Execute following command to format all python files of this project with PEP8 standards
    - make format-python



Note: An Excel sheet is provided in the documents folder, offering a clear overview of the schema and the mapping of new columns along with their respective data types.

Conclusion

This project is designed to ensure high-quality, maintainable, and testable code across both local and Foundry environments. By implementing advanced CI/CD practices, maintaining coding standards, and introducing tools like code coverage, the project enhances developer productivity, ensures comprehensive testing, and simplifies the execution process.