Project: Cost Data Processing Pipeline using Pandas and PySpark

This project implements a data transformation pipeline to process marketing cost data. It provides two solutions based on the dataset size:

- **Pandas Solution:** Ideal for small to medium datasets.
- **PySpark Solution:** Designed for large-scale datasets, offering better scalability and distributed processing.

Both solutions are triggered via a **BashOperator** in **Apache Airflow**, and are organized in separate folders:

Project Structure

1. `pandas_app/`

- `pandas_transformations.py`: Contains the main pipeline logic using Pandas.
- `test.py`: Includes validation functions for input and output data.
- `logs/`: Stores log files for each job run.
- `output/`: Saves the pipeline output.

2. `pyspark_app/`

- `pyspark_transformation.py`: Contains the main PySpark pipeline implementation.
- `tests_pyspark.py`: Defines validation functions for input and output data.
- `logs/`: Stores log files generated during execution.
- `output/`: Stores the final output file.

Validation Overview

The `tests_pyspark.py` file contains unit-style test functions for both input and output validation:

Input File Validations

- `validate_raw_costs()`
- `validate_exchange_rates()`
- `validate_account_info()`
- `validate_customer_account_relationship()`
- `validate_duplicates_in_raw_costs()`

Output Validations

- `validate_cost_eur_calculation()`
- `validate_rate_for_eur()`

Pipeline Stages

The pipeline in `pyspark_transformation.py` follows these structured stages:

1. **Input Validation**
2. `run_validations_input_files()`
3. Uses external validators from `tests_pyspark.py`
4. **Data Loading**
5. `load_data()`
6. **Preprocessing**
7. `preprocess_data()`
8. **Deduplication**
9. `drop_duplicates_keep_last()`
10. **Data Enrichment & Transformation**
11. `join_costs_with_accounts()`
12. `add_eur_cost_column()`
13. **Aggregation**
14. `aggregate_daily_spend()`
15. **Output Validation**
16. `validate_cost_eur_calculation()`
17. `validate_rate_for_eur()`
18. **Output Writing**
19. `write_df_to_file()`

This modular design ensures the pipeline is maintainable, testable, and scalable for evolving data needs.