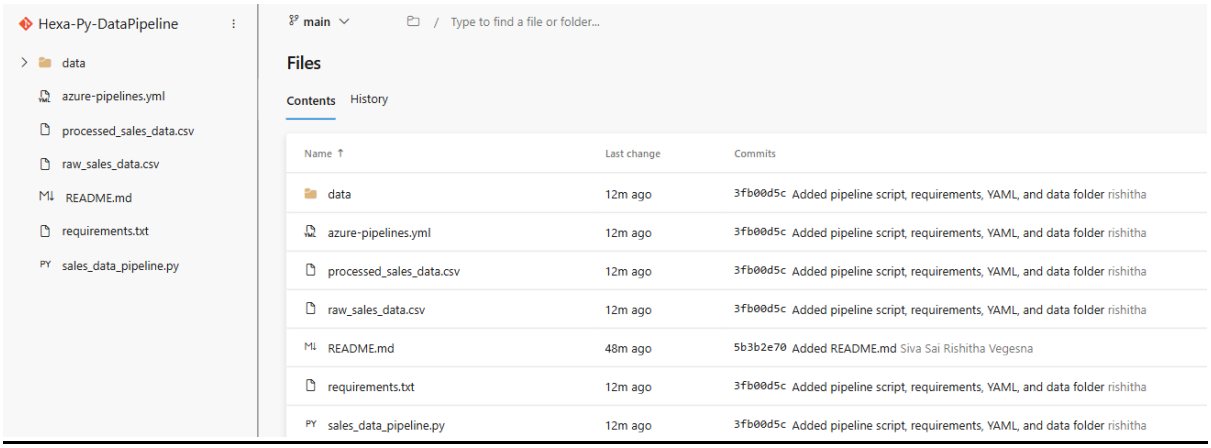
TASK-2 SCREENSHOTS

REPO STRUCTURE:



Local VS Code terminal showing script run success.

```
>>
✅ Files created: raw_sales_data.csv, processed_sales_data.csv
[Blob Upload] Skipped (no Azure credentials): raw_sales_data.csv
[Blob Upload] Skipped (no Azure credentials): processed_sales_data.csv
PS C:\Users\Rishitha\Desktop\Hexa-Py-DataPipeline>
```

Showing all the files

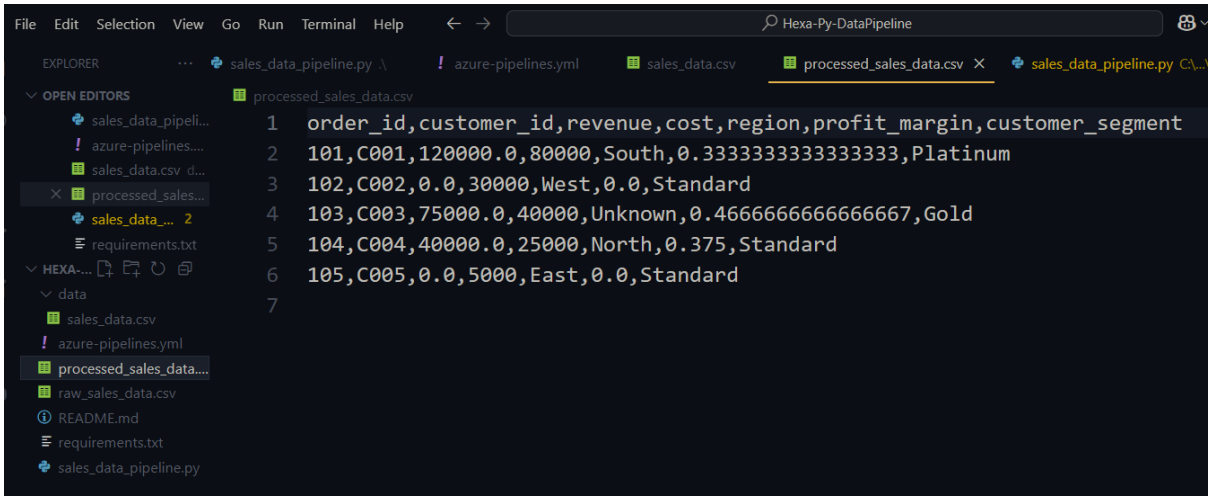
```
PS C:\Users\Rishitha\Desktop\Hexa-Py-DataPipeline> dir


    Directory: C:\Users\Rishitha\Desktop\Hexa-Py-DataPipeline


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
d-----        25-08-2025     15:28                data
-a----        25-08-2025     15:44            844 azure-pipelines.yml
-a----        25-08-2025     16:09            308 processed_sales_data.csv
-a----        25-08-2025     16:09            204 raw_sales_data.csv
-a----        25-08-2025     15:26            985 README.md
-a----        25-08-2025     15:30             28 requirements.txt
-a----        25-08-2025     15:43           2685 sales_data_pipeline.py


PS C:\Users\Rishitha\Desktop\Hexa-Py-DataPipeline>
```

Cleaned processed_sales_data.csv



BONUS QUESTIONS

1. Why is storing cleaned data in Azure Blob Storage important for real-time pipelines?

- Azure Blob Storage acts as a centralized, scalable, and cost-effective data lake.
- Storing cleaned data ensures that downstream systems (like Databricks, Synapse, Power BI, or ML models) consume consistent and high-quality data.
- It enables real-time analytics because multiple services can access the same “single source of truth” simultaneously.
- It also supports versioning, durability, and global accessibility, which are critical for production pipelines.

2. What’s the difference between pipeline artifacts and Blob Storage uploads?

- Pipeline Artifacts:
  - Used inside Azure DevOps to share files between pipeline stages/jobs.
  - Temporary, for CI/CD workflow execution.
  - Accessed only within Azure DevOps.
- Blob Storage Uploads:

    - Stored in Azure Storage account, outside DevOps.
    - Persistent, production-ready storage.
    - Accessible by applications, analytics platforms, and external services.

3. How would you handle failures in file uploads in a production setup?

- **Retry mechanism**: Implement automatic retries with exponential backoff.
- **Error handling & logging**: Log all upload attempts and errors for debugging.
- **Alerts/Monitoring**: Use Azure Monitor / Application Insights to trigger alerts when uploads fail.
- **Idempotent uploads**: Always overwrite or use unique file names to avoid duplicates.
- **Fallback strategy**: Store failed uploads in a “dead-letter queue” or backup location for later reprocessing.
- **Fail pipeline stage if critical**: Ensure pipeline doesn’t silently succeed if uploads are mandatory.