* **Project Overview**:
  + Practice building a data engineering pipeline using a real-world retail dataset to analyze delivery performance at scale.
  + The pipeline can be configured to use cloud storage solutions or MinIO, with the option to run processing locally using Docker.
* **Problem Statement**:
  + Retailers need an omni-channel approach to adapt to the digital age, especially those with brick-and-mortar investments.
  + The project aims to create a data foundation for analytics and modeling, providing summary reports for decision-makers.
* **Tools and Technologies**:
  + **Python**: Used for scripting and data manipulation.
  + **SQL**: Utilized for querying and data processing.
  + **Airflow**: Manages ETL workflows and job scheduling.
  + **Spark**: Handles large-scale data processing and analytics.
  + **Docker**: Enables local execution of the pipeline without cloud costs.
  + **Cloud Storage Options**:
    - **AWS S3**: Default storage solution.
    - **GCP Cloud Storage**: Optional storage solution.
    - **Azure Blob Storage**: Optional storage solution.
    - **MinIO**: Local storage alternative for S3-compatible operations.
* **ETL Pipeline**:
  + **Extract**: Download data from chosen cloud storage or MinIO using Python and Boto3 or equivalent libraries.
  + **Transform**: Use Spark to manipulate and clean the data, focusing on delivery performance metrics.
  + **Load**: Upload the cleaned dataset back to the chosen storage solution for further analytics.
* **Dataset**:
  + The dataset consists of tables from Ecommerce company.
  + It includes various CSV files representing different aspects of ecommerce operations.
* **Methodology**:
  + **Data Lake Setup**: Create a mock production data lake using the chosen storage solution with the provided table schema.
  + **Data Analysis**: Perform exploratory data analysis (EDA) to identify delivery performance issues.
  + **Spark SQL Job**: Join tables to determine which orders/sellers missed delivery deadlines.
* **Airflow Configuration**:
  + **Installation**: Install Airflow using pip install apache-airflow.
  + **Docker Usage**: Run Airflow in a Docker container for reproducibility and isolation.
  + **DAGs and Operators**: Write Python scripts to define ETL processes and steps.
* **Pipeline Steps**:
  + **Data Download**: Retrieve the Brazilian ecommerce data from the chosen storage solution.
  + **Data Processing**: Use Spark SQL to join tables and filter for missed delivery deadlines.
  + **Data Upload**: Save the processed data back to the storage solution in a designated folder for analytics.
* **Technical Details**:
  + **Boto3 or Equivalent**: Used for interacting with cloud storage in Python scripts.
  + **Jupyter Notebooks**: Employed for EDA and visualization using libraries like Pandas and Matplotlib.
  + **Spark Session**: Set up to run SQL operations and write results to CSV.
* **Airflow DAG Configuration**:
  + **File Download**: Initial step to download data from storage.
  + **Spark Job Execution**: Process data to identify missed delivery deadlines.
  + **Data Upload**: Final step to upload results to storage.
* **Job Scheduling**:
  + **Manual Triggering**: Use Airflow UI to manually trigger DAGs.
  + **Retry Mechanism**: Implement .set\_upstream() to handle job retries.
* **Future Work**:
  + Plan to integrate cloud-based processing using AWS EMR, GCP Dataproc, or Azure HDInsight for running Spark jobs on a cluster.