**Objective**

Design and implement a modular data engineering solution using Python. The goal is to process a public healthcare dataset, transform it into a well-structured format, and demonstrate how it could be saved to a bronze layer. The solution should include code, documentation, and a brief report summarizing your approach and findings.

**Scenario**

You are working as a data engineer for a healthcare analytics company. The company aims to build a robust, scalable, and reusable data pipeline to process patient data for analytical purposes. This pipeline will support data-driven decision-making in diabetes management and hospital resource allocation. Your task is to design and implement this data pipeline, ensuring it can handle large volumes of data efficiently and maintain high data quality standards.

**Dataset**

* **Dataset Name**: [Diabetes 130-US hospitals for years 1999-2008 Data Set](https://archive.ics.uci.edu/dataset/296/diabetes+130-us+hospitals+for+years+1999-2008)
* **Description**: This dataset includes over 100,000 instances of patient encounters from 130 hospitals in the US. It contains information on patient demographics, diagnoses, laboratory tests, medications, and more.
* **Source**: UCI Machine Learning Repository

**Requirements**

1. **Modular Program Structure**
   * Create a modular Python program that can be easily extended or modified.
   * The program should have separate modules for:
     + **Data Ingestion**: Load the dataset into a DataFrame.
     + **Data Validation**: Check for missing values, duplicates, invalid entries, outliers, and unknown values.
     + **Data Transformation**: Perform necessary transformations to create a well-structured dataset.
     + **Data Saving**: Demonstrate how the transformed dataset and encoding files could be saved to a file system (e.g., Parquet format).
2. **Robust Pipeline Design**
   * **Incremental Processing**: Implement batch processing to handle data in smaller chunks rather than processing the entire dataset at once.
   * **Error Handling and Monitoring**: Implement robust error handling mechanisms to capture and log errors without stopping the entire pipeline. Set up comprehensive monitoring and logging to track the pipeline's performance and identify issues in real-time.
   * **Scalability**: Design the pipeline to scale horizontally, allowing it to handle increasing data volumes by adding more processing nodes.
3. **Data Transformation**
   * Handle missing values appropriately. Document your approach to handling missing values.
   * Encode categorical columns (e.g., race, gender, age, admission\_type\_id, discharge\_disposition\_id, admission\_source\_id) as categorical data types and create encoding files/tables.
   * Handle outliers appropriately. Document your approach to identifying and handling outliers.
   * Handle unknown or invalid values appropriately. Document your approach to identifying and handling unknown or invalid values.
4. **Data Quality Tests**
   * Prepare tests to check data quality, ensuring that both categorical and non-categorical values are enforced. These tests should include:
     + **Validation of Categorical Values**: Ensure that categorical columns contain only valid values as defined in the code lists provided in IDS\_mapping and categories created from columns like gender.
     + **Validation of Non-Categorical Values**: Ensure that non-categorical columns (e.g., age, num\_lab\_procedures, time\_in\_hospital) contain valid and reasonable values.
     + **Consistency Checks**: Verify that the encoded values match the original values and that there are no discrepancies.
     + **Integrity Checks**: Ensure that there are no missing or invalid values in critical columns.
5. **Documentation**
   * Provide step-by-step instructions on how to run the program locally.
   * Explain your approach to handling data quality issues and transformations, including how you handle missing values, outliers, and unknown or invalid values.
   * Document each step of the process, including:
     + **Data Validation**: Document the presence of missing values, duplicates, outliers, unknown values, and any validation checks performed.
     + **Transformation Steps**: Detail each transformation applied to the data, including the rationale and methods used.
     + **Encoding Files**: Provide documentation for encoding categorical columns, including the mapping of original values to encoded values.
     + **Data Quality Report**: Summarize the results of data quality checks, including any issues identified and how they were addressed.
     + **Audit Log**: Maintain a log of all changes made during the migration, including timestamps and responsible personnel.
     + **Final Dataset Description**: Provide a comprehensive description of the final dataset, including metadata and any derived columns.
6. **Report**
   * Include a concise summary of your findings from the data transformation process.
   * Discuss any challenges faced during implementation and how you addressed them.

Deliverables

* **Code**:
  + Modular Python scripts for data ingestion, validation, transformation, and saving.
* **Documentation**:
  + Instructions on running the program locally.
  + Explanation of data transformation logic and handling of data quality issues.
  + Detailed documentation of each step in the process.
* **Report**:
  + Summary of data transformation outcomes.
  + Discussion of challenges and solutions.

Evaluation Criteria

* **Modularity**: Ease of extending or modifying the program.
* **Data Quality**: Handling of missing values, duplicates, invalid entries, outliers, and unknown values.
* **Documentation**: Clarity and completeness of instructions provided.
* **Insights**: Quality of data transformation and readiness for further analysis.