**Data Engineering**

**By Devi Sri Charan**

[Claims Management Data Engineering for Analytics 2](#_Toc132809985)

[Incremental Update Mechanism Development 3](#_Toc898969523)

[Implementation: 3](#_Toc667825593)

[Code Snippet: 3](#_Toc1922968083)

[Data Format Implementation 4](#_Toc1462900359)

[Implementation: 4](#_Toc127927471)

[Code Snippet: 4](#_Toc1848454230)

[Data Quality Automation 5](#_Toc1319934978)

[Implementation: 5](#_Toc1661589638)

[Code Snippet: 5](#_Toc93759856)

[Data Transformation 6](#_Toc1023341406)

[a. Dataset Creation 6](#_Toc1057670224)

[Implementation: 6](#_Toc808425241)

[Code Snippet: 6](#_Toc1165286430)

[b. PII Data Encryption and Masking 6](#_Toc1028537380)

[Implementation: 6](#_Toc1854920541)

[Code Snippet: 6](#_Toc907325547)

[Workflow Orchestration 6](#_Toc2037845990)

[Implementation: 6](#_Toc1333731621)

[Code Snippet: 6](#_Toc129272665)

# Claims Management Data Engineering for Analytics

## Incremental Update Mechanism Development

To reliably identify and extract new or updated entries from the database, I used the `updatedAt` timestamp from MongoDB. I created a Python function to query entries that have changed since the last run.

### Implementation:

* **Function: get\_last\_run\_date()**: Retrieves the timestamp of the last successful run to ensure only new or updated entries are processed.
* **Function: update\_last\_run\_date()**: Updates the timestamp of the last successful run.
* **Function: extract\_incremental\_updates()**:
  + Utilizes the last run date to query the MongoDB database for entries in claims, policies, and policyholders tables that have been updated since the last run.
  + Converts the queried data into pandas DataFrames and saves them as JSON files for further processing.
  + Updates the last run date upon successful extraction.

### Code Snippet:



## Data Format Implementation

### Implementation:

* **Function: save\_data\_formats()**:
  + Reads the previously saved JSON files.
  + Converts the data into CSV and Parquet formats, providing flexibility and efficiency in data storage and access.

### Code Snippet:

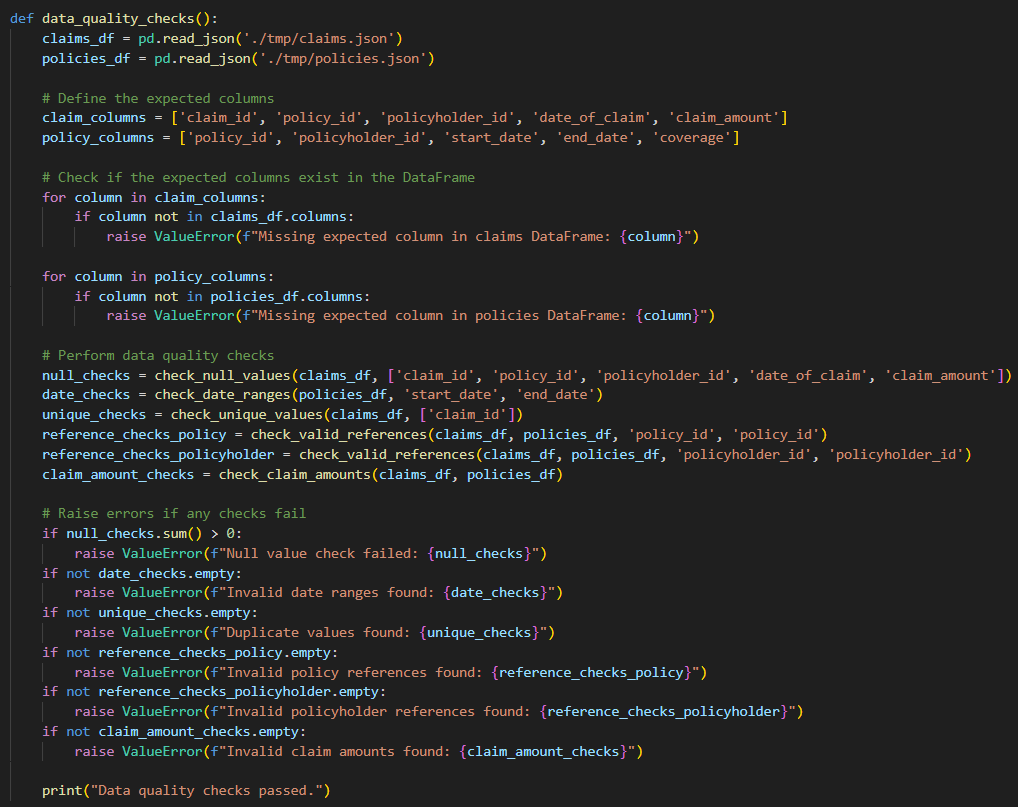
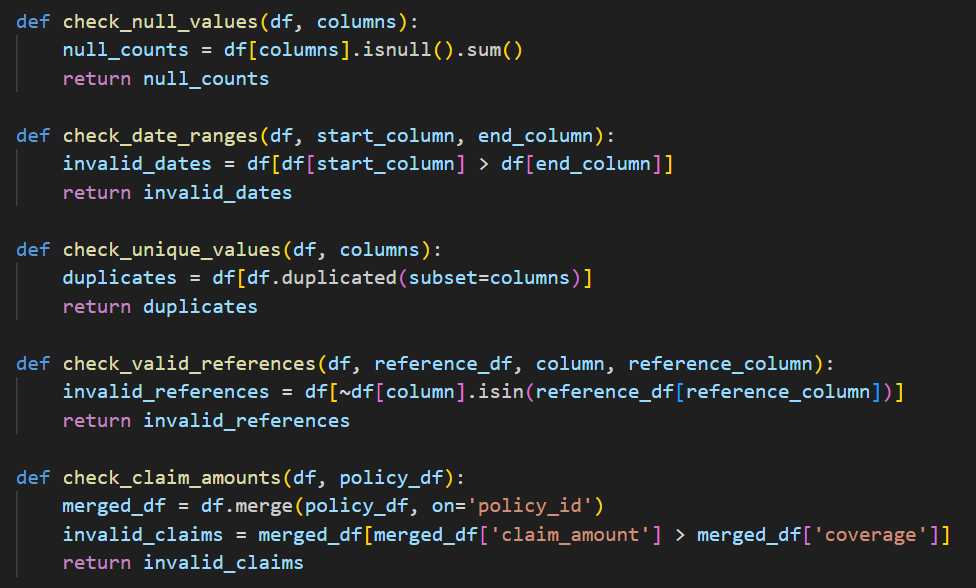


## Data Quality Automation

### Implementation:

* **Function: data\_quality\_checks()**:
  + Performs various checks on the claims and policies DataFrames, including:
    - Null value checks.
    - Date range validation.
    - Duplicate value checks.
    - Reference validity checks.
    - Claim amount validation against policy coverage.
  + Raises errors if any check fails to ensure data quality.

### Code Snippet:



## Data Transformation

### a. Dataset Creation

#### Implementation:

* **Function: transform\_data()**:
  + Merges claims, policies, and policyholders DataFrames to enrich the dataset with policy information.
  + Calculates the number of open and closed claims per policy.
  + Merges the counts back into the policies DataFrame.
  + Saves the transformed data for further processing.

#### Code Snippet:

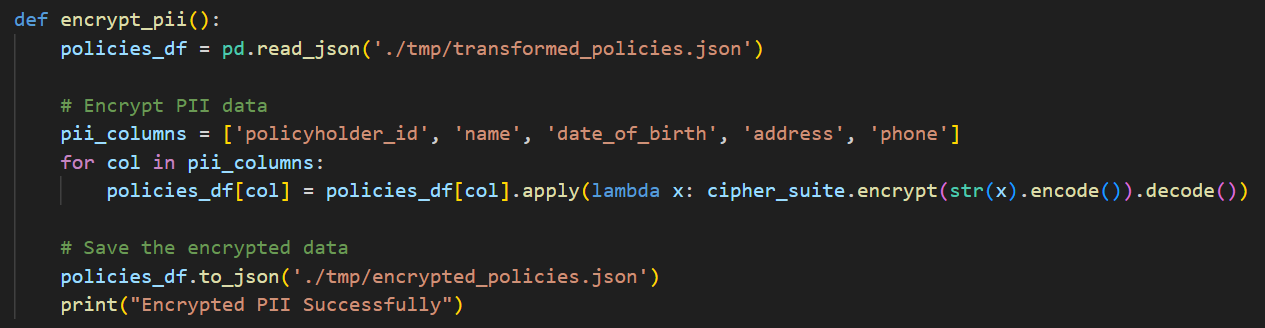


### b. PII Data Encryption and Masking

#### Implementation:

* **Function: encrypt\_pii()**:
  + Encrypts sensitive columns in the transformed policies DataFrame using the cryptography library.
  + Saves the encrypted data to a JSON file for secure storage.

#### Code Snippet:



## Workflow Orchestration

### Implementation:

* **Airflow DAG Configuration**:
  + Defines a directed acyclic graph (DAG) to schedule and execute the data pipeline tasks sequentially.
  + Tasks include extraction, data format conversion, data quality checks, data transformation, and PII encryption.
  + Error handling and logging are integrated into each task to ensure robust execution and monitoring.

### Code Snippet:

