PySpark Project

This project is a comprehensive workflow divided into four essential phases to ensure data is well-prepared for meaningful insights:

- Data Ingestion
- Data Cleaning
- Data Transformation and
- Data Loading

<u>Step 1: Data Ingestion</u>

Two essential files are ingested into the PyCharm IDE. The first file is stored in Parquet format and contains comprehensive details about various cities. The second file, available in CSV format, holds vital information about prescribers, including attributes like prescriber ID, specialty, and years of experience. Using Spark's versatile 'read' operation, these files are converted into dataframes, forming the foundation for subsequent operations.

<u>Step 2: Data Cleaning</u>

In this phase, dataframes are streamlined for relevance by selecting only the pertinent columns required for reporting. A meticulous check for null values is conducted. Null values are handled by either removal or imputation. Some null instances are eliminated, while others are replaced with the average value of their respective columns.

Within the prescribers' dataframe, further adjustments are made. The 'years of experience' column undergoes transformation through the application of the 'regex_replace' function of pyspark. This modified column is then cast into the integer data type. Additionally, the 'prescriber's first name' and 'prescriber's last name' columns are combined to generate a unified 'full name' column.

<u>Step 3: Data Transformation</u>

During the data transformation phase, focus was directed towards addressing crucial business inquiries. These efforts yielded two comprehensive reports by effectively addressing the following key questions:

First report:

1. *Number of Zip Codes per City:*

   To determine the count of zip codes associated with each city, we initially split the 'zips' column to extract individual zip codes. Then, we calculated the count of these zip codes and added a new column to the city dataframe to store this information. This process involved grouping the city dataframe by 'city' and 'state_id' and applying the 'sum' aggregate function to find the total count of zip codes per city.

2. *Prescriber and Claim Count per City:*

   Another analysis aimed to provide insights into the number of unique prescribers and the total claim count within each city. We accomplished this by grouping the prescribers' dataframe based on the 'presc_city' and 'presc_state' columns. Utilizing the 'countDistinct' transformation, we computed the distinct count of prescriber IDs, which represented the number of prescribers in each city. Additionally, we used the 'total_claim_amount' column to calculate the total claim count for each city.

3. *Cities with Assigned Prescribers:*

   Our next task involved identifying cities with actively assigned prescribers. This was achieved by performing an inner join between the dataframes obtained from the above two questions. This ensured that only cities with associated prescriber data were included in the result. The outcome was a consolidated dataframe containing relevant information about the number of prescribers, zip codes, and cities with assigned prescribers.

Second report:

1. *Top Prescribers within Experience Range:*

   For the second data analysis, prescribers whose experience fell within the range of 20 to 50 years are targeted. Utilized a filter operation to extract these prescribers from the dataset. Subsequently, applied a window function to rank prescribers within each state based on their total claim amount. Successfully highlighted experienced prescribers with significant claim performance by filtering the results to retain only the top 5 prescribers within each state.

Step 4: Data Loading:

During the data loading phase, integration with Azure Blob Storage is established to facilitate the migration of files from the local system to the Azure Blob Storage. This operation requires the utilization of the BlobServiceClient class, which is imported from the azure.storage.blob module. By employing the provided connection string, the code establishes a connection with Azure Blob Storage.

Subsequently, both of the generated reports are transferred to Azure Blob Storage. This process involves the seamless uploading of files, ensuring their secure storage and accessibility. This orchestration contributes to the effective management and dissemination of the generated reports through Azure's robust storage infrastructure.

Additionally, a .config file is created to encapsulate crucial details pertaining to loggers and their associated attributes.