ETL Report Guide

Group 2: Insurance

ETL Process

Introduction

This ETL report showcases the process required to extract and transform the three different datasets we have found through our research on health insurance. Each dataset displays different demographics that influence health insurance trends within the United States. Prior to cleaning our Indicators of Health Insurance Coverage, US Health Insurance, and SAHIE (Small Area Health Insurance Estimates) datasets, we created a producer and consumer streams in order to utilize our data. Utilizing these datasets, we cleaned and imported the data into a SQL database and used it to produce telling visuals.

Data Sources

Data.gov. (2020, July 10). *Indicators of Health Insurance Coverage at the Time of Interview*. Data.World. Retrieved August 1, 2022, from https://data.world/us-hhs-gov/0221e1ce-448c-4e75-96e1-3412df535d7f

Datta, A. (2020, February 16). *US Health Insurance Dataset*. Kaggle. Retrieved August 1, 2022, from https://www.kaggle.com/datasets/teertha/ushealthinsurancedataset

US Census Bureau. (2021, October 8). *2008 - 2019 Small Area Health Insurance Estimates (SAHIE) using the American Community Survey (ACS)*. Census.Gov. Retrieved August 1, 2022, from https://www.census.gov/data/datasets/time-series/demo/sahie/estimates-acs.html

Extraction

To obtain our data for this project we searched multiple search engines to find suitable and extractable datasets from Kaggle, the US Census Bureau, and Dataworld. Once we studied these datasets we came to the conclusion that after the proper cleaning and tuning, they would be usable. We downloaded each dataset from their original sources, which granted us csv files, and began altering them to the point required for our project.

Transformation

1. Kafka Producer
    a. Created Kafka error message for Producer
    b. Imported the necessary libraries provided to us by instructors
        i. Imported random
        ii. Imported sleep from time
    c. Set up strings for Producer using variables provided by instructors
        i. Changed topic name to “insurance-capstone2”
    d. Created proper topics using the strings
    e. Created producer object
    f. Defined mount point to read in data files
        i. Mounted the capstone container named “capstone-group2-data”
    g. Simulate live data feed by creating N instances of health care costs from the dataset and push to topic streamline
2. Kafka Consumer
    a. Imported pyspark.sql.functions
        i. Imported concat, col, lit
        ii. Imported StringType, DecimalType, IntegerType, ByteType
    b. Imported necessary libraries provided by instructors
    c. Created Kafka error messages for Consumer
    d. Set up the consumer using variables provided by instructors
        i. Changed topic name to “insurance-capstone2”
    e. Set up Consumer class
    f. Created code to consume all the messages from the producer until the producer stops sending messages
        i. Created “kafkaListDictionaries” list where those messages will be appended
3. Kaggle Health Insurance Dataset
    a. Read the csv from “rawdata” folder in our capstone container called “capstone-group2-data”
    b. Turned “kafkaListDictionaries” into spark dataframe named sparkdf
    c. Imported regxp_replace from pyspark.sql.functions
    d. Altered the “Smoker” column to display 1/0 instead of Y/N, making it similar to a boolean data type

e. Converted all the columns to the appropriate data types for the SQL database
    i. “Age” from string to integer
    ii. “BMI” from string to decimal
    iii. “Charges” from string to decimal
    iv. “Children” from string integer
    v. “Smoker” from string to bit (same as boolean in SQL)
f. Dropped null values and duplicates and saved into a new dataframe
    i. Df = sparkdf.dropna()
    ii. Df = df.dropDuplicates()
g. Created thresholds to make sure any new data would be filtered by the following parameters
    i. Age did not go lower than 18 and did not exceed 120
    ii. Amount of children did go lower than 0 and did not exceed 10
h. Used a new mount point to save the cleaned data to a new csv file in new directory in our container called “cleandata”

4. Indicators of Health Insurance Coverage at the Time of Interview
    a. Uploaded the raw csv file into “rawdata” directory in our container “capstone-group2-data”
    b. Imported pandas as pd
    c. Read the file into a dataframe called df
    d. Dropped unnecessary columns using df.drop(columns=["Quartile Range","Suppression Flag","Low CI","High CI","Confidence Interval"])
    e. Dropped the null values with df.dropna()
    f. Imported SparkSession from pyspark.sql
        i. Created PySpark SparkSession
        ii. Created PySpark DataFrame from pandas into df
    g. Saved the cleaned data in “cleandata” directory in our container
5. 2019 Small Area Health Insurance Estimates (SAHIE) using the American Community Survey (ACS)
    a. Imported necessary libraries
        i. Pandas as pd
        ii. pyspark.sql.functions
            1. Concat, col, lit, StringType, DecimalType, input_file_name, substring

b. Created a mount point that granted us access to the containers in gen10datafund2205
c. Imported the SAHIE dataset into “rawdata” directory in our container “capstone-group2-data”
d. Read the csv file into a dataframe called “raw_sahie” and skipped the first 79 rows due to data beginning at row 80
e. Dropped “year”, “version”, "NIPR", "NUI", "NIC", "statefips", "countyfips", and “margin of error (MOE)” columns
f. Stripped white space from state_name and county_name columns
    i. Used raw_sahie[‘state_name’].str.strip()
    ii. Used raw_sahie[‘county_name’].str.strip()
g. Renamed the columns and saved into new dataframe called “cleaned_sahie”
    i. 'geocat':'Geography_Category',
    ii. 'state_name':'State_Name',#
    iii. 'county_name':'County_Name',
    iv. 'agecat':'Age_Category',#
    v. 'racecat':'Race_Category',#
    vi. 'sexcat':'Sex_Category',#
    vii. 'iprcat':'Income_Category',
    viii. 'PCTUI':'Percent_of_Demographic_Uninsured_by_Income_Category',
    ix. 'PCTIC':'Percent_of_Demographic_Insured_by_Income_Category',
    x. 'PCTELIG':'Total_Percent_of_Demographic_Uninsured',
    xi. 'PCTLIIC':'Total_Percent_of_Demographic_Insured'
h. Deleted blanks using
    i. cleaned_sahie = cleaned_sahie[cleaned_sahie['Percent_of_Demographic_Insured_by_Income_Category'] != ' . ']
i. Changed data types for certain columns
    i. 'Demographic_Numbers_For_Income_Category':'int',
    ii. 'Number_Of_Uninsured':'int',
    iii. 'Number_Of_Insured':'int',
    iv. 'Percent_of_Demographic_Uninsured_by_Income_Category':'float',
    v. 'Percent_of_Demographic_Insured_by_Income_Category':'float',
    vi. 'Total_Percent_of_Demographic_Uninsured':'float',

vii. 'Total_Percent_of_Demographic_Insured':'float'

j. Created new mount point and dataframe called “spark_sahie” to create a new file called “clean_SAHIE” in our “cleandata” directory within our container

6. Creating SQL database
    a. Created a new SQL Server Database called capstoneGroup2Database inside the Gen10.Data.Fundamentals.22-05 resource group
    b. Create login user and password from the master in Azure Data Studio
    c. Create a new connection directly to the capstoneGroup2Database using the username and password created above
    d. Once connected, run DDL script which can be found in our github repository in the “AzureSQL-Schema” directory

Load

1. Loading data into SQL database
2. Set up our SQL database according to the ERD and DDL in the “AzureSQL-Schema” directory in our github repository
3. Each dataset
    a. Created a mount point in order to connect to our container that contains our clean data
    b. Read the dataset into a dataframe named “df”
    c. For each breakout table in the ERD
        i. Used distinct() to get the unique values to populate the breakout tables
        ii. Then renamed each column to match the column names within the SQL database
        iii. The following code is used to upload each breakout table to the SQL. The only things that changed for each set of code were the table name, and the name of the dataframe

```
server =
"gen10-data-fundamentals-22-05-sql-server.database.windows.net"
database = "capstoneGroup2Database"
user = "Christian"
password = "Str0ngP4ssw0rd!"
table = "dbo.Age"

age_df.write.format('jdbc').option("url",
```

```
f"jdbc:sqlserver://{server}:1433;databaseName={database};") \
    .mode("append") \
    .option("dbtable", table) \
    .option("user", user) \
    .option("password", password) \
    .option("driver", "com.microsoft.sqlserver.jdbc.SQLServerDriver") \
    .save()
```

d. For the InsuranceCoverage table, the Subgroup column has to be broken out into individual tables. First use df.drop() to drop the existing State column, then use the code below.

```
df = df.withColumn("Sex", when((df.Group == "By Gender"),df.Subgroup))

df = df.withColumn("Sex", lower(col("Sex")))

df = df.withColumn("Race", when((df.Group == "By Race/Hispanic ethnicity"),df.Subgroup))

df = df.withColumn("Age", when((df.Group == "By Age"),df.Subgroup))

df = df.withColumn("Education", when((df.Group == "By Education"),df.Subgroup))

df = df.withColumn("State", when((df.Group == "By State"),df.Subgroup))
```

e. To create each of the main tables we used joins on each of the breakout tables in order to get each ID. You can see the changes for each code in the exported databricks within our github. Below is one example of joining the breakout table with the main table.
    i. df = df.join(age_df, df.age == age_df.AgeLabel, "left")
f. Dropped the columns not in the SQL database. Below is an example used in the InsuranceCharges dataset.
    i. df.drop("age", "children", "region", "sex", "smoker", "timestamp", "SexLabel", "AgeLabel", "RegionLabel", "SmokerLabel", "ChildrenLabel")

g. Rename each of the columns as needed. Below is one example of a column being renamed in the InsuranceCharges dataset
    i. df = df.withColumnRenamed('charges', 'ChargeValue')
h. Load each table into SQL database. Below shows the code to load the InsuranceCharges table. Only change is the name of the table.

```
table = "dbo.InsuranceCharges"

df.write.format('jdbc').option("url",
f"jdbc:sqlserver://{server}:1433;databaseName={database};") \
    .mode("append") \
    .option("dbtable", table) \
    .option("user", user) \
    .option("password", password) \
    .option("driver", "com.microsoft.sqlserver.jdbc.SQLServerDriver") \
    .save()
```

4.

Conclusion

After following the process above step by step any user should be able to replicate the dataset which was used for data analysis in this project. We have displayed the process for creating a producer and consumer streams that allow us to transport our data to our SQL database. We have also displayed the process for extracting, transforming, and loading the three datasets regarding health insurance we have researched. Once the process is complete, the database should be fully populated with the data required to create visualizations.