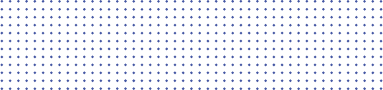
|  |  |  |  |
| --- | --- | --- | --- |
|  |  | Effective ETL Pipeline for Analyzing the Impact of Medicaid & Medicare Disbursement on Patient Satisfaction  **Learning Team 13: Ashek Ag Mohamed, Esneyder Gonzalez,**  **Isiasha Gordon, Bethel Ikejofor, Brian Noble** |  |
|  |  |  |  |
|  | | | |

|  |  |  |  |
| --- | --- | --- | --- |
|  | |  |  |
|  | TABLE OF CONTENTS  [1](#_Toc140329660)  [INTRODUCTION 3](#_Toc140329661)  [Data Procurement 6](#_Toc140329662)  [Data storage 8](#_Toc140329663)  [Data modeling 9](#_Toc140329664)  [ETL Pipeline 13](#_Toc140329665)  [Conclusion 17](#_Toc140329666) | |  |



|  |  |  |
| --- | --- | --- |
| INTRODUCTION |  | |
| Project Topic  President Lyndon B. Johnson signed into law the Social Security Act on July 30, 1965 which led to the Medicare and Medicaid programs. These two programs were established to provide health insurance to specific sets of the population. Congress has made various legislation changes to both programs that led to more people becoming eligible for them. While the federally funded Medicare program primarily serves individuals 65 years and older as well as some younger individuals with a disability, Medicaid is jointly funded by the federal government and individual states and primarily provides medical coverage to low-income individuals and families.  The future of the Medicare and Medicaid programs is increasingly in the news as some parties would like to expand their eligibility further as a vehicle toward a single-payer system while others advocate for cuts amid fiscal, political, and ideological concerns.  As data professionals and clinicians in this evolving healthcare landscape, we are particularly interested in analyzing payments made through these two programs and how the reimbursement rate affects the quality of care and patient satisfaction. By analyzing CMS payment data and physician ratings, the company can gain a deeper understanding of the healthcare industry and position itself as a leader in healthcare data analysis.    Client Analysis  The client provided two dashboards for guidance for the data analyst to meet business needs. The data engineer is thus tasked with building a data warehouse with these features in mind to provide the data analyst with the ability to build these dashboards. Key characteristics of the data warehouse include accuracy, reliability, scalability, performance, and maintainability.   1. For the first dashboard design:  * The data engineer needs to create transaction tables with fields for payment date, produce category, therapeutic area, originating state, and city. * They need to establish a filtering mechanism for each of these fields in the data warehouse. * They need to create an aggregation mechanism that can compute the total count or average amount of payments made. * They need to create a mechanism that can compute the total number of individuals covered by each payment category. * They need to create a mechanism that can compute the total payment amount received by the recipient state and city.  1. For the second dashboard design:  * The data engineer needs to create rating tables with fields for the rating date, rating platform, specialty type, hospital name, and doctor name. * They need to establish filtering mechanisms for rating date, rating platform, specialty type, hospital name, and doctor name in the data warehouse. * They need to create an aggregation mechanism that can aggregate ratings and specialties based on the selected filters. * They need to create a mechanism that can compute each doctor's average rating and total payment received. * They need to create a mechanism that can compute changes to each doctor's average rating over the specified semi-annual dates. * They need to create a mechanism that can compute overall changes to average ratings by time and platform.  1. In the data warehouse, the key features for the first dashboard design would look like this:  * A "Transaction" fact table with fields for payment date, product category, therapeutic area, payment amount, recipient state, and city. * Dimension tables for each filterable field (payment date, product category, therapeutic area, originating state, and city) that contain the unique values for each field and their respective IDs. These tables would be linked to the Transaction fact table using foreign keys. * A "Payment Coverage" fact table with fields for payment category and the number of individuals covered by that category. This table would be linked to the Transaction fact table using the payment category field. * A "Payment Location" fact table with fields for the total payment amount, recipient state, and city. This table would be linked to the Transaction fact table using the recipient state and city fields.  1. For the second dashboard design, the key features would look like this:  * A "Rating" fact table with fields for rating date, rating platform, specialty type, hospital name, doctor name, and rating score. * Dimension tables for each filterable field (rating date, rating platform, specialty type, hospital name, and doctor name) that contain the unique values for each field and their respective IDs. These tables would be linked to the Rating fact table using foreign keys. * A "Doctor Payment" fact table with fields for doctor name, total payment received, and average rating. This table would be linked to the Rating fact table using the doctor name field. * A "Doctor Rating Change" fact table with fields for doctor name, rating date, and rating change. This table would be linked to the Rating fact table using the doctor name and rating date fields. * An "Overall Rating Change" fact table with fields for rating platform, rating date, and overall rating change. This table would be linked to the Rating fact table using the rating platform and rating date fields.   Data Sources   |  |  | | --- | --- | | Date & Title | Description | | [2020 Research Payment Data](https://docs.google.com/document/u/0/d/1Jn5xRNl-kRcmkIAYzkDVr-zkADUzm4iUtWpMl0YZ0Zw/edit)\* | individual payments made with attributes such as hospital name, city, and address, type of procedure by the Center for Medicare and Medicaid Services | | [2021 Research Payment Data](https://docs.google.com/document/u/0/d/1Jn5xRNl-kRcmkIAYzkDVr-zkADUzm4iUtWpMl0YZ0Zw/edit)\* | individual payments made with attributes such as hospital name, city, and address, type of procedure by the Center for Medicare and Medicaid Services | | [2020 Ownership Payment Data](https://docs.google.com/document/u/0/d/1Jn5xRNl-kRcmkIAYzkDVr-zkADUzm4iUtWpMl0YZ0Zw/edit)\* | payments received by the doctor and their addresses, specialty, and other information such as the payment date by the Center for Medicare and Medicaid Services | | [2021 Ownership Payment Data](https://docs.google.com/document/u/0/d/1Jn5xRNl-kRcmkIAYzkDVr-zkADUzm4iUtWpMl0YZ0Zw/edit)\* | payments received by the doctor and their addresses, specialty, and other information such as the payment date by the Center for Medicare and Medicaid Services | | [2020 General Payment Data](https://docs.google.com/document/u/0/d/1Jn5xRNl-kRcmkIAYzkDVr-zkADUzm4iUtWpMl0YZ0Zw/edit)\* | All general (non-research, non-ownership related) payments from the 2020 program year | | [2021 General Payment Data](https://docs.google.com/document/u/0/d/1Jn5xRNl-kRcmkIAYzkDVr-zkADUzm4iUtWpMl0YZ0Zw/edit)\* | All general (non-research, non-ownership related) payments from the 2021 program year | | [Physician Profile Supplement](https://docs.google.com/document/u/0/d/1Jn5xRNl-kRcmkIAYzkDVr-zkADUzm4iUtWpMl0YZ0Zw/edit)\* | provides information about physicians who have received payments |   \*Each of the above links is to a data dictionary defining each data source to be used in the data warehouse.  I. Strengths:   * + Transparency: the open payments data provides transparency into the financial relationship between healthcare providers and manufacturers, which can help identify patient conflicts of interest   + Accessibility: the data is publicly available, making it easier for researchers and other stakeholders to analyze patterns and trends   + Comprehensive: the above datasets covers a broad range of payments in   + Standardized: that data is presented in a standardized format   + Accountability: hospital owner data can help promote accountability by providing information on the financial and legal relationships between hospitals and their owners   II. Weaknesses:   * + Lack of context: payment data does not provide contextual information about the patient population being served, such as demographic or medical history. This can make it difficult to draw meaningful conclusions about healthcare utilization or provider performance   + Incomplete Data: Some payments may not be reported accurately or may be missing   + Lack of Context: the data does not provide information on the quality of care provided by healthcare providers which could be an important favor to consider when evaluating the impact of financial relationships   + Limited Scope: hospital owner data only provides information on the ownership and control of hospitals, which may provide a complete picture of the overall healthcare system | |  |

|  |  |
| --- | --- |
| Data Procurement |  |

Data Requirements and Sources

* 1. Data consists of 8 flat file sets saved in CSV format.
  2. The following [data files](https://drive.google.com/drive/folders/1Z5j_WBqMy4JkWPNA66OEIFcn3_cspJR3) were needed:
     1. General Payment Data from [2020](https://openpaymentsdata.cms.gov/dataset/a08c4b30-5cf3-4948-ad40-36f404619019) and [2021](https://openpaymentsdata.cms.gov/dataset/0380bbeb-aea1-58b6-b708-829f92a48202)
     2. Research Payment Data from [2020](https://openpaymentsdata.cms.gov/dataset/9c248e7e-7c7f-478b-ab84-ce0919d72c1c) and [2021](https://openpaymentsdata.cms.gov/dataset/ce1d28dd-0094-5060-a036-580329439600)
     3. Ownership Payment Data from [2020](https://openpaymentsdata.cms.gov/dataset/a9a0bf48-6b96-4589-b4c2-3c5dcfbeaca2) and [2021](https://openpaymentsdata.cms.gov/dataset/b0c03b8d-06df-58f2-8ce2-4daeffee147e)
     4. [Physician Profile Supplemental data](https://openpaymentsdata.cms.gov/dataset/23160558-6742-54ff-8b9f-cac7d514ff4e) (supplemental)
     5. [Physician Ratings](https://data.cms.gov/provider-data/dataset/a174-a962)
  3. Purpose: To build an ETL pipeline that will be used by the data analysts responsible for creating a comprehensive dashboard for decision-making
  4. Objective: The above data requirements provide payment data for the analysis and determination of a possible  correlation between the payments and the ratings generated by patients

Data Collection

1. The data was downloaded from the CMS.gov website through an AWS Lambda function and uploaded to an AWS S3 bucket.
2. To assess the quality of the data and remove unusable columns, we generated a sample of 10,000 entries for each data set and excluded columns with at least 30% missing values as long as they are not needed for data analysis identified by our client.
3. Jupyter Notebook can be found [here](https://github.com/IsiashaG/ds4a-engineering_team13/tree/main)
4. The physician’s ratings were obtained from a provided CSV file sourced CMS.gov ([found here](https://data.cms.gov/provider-data/dataset/a174-a962))

Data Transformation and Integration

The following steps were taken to clean/process the data:

* Identify the data quality issues:
  + Began by assessing the quality of the data and identifying potential issues. These issues included missing values, duplicate entries, inconsistent formats, outliers, or incorrect data types.

* Handle missing data:
  + Complete Case Analysis. In this approach, columns that had a high percentage of missing values, greater than 30%, were entirely removed from the dataset. This method was used because the missing values were believed to be missing completely at random and the columns with missing values were not considered crucial for the analysis or did not contain substantial information

* Remove duplicates:
  + Duplicates can skew analysis and lead to incorrect conclusions. Identify and remove duplicate entries from the dataset.
  + There were no duplicates or significant outliers present in the data sets

* Address inconsistent formats:
  + The following data types were accessed and changed:
    - ID numbers were converted to integers. Generally it is more efficient to use integer data type for columns containing ID numbers with no decimals. The integer data type requires less storage space compared to float64, as it does not store decimal places. Additionally, integer operations tend to be faster and more efficient than floating-point operations.
    - Dates of payments were converted to “datetime” from strings. String data types for dates can limit the ability to perform date-specific operations and calculations.
    - remove any other unnecessary columns

* Inspect for data errors:
  + Outlier is in the data were assessed/viewed utilizing z-scores.
  + There were no outliers needing to be removed in the data sets
  + There were no duplicate entries in the datasets
  + Standardization of zip codes across data sets

* Validate data integrity:
  + The following step were taken to validate the data:
    - Data profiling
    - Data validation checks
      * format and data types checks
      * Consistency checks
  + CMS.gov is the official website of the Centers for Medicare & Medicaid Services (CMS), a federal agency in the U.S. Department of Health and Human Services. CMS is responsible for administering healthcare programs such as Medicare and Medicaid. The data provided on CMS.gov is sourced from various programs and initiatives and is intended to provide information on healthcare services, costs, providers, and quality measures.

|  |  |  |
| --- | --- | --- |
| Data storage |  | |
| Source Data Plan  **Source Data Storage**:   1. Process for data be storage and accessibility:    * 1. The raw payment data will be uploaded from the CMS.gov website through an AWS Lambda function.         1. We will apply our Python scripts saved in [GitHub.](https://github.com/IsiashaG/ds4a-engineering_team13/blob/main/Learning_Team_13_Review_1.ipynb)      2. The raw rating data will be manually uploaded from a personal computer      3. The refined data will be stored in S3 bucket. 2. Specifications:    1. Specific structure:       1. CSV files    2. Naming conventions:       1. 2020-general-date       2. 2021-general-date       3. 2020-ownership-date       4. 2021-ownership-date       5. 2020-research-date       6. 2021-research-date       7. physician-scores-date       8. Physician-profiles-date | |  |

|  |  |  |  |
| --- | --- | --- | --- |
| Data modeling | |  | |
| Resource Table   * The table below organizes all data resources for easy access and reference. * Data Dictionary: [click here](https://docs.google.com/spreadsheets/d/1xVG8cyZrqnFEoi-4yjJUwl3JnO87PKE6/edit?usp=sharing&ouid=105716959973116669495&rtpof=true&sd=true)  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | | **Resource Name** | **Type** | **Description** | **Format** | **Size** | **Source link** | **Storage place** | | 2020 General Payment Data | File | This data is from CMS.gov and provides all general (non-research, non-ownership related) payments from the 2020 program year | CSV | 3.7 GB | https://openpaymentsdata.cms.gov/dataset/a08c4b30-5cf3-4948-ad40-36f404619019 | S3 Bucket | | 2021 General Payment Data | File | This data is from CMS.gov and provides all general (non-research, non-ownership related) payments from the 2021 program year | CSV | 7.11 GB | https://openpaymentsdata.cms.gov/dataset/0380bbeb-aea1-58b6-b708-829f92a48202 | S3 Bucket | | 2020 Ownership Payment Data | File | This data is from CMS.gov and provides all ownership and investment payments from the 2020 program year | CSV | 1.6 MB | https://openpaymentsdata.cms.gov/dataset/a9a0bf48-6b96-4589-b4c2-3c5dcfbeaca2 | S3 Bucket | | 2021 Ownership Payment Data | File | This data is from CMS.gov and provides all ownership and investment payments from the 2021 program year | CSV | 1.3 MB | https://openpaymentsdata.cms.gov/dataset/b0c03b8d-06df-58f2-8ce2-4daeffee147e | S3 Bucket | | 2020 Research Payment Data | File | This data is from CMS.gov and provides research-related payments from the 2020  program year | CSV | 580.4 MB | https://openpaymentsdata.cms.gov/dataset/9c248e7e-7c7f-478b-ab84-ce0919d72c1c | S3 Bucket | | 2021 Research Payment Data | File | This data is from CMS.gov and provides research-related payments from the 2021  program year | CSV | 712 MB | https://openpaymentsdata.cms.gov/dataset/ce1d28dd-0094-5060-a036-580329439600 | S3 Bucket | | Physician Ratings | File | This file contains Merit-Based Incentive Payment System (MIPS) Final Scores and performance category scores for clinicians. For further details on 2021 MIPS scoring, see the 2021 Traditional MIPS Scoring Guide. | CSV | 51.4 MB | https://data.cms.gov/provider-data/search?theme=Doctors%20and%20clinicians | S3 Bucket | |  |  | **Total Capacity** |  | **12.1023 GB** |  |  |     Data Overview   1. The model will consist of two fact tables, physicians’ reviews and payments made for the services. Payments can be made to a hospital or directly to one or several physicians. The data source does not specify the physician portion of the payment when it is made to a hospital. 2. Some important facts inherent to the data sources to be considered::  * Some physicians are not affiliated to any hospital. * A single payment can be linked to multiple physicians, up to a maximum of six. * A single payment can be associated with a hospital or not. * Several specialties can be listed for a service when a payment is made to a hospital. * A physician can have one or more specialties. * Specialties are indicated in just one field concatenated by the pipe character (“|”). * There are five fields for specialties in the data source, but only the first one is used.  1. There are two situations to consider when interpreting the payment data:    1. When the Covered\_Recipient\_Type field is "Covered Recipient Teaching Hospital", the hospital receives the payment and one or more doctors are indicated as Principal Investigator 1, Principal Investigator 2, Principal Investigator 3, etc.    * Fields associated for Principal\_Investigator 1 are:      + Principal\_Investigator\_1\_Covered\_Recipient\_Type      + Principal\_Investigator\_1\_Profile\_ID      + Principal\_Investigator\_1\_NPI      + Principal\_Investigator\_1\_First\_Name      + Principal\_Investigator\_1\_Middle\_Name      + Principal\_Investigator\_1\_Last\_Name 2. When the Covered\_Recipient\_Type field is "Covered Recipient Physician", the physician receives direct payment and the information of the physician is indicated in the following fields    * Covered\_Recipient\_NPI    * Covered\_Recipient\_First\_Name    * Covered\_Recipient\_Middle\_Name    * Covered\_Recipient\_Last\_Name   IV. For each physician, geographic information is available:   * + City   + State   + zip code   + address   Facts & Dimensions   * At this point in our development, we elect to proceed with a Snowflake schema as our dimensional modeling technique. Thanks to its improved data normalization, this schema will allow us to reduce data redundancy in our pipeline while improving data integrity, scalability and maintainability. Given our limited budget and goal of remaining within the boundaries of the AWS free tier, we are attracted to the optimized storage attribute of this modeling technique. * Two fact tables:   + Reviews   + Payments * Nine dimension tables:   + City   + State   + Date   + Review platform   + Specialty   + Cover recipient type   + Hospital   + Physicians   + Category product therapeutic area. * Four relational tables between fact and dimension tables:   + Payment-Specialty   + Payment-Physician   + Payment-Hospital   + Payment-Category product therapeutic area   Schema   * A Snowflake scheme was used for data modeling with two fact tables and nine dimensions tables. * We have designed the diagram of our data model using [DBdiagram](https://dbdiagram.io/d/646696f1dca9fb07c45aa900).   A screenshot of a computer  Description automatically generated  Justification   1. We want to build an ETL pipeline that will serve as the basis allowing our client to design various dashboards and reports. Each table will use ID fields as primary keys due to integrity challenges in our data source. 2. Our client, the data analyst, needs to present our pipeline’s information in dashboards. Specifically, they would like to identify whether there is a correlation between payments made to physicians and the reviews they receive from patients. The information will be filtered in different ways, including, cities, states, time, therapeutic area, and specialty. 3. We will then have two fact tables: reviews and payments. The dimensions listed initially are connected with the two fact tables as shown in the diagram above in order to filter the information properly according to the requirement of the client.    1. The Date dimension was created to avoid calculating time fields during the filtering and analysis process.    2. The Specialty\_payment table is necessary because a payment can be associated with one or several specialties. Therefore, several records are required in Specialty\_payment when many specialties are associated with a single payment.    3. Since single payments are associated with  single cover\_type\_recipient fields, it is not necessary to have a relation table between payment and cover\_type\_recipient tables.    4. A Payment can be associated with none, a single, or several Product category or Therapeutic Area. Therefore, a relation table is required between the payment table and the product category table. | | |  |
|  |
|  |

|  |  |  |
| --- | --- | --- |
| ETL Pipeline |  | |
| Pipeline Accessibility   1. Data Warehouse    1. AWS login and credentials:       1. email: c1.de.team13@gmail.com       2. password: Correlation1! 2. Links to code    1. [GitHub](https://github.com/IsiashaG/ds4a-engineering_team13)       1. [Transformation code](https://github.com/IsiashaG/ds4a-engineering_team13/blob/main/Transformations_WIP.ipynb) | |  |

ETL Pipeline

A diagram of a software process

Description automatically generated

Extraction

1. First, we started by creating an AWS Lambda function. We configured the function to have appropriate permissions to access the necessary AWS services, such as S3. We also set the memory allocation and execution timeout according to our requirements.
2. We created a list of file names and another list of the corresponding download links from the CMS. We leveraged the `requests` library to handle the HTTP request. By using the `stream=True` parameter in the `requests.get` method, we enabled streaming mode, allowing us to download the file in chunks rather than loading it all at once.
3. We added a looping mechanism to download each of the files and assign them the corresponding name set above. Once the response was received for each of the files, we checked if the download was successful by verifying the HTTP status code. If the status code indicated a successful download (e.g., 200), we proceeded to save the data in a temporary file. We opened the temporary file in binary write mode and iterated through the response content using the `iter\_content` method. Each chunk of data was written to the temporary file, avoiding excessive memory consumption.
4. After saving the CSV data to the temporary file, we used the AWS SDK for Python (Boto3) to interact with the S3 service. We created an S3 client and specified the name of our S3 bucket.
5. Finally, we uploaded the temporary file to the S3 bucket using the `upload\_file` method provided by the S3 client. We passed the file path, bucket name, and desired file name as parameters to the method. The temporary file was then transferred to the specified S3 bucket.
6. Throughout the process, we handled potential errors and exceptions, ensuring that appropriate error messages were returned if any issues occurred. We also considered best practices for memory optimization in AWS Lambda, such as streaming the data instead of loading it entirely into memory.
7. By using AWS Lambda and the combination of `requests` library and Boto3, we were able to efficiently download CSV files directly from the CMS and upload them in the [extracted-data-swan](https://s3.console.aws.amazon.com/s3/buckets/extracted-data-swan?region=us-east-2) S3 bucket. However, we would like to note that this method does not return all the rows in each table as some of them contain millions of rows. For this project and with the purpose of remaining within the constraints of the AWS Free Tier plan, we understood and accepted that limitation. This approach allowed us to handle large files and avoid memory limitations often associated with traditional data processing methods.

Transformation

1. We created an AWS Lambda Function called [data-swan-transformed](https://sa-east-1.console.aws.amazon.com/lambda/home?region=sa-east-1#/functions/data-swan-transformed). That function is running a Python code that takes the files from the [extracted-data-swan](https://s3.console.aws.amazon.com/s3/buckets/extracted-data-swan?region=us-east-2) S3 bucket and transforms them in order to produce the CSV files with the schema required for our data warehouse.
   1. Those files represent our two fact tables as well as the dimension and relationship tables. After the transformation, the output files are uploaded into the [transformed-data-swan](https://s3.console.aws.amazon.com/s3/buckets/transformed-data-swan?region=us-east-2) S3 bucket.
2. The image below represents the Lambda function used in the Transformation step. It uses a trigger (to run the function after files have been uploaded into [extracted-data-swan](https://s3.console.aws.amazon.com/s3/buckets/extracted-data-swan?region=us-east-2) the S3 bucket), the Layer (running Python 3.10 with the Pandas package installed), and the maximum running time of 15 minutes.

A screenshot of a computer

Description automatically generated

1. The Transformation Python code mostly has operations of:

* deletion of unnecessary columns
* delete information not required (for example, information from other countries other than the United States)
* concatenation of dataframes
* dropping duplicates
* generation of id’s (incremental integer)
* exploding columns to get several rows from one cell.
* merge dataframes to get the corresponding id’s
* standardization of data
* A screen shot of a computer code

  Description automatically generatedA screen shot of a computer screen

  Description automatically generatedExamples:

1. Trigger for Loading Data

A screenshot of a computer

Description automatically generated

Load

1. We started by creating the IAM role to give access to users to load the data from the s3 bucket into Redshift.
   1. We then created a cluster.
2. We went into the management console and then went into Redshift.
   1. We created a cluster and made sure the region was set to the correct one.
   2. We specified the number of nodes and also specified the IAM role that we created at the beginning.
3. After creating the cluster we went ahead and created the database and the schema that will be used to create the tables.
   1. The tables that were created had to match the tables that were transformed in the transformation part.
4. After creating the tables we used the load data function to load each CSV file from the s3 bucket into the corresponding table that was created previously.
5. We then checked to make sure all the data was accurately loaded by running a SQL query to check the data and match it with the CSV files in the s3 bucket.

A screenshot of a computer

Description automatically generated

|  |
| --- |
| Conclusion |
| Benefits   * Several of us coming from a data science and data analysis background, we have enjoyed the process of designing an ETL pipeline. We have acquired a great amount of knowledge about data engineering, some of the tools and the implementation on AWS. * Transforming the files into output needed by the data warehouse was a challenging and vastly satisfying process. * The pipeline is scalable despite the 15-minute Lambda runtime limitation because additional memory can be purchased in an enterprise setting.   Limitations   * Our goal was to remain within the boundaries of the AWS Free Tier plan and not incur additional costs. However, an important part of the data analysis relates to the correlation between disbursed payments and physician ratings. These ratings are available from the Datashake API; access to which required paying that service. Had we had access to that data we could have incorporated it in our analysis instead of relying on physician score from the CMS site. * Relying on the AWS Free Tier plan, also limited our S3 bucket storage size. * The time constraints also limited our abilities to develop the pipeline as we would have hoped. * In the future, a future work could be to add data redundancy, a dashboard and have a fully functional app. * Several of us coming from a data science and data analysis background, we have enjoyed the process of designing an ETL pipeline. We have acquired a great amount of knowledge about data engineering, some of the tools and the implementation on AWS. Transforming the files into output needed by the data warehouse was a challenging and vastly satisfying process. | |