Cloud Technologies Assignment CSC1142-Group 11

<u>Spark powered insurance transparency pipeline</u>

1. Student Details:

ruchika.jha2@mail.dcu.ie [11141]

snehakrishnan.akavalapil2@mail.dcu.ie [13672]

reneemaria.simonrex2@mail.dcu.ie [11678]

chandana.daggupati2@mail.dcu.ie [10651]

Fig 1. A diagram representing the Spark powered Insurance Transparency Pipeline.

2. Link for the dataset and source code Git repository:

Dataset Link - https://transparency-in-coverage.uhc.com/

Git Repository Link - https://github.com/SnehaKrishnan96/Cloud_Assignment

3. Link to your video demonstration:

https://drive.google.com/file/d/1_b4TlxCkFUu2Ck-FsvcUAcWoqOP-Yvk9/view?usp=sharing

4. Introduction:

An extensive volume of data is generated in the insurance industry on a daily basis, which consist of details regarding the claims, policies, customer interactions, and the regulation records. But, the way this data is processed and analyzed is unclear and leads to inefficiencies, customer mistrust, and difficulties in following the rules and regulations. To overcome these problems, we suggest building a data pipeline using Apache Spark on Databricks. This system will process data faster, make it more accurate, and show clear results for key information. The aim is to build a simple, more reliable, and efficient insurance system.

Apache Spark is the core of this system, as it can handle large amounts of data quickly and efficiently. It is fast, scalable, and easy to connect with other tools. Databricks, an analytics platform built on Spark, makes it easier for teams to work together and manage complex data tasks. By adopting these tools, the pipeline can handle real time data analysis, work with large datasets, and easily connect to cloud services.

Motivation for the app:

The motivation behind the project came from the need to address the challenges in the insurance sector which includes,

- **Making Data Clear and Transparent:** Customers and governing bodies want to know how insurance companies handle and analyze data. The Spark- powered system shows exactly how the data is being processed and decisions are made, ensuring accountability.
- **Working Faster and Smarter:** Traditional methods of handling insurance data often struggle to handle the growing data with its volume and variety. Apache Spark uses powerful technology to quickly process large amounts of data, making the process faster and efficient.
- **Scalability and Cost-Effectiveness:** Cloud platforms like Databricks can adjust resources based on the workload. This flexibility can ensure that the system can handle big tasks without wasting money on unused resources.
- **Facilitating Compliance:** The system keeps detailed records of activities and creates accurate, real-time reports. This helps insurance companies follow regulations easily and avoid fines for non-compliance.

Choice of Spark Technology and Databricks:

We choose Apache Spark and Databricks because of their strong ability to handle large amounts of data in the cloud. Spark uses in-memory processing, which is much faster than the traditional systems that depend on the hard drives. This speed is especially useful for tasks like fraud detection, and processing claims, where quick insights are important.

Databricks works well with Spark by offering an easy-to-use, managed platform that connects well with major cloud providers like AWS, Azure, and Google Cloud. It also provides a shared workspace where data scientists, analysts, and engineers can work together to create, test, and run data systems more efficiently. Databricks automates much of the complex setup and maintenance that would typically be required for managing clusters. Using an auto-scaling facility, it can automatically adjust the size of the cluster based on the workload.

5. Related Work:

The existing systems and applications in the insurance domain are as follows:

1. **IBM Watson Analytics:**

   IBM Watson Analytics helps insurance companies to find patterns in data, like claims, fraud detection, and customer behaviour, by using automated predictive analysis. It is easy to use and offers insights driven by AI. But, this doesn't have the ability to handle very large datasets or process data in real-time.

2. **SAP Predictive Analytics:**

   SAP Predictive Analysis provides a platform for analyzing insurance data with predictive modeling and machine learning. SAP's systems are often designed to work only with their own technology, making it harder to connect with other cloud platforms.

3. **Oracle Insurance Analytics:**

   Oracle Insurance Analytics provides a platform for creating dashboards and reports for the insurance industry. But, it works for static reports and doesn't handle real-time data processing.

5.a. How does our solution differ from these?

The Spar powered insurance transparency pipeline differ from these systems by these features:

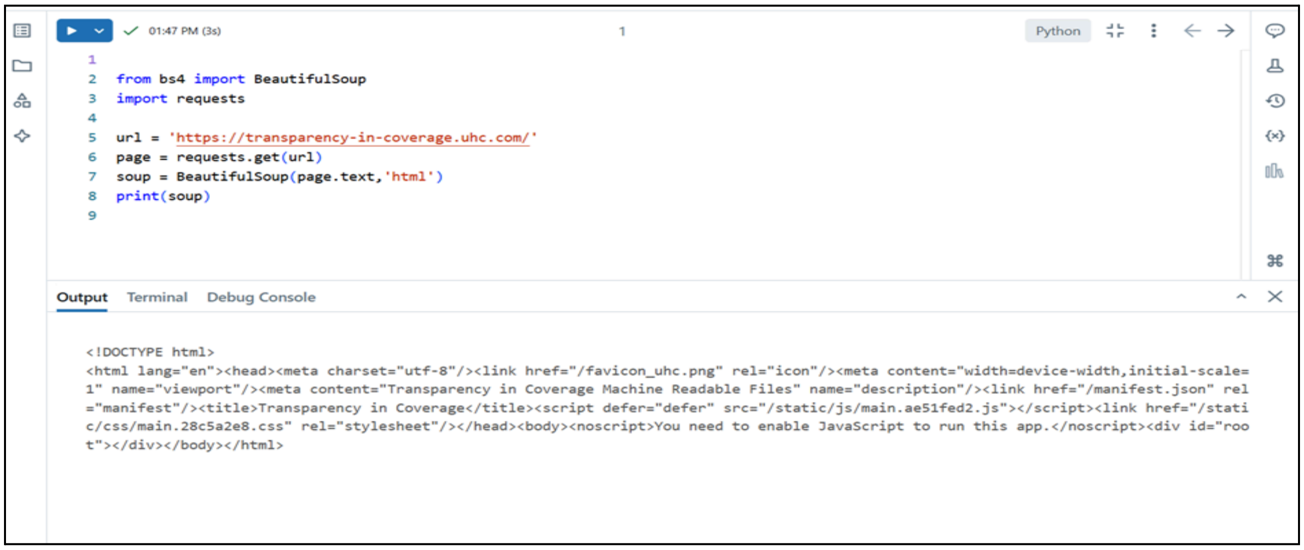
1. **Real Time Processing:** Unlike traditional systems that process data in batches, the Spark powered systems work in real-time, providing immediate insights for claims processing, fraud detection, and customer interactions.

2. **Open-Source Flexibility:** Using Apache Spark means the system isn't tied to a specific vendor, giving it the flexibility to work with different cloud services and existing insurance platforms.
3. **Scalable and Collaborative Cloud Platform:** Databricks integrates with the system, offering a flexible workspace where teams can easily work together on data tasks, increasing the creativity and productivity.
4. **Cost-Effectiveness:** By using open-source tools and adjusting cloud resources as needed, this system reduces the infrastructure costs compared to proprietary systems like SAP or Oracle.

6. Description of the Dataset:

6.a. Source of the data:

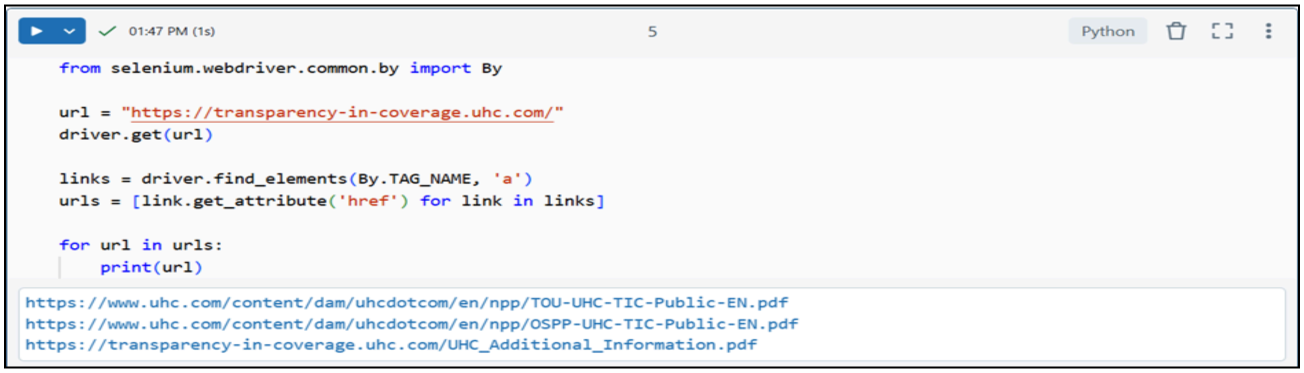
The data comes from the United Health Plan Transparency in Coverage Initiative (https://transparency-in-coverage.uhc.com/), which requires insurers to share pricing information. This dataset includes structured JSON files with detailed pricing for different healthcare services, such as in-network negotiated rates and out-of-network allowed amounts.



```python

from bs4 import BeautifulSoup
import requests

url = 'https://transparency-in-coverage.uhc.com/'
page = requests.get(url)
soup = BeautifulSoup(page.text,'html')
print(soup)

```

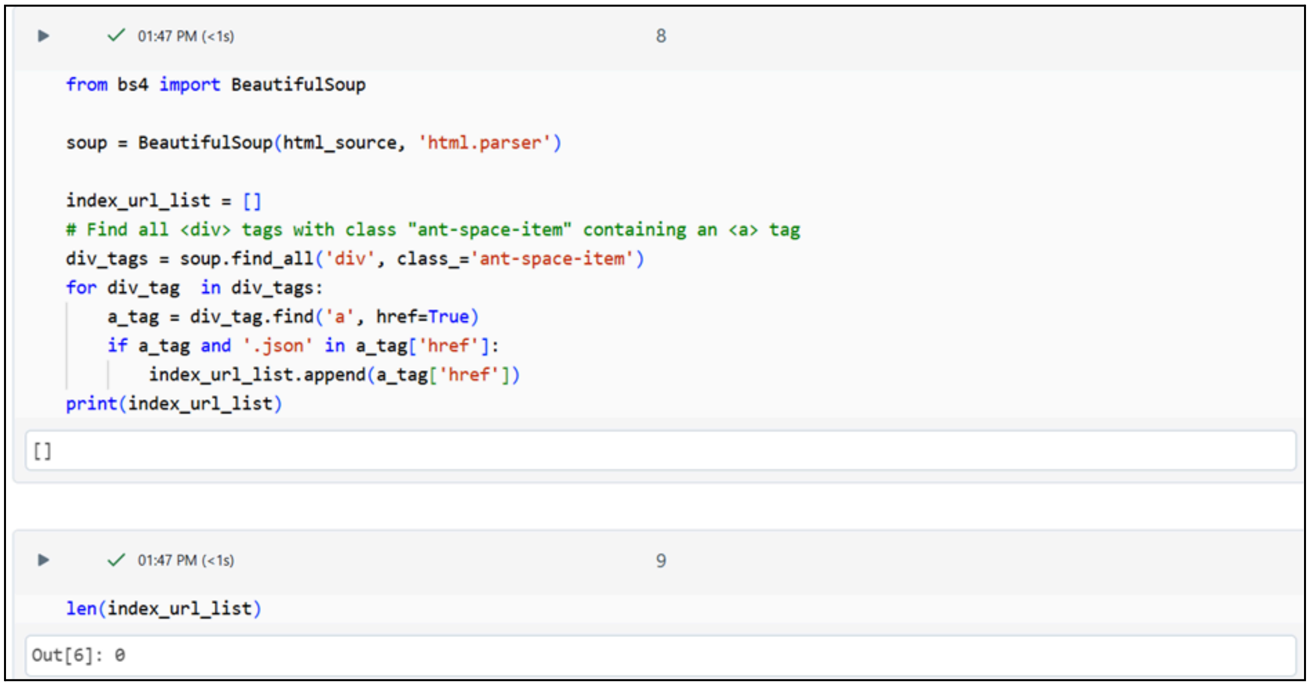
```
<!DOCTYPE html>
<html lang="en"><head><meta charset="utf-8"/><link href="/favicon_uhc.png" rel="icon"/><meta content="width=device-width,initial-scale=
1" name="viewport"/><meta content="Transparency in Coverage Machine Readable Files" name="description"/><link href="/manifest.json" rel
="manifest"/><title>Transparency in Coverage</title><script defer="defer" src="/static/js/main.ae51fed2.js"></script><link href="/stati
c/css/main.28c5a2e8.css" rel="stylesheet"/></head><body><noscript>You need to enable JavaScript to run this app.</noscript><div id="roo
t"></div></body></html>
```

Fig 2. Extraction process of Data using BeautifulSoup Python Library.



```python
from selenium.webdriver.common.by import By

url = "https://transparency-in-coverage.uhc.com/"
driver.get(url)

links = driver.find_elements(By.TAG_NAME, 'a')
urls = [link.get_attribute('href') for link in links]

for url in urls:
    print(url)
```

```
https://www.uhc.com/content/dam/uhcdotcom/en/npp/TOU-UHC-TIC-Public-EN.pdf
https://www.uhc.com/content/dam/uhcdotcom/en/npp/OSPP-UHC-TIC-Public-EN.pdf
https://transparency-in-coverage.uhc.com/UHC_Additional_Information.pdf
```



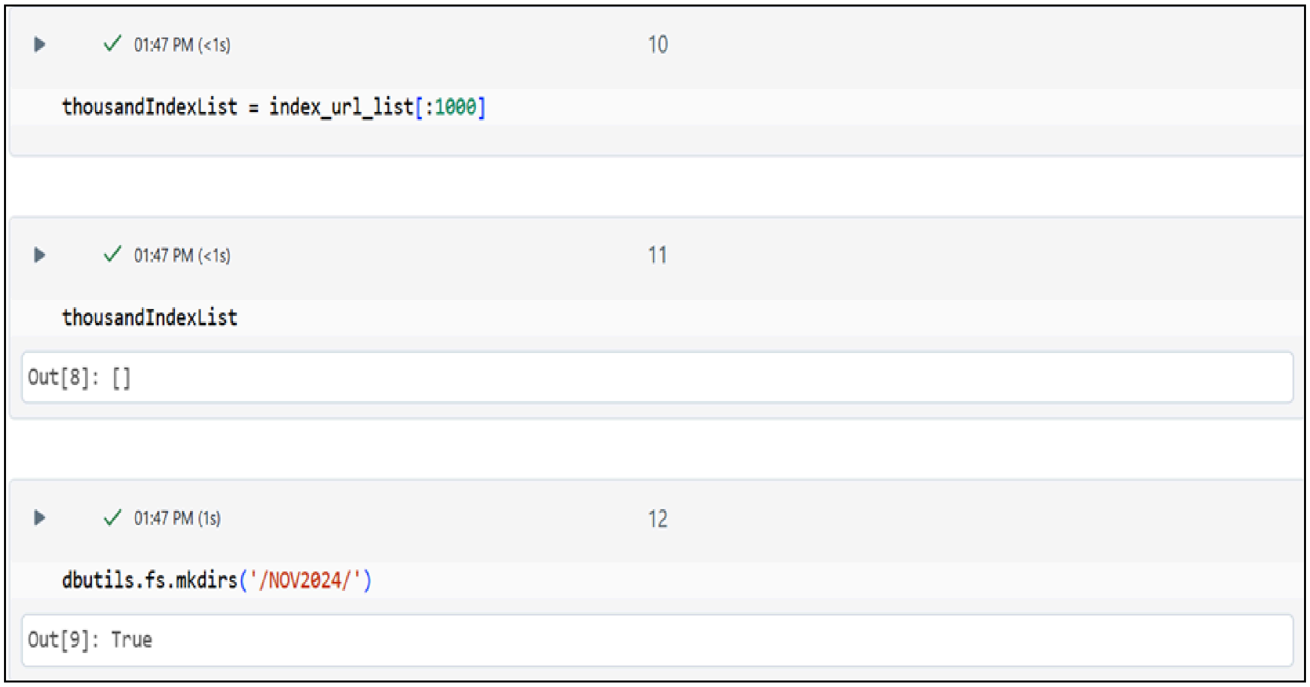
```
01:47 PM (<1s)    8

from bs4 import BeautifulSoup

soup = BeautifulSoup(html_source, 'html.parser')

index_url_list = []
# Find all <div> tags with class "ant-space-item" containing an <a> tag
div_tags = soup.find_all('div', class_='ant-space-item')
for div_tag  in div_tags:
    a_tag = div_tag.find('a', href=True)
    if a_tag and '.json' in a_tag['href']:
        index_url_list.append(a_tag['href'])
print(index_url_list)
```

```
[]
```

```
01:47 PM (<1s)    9

len(index_url_list)
```

```
Out[6]: 0
```

Fig 3. Understanding the structure of the JSON Files present in the datasource.



```
01:47 PM (<1s)    10

thousandIndexList = index_url_list[:1000]
```

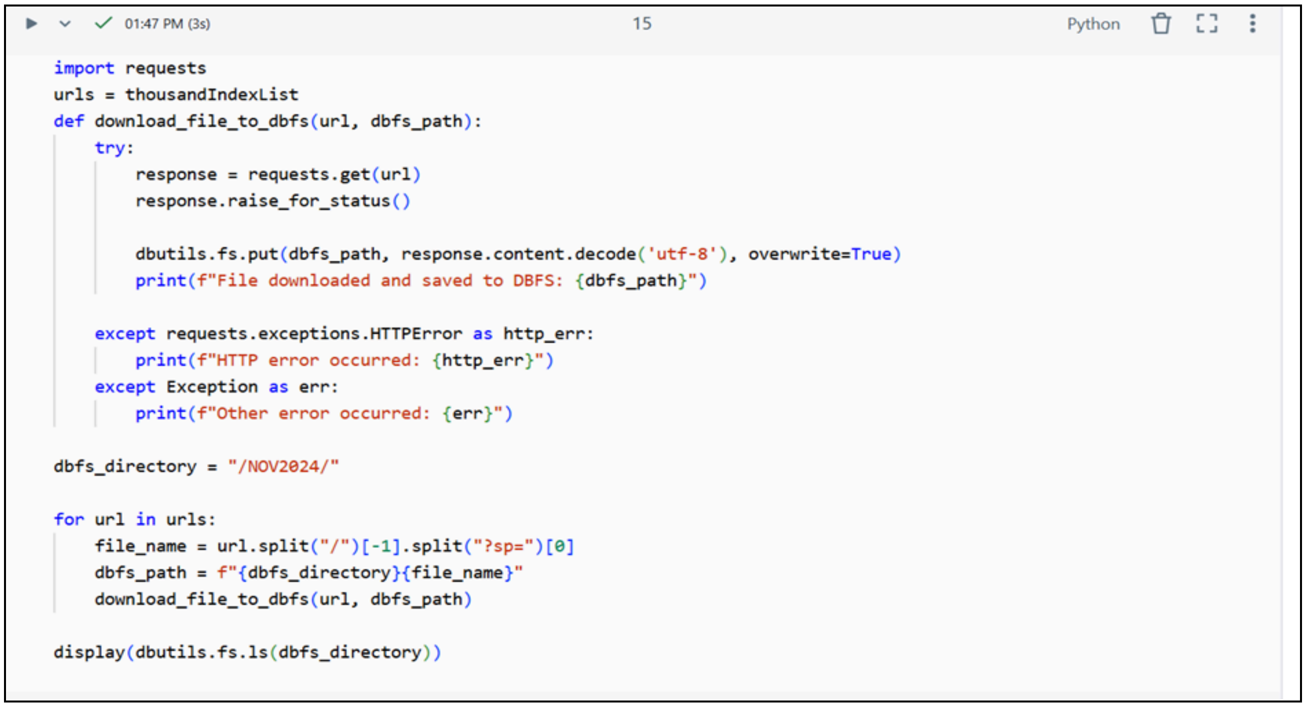
```
01:47 PM (<1s)    11

thousandIndexList
```

```
Out[8]: []
```

```
01:47 PM (1s)    12

dbutils.fs.mkdirs('/NOV2024/')
```

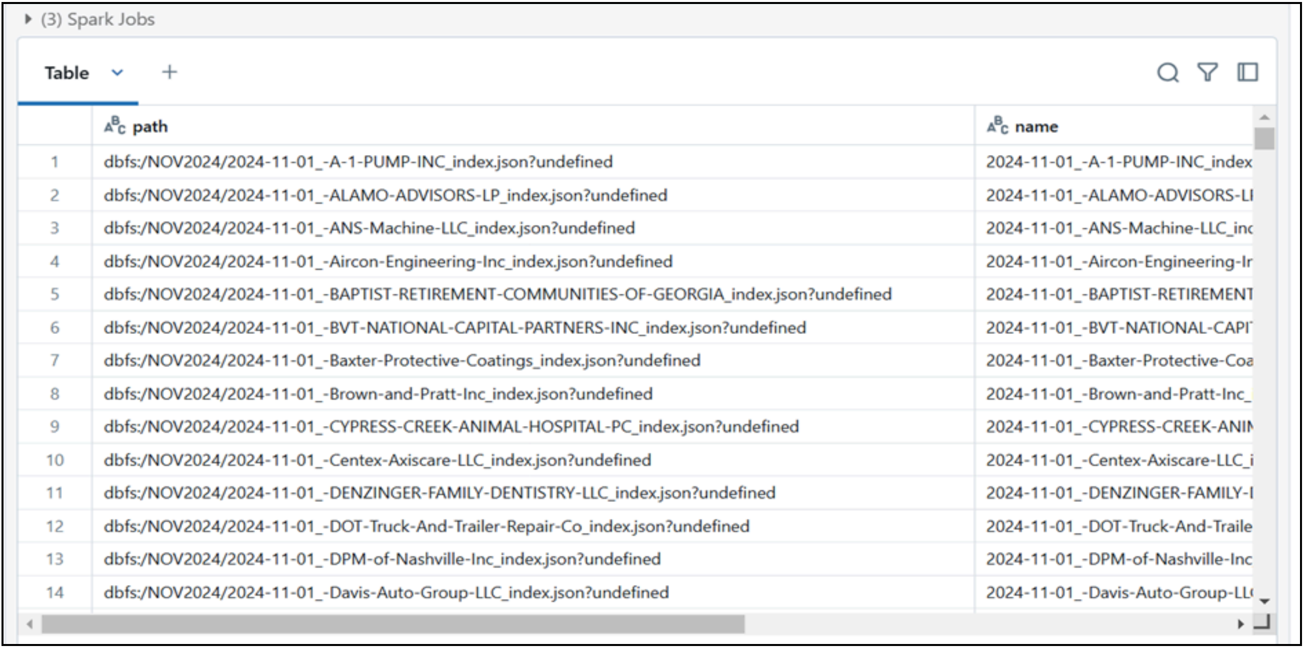
```
Out[9]: True
```

Fig 4. Creation of directory using ‘mkdir’.



```python
import requests
urls = thousandIndexList
def download_file_to_dbfs(url, dbfs_path):
    try:
        response = requests.get(url)
        response.raise_for_status()

        dbutils.fs.put(dbfs_path, response.content.decode('utf-8'), overwrite=True)
        print(f"File downloaded and saved to DBFS: {dbfs_path}")

    except requests.exceptions.HTTPError as http_err:
        print(f"HTTP error occurred: {http_err}")
    except Exception as err:
        print(f"Other error occurred: {err}")

dbfs_directory = "/NOV2024/"

for url in urls:
    file_name = url.split("/")[-1].split("?sp=")[0]
    dbfs_path = f"{dbfs_directory}{file_name}"
    download_file_to_dbfs(url, dbfs_path)

display(dbutils.fs.ls(dbfs_directory))
```

Fig 5. Downloading the JSON files from the source website.



	path	name
1	dbfs:/NOV2024/2024-11-01_-A-1-PUMP-INC_index.json?undefined	2024-11-01_-A-1-PUMP-INC_index
2	dbfs:/NOV2024/2024-11-01_-ALAMO-ADVISORS-LP_index.json?undefined	2024-11-01_-ALAMO-ADVISORS-L
3	dbfs:/NOV2024/2024-11-01_-ANS-Machine-LLC_index.json?undefined	2024-11-01_-ANS-Machine-LLC_inc
4	dbfs:/NOV2024/2024-11-01_-Aircon-Engineering-Inc_index.json?undefined	2024-11-01_-Aircon-Engineering-Ir
5	dbfs:/NOV2024/2024-11-01_-BAPTIST-RETIREMENT-COMMUNITIES-OF-GEORGIA_index.json?undefined	2024-11-01_-BAPTIST-RETIREMENT
6	dbfs:/NOV2024/2024-11-01_-BVT-NATIONAL-CAPITAL-PARTNERS-INC_index.json?undefined	2024-11-01_-BVT-NATIONAL-CAPI
7	dbfs:/NOV2024/2024-11-01_-Baxter-Protective-Coatings_index.json?undefined	2024-11-01_-Baxter-Protective-Coa
8	dbfs:/NOV2024/2024-11-01_-Brown-and-Pratt-Inc_index.json?undefined	2024-11-01_-Brown-and-Pratt-Inc_
9	dbfs:/NOV2024/2024-11-01_-CYPRESS-CREEK-ANIMAL-HOSPITAL-PC_index.json?undefined	2024-11-01_-CYPRESS-CREEK-ANI
10	dbfs:/NOV2024/2024-11-01_-Centex-Axiscare-LLC_index.json?undefined	2024-11-01_-Centex-Axiscare-LLC_i
11	dbfs:/NOV2024/2024-11-01_-DENZINGER-FAMILY-DENTISTRY-LLC_index.json?undefined	2024-11-01_-DENZINGER-FAMILY-
12	dbfs:/NOV2024/2024-11-01_-DOT-Truck-And-Trailer-Repair-Co_index.json?undefined	2024-11-01_-DOT-Truck-And-Traile
13	dbfs:/NOV2024/2024-11-01_-DPM-of-Nashville-Inc_index.json?undefined	2024-11-01_-DPM-of-Nashville-Inc
14	dbfs:/NOV2024/2024-11-01_-Davis-Auto-Group-LLC_index.json?undefined	2024-11-01_-Davis-Auto-Group-LL

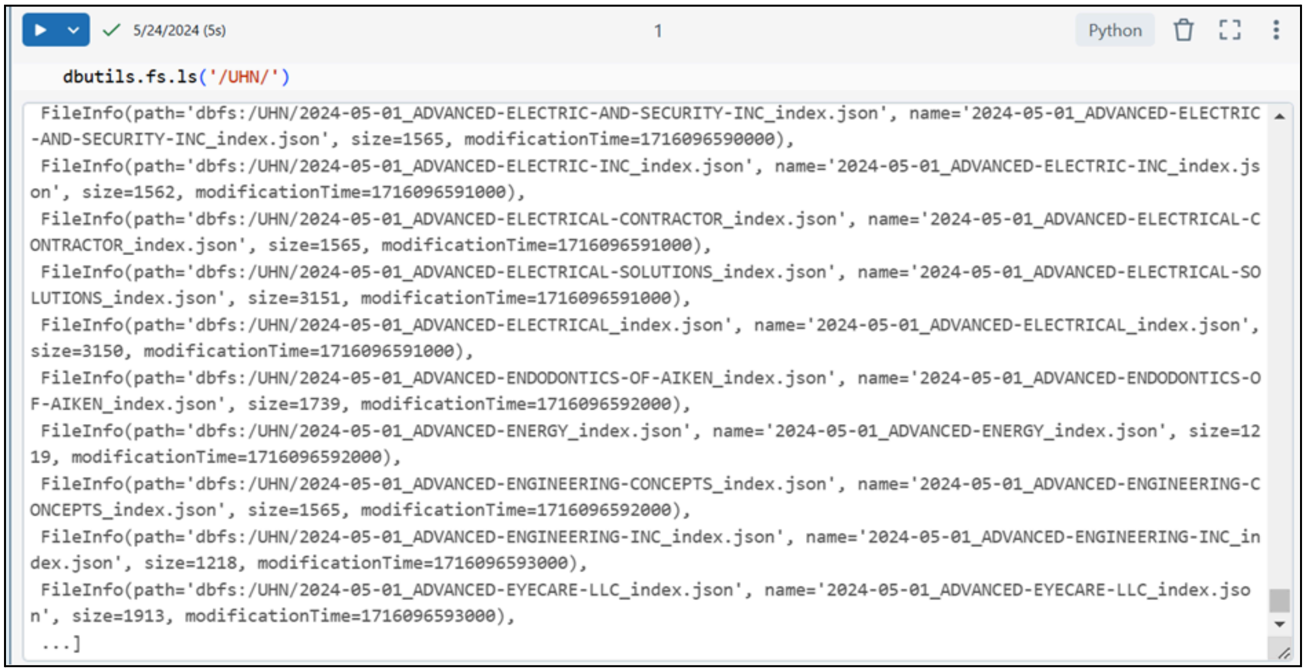
6.b. Process of Extraction/ Collection:

The data is extracted using a web scraping approach:

Step 1: A Python script which uses the BeautifulSoup library to extract the HTML content from the provided URL. This helps the script to find the links to the JSON files.

Step 2: Once the links are found, the script downloads the JSON files and saves them to a cloud storage for further analysis.

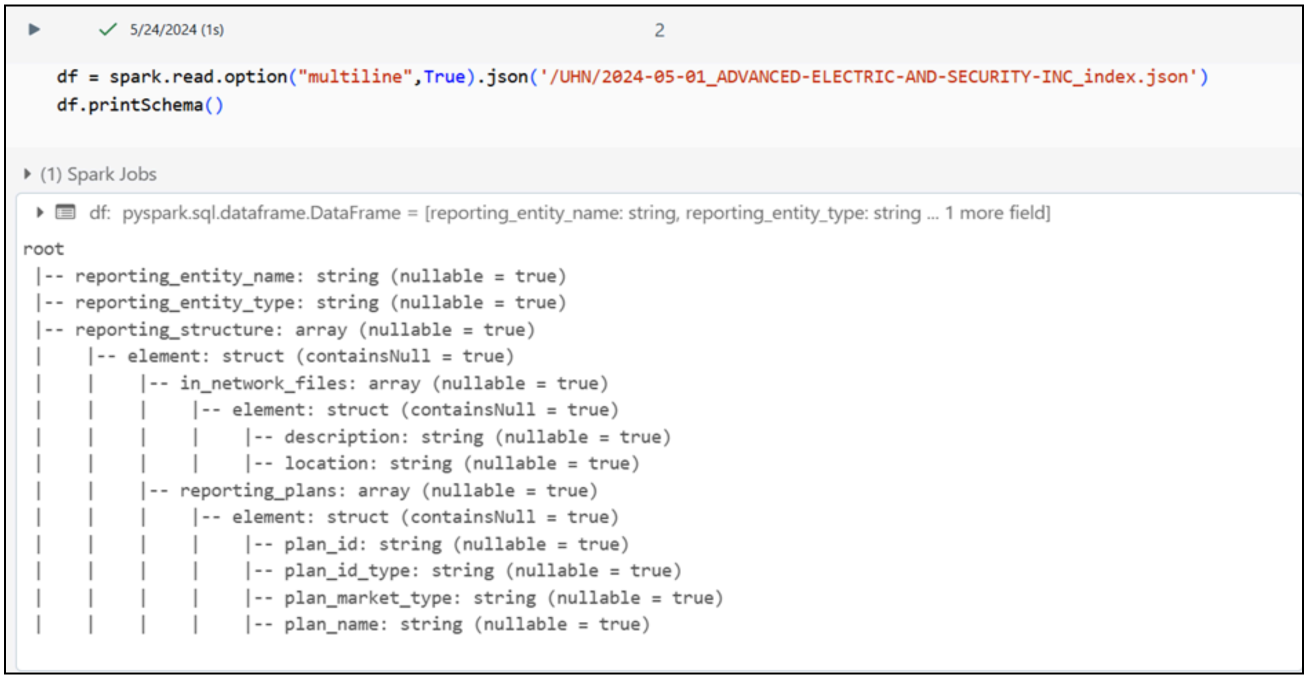
Step 3: Apache Spark is used to read and process the JSON files. These files are then loaded into a distributed system where data can be analyzed and transformed.



```
dbutils.fs.ls('/UHN/')
```

```
 FileInfo(path='dbfs:/UHN/2024-05-01_ADVANCED-ELECTRIC-AND-SECURITY-INC_index.json', name='2024-05-01_ADVANCED-ELECTRIC
-AND-SECURITY-INC_index.json', size=1565, modificationTime=1716096590000),
 FileInfo(path='dbfs:/UHN/2024-05-01_ADVANCED-ELECTRIC-INC_index.json', name='2024-05-01_ADVANCED-ELECTRIC-INC_index.js
on', size=1562, modificationTime=1716096591000),
 FileInfo(path='dbfs:/UHN/2024-05-01_ADVANCED-ELECTRICAL-CONTRACTOR_index.json', name='2024-05-01_ADVANCED-ELECTRICAL-C
ONTRACTOR_index.json', size=1565, modificationTime=1716096591000),
 FileInfo(path='dbfs:/UHN/2024-05-01_ADVANCED-ELECTRICAL-SOLUTIONS_index.json', name='2024-05-01_ADVANCED-ELECTRICAL-SO
LUTIONS_index.json', size=3151, modificationTime=1716096591000),
 FileInfo(path='dbfs:/UHN/2024-05-01_ADVANCED-ELECTRICAL_index.json', name='2024-05-01_ADVANCED-ELECTRICAL_index.json',
size=3150, modificationTime=1716096591000),
 FileInfo(path='dbfs:/UHN/2024-05-01_ADVANCED-ENDODONTICS-OF-AIKEN_index.json', name='2024-05-01_ADVANCED-ENDODONTICS-O
F-AIKEN_index.json', size=1739, modificationTime=1716096592000),
 FileInfo(path='dbfs:/UHN/2024-05-01_ADVANCED-ENERGY_index.json', name='2024-05-01_ADVANCED-ENERGY_index.json', size=12
19, modificationTime=1716096592000),
 FileInfo(path='dbfs:/UHN/2024-05-01_ADVANCED-ENGINEERING-CONCEPTS_index.json', name='2024-05-01_ADVANCED-ENGINEERING-C
ONCEPTS_index.json', size=1565, modificationTime=1716096592000),
 FileInfo(path='dbfs:/UHN/2024-05-01_ADVANCED-ENGINEERING-INC_index.json', name='2024-05-01_ADVANCED-ENGINEERING-INC_in
dex.json', size=1218, modificationTime=1716096593000),
 FileInfo(path='dbfs:/UHN/2024-05-01_ADVANCED-EYECARE-LLC_index.json', name='2024-05-01_ADVANCED-EYECARE-LLC_index.jso
n', size=1913, modificationTime=1716096593000),
 ...]
```

Fig 6. Listing the files available in the directory.



```
df = spark.read.option("multiline",True).json('/UHN/2024-05-01_ADVANCED-ELECTRIC-AND-SECURITY-INC_index.json')
df.printSchema()
```

```
df:  pyspark.sql.dataframe.DataFrame = [reporting_entity_name: string, reporting_entity_type: string ... 1 more field]
```

```
root
 |-- reporting_entity_name: string (nullable = true)
 |-- reporting_entity_type: string (nullable = true)
 |-- reporting_structure: array (nullable = true)
 |    |-- element: struct (containsNull = true)
 |    |    |-- in_network_files: array (nullable = true)
 |    |    |    |-- element: struct (containsNull = true)
 |    |    |    |    |-- description: string (nullable = true)
 |    |    |    |    |-- location: string (nullable = true)
 |    |    |-- reporting_plans: array (nullable = true)
 |    |    |    |-- element: struct (containsNull = true)
 |    |    |    |    |-- plan_id: string (nullable = true)
 |    |    |    |    |-- plan_id_type: string (nullable = true)
 |    |    |    |    |-- plan_market_type: string (nullable = true)
 |    |    |    |    |-- plan_name: string (nullable = true)
```

Fig 7. Checking the multiline structure of a JSON file.

6.c Data Pre-Processing - Preparation and Cleaning:

The raw data from the Transparency in coverage repository often has issues like errors, missing values, and duplicate information. To improve its quality, the following steps are taken:

1. Schema Validation: Spark checks the structure of the JSON files to make sure all records follow the same format.

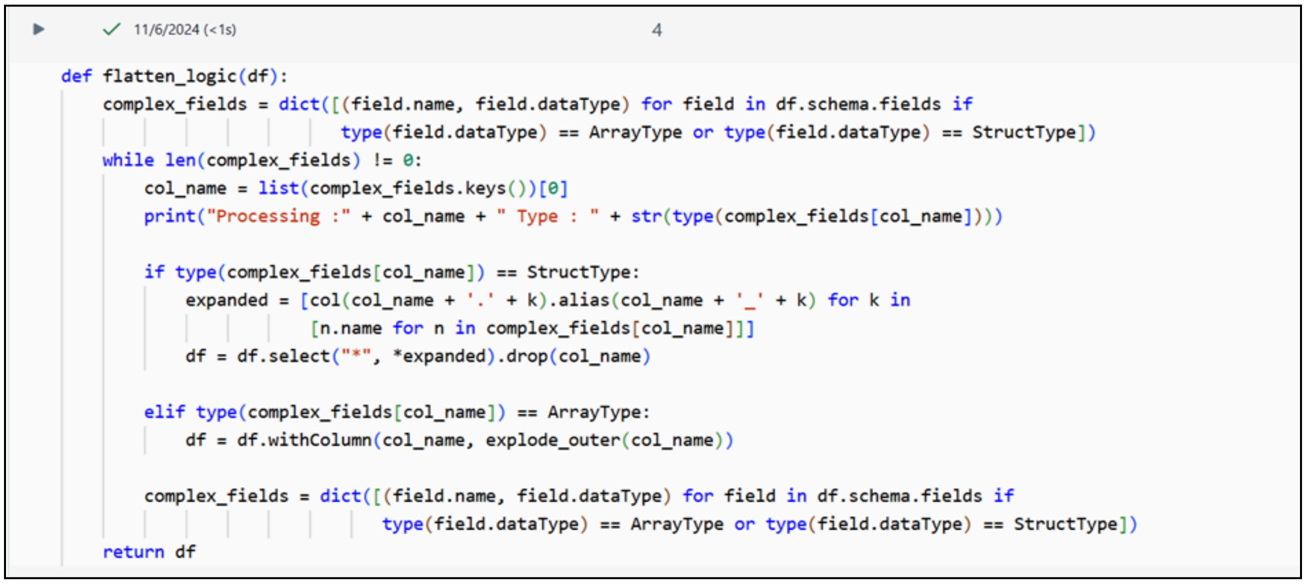
2. Data Cleaning: Missing values are filled in or removed based on set rules. Duplicate records are found and removed.

3. Normalization: Complex data in the JSON files is simplified into a table-like format to make it easier to analyze and use.

4. Data Transformation: Fields like dates and numbers are standardized to ensure consistency and compatibility with analysis tools.

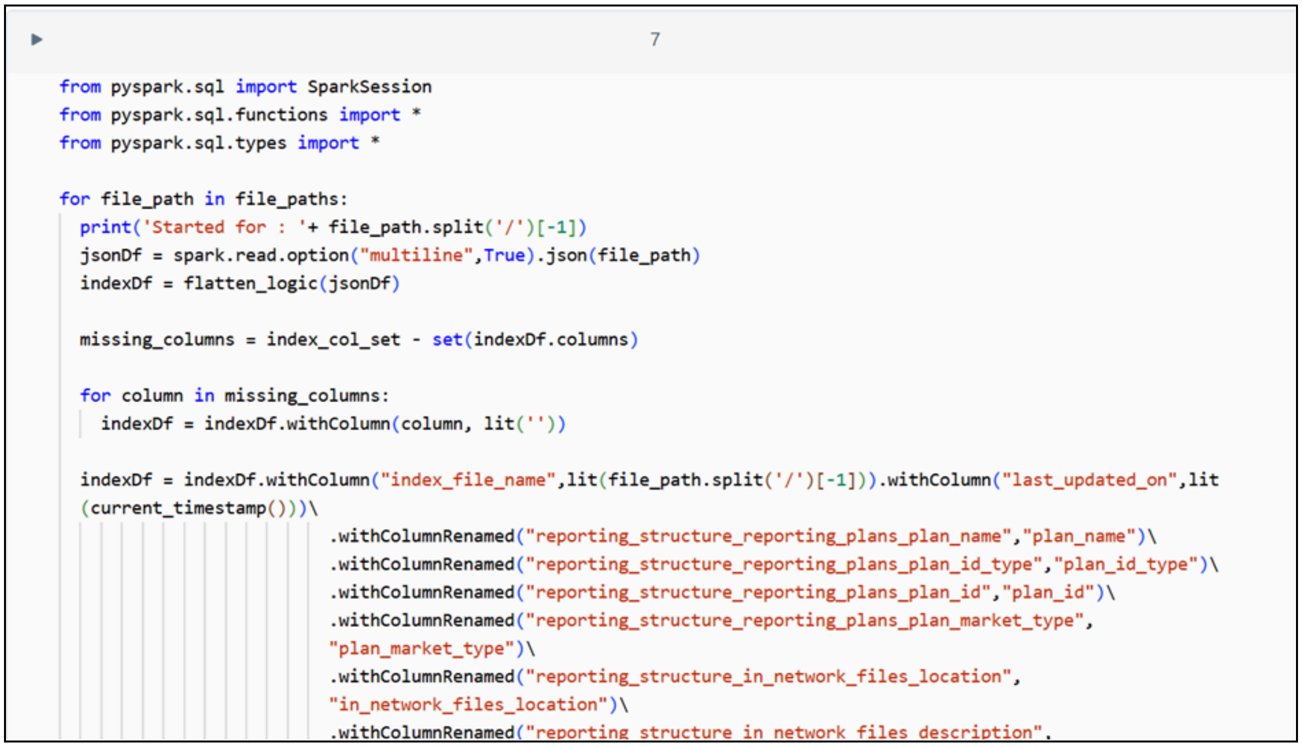
5. Partitioning and Indexing: The data is divided into parts and indexed in Spark to make searches faster and improve performance for real-time analysis.

This cleaning process ensures the data is clean, organized, and ready for accurate analysis, helping the transparency system provide useful insights.

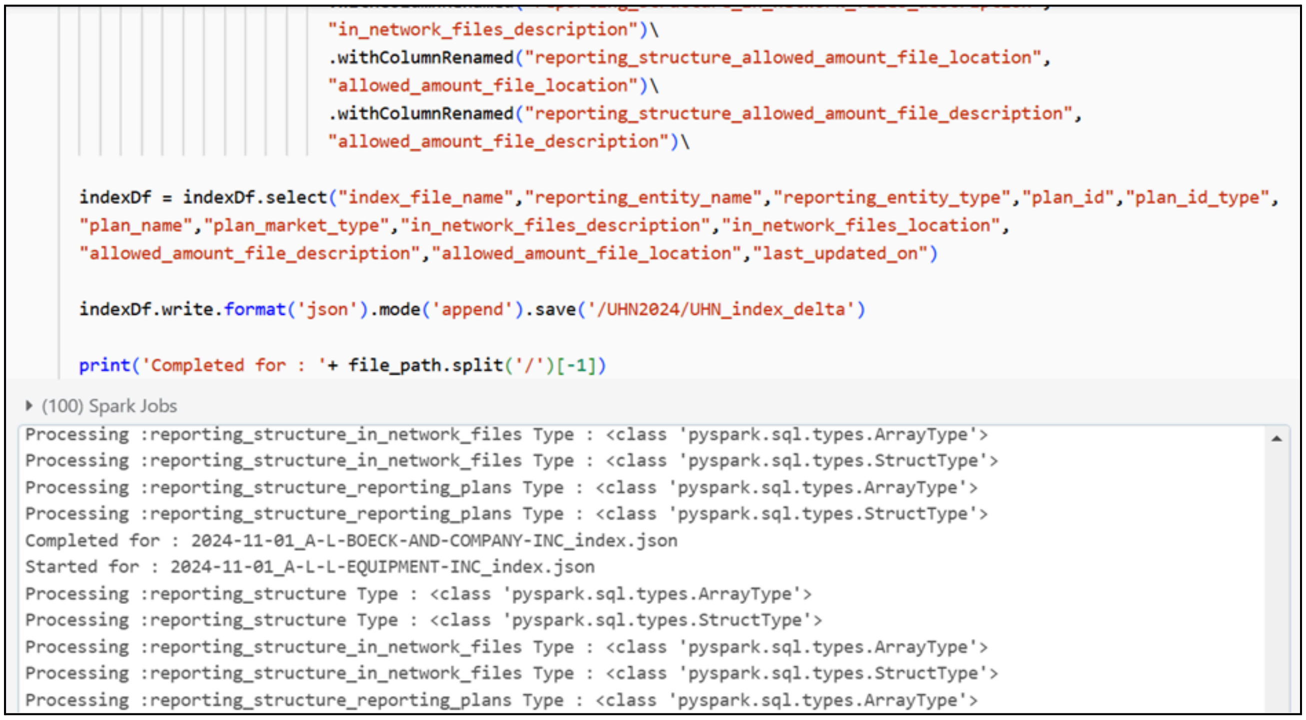

11/6/2024 (<1s) 4

```
def flatten_logic(df):
    complex_fields = dict([(field.name, field.dataType) for field in df.schema.fields if
                           type(field.dataType) == ArrayType or type(field.dataType) == StructType])
    while len(complex_fields) != 0:
        col_name = list(complex_fields.keys())[0]
        print("Processing :" + col_name + " Type : " + str(type(complex_fields[col_name])))

        if type(complex_fields[col_name]) == StructType:
            expanded = [col(col_name + '.' + k).alias(col_name + '_' + k) for k in
                        [n.name for n in complex_fields[col_name]]]
            df = df.select("*", *expanded).drop(col_name)

        elif type(complex_fields[col_name]) == ArrayType:
            df = df.withColumn(col_name, explode_outer(col_name))

        complex_fields = dict([(field.name, field.dataType) for field in df.schema.fields if
                               type(field.dataType) == ArrayType or type(field.dataType) == StructType])
    return df
```

Fig 8. Flattening Logic



```
from pyspark.sql import SparkSession
from pyspark.sql.functions import *
from pyspark.sql.types import *

for file_path in file_paths:
  print('Started for : '+ file_path.split('/')[-1])
  jsonDf = spark.read.option("multiline",True).json(file_path)
  indexDf = flatten_logic(jsonDf)

  missing_columns = index_col_set - set(indexDf.columns)

  for column in missing_columns:
    indexDf = indexDf.withColumn(column, lit(''))

  indexDf = indexDf.withColumn("index_file_name",lit(file_path.split('/')[-1])).withColumn("last_updated_on",lit
  (current_timestamp()))\
                        .withColumnRenamed("reporting_structure_reporting_plans_plan_name","plan_name")\
                        .withColumnRenamed("reporting_structure_reporting_plans_plan_id_type","plan_id_type")\
                        .withColumnRenamed("reporting_structure_reporting_plans_plan_id","plan_id")\
                        .withColumnRenamed("reporting_structure_reporting_plans_plan_market_type",
                        "plan_market_type")\
                        .withColumnRenamed("reporting_structure_in_network_files_location",
                        "in_network_files_location")\
                        .withColumnRenamed("reporting_structure_in_network_files_description",
```

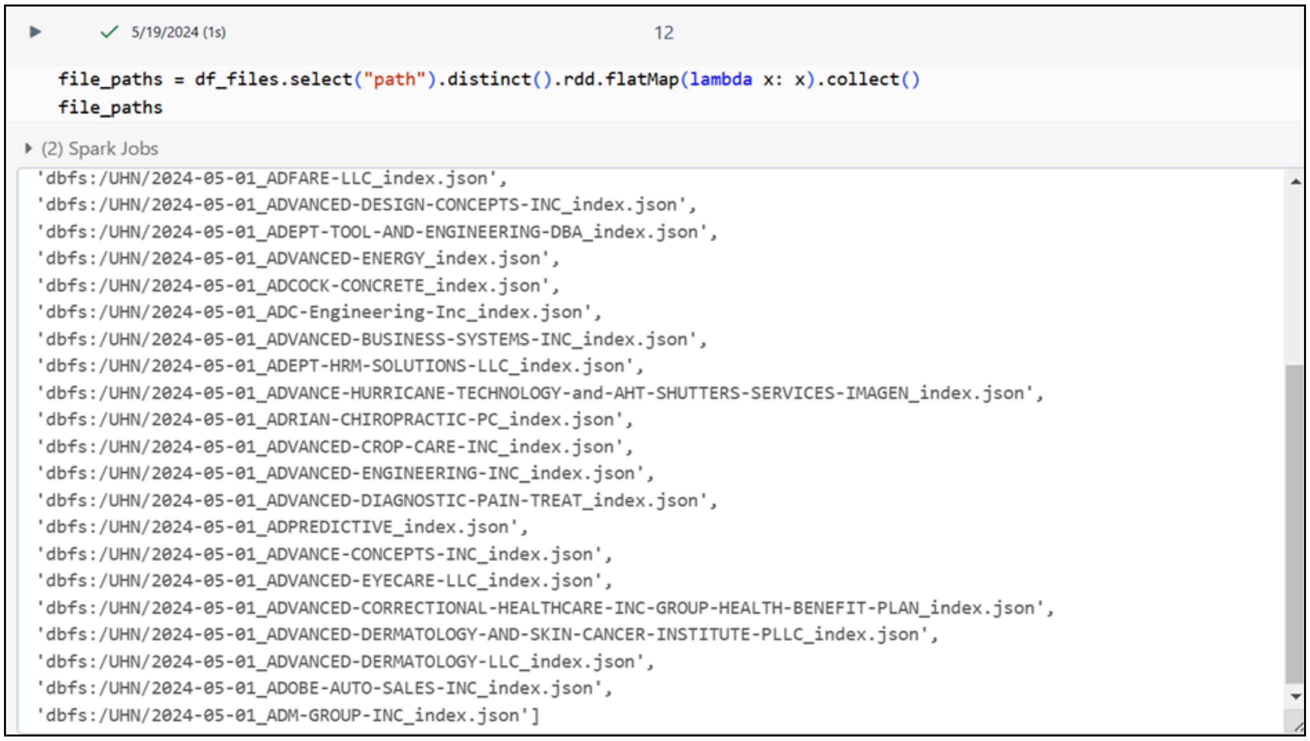


```
                        "in_network_files_description")\
                        .withColumnRenamed("reporting_structure_allowed_amount_file_location",
                        "allowed_amount_file_location")\
                        .withColumnRenamed("reporting_structure_allowed_amount_file_description",
                        "allowed_amount_file_description")\

  indexDf = indexDf.select("index_file_name","reporting_entity_name","reporting_entity_type","plan_id","plan_id_type",
  "plan_name","plan_market_type","in_network_files_description","in_network_files_location",
  "allowed_amount_file_description","allowed_amount_file_location","last_updated_on")

  indexDf.write.format('json').mode('append').save('/UHN2024/UHN_index_delta')

  print('Completed for : '+ file_path.split('/')[-1])
```

▸ (100) Spark Jobs

```
Processing :reporting_structure_in_network_files Type : <class 'pyspark.sql.types.ArrayType'>
Processing :reporting_structure_in_network_files Type : <class 'pyspark.sql.types.StructType'>
Processing :reporting_structure_reporting_plans Type : <class 'pyspark.sql.types.ArrayType'>
Processing :reporting_structure_reporting_plans Type : <class 'pyspark.sql.types.StructType'>
Completed for : 2024-11-01_A-L-BOECK-AND-COMPANY-INC_index.json
Started for : 2024-11-01_A-L-L-EQUIPMENT-INC_index.json
Processing :reporting_structure Type : <class 'pyspark.sql.types.ArrayType'>
Processing :reporting_structure Type : <class 'pyspark.sql.types.StructType'>
Processing :reporting_structure_in_network_files Type : <class 'pyspark.sql.types.ArrayType'>
Processing :reporting_structure_in_network_files Type : <class 'pyspark.sql.types.StructType'>
Processing :reporting_structure_reporting_plans Type : <class 'pyspark.sql.types.ArrayType'>
```

Fig 9. Indexing the Column names available in JSON Files.



```
5/19/2024 (1s)                                    12

file_paths = df_files.select("path").distinct().rdd.flatMap(lambda x: x).collect()
file_paths

(2) Spark Jobs
'dbfs:/UHN/2024-05-01_ADFARE-LLC_index.json',
'dbfs:/UHN/2024-05-01_ADVANCED-DESIGN-CONCEPTS-INC_index.json',
'dbfs:/UHN/2024-05-01_ADEPT-TOOL-AND-ENGINEERING-DBA_index.json',
'dbfs:/UHN/2024-05-01_ADVANCED-ENERGY_index.json',
'dbfs:/UHN/2024-05-01_ADCOCK-CONCRETE_index.json',
'dbfs:/UHN/2024-05-01_ADC-Engineering-Inc_index.json',
'dbfs:/UHN/2024-05-01_ADVANCED-BUSINESS-SYSTEMS-INC_index.json',
'dbfs:/UHN/2024-05-01_ADEPT-HRM-SOLUTIONS-LLC_index.json',
'dbfs:/UHN/2024-05-01_ADVANCE-HURRICANE-TECHNOLOGY-and-AHT-SHUTTERS-SERVICES-IMAGEN_index.json',
'dbfs:/UHN/2024-05-01_ADRIAN-CHIROPRACTIC-PC_index.json',
'dbfs:/UHN/2024-05-01_ADVANCED-CROP-CARE-INC_index.json',
'dbfs:/UHN/2024-05-01_ADVANCED-ENGINEERING-INC_index.json',
'dbfs:/UHN/2024-05-01_ADVANCED-DIAGNOSTIC-PAIN-TREAT_index.json',
'dbfs:/UHN/2024-05-01_ADPREDICTIVE_index.json',
'dbfs:/UHN/2024-05-01_ADVANCE-CONCEPTS-INC_index.json',
'dbfs:/UHN/2024-05-01_ADVANCED-EYECARE-LLC_index.json',
'dbfs:/UHN/2024-05-01_ADVANCED-CORRECTIONAL-HEALTHCARE-INC-GROUP-HEALTH-BENEFIT-PLAN_index.json',
'dbfs:/UHN/2024-05-01_ADVANCED-DERMATOLOGY-AND-SKIN-CANCER-INSTITUTE-PLLC_index.json',
'dbfs:/UHN/2024-05-01_ADVANCED-DERMATOLOGY-LLC_index.json',
'dbfs:/UHN/2024-05-01_ADOBE-AUTO-SALES-INC_index.json',
'dbfs:/UHN/2024-05-01_ADM-GROUP-INC_index.json']
```

Fig 10. Defining the Path for the JSON Files.

7. Description of Data processing:

The data processing pipeline is the main part of the Spark-powered insurance transparency application. The steps involved are as follows:

1. Ingestion (Loading the data in): Spark uses its tools to quickly load JSON files by reading them in pieces across multiple machines. It automatically figures out the structure of the data. The loaded data is stored in a cloud storage for further processing.

2. Transformation (Preparing the data): Complex data is broken into simple tables to make it easier to work with. Unwanted information is removed and useful details like plan names, customer names are extracted.

3. Loading (Using the data): The processed data is stored and extracted as a CSV file for further processing or analyzing. Using spark streaming the data can be sent to other systems for real-time analysis. Tools like Power BI can be used to turn the data into easy-to-read charts and reports.

4. Monitoring and Optimization (Keeping the system efficient): Spark's tools check how well the system is running. Depending on the workload to save cost, the computing power is scaled up or down. To handle errors, the system has features to automatically retry tasks or log issues.

By following these steps, the pipeline ensures that the data is accurate, up-to-date, and ready to provide valuable insights that improve transparency and efficiency in the insurance industry.

7.a. Query or code (Link to the file on the Repository):

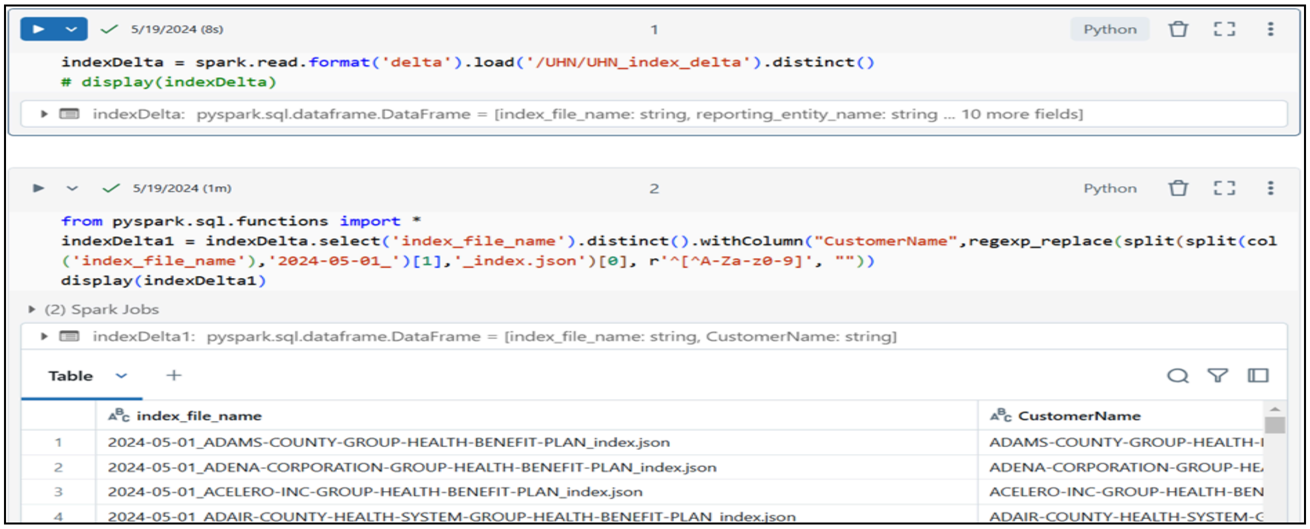
https://github.com/SnehaKrishnan96/Cloud_Assignment

8. Development of the pipeline:

The Spark-powered insurance transparency pipeline has three main parts: the data extraction tool, the data processing framework, and the interface for result visualization.

8.a. Data Extraction tool: The data extraction tool uses Python and Apache Spark to collect and handle data efficiently. Web scraping can be done using a python library called BeautifulSoup, which is used to go through a website, find the required information, and locate links to JSON files. The tool automatically downloads the JSON files and saves them in a cloud storage location. Spark reads the downloaded JSON files quickly by splitting the work across multiple machines. This automated process makes it easy to handle large amounts of insurance data and keeps the workflow smooth and scalable.

8.b. Data Processing tool: Apache Spark is the core of the data processing system, transforming raw data into useful insights. Spark automatically checks the structure of the data to ensure it is consistent. The data is cleaned, organized, and transformed into a format suitable for analysis. Spark processes data in memory, which speeds up the tasks, even with large and complex insurance datasets. Spark streaming analyzes data as it comes in, providing instant insights for fraud detection. Databricks adds more power to the process with features like shared workspaces, version control, and efficient resource management.



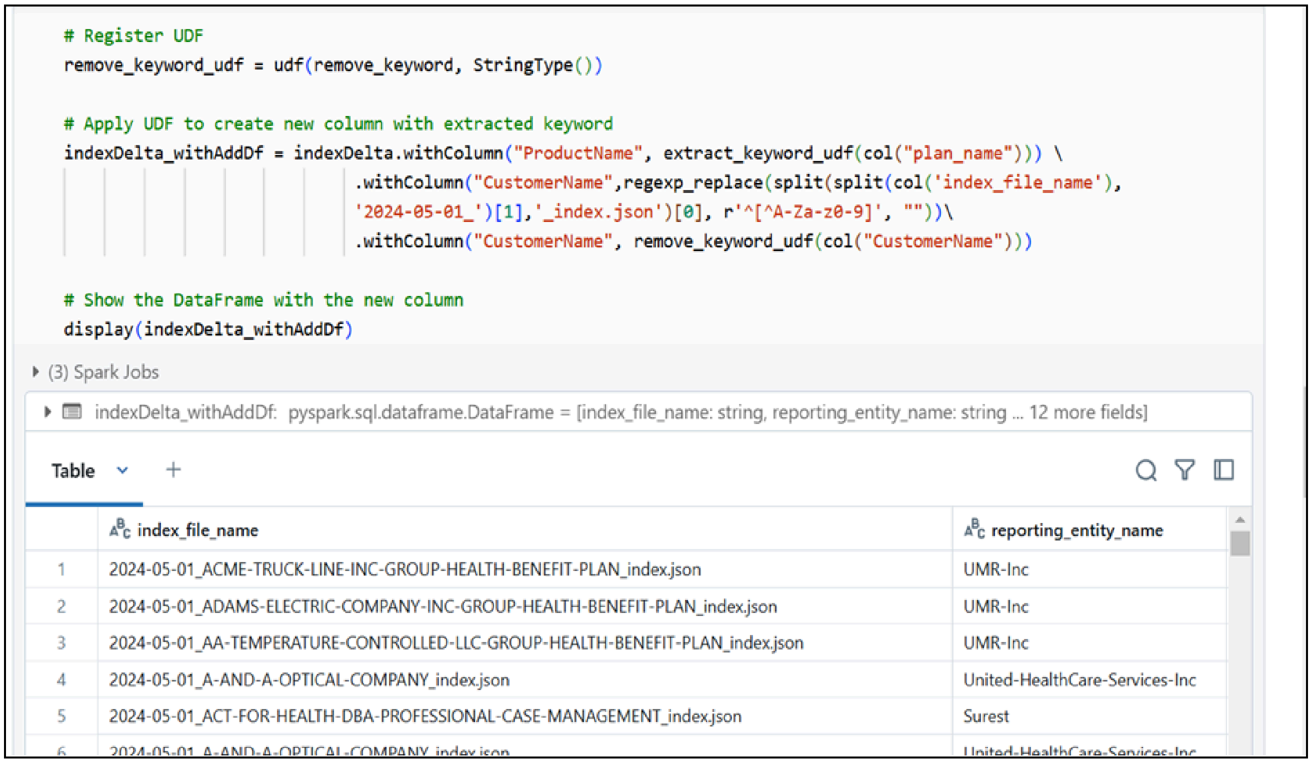
```
indexDelta = spark.read.format('delta').load('/UHN/UHN_index_delta').distinct()
# display(indexDelta)
```

```
indexDelta:  pyspark.sql.dataframe.DataFrame = [index_file_name: string, reporting_entity_name: string ... 10 more fields]
```

```
from pyspark.sql.functions import *
indexDelta1 = indexDelta.select('index_file_name').distinct().withColumn("CustomerName",regexp_replace(split(split(col
('index_file_name'),'2024-05-01_')[1],'_index.json')[0], r'^[^A-Za-z0-9]', ""))
display(indexDelta1)
```

```
(2) Spark Jobs
indexDelta1:  pyspark.sql.dataframe.DataFrame = [index_file_name: string, CustomerName: string]
```

	index_file_name	CustomerName
1	2024-05-01_ADAMS-COUNTY-GROUP-HEALTH-BENEFIT-PLAN_index.json	ADAMS-COUNTY-GROUP-HEALTH-I
2	2024-05-01_ADENA-CORPORATION-GROUP-HEALTH-BENEFIT-PLAN_index.json	ADENA-CORPORATION-GROUP-HE
3	2024-05-01_ACELERO-INC-GROUP-HEALTH-BENEFIT-PLAN_index.json	ACELERO-INC-GROUP-HEALTH-BEN
4	2024-05-01_ADAIR-COUNTY-HEALTH-SYSTEM-GROUP-HEALTH-BENEFIT-PLAN_index.json	ADAIR-COUNTY-HEALTH-SYSTEM-C



```
# Register UDF
remove_keyword_udf = udf(remove_keyword, StringType())

# Apply UDF to create new column with extracted keyword
indexDelta_withAddDf = indexDelta.withColumn("ProductName", extract_keyword_udf(col("plan_name"))) \
                                  .withColumn("CustomerName",regexp_replace(split(split(col('index_file_name'),
                                  '2024-05-01_')[1],'_index.json')[0], r'^[^A-Za-z0-9]', ""))\
                                  .withColumn("CustomerName", remove_keyword_udf(col("CustomerName")))

# Show the DataFrame with the new column
display(indexDelta_withAddDf)
```

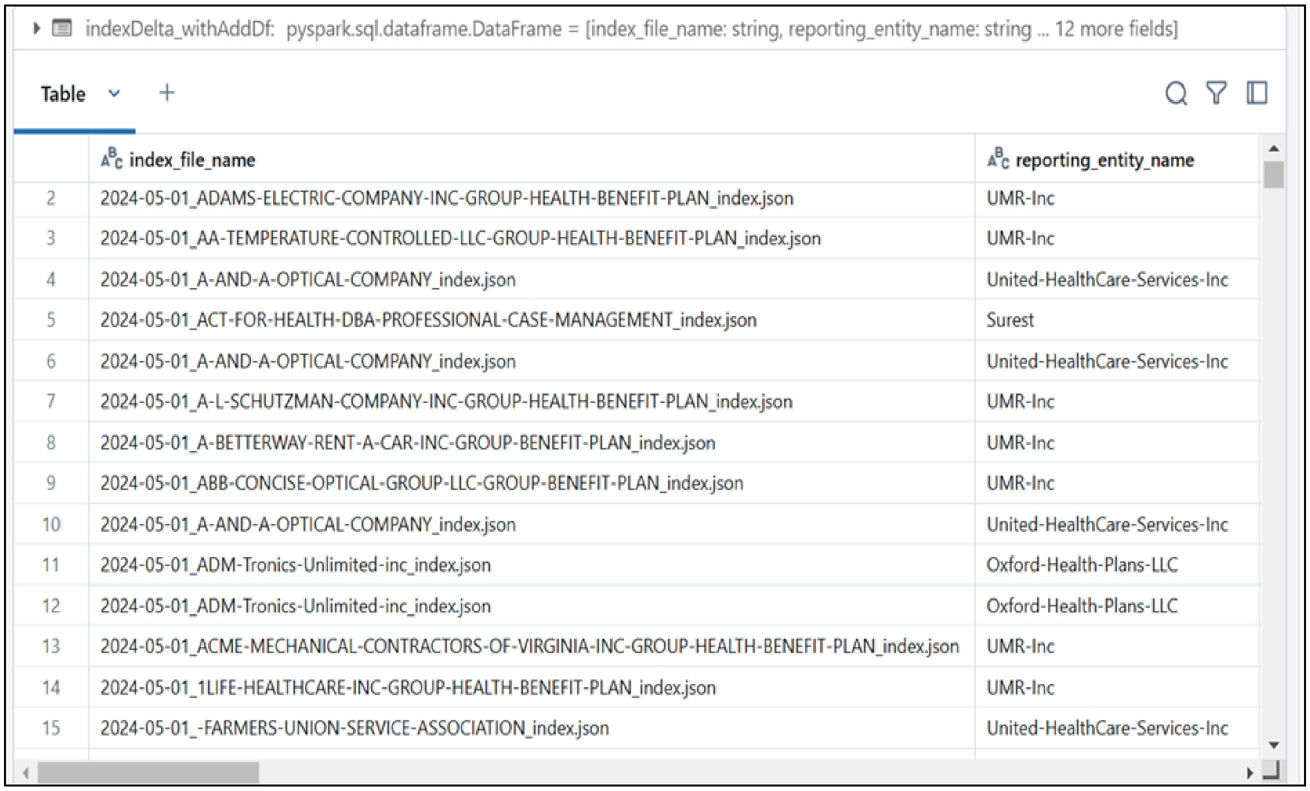
(3) Spark Jobs

indexDelta_withAddDf: pyspark.sql.dataframe.DataFrame = [index_file_name: string, reporting_entity_name: string ... 12 more fields]

Table

	index_file_name	reporting_entity_name
1	2024-05-01_ACME-TRUCK-LINE-INC-GROUP-HEALTH-BENEFIT-PLAN_index.json	UMR-Inc
2	2024-05-01_ADAMS-ELECTRIC-COMPANY-INC-GROUP-HEALTH-BENEFIT-PLAN_index.json	UMR-Inc
3	2024-05-01_AA-TEMPERATURE-CONTROLLED-LLC-GROUP-HEALTH-BENEFIT-PLAN_index.json	UMR-Inc
4	2024-05-01_A-AND-A-OPTICAL-COMPANY_index.json	United-HealthCare-Services-Inc
5	2024-05-01_ACT-FOR-HEALTH-DBA-PROFESSIONAL-CASE-MANAGEMENT_index.json	Surest
6	2024-05-01_A-AND-A-OPTICAL-COMPANY_index.json	United-HealthCare-Services-Inc

Fig 11. Renaming the 'plan_name' and 'index_file_name' columns.



indexDelta_withAddDf: pyspark.sql.dataframe.DataFrame = [index_file_name: string, reporting_entity_name: string ... 12 more fields]

Table

	index_file_name	reporting_entity_name
2	2024-05-01_ADAMS-ELECTRIC-COMPANY-INC-GROUP-HEALTH-BENEFIT-PLAN_index.json	UMR-Inc
3	2024-05-01_AA-TEMPERATURE-CONTROLLED-LLC-GROUP-HEALTH-BENEFIT-PLAN_index.json	UMR-Inc
4	2024-05-01_A-AND-A-OPTICAL-COMPANY_index.json	United-HealthCare-Services-Inc
5	2024-05-01_ACT-FOR-HEALTH-DBA-PROFESSIONAL-CASE-MANAGEMENT_index.json	Surest
6	2024-05-01_A-AND-A-OPTICAL-COMPANY_index.json	United-HealthCare-Services-Inc
7	2024-05-01_A-L-SCHUTZMAN-COMPANY-INC-GROUP-HEALTH-BENEFIT-PLAN_index.json	UMR-Inc
8	2024-05-01_A-BETTERWAY-RENT-A-CAR-INC-GROUP-BENEFIT-PLAN_index.json	UMR-Inc
9	2024-05-01_ABB-CONCISE-OPTICAL-GROUP-LLC-GROUP-BENEFIT-PLAN_index.json	UMR-Inc
10	2024-05-01_A-AND-A-OPTICAL-COMPANY_index.json	United-HealthCare-Services-Inc
11	2024-05-01_ADM-Tronics-Unlimited-inc_index.json	Oxford-Health-Plans-LLC
12	2024-05-01_ADM-Tronics-Unlimited-inc_index.json	Oxford-Health-Plans-LLC
13	2024-05-01_ACME-MECHANICAL-CONTRACTORS-OF-VIRGINIA-INC-GROUP-HEALTH-BENEFIT-PLAN_index.json	UMR-Inc
14	2024-05-01_1LIFE-HEALTHCARE-INC-GROUP-HEALTH-BENEFIT-PLAN_index.json	UMR-Inc
15	2024-05-01_-FARMERS-UNION-SERVICE-ASSOCIATION_index.json	United-HealthCare-Services-Inc

8.c. Interface to view results: The processed data is presented through simple charts using Microsoft Excel, to enable stakeholders to derive actionable insights. Using Spark streaming, the pipeline delivers real-time updates, ensuring that users have access to the most current data.

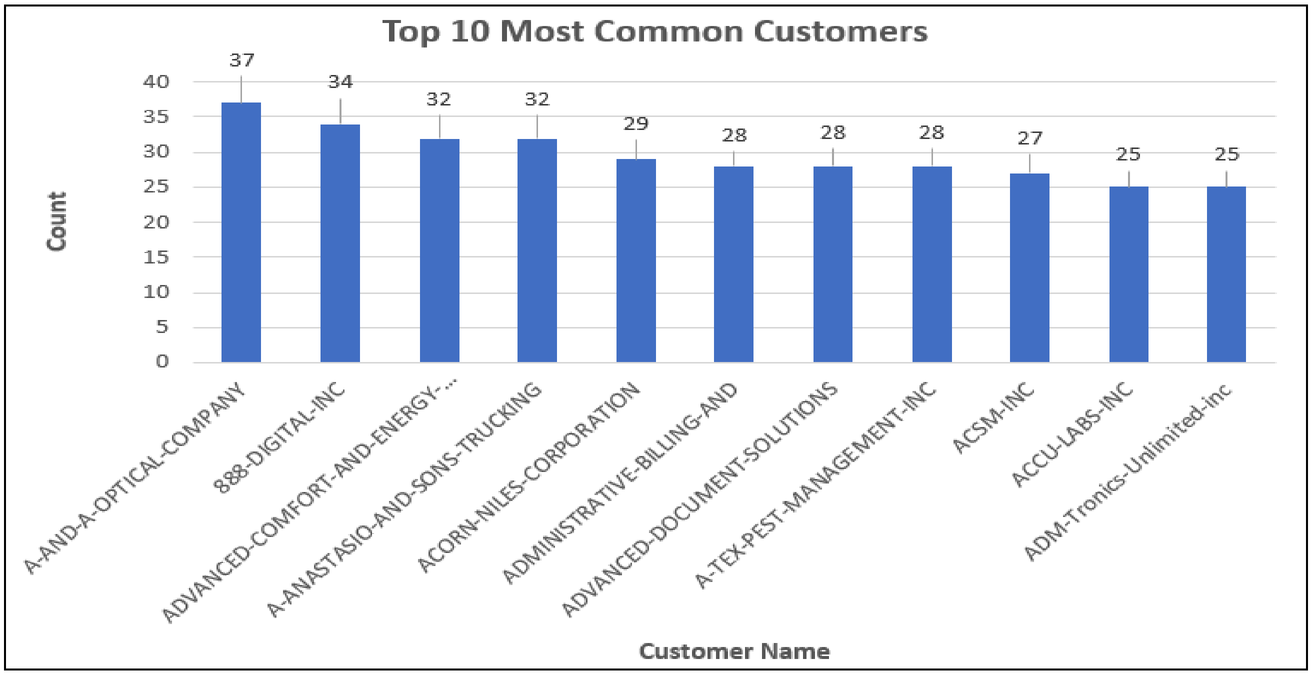


Fig 12. A visualization of Top 10 Most Common Customers.

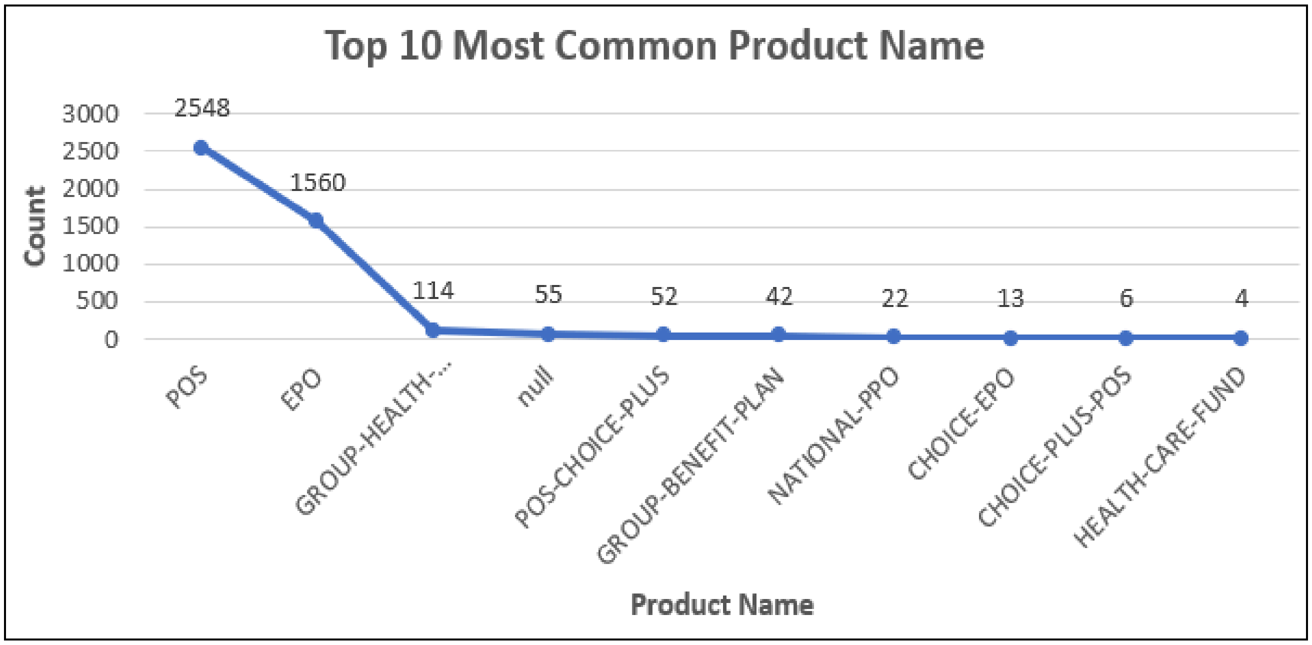


Fig 13. A visualization of Top 10 Most Common Product Names.

9. Challenges and lessons learned:

Challenges:

1. **Limited Support for JavaScript-Rendered Content:**
   Web scraping often requires handling JavaScript-rendered websites, which are not directly supported by BeautifulSoup. To address this limitation, we integrated additional tools like Selenium, **'chromedriver_autoinstaller'**, and Chrome options such as **'add_argument'**. This combination allowed us to interact with dynamic content and extract the necessary data effectively.
2. **Handling Large Datasets:**
   The dataset was extensive, causing system crashes during processing. To mitigate this, we limited the dataset size by reducing the index value to 1000, enabling the system to handle data more efficiently without compromising the results.
3. **Managing HTTP Errors:**
   While fetching data using '**requests.get(url)'**, we encountered a "500 Internal Server Error," likely due to server downtime. To ensure robustness, we implemented exception handling mechanisms that identified and managed such scenarios gracefully without disrupting the workflow.

Lessons Learned:

1. **The Importance of Robust Tool Selection:**
   Choosing appropriate tools and libraries for the task is crucial. Beautiful Soup, while effective for basic scraping, must be supplemented with tools like Selenium for dynamic content.
2. **Scalability Challenges in Big Data:**
   Processing large datasets demands careful planning and optimization to avoid overwhelming system resources. Techniques such as indexing and partitioning are essential for managing such data efficiently.
3. **Resilience in Web Scraping:**
   Handling exceptions like server errors ensures the continuity and reliability of data extraction processes. It also emphasizes the need for monitoring and fallback mechanisms to handle unexpected issues.
4. **Adaptability to Dynamic Web Structures:**
   Modern web scraping projects require flexibility to accommodate different data formats and content rendered by JavaScript. This insight underscores the value of combining multiple tools for a seamless workflow.

10. References:

1. Centres for Medicare & Medicaid Services (CMS), 2024. *Health Plan Price Transparency*. Available at: https://www.cms.gov/priorities/key-initiatives/healthplan-price-transparency.
2. United Healthcare, 2024. *Transparency in Coverage*. Available at: https://transparency-in-coverage.uhc.com/
3. Health Affairs, 2023. *It's Been A Year Of Transparency In Coverage Compliance Studies*. Available at: https://www.healthaffairs.org/content/forefronts/s-been-year-transparency-coverage-compliance-studies.
4. GeeksforGeeks, 2024. *Implementing Web Scraping in Python – Beautiful Soup*. Available at: https://www.geeksforgeeks.org/implementing-web-scraping-python-beautiful-soup/.
5. ScrapingBee, 2024. *Selenium Python: Web Scraping Tutorial with Examples*. Available at: https://www.scrapingbee.com/blog/selenium-python/.
6. BrowserStack, 2024. *Web Scraping Using Selenium and Python*. Available at: https://www.browserstack.com/guide/web-scraping-using-selenium-python.
7. Apache Spark, 2024. *Examples*. Available at: https://spark.apache.org/examples.html.
8. Apache Spark, 2024. *Getting Started with Spark SQL*. Available at: https://spark.apache.org/docs/latest/sql-getting-started.html.
9. Stack Overflow, 2024. *How to Manage Options in PySpark More Efficiently?*. Available at: https://stackoverflow.com/questions/71642411/how-to-manage-options-in-pyspark-more-efficiently.
10. Spark by {Examples}, 2024. *Spark Read Options*. Available at: https://sparkbyexamples.com/spark/spark-read-options/.