Exploratory Data Analysis of FDA Drug Reaction Dataset

- Technical Writeup

Team:

Roehit Ranganathan - rr4461
Abhinav Sivanandhan - as17904

Objective:

Analyze the FDA Drug Reaction dataset to extract insights on adverse drug reactions, drug usage trends, and associated patterns.intake. Analyzing ADEs is critical to identifying safety concerns, understanding potential drug interactions,and improving public health by guiding safer medication practices. A lot of information is missed during data collection, such as sufficient demographic information, dosages, product composition, clinical trial data, and so on. This missing information can lead to medications being disparately effective across different populations, especially minorities, due to lack of testing and unknown factors at the time.

This project is focused on understanding the causes and trends of adverse drug events (ADEs) using a big data approach. It aims to analyze, visualize, and report trends, causes, and patterns of ADEs by leveraging diverse data sources and a robust big data architecture. We utilize publicly available datasets from OpenFDA to gather extensive drug related data. We aim to identify patterns in ADEs and in relevant information such as clinical trials and drug recalls, as well as to assess the causes and trends of ADEs and the effectiveness of regulatory actions.

Dataset Details:

- **Source**: FDA's OpenFDA Drug Reaction Reports.
- **Size**:
    - Original compressed size: 102 GB.
    - Total Files: 1568 files
    - Unzipped size: 516 GB.
- **Format**: JSON files with nested structures. (File Format: Compressed .json.zip)
- **Fields**: Includes safetyreportid, drug name, reaction, etc., with some nested structures.

The biggest challenge was handling these large number of hundred MB files that are highly nested, making it difficult for even simple analysis. The data was also spread out over 20 years, so schema has varied over the years probably due to changes in data collection and reporting formats. So building a unified schema was a challenge

Most of our project involved analysing the dataset using various big data techniques to handle this data on our limited local devices [**MacBook Pro** (10-Core CPU, 16GB RAM) , **1TB SATA III External SSD** (500MB/s read/write rates)]. The read/write rates were the primary bottleneck of

our project, requiring us to use parallelised approaches to handle the dataset preprocessing and efficient storage techniques for the final dataset that we performed our EDA, visualisation and analysis on.

Tools & Technologies:

- **Dask**: File handling and parallel computation.
- **Spark**: Distributed data processing and exploratory analysis.
- **PyArrow**: Parquet file generation.
- **Pandas**: Local inspection and data validation.
- **JSON**: Data format for raw files.
- **Snappy**: Parquet file compression.

Data Preprocessing and Cleaning:

- Extracted JSON files from compressed .zip archives.
- Flattened nested JSON structures.
- Removed unnecessary fields such as rxcui, product_ndc, spl_id, etc.
- Logged and skipped corrupted or empty files

Link_retriever.py

This script extracts all hyperlinks (href attributes) from an HTML file and saves them to a text file, with one link per line..

Downloader.py:

This Python script automates the downloading of files or content from URLs provided in a text file. It downloads links within a specified range of lines and stores the files in a designated directory. Scalable for Large Datasets as it handles thousands of links in a single file with adjustable range (start_line, end_line).

Error Handling: Skips empty lines and continues downloading the next valid link. Logs errors for failed downloads but does not interrupt the entire process.

Unzipper.py:

This script is designed to parallelize the unzipping of large datasets from a directory of .zip files, while adhering to a size limit constraint (800GB in this case) to ensure the total extracted data remains manageable. The implementation balances efficiency and safety by using multithreading for parallel processing and enforcing size constraints with a thread-safe lock mechanism.

Fine_analysis.py:

This script analyzes the structure of JSON files within a specified directory and its subdirectories. It identifies nested key structures (up to a specified depth) and compares the structures across files to find common and unique keys. The process is parallelized using Dask, allowing for efficient processing of large datasets. A detailed log and a summary report of the analysis are generated.

Overview:

1. Logging:
    - Configures logging to output messages to both a log file and the console.
    - Provides clear insights into the script's execution and any errors encountered.
2. Key Structure Extraction:
    - Recursively extracts the nested structure of JSON objects up to a specified depth.
    - Outputs keys in the format parent.child for nested paths.
3. File Analysis:
    - Analyzes each JSON file for its key structure.
    - Handles exceptions gracefully and logs errors for files that cannot be processed.
4. Structure Comparison:
    - Compares the key structures of all analyzed files.
    - Identifies:
        - Common keys: Keys that are present across all files.
        - Unique keys: Keys that are specific to individual files.
5. Parallel Processing:
    - Uses **Dask** to process JSON files in parallel, leveraging multicore CPUs for faster execution.
    - Allows users to choose between thread-based or process-based parallelism.
6. Report Generation:
    - Saves the analysis summary (common keys, unique keys, and errors) to a JSON file.
    - Includes detailed information for debugging and further inspection.

Data_cleaner.py:

This script is designed to process and clean large nested JSON files in parallel by removing specific unwanted fields from the data. It leverages **Dask's** parallel processing capabilities to efficiently handle and clean massive datasets without overwhelming system memory.

Field Removal: The script removes specified fields (FIELDS_TO_REMOVE) from JSON files, even if these fields are deeply nested.

Parallel Processing with Dask: Uses Dask Bag to process multiple JSON files simultaneously, taking advantage of multicore CPUs for faster execution.

Handling Large Files: Processes files chunk-by-chunk to avoid memory overflow when dealing with large datasets.

Preserving Directory Structure: Ensures that the cleaned files are saved in an output directory (OUTPUT_DIR) with the same directory structure as the input directory.

Conversion to Parquet:

- JSON files were converted to **Parquet** format for efficient storage and performance.
- Files were compressed using **Snappy**.
- Data size was significantly reduced during conversion.

Overview

Reading JSON Files:

We processed a directory containing JSON files, where each file was loaded into memory. The data could be a single record, a list of records, or nested structures.Since JSON is already a serialized format, we did not explicitly serialize it further. But we do serialise it after removing some fields later on.

Normalizing Nested Data:

JSON files contain highly nested structures. We flattened the data using libraries like pandas.json_normalize() to convert nested fields into a tabular (row-column) format.

Handling Missing and Null Values:

Null values in the JSON were preserved as NaN (Not a Number) in pandas or as null in Parquet. These placeholders are compatible with the big data frameworks we used.

Ensuring a Common Schema:

JSON files have inconsistent schemas (e.g., missing fields) because the data is from a large timeframe. We defined a unified schema by checking all files and including all possible columns, filling missing data with nulls where necessary.

Data Cleaning:

We removed unnecessary fields to reduce data size:

Removed Fields: "rxcui", "product_ndc", "spl_id", "spl_set_id", "package_ndc", "nui", "application_number".

This was done during the conversion of JSON to a pandas.DataFrame, ensuring the resulting Parquet files didn't include these fields.

Partitioning:

To optimize storage and queries, we partitioned the data by a key column (e.g., receivedate) into directories. This allows tools like Spark or Dask to read only the required partitions.

Writing to Parquet:

After processing, the tabular data was written to Parquet format using PyArrow, a columnar storage format ideal for big data analysis due to its compression and query efficiency.

Handling Large Files:

The process was incremental, keeping track of processed files to allow restarting from where it left off in case of interruptions.

Schema Validation:

As Parquet requires a fixed schema, we ensured all files in a partition adhered to the same column names and types.

Why Parquet Instead of JSON:

Parquet is efficient for analytical queries due to its columnar format and compression, whereas JSON is better for data interchange but inefficient for querying or storage at scale. Instead, we focused on deserializing JSON into a usable form for processing and then converting it to Parquet for storage.

Exploratory Data Analysis (EDA):

- Summary statistics on key fields:
    - Number of unique drugs.
    - Frequency of adverse reactions.
    - Drugs with the most reports.
- Tools used:
    - Dask for distributed computation of large Parquet files.
    - DuckDB for querying metadata and analysing schemas.
    - Pandas for local inspection.

EDA insights(Mongo-spark connector):

a. Count reports by seriousness (e.g., "1" for serious cases):
(in order of thousands)
{'_id': '1', 'count': 35775}
{'_id': '2', 'count': 18266}

b. Top 10 most common reactions (reaction.reactionmeddrapt):

(in order of thousands)
{'_id': 'NAUSEA', 'count': 2634}
{'_id': 'DRUG INEFFECTIVE', 'count': 2172}
{'_id': 'DIZZINESS', 'count': 1881}
{'_id': 'HEADACHE', 'count': 1746}
{'_id': 'DEATH', 'count': 1712}
{'_id': 'VOMITING', 'count': 1697}
{'_id': 'DYSPNOEA', 'count': 1684}
{'_id': 'FATIGUE', 'count': 1575}
{'_id': 'ASTHENIA', 'count': 1504}
{'_id': 'PYREXIA', 'count': 1447}

c. Top drugs involved in reports (drug.medicinalproduct):
(in order of thousands)

{'_id': 'ORTHO EVRA', 'count': 2567}
{'_id': 'ASPIRIN', 'count': 2374}
{'_id': 'FORTEO', 'count': 2177}
{'_id': 'HUMIRA', 'count': 2053}
{'_id': 'PREMARIN', 'count': 1970}
{'_id': 'PREDNISONE', 'count': 1703}
{'_id': 'PROVERA', 'count': 1691}
{'_id': 'THALOMID', 'count': 1537}
{'_id': 'NAMENDA', 'count': 1436}
{'_id': 'METHOTREXATE', 'count': 1409}

d. Count reports by patient gender (patientsex):
(in order of thousands)

{'_id': '2', 'count': 31014}
{'_id': '1', 'count': 19065}
{'_id': None, 'count': 3219}
{'_id': '0', 'count': 743}

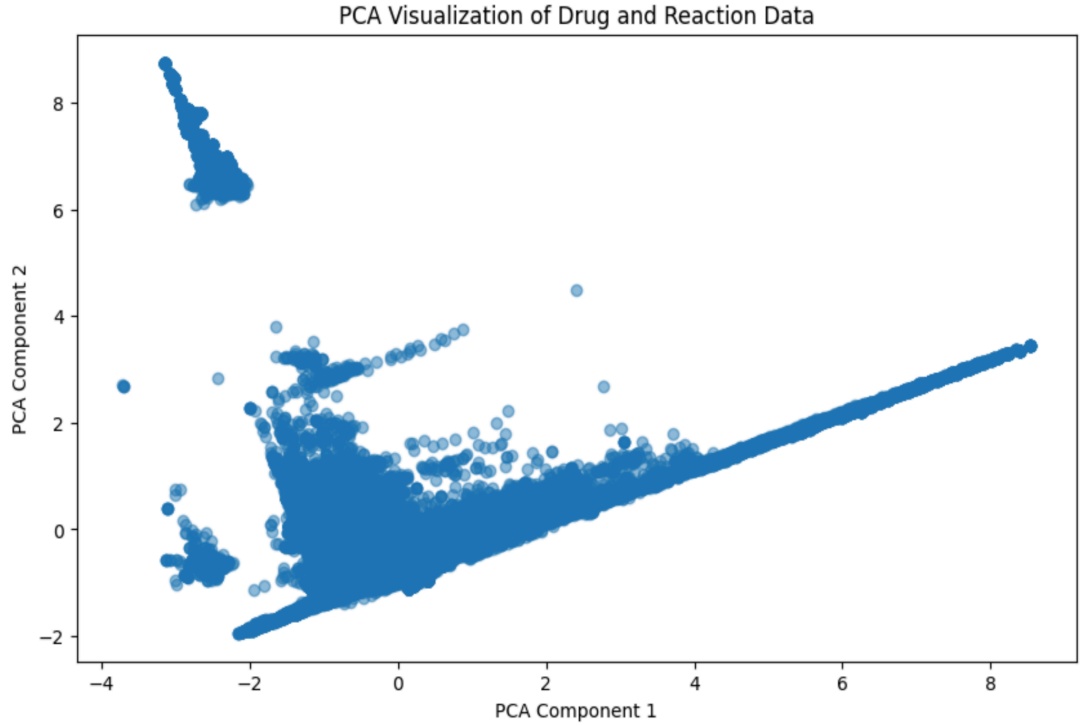
Advanced Insights
a. Serious reports grouped by drug:
(in order of thousands)

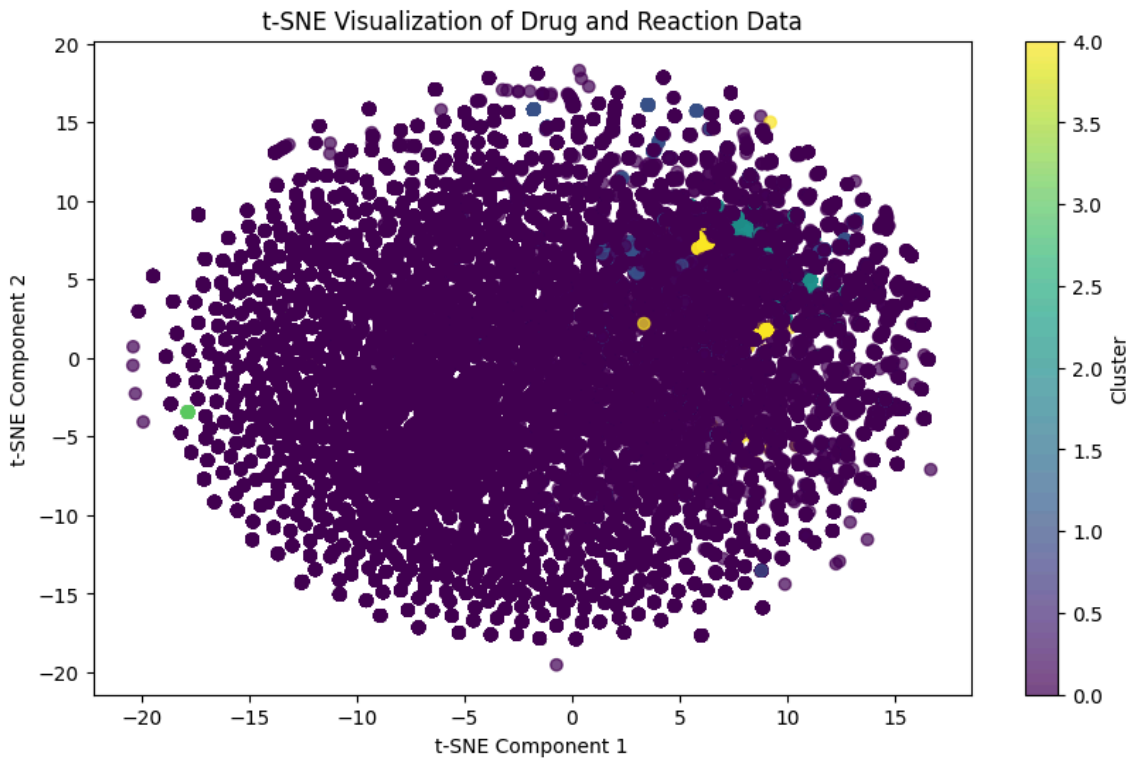
{'_id': 'PREMARIN', 'count': 1869}
{'_id': 'ASPIRIN', 'count': 1855}
{'_id': 'PROVERA', 'count': 1671}
{'_id': 'THALOMID', 'count': 1531}
{'_id': 'AVONEX', 'count': 1383}
{'_id': 'REMICADE', 'count': 1342}
{'_id': 'PREMPRO', 'count': 1178}
{'_id': 'PREDNISONE', 'count': 1073}
{'_id': 'CRESTOR', 'count': 1020}
{'_id': 'METHOTREXATE', 'count': 996}

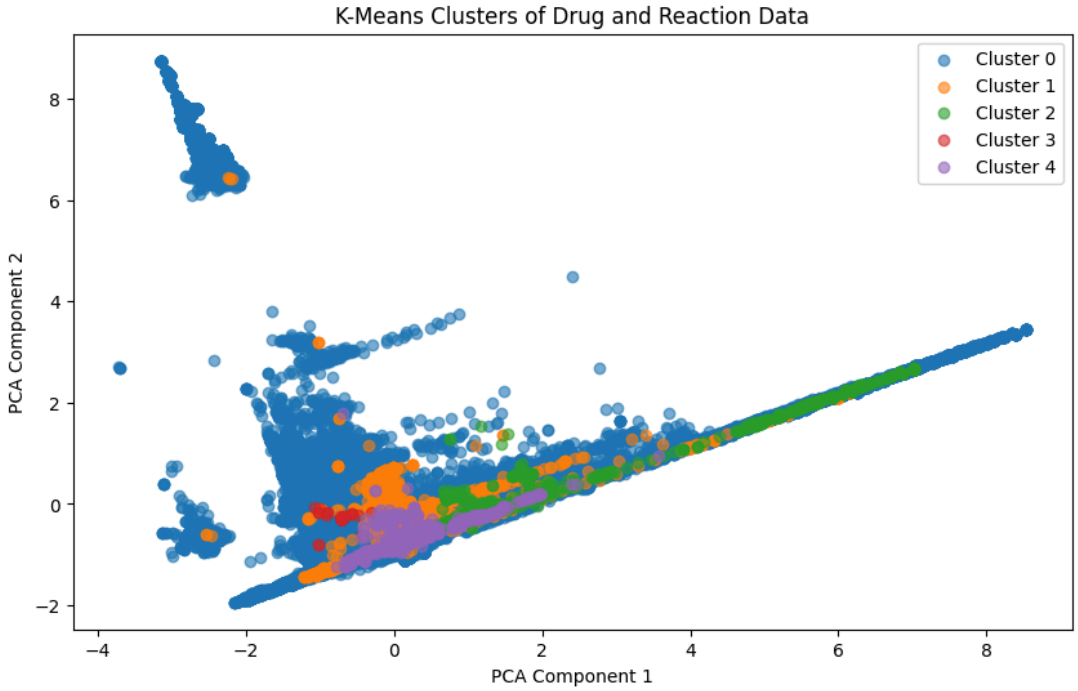
b. Common combinations of drugs and reactions:
(in order of thousands)

{'_id': {'drug': 'PROVERA', 'reaction': 'BREAST CANCER FEMALE'}, 'count': 1062}
{'_id': {'drug': 'PREMARIN', 'reaction': 'BREAST CANCER FEMALE'}, 'count': 1010}
{'_id': {'drug': 'THALOMID', 'reaction': 'DEATH'}, 'count': 940}
{'_id': {'drug': 'ORTHO EVRA', 'reaction': 'METRORRHAGIA'}, 'count': 743}
{'_id': {'drug': 'PREMPRO', 'reaction': 'BREAST CANCER FEMALE'}, 'count': 585}
{'_id': {'drug': 'PAXIL', 'reaction': 'DRUG WITHDRAWAL SYNDROME'}, 'count': 457}
{'_id': {'drug': 'PREMARIN', 'reaction': 'BREAST CANCER'}, 'count': 440}
{'_id': {'drug': 'FORTEO', 'reaction': 'INJECTION SITE HAEMORRHAGE'}, 'count': 433}
{'_id': {'drug': 'THALOMID', 'reaction': 'DISEASE PROGRESSION'}, 'count': 421}
{'_id': {'drug': 'PROVERA', 'reaction': 'BREAST CANCER'}, 'count': 394}

Feature selection helpers:

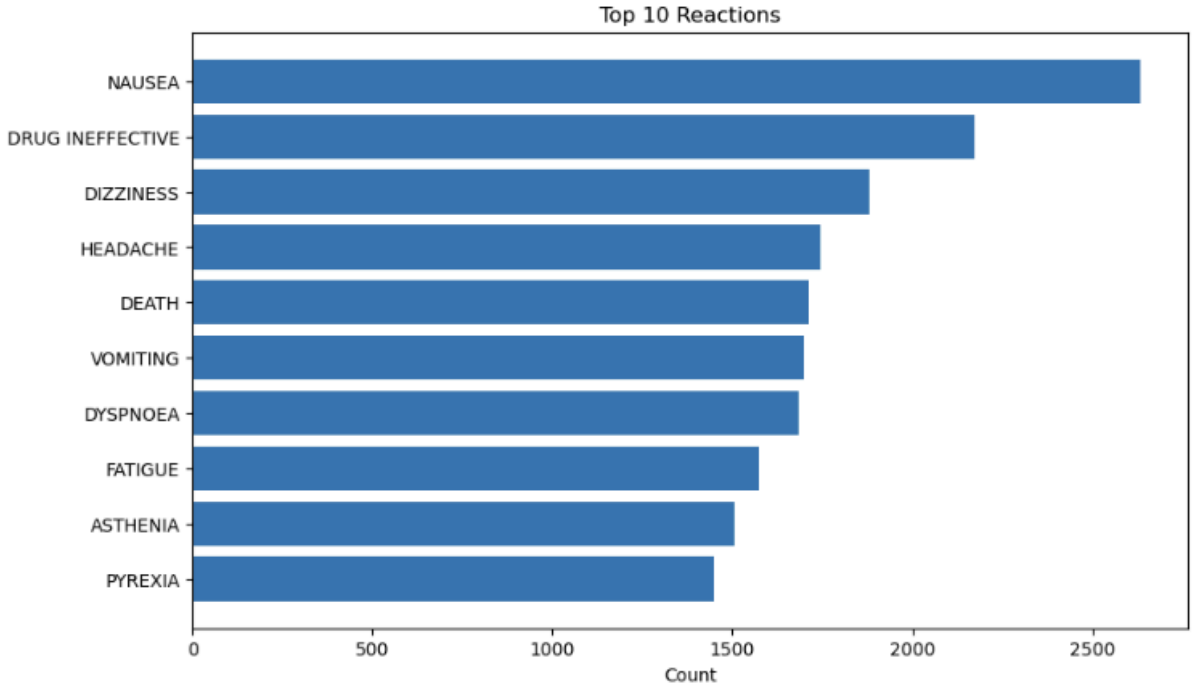


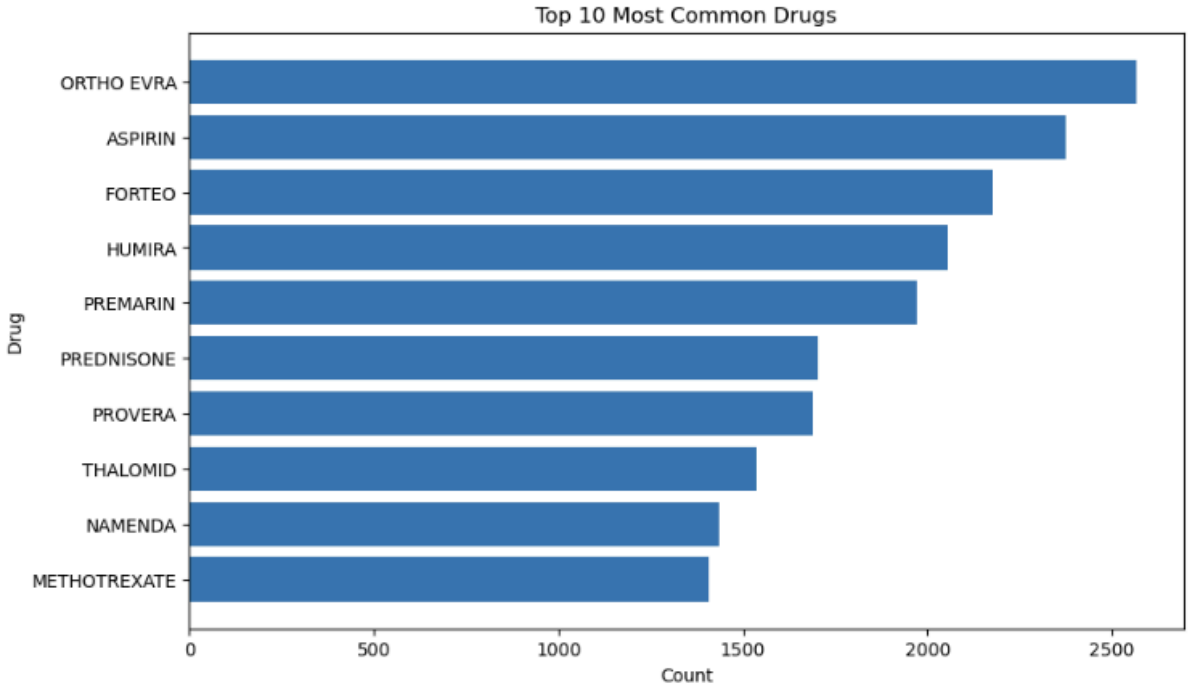


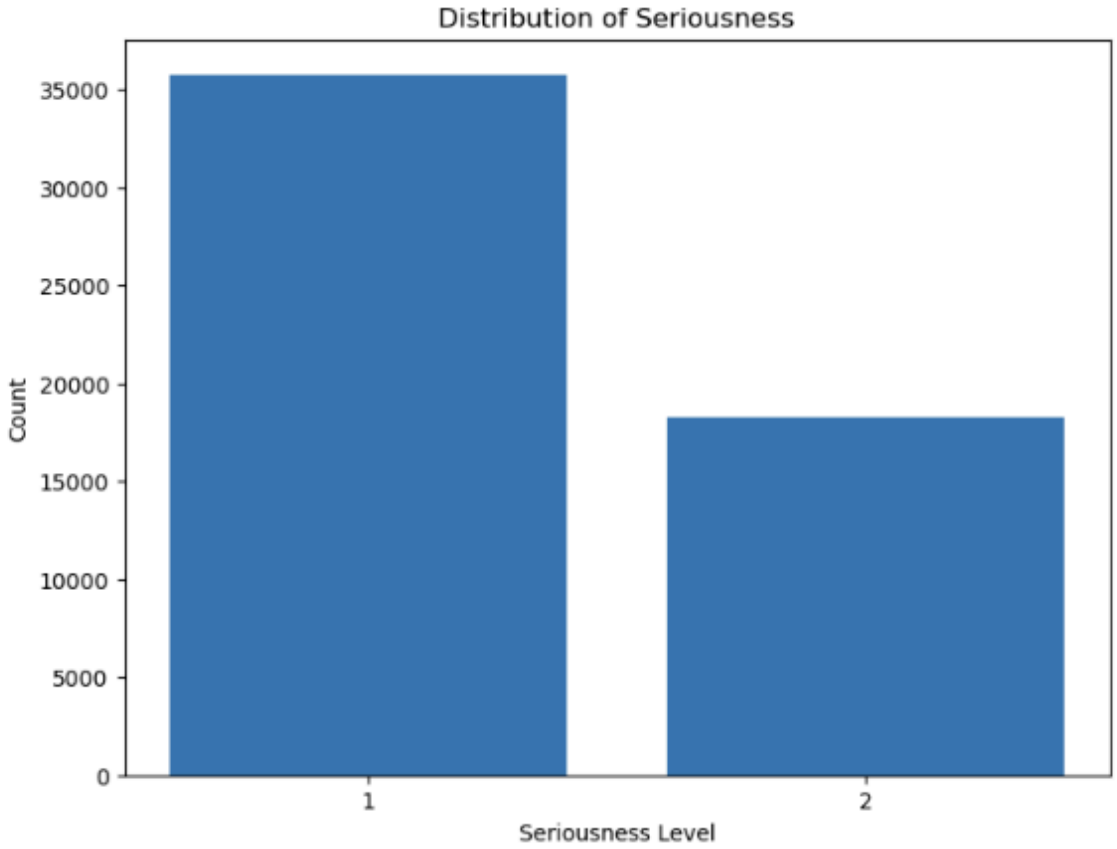


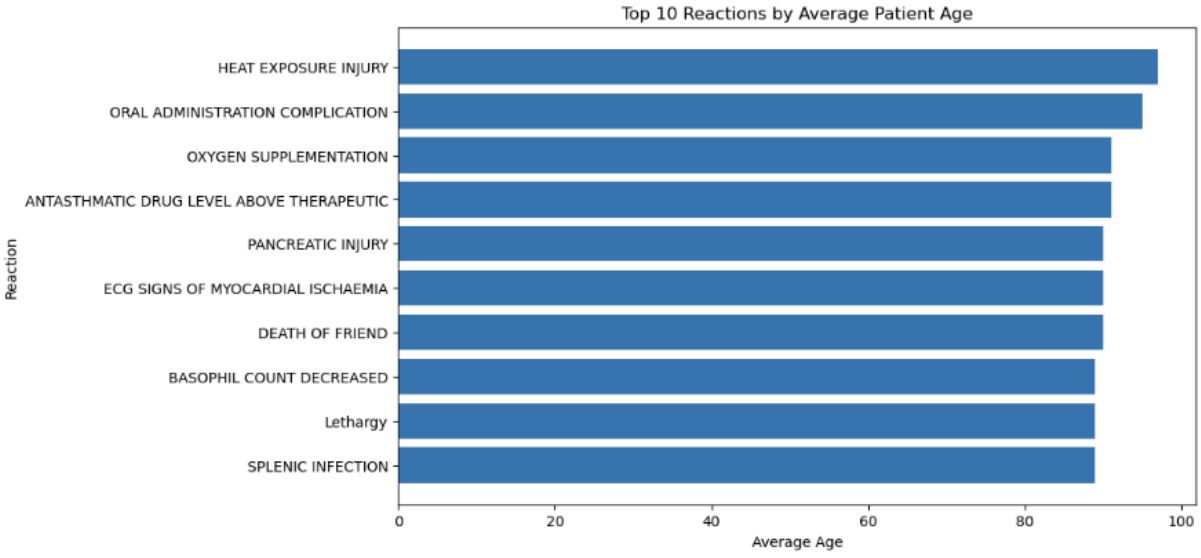
Note: K-means was done only on representative dataset

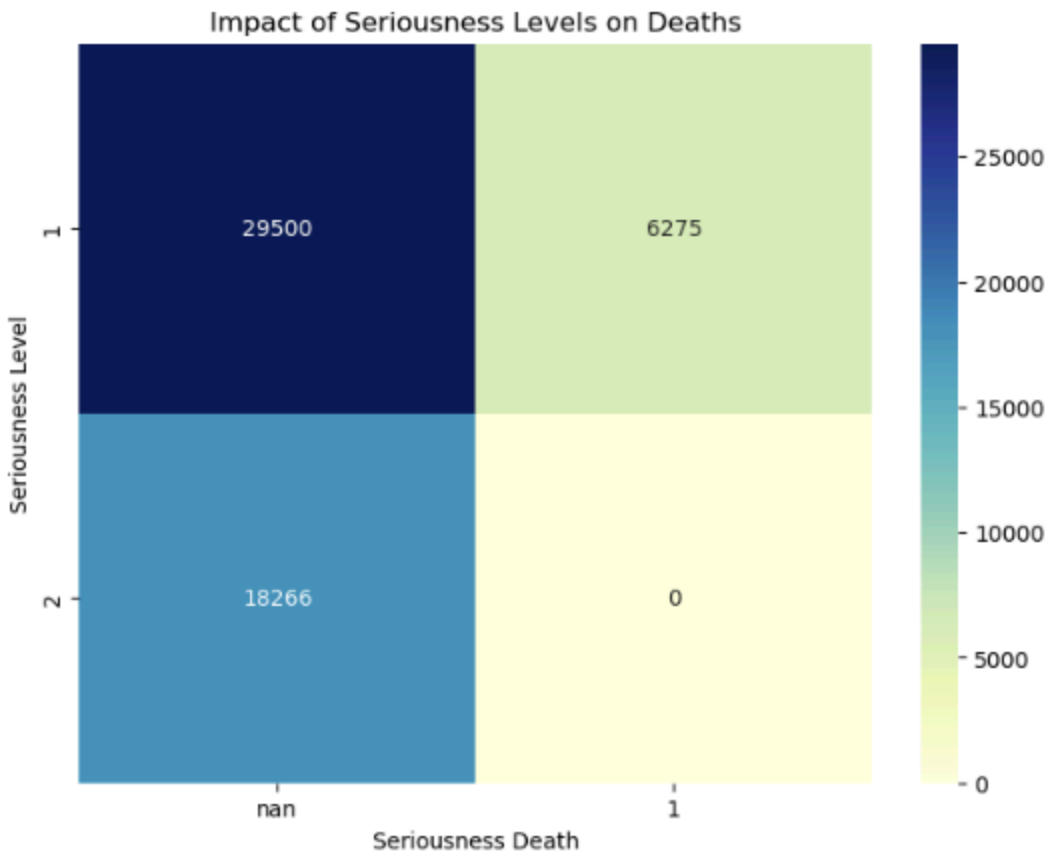
Visualisations (in order of thousands)

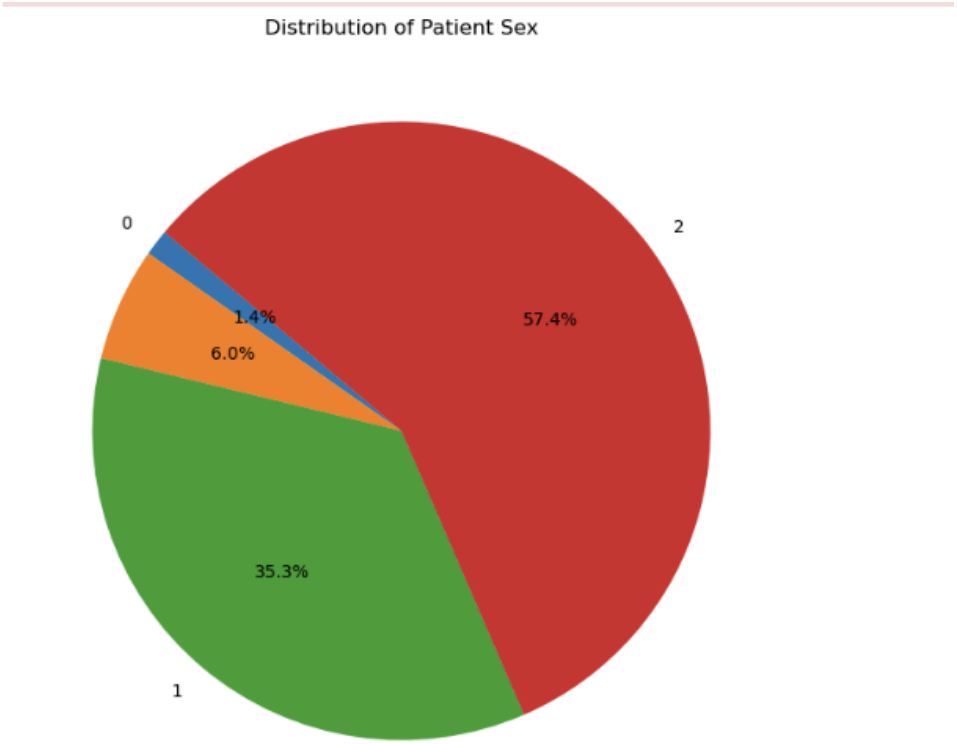





Distribution of Seriousness
Count
35000
30000
25000
20000
15000
10000
5000
0
1
2
Seriousness Level


Top 10 Reactions by Average Patient Age
Reaction
HEAT EXPOSURE INJURY
ORAL ADMINISTRATION COMPLICATION
OXYGEN SUPPLEMENTATION
ANTASTHMATIC DRUG LEVEL ABOVE THERAPEUTIC
PANCREATIC INJURY
ECG SIGNS OF MYOCARDIAL ISCHAEMIA
DEATH OF FRIEND
BASOPHIL COUNT DECREASED
Lethargy
SPLENIC INFECTION
0
20
40
60
80
100
Average Age


Impact of Seriousness Levels on Deaths
29500
6275
18266
0
Seriousness Level
1
2
nan
1
Seriousness Death
25000
20000
15000
10000
5000
0


Distribution of Patient Sex
0
2
1.4%
57.4%
6.0%
35.3%
1

Lessons Learned:

- Benefits of Parquet for space savings and performance: Parquet's columnar storage significantly reduces file size, enabling faster query execution and more efficient use of disk space. By compressing the dataset from 600 GB to ~15 GB using Snappy, the project avoided processing delays and storage bottlenecks. The columnar format allows selective access to necessary fields, reducing the data read into memory during analysis.
- Handling corrupted files and tracking errors: Robust error handling is critical when working with large datasets. Logging invalid files and skipping them during processing ensured the pipeline's continuity. Incorporating detailed error logs facilitates debugging and improving data quality over time.
- Efficient processing of large nested JSON files: Flattening nested structures and converting them to Parquet accelerated queries and simplified downstream tasks. Tools like Dask and PyArrow provided distributed computing capabilities to manage memory constraints and high file complexity effectively.

Future Work:

We could further clean the dataset to exclude outliers and improve error handling and handling of schema changes to derive more insights from the data.

We can also automate the identification of duplicate reports, outliers, and missing fields and use a centralized metadata repository to track file statuses. This was when there’s a data drift the big data analysis can still be efficient.

It is important to explore reaction severity, patient demographics, and drug-reaction associations using clustering and network analysis. Also, Automate EDA processes for new datasets.

In order to efficiently handle large-scale machine learning tasks on a 600GB dataset, leveraging cloud-based platforms such as AWS, Azure, or Google Cloud is essential for scalable computing. Services like Amazon S3 can be used for storage, while tools like EMR or Databricks enable distributed processing, effectively managing vast data volumes without local limitations. Distributed ML frameworks such as Spark MLlib or TensorFlow deployed on Kubernetes clusters are well-suited for handling such data. Preprocessing and creating stratified subsets for initial model training ensure manageable data chunks for efficient learning. GPU-enabled instances can accelerate deep learning tasks, significantly reducing computation time. To prevent crashes, resource managers like Apache YARN can monitor workloads and allocate resources dynamically, while implementing checkpointing ensures intermediate states are saved, allowing recovery in case of system failures. This combination of distributed computing, scalable frameworks, and robust failure handling enables efficient ML on massive datasets.

References

- Shamim, M. A., Shamim, M. A., Arora, P., & Dwivedi, P. (2024). Artificial intelligence and big data for pharmacovigilance and patient safety. *Journal of Medicine, Surgery, and Public Health*, 3, 100139.
- Kaas-Hansen, B. S., Gentile, S., Caioli, A., & Andersen, S. E. (2022). Exploratory pharmacovigilance with machine learning in big patient data: A focused scoping review. *Basic & Clinical Pharmacology & Toxicology*, Focused Review. First published: 21 December 2022. https://doi.org/10.1111/bcpt.13828