IRisk Lab Data Discovery and Consolidation - Task 2 : Midterm Report

Week 1 Report
Aug. 29
Rohit Valmeekam

What was accomplished

1. Created rough guidelines for the creation of the database system. We decided to group by Insurance category, as the data that the Task 1 team has gathered is grouped by Insurance category.
2. I initiated the thought process of how our database system would end up as, and I decided on the creation of a one-to-many relationship between "Insurance Category" and "Dataset Table" database tables to thoroughly display all the datasets in an organized manner. This is due to the nature of the datasets that were acquired by Team 1 and them being grouped by different insurance categories.
3. Started thinking about what technologies to use and we ended up deciding on MongoDB to create our database system due to the fact that it is faster for large databases, whereas MySQL would be slower in comparison.
    a. Still in the process of deciding whether to utilize a Relational Database Management System or a Document Database.
        i. Document Database is more horizontally scalable however if the data conforms to a rigid structure, we may have to utilize a Relational Database System.
4. Started reading a textbook on MongoDB to familiarize myself with the technology.
5. Sent email to Eli O'Donohue asking for more information on the steps necessary to create a Relational Database Management System, if we were to utilize one over a Document Database.

→next page

IRisk Lab Data Discovery and Consolidation - Task 2
Week 2 Report
Sep. 5
Rohit Valmeekam

What was accomplished

Dove Deeper into the benefits and downsides of using DQL as opposed to MongoDB

SQL (Structured Query Language)

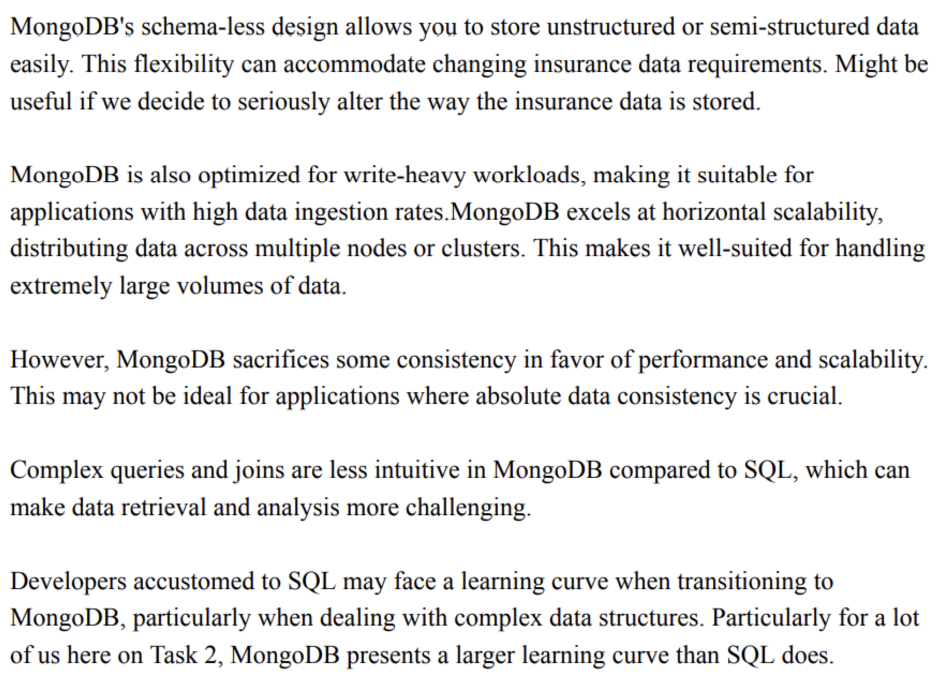
SQL databases are known for their strong data integrity, consistency, and adherence to ACID (Atomicity, Consistency, Isolation, Durability) properties. This makes them highly reliable for storing sensitive data. They offer well-established security features, including role-based access control, encryption, and authentication mechanisms. These features are crucial for maintaining high levels of data security. SQL databases excel at handling complex data relationships through the use of structured tables with defined schemas. This can be advantageous when dealing with intricate insurance data structures. As stated last week, it is critical to have different relationships between "Insurance Category" and "Dataset Table" database tables to thoroughly display all the datasets in an organized manner. This is due to the nature of the datasets that were acquired by Team 1 and them being grouped by different insurance categories.

SQL provides a powerful and standardized query language that is familiar to most developers, making it easier to work with and maintain.

However, SQL databases require a fixed database schema, which can be restrictive when dealing with evolving or unstructured data. Changes to the schema can be challenging. Assuming our data pipeline is created correctly, this should not be an issue.

For certain high-write, high-throughput scenarios, SQL databases might not perform as well as NoSQL databases due to the overhead of enforcing ACID properties. While scalability options exist, scaling SQL databases can be more complex and costly than scaling NoSQL databases, particularly for extremely large data volumes.

MongoDB (NoSQL)



MongoDB's schema-less design allows you to store unstructured or semi-structured data easily. This flexibility can accommodate changing insurance data requirements. Might be useful if we decide to seriously alter the way the insurance data is stored.

MongoDB is also optimized for write-heavy workloads, making it suitable for applications with high data ingestion rates.MongoDB excels at horizontal scalability, distributing data across multiple nodes or clusters. This makes it well-suited for handling extremely large volumes of data.

However, MongoDB sacrifices some consistency in favor of performance and scalability. This may not be ideal for applications where absolute data consistency is crucial.

Complex queries and joins are less intuitive in MongoDB compared to SQL, which can make data retrieval and analysis more challenging.

Developers accustomed to SQL may face a learning curve when transitioning to MongoDB, particularly when dealing with complex data structures. Particularly for a lot of us here on Task 2, MongoDB presents a larger learning curve than SQL does.

Week 3 Report
Sep. 13
Rohit Valmeekam

<u>What was accomplished</u>

Created basic data pipeline and tested it on discovered data , including the following steps :

- ❖ **Input Data Collection:**
  - ➢ The process begins with a Luigi task called ConsolidateData.
  - ➢ This task is responsible for collecting input data from a folder named input_data.
- ❖ **Input Data Parsing:**
  - ➢ Within the ConsolidateData task, Luigi identifies all files in the input_data folder with a .csv extension.
  - ➢ For each of these input files, a Luigi sub-task of type ReadCSV is created.
  - ➢ The ReadCSV task is responsible for reading the content of each input CSV file.
- ❖ **Dataframe Creation:**
  - ➢ Inside the ReadCSV task, the input CSV file is read using pandas, creating a dataframe containing the data.
- ❖ **Intermediary CSV Creation:**

- After reading the input CSV file, the data is written to an intermediary CSV file named "intermediate.csv".
- This intermediary CSV file serves as a temporary storage for the data before further processing within the ConsolidateData task.

❖ **Data Processing Algorithm:**
- Back in the ConsolidateData task, the intermediary CSV file is processed using a data processing algorithm.
- This algorithm includes several steps, such as categorization, folder creation, missing value removal, and output file placement.

❖ **Category Assignment:**
- For each input file, the algorithm prompts the user to specify the category to which the dataset belongs.
- This user input determines the subfolder where the processed data will be saved.

❖ **Folder Creation:**
- If the category-specific folder does not exist within the output directory, it is created.
- This ensures that datasets of the same category are stored together in their respective folders.

❖ **Missing Value Removal:**
- Before saving the processed dataset, any rows with missing values (NaN) are removed from the dataframe.
- This step ensures that only complete data is included in the output.

❖ **Data Output:**
- The processed data frame is then saved as a new CSV file within the appropriate category folder.
- The path to this output file includes both the output directory and the category-specific subfolder.

❖ **Temporary Database (Optional):**
- The cleaned dataset is then read into a dummy database created in SQLLite
- The input file is then deleted in from the input folder

❖ **Input File Deletion:**
- After successfully processing an input CSV file, it is deleted from the input_data folder.
- This ensures that only unprocessed files remain in the input_data folder.

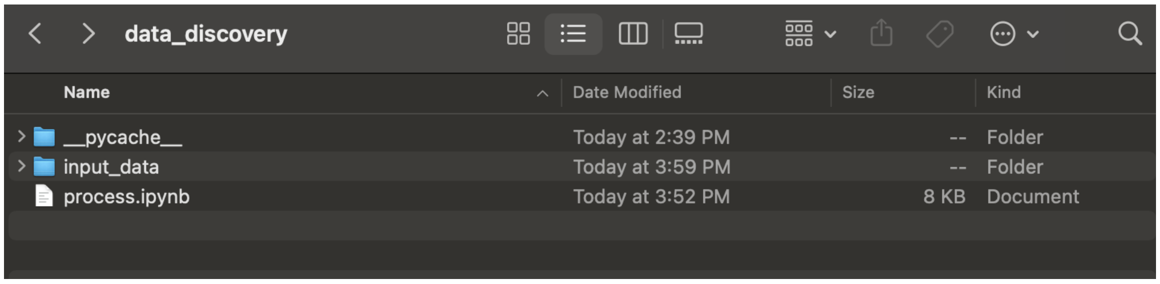
❖ **Check for Remaining Files:**
- Within the ConsolidateData task, there is a complete() function.

        - This function checks if there are any remaining CSV files in the input_data folder.
        - If there are still unprocessed files, the algorithm continues to run and process them.

This process is designed to systematically process a collection of CSV files, categorize them, clean the data, and organize the results into category-specific folders within the output directory. It also maintains the input_data folder by deleting processed files and checking for any remaining unprocessed ones.

Challenges

- ❖ Deciding the technologies necessary for the creation of the Python Script
- ❖ Structuring the data pipeline
- ❖ Deciding what processes the processing algorithm should cover
- ❖ Connecting the data pipeline to the database
    - ➢ Define tables, columns, and relationships to organize and store the data effectively.
    - ➢ Consider data types, constraints, and indexing for optimal database performance.


data_discovery

Name	Date Modified	Size	Kind
__pycache__	Today at 2:39 PM	--	Folder
input_data	Today at 3:59 PM	--	Folder
process.ipynb	Today at 3:52 PM	8 KB	Document

Example of the file structure before processing

Week 4 Report
Sep. 19
Rohit Valmeekam

What was accomplished

Expanded upon data pipeline and tested it on discovered data, including the following improvements:

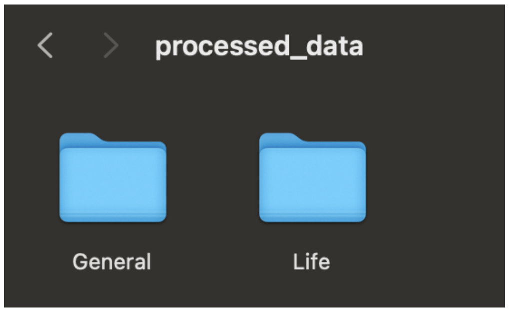
- ❖ Created Support for Other File Extensions (e.g., .xlsx and .txt):
    - ➢ Implemented functionality to handle various file extensions such as .xlsx (Microsoft Excel) and .txt (plain text) in addition to the existing file formats.
    - ➢ Utilized appropriate libraries or modules (e.g., openpyxl for .xlsx and standard file I/O for .txt) to read and process these file types.

    - ➢ Ensured compatibility and data integrity when converting these file formats into the desired .csv format.
- ❖ Consolidated All Data Types into a Singular .csv Format:
    - ➢ Developed a data transformation process to consolidate data from different file types (e.g., .xlsx, .txt, and any others) into a common .csv (Comma-Separated Values) format.
    - ➢ Mapped or transformed data fields from various formats into the corresponding columns in the .csv file.
    - ➢ Handled cases where data structures or headers in different file types varied and ensured they were consistent in the resulting .csv file.
- ❖ Added Error Checking for File Types to Check for Invalid File Types:
    - ➢ Implemented robust error handling mechanisms to validate the file types before processing.
    - ➢ Checked file extensions, MIME types, or other metadata to verify the format's correctness.
    - ➢ Generated informative error messages or logs to alert users or administrators when an invalid file type is encountered.
- ❖ File Type Conversion Logging:
    - ➢ Maintained a log of all file type conversions, including successful conversions and those that encountered errors.
    - ➢ Recorded essential details such as the source file, target format, timestamp, and outcome (success/failure) for auditing and debugging purposes.
- ❖ Data Validation:
    - ➢ Implemented data validation routines to identify and handle inconsistent or erroneous data within the input files.

Challenges

- ❖ Consolidating the data
- ❖ Structuring the data pipeline
- ❖ Deciding what processes the processing algorithm should cover
- ❖ Connecting the data pipeline to the database

Example of the file structure after processing



Week 5 Report

Sep. 26

Rohit Valmeekam

This research report details the development and enhancements of a data processing pipeline using Luigi, a Python library for workflow management. The primary objective of this week was to create an efficient data consolidation and categorization system for input files in various formats (CSV and Excel) and categorize them into specific output folders based on user-defined categories. Most of the work this week was fixing bugs and refactoring code so that other file types could easily be incorporated by utilizing other Python modules.

<u>What Was Accomplished</u>

- ❖ Step 1: Task Definitions
  Two Luigi tasks were defined for this workflow: ReadFile and ConsolidateData.
  - ➢ ReadFile: This task reads input files, supports both CSV and Excel formats, and generates standardized output file paths with a .csv extension.
  - ➢ ConsolidateData: It consolidates and categorizes processed data into output folders based on user-defined categories.
- ❖ Step 2: ReadFile Task Enhancements
  - ➢ Fixed xlsx to csv conversion using os module
  - ➢ Utilized the shutil module to create intermediary files for extension validation
  - ➢ Created framework for the application of the pdf reader so that whenever the pdf reader is ready, it can be easily added to the script in ReadFile
- ❖ Step 3: ConsolidateData Task Enhancements
  - ➢ Dependency Management:
  - ➢ The ConsolidateData task depends on the successful completion of all ReadFile tasks for input files.
  - ➢ This ensures that data consolidation occurs only after all files are processed.
  - ➢ Categorization and Output:
  - ➢ Fixed errors in the script where the program did not correctly delete input files after processing
  - ➢ Fixed errors in the script where the script would corrupt the file after use.

        - ➢ Created framework for the application of the pdf reader so that whenever the pdf reader is ready, it can be easily added to the script in ConsolidateData

<u>**Challenges**</u>

- ❖ Bugs with the refactored code
    - ➢ Not reading file extensions correctly
- ❖ Refactoring code to make space for other file types rather than hard coding to certain file types

Week 6 Report

Oct. 3

Rohit Valmeekam

<u>**What Was Accomplished**</u>

- ★ Analyzed Allstate dataset and how they have scammed big spenders
    - ○ Allstate Corporation sought to update its auto insurance rates in Maryland, citing outdated premiums for its 93,000 customers.
    - ○ The insurer proposed using a complex algorithm called a "customer retention model" to gradually adjust rates rather than implementing new rates all at once.
    - ○ The analysis found that the algorithm seemed to disproportionately target high-spending customers for larger price hikes.
    - ○ Customers already paying high premiums faced potential increases of up to 20%, while those with cheaper policies saw maximum increases of only 5%.
    - ○ Allstate's algorithm also denied meaningful decreases to customers overcharged according to the new risk profile.
    - ○ Maryland ultimately rejected the plan as discriminatory, but Allstate continued to propose similar models in other states.
    - ○ Some states accepted Allstate's retention models, but it remains unclear if they functioned the same way as the Maryland proposal.
    - ○ Personalized pricing based on algorithms that factor in individual behavior raises concerns about fairness and discrimination.

    - Critics argue that certain factors like race, poverty, and health can indirectly influence personalized pricing.
    - Auto insurance is mandatory in most states, making fair pricing crucial.
    - Some states have prohibited "price optimization" and complex pricing models used by insurers.
    - Allstate faced pushback from regulators and consumer advocates over its pricing methods.
    - The complexity of rate filings and lack of transparency make it difficult for consumers to know if they are overpaying.
    - Insurers often do not disclose full details of their pricing algorithms to regulators, further obscuring pricing decisions.
    - The document highlights the challenges regulators face in understanding and regulating insurance pricing models.
- ★ Analyzed how the investigators analyzed the Allstate algorithm
    - Allstate uses personalized pricing algorithms for auto insurance based on big data.
    - The company employs a retention model to estimate customer likelihood of switching insurers due to price changes.
    - Allstate calculates an "ideal price" based on risk factors and proposes a "transition price" to adjust rates gradually.
    - Customers paying higher premiums faced larger rate increases (up to 20%), while those with lower premiums received smaller increases (capped at 5.02%).
    - Middle-aged customers (aged 41 to 62), males, and communities with over 75% nonwhite populations were disproportionately affected.
    - Customers aged 63 and older were less likely to receive significant discounts.
    - Concerns exist about "price optimization" practices, including charging customers based on non-risk factors.
    - Some states have prohibited price optimization, but Allstate continued to use retention models in various states.
    - The report used customer-level data from a 2013 Maryland rate filing and employed statistical and machine learning analysis techniques.

    - Allstate disputed the report's findings and claimed that the Maryland filing was never used.

- ★ Found additional related auto insurance datasets
    - https://emcien.com/sample-data-sets-2/
        - Auto Insurance Claims – Automobile Insurance claims including location, policy type and claim amount
- ★ Found additional datasets
    - https://catalog.data.gov/dataset/fy11-eom-august-face-amount-of-life-insurance-coverage-by-program-by-state
        - Face value of insurance for each administered life insurance program listed by state
    - https://content.naic.org/cipr-topics/insurance-industry-snapshots-and-analysis-reports
        - 2022 Annual Health Insurance Industry Analysis Report
        - 2022 Annual Life/A&H Insurance Industry Analysis Report
        - 2022 Annual Property & Casualty and Title Insurance Industries Analysis Report

Challenges

- ★ Finding additional insurance datasets related to Allstate auto insurance algorithm

Overall Midterm Summary

1. Collaborated with team to find a consensus on the guidelines for our database schema
2. Learned about RDBMS as well as DBMS and the benefits and downsides of each
    a. Weighted the factors of each database type, while also taking into consideration data security
3. Created finished data pipeline that utilizes Python scripting and processing modules such as Luigi to…
    a. Take files from an input folder and categorize them into their respective insurance categories
    b. Read file types such as .txt and .xlsx and transform them into .csv files for ease of use

    c. (if needed) sort out missing values as needed at the user's request
    d. Allow for the user to finish processing hundreds of datasets in a matter of minutes
4. Communicated with Team 1 to create a comprehensive list of the file types needed to process
5. Fixed various user issues with Python program to allow for ease of use
6. Added error handling to help users find errors faster
7. Analyzed Allstate dataset and wrote detailed report on their auto insurance algorithm
8. Found 5 datasets related to Allstate's auto insurance

Next Steps

- ❖ Utilize web scraping to find more datasets related to auto insurance claims
    - ➢ Create web scraping program that will allow me to search for certain keywords on the internet
- ❖ Integrate data pipeline to database system