CST 1510 Assessment 2

Middlesex University Dubai

Multi-Domain Intelligence Platform Report

Student Details:

- Name: Stepan Pelc
- Student ID: M01087679
- Program: BSc Artificial Intelligence and Data Science

GitHub Repository: [https://github.com/StepanPelcx/multi_domain_platform]

1. Introduction

The Multi-domain Intelligence Platform is a web application developed using Python and Streamlit. The application is divided into two main parts. The first part is user authentication, and the second part is the three domains. The authentication takes care of all the user's needs. So, it handles user registration or login and the hashing of all the passwords before storing them in a "users" table in the database. In the authentication, there are many other functions for handling users, such as changing a password, getting the user's role, or checking the password strength. The application also handles the case where the user is not logged in, so they cannot access any of the three domains.

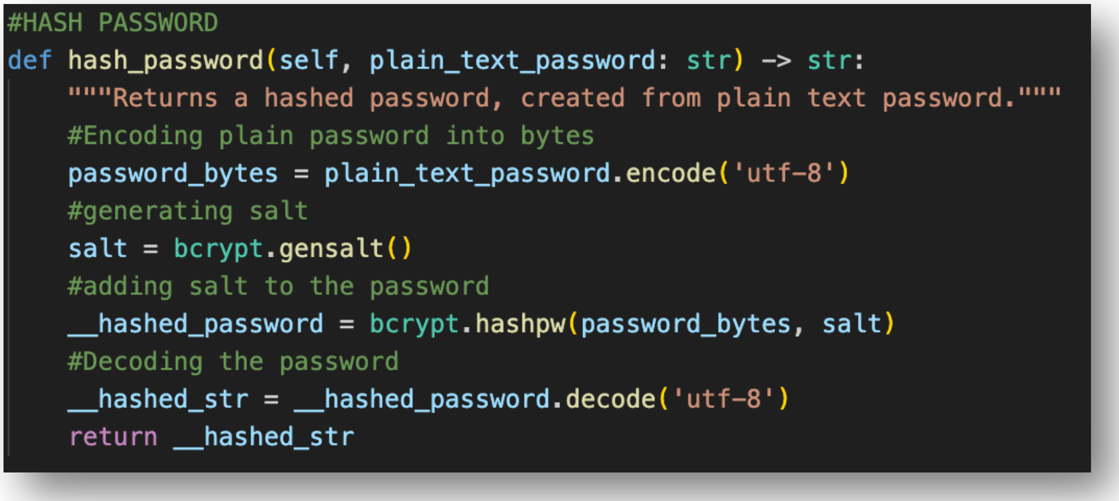
Moving on to the three domains, all of them are operating with data, which are stored in a database, in which for each domain, there is a table to store the samples. The application is divided into three pages for each of the fields, and each page is then divided into three tabs. The first tab shows the user a visualization of the data through graphs. Those graphs are based on user input, which means that the user can change the graph type or variable displayed on each axis. The second tab allows users to manage and analyze the data using CRUD functions, for example, inserting or deleting samples by their id. The third tab is focused on AI-powered assistants. Through the sidebar, the user can choose if they want to analyze a sample of the data or chat with an AI assistant with expertise in that field.

This application was designed for all three domains, and Tier 3 is implemented, because all the domains contain mandatory features, and are done using Object Oriented Programming.

2. How I built the system

2.1 Security &Database

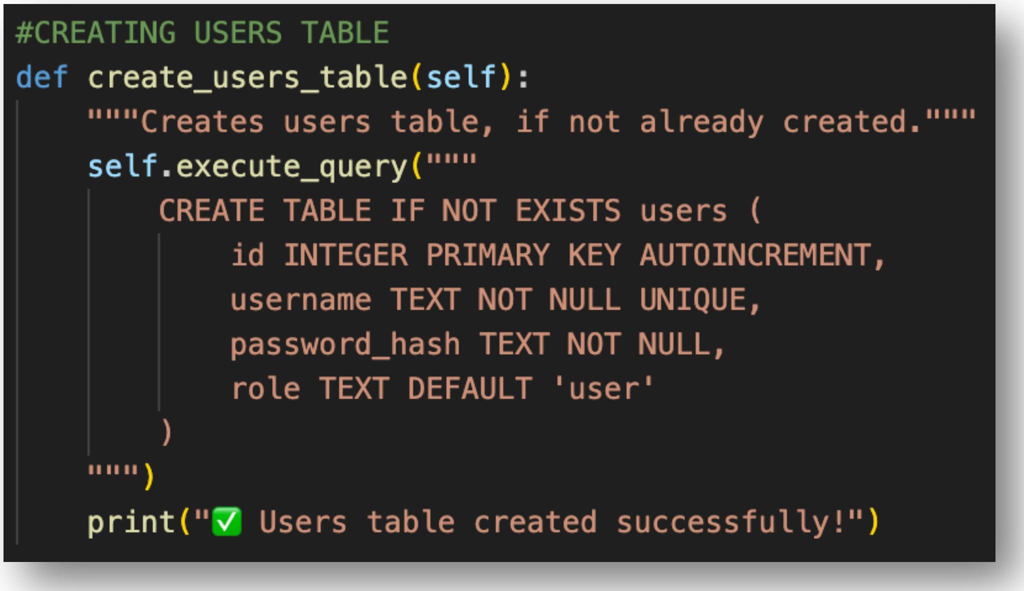
User data protection is a very important part when developing a web application, which is why we need to store all the user data securely. So, each time a user successfully registers or logs in, the password is then hashed using a hash function. This function encodes the plain-text password into a hashed password and adds salt to it, so the password is well protected before moving it to the database table for storage. For example, if we did not hash each password, anyone with access to the database could take that information and leak or use the password to access the application and use the services. We also hash the password before moving it into the database, because with that, we remove the possibility of leaking the password throughout the moving process.



```
#HASH PASSWORD
def hash_password(self, plain_text_password: str) -> str:
    """Returns a hashed password, created from plain text password."""
    #Encoding plain password into bytes
    password_bytes = plain_text_password.encode('utf-8')
    #generating salt
    salt = bcrypt.gensalt()
    #adding salt to the password
    __hashed_password = bcrypt.hashpw(password_bytes, salt)
    #Decoding the password
    __hashed_str = __hashed_password.decode('utf-8')
    return __hashed_str
```

Figure 1 - Password hash

For the storage of the data, my application has one database called “intelligent_platforms.db”. This database consists of four tables. The first table is used to store data from the users. For each user, the database stores their username, password, role, and each user has a unique id. The Other three tables are used to store samples for each of the domains. Every domain has specific properties, so for each table, the structure is the same, but the column names and count change. For the three domains, the first five hundred samples are migrated from CSV files, but the application allows the user to use CRUD functions to change the tables by adding, deleting, or changing the samples.



```
#CREATING USERS TABLE
def create_users_table(self):
    """Creates users table, if not already created."""
    self.execute_query("""
        CREATE TABLE IF NOT EXISTS users (
            id INTEGER PRIMARY KEY AUTOINCREMENT,
            username TEXT NOT NULL UNIQUE,
            password_hash TEXT NOT NULL,
            role TEXT DEFAULT 'user'
        )
    """)
    print("✅ Users table created successfully!")
```

Figure 2 - Users table (DB)

Security is a crucial part of this project, because the application is working with user-sensitive information, such as username, but most of all, the password. Because the user trusts the workspace with this information, the application must protect confidentiality, integrity, and access control. If we came across a data leakage, it could lead to unauthorized access, identity compromise, or misuse of personal data. Because of that, the application contains one way password hashing mechanism to minimize the risk of any data breach. The use of bcrypt protects the user data against common attacks.

2.2 System Structure

The Multi-domain Intelligence Platform lets the user interact with the system through a Streamlit web interface, which is the presentation layer of the application. This interface lets users navigate between pages, tabs, or sidebars. Before the user can access any domains, all their requests are passed through the authentication. The authentication handles user registration, login, and password hashing. Once the authentication is complete, the user can freely access all the domains. Each domain is independent, and the data is stored in separate tables. So, for example, if the user wants to update the status of a cybersecurity incident, they must log in/register from the home page, and the authentication system would check the user's information in the database, or for registration, it will store the information there and grant them access to the domains. We use Streamlit session state to save the information if we are logged in, in the session. Then the user can go to the cybersecurity domain, choose the tab with the CRUD functions. They can choose the update incident function and fill out the necessary information for a successful update. After they submit their change, the system would check if the information was provided in the correct type and access the database table to change the values.

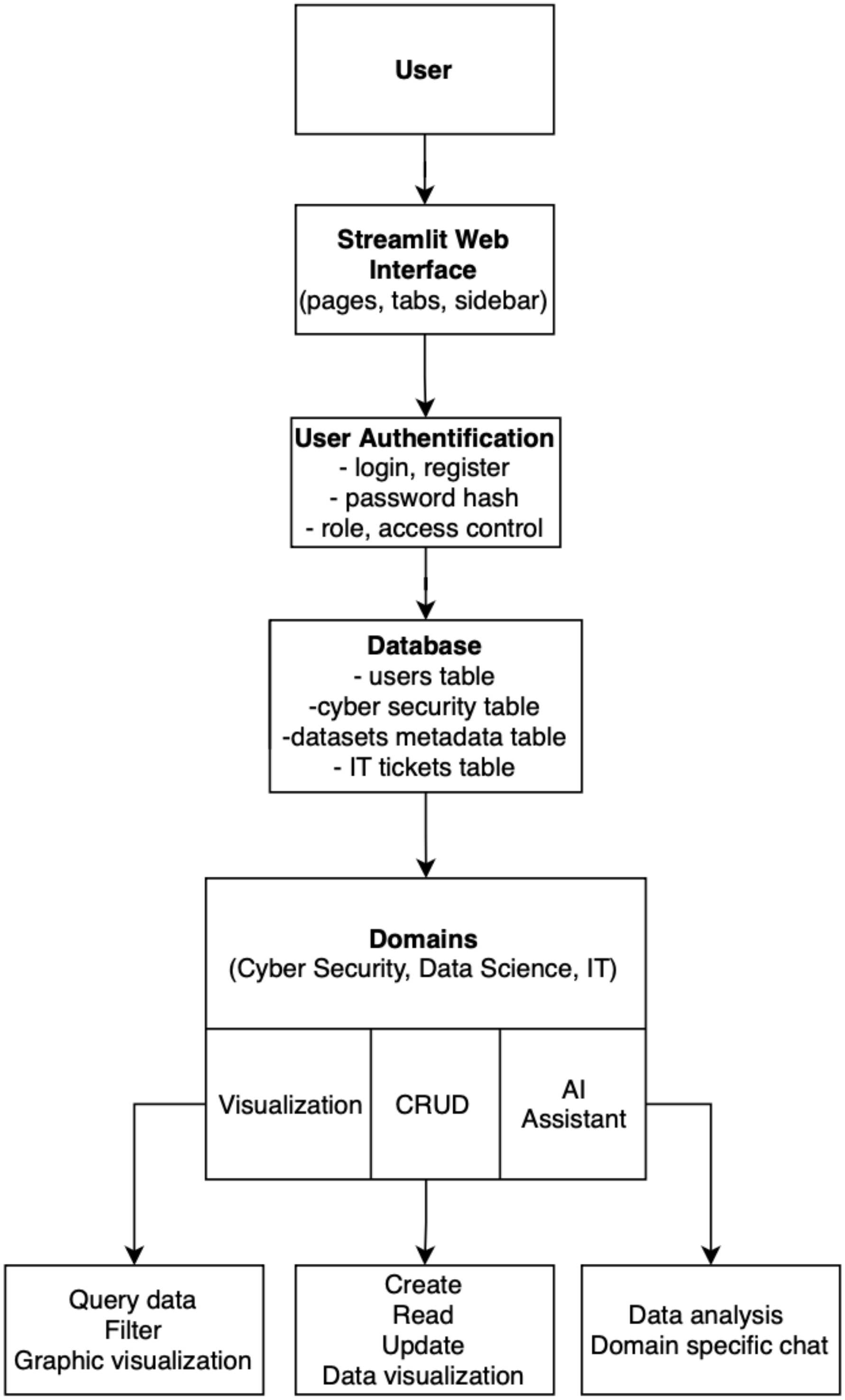


Figure 3 - Data-Flow Visualization

2.3 Code Organization (OOP Refactoring)

After I created all the functions for the user authentication, all the domains, and added AI-powered assistants, the whole code was unclear and a little bit messy. For someone who sees the code for the first time, it was very hard to orient and see how everything works. That is when implementing OOP makes perfect sense. I created new files to store each class for all the functions that I had. I moved all the functions to their respective classes, and with that, the code is so much cleaner and better formatted. Because some of the classes depend on each other or work with each other, the final code for each page looks cleaner and has been reduced by some margin, because all the functions are hidden in the classes. It also allowed me to edit my code or add functions very easily, without breaking the code and raising errors.

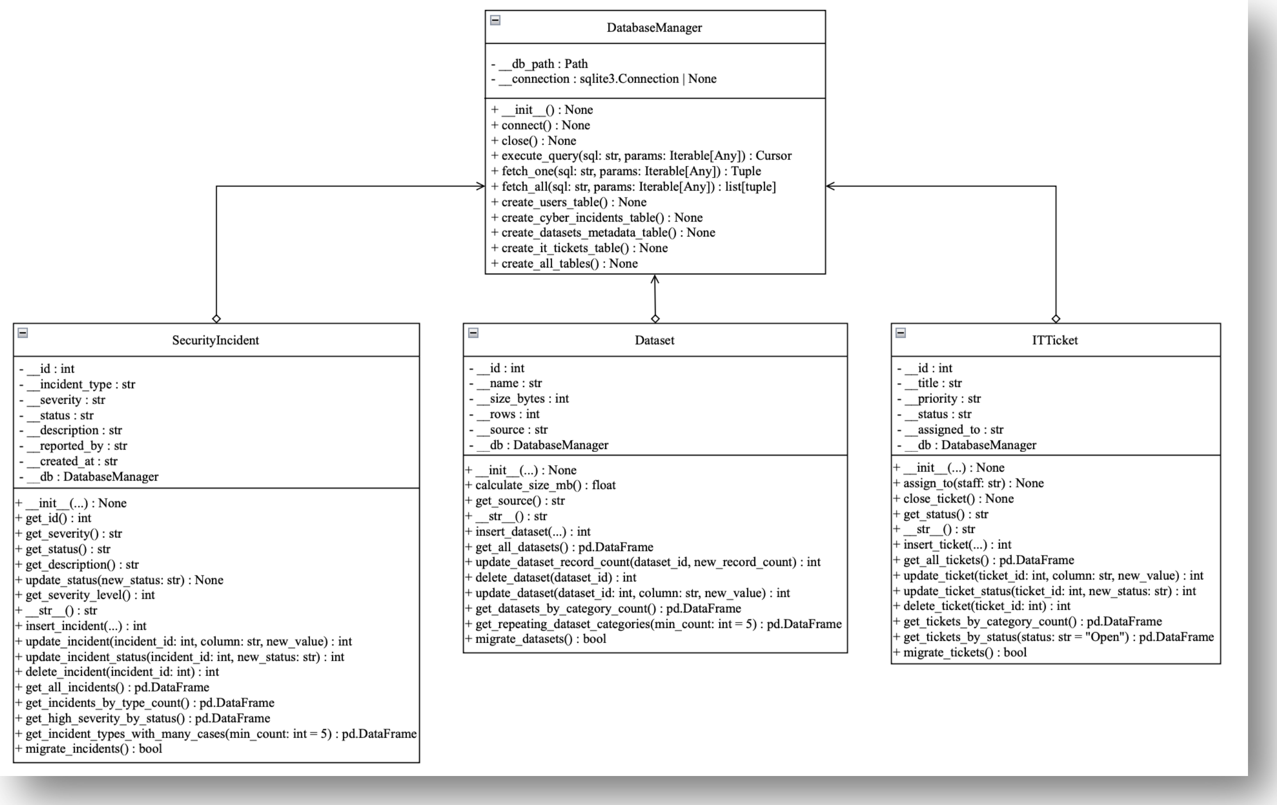

DatabaseManager
- __db_path : Path
- __connection : sqlite3.Connection | None
+ __init__() : None
+ connect() : None
+ close() : None
+ execute_query(sql: str, params: Iterable[Any]) : Cursor
+ fetch_one(sql: str, params: Iterable[Any]) : Tuple
+ fetch_all(sql: str, params: Iterable[Any]) : list[tuple]
+ create_users_table() : None
+ create_cyber_incidents_table() : None
+ create_datasets_metadata_table() : None
+ create_it_tickets_table() : None
+ create_all_tables() : None
SecurityIncident
- __id : int
- __incident_type : str
- __severity : str
- __status : str
- __description : str
- __reported_by : str
- __created_at : str
- __db : DatabaseManager
+ __init__(...) : None
+ get_id() : int
+ get_severity() : str
+ get_status() : str
+ get_description() : str
+ update_status(new_status: str) : None
+ get_severity_level() : int
+ __str__() : str
+ insert_incident(...) : int
+ update_incident(incident_id: int, column: str, new_value) : int
+ update_incident_status(incident_id: int, new_status: str) : int
+ delete_incident(incident_id: int) : int
+ get_all_incidents() : pd.DataFrame
+ get_incidents_by_type_count() : pd.DataFrame
+ get_high_severity_by_status() : pd.DataFrame
+ get_incident_types_with_many_cases(min_count: int = 5) : pd.DataFrame
+ migrate_incidents() : bool
Dataset
- __id : int
- __name : str
- __size_bytes : int
- __rows : int
- __source : str
- __db : DatabaseManager
+ __init__(...) : None
+ calculate_size_mb() : float
+ get_source() : str
+ __str__() : str
+ insert_dataset(...) : int
+ get_all_datasets() : pd.DataFrame
+ update_dataset_record_count(dataset_id, new_record_count) : int
+ delete_dataset(dataset_id) : int
+ update_dataset(dataset_id: int, column: str, new_value) : int
+ get_datasets_by_category_count() : pd.DataFrame
+ get_repeating_dataset_categories(min_count: int = 5) : pd.DataFrame
+ migrate_datasets() : bool
ITTicket
- __id : int
- __title : str
- __priority : str
- __status : str
- __assigned_to : str
- __db : DatabaseManager
+ __init__(...) : None
+ assign_to(staff: str) : None
+ close_ticket() : None
+ get_status() : str
+ __str__() : str
+ insert_ticket(...) : int
+ get_all_tickets() : pd.DataFrame
+ update_ticket(ticket_id: int, column: str, new_value) : int
+ update_ticket_status(ticket_id: int, new_status: str) : int
+ delete_ticket(ticket_id: int) : int
+ get_tickets_by_category_count() : pd.DataFrame
+ get_tickets_by_status(status: str = "Open") : pd.DataFrame
+ migrate_tickets() : bool

Figure 4 - OOP (DB Connection to Domains)

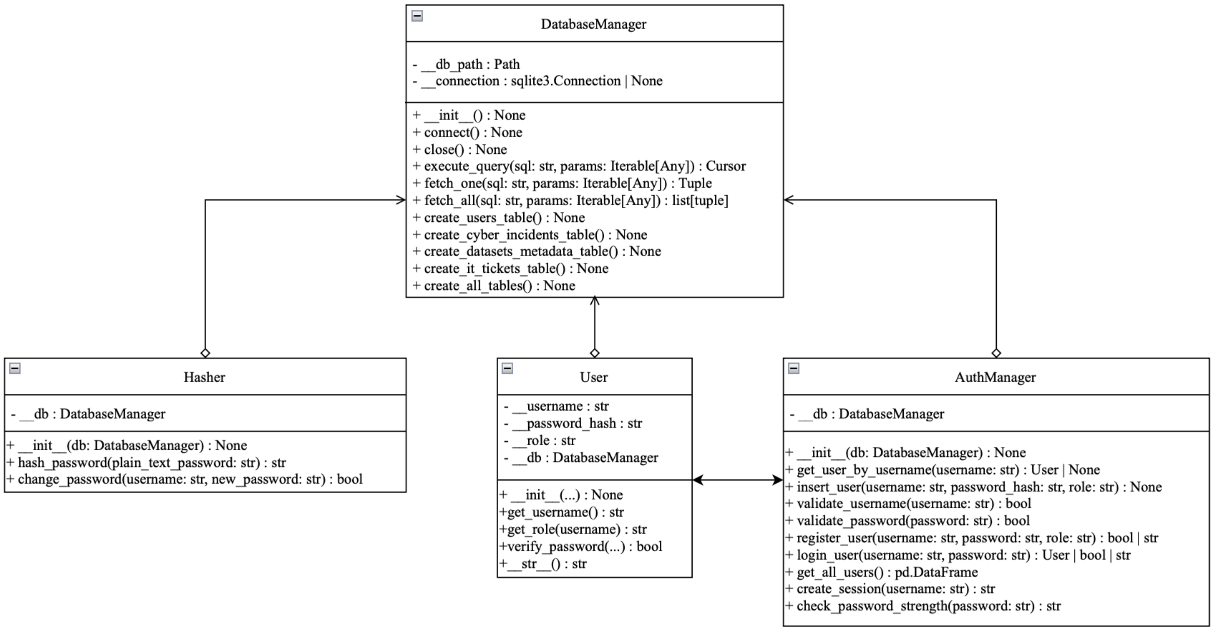

DatabaseManager
- __db_path : Path
- __connection : sqlite3.Connection | None
+ __init__() : None
+ connect() : None
+ close() : None
+ execute_query(sql: str, params: Iterable[Any]) : Cursor
+ fetch_one(sql: str, params: Iterable[Any]) : Tuple
+ fetch_all(sql: str, params: Iterable[Any]) : list[tuple]
+ create_users_table() : None
+ create_cyber_incidents_table() : None
+ create_datasets_metadata_table() : None
+ create_it_tickets_table() : None
+ create_all_tables() : None
Hasher
- __db : DatabaseManager
+ __init__(db: DatabaseManager) : None
+ hash_password(plain_text_password: str) : str
+ change_password(username: str, new_password: str) : bool
User
- __username : str
- __password_hash : str
- __role : str
- __db : DatabaseManager
+ __init__(...) : None
+get_username() : str
+get_role(username) : str
+verify_password(...) : bool
+__str__() : str
AuthManager
- __db : DatabaseManager
+ __init__(db: DatabaseManager) : None
+ get_user_by_username(username: str) : User | None
+ insert_user(username: str, password_hash: str, role: str) : None
+ validate_username(username: str) : bool
+ validate_password(password: str) : bool
+ register_user(username: str, password: str, role: str) : bool | str
+ login_user(username: str, password: str) : User | bool | str
+ get_all_users() : pd.DataFrame
+ create_session(username: str) : str
+ check_password_strength(password: str) : str

Figure 5 - OOP (DB Connection to Authentication)

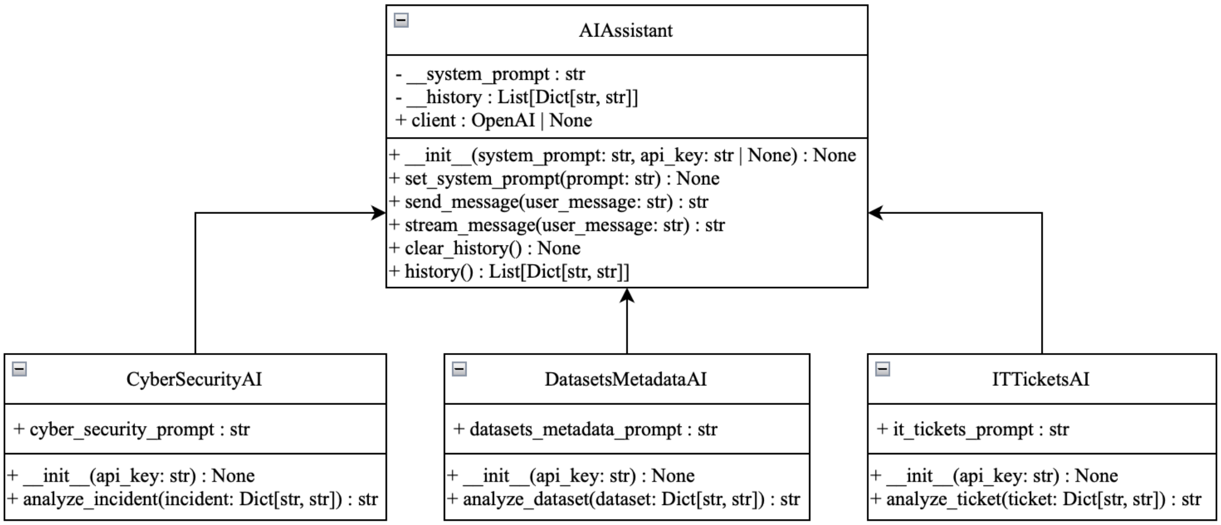
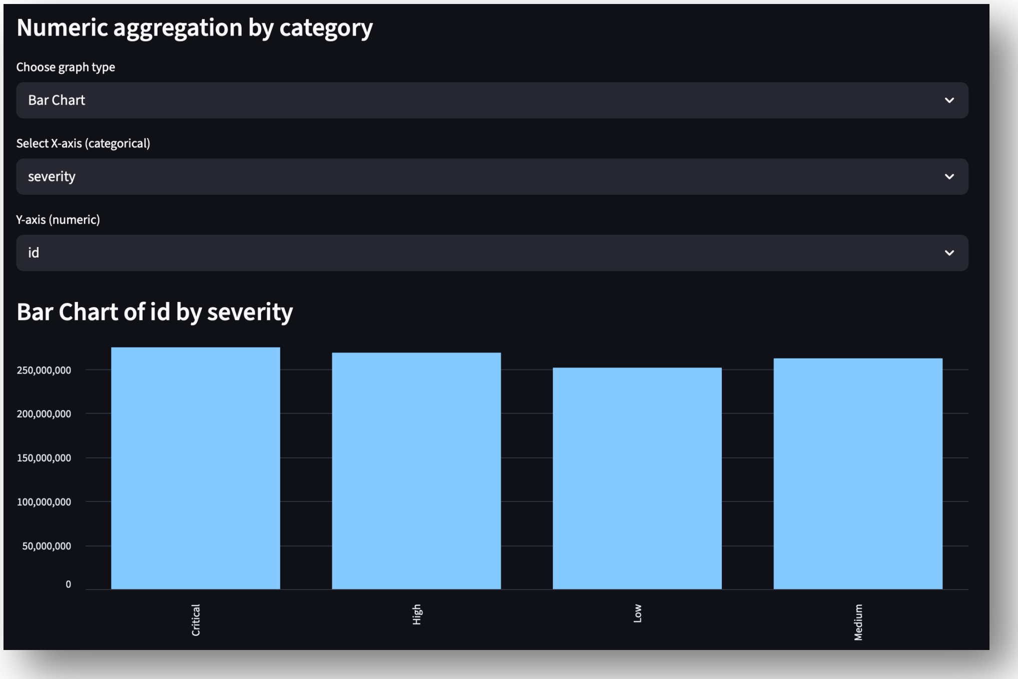

AIAssistant
- __system_prompt : str
- __history : List[Dict[str, str]]
+ client : OpenAI | None
+ __init__(system_prompt: str, api_key: str | None) : None
+ set_system_prompt(prompt: str) : None
+ send_message(user_message: str) : str
+ stream_message(user_message: str) : str
+ clear_history() : None
+ history() : List[Dict[str, str]]
CyberSecurityAI
+ cyber_security_prompt : str
+ __init__(api_key: str) : None
+ analyze_incident(incident: Dict[str, str]) : str
DatasetsMetadataAI
+ datasets_metadata_prompt : str
+ __init__(api_key: str) : None
+ analyze_dataset(dataset: Dict[str, str]) : str
ITTicketsAI
+ it_tickets_prompt : str
+ __init__(api_key: str) : None
+ analyze_ticket(ticket: Dict[str, str]) : str

Figure 6 - OOP (Connection between AI Assistants)

2.4 Key Features

For visualization, I created some charts, which are displayed based on the user input. The user can choose the graph type they want to see and choose what they want to display on each axis. The second chart is a pie chart, and the user also chooses what they want to display and how many counts they want to display. The chart will then show them the percentage of the category in the data frame. Through that, the user can analyze the data and see what data they are working with.



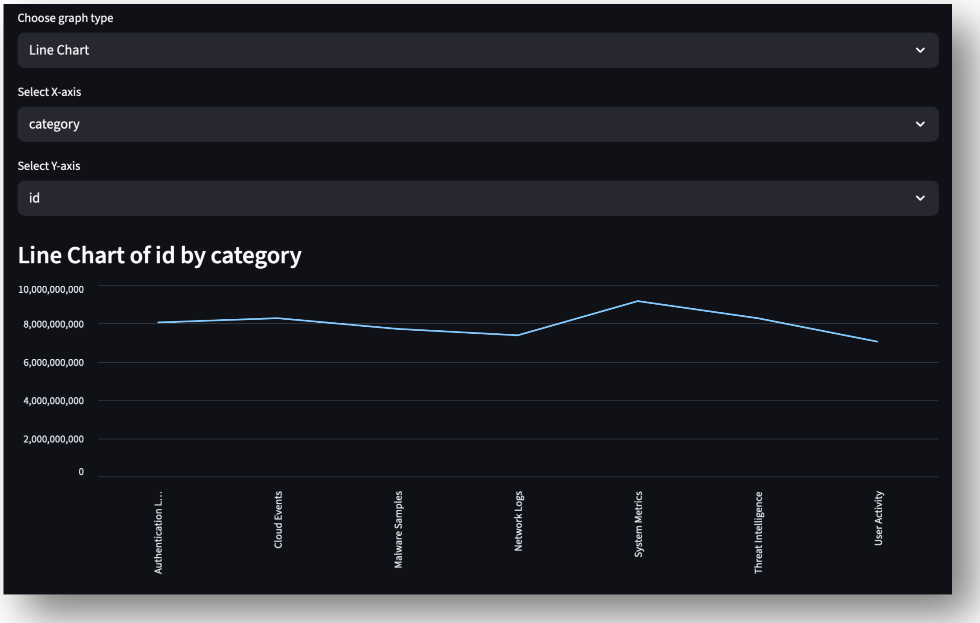
Numeric aggregation by category

Choose graph type: Bar Chart

Select X-axis (categorical): severity

Y-axis (numeric): id

Figure 7 – Bar Chart (Cyber Security)



Choose graph type

Line Chart

Select X-axis

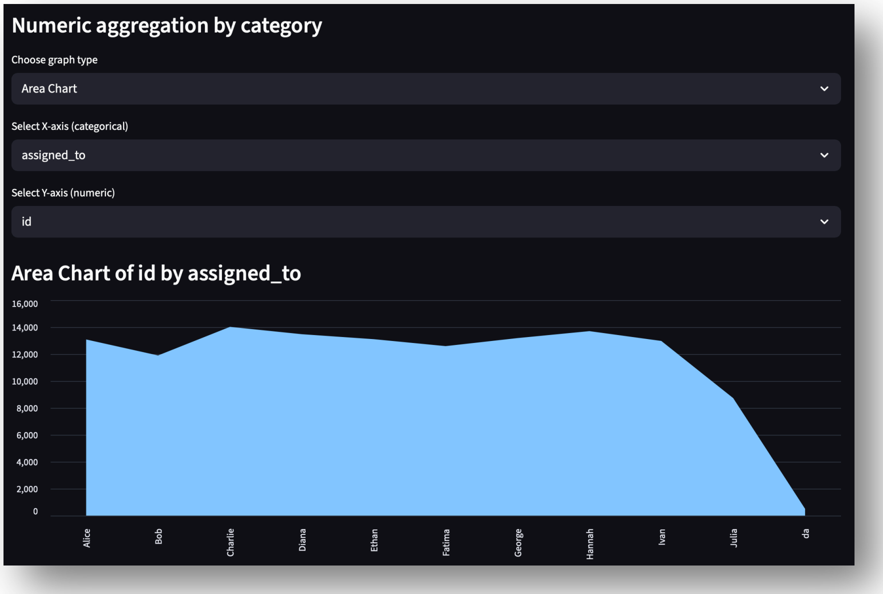
category

Select Y-axis

id

Line Chart of id by category

Figure 8 - Line Chart (Datasets Metadata)



Numeric aggregation by category

Choose graph type

Area Chart

Select X-axis (categorical)

assigned_to

Select Y-axis (numeric)

id

Area Chart of id by assigned_to

Figure 9 - Area Chart (IT Tickets)

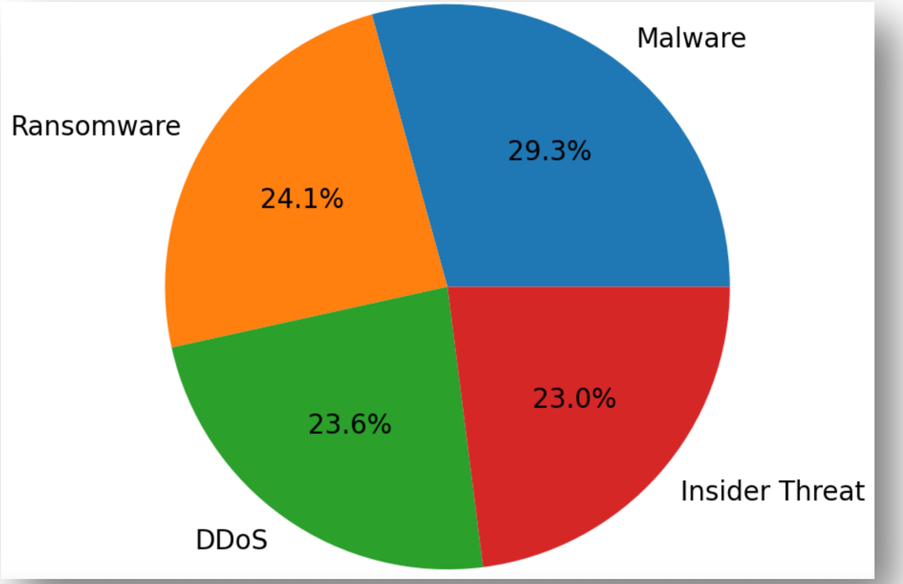
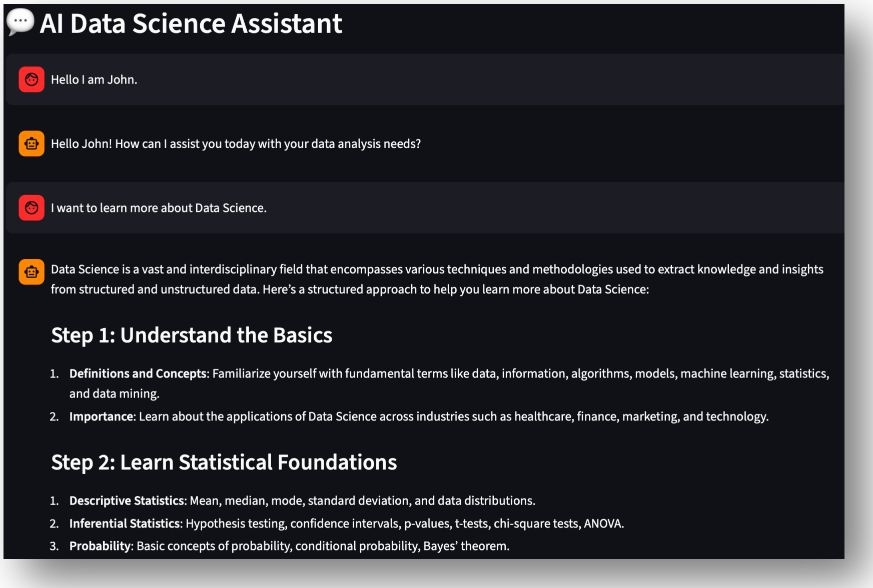


Figure 10 - Pie Chart (Cyber Security)

The AI-powered assistants are a part of each domain. In the AI tab, the user can choose between chat, where they can chat with the assistant specialized in that field, or analysis, where the user can choose one sample from the data frame and let the AI assistant analyze it. This assistance helps the user with the analysis of the data, and it can provide key information for the user's problems.



AI Data Science Assistant

Hello I am John.

Hello John! How can I assist you today with your data analysis needs?

I want to learn more about Data Science.

Data Science is a vast and interdisciplinary field that encompasses various techniques and methodologies used to extract knowledge and insights from structured and unstructured data. Here's a structured approach to help you learn more about Data Science:

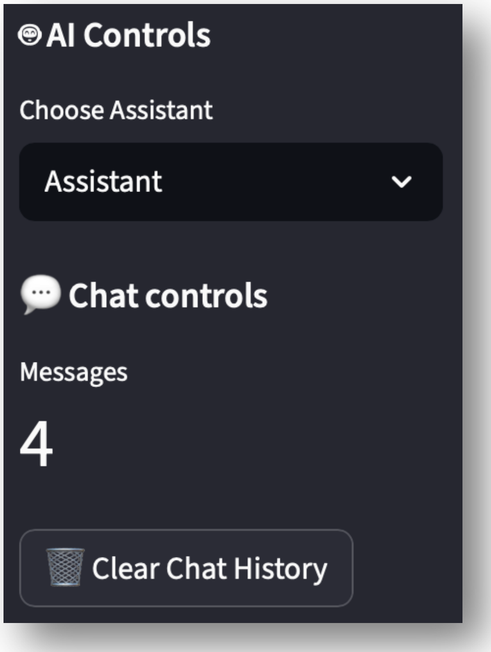
Step 1: Understand the Basics

1. **Definitions and Concepts**: Familiarize yourself with fundamental terms like data, information, algorithms, models, machine learning, statistics, and data mining.
2. **Importance**: Learn about the applications of Data Science across industries such as healthcare, finance, marketing, and technology.

Step 2: Learn Statistical Foundations

1. **Descriptive Statistics**: Mean, median, mode, standard deviation, and data distributions.
2. **Inferential Statistics**: Hypothesis testing, confidence intervals, p-values, t-tests, chi-square tests, ANOVA.
3. **Probability**: Basic concepts of probability, conditional probability, Bayes' theorem.

Figure 11 - AI Assistant Chat (Data Science)



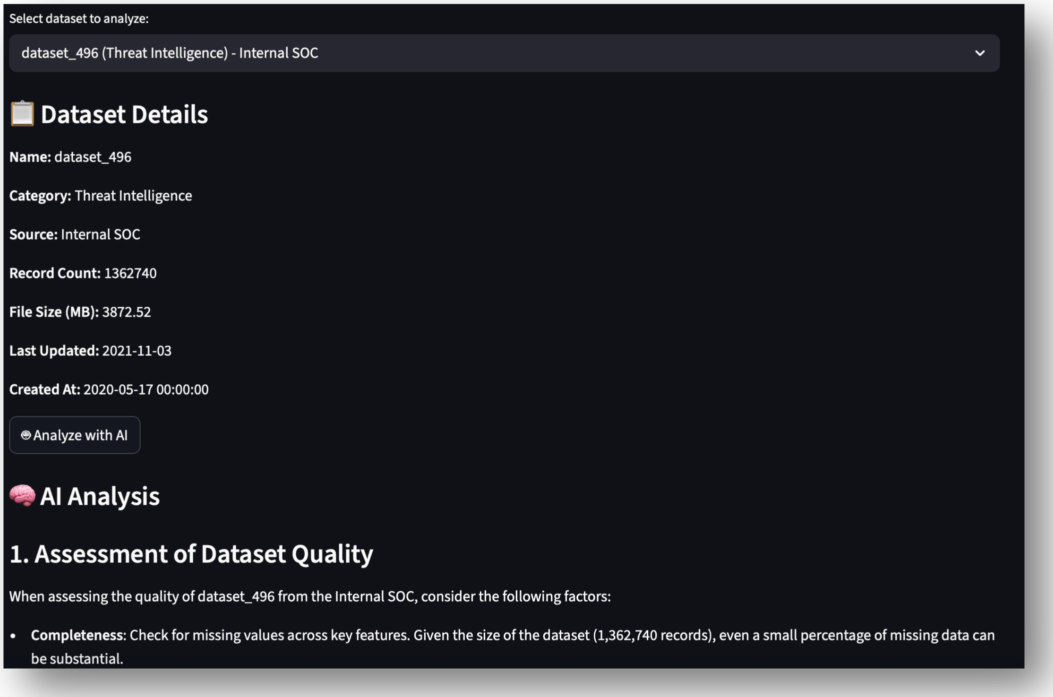
Choose Assistant

Assistant

Messages

4

Figure 12 - AI Assistant Sidebar Controls



Select dataset to analyze:

dataset_496 (Threat Intelligence) - Internal SOC

📋 Dataset Details

Name: dataset_496

Category: Threat Intelligence

Source: Internal SOC

Record Count: 1362740

File Size (MB): 3872.52

Last Updated: 2021-11-03

Created At: 2020-05-17 00:00:00

🧠 AI Analysis

1. Assessment of Dataset Quality

When assessing the quality of dataset_496 from the Internal SOC, consider the following factors:

- **Completeness:** Check for missing values across key features. Given the size of the dataset (1,362,740 records), even a small percentage of missing data can be substantial.

Figure 13 - AI Assistant Sample Analysis (Datasets Metadata)

3. Reflection & Conclusion

3.1 What I Learned

When I started this project in week 7, all I had was the authentication function. I remember that at that time, I was not fully sure what I was creating all the functions for, and I could not see the bigger picture. But with every week, my project was growing and started to make more sense, because I was adding more functions, and it all started to come together. Throughout the whole process of developing and working on this project, I have improved my programming skills quite a lot, and I also learned many new things. For me, the most valuable skill that I learnt is to develop a fully working program with many smaller pieces that must work together. In my eyes, another important part is that I got to understand Object Oriented Programming, which makes a big project very easy to manage and make changes to, and very clean looking. Throughout the process, I did not fully understand some of the features that I was implementing, and I had to spend more time trying to understand the logic. But as I was working on the project, my understanding became very clear for everything that I included.

3.2 Challenges I faced

The biggest technical problem that I had was with finishing week 8, working with the database. I did not face any challenges with understanding how the databases work, but the biggest challenge was to understand the Python implementation logic. In week 8, I had to create many functions, and I each time I finished a part of the lab, I was not sure if I did it correctly or not. But as I studied the subject more, I became confident in working with it. So, I learned that even if something seems scary at first, if you start and work hard to understand it, in time it will show that it is not that frightening. Other than that, I did not face any big issues with any other topic covered, and the journey was smooth.

3.3 Conclusion

I would consider the implementation of Object-Oriented Programming as the main achievement in my project, because it changed the whole project structure and made it far simpler than it was. I also think that learning to work with databases will be a key part of my future journey. And I cannot forget about the authentication. It could seem that it is very simple, but learning to understand the logic behind password hashing and the safety of storing data is important everywhere.

For the improvement of the project, I would include more functionality for different user roles. So, each user role would have access to different functions of the website. For example, a user role data scientist will gain access to more advanced analytics and statistics.