

Software Development with Agile Techniques

Hackcamp

Group **IRIS**:

Trinity Booth: Scrum Master

Luke Slattery: Product Owner

Berk Mehmedov: Database Architect

Waleed Khalid: Command Line Developer

Alex Wilson: Security Master

Ala Diab: Developer / Documentation & Reporting Specialist

Salma Aloush: Developer / Documentation & Reporting Specialist

Table of Contents

1. **Introduction**
2. **Sprint 1: Prototype Masking Script**
 - 3.1 Product Backlog
 - 3.2 Sprint Backlog
 - 3.3 Work Time Estimates
 - 3.4 Work Item Selection
 - 3.5 Daily Stand-Up Meetings
 - 3.6 Sprint Burndown Chart
 - 3.7 Client Introduction Meeting
 - 3.8 Customer Demonstration Meeting Summary
3. **Sprint 2: Database**
 - 4.1 Product Backlog
 - 4.2 Sprint Backlog
 - 4.3 Work Time Estimates
 - 4.4 Daily Stand-Up Meetings
 - 4.5 Customer Demonstration Meeting
4. **Sprint 3: Developing Final Script**
 - 5.1 Product Backlog
 - 5.2 Sprint Backlog
 - 5.3 Work Time Estimates
 - 5.4 Daily Stand-Up Meetings
 - 5.5 Customer Demonstration Meeting Summary
 - 5.6 Work Item Selection
 - 5.7 Requirements Analysis and System Design
5. **Details of Solution Testing and Evaluation**
6. **Conclusions, Summary, and Evaluation of Achieved Results**
7. **References**
8. **Appendices**

1. Introduction

This document outlines the progress of Sprints 1, 2, and 3, focusing on the key activities and achievements. The primary goal of Sprint 1 was to develop a functional prototype of the data masking script, laying the foundation for the entire system. The team successfully completed the prototype ahead of schedule by applying Agile methods, breaking tasks into manageable parts to ensure efficiency.

Throughout our project, the team adhered to agile methodologies as pushed by the scrum master during each sprint, with the guidance of a trello board. Scrum master highlighted the importance of effective communication and collaboration. Regular updates, client meetings, and frequent sprint backlog refinements ensured that everyone stayed aligned with project goals.

This script will enable a user to transfer data from a source to a target database, masking the sensitive information as required. This project report will detail the process of this project.

2. Sprint 1: Prototype Masking Script

3.1. Product Backlog

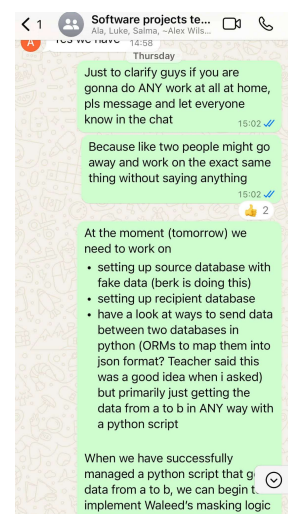
The Product Backlog for Sprint 1 focused on delivering the foundation for the project:

1. **Hardcoded test Database:**
 - Use integrated data hardcoded into the python script, as an initial test bench for the functionality of the data masking tool.
2. **Data Masking Logical Implementation:**
 - Develop the first set of rudimentary masking techniques, including character scrambling and fixed value substitution.
3. **Command-Line Interface (CLI):**
 - Create and structure CLI interface :allowing users to select their data and masking level preset, Interface also offers error handling/logging functionality.
 - At this moment in time the returned Masked data will be outputted exclusively to the CLI
4. **Testing and Validation:**
 - Assess the accuracy of masking techniques, testing on these on many different data sets
 - Force the script to trigger It's error correction : ensuring this happens appropriately and are helpful
 - ensure CLI output is consistent.

This backlog prioritised essential features for the prototype while allowing flexibility for future sprints.

3.2. Sprint backlog: Tasks selected for Sprint 1 included:

1. Database Setup:
 - Install and configure PGAdmin.
 - Populate the source database with mock sensitive data and prepare the target database.
2. Masking Logic Development:
 - Implement character scrambling and fixed masking rules.
 - Hard-code masking levels based on security requirements.
3. CLI Development:
 - Enable database connectivity through *argparse* and implement logging for masking tasks.
4. Collaboration and Documentation:

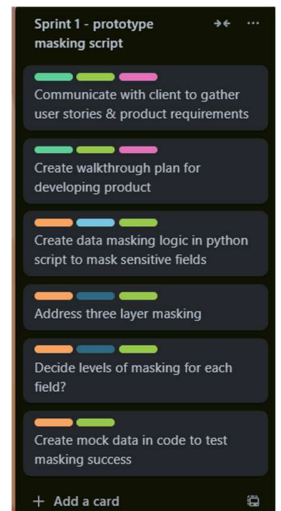
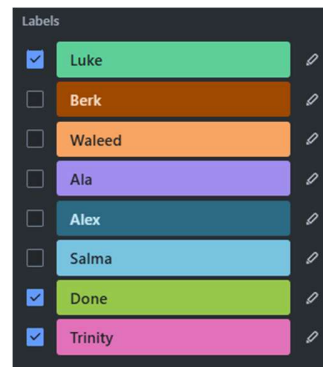


- Document progress incrementally through GitHub commit track tasks and ensure alignment.
- Conveniently our later discussed daily stand up's & digital communication mean frequent updates, thoughts and future plans are all documented through Trello.

3.3. Work time estimates:

Sprint 1 was completed by Friday of Week 1, with a working prototype script finalised by Wednesday. We accredit this quick turnaround to the following techniques:

- Task Allocation:
 - Waleed led scripting
 - Berk and Alex focused on database connectivity and masking logic.
 - Ala and Salma managed documentation, progress tracking and developed the JSON templates.
 - Luke checked on the progress of these tasks, ensuring coherence. Within the first few days ensuring full understanding with our Clients vision for the software was fully understood by all the group was a task in itself. Research on how masking tools worked, masking techniques and how the JSON script & our CLI would look, all to make our software as most usable & effective was then relayed back to the team in a shared document.
- Collaboration: Trinity facilitated daily communication through group chats and predominantly leading our daily-stand-ups to ensure progress and reassign tasks where beneficial.
- Alignment with vision : Luke spoke to all divisions of the team checking how their process aligned with the clients vision and how it integrates with what other teams were working on. Always considering future plans and ideas of improvement ensured we did not make any decisions we may later have come to regret.
- Digital Tools: GitHub and Trello ensured tracking and avoided task overlap, Shared documents allowed easy additions to documentation.



3.4. Work item selection:

In sprint 1 (and all sprints), tasks were divided based on roles. The scrum master ensured communication was effective whilst delivering instructions for the following day. Luke and Trinity arranged & partook in the client meetings then updating & devising sprints accordingly, whilst the development progressed through the prototype. Developers specified with security roles began addressing three-layer-masking, and decided on masking levels for the prototype to be hard-coded.

3.5. Daily stand-up meetings:

Daily meetings facilitated progress tracking and problem-solving:

	SPRINT	ACTIVITY	NOTES
December 2	Sprint 1 (Week 1)	Initial team and sprint planning	Defined roles, user stories, and sprint goals. Created a Trello board to track progress.
December 3	Sprint 1 (Week 1)	Developed initial data masking logic	Focused on three-layer masking; initial masking techniques such as scrambling implemented.
December 4	Sprint 1 (Week 1)	Client meeting 1: Requirements clarification	Clarified project scope, user stories, and masking logic. Adjustments made as necessary.
December 5	Sprint 1 (Week 1)	Created mock data and tested masking logic	Generated own data; testing showed partial success but revealed some issues.
December 6	Sprint 1 (Week 1)	Client meeting 2: Follow-up and feedback	The client asked us to refine the JSON templates and improve masking logic, which we did.
December 7	Sprint 1 (Week 1)	Internal testing and debugging	Code worked initially but later crashed. Identified and resolved issues in the script.

```

Commit a1bb45d
WaleedKhalid22 authored 5 days ago (Verified)

Add files via upload

cli.py master

1 file changed +72 -0 lines changed

cli.py
... @@ -8,8 +1,72 @@
1 + import argparse
2 + import pandas as pd
3 +
4 + def load_data(database):
5 +     # Simulate checking for database existence
6 +     if database == "non_existing_db":
7 +         print("Error: Database does not exist.")
8 +         return None # Return None or raise an exception
9 +
10 +
11 + # Placeholder function to load data from a database
12 + print(f"Loading data from {database}...")
13 + return pd.DataFrame({
14 +     'name': ['Waleed', 'Ali', 'Omar'],
15 +     'email': ['waleed@example.com', 'ali@example.com', 'omar@example.com'],
16 +     'ssn': ['123-45-6789', '987-65-4321', '555-123-4567']
17 + })
18 +
19 + def mask_data(data, template, sensitivity):
20 +     print(f"Masking data using template: {template}")

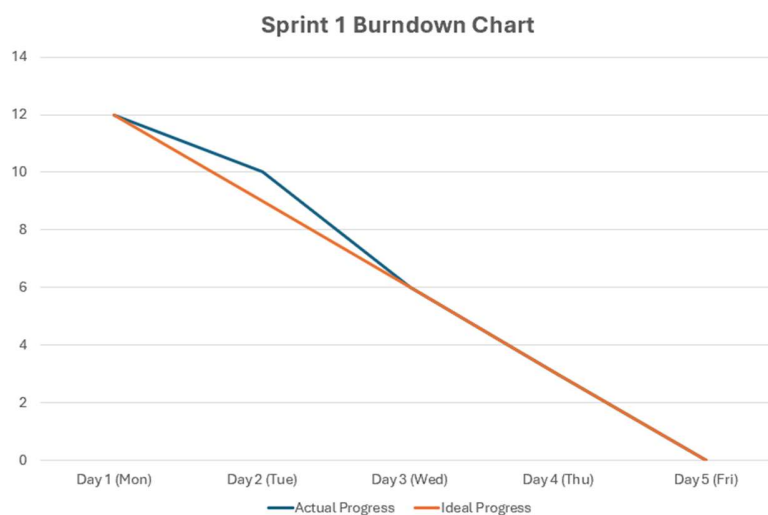
```

- **December 2 (Monday):** Roles and sprint tasks were defined. Waleed took responsibility in starting the initial script, while Berk and Alex began database setup. Ala and Salma started documenting progress.

- **December 3 (Tuesday):** The team reviewed masking logic progress. Waleed worked on initial implementation, while Luke & Alex researched advanced techniques.
- **December 4 (Wednesday):** Realising the sprint was not detailed enough, the team reconciled with our improved knowledge, understanding & ideas to gain a better outlook.
- **December 6 (Friday):** Discussed client feedback as a group and brainstormed adjustments to align the product with new user stories.

These meetings ensured all challenges were promptly addressed and progress was aligned with sprint goals.

3.6. Sprint burndown chart:



*The **x-axis** represents the days of our sprint, while the **y-axis** indicates the number of tasks remaining.*

3.7. Client introduction meeting :

The client introduction meeting provided clarity on the project's objectives and expectations. Key points discussed included:

1. Primary use cases for the tool.
2. Masking requirements for sensitive fields.
3. Deliverables and timelines.

This discussion informed the initial prototype design and set clear priorities for the sprint.

3.8. Customer Briefing with Client

Customer Demonstration Meeting Summary - Sprint 1

1. Meeting Overview

- Date: [December 2nd, 2024]
- Duration: [30 mins]
- Attendees: [All team members]

Meeting introduced everyone to the client and their business while Significantly improving our understanding of their data masking they desired along with our queries of what they wanted the CLI to look like and why they wanted one.

We presented to the client our initial plans to use Python & postgres, he gave us some useful suggestions for techniques and secondary software. Visualising our stricture to the client via the UML diagram shown in 5.7.

Key Achievements (sprint 1)

- Delivered a functional prototype meeting initial project requirements.
 - Built a CLI for database connectivity and masking execution.
 - Developed and tested masking logic to protect sensitive data.
 - Established a source and target database system for masking.
-

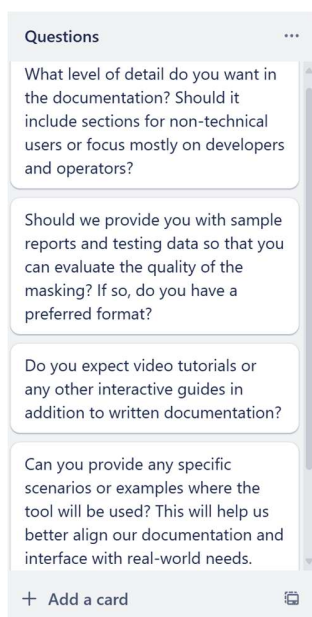
Challenges and Solutions

- **Challenge:** Understanding the various levels of sensitivity in data fields.
 - **Solution:** We collaborated as a team to define the masking rules for the prototypes.

- **Challenge:** Establishing communication between the source and the target databases.
 - **Solution:** Utilising PostgreSQL for its simplicity and versatility, making sure there is reliable data flow.
 - **Challenge:** Misunderstanding the client's requirements, due to the lack of tutor/mentor guidance leading to some incorrect assumptions about both the masking logic and data setup.
 - **Solution:** Following a critical meeting with the client, the team adjusted its approach to align with the re-clarified requirements, enabling successful completion of the prototype.
-

Feedback Request

Inviting the client to provide us with feedback on these following aspects that were pre-planned on Trello before the meeting:



What Went Well:

- **Collaboration:** All team members worked beyond assigned roles to ensure progress.
- **Communication:** Regular updates through stand-ups and Trello facilitated task alignment.
- **Prototype Completion:** The prototype was finalised ahead of schedule, setting a strong foundation for future sprints.

What Could Have Been Better:

- **Understanding Requirements:** Initial gaps in requirements caused minor delays.
 - **Task Prioritisation:** Refining task priorities earlier could have improved focus.
-

4. Sprint 2: Database

4.1. Product Backlog

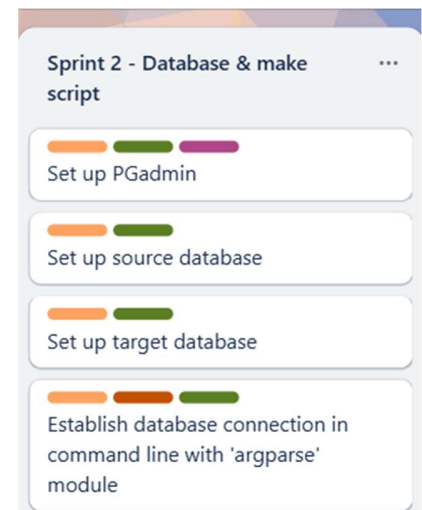
The Product Backlog for Sprint 2 built upon the prototype from Sprint 1, focusing on implementing database functionality and ensuring system reliability:

1. **Database Setup:**
 - Install and configure PostgreSQL (external DB software) & PG Admin 4 client for easy database management.
 - Create a source database, then populating this with sensitive mock data, In place of the clients databases.
 - Set up a target database (also in PGAdmin) to store masked data.
 2. **Database Connectivity:**
 - Establish command-line database connections to the PostGre using the *argparse* module.
 3. **Script Integration:**
 - Partially refactor the masking script to incorporate database connectivity with PostGre and enable data transfer.
 4. **Testing and Validation:**
 - ensure CLI output lines up with that of the target database.
 - Ensure no unexpected anomalies have been sent to target database.
 - Verify database connectivity and ensure the Masking Script still works reliably, as it did in Sprint 1.
-

4.2. Sprint Backlog

Key tasks for Sprint 2:

1. **Database Setup:**
 - Configure PGAdmin and set up source and target databases.
2. **Database Connectivity:**
 - Integrate the *argparse* module for command-line database connections.
3. **Script Integration:**
 - Update the masking script to handle data flow between the databases.
4. **Testing:**
 - Validate database integration and data masking accuracy.



4.3. Work time estimates: The team estimated this sprint would take two days:

- **Database Setup:** 1 day for Alex and Trinity to demonstrate PGAdmin installation and configuration.
- **Database Connectivity:** 1 day for Berk and Waleed with Salma and Ala's assistance to integrate and test connections.
- **Testing:** Collaborative effort to ensure the masking script masked and stored data correctly.

4.4. Daily stand up meetings:

December 8	Sprint 2 (Week 2)	Set up PGAdmin for database management	Source and target databases were configured and tested successfully.
December 9	Sprint 2 (Week 2)	Established database connections	Added CLI functionality for connecting to databases. Resolved minor connection issues.
December 10	Sprint 2 (Week 2)	Generated own data for testing	The database setup was finalised and working well.

Daily stand-ups facilitated alignment and problem-solving:

- **December 8 (Monday):** Alex and Trinity led PGAdmin setup, ensuring all members could create and configure databases.
- **December 9 (Tuesday):** Berk and Waleed integrated database connectivity into the script, with Alex addressing connection issues.
- **December 11 (Thursday):** The team reviewed database integration progress, resolved challenges, and prepared for the next sprint phase.

4.5. Customer demonstration meeting:

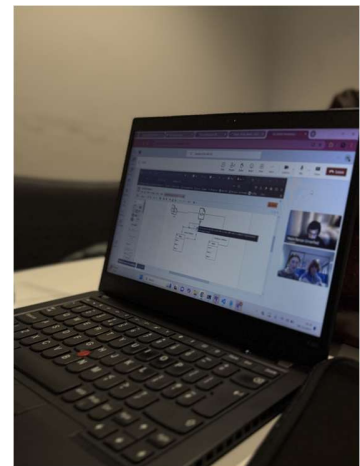
Sprint 2 primarily focused on internal development, testing, and refining the tool's database connectivity and mock data integration functionalities.

Customer Demonstration Meeting Summary - Sprint 1

1. Meeting Overview

- Date: [December 6th, 2024]
- Duration: [30 mins]
- Attendees: [All team members]

The main purpose of this meeting was to showcase the progress made during Sprint 1. The team demonstrated the initial prototype of the data masking tools and discussed its alignment with the client's requirements.



The Later progress made in Sprint 2 was then later incorporated into the Sprint 3 customer demonstration, where the full functionality of the tool was showcased to the client himself.

What Went Well:

- **Successful Database Setup:** Alex and Berk ensured all members could configure source and target databases.
- **Github Collaboration**
- **Command-Line Integration:** Waleed and Berk implemented database connections using *argparse*.
- **Collaboration:** Stand-ups and collaborative problem-solving kept the sprint on track.

What Could Have Been Better:

- **Initial Setup Challenges:** PGAdmin setup took longer than expected for some members.
 - **Testing Time:** More time for rigorous testing and debugging would have ensured greater feature validation.
 - **Client Input:** A customer meeting mid-sprint could have provided valuable feedback on database integration.
-

5. Sprint 3: Developing Final Script

5.1. Product Backlog

The Product Backlog for Sprint 3 represented the culmination of client feedback and focused on delivering the final product:

1. **JSON-Based Masking Templates:**
 - Develop flexible templates for custom masking rules with multiple sensitivity levels (e.g., partial masking, full redaction).
 2. **Advanced Masking Logic:**
 - Refine techniques to dynamically apply masking rules to complex fields (e.g., emails, credit cards).
 3. **Command-Line Interface Enhancements:**
 - Add template selection commands, configure database paths, and implement error handling.
 4. **Testing and Validation:**
 - Conduct rigorous testing to ensure functionality.
 5. **Client Feedback Integration:**
 - Incorporate feedback from the demonstration meeting to finalise features.
-

5.2. Sprint Backlog

The **Sprint Backlog** for Sprint 3 included tasks derived from the Product Backlog, focusing on completing high-priority deliverables.

1. **JSON Template Development:**
 - Ala and Salma worked on creating and testing JSON templates for masking rules.
 - Ensured templates were flexible and supported multiple security levels (e.g., high, medium, low sensitivity).
2. **Masking Logic Implementation:**
 - Alex and Waleed collaborated to refine the masking logic, integrating JSON templates into the script.
 - Ensured the logic dynamically applied masking rules to various field types.
3. **Command-Line Interface (CLI) Enhancements:**
 - Waleed added new commands to the CLI for selecting JSON templates and configuring database paths.
4. **Testing and Debugging:**

- Berk and Waleed tested the database integration to validate data flow from source to target.
 - Ala and Salma verified the accuracy of masked data and reviewed logs for potential issues.
5. **Finalisation and Preparation:**
- Conducted a code freeze to lock features and ensure stability.
 - Prepared for the final demonstration meeting by reviewing all deliverables against the user stories.
-

5.3. Work item estimates: The team allocated five days to complete Sprint 3:

1. JSON Template Development: 2 days for Ala and Salma to create and validate templates.
 2. Masking Logic Integration: 2 days for Alex and Waleed to integrate templates and refine rules.
 3. CLI Enhancements: 1 day for Waleed to implement new commands and error handling.
 4. Testing and Debugging: Collaborative effort across 1 day to ensure functionality.
-

5.4. Daily stand up meetings:

Daily stand-ups ensured alignment and addressed challenges:

- **December 11 (Monday):** Integration of masking logic and JSON templates began, guided by client feedback shared by Luke.
 - **December 12 (Tuesday):** Focused on refining templates and ensuring flexibility for masking rules.
 - **December 13 (Wednesday):** Resolved database connection issues and validated test data.
 - **December 14 (Thursday):** Finalised masking logic and ensured database stability.
 - **December 15 (Friday):** Completed code freeze and reflected on sprint progress in preparation for the demo day.
-

5.5. Customer Demonstration Meeting – Sprint 3

1. **Meeting Overview**
 - **Date:** December 11, 2024
 - **Duration:** [30 minutes]
 - **Attendees:** [Luke, Trinity]

This meeting marked a significant milestone in the project, showcasing the progress made during Sprints 2 and 3. The team saw a full demonstration of the project running via screen share on MS teams showcasing : integration of database connectivity, mock data generation, and updates to the command-line interface (CLI), as well as the initial implementation of JSON-based masking templates.

2. Objectives of the Meeting

- Present the cumulative progress from Sprints 2 and 3 to the client.
 - Highlight key features added, including database integration and enhanced masking logic.
 - Confirm alignment with the client's expectations for the final deliverable.
-

3. Key Features Demonstrated

The team demonstrated:

1. **Database Connectivity:** Transfer of data between source and target databases.
 2. **Masked Data Application:** Successfully applied masking rules dynamically from JSON templates.
 3. **CLI Updates:** Enhanced commands for template selection and logging.
-

4. Client Feedback

The client expressed satisfaction with the team's progress and praised the enhancements made during Sprints 2 and 3. Specific feedback included:

- "The database integration and CLI updates are impressive and practical for real-world applications."
 - "The use of JSON templates adds flexibility to the masking process, making the tool adaptable to various use cases."
 - "It's clear that the team is organised and progressing steadily toward the project goals."
-

5. Next Steps (Final Sprint Objectives)

- Finalise JSON-based masking templates and enhance masking logic for complex data fields.
- Conduct comprehensive end-to-end testing to ensure stability and usability.
- Create detailed user documentation, including setup and usage guides.
- Prepare for the final demonstration, addressing any client feedback from this meeting.

5.6. Work item selection:

Sprint 3 required extensive collaboration:

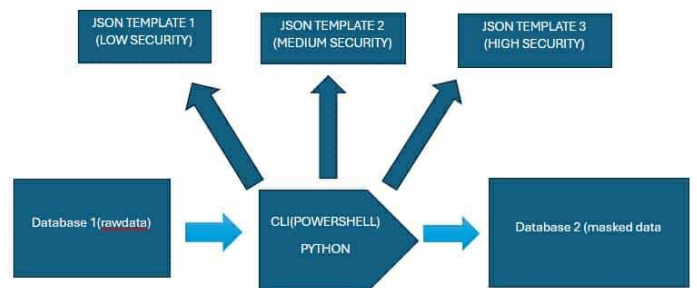
- **Database Setup:** Berk, with support from Trinity and Luke, helped set up PGAdmin for team members unfamiliar with the tool.
- **Command-Line Functionality:** Waleed and Berk worked together to establish a database connection using the *argparse* module.
- **Masking Logic Refinement:** Alex and Luke led research on masking levels (e.g., redaction for credit cards, partial masking for surnames) and integrated logic into the script with Waleed.
- **JSON Templates:** Salma and Ala developed and validated templates for flexible masking rules.
- **Issue Resolution:** Trinity provided support for debugging, resolving errors like infinite loops in the masking logic.

5.7. Requirements analysis and system design:

Database Design & Overall System Architecture:

Our database architect developed a drawing on day 1. It is a basic representation of the structure of the tool; the user passes in the needed information to the command line, the script will mask the data based on which JSON template security level they chose, and the masked data is stored in the target

database in its new form. The command line must take the script name, source database



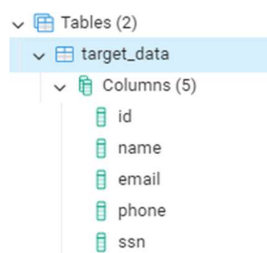
connection info, the target connection info, and the JSON file. This feature was added by Waleed and Berk, with the assistance of Salma and Ala.

The development process began by communicating with our client to grasp an understanding of the task. Product owner and scrum master had three meetings with the client. During the first meeting, the client communicated the vision and goals of the product. As product owner, he devised several user stories addressing the basic responsibilities of the tool. The team settled for PGAdmin4 to set up a source and target database.

```
PS C:\Users\trini> python "C:\Users\trini\Desktop\Projects\Agile Groupwork\IRIS-DMT\script.py" localhost postgres postgres 123 starterdata.source_table localhost target postgres 123 mask eddata.target_data "C:\Users\trini\Desktop\Projects\Agile Groupwork\IRIS-DMT\strict.json"
>> []
```

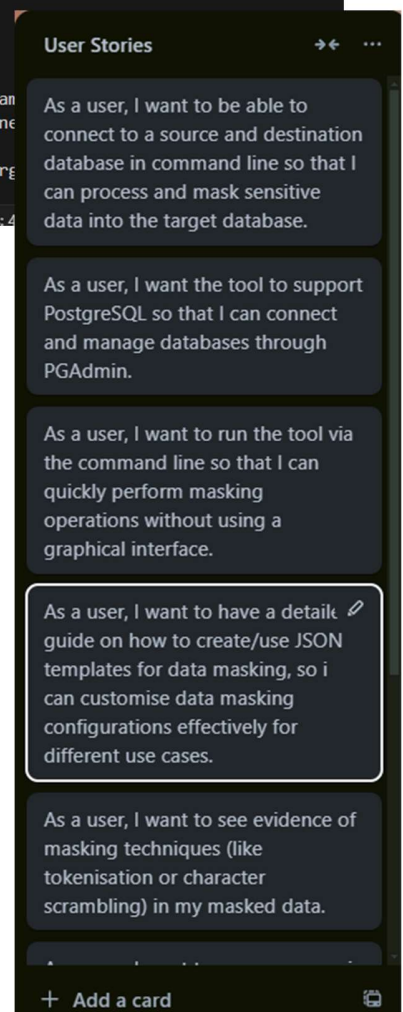
User stories:

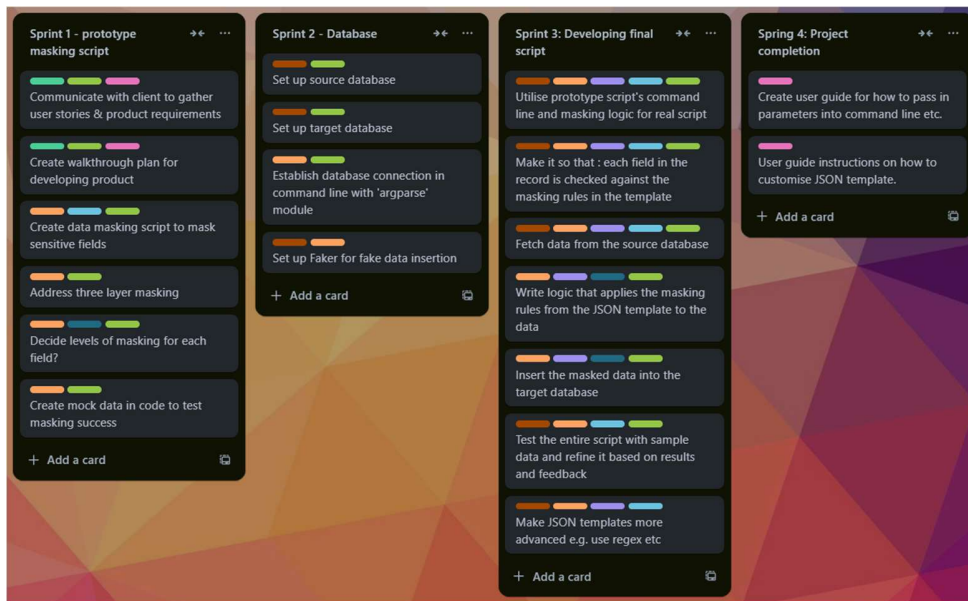
User stories are displayed in the screenshot to the right.



```
.taylor****@example.com', 'phone': '555-90(###) ###-', 'ssn': '****4567'}
Processing column: id
Processing column: name
Processing column: email
Processing column: phone
Processing column: ssn
Masked record: {'id': 10, 'name': 'taylor****@example.com', 'phone': '555-90(###) ###-', 'ssn': '****4567'}
Inserted 10 rows into the target database
PS C:\Users\trini>
```

The scrum master began to prepare the development team for creating the product by setting up GitHub for each team member for code sharing & pushing. Scrum master also prepared the team by developing a plan and trello board, and translated product owner's user stories into it for everyone to keep up with, and with the assistance of Salma and Ala, developed several sprints for the team to follow to meet the project's requirements. The board contains colour coded labels to mark who is assigned to which task. Some examples of sprints that refer directly to user stories include, 'As a user, I want to be able to connect to a source and destination database in command line so that I can process and mask sensitive data into the target database' which as a result, has created tasks 'Establish database connection in command line with 'argparse' module' and 'Write logic that applies the masking rules from the JSON template to the data'.





6. Details of the Solution Testing

Our code was designed to check over the masking type specified in the JSON file and mask the data accordingly.

```
# Data Masking Function
def mask_data(data, column_names, masking_rules):
    print(f"Applying masking rules from template: {masking_rules}")
    masked_data = []

    # Iterate over each record in the fetched data
    for record in data:
        masked_record = {}

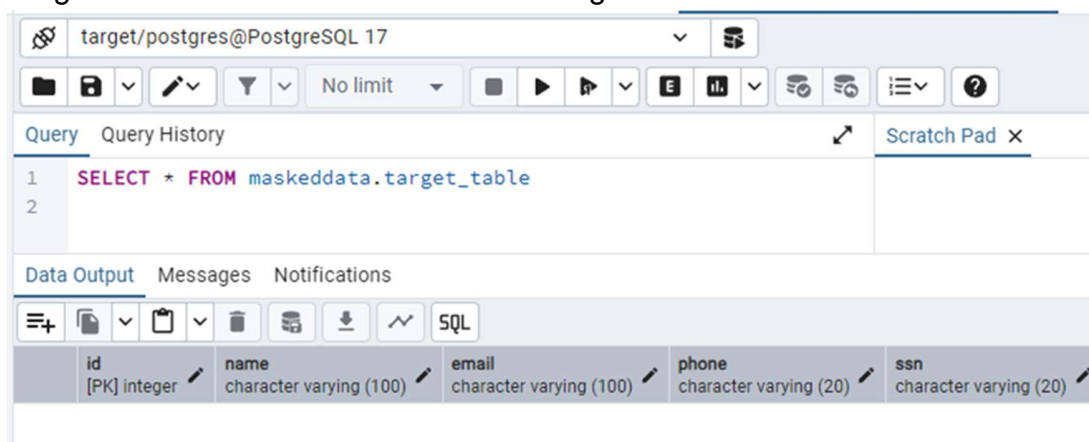
        # Iterate over columns
        for idx, column_name in enumerate(column_names):
            print(f"Processing column: {column_name}") # Add this line to log column names
            if column_name in masking_rules:
                rule = masking_rules[column_name]

                # Apply the masking based on the rule type
                if rule["maskingType"] == "redact":
                    masked_record[column_name] = rule["replacement"]
                elif rule["maskingType"] == "partial":
                    replacement = rule.get("replacement", "")
                    if column_name == "email":
                        # email masking logic
                        at_index = record[idx].find('@')
                        if at_index != -1:
                            masked_record[column_name] = record[idx][:at_index] + '*****' + record[idx][at_index:]
                        else:
                            masked_record[column_name] = '*****'
                    elif column_name == "ssn":
                        # Mask SSN (e.g., last 4 digits remain visible)
                        masked_record[column_name] = rule["replacement"] + record[idx][-4:]
                    else:
                        # Mask other columns partially
                        masked_record[column_name] = record[idx][:len(record[idx])//2] + rule["replacement"]
                elif rule["maskingType"] == "last_4":
                    # Show only the last 4 digits (e.g., for SSN)
                    masked_record[column_name] = rule["replacement"] + record[idx][-4:]
                else:
                    pass

    return masked_data
```

The script was tested by using PGAdmin to show whether the database was successfully updating. If successful, the masked data at each level displays in the target database.

Target table before medium JSON masking:



The screenshot shows the PGAdmin interface for a PostgreSQL database. The query editor displays the following SQL query:

```
1 SELECT * FROM maskeddata.target_table
2
```

The 'Data Output' tab is active, showing the results of the query in a table format. The table has five columns: id, name, email, phone, and ssn. The data is as follows:

id	name	email	phone	ssn
[PK] integer	character varying (100)	character varying (100)	character varying (20)	character varying (20)

User input in command line:

```
PS C:\Users\trini> python "C:\Users\trini\Desktop\Projects\Agile Groupwork\IRIS-DMT\script.py" localhost postgres postgres 123 starterdata.source_table localhost target postgres 123 maskeddata.target_table "C:\Users\trini\Desktop\Projects\Agile Groupwork\IRIS-DMT\medium.json"
```

Output in terminal:

```
Masked record: {'id': 6, 'name': 'Sarah ****', 'email': 'sarah.wilson****@example.com', 'phone': '555-67(###) ###-', 'ssn': '****1234'}
Processing column: id
Processing column: name
Processing column: email
Processing column: phone
Processing column: ssn
Masked record: {'id': 7, 'name': 'David M****', 'email': 'david.martinez****@example.com', 'phone': '555-78(###) ###-', 'ssn': '****2345'}
Processing column: id
Processing column: name
Processing column: email
Processing column: phone
Processing column: ssn
Masked record: {'id': 8, 'name': 'Laura A****', 'email': 'laura.anderson****@example.com', 'phone': '555-89(###) ###-', 'ssn': '****3456'}
Processing column: id
Processing column: name
Processing column: email
Processing column: phone
Processing column: ssn
Masked record: {'id': 9, 'name': 'James ****', 'email': 'james.taylor****@example.com', 'phone': '555-90(###) ###-', 'ssn': '****4567'}
Processing column: id
Processing column: name
Processing column: email
Processing column: phone
Processing column: ssn
Masked record: {'id': 10, 'name': 'Mary ****', 'email': 'mary.moore****@example.com', 'phone': '555-01(###) ###-', 'ssn': '****5678'}
Inserted 10 rows into the target database.
PS C:\Users\trini>
```

Results after running the script, the source and target database, and called 'medium' JSON

Query Query History Scratch Pad x

```
1 SELECT * FROM maskeddata.target_table
2
```

Data Output Messages Notifications

Showing rows: 1 to 10 Page No: 1 of 1

	d PK] Integer	name character varying (100)	email character varying (100)	phone character varying (20)	ssn character varying (10)
1	1	John****	john.doe****@example.com	555-12(###) ###-	****6789
2	2	Jane ****	jane.smith****@example.com	555-23(###) ###-	****7890
3	3	Robert ****	robert.johnson****@example.com	555-34(###) ###-	****8901
4	4	Emily****	emily.davis****@example.com	555-45(###) ###-	****9012
5	5	Michael****	michael.brown****@example.com	555-56(###) ###-	****0123
6	6	Sarah ****	sarah.wilson****@example.com	555-67(###) ###-	****1234
7	7	David M****	david.martinez****@example.com	555-78(###) ###-	****2345
8	8	Laura A****	laura.anderson****@example.com	555-89(###) ###-	****3456
9	9	James ****	james.taylor****@example.com	555-90(###) ###-	****4567
10	10	Mary ****	mary.moore****@example.com	555-01(###) ###-	****5678

The masked data successfully appeared in the target database indicating that the database changes were effective. Here is another successful demonstration of testing with PGAdmin for our 'strict.json' high security redacted option:

1 **SELECT** * **FROM** maskeddata.target_table

Data Output Messages Notifications

Showing rows: 1 to 10 Page No: 1 of 1

	id [PK] integer	name character varying (100)	email character varying (100)	phone character varying (20)	ssn character varying (20)
1	1	REDACTED	REDACTED@domain.com	****4567	****6789
2	2	REDACTED	REDACTED@domain.com	****5678	****7890
3	3	REDACTED	REDACTED@domain.com	****6789	****8901
4	4	REDACTED	REDACTED@domain.com	****7890	****9012
5	5	REDACTED	REDACTED@domain.com	****8901	****0123
6	6	REDACTED	REDACTED@domain.com	****9012	****1234
7	7	REDACTED	REDACTED@domain.com	****0123	****2345
8	8	REDACTED	REDACTED@domain.com	****1234	****3456
9	9	REDACTED	REDACTED@domain.com	****2345	****4567
10	10	REDACTED	REDACTED@domain.com	****3456	****5678

Basic level JSON file test result:

Data Output Messages Notifications

Showing rows: 1 to 10 Page No: 1 of 1

	id [PK] integer	name character varying (100)	email character varying (100)	phone character varying (20)	ssn character varying (20)
1	1	John****	john.doe****@example.com	****4567	****6789
2	2	Jane ****	jane.smith****@example.com	****5678	****7890
3	3	Robert ****	robert.johnson****@example.com	****6789	****8901
4	4	Emily****	emily.davis****@example.com	****7890	****9012
5	5	Michae****	michael.brown****@example.com	****8901	****0123
6	6	Sarah ****	sarah.wilson****@example.com	****9012	****1234
7	7	David M****	david.martinez****@example.com	****0123	****2345
8	8	Laura A****	laura.anderson****@example.com	****1234	****3456
9	9	James ****	james.taylor****@example.com	****2345	****4567
10	10	Mary ****	mary.moore****@example.com	****3456	****5678

Unit test:

This unit test below was created to test the masking logic & ability before our final script. This test passed.

```

s > trini > Desktop > Projects > Agile Groupwork > IRIS-DMT > unittest.py > ...
import unittest

class TestMaskingFunctions(unittest.TestCase):

    def test_mask_data(self):
        # Sample data and column names
        data = [
            ("john.doe@example.com", "123-45-6789"),
            ("alice.smith@example.com", "987-65-4321")
        ]
        column_names = ["email", "ssn"]
        masking_rules = {
            "email": {"maskingType": "partial", "replacement": "****"},
            "ssn": {"maskingType": "last_4", "replacement": "XXX-XX-"}
        }

        # Expected output after masking
        expected_output = [
            {"email": "john.doe@****.com", "ssn": "XXX-XX-6789"},
            {"email": "alice.smith@****.com", "ssn": "XXX-XX-4321"}
        ]

        # Apply the masking function
        masked_data = mask_data(data, column_names, masking_rules)

        # Check if the output matches the expected result
        self.assertEqual(masked_data, expected_output)

if __name__ == "__main__":
    unittest.main()

```

Conclusions, Summary and Evaluation of the Achieved Results

Conclusions, Summary, and Evaluation of Achieved Results (Sprints 1-3)

Sprint 1 focused on the primary goal of creating a basic prototype. Overall, Sprint 1 was the importance of having a working prototype which allowed us to identify potential roadblocks early in the project.

Sprint 2 was dedicated to setting up the database infrastructure and ensuring database connectivity with the data masking tool. Overall, Sprint 2 was for ensuring the system's foundational database setup was complete and functional. The team's effective collaboration ensured that all members were equipped to contribute. The sprint successfully delivered a reliable database infrastructure for the tool.

Sprint 3 was focused on refining the tool's core features and ensuring it met the client's requirements. Overall, sprint 3 was an outstanding success, culminating in a fully functional product that met the client's core requirements. The team's collaborative approach, clear

division of tasks, and continuous testing ensured a successful delivery. The feedback received from confirmed that the tool was well-received, and it aligned with the client's needs.

Conclusion

The team exhibited effective communication, task sharing, and problem-solving, allowing them to successfully meet project goals and deliver a well-functioning tool within the expected timeframe. The results of the sprints demonstrate that the tool meets the client's expectations and will provide a reliable solution for sensitive data masking.

The team effectively collaborated to deliver a functional prototype, establish database integration, and implement advanced masking logic. ~~Despite initial challenges, such as understanding client requirements and technical setup, these were addressed through regular communication, iterative refinement, and problem-solving.~~ While the prototype was minimal, it set the stage for the more complex tasks ahead.

The final product meets client expectations by providing a reliable and adaptable solution for sensitive data masking. Key achievements include implementing JSON-based masking templates, simple command-line functionality with error checking, robust data masking logic. Overall, the project highlights the importance of teamwork, agile practices, and continuous feedback in delivering a high-quality software solution.

References

Satori. (n.d.). *Data Masking: 8 Techniques and How to Implement Them Successfully*.

[online] Available at: [Data Masking: 8 Techniques and How to Implement Them Successfully - Satori](#)

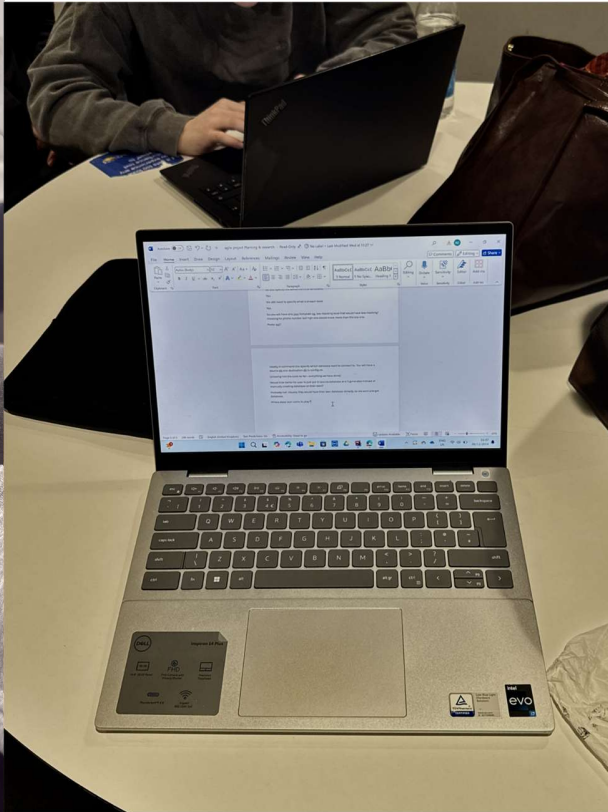
SQLBits. (2021, May 11). *Dynamic Data Masking in 20 Minutes*. YouTube.

<https://youtu.be/qEAVYtxtS2o?si=dACVQ2L1QQY1vSAM>

Imperva. (2024). *What is Data Masking? | Techniques & Best Practices* |

Imperva. Learning Center. <https://www.imperva.com/learn/data-security/data-masking/>

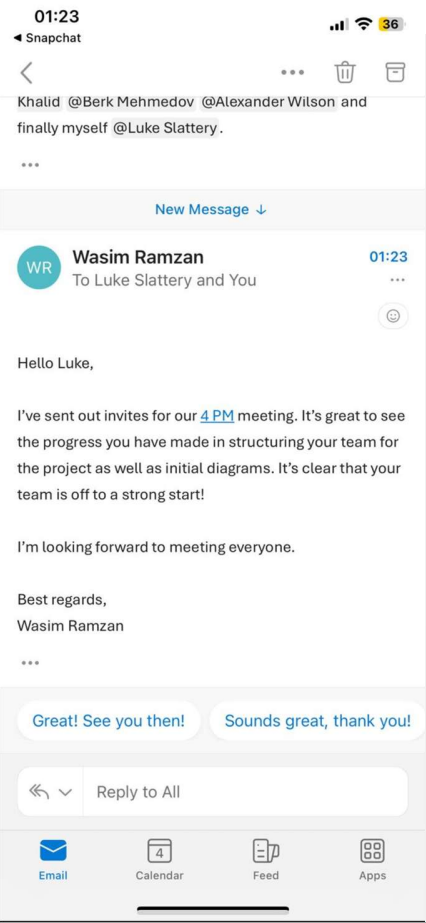
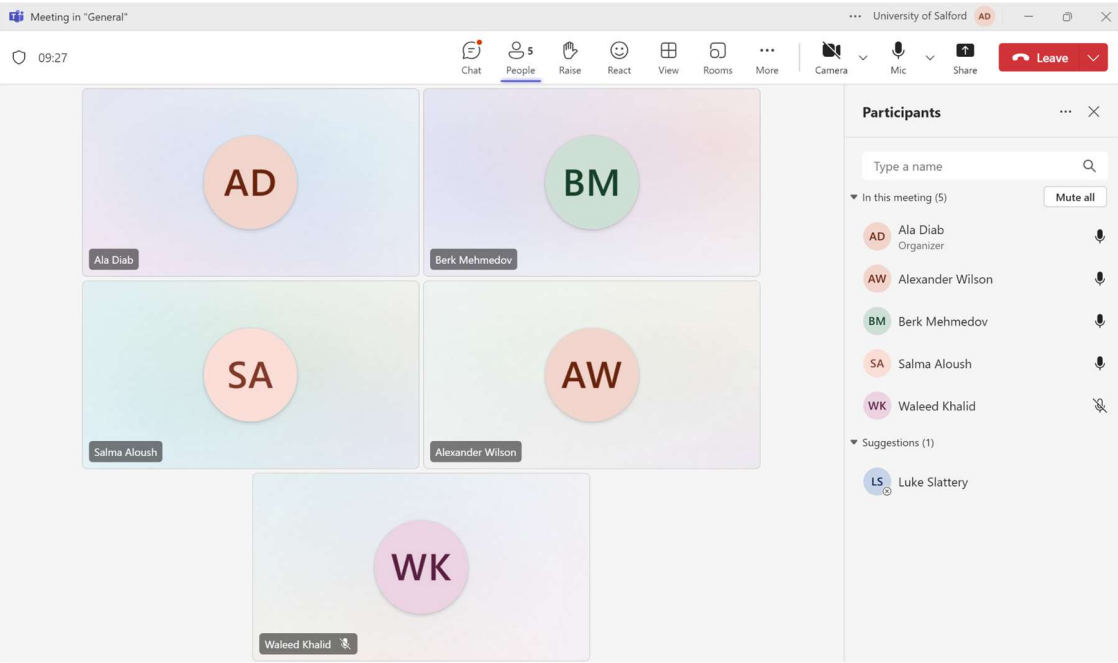
Appendices



Meeting in iris squad University of Salford AD

02:47 Chat People Raise React View Apps More Camera Mic Share Leave

AD Ala Diab	TB Trinity Booth
AW Alexander Wilson	SA Salma Aloush
BM Berk Mehmedov	WK Waleed Khalid



Data Masking Project: Sprint 2 Feedback

Date: 11/12/2024

Customer Name: Wasim Ramzan / IRIS Software Group

Group: 15

1. Overview of Feedback

Great progress since the first sprint and presented a working solution. Currently working on enhancements and later the documentation.

2. Strengths Observed

- A working solution was presented using redaction techniques.
- The JSON template approach shown met my requirements.

3. Areas for Improvement

- You are already aware of this but work on introducing other masking techniques.
- Hopefully making notes on the process so that later on the documentation and help guides can be done quickly.
- Outputs on the CLI after running the masking process. For e.g. x amount of rows were masked and y amount of rows were not masked etc. Also, anything else that might be useful.

4. Questions/Concerns

- Concern:** Testing to see if other databases are supported. Include supported databases in the documentation so we are aware of what can be used.

5. Closing Remarks

Not much else to add as you have mainly the enhancements to work on and the bonus tasks. As mentioned, amazing progress since the first sprint/demo!



Data Masking Project: Sprint 1 Feedback

Date: 06/12/2024

Customer Name: Wasim Ramzan / IRIS Software Group

Group: 15

1. Overview of Feedback

The progress in Sprint 1 was encouraging, particularly in partially achieving Phase 1 and beginning the JSON templates work. You have researched the various masking techniques given my requirements. There are some further recommendations I have provided.

2. Strengths Observed

- You are organised and have a strong sprint plan.
- Asking questions to clarify my requirements. As well as providing suggestions for different approaches.

3. Areas for Improvement

- Research best practices on what data masking techniques to use for certain fields.
- Next meeting hopefully a working demo can be presented.
- Start small by having a JSON template operational allowing for a more dynamic approach where we can configure different masking levels and add fields.
 - After this you can give the user an option to configure missing columns via CLI - if there are columns present in the database but not specified in the JSON template (a great suggestion but optional).

4. Questions/Concerns

- **Question:** Would you be able to deliver the bonus work based on your sprint plan?
- **Concern:** Ensure you research masking techniques with varying levels and apply the JSON template configurations. Any confusion then I can further clarify the requirements.

5. Closing Remarks

Good progress for the first sprint! Some confusion around the purpose of the JSON templates but hopefully this has now been rectified. I look forward to seeing how the team addresses the identified improvement areas and builds on this foundation in the next sprint. Let me know if additional resources or clarification from my side can help streamline the process.

```
'email': 'REDACTED@domain.com', 'phone': '****2345', 'ssn': '****4567'}
Processing column: id
Processing column: name
Processing column: python "C:\Users\trini\Desktop\Projects\Agile Groupwork\IRIS-DM\script.py" localhost postgres postgres 123 starterdata.source_table
localhost target postgres 123 maskeddata.target_data "C:\Users\trini\Desktop\Projects\Agile Groupwork\IRIS-DM\strict.json"ted 10 rows into the target d
atabase.
>> I:\Users\trini>
```

