SOCIETE GENERALE HACKATHON 2024

AUTOMATING VBA MACRO DOCUMENTATION AND TRANSFORMATION

Name : Samyuktha V (20PD24)
Course : M.Sc. Data Science (5-Year Integrated)
Department : Applied Mathematics and Computational Sciences

Objective: The goal is to develop a solution that automates the documentation and understanding of legacy VBA macros used in critical processes across DFIN, facilitating their transformation into modern technologies and IT platforms.

Why?

VBA (Visual Basic for Applications) macros have been widely used in various organizations for automating tasks within Microsoft Office applications. However, as technologies advance, these legacy macros pose several challenges:

- Maintenance Difficulty: Legacy VBA code can be hard to maintain, especially when the original developers are no longer available.
- Performance Issues: Older macros might not be optimized, leading to inefficiencies.
- Modernization Needs: Organizations are moving towards modern, efficient technologies and platforms, necessitating the transformation of legacy VBA macros.
- Documentation Gaps: Often, these macros lack comprehensive documentation, making it difficult for developers to understand and modify them.

What?

The solution aims to automate the documentation and understanding of VBA macros, providing detailed descriptions, summaries, and visualizations of the code. It also identifies inefficiencies and optimization opportunities, making it easier to transform these macros into modern technologies.

Solution Overview

The solution involves developing a platform using various technologies to automate the analysis, documentation, and visualization of VBA macros. The platform consists of the following components:

- Macro Analysis Tool: A script that parses and analyzes VBA macro code.
- Technology Integration: Using GenAI (GPT-2 and Claude API) and Machine Learning for embedding and text generation.
- User Interface: A web-based interface built using Streamlit for easy input and output visualization.
- Performance and Bottleneck Analysis: Functions to analyze the efficiency of the macro code.

Tools and Technologies Used

Retrieval Augmentation Generation:

The RAG mechanism, short for Retrieval, Augmentation, and Generation, is integral to automating and optimizing VBA macro analysis:

- Retrieval: Involves fetching VBA macro data from diverse sources such as websites (e.g., ExcelChamps), APIs, or databases using tools like BeautifulSoup and requests.
- Augmentation: Enhances retrieved data by refining, normalizing, and enriching it with additional context or metadata, ensuring higher quality and relevance for downstream processes.
- Generation: Employs advanced machine learning models like GPT-2 through APIs (e.g., Claude API) to automatically generate comprehensive descriptions and visual representations (using Graphviz) of macro functionalities.

GPT-2

Purpose:

GPT-2 is a pre-trained language model used for generating embeddings of VBA code.

Functionality:

- Tokenizes the VBA macro code and generates hidden state vectors.

- The mean of these hidden state vectors is used as the embedding for the macro code.
- These embeddings are essential for comparing and finding similar code snippets using FAISS.

FAISS

Purpose:

FAISS (Facebook AI Similarity Search) is a library used for efficient similarity search and clustering of dense vectors (embeddings).

Functionality:

- Indexes the embeddings of VBA macro code snippets.
- Performs fast and efficient nearest neighbor search to find similar code snippets.
- Helps in retrieving context examples for generating detailed descriptions of the macro code using the Claude API.

Claude API

Purpose:

Claude API (from Anthropic) is used for generating detailed descriptions and summaries of the VBA code.

Functionality:

- Takes a prompt (e.g., a VBA macro code) and generates human-like text responses.
- Generates summaries that concisely explain the steps of the VBA macro.
- Produces detailed descriptions that explain the functionality and purpose of the VBA macro, using context from similar code snippets.

Graphviz

Purpose:

Graphviz is used for visualizing the process flow of the macro.

Functionality:

- Generates visual representations (graphs) of the macro's logic flow.

- Creates nodes and edges based on the steps described in the generated summary.
- Helps users understand the flow of the macro in a visual format, making it easier to grasp complex logic.

Regex

Purpose:

Regular expressions (regex) are used for parsing VBA code to extract variables and loops.

Functionality:

- Identifies and extracts variable declarations (e.g., "Dim i As Integer") from the VBA code.
- Detects and counts loops (e.g., "For i = 1 To 100") within the macro.
- Helps in analyzing the structure and components of the VBA code for performance and complexity analysis.

Python Libraries

- numpy: Used for numerical operations, especially in handling embeddings and performing vector operations.
- torch: Used with GPT-2 for handling tensor operations and model inference.
- requests: Used for making HTTP requests, especially to interact with the Claude API.
- json: Used for parsing and handling JSON data, such as loading macro examples from a file.
- matplotlib: Used for plotting and visualizing data if needed.
- re: Provides support for regular expressions in Python, used for parsing VBA code.
- BeautifulSoup : Used for web-scraping in data collection.

Streamlit

Purpose:

Streamlit is used for building an interactive web-based user interface for the VBA macro analysis tool.

Functionality:

- Provides a user-friendly interface for users to input VBA macro code.
- Allows users to trigger various analysis functions (description generation, logic flow visualization, macro analysis) through buttons.
- Displays the results of the analyses, such as generated descriptions, visualized logic flows, and macro analysis results.

Data Collection:

Purpose: Extract VBA macro codes, titles, and descriptions from https://excelchamps.com/blog/useful-macro-codes-for-vba-newcomers/ and save in JSON format for analysis.

Process:

1. Import Libraries:
    - requests for fetching webpage content.
    - BeautifulSoup for parsing HTML.
    - json for saving data.
2. Fetch Webpage:
    - Use requests.get(url) to retrieve webpage content.
    - Parse HTML content with BeautifulSoup.
3. Initialize Storage:
    - Create an empty list macros for storing each macro's data.
4. Identify Macros:
    - Find all <h3> tags to locate macro titles.
5. Extract Data:
    - Extract title from each <h3> tag.
    - Collect description from <p> tags.
    - Gather code from <pre> tags.
6. Save to JSON:
    - Combine description parts into a single string.
    - Add title, description, and code to macros list.
    - Save macros list to vba_macros.json.
7. Confirmation:
    - Print confirmation message upon successful data saving.

Code Description:

Step 1: Load Data and Create FAISS Index

Step 2: Embed Code Function
Step 3: Generate Description and Summary
Step 4: Analyze VBA Macro
Step 5: Visualize Process Flow
Step 6: Streamlit UI

Flow Description:

1. User Input: The user inputs VBA macro code into the text area provided in the Streamlit interface.
2. Generate Description: Upon clicking the "Generate Description" button, the macro code is embedded using GPT-2, and a description is generated using the Claude API based on similar examples retrieved using FAISS.
3. Visualize Logic Flow: Clicking the "Visualize Logic Flow" button generates a summary of the macro code and visualizes the process flow using Graphviz.
4. Analyze Macro: Clicking the "Analyze Macro" button parses the VBA code to extract information such as lines of code, variables declared, loop counts, and cyclomatic complexity, and provides recommendations for improvement.

How?

The solution is implemented using the combination of tools and technologies to achieve the following:

1. Loading and Indexing Macro Data

- JSON File: Loads macro data (title, description, code) from a JSON file.
- FAISS Indexing: Embeds the macro code using GPT-2 and indexes these embeddings using FAISS for efficient similarity search.

2. User Interface with Streamlit

- Input: Users can input VBA macro code directly.
- Buttons: Users can trigger different functionalities (description generation, logic flow visualization, macro analysis) through buttons.
- Display: Results of the analysis are displayed on the interface.

3. Embedding and Description Generation

- GPT-2 Model: Tokenizes the VBA macro code and generates embeddings.

- Claude API: Uses these embeddings to generate detailed descriptions and concise summaries of the VBA code.

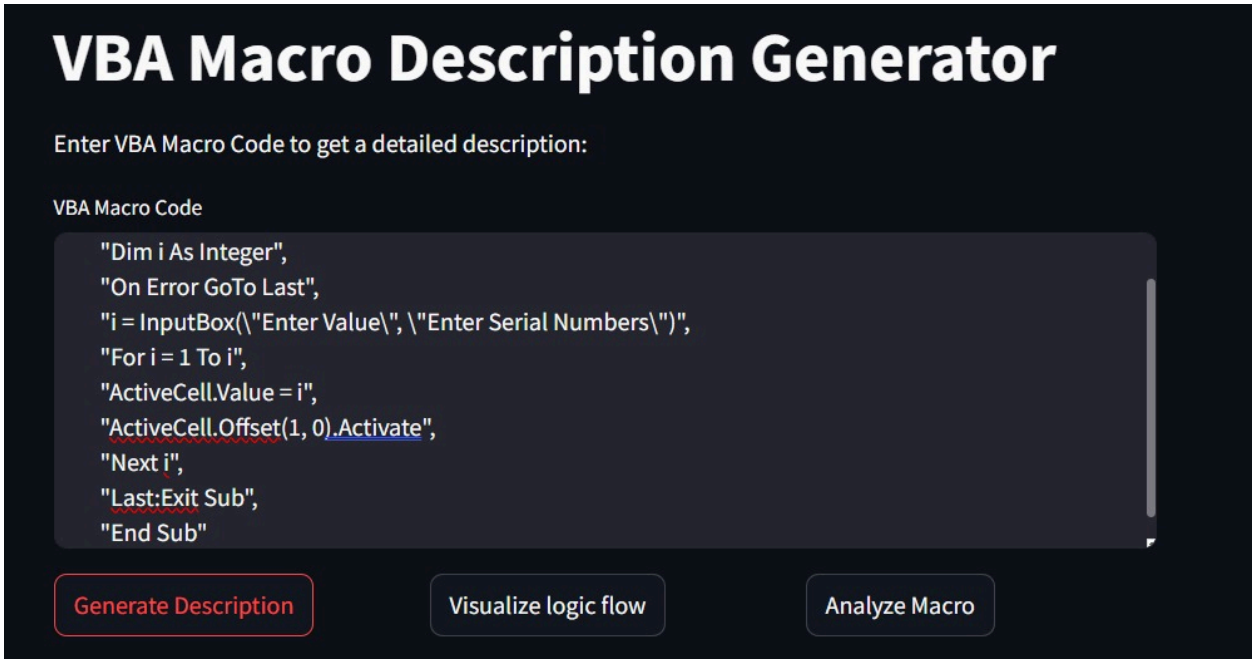
4. Process Flow Visualization

- Graphviz: Visualizes the process flow of the macro based on the generated summary, providing a clear and concise representation of the logic flow.

5. Macro Analysis

- Regex Parsing: Extracts variables and loops from the VBA code.
- Performance Metrics: Analyzes the code for lines of code, variables declared, loops count, and cyclomatic complexity.
- Bottleneck Identification: Identifies potential inefficiencies and provides recommendations for optimization.

Results:



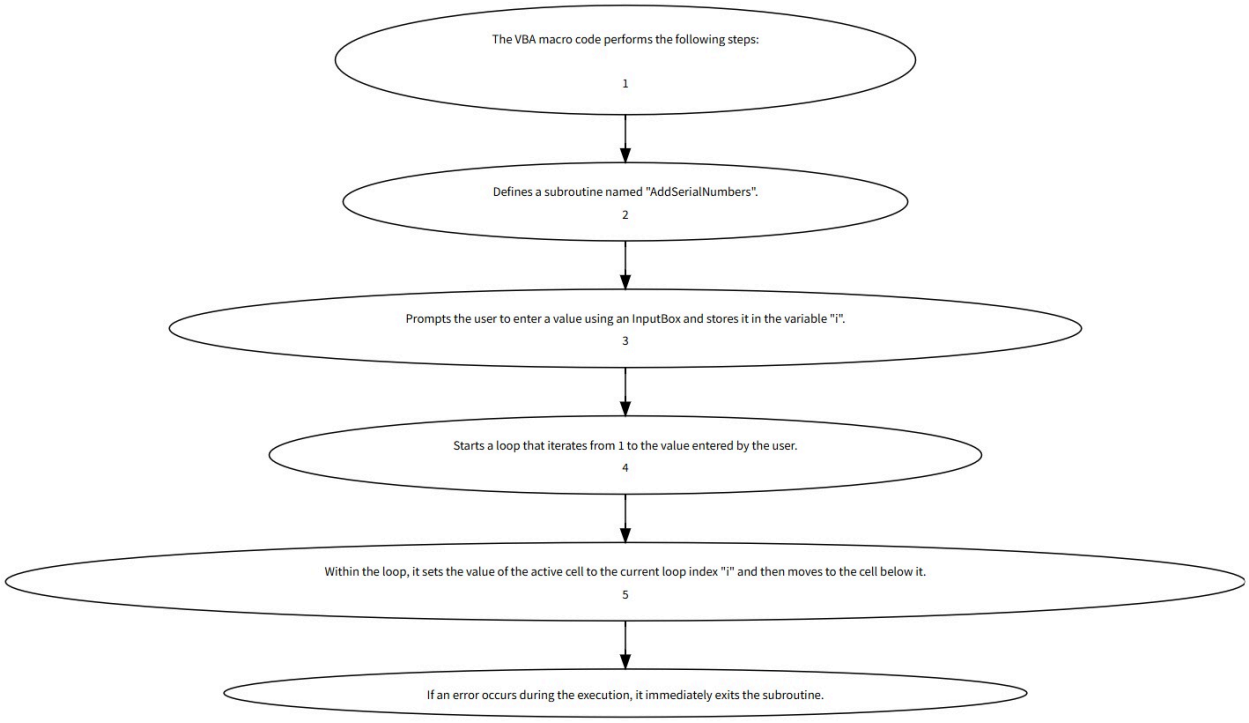
Generated Description

The provided macro code is a VBA subroutine named "AddSerialNumbers". Here's a detailed description of its functionality:

1. The macro starts with the declaration of a variable named "i" as an integer data type.
2. The "On Error GoTo Last" statement is used for error handling. If any error occurs during the execution of the macro, it will immediately jump to the line labeled "Last".
3. The macro displays an input box with the title "Enter Serial Numbers" and prompts the user to enter a value. The entered value is stored in the variable "i".
4. A "For" loop is initiated, which will run from 1 to the value entered by the user (stored in the variable "i").
5. Inside the loop, the following actions are performed:
    - The value of the loop counter "i" is assigned to the active cell (the currently selected cell in the worksheet).
    - The macro then moves the active cell one row down using the "Offset" method. The "Offset(1, 0)" means moving 1 row down and 0 columns to the right.
    - The "Activate" method is used to make the new cell (one row below) the active cell.
6. The loop continues until the value of "i" reaches the number entered by the user.
7. If any error occurs during the execution of the macro, it will jump to the line labeled "Last" using the "GoTo" statement.
8. The line labeled "Last" contains the "Exit Sub" statement, which will exit the subroutine and terminate the macro.
9. Finally, the macro ends with the "End Sub" statement.

In summary, this macro allows the user to enter a number, and it will populate the active cell and the cells below it with sequential numbers starting from 1 up to the entered number. For example, if the user enters 5, the macro will populate the active cell with 1, the cell below it with 2, and so on until 5.

The error handling part ensures that if any error occurs during the execution, the macro will exit gracefully without causing further issues.



VBA Macro Analysis Results:

- Lines of Code: 10
- Variables Declared: 1
- Loops Count: 1
- Cyclomatic Complexity: 2
- Recommendations: None

VBA Macro Description Generator

Enter VBA Macro Code to get a detailed description:

VBA Macro Code

```
Sub InefficientMacro()
    Dim i As Integer, j As Integer
    For i = 1 To 1000
        For j = 1 To 1000
            If i Mod 2 = 0 And j Mod 2 = 0 Then
                Cells(i, j).Value = i + j
            End If
        Next j
```

Generate Description | Visualize logic flow | Analyze Macro

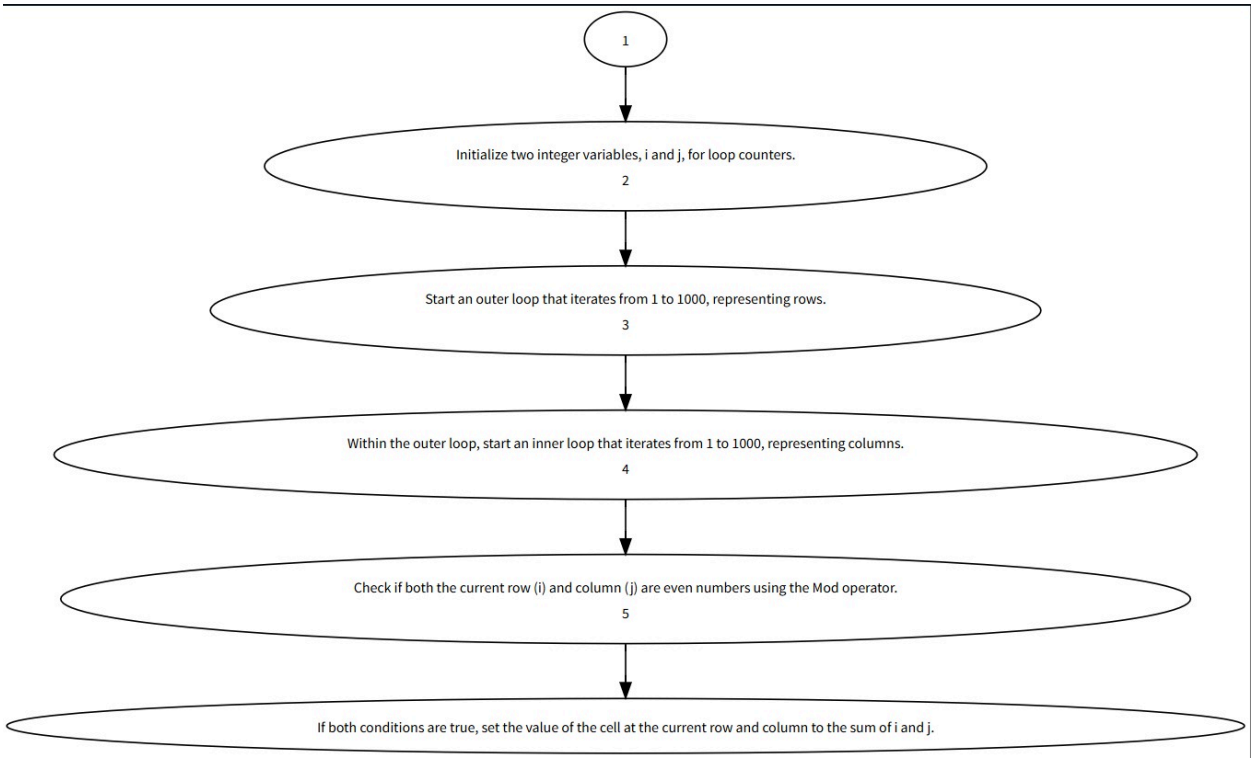
Generated Description

The provided macro code, named "InefficientMacro", performs the following actions:

1. It declares two integer variables, "i" and "j", which will be used as loop counters.
2. It starts an outer loop that iterates from 1 to 1000, controlled by the variable "i". This loop represents the rows in the worksheet.
3. Inside the outer loop, it starts an inner loop that also iterates from 1 to 1000, controlled by the variable "j". This loop represents the columns in the worksheet.
4. Within the inner loop, it checks two conditions using an If statement:
    - If the value of "i" is even (i.e., i Mod 2 = 0), and
    - If the value of "j" is even (i.e., j Mod 2 = 0)
5. If both conditions are true, it sets the value of the cell at the corresponding row (i) and column (j) to the sum of "i" and "j".
6. The inner loop continues to iterate through all the columns (j) for the current row (i).
7. After the inner loop finishes, the outer loop moves to the next row (i) and repeats the process.
8. The macro continues this nested loop iteration until it has processed all the cells in the range from (1, 1) to (1000, 1000).

It's important to note that this macro is labeled as "InefficientMacro" for a reason. It iterates through a large range of cells (1,000,000 cells in total) and performs a calculation only for cells where both the row and column indices are even. This means that the macro will be slow and resource-intensive, especially for large ranges.

A more efficient approach would be to use a step size of 2 in the loop counters to directly target the even rows and columns, reducing the number of iterations and skipping the unnecessary checks. Additionally, it's generally recommended to avoid excessive nested loops and to optimize the code for better performance.



VBA Macro Analysis Results:

- Lines of Code: 10
- Variables Declared: 1
- Loops Count: 2
- Cyclomatic Complexity: 3
- Recommendations:
- Review inefficient loops.