# **KONBRIEFING DOCUMENTATION:**

Introduction

The process of working with the initial script provided for the KonBriefing project posed several challenges. At the outset, the script was difficult to compile due to a number of dependencies that required installation. Specifically, libraries such as Pandas were missing, which prevented the code from being executed successfully. Once the necessary libraries were installed, we encountered additional issues with the script's execution, including coding errors and excessive output in the terminal.

The original code contained over 400 lines, which ultimately resulted in over 1000 lines of terminal output. The script was primarily intended to scrape data from three websites and generate a CSV output. However, instead of achieving this purpose efficiently, it generated a large amount of unnecessary HTML content, which was neither useful nor relevant for the task at hand.

Upon reviewing the code, it became clear that it was overly complex and contained repetitive operations for tasks such as pre-processing descriptions, dates, and country information. Furthermore, there was a lack of validation to ensure that the data extracted from the websites was clean and usable. This resulted in a need for multiple iterations of saving and reading CSV files.

To address these issues, we focused on simplifying and streamlining the script. By centralizing data cleaning tasks into dedicated functions, eliminating unnecessary HTML output, and optimizing the handling of CSV files, we were able to create a much more efficient and maintainable solution. The final script is now able to scrape the required data from the three websites, process it correctly, and produce a CSV output in a much cleaner and more effective manner.

**Conclusion:**

The original approach to scraping and organizing the data, while functional, presented several inefficiencies and complexities that made it difficult to maintain and use. After evaluating various solutions, it became clear that the best option was to **rebuild the script from scratch**. The new script is designed to efficiently extract and organize data from the three provided URLs and two local CSV files.

The process is as follows:

* **Extracting Data from Websites:** The script performs scraping on the three provided URLs, extracting key information for each cyber attack incident, including the date, title, description, and references.
* **Loading CSV Files:** It reads the two provided CSV files containing prior cyberattack data.
* **Organizing and Cleaning Data:** The script combines the data from both the CSV files and the scraped data, processing and formatting the dates. In cases where the date is unavailable, it assigns "No date specified."
* **Saving Data to a New CSV:** Finally, the script generates a new CSV file with all the data organized, ensuring the dates are correctly formatted or marked as "No date specified."

The result is a streamlined and efficient script that consolidates data from three web sources and two local files into a well-organized CSV, which can now be used for further analysis or reporting. This solution significantly simplifies the workflow and improves the overall quality of the data output.

Code Analysis

**Code Info**

- Logging Configuration:

Lines 4-6: Configures logging to record errors in error log.log. This is crucial for debugging and tracking issues. (CÓMO DOCUMENTAR ESTO, DADO QUE SON ERRORES MUY EXTENSOS):

Here’s a summary of the errors reported by the default Python script, organized into key points:

1. **Errors related to the number of columns:**
   * "Unexpected number of columns after splitting": When splitting the contents of the files kon\_briefing\_incidents\_2024-04-29.xlsx and kon\_briefing\_incidents\_2024-04-22.xlsx, 18 columns were found instead of the expected 4.
2. **Errors decoding files:**
   * "utf-8 codec can't decode byte": The files kon\_briefing\_incidents\_2024-04-29.xlsx and kon\_briefing\_incidents\_2024-04-22.xlsx contain characters that cannot be decoded using utf-8. This issue occurs specifically at position 22.
3. **Errors processing a CSV file:**
   * "No columns to parse from file": The file web\_scraped\_incidents.csv has no columns or is not a valid file.
4. **Errors related to the 'date' column:**
   * "The 'date' column is missing in the combined DataFrame": The date column is missing in the DataFrame after combining the data.
5. **Errors saving the combined file:**
   * "Permission denied": The file combined\_sorted\_incidents.csv cannot be saved due to permission issues in the specified path.
6. **General errors:**
   * "An error occurred: Permission denied": A permission issue prevents the combined file from being saved in the mentioned path.

- File Path Definitions:

Lines 9-10: Defines the paths to the Excel files. These paths are used throughout the script to locate the input files.

- File Existence Check:

Lines 12-17: check\_file\_exists(file\_path) function verifies if the specified file exists. Logs an error and raises FileNotFoundError if the file is missing.

- Reading Excel Files:

Lines 19-25: read\_excel\_file(file\_path) function reads an Excel file into a DataFrame. Logs an error and raises an exception if reading fails.

- Processing Files:

Lines 27-33: process\_file(file\_path) function reads and processes an Excel file, returning a DataFrame. Handles exceptions and logs errors.

- Combining and Sorting DataFrames:

Lines 35-43: combine\_and\_sort\_dataframes(dfs) function combines multiple DataFrames and sorts them by the date column. Logs an error if the date column is missing.

- Saving to CSV:

Lines 45-51: save\_to\_csv(df, output\_path) function saves a DataFrame to a CSV file. Logs an error and raises an exception if saving fails.

- Main Function:

Lines 53-66: main() function orchestrates the entire process: checking file existence, processing files, combining and sorting DataFrames, and saving the result to a CSV file. Logs any errors that occur during execution.

**Identified Problems**

The script encounters issues with reading and processing the Excel files, as indicated by the error logs. Specifically, there are problems with the number of columns after splitting and encoding errors.

* **Unnecessary Full HTML Output**: Printing the entire HTML content (print(pages[0].content.decode("utf-8"))) is not useful in the terminal and slows down the process.  
  Solution: Remove this line and work only with structured data extracted from the HTML.
* **Manual and Repetitive Data Processing**: Many operations, such as preprocessing descriptions, dates, and countries, are scattered and could be centralized.  
  Solution: Create specific functions for cleaning and transformation.
* **Duplication in CSV Writing:** Several parts of the code save and reread CSV files.  
  Solution: Consolidate DataFrame handling and save the data only once, at the end.
* **Lack of Validation for Extracted Data:** The code does not validate if the entries contain valid data before processing them.  
  Solution: Filter incomplete entries before building the DataFrame.

**Possible Optimization of the Script Already Attached by DeNexus**

* Line 94: Checking date\_processed for null values (pd.isna) can be ambiguous if unconvertible values are not handled correctly.
* Line 136-142: Text replacement for identifying countries (df['description\_preprocessed'].str.replace('USA', 'United States')) does not consider case sensitivity. The .str.contains() method in a loop can be slow due to repetitive processing in large datasets.
* Line 157: Grouping by intervals (pd.cut) might create overlapping or incorrect intervals if boundary values are not well-defined.

**Compilation Errors Detected in the default Python Script**

-Line 29-43: Incorrect use of ValueError in the else block of the parse\_div method.

Explanation In Python, an else block in a try-except statement runs *only if no exception is raised* in the try block.

However, in the given context, the else block is raising a ValueError, which contradicts its purpose. If the try block succeeds (i.e., no exception occurs), the code should *not* raise another exception.

-Line 45: Incorrect CSV file writing; df.to\_csv() works, but the /dbfs file prefix is redundant for this Databricks path.

The /dbfs prefix is often used in Databricks for interacting with DBFS from non-Databricks systems or local environments, but it is not necessary when writing files from within Databricks notebooks.

-Line 68: The string format used in pd.to\_datetime() may not match some non-standard formats, potentially causing exceptions.// pandas.to\_datetime() is a function that converts a string or series of strings into datetime objects. It is highly flexible, but it requires that the input string match a recognized date format. When it encounters a string that does not match an expected or specified format, it will raise an exception.

-Line 72-74: The isTimeFormat method does not handle edge cases like empty strings or null values.

Explanation pandas.to\_datetime() converts strings to datetime objects, but invalid formats can cause errors. Using errors='coerce' converts invalid dates to NaT instead of raising exceptions.

**Roadblock**

Encoding Issues: The script fails to read the Excel files due to encoding errors ('utf-8' codec can't decode byte 0x90).

Column Splitting: The script expects 4 columns after splitting but encounters 18 columns, leading to errors.

**Roadblock Solution**

Encoding Issues: Ensure the Excel files are saved with the correct encoding or specify the correct encoding when reading the files.

Column Splitting: Verify the structure of the Excel files and adjust the splitting logic to match the actual data format.

**Optimizable**

File Reading: Optimize the file reading process to handle different encodings and file formats more robustly.

Error Handling: Improve error handling to provide more informative messages and potentially recover from certain errors.

**Optimization**

File Reading: Use pd.read\_excel(file\_path, engine='openpyxl') to handle different Excel file formats and encodings.

Error Handling: Add more specific error messages and consider implementing retry logic for transient errors.

**Optimization Status**

File Reading: Not yet optimized. The current script does not handle different encodings or file formats robustly.

Error Handling: Partially optimized. The script logs errors but could provide more detailed messages and handle certain errors more gracefully.