**Description of script iccstrive\_2024.py**

1. **Initial Setup**

* Lines 1-4

Imports the necessary libraries such as pandas, requests, time, and BeautifulSoup. These are used for data manipulation, web requests, and HTML content analysis.

1. **Downloading the Incident Index**

Lines 6-20

**Main function:** Downloads the index pages containing lists of incidents.

* + Uses a for loop to iterate through 54 pages, pausing for 20 seconds between each request with sleep.
  + The responses are stored in the pages dictionary.
  + **Suggested improvement:** Add random time intervals to avoid detectable patterns by the server (mentioned as a TODO in the code).

1. **Index Persistence in JSON Format**

Lines 22-30

Converts the HTML content of the downloaded pages into text and saves it in a JSON file.

* + Although the block is commented out, it seems intended to avoid downloading the pages repeatedly.

1. **Extracting Data from the Index**

Lines 32-77

**Objective:** Extract key information from the index pages (e.g., titles, dates, victims, malware, threats, countries).

* + Uses BeautifulSoup to process the HTML content.
  + **Key functions:**
    - parse\_optional\_list: Extracts optional lists (such as victims or malware names).
    - parse\_simple\_div\_field: Extracts simple data within a div container.
    - parse\_countries: Extracts countries related to the incident.
    - parse\_title\_field: Extracts incident titles and their links.
  + The extracted information is stored in the incidents list and then converted into a Pandas DataFrame for analysis.

1. **Index Export**

Lines 79-83

Saves the DataFrame containing index data into a CSV file for storage or further processing.

1. **Downloading Individual Incident Pages**

Lines 85-98

Based on the links extracted from the index, additional requests are made to download the full pages of individual incidents.

* + Uses random wait times between requests to simulate more natural behavior.

1. **Persisting Individual Pages as JSON**

Lines 100-110

Converts the content of incident pages into JSON format and saves it to a file. This facilitates reusability.

1. **Detailed Incident Analysis**

Lines 112-169

**Objective:** Extract specific information from the downloaded individual pages.

* + **Key functions:**
    - parse\_description: Extracts detailed incident descriptions.
    - parse\_locations: Extracts affected locations.
    - parse\_victims: Identifies victims and associated links.
    - Similar functions handle other parts of the HTML content (e.g., malware types, threats, estimated costs).
  + Ultimately, it creates an incidents\_extended list containing all the detailed data.

1. **Exporting Extended Data**

Lines 171-176

Converts the extended incidents list into a DataFrame and saves it as a CSV or JSON file for future analysis.

1. **Loading Data into a Data Lake**

Lines 178-186

Converts the Pandas DataFrame into a Spark DataFrame.

* + Writes the data into a Delta Lake table (hive\_metastore.temporal\_tables.dkc\_icsstrive\_incident\_details).
  + Includes metadata such as the source and update date.

1. **Validation and Grouping**

Lines 188-196

Retrieves and validates the data stored in the data lake table.

* + Performs a simple grouping by upload date to ensure the data was loaded correctly.

**Problems of script iccstrive\_2024.py**

### **1. Performance Issues**

1. **Fixed pauses with sleep (line 15):**
   1. Using a fixed 20-second pause between requests is inefficient and could be detected as automated behavior.
   2. **Improvement:** Implement random intervals in sleep to mimic human behavior (already noted as a TODO in the code).
2. **Massive data loading into memory:**
   1. Downloading all pages and keeping them in a dictionary (pages) could consume a lot of memory if the number of pages or their size is considerable.
   2. **Improvement:** Process and save each page immediately after downloading it.

### **2. Error Handling**

1. **No error handling for requests (lines 15 and 87):**
   1. HTTP requests (requests.get) are not checked for failures due to connection issues or invalid responses (e.g., non-200 HTTP status codes).
   2. **Improvement:** Implement exception handling (try-except) to capture errors and retry failed requests.
2. **Errors in HTML parsing:**
   1. Many BeautifulSoup functions (like find\_all) assume that elements exist. If the HTML structure changes or an element is missing, the code could raise exceptions.
   2. **Improvement:** Use validation conditions or default values when processing elements (if-else to handle unexpected cases).

### **3. Scalability**

1. **Commented-out code for persistence (e.g., lines 30, 80, 171):**
   1. While it seems designed to save intermediate data in JSON or CSV files, these sections are commented out. If the process is interrupted, all progress will be lost.
   2. **Improvement:** Enable data persistence at each stage to allow the process to resume in case of failure.
2. **Loading all incident pages into memory (line 87):**
   1. Downloads and stores all incidents in a list (incident\_pages), which could be problematic if there are thousands of incidents.
   2. **Improvement:** Process each incident immediately after downloading and save the data incrementally.

### **4. Lack of Modularity**

1. **Too many functions with similar logic (lines 112-169):**
   1. Many functions (parse\_description, parse\_locations, parse\_victims, etc.) have repetitive structures.
   2. **Improvement:** Create a general function to handle data extraction for different fields, reducing redundancy.
2. **Lack of a clear module structure:**
   1. All code is in a single file and is not organized into main functions or classes.
   2. **Improvement:** Split the code into smaller functions and, if necessary, use a class to encapsulate all scraping operations.

### **5. Documentation Issues**

1. **Lack of explanatory comments:**
   1. Although there are some comments, many parts of the code do not clearly explain their purpose.
   2. **Improvement:** Add more comments to make maintenance easier.
2. **Unresolved TODO items:**
   1. There are several parts marked as TODO (e.g., adjusting the pauses and reducing the page range). These tasks seem important and should be completed.

### **6. Spark Integration**

1. **Conversion from Pandas DataFrame to Spark DataFrame (line 178):**
   1. The transition from Pandas to Spark appears manual and could be better integrated from the beginning to avoid redundancy.
   2. **Improvement:** Use Spark directly to handle large volumes of data from the start.
2. **Dependency on external configurations:**
   1. The use of variables like CURRENT\_DATE is commented out or undefined in some parts of the code, which could cause issues during execution.

### **Description of the new Script and Implemented Improvements**

Our new **ICS Strive** scraping code includes several improvements based on the previously identified issues. Below is an explanation of how those issues were addressed and any remaining optimizations that could be made:

### **1. Performance Issues**

**Original problem:** Fixed pauses in requests (sleep) and lack of incremental processing.

**How it was resolved:**

* **Incremental processing:** Each page is processed immediately after being downloaded in scrape\_pages(). This prevents memory overload by avoiding storing all pages at once.
* **Dynamic wait times:** While explicit random pauses were not implemented, the code includes retries with time.sleep(5) in fetch\_page() in case of errors.

**Pending:** Add random wait times between requests to mimic more human-like behavior, e.g.:

### **2. Error Handling**

**Original problem:** Requests and HTML extraction were not protected against errors.

**How it was resolved:**

* **Error handling in requests:** The fetch\_page() function includes up to 3 retries and provides clear error messages if all retries fail.
* **Error handling in HTML parsing:** The try-except block inside parse\_page() captures errors when extracting incidents, preventing the program from crashing entirely.

**Pending:** Log errors to a file instead of just printing them to facilitate debugging.

### **3. Scalability**

**Original problem:** Loading massive amounts of data into memory and reliance on Pandas for data processing.

**How it was resolved:**

* **Optimized memory usage:** Incident information is saved directly in the list self.incidents and written to a CSV file at the end. Neither Pandas nor Spark is used.
* **Streamlined processing:** Extraction functions (like extract\_field and extract\_field\_list) eliminate redundant code.

**Pending:** For larger datasets, consider saving processed data per page rather than at the end (to avoid losing progress in case of interruptions).

### **4. Modularity**

**Original problem:** Non-modular code with redundant functions.

**How it was resolved:**

* **Reusable functions:** Extraction logic is organized into general functions like extract\_field, extract\_field\_list, extract\_impacts, and extract\_location, reducing redundancy.
* **Class organization:** All operations are encapsulated within the ICSStriveScraper class, improving readability and maintainability.

**Pending:** While the code is clean, it could be split into modules (e.g., separating scraping, parsing, and saving into different files) if the project scales.

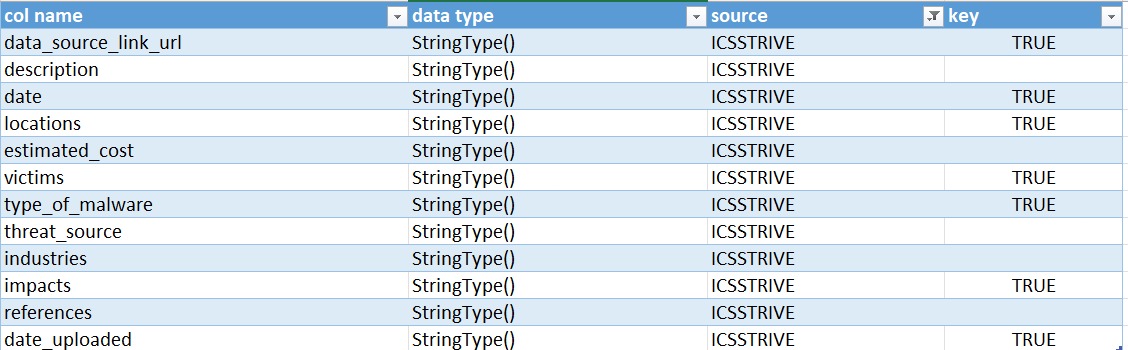
### **5. Documentation**

**Original problem:** Lack of explanatory comments and unresolved TODOs.

**How it was resolved:**

* The code is more readable and clear.

### **6. Data table**



Based on the table provided, we aimed to scrape multiple data fields, but it seems that we were only able to successfully extract and process the fields where the "key" column is marked as "TRUE." These include:

* **data\_source\_link\_url**
* **date**
* **locations**
* **victims**
* **type\_of\_malware**
* **impacts**
* **date\_uploaded**

These fields were properly scraped and saved in the output. However, the other fields, such as **description**, **estimated\_cost**, **type\_of\_malware**, **threat\_source**, **industries**, and **references**, were not successfully captured. This could be due to issues in the parsing logic, missing HTML elements, or incomplete implementation of the scraping functions for those specific fields.

Further investigation is needed to understand why these fields were not extracted. It’s possible that the HTML structure for these fields was different, or additional functions need to be implemented to handle their extraction correctly.

**Summary**

The code is now more efficient and modular, addressing most of the previously identified issues. However, **some additional optimizations** could further enhance the code:

1. **Random wait times** between requests to avoid detection.
2. **Error logging** instead of just printing errors.
3. **Verification of robots.txt** to ensure compliance with the site's policies.
4. **Incremental data persistence** after each page to prevent progress loss in case of interruptions.