Medical Device Data Retrieval: Web Scraping and API Integration

Course: Research Methodology

MSc, Biomedical Engineering

Liontou Agapi Konstantina, 1057757

Introduction

The accurate identification and management of medical devices are critical for ensuring patient safety and regulatory compliance in healthcare systems. Unique Device Identifiers (UDIs) serve as a standardized system for uniquely identifying medical devices, enabling traceability, and facilitating effective device monitoring. Additionally, cross-referencing device data with the Global Medical Device Nomenclature (GMDN) allows for standardized categorization and comparison of devices across the industry.

To address the challenges of retrieving and verifying device information, we present a state-of-the-art code implementation that combines web scraping techniques and integration with the AccessGUDID (Global Unique Device Identification Database) API. Our code offers a powerful solution for healthcare professionals and stakeholders seeking to access, verify, and analyze device data based on UDIs and GMDN names.

The code utilizes web scraping techniques to extract device information from the AccessGUDID website. Leveraging the capabilities of the BeautifulSoup library, it navigates the complex HTML structure of the website, enabling efficient retrieval of essential device details, such as company names, brand names, GMDN names, cross-references, and definitions. The web scraping approach ensures direct access to the most up-to-date device information available on the AccessGUDID platform.

Furthermore, our code integrates seamlessly with the AccessGUDID API, allowing for efficient retrieval of device details based on UDIs. By leveraging the API, we ensure compliance with data usage policies and facilitate access to the comprehensive device database maintained by AccessGUDID. This integration enhances the reliability and accuracy of the retrieved device information.

The significance of cross-checking device data with UDIs and GMDN names cannot be overstated. It enables healthcare professionals to validate device authenticity, verify device attributes, and categorize devices accurately. Our code streamlines this cross-checking process, empowering users to efficiently identify potential discrepancies, perform detailed analyses, and make informed decisions based on reliable device information obtained through scraping and API integration.

Background

Why is it important to have a device identification system?

Device identification systems are crucial for ensuring patient safety by accurately tracking and verifying medical devices used in patient care. They enable manufacturers and regulatory authorities to maintain quality control throughout the device lifecycle, identifying and addressing issues or defects. Compliance with regulatory requirements ensures devices meet safety and quality standards. These systems facilitate traceability, aiding in the quick identification and removal of recalled devices. They improve supply chain management, reduce waste, and enhance efficiency. Robust identification systems support post-market surveillance, allowing monitoring of device performance and safety. They also provide standardized data for research and analysis, leading to advancements in healthcare. Overall, device identification systems bring transparency, accountability, and efficiency to the medical device ecosystem, benefiting patients, manufacturers, providers, and regulatory authorities. [1]

What is the Unique Device Identification (UDI) System?

The Unique Device Identification (UDI) system is a regulatory framework implemented by authorities such as the FDA to uniquely identify medical devices. Its purpose is to enhance patient safety, improve post-market surveillance, and facilitate device traceability and recalls.

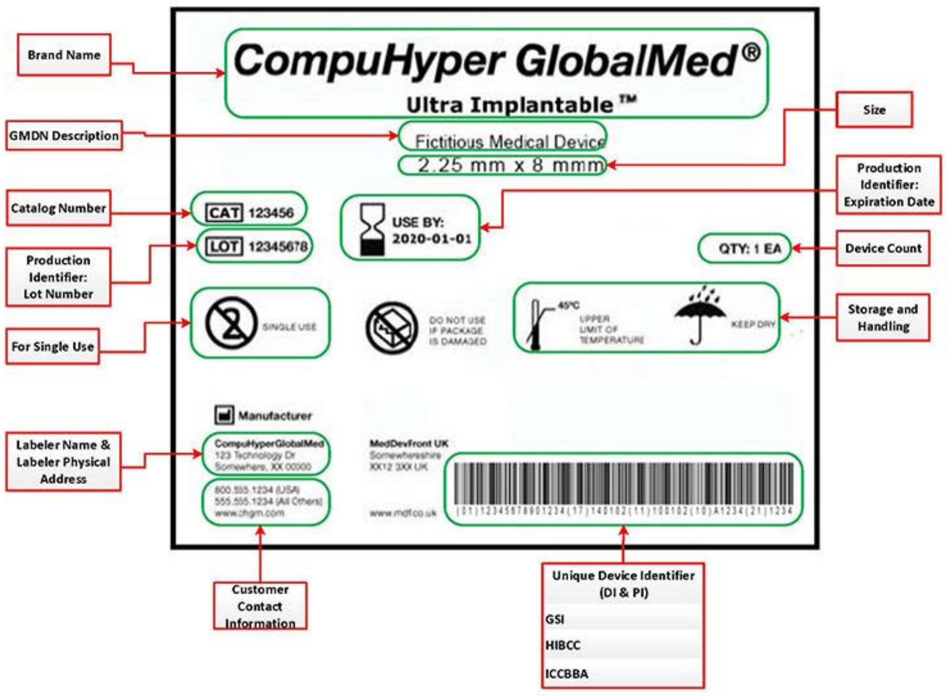
A UDI consists of two components: the Device Identifier (DI) and the Production Identifier (PI). The DI is a fixed portion that identifies the labeler and the specific version or model of a device. The PI is a variable portion that includes information like lot or batch numbers, serial numbers, expiration dates, manufacturing dates, or distinct identification codes.

UDIs are provided in two forms on labels and packages: easily readable plain-text and machine-readable format using automatic identification and data capture (AIDC) technology such as barcodes.

By implementing the UDI system, regulators, healthcare providers, and manufacturers can effectively track and manage medical devices throughout their lifecycle. This simplifies product recalls, facilitates adverse event reporting, and improves the accuracy of device data in electronic health records.

The UDI system also enables the establishment of publicly accessible databases like the FDA's Global Unique Device Identification Database (GUDID). These databases allow manufacturers to submit and maintain device information, which can be accessed by healthcare professionals, patients, and regulatory authorities.

The ultimate goal of the UDI system is to establish a standardized and globally recognized system for uniquely identifying medical devices, promoting consistency and improving overall device identification practices. [1]



Brand Name
CompuHyper GlobalMed®
Ultra Implantable™
GMDN Description
Fictitious Medical Device
2.25 mm x 8 mmm
Size
Catalog Number
CAT 123456
Production Identifier: Lot Number
LOT 12345678
USE BY: 2020-01-01
Production Identifier: Expiration Date
QTY: 1 EA
Device Count
For Single Use
SINGLE USE
DO NOT USE IF PACKAGE IS DAMAGED
45°C UPPER LIMIT OF TEMPERATURE
KEEP DRY
Storage and Handling
Manufacturer
Labeler Name & Labeler Physical Address
CompuHyperGlobalMed 123 Technology Dr Somewhere, XX 00000
800.555.1234 (USA) 555.555.1234 (All Others) www.chgm.com
MedDevFront UK Somewhereshire XX12 3XX UK
www.mdf.co.uk
Customer Contact Information
Unique Device Identifier (DI & PI)
GSI
HIBCC
ICCBBA

Figure 1: UDI label on a device

What is the Global Medical Device Nomenclature (GMDN)?

The Global Medical Device Nomenclature (GMDN) is an internationally recognized system that provides a common language and coding structure for naming and classifying medical devices. Managed by the Global Medical Device Nomenclature Agency (GMDNA), the GMDN aims to promote consistency and facilitate communication across different regions and regulatory frameworks.

Through the GMDN, unique names and codes are assigned to medical devices based on their intended use, principle of operation, and other defining characteristics. The system covers a wide range of devices and assigns a hierarchical code to each device, indicating its general category, primary purpose, mechanism of action, and additional distinguishing features.

The GMDN standardizes the terminology used to describe and identify medical devices, benefiting manufacturers, regulators, healthcare providers, and other stakeholders. It improves clarity, consistency, and interoperability in various processes involving medical devices, such as product registration, adverse event reporting, and supply chain management [2].

Relation between UDI and GMDN

UDI and GMDN are complementary systems. UDI provides a unique identification framework for medical devices, while GMDN offers a standardized naming and classification system. The use of GMDN codes can support the categorization and description of devices within the UDI system, contributing to better identification and communication in the medical device field.

State of the Art

Data Extraction Methods

In today's modern era, the extraction of data from a vast array of web sources has become an indispensable requirement for various business operations, research and development endeavors, academic studies, and more. This data extraction process involves retrieving pertinent data content, including metadata, which holds immense value across diverse objectives. Two distinct and effective methods for achieving this are traditional **web scraping** and the utilization of **APIs**. It is widely recognized that both web scraping and APIs involve automated techniques that are considered among the most practical approaches for harvesting data [3],[4]. These methods enable swift and accurate data collection from multiple web pages and data repositories. Once the extraction is complete, the acquired data is securely stored in a database for future analysis and utilization. When compared to manual data retrieval, these automated techniques significantly outperform in terms of accuracy and time efficiency.

Web Scraping

Web scraping, also known as web extraction or harvesting, refers to the process of extracting data from the World Wide Web (WWW) and storing it in a file system or database for future retrieval or analysis. It can be done manually by a user or automatically by a bot or web crawler using HTTP or a web browser [3].

Web scraping has gained significant recognition as an effective technique for collecting large volumes of heterogeneous data from the constantly evolving web. It has evolved from small ad hoc methods with human assistance to fully automated systems capable of transforming entire websites into well-structured datasets. Modern web scraping tools are not limited to parsing markup languages or JSON files; they can also integrate with computer visual analytics and natural language processing techniques to simulate human browsing behavior.

The ability to customize web scraping techniques allows them to adapt to various scenarios and serve as a powerful tool for collecting big data. State-of-the-art web scraping tools provide advanced functionalities and can handle complex data extraction tasks, contributing to data analysis and research in fields such as computer vision and natural language processing [3].

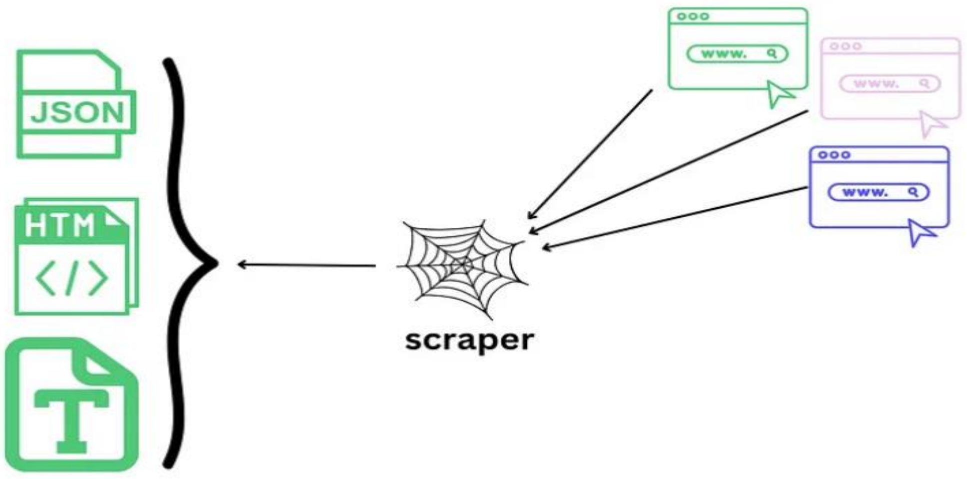


Figure 2: Data extraction process & formats in which data is extracted.

Web Scraping Concepts and Techniques

Data extraction from websites can be achieved using the HTTP protocol, which is utilized by web browsers. This process can be carried out manually through browsing or automated using web crawlers[7].

Various techniques are employed for web scraping:

- *Traditional copy and paste technique:* This involves manual copying and pasting of data, which is effective for small-scale tasks but becomes tedious and impractical for large datasets [8].
- *Text grabbing and regular expressions:* This method uses UNIX commands or regular expressions in programming languages to extract data from web pages[8].
- *Programming with HTTP:* By making HTTP requests to web servers using socket programming, data can be accessed from both static and dynamic webpages [8].
- *HTML parsing:* Semi-structured data query languages are used to parse the HTML code of webpages and retrieve and transform content [8].
- *DOM parsing:* This method involves embedding a full-featured web browser, such as Mozilla or Internet Explorer, to access dynamic content generated by client-side scripts. The browser controls parse webpages into a Document Object Model (DOM) tree, from which applications can extract content [8].
- *Web scraping software:* There are several software tools available today that provide custom web scraping capabilities. These tools can automatically identify the structure of web pages or provide a recording interface, eliminating the need for complex web scraping scripts. Some software even offers scripting functions for data

extraction and transformation, as well as database interfaces for storing scraped data locally [8].

- *Computer vision-based web page analyzers:* Computer vision and machine learning techniques are used to visually scan web pages and extract relevant information [8].

These techniques offer various approaches to web scraping, allowing data scientists to efficiently collect and analyze data from websites.

Web Scraping Process

The process of web scraping involves two main steps: acquiring web resources and extracting desired information from the acquired data. A web scraping program begins by sending a HTTP request to a targeted website to acquire the necessary resources. This request can be in the form of a URL with a GET query or a piece of HTTP message with a POST query. Once the request is received and processed by the website, the requested resource is retrieved and sent back to the web scraping program. The resource can be in various formats, such as HTML web pages, XML or JSON data feeds, or multimedia files like images, audio, or videos.

After the web data is downloaded, the extraction process continues by parsing, reformatting, and organizing the data in a structured manner. A web scraping program typically consists of two essential modules: one for composing HTTP requests, such as *Urllib2* or *Selenium*, and another for parsing and extracting information from the raw HTML code, such as *Beautiful Soup* or *Pyquery*.

The Urllib2 module provides functions for handling HTTP requests, including authentication, redirections, and cookies. Selenium is a web browser wrapper that allows users to automate the browsing process by programming, using browsers like Google Chrome or Internet Explorer.

For data extraction, Beautiful Soup is designed for scraping HTML and other XML documents. It offers convenient functions for navigating, searching, and modifying a parse tree, as well as tools for decomposing an HTML file and extracting desired information using lxml or html5lib. Beautiful Soup can also detect the encoding of the parsed content and convert it to a readable format.

On the other hand, Pyquery provides a set of Jquery-like functions for parsing XML documents. However, unlike Beautiful Soup, Pyquery primarily supports lxml for efficient XML processing [6].

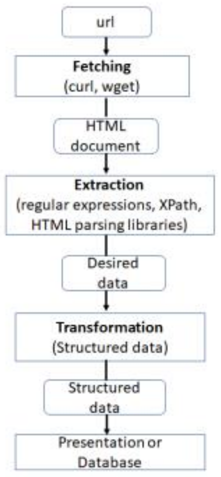


Figure 3: Web scraping process.

Advantages and disadvantages of web scraping

Web scraping offers several **advantages** for data collection and analysis. One of the main benefits is the time-saving aspect, as automating the data extraction process through web scraping can significantly reduce the time required compared to manual methods. Web scrapers can simultaneously pull data from multiple target websites, allowing for the collection of more relevant information than what can be achieved manually.

Web scraping enables the downloading and management of data on a local computer, making it easy to organize and analyze the data in spreadsheets or databases. Additionally, web scrapers can be scheduled to extract real-time data regularly and export it in the desired format, ensuring that the collected data is always up-to-date and readily available.

Data accuracy is crucial for data-driven businesses, and web scraping provides a solution to ensure consistent and accurate data collection. By automating the process, web scrapers reduce the risk of human errors and inconsistencies in the collected data.

Flexibility is another advantage of web scraping. Users have greater control over the amount of data collected and the frequency of scraping, allowing for customization based on specific data needs. This flexibility is particularly beneficial when compared to using APIs, which may offer more limited options in terms of data collection and frequency.

However, web scraping does have some **disadvantages** to consider. Websites often change their HTML structure, which can cause scrapers or crawlers to break. Regular maintenance is necessary to ensure that the scrapers are functioning correctly and can adapt to any changes in the website structure.

Processing and understanding the collected data can also be time-consuming and require effort. It is essential to properly read and interpret the data to derive meaningful insights from it.

Scraping large websites may pose challenges as it often involves a significant number of requests. Some websites may block IP addresses that generate a high volume of requests, requiring the use of proxy servers to bypass these restrictions. Free or cheap proxies may not be effective in such cases, as they are often already blocked.

Additionally, modern websites with dynamic content may render the data only when the page loads in the browser. Extracting data from these sites requires the use of headless browsers, which simulate browser behavior. However, this process can be resource-intensive and may require more time and hardware resources.

Despite these challenges, web scraping remains a valuable tool for data collection and analysis, providing researchers, businesses, and individuals with a wealth of information from various online sources [10] [11].

Application Programming Interface (API)

An API, which stands for Application Programming Interface, is a set of rules and protocols that allows different software applications to communicate and interact with each other. It serves as an intermediary between different software systems, enabling them to exchange data, request services, or perform specific actions [5].

APIs define the methods, parameters, and data formats that applications can use to interact with a particular system or service. They abstract the underlying implementation details, providing a standardized interface that developers can utilize without needing to understand the complexities of the system being accessed.

APIs can be thought of as a contract between the provider of a service (the API) and the consumer (the application). The provider exposes certain functionalities or data through the API, specifying how the consumer can access and utilize them. The consumer, in turn, makes requests to the API following the defined protocols and receives responses containing the requested data or the results of the requested actions.

APIs can be categorized into different types based on their purpose and functionality [12]. Here are a few common types:

- *Web APIs:* Web APIs, or HTTP APIs, are interfaces exposed over the internet that allow applications to interact with web-based services. They enable data retrieval, integration with third-party systems, and the development of web and mobile applications.
- *Operating System APIs:* Operating systems provide APIs that allow applications to access and utilize system resources and functionalities. These APIs provide abstractions for tasks such as file operations, network communication, hardware access, and more.
- *Library APIs:* Libraries and frameworks often provide APIs that encapsulate pre-built functions and classes, allowing developers to easily access and utilize their functionalities without having to write the code from scratch. These APIs simplify development by providing ready-made building blocks.

APIs use various protocols and data formats to facilitate communication between applications. Common protocols include REST (Representational State Transfer), SOAP (Simple Object Access Protocol), GraphQL, and more. Data formats like JSON (JavaScript Object Notation) and XML (eXtensible Markup Language) are commonly used for structuring and exchanging data between the API and the applications [10].

In summary, APIs are interfaces that enable software applications to interact and communicate with each other by defining a set of rules and protocols. They provide a standardized and controlled way for applications to access functionalities, retrieve data, and perform specific actions, enhancing interoperability and enabling the development of complex software systems.

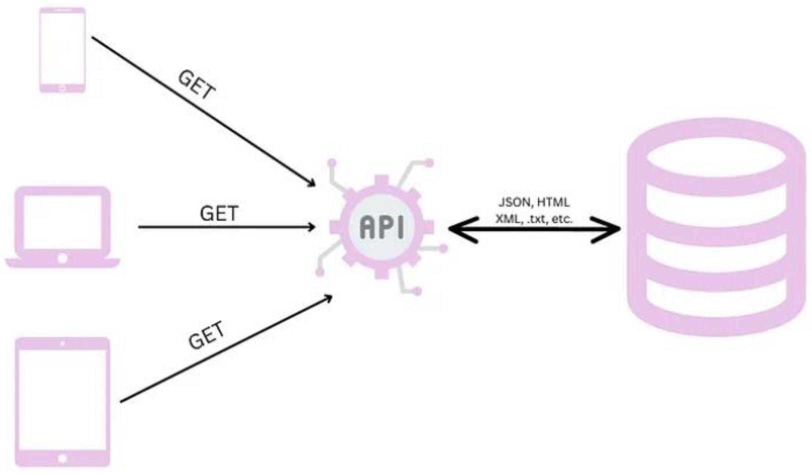


Figure 4: API is the bridge between two software / servers.

API scraping: advantages and disadvantages

API scraping involves collecting data by making requests to specific endpoints discovered through analyzing web server traffic between a site or application. This method has its own set of benefits and drawbacks.

One of the **advantages** of API scraping is that it does not impose additional load on hardware since it leverages existing APIs. It can easily be integrated into developer applications by utilizing the provided credentials. The results obtained from API scraping are often in structured formats like XML or JSON, making them convenient for further processing and analysis. Additionally, API scraping can help overcome challenges such as Javascript rendering and CAPTCHA avoidance. When dealing with large-scale data collection, APIs can offer faster and more efficient solutions compared to traditional web scraping methods.

However, API scraping also has its **limitations**. Not all data may be accessible through a single endpoint, requiring the use of multiple endpoints to gather a complete dataset. Some websites may not provide access to their APIs, restricting the availability of data to HTML content only. Many APIs have rate limits that restrict the frequency of data scraping, which can hinder the effectiveness of web-scraping operations relying on those APIs. Furthermore, APIs generally have limited functionality and are restricted to extracting data from a specific website or a set of predefined functions.

In contrast, traditional web scraping allows for data collection from multiple websites, providing more extensive coverage. It offers greater flexibility in terms of the range of functions and data sources that can be accessed.

In summary, API scraping offers benefits such as reduced hardware load, ease of integration, structured data formats, and faster data collection. However, it is limited by the availability of endpoints, API restrictions, and restricted functionality compared to traditional web scraping methods. The choice between API scraping and web scraping depends on the specific requirements and constraints of the data collection task [10] [11].

Web Scraping vs API

Web scraping involves extracting specific information from multiple websites and organizing it into a structured format, while APIs provide access to data from applications or software, with limitations set by the data owner. Web scraping allows for flexibility in extracting data from various websites using scraping tools, whereas APIs offer direct access to specific data, which may come with limitations and potential costs. Web scraping can collect data from multiple websites, while API access is typically limited to a single website. APIs provide access to a limited set of data, while web scraping allows for a wider range of data collection. Data obtained through web scraping may require cleaning and parsing, whereas APIs provide data in a machine-readable format. APIs also offer faster data extraction compared to web scraping. Despite their differences, both web scraping and API scraping serve the purpose of providing data to users. They enable the collection of customer information, insights, email addresses for marketing, lead generation, and more, opening up endless possibilities for data utilization.

Web scraping in the healthcare or medical device domain

Web scraping has **various applications in the healthcare and medical device domain**. One such application is in research and analysis, where web scraping can be used to gather data from medical research websites, scientific journals, and healthcare forums. This data can then be analyzed to identify trends, assess patient outcomes, and evaluate treatment efficacy, contributing to evidence-based research and advancements in medical technology.

Another area where web scraping proves useful is in monitoring regulatory compliance. By scraping regulatory authorities' websites, manufacturers and healthcare providers can stay updated on medical device regulations, guidelines, and safety alerts. This helps ensure compliance and facilitates adherence to regulatory changes.

Web scraping can also provide valuable competitive intelligence in the medical device market. By monitoring competitor websites, pricing information, product launches, and customer reviews, organizations can gain insights for strategic decision-making and market analysis.

In the realm of clinical trials and drug development, web scraping can gather data on ongoing trials, trial results, and adverse event reports from medical research databases. This information aids in monitoring trial progress, identifying potential drug interactions or adverse events, and supporting the development of new drugs.

For supply chain management, web scraping can assist in tracking and monitoring details related to the medical device supply chain. This includes information on suppliers, pricing data, product availability, and specifications. By leveraging web scraping, organizations can optimize inventory management, streamline procurement processes, and ensure efficient supply chain operations.

Web scraping can also be employed to gather patient feedback, reviews, and ratings from healthcare review websites and social media platforms. This data provides insights into patient experiences, satisfaction levels, and areas for improvement in healthcare services or medical devices.

It is crucial to ensure compliance with relevant laws and regulations, including data privacy and intellectual property rights, when conducting web scraping in the healthcare or medical device domain [13] [14].

Methodology

Python for Web Scraping

Python is widely regarded as an excellent choice for web scraping, and there are several reasons to support this assertion. Firstly, Python is one of the most popular programming languages, renowned for its simplicity and ease of learning. Even individuals with no prior programming experience can quickly grasp Python's concepts. Its popularity has fostered a vast and supportive community, ensuring that assistance is readily available when encountering challenges during the coding process. Given Python's extensive user base, it is highly likely that someone has encountered and resolved a similar problem, with solutions often documented and shared online. While this collaborative nature is not unique to Python, the language's accessibility and longstanding popularity have contributed to the development of one of the largest and most diverse programming communities today.

Another compelling reason to opt for Python when engaging in web scraping is its code readability. Python code is designed to be easily understood, promoting clear and concise programming practices. This readability not only facilitates efficient work during the development phase but also enables other programmers to comprehend the accomplishments of the code. Moreover, it significantly simplifies the process of revisiting and comprehending one's own code after a period of time. The ability to easily understand well-written Python code enhances code maintenance and promotes code reuse, resulting in more streamlined and efficient programming practices [12].

My code

The purpose of the presented code is to retrieve device information from the AccessGUDID (Global Unique Device Identification Database) API based on Unique Device Identifiers (UDIs) or perform advanced searches based on company name and brand name. It is used to gather detailed information about medical devices, such as their company name, brand name, GMDN (Global Medical Device Nomenclature) name, cross-references, and definitions.

The code allows users to input an Excel file containing UDIs and associated information (this excel file has data extracted from web-Praxis). It then utilizes the AccessGUDID API to fetch device information for each UDI or performs advanced searches based on company name and brand name. The retrieved information is processed and organized into a dictionary. Finally, the collected device data is stored in an Excel file for further analysis or reporting.

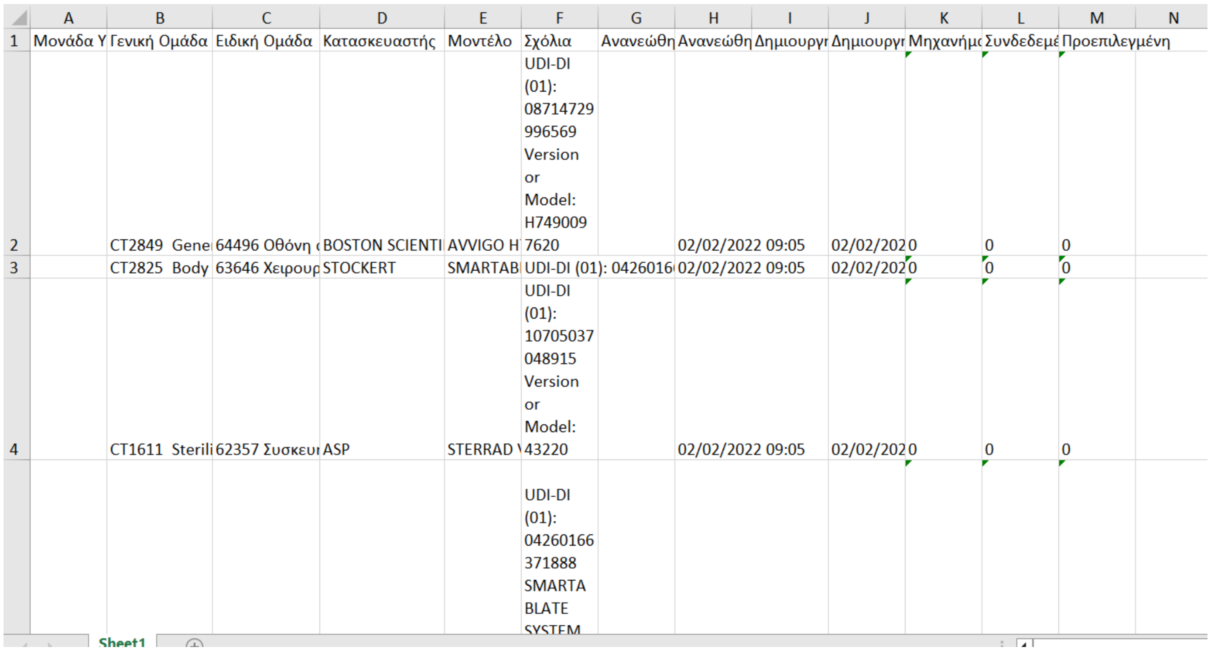
The code can be utilized in various scenarios, such as medical device management, regulatory compliance, research, and analysis. It provides a way to access comprehensive device information from a reliable and authoritative database, enabling users to gain insights, perform comparisons, and make informed decisions related to medical devices.

Data

The excel sheet I was given contained data from web praxis and was listed as follows (the ones we are interested in):

- 2nd column: General Group (A more general categorization of the medical machine group to which the machine belongs according to the Global Medical Device Nomenclature (GMDN), e.g., CT196 Centrifuges)
- 3rd column: Specific Group (The medical device group to which the machine belongs according to GMDN, e.g., 36465 Centrifuge, tabletop, low/medium speed)
- 4th column: Manufacturer
- 5th column: Model
- 6th column: Comments (here is usually the UDI)

In total there were 279 entries (this means 279 different medical devices)



	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1	Μονάδα Υ	Γενική Ομάδα	Ειδική Ομάδα	Κατασκευαστής	Μοντέλο	Σχόλια	Ανανεώθη	Ανανεώθη	Δημιουργι	Δημιουργι	Μηχανήμα	Συνδεδεμέ	Προεπιλεγμένη	
2		CT2849 Gene	64496 Οθόνη (	BOSTON SCIENTI	AVVIGO H	UDI-DI (01): 08714729996569 Version or Model: H7490097620		02/02/2022 09:05		02/02/202	0	0	0	
3		CT2825 Body	63646 Χειρουρ	STOCKERT	SMARTAB	UDI-DI (01): 0426016		02/02/2022 09:05		02/02/202	0	0	0	
4		CT1611 Sterili	62357 Συσκευι	ASP	STERRAD \	UDI-DI (01): 10705037048915 Version or Model: 43220		02/02/2022 09:05		02/02/202	0	0	0	
5						UDI-DI (01): 04260166371888 SMARTABLATE SYSTEM								

Sheet1

For the code (code provided in the Appendix)**:**

Firstly, I import the necessary libraries:

- ✓ ***requests*** for making HTTP requests,
- ✓ ***pandas*** for data manipulation,
- ✓ ***tkinter*** for creating a GUI,
- ✓ ***re*** for regular expressions,
- ✓ ***time*** for time-related operations, and
- ✓ ***BeautifulSoup*** for web scraping.

And then I make the functions:

- ✓ ***get_api:*** makes an HTTP GET request to the AccessGUDID API, passing a unique device identifier (UDI) as a parameter. It retrieves device information in JSON format and returns the JSON response.

- ✓ ***user_input:*** opens a file dialog using tkinter to allow the user to select an Excel file containing UDI data (this excel was provided). It reads specific columns from the file using pandas and returns the extracted data as lists.
- ✓ ***strip_english:*** uses regular expressions to remove non-English characters from a string, leaving only English terms. It returns a list of English words extracted from the input string.
- ✓ ***cross_check_words:*** compares a list of words with a string and checks if at least two words from the list are present in the string. It returns True if the condition is met, False otherwise.
- ✓ ***find_14_digit_number:*** searches a string for a 14-digit number using a regular expression pattern. It returns a list of matches found in the input string.
- ✓ ***format:*** initializes an error count and a dictionary (Device_Data) for storing device information. It returns the error count and the empty dictionary.
- ✓ ***make_keywords:*** takes an input string and extracts English keywords by calling the strip_english function. It returns a list of the extracted keywords.
- ✓ ***search:*** performs a search based on the provided UDI, keywords, and descriptions. It calls the get_api function to retrieve device information from the API. If the API call returns an error, the error count is incremented. If the nonNormalFlag parameter is set, indicating an incorrect UDI, the function populates the data dictionary with relevant information. Otherwise, it checks for keyword matches and populates the data dictionary accordingly. It returns the error count and the populated data dictionary.
- ✓ ***search_By_Queries:*** performs an advanced search based on company name and brand name. It constructs a query string and sends a GET request to the AccessGUDID website. It uses BeautifulSoup to parse the HTML response, extracts the device IDs from the search results, and returns a list of IDs.
- ✓ ***cross_ref_by_numbers:*** takes a list of device IDs, makes API calls for each ID, and counts the occurrences of GMDN names. It returns the GMDN name with the highest occurrence and its corresponding device ID.
- ✓ ***output:*** takes a dictionary of device data and an output directory. It creates a pandas DataFrame from the data and writes it to an Excel file using pd.ExcelWriter.
- ✓ ***main:*** is the entry point of the code. It initializes an empty output dictionary and calls the user_input function to get the UDI data, output directory, categories, company names, and brand names. It iterates over the UDI data, performing searches or advanced searches

depending on the availability of UDI values. It populates the output dictionary with device information if no error occurs. Finally, it calls the output function to generate the Excel file with the device data.

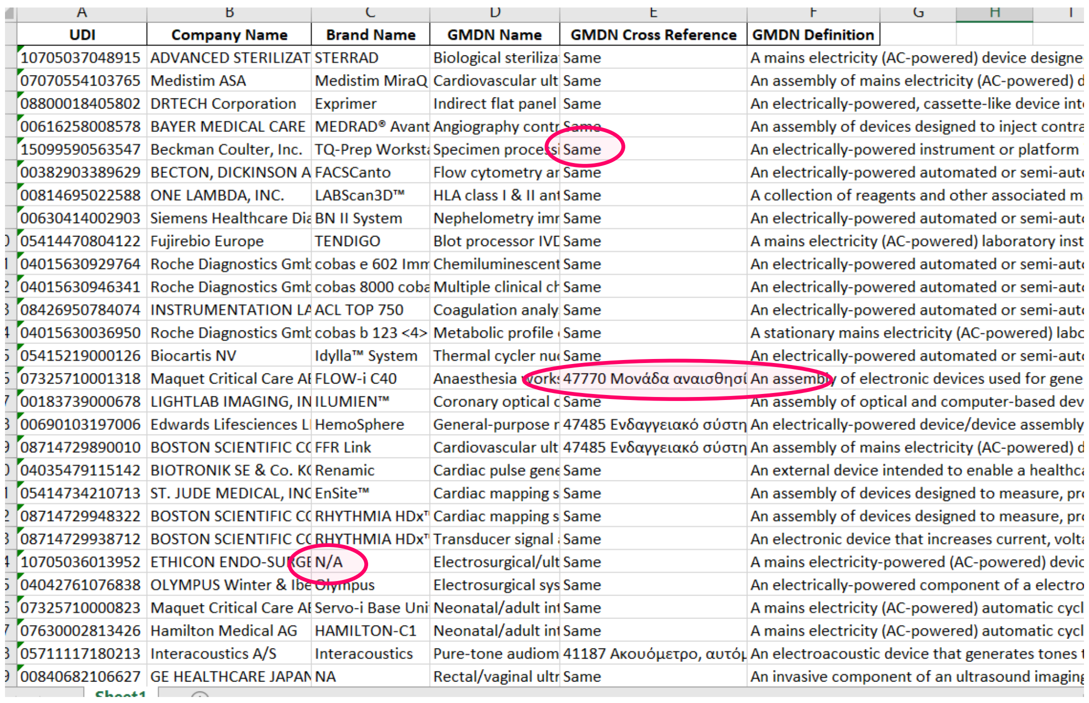
Code Functionalities (briefly):

1. Import data from the Excel file.
2. Extract the 14-digit UDI from the column ‘Σχόλια’.
3. Create an empty dictionary to store the information that will be collected.
4. Perform a search based on the provided UDI, retrieve device information from the API and populate the data dictionary accordingly.
5. Perform an advanced search based on the company name and brand name (for those devices that didn’t have UDI), retrieve the necessary device information through a combination of web scraping (by reconstructing the URL) and API usage, and populate the data dictionary accordingly.
6. Cross-reference the GMDN definition between the initial data and the one found on the website for each device.
7. Present the results.

Results

The results were quite satisfactory. Out of the 279 records initially provided, we found UDI matches for 193 records. By implementing a complex search based on the company name and brand name, we were able to identify 207 records. This indicates that the data provided by Praxis, although challenging to manage due to its unstructured format, demonstrate a fairly accurate recording of medical equipment, with correct UDIs, GMDN names, and definitions.

Specifically, the output of our code is an Excel file in the following format:



UDI	Company Name	Brand Name	GMDN Name	GMDN Cross Reference	GMDN Definition
10705037048915	ADVANCED STERILIZAT	STERRAD	Biological steriliza	Same	A mains electricity (AC-powered) device designe
07070554103765	Medistim ASA	Medistim MiraQ	Cardiovascular ult	Same	An assembly of mains electricity (AC-powered) d
08800018405802	DRTECH Corporation	Exprimer	Indirect flat panel	Same	An electrically-powered, cassette-like device int
00616258008578	BAYER MEDICAL CARE	MEDRAD® Avant	Angiography contr	Same	An assembly of devices designed to inject contra
15099590563547	Beckman Coulter, Inc.	TQ-Prep Workst	Specimen processi	Same	An electrically-powered instrument or platform
00382903389629	BECTON, DICKINSON A	FACSCanto	Flow cytometry ar	Same	An electrically-powered automated or semi-aut
00814695022588	ONE LAMBDA, INC.	LABScan3D™	HLA class I & II ant	Same	A collection of reagents and other associated m
00630414002903	Siemens Healthcare Dia	BN II System	Nephelometry imr	Same	An electrically-powered automated or semi-aut
05414470804122	Fujirebio Europe	TENDIGO	Blot processor IVD	Same	A mains electricity (AC-powered) laboratory inst
04015630929764	Roche Diagnostics Gmb	cobas e 602 Imm	Chemiluminescent	Same	An electrically-powered automated or semi-aut
04015630946341	Roche Diagnostics Gmb	cobas 8000 coba	Multiple clinical ch	Same	An electrically-powered automated or semi-aut
08426950784074	INSTRUMENTATION LA	ACL TOP 750	Coagulation analy	Same	An electrically-powered automated or semi-aut
04015630036950	Roche Diagnostics Gmb	cobas b 123 <4>	Metabolic profile	Same	A stationary mains electricity (AC-powered) labo
05415219000126	Biocartis NV	Idylla™ System	Thermal cycler nu	Same	An electrically-powered automated or semi-aut
07325710001318	Maquet Critical Care AI	FLOW-i C40	Anaesthesia work	47770 Μονάδα αναισθησί	An assembly of electronic devices used for gene
00183739000678	LIGHTLAB IMAGING, IN	ILUMIEN™	Coronary optical c	Same	An assembly of optical and computer-based dev
00690103197006	Edwards Lifesciences L	HemoSphere	General-purpose r	47485 Ενδαγγειακό σύστη	An electrically-powered device/device assembly
08714729890010	BOSTON SCIENTIFIC C	FFR Link	Cardiovascular ult	47485 Ενδαγγειακό σύστη	An assembly of mains electricity (AC-powered) d
04035479115142	BIOTRONIK SE & Co. K	Renamic	Cardiac pulse gene	Same	An external device intended to enable a healthc
05414734210713	ST. JUDE MEDICAL, INC	EnSite™	Cardiac mapping s	Same	An assembly of devices designed to measure, pr
08714729948322	BOSTON SCIENTIFIC C	RHYTHMIA HDx™	Cardiac mapping s	Same	An assembly of devices designed to measure, pr
08714729938712	BOSTON SCIENTIFIC C	RHYTHMIA HDx™	Transducer signal	Same	An electronic device that increases current, volt
10705036013952	ETHICON ENDO-SURGE	N/A	Electrosurgical/ult	Same	A mains electricity-powered (AC-powered) devic
04042761076838	OLYMPUS Winter & Ibe	Olympus	Electrosurgical sys	Same	An electrically-powered component of a electro
07325710000823	Maquet Critical Care AI	Servo-i Base Uni	Neonatal/adult int	Same	A mains electricity (AC-powered) automatic cycl
07630002813426	Hamilton Medical AG	HAMILTON-C1	Neonatal/adult int	Same	A mains electricity (AC-powered) automatic cycl
05711117180213	Interacoustics A/S	Interacoustics	Pure-tone audiom	41187 Ακουόμετρο, αυτόμ	An electroacoustic device that generates tones t
00840682106627	GE HEALTHCARE JAPAN	NA	Rectal/vaginal ultr	Same	An invasive component of an ultrasound imaging

Where:

- **1st column: UDI**, UDIs are listed. It includes both the UDIs that were initially matched and the "N/A" symbol for equipment that had no UDI in the initial data and required an advanced search.
- **2nd column: Company Name** as registered on the website.
- **3rd column: Brand Name** as registered on the website.
- **4th column: GMDN Name** as registered on the website.
- **5th column: GMDN Cross Reference**. The term "same" is displayed for records that have the same GMDN Name on both the website and the initial Excel data. For records that are not the same, a different description is provided. This description was obtained by the initial data.
- **6th column: GMDN Definition** as registered on the website for the devices that were "identified" by the UDI. For the devices that we didn't know the UDI and we made

an advanced search, we had many results. So, the GMDN definition was obtained by examining the first page of search results on the website generated by the advanced search. We identified all the GMDN names and applied a sorting algorithm to select the name that appeared most frequently.

Overall, the output provides a comprehensive overview of the matched UDIs, associated company names, brand names, GMDN names, cross-references, and definitions, allowing for further analysis and verification of the recorded medical equipment data.

Conclusion

In conclusion, the implemented code combining web scraping techniques and integration with the AccessGUDID API has proven to be an essential tool for retrieving, validating, and analyzing medical device data based on UDIs and Global Medical Device Nomenclature (GMDN) names.

The need for web scraping arises from the vast amount of device information available on websites like AccessGUDID. Web scraping allows for efficient extraction of specific data elements, such as company names, brand names, GMDN names, cross-references, and definitions, from complex HTML structures. By automating the data retrieval process, web scraping provides direct access to the most up-to-date device information, ensuring accuracy and timeliness.

The integration of an API, in this case, the AccessGUDID API, further enhances the code's functionality and reliability. By leveraging the API, the code establishes a secure connection to the comprehensive device database maintained by AccessGUDID. This integration ensures compliance with data usage policies, facilitates seamless data retrieval, and enhances the reliability and accuracy of the obtained device information.

Moreover, the utilization of UDIs plays a crucial role in device identification and traceability. UDIs provide a standardized system for uniquely identifying medical devices, enabling effective device monitoring, regulatory compliance, and patient safety. The code's ability to cross-check device data with UDIs and GMDN names ensures the authenticity of devices, verifies their attributes, and allows for accurate categorization. This cross-checking process enables healthcare professionals and stakeholders to identify and address potential discrepancies, ultimately enhancing patient safety and informed decision-making.

In summary, the combination of web scraping techniques, API integration, and the utilization of UDIs addresses the challenges of accessing, validating, and analyzing medical device data. The implemented code streamlines these processes, empowering users to efficiently retrieve reliable device information, verify device attributes, and make informed decisions. As technology continues to evolve, further advancements in web scraping and API integration will contribute to more efficient and accurate device data management and analysis in the healthcare industry.

Suggestions

To further enhance the effectiveness of the solution, several suggestions can be considered.

First, it is crucial to ensure data quality by implementing measures to validate and clean the input data, ensuring its accuracy and completeness. Additionally, automating the code execution process and developing batch processing capabilities can improve efficiency when handling large volumes of data. Integration with existing systems, such as healthcare or inventory management systems, would enable the synchronization of data and enhance decision-making capabilities.

To address data privacy and security concerns, implementing measures like data anonymization, encryption, or access controls is essential. Collaborations and partnerships with regulatory bodies, healthcare institutions, or manufacturers can facilitate data sharing and foster industry-wide improvements in device identification and data management practices.

Continuous data monitoring is also recommended to periodically retrieve and update device information from reliable sources, ensuring data accuracy and relevancy. Providing comprehensive documentation and user support materials would assist users in effectively utilizing the code and maximizing its potential.

By implementing these suggestions, the code can be enhanced to improve data management, system integration, and overall efficiency in device identification and data retrieval processes. These enhancements will contribute to a more reliable and streamlined approach in accessing, verifying, and analyzing device data based on UDIs and GMDN names.

References

[1] https://www.fda.gov/medical-devices/device-advice-comprehensive-regulatory-assistance/unique-device-identification-system-udi-system

[2] https://www.gmdnagency.org/

[3] https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=10007167

[4] C. Slamet, R. Andrain, D. Maylawati, Suhendar, W. Darmalaksana and M. Ramdhani, "Web scraping and Naive Bayes classification for job search engine," in IOP Conference Series, 2018. (8)

[5] https://academic.oup.com/bib/article/15/5/788/2422275?login=false

[6] https://www.researchgate.net/profile/Bo-Zhao-3/publication/317177787_Web_Scraping/links/5c293f85a6fdccfc7073192f/Web-Scraping.pdf

[7] http://ijasca.zuj.edu.jo/PapersUploaded/2021.3.11.pdf

[8] Sirisuriya, D. S., (2015). A comparative study on web scraping. Proceedings of 8th International Research Conference, KDU.

[9] https://www.researchgate.net/profile/Bo-Zhao-3/publication/317177787_Web_Scraping/links/5c293f85a6fdccfc7073192f/Web-Scraping.pdf idio me prin

[10] https://www.scrapingdog.com/blog/web-scraping-vs-api/

[11] https://scrape-it.cloud/blog/web-scraping-vs-api

[12] https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=10007167

[13] https://www.linkedin.com/pulse/how-does-web-scraping-assists-internet-healthcare-data-collection-

[14] https://python.plainenglish.io/20-ways-healthcare-companies-can-use-web-scraping-to-improve-patient-care-and-enhance-their-eb1786a3dc8

[15] http://ijasca.zuj.edu.jo/PapersUploaded/2021.3.11.pdf AGAIN

Appendix

```python
import requests
import pandas as pd
import tkinter as tk
from tkinter import filedialog
import re
import time
from bs4 import BeautifulSoup


def get_api(udi): # retrieves device information in JSON format and returns the JSON response
    g = requests.get(
        'https://accessgudid.nlm.nih.gov/api/v2/devices/lookup.json?di='+str(udi))
    return g.json()


def user_input():
    # Dialog box. Enter UDI list excel file
    root = tk.Tk()
    root.withdraw()
    input_file_path = filedialog.askopenfilename(
        filetypes=[('.xlsxfiles', '.xlsx')], title='Select UDI list excel file')
    i = input_file_path.rfind('/')
    # output folder path = input folder path
    output_folder_path = input_file_path[:i + 1]

    df = pd.read_excel(input_file_path, usecols=['Σχόλια'])
    udi_list = df['Σχόλια'].tolist()

    df2 = pd.read_excel(input_file_path, usecols=['Ειδική Ομάδα'])
    udi_list2 = df2['Ειδική Ομάδα'].tolist()

    df3 = pd.read_excel(input_file_path, usecols=['Κατασκευαστής'])
    udi_list3= df3['Κατασκευαστής'].tolist()

    df4 = pd.read_excel(input_file_path, usecols=['Μοντέλο'])
    udi_list4= df4['Μοντέλο'].tolist()

    udis = []
    specific_categories = []
    companyName = []
    brandName = []

    for i in range(0, len(udi_list)):
        companyName.append(udi_list3[i])
        brandName.append(udi_list4[i])
        specific_categories.append(udi_list2[i])
        udis.append(find_14_digit_number(udi_list[i]))

    return udis, output_folder_path, specific_categories, companyName, brandName


def strip_english(string): # find only English terms in specific categories in my excel
    string = re.sub(r"[^A-Za-z]", " ", string.strip())
    words = string.split()
    return words


def cross_check_words(words, string): # compares a list of words with a string
                                      # and checks if at least two words from the
                                      # list are present in the string
    confidence = 0
    for word in words:
        if word in string:
            confidence += 1

    if (confidence > 1): # how many same words
        return True
    else:
        return False


def find_14_digit_number(string): # searches a string for a 14-digit number
                                  # using a regular expression pattern
    pattern = r'\b\d{14}\b'
    matches = re.findall(pattern, string)
    return matches


def format(): #initialize my dictionary for storing my data
    error = 0
    Device_Data = {'UDI': [],
               'Company Name': [],
               'Brand Name': [],
               'GMDN Name': [],
               'GMDN Cross Reference': [],
               'GMDN Definition': []
               }
    return error, Device_Data


def make_keywords(items): # make keywords that will by used in advanced search
                          # for company name and brand name
    keywords = []
    keywords = strip_english(items)
    return keywords  # returns a list with the english keywords found in Excel


def search(udi, keywords, descriptions,nonNormalFlag=False): # performs a search based on the
                                                              # provided UDI, keywords, and
descriptions
    print('searching: ' + str(udi))
    error, data = format()
    call = get_api(udi)

    if ('error' in call):
        error += 1

    elif(nonNormalFlag): # when i have wrong udi
        data['UDI'] = "N/A"
        data['Company Name'] = call['gudid']['device']['companyName']
        data['Brand Name'] = call['gudid']['device']['brandName']
        data['GMDN Name'] = call['gudid']['device']['gmdnTerms']['gmdn'][0]['gmdnPTName']

        if (cross_check_words(keywords, data['GMDN Name'])):

            data['GMDN Cross Reference'] = "Same"
        else:
            data['GMDN Cross Reference'] = descriptions

        data['GMDN Definition'] =
call['gudid']['device']['gmdnTerms']['gmdn'][0]['gmdnPTDefinition']
    else:
        data['UDI'] = udi
        data['Company Name'] = call['gudid']['device']['companyName']
        data['Brand Name'] = call['gudid']['device']['brandName']
        data['GMDN Name'] = call['gudid']['device']['gmdnTerms']['gmdn'][0]['gmdnPTName']

        if (cross_check_words(keywords, data['GMDN Name'])):

            data['GMDN Cross Reference'] = "Same"
        else:
            data['GMDN Cross Reference'] = descriptions

        data['GMDN Definition'] =
call['gudid']['device']['gmdnTerms']['gmdn'][0]['gmdnPTDefinition']

    return error, data


def search_By_Queries(companyName,model): # performs an advanced search based on
                                          # company name and brand name
```

```python
    refs = []
    ids = []
    additive = "companyName:("+companyName+")"
    additive += " AND brandName:("+model+")"
    print("searching: " + additive)
    req = requests.get('https://accessgudid.nlm.nih.gov/devices/search?query='+additive)
    soup = BeautifulSoup(req.content, 'html.parser')
    soups = soup.find_all("div", {"class": "resultRow no-padding"})
    for soup in soups:
        refs.append(soup.find_all("a"))

    for i in range (0, len(refs)):
        te = find_14_digit_number(str(refs[i][0]))
        if(te!=[]):
            ids.append(te[0])
        else:
            return []

    return ids


def cross_ref_by_numbers(ids): # takes a list of device IDs, makes API calls for each ID,
                               # and counts the occurrences of GMDN names
    gmdn_names = {}
    for id in ids:
        print(id)
        ref = get_api(id)['gudid']['device']['gmdnTerms']['gmdn'][0]['gmdnPTName']
        if ref in gmdn_names:
            gmdn_names[ref][1] += 1
        else:
            gmdn_names[ref]=[id,1]
    k = ""
    i = 0
    for key in gmdn_names:

        if (gmdn_names[key][1]>i):
            k = key
            i = gmdn_names[key][1]

    return k,gmdn_names[k][0]



def output(data_dict, dir):
    df = pd.DataFrame(data_dict)  # dataframe containing comparison results

    path = dir + '/' + 'udi_checkresults_advanced.xlsx'

    with pd.ExcelWriter(path) as engine:
        df.to_excel(excel_writer=engine, index=False)


def main():

    output_dict = []
    udis, output_dir, categories, companyName, brandName = user_input()
    for i in range(0,len(udis)):
        if (udis[i] == []):
            #continue  # Comment this line for Advanced Search

            ids = search_By_Queries(companyName[i],brandName[i])
            if (ids == []):
                pass
            else:
                key,id = cross_ref_by_numbers(ids)
                er, D = search(id,make_keywords(brandName[i]),categories[i],True)

        else:
            er, D = search(udis[i][0], make_keywords(categories[i]), categories[i])
        time.sleep(0.1)

        if er == 0:
            output_dict.append(D)

    output(output_dict, output_dir)


main()
```