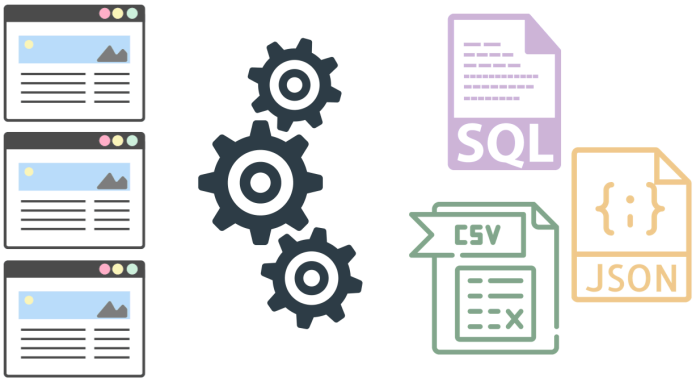
Data Scraping and Cleaning Methodologies for the StudyBuddy Hub Project



Work Distribution:

Scraped Courses, Colleges, Resources Data

Sarah Kayembe

Scraped Quizzes Data

Vivek Reddy Bhimavarapu

Cleaned Courses, Colleges, Quizzes Data

Rise Akizaki

Cleaned Resources Data

Jacob Craig

Table of Contents

Table of Contents
Data Scraping
    College Table
        1. Objective and Justification
        2. Methodology and Tools
        3. Data Source and Scraping Strategy
            1. Data Source
            2. Scraping Strategy
        4. Error Handling
        5. Output
    Course table
        1. Objective and Justification
        2. Methodology and Tools
        3. Data Source and Scraping Strategy
            1. Data Source
            2. Scraping Strategy
        4. Error Handling
        5. Output
    Resources Table
        1. Objective and Justification
        2. Methodology and Tools
        3. Data Source and Scraping Strategy
            1. GeeksforGeeks – Data Structures Section
            2. Khan Academy – Biology Units
            3. W3Schools (Python & Java Tutorials)
            4. Criminal Law (Multi-Source Scrape)
        4. Error Handling and Validation Techniques
            1. Network Errors:
            2. Data Validation:
            3. File Validation:
        5. Output
Data Cleaning
    usm_courses
        1. Algorithm For Cleaning
        2. Saving Results
    us_colleges
        1. Algorithm For Cleaning
        2. Saving Results
    quiz_data

1. Questions Data Frame
Algorithm For Cleaning / Saving Results
2. Answers Data Frame
Algorithm For Cleaning
3. Saving Results
course_resources
1. Objectives and Justification
2. Methodology & Tools
3. Cleaning Algorithm
4. Cleaned DataFrame
5. Validation and Output
Appendix

This report explains how we pulled and cleaned the data to make reliable academic datasets for the StudyBuddy Hub project. Our main aim was to automatically gather good information on colleges, courses, and resources so that way, we skip the mistakes that happen when people type data in by hand.

We used Python and tools like requests, BeautifulSoup, and pandas to grab the data from solid online places, including Wikipedia and college catalogs. After collecting it, we cleaned, checked, and removed repeats to make sure everything was accurate, looked consistent, and would work easily with the project's database.

Data Scraping

College Table

1. Objective and Justification

The U.S. College Name Scraper was created to create a standardized database of American colleges and universities by extracting institution names from Wikipedia's "Lists of American universities and colleges."

The primary goal of this scraping task was to prevent inconsistent or duplicate college entries within the application. Without a verified list, users could manually input college names with spelling variations or abbreviations (e.g., "MIT," "Massachusetts Inst. of Tech," "Massachusetts Institute of Technology"), which would fragment data and incorrectly categorize students under different institutions.

By automating the collection of verified college names, the system ensures uniformity and accurate filtering.

2. Methodology and Tools

The data collection was executed using a Python script (documented in scraping_college_names.ipynb) utilizing the following libraries:

Tool	Purpose
Python requests	Handled HTTP requests to retrieve raw HTML content.

Python BeautifulSoup	Parsed the HTML to locate and extract specific data elements.
Python pandas	Structured the data, removed duplicates, sorted the final output, and managed CSV persistence.

3. Data Source and Scraping Strategy

1. Data Source

The source was List of American Universities and Colleges on Wikipedia with the URL https://en.wikipedia.org/wiki/Lists_of_American_universities_and_colleges. The universities were listed within a U.S. state or territory in a tabular form under standardized classes.

2. Scraping Strategy

Hub Page Crawling:

- The scraper began by fetching the main hub page, which contains hyperlinks to all 50 states and territory-specific lists.
- Using BeautifulSoup, it located all <a> tags whose href attribute matched the pattern /wiki/List_of_colleges_and_universities_in_….
- Each link's absolute URL was reconstructed using urljoin, and the state name was extracted as the key identifier.

Iterative State-Level Scraping:

- For each state link, the scraper requested the page and searched for all <table class="wikitable"> elements.
- The table with the most rows was prioritized, as it generally contained the main list of institutions.
- The HTML for the selected table was then parsed into a pandas DataFrame using pd.read_html() for structured extraction.

Column Identification:

- Because Wikipedia's tables vary slightly between states, the scraper dynamically identified the correct column by testing for one of the following headers:
  ['School', 'Institution', 'University', 'College', 'Name', 0].

- The first valid match was used to extract college names.

Data Cleaning:

- Bracketed citation markers (e.g., [1], [a]) were removed from all names using string operations.

- Whitespace and special characters were stripped.

- Each entry was stored with its corresponding state for context.

Politeness and Retry Policy:

- All HTTP requests were made with a realistic browser User-Agent header to avoid being flagged as a bot.

- Each page was fetched up to three times in case of network or timeout errors.

- Randomized polite delays (time.sleep(1 + random.random())) were applied to prevent IP throttling.

4. Error Handling

- **Retry Logic:** Each HTTP request included up to three retries with incremental delays to handle temporary connectivity issues.

- **Graceful Failures:** States that could not be fetched or parsed after three attempts were skipped with a warning message, allowing the script to continue without interruption.

- **Fallback Safety:** If no valid table was found on a page, the scraper safely returned an empty list for that state rather than terminating execution.

5. Output

- **File Output:** us_colleges.csv, us_colleges_clean.csv

Column	Description
State	The U.S. state where the institution is located

College Name	Official name of the college or university as listed on Wikipedia

Entries were cleaned and deduplicated based on College Name + State, ensuring only one standardized instance of each institution exists.

Course table

1. Objective and Justification

The goal was to build a standardized database of higher education course identifiers and descriptions. This was done by scraping the course catalog of a single institution, the University of Southern Maine (USM), to establish a controlled, limited scope of authentic, real-world course data for system testing and demonstration purposes.

The scraper uses Python libraries such as requests, BeautifulSoup, and pandas to navigate, parse, and structure the course data. The output is stored in a CSV file (usm_courses.csv) within a data directory for further integration.

2. Methodology and Tools

The data collection was executed using a Python script (documented in scraping_courses.ipynb) utilizing the following libraries:

Tool	Purpose
Python requests	Fetched the course catalog's pagination pages.
Python BeautifulSoup	Parsed the HTML of each catalog page to locate and extract course tables.
Python pandas	Consolidated the data, removed duplicates, and managed the final CSV output.

3. Data Source and Scraping Strategy

1. **Data Source**

The source was the University of Southern Maine's (USM) public online course catalog with the base URL as https://catalog.usm.maine.edu/content.php. The content type was dynamic HTML pages displaying course listings filtered through query parameters (e.g, filter[cpage] for pagination).

2. Scraping Strategy

Pagination Detection:

- To determine the total number of catalog pages, the scraper first probed a known content page (Page 2).
- Using a regular expression search on strings containing the pattern "Page X of Y", it dynamically extracted the total page count (Y).
- If pagination text could not be located, the program defaulted to 18 pages as a safe fallback.

Iterative Extraction:

- For each page, the scraper constructed HTTP GET requests with the appropriate query parameters.
- It then parsed the resulting HTML to find all <a> tags whose href contained the substring "preview_course_nopop.php", indicating valid course entries.
- The visible text of each link (formatted as "CODE - COURSE NAME") was split into course_code and course_name.
- Each record was appended to a list with the associated course_link and page number for traceability.

Human Validation via Inspection:

- Prior to coding, multiple pages were manually inspected using Chrome DevTools to understand structural patterns and ensure the consistency of link and text placement.
- It was confirmed that the structure of <a> tags and link patterns remained consistent across all catalog pages.

Politeness Policy:

- A 0.5-second delay (time.sleep(0.5)) between requests was implemented to minimize load on the USM server and comply with polite scraping norms.

4. Error Handling

- **Request Exceptions:** Every HTTP call was wrapped in a try-except block to catch network errors and unexpected responses (RequestException).

- **Pagination Failures:** If a page failed to load or no courses were found mid-sequence, the scraper automatically stopped pagination to avoid redundant requests.

- **Fallback Safety:** Default values were used for both total page count and missing course metadata to ensure uninterrupted execution and dataset completeness.

5. Output

- **File Output:** usm_courses.csv

Column	Description
course_code	Official course (e.g., COS 170)
course_name	Full course title as listed in the catalog (e.g., Structured Programming Lab
course_link	URL to the individual course details page
page_number	Catalog page number from which the course was extracted

Duplicate entries were removed based on the combined fields course_code + course_name to ensure data uniqueness.

Resources Table

1. Objective and Justification

The purpose of this scraping module was to populate the studyBuddy Hub resource table with verified, structured learning materials relevant to specific courses in the form of links, articles, pdfs, videos, unit overviews, etc.

Crucial Strategy Shift: The initial approach using broad web crawling was discontinued due to the inconsistent quality and limited relevance of the retrieved content. Instead, a more focused strategy was adopted—targeting specific, pre-vetted educational websites to ensure that all collected resources are accurate and directly map to course content.

2. Methodology and Tools

The data collection was executed using a Python script (documented in scraping_resources.ipynb) utilizing the following libraries:

Tool	Purpose
Python requests	Sent HTTP requests and retrieved HTML page content from the target websites.
BeautifulSoup (bs4)	Parsed HTML and extracted specific elements like titles (<h1>/<b>), descriptions (<p>), and URLs (href).
pandas	Structured the extracted data into tabular form, handled duplicates, standardized column names, and appended records to the master CSV file (course_resources_scraped.csv).

3. Data Source and Scraping Strategy

The scraping process was designed to collect structured, course-specific educational resources from diverse and reputable open-access learning sites. Each source required manual inspection of its HTML structure to identify consistent tag hierarchies and reliable selectors for extracting topics, descriptions, and URLs. All scripts were implemented in Python using the requests, BeautifulSoup, and pandas libraries, with outputs consolidated into a unified CSV file (course_resources_scraped.csv) for integration into the resource table.

1. GeeksforGeeks – Data Structures Section

Purpose: Extract structured computer science topics with corresponding short descriptions and resource URLs.

URLs:
https://www.geeksforgeeks.org/dsa/introduction-to-data-structures/

https://www.geeksforgeeks.org/dsa/introduction-to-algorithms/

https://www.geeksforgeeks.org/web-tech/web-technology/

https://www.montclair.edu/computer-science-education/wp-content/uploads/sites/253/2024/02/3-5-8.1.5-Algorithms-Programming.pdf

Inspection Process:

- Used Chrome DevTools to inspect the structure of each page and verify the location of core tags such as <a rel="noopener">, <h2>, and <p dir="ltr">.

- Identified consistent placement of bolded topic titles inside <a> elements and accompanying text in adjacent <p> tags.

Scraping Logic:

- Parsed HTML using BeautifulSoup and iterated through the detected topic links.

- Extracted the topic name, description, and hyperlink.

- Applied error handling to skip missing or malformed entries.

Data Labeling:

- Each record was stored with ResourceType = 'LINK' and tagged to its corresponding CourseName (e.g., *Data Structures*, *Algorithms in Programming*, *Web Applications Development*).

- Additional PDF entries (e.g., *Algorithms Programming Concepts*) were manually appended to ensure coverage of offline resources.

2. Khan Academy – Biology Units

Purpose: Retrieve unit titles and summaries for biological sciences courses.

URLs: https://www.khanacademy.org/science/ap-biology/chemistry-of-life

https://www.khanacademy.org/science/ap-biology/cell-structure-and-function
https://www.khanacademy.org/science/ap-biology/cellular-energetics
https://www.khanacademy.org/science/ap-biology/cell-communication-and-cell-cycle
https://www.khanacademy.org/science/ap-biology/heredity
https://www.khanacademy.org/science/ap-biology/gene-expression-and-regulation
https://www.khanacademy.org/science/ap-biology/natural-selection
https://www.khanacademy.org/science/ap-biology/ecology-ap

https://www.khanacademy.org/science/ap-biology/x16acb03e699817e9:simulations
https://www.khanacademy.org/science/ap-biology/worked-examples-ap-biology
https://www.khanacademy.org/science/ap-biology/ap-biology-standards-mappings

Inspection Process:

- Reviewed the DOM hierarchy to confirm that unit identifiers were embedded in the URL slug rather than as structured headings.
- Verified that summary text appeared within containers matching patterns like main-content or unit-description.

Scraping Logic:

- Extracted TopicName directly from URL segments using regular expressions to transform slugs into readable titles (e.g., /cellular-energetics → Cellular Energetics).
- Could not retrieve the first meaningfulsentence of the page.

Validation:

- Used regular expressions to clean unwanted characters from titles.
- Applied checks for duplicates and malformed text.

3. W3Schools (Python & Java Tutorials)

Purpose: Build comprehensive topic lists for programming languages.

URLs: https://www.w3schools.com/java/

https://www.w3schools.com/python/

Inspection Process:

- Inspected the sidebar navigation (id="leftmenuinnerinner") to locate the full set of tutorial links.

- Confirmed consistent structural patterns across both Python and Java tutorials.

Scraping Logic:

- Collected all .asp links from the navigation menu using urljoin to form absolute URLs.
- For each page, extracted the main <h1> title and the first paragraph as a description snippet.
- Used regular expressions to isolate the first full sentence for clarity.

Data Validation:

- Ensured every entry contained a valid title, snippet, and resource URL.
- Added a short delay (time.sleep()) between requests as a politeness measure.

4. Criminal Law (Multi-Source Scrape)

Purpose: Gather legal resource summaries from government, academic, and MOOC sites.

Sources:

- Maine Legislature: https://legislature.maine.gov/statutes/17-a/title17-ach0sec0.html
- American Public University System (APUS): https://www.apu.apus.edu/area-of-study/security-and-global-studies/resources/what-is-criminal-law-and-why-does-it-matter/
- edX: https://www.edx.org/learn/criminal-law

Inspection Process:

- Inspected each site individually due to structural variation.
- Located heading tags (<h1>, <h2>) and primary content containers (<article>, <main>, .entry-content, .main-content).

Scraping Logic:

- Extracted titles from visible headers or <title> tags as a fallback.
- Captured the first descriptive paragraph using regex to isolate the initial complete sentence.
- For edX, incorporated meta descriptions when available for cleaner summaries.

4. Error Handling and Validation Techniques

1. Network Errors:

All requests were wrapped in a try-except block to catch timeout, connection, or 404 errors.

Example:

```
 try:
    response = requests.get(url, headers=HEADERS, timeout=10)
    response.raise_for_status()
except requests.exceptions.RequestException as e:
    print(f"   * ERROR: Failed to fetch {url}. Reason: {e}")
```

2. Data Validation:

- Ignored links with missing text or invalid URLs.
- Filtered duplicates using Pandas .drop_duplicates().
- Checked snippet lengths and trimmed overly long descriptions to 250-500 characters.

3. File Validation:

- Ensured all output columns matched the schema:
  ['CourseName', 'TopicName', 'ResourceURL', 'ContentSnippet', 'ResourceType'].

Validated CSV creation only if data existed:

```
 if not df.empty:
    df.to_csv("course_resources_scraped.csv", index=False)
```

5. Output

Each individual script appended its results to a shared CSV file (course_resources_scraped.csv), creating a cumulative dataset of cross-disciplinary educational resources.

Column	Description
CourseName	Name of the associated course (e.g., “Data Structures”)
TopicName	Topic title extracted from the source page
ResourceURL	Direct link to the resource
ContentSnippet	Short description of the topic
ResourceType	Type of resource (LINK, ARTICLE, or VIDEO)

Duplicates were removed based on CourseName + TopicName + ResourceURL.

Data Cleaning

usm_courses

- Compared the fields in the “courses” table, to the existing fields in “usm_courses”.
- Created a new data frame with fields that matched that of the “courses” table, so that the data from “usm_courses” could fit in it.
    - Fields: course_code, course_name, college_id
    - The field for the primary key (course_id) is unnecessary, since the field is auto-incrementing, and will be added automatically later via SQL indexing.

1. Algorithm For Cleaning

- For each row (ie, usm course) in the CSV file, I did the following:
    - We wanted the course abbreviation concatenated with the course catalog number to be the “course code”.
    - Since every course name had the course abbreviation, and the catalog number as the first 7 characters, followed by 2 more unnecessary characters, I needed to remove those first 9 characters from every course name.
        - Before I did so, I first took the first 7 characters, and excluded the 4th character, which was white space, and saved that as the row’s course code.
    - I then took every character in the course name from the 10th character onwards, and saved that as the row’s new course name.

2. Saving Results

- At the end of each iteration in the for loop, I added the following values to a new row in the new courses dataframe:
    - The value of course_code was the course code previously saved earlier in the loop.
    - The value of course_name was the new course name previously saved earlier in the loop.
    - The value of college_id was set to null, since the college_id field in the schema was a nullable field.

us_colleges

- Mostly a repeat of the data cleaning process for “usm_courses”, minus the different fields.
- Compared the fields in the “colleges” table, to the existing fields in “us_colleges”.
- Created a new data frame with fields that matched that of the “courses” table, so that the data from “us_colleges” could fit in it.
    - Fields: course_code, course_name, college_id
    - The field for the primary key (college_id) is unnecessary, since the field is auto-incrementing, and will be added automatically later via SQL indexing.

1. Algorithm For Cleaning

- For each row (ie, a US college) in the CSV file, I did the following:
    - Take the name of the college.
    - If the name of the college had any commas in it, replace it with an empty character.
        - This was to prevent truncation errors later on during indexing.
    - Save the new college name with no commas.

2. Saving Results

- At the end of each iteration in the for loop, I added the following values to a new row in the new colleges dataframe:
    - The value of college_name was the name of the college, updated to remove any commas.

quiz_data

- Compared the fields in the “question” table, and the “answer” table to the existing fields in “quiz_data”

- Created two new data frames, one for question, and one for answer, both with the same fields as their respective tables (ie,
    - Question: question_id, quiz_id, question_text, question_type, points
    - Answer: answer_id, question_id, answer_text, is_correct

1. Questions Data Frame

Algorithm For Cleaning / Saving Results

- First dealt with the **questions** data frame.
- I used a for loop to loop through all of the rows of the raw quiz data, and created a new row in the questions data frame to save all of the new values for each iteration.
    - The value of question_id is auto incrementing, so the value would just be the iteration i in the for loop.
    - quiz_id is 0 by default, since all these questions are part of the same quiz.
    - question_text is the same as the “question” field in “quiz_data”.
    - The question_type for all questions is “multiple_choice”, since the given data only includes multiple choice questions.
    - “points” has a value of 1 by default.

2. Answers Data Frame

Algorithm For Cleaning

- Next, dealt with the **answers** data frame.
    - Since the answers were multiple choice, I needed a way to separate, and clean all the individual answers in the “Answer” field of “quiz_data”. I created a helper method to do this.
        - The entire answer field was surrounded by square brackets, so I needed to remove those.
        - Every individual answer was surrounded by an apostrophe, followed by a comma, so I would also need to remove those.
        - To do this, I created a list called “answerCharacters” that would store only the characters of the actual answer, which would then be joined into a single string.
        - To exclude square brackets, I simply made the for loop continue with the loop if it encountered any.
        - To exclude commas, it would only store the characters in the answer that were non-commas.
        - To exclude apostrophes, and to make sure every individual answer was separated, I made a variable keeping track of the apostrophe count. If the count ever reached 2, we know that we already have the full answer

stored. It would then join all the characters in “answerCharacters”, and append it to the list of answers.
        - For subsequent iterations, I made sure to clear the apostrophe count, and the “answerCharacters” list.
        - Finally, this method returns a list of all the different answers.
    - I created a nested for loop to index all the different answers. The outer for loop is to iterate through a range equal to the number of questions, and the inner for loop is to iterate through all the answers of the question.
        - This way, I’m able to store all the answers as individual rows, and create auto-incrementing unique ids for each answer, while at the same time making sure that all of the answers’ question_id match up with their respective question’s id.

3. Saving Results

- Finally, after the end of each iteration, I add a new row at the end of the new answers dataframe, and add the values for each field in the row.
    - The value of answer_id was the current iteration of the inner for loop.
    - The value of question_id was the current iteration of the outer for loop.
    - The value of answer_text was the value of the current iteration of the inner for loop (ie, the current answer’s text).
    - is_correct was set to 0 (ie, false) by default.
        - While this marks every answer as incorrect, there wasn’t really a way to automatically, and correctly mark each question as right or wrong. At least, not without using AI, which was beyond the scope of the project.
        - The reason I set everything to false instead of true, is that every multiple choice question presumably only had one correct answer, so at least this way, 3/4ths of the answers are correctly marked as false.

course_resources

1. Objectives and Justification

- This cleaner takes a raw CSV of scraped data of course-related links and turns it into:
    - A cleaned CSV
    - A MySQL Insert script to populate the Resource table
- The goal of this is to:
    - Normalize Messy URLs
        - Base64 encoded links
        - Bing redirect wrappers
    - Generate readable titles
    - Place the resource under the correct filetype

    - Guarantee that every inserted row has consistent, valid data that matches the schema

2. Methodology & Tools

- Pandas
    - Loading CSVs, building cleaned DataFrames, and exporting a cleaned CSV
- os
- re
    - Detects base64 blobs and patterns in URLs/titles
- base64
    - Decodes base64 urls
- Urllib.parse
    - Parsing and unwrapping redirects URLs
- The python program expects an input CSV that went through the resource scraping process. The CSV should include:
    - A query or original search string used to find the resource
    - A raw title or label
    - A raw URL
- CSV is loaded with:
    - **df = pd.read_csv(INPUT_CSV, dtype=str, keep_default_na=False)**
        - Ensures all empty fields remain empty
        - All values are treated as strings

3. Cleaning Algorithm

- The cleaning logic has three main tasks:
    - Make URLs usable
    - Generate readable titles
    - Infer file types
- strip_weird(s: str)
    - Replaces non-breaking spaces such as \u00a0 and special characters like “>” with regular spaces
    - Strips leading and trailing white space
    - Multiple spaces in a row are converted to a single space
- URL Normalization
    - Raw CSV contained URLs that were wrapped in Bing redirect links
    - Instead of http or https URLs, there were many weird base64 blobs
    - Maybe_b64_url(s: str)
        - If string starts with http or https, it remains the same
        - If the string does not, it searches for base64 substrings
        - Converts “-” to “+” and “_” to “/” to make substring base64 compatible
        - Looks for “aHR0” which is base64 for http

    - unwrap_bing(u: str)
        - Parses URL
        - If a host ends with bing.com and starts with /ck/
            - Pulls the URL from the u query
            - Unquotes it
            - Passes through maybe_b64_url
    - normalize_url(u: str)
        - Calls the strip_weird function
        - If URL contains bing, calls unwrap_bing
        - It calls maybe_b64_url
        - Returns a clean url
- Title Normalization
    - Many scraped titles were not understandable or did not make sense
        - Examples of this include “home” and “login” or domains like “default”
    - title_from_url(u: str)
        - Takes a basename or the URL path
        - Strips the extension from the URL
        - Replaces “-” and “_” with spaces
    - clean_title(raw_title: str, url: str)
        - Calls strip_weird on the title
        - Removes leading http or https
        - A set of titles that should not be used as titles are defined
            - If the cleaned title is empty, less than 3 chars, just a domain or in this set of titles it uses url_fallback
    - url_ballback
        - Uses a non-empty URL path if it is available and follows a <host>/ template
        - Applies .title() with length limits
            - Examples include:
                - www.britannica.com / anthropology
                - mexico.internationaltrucks.com / camiones de carga
- Filetype inference
    - filetype_from_url(u: str)
        - Parses the URL path and extracts the file extension
        - Uses a map of known extensions
        - If the extension isn't found on the map, it defaults to “LINK”

4. Cleaned DataFrame

- After the data has been cleaned, the python program will build a new DataFrame
- Cdf = pd.DataFrame
    - Description is left blank for now (looking to improve upon this)
    - Title, filetype, and source are ready to load
- Drops any row where both the title and source are empty

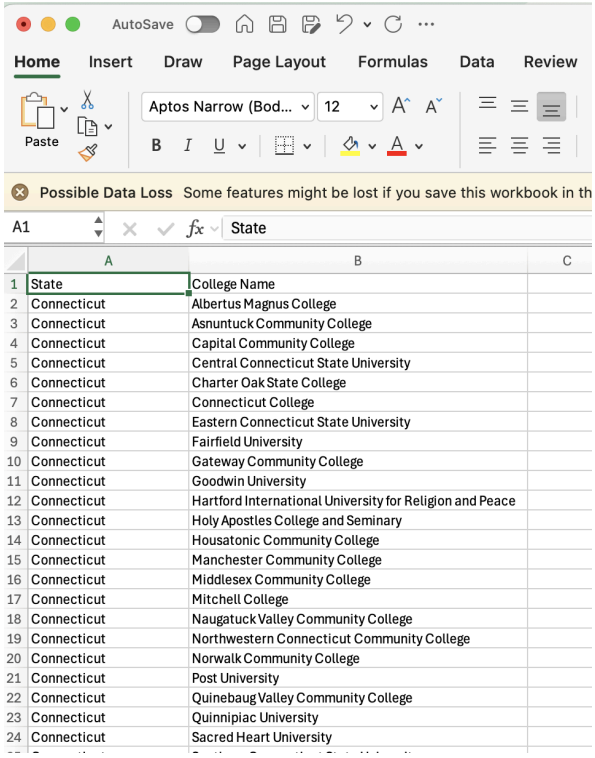
- Logs row count before and after to verify how many rows were removed
- SQL Building
    - Once the DataFrame is ready, a SQL script is generated
    - Starts with a f'SET @uploader_id := {DEFAULT_UPLOADER_ID}
    - For each cleaned row it generates:
        - INSERT INTO Resource (uploader_id, title, description, filetype, source)
        - VALUES (@uploader_id, '<title>', '<description>', '<filetype>', '<source>');
    - Handles quotes inside strings by replacing " ' " with " '' " so every value has its single quotes doubled
- Not a part of the Python Program, but added after is a seeder user and uploader_id binding
    - Since Resouce.uploader_id is a foreign key:
        - FOREIGN KEY (uploader_id) REFERENCES Users(user_id)
            - Ensures a system user exists and binds uploader_id to it
        - The dedicated system user is:
            - User_id = 1001
            - Email = resources@system.local
            - Name: System Seeder

5. Validation and Output

- To ensure that the cleaning and SQL generation worked:
    - Verified that the resource table in study buddy matched the columns created by the python program
    - Compared raw_count to the number of rows in course_resources_cleaned.csv
    - Reviewed titles and URLs to confirm they were readable and usable
    - Ran generated insert_resources.sql against the StudyBuddy schema and confirmed successful inserts

Appendix

us_colleges [Click or Hover Link to Data]



AutoSave

Home Insert Draw Page Layout Formulas Data Review

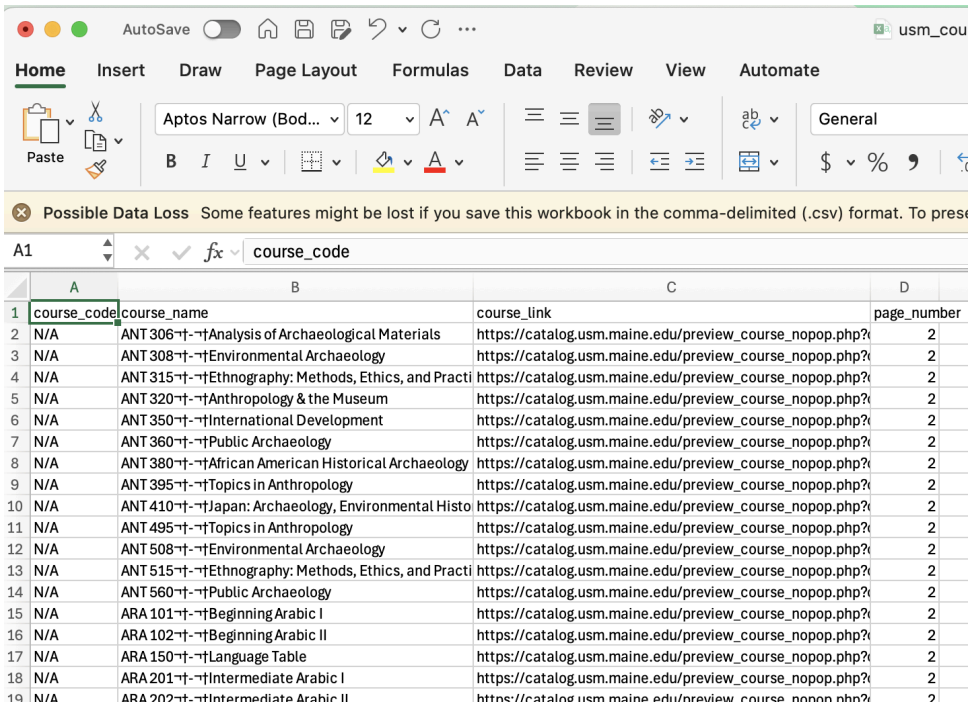
Paste | Aptos Narrow (Bod... | 12

Possible Data Loss Some features might be lost if you save this workbook in th

A1 | State

	A	B	C
1	State	College Name	
2	Connecticut	Albertus Magnus College	
3	Connecticut	Asnuntuck Community College	
4	Connecticut	Capital Community College	
5	Connecticut	Central Connecticut State University	
6	Connecticut	Charter Oak State College	
7	Connecticut	Connecticut College	
8	Connecticut	Eastern Connecticut State University	
9	Connecticut	Fairfield University	
10	Connecticut	Gateway Community College	
11	Connecticut	Goodwin University	
12	Connecticut	Hartford International University for Religion and Peace	
13	Connecticut	Holy Apostles College and Seminary	
14	Connecticut	Housatonic Community College	
15	Connecticut	Manchester Community College	
16	Connecticut	Middlesex Community College	
17	Connecticut	Mitchell College	
18	Connecticut	Naugatuck Valley Community College	
19	Connecticut	Northwestern Connecticut Community College	
20	Connecticut	Norwalk Community College	
21	Connecticut	Post University	
22	Connecticut	Quinebaug Valley Community College	
23	Connecticut	Quinnipiac University	
24	Connecticut	Sacred Heart University	

usm_courses [Click or Hover Link to Data]



AutoSave usm_cou

Home Insert Draw Page Layout Formulas Data Review View Automate

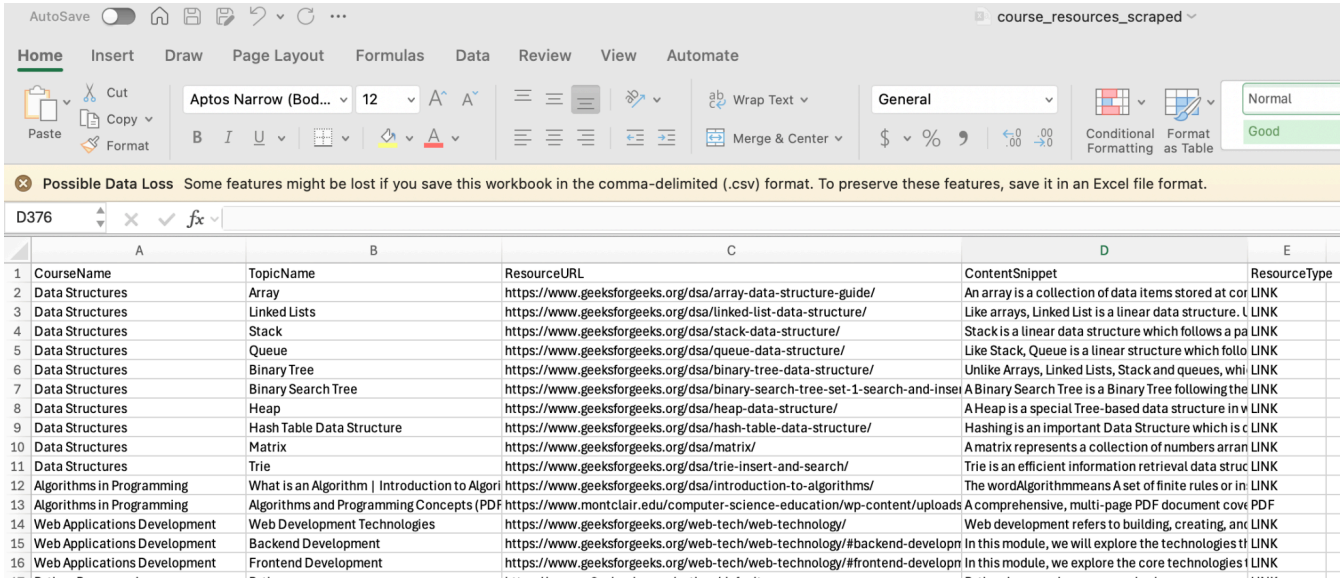
Paste | Aptos Narrow (Bod... | 12 | General

Possible Data Loss Some features might be lost if you save this workbook in the comma-delimited (.csv) format. To prese

A1 | course_code

	A	B	C	D
1	course_code	course_name	course_link	page_number
2	N/A	ANT 306¬†-¬†Analysis of Archaeological Materials	https://catalog.usm.maine.edu/preview_course_nopop.php?(	2
3	N/A	ANT 308¬†-¬†Environmental Archaeology	https://catalog.usm.maine.edu/preview_course_nopop.php?(	2
4	N/A	ANT 315¬†-¬†Ethnography: Methods, Ethics, and Practi	https://catalog.usm.maine.edu/preview_course_nopop.php?(	2
5	N/A	ANT 320¬†-¬†Anthropology & the Museum	https://catalog.usm.maine.edu/preview_course_nopop.php?(	2
6	N/A	ANT 350¬†-¬†International Development	https://catalog.usm.maine.edu/preview_course_nopop.php?(	2
7	N/A	ANT 360¬†-¬†Public Archaeology	https://catalog.usm.maine.edu/preview_course_nopop.php?(	2
8	N/A	ANT 380¬†-¬†African American Historical Archaeology	https://catalog.usm.maine.edu/preview_course_nopop.php?(	2
9	N/A	ANT 395¬†-¬†Topics in Anthropology	https://catalog.usm.maine.edu/preview_course_nopop.php?(	2
10	N/A	ANT 410¬†-¬†Japan: Archaeology, Environmental Histo	https://catalog.usm.maine.edu/preview_course_nopop.php?(	2
11	N/A	ANT 495¬†-¬†Topics in Anthropology	https://catalog.usm.maine.edu/preview_course_nopop.php?(	2
12	N/A	ANT 508¬†-¬†Environmental Archaeology	https://catalog.usm.maine.edu/preview_course_nopop.php?(	2
13	N/A	ANT 515¬†-¬†Ethnography: Methods, Ethics, and Practi	https://catalog.usm.maine.edu/preview_course_nopop.php?(	2
14	N/A	ANT 560¬†-¬†Public Archaeology	https://catalog.usm.maine.edu/preview_course_nopop.php?(	2
15	N/A	ARA 101¬†-¬†Beginning Arabic I	https://catalog.usm.maine.edu/preview_course_nopop.php?(	2
16	N/A	ARA 102¬†-¬†Beginning Arabic II	https://catalog.usm.maine.edu/preview_course_nopop.php?(	2
17	N/A	ARA 150¬†-¬†Language Table	https://catalog.usm.maine.edu/preview_course_nopop.php?(	2
18	N/A	ARA 201¬†-¬†Intermediate Arabic I	https://catalog.usm.maine.edu/preview_course_nopop.php?(	2
19	N/A	ARA 202¬†-¬†Intermediate Arabic II	https://catalog.usm.maine.edu/preview_course_nopop.php?(	2

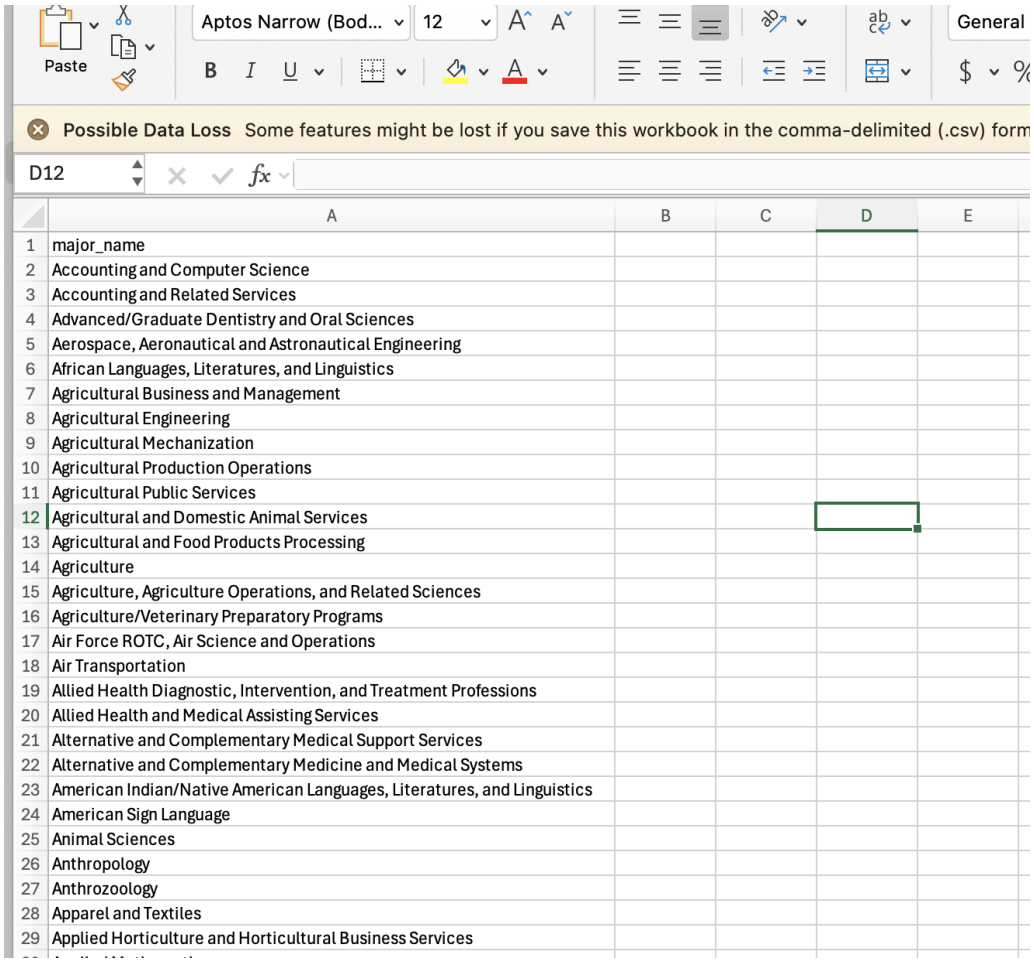
course_resources_scraped [Click or Hover Link to Data]



Possible Data Loss Some features might be lost if you save this workbook in the comma-delimited (.csv) format. To preserve these features, save it in an Excel file format.

	A	B	C	D	E
1	CourseName	TopicName	ResourceURL	ContentSnippet	ResourceType
2	Data Structures	Array	https://www.geeksforgeeks.org/dsa/array-data-structure-guide/	An array is a collection of data items stored at cor	LINK
3	Data Structures	Linked Lists	https://www.geeksforgeeks.org/dsa/linked-list-data-structure/	Like arrays, Linked List is a linear data structure. U	LINK
4	Data Structures	Stack	https://www.geeksforgeeks.org/dsa/stack-data-structure/	Stack is a linear data structure which follows a pa	LINK
5	Data Structures	Queue	https://www.geeksforgeeks.org/dsa/queue-data-structure/	Like Stack, Queue is a linear structure which follo	LINK
6	Data Structures	Binary Tree	https://www.geeksforgeeks.org/dsa/binary-tree-data-structure/	Unlike Arrays, Linked Lists, Stack and queues, whi	LINK
7	Data Structures	Binary Search Tree	https://www.geeksforgeeks.org/dsa/binary-search-tree-set-1-search-and-inse	A Binary Search Tree is a Binary Tree following the	LINK
8	Data Structures	Heap	https://www.geeksforgeeks.org/dsa/heap-data-structure/	A Heap is a special Tree-based data structure in w	LINK
9	Data Structures	Hash Table Data Structure	https://www.geeksforgeeks.org/dsa/hash-table-data-structure/	Hashing is an important Data Structure which is c	LINK
10	Data Structures	Matrix	https://www.geeksforgeeks.org/dsa/matrix/	A matrix represents a collection of numbers arran	LINK
11	Data Structures	Trie	https://www.geeksforgeeks.org/dsa/trie-insert-and-search/	Trie is an efficient information retrieval data struc	LINK
12	Algorithms in Programming	What is an Algorithm | Introduction to Algori	https://www.geeksforgeeks.org/dsa/introduction-to-algorithms/	The wordAlgorithmmeans A set of finite rules or in	LINK
13	Algorithms in Programming	Algorithms and Programming Concepts (PDF	https://www.montclair.edu/computer-science-education/wp-content/uploads	A comprehensive, multi-page PDF document cove	PDF
14	Web Applications Development	Web Development Technologies	https://www.geeksforgeeks.org/web-tech/web-technology/	Web development refers to building, creating, and	LINK
15	Web Applications Development	Backend Development	https://www.geeksforgeeks.org/web-tech/web-technology/#backend-developn	In this module, we will explore the technologies th	LINK
16	Web Applications Development	Frontend Development	https://www.geeksforgeeks.org/web-tech/web-technology/#frontend-developn	In this module, we explore the core technologies t	LINK

us_majors.csv



Possible Data Loss Some features might be lost if you save this workbook in the comma-delimited (.csv) form

	A
1	major_name
2	Accounting and Computer Science
3	Accounting and Related Services
4	Advanced/Graduate Dentistry and Oral Sciences
5	Aerospace, Aeronautical and Astronautical Engineering
6	African Languages, Literatures, and Linguistics
7	Agricultural Business and Management
8	Agricultural Engineering
9	Agricultural Mechanization
10	Agricultural Production Operations
11	Agricultural Public Services
12	Agricultural and Domestic Animal Services
13	Agricultural and Food Products Processing
14	Agriculture
15	Agriculture, Agriculture Operations, and Related Sciences
16	Agriculture/Veterinary Preparatory Programs
17	Air Force ROTC, Air Science and Operations
18	Air Transportation
19	Allied Health Diagnostic, Intervention, and Treatment Professions
20	Allied Health and Medical Assisting Services
21	Alternative and Complementary Medical Support Services
22	Alternative and Complementary Medicine and Medical Systems
23	American Indian/Native American Languages, Literatures, and Linguistics
24	American Sign Language
25	Animal Sciences
26	Anthropology
27	Anthrozoology
28	Apparel and Textiles
29	Applied Horticulture and Horticultural Business Services