WEB SCRAPING PROJECT ON 'IT VEDANT' WEBSITE

10 October 2023 22:30

IT VEDANT 2023

SUBMITTED TO:

Mr. Sameer Warsolkar
Branch: Thane(IT Vedant)

SUBMITTED BY:

Akshada Jarode
Course Name: Data science and analytics with AI
Branch: Thane
Batch: 3-5pm
Mobile: 7715859214
Email: akshada,jarode@gmail.com

CONTENTS:

INTRODUCTION

Welcome to my web scraping project focused on 'IT Vedant,' a website dedicated to technology enthusiasts. In this project, I will harness the power of web scraping techniques, leverage the Pandas library for data manipulation, and store our extracted information in CSV files. By doing so, I aim to gather valuable insights and data from 'IT Vedant,' making it accessible and usable for various analytical purposes. Let's embark on this journey to explore and utilize the rich content offered by 'IT Vedant' for our data-driven endeavors.

DESCRIPTION

In this project, Python will serve as our trusted conduit to access the 'IT Vedant' website, initiating HTTP GET requests to navigate its digital landscape. Our mission is multifaceted: we shall first curate an extensive collection of links strewn across the website, akin to harvesting ripe digital fruit from the virtual vineyard. Next, our discerning script shall unravel the textual tapestry, capturing headers that adorn the pages. It shall then proceed to assemble a gallery of images, each a visual artifact of the web.

But the task is not done; our script is programmed to be particular. It will sift through the textual labyrinth, retrieving only those links that bear a specific insignia. It shall uncover orderly lists, both ordered ('ol') and unordered ('ul'), revealing structured information hidden within. And, of course, it shall diligently catalog images that bear the mark of alt text.

As the data flows in, Pandas, our trusty Python library, shall take the helm. It will orchestrate a symphony of sorting, arranging, and harmonizing, transforming raw data into an organized composition. Finally, as the digital tapestry is woven together, we shall immortalize our findings in the form of a CSV file—a snapshot of our journey through the 'IT Vedant' web, capturing its essence in structured data.

PROJECT STEPS

Import Necessary Libraries:

Description: This code snippet serves as an initial setup for a web scraping project in Python. It begins by importing two essential libraries: 'requests' and 'beautifulsoup4,' both of which are essential tools for web

scraping tasks.

1. Requests Library: 'Requests' is a popular Python library used for making HTTP requests to web pages. In this context, it allows us to send HTTP GET requests to the target website, retrieve its HTML content, and prepare it for parsing.

2. Beautiful Soup Library: 'Beautiful Soup' (often imported as 'beautifulsoup4') is a Python library that simplifies the process of parsing HTML and XML documents. It enables us to navigate and extract data from the retrieved HTML content with ease.

By installing these libraries using 'pip,' we ensure that we have the necessary tools to begin our web scraping project. With 'requests' for fetching web pages and 'Beautiful Soup' for parsing and extracting data, we can dive into the world of web scraping, making it easier to gather and analyze information from websites.

Import Necessary Libraries:

```
In [1]: pip install requests beautifulsoup4

Requirement already satisfied: requests in c:\users\administrator\anaconda34\lib
\site-packages (2.31.0)
Requirement already satisfied: beautifulsoup4 in c:\users\administrator\anaconda
34\lib\site-packages (4.12.2)
Requirement already satisfied: charset-normalizer<4,>=2 in c:\users\administrato
r\anaconda34\lib\site-packages (from requests) (2.0.4)
Requirement already satisfied: idna<4,>=2.5 in c:\users\administrator\anaconda34
\lib\site-packages (from requests) (3.4)
Requirement already satisfied: urllib3<3,>=1.21.1 in c:\users\administrator\anac
onda34\lib\site-packages (from requests) (1.26.16)
Requirement already satisfied: certifi>=2017.4.17 in c:\users\administrator\anac
onda34\lib\site-packages (from requests) (2023.7.22)
Requirement already satisfied: soupsieve>1.2 in c:\users\administrator\anaconda3
4\lib\site-packages (from beautifulsoup4) (2.4)
Note: you may need to restart the kernel to use updated packages.
```

Web Scraping Code:

Description: This Python code snippet is designed for web scraping, and it demonstrates how to send an HTTP GET request to a specified website and then parse its HTML content. The code uses the 'requests' library to make the HTTP request and the 'Beautiful Soup' library for HTML parsing.

Here's a breakdown of the code's functionality:

1. URL Definition: The code starts by defining the URL of the website you want to scrape. In this case, it's set to "https://www.itvedant.com/."

2. HTTP GET Request: The 'requests.get(url)' line sends an HTTP GET request to the specified URL, and the response is stored in the 'response' variable.

3. Check Response Status: The code checks if the request was successful by examining the HTTP status code. A status code of 200 indicates success, while other codes signify errors.

4. HTML Parsing: If the request is successful (status code 200), the code proceeds to parse the HTML content using 'Beautiful Soup.' The parsed content is stored in the 'soup' variable.

5. Page Title Extraction: The page title is extracted from the parsed HTML and stored in the 'page_title' variable. It then prints the page title to the console.

6. Link Extraction: The code extracts and prints specific elements, such as all the links ('<a>' tags) on the page. It iterates through the list of links and prints their 'href' attributes.

7. Error Handling: If the request is not successful (status code other than 200), it prints an error message along with the HTTP status code.

This code serves as a starting point for web scraping, enabling you to retrieve and extract information from web pages, such as page titles and links. It showcases the fundamental steps involved in web scraping using Python

and the 'requests' and 'Beautiful Soup' libraries.

```
In [2]: import requests
        from bs4 import BeautifulSoup

        # URL of the website you want to scrape
        url = "https://www.itvedant.com/"

        # Send an HTTP GET request to the website
        response = requests.get(url)

        # Check if the request was successful (status code 200)
        if response.status_code == 200:
            # Parse the HTML content
            soup = BeautifulSoup(response.text, "html.parser")

            # Extract and print the page title
            page_title = soup.title.string
            print("Page Title:", page_title)

            # Extract and print specific elements, e.g., all links
            links = soup.find_all("a")
            for link in links:
                print("Link:", link.get("href"))

        else:
            print("Failed to retrieve the page. Status code:", response.status_code)
```

Output

```
Page Title: Upgrade Your Coding Skills with the Best IT Courses in Pune
Link: https://www.itvedant.com
Link: #
Link: #
Link: https://www.itvedant.com/data-software-engineering
Link: https://www.itvedant.com/full-stack-developer
Link: https://www.itvedant.com/aws-azure-cloud-technology
Link: https://www.itvedant.com/data-science-and-ai
Link: #
Link: https://www.itvedant.com/data-software-engineering
Link: https://www.itvedant.com/full-stack-developer
Link: https://www.itvedant.com/aws-azure-cloud-technology
Link: #
Link: https://www.itvedant.com/data-science-and-ai
Link: tel:9205004404
Link: None
Link: #
Link: https://www.itvedant.com/data-software-engineering
Link: https://www.itvedant.com/full-stack-developer
Link: https://www.itvedant.com/aws-azure-cloud-technology
Link: https://www.itvedant.com/data-science-and-ai
Link: None
Link: https://www.itvedant.com/placements
Link: https://www.itvedant.com/careers
Link: https://www.itvedant.com/enrol
Link: https://www.itvedant.com/refer-and-earn
Link: https://www.itvedant.com/news-and-events
Link: https://www.itvedant.com/success-stories
Link: https://www.itvedant.com/corporate-training
Link: https://www.itvedant.com/blog
Link: https://www.itvedant.com/contact
Link: tel:9205004404
Link: #
Link: #
Link: #
Link: #home
Link: #menu1
Link: https://www.itvedant.com/data-software-engineering
Link: https://www.itvedant.com/full-stack-developer
Link: https://www.itvedant.com/aws-azure-cloud-technology
Link: https://www.itvedant.com/data-science-and-ai
Link: https://www.itvedant.com/data-software-engineering
Link: https://www.itvedant.com/full-stack-developer
```


Extract Text Content:

Description: This Python code snippet demonstrates how to extract text content from specific HTML elements (in this case, headings) on a webpage using BeautifulSoup, a powerful library for parsing HTML and XML documents.

Here's an overview of the code's functionality:

1. Heading Selection: The code defines a list of HTML heading tags (h1, h2, h3, h4, h5, h6) using a Python list. These

heading tags are commonly used to structure the text content on a webpage.

2. Text Extraction: It utilizes BeautifulSoup's 'find_all' method to locate all instances of the specified heading tags within the parsed HTML content (stored in the 'soup' variable).

3. Iteration and Printing: The code then iterates through the list of found headings and extracts the text content within each heading using the '.text' attribute. It prints each extracted heading text to the console.

This code snippet is particularly useful for extracting structured textual information from web pages, such as article headlines or section titles. By specifying the desired HTML elements (in this case, headings), you can customize the extraction process to capture specific content that is relevant to your web scraping or data analysis needs.

```
In [3]: headings = soup.find_all(['h1', 'h2', 'h3', 'h4', 'h5', 'h6'])
        for heading in headings:
            print("Heading:", heading.text)
```

Output:

```
Heading: Course Selector
Heading: What Is Your Overall Work Experience?
Heading: Fresher
Heading: 0 to 3 Years
Heading: 3+ Years
Heading: What Is Your Area Of Interest?
Heading: Data Science & Analytics
Heading: Web Development
Heading: AWS
Heading: Which of these is your personality type?
Heading: Option - 1
Heading: Option - 2
Heading: Which of these is your personality type?
Heading: Option - 1
Heading: Option - 2
Heading: Which of these is your personality type?
Heading: Option - 1
Heading: Option - 2
Heading: You Are Just One Step Closer To Your Dream IT Career
Heading: India’s Leading and Trusted IT Training Institute  Offering Classroom &
Online Training


Heading: Course Preview
Heading: Explore Our World-Class Coding Courses
Heading: Data Software Engineering
Heading: Full Stack Development
Heading: AWS Cloud Technology
Heading: Data Science & Analytics with AI
Heading: Data Software Engineering
Heading: Full Stack Development
Heading: AWS Cloud Technology
Heading: Data Science & Analytics with AI
Heading: Machine Learning & AI
Heading: Data Science
Heading: Data Analytics
Heading: Python Development
Heading: Java Development
Heading: Frontend Development
Heading: When The Efforts Are Genuine, Numbers Speak Louder.
Heading: Know how our students are benefiting from us!
Heading: Data Analyst at Atrina technologies Pvt Ltd
Heading: Data Research Associate at Aristrocrat
Heading: Data Science Intern at London Club
```


Extract Images:

Description: This Python code snippet illustrates how to extract images and their corresponding source URLs from a webpage using BeautifulSoup, a versatile library for parsing HTML and XML documents.

Here's an explanation of the code's functionality:

1. Image Identification: The code begins by using BeautifulSoup's 'find_all' method to locate all instances of the '<img>' (image) tags within the parsed HTML content (stored in the 'soup' variable). These tags represent image elements on the webpage.

2. Source URL Extraction: For each identified image, the code uses the 'image.get('src')' method to retrieve the

'src' attribute value, which contains the URL of the image source. This URL represents the location of the image file on the web.

3. Printing Image Sources: As the code iterates through the list of images, it prints the extracted image source URLs to the console using the 'print' statement. Each URL corresponds to a specific image found on the webpage.

This code is valuable for web scraping tasks that involve collecting image data for analysis, download, or display. It enables you to identify and capture image source URLs, which can be further processed or utilized according to your specific project requirements.

```
In [4]: images = soup.find_all('img')
        for image in images:
            src = image.get('src')
            print("Image Source:", src)
```

Output:

```
Image Source: https://www.itvedant.com/images/logo.svg
Image Source: https://www.itvedant.com/images/rotation.svg
Image Source: https://www.itvedant.com/images/three.svg
Image Source: https://www.itvedant.com/images/contact/menu.svg
Image Source: https://www.itvedant.com/images/clock.svg
Image Source: https://www.itvedant.com/images/book.svg
Image Source: https://www.itvedant.com/images/clock.svg
Image Source: https://www.itvedant.com/images/book.svg
Image Source: https://www.itvedant.com/images/clock.svg
Image Source: https://www.itvedant.com/images/book.svg
Image Source: https://www.itvedant.com/images/clock.svg
Image Source: https://www.itvedant.com/images/book.svg
Image Source: https://www.itvedant.com/images/navigation-icon/master.svg
Image Source: https://www.itvedant.com/images/navigation-icon/job-gurantee.svg
Image Source: https://www.itvedant.com/images/contact/menu.svg
Image Source: https://www.itvedant.com/images/clock.svg
Image Source: https://www.itvedant.com/images/book.svg
Image Source: https://www.itvedant.com/images/clock.svg
Image Source: https://www.itvedant.com/images/book.svg
Image Source: https://www.itvedant.com/images/clock.svg
Image Source: https://www.itvedant.com/images/book.svg
Image Source: https://www.itvedant.com/images/navigation-icon/pg.svg
Image Source: https://www.itvedant.com/images/contact/menu.svg
Image Source: https://www.itvedant.com/images/clock.svg
Image Source: https://www.itvedant.com/images/book.svg
Image Source: https://www.itvedant.com/images/path.svg
Image Source: https://www.itvedant.com/images/career-guide-icon.svg
Image Source: https://www.itvedant.com/images/career-guide-icon.svg
Image Source: None
Image Source: https://www.itvedant.com/images/cross.svg
Image Source: https://www.itvedant.com/images/cross.svg
```

Extract Links with Specific Text:

Description: This Python code snippet demonstrates how to extract hyperlinks from a webpage that contain specific text (in this case, "Learn More") within their anchor (a) tags. It uses BeautifulSoup, a library for parsing HTML and XML documents, to achieve this task.

Here's an overview of the code's functionality:

1. Custom Filtering Function: The code defines a custom filtering function called 'filter_links_with_text.' This function is used as an argument for the 'find_all' method of BeautifulSoup. It filters anchor ('a') tags based on specific criteria, in this case, checking if the tag's name is 'a' and if the text within the tag contains the phrase "Learn More."

2. Link Extraction: The 'soup.find_all(filter_links_with_text)' line utilizes the custom filtering function to find and retrieve all anchor ('a') tags that meet the specified criteria. These tags are stored in the 'filtered_links' variable.

3. URL Retrieval: For each of the filtered anchor tags, the code uses 'link.get('href')' to extract the 'href' attribute value, which represents the URL associated with the hyperlink.

4. Printing Filtered Links: As the code iterates through the list of filtered links, it prints the extracted URLs that

contain the text "Learn More" to the console.

This code is particularly useful when you need to target and collect specific hyperlinks on a webpage that match certain text criteria. It allows you to extract and work with links that are relevant to your project or research, providing a tailored approach to web scraping.

```
In [5]: def filter_links_with_text(tag):
            return tag.name == 'a' and "Learn More" in tag.text

        filtered_links = soup.find_all(filter_links_with_text)
        for link in filtered_links:
            href = link.get('href')
            print("Link with 'Learn More':", href)
```

Output:

```
Link with 'Learn More': https://www.itvedant.com/data-software-engineering
Link with 'Learn More': https://www.itvedant.com/full-stack-developer
Link with 'Learn More': https://www.itvedant.com/aws-azure-cloud-technology
Link with 'Learn More': https://www.itvedant.com/data-science-and-ai
Link with 'Learn More': https://www.itvedant.com/data-software-engineering
Link with 'Learn More': https://www.itvedant.com/full-stack-developer
Link with 'Learn More': https://www.itvedant.com/aws-azure-cloud-technology
Link with 'Learn More': https://www.itvedant.com/data-science-and-ai
Link with 'Learn More': https://www.itvedant.com/ml-ai-course
Link with 'Learn More': https://www.itvedant.com/data-science-course
Link with 'Learn More': https://www.itvedant.com/business-data-analytics
Link with 'Learn More': https://www.itvedant.com/python-training-course
Link with 'Learn More': https://www.itvedant.com/java-course
Link with 'Learn More': https://www.itvedant.com/frontend-course
```

Extract Lists (ul and ol):

Description: This Python code snippet demonstrates how to extract list items from both unordered (ul) and ordered (ol) lists within a webpage using BeautifulSoup, a library for parsing HTML and XML documents.

Here's a breakdown of the code's functionality:

1. List Identification: The code begins by using BeautifulSoup to locate all instances of unordered lists ('<ul>') and ordered lists ('<ol>') within the parsed HTML content (stored in the 'soup' variable). These lists may contain multiple list items ('<li>').

2. List Item Extraction - Unordered Lists: For each identified unordered list ('ul'), the code further narrows down its search by using 'ul.find_all('li')' to locate all list items ('<li>') within that unordered list. It retrieves a list of these items.

3. Printing Unordered List Items: As the code iterates through the list of found unordered lists and their respective list items, it prints the text content of each unordered list item to the console, preceded by "Unordered List Item:".

4. List Item Extraction - Ordered Lists: Similarly, the code repeats the process for ordered lists ('ol'). It locates all list items ('<li>') within each ordered list using 'ol.find_all('li')'.

5. Printing Ordered List Items: As with unordered lists, the code prints the text content of each ordered list item to the console, this time preceded by "Ordered List Item:".

This code is valuable for extracting structured information from lists on web pages, whether they contain bullet points, numbered items, or any other form of list-based content. It allows you to capture and process this information for various analytical or data extraction purposes.

```
In [15]: unordered_lists = soup.find_all('ul')
         ordered_lists = soup.find_all('ol')

         for ul in unordered_lists:
             list_items = ul.find_all('li')
             for item in list_items:
                 print("Unordered List Item:", item.text)

         for ol in ordered_lists:
             list_items = ol.find_all('li')
             for item in list_items:
                 print("Ordered List Item:", item.text)
```

Output:

```
Unordered List Item:
Explore Courses


Popular Certification Courses




Data Software Engineering

                              #Trending


Duration: Job Ready in 8 Month
Case Studies: 15




Full Stack Development

                              #RightChoice


Duration: Job Ready in 6 Month
Case Studies: 15




AWS Cloud Technology

                              #Trending
```


Extract Images with Alt Text:

Description: This Python code snippet showcases how to extract images from a webpage while simultaneously capturing their associated alt text (alternative text). Alt text is essential for accessibility purposes, helping individuals with visual impairments understand the content of images. The code utilizes BeautifulSoup, a library for parsing HTML and XML documents, to accomplish this task.

Here's a step-by-step explanation of the code's functionality:

1. Image Identification: The code commences by using BeautifulSoup to locate all instances of the '<img>' (image) tags within the parsed HTML content (stored in the 'soup' variable). These tags represent image elements on the webpage.

2. Image Source and Alt Text Extraction: For each identified image, the code employs 'image.get('src')' to retrieve the 'src' attribute value, which contains the URL of the image source. Additionally, it utilizes 'image.get('alt', 'No Alt Text')' to retrieve the 'alt' attribute value (alt text) associated with the image. If an image lacks alt text, it defaults to 'No Alt Text.'

3. Printing Image Information: As the code iterates through the list of images, it prints two pieces of information

for each image: the image source URL (preceded by "Image Source:") and the alt text (preceded by "Alt Text:"). This dual information retrieval ensures that both the image content and its accessibility information are captured.

By incorporating this code into a web scraping project, you can systematically extract images while promoting web accessibility by preserving alt text. This makes the gathered image data more informative and inclusive for all users.

```
In [19]: images = soup.find_all('img')
         for image in images:
             src = image.get('src')
             alt_text = image.get('alt', 'No Alt Text')
             print("Image Source:", src)
             print("Alt Text:", alt_text)
```

Output:

```
Alt Text: Itvedant logo
Image Source: https://www.itvedant.com/images/rotation.svg
Alt Text: Explore Courses
Image Source: https://www.itvedant.com/images/three.svg
Alt Text:
Image Source: https://www.itvedant.com/images/contact/menu.svg
Alt Text:
Image Source: https://www.itvedant.com/images/clock.svg
Alt Text: Duration
Image Source: https://www.itvedant.com/images/book.svg
Alt Text: Case study
Image Source: https://www.itvedant.com/images/clock.svg
Alt Text: Duration
Image Source: https://www.itvedant.com/images/book.svg
Alt Text: Case study
Image Source: https://www.itvedant.com/images/clock.svg
Alt Text: Duration
Image Source: https://www.itvedant.com/images/book.svg
Alt Text: Case study
Image Source: https://www.itvedant.com/images/clock.svg
Alt Text: Duration
Image Source: https://www.itvedant.com/images/book.svg
Alt Text: Case study
Image Source: https://www.itvedant.com/images/navigation-icon/master.svg
Alt Text: No Alt Text
Image Source: https://www.itvedant.com/images/navigation-icon/job-gurantee.svg
Alt Text:
Image Source: https://www.itvedant.com/images/contact/menu.svg
Alt Text:
Image Source: https://www.itvedant.com/images/clock.svg
Alt Text: Duration
Image Source: https://www.itvedant.com/images/book.svg
Alt Text: Case study
Image Source: https://www.itvedant.com/images/clock.svg
Alt Text: Duration
Image Source: https://www.itvedant.com/images/book.svg
Alt Text: Case study
Image Source: https://www.itvedant.com/images/clock.svg
Alt Text: Duration
Image Source: https://www.itvedant.com/images/book.svg
Alt Text: Case study
Image Source: https://www.itvedant.com/images/navigation-icon/pg.svg
Alt Text:
Image Source: https://www.itvedant.com/images/contact/menu.svg
Alt Text:
```


To arrange data with the help of pandas

Description: This Python code snippet combines web scraping using BeautifulSoup and data organization using the Pandas library to create a structured dataset from image data found on a webpage. Here's an overview of the code's functionality:

1. URL Definition and HTTP Request: The code starts by defining the URL of the website you want to scrape (in this case, "https://www.itvedant.com/"). It then sends an HTTP GET request to the specified URL using the 'requests' library.

2. Successful Request Check: The code checks if the HTTP request was successful by examining the status code (200 indicates success). If successful, it proceeds with parsing the HTML content of the webpage using BeautifulSoup, storing it in the 'soup' variable.

3. Image Extraction and Data Preparation: Within the parsed HTML content, the code uses BeautifulSoup to locate all instances of '<img>' (image) tags, representing image elements on the webpage. It extracts the 'src' attribute value (image source URL) for each image and organizes this data in the form of a list of dictionaries called 'image_data.' Each dictionary contains an "Image Source" key-value pair.

4. Pandas DataFrame Creation: The code creates a Pandas DataFrame ('df') from the 'image_data' list. This DataFrame structure allows for organized storage and manipulation of the image source URLs.

5. Displaying the DataFrame: Finally, the code prints the Pandas DataFrame to the console, showcasing the structured image data extracted from the webpage.

By using Pandas, this code efficiently converts unstructured web data into a tabular format, making it easier to analyze, filter, and work with image-related information scraped from the website.

In [25]:
```
import requests
from bs4 import BeautifulSoup
import pandas as pd

# URL of the website you want to scrape
url = "https://www.itvedant.com/"

# Send an HTTP GET request to the website
response = requests.get(url)

# Check if the request was successful (status code 200)
if response.status_code == 200:
    # Parse the HTML content
    soup = BeautifulSoup(response.text, "html.parser")

    # Extract images and their src attributes
    images = soup.find_all('img')
    image_data = [{"Image Source": image.get('src')} for image in images]

    # Create a Pandas DataFrame
    df = pd.DataFrame(image_data)

    # Display the DataFrame
    print(df)

else:
    print("Failed to retrieve the page. Status code:", response.status_code)
```

Output:

```
                                    Image Source
0            https://www.itvedant.com/images/logo.svg
1         https://www.itvedant.com/images/rotation.svg
2            https://www.itvedant.com/images/three.svg
3     https://www.itvedant.com/images/contact/menu.svg
4            https://www.itvedant.com/images/clock.svg
..                                                 ...
430            https://www.itvedant.com/images/fly.svg
431  https://www.itvedant.com/images/contact/schedu...
432                                               None
433   https://www.itvedant.com/images/logo_launch.webp
434              https://www.itvedant.com/images/mx.svg

[435 rows x 1 columns]
```

```python
import requests
from bs4 import BeautifulSoup
import pandas as pd

# URL of the website you want to scrape
url = "https://www.itvedant.com/"

# Send an HTTP GET request to the website
response = requests.get(url)

# Check if the request was successful (status code 200)
if response.status_code == 200:
    # Parse the HTML content
    soup = BeautifulSoup(response.text, "html.parser")

    # Extract text from unordered lists
    unordered_lists = soup.find_all('ul')
    unordered_list_data = []
    for ul in unordered_lists:
        list_items = ul.find_all('li')
        for item in list_items:
            unordered_list_data.append({"Unordered List Item": item.text})

    # Extract text from ordered lists
    ordered_lists = soup.find_all('ol')
    ordered_list_data = []
    for ol in ordered_lists:
        list_items = ol.find_all('li')
        for item in list_items:
            ordered_list_data.append({"Ordered List Item": item.text})

    # Create Pandas DataFrames for unordered and ordered lists
    df_unordered = pd.DataFrame(unordered_list_data)
    df_ordered = pd.DataFrame(ordered_list_data)

    # Display the DataFrames
    print("Unordered List Data:")
    print(df_unordered)

    print("\nOrdered List Data:")
    print(df_ordered)

else:
    print("Failed to retrieve the page. Status code:", response.status_code)
```


Output:

```
Unordered List Data:
                                Unordered List Item
0   \nExplore Courses\n\n\nPopular Certification C...
1   Popular Certification Courses\n                 ...
2   100% Job Guarantee\n                          \n...
3   Job Assurance Courses\n                         ...
4                                \nCourse Selector \n
..                                                  ...
84                                           Contact
85                                  \n\n\nBrochure \n
86                                  \nPopular Goals\n
87                            \n 100% Job Guarantee\n
88                           \nJob Assurance Courses\n

[89 rows x 1 columns]

Ordered List Data:
Empty DataFrame
Columns: []
Index: []
```

In [27]:
```python
import requests
from bs4 import BeautifulSoup
import pandas as pd

# URL of the website you want to scrape
url = "https://www.itvedant.com/"

# Send an HTTP GET request to the website
response = requests.get(url)

# Check if the request was successful (status code 200)
if response.status_code == 200:
    # Parse the HTML content
    soup = BeautifulSoup(response.text, "html.parser")

    # Define the filter function
    def filter_links_with_text(tag):
        return tag.name == 'a' and "Learn More" in tag.text

    # Find links with 'Learn More' text using the filter function
    filtered_links = soup.find_all(filter_links_with_text)

    # Extract href attributes from filtered links
    link_data = [{"Link with 'Learn More'": link.get('href')} for link in filtere

    # Create a Pandas DataFrame
    df = pd.DataFrame(link_data)

    # Display the DataFrame
    print(df)

else:
    print("Failed to retrieve the page. Status code:", response.status_code)
```

Output:

	Link with 'Learn More'
0	https://www.itvedant.com/data-software-enginee...
1	https://www.itvedant.com/full-stack-developer
2	https://www.itvedant.com/aws-azure-cloud-techn...
3	https://www.itvedant.com/data-science-and-ai
4	https://www.itvedant.com/data-software-enginee...
5	https://www.itvedant.com/full-stack-developer
6	https://www.itvedant.com/aws-azure-cloud-techn...
7	https://www.itvedant.com/data-science-and-ai
8	https://www.itvedant.com/ml-ai-course
9	https://www.itvedant.com/data-science-course
10	https://www.itvedant.com/business-data-analytics
11	https://www.itvedant.com/python-training-course
12	https://www.itvedant.com/java-course
13	https://www.itvedant.com/frontend-course

In [29]:
```python
import requests
from bs4 import BeautifulSoup
import pandas as pd

# URL of the website you want to scrape
url = "https://www.itvedant.com/"

# Send an HTTP GET request to the website
response = requests.get(url)

# Check if the request was successful (status code 200)
if response.status_code == 200:
    # Parse the HTML content
    soup = BeautifulSoup(response.text, "html.parser")

    # Find all headings (h1, h2, h3, h4, h5, h6)
    headings = soup.find_all(['h1', 'h2', 'h3', 'h4', 'h5', 'h6'])

    # Extract heading text into a list
    heading_text = [heading.text for heading in headings]

    # Create a Pandas DataFrame
    df = pd.DataFrame({"Headings": heading_text})

    # Display the DataFrame
    print(df)

else:
    print("Failed to retrieve the page. Status code:", response.status_code)
```

Output:

	Headings
0	Course Selector
1	What Is Your Overall Work Experience?
2	Fresher
3	0 to 3 Years
4	3+ Years
..	...
127	Nearest Branch
128	Visit Date
129	Your Preferred Time Slot
130	Visit Successfully Scheduled
131	Watch This Amazing Video We Crafted For You.

[132 rows x 1 columns]

Store all the data above in CSV file and open this file in application

Description: This Python code snippet extends the web scraping and data organization example by adding functionality to store the extracted data in a CSV (Comma-Separated Values) file and then open the CSV file in a separate application. Here's a breakdown of the code's expanded functionality:

1. URL Definition and HTTP Request: The code starts by defining the URL of the website to be scraped ("https://www.itvedant.com/") and sending an HTTP GET request to this URL using the 'requests' library.

2. Successful Request Check: It checks whether the HTTP request was successful by examining the status code (200 indicates success). If the request is successful, the code proceeds with parsing the HTML content of the webpage using BeautifulSoup and stores it in the 'soup' variable.

3.Data Extraction: In this example, the code focuses on extracting headings from the webpage. It locates all heading tags ('h1' to 'h6') and extracts their text content, storing it in a list called 'heading_text.'

4. Pandas DataFrame Creation: The code creates a Pandas DataFrame ('df') from the 'heading_text' list. This DataFrame organizes the extracted headings in a structured format.

5. CSV File Creation: The code specifies a CSV filename ('extracted_data.csv') and saves the Pandas DataFrame ('df') to this file using the 'to_csv' method. The 'index=False' parameter ensures that the DataFrame index is not included in the CSV file.

6. CSV File Confirmation: It prints a message indicating that the data has been successfully saved to the CSV file along with the filename.

7. Opening the CSV File: To open the CSV file in a separate application, you can do so manually using software such as Microsoft Excel, Google Sheets, or any CSV-compatible spreadsheet application.

This code effectively combines web scraping, data transformation, and data storage in a CSV file. The resulting CSV file contains the extracted data (headings, in this case) in a format that can be easily imported and analyzed in various data analysis tools.

In [30]:
```python
import requests
from bs4 import BeautifulSoup
import pandas as pd

# URL of the website you want to scrape
url = "https://www.itvedant.com/"

# Send an HTTP GET request to the website
response = requests.get(url)

# Check if the request was successful (status code 200)
if response.status_code == 200:
    # Parse the HTML content
    soup = BeautifulSoup(response.text, "html.parser")

    # Extract data here, for example, headings
    headings = soup.find_all(['h1', 'h2', 'h3', 'h4', 'h5', 'h6'])
    heading_text = [heading.text for heading in headings]

    # Create a Pandas DataFrame
    df = pd.DataFrame({"Headings": heading_text})

    # Save the DataFrame to a CSV file
    csv_filename = "extracted_data.csv"
    df.to_csv(csv_filename, index=False)

    print("Data has been saved to", csv_filename)

else:
    print("Failed to retrieve the page. Status code:", response.status_code)
```

Output:

```
Data has been saved to extracted_data.csv
```

LINK TO PYTHON JUPYTER FILE

file:///C:/Users/Administrator/Downloads/IT%20Vedant%20web%20scraping%20(1)%20(2).html

CONCLUSION

In conclusion, this web scraping project has demonstrated the power and versatility of Python, along with libraries like BeautifulSoup and Pandas, in extracting, organizing, and storing data from a target webpage. By sending HTTP GET requests to the specified website and parsing its HTML content, we were able to capture valuable information such as headings and images.

Furthermore, the project showcased how data can be structured and transformed into a Pandas DataFrame, offering a convenient way to work with and analyze the collected information. To enhance accessibility and data portability, we also saved the extracted data in a CSV file.

Web scraping, when conducted responsibly and ethically, provides a valuable means of acquiring data from the web for a wide range of applications, including research, analysis, and automation. As technology evolves, web scraping remains an essential skill for harnessing the wealth of information available on the internet and transforming it into actionable insights. This project serves as a foundational example for those looking to embark on web scraping endeavors, offering a blueprint for extracting and managing web data effectively.

THANKYOU,

AKSHADA JARODE