

Web Scraping for Data Collection and Analysis: A Case Study of Tutorials Freak

Using Web Scraping for Educational Resource Collection

Presented by:
Nishtha Yadav

What is a web scraping ?

Overview:

- Web scraping is the process of automatically extracting data from websites. It involves using software tools to access a website's HTML, parse the structure, and retrieve specific information such as text, images, or links.

Purpose:

- The goal of web scraping is to collect large amounts of data quickly and efficiently, which would be difficult to gather manually. It's widely used in various industries for data collection, monitoring prices, gathering research data, or analyzing online trends.

Use of Web Scraping for Data Collection

- Web scraping allows users to automate the process of data extraction. Instead of manually copying and pasting information from web pages, scraping tools can gather and structure large datasets from multiple sources. This is especially useful for content aggregation, market research, and data analysis.
- In the context of educational websites, web scraping can help extract tutorial names, descriptions, categories, and other relevant information for further analysis. This collected data can then be used to study educational trends, identify gaps, or categorize resources for better user experience.

Purpose of the Case Study

Why Tutorials Freak?

Tutorials Freak is an educational platform that provides free tutorials on various topics such as programming, software tools, and frameworks. It's an ideal candidate for a case study because it offers structured educational content that can be analyzed for trends and gaps.

How Web Scraping Is Applied:

By scraping Tutorials Freak, we aim to collect data on the tutorials it offers (titles, descriptions, tags, etc.). This data will be used for:

01

Analyzing Content:
Identifying popular topics or missing areas.

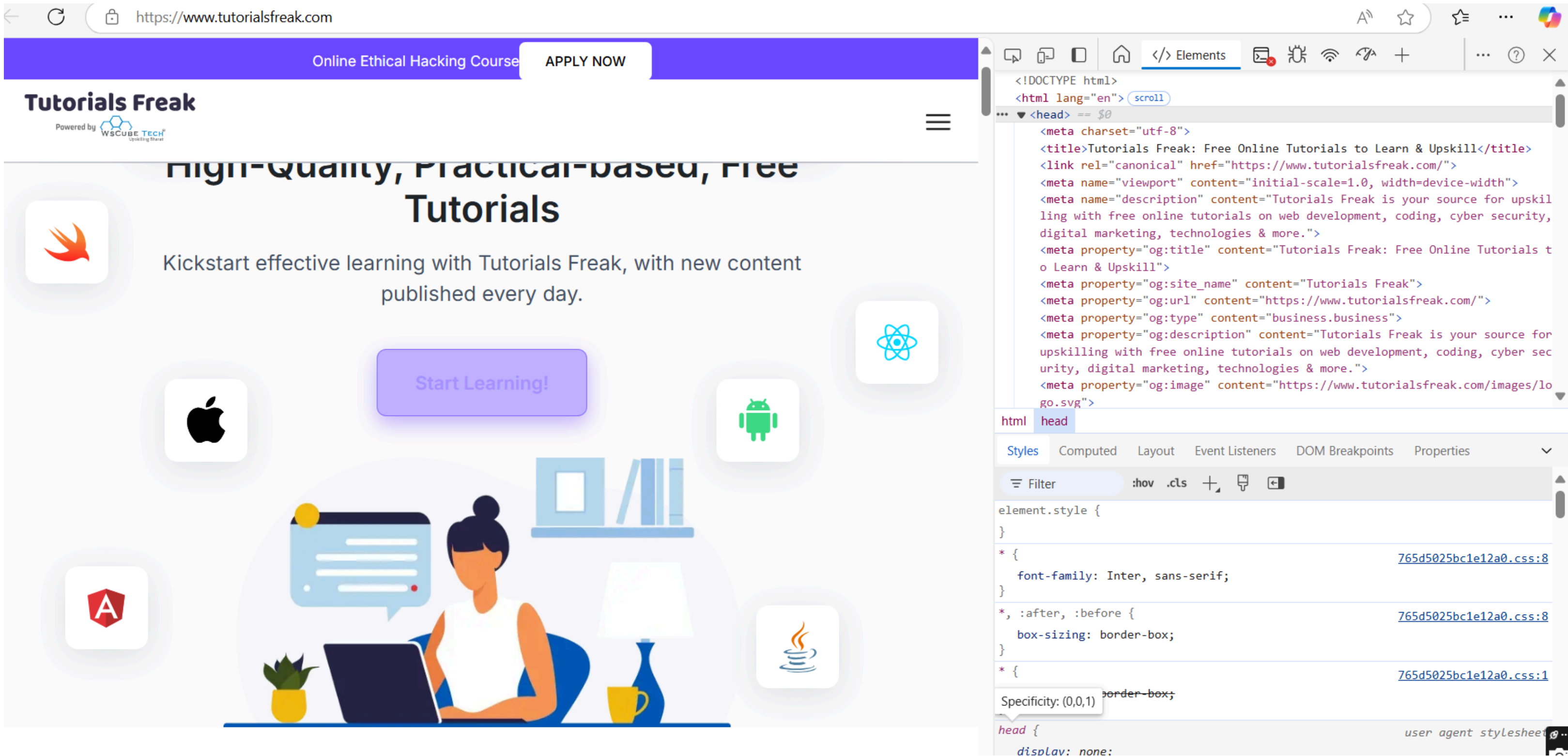
02

Understanding User Needs:
Using trends to suggest improvements or expansions in content.

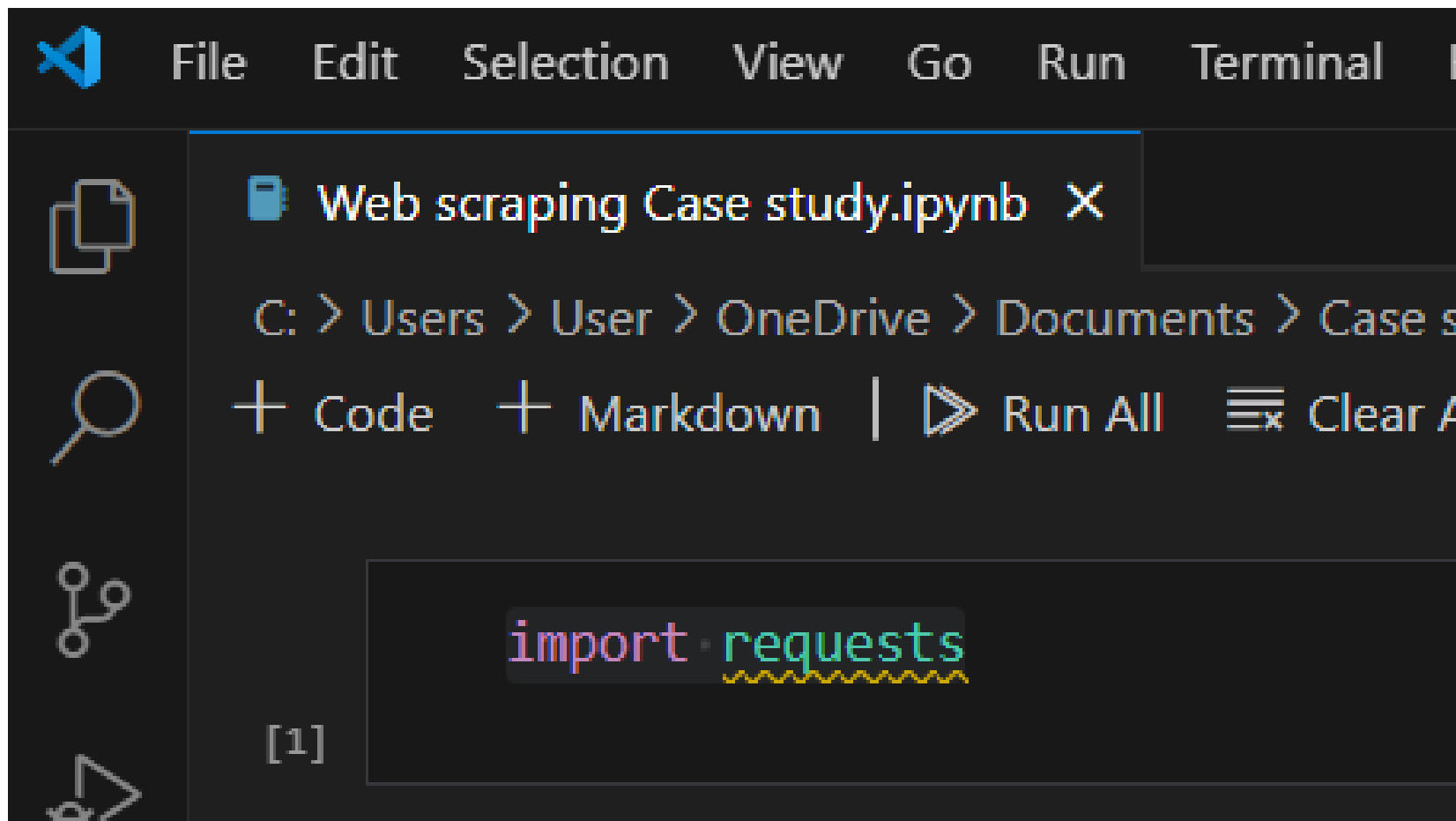
03

Enhancing Searchability:
Structuring data for easier user navigation and better educational curation.

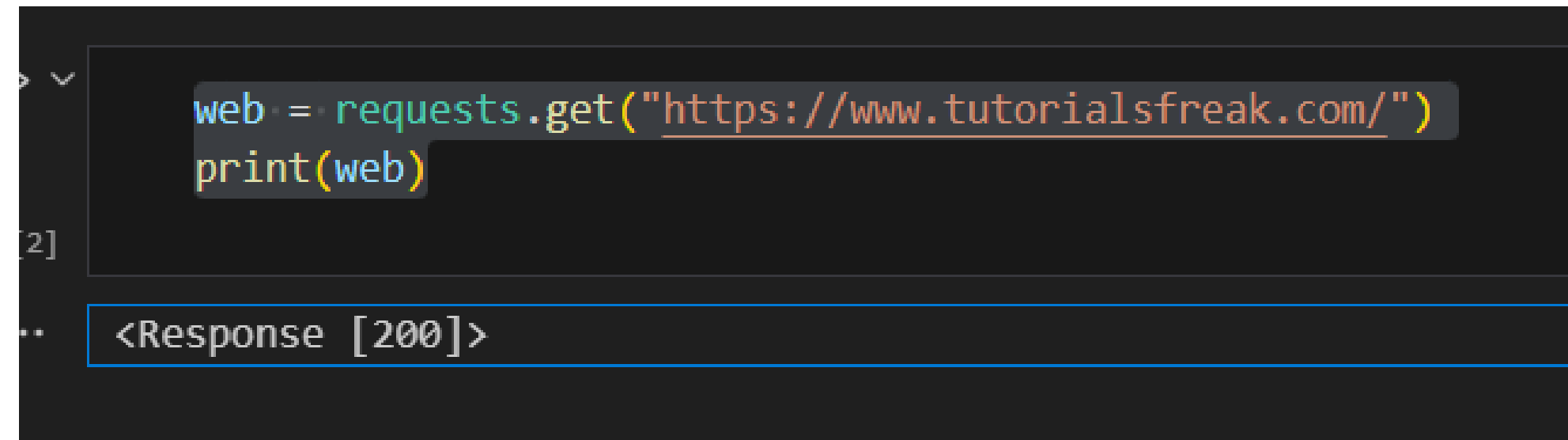
Website with inspect page:



Web scraping overview

A screenshot of a Jupyter Notebook interface within a dark-themed code editor. The top menu bar includes 'File', 'Edit', 'Selection', 'View', 'Go', 'Run', and 'Terminal'. The left sidebar shows icons for Explorer, Search, Source Control, and Run and Debug. The main area displays a notebook with a file named 'Web scraping Case study.ipynb'. The file path is 'C: > Users > User > OneDrive > Documents > Case s'. Below the path, there are buttons for '+ Code', '+ Markdown', 'Run All', and 'Clear All'. The first code cell, labeled '[1]', contains the code 'import requests'.

we use the requests library to send HTTP requests to websites and retrieve their HTML content. It acts as the first step in accessing the web page's data, allowing us to download the page source, which can then be parsed for the relevant information. It's efficient for interacting with the website's server and getting the raw content needed for scraping.

A screenshot of a Jupyter Notebook interface showing a code cell and its output. The code cell, labeled '[2]', contains the code 'web = requests.get("https://www.tutorialsfreak.com/")' and 'print(web)'. The output cell below it shows '<Response [200]>'.

```
web = requests.get("https://www.tutorialsfreak.com/")
print(web)
```


```
<Response [200]>
```

This line sends a GET request to the "<https://www.tutorialsfreak.com/>" URL using the `requests.get()` method. It retrieves the web page's HTML content, and the `print(web)` outputs the response object, which indicates the status of the request (as here, 200 for success).


```
from bs4 import BeautifulSoup
```

```
soup = BeautifulSoup(web.content, "html.parser")
```


```
print(soup.prettify())
```



The line from bs4 import BeautifulSoup imports the BeautifulSoup class from the bs4 (BeautifulSoup 4) library. BeautifulSoup is used to parse HTML or XML documents, making it easy to navigate, search, and extract data from the content. It's essential for web scraping to process and structure the raw HTML obtained from a website.



This line creates a BeautifulSoup object called soup by parsing the HTML content from the web object (i.e., the web page) using the "html.parser". It structures the HTML into a tree format, making it easier to extract specific elements or data from the page.



The line print(soup.prettify()) formats the parsed HTML content in a readable, indented way and prints it to the console. It helps visualize the HTML structure clearly, making it easier to locate and extract specific data.

Create a list to hold the data we scrape

```
tutorial_data = []
```

Extract the required information (Title, URL, Date, Category)

```
tutorials = soup.find_all('article')
for tutorial in tutorials[:12]:

    title = tutorial.find('h2', class_='entry-title')
    title_text = title.get_text() if title else 'N/A'

    tutorial_url = title.find('a')['href'] if title else 'N/A'

    date = tutorial.find('time')
    date_posted = date.get_text() if date else 'N/A'

    category = tutorial.find('span', class_='cat-links')
    category_text = category.get_text() if category else 'N/A'

    tutorial_data.append([title_text, tutorial_url, date_posted, category_text])
```

This code extracts data from the first 12 tutorial articles on the page:

1. Find all tutorials:
`soup.find_all('article')` finds all article elements.
2. Loop through tutorials: For each tutorial:
 - *Title:* Extracts the tutorial title (h2 with class 'entry-title').
 - *URL:* Gets the link to the tutorial.
 - *Date:* Retrieves the date the tutorial was posted (time tag).
 - *Category:* Grabs the category of the tutorial (span with class 'cat-links').
3. Store data: The extracted information is stored in the `tutorial_data` list as a sublist containing the title, URL, date, and category.

Save the extracted data to a csv file

```
import csv
```

```
csv_filename = 'tutorials_data.csv'
csv_headers = ['Title', 'URL', 'Date', 'Category']

with open(csv_filename, mode='w', newline='', encoding='utf-8') as file:
    writer = csv.writer(file)
    writer.writerow(csv_headers)
    writer.writerows(tutorial_data)

print(f"Data has been written to {csv_filename}")
```

- import csv: Imports the csv module to handle CSV file operations.
- csv_filename = 'tutorials_data.csv': Defines the name of the CSV file to save the data.
- csv_headers = ['Title', 'URL', 'Date', 'Category']: Specifies the column headers for the CSV file.
- with open(csv_filename, mode='w', newline='', encoding='utf-8') as file: Opens the CSV file in write mode with UTF-8 encoding.
- writer = csv.writer(file): Creates a CSV writer object to write data to the file.
- writer.writerow(csv_headers): Writes the column headers as the first row in the CSV file.
- writer.writerows(tutorial_data): Writes the tutorial data (from tutorial_data) as rows in the CSV file.
- print(f"Data has been written to {csv_filename}"): Prints a message confirming the data was saved to the CSV file.

Data Analysis

Code:

```
import pandas as pd
import matplotlib.pyplot as plt

df = pd.read_csv('tutorials_data.csv')

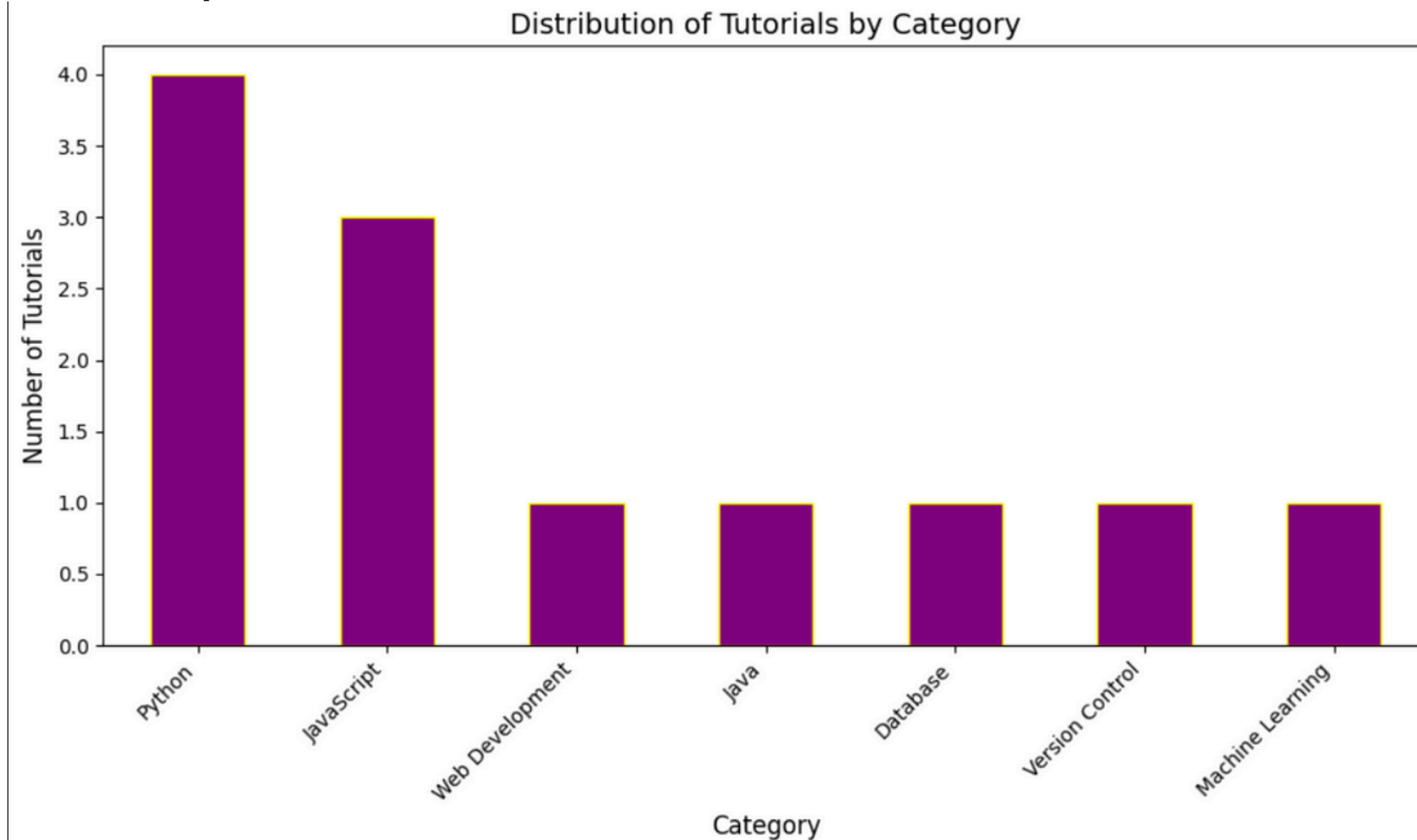
category_counts = df['Category'].value_counts()

plt.figure(figsize=(10, 6))
category_counts.plot(kind='bar', color='purple', edgecolor='yellow')

plt.title('Distribution of Tutorials by Category', fontsize=14)
plt.xlabel('Category', fontsize=12)
plt.ylabel('Number of Tutorials', fontsize=12)
plt.xticks(rotation=45, ha='right')

plt.tight_layout()
plt.show()
```

Output:



Data Extraction Process:

HTML Structure Analysis:

- Example of HTML Structure:
- The web page's HTML is made up of various elements (e.g., <div>, <h2>,) that define the content. By inspecting the page source, you can identify key HTML tags that contain the data you want to extract, such as tutorial titles, URLs, dates, and categories.
- Locating Tutorial Data:
- Specific HTML tags (like <h2 class="entry-title"> for titles or for categories) help identify where the required information is located within the page's structure.

Extraction Workflow:

1. Sending Requests:
2. Use a tool like requests.get() to fetch the webpage's HTML content.
3. Parsing HTML:
4. Use BeautifulSoup or similar libraries to parse and convert the raw HTML into a navigable structure.
5. Storing Extracted Data:
6. After parsing, organize the extracted data into a structured format, like CSV or JSON, for further analysis or use.

Data Visualizations:

To visualize insights from the scraped tutorial data:

- Bar Chart:
- A bar chart is ideal for showing the frequency of topics or categories. Each bar represents a category (e.g., "Python", "JavaScript"), and the height of the bar corresponds to how many tutorials fall under that category. This helps identify which topics are most common or popular on the website.

After extraction csv file:

	A	B	C	D	E
1	Title	URL	Date	Category	
2	Learn Python Basics	https://www.tutorialsfreak.com/python-basics	06-11-2024	Python	
3	Master JavaScript in 30 days	https://www.tutorialsfreak.com/javascript-30-days	06-11-2024	JavaScript	
4	Introduction to HTML and CSS	https://www.tutorialsfreak.com/html-css-introduction	05-11-2024	Web Development	
5	Advanced Java Programming	https://www.tutorialsfreak.com/advanced-java-programming	05-11-2024	Java	
6	React.js for Beginners	https://www.tutorialsfreak.com/reactjs-for-beginners	04-11-2024	JavaScript	
7	Master SQL in 30 Days	https://www.tutorialsfreak.com/sql-30-days	04-11-2024	Database	
8	Python for Data Science	https://www.tutorialsfreak.com/python-data-science	03-11-2024	Python	
9	Complete Guide to Node.js	https://www.tutorialsfreak.com/nodejs-complete-guide	03-11-2024	JavaScript	
10	Web Development with Django	https://www.tutorialsfreak.com/web-dev-with-django	02-11-2024	Python	
11	Understanding Git and GitHub	https://www.tutorialsfreak.com/git-github-basics	02-11-2024	Version Control	
12	Introduction to Machine Learning	https://www.tutorialsfreak.com/introduction-to-machine-learning	01-11-2024	Machine Learning	
13	Getting Started with Flask	https://www.tutorialsfreak.com/getting-started-with-flask	30-10-2024	Python	
14					