Date: / /
Page No. :

Principles of Data visualization

1) what is web scraping. Describe the steps to perform web scraping.

Web scraping is the process of automatically extracting data from websites. It allows users to gather informations from webpages and store it in a structured format for analysis such as in a database or a spreadsheet. Web scraping is widely used in applications like data analysis price comparison, lead generation, and research. The data is often scraped using automated tools and scripts, typically written in languages like python, which help navigate and extract the required information from HTML pages.

Steps to perform web scraping:

1) Identify the target website: choose a website from which data needs to be extracted. Analyse the structure and format of the website, ensuring that the data is publicly available and that scraping the site complies with it terms of service.

2) Inspect the web page: use the browser's developer tools (Inspect) to examine the HTML structure

Date: / /
Page No.:

of the web page. Identify the specific tags (such as , <table>, , etc.) that contains the data you want to scrape.
for example in a e-commerce site, product names and prices may be stored in specific HTML elements.

3) Choose the Appropriate Tools or Libraries:
Depending on the programming languages, choose appropriate libraries or frameworks for web scraping. In python popular libraries include:
* Beautifulsoup: for parsing HTML and extracting data.
* Selenium: for simulating browser interaction & scraping dynamic content.
* Scrapy: A comprehensive framework for large scale scraping projects.
* Requests: for sending HTTP requests to fetch

4) Send HTTP requests: use tools like Requests to send HTTP GET Requests to the server and retrieve the HTML content of the web page. The response from the server contains the raw HTML data.

```
import requests
response = requests.get('https://example.com')
html_content = response.content
```

5) parse the HTML content: once the HTML content is fetched, parse it using a library like BeautifulSoup the allows for easy navigation of the document

Date: / /
Page No.:

tree & extraction of data from specific tags & attributes from bs4 Import beautifulsoup.

```
soup = BeautifulSoup(html_content, 'HTML.parser')
titles = soup.find_all('h1')
```

6) Extract required data: use the HTML parsed to locate & extract the relevant data. this might involve searching for specific tags, classes or IDs that contain the info.

```
prices = soup.find_all('span', class_='price')
```

7) Handle pagination: many website split data across multiple pages. If the target website uses pagination identify the URL patterns or navigations buttons & write logic to scrape all relevant pages iteratively.

8) Store the data: After extraction, store the data in a structured format, such as csv file, json file or database.

```
import csv
with open('data.csv', 'w') as file:
    writer = csv.writer(file)
    writer.writerow(['Title', 'Price'])
    for title, price in zip(titles, prices):
        writer.writerow([title.text, price.text])
```

9) Handle Javascript Content: some websites load content dynamically using Javascript. Tools like Selenium can be used to simulate a browser environment and extract data after Javascript execution

Date: / /
Page No.:

1) Respect Ethical & Legal Boundaries: Always check the website's robots.txt file to understand the permissions for crawling or scraping. Ensure compliance with legal regulations, including copyright laws & terms of service, and avoid overloading servers with excessive requests. (use delays between requests).

2) Challenges in Performance Web Scraping.
Web scraping, while an effective method of extracting data from websites, comes with several challenges and limitations. These challenges can arise from legal technical or ethical considerations, making it crucial for developers to address them when performing scraping tasks.

1) Legal and Ethical Issues:
* Terms of Service (TOS) * Copyright and Data ownership * Robots.txt & crawling guidelines.
2) Dynamic content and Javascript Rendering:
* Javascript loaded content * Handling dynamic pages.
3) Website structure changes
* Frequent HTML structure changes
* Inconsistent HTML Tags.

Date: / /
Page No.:

4) CAPTCHA & Bot Detection:
* CAPTCHA systems * Anti scraping measures
* IP Blocking & Rate limiting.
5) Data Quality Issues: * Incomplete or Noisy Data
* Encoding and formatting Issues.
6) Ethical considerations & server overload:
* server load & Bandwidth usage
* Respect for website Bandwidth.
7) Data volatility and updates:
* Real time Data changes * consistency over Time.
8) Data privacy & Regulations:
* personal Data protection * Regulatory compliance.

3) Demonstrate a python web scraping script using beautifulsoup library for navigating any html.

A) Beautifulsoup is a python library that allows easy navigation, searching and modification of HTML and XML documents. It is particularly useful for web scraping as it helps parse HTML files & extract the desired data.

```
from bs4 import BeautifulSoup.
with open('example.html', 'r', encoding='utf-8') as file:
    html_content = file.read()
soup = BeautifulSoup(html_content, 'html.parser')
page_title = soup.title.text
```

```
print("Page title:", page_title)
books = soup.find_all('div', class_='book')
for book in books:
    title = book.find('h2').text
    print("Book title:", title)
    author = book.find('p').text
    print("author:", author)
    price = book.find_all('p')[1].text
    print('price:', price)
    print('---')
```

output: Page title : Book store.
Book title = Python for Beginners
Authors = John Doe
Price : $30
---

4) Illustrate the difference between loc & iloc function in python program with a code snippet.

A In python loc & iloc are functions provided by the pandas library to access data from a Dataframe.

*loc is used for label-based indexing. It allows you to select rows & columns by labels.
(row/column names or indices)

*iloc is used for position-based indexing. It selects rows and columns by integer positions (row/column index numbers).

```
import pandas as pd
data = {'Name': ['Akash', 'Nithik', 'Karthik'],
```

Date: / /
Page No.:

```
'Age': [22, 23, 26],
'score': [85 90 86]}.
df = pd.DataFrame(data, index=['a','b','i']).
print("original DataFrame:\n", df)
print("using loc to slect row 'b' & Columns:")
print(df.loc['b', ['name', 'Age']])
print("using iloc to slect row (1) & first two Columns:")
print(df.iloc[1, [0,1]])
print("using loc to slect multiple rows:")
print(df.loc['a':'b', ['name','score']])
print("using iloc to slect multiple rows:")
print(df.iloc[1:2[0,2]]).
```

output: original DataFrame.

	Name	Age	Score
a	Akash	22	85
b	Nithik	23	90
c	Karthik	26	86

using loc

	name	Age
b	Nithik	23

using iloc

	Name	Age
1	Akash	22

using loc to slect multiple rows

	Name	Score
a	Akash	85
c	Karthik	86

using iloc to slect multiple

	Name	Score
1	Nithik	90
2	Karthik	86

loc uses label to slect data.
iloc uses integer position to slect data.

write a python program to create 3×4 array, delete 3rd column & insert new column in the 3rd column. Insert array as np.

```
array = np.array([[1, 2, 3],
                  [4, 5, 6],
                  [7, 8, 9],[3]])
print("original array (3×4)")
print(array)
array_without_3rd_col = np.delete(array, 2, axis=1)
print("array without 3rd col")
array_with_3rd_col = np.insert(array, 2, axis=1)
newcolarray = [[10],[11],[12]]
new_arr = np.insert(array, 2, newcolarray, axis=1)
print("newly new")
```

original 3×4 array.

1	2	3	1
4	5	6	1
7	8	9	1

array after deleting 3rd column

1	2	1
4	5	1
7	8	1

array after inserting in 3rd column.

1	2	10	1
4	5	11	1
7	8	12	1