Date: / /
Page No.:

PDV Assignment.

1) Describe the primary purpose of a user Agent header in web scraping.

→ User-Agent header in web scrapping plays a crucial role in mimicking the behaviour of a real web browser.

Primary purpose of the user-Agent Header

i) Simulate a Real user/Browser:

the user Agent header tells the server what types of device or browser is making the request. When scrapping you can set this header to resemble a real browser like chrome, firefox etc) to prevent the server from blocking the request as suspicious or coming from an automated tool.

ii) Bypass Anti-scrapping Mechanisms:

Many websites use anti bot measures that detect and block requests with non standard or missing user Agent headers. These are often associated with web scrappers or bots.

iii) Handles Different Device or Browser specific content

Some websites deliver different content depending on the browser or device requesting it. By including a user-Agent header that mimics a specific browser

iv) Respect website Terms and conditions

→ While not a guarantee of Respect, using a valid user Agent header that identifies your scrapper as a specific browser can indicate to the server that the request is coming legitimate

Q2) What is the purpose of a robots.txt file on a website?

→ The robots.txt file is a text file placed in the root directory of a website that provides guideline for web crawlers access to website. It uses the robots exclusion protocol & can include rules to restrict specific crawl length or path on the site.

Ex: User-agent:
Disallow: /admin/
Disallow: /private/

* Disallow: Indicates which pages or directories should not be crawled by the specified user agent if the value is left empty (i.e Disallow) it means the crawler is allowed to access all parts of the site.

Q3) Scrape data contents from a single web page using Beautiful Soup.

→ Here's an example of how to scrape data from a web page.

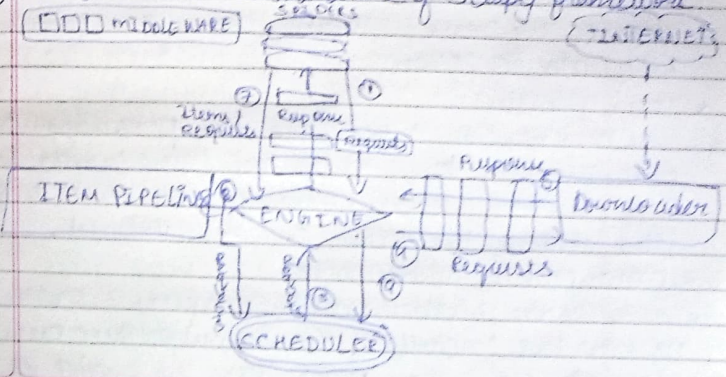
```
from bs4 import BeautifulSoup
import requests
url = "https://quotes.toscrape.com"
response = requests.get(url)
soup = BeautifulSoup(response.text, "html.parser")
quotes = soup.find_all("div", class_="quote")
for quote in quotes:
    text = quote.find("span", class_="text").get_text()
    author = quote.find("small", class_="author").get_text()
    print("Quote:", text, "Author:", author)
```

Q4) Scrape data contents navigating through multiple web page using Beautiful Soup library.

→

```
import requests
from bs4 import BeautifulSoup
base_url = "http://quotes.toscrape.com/page/"
for page in range(1,5):
    url = f"{base_url}{page}/"
    response = requests.get(url)
    soup = BeautifulSoup(response.text, "html.parser")
    quotes = soup.find_all("div", class_="quote")
    for quote in quotes:
        text = quote.find("span", class_="text").get_text()
        author = quote.find("small", class_="author").get_text()
        print(f"Page {page} Quote: {text}\n Author: {author}\n")
```

Q5) Describe the architecture of Scrapy framework.



The data flow is scrapy is controlled by the Execution Engine, and goes like this:

Date: / /
Page No.:

* Scrapy Engine:

This engine is responsible for controlling the data flow between all components of the system and triggering events when certain action occur. see the data flow section above for more details.

* Scheduler:-

The Scheduler receives requests from the engine and enqueues them for feeding them later (also to the engine) when the engine requests them.

* Downloader:-

The Downloader is responsible for fetching web pages and feeding them to the engine which, in turn, feeds them to the spiders.

* Spiders

Spiders are custom classes written by Scrapy users to parse responses & extract items from them or additional requests to follow.

* Item pipeline:

The items pipeline is responsible for processing the items once they have been extracted (or scraped) by the spider. Typical items include cleaning validation & persistance (like storing the item in a database).

* Downloader middlewares:

* Downloader middlewares are specific hooks that sit b/w the Engine & the downloader and process requests when they pass from the engine to the Downloader & responses that pass from downloader to the Engine.

Spider middlewares

Spider middlewares are specific hooks that sit b/w the engine & the spider and are able to process spider input (responses) and output (items and requests).

6) Demonstrate scrapy framework with an example to store information about a product having attributes.

→ Set up scrapy project.

scrapy startproject shopping-scraper.

it will create a folder structure.

Inside the shopping-scraper/items.py.

```
Import scrapy
class productItem (scrapy.Item)
    name = scrapy.Field()
    cost = scrapy.Field()
    manufacturer = scrapy.Field()
    rating = scrapy.Field()
```

shopping-scrapper/spiders → create new python file product_spider.py

```
import scrapy
from shopping_scrapper.items import productItem
class productspider(scrapy.Spider):
    name = 'product_spider'
    start_urls = ['https://www.snapdeal.com']
    def parse(self, response):
        for product in response.css('product'):
            yield {
                "name": product.css("name::text").get(),
                "cost": product.css("cost::text").get(),
                "manufacturer": product.css("manufacturer::text").get(),
```

"rating": product.css('rating ::text').get()

next_page = response.css('a.next_page::attr(href)').get()

if next_page:
    yield response.follow(next_page, self.parse)

7) Create a pandas dataframe from supermarket invoice data, and execute following operations

i) load the data

```
import pandas as pd
df = pd.read_csv("supermarket_invoice_data")
```

a) count the number of customers per product category or department.

```
customer_count = df.groupby('description')['customer ID'].nunique()
print(customer_count)
```

b) calculate the average customer visit per product category

```
avg_visits = df.groupby('description')['customer ID'].value_counts().mean()
print(avg_visits)
```

c) identify the products bought by the most customers

```
product_bought = df.groupby('description')['customerID'].nunique().idxmax()
print(product_bought)
```

d) sort customer based on age, products purchased or products cost

```
sorted_customer = df.sort_values(by=['Sales', 'Quantity'], ascending=[False, False])
print(sorted_customer)
```

8) Explain categories of exploratory visualizations based on the relationships b/w the three necessary players.

i) univariate visualizations (single variable)

These visualizations explore the characteristics of a single variable. The relationship here is b/w a single variable & its visual representation & how the user (observer) interpret patterns.

examples:

Histogram: Display the distribution of a continuous variable.

Bar: Show frequency counts or categories for a discrete variable.

ii) Bivariate visualizations (Two Variables)

These visualizations focus on the relationship b/w two variables, either numerical or categorical. The key relationship here is b/w two variables & their visual mapping.

iii) multivariate visualizations (multiple variables)

when exploring the relationships b/w three or more variable the visualization often becomes more complex.

The user is tasked with interpreting multiple data dimensions simultaneously here the visual representation could employ color, shape, size or even animation to represent more than

9) Define DAX Demonstrate any 10 DAX formulas with a context.

i) Count: counts the no of non-blank rows in a column typically used to count the no of rows with valid entries in a dataset.

ii) Distinct count: counts the distinct (unique) values in a column useful for identifying the no of unique entries, such as distinct customers products or transactions.

iii) Average: calculates the average of a column used to find the mean value such as average sales or average quantities.

iv) MIN: Returns the smallest value from a column useful for identifying the minimum numerical or date value, such as the lowest sales amount or earliest date.

v) MAX: Returns the largest value from a column helps identify the maximum value in a dataset such as highest sales or latest date.

6) Summarize: Returns a summary table for the requested columns and aggregates data used to group data and apply aggregations like sum or count.

7) Calculate: Modifies the context of a calculation with filters used to calculate results under specific conditions, such as sales for a specific product category.

8) IF: returns one value if a condition is true another if it's false enables conditional calculations such as classifying sales high.

9) Sum: Sums the values in a column used to add up numerical values such as sales amounts, quantities or profits.

10) SumX: Returns a specified result if an expression for each row summing the results. enables summation of complex expressions across rows, such as calculating total profits for each transaction

11) Switch: Evaluates an expression against a list of values and returns the corresponding result provides a simplified way to test multiple conditions like a case statement.

12) Discuss ten differences b/w calculated measure & calculated column.

Date: / /
Page No.:

Calculated Measure	Calculated Column
* Calculated on the fly based on context	* Row wise calculation stored in the model
* Based on filter and slicer context	* calculated for each row; independent of filters
* It not stored calculated dynamically	* stored in the data model.
* used in aggregations and summaries	* used for row level calculations or new dimensions
* Performance impact recalculated each time it is used	* Calculated once stored at load time.
* filter use cannot be used directly in filters	* can be used directly in filters & slicers.
* No impact on model structure	* Adds new columns to the model
* It is dynamic and changes with context.	* Static once calculated.
* It operates on aggregates or summaries	* Operates on individual values
* use func like sum() Average()	* often use IF() Calculation() etc.