Webscraping Scripts

All of the webscraping code can be found here.

- laptop_data = Title, Price, Rating, Touchscreen, Color, Ram
- gpu_model = GPU
- cpu_model = CPU
- reference_laptop_lenovo = Title, Price, Rating, Touchscreen, Color, Ram (Concatenated this at the end)

All Libraries needed:

In [ ]:
```
import requests
from bs4 import BeautifulSoup
import time # To add delay between requests
import pandas as pd
import os
```

Web scraping links from search results page: (Export: amazon_links_nosponsors.txt)

In [ ]:
```
# Grab URL from search results page

if __name__ == '__main__':

    # Headers to avoid being flagged as a bot
    HEADERS = ({
        'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36
        'Accept-Language': 'en-US, en;q=0.5'
    })

    # Base URL (first page of search results)
    base_url = 'https://www.amazon.com/s?k=gaming+laptop&rh=n%3A21512780011%2Cp_36%

    # List to store product links
    links_list = []
    current_url = base_url
    page_count = 0  # Counter to track the number of pages scraped

    while current_url and page_count < 5:  # Limit scraping to 5 pages
        print(f"Scraping URL: {current_url}")
        # Request to GET the data from the webpage
        webpage = requests.get(current_url, headers=HEADERS)

        # Check if GET request was successful
        if webpage.status_code == 200:
            # Parse the webpage
            soup = BeautifulSoup(webpage.content, 'html.parser')

            # Find the product links
```

```
            links = soup.find_all('a', attrs={'class': 'a-link-normal s-line-clamp-

            # Loop through all anchor tags and extract links
            for link in links:
                href = link.get('href') #get the href for all the links
                # Skip sponsored links
                if href and '/sspa/click' in href:  # Check for the sponsored segme
                    continue
                full_link = 'https://www.amazon.com' + href # this will complete th
                links_list.append(full_link) # Append the link to our list where ou

            # Increment page count
            page_count += 1
            print(f"Scraped page {page_count}")

            # Find the next page link
            next_page = soup.find('a', attrs={'class': 's-pagination-item s-paginat
            if next_page and 'href' in next_page.attrs:
                current_url = 'https://www.amazon.com' + next_page['href']
                time.sleep(2)  # Delay to avoid being blocked
            else:
                current_url = None  # No more pages available
        else:
            print(f"Failed to retrieve: {webpage.status_code}")
            break

    # Save the links to a file
    with open('amazon_links_nosponsors.txt', 'w') as file:
        for link in links_list:
            file.write(link + '\n')  # Write each link on a new line

    # Confirmation message
    print("Links saved to 'amazon_links_nosponsors.txt'")
```

Web scraping links product pages:

In [ ]:

```
# Webscrape: Products

# Grab the text file amazon_links_nosponsors.txt and run it through a script that w
def main(URL):

    # Create saved file to store gathered data, itca as a .csv

    with open('amazon_laptops_scraped.csv', 'a', encoding = 'utf-8') as File:

        HEADERS = ({  # User-Agent mimics a real browser
            'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/53
            'Accept-Language': 'en-US, en;q=0.5'
            })

        # Specify URL
        webpage = requests.get(URL, headers=HEADERS)

        # Creating the Soup Object containing all data
        soup = BeautifulSoup(webpage.content, "lxml")
```

```
        # scrape product title
        try:
            # Attempt to find the <span> element with an attribute id="productTitle
            title = soup.find("span", attrs={"id": 'productTitle'})
            # Get the string content (text) of the found <span> element
            title_value = title.string
            # Remove leading/trailing whitespace and replace commas with empty stri
            title_string = title_value.strip().replace(',', '')

        except AttributeError:
            # If an AttributeError occurs (e.g., element not found or no string con
            # Set the title_string to "None" (converted to NaN in DataFrames)
            title_string = None

        # scrape price
        try:
            # Find the element containing the whole number part of the price
            price_whole = soup.find("span", class_="a-price-whole")
            # Find the element containing the fractional part of the price
            price_fraction = soup.find("span", class_="a-price-fraction")
            if price_whole and price_fraction:
                # If both elements are found, concatenate their text content to for
                price = price_whole.get_text(strip=True) + price_fraction.get_text(
                # Introduce a delay (e.g., to avoid hitting request limits or mimic
                time.sleep(2)
            else:
                # If either part of the price is missing, set price to None
                price = None
        except AttributeError:
            # Handle cases where an AttributeError occurs (e.g., if elements are no
            price = None

        #scrape rating
        try:
            # Find the span using the 'data-hook' attribute
            rating = soup.find('span', attrs={"data-hook": "rating-out-of-text"})
            if rating:
                # Extract the text and split to get the numerical part (e.g., "4.4"
                rating_value = rating.get_text(strip=True).split(' ')[0]
            else:
                rating_value = None
        except AttributeError:
            rating_value = None

        # Scape to see if there Touchscreen in the title
        touchscreen = 'Yes' if title_string and 'touchscreen' in title_string.lower

        # scrape color
        # Initialize color as None to handle cases where the color cannot be found
        color = None
        try:
            # Find the <th> element with specific classes and containing the word '
            color_th = soup.find('th', class_="a-color-secondary a-size-base prodDe
            if color_th:
                # If the <th> element is found, find the next <td> element with the
```

```
                color_td = color_th.find_next('td', class_="a-size-base prodDetAttr
                if color_td:
                    # If the <td> element is found, extract its text content, remov
                    color = color_td.get_text(strip=True)
                    # Clean up the extracted color text by removing hyphens and ext
                    color = color.replace('-', '').strip()
        except AttributeError:
            # Handle cases where an AttributeError occurs by ensuring color remains
            color = None

        # Scrape Ram
        ram = None
        try:
        # Locate the <tr> element by its class
            ram_tr = soup.find('tr', class_="a-spacing-small po-ram_memory.installe
            if ram_tr:
        # Find the <td> with the class "a-span9" inside the row
                ram_td = ram_tr.find('td', class_="a-span9")
                if ram_td:
            # Extract the RAM value from the <span>
                    ram_span = ram_td.find('span', class_="a-size-base po-break-wor
                    if ram_span:
                        ram = ram_span.get_text(strip=True)
        except AttributeError:
            ram = None

        # Write headers only once (if the file is empty)
        File.seek(0, 2)  # Move to the end of the file
        if File.tell() == 0:  # If file is empty, write the header
            File.write("Title, Price, Rating, Touchscreen, Color, Ram\n")

        # saving data to csv
        File.write(f"{title_string}, {price}, {rating_value}, {touchscreen}, {color

        # Print results for verification
        print(f"Title: {title_string}")
        print(f"Price: {price}")
        print(f"Rating: {rating_value}")
        print(f"Touchscreen: {touchscreen}")
        print(f"Color: {color}")
        print(f"Ram: {ram}")
        time.sleep(2)

if __name__ == '__main__':
    #opening our url file to access URLs
    with open("amazon_links_nosponsors.txt", "r")  as file:
    #iterating over the urls
        for links in file.readlines():
            main(links.strip())
```

Web scraping GPU model from product pages (links):

In [ ]:

```
def scrape_gpu(product_url):

    with open('gpu_model_data.csv', 'a', encoding='utf-8') as File:
```

```
        HEADERS = ({
            'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/53
            'Accept-Language': 'en-US, en;q=0.5'
        })

        # Send an HTTP GET request to the given URL
        webpage = requests.get(product_url, headers=HEADERS)

        # Parse the HTML content with BeautifulSoup
        soup = BeautifulSoup(webpage.content, 'lxml')

        gpu_model = None

        try:
            # Locate the <th> with 'Graphics Coprocessor'
            gpu_th = soup.find('th', class_="a-color-secondary a-size-base prodDetS
                               string=lambda text: text and 'Graphics Coprocessor'
            if gpu_th:
                # Locate the corresponding <td> for the GPU
                gpu_td = gpu_th.find_next('td', class_="a-size-base prodDetAttrValu
                if gpu_td:
                    gpu_model = gpu_td.get_text(strip=True)
        except AttributeError:
            gpu_model = None

        # Remove the U+200E character if it exists
        if gpu_model:
            gpu_model = gpu_model.replace("\u200E", "")

        time.sleep(2)

        print('GPU:', gpu_model)

        # Save the GPU model to the file
        File.write(f'{gpu_model}\n')


if __name__ == '__main__':
    with open('amazon_links_nosponsors.txt', 'r') as file:
        for links in file.readlines():
            scrape_gpu(links.strip())

# Explore GPU model data
gpu = pd.read_csv('gpu_model_data.csv', names = ['gpu'])

# Export as .CSV with index as column for merging
gpu.to_csv('gpu_model.csv', index = True, index_label='index')
```

Explore GPU model data:

In [ ]:
```
# Explore GPU model data
gpu = pd.read_csv('gpu_model_data.csv', names = ['gpu'])
```

```
# Export as .CSV with index as column for merging
gpu.to_csv('gpu_model.csv', index = True, index_label='index')
```

Web scraping CPU model from product pages (links):

In [ ]:
```
def main(URL):
    # Create saved file to store gathered data, with UTF-8 encoding
    with open('cpu_model_data.csv', 'a', encoding='utf-8') as File:

        HEADERS = ({  # User-Agent mimics a real browser
            'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/53
            'Accept-Language': 'en-US, en;q=0.5'
            })

        # Specify URL
        webpage = requests.get(URL, headers=HEADERS)

        # Creating the Soup Object containing all data
        soup = BeautifulSoup(webpage.content, "lxml")

        # Try to find the 'th' with text containing 'CPU Model Number' more flexibl
        cpu_model = None  # Default to N/A if not found
        try:
            cpu_header = soup.find('th', class_='a-color-secondary a-size-base prod
            if cpu_header:
                # Get the next sibling td with the class 'a-size-base prodDetAttrVa
                cpu_td = cpu_header.find_next('td', class_='a-size-base prodDetAttr
                if cpu_td:
                    cpu_model = cpu_td.get_text(strip=True)
        except AttributeError:
            cpu_model = None

        time.sleep(2)

        print("CPU:", cpu_model)

        # Saving data to CSV
        File.write(f"{cpu_model}\n")

if __name__ == '__main__':
    # Opening our URL file to access URLs
    with open("amazon_links_nosponsors.txt", "r") as file:
        # Iterating over the URLs
        for links in file.readlines():
            main(links.strip())
```

Merging .CSVs for CPU model:

In [ ]:
```
# One webscrape would not work as not all results would return. Several scrapes nec

# Merge and Explore CPU model data

# Load the CPU model web scrape as a dictionary to load all files at once
CSVs = {
```

```
    'csv1': pd.read_csv('cpu_model_data.csv', header=None, encoding='utf-8'),
    'csv2': pd.read_csv('cpu_model_data_2.csv', header=None, encoding='utf-8'),
    'csv3': pd.read_csv('cpu_model_data_3.csv', header=None, encoding='utf-8'),
    'csv4': pd.read_csv('cpu_model_data_4.csv', header=None, encoding='utf-8'),
    'csv5': pd.read_csv('cpu_model_data_5.csv', header=None, encoding='utf-8'),
    'csv6': pd.read_csv('cpu_model_data_6.csv', header=None),
    'csv7': pd.read_csv('cpu_model_data_7.csv', header=None),
    'csv8': pd.read_csv('cpu_model_data_8.csv', header=None)
}

# Merge Dataframes
cpu = pd.concat([CSVs['csv1'], CSVs['csv2'], CSVs['csv3'], CSVs['csv4'], CSVs['csv5

# Fill N/A's with cpu name using bfill ( which takes the first valid value to the r
cpu = cpu.bfill(axis=1)

# Drop Columns, Keep first
cpu = cpu.iloc[:, :1]

# Rename column name
cpu.columns = ['cpu']

# Export as a .CSV file
cpu.to_csv('cpu_model.csv', index = True, index_label=['index']) #include index as
```

Web scraping reference computer product page:

In [ ]:

```
# Webscrape: Product (Reference Laptop)

def laptop(product_url):

    # Create saved file to store gathered data

    with open('reference_laptop.csv', 'a', encoding = 'utf-8') as File:

        HEADERS = ({  # User-Agent mimics a real browser
            'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/53
            'Accept-Language': 'en-US, en;q=0.5'
            })

        # Specify URL
        webpage = requests.get(product_url, headers=HEADERS)

        # Creating the Soup Object containing all data
        soup = BeautifulSoup(webpage.content, "lxml")

        # scrape title
        try:
            # Attempt to find the <span> element with an attribute id="productTitle
            title = soup.find("span", attrs={"id": 'productTitle'})
            # Get the string content (text) of the found <span> element
            title_value = title.string
            # Remove leading/trailing whitespace and replace commas with empty stri
            title_string = title_value.strip().replace(',', '')
```

```
        except AttributeError:
            # If an AttributeError occurs (e.g., element not found or no string con
            # Set the title_string to "NA" (Not Available)
            title_string = None
            # Print a message indicating the product title is unavailable


        # scrape price

        try:
            price_whole = soup.find("span", class_="a-price-whole")
            price_fraction = soup.find("span", class_="a-price-fraction")

            if price_whole and price_fraction:
                price = price_whole.get_text(strip = True) + price_fraction.get_tex
                time.sleep(2)
            else:
                price = None
        except AttributeError:
            price = None


        #scrape rating
        try:
            # Find the span using the 'data-hook' attribute
            rating = soup.find('span', attrs={"data-hook": "rating-out-of-text"})
            if rating:
                # Extract the text and split to get the numerical part (e.g., "4.4"
                rating_value = rating.get_text(strip=True).split(' ')[0]
            else:
                rating_value = None
        except AttributeError:
            rating_value = None


        # Scape to see if there Touchscreen in the title

        touchscreen = 'Yes' if title_string and 'touchscreen' and 'touch' in title_


        # scrape color
        color = None
        try:
            color_th = soup.find('th', class_="a-color-secondary a-size-base prodDe
            if color_th:
                color_td = color_th.find_next('td', class_="a-size-base prodDetAttr
                if color_td:
                    color = color_td.get_text(strip = True)
                    color = color.replace('-', '').strip()
        except AttributeError:
            color = None


        # Scrape Ram
        ram = None
        try:
```

```
            # Locate the <tr> element by its class
            ram_tr = soup.find('tr', class_="a-spacing-small po-ram_memory.installe
            if ram_tr:
            # Find the <td> with the class "a-span9" inside the row
                ram_td = ram_tr.find('td', class_="a-span9")
                if ram_td:
                    # Extract the RAM value from the <span>
                    ram_span = ram_td.find('span', class_="a-size-base po-break-wor
                    if ram_span:
                        ram = ram_span.get_text(strip=True)
        except AttributeError:
            ram = None


        #Scrape Gpu
        try:
            # Locate the <th> with 'Graphics Coprocessor'
            gpu_th = soup.find('th', class_="a-color-secondary a-size-base prodDetS
                               string=lambda text: text and 'Graphics Coprocessor'
            if gpu_th:
                # Locate the corresponding <td> for the GPU
                gpu_td = gpu_th.find_next('td', class_="a-size-base prodDetAttrValu
                if gpu_td:
                    gpu = gpu_td.get_text(strip=True)
        except AttributeError:
            gpu = None


        # Remove the U+200E character if it exists
        if gpu:
            gpu = gpu.replace("\u200E", "")


        # Try to find the 'th' with text containing 'CPU Model Number' more flexibl
        cpu = None  # Default to N/A if not found
        try:
            cpu_header = soup.find('th', class_='a-color-secondary a-size-base prod
            if cpu_header:
                # Get the next sibling td with the class 'a-size-base prodDetAttrVa
                cpu_td = cpu_header.find_next('td', class_='a-size-base prodDetAttr
                if cpu_td:
                    cpu = cpu_td.get_text(strip=True)
        except AttributeError:
            cpu = None

        # Write headers only once (if the file is empty)
        File.seek(0, 2)  # Move to the end of the file
        if File.tell() == 0:  # If file is empty, write the header
            File.write("title,price,rating,touchscreen,color,ram,gpu,cpu\n")

        # saving data to csv
        File.write(f"{title_string},{price},{rating_value},{touchscreen},{color},{r

        # Print results for verification
        print(f"Title: {title_string}")
        print(f"Price: {price}")
```

```
        print(f"Rating: {rating_value}")
        print(f"Touchscreen: {touchscreen}")
        print(f"Color: {color}")
        print(f"Ram: {ram}")
        print(f"Gpu: {gpu}")
        print(f"Cpu: {cpu}")
        time.sleep(2)

if __name__ == '__main__':
    #opening our url file to access URLs
    with open("reference_laptop_link.txt", "r")  as file:
    #iterating over the urls
        for links in file.readlines():
            laptop(links.strip())
```

Clean before merging: (reference laptop)

In [ ]:
```
# Clean before merging

# Convert to a dataframe
reference = pd.read_csv('reference_laptop.csv', index_col=None)

# Ensure consistent column names
reference.columns = reference.columns.str.lower()

# Add link column to df
link = ['https://www.amazon.com/Lenovo-i7-12700H-Fingerprint-Long-Lasting-Charging/

reference['link'] = link

# Convert the reference link column to HTML anchor tags (clickable links)
reference['link'] = reference['link'].apply(lambda x: f'<a href="{x}" target="_blan

# Export as .csv
reference.to_csv('reference_laptop_lenovo.csv')
```

Merging all the scraped data:

In [ ]:
```
# Load CSVs
cpu_model = pd.read_csv('cpu_model.csv')
gpu_model = pd.read_csv('gpu_model.csv')
laptop_data = pd.read_csv('laptop_data.csv')

# Data exploration: pre-merge
laptop_data.head()
cpu_model.head()
gpu_model.head()

cpu_model.info()
gpu_model.info()
laptop_data.info()

# Begin merging the dataset
product_data = laptop_data.merge(cpu_model, on = 'index', how = 'left').merge(gpu_m
```

```
# Explore merged data
product_data
```

Adding column 'links' to dataframe:

In [ ]:
```
# Add links to dataframe. (amazon_links_nosponsor.txt)
links_path = r'C:\Users\abrah\OneDrive\Documents\Data Projects\Web Scraping Amazon

# Open .txt file
with open(links_path, 'r') as file:
    links = file.readlines()

# Stripping each element in the list
links = [link.strip() for link in links]

# Create df with column name
links_data = pd.DataFrame(links, columns = ['link'])

# Adds index as a column
links_data = links_data.reset_index()

# Export to .csv
links_data.to_csv('links_data.csv', index = False)

# Explore df
links_data

# Convert the 'product_link' column to HTML anchor tags (clickable links)
links_data = links_data['link'].apply(lambda x: f'<a href="{x}" target="_blank">{x}

# merge link_data with dataframe
dataframe_final = product_data.merge(links_data, on = 'index', how = 'left')
```

Clean the newly merged dataframe and export final as .csv:

In [ ]:
```
# Create a copy
df = dataframe_final.copy()

# Look for duplicates specific to the title column (due to us merging links, all li
duplicates = df[df.duplicated(subset='title', keep=False)] # keep=False: Marks all

# View duplicates
duplicates.duplicated().value_counts() # 48 total duplicates

# Remove duplicates ('title' column) and keep only the first occurrence, route back
df = df.drop_duplicates(subset='title', keep='first')

#Look at dtypes and columns
df.info()

# remove whitespaces from column names
df.columns = df.columns.str.replace(' ', '')
```

```
# Delete index column and Unamed: 0 column
del df['index']
del laptop_data['Unnamed: 0']

# Clean 'color' column special characters [U200E] (We did this for the gpu_model to
df.loc[:, 'color'] = df['color'].str.replace('\u200e', '', regex=True)

# This will allow us to load the df with clickable links!!!
# CSS style to reduce the font size of the links
style = """
    <style>
        a {
            font-size: .5px; /* You can adjust this size */
        }
    </style>
"""
# Display the DataFrame with clickable links
from IPython.display import HTML
HTML(style + df.to_html(escape=False))

# Add the laptop we are referencing
reference_laptop = pd.read_csv('reference_laptop_lenovo.csv')
# Concat with dataframe
df = pd.concat([df, reference_laptop], axis = 0, ignore_index=True)
# Delete column (loaded again because of concat)
del df['Unnamed: 0']

# Convert proper datatypes
# Strip any leading/trailing spaces and then convert to numeric, replacing invalid
df['rating'] = pd.to_numeric(df['rating'].str.strip(), errors='coerce')
# Convert rating to float dtype
df['rating'] = df['rating'].astype('float')

# Export to .csv (no index)
df.to_csv('final_laptop_data.csv', index = False)
```

Overview:

We successfully created a complete dataset named final_laptop_data.csv, containing web-scraped data from Amazon to help my cousin find the perfect laptop. The dataset includes 33 rows and 9 columns, capturing the following details:

Columns: Title, Price, Rating, Touchscreen, Color, Ram, GPU, CPU and Link

Purpose: The goal of this project was to compare laptops similar to a specific model my cousin liked, while exploring other options within her price range that might offer better value or features.

Source: All laptops were sourced from Amazon's listings through web scraping.

Considerations: While the data is comprehensive, it may require cleaning during the analysis phase to address any inconsistencies or missing values. Some limitations exist and can be

addressed with further cleaning or webscraping with the scripts provided.