## **Redbus Data Scraping with Selenium & Dynamic Filtering using Streamlit**

### **1. Introduction**

**Project Overview:**

This project involves scraping data from the Redbus website using Selenium, storing the scraped data in an SQLite database, and developing a Streamlit application to dynamically filter and display the data. The goal is to automate the data extraction process and provide a user-friendly interface for data exploration.

**Objectives:**

* To scrape bus route details from Redbus.
* To store the scraped data in an SQLite database.
* To create a Streamlit application for dynamic filtering and visualization of the data.

### **2. Tools and Technologies**

* **Python:** For scripting and data manipulation.
* **Selenium:** For web scraping.
* **MySQL:** For data storage.
* **Streamlit:** For creating the web application.
* **Jupyter Notebook:** For development and testing.
* **Visual studio :** For development and testing.(streamlit)

### **3. Project Setup**

**Prerequisites:**

* Python installed on your machine.
* Required Python libraries: selenium, streamlit, pandas, mysql.
* Chrome WebDriver for Selenium.

**Installation:**

pip install selenium pandas pymysql streamlit

### 

### **4. Web Scraping with Selenium**

The code performs web scraping of bus transport data from the Redbus website and stores the scraped data in a MySQL database. It utilizes the Selenium library to interact with the web pages, extract relevant information, and handle dynamic content. The pymysql library is used to connect to the MySQL database and store the scraped data.

### **Web Scraping Process**

1. Initialization:

The Chrome WebDriver is initialized and maximized to ensure proper rendering of the web pages.

# Initialize the WebDriver

driver = webdriver.Chrome()

driver.maximize\_window()

1. Navigating to the Redbus Website:

The WebDriver navigates to the Redbus homepage and waits for the page to load.

# Open Redbus website

driver.get("https://www.redbus.in")

time.sleep(5) # Wait for the page to load

1. Clicking the 'View All' Button:

The script locates and clicks the 'View All' button in the Government Bus Corporations section to access the list of bus corporations.

# Click 'View All' button in the Government Bus Corporations section

view\_all\_button = driver.find\_element(By.XPATH, '//\*[@id="homeV2-root"]/div[3]/div[1]/div[2]/a')

view\_all\_button.click()

time.sleep(5)

1. Switching to the New Tab:

The script switches to the new tab that opens after clicking the 'View All' button.

# Switch to the new tab

driver.switch\_to.window(driver.window\_handles[1])

1. Scrolling to the Bottom of the Page:

The script scrolls to the bottom of the page to ensure all bus corporations are loaded.

* # Scroll down to the bottom of the page
* last\_height = driver.execute\_script("return document.body.scrollHeight")
* while True:
* driver.execute\_script("window.scrollTo(0, document.body.scrollHeight);")
* time.sleep(5) # Wait to load the page
* new\_height = driver.execute\_script("return document.body.scrollHeight")
* if new\_height == last\_height:
* break
* last\_height = new\_height

1. Selecting a Bus Corporation:

The script selects a specific bus corporation (e.g., KSRTC) and clicks on its link.

# Select a bus corporation (Example: KSRTC)

try:

bus\_corp = WebDriverWait(driver, 10).until(

EC.element\_to\_be\_clickable((By.XPATH, '//\*[@id="root"]/div/article[2]/div/div/ul[3]/li[3]/a'))

)

# Scroll the element into view

driver.execute\_script("arguments[0].scrollIntoView(true);", bus\_corp)

time.sleep(2)

# Click the bus corporation link

bus\_corp.click()

time.sleep(5)

except ElementClickInterceptedException:

print("Element is not clickable, trying again...")

bus\_corp.click()

time.sleep(5)

1. Scraping Route Names and Links:

The script scrapes the route names and their links by manually clicking through each page and extracting the data.

# Initialize list to store routes

routes = []

# Loop to manually click each page and scrape data

while True:

input("Please click the next page number button manually, wait for the page to load, then press Enter to confirm...")

# Scrape route names and their links

route\_elements = driver.find\_elements(By.CLASS\_NAME, 'route')

for route\_element in route\_elements:

route = route\_element.text # Get the route name

route\_link = route\_element.get\_attribute('href') # Get the route link

routes.append((route, route\_link))

# Check if there are more pages to click

more\_pages = input("Are there more pages to click? (yes/no): ")

if more\_pages.lower() != 'yes':

break

# Print the scraped routes to view them

print("Scraped Routes and Links:")

for route, link in routes:

print(f"Route: {route}, Link: {link}")

1. Scraping Bus Details:

The script iterates through each route link to scrape detailed bus information such as bus name, type, departing and reaching times, star rating, price, and seat availability.

* # Now use the scraped routes and links to get bus details
* bus\_details = []
* for route, route\_link in routes:
* driver.get(route\_link)
* time.sleep(5) # Wait for the page to load
* # Click the 'View Buses' button if present
* try:
* view\_buses\_button = driver.find\_element(By.CLASS\_NAME, 'button')
* view\_buses\_button.click()
* time.sleep(5)
* except NoSuchElementException:
* pass # No 'View Buses' button present
* # Scroll down to the bottom of the page
* last\_height = driver.execute\_script("return document.body.scrollHeight")
* while True:
* driver.execute\_script("window.scrollTo(0, document.body.scrollHeight);")
* time.sleep(2) # Wait to load the page
* new\_height = driver.execute\_script("return document.body.scrollHeight")
* if new\_height == last\_height:
* break
* last\_height = new\_height
* try:
* bus\_elements = driver.find\_elements(By.CSS\_SELECTOR, "div.bus-item")
* except NoSuchElementException:
* print("No bus elements found")
* continue
* for bus in bus\_elements:
* try:
* busname = bus.find\_element(By.CSS\_SELECTOR, "div.travels.lh-24.f-bold.d-color").text
* except NoSuchElementException:
* busname = "N/A"
* try:
* bustype = bus.find\_element(By.CSS\_SELECTOR, "div.bus-type.f-12.m-top-16.l-color.evBus").text
* except NoSuchElementException:
* bustype = "N/A"
* try:
* departing\_time = bus.find\_element(By.CSS\_SELECTOR, "div.dp-time.f-19.d-color.f-bold").text
* departing\_time\_dt = convert\_to\_datetime(departing\_time, datetime.now())
* except NoSuchElementException:
* departing\_time\_dt = None
* try:
* duration = bus.find\_element(By.CSS\_SELECTOR, "div.dur.l-color.lh-24").text
* except NoSuchElementException:
* duration = "N/A"
* try:
* reaching\_time = bus.find\_element(By.CSS\_SELECTOR, "div.bp-time.f-19.d-color.disp-Inline").text
* reaching\_time\_dt = convert\_to\_datetime(reaching\_time, datetime.now())
* if reaching\_time\_dt and departing\_time\_dt and reaching\_time\_dt < departing\_time\_dt:
* reaching\_time\_dt += timedelta(days=1)
* except NoSuchElementException:
* reaching\_time\_dt = None
* try:
* star\_rating = bus.find\_element(By.CSS\_SELECTOR, "div.rating-sec.lh-24").text
* star\_rating = float(star\_rating) if star\_rating != "N/A" else 0.0
* except NoSuchElementException:
* star\_rating = 0.0
* try:
* price = bus.find\_element(By.CSS\_SELECTOR, "span.f-19.f-bold").text
* price = float(price.replace('₹', '').replace(',', '').strip()) if price != "N/A" else None
* except NoSuchElementException:
* price = None
* try:
* seats\_available = bus.find\_element(By.CSS\_SELECTOR, "div.seat-left.m-top-16").text
* seats\_available = int(seats\_available.split()[0]) if seats\_available != "N/A" else 0
* except NoSuchElementException:
* seats\_available = 0
* bus\_details.append((route, route\_link, busname, bustype, departing\_time\_dt, duration, reaching\_time\_dt, star\_rating, price, seats\_available))
* # Print the scraped bus details to view them
* print("Scraped Bus Details:")
* for detail in bus\_details:
* print(detail)

### **SQL Data Storage Process**

1. Connecting to the MySQL Database:

The script establishes a connection to the MySQL database using pymysql.

* # Connect to the MySQL database
* conn = pymysql.connect(
* host='127.0.0.1',
* user='root',
* passwd='sripathi12345678',
* db='redbus\_data' # Make sure the database exists
* )
* cursor = conn.cursor()

1. Creating the Database Schema:

The script creates a table named bus\_routes if it doesn't already exist. The table schema is designed to accommodate the scraped data, with appropriate data types for each column.

* # Create the bus\_routes table if it doesn't exist
* cursor.execute('''
* CREATE TABLE IF NOT EXISTS bus\_routes (
* id INT AUTO\_INCREMENT PRIMARY KEY,
* route\_name TEXT,
* route\_link TEXT,
* busname TEXT,
* bustype TEXT,
* departing\_time DATETIME,
* duration TEXT,
* reaching\_time DATETIME,
* star\_rating FLOAT,
* price DECIMAL(10, 2),
* seats\_available INT
* )
* ''')
* Explanation of Table Columns:
  + id: An auto-incrementing primary key to uniquely identify each record.
  + route\_name: The name of the bus route.
  + route\_link: The URL link to the bus route page.
  + busname: The name of the bus operator.
  + bustype: The type of bus (e.g., AC, Non-AC, Sleeper).
  + departing\_time: The departure time of the bus, stored in DATETIME format.
  + duration: The duration of the bus journey.
  + reaching\_time: The arrival time of the bus, stored in DATETIME format.
  + star\_rating: The star rating of the bus, stored as a FLOAT.
  + price: The price of the bus ticket, stored as a DECIMAL with precision up to two decimal places.
  + seats\_available: The number of seats available on the bus, stored as an INT.
* d) Inserting Data into the Database:

The script iterates through the list of bus details (bus\_details) and inserts each record into the bus\_routes table.

* # Insert scraped data into the bus\_routes table
* for detail in bus\_details:
* cursor.execute('''
* INSERT INTO bus\_routes (
* route\_name, route\_link, busname, bustype, departing\_time,
* duration, reaching\_time, star\_rating, price, seats\_available
* ) VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s, %s)
* ''', detail)
* Explanation of Insertion Process:
  + The cursor.execute method is used to execute the SQL INSERT statement for each record in the bus\_details list.
  + The %s placeholders are used to safely insert the data into the SQL query, preventing SQL injection attacks.
  + Each record from the bus\_details list is unpacked and inserted into the corresponding columns of the bus\_routes table.
* e) Committing the Transaction and Closing the Connection:

After inserting all the data, the transaction is committed to the database to ensure the data is saved.

* # Commit the transaction and close the connection
* conn.commit()

The database connection is then closed to free up resources.

* conn.close()
* f) Printing Confirmation Message:

The script prints a confirmation message indicating that the data has been successfully saved to the database.

* print("Data has been successfully saved to the database.")

### **5. Streamlit Application**

The provided code is a Streamlit application designed to fetch bus transport data from a MySQL database, allow users to filter the data based on various criteria, and display the filtered data. The application also provides a download button to export the filtered data as a CSV file. Below is a detailed explanation of the code and its functionality.

**Creating the Streamlit App:**

1. Importing Required Libraries:

import streamlit as st

import pandas as pd

import mysql.connector

streamlit: The main library used to create the interactive web application.

pandas: A library used for data manipulation and analysis.

1. Database Connection Using mysql: password, host, port, and database name.
2. Fetching Data from the Database:

query = "SELECT \* FROM bus\_routes"

data = pd.read\_sql(query, engine)

The pd.read\_sql function is used to execute the SQL query and fetch the data from the bus\_routes table into a pandas DataFrame named data.

1. Streamlit App Layout:

st.title('Redbus Routes Data Filtering and Analysis')

This line sets the title of the Streamlit application.

1. Filters:

bustype\_filter = st.multiselect('Select Bus Type:', options=data['bustype'].unique())

route\_filter = st.multiselect('Select Route:', options=data['route\_name'].unique())

price\_filter = st.slider('Select Price Range:', min\_value=int(data['price'].min()), max\_value=int(data['price'].max()), value=(int(data['price'].min()), int(data['price'].max())))

star\_filter = st.slider('Select Star Rating Range:', min\_value=float(data['star\_rating'].min()), max\_value=float(data['star\_rating'].max()), value=(float(data['star\_rating'].min()), float(data['star\_rating'].max())))

availability\_filter = st.slider('Select Seat Availability Range:', min\_value=int(data['seats\_available'].min()), max\_value=int(data['seats\_available'].max()), value=(int(data['seats\_available'].min()), int(data['seats\_available'].max())))

Multiselect Filters:

* bustype\_filter: Allows users to select multiple bus types from a dropdown.
* route\_filter: Allows users to select multiple routes from a dropdown.

Slider Filters:

* price\_filter: Allows users to select a price range using a slider.
* star\_filter: Allows users to select a star rating range using a slider.
* availability\_filter: Allows users to select a seat availability range using a slider.

1. Filtering Data Based on User Inputs:

filtered\_data = data

if bustype\_filter:

filtered\_data = filtered\_data[filtered\_data['bustype'].isin(bustype\_filter)]

if route\_filter:

filtered\_data = filtered\_data[filtered\_data['route\_name'].isin(route\_filter)]

filtered\_data = filtered\_data[(filtered\_data['price'] >= price\_filter[0]) & (filtered\_data['price'] <= price\_filter[1])]

filtered\_data = filtered\_data[(filtered\_data['star\_rating'] >= star\_filter[0]) & (filtered\_data['star\_rating'] <= star\_filter[1])]

filtered\_data = filtered\_data[(filtered\_data['seats\_available'] >= availability\_filter[0]) & (filtered\_data['seats\_available'] <= availability\_filter[1])]

* This code filters the original DataFrame data based on the user's selections.
* The isin method is used to filter rows based on the selected bus types and routes.
* The between condition is used to filter rows based on the selected price range, star rating range, and seat availability range.

1. Displaying Filtered Data:

st.write('Filtered Data:')

st.dataframe(filtered\_data)

This code displays the filtered data in the Streamlit application.

1. Download Button:

if not filtered\_data.empty:

st.download\_button(

label="Download Filtered Data",

data=filtered\_data.to\_csv(index=False),

file\_name="filtered\_data.csv",

mime="text/csv"

)

else:

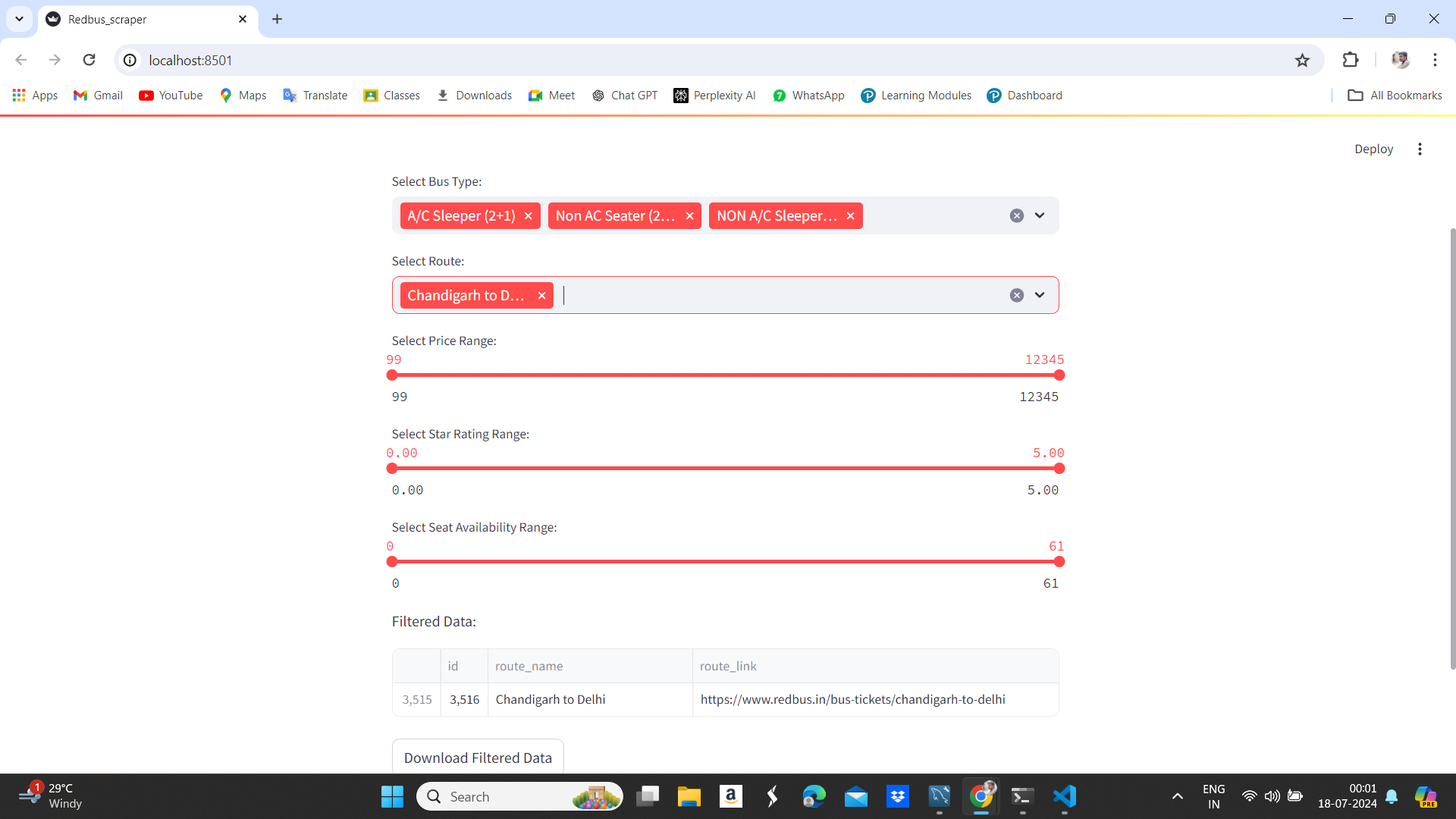
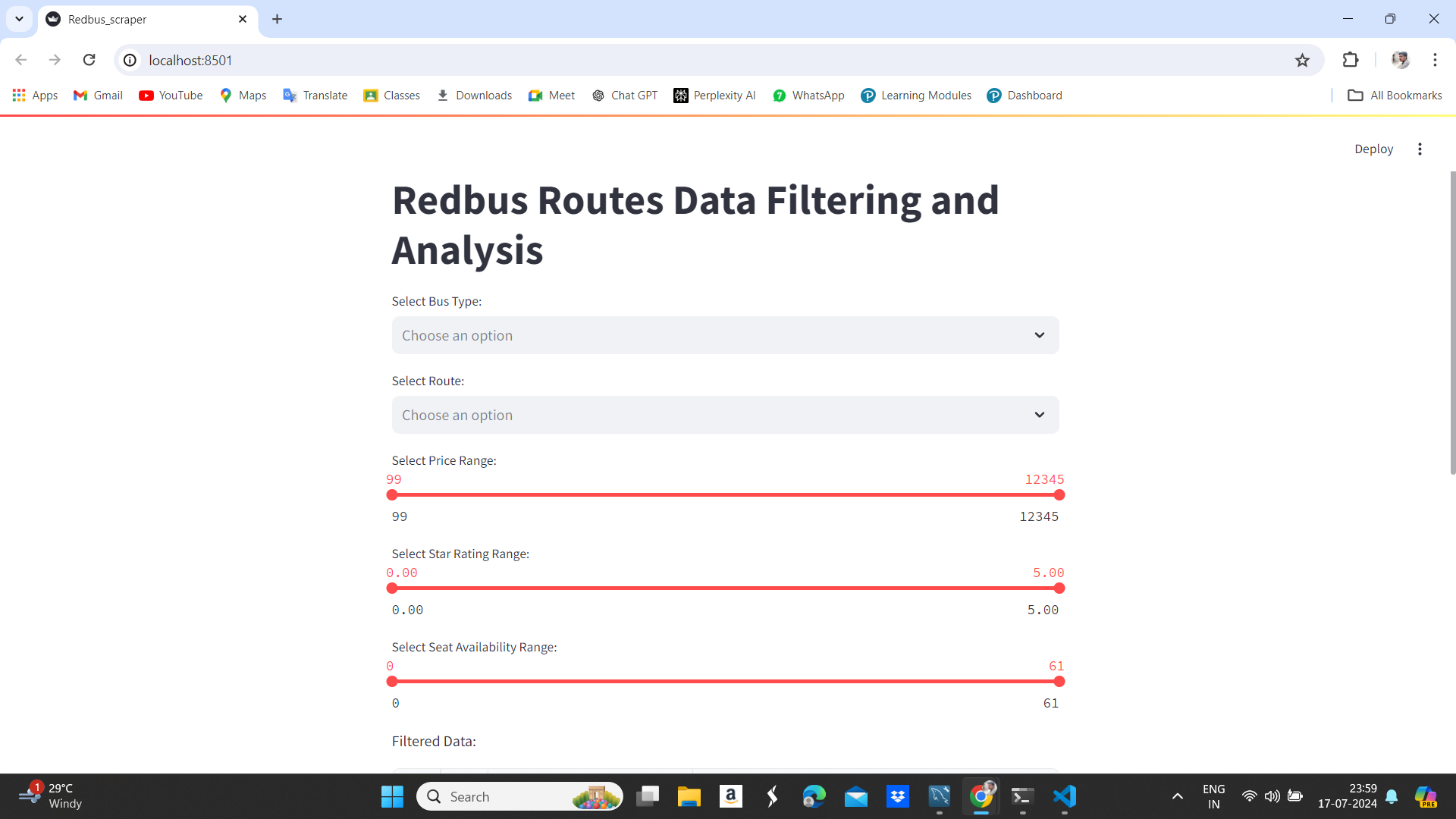
st.warning("No data available with the selected filters.")

This code adds a download button to the Streamlit application, allowing users to download the filtered data as a CSV file. If the filtered data is empty (i.e., no data matches the selected filters), a warning message is displayed instead.

**Running the Streamlit App:**

streamlit run your\_script\_name.py

**Screenshots :**

****

**Link :**

**Local URL:** [**http://localhost:8501**](http://localhost:8501)

**Network URL:** [**http://192.168.1.3:8501**](http://192.168.1.3:8501)

### **6. Results**

**Expected Outcomes:**

* Successfully scrape a minimum of 10 Government State Bus Transport data from Redbus website using Selenium. Also include the private bus information for the selected routes.
* Store the data in a structured SQL database.
* Develop an interactive Streamlit application for data filtering.
* Ensure the application is user-friendly and efficient.

### **7. Project Evaluation Metrics**

**Data Scraping Accuracy:**

* Completeness and correctness of the scraped data.

**Database Design:**

* Effective and efficient database schema.

**Application Usability:**

* User experience and ease of use of the Streamlit application.

**Filter Functionality:**

* Effectiveness and responsiveness of data filters.

**Code Quality:**

* Adherence to coding standards and best practices.

### **8. Technical Tags**

* Web Scraping
* Selenium
* Streamlit
* SQL
* Data Analysis
* Python
* Interactive Application

### **9. Conclusion**

**Summary:** Summarize the project, the process of scraping data, storing it, and displaying it using Streamlit.

**Future Work:**

* Improvements in data scraping.
* Adding more features to the Streamlit app.

### **10. References**

* Links to resources and documentation used in the project.

[Selenium Documentation](https://www.selenium.dev/documentation/)

[Streamlit Documentation](https://docs.streamlit.io/)

[PyMySQL Documentation](https://pymysql.readthedocs.io/en/latest/)