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 a MYSQL 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 MYSQL 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.
- **Pycharm:** For development and testing.(streamlit)

3. Project Setup

Prerequisites:

- Python – IDE installed on your machine.
- Required Python libraries: selenium, streamlit, pandas, pymysql.
- 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

a) 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()
```

b) Defining the functions for extracting routes:

```
def collect_routes_from_page(driver):
    route_names = []
    route_links = []
    routes = driver.find_elements(By.XPATH, "//a[@class='route']")
    for route in routes:
        route_names.append(route.get_attribute('title'))
        route_links.append(route.get_attribute('href'))
    return route_names, route_links
```

c) Defining a function to extraxt bus details and Storing the Route names and links to a list:

```
def extract_bus_details(driver, route_name, route_link):
    bus_details_list = []
    time.sleep(5)  # Consider replacing this with a wait for specific
elements

    bus_items = driver.find_elements(By.XPATH, '//div[contains(@class,
"bus-item")]')

    for bus_item in bus_items:
        bus_details = {
            "Bus Name": bus_item.find_element(By.XPATH,
'.//div[contains(@class, "travels")]').text.strip() or 'N/A',
            "Bus Type": bus_item.find_element(By.XPATH,
'.//div[contains(@class, "bus-type")]').text.strip() or 'N/A',
            "Start of Journey": bus_item.find_element(By.XPATH,

'.//div[contains(@class, "dp-time")]').text.strip() or 'N/A',
            "End of Journey": bus_item.find_element(By.XPATH,

'.//div[contains(@class, "bp-time")]').text.strip() or 'N/A',
            "Duration": bus_item.find_element(By.XPATH,
'.//div[contains(@class, "dur")]').text.strip() or 'N/A',
            "Price": bus_item.find_element(By.XPATH,
                                                         './/div[contains(@class,
"fare")]//span[contains(@class, "f-19 f-bold")]').text.strip() or 'N/A',
            "Star Rating": bus_item.find_element(By.XPATH,
                                                               './/div[contains(@class,
"rating")]//span').text.strip() or 'N/A',
            "Seat Availability": bus_item.find_element(By.XPATH,

'.//div[contains(@class, "seat-left")]').text.strip() or 'N/A',
            "Route Name": route_name,
            "Route Link": route_link
        }
        bus_details_list.append(bus_details)

    return bus_details_list
```

d) Initiating the chrome driver and loading each state's url to the chrome driver:

```
e) driver = webdriver.Chrome()

   try:
       # Open the desired URL
       driver.get("https://www.redbus.in/online-booking/rsrtc")
       wait = WebDriverWait(driver, 20)

       # Wait for the pagination container element
       pagination_container =
   wait.until(EC.presence_of_element_located((By.CLASS_NAME,
   'DC_117_paginationTable')))

       all_route_names = []
       all_route_links = []

       # Iterate through each page
       for page in range(1, 3):  # Adjust the range based on your
   requirement
           xpath_expression = f'//div[contains(@class, "DC_117_pageTabs")
   and contains(text(), "{page}")]'
           page_button = pagination_container.find_element(By.XPATH,
   xpath_expression)

           actions = ActionChains(driver)
           actions.move_to_element(page_button).perform()
           page_button.click()
           time.sleep(3)  # Wait for the new page to load

           # Collect routes from the current page
           route_names, route_links = collect_routes_from_page(driver)
           all_route_names.extend(route_names)
           all_route_links.extend(route_links)

       # Container for all bus details
       all_bus_details = []
```

```
    for route_link, route_name in zip(all_route_links,
all_route_names):
        driver.get(route_link)
        driver.maximize_window()
        time.sleep(2)

        # Click all "View Buses" buttons
        try:
            view_buses_buttons = wait.until(
                EC.presence_of_all_elements_located(
                    (By.XPATH, "//div[@class='button' and
contains(text(),'View Buses')]"))
            )
            time.sleep(5)

            for button in reversed(view_buses_buttons):
                try:

driver.execute_script("arguments[0].scrollIntoView(true);", button)
                    time.sleep(1)
                    button.click()
                    time.sleep(2)
                except Exception as e:
                    print(f"Error clicking button: {e}")
                    continue
        except Exception as e:
            print(f"Error during 'View Buses' button processing: {e}")

        # Scroll to the bottom of the page to ensure all buses are
loaded
        scroll_pause_time = 2
        last_height = driver.execute_script("return
document.body.scrollHeight")
        while True:
            driver.execute_script("window.scrollTo(0,
document.body.scrollHeight);")
            time.sleep(scroll_pause_time)

            new_height = driver.execute_script("return
document.body.scrollHeight")
            if new_height == last_height:
                break
            last_height = new_height

        # Extract bus details after all content is loaded
        try:
            bus_details = extract_bus_details(driver, route_name,
route_link)
            all_bus_details.extend(bus_details)
        except Exception as e:
            print(f"Error extracting bus details for route
{route_name}: {e}")

finally:
    # Close the browser
    driver.quit()
```

This script locates all the routes in a state . and extracts the bus details in each route. The loop is iterated to extract the details from all the pages within the range.

f) Scrolling to the Bottom of the Page:

```
scroll_pause_time = 2
last_height = driver.execute_script("return document.body.scrollHeight")
while True:
    driver.execute_script("window.scrollTo(0, document.body.scrollHeight);")
    time.sleep(scroll_pause_time)

    new_height = driver.execute_script("return document.body.scrollHeight")
    if new_height == last_height:
        break
    last_height = new_height
```

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

g) Selecting a Bus Corporation:

```
try:
    bus_details = extract_bus_details(driver, route_name, route_link)
    all_bus_details.extend(bus_details)
except Exception as e:
    print(f"Error extracting bus details for route {route_name}: {e}")
```

h) All the extracted details are converted to data frame.

```
# Convert bus details to a DataFrame and remove duplicates
df = pd.DataFrame(all_bus_details)
# Display the DataFrame
print(df.head())
```

i) Connecting mysql using pymusql:

```
user = 'root'
password = 'Prad@123'
host = '127.0.0.1'
database = 'red'

# Connect to the MySQL database
conn = mysql.connector.connect(user=user, password=password, host=host,
database=database)
cursor = conn.cursor()
```

j) Creating the Database Schema:

```
k)  create_table_query = """
    CREATE TABLE IF NOT EXISTS RSRTC_Bus_Details(
        id INT AUTO_INCREMENT PRIMARY KEY,
        Bus_Name VARCHAR(100),
        Bus_Type VARCHAR(100),
        Start_of_Journey VARCHAR(100),
        End_of_Journey VARCHAR(100),
        Duration VARCHAR(100),
        Price FLOAT,
        Star_Rating FLOAT,
        Seat_Availability VARCHAR(100),
        Route_Name VARCHAR(255),
        Route_link VARCHAR(255)
    )
    """
    cursor.execute(create_table_query)
```

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.

k) 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.

```
for index, row in df.iterrows():
    try:
        insert_query = """
            INSERT INTO RSRTC_Bus_Details (
                Bus_Name, Bus_Type, Start_of_Journey, End_of_Journey,
Duration, Price, Star_Rating, Seat_Availability, Route_Name, Route_link
            ) VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s, %s)
        """
        price = float(row["Price"].replace(",", "")) if row["Price"]
not in ['N/A', ''] else None
        star_rating = float(row["Star Rating"]) if row["Star Rating"]
not in ['N/A', ''] else None
```

```
            cursor.execute(insert_query, (
                row["Bus Name"], row["Bus Type"], row["Start of Journey"],
    row["End of Journey"], row["Duration"],
                price, star_rating,
                row["Seat Availability"], row["Route Name"], row["Route
    Link"]
            ))
        except Exception as e:
            print(f"Error inserting row {index}: {e}")

    # Commit and close the connection
    conn.commit()
    cursor.close()
    conn.close()
```

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.

l) 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 and close the connection
conn.commit()
cursor.close()
conn.close()
```

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.

Creating the Streamlit App:

a) Importing Required Libraries:

```
b)  import streamlit as st
    import pandas as pd
    import pymysql
    import re  # Import regular expressions module for pattern matching
    import requests  # Import requests for Lottie animation loading
    from streamlit_lottie import st_lottie
```

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

pymysql: A library used to interact with the MySQL database.

pandas: A library used for data manipulation and analysis.

c) Fetching Data from the Database:

```
d) user = 'root'
   password = 'Prad@123'
   host = '127.0.0.1'
   database = 'red'

   # Connect to MySQL database
   conn = pymysql.connect(
       user=user,
       password=password,
       host=host,
       database=database
   )
```

e) Filters:

```
f) st.sidebar.success("Select a page from above")
   rtc_options = [
       'APSRTC_Bus_Details',
       'ASTC_Bus_Details',
       'BSRTC_Bus_Details',
       'HRTC_Bus_Details',
       'KSRTC_Bus_Details',
       'KTCL_Bus_Details',
       'PEPSU_Bus_Details',
       'SBSTC_Bus_Details',
       'RSRTC_Details',
       'UPSRTC_Bus_Details'
   ]
   selected_rtc = st.selectbox('Select RTC:', rtc_options)
   # Construct SQL query based on selected RTC
   query = f"SELECT * FROM {selected_rtc}"
 try:
    df = pd.read_sql(query, conn)
except Exception as e:
    st.error(f"Error fetching data: {str(e)}")
    df = pd.DataFrame()  # Empty DataFrame in case of error

# Display RTC data table
st.subheader(f'Table for {selected_rtc}')
st.write(df)
```

this script selects the bus details of different states based on the input.

g) Multiple selectors:

```
selected_routes = st.multiselect('Select Bus Route(s):',
df['Route_Name'].unique() if not df.empty else [])
selected_bus_types = st.multiselect('Select Bus Type(s):',
df['Bus_Type'].unique() if not df.empty else [])

# Calculate min and max price range from the data
min_price = int(df['Price'].min()) if not df.empty else 0
max_price = int(df['Price'].max()) if not df.empty else 1000
price_range = st.slider('Price Range:', min_value=min_price,
max_value=max_price, value=(min_price, max_price))

star_rating = st.slider('Star Rating:', min_value=1.0, max_value=5.0,
step=0.1, value=(1.0, 5.0))
seats_available = st.slider('Seats Available:', min_value=0, max_value=100,
value=(0, 100))

# Apply filters and fetch data based on user selections
filters = []
```

h) fetch data based on input

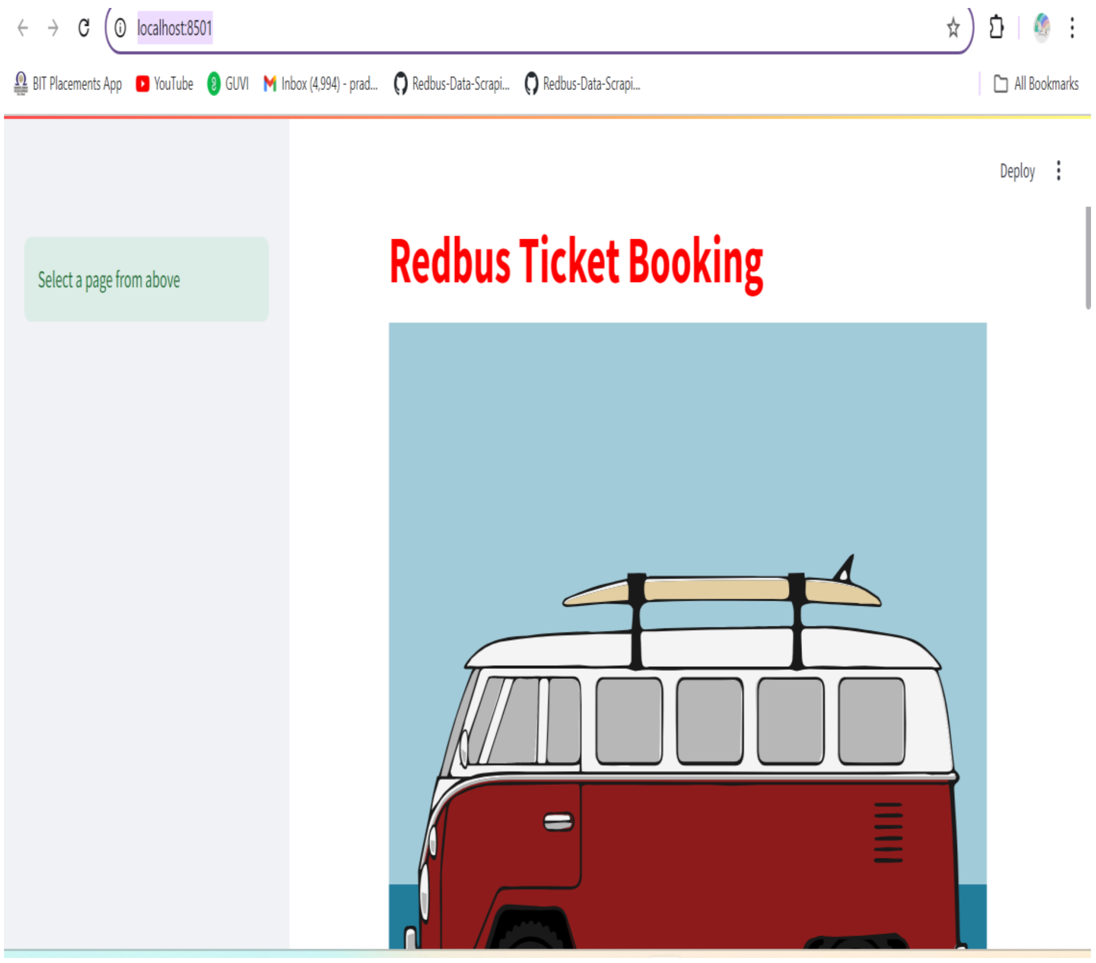
```
if selected_routes:
    route_conditions = " OR ".join([f"Route_Name = '{route}'" for route in
selected_routes])
    filters.append(f"({route_conditions})")

if selected_bus_types:
    type_conditions = " OR ".join([f"Bus_Type = '{bus_type}'" for bus_type in
selected_bus_types])
    filters.append(f"({type_conditions})")

filters.append(f"Price BETWEEN {price_range[0]} AND {price_range[1]}")
filters.append(f"Star_Rating BETWEEN {star_rating[0]} AND {star_rating[1]}")
filters.append(f"Seat_Availability BETWEEN {seats_available[0]} AND
{seats_available[1]}")

# Construct SQL query with filters
if filters:
    filter_query = " AND ".join(filters)
    filtered_query = f"{query} WHERE {filter_query}"
else:
    filtered_query = query

# Execute filtered SQL query and fetch data into a DataFrame
try:
    filtered_df = pd.read_sql(filtered_query, conn)
    st.subheader('Filtered Results')
    st.write(filtered_df)
except Exception as e:
    st.error(f"Error fetching filtered data: {str(e)}")
conn.close()
```

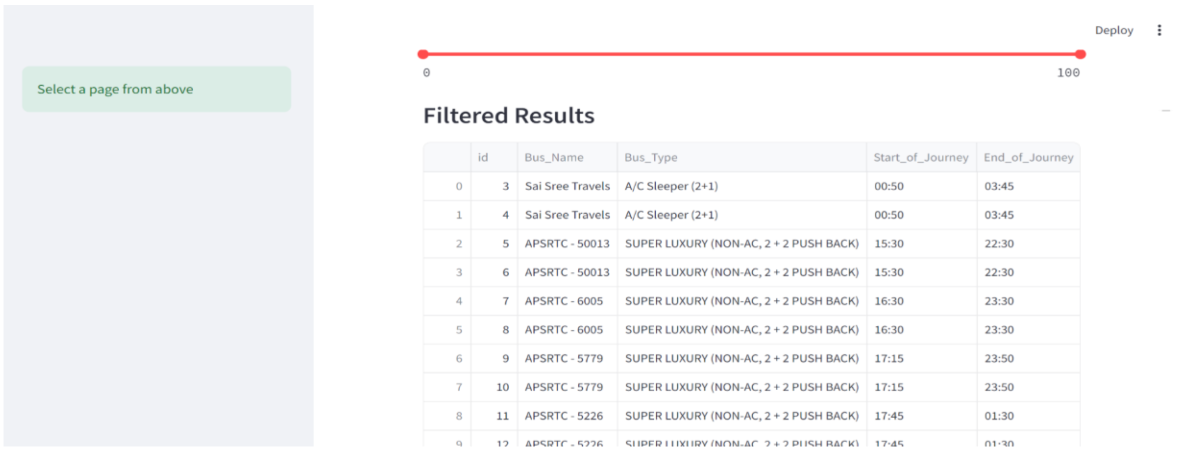
.

Running the Streamlit App:

streamlit run app.py

Screenshots :

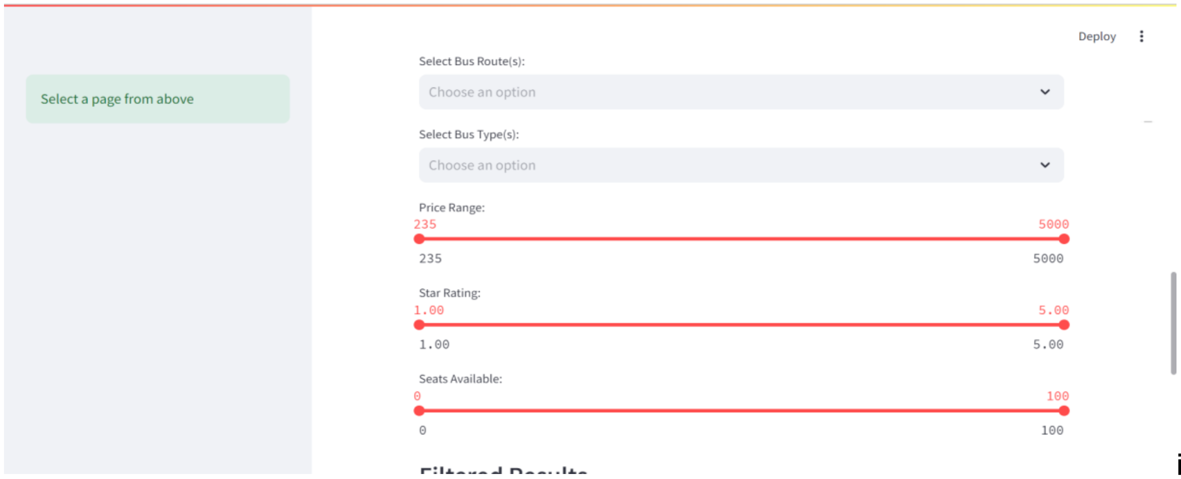




Filtered Results

	id	Bus_Name	Bus_Type	Start_of_Journey	End_of_Journey
0	3	Sai Sree Travels	A/C Sleeper (2+1)	00:50	03:45
1	4	Sai Sree Travels	A/C Sleeper (2+1)	00:50	03:45
2	5	APSRTC - 50013	SUPER LUXURY (NON-AC, 2 + 2 PUSH BACK)	15:30	22:30
3	6	APSRTC - 50013	SUPER LUXURY (NON-AC, 2 + 2 PUSH BACK)	15:30	22:30
4	7	APSRTC - 6005	SUPER LUXURY (NON-AC, 2 + 2 PUSH BACK)	16:30	23:30
5	8	APSRTC - 6005	SUPER LUXURY (NON-AC, 2 + 2 PUSH BACK)	16:30	23:30
6	9	APSRTC - 5779	SUPER LUXURY (NON-AC, 2 + 2 PUSH BACK)	17:15	23:50
7	10	APSRTC - 5779	SUPER LUXURY (NON-AC, 2 + 2 PUSH BACK)	17:15	23:50
8	11	APSRTC - 5226	SUPER LUXURY (NON-AC, 2 + 2 PUSH BACK)	17:45	01:30
9	12	APSRTC - 5226	SUPER LUXURY (NON-AC, 2 + 2 PUSH BACK)	17:45	01:30

L



Select Bus Route(s): Choose an option

Select Bus Type(s): Choose an option

Price Range: 235 – 5000 (235 5000)

Star Rating: 1.00 – 5.00 (1.00 5.00)

Seats Available: 0 – 100 (0 100)

ink :

Local URL: http://localhost:8501

Network URL: http://192.168.48.253:8501

6. Results

Outcomes:

- Successfully scraped 10 State Bus Transport data from Redbus website using Selenium. Also included the private bus information for the selected routes.
- Stored the data in a structured SQL database.
- Developed an interactive Streamlit application for data filtering.

7. Technical Tags

- Web Scraping
- Selenium
- Streamlit
- SQL
- Data Analysis
- Python

9. Conclusion

Summary:

Successfully scraped 10 State Bus Transport data from Redbus website using Selenium.

Also included the private bus information for the selected routes. Stored the data in a structured SQL database. Developed an interactive Streamlit application for data filtering.

10.Future Work:

Scraping more data to track the live data.

Including more inputs for filtrations.

11. References

- Links to resources and documentation used in the project.

  Selenium Documentation

  Streamlit Documentation

  PyMySQL Documentation