Redbus Data Scraping with Selenium & Dynamic Filtering using Streamlit

1. Introduction

Problem Statement:

The "Redbus Data Scraping and Filtering with Streamlit Application" aims to revolutionize the transportation industry by providing a comprehensive solution for collecting, analyzing, and visualizing bus travel data. By utilizing Selenium for web scraping, this project automates the extraction of detailed information from Redbus, including bus routes, schedules, prices, and seat availability. This project can significantly improve operational efficiency and strategic planning in the transportation industry by streamlining data collection and providing powerful tools for data-driven decision-making.

Data Set:

Source: Data will be scraped from the Redbus website.
Link- https://www.redbus.in/
Format: The scraped data will be stored in a SQL database.
Required Fields: Bus routes Link,Bus route Name, Bus name, Bus Type(Sleeper/Seater), Departing Time, Duration, Reaching_Time, Star-rating, Price, Seat_availability.

Data Set Requirements & Explanation:

The scraped dataset for this project should contain detailed information about bus services available on Redbus, covering various aspects critical to travelers and service providers. Here is a breakdown of the fields required:

- **Bus Routes Name:** This field captures the start and end locations of each bus journey, providing crucial information about the routes serviced.
- **Bus Routes Link**: Link for all the route details.
- **Bus Name:** The name of the bus or the service provider, which helps in identifying the specific operator.
- **Bus Type (Sleeper/Seater/AC/Non-AC):** This field specifies whether the bus is a sleeper or seater type, indicating the seating arrangements and comfort level offered.
- **Departing Time:** The scheduled departure time of the bus, essential for planning travel schedules.
- **Duration:** The total duration of the journey from the departure point to the destination, helping passengers estimate travel time.
- **Reaching Time:** The expected arrival time at the destination, allowing for better planning of onward travel or activities.
- **Star Rating:** A rating provided by passengers, indicating the quality of service based on factors such as comfort, punctuality, and staff behavior.

- **Price:** The cost of the ticket for the journey, which can vary based on factors like bus type and demand.
- **Seat Availability:** The number of seats available at the time of data scraping, giving real-time insight into the occupancy levels.

Project Overview:

In this project, data will be scraped from the Redbus website using Selenium, stored in an SQL database, and displayed through a Streamlit application. The goal is to provide an intuitive user interface for dynamic data filtering and exploration while automating the data extraction process.

Objectives:

- To scrape bus route details from Redbus.
- To store the scraped data in an SQL 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.

3. Project Setup

Prerequisites:

- Python installed on your machine.
- Required Python libraries: selenium, sqlalchemy, streamlit, pandas,matplotlib,plotly, pymysql.
- Chrome WebDriver for Selenium.

Installation:

pip install selenium

pip install sqlalchemy

pip install streamlit

pip install pandas

pip install pymysql

pip install matplotlib

pip install plotly

4. Web Scraping using Selenium

The code performs web scraping of bus transportation data from the Redbus website, using Selenium to interact with web pages, extract relevant information, and manage dynamic content. It employs the pymysql library to connect to a MySQL database and efficiently store the scraped data.

Web Scraping Process

a) **Initialization:**

Set the path for ChromeDriver and create a Service object. Configure browser options with Options, including disabling the sandbox with --no-sandbox if needed. Initialize the Chrome WebDriver with these settings and maximize the window for proper rendering..

```
chrome_driver_path = r'C:\Users\SAROBIN SILVIYA\RED BUS PROJECT\chromedriver-win64\chromedriver.exe'

service = Service(executable_path=chrome_driver_path)

options = Options()

options.add_argument('--no-sandbox')


driver = webdriver.Chrome(service=service, options=options)

driver.maximize_window()
```

b) **Navigating to Redbus Website:**

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

```
driver.get("https://www.redbus.in")
time.sleep(5)  # Wait for the page to load
```

c) **Clicks 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.

```
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)
```

d) Switching to New Tab:

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

```
driver.switch_to.window(driver.window_handles[1])
```

e) Scrolling to the Bottom of the Page:

The script scrolls down the page in a loop until it reaches the bottom. This ensures that all content is loaded.

```
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
```

f) Selecting a Bus Corporation:

The script selects a specific bus corporation and clicks on its link.

```
try:

    bus_corp = WebDriverWait(driver, 10).until(

        EC.element_to_be_clickable((By.XPATH,
'//*[@id="root"]/div/article[2]/div/div/ul[3]/li[6]/a'))

    )

    driver.execute_script("arguments[0].scrollIntoView(true);", bus_corp)

    time.sleep(2)

    bus_corp.click()

    time.sleep(5)

except ElementClickInterceptedException:

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

    bus_corp.click()
```

```
time.sleep(5)
```

g) Scraping Route Names and Links:

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

```
routes = []

while True:

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

    route_elements = driver.find_elements(By.CLASS_NAME, 'route')

    for route_element in route_elements:

        route = route_element.text

        route_link = route_element.get_attribute('href')

        routes.append((route, route_link))

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

    if more_pages.lower() != 'yes':

        break

    print("Scraped Routes and Links:")

    for route, link in routes:

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

h) 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.

```
bus_details = []
```

```
for route, route_link in routes:
    driver.get(route_link)
    time.sleep(5)  # Wait for the page to load

    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

    last_height = driver.execute_script("return document.body.scrollHeight")
    while True:
        driver.execute_script("window.scrollTo(0, document.body.scrollHeight);")
        time.sleep(2)
        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:
                try:
                    seats_available = bus.find_element(By.CSS_SELECTOR, "div.seat-left.m-top-16").text
```

```
            except NoSuchElementException:
                seats_available = bus.find_element(By.CSS_SELECTOR, "div.seat-left.m-top-30").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("Scraped Bus Details:")
    for detail in bus_details:
        print(detail)
```

SQL Data Storage Process

a) **Connecting to the MySQL Database:**

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

```
conn = pymysql.connect(
    host='127.0.0.1',
    user='root',
    passwd='silviya123',
    db='red_bus'  # Make sure the database exists
)
mycursor = conn.cursor()
```

b) **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.

```
mycursor.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 TIME,
        duration TEXT,
        reaching_time TIME,
        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 TIME format.
duration: The duration of the bus journey.
reaching_time: The arrival time of the bus, stored in TIME 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.

```
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.

```
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. Creation of Streamlit Application

The code that is supplied creates a Streamlit application that can retrieve bus transit data from a MySQL database, let users filter the data according to different standards, and then show the data that has been filtered. A download button to export the filtered data as a CSV file is also provided by the program. A thorough description of the code's operation may be found below.

Creating the Streamlit App:

a) **Importing Required Libraries**:

```
import streamlit as st

import pandas as pd

import mysql.connector

import matplotlib.pyplot as plt

import plotly.express as px
```

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

pandas: A library used for data manipulation and analysis.

matplotlib.pyplot:A library for creating static, animated, and interactive visualizations.

Plotly:A library for creating interactive plots.

b) Database Connection:

```
def fetch_data(query, params):

    conn = mysql.connector.connect(

        host="localhost",

        user="root",

        password="silviya123",

        database="red_bus"

    )

    try:

        data = pd.read_sql(query, conn, params=params)

    finally:
```

```
    conn.close()

  return data
```

mysql.connector.connect: Establishes a connection to the MySQL database.

pd.read_sql: Executes the SQL query and fetches data into a DataFrame.

conn.close(): Closes the database connection after fetching the data.

c) Fetching Distinct Routes for Dropdown:

```
def fetch_routes():
    conn = mysql.connector.connect(
        host="localhost",
        user="root",
        password="silviya123",
        database="red_bus"
    )
    query = "SELECT DISTINCT route_name FROM bus_routes"
    routes = pd.read_sql(query, conn)
    conn.close()
    return routes['route_name'].tolist()
```

SELECT DISTINCT route_name: SQL query to get unique route names.

routes['route_name'].tolist(): Converts the route names to a list for use in the dropdown.

d) Streamlit App Layout and Sidebar Filter:

```
def main():

    st.set_page_config(page_title="RedBus Data", page_icon=r"C:\Users\SAROBIN
SILVIYA\Downloads\red_bus_logo.webp", layout='wide')



    st.sidebar.title("Filter Options")



    min_price = st.sidebar.number_input('Min Price', min_value=0)

    max_price = st.sidebar.number_input('Max Price', min_value=0)

    min_departing_time = st.sidebar.time_input('Min Departing Time',
 value=pd.Timestamp('09:00').time())

    max_departing_time = st.sidebar.time_input('Max Departing Time',
value=pd.Timestamp('21:00').time())

    min_reaching_time = st.sidebar.time_input('Min Reaching Time',
value=pd.Timestamp('10:00').time())

    max_reaching_time = st.sidebar.time_input('Max Reaching Time',
value=pd.Timestamp('22:00').time())



    with st.sidebar.expander("Advanced Filters"):

        min_duration = st.text_input('Min Duration', '0 hours')

        max_duration = st.text_input('Max Duration', '24 hours')

        star_rating = st.slider('Star Rating', 0.0, 5.0, (0.0, 5.0))
```

bus_type = st.selectbox('Bus Type', ['All', 'Seater', 'Sleeper', 'AC', 'Non-AC'])

routes = fetch_routes()

route = st.sidebar.selectbox('Bus Route', [''] + routes) # Adding an empty option for no route filter

st.title("RedBus Data Filtering and Visualization")

st.markdown("Use the sidebar filters to explore bus routes, prices, and travel times.**")**

st.set_page_config: Configures the page title, icon, and layout.

st.sidebar.number_input: Creates number input fields for min and max price.

st.sidebar.time_input: Creates time input fields for departure and reaching times.

st.sidebar.expander: Groups advanced filters under an expandable section.

st.sidebar.selectbox: Creates a dropdown for bus routes.

e) SQL Query and Parameters:

query = """

SELECT *

FROM bus_routes

WHERE price BETWEEN %s AND %s

AND departing_time BETWEEN %s AND %s

```
  AND reaching_time BETWEEN %s AND %s

  AND duration BETWEEN %s AND %s

  AND star_rating BETWEEN %s AND %s

  AND (bustype = %s OR %s = 'All')

  AND route_name LIKE %s

"""



params = (

    min_price,

    max_price,

    min_departing_time.strftime('%H:%M:%S'),

    max_departing_time.strftime('%H:%M:%S'),

    min_reaching_time.strftime('%H:%M:%S'),

    max_reaching_time.strftime('%H:%M:%S'),

    min_duration,

    max_duration,

    star_rating[0],
```

```
        star_rating[1],

        bus_type,

        bus_type,

        f'%{route}%' if route else '%'

    )
```

SQL Query: Selects all fields from bus_routes table with filtering conditions.

params: Parameters for the SQL query based on user inputs.

f) Fetching and Displaying Data:

```
    if st.sidebar.button('Get Data'):
        with st.spinner('Fetching data...'):
            filtered_data = fetch_data(query, params)
            st.write(filtered_data)

            if not filtered_data.empty:
                st.subheader("Data Visualization")

                st.write("Price Distribution")
                price_counts = filtered_data['price'].value_counts().sort_index()
                fig, ax = plt.subplots()
                ax.bar(price_counts.index, price_counts.values, edgecolor='black')
                ax.set_xlabel('Price')
                ax.set_ylabel('Frequency')
                ax.set_title('Price Distribution')
                st.pyplot(fig)

                st.write("Star Rating Distribution")
                star_rating_counts = filtered_data['star_rating'].value_counts().sort_index()
                fig, ax = plt.subplots()
                ax.bar(star_rating_counts.index, star_rating_counts.values, edgecolor='black')
                ax.set_xlabel('Star Rating')
                ax.set_ylabel('Frequency')
```

```
            ax.set_title('Star Rating Distribution')
            st.pyplot(fig)

            st.write("Bus Type Distribution")
            bus_type_counts = filtered_data['bustype'].value_counts()
            fig = px.pie(values=bus_type_counts.values, names=bus_type_counts.index, title="Bus Type Distribution")
            st.plotly_chart(fig)

            csv = filtered_data.to_csv(index=False)
            st.download_button("Download CSV", data=csv, file_name='filtered_data.csv', mime='text/csv')
```

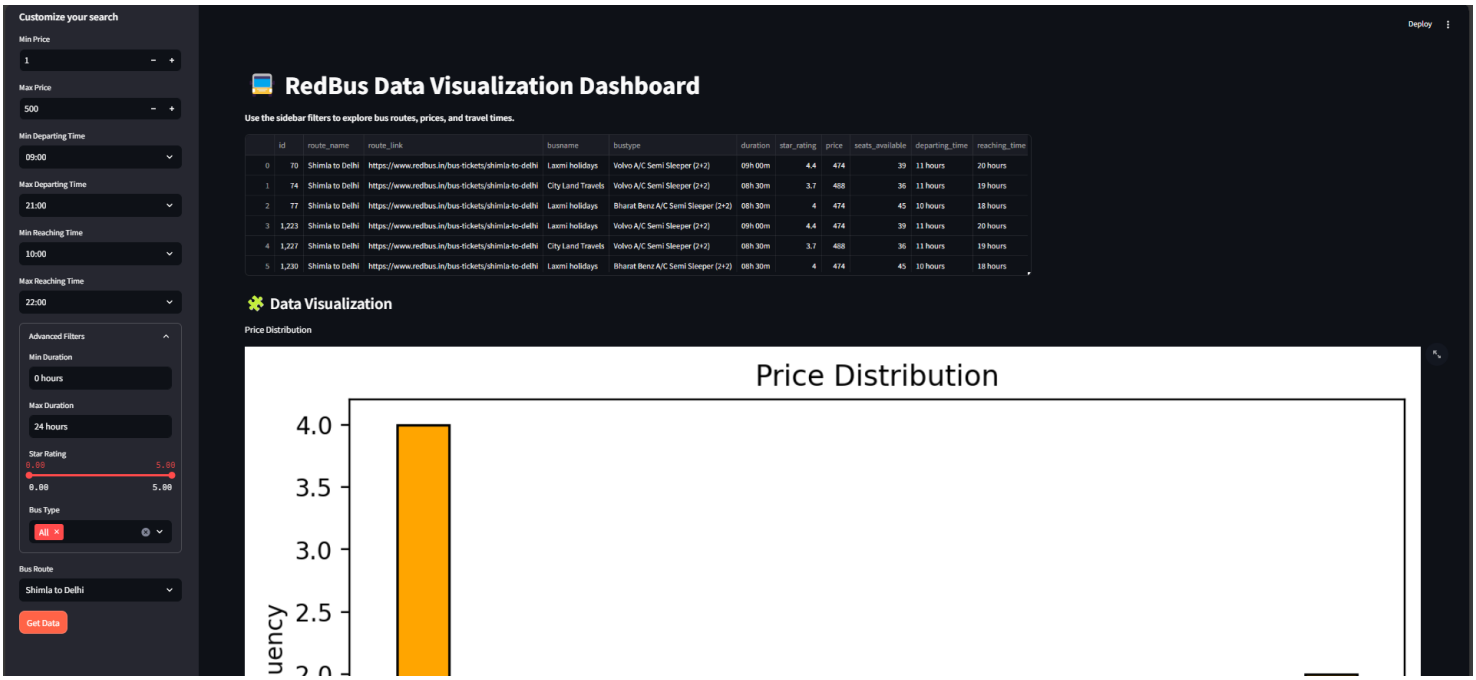
st.spinner: Shows a loading spinner while data is being fetched.
st.write: Displays the filtered data.
Visualization:

- Price Distribution: Bar chart of price distribution.
- Star Rating Distribution: Bar chart of star rating distribution.
- Bus Type Distribution: Pie chart of bus type distribution.

g) Running the Streamlit App:

```
streamlit run  "C:\Users\SAROBIN SILVIYA\RED BUS PROJECT\app.py"
```

Screenshots :



Link :

Local URL: http://localhost:8501

Network URL: http://172.20.10.2:8501

6. Results

Expected Outcomes:

- Use Selenium to successfully scrape at least 10 Government State Bus Transport data points from the Redbus website. Include information on private buses on the chosen routes as well.
- Put the information in an organized SQL database.
- Create an interactive data filtering application using Streamlit.
- Make sure the application is effective and easy to use.

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.

  https://www.redbus.in/

  Selenium Documentation

  Streamlit Documentation

  PyMySQL Documentation