## **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 SQL 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 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.
* **Visual studio :** For development and testing.(streamlit)

### **3. Project Setup**

**Prerequisites:**

* Python 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 Government 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.

# def initialize\_driver():

driver = webdriver.Chrome()

driver.maximize\_window()

return driver

1. Navigating to the Redbus Government bus transport Website (Desired URL):

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

# Open def load\_page(driver, url):

driver.get(url)

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.

# def scrape\_bus\_routes(driver):

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

bus\_routes\_link = [route.get\_attribute('href') for route in route\_elements]

bus\_routes\_name = [route.text.strip() for route in route\_elements]

return bus\_routes\_link, bus\_routes\_name

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
* def scrape\_bus\_details(driver, url, route\_name):
* try:
* driver.get(url)
* time.sleep(5)
* try:
* view\_buses\_button = WebDriverWait(driver, 10).until(
* EC.element\_to\_be\_clickable((By.CLASS\_NAME, "button"))
* )
* driver.execute\_script("arguments[0].click();", view\_buses\_button)
* time.sleep(5)
* except:
* print(f"No 'View Buses' button found for {url}")
* driver.execute\_script("window.scrollTo(0, document.body.scrollHeight);")
* time.sleep(5)
* bus\_name\_elements = driver.find\_elements(By.CSS\_SELECTOR, ".travels.lh-24.f-bold.d-color")
* bus\_type\_elements = driver.find\_elements(By.CSS\_SELECTOR, ".bus-type.f-12.m-top-16.l-color")
* departing\_time\_elements = driver.find\_elements(By.CSS\_SELECTOR, ".dp-time.f-19.d-color.f-bold")
* duration\_elements = driver.find\_elements(By.CSS\_SELECTOR, ".dur.l-color.lh-24")
* reaching\_time\_elements = driver.find\_elements(By.CSS\_SELECTOR, ".bp-time.f-19.d-color.disp-Inline")
* star\_rating\_elements = driver.find\_elements(By.XPATH, "//div[@class='rating-sec lh-24']")
* price\_elements = driver.find\_elements(By.CSS\_SELECTOR, ".fare.d-block")
* seat\_availability\_elements = driver.find\_elements(By.XPATH, "//div[contains(@class, 'seat-left m-top-30') or contains(@class, 'seat-left')]")
* bus\_details = []
* for i in range(len(bus\_name\_elements)):
* try:
* seat\_availability = seat\_availability\_elements[i].text if i < len(seat\_availability\_elements) else 'N/A'
* seat\_availability = int(seat\_availability) if seat\_availability.isdigit() else 0
* price\_text = price\_elements[i].text
* price\_numeric = ''.join(filter(str.isdigit, price\_text))
* bus\_detail = {
* "Route\_Name": route\_name,
* "Route\_Link": url,
* "Bus\_Name": bus\_name\_elements[i].text,
* "Bus\_Type": bus\_type\_elements[i].text,
* "Departing\_Time": departing\_time\_elements[i].text,
* "Duration": duration\_elements[i].text,
* "Reaching\_Time": reaching\_time\_elements[i].text,
* "Star\_Rating": star\_rating\_elements[i].text if i < len(star\_rating\_elements) else '0',
* "Price": price\_numeric,
* "Seat\_Availability": seat\_availability\_elements[i].text if i < len(seat\_availability\_elements) else 'N/A'
* }
* bus\_details.append(bus\_detail)
* except Exception as e:
* print(f"Error occurred while scraping bus details: {str(e)}")
* return bus\_details
* except Exception as e:
* print(f"Error occurred while accessing {url}: {str(e)}")
* return []

1. Scraping Bus Details for remaining pages:

The script iterates through each page and 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 for remaining pages

def scrape\_all\_pages():

all\_bus\_details = []

for page in range(1, 6):

try:

driver = initialize\_driver()

load\_page(driver, URL)

if page > 1:

pagination\_tab = WebDriverWait(driver, 10).until(

EC.presence\_of\_element\_located((By.XPATH, f"//div[contains(@class, 'DC\_117\_pageTabs')][text()='{page}']"))

)

driver.execute\_script("arguments[0].scrollIntoView();", pagination\_tab)

driver.execute\_script("arguments[0].click();", pagination\_tab)

time.sleep(5)

all\_bus\_routes\_link, all\_bus\_routes\_name = scrape\_bus\_routes(driver)

for link, name in zip(all\_bus\_routes\_link, all\_bus\_routes\_name):

bus\_details = scrape\_bus\_details(driver, link, name)

if bus\_details:

all\_bus\_details.extend(bus\_details)

driver.quit()

except Exception as e:

print(f"Error occurred while accessing page {page}: {str(e)}")

return all\_bus\_details

***Same scrapping process followed for remaining Government State Bus Transport data from Redbus website using Selenium and stored data in MySQL database.***

### **5. 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
* import pandas as pd
* import pymysql
* # Function to insert data from DataFrame to MySQL
* def insert\_data\_from\_df\_to\_mysql(df):
* try:
* # Establish connection to the MySQL database
* connection = pymysql.connect(
* host='127.0.0.1',
* user='root',
* password='Root',
* database='red\_bus',
* cursorclass=pymysql.cursors.DictCursor
* )
* with connection:
* with connection.cursor() as cursor:

1. Creating the Database Schema:

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

* # Create table if it does not exist
* cursor.execute("""
* CREATE TABLE IF NOT EXISTS bus\_details (ID int auto\_increment primary key,
* Route\_Name TEXT,
* Route\_Link TEXT,
* Bus\_Name TEXT,
* Bus\_Type TEXT,
* Departing\_Time TIME,
* Duration TEXT,
* Reaching\_Time TIME,
* Star\_Rating FLOAT,
* Price DECIMAL(10,2),
* Seat\_Availability VARCHAR(255)
* )
* """)
* 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.
  + Bus\_Name: The name of the bus operator.
  + Bus\_Type: 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\_Availability: The number of seats available on the bus, stored as an varchar.
* d) Inserting Data into the Database:

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

* # Insert data into the table
* for \_, detail in df.iterrows():
* cursor.execute("""
* INSERT INTO bus\_details (
* Route\_Name, Route\_Link, Bus\_Name, Bus\_Type, Departing\_Time,
* Duration, Reaching\_Time, Star\_Rating, Price, Seat\_Availability
* ) VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s, %s)
* """, (
* detail['Route\_Name'], detail['Route\_Link'], detail['Bus\_Name'],
* detail['Bus\_Type'], detail['Departing\_Time'], detail['Duration'],
* detail['Reaching\_Time'], detail['Star\_Rating'], detail['Price'],
* detail['Seat\_Availability']
* ))
* 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\_details 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
* connection.commit()
* print("Data inserted successfully into MySQL database.")
* except pymysql.MySQLError as err:
* print(f"Error: {err}")
* print("Data inserted successfully into MySQL database.")
* The database connection is then closed to free up resources and prints a confirmation message indicating that the data has been successfully saved to the database.
* conn.close()
* f) Insert the List of CSV file into database and Printing Confirmation Message:
* # List of CSV files to be inserted into the database
* csv\_files = [
* 'APSRTC\_bus\_details.csv',
* 'WBTC\_bus\_details.csv',
* 'UPSRTC\_bus\_details.csv',
* 'TSRTC\_bus\_details.csv',
* 'SBSTC\_bus\_details.csv',
* 'RSRTC\_bus\_details.csv',
* 'PEPSU\_bus\_details.csv',
* 'JKSRTC\_bus\_details.csv',
* 'NBSTC\_bus\_details.csv',
* 'KSRTC\_bus\_details.csv',
* 'Kadamba\_bus\_details.csv',
* 'HSRTC\_bus\_details.csv',
* 'Chandigarh\_bus\_details.csv',
* 'Assam\_bus\_details.csv'
* ]
* # Loop through each CSV file and insert data into MySQL
* for csv\_file in csv\_files:
* df = pd.read\_csv(csv\_file)
* insert\_data\_from\_df\_to\_mysql(df)
* The script saves the scrapped data into data frame from CSV format.

### **6. 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 pymysql

import pandas as pd

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.

1. Database Connection Using pymysql:

def get\_connection():

return pymysql.connect(host='127.0.0.1', user='root', passwd='Root', database='red\_bus')

This code sets up a connection to the MySQL database using pymysql. Replace the credentials with your actual database username, password, host, port, and database name.

1. Fetching Data from the Database:

# Function to fetch route names starting with a specific letter, arranged alphabetically

def fetch\_Route\_Name(connection, starting\_letter):

query = f"SELECT DISTINCT Route\_Name FROM bus\_details WHERE Route\_Name LIKE '{starting\_letter}%' ORDER BY Route\_Name"

Route\_Name = pd.read\_sql(query, connection)['Route\_Name'].tolist()

return Route\_Name

# Function to fetch data from MySQL based on selected Route\_Name and price sort order

def fetch\_data(connection, Route\_Name, Price\_sort\_order):

Price\_sort\_order\_sql = "ASC" if Price\_sort\_order == "Low to High" else "DESC"

query = f"SELECT \* FROM bus\_details WHERE Route\_Name = %s ORDER BY Star\_Rating DESC, Price {Price\_sort\_order\_sql}"

df = pd.read\_sql(query, connection, params=(Route\_Name))

return df

# Function to filter data based on Star\_Rating and Bus\_Type

def filter\_data(df, star\_rating, Bus\_type):

filtered\_df = df[df['Star\_Rating'].isin(star\_rating) & df['Bus\_Type'].isin(Bus\_type)]

return filtered\_df

The pd.dataframe sql function is used to execute the SQL query and fetch the data from the bus\_details table into a pandas DataFrame named data.

1. Streamlit App Layout:

def main():

st.title("RedBus")

logo = "D:\RedBus\logo1.png"

caption\_image = "D:\RedBus\Caption Image.png"

st.header("India's No. 1 Online Bus Ticket Booking Site", divider="gray")

st.image(caption\_image)

st.sidebar.image(logo, caption='Travel company', use\_column\_width=True)

connection = get\_connection()

This def function sets the title, logo, caption, header and sidebar image of the Streamlit application.

1. Filters:

# Sidebar - Input for starting letter

starting\_letter = st.sidebar.text\_input('Enter starting letter of Route Name', 'A')

# Fetch route names starting with the specified letter

if starting\_letter:

Route\_name = fetch\_Route\_Name(connection, starting\_letter.upper())

if Route\_name:

# Sidebar - Selectbox for Route\_Name

selected\_route = st.sidebar.radio('Select Route Name', Route\_name)

if selected\_route:

# Sidebar - Selectbox for sorting preference

Price\_sort\_order = st.sidebar.selectbox('Sort by Price', ['Low to High', 'High to Low'])

# Fetch data based on selected Route\_Name and price sort order

data = fetch\_data(connection, selected\_route, Price\_sort\_order)

if not data.empty:

# Display data table with a subheader

st.write(f"### Data for Route: {selected\_route}")

st.write(data)

# Filter by Star\_Rating and Bus\_Type

Bus\_Type = data['Bus\_Type'].unique().tolist()

selected\_Bus\_Type = st.multiselect('Filter by Bus Type', Bus\_Type)

Star\_rating = data['Star\_Rating'].unique().tolist()

selected\_ratings = st.multiselect('Filter by Star Rating', Star\_rating)

if selected\_ratings and selected\_Bus\_Type:

filtered\_data = filter\_data(data, selected\_ratings, selected\_Bus\_Type)

Multiselect Filters:

* Enter starting letter of Route Name: Allows users to type the exact route name or destination name.
* Select Route Name: Based on above filter search result Allows users to select respective routes.
* Sort by Price: Allows users to sort the price from Low to High or High to Low.
* Filter by Star Rating: Allows users to select the star rating which they want.
* Filter by Bus Type: Allows users to select the bus type which they want.

1. Displaying Data Based on User Inputs:

* # Display filtered data table with a subheader
* st.write(f"### Filtered Data for Star Rating: {selected\_ratings} and Bus Type: {selected\_Bus\_Type}")
* st.write(filtered\_data)
* else:
* st.write(f"No data found for Route: {selected\_route} with the specified price sort order.")
* else:
* st.write("No routes found starting with the specified letter.")
* finally:
* connection.close()
* if \_\_name\_\_ == '\_\_main\_\_':
* main()

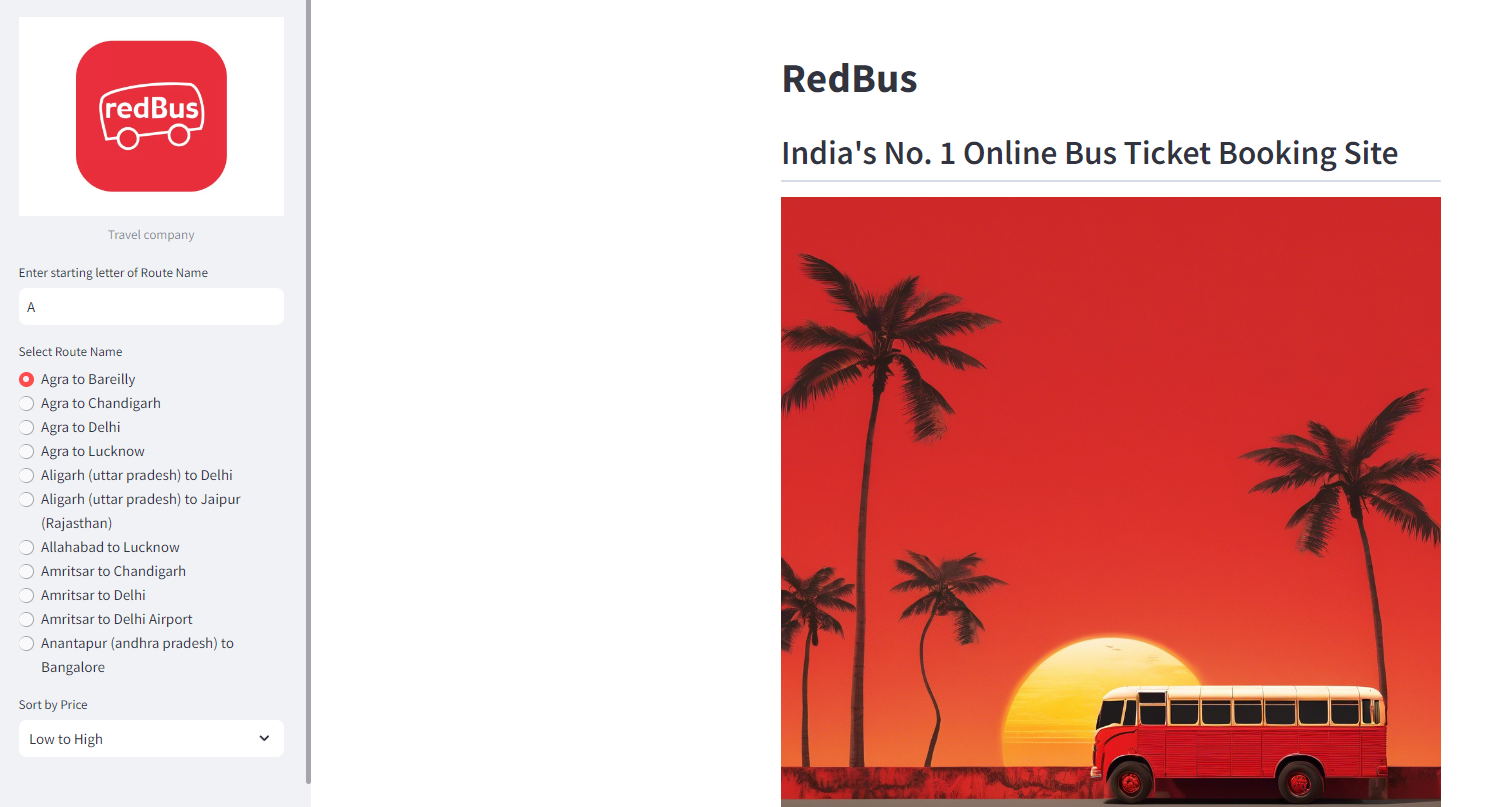
This code displays the filtered data with two sub headers

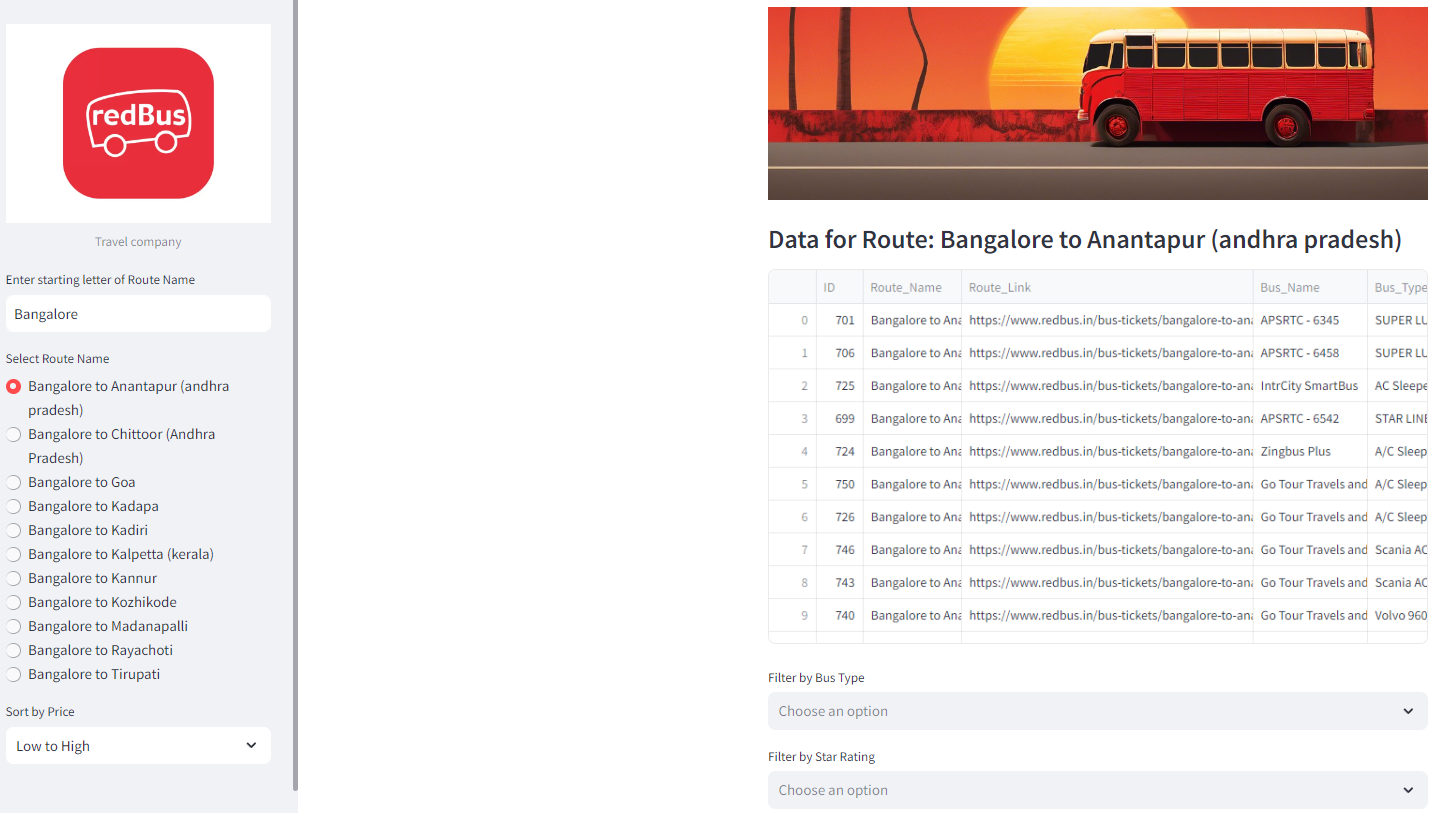
* Filter by Star Rating: Allows users to select the star rating which they want.
* Filter by Bus Type: Allows users to select the bus type which they want.

**Running the Streamlit App:**

streamlit run your\_script\_name.py

**Screenshots :**

****

****

**Link :**

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

**Network URL:** [**http://192.168.0.106:8501**](http://192.168.0.106:8501)

### **7. Results**

**Expected Outcomes:**

* Successfully scrape a minimum of 14 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.

### **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/)