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

**1. Project Overview**

The **Redbus Data Scraping with Selenium & Dynamic Filtering using Streamlit** project offers an innovative approach to automating data extraction, analysis, and visualization from the Redbus platform. By integrating **Selenium**, **mysql**, and **Streamlit**, this project enables users to explore bus travel data, including schedules, routes, prices, and seat availability, in an interactive and user-friendly environment.

**2. Objectives**

* **Automate Data Collection**: Extract bus details (routes, schedules, prices, and availability) from the Redbus platform using Selenium.
* **Streamline Data Storage**: Store the extracted data in structured formats for efficient retrieval and processing.
* **Enable Interactive Analysis**: Provide a web application to dynamically filter, analyze, and visualize data.

**3. 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.

**4. System Architecture**

1. **Web Scraping:**
   * Automate the extraction of dynamic data from Redbus using Selenium.
   * Collect essential fields such as bus operator, departure/arrival times, travel duration, and prices.
2. **Data Storage**:
   * Save raw data as CSV files.
   * Use mysql to manage data for structured querying and analysis.
3. **Data Processing**:
   * Utilize pandas for data cleaning and preprocessing.
   * Apply filters based on user-defined criteria, such as price range or travel time.
4. **Interactive Visualization**:
   * Develop an intuitive Streamlit application for data interaction.

**4. Tools and Technologies**

* **Selenium**: Web scraping automation.
* **Streamlit**: Building interactive dashboards.
* **mysql**: Database management for structured data storage.
* **pandas**: Data manipulation and analysis.
* **Python**: Core programming language.

**5. Project Setup**

**Prerequisites:**

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

**6. Web Scraper (redbus\_selenium.py)**

* Automates navigation and data extraction from Redbus.
* Example fields scraped:
  + Bus operator name
  + Departure and arrival times
  + Travel duration
  + Price per ticket

**2. Database Management (redbus\_sql.py)**

* Schema design and database connection setup.
* Facilitates CRUD operations for the application.

**3. Data Processing and Dashboard (RBstream.py)**

* Cleans and preprocesses raw data.
* Implements filtering by parameters like price range, travel duration, and departure times.
* Provides an interactive Streamlit interface for uploading, analyzing, and visualizing data.

**7. Code Snippets**

**Scraper Module**

Initialization:

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

def initialize\_driver():

    driver = webdriver.Chrome(service=Service(ChromeDriverManager().install()))

    driver.maximize\_window()

    return driver

Navigating to Redbus Website:

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

click 'view buses' button if available

            clicks = driver.find\_element(By.XPATH, "//div[@class='button']")

            clicks.click()

Scroll until page content stops updating

        driver.execute\_script("window.scrollTo(0, document.body.scrollHeight);")

Scroll the element into view

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

time.sleep(2)

Scraping Bus Details:snippet

 # Extract bus details

bus\_elements = driver.find\_elements(By.CSS\_SELECTOR, "div.bus-item")

bus\_name = driver.find\_elements(By.XPATH, "//div[@class='travels lh-24 f-bold d-color']")

# Collect bus details

for i in range(len(bus\_name)):

            bus\_detail = {

                "Route\_Name": routes,

                "Route\_Link": link,

                "Bus\_Name": bus\_name[i].text if i < len(bus\_name) else 'N/A',

Save route data to CSV

    df\_routes = pd.DataFrame({"Route\_name": all\_routes\_name, "Route\_link": all\_routes\_link})

    df\_routes.to\_csv("ksrtc\_bus\_routes.csv", index=False)

**8. SQL Database Integration**

**Approach:**Use Python's sqlite3 or another SQL database connector (like mysql-connector-python for MySQL) to store the scraped data.

**Connecting to the MySQL Database:**

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

# Step 2: Connect to MySQL

    try:

        connection = mysql.connector.connect(

            host=DB\_HOST,

            user=DB\_USER,

            password=DB\_PASSWORD,

            database=DB\_NAME

        )

        cursor = connection.cursor()

**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
* )
* ''')

**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)

**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()

**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.")

**9. Streamlit App Development**

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

Importing Required Libraries:

import streamlit as st

from sqlalchemy import create\_engine

import pandas as pd

Database Connection Using SQLAlchemy:

engine = create\_engine('mysql+pymysql://root:sripathi12345678@127.0.0.1:3306/redbus\_data')

Fetching Data from the Database:

query = "SELECT \* FROM bus\_routes"

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

Streamlit App Layout:

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

**Filters: code snippets**

* 1. route\_filter = st.multiselect('Select Route:', options=data['route\_name'].unique())
  2. 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())))

Multiselect Filters: Allows users to select multiple bus types from a dropdown.

**Filtering Data Based on User Inputs**:

filtered\_data = data

1. if route\_filter:

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

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

Data Frame data is filtered by this code according to the user's choices . Rows are filtered using the is in technique according to the bus kinds and routes that have been chosen. Rows are filtered according to the chosen price range, star rating range, and seat availability range using the between condition.

**Displaying Filtered Data:**

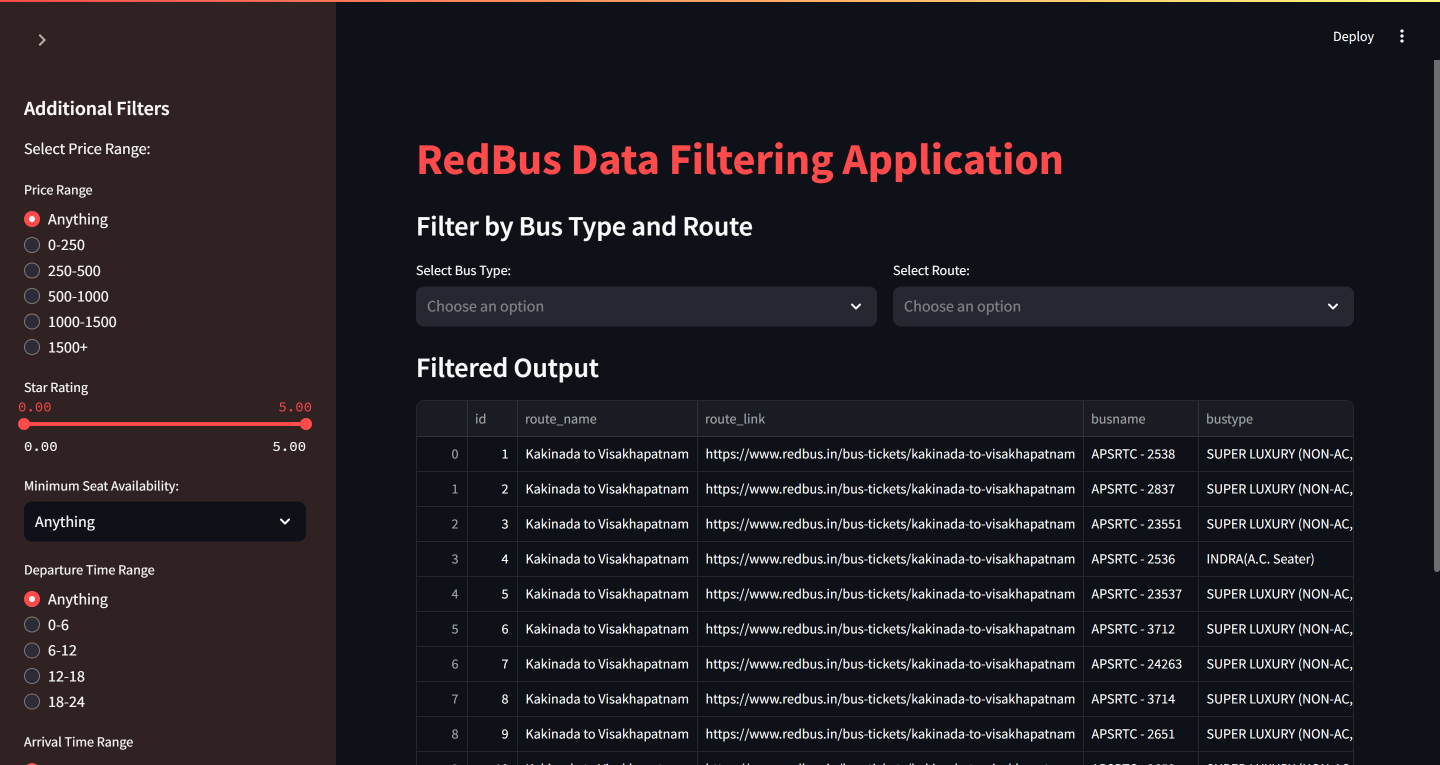
st.write('Filtered Data:')

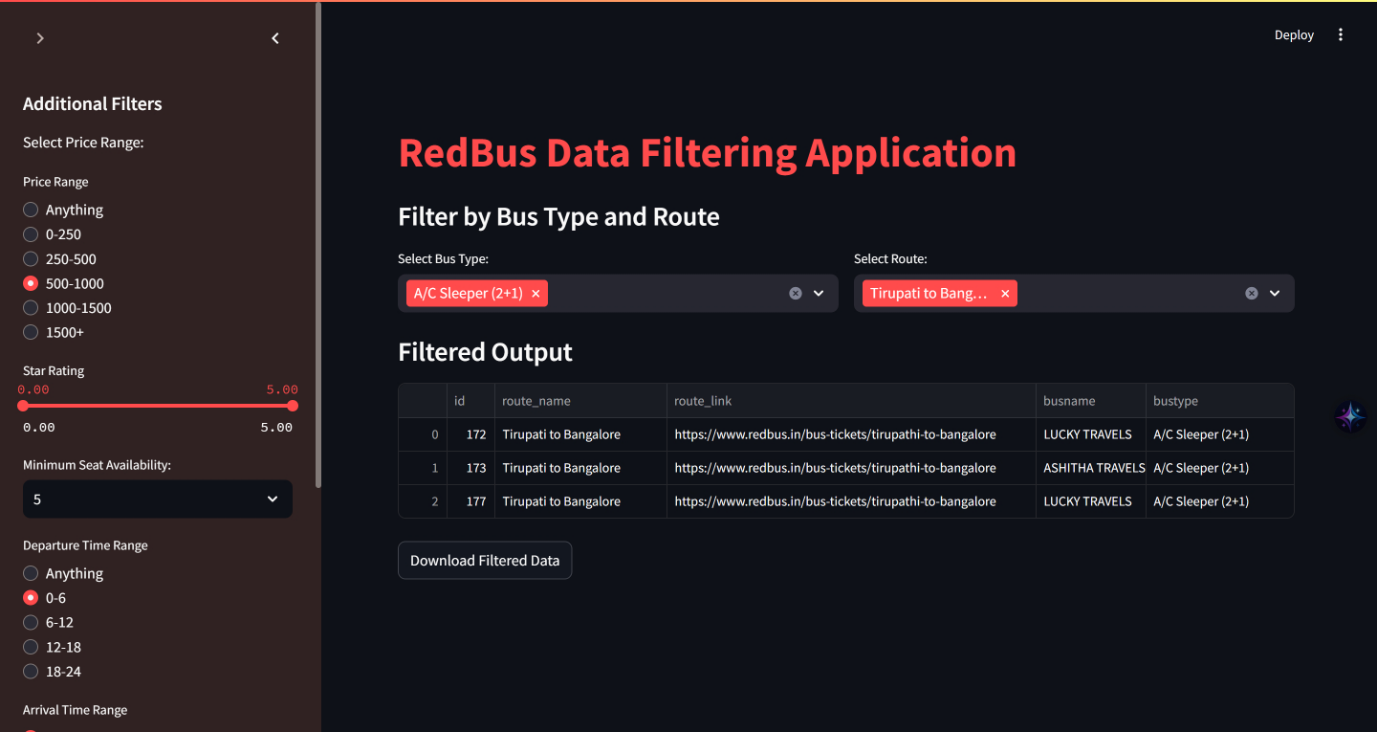
st.dataframe(filtered\_data)

**Running the Streamlit App:**

* streamlit run your\_script\_name.py

**10. 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.

**11. Conclusion**

The **Redbus Data Scraping with Selenium & Dynamic Filtering using Streamlit** project showcases a comprehensive solution for bus travel data management. By automating data extraction, streamlining analysis, and providing interactive visualization, this project empowers users to derive actionable insights efficiently. The modular structure ensures future scalability and adaptability to other transportation data sources or analytics use cases.