GET YOUR VISION IN INDIAN

Poonamallee High Road Chennai, -600100

A

Project Report on

“ Redbus Data Scraping with Selenium &

Dynamic Filtering using Streamlit ”

Submitted in partial fulfilment of the requirement

For the award of the course

DATA SCIENCE

SUBMITTED BY

SHALINI .R

[MDTM 30]

DECLARATION

I, Shalini R, hereby declare that the project report titled **"Redbus Data Scraping with Selenium & Dynamic Filtering using Streamlit"** is my original work. This project has been carried out as a part of the course requirements for the **Data Science program at GUVI.**

Thanking you,

SHALINI .R
[MDTM 30]

ACKNOWLEDGEMENT

I would like to express my sincere gratitude to all those who have contributed to the successful completion of this project.

First and foremost, I would like to thank my mentor and project supervisor for their invaluable guidance and support throughout the project. Their insights and expertise have significantly enhanced my understanding of data scraping and data analysis.

I am also grateful to the faculty members of the **Data Science** department at GUVI for providing a strong foundation in the skills and techniques necessary for this project. Their dedication to teaching and commitment to student success have been truly inspiring.

I would like to extend my appreciation to my peers and colleagues for their encouragement and collaboration during the course of this project. Their camaraderie has made this learning experience enjoyable and memorable.

Finally, I would like to thank my family and friends for their unwavering support and motivation. Their belief in me has been a constant source of strength.

Thank you all for your contributions to my academic journey and for making this project possible.

SHALINI .R
[MDTM 30]

ABSTRACT

This project report presents **"Redbus Data Scraping with Selenium & Dynamic Filtering using Streamlit,"** aimed at revolutionizing the transportation industry by automating the extraction and analysis of bus travel data from the Redbus platform. With the increasing demand for efficient travel solutions, this project leverages Selenium for web scraping, enabling the collection of comprehensive data related to bus routes, schedules, prices, and seat availability.

The project involves a systematic approach to data scraping, where detailed information is extracted and stored in a structured SQL database for effective querying. A user-friendly Streamlit application is then developed to provide interactive data filtering capabilities, allowing users to sort and analyze bus travel data based on various parameters such as bus type, route, price range, star rating, and seat availability.

The results of this project aim to provide valuable insights for various stakeholders, including travel aggregators, market analysts, and customers, thereby enhancing decision-making processes and improving operational efficiency in the transportation sector. The application also emphasizes user experience, ensuring that it is intuitive and responsive to user inputs.

In conclusion, this project not only demonstrates the technical feasibility of web scraping and data analysis using Python but also contributes to the broader goal of leveraging technology to optimize travel experiences.

INDEX

1. INTRODUCTION

1.1 Background

The transportation industry has seen a significant transformation in recent years, with digital platforms playing a crucial role in enhancing accessibility and convenience for travelers. Redbus is one of the largest online bus ticketing platforms in India, offering a wide range of bus services from various operators. As the demand for efficient and reliable travel information continues to grow, there is an increasing need for automated tools that can extract and analyze data from such platforms. Web scraping, a technique used to extract data from websites, offers an effective solution for gathering travel-related information. This project focuses on leveraging Selenium, a popular web scraping tool, to automate the extraction of data from Redbus and present it through a dynamic filtering interface developed with Streamlit.

1.2 Problem Statement

Despite the abundance of information available on the Redbus platform, users often face challenges in accessing and analyzing bus travel data efficiently. Manually searching for routes, schedules, prices, and seat availability can be time-consuming and cumbersome. Additionally, transportation providers and travel aggregators lack comprehensive tools to analyze market trends and customer preferences effectively. This project aims to address these challenges by creating a web scraping solution that automates data extraction and develops an interactive application for filtering and visualizing the data, thereby enhancing user experience and decision-making in the transportation sector.

1.3 Objectives

The primary objectives of this project are:

- To develop a robust web scraping tool using Selenium to extract detailed bus travel data from the Redbus website, including routes, schedules, prices, and seat availability.
- To store the scraped data in a structured SQL database for easy retrieval and analysis.
- To create an interactive web application using Streamlit that allows users to filter and analyze the scraped data based on various parameters such as bus type, route, price range, star rating, and seat availability.
- To provide valuable insights into travel patterns and preferences for stakeholders in the transportation industry, including travel aggregators and service providers.

1.4 Scope of the Project

This project focuses on the extraction and analysis of bus travel data specifically from the Redbus platform. The scope includes:

- Utilizing Selenium for web scraping to automate the data collection process.
- Storing the extracted data in a SQL database to facilitate efficient data management and querying.
- Developing a user-friendly Streamlit application that allows users to filter and visualize the bus travel data dynamically.

- Analyzing the collected data to identify trends, customer preferences, and operational insights relevant to the transportation industry.
- The project will not encompass data extraction from other travel platforms or advanced predictive analytics, as the primary focus is on the Redbus platform and enhancing user experience through dynamic filtering.

2. SYATEM ANALYSIS

2.1 System Requirements

2.1.1 Hardware Requirements

- **Processor:** Minimum of Intel i3 or equivalent for efficient data processing.
- **RAM:** At least 8 GB to handle multiple processes and large datasets during scraping and analysis.
- **Storage:** Sufficient disk space (at least 100 GB) to store scraped data, databases, and application files.

2.1.2 Software Requirements

- **Operating System:** Windows, macOS, or Linux (Ubuntu preferred for compatibility with most scraping tools).
- **Python:** Version 3.6 or higher for running the web scraping and Streamlit applications.
- **Libraries:**
    - Selenium: For automating web browser interactions.
    - Beautiful Soup: For parsing HTML content (if needed).
    - Pandas: For data manipulation and analysis.
    - Streamlit: For creating the interactive web application.
    - SQLAlchemy: For database interaction.

2.1.3 Network Requirements

- Stable internet connection for accessing the Redbus website and scraping data.

3. SYSTEM DESIGN

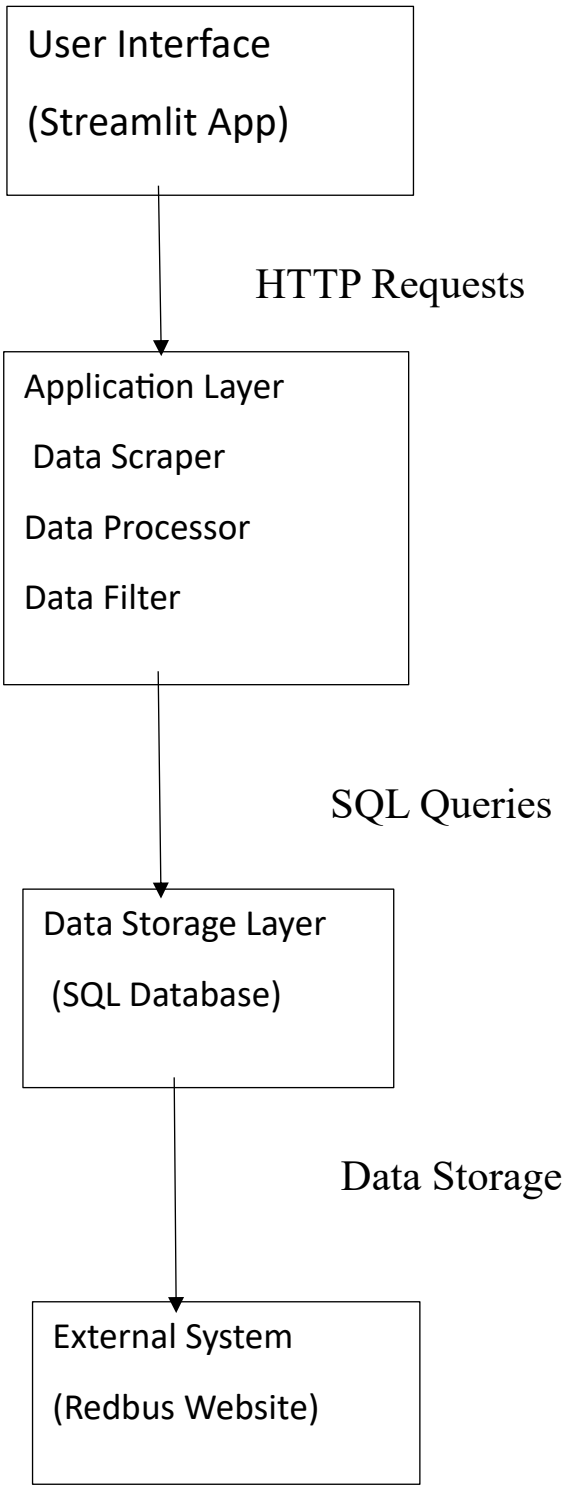
3.1 System Components

1. **User Interface Layer**
    - **Streamlit Application:** A web-based interface that allows users to interact with the data. Users can view bus details, apply filters, and get results based on various parameters like bus type, price range, and seat availability.
2. **Application Layer**
    - **Data Scraper Module:** Uses Selenium to automate the process of navigating the Redbus website and extracting data on bus routes, schedules, prices, and seat availability.
    - **Data Processing Module:** Cleans, formats, and prepares the scraped data for storage. This module removes duplicates, handles missing values, and ensures data consistency.
    - **Data Filtering Module:** Implements logic to filter the data based on user inputs. This module interacts with the SQL database to fetch and filter data dynamically.
3. **Data Storage Layer**
    - **SQL Database:** Stores all scraped data in a structured format. The database schema includes tables for bus routes, bus details, operators, and user preferences. This layer is responsible for CRUD (Create, Read, Update, Delete) operations on the data.
4. **External System**
    - **Redbus Website:** The primary data source from which the application extracts bus information.

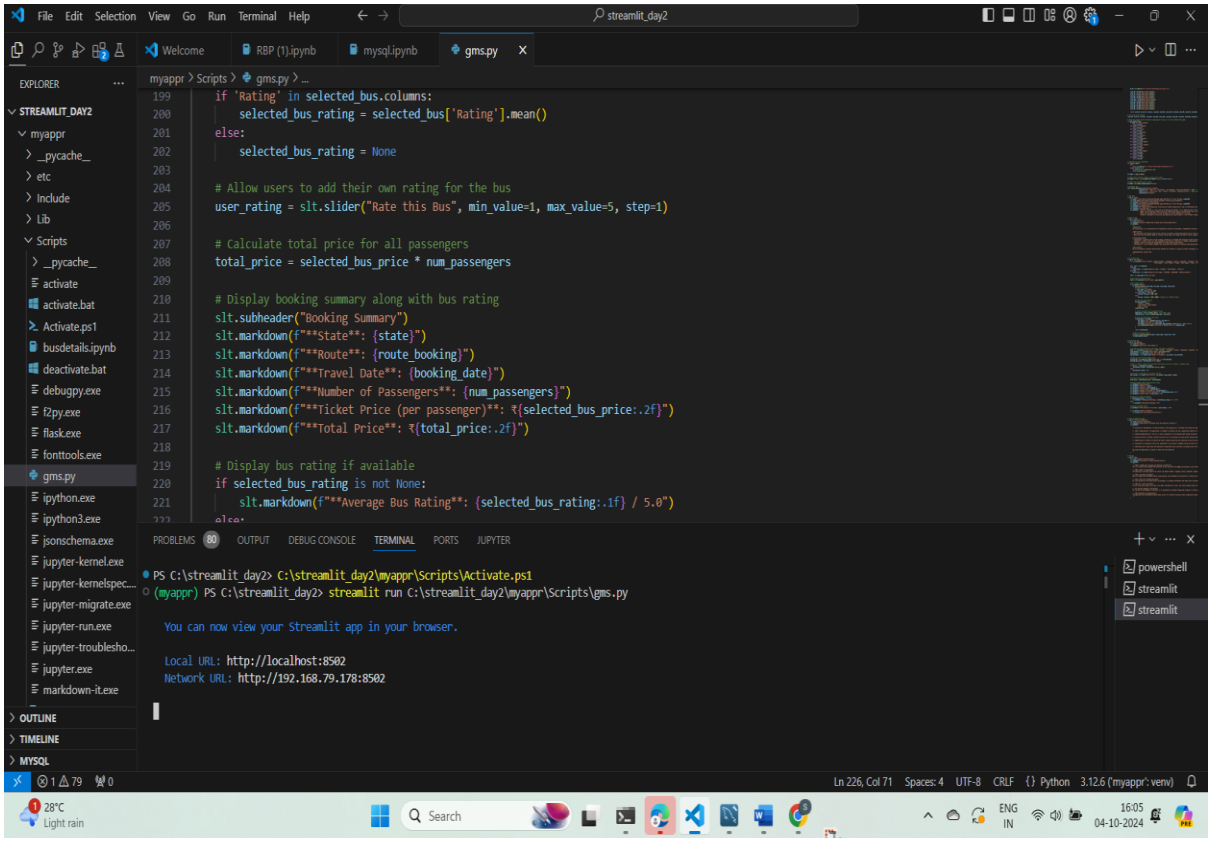
3.2 System Architecture

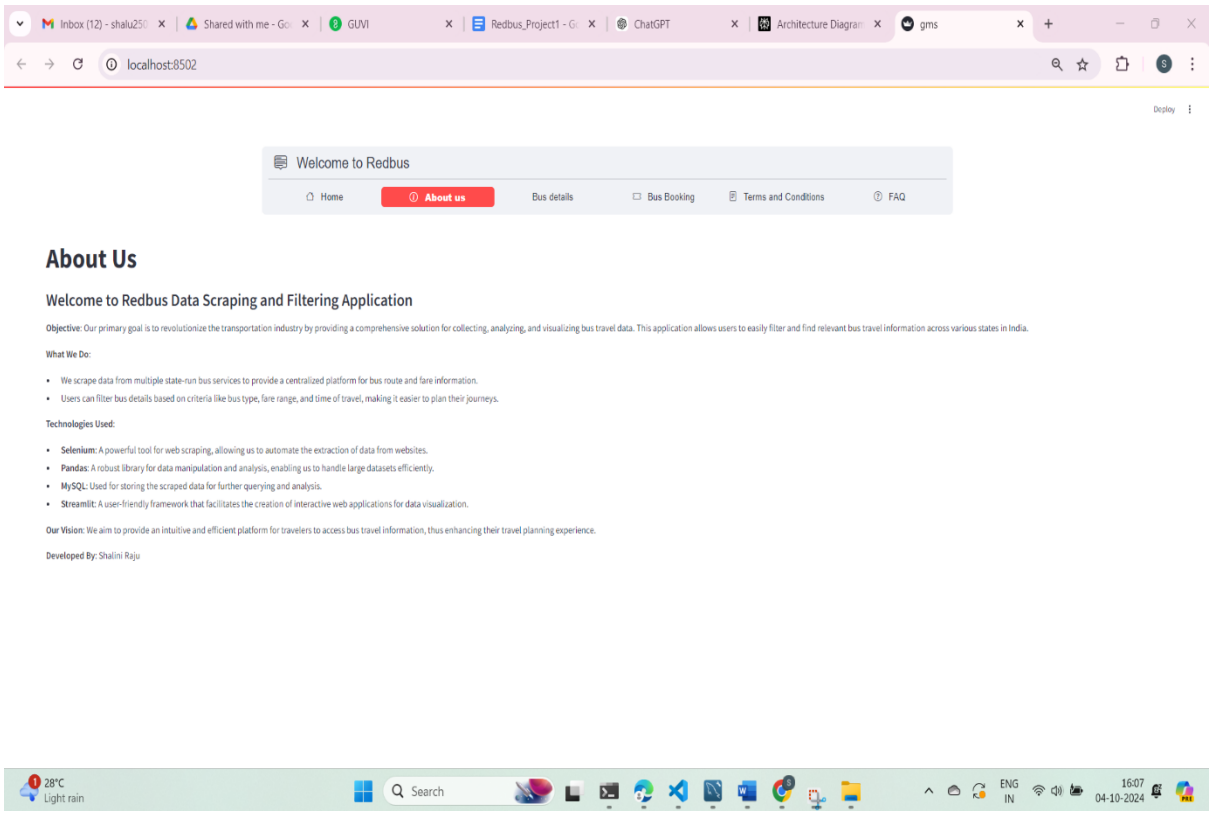
The system architecture consists of three main layers: **User Interface**, **Application**, and **Data Storage**.

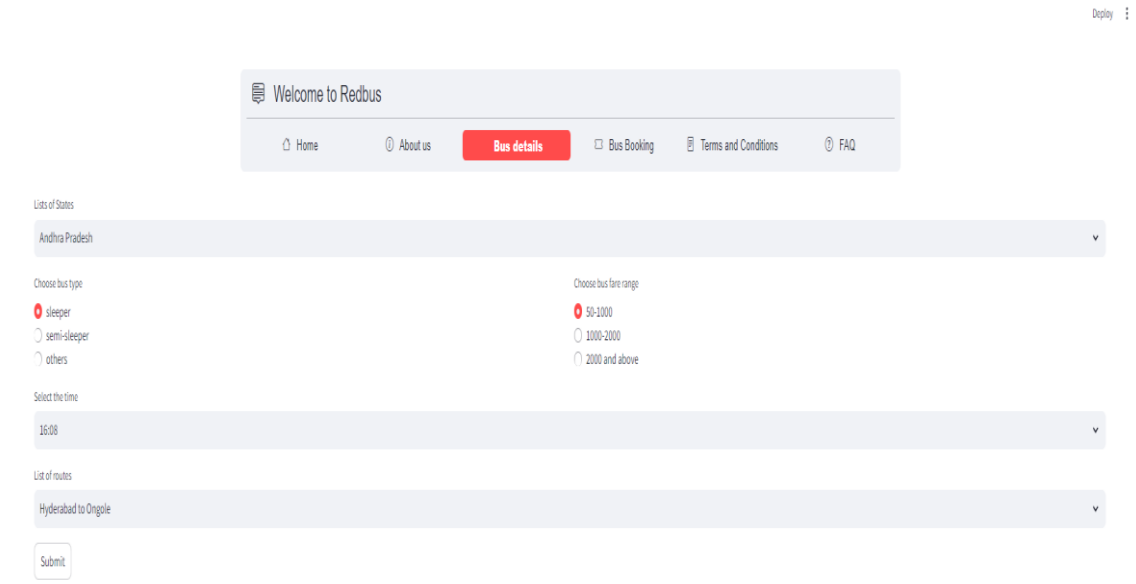
Architecture Overview:

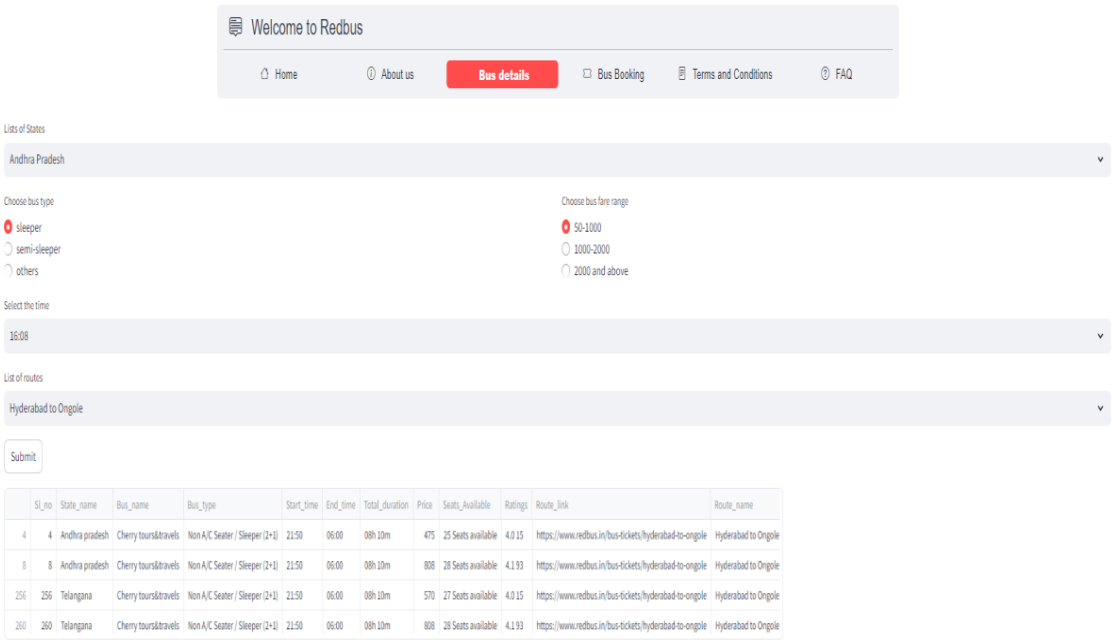


INPUT FOMS DESIGN:

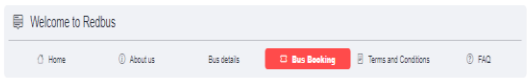








	Sl_no	State_name	Bus_name	Bus_type	Start_time	End_time	Total_duration	Price	Seats_Available	Ratings	Route_link	Route_name
4	4	Andhra pradesh	Cherry tours&travels	Non A/C Seater / Sleeper (2+1)	21:50	06:00	08h 10m	475	25 Seats available	4.0 15	https://www.redbus.in/bus-tickets/hyderabad-to-ongole	Hyderabad to Ongole
8	8	Andhra pradesh	Cherry tours&travels	Non A/C Seater / Sleeper (2+1)	21:50	06:00	08h 10m	808	28 Seats available	4.1 93	https://www.redbus.in/bus-tickets/hyderabad-to-ongole	Hyderabad to Ongole
256	256	Telangana	Cherry tours&travels	Non A/C Seater / Sleeper (2+1)	21:50	06:00	08h 10m	570	27 Seats available	4.0 15	https://www.redbus.in/bus-tickets/hyderabad-to-ongole	Hyderabad to Ongole
260	260	Telangana	Cherry tours&travels	Non A/C Seater / Sleeper (2+1)	21:50	06:00	08h 10m	808	28 Seats available	4.1 93	https://www.redbus.in/bus-tickets/hyderabad-to-ongole	Hyderabad to Ongole



Bus Booking

Select your travel details:

Choose State

Andhra Pradesh

Choose Route

Hyderabad to Ongole

Select the travel date

2024/10/04

Number of Passengers

2

Rate this Bus

4

1

5

Booking Summary

State: Andhra Pradesh

Route: Hyderabad to Ongole

Travel Date: 2024-10-04

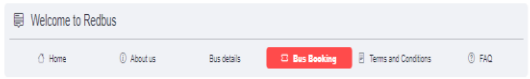
Number of Passengers: 2

Ticket Price (per passenger): ₹665.25

Total Price: ₹1330.50

Average Bus Rating: N/A

Your Rating for this Bus: 4 / 5.0



Bus Booking

Select your travel details:

Choose State

Andhra Pradesh

Choose Route

Hyderabad to Ongole

Select the travel date

2024/10/04

Number of Passengers

1

Rate this Bus

1

1

5

Booking Summary

State: Andhra Pradesh

Route: Hyderabad to Ongole

Travel Date: 2024-10-04

Number of Passengers: 1

Ticket Price (per passenger): ₹665.25

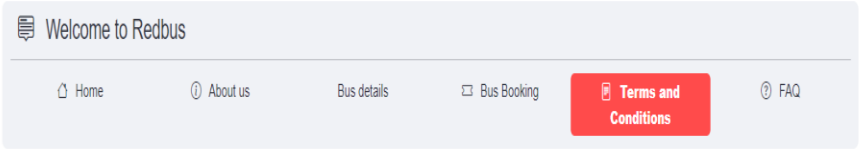
Total Price: ₹665.25

Average Bus Rating: N/A

Your Rating for this Bus: 1 / 5.0

Proceed to Payment

Bus ticket booked successfully!

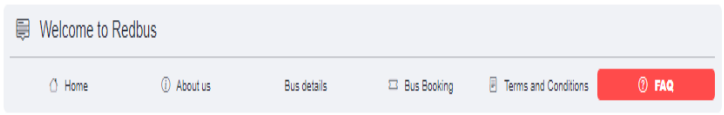


Terms and Conditions

Please read the following terms and conditions carefully:

1. **Accuracy of Information**: The data provided in this application is collected from various bus operators and may not always be up-to-date. While we strive to ensure the accuracy of the information, we cannot guarantee that all details are correct at all times.
2. **Use of Application**: The application is intended for personal use only. Unauthorized commercial use of this application is strictly prohibited.
3. **Booking Responsibility**: The user is solely responsible for any bookings made through third-party payment gateways. We are not liable for any issues that arise during or after the booking process.
4. **Privacy Policy**: The data collected from users will not be shared with third parties without explicit consent. However, anonymized usage statistics may be collected for improving the service.
5. **Modification of Terms**: We reserve the right to modify these terms and conditions at any time without prior notice. It is the user's responsibility to stay updated with the latest version of the terms.
6. **Limitation of Liability**: We are not responsible for any losses or damages arising from the use of this application, including but not limited to direct, indirect, incidental, punitive, and consequential damages.
7. **Governing Law**: These terms and conditions are governed by and construed in accordance with the laws of India. Any disputes arising in connection with these terms shall be subject to the exclusive jurisdiction of the courts in India.

By using this application, you agree to these terms and conditions.



Frequently Asked Questions

Find answers to common questions below:

1. **What is Redbus Data Scraping with Selenium & Streamlit?**
   This is a project that automates the extraction of bus route data from Redbus and provides a user-friendly interface for filtering and exploring the data.
2. **What states are supported?**
   The application currently supports bus routes from Andhra Pradesh, Telangana, Kerala, Rajasthan, Himachal Pradesh, Chandigarh, South Bengal, Uttar Pradesh, Punjab, West Bengal, Bihar, and Assam.
3. **How is the data collected?**
   Data is scraped from the Redbus website using Selenium, which automates the extraction of relevant bus details and routes.
4. **Can I book a bus through this app?**
   No, this app does not facilitate direct bus bookings. It provides information that helps users find buses and plan their journeys, but booking must be done through other platforms.
5. **How can I filter bus data?**
   You can filter data based on bus type, fare range, availability of seats, and travel timings using the app's built-in filters.
6. **Is the data updated in real-time?**
   No, the data is not updated in real-time. It is periodically refreshed through web scraping, so there may be some delay between updates.
7. **Who developed this application?**
   The application was developed by Shalini Raju as part of a project focusing on data scraping and visualization.

4.CODING

4.1 Selenium scraping data

```
#importing libraries
from selenium import webdriver
from selenium.webdriver import ActionChains
from selenium.webdriver.common.by import By
from selenium.webdriver.common.keys import Keys
from selenium.common.exceptions import TimeoutException, NoSuchElementException
import time
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC
import pandas as pd
!pip install selenium
#10 states links
state_links=["https://www.redbus.in/online-booking/apsrtc/?utm_source=rtchometile",
        "https://www.redbus.in/online-booking/tsrtc/?utm_source=rtchometile",
        "https://www.redbus.in/online-booking/ksrtc-kerala/?utm_source=rtchometile",
        "https://www.redbus.in/online-booking/rsrtc/?utm_source=rtchometile",
        "https://www.redbus.in/online-booking/hrtc/?utm_source=rtchometile",
        "https://www.redbus.in/online-booking/chandigarh-transport-undertaking-ctu/?utm_source=rtchometile",
        "https://www.redbus.in/online-booking/south-bengal-state-transport-corporation-sbstc/?utm_source=rtchometile",
        "https://www.redbus.in/online-booking/upsrtc",
        "https://www.redbus.in/online-booking/pepsu",
        "https://www.redbus.in/online-booking/wbtc-ctc/?utm_source=rtchometile",
        "https://www.redbus.in/online-booking/bihar-state-road-transport-corporation-bsrtc/?utm_source=rtchometile",
        "https://www.redbus.in/online-booking/astc"
]
#open the browser
```

```
driver = webdriver.Chrome()

#load the webpage
driver.get("https://www.redbus.in")
driver.get("https://www.redbus.in/online-booking/apsrtc/?utm_source=rtchometile")

time.sleep(3)

driver.maximize_window()
from selenium.webdriver.common.by import By
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC
from selenium.common.exceptions import NoSuchElementException,
ElementClickInterceptedException
import time

#implicit wait
def apsrtc_link_route(route_xpath):
    LINKS_APSRTC = []  # List to store APSRTC bus links
    ROUTE_APSRTC = []  # List to store APSRTC route names

    # Assuming 'paths' is passed as the route_xpath, which is XPath for bus routes.
    # Wait for the route elements to be present (update this XPath based on APSRTC page)
    wait = WebDriverWait(driver, 20)

    try:
        # Loop over multiple pages
        i = 0
        while True:
            # Wait for the route elements on the current page
            paths = wait.until(EC.presence_of_all_elements_located((By.XPATH, route_xpath)))
```

```
        # Retrieve names of the routes and their corresponding links
        for route in paths:
            ROUTE_APSRTC.append(route.text)
            LINKS_APSRTC.append(route.get_attribute("href"))  # Assuming it's an <a> tag

        # Wait for the pagination element to be present (update the XPath accordingly)
        pagination = wait.until(EC.presence_of_element_located((By.XPATH, '//*[@id="root"]/div/div[4]/div[12]')))

        # Construct the next button XPath dynamically (adjust according to actual button structure)
        next_button_xpath = f'//*[@id="root"]/div/div[4]/div[12]/div[1]//a[text()="{i+1}"]'

        if len(driver.find_elements(By.XPATH, next_button_xpath)) > 0:
            next_button = driver.find_element(By.XPATH, next_button_xpath)

            # Scroll into view and click the next button using JavaScript
            driver.execute_script("arguments[0].scrollIntoView(true);", next_button)
            time.sleep(2)  # Ensure some time for scrolling

            # Explicitly wait for the next button to be clickable
            wait.until(EC.element_to_be_clickable((By.XPATH, next_button_xpath)))
            driver.execute_script("arguments[0].click();", next_button)  # Click using JavaScript

            i += 1  # Move to the next pagination step
        else:
            print(f"No more pages to paginate at step {i}")
            break  # No more pages, exit the loop
```

```
    except (NoSuchElementException, ElementClickInterceptedException) as e:
        print(f"Encountered an issue at step {i}: {e}")

    return LINKS_APSRTC, ROUTE_APSRTC

# Example route XPath for APSRTC buses (you may need to inspect the page for the exact route element)
route_xpath = "//a[@class='route']"

# Calling the function to get links and routes
LINKS_APSRTC, ROUTE_APSRTC = apsrtc_link_route(route_xpath)

df_a = pd.DataFrame({"Route_name": ROUTE_APSRTC, "Route_link": LINKS_APSRTC})
df_a
df=df_a
#retrive the bus details
driver_k = webdriver.Chrome()
Bus_names_k = []
Bus_types_k = []
Start_Time_k = []
End_Time_k = []
Ratings_k = []
Total_Duration_k = []
Prices_k = []
Seats_Available_k = []
Route_names = []
Route_links = []

for i,r in df.iterrows():
    link=r["Route_link"]
    routes=r["Route_name"]
```

```
# Loop through each link
    driver_k.get(link)
    time.sleep(2)

    # Click on elements to reveal bus details
    elements = driver_k.find_elements(By.XPATH, f"//a[contains(@href, '{link}')]")
    for element in elements:
        element.click()
        time.sleep(2)

    # click elements to views bus
    try:
        clicks = driver_k.find_element(By.XPATH, "//div[@class='button']")
        clicks.click()
    except:
        continue
    time.sleep(2)

    scrolling = True
    while scrolling:
        old_page_source = driver_k.page_source

        # Use ActionChains to perform a PAGE_DOWN
        ActionChains(driver_k).send_keys(Keys.PAGE_DOWN).perform()

        time.sleep(5)

        new_page_source = driver_k.page_source
```

```
        if new_page_source == old_page_source:
            scrolling = False


    # Extract bus details
    bus_name = driver_k.find_elements(By.XPATH, "//div[@class='travels lh-24 f-bold d-color']")
    bus_type = driver_k.find_elements(By.XPATH, "//div[@class='bus-type f-12 m-top-16 l-color evBus']")
    start_time = driver_k.find_elements(By.XPATH, "//*[@class='dp-time f-19 d-color f-bold']")
    end_time = driver_k.find_elements(By.XPATH, "//*[@class='bp-time f-19 d-color disp-Inline']")
    total_duration = driver_k.find_elements(By.XPATH, "//*[@class='dur l-color lh-24']")
    try:
        rating = driver_k.find_elements(By.XPATH,"//div[@class='clearfix row-one']/div[@class='column-six p-right-10 w-10 fl']")
    except:
        continue
    price = driver_k.find_elements(By.XPATH, '//*[@class="fare d-block"]')
    seats = driver_k.find_elements(By.XPATH, "//div[contains(@class, 'seat-left')]")


    # Append data to respective lists
    for bus in bus_name:
        Bus_names_k.append(bus.text)
        Route_links.append(link)
        Route_names.append(routes)
    for bus_type_elem in bus_type:
        Bus_types_k.append(bus_type_elem.text)
    for start_time_elem in start_time:
        Start_Time_k.append(start_time_elem.text)
    for end_time_elem in end_time:
        End_Time_k.append(end_time_elem.text)
```

```
    for total_duration_elem in total_duration:
        Total_Duration_k.append(total_duration_elem.text)
    for ratings in rating:
        Ratings_k.append(ratings.text)
    for price_elem in price:
        Prices_k.append(price_elem.text)
    for seats_elem in seats:
        Seats_Available_k.append(seats_elem.text)

print("Successfully Completed")
# from list to convert data frame
data = {
    'Bus_name': Bus_names_k,
    'Bus_type': Bus_types_k,
    'Start_time': Start_Time_k,
    'End_time': End_Time_k,
    'Total_duration': Total_Duration_k,
    'Price': Prices_k,
    "Seats_Available":Seats_Available_k,
    "Ratings":Ratings_k,
    'Route_link': Route_links,
    'Route_name': Route_names
}

df1 = pd.DataFrame(data)
#convert dataframe to csv

df1
df1.to_csv("C:/Users/Shalini/Downloads/AP_data.csv")#done1
```

```
from selenium.webdriver.common.by import By
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC
from selenium.common.exceptions import NoSuchElementException,
ElementClickInterceptedException
import time

driver = webdriver.Chrome()

#load the webpage
driver.get("https://www.redbus.in")
driver.get("https://www.redbus.in/online-booking/tsrtc/?utm_source=rtchometile")

time.sleep(3)

driver.maximize_window()

def tsrtc_link_route(route_xpath):
    LINKS_TSRTC = []  # List to store TSRTC bus links
    ROUTE_TSRTC = []  # List to store TSRTC route names

    # Wait setup
    wait = WebDriverWait(driver, 10)

    try:
        i = 0
        while True:
            # Wait for the route elements to be present
            paths = wait.until(EC.presence_of_all_elements_located((By.XPATH, route_xpath)))
```

```
            # Retrieve names of the routes and their corresponding links
            for route in paths:
                ROUTE_TSRTC.append(route.text)
                LINKS_TSRTC.append(route.get_attribute("href"))  # Assuming it's an <a> tag

            # Wait for the pagination element to be present (update the XPath if needed)
            pagination = wait.until(EC.presence_of_element_located((By.XPATH,
'//*[@id="root"]/div/div[4]/div[12]')))

            # Construct the next button XPath dynamically (adjust according to actual button
structure)
            next_button_xpath = f'//*[@id="root"]/div/div[4]/div[12]/div[1]//a[text()="{i+1}"]'

            if len(driver.find_elements(By.XPATH, next_button_xpath)) > 0:
                next_button = driver.find_element(By.XPATH, next_button_xpath)

                # Scroll into view and click the next button using JavaScript
                driver.execute_script("arguments[0].scrollIntoView(true);", next_button)
                time.sleep(2)  # Ensure some time for scrolling

                # Explicitly wait for the next button to be clickable
                wait.until(EC.element_to_be_clickable((By.XPATH, next_button_xpath)))
                driver.execute_script("arguments[0].click();", next_button)  # Click using
JavaScript

                i += 1  # Move to the next pagination step
            else:
                print(f"No more pages to paginate at step {i}")
                break  # No more pages, exit the loop

        except (NoSuchElementException, ElementClickInterceptedException) as e:
```

```
        print(f"Encountered an issue at step {i}: {e}")

    return LINKS_TSRTC, ROUTE_TSRTC

# Example route XPath for TSRTC buses (you may need to inspect the page for the exact route element)
route_xpath = "//a[@class='route']"

# Calling the function to get links and routes
LINKS_TSRTC, ROUTE_TSRTC = tsrtc_link_route(route_xpath)

# Create a DataFrame for TSRTC
df_tsrtc = pd.DataFrame({"Route_name": ROUTE_TSRTC,"Route_link": LINKS_TSRTC})
# Display the DataFrame
df_tsrtc

df=df_tsrtc
#retrive the bus details
driver_k = webdriver.Chrome()
Bus_names_k = []
Bus_types_k = []
Start_Time_k = []
End_Time_k = []
Ratings_k = []
Total_Duration_k = []
Prices_k = []
Seats_Available_k = []
Route_names = []
Route_links = []

for i,r in df.iterrows():
```

```
    link=r["Route_link"]
    routes=r["Route_name"]

# Loop through each link
    driver_k.get(link)
    time.sleep(2)

    # Click on elements to reveal bus details
    elements = driver_k.find_elements(By.XPATH, f"//a[contains(@href, '{link}')]")
    for element in elements:
        element.click()
        time.sleep(2)

    # click elements to views bus
    try:
        clicks = driver_k.find_element(By.XPATH, "//div[@class='button']")
        clicks.click()
    except:
        continue
    time.sleep(2)

    scrolling = True
    while scrolling:
        old_page_source = driver_k.page_source

        # Use ActionChains to perform a PAGE_DOWN
        ActionChains(driver_k).send_keys(Keys.PAGE_DOWN).perform()

        time.sleep(5)
```

```
        new_page_source = driver_k.page_source


        if new_page_source == old_page_source:
            scrolling = False


    # Extract bus details
    bus_name = driver_k.find_elements(By.XPATH, "//div[@class='travels lh-24 f-bold d-color']")
    bus_type = driver_k.find_elements(By.XPATH, "//div[@class='bus-type f-12 m-top-16 l-color evBus']")
    start_time = driver_k.find_elements(By.XPATH, "//*[@class='dp-time f-19 d-color f-bold']")
    end_time = driver_k.find_elements(By.XPATH, "//*[@class='bp-time f-19 d-color disp-Inline']")
    total_duration = driver_k.find_elements(By.XPATH, "//*[@class='dur l-color lh-24']")
    try:
        rating = driver_k.find_elements(By.XPATH,"//div[@class='clearfix row-one']/div[@class='column-six p-right-10 w-10 fl']")
    except:
        continue
    price = driver_k.find_elements(By.XPATH, '//*[@class="fare d-block"]')
    seats = driver_k.find_elements(By.XPATH, "//div[contains(@class, 'seat-left')]")


    # Append data to respective lists
    for bus in bus_name:
        Bus_names_k.append(bus.text)
        Route_links.append(link)
        Route_names.append(routes)
    for bus_type_elem in bus_type:
        Bus_types_k.append(bus_type_elem.text)
    for start_time_elem in start_time:
        Start_Time_k.append(start_time_elem.text)
```

```
    for end_time_elem in end_time:
        End_Time_k.append(end_time_elem.text)
    for total_duration_elem in total_duration:
        Total_Duration_k.append(total_duration_elem.text)
    for ratings in rating:
        Ratings_k.append(ratings.text)
    for price_elem in price:
        Prices_k.append(price_elem.text)
    for seats_elem in seats:
        Seats_Available_k.append(seats_elem.text)

print("Successfully Completed")
# from list to convert data frame
data = {
    'Bus_name': Bus_names_k,
    'Bus_type': Bus_types_k,
    'Start_time': Start_Time_k,
    'End_time': End_Time_k,
    'Total_duration': Total_Duration_k,
    'Price': Prices_k,
    "Seats_Available":Seats_Available_k,
    "Ratings":Ratings_k,
    'Route_link': Route_links,
    'Route_name': Route_names
}

df1 = pd.DataFrame(data)
#convert dataframe to csv

df1
```

```
df1.to_csv("C:/Users/Shalini/Downloads/tsrtc_data2.csv")#done2



from selenium.webdriver.common.by import By

from selenium.webdriver.support.ui import WebDriverWait

from selenium.webdriver.support import expected_conditions as EC

from selenium.common.exceptions import NoSuchElementException,
ElementClickInterceptedException

import time


driver = webdriver.Chrome()


#load the webpage

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

driver.get("https://www.redbus.in/online-booking/ksrtc-kerala/?utm_source=rtchometile")


time.sleep(3)


driver.maximize_window()



def ksrtc_link_route(route_xpath):
    LINKS_KSRTC = []  # List to store KSRTC bus links
    ROUTE_KSRTC = []  # List to store KSRTC route names


    # Wait setup
    wait = WebDriverWait(driver, 10)


    try:
        i = 0
        while True:
```

```
        # Wait for the route elements to be present
        paths = wait.until(EC.presence_of_all_elements_located((By.XPATH, route_xpath)))

        # Retrieve names of the routes and their corresponding links
        for route in paths:
            ROUTE_KSRTC.append(route.text)
            LINKS_KSRTC.append(route.get_attribute("href"))  # Assuming it's an <a> tag

        # Wait for the pagination element to be present (update the XPath if needed)
        pagination = wait.until(EC.presence_of_element_located((By.XPATH,
'//*[@id="root"]/div/div[4]/div[12]')))

        # Construct the next button XPath dynamically (adjust according to actual button
structure)
        next_button_xpath = f'//*[@id="root"]/div/div[4]/div[12]/div[1]//a[text()="{i+1}"]'

        if len(driver.find_elements(By.XPATH, next_button_xpath)) > 0:
            next_button = driver.find_element(By.XPATH, next_button_xpath)

            # Scroll into view and click the next button using JavaScript
            driver.execute_script("arguments[0].scrollIntoView(true);", next_button)
            time.sleep(2)  # Ensure some time for scrolling

            # Explicitly wait for the next button to be clickable
            wait.until(EC.element_to_be_clickable((By.XPATH, next_button_xpath)))
            driver.execute_script("arguments[0].click();", next_button)  # Click using
JavaScript

            i += 1  # Move to the next pagination step
        else:
            print(f"No more pages to paginate at step {i}")
```

```
            break  # No more pages, exit the loop

    except (NoSuchElementException, ElementClickInterceptedException) as e:
        print(f"Encountered an issue at step {i}: {e}")

    return LINKS_KSRTC, ROUTE_KSRTC

# Example route XPath for KSRTC buses (you may need to inspect the page for the exact route element)
route_xpath = "//a[@class='route']"

# Calling the function to get links and routes
LINKS_KSRTC, ROUTE_KSRTC = ksrtc_link_route(route_xpath)

import pandas as pd

# Create a DataFrame for KSRTC
df_ksrtc = pd.DataFrame({"Route_name": ROUTE_KSRTC,"Route_link": LINKS_KSRTC})

# Display the DataFrame (in Jupyter notebook or similar environments)
df_ksrtc

df = df_ksrtc
#retrive the bus details
driver_k = webdriver.Chrome()
Bus_names_k = []
Bus_types_k = []
Start_Time_k = []
End_Time_k = []
Ratings_k = []
```

```
Total_Duration_k = []
Prices_k = []
Seats_Available_k = []
Route_names = []
Route_links = []

for i,r in df.iterrows():
    link=r["Route_link"]
    routes=r["Route_name"]

# Loop through each link
    driver_k.get(link)
    time.sleep(2)

    # Click on elements to reveal bus details
    elements = driver_k.find_elements(By.XPATH, f"//a[contains(@href, '{link}')]")
    for element in elements:
        element.click()
        time.sleep(2)

    # click elements to views bus
    try:
        clicks = driver_k.find_element(By.XPATH, "//div[@class='button']")
        clicks.click()
    except:
        continue
    time.sleep(2)

    scrolling = True
    while scrolling:
```

```
        old_page_source = driver_k.page_source

        # Use ActionChains to perform a PAGE_DOWN
        ActionChains(driver_k).send_keys(Keys.PAGE_DOWN).perform()

        time.sleep(5)

        new_page_source = driver_k.page_source

        if new_page_source == old_page_source:
            scrolling = False

    # Extract bus details
    bus_name = driver_k.find_elements(By.XPATH, "//div[@class='travels lh-24 f-bold d-color']")
    bus_type = driver_k.find_elements(By.XPATH, "//div[@class='bus-type f-12 m-top-16 l-color evBus']")
    start_time = driver_k.find_elements(By.XPATH, "//*[@class='dp-time f-19 d-color f-bold']")
    end_time = driver_k.find_elements(By.XPATH, "//*[@class='bp-time f-19 d-color disp-Inline']")
    total_duration = driver_k.find_elements(By.XPATH, "//*[@class='dur l-color lh-24']")
    try:
        rating = driver_k.find_elements(By.XPATH,"//div[@class='clearfix row-one']/div[@class='column-six p-right-10 w-10 fl']")
    except:
        continue
    price = driver_k.find_elements(By.XPATH, '//*[@class="fare d-block"]')
    seats = driver_k.find_elements(By.XPATH, "//div[contains(@class, 'seat-left')]")

    # Append data to respective lists
    for bus in bus_name:
```

```
            Bus_names_k.append(bus.text)
            Route_links.append(link)
            Route_names.append(routes)
        for bus_type_elem in bus_type:
            Bus_types_k.append(bus_type_elem.text)
        for start_time_elem in start_time:
            Start_Time_k.append(start_time_elem.text)
        for end_time_elem in end_time:
            End_Time_k.append(end_time_elem.text)
        for total_duration_elem in total_duration:
            Total_Duration_k.append(total_duration_elem.text)
        for ratings in rating:
            Ratings_k.append(ratings.text)
        for price_elem in price:
            Prices_k.append(price_elem.text)
        for seats_elem in seats:
            Seats_Available_k.append(seats_elem.text)

print("Successfully Completed")
# from list to convert data frame
data = {
    'Bus_name': Bus_names_k,
    'Bus_type': Bus_types_k,
    'Start_time': Start_Time_k,
    'End_time': End_Time_k,
    'Total_duration': Total_Duration_k,
    'Price': Prices_k,
    "Seats_Available":Seats_Available_k,
    "Ratings":Ratings_k,
    'Route_link': Route_links,
```

```
    'Route_name': Route_names
}

df1 = pd.DataFrame(data)
#convert dataframe to csv

df1
df1.to_csv("C:/Users/Shalini/Downloads/ksrtc_data3.csv") #done3



from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC
from selenium.common.exceptions import NoSuchElementException,
ElementClickInterceptedException
import time
import pandas as pd

# Initialize the WebDriver (make sure you have the appropriate WebDriver installed)
driver = webdriver.Chrome()

# Load the RSRTC webpage
driver.get("https://www.redbus.in")
driver.get("https://www.redbus.in/online-booking/rsrtc/?utm_source=rtchometile")

time.sleep(3)

driver.maximize_window()

def rsrtc_link_route(route_xpath):
```

```
    LINKS_RSRTC = []  # List to store RSRTC bus links
    ROUTE_RSRTC = []  # List to store RSRTC route names

    # Wait setup
    wait = WebDriverWait(driver, 10)

    try:
        i = 0
        while True:
            # Wait for the route elements to be present
            paths = wait.until(EC.presence_of_all_elements_located((By.XPATH, route_xpath)))

            # Retrieve names of the routes and their corresponding links
            for route in paths:
                ROUTE_RSRTC.append(route.text)
                LINKS_RSRTC.append(route.get_attribute("href"))  # Assuming it's an <a> tag

            # Wait for the pagination element to be present (update the XPath if needed)
            pagination = wait.until(EC.presence_of_element_located((By.XPATH,
'//*[@id="root"]/div/div[4]/div[12]')))

            # Construct the next button XPath dynamically (adjust according to actual button
structure)
            next_button_xpath = f'//*[@id="root"]/div/div[4]/div[12]/div[1]//a[text()="{i+1}"]'

            if len(driver.find_elements(By.XPATH, next_button_xpath)) > 0:
                next_button = driver.find_element(By.XPATH, next_button_xpath)

                # Scroll into view and click the next button using JavaScript
                driver.execute_script("arguments[0].scrollIntoView(true);", next_button)
                time.sleep(2)  # Ensure some time for scrolling
```

```
            # Explicitly wait for the next button to be clickable
            wait.until(EC.element_to_be_clickable((By.XPATH, next_button_xpath)))
            driver.execute_script("arguments[0].click();", next_button)  # Click using
JavaScript

            i += 1  # Move to the next pagination step
        else:
            print(f"No more pages to paginate at step {i}")
            break  # No more pages, exit the loop

    except (NoSuchElementException, ElementClickInterceptedException) as e:
        print(f"Encountered an issue at step {i}: {e}")

    return LINKS_RSRTC, ROUTE_RSRTC

# Example route XPath for RSRTC buses (you may need to inspect the page for the exact
route element)
route_xpath_rsrtc = "//a[@class='route']"

# Calling the function to get links and routes
LINKS_RSRTC, ROUTE_RSRTC = rsrtc_link_route(route_xpath_rsrtc)

import pandas as pd

# Create a DataFrame for RSRTC
df_rsrtc = pd.DataFrame({"Route_name": ROUTE_RSRTC,"Route_link": LINKS_RSRTC})
# Display the DataFrame
(df_rsrtc)
df=df_rsrtc
#retrive the bus details
```

```
driver_k = webdriver.Chrome()
Bus_names_k = []
Bus_types_k = []
Start_Time_k = []
End_Time_k = []
Ratings_k = []
Total_Duration_k = []
Prices_k = []
Seats_Available_k = []
Route_names = []
Route_links = []

for i,r in df.iterrows():
    link=r["Route_link"]
    routes=r["Route_name"]

# Loop through each link
    driver_k.get(link)
    time.sleep(2)

    # Click on elements to reveal bus details
    elements = driver_k.find_elements(By.XPATH, f"//a[contains(@href, '{link}')]")
    for element in elements:
        element.click()
        time.sleep(2)

    # click elements to views bus
    try:
        clicks = driver_k.find_element(By.XPATH, "//div[@class='button']")
        clicks.click()
```

```
    except:
        continue
    time.sleep(2)

    scrolling = True
    while scrolling:
        old_page_source = driver_k.page_source

        # Use ActionChains to perform a PAGE_DOWN
        ActionChains(driver_k).send_keys(Keys.PAGE_DOWN).perform()

        time.sleep(5)

        new_page_source = driver_k.page_source

        if new_page_source == old_page_source:
            scrolling = False

    # Extract bus details
    bus_name = driver_k.find_elements(By.XPATH, "//div[@class='travels lh-24 f-bold d-color']")
    bus_type = driver_k.find_elements(By.XPATH, "//div[@class='bus-type f-12 m-top-16 l-color evBus']")
    start_time = driver_k.find_elements(By.XPATH, "//*[@class='dp-time f-19 d-color f-bold']")
    end_time = driver_k.find_elements(By.XPATH, "//*[@class='bp-time f-19 d-color disp-Inline']")
    total_duration = driver_k.find_elements(By.XPATH, "//*[@class='dur l-color lh-24']")
    try:
        rating = driver_k.find_elements(By.XPATH,"//div[@class='clearfix row-one']/div[@class='column-six p-right-10 w-10 fl']")
    except:
```

```
        continue
    price = driver_k.find_elements(By.XPATH, '//*[@class="fare d-block"]')
    seats = driver_k.find_elements(By.XPATH, "//div[contains(@class, 'seat-left')]")

    # Append data to respective lists
    for bus in bus_name:
        Bus_names_k.append(bus.text)
        Route_links.append(link)
        Route_names.append(routes)
    for bus_type_elem in bus_type:
        Bus_types_k.append(bus_type_elem.text)
    for start_time_elem in start_time:
        Start_Time_k.append(start_time_elem.text)
    for end_time_elem in end_time:
        End_Time_k.append(end_time_elem.text)
    for total_duration_elem in total_duration:
        Total_Duration_k.append(total_duration_elem.text)
    for ratings in rating:
        Ratings_k.append(ratings.text)
    for price_elem in price:
        Prices_k.append(price_elem.text)
    for seats_elem in seats:
        Seats_Available_k.append(seats_elem.text)

print("Successfully Completed")
# from list to convert data frame
data = {
    'Bus_name': Bus_names_k,
    'Bus_type': Bus_types_k,
    'Start_time': Start_Time_k,
```

```
    'End_time': End_Time_k,
    'Total_duration': Total_Duration_k,
    'Price': Prices_k,
    "Seats_Available":Seats_Available_k,
    "Ratings":Ratings_k,
    'Route_link': Route_links,
    'Route_name': Route_names
}

df1 = pd.DataFrame(data)
#convert dataframe to csv

df1
df1.to_csv("C:/Users/Shalini/Downloads/rsrtc_data4.csv") #done4



from selenium.webdriver.common.by import By
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC
from selenium.common.exceptions import NoSuchElementException,
ElementClickInterceptedException
import time
import pandas as pd
from selenium import webdriver

# Initialize WebDriver
driver = webdriver.Chrome()

# Load the HRTC webpage
driver.get("https://www.redbus.in")
driver.get("https://www.redbus.in/online-booking/hrtc/?utm_source=rtchometile")
```

```
time.sleep(3)
driver.maximize_window()

def hrtc_link_route(route_xpath):
    LINKS_HRTC = []  # List to store HRTC bus links
    ROUTE_HRTC = []  # List to store HRTC route names

    # Wait setup
    wait = WebDriverWait(driver, 10)

    try:
        i = 0
        while True:
            # Wait for the route elements to be present
            paths = wait.until(EC.presence_of_all_elements_located((By.XPATH, route_xpath)))

            # Retrieve names of the routes and their corresponding links
            for route in paths:
                ROUTE_HRTC.append(route.text)
                LINKS_HRTC.append(route.get_attribute("href"))  # Assuming it's an <a> tag

            # Wait for the pagination element to be present (update the XPath if needed)
            pagination = wait.until(EC.presence_of_element_located((By.XPATH,
'//*[@id="root"]/div/div[4]/div[12]')))

            # Construct the next button XPath dynamically (adjust according to actual button
structure)
            next_button_xpath = f'//*[@id="root"]/div/div[4]/div[12]/div[1]//a[text()="{i+1}"]'

            if len(driver.find_elements(By.XPATH, next_button_xpath)) > 0:
```

```
            next_button = driver.find_element(By.XPATH, next_button_xpath)

            # Scroll into view and click the next button using JavaScript
            driver.execute_script("arguments[0].scrollIntoView(true);", next_button)
            time.sleep(2)  # Ensure some time for scrolling

            # Explicitly wait for the next button to be clickable
            wait.until(EC.element_to_be_clickable((By.XPATH, next_button_xpath)))
            driver.execute_script("arguments[0].click();", next_button)  # Click using
JavaScript

            i += 1  # Move to the next pagination step
        else:
            print(f"No more pages to paginate at step {i}")
            break  # No more pages, exit the loop

    except (NoSuchElementException, ElementClickInterceptedException) as e:
        print(f"Encountered an issue at step {i}: {e}")

    return LINKS_HRTC, ROUTE_HRTC

# Example route XPath for HRTC buses (you may need to inspect the page for the exact route
element)
route_xpath_hrtc = "//a[@class='route']"

# Calling the function to get links and routes
LINKS_HRTC, ROUTE_HRTC = hrtc_link_route(route_xpath_hrtc)



# Create a DataFrame for HRTC
df_hrtc = pd.DataFrame({"Route_name": ROUTE_HRTC,"Route_link": LINKS_HRTC})
```

```
(df_hrtc)
df=df_hrtc
#retrive the bus details
driver_k = webdriver.Chrome()
Bus_names_k = []
Bus_types_k = []
Start_Time_k = []
End_Time_k = []
Ratings_k = []
Total_Duration_k = []
Prices_k = []
Seats_Available_k = []
Route_names = []
Route_links = []

for i,r in df.iterrows():
    link=r["Route_link"]
    routes=r["Route_name"]

# Loop through each link
    driver_k.get(link)
    time.sleep(2)

    # Click on elements to reveal bus details
    elements = driver_k.find_elements(By.XPATH, f"//a[contains(@href, '{link}')]")
    for element in elements:
        element.click()
        time.sleep(2)

    # click elements to views bus
```

```
    try:
        clicks = driver_k.find_element(By.XPATH, "//div[@class='button']")
        clicks.click()
    except:
        continue
    time.sleep(2)

    scrolling = True
    while scrolling:
        old_page_source = driver_k.page_source

        # Use ActionChains to perform a PAGE_DOWN
        ActionChains(driver_k).send_keys(Keys.PAGE_DOWN).perform()

        time.sleep(5)

        new_page_source = driver_k.page_source

        if new_page_source == old_page_source:
            scrolling = False

    # Extract bus details
    bus_name = driver_k.find_elements(By.XPATH, "//div[@class='travels lh-24 f-bold d-color']")
    bus_type = driver_k.find_elements(By.XPATH, "//div[@class='bus-type f-12 m-top-16 l-color evBus']")
    start_time = driver_k.find_elements(By.XPATH, "//*[@class='dp-time f-19 d-color f-bold']")
    end_time = driver_k.find_elements(By.XPATH, "//*[@class='bp-time f-19 d-color disp-Inline']")
    total_duration = driver_k.find_elements(By.XPATH, "//*[@class='dur l-color lh-24']")
    try:
```

```
        rating = driver_k.find_elements(By.XPATH,"//div[@class='clearfix row-
one']/div[@class='column-six p-right-10 w-10 fl']")
    except:
        continue
    price = driver_k.find_elements(By.XPATH, '//*[@class="fare d-block"]')
    seats = driver_k.find_elements(By.XPATH, "//div[contains(@class, 'seat-left')]")

    # Append data to respective lists
    for bus in bus_name:
        Bus_names_k.append(bus.text)
        Route_links.append(link)
        Route_names.append(routes)
    for bus_type_elem in bus_type:
        Bus_types_k.append(bus_type_elem.text)
    for start_time_elem in start_time:
        Start_Time_k.append(start_time_elem.text)
    for end_time_elem in end_time:
        End_Time_k.append(end_time_elem.text)
    for total_duration_elem in total_duration:
        Total_Duration_k.append(total_duration_elem.text)
    for ratings in rating:
        Ratings_k.append(ratings.text)
    for price_elem in price:
        Prices_k.append(price_elem.text)
    for seats_elem in seats:
        Seats_Available_k.append(seats_elem.text)

print("Successfully Completed")
# from list to convert data frame
data = {
    'Bus_name': Bus_names_k,
```

```
    'Bus_type': Bus_types_k,
    'Start_time': Start_Time_k,
    'End_time': End_Time_k,
    'Total_duration': Total_Duration_k,
    'Price': Prices_k,
    "Seats_Available":Seats_Available_k,
    "Ratings":Ratings_k,
    'Route_link': Route_links,
    'Route_name': Route_names
}

df1 = pd.DataFrame(data)
#convert dataframe to csv

df1
df1.to_csv("C:/Users/Shalini/Downloads/hrtc_data5.csv") #done5




from selenium.webdriver.common.by import By
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC
from selenium.common.exceptions import NoSuchElementException,
ElementClickInterceptedException
import time
import pandas as pd
from selenium import webdriver

# Initialize WebDriver
driver = webdriver.Chrome()
```

```
# Load the CTU webpage on Redbus
driver.get("https://www.redbus.in")
driver.get("https://www.redbus.in/online-booking/chandigarh-transport-undertaking-ctu/?utm_source=rtchometile")  # Update the URL if necessary

time.sleep(3)
driver.maximize_window()

def ctu_link_route(route_xpath):
    LINKS_CTU = []  # List to store CTU bus links
    ROUTE_CTU = []  # List to store CTU route names

    # Wait setup
    wait = WebDriverWait(driver, 10)

    try:
        i = 0
        while True:
            # Wait for the route elements to be present
            paths = wait.until(EC.presence_of_all_elements_located((By.XPATH, route_xpath)))

            # Retrieve names of the routes and their corresponding links
            for route in paths:
                ROUTE_CTU.append(route.text)
                LINKS_CTU.append(route.get_attribute("href"))  # Assuming it's an <a> tag

            # Wait for the pagination element to be present (update the XPath if needed)
            pagination = wait.until(EC.presence_of_element_located((By.XPATH, '//*[@id="root"]/div/div[4]/div[12]')))  # Adjust as per actual pagination structure
```

```
            # Construct the next button XPath dynamically (adjust according to actual button structure)
            next_button_xpath = f'//*[@id="root"]/div/div[4]/div[12]/div[1]//a[text()="{i+1}"]'

            if len(driver.find_elements(By.XPATH, next_button_xpath)) > 0:
                next_button = driver.find_element(By.XPATH, next_button_xpath)

                # Scroll into view and click the next button using JavaScript
                driver.execute_script("arguments[0].scrollIntoView(true);", next_button)
                time.sleep(2)  # Ensure some time for scrolling

                # Explicitly wait for the next button to be clickable
                wait.until(EC.element_to_be_clickable((By.XPATH, next_button_xpath)))
                driver.execute_script("arguments[0].click();", next_button)  # Click using JavaScript

                i += 1  # Move to the next pagination step
            else:
                print(f"No more pages to paginate at step {i}")
                break  # No more pages, exit the loop

        except (NoSuchElementException, ElementClickInterceptedException) as e:
            print(f"Encountered an issue at step {i}: {e}")

        return LINKS_CTU, ROUTE_CTU

    # Example route XPath for CTU buses (you may need to inspect the page for the exact route element)
    route_xpath_ctu = "//a[@class='route']"  # Adjust based on actual page structure

    # Calling the function to get links and routes
```

```
LINKS_CTU, ROUTE_CTU = ctu_link_route(route_xpath_ctu)

# Create DataFrame
df_ctu = pd.DataFrame({"Route_name": ROUTE_CTU, "Route_link": LINKS_CTU})
df_ctu
df =df_ctu
#retrive the bus details
driver_k = webdriver.Chrome()
Bus_names_k = []
Bus_types_k = []
Start_Time_k = []
End_Time_k = []
Ratings_k = []
Total_Duration_k = []
Prices_k = []
Seats_Available_k = []
Route_names = []
Route_links = []

for i,r in df.iterrows():
    link=r["Route_link"]
    routes=r["Route_name"]

# Loop through each link
    driver_k.get(link)
    time.sleep(2)

    # Click on elements to reveal bus details
    elements = driver_k.find_elements(By.XPATH, f"//a[contains(@href, '{link}')]")
    for element in elements:
```

```
        element.click()
        time.sleep(2)

    # click elements to views bus
    try:
        clicks = driver_k.find_element(By.XPATH, "//div[@class='button']")
        clicks.click()
    except:
        continue
    time.sleep(2)

    scrolling = True
    while scrolling:
        old_page_source = driver_k.page_source

        # Use ActionChains to perform a PAGE_DOWN
        ActionChains(driver_k).send_keys(Keys.PAGE_DOWN).perform()

        time.sleep(5)

        new_page_source = driver_k.page_source

        if new_page_source == old_page_source:
            scrolling = False

    # Extract bus details
    bus_name = driver_k.find_elements(By.XPATH, "//div[@class='travels lh-24 f-bold d-
color']")
    bus_type = driver_k.find_elements(By.XPATH, "//div[@class='bus-type f-12 m-top-16 l-
color evBus']")
```

```
    start_time = driver_k.find_elements(By.XPATH, "//*[@class='dp-time f-19 d-color f-bold']")
    end_time = driver_k.find_elements(By.XPATH, "//*[@class='bp-time f-19 d-color disp-Inline']")
    total_duration = driver_k.find_elements(By.XPATH, "//*[@class='dur l-color lh-24']")
    try:
        rating = driver_k.find_elements(By.XPATH,"//div[@class='clearfix row-one']/div[@class='column-six p-right-10 w-10 fl']")
    except:
        continue
    price = driver_k.find_elements(By.XPATH, '//*[@class="fare d-block"]')
    seats = driver_k.find_elements(By.XPATH, "//div[contains(@class, 'seat-left')]")

    # Append data to respective lists
    for bus in bus_name:
        Bus_names_k.append(bus.text)
        Route_links.append(link)
        Route_names.append(routes)
    for bus_type_elem in bus_type:
        Bus_types_k.append(bus_type_elem.text)
    for start_time_elem in start_time:
        Start_Time_k.append(start_time_elem.text)
    for end_time_elem in end_time:
        End_Time_k.append(end_time_elem.text)
    for total_duration_elem in total_duration:
        Total_Duration_k.append(total_duration_elem.text)
    for ratings in rating:
        Ratings_k.append(ratings.text)
    for price_elem in price:
        Prices_k.append(price_elem.text)
    for seats_elem in seats:
```

```
        Seats_Available_k.append(seats_elem.text)

print("Successfully Completed")
# from list to convert data frame
data = {
    'Bus_name': Bus_names_k,
    'Bus_type': Bus_types_k,
    'Start_time': Start_Time_k,
    'End_time': End_Time_k,
    'Total_duration': Total_Duration_k,
    'Price': Prices_k,
    "Seats_Available":Seats_Available_k,
    "Ratings":Ratings_k,
    'Route_link': Route_links,
    'Route_name': Route_names
}

df1 = pd.DataFrame(data)
#convert dataframe to csv

df1
df1.to_csv("C:/Users/Shalini/Downloads/ctu_data6.csv") #done6



from selenium.webdriver.common.by import By
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC
from selenium.common.exceptions import NoSuchElementException,
ElementClickInterceptedException
import time
import pandas as pd
```

```
from selenium import webdriver

# Initialize the WebDriver
driver = webdriver.Chrome()

# Load the Redbus homepage
driver.get("https://www.redbus.in")
time.sleep(3)
driver.maximize_window()

# Load the SBSTC page on Redbus
driver.get("https://www.redbus.in/online-booking/south-bengal-state-transport-corporation-sbstc/?utm_source=rtchometile")
time.sleep(3)

# Function to scrape SBSTC route links and names
def sbstc_link_route(route_xpath):
    LINKS_SBSTC = []  # List to store SBSTC bus links
    ROUTE_SBSTC = []  # List to store SBSTC route names

    # Wait setup
    wait = WebDriverWait(driver, 20)

    try:
        i = 0
        while True:
            # Wait for the route elements on the current page
            paths = wait.until(EC.presence_of_all_elements_located((By.XPATH, route_xpath)))

            # Retrieve names of the routes and their corresponding links
            for route in paths:
```

```
            ROUTE_SBSTC.append(route.text)
            LINKS_SBSTC.append(route.get_attribute("href"))  # Assuming it's an <a> tag

        # Wait for the pagination element to be present
        pagination = wait.until(EC.presence_of_element_located((By.XPATH,
'//*[@id="root"]/div/div[4]/div[2]/div[1]/a')))

        # Construct the next button XPath dynamically
        next_button_xpath = f'//*[@id="root"]/div/div[4]/div[12]/div[1]//a[text()="{i+1}"]'

        if len(driver.find_elements(By.XPATH, next_button_xpath)) > 0:
            next_button = driver.find_element(By.XPATH, next_button_xpath)

            # Scroll into view and click the next button using JavaScript
            driver.execute_script("arguments[0].scrollIntoView(true);", next_button)
            time.sleep(2)  # Ensure some time for scrolling

            # Explicitly wait for the next button to be clickable
            wait.until(EC.element_to_be_clickable((By.XPATH, next_button_xpath)))
            driver.execute_script("arguments[0].click();", next_button)  # Click using
JavaScript

            i += 1  # Move to the next pagination step
        else:
            print(f"No more pages to paginate at step {i}")
            break  # No more pages, exit the loop

    except (NoSuchElementException, ElementClickInterceptedException) as e:
        print(f"Encountered an issue at step {i}: {e}")

    return LINKS_SBSTC, ROUTE_SBSTC
```

```
# Example route XPath for SBSTC buses (you may need to inspect the page for the exact route element)
route_xpath_sbstc = "//a[@class='route']"

# Calling the function to get links and routes
LINKS_SBSTC, ROUTE_SBSTC = sbstc_link_route(route_xpath_sbstc)
df_sbstc =pd.DataFrame({"Route_name":ROUTE_SBSTC, "Route_link":LINKS_SBSTC})
df_sbstc
df=df_sbstc
from selenium.webdriver.common.by import By
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC
from selenium.common.exceptions import NoSuchElementException, ElementClickInterceptedException
from selenium.webdriver.common.action_chains import ActionChains
from selenium.webdriver.common.keys import Keys  # Import Keys
import time
import pandas as pd
from selenium import webdriver

# Initialize the WebDriver
driver_k = webdriver.Chrome()

# Initialize lists for storing data
Bus_names_k = []
Bus_types_k = []
Start_Time_k = []
End_Time_k = []
Ratings_k = []
Total_Duration_k = []
```

```
Prices_k = []
Seats_Available_k = []
Route_names = []
Route_links = []

# Loop through each link in your dataframe
for i, r in df.iterrows():
    link = r["Route_link"]
    routes = r["Route_name"]

    # Open the route link
    driver_k.get(link)
    time.sleep(2)

    # Click on elements to reveal bus details
    elements = driver_k.find_elements(By.XPATH, f"//a[contains(@href, '{link}')]")
    for element in elements:
        element.click()
        time.sleep(2)

    # Click to view bus details
    try:
        clicks = driver_k.find_element(By.XPATH, "//div[@class='button']")
        clicks.click()
    except:
        continue
    time.sleep(2)

    # Scroll the page using ActionChains and Keys.PAGE_DOWN
    scrolling = True
```

```
    while scrolling:
        old_page_source = driver_k.page_source

        # Use ActionChains to perform a PAGE_DOWN
        ActionChains(driver_k).send_keys(Keys.PAGE_DOWN).perform()

        time.sleep(5)

        new_page_source = driver_k.page_source

        if new_page_source == old_page_source:
            scrolling = False

    # Extract bus details
    bus_name = driver_k.find_elements(By.XPATH, "//div[@class='travels lh-24 f-bold d-color']")
    bus_type = driver_k.find_elements(By.XPATH, "//div[@class='bus-type f-12 m-top-16 l-color evBus']")
    start_time = driver_k.find_elements(By.XPATH, "//*[@class='dp-time f-19 d-color f-bold']")
    end_time = driver_k.find_elements(By.XPATH, "//*[@class='bp-time f-19 d-color disp-Inline']")
    total_duration = driver_k.find_elements(By.XPATH, "//*[@class='dur l-color lh-24']")
    try:
        rating = driver_k.find_elements(By.XPATH, "//div[@class='clearfix row-one']/div[@class='column-six p-right-10 w-10 fl']")
    except:
        continue
    price = driver_k.find_elements(By.XPATH, '//*[@class="fare d-block"]')
    seats = driver_k.find_elements(By.XPATH, "//div[contains(@class, 'seat-left')]")

    # Append data to respective lists
```

```
    for bus in bus_name:
        Bus_names_k.append(bus.text)
        Route_links.append(link)
        Route_names.append(routes)
    for bus_type_elem in bus_type:
        Bus_types_k.append(bus_type_elem.text)
    for start_time_elem in start_time:
        Start_Time_k.append(start_time_elem.text)
    for end_time_elem in end_time:
        End_Time_k.append(end_time_elem.text)
    for total_duration_elem in total_duration:
        Total_Duration_k.append(total_duration_elem.text)
    for ratings in rating:
        Ratings_k.append(ratings.text)
    for price_elem in price:
        Prices_k.append(price_elem.text)
    for seats_elem in seats:
        Seats_Available_k.append(seats_elem.text)

print("Successfully Completed")

# from list to convert data frame
data = {
    'Bus_name': Bus_names_k,
    'Bus_type': Bus_types_k,
    'Start_time': Start_Time_k,
    'End_time': End_Time_k,
    'Total_duration': Total_Duration_k,
    'Price': Prices_k,
    "Seats_Available":Seats_Available_k,
```

```
    "Ratings":Ratings_k,
    'Route_link': Route_links,
    'Route_name': Route_names
}

df1 = pd.DataFrame(data)
#convert dataframe to csv

df1
df1.to_csv("C:/Users/Shalini/Downloads/sbstc_data6.csv") #done7




from selenium.webdriver.common.by import By
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC
from selenium.common.exceptions import NoSuchElementException,
ElementClickInterceptedException
import time
from selenium import webdriver

driver = webdriver.Chrome()

# Load the UPSRTC webpage
driver.get("https://www.redbus.in/online-booking/upsrtc")  # UPSRTC Redbus page URL

time.sleep(3)
driver.maximize_window()

def upsrtc_link_route(route_xpath):
    LINKS_UPSRTC = []  # List to store UPSRTC bus links
```

```
ROUTE_UPSRTC = []  # List to store UPSRTC route names

# Wait setup
wait = WebDriverWait(driver, 10)

try:
    i = 0
    while True:
        # Wait for the route elements to be present
        paths = wait.until(EC.presence_of_all_elements_located((By.XPATH, route_xpath)))

        # Retrieve names of the routes and their corresponding links
        for route in paths:
            ROUTE_UPSRTC.append(route.text)
            LINKS_UPSRTC.append(route.get_attribute("href"))  # Assuming it's an <a> tag

        # Wait for the pagination element to be present (update the XPath if needed)
        pagination = wait.until(EC.presence_of_element_located((By.XPATH,
'//*[@id="root"]/div/div[4]/div[12]')))

        # Construct the next button XPath dynamically (adjust according to actual button
structure)
        next_button_xpath = f'//*[@id="root"]/div/div[4]/div[12]/div[1]//a[text()="{i+1}"]'

        if len(driver.find_elements(By.XPATH, next_button_xpath)) > 0:
            next_button = driver.find_element(By.XPATH, next_button_xpath)

            # Scroll into view and click the next button using JavaScript
            driver.execute_script("arguments[0].scrollIntoView(true);", next_button)
            time.sleep(2)  # Ensure some time for scrolling
```

```
            # Explicitly wait for the next button to be clickable
            wait.until(EC.element_to_be_clickable((By.XPATH, next_button_xpath)))
            driver.execute_script("arguments[0].click();", next_button)  # Click using
JavaScript

            i += 1  # Move to the next pagination step
        else:
            print(f"No more pages to paginate at step {i}")
            break  # No more pages, exit the loop

    except (NoSuchElementException, ElementClickInterceptedException) as e:
        print(f"Encountered an issue at step {i}: {e}")

    return LINKS_UPSRTC, ROUTE_UPSRTC

# Example route XPath for UPSRTC buses (you may need to inspect the page for the exact
route element)
route_xpath = "//a[@class='route']"  # Adjust according to UPSRTC route elements on the
Redbus page

# Calling the function to get links and routes
LINKS_UPSRTC, ROUTE_UPSRTC = upsrtc_link_route(route_xpath)



df_upsrtc = pd.DataFrame({"Route_name": ROUTE_UPSRTC,"Route_link":
LINKS_UPSRTC})

df_upsrtc
df=df_upsrtc
#retrive the bus details
driver_k = webdriver.Chrome()
Bus_names_k = []
```

```
Bus_types_k = []
Start_Time_k = []
End_Time_k = []
Ratings_k = []
Total_Duration_k = []
Prices_k = []
Seats_Available_k = []
Route_names = []
Route_links = []

for i,r in df.iterrows():
    link=r["Route_link"]
    routes=r["Route_name"]

# Loop through each link
    driver_k.get(link)
    time.sleep(2)

    # Click on elements to reveal bus details
    elements = driver_k.find_elements(By.XPATH, f"//a[contains(@href, '{link}')]")
    for element in elements:
        element.click()
        time.sleep(2)

    # click elements to views bus
    try:
        clicks = driver_k.find_element(By.XPATH, "//div[@class='button']")
        clicks.click()
    except:
        continue
```

```
    time.sleep(2)


    scrolling = True
    while scrolling:
        old_page_source = driver_k.page_source


        # Use ActionChains to perform a PAGE_DOWN
        ActionChains(driver_k).send_keys(Keys.PAGE_DOWN).perform()


        time.sleep(5)


        new_page_source = driver_k.page_source


        if new_page_source == old_page_source:
            scrolling = False


    # Extract bus details
    bus_name = driver_k.find_elements(By.XPATH, "//div[@class='travels lh-24 f-bold d-color']")
    bus_type = driver_k.find_elements(By.XPATH, "//div[@class='bus-type f-12 m-top-16 l-color evBus']")
    start_time = driver_k.find_elements(By.XPATH, "//*[@class='dp-time f-19 d-color f-bold']")
    end_time = driver_k.find_elements(By.XPATH, "//*[@class='bp-time f-19 d-color disp-Inline']")
    total_duration = driver_k.find_elements(By.XPATH, "//*[@class='dur l-color lh-24']")
    try:
        rating = driver_k.find_elements(By.XPATH,"//div[@class='clearfix row-one']/div[@class='column-six p-right-10 w-10 fl']")
    except:
        continue
    price = driver_k.find_elements(By.XPATH, '//*[@class="fare d-block"]')
```

```
    seats = driver_k.find_elements(By.XPATH, "//div[contains(@class, 'seat-left')]")

    # Append data to respective lists
    for bus in bus_name:
        Bus_names_k.append(bus.text)
        Route_links.append(link)
        Route_names.append(routes)
    for bus_type_elem in bus_type:
        Bus_types_k.append(bus_type_elem.text)
    for start_time_elem in start_time:
        Start_Time_k.append(start_time_elem.text)
    for end_time_elem in end_time:
        End_Time_k.append(end_time_elem.text)
    for total_duration_elem in total_duration:
        Total_Duration_k.append(total_duration_elem.text)
    for ratings in rating:
        Ratings_k.append(ratings.text)
    for price_elem in price:
        Prices_k.append(price_elem.text)
    for seats_elem in seats:
        Seats_Available_k.append(seats_elem.text)

print("Successfully Completed")
# from list to convert data frame
data = {
    'Bus_name': Bus_names_k,
    'Bus_type': Bus_types_k,
    'Start_time': Start_Time_k,
    'End_time': End_Time_k,
    'Total_duration': Total_Duration_k,
```

```
    'Price': Prices_k,
    "Seats_Available":Seats_Available_k,
    "Ratings":Ratings_k,
    'Route_link': Route_links,
    'Route_name': Route_names
}

df1 = pd.DataFrame(data)
#convert dataframe to csv

df1
df1.to_csv("C:/Users/Shalini/Downloads/upsrtc_data.csv") #done8




from selenium.webdriver.common.by import By
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC
from selenium.common.exceptions import NoSuchElementException,
ElementClickInterceptedException
import time
from selenium import webdriver

driver = webdriver.Chrome()

# Load the PEPSU webpage
driver.get("https://www.redbus.in/online-booking/pepsu")  # PEPSU Redbus page URL

time.sleep(3)
driver.maximize_window()
```

```
def pepsu_link_route(route_xpath):
    LINKS_PEPSU = []  # List to store PEPSU bus links
    ROUTE_PEPSU = []  # List to store PEPSU route names

    # Wait setup
    wait = WebDriverWait(driver, 10)

    try:
        i = 0
        while True:
            # Wait for the route elements to be present
            paths = wait.until(EC.presence_of_all_elements_located((By.XPATH, route_xpath)))

            # Retrieve names of the routes and their corresponding links
            for route in paths:
                ROUTE_PEPSU.append(route.text)
                LINKS_PEPSU.append(route.get_attribute("href"))  # Assuming it's an <a> tag

            # Wait for the pagination element to be present (update the XPath if needed)
            pagination = wait.until(EC.presence_of_element_located((By.XPATH,
'//*[@id="root"]/div/div[4]/div[12]')))

            # Construct the next button XPath dynamically (adjust according to actual button
structure)
            next_button_xpath = f'//*[@id="root"]/div/div[4]/div[12]/div[1]//a[text()="{i+1}"]'

            if len(driver.find_elements(By.XPATH, next_button_xpath)) > 0:
                next_button = driver.find_element(By.XPATH, next_button_xpath)

                # Scroll into view and click the next button using JavaScript
                driver.execute_script("arguments[0].scrollIntoView(true);", next_button)
```

```
            time.sleep(2)  # Ensure some time for scrolling

            # Explicitly wait for the next button to be clickable
            wait.until(EC.element_to_be_clickable((By.XPATH, next_button_xpath)))
            driver.execute_script("arguments[0].click();", next_button)  # Click using
JavaScript

            i += 1  # Move to the next pagination step
        else:
            print(f"No more pages to paginate at step {i}")
            break  # No more pages, exit the loop

    except (NoSuchElementException, ElementClickInterceptedException) as e:
        print(f"Encountered an issue at step {i}: {e}")

    return LINKS_PEPSU, ROUTE_PEPSU

# Example route XPath for PEPSU buses (you may need to inspect the page for the exact
route element)
route_xpath = "//a[@class='route']"  # Adjust according to PEPSU route elements on the
Redbus page

# Calling the function to get links and routes
LINKS_PEPSU, ROUTE_PEPSU = pepsu_link_route(route_xpath)
df_pepsu = pd.DataFrame({"Route_name": ROUTE_PEPSU,"Route_link":
LINKS_PEPSU})
df_pepsu
df=df_pepsu
#retrive the bus details
driver_k = webdriver.Chrome()
Bus_names_k = []
Bus_types_k = []
```

```
Start_Time_k = []
End_Time_k = []
Ratings_k = []
Total_Duration_k = []
Prices_k = []
Seats_Available_k = []
Route_names = []
Route_links = []

for i,r in df.iterrows():
    link=r["Route_link"]
    routes=r["Route_name"]

# Loop through each link
    driver_k.get(link)
    time.sleep(2)

    # Click on elements to reveal bus details
    elements = driver_k.find_elements(By.XPATH, f"//a[contains(@href, '{link}')]")
    for element in elements:
        element.click()
        time.sleep(2)

    # click elements to views bus
    try:
        clicks = driver_k.find_element(By.XPATH, "//div[@class='button']")
        clicks.click()
    except:
        continue
    time.sleep(2)
```

```
    scrolling = True
    while scrolling:
        old_page_source = driver_k.page_source

        # Use ActionChains to perform a PAGE_DOWN
        ActionChains(driver_k).send_keys(Keys.PAGE_DOWN).perform()

        time.sleep(5)

        new_page_source = driver_k.page_source

        if new_page_source == old_page_source:
            scrolling = False

    # Extract bus details
    bus_name = driver_k.find_elements(By.XPATH, "//div[@class='travels lh-24 f-bold d-color']")
    bus_type = driver_k.find_elements(By.XPATH, "//div[@class='bus-type f-12 m-top-16 l-color evBus']")
    start_time = driver_k.find_elements(By.XPATH, "//*[@class='dp-time f-19 d-color f-bold']")
    end_time = driver_k.find_elements(By.XPATH, "//*[@class='bp-time f-19 d-color disp-Inline']")
    total_duration = driver_k.find_elements(By.XPATH, "//*[@class='dur l-color lh-24']")
    try:
        rating = driver_k.find_elements(By.XPATH,"//div[@class='clearfix row-one']/div[@class='column-six p-right-10 w-10 fl']")
    except:
        continue
    price = driver_k.find_elements(By.XPATH, '//*[@class="fare d-block"]')
    seats = driver_k.find_elements(By.XPATH, "//div[contains(@class, 'seat-left')]")
```

```
    # Append data to respective lists
    for bus in bus_name:
        Bus_names_k.append(bus.text)
        Route_links.append(link)
        Route_names.append(routes)
    for bus_type_elem in bus_type:
        Bus_types_k.append(bus_type_elem.text)
    for start_time_elem in start_time:
        Start_Time_k.append(start_time_elem.text)
    for end_time_elem in end_time:
        End_Time_k.append(end_time_elem.text)
    for total_duration_elem in total_duration:
        Total_Duration_k.append(total_duration_elem.text)
    for ratings in rating:
        Ratings_k.append(ratings.text)
    for price_elem in price:
        Prices_k.append(price_elem.text)
    for seats_elem in seats:
        Seats_Available_k.append(seats_elem.text)

print("Successfully Completed")
# from list to convert data frame
data = {
    'Bus_name': Bus_names_k,
    'Bus_type': Bus_types_k,
    'Start_time': Start_Time_k,
    'End_time': End_Time_k,
    'Total_duration': Total_Duration_k,
    'Price': Prices_k,
```

```
    "Seats_Available":Seats_Available_k,
    "Ratings":Ratings_k,
    'Route_link': Route_links,
    'Route_name': Route_names
}

df1 = pd.DataFrame(data)
#convert dataframe to csv

df1
df1.to_csv("C:/Users/Shalini/Downloads/pepsu_data.csv") #done9




from selenium.webdriver.common.by import By
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC
from selenium.common.exceptions import NoSuchElementException,
ElementClickInterceptedException
import time
from selenium import webdriver

driver = webdriver.Chrome()

# Load the WBTC webpage
driver.get("https://www.redbus.in/online-booking/wbtc-ctc/?utm_source=rtchometile")  #
WBTC Redbus page URL

time.sleep(3)
driver.maximize_window()
```

```
def wbtc_link_route(route_xpath):
    LINKS_WBTC = []  # List to store WBTC bus links
    ROUTE_WBTC = []  # List to store WBTC route names

    # Wait setup
    wait = WebDriverWait(driver, 10)

    try:
        i = 0
        while True:
            # Wait for the route elements to be present
            paths = wait.until(EC.presence_of_all_elements_located((By.XPATH, route_xpath)))

            # Retrieve names of the routes and their corresponding links
            for route in paths:
                ROUTE_WBTC.append(route.text)
                LINKS_WBTC.append(route.get_attribute("href"))  # Assuming it's an <a> tag

            # Wait for the pagination element to be present (update the XPath if needed)
            pagination = wait.until(EC.presence_of_element_located((By.XPATH,
'//*[@id="root"]/div/div[4]/div[12]')))

            # Construct the next button XPath dynamically (adjust according to actual button
structure)
            next_button_xpath = f'//*[@id="root"]/div/div[4]/div[12]/div[1]//a[text()="{i+1}"]'

            if len(driver.find_elements(By.XPATH, next_button_xpath)) > 0:
                next_button = driver.find_element(By.XPATH, next_button_xpath)

                # Scroll into view and click the next button using JavaScript
                driver.execute_script("arguments[0].scrollIntoView(true);", next_button)
```

```
            time.sleep(2)  # Ensure some time for scrolling

            # Explicitly wait for the next button to be clickable
            wait.until(EC.element_to_be_clickable((By.XPATH, next_button_xpath)))
            driver.execute_script("arguments[0].click();", next_button)  # Click using
JavaScript

            i += 1  # Move to the next pagination step
        else:
            print(f"No more pages to paginate at step {i}")
            break  # No more pages, exit the loop

    except (NoSuchElementException, ElementClickInterceptedException) as e:
        print(f"Encountered an issue at step {i}: {e}")

    return LINKS_WBTC, ROUTE_WBTC

# Example route XPath for WBTC buses (you may need to inspect the page for the exact
route element)
route_xpath = "//a[@class='route']"

# Calling the function to get links and routes
LINKS_WBTC, ROUTE_WBTC = wbtc_link_route(route_xpath)

df_wbtc = pd.DataFrame({"Route_name": ROUTE_WBTC,"Route_link": LINKS_WBTC})
df_wbtc
df=df_wbtc
#retrive the bus details
driver_k = webdriver.Chrome()
Bus_names_k = []
Bus_types_k = []
```

```
Start_Time_k = []
End_Time_k = []
Ratings_k = []
Total_Duration_k = []
Prices_k = []
Seats_Available_k = []
Route_names = []
Route_links = []

for i,r in df.iterrows():
    link=r["Route_link"]
    routes=r["Route_name"]

# Loop through each link
    driver_k.get(link)
    time.sleep(2)

    # Click on elements to reveal bus details
    elements = driver_k.find_elements(By.XPATH, f"//a[contains(@href, '{link}')]")
    for element in elements:
        element.click()
        time.sleep(2)

    # click elements to views bus
    try:
        clicks = driver_k.find_element(By.XPATH, "//div[@class='button']")
        clicks.click()
    except:
        continue
    time.sleep(2)
```

```
    scrolling = True
    while scrolling:
        old_page_source = driver_k.page_source

        # Use ActionChains to perform a PAGE_DOWN
        ActionChains(driver_k).send_keys(Keys.PAGE_DOWN).perform()

        time.sleep(5)

        new_page_source = driver_k.page_source

        if new_page_source == old_page_source:
            scrolling = False

    # Extract bus details
    bus_name = driver_k.find_elements(By.XPATH, "//div[@class='travels lh-24 f-bold d-color']")
    bus_type = driver_k.find_elements(By.XPATH, "//div[@class='bus-type f-12 m-top-16 l-color evBus']")
    start_time = driver_k.find_elements(By.XPATH, "//*[@class='dp-time f-19 d-color f-bold']")
    end_time = driver_k.find_elements(By.XPATH, "//*[@class='bp-time f-19 d-color disp-Inline']")
    total_duration = driver_k.find_elements(By.XPATH, "//*[@class='dur l-color lh-24']")
    try:
        rating = driver_k.find_elements(By.XPATH,"//div[@class='clearfix row-one']/div[@class='column-six p-right-10 w-10 fl']")
    except:
        continue
    price = driver_k.find_elements(By.XPATH, '//*[@class="fare d-block"]')
    seats = driver_k.find_elements(By.XPATH, "//div[contains(@class, 'seat-left')]")
```

```
    # Append data to respective lists
    for bus in bus_name:
        Bus_names_k.append(bus.text)
        Route_links.append(link)
        Route_names.append(routes)
    for bus_type_elem in bus_type:
        Bus_types_k.append(bus_type_elem.text)
    for start_time_elem in start_time:
        Start_Time_k.append(start_time_elem.text)
    for end_time_elem in end_time:
        End_Time_k.append(end_time_elem.text)
    for total_duration_elem in total_duration:
        Total_Duration_k.append(total_duration_elem.text)
    for ratings in rating:
        Ratings_k.append(ratings.text)
    for price_elem in price:
        Prices_k.append(price_elem.text)
    for seats_elem in seats:
        Seats_Available_k.append(seats_elem.text)

print("Successfully Completed")
# from list to convert data frame
data = {
    'Bus_name': Bus_names_k,
    'Bus_type': Bus_types_k,
    'Start_time': Start_Time_k,
    'End_time': End_Time_k,
    'Total_duration': Total_Duration_k,
    'Price': Prices_k,
```

```
    "Seats_Available":Seats_Available_k,
    "Ratings":Ratings_k,
    'Route_link': Route_links,
    'Route_name': Route_names
}

df1 = pd.DataFrame(data)
#convert dataframe to csv

df1
df1.to_csv("C:/Users/Shalini/Downloads/wbtc_data.csv") #done10




from selenium.webdriver.common.by import By
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC
from selenium.common.exceptions import NoSuchElementException,
ElementClickInterceptedException
import time
from selenium import webdriver

driver = webdriver.Chrome()

# Load the BSRTC webpage
driver.get("https://www.redbus.in/online-booking/bihar-state-road-transport-corporation-
bsrtc/?utm_source=rtchometile")  # BSRTC Redbus page URL

time.sleep(3)
driver.maximize_window()
```

```
def bsrtc_link_route(route_xpath):
    LINKS_BSRTC = []  # List to store BSRTC bus links
    ROUTE_BSRTC = []  # List to store BSRTC route names

    # Wait setup
    wait = WebDriverWait(driver, 10)

    try:
        i = 0
        while True:
            # Wait for the route elements to be present
            paths = wait.until(EC.presence_of_all_elements_located((By.XPATH, route_xpath)))

            # Retrieve names of the routes and their corresponding links
            for route in paths:
                ROUTE_BSRTC.append(route.text)
                LINKS_BSRTC.append(route.get_attribute("href"))  # Assuming it's an <a> tag

            # Wait for the pagination element to be present (update the XPath if needed)
            pagination = wait.until(EC.presence_of_element_located((By.XPATH,
'//*[@id="root"]/div/div[4]/div[12]')))

            # Construct the next button XPath dynamically (adjust according to actual button
structure)
            next_button_xpath = f'//*[@id="root"]/div/div[4]/div[12]/div[1]//a[text()="{i+1}"]'

            if len(driver.find_elements(By.XPATH, next_button_xpath)) > 0:
                next_button = driver.find_element(By.XPATH, next_button_xpath)

                # Scroll into view and click the next button using JavaScript
                driver.execute_script("arguments[0].scrollIntoView(true);", next_button)
```

```
            time.sleep(2)  # Ensure some time for scrolling

            # Explicitly wait for the next button to be clickable
            wait.until(EC.element_to_be_clickable((By.XPATH, next_button_xpath)))
            driver.execute_script("arguments[0].click();", next_button)  # Click using
JavaScript

            i += 1  # Move to the next pagination step
        else:
            print(f"No more pages to paginate at step {i}")
            break  # No more pages, exit the loop

    except (NoSuchElementException, ElementClickInterceptedException) as e:
        print(f"Encountered an issue at step {i}: {e}")

    return LINKS_BSRTC, ROUTE_BSRTC

# Example route XPath for BSRTC buses (you may need to inspect the page for the exact
route element)
route_xpath = "//a[@class='route']"  # Adjust according to BSRTC route elements on the
Redbus page

# Calling the function to get links and routes
LINKS_BSRTC, ROUTE_BSRTC = bsrtc_link_route(route_xpath)



df_bsrtc = pd.DataFrame({"Route_name": ROUTE_BSRTC,"Route_link": LINKS_BSRTC})

df_bsrtc
df=df_bsrtc
#retrive the bus details
```

```
driver_k = webdriver.Chrome()
Bus_names_k = []
Bus_types_k = []
Start_Time_k = []
End_Time_k = []
Ratings_k = []
Total_Duration_k = []
Prices_k = []
Seats_Available_k = []
Route_names = []
Route_links = []

for i,r in df.iterrows():
    link=r["Route_link"]
    routes=r["Route_name"]

# Loop through each link
    driver_k.get(link)
    time.sleep(2)

    # Click on elements to reveal bus details
    elements = driver_k.find_elements(By.XPATH, f"//a[contains(@href, '{link}')]")
    for element in elements:
        element.click()
        time.sleep(2)

    # click elements to views bus
    try:
        clicks = driver_k.find_element(By.XPATH, "//div[@class='button']")
        clicks.click()
```

```
    except:
        continue
    time.sleep(2)

    scrolling = True
    while scrolling:
        old_page_source = driver_k.page_source

        # Use ActionChains to perform a PAGE_DOWN
        ActionChains(driver_k).send_keys(Keys.PAGE_DOWN).perform()

        time.sleep(5)

        new_page_source = driver_k.page_source

        if new_page_source == old_page_source:
            scrolling = False

    # Extract bus details
    bus_name = driver_k.find_elements(By.XPATH, "//div[@class='travels lh-24 f-bold d-color']")
    bus_type = driver_k.find_elements(By.XPATH, "//div[@class='bus-type f-12 m-top-16 l-color evBus']")
    start_time = driver_k.find_elements(By.XPATH, "//*[@class='dp-time f-19 d-color f-bold']")
    end_time = driver_k.find_elements(By.XPATH, "//*[@class='bp-time f-19 d-color disp-Inline']")
    total_duration = driver_k.find_elements(By.XPATH, "//*[@class='dur l-color lh-24']")
    try:
        rating = driver_k.find_elements(By.XPATH,"//div[@class='clearfix row-one']/div[@class='column-six p-right-10 w-10 fl']")
    except:
```

```
            continue
        price = driver_k.find_elements(By.XPATH, '//*[@class="fare d-block"]')
        seats = driver_k.find_elements(By.XPATH, "//div[contains(@class, 'seat-left')]")

        # Append data to respective lists
        for bus in bus_name:
            Bus_names_k.append(bus.text)
            Route_links.append(link)
            Route_names.append(routes)
        for bus_type_elem in bus_type:
            Bus_types_k.append(bus_type_elem.text)
        for start_time_elem in start_time:
            Start_Time_k.append(start_time_elem.text)
        for end_time_elem in end_time:
            End_Time_k.append(end_time_elem.text)
        for total_duration_elem in total_duration:
            Total_Duration_k.append(total_duration_elem.text)
        for ratings in rating:
            Ratings_k.append(ratings.text)
        for price_elem in price:
            Prices_k.append(price_elem.text)
        for seats_elem in seats:
            Seats_Available_k.append(seats_elem.text)

    print("Successfully Completed")
    # from list to convert data frame
    data = {
        'Bus_name': Bus_names_k,
        'Bus_type': Bus_types_k,
        'Start_time': Start_Time_k,
```

```
    'End_time': End_Time_k,
    'Total_duration': Total_Duration_k,
    'Price': Prices_k,
    "Seats_Available":Seats_Available_k,
    "Ratings":Ratings_k,
    'Route_link': Route_links,
    'Route_name': Route_names
}

df1 = pd.DataFrame(data)
#convert dataframe to csv

df1
df1.to_csv("C:/Users/Shalini/Downloads/bsrtc_data01.csv") #done11




from selenium.webdriver.common.by import By
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC
from selenium.common.exceptions import NoSuchElementException,
ElementClickInterceptedException
import time
from selenium import webdriver

driver = webdriver.Chrome()

# Load the ASTC webpage
driver.get("https://www.redbus.in/online-booking/astc")  # ASTC Redbus page URL

time.sleep(3)
```

```
driver.maximize_window()


def astc_link_route(route_xpath):
    LINKS_ASTC = []  # List to store ASTC bus links
    ROUTE_ASTC = []  # List to store ASTC route names

    # Wait setup
    wait = WebDriverWait(driver, 10)

    try:
        i = 0
        while True:
            # Wait for the route elements to be present
            paths = wait.until(EC.presence_of_all_elements_located((By.XPATH, route_xpath)))

            # Retrieve names of the routes and their corresponding links
            for route in paths:
                ROUTE_ASTC.append(route.text)
                LINKS_ASTC.append(route.get_attribute("href"))  # Assuming it's an <a> tag

            # Wait for the pagination element to be present (update the XPath if needed)
            pagination = wait.until(EC.presence_of_element_located((By.XPATH,
'//*[@id="root"]/div/div[4]/div[12]')))

            # Construct the next button XPath dynamically (adjust according to actual button
structure)
            next_button_xpath = f'//*[@id="root"]/div/div[4]/div[12]/div[1]//a[text()="{i+1}"]'

            if len(driver.find_elements(By.XPATH, next_button_xpath)) > 0:
                next_button = driver.find_element(By.XPATH, next_button_xpath)
```

```
            # Scroll into view and click the next button using JavaScript
            driver.execute_script("arguments[0].scrollIntoView(true);", next_button)
            time.sleep(2)  # Ensure some time for scrolling

            # Explicitly wait for the next button to be clickable
            wait.until(EC.element_to_be_clickable((By.XPATH, next_button_xpath)))
            driver.execute_script("arguments[0].click();", next_button)  # Click using
JavaScript

            i += 1  # Move to the next pagination step
        else:
            print(f"No more pages to paginate at step {i}")
            break  # No more pages, exit the loop

    except (NoSuchElementException, ElementClickInterceptedException) as e:
        print(f"Encountered an issue at step {i}: {e}")

    return LINKS_ASTC, ROUTE_ASTC

# Example route XPath for ASTC buses (you may need to inspect the page for the exact route
element)
route_xpath = "//a[@class='route']"  # Adjust according to ASTC route elements on the
Redbus page

# Calling the function to get links and routes
LINKS_ASTC, ROUTE_ASTC = astc_link_route(route_xpath)



# Create a DataFrame for ASTC
df_astc = pd.DataFrame({"Route_name": ROUTE_ASTC,"Route_link": LINKS_ASTC})
# Display the DataFrame
```

```
df_astc


df = df_astc
#retrive the bus details
driver_k = webdriver.Chrome()
Bus_names_k = []
Bus_types_k = []
Start_Time_k = []
End_Time_k = []
Ratings_k = []
Total_Duration_k = []
Prices_k = []
Seats_Available_k = []
Route_names = []
Route_links = []

for i,r in df.iterrows():
    link=r["Route_link"]
    routes=r["Route_name"]

# Loop through each link
    driver_k.get(link)
    time.sleep(2)

    # Click on elements to reveal bus details
    elements = driver_k.find_elements(By.XPATH, f"//a[contains(@href, '{link}')]")
    for element in elements:
        element.click()
        time.sleep(2)
```

```
    # click elements to views bus
    try:
        clicks = driver_k.find_element(By.XPATH, "//div[@class='button']")
        clicks.click()
    except:
        continue
    time.sleep(2)

    scrolling = True
    while scrolling:
        old_page_source = driver_k.page_source

        # Use ActionChains to perform a PAGE_DOWN
        ActionChains(driver_k).send_keys(Keys.PAGE_DOWN).perform()

        time.sleep(5)

        new_page_source = driver_k.page_source

        if new_page_source == old_page_source:
            scrolling = False

    # Extract bus details
    bus_name = driver_k.find_elements(By.XPATH, "//div[@class='travels lh-24 f-bold d-color']")
    bus_type = driver_k.find_elements(By.XPATH, "//div[@class='bus-type f-12 m-top-16 l-color evBus']")
    start_time = driver_k.find_elements(By.XPATH, "//*[@class='dp-time f-19 d-color f-bold']")
    end_time = driver_k.find_elements(By.XPATH, "//*[@class='bp-time f-19 d-color disp-Inline']")
    total_duration = driver_k.find_elements(By.XPATH, "//*[@class='dur l-color lh-24']")
```

```
    try:
        rating = driver_k.find_elements(By.XPATH,"//div[@class='clearfix row-
one']/div[@class='column-six p-right-10 w-10 fl']")
    except:
        continue
    price = driver_k.find_elements(By.XPATH, '//*[@class="fare d-block"]')
    seats = driver_k.find_elements(By.XPATH, "//div[contains(@class, 'seat-left')]")

    # Append data to respective lists
    for bus in bus_name:
        Bus_names_k.append(bus.text)
        Route_links.append(link)
        Route_names.append(routes)
    for bus_type_elem in bus_type:
        Bus_types_k.append(bus_type_elem.text)
    for start_time_elem in start_time:
        Start_Time_k.append(start_time_elem.text)
    for end_time_elem in end_time:
        End_Time_k.append(end_time_elem.text)
    for total_duration_elem in total_duration:
        Total_Duration_k.append(total_duration_elem.text)
    for ratings in rating:
        Ratings_k.append(ratings.text)
    for price_elem in price:
        Prices_k.append(price_elem.text)
    for seats_elem in seats:
        Seats_Available_k.append(seats_elem.text)

print("Successfully Completed")
# from list to convert data frame
data = {
```

```
    'Bus_name': Bus_names_k,
    'Bus_type': Bus_types_k,
    'Start_time': Start_Time_k,
    'End_time': End_Time_k,
    'Total_duration': Total_Duration_k,
    'Price': Prices_k,
    "Seats_Available":Seats_Available_k,
    "Ratings":Ratings_k,
    'Route_link': Route_links,
    'Route_name': Route_names
}

df1 = pd.DataFrame(data)
#convert dataframe to csv

df1
df1.to_csv("C:/Users/Shalini/Downloads/astc_data.csv") #done12
```

4.2 MYSQL

```
import pandas as pd

import pymysql

myconnection = pymysql.connect(host='127.0.0.1',user='root',passwd='Diya')

myconnection.cursor().execute("create database mysqlapp")

pd.read_csv("redbusdata12.csv")

df=pd.read_csv("redbusdata12.csv")

df.columns

for i,j in  zip(df.columns,df.dtypes):

   print(i,j)

", ".join(f"{i} {j}"

for i,j in zip(df.columns, df.dtypes)).replace("float64", "FLOAT").replace("category",
"TEXT").replace("int64", "INTEGER").replace("object", "TEXT")


columns  = ", ".join(f"{i} {j}"

for i,j in zip(df.columns, df.dtypes)).replace("float64", "FLOAT").replace("category",
"TEXT").replace("int64", "INTEGER").replace("object", "TEXT")


myconnection.cursor().execute(f"create table mysqlapp.redbusdata12({columns})")

df

df.iloc[0]

for i in range (len(df)):

   print(i)

for i in range(len(df)):

   print(tuple(df.iloc[i]))

df = df.where(pd.notnull(df), None)  # Replace NaN with None


for i in range(len(df)):

   placeholders = ', '.join(['%s'] * len(df.columns))

   query = f"INSERT INTO mysqlapp.redbusdata12 VALUES ({placeholders})"
```

```
# Convert row to tuple and insert into the database
myconnection.cursor().execute(query, tuple(df.iloc[i]))
myconnection.commit()
```

4.3 STREAMLIT

```
import streamlit as slt
slt.set_page_config(layout="wide")  # This must be the first Streamlit command

import pandas as pd
from streamlit_option_menu import option_menu
from datetime import datetime

# Load bus route lists from CSV files
def load_route_lists():
df_AP = pd.read_csv(r"C:/streamlit_day2/myappr/AP_data.csv")
df_TS = pd.read_csv(r"C:/streamlit_day2/myappr/tsrtc_data2.csv")
df_KL = pd.read_csv(r"C:/streamlit_day2/myappr/ksrtc_data3.csv")
df_RS = pd.read_csv(r"C:/streamlit_day2/myappr/rsrtc_data4.csv")
df_HR = pd.read_csv(r"C:/streamlit_day2/myappr/hrtc_data5.csv")
df_CTU = pd.read_csv(r"C:/streamlit_day2/myappr/ctu_data6.csv")
df_SB = pd.read_csv(r"C:/streamlit_day2/myappr/sbstc_data6.csv")
df_UP = pd.read_csv(r"C:/streamlit_day2/myappr/upsrtc_data.csv")
df_PE = pd.read_csv(r"C:/streamlit_day2/myappr/pepsu_data.csv")
df_WB = pd.read_csv(r"C:/streamlit_day2/myappr/wbtc_data.csv")
df_BS = pd.read_csv(r"C:/streamlit_day2/myappr/bsrtc_data01.csv")
df_AS = pd.read_csv(r"C:/streamlit_day2/myappr/astc_data.csv")

lists_AP = df_AP["Route_name"].tolist()
lists_TS = df_TS["Route_name"].tolist()
lists_KL = df_KL["Route_name"].tolist()
lists_RS = df_RS["Route_name"].tolist()
lists_HR = df_HR["Route_name"].tolist()
lists_CTU = df_CTU["Route_name"].tolist()
lists_SB = df_SB["Route_name"].tolist()
```

```
lists_UP = df_UP["Route_name"].tolist()
lists_PE = df_PE["Route_name"].tolist()
lists_WB = df_WB["Route_name"].tolist()
lists_BS = df_BS["Route_name"].tolist()
lists_AS = df_AS["Route_name"].tolist()

return lists_AP, lists_TS, lists_KL, lists_RS, lists_HR, lists_CTU, lists_SB, lists_UP,
lists_PE, lists_WB, lists_BS, lists_AS

# Load route lists and data
lists_AP, lists_TS, lists_KL, lists_RS, lists_HR, lists_CTU, lists_SB, lists_UP, lists_PE,
lists_WB, lists_BS, lists_AS = load_route_lists()

# Define get_routes function outside of page-specific logic so it can be reused across pages
def get_routes(state):
if state == "Andhra Pradesh":
return lists_AP
elif state == "Telangana":
return lists_TS
elif state == "Kerala":
return lists_KL
elif state == "Rajasthan":
return lists_RS
elif state == "Himachal":
return lists_HR
elif state == "Chandigarh":
return lists_CTU
elif state == "South Bengal":
return lists_SB
elif state == "Uttar Pradesh":
return lists_UP
```

```
elif state == "Punjab":
return lists_PE
elif state == "West Bengal":
return lists_WB
elif state == "Bihar":
return lists_BS
elif state == "Assam":
return lists_AS

# Load CSV file into a DataFrame
def load_csv_data():
try:
return pd.read_csv(r"C:/streamlit_day2/myappr/redbusdata12.csv")
except Exception as e:
slt.error(f"Error loading data: {e}")
return pd.DataFrame()

df_redbus = load_csv_data()

# Convert Price column to numeric, coerce errors to NaN
df_redbus['Price'] = pd.to_numeric(df_redbus['Price'], errors='coerce')

# Remove rows with NaN values in Price
df_redbus = df_redbus.dropna(subset=['Price'])

# Navigation menu
web = option_menu(menu_title="Welcome to Redbus",
options=["Home", "About us", "Bus details", "Bus Booking", "Terms and Conditions",
"FAQ"],  # Add "FAQ"
icons=["house", "info-circle", "bus", "ticket", "file-text", "question-circle"],  # Added relevant
icon
```

```
orientation="horizontal")


# Home page

if web == "Home":

slt.image("C:/Users/Shalini/Downloads/WhatsApp Image 2024-09-26 at 8.40.12 PM.jpeg",
width=200)

slt.title("Redbus Data Scraping with Selenium & Dynamic Filtering using Streamlit")

slt.subheader(":blue[Domain:] Transportation")

slt.image("C:/Users/Shalini/Downloads/WhatsApp Image 2024-09-26 at 8.43.07 PM.jpeg",
width=400)

slt.subheader(":blue[Objective:]")

slt.markdown("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.")

slt.subheader(":blue[Overview:]")

slt.markdown("'Selenium: Selenium is a tool used for automating web browsers. It is
commonly used for web scraping, which involves extracting data from websites.

Pandas: Use the powerful Pandas library to transform the dataset from CSV format into a
structured dataframe.

MySQL: The extracted data is stored in MySQL for further querying and filtering.

Streamlit: Developed an interactive web application using Streamlit, a user-friendly
framework for data visualization and analysis.'")


# About us page

if web == "About us":

slt.title("About Us")

slt.subheader("Welcome to Redbus Data Scraping and Filtering Application")

slt.markdown(

"""

**Objective**:

Our primary goal is to revolutionize the transportation industry by providing a comprehensive
solution for collecting, analyzing, and visualizing bus travel data. This application allows
users to easily filter and find relevant bus travel information across various states in India.
```

What We Do:

- We scrape data from multiple state-run bus services to provide a centralized platform for bus route and fare information.

- Users can filter bus details based on criteria like bus type, fare range, and time of travel, making it easier to plan their journeys.

Technologies Used:

- **Selenium**: A powerful tool for web scraping, allowing us to automate the extraction of data from websites.

- **Pandas**: A robust library for data manipulation and analysis, enabling us to handle large datasets efficiently.

- **MySQL**: Used for storing the scraped data for further querying and analysis.

- **Streamlit**: A user-friendly framework that facilitates the creation of interactive web applications for data visualization.

Our Vision:

We aim to provide an intuitive and efficient platform for travelers to access bus travel information, thus enhancing their travel planning experience.

Developed By: Shalini Raju

"""

)

Bus details page

if web == "Bus details":

S = slt.selectbox("Lists of States", ["Andhra Pradesh", "Telangana", "Kerala", "Rajasthan", "Himachal", "Chandigarh",

"South Bengal", "Uttar Pradesh", "Punjab", "West Bengal", "Bihar", "Assam"])

col1, col2 = slt.columns(2)

with col1:

select_type = slt.radio("Choose bus type", ("sleeper", "semi-sleeper", "others"))

with col2:

```
select_fare = slt.radio("Choose bus fare range", ("50-1000", "1000-2000", "2000 and above"))

TIME = slt.time_input("Select the time")

# Route-specific filtering logic
route = slt.selectbox("List of routes", get_routes(S))

# Add a submit button
if slt.button("Submit"):
def query_bus_details(route_name, bus_type, fare_range, time_value):
# Define fare range
if fare_range == "50-1000":
fare_min, fare_max = 50, 1000
elif fare_range == "1000-2000":
fare_min, fare_max = 1000, 2000
else:
fare_min, fare_max = 2000, 100000  # High max for "2000 and above"

# Define bus type condition
bus_type_condition = {
"sleeper": "Sleeper",
"semi-sleeper": "Semi Sleeper",
"others": "Other"
}.get(bus_type, "")

# Convert selected time to comparable format
dummy_date = datetime.today().date()  # Use today's date
combined_time = datetime.combine(dummy_date, time_value)

# Filter CSV data based on user input
```

```
filtered_data = df_redbus[
(df_redbus['Price'].between(fare_min, fare_max)) &
(df_redbus['Route_name'] == route_name) &
(df_redbus['Bus_type'].str.contains(bus_type_condition, na=False)) &  # added na=False
(pd.to_datetime(df_redbus['Start_time'], errors='coerce') >= combined_time)
]

return filtered_data

# Query and display results
df_result = query_bus_details(route, select_type, select_fare, TIME)
slt.dataframe(df_result)



# Bus Booking page
if web == "Bus Booking":
slt.title("Bus Booking")
slt.subheader("Select your travel details:")

# Let the user select the state, route, date, and number of passengers
state = slt.selectbox("Choose State", ["Andhra Pradesh", "Telangana", "Kerala", "Rajasthan",
"Himachal", "Chandigarh", "South Bengal", "Uttar Pradesh", "Punjab", "West Bengal",
"Bihar", "Assam"])
route_booking = slt.selectbox("Choose Route", get_routes(state))
booking_date = slt.date_input("Select the travel date")
num_passengers = slt.number_input("Number of Passengers", min_value=1, max_value=10)

# Fetch the price for the selected route
selected_bus = df_redbus[df_redbus['Route_name'] == route_booking]
selected_bus_price = selected_bus['Price'].mean()
```

```
# Check if 'Rating' column exists and fetch the rating, else set it to None or a default value
if 'Rating' in selected_bus.columns:
selected_bus_rating = selected_bus['Rating'].mean()
else:
selected_bus_rating = None

# Allow users to add their own rating for the bus
user_rating = slt.slider("Rate this Bus", min_value=1, max_value=5, step=1)

# Calculate total price for all passengers
total_price = selected_bus_price * num_passengers

# Display booking summary along with bus rating
slt.subheader("Booking Summary")
slt.markdown(f"**State**: {state}")
slt.markdown(f"**Route**: {route_booking}")
slt.markdown(f"**Travel Date**: {booking_date}")
slt.markdown(f"**Number of Passengers**: {num_passengers}")
slt.markdown(f"**Ticket Price (per passenger)**: ₹{selected_bus_price:.2f}")
slt.markdown(f"**Total Price**: ₹{total_price:.2f}")

# Display bus rating if available
if selected_bus_rating is not None:
slt.markdown(f"**Average Bus Rating**: {selected_bus_rating:.1f} / 5.0")
else:
slt.markdown("**Average Bus Rating**: N/A")

# Display user-added rating
slt.markdown(f"**Your Rating for this Bus**: {user_rating} / 5.0")
```

```
if slt.button("Proceed to Payment"):
slt.success("Bus ticket booked successfully!")


# Terms and Conditions page
if web == "Terms and Conditions":
slt.title("Terms and Conditions")
slt.subheader("Please read the following terms and conditions carefully:")
slt.markdown(
"""
1. **Accuracy of Information**: The data provided in this application is collected from various bus operators and may not always be up-to-date. While we strive to ensure the accuracy of the information, we cannot guarantee that all details are correct at all times.

2. **Use of Application**: The application is intended for personal use only. Unauthorized commercial use of this application is strictly prohibited.

3. **Booking Responsibility**: The user is solely responsible for any bookings made through third-party payment gateways. We are not liable for any issues that arise during or after the booking process.

4. **Privacy Policy**: The data collected from users will not be shared with third parties without explicit consent. However, anonymized usage statistics may be collected for improving the service.

5. **Modification of Terms**: We reserve the right to modify these terms and conditions at any time without prior notice. It is the user's responsibility to stay updated with the latest version of the terms.

6. **Limitation of Liability**: We are not responsible for any losses or damages arising from the use of this application, including but not limited to direct, indirect, incidental, punitive, and consequential damages.
```

7. **Governing Law**: These terms and conditions are governed by and construed in accordance with the laws of India. Any disputes arising in connection with these terms shall be subject to the exclusive jurisdiction of the courts in India.

By using this application, you agree to these terms and conditions.

"""

)

FAQ page

if web == "FAQ":

slt.title("Frequently Asked Questions")

slt.subheader("Find answers to common questions below:")

slt.markdown(

"""

1. **What is Redbus Data Scraping with Selenium & Streamlit?**

This is a project that automates the extraction of bus route data from Redbus and provides a user-friendly interface for filtering and exploring the data.

2. **What states are supported?**

The application currently supports bus routes from Andhra Pradesh, Telangana, Kerala, Rajasthan, Himachal Pradesh, Chandigarh, South Bengal, Uttar Pradesh, Punjab, West Bengal, Bihar, and Assam.

3. **How is the data collected?**

Data is scraped from the Redbus website using Selenium, which automates the extraction of relevant bus details and routes.

4. **Can I book a bus through this app?**

No, this app does not facilitate direct bus bookings. It provides information that helps users find buses and plan their journeys, but booking must be done through other platforms.

5. **How can I filter bus data?**

You can filter data based on bus type, fare range, availability of seats, and travel timings using the app’s built-in filters.

6. **Is the data updated in real-time?**

No, the data is not updated in real-time. It is periodically refreshed through web scraping, so there may be some delay between updates.

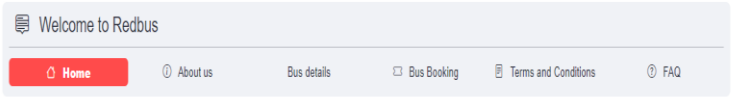
7. **Who developed this application?**

The application was developed by Shalini Raju as part of a project focusing on data scraping and visualization.

"""

)

5.OUTPUT SCREEANS



Redbus Data Scraping with Selenium & Dynamic Filtering using Streamlit

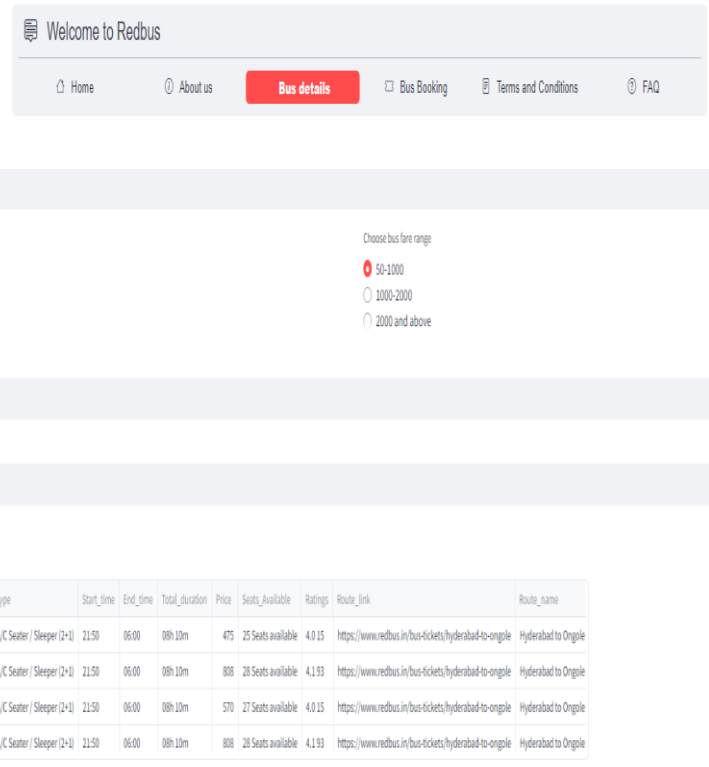
Domain: Transportation

Objective:

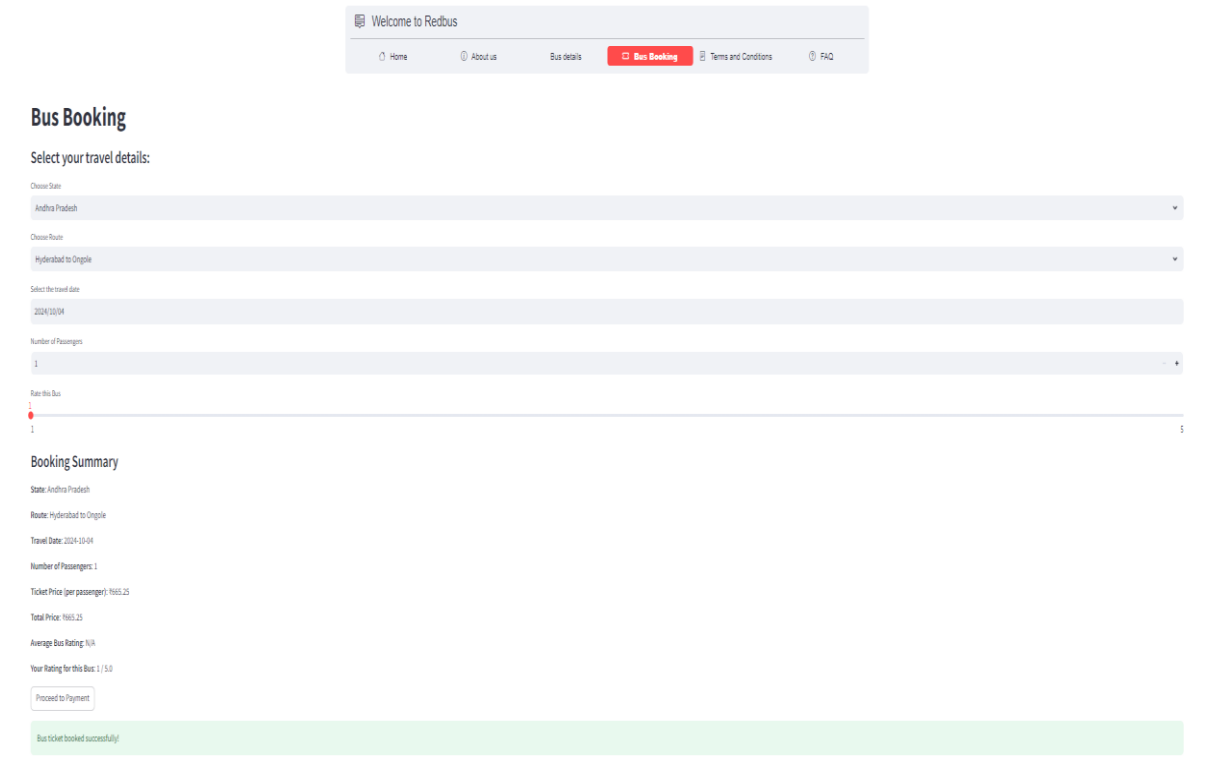
The 'Redbus Data Scraping and Filtering with Streamlit Application' aims to revolutionize the transportation industry by providing a comprehensive solution for collecting, analyzing, and visualizing bus travel data.

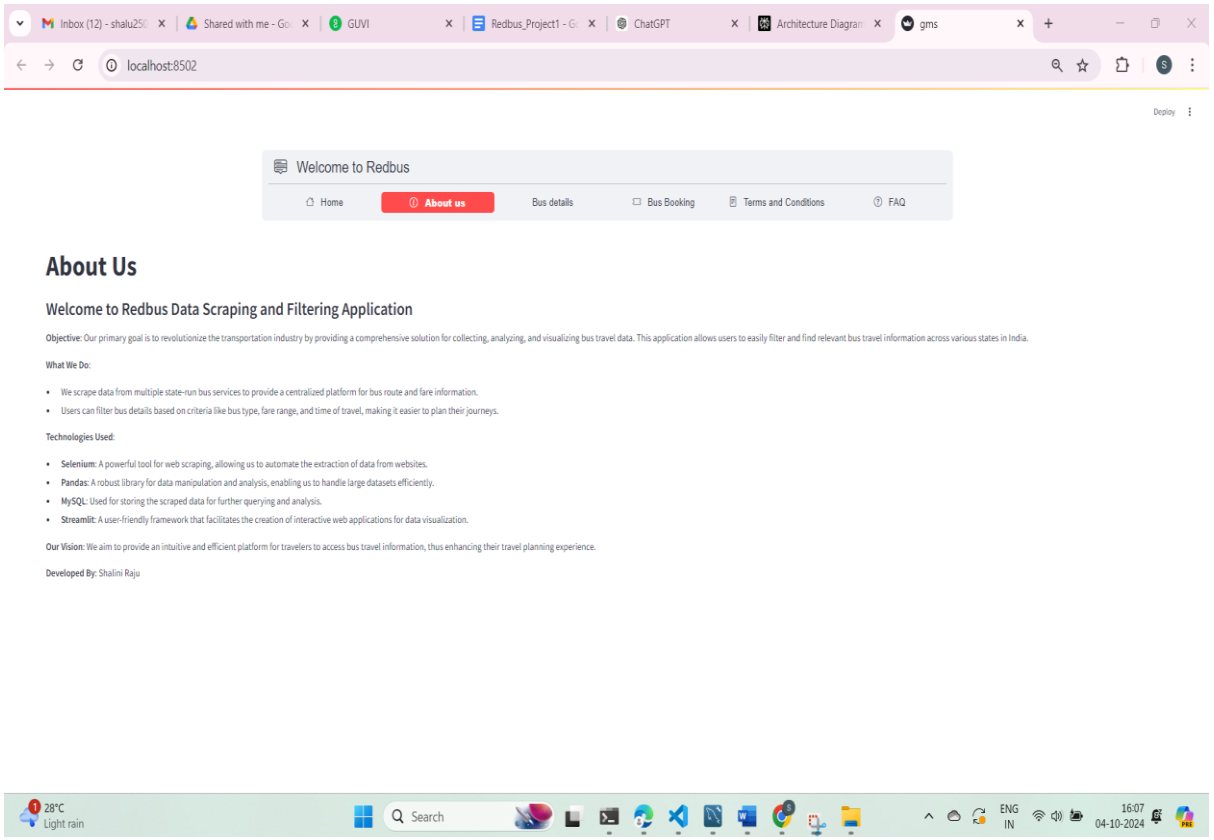
Overview:

Selenium: Selenium is a tool used for automating web browsers. It is commonly used for web scraping, which involves extracting data from websites. Pandas: Use the powerful Pandas library to transform the dataset from CSV format into a structured dataframe. MySQL: The extracted data is stored in MySQL for further querying and filtering. Streamlit: Developed an interactive web application using Streamlit, a user-friendly framework for data visualization and analysis.



	Sl_no	State_name	Bus_name	Bus_type	Start_time	End_time	Total_duration	Price	Seats_Available	Ratings	Route_link	Route_name
4	4	Andhra pradesh	Cherry tours&travels	Non A/C Seater / Sleeper (2+1)	21:50	06:00	08h 10m	475	25 Seats available	4.0 15	https://www.redbus.in/bus-tickets/hyderabad-to-ongole	Hyderabad to Ongole
8	8	Andhra pradesh	Cherry tours&travels	Non A/C Seater / Sleeper (2+1)	21:50	06:00	08h 10m	808	28 Seats available	4.1 93	https://www.redbus.in/bus-tickets/hyderabad-to-ongole	Hyderabad to Ongole
256	256	Telangana	Cherry tours&travels	Non A/C Seater / Sleeper (2+1)	21:50	06:00	08h 10m	570	27 Seats available	4.0 15	https://www.redbus.in/bus-tickets/hyderabad-to-ongole	Hyderabad to Ongole
260	260	Telangana	Cherry tours&travels	Non A/C Seater / Sleeper (2+1)	21:50	06:00	08h 10m	808	28 Seats available	4.1 93	https://www.redbus.in/bus-tickets/hyderabad-to-ongole	Hyderabad to Ongole


Welcome to Redbus
Home
About us
Bus details
Bus Booking
Terms and Conditions
FAQ
Bus Booking
Select your travel details:
Choose State
Andhra Pradesh
Choose Route
Hyderabad to Ongole
Select the travel date
2024/10/04
Number of Passengers
1
Rate this Bus
1
5
Booking Summary
State: Andhra Pradesh
Route: Hyderabad to Ongole
Travel Date: 2024-10-04
Number of Passengers: 1
Ticket Price (per passenger): ₹665.25
Total Price: ₹665.25
Average Bus Rating: N/A
Your Rating for this Bus: 1 / 5.0
Proceed to Payment
Bus ticket booked successfully!


localhost:8502
Deploy
Welcome to Redbus
Home
About us
Bus details
Bus Booking
Terms and Conditions
FAQ
About Us
Welcome to Redbus Data Scraping and Filtering Application
Objective: Our primary goal is to revolutionize the transportation industry by providing a comprehensive solution for collecting, analyzing, and visualizing bus travel data. This application allows users to easily filter and find relevant bus travel information across various states in India.
What We Do:
We scrape data from multiple state-run bus services to provide a centralized platform for bus route and fare information.
Users can filter bus details based on criteria like bus type, fare range, and time of travel, making it easier to plan their journeys.
Technologies Used:
Selenium: A powerful tool for web scraping, allowing us to automate the extraction of data from websites.
Pandas: A robust library for data manipulation and analysis, enabling us to handle large datasets efficiently.
MySQL: Used for storing the scraped data for further querying and analysis.
Streamlit: A user-friendly framework that facilitates the creation of interactive web applications for data visualization.
Our Vision: We aim to provide an intuitive and efficient platform for travelers to access bus travel information, thus enhancing their travel planning experience.
Developed By: Shalini Raju
28°C
Light rain
Search
ENG
IN
16:07
04-10-2024

6.FUTURE ENHANCEMENTS

FUTURE ENHANCEMENTS

5.1 Additional Data Sources

- Expand Data Sources: Include data from other transportation platforms, such as Goibibo, Makemytrip, and Yatra, to provide users with a more comprehensive view of available travel options.
- Multi-modal Transportation: Integrate additional modes of transport, like trains and flights, allowing users to compare various travel options within a single platform.

5.2 Advanced Filtering and Sorting Options

- More Filter Options: Add filters for bus amenities like Wi-Fi, charging points, and on-board entertainment, to give users a detailed view of available facilities.
- Dynamic Sorting: Allow users to sort buses by custom parameters, such as travel time, departure time, or star rating, to enhance the user experience.

5.3 Real-time Data Updates

- Live Seat Availability: Integrate real-time seat availability by directly fetching data from the Redbus API or using automated scraping at regular intervals.
- Price Alerts: Allow users to set price alerts and receive notifications when prices drop for their selected routes or times.

5.4 User Personalization and Recommendations

- User Profiles: Enable users to create accounts and save their preferences, which will help the system provide personalized recommendations.
- Travel History and Recommendations: Utilize users' previous travel data to recommend similar or preferred travel options, making the application more tailored to individual needs.

5.5 Mobile Application Development

- Mobile App: Develop a mobile version of the application, ensuring compatibility with both Android and iOS platforms. This would make the tool more accessible and convenient for users on the go.
- Push Notifications: Use push notifications to alert users of any changes to their bookings, special offers, or reminders.

5.6 Enhanced Data Visualization

- Interactive Charts and Maps: Implement interactive charts and route maps to visualize bus routes, availability, and pricing trends over time.
- Heatmaps: Show heatmaps of popular routes or peak travel times, helping users plan their trips accordingly.

5.7 Integration with Payment Gateways

- In-app Booking and Payment: Allow users to directly book tickets within the app, streamlining the process by integrating with payment gateways like Razorpay, PayPal, or UPI.
- Wallet Integration: Enable wallet functionality where users can store funds for quicker transactions and receive refunds or credits from cancellations.

5.8 Data Analytics and Reporting

- Advanced Analytics Dashboard: Build an analytics dashboard for transportation companies to view key metrics, such as seat occupancy rates, pricing trends, and route popularity.
- Report Generation: Enable users and businesses to generate custom reports on travel patterns, user preferences, and booking statistics for better decision-making.

5.9 Machine Learning and AI Integration

- Demand Prediction: Implement machine learning algorithms to forecast demand for bus routes based on historical data, aiding in dynamic pricing and resource allocation.
- Chatbot Support: Integrate a chatbot to provide users with instant answers to their queries, guide them through the booking process, and offer travel advice.

5.10 Localization and Multi-language Support

- Multi-language Support: Translate the application into multiple languages to cater to a broader audience across various regions.
- Localization: Customize features like currency, date formats, and language based on the user’s location for a more personalized experience.

7.CONCLUSION

CONCLUSION

The **"Redbus Data Scraping with Selenium & Dynamic Filtering using Streamlit"** project successfully demonstrates how data scraping and dynamic filtering techniques can be leveraged to provide valuable insights into transportation data. By automating the extraction of bus travel information from Redbus and presenting it through a user-friendly Streamlit application, this project addresses key challenges in accessing and analyzing transportation data for informed decision-making.

Through Selenium, we were able to effectively gather detailed information on various aspects such as bus routes, schedules, seat availability, and prices. This data is then stored in a structured SQL database, enabling seamless integration with the Streamlit application. The application allows users to filter data based on multiple parameters, including bus type, price range, seat availability, and ratings, providing an intuitive interface for users to explore and analyze data.

This project has several valuable applications within the transportation industry, such as aiding travel aggregators, conducting market analysis, and improving customer service. With accurate, real-time information, users can make informed travel decisions, while businesses can gain insights into customer preferences, route popularity, and competitive pricing.

The project also highlights the scalability of web scraping and data filtering tools in handling extensive and complex datasets, making it a powerful approach for organizations looking to enhance their data-driven strategies. Moreover, the future enhancements outlined—such as real-time data updates, advanced filtering options, and machine learning integration—open avenues for further development, which could transform this tool into a comprehensive solution for transportation data analysis.

In conclusion, the successful implementation of this project not only provides a functional solution to the problem statement but also lays a foundation for continued innovation in the field of transportation data analytics. This project has the potential to be expanded and refined to meet evolving industry needs, providing enhanced value to both end-users and transportation service providers.

8.BIBLIOGRAPHY

BIBLIOGRAPHY

1. **Books and Articles**

- Mitchell, Ryan. *Web Scraping with Python: Collecting Data from the Modern Web.* O'Reilly Media, 2015.
- McKinney, Wes. *Python for Data Analysis: Data Wrangling with Pandas, NumPy, and IPython.* O'Reilly Media, 2017.
- Zeller, John. *Learning Selenium Testing Tools - Third Edition.* Packt Publishing, 2015.

2. **Web Resources**

- Redbus India. *Bus Tickets Online - Book Bus Tickets Online.* Accessed September 2024. https://www.redbus.in
- Streamlit Documentation. *Streamlit - The Fastest Way to Build Data Apps.* Accessed September 2024. https://docs.streamlit.io
- Selenium Documentation. *Selenium WebDriver.* Accessed September 2024. https://www.selenium.dev/documentation/en/
- Pandas Documentation. *Pandas Documentation - Python Data Analysis Library.* Accessed September 2024. https://pandas.pydata.org/docs/
- SQLite Documentation. *SQLite Home Page.* Accessed September 2024. https://sqlite.org/docs.html

3. **Software and Tools**

- Python Software Foundation. *Python 3.* https://www.python.org
- Jupyter Notebook. *Project Jupyter.* https://jupyter.org