Redbus Data Scraping with Selenium & Dynamic Filter Using Streamlit

SUMMITED BY

-PRADEEP RAJA G

TABLE OF CONTENTS

1. CASE STUDY

1.1 ABSTRACT

The project Redbus Data Scraping with Selenium & Dynamic Filtering using Streamlit focuses on extracting bus route information from the Redbus website using automated browser interactions. The project aims to streamline the process of collecting real-time bus route data from multiple state transport corporations, process it, and provide an intuitive interface for dynamic filtering and search functionalities. The system is designed to gather detailed information such as bus names, routes, timings, and prices, and present it in an accessible and interactive manner using Streamlit.

1.2 INTRODUCTION

The increasing demand for real-time travel data presents opportunities for automation in data collection. Traditional manual methods of extracting information from websites are often time-consuming and inefficient. This project leverages Selenium, a powerful browser automation tool, and Streamlit, an easy-to-use framework for web applications, to dynamically scrape bus route information and present it to users in a structured format. The solution automates the process of navigating multiple web pages, extracting detailed bus information, and providing users with filtering options to narrow down their choices based on specific criteria.

1.3 OVERVIEW

The primary objective of this project is to build a data collection system that extracts and processes bus route information from the Redbus platform. It further aims to create an interactive interface where users can dynamically filter the collected data using various parameters like route names, timings, prices, and more. The combination of web scraping with dynamic filtering enhances user experience and efficiency in accessing relevant information.

Technical skills and **Technologies** used:

- ✓ Web Scraping , Python Programming
- ✓ Data Storage & Databases
- ✓ Web Application Development
- ✓ Data Filtering & Querying

Business use cases :

1. Travel Aggregators: Providing real-time bus schedules, seat availability, and pricing information for customers on travel booking platforms.
2. Market Analysis: Analyzing travel patterns and preferences to offer insights for market research and demand forecasting.
3. Customer Service Enhancement: Offering personalized travel options and recommendations based on customer preferences and past travel behavior.
4. Competitor Analysis: Comparing pricing, service levels, and routes with competitors to optimize service offerings and gain a competitive edge.

2. METHODOLOGY

2.1 Scraping with Selenium

Selenium is employed for web scraping due to its ability to interact with web elements dynamically loaded via JavaScript. The scraping is designed to cover multiple pages and states listed on the Redbus platform, capturing route details such as bus names, departure times, duration, and available seats.

2.2Streamlit for Visualization

Once the data is scraped and stored in CSV format, Streamlit is utilized to build a user interface for data visualization. Users can interact with the dataset, search for specific routes, and apply various filters, making the system highly user-friendly and efficient.

2.3Tools Used

- Selenium: To automate browser interactions for dynamic web scraping.
- Pandas: For data handling, transformation, and storage in CSV format.
- Streamlit: To build the user interface for data exploration and filtering.
- WebDriver: Chrome WebDriver is used to automate web browser interactions.

3. CODE STRUCTURE

The project code is organized into four scripts:

1. **Scraping1.py**: This script is responsible for scraping route links from specific states on the Redbus website. It navigates multiple pages, collects route names and corresponding URLs, and stores them in CSV files. It handles pagination, scrolling, and clicks through Selenium automation.

```
from selenium import webdriver

from selenium.webdriver.common.by import By

from selenium.common.exceptions import TimeoutException, NoSuchElementException, ElementClickInterceptedException

from selenium.webdriver.support.ui import WebDriverWait

from selenium.webdriver.support import expected_conditions as EC

import time

import pandas as pd

import os



# Open the browser

driver = webdriver.Chrome()



# Load the webpage

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

time.sleep(3)
```

```
# Maximize the browser window
driver.maximize_window()


# WebDriverWait for element waiting
wait = WebDriverWait(driver, 20)


# Scroll to an element before clicking it
def scroll_to_element(driver, element):
    driver.execute_script("arguments[0].scrollIntoView(true);", element)


# JavaScript click as a fallback method
def javascript_click(driver, element):
    driver.execute_script("arguments[0].click();", element)


# Function to retrieve bus links and route names
def Kerala_link_route(path):
    LINKS_KERALA = []
    ROUTE_KERALA = []
```

```
    for i in range(1, 3):  # Set pagination limit to 2 for demonstration
        paths = driver.find_elements(By.XPATH, path)

        for links in paths:
            d = links.get_attribute("href")
            if d:  # Check if link is not None
                LINKS_KERALA.append(d)

        for route in paths:
            ROUTE_KERALA.append(route.text)

        try:
            # Wait for the pagination element to be present
            pagination = wait.until(EC.presence_of_element_located((By.XPATH, '//*[@class="DC_117_paginationTable"]')))
            next_button = pagination.find_element(By.XPATH, f'//div[@class="DC_117_pageTabs " and text()="{i+1}"]')

            # Scroll to the next button
            scroll_to_element(driver, next_button)
            time.sleep(3)  # Give time for the page to load
```

```
            try:
                # Click the button with JavaScript as a fallback if it's intercepted
                next_button.click()
            except ElementClickInterceptedException:
                print("Element click intercepted. Trying JavaScript click.")
                javascript_click(driver, next_button)

        except NoSuchElementException:
            print(f"No more pages to paginate at step {i}")
            break
        except TimeoutException:
            print(f"Timeout while waiting for pagination at step {i}")
            break

    return LINKS_KERALA, ROUTE_KERALA

# Call the function to get bus route links and names
LINKS_KERALA, ROUTE_KERALA = Kerala_link_route("//a[@class='route']")
```

```
# Create a DataFrame to store the route names and links

df_k = pd.DataFrame({"Route_name": ROUTE_KERALA, "Route_link": LINKS_KERALA})



# Specify the path to save the CSV

csv_path = r"C:\Users\ADMIN\Documents\10 state\df_k.csv"



# Ensure the directory exists before saving the file

directory = os.path.dirname(csv_path)

if not os.path.exists(directory):

    os.makedirs(directory)



# Save the DataFrame as a CSV file

df_k.to_csv(csv_path, index=False)



# Print the DataFrame for confirmation

print(df_k)



# Close the browser

driver.quit()
```

```
# Open the browser

driver_A = webdriver.Chrome()



# Load the webpage

driver_A.get("https://www.redbus.in/online-booking/apsrtc/?utm_source=rtchometile")

time.sleep(3)



# Maximize the browser window

driver_A.maximize_window()



# WebDriverWait for element waiting

wait = WebDriverWait(driver_A, 20)



# Scroll to an element before clicking it

def scroll_to_element(driver, element):

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



# JavaScript click as a fallback method

def javascript_click(driver, element):

    driver.execute_script("arguments[0].click();", element)
```

```
# Function to retrieve bus links and route names
def Andhra_link_route(path):
    LINKS_ANDHRA = []
    ROUTE_ANDHRA = []

    # Retrieve route links and names across pagination
    for i in range(1, 5):  # Set pagination limit to 4
        paths = driver_A.find_elements(By.XPATH, path)

        for links in paths:
            d = links.get_attribute("href")
            if d:  # Check if link is not None
                LINKS_ANDHRA.append(d)

        for route in paths:
            ROUTE_ANDHRA.append(route.text)

        try:
            # Wait for the pagination element to be present
```

```
        pagination = wait.until(EC.presence_of_element_located((By.XPATH, '//*[@class="DC_117_paginationTable"]')))

        next_button = pagination.find_element(By.XPATH, f'//div[@class="DC_117_pageTabs " and text()="{i+1}"]')



        # Scroll to the next button

        scroll_to_element(driver_A, next_button)

        time.sleep(3)  # Give time for the page to load



        try:

            # Click the button with JavaScript as a fallback if it's intercepted

            next_button.click()

        except ElementClickInterceptedException:

            print("Element click intercepted. Trying JavaScript click.")

            javascript_click(driver_A, next_button)



    except NoSuchElementException:

        print(f"No more pages to paginate at step {i}")

        break

    except TimeoutException:

        print(f"Timeout while waiting for pagination at step {i}")
```

```
            break


    return LINKS_ANDHRA, ROUTE_ANDHRA


# Call the function to get bus route links and names

LINKS_ANDHRA, ROUTE_ANDHRA = Andhra_link_route("//a[@class='route']")


# Create a DataFrame to store the route names and links

df_A = pd.DataFrame({"Route_name": ROUTE_ANDHRA, "Route_link": LINKS_ANDHRA})


# Specify the path to save the CSV

path = r"C:\Users\ADMIN\Documents\10 state\df_andhra.csv"  # Use a different filename for Andhra routes


# Ensure the directory exists before saving the file

directory = os.path.dirname(path)

if not os.path.exists(directory):

    os.makedirs(directory)


# Save the DataFrame as a CSV file
```

```
df_A.to_csv(path, index=False)


# Print the DataFrame for confirmation

print(df_A)


# Close the browser

driver_A.quit()

# Open the browser

driver_T = webdriver.Chrome()


# Load the webpage

driver_T.get("https://www.redbus.in/online-booking/tsrtc/?utm_source=rtchometile")

time.sleep(3)


# Maximize the browser window

driver_T.maximize_window()


# WebDriverWait for element waiting

wait = WebDriverWait(driver_T, 20)
```

```
# Scroll to an element before clicking it
def scroll_to_element(driver, element):
    driver.execute_script("arguments[0].scrollIntoView(true);", element)


# JavaScript click as a fallback method
def javascript_click(driver, element):
    driver.execute_script("arguments[0].click();", element)


# Function to retrieve bus links and route names
def Telangana_link_route(path):
    LINKS_TELANGANA = []
    ROUTE_TELANGANA = []


    # Loop through pagination (set limit to 3 pages for this example)
    for i in range(1, 4):
        paths = driver_T.find_elements(By.XPATH, path)


        for links in paths:
            d = links.get_attribute("href")
            if d:  # Check if the link is valid
```

```
            LINKS_TELANGANA.append(d)


        for route in paths:
            ROUTE_TELANGANA.append(route.text)


        try:
            # Wait for the pagination element to be present
            pagination = wait.until(EC.presence_of_element_located((By.XPATH, '//*[@class="DC_117_paginationTable"]')))
            next_button = pagination.find_element(By.XPATH, f'//div[@class="DC_117_pageTabs " and text()="{i+1}"]')

            # Scroll to the next button
            scroll_to_element(driver_T, next_button)
            time.sleep(3)  # Give time for the page to load

            try:
                # Click the next button
                next_button.click()
            except ElementClickInterceptedException:
                print("Element click intercepted. Trying JavaScript click.")
```

```
            javascript_click(driver_T, next_button)


        except NoSuchElementException:
            print(f"No more pages to paginate at step {i}")
            break
        except TimeoutException:
            print(f"Timeout while waiting for pagination at step {i}")
            break


    return LINKS_TELANGANA, ROUTE_TELANGANA


# Call the function to get bus route links and names
LINKS_TELANGANA, ROUTE_TELANGANA = Telangana_link_route("//a[@class='route']")


# Create a DataFrame to store the route names and links
df_T = pd.DataFrame({"Route_name": ROUTE_TELANGANA, "Route_link": LINKS_TELANGANA})


# Specify the path to save the CSV (update the file name)
csv_path = r"C:\Users\ADMIN\Documents\10 state\df_telangana.csv"
```

```
# Ensure the directory exists before saving the file
directory = os.path.dirname(csv_path)
if not os.path.exists(directory):
    os.makedirs(directory)

# Save the DataFrame as a CSV file
df_T.to_csv(csv_path, index=False)

# Print the DataFrame for confirmation
print(df_T)

# Close the browser
driver_T.quit()
# Open the browser
driver_G = webdriver.Chrome()

# Load the webpage
driver_G.get("https://www.redbus.in/online-booking/ktcl/?utm_source=rtchometile")
time.sleep(3)
```

```
# Maximize the browser window
driver_G.maximize_window()


# WebDriverWait for element waiting
wait = WebDriverWait(driver_G, 20)


# Scroll to an element before clicking it
def scroll_to_element(driver, element):
    driver.execute_script("arguments[0].scrollIntoView(true);", element)


# JavaScript click as a fallback method
def javascript_click(driver, element):
    driver.execute_script("arguments[0].click();", element)


# Function to retrieve bus links and route names
def Kadamba_link_route(path):
    LINKS_KADAMBA = []
    ROUTE_KADAMBA = []
```

```
    # Loop through pagination (set limit to 3 pages for this example)
    for i in range(1, 4):
        paths = driver_G.find_elements(By.XPATH, path)

        for links in paths:
            d = links.get_attribute("href")
            if d:  # Check if the link is valid
                LINKS_KADAMBA.append(d)

        for route in paths:
            ROUTE_KADAMBA.append(route.text)

        try:
            # Wait for the pagination element to be present
            pagination = wait.until(EC.presence_of_element_located((By.XPATH, '//*[@class="DC_117_paginationTable"]')))
            next_button = pagination.find_element(By.XPATH, f'//div[@class="DC_117_pageTabs " and text()="{i+1}"]')

            # Scroll to the next button
            scroll_to_element(driver_G, next_button)
```

```
            time.sleep(3)  # Give time for the page to load


            try:
                # Click the next button
                next_button.click()
            except ElementClickInterceptedException:
                print("Element click intercepted. Trying JavaScript click.")
                javascript_click(driver_G, next_button)


        except NoSuchElementException:
            print(f"No more pages to paginate at step {i}")
            break


    return LINKS_KADAMBA, ROUTE_KADAMBA


# Call the function to get bus route links and names
LINKS_KADAMBA, ROUTE_KADAMBA = Kadamba_link_route("//a[@class='route']")


# Create a DataFrame to store the route names and links
df_G = pd.DataFrame({"Route_name": ROUTE_KADAMBA, "Route_link": LINKS_KADAMBA})
```

```
# Specify the path to save the CSV

csv_path = r"C:\Users\ADMIN\Documents\10 state\df_kadamba.csv"



# Ensure the directory exists before saving the file

directory = os.path.dirname(csv_path)

if not os.path.exists(directory):

    os.makedirs(directory)



# Save the DataFrame as a CSV file

df_G.to_csv(csv_path, index=False)



# Print the DataFrame for confirmation

print(df_G)



# Close the browser

driver_G.quit()

# Open the browser

driver_R = webdriver.Chrome()
```

```
# Load the webpage
driver_R.get("https://www.redbus.in/online-booking/rsrtc/?utm_source=rtchometile")
time.sleep(3)

# Maximize the browser window
driver_R.maximize_window()

# WebDriverWait for element waiting
wait = WebDriverWait(driver_R, 20)

# Scroll to an element before clicking it
def scroll_to_element(driver, element):
    driver.execute_script("arguments[0].scrollIntoView(true);", element)

# JavaScript click as a fallback method
def javascript_click(driver, element):
    driver.execute_script("arguments[0].click();", element)

# Function to retrieve bus links and route names
def Rajastan_link_route(path):
```

```
LINKS_RAJASTAN = []

ROUTE_RAJASTAN = []



# Loop through pagination (set limit to 3 pages for this example)

for i in range(1, 4):

    paths = driver_R.find_elements(By.XPATH, path)



    for links in paths:

        d = links.get_attribute("href")

        if d:  # Check if the link is valid

            LINKS_RAJASTAN.append(d)



    for route in paths:

        ROUTE_RAJASTAN.append(route.text)



    try:

        # Wait for the pagination element to be present

        pagination = wait.until(EC.presence_of_element_located((By.XPATH, '//*[@class="DC_117_paginationTable"]')))

        next_button = pagination.find_element(By.XPATH, f'//div[@class="DC_117_pageTabs " and text()="{i+1}"]')
```

```
            # Scroll to the next button
            scroll_to_element(driver_R, next_button)
            time.sleep(3)  # Give time for the page to load

            try:
                # Click the next button
                next_button.click()
            except ElementClickInterceptedException:
                print("Element click intercepted. Trying JavaScript click.")
                javascript_click(driver_R, next_button)

        except NoSuchElementException:
            print(f"No more pages to paginate at step {i}")
            break

    return LINKS_RAJASTAN, ROUTE_RAJASTAN

# Call the function to get bus route links and names
LINKS_RAJASTAN, ROUTE_RAJASTAN = Rajastan_link_route("//a[@class='route']")
```

```
# Create a DataFrame to store the route names and links

df_R = pd.DataFrame({"Route_name": ROUTE_RAJASTAN, "Route_link": LINKS_RAJASTAN})


# Specify the path to save the CSV (corrected for Rajasthan)

csv_path = r"C:\Users\ADMIN\Documents\10 state\df_rajasthan.csv"


# Ensure the directory exists before saving the file

directory = os.path.dirname(csv_path)

if not os.path.exists(directory):

    os.makedirs(directory)


# Save the DataFrame as a CSV file

df_R.to_csv(csv_path, index=False)


# Print the DataFrame for confirmation

print(df_R)


# Close the browser

driver_R.quit()
```

```
# Open the browser

driver_SB = webdriver.Chrome()



# Load the webpage

driver_SB.get("https://www.redbus.in/online-booking/south-bengal-state-transport-corporation-sbstc/?utm_source=rtchometile")

time.sleep(3)



# Maximize the browser window

driver_SB.maximize_window()



# WebDriverWait for element waiting

wait = WebDriverWait(driver_SB, 20)



# Scroll to an element before clicking it

def scroll_to_element(driver, element):

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



# JavaScript click as a fallback method

def javascript_click(driver, element):

    driver.execute_script("arguments[0].click();", element)
```

```
# Function to retrieve bus links and route names
def Southbengal_link_route(path):
    LINKS_SOUTHBENGAL = []
    ROUTE_SOUTHBENGAL = []

    # Loop through pagination (set limit to 5 pages for this example)
    for i in range(1, 6):
        paths = driver_SB.find_elements(By.XPATH, path)

        for links in paths:
            d = links.get_attribute("href")
            if d:  # Check if the link is valid
                LINKS_SOUTHBENGAL.append(d)

        for route in paths:
            ROUTE_SOUTHBENGAL.append(route.text)

        try:
            # Wait for the pagination element to be present
```

```
        pagination = wait.until(EC.presence_of_element_located((By.XPATH, '//*[@class="DC_117_paginationTable"]')))

        next_button = pagination.find_element(By.XPATH, f'//div[@class="DC_117_pageTabs " and text()="{i+1}"]')


        # Scroll to the next button

        scroll_to_element(driver_SB, next_button)

        time.sleep(3)  # Give time for the page to load


        try:

            # Click the next button

            next_button.click()

        except ElementClickInterceptedException:

            print("Element click intercepted. Trying JavaScript click.")

            javascript_click(driver_SB, next_button)


    except NoSuchElementException:

        print(f"No more pages to paginate at step {i}")

        break


return LINKS_SOUTHBENGAL, ROUTE_SOUTHBENGAL
```

```
# Call the function to get bus route links and names

LINKS_SOUTHBENGAL, ROUTE_SOUTHBENGAL = Southbengal_link_route("//a[@class='route']")


# Create a DataFrame to store the route names and links

df_SB = pd.DataFrame({"Route_name": ROUTE_SOUTHBENGAL, "Route_link": LINKS_SOUTHBENGAL})


# Specify the path to save the CSV (corrected for South Bengal)

csv_path = r"C:\Users\ADMIN\Documents\10 state\df_southbengal.csv"


# Ensure the directory exists before saving the file

directory = os.path.dirname(csv_path)

if not os.path.exists(directory):

    os.makedirs(directory)


# Save the DataFrame as a CSV file

df_SB.to_csv(csv_path, index=False)


# Print the DataFrame for confirmation
```

```
print(df_SB)


# Close the browser

driver_SB.quit()

# Open the browser

driver_H = webdriver.Chrome()


# Load the webpage

driver_H.get("https://www.redbus.in/online-booking/hrtc/?utm_source=rtchometile")

time.sleep(3)


# Maximize the browser window

driver_H.maximize_window()


# WebDriverWait for element waiting

wait = WebDriverWait(driver_H, 20)


# Scroll to an element before clicking it

def scroll_to_element(driver, element):

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

```
# JavaScript click as a fallback method
def javascript_click(driver, element):
    driver.execute_script("arguments[0].click();", element)


# Function to retrieve bus links and route names
def Haryana_link_route(path):
    LINKS_HARYANA = []
    ROUTE_HARYANA = []


    # Loop through pagination (set limit to 5 pages for this example)
    for i in range(1, 6):
        paths = driver_H.find_elements(By.XPATH, path)


        for links in paths:
            d = links.get_attribute("href")
            if d:  # Check if the link is valid
                LINKS_HARYANA.append(d)


        for route in paths:
```

```
            ROUTE_HARYANA.append(route.text)


        try:
            # Wait for the pagination element to be present
            pagination = wait.until(EC.presence_of_element_located((By.XPATH, '//*[@class="DC_117_paginationTable"]')))
            next_button = pagination.find_element(By.XPATH, f'//div[@class="DC_117_pageTabs " and text()="{i+1}"]')

            # Scroll to the next button
            scroll_to_element(driver_H, next_button)
            time.sleep(3)  # Give time for the page to load

            try:
                # Click the next button
                next_button.click()
            except ElementClickInterceptedException:
                print("Element click intercepted. Trying JavaScript click.")
                javascript_click(driver_H, next_button)

        except NoSuchElementException:
```

```
            print(f"No more pages to paginate at step {i}")
            break

    return LINKS_HARYANA, ROUTE_HARYANA

# Call the function to get bus route links and names
LINKS_HARYANA, ROUTE_HARYANA = Haryana_link_route("//a[@class='route']")

# Create a DataFrame to store the route names and links
df_H = pd.DataFrame({"Route_name": ROUTE_HARYANA, "Route_link": LINKS_HARYANA})

# Specify the path to save the CSV (corrected for Haryana)
csv_path = r"C:\Users\ADMIN\Documents\10 state\df_haryana.csv"

# Ensure the directory exists before saving the file
directory = os.path.dirname(csv_path)
if not os.path.exists(directory):
    os.makedirs(directory)

# Save the DataFrame as a CSV file
```

```
df_H.to_csv(csv_path, index=False)


# Print the DataFrame for confirmation

print(df_H)


# Close the browser

driver_H.quit()

# Open the browser

driver_AS = webdriver.Chrome()


# Load the webpage

driver_AS.get("https://www.redbus.in/online-booking/astc/?utm_source=rtchometile")

time.sleep(3)


# Maximize the browser window

driver_AS.maximize_window()


# WebDriverWait for element waiting

wait = WebDriverWait(driver_AS, 20)
```

```
# Scroll to an element before clicking it
def scroll_to_element(driver, element):
    driver.execute_script("arguments[0].scrollIntoView(true);", element)


# JavaScript click as a fallback method
def javascript_click(driver, element):
    driver.execute_script("arguments[0].click();", element)


# Function to retrieve bus links and route names for Assam
def Assam_link_route(path):
    LINKS_ASSAM = []
    ROUTE_ASSAM = []


    # Loop through pagination (set limit to 5 pages for this example)
    for i in range(1, 5):
        paths = driver_AS.find_elements(By.XPATH, path)


        for links in paths:
            d = links.get_attribute("href")
            if d:  # Check if the link is valid
```

```
            LINKS_ASSAM.append(d)


        for route in paths:
            ROUTE_ASSAM.append(route.text)


        try:
            # Wait for the pagination element to be present
            pagination = wait.until(EC.presence_of_element_located((By.XPATH, '//*[@class="DC_117_paginationTable"]')))
            next_button = pagination.find_element(By.XPATH, f'//div[@class="DC_117_pageTabs " and text()="{i+1}"]')

            # Scroll to the next button
            scroll_to_element(driver_AS, next_button)
            time.sleep(3)  # Give time for the page to load

            try:
                # Click the next button
                next_button.click()
            except ElementClickInterceptedException:
                print("Element click intercepted. Trying JavaScript click.")
```

```
                javascript_click(driver_AS, next_button)


        except NoSuchElementException:
            print(f"No more pages to paginate at step {i}")
            break


    return LINKS_ASSAM, ROUTE_ASSAM


# Call the function to get bus route links and names for Assam
LINKS_ASSAM, ROUTE_ASSAM = Assam_link_route("//a[@class='route']")


# Create a DataFrame to store the route names and links
df_AS = pd.DataFrame({"Route_name": ROUTE_ASSAM, "Route_link": LINKS_ASSAM})


# Specify the path to save the CSV (corrected for Assam)
csv_path = r"C:\Users\ADMIN\Documents\10 state\df_assam.csv"


# Ensure the directory exists before saving the file
directory = os.path.dirname(csv_path)
if not os.path.exists(directory):
```

```
    os.makedirs(directory)


# Save the DataFrame as a CSV file

df_AS.to_csv(csv_path, index=False)


# Print the DataFrame for confirmation

print(df_AS)


# Close the browser

driver_AS.quit()

# Open the browser

driver_UP = webdriver.Chrome()


# Load the webpage

driver_UP.get("https://www.redbus.in/online-booking/uttar-pradesh-state-road-transport-corporation-upsrtc/?utm_source=rtchometile")

time.sleep(3)


# Maximize the browser window

driver_UP.maximize_window()
```

```
# WebDriverWait for element waiting
wait = WebDriverWait(driver_UP, 20)


# Scroll to an element before clicking it
def scroll_to_element(driver, element):
    driver.execute_script("arguments[0].scrollIntoView(true);", element)


# JavaScript click as a fallback method
def javascript_click(driver, element):
    driver.execute_script("arguments[0].click();", element)


# Function to retrieve bus links and route names for Uttar Pradesh
def UP_link_route(path):
    LINKS_UP = []
    ROUTE_UP = []


    # Loop through pagination (set limit to 5 pages for this example)
    for i in range(1, 6):
        paths = driver_UP.find_elements(By.XPATH, path)
```

```
        # Retrieve the route links
        for links in paths:
            d = links.get_attribute("href")
            if d:  # Check if the link is valid
                LINKS_UP.append(d)

        # Retrieve the names of the routes
        for route in paths:
            ROUTE_UP.append(route.text)

        try:
            # Wait for the pagination element to be present
            pagination = wait.until(EC.presence_of_element_located((By.XPATH, '//*[@class="DC_117_paginationTable"]')))
            next_button = pagination.find_element(By.XPATH, f'//div[@class="DC_117_pageTabs " and text()="{i+1}"]')

            # Scroll to the next button
            scroll_to_element(driver_UP, next_button)
            time.sleep(3)  # Give time for the page to load
```

```
            try:
                # Click the next button
                next_button.click()
            except ElementClickInterceptedException:
                print("Element click intercepted. Trying JavaScript click.")
                javascript_click(driver_UP, next_button)

        except NoSuchElementException:
            print(f"No more pages to paginate at step {i}")
            break

    return LINKS_UP, ROUTE_UP

# Call the function to get bus route links and names for Uttar Pradesh
LINKS_UP, ROUTE_UP = UP_link_route("//a[@class='route']")

# Create a DataFrame to store the route names and links
df_UP = pd.DataFrame({"Route_name": ROUTE_UP, "Route_link": LINKS_UP})

# Specify the path to save the CSV (corrected for Uttar Pradesh)
```

```
csv_path = r"C:\Users\ADMIN\Documents\10 state\df_uttarpradesh.csv"


# Ensure the directory exists before saving the file

directory = os.path.dirname(csv_path)

if not os.path.exists(directory):

    os.makedirs(directory)


# Save the DataFrame as a CSV file

df_UP.to_csv(csv_path, index=False)


# Print the DataFrame for confirmation

print(df_UP)


# Close the browser

driver_UP.quit()

# Open the browser

driver_WB = webdriver.Chrome()


# Load the webpage

driver_WB.get("https://www.redbus.in/online-booking/wbtc-ctc/?utm_source=rtchometile")
```

```
time.sleep(3)


# Maximize the browser window

driver_WB.maximize_window()


# WebDriverWait for element waiting

wait = WebDriverWait(driver_WB, 20)


# Function to retrieve bus links and route names for West Bengal

def Westbengal_link_route(path):

    LINKS_WESTBENGAL = []

    ROUTE_WESTBENGAL = []


    # Loop through pagination (set limit to 5 pages for this example)

    for i in range(1, 6):

        paths = driver_WB.find_elements(By.XPATH, path)


        # Retrieve the route links

        for links in paths:

            d = links.get_attribute("href")
```

```
            if d:  # Check if the link is valid
                LINKS_WESTBENGAL.append(d)


        # Retrieve the names of the routes
        for route in paths:
            ROUTE_WESTBENGAL.append(route.text)


        try:
            # Wait for the pagination element to be present and scroll into view
            pagination = wait.until(EC.presence_of_element_located((By.XPATH, '//*[@class="DC_117_paginationTable"]')))
            next_button = pagination.find_element(By.XPATH, f'//div[@class="DC_117_pageTabs " and text()={i+1}]')

            # Scroll to the next button and wait until it's clickable
            driver_WB.execute_script("arguments[0].scrollIntoView();", next_button)
            time.sleep(2)  # Give it a bit of time to settle

            # Wait until the element is clickable
            wait.until(EC.element_to_be_clickable((By.XPATH, f'//div[@class="DC_117_pageTabs " and text()={i+1}]')))
```

```
            # Try clicking the next button, handle any intercepted exception
            try:
                next_button.click()
            except ElementClickInterceptedException:
                print(f"Click intercepted at page {i}, attempting scroll and retry...")
                driver_WB.execute_script("arguments[0].scrollIntoView();", next_button)
                time.sleep(2)
                next_button.click()

        except NoSuchElementException:
            print(f"No more pages to paginate at step {i}")
            break

    return LINKS_WESTBENGAL, ROUTE_WESTBENGAL

# Call the function to get bus route links and names for West Bengal
LINKS_WESTBENGAL, ROUTE_WESTBENGAL = Westbengal_link_route("//a[@class='route']")

# Create a DataFrame to store the route names and links
```

```
df_WB = pd.DataFrame({"Route_name": ROUTE_WESTBENGAL, "Route_link": LINKS_WESTBENGAL})


# Save the DataFrame as a CSV file for West Bengal

path_WB = r"C:\Users\ADMIN\Documents\10 state\df_westbengal.csv"

df_WB.to_csv(path_WB, index=False)


# Assuming df_k, df_A, df_T, df_G, df_R, df_H, df_SB, df_AS, df_UP are pre-loaded DataFrames for each state

# Concatenate all the bus route DataFrames into one

df_final = pd.concat([df_k, df_A, df_T, df_G, df_R, df_H, df_SB, df_AS, df_UP, df_WB], ignore_index=True)


# Save the final concatenated DataFrame as a CSV

final_path = r"C:\Users\ADMIN\Documents\10 state\df_all_routes.csv"

df_final.to_csv(final_path, index=False)

# Print the final DataFrame for confirmation

print(df_final)

# Close the browser

driver_WB.quit()
```

2. **Scraping2.py**: This script builds on the work of Scraping10.py by visiting each route link and collecting more granular details such as bus names, start and end times, total travel duration, prices, and ratings. It processes dynamic data elements and compiles the information in a structured CSV file format, ready for further analysis.

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

# State links
state_links = [
    "https://www.redbus.in/online-booking/ksrtc-kerala/?utm_source=rtchometile",
    "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/ktcl/?utm_source=rtchometile",
    "https://www.redbus.in/online-booking/rsrtc/?utm_source=rtchometile",
    "https://www.redbus.in/online-booking/south-bengal-state-transport-corporation-sbstc/?utm_source=rtchometile",
    "https://www.redbus.in/online-booking/hrtc/?utm_source=rtchometile",
    "https://www.redbus.in/online-booking/astc/?utm_source=rtchometile",
    "https://www.redbus.in/online-booking/uttar-pradesh-state-road-transport-corporation-upsrtc/?utm_source=rtchometile",
    "https://www.redbus.in/online-booking/wbtc-ctc/?utm_source=rtchometile"
]

# Initialize the WebDriver
driver = webdriver.Chrome()
wait = WebDriverWait(driver, 20)
```

```python
# Helper function to scroll to an element and click it
def scroll_to_and_click(driver, element):
    driver.execute_script("arguments[0].scrollIntoView(true);", element)
    time.sleep(2)  # Ensure some time for scrolling
    driver.execute_script("arguments[0].click();", element)

# Function to retrieve bus routes for a state
def get_state_bus_routes(state_link, path="//a[@class='route']"):
    driver.get(state_link)
    time.sleep(3)
    driver.maximize_window()

    links = []
    routes = []

    for i in range(1, 4):  # Adjust the range based on the number of pages
        paths = driver.find_elements(By.XPATH, path)

        # Collect route links and names
        for link in paths:
            href = link.get_attribute("href")
            route_name = link.text.strip()  # Ensure full route name is collected
            if href and route_name:  # Ensure both name and link are not empty
                links.append(href)
                routes.append(route_name)
```

```
def get_state_bus_routes(state_link, path="//a[@class='route']"):
        try:
            # Check if next page exists and navigate
            pagination = wait.until(EC.presence_of_element_located((By.XPATH, '//*[@class="DC_117_paginationTable"]')))
            next_button_xpath = f'//div[@class="DC_117_pageTabs " and text()="{i+1}"]'
            next_button = driver.find_element(By.XPATH, next_button_xpath)
            scroll_to_and_click(driver, next_button)

        except NoSuchElementException:
            print(f"No more pages for state: {state_link}")
            break

    # Return DataFrame with full route names and links
    return pd.DataFrame({"Route_name": routes, "Route_link": links})

# Initialize final DataFrame to collect all state data
all_states_data = pd.DataFrame()

# Iterate through each state and collect data
for state_link in state_links:
    df_state = get_state_bus_routes(state_link)
    all_states_data = pd.concat([all_states_data, df_state], ignore_index=True)

# Check if file path is accessible before saving
output_path_routes = r"C:\Users\ADMIN\Documents\red bus projects\all_routes_pradeep.csv"  # Change path if needed
if os.path.exists(output_path_routes):
    if not os.access(output_path_routes, os.W_OK):
        print(f"File at {output_path_routes} is not writable. Please check permissions.")
```

```
else:
    print(f"Saving routes data to {output_path_routes}")
    all_states_data.to_csv(output_path_routes, index=False, encoding='utf-8-sig')

# Function to collect detailed bus information from each route link
def get_bus_details(df):
    Bus_names = []
    Bus_types = []
    Start_Time = []
    End_Time = []
    Total_Duration = []
    Prices = []
    Seats_Available = []
    Ratings = []
    Route_links = []
    Route_names = []

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

        driver.get(link)
        time.sleep(2)
```

```
    try:
        bus_name_elements = driver.find_elements(By.XPATH, "//div[@class='travels lh-24 f-bold d-color']")
        bus_type_elements = driver.find_elements(By.XPATH, "//div[@class='bus-type f-12 m-top-16 l-color evBus']")
        start_time_elements = driver.find_elements(By.XPATH, "//*[@class='dp-time f-19 d-color f-bold']")
        end_time_elements = driver.find_elements(By.XPATH, "//*[@class='bp-time f-19 d-color disp-Inline']")
        total_duration_elements = driver.find_elements(By.XPATH, "//*[@class='dur l-color lh-24']")
        rating_elements = driver.find_elements(By.XPATH, "//div[@class='clearfix row-one']/div[@class='column-six p-right-10 w-10 fl']")
        price_elements = driver.find_elements(By.XPATH, '//*[@class="fare d-block"]')
        seats_elements = driver.find_elements(By.XPATH, "//div[contains(@class, 'seat-left')]")

        for bus in bus_name_elements:
            Bus_names.append(bus.text)
            Route_links.append(link)
            Route_names.append(route)

        for bus_type in bus_type_elements:
            Bus_types.append(bus_type.text)

        for start_time in start_time_elements:
            Start_Time.append(start_time.text)

        for end_time in end_time_elements:
            End_Time.append(end_time.text)

        for duration in total_duration_elements:
            Total_Duration.append(duration.text)

        for rating in rating_elements:
            Ratings.append(rating.text)
```

Activate Windo
Go to Settings to

```
def get_bus_details(df):
                for price in price_elements:
                    Prices.append(price.text)

                for seat in seats_elements:
                    Seats_Available.append(seat.text)

            except NoSuchElementException:
                print(f"Failed to extract data for route: {route}")
                continue

        # Create DataFrame from collected bus details
        bus_data = pd.DataFrame({
            "Bus_name": Bus_names,
            "Bus_type": Bus_types,
            "Start_time": Start_Time,
            "End_time": End_Time,
            "Total_duration": Total_Duration,
            "Price": Prices,
            "Seats_Available": Seats_Available,
            "Ratings": Ratings,
            "Route_link": Route_links,
            "Route_name": Route_names
        })

        return bus_data

    # Get detailed bus information from the routes
    bus_details_df = get_bus_details(all_states_data)
```

```python
# Check if file path is accessible before saving bus details
output_path_bus_details = r"C:\Users\ADMIN\Documents\red bus projects_pradeep.csv"  # Change path if needed
if os.path.exists(output_path_bus_details):
    if not os.access(output_path_bus_details, os.W_OK):
        print(f"File at {output_path_bus_details} is not writable. Please check permissions.")
else:
    print(f"Saving bus details to {output_path_bus_details}")
    bus_details_df.to_csv(output_path_bus_details, index=False, encoding='utf-8-sig')

# Close the browser
driver.quit()
print("Data collection completed and saved.")
```

3. **Database and Table Creation for Redbus Travel Project:** The database `redbus_travel` is designed to store bus travel information, including bus details, routes, and availability. The primary table, `project_info`, holds data such as bus name, type, timings, price, available seats, ratings, and route links. Additionally, individual tables for each state (e.g., `kerala_routes`, `andhra_routes`) store route-specific details, ensuring efficient data organization for different regions. Each route table contains fields for route names and links, enabling easy retrieval of travel information per state.

```sql
-- Step 1: Create the Database
CREATE DATABASE redbus_travel;

-- Step 2: Use the created database
USE redbus_travel;

-- Step 3: Create the 'project_info' table for bus details
CREATE TABLE project_info (
    id INT AUTO_INCREMENT PRIMARY KEY,  -- Make 'id' auto-incrementing for easier insertion
    Bus_name VARCHAR(50),
    Bus_type VARCHAR(50),
    Start_time TIME,
    End_time TIME,
    Total_duration VARCHAR(50),
    Price DECIMAL(10, 2),
    Seats_Available INT,
    Ratings FLOAT,
    Route_link VARCHAR(500),  -- Typo correction: "Root_link" should be "Route_link"
    Route_name VARCHAR(50)
);

-- Step 4: Create the route tables for each state

```

```sql

CREATE TABLE kerala_routes (
    Route_name VARCHAR(255),
    Route_link VARCHAR(500)
);

CREATE TABLE andhra_routes (
    Route_name VARCHAR(255),
    Route_link VARCHAR(500)
);

CREATE TABLE telangana_routes (
    Route_name VARCHAR(255),
    Route_link VARCHAR(500)
);

CREATE TABLE kadamba_routes (
    Route_name VARCHAR(255),
    Route_link VARCHAR(500)
);

CREATE TABLE rajasthan_routes (
    Route_name VARCHAR(255),
    Route_name VARCHAR(255),
    Route_link VARCHAR(500)
);

CREATE TABLE southbengal_routes (
    Route_name VARCHAR(255),
    Route_link VARCHAR(500)
);

CREATE TABLE haryana_routes (
    Route_name VARCHAR(255),
    Route_link VARCHAR(500)
);

CREATE TABLE assam_routes (
    Route_name VARCHAR(255),
    Route_link VARCHAR(500)
);

CREATE TABLE uttarpradesh_routes (
    Route_name VARCHAR(255),
    Route_link VARCHAR(500)
);
```

```sql
);

CREATE TABLE westbengal_routes (
    Route_name VARCHAR(255),
    Route_link VARCHAR(500)
);
```

4. **app.py**: The app.py script handles the front-end visualization using Streamlit. It loads the CSV data generated by the previous scripts and presents it in an interactive dashboard where users can filter bus routes based on state, timing, price, and other relevant criteria. The dashboard provides an easy-to-use interface to explore the bus data efficiently.

```python
import streamlit as st
import pymysql
import pandas as pd

# Connect to MySQL database
def get_connection():
    return pymysql.connect(host='localhost', user='root', passwd='bby dragon', database='redbus_travel')

# Function to fetch route names from a specific state table starting with a specific letter
def fetch_route_names(connection, starting_letter, state):
    query = f"SELECT DISTINCT Route_name FROM {state}_routes WHERE Route_name LIKE '{starting_letter}%' ORDER BY Route_name"
    route_names = pd.read_sql(query, connection)['Route_name'].tolist()
    return route_names

# Function to fetch data from the project_info table based on selected ROUTE_NAME
def fetch_data(connection, route_name):
    query = "SELECT * FROM project_info WHERE Route_name = %s"
    df = pd.read_sql(query, connection, params=(route_name,))
    return df

# Function to filter data based on RATING and BUS_TYPE
def filter_data(df, ratings, bus_types):
    filtered_df = df[df['Ratings'].isin(ratings) & df['Bus_type'].isin(bus_types)]
    return filtered_df
```

```python
# Terms and Conditions Content
def terms_and_conditions():
    st.header("Terms and Conditions")
    st.write("""
    1. General Terms
    - All users must provide accurate personal information for booking.
    - Bookings are non-refundable unless otherwise specified.

    2. Payment Terms
    - Payment must be completed online using valid payment methods.
    - The system will display total pricing, including applicable taxes.

    3. Cancellation and Refunds
    - Cancellations are subject to the terms set by each bus operator.
    - Refunds will be processed based on the cancellation policy of the operator.

    4. User Responsibilities
    - Users are responsible for providing correct travel details and adhering to the operator's policies.
    """)

# FAQ Content
def faq_section():
    st.header("Frequently Asked Questions (FAQ)")
    st.write("""
    1. How do I book a bus ticket?
    - You can book a bus ticket by selecting your desired route and bus type, then following the booking process on the homepage.

    2. What payment methods do you accept?
    - We accept major credit cards, debit cards, and online banking.
```

```
def faq_section():
    3. Can I cancel my booking?
    - Yes, you can cancel your booking through the "Manage Booking" section. However, cancellations are subject to the operator's policies.

    4. How will I receive my booking confirmation?
    - Once your booking is successful, you will receive a confirmation email with your ticket details.
    """)

# Main Streamlit app
def main():
    st.sidebar.title("Navigation")

    # Sidebar - Option to choose between Booking, Terms and Conditions, and FAQ
    page = st.sidebar.radio("Go to", ["Bus Booking", "Terms and Conditions", "FAQ"])

    if page == "Bus Booking":
        st.header('Easy and Secure Online Bus Tickets Booking')

        connection = get_connection()

        try:
            # Sidebar - Input for starting letter
            starting_letter = st.sidebar.text_input('Enter Starting Letter of Route Name', 'A')

            # Sidebar - Selectbox for state
            states = ['kerala', 'andhra', 'telangana', 'kadamba', 'rajasthan',
                      'southbengal', 'haryana', 'assam', 'uttarpradesh', 'westbengal']
            selected_state = st.sidebar.selectbox('Select State', states)
```

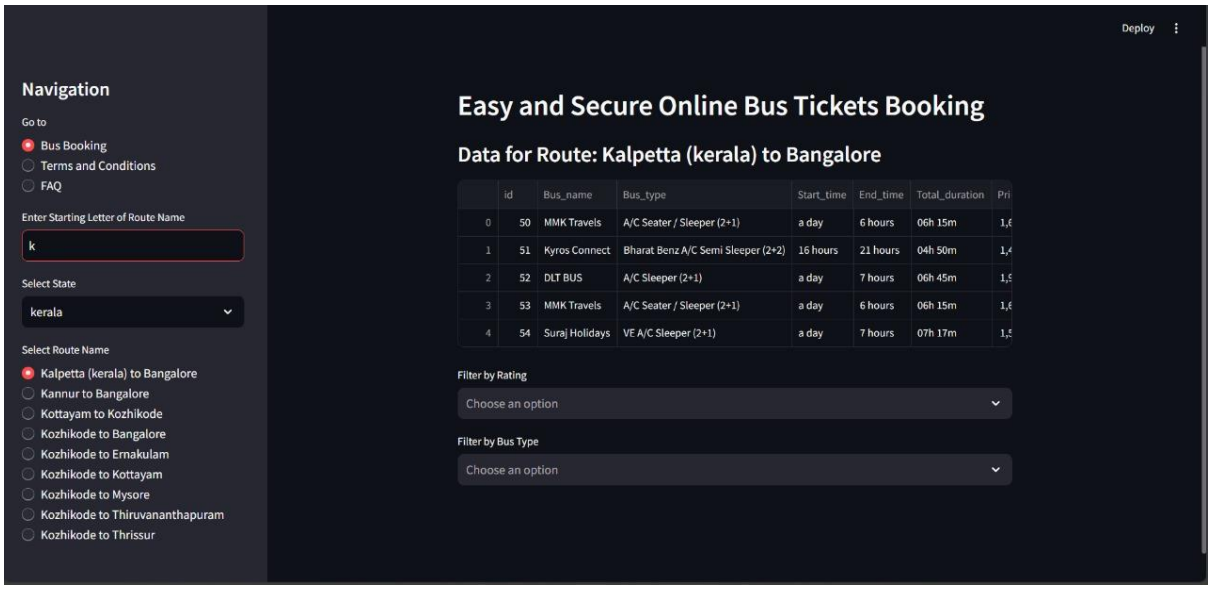
```
            # Fetch route names starting with the specified letter
            if starting_letter and selected_state:
                route_names = fetch_route_names(connection, starting_letter.lower(), selected_state)

                if route_names:
                    # Sidebar - Selectbox for ROUTE_NAME
                    selected_route = st.sidebar.radio('Select Route Name', route_names)

                    if selected_route:
                        # Fetch data based on selected ROUTE_NAME
                        data = fetch_data(connection, selected_route)

                        if not data.empty:
                            # Display data table with a subheader
                            st.write(f"### Data for Route: {selected_route}")
                            st.write(data)

                            # Filter by RATING and BUS_TYPE
                            ratings = data['Ratings'].unique().tolist()
                            selected_ratings = st.multiselect('Filter by Rating', ratings)

                            bus_types = data['Bus_type'].unique().tolist()
                            selected_bus_types = st.multiselect('Filter by Bus Type', bus_types)

                            if selected_ratings and selected_bus_types:
                                filtered_data = filter_data(data, selected_ratings, selected_bus_types)
                                # Display filtered data table with a subheader
                                st.write(f"### Filtered Data for Rating: {selected_ratings} and Bus Type: {selected_bus_types}")
                                st.write(filtered_data)
```

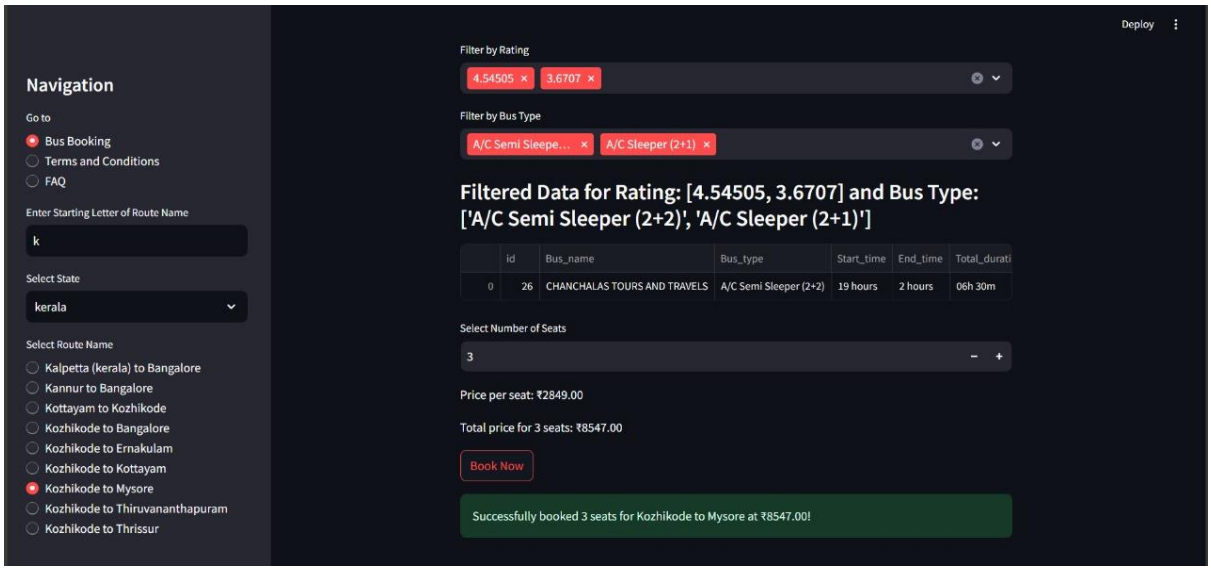
```
64  def main():
114                             # Select number of seats
115                             available_seats = filtered_data['Seats_Available'].sum()
116                             if available_seats > 0:
117                                 num_seats = st.number_input('Select Number of Seats', min_value=1, max_value=available_seats)
118
119                                 # Display price information
120                                 price_per_seat = filtered_data['Price'].mean()  # Average price for selected buses
121                                 total_price = num_seats * price_per_seat
122
123                                 st.write(f"Price per seat: ₹{price_per_seat:.2f}")
124                                 st.write(f"Total price for {num_seats} seats: ₹{total_price:.2f}")
125
126                                 # "Book Now" button
127                                 if st.button('Book Now'):
128                                     # Success message for successful booking
129                                     st.success(f"Successfully booked {num_seats} seats for {selected_route} at ₹{total_price:.2f}!")
130
131                                     # Display celebratory balloons
132                                     st.balloons()
133                             else:
134                                 st.warning("No seats available for the selected filters.")
135                     else:
136                         st.write(f"No data found for Route: {selected_route}.")
137                 else:
138                     st.write("No routes found starting with the specified letter.")
139         finally:
140             connection.close()
```

```
142     elif page == "Terms and Conditions":
143         terms_and_conditions()
144
145     elif page == "FAQ":
146         faq_section()
147
148 if __name__ == "__main__":
149     main()
```

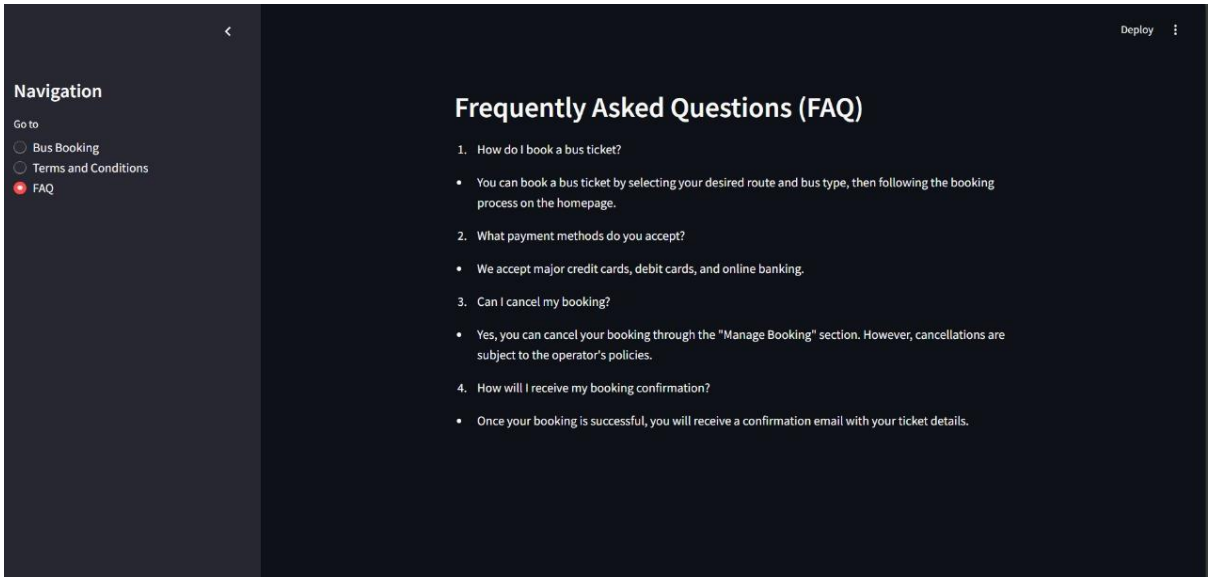
5.OUTPUT

Here I have attached screenshot my output

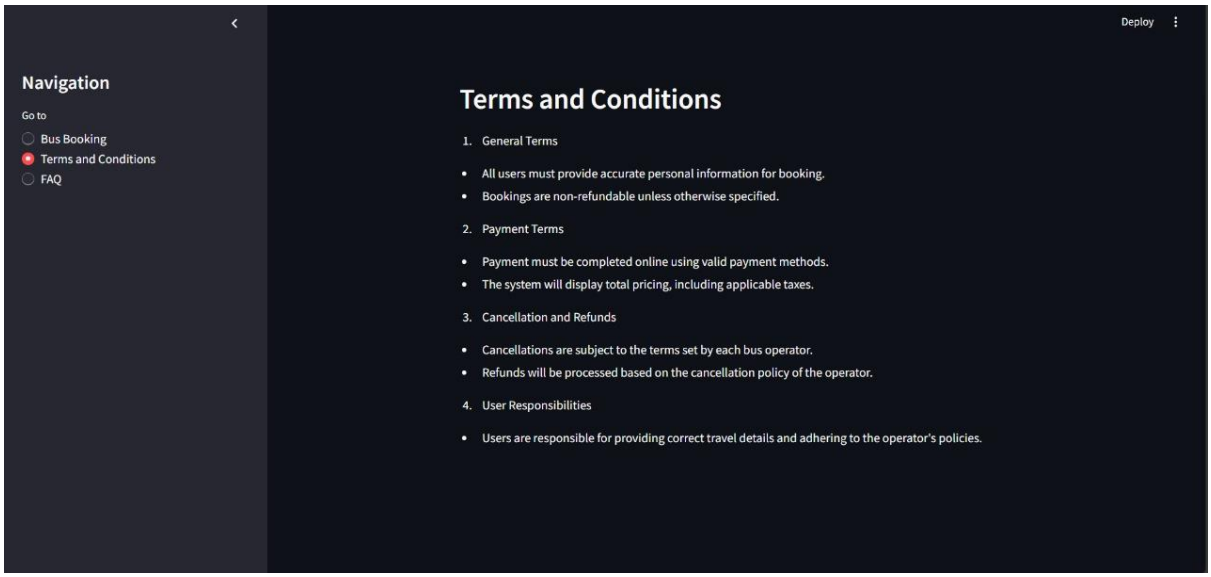




FAQ



TERMS AND CONDITION



6.CHALLENGES

Challenge: Managing Large Data Volumes

- Problem: The volume of data being scraped (multiple bus routes, schedules, prices) could be large, which may lead to performance issues when loading and filtering in the Streamlit application.
- Solution: Optimize the database schema to ensure fast query execution. Use pagination or batching techniques for scraping and storing data incrementally. In the Streamlit app, load only the required data for each user query to avoid handling unnecessary data all at once.

3. Challenge: Maintaining Scraping Accuracy

- Problem: Ensuring the scraped data is accurate, especially when dealing with inconsistent or frequently updated data from Redbus.
- Solution: Use robust error-handling mechanisms to deal with missing or incorrectly formatted data. Regularly monitor the website for layout changes and update the scraping logic as needed. Store timestamps with each entry to track data freshness and allow periodic re-scraping.

7. CONCLUSION

The Redbus Data Scraping with Selenium & Dynamic Filtering using Streamlit project successfully automates the collection of bus route data and offers a dynamic and interactive interface for users. The combination of Selenium for scraping and Streamlit for presenting the data provides a powerful tool for exploring and filtering large datasets in real time. This project demonstrates the effectiveness of automation in data collection and highlights the ease with which data can be processed and visualized for end users. Future improvements could include real-time updates and integration with additional data sources for more comprehensive travel information.

FUTURE SCOPE

Integration with Booking Platforms: Integrating the application with online booking platforms would provide a seamless user experience for travelers.

Real-time Alerts: Implementing real-time alerts for changes in bus schedules or prices could keep users informed and help them make timely decisions.