**REDBUS PROJECT**

**1.Redbus project overview:**

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

**Objective**:The project aims to automate the scraping of bus travel data from Redbus and provide real-time insights through a user-friendly Streamlit interface.

|  |  |
| --- | --- |
| **Project Title** | **Redbus Data Scraping with Selenium & Dynamic Filtering using Streamlit** |
| **Skills take away From This Project** | **Web Scraping using Selenium, Python, Streamlit , SQL** |
| **Domain** | **Transportation** |

# **2. . Business Use Cases**

● Travel Aggregators: Providing real-time bus seat availability for customers.

● Market Analysis: Analyzing travel patterns and preferences for market research.

● Customer Service: Enhancing user experience by offering customized travel options based on data insights.

● Competitor Analysis: Comparing pricing and service levels with competitors

**Travel Aggregators**

* + Use Case: Real-Time Bus Schedules and Seat Availability
  + Travel aggregators benefit significantly from the integration of real-time data into their platforms. By utilizing this solution, they can provide up-to-the-minute information on bus schedules, routes, and seat availability. This enhances customer satisfaction by allowing users to make informed decisions instantly.
  + **Key Features**:
  + Real-time updates on bus schedules and delays
  + Seat availability tracking
  + Route optimization for customers
  + User notifications for schedule changes
  + **Business Impact**:
  + Increases customer retention by offering more value through real-time information
  + Reduces manual intervention needed for updating schedules
  + Improves efficiency in managing peak-hour bookings

**Technical Tags:**

● Web Scraping

● Selenium

● Streamlit

● SQL

● Data Analysis

● Python

**4.Data Scraping with Selenium**

**Objective:**Automate the extraction of bus data from Redbus, focusing on key parameters such as routes, schedules, prices, and seat availability.

**Implementation Steps:**

* **Setup Environment**:
  + Install Selenium and required web drivers (e.g., ChromeDriver).
  + Set up a Python environment using virtualenv or Anaconda.
* **Write the Scraping Script**:
  + Import necessary libraries: selenium, pandas, etc.
  + Use Selenium to launch a browser and navigate to the Redbus website.
  + Locate elements containing the required data (e.g., using XPath or CSS selectors).
  + Extract data into structured formats (lists or dictionaries).
* **Data Points to Scrape**:
  + **Routes**: Start and destination locations.
  + **Schedules**: Departure and arrival times.
  + **Prices**: Ticket prices for different bus types.
  + **Seat Availability**: Number of available seats for each bus.

.

Codes

#scraping data using selenium from webpage

import pandas as pd

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

# 10 states webpage 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/cha/?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>"

]

driver=webdriver.Chrome()

#load the webpage

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

wait = WebDriverWait(driver, 20)

def ap\_link\_route(path):

LINKS\_ap = []

ROUTE\_ap= []

# Retrieve the route links and route names

for i in range(1, 6): # Adjust the range based on the actual number of pages, e.g.,

paths = driver.find\_elements(By.XPATH, path)

# Retrieve links

for links in paths:

d = links.get\_attribute("href")

LINKS\_ap.append(d)

# Retrieve names of the routes

for route in paths:

ROUTE\_ap.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"]')))

# Check if next button for the next page exists

next\_button\_xpath = f'//div[@class="DC\_117\_pageTabs " and 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(next\_button))

driver.execute\_script("arguments[0].click();", next\_button)

# Click using JavaScript

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}")

break

return LINKS\_ap, ROUTE\_ap

# Calling the function to get links and routes

LINKS\_ap, ROUTE\_ap = ap\_link\_route("//a[@class='route']")

df\_k=pd.DataFrame({"Route\_name":ROUTE\_ap,"Route\_link":LINKS\_ap})

df\_k

Route\_name Route\_link

0 Hyderabad to Vijayawada <https://www.redbus.in/bus-tickets/hyderabad-to>...

1 Vijayawada to Hyderabad <https://www.redbus.in/bus-tickets/vijayawada-t>...

2 Hyderabad to Ongole <https://www.redbus.in/bus-tickets/hyderabad-to>...

3 Kakinada to Visakhapatnam <https://www.redbus.in/bus-tickets/kakinada-to>-...

4 Bangalore to Tirupati <https://www.redbus.in/bus-tickets/bangalore-to>...

5 Bangalore to Kadapa <https://www.redbus.in/bus-tickets/bangalore-to>...

6 Ongole to Hyderabad <https://www.redbus.in/bus-tickets/ongole-to-hy>...

7 Kadapa to Bangalore <https://www.redbus.in/bus-tickets/kadapa-to-ba>...

8 Chittoor (Andhra Pradesh) to Bangalore <https://www.redbus.in/bus-tickets/chittoor-and>...

9 Visakhapatnam to Kakinada <https://www.redbus.in/bus-tickets/visakhapatna>...

10 Bangalore to Anantapur (andhra pradesh) <https://www.redbus.in/bus-tickets/bangalore-to>...

11 Bangalore to Chittoor (Andhra Pradesh) <https://www.redbus.in/bus-tickets/bangalore-to>...

12 Anantapur (andhra pradesh) to Bangalore <https://www.redbus.in/bus-tickets/ananthapur-t>...

13 Hyderabad to Kurnool <https://www.redbus.in/bus-tickets/hyderabad-to>...

14 Tirupati to Bangalore <https://www.redbus.in/bus-tickets/tirupathi-to>...

15 Narasaraopet to Hyderabad <https://www.redbus.in/bus-tickets/narsaraopet>-...

16 Vinukonda to Hyderabad <https://www.redbus.in/bus-tickets/vinukonda-to>...

17 Hyderabad to Vinukonda <https://www.redbus.in/bus-tickets/hyderabad-to>...

18 Visakhapatnam to Vijayawada <https://www.redbus.in/bus-tickets/visakhapatna>...

19 Bangalore to Rayachoti <https://www.redbus.in/bus-tickets/bangalore-to>...

20 Hyderabad to Guntur (Andhra Pradesh) <https://www.redbus.in/bus-tickets/hyderabad-to>...

21 Guntur (Andhra Pradesh) to Hyderabad <https://www.redbus.in/bus-tickets/guntur-to-hy>...

22 Hyderabad to Eluru <https://www.redbus.in/bus-tickets/hyderabad-to>...

23 Eluru to Hyderabad <https://www.redbus.in/bus-tickets/eluru-to-hyd>...

24 Bangalore to Kadiri <https://www.redbus.in/bus-tickets/bangalore-to>...

25 Madanapalli to Bangalore <https://www.redbus.in/bus-tickets/madanapalli>-...

26 Bangalore to Madanapalli <https://www.redbus.in/bus-tickets/bangalore-to>...

27 Macherla (andhra pradesh) to Hyderabad <https://www.redbus.in/bus-tickets/macherla-to>-...

28 Rajahmundry to Visakhapatnam <https://www.redbus.in/bus-tickets/rajahmundry>-...

29 Nandyal to Hyderabad <https://www.redbus.in/bus-tickets/nandyala-to>-...

driver\_k = webdriver.Chrome()

Bus\_names\_k = []

Bus\_types\_k = []

Start\_Time\_k = []

End\_Time\_k = []

**6. Data Storage in SQL Database**

**Objective**Store the scraped data in a structured SQL database for easy retrieval and analysis.

**Implementation Steps**

* Set Up SQL Database:
  + Choose a SQL database (e.g., MySQL, PostgreSQL).
  + Create a database and relevant tables (e.g., buses, routes, schedules).
* **Store Data**:
  + Use SQLAlchemy or a similar ORM for database interactions.
  + Insert scraped data into the appropriate tables.

**Example Code Snippet**

python

Copy code

from sqlalchemy import create\_engine

import pandas as pd

# Create a database connection

engine = create\_engine('mysql+pymysql://username:password@localhost/redbus')

# Create a DataFrame from scraped data

data = {'route': [], 'schedule': [], 'price': [], 'availability': []}

df = pd.DataFrame(data)

# Store DataFrame in SQL database

df.to\_sql('buses', con=engine, if\_exists='append', index=False)

**3. Streamlit Application Development**

**Objective**

Develop a user-friendly web application using Streamlit to display and filter the scraped data.

**Implementation Steps**

* **Set Up Streamlit Environment**:
  + Install Streamlit: pip install streamlit.
* **Develop the Application**:
  + Create a Streamlit script (e.g., app.py).
  + Use SQL queries to retrieve data from the database.
  + Implement user input forms and filters for bus type, route, price range, star rating, and availability.

**Example Code Snippet**

python

Copy code

import streamlit as st

import pandas as pd

from sqlalchemy import create\_engine

**# Database connection**

engine = create\_engine('mysql+pymysql://username:password@localhost/redbus')

# Streamlit UI

st.title("Redbus Data Filter")

# Filters

bus\_type = st.selectbox("Select Bus Type", ["Sleeper", "Semi-Sleeper", "AC"])

price\_range = st.slider("Price Range", 0, 5000, (0, 1000))

# Query to filter data

query = f"SELECT \* FROM buses WHERE bus\_type = '{bus\_type}' AND price BETWEEN {price\_range[0]} AND {price\_range[1]}"

filtered\_data = pd.read\_sql(query, con=engine)

# Display Data

st.dataframe(filtered\_data)

**Data Analysis/Filtering using Streamlit**

**Objective**

Enhance user interaction by allowing users to filter and analyze the bus data through the Streamlit application.

**Implementation Steps**

* **SQL Query Integration**:
  + Write SQL queries that incorporate user inputs for filtering.
* **Interactive Features**:
  + Use Streamlit widgets to allow users to select different filters.
  + Display the filtered results dynamically based on user selections.

**Example Code Snippet**

Python

Copy code

# Implementing additional filters

availability = st.selectbox("Availability", ["Available", "Unavailable"])

# Modify query based on availability filter

if availability == "Available":

query += " AND availability > 0"

# Execute and display updated results

filtered\_data = pd.read\_sql(query, con=engine)

st.dataframe(filtered\_data)

**Conclusion**

This project encompasses the entire pipeline from data extraction to visualization, providing valuable insights into bus schedules, prices, and availability. The integration of Selenium, SQL, and Streamlit creates a robust solution for end-users seeking real-time travel data.

* + **2. Market Analysis**
  + **Use Case: Analyzing Travel Patterns and Preferences**
  + This solution provides the tools necessary to gather and analyze vast amounts of travel data. Aggregating this information can give businesses insights into customer preferences and travel trends.

**Project Evaluation Metrics**

**1. Data Scraping Accuracy**

**Objective**

Assess the completeness and correctness of the data extracted from the Redbus website.

**Evaluation Criteria**

* **Completeness**:
  + Percentage of required data points successfully scraped (e.g., routes, schedules, prices, seat availability).
  + Number of missing data entries compared to the expected total.
* **Correctness**:
  + Validation of data against known benchmarks or sample data.
  + Manual verification of a random sample of scraped data for accuracy (e.g., checking a few routes and prices).

**Metrics:**

* **Completeness Rate**: Completeness Rate=Total Scraped Data PointsTotal Required Data Points×100\text{Completeness Rate} = \frac{\text{Total Scraped Data Points}}{\text{Total Required Data Points}} \times 100Completeness Rate=Total Required Data PointsTotal Scraped Data Points​×100
* **Error Rate**: Error Rate=Number of Incorrect Data PointsTotal Sample Size×100\text{Error Rate} = \frac{\text{Number of Incorrect Data Points}}{\text{Total Sample Size}} \times 100Error Rate=Total Sample SizeNumber of Incorrect Data Points​×100

**2. Database Design**

**Objective**

Evaluate the effectiveness and efficiency of the database schema used for storing scraped data.

**Evaluation Criteria**

* **Normalization**:
  + Check for appropriate normalization to reduce redundancy.
  + Ensure that the design follows best practices for relational databases.
* **Query Performance**:
  + Assess the speed and efficiency of common SQL queries used in the application.
* **Scalability**:
  + Evaluate how well the database design can handle an increase in data volume and user queries.

**Metrics:**

* **Normalization Level**:
  + Review for compliance with the normal forms (1NF, 2NF, 3NF).
* **Average Query Response Time**: Average Query Time=Total Query TimeNumber of Queries\text{Average Query Time} = \frac{\text{Total Query Time}}{\text{Number of Queries}}Average Query Time=Number of QueriesTotal Query Time​

**3. Application Usability**

**Objective**

Measure the user experience and ease of use of the Streamlit application.

**Evaluation Criteria**

* **User Interface**:
  + Assess the visual layout and organization of the application.
  + Ensure the interface is intuitive and accessible to users of varying technical skill levels.
* **User Feedback**:
  + Collect feedback from users regarding their experience using the application.

**Metrics:**

* **User Satisfaction Score**:
  + Conduct surveys or polls to gather user feedback (e.g., Likert scale ratings).
* **Task Completion Rate**: Task Completion Rate=Number of Successful TasksTotal Tasks Attempted×100\text{Task Completion Rate} = \frac{\text{Number of Successful Tasks}}{\text{Total Tasks Attempted}} \times 100Task Completion Rate=Total Tasks AttemptedNumber of Successful Tasks​×100

1. **Filter Functionality**

**Objective**

Evaluate the effectiveness and responsiveness of the data filters within the Streamlit application.

**Evaluation Criteria**

* **Responsiveness**:
  + Measure the time taken for the application to update results based on user inputs.
* **Filter Accuracy**:
  + Assess how accurately the filters narrow down results according to user selections.

**Metrics:**

* **Average Response Time for Filters**: Average Filter Response Time=Total Filter Response TimeNumber of Filter Actions\text{Average Filter Response Time} = \frac{\text{Total Filter Response Time}}{\text{Number of Filter Actions}}Average Filter Response Time=Number of Filter ActionsTotal Filter Response Time​
* **Filter Accuracy Rate**: Filter Accuracy Rate=Correctly Filtered ResultsTotal Filtered Results×100\text{Filter Accuracy Rate} = \frac{\text{Correctly Filtered Results}}{\text{Total Filtered Results}} \times 100Filter Accuracy Rate=Total Filtered ResultsCorrectly Filtered Results​×100

**Code Quality**

**Objective**

Assess the adherence to coding standards and best practices throughout the project.

**Evaluation Criteria**

* **Code Readability**:
  + Evaluate the clarity and organization of the code, including naming conventions and comments.
* **Error Handling**:
  + Check for appropriate error handling mechanisms within the code.
* **Testing**:
  + Ensure that unit tests are present and effectively validate the functionality of the code.

**Metrics:**

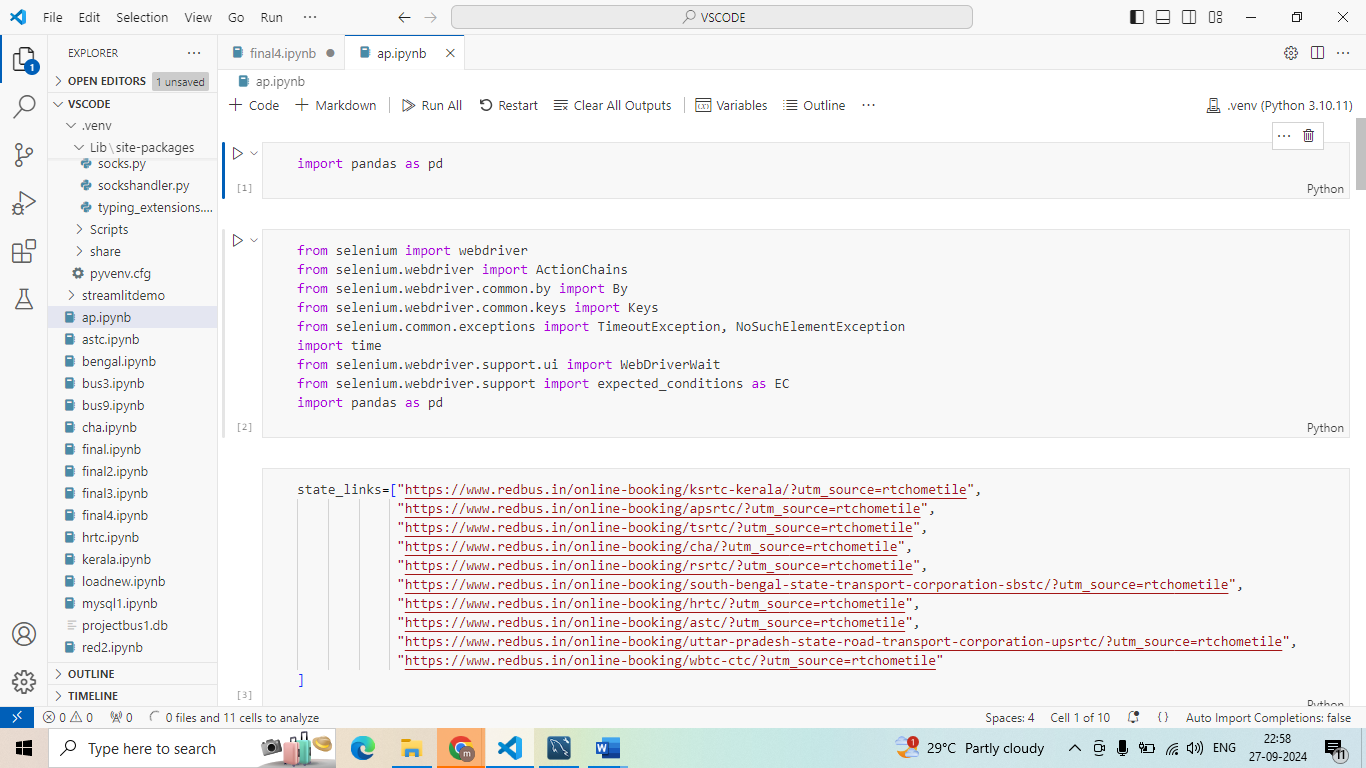
* **Code Review Score**:
  + Conduct peer reviews and assign scores based on predefined criteria (e.g., readability, organization).
* **Test Coverage Percentage**: Test Coverage=Number of Tested Code LinesTotal Code Lines×100\text{Test Coverage} = \frac{\text{Number of Tested Code Lines}}{\text{Total Code Lines}} \times 100Test Coverage=Total Code LinesNumber of Tested Code Lines​×100

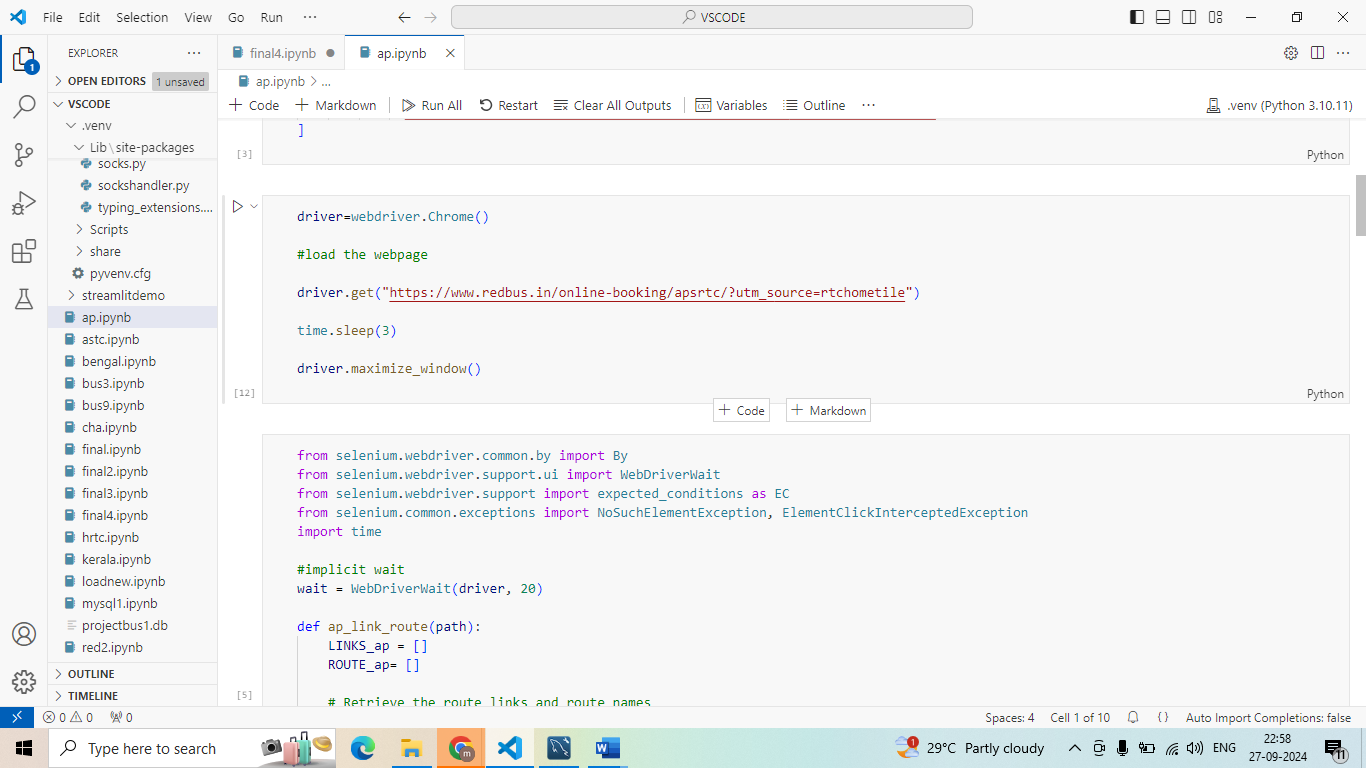
**Conclusion**

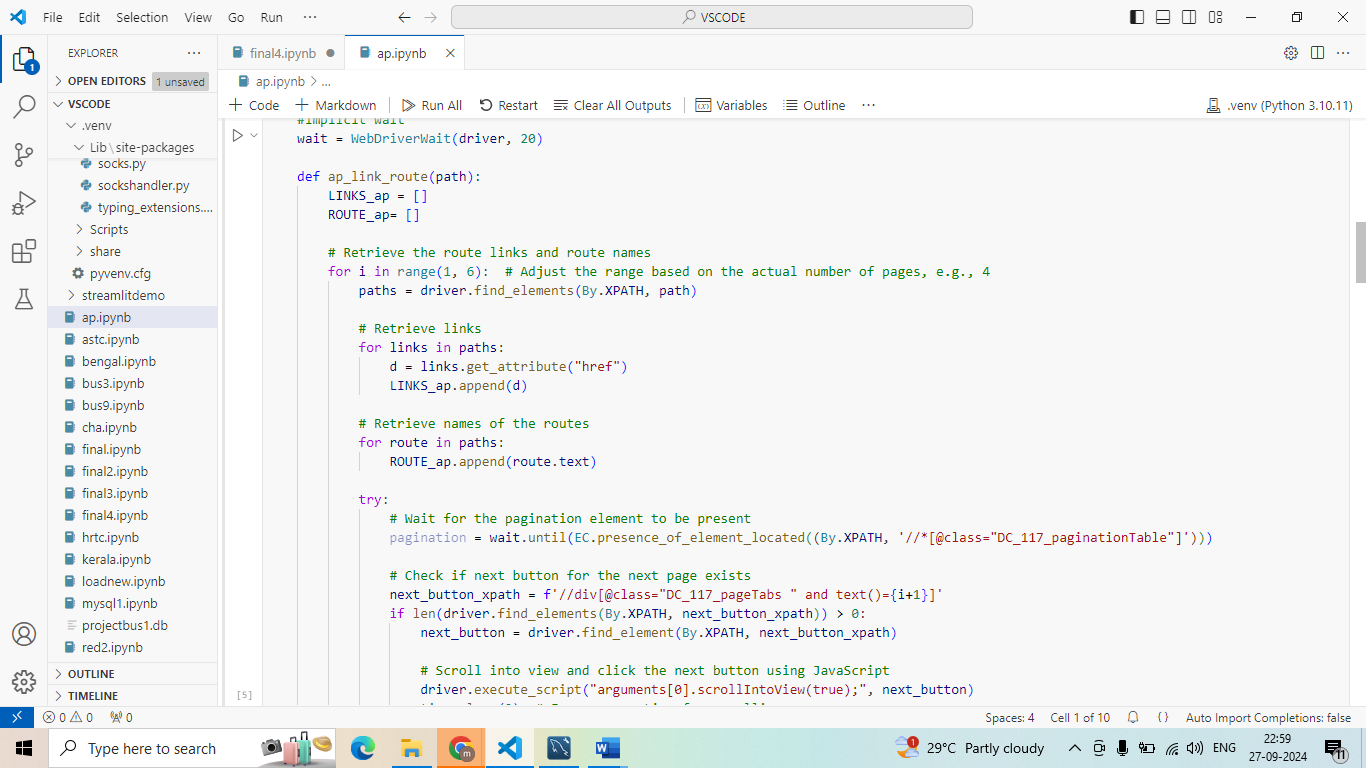
These evaluation metrics will provide a comprehensive assessment of the project's effectiveness and quality across various dimensions. By measuring these aspects, you can identify areas for improvement and ensure that the solution meets user needs and expectations.

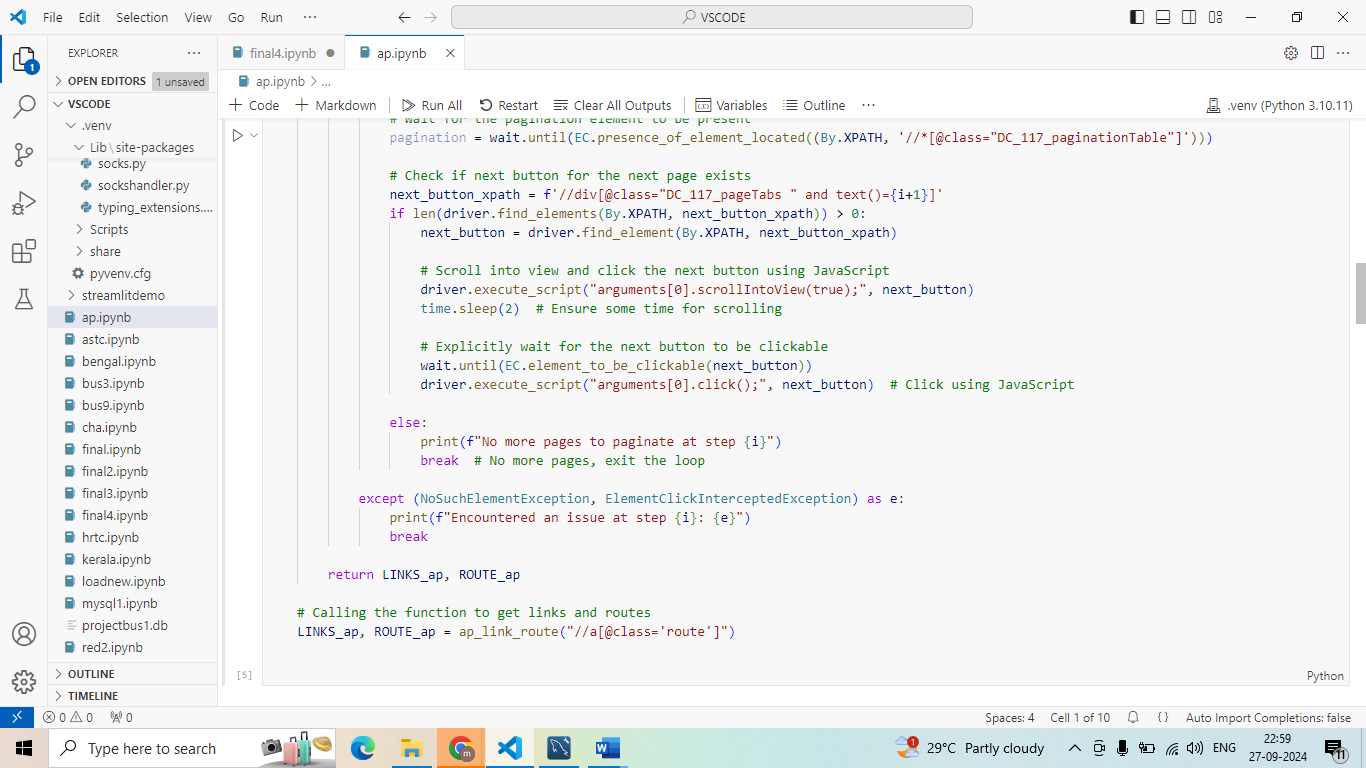
* + **Key Features**:
  + Data-driven insights on popular routes and travel times
  + Analytics for peak travel seasons
  + Demographic segmentation for customer travel behavior
  + **Business Impact**:
  + Helps companies identify new business opportunities and markets
  + Optimizes marketing strategies based on customer preferences
  + Assists in forecasting demand and planning for high-traffic periods
  + **3. Customer Service**
  + **Use Case: Offering Customized Travel Options**
  + Customer service teams can greatly enhance the user experience by leveraging travel data. This allows the provision of personalized recommendations based on user history, preferences, and real-time conditions.
  + **Key Features**:
  + Customized travel suggestions based on past travel history
  + User-specific deals and discounts
  + Integration with customer support channels for real-time problem resolution
  + **Business Impact**:
  + Boosts customer satisfaction through personalized recommendations
  + Increases sales of add-ons and upgrades by offering timely promotions
  + Enhances customer loyalty by delivering tailored travel experiences
  + **4. Competitor Analysis**
  + **Use Case: Comparing Pricing and Service Levels with Competitors**
  + The solution enables businesses to continuously monitor competitor offerings, pricing structures, and service levels, enabling strategic adjustments to stay competitive in the market.
  + **Key Features**:
  + Real-time competitor price tracking
  + Analysis of competitor service offerings (e.g., travel routes, seat availability)
  + Benchmarking tools for comparing customer ratings and feedback
  + **Business Impact**:
  + Allows businesses to adjust pricing dynamically in response to competitors
  + Provides insight into potential gaps in service that can be exploited
  + Helps in the creation of competitive marketing campaigns based on real-time data
  + **Conclusion**
  + By leveraging this solution across various business scenarios, companies in the travel sector can optimize operations, improve customer experiences, and stay ahead of their competitors. The real-time data and insights generated will enable businesses to make data-driven decisions, leading to higher revenue and stronger customer relationships.

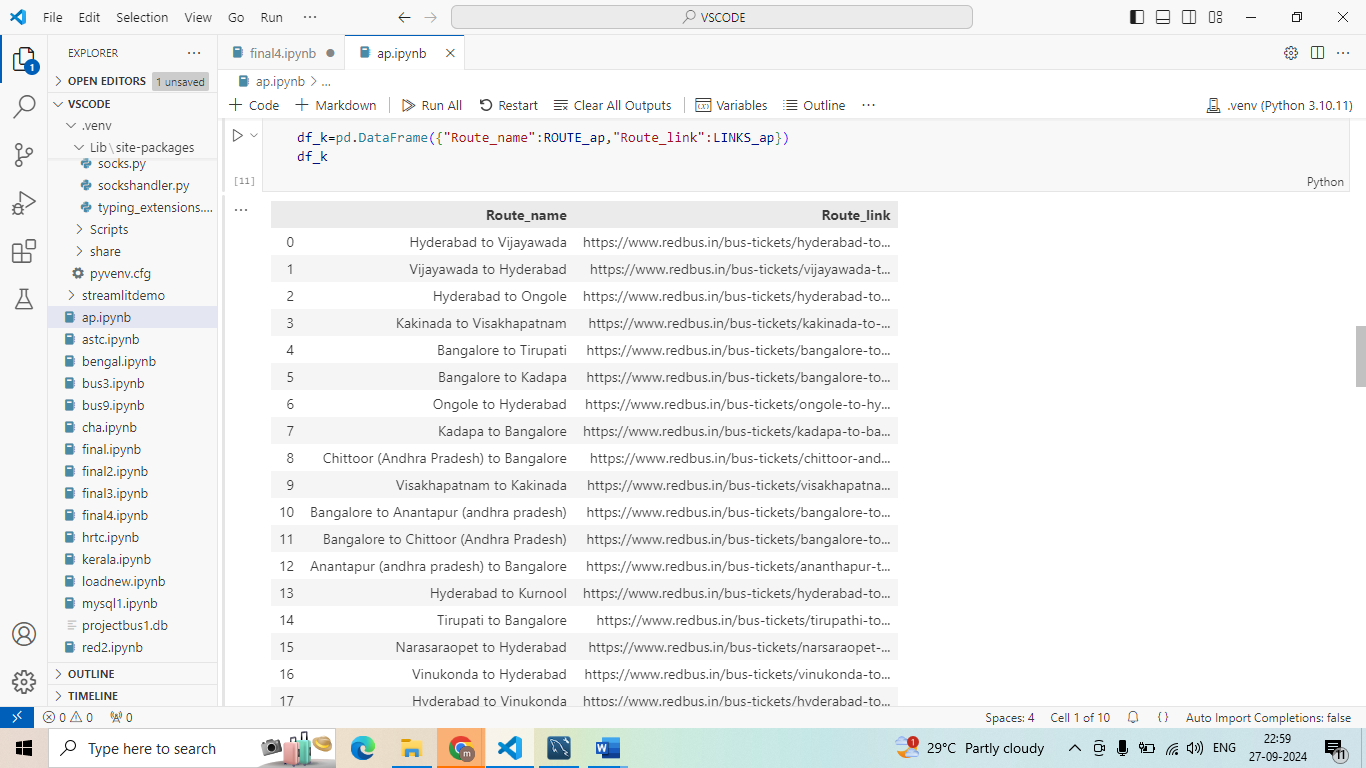
...











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")

# collecting information into data

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,

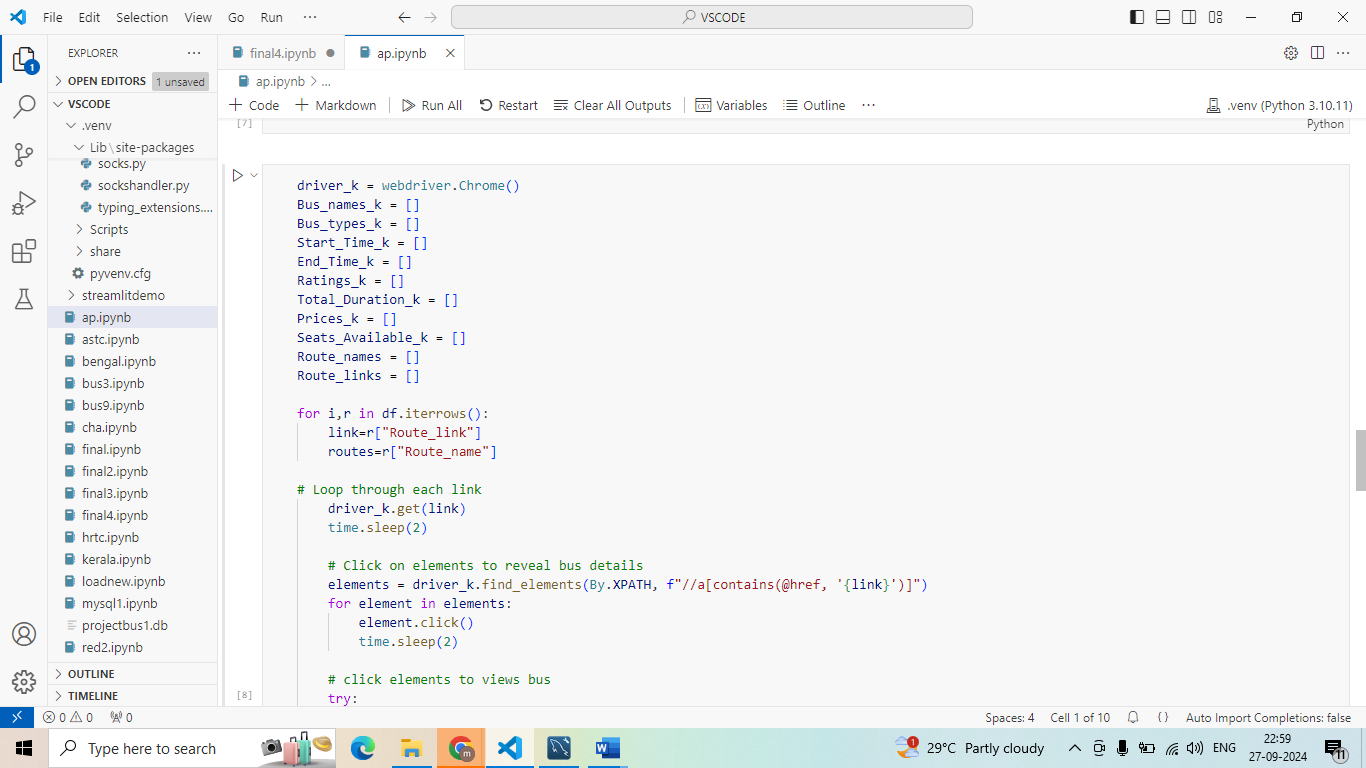
'Price': Prices\_k,

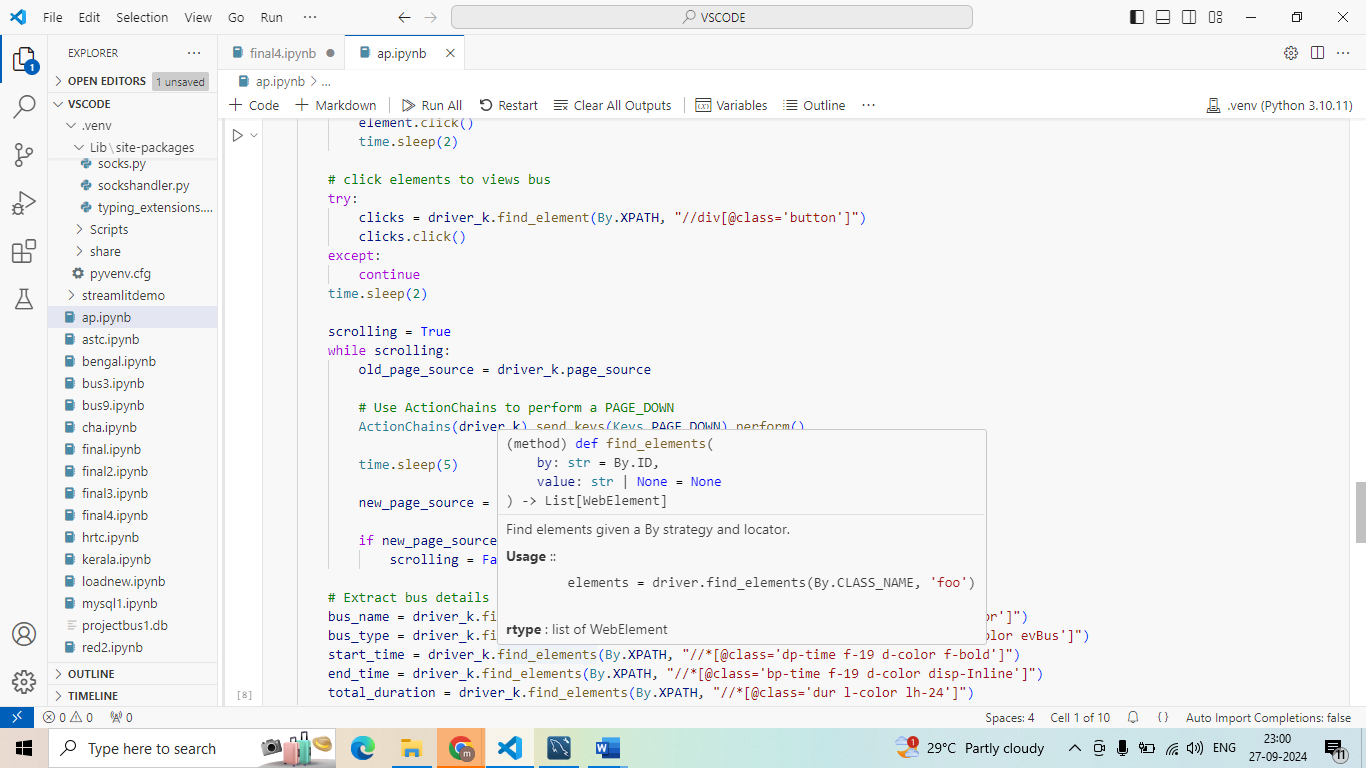
"Seats\_Available":Seats\_Available\_k,

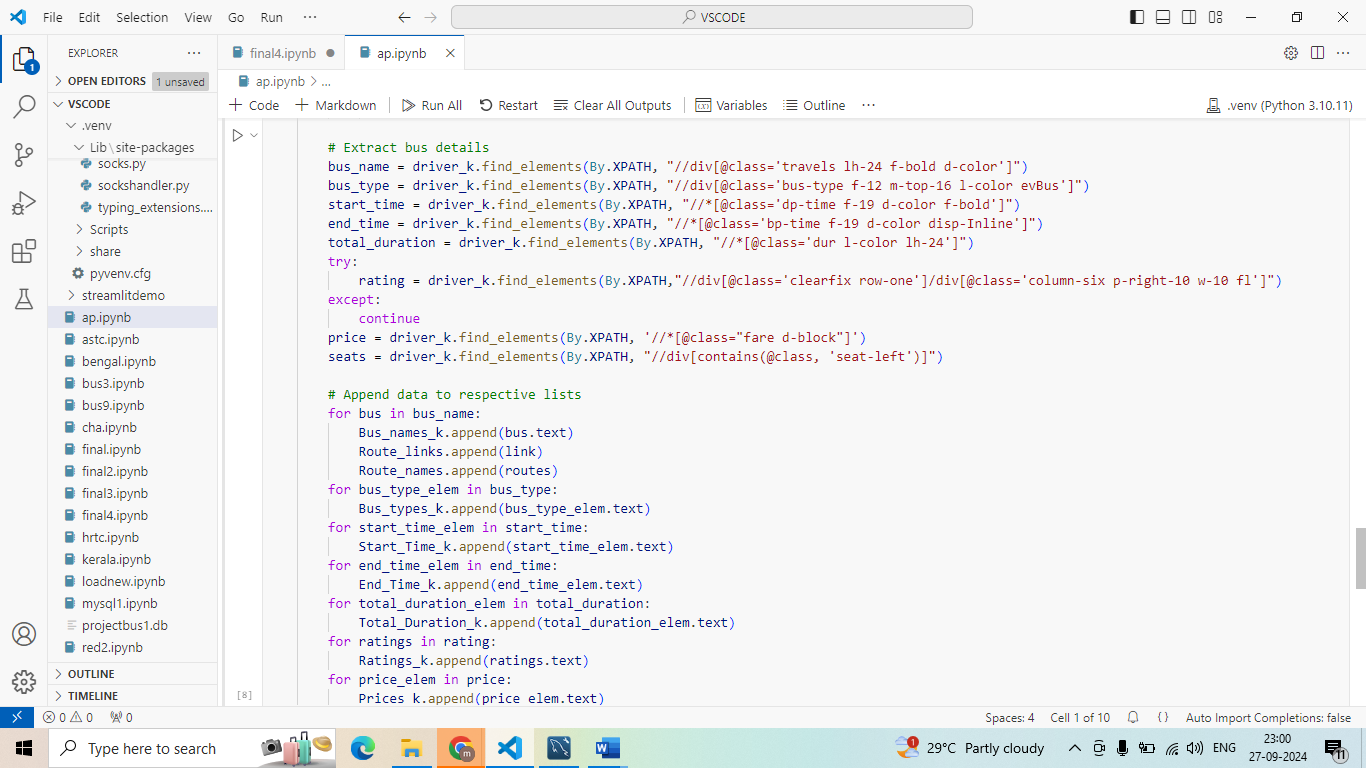
"Ratings":Ratings\_k,

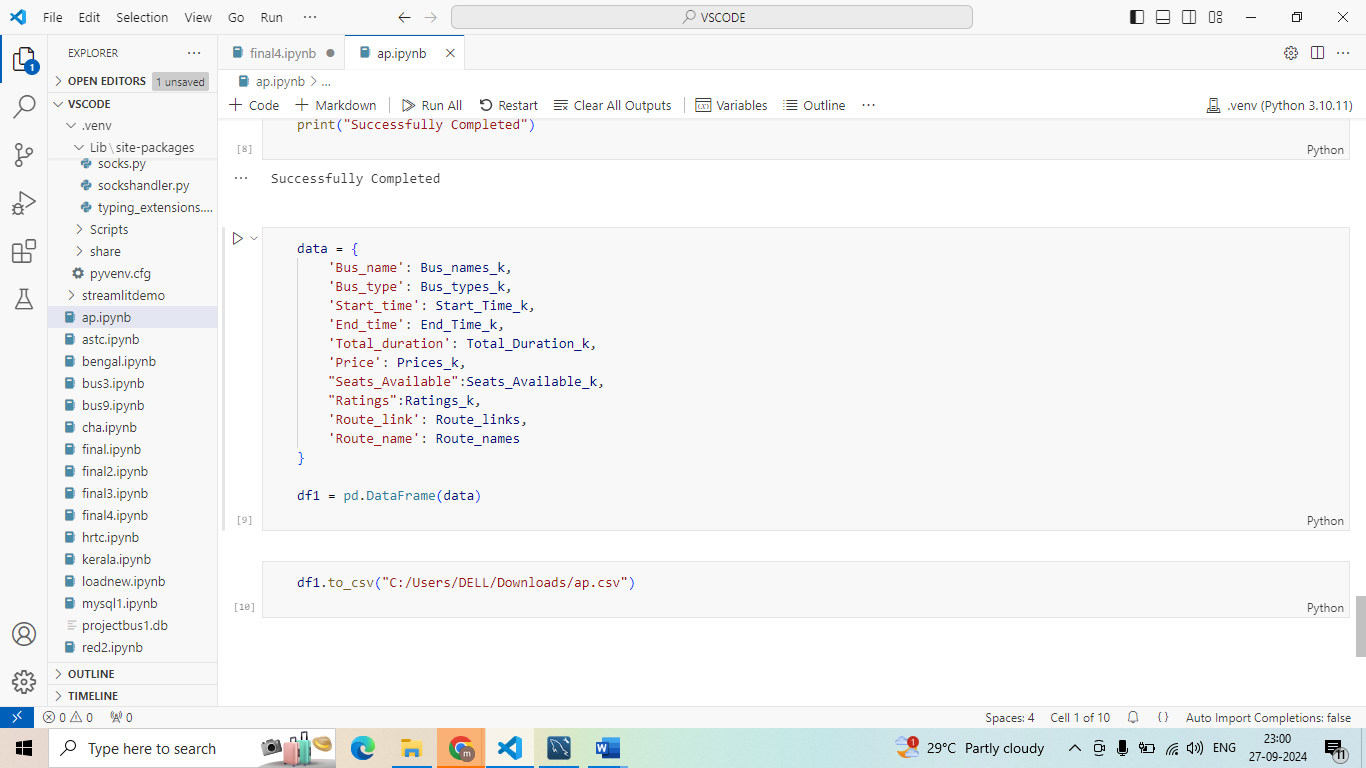
'Route\_link': Route\_links,

'Route\_name': Route\_names}









**Storing all route datas in one table**

df1 = pd.DataFrame(data)

#downloading the file

df1.to\_csv("C:/Users/DELL/Downloads/ap.csv")

#like above download for 10 states

df1 = pd.read\_csv("C:/redbus5/wbtcl.csv")

df2 = pd.read\_csv("C:/redbus5/upl.csv")

df3 = pd.read\_csv("C:/redbus5/tsrtcl.csv")

df4 = pd.read\_csv("C:/redbus5/rsrtc1.csv")

df5 = pd.read\_csv("C:/redbus5/kerala1.csv")

df6 = pd.read\_csv("C:/redbus5/hrtc1.csv")

df7 = pd.read\_csv("C:/redbus5/bengal1.csv")

df8 = pd.read\_csv("C:/redbus5/astc.csv")

df9 = pd.read\_csv("C:/redbus5/ap.csv")

df10 = pd.read\_csv("C:/redbus5/cha.csv")

#cocatinate the 10 state files

df = pd.concat([df1, df2,df3,df4,df5,df6,df7,df8,df9,df10])

#make cocatinating file as one df

df=pd.read\_csv("C:/redbus5/redbus.csv")

**6 ..#python and mysql connect**

import pandas as pd

import numpy as py

import pymysql

df1 = pd.read\_csv("C:/redbus5/wbtcl.csv")

df2 = pd.read\_csv("C:/redbus5/upl.csv")

df3 = pd.read\_csv("C:/redbus5/tsrtcl.csv")

df4 = pd.read\_csv("C:/redbus5/rsrtc1.csv")

df5 = pd.read\_csv("C:/redbus5/kerala1.csv")

df6 = pd.read\_csv("C:/redbus5/hrtc1.csv")

df7 = pd.read\_csv("C:/redbus5/bengal1.csv")

df8 = pd.read\_csv("C:/redbus5/astc.csv")

df9 = pd.read\_csv("C:/redbus5/ap.csv")

df10 = pd.read\_csv("C:/redbus5/cha.csv")

df = pd.concat([df1, df2,df3,df4,df5,df6,df7,df8,df9,df10])

df=pd.read\_csv("C:/redbus5/redbus.csv")

df['id'] = range(1, len(df) + 1)

if 'Price' in df.columns:

df['Price'] = df['Price'].fillna('').astype(str).str.replace('INR ', '', regex=False)

if 'Seat\_Availability' in df.columns:

df['Seat\_Availability'] = df['Seat\_Availability'].fillna('').str.extract(r'(\d+)')

df.to\_csv("redbus.csv", index=False)

myconnection = pymysql.connect(host='127.0.0.1', user='root', passwd='Ajay2017', database="projectbus")

df = df.dropna()

dtype\_map = {

'float64': 'FLOAT',

'object': 'TEXT',

'int64': 'INT'

}

columns\_and\_types = ", ".join(f"{col} {dtype\_map[str(dtype)]}" for col, dtype in df.dtypes.items())

table\_name = "redbus"

with myconnection.cursor() as cursor:

cursor.execute(f"CREATE TABLE IF NOT EXISTS {table\_name} ({columns\_and\_types})")

with myconnection.cursor() as cursor:

for i in range(len(df)):

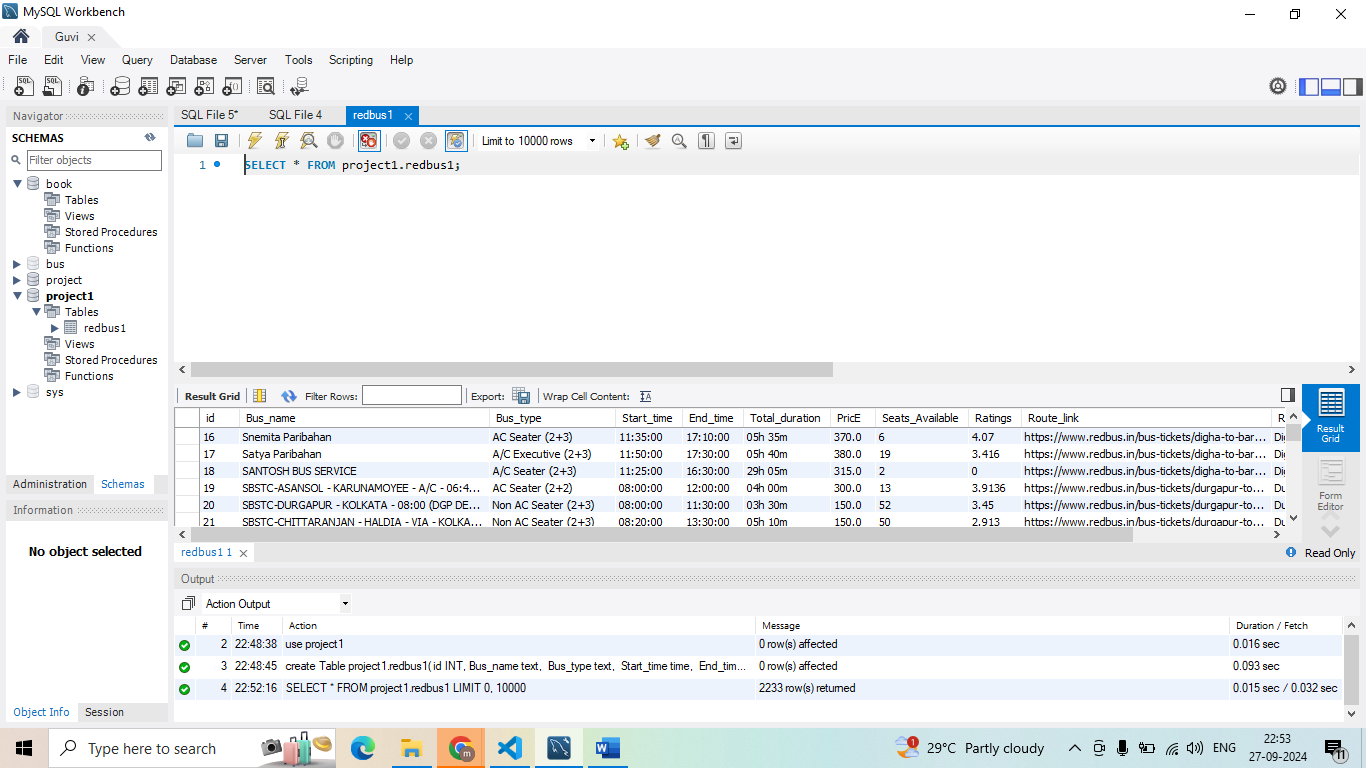
# Use parameterized queries for security and performance

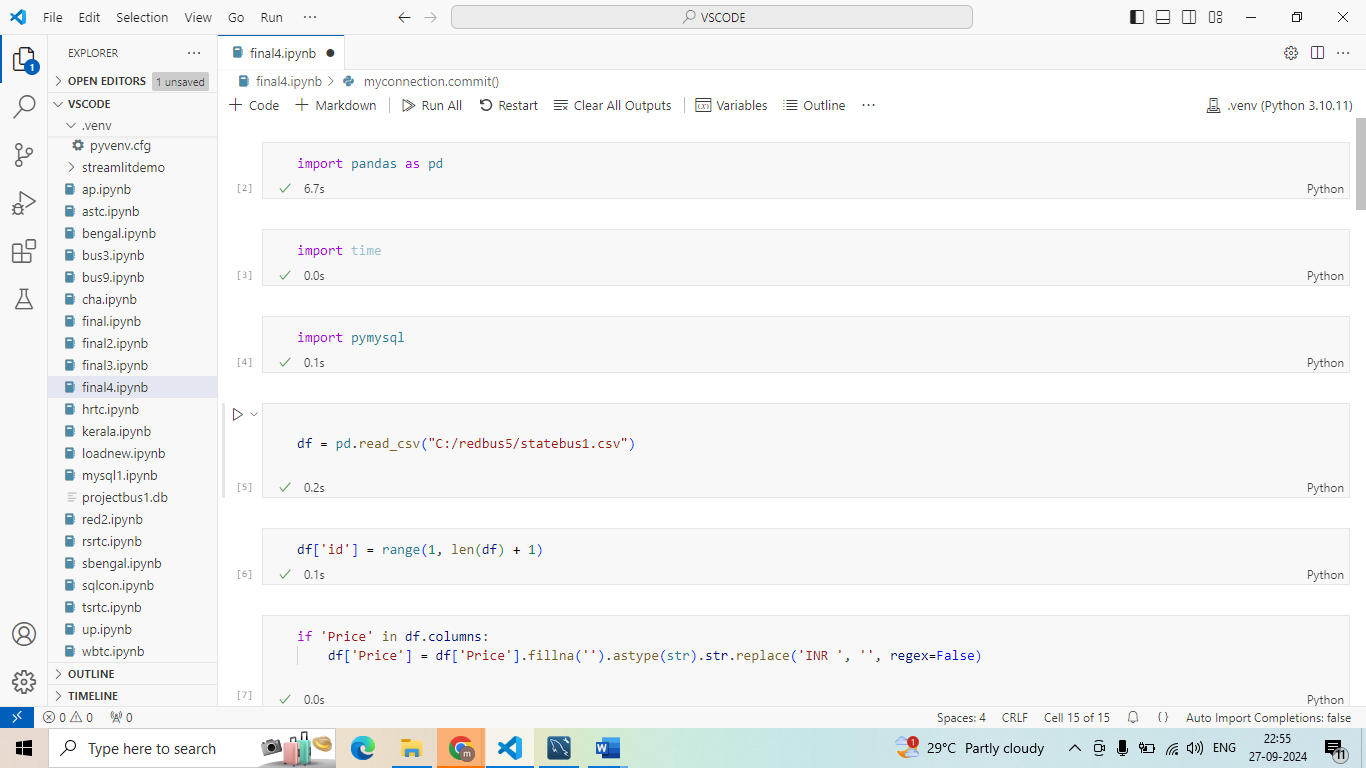
sql = f"INSERT INTO {table\_name} VALUES ({','.join(['%s'] \* len(df.columns))})"

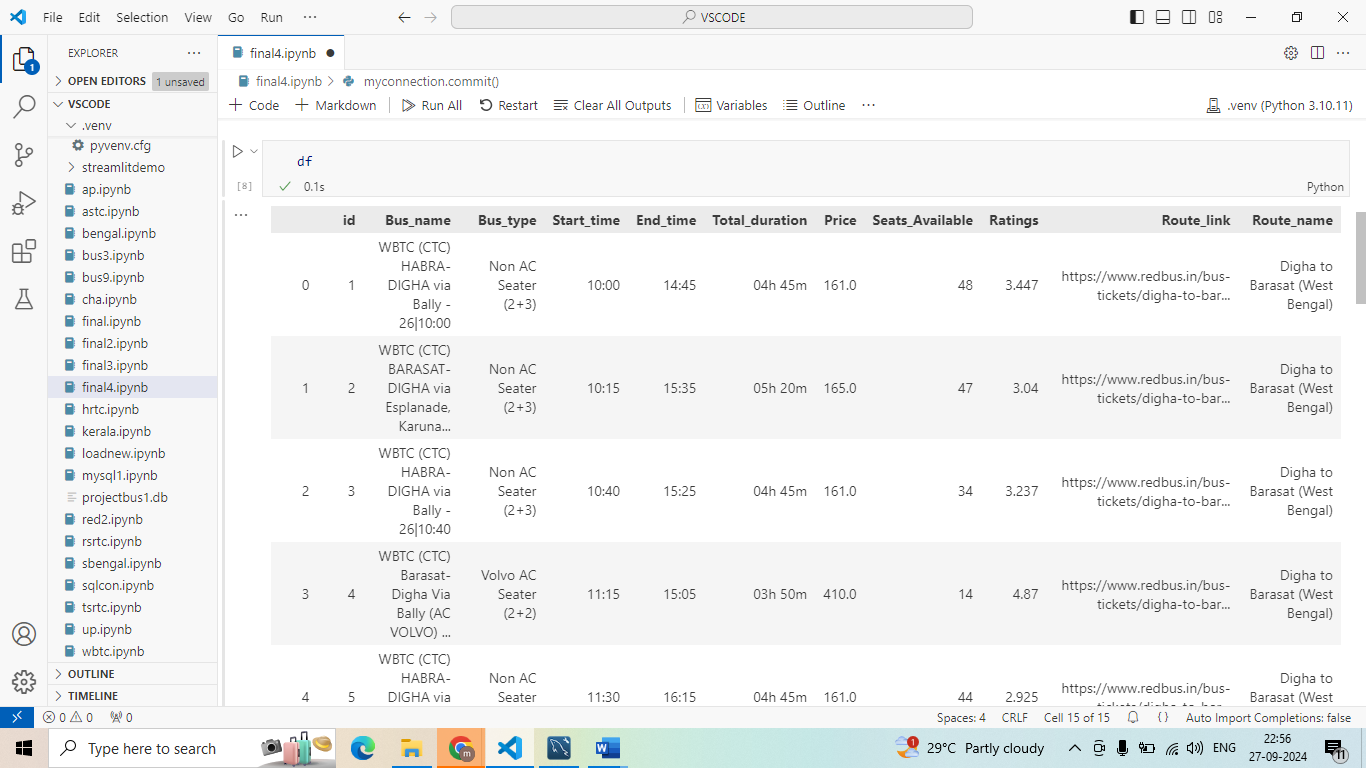
cursor.execute(sql, tuple(df.iloc[i]))

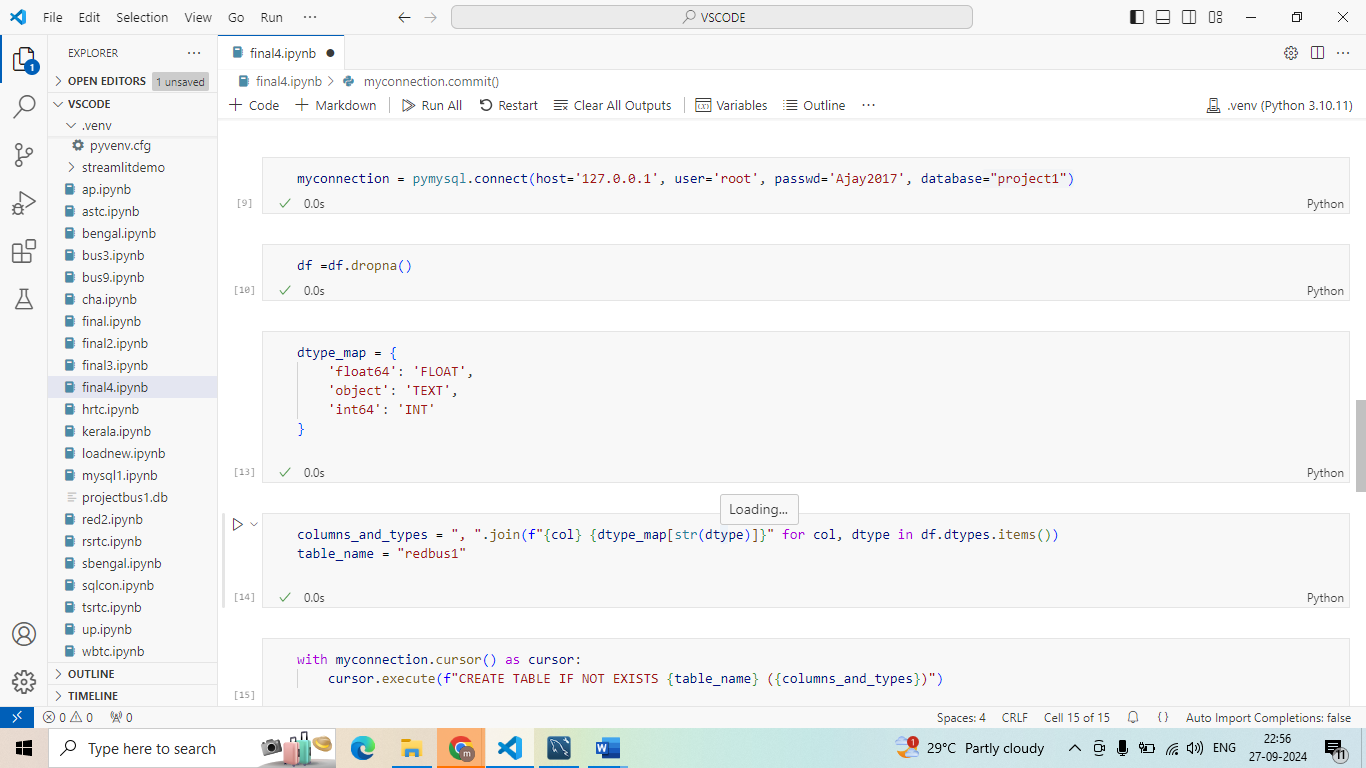
myconnection.commit()

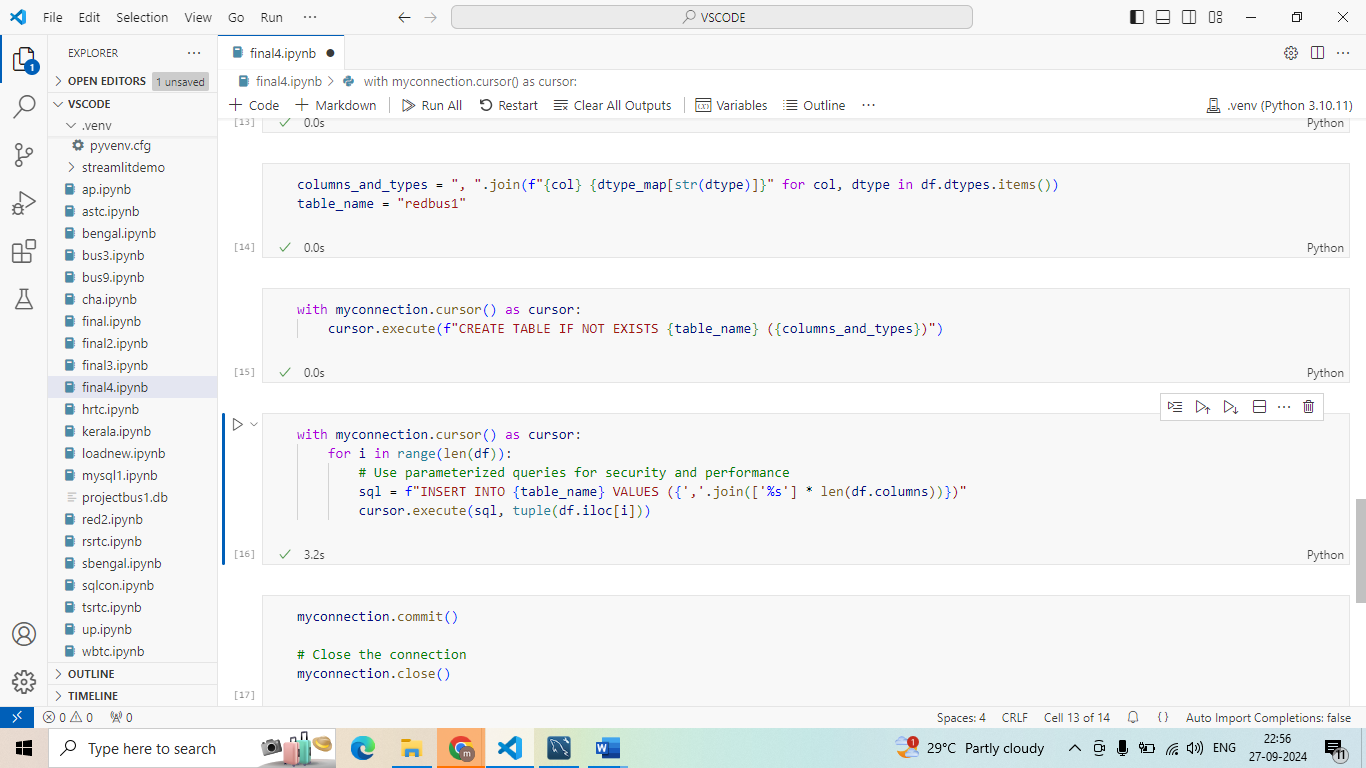
myconnection.close()











**deployment**

#streamlit connecton

import streamlit as st

import pymysql

import pandas as pd

# Connect to MySQL database

def get\_connection():

return pymysql.connect(host='127.0.0.1', user='root', passwd='Ajay2017', database='projectbus')

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

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

query = f"SELECT DISTINCT Route\_name FROM redbus WHERE Route\_name LIKE '{starting\_letter}%' ORDER BY Route\_name"

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

return Route\_name

# Function to fetch data from MySQL based on selected ROUTE\_NAME and price sort order

def fetch\_data(connection, Route\_name, price\_sort\_order):

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

query = f"SELECT \* FROM redbus WHERE Route\_name = %s ORDER BY Ratings DESC, PRICE {price\_sort\_order\_sql}"

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\_type):

filtered\_df = df[df['Ratings'].isin(ratings) & df['Bus\_type'].isin(Bus\_type)]

return filtered\_df

# Main Streamlit app

def main():

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')

# Fetch route names starting with the specified letter

if starting\_letter:

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

if Route\_name:

# Sidebar - Selectbox for ROUTE\_NAME

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

if selected\_route:

# Sidebar - Selectbox for sorting preference

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

# Fetch data based on selected ROUTE\_NAME and price sort order

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

if not data.empty:

# Display data table with a subheader

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

st.write(data)

# Filter by RATING and BUS\_TYPE

Ratings = data['Ratings'].unique().tolist()

selected\_ratings = st.multiselect('Filter by Ratings', Ratings)

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

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

if selected\_ratings and selected\_Bus\_type:

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

# Display filtered data table with a subheader

st.write(f"### Filtered Data for Rating: {selected\_ratings} and Bus\_type: {selected\_Bus\_type}")

st.write(filtered\_data)

else:

st.write(f"No data found for Route: {selected\_route} with the specified price sort order.")

else:

st.write("No routes found starting with the specified letter.")

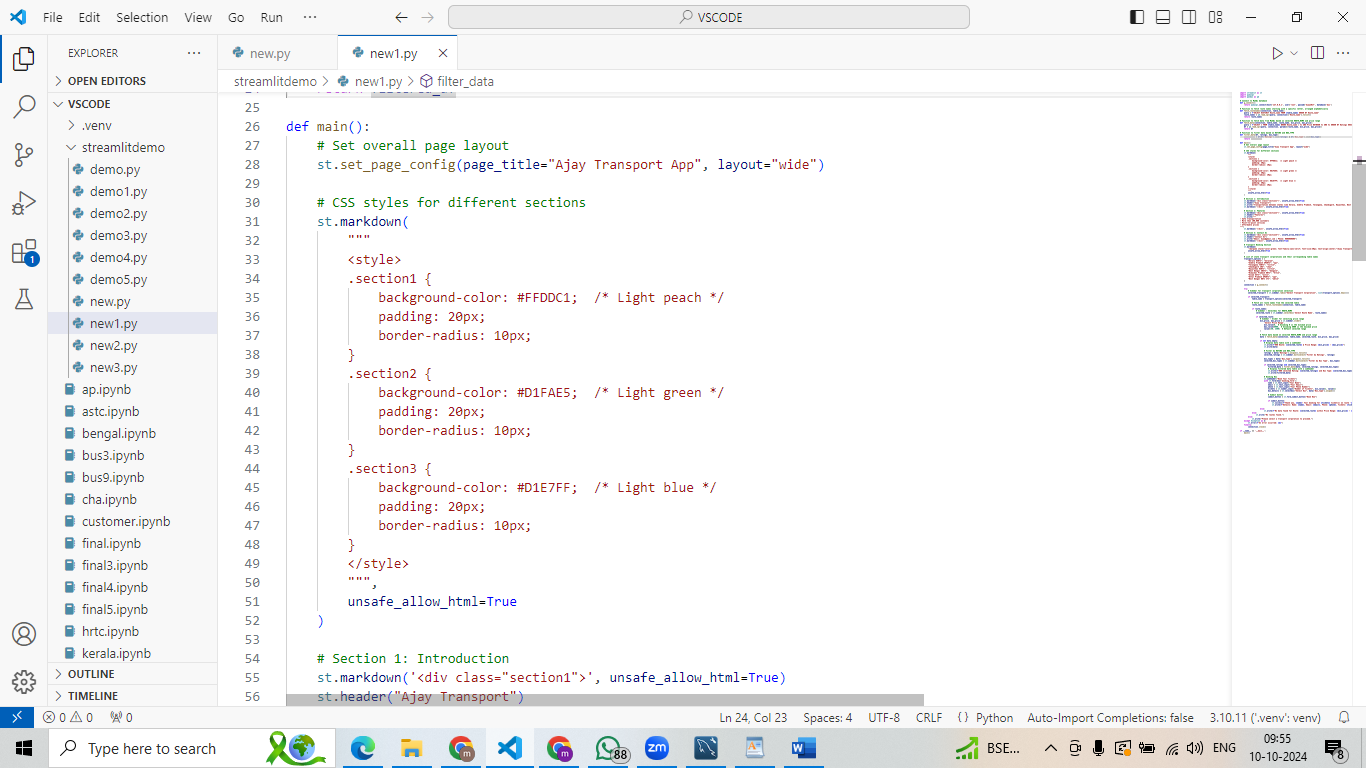
finally:

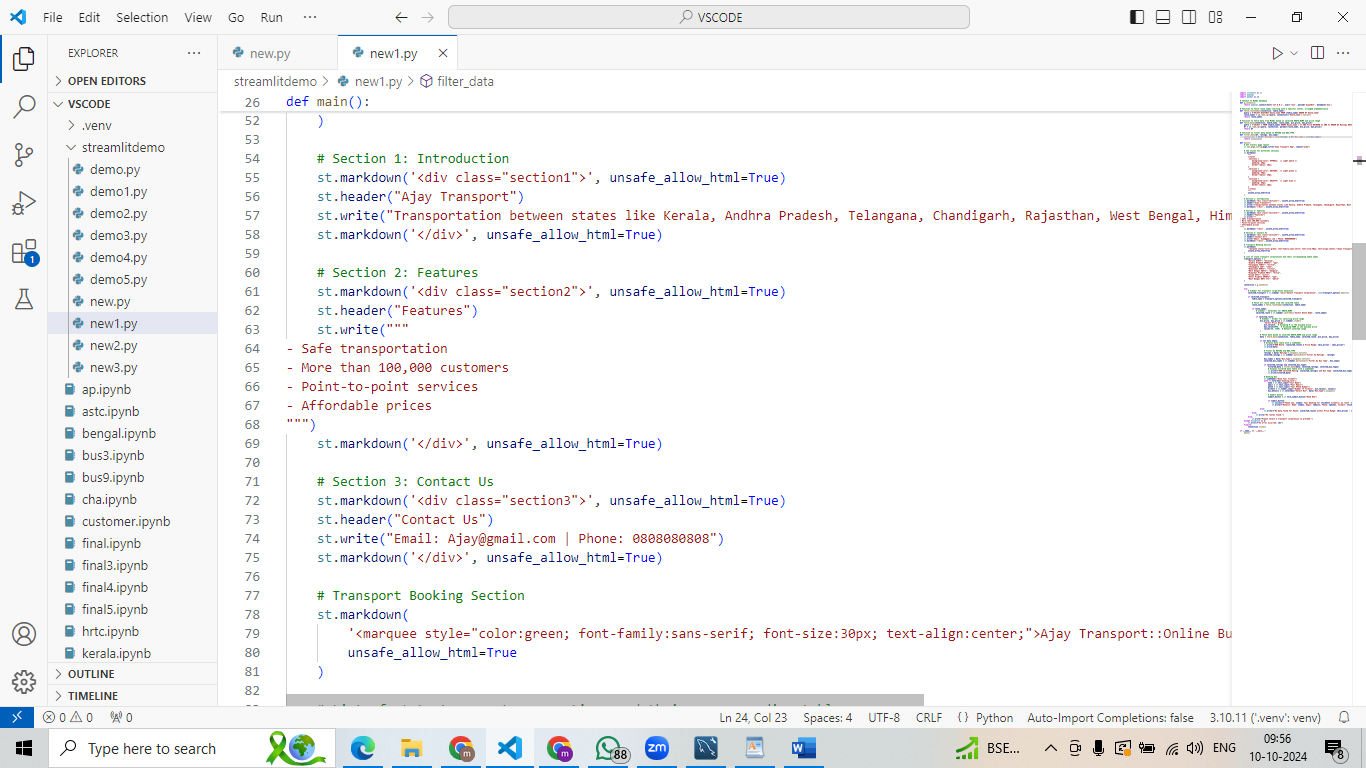
connection.close()

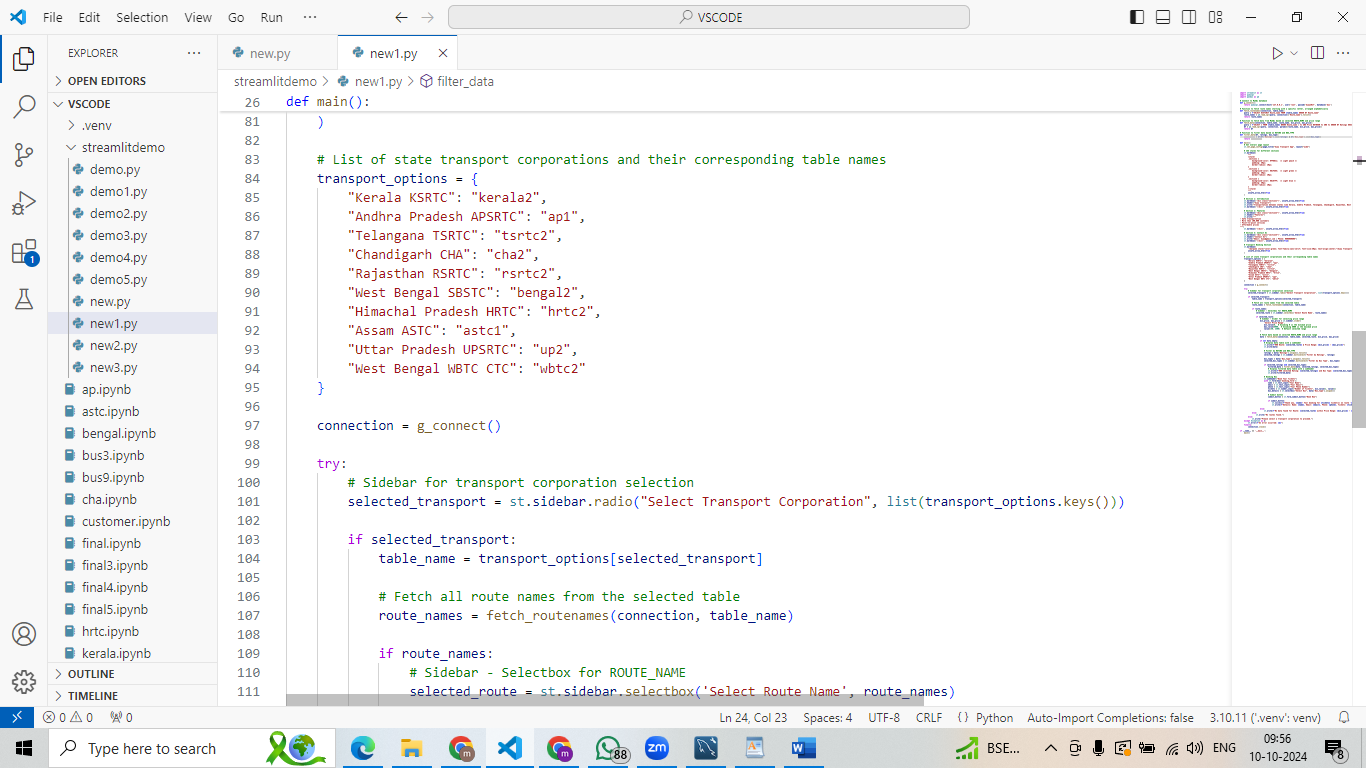
if \_\_name\_\_ == '\_\_main\_\_':

main()

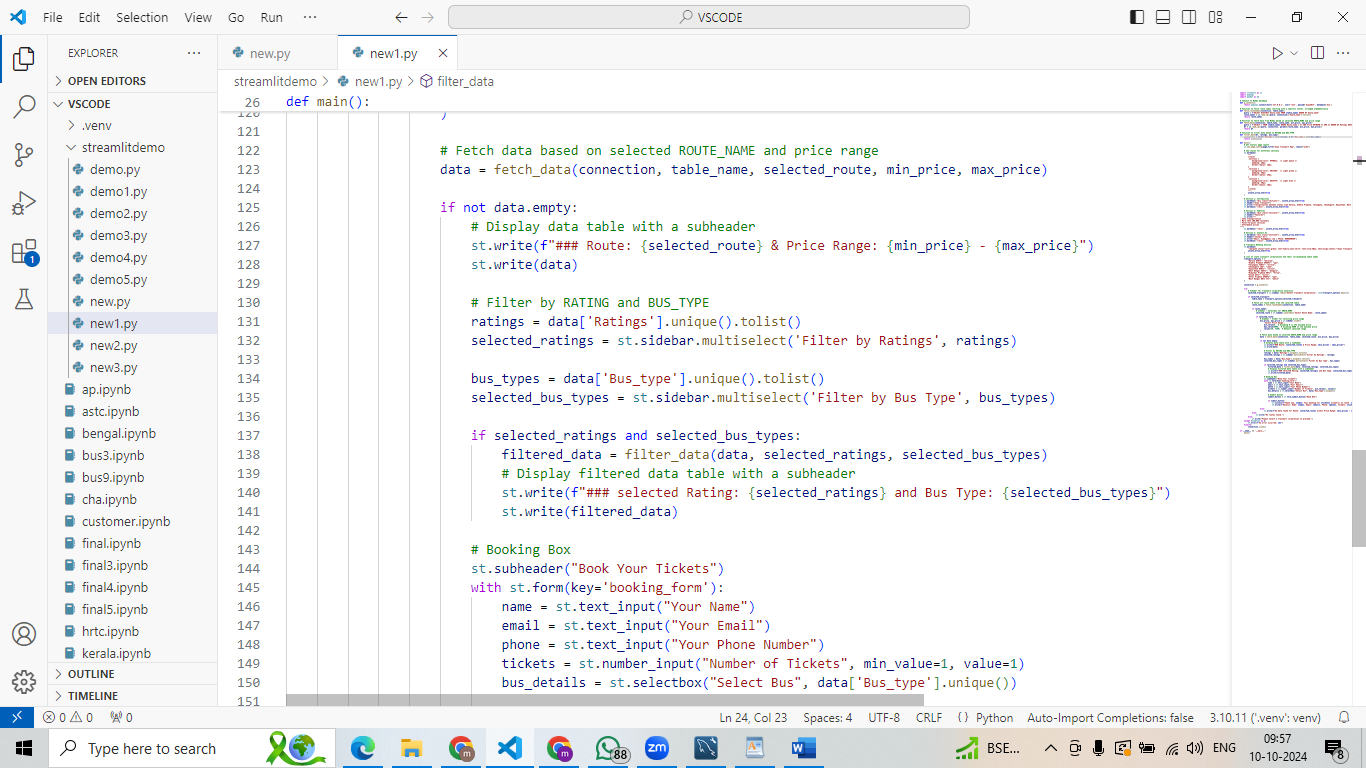


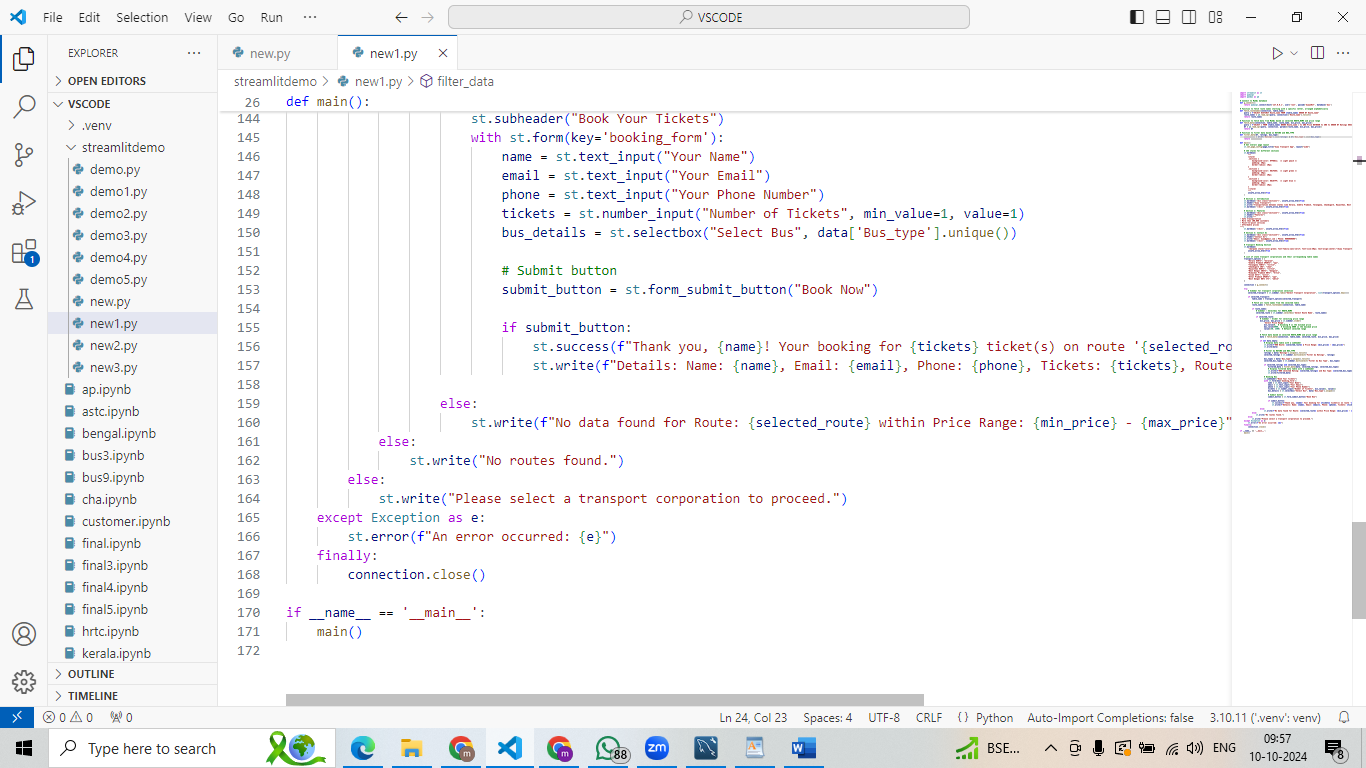




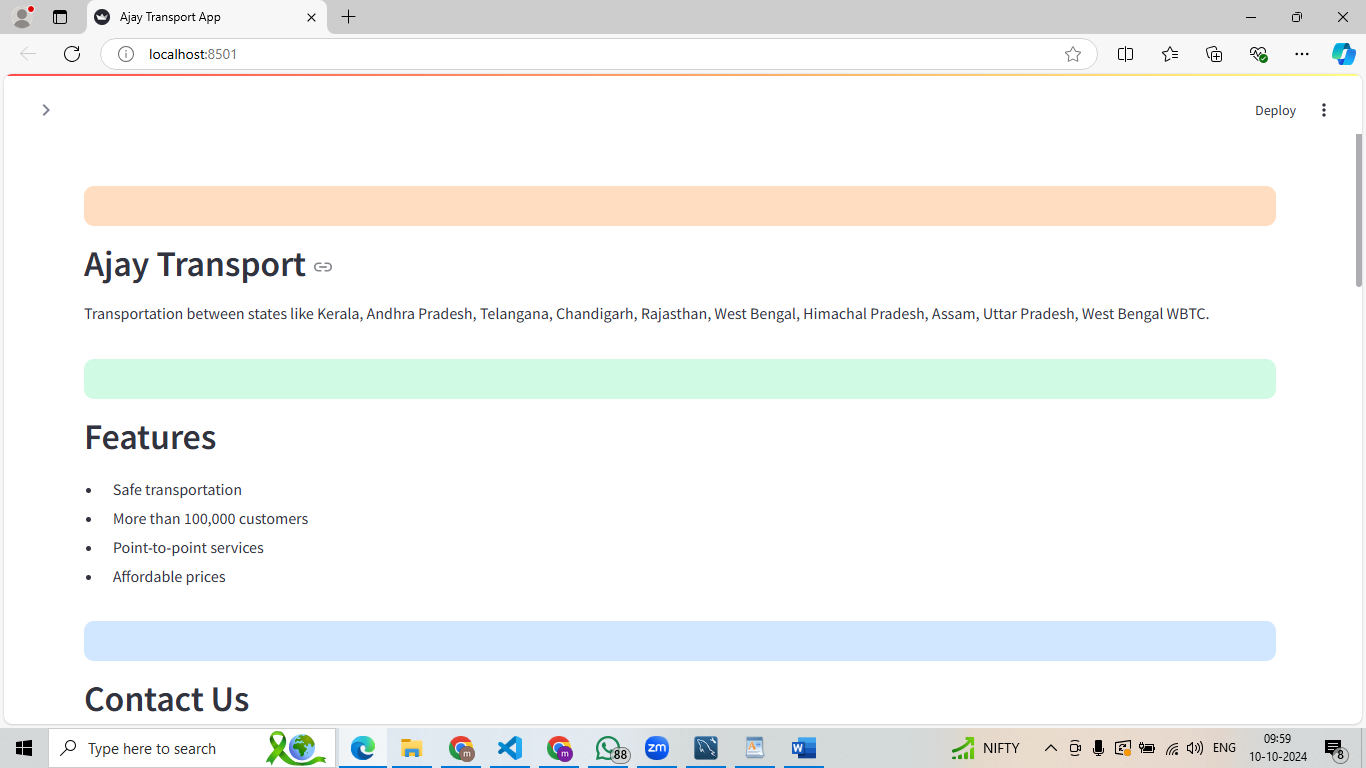


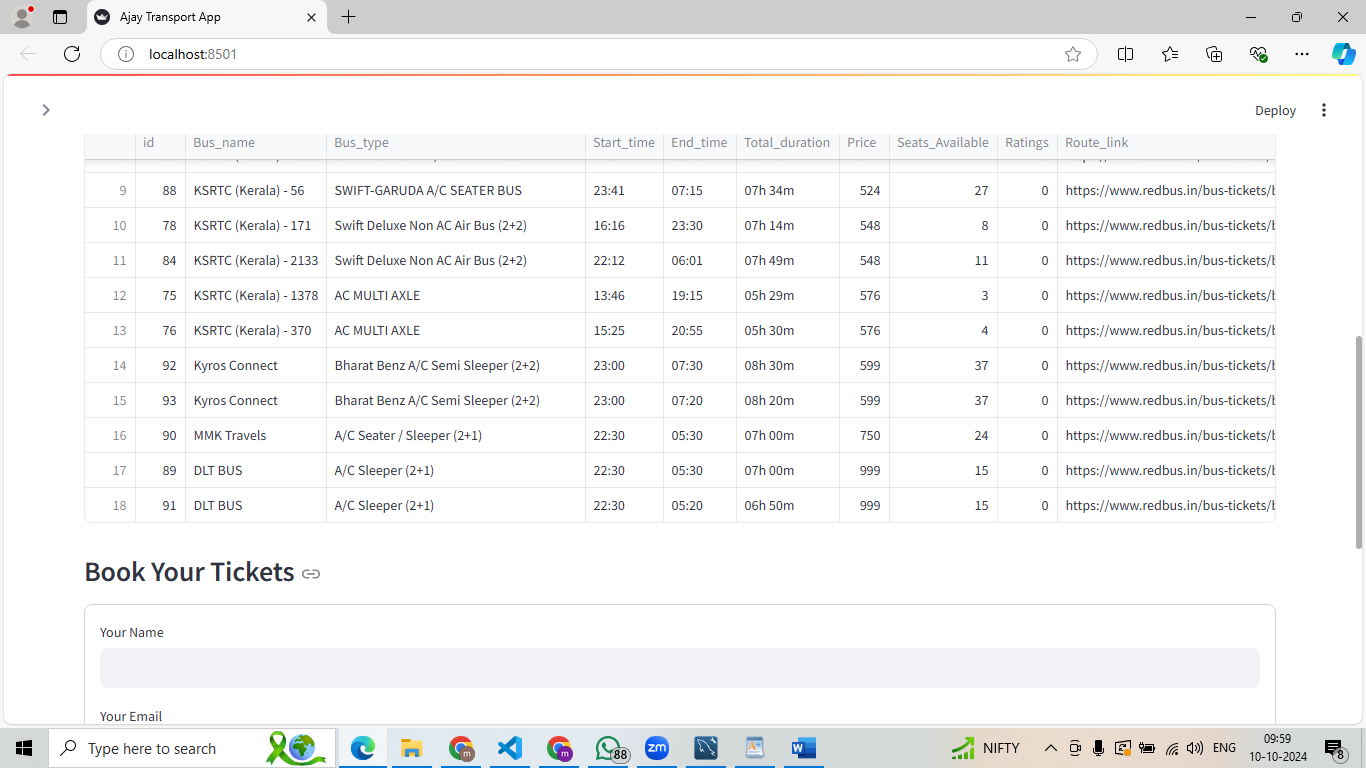


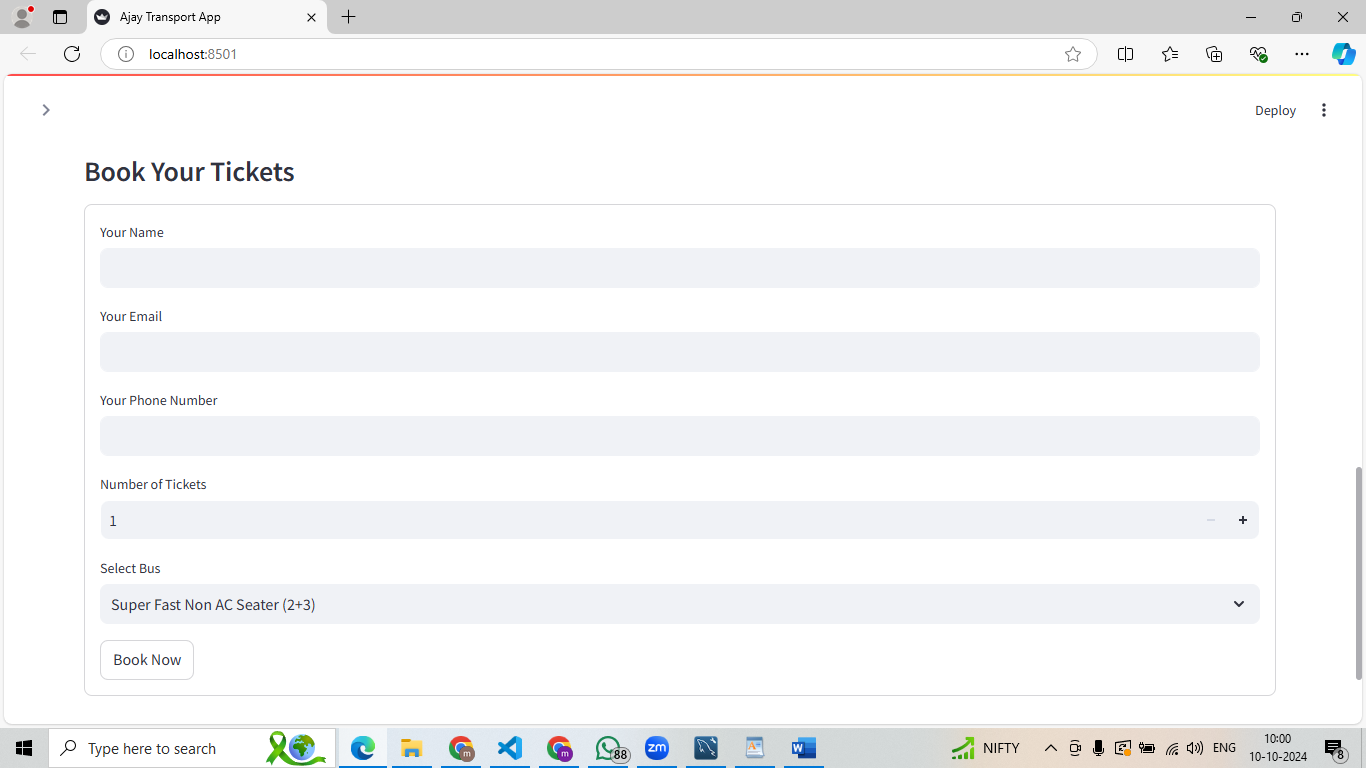


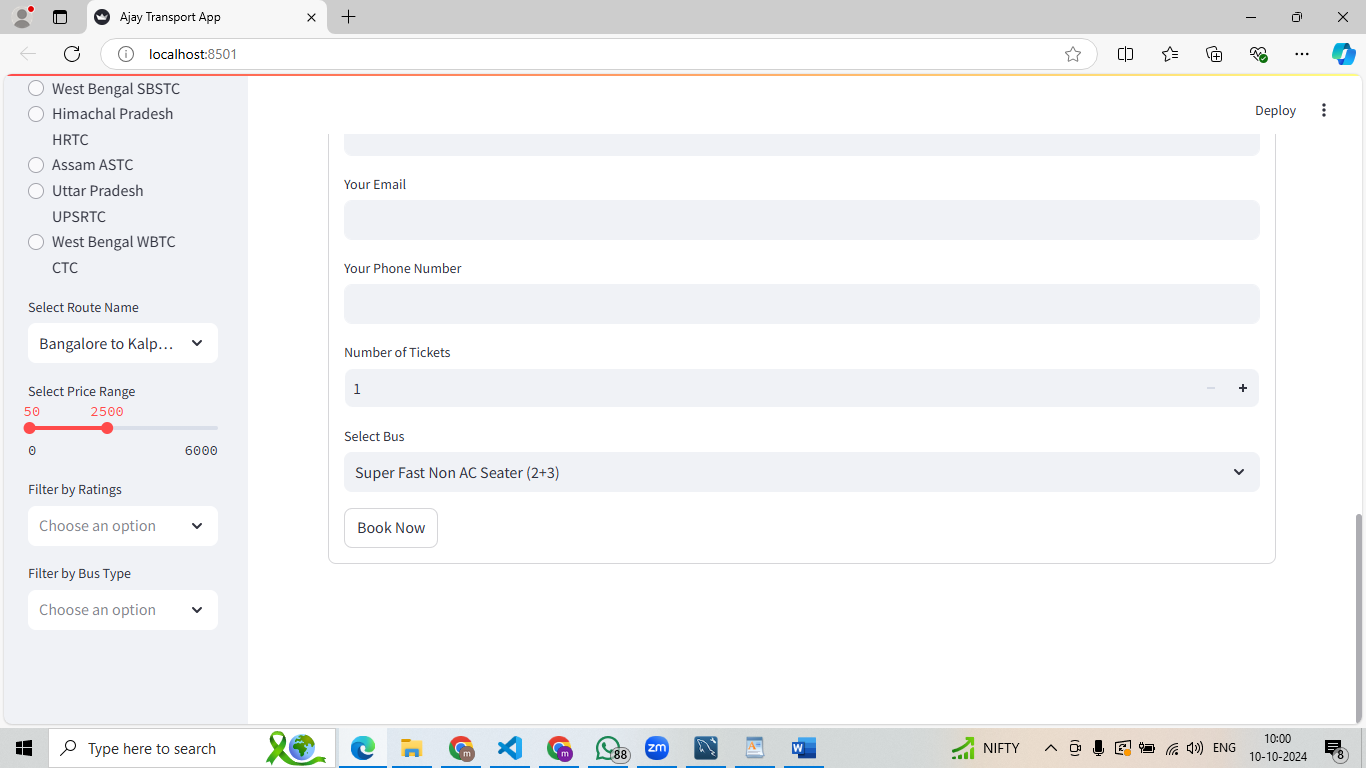


Output









**conclusion:**

**Summary of the Project:** The Redbus Streamlit application automates the scraping of bus travel data from Redbus and presents a user-friendly interface for filtering and analyzing information. Utilizing tools like Selenium, Streamlit, and SQL, the application is a helpful resource for travel aggregators, market researchers, and individuals planning bus journeys.

**Potential Impact:** The application offers several potential benefits to the transportation industry:

* **Improved User Experience:** It provides real-time, personalized information, enhancing the planning process for bus travelers.
* **Data-Driven Decision Making:** Businesses can leverage the application's insights to optimize operations and marketing strategies.
* **Increased Efficiency:** Automation of data scraping and analysis reduces manual work and enhances operational efficiency.

**Future Directions:** Potential future developments include:

* **Integration with Booking Platforms:** Offering direct booking functionality for a more seamless experience.
* **Real-time Alerts:** Providing updates on bus schedules or price changes to keep users informed.
* **Expansion to Other Transportation Modes:** Including data for other modes of transport like trains and flights to offer comprehensive travel planning.

With continued development, the Redbus Streamlit application could become a significant asset to the transportation industry, improving the travel experience for users.