

AUTOMATED FLIGHT DATA GATHERING SYSTEM

I. INTRODUCTION

The system generates a large amount of data related to flights, including departure and arrival times, and price. Traditionally, collecting this data involved manual data entry, relying on multiple sources, such as airline websites, airport websites, and third-party platforms. This method is not only time-consuming but also prone to human errors. Flight data is subject to frequent updates, making it difficult to maintain accuracy and timeliness manually. To solve this problem the system has used web scraping techniques to automatically extract flight-related information from various online sources, ensuring accuracy and real-time updates.

II. DATA COLLECTION

Building an Automated Flight Data Gathering System using web scraping involves collecting data from various sources on the internet. Here are some of the key data sources and the types of information I collected for the system:

Flight Booking Platforms:

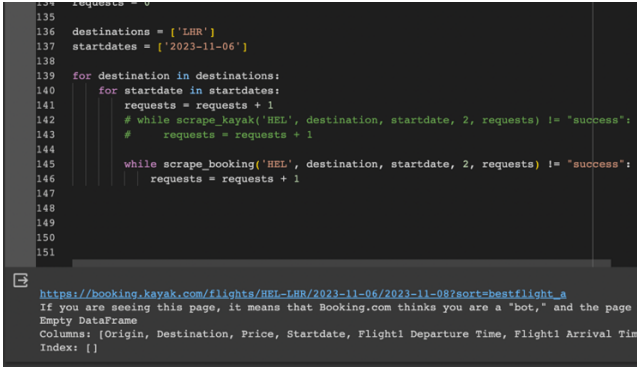
- Flight Schedules: Retrieve flight schedules, including departure and arrival times.
- Ticket Prices: Gather pricing information for different flight routes and seat classes.
- Departure and Destination

III. DATA ANALYSIS

I chose booking.com, Kayak and Skyscanner to create the system. All the three websites had some challenges to work on. But Kayak was easy to analyze because it didn't have many problems compared to the other websites.

Problems Faced:

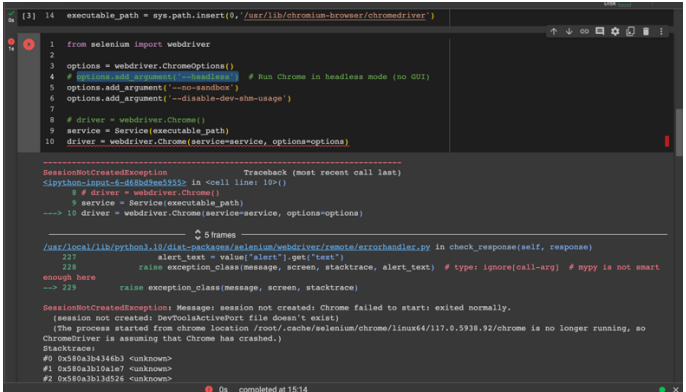
In Skyscanner there is a Captcha that blocks automated systems. Since I used Selenium, it detected the program as a bot, and I could not complete the analysis from that. Also booking.com needed to login the customer first to continue with the search. The below screenshot shows it was detected as a robot when I use google colab.



```
134 requests = 0
135
136 destinations = ['LHR']
137 startdates = ['2023-11-06']
138
139 for destination in destinations:
140     for startdate in startdates:
141         requests = requests + 1
142         # while scrape_kayak('HEL', destination, startdate, 2, requests) != "success":
143             requests = requests + 1
144         while scrape_booking('HEL', destination, startdate, 2, requests) != "success":
145             requests = requests + 1
146
147
148
149
150
151
```

https://booking.kayak.com/flights/HEL-LHR/2023-11-06/2023-11-08?sort=bestflight_a
If you are seeing this page, it means that Booking.com thinks you are a "bot," and the page is empty.
Columns: [Origin, Destination, Price, Startdate, Flight1 Departure Time, Flight1 Arrival Time]
Index: []

Also, the below error was occurred due to selenium and chrome driver in google colab even though the script was working perfectly in my local machine as a python scripts.



```
14 executable_path = sys.path.insert(0, '/usr/lib/chromium-browser/chromedriver')
15
16 from selenium import webdriver
17
18 options = webdriver.ChromeOptions()
19 # options.add_argument('--headless') # Run Chrome in headless mode (no GUI)
20 options.add_argument('--no-sandbox')
21 options.add_argument('--disable-dev-shm-usage')
22
23 driver = webdriver.Chrome()
24 service = Service(executable_path)
25 driver = webdriver.Chrome(service=service, options=options)
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
```

SessionNotCreatedException Traceback (most recent call last)
File ~/lib/python3.10/site-packages/selenium/webdriver/remote/executor.py in check_response(self, response)
227 alert_text = value["alert"].get("text")
228 raise exception_class(message, screen, stacktrace, alert_text) # type: ignore[call-arg] # mypy is not smart enough here
--> 229 raise exception_class(message, screen, stacktrace)
SessionNotCreatedException: Message: session not created: Chrome failed to start: exited normally.
(session not created: DevToolsActivePort file does not exist)
[The process started from chrome location /root/.cache/selenium/chrome/linux64/117.0.5938.92/chrome is no longer running, so ChromeDriver is assuming that Chrome has crashed.]
Stacktrace:
#0 0x540a1b1440: <unknown>
#1 0x540a1b101e: <unknown>
#2 0x540a1b13d52e: <unknown>

For google colab then I used requests library instead of Selenium with below code.

```
headers = {'User-Agent': 'Mozilla/5.0
(Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/58.0.3029.110 Safari/537.3'}
```

```
response = r.get(url, headers=headers)
```

Then I was able to find a solution for the captcha issue using the direct url with destination, origin and the date. First, I loaded the website using Selenium and I used BeautifulSoup to scrape the result data from the website. Then I used lists for each variable I needed and then created a Data Frame using the List. Finally, that data frame was converted into a CSV to store data. The program was developed using Visual Studio Code and program can be run using the command “python main.py”.

This is Data frame I got as a result. It Includes Origin, Destination, Flight times and Price.

The final outputs are Booking.csv, Kayak.csv and Skyscanner.csv.

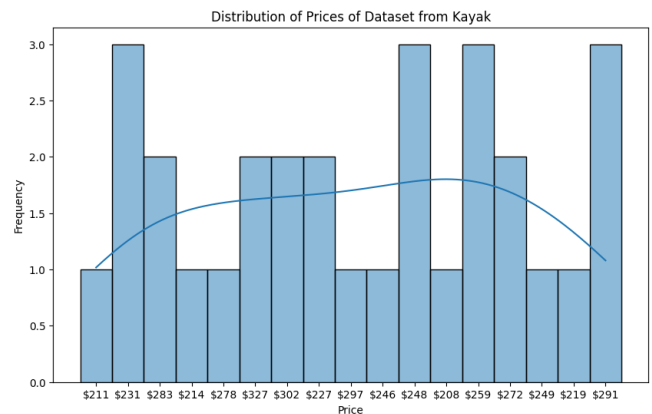
Kayak						
Origin	Destination	Price	Startdate	Flight1 Time	Flight2 Time	Enddate
HEL	LHR	\$211	2023-11-05	7:50 am – 9:10 am	7:30 am – 12:20 pm	2023-11-07
HEL	LHR	\$231	2023-11-05	2:00 pm – 3:15 pm	7:30 am – 12:20 pm	2023-11-07
HEL	LHR	\$283	2023-11-05	7:50 am – 9:10 am	4:15 pm – 9:05 pm	2023-11-07
HEL	LHR	\$214	2023-11-05	7:35 pm – 8:40 pm	7:30 am – 12:20 pm	2023-11-07
HEL	LHR	\$278	2023-11-05	11:20 am – 12:35 pm	7:30 am – 12:20 pm	2023-11-07
HEL	LHR	\$231	2023-11-05	4:00 pm – 5:00 pm	7:30 am – 12:20 pm	2023-11-07
HEL	LHR	\$283	2023-11-05	7:35 pm – 8:40 pm	4:15 pm – 9:05 pm	2023-11-07
HEL	LHR	\$327	2023-11-05	7:35 pm – 8:40 pm	7:30 am – 12:20 pm	2023-11-07
HEL	LHR	\$302	2023-11-05	4:00 pm – 5:00 pm	4:15 pm – 9:05 pm	2023-11-07
HEL	LHR	\$227	2023-11-05	7:35 pm – 8:40 pm	6:10 pm – 11:00 pm	2023-11-07
HEL	LHR	\$297	2023-11-05	11:20 am – 12:35 pm	6:10 pm – 11:00 pm	2023-11-07
HEL	LHR	\$327	2023-11-05	7:50 am – 9:10 am	7:30 am – 12:20 pm	2023-11-07
HEL	LHR	\$227	2023-11-05	7:50 am – 9:10 am	6:10 pm – 11:00 pm	2023-11-07
HEL	LHR	\$302	2023-11-05	2:00 pm – 3:15 pm	4:15 pm – 9:05 pm	2023-11-07
HEL	LHR	\$246	2023-11-05	4:00 pm – 5:00 pm	6:10 pm – 11:00 pm	2023-11-07
HEL	LHR	\$248	2023-11-06	7:50 am – 9:10 am	7:30 am – 12:20 pm	2023-11-08
HEL	LHR	\$208	2023-11-06	7:35 pm – 8:40 pm	7:30 am – 12:20 pm	2023-11-08
HEL	LHR	\$259	2023-11-06	7:50 am – 9:10 am	4:15 pm – 9:05 pm	2023-11-08
HEL	LHR	\$248	2023-11-06	2:00 pm – 3:15 pm	7:30 am – 12:20 pm	2023-11-08
HEL	LHR	\$259	2023-11-06	2:00 pm – 3:15 pm	4:15 pm – 9:05 pm	2023-11-08
HEL	LHR	\$272	2023-11-06	7:50 am – 9:10 am	6:10 pm – 11:00 pm	2023-11-08
HEL	LHR	\$249	2023-11-06	7:35 pm – 8:40 pm	10:20 am – 3:15 pm	2023-11-08
HEL	LHR	\$219	2023-11-06	7:35 pm – 8:40 pm	4:15 pm – 9:05 pm	2023-11-08
HEL	LHR	\$291	2023-11-06	7:50 am – 9:10 am	10:20 am – 3:15 pm	2023-11-08

IV. DATA VISUALIZATION

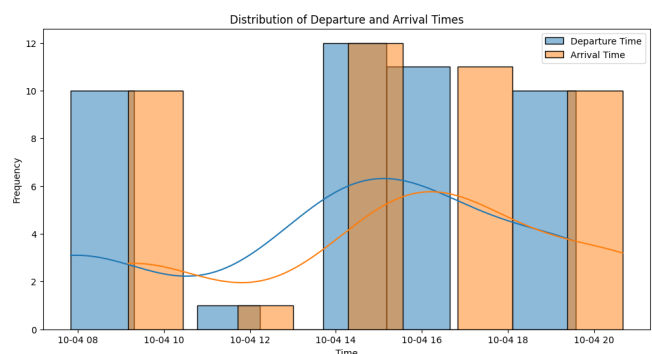
To visualize the data I used seaborn, matplotlib, pandas libraries and the data sets I selected was Kayak.csv and Skyscanner.csv.

I visualize them using,

Price ranges



Duration of the flight



CONCLUSION

As for my understanding python scripts are a better option for developing than using Google Colab. Even though Google Colab offers a faster way to run the script as a whole program like this the best option is to use an IDE like Visual Studio Code. But when it comes to for the visualization and data analysis Colab is a good platform.

Python web scraping offers a robust solution for collecting flight details. By automating data

gathering, providing real-time updates, and offering customization options, a well-designed flight data scraping system can significantly

enhance the user experience and support informed decision-making.