Mini Project 1: Where do I fly next?

Naimur Rahman

1 Introduction

Given a specific travel destination and a one-way flight date, my goal is to find the most favorable deal for a trip from Helsinki to Paris. Popular travel websites like kayak.com, momondo.com, expedia.com, skyscanner.com, booking.com, and others offer a wealth of information on available flights, including details like pricing, locations, layover times, and layover locations. However, having a systematic method to compare flight offers from these different platforms all at once would be highly beneficial.

To address this challenge, I plan to collect data from these websites (kayak.com, momondo.com, expedia.com) and apply data processing techniques to obtain the desired results. The data will be gathered through web scraping, a method involving the extraction of data from websites by leveraging their HTML structures or APIs. While APIs are not always accessible and may require purchase, web scraping can often serve as a viable solution. Through these means, I aim to consolidate all the relevant flight data in one place, allowing for comprehensive comparisons and ultimately presenting users with the best available deals.

2 Data Collection

I have acquired data from three distinct websites through web scraping. These websites are:

www.kayak.com
www.momondo.com
www.expedia.com

Table 1. Scraped websites

In order to obtain the data, I initiated the process with specific predefined parameters, essentially simulating a user's preferences. These parameters include:

- **Destination:** Paris, France
- **Departure Date:** October 28, 2023
- **Type of Trip:** One-way
- **Number of Passengers:** 1 adult

I employed these parameters to conduct web scraping on the three websites, resulting in the retrieval of extensive flight data. This data includes numerous features or columns such as arrival times, departure times, the number of stops, prices, flight durations, airline information, layover details, the layover city, and the source website's name. Each website yielded data for a minimum of 50 flights, accumulating a total of 308 records for flights scheduled on October 28th from all three websites combined.

The primary tools I utilized for this scraping process were:

- Selenium
- Selenium-Stealth

Before commencing the scraping procedure, it's crucial to adhere to community standards established by these websites. None of the websites I used permit the utilization of their data for commercial purposes. However, scraping for personal testing and analysis is generally allowed. To retrieve the necessary information,

I employed both class names and XPath. XPath, due to its ease of use and faster coding, emerged as the preferred method for me to locate web elements.

To manage the collected data efficiently, I performed the scraping separately for each website, as processing all the data in a single Colab file was time-consuming. Subsequently, I exported the datasets into CSV files for all three websites. These datasets were later merged for preprocessing, exploratory data analysis (EDA), and to facilitate user interaction options.

In essence, I've created four distinct Jupyter Notebook files along with three corresponding CSV files for data processing purposes. Here's the breakdown of these files:

- **kayak.ipynb**
- **momondo.ipynb**
- **expedia.ipynb**
- **eda.ipynb**
- **kayak.csv**
- **momondo.csv**
- **expedia.csv**

To acquire the CSV files (kayak.csv, momondo.csv, expedia.csv), one should execute kayak.ipynb, momondo.ipynb, and expedia.ipynb separately. Subsequently, for exploratory data analysis (EDA) and user interaction capabilities, running eda.ipynb is necessary. Within eda.ipynb, one can find a range of visualizations and interactive elements.

In total, I have obtained eight distinct variables from these datasets, encompassing six quantitative variables and two categorical variables.

3 Data Presentation

I had to preprocess the data to facilitate further analysis. The CSV files I gathered originate from three distinct websites, each with its own unique data format. To ensure a meaningful comparison, it was crucial to standardize the data. Therefore, I formatted them individually based on their specific requirements. Once these datasets were uniformly shaped, I merged them together.

For instance, the arrival and departure times initially appeared in AM/PM format, but for expedia.com, I needed to extract this information from lengthy text strings. The "stops" column was originally in string format, but I converted it to integers to simplify analysis. Some missing values were encountered in the airline names and layover times columns, and I replaced them with "none" instead of "NaN" or empty strings. Additionally, any trailing white spaces were removed as needed.

4 Data Analysis

In the data analysis phase, I began by examining the numerical data. The average flight price was found to be approximately 450.58 euros, and the mean duration was 9 hours and 3 minutes. On average, flights had a duration of 8 hours, with an average layover duration of around 2 hours. The visual representations of these findings are displayed below.

	stops	prices	duration	flight_duration	layover_duration	Hour
count	308.000000	308.000000	308	308.000000	308.000000	308.000000
mean	1.464286	450.581169	0 days 09:03:02.727272727	8.607143	2.733766	10.428571
std	0.616302	295.822273	0 days 03:42:41.574946311	3.752725	2.904657	3.989983
min	0.000000	96.000000	0 days 03:05:00	3.000000	0.000000	5.000000
25%	1.000000	265.000000	0 days 06:32:30	6.000000	1.000000	7.000000
50%	2.000000	403.500000	0 days 08:55:00	8.000000	2.000000	10.500000
75%	2.000000	569.250000	0 days 11:10:00	11.000000	4.000000	13.000000
max	3.000000	2972.000000	1 days 14:25:00	38.000000	20.000000	22.000000

Fig 1. Data analysis on numeric data

Next, I generated various plots using two distinct libraries, namely:

- Matplotlib
- Seaborn

These plots encompassed a variety of chart types, including:

- Bar charts
- Pie charts
- Scatter plots

An interesting observation from the scatter plot was

that the majority of layovers took place in the evening.

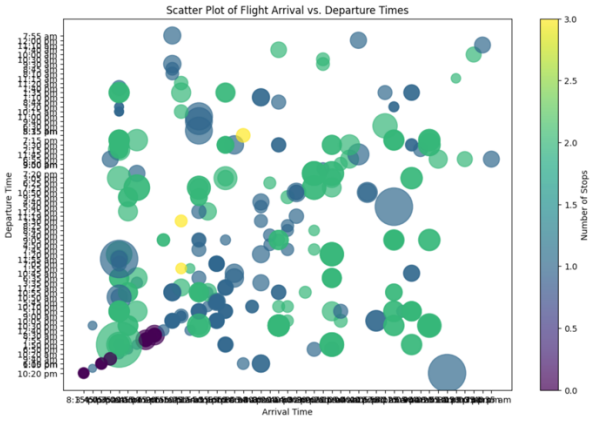

Fig 2. Compare arrival, departure time with the number of stops using scatter plot

Among all the airlines, Lufthansa stands out with the highest number of layovers. Conversely, combined airlines have notably fewer layovers, which is logical since an airline company has more flexibility in forming partnerships with multiple airlines instead of customizing its own schedule.

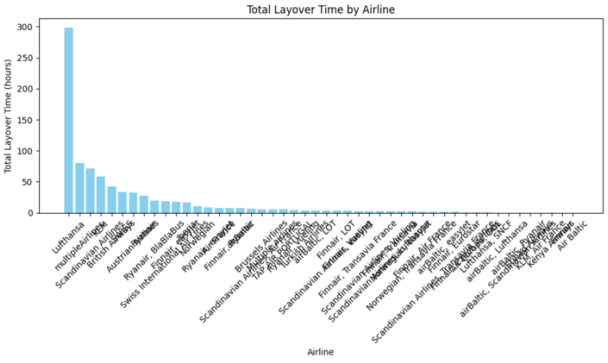

Fig 3. Compare total layover time with every flight

On average, RyanAir and Austrian Airways have the lengthiest flight durations, whereas AirFrance, British Airways, and KLM boast the shortest ones.

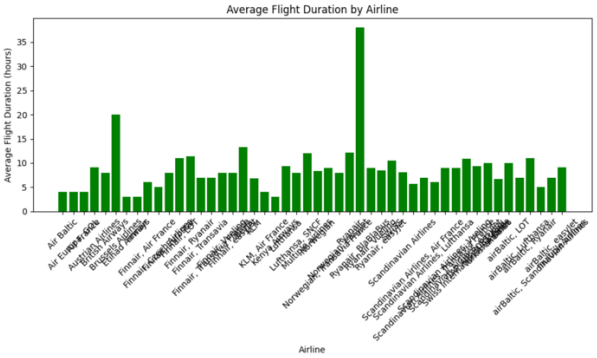

Fig 4. Compare travel duration for every flight

RyanAir and EasyJet are the priciest options, which might come as a surprise considering that a single RyanAir flight is the most budget-friendly choice in terms of cost.

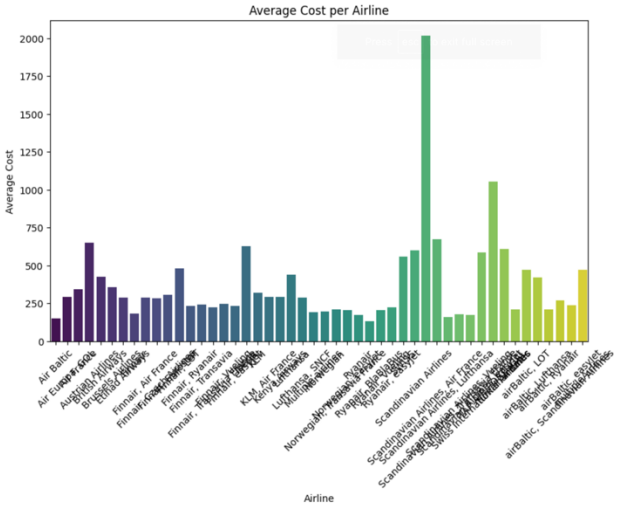

Fig 5. Average cost for a flight

Lufthansa and Scandinavian Airlines emerge as the top-ranking airlines in terms of the highest number of flights operated consecutively. On the other hand, the location where the majority of layovers take place is Amsterdam, Netherlands.

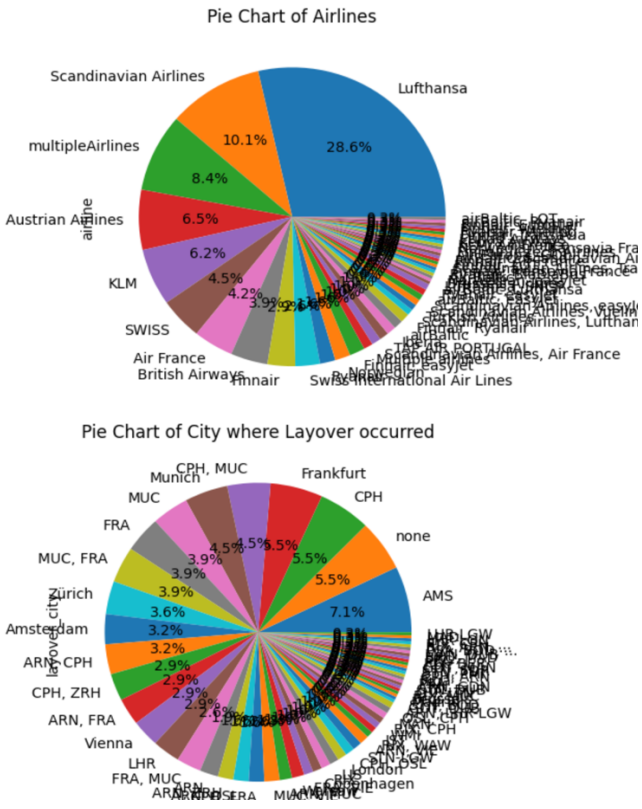

Fig 6. Pie chart to show the amount for every airlines, and show the frequency of places where layover occurred

During the morning hours, there is a higher frequency of flights compared to the period after evening.

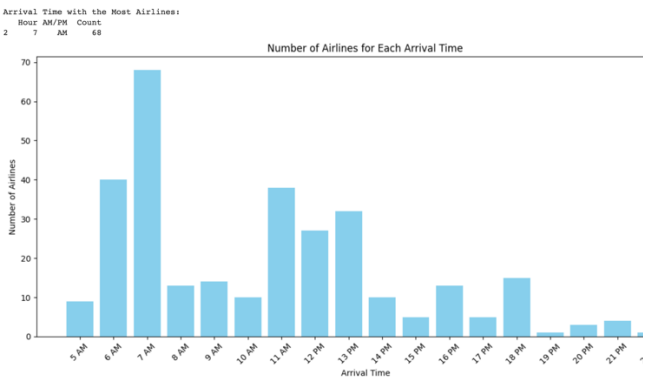

Fig 7. Frequency of flights in every hour (in a day)

place. Below is a snippet exemplifying the cheapest and fastest flight options.

```
Cheapest Flight:
Airline: Ryanair
Total Duration: 0 days 22:20:00
Direct Flight: No
Layover City: WMI
Layover Time: 18h 15m
Aircraft Type: Ryanair
Website Name: www.kayak.com
Total Cost (in euro): 96

Fastest Flight:
Airline: Finnair
Total Duration: 0 days 03:05:00
Direct Flight: Yes
Aircraft Type: Finnair
Website Name: www.kayak.com
Total Cost (in euro): 168
```

Fig 8. Output for user inputs

5 User Interaction

To engage with the user effectively, I've devised an interactive filtering system that relies on four types of user input:

1. Price Range: The user specifies the desired price range for the flight.

2. Maximum Trip Duration: The user sets a maximum allowable trip duration.

3. Maximum Preferred Stops: Users can indicate the maximum number of preferred stops for their flight.

4. Sorting Criteria: Users can select whether they want the results sorted by price or trip duration.

Based on these inputs, I provide three key pieces of information:

1. Sorted Dataframe: I present a dataframe sorted according to the user's chosen criteria.

2. Cheapest Flight: I identify and display the cheapest available flight meeting the user's criteria.

3. Fastest Flight: I pinpoint and showcase the fastest available flight fitting the user's requirements.

Additionally, in cases where there are layovers during the flight, I offer details on the layover duration and the location where the layover takes

6 Conclusion

Web scraping serves as a valuable tool for extracting pertinent information from various websites. However, in today's world, where data holds immense value, many websites have implemented measures to thwart web scraping activities. Take, for instance, the case of Selenium, which can effectively scrape data during the initial interaction with a website. However, with subsequent attempts, it often encounters issues and may even crash. To address this challenge, I employed Selenium-stealth, a solution that allows users to scrap multiple times.

In addition to the mentioned websites, I also attempted to scrape data from other platforms like booking.com and skyscanner.com. However, these websites did not consistently provide stable responses, leading me to exclude them from the project. Furthermore, I encountered issues when attempting to scrape data from all the websites simultaneously. The process was time-consuming and prone to instability, prompting me to adopt a strategy where I scraped data from different websites separately.

It is worth noting that different websites structure their data in distinct ways. To ensure uniformity across the dataset, I had to implement various data processing functions.

Another significant challenge was handling lazy loading, a technique employed by some websites to retrieve data from the backend. Lazy loading introduces delays in data retrieval. To mitigate this issue, I leveraged explicit wait functionality from the Selenium package. This allowed me to pause execution until the required elements had fully loaded, ensuring the accurate capture of data.

Throughout this project, I encountered and successfully addressed various issues commonly encountered in web scraping. It provided an excellent opportunity to gain hands-on experience in data processing, exploratory data analysis (EDA), and user interaction with real-world data.