Mini Project 1 – Where do I fly next?

Ali Kaya
Åbo Akademi University (ÅAU)
ali.kaya@abo.fi

October 2, 2023

Abstract

The aviation industry is a dynamic and ever-evolving sector, with travelers constantly seeking the most convenient and cost-effective flight options. This report presents a systematic way for Exploratory Data Analysis (EDA) of flight data obtained from OTA(Online Travel Agent) websites like Expedia, Momondo and Kyayk using Python.

In this study, I leveraged web scraping techniques to collect comprehensive flight information, including prices, airlines, departure times, arriving times, layover cities and their corresponding layover times, aircraft types, trip durations and more, from those OTA sources. The collected dataset was cleaned and preprocessed to ensure data quality and consistency. Python libraries such as selenium_stealth and BeautifulSoup were employed for web scraping, while Pandas, Seaborn, Matplotlib were utilized for data manipulation and visualization. Finally, Ipywidgets were used to realize real-time interactions with end users.

1 Introduction

Flight booking websites have become an essential tool for travelers, providing a convenient way to compare prices and book flights from a variety of airlines. However, these websites can be overwhelming, with users often having to sift through thousands of results to find the best flight for their needs.

In order to tackle this problem, in this report, different techniques were employed to effectively and efficiently present the steps for helping travelers to make comprehensive conclusion in regards to choosing the ideal flight. At the very first step, acquiring consistent and instant data in a good format should be well considered. Although there are plenty of APIs nowadays offered by OTAs for easy data access purpose, it costs a lot. To eliminate the monetary cost, I utilize the Python library selenium_stealth to scrape flight data from OTAs and BeautifulSoup for data extraction. The entire process is tedious but worthy, since running the scrapping codes could get exact real-time data just like browsing the corresponding OTA website.

After saving dataset into the form of DataFrame with Pandas Library, the succeeding task would be data cleaning and preprocessing. Some of the data should be converted to numerical data and ordinal data, while the others should be splitted or combined in accordance with its nature of use.

The next step is the Exploratory Data Analysis (EDA) of flight data by using various libraries, e.g., Seaborn and Matplotlib for plotting, Re and Pytimeparse for string conversion, etc., which play a vital role in this report. The goal of the EDA is to identify patterns and trends in the flight data, and to gain insights that could be further used to enhance user experience in the following user interaction part.

Last but not least, in order to help the users to quickly choose the ideal flights based on criteria like cheapest or fastest with other constrains, a concurrent interacting widget was developed with the help library Ipywidgets.

2 Data collection

2.1 Data Understanding

2.1.1 Data Selection

The objective of this project is to meticulously analyze flight information in order to provide a comprehensive understanding to the end user. To achieve this, careful consideration must be given to balancing the quantity of data elements that need to be extracted and the ease with which they can be obtained. In this project, I have defined the following data elements to be extracted:

- 'Price': This pertains to the cost of a single flight ticket for a given trip.
- 'No of Stop': This refers to the count of stops involved in a single trip.
- 'Airline': This encompasses the various airline companies involved in operating the flights,

which may include multiple layovers, for a single trip.

- ’Departure Time’: This indicates the precise departure time for each single trip.
- ’Arriving Time’: This signifies the exact arrival time for each single trip.
- ’Is Direct Flight’: This specifically denotes whether the flight is a direct one for the single trip.
- ’Layover Cities’: This enumerates the cities where layovers occur during a single trip.
- ’Separated Layover Time in Hours’: This provides detailed information about the duration of each layover during a single trip, measured in hours.
- ’Total Trip Duration in Hours’: This denotes the overall duration of the entire trip for a single journey, measured in hours.
- ’Type of Aircraft’: This refers to the type of aircraft or aircrafts (in the case of multiple layovers) used for a single trip.
- ’Source of Info’: This crucially points to the origin of the information for each single trip.

With these meticulously defined data elements, I aim to ensure the utmost accuracy and clarity in the flight information analysis for the benefit of end users.

2.1.2 Data Structure in the Website

As the aforementioned data elements will ultimately be extracted from OTA websites, this project focuses on scraping flight information from Helsinki to Paris on November 15, 2023. It is imperative to thoroughly analyze the layout representations of this data to facilitate the preparation of scraping source codes.

In the case of Expedia, all data elements, except for aircraft information, can be located on the search result page. They are encapsulated within an unordered (bulleted) list, utilizing the <ul> and <li> HTML tags. To retrieve all flight information from the search result page, a "show more" button must be continuously clicked until it disappears, indicating that all flight information has been fully loaded. Notably, the aircraft type is situated within the details of a pop-up window, which presents a challenge as the content of the pop-up window changes when opting for other flights, complicating the scraping process.

Conversely, in Momondo and Kayak, all data elements, excluding aircraft details, are accessible on the search result page. They are contained within a series of HTML elements enclosed in <div> tags with the class ’ncr6’. Similar to Expedia, loading all flight information on the search result page requires clicking a ’show more results’ button until it vanishes, signifying the complete loading of flight information. The aircraft type, however, is embedded within a hidden HTML element using the <div> tag with the class ’o-C7’. Unlike Expedia, it is possible to retain this hidden element displayed per flight when multiple flights are clicked. Scraping this specific element becomes straightforward after initially loading all flight information and subsequently clicking on each flight entry.

2.1.3 Data Saving Format Preparation

As the data scraped from OTA websites will be utilized for exploratory data analysis (EDA) purposes, it is crucial to consider establishing a well-organized dataset structure even prior to the scraping process. Here are some key highlights:

For data elements such as ’Price,’ ’Number of Stops,’ ’Departure Time,’ ’Arriving Time,’ ’Is Direct Flight,’ ’Total Trip Duration in Hours,’ and similar attributes, the relationship is one-to-one per trip. Therefore, it is prudent to employ a single cell to store such elements, utilizing necessary data trimming, splitting, or conversion techniques to extract relevant information from the strings displayed on the search result page.

Conversely, for attributes like ’Airline Companies,’ ’Layover Cities,’ ’Layover Times,’ and ’Type of Aircraft,’ which can vary due to multiple layovers per trip, the relationship is many-to-one per trip. In order to efficiently store and subsequently use this corresponding information for EDA, adopting a list-style format is recommended. For instance, if this trip consists two layovers from Helsinki to Paris, this information should be stored in a cell as follows:

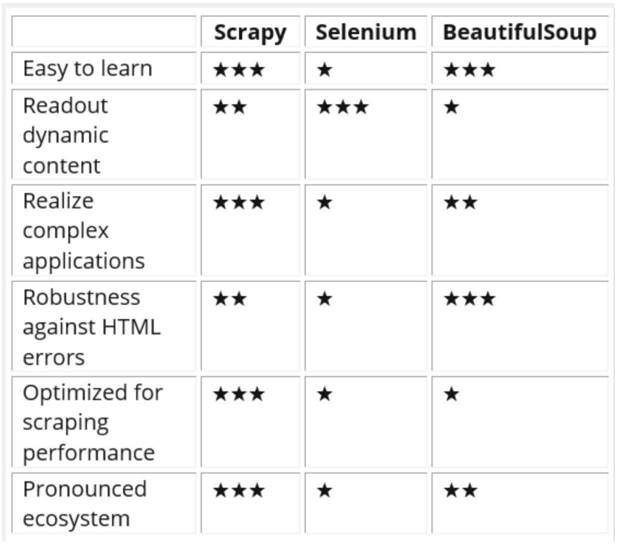
- ’Airline Companies’: [’A Company’, ’B Company’, ’C Company’]
- ’Layover Cities’: [’City A’, ’City B’]
- ’Layover Times’: [’Time A’, ’Time B’]
- ’Type of Aircrafts’: [’A Aircraft’, ’B Aircraft’, ’C Aircraft’]

2.2 Data Scraping

2.2.1 Tools comparison

Scrapy, BeautifulSoup, and Selenium are all popular Python libraries used for web scraping, but they serve different purposes and have distinct features. Detailed comparison can be found in Figure 1.

- Scrapy is a comprehensive web scraping framework specifically designed for web crawling and scraping large websites or web applications. It is ideal for building robust and scalable web scraping spiders.
- BeautifulSoup is primarily a parsing library for HTML and XML documents. It helps in navigating and manipulating the contents of a web page.
- Selenium is an automation tool for web browsers. It allows one to automate interactions with a website by programmatically controlling a web browser (e.g., Chrome or Firefox)



	Scrapy	Selenium	BeautifulSoup
Easy to learn	★★★	★	★★★
Readout dynamic content	★★	★★★	★
Realize complex applications	★★★	★	★★
Robustness against HTML errors	★★	★	★★★
Optimized for scraping performance	★★★	★	★
Pronounced ecosystem	★★★	★	★★

Figure 1: A high level comparison among Scrapy BeautifulSoup and Selenium[1]

2.3 Data Extracting

2.3.1 Scraping Challenges

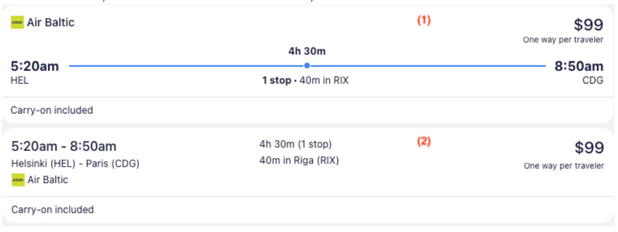
Scraping flight information from OTA (Online Travel Agency) websites poses a significant challenge due to the robust anti-bot measures put in place by these platforms. They have implemented various strategies to detect and prevent automated web scraping, as they aim to protect their data and maintain a level playing field for all users. These anti-bot mechanisms may include CAPTCHA challenges, IP blocking, user-agent detection, and session monitoring.
Because of the aforementioned challenges and the unique features of each web scraping tool, I decided to forgo using BeautifulSoup and Scrapy, opting instead for Selenium for data scrapping. This choice was motivated by the necessity to extract dynamic content during the data scraping process.
However, standard Selenium libraries proved ineffective for scraping OTA websites. I dedicated considerable time to fine-tuning Selenium's parameters, only to encounter repeated failures due to the websites' formidable anti-bot mechanisms and Selenium detection methods. My breakthrough came when tutor introduced a more specialized Selenium package called "selenium_stealth," which effectively mitigates nearly all detection efforts by OTA websites.

2.3.2 Selenium_stealth

To achieve successful data extraction, several key configurations were implemented when using Selenium and Selenium_stealth:

- Disabling popup blocking: Popup windows were intentionally left enabled to ensure seamless interaction with dynamic content.
- Disabling extensions: Browser extensions were disabled to minimize any potential interference.
- Establishing various user agents and randomly selecting one per session: This approach simulates a diverse range of browser cores to mimic human behavior.
- Utilizing the 'webdriver.click().perform()' method: This was employed to programmatically click the 'show more' button, enabling access to dynamic data without detection.

One thing should be noticed here is the 'user_agent' parameter in Selemium. Expedia behaves entirely different when loading pages for different Selenium WebDrivers, i.e. User Browsers. I scrapped the data at the very beginning using randomization of user agents, and I found the HTML layouts vary a lot, details can be reviewed in figure 2. Foe Expedia, I devote myself using the follwing agent: 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/109.0.0.0 Safari/537.36'.



Air Baltic (1) $99
One way per traveler
4h 30m
5:20am HEL — 1 stop · 40m in RIX — 8:50am CDG
Carry-on included

5:20am - 8:50am
Helsinki (HEL) - Paris (CDG)
Air Baltic
4h 30m (1 stop)
40m in Riga (RIX)
(2) $99
One way per traveler
Carry-on included

Figure 2: Mozilla/5.0 for Macintosh-Intel Mac OS X (1) v.s. Windows NT (2)

Given the ever-changing layout and the evolving anti-scraping measures, my approach is to preserve the entire webpage by saving it as plain HTML files. The subsequent data extraction, as per my strategy, will be accomplished using BeautifulSoup.

2.3.3 BeautifulSoup

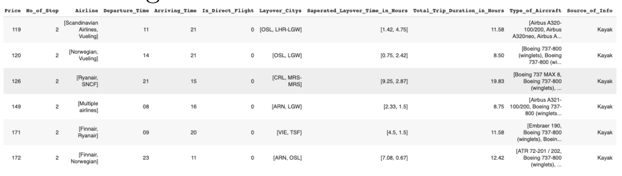
Extracting data from scraped HTML files becomes a straightforward process with the use of BeautifulSoup. Key considerations for this task include:

- Analyzing Layout: Begin by examining the layout of the iterated blocks and identifying the most suitable outermost layer for data extraction.
- Block Iteration: Iterate through the blocks and directly locate the desired elements within.
- Handling Multiple Elements: If multiple elements meet your specific criteria, iterate through them again to pinpoint the correct one.

When extracting each element, the following points should be observed:

- For 'Price' and 'Source of Info,' employ regular text extraction methods.
- 'No of Stop' should be converted, such as changing 'nonstop' to 0, in order to establish an ordinal scale ranging from 0 to N for analysis.
- 'Airline,' 'Layover Cities,' and 'Type of Aircraft' should be saved in list objects per trip, leveraging the website's natural formatting.
- For 'Departure Time' and 'Arriving Time,' conduct regular text extraction without converting to numerical values (note that some arriving times may be on a separate day; retaining the original format is essential for subsequent EDA analysis).
- 'Is Direct Flight' can be handled by setting up a flag with values of 0 or 1 using conditional statements.
- 'Total Trip Duration in Hours' and 'Separated Layover Time in Hours' should be saved in list objects per trip, utilizing the inherent format on the website. Additionally, parse these values into hours to ensure uniform scaling.

With each iteration, append the extracted flight information to a DataFrame, ensuring it is aggregated and ready for future use. Detailed DataFrame can be found in Figure 3



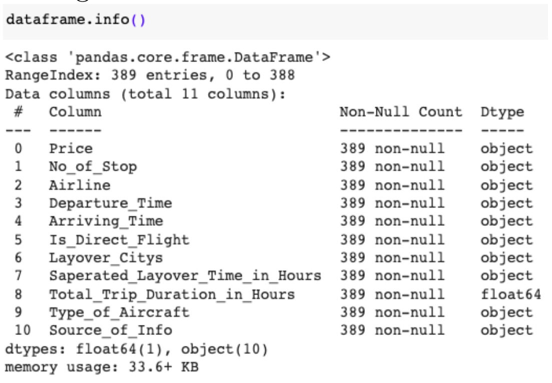
Price	No_of_Stop	Airline	Departure_Time	Arriving_Time	Is_Direct_Flight	Layover_Citys	Saperated_Layover_Time_in_Hours	Total_Trip_Duration_in_Hours	Type_of_Aircraft	Source_of_Info
119	2	[Scandinavian Airlines, Vueling]	11	21	0	[OSL, LHR-LGW]	[1.42, 4.75]	11.58	[Airbus A320-100/200, Airbus A320neo, Airbus A...	Kayak
120	2	[Norwegian, Vueling]	14	21	0	[OSL, LGW]	[0.75, 2.42]	8.50	[Boeing 737-800 (winglets), Boeing 737-800 (wi...	Kayak
126	2	[Ryanair, SNCF]	21	15	0	[CRL, MRS-MRS]	[9.25, 2.87]	19.83	[Boeing 737 MAX 8, Boeing 737-800 (winglets), ...	Kayak
149	2	[Multiple airlines]	08	16	0	[ARN, LGW]	[2.33, 1.5]	8.75	[Airbus A321-100/200, Boeing 737-800 (winglets...	Kayak
171	2	[Finnair, Ryanair]	09	20	0	[VIE, TSF]	[4.5, 1.5]	11.58	[Embraer 190, Boeing 737-800 (winglets), Boein...	Kayak
172	2	[Finnair, Norwegian]	23	11	0	[ARN, OSL]	[7.08, 0.67]	12.42	[ATR 72-201 / 202, Boeing 737-800 (winglets), ...	Kayak

Figure 3: Aggregated Flight Information in a DataFrame

3 Data analysis(EDA)

3.1 Overview

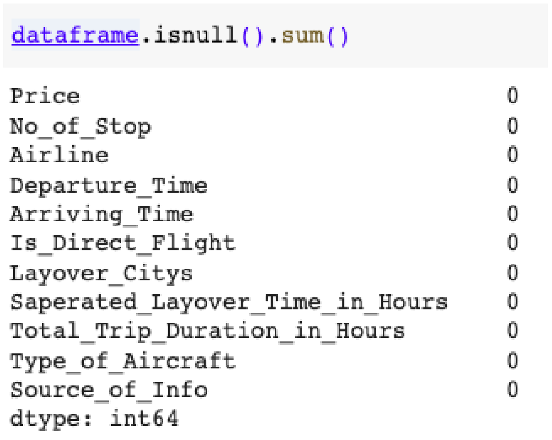
Navigating the compiled flight information within the DataFrame serves as a pivotal initial phase in the Exploratory Data Analysis (EDA) endeavor. This step is instrumental in acquiring a thorough understanding of the dataset's inherent traits and content. Through the scrutiny of data types, missing entries, summary statistics, and distinct values, an evaluation of data quality and structural aspects becomes achievable. Moreover, the utilization of visualization tools, filtering mechanisms, and aggregation methodologies used in later sections will be harnessed to facilitate the discernment of underlying trends, patterns, and any exceptional data points or anomalies.
As I have already addressed the essential data format considerations prior to scraping data from OTAs, the initial assessment of the dataset appears favorable. For a comprehensive breakdown of the DataFrame's specifics, please refer to the accompanying figure 4 and figure 5



```
dataframe.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 389 entries, 0 to 388
Data columns (total 11 columns):
 #   Column                           Non-Null Count  Dtype  
---  ------                           --------------  -----  
 0   Price                            389 non-null    object 
 1   No_of_Stop                       389 non-null    object 
 2   Airline                          389 non-null    object 
 3   Departure_Time                   389 non-null    object 
 4   Arriving_Time                    389 non-null    object 
 5   Is_Direct_Flight                 389 non-null    object 
 6   Layover_Citys                    389 non-null    object 
 7   Saperated_Layover_Time_in_Hours  389 non-null    object 
 8   Total_Trip_Duration_in_Hours     389 non-null    float64
 9   Type_of_Aircraft                 389 non-null    object 
 10  Source_of_Info                   389 non-null    object 
dtypes: float64(1), object(10)
memory usage: 33.6+ KB
```

Figure 4: DataFrame Information



```
dataframe.isnull().sum()

Price                              0
No_of_Stop                         0
Airline                            0
Departure_Time                     0
Arriving_Time                      0
Is_Direct_Flight                   0
Layover_Citys                      0
Saperated_Layover_Time_in_Hours    0
Total_Trip_Duration_in_Hours       0
Type_of_Aircraft                   0
Source_of_Info                     0
dtype: int64
```

Figure 5: DataFrame Missing Value

3.2 Data Preprocessing

Before progressing to dataset visualization, it is vital to address specific data elements that may require type conversion or format adjustments. This initial data processing phase ensures that the data is well-structured and coherent, establishing a strong foundation for effective visualization and analysis. The remaining data processing tasks include:

- Converting 'Price,' 'No_of_Stop,' and 'Is_Direct_Flight' from string data type to integers.
- Transforming 'Airline' from a single string into a list, similar to other aggregated columns like 'Layover_Cities.'
- Converting 'Departure_Time' and 'Arriving_Time' into hours format to ensure a consistent scale for analysis purposes and copy originals for further display purpose.

3.3 Data Visualization

3.3.1 Price

Figure 6 centers on comprehending the distribution of prices, encompassing measures such as the minimum, first quartile, median, third quartile, maximum and range. Additionally, it will involve identifying any outliers or exceptionally high or low values that could suggest errors or unique circumstances.

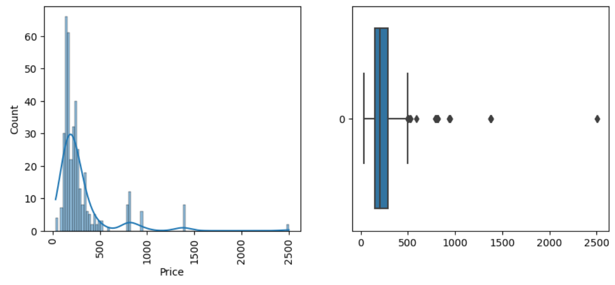


Figure 6: Price Distribution and its Box-Plot

3.3.2 Number of Stops

Figure 7 concentrates on generating a summary of the stop counts, encompassing non-stop flights, and conducting an analysis to recognize prevalent stop count patterns. Additionally, it will explore the correlation between the number of stops and price. The correlation between the number of stops and the cost per trip is clearly evident: as the number of stops increases, the ticket price per journey also rises.

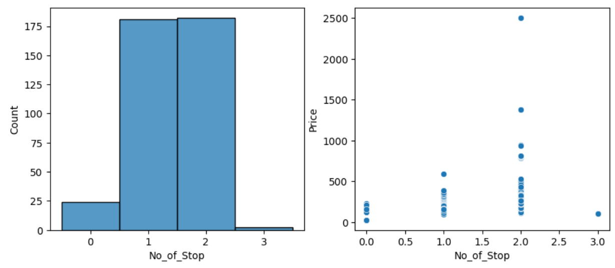


Figure 7: No. of Stops Distribution and its scatterplot with Price

3.3.3 Airline

Figure 8 illustrates the task of identifying the airlines that are most commonly operated for this route. The top three most frequent Airlines are:

- Finnair: Finnair is the flag carrier and largest airline of Finland.
- Lufthansa: Lufthansa is the largest airline in Germany and one of the largest in Europe.
- Scandinavian Airlines (SAS): Scandinavian Airlines is the flag carrier of Denmark, Norway, and Sweden, collectively known as Scandinavia.

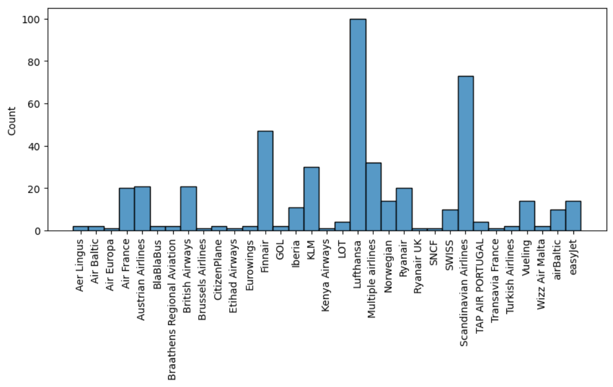


Figure 8: Airline distribution

3.3.4 Departure Time and Arriving Time

Figure 9 unmistakably depicts that the highest frequency of flights from Helsinki to France takes place on November 15th, with departures typically occurring around 07:00 AM and arrivals spanning from 3:00 PM to 8:00 PM in the afternoon. Additionally, although the correlation between departure time and prices isn't notably strong, it's evident that ticket prices tend to be relatively higher in the morning compared to the afternoon. Similarly, when considering arrival times, flights in the afternoon generally have higher prices than those arriving in the morning. In fact, the majority of expensive flights depart in the morning and arrive in the afternoon.

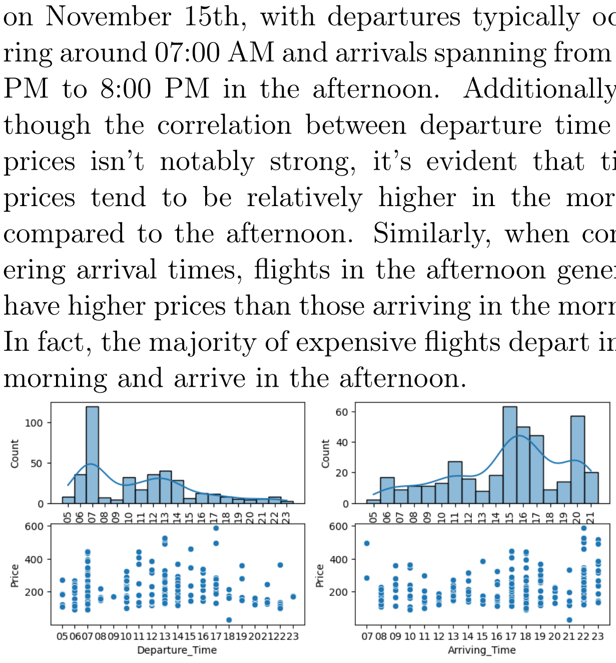


Figure 9: Departure and Arriving Time Distribution, and those correlations with Price

3.3.5 Layover Cities

Figure 10 depicts the prevalence of layover cities for journeys from Helsinki to Paris. The top three most frequent layover cities are:

- ARN: Stockholm Arlanda Airport, Sweden
- MUC: Munich Airport, Germany
- FRA: Frankfurt am Main Airport, Germany

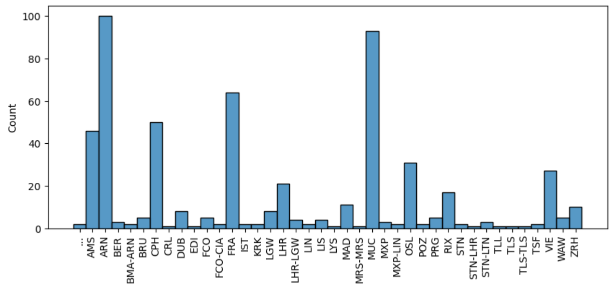


Figure 10: Layover cities during the trip from Helsinki to Paris

3.3.6 Layover Time

Figure 11 focuses on gaining an understanding of the layover time distribution, including statistical measures like the minimum, first quartile, median, third quartile, maximum and range. Furthermore, it entails the detection of any outliers or unusually high or low values that may indicate errors or special situations.

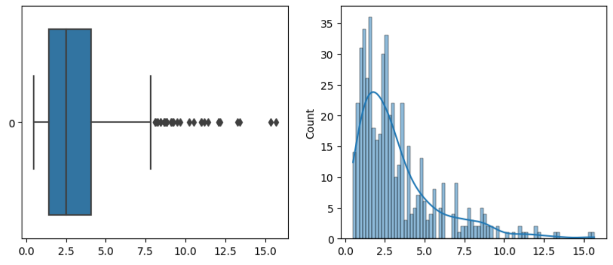


Figure 11: Layover Time Distribution and its BoxPlot

3.3.7 Total Trip Duration

Figure 12 is dedicated to developing an insight into the distribution of total trip duration, which encompasses statistical metrics such as the minimum, first quartile, median, third quartile, maximum and range. A point-to-point air travel journey, in this case, from Helsinki to Paris, typically lasting less than 9 hours, including layovers, is quite reasonable.

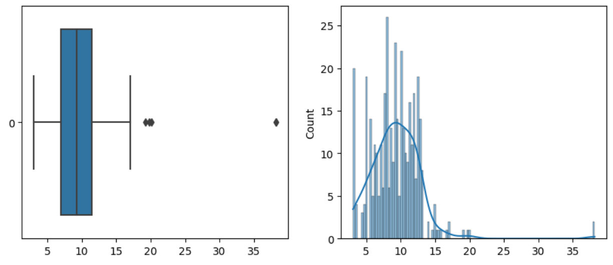


Figure 12: Total trip duration Distribution and its BoxPlot

3.3.8 Aircraft Type

Figure 13 illustrates the task of identifying the Aircraft Type that are most commonly chosen for this route. The top three most frequent Aircraft Type are:

- Airbus A320 with Sharklets
- Airbus A320neo (New Engine Option)
- Boeing 737-800 with Winglets

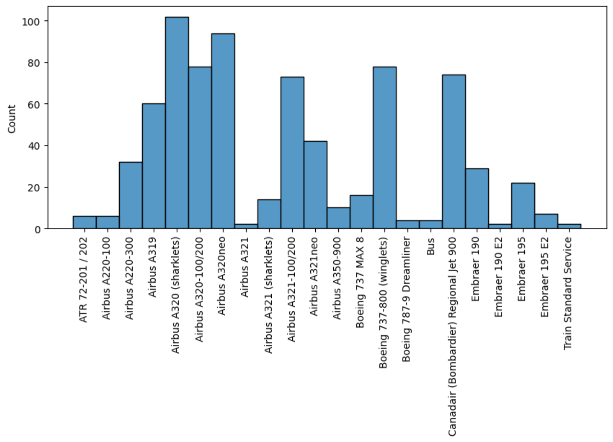


Figure 13: Aircraft Type Distribution

4 User Interaction

4.1 Ipywidgets Package

Investigating the interactive functionalities offered by the ipywidgets library in Python involves the utilization of widgets such as sliders, buttons, dropdown menus, and various other interactive elements. These widgets empower end users to modify data, adjust parameters, and observe instant updates in visualizations or calculations.

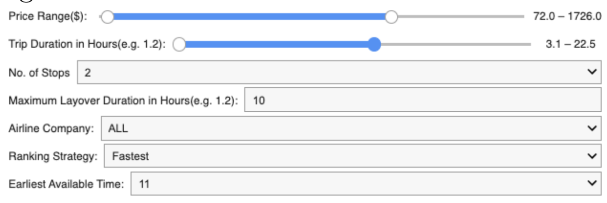
4.2 User Interaction Design and Implementation

- Generate a variety of widgets, including sliders, dropdown menus, textboxes, and more.
- Define the minimum and maximum values of each widget, either through numeric ranges or finite options that align with the characteristics of each element.
- Execute the widgets to collect immediate input from end users.
- Establish a DataFrame Cast based on the user-provided input.

- Obtain the filtered DataFrame result by applying the established Cast.

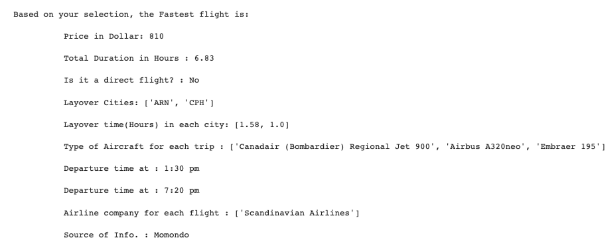
- Display the formatted flight information through printing or presentation.

The detailed output can be found in figure 14 and in figure 15



Price Range($): 72.0 – 1726.0
Trip Duration in Hours(e.g. 1.2): 3.1 – 22.5
No. of Stops: 2
Maximum Layover Duration in Hours(e.g. 1.2): 10
Airline Company: ALL
Ranking Strategy: Fastest
Earliest Available Time: 11

Figure 14: UI Design



```
Based on your selection, the Fastest flight is:

          Price in Dollar: 810

          Total Duration in Hours : 6.83

          Is it a direct flight? : No

          Layover Cities: ['ARN', 'CPH']

          Layover time(Hours) in each city: [1.58, 1.0]

          Type of Aircraft for each trip : ['Canadair (Bombardier) Regional Jet 900', 'Airbus A320neo', 'Embraer 195']

          Departure time at : 1:30 pm

          Departure time at : 7:20 pm

          Airline company for each flight : ['Scandinavian Airlines']

          Source of Info. : Momondo
```

Figure 15: Formatted Outputs

5 Conclusion

In summary, this project has provided me with a holistic experience of the key stages in the data lifecycle—starting from data acquisition, progressing to information extraction, and culminating in knowledge generation. Despite its inherent challenges, the endeavor has proven to be immensely worthwhile.

- The process of data acquisition encompasses diverse tools, and I can manage them in a good way. Meanwhile, scrapping data makes me underscore the importance of adhering to data privacy and protection regulations in practice.

- This project has reinforced the significance of meticulous initial design, as it substantially streamlines subsequent data cleaning and early-stage Exploratory Data Analysis (EDA). A well-structured data framework ensures that EDA objectives can be efficiently and effectively met.

- The Exploratory Data Analysis phase has enabled me to grasp crucial insights, including the distribution of flight durations, the identification of frequently encountered layover cities, and the exploration of the impact of departure times on ticket prices. These findings serve as the cornerstone for more intricate analyses to come.

References

[1] S. Ravinder. Web scrapping dynamic content using api and beautiful soup, 2021. URL `https://www.numpyninja.com/post/web-scrapping-dynamic-content-using-api-and-beaut`