# Flight and Hotel Price Analysis from Kayak.ae

توفيق كامل توفيق حسين 23011240

ساهر سامي 23011280

نور الدين امجد عبد الفتاح احمد محمد 23011596

## Brief Description:

This project involves scraping data from Kayak.ae, a travel search engine that aggregates prices for flights,

hotels, and car rentals. The focus is on extracting flight and hotel pricing(BLOCKED) data for various destinations and

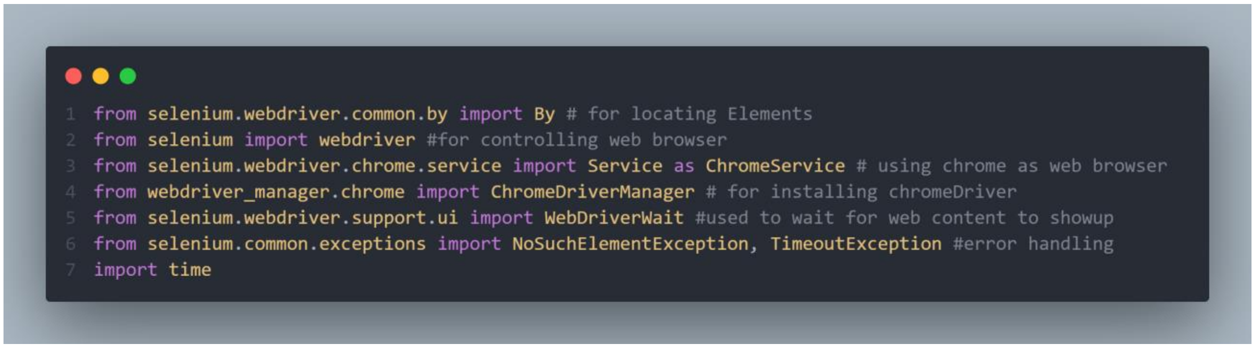
analyzing how prices vary based on multiple factors.

## Goal of the Project:

- Analyze pricing trends for flights and hotels based on travel dates, destinations, and booking time.
- Answer questions such as:
- How do prices vary by day of the week or booking time?
- What are the most affordable destinations from a specific city?
- Are there noticeable seasonal trends?

## Data Extraction Approach:

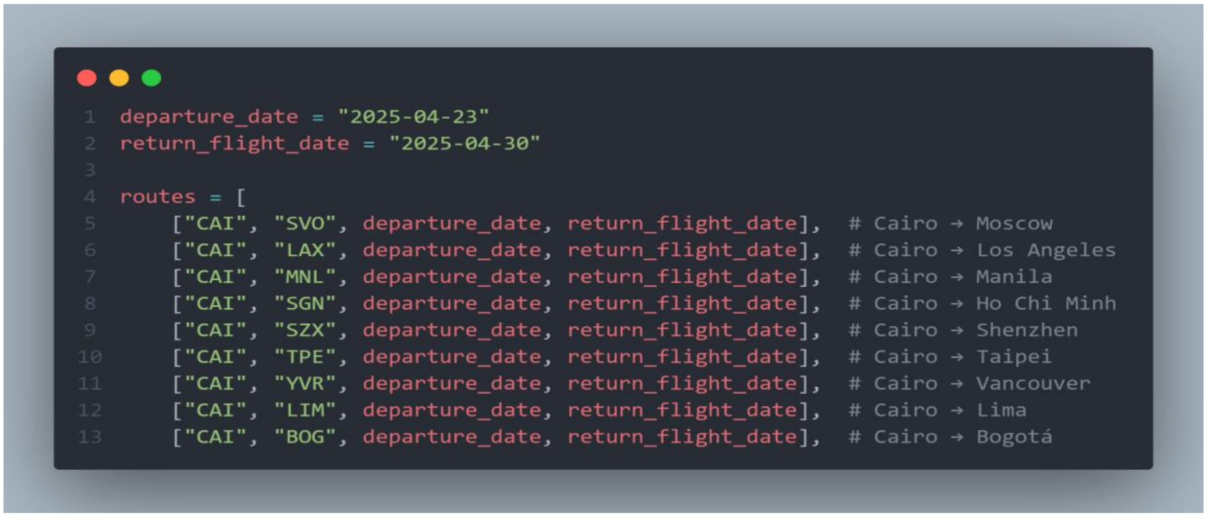
- Use Python with Selenium to scrape dynamic content.



```
from selenium.webdriver.common.by import By # for locating Elements
from selenium import webdriver #for controlling web browser
from selenium.webdriver.chrome.service import Service as ChromeService # using chrome as web browser
from webdriver_manager.chrome import ChromeDriverManager # for installing chromeDriver
from selenium.webdriver.support.ui import WebDriverWait #used to wait for web content to showup
from selenium.common.exceptions import NoSuchElementException, TimeoutException #error handling
import time
```

We use chrome driver to simulate web browser to scrape the flight data

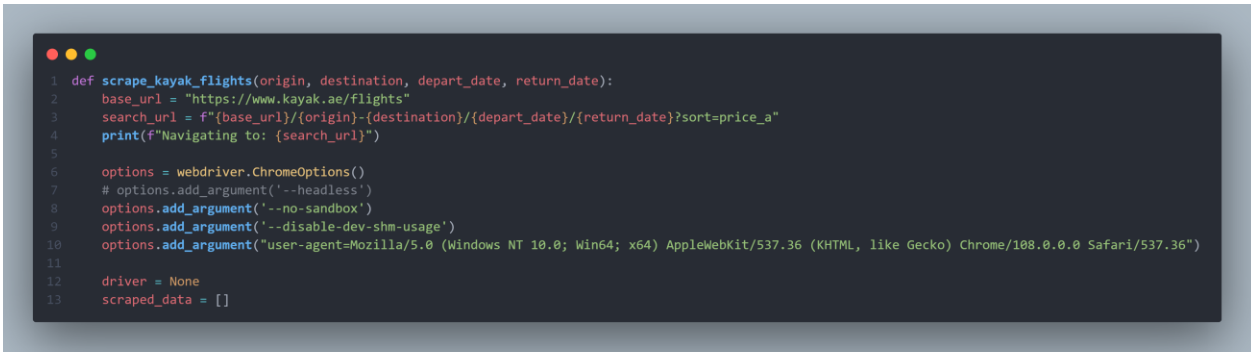
- Simulate user searches for routes and hotel stays(BLOCKED).



```
departure_date = "2025-04-23"
return_flight_date = "2025-04-30"

routes = [
    ["CAI", "SVO", departure_date, return_flight_date],  # Cairo → Moscow
    ["CAI", "LAX", departure_date, return_flight_date],  # Cairo → Los Angeles
    ["CAI", "MNL", departure_date, return_flight_date],  # Cairo → Manila
    ["CAI", "SGN", departure_date, return_flight_date],  # Cairo → Ho Chi Minh
    ["CAI", "SZX", departure_date, return_flight_date],  # Cairo → Shenzhen
    ["CAI", "TPE", departure_date, return_flight_date],  # Cairo → Taipei
    ["CAI", "YVR", departure_date, return_flight_date],  # Cairo → Vancouver
    ["CAI", "LIM", departure_date, return_flight_date],  # Cairo → Lima
    ["CAI", "BOG", departure_date, return_flight_date],  # Cairo → Bogotá
```

Sample of routes that we used in the scrapping in the full routs file it contain 102 routs

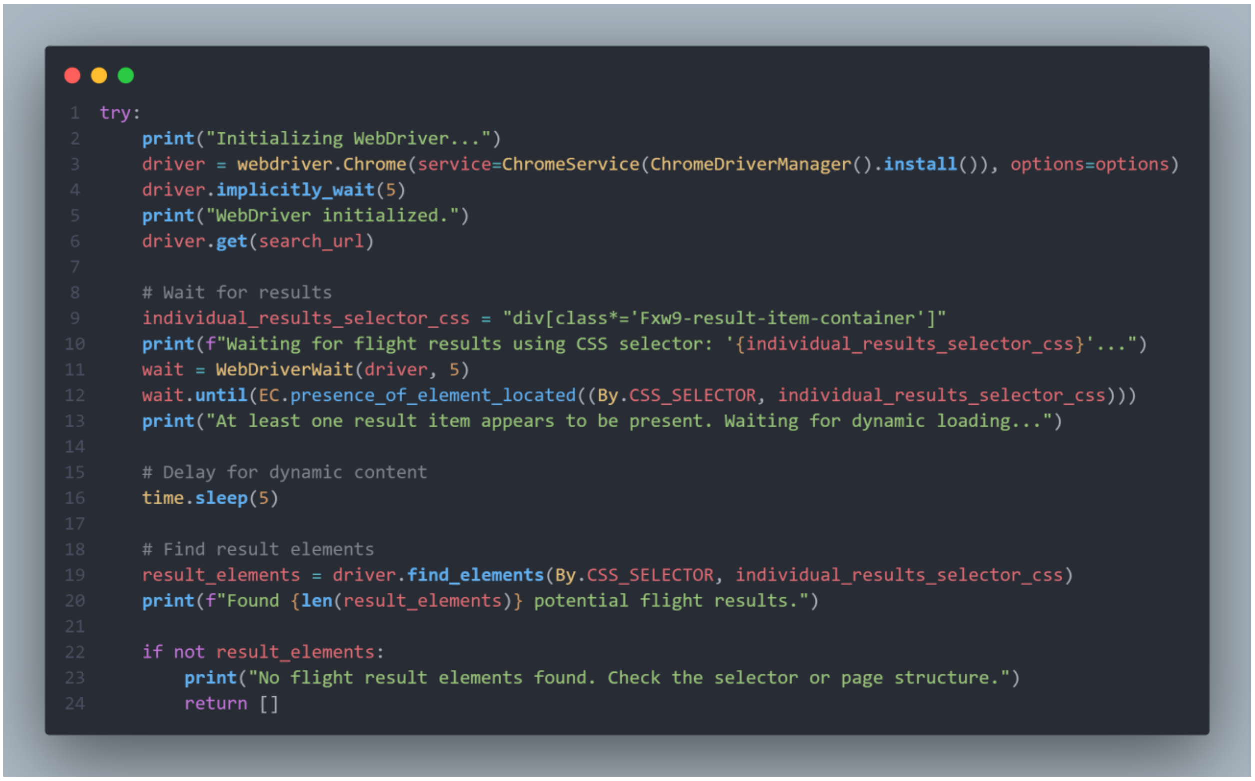
- Extract destination, price, date, airline/hotel name, and rating(BLOCKED)



```python
def scrape_kayak_flights(origin, destination, depart_date, return_date):
    base_url = "https://www.kayak.ae/flights"
    search_url = f"{base_url}/{origin}-{destination}/{depart_date}/{return_date}?sort=price_a"
    print(f"Navigating to: {search_url}")

    options = webdriver.ChromeOptions()
    # options.add_argument('--headless')
    options.add_argument('--no-sandbox')
    options.add_argument('--disable-dev-shm-usage')
    options.add_argument("user-agent=Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/108.0.0.0 Safari/537.36")

    driver = None
    scraped_data = []
```

We start with defining the scrape function which take the origin, destination, departure date and return date

In the second part we define the url that will be scraped and options for the chrome driver



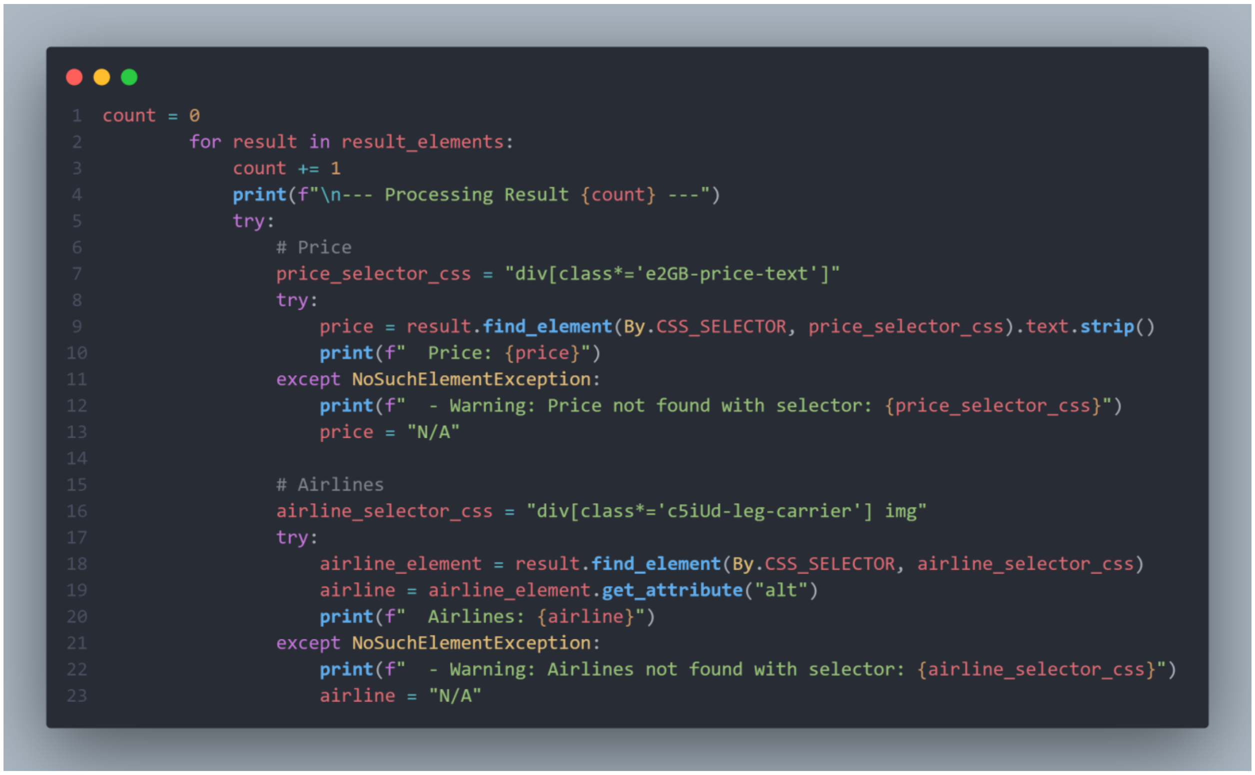
```python
try:
    print("Initializing WebDriver...")
    driver = webdriver.Chrome(service=ChromeService(ChromeDriverManager().install()), options=options)
    driver.implicitly_wait(5)
    print("WebDriver initialized.")
    driver.get(search_url)

    # Wait for results
    individual_results_selector_css = "div[class*='Fxw9-result-item-container']"
    print(f"Waiting for flight results using CSS selector: '{individual_results_selector_css}'...")
    wait = WebDriverWait(driver, 5)
    wait.until(EC.presence_of_element_located((By.CSS_SELECTOR, individual_results_selector_css)))
    print("At least one result item appears to be present. Waiting for dynamic loading...")

    # Delay for dynamic content
    time.sleep(5)

    # Find result elements
    result_elements = driver.find_elements(By.CSS_SELECTOR, individual_results_selector_css)
    print(f"Found {len(result_elements)} potential flight results.")

    if not result_elements:
        print("No flight result elements found. Check the selector or page structure.")
        return []
```

In this code, we initialize a Chrome WebDriver using `ChromeDriverManager` to automatically handle the driver installation. After setting an implicit wait, we navigate to the target URL (`search_url`).

We then define a CSS selector to identify the container holding individual flight results. To ensure the page is fully loaded, we explicitly wait until at least one matching element appears in the DOM.

Afterward, we add an extra short sleep to allow any dynamic content to finish rendering.

Finally, we search for all elements matching the flight result container selector. If no results are found, a message is printed to indicate that the selector might be incorrect or the page structure may have changed.



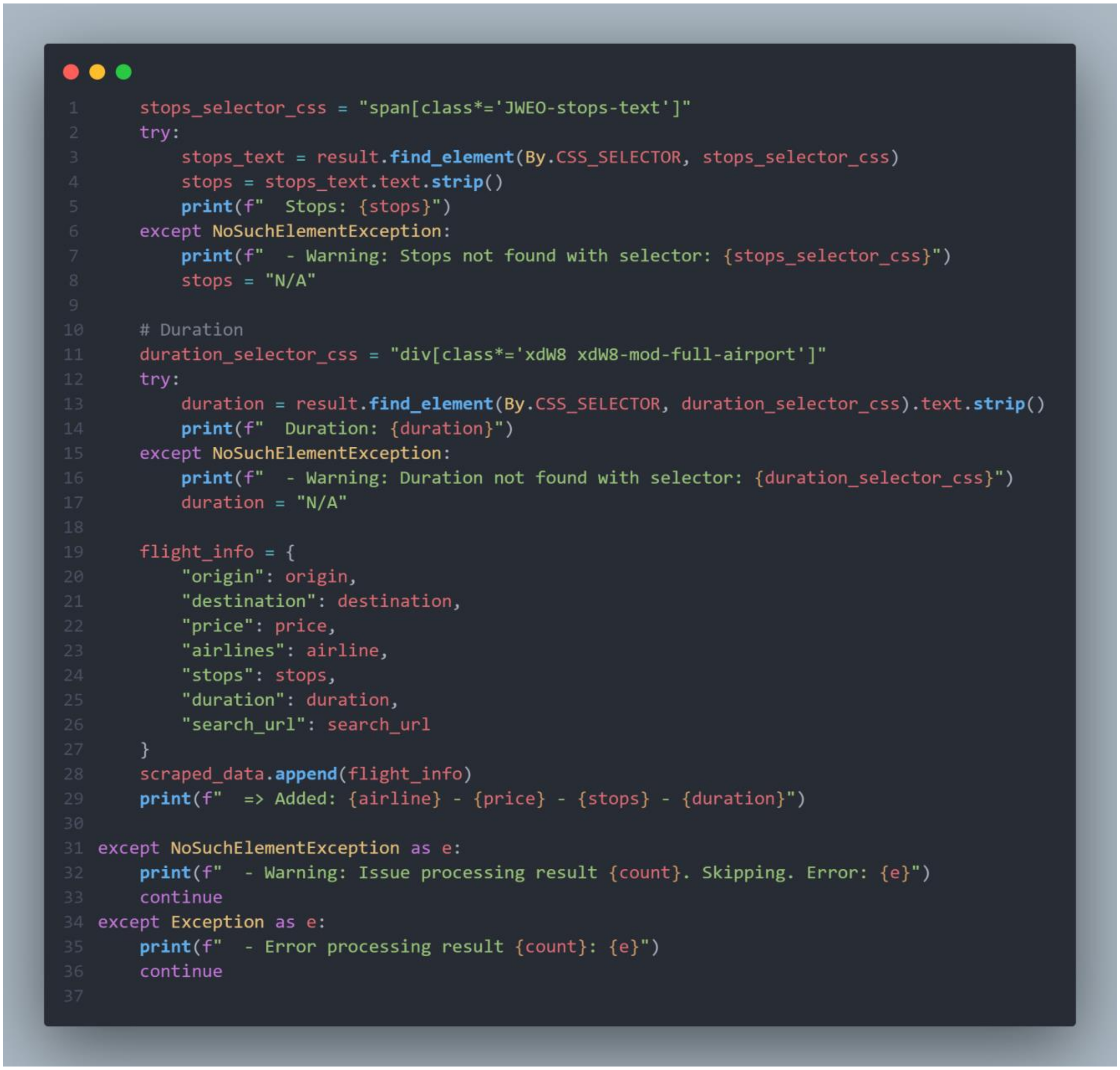
```
count = 0
        for result in result_elements:
            count += 1
            print(f"\n--- Processing Result {count} ---")
            try:
                # Price
                price_selector_css = "div[class*='e2GB-price-text']"
                try:
                    price = result.find_element(By.CSS_SELECTOR, price_selector_css).text.strip()
                    print(f"  Price: {price}")
                except NoSuchElementException:
                    print(f"  - Warning: Price not found with selector: {price_selector_css}")
                    price = "N/A"

                # Airlines
                airline_selector_css = "div[class*='c5iUd-leg-carrier'] img"
                try:
                    airline_element = result.find_element(By.CSS_SELECTOR, airline_selector_css)
                    airline = airline_element.get_attribute("alt")
                    print(f"  Airlines: {airline}")
                except NoSuchElementException:
                    print(f"  - Warning: Airlines not found with selector: {airline_selector_css}")
                    airline = "N/A"
```

After retrieving the container that holds all the flight data, we iterate through each result element.

For each flight entry, we first attempt to extract the price by locating the appropriate sub-element using its CSS selector. If the price is successfully found, we extract and clean its text; if not, we assign "N/A" as a fallback value.

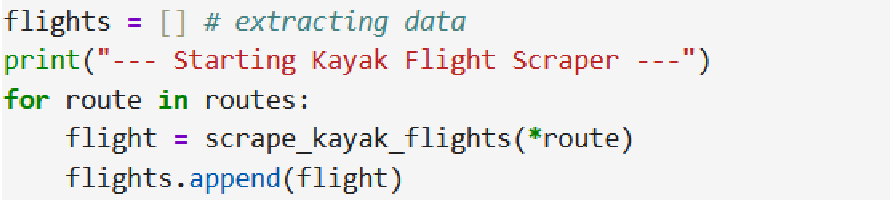
Similarly, we attempt to extract the airline name by finding the airline logo element and reading its alt attribute. If the airline information cannot be found, we also assign "N/A" as a default.

This ensures that missing fields do not cause the script to crash and that each flight entry remains consistent even if some data is unavailable.

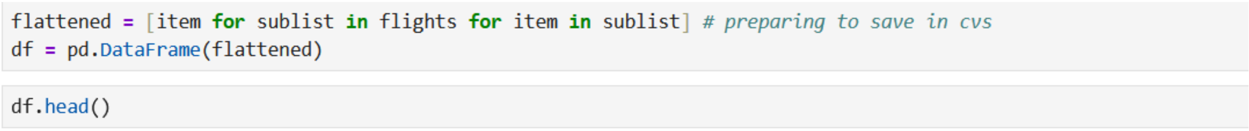


```
    stops_selector_css = "span[class*='JWEO-stops-text']"
    try:
        stops_text = result.find_element(By.CSS_SELECTOR, stops_selector_css)
        stops = stops_text.text.strip()
        print(f"  Stops: {stops}")
    except NoSuchElementException:
        print(f"  - Warning: Stops not found with selector: {stops_selector_css}")
        stops = "N/A"

    # Duration
    duration_selector_css = "div[class*='xdW8 xdW8-mod-full-airport']"
    try:
        duration = result.find_element(By.CSS_SELECTOR, duration_selector_css).text.strip()
        print(f"  Duration: {duration}")
    except NoSuchElementException:
        print(f"  - Warning: Duration not found with selector: {duration_selector_css}")
        duration = "N/A"

    flight_info = {
        "origin": origin,
        "destination": destination,
        "price": price,
        "airlines": airline,
        "stops": stops,
        "duration": duration,
        "search_url": search_url
    }
    scraped_data.append(flight_info)
    print(f"  => Added: {airline} - {price} - {stops} - {duration}")

except NoSuchElementException as e:
    print(f"  - Warning: Issue processing result {count}. Skipping. Error: {e}")
    continue
except Exception as e:
    print(f"  - Error processing result {count}: {e}")
    continue
```

continuing as above but we include flights stops we attempt to extract the stops by locating the appropriate
sub-element using its CSS selector. If the stops is successfully found, we extract and clean its text; if not, we assign "N/A" as a fallback value
And if there is any complication during the process it will throw an exception
And finally, we store all our finding into a dictionary so we can process it later

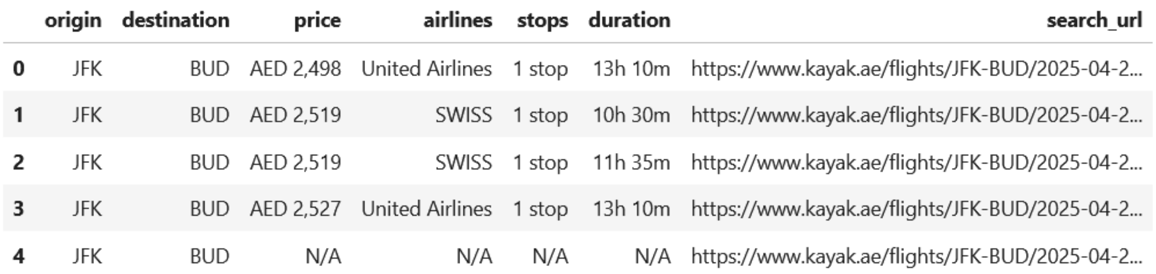


```
flights = [] # extracting data
print("--- Starting Kayak Flight Scraper ---")
for route in routes:
    flight = scrape_kayak_flights(*route)
    flights.append(flight)
```

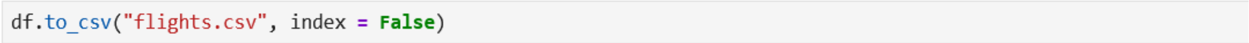


```
flattened = [item for sublist in flights for item in sublist] # preparing to save in cvs
df = pd.DataFrame(flattened)
```

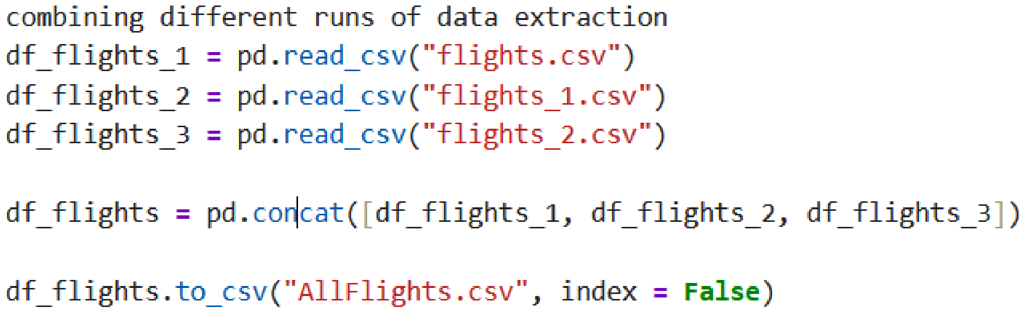
```
df.head()
```



| | origin | destination | price | airlines | stops | duration | search_url |
|---|---|---|---|---|---|---|---|
| **0** | JFK | BUD | AED 2,498 | United Airlines | 1 stop | 13h 10m | https://www.kayak.ae/flights/JFK-BUD/2025-04-2... |
| **1** | JFK | BUD | AED 2,519 | SWISS | 1 stop | 10h 30m | https://www.kayak.ae/flights/JFK-BUD/2025-04-2... |
| **2** | JFK | BUD | AED 2,519 | SWISS | 1 stop | 11h 35m | https://www.kayak.ae/flights/JFK-BUD/2025-04-2... |
| **3** | JFK | BUD | AED 2,527 | United Airlines | 1 stop | 13h 10m | https://www.kayak.ae/flights/JFK-BUD/2025-04-2... |
| **4** | JFK | BUD | N/A | N/A | N/A | N/A | https://www.kayak.ae/flights/JFK-BUD/2025-04-2... |



```
df.to_csv("flights.csv", index = False)
```

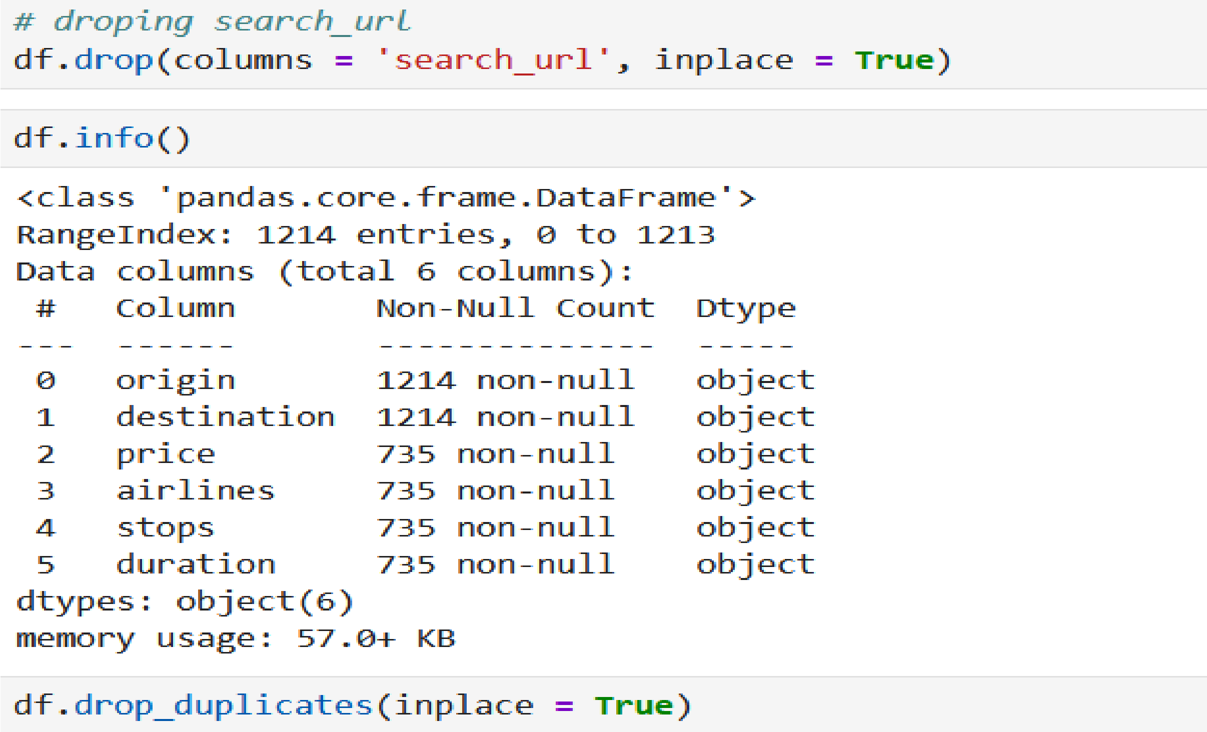


```
combining different runs of data extraction
df_flights_1 = pd.read_csv("flights.csv")
df_flights_2 = pd.read_csv("flights_1.csv")
df_flights_3 = pd.read_csv("flights_2.csv")

df_flights = pd.concat([df_flights_1, df_flights_2, df_flights_3])

df_flights.to_csv("AllFlights.csv", index = False)
```

Running the script multiple time and saving the all results in AllFlights.csv

## Data Cleaning:

- Normalize dates and prices.
- Handle missing or inconsistent entries.



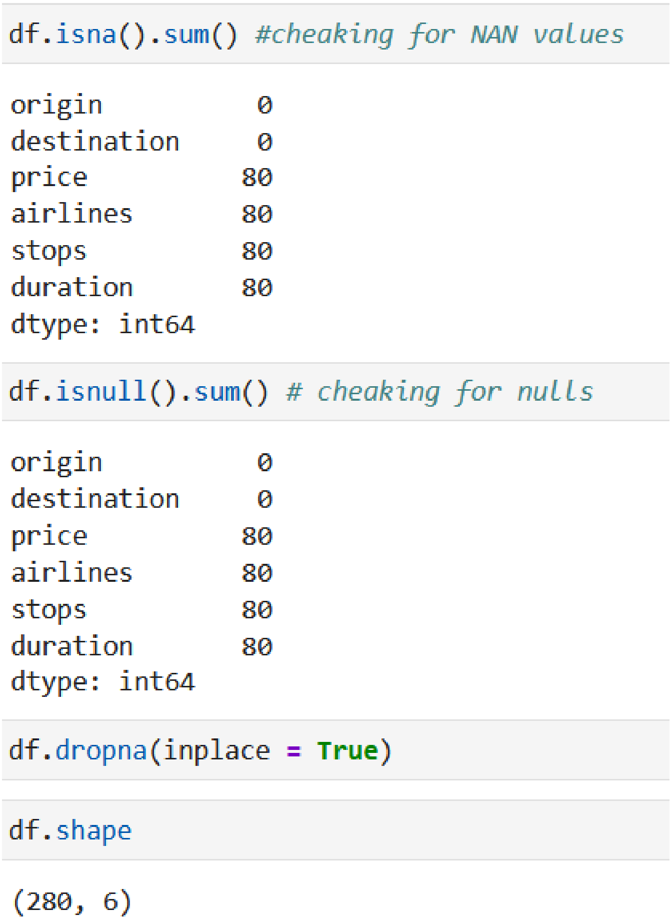
```
# droping search_url
df.drop(columns = 'search_url', inplace = True)
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1214 entries, 0 to 1213
Data columns (total 6 columns):
 #   Column       Non-Null Count  Dtype 
---  ------       --------------  ----- 
 0   origin       1214 non-null   object
 1   destination  1214 non-null   object
 2   price        735 non-null    object
 3   airlines     735 non-null    object
 4   stops        735 non-null    object
 5   duration     735 non-null    object
dtypes: object(6)
memory usage: 57.0+ KB
```

```
df.drop_duplicates(inplace = True)
```

After reading the csv file into pandas data frame we are dropping the Search_url because it unnecessary column and dropping duplicates



```
df.isna().sum() #cheaking for NAN values
```

```
origin          0
destination     0
price          80
airlines       80
stops          80
duration       80
dtype: int64
```

```
df.isnull().sum() # cheaking for nulls
```

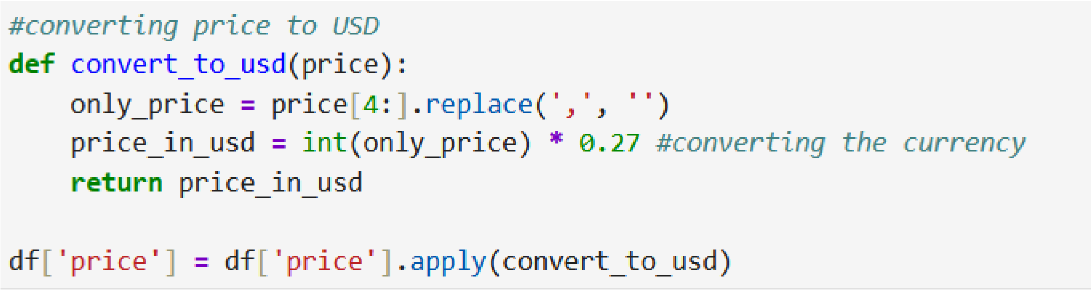
```
origin          0
destination     0
price          80
airlines       80
stops          80
duration       80
dtype: int64
```

```
df.dropna(inplace = True)
```

```
df.shape
```

```
(280, 6)
```

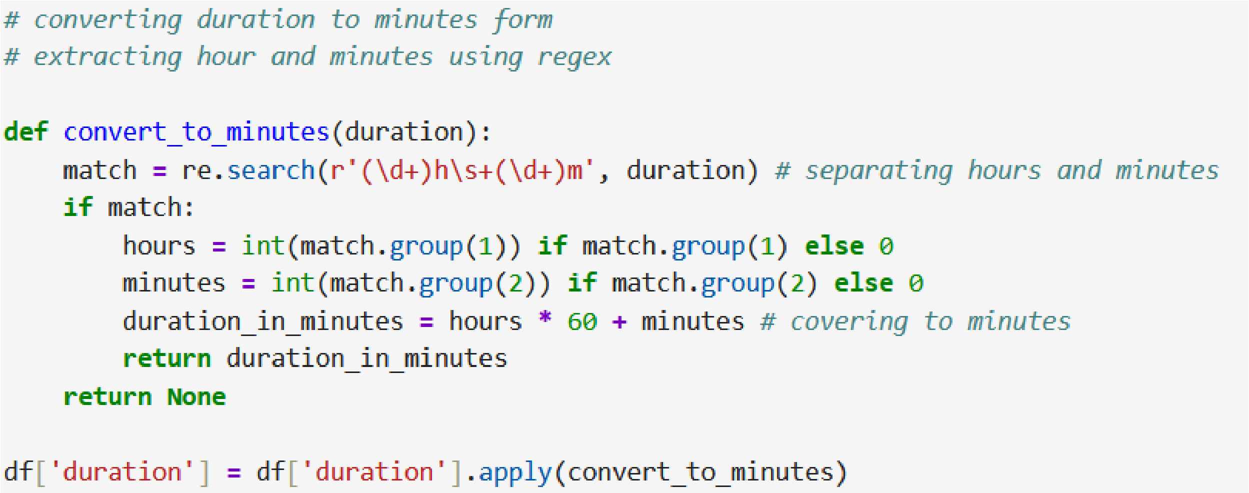
Checking for Null or NaN values and dropping them



```
#converting price to USD
def convert_to_usd(price):
    only_price = price[4:].replace(',', '')
    price_in_usd = int(only_price) * 0.27 #converting the currency
    return price_in_usd

df['price'] = df['price'].apply(convert_to_usd)
```

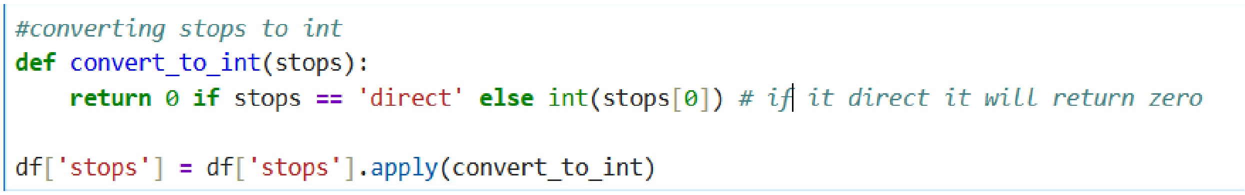
Using the latest exchange rate from United Arab Emirates Dirham to USD, we convert the price

- Convert text to categorical/numerical formats.



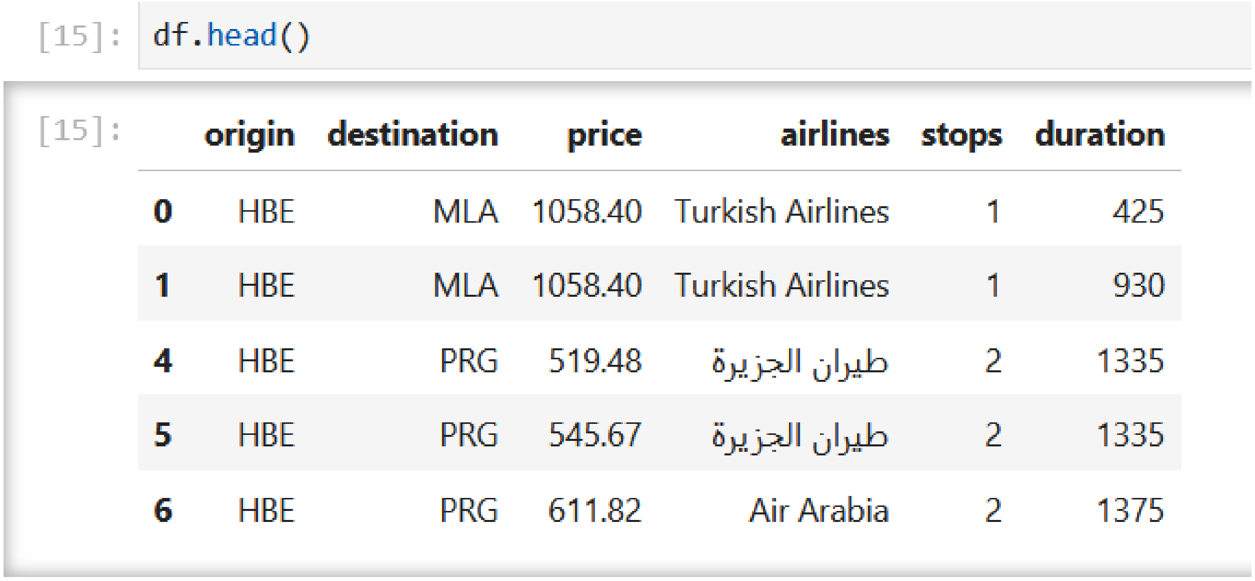
```python
# converting duration to minutes form
# extracting hour and minutes using regex

def convert_to_minutes(duration):
    match = re.search(r'(\d+)h\s+(\d+)m', duration) # separating hours and minutes
    if match:
        hours = int(match.group(1)) if match.group(1) else 0
        minutes = int(match.group(2)) if match.group(2) else 0
        duration_in_minutes = hours * 60 + minutes # covering to minutes
        return duration_in_minutes
    return None

df['duration'] = df['duration'].apply(convert_to_minutes)
```

Using regex, we successfully extracted hours and minutes from the duration string  and converted them into total minutes



```python
#converting stops to int
def convert_to_int(stops):
    return 0 if stops == 'direct' else int(stops[0]) # if it direct it will return zero

df['stops'] = df['stops'].apply(convert_to_int)
```

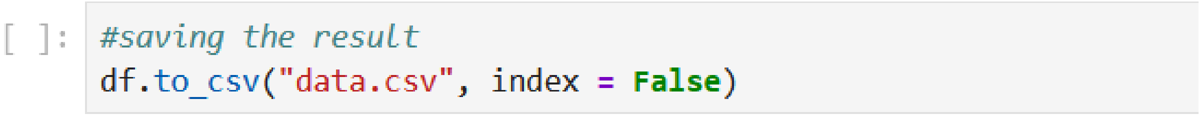
Coverting stops to int and if it direct replace it with zero



```
[15]: df.head()
```

[15]:

|  | origin | destination | price | airlines | stops | duration |
|---|---|---|---|---|---|---|
| **0** | HBE | MLA | 1058.40 | Turkish Airlines | 1 | 425 |
| **1** | HBE | MLA | 1058.40 | Turkish Airlines | 1 | 930 |
| **4** | HBE | PRG | 519.48 | طيران الجزيرة | 2 | 1335 |
| **5** | HBE | PRG | 545.67 | طيران الجزيرة | 2 | 1335 |
| **6** | HBE | PRG | 611.82 | Air Arabia | 2 | 1375 |

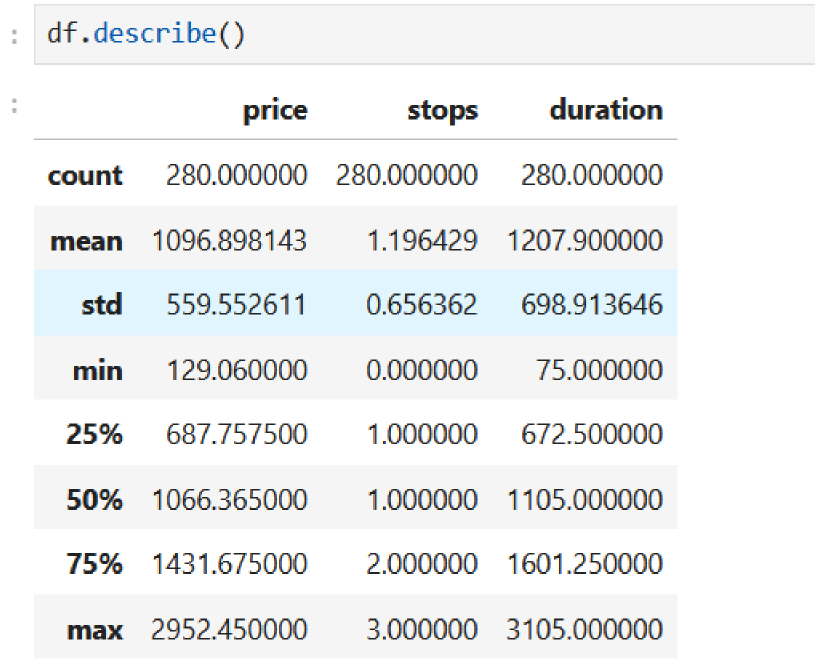


```python
[ ]: #saving the result
     df.to_csv("data.csv", index = False)
```

The data after it cleaned

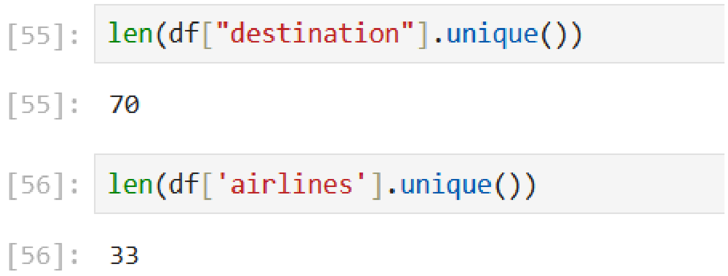
## Analysis & Visualization:

- Use Pandas for EDA (Exploratory Data Analysis).



```
df.describe()
```

| | price | stops | duration |
|---|---|---|---|
| **count** | 280.000000 | 280.000000 | 280.000000 |
| **mean** | 1096.898143 | 1.196429 | 1207.900000 |
| **std** | 559.552611 | 0.656362 | 698.913646 |
| **min** | 129.060000 | 0.000000 | 75.000000 |
| **25%** | 687.757500 | 1.000000 | 672.500000 |
| **50%** | 1066.365000 | 1.000000 | 1105.000000 |
| **75%** | 1431.675000 | 2.000000 | 1601.250000 |
| **max** | 2952.450000 | 3.000000 | 3105.000000 |



```
[55]: len(df["destination"].unique())

[55]: 70

[56]: len(df['airlines'].unique())

[56]: 33
```

### Price:

- Mean: 1,096.90
- Range: 129.06 – 2,952.45

### Stops:

- Most flights have 1 or 2 stops

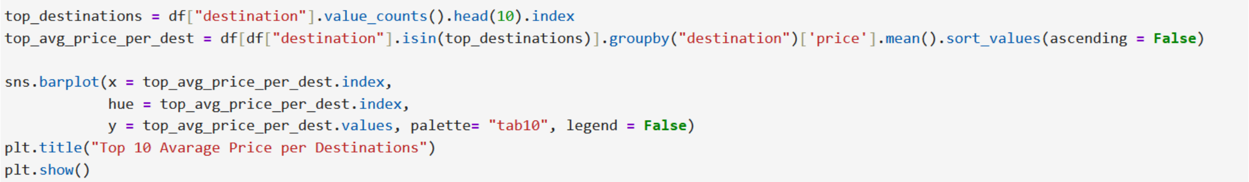
## Destinations:

- 70 unique destinations

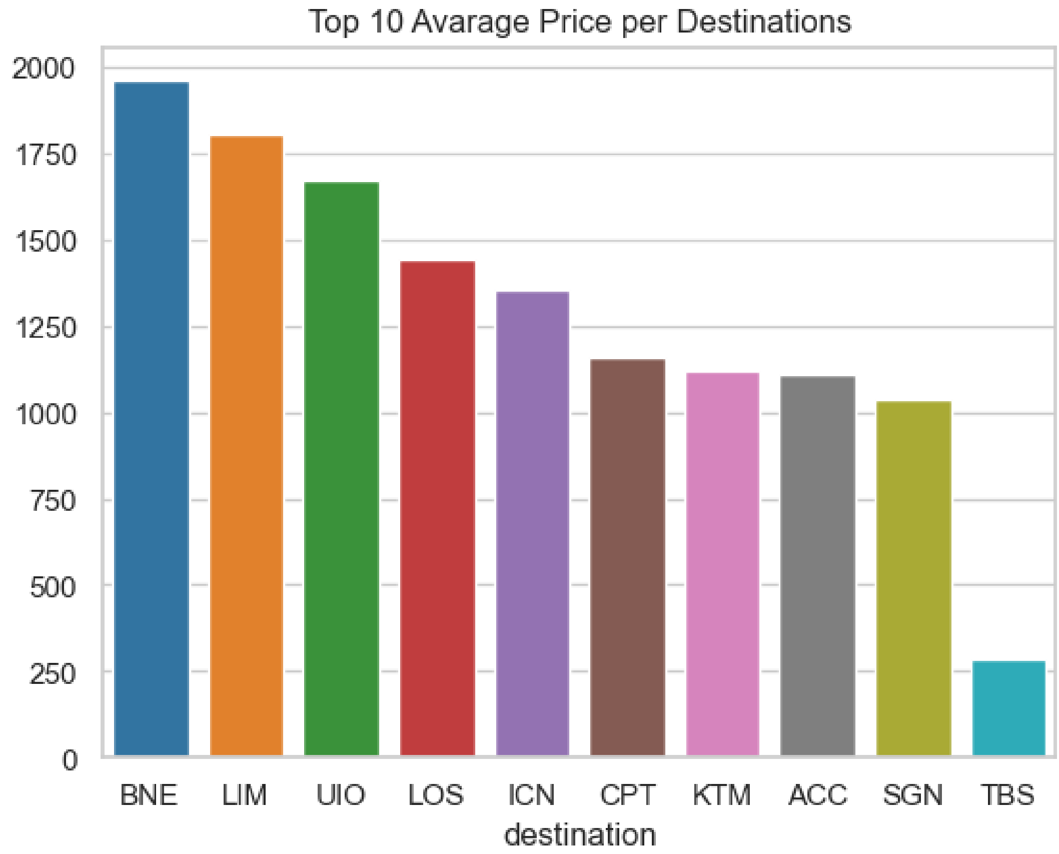
## Airlines:

- 33 unique airlines

- Visualize with Matplotlib and Seaborn.

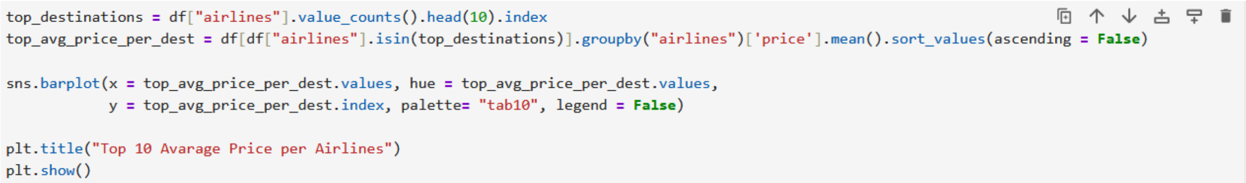


```
top_destinations = df["destination"].value_counts().head(10).index
top_avg_price_per_dest = df[df["destination"].isin(top_destinations)].groupby("destination")['price'].mean().sort_values(ascending = False)

sns.barplot(x = top_avg_price_per_dest.index,
            hue = top_avg_price_per_dest.index,
            y = top_avg_price_per_dest.values, palette= "tab10", legend = False)
plt.title("Top 10 Avarage Price per Destinations")
plt.show()
```

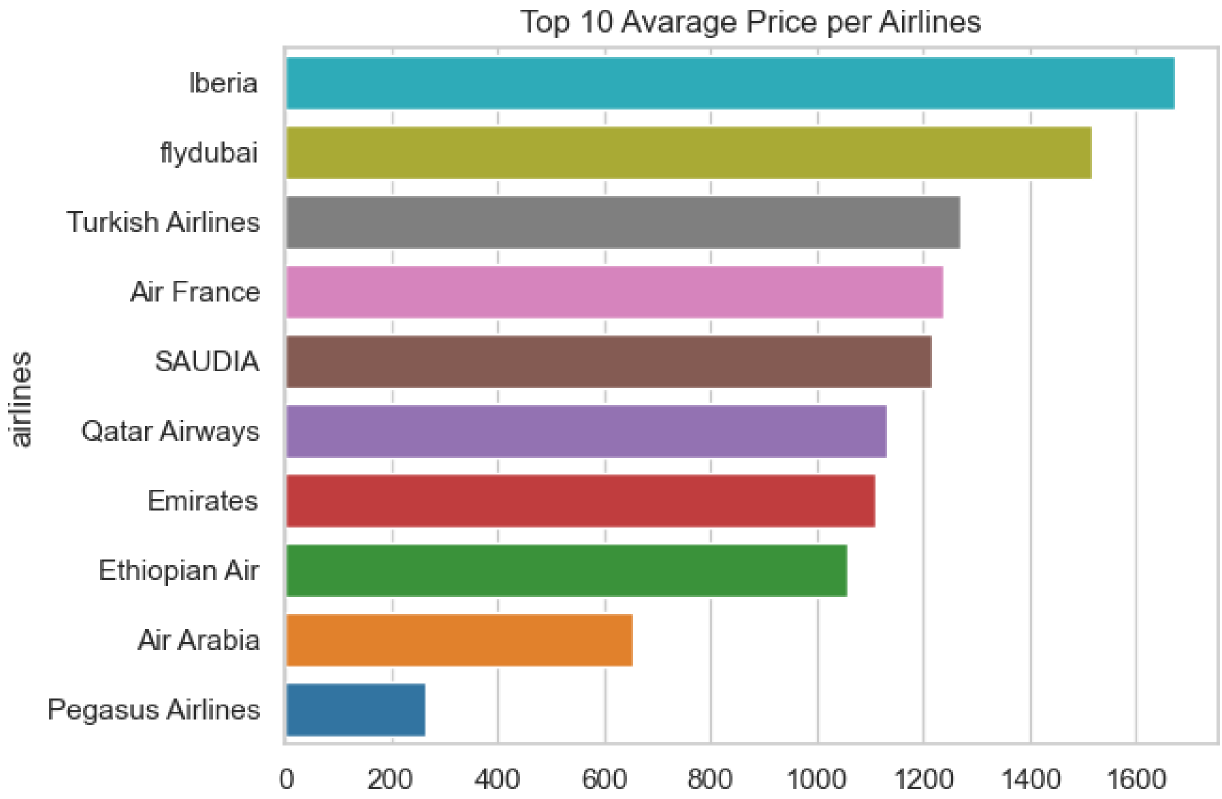


Some destinations (like SGN and TBS) have relatively lower average prices compared to others

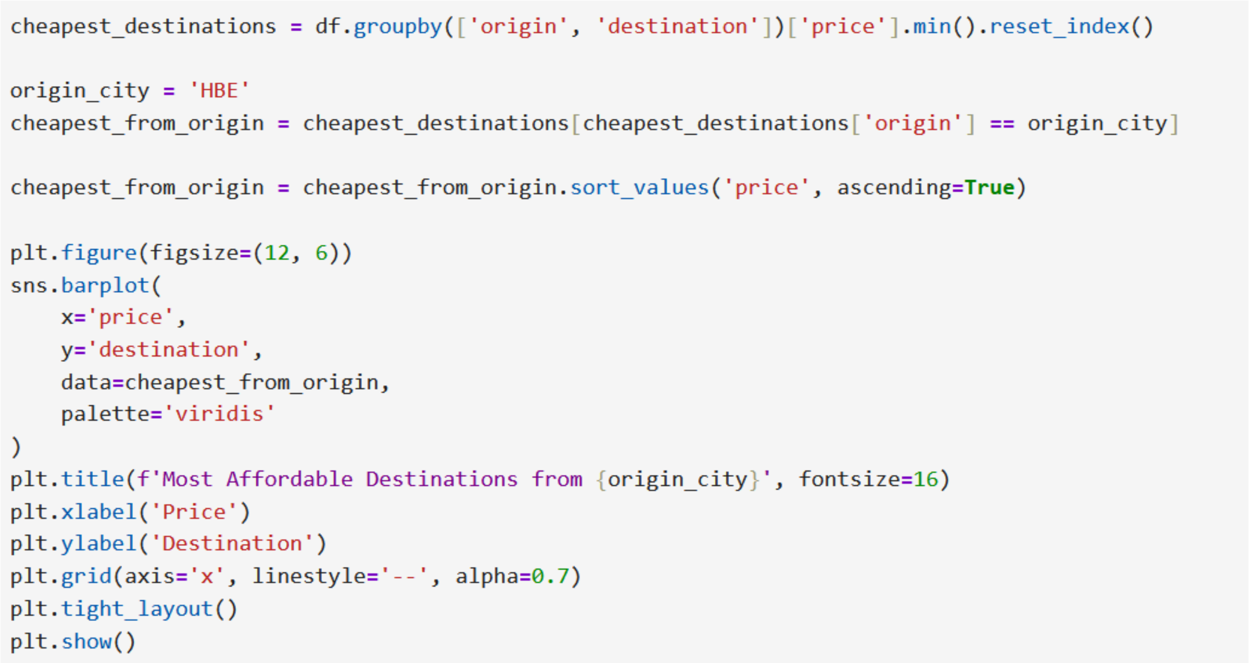
Prices vary significantly by location, likely due to distance or airline competition.



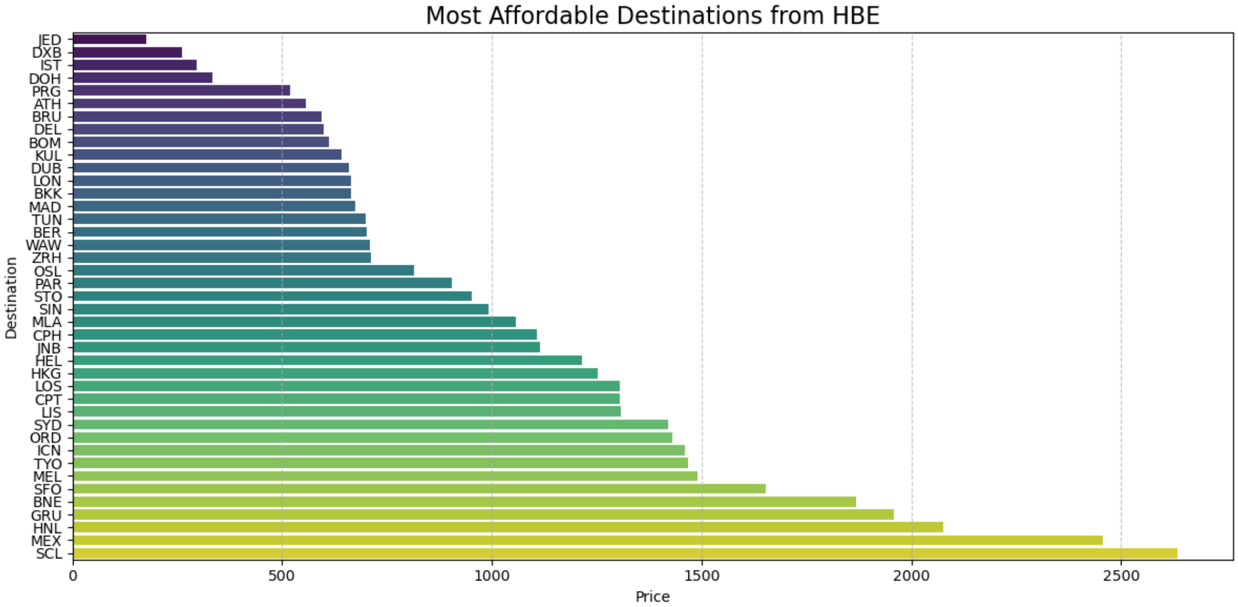
```
top_destinations = df["airlines"].value_counts().head(10).index
top_avg_price_per_dest = df[df["airlines"].isin(top_destinations)].groupby("airlines")['price'].mean().sort_values(ascending = False)

sns.barplot(x = top_avg_price_per_dest.values, hue = top_avg_price_per_dest.values,
            y = top_avg_price_per_dest.index, palette= "tab10", legend = False)

plt.title("Top 10 Avarage Price per Airlines")
plt.show()
```



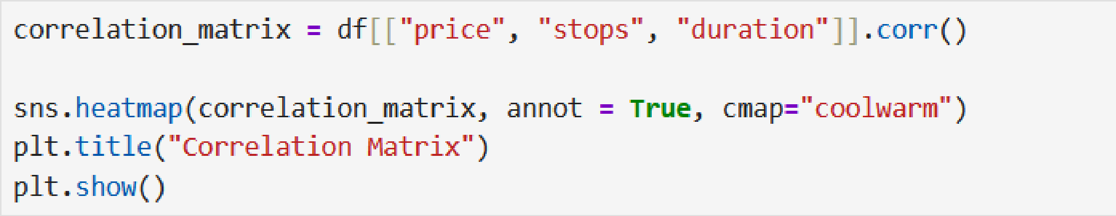
Airlines like Iberia and flydubai show higher average ticket prices.
Budget carriers like Wizz Air and Air Arabia offer lower average prices.



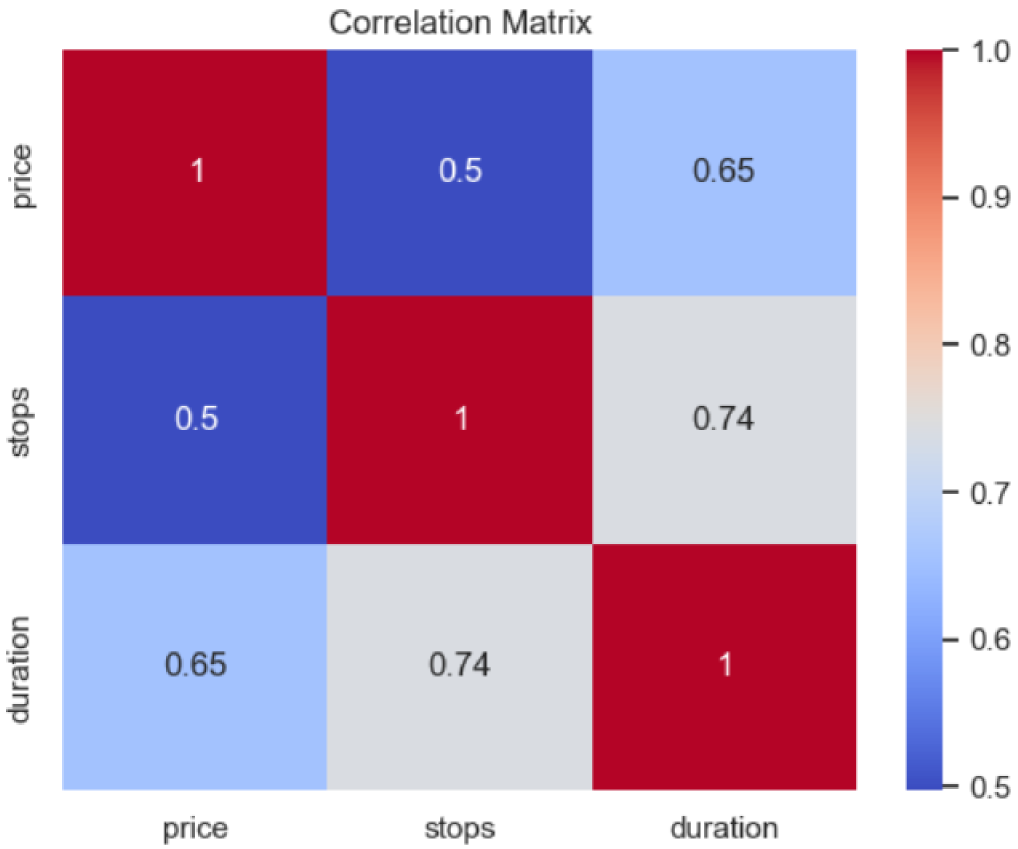
```
cheapest_destinations = df.groupby(['origin', 'destination'])['price'].min().reset_index()

origin_city = 'HBE'
cheapest_from_origin = cheapest_destinations[cheapest_destinations['origin'] == origin_city]

cheapest_from_origin = cheapest_from_origin.sort_values('price', ascending=True)

plt.figure(figsize=(12, 6))
sns.barplot(
    x='price',
    y='destination',
    data=cheapest_from_origin,
    palette='viridis'
)
plt.title(f'Most Affordable Destinations from {origin_city}', fontsize=16)
plt.xlabel('Price')
plt.ylabel('Destination')
plt.grid(axis='x', linestyle='--', alpha=0.7)
plt.tight_layout()
plt.show()
```



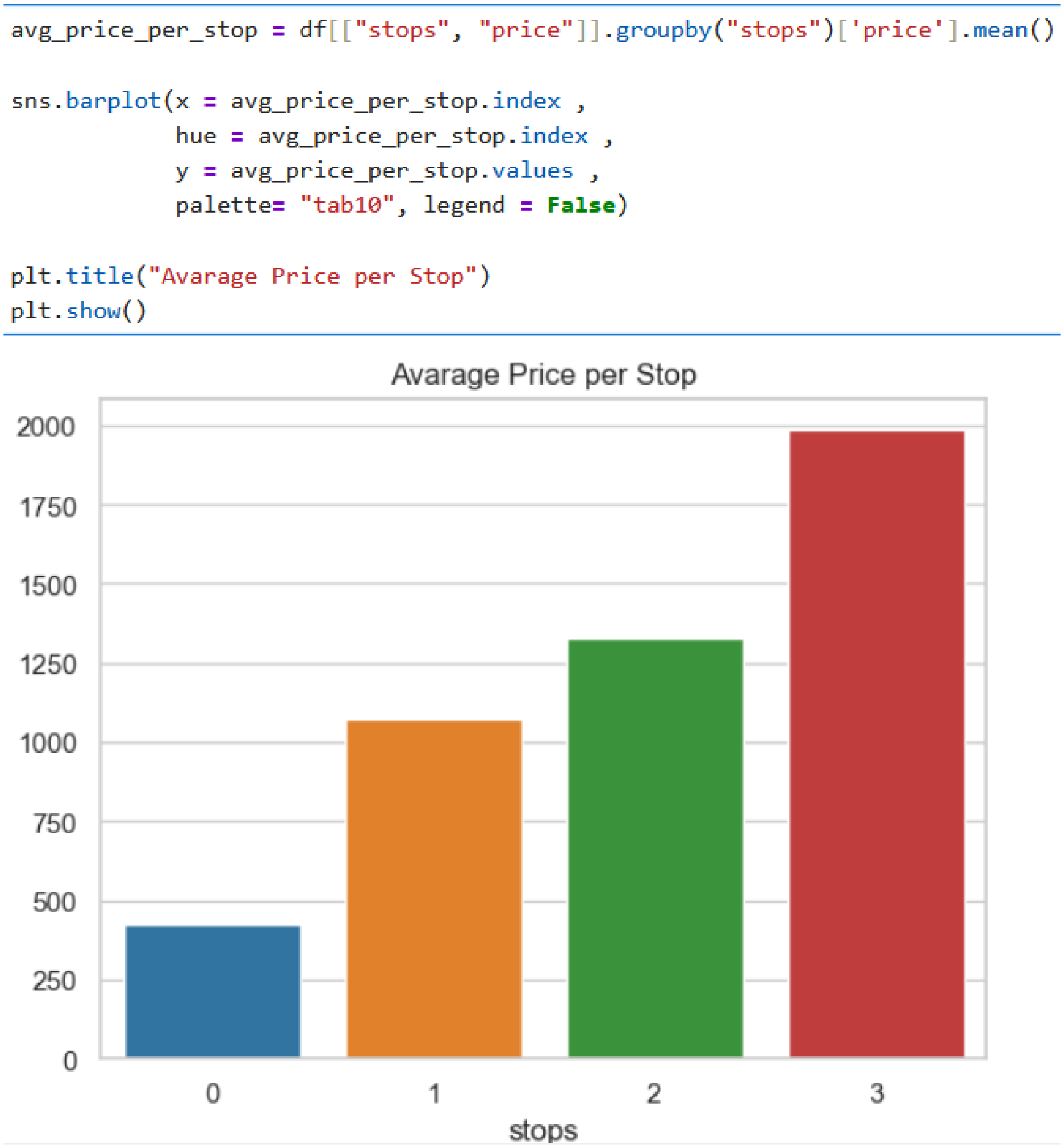
The visualization above shows — destinations like Jeddah, Dubai, and Istanbul are the most affordable from HBE(Borg El Arab)



```
correlation_matrix = df[["price", "stops", "duration"]].corr()

sns.heatmap(correlation_matrix, annot = True, cmap="coolwarm")
plt.title("Correlation Matrix")
plt.show()
```

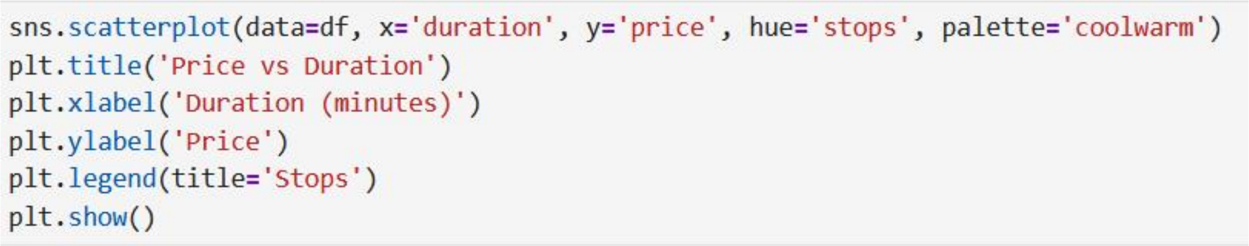


The heatmap reveals a strong correlation between duration and the number of stops, as well as a significant correlation between price and duration.

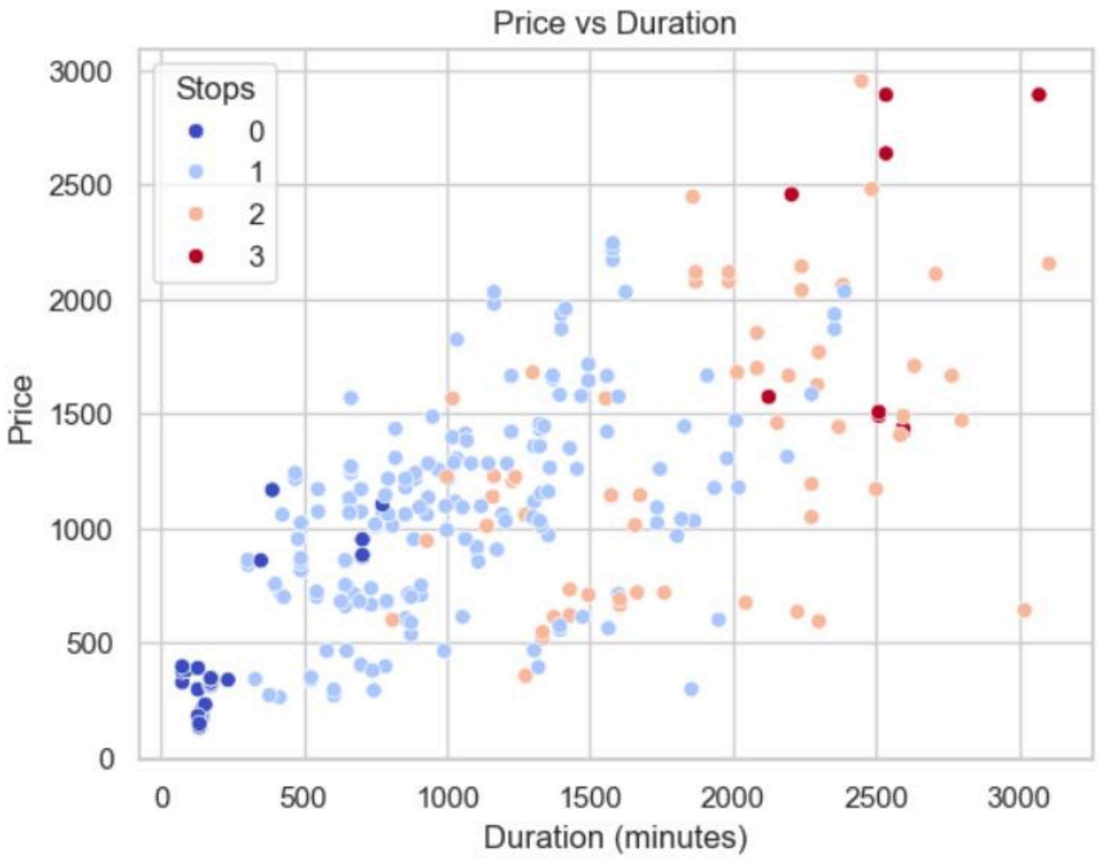


```python
avg_price_per_stop = df[["stops", "price"]].groupby("stops")['price'].mean()

sns.barplot(x = avg_price_per_stop.index ,
            hue = avg_price_per_stop.index ,
            y = avg_price_per_stop.values ,
            palette= "tab10", legend = False)

plt.title("Avarage Price per Stop")
plt.show()
```

increasing the number of stops will increase the price



```
sns.scatterplot(data=df, x='duration', y='price', hue='stops', palette='coolwarm')
plt.title('Price vs Duration')
plt.xlabel('Duration (minutes)')
plt.ylabel('Price')
plt.legend(title='Stops')
plt.show()
```



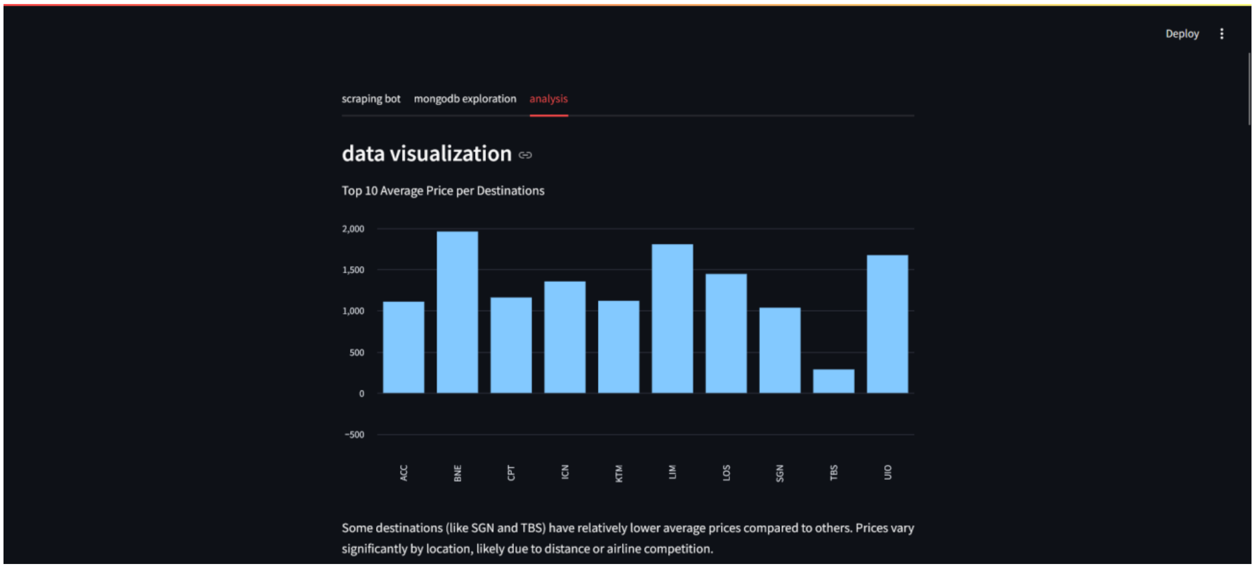
How do prices vary by day of the week or booking time?

Could not answer because the data all in the same week

Are there noticeable seasonal trends?

Could not answer because there is not historical data

- Build interactive dashboards with Streamlit.

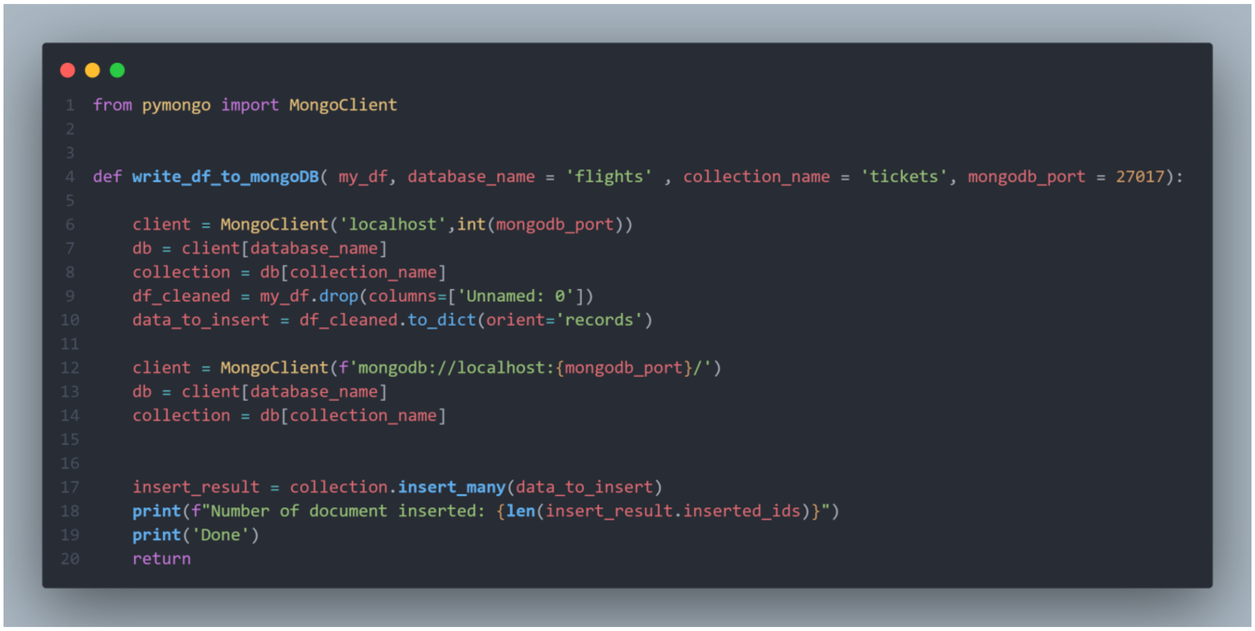


data visualization

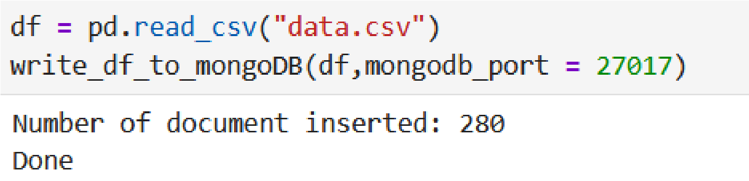
Top 10 Average Price per Destinations

Some destinations (like SGN and TBS) have relatively lower average prices compared to others. Prices vary significantly by location, likely due to distance or airline competition.

## Data Storage:

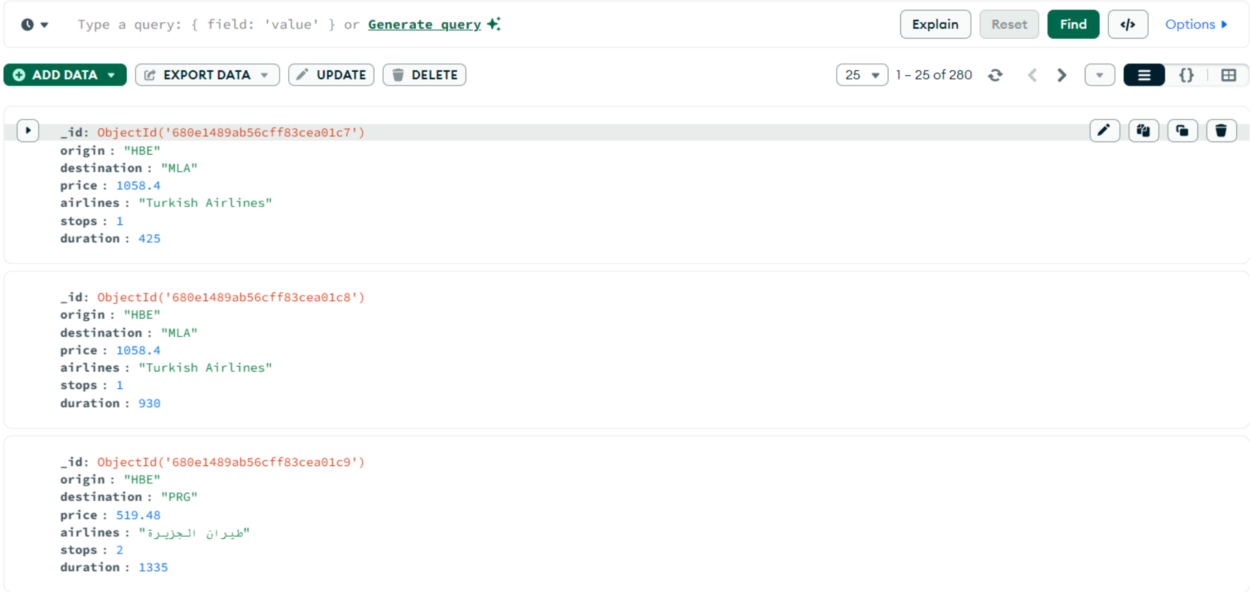


```python
from pymongo import MongoClient


def write_df_to_mongoDB( my_df, database_name = 'flights' , collection_name = 'tickets', mongodb_port = 27017):

    client = MongoClient('localhost',int(mongodb_port))
    db = client[database_name]
    collection = db[collection_name]
    df_cleaned = my_df.drop(columns=['Unnamed: 0'])
    data_to_insert = df_cleaned.to_dict(orient='records')

    client = MongoClient(f'mongodb://localhost:{mongodb_port}/')
    db = client[database_name]
    collection = db[collection_name]


    insert_result = collection.insert_many(data_to_insert)
    print(f"Number of document inserted: {len(insert_result.inserted_ids)}")
    print('Done')
    return
```



```python
df = pd.read_csv("data.csv")
write_df_to_mongoDB(df,mongodb_port = 27017)
```

```
Number of document inserted: 280
Done
```



```
_id: ObjectId('680e1489ab56cff83cea01c7')
origin : "HBE"
destination : "MLA"
price : 1058.4
airlines : "Turkish Airlines"
stops : 1
duration : 425

_id: ObjectId('680e1489ab56cff83cea01c8')
origin : "HBE"
destination : "MLA"
price : 1058.4
airlines : "Turkish Airlines"
stops : 1
duration : 930

_id: ObjectId('680e1489ab56cff83cea01c9')
origin : "HBE"
destination : "PRG"
price : 519.48
airlines : "طيران الجزيرة"
stops : 2
duration : 1335
```