

Overview of Final Project

By: Bryan Hlavin and Tommaso Fazzi




JEM207: Data Processing in Python

5 February, 2025

Introduction

The goal of this final project was to scrape every medal from every Olympic games (summer & winter) and insert them into a dataframe. The data frame would have a list of all countries that were participants in the olympics, as well as their bronze, silver, gold, and total medal results. These results could be found on the official olympics database website. Upon inspection of all elements of the website, it was found out that medal counts for each specific type (gold, silver, bronze, total) were all using different identifiers. The scraping program that was made in Python therefore was able to easily sift out exactly where in the “soup” to look for the right medal, and was able to properly build a tally for a dataframe.

Paris 2024

Team	Gold	Silver	Bronze	Total
 AIN	1	3	1	5
 Albania	-	-	2	2
 Algeria	2	-	1	3

Sample from the 2024 Paris Results Website

```
<div class="Medal-styles__wrapper-sc-645148e1-0 fEoULw" data-cy="medal-module" data-medal-id="gold-medals-row-1" title="Gold">flex</div>  
<span class="Medal-styles__Medal-sc-645148e1-1 jFzKKI" data-cy="medal-main">flex</span>  
<span class="OcsText-styles__StyledText-sc-bf256156-0 cJPVfu text--sm-body" data-cy="ocs-text-module">1</span>  
</span>
```

Snippet of Code Leading to the Gold Medal Count for “AIN” at the Paris 2024 Olympics

Once done with the scraping and inserting of medals into the dataframe, it is then up to the user to select which country the user wants to plot medals for over time. Once the user has imputed a valid Olympic year (and season), the medals are then counted and graphed. The graphs can come in one of two fashions. The first fashion is where the user can ask for a specific country and a specific year of the Olympics. The second fashion is where the user can ask for a specific olympics, winter or summer, and then the results for the top 10 medalists for that Olympics are graphed.

Scraping the Data

The scraping of the data begins at the beginning of running the python code. Once run, a list appears of all the medals that were scraped. The list of locations and years of Olympic games are from the official Olympics database website, which lists out all the medals. In order to properly read in the locations and years into a correct URL for scraping, a JSON file was created to specifically hold the seasons, years, and locations of each Olympic games.

```

{
  "2020": {
    "city": [ "tokyo" ],
    "seasons": [ "summer" ]
  },
  "2022": {
    "city": [ "beijing" ],
    "seasons": [ "winter" ]
  },
  "2024": {
    "city": [ "paris" ],
    "seasons": [ "summer" ]
  }
}

```

Section of Olympic Games JSON File

Once done, a loop was made to collect the data of each website scrape and append into the same dataframe. Text was taken directly from each JSON selection in the loop and then the text for year and location were added to a premade URL. Parsing was then able to be completed in this manner. This was done over the whole length of the JSON file. The parsing was done with the BeautifulSoup functions as well.

The resulting data was not only to be added to a dataframe, but also to a JSON file as well in the event the user wanted to see the resulting data without having to run the Python code. The below image is a sample of the outputted JSON file.

```

{
  "tokyo-2020": {
    "year": "2020",
    "city": "tokyo",
    "season": "summer",
    "results": "{ 'Afghanistan': { '2020_gold': nan, '2020_silver':
  },
  "beijing-2022": {
    "year": "2022",
    "city": "beijing",
    "season": "winter",
    "results": "{ 'Afghanistan': { '2022_gold': nan, '2022_silver':
  },
  "paris-2024": {
    "year": "2024",
    "city": "paris",
    "season": "summer",
    "results": "{ 'AIN': { '2024_gold': '1', '2024_silver': '3', '20
}

```

Outputted JSON File with Results

```

def scrape_olympics_websites(df_olymp):
    print(f'Proceeding to scrape all olympics websites....')
    print(' ')

    # Making an empty json dictionary
    with open("olymp_database.json", 'r') as f:
        olymp_data = json.load(f)
        olymp_data = {}

    # Splitting the URL to prevent IDE issues
    url_1 = 'https://'
    url_2 = 'olympics.com/en/olympic-games/'

    # Making a dataframe from the dictionary (2 rows, 38 columns for every olympics)
    df_olymp.index = df_olymp.index.astype(int)
    df_filter1 = df_olymp.loc[pd.to_numeric(1896):pd.to_numeric(2024)]

    df_total_results = pd.DataFrame(columns = ['Country'])

    # Loop through each year's olympics
    for index, city in enumerate(df_filter1['city']):
        for loc in city:
            location_in_series = loc
            year_in_series = df_filter1.index[index]
            season = df_filter1['seasons'].iloc[index][city.index(loc)]
            print(f"Getting results: {year_in_series} - {location_in_series} - {season}")
            url_specific = f"{url_1}{url_2}{location_in_series}-{year_in_series}/medals?displayAsWebViewLight=true&displayAsWebView=true"

            # Gathering countries and medal counts
            soup_countries_list, soup_medals_list = bs_scrape(url_specific, location_in_series)

```

Code with URL Operations to Properly Read JSON Names and Years

Dataframe Operations

In order to properly insert the data into the data frame, the matching of columns from the data scraping to the dataframe had to be made perfectly for each iteration of the loop. The following code demonstrates the ordering of the code and columns for each year's medals. Similarly ordered code was used in order to place the results in the results JSON file.

```

# Placing values in dataframe
df_total_results = place_values_in_dataframe_and_json(soup_countries_list, soup_medals_list, year_in_series, df_total_results, season)
df_results_json = df_total_results.loc[:, ['Country', f'{year_in_series}.gold_{season}', f'{year_in_series}.silver_{season}', f'{year_in_series}.bronze_{season}', f'{year_in_series}.total']]
df_results_json = df_results_json.where(df_results_json != '-', '0')
df_results_json = df_results_json.where(df_results_json != '', np.nan)
df_results_json = df_results_json.sort_values('Country')
df_results_json = df_results_json.to_dict(orient = 'index')

# Placing values in json file
json_database_data = {
    f'{location_in_series}-{year_in_series}':
        {
            "year": f'{year_in_series}',
            "city": f'{location_in_series}',
            "season": f'{season}',
            "results": f'{df_results_json}'
        }
}

olymp_data.update(json_database_data)

with open("olymp_database.json", 'w') as f:
    json.dump(olymp_data, f, indent=4)

```

Placement of Values in the Dataframe

Prompting the User

Once done with all the scraping and the data collection, the user must be asked which country's plot they wish to see from the data. A bunch of logical if statements had to be made in order to ensure that the user chose a country correctly, or if they simply wanted to. This includes fail safes to prevent the user from selecting a country that doesn't exist. With a proper selection, the user can then see the graph of all the medals that the country has achieved through time.

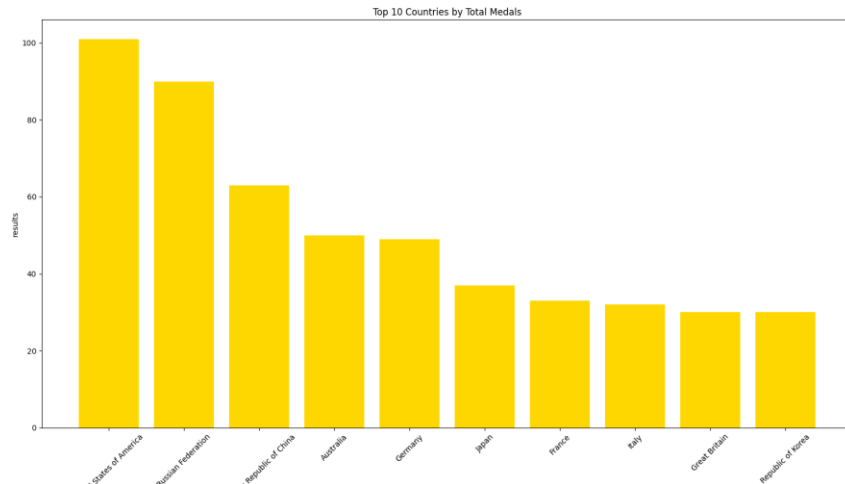
```

Do you wish to see a country's medal statistics, or the top 10 results for a specific Olympic games? Type country or year for either.
Enter request: country
Enter Country: france
Type 's' or 'summer' for the Summer Olympics, and 'w' or 'winter' for Winter Olympics
Enter season: summer
You have selected the summer olympics for france.

```

Plotting the Data

Depending on what data the user wants, the code written takes a different form. If the user wants the top ten countries in the Olympics for a specific game, then the user can simply request that after the data frame has been made.



Plot of Top 10 Results from the 2004 Olympic Games (Summer)

If the user asked for a specific country's performance over time, then all the years a country has participated in the Olympics can be shown.

