Declaration of Authorship

We hereby declare

- that I have written this essay without any help from others and without the use of documents and aids other than those stated above;
- that I have mentioned all the sources used and that I have cited them correctly according to established academic citation rules;
- that I have acquired any immaterial rights to materials I may have used such as images or graphs, or that I have produced such materials myself;
- that the topic or parts of it are not already the object of any work or examination of another course unless this has been explicitly agreed on with the faculty member in advance and is referred to in the thesis;
- that I will not pass on copies of this work to third parties or publish them without the University's written consent if a direct connection can be established with the University of St.Gallen or its faculty members;
- that I am aware that my work can be electronically checked for plagiarism and that I hereby grant the University of St.Gallen copyright in accordance with the Examination Regulations in so far as this is required for administrative action;
- that I am aware that the University will prosecute any infringement of this declaration of authorship and, in particular, the employment of a ghostwriter, and that any such infringement may result in disciplinary and criminal consequences which may result in my expulsion from the University or my being stripped of my degree."

Date and signature:

December 22nd 2022

Petr Matejovsky

Philipp Thießen

Mattia Briganti

By submitting this academic paper, we confirm through my conclusive action that I am submitting the Declaration of Authorship, that I have read and understood it, and that it is true.

School of Management, Economics, Law,
Social Sciences, International Affairs, and Computer Science

Skills: Programming with Advanced Computer Languages

University of St. Gallen

7,789,1.00 Skills: Programming with Advanced Computer Languages

Dr. Mario Silic

Mattia Briganti (6299-1200-2725-2769)

Philipp Thießen (22-605-026)

Petr Matejovsky (22-603-393)

Submission Date:

December 22th 2022

Table of Contents

1. Introduction

1.1 Overview

When the Wright brothers conducted the first real flight, mankind wondered how far and how high can we fly. Rapid development in the construction of airplanes and progress in engineering eventually led to the development of the first rockets (V1 and V2). After the Second World War, scientists from both the United States and the Soviet Union rushed to design advanced rockets that were able to reach orbit. The competition between the two world-leading superpowers finally transformed into a full space race that culminated in 1969 with the American Neil Armstrong being the first person to walk on the Moon (Mullen, 2010). After that came a period of stagnation during which relatively little progress was made in colonizing the space. However, the last decade has brought a renewed interest in "sending people to stars". Private enterprises are seeking to commercialize space travel and major world powers are seriously considering colonizing Mars (Minter, 2022). There are now several space launches conducted each month and we think that it can be beneficial for anyone to have an overview of not only what launches took place in the past but also what are the upcoming space flights. For that reason, we decided to develop a program in Python that scraps data from a given website and presents them to the user in an easily understandable chart. In short, the purpose of this project is to provide an easy-to-use tool to obtain data from a website, store it and manipulate it.

Web scraping is a technique used to extract data from websites. It involves using a computer program to sift through a website's HTML code, looking for specific pieces of information, and then organizing the data into a useful format (Hillier, 2021). To perform web scraping, we write a computer program that sends an HTTP request to a website's server, requesting the HTML code for the page. The server responds by sending the HTML code to the program, which then parses the code and extracts the desired data.

The program can use a variety of techniques to parse the HTML code and extract the data, such as searching for specific tags or attributes or using regular expressions to match specific patterns in the code. The extracted data is then organized into a data structure, such as a table or a list, and can be saved to a file or used in other applications (Hillier, 2021). Normally, web scraping is often used to collect large amounts of data from websites, such as product information from online stores, or real estate listings from property websites. It can also be used to automate tasks, such as filling out online forms or performing searches.

Generally, there are some advantages and disadvantages to web scraping. It can be a powerful tool for extracting data from websites, but it can also be abused. The main advantage is speed, as data scraping programs allow for faster collection of data than would be otherwise possible if one did everything manually. It is also very cost-effective and once successfully implemented, the program can be scaled with minimal problems and changes to the code (Lekh, 2022). There are also several disadvantages. Web scraping programs need perpetual maintenance of the code to work properly. Furthermore, one has to remember that data extraction is not the same as data analysis. Simply scraping data from a website is not enough and a good program must have some method to present the data in an understandable format (Lekh, 2022). Many websites have terms of service that prohibit the use of web scraping, and some have implemented measures to block or detect scraping programs.

1.2 Website Description

The website used for the purpose of this project is: https://nextspaceflight.com/launches/. This site provides information about upcoming space launches. The information contained lists the dates, times, and locations of upcoming launches, as well as the names of the launch vehicles and payloads. This website also provides information about the purpose of the launch and the status of the mission. Furthermore, the website is regularly updated with new launch information as it becomes available.

The reason why we picked this website specifically out of all available websites that provide information on space travel is because it has no scraping protection which are measures taken by websites to prevent or detect the use of web scraping. These measures can include blocking requests from known scraping programs, requiring users to solve captchas before accessing the website's content, or monitoring access patterns to identify and block suspicious activity. Scraping protections are often implemented to prevent the abuse of a website's content. Website owners may not want their content to be scraped and republished on other sites, or they may not want their servers to be overloaded by excessive scraping. Scraping protections can also be used to prevent malicious actors from using scraping to collect sensitive information, such as login credentials or personal data.

Lastly, nextspaceflight.com has a relatively simple content structure, i.e. a simple structure of the HTML document. There are several basic tags that are needed to create a simple html structure (Hobson, 2022). We can inspect this structure by using a browser inspector. Then it is possible to confirm that the website has been coded with a series of rather basic elements, represented by tags, which enables us to create an efficient web data scraping program.

1.3 Libraries Used

Several Python libraries were used for the purpose of data scraping and then visualizing the data in a digestible format.

UrlLib: The Python UrlLib library is a collection of modules that provide functions for working with URLs. The library contains a set of tools for opening URLs, sending HTTP requests, and handling responses. It also includes tools for parsing URLs and extracting information from them, as well as tools for encoding and decoding URLs.

Pandas: Pandas is a powerful and flexible data manipulation and analysis library for Python. It provides a wide range of tools for working with data, including tools for reading and writing data to various formats, filtering and sorting data, and performing statistical analysis. Pandas is built on top of the popular NumPy library and provides a rich set of functions and methods for working with data.

Numpy: Numpy is a Python library for working with numerical data. It provides a powerful array object for representing and manipulating data, and a large collection of mathematical functions for working with arrays. Numpy is widely used in scientific and mathematical computing and is an essential library for data science and machine learning in Python.

BeautifulSoup4: BeautifulSoup is a Python library for parsing HTML and XML documents. It provides a simple and elegant interface for navigating, searching, and modifying the structure of an HTML or XML document. Beautiful Soup is typically used in web scraping applications, where the goal is to extract data from a website's HTML code.

Matplotlib: Matplotlib is a popular data visualization library for Python. It provides a wide range of plotting and charting tools, allowing you to create static, animated, and interactive visualizations of your data. Matplotlib can be used to create a variety of different plots, including scatter plots, line plots, bar charts, histograms, and many more.

Display: The display module in Python provides tools for displaying data in a variety of formats, including text, images, and videos. It is often used in conjunction with other libraries, such as Matplotlib, to display visualizations of data. The display module can be used in Jupyter Notebook and other interactive environments, as well as in standalone scripts.

2. Installing and Running the Code

2.1 Installing the Code

We used the PyCharm software to run this program. Installing a Python code using PyCharm is a straightforward process that allows user to easily develop and run Python projects. PyCharm is a popular integrated development environment (IDE) for Python that provides a range of useful features and tools for writing, testing, and debugging code. Alternatively, the user can use any other program such as Atom. However, we recommend using PyCharm because it enables an easy method to install the necessary libraries.

The list of steps to install and run the Python code using PyCharm is following:

1. In the first step, the user downloads and installs PyCharm on a computer.
2. The user opens the PyCharm and creates a new project by selecting "Create New Project" from the welcome screen.
3. In the project setup window, the user selects the location where he/she wants to save the project and chooses a project interpreter. This is basically a virtual machine that emulates a physical computer. By default, the user can select the packaged interpreter that comes with PyCharm.
4. Once the project is set up, the user can run the web scraping program by going to File > Open and selecting the main.py file that is part of this zip package.
5. After opening the file, PyCharm should show the user all the lines of code along with the comments market by #.
6. When the user is ready to run your program, he or she can select the Run option from the top menu or use the keyboard shortcut Shift+F10. This will run the data scraping program and show the output in the console window at the bottom of the PyCharm window.

2.2 Running main.py

The code is stored on the main.py program. After successfully opening it in PyCharm or any other IDE software, the user can run the program. As described in the last step above, the Run option will run the entire code. If the program runs successfully, it will provide output in the terminal window. If there are any errors in the program, they will be displayed in the terminal window, along with a traceback showing where the error occurred in the code.

2.3 Opening .pdf and .csv files

The data collected through web scraping use two functions: scrape_past_launches and scrape_future_launches. These functions use additional support functions to extract data from web pages and save it into the .csv file which then is used to generate charts to visualize the data. After running the program, a .csv file is created and stored in the project folder.

The function scrape_page is used to access the web page of interest, scrape through the html code, and extract and save launch information. It also calls the function get_detailed_info to gather more detailed information about each launch in a similar manner. Once the information has been collected, it is saved in a .csv file. The read_csv function is used to simplify access to the .csv file.

The scrape_past_launches and scrape_future_launches functions work similarly, but they extract either past or future launches, respectively. These functions call the scrape_page function to scrape all of the web pages specified by the user. It takes an argument, either "Future" or "Past", which determines which file to access. If the argument is "Future", the function will return the .csv file containing data on launches from 2022 and beyond. If the argument is "Past", the function will return the launches_until_2022 file, which contains data on launches up until 2022.

Lastly, a pdf file is created which provides a visual output on the number of past or future launches conducted by a given country.

3. Code Overview

3.1 Introduction

This section will give an overview of the main parts of the code. First, the overall structure will be explained followed by a more detailed explanation regarding its functionalities. Explanations are supported by extracts from the main code and an external website. For a detailed explanation of certain lines of code please see the file main.py.

3.2 Program's aim

The objective of this program is to scrape an HTML webpage and manipulate data obtained in this way. For this reason, we used a website with a simple and repetitive HTML structure with carefully scripted CSS in it, in order to iterate, hence scrape, easily data. In addition, the website we chose did not have any scraping protection active.

3.3 Structure

The code is structured in six main parts. First, necessary (1) *libraries* are imported. Second, (2) *global variables* are defined. The base URL of Next Spaceflight is defined as base_url. Based on this constant later specific pages of the website can be defined such as these for upcoming or past launches. Next, (3) *support functions* are defined which will be used by the actual scraping functions of past and future launches. The (4) *scraping functions* scrape_past_launches and scrape_future_launches use the previously mentioned support functions to scrape the web pages and save the data extracted into a CSV file. Lastly, the (5) *chart generation* runs the respective scraping function and generates various charts to visualize the extracted data.

(1) Libraries

Libraries are urllib, pandas, numpy, BeautifulSoup and matplotlib. Urllib and BeautifulSoup support website scraping and HTML processing while pandas, numpy and matplotlib aid in processing and visualizing the obtained data.

(2) Global Variables

```
50      base_url = "https://nextspaceflight.com"
```

The URL is defined as a constant base_url such that later specific parts of the website can be accessed more easily. See an example below:

```
79          url = base_url + "/launches/?page={0}".format(page)
```

(3) Support functions

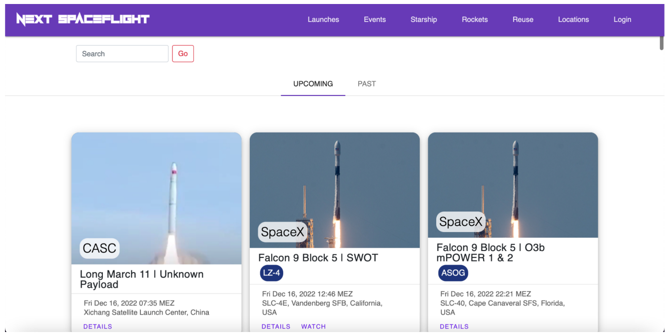
```
# ********************************************************************************************************
# ***********************************                  ***************************************************
# ***********************************   SUPPORT FUNCTIONS   ***************************************************
# ***********************************                  ***************************************************
# ********************************************************************************************************

# 1) scrape_page:
# This Function first downloads the HTML page and loops through the 30 the displayed launches (30 launches
# per page) including the link to the 'detail' page of each launch.
#   As Arguments, the function takes the page (integer, here not defined as int) and a flag used to scrape
#   either "Future" or "Past" launches
#   It returns a DataFrame with the scraped launches in it.
def scrape_page(page, future=False):...


# 2) get_detailed_info
# This Function opens the page including the data of a specific launch, stores the elements and returns
# the findings.
#   As Arguments, the function takes the rocket launch id (integer)
#   It returns a panda DataFrame with the detailed information of a launch in it as a dictionary and their
#   relative unique id.
def get_detailed_info(rocket_id):...


# 3) read CSV
# This Function reads the CSV file created by the scraping functions and returns data in a DataFrame
# structure
#   As Arguments, the function takes a String indicating whether past or future launches have to be read
#   It returns a panda DataFrame with the detailed information of a launch read from the csv file
def read_csv(horizon):...
```

scrape_page The overall functionality of the function scrape_page is to crawl a given page and return the data in a pandas Dataframe. Depending on the arguments the function either crawls past or upcoming launches. Irrespective of the launches to crawl, one page covers 30 entries in a 3x10 layout, through which the function loops. The screenshot shows the first page of upcoming launches.



The URL of the page has the format (https://nextspaceflight.com/launches/?page=1&search=). Which is equal to the global constant base_url defined earlier plus the specific path and page number. The argument "page" is used to define the number of pages we are scraping and is here initialized.

```
if future:
    url = base_url + "/launches/?page={0}".format(page)
else:
    url = base_url + "/launches/past/?page={0}".format(page)
```

The Urllib library function request is used to access the previously defined URL.

```
html = urllib.request.urlopen(url)
```

Following, the bs4 library is used to parse through the html code. The function looks for the div class "mdl-grid" and inside that, for the div class "mdl cell". The grid locates the 3 by 10 arrangement of launches shown earlier to locate each mdl cell representing one launch.

```
<div class="mdl-grid" style="margin-top: 5px; width: auto; justify-content: center"> == $0
  <div class="mdl-cell mdl-cell--6-col" style="width: auto">…</div>
  <div class="mdl-cell mdl-cell--6-col" style="width: auto">…</div>
  <div class="mdl-cell mdl-cell--6-col" style="width: auto">…</div>
  <div class="mdl-cell mdl-cell--6-col" style="width: auto">…</div>
  <div class="mdl-cell mdl-cell--6-col" style="width: auto">…</div>
  <div class="mdl-cell mdl-cell--6-col" style="width: auto">…</div>
  <div class="mdl-cell mdl-cell--6-col" style="width: auto">…</div>
  <div class="mdl-cell mdl-cell--6-col" style="width: auto">…</div>
  <div class="mdl-cell mdl-cell--6-col" style="width: auto">…</div>
  <div class="mdl-cell mdl-cell--6-col" style="width: auto">…</div>
  <div class="mdl-cell mdl-cell--6-col" style="width: auto">…</div>
  <div class="mdl-cell mdl-cell--6-col" style="width: auto">…</div>
  <div class="mdl-cell mdl-cell--6-col" style="width: auto">…</div>
  <div class="mdl-cell mdl-cell--6-col" style="width: auto">…</div>
  <div class="mdl-cell mdl-cell--6-col" style="width: auto">…</div>
  <div class="mdl-cell mdl-cell--6-col" style="width: auto">…</div>
  <div class="mdl-cell mdl-cell--6-col" style="width: auto">…</div>
  <div class="mdl-cell mdl-cell--6-col" style="width: auto">…</div>
  <div class="mdl-cell mdl-cell--6-col" style="width: auto">…</div>
  <div class="mdl-cell mdl-cell--6-col" style="width: auto">…</div>
  <div class="mdl-cell mdl-cell--6-col" style="width: auto">…</div>
  <div class="mdl-cell mdl-cell--6-col" style="width: auto">…</div>
  <div class="mdl-cell mdl-cell--6-col" style="width: auto">…</div>
  <div class="mdl-cell mdl-cell--6-col" style="width: auto">…</div>
  <div class="mdl-cell mdl-cell--6-col" style="width: auto">…</div>
  <div class="mdl-cell mdl-cell--6-col" style="width: auto">…</div>
  <div class="mdl-cell mdl-cell--6-col" style="width: auto">…</div>
  <div class="mdl-cell mdl-cell--6-col" style="width: auto">…</div>
  <div class="mdl-cell mdl-cell--6-col" style="width: auto">…</div>
  <div class="mdl-cell mdl-cell--6-col" style="width: auto">…</div>
</div>
```

```
soup = BeautifulSoup(html, "html.parser")
table = soup.find('div', {'class': 'mdl-grid'})
cells = table.find_all('div', {'class': 'mdl-cell'})
```

In a similar fashion the id, date of launch, title, company, the base of launch, and link to detailed information are extracted. For convenience, these are saved in a dictionary named launch.

```
launch = {
    'id': rocket_id,
    'date': date,
    'title_1': title_1,
    'title_2': title_2,
    'company': company,
    'base': base,
    'link': link,
}
```

We now have the launch information but we miss the details of each one of them. These have to be crawled from a different page, hence we need the second support function we will define later. Once obtained, we can then store the crawled information in our lists and proceed to save them in a csv file.

```
# We then proceed to get detailed information for this rocket by calling the get_detailed_info()
# function defined below
detail = get_detailed_info(rocket_id)

# We append the DataFrame (i.e. the indexed dictionaire with detailed info returned by the function 2)
# to the list of DataFrames we defined earlier
page_details.append(detail)
# We also append the launch to the list of dictionaries we created to be able to go through
# them independently
page_launches.append(launch)
```

The variable res is defined as a pandas dataframe containing the page_launches list and the index set to id.

```
res.to_csv("file.csv")
```

The variable is saved in a csv file. Finally, the variable res is returned.

get_detailed_info() when this function is called it returns the launch information as a keyed dataframe including a unique identifier and the list of features. As a reminder, note that the information on each launch is stored on a different page from the one we scraped in the previous function. The link to each page of every launch is each one of the 30 launches (a button) of the main page, hence we will later loop through them and call iteratively this function.

Summarizing what has happened up to this point, the function scrape_page accesses the Url of interest, scraped through the html code finding, and saves launch information. Further, it calls the function

get_detailed_info to extract further information in a similar fashion. Finally, it creates a csv file containing the mentioned data.

read_csv This function takes the time horizon as an argument and thereby simplifies the accessing of the csv file of interest. The argument “Future” returns the csv file containing data on launches from 2022 onwards while “Past” reads the launches_until_2022 file.

```
266         if horizon == "Future":
267             res = pd.read_csv("launches_from_2022.csv", parse_dates=['date'])
268         elif horizon == "Past":
269             res = pd.read_csv("launches_until_2022.csv", parse_dates=['date'])
```

(4) Scraping functions

scrape_past_launches/scrape_future_launches The functions scrape_past_launches/ scrape_future_ launches work the same but extract either future or past launches. The main component of the function is calling the previously defined scrape_page function to scrape all pages the user defined to scrape. The returning res panda dataframes for the pages from 1 to N_PAGES (defined by the user) are saved in the variable res. The variable is saved in a csv file "launches_until_2022.csv"

```
res = pd.concat([scrape_page(page) for page in range(1, N_PAGES + 1)], sort=True)
```

(5) Chart generation

Firstly, the previously explained function scrape_past_launches is run. At this point in time, the scraping begins, which might take some time depending on the number of pages the user indicated.

```
page_scraped = int(input("How many pages you'd like to scrape?
                         " scrape them all"))
scrape_past_launches(page_scraped)
```

Following the functions plot_launches_by_country, plot_launches_byCountryYear, and plot_launches_USAvsRussia are defined. The functions use matplotlib to create the charts. See the example below:

```
488  def plot_launches_USAvsRUSSIA(res):
489      fig, ax1 = plt.subplots(1, 1, figsize=(12, 6))
490      ax2 = ax1.twinx()
491      
492      res['Country'] = res['base'].str.split(", ").apply(lambda x: x[-1])
493      res['year'] = res['date'].dt.year
494      te = res[res['Country'].isin(['USA', 'Russia'])]
495  
496      te.groupby(['year', 'Country']).size().unstack().cumsum().fillna(0).plot(
497          ax=ax2, marker='o', color=['r', 'b'], markersize=10)
498  
499      te.groupby(['year', 'Country']).size().unstack().fillna(0).plot(
500          ax=ax1, color=['r', 'b'], kind='area', alpha=0.1, legend=False, stacked=False)
501  
502      ax1.xaxis.set_label_text("");
503      ax2.yaxis.set_label_text("TotalLaunches");
504      ax1.yaxis.set_label_text("LaunchesperYear");
505      plt.savefig("usa_russia.pdf");
506      return
```

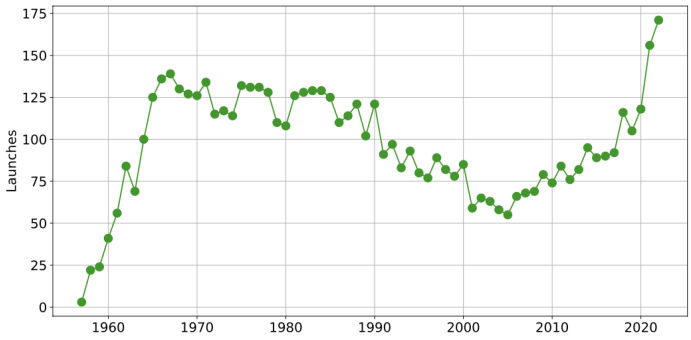
Core program

Finally, the functions are run and the charts are created. These are saved as pdf files in the project folder. The support function read_csv is called based on the future or past.

```
503  res = read_csv("Past")
504  
505  plot_launches_by_country(res)
506  plot_launches_byCountryYear(res)
507  plot_launches_USAvsRUSSIA(res)
```

Output

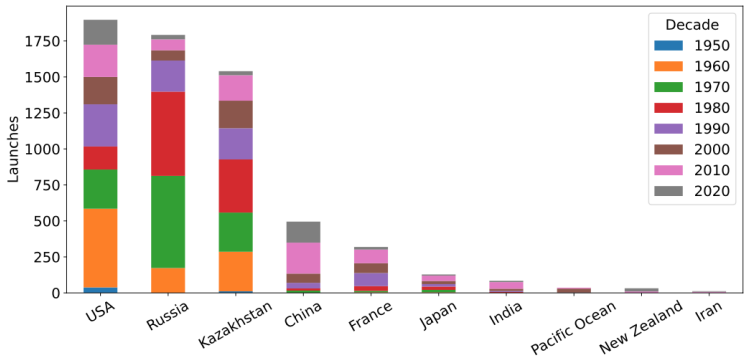
plot_launches_by_country



Different than the title suggests, this chart plots the total rocket launches per year. We see a steep increase in the late 1950s and 1960s as well as around the year 2020. Both can most likely be attributed to advancements in rocket and satellite technology.

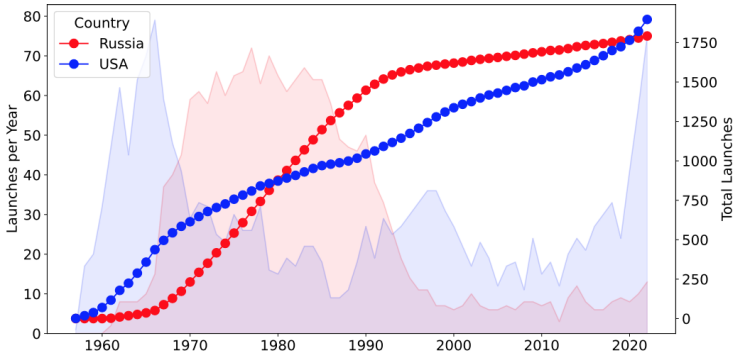
In 1980 about 110 rockets were launched. In the below graphic plot_launches_USAvsRussia we see that there were 70 launches by the two countries. Thus, interestingly about 60% of launches can be linked to the US and Russia alone

plot_launches_byCountryYear



The above chart plots the total launches per country as of 2022 and graphically depicts how many launches can be attributed to which era. It may be surprising that Kazakhstan is in third place, however, this graph only plots the country in which a certain launch has happened. That's due to some advantages Kazakhstan offers. One of the main advantages of launching rockets from Kazakhstan is its location, which allows for rockets to be launched into a variety of orbits, including polar, sun-synchronous, and geostationary orbits.

plot_launches_USAvsRUSSIA



This chart plots both the total launches and launches per year for the US and Russia respectively. It is created with the twinx function of the matplot library, which creates an independent set of "twin" axes that shares the same x-axis as the original axes, but has a separate y-axis and its own set of tick labels.

We clearly see an increasing number of launches per year which can be attributed to the space race between the USSR and the US around the 1960s. The blue line plot slows after 1969, the year of the lunar landing by the United States. Around 1980, Russia's yearly rocket launches overtook those of the US and stayed higher until around 2020. To see who will have the heads up in the coming months and years the csv file with upcoming launches can be used to plot similar charts.

4. Conclusion

In conclusion, the main objective of this program is to scrape data from an HTML webpage and manipulate it. To do so, the program imports necessary libraries, defines global variables and support functions, and uses scraping functions to extract the data from the website and save it in a CSV file. The program is structured into six main parts, including the import of libraries, definition of global variables, support functions for website scraping, actual scraping functions for past and future launches, and chart generation to visualize the extracted data. The support functions scrape_page and get_detailed_info are used to crawl the website and extract data on past and future launches, which is then saved in a CSV file and used to generate charts for visualization. Overall, this program demonstrates the usefulness of python code and its various features enabled through various libraries for efficiently scraping and manipulating data from an HTML webpage.

5. References

Hillier, W. (2021, August 13). *What is web scraping? [a complete step-by-step guide]*. CareerFoundry. Retrieved December 16, 2022, from https://careerfoundry.com/en/blog/data-analytics/web-scraping-guide/

Hobson, S. (2022, October 30). *Document and website structure - learn web development: MDN*. Learn web development | MDN. Retrieved December 17, 2022, from https://developer.mozilla.org/en-US/docs/Learn/HTML/Introduction_to_HTML/Document_and_website_structure

Lekh, N. (2022, November 15). *Pros and cons of web scraping*. Apify. Retrieved December 16, 2022, from https://blog.apify.com/pros-and-cons-of-web-scraping/

Minter, A. (2022, July 25). *Musk, Mars and the spacex competitors: The Space Race Is Going Private*. Bloomberg.com. Retrieved December 16, 2022, from https://www.bloomberg.com/opinion/articles/2022-07-25/musk-mars-and-the-spacex-competitors-the-space-race-is-going-private

Mullen, M. (2010, February 22). *The space race*. History.com. Retrieved December 16, 2022, from https://www.history.com/topics/cold-war/space-race

Dutta, R. (2021, September 5). *Applied-data-science-capstone/web scraping Falcon 9 and Falcon Heavy launches records from wikipedia.ipynb at master · Ranajoy-Dutta/applied-data-science-capstone*. GitHub. Retrieved December 22, 2022, from https://github.com/ranajoy-dutta/Applied-Data-Science-Capstone/blob/master/Web%20scraping%20Falcon%209%20and%20Falcon%20Heavy%20Launches%20Records%20from%20Wikipedia.ipynb