project1

October 2, 2022

Part 1

[1]:
```
import pandas as pd
import numpy as np
from bs4 import BeautifulSoup
import requests
```

[2]:
```
# Step 1
# Get raw data from website
r = requests.get("https://cmsc320.github.io/files/top-50-solar-flares.html")
# Parse data to html
soup = BeautifulSoup(r.text, 'html.parser')
# Prettify the table to find the table
soup.prettify()
# print(soup.prettify)
# Find the desired table froup the html
table = soup.find("table", {"class":"table table-striped table-responsive-md"})
# Convert table html to dataframe
df = pd.read_html(str(table))[0]
# Rename dataframe columns
df = df.rename(columns={"Unnamed: 0":"Rank", "Unnamed: 1":"x_classification",␣
 ↪"Unnamed: 2":"Date", "Start":"Start_Time", "Maximum":"Maximum_Time", "End":
 ↪"End_Time", "Unnamed: 7":"Movie"})

print(df.head())
```

```
   Rank x_classification        Date  Region Start_Time Maximum_Time End_Time  \
0     1             X28+  2003/11/04     486      19:29        19:53    20:06
1     2             X20+  2001/04/02    9393      21:32        21:51    22:03
2     3           X17.2+  2003/10/28     486      09:51        11:10    11:24
3     4             X17+  2005/09/07     808      17:17        17:40    18:03
4     5            X14.4  2001/04/15    9415      13:19        13:50    13:55

               Movie
0  MovieView archive
1  MovieView archive
2  MovieView archive
3  MovieView archive
```

```
4  MovieView archive
```

[3]:

```
# Step 2
# Drop the final column
    # Note that rerunning this cell will error, trying to delete nonexistent row
df = df.drop("Movie", axis=1)
# Combine dates and times in time columns,
for row in range(len(df)):
    df.at[row, "Start_Time"] = pd.to_datetime(df.at[row, "Start_Time"] + " " +␣
  ↪df.at[row, "Date"])
    df.at[row, "Maximum_Time"] = pd.to_datetime(df.at[row, "Maximum_Time"] + "␣
  ↪" + df.at[row, "Date"])
    df.at[row, "End_Time"] = pd.to_datetime(df.at[row, "End_Time"] + " " + df.
  ↪at[row, "Date"])
# then drop date column,
df = df.drop("Date", axis=1)
# rename columns,
df = df.rename(columns={"Start_Time":"Start_Datetime", "Maximum_Time":
  ↪"Max_Datetime", "End_Time":"End_Datetime"})
# and move Region to the end
df = df[["Rank", "x_classification", "Start_Datetime", "Max_Datetime",␣
  ↪"End_Datetime", "Region"]]
# Replace missing Region values with NAN
df = df.replace("-", np.nan)

print(df.head())
```

```
   Rank x_classification       Start_Datetime         Max_Datetime  \
0     1             X28+  2003-11-04 19:29:00  2003-11-04 19:53:00
1     2             X20+  2001-04-02 21:32:00  2001-04-02 21:51:00
2     3           X17.2+  2003-10-28 09:51:00  2003-10-28 11:10:00
3     4             X17+  2005-09-07 17:17:00  2005-09-07 17:40:00
4     5            X14.4  2001-04-15 13:19:00  2001-04-15 13:50:00

          End_Datetime  Region
0  2003-11-04 20:06:00     486
1  2001-04-02 22:03:00    9393
2  2003-10-28 11:24:00     486
3  2005-09-07 18:03:00     808
4  2001-04-15 13:55:00    9415
```

[4]:

```
# Step 3
# Get raw data from website
r2 = requests.get("https://cmsc320.github.io/files/waves_type2.html")
# Parse data to html
soup2 = BeautifulSoup(r2.text, 'html.parser')
# Prettify html and notice it's stored as strings separated by newlines
```

```
text2 = soup2.prettify()
# So, split lines for each newline
lst = text2.split("\n")
# Then, delete every line that doesn't start with a number,
    # as only table rows start with a number
lst = [x for x in lst if len(x) > 0 and x[0].isnumeric()]

# Long process to convert each row to data entries
# Create list to append values to
biglst = []
# Loop through each line
for line in lst:
    # Split each line across spaces
    temp = line.split(" ")
    # Initialize loop variable; index for ^^ split
    y=0
    # To summarize,
    # in each row, split into strings whenever a space appears,
    # and then loop across those strings
    while y < len(temp):
        # If current string is empty (when table has two consecutive spaces),␣
 ↪delete
        if len(temp[y]) == 0:
            del temp[y]
            y = y - 1
        # Many of our row elements are of the form:
        # <a href="some url">VALUE</a>
        # We want to extract VALUE
        # However, when splitting with spaces, "<a href" -> "<a", "href"
        # So, delete the first of those,
        elif temp[y] == "<a":
            del temp[y]
            y = y - 1
        # before parsing VALUE out of "href="some url">VALUE</a>
        elif "href" in temp[y]:
            tempstr = temp[y]
            templst = tempstr.split(">")
            temp[y] = templst[1][:-3]
        y = y + 1
    # Delete empty columns and add to list
    del temp[14:]
    biglst.append(temp)

# Finally, make a dataframe with appropriate column labels
df2 = pd.DataFrame(biglst, columns=["Start_Date", "Start_Time", "End_Date",
                                   "End_Time", "Start_Frequency",␣
 ↪"End_Frequency",
                                   "Flare_Location", "Flare_Region",␣
↪"Flare_Classification",
                                   "CME_Date", "CME_Time", "CME_Angle",␣
↪"CME_Width", "CME_Speed"])

print(df2.head())
```

```
   Start_Date Start_Time End_Date End_Time Start_Frequency End_Frequency  \
0  1997/04/01      14:00    04/01    14:15            8000          4000
1  1997/04/07      14:30    04/07    17:30           11000          1000
2  1997/05/12      05:15    05/14    16:00           12000            80
3  1997/05/21      20:20    05/21    22:00            5000           500
4  1997/09/23      21:53    09/23    22:16            6000          2000

  Flare_Location Flare_Region Flare_Classification CME_Date CME_Time  \
0         S25E16         8026                 M1.3    04/01    15:18
1         S28E19         8027                 C6.8    04/07    14:27
2         N21W08         8038                 C1.3    05/12    05:30
3         N05W12         8040                 M1.3    05/21    21:00
4         S29E25         8088                 C1.4    09/23    22:02

  CME_Angle CME_Width CME_Speed
0        74        79       312
1      Halo       360       878
2      Halo       360       464
3       263       165       296
4       133       155       712
```

[5]:

```
# Step 4

# Note that for efficiency, all of these for loops could be condensed into one.
# However, to preserve the structure of the assignment, they are separate.

# Find all missing entry types, then recode as NaN
# Loop through each row
for row in range(len(df2)):
    # Then loop through list of columns
    for col in df2.columns:
        # All missing entries start with a hyphen
        if df2.at[row, col][0] == "-":
            df2.at[row, col] = np.nan

# Create column of booleans if cme_angle is halo or not; change halo values in␣
↪cme_angle to NaN
# Initialize list of booleans for halo or not
halo = []
# For each row, if CME angle is halo, append true and change to nan, else false
```

```
for row in range(len(df2)):
    if df2.at[row, "CME_Angle"] == "Halo":
        halo.append(True)
        df2.at[row, "CME_Angle"] = np.nan
    else:
        halo.append(False)
df2["Is_Halo"] = halo

# cme_width indicates if value is lower bound (int or >int);
    # create column of bools for lower bound or not and make cme_width int only
# The same concept and code as for halo

lower = []
for row in range(len(df2)):
    # Assignment to avoid duplicate .at calls
    curr = df2.at[row, "CME_Width"]
    # Needed case since NaN, as missing entry replacement, is type float, not string
    if type(curr) == float:
        lower.append(False)
    elif curr[0] == ">":
        lower.append(True)
        # Slice used to update value as all but first character, which is >
        df2.at[row, "CME_Width"] = curr[1:]
    else:
        lower.append(False)
df2["Width_Lower_Bound"] = lower

# Combine date/time columns for start, end, and cme
    # Prob gonna need to take year off of start date for end date and cme date; visit each row once
# Combine dates and times in time columns:
for row in range(len(df2)):
    # Repeating block to convert 24:00 to 00:00 for to_datetime to parse
    time = df2.at[row, "Start_Time"]
    # If time is missing entry (nan is a float), keep nan
    if type(time) == float:
        df2.at[row, "Start_Time"] = np.nan
    else:
        if time[:2] == "24":
            time = "00" + time[2:]
        df2.at[row, "Start_Time"] = pd.to_datetime(time + " " + df2.at[row, "Start_Date"])

    # Save year since it only appears in start time
    year = df2.at[row, "Start_Date"][:4]
    time = df2.at[row, "CME_Time"]
    if type(time) == float:
        df2.at[row, "Start_Time"] = np.nan
    else:
        if time[:2] == "24":
            time = "00" + time[2:]
        df2.at[row, "CME_Time"] = pd.to_datetime(time + " " + year + "/" + df2.
 ↪at[row, "CME_Date"])

    time = df2.at[row, "End_Time"]
    if type(time) == float:
        df2.at[row, "Start_Time"] = np.nan
    else:
        if time[:2] == "24":
            time = "00" + time[2:]
        df2.at[row, "End_Time"] = pd.to_datetime(time + " " + year + "/" + df2.
 ↪at[row, "End_Date"])

# then drop date columns,
df2 = df2.drop("Start_Date", axis=1).drop("End_Date", axis=1).drop("CME_Date",␣
 ↪axis=1)
# and rename columns
df2 = df2.rename(columns={"Start_Time":"Start_Datetime",
                        "CME_Time":"CME_Datetime", "End_Time":"End_Datetime"})

print(df2.head())
```

```
        Start_Datetime         End_Datetime Start_Frequency End_Frequency  \
0  1997-04-01 14:00:00  1997-04-01 14:15:00             8000          4000
1  1997-04-07 14:30:00  1997-04-07 17:30:00            11000          1000
2  1997-05-12 05:15:00  1997-05-14 16:00:00            12000            80
3  1997-05-21 20:20:00  1997-05-21 22:00:00             5000           500
4  1997-09-23 21:53:00  1997-09-23 22:16:00             6000          2000

  Flare_Location Flare_Region Flare_Classification         CME_Datetime  \
0         S25E16         8026                 M1.3  1997-04-01 15:18:00
1         S28E19         8027                 C6.8  1997-04-07 14:27:00
2         N21W08         8038                 C1.3  1997-05-12 05:30:00
3         N05W12         8040                 M1.3  1997-05-21 21:00:00
4         S29E25         8088                 C1.4  1997-09-23 22:02:00

  CME_Angle CME_Width CME_Speed  Is_Halo  Width_Lower_Bound
0        74        79       312    False              False
1       NaN       360       878     True              False
2       NaN       360       464     True              False
3       263       165       296    False              False
4       133       155       712    False              False
```

Part 2

[6]:

```python
# Question 1
# Is it possible to replicate the top 50 solar flares from NASA data? (Rank based on x classification)
# Make a list of all classifications
classes = df2.loc[:, "Flare_Classification"]
# Initialize and populate a list of just the x classifications (highest rank)
onlyx = []
for c in range(len(classes)):
    if type(classes[c]) == str and classes[c][0] == "X":
        onlyx.append((c, classes[c]))
# Sort the list based on the floats after the X
sort = sorted(onlyx, key = lambda x : float(x[1][1:]), reverse=True)
# Take the top 50 of those floats
top50 = sort[:50]
# And find the associated indices
top50i = [i for (i, c) in top50]
# Then, make a dataframe with the same columns as the NASA dataframe
df50 = pd.DataFrame(columns=df2.columns)
# And add the rows of the top 50 flares to that dataframe
j = 0
for i in top50i:
    df50.loc[j] = df2.iloc[i]
    j = j + 1

# print(df["Start_Datetime"])
# print(df50["Start_Datetime"])

# By simply comparing the lists of dates from the original 50 and the NASA 50,
# we see within the first ten entries multiple discrepencies.
# For one, NASA is missing a few, and has some NaN that the original doesn't
```

[7]:

```python
# Question 2
# For each top 50 flare, find best matching row in NASA
# Add column to NASA indicating rank, if it appears in that dataset

# We use dates as our first point of comparison
# Regardless of differences between measurements or classifications,
# dates are absolute enough that the only errors that may appear are near ends of days,
# which is a small margin, whereas classification, for example, could be off by
# a few tenths in a harder way to predict.
# Then, if a date has two or no flares in the NASA dataset, we pick the higher classification
# or skip that rank, respectively
# We create a list of all the start dates in the NASA dataset
start_dates = [date.date() if type(date) is not float else np.nan for date in␣
 ↪df2["Start_Datetime"]]
matchlist = []
# Then, we loop through the top 50 dataset
for row in range(len(df)):
    date = df.loc[row, "Start_Datetime"].date()
    # Find all matching dates
    matches = np.where(np.array(start_dates) == date)[0]
    # And select our best match
    # Arbitrary to select if only one match
    if len(matches) == 1:
        matchlist.append(matches[0])
    # If two matches, pick the larger classification, preferring x as a prefix
    elif len(matches) > 1:
        maxclass = 0
        maxidx = 0
        for index in matches:
            cla = df2.loc[index, "Flare_Classification"]
            if type(cla) is not float:
                if cla[0] == "X":
                    old = maxclass
                    maxclass = max(maxclass, float(cla[1:]))
                    if maxclass is not old:
                        maxidx = index
        matchlist.append(maxidx)
    # If no matches, add a placeholder to the list to skip this rank
    else:
        matchlist.append(np.nan)
# Create a new column with -1 as unranked for type consistency
df2["Rank"] = [-1] * len(df2)
rank = 1
# Loop through matches, assigning appropriate rank to where matches appear
for i in matchlist:
    if i is np.nan:
        rank = rank + 1
    else:
        df2.at[i, "Rank"] = rank
        rank = rank + 1
# Tail is shown as result because it has ranked entries
print(df2.tail())
```

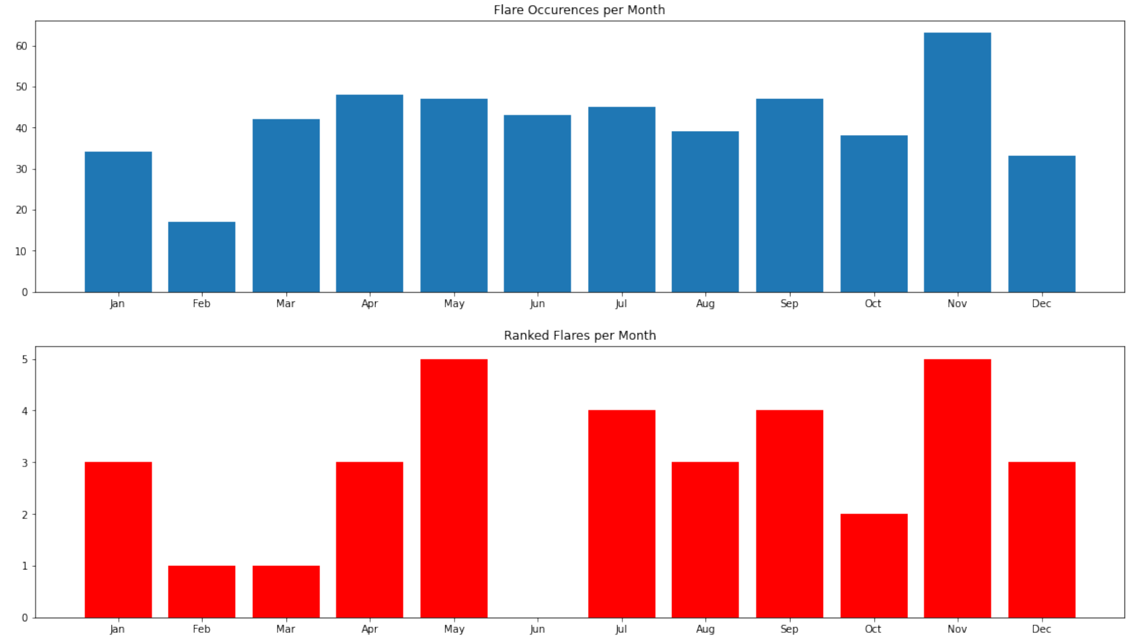
```
          Start_Datetime         End_Datetime Start_Frequency End_Frequency  \
513  2017-09-04 20:27:00  2017-09-05 04:54:00           14000           210
514  2017-09-06 12:05:00  2017-09-07 08:00:00           16000            70
515  2017-09-10 16:02:00  2017-09-11 06:50:00           16000           150
516  2017-09-12 07:38:00  2017-09-12 07:43:00           16000         13000
```

```
517  2017-09-17 11:45:00  2017-09-17 12:35:00           16000           900

    Flare_Location Flare_Region Flare_Classification         CME_Datetime  \
513         S10W12        12673                 M5.5  2017-09-04 20:12:00
514         S08W33        12673                 X9.3  2017-09-06 12:24:00
515         S09W92          NaN                 X8.3  2017-09-10 16:00:00
516         N08E48        12680                 C3.0  2017-09-12 08:03:00
517        S08E170          NaN                  NaN  2017-09-17 12:00:00

    CME_Angle CME_Width CME_Speed  Is_Halo  Width_Lower_Bound  Rank
513       NaN       360      1418     True              False    -1
514       NaN       360      1571     True              False     8
515       NaN       360      3163     True              False    11
516       124        96       252    False              False    -1
517       NaN       360      1385     True              False    -1
```

[8]:

```python
# Question 3
# Prepare a plot showing the top 50 in context with all NASA data

from matplotlib import pyplot as plt

# One interesting question about a lot of enviromental behaviour on Earth is when it happens
# With seasons and human-induced climate change, we care about when certain things happen
# While these factors don't necessarily exist on the sun, the question remains:
    # Does time have a noticeable impact on strong flare occurences?

months = [0] * 12
ranked = [0] * 12
for row in range(len(df2)):
    month = df2.at[row, "Start_Datetime"]
    if month is not np.nan:
        month = month.date().month
        months[month - 1] += 1
        if df2.at[row, "Rank"] > 0:
            ranked[month - 1] += 1

fig, ax = plt.subplots(2, figsize=(19.2,10.8))
ax[0].bar(height=months, x=["Jan","Feb","Mar","Apr","May","Jun","Jul","Aug","Sep","Oct","Nov","Dec"])
ax[0].set_title("Flare Occurences per Month")
ax[1].bar(height=ranked, x=["Jan","Feb","Mar","Apr","May","Jun","Jul","Aug","Sep","Oct","Nov","Dec"],
          color="red")
ax[1].set_title("Ranked Flares per Month")
fig.show()
```

```
# Here, we plot the number of flare occurences in each month
# Below that, we show the number of top 50 flares per month

# There are a few observations we can draw from these graphs.
# Firstly, February and November are the lowest and highest in both graphs, respectively.
# We might expect a slight dip in Febryary simply due to its lower number of days,
# but not to the extent that appears here.
# November sticks out significantly. If I were investigating flares, I would
# try to plan experiments around November for more results, or if flares effect my tests,
# I would work in the lower occurence months such as December-January.
```



[ ]: