STAT 243 Problem Set 2

Treves Li

2024-09-19

Collaboration Statement

I did not collaborate with anyone.

Question 1

> A friend of mine is planning to get married in Death Valley National Park in March (this problem is based on real events…). She wants to hold it as late in March as possible but without having a high chance of a very hot day. This problem will automate the task of generating information about what day of March to hold the wedding using data from the Global Historical Climatology Network. All of your operations should be done using the bash shell except part (c). Also, ALL of your work should be done using shell commands that you save in your solution file. So you can't say "I downloaded the data from such-and-such website" or "I unzipped the file"; you need to give us the bash code that we could run to repeat what you did. This is partly for practice in writing shell code and partly to enforce the idea that your work should be reproducible and documented.

Question 1a

> Download yearly climate data for a set of years of interest into a temporary directory. Do not download all the years and feel free to focus on a small number of years to reduce the amount of data you need to download. Note that data for Death Valley is only present in the last few decades. As you are processing the files, report the number of observations in each year by printing the information to the screen (i.e., `stdout`), including if there are no observations for that year.

I used `wget` and a `for` loop to download data over a range of years. This required testing the `for` loop first on 2 years of data, before running the loop on a bigger range of 10 years. I did this since the datasets are quite big and would take a while (at least on my computer) to `wget`!

```bash
# First we make a temporary directory
mkdir -p "tmp"

for year in {2014..2024};
do
    url="https://www1.ncdc.noaa.gov/pub/data/ghcn/daily/by_year/${year}.csv.gz"

    # `wget` climate data recent years to 2024
    # Save files into temp directory
    # Use `-q` flag to silence output; `-O` flag to specify output filename
    wget -qO "tmp/${year}.csv.gz" $url

    # Unzip files (`gunzip` removes the `.gz`s automatically)
    # `-f` to force overwrite if .csv already exists
    gunzip -f "tmp/${year}.csv.gz"

    # Report if there are no observations for a year
    # `-s` checks if the file exists, and is not empty
    if [ ! -s "tmp/${year}.csv" ]; then
        echo "No records found for ${year}."
    else
        # Report the number of observations in each year
        echo "${year} had $(wc -l < tmp/${year}.csv) observations."
    fi
done
```

```
2014 had 36201914 observations.
2015 had 36760875 observations.
2016 had 37105720 observations.
```

```
2017 had 36928217 observations.
2018 had 36832837 observations.
2019 had 36461979 observations.
2020 had 36866372 observations.
2021 had 37809415 observations.
2022 had 37811003 observations.
2023 had 37729342 observations.
2024 had 23820517 observations.
```

Question 1b

> Subset to the station corresponding to Death Valley, to the TMAX (maximum daily temperature) variable, and to March, and put all the data into a single file. In subsetting to Death Valley, get the information programmatically from the `ghcnd-stations.txt` file one level up in the website. Do NOT type in the station ID code when you retrieve the Death Valley data from the yearly files.

First we have to find Death Valley's station ID by accessing the `.txt` file using `wget`.

```
# Fetch `ghcnd-stations.txt`
stations="ghcnd-stations.txt"
wget -qO "tmp/$stations" "https://www1.ncdc.noaa.gov/pub/data/ghcn/daily/$stations"

# Get station ID using `grep` and `cut`, and save the output as variable `dvnp_id`
# `-i` flag indicates case-insensitive match
# Had to open the .txt to find that the .txt was space delimited
dvnp_id=$(grep -i "death valley" tmp/$stations | cut -d" " -f1)
echo "The station ID code for Death Valley is $dvnp_id."

# Delete the stations .txt file
rm tmp/$stations
```

```
The station ID code for Death Valley is USC00042319.
```

Then for each year, find the value(s) corresponding to DVNP & TMAX & March. I use `awk` instead of `grep` since it's more adept at multiple column filtering. Again, I first tested the `for` loop on a handful of years, before extending to the larger range of years.

```
# First create a textfile in which to save the data
touch dv_tmax_march.txt

# `awk` filtering on three substrings, three columns
# Use regex on col2 to extract dates for March
# Print commas as delimiters
# Append filtered data to textfile
for year in {2014..2024};
do
    awk -F',' -v col1=$dvnp_id -v col2=".{4}03.{2}" -v col3="TMAX" \
        '$1 ~ col1 && $2 ~ col2 && $3 ~ col3 \
        {print $1 "," $2 "," $3 "," $4}' tmp/${year}.csv \
        >> dv_tmax_march.txt
done
```

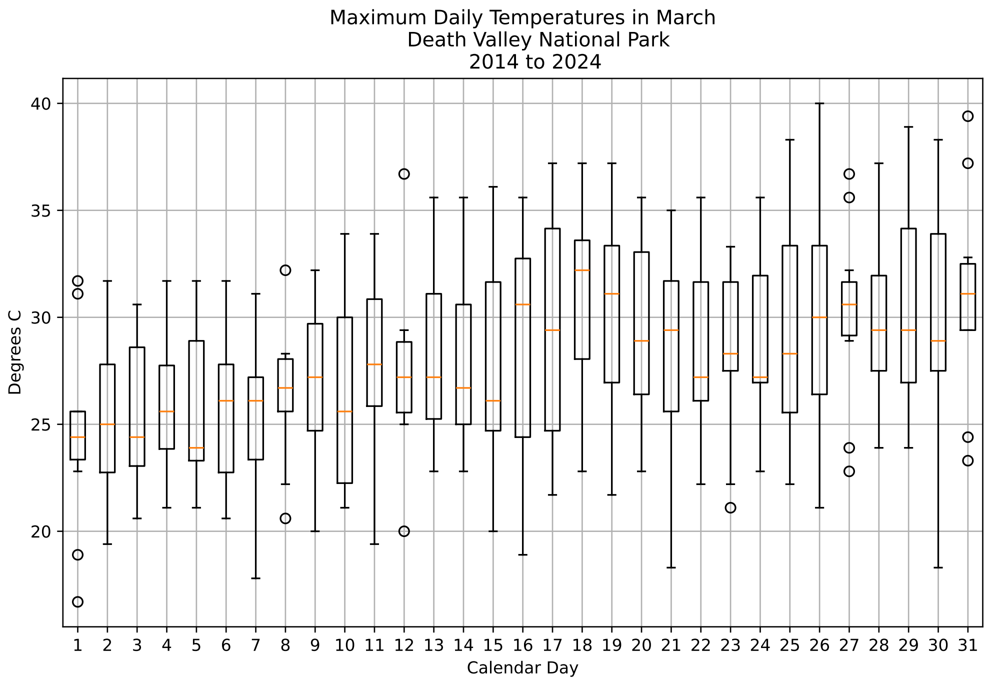
Question 1c

> Create a Python chunk (or R would be fine too) that takes as input your single file from (b) and makes a single plot showing side-by-side boxplots containing the maximum daily temperatures on each calendar day in March. (If you somehow really have trouble mixing Python and bash chunks, it's ok to insert this figure manually, after running the Python code separately. In this case you could use the `jupyter` engine provided that a bash kernel is available for Jupyter.)

If first need to read in my `.txt` file as a DataFrame (or similar Python object). Then I need to group the data in the `.txt` file by day of the calendar month. Lastly, I need to look at the documentation for how to make boxplots using `matplotlib`.

It's important to note that the raw `TMAX` values are in tenths of degrees Celsius. It took me too long to figure this out, but it is documented in NOAA's readme for this dataset. So I had to divide all the raw values by 10.

```python
import pandas as pd
import matplotlib.pyplot as plt

# Read in dv_tmax_march.txt file as df
df = pd.read_csv("dv_tmax_march.txt", header=None)

# Rename headers to make things easier to work with
df = df.rename(
    columns={
        0: "station_id",
        1: "date",
        3: "tmax",
    }
)

# Change date column to string, for better string matching
df["date"] = df["date"].astype(str)

# Initialise a list to store tmax values grouped by day
tmax_march_days = []

# Initialise arrays for each day of March (31 days in March )
for day in range(1, 32):
    # Change `day` format so it's a 2 digit string
    day = f"{day:02d}"
    # Do filtering
    tmax_by_day = df.loc[df["date"].str[-2:] == day, "tmax"].values

    # Divide values by 10 to get degrees C
    tmax_by_day = tmax_by_day / 10

    # Append information to tmax_march_days
    tmax_march_days.append(tmax_by_day)

# Convert list to dataframe
df_tmax = pd.DataFrame(tmax_march_days)

# Rename columns to years in df_tmax
start_year = 2014
end_year = 2024
years = list(range(start_year, end_year + 1))
df_tmax.columns = years

# Re-index rows in df_tmax to correspond to days of months (i.e., start at 1)
df_tmax.index = df_tmax.index + 1

# Print output to check if it's what we want
# print(df_tmax)

# Use matplotlib to make a box chart
plt.figure(figsize=(10, 6))
plt.boxplot(df_tmax.values.T);
plt.title(f"Maximum Daily Temperatures in March\n \
    Death Valley National Park\n\
    {start_year} to {end_year}")
plt.xlabel("Calendar Day")
plt.ylabel("Degrees C")
plt.grid(True)
plt.show()
```



Lastly, I linted the code above to make it adhere to standard (I worked on the Python code in a separate file called `q1c_test.py`):

```
ruff check q1c_test.py
ruff format q1c_test.py
```

```
All checks passed!
1 file left unchanged
```

Question 1d

Now generalize your code from parts (a) and (b). Write a shell function that takes as arguments a string for identifying the location, the weather variable of interest, and the time period (i.e., the years of interest and the month of interest), and returns the results. Your function should detect if the user provides the wrong number of arguments or a string that doesn't allow one to identify a single weather station and return a useful error message. It should also give useful help information if the user invokes the function as: `get_weather -h`. Finally the function should remove the raw downloaded data files (or you should download into your operating system's temporary file location).

Hint: to check for equality in an if statement, you generally need syntax like:

```
if [ "${var}" == "7" ]
```

When the `-h` flag is entered, I used ChatGPT to give ideas on how to incorporate it into my code. I don't know if using multiple `echo` statements is the most efficient, but it allowed me to format the output text the way I wanted.

To counts the number of `args` passed to the function in bash, I relied on this Stack Overflow thread.

Note that in bash, `return 1` in a function indicates exit with failure. (On the other hand, `return 0` indicates to exit the function with success.) I used this any time the number of `args` inputted was wrong, or if no match was foudn for a location.

I wrote the code so that if there are multiple matches for a place name, only the the first weather station that is matched is retrieved. Otherwise, if I called something like "Fairbanks", it would return a lot of weather station IDs and the function would take too long to run.

```bash
function get_weather() {

    ### INPUTS ###
    # $1: station ID (str)
    # $2: weather variable of interest (str)
    # $3: start year of interest (int)
    # $4: end year of interest (int)
    # $5: month of interest (int)

    ### OUTPUTS ###
    # Death Valley data to `stdout`

    ### HELP INFORMATION ###
    # Check if help flag is invoked
    if [ "$1" == "-h" ]; then
        echo "Usage: get_weather [station_id] [weather_var] [start_year] [end_year] [month]"
        echo
        echo
        echo "Retrieves certain GHCN weather data for a given year range and month."
        echo
        echo "Arguments:"
        echo "  station_id  The weather station ID (e.g., "USC00042319")"
        echo "  weather_var See below for main variables"
        echo "                  PRCP = Precipitation (tenths of mm)"
        echo "                  SNOW = Snowfall (mm)"
        echo "                  SNWD = Snow depth (mm)"
        echo "                  TMAX = Maximum temperature (tenths of degrees C)"
        echo "                  TMIN = Minimum temperature (tenths of degrees C)"
        echo "  start_year  The starting year of the range (e.g., 2019)"
        echo "  end_year    The ending year of the range (e.g., 2024)"
        echo "  month       The month in "dd" format"
        echo
        echo "Example:"
        echo "  get_weather \"death valley\" \"TMAX\" 2023 2024 8"

        # Exit function with success
        return 0
    fi

    ### ERROR CHECKING ###
    # Check if user provides wrong number of args
    if  [ "$#" -ne 5 ]; then
        echo "Error: Function requires 5 arguments."
        return 1
    fi

    # Fetch `ghcnd-stations.txt`
    stations="ghcnd-stations.txt"
    wget -qO "$stations" "https://www1.ncdc.noaa.gov/pub/data/ghcn/daily/$stations"

    # Check if input string $1 doesn't match a station in `ghcnd-stations.txt`
    # `-i` flag in grep indicates case-insensitive match
    # `-q` flag to silence grep output
    if ! cut -f1 $stations | grep -iq "$1"; then
        echo "Error: No weather station found matching \"$1\"."
        return 1
```

```bash
else
    # Get station_id using `grep` and `cut`
    # `-i` flag indicates case-insensitive match
    # It will only get the first station ID match!
    station_id=$(grep -i "$1" $stations | cut -d' ' -f1 | head -n 1)
    echo "Found weather station(s) matching \"$1\" ("$station_id")."
fi

### GET DATA ###
# Make a temporary directory in which to save files
tmp_dir="tmp_func"
mkdir -p "$tmp_dir"

# Process month as two digits
if [ $5 -lt 10 ]; then
    month="0$5"
else
    month="$5"
fi

# From part (a): get data from NOAA GHCN
# From part (b): output the data
for year in $(seq "$3" "$4");
do
    url="https://www1.ncdc.noaa.gov/pub/data/ghcn/daily/by_year/${year}.csv.gz"

    # `wget` climate data for user specified number of years
    # Save files into temp
    # Use `-q` flag to silence output; `-O` flag to specify output filename
    wget -qO "${tmp_dir}/${year}.csv.gz" "$url"

    # Unzip files (`gunzip` removes the `.gz`s automatically)
    # `-f` to force overwrite if .csv already exists
    gunzip -f "${tmp_dir}/${year}.csv.gz"

    awk -F',' -v col1="$station_id" -v col2="$month" -v col3=$2 \
        '$1 == col1 && substr($2, 5, 2) == col2 && $3 == toupper(col3) \
        {print $1 "," $2 "," $3 "," $4}' \
        "${tmp_dir}/${year}.csv"

    if ! grep $station_id "${tmp_dir}/${year}.csv"; then
        echo "No records found for \"$1\" in ${year}."
        fi
    done

    # Remove raw downloaded data files (even if error above)
    rm -r "$tmp_dir" "$stations"
}

# Try a few examples:

echo "The following line should throw an error:"
get_weather "death valley" "TMAX" 2023 3
echo

echo "The following line should throw an error:"
get_weather "mordor" "TMAX" 2023 2024 3
echo

echo "The following line should give help info:"
get_weather -h
echo

echo "The following line should work (but no records found):"
get_weather "death valley" "TMAX" 1770 1771 2
echo

echo "The following line should work (but no records found):"
get_weather "north pole" "TMIN" 1994 1995 12
echo

echo "The following line should work (output truncated to 10 lines):"
get_weather "death valley" "TMAX" 2023 2024 8 | head -n 10
echo

echo "The following line should work (output truncated to 10 lines):"
get_weather "KOTZEBUE" "tmin" 1994 1995 12 | head -n 10
echo
```

```
The following line should throw an error:
Error: Function requires 5 arguments.

The following line should throw an error:
Error: No weather station found matching "mordor".
```

```
The following line should give help info:
Usage: get_weather [station_id] [weather_var] [start_year] [end_year] [month]


Retrieves certain GHCN weather data for a given year range and month.

Arguments:
  station_id  The weather station ID (e.g., USC00042319)
  weather_var See below for main variables
                PRCP = Precipitation (tenths of mm)
                SNOW = Snowfall (mm)
                SNWD = Snow depth (mm)
                TMAX = Maximum temperature (tenths of degrees C)
                TMIN = Minimum temperature (tenths of degrees C)
  start_year  The starting year of the range (e.g., 2019)
  end_year    The ending year of the range (e.g., 2024)
  month       The month in dd format

Example:
  get_weather "death valley" "TMAX" 2023 2024 8

The following line should work (but no records found):
Found weather station(s) matching "death valley" (USC00042319).
No records found for "death valley" in 1770.
No records found for "death valley" in 1771.

The following line should work (but no records found):
Found weather station(s) matching "north pole" (ASN00004088).
No records found for "north pole" in 1994.
No records found for "north pole" in 1995.

The following line should work (output truncated to 10 lines):
Found weather station(s) matching "death valley" (USC00042319).
USC00042319,20230801,TMAX,456
USC00042319,20230802,TMAX,478
USC00042319,20230803,TMAX,456
USC00042319,20230804,TMAX,456
USC00042319,20230805,TMAX,489
USC00042319,20230806,TMAX,472
USC00042319,20230807,TMAX,472
USC00042319,20230808,TMAX,456
USC00042319,20230809,TMAX,450
```

```
The following line should work (output truncated to 10 lines):
Found weather station(s) matching "KOTZEBUE" (USC00505051).
USC00505051,19941201,TMIN,-206
USC00505051,19941202,TMIN,-189
USC00505051,19941203,TMIN,-294
USC00505051,19941204,TMIN,-361
USC00505051,19941205,TMIN,-389
USC00505051,19941206,TMIN,-411
USC00505051,19941207,TMIN,-400
USC00505051,19941208,TMIN,-400
USC00505051,19941209,TMIN,-300
```

Question 2

Add documentation, error-trapping and testing for your code from Problem 4, parts (b) and (c) of PS1. You may use a modified version of your PS1 solution, perhaps because you found errors in what you did or wanted to make changes based on Chris' solutions (to be distributed in class on Friday Sep. 13) or your discussions with other students. These topics will be covered in Lab 2 (Sep. 13) and are also discussed in Unit 4.

Question 2a

Add informative doc strings to your functions.

I copied my code/functions from Problem Set 1. I then used the docstring conventions proposed suggested by PEP to format my docstrings. I also used Chris's solutions from PS1 as a guideline for how much information to include in the docstring. I took his solution one step further by including sections on arguments and returns (even though for these simple functions, they're probably not needed).

```python
import subprocess
import requests
import pandas as pd
from bs4 import BeautifulSoup as bs

def get_scholar_citations(scholar_id):
    """Given a valid scholar ID as input,
    construct an http GET request
    (and submit that request) to get a
    scholar's first citations page from Google Scholar.

    Args:
      scholar_id (str): Google scholar ID

    Returns:
      citations_html (python object): The scholar's citations
    """

    # Define the url
    url = (
        "https://scholar.google.com/citations?user="
        + str(scholar_id)
        + "&hl=en&oi=ao"
    )

    # Define the saved html; give it a filename
    saved_html = "scholar.html"

    # Download the html via UNIX to avoid 429 errors
    subprocess.run(
        ["wget", "-qO", saved_html, url],  # save to scholar.html
        check=True,  # raise errors if any
    )

    # Open and read the html using BeautifulSoup
    with open(saved_html, "r", encoding="ISO-8859-1") as file:
        citations_html = bs(file, "html.parser")

    return citations_html

def get_scholar_info(filename):
    """Given the filename for a raw HTML,
    process the html to get scholar citation
    information stored as a dataframe.

    Args:
      html (object): html with scholar's citations

    Returns:
      df (Pandas dataframe): The scholar's information
    """

    # Open and read the html using BeautifulSoup
    with open(filename, "r", encoding="ISO-8859-1") as file:
        citations_html = bs(file, "html.parser")

    # Define df to store info with defined column labels
    df = pd.DataFrame(columns=["title", "authors", "journal", "year", "num_cite"])

    # Identify the scholar
    scholar = citations_html.find("div", attrs={"id": "gsc_prf_in"}).text

    # Create a list of citations
    citations = citations_html.find_all("tr", attrs={"class": "gsc_a_tr"})
    print(f"Found {len(citations)} entries for {scholar}.")

    # For each entry, extract relevant data
    # Note that authors and journal share html tags
    # so need to call them by their element
    for entry in citations:
        title = entry.find("a", class_="gsc_a_at").text
        authors = entry.find_all("div", class_="gs_gray")[0].text
        journal = entry.find_all("div", class_="gs_gray")[1].text
        year = entry.find("span", class_="gsc_a_h").text
        num_cite = entry.find("a", class_="gsc_a_ac").text

        # Append all data to a new row in the df
        new_row = pd.DataFrame(
            [
                {
                    "title": title,
                    "authors": authors,
                    "journal": journal,
                    "year": year,
                    "num_cite": num_cite,
                }
            ]
        )
        df = pd.concat([df, new_row], ignore_index=True)

    return df
```

Question 2b

> Add exceptions for handling run-time errors. You should try to catch the various incorrect inputs a user could provide and anything else that could go wrong (e.g., what happens if the server refuses the request or if one is not online?). In some cases you will want to raise an error, but in others you may want to catch an error with `try-except` and return `None`.

I was unfamiliar with `try-except`, so I relied on this Stack Overflow thread as well as the Python documentation/tutorial on errors and exceptions to understand how they work. I also used ChatGPT to assist with building the syntax for my two functions.

Exceptions to try for `get_scholar_citations()`:

- Server does not exist or is not online.
- Server rejects the request (e.g. 429 error).
- HTML is in a bad or un-parseable format.

Exceptions to try for `get_scholar_info()`:

- Filename doesn't exist or can't be opened.
- HTML can't be parsed for whatever reason (somehwat duplicated from `get_scholar_citations()`)

```python
def get_scholar_citations(scholar_id):
    """Given a valid scholar ID as input,
    construct an http GET request
    (and submit that request) to get a
    scholar's first citations page from Google Scholar.

    Args:
      scholar_id (str): Google scholar ID

    Returns:
      citations_html (python object): The scholar's citations, or None if error
    """
    url = (
        "https://scholar.google.com/citations?user="
        + str(scholar_id)
        + "&hl=en&oi=ao"
    )
    saved_html = "scholar.html"

    try:
        # Check if we can access URL
        response = requests.get(url)
        response.raise_for_status()
    except requests.RequestException as message:
        print(f"Error fetching URL: {message}")
        return None

    try:
        # Check for 429 errors (too many requests)
        subprocess.run(
            ["wget", "-qO", saved_html, url],
            check=True,
        )
    except subprocess.CalledProcessError as message:
        print(f"Error downloading file: {message}")
        return None

    try:
        # Check if there are issues with the html
        with open(saved_html, "r", encoding="ISO-8859-1") as file:
            citations_html = bs(file, "html.parser")
    except FileNotFoundError:
        print("Error: File not Found.")
        return None
    except Exception as message:
        print(f"Error with file: {message}")
        return None

    return citations_html



def get_scholar_info(filename):
    """Given the filename for a raw HTML,
    process the html to get scholar citation
    information stored as a dataframe.

    Args:
      filename (str): Path to the raw HTML file

    Returns:
      df (Pandas dataframe): The scholar's information, or None error
    """
    try:
        # Open and read the html using BeautifulSoup
        with open(filename, "r", encoding="ISO-8859-1") as file:
            citations_html = bs(file, "html.parser")
    except FileNotFoundError:
        print(f"Error: '{filename}' not found.")
        return None
    except IOError:
        print("Error: Can't read file.")
        return None
    except Exception as message:
        print(f"Error reading file: {message}")
        return None

    # Define df to store info with defined column labels
    df = pd.DataFrame(columns=["title", "authors", "journal", "year", "num_cite"])

    try:
        # Identify the scholar
        scholar = citations_html.find("div", attrs={"id": "gsc_prf_in"}).text

        # Create a list of citations
        citations = citations_html.find_all("tr", attrs={"class": "gsc_a_tr"})

        print(f"Found {len(citations)} entries for {scholar}.")

        # For each entry, extract relevant data
        for entry in citations:
            title = entry.find("a", class_="gsc_a_at").text
            authors = entry.find_all("div", class_="gs_gray")[0].text
            journal = entry.find_all("div", class_="gs_gray")[1].text
            year = entry.find("span", class_="gsc_a_h").text
            num_cite = entry.find("a", class_="gsc_a_ac").text

            # Append all data to a new row in the df
            new_row = pd.DataFrame([{
                        "title": title,
                        "authors": authors,
                        "journal": journal,
                        "year": year,
                        "num_cite": num_cite,
                    }])
            df = pd.concat([df, new_row], ignore_index=True)

    except Exception as message:
        print(f"Error processing html: {message}")
        return None

    return df
```

Question 2c

> Use the `pytest` package to set up a thoughtful set of unit tests of your functions.

I used the notes from Lab 2 of this class to develop my unit tests. Generally I want to build one suite of tests where I get the desired response, and another that tests where the functions should run with errors.

For the first suite, I set up two cases where the code should work:

- `get_scholar_citations` returns a `BeautifulSoup` object.
- `get_scholar_info` does not return `None` when it finds a valid HTML file.

For the second suite of tests, I created another three tests where the functions should **not** run:

- An invalid scholar ID is given.
- The HTML file does not exist or is missing.
- A bad HTML is passed as the file to be parsed.

```python
def test_get_scholar_citations():
    # given a scholar id, get citations as bs object
    scholar_id = "yxUduqMAAAAJ"
    result = get_scholar_citations(scholar_id)
    assert result is isinstance(result, bs)


def test_get_scholar_info():
    # checks that we don't return None
    # when a valid html exists
    filename = r"../ps1/scholar2.html"
    result = get_scholar_info(filename)
    assert result is not None


def test_get_scholar_citations_invalid_id():
    # what if the scholar ID is invalid?
    scholar_id = "some_random_string"
    result = get_scholar_citations(scholar_id)
    assert result is None  # scholar ID is invalid


def test_get_scholar_info_no_file():
    # what if no file found?
    filename = "random.html"
    result = get_scholar_info(filename)
    assert result is None


def test_get_scholar_info_bad_html():
    # Intentionally corrupted some html
    filename = "scholar2_bad.html"
    result = get_scholar_info(filename)
    assert result is None
```

I linted the Python code (I worked on it in a separate `.py` file):

```
ruff check q2d_functions.py
ruff format q2d_functions.py
```

```
All checks passed!
1 file left unchanged
```

And finally, I run the `pytest` unit tests:

```
import pytest

pytest.main(["q2d_functions.py"])
```

```
============================= test session starts ==============================
platform linux -- Python 3.10.12, pytest-8.3.3, pluggy-1.5.0
rootdir: /home/treves/treves/ps2
collected 5 items

q2d_functions.py .....                                                   [100%]

============================== 5 passed in 1.55s ===============================
<ExitCode.OK: 0>
```