STAT 243 Problem Set 2

Treves Li

2024-09-17

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.

This required testing the `for` loop on 2 years of data, before running the loop on a bigger range of 10 years, since the datasets are quite big and would take a while to `wget`!

```
# 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 for last 10 years (2014-2024)
    # Save files into temp
    # 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 the number of observations in each year
    echo "${year} had $(wc -l < tmp/${year}.csv) observations."
done
```

```
2014 had 36201912 observations.
2015 had 36760875 observations.
2016 had 37105720 observations.
2017 had 36928217 observations.
2018 had 36832837 observations.
2019 had 36461963 observations.
2020 had 36866367 observations.
2021 had 37809415 observations.
2022 had 37811001 observations.
2023 had 37728213 observations.
2024 had 23487750 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.

```
# Fetch `ghcnd-stations.txt`
stations="ghcnd-stations.txt"
wget -q0 "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 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

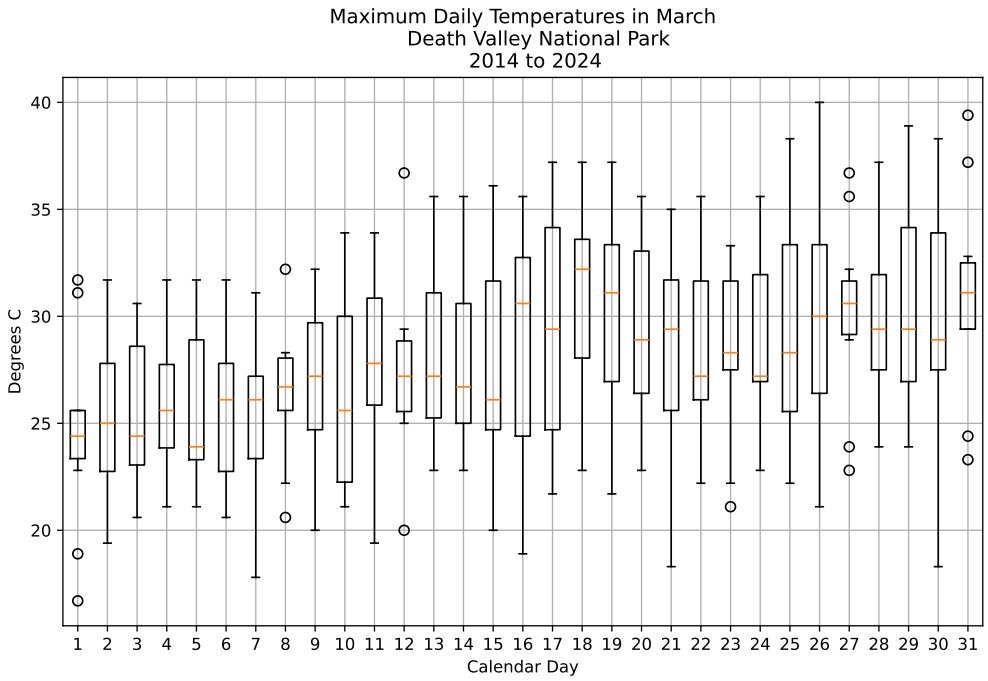
> Create a Python chunk (or R would be fine too) that takes as input your single file from (b) and makes a single plot showing side-by-side boxplots containing the maximum daily temperatures on each calendar day in March. (If you somehow really have trouble mixing Python and bash chunks, it's ok to insert 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.)

General steps would be 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 `TMAX` is in tenths of degrees C. Took me too long to figure this out, but it is documented in NOAA's readme for this dataset.

```
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
# 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 lint the code above to make it adhere to standard (I worked on the python code in a separate `.py` file, then pasted the linted code above):

```
ruff check
ruff format
```

```
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" ]
```

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.

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`.

Question 2c

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