

Big Data Exercises

Afraa Noureen

Bárbara Flores

In these exercises we will work on data from a series of global weather monitoring stations used to measure climate trends to examine long-term trends in temperature for your home locality. This data comes from the Global Historical Climatology Network, and is the actual raw data provided by NOAA. The only changes I have made to this data are a few small formatting changes to help meet the learning goals of this exercise.

To do these exercises, first please download the data for this exercise [from here](#). Note this is a big file (this is a big-data exercise, after all), so be patient.

```
In [ ]: import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

pd.set_option("mode.copy_on_write", True)
```

(1) The data we'll be working with can be found in the file `ghcnd_daily.tar.gz`. It includes daily weather data from thousands of weather stations around the world over many decades.

Begin by unzipping the file and checking its size -- it should come out to be *about* 4gb, but will expand to about 12 gb in RAM, which means there's just no way most students (who usually have, at most, 16gb of RAM) can import this dataset into pandas and manipulate it directly.

(Note: what we're doing can be applied to much bigger datasets, but they sometimes takes hours to work with, so we're working with data that's just a *little* big so we can get exercises done in reasonable time).

(2) Thankfully, we aren't going to be working with *all* the data today. Instead, everyone should pick three weather stations to examine during this analysis.

To pick your stations, we'll need to open the `ghcnd-stations.txt` file in the directory you've downloaded. It includes both station codes (which is what we'll find in the `ghcnd_daily.csv` data, as well as the name and location of each station).

When picking a weather station, make sure to pick one flagged as being in either GSN, HCN, or CRN (these designate more formalized stations that have been around a long time, ensuring you'll get a station with data that has been recorded over a longer period).

Note that Station IDs start with the two-letter code of the country in which they are located, and the "NAME" column often contains city names.

The `ghcnd-stations.txt` is a "fixed-width" dataset, meaning that instead of putting commas or tabs between observations, all columns have the same width (in terms of number of characters). So to import this data you'll have to (a) read the notes about the data in the project README.txt, and (b) read about how to read in fixed-width data in pandas. When entering column specifications, remember that normal people count from 1 and include end points, while Python counts from 0 and doesn't include end points (so if the readme says data is in columns 10-20, in Python that'd be 9 through 20).

The three weather stations that we pick are

1. AG000060390
2. AG000060611
3. AG000060680

```
In [ ]: column_names = [
        "ID",
        "LATITUDE",
        "LONGITUDE",
        "ELEVATION",
        "STATE",
        "NAME",
        "GSN FLAG",
        "HCN/CRN FLAG",
        "WMO ID",
    ]
path = "global_climate_data/ghcnd-stations.txt"

stations = pd.read_fwf(path, names=column_names)

stations[stations["ID"].isin(["AG000060390", "AG000060611", "AG000060680"])]
```

```
Out [ ]:
```

	ID	LATITUDE	LONGITUDE	ELEVATION	STATE	NAME	GSN FLAG
10	AG000060390	36.7167	3.2500	24.0	ALGER-DAR EL BEIDA	NaN	GSN
12	AG000060611	28.0500	9.6331	561.0	IN-AMENAS	NaN	GSN
13	AG000060680	22.8000	5.4331	1362.0	TAMANRASSET	NaN	GSN

(3) Now that we something about the observations we want to work with, we can now turn to our actual weather data.

Our daily weather can be found in `ghcnd_daily.csv`, which you get by unzipping `ghcnd_daily.tar.gz`. Note that the README.txt talks about this being a fixed-width file. Since you've already dealt with one fixed-width file, I've just converted this to a CSV, and dropped all the data that isn't "daily max temperatures".

Let's start with the fun part. **SAVE YOUR NOTEBOOK AND ANY OTHER OPEN FILES!** Then just try and import the data (`ghcnd_daily.csv`) while watching your Activity Monitor (Mac) or Resource Monitor (Windows) to see what happens.

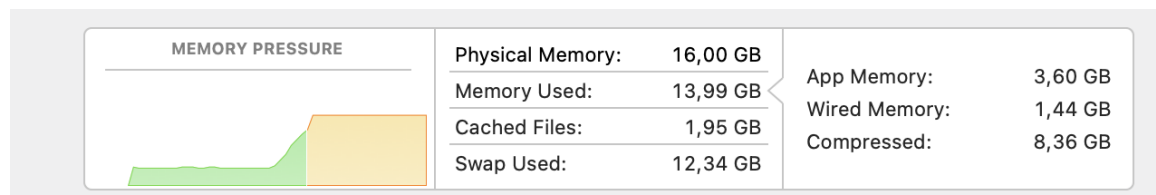
If you have 8GB of RAM, this should fail miserably.

If you have 16GB of RAM, you might just get away with this. But if it *does* load, try sorting the data by year and see how things go.

(If you have 32GB of RAM: you're actually probably fine with data this size. Sorry -- datasets big enough to cause big problems for people with 32GB take a long time to chunk on an 8GB computer, and these exercises have to be fast enough to finish in a class period! There are some exercises at the bottom with a REALLY big dataset you can work with.)

You may have to kill your kernel, kill VS Code, and start over when this explodes...

```
In [ ]: # path = "global_climate_data/ghcnd_daily.csv"
# df = pd.read_csv(path)
```



On importing the dataset, the memory usage increased significantly as shown above (taken from the Activity Monitor). It took about 5m 46.8s for the dataset to load successfully.

(4) Now that we know that we can't work with this directly, it's good with these big datasets to just import ~200 lines so you can get a feel for the data. So load *just 200 lines* of `ghcnd_daily.csv`.

```
In [ ]: path = "global_climate_data/ghcnd_daily.csv"
df_200 = pd.read_csv(path, nrows=200)
df_200
```

Out []:

	id	year	month	element	value1	mflag1	qflag1	sflag1	value2	mf
0	ACW00011604	1949	1	TMAX	289	NaN	NaN	X	289	
1	ACW00011604	1949	2	TMAX	267	NaN	NaN	X	278	
2	ACW00011604	1949	3	TMAX	272	NaN	NaN	X	289	
3	ACW00011604	1949	4	TMAX	278	NaN	NaN	X	283	
4	ACW00011604	1949	5	TMAX	283	NaN	NaN	X	283	
...
195	AE000041196	1981	9	TMAX	-9999	NaN	NaN	NaN	-9999	
196	AE000041196	1981	10	TMAX	-9999	NaN	NaN	NaN	350	
197	AE000041196	1981	11	TMAX	330	NaN	NaN	I	310	
198	AE000041196	1981	12	TMAX	270	NaN	NaN	I	290	
199	AE000041196	1982	1	TMAX	245	NaN	NaN	I	230	

200 rows × 128 columns

(5) Once you have a sense of the data, write code to chunk your data: i.e. code that reads in all blocks of the data that will fit in ram, keeps only the observations for the weather stations you've selected to focus on, and throws away everything else.

In addition to your own three weather stations, please also include station USC00050848 (a weather station from near my home!) so you can generate results that we can all compare (to check for accuracy).

Note you will probably have to play with your chunk sizes (probably while watching your RAM usage?). That's because small chunk sizes, while useful for debugging, are very slow.

Every time Python processes a chunk, there's a fixed processing cost, so in a dataset with, say, 10,000,000 rows, if you try to do chunks of 100 rows, that fixed processing cost has to be paid 100,000 times. Given that, the larger you can make your chunks the better, so long as your chunks don't use up all your RAM. Again, picking a chunk size then watching your RAM usage is a good way to see how close you are to the limits of your RAM.

```
In [ ]: list = []

for chunk in pd.read_csv(path, chunksize=100000, low_memory=False):
    selected_station_chunk = chunk[
        chunk["id"].isin(["AG000060390", "AG000060611", "AG000060680", "USC0
    ])
    list.append(selected_station_chunk)
```

```
selected_station_chunk_final = pd.concat(list, ignore_index=True)
```

(6) Now, for each weather station, figure out the *earliest* year with data. Keep `USC00050848` and the two of the three weather stations you picked with the best data (i.e., you should have 3 total, two you picked and `USC00050848`).

```
In [ ]: station_1 = "USC00050848"
station_2 = "AG000060390"
station_3 = "AG000060611"
station_4 = "AG000060680"

min_year_station_1 = selected_station_chunk_final[
    selected_station_chunk_final["id"] == station_1
]["year"].min()

min_year_station_2 = selected_station_chunk_final[
    selected_station_chunk_final["id"] == station_2
]["year"].min()

min_year_station_3 = selected_station_chunk_final[
    selected_station_chunk_final["id"] == station_3
]["year"].min()

min_year_station_4 = selected_station_chunk_final[
    selected_station_chunk_final["id"] == station_4
]["year"].min()

print(f"The earliest year for station {station_1} is {min_year_station_1}.")
print(f"The earliest year for station {station_2} is {min_year_station_2}.")
print(f"The earliest year for station {station_3} is {min_year_station_3}.")
print(f"The earliest year for station {station_4} is {min_year_station_4}.")

selected_station_chunk_3 = selected_station_chunk_final[
    selected_station_chunk_final["id"].isin(
        ["AG000060390", "AG000060680", "USC00050848"]
    )
]
```

The earliest year for station USC00050848 is 1893.
The earliest year for station AG000060390 is 1940.
The earliest year for station AG000060611 is 1958.
The earliest year for station AG000060680 is 1940.

The earliest years are for stations

- USC00050848 (Year 1893)
- AG000060390 (Year 1940)
- AG000060680 (Year 1940)

so we picked these three stations as it implies they have data recorded over a longer period of time.

(7) Now calculate the average max temp for each weather station / month in the data. Note that in a few weeks, we'll have the skills to do this by reshaping our data so each row is a single day, rather than a month. But for the moment, just sum the columns, watching out for weird values.


To sum across the value columns, we can combine:

```
weather_data.filter(like='value')  
(to just get the columns whose names start with "value") with  
.mean(axis='columns') (which averages across columns (along rows) rather than  
the usual averaging across rows (along columns)).
```

```
In [ ]: selected_station_chunk_3_without_99999 = selected_station_chunk_3.replace(-99999, np.nan)  
  
selected_station_chunk_3_without_99999["average_max_temp"] = (  
    selected_station_chunk_3_without_99999.filter(like="value").mean(axis="columns")  
    ) / 10  
  
# average_max_temp_monthly  
selected_station_chunk_3_without_99999["average_max_temp"]
```

```
Out [ ]: 0      15.680645  
1      19.527586  
2      20.929032  
3      21.483333  
4      23.709677  
...  
3881   17.648387  
3882   26.370000  
3883   31.596774  
3884   32.268966  
3885   29.695833  
Name: average_max_temp, Length: 3177, dtype: float64
```

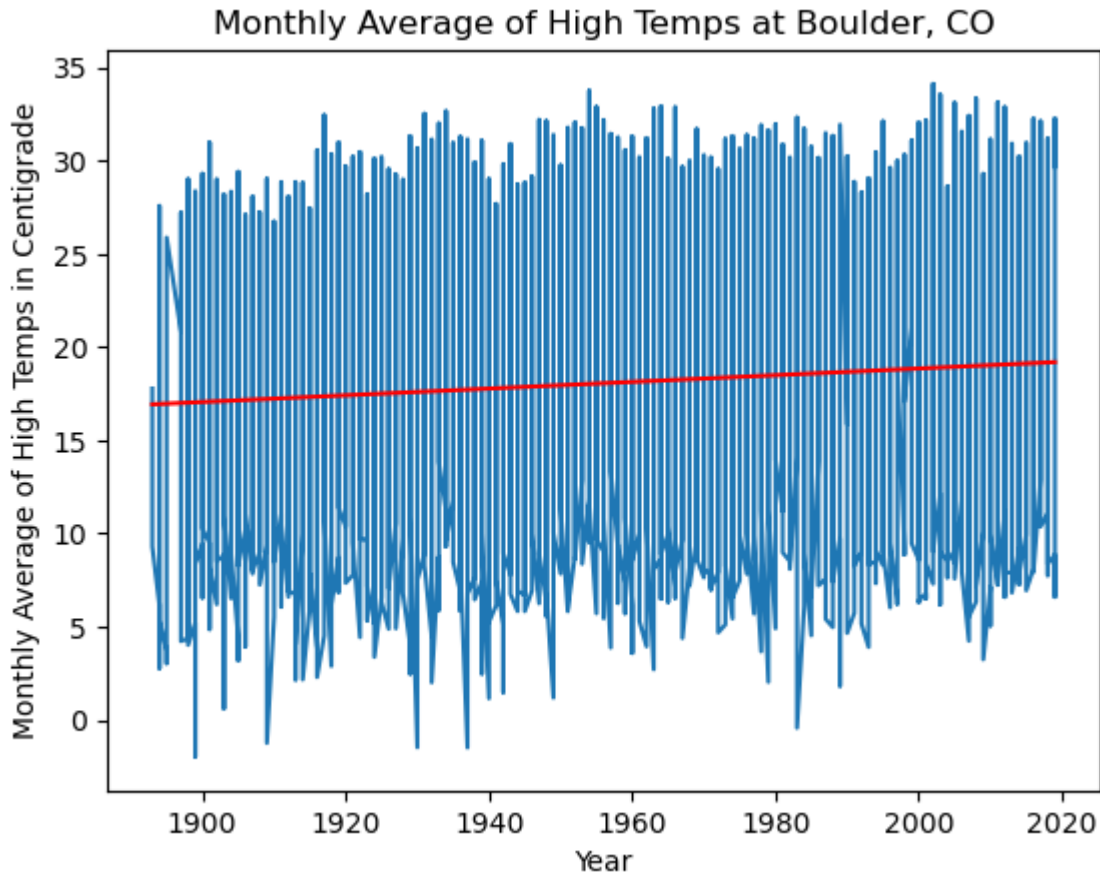
(6) Now for each weather station, generate a separate plot of the daily temperatures over time. You should end up with a plot that looks something like this:

 temp_plots_Colorado

NOTE: If your plot has little horizontal lines at the tops and bottoms of the temperature plots connecting perfectly vertical temperature lines, it means you made a mistake in how you plotted your data!

```
In [ ]: # Plotting for station 1  
df_new_station_1 = selected_station_chunk_3_without_99999[  
    selected_station_chunk_3_without_99999["id"] == "USC00050848"  
]  
  
year = df_new_station_1["year"].values  
average_max_temp = df_new_station_1["average_max_temp"].values  
slope, intercept = np.polyfit(year, average_max_temp, 1)  
regression_line = slope * year + intercept
```

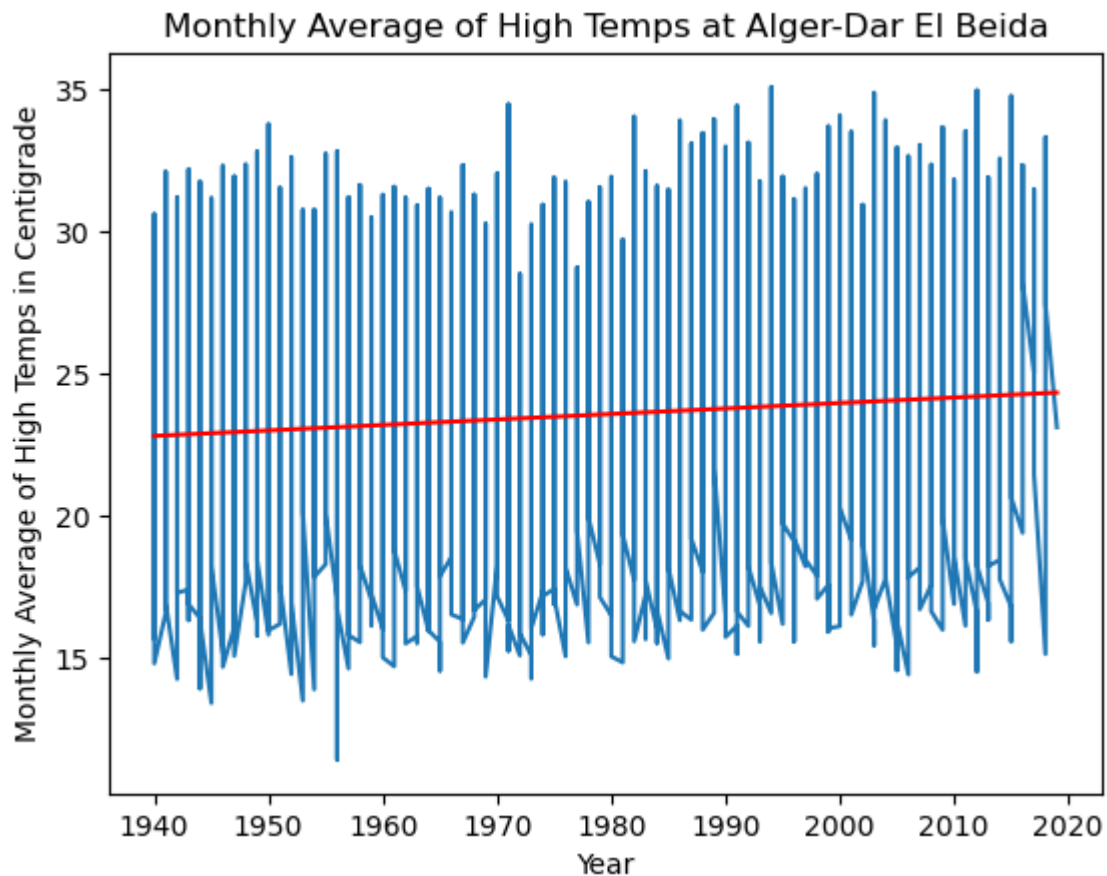
```
plt.plot(year, average_max_temp)
plt.plot(year, regression_line, label="Linear Regression Line", color="red")
plt.title("Monthly Average of High Temps at Boulder, CO")
plt.xlabel("Year")
plt.ylabel("Monthly Average of High Temps in Centigrade")
plt.show()
```



```
In [ ]: # Plotting for station 2
df_new_station_2 = selected_station_chunk_3_without_99999[
    selected_station_chunk_3_without_99999["id"] == "AG000060390"
]

year = df_new_station_2["year"].values
average_max_temp = df_new_station_2["average_max_temp"].values
slope, intercept = np.polyfit(year, average_max_temp, 1)
regression_line = slope * year + intercept

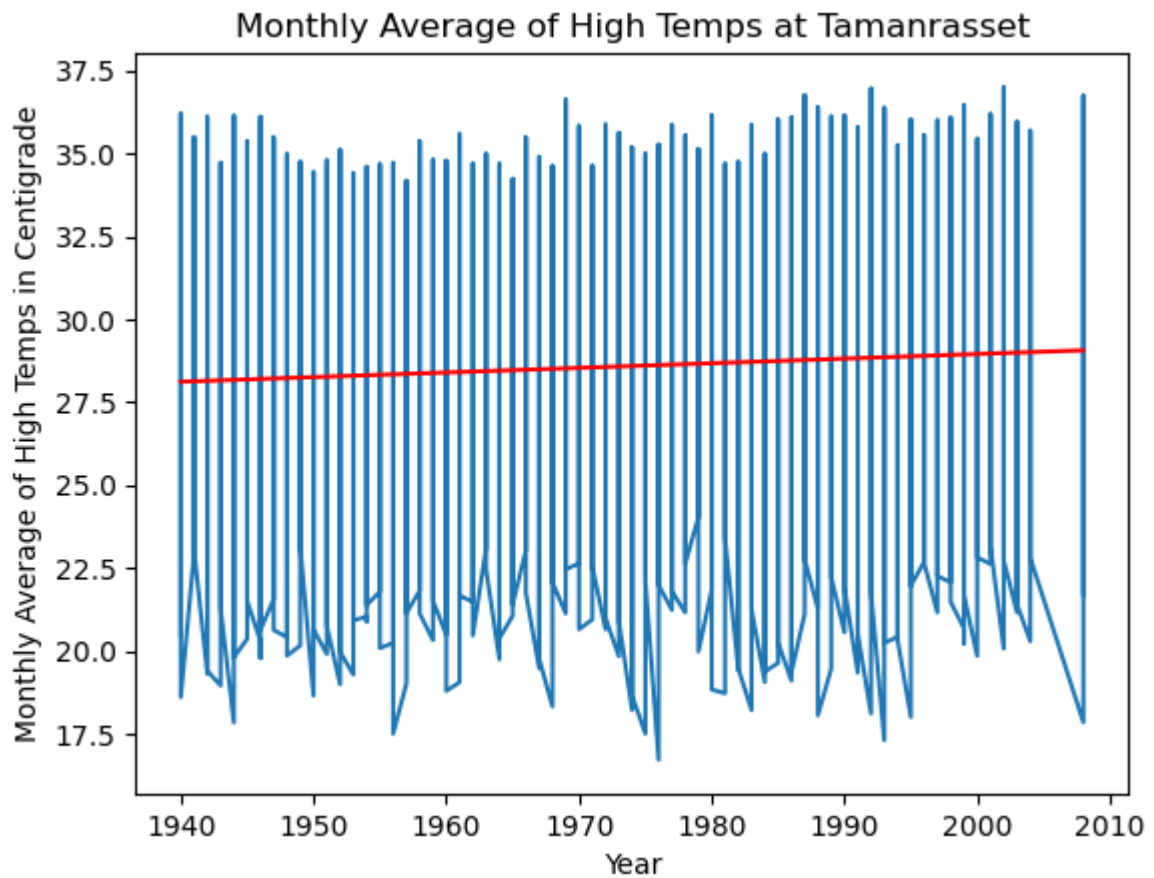
plt.plot(year, average_max_temp)
plt.plot(year, regression_line, label="Linear Regression Line", color="red")
plt.title("Monthly Average of High Temps at Alger-Dar El Beida")
plt.xlabel("Year")
plt.ylabel("Monthly Average of High Temps in Centigrade")
plt.show()
```



```
In [ ]: # Plotting for station 3
df_new_station_3 = selected_station_chunk_3_without_99999[
    selected_station_chunk_3_without_99999["id"] == "AG000060680"
]

year = df_new_station_3["year"].values
average_max_temp = df_new_station_3["average_max_temp"].values
slope, intercept = np.polyfit(year, average_max_temp, 1)
regression_line = slope * year + intercept

plt.plot(year, average_max_temp)
plt.plot(year, regression_line, label="Linear Regression Line", color="red")
plt.title("Monthly Average of High Temps at Tamanrasset")
plt.xlabel("Year")
plt.ylabel("Monthly Average of High Temps in Centigrade")
plt.show()
```

Want More Practice?

If you *really* want a challenge, the file `ghcnd_daily_30gb.tar.gz` will decompress into `ghcnd_daily.dat`, the full version of the GHCND daily data. It contains not only daily high temps, but also daily low temps, precipitation, etc. Moreover, it is still in fixed-width format, and is about 30gb in raw form.

Importing and chunking this data (with moderate optimizations) took about 2 hours on my computer.

If you're up for it, it's a great dataset to wrestling with data in weird formats and chunking.

Pro-tip: strings take up way more space in RAM than numbers, so some columns can be converted to keep the memory footprint of the data down.