A4 - Visualizing Big Data Sets

Work on this assignment in groups of up to 3 students.

Overview

- In this assignment, you will learn to work with a very large dataset. You will first clean and verify the data and then visualize various aspects by creating charts.
- In the next assignment, you will learn a method for analyzing the data and visualizing trends.

Background

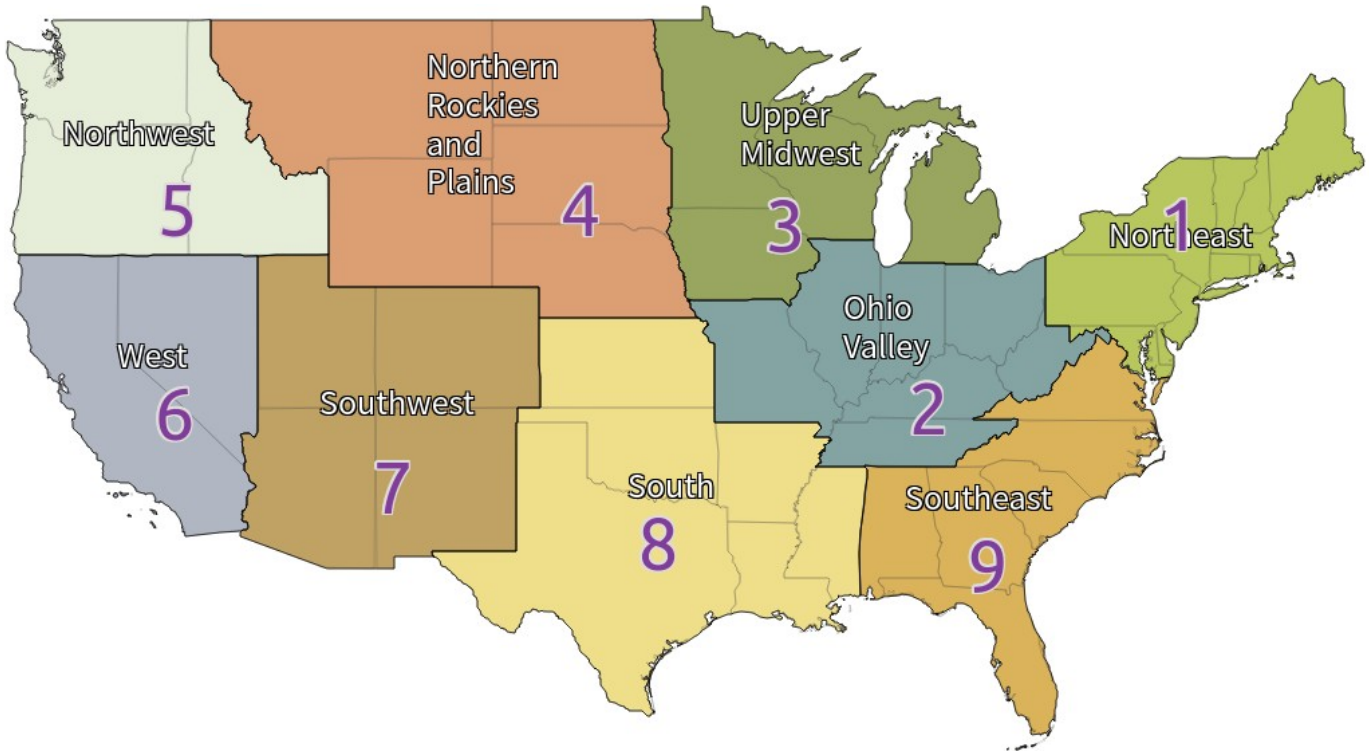
- You are given a dataset containing daily high and low temperatures for 210 U.S. cities, dating back to the late 1800s, and ending in 2023.
- Note that the temperatures are in Fahrenheit, not Celsius.
- Each city's data is in its own file.
- The file "city_info.csv" is a directory where you can find the file matching each city, the region to which the city belongs, as well as the date range of the temperature information for that city.

The dataset is courtesy of Yuchuan Lai at Carnegie Mellon University.

Note: The file "city_info.csv" is a modified version of the dataset's original directory file. You can still see the original if you wish - it has been renamed to "city_info_original.csv" - but we won't be using it in this assignment.

Regions

The cities are assigned to nine different regions. In the "regional" folder, you'll find directory files that tell you which states belong to which regions.



Missing Data

- In large datasets, it is common for some data to be missing, and we must decide how to deal with it.
- Fortunately, all dates are present from the starting date to the ending date in this dataset. So, we don't have to worry about missing dates.
- However, some of the dates are missing either the high temperature, or the low temperature, or both. (These data points are recorded as "NA".)
  - For some cities, only a few dates have missing temperatures, but for other cities, there are large ranges of dates with missing data. (For example, the data for Huntsville AL is missing both the high and the low for all dates from 1959 - 2017!)
- Our overall strategy is to fill in missing data by taking the average of the nearest preceding point and the nearest following point.
  - We consider a record to be unusable if it is missing the high or low temperature (or both) for a certain date **AND**
    1. there is no record in the ten days preceding that contains the data point; **OR**
    2. there is no record in the ten days following that contains the data point.

    In other words, if a data point is missing, there must be a data point BOTH in the previous ten days prior AND in the following ten days.

- We consider a city to be unacceptable if it contains any unusable record(s) for the date range we are interested in.
- For acceptable cities where either the high or the low is missing, we will fill in the missing temperature by averaging the closest earlier temperature and the closest later temperature.

Examples of Filling in Missing Temperatures

• The records for 1928-01-13, 1928-01-14, and 1928-01-15 are all usuable.
• The record for 1928-01-13 is missing the low temperature. To fill it in, we take the closest preceding low (37) and the closest following low (-3) and average them to get 17 ((37-3)/2).
• The record for 1928-01-14 is missing both the high and the low. The high will become (38+21)/2 = 29.5, and the low will become (37-3)/2 = 17.
• The record for 1928-01-15 is missing the high. The new high will be 29.5, by the same calculation as the high for 1928-01-14.
	`"12794",1928-01-12,48,37,0`
	`"12795",1928-01-13,38,NA,0.13`
	`"12796",1928-01-14,NA,NA,0.02`
	`"12797",1928-01-15,NA,-3,0.01`
	`"12798",1928-01-16,21,11,0`

Part A - Find Acceptable Cities in a Date Range

1. In "a4functions.py", write the function *get_acceptable_cities(start_year, end_year)* The function should return a dictionary mapping the nine regions to sorted lists of acceptable cities in that region.
2. With your submission, include a session log that shows testing of this function.

Some Expected Counts

For each of the following, you are given a text file containing the detailed breakdown of cities in regions.

- get_acceptable_cities(1874, 2023) should yield 23 cities.
- get_acceptable_cities(1914, 2023) should yield 98 cities.
- get_acceptable_cities(1954, 2023) should yield 150 cities.
- get_acceptable_cities(1994, 2023) should yield 167 cities.

Part B - Create Regional Files for a Set of Cities and a Date Range

1. Decide on a timeframe for your dataset for the remainder of the assignment. (In the samples, I have used 1904 - 2023.) Your timeframe should be at least 50 years.
2. From each of the nine regions, choose three acceptable cities, for a total of 27 cities. (Note: if there are fewer than three acceptable cities in any region, you'll need to choose a different timeframe.)
3. In "a4functions.py", write the function *create_regional_file(cities, start_year, end_year, file_name)* This function does several things:
   - For each date in the range, it averages the high temperatures for three cities (in a single region) to get an average high; and ditto for the low temperature.
     - For records where the high or the low is missing, fill in the missing data as described in the Background section above.
   - It then averages the high and low averages to get a single daily mean temperature for each date.
   - It converts the daily temperature from Fahrenheit to Celsius.
   - It writes the data to a new file, with the temperatures stored with one decimal place of precision.
     - The first line of the output file should be the column headers "Date,Temperature".
     - Make sure the dates are in YYYY-MM-DD format (as in the source files).
     - Make sure there are no leading or trailing spaces in any of the data points.
4. Write a script, called "create_regional_files.py", that uses your *create_regional_file* function to create a file for each region. The time span should be at least 50 years. (Note that you can use the function *get_acceptable_cities* to find suitable cities.)

Part C - Consolidate the Regional Files

1. In "a4functions.py", write a function called *consolidate_regions*. This function should consolidate the nine regional files (created in Part B) into a single file.
   - The first line of the new file should be the column headers "Region,Date,Temperature".
2. Run this function however you wish. (You may run it in interactive mode, or write a script to call it, or call it from "create_regional_files.py" after the regional files have been created.)
3. Write a script, called "check_consolidated_file.py" to prove that your data set is complete. The number of lines in the file, excluding the header line, should be `((365*`Y_N`) + (366*`Y_L`)) * 9` , where Y_N is the number of normal years, and Y_L is the number of leap years. (The multiplier 9 comes from the fact that there are 9 regions.)
4. Hand in your source code, a sample file, and a screen capture of running "check_consolidated_file.py" proving that the file is complete.

Part D - Create a Helper Class

1. To make the data easier to work with, create a helper class, called TemperatureHelper, that:
   - loads the data from a consolidated file (created in Part C) when an instance is constructed
   - separates the dates into year, month, and day
   - stores the data as nested dictionaries, where:
     - the region name is the key for level 1, and the value is a dictionary of years
     - the year is the key for level 2, and the value is a dictionary of months
     - the month is the key for level 3, and the value is a dictionary of days
     - the day is the key for level 4, and the value is the temperature.

   Example

   If a consolidated file contains data for the nine regions for all dates from 1990 to 1999, and if the helper class stores this in a field called `data`, then:

   `data` is a dictionary with region names as keys

   `data['Region2']` is a dictionary with years as keys

   `data['Region2'][1995]` is a dictionary with months as keys

   `data['Region2'][1995][2]` is a dictionary with days as keys

   `data['Region2'][1995][2][28]` is a the daily temperature

   - Provides a method, *get_daily_temperature(region, year, month, day)*. This method returns the temperature for the specified region and date.
   - Provides a method, *get_yearly_temperatures(region, year)*. This method returns a numpy array containing the temperatures for the specified region for each day of the specified year.
     - You will have to import numpy.
     - A NumPy array is similar to a list, but it's easier to perform mathematical operations on it. We'll use it later.
2. Write a script, called "try_TemperatureHelper.py", that creates an instance of TemperatureHelper using the consolidated file that you created in Part C and tests the two functions with a few different calls each.
3. Hand in your source code, your test script, and a screen capture showing the testing.

Part E - Create Charts

1. Learn to use Matplotlib (see the demos and the Matplotlib Website) and write the following functions in "a4functions.py". (See the code documentation for details.)
   - *annual_means_per_region(regions, years, consolidatedFile)*
   - *annual_means_combined(regions, years, patterns, consolidatedFile)*
   - *single_day_per_region(regions, years, month, day, consolidatedFile)*
   - *single_day_combined(regions, years, month, day, patterns, consolidatedFile)*
2. Choose four days spaced well throughout the year (e.g., Feb 3, May 6, Aug 9, Nov 12). Write a script, called "create_charts.py", that uses the functions you wrote in step 1 and the consolidated file from Part C to create 50 charts (PNG files) showing:
   a. the temperatures on day1 of each year for each region (nine charts). Name these files "[day1 - cityN].png". For example, if day1 is Feb 3, then the file names should be:
      - Feb 3 - Region1.png
      - Feb 3 - Region2.png
      - Feb 3 - Region3.png
      - Feb 3 - Region4.png
      - Feb 3 - Region5.png
      - Feb 3 - Region6.png
      - Feb 3 - Region7.png
      - Feb 3 - Region8.png
      - Feb 3 - Region9.png

   b. the temperatures on day2 of each year for each region (nine charts)
   c. the temperatures on day3 of each year for each region (nine charts)
   d. the temperatures on day4 of each year for each region (nine charts)
   e. the temperatures on day1 of each year for all regions, as nine series on a single chart. Name this file "[day1] - All Cities.png". For example, if day1 is Feb 3, then the name is "Feb 3 - All Regions.png"
   f. the temperatures on day2 of each year for all regions on a single chart
   g. the temperatures on day3 of each year for all regions on a single chart
   h. the temperatures on day4 of each year for all regions on a single chart
   i. the annual means per city, one chart per region (nine charts). Name these files "Annual Means - [RegionN].png"
   j. the annual means of all regions as nine series on a single chart. Name this file "Annual Means - All Regions.png"
3. Hand in your source code and your 50 charts.

Submission Checklist

Python Files

- a4functions.py - contains the functions:
  - *get_acceptable_cities*
  - *create_regional_file*
  - *consolidate_regions*
  - *annual_means_per_city*
  - *annual_means_combined*
  - *single_day_per_region*
  - *single_day_combined*
- create_regional_files.py
- check_consolidated_file.py
- TemperatureHelper.py (class)
- try_TemperatureHelper.py
- create_charts.py

Logs and Screen Shots

- Session log of testing get_acceptable_cities
- Running the script "check_consolidated_file.py"
- Running the script "try_TemperatureHelper.py"

Artifacts

- Sample file created by "create_consolidated_file.py"
- 50 images created by "create_charts.py"