CSCI 203 FINAL PROJECT - PART 1
Fall 2022
Analysis of Heat-Related Hospitalizations

PROJECT INTRODUCTION

Making good decisions depends on accurate knowledge of our world. For example, could knowing the predicted temperature and humidity help hospitals plan for staffing needs for an upcoming heat wave? Are different areas of the United States better equipped to handle extreme-heat events? For Part 1 of this project, you will explore the correlation of the heat index with heat-related hospitalizations.[1] For Part 2 of this project, you will pose a question about a heat-related effect on human health, select data to answer your question, and then produce a visualization of the answer to your question.

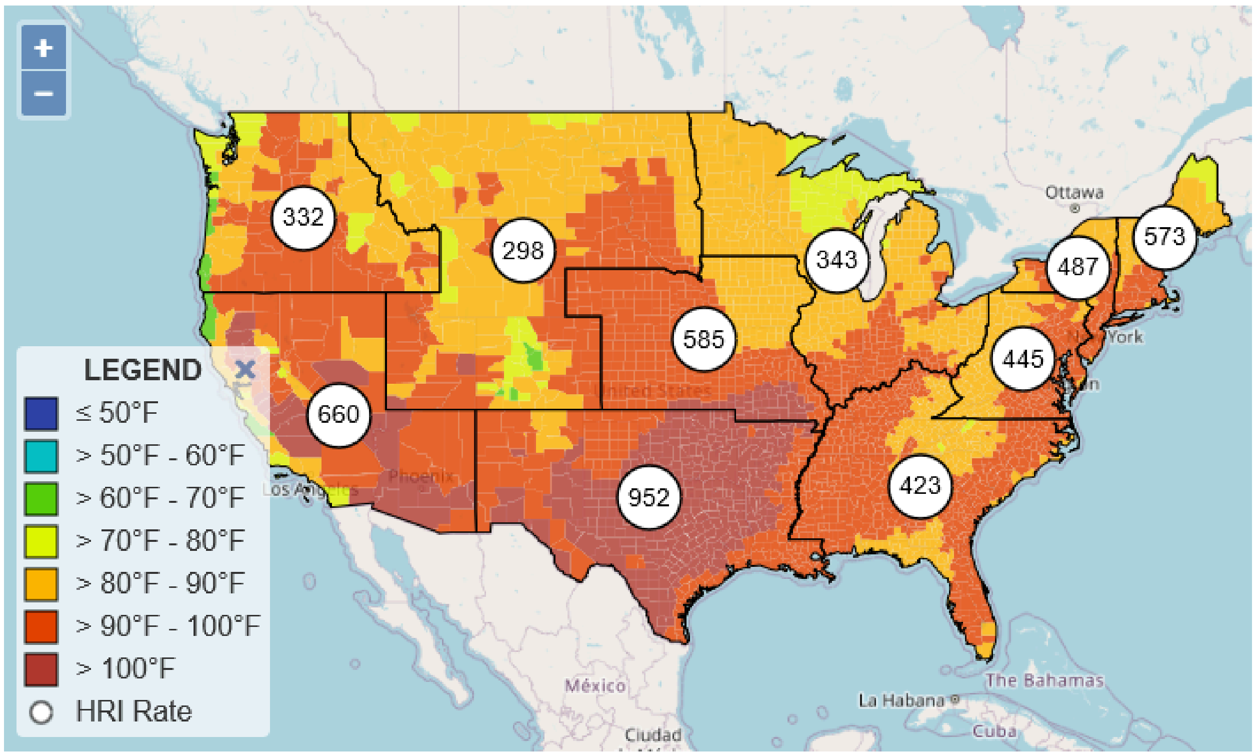

Figure 1. The average maximum temperature on July 20, 2022 for each county is shown with the color scale. The heat-related emergency department visits per 100,000 emergency department visits are shown with the circled numbers.
https://ephtracking.cdc.gov/Applications/heatTracker/

[1] This is an extension of a CSCI 203 project done by Wutt Kyi '22. Thank you Wutt for the inspiration!

To be completed **INDIVIDUALLY**

You ***cannot*** ***visually or electronically*** *share code with each other. Any resources you use online or any help you receive* ***must be clearly cited****.*

Where should you code?

The final project provides you with the opportunity to show us what you have learned this semester. All coding must be completed on your own in the project replit on replit.com. The progression of your work is then clearly documented. Your submissions on gradescope.com must match with the documented work on replit.com.

Can I use the internet?

- 👍 You **are** allowed to use online resources to help you *understand* a topic in this course.
- ❗ Any online sources that you use **must be cited as a comment in your code.**
- 🚫 You are **not** allowed to search for *specific* code or solutions to specific problems you're assigned

What kind of collaboration is allowed?

- 👍 You **are** allowed to *talk* to students about approaches to a problem, resources used, and about general strategies.
- 👍 You **are** allowed to get help from professors and TAs.
- 🚫 You are **not** allowed to show another student your code for this project. This means that you should not let another student "borrow" the code. This is the *most common case* that we (regretfully) bring to the Board of Review.
- 🚫 You are also not allowed to publicly post your code to assignments in this class - either on Github or in a publicly-accessible cloud storage folder (like Drive or Dropbox).

PROJECT DEADLINES

PARTS 1 DEADLINE: Monday, November 28, 2022 at 11 PM
PART 2 PROPOSAL: Tuesday, November 29, 2022 at 11 PM
PARTS 1 and 2 DEADLINE: Monday, December 5, 2022 at 11 PM
PRESENTATIONS OF PART 2: Tuesday, December 6, 2022 in lab

PART 1 ANALYSIS OF HEAT-RELATED HOSPITALIZATIONS

For Part 1 of the Final Project, you will study the correlation between the number of days between May and September when the heat index exceeds 90°F and the number of heat-related hospitalizations.

DATA FOR YOUR ANALYSIS

You are provided with five csv files to use in your analysis. These files are:

- A csv file **heat_index.csv** that includes the number of days between May and September when the heat index exceeds 90°F for each state except Alaska and Hawaii from 1979 to 2021.
    - The file was accessed on 11/08/2022 from https://ephtracking.cdc.gov/DataExplorer using the following search terms:

Search Selection	**Terms Selected**
Content	Content Area: Heat and Heat-related Illness Indicator: Historical Temperature & Heat Index Measure: Annual Number of Extreme Heat Days from May to September
Geography Type	Geographic Type: National by County
Geography	All Counties
Time	All Years
Advanced Options	Absolute Threshold: 90 degrees F Heat Metric: Daily Maximum Heat Index

    - Details on how the heat index was calculated can be found at: https://ephtracking.cdc.gov/indicatorPages with the selections:
        - Content Area: Heat & Heat-related Illness
        - Indicator: Historical Temperature & Heat Index

- A csv file **hospitalizations.csv** that includes the annual number of admissions to hospitals for heat-related illnesses for days between May 1 and September 30. The entries are organized by state and county. Data are missing for some states and some dates when numbers were not provided to the CDC or did not meet the standards set for data collection.

- The file was accessed on 11/06/2022 from https://ephtracking.cdc.gov/DataExplorer using the following search terms:

Search Selection	**Terms Selected**
Content	Content Area: Heat and Heat-related Illness Indicator: Hospitalizations for Heat-related Illness Measure: Annual Number of Hospitalizations
Geography Type	Geographic Type: National by State
Geography	All States
Time	All Years
Advanced Options	Age Group: 0 - 4, 5 - 14, 15-34, 35 -64, >=65 Gender: Male, Female

- Details on heat-related illnesses can be found at: https://ephtracking.cdc.gov/indicatorPages with the selections:
  - Content Area: Heat & Heat-related Illness
  - Indicator: Hospitalizations for Heat-related Illness
- The method for coding heat-related illnesses changed on October 1, 2015. To compare heat-related illnesses using the same reporting system, only hospitalizations before 2016 will be used.

- A csv file **test_heat_index.csv** that is an abridged version of heat_index.csv for testing your code.
- A csv file **test_data.csv** includes data from one county in New Jersey (Bergen) and one county in Pennsylvania (Philadelphia) for the years 2012 to 2021. The heat index value was removed from the row for the year 2013 for Bergen county to allow testing of the read_data_file function when a value is missing from the file.
- A csv file **test_hospitalizations.csv** that is an abridged version of hospitalizations.csv for testing your code.

CODE FOR YOUR ANALYSIS

You are provided with a template file - `main.py` - where you will do your coding. This file includes headers for code that you need to complete for three classes (Data, Hospitalizations, and ExtremeHeatDays) and functions to analyze the data. Here is a brief description of the functions and classes used in this project:

- **`read_data_file`** - a function to read a csv file and return two lists: `years` (the years when the data was measured) and `data` (hospitalization or heat index numbers).
- **`Data class`** **-** a class to read and organize data for a given state. This class has the following methods:
  - `__init__(self, filename, year_index, state, data_name, data_index)`: the constructor for the `Data` class.
  - __repr__(self): the method to represent the `Data` class.
  - `group_yearly_nums(self)`: a method to organize data by year
- **`Hospitalizations class`** **-** a class to organize the hospitalizations as yearly sums. This class is a child of the `Data` class. This class has the following methods:
  - `__init__(self, filename, state)`: the constructor of the `Hospitalizations` class
  - `__repr__(self)`: the method to represent the `Hospitalizations` class.
  - `make_yearly_sum(self)`: a method to sum the yearly hospitalizations. The method returns a dictionary where the key is the year as an integer and the value is the sum of the hospitalizations for that year.
- **HeatIndex class -** a class to find the yearly average of the number of days reported by the state counties. This class is a child of the `Data` class. This class has the following methods:
  - `__init__(self, filename, state)`: the constructor for the `HeatIndex` class
  - `__repr__(self)`: the method to represent the `HeatIndex` class.
  - `make_yearly_av(self)`: a method that returns a dictionary where the key is the year as an integer and the value is the yearly average of the number of days with a heat index over 90°F recorded for a state.
- **`get_years_nums_from_d(d)`**: a function that returns the lists of the keys and values of a dictionary `d` for use with the function `plot_data_by_year`.
- **`plot_data_by_year(state, heat_years, heat_nums,hosp_years, hosp_nums)`**: a function to plot for a given state:
  - the yearly sum of hospitalizations as a function of years
  - the yearly average of the number of days with a heat index over 90°F as a function of years.
- **`get_hosp_heat_from_d(d_heat, d_hosp)`**: a function that returns lists with data for the years that hospitalization numbers are available for use with the function `plot_heat_hosp(heat_nums, hosp_nums)` .
- **`get_stats(x, y)`**: a function to perform a linear regression on data.
- **`plot_heat_hosp(state, heat_nums, hosp_nums)`**: a function to plot for a given state the yearly sum of heat-related hospitalizations as a function of the yearly average of the number of days with a heat index over 90 degrees.

- **`select_a_state()`**: a function to obtain the state to be analyzed from the user.
- **`menu()`**: a function to print the options given by the strings in the list `OPTIONS` defined at the top of the file `main`.
- **`select_option()`**: a function to obtain a selection from the user from options provided in the list `OPTIONS`.
- **`main()`**: the function that runs the analysis

TO DO - Complete the function `read_data_file`

Complete the function **`read_data_file(filename, year_index, state, data_index)`** to read a csv file and return two lists:

- `years`, a list of integers, years that the data was recorded
- `data`, a list of integers - the data

The parameters are:

- `filename`: a string, the name of the csv file
- `state`: a string, the name of the state for which the data will be analyzed
- `year_index`: an integer, the column of the csv file with the year of the measurements
- `data_index`: an integer, the column of the csv file with the data

For **an individual row** in the csv file, the function should only add a value to the list `years` **and** a value to the list `data` if:

- the values for the year, the data, and the state read from the csv file are not empty strings,
- the parameter `state` is the same as the value for state read from the csv file, and
- the integer value for the year read from the csv file is less than 2016.

The indices needed to access columns in the data files are given at the top of the main.py file as:

- `STATE_INDEX = 1`
- `YEAR_INDEX_HOSP = 2`
- `HOSP_INDEX = 3`
- `YEAR_INDEX_HEAT = 4`
- `HEAT_INDEX = 5`

- All csv files have the state name in the column with an index of 1.

- For the files with the hospitalization data, the year and the number of hospitalizations from May to September for the year are given in columns 2 and 3, respectively.
- For the files with the heat index data, the year and the number of days when the heat index exceeded 90°F from May to September for the year are given in columns 4 and 5, respectively.

TIP

You may find it easier to read the csv file one row at a time instead of reading it as a dictionary (as you did for your midterm project). The following code is already added to the function `read_data_file` to read one row at a time:

```
# creates a file object
with open(filename, newline='') as csvfile:
  # set to read a csv file
  csvreader = csv.reader(csvfile, delimiter=',')
  # reads one line to skip over the header
  next(csvreader)
  # reads one row at a time after the header
  for row in csvreader:
    print(row)
    print("row is data type: ", type(row), "\n")
```

Try the following:

- Remove the hashtags in the function `read_data_file` so the row and data type of the row will be printed when the function is called.
- Hit "Run".
- Enter the following line at the console:

```
read_data_file("test_data.csv", YEAR_INDEX_HEAT, "New Jersey",
HEAT_INDEX)
```

What is the data type of `row`?
How can you use the `row` to complete the function `read_data_file`?
Remember that this code should work for both csv files: hospitalization.csv and heat_index.csv. Don't hardcode the columns to be used. Use the values for `year_index` and `data_index` passed into the function `read_data_file`.

Remove the continue line, print(row) and print(type(row)) lines after you have completed this function.

Here is an example of how the function `read_data_file` should work with the file test_data.csv: (Note: The file test_data.csv is missing data for 2013 so that year should not be read.)

```
>years, data = read_data_file("test_data.csv", YEAR_INDEX_HEAT, "New
Jersey", HEAT_INDEX)
>years
[2012, 2014, 2015]
>data
[35, 20, 29]
```

TO DO - Complete the `Data` class

The `Data` class organizes the data for a given state into lists of values for each year. You are provided with the following methods:

- `__init__(self, filename, year_index, state, data_name, data_index)`: the constructor for `Data` class. The parameters are:
    - `filename`: a string, the name of the csv file
    - `year_index`: an integer, the column of the csv file with the year of the measurements
    - `state`: a string, the name of the state for which the data will be analyzed
    - `data_name`: a string, indicates the type of data: "Hospitalizations" or "Heat Index".
    - `data_index`: an integer, the column of the csv file with the data
- __repr__(self): the method to represent the `Data` class.

Here is an example of creating an instance of the `Data` class and showing the representation of the instance.

```
>data_instance = Data("test_data.csv", YEAR_INDEX_HEAT, "New Jersey",
"Heat Index", HEAT_INDEX)
>data_instance
------------------------------

     Data for New Jersey

   Year         Heat Index

------------------------------
```

```
   2012            35
   2014            20
   2015            29
```

TO DO - Complete the method `group_yearly_nums` for the `Data` class

The method `group_yearly_nums` organizes the data by year and returns a dictionary with year as the key and lists of the data for that year as the value. The method should work as follows:

```
>data_instance = Data("test_hospitalizations.csv", YEAR_INDEX_HOSP,
"New Jersey", "Hospitalizations", HOSP_INDEX)
>data_instance
------------------------------
     Data for New Jersey
   Year     Hospitalizations
------------------------------
   2012            70
   2012            14
   2012            42
   2012            37
   2013            67
   2013            15
   2013            47
   2013            37
   2014            22
   2014            4
   2014            18
   2014            10
   2015            49
   2015            3
   2015            23
   2015            22
>data_instance.group_yearly_nums()
{2012: [70, 14, 42, 37],
 2013: [67, 15, 47, 37],
 2014: [22, 4, 18, 10],
 2015: [49, 3, 23, 22]}
```

TO DO - Complete the Hospitalizations class

The `Hospitalizations` class, a child of the `Data` class, organizes the hospitalizations as yearly sums and has the following three methods:

- `__init__(self, filename, state)`: the constructor of the `Hospitalizations` class
- `__repr__(self)`: the method to represent the `Hospitalizations` class.
- `make_yearly_sum(self)`: a method to sum the yearly hospitalizations. The method returns a dictionary where the key is the year, an integer, and the value is the sum of the hospitalizations for that year.

The following text describes the provided `__init__` method and the two methods (`__repr__` and `make_yearly_sum`) that you need to complete.

GIVEN - the method __init__ for the Hospitalizations class

The parameters of the method `__init__(self, filename, state)` are:

- `filename`: a string, the name of the csv file
- `state`: a string, the name of the state for which the data will be analyzed

To use the constructor of the parent class (`Data` class) for the child class (`Hospitalizations` class), `super` is called:

```
super().__init__(filename, YEAR_INDEX_HOSP, state, "Hospitalizations",
HOSP_INDEX)
```

The parameters for the call of `super` , the `__init__` method for the `Data` class are:

- `filename`: a string, the name of the csv file. The `filename` is passed in as a parameter of `__init__` for the `Hospitalizations` class.
- `year_index`: an integer, the column of the csv file with the year. For the Hospitalization class, the `year_index` is 2. Notice that the constant `YEAR_INDEX_HOSP` is defined as 2.
- `state`: a string, the name of the state for which the data will be analyzed. The `string` is passed in as a parameter of `__init__` for the `Hospitalizations` class.
- `data_name`: a string, indicates the type of data: "Hospitalizations"

- `data_index`: an integer, the column of the csv file with the yearly number of heat-related hospitalizations. For the Hospitalization class, the `data_index` is 3. Notice that the constant `HOSP_INDEX` is defined as 3.

The attributes of the `Hospitalizations` class are:

- The attributes from the `Data` class:
    - `self.state`
    - `self.data_name`
    - `self.years`
    - `self.data`
- `self.d_hosp`, a dictionary where the key is the year, an integer, and the value is the sum of the hospitalizations for that year.

NOTE: You need to complete the `make_yearly_sum` method for `self.d_hosp` to be the correct dictionary.

TO DO - Write the method `make_yearly_sum` for the `Hospitalizations` class

Write the method `make_yearly_sum(self)`to sum the yearly hospitalizations. The method returns a dictionary where the key is the year, an integer, and the value is the sum of the hospitalizations for that year. The method should work as follows:

```
>hospitalizations_instance =
Hospitalizations("test_hospitalizations.csv", "New Jersey")
>hospitalizations_instance.make_yearly_sum()
{2012: 163, 2013: 166, 2014: 54, 2015: 97}
```

TIP
You can call the method `group_yearly_nums` from the parent class to get lists that are grouped by year.

TO DO - Write a __repr__ method for the Hospitalizations class

Add a `__repr__` method to the Hospitalization class. This method will not be autograded so you can be creative with the format. You need to display the:

- the state name
- the years and number of hospitalizations in tabular form with titles for the columns.

Here is an example:

```
>hospitalizations_instance =
Hospitalizations("test_hospitalizations.csv", "New Jersey")
>hospitalizations_instance
-----------------------------------
           Number of Heat-Related
   Year       Hospitalizations
                for New Jersey
-----------------------------------
   2012             163
   2013             166
   2014             54
   2015             97
```

TO DO - Complete the HeatIndex class

The `HeatIndex` class is a child of the `Data` class and organizes the number of days with a heat index over 90°F as yearly averages. You are to complete the three methods of the HeatIndex class:

- `__init__(self, filename, state)`: the constructor of the `HeatIndex` class
- `__repr__(self)`: the method to represent the `HeatIndex` class.
- `make_yearly_av(self)`: a method to average the number of days with a heat index over 90°F as yearly averages. The method returns a dictionary where the key is the year, an integer, and the value is the average of the number of days with a heat index over 90°F for that year.

TO DO - Complete the method __init__ for the HeatIndex class

The parameters of the method `__init__(self, filename, state)` are:

- `filename`: a string, the name of the csv file
- `state`: a string, the name of the state for which the data will be analyzed

The attributes of the `HeatIndex` class are:

- The attributes from the `Data` class:
    - `self.state`
    - `self.data_name`

    - `self.years`
    - `self.data`
- `self.d_heat`, a dictionary where the key is the year, an integer, and the value is the average of the number of days with a heat index over 90°F for that year.

Write the `__init__` method for the `HeatIndex` class. Use the built-in Python `super()` function as you did for the `Hospitalizations` class. How will you define the parameters for `year_index`, `data_name`, and `data_index` in the call to `super()`? (You can use "Heat Index" as the `data_name`.) **NOTE:** You need to complete the `make_yearly_av` method for `self.d_heat` to be the correct dictionary.

TO DO - Write the method `make_yearly_av` for the `HeatIndex` class

Write the method `make_yearly_av(self)` that returns a dictionary where the key is the year, an integer, and the value is a float, the average of the number of days with a heat index over 90°F over all the counties in the state for that year. Round the average to two decimal places. The method should work as follows:

```
>heat_index_instance = HeatIndex("test_heat_index.csv", "New Jersey")
>heat_index_instance.make_yearly_av()
{2010: 49.67, 2011: 35.0, 2012: 41.0, 2013: 39.67, 2014: 24.67, 2015:
34.33}
```

TO DO - Write a `__repr__` method for the HeatIndex class

Add a `__repr__` method to the `HeatIndex` class. This method will not be autograded so you can be creative with the format. You need to display the:

- the state name
- the years and average number of days per year for a state when the heat index was over 90°F. The columns in your table should be labeled.

Here is an example:

```
>heat_index_instance = HeatIndex("test_heat_index.csv", "New Jersey")
>heat_index_instance
-----------------------------------

            Number of Days with a

   Year   Heat Index > 90 degrees F

                for New Jersey
```

```
-----------------------------------
   2010              49.67
   2011              35.00
   2012              41.00
   2013              39.67
   2014              24.67
   2015              34.33
```

TO DO - Complete helper functions to visualize your analysis

Two plotting functions are provided to visualize your data:

- `plot_data_by_year(heat_years, heat_nums,hosp_years, hosp_nums)`
- `plot_heat_hosp(heat_nums, hosp_nums)`

Three helper functions are used to supply data for the plotting functions:

- `get_years_nums_from_d(d)`
- `get_hosp_heat_from_d(d_heat, d_hosp)`
- `get_stats(x, y)`

None of the above functions are part of any of the class definitions. You need to complete `get_years_nums_from_d(d)` and `get_hosp_heat_from_d(d_heat, d_hosp)`. The function `get_stats(x, y)` is provided. See the following text for more information on each function.

TO DO - Complete the function `get_years_nums_from_d`

Complete the function **`get_years_nums_from_d(d)`** with the parameter `d`, a dictionary, that returns lists for use with the function `plot_data_by_year`. The lists are:

- `years`, a list of integers, the keys of the dictionary `d`
- `nums`, a list of floats or integers, the values of the dictionary `d`

An example of how the function should work is:

```
>hosp = Hospitalizations("test_hospitalizations.csv", "New Jersey")
>hosp.d_hosp
{2012: 163, 2013: 166, 2014: 54, 2015: 97}
>years, nums = get_years_nums_from_d(hosp.d_hosp)
```

```
>years
[2012, 2013, 2014, 2015]
>nums
[163, 166, 54, 97]
```

TO DO - Complete the function `get_hosp_heat_from_d`

Complete the function **`get_hosp_heat_from_d(d_heat_plot, d_hosp_plot)`** that returns lists for use with the function `plot_heat_hosp`. The lists should include data for **only** the years that hospitalization numbers are available. The lists are:

- `heat_nums`, a list of floats, the yearly averages of the days when the heat index exceeds 90°F
- `hosp_nums`, a list of integers, the yearly sums of heat-related hospitalizations

The parameters of the function are:

- `d_heat_plot`, a dictionary where the key is the year, an integer, and the value is a float, the average of the number of days with a heat index over 90°F for that year.
- `d_hosp_plot`, a dictionary where the key is the year, an integer, and the value is the sum of the hospitalizations for that year.

The function should work as follows:

```
>hosp = Hospitalizations("test_hospitalizations.csv", "New Jersey")
>hosp.d_hosp
{2012: 163, 2013: 166, 2014: 54, 2015: 97}
>heat = HeatIndex("test_heat_index.csv", "New Jersey")
>heat.d_heat
{2010: 49.67, 2011: 35.0, 2012: 41.0, 2013: 39.67, 2014: 24.67, 2015:
34.33}
>get_hosp_heat_from_d(heat.d_heat, hosp.d_hosp)
([41.0, 39.67, 24.67, 34.33], [163, 166, 54, 97])
```

GIVEN - the function `get_stats`

The function `get_stats(x , y)` with parameters x, the independent variable, and y, the dependent variable, returns:

- `correlation`, a float, the correlation coefficient between the two variables
- `y_predicted`, a list, the values for the predicted y values for given x values

The values for y_predicted are calculated using: y_predicted = slope * x + intercept where the slope and intercept are determined from a linear regression using the function `polyfit(x, y, 1)` from the Python numpy library. The correlation coefficient is determined with the function `corrcoef(x, y)` that returns a 2-D correlation array:

[[correlation between x and x, correlation between x and y],
[correlation between x and y, correlation between y and y]]

We are interested in the correlation coefficient between x and y ie. the [0][1] or [1][0] position in the 2-D array. (If you are unfamiliar with the term correlation coefficient, look at this website. In brief, two variables are perfectly correlated, perfectly negatively correlated, or not correlated if the correlation coefficient is +1, -1, or 0, respectively. The closeness of the absolute value of the correlation coefficient to 1 is a measure of how correlated the two variables are.) Here is an example of how the function works:

```
>x = [41.0, 39.67, 24.67, 34.33]
>y = [163, 166, 54, 97]
>get_stats(x, y)
(0.968,
 [163.1466501575806, 153.7121997326596, 47.30862351174582,
115.83252659801428])
```

VISUALIZATION OF YOUR ANALYSIS

Two plotting functions are provided to visualize your data:

- `plot_data_by_year(heat_years, heat_nums,hosp_years, hosp_nums)`
- `plot_heat_hosp(heat_nums, hosp_nums)`

GIVEN - the function `plot_data_by_year`

The function **`plot_data_by_year(state, heat_years, heat_nums, hosp_years, hosp_nums)`** is provided for you. This function plots:

- the yearly sum of heat-related hospitalizations as a function of years
- the yearly average of the number of days with a heat index over 90°F as a function of years.

The parameters of the function are:

- `state`, a string, the name of the state
- `heat_years`, a list of integers, the years for which the heat index data will be plotted

- `heat_nums`, a list of floats, the yearly average numbers of days when the heat index exceeded 90°F
- `hosp_years`, a list of integers, the years for which the hospitalization data will be plotted
- `hosp_nums`, a list of integers, the yearly sums of heat-related hospitalizations

The command `plt.savefig("DataByYear.png")` at the end of the function saves the plot. Figure 2 shows an example of the plot.

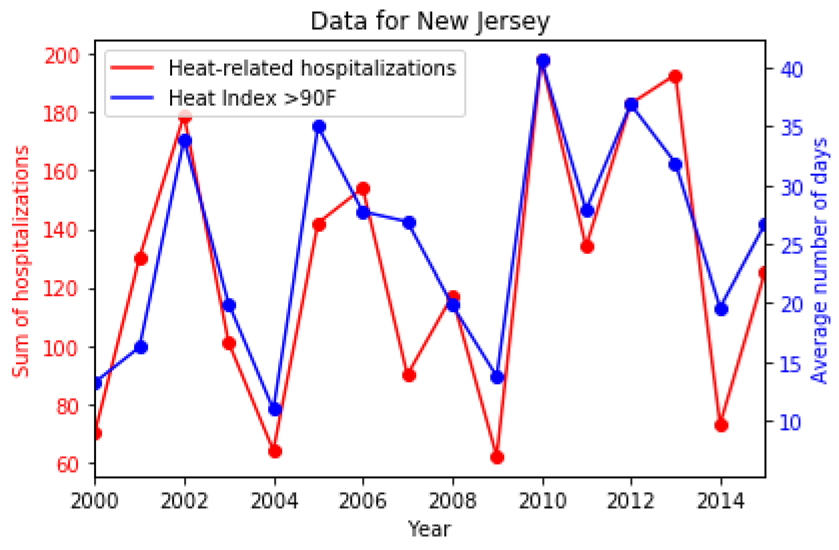

Figure 2. An example of output from the `plot_data_by_year` function using data for New Jersey with the files `"hospitalizations.csv"` and `"heat_index.csv"`

GIVEN - the function `plot_heat_hosp`

The function **`plot_heat_hosp(state, heat_nums, hosp_nums)`** plots the yearly sum of heat-related hospitalizations as a function of the yearly average of the number of days with a heat index over 90°F. The parameters are:

- `state`, a string, the name of the state
- `heat_nums`, a list of floats, the yearly average values of the heat index that includes data for only the years that data are available for hospitalizations
- `hosp_nums`, a list of integers, the yearly sum of heat-related hospitalization

First, a scatter plot is made with the `heat_nums` on the horizontal (x) axis and hosp_nums on the vertical (y) axis.
Second, the function `get_stats(x, y)` is called using `heat_nums` as the independent variable (x) and `hosp_nums` as the dependent variable (y). A solid line is added to the plot using the predicted values for heat_nums determined with a linear regression (the

returned `y_predicted` from function call to `get_stats(x, y)`) and the values for hosp_nums. This provides a visual idea of how well the data are correlated.

Third, a label is added to the plot with the correlation coefficient returned from `get_stats(x, y)` using the following command:

```
plt.text(x_for_text, y_for_text, "Correlation = " + str(correlation))
```

where `x_for_text` and `y_for_text` are the x and y coordinates, respectively, for the position of the text and `correlation` is the correlation coefficient returned by the call to `get_stats`. Note the coordinates `x_for_text` and `y_for_text` are in terms of the scale on the axis of the graph.

The plot is saved using the command `plt.savefig("HeatHospData.png").` Figure 3 shows an example of how the plot looks using the files `"hospitalizations.csv"` and `"heat_index.csv"`.

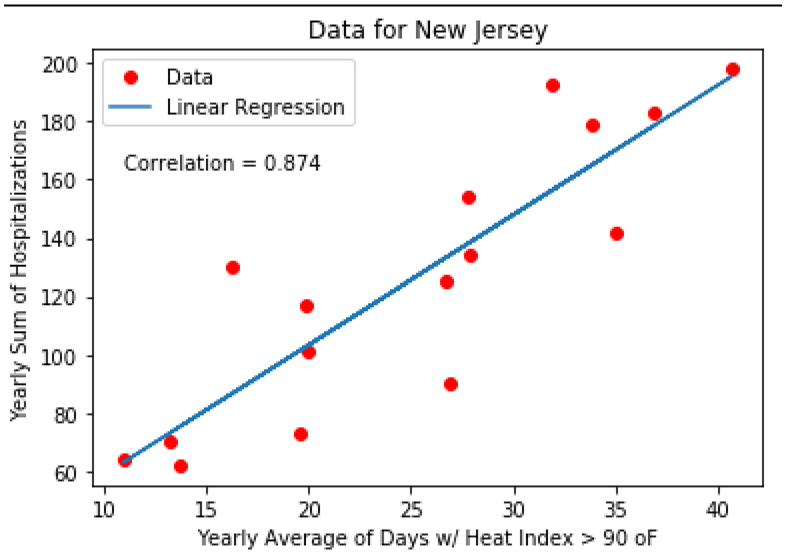

Figure 3. An example of output from the `plot_heat_hosp` function using data for New Jersey with the files `"hospitalizations.csv"` and `"heat_index.csv"`

AUTOMATION OF YOUR ANALYSIS

If you were to run your analysis for different states, you probably would not want to have to enter each needed command as we have done for testing your functions. Let's automate the process. To do such, the following functions will be used:

- **`select_a_state()`**: a function to obtain the state to be analyzed from the user.
- **`menu()`**: a function to print the options given by the strings in the list OPTIONS defined at the top of the file main.
- **`select_option()`**: a function to obtain a selection from the user from options provided in the list `OPTIONS`.
- **`main()`**: the function that runs the analysis

The first two functions are provided. You will complete the last two. See the following text for details on each function.

GIVEN - the function `select_a_state`

The function `select_a_state()` repeatedly asks the user for a state until the user supplies a state given in the list `STATES`. The given state is then returned as a string. The function prints an error message if the user does not enter a state in the list `STATES`. Here is an example of how the function could work:

```
>select_a_state()
Select a state: Texas
You entered a state without hospitalization data or an incorrect
spelling
Select a state: Nw Jerzee
You entered a state without hospitalization data or an incorrect
spelling
Select a state: New Jersey
'New Jersey'
```

GIVEN - the function `menu`

The function `menu()` prints the number that the user should use to select an option (1, 2, 3, 4, or 5) along with the option from the list `OPTIONS`. Notice that the index of the option in the list `OPTIONS` is one off from the number the user must enter to select that option. For example, the option "Quit" can be accessed from the list using an index of 4, but the user needs to select 5 to quit. The following example shows the numbers for selection and the corresponding options.

```
>menu()
************************************************************
Options
1 Print hospitalization data
2 Print heat index data
3 Plot data as a function of year
4 Plot hospitalization data as a function of heat index data
5 Quit
************************************************************
```

TO DO - Complete the function `select_option`

Complete the function `select_option()` to print a menu of options and obtain a selection from the user. (Remember that you have the helper function `menu()`.) If the user enters an invalid selection:

- An error message should be displayed.
- The menu of options should be displayed.
- The user should be prompted to enter a new selection.

The above should be repeated until the user enters a valid selection.
The valid selection is then returned as an integer.
This function will not be autograded so you can use a different format or error message than the following example.

```
>select_option()
************************************************************
Options
1 Print hospitalization data
2 Print heat index data
3 Plot data as a function of year
4 Plot hospitalization data as a function of heat index data
5 Quit
************************************************************
Enter a number from the menu: cat
Invalid selection. You must enter a number from 1 to 5.
```

```
************************************************************
Options
1 Print hospitalization data
2 Print heat index data
3 Plot data as a function of year
4 Plot hospitalization data as a function of heat index data
5 Quit
************************************************************
Enter a number from the menu: 8
Invalid selection. You must enter a number from 1 to 5.
************************************************************
Options
1 Print hospitalization data
2 Print heat index data
3 Plot data as a function of year
4 Plot hospitalization data as a function of heat index data
5 Quit
************************************************************
Enter a number from the menu: 2
2
```

TO DO - Complete the function `main`

Wow! You have completed all of the functions needed to run your analysis! Your last function to complete for Part 1 is the `main` function. This function allows the user to select a state and then repeatedly asks the user to select what aspect of the data to see (print or plot options) or to quit the program. (The code will not ask the user for a new state. To switch states, the user will need to run the program anew.) Here is an outline of what you need to do:

- Replace how the variable `state` is defined by using a function call to `select_a_state()` to allow the user to select the state.
- Keep the lines assigning the filenames.
- Keep the lines creating an instances of the classes:
    - hosp as an instance of the Hospitalizations class
    - heat as an instance of the HeatIndex class

- Replace selection = 5 with a call to the `select_option()` function.
- Add code so the user will be able to repeatedly select from the options until the user wishes to quit.

Skeleton code is provided for this function. Add or rearrange code where needed.

FINAL PROJECT PART 1 SUBMISSION

Here is a list of functions that you need to write:

- `read_data_file`
- `Data class method: group_yearly_nums`
- `Hospitalizations class method: __repr__`
- `Hospitalizations class method: make_yearly_sum`
- `HeatIndex class method:__init__`
- `HeatIndex class method:__repr__`
- `HeatIndex class method: make_yearly_av`
- `get_years_nums_from_d`
- `get_hosp_heat_from_d`
- `select_option`
- `main`