Assignment 3: Hypertension and Low Income in Toronto Neighbourhoods

Goals of this Assignment

In this assignment, you will practise working with files, building and using dictionaries, designing functions using the Function Design Recipe, reading documentation, and writing unit tests.

Files to Download

Download A3.zip which contains starter code (`a3.py` and `test_a3.py`), the checker (`a3_checker.py`), its configuration files (`checker_generic.py` and `a3_pythonta.json`), and two sizes of each type of data file.

Background

A commonly-held belief is that an individual's health is largely influenced by the choices they make. However, there is lots of evidence that health is affected by systemic factors.

Health researchers often study the relationships between an individual's health outcomes and factors related to their physical environment, social and economic situations, and geographic location. Studies such as this one investigate how a particular health outcome (living with hypertension) are tied to a systemic factor (the income level of a country).

In this assignment, you will write code to assist with analyzing data on the relationship between hypertension (also known as high blood pressure) and income levels in Toronto neighbourhoods. The data you will work with is real data, however we have simplified it somewhat to make this assignment clearer for you.

A note on math and stats

The data analysis that your code will do will include some statistical analysis that we have not talked about in the course. You do NOT need to understand the underlying statistics to complete this assignment. The code you write will do some simple mathematical operations, like adding up some numbers, or finding ratios using division. We will use Pearson correlation for the more advanced analysis and you will use existing functions that we have imported for you. You will need to take a look at the examples of these functions in order to figure out what arguments you need to pass to them, and what types of data they return, but you do not need to understand how they work in any detail.

Correlation is a single coefficient expressing the tendency of one set of data to grow linearly, in the same or opposite direction, with another set of data. This is done by comparing whether points that have been paired between the two sets are similarly greater or less than their set's respective averages. For example, if we wanted to compare whether for students in the class, age is correlated with height, we would have two sets of data, birth date (which we could express as, say, number of weeks old for finer granularity), and heights. Numbers from each set are ordered in the same way so that each height value corresponds to the age value for the same student. What is nice about the correlation metric we are using, is that it is normalized to be between -1 and 1, with these values giving us a nice human interpretation. A value of 1 means that the points make a straight line. In our example, this means, for some increase in age, we have a consistent increase in height. Similarly, a value of -1 is the same relationship but with a flip of direction, where older students would be shorter than younger ones. Finally, a value of 0 would say that there is no consistent increase or decrease in height for a change in age. We will use this to investigate the relationship between low income rates and hypertension, for any tendency to increase or decrease together.

If you are a statistics person, keep in mind that the learning goals of the assignment are about writing code using what we've learned in the course, not about doing a proper statistical analysis :)

Dataset descriptions

This assignment uses data files related to one of the two variables of interest (i.e. hypertension data or income data). The files are CSV (comma separated values) files, where each column in a line is separated by a comma. You can assume there are no commas anywhere else in the files, other than to separate columns, and that any file given is in the correct format. The two file types are described below.

Neighbourhood hypertension data files

The first row in a neighbourhood hypertension file contain header information, and the remaining rows each contain data relating to hypertension prevalence in a particular Toronto neighbourhood.

Here is a description of the different columns of the dataset:

Column index	Description
0	An ID that uniquely identifies each neighbourhood.
1	The name of the neighbourhood. Neighbourhood names are unique.
2	The number of people aged 20 to 44 with hypertension in the neighbourhood.
3	The total number of people aged 20 to 44 in the neighbourhood.
4	The number of people aged 45 to 64 with hypertension in the neighbourhood.
5	The total number of people aged 45 to 64 in the neighbourhood.
6	The number of people aged 65 and older with hypertension in the neighbourhood.
7	The total number of people aged 65 and older in the neighbourhood.

Neighbourhood income data files

The first row in a neighbourhood income data file contains header information, and the remaining rows each contain data about low income status.

Here is a description of the different columns of the dataset:

Column index	Description
0	An ID that uniquely identifies each neighbourhood.
1	The name of the neighbourhood. Neighbourhood names are unique.
2	The total population in the neighbourhood.
3	The number of people in the neighbourhood with low income status.

Neighbourhood names and ids are the same between our hypertension data files and our low income data files. However, the total population of a neighbourhood can be different between the two data files, as they were collected at different times.

The `CityData` Type

The code you will write for this assignment will build and then use a dictionary that contains hypertension and low income data about neighbourhoods in a city. This section describes the format of that dictionary.

Key/value pairs in a `CityData` dictionary

Each key in a `CityData` dictionary is a string representing the name of a neighbourhood. As is necessary for dictionary keys, all neighbourhood names will be unique.

The values in a `CityData` dictionary are dictionaries containing information about a neighbourhood. These inner dictionaries contain specific keys that label a neighbourhood's data.

Format of the inner dictionaries

A dictionary that is a value in a dictionary of type `CityData` has the following key/value pairs:

Key	(Type) Value
`'id'`	(`int`) The id number of this neighbhourhood.
`'total'`	(`int`) The total population of this neighbourhood, as given in the low income data file.
`'low_income'`	(`int`) The number of people in this neighbourhood who are classfied as low income.
`'hypertension'`	(`list[int]`) A list of the hypertension data of this neighbourhood. This list will have length exactly 6, and the values will be the numbers from columns 2 through 7 as described in the section above on neighbourhood hypertension data files.

An example `CityData` dictionary

The following is an example of a `CityData` dictionary. We have also provided this dictionary for you to use in your docstring examples and other testing in the starter code file. Note that we have formatted the dictionary below for easier reading, however you will not see this formatting in your code.

```
{
    "West Humber-Clairville": {
        "id": 1,
        "hypertension": [703, 13291, 3741, 9663, 3959, 5176],
        "total": 33230,
        "low_income": 5950,
    },
    "Mount Olive-Silverstone-Jamestown": {
        "id": 2,
        "hypertension": [789, 12906, 3578, 8815, 2927, 3902],
        "total": 32940,
        "low_income": 9690,
    },
    "Thistletown-Beaumond Heights": {
        "id": 3,
        "hypertension": [220, 3631, 1047, 2829, 1349, 1767],
        "total": 10365,
        "low_income": 2005,
    },
    "Rexdale-Kipling": {
        "id": 4,
        "hypertension": [201, 3669, 1134, 3229, 1393, 1854],
        "total": 10540,
        "low_income": 2140,
    },
    "Elms-Old Rexdale": {
        "id": 5,
        "hypertension": [176, 3353, 1040, 2842, 948, 1322],
        "total": 9460,
        "low_income": 2315,
    },
}
```

The sample `CityData` dictionary above represents hypertension and low income data for five neighbourhoods: West Humber-Clairville, Mount Olive-Silverstone-Jamestown, Thistletown-Beaumond Heights, Rexdale-Kipling, and Elms-Old Rexdale.

Let's take a closer look at the data for Elms-Old Rexdale. This neighbourhood is represented by the key/value pair where the key is `'Elms-Old Rexdale'`. The id of this neighbourhood is 5. The hypertension data for this neighbourhood is as follows: 3353 people are between the ages of 20 and 44, 176 of whom have hypertension. There are 2842 people between the ages of 45 and 64, 1040 of whom have hypertension, and there are 1322 people aged 65 and up, 948 of whom have hypertension. The low income data for this neighbourhood is that 2315 people are classified as low income, from a total population of 9460 people.

Note that the totals do not match between the low income and the hypertension data – this is because the low income data was collected before the hypertension data, and the size of the neighbourhoods changed. For the purposes of this assignment, we will assume the collection of these two datasets is close enough in time to compare them to each other. You do not need to do anything about these differing totals, other than to make sure you are using the correct total when computing rates, as described later.

Age standardization

This section describes the process of age standardization that we will use in this assignment to perform a more accurate analysis. **Note that we have given you a function that computes the age standardized rate from the raw rate (described in Task 3). This section is for your information only; we have already implemented this for you.**

Our dataset will let us calculate the rate of hypertension in each Toronto neighbourhood. One complicating factor is that different neighbourhoods have different age demographics. For example, the Henry Farm neighbourhood has a significantly lower proportion of 65+ residents than Hillcrest Village. And because people aged 65+ have a higher overall rate of hypertension, this demographic different alone would cause us to expect to see a difference in the overall hypertension between these neighbourhoods.

So because we care about the impact of low income status and hypertension rates, we want to remove the impact of different age demographics between the neighbourhoods. To do so, we will use a process called **age standardization** to calculate an adjusted hypertension rate that ignores differences in ages. This process involves the following steps for each neighbourhood:

1. First,we'll calculate the hypertension rate within each of the following age groups: 20-44, 45-64, and 65+. We'll report these rates as percentages, which you can think of as being "X out cases of hypertension per 100 people aged 20-44".
2. Then, we'll pick one standard population with certain numbers of people in these age groups. For the purpose of this assignment, we'll use the total Canadian population from the 1991 census:

Age Group	Population
20-44	11,199,830
45-64	5,365,865
65+	3,169,970
Total (20+)	19,735,665

3. Then, we'll use the neighbourhood rates to calculate the hypothetical number of people in the standard population who would have hypertension. For example, if the rates for neighbourhood X were 20% of 20-44, 30% of 45-64, and 66% of 65+, the total number of people with hypertension in the standard population would be ...
4. Finally, divide this number of people with hypertension by the total size of the standard population, yielding a final percentage. This percentage is the **age standardized rate** for the neighbourhood.

If you are interested, you can read more about age standardized rates here.

Required Functions

In the starter code file `a3.py`, follow the Function Design Recipe to complete the functions described below.

You will need helper functions (i.e. functions you define yourself to be called in other functions) for some of the required functions, but likely not for all of them. Helper functions also require complete docstrings. We strongly recommend you also follow any suggestions about helper functions in the table below; we give you these hints to make your programming task easier.

Some indicators that you should consider writing a new helper function, or using something you've already written as a helper are:

- Rewriting code to solve a task you have already solved in another function
- Getting a warning from the checker that your function is too long
- Getting a warning from the checker that your function has too many nested blocks or too many branches
- Realizing that your function can be broken down into smaller sub-problems (with a helper function for each)

For each of the functions below, other than the file reading functions in Task 1, write at least two examples in the docstring. You can use the provided `SAMPLE_DATA` dictionary, and you should also create another small `CityData` dictionary for examples and testing. If your helper function takes an open file as an argument, you do NOT need to write any examples in that function's docstring. Otherwise, for any helper functions you add, write at least two examples in the docstring.

Your functions should not mutate their arguments, unless the description says that is what they do.

Assumptions

Assume the following about the data:

- All neighbourhood ids and names are unique, and will appear the same in all data files. That is, no neighbourhood will have a different id between files, or a different name.
- In all tasks except Task 1, the dictionary parameter will have both hypertension and low income data for every neighbourhood. That is, it will be a valid `CityData` dictionary.
- All float values should be left as is; do not round any of them.

Using Constants

The starter code contains constants that you should use in your solution for the list indices and key identifiers for the CityData dictionary as well as the column numbers for the input files. You may add other constants if you wish.

Task 1: Building the data dictionary

In this task, you will write functions that read in files and build the dictionary of neighbourhood data. You will write two functions – one that adds hypertension data to a dictionary, and one that adds low income data. You will almost certainly also need to define one or more helper functions to help you solve this task.

These functions will be used to build a `CityData` dictionary, however the dictionary that is passed to the functions may not yet contain all of the data.

To illustrate this, we have provided two small data files. After passing the same dictionary to both functions with each of those small files, the dictionary should be a `CityData` dictionary that contains the same information as the provided `SAMPLE_DATA` dictionary. Using the small hypertension file and an empty dictionary as arguments to `get_hypertension_data`, the result should be that the dictionary now contains the hypertension data as in `SAMPLE_DATA`, but not the low income data.

```
{
    "West Humber-Clairville": {
        "id": 1,
        "hypertension": [703, 13291, 3741, 9663, 3959, 5176],
    },
    "Mount Olive-Silverstone-Jamestown": {
        "id": 2,
        "hypertension": [789, 12906, 3578, 8815, 2927, 3902],
    },
    "Thistletown-Beaumond Heights": {
        "id": 3,
        "hypertension": [220, 3631, 1047, 2829, 1349, 1767],
    },
    "Rexdale-Kipling": {
        "id": 4,
        "hypertension": [201, 3669, 1134, 3229, 1393, 1854],
    },
    "Elms-Old Rexdale": {
        "id": 5,
        "hypertension": [176, 3353, 1040, 2842, 948, 1322],
    },
}
```

Similarly, using the small low income file and an empty dictionary as arguments to `get_low_income_data`, the result should be that the dictionary now contains the low income data as in `SAMPLE_DATA`, but not the hypertension data.

```
{
    "West Humber-Clairville": {"id": 1, "total": 33230, "low_income": 5950},
    "Mount Olive-Silverstone-Jamestown": {"id": 2, "total": 32940, "low_income": 9690},
    "Thistletown-Beaumond Heights": {"id": 3, "total": 10365, "low_income": 2005},
    "Rexdale-Kipling": {"id": 4, "total": 10540, "low_income": 2140},
    "Elms-Old Rexdale": {"id": 5, "total": 9460, "low_income": 2315},
}
```

A complete `CityData` dictionary will have been passed to both functions. See the sample usage at the end of the starter code file for an example of how both functions are used to build a `CityData` dictionary.

Note: While this is the first task, it is not necessarily the easiest. If you are stuck while working on this task, we suggest moving on to other tasks and coming back to this later.

Recall that `TextIO` as the parameter type means the file is already open.

Function name: (Parameter types) -> Return type	Full Description (paraphrase to get a proper docstring description)
`get_hypertension_data`: `(dict, TextIO) -> None`	The first parameter is a dictionary representing hypertension and/or low income data for a neighbourhood and the second parameter is a hypertension data file that is open for reading. This function should modify the dictionary so that it contains the hypertension data in the file. If a neighbourhood with data in the file is already in the dictionary then its hypertension data should be updated. Otherwise it should be added to the dictionary with its hypertension data. After this function is called, the dictionary should contain key/value pairs whose keys are the names of every neighbourhood in the hypertension data file, and whose values are dictionaries which contain at least the keys `'id'` and `'hypertension'` for those neighbourhoods.
`get_low_income_data`: `(dict, TextIO) -> None`	The first parameter is a dictionary representing hypertension and/or low income data for a neighbourhood and the second parameter is a low income data file that is open for reading. This function should modify the dictionary so that it contains the low income data in the file. If a neighbourhood with data in the file is already in the dictionary then its low income data should be updated. Otherwise it should be added to the dictionary with its low income data. After this function is called, the dictionary should contain key/value pairs whose keys are the names of every neighbourhood in the low income data file, and whose values are dictionaries which contain at least the keys `'id'`, `'total'`, and `'low_income'` for those neighbourhoods.

Task 2: Neighbourhood-level Analysis

Function name: (Parameter types) -> Return type	Full Description (paraphrase to get a proper docstring description)
`get_bigger_neighbourhood`: `(CityData, str, str)` `-> str`	The first parameter is a `CityData` dictionary, and the second and third parameters are strings representing the names of neighbourhoods. The function returns the name of the neighbourhood that has a higher population, according to the low income data. Assume that the two neighbourhood names are different. If a name is not in the dictionary, assume it has a population of 0. If the two neighbourhoods are the same size, return the first name (i.e. the leftmost one in the parameters list, *not* alphabetically).
`get_high_hypertension_rate`: `(CityData, float)` `->` `list[tuple[str, float]]`	The first parameter is a `CityData` dictionary, and the second parameter is a number between 0.0 and 1.0 (inclusive) that represents a threshold. This function should return a list of tuples representing all neighbourhoods with a hypertension rate greater than or equal to the threshold. In each tuple, the first item is the neighbourhood name and the second item is the hypertension rate in that neighbourhood. Compute the overall hypertension rate for a neighbourhood by dividing the total number of people with hypertension by the total number of adults in the neighbourhood. If this function was called with the provided `SAMPLE_DATA` dictionary and a threshold of `0.3` as arguments, then the returned value would be `[('Thistletown-Beaumond Heights', 0.31797739151574084), ('Rexdale-Kipling', 0.3117001828153565)]` The order of the tuples in the list does not matter.
`get_ht_to_low_income_ratios`: `(CityData) -> dict[str, float]`	The parameter is a `CityData` dictionary. This function should return a dictionary where the keys are the same as in the parameter, and the values are the ratio of the hypertension rate to the low income rate for that neighbourhood. For the denominators for each rate, use the total number of people as given in the corresponding data file. That is, for calculating the low income rate, use the total population in the neighbourhood from the low-income data file; and for the hypertension rate, use the sum of the total people in all three age groups in the hypertension data. For example, if this function was called with the provided `SAMPLE_DATA` dictionary as an argument, then the returned dictionary should include the key/value pairs `'West Humber-Clairville': 1.6683148168616895` and `'Mount Olive-Silverstone-Jamestown': 0.9676885451091314`, as well as the pairs for the other three neighbourhoods. You may find that writing a helper function would be useful here.

`calculate_ht_rates_by_age_group` : `(CityData, str) -> tuple[float, float, float]`	The first parameter is a `CityData` dictionary, and the second parameter represents a neighbourhood name that is a key in the dictionary. The function returns a tuple of three values, representing the hypertension rate for each of the three age groups in the neighbourhood as a percentage. (Note that this is different from the previous two functions, where the rate was calculated using the total numbers, and was *not* expressed as a percentage.) For example, consider the neighbourhood with the name `'Elms-Old Rexdale'` in the provided `SAMPLE_DATA` dictionary. The rate of hypertension for the 20-44 age group in the neighbourhood is computed by dividing the number of people aged 20-44 with hypertension by the total number of people aged 20-44. For this neighbourhood, that is 176 / 3353. To get this rate as a percentage, we then multiply by 100, for a rate of 5.24903071875932. The rate is calculated in the same way for the 45-64 and 65+ age groups. Thus, calling this function with the provided `SAMPLE_DATA` dictionary and the string `'Elms-Old Rexdale'` should return `(5.24903071875932, 36.593947923997185, 71.70953101361573)`. Notice that this function is used as a helper in the `get_age_standardized_ht_rate` function that we have provided for you.

Task 3: Finding the Correlation

Function name: (Parameter types) -> Return type	Full Description (paraphrase to get a proper docstring description)
`get_stats_summary` : `(CityData) -> float`	The parameter for this function is a `CityData` dictionary. This function returns the correlation between age standardized hypertension rates and low income rates across all neighbourhoods. To complete this function, you will need to use the `correlation` function in the module `statistics`. Refer to the documentation for the function to determine what arguments to pass to the `correlation` function, and how to use its returned value. You can find the documentation online here or using `help(statistics.correlation)`. Remember that to call help on a function from another module, you first need to import the module. You will need to use the provided function `get_age_standardized_ht_rate` as a helper to get the age-standardized rate for each neighbourhood. If this function was called with the provided `SAMPLE_DATA` dictionary, then the returned value would be `0.28509539188554994`.

Task 4: Order by Ratio

Function name: (Parameter types) -> Return type	Full Description (paraphrase to get a proper docstring description)
`order_by_ht_rate`: `(CityData) -> list[str]`	The parameter is a `CityData` dictionary. This function will return a list of the names of the neighbourhoods, ordered from lowest to highest age-standardized hypertension rate. We use the age-standardized rate because we are comparing across neighbourhoods. Assume every neighbourhood has a unique hypertension rate; i.e. that there are no ties. For example, if this function is called with the `CityData` dictionary provided in the starter code, it will return `['Elms-Old Rexdale', 'Rexdale-Kipling', 'Thistletown-Beaumond Heights', 'West Humber-Clairville', 'Mount Olive-Silverstone-Jamestown']` There are multiple ways to solve this problem. You may choose to solve this problem by writing your own sorting code, but you do not have to do this. You can also use `list.sort` as part of your solution, if you choose.

Task 5: Required Testing (`pytest`)

Write and submit a set of pytests for the `get_bigger_neighbourhood` function. We have provided starter code in the `test_a3.py` file. We have included one test that you can use as a template to write your other test functions. For each test function, include a brief docstring description specifying what is being tested. Do not write examples in the docstrings. Your set of tests should all pass on correct code, and your tests should be thorough enough that at least one of them will fail on a buggy version of the function. There is no required number of tests; we will mark your tests by running them on the correct code as well as several buggy versions.

Marking

These are the aspects of your work that will be marked for Assignment 3:

- **Correctness (70%)**: Your functions should perform as specified. Correctness, as measured by our tests, will count for the largest single portion of your marks. Once your assignment is submitted, we will run additional tests, not provided in the checker. Passing the checker does not mean that your code will earn full marks for correctness.
- **Testing (10%)**: Your test suite (task 5) will be checked by running in on incorrect/broken implementations. Your tests should all pass on a correct version of the function, and at least one should fail on each of our broken implementations.
- **Coding style (20%)**: Make sure that you follow Python style guidelines that we have introduced and the Python coding conventions that we have been using throughout the semester. Although we don't provide an exhaustive list of style rules, the checker tests for style are complete, so if your code passes the checker, then it will earn full marks for coding style with one exception: docstrings will be evaluated separately. For each occurrence of a PyTA error, a 1 mark (out of 20) deduction will be applied. For example, if a C0301 (line-too-long) error occurs 3 times, then 3 marks will be deducted.