COMP202 Assignment 4

Due: Dec 4 at 23:59

This is an individual assignment.

Flu Pandemic!

It's the near-future, and Montreal is having a flu pandemic (oh no!). It's not going well and somehow now you're the one in charge of getting data on the pandemic to the epidemiologists.

You've been given one large file of raw data about the early days of the pandemic. But the data was recorded by dozens of different people and systems. In the chaos of the pandemic, data was not recorded in consistent ways. Some lines are recorded in French. Others in English. Some lines separate the information with tabs, others with commas.

"Data cleaning" refers to the process of taking raw data and processing it into a state that can be used for empirical analysis. Your task in this assignment is to clean the raw data file you've been given.

Instructions

It is very important that you follow the directions as closely as possible. The directions, while perhaps tedious, are designed to make it as easy as possible for the TAs to mark the assignments by letting them run your assignment, in some cases through automated tests. While these tests will never be used to determine your entire grade, they speed up the process significantly, which allows the TAs to provide better feedback and not waste time on administrative details. Plus, if the TA is in a good mood while he or she is grading, then that increases the chance of them giving out partial marks. :)

Up to 30% can be removed for bad indentation of your code as well as omitting comments, or poor coding structure.

To get full marks, you must:

- Follow all directions below
  - In particular, make sure that all function and variable names are **spelled exactly** as described in this document. Else a 50% penalty will be applied.
- Make sure that your code runs.
  - Code with errors will receive a very low mark.
- Write your name and student ID as a comment in all .py files you hand in
- Name your variables and helper functions appropriately
  - The purpose of each variable should be obvious from the name
- Comment your work
  - A comment every line is not needed, but there should be enough comments to fully understand your program

Errata and Frequently Asked Questions

On MyCourses we have a discussion forum titled Assignment 4. **A thread will be pinned in the forum with any errata and frequently asked questions. If you are stuck on the assignment, start by checking that thread.**

We strongly encourage starting the assignment early. For example, office hours the week before the deadline are not very crowded, but office hours close to the deadline will be.

What To Submit

Please put all your files in a folder called Assignment4. Zip the folder (DO NOT RAR it) and submit it in MyCourses. If you do not know how to zip files, please ask any search engine or friends. Google will be your best friend with this, and a lot of different little problems as well.

Inside your zipped folder, there must be the following files. **Do not submit any other files.** Any deviation from these requirements may lead to lost marks.

1. `initial_clean.py`
2. `time_series.py`
3. `time_series.png`
4. `construct_patients.py`
5. `fatality_by_age.png`
6. `README.txt` In this file, you can tell the TA about any issues you ran into doing this assignment. If you point out an error that you know occurs in your program, it may lead the TA to give you more partial credit.

   This file is also where you should make note of anybody you talked to about the assignment. Remember this is an individual assignment, but you can talk to other students using the **Gilligan's Island Rule**: you can't take any notes/writing/code out of the discussion, and afterwards you must do something inane like watch television for at least 30 minutes.

   If you didn't talk to anybody nor have anything you want to tell the TA, just say "nothing to report" in the file.

Style

There are 70 marks for completing functions in this assignment.
There are 30 marks for the style of your code.

Some tips:

- Call helper functions rather than copy and paste (or reinvent) code
- Create helper functions where appropriate
- Use descriptive variable names
- Lines of code should NOT require the TA to scroll horizontally to read the whole thing
- Add blank lines between "chunks" of code to improve readibility

Your Data File

Every student in the class has a unique file to clean. To download yours, go to `https://www.cs.mcgill.ca/~patitsas/comp202/files/YOURSTUDENTNUMBER.txt`.

Important: at this url, save the file. To do so, right click and use "save page as". Do not copy/paste the file contents because your browser could have messed up the accents in the file.

Suggested: the file is rather large. We recommend starting with only the first 5-10 lines of your file, then adding in more lines as you've tested your code. A 15-line version of your file can be found at: `https://www.cs.mcgill.ca/~patitsas/comp202/files/YOURSTUDENTNUMBER-short.txt`

About The Data

Each line of your raw file contains the following information.

1. A number representing who recorded the data
2. A number representing the patient — each patient has a unique number. The first patient diagnosed with the flu has the patient number 0, the second patient has the patient number 1, etc. There could be multiple rows for the same patient — for example they could be diagnosed on one day, and then die on another day.
3. The date this entry was made
4. The patient's date of birth
5. The patient's sex/gender (some have sex recorded, some gender)
6. The patient's home postal code
7. The patient's state: Infected / Recovered / Dead (at the time the entry was made)
8. The patient's temperature at the time the entry was made
9. How many days the patient has been symptomatic

Here's what it could like if patient #21 is recorded as infected for three days, and then recovers (and so the number of days symptomatic does not increase in the last entry):

```
6 21 2022/11/28 1980/2/14 X          H1Z I   40    5
2 21 2022.11.29 1980.2.14 non-binary H1Z inf 41.3C 6
6 21 2022/11/30 1980/2/14 X          H1Z I   39    7
1 21 2022-12-01 1980-2-14 genderqueer    H1Z Recovered 37.2 7
```

Safe Assumptions

You can assume:

1. The columns will always appear in the same order
2. Every column will be present
3. Each row is in chronological order
4. There are no spelling mistakes
5. Each recorder records data in a consistent way
6. All dates are recorded in ISO format: year-month-day, where the year is four digits, and the month is the number (e.g. 2019-11-30), but could be delimited with any of '.', '/' or '-'.
7. Each unique patient will have an entry in the file for every day when they're infected, up to the last day of the file.
8. If a patient dies or recovers, the entry that notes they died/recovered will be the last time the patient appears in the file

1 Initial Clean [16 points]

Create a new module `initial_clean.py` and put your name and student ID at the top. All of the code for this section will go into this module. You may not import any modules.

1.1 Which Delimiter [5 points]

Create, document and test the function, **which_delimiter**:

- Input: one string
- A **delimiter** is the name for a string that used to separate columns of data on a single line
- Returns: the most commonly used delimiter in the input string; will be one of space/comma/tab
- Example:

```
>>> which_delimiter('0 1 2,3')
' '
```

- Raise a AssertionError exception if there is no space/comma/tab (*Note:* don't worry that we have not seen AssertionError in class! We are deliberately using a different kind of error than TypeError/ValueError/etc so the autograder can tell the difference between your raised exceptions and any issues in your code.)
- You can assume that you do not have to deal with ties

1.2 Stage 1: Delimiting and Capitals [6 points]

Create, document and test the function, **stage_one**:

- Two inputs: input_filename and output_filename
- This will open the file with the name input_filename, and read the file line by line
- We will be making changes to each line and then writing the new version of the line to a new file named `output_filename`
- Because there is French in the files we need to add `encoding = 'utf-8'` as a parameter to all calls to `open`, so we can support the accents. This looks like:

```
out_file = open(out_filename, 'w', encoding = 'utf-8')
```

- The changes to make to the data:
  1. Change the most common delimiter to tab (if it is not already tab-delimited)
  2. Change all text to be upper case
  3. Change any / or . in the dates to hyphens (e.g. 2022/11/28 becomes 2022-11-28)
- Return an integer: how many lines were written to `output_filename`

```
>>> stage_one('1111111.txt', 'stage1.tsv')
3000
```

- Why do I use .tsv now instead of .txt? The data is now all tab separated!
- See next page for example of how the data changes

- Example: if we start with data that looks like:

  ```
  6 0 2022/11/28 1980/2/14 F H3Z I 40 3
  7   1   2022.11.29  1949.8.24   HOMME   H1M2B5  INF   40C   4
  10 0 2022/11/29 1980/2/14 femme h3z3l2 infectée 39,13 C 4
  11,2,2022.11.29,1982.1.24,femme,h3x1r7,morte,39,3 C,3
  ```

  After stage_one, the start of our output file should look like:

  ```
  6   0 2022-11-28 1980-2-14 F       H3Z     I         40      3
  7   1 2022-11-29 1949-8-24 HOMME   H1M2B5  INF       40C     4
  10  0 2022-11-29 1980-2-14 FEMME   H3Z3L2  INFECTÉE  39,13   C      4
  11  2 2022-11-29 1982-1-24 FEMME   H3X1R7  MORTE     39      3 C    3
  ```

1.3 Stage 2: Consistent Columns [5 points]

Create, document and test the function, **stage_two**:

- Two inputs: input_filename and output_filename
- This will open the file with the name `input_filename`, and read the file line by line
- Like in Stage 1, we will be making changes to each line and then writing the new version of the line to a new file named `output_filename`
- Because there is French in the files we need to add `encoding = 'utf-8'` as a parameter to all calls to `open`, so we can support the accents. This looks like:

  ```
      out_file = open(out_filename, 'w', encoding = 'utf-8')
  ```

- The changes to make to the data:

  1. All lines should have 9 columns
  2. Any lines with more than 9 columns should be cleaned so the line is now 9 columns. For example, in French the comma is used for decimal points, so the temperature '39,2' could have been broken into 39 and 2.

- Example: if our input file is the output file from Stage 1's example, we now have:

  ```
  6   0 2022-11-28 1980-2-14 F       H3Z     I         40        3
  7   1 2022-11-29 1949-8-24 HOMME   H1M2B5  INF       40C       4
  10  0 2022-11-29 1980-2-14 FEMME   H3Z3L2  INFECTÉE  39,13C    4
  11  2 2022-11-29 1982-1-24 FEMME   H3X1R7  MORTE     39.3C     3
  ```

- Return an integer: how many lines were written to `output_filename`

  ```
      >>> stage_two('stage1.tsv', 'stage2.tsv')
      3000
  ```

2 Pandemic Over Time [18 points]

Create a new module time_series.py and put your name and student ID at the top. All of the code for this section will go into this module.

You may import the Python modules datetime, numpy and matplotlib, including their submodules (e.g. pyplot)

2.1 Date Diff [5 points]

Create, document and test the function, **date_diff**:

- Input: two strings representing dates in ISO format (eg. 2019-11-29)
- Returns: how many days apart the two dates are, as an integer
- If the first date is earlier than the second date, the number should be positive; otherwise the number should be negative
- Example:

```
>>> date_diff('2019-10-31', '2019-11-2')
2
```

- *Tip:* Python offers a module called datetime that can you help you with this. Since we have not covered this module in class, here are some important things to know about it:
  - You can create date objects. Here are a few examples:

```
import datetime
date1 = datetime.date(2019, 10, 31) # Year, month, day
print(date1.year)                   # will be 2019
date2 = datetime.date(2019, 11, 2)
print(date2.month)                  # will be 11
diff = date1 - date2
```

  - You can subtract two date objects. The result is a timedelta object, which has one attribute: days. This is how many days apart the two dates are.
- You can read more here: https://docs.python.org/3/library/datetime.html

2.2 Get Age [3 points]

Create, document and test the function, **get_age**:

- Input: two strings representing dates in ISO format (eg. 2019-11-29)
- Returns: how many complete years apart the two dates are, as an integer
- Assume one year is 365.2425 days
- If the first date is earlier than the second date, the number should be positive; otherwise the number should be negative
- Examples:

```
>>> get_age('2018-10-31', '2019-11-2')
1
>>> get_age('2018-10-31', '2000-11-2')
-17
```

2.3 Stage Three [5 points]

Create, document and test the function, **stage_three**:

- Two inputs: input_filename and output_filename
- This will open the file with the name `input_filename`, and read the file line by line. Remember we want utf-8 encoding like previous stages:

  ```
  out_file = open(out_filename, ’w’, encoding = ’utf-8’)
  ```

- We will be making changes to each line and then writing the new version of the line to a new file named `output_filename`
- First, determine the **index date**: the first date in the first line of the file (2022-11-28 in our running example)
- The changes to make to the data:

  1. Replace the date of each record with the date_diff of that date and the **index date**
  2. Replace the date of birth with age at the time of the **index date**
  3. Replace the status with one of I, R and D. (Representing Infected, Recovered, and Dead; the French words are infecté(e), récupéré(e) and mort(e).)

- Example: if our input file is the output file from Stage 2's example, we now have:

```
6     0    0    42    F      H3Z        I    40        3
7     1    1    73    HOMME  H1M2B5     I    40C       4
10    0    1    42    FEMME  H3Z3L2     I    39,13 C   4
11    2    1    40    FEMME  H3X1R7     D    39 C      3
```

- Return: a dictionary. The keys are each day of the pandemic (integer). The values are a dictionary, with how many people are in each state on that day. Example:

```
>>> stage_three(’stage2.tsv’, ’stage3.tsv’)
{0: {’I’: 1, ’D’: 0, ’R’: 0},  1: {’I’: 2, ’D’: 1, ’R’: 0}}
```

2.4 Plot Time Series [5 points]

Create, document and test the function, **plot_time_series**:

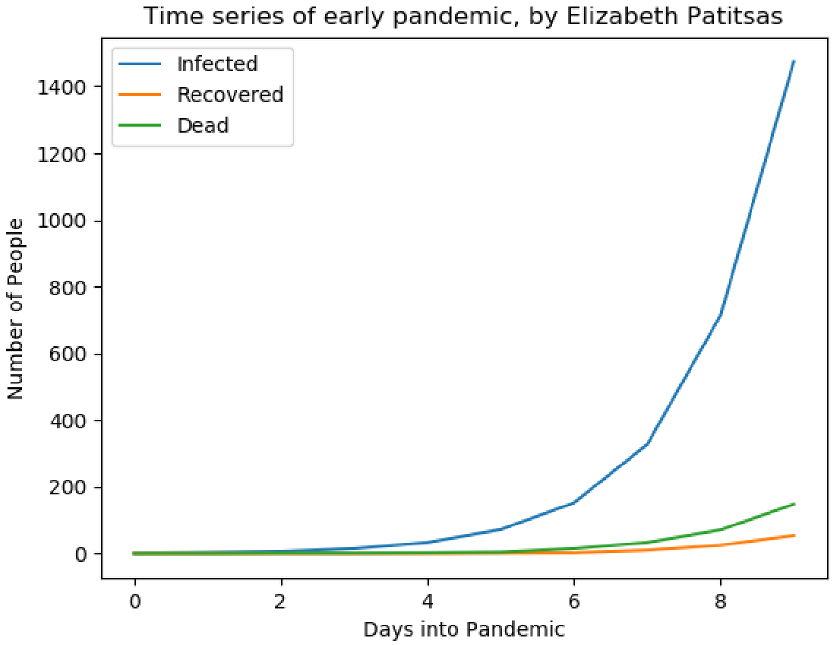
- Input: a dictionary of dictionaries, formatted as the return value of Stage Three
- Return: a list of lists, where each sublist represents each day of the pandemic. Each sublist [how many people infected, how many people recovered, how many people dead]

```
>>> d = stage_three('stage2.tsv', 'stage3.tsv')
>>> plot_time_series(d)
[[1, 0, 0], [2, 0, 1]]
```

- In the function, also plot that list with matplotlib's plot function, and save the png as `time_series.png`
  - Set the xlabel as 'Days into Pandemic'
  - Set the ylabel as 'Number of People'
  - Create a legend with Infected, Recovered and Dead. You can do this with:

    ```
    plt.legend(['Infected', 'Recovered', 'Dead'])
    ```

  - Title the plot 'Time series of early pandemic, by ' and then append your name
  - Save the file as `time_series.png`
- You should get a plot with three increasing lines; the slopes will vary from person to person, and could look like:



3 Patients [34 points]

Create a new module `construct_patients.py` and put your name and student ID at the top. All of the code for this section will go into this module.

You may import `datetime`, `numpy` and `matplotlib`, including sub-modules (e.g. `pyplot`)

3.1 Patient Class

Create, document and test the class `Patient`. Its methods are:

1. __init__ [15 points]
    - Input (all strings): the number of the patient, the day into the pandemic they were diagnosed, the age of the patient, the sex/gender of the patient, the postal code of the patient, the state of the patient, the temperature of the patient, and the days the patient has been symptomatic
    - Initialize these attributes:
        - `self.num`: the number of the patient, an int
        - `self.day_diagnosed`: which day into the pandemic they were diagnosed, an int
        - `self.age`: the age of the patient, an int
        - `self.sex_gender`: the sex/gender of the patient, a string that is either M, F or X.
            - These are for man/male, woman/female or non-binary.
            - The French word for woman is ‘femme’; the French word for man is ‘homme’.
            - The value ‘H’ is short for ‘homme’.
            - Variants like boy/girl may appear in your data.
            - Look up any genders in your data that you do not recognize. A list of non-binary identities is available here: `https://nonbinary.miraheze.org/wiki/List_of_nonbinary_identities`
        - `self.postal`: the first three characters of the patient’s postal code, a string.
            - If they do not have a valid postal code (e.g. ‘N.A.’), use ‘000’.
            - A valid Montreal postal code should start with H, then a number, then a letter. (You do not have to validate the characters after the first three).
        - `self.state`: the state of the patient. Assume the input will be one of I, R or D.
        - `self.temps`: a list of floats, recording all the temperatures observed for this patient **in Celsius** (starting with the one given as input).
            - Note: in French, the comma is used for decimal points.
            - The input could be in Fahrenheit, so convert any temperature above 45 to Celsius. Round it to two decimals.
            - If you get a string which does not contain a number (e.g. ‘N.A.’ because the patient died), record this as 0.
        - `self.days_symptomatic`: how many days the patient has been symptomatic, an int

2. __str__ [4 points]

   - Return a string of the following attributes, separated by tabs: self.num, self.age, self.sex_gender, self.postal, self.day_diagnosed, self.state, self.days_symptomatic, and then all the temperatures observed separated by semi-colons
   - Example:

     ```
     >>> p = Patient('0', '0', '42', 'Woman', 'H3Z2B5', 'I', '102.2', '12')
     >>> print(str(p))
     '0   42   F   H3Z   0   I   12  39.0'
     ```

3. update [5 points]

   - Input: another Patient object
   - You can assume this object is based on an entry that was made *after* the one the current Patient is based on
   - If this other object's number, sex/gender, and postal code are all the same as the current patient:
     - Update the days the patient is symptomatic to the newer one
     - Update the state of the patient to the newer one
     - Append the new temperature observed about the patient. You can assume the other Patient has only one temperature stored in their `temps`.
   - Example:

     ```
     >>> p = Patient('0', '0', '42', 'Woman', 'H3Z2B5', 'I', '102.2', '12')
     >>> p1 = Patient('0', '1', '42', 'F', 'H3Z', 'I', '40,0 C', '13')
     >>> p.update(p1)
     >>> print(str(p))
     '0   42   F   H3Z   0   I   13  39.0;40.0'
     ```

   - Raise an AssertionError exception if num/sex_gender/postal are not the same

3.2 Stage Four [5 points]

Create, document and test the function, **stage_four**:

- Two inputs: input_filename and output_filename
- This will open the file with the name `input_filename`, and read the file line by line. As with other stages, be sure to set the encoding to utf-8.
- Create a new Patient object for each line. Do not do any conversions here — all the conversions should take place in the Patient initialization.
- Keep (and return) a dictionary of all the patients:
  - Use the patient’s number (as int) for the key, and the Patient objects for the values.
  - Whenever you see a new entry for an existing patient, update the existing Patient object rather than overwrite it.
- Write to the output file: every Patient converted to a string, sorted by patient number (separated by new lines)
- Example: if our input file is the output file from Stage 3’s example, we now have:

```
0   42  F   H3Z 0   I   12  40.0;39.13;39.45;39.5;39.36;39.2;39.0;39.04;38.82;37.7
1   73  M   H1M 1   I   5   40.0;0.0
2   40  F   H3X 1   I   9   39.0;39.0;39.22;39.2;38.2;37.4;37.4
3   18  F   H1T 2   I   8   39.2;39.93;40.0;38.5
```

- Return the dictionary of Patients
- Example:

```
>>> p = stage_four(’stage3.tsv’, ’stage4.tsv’)
>>> len(p)
1716
>>> print(str(p[0]))
0   42  F   H3Z 0   I   12  40.0;39.13;39.45;39.5;39.36;39.2;39.0;39.04;38.82;37.7
```

3.3 Fatality Probability by Age [5 points]

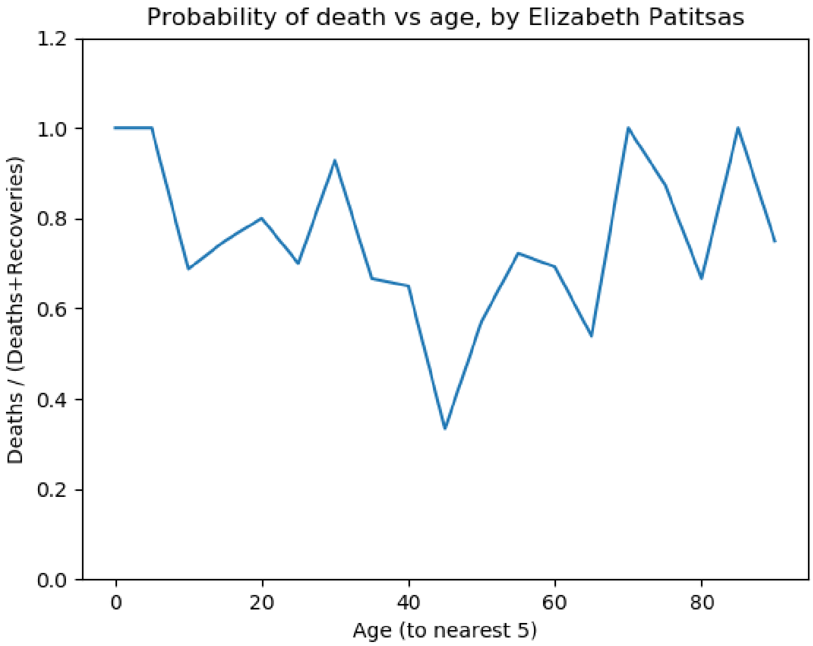
Create, document and test the function, **fatality_by_age**:

- Input: a dictionary of Patient objects
- Goal: plot the probability of fatality versus age
- For this plot, round patients' ages to the nearest 5 (e.g 23 becomes 25)
- To calculate probability of fatality, for each age group:

  how many people died / (how many people died + how many people recovered)

- Plot info:
  - Save your plot as `fatality_by_age.png`
  - Set the xlabel as 'Age'
  - Set the ylabel as 'Deaths / (Deaths+Recoveries)'
  - Set the y axis range from 0 to 1.2. You can do this with:

    ```
    plt.ylim((0, 1.2))
    ```
  - Title the plot 'Probabilty of death vs age ' and then append your name
- You should get a plot with one line, and could look like this:



- Return: list of probabilites of death by age group.
- Example (matches the plot, not the example files):

```
>>> p = stage_four('stage3.tsv', 'stage4.tsv')
>>> fatality_by_age(p)
[1.0, 1.0, 0.6875, 0.75, 0.8, 0.7, 0.9285714285714286, 0.6666666666666666,
0.65, 0.3333333333333333, 0.5714285714285714, 0.7222222222222222,
0.6923076923076923, 0.5384615384615384, 1.0, 0.875, 0.6666666666666666, 1.0, 0.75]
```

Closing Notes

There are many more things worth analyzing in the file you've cleaned! Some things epidemiologists would look at include:

- Estimating a basic reproduction number — how many people a person with the flu will infect
- Looking at a heat map of infections by part of the city (from postal codes)
- Whether there is a correlation between the average/maximum fever a patient has an their chance of death

If you want more practice with numpy and matplotlib, you might want to try plotting/fitting your data to figure those out!

Getting good data quickly is an important task for epidemiologists, to determine whether a pandemic has started and how to contain it. Speedy containment is vital for stopping pandemics.

Influenza is a family of viruses with many strains that have caused catastrophic pandemics. Spanish Flu in 1918 killed 20-100 million people, far more than World War I. The more recent Asian Flu (1957-8) and Hong Kong Flu (1968-9) both killed about a million people each.

Even the 'ordinary' seasonal varieties of influenza can kill many people. People who with compromised immune systems (e.g due to cancer, AIDS) are most at risk, which is why herd immunity is important. Get your flu shot!

Real Data Is Ugly

Go back to page 3 of the assignment, and revisit those 'Safe Assumptions'. Relaxing any of those makes data much harder to clean!

For example, if we don't restrict the dates to ISO format, you now have to figure out how dates are ordered. Sound tricky? Here's such a file you can try it out with: `https://www.cs.mcgill.ca/~patitsas/comp202/files/challenge.txt`

Real world data is often much uglier than what you saw in this assignment: missing entries and misspellings are common. And in some cases you could even have to deal with malicious (fake) entries that you have to try and identify to remove.

Data Science

If you enjoyed this assignment, you might want to find a summer job doing data science! Data cleaning is a huge part of what is called *data science*: using computational practices to analyse (often unstructured) data.

This skill set is in demand in the workforce! If you'd like to pursue this for work, you'll also want to take more statistics classes, and more computer science classes like COMP 250 to write code to efficiently process giant data sets. Hope to see you in COMP 250!