Part A (*Total 9 marks*)

For Part A, download the complete "Our World in Data COVID-19 dataset" ("owid-covid-data") from https://covid.ourworldindata.org/data/owid-covid-data.csv.

Part A Task 1 Data pre-processing (*3 marks*)

Program in python to produce a dataframe by

1. (2 marks) aggregating the values of the following four variables:

   - **total_cases**
   - **new_cases**
   - **total_deaths**
   - **new_deaths**

   by **month** and **location** in the year 2020.

   The dataframe should contain the following columns after completion of this sub-task:

   - **location**
   - **month**
   - **total_cases**
   - **new_cases**
   - **total_deaths**
   - **new_deaths**

   Note: if there are no entries for certain combinations of locations and months, there should be no entry for those combinations in the dataframe.

2. (1 mark) adding a new variable, **case_fatality_rate**, to the dataframe produced from sub-task 1. The variable, case_fatality_rate, is defined as *the number of deaths per confirmed case in a given period*. Do not impute missing values.

   The final dataframe should contain the columns **in the following order**:

   - **location**
   - **month**
   - **case_fatality_rate**
   - **total_cases**
   - **new_cases**
   - **total_deaths**
   - **new_deaths**

   and the rows are to be sorted by location and month in **ascending** order.

Print the first 5 rows of the final dataframe to the standard output.
Save the new dataframe to a CSV file named, "**owid-covid-data-2020-monthly.csv**" in the same directory as the python program. Your program should be called from the command line as follows:

```
python parta1.py owid-covid-data-2020-monthly.csv
```

Hint: You will need to use appropriate functions for the aggregation based on your understandings of the variables.

Part A Task 2 Visualisation (*2 marks*)

Program in python to produce two scatter plots:

1. (1 mark) a scatter plot of case_fatality_rate (on the y-axis) and confirmed new cases on the x-axis) by **locations** in the year **2020**.
   Output the plot to scatter-a.png in the same directory as the python program.
2. (1 mark) a second scatter plot of the same data with only one change: the x-axis is changed to a log-scale.
   Output the plot to scatter-b.png in the same directory as the python program. For this plot, apply preprocessing if necessary.

Your program should be called from the command line as follows:

```
python parta2.py scatter-a.png scatter-b.png
```


Part A Task 3 Discussion and visual analysis (*4 marks*)

A short report of your visual analysis of the two plots produced from Task 2.

It is expected that the visual analysis would include:

1. (1.5 marks) a brief introduction/description of the raw data, including the source, any limitations you observe in the data and all preprocessing steps taken on the raw data to produce the visualisations,
2. (1.5 marks) explanation of the plots and patterns observed, and
3. (1 mark) a discussion contrasting the two scatter plots.

The report is to be 500 - 600 (maximum) words excluding figures, about 1 page, in **pdf** format, and **must include the two plots, scatter-a.png and scatter-b.png, produced from Part A Task 2**.
The filename of the report must be "*owid-covid-2020-visual-analysis.pdf*".

Part B (*Total 9 marks*)

For Part B, download the cricket dataset from the LMS. This dataset contains a sample of cricket-related articles from BBC News. We wish to build a search engine that will allow a user to specify keywords and find all articles related to those keywords.

Part B Task 1: Regular Expressions (*1 mark*)

Each article contains a document ID which uniquely identifies the document. This document ID is comprised of four letters followed by a hyphen, followed by three numbers and optionally ending in a letter. For example, each of the following are valid document IDs:

- ABCD-123
- ABCD-123V
- XKCD-999A
- COMP-200

The document IDs are not located in a consistent place in each article. Use a regular expression to identify the document ID for each document in the dataset. Write a Python program in **partb1.py** that produces a CSV file called **partb1.csv** containing the filenames and Document IDs for each document in the dataset. Your CSV file should contain the following columns in the order below:

- **filename**
- **documentID**

Your program should be called from the command line along with the name of the CSV file:


```
python partb1.py partb1.csv
```


Part B Task 2: Preprocessing (*1 mark*)

We now wish to perform the following preprocessing on each article in the cricket folder in order to make them easier to search:

- Remove all non-alphabetic characters (for example, numbers and punctuation characters), except for spacing characters such as whitespaces, tabs and newlines.
- Convert all spacing characters such as tabs and newlines to whitespace and ensure that only one whitespace character exists between each word
- Change all uppercase characters to lower case

Create a Python program in **partb2.py** that performs this preprocessing.
Your program should be called from the command line along with the filename of a document. For example:

```
python partb2.py cricket001.txt
```

Your program should then load the specified file, perform the preprocessing steps above and print the results to standard output.

Hint: You may wish to create a function for performing this preprocessing as you will need to perform this pre-processing as part of each task in Part B

Part B Task 3: Basic Search (*2 marks*)

Create a Python program in **partb3.py** that will allow the user to search for articles containing particular keywords. Your program should be called from the command line along with the keywords being searched for. For example:

```
python partb3.py keyword1 keyword2 keyword3
```

You can assume each keyword will be separated by a whitespace character and that between 1 and 5 keywords will be entered. Your program should then return the document IDs of the documents that contain *all* of the keywords in the user's search query. For this task:

- You should check for matches *after* performing the preprocessing in Task 2. For example, searching for the word 'old' should return articles containing the words 'Old' or 'OLD'.
- The keywords that the user searches for are *separate* keywords. You are *not* required to match exact phrases. For example, if a user searches for the keywords 'captain early', these words do *not* need to appear consecutively in the document to constitute a match.
- Only documents that contain the actual keyword should return a match. For example, searching for the word 'old' should *not* return articles containing the word 'golden'.

Your program should output the document IDs of each article containing all of the specified keywords.

Hint: You may wish to load partb1.pkl back into your program.

Part B Task 4: Advanced Search (*2 marks*)

We now wish to expand the search features to enable inexact matching. For example, a user should be able to specify the keyword 'missing' and the search should also return articles containing the related words 'missed' or 'miss'. Create a Python program in **partb4.py** based on your response to Task 3 that uses a Porter Stemmer to enable this inexact matching. Your program should be called from the command line along with the keywords being searched for. For example:

```
python partb4.py keyword1 keyword2 keyword3
```

Your program should output the document IDs of each article containing all of the specified keywords, or words considered by the Porter Stemmer to have the same base. For this task:

- You should check for matches *after* performing the preprocessing in Task 2. For example, searching for the word 'old' should return articles containing the words 'Old' or 'OLD'.
- The keywords that the user searches for are *separate* keywords. You are *not* required to match exact phrases. For example, if a user searches for the keywords 'captain early', these words do *not* need to appear consecutively in the document to constitute a match.
- Other than inexact matches permitted by the Porter Stemmer, only documents that contain the actual keyword should return a match. For example, searching for the word 'old' should *not* return articles containing the word 'golden'.

Note that other than the final point this list of requirements is the same as for Task 3.

Part B Task 5: Search Rankings (*3 marks*)

We wish to further expand the search feature to enable documents to be ranked, so that those most relevant to the user's keywords are displayed at the top of the list. One way of computing such a ranking is to use TF-IDF along with the cosine similarity measure as discussed in lectures. Create a Python program in **partb5.py** based on your response to Task 4 that ranks articles returned by Task 4 by cosine similarity score.

Your program should be called from the command line along with the keywords being searched for. For example:

```
python partb5.py keyword1 keyword2 keyword3
```

Your program should output:

- The headings 'documentID' and 'score'
- The document IDs of each article containing all of the specified keywords, or words considered by the Porter Stemmer to have the same base.
- The cosine similarity score between the vector of stemmed keywords and the vector of stemmed words appearing in the document for each document matched, rounded to four decimal places.

You should assume that the collection being used by TF-IDF is the complete list of stemmed words contained in articles returned by your Task 4 search. The output should be sorted in descending order by cosine similarity score with the search query. For example, one sample output might look like this:


```
documentID score
JDKC-105M 0.0618
BTAR-174V 0.0182
```

Part C(*Total 2 marks*)

GitHub Submission

Ensure all of your completed code files as well as your report have been pushed to the github repository you created in the 'Getting Started' section. We strongly encourage you to push an updated version of your code to your github repository each time you make a major change. Your repository must also contain a README file, which must contain your name and student ID. It must also contain a brief description of your project and a list of dependencies.

Submission Instructions

Submit all pythin scripts and the pdf discussion report via LMS. A complete submittion includes the following items:

1. parta1.py
2. parta2.py

3. owid-covid-2020-visual-analysis.pdf
4. partb1.py
5. partb2.py
6. partb3.py
7. partb4.py
8. partb5.py
9. A link to your GitHub repository

You must also have pushed the above files to your github repository, which the teaching staff already have access to.

Extensions and late submission penalties

If requesting an extension due to illness, please submit a medical certificate to the lecturer. If there are any other exceptional circumstances, please contact the lecturer with plenty of notice. Late submissions without an approved extension will attract the following penalties

- $0 < hourslate <= 24$ (2 marks deduction)
- $24 < hourslate <= 48$ (4 marks deduction)

- $48 < hourslate <= 72$: (6 marks deduction)
- $72 < hourslate <= 96$: (8 marks deduction)

- $96 < hourslate <= 120$: (10 marks deduction)
- $120 < hourslate <= 144$: (12 marks deduction)
- $144 < hourslate$: (20 marks deduction)

where *hourslate* is the elapsed time in hours (or fractions of hours).