Information and Due date

The assignment is an individual assignment and is worth **20** marks, (**20%** of subject grade).

Due date

5 pm Friday 21 April 2023 Australia/Melbourne time. The estimated time commitment 20 - 30 hours.

Submission instructions

The assignment consists of 6 tasks. They are to be submitted via the Code Challenge '***Tasks (20 marks)***' slide below. Specifications to the tasks are in the *Description* section.

Tasks 1 to 5 are coding tasks; each task is to be implemented in a function inside a python script. For example, function `task1(..)` in `task1.py` implements task 1 and function `task2(..)` in `task2.py` implements task 2.

> You are free to write any helper functions (even in another file) if you like, but *do not change the function definition of* `task1(..)`. For example, do not add any arguments to `task1(..)`. This applies to all tasks. In addition, do not modify `main.py`.

Each coding task produces one or more output files in either `csv`, `json`, or `png` formats. Tasks 1 to 5 are scaffolded in the Code Challenge. You can develop and test your solutions directly in the Code Challenge.

Task 6 is a short written report on the analysis of the data based on the coding tasks 1 - 5. You will need to incorporate the output from tasks 1 - 5 into your report. To run your solution for a specific task, open the terminal and run `python main.py the_task mode`, for example run `python main.py task1 full` to run task 1 on the `full` dataset. You can then download the outputs and use them in the written report.

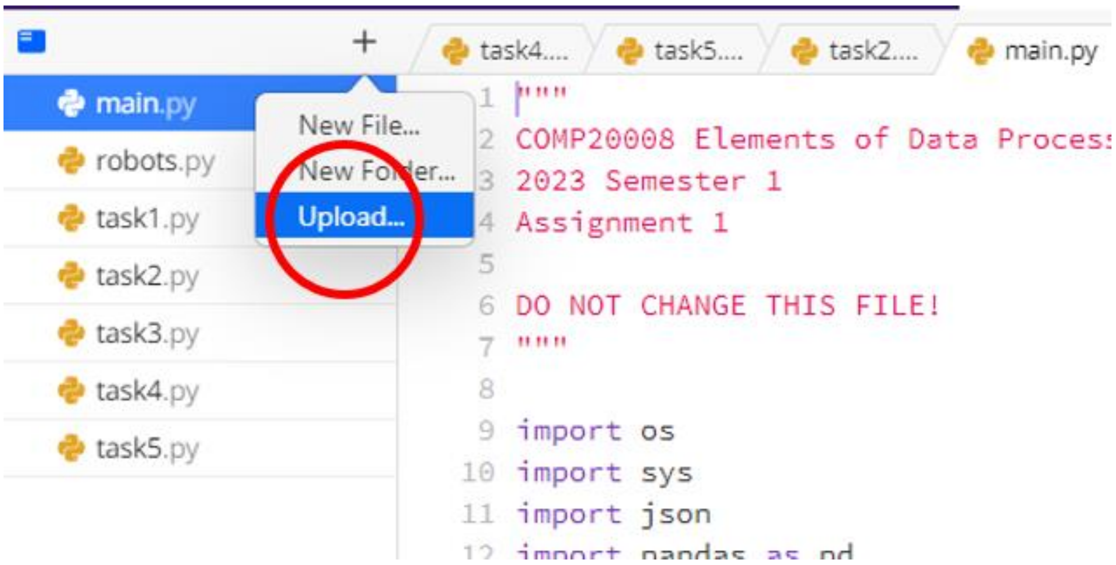
Tasks can be run using commands typed into the terminal. The terminal will appear underneath the code you are editing, down the bottom of the page.

To run all tasks, open the terminal and run `python main.py all full`

To run specific tasks, we provide examples in the "Tasks" section.

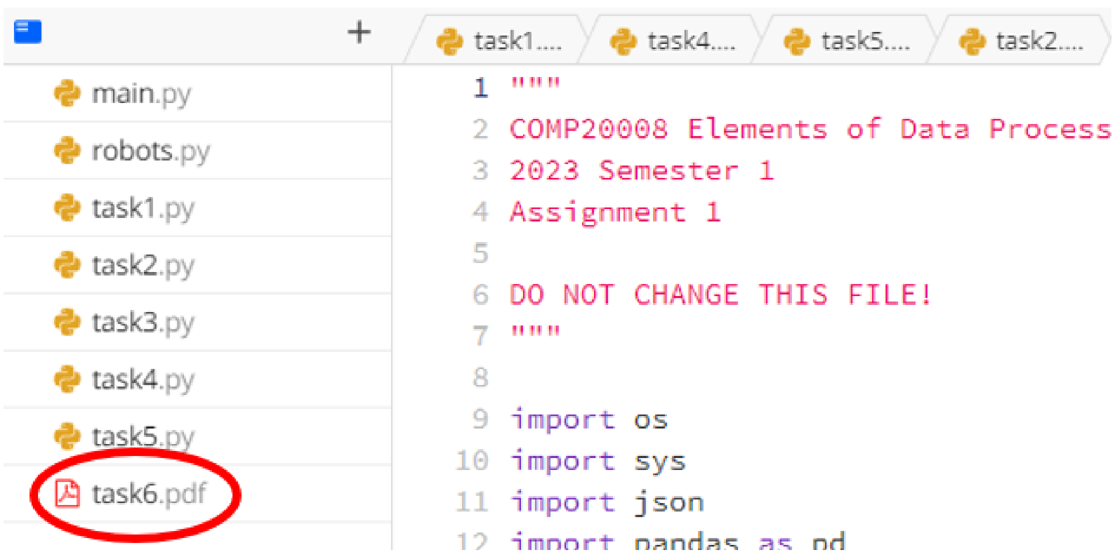
You need to **upload** the completed report to the Code Challenge for submission.

Click on the `+` to upload the report.



```
"""
COMP20008 Elements of Data Proces
2023 Semester 1
Assignment 1

DO NOT CHANGE THIS FILE!
"""

import os
import sys
import json
import pandas as pd
```

You should see the file after it is uploaded.

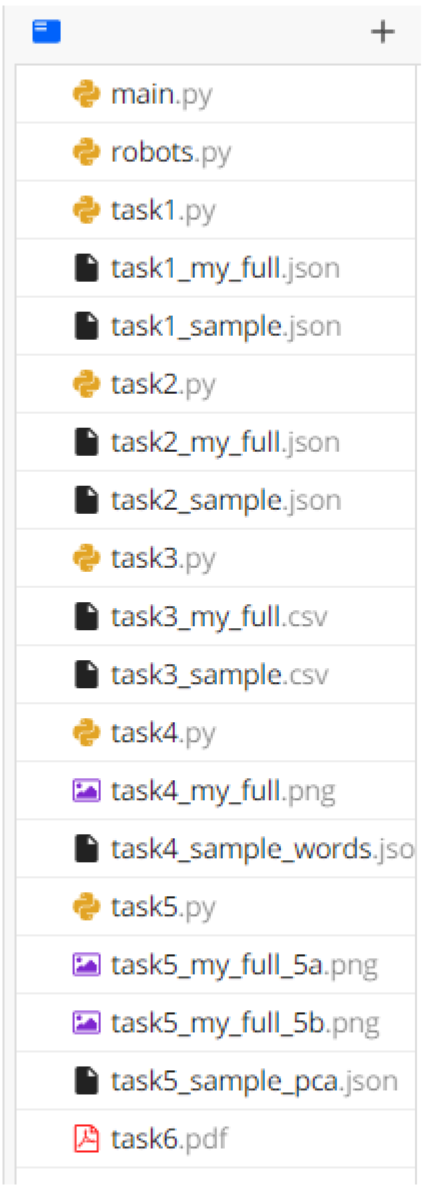


```
"""
COMP20008 Elements of Data Process
2023 Semester 1
Assignment 1

DO NOT CHANGE THIS FILE!
"""

import os
import sys
import json
import pandas as pd
```

Submission instructions

You can submit your solutions multiple times. We will only mark the final version before the due date (and time). While you can incrementally test individual tasks, each submission includes all 6 tasks in its entirety.

All of the output files from the coding tasks must be included in the final submission. An example of the required files for a correct and complete submission is shown below.



main.py
robots.py
task1.py
task1_my_full.json
task1_sample.json
task2.py
task2_my_full.json
task2_sample.json
task3.py
task3_my_full.csv
task3_sample.csv
task4.py
task4_my_full.png
task4_sample_words.jso
task5.py
task5_my_full_5a.png
task5_my_full_5b.png
task5_sample_pca.json
task6.pdf

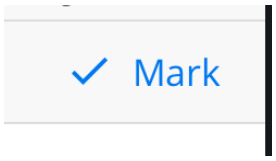
They include

- all python scripts for coding tasks 1 - 5
- all output files (`csv`, `json`, or `png` files) from tasks 1 - 5.

- the final report `task6.pdf`

Make sure that you **run your final scripts to generate the most up to date output files** and the final version of the written report is uploaded in the Code Challenge before making the final submission.

Run `python main.py all full` in the terminal then submit your solutions by clicking on the '**Mark**' button in the Code Challenge



Checking your submissions

After you make a submission, you will receive feedback from a set of basic verifications. If you pass all the tests, it means that you have included all the expected files, including the source code scripts, outputs, and report.

If you see errors after a submission, please check the errors, fix them and resubmit before the deadline. It is your responsibility to follow the requirements.

*Careless mistakes such as forgetting to include output files produced by the code in your submission will result in **losing** marks, even if your implementation is perfect.*

Background

When searching for websites in a search engine, pages from a wide range of diverse areas may be returned, e.g., topics from mathematics to politics to weather to sports. A well known website which organises a wide variety of these topics is Wikipedia, the online collaborative encyclopaedia.

In this assignment, you will be working on analysing data from a site hosting over 500 pages taken from Wikipedia.

Using this site, you will have an opportunity to learn and practice how to retrieve content and visualise the different vocabularies used across these pages.

Learning outcomes

- Be able to read and write data in different formats, and combine different data sources.
- Practice text processing techniques using Python.
- Practice data processing, exploration and visualisation techniques with Python.
- Practice visual analysis and written communication skills.

Dataset

There are two datasets provided, these are in the form of websites. The sample website crawl has 10 pages, and the full website crawl has over 500 pages.

The sample website crawl branches out from two seed links:

- http://115.146.93.142/samplewiki/A12_scale
- http://115.146.93.142/samplewiki/Gerard_Maley

The full website crawl branches out from two seed links:

- http://115.146.93.142/fullwiki/A12_scale
- http://115.146.93.142/fullwiki/Gerard_Maley

Note that there may be some crawler traps and unrelated links present in the pages. All pages you need to crawl will be in the `/samplewiki/` or `/fullwiki/` section of the server, other links can freely be ignored. The dataset may also be easily misread, when in doubt, use the `apparent_encoding` attribute from the requests library's `get` response.

Tasks (20 marks)

Task 1: Get All Links (3 marks)

> Guiding context: It may be useful for evaluating the quality and correctness of the process to initially examine the links which are in scope. Links excluded from the analysis will be those which will not be successfully retrieved or those which are beyond the scope of your task in evaluating the pipeline (e.g. links outside the current domain). In verifying your work, it is worth keeping in mind that errors earlier in the pipeline are more likely to lead to larger issues later in the pipeline.

Implement the function `task1(..)` in `task1.py` that takes two arguments:

- The first argument is a list of starting links in the format of a list of strings and
- The second argument is a filename.

Your function should crawl each starting link and output a `json` formatted output file with the filename given in second argument and should be in the following format:

```
{pageX: [linkXA, linkXB, ...], pageY: [linkYA, linkYB], ...}
```

where `pageX` and `pageY` are the starting link strings, `linkXA` and `linkXB` are the pages found through crawling starting from `pageX`, and `linkYA` and `linkYB` are pages found through crawling starting from `pageY`.

All URL strings should be fully qualified, including the protocol e.g. `http://115.146.93.142/samplewiki/A12_scale` should be used, even if the actual link found on the page is `/samplewiki/A12_scale`. Each *webpage* should be included in the list for each starting link only once and the list should be sorted alphabetically. You must crawl only links in the same domain as the starting link strings and should not include any links in the output which are not successfully retrieved.

> You are free to write any helper functions (even in another file) if you like, but *do not change the function definition of* `task1(..)`. For example, do not add any additional arguments to `task1(..)`. This applies to all tasks.

To run your solution, open the terminal and run `python main.py task1 full` . You can verify your answer against the sample data using `python main.py task1 sample`, this will check the output against the sample data.

> Note that for all tasks, the sample data verification is not intended to verify the correctness of your approach or the structure of your solution. Verifying your output and process are correct is part of all tasks, so you should treat it as a sanity check, ensuring you haven't missed any aspects of basic formatting. In all tasks, this sample used for verification should be considered arbitrarily selected, so there is no implicit guarantee or expectation that it will cover all cases or highlight issues present in the full scale task.
>
> For performance reasons, your implementation of all tasks should cache (i.e. download once and keep in memory for the duration of the task) and respect the `robots.txt` on the site. For Task 2, your code will not be tested on URLs which are excluded under `robots.txt`, but you are certainly welcome to handle this behaviour there as well if you like.

Task 2: Extracting Words from a Page (4 marks)

> Guiding context: For this task, we are looking at a single page to see that we are successfully processing its content before we apply this process to all pages in later tasks. To make this as useful as possible for diagnosing issues which might affect later tasks, we will perform all steps in the pre-processing, this will include removing common elements not related to the content of the page (e.g. navigation menus and quality issues) and to remove low information words. The output is intended to be a list of words, useful to compare against the HTML page to see what has been removed and what has been retained.
>
> As a warning, this kind of task is historically error prone, it is critically important to understand what you are doing at each and every step and to consider additional methods to verify your approach - any error in any step of your approach can lead to either subtle errors or major errors, both of which are unacceptable in good data science. At some point in your approach, it is strongly recommended you spend a bit of time thinking about and trying alternative approaches and that you investigate any discrepancies.

For this task you should use BeautifulSoup. Each page will contain a `div` with the `id` of `mw-content-text` , which processing will be restricted to. You can use inspect element tool on one of the pages (or through Wikipedia itself) to see what this typically covers.

Implement the function `task2(..)` in `task2.py` which takes a URL and a filename and outputs a `json` formatted file with that filename. The format of the JSON object output should be the name of the URL as the string and the value should be a list of words. If the page cannot be retrieved, the list should contain no values.

> The page fetched should be encoded to its `apparent_encoding` attribute.

Your function should save its output to the filename given in the second argument.

Two stages of pre-processing should occur, the first narrowing down and removing all irrelevant elements from the page, and the second which produces word tokens and removes irrelevant tokens. The first stage comprises eight steps:

1. For the rest of the steps, operate on the `div` element with `id` of `mw-content-text` discarding other parts of the page.
2. From the remaining page tree, remove all `th` elements with the `class` of `infobox-label` .
3. From the remaining page tree, remove all `div` elements with the `class` of `printfooter` .
4. From the remaining page tree, remove all `div` elements with the `id` of `toc` .
5. From the remaining page tree, remove all `table` elements with the `class` of `ambox`.
6. From the remaining page tree, remove all `div` elements with the `class` of `asbox`.
7. From the remaining page tree, remove all `span` elements with the `class` of `mw-editsection`.

8. From the remaining `mw-content-text` tree, extract the text from the page using BeautifulSoup, this extraction should use *a space separator* (`' '`) between elements, and should **not** connect text from adjacent elements. For example if a section of the remaining tree contents were `<p>a</p><p>b</p>` it should become `'a b'` , not `'ab'`.

After the first stage's step 1 -- 8 have been completed, the remaining text should be converted to tokens in the following steps

1. Change all characters to their `casefold`ed form and then normalize all page text to its `NFKD` form.
2. Convert all non-alphabetic characters (for example, numbers, apostrophes and punctuation), except for spacing characters (for example, whitespaces, tabs and newlines) and the backslash character (`'\'`) to single-space characters. For example, `'&'` should be converted to `' '`. You should consider non-English alphabetic characters as non-alphabetic for the purposes of this conversion.
3. Convert all spacing characters such as tabs and newlines into single-space characters, and ensure that only one whitespace character exists between each token.

4. The text should then be converted to explicit tokens. This process should use single-space characters as the boundary between tokens.
5. Remove all stop words in `nltk`’s list of English stop words from the tokens in the page text.
6. Remove all remaining tokens that are less than two characters long from the page text.
7. Each token should then be converted to its Porter stemming algorithm stemmed form. This should use all the default improvements present in the NLTK PorterStemmer (including both the additional improvements beyond the original algorithm and NLTK's improvements).

Though you are expected to ensure and verify all steps are correctly carried out, if you decide to use library functions for step 4, it is critically important you carefully read and fully understand the documentation as failing to read documentation could lead to major errors in pre-processing steps.

Once steps 1 -- 7 are done, build a JSON file representing the page. The JSON file should contain a JSON object containing one string matching the fully qualified url requested and the associated value should be an array of strings corresponding to the tokens produced by the above pre-processing.

If the page is not successfully retrieved, the array should contain no strings.

To run your solution, open the terminal and run `python main.py task2 full` - this runs the task against a different url to the sample data. You can verify your answer against the sample data using `python main.py task2 sample`, this will check the output against the sample data. For this task an additional option is provided, `extra`, by adding a link at the end of the task, you can view the output for a given URL, e.g. `python main.py task2 extra http://115.146.93.142/samplewiki/Pythagorean_tuning` .

Task 3: Producing a Bag Of Words for All Pages (2 marks)

> Guiding Context: For this task, we're looking at putting the first two tasks together to get all the words on each page from all pages to produce a representation of all our data which is easily processed later. This will also allow us to look for procedural errors in our text processing pipeline which may not have been present in our initial sample of the dataset.

Implement the function `task3(..)` in `task3.py` which takes a dictionary containing key/value pairs matching the output format of Task 1 and a filename and outputs a `csv` formatted file with that filename.

For each key in the dictionary, all links should be visited. The key should be considered the `seed_url` and the link should be considered the `link_url`. The HTML page for each `link_url` should be retrieved and processed into a list of tokens according to both stages of pre-processing in Task 2, this list of tokens should then be joined with a single space and form the `words` string associated with the page. Once all pages are processed, a dataframe should be created which is output to a new `csv` file with the given filename.

The `csv` output should contain the headings `link_url`, `words`, `seed_url` and each row should contain the details of one page, with

- `link_url` being the fully qualified URL of the page the bag of words corresponds to,
- `words` being a string comprising all tokens for the `link_url`'s page, separated by exactly one space character and
- `seed_url` being the key which the link URL was taken from in the dictionary argument.

If no tokens are retrieved after pre-processing, the `words` string should be an empty string `''`.

The rows in the `csv` file should be in ascending order of `link_url` and then (where the `link_url` is the same for two or more pages) `seed_url`.

To run your solution, open the terminal and run `python main.py task3 full` - this will first run `task1` and then use the output of `task1` as input to `task3`. You can verify your answer against the sample data using `python` `main.py task3 sample`, this will check the output against the provided sample data.

Task 4: Plotting the Most Common Words (2 marks)

> Guiding Context: For this task, we're interested in what common words are present in each of the seed URL's crawls. We'd like to know whether our pre-processing so far is effective in removing words which aren't meaningful and to get a very general feel for how much overlap there is in the two vocabularies.

Implement the function `task4(..)` in `task4.py` which takes a pandas dataframe matching the output format of Task 3 and one filename and outputs a plot to that filename. `task4(..)` should return a dictionary where each `seed_url` is a key and the value is the list of 10 most common words.

In this task you should generate a plot which should allow the comparison of the top 10 most common words in each `seed_url` . Here the metric of number of total occurrences across all pages should be used (e.g. if one page has two uses of a word, both should be counted in the total for the `seed_url`). For consistency in your output, if multiple words appear equally frequently, they should appear in alphabetical order in your plot.

To run your solution, open the terminal and run `python main.py task4 full` . To verify the top 10 words for the sample dataset, use `python main.py task4 sample` .

Task 5: Dimensionality Reduction (3 marks)

> Guiding Context: Task 4 provided some general insight into what words are often present in each general topic. This gives us a bit of an idea of what we might want to look for, but if we want to work out which topic a particular page is about, this alone might not be the most helpful tool! In this task we're interested in visualising the high dimensionality space produced by the bag of words by reducing it down to two components to plot. This is another step in verifying our results and determining whether our approach appears sensible, as with all tasks, it may be useful to look at the question you are trying to answer. This allows us to determine whether the processing has clearly separated the data or whether there's a high degree of ambiguity.

Implement the function `task5(..)` in `task5.py` which takes one pandas dataframe matching the output format of Task 3 and two filenames and outputs a plot to each of the given filenames.

In `task5(..)` you should first produce a bag of words representation of the words across all pages and then perform normalisation using sklearn's `max` norm fit to the dataframe, following this, you should perform Principal Component Analysis using `2` components to the normalised dataframe.

`task5()` should output two files:

- The plot output to the first file should show the top 10 most positively weighted tokens and their weights and 10 most negatively weighted tokens and their weights for each component.
- The plot output to the second file should show where articles from each `seed_url` fall on the 2-component axes. The links from both dataframes should be included in the plot.

`task5()` should return a dictionary where the PCA component ID is the key (`0` for the component explaining the greatest variance and `1` for the component explaining the next greatest variance) and the value is also a dictionary, where this nested dictionary has four keys, `"positive"`, `"negative"`, `"positive_weights"`, `"negative_weights"` and the associated value of each key is as follows:

- The value for the `positive` key should be a list of the top 10 tokens with highest magnitude positive weights.
- The value for the `negative` key should be a list of the top 10 tokens with highest magnitude negative weights.
- The value for the `positive_weights` key should be the weight in the PCA decomposition of the relevant token in the `positive` token list.
- The value for the `negative_weights` key should be the weight in the PCA decomposition of the relevant token in the `negative` token list.

Where a `random_state` option is provided, you should use `535`.

To run your solution, open the terminal and run `python main.py task5 full` . To verify the top 10 PCA components for the sample dataset, use `python main.py task5 sample` .

> Note that minor numerical errors could produce variation in your results, but non-numerical errors can also produce variation in your results - these two factors mean that as with all sample dataset verification you should not blindly trust your results are correct, but instead ensure you verify you are following the correct process.

Task 6: Analysis Report (6 marks)

Write a brief report of no more than 700 words to summarise your analytical findings. You should incorporate the plots in tasks 4 and 5 for the **full** dataset in your analysis and also include the following:

- Comments comparing the ten most common words in each `seed_url` in `task4.png` and why differences might be present (1 mark)
- An interpretation of what words we might not be surprised to find in articles for each `seed_url` based on the information in `task5a.png` and `task5b.png` (1 mark)
- An interpretation of the distribution of `urls` shown in `task5b.png` and whether you think you could determine which `seed_url` a new

unseen link originated from when plotted in the 2D space after applying PCA (1 mark)

- A brief discussion of the limitations of this dataset, the limitations of the processing techniques and what could be done in the future to provide further insights.' (2 mark)

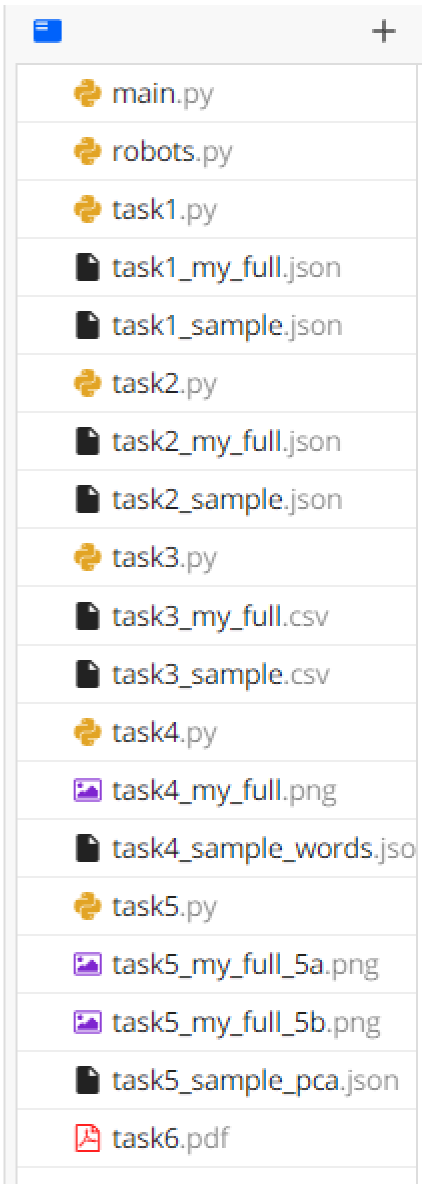
The report should be coherent, clear, and concise. Use of bullet points is acceptable. (1 mark)

Submit your report by uploading the pdf report called `task6.pdf`

To check if you have uploaded the file correctly, open the terminal and run `python main.py task6`

Submission

All of the output files from the codes must be included in the submission. An example of the required files for a correct and complete submission is shown below.



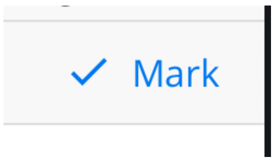
main.py
robots.py
task1.py
task1_my_full.json
task1_sample.json
task2.py
task2_my_full.json
task2_sample.json
task3.py
task3_my_full.csv
task3_sample.csv
task4.py
task4_my_full.png
task4_sample_words.jso
task5.py
task5_my_full_5a.png
task5_my_full_5b.png
task5_sample_pca.json
task6.pdf

They include

- python scripts for coding tasks 1 - 5
- output csv, json, and png files from tasks 1 - 5.
- the final report task6.pdf

Make sure that you **run your final scripts to generate the most up to date output files** and the final version of the written report is uploaded in the Code Challenge before making the final submission.

Submit your solutions by clicking on the '**Mark**' button in the Code Challenge



After you make a submission, you will receive feedback from a set of basic verifications. If you pass all the tests, it means that you have included all the expected files, including the source code scripts, outputs, and report.

If you see errors after a submission, please check the errors, fix them and resubmit before the deadline. It is your responsibility to follow the requirements. Careless mistakes such as forgetting to include output files produced by the code in your submission will result in **losing** marks, even if your implementation is perfect.