
# FIT9133 Assignment #3
# Building a Stylometric Analyser
# Semester 1 2018

Dr Jojo Wong
Lecturer, Faculty of IT.
Email: Jojo.Wong@monash.edu


May 14, 2018

## Revision Status

$Id: FIT9133-Assignment-03.tex, Version 1.0 2018/05/04 19:05 Jojo $

$Id: FIT9133-Assignment-03.tex, Version 1.1 2018/05/08 19:35 Jojo $

$Id: FIT9133-Assignment-03.tex, Version 1.2 2018/05/14 10:35 Jojo $

# Contents

# 1 Introduction

This assignment is due on **27 May 2018 (Sunday) by 23:55pm**. It is worth **10% of the total unit mark**. A penalty of 5% per day will apply for late submission. Refer to the FIT9133 Unit Guide for the policy on extensions or special considerations.

Note that this is **an individual assignment** and **must be your own work**. Please pay attention to Section 4.2 of this document on the university policies for the *Academic Integrity, Plagiarism and Collusion*.

This second assignment consists of **five tasks** and all the tasks should be submitted as **separate Python modules (classes or executable programs)** with supporting documentation on its usage. All the Python files and any supporting documents should be compressed into one single file for submission. (The submission details are given in Section 4.)

**Assessment: Task 1 and Task 2 are weighted** 15% **each; Task 3 is weighted** 20%**; Task 4 and Task 5 are weighted** 25% **each.**

# 2 Stylometric Analyser

### Stylometry

Stylometry is an application of statistical methods for linguistic style analysis. It is often used to attribute authorship to anonymous or disputed works (often in written language) based on the linguistic behaviours or characteristics manifested in the texts. One of the classic authorship attribution studies was whether William Shakespeare wrote all his works, in particular one of his popular plays — *Henry VI Trilogy* — was highly disputed to have been written or co-authored by Christopher Marlowe.

In this assignment, you will implement a simple stylometric analyser to investigate the linguistic behaviours of both authors. The analyser is able to perform basic statistical analysis on a number of linguistic features and also present the analysis results using some form of visualisation.

### The Dataset

For the purpose of this assignment, six specific written works were selected from a public available corpus[1] — two of which where the authorship was attributed to the specific author; and two other works which were known to be disputed between the two authors (but were attributed to Shakespeare). Table 1 presents the title of each work and the corresponding file name.

| Writer | Title of the Text | File Name |
|---|---|---|
| Marlowe | Edward II | `Edward_II_Marlowe.tok` |
| Marlowe | The Jew of Malta | `Jew_of_Malta_Marlowe.tok` |
| Shakespeare | Richard II | `Richard_II_Shakespeare.tok` |
| Shakespeare | Hamlet | `Hamlet_Shakespeare.tok` |
| Shakespeare | Henry VI Part 1 | `Henry_VI_Part1_Shakespeare.tok` |
| Shakespeare | Henry VI Part 2 | `Henry_VI_Part2_Shakespeare.tok` |

Table 1: Six selected works from the two authors, Shakespeare and Marlowe

The dataset provided to you (available on Moodle) is in textual form as well as in the *tokenised* format, where individual tokens are separated by spaces. This format enables you to determine the boundaries between words and punctuations. A short excerpt from *Hamlet* by Shakespeare is presented in Figure 1.

[1] `https://sites.google.com/site/nealpfox/research/authorship`

```
Who 's there ?
Nay , answer me : stand , and unfold yourself .
Long live the king !
Bernardo ?
He .
You come most carefully upon your hour .
' Tis now struck twelve ; get thee to bed , Francisco .
For this relief much thanks : 'tis bitter cold ,
And I am sick at heart .
Have you had quiet guard ?
Not a mouse stirring .
Well , good night .
If you do meet Horatio and Marcellus ,
The rivals of my watch , bid them make haste .
```

Figure 1: An excerpt of Hamlet by Shakespeare

## 2.1 Task 1: Setting up the preprocessor

In the first task, you are required to define a class that will perform the basic preprocessing on each input text. This class should have one instance variable which is a list that holds the individual tokens of the entire text as a result of applying the `tokenise()` method defined in this class.

The implementation of this preprocessor class should include the following four methods:

- `__init__(self)`:
  This is the constructor that is required for creating instances of this class. You should define a Python list to hold the individual tokens of a given input text.

- `__str__(self)`:
  Re-define this method to present the total number of tokens for the tokenised input text in readable format. This method should return a string.

- `tokenise(self, input_sequence)`:
  This is the method that tokenises or splits a given input text (indicated by the argument `input_sequence`) into individual tokens — words and punctuations. You should not return any values from this method.

- `get_tokenised_list(self)`:
  This method is defined to return the tokenised list (i.e. the instance variable of this class).

  **Note:** You are allowed to use the built-in String methods if necessary.

You should name this class as "Preprocessor" and the Python file as `preprocessor_StudentID.py`.

## 2.2 Task 2: Building a class for analysing characters

In this task, you are required to define a class for analysing the number of occurrences for each character from a given tokenised list, including any letters ('**A**' to '**Z**'), numerals ('**0**' to '**9**'), and punctuations. This class should have one instance variable for which the data type you have to determine. You should not use the Python dictionary in this assignment; but to consider carefully a data type from either **Numpy** or **Pandas** which is suitable for keeping track of the number of occurrences (or frequency) for each character.

The implementation of this character analyser class should include the following four methods:

- `__init__(self)`:
  This is the constructor that is required for creating instances of this class. You should define a data structure based on your choice of data type for storing each character frequency (i.e. the instance variable) in the constructor.

- `__str__(self)`:
  Re-define this method to present the number of occurrences for each character in a readable format. You should return a formatted string in this method.

- `analyse_characters(self, tokenised_list)`:
  This is the method that performs the frequency analysis on a given tokenised list at the character level. This method should accept the tokenised list as the argument, and attempt to count the occurrences for each of the characters (letters, numerals, and punctuations) encountered in the given tokenised list. The character frequencies should be stored in the data structure (i.e. instance variable) defined in the constructor.

- `get_punctuation_frequency(self)`:
  This method is defined to return only the frequency of all different punctuations. You should determine the suitable data type to be returned from this method.

You should name this class as "CharacterAnalyser" and the corresponding Python file as `character_StudentID.py`.

## 2.3 Task 3: Building a class for analysing words

In this task, you are required to define a class for analysing the number of occurrences for each word from a given tokenised list. This class should also have one instance variable for which the data type you have to determine. Similar to Task 2, you should not use the Python dictionary in this class when defining the instance variable; but to consider carefully a data type from either **Numpy** or **Pandas** which is suitable for keeping track of the number of occurrences (or frequency) for each word.

The implementation of this word analyser class should include the following five methods:

- `__init__(self)`:
  This is the constructor that is required for creating instances of this class. You should define a data structure based on your choice of data type for storing each word frequency (i.e. the instance variable) in the constructor.

- `__str__(self)`:
  Re-define this method to present the number of occurrences for each word in a readable format. You should return a formatted string in this method.

- `analyse_words(self, tokenised_list)`:
  This is the method that performs the frequency analysis on a given tokenised list at the word level. This method should accept the tokenised list as the argument, and attempt to count the occurrences for each of the words. The word frequencies should be updated in the data structure (i.e. instance variable) defined in the constructor.

- `get_stopword_frequency(self)`:
  This method is defined to return only the frequency of *stopwords* or *function words*. (This type of words is often considered as context independent since they have little lexical meaning in a sentence and are used to express grammatical relationships between words.) For the purpose of this assignment, we consider a basic list of stopwords obtained from Onix Text Retrieval Toolkit[2]. You should determine the suitable data type to be returned from this method.

- `get_word_length_frequency(self)`:
  This method is defined to return the number of occurrences for words that are of the same length. You will have to first determine all the different word lengths before you could count how many of those words are of the same word length. You should determine the suitable data type to be returned from this method.

You should name this class as "WordAnalyser" and the Python file as `word_StudentID.py`.

[2] `http://www.lextek.com/manuals/onix/stopwords1.html`

## 2.4 Task 4: Building a class for visualising the analysis

You are required to define a class in this task for visualising the stylometric analysis. You will define a number of methods in this class (described below) to present four stylistic features for each of the six input texts that you would have analysed through the methods provided in classes from Task 2 and Task 3.

For the purpose of this assignment, we will focus on the following four stylistic features:

- **character frequency:** the relative frequency of each different character — the frequency of each character is based on the length of a given text;
- **punctuation frequency:** the relative frequency of each different punctuation — the frequency of each punctuation is based on the length of a given text;
- **stopword frequency:** the relative frequency of each different stopword — the frequency of each stopword is based on the length of a given text;
- **word length frequency:** the relative frequency of each different word length — the frequency of each word length is based on the length of a given text.

This class should have one instance variable for which the data type you have to determine. This data structure should be able to represent all the different stylistic features extracted for each of the six input texts. Similar to Task 2 and Task 3, you should not use the Python dictionary in this class when defining the instance variable; but to consider carefully a suitable data type from either **Numpy** or **Pandas**.

The implementation of this sentence analyser class should include the following five methods:

- `__init__(self, all_text_stats)`:
  This is the constructor that is required for creating instances of this class. You should define and initialise a data structure (i.e. the instance variable) for representing all relative frequencies for each of the four stylistic features analysed for the six input texts. The argument for this constructor, `all_text_stats`, should contain all the statistics of the six input texts.
- `visualise_character_frequency(self)`:
  This method plots a suitable graph to present the relative frequencies of each different character for the six input texts. (Note: You should disregard the frequencies of punctuations in this method.)
- `visualise_punctuation_frequency(self)`:
  This method plots a suitable graph to present the relative frequencies of each different punctuation for the six input texts.

- `visualise_stopword_frequency(self)`:
  This method plots a suitable graph to present the relative frequencies of each different stopword for the six input texts.

- `visualise_word_length_frequency(self)`:
  This method plots a suitable graph to present the relative frequencies of each different word length for the six input texts.

  **Note:** You may use either the built-in functions from **Matplotlib** or **Pandas** for plotting the graphs. Also note that, none of these methods are required to return any values.

You should name this class as "AnalysisVisualiser" and the corresponding Python file as `visualiser_StudentID.py`.

## 2.5 Task 5: Putting all the classes together

In this final task, you will design the main program that enables you to compare the different stylistic features analysed from the six selected texts authored by Shakespeare and Marlowe.

You should at least define two functions, although you may define additional functions if required.

- The `main()` function is to drive the flow of execution of the program. You would have to create multiple instances or objects from Tasks 1-4.

- The `read_input()` function is for reading in each of the six input texts. You have to consider any potential I/O exceptions or any other exceptions that might occur during the file input process. Hence, the `main()` function should be implemented to handle any of these exceptions that might be thrown from `read_input()`.

**Note:** The flow of execution of the program is at your discretion. You will be assessed on how the different classes defined in Tasks 1-4 are appropriately used. Also note that, if your classes from Tasks 1-4 are throwing some exceptions, these execptions should be handled appropriately in the `main()` function.

You should name the Python file of this task as `main_StudentID.py`.

# 3 Important Notes

## 3.1 Documentation

Commenting your code is essential as part of the assessment criteria (refer to Section 3.2). You should also include comments at the beginning of your program file, which specify your name, your Student ID, the start date and the last modified date of the program, as well as with a high-level description of the program. In-line comments within the program are also part of the required documentation.

## 3.2 Marking Criteria

The assessment of this assignment will be based on the following marking criteria. The same marking criteria will be applied on both tasks:

- 60% for working program — functionality;
- 10% for code architecture — algorithms, data types, control structures, as well as use of internal libraries and/or external packages;
- 10% for coding style — clear logic, clarity in variable/function/class names, and readability;
- 20% for documentation — program comments and user documentation.

**Note: There will be no interviews for this assessment.**

# 4 Submission

There will be NO hard copy submission required for this assignment. You are required to submit your assignment as a `.zip` file named with your Student ID. For example, if your Student ID is `12345678`, you would submit a zipped file named `12345678_A3.zip`. Note that marks will be deducted if this requirement is not strictly complied with.

Your submission is via the assignment submission link on the FIT9133 S1 2018 Moodle site by the deadline specified in Section 1, i.e. **27 May 2018 (Sunday) by 23:55pm**.

## 4.1 Deliverables

Your submission should contain the following documents:

- A completed the assignment cover sheet for online submission available on the FIT9133 S1 2018 Moodle site.
- An user documentation (at least 2 pages but not more than 4 pages) in PDF format with clear and complete instructions on how to run your programs. This should be a well formatted document with headings and sub-headings, and a table of contents should be included. (Note that your programs must at least run on the computers in the University's computer labs. Any submission that does not run accordingly will receive no marks.)
- Electronic copies of ALL your files that are needed to run your programs.

Marks will deducted for any of these requirements that are not strictly complied with.

## 4.2 Academic Integrity: Plagiarism and Collusion

**Plagiarism** means to take and use another person's ideas and or manner of expressing them and to pass them off as your own by failing to give appropriate acknowledgement. This includes materials sourced from the Internet, staff, other students, and from published and unpublished works.

**Collusion** means unauthorised collaboration on assessable work (written, oral, or practical) with other people. This occurs when you present group work as your own or as the work of another person. Collusion may be with another Monash student or with people or students external to the University. This applies to work assessed by Monash or another university.

It is your responsibility to make yourself familiar with the University's policies and procedures in the event of suspected breaches of academic integrity. (Note: Students will be asked to attend an interview should such a situation is detected.)

The University's policies are available at: `http://www.monash.edu/students/academic/policies/academic-integrity`