**1 - INITIAL PREPARATION**

**1.1 – Gather the email data**

* Parse through emails and copy their text
* Paste this text into a .CSV file
* Save these .CSV files in the project folder – one for positive sentiment (pos\_sentiment.csv) and one for negative sentiment (neg\_sentiment.csv)

**1.2 – Split data into 3 sets (testing training, validation)**

* Will likely be done within the code itself (just like how the algorithm was trained in my proof of concept)
  + Will need three variables – test, train, and validate – which are ran against the algorithm to produce the expected result (i.e. training the algorithm, testing it and evaluating its performance)

**1.3 – Implement logging procedure**

* Investigate the use of the **logging** module in Python
* Read/follow some tutorials on implementing this module
  + Logging should occur for various events throughout the program’s lifecycle (i.e. debugging, displaying error/warning messages, etc.)
* Implement the module into my own project with the aid of these tutorials
  + All logging should be carried out & saved to an external file for easy access
* Format the logging output so it is easier to read

**1.4 – Test logging procedure**

* Based off of the code from my initial prototype, ensure that any logging functionality is working as intended

**1.5 – Cleaning and preparation of datasets**

* Similar to what was done in initial prototype:
  1. Give each line of text in the CSV file a value between 0 and 1 to rank its level of emotion, with 0 being incredibly angry/upset and 1 being extremely happy/ecstatic
  2. Tokenize the words in the test data
  3. Split the data into the testing/training/validation datasets as previously discussed
  4. Shuffle the data and split it to make sure the same data isn’t constantly being passed against the algorithm
* Research any other necessary cleaning/preparation methods for the data that weren’t in the proof of concept

**2 – BUILDING THE ALGORITHM**

**2.1 – Code the initial Naïve Bayes algorithm**

* Some coding has already been done in proof of concept; code needs to be double checked to ensure it is robust & covering the functionality needed for the final app
* While a GUI will be implemented later on, initial development and testing of the algorithm will be done via the command line to keep things simple
* Research & potentially follow some extra tutorials to implement any additional functionality

**2.2 – Test the initial Naïve Bayes algorithm**

* Using multiple different shuffled datasets, test the algorithm and evaluate its accuracy overall
* Make a note of these results in my dissertative document as I go along

**2.3 – Validate the initial Naïve Bayes algorithm using F1 scores**

* Using the necessary library/libraries (i.e. sklearn.metrics.f1\_score), calculate the F1 score of the algorithm and display it in the program’s command line output (similar to how the accuracy of the algorithm itself was displayed in the proof of concept)

**2.4 – Colour code the initial algorithm’s output**

* As mentioned, the algorithm’s output will initially be kept to the command line for the sake of development/testing; GUI-based colour coding will be implemented later on once the GUI has been designed
* Some basic colour coding is already in place (i.e. green = positive, red = negative) – however, the final results from the algorithm won’t be so binary
  + Need to implement a colour gradient/scale based on a sentence’s level of emotion
  + Based on the percentage value assigned to each line of text passed into the algorithm, colour-grade the output to be on a scale of red (0) to yellow (0.5) to green (1)
    - Gradient colours will obviously change depending on how high or low the score is, quickly conveying the level of emotion

**2.5 – Test the colour coding of the initial algorithm’s output**

* Run and evaluate the algorithm multiple times to test if the colour coding is working correctly
* Make a note of any important results/occurrences in my dissertative document

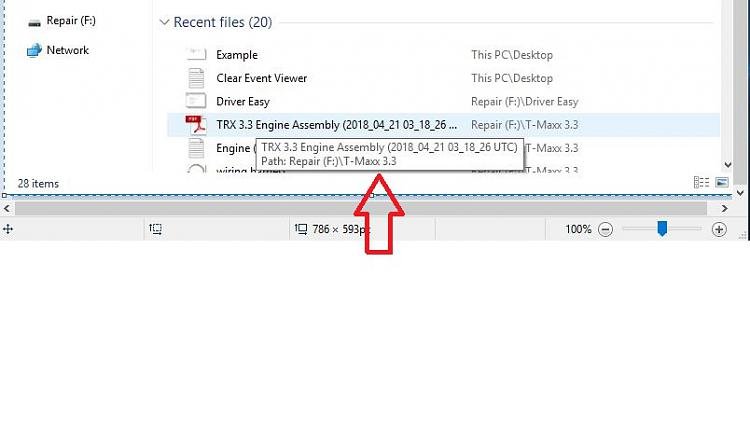
**3 – BUILDING THE SOFTWARE/GUI**

**3.1 – Build the GUI structure & pages**

* Some simple GUI design/testing will have been done at this point, following some basic tutorials based on what was created for the prototype/proof of concept
* Based on initial wireframes contained in the dissertative document, create 2-3 pages for the app’s GUI
* Follow some tutorials to implement the libraries needed to do this (i.e. PySimpleGUI)

**3.2 – Implement extra detail when clicking on a sentence**

* Make it so that when the user hovers over a sentence, their mouse icon changes showing they can click on the sentence to display more detail about its level of emotion
  + Also have pop-up text telling them this, i.e.:



* + Once the user clicks on a sentence, the pane on the left side of the screen will be used to display a detailed breakdown of the emotion in the sentence they have clicked on; it will also show a sample of the sentences in the email that are considered positive/neutral/negative alongside their percent score
  + The original email entered by the user itself will also be colour coded (if possible!)

Graphical user interface, website

Description automatically generated

**3.3 – Implement loading screens where necessary**

* When the user clicks the “analyse text” button upon entering their email, a loading screen should appear to let the user know the program received their feedback & is working to analyse the text
* Research & follow some tutorials/guidelines on how to implement this – may need to use some additional libraries/other features of the library being used to code the GUI itsefl

Graphical user interface, website

Description automatically generated

**3.4 – Implement formal error messaging (via pop-up boxes)**

* Similar to the loading screen, pop-ups will need to be displayed if and when something goes wrong in the code (i.e. an error with the algorithm itself, the user clicking “analyse text” when no text has been entered into the text box, etc.)
* These errors will also be logged accordingly as described earlier on

**3.5 – Ensure deletion of cached text upon exiting the program**

* Use a specific library (i.e. linecache) to cache any text that the user analyses, to speed up the process & prevent the algorithm from analysing the same text multiple times (if the algorithm is accurate, this shouldn’t be necessary)
* If a cache is being used to store any text the user has entered into the program, this will need to be deleted from the program’s source code once the user shuts dow

**3.6 – Implement word limit of 500 words**

* Will likely use the Tkinter text box widget on the program’s main page; this allows for the coding of a text box that can have a word limit
* Limit will be set to 500 words so as not to overwhelm the algorithm

**3.7 – Thoroughly test the overall GUI**

* The GUI will need to be debugged, and its individual features tested as thoroughly as possible
* All testing results/findings will be noted in dissertative document

**3.8 – Implement save/load functionality**

* Once a piece of text has been analysed, the user will be able to save it to a file for easy access later on
  + This will likely be a .txt file or PDF
  + Will need to code the pop-up box allowing the user to choose where they want to save the file to, as well as choosing its name
  + May also need a loading screen for saving the box, as well as a confirmation dialogue prompt when saving has been completed
* The user should also be able to load a compatible file of their choice back into the program and analyse it if they so wish (even if it has not been analysed in the program before)
  + This means there will need to be code to check that the file type is compatible, before displaying the normal loading screen while the algorithm is carrying out its evaluation

**3.9 – Test the save/load functionality**

* This save/load functionality will be tested and debugged, and error pop-ups/logging will be fully implemented wherever necessary

**3.10 – Final testing of overall application & algorithm**

* Once all the GUI and algorithmic elements have been put in place, do a thorough final test of the artefact itself to make sure it is working as intended & any errors are handled smoothly and gracefully

**3.11 – Report findings of project in dissertative document**

* Record testing procedures
* Record results of algorithm’s analyses/how accurate it is overall
* Record the design of the GUI and why design choices were made
* Record the overall findings and conclusions of the project