MA3831 Capstone

Done by:

- **Joshua Timothy Gratio Wibowo (14145466)**
- **Richard Reynard (14170539)**
- **Sally Shue Yan Pang (14174354)**

Table of Contents

1. Overview

In this digital era, internet users are often loaded with tons of data and information. However, a lot of these data go to waste simply due to the time constraint and hence people are not able to make full use of them. Therefore, we aim to help businesses and users to utilise the resources efficiently by providing the essential information to both parties. Our Minimum Viable Product (MVP) aims to solve this exact issue as it aims to efficiently condense large amounts of information into a shorter and more manageable format, while still preserving the most important and relevant information. This can help businesses save time and resources by quickly identifying key information, make better and more informed decisions, and improve their overall productivity and efficiency. Additionally, it can also help businesses to improve customer experience by providing more accessible information and increase their reach by providing more engaging and shareable content.

Furthermore, the MVP will also provide the key information on the summary and condense them in a structured format to improve the readability for the user. Some of the key information that will be presented are named entities (people, organisations, locations), dates, and facts. These can then be further utilised on several tasks such as content curation, sentiment analysis, and knowledge base construction. Businesses can also adopt the MVP into their framework as it may aid them in monitoring the market competition, improve customer relations, identify trends for future growth, automate data entry and refine decision making process. These are all due to the enhanced data processing which increases efficiency while reducing the non-essential costs.

2. Data Extraction

For this task, we drew on our knowledge with data scraping from several news platforms such as CNN, NBC, BBC and so on. Instead of using the more traditional and older method, such as the built-in feature of Beautiful Soup to scrape the data, we moved towards the direction of RSS (Really Simple Syndication), which is a way to automatically collect and extract data from websites that have an RSS feed. It is known as a scraping tool or piece of software that can subscribe to an RSS feed and automatically retrieve new content as it is published, eliminating the need to manually monitor the website for updates.

In this scraping procedure, we have saved 23 URLs to news websites together with the names of the websites in a file called "NewsPapers.json." Each website has a cap of 200 articles that can be downloaded at once. Moving on, for the sake of consistency, we have also included codes to check whether there is no publish date identified, in which case the article would be skipped. The program also has the ability to switch to the next link once 10 articles from the same newspaper have been downloaded in order to prevent duplicate content. After the function has finished its work, we will store all of the articles into a csv file called "news_dataset.csv" that contains the title, authors, text, image, video, link, and publish date of each article.

Data cleaning is one of the crucial steps to take before we move on to develop the summarizer model. As we can see, the dataset's first three rows contain important, brief text, including the following: "1. How relevant is this advertisement to you?...". To get rid of this, we make the assumption that any text that is fewer than 200 words is an advertisement, which led to 1134 articles out of 2983 being advertisements. Additionally, we have verified that every article is in a neat paragraph without extra space that could have occurred during the scraping process. Furthermore, duplicate rows that the program might have missed during the scraping process as well as rows that might contain advertisements have been effectively removed using the function in the cleaning data process. This will complete the data cleaning procedure and write the results into a new csv file (“new_dataset.csv").

Moreover, along with sentiment analysis, a natural language processing (NLP) approach, we are able to determine the positivity, negativity, and neutrality of the data before proceeding to the model. Textual data is frequently subjected to sentiment analysis in order to assist businesses in tracking the tone of client reviews of their brands and products and comprehending their needs. In this instance, we are able to determine whether the majority of the scraped article was positive, negative, or neutral. The sentiment of the text as a whole is captured by the average compound score. Our findings indicate that the text appears to be largely positive based on the average compound score.

3. NLP Task

Sentiment Analysis

Our first NLP task is sentiment analysis (SA), in which we analyse the wordings of the articles to determine if they are written with positive or negative tone. Medhat, W. et. al (2014) states that SA is "the computational treatment of opinions, sentiments and subjectivity of text." SA will then look for the opinions in the text, determine the emotions they convey, then categorise their polarity.

```
import pandas as pd
from nltk.sentiment import SentimentIntensityAnalyzer
nltk.download('vader_lexicon')

# Select the column you want to analyze
text_column = df['Text']

initial = SentimentIntensityAnalyzer()

# Analyze the sentiment for each text in the column
for text in text_column:
    sentiment = initial.polarity_scores(text)
    print(sentiment)
```

Figure 1. Sentiment Analysis Code

We use the pre-built SentimentIntensityAnalyzer function from the NLTK library to analyse the sentiment for each text in the dataframe's column. It will then display the sentiment for each of the articles and we can interpret the result relatively easily. This will be especially beneficial for business owners who would like to analyse the type of content they are aiming for and what their demographic are interested in. Hence, this is a good addition to the next 2 tasks as it will enhance the value of the project to be more appealing to potential businesses.

Text Summarisation

In the recent decades, the internet penetration rate has increased significantly, especially after the year 2000 as seen from the graph below (Our World In Data, 2015).

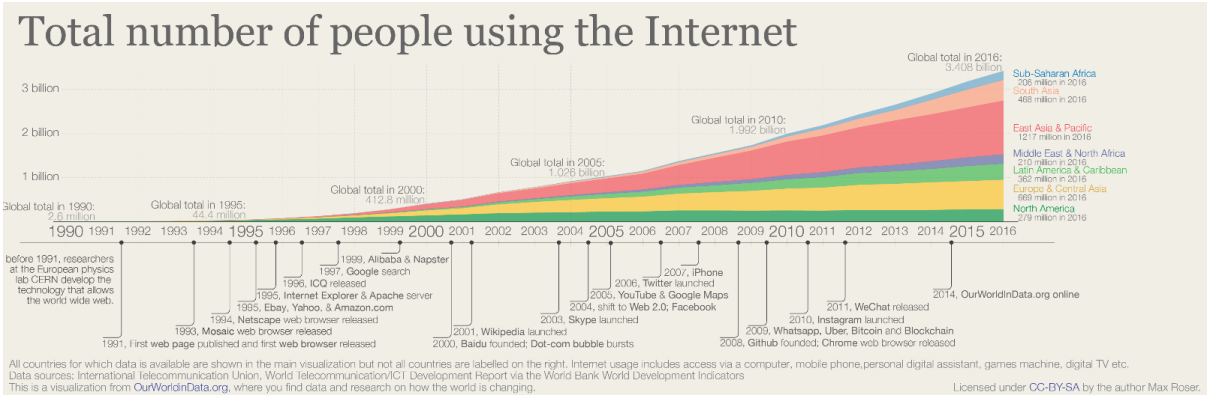

Figure 2. Internet Penetration Rate (Roser, M., et. al., 2015)

As a result of the boom, many hard printed information have migrated to the internet to maintain their viewership. Consequently, users of the internet are frequently loaded with information from several sources. However, due to the time constraint, these valuable information might go to waste before they catch the viewers' attention. Therefore, our NLP task on text summarisation would like to address this issue.

Rathi et. al. in their study "A Review of State-Of-The-Art Automatic Text Summarisation", state that text summarisation is the act of transforming a lengthy sentence into a brief and condensed version. There are several text summarisation methods - extractive, abstractive and hybrid. Extractive text summarisation selects the relevant sentences from the original texts and compiles them into a new sentence, while the abstractive method transforms the inputs into a representative code and uses them to generate a sentence that captures the essence of the input. The hybrid method combines both these characteristics, but leans more towards the extractive method. While the abstractive method is able to produce a more comprehensive result, the extractive method is easier to implement and more efficient and hence will be implemented in our NLP task.

News publishers today usually scatter several subjects into the same page, and combined with the notifications and other distractions, humans' attention span is shorter than ever (Lorenz-Spreen et. al.). This then results in lower engagement rate between the publishers and the users, which affect their revenue stream as with lower viewership there will also be less sponsorship and advertisement.

For our project, we will use the extractive method because it requires less computing power, is not too complex, and more efficient to train and run than abstractive summarisation. Extractive summarization is also generally considered to be more reliable and accurate than abstractive summarization, because it only selects important sentences from the original text, rather than generating new sentences. Since it only selects already existing phrases and sentences, it tends to be less prone to errors and is more easily verifiable by comparing the summary to the original text. Our algorithm will be similar to the TextRank algorithm, which is a graph-based algorithm that ranks sentences based on their importance within the text.

```
def read_article_text(article_id):
    article = df['Text'][article_id].split(". ")
    sentences = []

    for sentence in article:
        sentences.append(sentence.replace("[^a-zA-Z]", " ").split(" "))
    sentences.pop()

    return sentences
```

Figure 3. Data Preprocessing Code

To do it, we first need to pre-process the data. As we can see from the code on the red box, After filtering from the web scraping is finished, we perform preprocessing once more. This function is given the 'Text' Id. Text is taken from the DataFrame's "Text" column and broken up into sentences with periods. The sentence list is filled in and all spaces are used to substitute non-characters and sentences will be a list of all tokenized sentences.

In order to develop a summarizer, we will generate a matrix containing NxN zeros, where N is the number of sentences. Next, determine whether the two sentences are similar or otherwise, then send both sentences to the sentence similarity function. The matrix is updated based on the sentence index position and the cosine similarity calculated by the sentence similarity function. Therefore fill up the entire NxN similarity matrix. The quantity of unique tokens in all words is divided into two vectors, each of which is filled with 0. Update the token index in Vector1 and Vector2 after searching for each token in the list of all words. Lastly, update the similarity values for two sentences by returning the Cosine distance and returning back to the previous function.

Bellow are the codes:

```
def create_similarity_matrix(sentences, stop_words):

    # Create an empty similarity matrix
    similarity_matrix = np.zeros((len(sentences), len(sentences)))

    for index1 in range(len(sentences)):

        for index2 in range(len(sentences)):
            if index1 == index2: #ignore if both are same sentences

                continue

            similarity_matrix[index1][index2] = sentence_similarity(sentences[index1], sentences[index2], stop_words)


    return similarity_matrix
```

Figure 4. Similarity Matrix Function

```
def sentence_similarity(sent1, sent2, stopwords=None):
    if stopwords is None:
        stopwords = []

    sent1 = [w.lower() for w in sent1]
    sent2 = [w.lower() for w in sent2]

    all_words = list(set(sent1 + sent2))

    vector1 = [0] * len(all_words)
    vector2 = [0] * len(all_words)

    # build the vector for the first sentence
    for w in sent1:
        if w in stopwords:
            continue

        vector1[all_words.index(w)] += 1


    # build the vector for the second sentence
    for w in sent2:
        if w in stopwords:
            continue
        vector2[all_words.index(w)] += 1


    return (1 - cosine_distance(vector1, vector2))
```

Figure 5. Vectoring Words

We also put some preprocessing code in our sentence_similarity function (the codes in red boxes), where it lowers all the characters and also skip the stop words when building the vector form of a sentence. The reason why we don't do all of this outside of the function is because we want the output (the summary) of the code to be similar to the original news text, which includes capital letters, stop words, and punctuations.

We then make the text_summary function to assist in creating the text summary by using the Text id and number of top sentences as arguments. Then pass on to the other functions, such as read_article_text and sentence_similarity_martix. It also produces the sentence similarity graph using a sentence similarity matrix. Scores are computed using the "pagerank" algorithm. Moving on, we will then sort to rank them and combine the best three sentences in the rankings and show the summarised text. Below is the code for the text_summary function.

```python
def text_summary(article_id, top_n=3):
   # nltk.download("stopwords")
    stop_words = stopwords.words('english')
    summarize_text = []

    try:
        # Step 1 - Read text and split it to setences.
        sentences =  read_article_text(article_id)

        # Step 2 - Generate Similary Martix across sentences
        sentence_similarity_martix = create_similarity_matrix(sentences, stop_words)

        # Step 3 - Rank sentences in similarity martix
        sentence_similarity_graph = nx.from_numpy_array(sentence_similarity_martix)

        scores = nx.pagerank(sentence_similarity_graph) # PageRank (PR) is an algorithm used by Google Search to rank websites in their search engine
       # score is dictionary with key = node and value is its rank

        # creating the graph for similarity matrix.
        nx.draw(sentence_similarity_graph,with_labels=True)

        # Step 4 - Sort the rank and pick top sentences
        ranked_sentence = sorted(((scores[i],s) for i,s in enumerate(sentences)), reverse=True)
        # ranked_sentences are the list sorted in descending order sentences with their rank.

        for i in range(top_n):
          summarize_text.append(" ".join(ranked_sentence[i][1]))

        # Step 5 - Offcourse, output the summarize text
        print("Summarize Text: \n", ". ".join(summarize_text) + ".")

        # Step 6 writing the summarize text into a file
        with open('summary.txt', 'w') as f:
            for text in summarize_text:
                text = text.encode('utf-8')
                print(text, file=f)
        # Step 7 generating word cloud of summarize text
        word_cloud_generate('./summary.txt')
    except IndexError:
        print("This article is not in English language")
```

Figure 6. Text Summary Code

Information Extraction

To further improve the feature of our MVP, we integrated the function of Information Extraction (IE), which is defined as the act of derivation of detailed information from passages (Suresh et. al.). Despite the summarised text having captured the main key points of the original passage, incorporating IE into the system will enhance the usability as it can group the named entities into their categories and hence can be capitalised for a more focused and well targeted audience. This will then be able to entice the users to stay on the platform reading the articles that they are interested in while at the same time ensuring that the publishers are receiving a stable revenue stream.

Our NLP task will include the extraction of key information such as people or organisations, and relations between those entities, such as "employee of" or "located in." IE is a key component of natural language processing (NLP) and is often used in a wide range of applications, such as question answering, text summarization, and knowledge base construction. Information extraction is an active research area in NLP, with many recent advances in deep learning-based models.

We choose to combine information extraction with text summarisation, because it makes it possible to automatically process and understand large amounts of unstructured text data, which can save time and effort compared to manual processing, and can also uncover insights that would be difficult or impossible to find manually.

```
nlp = spacy.load("en_core_web_sm")

# Load txt file to read the summarised text
with open('/Users/sallypang/Library/CloudStorage/OneDrive-JamesCookUniversity/MA 3831/MA 3831 - Assessment02/summary.txt', 'r') as file:
    article = file.read()

doc = nlp(article)

for ent in doc.ents:
    print('{} - {}'.format(ent.text, ent.label_))
print(f"Number of Entities: {len(doc.ents)} - {doc.ents}")
```

```
Hyun Yang - PERSON
Epigenetic - NORP
three - CARDINAL
worse,\xe2\x80\x9d Sinclair - ORG
Sinclair - ORG
Number of Entities: 5 - (Hyun Yang, Epigenetic, three, worse,\xe2\x80\x9d Sinclair, Sinclair)
```

Figure 7. Information Extraction Code

To do IE, we will use the spacy library. As we can see from the code above, we first load in the English language pipeline that has been pre trained and is ready to use. We then feed in the text to that pipeline and then extract the entities that are in that text. From the output, we can see that the text summary has 5 entities, which can be used to further shorten the reading time needed to get the important information. For example, a busy businessman can just read the title of this news and then read the entities that are in this news to understand what the news is trying to say, if not then he can continue on reading the summary to get a better understanding.

4. Result

There are many parameters against which we can evaluate the summarization system. However, in our studies, we have come to a conclusion to utilise the ChatGPT (Generative Pre-trained Transformer), a potent tool for chatbots and other conversational AI applications, which it has also fall into the top 2 rank of 'The 17 Best AI Chatbots for Business in 2023 and Beyond' referring to the figure below.

Best AI Chatbots for 2023

Rank	AI Chatbot	Rating (Out of 5 Stars)
1.	Netomi	5
2.	ChatGPT	4.85
3.	atSpoke	4.8
4.	WP-Chatbot	4.7
5.	Microsoft Bot Framework	4.6

Figure 8. Best AI Chatbox (Max, D., 2023)

Based on this well-trained NLP model, we are able to determine the accuracy of our output. Through this we have achieved a highest score of 80 including the explanation of the score indicating the screenshots of the result.

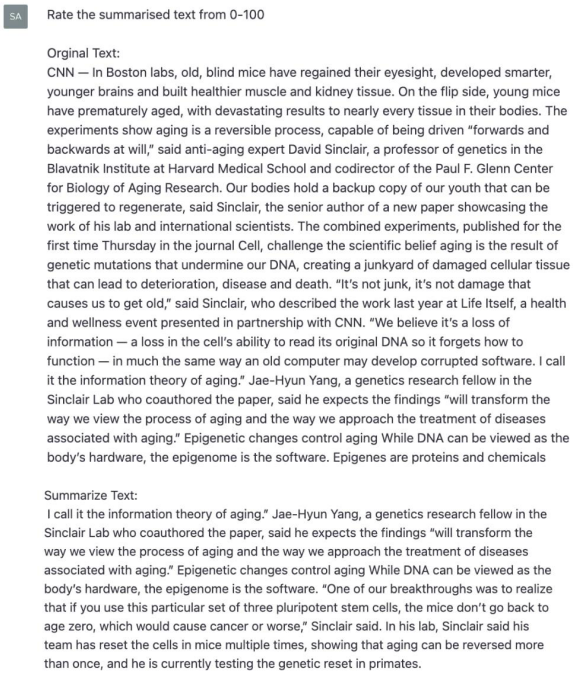

Figure 9. Sample Text and Summarised Text (ChatGPT, Accessed on 16 January 2023)

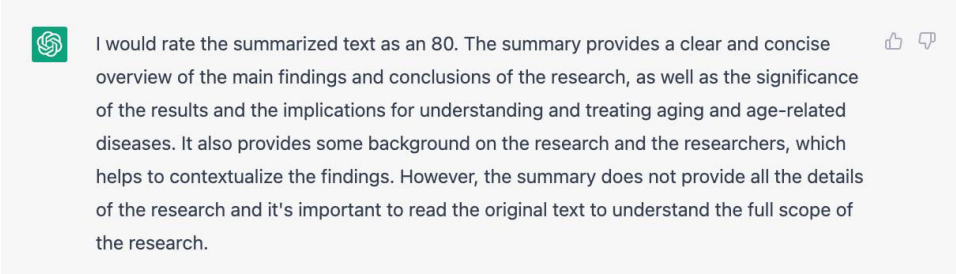

Figure 10. Rating for Summarised Text (ChatGPT, Accessed on 16 January 2023)

According to the model, the output is a "clear and concise overview of the main findings and conclusions of the research," which is essentially what a summarizer should provide readers to save time while also not leaving out any information. It also stated that not all of the study's specifics were provided by the model, which resulted in a score of 80.

Furthermore, because extractive methods are frequently used to extract and combine words together, it is very common for distinct sentences to lack a smooth transition between ideas, concepts, or themes. Particularly when numerous sentences are merged to produce a summary, it may not always have perfect grammar because it may have certain flaws like subject-verb agreement, missing conjunctions, or any other grammatical mistakes. Utilising Grammarly, an American cloud-based typing assistant, we were able to verify these issues. It examines problems in English writings' spelling, grammar, punctuation, clarity, engagement, and delivery, finds instances of plagiarism, and offers corrections for the mistakes found. Based on the content that has been summarised above, we have received a performance score of 93 for this project. It also stated that using The New York Times, the readability of the summarised text will be able to be understood by readers who are at least 16 years old. This is considered as a beneficial improvement as most articles are typically difficult for children to comprehend by children.

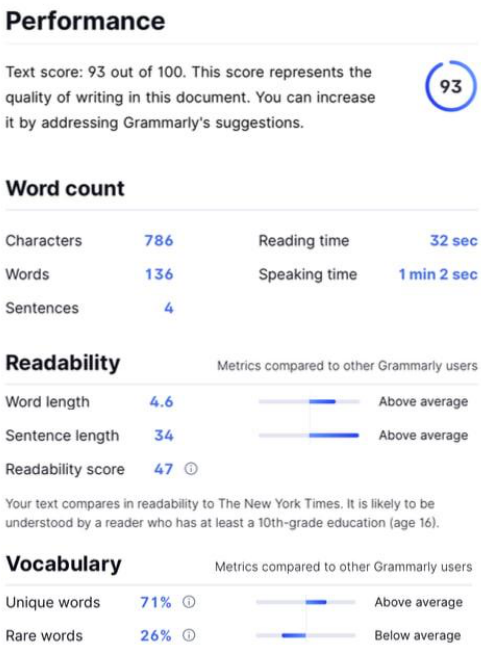

Figure 11. Rating for Readability of Summarised Text (Grammarly, Accessed on 16 January 2023)

Last but not least, just like any other studies, ours have limitations as well, such as the inability to summarise non-English texts and the accuracy of our information extraction model. To advance this project, we would like to train other languages and offer reduced readability to increase the number of readers in lower grades, such as upper primary school pupils in our future project. Through this, they will be able to digest more information and gain more knowledge in a shorter amount of time.

5. References

Rathi, K., Raj, S., Mohan, S., & Singh, Y. V. (2022). A Review of State-Of-The-Art Automatic Text Summarisation. *International Journal of Creative Research Thoughts (2022)*.

Roser, M., Ritchie, H., & Ortiz-Ospina, E. (2015) - "Internet". Published online at OurWorldInData.org. Retrieved from: 'https://ourworldindata.org/internet' [Online Resource]

Suresh, Y., & Manusha Reddy, A. (2021). A contextual model for information extraction in resume analytics using NLP's spacy. In *Inventive Computation and Information Technologies*(pp. 395-404). Springer, Singapore.

Lorenz-Spreen, P., Mønsted, B. M., Hövel, P., & Lehmann, S. (2019). Accelerating dynamics of collective attention. *Nature communications*, *10*(1), 1-9.

Max, D. (2023). The 17 Best AI Chatbots for Business in 2023 and Beyond [Review and Key Features]. *Netomi*. Retrieved from: 'https://www.netomi.com/best-ai-chatbot' on 16 January 2023.

ChatGPT. https://openai.com/blog/chatgpt/. Accessed on 16 January 2023.

Grammarly. https://www.grammarly.com/. Accessed on 16 January 2023.

Medhat, W., Hassan, A., & Korashy, H. (2014). Sentiment analysis algorithms and applications: A survey. *Ain Shams engineering journal*, *5*(4), 1093-1113.