Configuration Manual

Media Content Analysis of Covid-19 Virus Using Natural Language Processing Techniques

MSc Research Project
Data Analytics

Anaelle Rouxel
Student ID: X15022421

School of Computing
National College of Ireland

Supervisor: Catherine Mulwa

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Anaelle Rouxel……………………………………………………………………………….…………
Student ID:	X15022421…………………………………………………………………..…….……………
Programme:	Master of Science in Data Analytics **Year:** 2020….……….. ………………………………………………………….……………
Module:	Research Project……………………………………………………………………….………
Lecturer:	Catherine Mulwa………………………………………………………….………….………
Submission Due Date:	17th August 2020.……………………………………………………….…………..………
Project Title:	Media Content Analysis of Covid-19 Virus Using Natural Language Processing Techniques …………………..……………………………….…………….………
Word Count:	…9,402…………………………… **Page Count:** …83 pages………………….…….………

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: …Anaelle Rouxel ………………………………………………………………………………

Date: …17th August 2020 …………………………………………………………………………

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Anaelle Rouxel
Student ID: X15022421

1 Introduction

This configuration manual describes the hardware and software configurations, including the installation process for each tool used for the research. The detailed steps undertaken as part of the research project for Media Content Analysis of Covid-19 Virus Using Natural Language Processing Techniques are presented. First, the datasets creation will be explained. The following section presents the pre-processing steps in order to obtain the data format desired for analysis. The next part consists in detailing the techniques implemented on the data for exploratory analysis on the cleaned text data and also for modelling. To finish, the last two sections presents the extra implementation carried out but not successful and the additional literature research conducted.

2 Hardware Configuration

All the project research, data preparation and analysis were carried out on a Dell Inspiron 5570 laptop and 64-bit Windows Operating System. The specifications are detailed in Figure 1.

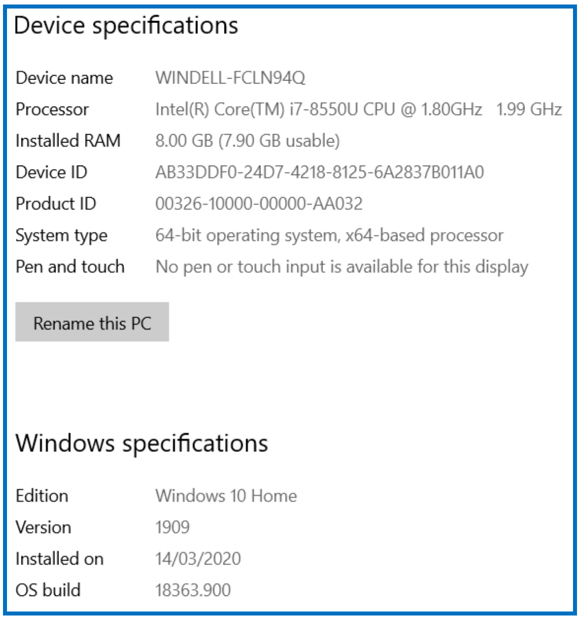


Figure 1: Machine specifications

3 Software Configuration

The laptop uses Windows 10 Operating System and MS Office package. The programmes used as part of this package are related to word processing (Word), spreadsheets (Excel), presentation and diagram flow designs (PowerPoint).
Additional software and tool were installed for the project:

- Python version 3.7.4 for data processing and machine learning applications,
- Anaconda for package management and deployment, it comes with Anaconda Navigator (a desktop graphical user interface (GUI) included in Anaconda distribution),
- JupyterLab shell for coding,
- R version 3.6.1 and RStudio for data processing, computation, graphs,
- Tableau for data visualisation,
- Excel for visualising data spreadsheet and designing diagrams for the technical report and configuration manual.

3.1 Python

The version installed on the machine is Python 3.7.4 as confirmed in Fig.2. It was already installed before the start of the project and not re-installed.

```
(base) C:\Users\anael>python -V
Python 3.7.4
```

Figure 2: Python version installed

1. Go to Python website[1] to download the version wanted (Fig.3).

Looking for a specific release?

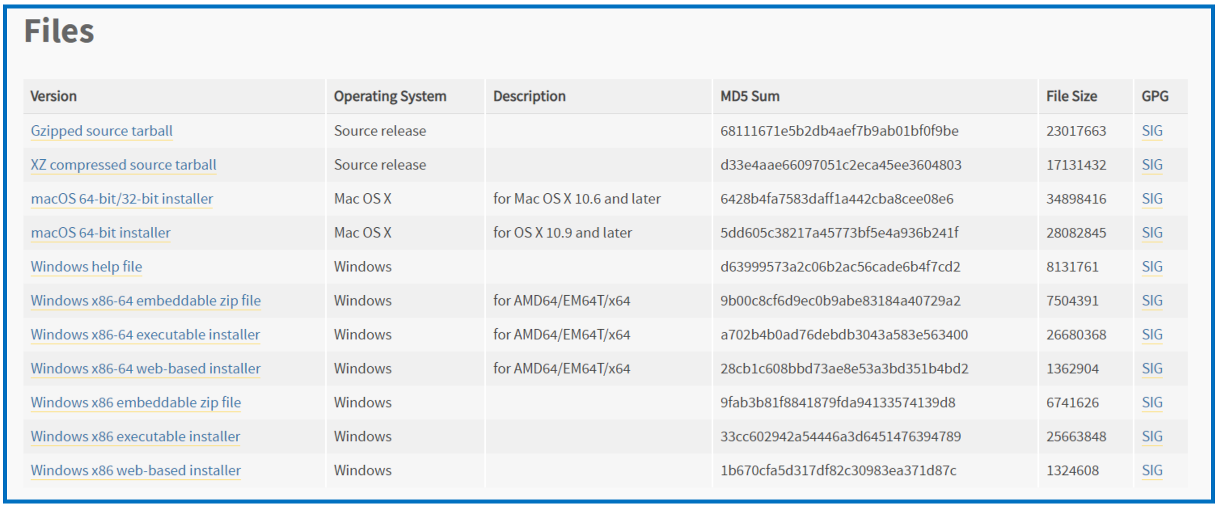
Python releases by version number:

Release version	Release date		Click for more
Python 3.6.10	Dec. 18, 2019	Download	Release Notes
Python 3.5.9	Nov. 2, 2019	Download	Release Notes
Python 3.5.8	Oct. 29, 2019	Download	Release Notes
Python 2.7.17	Oct. 19, 2019	Download	Release Notes
Python 3.7.5	Oct. 15, 2019	Download	Release Notes
Python 3.8.0	Oct. 14, 2019	Download	Release Notes
Python 3.7.4	July 8, 2019	Download	Release Notes

Figure 3: Select the version to download

2. We are directed to another webpage[2] and must scroll down to Files section.
3. Click on Windows x86-64 executable installer as shown on Fig. 4. This is for Python 64 bit installer.

[1] https://www.python.org/downloads/
[2] https://www.python.org/downloads/release/python-374/



Files

Version	Operating System	Description	MD5 Sum	File Size	GPG
Gzipped source tarball	Source release		68111671e5b2db4aef7b9ab01bf0f9be	23017663	SIG
XZ compressed source tarball	Source release		d33e4aae66097051c2eca45ee3604803	17131432	SIG
macOS 64-bit/32-bit installer	Mac OS X	for Mac OS X 10.6 and later	6428b4fa7583daff1a442cba8cee08e6	34898416	SIG
macOS 64-bit installer	Mac OS X	for OS X 10.9 and later	5dd605c38217a45773bf5e4a936b241f	28082845	SIG
Windows help file	Windows		d63999573a2c06b2ac56cade6b4f7cd2	8131761	SIG
Windows x86-64 embeddable zip file	Windows	for AMD64/EM64T/x64	9b00c8cf6d9ec0b9abe83184a40729a2	7504391	SIG
Windows x86-64 executable installer	Windows	for AMD64/EM64T/x64	a702b4b0ad76debdb3043a583e563400	26680368	SIG
Windows x86-64 web-based installer	Windows	for AMD64/EM64T/x64	28cb1c608bbd73ae8e53a3bd351b4bd2	1362904	SIG
Windows x86 embeddable zip file	Windows		9fab3b81f8841879fda94133574139d8	6741626	SIG
Windows x86 executable installer	Windows		33cc602942a54446a3d6451476394789	25663848	SIG
Windows x86 web-based installer	Windows		1b670cfa5d317df82c30983ea371d87c	1324608	SIG

Figure 4: Select Windows x86-64 executable

4. Open the installer file downloaded to execute the installation (Fig.5).

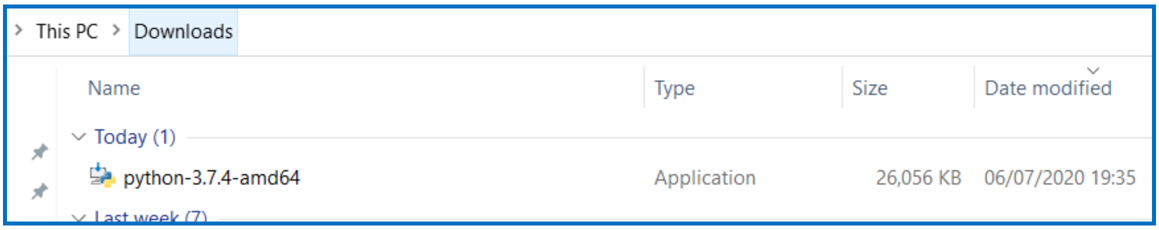


Figure 5: Execute the downloaded file

Instructions continue by showing screenshots taken from a tutorial[3] of the installation process.

5. Tick the Install launcher for all users and Add Pyhton 3.7 to PATH.
6. Click on Install Now as shown in Fig. 6.

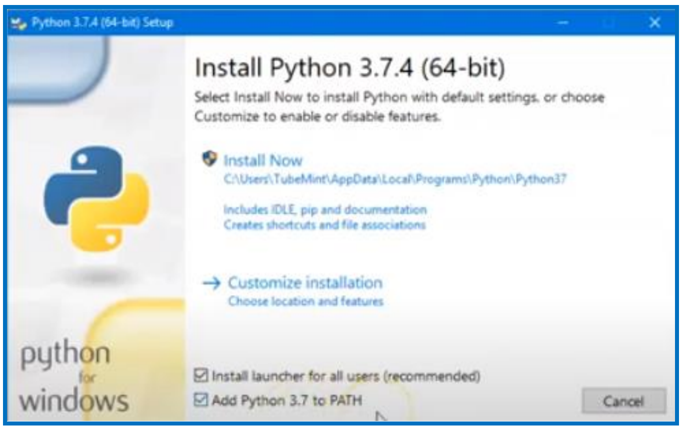


Figure 6: Tick Install launcher box and Install Now

[3] Screenshots taken from https://tubemint.com/download-install-python-3-7-4-on-windows-10/

7. Wait for the Setup to complete

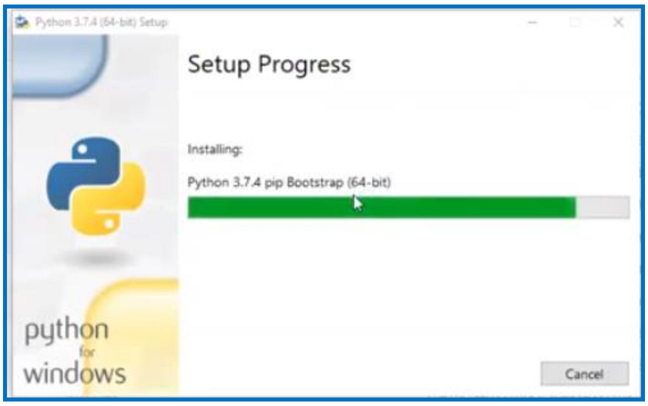


Figure 7: Installation in progress

8. Once the setup is completed, click on Close button seen in Fig. 8.

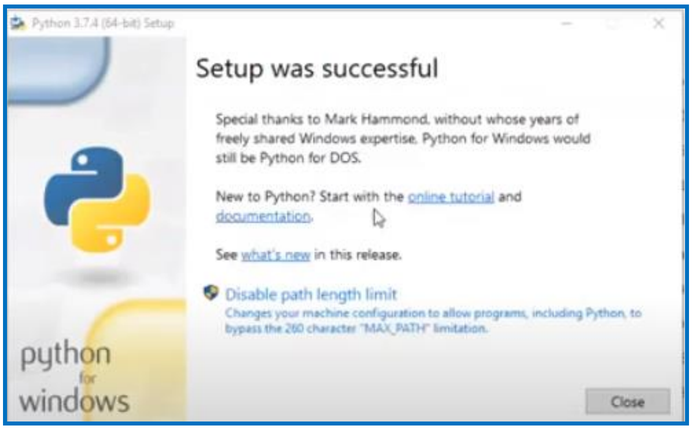


Figure 8: Close the window

9. To verify Python is successfully installed on the machine, open Anaconda Prompt and type "python". It shows in this case Fig. 9 the version 3.7.4 was installed in August 2019 on my machine.

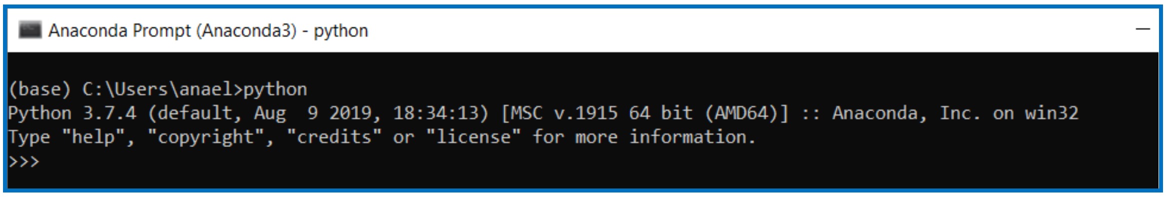


Figure 9: Verify installation is successful

3.2 Anaconda

Anaconda is a distribution software which is composed of Anaconda Navigator and Conda, a package manager that is a program to update and install various packages.

1. Download Anaconda from the official website[4] by clicking on Windows 64-Bit Graphical Installer at the very bottom of the screen as shown in Fig. 10. The latest version of Python is 3.8 at the time of this Configuration Manual.

Figure 10: Select Anaconda Installer

Installing Anaconda on Windows is explained step by step in the official documentation[5]. Anaconda is used to install Python packages, the list of packages required for this project is in Table 1.

Table 1: List of Python packages used

Python Package	Description	Version
beautifulsoup4	to scrape information from web pages	4.8.0
datetime		
gensim	for vector space modelling, topic modelling, document indexing and similarity retrieval with large corpora	3.8.3
keras		2.4.3
math		
matplotlib	data analysis and numerical plotting	3.1.1
nltk	natural language processing	3.4.5
numpy	Scientific computing	1.16.5
os		
Pandas	Data structures and data anslysis	0.25.1
pprint		
pyLDAvis	for interactive topic model visualization	2.1.2
re	Regular expressions	
scikit-learn		0.21.3
seaborn		0.9.0
spacy		2.3.0
string		
textblob	processing textual data, natural language processing tasks (part-of-speech tagging, sentiment analysis)	0.15.3
tweet-preprocessor	Pre-processing library designed for tweet data	0.6.0
twitterscraper	Tool for scraping Tweets	1.4.0
wordcloud		1.7.0

[4] https://www.anaconda.com/products/individual
[5] https://docs.anaconda.com/anaconda/install/windows/

3.3 Jupyter Lab

JupyterLab is an Integrated Development Environment (IDE) for Python language. It is accessible through Anaconda Navigator.

1. In Windows search box, type “Anaconda” and open Anaconda Navigator.

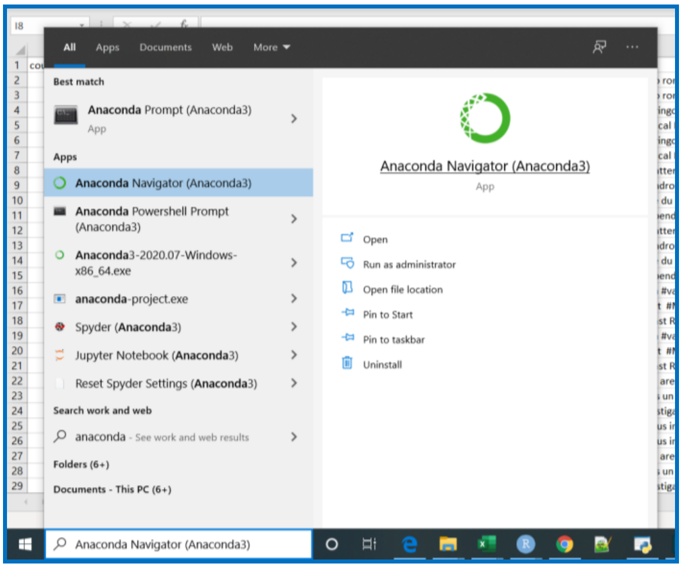


Figure 11: Open Anaconda Navigator

2. On the homepage, launch JupyterLab. Other tools are available as seen from Fig. 12.

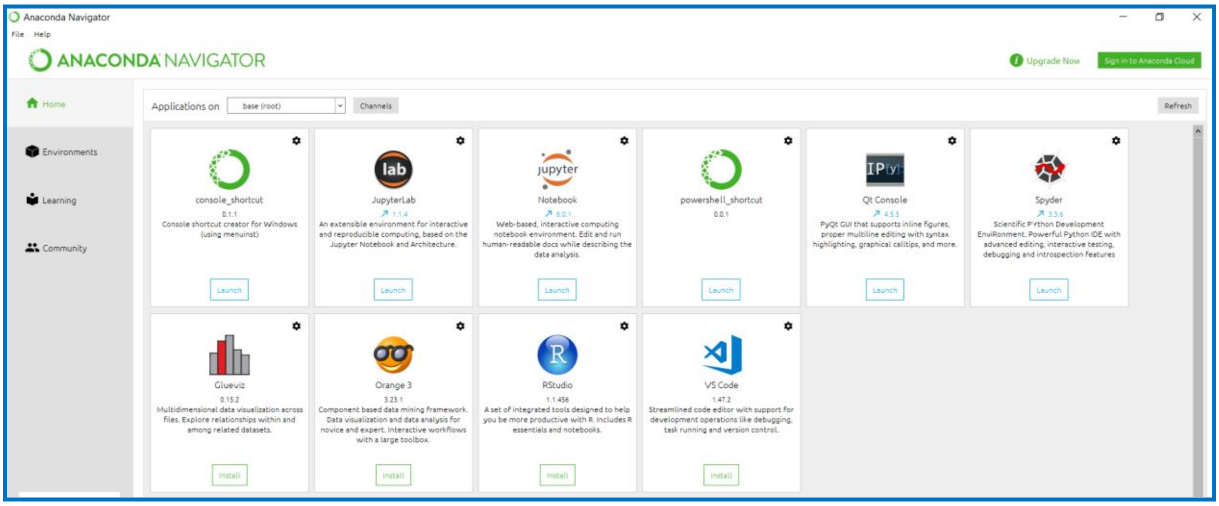


Figure 12: Launch JupyterLab

3. JupyterLab opens in the browser with URL ‘http://localhost:8888’ (8888 is a default port number) (Fig.13).

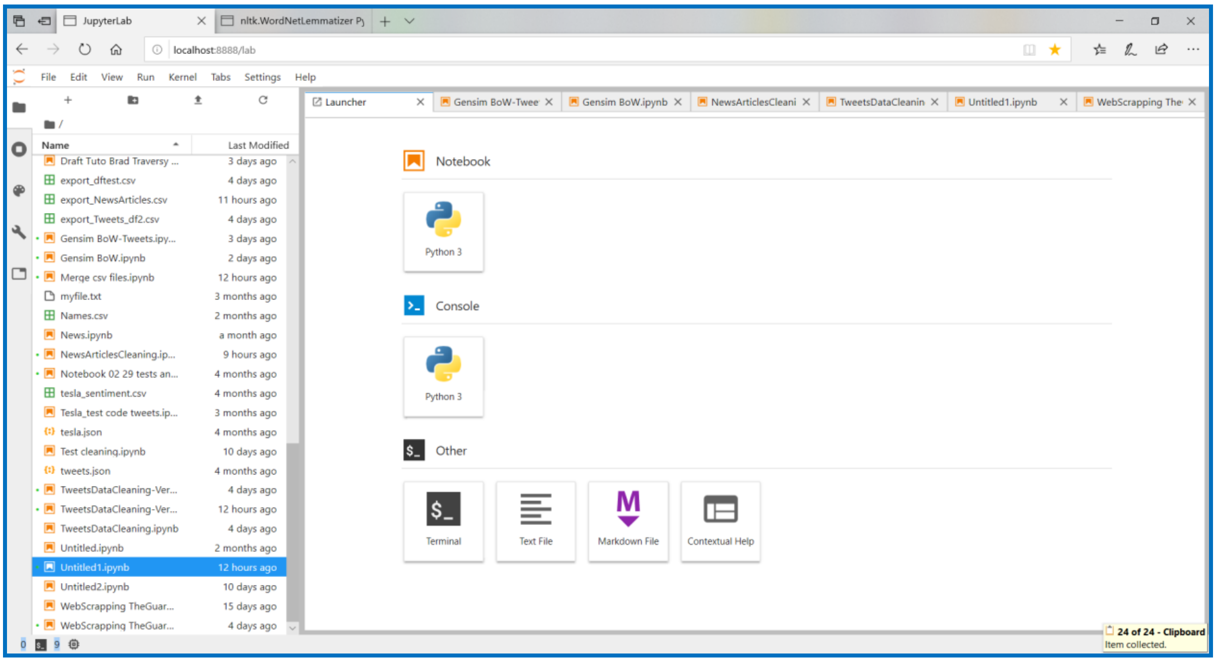


Figure 13: JupyterLab homepage

4. From this page, new Notebooks can be created to write code. Files are saved locally on the machine as seen in Fig. 14.

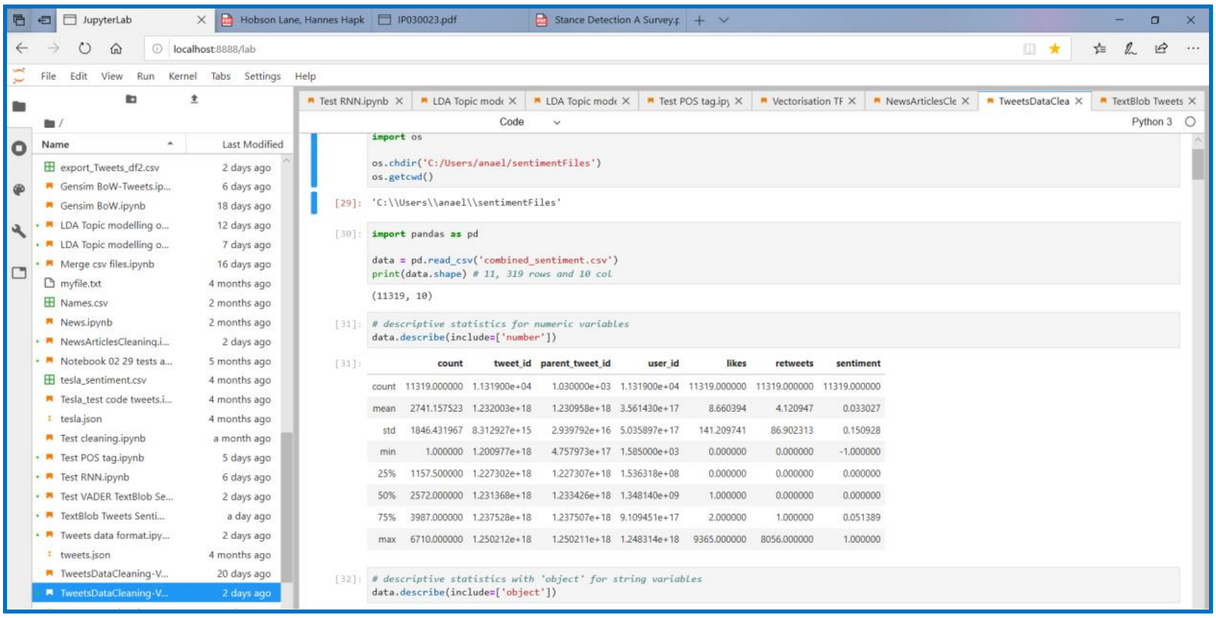


Figure 14: Notebooks examples

3.4 R

R and RStudio were used for data processing, computations, analysis and graphs. The version 3.6.1 was already installed on the machine prior to the project and not re-installed or upgraded entirely (Fig.15). Some packages were updated for the purpose of the project and the procedure will be explained where relevant and implemented. The exhaustive list of packages required is presented in the next section showing RStudio installation.

```
> version
               _
platform       x86_64-w64-mingw32
arch           x86_64
os             mingw32
system         x86_64, mingw32
status
major          3
minor          6.1
year           2019
month          07
day            05
svn rev        76782
language       R
version.string R version 3.6.1 (2019-07-05)
nickname       Action of the Toes
```

Figure 15: R version details

1. Go to the CRAN R project website[6] to download R and elect Download for Windows.

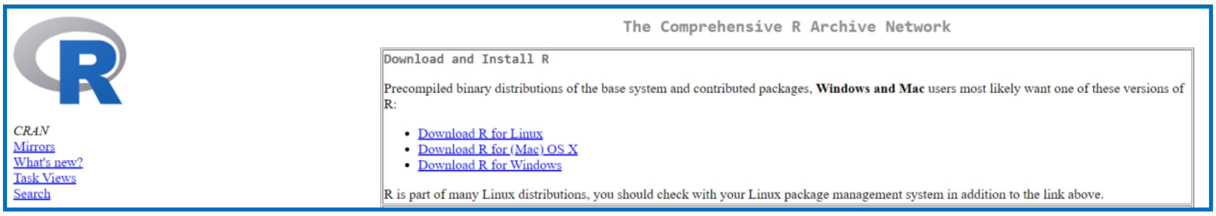


Figure 16: Download R for Windows

2. Click on the link to Install R for the first time

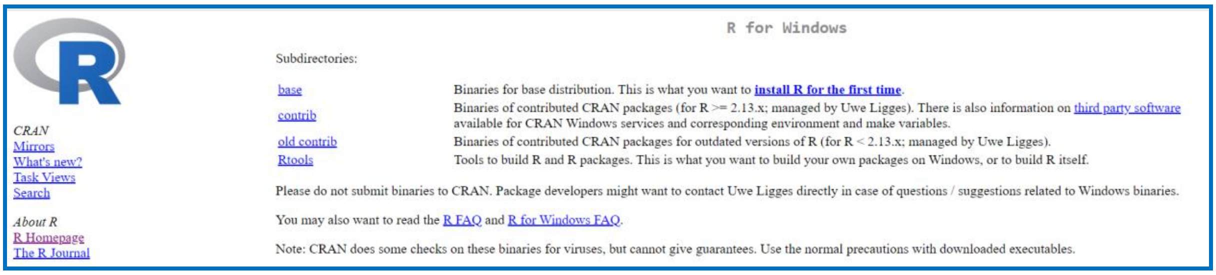


Figure 17: Install R for the first time

[6] https://cran.rstudio.com/

3. The latest version available since June 2020 is R 4.0.2. The version used for the project is 3.6.1 and is available through the link “Previous releases”.

Figure 18: Go to Previous releases

4. Click on the version wanted (Fig.19).

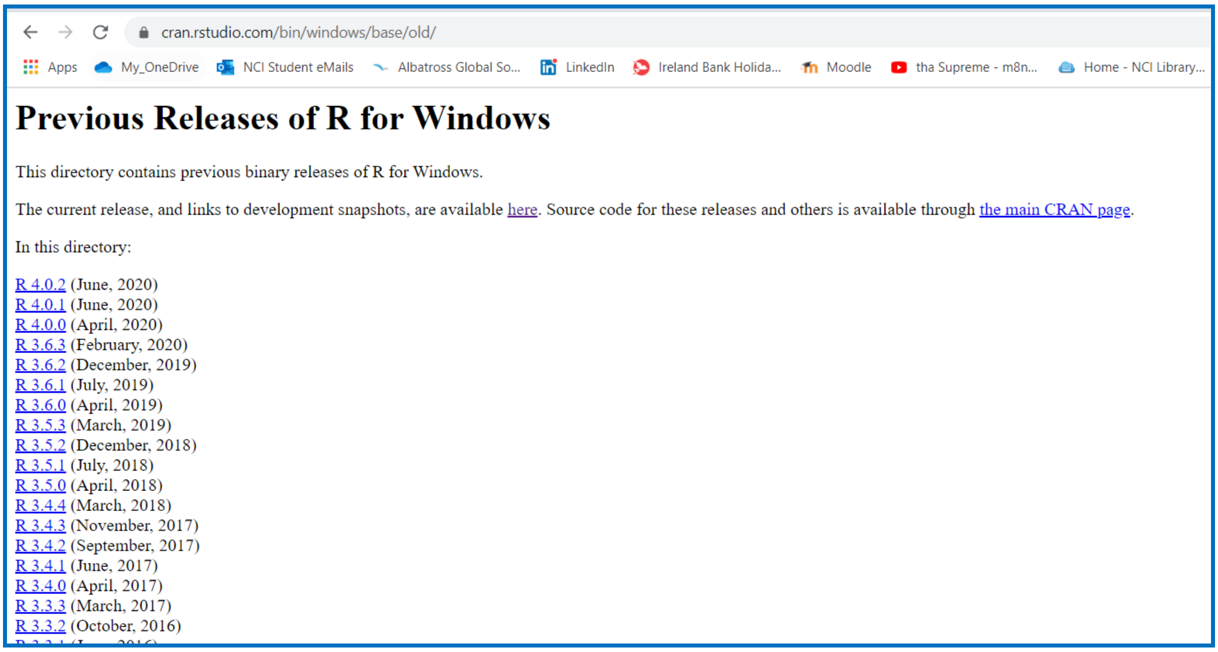


Figure 19: Download R 3.6.1

5. Click on the hyperlink Download R 3.6.1 for Windows on at the top of the screen.

Figure 20: Download R 3.6.1 for Windows

The following instructions and screenshots were taken from a tutorial for beginners prepared by the Techvidvan Team (2020)[7].

6. Open the zip folder downloaded to execute the installer file. Select the language and click on OK (Fig.21).

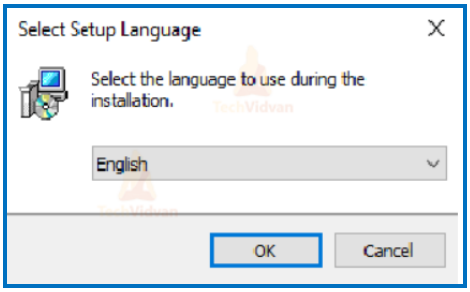


Figure 21: Select the language

7. Read and accept the terms of the licence agreement, click on Next (Fig.22).

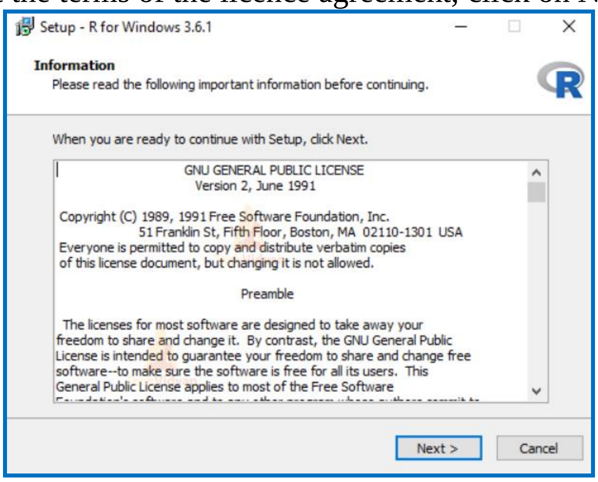


Figure 22: License agreement terms

[7] https://techvidvan.com/tutorials/install-r/#install-r-windows

8. Select all the components to install and click on Next (Fig.23).

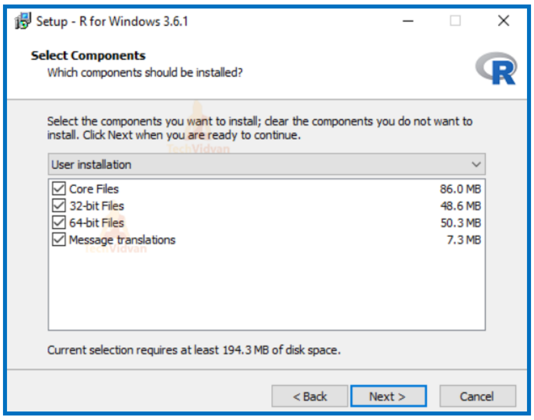


Figure 23: Select all components

9. Enter the path to the location where you want to install R on the machine and click on Next.

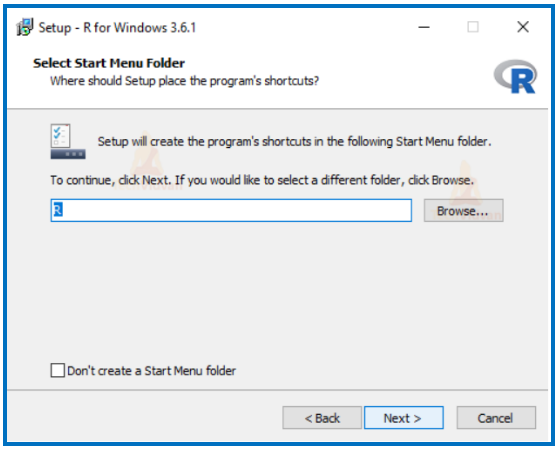


Figure 24: Enter the path

10. Select the desired additional tasks as shown in Fig. 25 and click on Next.

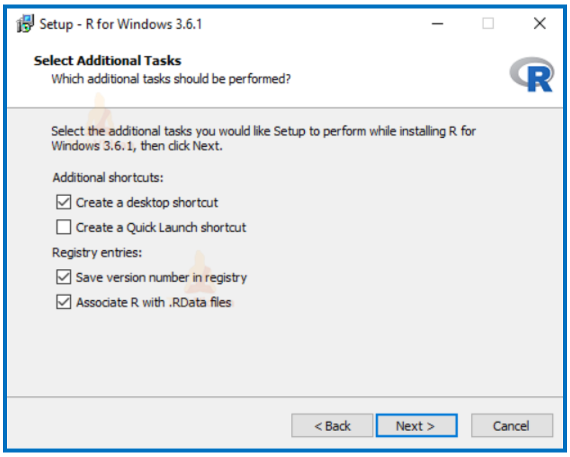


Figure 25: Select Additional Tasks

11. Wait until the installation is completed.

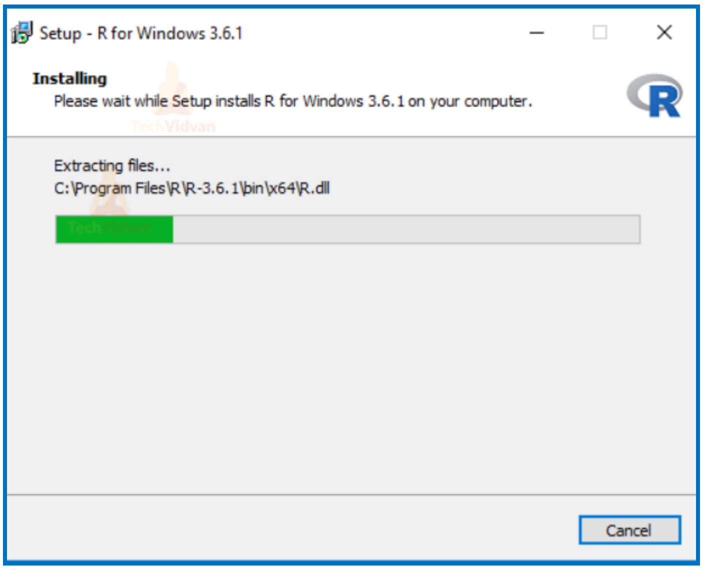


Figure 26: Installation in progress

12. Click on Finish on the installation setup wizard to complete the process (Fig.27).

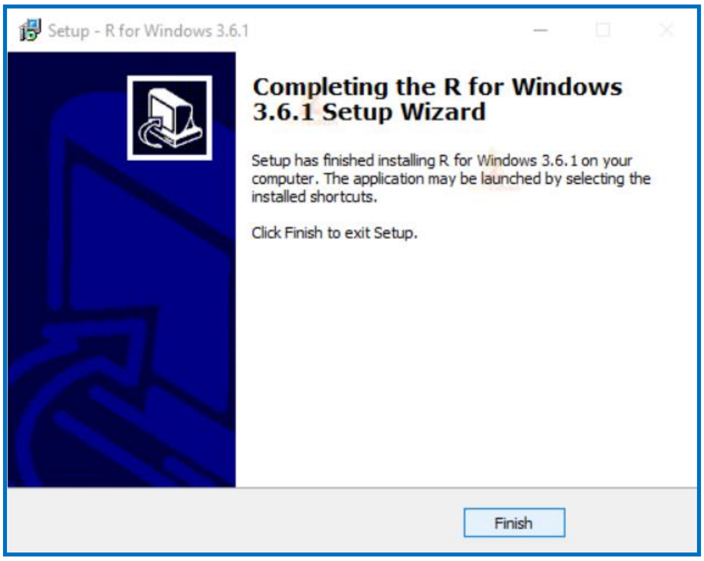


Figure 27: Finish the process

3.5 RStudio

Installing R is a pre-requisite to this section. The following steps show how to install RStudio on a Windows machine.

1. Go to RStudio website[8], the free version RStudio Desktop is suitable and sufficient in terms of features. Click on download button (Fig.28).

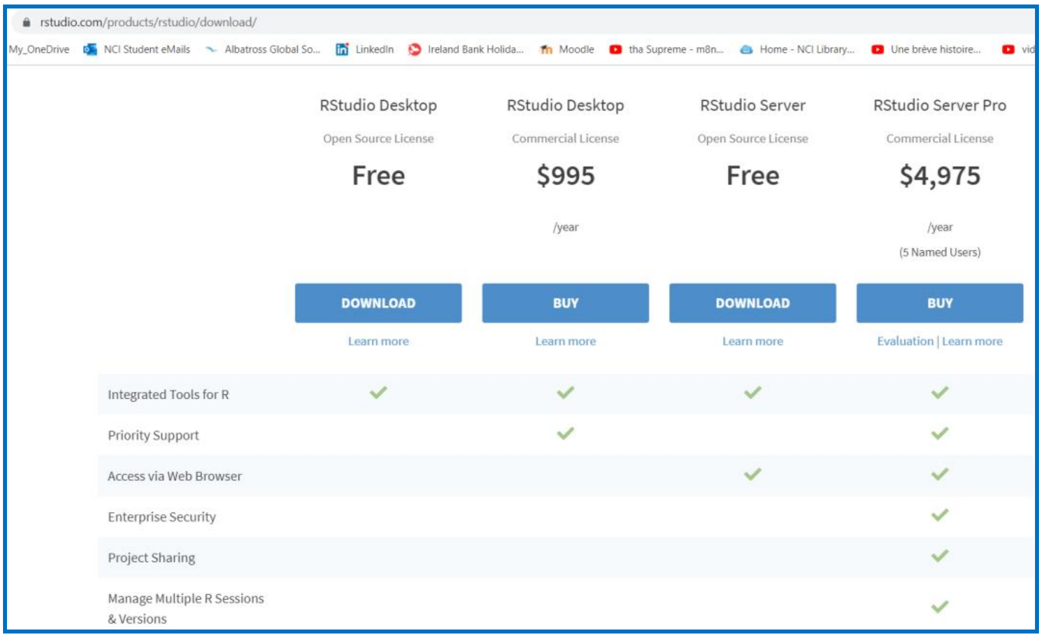


Figure 28: Download RStudio

We are directed to the page with instructions. We have already completed the first step by downloading the R base.

2. Click on Download RStudio for Windows (Fig.29). The installer file will start to download automatically.

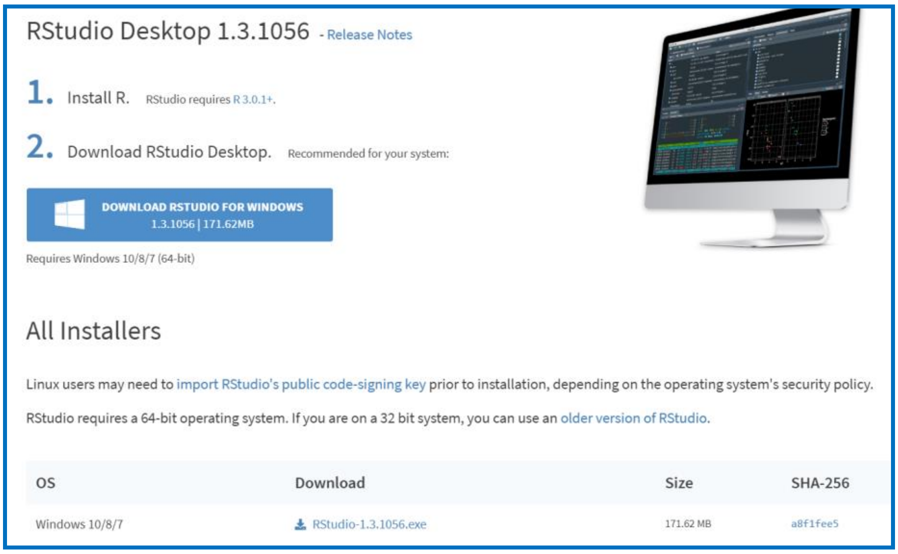


Figure 29: Download RStudio

[8] https://rstudio.com/products/rstudio/download/

The following instructions and screenshots were also taken from the tutorial prepared by the Techvidvan Team (2020)[9].

3. Open the file downloaded to execute it. Click on Next on the first window (Fig.30).

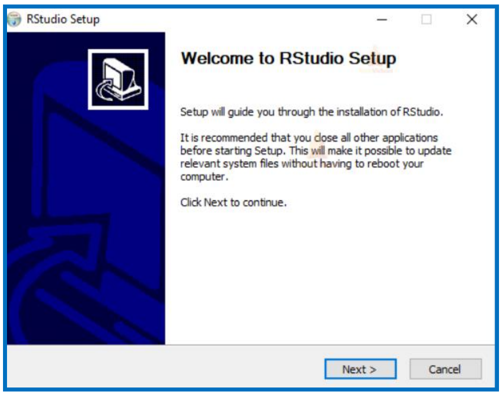


Figure 30: Start the installation of RStudio

4. Enter the path to the location where you want to install RStudio on the machine and click on Next.

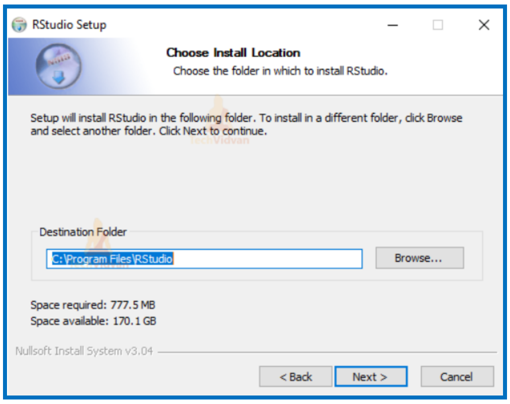


Figure 31: Enter the path

5. Select the Start Menu folder for creating the shortcut.

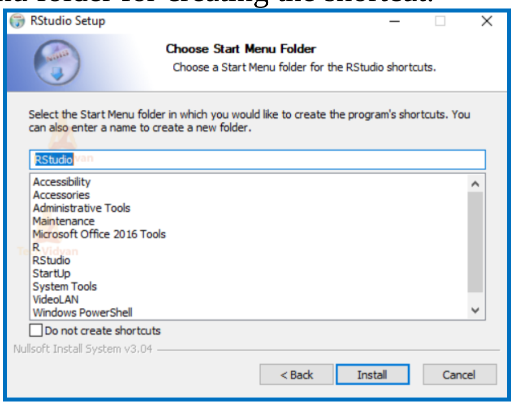


Figure 32: Create a shortcut

[9] https://techvidvan.com/tutorials/install-r/#install-r-windows

6. Wait for the installation to be completed (Fig.33).

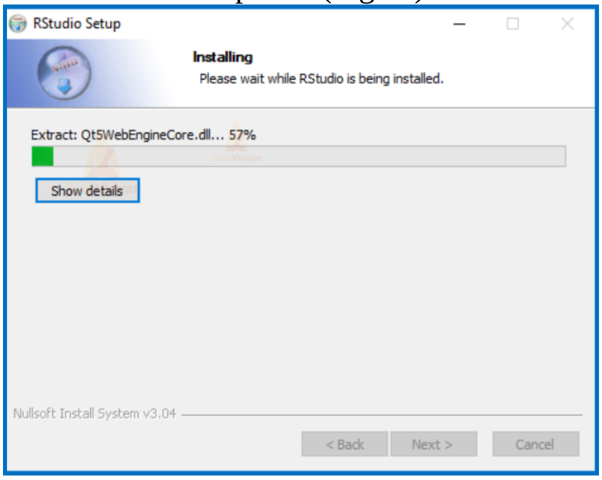


Figure 33: Installation in progress

7. Click on Finish in the setup wizard to complete the installation process (Fig.34).

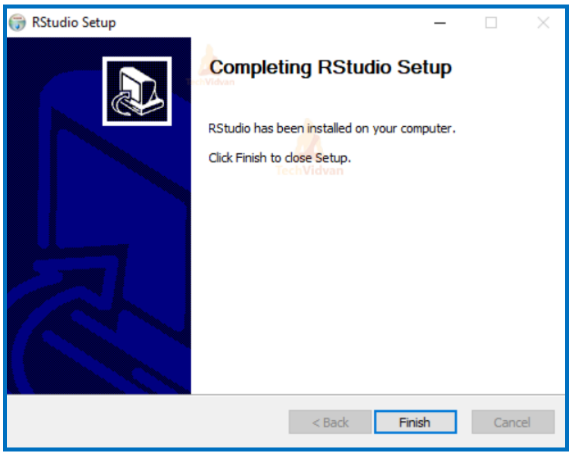


Figure 34: Finish the installation

In Rstudio, a number of packages were installed for the analysis, the list is detailed in Table 2.

Table 2: List of RStudio packages used

R Package Name	Description
syuzhet	Emotions analysis
tm	For text mining, corpus handling, creation of term-document matrices.
wordcloud	Create word clouds, visualize differences and similarity between documents
ggplot2	Data visualization

3.6 Tableau

Tableau Desktop version 2020.2 was installed to create visualisations for data exploratory analysis. Students from NCI can request a licence key to use the software.

1. Fill out details to request a licence key (Fig.35).

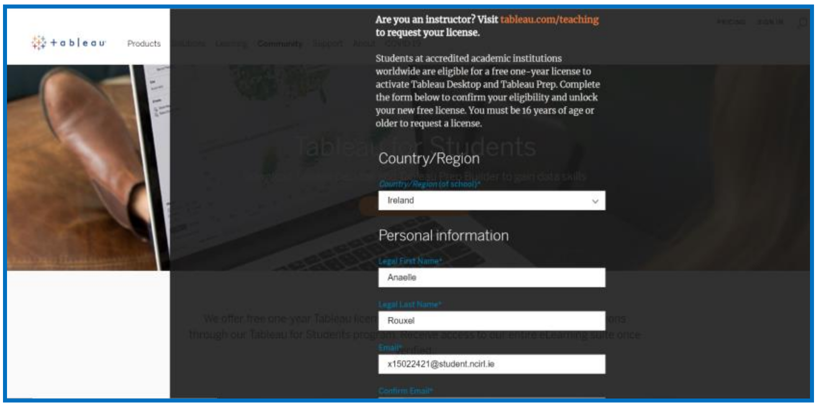


Figure 35: Request a license key

2. The key obtained is **TCZQ-EC37-33C0-58F1-A976**
3. The installation file 2020.2 can be downloaded from Tableau website[10].
4. Open the installer file as shown in Fig.36.

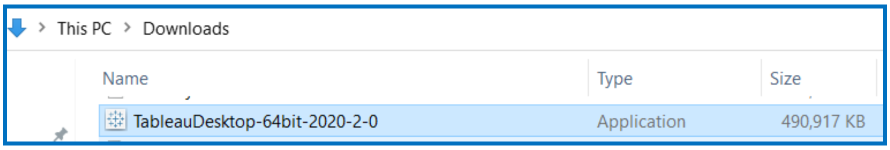


Figure 36: Open the installer file

5. Tick the box to confirm terms were read and are accepted. It will allow to move to the next step: click on Install (Fig.37).

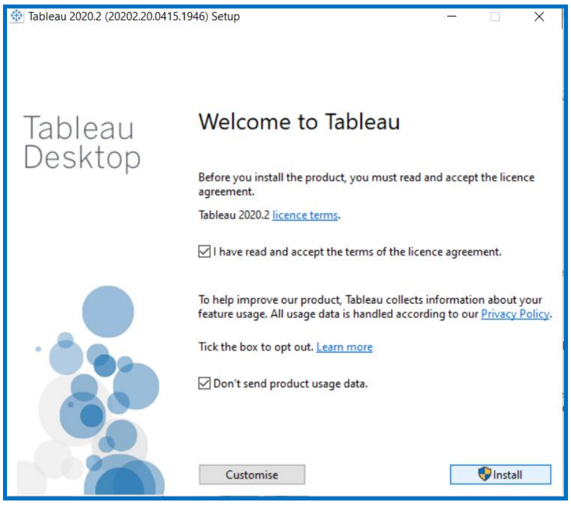


Figure 37: Accept terms and confirm the installation

[10] https://www.tableau.com/support/releases

6. Setup starts as shown in Fig. 38.

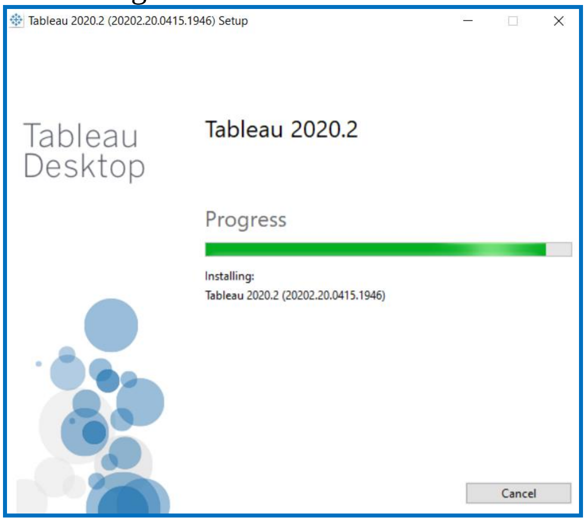


Figure 38: Installation in progress

7. The Registration step has not been taken in screenshot not to share personal details such as phone number and address. Once the registration is completed, Tableau Desktop opens to the homepage seen in Fig. 39.

Figure 39: Tableau Desktop homepage

4 Datasets

Datasets were created by scraping data from Twitter and The Guardian newspaper (section 4.1 Data creation). Then, Data Exploration and Pre-processing are detailed in sections 4.2 and 4.3.

4.1 Data Creation

Processes are explained respectively in sections 4.1.1 Scrapped Twitter Data and 4.1.3. News Data. As part of dataset creation, Twitter data was scrapped and also involved a part of formatting that is detailed in section 4.1.2.

4.1.1 Scrapped Twitter Data

Tweets were scraped with twitterscrapper library using Python via Anaconda prompt. The search tool Hashtagify[11] is designed for marketing purpose to assist in discovering the trending and associated hashtags on Twitter in order to reach audience. See Fig. 40 for the results suggested when searching for “covid”.

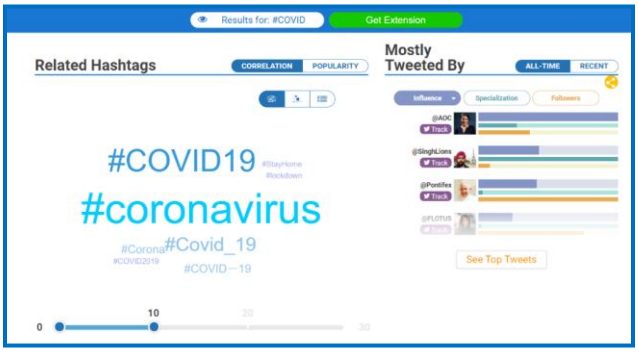


Figure 40: Popular hashtags related to “covid” on Twitter

The two keywords searched to scrap Tweets were #coronavirus and #covid. These two words are used internationally to describe the same pandemic disease, which started to be reported on social media and the news from mid-December 2019. The period targeted was from 01/01/2019 to 23/03/2020. The instruction given through the prompt was to get 10,000 Tweets for each of these two keywords and save data in a .json file.

Using the command from Anaconda Prompt, a first batch of tweets was scrapped on 24th March 2020 and a second batch on 24th June 2020. The screenshots below show the collection of data for this second batch of tweets, targeting the period from 24th March to 23rd June 2020.

1. Install the library ‘twitterscraper’ in Anaconda Prompt as shown in Fig. 41. The successful installation is shown in Fig. 42.

Figure 41: Install twitterscraper library

[11] https://hashtagify.me/hashtag/covid19

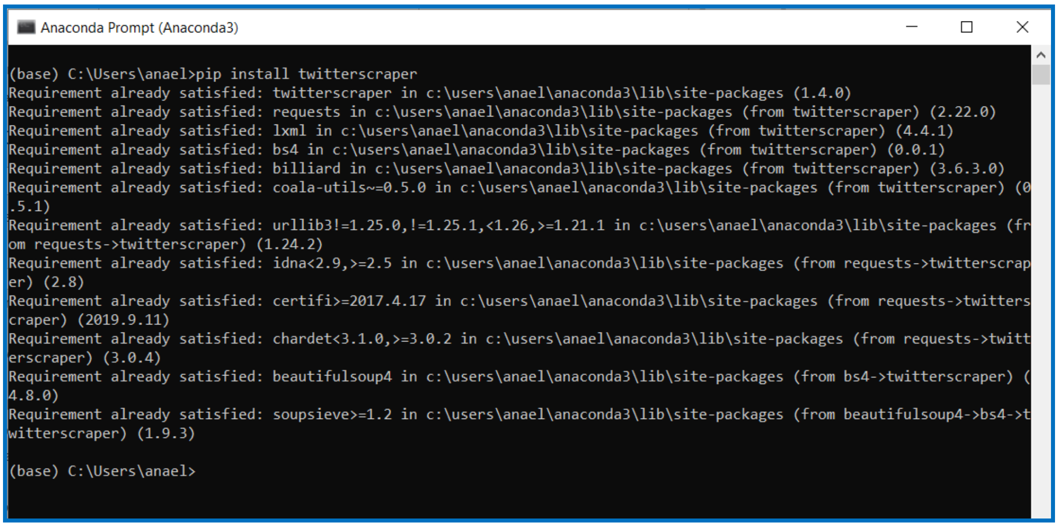


Figure 42: Installation Completed

2. Once the installation is completed (Fig. 42), run the command to scrap up to 10,000 tweets containing the hashtag #coronavirus and from the period 24th March 2020 to 23rd June 2020. The instruction syntax is shown in Fig.43.

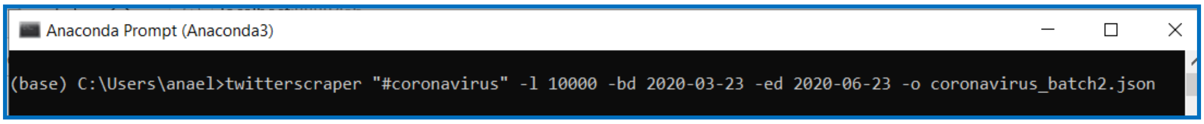


Figure 43: Instruction to scrap tweets with #coronavirus

3. With this instruction in Fig.43, the response appears in the Prompt. The beginning and end are shown in Fig.44. In this second batch, only 516 tweets with #coronavirus were scraped and saved in json file.

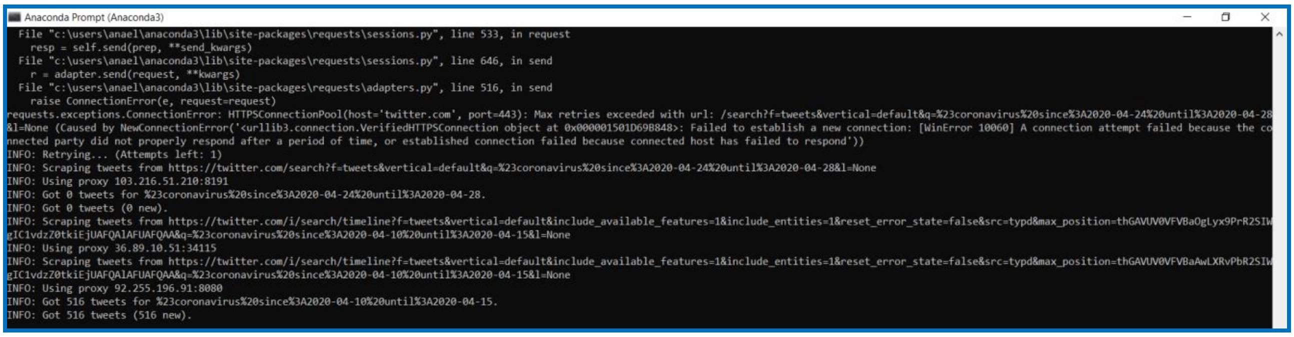


Figure 44: Start of Response in Anaconda Prompt

4. Data is saved as a json file and will be read in JupyterLab (Refer to the following section 4.1.2 Formatting Twitter Data). The same method is applied for scraping the second batch of tweets with the hashtag #covid. Fig.45 shows the query syntax for scraping the first set of #covid tweets, and Fig. 46 shows the data query of what is named in this project " #covid batch 2 tweets".

```
(base) C:\Users\anael>twitterscraper "#covid" -l 10000 -bd 2019-12-01 -ed 2020-03-23 -o covid_batch2.json
```

Figure 45: Instruction to scrap tweets with #covid on 23rd March

```
(base) C:\Users\anael>twitterscraper "#covid" -l 10000 -bd 2020-03-23 -ed 2020-06-23 -o covid_batch2.json
```

Figure 46: Instruction to scrap tweets with #covid on 24th June

5. Errors messages were returned when querying twitter on the 24th June (Fig.47). A connection issue occurred, and the query failed. Several attempts were made and were unsuccessful. No tweets were retrieved.

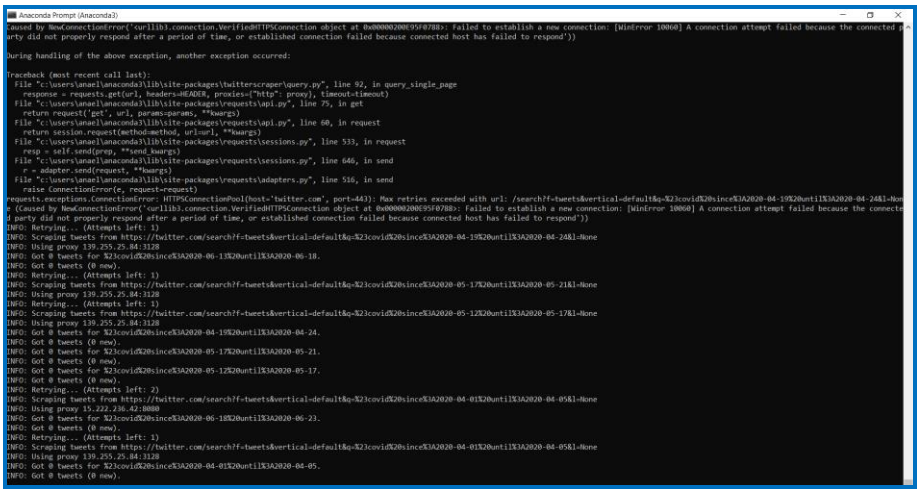


Figure 47: Response with Error

6. Tweets scraped were saved into 3 json files (Fig.48) before being formatted using JupyterLab and Python scripts.

Name	Date modified	Type	Size
coronavirus.json	24/03/2020 17:42	JSON File	13,411 KB
coronavirus_batch2.json	24/06/2020 08:13	JSON File	1,175 KB
covid.json	24/03/2020 17:56	JSON File	9,352 KB

Figure 48: Json files containing tweets

4.1.2 Formatting Twitter Data

Each .json file was formatted to save structured data into a .csv file. The following screenshots show the process for tweets extracted with #coronavirus on 24th March 2020. The corresponding Python script is named "Coronavirus_scrap_2020 06 06". It was taken from my previous project of Data warehousing module in Postgraduate Diploma and adapted to add the language feature.

1. Import required libraries (Fig. 49).

```
[1]: import codecs, json, csv, re
     from textblob import TextBlob
     import re
     import nltk
     nltk.download('punkt')
     nltk.download('wordnet')

     [nltk_data] Downloading package punkt to
     [nltk_data]     C:\Users\anael\AppData\Roaming\nltk_data...
     [nltk_data]   Package punkt is already up-to-date!
     [nltk_data] Downloading package wordnet to
     [nltk_data]     C:\Users\anael\AppData\Roaming\nltk_data...
     [nltk_data]   Package wordnet is already up-to-date!

[1]: True
```

Figure 49: Import libraries

2. Compute sentiment on raw tweet posts (Fig. 50) for using at a later stage in the research project (this feature will be renamed "sentiment1" in the pre-processing stage).

```
[2]: def sentiment_text(text):
         sent_sentences =[]
         blob = TextBlob(text)
         for sentence in blob.sentences:
             sent_sentences.append(sentence.sentiment.polarity)
         return sum(sent_sentences) / float(len(sent_sentences))
```

Figure 50: Compute sentiment

3. Read .json file and extract the language indicator in a separate .json file before bringing back the feature into the data (Fig. 51).

```
[3]: # Read a json file downloaded with twitterscraper
     # Anaconda Prompt on 24/03/20 => twitterscraper "#coronavirus" -l 10000 -bd 2019-12-01 -ed 2020-03-23 -o coronavirus.json

     with codecs.open('C:\\Users\\anael\\coronavirus.json','r','utf-8') as f: #open the file with format utf used for web pages
         tweets = json.load(f,encoding='utf-b')

[21]: # Extract language feature from 'text_html'

      pattern = re.compile('lang=\"(\w+)\"')

      for tweet in tweets:
          lang = pattern.search(tweet['text_html']).group(1)
          tweet['lang'] = lang
          # Si tweet est en anglais
          # if 'lang="en"' in tweet["text_html"]:

      with codecs.open('C:\\Users\\anael\\coronavirus_with_langs.json', 'w', 'utf-8') as file: #"with open" means we don't need to close the file
          file.write(json.dumps(tweets))

[22]: with codecs.open('C:\\Users\\anael\\coronavirus_with_langs.json', 'r', 'utf-8') as file:
          tweets = json.load(file)
          print(tweets[0]['lang'])

      en

[15]: print(tweets[0])

      {'has_media': True, 'hashtags': ['Bacillus', 'Coronavirus'], 'img_urls': ['https://pbs.twimg.com/media/EMfBrSqXYAIek9E.jpg'], 'is_replie
      d': True, 'is_reply_to': False, 'likes': 73, 'links': [], 'parent_tweet_id': '', 'replies': 2, 'reply_to_users': [], 'retweets': 11, 'scre
      en_name': 'macerwan', 'text': 'My surprise when I discovered that two roman characters from the asterix and obelix comics are called #Baci
      llus and #Coronavirus. pic.twitter.com/vkRLietaSl', 'text_html': '<p class="TweetTextSize js-tweet-text tweet-text" data-aria-label-part
      ="0" lang="en">My surprise when I discovered that two roman characters from the asterix and obelix comics are called <a class="twitter-has
      htag pretty-link js-nav" data-query-source="hashtag_click" dir="ltr" href="/hashtag/Bacillus?src=hash"><s>#</s><b>Bacillus</b></a> and <a
      class="twitter-hashtag pretty-link js-nav" data-query-source="hashtag_click" dir="ltr" href="/hashtag/Coronavirus?src=hash"><s>#</s><b><st
      rong>Coronavirus</strong></b></a>. <a class="twitter-timeline-link u-hidden" data-pre-embedded="true" dir="ltr" href="https://t.co/vkRLiet
```

Figure 51: Read .json file and extract language feature

4. Create the .csv file "coronavirus_sentiment.csv" with the 10 required tweets features: 'count','tweet_id','timestamp','parent_tweet_id','user_id','lang','likes','retweets','sentiment','text' (Fig.52).

```
[23]: # Save into a .csv file
      file = "coronavirus_sentiment.csv"  #file name

[24]: # Open .csv file to append
      target_file = open(file, 'w', encoding='utf-8', newline='')
      csv_file = csv.writer(target_file, delimiter=',', quotechar='"')
      csv_file.writerow(['count','tweet_id','timestamp','parent_tweet_id','user_id','lang','likes','retweets','sentiment','text'])  #save the fi

[24]: 85

[25]: count=0 #a counter

      for tweet in tweets:
          count=count+1
          tweet_content = re.sub("[^A-Za-z0-9.:,'/?!#@]", " ", tweet['text'])   #uncomment this line to strip strange characters
      #   tweet_content = tweet['text'].strip()
          sentiment = sentiment_text (tweet_content)
          tw_timestamp =  re.sub("T"," ",tweet['timestamp'])
          print(count,tweet['tweet_id'],tw_timestamp,tweet['parent_tweet_id'],tweet['user_id'],tweet['likes'],tweet['retweets'],sentiment,tweet_c
          print("==============================================")
          csv_file.writerow([count,tweet['tweet_id'],tw_timestamp, tweet['parent_tweet_id'],tweet['user_id'],tweet['lang'],tweet['likes'],tweet[

      1 1209148016722169856 2019-12-23 16:25:26  301646116 73 11 0.0 My surprise when I discovered that two roman characters from the asterix
      and obelix comics are called #Bacillus and #Coronavirus. pic.twitter.com/vkRLietaSl en
      ==============================================
      2 1209148016722169856 2019-12-23 16:25:26  301646116 73 11 0.0 My surprise when I discovered that two roman characters from the asterix
      and obelix comics are called #Bacillus and #Coronavirus. pic.twitter.com/vkRLietaSl en
      ==============================================
      3 1201998948950577152 2019-12-03 22:57:36  989521438825746433 17 1 0.0 Amy s a survivor! #bariclab #pnnl #movingon #coronavirus #bsl3 #s
      cience #grownups #professorlifepic.twitter.com/sND6q0r52I en
      ==============================================
      4 1200977067266990080 2019-12-01 03:17:00  2414645220 0 0 0.0 A review of asymptomatic and sub clinical Middle East Respiratory Syndrome
      #Coronavirus Infections https://academic.oup.com/epirev/advance article/doi/10.1093/epirev/mxz009/5634013#.XeMwoBj5eW4.twitter   en
      ==============================================

[12]: # Close the .csv file
      target_file.close()
```

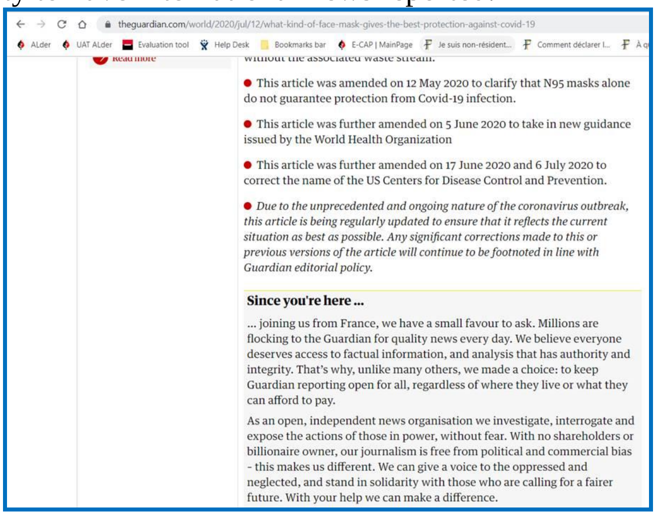
Figure 52: Create the dataframe

4.1.3 News data

Articles were collected from the English newspaper The Guardian. The online version has a section dedicated to coronavirus that displays a selection of news articles.

4.1.3.1 Scraped Articles from The Guardian

This newspaper was chosen for the text in English language, the reliability of its content (Fig. 53) and the opportunity to have international news reported.



theguardian.com/world/2020/jul/12/what-kind-of-face-mask-gives-the-best-protection-against-covid-19

● This article was amended on 12 May 2020 to clarify that N95 masks alone do not guarantee protection from Covid-19 infection.

● This article was further amended on 5 June 2020 to take in new guidance issued by the World Health Organization

● This article was further amended on 17 June 2020 and 6 July 2020 to correct the name of the US Centers for Disease Control and Prevention.

● *Due to the unprecedented and ongoing nature of the coronavirus outbreak, this article is being regularly updated to ensure that it reflects the current situation as best as possible. Any significant corrections made to this or previous versions of the article will continue to be footnoted in line with Guardian editorial policy.*

Since you're here ...

... joining us from France, we have a small favour to ask. Millions are flocking to the Guardian for quality news every day. We believe everyone deserves access to factual information, and analysis that has authority and integrity. That's why, unlike many others, we made a choice: to keep Guardian reporting open for all, regardless of where they live or what they can afford to pay.

As an open, independent news organisation we investigate, interrogate and expose the actions of those in power, without fear. With no shareholders or billionaire owner, our journalism is free from political and commercial bias - this makes us different. We can give a voice to the oppressed and neglected, and stand in solidarity with those who are calling for a fairer future. With your help we can make a difference.

Figure 53: The Guardian statement

News articles were scraped from the dedicated page to coronavirus of The Guardian[12] between the 21st June and 9th July 2020. Using Python and the tutorial of Miguel Fernandez Zafra "Web Scraping news articles in Python" [13] (July 2019), the script was adapted to scrap all the news articles presented on the webpage on the given date when running the script.

The screenshots presented in this manual were taken on 9th July, where 37 articles were scraped on that day. Note this was the lowest number of articles collected from this webpage during the period where news articles were scraped.

1. Script is shown from Fig. 54 where the content of the webpage is screened through. To be noted in this figure, the result cg_nmart is 56 and corresponds to the number of headings "h3" found on the page. All of these are not news articles.

```
[1]: import requests
     from bs4 import BeautifulSoup
     import numpy as np
     import pandas as pd
     import time

[2]: #Obtain list of news from the coverpage
     #URL definition:
     url = "https://www.theguardian.com/world/coronavirus-outbreak"

[3]: #List of news

     # Request
     r1 = requests.get(url)
     r1.status_code

     # We'll save in coverpage the cover page content
     coverpage = r1.content

     # Soup creation
     soup1 = BeautifulSoup(coverpage, 'html5lib')

     # News identification
     coverpage_news = soup1.find_all('h3', class_='fc-item__title')
     cg_numart = len(coverpage_news)
     cg_numart

[3]: 56
```

Figure 54: Screen through the webpage content

2. Create the elements necessary for the function (Fig. 55).

```
[4]: #Now we have a list in which every element is a news article:
     #coverpage_news
     first_article = coverpage_news[0]
     last_article = coverpage_news[cg_numart-1]
     last_article

[4]: <h3 class="fc-item__title"><a class="fc-item__link" data-link-name="article" href="https://www.theguardian.com/world/2020/jul/09/covid-19-pandemic-accelerating-says-who-as-review-panel-named"> <span class="headline-list__body fc-item__headline"> <span class="js-headline-text">Covid-19 pandemic accelerating says WHO as review panel named</span></span> </a></h3>

[5]: #Let's extract the text from the articles:
     #First, we'll define the number of articles we want:

     number_of_articles = cg_numart #set the nb of articles as = to count number "cg_numart"
```

Figure 55: Initialize elements

[12] https://www.theguardian.com/world/coronavirus-outbreak
[13] https://towardsdatascience.com/web-scraping-news-articles-in-python-9dd605799558

3. Run the script in Fig. 56 to extract the news articles title, content and address link.

```
[6]: # Empty lists for content, links and titles
news_contents = []
list_links = []
list_titles = []

for n in np.arange(0, number_of_articles):

    # We need to ignore "live" pages since they are not articles
    if "live" in coverpage_news[n].find('a')['href']:
        continue

    # Getting the link of the article
    link = coverpage_news[n].find('a')['href']
    list_links.append(link)

    # Getting the title
    title = coverpage_news[n].find('a').get_text()
    list_titles.append(title)

    # Reading the content (it is divided in paragraphs)
    article = requests.get(link)
    article_content = article.content
    soup_article = BeautifulSoup(article_content, 'html5lib')
    body = soup_article.find_all('div', class_='content__article-body from-content-api js-article__body')
    #print(len(body)) # it shows 1 or 0 to say if the body of articles contains something (abscence = 0)
    if (len(body)>0):
        x = body[0].find_all("p")

        # Unifying the paragraphs
        list_paragraphs = []
        for p in np.arange(0, len(x)):
            paragraph = x[p].get_text()
            list_paragraphs.append(paragraph)
            final_article = " ".join(list_paragraphs)

        news_contents.append(final_article)
```

Figure 56: Scrap news titles, content and hyperlink

4. Articles content are saved in a dataframe and the titles and links to articles are saved in a different dataframe (Fig. 57).

```
[7]: #Let's put them into:
#a dataset with the content of articles and the source (df_features)
#a dataset with the title and the link (df_show_info)

# df_features
df_features = pd.DataFrame(
     {'Article Content': news_contents,
      'Newspaper': 'The Guardian'})

# df_show_info
df_show_info = pd.DataFrame(
    {'Article Title': list_titles,
     #'Article Content': news_contents,
     'Article Link': list_links,
     'Newspaper': 'The Guardian'})

[8]: df_features.shape

[8]: (34, 2)

[9]: df_show_info.shape

[9]: (48, 3)
```

Figure 57: Save dataframes

The two dataframes are of different lengths because the loop when through each element on the webpage. Some headings “h3”, therefore titles on the webpage scraped, had no article retrieved due to the format of the text body on the website. This is illustrated with an example from Fig.59 to Fig. 61, in the section 4.1.3.2.Length of News Dataframes”.

5. Export dataframes as .csv files, names include the date of extraction (Fig.58).

```
[10]: ### NOTE: update the files name not to overwrite data !

      #save df_features in a csv
      df_news = pd.DataFrame(df_features, columns= ['Article Content', 'Newspaper'])
      df_news.to_csv (r'C:/Users/anael/News_data/exportdf_Guardian_news_scrap_20200709.csv', index = False, header=True) #export as csv file
      #print (df_news)

      #save df_show_info in a csv
      df_news_details = pd.DataFrame(df_show_info, columns= ['Article Title','Article Link','Newspaper'])
      df_news_details.to_csv (r'C:/Users/anael/News_data/exportdf_Guardian_newsdetails_scrap_20200709.csv', index = False, header=True) #export as cs
      #print (df_news_details)
```

Figure 58: Export dataframes as .csv

4.1.3.2 Length of News Dataframes

Two dataframes were created and they have different number of instances. If we look at a particular news article shown in Fig. 59 on The Guardian webage, and its content on Fig.60. We can note what happened on the 9th July during the scraping process. We see that the first article content retrieved by the script and presented on Excel spreadsheet in Fig. 61 (left side) corresponds to the second title scraped on Fig. 61 (right). When looking at the website coverpage (Fig. 59) the title *"Covid-19 cases tied to fraternity parties disrupt UC Berkeley's reopening plans"* is the 5th on the webpage.

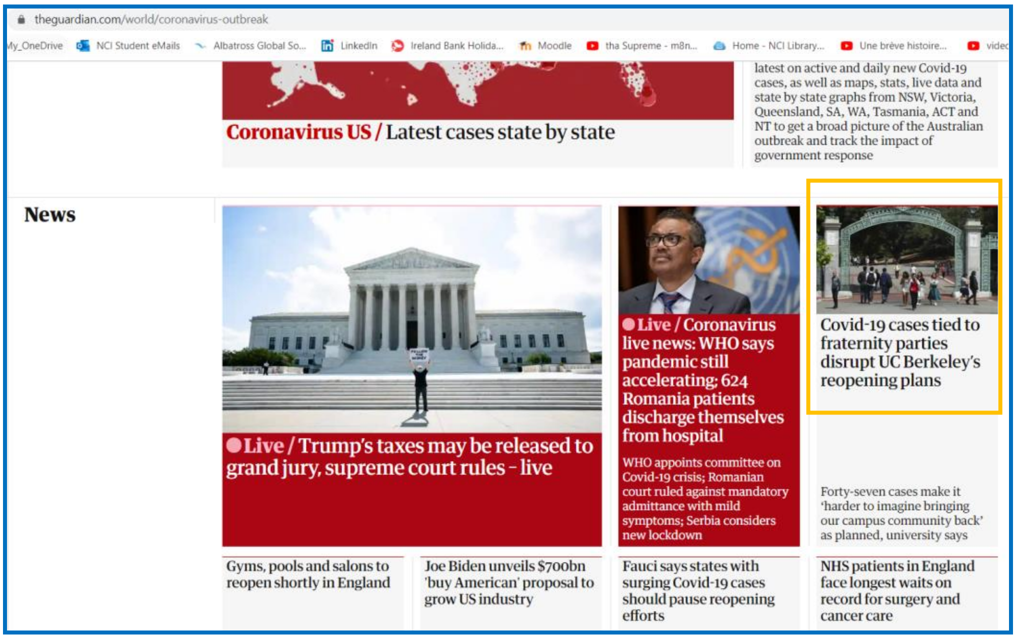


Figure 59: Coverpage on 9th July 2020

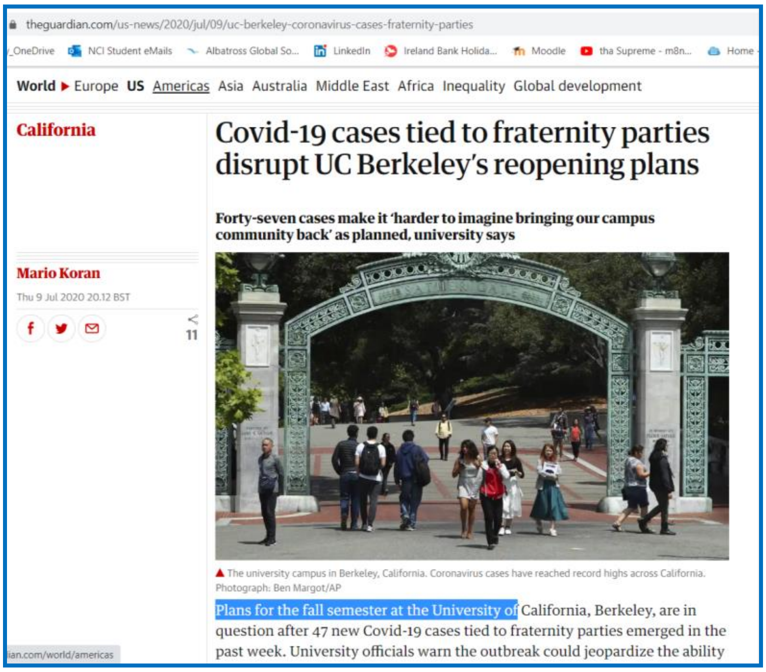


Figure 60: Article body

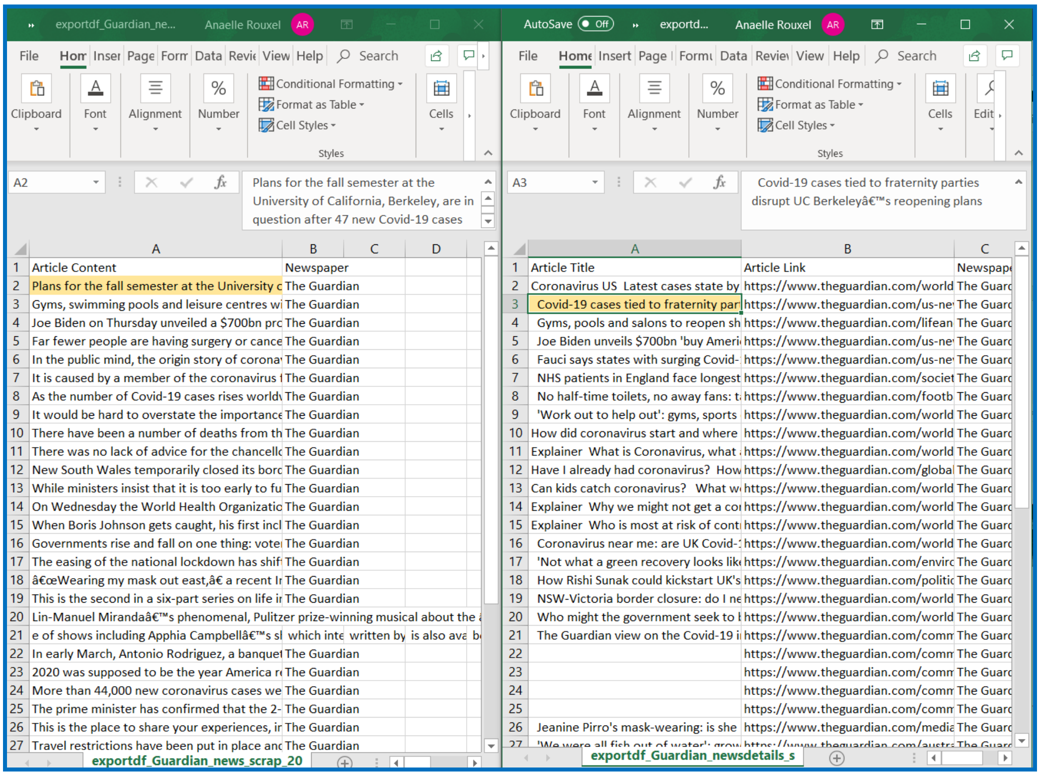


Figure 61: Scrapped data displayed in Excel

The news details (titles and links) are kept for information, only news articles are analysed with Natural Language Processing techniques in this project due to time constraints. The reason being the text content is of larger size and provide enough material for semantic and stance-based analysis.

The exact same script was run on multiple days to collect data. The extraction date in the name of the .csv file should be manually amended when saving data scraped.

The scraped news articles (text body) and news details (titles and links) were saved on multiple days as .csv files. New articles were merged together in the file "combined_newsarticles.csv" and the news details into "combined_newsdetails.csv" using one method to select the files based on their name (Fig. 62). The pre-requisite was to name the data in a relevant fashion to perform a search and select method in the files' name.

```
[11]: #Merge News csv files
      import os, glob
      import pandas as pd

      os.chdir("C:/Users/anael/News_data")
      extension = 'csv'
      #Match CSV files by pattern : name starting with 'exportdf_Guardian_newsdetails'
      all_filenames = [i for i in glob.glob('exportdf_Guardian_newsdetails*.{}'.format(extension))]
      print(all_filenames)

      ['exportdf_Guardian_newsdetails_scrap_20200621.csv', 'exportdf_Guardian_newsdetails_scrap_20200623.csv', 'exportdf_Guardian_newsdetails_scrap_20200625.csv', 'exportdf_Guardian_newsdetails_scrap_20200629.csv', 'exportdf_Guardian_newsdetails_scrap_20200704.csv', 'exportdf_Guardian_newsdetails_scrap_20200709.csv']

[12]: #combine all files in the list
      combined_csv = pd.concat([pd.read_csv(f) for f in all_filenames ])
      #export to csv
      combined_csv.to_csv( "combined_newsdetails.csv", index=False, encoding='utf-8-sig')

[13]: #Combine news articles from several csv files too
      #Match CSV files by pattern : name starting with 'exportdf_Guardian_news_scrap'
      all_articles_filenames = [i for i in glob.glob('exportdf_Guardian_news_scrap*.{}'.format(extension))]
      print(all_articles_filenames)
      #combine all files in the list
      combined_articles_csv = pd.concat([pd.read_csv(f) for f in all_articles_filenames ])
      #export to csv
      combined_articles_csv.to_csv( "combined_newsarticles.csv", index=False, encoding='utf-8-sig')

      ['exportdf_Guardian_news_scrap_20200621.csv', 'exportdf_Guardian_news_scrap_20200623.csv', 'exportdf_Guardian_news_scrap_20200625.csv', 'exportdf_Guardian_news_scrap_20200629.csv', 'exportdf_Guardian_news_scrap_20200704.csv', 'exportdf_Guardian_news_scrap_20200709.csv']
```

Figure 62: Technique to merge .csv files

A dataset of 227 articles was obtained, containing 158 unique articles. Further information will be details in the pre-processing section 4.3.2. News Articles.

4.2 Data Exploration

Datasets exploration was done using Tableau and Python.

4.2.1 Exploration of Tweets

Initial exploration before applying cleaning techniques was done using visuals on Tableau and then Python for the data structure and missing values.

4.2.1.1 Tableau

1. Open Tableau and connect the file containing tweets named combined_tweets.csv. Dataset loaded is in Fig.63.

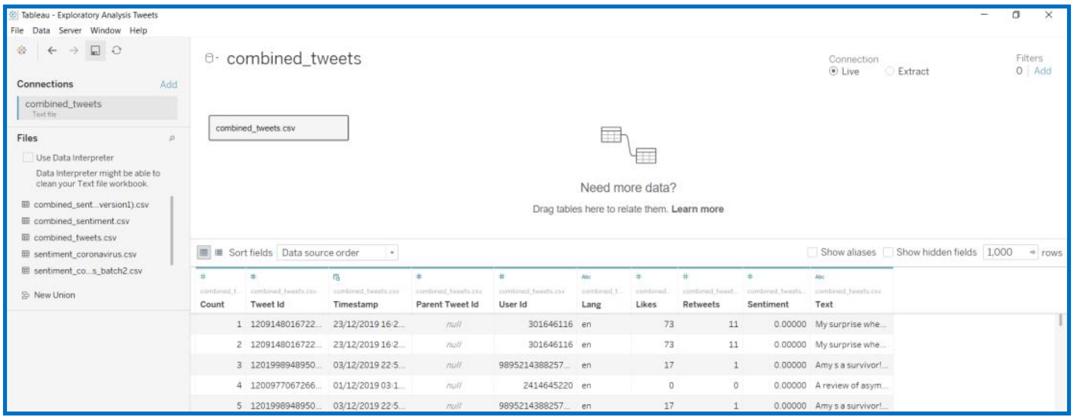


Figure 63: Dataset loaded to Tableau

2. Go to sheet 1 and rename it – Time period. We want to see the number of tweets collected over the period. There were only 38 tweets in December 2019 as seen in Fig. 64.

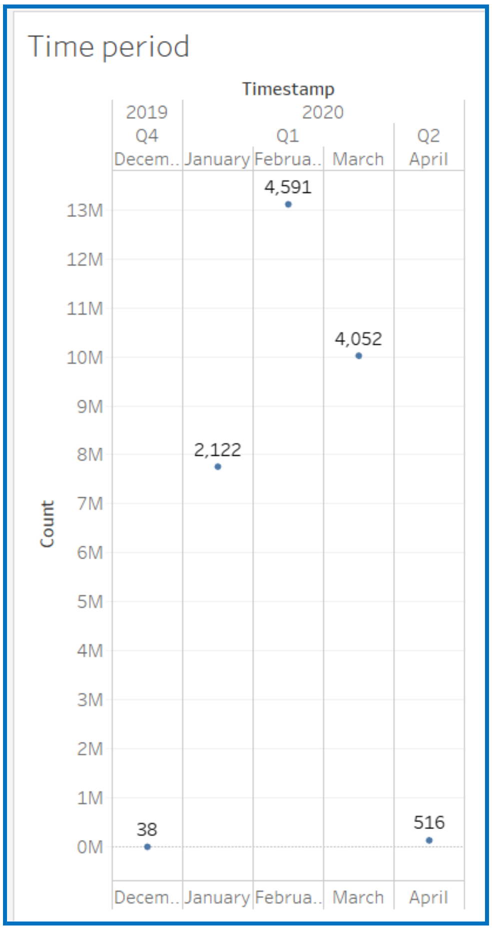


Figure 64: Number of tweets per month

3. If we click on "+" sign beside "month" we can increase the granular level to day (Fig.65).

Figure 65: Number of tweets per day

4. The majority of tweets is posted at 23.00 (Fig. 66).

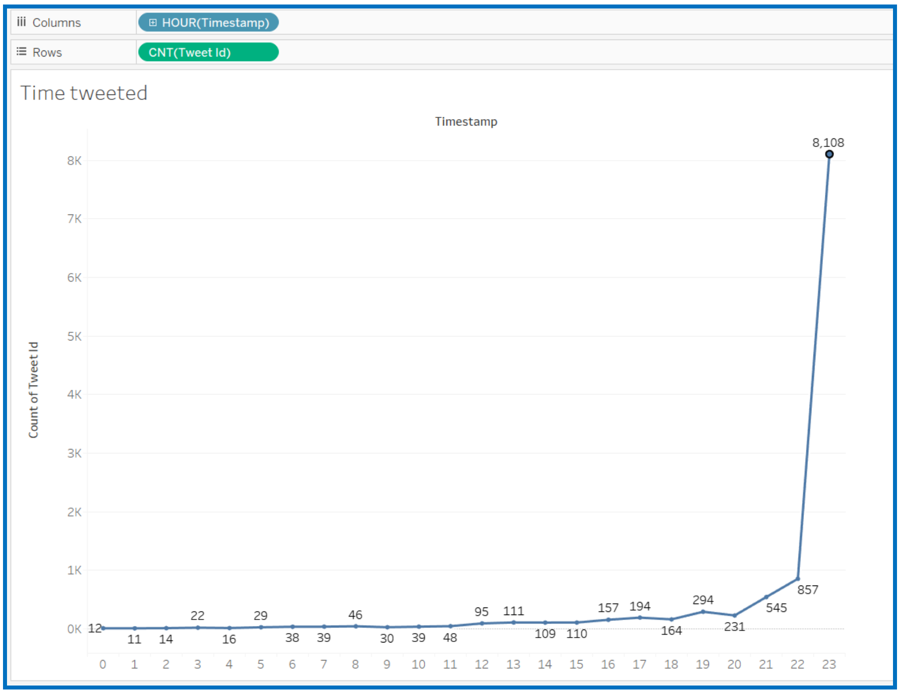


Figure 66: Time posted

5. The sum of retweets per week shows when the topic of coronavirus peaked on Twitter. We see three peaks: 14th January, 19th January and 28th February 2020 (Fig. 67).

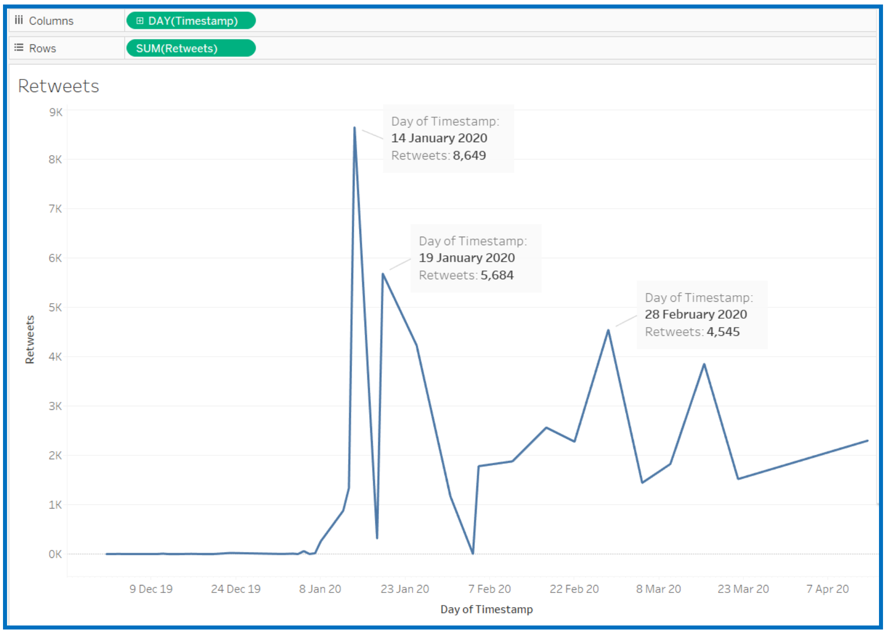


Figure 67: Retweets

6. When exploring the language of tweets, we can see the majority are written in English. Fig. 68 shows the count of distinct tweets_id to exclude duplicates and ensure the dataset will be of sufficient size for the project (greater than 5,000 items).

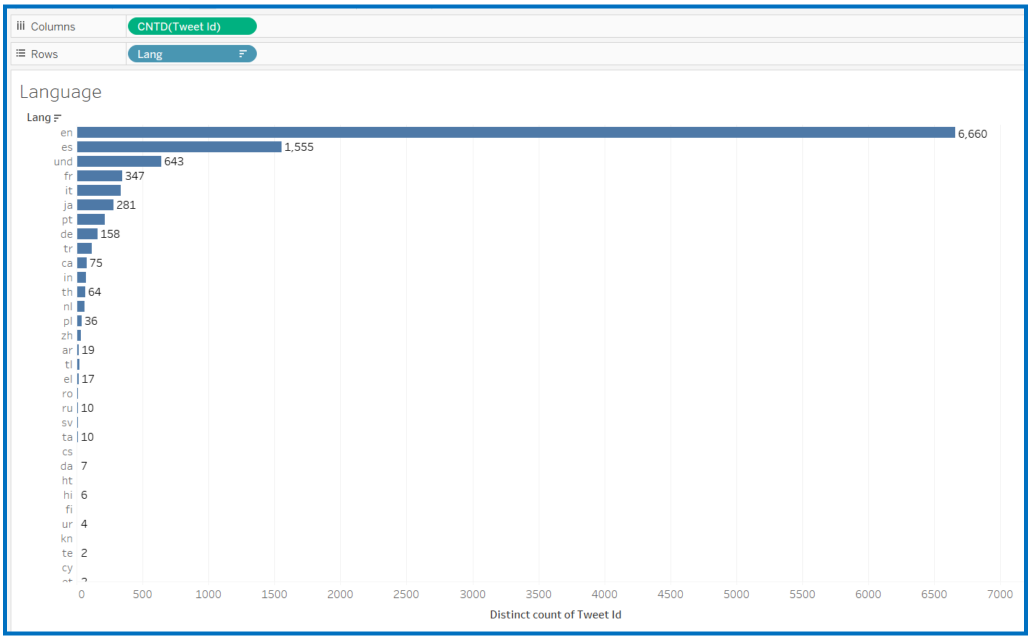


Figure 68: Unique tweets per language

7. Fig. 69 shows how to select “count distinct” measure. There are 6,950 English tweets in total and 6,660 unique English tweets. Note that “und” stands for “undetermined” language.

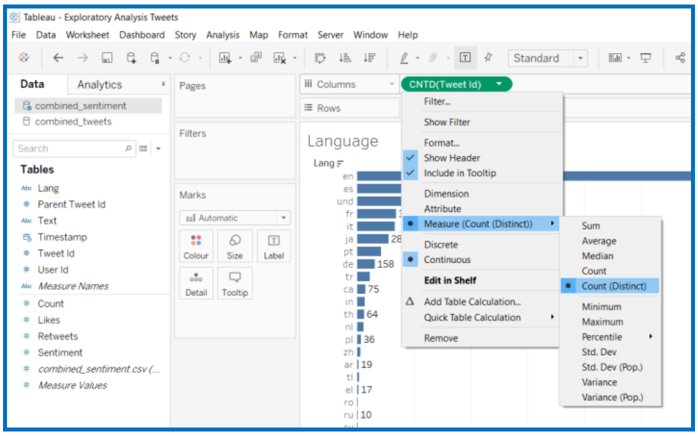


Figure 69: Select measure "count distinct"

8. The average sentiment of the entire dataset (Fig.70) during the period analysed is rather neutral with 0.03303 polarity score (close to zero).

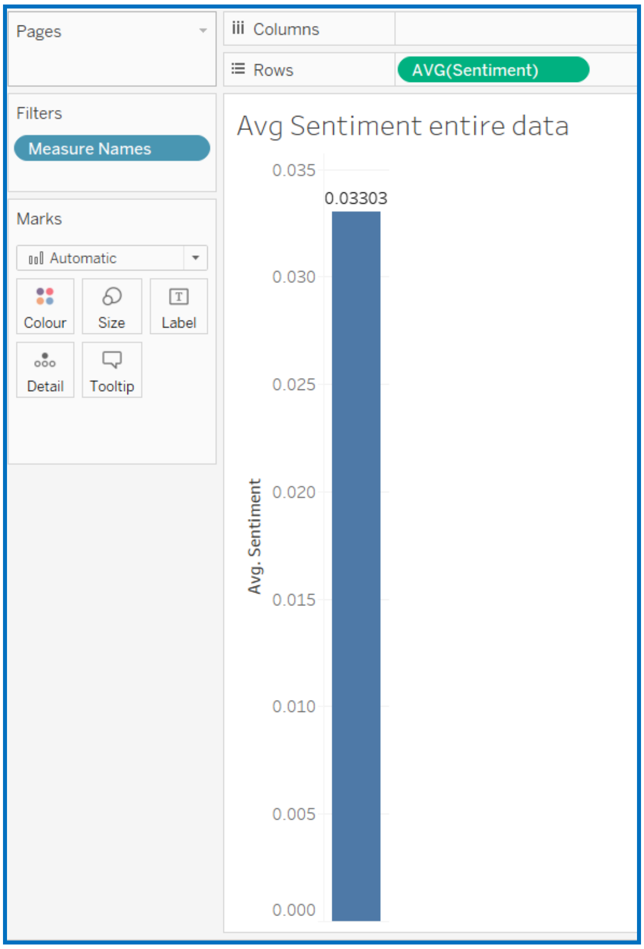


Figure 70: Tweets average sentiment

9. The average sentiment can be plotted per day (Fig. 71) to translate the morale of people posting message on the social platform.

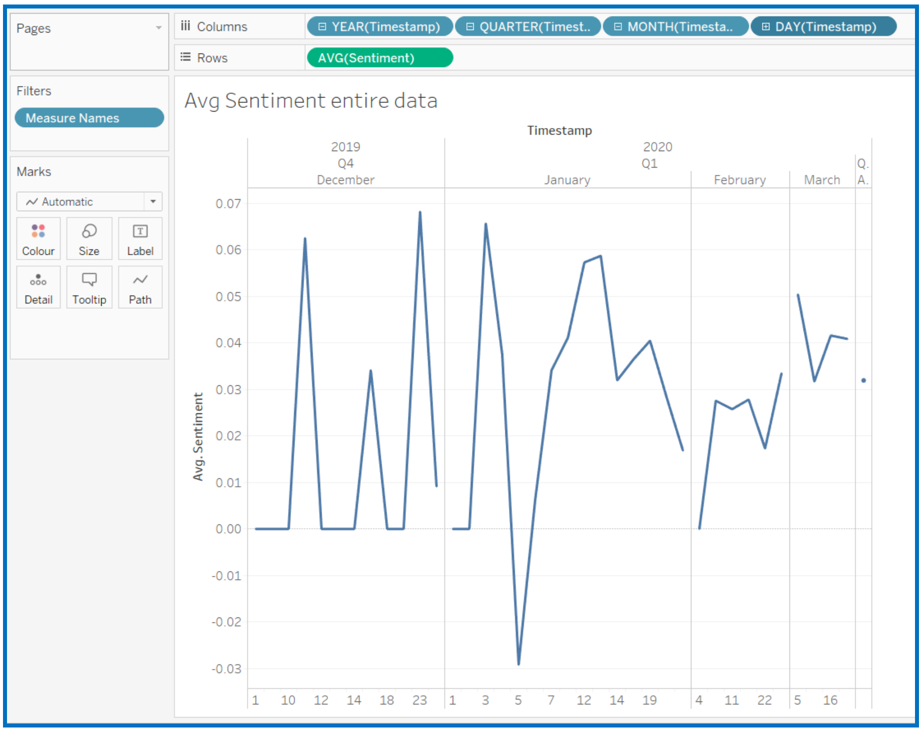


Figure 71: Average sentiment per day

Tweets data was then further explored using Python.

4.2.1.2 Python

One challenge was to add a label with the language to tweets (see section 4.1.2 Formatting Twitter Data). The information was available from the element "text_html" and extracted as a separate element to give a clear language category to Tweets. The project scope is limited to English tweets and news for fake information detection. But we can replicate the models and analysis to other languages available in the Tweets and compare to reliable news sources in the relevant language.

After labelling the language, Tweets containing #coronavirus and #covid were merged into a single dataset. A dataframe was created and fields retained are presented in Table 3.

Table 3: Tweet features extracted

No	Data Field	Description
1	count	Indexation automatically generated during the extraction
2	tweet_id	Unique id of the tweet
3	tw_timestamp	Tweet date and time
4	parent_id	Unique id of the original tweet
5	user_id	Unique id of the platform User
6	Lang	Language of the tweet
7	likes	Number of likes
8	retweets	Number of retweets
9	sentiment	Sentiment of the tweet
10	Text	Tweet text message content

Data exploration is necessary to handle missing values and decide on how to clean text.

1. Import libraries, read the file combined_tweets.csv and get the dataset structure (Fig.72).

```
[1]: ### Twitter data pre-processing ###

     import os
     import pandas as pd
     import numpy as np
     import preprocessor as p # Designed to clean tweets
     import re # For regular expressions
     import string # For handling string
     import gensim # Used for stopwords here. Library for topic modelling, document indexing and similarity retrieval with large corpora
     from nltk.tokenize import sent_tokenize, word_tokenize
     from nltk.stem import WordNetLemmatizer

     os.chdir('C:/Users/anael/sentimentFiles')
     os.getcwd()

[1]: 'C:\\Users\\anael\\sentimentFiles'

[2]: # Read .csv file and verify structure
     data = pd.read_csv('combined_tweets.csv')
     print(data.shape) # 11,319 rows and 10 col
     print(data.columns)

     (11319, 10)
     Index(['count', 'tweet_id', 'timestamp', 'parent_tweet_id', 'user_id', 'lang',
            'likes', 'retweets', 'sentiment', 'text'],
           dtype='object')
```

Figure 72: Import libraries and dataset

2. Explore the numeric and string features with descriptive statistics (Fig. 73).

```
[3]: # Descriptive statistics for numeric variables
     data.describe(include=['number'])
```

[3]:

	count	tweet_id	parent_tweet_id	user_id	likes	retweets	sentiment
count	11319.000000	1.131900e+04	1.030000e+03	1.131900e+04	11319.000000	11319.000000	11319.000000
mean	2741.157523	1.232003e+18	1.230958e+18	3.561430e+17	8.660394	4.120947	0.033027
std	1846.431967	8.312927e+15	2.939792e+16	5.035897e+17	141.209741	86.902313	0.150928
min	1.000000	1.200977e+18	4.757973e+17	1.585000e+03	0.000000	0.000000	-1.000000
25%	1157.500000	1.227302e+18	1.227307e+18	1.536318e+08	0.000000	0.000000	0.000000
50%	2572.000000	1.231368e+18	1.233426e+18	1.348140e+09	1.000000	0.000000	0.000000
75%	3987.000000	1.237528e+18	1.237507e+18	9.109451e+17	2.000000	1.000000	0.051389
max	6710.000000	1.250212e+18	1.250211e+18	1.248314e+18	9365.000000	8056.000000	1.000000

```
[3]: # Descriptive statistics with 'object' for string variables
     data.describe(include=['object'])
```

[3]:

	timestamp	lang	text
count	11319	11319	11319
unique	8242	41	10679
top	2020-03-16 23:59:59	en	#COVID 19
freq	16	6950	19

Figure 73: Descriptive statistics

3. Check for missing values (Fig.74). This verification shows that 90.90% of parent_tweet_id values are missing.

```
[4]: # Check is there any missing values in dataframe as a whole
     data.isnull()
```

[4]:

	count	tweet_id	timestamp	parent_tweet_id	user_id	lang	likes	retweets	sentiment	text
0	False	False	False	True	False	False	False	False	False	False
1	False	False	False	True	False	False	False	False	False	False
2	False	False	False	True	False	False	False	False	False	False
3	False	False	False	True	False	False	False	False	False	False
4	False	False	False	True	False	False	False	False	False	False
...	...	...	...	...	...	...	...	...	...	...
11314	False	False	False	True	False	False	False	False	False	False
11315	False	False	False	True	False	False	False	False	False	False
11316	False	False	False	True	False	False	False	False	False	False
11317	False	False	False	False	False	False	False	False	False	False
11318	False	False	False	True	False	False	False	False	False	False

11319 rows × 10 columns

```
[5]: # Check missing values across columns
     data.isnull().any()

[5]: count              False
     tweet_id           False
     timestamp          False
     parent_tweet_id     True
     user_id            False
     lang               False
     likes              False
     retweets           False
     sentiment          False
     text               False
     dtype: bool

[6]: # Check how many missing values across columns
     data.isnull().sum() #parent tweet ID is incomplete for 90.90% of records

[6]: count                  0
     tweet_id               0
     timestamp              0
     parent_tweet_id    10289
     user_id                0
     lang                   0
     likes                  0
     retweets               0
     sentiment              0
     text                   0
     dtype: int64
```

Figure 74: Check missing values

4. Check the data types as shown in Fig.75.

```
[7]: # Check data type of each columns
     print(data.dtypes)

     count                int64
     tweet_id             int64
     timestamp           object
     parent_tweet_id    float64
     user_id              int64
     lang                object
     likes                int64
     retweets             int64
     sentiment          float64
     text                object
     dtype: object
```

Figure 75: Data types

5. Drop the column parent_tweet_id given the low presence of values in this feature (9%).
6. Create df1 by removing duplicates based on tweet_id
7. Filter on English tweets only, these tweets are saved in a new dataframe called df2. This means we can start from using df1, where data has been pre-processed, if we want to work with a different language.
8. Check the period of tweets, they were posted from 1st December 2019 until 14th April 2020.

Steps 5 to 8 are shown in Fig.76.

```
[8]: #Drop column 'parent_tweet_id'
     data = data.drop(['parent_tweet_id'], axis=1)

     #Remove duplicated tweet_id
     df1 = data.drop_duplicates(subset=['tweet_id'])
     print(df1.shape)# 10,799 rows and 9 col.

     (10799, 9)

[9]: #Create df2 with English tweets only
     df2 = df1[df1['lang'] == 'en']
     df2.shape #6,660 tweets in English

[9]: (6660, 9)

[10]: # See remaining columns in df
      df2.columns

[10]: Index(['count', 'tweet_id', 'timestamp', 'user_id', 'lang', 'likes',
             'retweets', 'sentiment', 'text'],
            dtype='object')

[11]: # Period of tweets
      datemin = df2.timestamp.min()
      datemax = df2.timestamp.max()
      print('Tweets from',datemin, 'To ',datemax)

      Tweets from 2019-12-01 03:17:00 To  2020-04-14 23:59:58
```

Figure 76: Create df2 with English tweets, without duplicates and missing values

4.2.2 Exploration of News from The Guardian

Using Python, the dataset of news articles scraped from the webpage was explored.

1. Dataset contains 2 features with text content only: the body of the article and the newspaper name. There are 227 instances in the data among which 158 articles are unique (Fig. 77).

```
[2]: os.chdir('C:/Users/anael/News_data')
     os.getcwd()

     # Importing dataset
     df = pd.read_csv('combined_newsarticles.csv')
     print("Shape of data=>",df.shape)

     Shape of data=> (227, 2)

[3]: # 1. Exploratory Analysis of the data

     # Descriptive statistics
     df.describe() #158 unique articles
```

[3]:

	Article Content	Newspaper
count	227	227
unique	158	1
top	Click over to Google, type in “coronavirus”,...	The Guardian
freq	6	227

Figure 77: News dataset structure

2. There are no missing values. Duplicated articles are dropped from the dataset to retain 158 instances (Fig. 78).

```
[4]: df.head(5)
```

[4]:

	Article Content	Newspaper
0	Feeling overwhelmed by the sheer volume of inf...	The Guardian
1	Anyone in the UK aged five and over with sympt...	The Guardian
2	It is caused by a member of the coronavirus fa...	The Guardian
3	Workplaces pose a high risk of triggering a re...	The Guardian
4	Customers in England may be asked to check in ...	The Guardian

```
[5]: #Check if there are null values in the dataset
     df.isnull().sum()

[5]: Article Content    0
     Newspaper          0
     dtype: int64

[6]: #No Null values. In case we wanted to frop them:
     #df.dropna(inplace=True)

     #Drop duplicated articles (in their unprocessed form)
     from pandas import DataFrame
     df = DataFrame.drop_duplicates(df)
     df.shape

[6]: (158, 2)
```

Figure 78: Check for missing values and drop duplicates

News articles were scraped at a frequency of every 2 or 3 days and between 21st June and 9th July. Given that one news article remains accessible up more than 2 or 3 days in a row, I have dropped duplicates from the dataframe. Duplicates were searched through the unprocessed form of the articles scraped. Text was therefore in its raw form with special and unwanted characters (spaces, etc). All these characteristics of non-processed text data create a proxy to a unique identifier for each article.

4.3 Pre-processing

4.3.1 Tweets Data

The package tweet-preprocessor dedicated to Tweet data was used to clean text feature. This library available in Python facilitates cleaning and handling specific characteristics to Tweet language, it is also possible to parse and tokenize tweets. The documentation is available here https://pypi.org/project/tweet-preprocessor/. It was released on 24th May 2020. With this library, we start with a basic cleaning by removing URL, hashtags, reserved words (RT for retweets, FAV), emojis and smileys.

1. Install the library using Anaconda Prompt and the command pip install tweet_preprocessor (Fig.79).

```
Anaconda Prompt (Anaconda3)

(base) C:\Users\anael>pip install tweet-preprocessor
```

Figure 79: Install library

2. See an example in Fig. 80 of the characters from Tweets it removes. Clean 'text' feature with p.clean(). The cleaned version of Tweets messages from 'text' feature is appended to the df in a new column named 'p_processed_text' (Fig.81).

```
[90]: example =('Preprocessor is :) #awesome 👍 !@?-= https://github.com/s/preprocessor')
      p.clean(example)

[90]: 'Preprocessor is !?-='
```

Figure 80: Example of string cleaned with preprocessor

```
[12]: ### Data cleaning
      # 1. With preprocessor library, designed for cleaning tweets

      # Loop over 'text' feature to clean
      tweets = df2.text
      p_processed_text = []
      for tweet in tweets:
          p_processed_text.append(p.clean(tweet))
      df2["p_processed_text"] = p_processed_text
      print (df2.head())

   count             tweet_id            timestamp             user_id lang  \
0      1  1209148016722169856  2019-12-23 16:25:26           301646116   en
2      3  1201998948950577152  2019-12-03 22:57:36  989521438825746433   en
3      4  1200977067266990080  2019-12-01 03:17:00          2414645220   en
6      7  1206491324389756928  2019-12-16 08:28:41          2591058223   en
9     10  1205223525382049792  2019-12-12 20:30:55          2414645220   en

   likes  retweets  sentiment  \
0     73        11   0.000000
2     17         1   0.000000
3      0         0   0.000000
6      5         2   0.034091
9      4         0   0.000000

                                                text  \
0  My surprise when I discovered that two roman c...
2  Amy s a survivor! #bariclab #pnnl #movingon #c...
3  A review of asymptomatic and sub clinical Midd...
6  New in @TheLancetPlanet  Enzootic patterns of ...
9  Molecular mechanism for antibody dependent enh...

                                    p_processed_text
0  My surprise when I discovered that two roman c...
2                                Amy s a survivor!
3  A review of asymptomatic and sub clinical Midd...
6  New in Enzootic patterns of Middle East respir...
```

Figure 81: Clean 'text' from Tweets and create a new column containing the output

3. The “cleaned” text obtained after applying the package ‘preprocessor’ is deemed insufficient when verifying one tweet (Fig.82). Therefore, further cleaning steps are required.

```
[13]: #Check one tweet to verify the quality of cleaning
      df2['p_processed_text'][3] # Manual cleaning needed to improve

[13]: 'A review of asymptomatic and sub clinical Middle East Respiratory Syndrome Infections article/doi/10.1093/epirev/mxz009/5634013.XeMwoBj5e
      W4.twitter'
```

Figure 82: Example of tweet low cleaning quality

Cleaning tasks carried out through a sequence of steps using regular expressions are detailed below in Fig 83 and Fig. 84.

4. Remove URL
5. Convert to lowercase
6. Remove digits and words containing digits
7. Remove extra spaces
8. Remove punctuations
9. Remove stopwords using genism list of words

```
[14]: # 2. Further cleaning instructions
      # Using regular expressions and lambda functions

      # 2.1 Remove URL
      df2['cleaned']=df2['p_processed_text'].apply(lambda x: re.sub(r"http\S+",'', x))

      #2.2 convert text to lowercase with lower() function
      df2['cleaned']=df2['cleaned'].apply(lambda x: x.lower())

      #2.3 Remove digits and words containing digits
      df2['cleaned']=df2['cleaned'].apply(lambda x: re.sub('\w*\d\w*','', x))

      #2.4 Remove Punctuations
      df2['cleaned']=df2['cleaned'].apply(lambda x: re.sub('[%s]' % re.escape(string.punctuation), '', x))

      #2.5 Removing extra spaces
      df2['cleaned']=df2['cleaned'].apply(lambda x: re.sub(' +',' ',x))

      # 2.6 Remove Punctuations
      df2['cleaned']=df2['cleaned'].apply(lambda x: re.sub('[%s]' % re.escape(string.punctuation), '', x))

      #Check the same tweet
      df2['cleaned'][3]
```

Figure 83: Cleaning functions

```
[15]: # 2.7 Remove stopwords from 'cleaned' using gensim (preferred to nltk as gensim provides a longer list of words to exclude, i.e. would, cou

      # Create list of stopwords using gensim library
      stop = gensim.parsing.preprocessing.STOPWORDS
      #print(stop)

      # Exclude stopwords with Python's list comprehension and pandas.DataFrame.apply.
      df2['cleaned'] = df2['cleaned'].apply(lambda x: ' '.join([word for word in x.split() if word not in (stop)]))
      #print(df2.head(20))

      # 2.8 Remove words of 1 character length
      df2['cleaned'] = df2['cleaned'].apply(lambda x: ' '.join([word for word in x.split()  if len(word)>1]))
```

Figure 84: Remove stopwords

10. Tokenize Tweets using nltk library as shown in Fig.85.

```
[17]: # 3.Tokenise with nltk

      # loop to tokenize 'cleaned' feature
      tweets = df2["cleaned"]
      tokenized_tweet = []
      for tweet in tweets:
          tokenized_tweet.append(word_tokenize(tweet))
      df2["Tokenized_Tweet"] = tokenized_tweet
```

Figure 85: Tokenize Tweets

11. Print the top 5 rows of the dataframe for a visual check of the new features added (Fig. 86).

```
[18]: df2.head(5)
      #print(df2[0:20])
```

[18]:

count	tweet_id	timestamp	user_id	lang	likes	retweets	sentiment	text	p_processed_text	cleaned	Tokenized_Tweet
1	1209148016722169856	2019-12-23 16:25:26	301646116	en	73	11	0.000000	My surprise when I discovered that two roman c...	My surprise when I discovered that two roman c...	surprise discovered roman characters asterix o...	[surprise, discovered, roman, characters, aste...
3	1201998948950577152	2019-12-03 22:57:36	989521438825746433	en	17	1	0.000000	Amy s a survivor! #bariclab #pnnl #movingon #c...	Amy s a survivor!	amy survivor	[amy, survivor]
4	1200977067266990080	2019-12-01 03:17:00	2414645220	en	0	0	0.000000	A review of asymptomatic and sub clinical Midd...	A review of asymptomatic and sub clinical Midd...	review asymptomatic sub clinical middle east r...	[review, asymptomatic, sub, clinical, middle, ...
7	1206491324389756928	2019-12-16 08:28:41	2591058223	en	5	2	0.034091	New in @TheLancetPlanet Enzootic patterns of ...	New in Enzootic patterns of Middle East respir...	new enzootic patterns middle east respiratory ...	[new, enzootic, patterns, middle, east, respir...
10	1205223525382049792	2019-12-12 20:30:55	2414645220	en	4	0	0.000000	Molecular mechanism for antibody dependent enh...	Molecular mechanism for antibody dependent enh...	molecular mechanism antibody dependent enhance...	[molecular, mechanism, antibody, dependent, en...

Figure 86: Output for the first 5 rows

12. Lemmatize tokens (Fig.87).

```
[20]: # 4.Lemmatization

      wordnet_lemmatizer = WordNetLemmatizer()

      lemmatized_text = []

      for index, row in df2.iterrows():
          lemma_article = []
          row = row['Tokenized_Tweet']
          for w in row:
              word1 = wordnet_lemmatizer.lemmatize(w, pos = "n")
              word2 = wordnet_lemmatizer.lemmatize(word1, pos = "v")
              word3 = wordnet_lemmatizer.lemmatize(word2, pos = ("a"))
              lemma_article.append(word3)
          lemmatized_text.append(lemma_article)

      print(lemmatized_text)
      df2['lemmatized_text'] = lemmatized_text # create new col. in df with lemmatized output
```

Figure 87: Lemmatization

13. Rename the column ‘sentiment’, keep required features and export cleaned data as a .csv file named “export_Tweets_df2.csv” (Fig.88).

```
[22]: # Rename col. sentiment
df2.rename(columns={'sentiment': 'sentiment1'}, inplace=True)

# Tidy up the df
df2 =df2[['tweet_id', 'timestamp', 'user_id', 'lang', 'likes', 'retweets', 'sentiment1', 'text', 'cleaned', 'lemmatized_text']]

#Export df to .csv file for further visual checks before Exploratory Analysis and Implementation
df2.to_csv (r'C:\Users\anael\export_Tweets_df2.csv', index = False, header=True)
```

Figure 88: Tidy and export data

The output from this pre-processing operation is a dataframe containing the tweets original features, the clean text without stop words and lemmatized tokens in column “lemmatized_text”. It will be used as a source file in RStudio and Python at later stages in Exploratory Analysis and Implementation.

4.3.2 News Articles

Scraped articles body from The Guardian/coronavirus outbreak page at several points of time. Data was saved in csv files and merged into a single dataset.

Text must be normalised. The approach from Abhishek Sharma[14] was used for guidance on the Exploratory Data Analysis on text data, from Joyce Annie George[15] for NLP techniques and from Shubham Singh[16] for guidance on the text normalization steps and using NLTK with Python.The pre-processing will consist in cleaning the text feature to remove unwanted characters and do an exploratory analysis using Python.

1. Import libraries (Fig.89).

```
### News Artciles from The Guardian

import numpy as np
import pandas as pd
# to drop duplicates:
from pandas import DataFrame
# For regular expressions
import re
# For handling string
import string
# For tokenization and lemmatization
import nltk
from nltk.tokenize import word_tokenize # Method to split a sentence into tokens or words
from nltk.stem import WordNetLemmatizer # For lemmatization
# To remove stopwords
import gensim
# To select Operating system
import os
```

Figure 89: Import libraries

[14]https://www.analyticsvidhya.com/blog/2020/04/beginners-guide-exploratory-data-analysis-text-data/#2
[15]https://medium.com/analytics-vidhya/fake-news-detection-using-nlp-techniques-c2dc4be05f99
[16]https://www.analyticsvidhya.com/blog/2019/08/how-to-remove-stopwords-text-normalization-nltk-spacy-gensim-python/

2. Fig. 90 prints the data in “Article Content” feature.

```
[7]: # 2. Basic pre-processing for Cleaning Text
     #ex: remove null value imputation and removal of unwanted data.
     #https://www.analyticsvidhya.com/blog/2020/04/beginners-guide-exploratory-data-analysis-text-data/#2

     # See text in this feature
     df['Article Content'].unique()

[7]: array(['Feeling overwhelmed by the sheer volume of information on coronavirus? Help is at hand. Our journalists will explain the week’s
     events and give you some facts you can count on, even in this constantly changing situation. No sign-up button? Users viewing this page
     via Google Amp may experience a technical fault. Please click here to reload the page which should correct the problem. *** The Guardia
     n’s newsletters include content from our website, which may be funded by outside parties. Newsletters may also display information about
     Guardian News and Media’s other products, services or events (such as Guardian Jobs or Masterclasses), chosen charities or online advert
     isements.',
            'Anyone in the UK aged five and over with symptoms is now eligible for a coronavirus test. The Department for Health and Social C
     are says it is currently unsuitable for children under five. The options are to get a drive-through appointment or a home-testing kit bu
     t either way they involve a similar procedure. On 18 May, Matt Hancock announced coronavirus testing is being extended in the UK to anyo
     ne over the age of five with symptoms. Before then,\xa0eligibility had been limited\xa0to a series of groups including key workers, thos
     e aged over 65, people who could not work from home, or people who lived with someone from one of these groups. Those with symptoms can
```

Figure 90: Visual inspection

3. Replace the string “\xa0” by a space between words and replace special characters (- ; “ ; ” ; ’) by blank (Fig.91).

```
[10]: #A lambda function can take any number of arguments, but can only have one expression
      #df['Article Content']=df['Article Content'].apply(lambda x: x.split('\xa0',0))

      # 2.1. Use the vectorised str method to replace:
      df['Article Content'] =df['Article Content'].str.replace('\xa0',' ')

      # look again how text is in this column
      df['Article Content'].unique()

[10]: array(['Feeling overwhelmed by the sheer volume of information on coronavirus? Help is at hand. Our journalists will explain the week’s
      events and give you some facts you can count on, even in this constantly changing situation. No sign-up button? Users viewing this page
      via Google Amp may experience a technical fault. Please click here to reload the page which should correct the problem. *** The Guardia
      n’s newsletters include content from our website, which may be funded by outside parties. Newsletters may also display information about
      Guardian News and Media’s other products, services or events (such as Guardian Jobs or Masterclasses), chosen charities or online advert
      isements.',
             'Anyone in the UK aged five and over with symptoms is now eligible for a coronavirus test. The Department for Health and Social C
      are says it is currently unsuitable for children under five. The options are to get a drive-through appointment or a home-testing kit bu
      t either way they involve a similar procedure. On 18 May, Matt Hancock announced coronavirus testing is being extended in the UK to anyo
      ne over the age of five with symptoms. Before then, eligibility had been limited to a series of groups including key workers, those aged
      over 65, people who could not work from home, or people who lived with someone from one of these groups. Those with symptoms can now use

[11]: # 2.2 Replace " " by nothing
      df['Article Content'] =df['Article Content'].str.replace("“","")
      df['Article Content'] =df['Article Content'].str.replace("”","")
      df['Article Content'] =df['Article Content'].str.replace("–","")

      # Replace ’ by apostrophe
      df['Article Content'] =df['Article Content'].str.replace("’","'")
```

Figure 91: Search and replace unwanted characters

Data was visually checked to spot strange characters. Other characters were removed as shown in Fig. 91 as they stood out during the exploration analysis after the initial cleaning operation. No further text replacement was deemed necessary given the text seen.

4. Expand contractions such as “wouldn’t” becomes “would not” (Fig.92).

```
[13]: # 2.3 Expand Contractions

# Dictionary of English Contractions
contractions_dict = { "ain't": "are not","'s":" is","aren't": "are not",
                     "can't": "cannot","can't've": "cannot have",
                     "'cause": "because","could've": "could have","couldn't": "could not",
                     "couldn't've": "could not have", "didn't": "did not","doesn't": "does not",
                     "don't": "do not","hadn't": "had not","hadn't've": "had not have",
                     "hasn't": "has not","haven't": "have not","he'd": "he would",
                     "he'd've": "he would have","he'll": "he will", "he'll've": "he will have",
                     "how'd": "how did","how'd'y": "how do you","how'll": "how will",
                     "I'd": "I would", "I'd've": "I would have","I'll": "I will",
                     "I'll've": "I will have","I'm": "I am","I've": "I have", "isn't": "is not",
                     "it'd": "it would","it'd've": "it would have","it'll": "it will",
                     "it'll've": "it will have", "let's": "let us","ma'am": "madam",
                     "mayn't": "may not","might've": "might have","mightn't": "might not",
                     "mightn't've": "might not have","must've": "must have","mustn't": "must not",
                     "mustn't've": "must not have", "needn't": "need not",
                     "needn't've": "need not have","o'clock": "of the clock","oughtn't": "ought not",
                     "oughtn't've": "ought not have","shan't": "shall not","sha'n't": "shall not",
                     "shan't've": "shall not have","she'd": "she would","she'd've": "she would have",
                     "she'll": "she will", "she'll've": "she will have","should've": "should have",
                     "shouldn't": "should not", "shouldn't've": "should not have","so've": "so have",
                     "that'd": "that would","that'd've": "that would have", "there'd": "there would",
                     "there'd've": "there would have", "they'd": "they would",
                     "they'd've": "they would have","they'll": "they will",
                     "they'll've": "they will have", "they're": "they are","they've": "they have",
                     "to've": "to have","wasn't": "was not","we'd": "we would",
                     "we'd've": "we would have","we'll": "we will","we'll've": "we will have",
                     "we're": "we are","we've": "we have", "weren't": "were not","what'll": "what will",
                     "what'll've": "what will have","what're": "what are", "what've": "what have",
                     "when've": "when have","where'd": "where did", "where've": "where have",
                     "who'll": "who will","who'll've": "who will have","who've": "who have",
                     "why've": "why have","will've": "will have","won't": "will not",
                     "won't've": "will not have", "would've": "would have","wouldn't": "would not",
                     "wouldn't've": "would not have","y'all": "you all", "y'all'd": "you all would",
                     "y'all'd've": "you all would have","y'all're": "you all are",
                     "y'all've": "you all have", "you'd": "you would","you'd've": "you would have",
                     "you'll": "you will","you'll've": "you will have", "you're": "you are",
                     "you've": "you have"}

# Regular expression for finding contractions
contractions_re=re.compile('(%s)' % '|'.join(contractions_dict.keys()))

# Function for expanding contractions
def expand_contractions(text,contractions_dict=contractions_dict): #function expects 2 arguments: text + contractions_dict
  def replace(match):
    return contractions_dict[match.group(0)]# The 1 in match.group(1) represents the first parenthesised subgroup. 0 here is for the entir
  return contractions_re.sub(replace, text)

# Expanding Contractions in the reviews
df['Article Content']=df['Article Content'].apply(lambda x:expand_contractions(x))

[14]: df['Article Content'][4]

has been lobbying the government to relax the 2-metre rule, warning that it would make many businesses uneconomic even when they are all
owed to reopen  and trigger a fresh wave of job losses. Case is expected to conclude that the blanket 2-metre restriction can be lifted,
and replaced with a 1-metre plus rule, that would allow the public to be closer together, if other safety precautions are taken. Asked a
bout that idea, Hancock said: That is the sort of thing we are looking at to how do you make it safe to open things  and as safe as poss
ible, to open as much as possible. Things like wearing a face mask, which reduces the transmission, clearly; how the seating is arranged
because face-to-face is much more dangerous than back-to-back, and there is much more transmission than side-to-side. Asked whether hair
dressers would be included in the reopening plans, Hancock said: A lot of the country does need a haircut: we need to do that in a safe
way. People over 60 or with health issues should wear a medical-grade mask when they are out and cannot socially distance, according to
new guidance from the World Health Organization, while all others should wear a three-layer fabric mask. The WHO guidance, announced on
5 June, is a result of research commissioned by the organisation. It is still unknown whether the wearers of masks are protected, say it
s experts, but the new design it advocates does give protection to other people if properly used. The WHO says masks should be made of t
hree layers  with cotton closest to the face, followed by a polypropylene layer and then a synthetic layer that is fluid-resistant. Thes
```

Figure 92: Function for expanding contractions

Fig. 93 illustrates steps 5 to 8:

5. Convert text to lowercase using the dedicated function available in Python. A visual check is shown by printing article index 4.

6. Remove digits and words containing digits because these will not be used for analysis as the focus is on Natural language processing and text analysis.

7. Remove punctuations using string.punctuations function and a regular expression to search and remove them from text. Punctuation is important for English grammar but not for text analysis. We will remove marks such as commas, hyphens, full stops, etc.

8. Then we have to remove the extra spaces present in the data. This is because one space was added when removing punctuations and digits from the text.

```
[15]: #2.4 convert text to lowercase with lower() function
df['cleaned']=df['Article Content'].apply(lambda x: x.lower())

#2.5 Remove digits and words containing digits
df['cleaned']=df['cleaned'].apply(lambda x: re.sub('\w*\d\w*','', x))

#2.6 Remove Punctuations
df['cleaned']=df['cleaned'].apply(lambda x: re.sub('[%s]' % re.escape(string.punctuation), '', x))

# 2.7 Removing extra spaces
df['cleaned']=df['cleaned'].apply(lambda x: re.sub(' +',' ',x))

[16]: df['cleaned'][4]

[16]: 'customers in england may be asked to check in when they arrive at pubs and restaurants as part of the the government is plan for reopen
ing the hospitality sector matt hancock has said appearing on sophy ridge on sunday on skynews the health secretary confirmed the govern
ment hoped to reopen pubs and restaurants on july in line with boris johnson is roadmap published last month in that plan it states that
on around july we will take further measures if it is safe to do so we talk about hospitality and outdoor hospitality in those plans he
said adding we are clearly on track for that plan asked about reports that ministers are considering plans to ask diners and drinkers to
register as they enter a venue he said i would not rule that out there are other countries in the world that take that approach in new z
ealand the public use their phones to scan codes as they go into hospitality outlets to build up a digital diary of where they have been
so that if a new case emerges anyone who has been at the same outlet can be contacted easily johnson is review into the metre social dis
tancing rule is expected to conclude this week perhaps as early as tuesday carried out by the no permanent secretary simon case it has c
onsidered scientific advice but also the impact of the rule on the economy the hospitality industry has been lobbying the government to
relax the metre rule warning that it would make many businesses uneconomic even when they are allowed to reopen and trigger a fresh wave
```

Figure 93: Cleaning lambda functions

9. Tokenize articles into words using nltk library and word_tokenize module (Fig.94).

```
[17]: # 3.Preparing Text Data for analysis and implementation

# 3.1 Tokenise
df['tokenized_text'] = df['cleaned'].apply(word_tokenize)
df.head(5)
```

[17]:

	Article Content	Newspaper	cleaned	tokenized_text
0	Feeling overwhelmed by the sheer volume of inf...	The Guardian	feeling overwhelmed by the sheer volume of inf...	[feeling, overwhelmed, by, the, sheer, volume,...
1	Anyone in the UK aged five and over with sympt...	The Guardian	anyone in the uk aged five and over with sympt...	[anyone, in, the, uk, aged, five, and, over, w...
2	It is caused by a member of the coronavirus fa...	The Guardian	it is caused by a member of the coronavirus fa...	[it, is, caused, by, a, member, of, the, coron...
3	Workplaces pose a high risk of triggering a re...	The Guardian	workplaces pose a high risk of triggering a re...	[workplaces, pose, a, high, risk, of, triggeri...
4	Customers in England may be asked to check in ...	The Guardian	customers in england may be asked to check in ...	[customers, in, england, may, be, asked, to, c...

Figure 94: Tokenize articles

10. Stopwords are the most common words of a language, they do not add meaning to a document and rather dilute meaningful words in its content. In order to reduce the dataset size and focus on important words I remove words such as: ‘I’, ‘this’, ‘is’, ‘in’. Remove stopwords using genism list of words (this list was selected because it

contains more words than nltk stopwords list, and text data was not deemed cleaned enough using nltk stopwords after the first pre-processing operation) (Fig.95).

```
[18]: # 3.2 Remove stopwords using gensim

      # Set of stop words
      stop_words = gensim.parsing.preprocessing.STOPWORDS
      #print(stop_words)

      filtered_sentences = []

      for index, row in df.iterrows():
          filtered_article = []
          row = row['tokenized_text']
          for w in row:
              if w not in stop_words:
                  filtered_article.append(w)
          filtered_sentences.append(filtered_article)

      #print(filtered_sentences)
      df['text_without_stopwords'] = filtered_sentences # create new col. in df with output
```

Figure 95: Remove stopwords

11. Lemmatize tokens (Fig.96). The technique of lemmatization consists in reducing a word token to its lemma using the appropriate part-of-speech (POS) tag. It is a process to stem words to their base form. A simple lemmatization was initially carried out on Tweets but it is less precise because the same word can have a multiple lemmas based on the meaning or context. Therefore lemmatizing with POS tag is a more suitable technique to take into account the semantic[17].

```
[19]: # 3.3 Lemmatization

      wordnet_lemmatizer = WordNetLemmatizer()

      lemmatized_text = []

      for index, row in df.iterrows():
          lemma_article = []
          row = row['text_without_stopwords']
          for w in row: # lemma_article must be a list of tokens => use df.iterrows() again
              word1 = wordnet_lemmatizer.lemmatize(w, pos = "n")
              word2 = wordnet_lemmatizer.lemmatize(word1, pos = "v")
              word3 = wordnet_lemmatizer.lemmatize(word2, pos = ("a"))
              lemma_article.append(word3)
          lemmatized_text.append(lemma_article)

      #print(lemmatized_text)
      df['lemmatized_text'] = lemmatized_text # create new col. in df with output
      # Create column ['lemmatized'] for Exploratoty Analysis and Implementation

[20]: #Export df to .csv file for further visual checks
      df.to_csv (r'C:\Users\anael\export_NewsArticles.csv', index = False, header=True)
```

Figure 96: Lemmatization

Dataframe of pre-processed news articles is exported in a .csv file for visual checks and will be used for Exploratory Analysis and Implementation.

[17]https://www.machinelearningplus.com/nlp/lemmatization-examples-python/#spacylemmatization

5 Exploratory Data Analysis

An extensive Exploratory Analysis was done using RStudio and Python to discover the content of text features analysed in the two datasets, assess and improve pre-processing when needed, and support the interpretation of results obtained from the implementation of unsupervised models (details in section 6. Implementation).

5.1 Tweets Words Exploration

5.1.1 Wordclouds and Frequencies

The .csv file named "export_Tweets_df2.csv" is read as a dataframe called "tweets" in RStudio. The feature "lemmatized_text" contains pre-processed text and is used for building worclouds and analyzing word frequencies. This tool depicts the most frequent and common words used in a corpus of text. The bigger and the bolder a word is displayed, the higher frequency it has in the corpus. Fig. 97 shows the creation of a corpus necessary to design the wordcloud. In RStudio the two packages "tm" and "wordcloud" must be installed for this purpose.

The first wordcloud displayed in Fig. 97 is built with 50 words and frequency of at least 100 times.

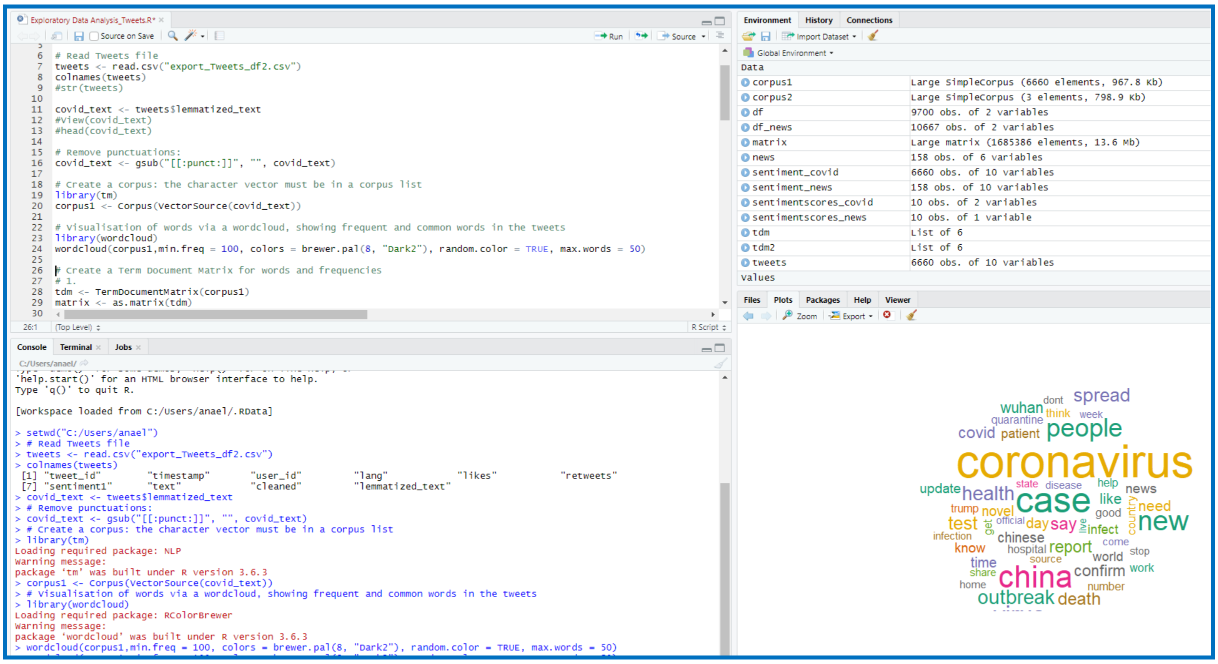


Figure 97: Tweets word frequencies and cloud

Other settings are tested to build several versions. A wordcloud with 100 words and words frequencies higher than 100 is shown in Fig. 98. Fig. 99 displays a less dense wordcloud with a maximum of 100 words with frequencies higher than 200. We see less words being displayed, there are 34 words with a minimum frequency of 200 times as shown in Fig. 100.

We can see that "coronavirus" is more present than "covid" in the text. The most recurring words are more easily visible in the wordcloud displaying frequency above 200 times (Fig. 98), which is less clunky.

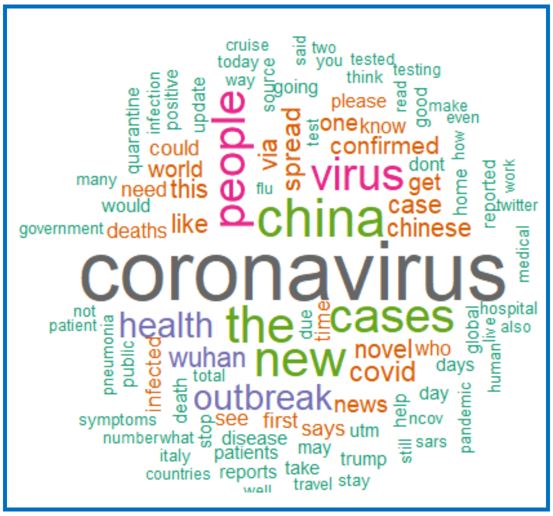


Figure 98: Wordcloud (100 words, freq>200) **Figure 99: Wordcloud (100 words, freq>100)**

```
> findFreqTerms(tdm, lowfreq = 200)
 [1] "new"      "virus"    "china"     "health"    "infected"  "novel"   "outbreak"    "via"
 [9] "say"      "update"   "chinese"   "one"       "patient"   "repo"    "coronavirus" "wuhan"
[17] "case"     "people"   "world"     "confirmed" "get"       "news"    "like"        "know"
[25] "test"     "need"     "spread"    "time"      "country"   "covid"   "day"         "death"
[33] "first"    "see"
```

Figure 100: Words occurrence > 200 times

Then I create a TermDocumentMatrix containing all the words from "corpus1". I create a dataframe "df" with these words and their respective frequency, they are ordered by descending order of frequency. The first 10 words with the highest frequency are listed in the console in Fig. 101. I also plot word frequencies on a bar chart with the top 20 word frequencies on the right-hand side of Fig.101.

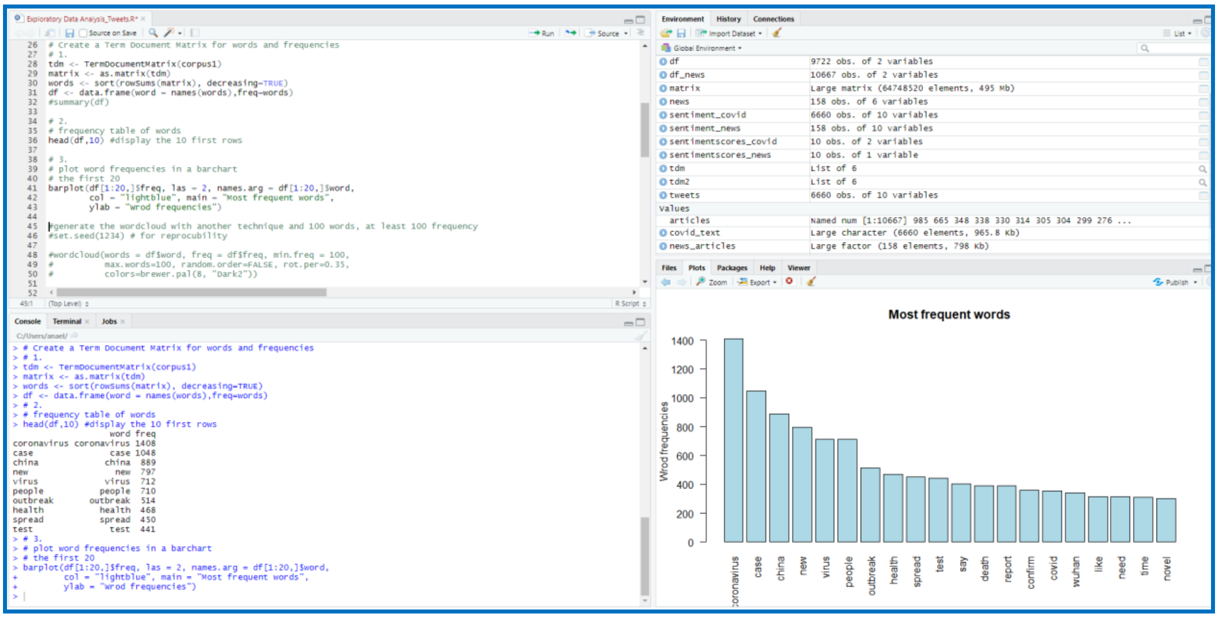


Figure 101: TDM for tweets word frequencies

When looking at the df ranking words by descending order, it confirms what we see in the wordcloud and gives more details with the exact frequency associated to each word. Fig. 102 shows the 50 words with the highest frequency.

```
> head(df,50) #disply the 50 first rows
                   word freq
coronavirus coronavirus 1351
china             china  822
the                 the  775
new                 new  768
cases             cases  714
virus             virus  672
people           people  664
outbreak       outbreak  486
health           health  448
wuhan             wuhan  338
covid             covid  334
spread           spread  310
novel             novel  299
like               like  293
chinese         chinese  290
case               case  288
via                 via  286
this               this  283
confirmed     confirmed  276
news               news  252
```

Figure 102: Excerpt of the 50 most frequent words (only 20 rows displayed in Console)

5.1.2 Frequent Words Associations

Explore association between frequent terms with findAssocs() function. With the examples of "virus", "government" and "people" in Fig. 103. We see that "virus" is strongly associated (greater than 0.15) with "pleasefollow" with a score of 0.46 and "click" with 0.39. This result shows a poor example of association of a word that seems to be key in the analysis of the Covid virus.

```
> # Explore association between frequent terms
> findAssocs(tdm, terms = "virus", corlimit = 0.15)
$virus
pleasefollow        click     breaking         read         live       source       corona      mystery         sars
        0.46         0.39         0.28         0.24         0.23         0.22         0.19         0.18         0.18
       china         news     newsthe
        0.15         0.15        0.15
```

Figure 103: Words associated with "virus"

In the examples with "government" and "people" (Fig.104), the correlation limits are set down to 0.1 to retrieve the lists of frequently associated words.

```
> findAssocs(tdm, terms = "government", corlimit = 0.1)
$government
     guangzhou         citizen       hoursdont          humble       jharkhand          murillo            oega       organizes
          0.17            0.16            0.16            0.16            0.16             0.16            0.16            0.16
         eamon       formation       behaviour        cultural   foreknowledge            mixed         iranian           drill
          0.16            0.16            0.16            0.16            0.16             0.14            0.12            0.11
         lying    totalitarian         welding         exploit   authoritarian             ryan      benevolent           relay
          0.11            0.11            0.11            0.11            0.11             0.11            0.11            0.11
      suppocmp
          0.11

> findAssocs(tdm, terms = "people", corlimit = 0.1)
$people
corporation        died    hardship       alike    conspire   conspired      oveurn       fewer         put
       0.17        0.15        0.15        0.14        0.14        0.14        0.14        0.12        0.11
       many         die       focus     reoccur     comrade      ethnic  resolutely       unite     cookout
       0.10        0.10        0.10        0.10        0.10        0.10        0.10        0.10        0.10
   observed      arisen
       0.10        0.10
```

Figure 104: Association score with "government" and "people"

5.2 News Words Exploration

The same word exploratory analysis is performed on news articles to have a similar comparison basis between the two datasets.

5.2.1 Wordclouds and Frequencies

Open the .csv file "export_NewsArticles.csv" and read the feature “text_lemmatized” as vector named “news_articles”. The corpus2 is created to build wordclouds and explore word frequencies (Fig.105). This wordcloud displays a maximum of 50 words with frequencies above 100 times.

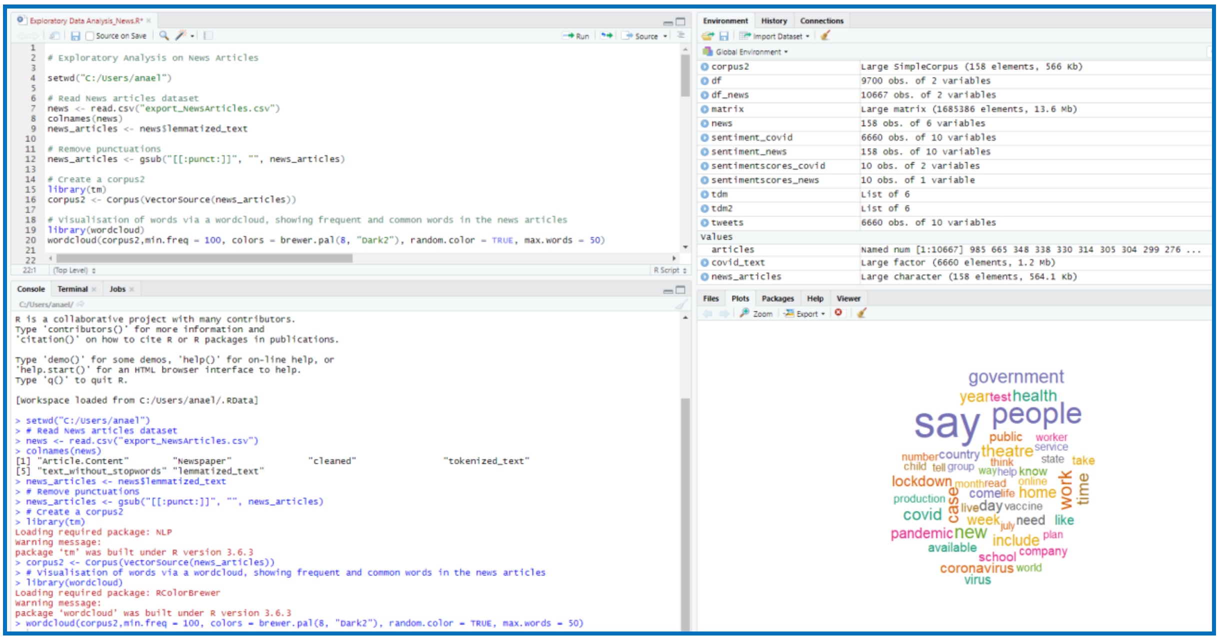


Figure 105: News word frequencies and cloud

Wordclouds with a maximum of 100 words displayed and frequencies greater than 200 and greater than 100 are built and shown in Fig. 106 and Fig.107.

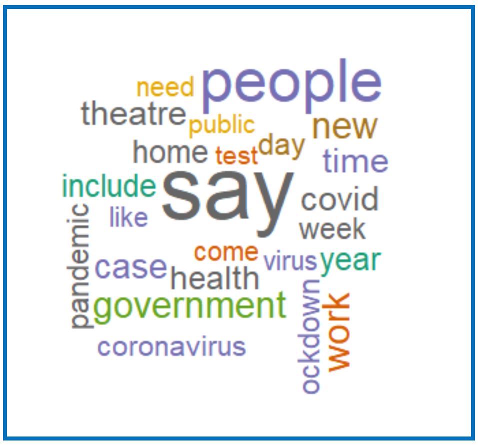


Figure 106: Wordcloud News (freq>200)

Figure 107: Wordcloud News (freq>100)

The TermDocumentMatrix is created to evaluate the word frequencies. Words and their associated frequency are saved in a dataframe where they are sorted by descending order. The top 10 words with the highest frequency are listed in the console. The top 20 frequent words are plotted on a bar chart for visualization (Fig.108).

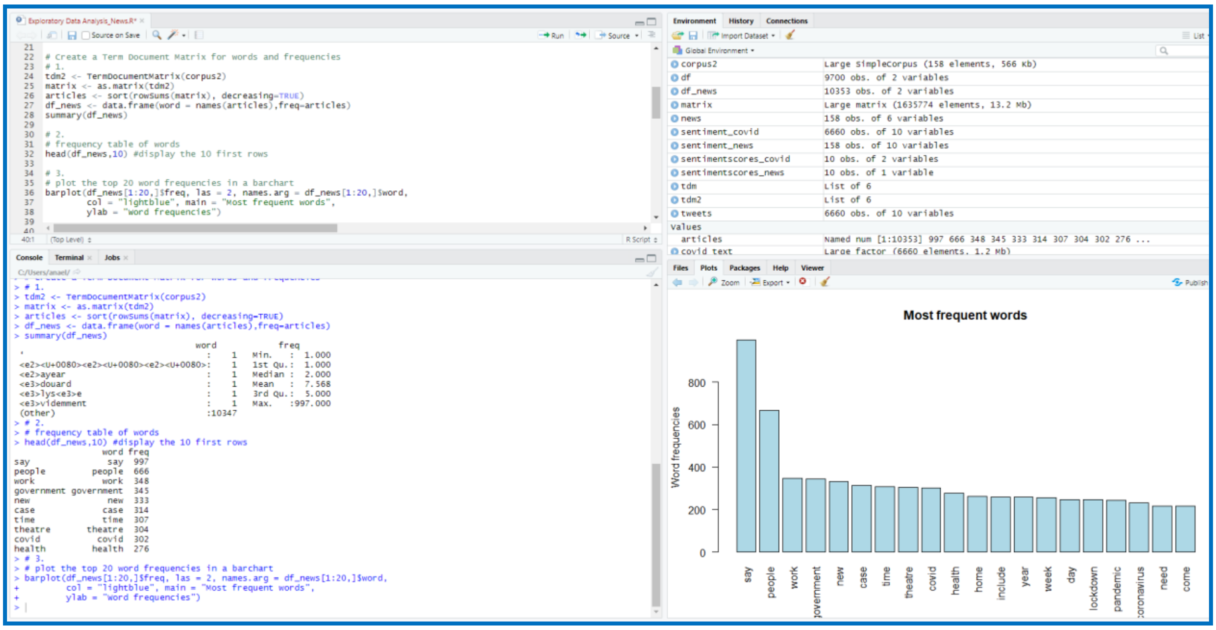


Figure 108: TDM for news word frequencies

When displaying the top of this df with summary() in Fig. 108, we see the text data is not perfectly clean and there are still strange characters and/or words appearing at the top of the list. The issue comes from the type of text typed in the original articles content and scraped from the internet. It happens that typos and unwanted characters remained and not 100% of the tokens are sensible words in the English language.

Text data had initially been pre-processed in Python, and the cleaned data obtained has been cleaned a second time with RStudio for testing and ensuring the content was as tidy as possible. Same results were obtained at this stage when verifying the summary(news_df). It was decided to go ahead with this data as it is the best that can be obtained given the two pre-processing operations done using two tools, the coding skills and timeframe for the research.

24 words appear at least 200 times in the matrix, in Fig. 109 and compare with the output from tweets in Fig. 100 (page 47).

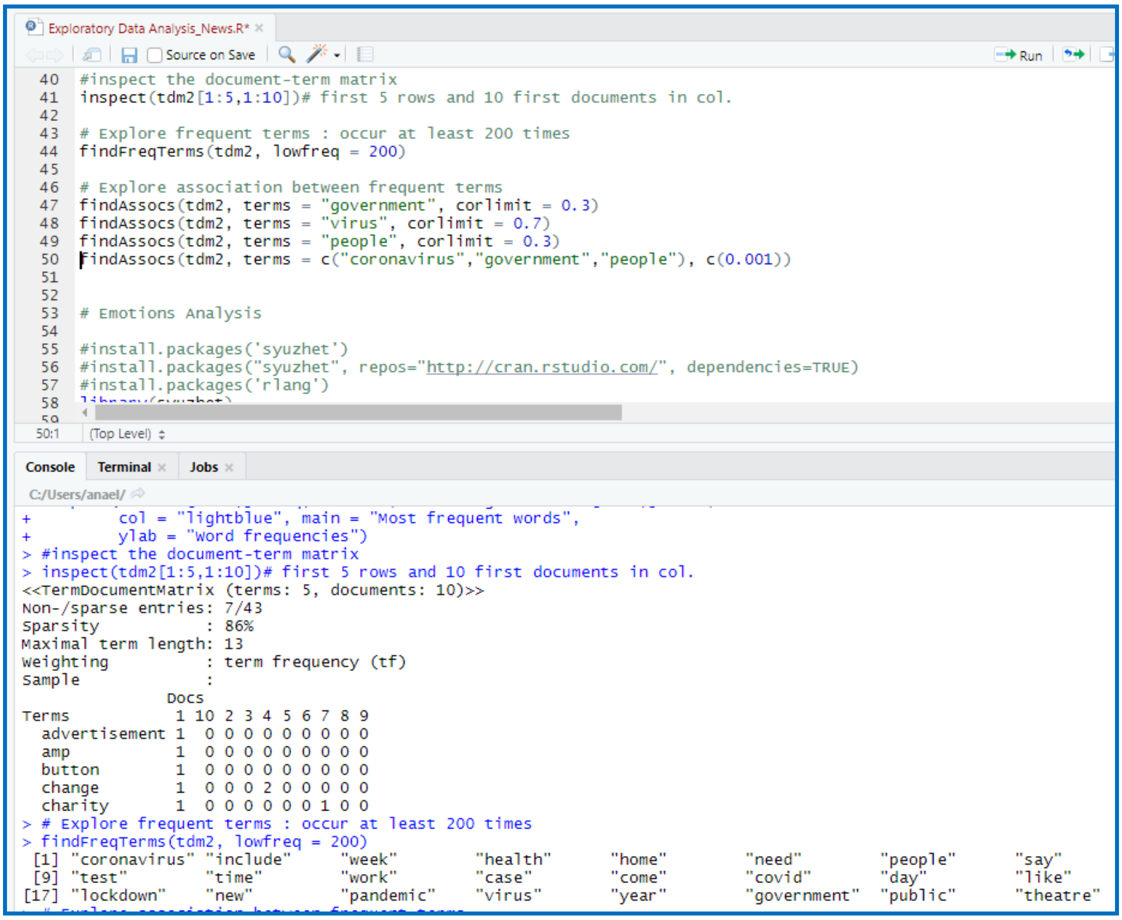


Figure 109: Word occurrence > 200 times

5.2.2 Frequent Words Associations

The same three words associations as for tweets will be compared to identify the correlation similarities or differences.

In news articles, the word "government" has numerous associations with correlations greater than 0.3, on the opposite from tweets where the highest correlation was 0.17 and "Guanzhou" (Fig.110).

```
> findAssocs(tdm2, terms = "government", corlimit = 0.3)
$government
       labour          raise       announce      secretary         afford       minister           need
         0.43           0.40           0.39           0.38           0.37           0.37           0.36
         risk          chief       solution         arrive           lack         public        johnson
         0.36           0.36           0.36           0.35           0.35           0.34           0.34
        blame   increasingly           tory        failure       chairman        provide           mean
         0.34           0.34           0.34           0.33           0.33           0.32           0.32
    emergency         sector         brexit        startle   conservative          blair           tony
         0.32           0.32           0.32           0.32           0.32           0.32           0.32
      england         extend           home         remain            cut      transport        council
         0.31           0.31           0.31           0.31           0.31           0.31           0.31
       regard      addiction       antidote    approachaby      belatedly    comfortably  conservatism
         0.31           0.31           0.31           0.31           0.31           0.31           0.31
         cosy        disdain disintegration     distortion    electorally     electorate      enjoyment
         0.31           0.31           0.31           0.31           0.31           0.31           0.31
         gove      heartland   identifiable     impression   infantilising        killjoy         lawyer
         0.31           0.31           0.31           0.31           0.31           0.31           0.31
    marginals  nightwatchman      oakeshott optimistically   participation      pensioner  preoccupation
         0.31           0.31           0.31           0.31           0.31           0.31           0.31
  prosecution         rarely      readiness   recklessness   republicanrun         revere    rightwingers
         0.31           0.31           0.31           0.31           0.31           0.31           0.31
   supposedly    temperament          terse     unrewarded   wolverhampton      worldview           week
         0.31           0.31           0.31           0.31           0.31           0.31           0.30
       social        deliver     distribute          solve    preemptively      immigrant       disorder
         0.30           0.30           0.30           0.30           0.30           0.30           0.30
```

Figure 110: Word association with "government"

In news articles, the word “virus” has significant correlations. The parameter is set at 0.7 to retrieve a short list only, it will be sufficient for comparison purpose (Fig.111).

```
> findAssocs(tdm2, terms = "virus", corlimit = 0.7)
$virus
coronaviruses          iowa       perlman          sars     circulate        evolve     evolution
         0.79          0.79          0.79          0.78          0.75          0.73          0.72
```

Figure 111: Word association with "virus"

The association score of “virus” in tweets started at 0.46 with “please =follow” and 0.39 with “click”, and then “breaking, “read”, “live” … these words seem irrelevant to gain insight on the virus. The high association scores obtained for “virus” in the news articles are more relevant and shows numerous associations. The top three associations do not give enough insights: “coronaviruses” can be seen as a synonym to the Covid-19 virus and this generate a high correlation between the two terms. Then “iowa” seems to be an outlier in the relation with virus, and “perlman” is proper noun. The next correlations with “sars”, “circulate” and “evolve/evolution” are relevant to the topic of the virus.

The word “people” also shows numerous associations above a correlation score of 0.3 as set in the parameters (Fig.112). Associated words convey actions with verbs, physical people (with the terms “staff”, “colleague”, “patient”, “nurse”, “psychologist”) and the lexical field is mostly related to the medical domain (pneumonia, hospital, transplant, prostate, trial, plague, intravenous, dependency).

```
> findAssocs(tdm2, terms = "people", corlimit = 0.3)
$people
         say          get         hard      illness        think     everyone         know
        0.60         0.53         0.53         0.52         0.51         0.50         0.50
     patient         main         cold         drug          ill     feverish       unwell
        0.50         0.50         0.48         0.48         0.48         0.48         0.47
         lot        chest          die       family          six         sore         felt
        0.47         0.46         0.46         0.46         0.46         0.46         0.46
        fine          day     hospital       grumpy     tolerate        roast          tub
        0.46         0.45         0.45         0.45         0.45         0.45         0.45
       place        start        staff    colleague        covid       arrive         tell
        0.44         0.44         0.44         0.44         0.43         0.43         0.43
      parcel     prostate     telltale apprehensive    overwhelm         stay    pneumonia
        0.43         0.43         0.43         0.43         0.42         0.42         0.42
        weve         ever    hopefully         even         work        nasal        never
        0.42         0.42         0.42         0.41         0.41         0.41         0.41
      happen            ‘       bounce  psychologist         home         come        nurse
        0.41         0.41         0.41         0.41         0.40         0.40         0.40
    remember      sleeper       settle   transplant      gilbert         lion    community
        0.40         0.40         0.40         0.40         0.40         0.40         0.39
       drive      collect       plague       notice        worry        trial  containment
```

Figure 112: Word association with "people"

5.3 Bigrams Analysis

Bigrams analysis shows word associations and gives insights on themes discussed in tweets and news. The article "Explore COVID-19 Infodemic" on Towards Data Science website shows findings from an exploratory analysis on true/fake news articles covering Covid-19[18]. I used this article for comparing findings and a piece of code to extract bigrams using Python (the function get_top_n_bigram() shown in Fig.113.

[18] https://towardsdatascience.com/explore-covid-19-infodemic-2d1ceaae2306

```python
# Import libraries
import pandas as pd
from sklearn.feature_extraction.text import CountVectorizer
```

```python
[6]: # Read tweets dataset
df_tweets = pd.read_csv('export_Tweets_df2.csv')

# Define the function (Li, 2020)
def get_top_n_bigram(corpus, n=None):
    vec = CountVectorizer(ngram_range=(2, 2), stop_words='english').fit(corpus)
    bag_of_words = vec.transform(corpus)
    sum_words = bag_of_words.sum(axis=0)
    words_freq = [(word, sum_words[0, idx]) for word, idx in vec.vocabulary_.items()]
    words_freq =sorted(words_freq, key = lambda x: x[1], reverse=True)
    return words_freq[:n]
```

Figure 113: Function to extract bigrams

Bigrams are extracted from tweets by applying the function to the lemmatized_text feature. The output is shown in Fig. 114 and in Fig. 115 for bigrams from news articles.

```python
# Get top bigrams from tweets
common_words = get_top_n_bigram(df_tweets['lemmatized_text'], 20)
for word, freq in common_words:
    print(word, freq)
```

```
confirm case 166
new case 130
test positive 103
novel coronavirus 88
coronavirus outbreak 87
cruise ship 80
new coronavirus 75
coronavirus case 74
public health 63
death toll 62
world health 54
case coronavirus 53
corona virus 52
utm medium 50
wash hand 49
report new 49
utm source 48
coronavirus spread 46
health official 46
novel ncov 46
```

Figure 114: Bigrams from tweets

```python
[11]: # Read news articles dataset
df_news = pd.read_csv('export_NewsArticles.csv')

# Get top bigrams from news articles
common_words = get_top_n_bigram(df_news['lemmatized_text'], 20)
for word, freq in common_words:
    print(word, freq)
```

```
read review 80
public health 74
prime minister 68
social distance 44
new case 39
young people 37
care home 36
local authority 34
face mask 32
theatre company 31
boris johnson 28
chief executive 27
wear mask 27
number case 25
new york 25
number people 25
world health 23
pub restaurant 23
coronavirus crisis 23
old vic 23
```

Figure 115: Bigrams from news

The interpretation of results is in the technical report section 4.1.2 Bigrams Analysis.

5.4 Emotions Detection

Using RStudio, emotions detection analysis was done to study emotions from Twitter user-generated (public) and journalists, using tweets and news articles separately.

The function get_nrc () calls the NRC sentiment dictionary to calculate the presence of eight emotions and their corresponding prevalence in a text corpus.

5.4.1 Install Packages

We need to install the package syuzhet, a pre-requisite to this is having the right version of rlang package. If the version already installed dis not suitable, there will be an error message as in Fig. 116.

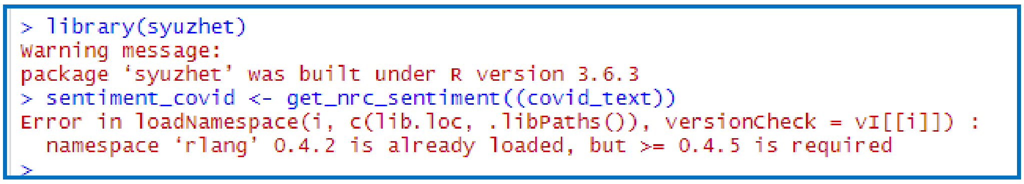


Figure 116: Error message for rlang

If this issue is encountered, update the rlang package already installed by clicking on “Update” button in the Packages view (Fig 117).

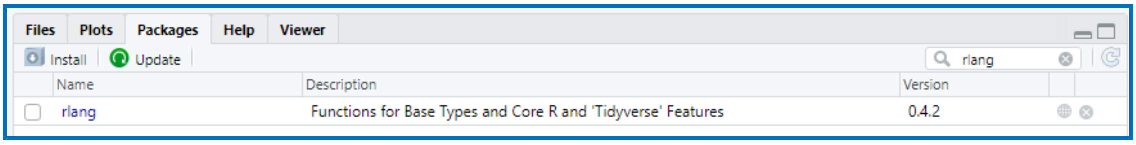


Figure 117: Check version installed and update

Update package 0.4.2 to 0.4.7, the most recent version available and offered at this point in time (Fig. 118).

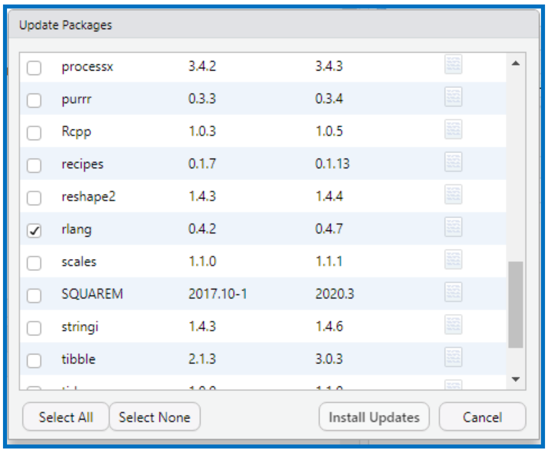


Figure 118: Select package to update

In case of issues to update the package as in Fig. 119, change the properties in RStudio settings to “Run as Administrator” and allow the application to make changes to my machine.

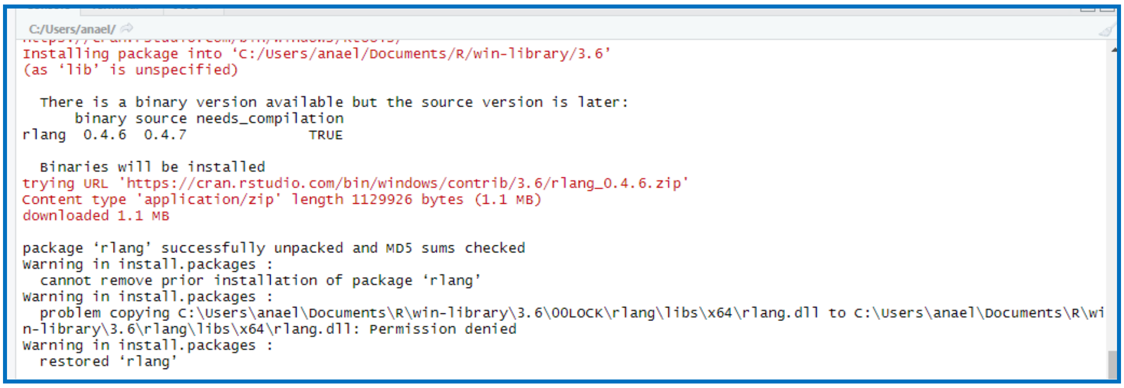


Figure 119: No Administrator rights

RStudio will need to restart in order to complete the installation Fig. 120.

```
> install.packages("rlang")
WARNING: Rtools is required to build R packages but is not currently installed. Please download and install the appropriate v
ersion of Rtools before proceeding:

https://cran.rstudio.com/bin/windows/Rtools/
Installing package into ‘C:/Users/anael/Documents/R/win-library/3.6’
(as ‘lib’ is unspecified)

  There is a binary version available but the source version is later:
      binary source needs_compilation
rlang  0.4.6  0.4.7              TRUE

  Binaries will be installed
trying URL 'https://cran.rstudio.com/bin/windows/contrib/3.6/rlang_0.4.6.zip'
Content type 'application/zip' length 1129926 bytes (1.1 MB)
downloaded 1.1 MB

package ‘rlang’ successfully unpacked and MD5 sums checked

The downloaded binary packages are in
        C:\Users\anael\AppData\Local\Temp\RtmpchTpkm\downloaded_packages
> library(syuzhet)
```

Figure 120: Update completed

After this, the emotions analysis can start.

5.4.2 Tweets Emotions – Public Opinions

This is the application of NLP to extract subjective information from text that relates to feelings. Usually pieces of text are analysed to identify the positivity or negativity of claims made about a particular topic or at a specific moment. It is useful in the analysis of tweets which are short messages, user-generated at a high velocity on Twitter platform.

We call the necessary libraries and the function get_nrc_sentiment((covid_text)), create the scores and names to assign to rows. At the end, we use ggplot2 library to design the graph representing the eight emotions and the scores in a histogram (Fig. 121 and Fig. 122).

```
Exploratory Data Analysis_Tweets.R*   Exploratory Data Analysis_News.R   Tweets_Exploration after cleaning (wor...
Source on Save                                                                 Run   Source
# Emotions Analysis

#install.packages('syuzhet')
#install.packages("syuzhet", repos="http://cran.rstudio.com/", dependencies=TRUE)
#install.packages('rlang')
library(syuzhet)

# get_nrc : Calls the NRC sentiment dictionary to calculate the
# presence of eight different emotions and their corresponding prevalence in a text file
emotion_covid <- get_nrc_sentiment((covid_text))
# Show nrc_sentiment for 10 first tweets (takes time to run)
#head(sentiment_covid,10)

# Calculating the total score for each sentiment
emotionscores_covid <- data.frame(colSums(emotion_covid[,]))
# Show cumulated scores per emotion
emotionscores_covid

names(emotionscores_covid) <- "Score"
emotionscores_covid <- cbind("emotion" = rownames(emotionscores_covid), emotionscores_covid)
rownames(emotionscores_covid) <- NULL

# Visualise emotions with scores (counts)
library(ggplot2) # package NLP
ggplot(data = emotionscores_covid, aes(x=emotion, y=Score))+geom_bar(aes(fill=emotion), stat = "identity")+ theme(le
```

Figure 121: Emotions analysis in tweets

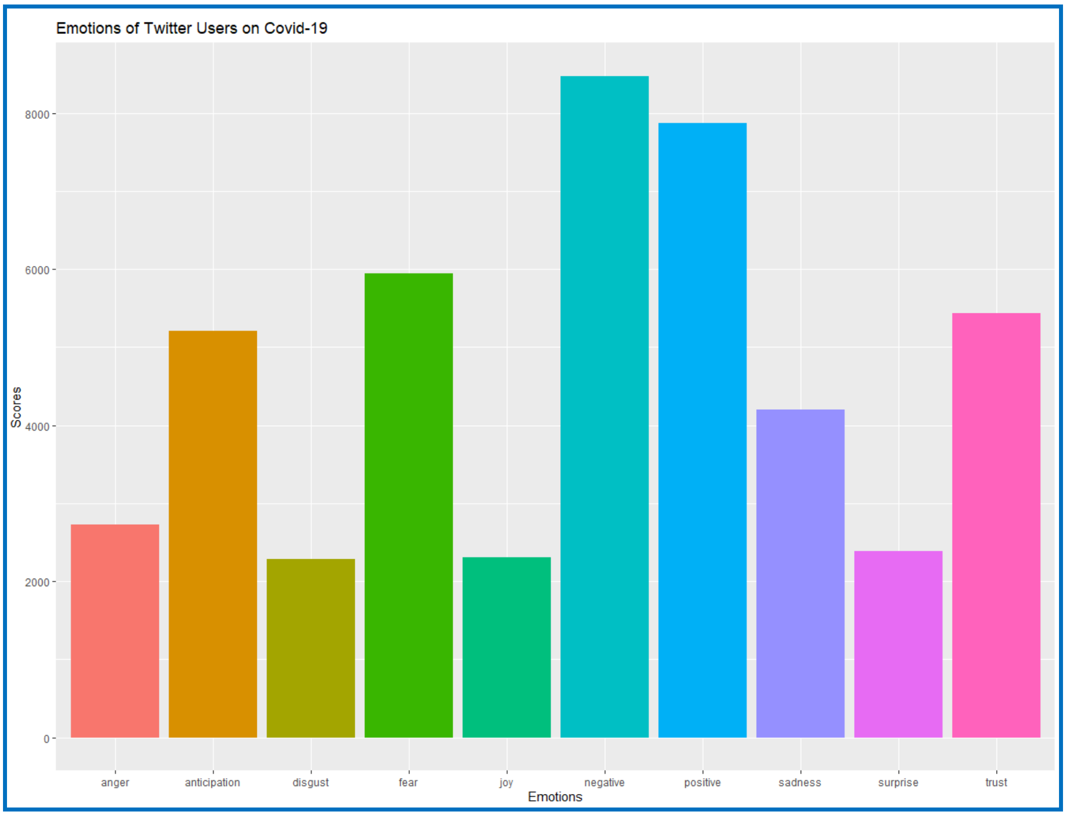


Figure 122: Tweets emotions scores

Two sentiments (positive and negative) are depicted beside 8 emotions. These two features are removed from the data fame. Scores (representing sum of word occurrences in each document) are enriched by percentage to show the valence of each emotion in the corpus of tweets. The data frame is displayed in the console and the graph in RStudio (Fig.123).

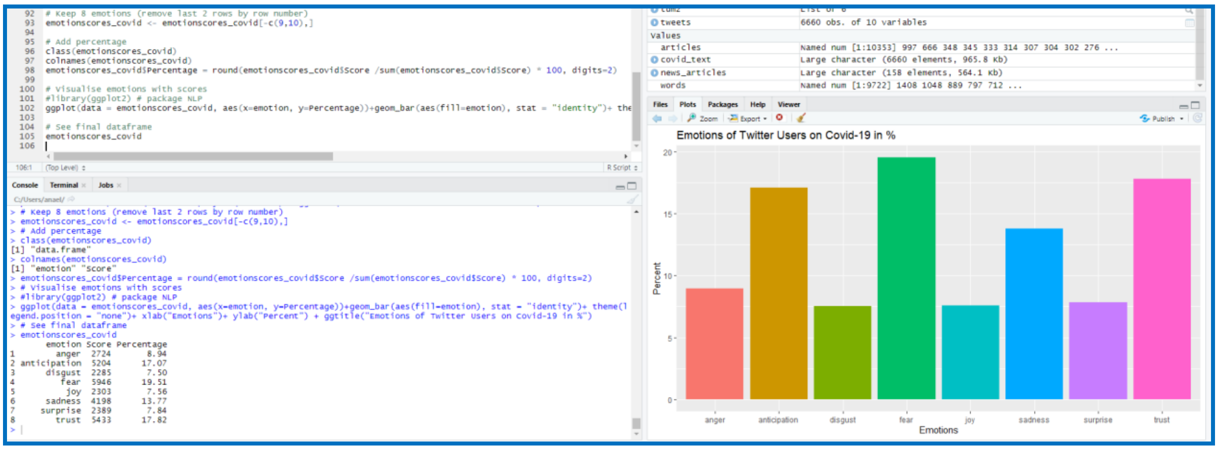


Figure 123: Emotions in tweets in % valence

5.4.3 News Articles Emotions – Official Authorities Opinions

The emotions analysis is carried out on news articles, this is to obtain the exact same output to allow for comparison of emotions evoked in both datasets.

The analysis requires calling the package syuzhet to calculate the scores for each emotion. They are plotted with a histogram them using ggplot2 (Fig. 124)

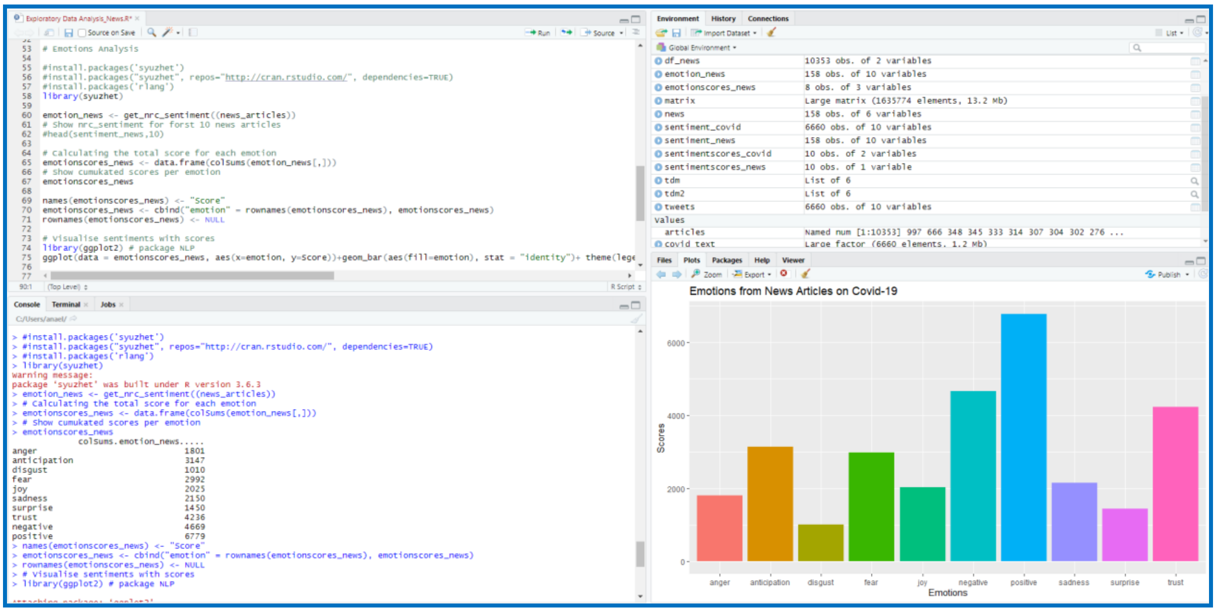


Figure 124: Code and histogram for emotions analysis from news

The two sentiments (positive and negative) are excluded, percentage of valence for each emotion is added to the data frame. Figures are displayed in the console in Fig. 125 and represented on a graph in Fig. 126.

```
> # see final dataframe
> emotionscores_news
       emotion Score Percentage
1        anger  1801       9.57
2 anticipation  3147      16.73
3      disgust  1010       5.37
4         fear  2992      15.91
5          joy  2025      10.76
6      sadness  2150      11.43
7     surprise  1450       7.71
8        trust  4236      22.52
>
```

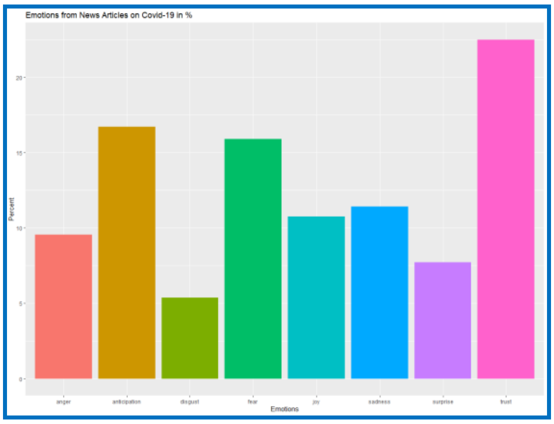


Figure 125: Emotions scores and % valence **Figure 126: Emotions in news in % valence**

Visually, we can compare the graphs obtained from tweets with this one obtained from the news. At first, we can notice the positivity feeling is higher in news, and the negativity sentiment score is lower in news compared to tweets written online by the public.

Characteristics of data being compared is different and was reflected in emotion scores as absolute values. News articles scored high on each emotion (scale up to 8,500) whereas tweets scale went up to 6,000. The reason being related to the size of each dataset (6,660 tweets and 158 news articles).

Fig.127 compares emotion weights as a percentage of the total emotions extracted from both tweets and articles. Tweets tend to convey negative emotions with higher scores in anger, disgust, fear and sadness. Taking into account the number of documents in each dataset and the length of text (up to 280 characters for tweets vs. unlimited for news articles). We can assume that with larger word counts, articles convey more information content and semantic is better captured. Therefore, it is important to compare the valence of each emotion as a percentage of the corpus. The top 4 emotions in each corpus are the same but in a different descending order. Negative emotions represent 49.72% of tweets and 42.28% in news. Positive emotions are more present in news with 33.28% against 25.38% in tweets. The two neural emotions of anticipation and surprise represent the same share in both datasets (respectively 24.44% for tweets and 24.91% for news).

Emotion as %	Tweets	News
Anger	8.94	9.57
Anticpication	17.07	16.73
Disgust	7.5	5.37
Fear	19.51	15.91
Joy	7.56	10.76
Sadness	13.77	11.43
Surprise	7.84	7.71
Trust	17.82	22.52
Total	100.0	100.0

Emotion Class	Tweets	News
Negative	49.72	42.28
Neutral	24.91	24.44
Positive	25.38	33.28
Total	100.0	100.0

Figure 127: Comparison table for emotion valence in %

As a conclusion of emotions recognition analysis, positivity towards Covid-19 is more significantly evoked in news, whereas approximately half of tweets evoke negative emotions on the pandemic.

6 Implementation

JyputerLab was used to write Python scripts and implement LDA topic modelling and sentiment analysis.

6.1 Latent Dirichlet Allocation Topic Modelling

This implementation consists in performing a topic modelling with LDA algorithm to identify the sub-topics of tweets and news articles (i.e. one article is a combination of topics, and a topic is characterised by a set of words). Topics were extracted from tweets and news in Python, the library pyLDAvis for interactive topic model visualization was used. Parts of the tutorial from Selva Prabhakaran[19] were followed and pieces of code used for the implementation.

Key factors to obtain a good topic models are:

- Text pre-processing quality,
- The variety of topics covered in documents,
- The topic modelling algorithm selection,
- K number of latent topics to extract,
- The algorithm tuning parameters (detailed in Table 4.page 60).

[19] https://www.machinelearningplus.com/nlp/topic-modeling-visualization-how-to-present-results-lda-models/#5.-Build-the-Topic-Model

1. Download pyLDAvis via Anaconda Prompt, this library is dedicated to interactive topic model visualization.

```
(base) C:\Users\anael>pip install pyLDAvis
```

Figure 128: Download pyLDAvis

2. Import libraries as shown in Fig.129.

```python
# Import libraries

import re
import numpy as np
import pandas as pd
from pprint import pprint

# Gensim
import gensim
import gensim.corpora as corpora
from gensim.utils import simple_preprocess
from gensim.models import CoherenceModel

# Plotting tools
import pyLDAvis # Python library for interactive topic model visualization
import pyLDAvis.gensim  # don't skip this
import matplotlib.pyplot as plt
%matplotlib inline

# For wordclouds
from wordcloud import WordCloud, STOPWORDS
import matplotlib.colors as mcolors
```

Figure 129: Import libraries for LDA

3. Create the variables. Fig. 130 shows the example for tweets, the same process is followed for news articles and a dedicated Python script is available in deliverables.

```python
[3]: # Read dataset
     df = pd.read_csv('export_Tweets_df2.csv')
     df.head()

     # Prepare data input for modeling
     lem = df['lemmatized_text'].values.tolist()

     lemmatized_list = []
     for lemmatized_item in lem:
         lemmatized_list.append(eval(lemmatized_item)) # Use eval () to pass string as a Python expression and returns the result

     # print(lemmatized_list[:2])

[4]: # 1. Create the Dictionary (id2word) and Corpus needed for Topic Modeling

     # Create Dictionary
     id2word = corpora.Dictionary(lemmatized_list)

     # Create Corpus
     texts = lemmatized_list

     # Term Document Frequency
     corpus = [id2word.doc2bow(text) for text in texts]

     # View
     print(corpus[:1])

     # Gensim creates a unique id for each word in the document
     # The corpus is a mapping of (word_id, word_frequency)

     [[(0, 1), (1, 1), (2, 1), (3, 1), (4, 1), (5, 1), (6, 1), (7, 1)]]

[5]: # Pass the id as a key to the dictionary to see what word a given id corresponds to
     id2word[0]

[5]: 'asterix'
```

Figure 130: Prepare variables (tweets example)

4. Build the model (Fig. 131). Topic number k = 10 is selected for tweets, and k=5 for news articles. Parameters selected are set are explained in Table 4.

```
[7]: # 2. Building the Topic Model
     # In addition to the corpus and dictionary, you need to provide the number of topics
     # alpha and eta are hyperparameters that affect sparsity of the topics. According to the Gensim docs, both defaults to 1.0/num_topics prior.
     # chunksize is the number of documents to be used in each training chunk.
     # update_every determines how often the model parameters should be updated and passes is the total number of training passes.

     # Build LDA model
     lda_model = gensim.models.ldamodel.LdaModel(corpus=corpus,
                                                id2word=id2word,
                                                num_topics=10,
                                                random_state=100,
                                                update_every=1,
                                                chunksize=100,
                                                passes=10,
                                                alpha='auto',
                                                per_word_topics=True)
```

Figure 131: Build the LDA model

Table 4: LDA topic model parameters

Parameter	Functionality
num_topics=k	number topics to be extracted from the training corpus
random_state=100	generates a seed for reproducibility
update_every=1	number of documents to be iterated through for each update. Set 1 for online iterative learning
chunksize=100	number of documents to be used in each training chunk
passes=10	# number of passes through the corpus during training
alpha='auto'	learns an asymmetric prior from the corpus
per_word_topics=True	computes a list of topics sorted in descending order of most likely topics for each word, along with their phi values multiplied by the feature length (i.e. word count)

Respective k numbers have been selected after several modeling attempts and retained when they achieved the highest coherence score and lowest perplexity score (Fig. 133 and Fig.137 for details of scores obtained for each dataset).

6.1.1 Topics Extraction from Tweets

1. Extraction of 10 topics from tweets in Fig. 130. The list of keywords and their corresponding weights to the component extracted are shown in Fig. 132. Lists of keywords from tweets and news, with topic labels are available in Table 5, in the section 6.1.3. Comparison.

```
[8]: # 3. View the topics in LDA model
     # The LDA model is built with 10 different topics where each topic is a combination of keywords
     # and each keyword contributes a certain weightage to the topic.You can see this using lda_model.print_topics()

     # Print the top 10 keywords in the 10 topics
     pprint(lda_model.print_topics())
     doc_lda = lda_model[corpus]

     [(0,
       '0.033*"india" + 0.032*"global" + 0.026*"come" + 0.024*"infection" + '
       '0.021*"high" + 0.019*"look" + 0.017*"corona" + 0.017*"information" + '
       '0.015*"fear" + 0.015*"die"'),
      (1,
       '0.038*"follow" + 0.024*"vaccine" + 0.024*"ship" + 0.024*"supply" + '
       '0.018*"crisis" + 0.018*"cruise" + 0.017*"cancel" + 0.016*"epidemic" + '
       '0.014*"cov" + 0.013*"likely"'),
      (2,
       '0.105*"case" + 0.042*"confirm" + 0.041*"death" + 0.038*"country" + '
       '0.025*"dont" + 0.024*"utm" + 0.024*"agency" + 0.021*"week" + 0.020*"medium" '
       '+ 0.019*"source"'),
```

Figure 132: Extraction of 10 topics from tweets

2. Calculation of coherence and perplexity scores (Fig. 133) with “c_v” measure. According to Shashank Kapadia[20] is it a measure “based on a sliding window, one-set segmentation of the top words and an indirect confirmation measure that uses normalized pointwise mutual information (NPMI) and the cosine similarity” (Kapadia, 2019).

```
[10]: # 4. Compute Model Perplexity and Coherence Score
      # Measure to judge how good a given topic model is

      # Compute Perplexity
      print('\nPerplexity: ', lda_model.log_perplexity(corpus))  # a measure of how good the model is. lower the better.

      # Compute Coherence Score
      coherence_model_lda = CoherenceModel(model=lda_model, texts=lemmatized_list, dictionary=id2word, coherence='c_v')
      coherence_lda = coherence_model_lda.get_coherence()
      print('\nCoherence Score: ', coherence_lda)

      # topic coherence score of 0.329 for Tweets k=20 topics ; 0.363 with k=15 ; 0.381 with k=10 ; 0.329 with k=5( vs. 0.475 for News topics with k=5)
```

Figure 133: Perplexity and coherence scores for tweets

3. Topics extracted are visualised via an interactive graph in JupyterLab built using the library pyLDAvis (Fig. 134).The interactive graphs shows bubbles representing news topics are spread across the chart while 7 tweets topics overlap and 3 are clearly apart and distinct from the others (Fig.135).

```
[11]: # 5. Visualize the topics-keywords
      pyLDAvis.enable_notebook()
      vis = pyLDAvis.gensim.prepare(lda_model, corpus, id2word)
      vis
```

Figure 134: Build interactive graph

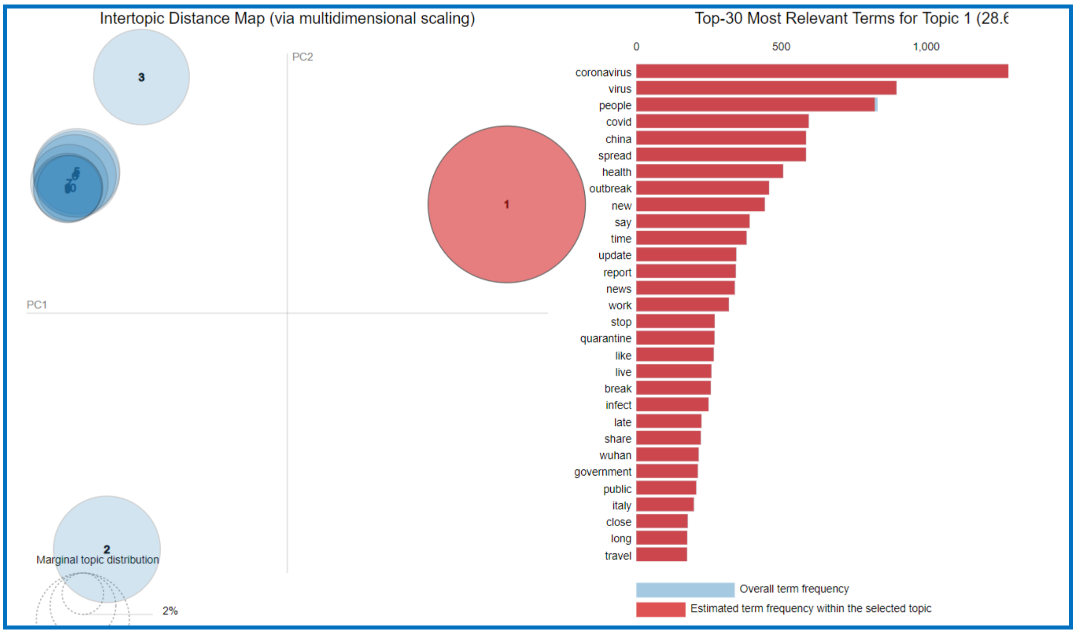


Figure 135: Interactive graphs in JupyterLab

[20] https://towardsdatascience.com/evaluate-topic-model-in-python-latent-dirichlet-allocation-lda-7d57484bb5d0

To read this type of graph, we look at the size of the bubbles that translates how dominant a topic is across all documents from the corpus. Close bubbles mean they have similar topic (on the opposite, the more apart they are, the less similar topics are). Preferably, a good model would show non-overlapping bubbles and as much spread as possible across the graph.
Keywords on the right-hand side drive that topic selected when we hover over with the mouse in JupyterLab.

6.1.2 Topic Extraction from News Articles

1. Topic model implemented on news articles is built to extract k=5 topics (Fig. 136).

```
[8]: # 3. View the topics in LDA model
     # The LDA model is built with 5 different topics where each topic is a combination of keywords
     # and each keyword contributes a certain weightage to the topic.You can see this using lda_model.print_topics()

     # Print the top 10 keywords in the 5 topics
     pprint(lda_model.print_topics())
     doc_lda = lda_model[corpus]

     [(0,
       '0.014*"case" + 0.014*"say" + 0.009*"test" + 0.009*"health" + 0.006*"u" + '
       '0.006*"outbreak" + 0.006*"people" + 0.006*"public" + 0.005*"country" + '
       '0.005*"new"'),
      (1,
       '0.009*"plastic" + 0.007*"bag" + 0.005*"crisis" + 0.004*"industry" + '
       '0.004*"pandemic" + 0.004*"ban" + 0.003*"say" + 0.003*"year" + 0.003*"group" '
       '+ 0.003*"singleuse"'),
      (2,
       '0.016*"theatre" + 0.009*"production" + 0.009*"available" + 0.008*"online" + '
       '0.006*"film" + 0.006*"company" + 0.006*"full" + 0.006*"include" + '
       '0.006*"show" + 0.006*"play"'),
      (3,
       '0.015*"say" + 0.011*"people" + 0.007*"would" + 0.006*"get" + 0.006*"go" + '
       '0.006*"one" + 0.005*"work" + 0.004*"time" + 0.004*"vaccine" + 0.004*"day"'),
      (4,
       '0.014*"say" + 0.007*"government" + 0.006*"food" + 0.006*"would" + '
       '0.005*"home" + 0.005*"people" + 0.004*"minister" + 0.004*"job" + '
       '0.004*"plan" + 0.004*"one"')]
```

Figure 136: Extraction of 5 topics from news

2. Perplexity and coherence are also computed to verify the quality of this model (Fig. 137). For coherence, the "c_v" measure is used to build the news model. Scores obtained provide a consistence basis for comparison with tweets topic modelling in Fig.133 (page 61).

```
[9]: # 4. Compute News Model Perplexity and Coherence Score
     # Measure to judge how good a given topic model is

     # Compute Perplexity
     print('\nPerplexity: ', lda_model.log_perplexity(corpus))  # a measure of how good the model is. lower the better

     # Compute Coherence Score
     coherence_model_lda = CoherenceModel(model=lda_model, texts=lemmatized_list, dictionary=id2word, coherence='c_v')
     coherence_lda = coherence_model_lda.get_coherence()
     print('\nCoherence Score: ', coherence_lda)

     # topic coherence score of 0.475 with k=5 topics ; 0.451 with k=10, 0.413 with k=0.433, 0.453 with k=20 topics


     Perplexity:  -8.016519840575398

     Coherence Score:  0.47506130501049454
```

Figure 137: Perplexity and coherence scores for news

3. The interactive graphs for news topics is shown in Fig.138.

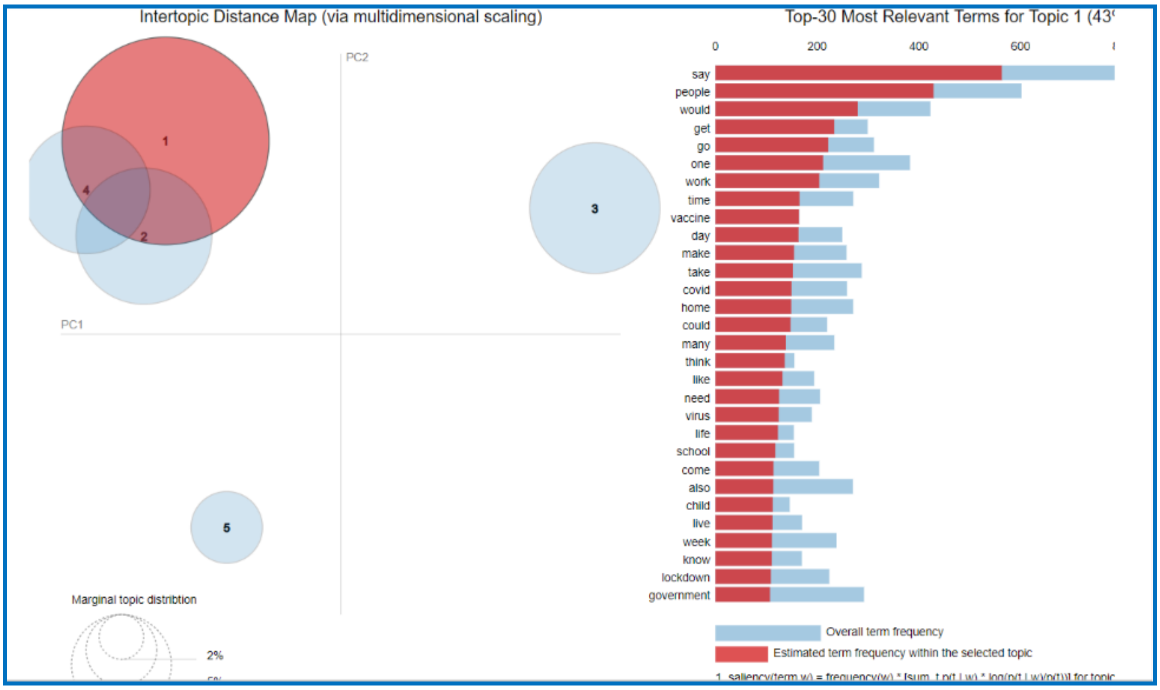


Figure 138: News Topic Modeling

6.1.3 Comparison

10 topics were extracted from tweets to obtain the highest coherence score (0.381) and 5 topics from news to obtain a coherence of 0.475. Table 5 shows the list of keywords returned for each component and the themes inferred by the panel of members to assign a topic label to each. Some words were highlighted to show **positive word** or **negative word** in the context of Covid-19, and *outlier, typo, abbreviation* that appeared in the keywords output from topic modelling.

Table 5: LDA Topic modelling output with keywords, suggestions and inferred topics

Source_topic_index	Keywords	Researcher	Panel Member1	Panel Member2	Panel Member3	Panel Member4	Inferred Topics
News_topic_0	**case**, say, test, **health**, *u*, **outbreak**, people, public, country, new	Development, Cases Update	Spread of Covid	Coronavirus	Spike in Covid-19 along border counties	COVID cases on rise	Pandemic development update, Information giving
News_topic_1	plastic, bag, **crisis**, industry, **pandemic**, **ban**, say, year, group, *singleuse*	Pollution	Environment	Environment	Pandemic results in increase of single use plastics	Waste and increase of single-use items usage	Environmental impact of Covid
News_topic_2	theatre, production, available, online, film, company, full, include, show, play	Entertainment	Concert	Film/Theatre	Review of online film and theatre	Online productions of stage play	Cultural Entertainment
News_topic_3	say, people, would, get, go, one, work, time, **vaccine**, day	Vaccine, End of lockdown	Covid vaccine	Coronavirus	Vaccine could enable people to return to work safely	Attitudes towards risk of new vaccine	Vaccine

News_topic_4	say, government, food, would, **home**, people, minister, job, **plan**, one	Lifestyle, Economy	Lockdown	Relationships	Government hopeful workers can return to workplace by end of year	Quarantine, people cook at home	Economic impact of lockdown
Tweets_topic_0	india, **global**, come, **infection**, **high**, look, corona, information, **fear**, **die**	Development, Cases Update	Statistics	Medicine	Coronavirus tracking concerns in India	Updated details on COVID	Pandemic development in India
Tweets_topic_1	follow, vaccine, ship, supply, **crisis**, cruise, cancel, **epidemic**, cov, likely	Diamond Princess	Spread of Covid	Quarantine	Cluster of cases on the cruise ship Diamond Princess	Outbreak on cruise ship	Diamond Princess infection cluster
Tweets_topic_2	**case**, confirm, **death**, country, *dont*, *utm*, agency, week, medium, source	Development, Cases Update	Report	Politics	Government update on the pandemic	COVID cases update	Pandemic development update, Information giving
Tweets_topic_3	patient, way, hospital, go, medical, **concern**, *click*, *cdc*, **care**, country	Treatment	Pandemic	Medical care	When is it ok to visit your GP?	Care for those struggling with fear	Medical care
Tweets_topic_4	coronavirus, **virus**, people, covid, china, **spread**, **health**, **outbreak**, new, say	Origin of outbreak	Pandemic	Coronavirus	Search for source of coronavirus	Outbreak in China	Search for source of coronavirus
Tweets_topic_5	prepare, **disease**, **home**, **stay**, night, **recommend**, **hit**, amid, grow, **increase**	Protection Measures	Covid	Coronavirus	People look for clear instruction during lockdown	Recommendations for staying home	Instructions to behave during the crisis

Tweets_topic_6	**test**, read, **symptom**, **kit**, positive, iran, thing, **risk**, **try**, **protect**	Screening	Covid remedy	Medicine	Roll out for Covid-19 testing	Positive outcome for most tested	Covid testing
Tweets_topic_7	day, know, world, monitor, **response**, *rt*, **prevention**, **wait**, **best**, *govt*	Protection Measures	Quarantine	Politics	Instructions to monitor the propagation and outlook for Covid restrictions lifting	Government response	Instructions to behave during the crisis
Tweets_topic_8	need, think, **pandemic**, focus, get, **good**, hand, use, trump, let	Politics	International response	Politics	Support to Trump's Covid crisis management	Government response	American response the Covid crisis
Tweets_topic_9	help, spy, today, **hope**, question, ask, *accord*, video, understand, **possible**	Politics	Media	Politics	IT fault on Guardian website	?	Outlier

6.2 Sentiment Analysis

Natural Language Toolkit (NLTK) available in Python offer three sophisticated lexicons: VADER (Valence Aware Dictionary and Sentiment Reasoner – attuned for social media text content), TextBlob and Sentiwordnet. TextBlob is a popular lexicon and can be used on any text data. It is suitable for analyzing sentiments from social media content and news media content. TextBlob computes a polarity score, which measures how positive or negative the emotion of a given text is, and a corresponding subjectivity measure referring to opinions or views (Bold, 2019).

The sentiment analysis is done using TextBlob library in Python. TextBlob handles negation (i.e. 'great' has a polarity of 0.8 and 'not great' scores -0.4, both have the same subjectivity of 0.75) and modifiers such as 'very' (i.e. 'very great' has the maximum polarity of 1.00 and subjectivity of 0.975 whereas 'not very great' scores -0.31 in polarity and 0.58 in subjectivity) (Fig. 139).

```
# Examples for report
examples = ["great", "not great", "very great", "not very great",
            "The situation is serious",
            "The situation is very serious",
            "The virus spreads rapidly",
            "The new virus spreads quickly worldwide",
            "The virus spreads quickly worldwide"]

for e in examples:
    print(TextBlob(e).sentiment, e)
```

```
Sentiment(polarity=0.8, subjectivity=0.75) great
Sentiment(polarity=-0.4, subjectivity=0.75) not great
Sentiment(polarity=1.0, subjectivity=0.9750000000000001) very great
Sentiment(polarity=-0.3076923076923077, subjectivity=0.5769230769230769) not very great
Sentiment(polarity=-0.3333333333333333, subjectivity=0.6666666666666666) The situation is serious
Sentiment(polarity=-0.43333333333333335, subjectivity=0.8666666666666667) The situation is very serious
Sentiment(polarity=0.0, subjectivity=0.0) The virus spreads rapidly
Sentiment(polarity=0.23484848484848483, subjectivity=0.4772727272727273) The new virus spreads quickly worldwide
Sentiment(polarity=0.3333333333333333, subjectivity=0.5) The virus spreads quickly worldwide
```

Figure 139: Examples with TextBlob

Required libraries are imported as shown in Fig.140.

```
[1]: # Sentiment Analysis with TextBlob

import os
import pandas as pd
from textblob import TextBlob
import numpy as np

# For confusion matrix and metrics:
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report

# For statistics
import statistics
import math
from math import sqrt
from numpy import mean
from scipy.stats import t

#For Visualisation
from matplotlib import pyplot as plt
import seaborn as sns

# For T-test using spicy
from numpy.random import seed
from numpy.random import randn
from scipy.stats import ttest_rel # For T-test on dependent samples
from scipy.stats import ttest_ind # For T-test on independent samples
```

Figure 140: Import libraries

6.2.1 Sentiment Analysis on Tweets

1. Polarity scores calculated for tweets are named "sentiment2" (Fig. 141) to distinguish from the feature "sentiment1" that refers to scores computed on raw tweet text (before the pre-processing task).

```
[2]: # Read tweets dataset
os.chdir('C:/Users/anael')
df2 = pd.read_csv('export_Tweets_df2.csv')

# Sentiment analysis with polarity and subjectivity scores. Polarity is named 'sentiment2'
df2[['sentiment2', 'subjectivity']] = df2['lemmatized_text'].apply(lambda lemmatized_text: pd.Series(TextBlob(lemmatized_text).sentiment))
```

Figure 141: Sentiment analysis on tweets

2. Sentiment2 scores are binned into 3 classes (positive, neutral, negative) to make a confusion matrix and analyse the distribution of tweets across these 3 sentiment categories (Fig.142).

```
[18]: # Change sentiment & polarity scores (float) to integers to make 3 categories :
# score > 0.3 = 1 (positive sentiment)
# score < -0.3 = -1 (negative)
# else : score > -0.3 and > 0.3 = 0 (neutral)

sentiment2_class = []

for index, score in df2.iterrows():
    score = score['sentiment2']
    if score > 0.3:
        score_class = 1 #"positive"
    elif score < -0.3:
        score_class = -1 # "negative"
    else:
        score_class = 0 # "neutral"
    sentiment2_class.append(score_class)

    #print (polarity_class)
df2['sentiment2_class'] = sentiment2_class # create new col. in df with output

sentiment1_class = []

for index, score in df2.iterrows():
    score = score['sentiment1']
    if score > 0.3:
        score_class = 1 # "positive"
    elif score < -0.3:
        score_class = -1 # "negative"
    else:
        score_class = 0 # "neutral"
    sentiment1_class.append(score_class)
    #print (sentiment_class)
df2['sentiment1_class'] = sentiment1_class # print values and add te col. in df
```

Figure 142: Categorize sentiment scores

3. The accuracy score obtained is 86.68%. A confusion matrix is drawn to compare classification of sentiment1 and sentiment2 in order to see the classification accuracy of tweets sentiment before and after text pre-processing (Fig.143). The matrix obtained is redesigned using Excel (Fig.144) for readability of a 3 by 3 confusion matrix and comparison with a classification model using cut-off at 0.2 and -0.2 for sentiment classes.

```
# Actual classes corresponds to 'sentiment1' (before text pre-processing) and Predicted classes corresponds to 'sentiment2' (after)
print ('Confusion Matrix :')
print(confusion_matrix(sentiment1_class, sentiment2_class))
print('Accuracy Score :', accuracy_score(sentiment1_class, sentiment2_class))
print ('Classification Report : ')
print(classification_report(sentiment1_class, sentiment2_class))

Confusion Matrix :
[[ 133   38    3]
 [ 237 5354  415]
 [   1  193  286]]
Accuracy Score : 0.8668168168168168
Classification Report :
              precision    recall  f1-score   support

          -1       0.36      0.76      0.49       174
           0       0.96      0.89      0.92      6006
           1       0.41      0.60      0.48       480

    accuracy                           0.87      6660
   macro avg       0.57      0.75      0.63      6660
weighted avg       0.90      0.87      0.88      6660
```

Figure 143: Classification performance metrics

4. To interpret the 3 by 3 confusion matrix, I recommend the article from Neo Yi Peng published on Towards Data Science website[21] that is very well explained and complemented with visuals (Fig.140) to locate true positive (TP), true negative (TN), false positive (FP) and false negative (FN) in the matrix.

Model 1 : 0.3 cut-off

Accuracy 86.68%

		Actual Sentiment Class				
		Negative (-1)	Neutral (0)	Positive (1)	Count =	
Predicted	Negative (-1)	133 A	38 B	3 C	174	2.61%
	Neutral (0)	237 D	5354 E	415 F	6006	90.18%
	Positive (1)	1 G	193 H	286 I	480	7.21%
	Count =	371	5585	704		
		5.57%	83.86%	10.57%		

Model 2 : 0.2 cut-off

Accuracy 80.12%

		Actual Sentiment Class				
		Negative (-1)	Neutral (0)	Positive (1)	Count =	
Predicted	Negative (-1)	242 A	69 B	6 C	317	4.76%
	Neutral (0)	340 D	4516 E	512 F	5368	80.60%
	Positive (1)	31 G	366 H	578 I	975	14.64%
	Count =	613	4951	1096		
		9.20%	74.34%	16.46%		

Figure 144: Confusion matrices

5. The distribution of tweets per class (Fig.145) shows sentiment2 feature contains 90.18% of instances categorised in the neutral class based on the polarity score computed on pre-processed text and classes defined with a cut off at 0.3 and -0.3. whereas sentiment1 feature shows a slightly more balanced distribution over the three classes, with again a dominance of the neutral class (83.86% of tweets). In sentiment2 feature, 7.21% of tweets are positive and 2.61% are negative.

[21] https://towardsdatascience.com/simplifying-precision-recall-and-other-evaluation-metrics-d066b527c6bb

```
[19]: # Descriptive statisctics per sentiment2_class
      #print(df2.groupby(by=['sentiment2_class']).describe())

      # Count of occurences of each of the unique values in the columns stated
      print('Counts in Sentiment2 Class (computed after pre-processing text): ')
      print(df2['sentiment2_class'].value_counts())
      print('Counts in Sentiment1 Class (computed on raw text): ')
      print(df2['sentiment1_class'].value_counts())

      Counts in Sentiment2 Class (computed after pre-processing text):
       0    5585
       1     704
      -1     371
      Name: sentiment2_class, dtype: int64
      Counts in Sentiment1 Class (computed on raw text):
       0    6006
       1     480
      -1     174
      Name: sentiment1_class, dtype: int64
```

Figure 145: Distribution per class for sentiment1 and sentiment2

6. Tweets frequency is clearer to visualise on normalised histogram graphs in Fig. 146 and Fig.147.

```
[12]: # Visualise distribution of sentiment1 scores

      x1 = df2.loc[df2.sentiment1_class==1, 'sentiment1']
      x2 = df2.loc[df2.sentiment1_class==0, 'sentiment1']
      x3 = df2.loc[df2.sentiment1_class==-1, 'sentiment1']

      # Not normalized
      #kwargs = dict(alpha=0.5, bins=5)
      # Normalize
      kwargs = dict(alpha=0.5, bins=5, density=True, stacked=True)

      plt.hist(x1, **kwargs, color='g', label='Positive')
      plt.hist(x2, **kwargs, color='y', label='Neutral')
      plt.hist(x3, **kwargs, color='r', label='Negative')
      plt.gca().set(title='Tweets Frequency Histogram of Sentiment1 Score', ylabel='Tweets Frequency')
      plt.xlim(-1,1)
      plt.legend();
```

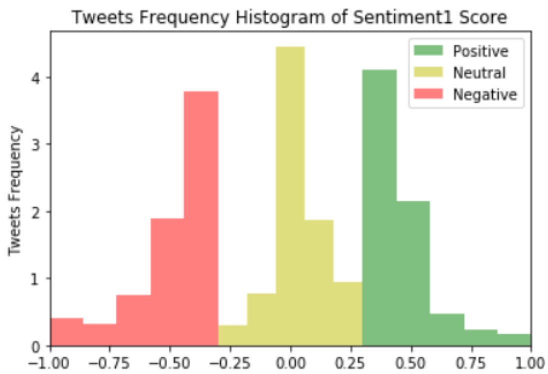


Figure 146: Normalized frequency sentiment1

```
[11]: # Visualise distribution of sentiment2 scores

x1 = df2.loc[df2.sentiment2_class==1, 'sentiment2']
x2 = df2.loc[df2.sentiment2_class==0, 'sentiment2']
x3 = df2.loc[df2.sentiment2_class==-1, 'sentiment2']

# Not normalized
#kwargs = dict(alpha=0.7, bins=5)
# Normalize
kwargs = dict(alpha=0.7, bins=5, density=True, stacked=True)

plt.hist(x1, **kwargs, color='g', label='Positive')
plt.hist(x2, **kwargs, color='b', label='Neutral')
plt.hist(x3, **kwargs, color='r', label='Negative')
plt.gca().set(title='Tweets Frequency Histogram of Sentiment2 Score', ylabel='Tweets Frequency')
plt.xlim(-1,1)
plt.legend();
```

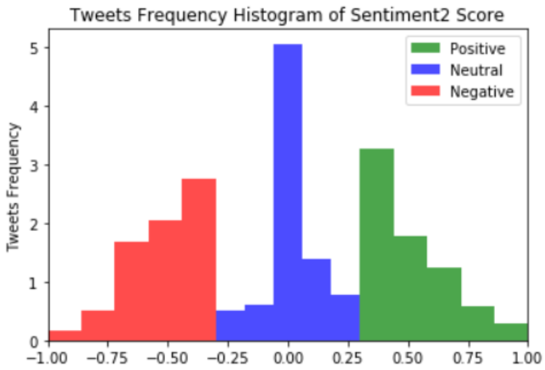


Figure 147: Normalized frequency sentiment2

7. Tweets distribution per class can also be visualized with categorical dot plots as seen on Fig. 148 and Fig. 149. These plots depict several axes-level functions that show the relationship between a numerical and one or more categorical variables. Here the categorical variable is the sentiment class (positive, negative, neutral) and numeric variable is the polarity score computed.

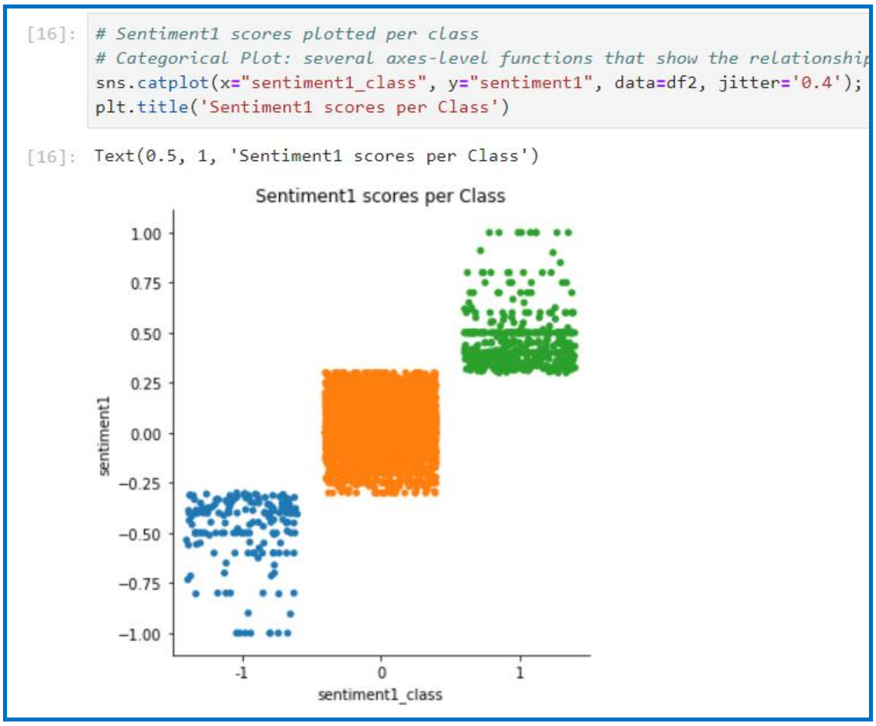


Figure 148: Sentiment1 scores per Class

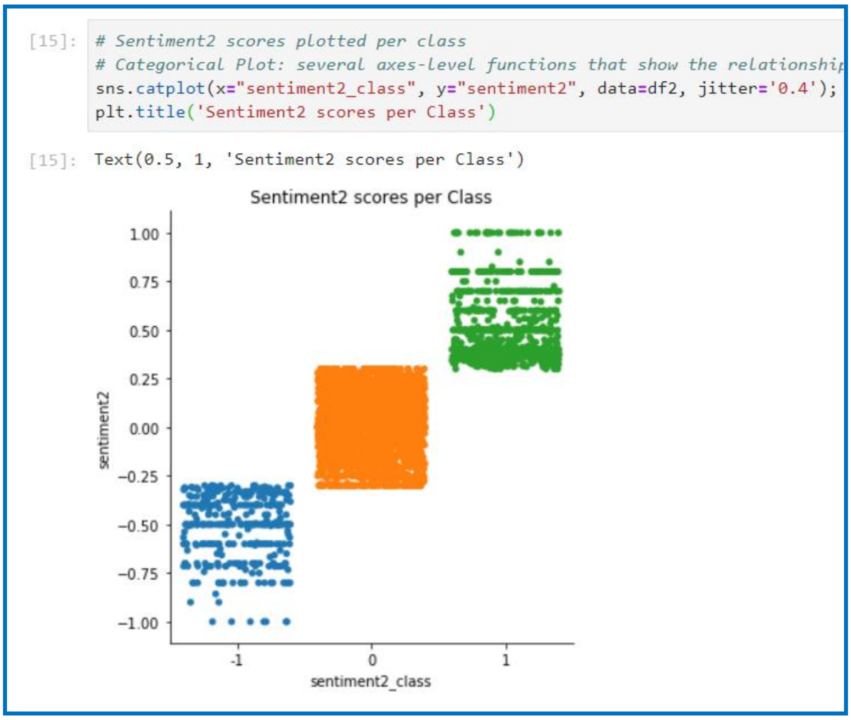


Figure 149: Sentiment2 scores per Class

8. After visual comparison of tweets sentiment scores and corresponding classes, before and after pre-processing text data, the paired t-test is computed (Fig. 150) to test the impact of pre-processing by stating the following hypotheses (please refer to the technical report section 5.2.2).

```
[7]: # Paired Student's t-test with spicy library

'''Compare tweets sentiment score before and after cleaning
Where we collect some observations on a sample from the population, then apply some treatment,
and then collect observations from the same sample'''

# Code sourced from Brownlee, 2018
# https://machinelearningmastery.com/how-to-code-the-students-t-test-from-scratch-in-python/

# generate two independent samples
data1 = df2['sentiment1'].values.tolist()
data2 = df2['sentiment2'].values.tolist()
# compare samples
stat, p = ttest_rel(data1, data2)
print('Statistics=%.3f, p=%.3f' % (stat, p))

Statistics=4.431, p=0.000
```

Figure 150: Paired t-test

6.2.2 Sentiment Analysis on News

Sentiment scores are computed on news articles. From the descriptive statistics we see they range from -0.121956 to 0.219396 (Fig.151)., meaning they evoke neutral sentiment based on the bins we designed for both tweets and news scores as shown in Fig.152.

```
[11]: # Sentiment Analysis of News articles

os.chdir('C:/Users/anael')

# Read file
df_news = pd.read_csv('export_NewsArticles.csv')

# Sentiment analysis with polarity and subjectivity scores. Polarity score on news is named 'sentiment3'
df_news[['sentiment3', 'subjectivity']] = df_news['lemmatized_text'].apply(lambda lemmatized_text: pd.Series(TextBlob(lemmatized_text).sentiment))

[12]: df_news.describe(include=['number'])
```

[12]:

	sentiment3	subjectivity
count	158.000000	158.000000
mean	0.059065	0.432808
std	0.062496	0.069621
min	-0.121956	0.218750
25%	0.018773	0.398425
50%	0.062673	0.441871
75%	0.104403	0.478800
max	0.219396	0.659259

Figure 151: Sentiment analysis on news

```python
[13]: # Change polarity scores (float) from news to integers to make 3 categories :
      # score > 0.3 = 1 (positive sentiment)
      # score < -0.3 = -1 (negative)
      # else : score > -0.3 and > 0.3 = 0 (neutral)

      sentiment3_class = []

      for index, score in df_news.iterrows():
          score = score['sentiment3']
          if score > 0.3:
              score_class = 1 #"positive"
          elif score < -0.3:
              score_class = -1 # "negative"
          else:
              score_class = 0 # "neutral"
          sentiment3_class.append(score_class)

          #print (polarity_class)
      df_news['sentiment3_class'] = sentiment3_class
```

Figure 152: Categorize sentiment scores

The distribution of news articles per sentiment class (Fig.153) shows sentiment3 feature falls at 100% into the neutral class.

```python
[14]: # Visualise distribution of sentiment scores from news

      x1 = df_news.loc[df_news.sentiment3_class==1, 'sentiment3']
      x2 = df_news.loc[df_news.sentiment3_class==0, 'sentiment3']
      x3 = df_news.loc[df_news.sentiment3_class==-1, 'sentiment3']

      # Not normalized
      kwargs = dict(alpha=0.9, bins=10)
      # Normalize
      #kwargs = dict(alpha=0.5, bins=5, density=True, stacked=True)

      plt.hist(x1, **kwargs, color='g', label='Positive')
      plt.hist(x2, **kwargs, color='y', label='Neutral')
      plt.hist(x3, **kwargs, color='r', label='Negative')
      plt.gca().set(title='News Frequency Histogram of Sentiment Score', ylabel='News Frequency')
      plt.xlim(-1,1)
      plt.legend();
```

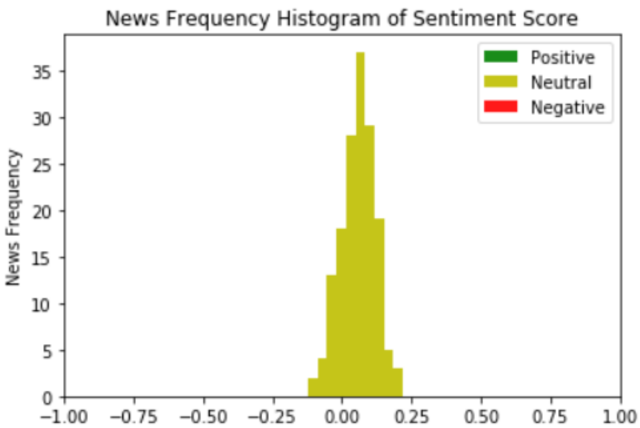


Figure 153: News frequency

Polarity scores computed on news articles ranged from -0.3 to 0.3 (Fig. 154).

```
[15]: # Sentiment3 scores plotted per class
      # Categorical Plot: several axes-level functions that show the relationship between a numerical and one or more categorical variables
      sns.catplot(x="sentiment3_class", y="sentiment3", data=df_news, jitter='0.4');
      plt.title('News Sentiment3 scores per Class')

      # All news are assigned to the neutral class

[15]: Text(0.5, 1, 'News Sentiment3 scores per Class')
```

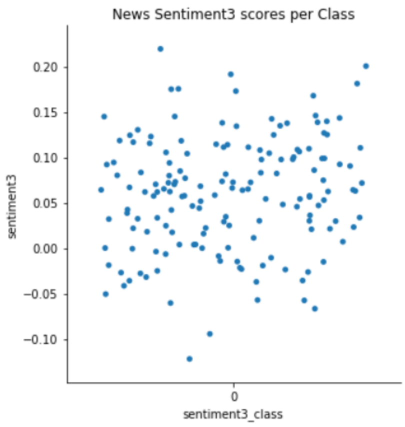


Figure 154: Distribution per class

The Welch's non-parametric t-test on independent samples is used to test the average polarity from tweets to average sentiment from news (Fig.155) (please refer to the technical report section 5.2.3)

```
[33]: # Welch's test on two independent samples of unequal variances

      # Assign the two independent samples
      data1_news = df_news['sentiment3'].values.tolist()
      data2_tweets = df2['sentiment2'].values.tolist()

      # Verify variance of both samples
      print('Variance sentiment3: ', round(statistics.stdev(data1_news),3))
      print('Variance sentiment2: ', round(statistics.stdev(data2_tweets),3))

      # Calculate Welch's t-test
      t_score = stats.ttest_ind_from_stats(mean1=mean(data1_news), std1=np.std(data1_news), nobs1=len(data1_news), \
                                      mean2=mean(data2_tweets), std2=np.std(data2_tweets), nobs2=len(data2_tweets), \
                                      equal_var=False) # If False: Welch's t-test does not assume equal population variance
      t_score

      Variance sentiment3:  0.062
      Variance sentiment2:  0.236
[33]: Ttest_indResult(statistic=3.113235824685355, pvalue=0.0020407671759979716)
```

Figure 155: Welch's unpaired t-test

7 Extra Implementation

7.1 Troubleshooting Installation of Python Libraries

7.1.1 Spacy

1. Download spacy library using Anaconda Prompt (Fig.156).

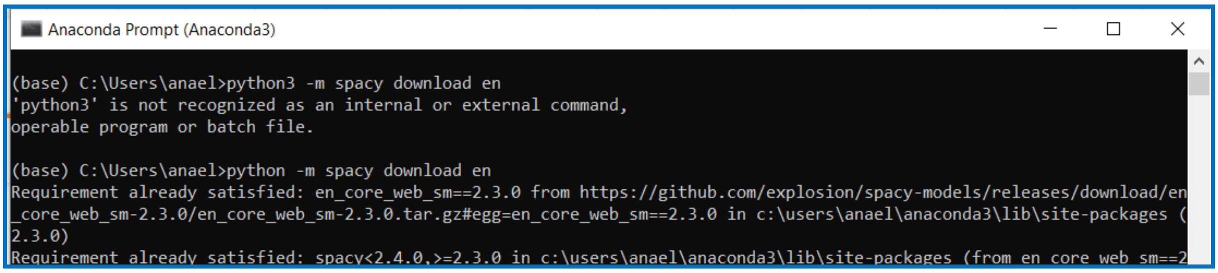


Figure 156: Download SpaCy

2. At the end of the download, an error was noted. The model “en” necessary for the topic modelling was successfully imported but not linked on the machine.

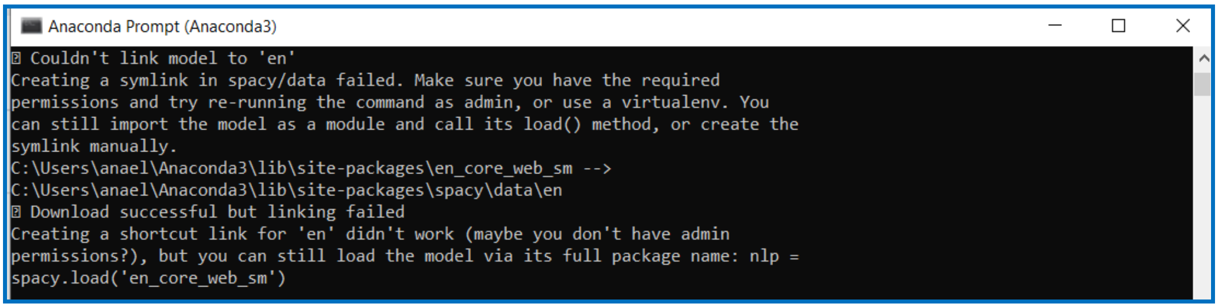


Figure 157: Error to create a link

3. To cope with this, use this syntax in Python script:

`nlp = spacy.load('en_core_web_sm'` instead of `nlp = spacy.load('en’)`

7.1.2 Keras

1. Install the library Keras

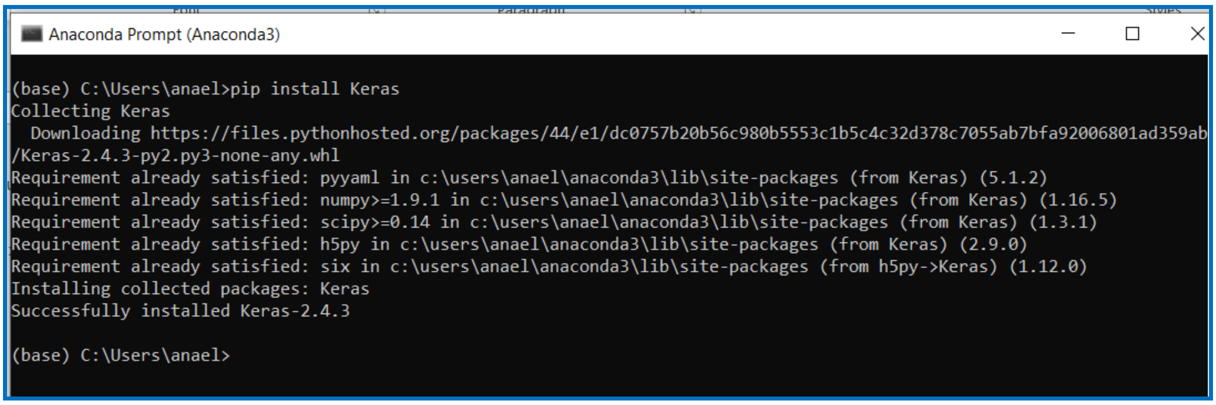


Figure 158: Download Keras libary

2. When importing the module into Python, I get an error message and the instruction to download another package.

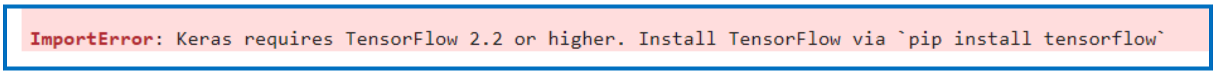


Figure 159: Error Message when importing Keras

3. Download the recommended library tensorflow as requested.

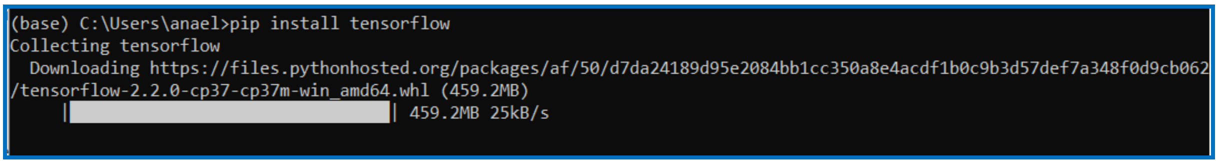


Figure 160: Download tensorflow

4. See the official website[22] for solutions. If the error occurs again, install the following from Anaconda Prompt Command: pip install tensorflow==2.2.0rc2
   This command will uninstall and re-install tensorflow 2.2 (Fig. 160 and Fig.161).

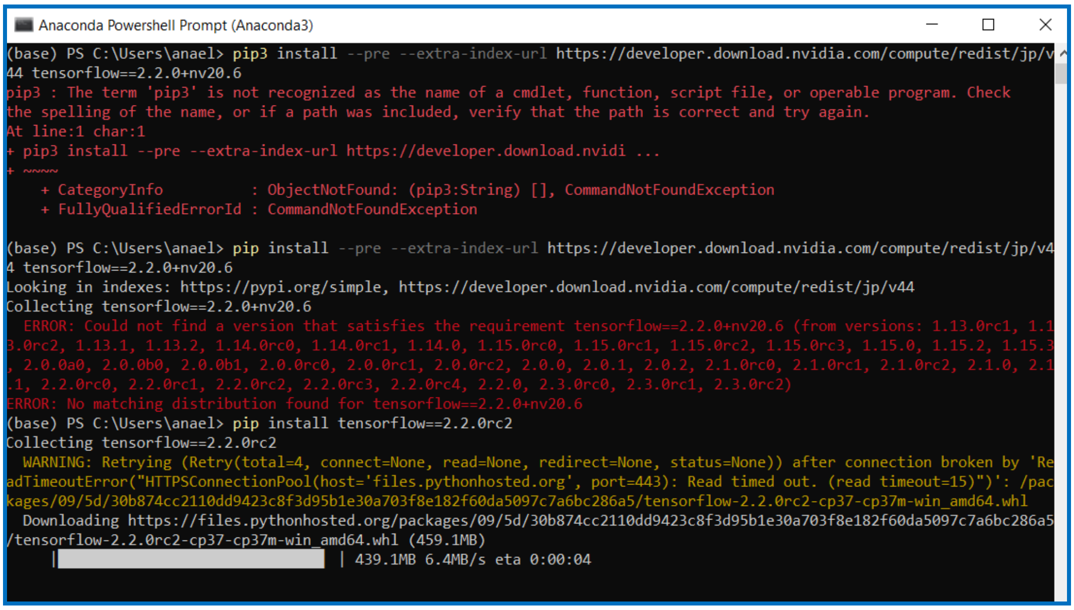
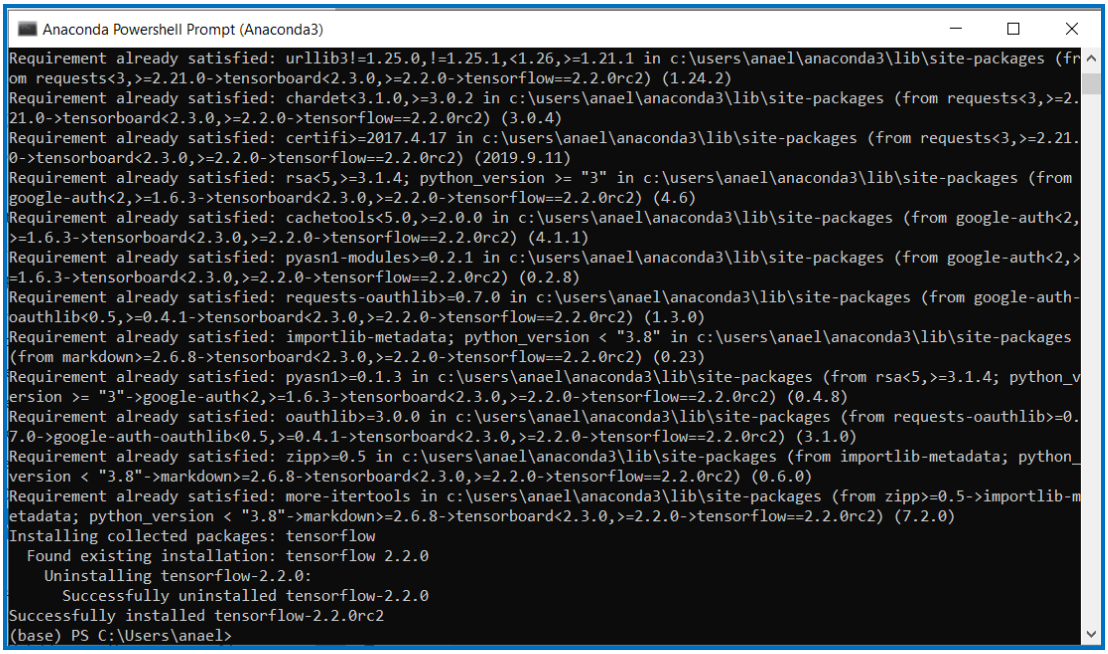


Figure 161: Re-install tensorflow (1/2)

[22] https://tensorflow.google.cn/install/pip



```
Anaconda Powershell Prompt (Anaconda3)
Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1 in c:\users\anael\anaconda3\lib\site-packages (fr
om requests<3,>=2.21.0->tensorboard<2.3.0,>=2.2.0->tensorflow==2.2.0rc2) (1.24.2)
Requirement already satisfied: chardet<3.1.0,>=3.0.2 in c:\users\anael\anaconda3\lib\site-packages (from requests<3,>=2.
21.0->tensorboard<2.3.0,>=2.2.0->tensorflow==2.2.0rc2) (3.0.4)
Requirement already satisfied: certifi>=2017.4.17 in c:\users\anael\anaconda3\lib\site-packages (from requests<3,>=2.21.
0->tensorboard<2.3.0,>=2.2.0->tensorflow==2.2.0rc2) (2019.9.11)
Requirement already satisfied: rsa<5,>=3.1.4; python_version >= "3" in c:\users\anael\anaconda3\lib\site-packages (from
google-auth<2,>=1.6.3->tensorboard<2.3.0,>=2.2.0->tensorflow==2.2.0rc2) (4.6)
Requirement already satisfied: cachetools<5.0,>=2.0.0 in c:\users\anael\anaconda3\lib\site-packages (from google-auth<2,
>=1.6.3->tensorboard<2.3.0,>=2.2.0->tensorflow==2.2.0rc2) (4.1.1)
Requirement already satisfied: pyasn1-modules>=0.2.1 in c:\users\anael\anaconda3\lib\site-packages (from google-auth<2,>
=1.6.3->tensorboard<2.3.0,>=2.2.0->tensorflow==2.2.0rc2) (0.2.8)
Requirement already satisfied: requests-oauthlib>=0.7.0 in c:\users\anael\anaconda3\lib\site-packages (from google-auth-
oauthlib<0.5,>=0.4.1->tensorboard<2.3.0,>=2.2.0->tensorflow==2.2.0rc2) (1.3.0)
Requirement already satisfied: importlib-metadata; python_version < "3.8" in c:\users\anael\anaconda3\lib\site-packages
(from markdown>=2.6.8->tensorboard<2.3.0,>=2.2.0->tensorflow==2.2.0rc2) (0.23)
Requirement already satisfied: pyasn1>=0.1.3 in c:\users\anael\anaconda3\lib\site-packages (from rsa<5,>=3.1.4; python_v
ersion >= "3"->google-auth<2,>=1.6.3->tensorboard<2.3.0,>=2.2.0->tensorflow==2.2.0rc2) (0.4.8)
Requirement already satisfied: oauthlib>=3.0.0 in c:\users\anael\anaconda3\lib\site-packages (from requests-oauthlib>=0.
7.0->google-auth-oauthlib<0.5,>=0.4.1->tensorboard<2.3.0,>=2.2.0->tensorflow==2.2.0rc2) (3.1.0)
Requirement already satisfied: zipp>=0.5 in c:\users\anael\anaconda3\lib\site-packages (from importlib-metadata; python_
version < "3.8"->markdown>=2.6.8->tensorboard<2.3.0,>=2.2.0->tensorflow==2.2.0rc2) (0.6.0)
Requirement already satisfied: more-itertools in c:\users\anael\anaconda3\lib\site-packages (from zipp>=0.5->importlib-m
etadata; python_version < "3.8"->markdown>=2.6.8->tensorboard<2.3.0,>=2.2.0->tensorflow==2.2.0rc2) (7.2.0)
Installing collected packages: tensorflow
  Found existing installation: tensorflow 2.2.0
    Uninstalling tensorflow-2.2.0:
      Successfully uninstalled tensorflow-2.2.0
Successfully installed tensorflow-2.2.0rc2
(base) PS C:\Users\anael>
```

Figure 162: Re-install tensorflow (2/2)

7.2 Extra Sentiment Analysis on Tweets

A second classification model was built with sentiment classes cut-off points at -0.02 and 0.2 to bin scores of ‘sentiment1’ and ‘sentiment2’ into positive, negative or neutral categories. With such criteria, the accuracy metric went down to 80.12%[23]. This confirms findings from the literature review where contribution of the neutral class improves overall accuracy.

Sentiment Analysis model with neutral class bound from -0.2 to +0.2 scores. Confusion matrix shows a prediction accuracy of 80.12% (Fig.163).

[23] This second approach was not retained for the technical report but is presented in the Configuration Manual, section 7.2. Extra Sentiment Analysis on Tweets

```
[ ]: ### Sentiment Analysis Model 2 ###

     # Using classes cutoff at -0.2 and +0.2

     '''
     positive class > 0.2
     neutral class < 0-02 or > 0.02
     negative class < -0.2
     '''
```

```
[34]: # For tweets dataset

      sentiment2_class = []

      for index, score in df2.iterrows():
          score = score['sentiment2']
          if score > 0.2:
              score_class = 1 #"positive"
          elif score < -0.2:
              score_class = -1 # "negative"
          else:
              score_class = 0 # "neutral"
          sentiment2_class.append(score_class)

          #print (polarity_class)
      df2['sentiment2_class'] = sentiment2_class # create new col. in df with output
```

```
sentiment1_class = []

for index, score in df2.iterrows():
    score = score['sentiment1']
    if score > 0.2:
        score_class = 1 # "positive"
    elif score < -0.2:
        score_class = -1 # "negative"
    else:
        score_class = 0 # "neutral"
    sentiment1_class.append(score_class)
    #print (sentiment_class)
df2['sentiment1_class'] = sentiment1_class # print values and add te col. in df

# Actual classes corresponds to 'sentiment1' (before text pre-processing) and Predicted
print ('Confusion Matrix :')
print(confusion_matrix(sentiment1_class, sentiment2_class))
print('Accuracy Score :', accuracy_score(sentiment1_class, sentiment2_class))
print ('Classification Report : ')
print(classification_report(sentiment1_class, sentiment2_class))
```

```
Confusion Matrix :
[[ 242   69    6]
 [ 340 4516  512]
 [  31  366  578]]
Accuracy Score : 0.8012012012012012
Classification Report :
              precision    recall  f1-score   support

          -1       0.39      0.76      0.52       317
           0       0.91      0.84      0.88      5368
           1       0.53      0.59      0.56       975

    accuracy                           0.80      6660
   macro avg       0.61      0.73      0.65      6660
weighted avg       0.83      0.80      0.81      6660
```

Figure 163: Neutral sentiment class bounded to -0.2 to 0.2

Additional visuals to illustrate the sentiment scores distribution across three classes and compare scores obtained before and after pre-processing (Fig.164).

```
[17]: # Boxplots to compare scores before and after pre-processing tweets
f, (ax1,ax2) = plt.subplots(1,2,figsize=(14,4))
bp_sentiment = sns.boxplot(x='sentiment1_class',y='sentiment1',
                hue='sentiment1_class', data=df2, palette="Set2", ax=ax1)
bp_polarity = sns.boxplot(x='sentiment2_class',y='sentiment2',
                hue='sentiment2_class', data=df2, palette="Set2", ax=ax2)
t = f.suptitle('Vizualizing Tweets Sentiment (1-before and 2-after pre-processing)', fontsize=14)
```

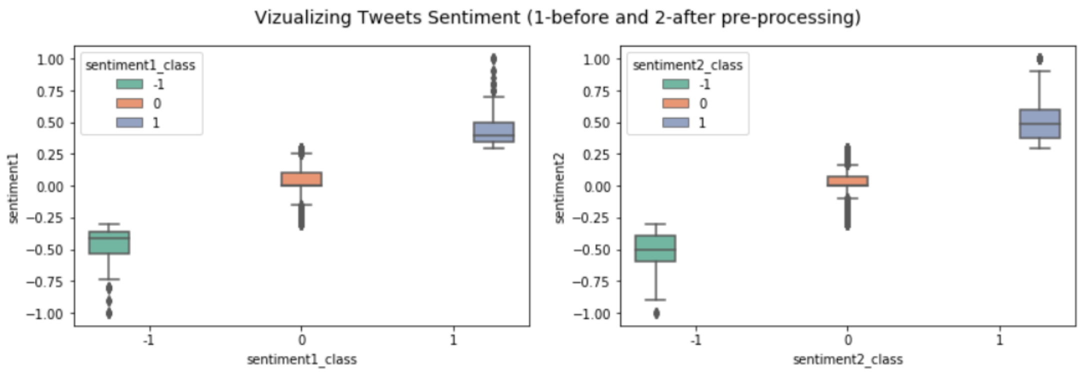


Figure 164: Tweets classification, before and after pre-processing

Score differences computed tweet by tweet, results are float and not in absolute values (Fig.165).

```
[10]: # Compute the difference between 'sentiment' and 'polarity'
df2['score_difference'] = df2['sentiment2']-df2['sentiment1']

# Five largest values in score_difference to see the significant misinterpretation of language before and after cleaning text
df2.nlargest(5, ['score_difference'])
```

[10]:

tweet_id	timestamp	user_id	lang	likes	retweets	sentiment1	text	cleaned	lemmatized_text	sentiment2	subjectivity	score_difference
1363843621711872	2020-02-22 23:43:12	923767416823386112	en	4	1	-0.350000	Oh boy, this is not good at all... #coronaviru...	oh boy good day incubation period coronavirus ...	['oh', 'boy', 'good', 'day', 'incubation', 'pe...	0.7	0.6	1.050000
1876783747731457	2020-03-22 23:57:52	229794164	en	0	0	-1.000000	Malik Supermarket apologises for 'disgusting p...	malik supermarket apologises disgusting profit...	['malik', 'supermarket', 'apologise', 'disgust...	0.0	0.0	1.000000
3541684081954822	2020-02-28 23:57:10	1172561767165300739	en	3	1	0.166667	South Korea has drive through tests. It takes ...	south korea drive tests takes min cdc generate...	['south', 'korea', 'drive', 'test', 'take', 'm...	1.0	0.3	0.833333
5716157996290048	2020-03-05 23:57:45	2194496264	en	1	1	-0.116667	A #CoronaVirus case in #Canada has no links to...	case links travel household contact good	['case', 'link', 'travel', 'household', 'conta...	0.7	0.6	0.816667
3552530176212992	2020-02-17 23:45:41	556445447	en	0	0	0.000000	Is there a stealth outbreak happening now in t...	stealth outbreak happening honolulu airport co...	['stealth', 'outbreak', 'happen', 'honolulu', ...	0.8	0.9	0.800000

Figure 165: Tweets with the highest score difference in sentiment1 vs. sentiment2

Manual calculation of the paired t-test (Fig. 166 and Fig. 167) and unpaired t-test (Fig. 168 and Fig.169) for learning purpose on Python. The result obtained manually for the paired t-test shows a rounding discrepancy compared to the test function integrated to Python library.

[8]:

```python
# T-test on paired (dependent samples) Step by step for learning purpose

# Code sourced from Brownlee, 2018 and adapted
# https://machinelearningmastery.com/how-to-code-the-students-t-test-from-scratch-in-python/

# function for calculating the t-test for two dependent samples
def dependent_ttest(data1, data2, alpha):
    # calculate means of each sample
    mean1, mean2 = mean(data1), mean(data2)
    # number of paired samples
    n = len(data1)
    # sum of the squared difference between observations
    d1 = sum([(data1[i]-data2[i])**2 for i in range(n)])
    # sum difference between observations
    d2 = sum([data1[i]-data2[i] for i in range(n)])
    # standard deviation of the difference between means
    sd = sqrt((d1 - (d2**2 / n)) / (n - 1))
    # standard error of the difference between the means
    sed = sd / sqrt(n)
    # calculate the t statistic
    t_stat = (mean1 - mean2) / sed
    # degrees of freedom
    degreef = n - 1
    # calculate the critical value
    cv = t.ppf(1.0 - alpha, degreef)
    # calculate the p-value
    p = (1.0 - t.cdf(abs(t_stat), degreef)) * 2.0
    # return everything
    return t_stat, degreef, cv, p

# assign the 2 dependent samples to compare
data1 = df2['sentiment1'].values.tolist()
data2 = df2['sentiment2'].values.tolist()
```

Figure 166: Manual paired t-test (1/2)

```python
# calculate the t test
alpha = 0.05
t_stat, degreef, cv, p = dependent_ttest(data1, data2, alpha)
print('t=%.3f, degreef=%d, cv=%.3f, p=%.3f' % (t_stat, degreef, cv, p))
# interpret via critical value
print('Result with critical value: ')
if abs(t_stat) <= cv:
    print('Accept null hypothesis that the means are equal.')
else:
    print('Reject the null hypothesis that the means are equal.')
# interpret via p-value
print('Result with p-value: ')
if p > alpha:
    print('Accept null hypothesis that the means are equal.')
else:
    print('Reject the null hypothesis that the means are equal.')
```

```
t=4.431, degreef=6659, cv=1.645, p=0.000
Result with critical value:
Reject the null hypothesis that the means are equal.
Result with p-value:
Reject the null hypothesis that the means are equal.
```

[9]:

```python
data1 = df2['sentiment1'].values.tolist()
data2 = df2['sentiment2'].values.tolist()

#compute means
mean1 = statistics.mean(data1)
mean2 = statistics.mean(data2)

# difference between means
diff_means = mean1 - mean2

print('Mean1  :', round(mean1,4), 'Mean2 :', round(mean2,4))
print('Difference between means: ',round(diff_means,4))
```

```
Mean1  : 0.0508 Mean2 : 0.0412
Difference between means:  0.0096
```

Figure 167: Manual paired t-test (2/2)

```
[17]: # T-test for independent samples step by step - Done for learning purpose
      # Results differ from Welch's test - Do not use for project

      # Code sourced from Brownlee, 2018 and adapted
      # https://machinelearningmastery.com/how-to-code-the-students-t-test-from-scratch-in-python/

      # Step by step
      from math import sqrt
      from numpy.random import seed
      from numpy.random import randn
      from numpy import mean
      from scipy.stats import sem
      from scipy.stats import t

      # function for calculating the t-test for two independent samples
      def independent_ttest(data1_news, data2_tweets, alpha):
          # calculate means
          mean1_news, mean2_tweets = statistics.mean(data1_news), statistics.mean(data2_tweets)
          # calculate standard errors
          se1, se2 = sem(data1_news), sem(data2_tweets)
          # standard error on the difference between the samples
          sed = sqrt(se1**2.0 + se2**2.0)
          # calculate the t statistic
          t_stat = (mean1_news - mean2_tweets) / sed
          # degrees of freedom
          df = len(data1_news) + len(data2_tweets) - 2
          # calculate the critical value
          cv = t.ppf(1.0 - alpha, df)
          # calculate the p-value
          p = (1.0 - t.cdf(abs(t_stat), df)) * 2.0
          # return everything
          return t_stat, df, cv, p
```

Figure 168: Manual unpaired t-test (1/2)

```
# Assign the two independent samples
data1_news = df_news['sentiment3'].values.tolist()
data2_tweets = df2['sentiment2'].values.tolist()
# calculate the t test
alpha = 0.05
t_stat, df, cv, p = independent_ttest(data1_news, data2_tweets, alpha)
print('t=%.3f, df=%d, cv=%.3f, p=%.3f' % (t_stat, df, cv, p))
# interpret via critical value
print('Result with critical value: ')
if abs(t_stat) <= cv:
    print('Accept null hypothesis that the means are equal.')
else:
    print('Reject the null hypothesis that the means are equal.')
# interpret via p-value
print('Result with p-value: ')
if p > alpha:
    print('Accept null hypothesis that the means are equal.')
else:
    print('Reject the null hypothesis that the means are equal.')

t=3.106, df=6816, cv=1.645, p=0.002
Result with critical value:
Reject the null hypothesis that the means are equal.
Result with p-value:
Reject the null hypothesis that the means are equal.
```

Figure 169: Manual unpaired t-test (2/2)

VADER sentiment analysis on a sample of tweets, “compound” is the aggregation of positive, negative and neutral scores computed. This lexicon is based on sentiment-related words and adapted to social media analysis. It takes into account punctuation and text case to compute polarity. This method was not retained for the project given that text was pre-processed with removing punctuations and lower-cased text; VADER is suitable for social media text feature but in the research project, news media content was also analysed. Therefore TextBlob sentiment lexicon was retained to perform the analysis on both corpuses.

11 sampled sentences are shown in Fig.170, with TextBlob polarity scores. Fig. 171 shows scores computed with VADER method.

TextBlob returns 3 positive and 1 negative polarity scores out of the sample (sentences 2, 4, 5 and 6). On the opposite, VADER returns only two negative compound scores, for sentences 4 and 9.

Sentence 4 is "The virus outbreak caused one million further deaths and it sucks". TextBlob assigns a score of -0.15 while VADER returns score of -0.3612, which is twice more negative.

```
[121]: # Examples with Covid
samples = ["Covid outbreak caused 1,000 further deaths",
           "Covid outbreak caused more than 1,000 further deaths",
           "The virus outbreak caused one million further deaths",
           "The virus outbreak caused one million further deaths and it sucks",
           "The new virus spreads quickly worldwide",
           "The USA face more than 150,000 deaths",
           "Covid continues spreading",
           "The virus continues spreading",
           "There is a shortage of facemasks",
           "We finally received facemask supplies",
           "Thank god we got facemasks"]

for s in samples:
    print(TextBlob(s).sentiment, s)

Sentiment(polarity=0.0, subjectivity=0.5) Covid outbreak caused 1,000 further deaths
Sentiment(polarity=0.25, subjectivity=0.5) Covid outbreak caused more than 1,000 further deaths
Sentiment(polarity=0.0, subjectivity=0.5) The virus outbreak caused one million further deaths
Sentiment(polarity=-0.15, subjectivity=0.4) The virus outbreak caused one million further deaths and it sucks
Sentiment(polarity=0.23484848484848483, subjectivity=0.4772727272727273) The new virus spreads quickly worldwide
Sentiment(polarity=0.5, subjectivity=0.5) The USA face more than 150,000 deaths
Sentiment(polarity=0.0, subjectivity=0.0) Covid continues spreading
Sentiment(polarity=0.0, subjectivity=0.0) The virus continues spreading
Sentiment(polarity=0.0, subjectivity=0.0) There is a shortage of facemasks
Sentiment(polarity=0.0, subjectivity=1.0) We finally received facemask supplies
Sentiment(polarity=0.0, subjectivity=0.0) Thank god we got facemasks
```

Figure 170: Covid-19 sample with TextBlob

```
[122]: # VADER Lexicon
       # From https://medium.com/analytics-vidhya/simplifying-social-media-sentiment-analysis-using-vader-in-python-f9e6ec6fc52f

       from vaderSentiment.vaderSentiment import SentimentIntensityAnalyzer
       analyser = SentimentIntensityAnalyzer()

       # define the function (Pandey, 2018)
       def sentiment_analyzer_scores(sentence):
           score = analyser.polarity_scores(sentence)
           print("{:-<40} {}".format(str(score), sentence))

[123]: for s in samples:
           print (sentiment_analyzer_scores(s))

       for s in samples:
           print(TextBlob(s).sentiment, s)

       {'neg': 0.0, 'neu': 1.0, 'pos': 0.0, 'compound': 0.0} Covid outbreak caused 1,000 further deaths
       None
       {'neg': 0.0, 'neu': 1.0, 'pos': 0.0, 'compound': 0.0} Covid outbreak caused more than 1,000 further deaths
       None
       {'neg': 0.0, 'neu': 1.0, 'pos': 0.0, 'compound': 0.0} The virus outbreak caused one million further deaths
       None
       {'neg': 0.2, 'neu': 0.8, 'pos': 0.0, 'compound': -0.3612} The virus outbreak caused one million further deaths and it sucks
       None
       {'neg': 0.0, 'neu': 1.0, 'pos': 0.0, 'compound': 0.0} The new virus spreads quickly worldwide
       None
       {'neg': 0.0, 'neu': 1.0, 'pos': 0.0, 'compound': 0.0} The USA face more than 150,000 deaths
       None
       {'neg': 0.0, 'neu': 1.0, 'pos': 0.0, 'compound': 0.0} Covid continues spreading
       None
       {'neg': 0.0, 'neu': 1.0, 'pos': 0.0, 'compound': 0.0} The virus continues spreading
       None
       {'neg': 0.286, 'neu': 0.714, 'pos': 0.0, 'compound': -0.25} There is a shortage of facemasks
       None
       {'neg': 0.0, 'neu': 1.0, 'pos': 0.0, 'compound': 0.0} We finally received facemask supplies
       None
```

Figure 171: Covid-19 sample with VADER

Length of news articles and first 100 tweets (before text pre-processing) in number of characters in Fig.172.

```
[112]: # Length of news articles (number of characters)
       articles_length = []

       for index, article in df_news.iterrows():
           article = article['Article Content']
           length= len(article)
           articles_length.append(length)
       print('Articles length (raw text): ', articles_length)

       Articles length (raw text):  [684, 2238, 4385, 5651, 4756, 5509, 3877, 4091, 2181, 1071, 1322, 9651, 10813, 3470, 2756, 12958, 5217, 11588, 5894,
       3482, 2817, 5485, 5848, 5797, 6090, 7370, 1398, 10507, 7208, 7738, 8194, 37079, 1739, 836, 1038, 767, 6312, 1344, 5450, 12134, 5733, 5234, 6285,
       11552, 6599, 5228, 5347, 2444, 3369, 5491, 3794, 5628, 5189, 4655, 5118, 4862, 37050, 828, 2920, 5625, 4529, 3717, 4291, 2195, 3286, 7507, 3791,
       4690, 5331, 2478, 4635, 2059, 6934, 4247, 6697, 5190, 6893, 9895, 4417, 35916, 3100, 5079, 2352, 2995, 4649, 2212, 5534, 2936, 5052, 4055, 8499,
       2355, 5349, 4967, 6898, 4640, 8569, 10933, 5629, 4856, 5703, 883, 2935, 2755, 3644, 6055, 5135, 3257, 4918, 5866, 11338, 3057, 6750, 1610, 1956,
       4942, 3194, 3797, 1347, 11151, 5249, 3861, 4671, 10751, 29127, 5477, 4702, 20845, 1188, 1071, 2271, 6760, 6056, 5257, 3806, 12832, 2271, 5135, 41
       67, 4401, 2440, 8016, 4802, 7921, 8990, 4069, 5430, 5291, 6468, 3500, 14596, 33933, 7082, 7450, 929, 6378, 3423, 8980]

[113]: # Length of tweets (number of characters)
       tweets_length = []

       for index, tweet in df2.iterrows():
           tweet = tweet['text']
           length= len(tweet)
           tweets_length.append(length)
       print('Tweets length: ', tweets_length[:100])

       Tweets length:  [156, 122, 201, 288, 134, 264, 306, 219, 324, 330, 112, 335, 205, 243, 193, 393, 86, 206, 268, 228, 270, 203, 367, 268, 211, 330,
       186, 230, 155, 105, 109, 107, 224, 314, 119, 272, 346, 277, 227, 216, 257, 324, 214, 151, 105, 118, 300, 300, 101, 207, 177, 171, 329, 78, 161, 1
       42, 125, 327, 45, 329, 280, 148, 138, 123, 153, 205, 72, 82, 162, 172, 193, 63, 45, 130, 233, 104, 45, 221, 198, 167, 112, 125, 141, 154, 185, 28
       1, 302, 123, 228, 79, 312, 33, 276, 234, 218, 166, 234, 68, 273, 210]
```

Figure 172: Length of news articles and tweets

8 References

Bold, A. (2019, February 7). *Sentiment Analysis - The Lexicon Based Approach*. Retrieved from Alpha Bold: https://alphabold.com/sentiment-analysis-the-lexicon-based-approach/