MA5851 A3 Document Number 2 Hugh McMullan

Task 2 - Web crawler and web scraping

a. Websites to be consumed

DJI drone users visit forums and post articles and comments to the following websites (see logos/links in Figure 1 below):

MavicPilots.com

PhantomPilots.com

FPVDronePilots.com

MatricePilots.com

CommercialDronePilots.com

InspirePilots.com

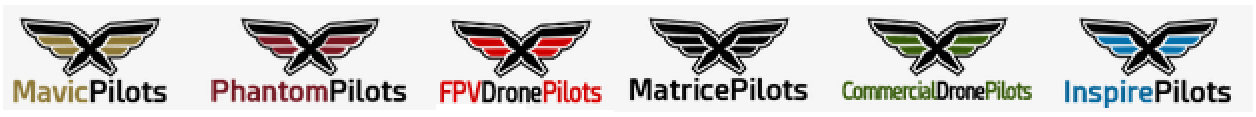


Figure 1: Drone-related websites for DJI drone users

b. Rationale for extracting the web content

Most Australian drone pilots use DJI drones, DJI has the dominant market share (about 70% of drones in use) source *DII*.

Commercial drone users strongly tend towards the *Matrice*, *CommercialDronePilots* and *Inspire* sites [for larger drones]. Private/hobbyist drone users tend to access the other 3 sites.

Each site has areas for users to post new topics, and to comment on other users' posts.

Posted text comments, that present drone users' **interests** and **concerns** and **sentiments**, have been accessed and assessed by:

(1) **searching** the forums for the phrase **casa**; and

(2) working through the lists of topic postsings returned by the search [which spread across multiple pages]; then

(3) reading every user comment against those topics.

c. Content coverage of the data extracted

The data items retrieved include:

(1) the **name of the site** accessed ['mavicpilots', 'phantompilots' etc];

(2) the **thread title** of the originally-posted topic [e.g. 'CASA rules on night flying ...' etc];

(3) the **date/time** the post was recorded;

(4) the **text** of the post/comment.

Based on the site name, a **site group** was recorded for each post ['Commercial' for the 'matrice', 'inspire' or 'commercialdronepilots' sites; or 'Private' for the other 3 sites].

The user identifier for the post was deliberately not recorded - to avoid the holding and potential spilling of any **personally-identifying information**

d. Complexity of the content layout

For each site, a search for 'casa' returned 1 or more pages of topic listings [up to 20 topics per page]. So a process was required to programmatically 'click' on the page-specific 'next page' [or 'page-number' links]. A maximum number of 10 pages of topics was used, for each site.

Each topic 'clicked into' returned 1 or more pages of posts for the clicked topic thread, up to 20 posts per page. While another 'page-number clicking' process could have been used, a limit of 20 posts per thread was convenient as a limiter on the total numbers of posts returned.

On the post pages, data items were *not* in named html fields - they needed to be extracted and parsed using specific *css* class types.

The datetime data was converted from an ISO8601-formatted string, to a python datetime data item.

e. Website/data copyright considerations

The data extracted from these websites was solely for the purposes of this assessment, and falls under the general category of *academic research*.

There are no other purposes for this extraction of data, commercial or otherwise.

No personally-identifiable information has been extracted, or used in any way in this process.

f. Metadata supplementation

All data needed for the assessment is obtained from the data extraction described here. No additional metadata was required.

g. Content extractor to export the data

BeautifulSoup was the preferred toolset for parsing pages. The site pages were not able to be retieved with simple *request* calls, so *Selenium* was used to access the sites, passing called page resonses direct to *soup* objects.

Each site was structured identically, so a process looping through a set of site names and url locations was used.

From each site's home page, a Search for 'casa' was effected by extending the base URL as follows [the 'search_identifier' was a constant 5- or 6-digit figure required by each site separately]:

```
# syntax for mavic/phantom etc websites
  search_url_base = url_base + "/search/" + search_identifier + "/?q=casa&o=relevance"
```

The Search would return multiple pages - those were iterated through for [arbitrarily-chosen] up to 10 pags of results, loading each reurned page of search results into a *BeautifulSoup* object [described below], using this logic:

```
for page in range(max_pages):
  url_page = search_url_base + "&page=" + str(page + 1)   # incorporate search page no
  soup = fetch_and_scrape_website(url_page)               # load the page into soup
```

Each returned *BeautifulSoup* object containing a web page of topic results was then iterated through, for each *thread* on the page, by searching the page for all occurrences of:

- the html 'h3' string, to get the thread *title*, and
- the *url extension* ['href'] leading to the page of posts for that particular thread.

For each thread found, a call was made to the *scrape_and_save_thread_pages* function, which itself would iterate through the individual posts found on that thread-related page.

```
# Run through all the threads found by the search
threads = soup.find_all('h3',{'class':'contentRow-title'})
items_count = 0
for thread in threads:
  href = thread.find("a")["href"]
  thread_name = thread.find("a").text

  thread_url = url_base + strip_thread(href)

  scrape_and_save_thread_pages(dronesite_name, dronesite_group, thread_name, thread_url,
```

Each thread page was seached for every html 'div' of class 'message-main' (to find the block of data for each comment posted), and that div was searched [again using html div classes] for these post-specific *datetime* and *text* fields.

The post data assembled was then saved by appending to a .csv file, as per the code below:

```
# Save the posts data
list_to_save = [dronesite_name, dronesite_group, thread_name, thread_no, post_datetime, po
save_post_to_csv(output_filename, list_to_save)
```

h. Relevant python coding

```
#Importing Selenium packages for crawling and scraping web pages
!pip install selenium
from selenium import webdriver
from selenium.webdriver.common.keys import Keys

# !apt-get update # to update ubuntu to correctly run apt install
# !apt install chromium-chromedriver
# !cp /usr/lib/chromium-browser/chromedriver /usr/bin

import sys
sys.path.insert(0,'/usr/lib/chromium-browser/chromedriver')
from selenium import webdriver
chrome_options = webdriver.ChromeOptions()
chrome_options.add_argument('--headless')
chrome_options.add_argument('--no-sandbox')
chrome_options.add_argument('--disable-dev-shm-usage')


#Import other libraries
from bs4 import BeautifulSoup     # BeautifulSoup for parsing scraped pages
import re                         # Regex functions
import csv                        # Writing to csv file
import datetime                   # Generating timestamps


# function to fetch and scrape a website page
def fetch_and_scrape_website(url):
  # Fetch the web page [with Selenium]
  print("\nFETCHING ...", url)
  driver.get(url)

  # Selenium hands the page source to Beautiful Soup
  soup = BeautifulSoup(driver.page_source, 'lxml')

  # Closes all browser windows and safely ends the session
  #driver.close()

  # return the soup object
  return(soup)


def strip_thread(thread):
  rtn = thread
  if thread[-12:-7] == "post-":
    rtn = thread[:-12]
  return(rtn)


# function to scrape a set of SEARCH RESULT pages using a page_no
def scrape_and_save_search_pages(dronesite_details_list, max_pages, max_items, output_file
  dronesite_name = dronesite_details_list[0]
  url_base = dronesite_details_list[1]
  search_identifier = dronesite_details_list[2]
  dronesite_group = dronesite_details_list[3]
  # print(dronesite_name)
  # print(dronesite_group)
  # print(url_base)
  # print(search_identifier)

  # syntax for mavic/phantom etc websites
  search_url_base = url_base + "/search/" + search_identifier + "/?q=casa&o=relevance"
  # print("Search thread:", search_url_base)
  # print("\n")

  for page in range(max_pages):
    url_page = search_url_base + "&page=" + str(page + 1)   # incorporate search page no
    soup = fetch_and_scrape_website(url_page)               # load the page into soup

    # Run through all the threads found by the search
    threads = soup.find_all('h3',{'class':'contentRow-title'})
    items_count = 0
    for thread in threads:
      href = thread.find("a")["href"]
      thread_name = thread.find("a").text

      # Print this block so scraping can be monitored
      print("\n=================================================================")
      print("Search thread:", thread_name)
      print("=================================================================")

      thread_url = url_base + strip_thread(href)

      scrape_and_save_thread_pages(dronesite_name, dronesite_group, thread_name, thread_ur

      items_count += 1
      if items_count > max_items:
          break


# function to scrape the FIRST THREAD page (even if there are multiple pages)
def scrape_and_save_thread_pages(dronesite_name, dronesite_group, thread_name, thread_url,
  thread_no = thread_url[-7:]
  thread_no = thread_no[:-1]
  #print(thread_no, thread_name, thread_url)

  soup = fetch_and_scrape_website(thread_url)               # load the page into soup

  # Run through all the threads found by the search
  posts = soup.find_all('div',{'class':'message-main'})
  items_count = 0
  for post in posts:
    # Find and extract post data from the thread html
    post_datetime = post.find('time')["datetime"]
    post_article = post.find('article')
    post_text = post_article.find('div',{'class':'bbWrapper'})
    post_text = post_text.text
    # print("\n--------------------------------------------------------------------")
    # print("Posted at:", post_datetime)
    # print("--------------------------------------------------------------------")
    # print("Post text:", post_text)

    # Save the posts data
    list_to_save = [dronesite_name, dronesite_group, thread_name, thread_no, post_datetime
    save_post_to_csv(output_filename, list_to_save)

    # Stop the loop if max no of post items is reached
    items_count += 1
    if items_count > max_items:
        break


# Write the passed-in list of post details to a line in the .csv file
def save_post_to_csv(output_filename, list_input):
  try:
    with open(output_filename, "a" ) as fopen:    # Open the csv file
      csv_writer = csv.writer(fopen)
      csv_writer.writerow(list_input)
  except:
      return False



# Set a list website targets for posts scraping
drone_website_targets = [
                         ["MavicPilots", "https://mavicpilots.com", "707065", "Private"],
                         ["PhantomPilots", "https://phantompilots.com", "213313", "Private
                         ["FPVDronePilots", "https://fpvdronepilots.com", "15671", "Privat
                         ["CommercialDronePilots", "https://commercialdronepilots.com", "1
                         ["MatricePilots", "https://matricepilots.com", "49374", "Commerci
                         ["InspirePilots", "https://inspirepilots.com", "56174", "Commerci
]

# Set arbitrary limits to avoid too many calls to website(s)
max_page_calls = 10
max_items_per_page = 20

# Initialise dronesite_output file
datetime_now = datetime.datetime.now()
timestamp = datetime_now.strftime("%Y%m%d%H%M")
dronesite_output_filename = "DronePosts_" + timestamp + ".csv"
#print(dronesite_output_filename)
```

```
# Run through the 'CASA' Search pages for each target website
for dronesite_details_list in drone_website_targets:
  scrape_and_save_search_pages(dronesite_details_list, max_page_calls, max_items_per_page,
```

i. Demonstration of the application running

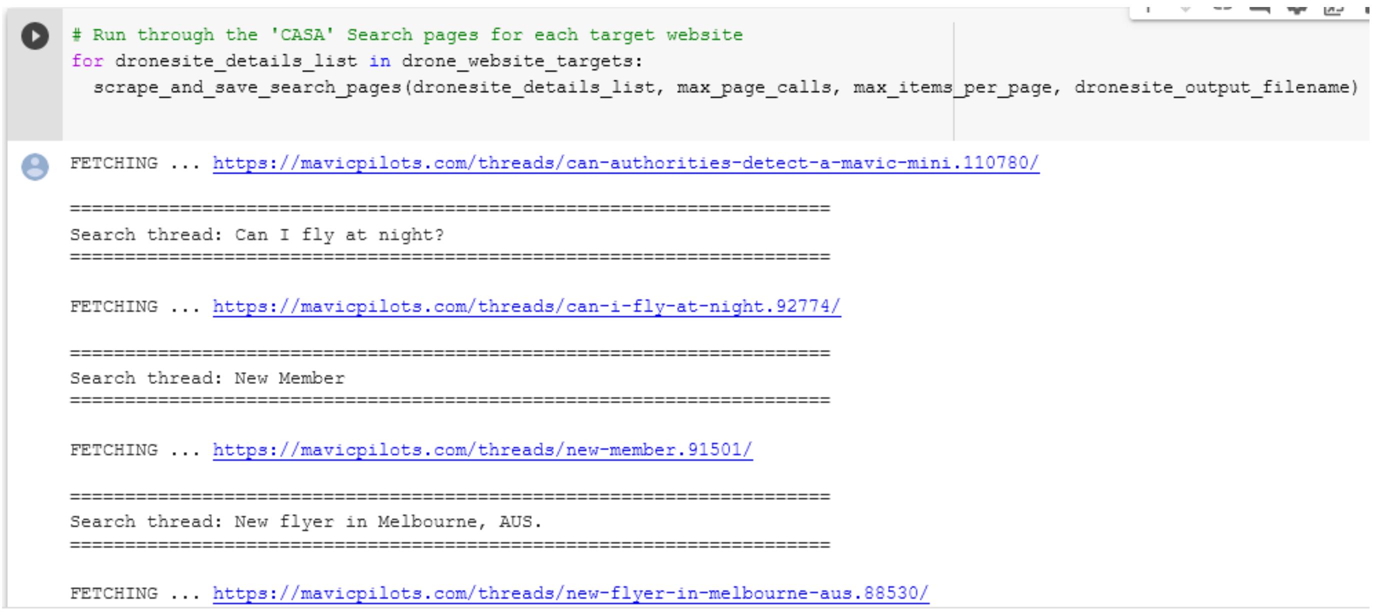
SAMPLE SCREENSHOT DURING PRODUCTION RUN

Figure 2 below shows a brief screenshot used to monitor **the production run**, crawling and scraping and exporting process across all 6 sites, up to 10 pages of search results per site, up to 20 posts per thread on the seach pages.

In this run, the debug monitoring output was turned OFF.

The monitoring listed just each Selenium page access.

The production process ran for approximately 5 hours.



```
# Run through the 'CASA' Search pages for each target website
for dronesite_details_list in drone_website_targets:
  scrape_and_save_search_pages(dronesite_details_list, max_page_calls, max_items_per_page, dronesite_output_filename)

FETCHING ... https://mavicpilots.com/threads/can-authorities-detect-a-mavic-mini.110780/

======================================================================
Search thread: Can I fly at night?
======================================================================

FETCHING ... https://mavicpilots.com/threads/can-i-fly-at-night.92774/

======================================================================
Search thread: New Member
======================================================================

FETCHING ... https://mavicpilots.com/threads/new-member.91501/

======================================================================
Search thread: New flyer in Melbourne, AUS.
======================================================================

FETCHING ... https://mavicpilots.com/threads/new-flyer-in-melbourne-aus.88530/
```

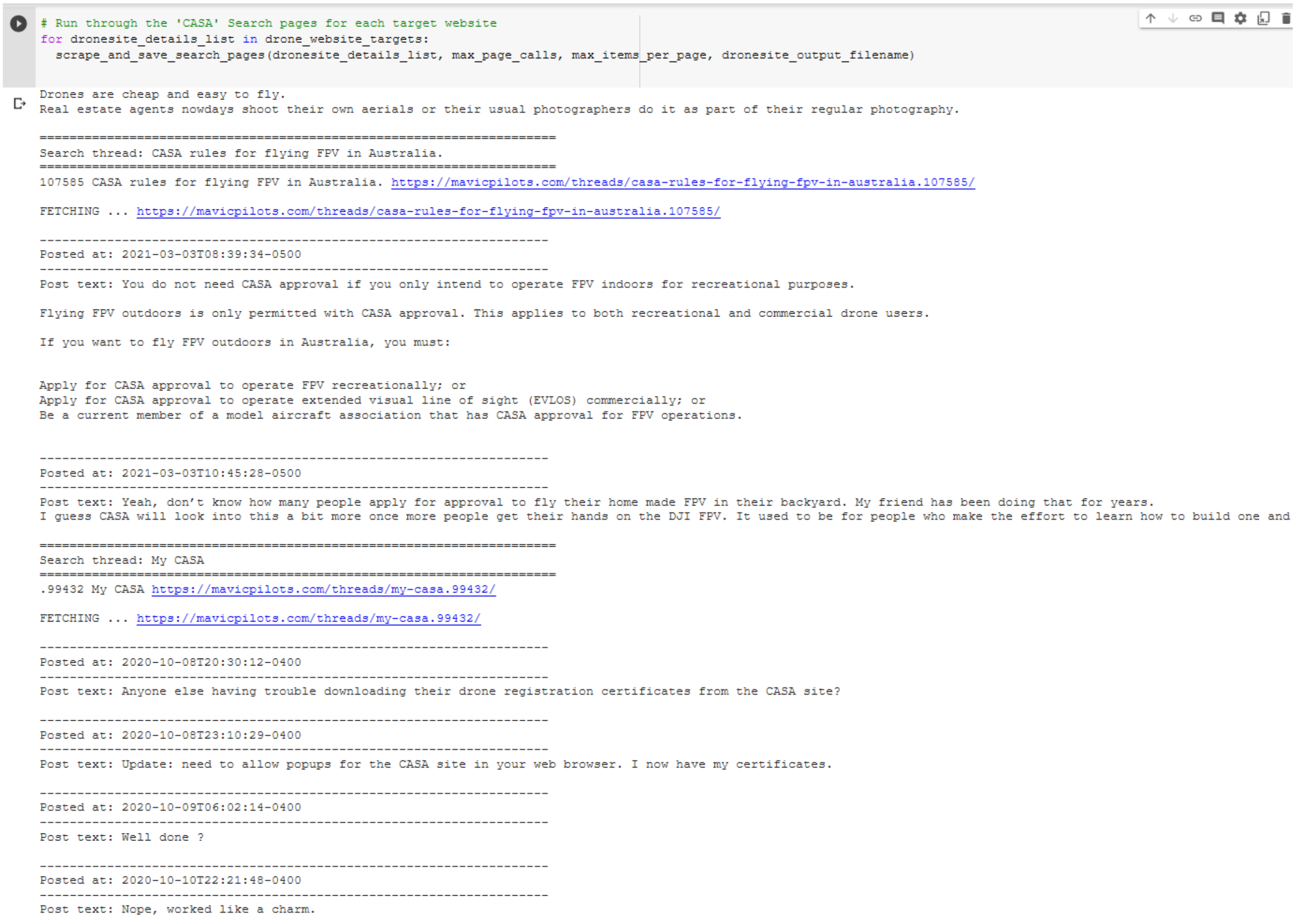
Figure 2: Screenshot iterating the 'casa' search/threads/posts pages

SAMPLE SCREENSHOT DURING TEST RUN

Figure 3 below shows a screenshot from a **limited test process**, with **all monitoring messages** turned ON [including displaying parsed text].

This test process searched just 1 site, maximum 2 search pages, max 4 posts per thread.

This test process ran for under 1 minute.



```
# Run through the 'CASA' Search pages for each target website
for dronesite_details_list in drone_website_targets:
  scrape_and_save_search_pages(dronesite_details_list, max_page_calls, max_items_per_page, dronesite_output_filename)
```

```
Drones are cheap and easy to fly.
Real estate agents nowdays shoot their own aerials or their usual photographers do it as part of their regular photography.

=====================================================================
Search thread: CASA rules for flying FPV in Australia.
=====================================================================
107585 CASA rules for flying FPV in Australia. https://mavicpilots.com/threads/casa-rules-for-flying-fpv-in-australia.107585/

FETCHING ... https://mavicpilots.com/threads/casa-rules-for-flying-fpv-in-australia.107585/

--------------------------------------------------------------------
Posted at: 2021-03-03T08:39:34-0500
--------------------------------------------------------------------
Post text: You do not need CASA approval if you only intend to operate FPV indoors for recreational purposes.

Flying FPV outdoors is only permitted with CASA approval. This applies to both recreational and commercial drone users.

If you want to fly FPV outdoors in Australia, you must:


Apply for CASA approval to operate FPV recreationally; or
Apply for CASA approval to operate extended visual line of sight (EVLOS) commercially; or
Be a current member of a model aircraft association that has CASA approval for FPV operations.


--------------------------------------------------------------------
Posted at: 2021-03-03T10:45:28-0500
--------------------------------------------------------------------
Post text: Yeah, don't know how many people apply for approval to fly their home made FPV in their backyard. My friend has been doing that for years.
I guess CASA will look into this a bit more once more people get their hands on the DJI FPV. It used to be for people who make the effort to learn how to build one and

=====================================================================
Search thread: My CASA
=====================================================================
.99432 My CASA https://mavicpilots.com/threads/my-casa.99432/

FETCHING ... https://mavicpilots.com/threads/my-casa.99432/

--------------------------------------------------------------------
Posted at: 2020-10-08T20:30:12-0400
--------------------------------------------------------------------
Post text: Anyone else having trouble downloading their drone registration certificates from the CASA site?

--------------------------------------------------------------------
Posted at: 2020-10-08T23:10:29-0400
--------------------------------------------------------------------
Post text: Update: need to allow popups for the CASA site in your web browser. I now have my certificates.

--------------------------------------------------------------------
Posted at: 2020-10-09T06:02:14-0400
--------------------------------------------------------------------
Post text: Well done ?

--------------------------------------------------------------------
Posted at: 2020-10-10T22:21:48-0400
--------------------------------------------------------------------
Post text: Nope, worked like a charm.
```

Figure 3: Screenshot of TEST process running with all monitoring turned ON

j. Methodology of process, cleaning and storing harvested data for NLP tasking

Saved extracted posts data: During the crawling and scraping process [as described above] each extracted post was saved, one-by-one as they were created, to a .csv file.

The output file name was set before the run started:

```
dronesite_output_filename = "DronePosts_" + timestamp + ".csv"
```

The **code for the cleaning process was initialised** as follows:

```
# Import libraries needed for this notebook
import pandas as pd
import numpy as np
from datetime import datetime, timedelta
import matplotlib.pyplot as plt
```

```
# The data files are located on the Google Drive, so do a drive.mount
from google.colab import drive
drive.mount('/content/drive')
```

```
    Mounted at /content/drive
```

The actual output file name for the production run was: **DronePosts_202112010705.csv**

To start the pre-processing work, the data was retrieved from that file [located on Google Drive, for convenience working with *Google Colab*.

The data was read back in from the file as follows:

```
# Read in the DronePosts.csv data, created in the Part 1 notebook
dronepost_header_list = ["dronesite_name", "dronesite_group", "thread_name", "thread_no",
dfDronePosts = pd.read_csv('/content/drive/MyDrive/DronePosts_202112010705.csv', names=dro
display(dfDronePosts)
```

	dronesite_name	dronesite_group	thread_name	thread_no	post_datetime	
0	MavicPilots	Private	Latest CASA newsletter re VLOS	117004	2021-10-07T19:15:52-0400	..\ ı VLC
1	MavicPilots	Private	Latest CASA newsletter re VLOS	117004	2021-10-07T19:27:14-0400	I'm gu makir
2	MavicPilots	Private	Latest CASA newsletter re VLOS	117004	2021-10-07T21:24:11-0400	If i the ex
3	MavicPilots	Private	Latest CASA newsletter re VLOS	117004	2021-10-07T22:07:39-0400	CAS to tell
4	MavicPilots	Private	Latest CASA newsletter re VLOS	117004	2021-10-07T23:27:14-0400	\n s
...	...	...	...	...	...	
			Price of		2015 06	

For sorting and selecting by month and year posted, 2 additional fields *post_YYYY* and *post_YYMM* were added as follows:

```
dfDronePosts['post_YYMM'] = dfDronePosts['post_datetime'].apply(lambda s: datetime.fromiso
dfDronePosts['post_YYYY'] = dfDronePosts['post_YYMM'].apply(lambda s: "20" + s[:-2])
display(dfDronePosts)
```

	dronesite_name	dronesite_group	thread_name	thread_no	post_datetime	
0	MavicPilots	Private	Latest CASA newsletter re VLOS	117004	2021-10-07T19:15:52-0400	..\ r VLC
1	MavicPilots	Private	Latest CASA newsletter re VLOS	117004	2021-10-07T19:27:14-0400	I'm gu makir
2	MavicPilots	Private	Latest CASA newsletter re VLOS	117004	2021-10-07T21:24:11-0400	If i the ex
3	MavicPilots	Private	Latest CASA newsletter re VLOS	117004	2021-10-07T22:07:39-0400	CAS to tell
4	MavicPilots	Private	Latest CASA newsletter re	117004	2021-10-07T23:27:14-	\n s

Numbers of posts by Site by Year

To select the best numbers of posts (between 5-10,000, spread across Commercial and Private drone users), 2 pivot tables and a stacked bar chart were generated:

```
pivotPostsBySite = dfDronePosts.pivot_table(values='dronesite_group', index='post_YYYY',
                                            columns='dronesite_name', aggfunc='count', fil
pivotPostsBySite
```

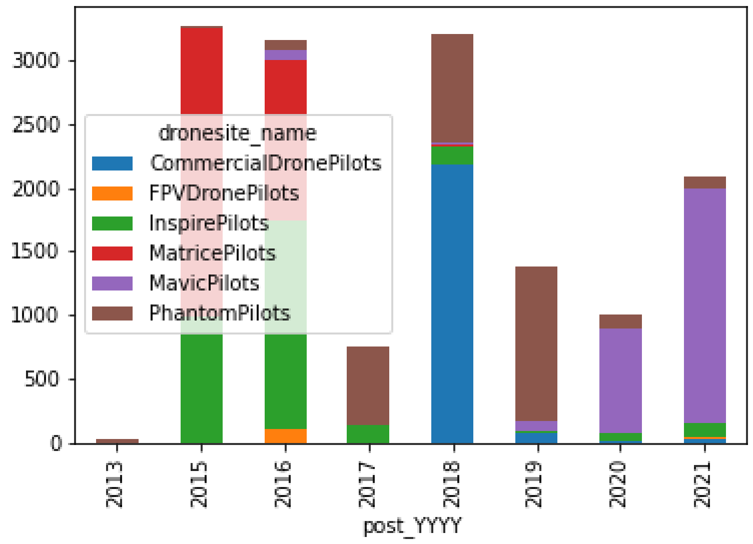
dronesite_name	CommercialDronePilots	FPVDronePilots	InspirePilots	MatricePilots
post_YYYY				
2013	0	0	0	0
2015	0	0	984	2278
2016	0	110	1637	1255
2017	0	0	139	1
2018	2188	0	143	9
2019	76	0	17	0
2020	14	0	52	0
2021	17	30	100	0

```
pivotPostsByGroup = dfDronePosts.pivot_table(values='dronesite_name', index='post_YYYY',
                                             columns='dronesite_group', aggfunc='count', f
pivotPostsByGroup
```

dronesite_group	Commercial	Private
post_YYYY		
2013	0	20
2015	3262	2
2016	2892	270
2017	140	609
2018	2340	867
2019	93	1281

```
dfDronePosts.groupby('post_YYYY')['dronesite_name']\
    .value_counts()\
    .unstack(level=1)\
    .plot.bar(stacked=True)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f7624489a10>
```



That lead to a choice to **take posts from 2018 onwards**

```
dfSince2018 = dfDronePosts[dfDronePosts['post_YYYY'] >= '2018'][['dronesite_group', 'threa
display(dfSince2018)
```

	dronesite_group	thread_name	post_YYMM	post_YYYY	post_text
0	Private	Latest CASA newsletter re VLOS	2110	2021	..\nLatest CASA newsletter has VLOS article wh...
1	Private	Latest CASA newsletter re	2110	2021	I'm guessing they're making
		VLOS			expectation is

Exploratory Data Analysis on extracted data

The dataframe containg the scraped data [since 2018] had the shape below:

```
dfSince2018.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 7668 entries, 0 to 14842
Data columns (total 5 columns):
 #   Column           Non-Null Count  Dtype 
---  ------           --------------  ----- 
 0   dronesite_group  7668 non-null   object
 1   thread_name      7668 non-null   object
 2   post_YYMM        7668 non-null   object
 3   post_YYYY        7668 non-null   object
 4   post_text        7667 non-null   object
dtypes: object(5)
memory usage: 359.4+ KB
```

There was a mismatch in column lengths - corrected by dropping NA values:

```
dfSince2018 = dfSince2018.dropna()
dfSince2018
```

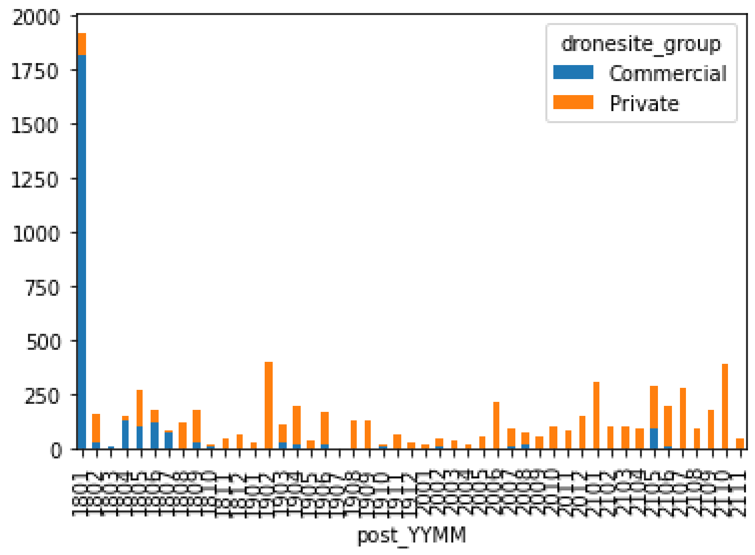
	Unnamed: 0	dronesite_group	thread_name	post_YYMM	post_YYYY	post_text	
0	0	Private	Latest CASA newsletter re VLOS	2110	2021	..\nLatest CASA newsletter has VLOS article wh...	V
			Latest CASA			I'm guessing	

A histogram showed an imbalanced pile of Commercial posts at the start of 2018. These were retained, initially, to get a reasonable number of posts for the topic modelling.

			Latest CASA			If it's like the	

```
dfSince2018.groupby('post_YYMM')['dronesite_group']\
    .value_counts()\
    .unstack(level=1)\
    .plot.bar(stacked=True)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f761ec81350>
```



The extracted data was saved, and a new dataframe **dfPreppedText** was used to keep the extracted data separate from the pre-processed data, to enable the pre=processing to be restarted if required.

```
dfSince2018.to_csv('/content/drive/MyDrive/DronePostsSince2018_202112041100.csv')
```

```
dfPreppedText = dfDronePostsSince2018
```

NLP pre-processing libraries were loaded

```
# Import libraries needed for this notebook
import re

import nltk
```

```
from nltk.tokenize import RegexpTokenizer, word_tokenize
from nltk.stem import PorterStemmer
from nltk.corpus import stopwords
nltk.download('stopwords')
nltk.download('punkt')
nltk.download('wordnet')

from sklearn.feature_extraction.text import CountVectorizer

from wordcloud import WordCloud
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
```

Constants were established for code readability

```
# NLTK constants
PATTERN_S = re.compile("\'s")           # matches `'s` from text
PATTERN_R = re.compile("\r")            #matches `\r`
PATTERN_N = re.compile("\n")            #matches `\r` and `\n`
PATTERN_PUNC = re.compile(r"[^\w\s]")   # matches all non 0-9 A-z whitespace
STOPWORDS = set(stopwords.words('english'))
```

Functions were set up to remove unwanted characters and convert text to lowercase, remove stopwords, and store the resulatant text in a new colum *cleaned_text*

```
# function for text cleaning
def clean_text_using_nltk(text):
    text = text.lower()                     # Convert text to lowercase

    text = re.sub(PATTERN_S, ' ', text)     # Remove apostrophe-s
    text = re.sub(PATTERN_R, ' ', text)     # Remove escaped \r characters
    text = re.sub(PATTERN_N, ' ', text)     # Remove escaped \n characters
    text = re.sub(PATTERN_PUNC, ' ', text)  # Remove non 0-9 A-z whitespace

    return text

# Apply the text cleaning function to the target_text
dfPreppedText['cleaned_text'] = dfPreppedText['target_text'].apply(lambda s: clean_text_us


# function to remove stopwords
def remove_stopwords(text):
    no_stopword_text = [w for w in text.split() if not w in STOPWORDS]
    return ' '.join(no_stopword_text)

# Apply the stopwords removal function to the corpus text
dfPreppedText['cleaned_text'] = dfPreppedText['cleaned_text'].apply(lambda s: remove_stopw
```

A function was established to tokenise the cleaned_text and store the result in a new column *tokenized_text*

```
# function to tokenize text
def apply_tokenizer(text):
    tokenizer = RegexpTokenizer('[a-zA-Z]\w+\'?\w*')    # use NLTK Regexp tokenizer
    return tokenizer.tokenize(text)

# Apply the stopwords removal function to the corpus text
dfPreppedText['tokenized_text'] = dfPreppedText['cleaned_text'].apply(lambda s: apply_toke
```

Now also STEMMING the text as well

When Topic Modelling [see Document 3] was carried, the key words included **both** 'fly' and 'flying'.

That was unsatisfactory, so STEMMING was added, saving the new field as *stemmed_text*:

```
# function to stem text
def apply_stemmer(text):
  stemmer=PorterStemmer()
  words = [stemmer.stem(word) for word in text]
  return words

# Apply the stemming function to the tokenized corpus text
dfPreppedText['stemmed_text'] = dfPreppedText['tokenized_text'].apply(lambda s: apply_stem
```

Finally the pre-processed data was stored into .csv as well

```
dfPreppedText.to_csv('/content/drive/MyDrive/DronePostsPreppedText_202112051330.csv')
```

Last, separate dataframes were selected [and saved] for **Commercial** and **Private** [drone users]:

```
# Separate COMMERCIAL corpus from dfPreppedText
dfCommercial = dfPreppedText[dfPreppedText["dronesite_group"]=="Commercial"]


dfCommercial.to_csv('/content/drive/MyDrive/DronePostsCommercial_202112051330.csv')


# Separate PRIVATE corpus from dfPreppedText
dfPrivate = dfPreppedText[dfPreppedText["dronesite_group"]=="Private"]


dfPrivate.to_csv('/content/drive/MyDrive/DronePostsPrivate_202112051330.csv')
```

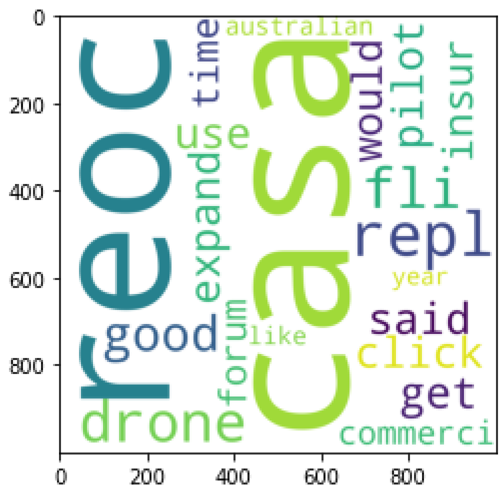
Exploratory Data Analysis on data ready for NLP

Just as simple pre-test before commencing the NLP, WordClouds were generated for the Commercial and Private datasets, showing the 'Top 20' words in each

```
# Top 20 Words in Commercial posts
comm_top20_tuples = most_frequent_words([word for post in dfCommercial.stemmed_text for wo
comm_top20_dictionary = convert_tuples_to_dictionary(comm_top20_tuples)
print(comm_top20_dictionary)

# WordCloud for Top 20 Words in Commercial posts
comm_top20_wordcloud = WordCloud(background_color="white", width=1000, height=1000,
                                 max_words=20, relative_scaling=0.5,
                                 normalize_plurals=False).generate_from_frequencies(comm_t
plt.imshow(comm_top20_wordcloud)
```

```
{'casa': 2818, 'reoc': 2465, 'repl': 2256, 'drone': 1921, 'fli': 1855, 'good': 1403,
<matplotlib.image.AxesImage at 0x7f1b0e072e90>
```



```
# Top 20 Words in Commercial posts
priv_top20_tuples = most_frequent_words([word for post in dfPrivate.stemmed_text for word
priv_top20_dictionary = convert_tuples_to_dictionary(priv_top20_tuples)
print(priv_top20_dictionary)

# WordCloud for Top 20 Words in Commercial posts
priv_top20_wordcloud = WordCloud(background_color="white", width=1000, height=1000,
                                 max_words=20, relative_scaling=0.5,
                                 normalize_plurals=False).generate_from_frequencies(priv_t
plt.imshow(priv_top20_wordcloud)
```

```
{'fli': 6922, 'drone': 6589, 'said': 2831, 'casa': 2523, 'click': 2415, 'expand': 238
<matplotlib.image.AxesImage at 0x7f1b04f06510>
```

0
200
click
know
rule
would
fli
go
ight
get

These simple WordClouds are not significant - but still interesting.

Commercial users require **Remote Operator Certificate** (known as the '**ReOc**'). This ReOc is extremely difficult and expensive to obtain, onerous to report on and comply with, it allows businesses to provide commercial drone services - and it is the bane of commercial operators' existences. So to see that in the WordCloud for these operators is encouraging.

References

CommercialDronePilots.com: https://CommercialDronePilots.com

Drone Industry Insights. (2021). Drone Manufacturers Ranking. Retrieved from Drone Industry Insights: https://droneii.com/product/drone-manufacturers-ranking

FPVDronePilots. (2021). FPVDronePilots.com. Retrieved from FPVDronePilots.com: https://FPVDronePilots

InspirePilots. (2021). InspirePilots.com. Retrieved from InspirePilots.com: https://InspirePilots.com

MatricePilots. (2021). MatricePilots.com. Retrieved from MatricePilots.com: https://MatricePilots.com

MavicPilots. (2021). MavicPilots.com. Retrieved from MavicPilots.com: https://MavicPilots.com

PhantomPilots. (2021). PhantomPilots.com. Retrieved from PhantomPilots.com: https://phantompilots.com/