Analysis and Classification of Tweets to investigate COVID response.

0. Introduction

This report explores twitter data using tweets that contain information about Covid-19 with the end goal being a statistical model that can predict if a tweet contains misinformation about Covid-19.

This report is split up into 4 sections.

1. Task A Pre-processing: Getting the data ready for the exploratory and statistical analysis
2. Task B Text Analysis: Exploring the data and investigating the difference between groups of tweets (groups being the annotations)
3. Task C Affect Analysis: Using a lexicon, that is a set dictionary of words to match up with the words contained in the tweets to see if the tweet contains positive or negative sentiment.
4. Task D Text Classificiation: Building a predictive model to predict whether a tweet contains misinformation or valid information.

The end result of this project is building a classifier that can detect if tweets contain misinformation in regards to covid. We built a Support Vector Machine model that predicts with 82.4% accuracy if a tweet contains misinformation or not.

1. Task A Pre-processing

This section carries out the pre-processing steps outlined in Task A with the exception of hashtags. In this section hashtags are not removed because the hashtags will be used later on in the analysis.

Some further cleaning steps are also done, which will be highligted below.

1.1 Load libraries

Although the list above imports packages, not all are used. Some libraries are nice to have with NLP for exploratory purposes . The main packages we use for this report are:

- Spacy: For pre-processing of the tweets
- Textacy: For extraction of word types
- Nltk: For the vader dictionary and affect analysis
- sklearn SVM: In order to build a SVM model

In [280…

```
#!pip install squarify
#!pip install textacy
import nltk
from nltk.tokenize import word_tokenize
from nltk.stem import WordNetLemmatizer
from nltk.stem.snowball import SnowballStemmer
from nltk.stem.porter import *
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

import re
from nltk.corpus import stopwords
from nltk.stem.wordnet import WordNetLemmatizer
from sklearn.feature_extraction.text import CountVectorizer, TfidfTransformer
from sklearn.naive_bayes import MultinomialNB
from sklearn.model_selection import train_test_split
from sklearn.pipeline import Pipeline
from sklearn.metrics import confusion_matrix, classification_report,accuracy_score
```

Read in the data

In [281…

```
import pandas as pd
df = pd.read_csv('Data/ms4h05_data_anon.csv')
#print(df.to_string())
```

1.2 Change @twitterhandle to @user

Tweets that has a user mention, for example @JoeTalkShow is replaced with @user. An example tweet with a user mention is shown as a demonstration, and after using regular expressions to pick up user mentions and change them to @user, the same tweet is shown, having @user where @JoeTalkShow was previously

Example of a tweet with a mention, we can see @JoeTalkShow host appears in this tweet:

In [282…

```
df["text"][0] #example of a tweet with @username
```

Out[282…

```
'PAGS PARODY! Plus, Joe talks coronavirus and has special guest Peter J Hasson on his new book. - @JoeTalkShow https://t.co/IJE4gZwI0V'
```

We use regular expression to pick up any mention and replace it with @user

In [283…

```
import re
df['text'] = [re.sub(r'@\w+', '@User', x) for x in df["text"]] # eliminate usernames
```

Now look at the same tweet above and how the mention @JoeTalkShow has changed to @User

In [284…

```
df['text'][0] #above changes @JoeTalkShow to @User
```

Out[284…

```
'PAGS PARODY! Plus, Joe talks coronavirus and has special guest Peter J Hasson on his new book. - @User https://t.co/IJE4gZwI0V'
```

1.3 Clean tweets

Now we clean the tweets so they don't impact the analysis. The following steps were taken so tweets and the words are cleaned. If there are any characters next to valid words that may impact how we count up the frequency of words. For example, we eliminate brackets from this text "(good)" to make "good" and therefore the original text can be bundled with the other words "good" in corpus.

Here are the steps we will take to clean up the text, again using regular expressions:

1. Convert html expressions such as \n and & to characters
2. Delete any html tags such as <'tab'>
3. We will keep URLS as they may help for the classification in Task D. Sometimes it's good to remove URLS from a text however in this case perhaps certain tweets with misinformation often link to a website so it could be possible that a URL helps to predict a tweet with misinformation.
4. Eliminate any brackets
5. Clean up hashes that don't form a hashtag. So we remove a text like this: &#- but we keep text like this "#hello".
6. Clean up hyphens
7. Clean up white spaces to just one white space

We create a program that creates the above steps and after we apply that to the data.

Here is a program that will clean up the text:

In [285…

```
import html

def clean(text):
    # convert html escapes like &amp; to characters.
    text = html.unescape(text)
    # tags like <tab>
    text = re.sub(r'<[^<>]*>', ' ', text)
    # markdown URLs like [Some text](https://....)
    #text = re.sub(r'\[([^\[\]]*)\]\([^\(\)]*\)', r'\1', text)
    # text or code in brackets like [0]
    text = re.sub(r'\[[^\[\]]*\]', ' ', text)
    # standalone sequences of specials, matches &# but not #cool
    text = re.sub(r'(?:^|\s)[&#<>{}\[\]+|\\:-]{1,}(?:\s|$)', ' ', text)
    # standalone sequences of hyphens like --- or ==
    text = re.sub(r'(?:^|\s)[\-=\+]{2,}(?:\s|$)', ' ', text)
    # sequences of white spaces
    text = re.sub(r'\s+', ' ', text)
    #no hashtags
    #text = re.sub('\#[\w\_]+',' ', text)
    return text.strip()
```

Apply the cleaning program we just created to all the tweets and call these cleaned tweets "clean_text" in our dataframe.

In [286…

```
df['clean_text'] = df['text'].map(clean)
```

Example tweet before clean:

In [287…

```
df['text'][352]
```

Out[287…

```
"Hey @User if you are done with cocaine and gay orgies, take some similar diplomatic initiatives..\n\nStop being Bajwa's bitch for once and show your balls, if you have any..\n\n#imrankhanPTI #Pakistan #CoronaAlert \n#CoronaOutbreak #CoronavirusPandemic #CoronaVirus #SAARC https://t.co/55n0bN9ZRv"
```

Same tweet but now cleaned up. Notice how the \n tags are now removed, making a much cleaner tweet.

In [288…

```
df['clean_text'][352]
df['ctext_nh'] = df['clean_text']
```

1.4 A point about hashtags

In this part we will not clean up the hashtags because using all the hashtags will be necessary later on. Hashtags will be removed later on in section 2.1.

1.4 Convert all tweets to lowercase for tokenisation

We convert to lowercase so words like "corona" and "Corona" which means in the analysis both words come under the label "corona". This is important because we don't want these to be treated as two different words just because one is capitalised - that would make the statistical analysis weaker.

In [289…

```
df = df.applymap(lambda s:s.lower() if type(s) == str else s) #convert all to lowerca
df['text5'] = df['ctext_nh']
```

1.5 A note about hashtags

Removal of hashtags has been asked in this section however I have decided to leave them in because this pre-processing will later go on to be used in the classifier. These hashtags potentially could help classify whether a tweet has misinformation - perhaps people who give misinformation like to use certain hashtags. Hashtags wil be removed in section 2 to complete those tasks.

1.6 Normalisation

Normalisation is doing further cleaning of the text which means we can explore the data better and build a more rigorous statistical model. We use the package textacy to do this as the normalisation steps are nicely available as commands.

The normalisation steps are the following:

1. Eliminate hyphens in hyphenated words
2. Delete quotation marks
3. Delete unicode
4. Remove accents (such as accented vowels)

First we create a program that does all these steps, and then we apply this program to the corpus of tweets

In [290…

```
from  textacy import preprocessing as tprep
def normalise(text):
    text = tprep.normalize.hyphenated_words(text)
    text = tprep.normalize.quotation_marks(text)
    text = tprep.normalize.unicode(text)
    text = tprep.remove.accents(text)
    return text
```

In [291…

```
df['text6'] = df['text5'].map(normalise)
```

1.7 Tokenisation

We load in the spacy package which used for text analysis. Spacy is fast when it comes to NLP, simple, popular with a lot of support and for pre-processing its commands are simple.

First we tokenise the tweets, which means splitting the tweets up into individual words.

In [292…

```
import spacy
nlp = spacy.load('en_core_web_sm')
```

In [293…

```
df['text7'] = df['text6'].map(nlp)
```

After we apply this tokenisation, we can now see that each tweet below, given by a row, is split into words. Each word is followed by a comma.

In [294…

```
df['text7']
```

Out[294…

```
0       (pags, parody, !, plus, ,, joe, talks, coronav...
1       (#, coronavirus, shows, that, racists, have, a...
2       (@user, @user, i, would, expect, front, line, ...
3       (the, coronavirus, is, not, only, affecting, t...
4       (i, hope, your, father, recovers, from, the, v...
                              ...
3637    (da, fark, is, wrong, with, these, people, ?, ...
3638    (doctor, finds, cure, to, covid-19, (, without...
3639    (i, ca, n't, believe, people, legitimately, th...
3640    (..., (, wrong, with, trump, continues, ), ,, ...
3641    (for, anyone, who, still, believes, that, covi...
Name: text7, Length: 3642, dtype: object
```

1.8 Delete stop words

These words don't help predict or tell us the opinion of a tweet, so we don't need them and they just add noise to the data if we keep them. Here's a program that deletes stop words.

For example, we eliminate words such as "that" or "the". It tells us nothing about the content of the tweet, it's just noise, so we eliminate them.

First we create a program that gives instructions on how to remove stop words, then we apply it to the tweets

In [295…

```
def stop_words(doc):
    non_stop = [t for t in doc if not t.is_stop and not t.is_punct]
    return non_stop
```

Use the program on our tweets:

In [296…

```
df['text8'] = df['text7'].map(stop_words)
```

1.9 Lemmatisation

This process makes words such as "closed", "closing", "closes" all the same, "closes". Tense doesn't tell us anything about the tweet so we make these words the same as they have the same meaning.

This is useful in the analysis because we want to group all types of the word "show", be it "show, shows, showing, showed" into one lemma: "show" because they essentially all have the same meaning.

See how in the following tweet, the 2nd word is "show". The original tweet has "shows" but with lemmatisation this changes to "show".

We will use the tweets of this section in section 4. We will use tweets from section 1.6 for section 2.

In [297…

```
test = [t.lemma_ for t in df['text8'][1]]
test
```

Out[297…

```
['coronavirus',
 'show',
 'racist',
 'absolutely',
 '2',
 'time',
 'hand',
 'firstly',
 'racist',
 'hate',
 'poc',
 'r',
 'stick',
 'xenophobicworld',
 'proud',
 'deplorable',
 'mean',
 'beleev',
 'coronavirus',
 'exist',
 'day',
 '2',
 'ago',
 'trump',
 'quick',
 'study',
 'https://t.co/p22j3ti6kk']
```

Create a program that does the lemmatisation to our tweets then apply it to the data

In [298…

```
def lemma(doc):
    doclems = [t.lemma_ for t in doc]
    return doclems
```

In [299…

```
df['text9'] = df['text8'].map(lemma)
```

---

2. Task B:Text Analysis

In this section all tasks outlined in Task B are carried out. First we will do more pre-processing steps to get the tweets ready to be analysed.

2.1 Pre-processing for the tasks in Task B

Although in sections 1.7 to 1.9 we used spacy to do some pre-processing steps we are going to repeat these steps with textacy as textacy is good to use for doing word type analysis - extracting word types is quite straight forward with this package. We actually use the cleaned tweets from "1.6 Normalisation" for section 2.

Section 4 will use the tweets from section 1.9.

Below we make a program that allows us to extract the word types from a tweet. That means giving each word a label telling us the word type. For example, if the word "eat" is in a tweet, we give it the label "verb". This will be used for the word type analysis outlined in the task.

We use a packages called textacy to do this. We can use this program and tell it to give us adjectives, nouns, verbs and more.

Although we used spacy to lemmatise above, we will use textacy to do this and also to extract other word types.

We will create programs to extract our word types. The word types are as follows:

1. Lemmas (These a words but withot syntax)
2. Noun phrases (two nouns together)
3. Verb phrases (two verbs together)
4. Entities (these are organisations or names, such as Donald Trump or CIA)

We will use the tweets after pre-processing only up to section 1.6 as doing the following processing step also does the following:

1. Eliminates stop words
2. Tokenises
3. Lemmatises

In [300…

```
# import textacy

# tokens = textacy.extract.words(doc,
#             filter_stops = True,           # default True, no stopwords
#             filter_punct = True,           # default True, no punctuation
#             filter_nums = True,            # default False, no numbers
#             include_pos = ['ADJ', 'NOUN'], # default None = include all
#             exclude_pos = None,            # default None = exclude none
#             min_freq = 1)                  # minimum frequency of words
```

2.1a A program to extract lemmas.

In [301…

```
#program to extract lemmas
def extract_lemmas(doc, **kwargs):
    return [t.lemma_ for t in textacy.extract.words(doc, **kwargs)]
```

2.1b A program to extract Noun phrases

The output below the next cell gives an example of a noun phrase.

In [302…

```
def extract_noun_phrases(doc, preceding_pos=['NOUN'], sep='_'):
    patterns = []
    for pos in preceding_pos:
        patterns.append(f"POS:{pos} POS:NOUN:+")
    spans = textacy.extract.matches.token_matches(doc, patterns=patterns)
    return [sep.join([t.lemma_ for t in s]) for s in spans]

print(*extract_noun_phrases(df['text7'][7], ['NOUN']), sep='|')
```

```
health_authority|human_transmission
```

2.1c A program to extract Verb phrases

The output below the next cell gives an example of a verb phrase.

In [303…

```
def extract_verb_phrases(doc, preceding_pos=['VERB'], sep='_'):
    patterns = []
    for pos in preceding_pos:
        patterns.append(f"POS:{pos} POS:VERB:+")
    spans = textacy.extract.matches.token_matches(doc, patterns=patterns)
    return [sep.join([t.lemma_ for t in s]) for s in spans]

print(*extract_verb_phrases(df['text7'][12], ['VERB']), sep='|')
```

```
drink_boil
```

2.1d A program to extract Entities

Entities are things like people, "Donald Trump" or organisations, "CIA".

In [304…

```
def extract_entities(doc, include_types=None, sep='_'):

    ents = textacy.extract.entities(doc,
             include_types=include_types,
             exclude_types=None,
             drop_determiners=True,
             min_freq=1)

    return [sep.join([t.lemma_ for t in e])+'/'+e.label_ for e in ents]
```

2.1e Use all the programs above and put it into one program so you can label all the words with word types just using one piece of code

In [305…

```
def extract_nlp(doc):
    return {
    'lemmas'          : extract_lemmas(doc,
                                    exclude_pos = ['PART', 'PUNCT',
                                       'DET', 'PRON', 'SYM', 'SPACE'],
                                    filter_stops = True),
    'adjs'      : extract_lemmas(doc, include_pos = ['ADJ', 'VERB']),
    'verbs'      : extract_lemmas(doc, include_pos = ['VERB']),
    'nouns'           : extract_lemmas(doc, include_pos = ['NOUN', 'PROPN']),
    'noun_phrases'    : extract_noun_phrases(doc, ['NOUN']),
    'verb_phrases'    : extract_verb_phrases(doc, ['ADV']),
    'adj_noun_phrases': extract_noun_phrases(doc, ['ADJ']),
    'entities'        : extract_entities(doc, ['PERSON', 'ORG', 'GPE', 'LOC']),
    #'hashtags'        : extract_hashtags(text)
    }
```

2.1f Remove hashtags

Create a program and apply it to our data. This removes hashtags because this interferes with our word type analysis.

In [306…

```
def remove_hash(text):
    text = re.sub('\#[\w\_]+',' ', text)
    return text.strip()
```

In [307…

```
df['ctext_nh2'] = df['text6'].map(remove_hash)
```

2.1g Execute the program created in 2.1e to extract word types from each tweet

First we add the columns to the dataframe which will give us a list of the word types that appear in each tweet on a word type basis (looking at the table below will make this clearer).

We then execute the program on our tweets and inspect the resulting table

In [308…

```
nlp_columns = list(extract_nlp(nlp.make_doc('')).keys())
#print(nlp_columns)
for col in nlp_columns:
    df[col] = None
```

In [309…

```
batch_size = 50

for i in range(0, len(df), batch_size):
    docs = nlp.pipe(df['ctext_nh2'][i:i+batch_size])

    for j, doc in enumerate(docs):
        for col, values in extract_nlp(doc).items():
            df[col].iloc[i+j] = values
```

```
C:\Users\T430\Anaconda3\lib\site-packages\pandas\core\indexing.py:1637: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  self._setitem_single_block(indexer, value, name)
```

2.1h The resulting table

Now our dataframe contains our original tweets "text" and the words are split into word types. Scroll to the right of the following table to see how the words of the tweets are now allocated to columns depending on the word type. For example, the first row, the first tweet contains the word "talks". This word is put into the "verbs" column (at the far right of the table).

In [310…

```
df.head()
```

Out[310…

	status_id	text	annontation	clean_text	ctext_nh	text5	text6	text7
0	0	pags parody! plus, joe talks coronavirus and h...	irrelevant	pags parody! plus, joe talks coronavirus and h...	pags parody! plus, joe talks coronavirus and h...	pags parody! plus, joe talks coronavirus and h...	pags parody! plus, joe talks coronavirus and h...	(pags, parody, !, plus, ,, joe, talks, coronav...
1	1	#coronavirus shows that racists have absolutel...	irrelevant	#coronavirus shows that racists have absolutel...	#coronavirus shows that racists have absolutel...	#coronavirus shows that racists have absolutel...	#coronavirus shows that racists have absolutel...	(#, coronavirus, shows, that, racists, have, a...
2	2	@user @user i would expect front line staff to...	irrelevant	@user @user i would expect front line staff to...	@user @user i would expect front line staff to...	@user @user i would expect front line staff to...	@user @user i would expect front line staff to...	(@user, @user, i, would, expect, front, line, ...
3	3	the coronavirus is not only affecting the way ...	irrelevant	the coronavirus is not only affecting the way ...	the coronavirus is not only affecting the way ...	the coronavirus is not only affecting the way ...	the coronavirus is not only affecting the way ...	(the, coronavirus, is, not, only, affecting, t...
4	4	i hope your father recovers from the virus as ...	politics	i hope your father recovers from the virus as ...	i hope your father recovers from the virus as ...	i hope your father recovers from the virus as ...	i hope your father recovers from the virus as ...	(i, hope, your, father, recovers, from, the, v...

2.1i Extract all hashtags from original tweets.

We get the hashtags from the original tweets and make a new column which contains these tags called "hashtags_orig"

We use this column to delete any of the hashtags listed in the assignment to be removed, making it a column called "hashtags_del". This column contains all the hashtags of the tweets except for the tweets listed in the assignment.

In 2.1f we simply deleted the hashtags from the data. This time we want to take them and put them into a column, not actually remove them. Therefore we define a new program that does this.

In [311…

```
import re
def extract_hashtags(text):
    # the regular expression
    regex = "#(\w+)"
    # extracting the hashtags
    hashtag_list = re.findall(regex, text)
    return [t for t in hashtag_list]



#make hashtags column
df['hashtags_orig'] = df['text'].map(extract_hashtags)
#extract hashtags but where the hashtags we don't want are deleted

import re
#store stated hashtags
remove_words = ['ncov20199',
'coronaoutbreak',
'coronavirus',
'coronaviruscoverup',
'coronavirusoutbreak',
'covid19',
'coronavirus',
'wuhancoronavirus',
'coronaviris',
'wuhan']
```

Make a new variable that lists the hashtags but excluding the ones listed in the assignment.

In [312…

```
#this removes the hashtags stated from the hashtag column
df['hashtags_del'] = df.hashtags_orig.apply(lambda x: [i for i in x if i not in remove
```

Here is an example of the new variables created, showing how we have extracted the hashtags.

In [313…

```
df[['text','hashtags_orig', 'hashtags_del']].head()
```

Out[313…

	text	hashtags_orig	hashtags_del
0	pags parody! plus, joe talks coronavirus and h...	[]	[]
1	#coronavirus shows that racists have absolutel...	[coronavirus, xenophobicworld, deplorables, co...	[xenophobicworld, deplorables]
2	@user @user i would expect front line staff to...	[]	[]
3	the coronavirus is not only affecting the way ...	[]	[]
4	i hope your father recovers from the virus as ...	[donthecon, disinfectantdon, hydroxychloroquin...	[donthecon, disinfectantdon, hydroxychloroquin...

2.2 Distribution of word types in corpus and each category

Create a column that tells us the amount of each word type for each tweet, these columns have the suffice "_num". It takes the length of the list in each word type so we get a number of how many times the word type appears in the tweet. See the table and scroll to the far right for it to be clear.

In [314…

```
nlp_columns
for column in nlp_columns:
    df[f'{column}_num'] = [len(x) for x in df[column]]
```

In [315…

```
df.head()
```

Out[315…

	status_id	text	annontation	clean_text	ctext_nh	text5	text6	text7
0	0	pags parody! plus, joe talks coronavirus and h...	irrelevant	pags parody! plus, joe talks coronavirus and h...	pags parody! plus, joe talks coronavirus and h...	pags parody! plus, joe talks coronavirus and h...	pags parody! plus, joe talks coronavirus and h...	(pags, parody, !, plus, ,, joe, talks, coronav...
1	1	#coronavirus shows that racists have absolutel...	irrelevant	#coronavirus shows that racists have absolutel...	#coronavirus shows that racists have absolutel...	#coronavirus shows that racists have absolutel...	#coronavirus shows that racists have absolutel...	(#, coronavirus, shows, that, racists, have, a...
2	2	@user @user i would expect front line staff to...	irrelevant	@user @user i would expect front line staff to...	@user @user i would expect front line staff to...	@user @user i would expect front line staff to...	@user @user i would expect front line staff to...	(@user, @user, i, would, expect, front, line, ...
3	3	the coronavirus is not only affecting the way ...	irrelevant	the coronavirus is not only affecting the way ...	the coronavirus is not only affecting the way ...	the coronavirus is not only affecting the way ...	the coronavirus is not only affecting the way ...	(the, coronavirus, is, not, only, affecting, t...
4	4	i hope your father recovers from the virus as ...	politics	i hope your father recovers from the virus as ...	i hope your father recovers from the virus as ...	i hope your father recovers from the virus as ...	i hope your father recovers from the virus as ...	(i, hope, your, father, recovers, from, the, v...

5 rows × 29 columns

2.1a A chart of the frequency of word types in the entire dataset.

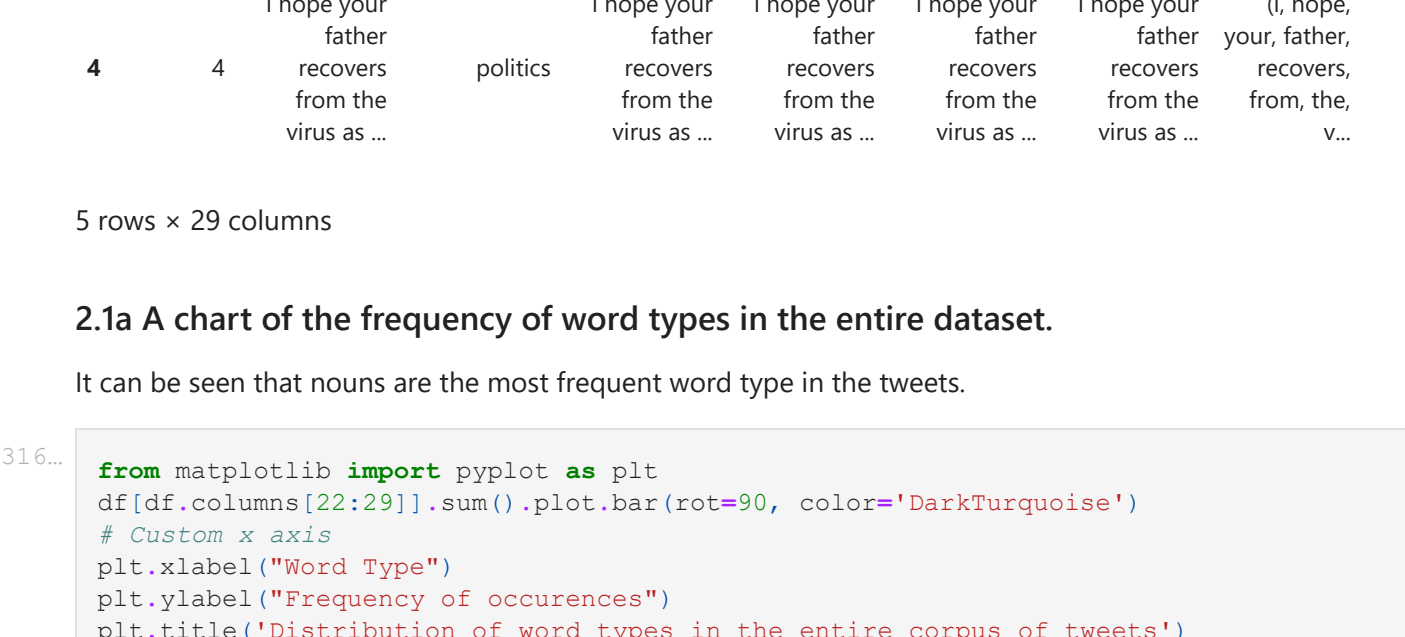
It can be seen that nouns are the most frequent word type in the tweets.

In [316…

```
from matplotlib import pyplot as plt
df[df.columns[22:29]].sum().plot.bar(rot=90, color='DarkTurquoise')
# Custom x axis
plt.xlabel("Word Type")
plt.ylabel("Frequency of occurences")
plt.title('Distribution of word types in the entire corpus of tweets')
```

Out[316…

```
Text(0.5, 1.0, 'Distribution of word types in the entire corpus of tweets')
```



2.1b Distribution of word types in each Annotation

Do this same analysis but for each group (annotation) of tweets, so we sum up the number of word types by the annotations.

The plot below shows the dsitrubtion of word types.

It can be seen the annotation "false public health response" has a high proportion of entity word type as it has a larger red section.

"Calling out or correction" has a big green section meaning it has a lot of adjectives.

In [317…

```
#get dataframe to sum over
import numpy as np
df1 = df[df.columns[np.concatenate([range(2,3),range(22,29)])]]
```

In [318…

```
df2 = df1.groupby('annontation', as_index=False).sum()
```

Here we see the table showing the annotations and how many word types each annotation has

In [319…

```
df2
```

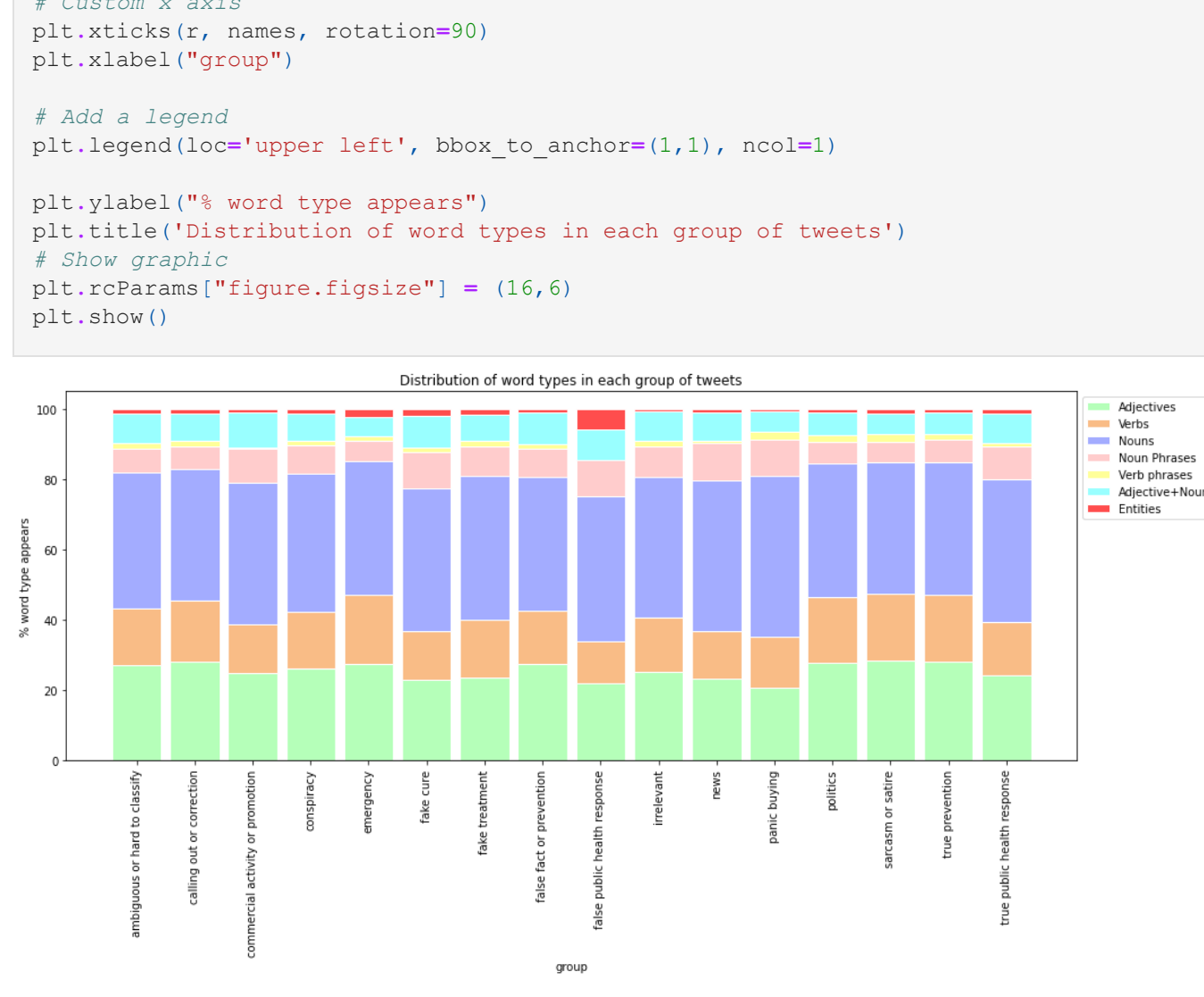
Out[319…

	annotation	adjs_num	verbs_num	nouns_num	noun_phrases_num	verb_phrases_num
0	ambiguous or hard to classify	625	373	894	155	33
1	calling out or correction	6954	4304	9137	1645	398
2	commercial activity or promotion	173	97	279	66	3
3	conspiracy	2665	1663	3999	796	147
4	emergency	125	90	173	25	6
5	fake cure	359	215	629	161	17
6	fake treatment	103	73	179	36	7
7	false fact or prevention	1574	877	2179	454	74
8	false public health response	15	8	28	7	0
9	irrelevant	557	335	873	191	39
10	news	385	227	711	176	12
11	panic buying	260	180	572	131	27
12	politics	2725	1814	3685	609	178
13	sarcasm or satire	2001	1355	2629	411	141
14	true prevention	1047	714	1404	236	66
15	true public health response	819	502	1367	318	27

In [ ]:

In [320…

```
# libraries
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import rc
import pandas as pd

# Data
r = list(df2['annontation'])

# From raw value to percentage
totals = [i+j+k+l+m+n+o for i,j,k,l,m,n,o in zip(df2['adjs_num'], df2['verbs_num'],
                                  df2['nouns_num'], df2['noun_phrases_num'], d

adjs_num = [i / j * 100 for i,j in zip(df2['adjs_num'], totals)]
verbs_num = [i / j * 100 for i,j in zip(df2['verbs_num'], totals)]
nouns_num = [i / j * 100 for i,j in zip(df2['nouns_num'], totals)]
noun_phrases_num = [i / j * 100 for i,j in zip(df2['noun_phrases_num'], totals)]
verb_phrases_num = [i / j * 100 for i,j in zip(df2['verb_phrases_num'], totals)]
adj_noun_phrases_num = [i / j * 100 for i,j in zip(df2['adj_noun_phrases_num'], totals
entities_num = [i / j * 100 for i,j in zip(df2['entities_num'], totals)]

# plot
barWidth = 0.85
names = tuple(list(df2['annontation']))
# Create green Bars
plt.bar(r, adjs_num, color='#b5ffb9', edgecolor='white', width=barWidth, label="Adject
#Create orange Bars
plt.bar(r, verbs_num, bottom=adjs_num, color='#f9bc86', edgecolor='white', width=barWi
# Create blue Bars
plt.bar(r, nouns_num, bottom=[i+j for i,j in zip(adjs_num, verbs_num)], color='#a3acf
plt.bar(r, noun_phrases_num, bottom=[i+j+k for i,j,k in zip(adjs_num, verbs_num, nouns
plt.bar(r, verb_phrases_num, bottom=[i+j+k+l for i,j,k,l in zip(adjs_num, verbs_num, n
plt.bar(r, adj_noun_phrases_num, bottom=[i+j+k+l+m for i,j,k,l,m in zip(adjs_num, verb
plt.bar(r, entities_num, bottom=[i+j+k+l+m+n for i,j,k,l,m,n in zip(adjs_num, verbs_nu


# Custom x axis
plt.xticks(r, names, rotation=90)
plt.xlabel("group")

# Add a legend
plt.legend(loc='upper left', bbox_to_anchor=(1,1), ncol=1)

plt.ylabel("% word type appears")
plt.title('Distribution of word types in each group of tweets')
# Show graphic
plt.rcParams["figure.figsize"] = (16,6)
plt.show()
```



2.3 Similarities

Here we use cosine similarity to compare what annotations are similar.

We convert out tweets to a matrix whhich has for each column a word that appears in the corpus. The numbers in the matrix respond to the frequency of these words.

Using this matrix we can use cosine similarity which gives us how similar vectors are. Since our matrix is just a series of vectors, we can apply this to our matrix and in turn our tweets.

This section 2.3 ends with a heat map which tells us how similar the annotations are. If the colour is more white, that means those two annotations are similar.

In our heatmap, we can see annotation 1 and 13 are similar, with a cosine similarity sore of 0.95. These annotations correspond to "calling out or correction" and "sarcasm".

First we take all the text for each group and put it into one row as we are asked to do this not on an individual tweet basis.

In [321…
```
dfg = df.groupby("annontation")
```

In [322…
```
column_names= ["Annotation", "Text"]
df_groups = pd.DataFrame(columns = column_names)
annots = []
fulltexts = []

for x, y in dfg:
    #print(x)
    #print(y['text'].str.cat(sep=', '))
    annots.append(x)
    fulltexts.append(y['text'].str.cat(sep=', '))
```

In [323…
```
dfg2 = pd.DataFrame({"Annotation": annots, "Text": fulltexts})
```

In [324…
```
dfg2
```

Out[324…

	Annontation	Text
0	ambiguous or hard to classify	are hand dryers effective in killing the new c...
1	calling out or correction	no, lime juice won't immunize you: some people...
2	commercial activity or promotion	visit https://t.co/gqrtz87i0w for new jersey #...
3	conspiracy	do you think the coronavirus is a natural occu...
4	emergency	bfhp is in need of essential supplies due to c...
5	fake cure	"anti-vaxxers have the cure to the coronavirus...
6	fake treatment	i currently have the flu (haven't been tested ...
7	false fact or prevention	if you went to dr. ricciardi in east orange as...
8	false public health response	italy allows malaria and hiv drugs for coronav...
9	irrelevant	pags parody! plus, joe talks coronavirus and h...
10	news	health authorities have identified a new coron...
11	panic buying	as the demand for hand sanitisers soars in the...
12	politics	i hope your father recovers from the virus as ...
13	sarcasm or satire	i protected myself from the coronavirus by wea...
14	true prevention	wash your hands as often as possible #cureforc...
15	true public health response	delhi hc restrains publication of certain adve...

Then we convert the text into a matrix where each word is a column and it gets a value 1 if it appears in the annotation.

In [325…
```
dt2 = CountVectorizer().fit_transform(dfg2['Text'])
```

We use cosine similarity to produce what can be interpreted as akin to a correlation matrix.

In [326…
```
from sklearn.metrics.pairwise import cosine_similarity
simi = pd.DataFrame(cosine_similarity(dt2,dt2))
```

In [327…
```
simi
```

Out[327…

	0	1	2	3	4	5	6	7	8	9	10
0	1.000000	0.951407	0.710882	0.928218	0.762031	0.781331	0.755302	0.888132	0.374929	0.894340	0.764297
1	0.951407	1.000000	0.768467	0.929981	0.811118	0.807141	0.771616	0.914836	0.447666	0.898867	0.809751
2	0.710882	0.768467	1.000000	0.719620	0.725741	0.647337	0.653725	0.824105	0.573720	0.742712	0.811023
3	0.928218	0.929981	0.719620	1.000000	0.757918	0.796624	0.731955	0.858089	0.415083	0.893473	0.834459
4	0.762031	0.811118	0.725741	0.757918	1.000000	0.678575	0.672576	0.814522	0.436440	0.734060	0.733205
5	0.781331	0.807141	0.647337	0.796624	0.678575	1.000000	0.785798	0.747627	0.487822	0.735319	0.722016
6	0.755302	0.771616	0.653725	0.731955	0.672576	0.785798	1.000000	0.744831	0.435436	0.693030	0.693619
7	0.888132	0.914836	0.824105	0.858089	0.814522	0.747627	0.744831	1.000000	0.463483	0.864897	0.804748
8	0.374929	0.447666	0.573720	0.415083	0.436440	0.487822	0.435436	0.463483	1.000000	0.373890	0.574430
9	0.894340	0.898867	0.742712	0.893473	0.734060	0.735319	0.693030	0.864897	0.373890	1.000000	0.813790
10	0.764297	0.809751	0.811023	0.834459	0.733205	0.722016	0.693619	0.804748	0.574430	0.813790	1.000000
11	0.821555	0.855873	0.781266	0.854849	0.733672	0.716538	0.682270	0.837533	0.450327	0.897840	0.862492
12	0.894777	0.918822	0.739685	0.901658	0.778550	0.788127	0.737589	0.854271	0.408884	0.878635	0.800070
13	0.935070	0.953431	0.717877	0.914212	0.789106	0.786461	0.725426	0.890421	0.376851	0.913416	0.762752
14	0.845954	0.885512	0.832002	0.817155	0.819130	0.689336	0.684972	0.905747	0.442588	0.832903	0.769123
15	0.819947	0.867512	0.838570	0.874116	0.813748	0.787059	0.741265	0.871630	0.549838	0.858732	0.920379

We display this in a heat map, with lighter colours representing a higher similarity. For example, 13 and 1 have a value of 0.95. These groups of tweets are very similar. What groups are they?

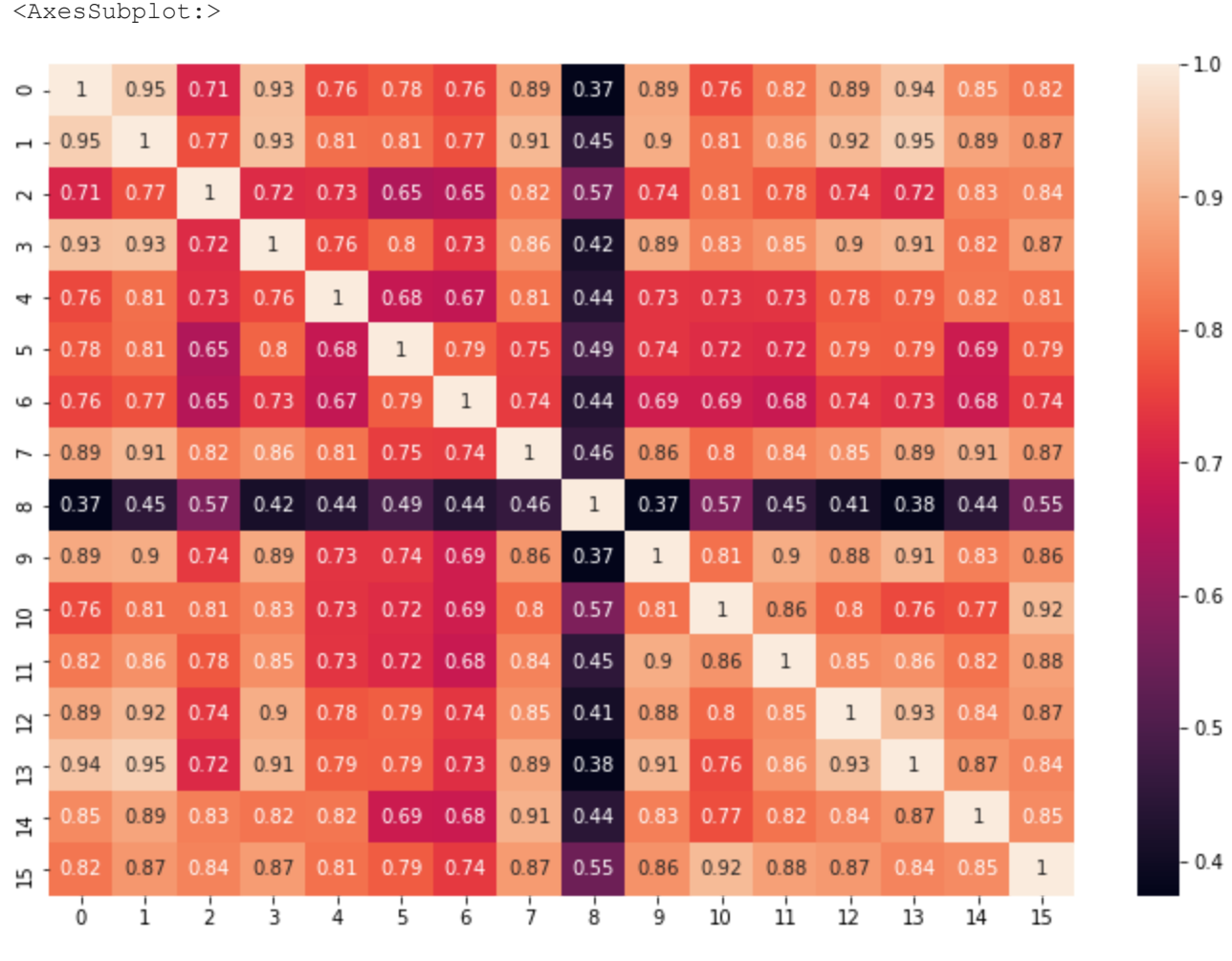
In [328…
```
annots[1], annots[13]
```

Out[328…
```
('calling out or correction', 'sarcasm or satire')
```

So calling out/correction and sarcasm tweets are similar.

In [329…
```
#plt.xticks(r, names, rotation=90)
#plt.xlabel("group")
plt.figure(figsize=(12, 8))
sns.heatmap(simi, annot=True)
```

Out[329…
```
<AxesSubplot:>
```



Use the following as a key to link the numbers with annotations

In [330…
```
annots
```

Out[330…
```
['ambiguous or hard to classify',
 'calling out or correction',
 'commercial activity or promotion',
 'conspiracy',
 'emergency',
 'fake cure',
 'fake treatment',
 'false fact or prevention',
 'false public health response',
 'irrelevant',
 'news',
 'panic buying',
 'politics',
 'sarcasm or satire',
 'true prevention',
 'true public health response']
```

2.4 What is the over all distribution of nouns and noun phrases, and verbs and verb phrases, in the entire corpus?

Using the tables below it can be seen that the amount of nouns is 28,730 and nouns are the most common word type. We display these word types graphically as a percentage below.

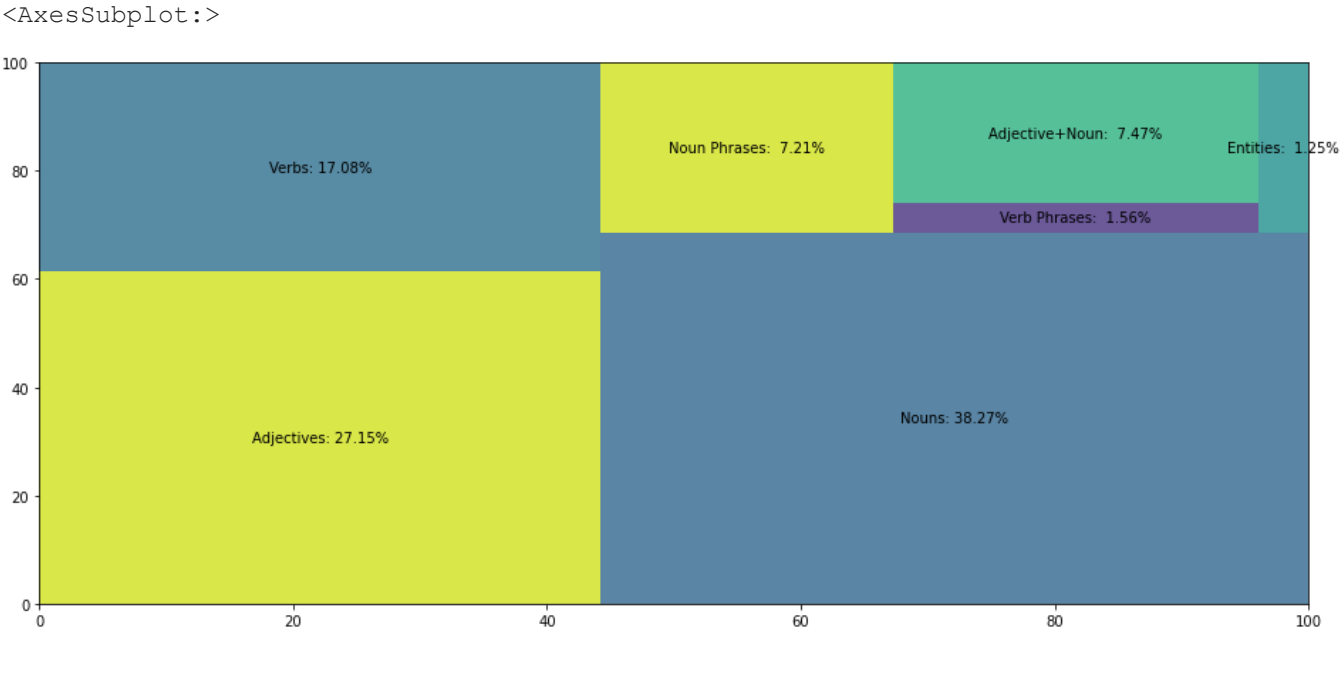
We sum up the number of each word type and display as a percentage of the entire dataset

In [331…
```
dfb3 = df1.sum()
dfb3[1:8]
```

Out[331…
```
adjs_num                20387
verbs_num               12827
nouns_num               28738
noun_phrases_num         5417
verb_phrases_num         1175
adj_noun_phrases_num     5606
entities_num              938
dtype: object
```

In [332…
```
#!pip install squarify
import squarify
label= ["Adjectives", "Verbs", "Nouns", "Noun Phrases", "Verb Phrases", "Adjective+Nou

perc = [f'{i/dfb3[1:8].sum()*100:5.2f}%' for i in dfb3[1:8]]
lbl = [f'{el[0]}: {el[1]}' for el in zip(label, perc)]
plt.figure(figsize=(16, 7))
squarify.plot(sizes=dfb3[1:8], label=lbl, alpha=.8 )
#squarify.plot(sizes=dfb3[1:6], label =["Adjectives", "Verbs", "Nouns", "Noun Phrases
#plt.axis('off')
#plt.show()
```

Out[332…
```
<AxesSubplot:>
```



2.5 Which annotated category has the highest number of user mentions?

We count the number of mentions per tweet then aggregate up for each group of tweets to see which has the most mentions. We then use the number of mentions and divide it by the amoount of tweets to see the percentage amount the tweets in each annotation contain a mention. If an annotation has over 100% that means on average the tweets in that annotation have more than 1 user mention per tweet.

From the table below, fake treatment has the most mentions per tweet with ~105%.

In [333…
```
df['numuser'] = df.text.str.count("@user")
```

In [334…
```
dfb4 = df[df.columns[np.concatenate([range(2,3),range(29,30)])]]
```

Sum up the user mentions in each annotation

In [335…
```
dfb4a = dfb4.groupby('annontation', as_index=False).sum()
dfb4a
```

Out[335…

	annontation	numuser
0	ambiguous or hard to classify	38
1	calling out or correction	381
2	commercial activity or promotion	6
3	conspiracy	166
4	emergency	8
5	fake cure	71
6	fake treatment	22
7	false fact or prevention	51
8	false public health response	0
9	irrelevant	22
10	news	31
11	panic buying	17
12	politics	190
13	sarcasm or satire	78
14	true prevention	32
15	true public health response	53

Convert this to a percentage of the amount of tweets in each annotation

In [336…
```
totals = dfb4.groupby('annontation', as_index=False).agg('count')
dfb4a['percent'] = (dfb4a['numuser']/totals['numuser'])*100
```

From the table below, fake treatment has the most mentions per tweet with ~105%.

In [337…
```
dfb4a # one entry can have more than mention, hence fake treatment has more mentions
```

Out[337…

	annontation	numuser	percent
0	ambiguous or hard to classify	38	33.628319
1	calling out or correction	381	31.936295
2	commercial activity or promotion	6	20.689655
3	conspiracy	166	32.549020
4	emergency	8	50.000000
5	fake cure	71	77.173913
6	fake treatment	22	104.761905
7	false fact or prevention	51	20.158103
8	false public health response	0	0.000000
9	irrelevant	22	20.183486
10	news	31	33.695652
11	panic buying	17	26.153846
12	politics	190	43.378995
13	sarcasm or satire	78	19.647355
14	true prevention	32	20.000000
15	true public health response	53	35.099338

2.5 Which annotated category uses the most hashtags in a tweet on average?

We use the same technique above but with hashtags. We can see Commercial activity has the most hash tags per tweet at ~241% (that's 2.4 hashtags per tweet)

In [338…
```
df['hash_count'] = df['hashtags_orig'].str.len()
```

In [339…
```
dfb5 = df[df.columns[np.concatenate([range(2,3),range(30,31)])]]
dfb5
dfb5a = dfb5.groupby('annontation', as_index=False).sum()
totals = dfb5.groupby('annontation', as_index=False).agg('count')
dfb5a['totals'] = totals['hash_count']
dfb5a['percent'] = (dfb5a['hash_count']/totals['hash_count'])*100
```

Table of percentage of hashtag in each annotation.

In [340…
```
dfb5a
```

Out[340…

	annontation	hash_count	totals	percent
0	ambiguous or hard to classify	116	113	102.654867
1	calling out or correction	1017	1193	85.247276
2	commercial activity or promotion	70	29	241.379310
3	conspiracy	739	510	144.901961
4	emergency	16	16	100.000000
5	fake cure	109	92	118.478261
6	fake treatment	16	21	76.190476
7	false fact or prevention	365	253	144.268775
8	false public health response	0	3	0.000000
9	irrelevant	184	109	168.807339
10	news	72	92	78.260870
11	panic buying	100	65	153.846154
12	politics	534	438	121.917808
13	sarcasm or satire	238	397	59.949622
14	true prevention	210	160	131.250000
15	true public health response	183	151	121.192053

2.6 What are the top 3 hashtags for each category, including and excluding the hashtags used to filter the tweets?

We print out tables below showing the top hashtags for each category including and excluding the hashtags. For example, for the annotation "fake cure" the most used hashtag was "coronil"

Create a program that counts the frequency of word appearance.

In [341…
```
from collections import Counter
def count_words(df, column='tokens', preprocess=None, min_freq=2):

    # process tokens and update counter
    def update(doc):
        tokens = doc if preprocess is None else preprocess(doc)
        counter.update(tokens)

    # create counter and run through all data
    counter = Counter()
    df[column].map(update)

    # transform counter into a DataFrame
    freq_df = pd.DataFrame.from_dict(counter, orient='index', columns=['freq'])
    freq_df = freq_df.query('freq >= @min_freq')
    freq_df.index.name = 'token'

    return freq_df.sort_values('freq', ascending=False)
```

In [342…
```
hash_freq_tab =count_words(df, 'hashtags_orig').head(3)
#count_words(df, 'noun_p').head(10).plot(kind='barh').invert_yaxis()
```

Top 3 hashtags per category **including** all tweets

In [343…
```
for x in dfb5a['annontation']:
    print(x)
    print(count_words(df.loc[df['annontation'] == x], 'hashtags_orig').head(3))
    print('\n')
```

```
ambiguous or hard to classify
             freq
token
covid19        17
coronavirus    14
bioweapon       4


calling out or correction
                  freq
token
coronavirus        179
covid19            167
coronavirusfacts    23


commercial activity or promotion
             freq
token
covid19         9
coronavirus     8
covidscams      2


conspiracy
             freq
token
coronavirus   111
covid19        63
bioweapon      62


emergency
                         freq
token
coronavirus                 4
covid19                     2
donateplasmarescuealife     2


fake cure
             freq
token
coronil         8
covid19         8
coronavirus     6


fake treatment
                    freq
token
hydroxychloroquine     2
coronavirus            2


false fact or prevention
             freq
token
coronavirus    39
covid19        35
covid_19       18


false public health response
Empty DataFrame
Columns: [freq]
Index: []


irrelevant
             freq
token
coronavirus    29
covid19        21
cooking         4


news
             freq
token
coronavirus    18
covid19         8
covid           3


panic buying
             freq
token
coronavirus    20
covid19         7
covid-19        4


politics
             freq
token
coronavirus    60
covid19        54
trump          37


sarcasm or satire
                  freq
token
coronavirus         38
covid19             34
socialdistancing     6


true prevention
             freq
token
covid19        48
coronavirus    26
covid_19        6


true public health response
                    freq
token
covid19               28
coronavirus           23
hydroxychloroquine     6
```

Top 3 hashtags per category **excluding** tweets quoted in the assignment

In [344…
```
for x in dfb5a['annontation']:
    print(x)
    print(count_words(df.loc[df['annontation'] == x], 'hashtags_del').head(3))
    print('\n')
```

```
ambiguous or hard to classify
           freq
token
bioweapon     4
billgates     3
covid         3


calling out or correction
                  freq
token
coronavirusfacts    23
lie                 23
covidiots           22


commercial activity or promotion
              freq
token
covidscams       2
disinfectant     2
cleaning         2


conspiracy
           freq
token
bioweapon    62
china        26
5g           14


emergency
                         freq
token
donateplasmarescuealife     2


fake cure
                    freq
token
coronil               8
covid                 6
hydroxychloroquine    6


fake treatment
                    freq
token
hydroxychloroquine     2


false fact or prevention
                    freq
token
covid_19              18
hydroxychloroquine     7
health                 6


false public health response
Empty DataFrame
Columns: [freq]
Index: []


irrelevant
                freq
token
cooking            4
quarantinelife     4
covid-19           3


news
             freq
token
covid           3
saudiarabia     2
icis            2


panic buying
             freq
token
covid-19        4
toiletpaper     3
covid2019       3


politics
             freq
token
trump          37
trumpvirus     12
donaldtrump     8


sarcasm or satire
                  freq
token
socialdistancing     6
trump                6
covid_19             4


true prevention
              freq
token
covid_19         6
disinfectant     4
staysafe         4


true public health response
                    freq
token
hydroxychloroquine     6
covidscams             4
who                    4
```

3. Task C - Affect Analysis

In this analysis we use a dictionary known as Vader, which is a popular dictionary to use with twitter data. We link the words in our tweets to that dictionary and it then gives each word a positive or negative score. The word "good" in a tweet for example would get a score +1, and the word "horrible" might get a score of -2. Using this method, we can see for each annotation how positive/ negative it is.

At the end of this section there is a plot which shows how positive or negative each annotation is. Scores above 0 are deemed positive with a maximum positive score of 1. Scores below 0 are deemed negative with a minimum negative score of -1.

In [345…
```
from nltk.sentiment.vader import SentimentIntensityAnalyzer

sid = SentimentIntensityAnalyzer()
```

In [346…
```
dfg2['Text'] ####here is the column we'll do the analysis on
```

Out[346…
```
0     are hand dryers effective in killing the new c...
1     no, lime juice won't immunize you: some people...
2     visit https://t.co/gqrtz87i0w for new jersey #...
3     do you think the coronavirus is a natural occu...
4     bfhp is in need of essential supplies due to c...
5     "anti-vaxxers have the cure to the coronavirus...
6     i currently have the flu (haven't been tested ...
7     if you went to dr. ricciardi in east orange as...
8     italy allows malaria and hiv drugs for coronav...
9     pags parody! plus, joe talks coronavirus and h...
10    health authorities have identified a new coron...
11    as the demand for hand sanitisers soars in the...
12    i hope your father recovers from the virus as ...
13    i protected myself from the coronavirus by wea...
14    wash your hands as often as possible #cureforc...
15    delhi hc restrains publication of certain adve...
Name: Text, dtype: object
```

This prepares a dataset with the group and all the text within the group.

In [347…
```
column_names= ["Annontation", "Text"]
df_groups = pd.DataFrame(columns = column_names)
annots = []
fulltexts = []

for x, y in dfg:
    #print(x)
    #print(y['text'].str.cat(sep=', '))
    annots.append(x)
    fulltexts.append(y['text5'].str.cat(sep=', '))

dfg2 = pd.DataFrame({"Annontation": annots, "Text": fulltexts})
```

We now use the vader dictionary on the text. This matches up words in the dictionary with the words in the tweets and gives it a score based on those words. For example, if the text contains the word "good" then it will receive +1, if it contains the word "bad" it will receive -1 points. If it contains "very bad" it will get -2, for example. Depending on how positive/negative the word/phrase it, it gets more/less points. From this we can give each body of a text a score for positive or negative to see if each groups' tweets are positive or negative.

In [348…
```
dfg2['scores'] = dfg2['Text'].apply(lambda review: sid.polarity_scores(review))
dfg2
```

Out[348…

	Annontation	Text	scores
0	ambiguous or hard to classify	are hand dryers effective in killing the new c...	{'neg': 0.129, 'neu': 0.754, 'pos': 0.117, 'co...
1	calling out or correction	no, lime juice won't immunize you: some people...	{'neg': 0.178, 'neu': 0.719, 'pos': 0.103, 'co...
2	commercial activity or promotion	visit https://t.co/gqrtz87i0w for new jersey #...	{'neg': 0.068, 'neu': 0.799, 'pos': 0.133, 'co...
3	conspiracy	do you think the coronavirus is a natural occu...	{'neg': 0.14, 'neu': 0.766, 'pos': 0.094, 'com...
4	emergency	bfhp is in need of essential supplies due to c...	{'neg': 0.084, 'neu': 0.79, 'pos': 0.126, 'com...
5	fake cure	"anti-vaxxers have the cure to the coronavirus...	{'neg': 0.075, 'neu': 0.836, 'pos': 0.09, 'com...
6	fake treatment	i currently have the flu (haven't been tested ...	{'neg': 0.025, 'neu': 0.808, 'pos': 0.167, 'co...
7	false fact or prevention	if you went to dr. ricciardi in east orange as...	{'neg': 0.087, 'neu': 0.716, 'pos': 0.197, 'co...
8	false public health response	italy allows malaria and hiv drugs for coronav...	{'neg': 0.0, 'neu': 0.947, 'pos': 0.053, 'comp...
9	irrelevant	pags parody! plus, joe talks coronavirus and h...	{'neg': 0.108, 'neu': 0.781, 'pos': 0.112, 'co...
10	news	health authorities have identified a new coron...	{'neg': 0.118, 'neu': 0.813, 'pos': 0.069, 'co...
11	panic buying	as the demand for hand sanitisers soars in the...	{'neg': 0.114, 'neu': 0.821, 'pos': 0.065, 'co...
12	politics	i hope your father recovers from the virus as ...	{'neg': 0.156, 'neu': 0.731, 'pos': 0.112, 'co...
13	sarcasm or satire	i protected myself from the coronavirus by wea...	{'neg': 0.112, 'neu': 0.752, 'pos': 0.136, 'co...
14	true prevention	wash your hands as often as possible #cureforc...	{'neg': 0.06, 'neu': 0.758, 'pos': 0.182, 'com...
15	true public health response	delhi hc restrains publication of certain adve...	{'neg': 0.087, 'neu': 0.798, 'pos': 0.116, 'co...

Can see from this table if a category is negative or positive. Positive are any scored >0, negative are any scored <0. "comp_score" column tells us if the category is positive or negative.

In [349…
```
dfg2['compound']  = dfg2['scores'].apply(lambda score_dict: score_dict['compound'])
dfg2['comp_score'] = dfg2['compound'].apply(lambda c: 'pos' if c >=0 else 'neg')
dfg2
```

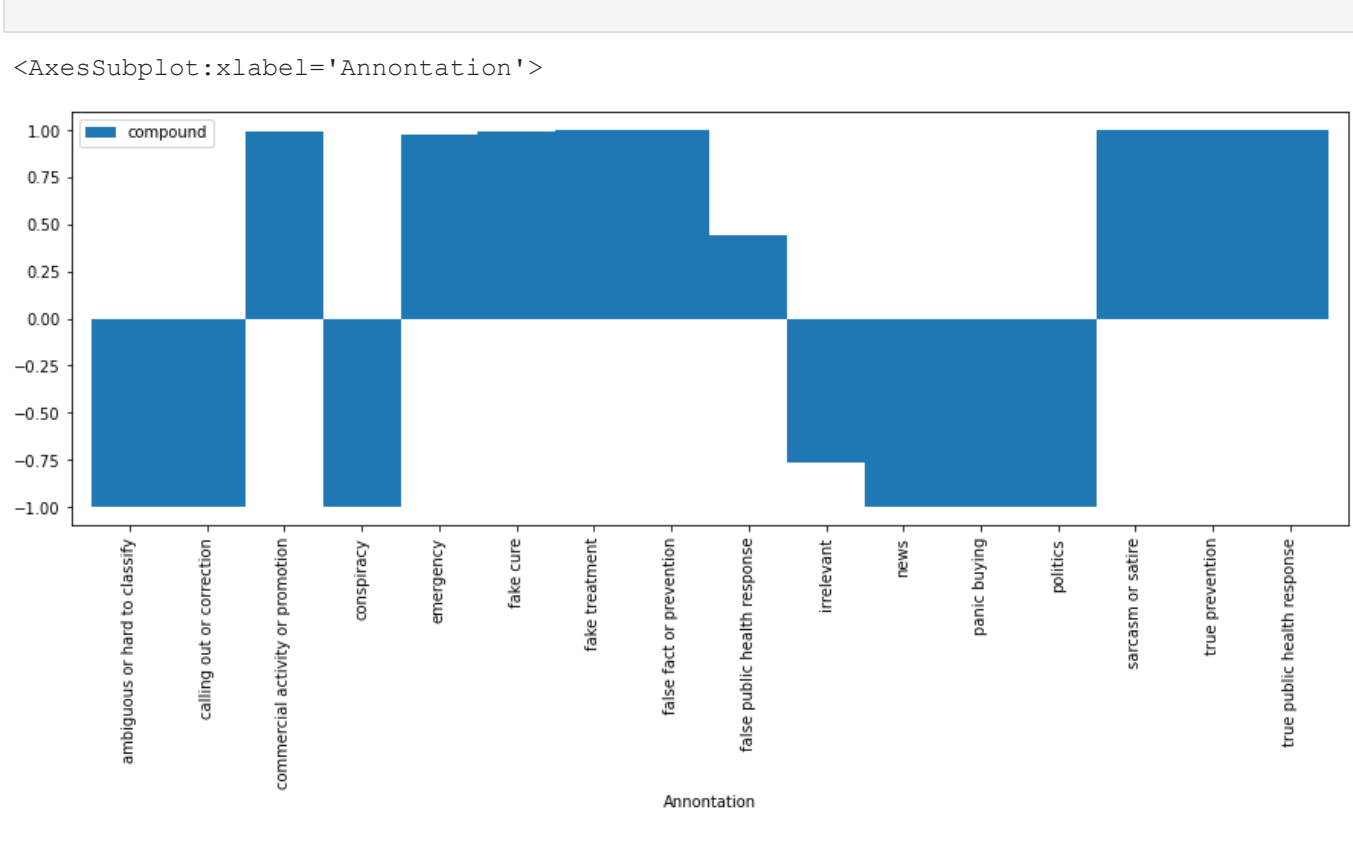
Out[349…

	Annontation	Text	scores	compound	comp_score
0	ambiguous or hard to classify	are hand dryers effective in killing the new c...	{'neg': 0.129, 'neu': 0.754, 'pos': 0.117, 'co...	-0.9973	neg
1	calling out or correction	no, lime juice won't immunize you: some people...	{'neg': 0.178, 'neu': 0.719, 'pos': 0.103, 'co...	-1.0000	neg
2	commercial activity or promotion	visit https://t.co/gqrtz87i0w for new jersey #...	{'neg': 0.068, 'neu': 0.799, 'pos': 0.133, 'co...	0.9941	pos
3	conspiracy	do you think the coronavirus is a natural occu...	{'neg': 0.14, 'neu': 0.766, 'pos': 0.094, 'com...	-1.0000	neg
4	emergency	bfhp is in need of essential supplies due to c...	{'neg': 0.084, 'neu': 0.79, 'pos': 0.126, 'com...	0.9764	pos
5	fake cure	"anti-vaxxers have the cure to the coronavirus...	{'neg': 0.075, 'neu': 0.836, 'pos': 0.09, 'com...	0.9915	pos
6	fake treatment	i currently have the flu (haven't been tested ...	{'neg': 0.025, 'neu': 0.808, 'pos': 0.167, 'co...	0.9981	pos
7	false fact or prevention	if you went to dr. ricciardi in east orange as...	{'neg': 0.087, 'neu': 0.716, 'pos': 0.197, 'co...	1.0000	pos
8	false public health response	italy allows malaria and hiv drugs for coronav...	{'neg': 0.0, 'neu': 0.947, 'pos': 0.053, 'comp...	0.4404	pos
9	irrelevant	pags parody! plus, joe talks coronavirus and h...	{'neg': 0.108, 'neu': 0.781, 'pos': 0.112, 'co...	-0.7671	neg
10	news	health authorities have identified a new coron...	{'neg': 0.118, 'neu': 0.813, 'pos': 0.069, 'co...	-0.9989	neg
11	panic buying	as the demand for hand sanitisers soars in the...	{'neg': 0.114, 'neu': 0.821, 'pos': 0.065, 'co...	-0.9970	neg
12	politics	i hope your father recovers from the virus as ...	{'neg': 0.156, 'neu': 0.731, 'pos': 0.112, 'co...	-1.0000	neg
13	sarcasm or satire	i protected myself from the coronavirus by wea...	{'neg': 0.112, 'neu': 0.752, 'pos': 0.136, 'co...	0.9998	pos
14	true prevention	wash your hands as often as possible #cureforc...	{'neg': 0.06, 'neu': 0.758, 'pos': 0.182, 'com...	1.0000	pos
15	true public health response	delhi hc restrains publication of certain adve...	{'neg': 0.087, 'neu': 0.798, 'pos': 0.116, 'co...	0.9992	pos

We plot this on a bar chart. If an annotation is positive it goes above the 0. If it's negative it goes below.

In [350…
```
# Custom x axis
dfg2.plot.bar(figsize=(15,5),width=1, x='Annontation')
```

Out[350…
```
<AxesSubplot:xlabel='Annontation'>
```



4. Task D: Text Classification

We want to create a binary classifier of "misinformation" and "no misinformation".

First we combined tweets using annotations. We combine these tweets to form just 2 groups, misinformation and information. The following describes how the annotations were grouped.

Misinformation

Annotations:

- Conspiracy

- Fake Cure
- Fake Treatment
- False Fact or Prevention

Information

Annotations:

- True treatment
- True prevention
- True public health response
- Calling out or correction
- Emergency
- Irrelevant
- Commercial activity or promotion
- Panic buying
- Politics

We exclude "sarcasm" from this analysis. Sarcasm could confuse the classifier as the classifier probably won't be able to pick up on a sarcastic tone and may interpret sarcasm as misinformation

We use these tweets to split into test and train.

The train tweets will be used to build a Support Vector Machine model (SVM models are good for sparse data which we see in text data, that is matrices with many 0s).

We will then use the test tweets to see how well the model can correctly label these tweets as information or mis-information.

Since we know these tweets are classed as information/mis-information, we can cross check what the model predicts with the reality.

In this case, we build a model with 82.4% accuracy.

We also carried out a grid search later on in this section. This takes many parameters that can be inputted into the model to see which gives the best accuracy score. Using this, we used the best parameters in our model.

In [383…

```
## merge tweets
df

misinfo = ["conspiracy",
           "fake cure", "fake treatment", "false fact or prevention"]
no_misinfo = ["true treatment", "true prevention", "true public health response", "ca
              "irrelevant", "commercial activity or promotion", "panic buying", "polit
```

Assign each tweet a misinfo or no misinfo label

In [384…

```
# create a list of our conditions
conditions = [
    (df["annontation"].isin(misinfo)),
    (df["annontation"].isin(no_misinfo)),
    (~df["annontation"].isin(misinfo)) & (~df["annontation"].isin(no_misinfo))
    ]

# create a list of the values we want to assign for each condition
values = ['misinfo', 'no_misinfo', 'n/a']

# create a new column and use np.select to assign values to it using our lists as argu
df['class1'] = np.select(conditions, values)
```

A table showing an annotation and how it's classified (class1). Irrelevant is classified as no_misinfo which we want.

In [385…

```
with pd.option_context('display.max_rows', None, 'display.max_columns', None):
    print(df[['annontation', 'class1']].head())
```

```
  annontation      class1
0  irrelevant  no_misinfo
1  irrelevant  no_misinfo
2  irrelevant  no_misinfo
3  irrelevant  no_misinfo
4    politics  no_misinfo
```

Drop the tweets we don't deem relevant to the analysis.

In [386…

```
#drop n/a
dfd1 = df[df.class1 != 'n/a']
```

The column 'text9' is after our pre-processing steps. We change it to a list to a string so it can be used in the classifier.

In [387…

```
#hello = ' '.join(dfd1['text9'])
#dfd1['text9']
dfd1['text10'] = [' ' .join(x) for x in dfd1['text9']]
```

```
<ipython-input-387-59c8dd610ef5>:3: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/use
r_guide/indexing.html#returning-a-view-versus-a-copy
  dfd1['text10'] = [' ' .join(x) for x in dfd1['text9']]
```

In [388…

```
#dfd1['text10'][352] #= dfd1['text10'].map(remove_hash)
df['text8'][352]
```

Out[388…

```
[hey,
 @user,
 cocaine,
 gay,
 orgies,
 similar,
 diplomatic,
 initiatives,
 stop,
 bajwa,
 bitch,
 balls,
 imrankhanpti,
 pakistan,
 coronaalert,
 coronaoutbreak,
 coronaviruspandemic,
 coronavirus,
 saarc,
 https://t.co/55n0bn9zrv]
```

Split the data into train and test. Train data is used to create the model. We then use the test data to see if the model is accurate on data we didn't use in the model to see if it can make predictions on new data.

In [389…

```
#use text6
X_train, X_test, Y_train, Y_test = train_test_split(dfd1['text10'],
                                                    dfd1['class1'],
                                                    test_size=0.2,
                                                    random_state=42,
                                                    stratify=dfd1['class1']
                                                   )

print('Size of Training Data ', X_train.shape[0])
print('Size of Test Data ', X_test.shape[0])
```

```
Size of Training Data  2503
Size of Test Data  626
```

Transform the test data into a matrix of numbers, where each word gets a number of how frequently it appears in each class of info and misinfo. The grid search below, which tells us the best parameters to use for this model, tells us that an ngram range of 1,3 is the best. That means we include phrases of 1, 2 and 3 words long in the analysis. For example for a 3 gram phrase, "coronavirus is bad" would be included. For a 2 gram phrase "go away" would be included.

In [390…

```
from sklearn.feature_extraction.text import TfidfVectorizer
tfidf = TfidfVectorizer(min_df = 10, ngram_range=(1,3), stop_words="english")
X_train_tf = tfidf.fit_transform(X_train)
```

We use a Support Vector Machine to classify as this is better with text data as it's sparse data (the matrix has a lot of 0s in, hence it is sparse).

In [391…

```
#from sklearn.svm.LinearSVC import LinearSVC
from sklearn.svm import LinearSVC
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.datasets import make_classification
model1 = LinearSVC(random_state=0, tol=1e-5)
model1.fit(X_train_tf, Y_train)
```

Out[391…

```
LinearSVC(random_state=0, tol=1e-05)
```

In [392…

```
LinearSVC(C=1.0, class_weight=None, dual=True, fit_intercept=True,
          intercept_scaling=1, loss='squared_hinge', max_iter=10000,
          multi_class='ovr', penalty='l2', random_state=0, tol=1e-05,
          verbose=0)
```

Out[392…

```
LinearSVC(max_iter=10000, random_state=0, tol=1e-05)
```

We test the model on our test data to see how accurate it is. For each tweet we let the model predict if it's information or misinformation and we compare it to the actual label. What percentage of the time is it correct, the accuracy score tells us that, 82.4%

In [393…

```
X_test_tf = tfidf.transform(X_test)

Y_pred = model1.predict(X_test_tf)
print ('Accuracy Score - ', accuracy_score(Y_test, Y_pred))
```

```
Accuracy Score -  0.8242811501597445
```

This is a confusion matrix which tells us how well the model performs.

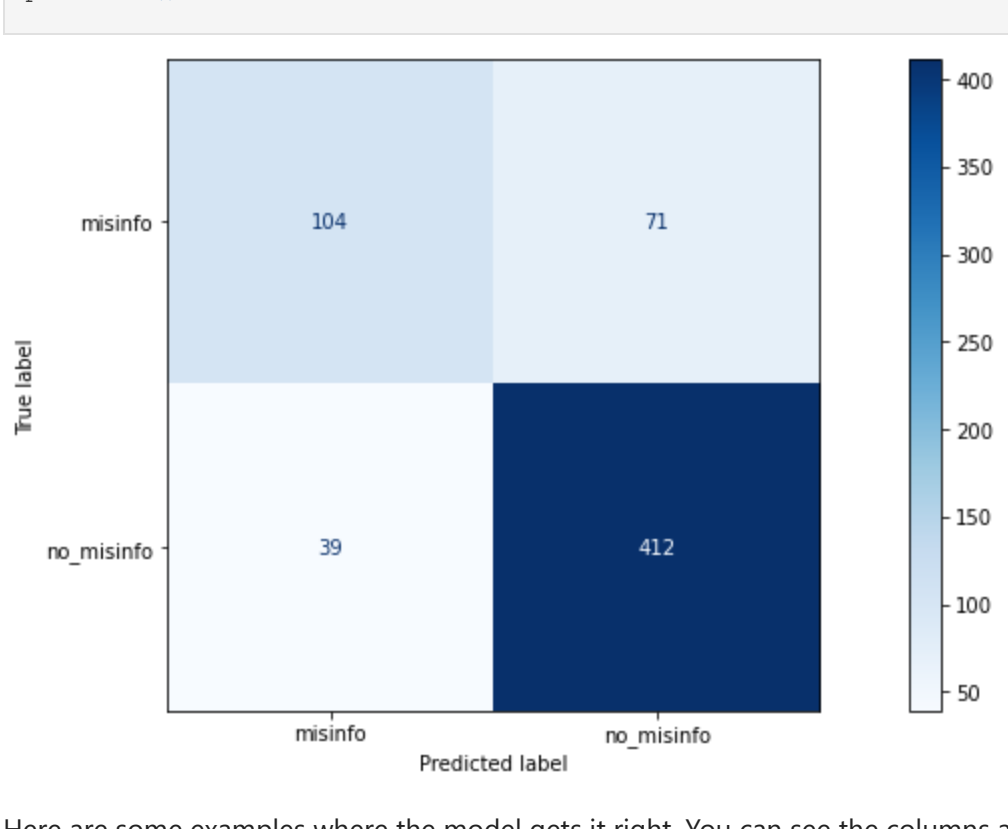
For the test data, and for the misinformation tweets, the model gets it right 104 times, but gets it wrong 71 times. That's not so good. That means given the tweet has misinformation, it correctly identifies it as misinformation 59% of the time.

For the no_information, it gets it wrong 39 times but gets it right 412 times. That's good. That means if a tweet comes in that has no information, we can correctly determine using the model with a rate of 91% that it has no information.

So it's good at detecting tweets with no misinformation but not as good det picking up tweets with misinformation.

In [394…

```
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(model1,X_test_tf,
                      Y_test, values_format='d',
                      cmap=plt.cm.Blues)
plt.show()
```



Here are some examples where the model gets it right. You can see the columns actual and predicted match up.

In [395…

```
# Create a DataFrame combining the Title and Description,
# Actual and Predicted values that we can explore
frame = { 'text': X_test, 'actual': Y_test, 'predicted': Y_pred }
result = pd.DataFrame(frame)


result[((result['actual'] == 'no_misinfo') | (result['actual'] == 'misinfo')) &(result
```

Out[395…

	text	actual	predicted
115	bacterial horror hot air hand dryers https://t...	no_misinfo	no_misinfo
607	alcohol disinfectant chlorine effective clean ...	no_misinfo	no_misinfo
2149	relative maga actually covid 19 fake news trum...	no_misinfo	no_misinfo
1338	day trump give suggestion case people intoxica...	no_misinfo	no_misinfo

Here are some examples where the model gets it wrong. The table shows the actual label, which is what the tweet actually is, and the predicted label, what the model predicts the tweet to be.

In [396…

```
result[((result['actual'] == 'misinfo')) &(result['actual'] != result['predicted'])].s
```

Out[396…

	text	actual	predicted
1249	@user see precious tweet family corona viruse ...	misinfo	no_misinfo
3605	look recent study claim people hospitalize w c...	misinfo	no_misinfo
2159	comment researcher find cheap widely available...	misinfo	no_misinfo
2958	big concern pay attention fact covid 19 airbor...	misinfo	no_misinfo

Here is an example tweet that the classifier gets wrong

In [399…

```
dfd1['text'][3605]
```

Out[399…

```
'i looked at the recent study that claims that people hospitalized w/covid-19 did bett
er on hydroxy than without.\n\n1. people given hydroxy were also more likely to be giv
en steroids (know to be helpful). 79% vs 36%\n2. people given hydroxy were on average
5 years younger.\n\n*1/2* https://t.co/6957bszlm8'
```

4.1 Thoughts

Although the model performs reasonably well, how it classifies tweets with misinformation isn't so accurate. Including "conspiracy" annotation is debatable in the misinformation, since it doesn't really give advice on how to deal with covid. The model was run without the conspiracy annotation and unfortunately the number of misinformation tweets was too low to build a stable model. If this analysis were to be run again more tweets about misinformation would make a big improvement.

4.2 Appendix

This is a grid search which tells us the best parameters to use. This is commented out because it only needs to be run once and the parameters from this and used in the SVM model creation in section 4.

In [398…

```
# from sklearn.pipeline import Pipeline
# from sklearn.model_selection import GridSearchCV
# training_pipeline = Pipeline(
#     steps=[('tfidf', TfidfVectorizer(stop_words="english")),
#            ('model', LinearSVC(random_state=42, tol=1e-5))])

# grid_param = [{
#     'tfidf__min_df': [5, 10],
#     'tfidf__ngram_range': [(1, 3), (1, 6)],
#     'model__penalty': ['l2'],
#     'model__loss': ['hinge'],
#     'model__max_iter': [10000]
# }, {
#     'tfidf__min_df': [5, 10],
#     'tfidf__ngram_range': [(1, 3), (1, 6)],
#     'model__C': [1, 10],
#     'model__tol': [1e-2, 1e-3]
# }]

# gridSearchProcessor = GridSearchCV(estimator=training_pipeline,
#                                    param_grid=grid_param,
#                                    cv=5)
# gridSearchProcessor.fit(dfd1['text10'], dfd1['class1'])

# best_params = gridSearchProcessor.best_params_
# print("Best alpha parameter identified by grid search ", best_params)

# best_result = gridSearchProcessor.best_score_
# print("Best result identified by grid search ", best_result)
```