**Journalistic Voice: exploring the differences between the writing styles of political editors for four UK news sites**

**Project Proposal**

**1 Aims, objectives and background**

**1.1 Introduction**[**¶**](https://d3c33hcgiwev3.cloudfront.net/cmYGiKNgTg2mBoijYK4NXg_2ead7d3d1dd14ff89e2a549e15f085e1_Journalistic-Voice-Project-Proposal-2-.html?Expires=1686873600&Signature=h5vm0F7Mi12HC2K1jsTkOvcOSLrbp~qQJSplGQUbSTT-5~3By0DWESTyS242wjBQ41CXANBlPtCzTV8XD49cYeGP7BlFraDeTlWQmcIQZJg5qxb-8M-Mk0-G9EBCdJLSdwbKNH3Ei25YcUXjTd1p9vilrE9ezb44T6EDBVUN6mc_&Key-Pair-Id=APKAJLTNE6QMUY6HBC5A#1.1-Introduction)

By far the most popular application of data science to the world of journalism is the use of machine learning to classify 'fake news' from 'real news'. However, there exists the potential to use a wider range of data science tools to extract interesting insight from news stories - insights which have previously been the realm of journalism experts writing analyses of texts.

This project was inspired by a recent paper by Koivunen et al. exploring affectivity in news journalism [1]. The authors used data science techniques to explore politeness (via hedging words) and affectivity in Finnish news articles around the topic of a political conflict between the Finnish government and trade unions. They had the resource of a linguist to classify affective/emotive words, however I believe a suitable proxy can be created.

I wanted to explore the ideas of using Natural Language Processing and data science techniques to create metrics to measure features which are often considered subjective, like affectivity, politeness and uniqueness of word choice - attempting to capture the 'voice' of a journalist.

I decided to focus on political journalism within the UK: specifically the work of political editors of popular news publications which publish articles online. To my knowledge no similar work has been done exploring the writing style of political editors in the UK using data science techniques.

**1.2 Aims and objectives**

Within this project, I would like to explore the following:

* Creating measures of voice
  + Measuring affectivity - how much emotional language is in each article?
  + Politeness - using the proxy of hedging words
  + Measuring uniqueness of word choice with TF-IDF
* Analysing article content
  + Which topics each editor has chosen to focus on?
  + Which politicians does each editor talk about?
* Look for relationships between our measures of politeness, affectivity and other features in order to tell a story about the voices of different editors - final visualisations and conclusions should be accessible to someone whose area is journalism studies rather than data science
* As well as the bodies of articles, we can explore some of these questions for article titles

For this project proposal, my aims are to:

1. Decide how much data is required to sufficiently explore the above questions, taking into consideration constraints on time and resource
2. Decide which publications and articles should be used to allow for a coherent analysis
3. Collect the data via webscraping and store in a form which allows for data cleaning and analysis
4. Clean and transform the data such that it is suitable for use in the techniques I would like to utilise later
5. Carry out some exploratory data analysis to identify that there are trends within the data which make it viable for further exploration

**1.3 Data**

**1.3.1 Data requirements**

Four publications were chosen because with more than four publications, it becomes difficult to directly compare the style of editors to each other and explore each one in depth. Graphs and visualisations become more crowded, making it less easy for readers to easy interpret information.

I decided each of the publications should have an equal number of articles, even if this results in an uneven number of total words. This is because one of the techniques I am planning to use within the project is TF-IDF, which in part relies on number of corpora a term appears in. In this context, TF-IDF results would be skewed if some editors were over-represented in regards to number of articles, and others under-represented. Other techniques I plan to use should not be impacted by an unequal number of words across the corpora.

**1.3.2 Choice of publications**

Article sources and political editors chosen:

* Selection of authord redacted

In order to examine different journalistic voices, it made sense to have publications with different affiliations, priorities and audiences:

* BBC News is by far the most popular website or app for news in the UK. According to the 2021 Ofcom report about news consumption in the UK [2], 67% of people who use websites or apps for news use the BBC. The BBC is designed to appeal to as many people in the UK as possible. One of their key values is impartiality [3].
* The Guardian is one of the 4 most popular news websites or apps in the UK, with the majority of it's readership now accessing content online rather than through print newspapers [2]. It is generally considered left-centrist [4].
* The Sun is not one of the 15 most popular news websites or apps in the UK, but remains popular in print format, with only about a quarter of it's readership accessing content online [2]. It is generally considered right-leaning [4].
* The Daily Mirror (referred to simply as the 'Mirror' from this point) was selected as, like the Sun, it is a tabloid newspaper which now publishes content online. It is not one of the 15 most popular news websites or apps in the UK, and only about a third of it's readership accesses content online [2]. In contrast to the Sun, which is considered right-leaning, the Mirror is considered left-leaning [4], meaning they have likely have different audiences.

The Daily Mail was originally chosen as one of the four publications. Their terms and conditions did not forbid webscraping, however, did specify that consent was required to use the content of their websites even for personal and non-commercial use. Consent was requested, and the DMG Media (who own the Daily Mail) licensing team specified the usage fee was £10 per article, which was not feasible.

As it took some time for the licensing team to respond, I had already written a function for extracting data from scraped Daily Mail webpages in the hope that consent would be granted. I have kept this function within this document, although it is not used.

**1.3.3 Choice of articles: methodology**

For each political editor, their 25 most recent articles were selected on the basis that:

* The political editor was listed as the sole author of the article
* The article text was the main element of the article (e.g. the text was not a companion piece to a podcast)
* UK politics was a major focus of the article (e.g. an article focusing primarily on China's nuclear program, with a short quote from someone within the UK government, was excluded). This is because the vast majority of articles (over 95%) focused specifically on UK politics, and including a few articles where this was not a major focus of the article would have created outliers which would have made comparative analysis more difficult in places.

Each article needed to be located, checked for suitability individually and added to the URLs CSV. This takes 2-5 minutes per article depending on the depth an article needs to be checked. Working at speed, the average time spent locating, checking and adding each URL is approximately 3 minutes. Four editors at 25 articles each is 300 minutes (5 hours). Assuming 25-30 hours is a ballpark estimate for number of hours required for a project proposal, it seems reasonable not to spend more than 20% of that time to compile the list of URLs to use within the project.

**1.3.4 Limitations and constraints of the data**

**1.3.4.1 Time periods covered by the corpora**

***Shortness of the time period covered***

The articles which will be used were published over a maximum period of 4 months. This means we will be examining the voice and writing style of each political editor in the last few months, rather than their career as a political editor overall.

Different events and topics may influence a publication and editor's writing style. An editor may feel some topics warrant a relatively polite and neutrally-worded account, while with others allow them to write more affective pieces. A publication's stance (or agenda) on certain topics may also influence writing style - an article where a publication encourages their readers to lobby for change will have a different style to one which is simply reporting information in which the editor or publication has little investment.

Therefore the level of affectivity, politeness and other metrics we can measure could be different for a corpus of 25 articles from a different time period.

Examining a greater number of articles for each author over a larger time period (for example at least two or three years) would yield a much more accurate analysis of their journalistic style. It is not possible here due to time and resource constraints, therefore conclusions risk being unrepresentative of an editor's journalistic style and voice.

***Difference in time periods covered by different publications***

One issue with the data which was identified after the URLs had been compiled was that all the editors published articles at different rates.

For example, the 25 most recent BBC articles which fit the criteria specified covered a period from early July to late October. The 25 most recent Guardian articles fitting the criteria, on the contrary, cover a period from 21 September to 27 October. The Sun and Mirror corpora had timeframes more similar to the Guardian than the BBC.

In an ideal scenario, each of the corpora would span over the same time period. One solution would be to gather all the URLs which fit the criteria for inclusion in a certain time period, and then for each publication randomly select 25 articles to use within the project. Given the time it took to collect the 100 URLs we are using for this project (over 5 hours), and the fact that it is likely it would take two to four times as long to implement this 'ideal solution' for choice of URLs, we have to accept the limitation of uneven time periods.

**1.3.4.2 Uncertain authorship**

Each article has been chosen so that the political editor is listed as the sole author. However, in some news agencies, there is a practice of adding the political editor of a publication as an author when the article was written by a more junior colleague [5], or articles are otherwise misattributed (e.g. neglecting to add other journalists as authors, even if they have worked on a piece). It is impossible for us to identify articles which may not have been solely written by the political editor of a publication, so we cannot mitigate this issue. However, at the least, we expect that if an article is attributed solely to a political editor, that editor has (a) reviewed the article, and (b) feel it reflects their voice and writing style sufficiently to allow it to be published under their name. In this sense, the project can be viewed as examining the voice of a political editor as an *entity* of the publication, rather than one specific person.

**1.4 Ethical considerations**

**1.4.1 Use of article text**

None of the websites has a clause forbidding webscraping in their terms and conditions. Additionally, I checked the robots.txt of each website to ensure there was no Disallow clause on the sections of the websites I wanted to scrape.

Consent to use content:

* The BBC terms and conditions specify that consent must be sought to use content from the BBC website. Permission was sought from the BBC News Permission Team. Consent to use the text of the articles was granted.
* The Guardian terms and conditions state content use is allowed for 'personal and non-commercial use, provided you maintain and abide by any author attribution, copyright or trademark notice or restriction in any material that you download'.
* The Sun terms and conditions specify that users are permitted to use services and intellectual property for personal, private and non-commercial purposes.
* The Mirror terms and conditions state that 'you may download and print portions of the Site for your personal, non-commercial use.'

All data, including author attribution, is in the public sphere, so no anonymisation of data is required.

**1.4.2 Use of logos and other images**

The BBC, Guardian, Sun and Mirror logos are used later in this document, as well as an image of the BBC News globe. These images are used under the 'Fair Dealing' exception to copyright which allows copyrighted material to be used without seeking consent in certain limited circumstances, one of which is 'research and private study' [6].

**1.4.3 Onward use / reuseage of data and derived data**

Any person desiring to use the source data (that is, the article text) would need to abide by the terms and conditions of each publication at the time, and independently seek permission for use where required. Data derived from the source text which contains words from the source would fall under the same restrictions as the source text.

Analysis and conclusions are my own.

**1.4.4 Potential impacts of using article data for the proposes analyses**

Some data uses have potentially negative impacts on people or businesses, such as perpetuating harmful and incorrect assumptions, or causing loss of revenue.

I have assessed the potential harms of this project, and the way it will utilise the source data, as minimal for the following reasons:

* This project will focus on analysing objective features of text, rather than make judgements about the political editors who wrote them, their publications, the publication's readership or any other persons.
* The conclusions of this project will not claim to be a fully representative analysis of any editor's journalistic style or voice. Limitations of the data used in this project have been outlined above. Limitations of techniques used will be discussed throughout the project.

In [1]:

*# Import libraries and modules*

**import** **pandas** **as** **pd**

**from** **bs4** **import** BeautifulSoup

**import** **requests**

**import** **nltk**

**from** **nltk.sentiment.vader** **import** SentimentIntensityAnalyzer

**from** **nltk.corpus** **import** stopwords

**from** **nltk.tokenize** **import** word\_tokenize

**from** **wordcloud** **import** WordCloud, ImageColorGenerator

**import** **matplotlib.pyplot** **as** **plt**

**import** **seaborn** **as** **sns**

**import** **numpy** **as** **np**

**from** **PIL** **import** Image

**import** **re**

**from** **re** **import** sub

**import** **time**

**from** **nltk.stem** **import** WordNetLemmatizer

*# Show all matplotlib graphs inline*

%**matplotlib** inline

*# Set all graphs to a seaborn style with a grey background grid which makes reading graphs easier*

sns.set()

**2 Webscraping articles**

**2.1 Defining scraping and extraction functions**

The list of URLs to be analysed during this project is in a CSV file. Prior to importing this file and scraping the content of the URLs, we need to define a function which we can call to scrape the data as we iterate through the URLs. Each website we are going to scrape from has different HTML, so we also need to write a different function to extract the required information from each.

Please note the webscraping takes 5-10 minutes because the Mirror's robots.txt specifies a crawl delay requirement of 10 seconds. Any faster and we'll be relegated to the website's ban list!

The below function checks if a webpage is accessible and returns the content if it is:

In [2]:

**def** getParsedWebpage(url, website):

*# Check that page is accessible for scraping*

headers = {"User-Agent":"Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:66.0) Gecko/20100101 Firefox/93.0",

"Upgrade-Insecure-Requests":"1",

"Accept":"text/html,application/xhtml+xml,application/xml;q=0.9,\*/\*;q=0.8",

"Accept-Encoding":"gzip, deflate, br",

"Accept-Language": "en-GB,en-US;q=0.9,en;q=0.8",

"DNT":"1"}

**if** website == 'mirror':

time.sleep(12)

response = requests.get(url, headers=headers)

**if** response.status\_code != 200:

soup = 'error'

**else**:

content = response.content

*# Parse the webpage now we've checked it's accessible*

soup = BeautifulSoup(response.content, 'html.parser')

**return** soup

**Extracting information from BBC webpages**

The following function extracts title, author and article text from a BBC News webpage:

In [3]:

*# code redacted*

**Extracting information from The Guardian webpages**

The following function extracts title, author and article text from a Guardian webpage:

In [4]:

**def** extractInfoGuardian(soup):

*# Extract article title*

title = soup.find(['h1'])

*# Extract article author*

author = soup.find('a', {'rel': 'author'})

*# Extract article text*

article = soup.find('div', {'id': 'maincontent'}).find\_all("p")

*# Complile all article text into one string and ensure there is 1 space between each word*

article\_text = ''

**for** content **in** article:

article\_text = article\_text + ' ' + content.get\_text().lstrip().rstrip()

*# Guardian quotation marks are formatted “ and ” rather than ", and not recognised as standard quotation marks*

*# by python. Therefore we replace them at this stage.*

article\_text = article\_text.replace('“', '"')

article\_text = article\_text.replace('”', '"')

*#Same issue with ` and ’ rather than '*

article\_text = article\_text.replace('`', "'")

article\_text = article\_text.replace('’', "'")

**return** title.get\_text(), author.get\_text(), article\_text[1:]

**Extracting information from The Sun webpages**

The following function extracts title, author and article text from a The Sun webpage:

In [5]:

*# code redacted*

**Extracting information from Mirror webpages**

The following function extracts title, author and article text from a Daily Mail webpage.

In [6]:

*# code redacted*

**Extracting information from Daily Mail webpages**

The following function extracts title, author and article text from a Daily Mail webpage. It will not be used for this project.

In [7]:

*# code redacted*

**2.2 Import URLs and scrape**

**2.2.1 Read in URLs and get website name**

Now we have defined all the functions we are going to use, we read the CSV of URLs into a dataframe and the extract website name from the URL:

In [8]:

urls = pd.read\_csv("URLs.csv")

In [9]:

*# Get publication name from the url and add as a new column in the dataframe*

urls['Website'] = ''

**for** i **in** range(0, len(urls)):

start\_index = (urls.iloc[i].URL.find('www.')) + 4

end\_index = urls.iloc[i].URL.find('.co')

urls['Website'][i] = urls.iloc[i].URL[start\_index:end\_index]

In [ ]:

urls

**2.2.2 Scrape data and save to dataframe**

Next we will attempt to scrape the articles at each of the URLs we have imported using the functions defined earlier. If successful, we will append to a dataframe of scraped information.

In [11]:

scraped\_data = pd.DataFrame(columns=['URL\_ID', 'Publication', 'Author', 'Title', 'Text'])

**for** i **in** range(0, len(urls)):

url = urls['URL'][i]

*# Check webpage is accessible and get parsed webpage, if not accessible soup will be returned as 'error'*

soup = getParsedWebpage(urls['URL'][i], urls['Website'][i])

*# If there was an error accessing the webpage, continue onto next iteration of loop*

**if** soup == 'error':

print("Error fetching webpage, data not scraped:", url)

**continue**

publication = urls['Website'][i]

*# If url is from a publication we have planned to scrape, extract information using appropriate function*

**if** publication == 'bbc':

title, author, article\_text = extractInfoBBC(soup)

pub\_name = 'BBC News'

**elif** publication == 'theguardian':

title, author, article\_text = extractInfoGuardian(soup)

pub\_name = 'The Guardian'

**elif** publication == 'thesun':

title, author, article\_text = extractInfoSun(soup)

pub\_name = 'The Sun'

**elif** publication == 'mirror':

title, author, article\_text = extractInfoMirror(soup)

pub\_name = 'The Mirror'

**else**:

print("The following URL is from", publication, "which is not a website we have planned to scrape:", url)

**continue**

article\_info = {'URL\_ID': i, 'Publication': pub\_name, 'Author': author, 'Title': title, 'Text': article\_text}

scraped\_data = scraped\_data.append(article\_info, ignore\_index=**True**)

**2.3 Check scraped data**

We need to check that the data was scraped as intended, and that the data is not outside of expected bounds.

One of the checks we can carry out ensuring the number of words in the title and the article text falls into normal bounds. First we need to calculate the length of the title and text:

In [12]:

*# Calculate length (number of words) of each article title and text, add to dataframe*

scraped\_data['Text Length'] = 0

scraped\_data['Title Length'] = 0

**for** i **in** range(0, len(scraped\_data)):

text\_len = len(scraped\_data['Text'][i].split(' '))

title\_len = len(scraped\_data['Title'][i].split(' '))

scraped\_data.loc[i,['Text Length']] = text\_len

scraped\_data.loc[i,['Title Length']] = title\_len

In [13]:

scraped\_data.describe().round(1)

Out[13]:

|  | **Text Length** | **Title Length** |
| --- | --- | --- |
| **count** | 100.0 | 100.0 |
| **mean** | 561.4 | 12.7 |
| **std** | 226.3 | 3.1 |
| **min** | 129.0 | 6.0 |
| **25%** | 375.5 | 10.0 |
| **50%** | 556.5 | 13.0 |
| **75%** | 717.2 | 14.2 |
| **max** | 1115.0 | 21.0 |

The below code will print a warning message if there are any issues with our data:

In [14]:

*# Checking scraped\_data has the same number as rows as the number of URLs we intended for scraping*

**if** len(urls) == len(scraped\_data):

print("All articles scraped and inserted into scraped\_data.")

**else**:

print(len(urls)-len(scraped\_data), "articles were not scraped correctly and were not entered into scraped\_data.")

*# Checking there is no missing (null, NA, NaN) data in articles*

total\_missing = 0

**for** column **in** scraped\_data:

num\_missing = scraped\_data[column].isna().sum()

total\_missing =+ num\_missing

**if** num\_missing > 0:

print(column, "has", num\_missing, "pieces of missing (null, NA, NaN) data.")

**if** total\_missing == 0:

print("There is no missing (null, NA, NaN) data in the scraped\_data dataframe.")

**else**:

print("There is a total of", total\_missing, "pieces of missing (null, NA, NaN) data in the scraped\_data dataframe.")

*# Check there are 4 unique editors and 4 unique publications*

**if** len(scraped\_data['Author'].unique()) != 4:

print("There are", len(scraped\_data['Author'].unique()), "unique values in author, but there should be 4.")

**if** len(scraped\_data['Publication'].unique()) != 4:

print("There are", len(scraped\_data['Publication'].unique()), "unique values in publication, but there should be 4.")

*# Check title and text word count*

**if** scraped\_data['Text Length'].min() < 125:

print("There is an article with less than 125 words - out of expected bounds. Check for incorrectly scraped data.")

**if** scraped\_data['Text Length'].min() > 1200:

print("There is an article with more than 1200 words - out of expected bounds. Check for incorrectly scraped data.")

**if** scraped\_data['Title Length'].min() < 5:

print("There is an article title with less than 5 words - out of expected bounds. Check for incorrectly scraped data.")

**if** scraped\_data['Title Length'].min() > 25:

print("There is an article title with more than 25 words - out of expected bounds. Check for incorrectly scraped data.")

All articles scraped and inserted into scraped\_data.

There is no missing (null, NA, NaN) data in the scraped\_data dataframe.

In [ ]:

scraped\_data

In [ ]:

*# Review article text to ensure no obvious issues are visible*

**for** i **in** range(0, len(scraped\_data)):

print(scraped\_data['Text'][i])

print()

**2.4 Import previously scraped data**

The risk of the news article data changing over the duration of the project is relatively low as articles are published with the intention that they are archived in the long term and the text not updated except in unavoidable circumstances (for example corrections). However, there is still the risk that article webpages may undergo the following changes:

* Changes to HTML of the webpage
* Corrections and amendments to the article text
* Addition of text to state that an article is more than a year old (or other time period) and may contain out of date information, which may be inadvertently included in article text during scraping
* In exceptional circumstances, deletion of controversial articles (for example, where the subject of an article has threatened or initiated legal action against the publisher)

To ensure consistency and replicable results, for the remainder of the project we will use previously scraped and saved article data.

In [17]:

*# Scraped data was previously saved to CSV*

*# scraped\_data.to\_csv('articles.csv')*

In [18]:

*# Read previously scraped data*

articles = pd.read\_csv('articles.csv', index\_col=0)

In [19]:

*# Previously scraped data should be the same as freshly scraped data*

(scraped\_data == articles).describe()

Out[19]:

|  | **URL\_ID** | **Publication** | **Author** | **Title** | **Text** | **Text Length** | **Title Length** |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **count** | 100 | 100 | 100 | 100 | 100 | 100 | 100 |
| **unique** | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| **top** | True | True | True | True | True | True | True |
| **freq** | 100 | 100 | 100 | 100 | 100 | 100 | 100 |

**3 Data cleaning and processing**

Different analysis techniques will need versions of the data with different types of processing:

* Need full corpora, including stop words, for affectivity and politeness analysis. For example, some hedging words (which we will use to create a measure of politeness) can also be considered stop words.
* Will need versions of the corpora without stop words, and versions which have been stemmed or lemmatized, for techniques exploring topics and key points of discussion.

**Vectorisation:** For some of the techniques in this project I will need to vectorise my data, for example using scikit-learn's CountVectorizer. I am as yet unsure of (a) what data I need to vectorise and (b) what the data structure I will need to store my vectorised data in to make it most easily accessible for the techniques.

Beyond the scope of EDA therefore not included.

**3.1 Editors dataframe**

**3.1.1 Create Editors dataframe**

We create a dataframe with a row for each editor where we can store aggregated information about each editor's corpora. Initially it only contains editor name and publication, but more data will be appended later:

In [ ]:

editor\_names = articles['Author'].unique().tolist()

publications = articles['Publication'].unique().tolist()

editors = pd.DataFrame({'Editor': editor\_names, 'Publication': publications})

editors

**3.1.2 Compile corpora**

For some later analysis we will do, it would be useful to have aggregated corpora for each editor. Here, we will consider article titles as part of an author's full corpus.

It might be useful to analyse article titles as a separate concept: within journalism, the title often holds special consideration. This is the first point of contact for readers - the title has to draw them in. We will create a corpus of titles for each author.

In [21]:

*# Create a new column to store the full corpus and title corpus for each editor*

editors['Full corpus']=''

editors['Title corpus']=''

*# Iterate through each article, appending article title and text onto corresponding corpus*

**for** i **in** range(0, len(articles)):

**if** articles['Publication'][i] == 'BBC News':

editors.loc[0,['Full corpus']] += ' ' + articles['Title'][i] + ' ' + articles['Text'][i]

editors.loc[0,['Title corpus']] += ' ' + articles['Title'][i]

**elif** articles['Publication'][i] == 'The Guardian':

editors.loc[1,['Full corpus']] += ' ' + articles['Title'][i] + ' ' + articles['Text'][i]

editors.loc[1,['Title corpus']] += ' ' + articles['Title'][i]

**elif** articles['Publication'][i] == 'The Sun':

editors.loc[2,['Full corpus']] += ' ' + articles['Title'][i] + ' ' + articles['Text'][i]

editors.loc[2,['Title corpus']] += ' ' + articles['Title'][i]

**elif** articles['Publication'][i] == 'The Mirror':

editors.loc[3,['Full corpus']] += ' ' + articles['Title'][i] + ' ' + articles['Text'][i]

editors.loc[3,['Title corpus']] += ' ' + articles['Title'][i]

In [22]:

*# We can now look at some summary statistics for each author*

**for** i **in** range(0, len(editors)):

msg = """**{Publication}** has a total corpus of **{ctotal}** words, which is an average of **{caverage}** words per article.

Each title has an average of **{taverage}** words.""". format(

Publication = editors['Publication'][i],

ctotal = len(editors['Full corpus'][i].split(' ')),

caverage = round(len(editors['Full corpus'][i].split(' '))/25),

taverage = round(len(editors['Title corpus'][i].split(' '))/25)

)

print(msg)

print()

len(articles['Text'][i].split(' '))

BBC News has a total corpus of 13549 words, which is an average of 542 words per article.

Each title has an average of 10 words.

The Guardian has a total corpus of 17459 words, which is an average of 698 words per article.

Each title has an average of 11 words.

The Sun has a total corpus of 10350 words, which is an average of 414 words per article.

Each title has an average of 16 words.

The Mirror has a total corpus of 16056 words, which is an average of 642 words per article.

Each title has an average of 14 words.

**3.2 Removing stop words**

We need to have versions of our corpora without stop words. Some analyses look at the frequency of words, and without removing stop words, some words (such as "the") will appear with high frequency despite not being meaningful for our analysis.

**3.2.1 Identifying stop words not covered by NLTK's stop words**

Stop words can be removed using nltk's stop words. However, there may be other words in the corpus which can be functionally considered stop words and which are of little use to us in comparative analysis. To spot some such words and remove them, we first analyse frequency of terms in each corpus, to see which are most common:

In [ ]:

*# First we define a function to get the word frquencies of each corpus*

*# At this stage we will also remove stop words as defined by nltk's stopwords set*

nltk.download('stopwords')

**def** getFreqs(corpus):

*# Remove certain punctuation using regex. This will stop 'said' and 'said.' appearing as different words*

*# Do not remove apostrophes*

corpus = re.sub('\.|\,|**\"**|\“|\”|\—|\–|\-|\?|\!|\:|\;|\(|\)', '', corpus)

*# Split the corpus into a list of words*

corpus\_lst = corpus.lower().split()

*# Converting to a set means all duplicate words are removed*

unique\_words\_full = set(corpus\_lst)

*# Remove stop words*

unique\_words = [word **for** word **in** unique\_words\_full **if** **not** word **in** stopwords.words('english')]

*# The dataframe within which we will store our word-frequency pairs*

word\_freqs = pd.DataFrame(columns=['Word', 'Frequency'])

**for** word **in** unique\_words :

word\_freq\_pair = {'Word': word, 'Frequency': corpus\_lst.count(word)}

word\_freqs = word\_freqs.append(word\_freq\_pair, ignore\_index=**True**)

**return** word\_freqs

We can use the above function to get the word frequencies of each corpus and assess if we would like to treat any of the most frequent words as stop words.

In [24]:

*# Generate word frequencies*

BBC\_word\_freqs = getFreqs(editors['Full corpus'][0])

Guardian\_word\_freqs = getFreqs(editors['Full corpus'][1])

Sun\_word\_freqs = getFreqs(editors['Full corpus'][2])

Mirror\_word\_freqs = getFreqs(editors['Full corpus'][3])

Let's have a look at many unique words each corpus has (with nltk stop words removed):

In [25]:

print('BBC:', BBC\_word\_freqs.index.max()+1)

print('Guardian:', Guardian\_word\_freqs.index.max()+1)

print('Sun:', Sun\_word\_freqs.index.max()+1)

print('Mirror:', Mirror\_word\_freqs.index.max()+1)

BBC: 2741

Guardian: 3352

Sun: 2397

Mirror: 3068

That's a lot of words! If we look at the top 300 (approximately 10%) of most common words in each corpus, we can remove the most common stop words that aren't in nltk's stop word corpus. Many of the top words in each corpus will be the same, so we can save ourselves some time by combining the top 300 words from each corpus and then only displaying unique words.

In [ ]:

*# Get lists of 300 most frequent words in each corpus*

BBC\_top\_words = list((BBC\_word\_freqs.sort\_values('Frequency', ascending=**False**).head(300))['Word'])

Guardian\_top\_words = list((Guardian\_word\_freqs.sort\_values('Frequency', ascending=**False**).head(300))['Word'])

Sun\_top\_words = list((Sun\_word\_freqs.sort\_values('Frequency', ascending=**False**).head(300))['Word'])

Mirror\_top\_words = list((Mirror\_word\_freqs.sort\_values('Frequency', ascending=**False**).head(300))['Word'])

all\_top\_words = BBC\_top\_words + Guardian\_top\_words + Sun\_top\_words + Mirror\_top\_words

print(set(all\_top\_words))

I've chosen to classify the following as stop words, and remove them from the corpora:

In [27]:

custom\_stopwords = ['said', 'party', 'could', 'would', 'might', 'however', "that's", "i'm", "we're", "there's", 'perhaps', 'also', 'that’s', 'it’s']

print(custom\_stopwords)

['said', 'party', 'could', 'would', 'might', 'however', "that's", "i'm", "we're", "there's", 'perhaps', 'also', 'that’s', 'it’s']

'Party' was removed because it is almost always used in conjunction with Conservative/Tories/Labour etc, which are the terms which give us the useful information - party becomes reductive.

I chose to be conservative with the words I additionally classified as stop words in order to retain information. We can always remove more words from the corpora at a later stage.

**3.2.2 Create filtered corpora**

Now we have decided on stop words, we can define our stopword removal function:

In [ ]:

nltk.download('punkt')

**def** removeStopwords(corpus):

corpus = corpus.lower()

*# Remove certain punctuation using regex. This will stop 'said' and 'said.' appearing as different words*

*# Do not remove apostrophes*

corpus = re.sub('\.|\,|**\"**|\“|\”|\—|\–|\-|\?|\!|\:|\;|\/|\(|\)', '', corpus)

*# Tokenize the corpus - this creates a list of all the words in the corpus*

tokens = word\_tokenize(corpus)

*# As well as stopwords, we want to remove tokens with only punctuation, or suffixes separated from words*

punct = ["'", "''", '``', '(', ')', '%', '&', '...', '…', "‘", "’"]

suffixes = ["'s", "n't", "'ve", "'ll", "'re", "'d"]

remove\_words = punct + suffixes + custom\_stopwords + stopwords.words('english')

*# Create a new list with stop words removed from the corpus*

filtered\_tokens = [word **for** word **in** tokens **if** **not** word **in** remove\_words]

*# Reconstruct corpus as a string*

filtered\_corpus = ' '.join(filtered\_tokens)

**return** filtered\_corpus

Now we create our filtered corpora:

In [29]:

*# Filter the editors corpora and add as a new column*

editors['Filtered corpus']=''

editors['Filtered title corpus']=''

**for** i **in** range(0, len(editors)):

editors.loc[i, 'Filtered corpus'] = removeStopwords(editors['Full corpus'][i])

editors.loc[i, 'Filtered title corpus'] = removeStopwords(editors['Title corpus'][i])

In [30]:

*# Filter the articles corpora and add as a new column*

articles['Filtered text']=''

articles['Filtered title']=''

**for** i **in** range(0, len(articles)):

articles.loc[i, 'Filtered text'] = removeStopwords(articles['Text'][i])

articles.loc[i, 'Filtered title'] = removeStopwords(articles['Title'][i])

**3.3 Lemmatization**

We can 'distill' the words we have into their roots to make them more meaningful for analysis. We can either use stemming or lemmatization. Stemming removes suffixes to create a root stem (so 'studies' would become 'stud'), while lemmatization will return a root word ('study').

In our later analysis, we want to be able to see whole, meaningful words and make impactful visualisations using them. Therefore in this case it is better to use lemmatization to get root words.

Because creating the root of words may sometimes give us unexpected results, we will create a separate column for lemmatized corpora. We'll want to use the filtered corpora with stop words removed, but without lemmatization, for some of our exploratory data analysis later.

Below we create an instance of the wordnet lemmatizer, create a lemmatizing function and apply to the filtered corpora:

In [31]:

lemmatizer = WordNetLemmatizer()

**def** lemmatizeCorpus(corpus):

tokens = word\_tokenize(corpus)

lemmatized\_tokens = [lemmatizer.lemmatize(word) **for** word **in** tokens]

**return** ' '.join(lemmatized\_tokens)

In [32]:

*# Lemmatize the editors corpora and add as a new column*

editors['Lemmatized corpus']=''

editors['Lemmatized title corpus']=''

**for** i **in** range(0, len(editors)):

editors.loc[i, 'Lemmatized corpus'] = lemmatizeCorpus(editors['Filtered corpus'][i])

editors.loc[i, 'Lemmatized title corpus'] = lemmatizeCorpus(editors['Filtered title corpus'][i])

In [33]:

*# Lemmatize the articles corpora and add as a new column*

articles['Lemmatized text']=''

articles['Lemmatized title']=''

**for** i **in** range(0, len(articles)):

articles.loc[i, 'Lemmatized text'] = lemmatizeCorpus(articles['Filtered text'][i])

articles.loc[i, 'Lemmatized title'] = lemmatizeCorpus(articles['Filtered title'][i])

**4 Generating features**

**4.1 Number of quotations**

The number of quotes is a useful metric to calculate, and we can later explore the relationship between number of quotations in an article and other characteristics of articles. Additionally, calculate the number of quotes relative to length of each article as this is a more 'standardised' way to look at number of quotations.

In [34]:

articles['Quotes'] = 0

articles['Quotes per 1000 words'] = 0

**for** i **in** range(0, len(articles)):

quotes = articles['Text'][i].count('"')

articles.loc[i,['Quotes']] = quotes/2

*# Because dividing number of quotes by length of text would generate a small number, we instead calculate*

*# number of quotes (defined above as quotation marks divided by 2) per 1000 words*

articles.loc[i,['Quotes per 1000 words']] = (quotes/articles['Text Length'][i])\*500

Note that the above method for calculating quotes is a rough approximation. There are times when quotes will have more than two quotation marks. For example, the following quote from a BBC article has five quotation marks:

"Ok is the first Icelandic glacier to lose its status as glacier," it reads.

"In the next 200 years all our main glaciers are expected to follow the same path. This monument is to acknowledge that we know what is happening and what needs to be done.

"Only you know if we did it."

We can make a broad assumption that journalists will use the same conventions in regards to using quotation marks (to maximise readability and clarity), and that therefore the number of quotation marks is still a suitable proxy for number of quotes when comparing articles.

**4.2 'Said' density**

In addition to quotation marks, the word 'said' could be used as a proxy for quotations, including paraphrased quotations or those not included in quotation marks, for example: 'Today the Prime Minister said that the government would not...'

Let's count the number of times 'said' is used in each article as density per 1000 words. We'll explore whether this feature correlates with any others in the exploratory data analysis.

In [35]:

*# Count number of times 'said' is used per 1000 words*

articles['Said density per 1000 words'] = 0

**for** i **in** range(0, len(articles)):

word\_lst = articles['Text'][i].lower().split()

articles.loc[i,['Said density per 1000 words']] = (word\_lst.count('said')/articles['Text Length'][i])\*1000

**4.3 Sentiment analysis**

I will be using nltk's sentiment analyser with the VADER lexicon to generate sentiment measurements. I have chosen to run the sentiment analyser on full, unfiltered corpora because using filtered corpora sometimes means losing information through loss of context, whereas there is relatively little cost to keeping stop words (etc).

Negative, neutral, and positive sentiments should add up to one (as they represent proportions). Compound sentiment is a normalised measurement with adjusted intensity, which outputs a value from 1 to -1 (positive to negative) [7].

In [ ]:

*# Download VADER (Valence Aware Dictionary and sEntiment Reasoner) lexicon, a sentiment analysis tool*

nltk.download('vader\_lexicon')

*# Create an instance of the nltk sentiment analyser*

analyser = SentimentIntensityAnalyzer()

In [37]:

*# For each article, add the negative, neutral, positive and compound sentiment score as four new columns*

articles['Neg\_Sentiment'] = 0

articles['Neu\_Sentiment'] = 0

articles['Pos\_Sentiment'] = 0

articles['Comp\_Sentiment'] = 0

**for** i **in** range(0, len(articles)):

*# Get sentiment score - sen\_score has type dictionary*

sen\_score = analyser.polarity\_scores(articles['Text'][i])

*#Extract individual sentiment scores from the dictionary and enter them into the dataframe*

articles.loc[i,['Neg\_Sentiment']] = sen\_score['neg']

articles.loc[i,['Neu\_Sentiment']] = sen\_score['neu']

articles.loc[i,['Pos\_Sentiment']] = sen\_score['pos']

articles.loc[i,['Comp\_Sentiment']] = sen\_score['compound']

**5 Exploratory data analysis**

**5.1 Wordclouds**

We can see what each editor has chosen to focus on in their most recent articles by creating a wordcloud of their corpus, using the filtered corpora we created earlier with stop words removed.

Wordclouds are a visualisation and visualisation is all about communication. If we generate only standard wordclouds, it is not immediately obvious which cloud corresponds to which editor. We can use masking to make our wordclouds more visually effective and linked to each publication's branding.

The wordcloud masks used:

* BBC: both the BBC logo and the BBC News globe were trialled. Using the logo, it is a bit difficult to make out the letters 'BBC' due to the detailed nature of the image. The wordcloud we get from using the BBC News globe as a mask is more visually effective.
* The Guardian: the Guardian 'g' logo
* The Sun: The Sun logo (with 'The' removed to just leave 'Sun')
* The Mirror: the Mirror 'M' logo

In [38]:

*# A function to create a wordcloud given a corpus and an image to use as a mask*

**def** createMaskedWordcloud(corpus, maskimage, size):

mask = np.array(Image.open(maskimage))

wordcloud = WordCloud(background\_color="white", mask=mask).generate(corpus)

*# Extract colours from image and use in wordcloud*

colours = ImageColorGenerator(mask)

plt.figure(figsize=size)

plt.imshow(wordcloud.recolor(color\_func=colours), interpolation="bilinear")

plt.axis("off")

plt.show()

In [ ]:

*# Creating a wordcloud for the BBC using the BBC logo*

createMaskedWordcloud(editors['Filtered corpus'][0], "Images/bbc\_logo\_red.jpg", [12,12])

*# wordcloud output redacted from this section*

In [ ]:

*# BBC News globe*

createMaskedWordcloud(editors['Filtered corpus'][0], "Images/BBC\_globe.jpg", [10,10])

From the BBC's wordcloud, there is the impression that the editor has given almost equal coverage to both the sitting government and the opposition - although Boris Johnson and the Conservatives have received slightly more mention. 'One' is one of the most prominent words, but it unclear what it refers to. Later in the project we can use regular expressions to extract the words before and after instances of 'one' to identify its context.

In [ ]:

*# The Guardian*

createMaskedWordcloud(editors['Filtered corpus'][1], "Images/guardian\_g.jpg", [10, 10])

From the Guardian's word cloud, there is a sense that that the current government (the Conservative party), the opposition (the Labour party), and their respective leaders receive a similar amount of coverage. Rishi Sunak also appears to feature heavily in Guardian news articles. 'Public' is one of the most prominent words - likely, it refers to the British public.

In [ ]:

*# The Sun*

createMaskedWordcloud(editors['Filtered corpus'][2], "Images/Sun\_logo.jpg", [20,50])

We can see some interesting trends in The Sun's wordcloud: Rishi Sunak's name is bigger than Boris Johnson's, and 'Chancellor', Rishi Sunak's position, is one of the most prominent words. 'Budget' and 'price' are also prominent, indicating The Sun's political editor has chosen to focus on financial matters, and Rishi Sunak in particular, in recent months. The Labour party and their leader, Kier Starmer, appear to be mentioned significantly less in The Sun than in other publications.

In [ ]:

*# The Mirror*

createMaskedWordcloud(editors['Filtered corpus'][3], "Images/mirror\_M\_logo.jpg", [10,10])

We can very clearly see the Mirror has two priorities: Labour and People. In this context, 'people' likely refers to the British public. In contrast to the other publications, the Mirror appears to focus much more strongly on the opposition party rather than the sitting government - and they seem to be focusing on the Labour party holistically, rather than on it's leader, Kier Starmer.

**5.2 Word count boxplot**

Earlier we saw that the Guardian had the highest word count per article on average, the Sun had the lowest. Let's create a box plot to explore the distribution of word counts across each publication:

In [44]:

*# Create a colour palette for graphs where we want to differenciate the four different publications*

pub\_palette = sns.blend\_palette(['crimson', 'blue', 'goldenrod', 'darkcyan'], 4)

pub\_palette

Out[44]:

In [45]:

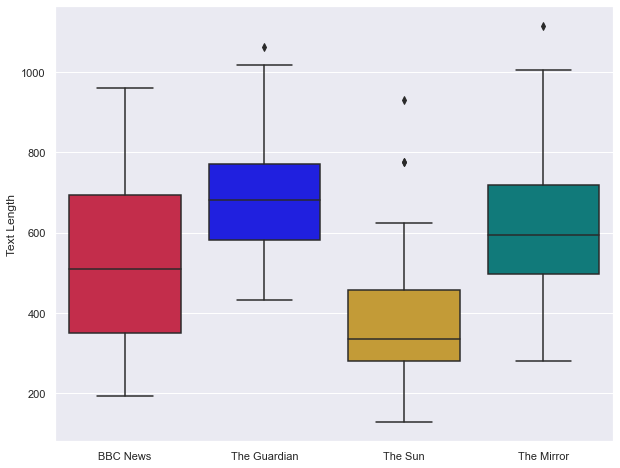
plt.gcf().set\_size\_inches(10, 8)

ax = sns.boxplot(x='Publication', y='Text Length', data=articles, palette=pub\_palette)

ax.set(xlabel=**None**)

Out[45]:

[Text(0.5, 0, '')]



Here we can see that for the Guardian and the Sun, their mean word count is representative of the length of article their editors tend to write: the Guardian editor often writes long articles relative to the cohort, and the Sun editor (apart from a couple of outliers) tends to write relatively short articles.

The Mirror has a wider spread, but BBC News stands out as different from the rest of the publications by having a much wider variety of article lengths - the BBC editor does not seem to stick to a 'typical word length' as much as the other editors.

**5.3 Sentiment boxplots**

Let's look at how sentiment varies across publications.

It makes sense to look at negative, neutral, and positive sentiments together as they are interrelated and should add up to one. Compound sentiment has a different scale (1 to -1, positive to negative), so we should look at this separately.

We can explore sentiment through boxplots, however creating one boxplot to display negative, neutral and positive sentiments for each publication isn't very easy to do from our articles dataframe as it currently stands. Let's use the pandas melt function to create a new dataframe of 'unpivoted' sentiment scores.

In [46]:

sentiments = pd.melt(articles, id\_vars =['Publication'],

value\_vars =['Neg\_Sentiment', 'Neu\_Sentiment', 'Pos\_Sentiment'])

sentiments

Out[46]:

|  | **Publication** | **variable** | **value** |
| --- | --- | --- | --- |
| **0** | BBC News | Neg\_Sentiment | 0.136 |
| **1** | BBC News | Neg\_Sentiment | 0.122 |
| **2** | BBC News | Neg\_Sentiment | 0.078 |
| **3** | BBC News | Neg\_Sentiment | 0.068 |
| **4** | BBC News | Neg\_Sentiment | 0.181 |
| **...** | ... | ... | ... |
| **295** | The Mirror | Pos\_Sentiment | 0.070 |
| **296** | The Mirror | Pos\_Sentiment | 0.162 |
| **297** | The Mirror | Pos\_Sentiment | 0.157 |
| **298** | The Mirror | Pos\_Sentiment | 0.074 |
| **299** | The Mirror | Pos\_Sentiment | 0.123 |

300 rows × 3 columns

Now we can create a box plot:

In [47]:

*# Create boxplot for negative, neutral and positive sentiment scores*

plt.gcf().set\_size\_inches(15, 8)

ax = sns.boxplot(x='variable', y='value', data=sentiments, hue='Publication',

palette=pub\_palette)

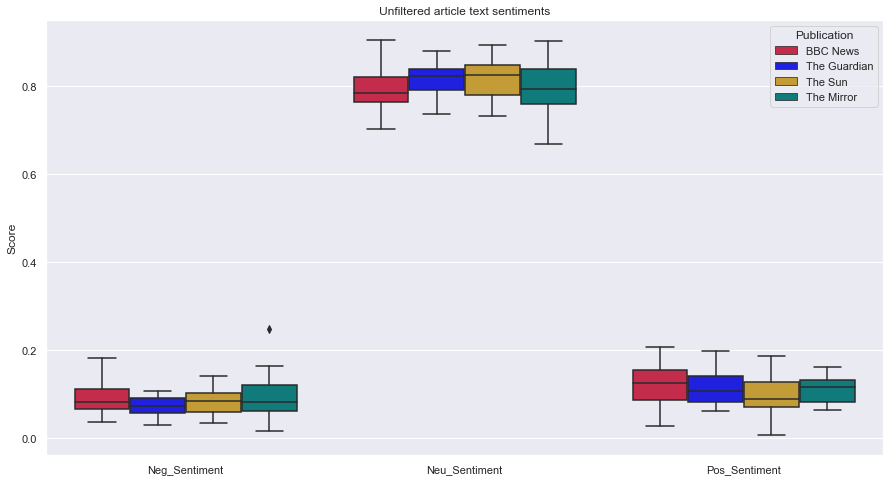
ax.set(title='Unfiltered article text sentiments', xlabel=**None**, ylabel='Score')

Out[47]:

[Text(0.5, 1.0, 'Unfiltered article text sentiments'),

Text(0.5, 0, ''),

Text(0, 0.5, 'Score')]



Here, we can see there isn't much variation between the sentiments of the different publications, and most of the sentiment is overwhelmingly neutral. Let's see if the compound sentiment will give us some more useful information to differentiate typical sentiment between publications:

In [48]:

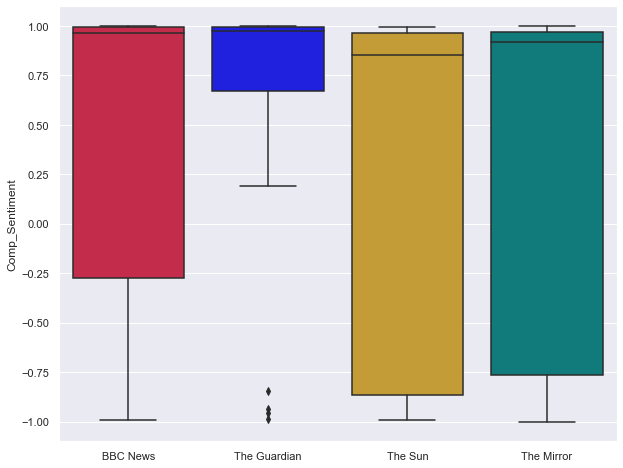
plt.gcf().set\_size\_inches(10, 8)

ax = sns.boxplot(x='Publication', y='Comp\_Sentiment', data=articles, palette=pub\_palette)

ax.set(xlabel=**None**)

Out[48]:

[Text(0.5, 0, '')]



This is a unexpected result - the vast majority of articles are classified as overwhelmingly positive, which doesn't seem right given that news agencies have a reputation for reporting on negative events more often than positive ones.

Some say the VADER lexicon is best used on smaller texts of less than 300 words [7] (for example tweets would be a great use case). It would be worth exploring alternatives for sentiment analysis more suitable for longer form text.

**5.4 Word count and quotation density correlation**

We can look at the relationship between article features to ask questions such as 'do editors use more quotes (per word) in longer articles?'

Below is a scatter plot with a line of best fit for each publication exploring this question:

In [49]:

plt.gcf().set\_size\_inches(15, 8)

*# Plot a regplot (regression plot) as a 'line of best fit' for each publication on top of a scatter plot*

ax = sns.regplot(x='Text Length', y='Quotes per 1000 words', data=articles[articles['Publication']=='BBC News'],

scatter=**False**, ci=**None**,

line\_kws={'color':'crimson'})

ax = sns.regplot(x='Text Length', y='Quotes per 1000 words', data=articles[articles['Publication']=='The Guardian'],

scatter=**False**, ci=**None**,

line\_kws={'color':'blue'})

ax = sns.regplot(x='Text Length', y='Quotes per 1000 words', data=articles[articles['Publication']=='The Sun'],

scatter=**False**, ci=**None**,

line\_kws={'color':'goldenrod'})

ax = sns.regplot(x='Text Length', y='Quotes per 1000 words', data=articles[articles['Publication']=='The Mirror'],

scatter=**False**, ci=**None**,

line\_kws={'color':'darkcyan'})

ax = sns.scatterplot(x='Text Length', y='Quotes per 1000 words', data=articles,

hue='Publication', alpha=0.65, s=80, palette=pub\_palette, edgecolor=**None**)

ax.set(title='Text Length vs Quote Density')

plt.rcParams['axes.labelsize'] = 14

plt.rcParams['xtick.labelsize'] = 14

plt.rcParams['ytick.labelsize'] = 14

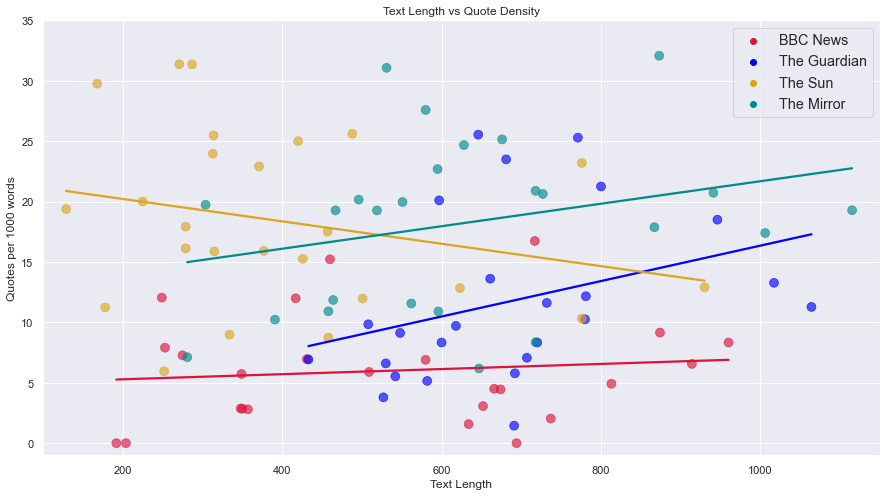
plt.rcParams['axes.titlesize'] = 18

plt.legend(fontsize='large')

plt.xlim(100, 1150)

plt.ylim(-1, 35)

plt.show()



Here we see several patterns:

* The BBC uses the least number of quotes, and has little correlation between text length and quote density
* Both negative and positive correlations exist - with The Sun's editor being alone in utilising less quotes on longer articles than shorter articles
* The Guardian and Mirror lines of best fit have similar gradients, and their points are clustered closely compared to the other two publications - it would be interesting to know what other similarities exist between these two corpora

Although this is only one example, it does demonstrate that there are differences between the corpora which can be investigated.

**5.5 PairGrid**

The above is a detailed look at the relationship between two features, however looking at each set of two features individually to see if there is an interesting relationship is not always the best approach.

Instead, one way we can spot the strongest relationships between features is by using a PairGrid.

The PairGrid below plots some of the features we have already looked at against each other in both scatterplot and regplot ('line of best fit') form. On the diagonal we have a KDE plot, which can be described as a 'smoothed out histogram', which lets us look at how the distribution of each variable varies by publication.

In [50]:

pg = sns.PairGrid(articles, hue='Publication', palette=pub\_palette, diag\_sharey=**False**,

vars=['Text Length', 'Title Length', 'Comp\_Sentiment', 'Quotes per 1000 words', 'Said density per 1000 words'])

pg.map\_diag(sns.kdeplot)

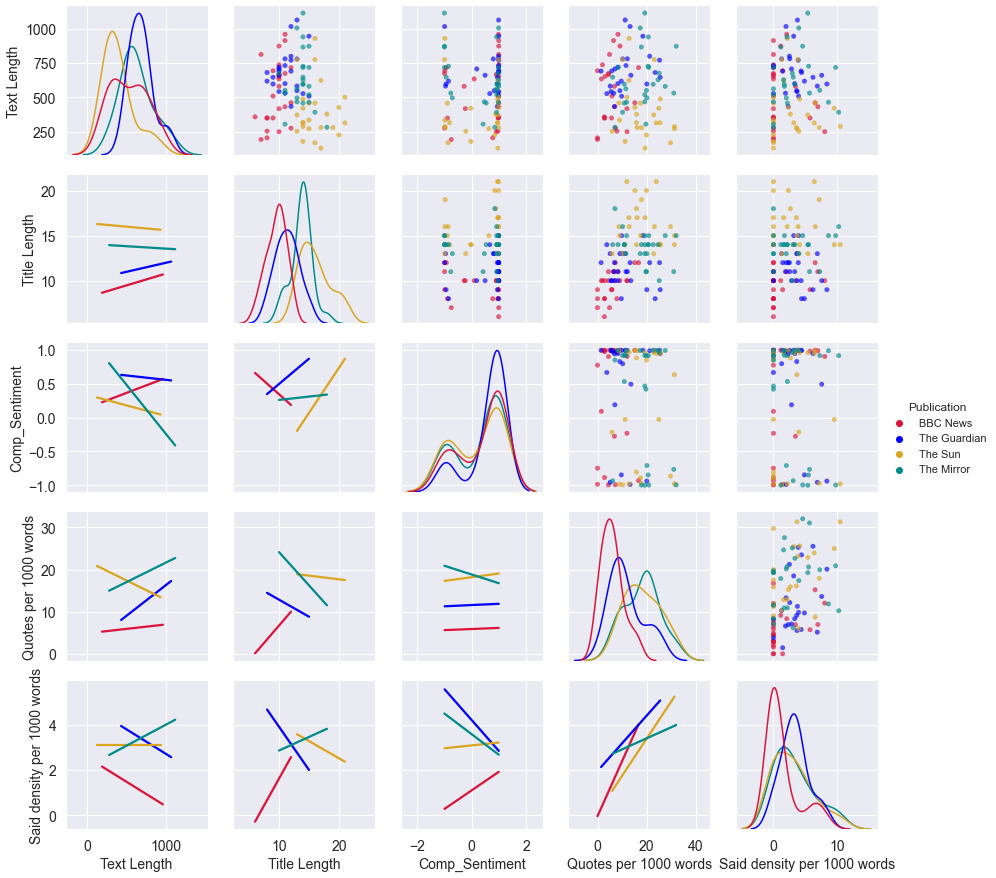
pg.map\_lower(sns.regplot, ci=**False**, scatter=**False**)

pg.map\_upper(sns.scatterplot, alpha=0.65, s=20, edgecolor=**None**)

pg.add\_legend()

Out[50]:

<seaborn.axisgrid.PairGrid at 0x1bbf7c0b160>



We can see some interesting trends emerging which we can explore further in the second part of the project:

* Said density and number of quotes per 1000 words seem strongly correlated on their regplot. However, when looking at the correlation of these counts with other variables (text and title length, sentiment), they often have different relationships with the variables.
* Most of the time publications have their own trends with very different gradients to one another, rather than collective trends. We should continue taking each publication into account individually when looking at relationships in order to extract insightful information.
* The KDE plot for composite sentiment shows all the publications have a similar distribution. The scatterplots demonstrate that composite sentiment tends to be either very positive or very negative, with few values in between, indicating perhaps the sentiment analysis method we have used is not the most suitable for long form text.

**6 Summary**

**6.1 Conclusions**

Through the exploratory data analysis, we have been able to see that each editor has different priorities as to what and who they discuss within their articles. We can see that there are relationships and trends within the data which make this dataset viable for further exploration.

Within the project, it would be useful to explore sentiment analysis tools to find the most suitable method for long form text, as the current method used is likely not giving us the most useful information.

**6.2 Summary of prepared data**

**6.2.1 Articles dataframe:**

In [51]:

articles.columns

Out[51]:

Index(['URL\_ID', 'Publication', 'Author', 'Title', 'Text', 'Text Length',

'Title Length', 'Filtered text', 'Filtered title', 'Lemmatized text',

'Lemmatized title', 'Quotes', 'Quotes per 1000 words',

'Said density per 1000 words', 'Neg\_Sentiment', 'Neu\_Sentiment',

'Pos\_Sentiment', 'Comp\_Sentiment'],

dtype='object')

For the articles dataframe:

* Filtered and lemmatized text and titles will be used to explore topics and politicians discussed in each article
* Text will be used for TF-IDF
* Text (unfiltered) can be used for politeness and affectivity metrics
* Features will be used for comparative/relational analysis

**6.2.2 Editors dataframe:**

In [52]:

editors.columns

Out[52]:

Index(['Editor', 'Publication', 'Full corpus', 'Title corpus',

'Filtered corpus', 'Filtered title corpus', 'Lemmatized corpus',

'Lemmatized title corpus'],

dtype='object')

For the editors dataframe:

* Filtered and lemmatized corpora and title corpora will be used to explore which topics and politicians each editor focuses on most overall
* Full corpus (unfiltered) can be used for politeness and affectivity metrics

**7 References and Resources**

**7.1 References**

[1] A. Koivunen, A. Kanner, M. Janicki, A. Harju, J. Hokkanen, E. Mäkelä, "Emotive, evaluative, epistemic: A linguistic analysis of affectivity in news journalism", *Journalism*, vol. 22, issue 5, pp. 1190-1206, Feb 2021.  
[2] Ofcom. (2021, July 27). *News Consumption in the UK: 2021 report* [Online]. Available: <https://www.ofcom.org.uk/__data/assets/powerpoint_doc/0026/222479/news-consumption-in-the-uk-2021-report.pptx>  
[3] BBC. *Mission, values and public purposes* [Online]. Available: <https://www.bbc.com/aboutthebbc/governance/mission> [4] Wikipedia. *List of newspapers in the United Kingdom* [Online]. Available: <https://en.wikipedia.org/wiki/List_of_newspapers_in_the_United_Kingdom>  
[5] Wikipedia. *Political editor* [Online]. Available: <https://en.wikipedia.org/wiki/Political_editor>  
[6] BBC. (2021, Jan. 4). *Exceptions to copyright* [Online]. Available: <https://www.gov.uk/guidance/exceptions-to-copyright>  
[7] Y. Ma. (2020, Feb. 5). *NLP: How does NLTK.Vader Calculate Sentiment?* [Online]. Available: <https://medium.com/ro-codes/nlp-how-does-nltk-vader-calculate-sentiment-6c32d0f5046b>

**7.2 Resources used**

**Webscraping**

* Webscraping lecture and lab, Dr Sean McGrath
* P. de Wulf. (2021, Mar. 26). *Web Scraping with BeautifulSoup* [Online]. Available: <https://www.scrapingbee.com/blog/python-web-scraping-beautiful-soup/>
* Skytowner. (2021, Mar. 18). *Beautiful Soup | find\_all method* [Online]. Available: <https://www.skytowner.com/explore/beautiful_soup_find_all_method>

**Data cleaning and processing, feature generation**

* Natural Language Processing lab, Dr Sean McGrath
* GeeksforGeeks. (2020, Mar. 31). *Find frequency of each word in a string in Python* [Online]. Available: <https://www.geeksforgeeks.org/find-frequency-of-each-word-in-a-string-in-python/>
* K. Pykes. (2021, Mar. 17). *A Guide To Cleaning Text in Python* [Online]. Available: <https://towardsdatascience.com/a-guide-to-cleaning-text-in-python-943356ac86ca>
* GeeksforGeeks. (2021, May 31). *Removing stop words with NLTK in Python* [Online]. Available: <https://www.geeksforgeeks.org/removing-stop-words-nltk-python/>
* D. Singh. (2019, July 19). *Building Features from Text Data* [Online]. Available: <https://www.pluralsight.com/guides/building-features-from-text-data>
* TutorialsPoint. *Stemming & Lemmatization* [Online]. Available: <https://www.tutorialspoint.com/natural_language_toolkit/natural_language_toolkit_stemming_lemmatization.htm>

**Exploratory data analysis**

* Natural Language Processing lab, Dr Sean McGrath
* K. Fessel. (2020, June 29). *Introduction to Seaborn YouTube series* [Online]. Available: <https://www.youtube.com/playlist?list=PLtPIclEQf-3cG31dxSMZ8KTcDG7zYng1j>
* Matplotlib. *Customizing Matplotlib with style sheets and rcParams* [Online]. Available: <https://matplotlib.org/stable/tutorials/introductory/customizing.html>
* E. Marsja. (2019, Dec. 22). *How to Change the Size of Seaborn Plots* [Online]. Available: <https://www.marsja.se/how-to-change-size-of-seaborn-plot/>
* Zach. (2020, Nov. 22). *How to Change Legend Font Size in Matplotlib* [Online]. Available: <https://www.statology.org/matplotlib-legend-font-size/>
* D. Vu. (2019, Nov. 08). *Generating WordClouds in Python* [Online]. Available: <https://www.datacamp.com/community/tutorials/wordcloud-python>

In [ ]: