Week 5 Notebook

Exploratory Analysis with Text Data

Student Name: Muhammad Rakibul Islam

Date:

Assignment Due:

Instructor: Lisa Rhody

Methods of Text Analysis, Spring 2023

Note:

This week we are switching from the NLTK Book to using a companion Google Colab notebook for the O'Reilly series of books titled *Blueprints for Text Analytics Using Python*. This week, the reading is focused on the kinds of questions we can ask with text data, and so it's fitting for you to start learning what the preliminary process is for exploring a text corpus in order to begin thinking about waht questions could be asked of it.

However, this notebook is not exactly the same as the one from the Blueprints book, because interspersed in between exercises are questions to answer. So, my recommendation is that you go through the PDF of the chapter in the Commons library while you work through this notebook. At the same time, you'll want to respond to the additional questions with the help of the readings from this week.

Blueprints for Text Analysis Using Python

Jens Albrecht, Sidharth Ramachandran, Christian Winkler

If you like the book or the code examples here, please leave a friendly comment on Amazon.com!

Chapter 1:

Gaining Early Insights from Textual Data

Remark

The code in this notebook differs slightly from the printed book. For example we frequently use pretty print (`pp.pprint`) instead of `print` and `tqdm`'s `progress_apply` instead of Pandas' `apply`.

Moreover, several layout and formatting commands, like `figsize` to control figure size or subplot commands are removed in the book.

You may also find some lines marked with three hashes ###. Those are not in the book as well as they don't contribute to the concept.

All of this is done to simplify the code in the book and put the focus on the important parts instead of formatting.

Setup

Set directory locations. If working on Google Colab: copy files and install required libraries.

```
import sys, os
ON_COLAB = 'google.colab' in sys.modules

if ON_COLAB:
    GIT_ROOT = 'https://github.com/blueprints-for-text-analytics-python/blueprints-tex
    os.system(f'wget {GIT_ROOT}/ch01/setup.py')

%run -i setup.py
```

```
You are working on Google Colab.
Files will be downloaded to "/content".
Downloading required files ...
!wget -P /content https://github.com/blueprints-for-text-analytics-python/bluepr
!wget -P /content/data/un-general-debates https://github.com/blueprints-for-text
!wget -P /content/ch01 https://github.com/blueprints-for-text-analytics-python/b

Additional setup ...
!pip install -r ch01/requirements.txt
```

Load Python Settings

Common imports, defaults for formatting in Matplotlib, Pandas etc.

```
%run "$BASE_DIR/settings.py"

%reload_ext autoreload
```

```
%autoreload 2
%config InlineBackend.figure_format = 'png'
```

What you will learn and what we will build

Exploratory Data Analysis

Introducing the Dataset

```
pd.options.display.max_colwidth = 150 ###
file = "un-general-debates-blueprint.csv"
file = f"{BASE_DIR}/data/un-general-debates/un-general-debates-blueprint.csv.gz" ### r
df = pd.read_csv(file)
df.sample(2, random_state=53)
```

	session	year	country	country_name	speaker	position	tex
3871	51	1996	PER	Peru	Francisco Tudela Van Breughel Douglas	Minister for Foreign Affairs	At the outset, allo me,\nSir, to convey to yo and to this Assembly th greetings\nar congratulations of th Peruvian people, as we

Blueprint: Getting an Overview of the Data with Pandas

Calculating Summary Statistics for Columns

```
df['length'] = df['text'].str.len()

df.describe().T
```

	count	mean	std	min	25%	50%	75%	max

```
df[['country', 'speaker']].describe(include='O').T
```

	count	unique	top	freq
country	7507	199	ALB	46
speaker	7480	5428	Seyoum Mesfin	12

Checking for Missing Data

```
df.isna().sum()
```

```
session            0
year               0
country            0
country_name       0
speaker           27
position        3005
text               0
length             0
dtype: int64
```

```
df['speaker'].fillna('unkown', inplace=True)
```

```
df[df['speaker'].str.contains('Bush')]['speaker'].value_counts()
```

```
George W. Bush        4
Mr. George W. Bush    2
Bush                  1
George Bush           1
Mr. George W Bush     1
Name: speaker, dtype: int64
```

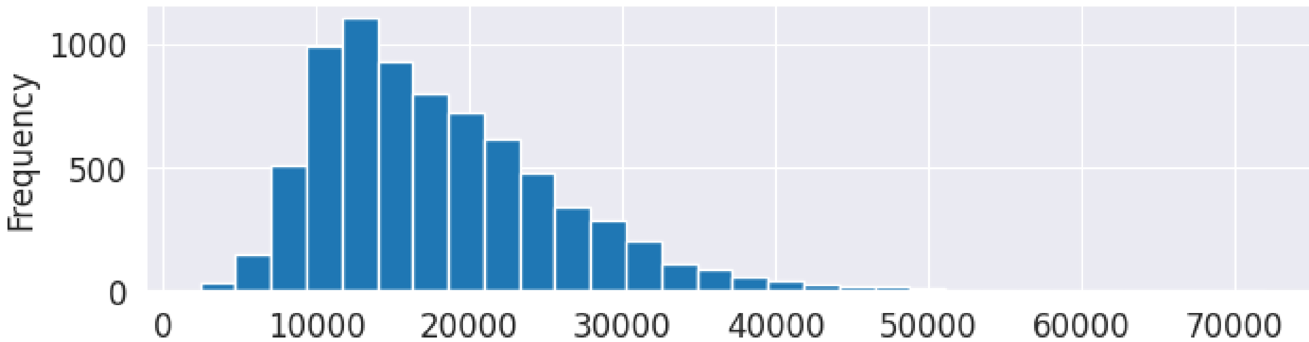
Plotting Value Distributions

```
df['length'].plot(kind='box', vert=False, figsize=(8, 1))
```

```
<AxesSubplot:>
```

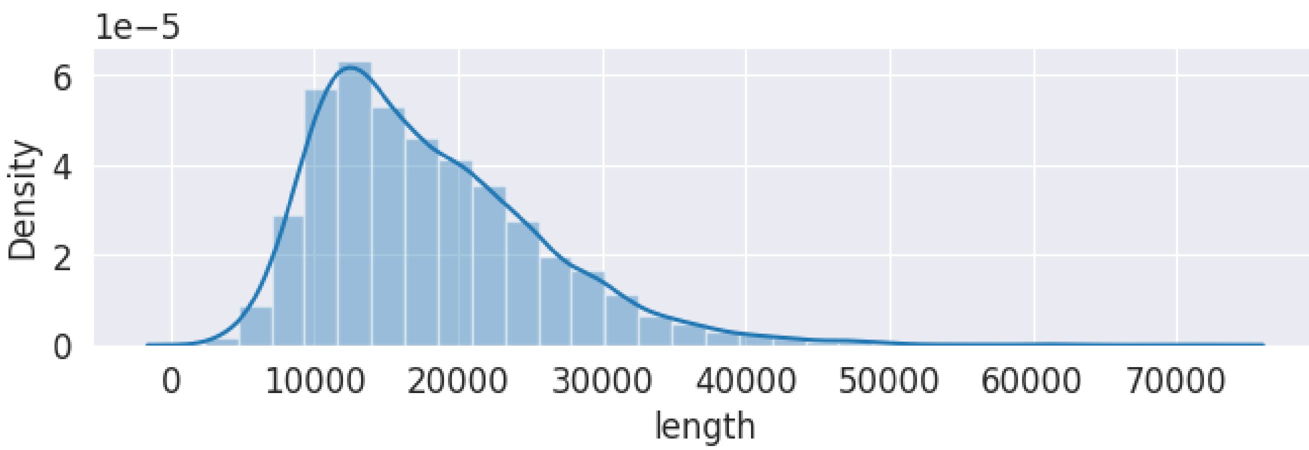
```
df['length'].plot(kind='hist', bins=30, figsize=(8,2))
```

```
<AxesSubplot:ylabel='Frequency'>
```




```
# Not in book: seaborn plot with gaussian kernel density estimate
import seaborn as sns

plt.figure(figsize=(8, 2))
sns.distplot(df['length'], bins=30, kde=True);
```



Question 1:

In this section, you are learning how to get a general insight into the dataset by looking at its length, finding missing data types, and evaluating things that are duplicate values because there are multiple references or multiple forms of textual representation. What is the benefit to computational text analysis of blueprint or (as Salganick calls them, "readymades")?

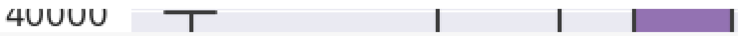
Response:

Blueprints or "readymades", as Salganick calls them, is beneficial because it provides the framework for a standardized and a reusable technique for text analysis that can help save time as well as make it easy to compare results of different projects.

Comparing Value Distributions across Categories

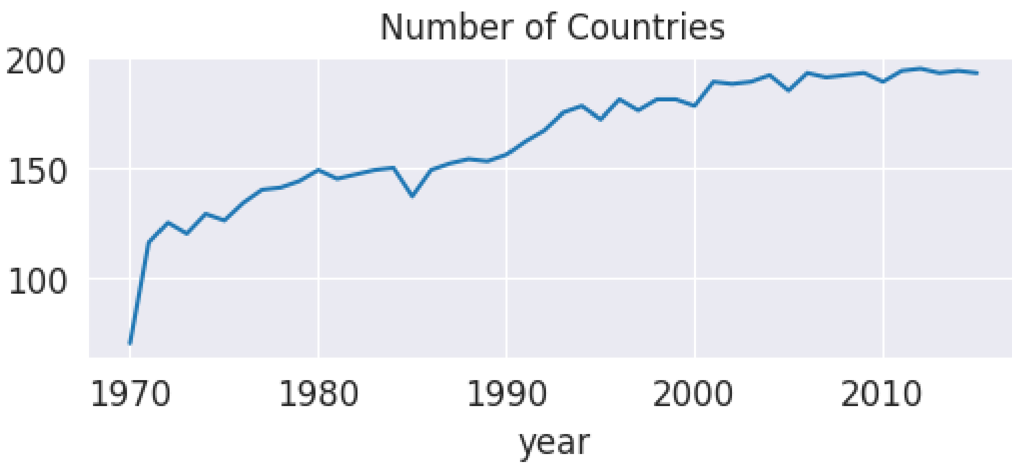
```
where = df['country'].isin(['USA', 'FRA', 'GBR', 'CHN', 'RUS'])
g = sns.catplot(data=df[where], x="country", y="length", kind='box')
g.fig.set_size_inches(4, 3) ###
g.fig.set_dpi(100) ###
g = sns.catplot(data=df[where], x="country", y="length", kind='violin')
g.fig.set_size_inches(4, 3) ###
g.fig.set_dpi(100) ###
```

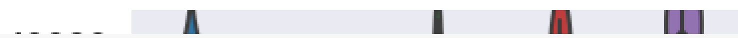
Visualizing Developments over Time



```
df.groupby('year').size().plot(title="Number of Countries", figsize=(6,2))
```

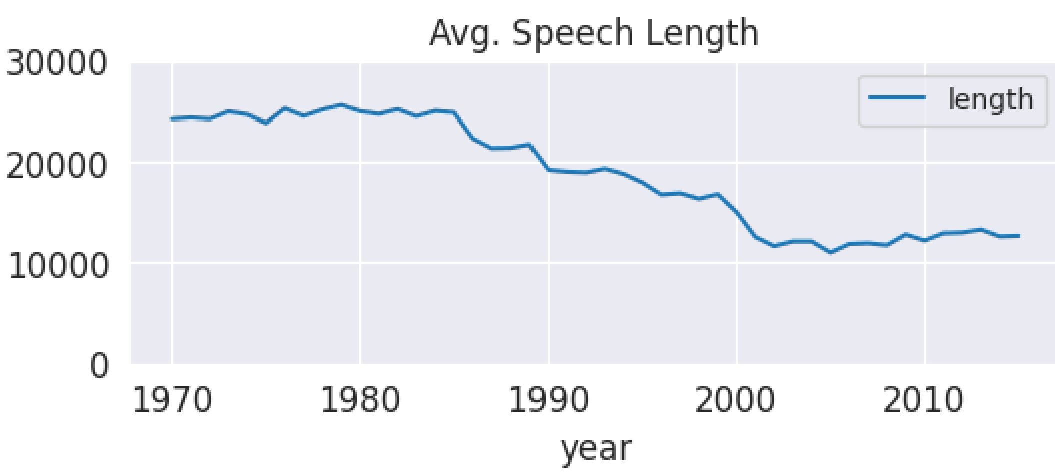
```
<AxesSubplot:title={'center':'Number of Countries'}, xlabel='year'>
```





```
df.groupby('year').agg({'length': 'mean'}) \
  .plot(title="Avg. Speech Length", ylim=(0,30000), figsize=(6,2))
```

```
<AxesSubplot:title={'center':'Avg. Speech Length'}, xlabel='year'>
```



Question 2:

What kinds of questions can you ask when you can visualize distributions or change? What kind of data do you need in order for these visualizations to work? How does access to or limitations of data such as categorical labels or years change the kind of questions you can ask?

Response:

Visualizing distribution or change can help answer questions around understanding the trend and the relationship between variables in the data. We can also see if there are any particular outliers and other anomalies in the data and ask ourselves why so.

For distributions, continuous numerical data makes the most sense when it particularly comes to questions like trends and relationships.

Limitations of data such as categorical labels or years would hinder the aspects of asking questions particular when it comes to studying the trends and relationships over time and to visualize the changes (if any) during the period of study.

Blueprint: Building a Simple Text Preprocessing Pipeline

Tokenization with Regular Expressions

```
import regex as re

def tokenize(text):
    return re.findall(r'[\w-]*\p{L}[\w-]*', text)
```

```
text = "Let's defeat SARS-CoV-2 together in 2020!"
tokens = tokenize(text)
print("|".join(tokens))
```

Treating Stop Words

```
import nltk
# not in book: make sure stop words are available
nltk.download('stopwords')
```

```
import nltk

stopwords = set(nltk.corpus.stopwords.words('english'))
```

```
def remove_stop(tokens):
    return [t for t in tokens if t.lower() not in stopwords]
```

```
include_stopwords = {'dear', 'regards', 'must', 'would', 'also'}
exclude_stopwords = {'against'}
```

```
stopwords |= include_stopwords
stopwords -= exclude_stopwords
```

Question 3:

In *Blueprints*, the authors are particularly careful to say that the way to remove stopwords isn't always a one-size-fits-all solution. Taking into consideration, DÍgnazio and Klein's chapter on "rational, scientific, objective viewpoints," when does it make sense to use standard stop word lists, and when should stop word lists be adjusted?

Response:

It would make sense to use standard stop word lists when conducting a rather generalized study of the overall content of text(s).

On the other hand a standard stop word list might not make sense when the particular context of the text being studies calls for it. Like in a particular domain of work, certain words that would otherwise be stop words might be relevant for analysis in that respective context. In that case, building an adjusted, custom stop word list would be more effective.

An adjusted stop word list can also make sense if the analysis being conducted is on a text of a different language as separate languages can have different words that add meaning which it does not in English.

Processing a Pipeline with one Line of Code

```
pipeline = [str.lower, tokenize, remove_stop]

def prepare(text, pipeline):
    tokens = text
    for transform in pipeline:
        tokens = transform(tokens)
    return tokens
```

```
df['tokens'] = df['text'].progress_apply(prepare, pipeline=pipeline)
```

```
df['num_tokens'] = df['tokens'].progress_map(len)
```

Blueprints for Word Frequency Analysis

Blueprint: Counting Words with a Counter

```
from collections import Counter

tokens = tokenize("She likes my cats and my cats like my sofa.")

counter = Counter(tokens)
print(counter)
```

```
more_tokens = tokenize("She likes dogs and cats.")
counter.update(more_tokens)
print(counter)
```

```
counter = Counter()

_ = df['tokens'].map(counter.update)
```

```
pp.pprint(counter.most_common(5))
```

```
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].progress_map(update)

    # transform counter into data frame
    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)
```

```
freq_df = count_words(df)
freq_df.head(5)
```

```
# top words with 10+ characters
count_words(df, column='text',
            preprocess=lambda text: re.findall(r"\w{10,}", text)).head(5)
```

Blueprint: Creating a Frequency Diagram

```
ax = freq_df.head(15).plot(kind='barh', width=0.95, figsize=(8,3))
ax.invert_yaxis()
ax.set(xlabel='Frequency', ylabel='Token', title='Top Words')
```

Blueprint: Creating Word Clouds

```
from wordcloud import WordCloud
from matplotlib import pyplot as plt

text = df.query("year==2015 and country=='USA'")['text'].values[0]

plt.figure(figsize=(4, 2)) ###
wc = WordCloud(max_words=100, stopwords=stopwords)
wc.generate(text)
plt.imshow(wc, interpolation='bilinear')
plt.axis("off")
```

```
from wordcloud import WordCloud ###
from collections import Counter ###

def wordcloud(word_freq, title=None, max_words=200, stopwords=None):

    wc = WordCloud(width=800, height=400,
                   background_color= "black", colormap="Paired",
                   max_font_size=150, max_words=max_words)

    # convert data frame into dict
    if type(word_freq) == pd.Series:
        counter = Counter(word_freq.fillna(0).to_dict())
    else:
        counter = word_freq

    # filter stop words in frequency counter
    if stopwords is not None:
        counter = {token:freq for (token, freq) in counter.items()
                              if token not in stopwords}
    wc.generate_from_frequencies(counter)

    plt.title(title)
```

```
    plt.imshow(wc, interpolation='bilinear')
    plt.axis("off")
```

```
freq_2015_df = count_words(df[df['year']==2015])
plt.figure(figsize=(12,4))
plt.subplot(1,2,1)###
wordcloud(freq_2015_df['freq'], max_words=100)
plt.subplot(1,2,2)###
wordcloud(freq_2015_df['freq'], max_words=100, stopwords=freq_df.head(50).index)
#plt.tight_layout()###
```

Question 4:

Counting seems to be a significant issue for feminist scholars. On the one hand, being counted is a political act (as we remember from the readings in Week 4 as well as in Mandell's chapter that we read for this week). How do the kinds of counting that we can do with text analysis create challenges for the feminist scholar according to Laura Mandell? How do you see what she describes at play in these blueprint text processes?

Response:

We have to first realize that tha counting we are doing with text analysis is being done through certain digital tools that we are using. Now these digital tools themselves can inherently reflect the biases that already exist in the texts that they are trained on. Even the development of these tools are being done by humans which can by itself allow the human bias to contaminate the process.

For example, a counting method trained on text written by mostly cis, white men might not be able to accurately analyze text written by women and members of others marginalized populations. As such, the conclusions drawn will themselves incorporate the bias.

Counting can also consider language at its rather strict quantitative value, completely missing the qualitative element that might hide a more subtle form of bias or discrimination within words themselves.

Now the points mentioned above can very well be applied to the context of these blueprint text processes as the standardized models of these blueprints can carry on the bias and discrimination that the text they are utilized and tested for carry as well the ones of the human(s) writing the particular blueprint and this pose challenges to the feminist scholar.

Blueprint: Ranking with TF-IDF

```
def compute_idf(df, column='tokens', preprocess=None, min_df=2):

    def update(doc):
        tokens = doc if preprocess is None else preprocess(doc)
        counter.update(set(tokens))

    # count tokens
    counter = Counter()
    df[column].progress_map(update)

    # create data frame and compute idf
    idf_df = pd.DataFrame.from_dict(counter, orient='index', columns=['df'])
    idf_df = idf_df.query('df >= @min_df')
    idf_df['idf'] = np.log(len(df)/idf_df['df'])+0.1
    idf_df.index.name = 'token'
    return idf_df
```

```
idf_df = compute_idf(df)
```

```
# Not in book: sample of IDF values
# high IDF means rare (interesting) term
idf_df.sample(5)
```

```
freq_df['tfidf'] = freq_df['freq'] * idf_df['idf']
```

```
# not in book: for more data: joining is faster
freq_df = freq_df.join(idf_df)
freq_df['tfidf'] = freq_df['freq'] * freq_df['idf']
```

```
freq_1970 = count_words(df[df['year'] == 1970])
freq_2015 = count_words(df[df['year'] == 2015])

freq_1970['tfidf'] = freq_1970['freq'] * idf_df['idf']
freq_2015['tfidf'] = freq_2015['freq'] * idf_df['idf']

plt.figure(figsize=(12,6)) ###
#wordcloud(freq_df['freq'], title='All years', subplot=(1,3,1))
plt.subplot(2,2,1)###
wordcloud(freq_1970['freq'], title='1970 - TF',
          stopwords=['twenty-fifth', 'twenty-five'])
plt.subplot(2,2,2)###
wordcloud(freq_2015['freq'], title='2015 - TF',
          stopwords=['seventieth'])
```

```
plt.subplot(2,2,3)###
wordcloud(freq_1970['tfidf'], title='1970 - TF-IDF',
          stopwords=['twenty-fifth', 'twenty-five', 'twenty', 'fifth'])
plt.subplot(2,2,4)###
wordcloud(freq_2015['tfidf'], title='2015 - TF-IDF',
          stopwords=['seventieth'])
```

Question 5:

What is the attraction to scholars like Ted Underwood in the ability to make use of classification analyses like TF-IDF? What kinds of questions could be asked of a text corpora using an unsupervised classification study like the one above?

Response:

Scholars like Ted Underwood like to use classification analyses like TF-IDF for the ability it provides to analyze and identify particular trends and patterns in large corpus of documents.

To understand better, using the context of TF-IDF, it helps identity the importance of particular or set of words in a document analyzing the frequency of their apperance in that particular document (TF: Term Frequency) while identifying how rare they are in the overall corpus of documents (IDF: Inverse Document Frequency). Now this can help scholars identify the words / set of words that most likely indicators of a particular category of text (whether it is genre, author, writing style, writing period etc.)

Unsupervised classification study like the one above can be used to ask various questions like what set of texts within a corpus of texts share similarities like certain words or phrases that are more common which can be indicative of its genre, writing style, author etc.

If a time factor (like year of writing) is associated, changes in language use can be identified by clustering text from particular time periods and studying them.

In summary, it can help identify trends, patterns and relationships that might otherwise not be easy to identify given the breadth of large corpus of texts.

Blueprint: Finding a Keyword in Context (KWIC)

Note: textacy's API had major changes from version 0.10.1 (as used in the book) to 0.11. Here, `textacy.text_utils.KWIC` became `textacy.extract.kwic.keyword_in_context` (see textacy documentation).

```
import textacy

if textacy.__version__ < '0.11': # as in printed book
    from textacy.text_utils import KWIC

else: # for textacy 0.11.x
    from textacy.extract.kwic import keyword_in_context

    def KWIC(*args, **kwargs):
        # call keyword_in_context with all params except 'print_only'
        return keyword_in_context(*args,
                           **{kw: arg for kw, arg in kwargs.items()
                            if kw != 'print_only'})
```

```
def kwic(doc_series, keyword, window=35, print_samples=5):

    def add_kwic(text):
        kwic_list.extend(KWIC(text, keyword, ignore_case=True,
                              window_width=window, print_only=False))

    kwic_list = []
    doc_series.progress_map(add_kwic)

    if print_samples is None or print_samples==0:
        return kwic_list
    else:
        k = min(print_samples, len(kwic_list))
        print(f"{k} random samples out of {len(kwic_list)} " + \
              f"contexts for '{keyword}':")
        for sample in random.sample(list(kwic_list), k):
            print(re.sub(r'[\n\t]', ' ', sample[0])+'  '+ \
                  sample[1]+'  '+\
                  re.sub(r'[\n\t]', ' ', sample[2]))
```

```
random.seed(22) ###
kwic(df[df['year'] == 2015]['text'], 'sdgs', print_samples=5)
```

Blueprint: Analyzing N-Grams

```
def ngrams(tokens, n=2, sep=' '):
    return [sep.join(ngram) for ngram in zip(*[tokens[i:] for i in range(n)])]

text = "the visible manifestation of the global climate change"
tokens = tokenize(text)
print("|".join(ngrams(tokens, 2)))
```

```
def ngrams(tokens, n=2, sep=' ', stopwords=set()):
    return [sep.join(ngram) for ngram in zip(*[tokens[i:] for i in range(n)])
            if len([t for t in ngram if t in stopwords])==0]

print("Bigrams:", "|".join(ngrams(tokens, 2, stopwords=stopwords)))
print("Trigrams:", "|".join(ngrams(tokens, 3, stopwords=stopwords)))
```

```
df['bigrams'] = df['text'].progress_apply(prepare, pipeline=[str.lower, tokenize]) \
                          .progress_apply(ngrams, n=2, stopwords=stopwords)

count_words(df, 'bigrams').head(5)
```

```
idf_df = compute_idf(df) ### re-initialize to be safe
# concatenate existing IDF data frame with bigram IDFs
idf_df = pd.concat([idf_df, compute_idf(df, 'bigrams', min_df=10)])

freq_df = count_words(df[df['year'] == 2015], 'bigrams')
freq_df['tfidf'] = freq_df['freq'] * idf_df['idf']
```

```
plt.figure(figsize=(12,6)) ###
plt.subplot(1,2,1) ###
wordcloud(freq_df['tfidf'], title='all bigrams', max_words=50)

plt.subplot(1,2,2) ###
# plt.tight_layout() ###
where = freq_df.index.str.contains('climate')
wordcloud(freq_df[where]['freq'], title='"climate" bigrams', max_words=50)
```

Blueprint: Comparing Frequencies across Time-Intervals and Categories

Creating Frequency Timelines

```
def count_keywords(tokens, keywords):
    tokens = [t for t in tokens if t in keywords]
    counter = Counter(tokens)
    return [counter.get(k, 0) for k in keywords]
```

```
keywords = ['nuclear', 'terrorism', 'climate', 'freedom']
tokens = ['nuclear', 'climate', 'climate', 'freedom', 'climate', 'freedom']

print(count_keywords(tokens, keywords))
```

```
def count_keywords_by(df, by, keywords, column='tokens'):

    df = df.reset_index(drop=True) # if the supplied dataframe has gaps in the index
    freq_matrix = df[column].progress_apply(count_keywords, keywords=keywords)
    freq_df = pd.DataFrame.from_records(freq_matrix, columns=keywords)
    freq_df[by] = df[by] # copy the grouping column(s)

    return freq_df.groupby(by=by).sum().sort_values(by)
```

```
freq_df = count_keywords_by(df, by='year', keywords=keywords)
```

```
pd.options.display.max_rows = 4
```

```
pd.options.display.max_rows = 60
```

```
freq_df.plot(kind='line', figsize=(8, 3))
```

```
random.seed(23) ###
# analyzing mentions of 'climate' before 1980
kwic(df.query('year < 1980')['text'], 'climate', window=35, print_samples=5)
```

Creating Frequency Heat Maps

```
keywords = ['terrorism', 'terrorist', 'nuclear', 'war', 'oil',
            'syria', 'syrian', 'refugees', 'migration', 'peacekeeping',
            'humanitarian', 'climate', 'change', 'sustainable', 'sdgs']

freq_df = count_keywords_by(df, by='year', keywords=keywords)

# compute relative frequencies based on total number of tokens per year
freq_df = freq_df.div(df.groupby('year')['num_tokens'].sum(), axis=0)
# apply square root as sublinear filter for better contrast
freq_df = freq_df.apply(np.sqrt)

plt.figure(figsize=(10, 3)) ###
sns.set(font_scale=1) ###
sns.heatmap(data=freq_df.T,
            xticklabels=True, yticklabels=True, cbar=False, cmap="Reds")
sns.set(font_scale=1) ###
```

Closing Remarks

Question 6:

What is the value of exploratory analysis in computational text analysis? How can it be helpful? In what ways is it insufficient?

Response:

Exploratory analysis is useful in computational text analysis because it can better help understand the data being analyzed. It can help identify trends, relationships and patterns. Not only that but it can help in the data cleaning process finding missing values, duplicates, outliers etc.

But it has its limitations. We have to understand that exploratory analysis is a basic, initial analytical tool and lacks the strength of analysis required to draw a deeper understanding of the text which will usually need further analysis.

It is also important to understand that exploratory analysis is being done by a researcher who themselves can be subjective and biased because everyone has some sort of a bias themselves. So it might not provide a definitive conclusion which can only be drawn after further, more complex analysis is done and aware that you might introduce your own biases and work to counteract them.

Colab paid products - Cancel contracts here