A02_TextProc_Yash_Potdar

February 28, 2020

1 Sentiment Analysis on American Presidents

1.1 Yash Potdar

1.1.1 Data Science UCSD '23

- Question: How does sentiment analysis perceive Franklin D. Roosevelt, Andrew Jackson, and Donald J. Trump's biographies on Wikipedia?
- Background: Throughout American history, there have been presidents that have been widely regarded and praised, loathed by the public. However, there have also been presidents who have controversial images due to a mixture of good and bad actions. Although we, as humans, can form our own opinions about these presidents, can sentiment analysis using Vader Lexicon confirm our opinions?
  - Franklin Delano Roosevelt, the 32nd President, has generally been praised due to his progressive New Deal policies that uplifted the country out of the Great Depression. He was the only president elected for four terms.
  - Andrew Jackson, the 7th President, has a controversial, mixed image. On one hand, Jackson strongly supported slavery and demanded the forced removal of Native Americans on the Trail of Tears, displacing hundreds of thousands of Native Americans. On the other hand, he was known as a common man and destroyed the Second Bank of the US, which would protect individual liberties.
  - Donald Trump, the 45th President, has a negative image due to his reversals of many progressive actions of prior presidents, and his general misogynistic and racist behavior.
- **Part 1**: Determine the Average Compound Score for each President's Wiki Page
- **Part 2**: Visualizing the Most Extreme Words for the Presidents

1.2 Part 1: Average Compound Sentiment Score for each President

1.2.1 Step 1: Import Relevant Packages

[1]:
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from nltk.sentiment.vader import SentimentIntensityAnalyzer
import requests
import re
```

```python
from urllib.parse import urlparse
import urllib.robotparser
from bs4 import BeautifulSoup
from wordcloud import WordCloud, STOPWORDS

# This code checks the robots.txt file
def canFetch(url):

    parsed_uri = urlparse(url)
    domain = '{uri.scheme}://{uri.netloc}/'.format(uri=parsed_uri)

    rp = urllib.robotparser.RobotFileParser()
    rp.set_url(domain + "/robots.txt")
    try:
        rp.read()
        canFetchBool = rp.can_fetch("*", url)
    except:
        canFetchBool = None

    return canFetchBool
```

1.2.2 Step 2: Permissions for Web Scraping

Before I can retrieve text from the three Wikipedia pages, I must first send requests to the three Wikipedia websites, which are represented in the `sites` list. The package I am using to communicate with the websites is `requests`, from which I invoke the `get` function on each site. This gives the HTTP status code. A 200 response returned from the function call means the page was found and the request succeeded. After checking the HTTP status code for each website, I also used the `canFetch` method, which lets me know whether there is permission to use a robot to access the site. All the sites yielded True after calling this function, so web scraping was permitted is a green light, and means that web scraping is permitted.

[2]:
```python
sites = ["https://en.wikipedia.org/wiki/Franklin_D._Roosevelt", "https://en.
↪wikipedia.org/wiki/Andrew_Jackson",
         "https://en.wikipedia.org/wiki/Donald_Trump"]
for site in sites:
    print('* Site: {0}\n\tHTTP Response: {1}\n\tWeb Scrape Permitted: {2}'\
          .format(site, requests.get(site), canFetch(site)))
```

```
* Site: https://en.wikipedia.org/wiki/Franklin_D._Roosevelt
        HTTP Response: <Response [200]>
        Web Scrape Permitted: True
* Site: https://en.wikipedia.org/wiki/Andrew_Jackson
        HTTP Response: <Response [200]>
        Web Scrape Permitted: True
* Site: https://en.wikipedia.org/wiki/Donald_Trump
```

```
HTTP Response: <Response [200]>
Web Scrape Permitted: True
```

1.2.3 Step 3: Retrieving Data from the 3 Wikipedia pages

Next, I accessed each president's corresponding URL from the `sites` list from Step 1. For example, FDR would correspond to index 0 and Trump would correspond to index 2. Then I retrieved the unicode response from each wiki page using the text property of the response from the GET HTTP method. I stored these in each president's `urlText` string. FDR's unicode text from his Wiki is below as an example. This is nowhere near the required form for sentiment analysis, so further cleaning is required.

[3]:
```python
def get_url_text(url_str):
    return requests.get(url_str, timeout = 5).text

# FDR: Franklin Delano Roosevelt
fdr_url = sites[0]
fdr_urlText = get_url_text(fdr_url)

# AJ: Andrew Jackson
aj_url = sites[1]
aj_urlText = get_url_text(aj_url)

# DJT: Donald Trump
djt_url = sites[2]
djt_urlText = get_url_text(djt_url)

#fdr_urlText
```

1.2.4 Step 4: Cleaning Text Using BeautifulSoup

Next, to clean the HTML and convert it into a more readable form, I used the BeautifulSoup package. For each president, I used the HTML parser in BeautifulSoup to extract the page's content. Finally, I filtered all irrelevant HTML tags by using the `find_all` method of `page_content`, a BeautifulSoup object. By focusing on only the paragraphs, we are one step closer to sentiment analysis. To verify that we indeed have paragraphs, the nested for loop prints all the paragraphs on Trump's Wiki page.

[4]:
```python
def get_page_paragraphs(url_text_str):
    soup = BeautifulSoup(url_text_str, 'html.parser')
    return soup.find_all("p")

# FDR: Franklin Delano Roosevelt
fdr_paragraphs = get_page_paragraphs(fdr_urlText)

# AJ: Andrew Jackson
```

```
aj_paragraphs = get_page_paragraphs(aj_urlText)

# DJT: Donald Trump
djt_paragraphs = get_page_paragraphs(djt_urlText)

#for paragraph in djt_paragraphs:
#     print(paragraph.text)
```

1.2.5 Step 5: Create List with All Cleaned Sentences from Wiki

Next, for each president, I made a list that would contain all the paragraphs. Then, for every paragraph, I used Regular Expressions to remove items from brackets and parentheses. This is due to the fact that Wikipedia has references in brackets and parentheses which are irrelevant to my analysis. Next, I joined all the paragraphs in the list into one string so they could be easier cleaned. I then removed commas and quotation marks since they would be attached to words. I split the text with a period delimiter to have a list with all the sentences in the Wiki page. Finally, I used a list comprehension to filter out "sentences" that were not actually sentences because they were acronyms split by the function call (ex: U.S. would be split into 2 elements in `all_sentences_fdr`).

```
[5]: regex_brackets = "\[.*\]|\s-\s.*"
regex_parentheses = "\(.*\)|\s-\s.*"

def get_all_text_cleaned(paragraphs):
    lst_paragraphs = []
    for par in paragraphs:
        lst_paragraphs.append(par.text)
    for item in range(len(lst_paragraphs)):
        lst_paragraphs[item] = lst_paragraphs[item].strip('\n').lower().
 ↪replace(',', '').replace('"', '')
        lst_paragraphs[item] = re.sub(regex_brackets, "", lst_paragraphs[item])
        lst_paragraphs[item] = re.sub(regex_parentheses, "",␣
 ↪lst_paragraphs[item])
    all_cleaned_text = " ".join(lst_paragraphs).strip()
    return all_cleaned_text

def get_all_sentences(all_text_str):
    sentences = all_text_str.split('.')
    sentences = [s for s in sentences if len(s.split()) > 1]
    return sentences


# FDR: Franklin Delano Roosevelt
all_text_fdr = get_all_text_cleaned(fdr_paragraphs)
all_sentences_fdr = get_all_sentences(all_text_fdr)

# AJ: Andrew Jackson
```

```
all_text_aj = get_all_text_cleaned(aj_paragraphs)
all_sentences_aj = get_all_sentences(all_text_aj)

# DJT: Donald Trump
all_text_djt = get_all_text_cleaned(djt_paragraphs)
all_sentences_djt = get_all_sentences(all_text_djt)

all_sentences_aj[:15]
```

[5]:
```
['andrew jackson  was an american soldier and statesman who served as the
seventh president of the united states from 1829 to 1837',
 ' before being elected to the presidency jackson gained fame as a general in
the united states army and served in both houses of the u',
 ' as president jackson sought to advance the rights of the common man and to
preserve the union',
 ' born in the colonial carolinas to a scotch-irish family in the decade before
the american revolutionary war jackson became a frontier lawyer and married
rachel donelson robards',
 ' he served briefly in the united states house of representatives and the
united states senate representing tennessee',
 ' after resigning he served as a justice on the tennessee supreme court from
1798 until 1804',
 ' jackson purchased a property later known as the hermitage and became a
wealthy slaveowning planter',
 ' in 1801 he was appointed colonel of the tennessee militia and was elected its
commander the following year',
 ' he led troops during the creek war of 1813-1814 winning the battle of
horseshoe bend',
 ' the subsequent treaty of fort jackson required the creek surrender of vast
lands in present-day alabama and georgia',
 " in the concurrent war against the british jackson's victory in 1815 at the
battle of new orleans made him a national hero",
 ' jackson then led u',
 ' forces in the first seminole war which led to the annexation of florida from
spain',
 " jackson briefly served as florida's first territorial governor before
returning to the senate",
 ' he ran for president in 1824 winning a plurality of the popular and electoral
vote']
```

1.2.6 Step 6: Sentiment Analysis on Every Sentence

Next, I used the Vader `SentimentIntensityAnalyzer` package to analyze the sentiments of every sentence in each president's Wiki. I created lists for each president to store the results of the sentiment calculation.

```
[6]: sid = SentimentIntensityAnalyzer()

     def get_sentiment_lst(word_or_sentence_lst):
         """
         This function can be used to find the sentiments of either (1) a list of␣
      ↪all the sentences on the page \
         or (2) a list of all the words on the page
         """
         sentiments = []
         for phrase in word_or_sentence_lst:
             phrase_Sentiment = sid.polarity_scores(phrase)
             phrase_Sentiment['text'] = phrase
             sentiments.append(phrase_Sentiment)
         return sentiments


     # FDR: Franklin Delano Roosevelt
     fdr_Sentiments = get_sentiment_lst(all_sentences_fdr)

     # AJ: Andrew Jackson
     aj_Sentiments = get_sentiment_lst(all_sentences_aj)

     # DJT: Donald Trump
     djt_Sentiments = get_sentiment_lst(all_sentences_djt)

     djt_Sentiments[:5]
```

```
[6]: [{'neg': 0.0,
       'neu': 0.811,
       'pos': 0.189,
       'compound': 0.4215,
       'text': 'donald john trump  is the 45th and current president of the united
     states'},
      {'neg': 0.0,
       'neu': 1.0,
       'pos': 0.0,
       'compound': 0.0,
       'text': ' before entering politics he was a businessman and television
     personality'},
      {'neg': 0.0,
       'neu': 1.0,
       'pos': 0.0,
       'compound': 0.0,
       'text': " trump was born and raised in queens a borough of new york city and
     received a bachelor's degree in economics from the wharton school"},
      {'neg': 0.0,
       'neu': 1.0,
       'pos': 0.0,
       'compound': 0.0,
       'text': " he took charge of his family's real-estate business in 1971 renamed
     it the trump organization and expanded its operations from queens and brooklyn
     into manhattan"},
      {'neg': 0.0,
       'neu': 1.0,
       'pos': 0.0,
       'compound': 0.0,
       'text': ' the company built or renovated skyscrapers hotels casinos and golf
     courses'}]
```

1.2.7 Step 7: DataFrame for Each President

Next, I made a dataframe for each president, which would hold the results from the sentiment analysis on each sentence. The first 15 rows of the sentiment DataFrame for Jackson.

[7]:
```
fdrSentimentDf = pd.DataFrame(fdr_Sentiments)
ajSentimentDf = pd.DataFrame(aj_Sentiments)
djtSentimentDf = pd.DataFrame(djt_Sentiments)

ajSentimentDf.head(10)
```

[7]:
```
   compound    neg    neu    pos  \
0    0.4215  0.000  0.882  0.118
1    0.8074  0.000  0.707  0.293
2    0.0000  0.000  1.000  0.000
3   -0.5994  0.140  0.860  0.000
4    0.6808  0.000  0.728  0.272
5    0.7269  0.091  0.574  0.335
6    0.3612  0.000  0.828  0.172
7    0.0000  0.000  1.000  0.000
8   -0.4767  0.297  0.548  0.155
9    0.0000  0.000  1.000  0.000

                                                text
0  andrew jackson  was an american soldier and st…
1   before being elected to the presidency jackso…
2   as president jackson sought to advance the ri…
3   born in the colonial carolinas to a scotch-ir…
4   he served briefly in the united states house …
5   after resigning he served as a justice on the…
6   jackson purchased a property later known as t…
7   in 1801 he was appointed colonel of the tenne…
8   he led troops during the creek war of 1813-18…
9   the subsequent treaty of fort jackson require…
```

1.2.8 Step 8: Combined DataFrame for Average Compound Sentiment Scores

Next, I made a list that contained the averages of the `compound` column of each of the DataFrames from Step 8. Then I made a single DataFrame that contained the average compound sentiment scores for each president's Wiki.

```
[8]: avg_compound = [np.mean(pres_df['compound']) for pres_df in [fdrSentimentDf,␣
      ↪ajSentimentDf, djtSentimentDf]]
     df_compound = pd.DataFrame({'President':['FDR', 'Jackson', 'Trump'], 'Avg␣
      ↪Compound':avg_compound})
     df_compound
```

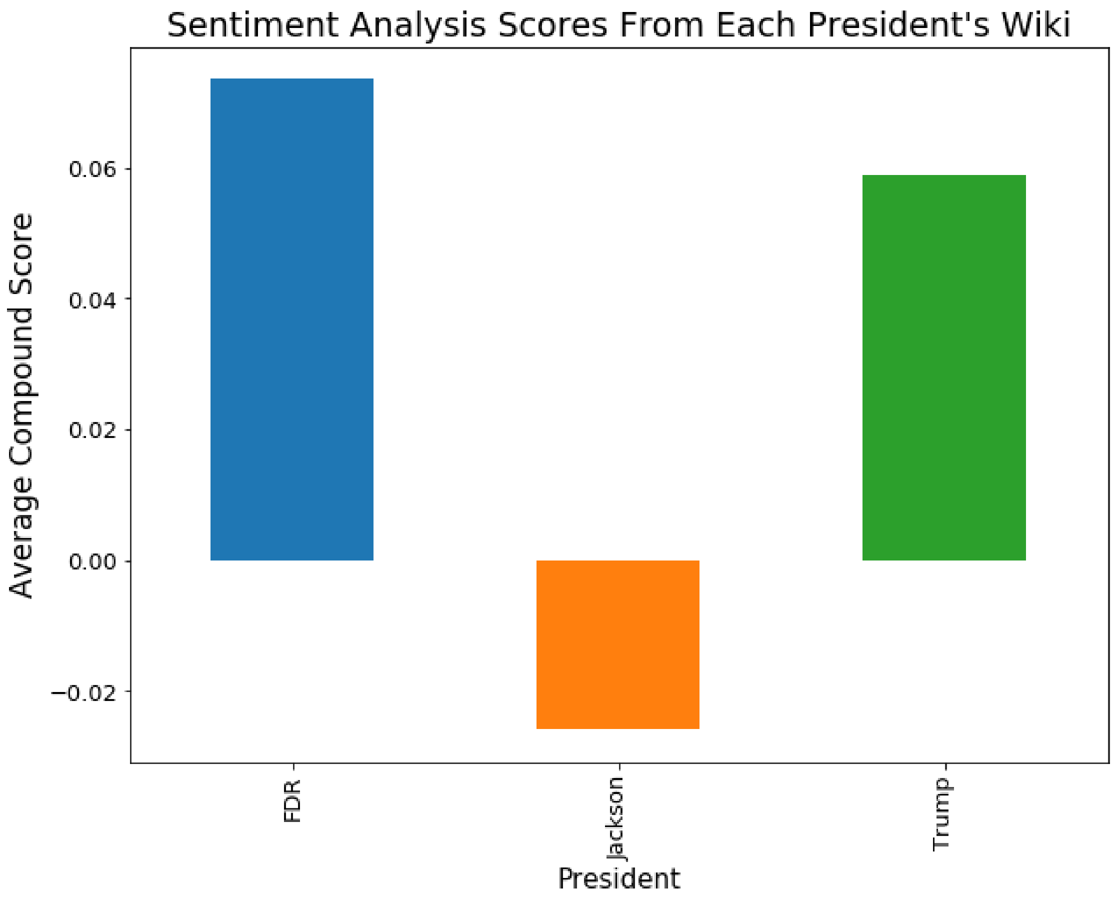
```
[8]:   President  Avg Compound
     0       FDR      0.073537
     1   Jackson     -0.025839
     2     Trump      0.058839
```

1.2.9 Step 9: Visualizing Average Compound Sentiment Scores for Each President

To better visualize the results, I made a bar graph of the Average Compound Sentiment Scores.

```
[9]: ax = df_compound.plot(x ='President', y ='Avg Compound', kind = 'bar', legend =␣
      ↪False, \
         figsize = [10, 30/4], fontsize = 13)
     ax.set_title('Sentiment Analysis Scores From Each President\'s Wiki',fontsize=␣
      ↪19)
     ax.set_xlabel('President', fontsize = 15)
     ax.set_ylabel('Average Compound Score', fontsize = 17)
```

```
[9]: Text(0, 0.5, 'Average Compound Score')
```



1.2.10 Conclusion for Part 1:

The results for Part 1 were not what I had expected, and it suggests that Vader lexicon was not able to correctly reflect the public's perception of the Presidents. I would expect FDR's compound score to be much higher than Trump's and Trump's to be a high negative percent. Jackson's was the only score that I expected to be around neutral since he had such a controversial legacy. The results may be due to the fact that this was from Wikipedia, so the sentences may be written in a more objective manner, which draws the sentiment closer to neutral.

1.3 Part 2: Visualizing the Most Polarizing Words

1.4 Steps 1-5 from Part 1

A key difference in the sentiment analysis from Part 1 and Part 2 is that in Part 1, analysis was done on sentences as a whole. In Part 2, since the goal is to make a word cloud of the most polarizing words, analysis will be done on individual words. For my analysis, I will be using `all_text_fdr`, all_text_aj, and all_text_djt from Step 5 of Part 1. These three variables represent strings that contain all the cleaned text from the Wikipedia page.

```
[10]: def get_individual_words(all_text_str):
          return all_text_str.replace('.','').split()

      all_words_fdr = get_individual_words(all_text_fdr)
      all_words_aj = get_individual_words(all_text_aj)
      all_words_djt = get_individual_words(all_text_djt)

      all_words_fdr[:15]
```

```
[10]: ['franklin',
       'delano',
       'roosevelt',
       'often',
       'referred',
       'to',
       'by',
       'his',
       'initials',
       'fdr',
       'was',
       'an',
       'american',
       'politician',
       'who']
```

1.5 Step 6: Sentiment Analysis on All the Text

In this step, I did sentiment analysis on every word in each president's Wiki page. An initial thought that came to mind was to filter out duplicates since it would decrease the runtime of the code. However, this would entirely defeat the purpose of the end goal of making a wordcloud, since a wordcloud uses frequencies of words to determine scaling. Moreover, I was debating whether to filter out stop words, but decided against it since I could use the STOPWORD package in WordCloud which would be more precise than determining if a word is a stop word just on the basis of length. Similar to Step 6 of Part 1, I created lists for each president to store the results of the sentiment calculation.

```
[11]: # FDR: Franklin Delano Roosevelt
      fdr_Sentiments2 = get_sentiment_lst(all_words_fdr)

      # AJ: Andrew Jackson
      aj_Sentiments2 = get_sentiment_lst(all_words_aj)

      # DJT: Donald Trump
      djt_Sentiments2 = get_sentiment_lst(all_words_djt)
```

```
djt_Sentiments2[:15]
```

```
[11]: [{'neg': 0.0, 'neu': 1.0, 'pos': 0.0, 'compound': 0.0, 'text': 'donald'},
       {'neg': 0.0, 'neu': 1.0, 'pos': 0.0, 'compound': 0.0, 'text': 'john'},
       {'neg': 0.0, 'neu': 1.0, 'pos': 0.0, 'compound': 0.0, 'text': 'trump'},
       {'neg': 0.0, 'neu': 1.0, 'pos': 0.0, 'compound': 0.0, 'text': 'is'},
       {'neg': 0.0, 'neu': 1.0, 'pos': 0.0, 'compound': 0.0, 'text': 'the'},
       {'neg': 0.0, 'neu': 1.0, 'pos': 0.0, 'compound': 0.0, 'text': '45th'},
       {'neg': 0.0, 'neu': 1.0, 'pos': 0.0, 'compound': 0.0, 'text': 'and'},
       {'neg': 0.0, 'neu': 1.0, 'pos': 0.0, 'compound': 0.0, 'text': 'current'},
       {'neg': 0.0, 'neu': 1.0, 'pos': 0.0, 'compound': 0.0, 'text': 'president'},
       {'neg': 0.0, 'neu': 1.0, 'pos': 0.0, 'compound': 0.0, 'text': 'of'},
       {'neg': 0.0, 'neu': 1.0, 'pos': 0.0, 'compound': 0.0, 'text': 'the'},
       {'neg': 0.0, 'neu': 0.0, 'pos': 1.0, 'compound': 0.4215, 'text': 'united'},
       {'neg': 0.0, 'neu': 1.0, 'pos': 0.0, 'compound': 0.0, 'text': 'states'},
       {'neg': 0.0, 'neu': 1.0, 'pos': 0.0, 'compound': 0.0, 'text': 'before'},
       {'neg': 0.0, 'neu': 1.0, 'pos': 0.0, 'compound': 0.0, 'text': 'entering'}]
```

1.6 Step 7: Dataframe

Similar to Step 7 of Part 1, I made a dataframe for each president, which would hold the results from the sentiment analysis on each word. The first 15 rows of the sentiment DataFrame for Trump.

```
[12]: fdrSentimentDf2 = pd.DataFrame(fdr_Sentiments2)
      ajSentimentDf2 = pd.DataFrame(aj_Sentiments2)
      djtSentimentDf2 = pd.DataFrame(djt_Sentiments2)

      djtSentimentDf2.head(15)
```

```
[12]:     compound  neg  neu  pos       text
      0     0.0000  0.0  1.0  0.0     donald
      1     0.0000  0.0  1.0  0.0       john
      2     0.0000  0.0  1.0  0.0      trump
      3     0.0000  0.0  1.0  0.0         is
      4     0.0000  0.0  1.0  0.0        the
      5     0.0000  0.0  1.0  0.0       45th
      6     0.0000  0.0  1.0  0.0        and
      7     0.0000  0.0  1.0  0.0    current
      8     0.0000  0.0  1.0  0.0  president
      9     0.0000  0.0  1.0  0.0         of
      10    0.0000  0.0  1.0  0.0        the
      11    0.4215  0.0  0.0  1.0     united
      12    0.0000  0.0  1.0  0.0     states
      13    0.0000  0.0  1.0  0.0     before
      14    0.0000  0.0  1.0  0.0   entering
```

1.7 Step 8: Filtering the Most Positive or Negative Words

In this step, I decided to find the most positive words for FDR, most negative words for Trump, and both for Jackson. I first sorted each DataFrame by the `compound` column since this was a better indicator of the positivity/negativity of the word than the `pos` or `neg` columns. I decided to include only the 100 most extreme words since I did not want to reach any neutral words, and I wanted there to be repetition between the words so it could be seen in the wordcloud. I then joined the words in the list into a string, which would then be used to generate a wordcloud.

[13]:
```
def find_most_polarizing_words(sentiment_df):
    most_positive_lst = list(sentiment_df.sort_values(by = ['compound'],␣
 ↪ascending = False)[:100]['text'])
    most_positive_str = " ".join(most_positive_lst)
    most_negative_lst = list(sentiment_df.sort_values(by = ['compound'],␣
 ↪ascending = True)[:100]['text'])
    most_negative_str = " ".join(most_negative_lst)
    yield most_positive_str
    yield most_negative_str


# FDR: Franklin Delano Roosevelt
fdr_generator = find_most_polarizing_words(fdrSentimentDf2)
most_pos_fdr = next(fdr_generator)
most_neg_fdr = next(fdr_generator)

# AJ: Andrew Jackson
aj_generator = find_most_polarizing_words(ajSentimentDf2)
most_pos_aj = next(aj_generator)
most_neg_aj = next(aj_generator)

# DJT: Donald Trump
djt_generator = find_most_polarizing_words(djtSentimentDf2)
most_pos_djt = next(djt_generator)
most_neg_djt = next(djt_generator)

most_neg_djt
```

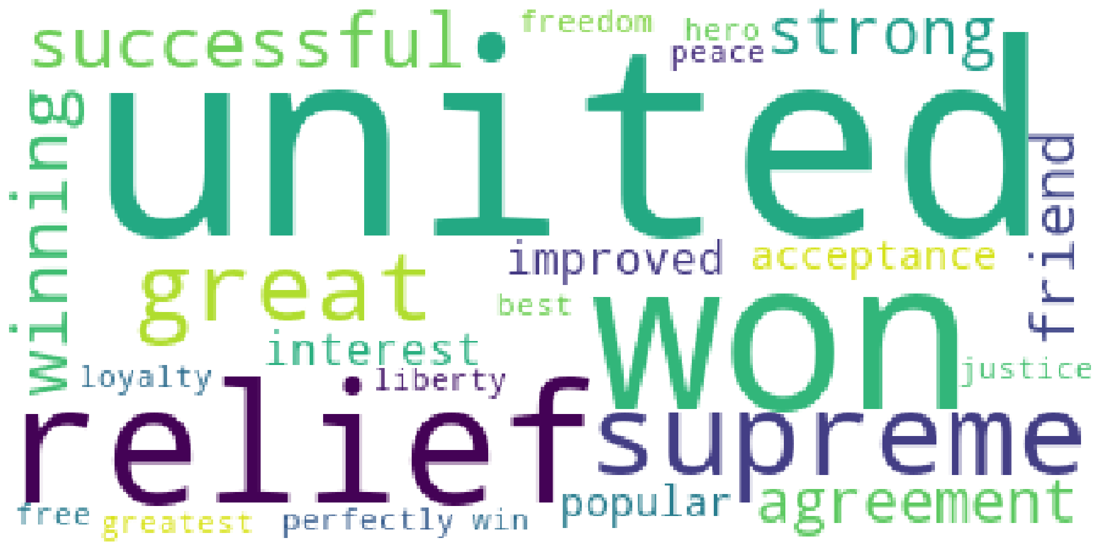
[13]:
```
'rape terrorism abuse abuse abuse violence racist racist betrayed racist racist
war death death war fraud assault negative hate negative criminals illegal fired
ban illegal ban fired felony violating enemy conspiracy criminal threat
conspiracy lying criminal violations conspiracy failed prosecution rejects
defeated unfair fake attack suspended unfair gross offensive forced criticism
block attacks weapons discredited attacks criticism denied guilty bitter guilty
disagreements deficit losses losses misleading strained losses shocking losses
tensions problems misleading losses fight upset adverse argued defections ignore
debt suspicions criticized warning resign fire argue pressuring gun loss leaked
conflict restricts misinterpreted contradicting weakened lost lost forbids lost'
```

1.8 Step 8: Visualizations of Most Frequent Extreme Words

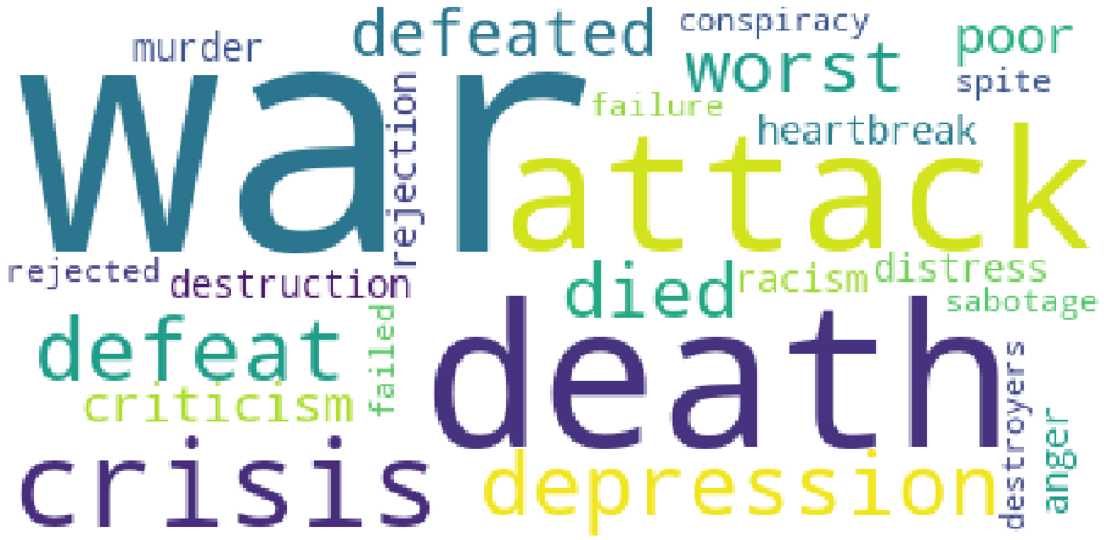
Here, I used a WordCloud object to display the most extreme words.

[14]:
```
cloud = WordCloud(background_color = "white", max_words = 25, collocations =␣
↪False, stopwords = set(STOPWORDS))

def make_wordCloud(most_polarizing_words_str):
    cloud.generate(most_polarizing_words_str)
    plt.figure(figsize = (9, 9), facecolor = None)
    plt.imshow(cloud)
    plt.axis("off")
    plt.tight_layout(pad = 0)


# FDR: Franklin Delano Roosevelt
make_wordCloud(most_pos_fdr)
plt.title('Most Frequent Positive Words About Franklin D. Roosevelt')

make_wordCloud(most_neg_fdr)
plt.title('Most Frequent Negative Words About Franklin D. Roosevelt')

# AJ: Andrew Jackson
make_wordCloud(most_pos_aj)
plt.title('Most Frequent Positive Words About Andrew Jackson')

make_wordCloud(most_neg_aj)
plt.title('Most Frequent Negative Words About Andrew Jackson')

# DJT: Donald Trump
make_wordCloud(most_pos_djt)
plt.title('Most Frequent Positive Words About Donald J. Trump')

make_wordCloud(most_neg_djt)
plt.title('Most Frequent Negative Words About Donald J. Trump')


plt.show()
```

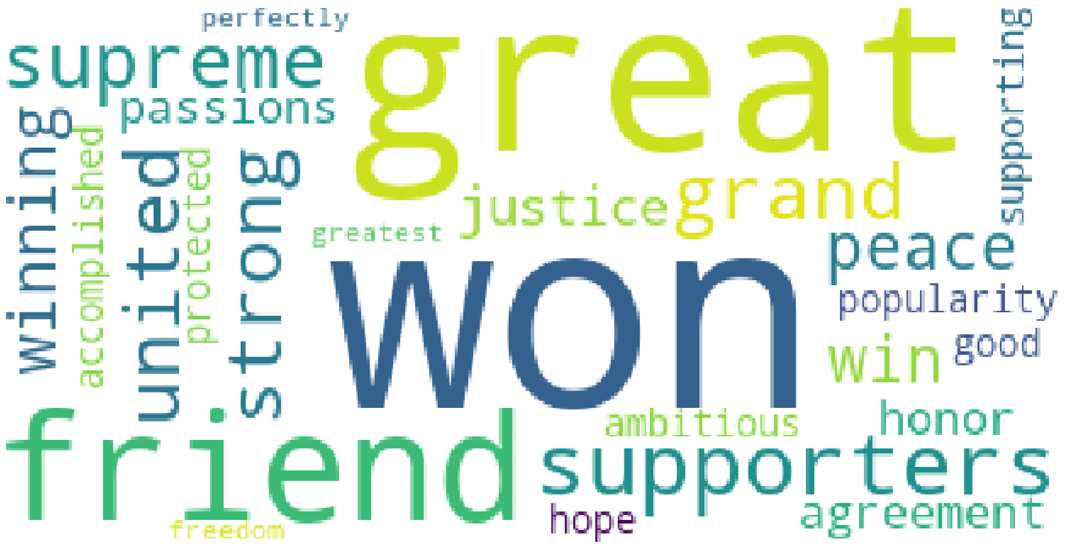
Most Frequent Positive Words About Franklin D. Roosevelt



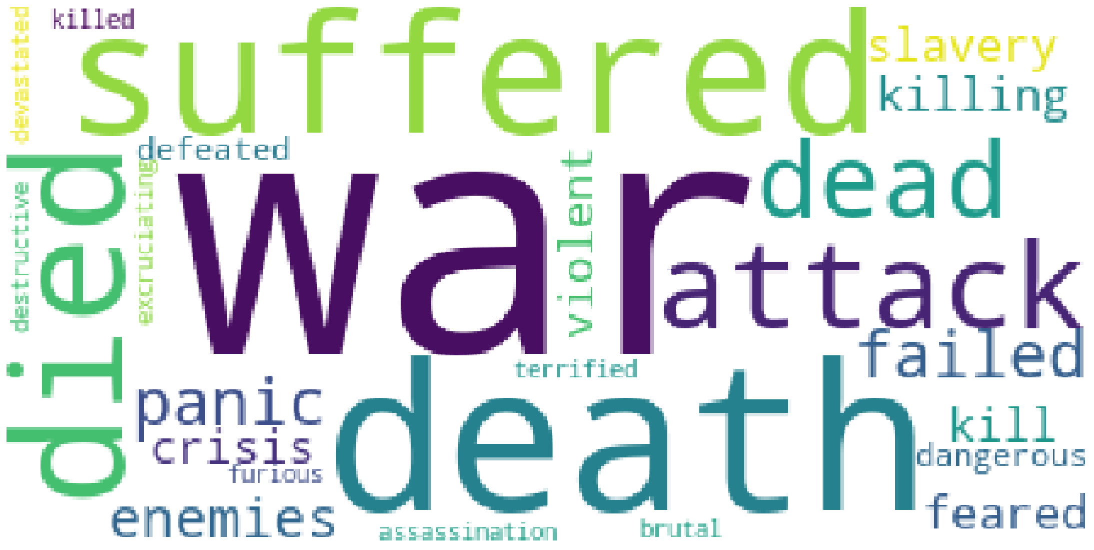
Most Frequent Negative Words About Franklin D. Roosevelt

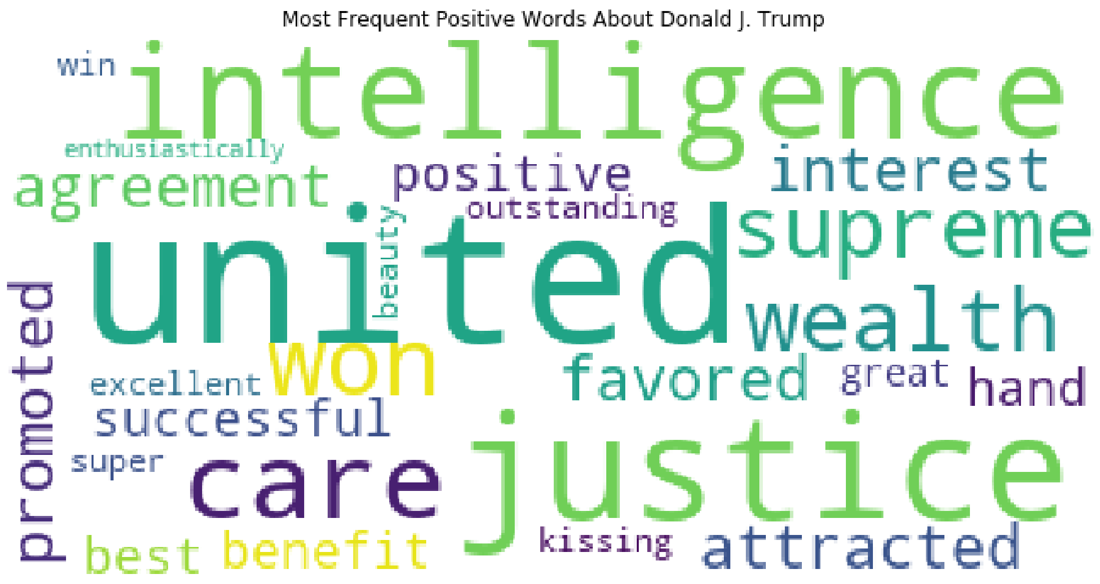


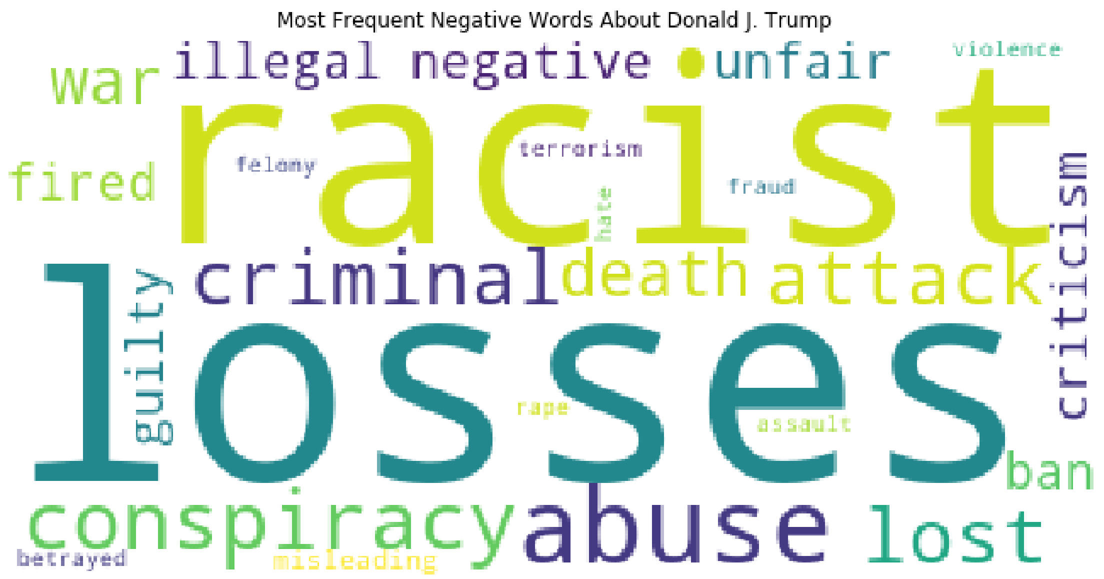
Most Frequent Positive Words About Andrew Jackson



Most Frequent Negative Words About Andrew Jackson







1.8.1 Conclusion for Part 2:

From the wordclouds, it's apparent that the words included are the most extreme, and suggest that Vader lexicon was a good way to gauge how positive or negative a word was, and its compound score was a good estimator.

2 Further Study:

In the future, I will be doing further experimentation with text analysis on Presidents or candidates during elections over time by comparing text in their speeches. This could very well demonstrate how the field of politics has changed over the past decades.

```
[ ]:
```