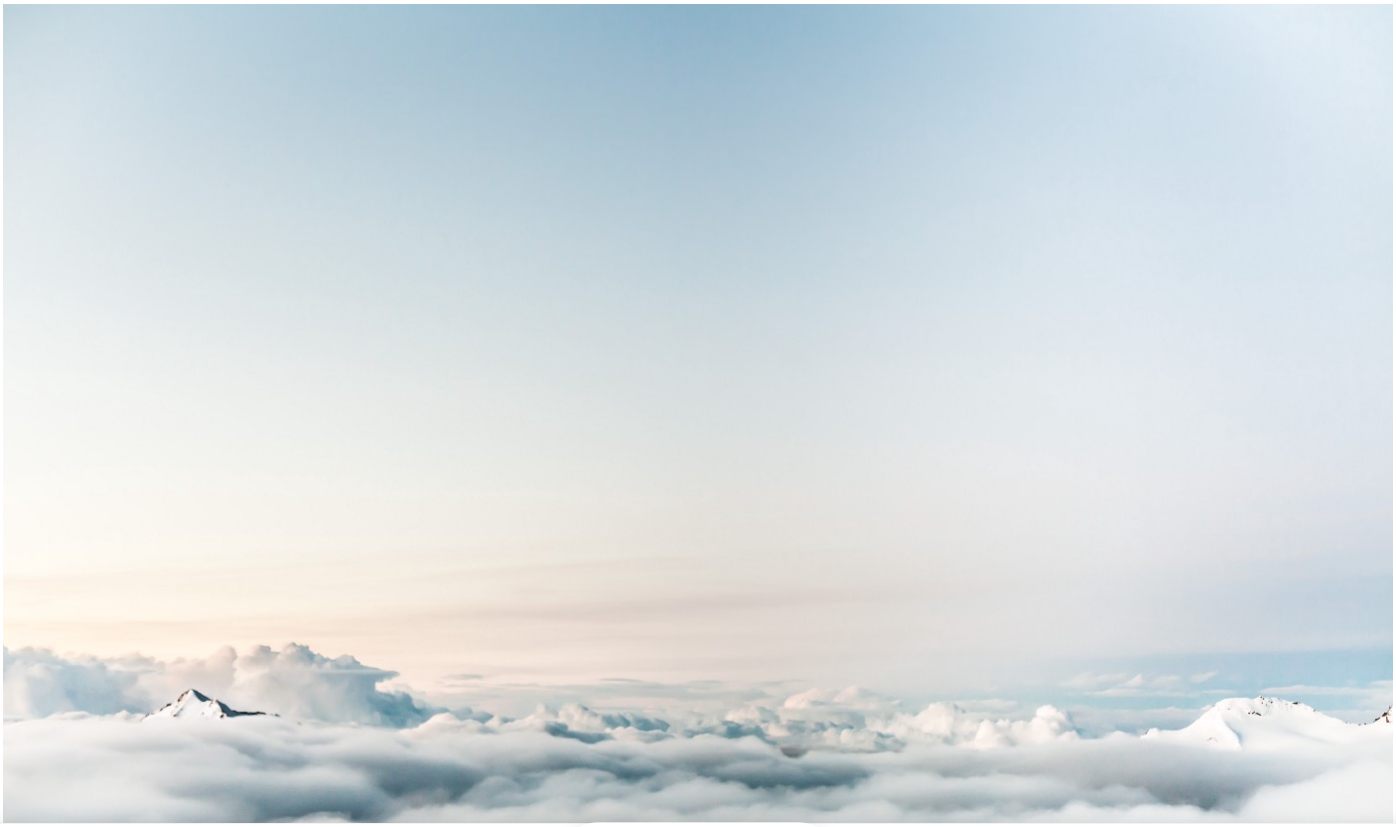
Published in Towards Data Science

Anupama Garla Follow

Feb 9, 2021 · 6 min read · ✨ · Listen

How to make Word Clouds in Python that Don’t Suck

A brief tutorial on making beautiful and meaningful word clouds in python



Motivation

During a recent NLP analysis on Presidential Inauguration Speeches, I found the results intriguing, but realized that most people wouldn't be so compelled by them as-is. I typically think word clouds can be horrendous, but thought they made sense here — to make the TF-IDF results more consumable immediately, I created word clouds from the TF-IDF Vectorization on the inaugural speeches of Biden, Trump, Obama, and Bush.

Word Clouds as an NLP Workflow Tool

The focus of this tutorial is to learn how to make word clouds that don't suck on a visual and content-based level. There are tons a free tools online for non-programmers to make word clouds, but visualizing your data within your python pipeline as you iterate parameters during an NLP project can be a powerful way to evaluate the results of your decisions, and present your process to non-technical stakeholders. Having a visual design background, I find the out-of-the-box word cloud package to be sort-of horrendous, so I will offer a few tweaks that will make your word clouds not suck.

How to make a word cloud in Python and Jupyter Notebook, Step by Step:

Here's an overview, but I will dive-in further as you proceed.

1. Install the wordcloud package.
2. Import wordcloud and matplotlib into your notebook.
3. Create a term-document matrix with TF-IDF values (Optional Step)
4. Run Word Cloud with text or matrix.
5. Adjust settings to make your Word Cloud not suck.
6. Wrap in a function and Iterate.

Here we go!

```
conda install -c conda-forge wordcloud
```

Alternatives here.

2. Import Packages

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

3. Create a term-document matrix with TF-IDF values (Optional Step)

You definitely do not need a TF-IDF Matrix to construct a word cloud — you can just use the text that you want to make a word cloud from. However, one way to make your word cloud not suck is to use a more meaningful dataset — one that has been massaged by TF-IDF. The advantages to using a TF-IDF Matrix in that you can control the types of words you are looking at in your word cloud — as opposed to the most frequently used, you can look at the most unique ones. Even using a count vectorizer and adjusting the settings prior to making a word cloud can be an effective workflow. This extra step of creating a TF-IDF Matrix makes comparative word clouds much more meaningful, and can be helpful to interpret the effects of adjusting settings in your nlp workflow.

To learn how to generate a TF-IDF Matrix, you can look at my previous article here.:

TF-IDF : A visual explainer and Python implementation on Presidential Inauguration Speeches

Ever asked to explain TF-IDF to non-technical audiences? Here is a visual unpacking of TF-IDF (Term Frequency — Inverse...

towardsdatascience.com

Building off my TF-IDF Matrix, I want a dataframe where each column is a document

```
data = df.transpose()

data.columns = ['document_bush_2001', 'document_obama_2009',
'document_trump_2017', 'document_biden_2021']

data.head()
```

This is the shape of the data I want to use:

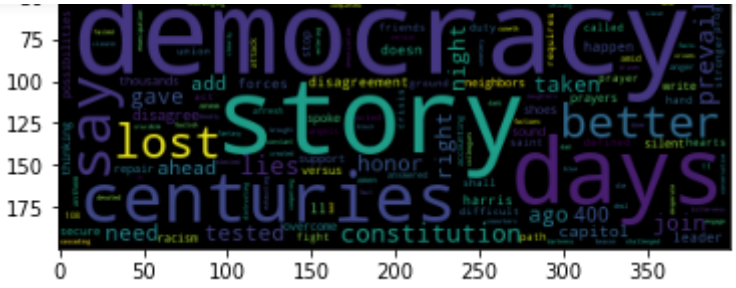
	document_bush_2001	document_obama_2009	document_trump_2017	document_biden_2021
000	0.0	0.0	0.000000	0.032037
108	0.0	0.0	0.000000	0.032037
11	0.0	0.0	0.000000	0.032037
1863	0.0	0.0	0.000000	0.032037
20	0.0	0.0	0.045494	0.000000

4. Run WordCloud with text.

Once you have your TF-IDF Matrix, you can use wordcloud to generate from frequencies and matplotlib to plot.

```
wordcloud =
WordCloud().generate_from_frequencies(data['document_biden_2021'])

plt.imshow(wordcloud)
```

The out of the box result looks like this:

Aside from the size and the inclusion of the axes, the colors of the words are mostly illegible, and don't correlate with the dataset. The sizes of the words are adjusted which is nice, but this is not great.

5. Adjust your settings to make your WordCloud not suck.

In addition to size and removing the axes, here are a few additional considerations:

Color — I am a traditionalist and use as little color as possible to make my point. For me, black text on a white or grey background works well. I don't think using both color and size to differentiate words based on one dataset of TF-IDF is necessary. However, if you were to use the color to represesent another dataset with the same words, that works for me. For instance, I would support the use of color if added another later of information like the Term Frequency or Inverse Document Frequency on its own.

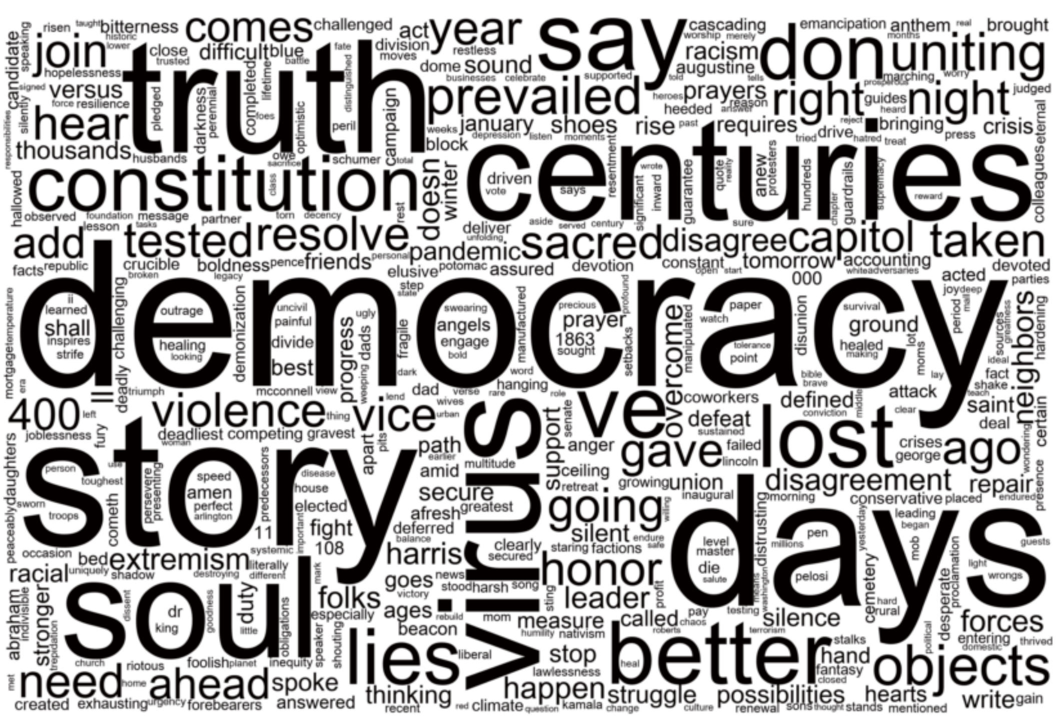
max words — I like the way paintings typically work in that you can understand the idea of the image at first glance, but when you dwell, you understand more. I think WordClouds can work like this so I like to up the number of words on the page because it allows one to simply understand the document at first glance, and then delve further in and dwell in the finer points of the document.

font_path — I despise the out of the box fonts and I will stick with arial or helvetica if I have the option, especially with a 'modern' looking visualizaton like word cloud.

```
# change the value to black
def black_color_func(word, font_size,
```

```
wordcloud = WordCloud(font_path = '/Library/Fonts/Arial Unicode.ttf',
background_color="white", width=3000, height=2000,
max_words=500).generate_from_frequencies(data['document_biden_2021'])

# set the word color to black
wordcloud.recolor(color_func = black_color_func)

# set the figsize
plt.figure(figsize=[15,10])

# plot the wordcloud
plt.imshow(wordcloud, interpolation="bilinear")

# remove plot axes
plt.axis("off")

# save the image
plt.savefig('biden_wc.png')
```

Here's a word cloud that doesn't suck:



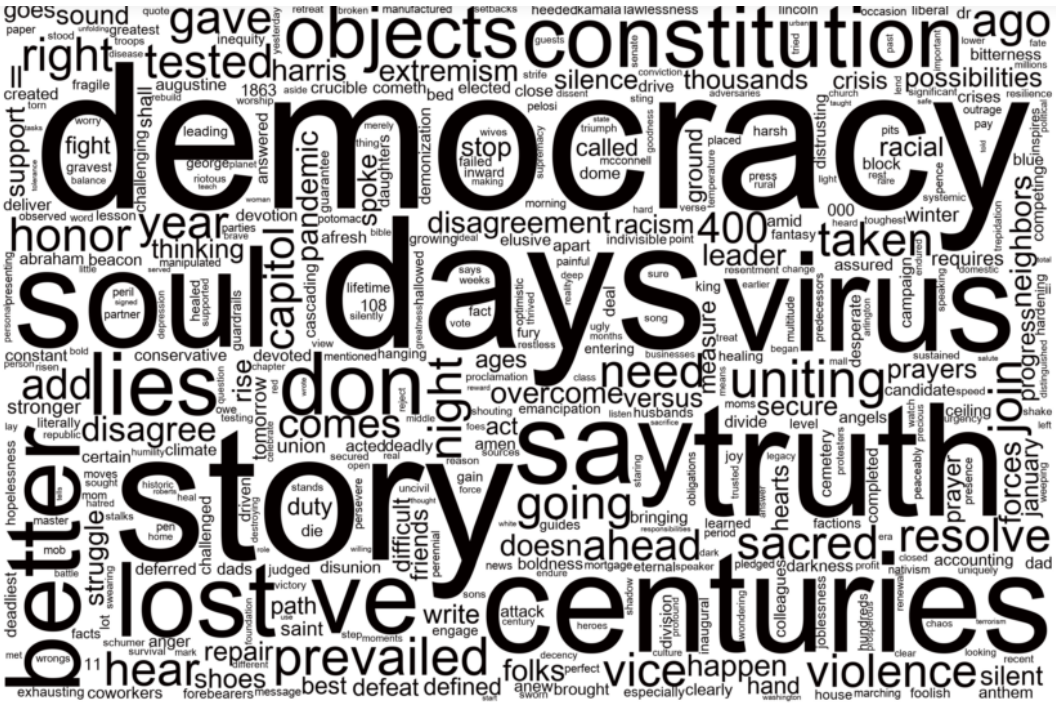
find out more here.

6. Wrap in a function and Iterate.

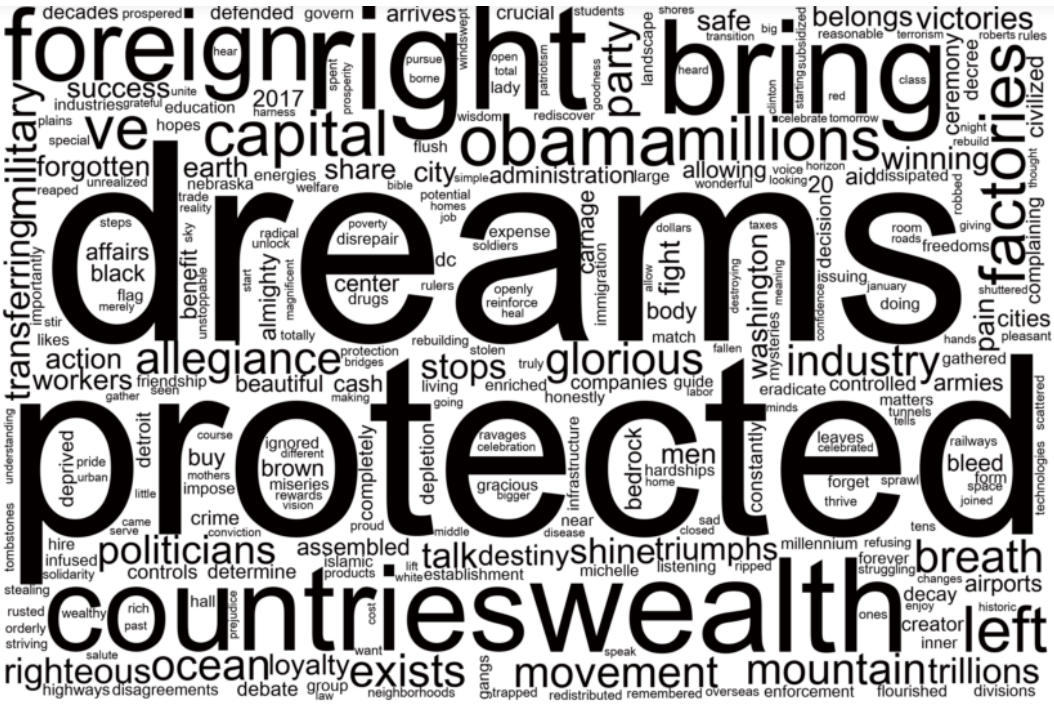
Once I have settings that I like, I wrap it into a function. As I progress with this analysis, I will likely iterate on my wordclouds to evaluate my TF-IDF settings like max and min tf, or a focus on particular parts of speech — nouns, adjectives, etc. To speed up my process, I may want to change the width and height of my output in my function, but this is a great way to get a feel for the decisions you take during nlp.

```
filenames = ['bush_2001_.png', 'obama_2009_.png', 'trump_2017_.png',
'biden_2021_.png']

def word_cloud(data, filenames):
    def black_color_func(word, font_size, position, orientation,
random_state=None, **kwargs):
        return("hsl(0,100%, 1%)")
    columns  = list(data)

for i in range(4):
        wordcloud = WordCloud(font_path = '/Library/Fonts/Arial
Unicode.ttf', background_color="white", width=3000, height=2000,
max_words=500).generate_from_frequencies(data[columns[i]])
        wordcloud.recolor(color_func = black_color_func)
        plt.figure(figsize=[15,10])
        plt.imshow(wordcloud, interpolation="bilinear")
        plt.axis("off")
        plt.savefig(filenames[i])

# plot the wordcloud
plt.imshow(wordcloud, interpolation="bilinear")

# remove plot axes
plt.axis("off")

# save the image
plt.savefig('biden_wc.png')
```

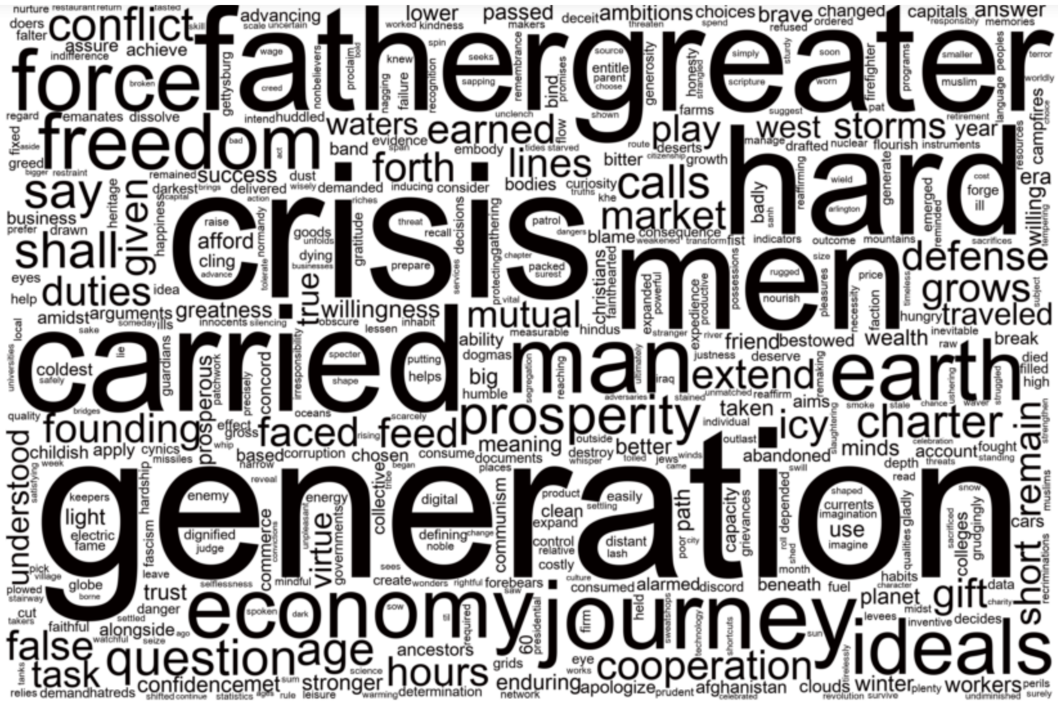
Comparative Word Clouds Here:



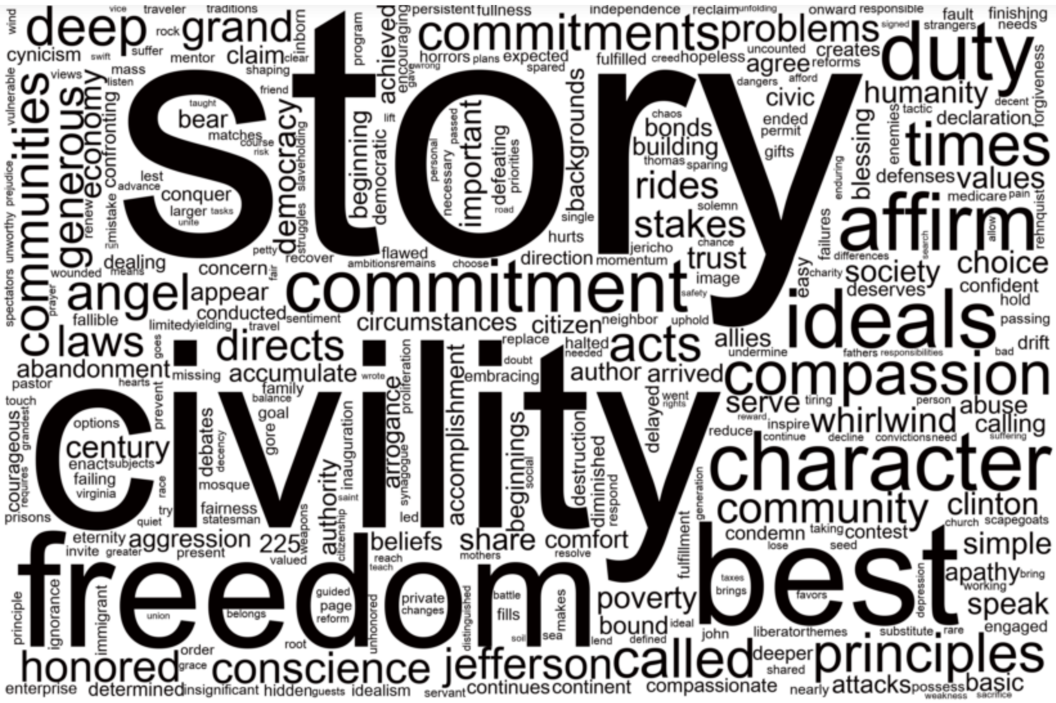
Biden 2021 Word Cloud of Inauguration Speech by Anupama Garla



Trump 2017 Word Cloud of Inauguration Speech by Anupama Garla



Obama 2009 Word Cloud of Inauguration Speech by Anupama Garla



Bush 2001 Word Cloud of Inauguration Speech by Anupama Garla

Unpacking the results of this study here:

Comparing Inaugural Addresses: Democracy is to Dreams as Biden is to Trump

A Comparative Content Analysis using natural language processing tools.

anupamagarla.medium.com