CS 193.1

Text Mining

What is in text?

Python (programming language)

From Wikipedia, the free encyclopedia

For other uses, see Python.

Python is an interpreted, high-level, general-purpose programming language. Created by Guido van Rossum and first released in 1991, Python's design philosophy emphasizes code readability with its notable use of significant whitespace. Its language constructs and object-oriented approach aims to help programmers write clear, logical code for small and large-scale projects.[27]

Python is dynamically typed and garbage-collected. It supports multiple programming paradigms, including procedural, object-oriented, and functional programming. Python is often described as a "batteries included" language due to its comprehensive standard library.[28]

Python was conceived in the late 1980s as a successor to the ABC language. Python 2.0, released 2000, introduced features like list comprehensions and a garbage collection system capable of collecting reference cycles. Python 3.0, released 2008, was a major revision of the language that is not completely backward-compatible, and much Python 2 code does not run unmodified on Python 3. Due to concern about the amount of code written for Python 2, support for Python 2.7 (the last release in the 2.x series) was extended to 2020. Language developer Guido van Rossum shouldered sole responsibility for the project until July 2018 but now shares his leadership as a member of a five-person steering council.[29][30][31]

Python interpreters are available for many operating systems. A global community of programmers develops and maintains CPython, an open source[32] reference implementation. A non-profit organization, the Python Software Foundation, manages and directs resources for Python and CPython development.

Cadence of Hyrule ~ Crypt of the NecroDancer Featuring The Legend of Zelda ~

Release date: June 13, 2019

Buy digital

Keep the beat as you play as Cadence, Link, or Princess Zelda to explore a randomly generated Hyrule and procedurally generated dungeons.

In the latest rhythmic action-adventure from Brace Yourself Games, you can enjoy the gameplay of Crypt of the NecroDancer in the setting of *The Legend of Zelda*™ series. As Link—or even as Princess Zelda—you'll explore the randomly generated overworld and procedurally generated dungeons on a quest to save Hyrule. Every beat of each remixed *Legend of Zelda* tune is a chance to move, attack, defend, and more, so stay one step ahead of each enemy and boss...or face the music.

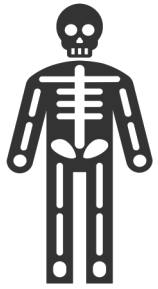
Text can have many parts.

The simple things:

Nouns, adjectives, verbs

Numerals, punctuations

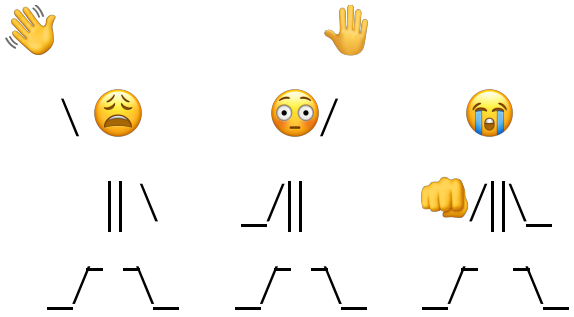
Phrases, clauses



More complex concepts:

Parts-of-speech, word dependencies

Abstract representations of meaning



We’re looking for these:

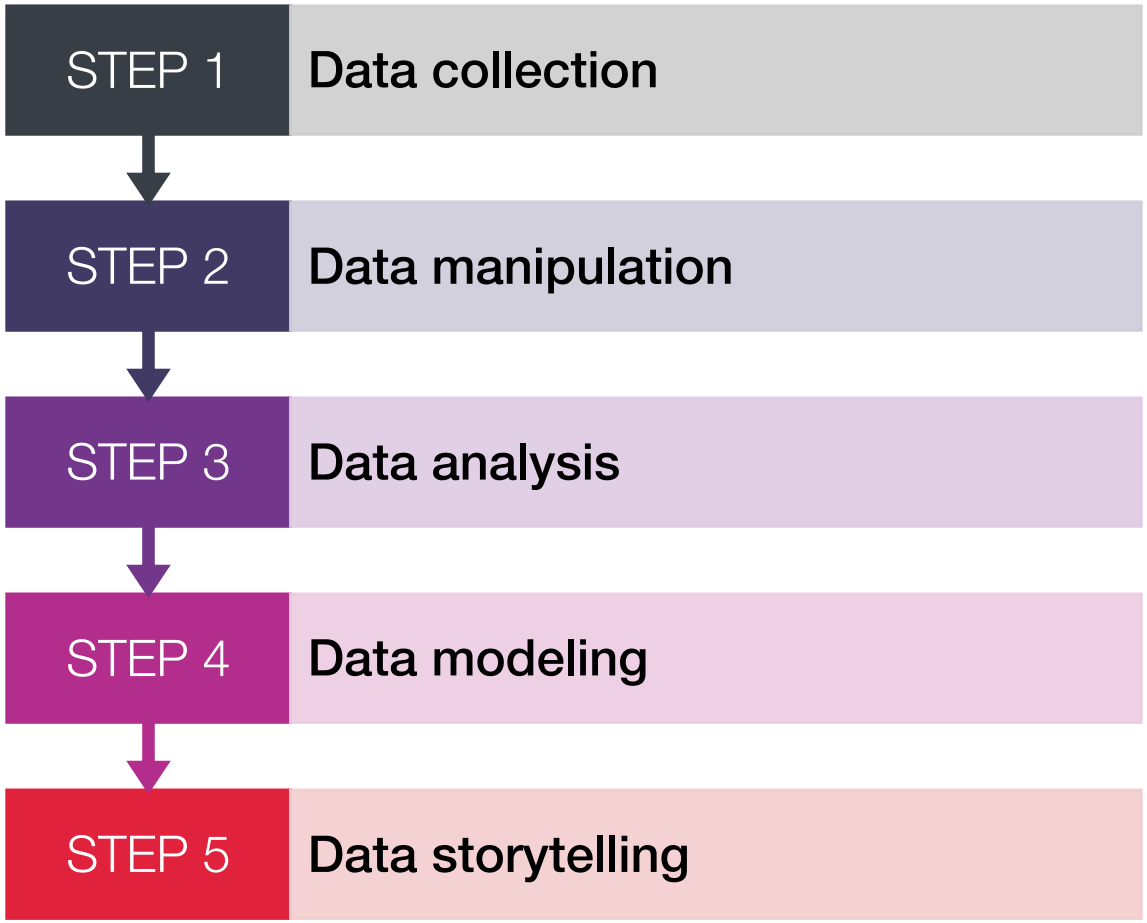
Language
(syntax, grammar)

Meaning
(semantics, context)

Framing

Personality

We can explore using text mining.

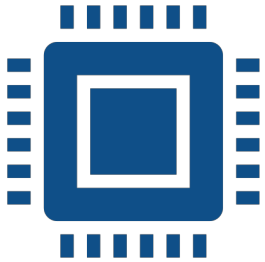


What is text mining?

- There's a story to be told in text.
  - Patterns in stories and narratives
  - Patterns in discourse and conversations

- There's value in the **wisdom of the crowd**!
  - That's why we crowdsource data
  - Put structure into unstructured text
  - Necessary to quantify qualitative data.

Text mining vs. NLP



Text mining is “simple.”

Grammar is removed

Simple and sufficient

Semantics are lost

Natural language processing is more complex.

Grammar and semantics are intact

Knowable parts of speech, dependencies

Named entity recognition is possible

Certain considerations for text mining:

Begin with data sources in mind.

Is data available or generatable?

Data can't solve everything.

Even if you have data, not everything is answerable.

Know your algorithms.

Blackboxing may lead to a lack of depth in knowledge.

How small is small?

- A small amount of data can only go so far.
  - Results may not **generalize** in the real world.
  - **Biases** and **overfitting** arise.
  - We want variance in our data!
- As much as possible, we want **big data**.

How do we gather data?

Outside of the “copy and paste” process.

Online data extraction

Create an account

Get API access keys

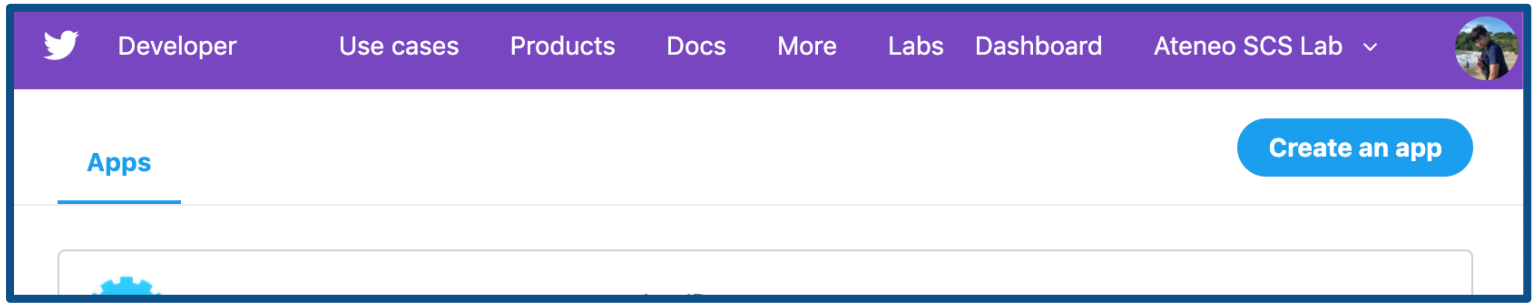
Transform data into dataframes

Clean dataframes

Determine analysis methods

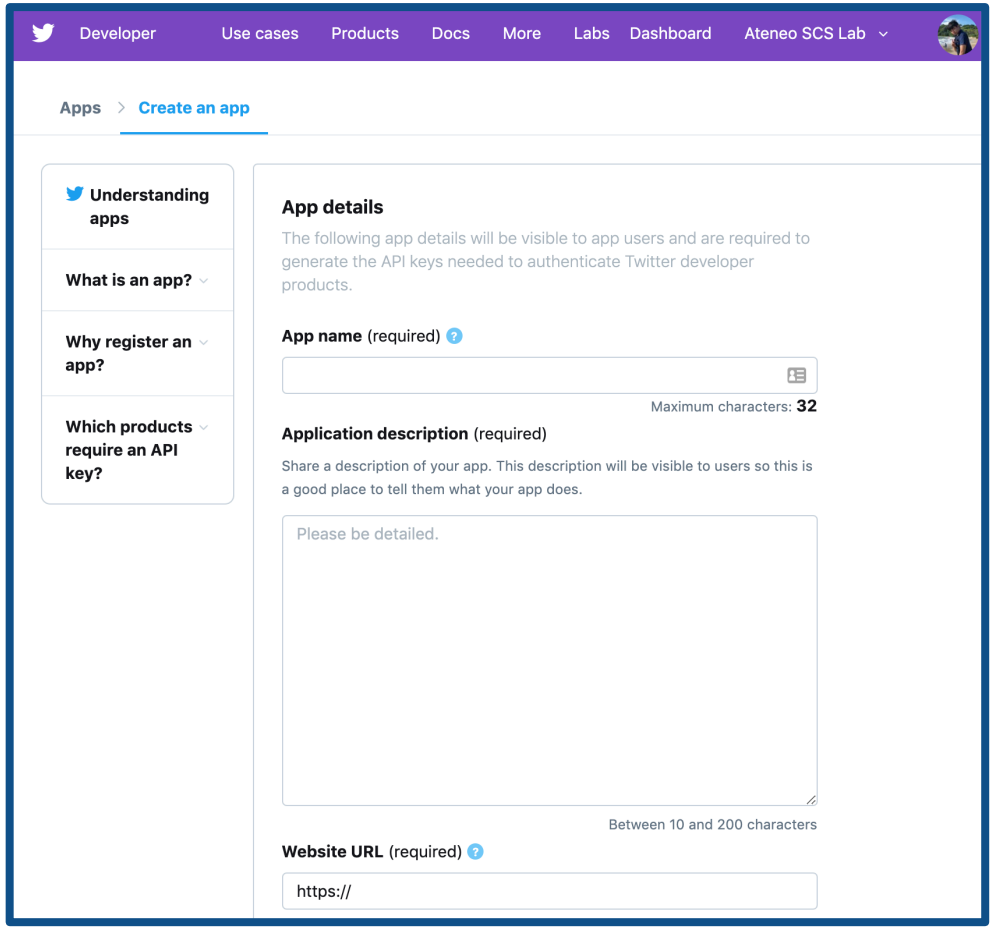
STEP 1 **Data collection**

Create a Twitter app



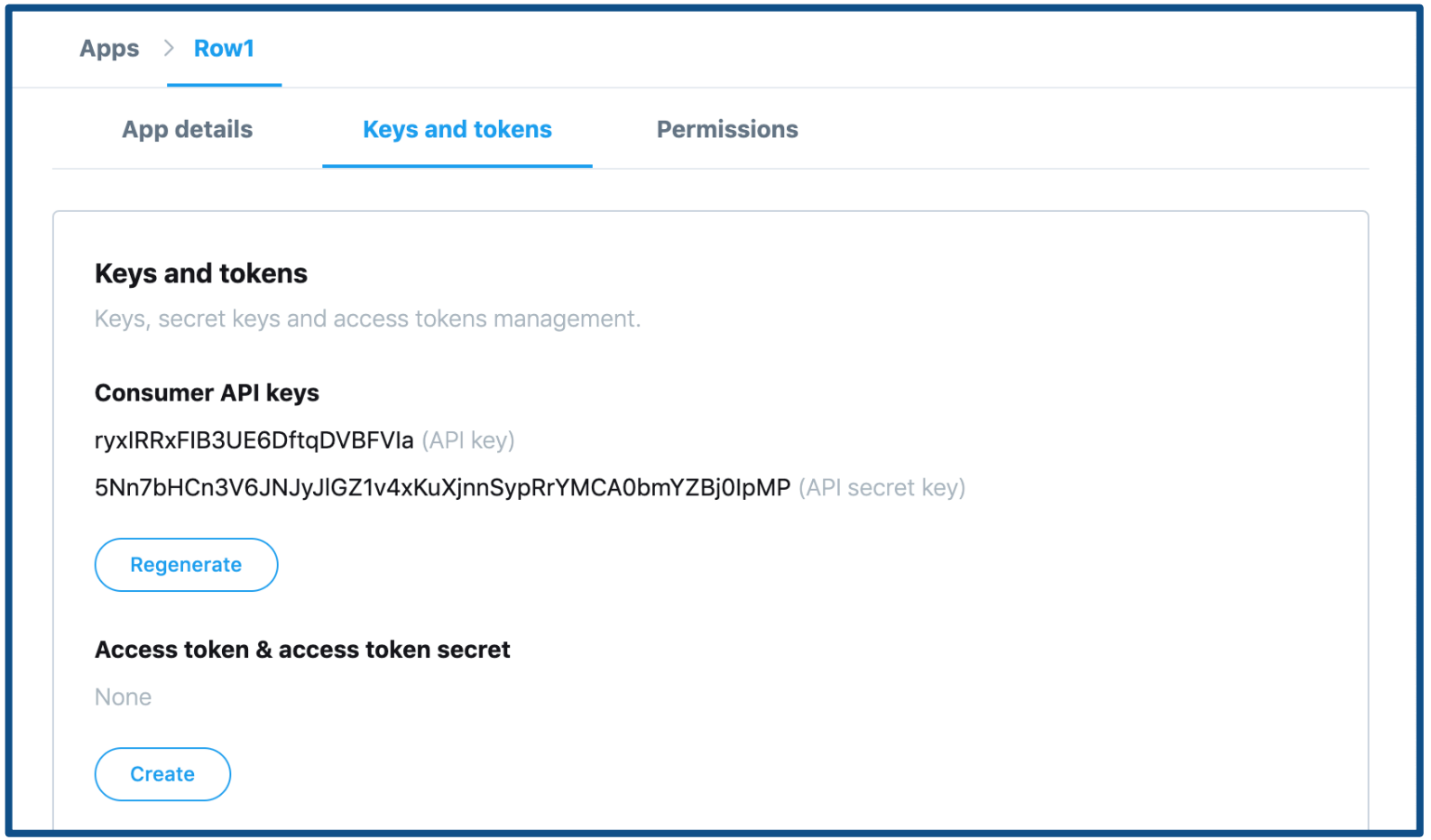
- Create a Twitter account
- Apply for a developer account
- Create an app

Create a Twitter app



- Fill out necessary details.

Create a Twitter app



- Get the API access keys

Setup extraction code

```
[2]: import tweepy
     import pandas as pd
```

```
[3]: # Application keys
     consumer_key = 'VpyTYjQp1p4RH5HLBESCQQYBe'
     consumer_key_secret = 'ggKmMrIz5Hw7xmR4eskK93MoHvaX33kM6npShNGggP0XayNObo'
     access_token = ''
     access_token_secret = ''
```

```
[3]: def create_api(consumer_key, consumer_key_secret):
         auth = tweepy.OAuthHandler(consumer_key, consumer_key_secret)
         auth.set_access_token(access_token, access_token_secret)
         api = tweepy.API(auth)
         return api
```

- Place API access keys for authentication
- Create an API instance

Setup extraction code

```
[4]: def search_tweets(api, query, ignore_rt=False, max_tweets=1000, lang='en'):
         if ignore_rt:
             query = query + ' -filter:retweets'
         search = tweepy.Cursor(api.search, 'apple', lang=lang).items(max_tweets)
         print("Downloaded {0} tweets".format(max_tweets))
         return search
```

```
[5]: def search_to_df(search_results):
         tweets = [[t.id, t.user.screen_name, t.created_at, t.text] for t in search_res
         tweets_df = pd.DataFrame(tweets, columns=['id', 'handle', 'created_at', 'text'
         return tweets_df
```

- Define a tweet search function
- Define a function that transforms search results into a Pandas dataframe

Execute extraction

```
[6]: api = create_api(consumer_key, consumer_key_secret)
     search_results = search_tweets(api, 'apple', ignore_rt=True)
     tweets = search_to_df(search_results)
     tweets.head()
```

```
Downloaded 1000 tweets
```

[6]:

	id	handle	created_at	text
0	1140785914790084608	CrowGutz	2019-06-18 00:58:51	RT @Misfits: 🎉 IT'S OUR BIRTHDAY 🎉 \n\n🏃 tim...
1	1140785909249351680	bt21bt21bt21JH	2019-06-18 00:58:50	RT @jungkookgallery: night apple is poison app...
2	1140785905285791744	itsxcaroline	2019-06-18 00:58:49	RT @beforeyouexit: beyond excited to announce ...
3	1140785904845578241	muddybike	2019-06-18 00:58:49	RT @airscottdenning: Have you seen the incredi...
4	1140785904279330817	daeshonstuff	2019-06-18 00:58:49	i spent 2 hours in the apple store waiting on ...

```
[ ]: # Export the tweets into CSV
     tweets.to_csv('tweets.csv', index=False)
```

- Run the functions with provided arguments.
- Don’t forget to save your tweets.

Why do we need to clean text?

Shouldn’t we try to extract everything from a text?

Text cleaning

Replace blank texts

Transform into lowercase

Delete special characters

Erase whitespace

Remove stopwords

Clean the tweets

Replace blank texts

```
['Hi, I'm new to twitter 😅',
'    lol aF 💯', 'The best!']
```

Transform to lowercase

```
['hi, i'm new to twitter 😅',
'    lol af 💯', 'the best!']
```

Delete special characters

```
['hi, i'm new to twitter ',
'    lol af ', 'the best!']
```

Erase whitespace

```
['hi, i'm new to twitter',
'lol af', 'the best!']
```

Remove stopwords

```
['hi, i'm new twitter',
'lol af', 'best!']
```

How do we quantify text?

We **assume** that a certain way of counting signifies its importance.

Vector representation

- A **corpus** is a collection of written text.
- A corpus is composed of **documents**.
- We can represent text as bags of words.
  - **Document Term Matrix** (TDM), where rows are terms and columns are documents
  - **N-grams**, where n-length phrases are considered
  - **Term Frequency Inverse Document Frequency** (TFIDF), where frequency of terms are adjusted by their rarity in documents

Visualize Nursery Rhymes

Corpus of Nursery Rhymes

Document 1: Jack and Jill Went Up the Hill

Document 2 : Humpty dumpty sat on the wall

Document 3 : The quick brown fox jumped over the moon

Document 4 : The quick brown fox jumped over the lazy cat

Visualize Nursery Rhymes

DTM	‘and’	‘hill’	‘jack’	‘jill’	‘the’	‘up’	'went’	…
doc1	1	1	1	1	1	1	1	…
doc2					1			…

Bi-gram	‘jack and’	‘and jill’	‘on the’	‘over the’	‘the moon’	‘the lazy’	‘lazy cat’	…
doc1	1	1	1	1	1	1	1	…
doc2			1					…
doc3				1	1			
doc4				1		1	1	

What counts?

- **Binary count**: does it exist?
- **Term frequency** (TF): how frequent does a term appear in a document?
- **Inverse document frequency** (IDF): how important is this term?
- **TFIDF**: how relevant is this term?
  - TF = term occurrence/total terms
  - IDF = $\log_e$(total docs)/total docs with term
  - TFIDF = TF*IDF

TFIDF is like your favorite food.

Plain old rice

Low TFIDF

Appears almost every time
= not special

Mom's spaghetti

High TFIDF

Rave reviews + rarity =
special place in your heart

Transform text into vectors

```python
def tweets_to_dtm(tweets):
    tweets = tweets['cleaned_text']
    vectorizer = CountVectorizer(max_features=2000)
    dtm = vectorizer.fit_transform(tweets)
    pickle.dump(vectorizer, open('out/dtm.pk', 'wb'))
    return dtm, vectorizer

def tweets_to_ngram(tweets, n=2):
    tweets = tweets['cleaned_text']
    vectorizer = CountVectorizer(
        ngram_range=(n, n),
        token_pattern=r'\b\w+\b',
        min_df=1,
        max_features=2000)
    dtm = vectorizer.fit_transform(tweets)
    pickle.dump(vectorizer, open('out/ngram.pk', 'wb'))
    return dtm, vectorizer

def tweets_to_tfidf(tweets):
    tweets = tweets['cleaned_text']
    vectorizer = TfidfVectorizer(max_features=2000)
    tfidf = vectorizer.fit_transform(tweets)
    pickle.dump(vectorizer, open('out/tfidf.pk', 'wb'))
    return tfidf, vectorizer
```

- These functions convert text into document-term, n-gram, and TFIDF matrices.
- Notice the **`pickle`**. **Question**: What does it do?

Transform text into vectors

```
# Get document-term matrix
dtm, dtm_v = tweets_to_dtm(cleaned_tweets)
print('DTM shape:', dtm.toarray().shape)
list(dtm_v.vocabulary_.items())[0:5]
```

```
# Get ngram matrix
ngram, ngram_v = tweets_to_ngram(cleaned_tweets, n=2)
print('Ngram matrix shape:', ngram.toarray().shape)
list(ngram_v.vocabulary_.items())[0:5]
```

```
# Get TFIDF matrix
tfidf, tfidf_v = tweets_to_tfidf(cleaned_tweets)
print('TFIDF matrix shape:', tfidf.toarray().shape)
list(tfidf_v.vocabulary_.items())[0:5]
```

- The code will apply the vector representation transformations on the cleaned texts.

Get frequencies from the vectorizers

```
def vector_to_frequency(vector, vectorizer):
    """
    Return a list of words and their corresponding occurence in the corpus
    """
    total = vector.sum(axis=0)
    frequency = [(w, total[0, i]) for w, i in vectorizer.vocabulary_.items()]
    frequency = pd.DataFrame(frequency, columns=['term', 'frequency'])
    frequency = frequency.sort_values(by='frequency', ascending=False).reset_index(drop=
    return frequency
```

```
freq_dtm = vector_to_frequency(dtm, dtm_v)
freq_dtm.to_csv('out/frequency_dtm.csv', index=False)
freq_dtm.head()
```

```
freq_ngram = vector_to_frequency(ngram, bigram_v)
freq_ngram.to_csv('out/frequency_ngram.csv', index=False)
freq_ngram.head()
```

```
freq_tfidf = vector_to_frequency(tfidf, tfidf_v)
freq_tfidf.to_csv('out/frequency_tfidf.csv', index=False)
freq_tfidf.head()
```

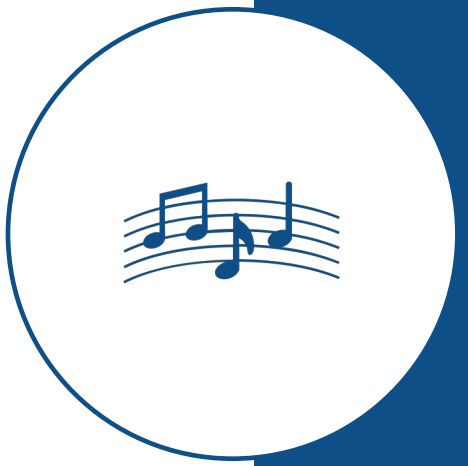
- Run the functions with provided arguments.
- Don’t forget to save your tweets.

How do we analyze text?

What kind of information are we trying to extract?

We make our data sing!

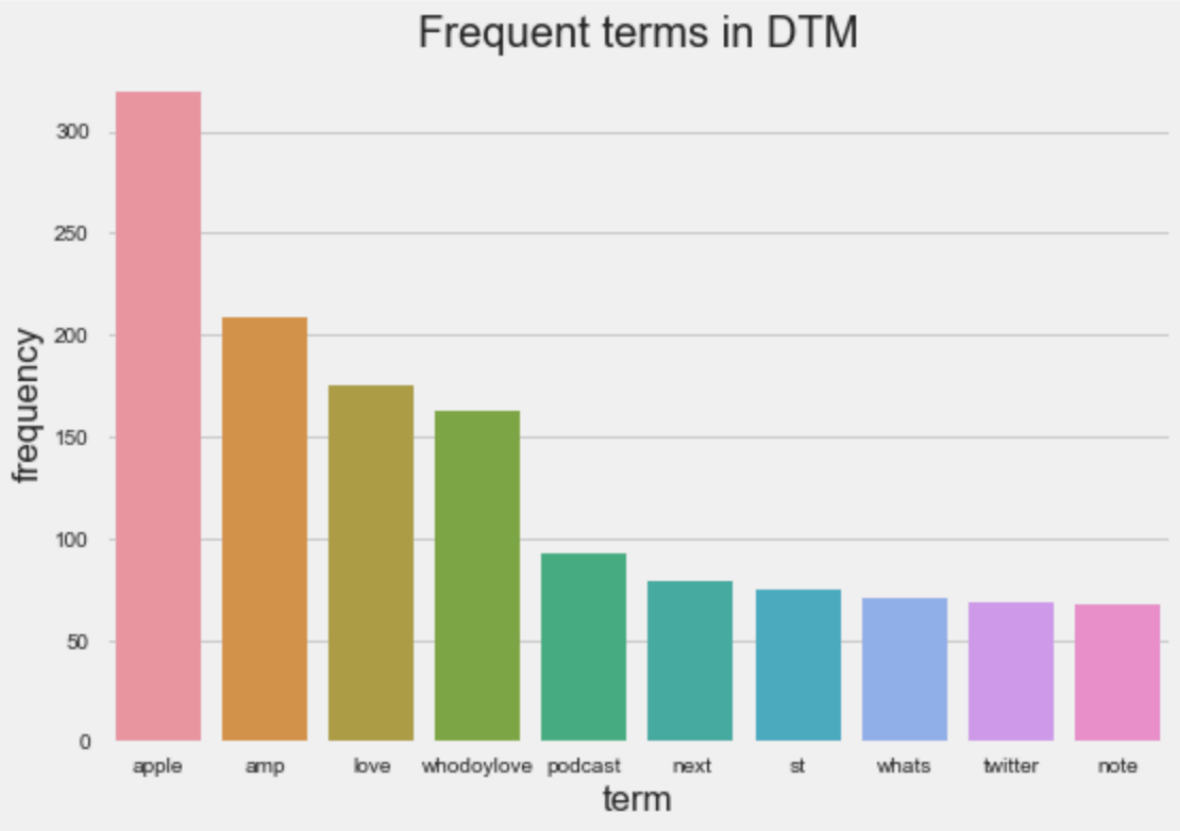
- An **exploratory data analysis** will look are things that pop, patterns that jump out, and attributes that are innately there.
  - Term frequency visualization
  - Wordclouds
  - Sentiment and subjectivity



Visualize frequencies

```
# Visualize frequencies
sns.barplot(data=freq_dtm.head(10), x='term',
            y='frequency').set_title('Frequent terms in DTM')
```

```
Text(0.5, 1.0, 'Frequent terms in DTM')
```



Generate wordclouds

```
# Generate a wordcloud
create_wordcloud(cleaned_tweets)
```

What is sentiment and subjectivity?

- The **sentiment** of a term is how polarized it is among certain classes, usually between **positive and negative**.
  - The Bing lexicon contains thousands of words with their assigned sentiment polarity.
  - Subjectivity works similarly: the use of personal-minded terms increases subjectivity.
- Lexicons are manually given sentiment and subjectivity values. **Q**: Is this problematic?

Assess sentiment and subjectivity

```
def analyze_tweets(tweets):
    """Analyzes the sentiment polarity and subjectivity of tweets"""
    sentiment = []
    subjectivity = []
    for text in tweets['text']:
        blob = tb(text)
        sentiment.append(blob.polarity)
        subjectivity.append(blob.subjectivity)
    tweets['sentiment'] = sentiment
    tweets['subjectivity'] = subjectivity
    return tweets
```

```
# Analyze the sentiment and subjectivity of tweets
analyzed_tweets = analyze_tweets(cleaned_tweets)
analyzed_tweets.head()
```

id	handle	created_at	text	cleaned_text	sentiment	subjectivity
36172530274304	ConnDiandra	2019-06-18 04:18:33	RT @SaraCarterDC: #TheSaraCarterShow: What's N...	thesaracaershow whats next russiainvestigation...	0.000000	0.000000
836171112747008	lillerik	2019-06-18 04:18:33	RT @viticci: Never seen this alert before – Ap...	never seen ale apple tells app youre deleting ...	0.283333	0.600000
36170051534848	bae_hon	2019-06-18 04:18:33	RT @Shazam: We love @OfficialMonstaX & @Fr...	love amp whodoylove	0.500000	0.600000
36169279631360	megandurazo	2019-06-18 04:18:33	RT @speriod: we need a new fiona apple album	need new fiona apple album	0.136364	0.454545
836169174749184	Nahirk	2019-06-18 04:18:33	RT @ij_baird: Help make FaceTime awesome, by a...	baird help make facetime awesome applying swee...	0.675000	0.825000

Visualize sentiment vs. subjectivity

```
# Visualize distribution of sentiment and subjectivity
sns.lmplot(data=analyzed_tweets, x='sentiment', y='subjectivity', fit_reg=True)
```

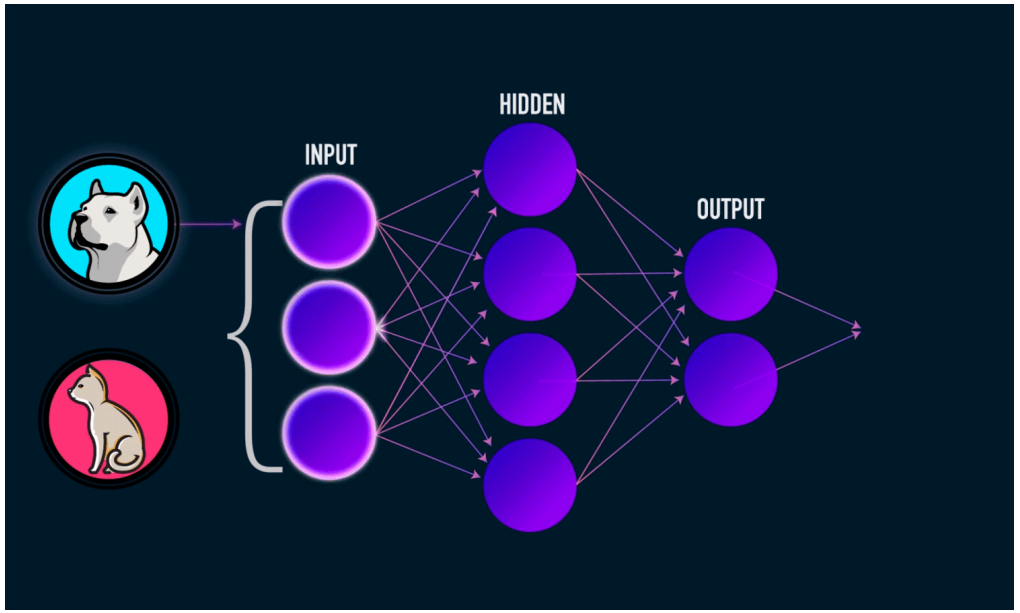
```
<seaborn.axisgrid.FacetGrid at 0x125c989e8>
```

How can we create text models?

Can we use data in prediction or clustering?

Not always a model

Always a model



Photos courtesy of Universal Records and Towards Data Science.

We use the mathematical definition of a model.

- A model is an object with:
  - A collection of finite operations
  - Relations that are defined on it
  - Ability to satisfy a collection of “axioms”
- **Q**: what things in computer science are also models?

Loading previous models

```
# Load existing vectorizers
dtm_v = pickle.load(open('out/dtm.pk', 'rb'))
ngram_v = pickle.load(open('out/ngram.pk', 'rb'))
tfidf_v = pickle.load(open('out/tfidf.pk', 'rb'))
```

```
# Load terms from existing vectorizers
dtm_terms = dtm_v.get_feature_names()
ngram_terms = ngram_v.get_feature_names()
tfidf_terms = tfidf_v.get_feature_names()
```

```
# Transform texts into representations
dtm = dtm_v.fit_transform(texts)
ngram = ngram_v.fit_transform(texts)
tfidf = tfidf_v.fit_transform(texts)
```

- We load our existing models from the pickled files.
- We can extract attributes from it, and even use it to transform other texts as well.

Topic modelling

Unsupervised learning task that aims to extract main topics (sets of words) that occur in a corpus.

For this, we use **Latent Dirichlet Allocation**

Uses data to explain why some parts of the data are similar.

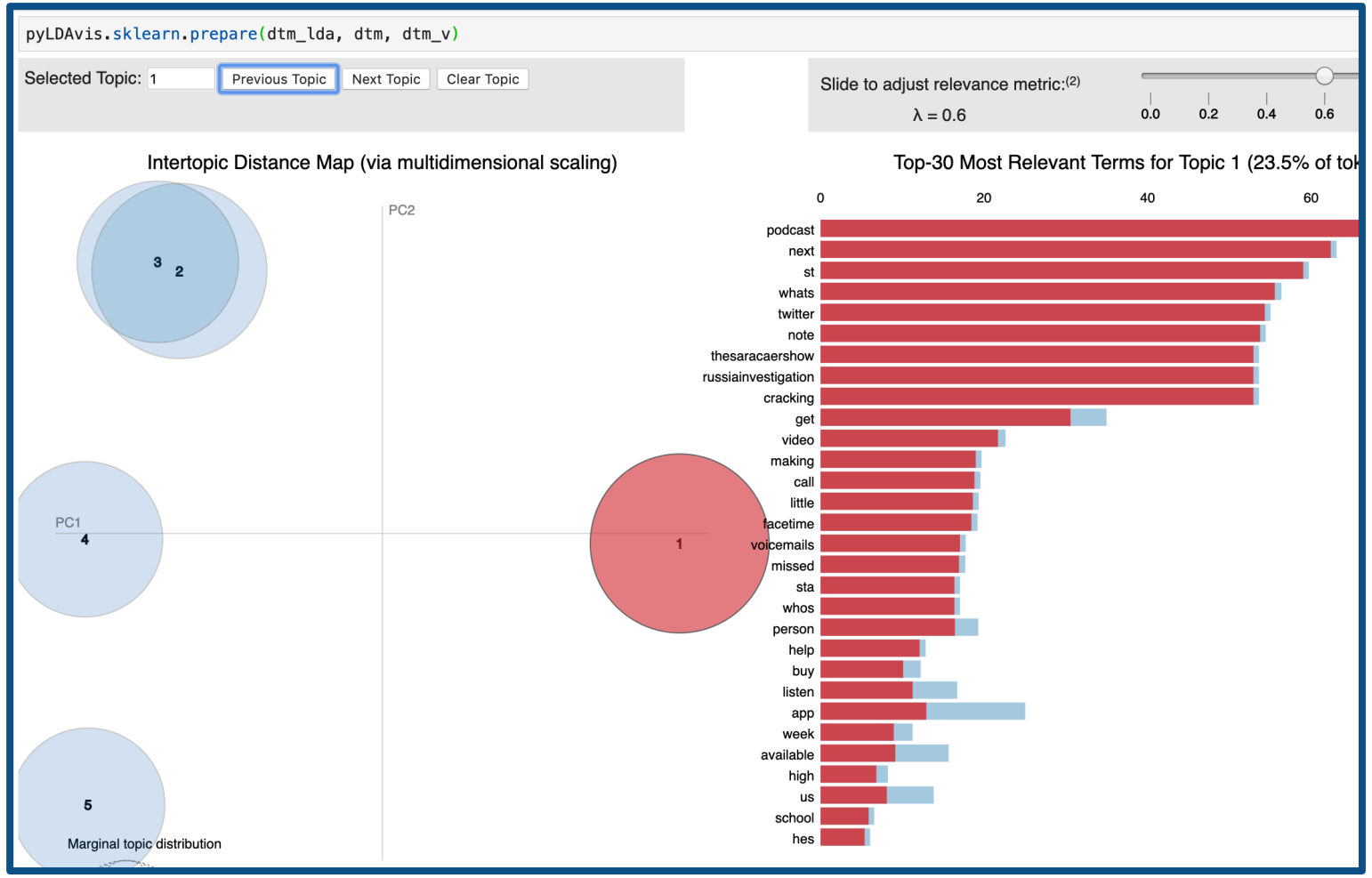
Performing LDA

```
def lda(texts):
    """
    Executes LDA algorithm to determine topics of the corpus
    """
    lda_model = LatentDirichletAllocation(
        n_components=5, max_iter=10,
        learning_method='online',
        learning_offset=50., random_state=0)
    lda_model.fit_transform(texts)
    return lda_model
```

```
# Execute LDA
dtm_lda = lda(dtm)
ngram_lda = lda(ngram)
tfidf_lda = lda(tfidf)
```

- Here, the model looks for 5 topics amongst all the tweets.
- It uses the vector representations we created beforehand.

Visualizing LDA



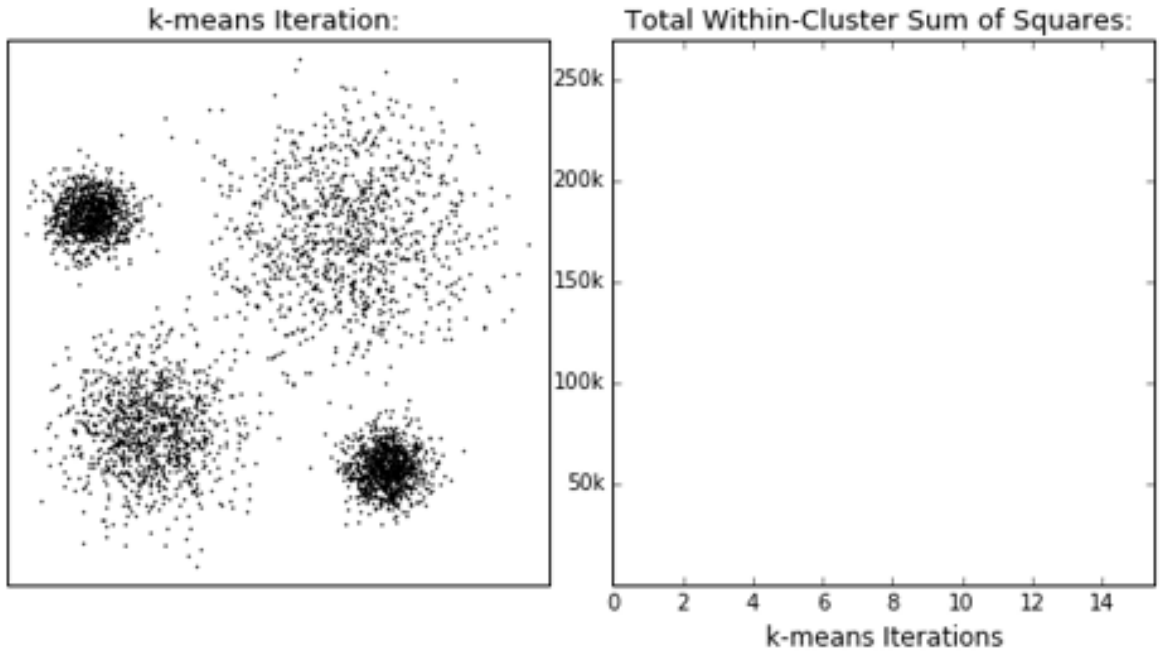
- We use the `pyLDAvis` package to interactively visualize results.
- Check it out on your notebooks ☺

Clustering

We can also group our tweets by reducing their variances using **K-means**.

We also need to visualize our clusters via dimensionality reduction with **PCA** and **t-SNE**.

K-means visualized



Clustering, visualizing tweets

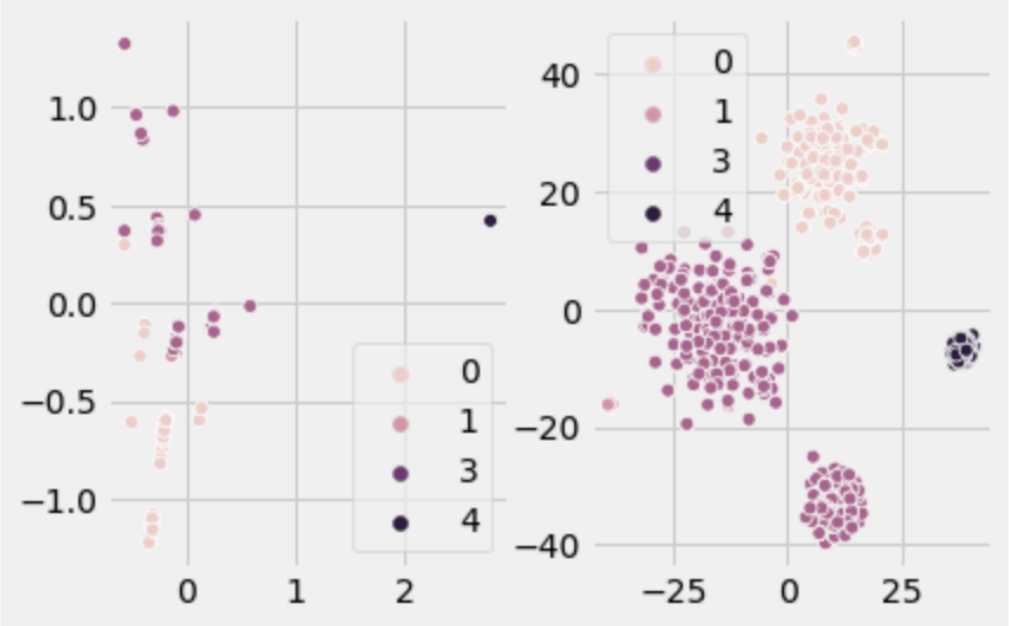
```python
def kmeans(texts, clusters=5):
    """
    Execute Kmeans algorithm to generate clusters
    """
    kmeans_model = KMeans(n_clusters=clusters, max_iter=1000)
    kmeans_model.fit(texts)
    kmeans_result = kmeans_model.predict(texts)
    return kmeans_result
```

```python
def plot_kmeans(texts, clusters):
    """
    Generates visualizations of clusters using Prinicipal Component Analysis
    and t-Distributed Stochastic Neighbor Embedding
    """
    pca = PCA(n_components=2).fit_transform(texts.todense())
    tsne = TSNE().fit_transform(PCA(n_components=50).fit_transform(texts.todense
    pca_a = pca[:, 0]
    pca_b = pca[:, 1]
    tsne_a = tsne[:, 0]
    tsne_b = tsne[:, 1]

    f, ax = plt.subplots(1, 2)
    sns.scatterplot(x=pca_a, y=pca_b, hue=clusters, ax=ax[0], )
    sns.scatterplot(x=tsne_a, y=tsne_b, hue=clusters, ax=ax[1], )
```

Clustering, visualizing tweets

```
dtm_kmeans = kmeans(dtm)
plot_kmeans(dtm, dtm_kmeans)
```



How do we tell data stories?

Why is storytelling important?

Storytelling is not just showing results.

- It’s like painting a picture.
  - Start with structured intent (asking “wicked” questions)
  - Assess what you have (data, methods, tools)
  - Describe your experiments
  - Generate insights (answer your wicked questions)

Next time…

- Major requirement topic proposal
- Tool exercise 1: Text mining
- Natural language processing