# DATA SOCIETY®

Advanced text mining - part 1

*"One should look for what is and not what he thinks should be."*
*-Albert Einstein.*

## Who we are

- Data Society's mission is to **integrate Big Data and machine learning best practices across entire teams** and empower professionals to identify new insights
- We provide:
  - High-quality data science training programs
  - Customized executive workshops
  - Custom software solutions and consulting services
- Since 2014, we've worked with thousands of professionals to make their data work for them

# Using Zoom

**Raise your hand if you can hear me OK.**

- From the toolbar (probably on the bottom of your screen), ***select the button marked "Reactions" and choose the hand***

**In the chat box, tell everyone what you see out the nearest window.**

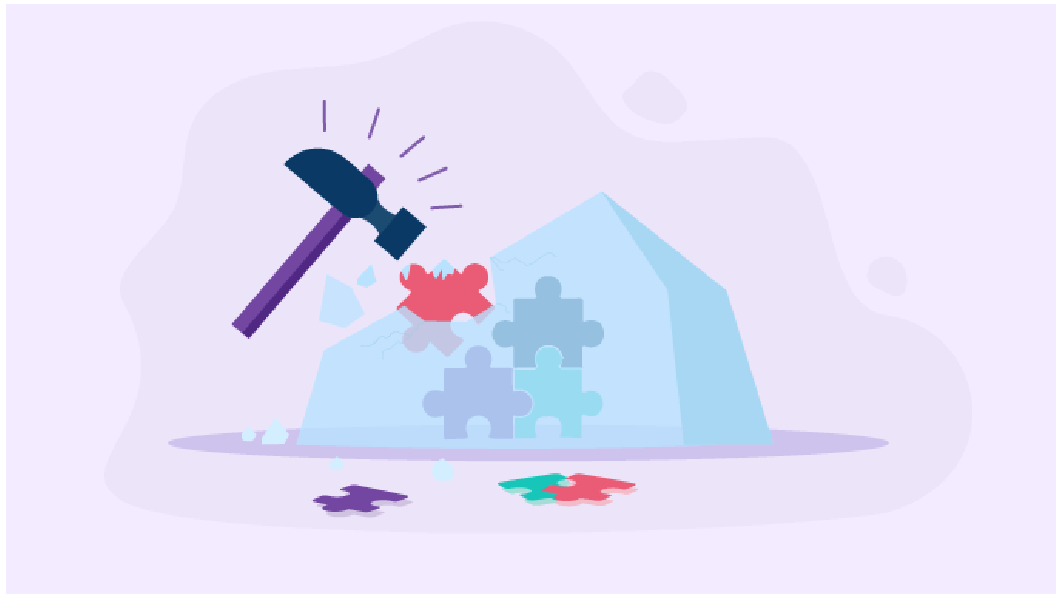
- From the toolbar (probably on the bottom of your screen), ***select the button marked "Chat."*** The chat box should appear. On smaller screens, the "Chat" button may be hiding under the "More" menu.

# Best practices for virtual classes

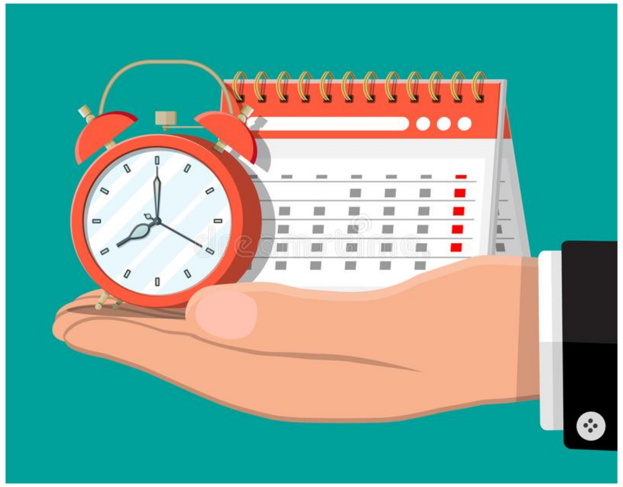
1. Find a quiet place, free of as many distractions as possible. Headphones are recommended.
2. Stay on mute unless you are speaking.
3. Remove or silence alerts from cell phones, e-mail pop-ups, etc.
4. Participate in activities and ask questions. This will be interactive!
5. Give your honest feedback so we can troubleshoot problems and improve the course.

## Getting to know your classmates

- Let's head into a breakout room and introduce ourselves
- You'll have 5-10 minutes to:
  - exchange your names and departments
  - discuss previous text mining classes that you've taken
  - talk about what problems you hope to solve by taking this course
- When you come back, be ready to share 1-2 topic areas that came up as project interests with the whole group!



## How we teach



- We'll walk through the concepts and code together, then you'll have the opportunity to answer questions and practice
- You should have the following:
  - Code files to follow along with the slides
  - Links to interactive knowledge checks
  - Exercise files (we give you 2 files and one has the answers)
- Recordings will be made available

# Warmup

Before we get started, here's a scenario to remind ourselves what we already know about text mining.

- Suppose you've been tasked with analyzing about **5,000 posts** to an **online feedback forum** for the U.S. Department of Veterans Affairs
  - What **steps** would you need to take to **manually code** this data?
  - How could you **determine** the type and number of **topics**?
  - How would you **assign** posts to a **topic**?
  - What **problems** could arise in attempting to draw **conclusions**?

# Module completion checklist

| Objective | Complete |
|---|---|
| Visualize similar documents using interactive network graph | |
| Explain hierarchical clustering (hclust) and how it applies to text | |
| Build out hclust model on text data | |
| Visualize clustering results as a dendrogram and compare results with LDA | |

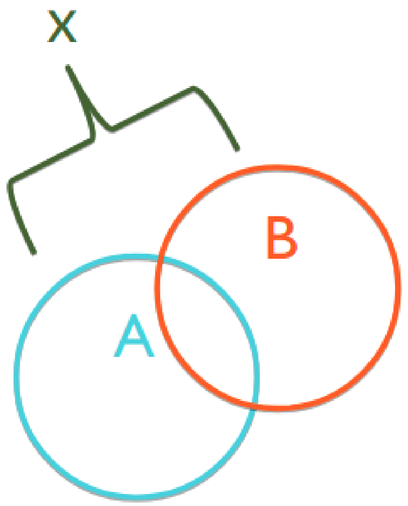
## Comparing terms and documents in text

- One of the most common analyses in text mining is **finding similar terms and documents**
- To find **two terms or documents** that are **similar** in their meaning, we use a metric called **cosine similarity** to **measure the distance** between them

# Recap: Cosine similarity

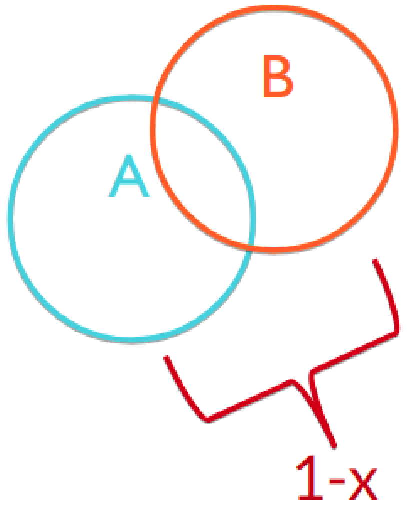
## Cosine similarity

- Measure of how close two objects are
- Similarity between `A` and `B` is high, because the objects are close to each other
- If `similarity(A, B) = x`



## Cosine distance

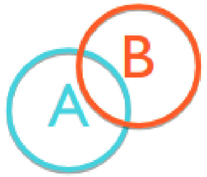
- Measure of how far away two objects are
- Distance between `A` and `B` is small, because the objects are close to each other
- Then `distance(A, B) = 1-x`



# Recap: Cosine similarity (cont'd)

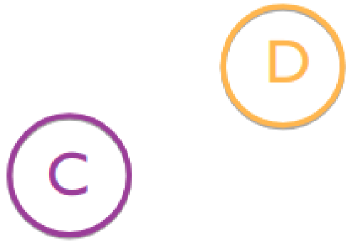
## Cosine similarity

- If `similarity(A, B) = 0.8`
- If `similarity(C, D) = 0.1`



## Cosine distance

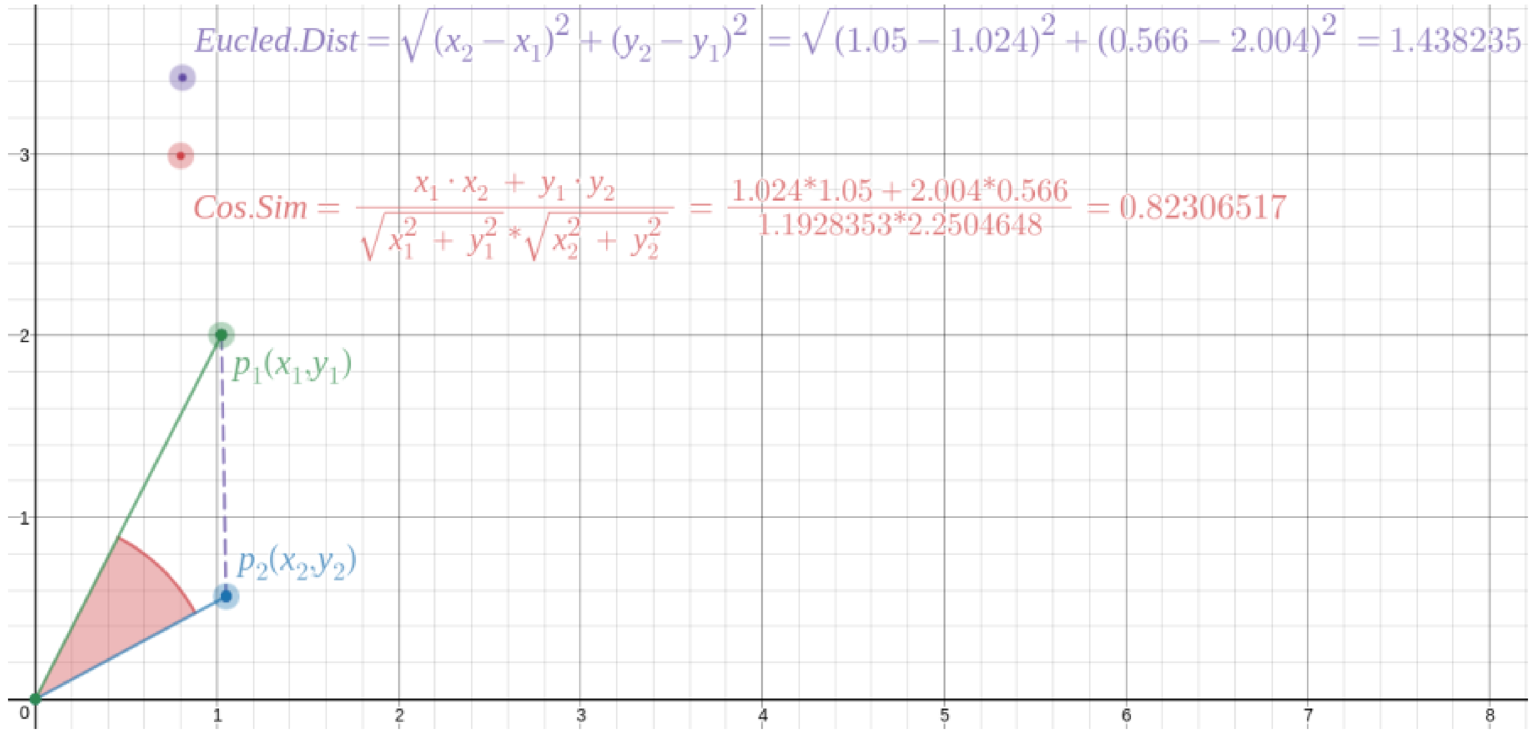
- then `distance(A, B) = 0.2`
- then `distance(C, D) = 0.9`



*Note that cosine similarity & cosine distance will always be values between `[0,1]`!*

# Recap: Cosine similarity (cont'd)

- **Cosine similarity** is $cos(a)$, where $a$ is an angle between the two *vectors*
- These vectors are either *terms* or *documents*, depending on what we would like to compare
- In order **to compute cosine similarity between documents, we need to know their numeric representation**



$Eucled.Dist = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2} = \sqrt{(1.05 - 1.024)^2 + (0.566 - 2.004)^2} = 1.438235$
$Cos.Sim = \frac{x_1 \cdot x_2 + y_1 \cdot y_2}{\sqrt{x_1^2 + y_1^2} * \sqrt{x_2^2 + y_2^2}} = \frac{1.024*1.05 + 2.004*0.566}{1.1928353*2.2504648} = 0.82306517$
$p_1(x_1,y_1)$
$p_2(x_2,y_2)$

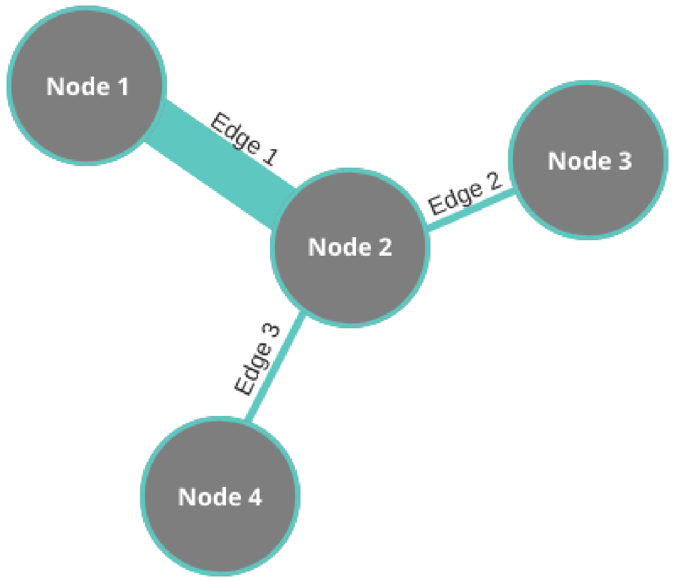
# Network graphs: visualizing term similarity

- We can visualize the similarity of text data as a **network of terms or documents**
- The terms or documents represent **nodes**
- The **cosine similarity scores** influence connecting **edges and their weights**

Edges

| source | target | weight |
|---|---|---|
| Node 1 | Node 2 | 0.99 |
| Node 2 | Node 3 | 0.1 |
| Node 2 | Node 4 | 0.1 |

Network graph



## Chat discussion

So here's some scenarios to get us thinking:

- Recommendation engines look for similarities between something you like and something related that you might also like
- What kind of **text data** might be useful in trying to build a recommendation engine for:
  - a new kind of ***dinner*** to prepare?
  - a new ***movie*** to watch?
  - the next ***song*** in an automatically generated playlist?

# Our goal

- In this module, we will evaluate snippets of New York Times articles for their **subject** and potential **topic**
- We will also evaluate **all snippets** for general word **distribution** and overall **patterns**

# Data we will be working with

- The `data` folder inside of your class folder contains the files that you will need
- The file that contains the text for analysis is `NYT_article_data.csv`
- In this case, we will consider
  - Each **article snippet** as a **document**
  - **All article snippets** together as a **corpus**

The New York Times

DATA PROVIDED BY
The New York Times

# A note on class data

- The data that we'll use has been pre-processed so that we can focus on advanced text mining topics
- If you want to practice text mining fundamentals after class, your steps would be as follows:
  - import the original `NYT_article_data` file
  - clean it using the `Bag of Analysis` approach
  - create DTM and DTM TF-IDF matrices
  - get topics and their probablities using LDA
  - compute a similarity matrix using cosine similarity

# Import packages

- Let's start by importing the packages that we'll need

```
# Helper packages.
import os
import pickle
import pandas as pd
import numpy as np
```

```
# Cosine similarity and clustering packages.
from sklearn.metrics.pairwise import cosine_similarity
from scipy.cluster.hierarchy import ward, dendrogram, fcluster
from gensim import matutils
```

```
# Network creation and visualization.
import networkx as nx
from pyvis.network import Network
```

```
# Other plotting tools.
import pyLDAvis
import pyLDAvis.gensim
import matplotlib.pyplot as plt
```

```
/Library/Frameworks/R.framework/Versions/3.6/Resources/library/reticulate/python/rpytools/loader.py:19:
DeprecationWarning: the imp module is deprecated in favour of importlib; see the module's documentation
for alternative uses
  module = _import(
```

# Directory settings

- In order to maximize the efficiency of your workflow, you should encode your directory structure into `variables`
- Let the `main_dir` be the variable corresponding to your `booz-allen-hamilton` folder

```
from pathlib import Path
# Set `home_dir` to the root directory of your computer.
home_dir = Path.home()

# Set `main_dir` to the location of your `booz-allen-hamilton` folder.
main_dir = home_dir / "Desktop" / "booz-allen-hamilton"

# Make `data_dir` from the `main_dir` and remainder of the path to data directory.
data_dir = main_dir / "data"

# Make `plot_dir` from the `main_dir` and remainder of the path to data directory.
plot_dir = main_dir / "plots"
```

## Working directory

- Set the working directory to the `data_dir` variable

```
# Set working directory.
os.chdir(data_dir)
```

```
# Check working directory.
print(os.getcwd())
```

```
/home/[user-name]/Desktop/booz-allen-hamilton/data
```

# Import data

```
# Load NYT article data from original file.
NYT = pd.read_csv('NYT_article_data.csv')
```

```
# Load pickled data and models.

# Cosine similarity matrix computed from DTM TF-IDF matrix
similarity_df = pickle.load(open("similarity_df.sav","rb"))
similarity = pickle.load(open("similarity.sav","rb"))

# Indices of documents with word counts over `5`
valid_snippets = pickle.load(open("valid_snippets.sav","rb"))

# A list of document id, topic id and topic probability for that document derived from LDA model
doc_topic_pairs_df = pickle.load(open("doc_topic_pairs_df.sav","rb"))

# An array of word counts per snippet.
word_counts_array = pickle.load(open("word_counts_array.sav","rb"))
```

# Compute a graph from similarity object

- Again, our starting point today is the similarity matrix
- To convert the similarity matrix to a `graph` or `network` object, we will use the `networkx` package that we imported as `nx`
- We can make use of the `nx.from_pandas_adjacency()` function, which takes a `pandas` *adjacency* dataframe as its only required argument
- Since the similarity matrix (a.k.a. *adjacency* matrix) is in the form of a `pandas` dataframe, we will make use of it to create a `digraph` object

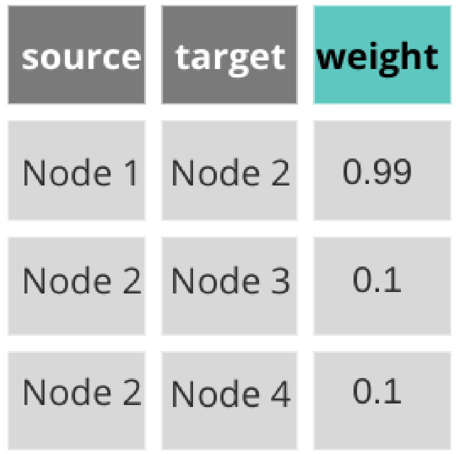
```
# Create a graph object from the similarity matrix.
graph = nx.from_pandas_adjacency(similarity_df)
```

- A `graph` object is a network object that is used to define **nodes**, **edges**, and **weights** of the graph
- Learn more about graph types and their inner workings ***here***

# Compute an edgelist from graph

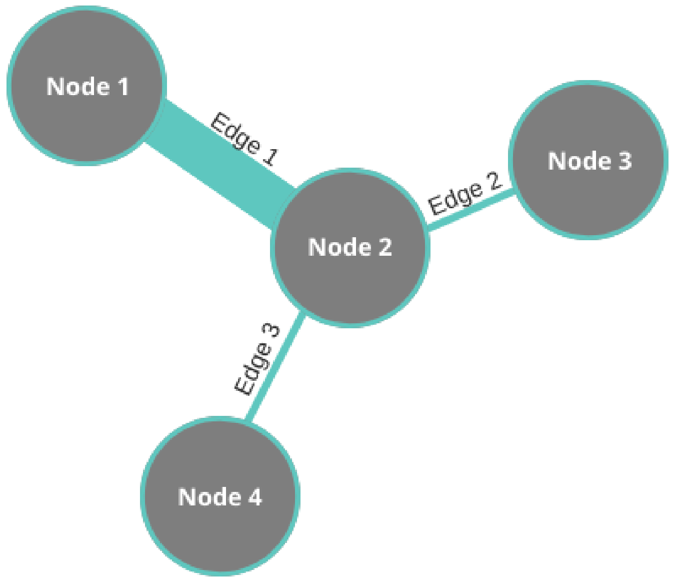
- Instead of using the `graph` object itself, we will convert it back to a dataframe suitable for drawing a network graph
- This dataframe is also known as an `edgelist`, and it will be in this form:

Edges



| source | target | weight |
|---|---|---|
| Node 1 | Node 2 | 0.99 |
| Node 2 | Node 3 | 0.1 |
| Node 2 | Node 4 | 0.1 |

Network graph



# Compute an edgelist from graph

- We will make use of another function from the `networkx` package to convert a `graph` to an `edgelist`
  - `nx.to_pandas_edgelist(graph)`

```
# Convert it to a dataframe in a form of an edgelist.
edgelist_df = nx.to_pandas_edgelist(graph)
```

```
# Take a look at the dataframe of edges.
print(edgelist_df.head())
```

```
   source  target    weight
0       0       0  1.000000
1       0       7  0.101372
2       0       8  0.100684
3       0       9  0.052663
4       0      11  0.084635
```

```
print(edgelist_df.shape)
```

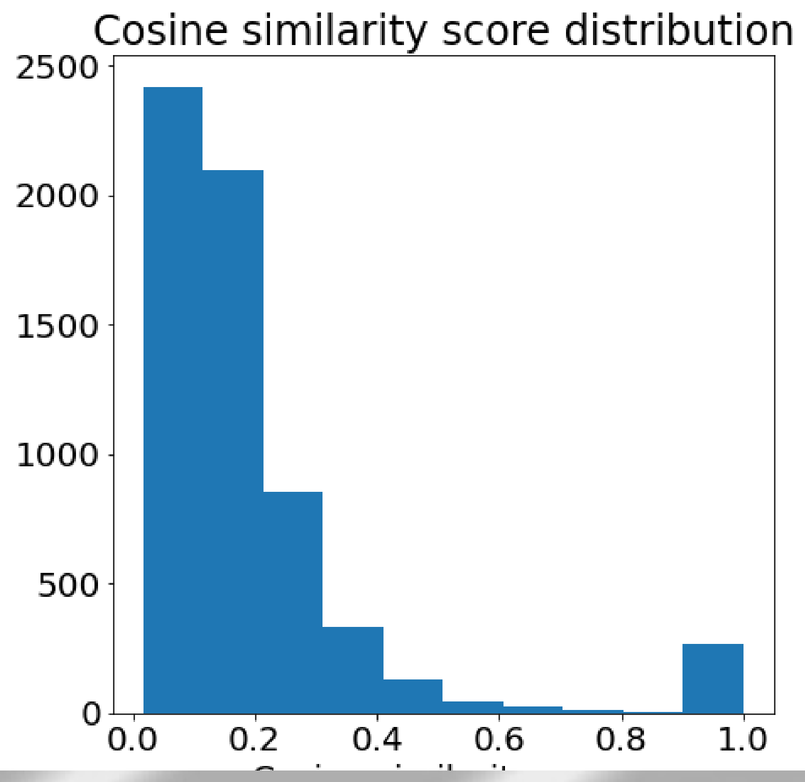
```
(6163, 3)
```

# Cosine similarity score distribution

- Notice that a lot of scores are `0`, with most scores under `0.4`
- You can **create an arbitrary threshold and subset your term similarity pairs** using it
- That way we can **drop scores that are too low or too high to be meaningful**
- Very low scores are indicate low similarity between documents, while scores close to `1` indicate pairs of the same documents

```
# Plot the weights of edges (i.e. similarity scores).
plt.hist(edgelist_df['weight'])
```

```
plt.xlabel('Cosine similarity score')
plt.title('Cosine similarity score distribution')
plt.show()
```

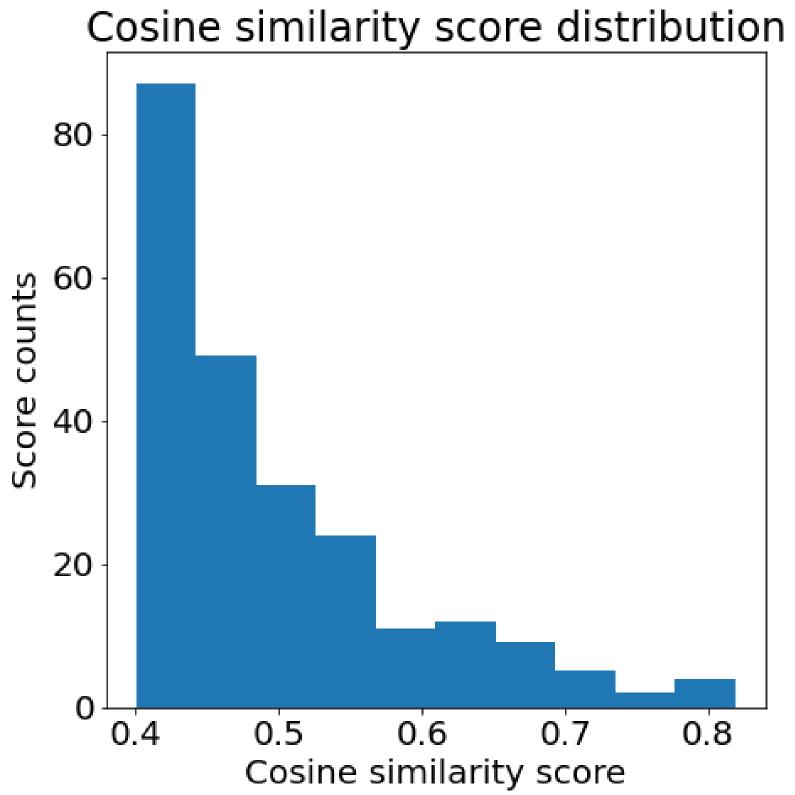


# Filter out edges

- Let's filter out all pairs of documents with weights below or equal to `0.4` and greater than `0.9`

```
# Filter out all entries below 0.4 and above
0.9.
edgelist_df = edgelist_df.query('weight>0.4 and
weight<0.9')
```

```
# Plot the weights of edges (i.e. similarity
scores).
plt.hist(edgelist_df['weight'])
plt.xlabel('Cosine similarity score')
plt.ylabel('Score counts')
plt.title('Cosine similarity score
distribution')
plt.show()
```



## Check edges data and number of edges

```
# Take a look at the dataframe of edges.
print(edgelist_df.head())
```

```
     source  target    weight
7         0      17  0.402407
18        0      60  0.412396
127       2     234  0.419921
152       2     219  0.419921
180       3      58  0.616884
```

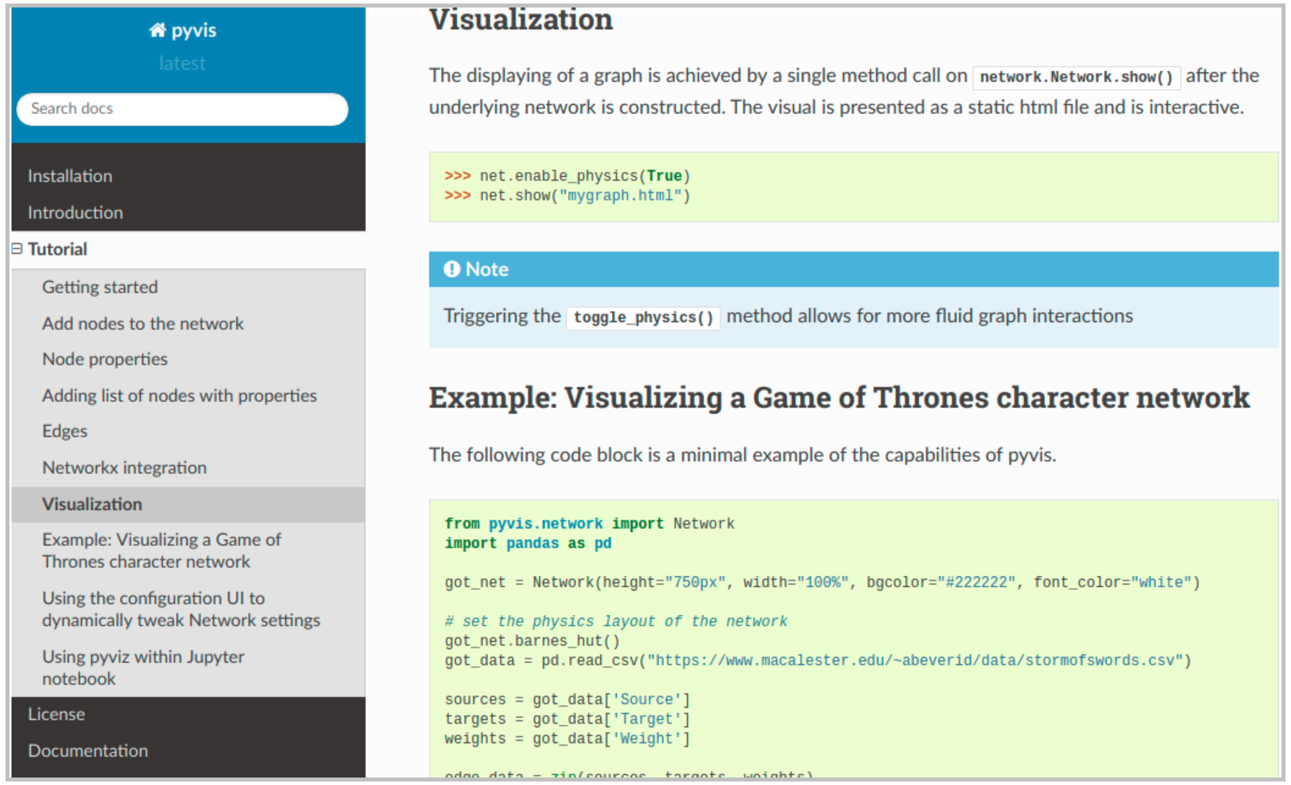
```
print(edgelist_df.shape)
```

```
(234, 3)
```

- **234 rows** means that there are 234 edges
- All 248 documents are still being accounted for, and some share edges

# Create network with pyvis

- To create a network plot of the data we have, we will use the `pyvis` package and its `network` module we imported earlier
- For more information on this package, detailed documentation, and base examples, visit ***this page***



pyvis
latest
Search docs

Installation
Introduction
⊟ Tutorial
Getting started
Add nodes to the network
Node properties
Adding list of nodes with properties
Edges
Networkx integration
Visualization
Example: Visualizing a Game of Thrones character network
Using the configuration UI to dynamically tweak Network settings
Using pyviz within Jupyter notebook
License
Documentation

**Visualization**

The displaying of a graph is achieved by a single method call on `network.Network.show()` after the underlying network is constructed. The visual is presented as a static html file and is interactive.

```
>>> net.enable_physics(True)
>>> net.show("mygraph.html")
```

**Note**

Triggering the `toggle_physics()` method allows for more fluid graph interactions

**Example: Visualizing a Game of Thrones character network**

The following code block is a minimal example of the capabilities of pyvis.

```
from pyvis.network import Network
import pandas as pd

got_net = Network(height="750px", width="100%", bgcolor="#222222", font_color="white")

# set the physics layout of the network
got_net.barnes_hut()
got_data = pd.read_csv("https://www.macalester.edu/~abeverid/data/stormofswords.csv")

sources = got_data['Source']
targets = got_data['Target']
weights = got_data['Weight']

edge_data = zip(sources, targets, weights)
```

# Create network and set base parameters

- To initialize a network visualization object, we simply call a `Network` function and give it the following parameters:
  - `height`: height of the graph either in `%` of your screen size or in pixels
  - `width`: width of the graph either in `%` of your screen size or in pixels
  - `bgcolor`: background color in `RGB` hexadecimal code (e.g. `#FFFFFF` stands for white, you can get RGB hex codes ***here***) or in words (i.e. `white`)
  - `font_color`: font color in `RGB` hexadecimal code (e.g. `#000000` stands for black) or in words (i.e. `black`)
- We can add the **physics** parameter that controls how the network nodes and edges are arranged and shaped (we will show you how to adjust them later)
- We can also make the **edges smooth** using **set_edge_smooth()**. When the edges are made smooth, they are drawn as curves and this makes it look better.

## Create network and set base parameters (cont'd)

```
# Create an empty network object.
network = Network(height="100%",
                  width="60%",
                  bgcolor="#FFFFFF",
                  font_color="#000000")

# Set the physics layout of the network.
network.force_atlas_2based()
network.set_edge_smooth('dynamic')
print(network)
```

```
{
    "Nodes": [],
    "Edges": [],
    "Height": "100%",
    "Width": "60%",
    "Heading": ""
}
```

- Our current network is empty, but is set and ready!

# Populate network with edge and node data

- The empty network now needs to be populated with data about nodes and edges
- We will use our `edgelist_df` to do it
- Let's zip the three necessary columns `source`, `target`, and `weight` into an iterable set of tuples
- We will loop through each tuple and add nodes to our network one by one

```
# Zip columns of edgelist data into a set of tuples.
edge_data = zip(edgelist_df['source'], edgelist_df['target'], edgelist_df['weight'])

# Iterate through the edge data.
for e in edge_data:
    src = e[0] #<- get the source node
    dst = e[1] #<- get the destination (i.e. target node)
    w = e[2]   #<- get the weight of the edge

    # Add a source node with its information.
    network.add_node(src, src, title = src)
    # Add a destination node with its information.
    network.add_node(dst, dst, title = dst)
    # Add an edge between source and destination nodes with weight w.
    network.add_edge(src, dst, value = w)
```

## Inspect populated network

- To view the nodes, just print `network.nodes` of your choice

```
print(network.nodes[0:5])
```

```
[{'title': 0, 'id': 0, 'label': 0, 'shape': 'dot', 'font': {'color': '#000000'}}, {'title': 17, 'id':
17, 'label': 17, 'shape': 'dot', 'font': {'color': '#000000'}}, {'title': 60, 'id': 60, 'label': 60,
'shape': 'dot', 'font': {'color': '#000000'}}, {'title': 2, 'id': 2, 'label': 2, 'shape': 'dot',
'font': {'color': '#000000'}}, {'title': 234, 'id': 234, 'label': 234, 'shape': 'dot', 'font':
{'color': '#000000'}}]
```

- To view the edges, just print `network.edges` of your choice

```
print(network.edges[0:5])
```

```
[{'value': 0.4024067223072052, 'from': 0, 'to': 17}, {'value': 0.41239631175994873, 'from': 0, 'to':
60}, {'value': 0.4199207127094269, 'from': 2, 'to': 234}, {'value': 0.4199207127094269, 'from': 2,
'to': 219}, {'value': 0.6168836951255798, 'from': 3, 'to': 58}]
```

## Get neighbor map for each node

- Wouldn't it be nice to know the neighbors (i.e. nodes that are directly connected to a given node) of each node?
- Those nodes are the documents most similar to a given document

```
# Get a list of node neighbors.
neighbor_map = network.get_adj_list()
```

- For instance, to see the document IDs that are most similar to document `17`, we can look at the neighbor map like so:

```
# Show documents most similar to document 17.
print(neighbor_map[17])
```

```
{0, 140, 236, 141}
```

# Add similarity information into the hover over

- Since the `pyvis.network` graph is fully interactive, we can hover over the nodes which represent documents and add any information to the tooltip
- Let's add information about the most similar documents to each of the documents (i.e. nodes)
  - `node["title"]` is the text that appears when hovering over the node
  - it can be simple text or `HTML` code to look nice
- The code below includes a bit of `HTML`, and we will just show you how to add it to make the tooltip look nice and formatted (you don't have to be able to do it on your own)

```
# Add neighbor data to node hover data.
for node in network.nodes:
    title = "Most similar articles: <br>"
    neighbors = list(neighbor_map[node["id"]])
    title = title + "<br>".join(str(neighbor) for neighbor in neighbors)
    node["title"] = title
```

```
print(network.nodes[0])
```

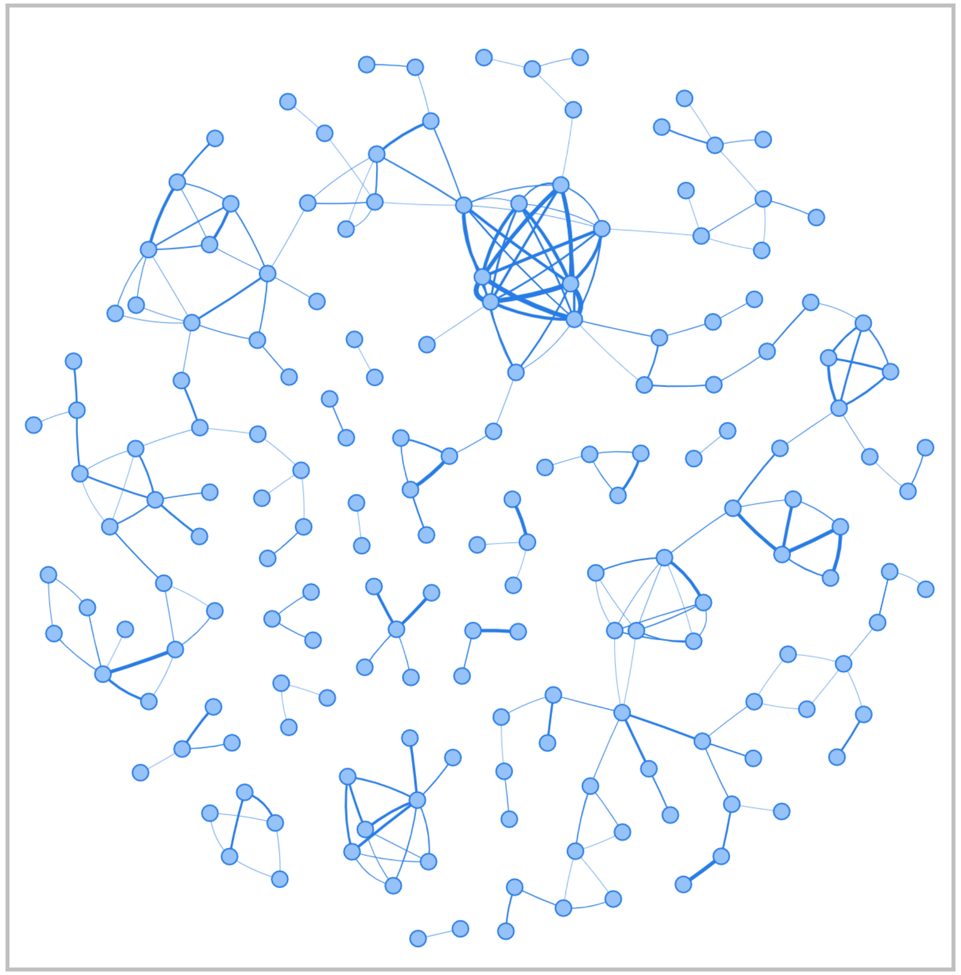
```
{'title': 'Most similar articles: <br>17<br>60', 'id': 0, 'label': 0, 'shape': 'dot', 'font': {'color':
'#000000'}}
```

# Save and show the network graph

- To view and save the plot, you can use the `.show()` function with a path and name of the document where you would like to save your interactive graph

```
# Save html and show graph in browser.
network.show(plot_dir + "/NYT_similar_snippets.html")
```

- The graph should now open in your browser
- If it doesn't, you can go to your `plots` folder and open the `NYT_similar_snippets.html` as a normal webpage



# Zooming into the graph

- You can zoom in and out of the nodes and edges by using the wheel on your mouse or stretching out the trackpad on your laptop
- You can also drag the nodes around to reposition them (it serves little purpose, but it's fun!)

## Hovering over the nodes

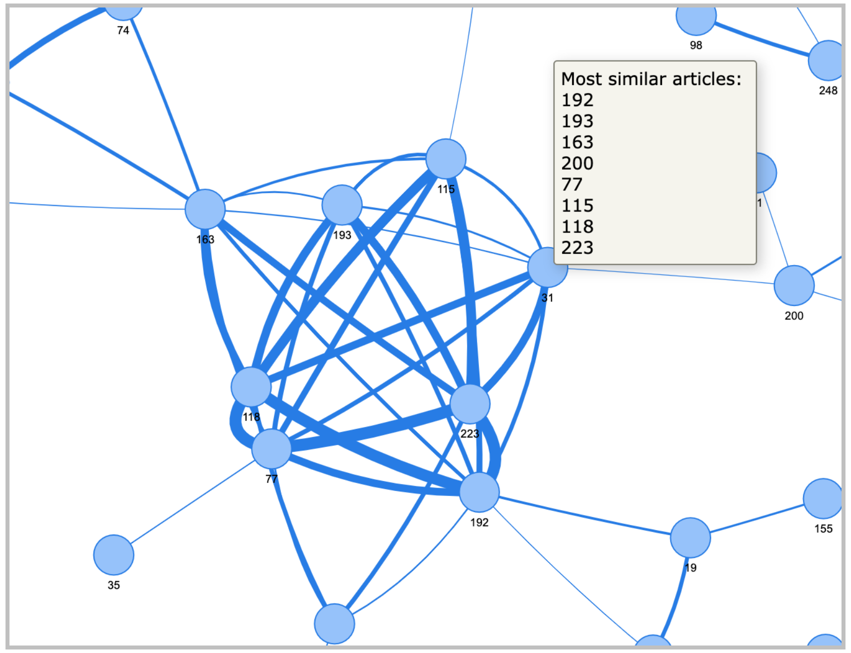
- When you hover over a node, you can see a list of all of its neighbors (i.e., the most similar articles)

# What does "most similar articles" mean?

- Those are the articles (i.e., `target` nodes) from our `edgelist`
- Each pair of linked nodes correspond to the document-document pairs, with their respective cosine similarity scores represented by the `weight` of the edge
- The thicker the edge, the higher the similarity, the closer the two articles in meaning!

```
edgelist_df_subset = edgelist_df.query("source==31")
print(edgelist_df_subset)
```

```
      source  target    weight
1321      31      77  0.539880
1326      31     115  0.485577
1328      31     118  0.659343
1330      31     200  0.401721
1331      31     192  0.526407
1338      31     223  0.659343
1342      31     193  0.447311
1346      31     163  0.423019
```



Most similar articles:
192
193
163
200
77
115
118
223

# Look up the similar article pairs

- Let's look up the most similar article(s) to 31
- In this case, there are two: 118 and 223

```
print(edgelist_df_subset)
```

```
      source  target    weight
1321      31      77  0.539880
1326      31     115  0.485577
1328      31     118  0.659343
1330      31     200  0.401721
1331      31     192  0.526407
1338      31     223  0.659343
1342      31     193  0.447311
1346      31     163  0.423019
```

- We can see that these three articles are similar, because their snippets all start and end the same way: *The Latest on ... (all times local)*
- How can we ensure our similarity score is not affected by "junk" words like that? *Text cleanup that removes stopwords and frequently occuring words*

```
print(NYT.iloc[31, 2])
```

```
The Latest on a regional meeting about
Venezuela's political crisis. (all
times local):
```

```
print(NYT.iloc[118, 2])
```

```
The Latest on the fatal shooting of a
7-year-old girl in Houston (all times
local):
```

```
print(NYT.iloc[223, 2])
```

```
The Latest on California Gov.-elect
Gavin Newsom's inauguration (all times
local):
```

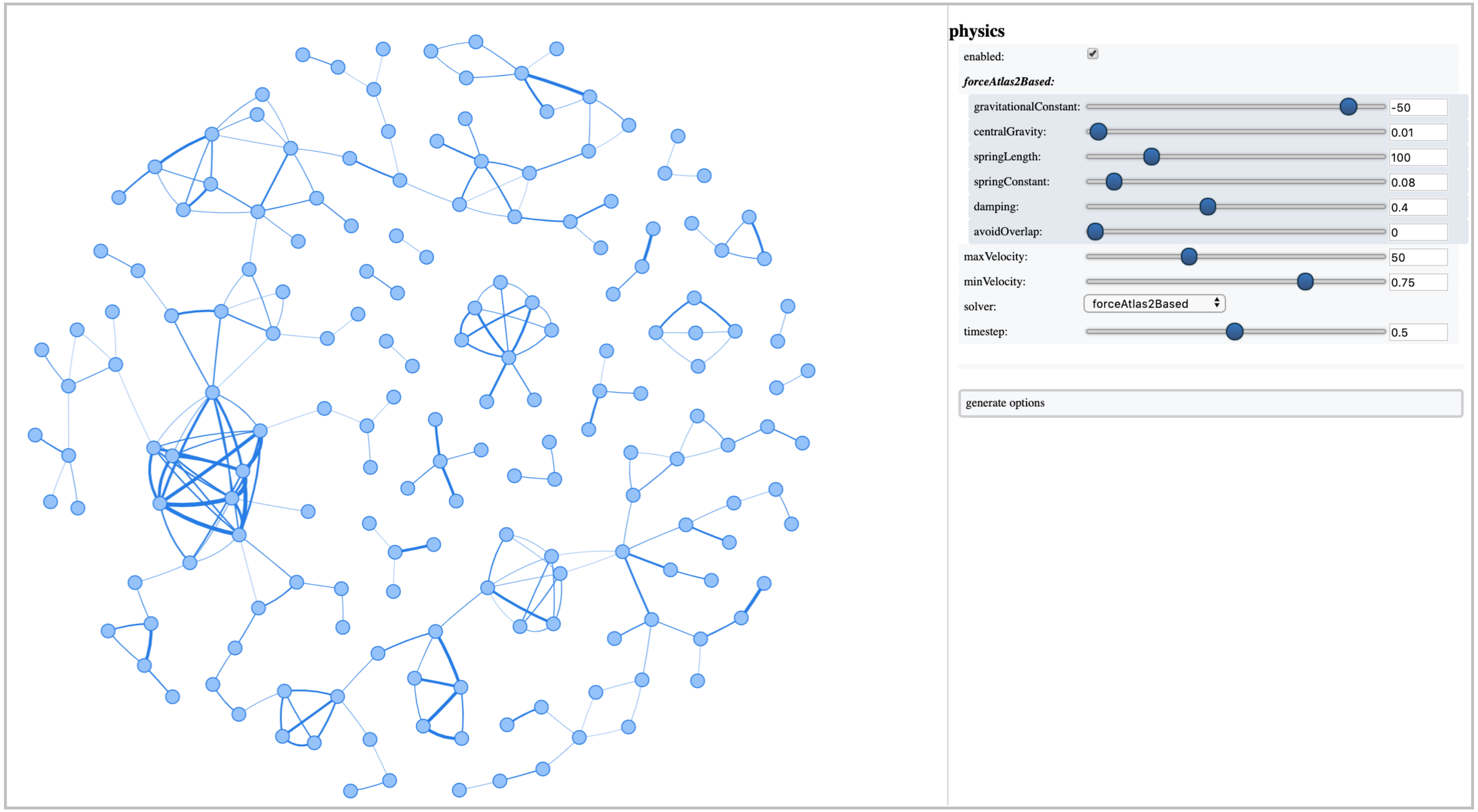
# Show buttons to modify graph look

- If you would like to modify the way the graph looks, you can use `.show_buttons()` function and pass these parameters to it:
  - *'nodes'*
  - *'edges'*
  - *'physics'*
- Let's show buttons for *'physics'* options and resave the graph

```
# Show buttons to modify the look.
network.show_buttons(filter_=['physics'])
```

```
# Save html and show graph in browser.
network.show(plot_dir+"/NYT_similar_snippets.html")
```

# Modify the graph physics



**physics**

enabled: ✓

***forceAtlas2Based:***

| | |
|---|---|
| gravitationalConstant: | -50 |
| centralGravity: | 0.01 |
| springLength: | 100 |
| springConstant: | 0.08 |
| damping: | 0.4 |
| avoidOverlap: | 0 |
| maxVelocity: | 50 |
| minVelocity: | 0.75 |
| solver: | forceAtlas2Based |
| timestep: | 0.5 |

## Module completion checklist

| Objective | Complete |
|---|---|
| Visualize similar documents using interactive network graph | ✔ |
| Explain hierarchical clustering (hclust) and how it applies to text | |
| Build out hclust model on text data | |
| Visualize clustering results as a dendrogram and compare results with LDA | |

# What is clustering?

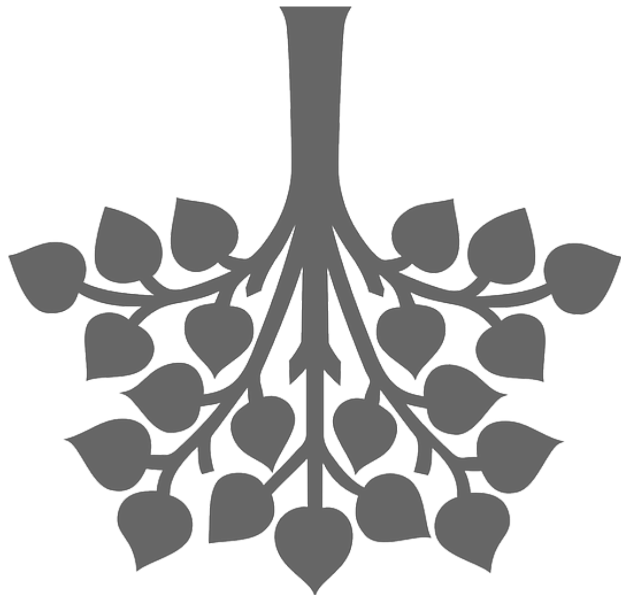
- **Clustering** is an **unsupervised machine learning** technique for **finding groups or clusters** of similar observations in our data
- It is useful when we have **unlabeled data** and not much information about its structure
- The way a particular clustering method works and the output it produces depends on the **distance measure** used
  - *Euclidean distance* is used most commonly, but since we are comparing text documents, we will use *cosine distance* instead

# What is hclust?

- `hclust` is a host of algorithms under a common name known as **hierarchical clustering** that allow us to look for patterns within our data
- It produces a **hierarchy of clusters** of the data without any prior knowledge about its initial structure
- It is solely based on the **distance** between observations

# Agglomerative clustering

- We will be working with **agglomerative clustering**
  - This kind of hierarchical clustering starts by grouping the most similar observations called **leaves** and connects them into **branches**
  - It then connecting smaller branches into bigger ones all the way until it reaches the **root**
- A common way to view the results of `hclust` is to build a **dendrogram** (i.e. a **tree diagram**), which shows how observations are grouped together



# Hierarchical clustering: compute distance

- We already have the `similarity` matrix of our documents
- To get a *distance* matrix, we simply need to subtract `similarity` from `1`

```
# Compute distance matrix by subtracting similarity from 1.
distance = 1 - similarity
```

# Cosine distance matrix characteristics

| | doc_0 | doc_1 | doc_2 |
|---|---|---|---|
| doc_0 | **0** | 1 | 0.979 |
| doc_1 | 1 | **0** | 0.11 |
| doc_2 | 0.979 | 0.11 | **0** |

- **It is *square*:** the number of rows is the same as the number of columns
- **It is *symmetric*:** the entries above the diagonal are mirror images of entries below
- Values along the **diagonal are always** 0

*Note: This is only true when comparing documents, sentences, words, etc to each other. If we make a matrix of document-to-query distance, for instance, the matrix is going to be neither square, nor symmetric, nor will it have 1's on the diagonal!*

# Cosine distance matrix: interpretation

- When we want to compute cosine distance between **documents**, we give the function a `DTM` that has `248` rows
  - As a result, we get `248x248` distance matrix
  - Each row and each column represents a document
  - The value at the intersection is the distance score between them
- If we wanted to compute cosine distance between **terms**, what would we give as input to the function and what would be the output?
- Why do you think we have all `0`s along the diagonal?
- Why do you think the values above the diagonal are mirror images of the values below the diagonal?

# Cosine distance matrix: interpretation

- 
  - If we wanted to compute cosine distance between **terms**, what would we give as input to the function and what would be the output?
    *Input: TDM that has 1921 rows, Output: 1921X1921 Distance matrix Each row here represents a 'term' present in the corpus*
  - Why do you think we have all `0`s along the diagonal?
    *Hint: cosine distance of `0` means the documents are perfectly similar, `1` means they are completely dissimilar!*
  - Why do you think the values above the diagonal are mirror images of the values below the diagonal?
    *Hint: The matrix is a distance representation of each term against every other term*

# Which one is which?

| | doc_0 | doc_1 | doc_2 |
|---|---|---|---|
| **doc_0** | **0** | 1 | 0.979 |
| **doc_1** | 1 | **0** | 0.11 |
| **doc_2** | 0.979 | 0.11 | **0** |

| | doc_0 | doc_1 | doc_2 |
|---|---|---|---|
| **doc_0** | **1** | 0 | 0.021 |
| **doc_1** | 0 | **1** | 0.89 |
| **doc_2** | 0.021 | 0.89 | **1** |

## Answers

**distance matrix**

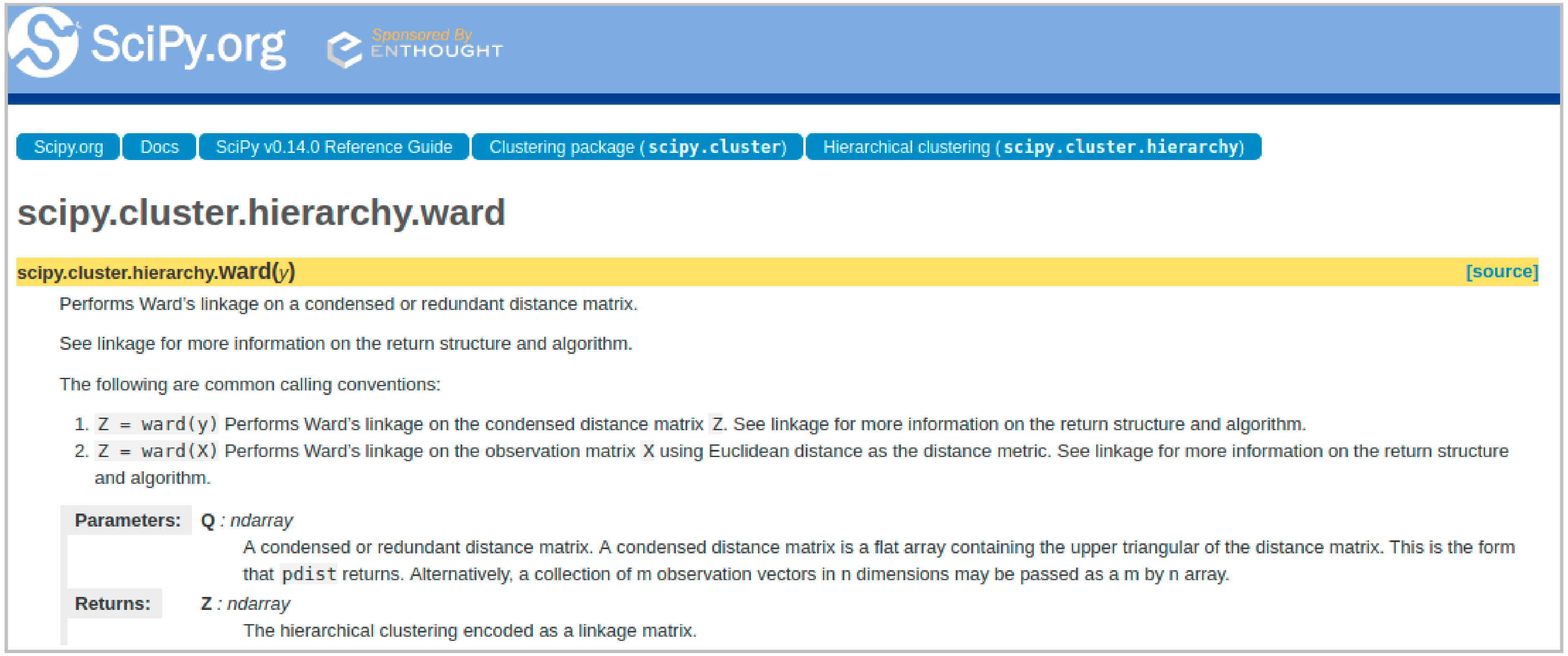
| | doc_0 | doc_1 | doc_2 |
|---|---|---|---|
| **doc_0** | **0** | 1 | 0.979 |
| **doc_1** | 1 | **0** | 0.11 |
| **doc_2** | 0.979 | 0.11 | **0** |

**similarity matrix**

| | doc_0 | doc_1 | doc_2 |
|---|---|---|---|
| **doc_0** | **1** | 0 | 0.021 |
| **doc_1** | 0 | **1** | 0.89 |
| **doc_2** | 0.021 | 0.89 | **1** |

# Hierarchical clustering: ward linkage method

- To compute the linkage matrix, we will use the `ward` method, developed by **Joe H. Ward, Jr.**
- He suggested a **minimum variance method** to link two leaves or branches into clusters
- For more information about `ward` and other linkage algorithms, visit `scipy` ***documentation***


SciPy.org
Sponsored By ENTHOUGHT
Scipy.org
Docs
SciPy v0.14.0 Reference Guide
Clustering package (scipy.cluster)
Hierarchical clustering (scipy.cluster.hierarchy)

scipy.cluster.hierarchy.ward

scipy.cluster.hierarchy.**ward**(*y*) [source]

Performs Ward's linkage on a condensed or redundant distance matrix.

See linkage for more information on the return structure and algorithm.

The following are common calling conventions:

1. `Z = ward(y)` Performs Ward's linkage on the condensed distance matrix `Z`. See linkage for more information on the return structure and algorithm.
2. `Z = ward(X)` Performs Ward's linkage on the observation matrix `X` using Euclidean distance as the distance metric. See linkage for more information on the return structure and algorithm.

**Parameters:** **Q** : *ndarray*

A condensed or redundant distance matrix. A condensed distance matrix is a flat array containing the upper triangular of the distance matrix. This is the form that `pdist` returns. Alternatively, a collection of m observation vectors in n dimensions may be passed as a m by n array.

**Returns:** **Z** : *ndarray*

The hierarchical clustering encoded as a linkage matrix.

# Knowledge check 1

## Exercise 1

# Module completion checklist

| Objective | Complete |
|---|---|
| Visualize similar documents using interactive network graph | ✔ |
| Explain hierarchical clustering (hclust) and how it applies to text | ✔ |
| Build out hclust model on text data | |
| Visualize clustering results as a dendrogram and compare results with LDA | |

# Compute and interpret linkage matrix

```
# Define the `linkage_matrix` using
# `ward` clustering algorithm.
linkage_matrix = ward(distance)
print(linkage_matrix[0:10])
```

```
[[  0. 139.   0.   2.]
 [  1. 140.   0.   2.]
 [  2. 146.   0.   2.]
 [  3. 141.   0.   2.]
 [  4. 142.   0.   2.]
 [  5. 145.   0.   2.]
 [  6. 147.   0.   2.]
 [ 22.  52.   0.   2.]
 [126. 255.   0.   3.]
 [192. 256.   0.   4.]]
```

- A **linkage matrix** is a matrix where each row represents the 2 clusters connected to each other and information about them in the following format: `[id_1, id_2, dist, sample_count]`
  - `id_1` and `id_2` are the IDs of the 1st and 2nd clusters respectively
  - `dist` is the distance between the two clusters
  - `sample_count` is the number of observations that were in the resulting cluster
- **Why do you think the distance is `0` for the points that we see?**

# Compute and interpret linkage matrix (cont'd)

```
# Print shape of the matrix.
print(linkage_matrix.shape)
```

```
(247, 4)
```

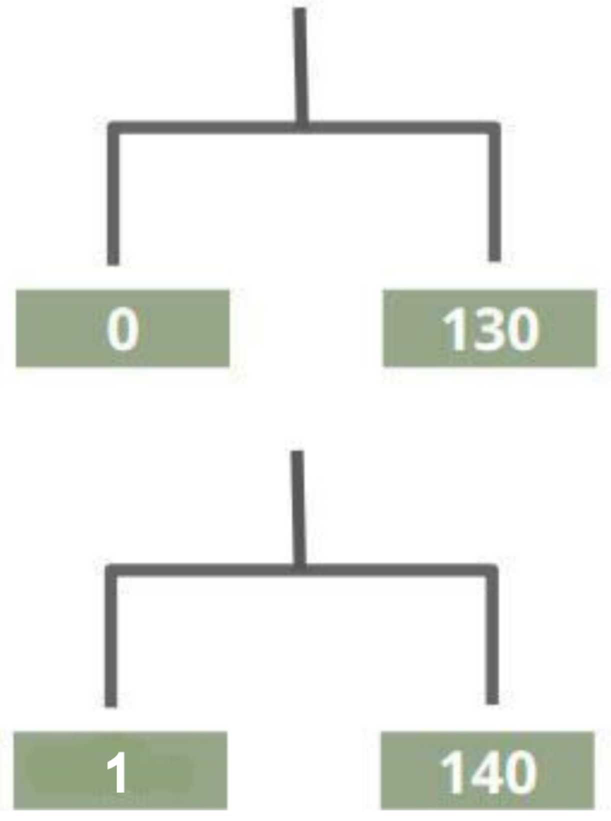
- The shape of the linkage matrix will always be `(n-1) x 4`
  - `n` is the number of observations in data: since we have `248` article snippets, the linkage matrix has `248 - 1 = 247` rows
  - `4` is the number of columns which are described in the previous slide

# Compute and interpret linkage matrix (cont'd)

```
print(linkage_matrix[0:2])
```

```
[[  0. 139.    0.    2.]
 [  1. 140.    0.    2.]]
```

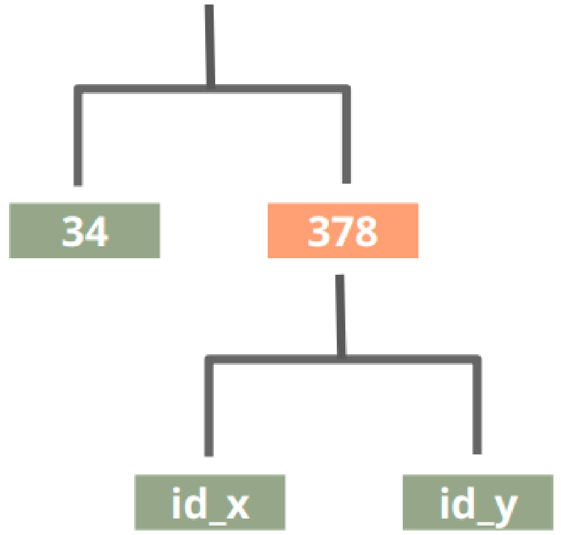
- We can see that the first link that the algorithm made was between clusters `0` and `139`, the distance between them is `0`, and the newly formed cluster contains a total of `2` observations
- The second link that the algorithm made was between clusters `1` and `140`, the distance between them is `0`, and the newly formed cluster contains a total of `2` observations
- This means that, in its first steps, the algorithm was simply merging observations into 2-sample clusters and the cluster IDs are nothing more than our observation IDs



# Compute and interpret linkage matrix (cont'd)

```
print(linkage_matrix[150])
```

```
[ 34.          378.
1.86279919    3.          ]
```

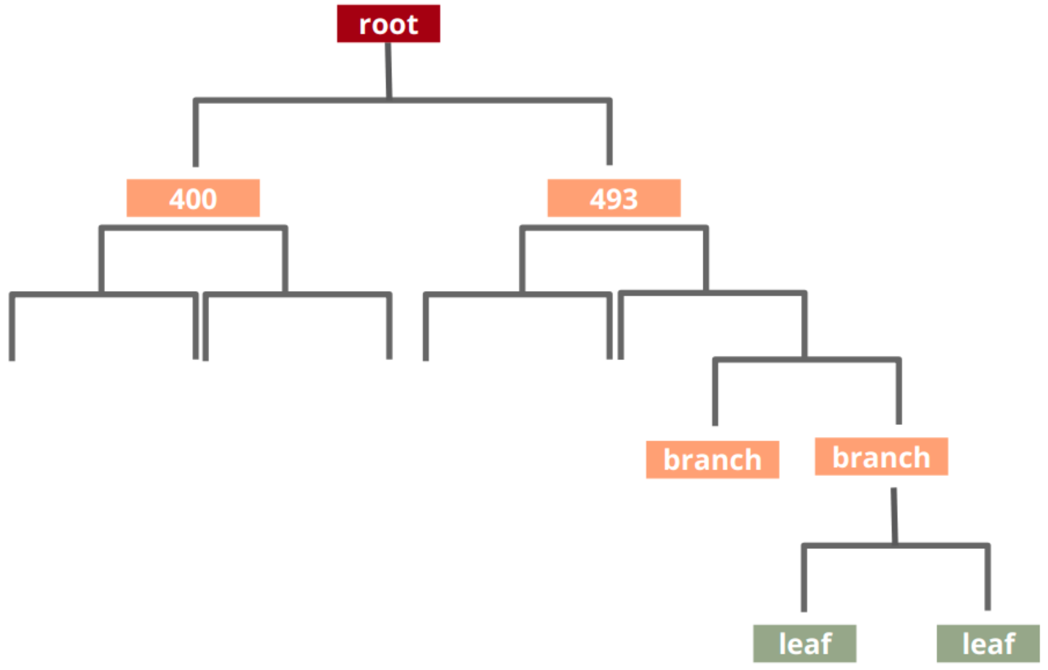


- The 151st link that the algorithm made was between clusters `34` and `378`, with distance of about `1.86` and the number of observations in the new cluster being `3`
- This means that it merged observation `34` with an already formed cluster of `2` observations
- The general formula for a cluster number is `which_cluster = id_cluster - num_observations`
- The ID of clusters of `2` observations is easy to trace: in this case, it is cluster `130 = 378 - 248`, i.e. the 130th cluster formed by the algorithm, or the cluster ID minus the total number of observations

# Compute and interpret linkage matrix (cont'd)

```
print(linkage_matrix[246])
```

```
[400.         493.           8.22551818 248.        
]
```



- The very last cluster formed is from clusters `400` and `493` with distance between them being approximately `8.23` and the total number of observations within the newly formed cluster being `248`
- This means that the algorithm linked `2` big clusters: `152 = 400 - 248` and `245 = 493 - 248` and made one giant cluster of `248` observations, which is our entire dataset
- The **last** cluster in the linkage matrix always includes all observations and is called a **root** of our hierarchy (a.k.a. a root of a tree)

## Module completion checklist

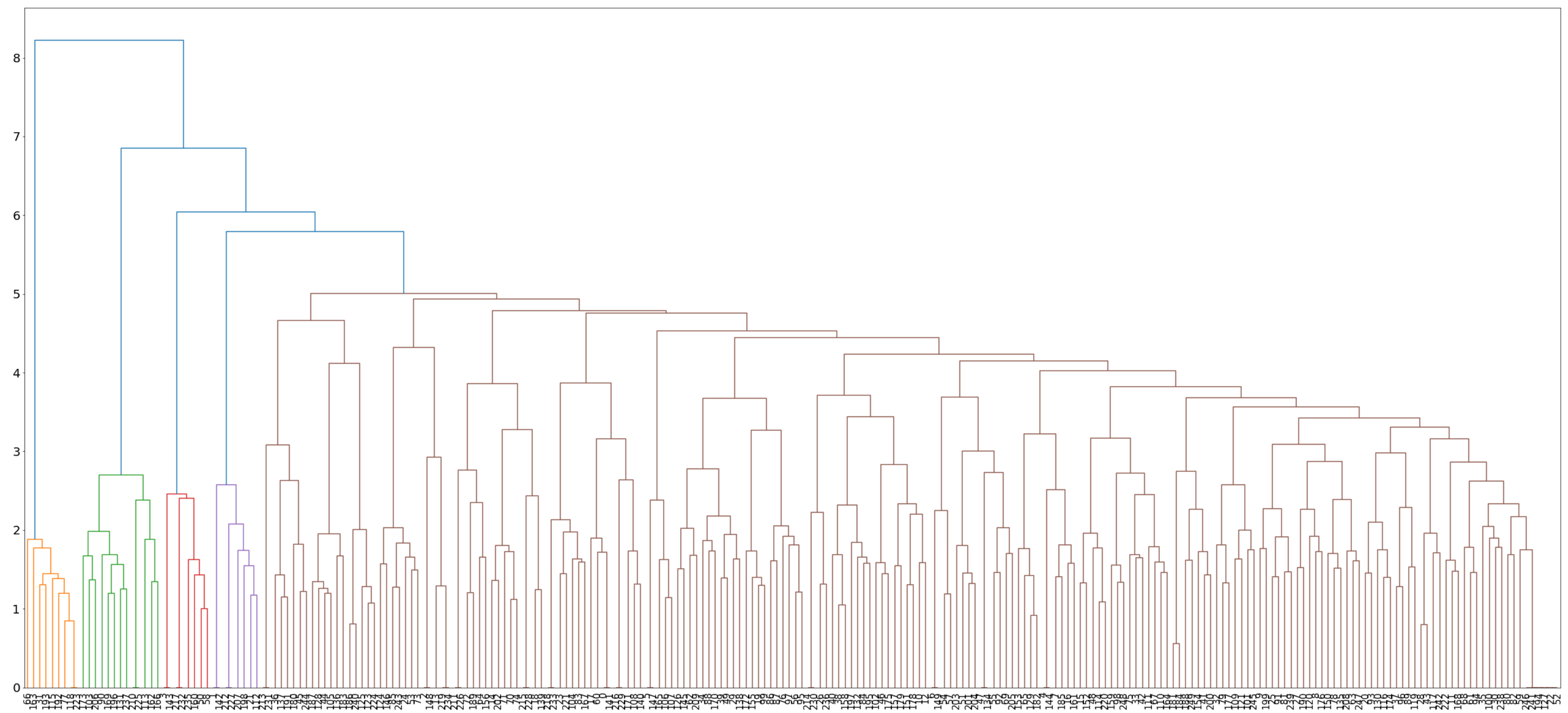
| Objective | Complete |
|---|---|
| Visualize similar documents using interactive network graph | ✔ |
| Explain hierarchical clustering (hclust) and how it applies to text | ✔ |
| Build out hclust model on text data | ✔ |
| Visualize clustering results as a dendrogram and compare results with LDA | |

## Hierarchical clustering: visualize results

- Now let's try plotting the hierarchical clusters as a dendrogram.

```
# Now we can plot the hierarchical clusters.
fig, axes = plt.subplots(figsize = (15, 40))
axes = dendrogram(linkage_matrix,
                  orientation = "right",
                  labels = valid_snippets,
                  leaf_font_size = 11)
```

# Hierarchical clustering: visualize results



# Hierarchical clustering: get cluster labels

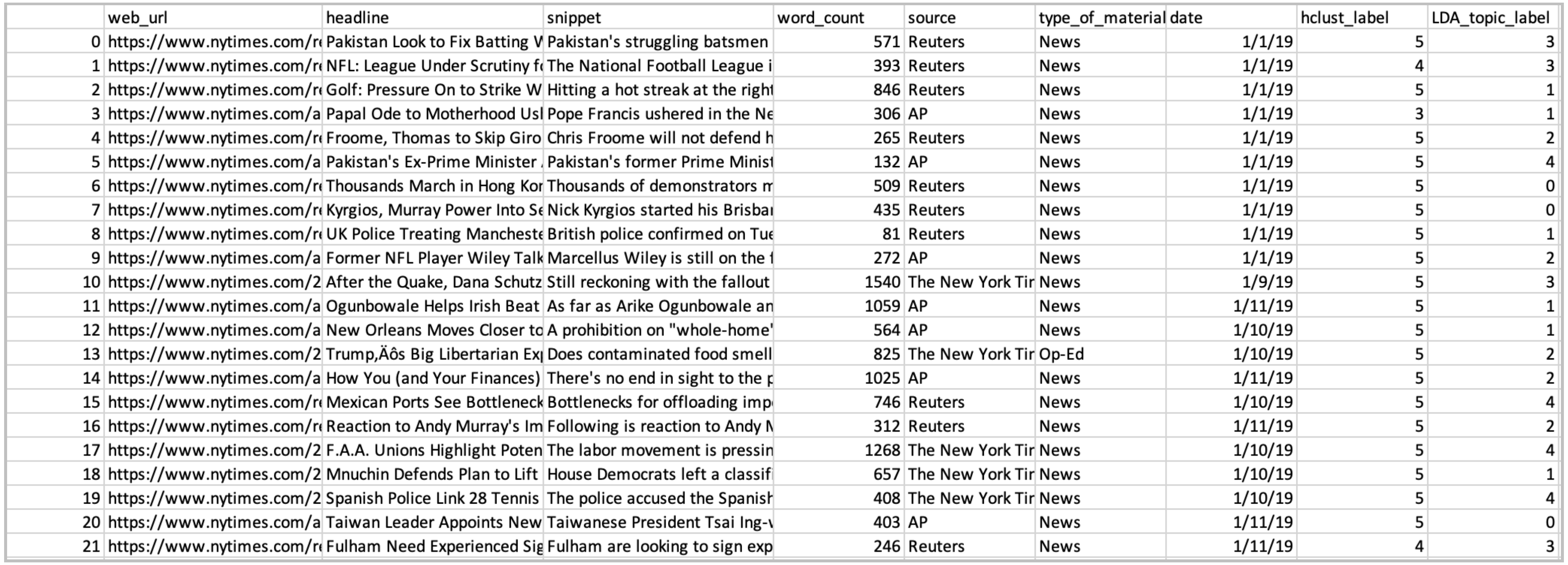
- You can split your dendrogram and extract cluster labels for each cluster by using the `fcluster` function
- It takes the `linkage` object, the maximum number of clusters, and the criterion by which to split the dendrogram (i.e. `maxclust` is what we need)

```
# Set k - the max number of clusters.
k = 5
# Get cluster labels for each snippet.
cluster_labels = fcluster(linkage_matrix,         #<- linkage matrix
                          k,                      #<- max number of clusters
                          criterion = 'maxclust') #<- criterion maxclust
print(cluster_labels)
```

```
[5  4  5  3  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  4  5  5  5  5  5  5  5  5  5  1  5  5  5  5  5
 5  5  5  5  5  5  5  5  5  5  5  5  5  3  5  5  5  5  5  5  5  3  5  5  5  5  5  5  5  1  5  5  5  5  5  5  5
 5  5  5  1  5  5  5  5  5  5  5  5  5  5  5  5  2  5  5  5  5  5  5  5  5  5  5  5  5  2  5  5  5  5  5  5  5
 5  4  2  5  1  5  5  1  5  5  5  5  5  5  5  5  5  5  5  2  5  5  5  5  5  5  5  5  5  4  3  5  5  5  5  5  5
 5  5  5  5  5  5  5  5  5  5  3  5  2  1  5  5  2  5  5  2  5  5  5  2  5  5  5  5  5  5  5  5  5  5  5  5  5
 5  5  5  5  5  1  1  5  5  2  5  4  5  5  5  5  5  5  5  2  4  5  5  2  5  4  5  5  5  5  3  5  5  5  5  5  1
 5  2  5  4  5  5  5  5  3  5  5  3  5  2  5  5  5  5  5  5  5  5  5  5  5  5]
```

# Combine data with hclust and LDA cluster labels

```
NYT_valid_articles = NYT.loc[valid_snippets]
NYT_valid_articles['hclust_label'] = cluster_labels
doc_topic_pairs_df = doc_topic_pairs_df.sort_values(by = "doc_id")
NYT_valid_articles['LDA_topic_label'] = doc_topic_pairs_df['best_topic']
```



| | web_url | headline | snippet | word_count | source | type_of_material | date | hclust_label | LDA_topic_label |
|---|---|---|---|---|---|---|---|---|---|
| 0 | https://www.nytimes.com/r | Pakistan Look to Fix Batting V | Pakistan's struggling batsmen | 571 | Reuters | News | 1/1/19 | 5 | 3 |
| 1 | https://www.nytimes.com/r | NFL: League Under Scrutiny f | The National Football League i | 393 | Reuters | News | 1/1/19 | 4 | 3 |
| 2 | https://www.nytimes.com/r | Golf: Pressure On to Strike W | Hitting a hot streak at the right | 846 | Reuters | News | 1/1/19 | 5 | 1 |
| 3 | https://www.nytimes.com/a | Papal Ode to Motherhood Usl | Pope Francis ushered in the Ne | 306 | AP | News | 1/1/19 | 3 | 1 |
| 4 | https://www.nytimes.com/r | Froome, Thomas to Skip Giro | Chris Froome will not defend h | 265 | Reuters | News | 1/1/19 | 5 | 2 |
| 5 | https://www.nytimes.com/a | Pakistan's Ex-Prime Minister | Pakistan's former Prime Minist | 132 | AP | News | 1/1/19 | 5 | 4 |
| 6 | https://www.nytimes.com/r | Thousands March in Hong Kor | Thousands of demonstrators m | 509 | Reuters | News | 1/1/19 | 5 | 0 |
| 7 | https://www.nytimes.com/r | Kyrgios, Murray Power Into Se | Nick Kyrgios started his Brisbar | 435 | Reuters | News | 1/1/19 | 5 | 0 |
| 8 | https://www.nytimes.com/r | UK Police Treating Mancheste | British police confirmed on Tue | 81 | Reuters | News | 1/1/19 | 5 | 1 |
| 9 | https://www.nytimes.com/a | Former NFL Player Wiley Talk | Marcellus Wiley is still on the | 272 | AP | News | 1/1/19 | 5 | 2 |
| 10 | https://www.nytimes.com/2 | After the Quake, Dana Schutz | Still reckoning with the fallout | 1540 | The New York Tir | News | 1/9/19 | 5 | 3 |
| 11 | https://www.nytimes.com/a | Ogunbowale Helps Irish Beat | As far as Arike Ogunbowale an | 1059 | AP | News | 1/11/19 | 5 | 1 |
| 12 | https://www.nytimes.com/a | New Orleans Moves Closer to | A prohibition on "whole-home' | 564 | AP | News | 1/10/19 | 5 | 1 |
| 13 | https://www.nytimes.com/2 | Trump,Äôs Big Libertarian Ex | Does contaminated food smell | 825 | The New York Tir | Op-Ed | 1/10/19 | 5 | 2 |
| 14 | https://www.nytimes.com/a | How You (and Your Finances) | There's no end in sight to the p | 1025 | AP | News | 1/11/19 | 5 | 2 |
| 15 | https://www.nytimes.com/r | Mexican Ports See Bottleneck | Bottlenecks for offloading imp | 746 | Reuters | News | 1/10/19 | 5 | 4 |
| 16 | https://www.nytimes.com/r | Reaction to Andy Murray's Im | Following is reaction to Andy M | 312 | Reuters | News | 1/11/19 | 5 | 2 |
| 17 | https://www.nytimes.com/2 | F.A.A. Unions Highlight Poten | The labor movement is pressin | 1268 | The New York Tir | News | 1/10/19 | 5 | 4 |
| 18 | https://www.nytimes.com/2 | Mnuchin Defends Plan to Lift | House Democrats left a classifi | 657 | The New York Tir | News | 1/10/19 | 5 | 1 |
| 19 | https://www.nytimes.com/2 | Spanish Police Link 28 Tennis | The police accused the Spanish | 408 | The New York Tir | News | 1/10/19 | 5 | 4 |
| 20 | https://www.nytimes.com/a | Taiwan Leader Appoints New | Taiwanese President Tsai Ing-v | 403 | AP | News | 1/11/19 | 5 | 0 |
| 21 | https://www.nytimes.com/r | Fulham Need Experienced Sig | Fulham are looking to sign exp | 246 | Reuters | News | 1/11/19 | 4 | 3 |

# Save plots and data

```
fig.savefig(plot_dir + '/NYT_hclust.png')
NYT_valid_articles.to_csv(data_dir + '/NYT_snippets_with_cluster_labels.csv')
```

- Once you've saved your data, take a look at it and explore how the articles were grouped into clusters

## Compare results of LDA and hclust

- Let's consider the results of `hclust`:
  - Are clusters approximately even in size?
  - Take one cluster and explore the original articles, are they in fact similar in meaning?
- Let's consider the results of `LDA`:
  - Are topics approximately even in size?
  - Take one topic and explore the original articles, are they in fact similar in meaning?
  - Now take the documents that belonged to the cluster you have picked from `hclust` and compare it to the `LDA` labels; are all documents in that cluster also labeled with the same topic label?
- Which algorithm performed better?

# Knowledge check 2

## Exercise 2

## Next steps

- Here is what you can try out to further improve the results:
  - Run LDA with the optimal number of topics and/or tune other LDA parameters
  - Identify junk words such as “The latest on….(all times local)”, and remove these stopwords in the cleaning steps

## Module completion checklist

| Objective | Complete |
|---|---|
| Visualize similar documents using interactive network graph | ✔ |
| Explain hierarchical clustering (hclust) and how it applies to text | ✔ |
| Build out hclust model on text data | ✔ |
| Visualize clustering results as a dendrogram and compare results with LDA | ✔ |

## This completes our module

**Congratulations!**