Topic Modeling in NLP - Topic modeling - 4

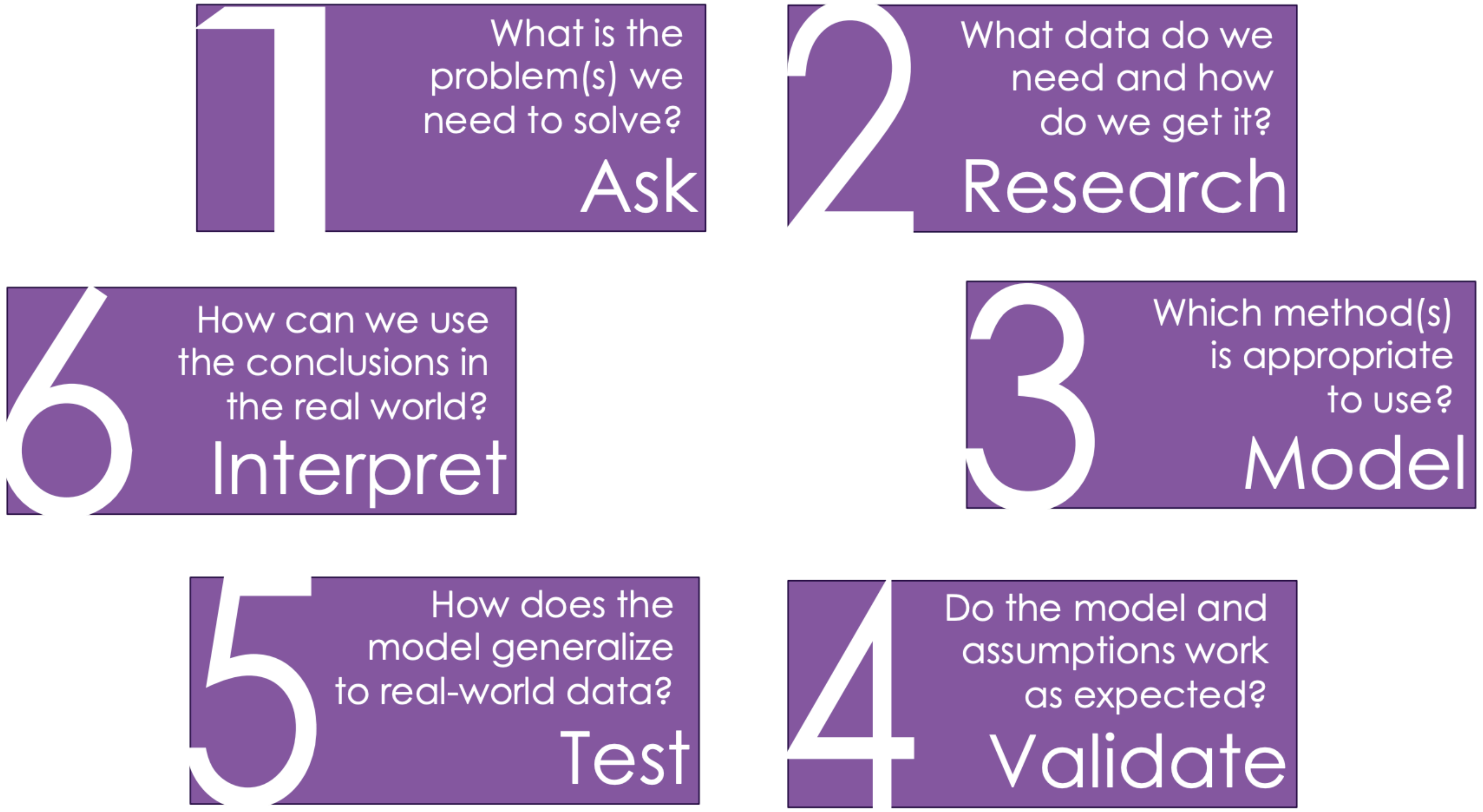
One should look for what is and not what he thinks should be. (Albert Einstein)

Module completion checklist

Objective	Complete
Visualize results of LDA using interactive LDAvis plot	
Extract and interpret document-topic information	

Data wrangling and exploration

- Remember, a data scientist must be able to explore the data to generate a hypothesis
- Clustering and visualization are two great methods to explore and look for patterns in your data!
- In this module, we will use 5 topics for the detailed visualizations



Visualize topics generated with LDA

- We have performed LDA on the `snippet` column of `df` and assessed the numerical metrics of our model's performance
- Now let's look at how all of those metrics and numbers fit together by **visualizing the LDA**
- We will be using `pyLDAvis` package that is a Python wrapper around a very popular R package called `LDAvis`
- You can find the original publication of `LDAvis` ***here***

Visualize topics generated with LDA: pyLDAvis

- We created our LDA model using `gensim`, which integrates easily with `pyLDAvis` through module `pyLDAvis.gensim`
- The method that generates a visualization object is called `pyLDAvis.gensim.prepare()` and it takes the following `gensim` objects as arguments:
  - LDA model object
  - the corpus object
  - the dictionary
- We have created all three in the previous module as the following variables:
  - `lda_model_tfidf`
  - `corpus_tfidf`
  - `dictionary`

Visualize topics generated with LDA

- Let's prepare the visualization object for plotting

```
# Prepare LDA vis object by providing:
vis = pyLDAvis.gensim.prepare(lda_model_tfidf,   #<- model object
                              corpus_tfidf,      #<- corpus object
                              dictionary)        #<- dictionary object
```

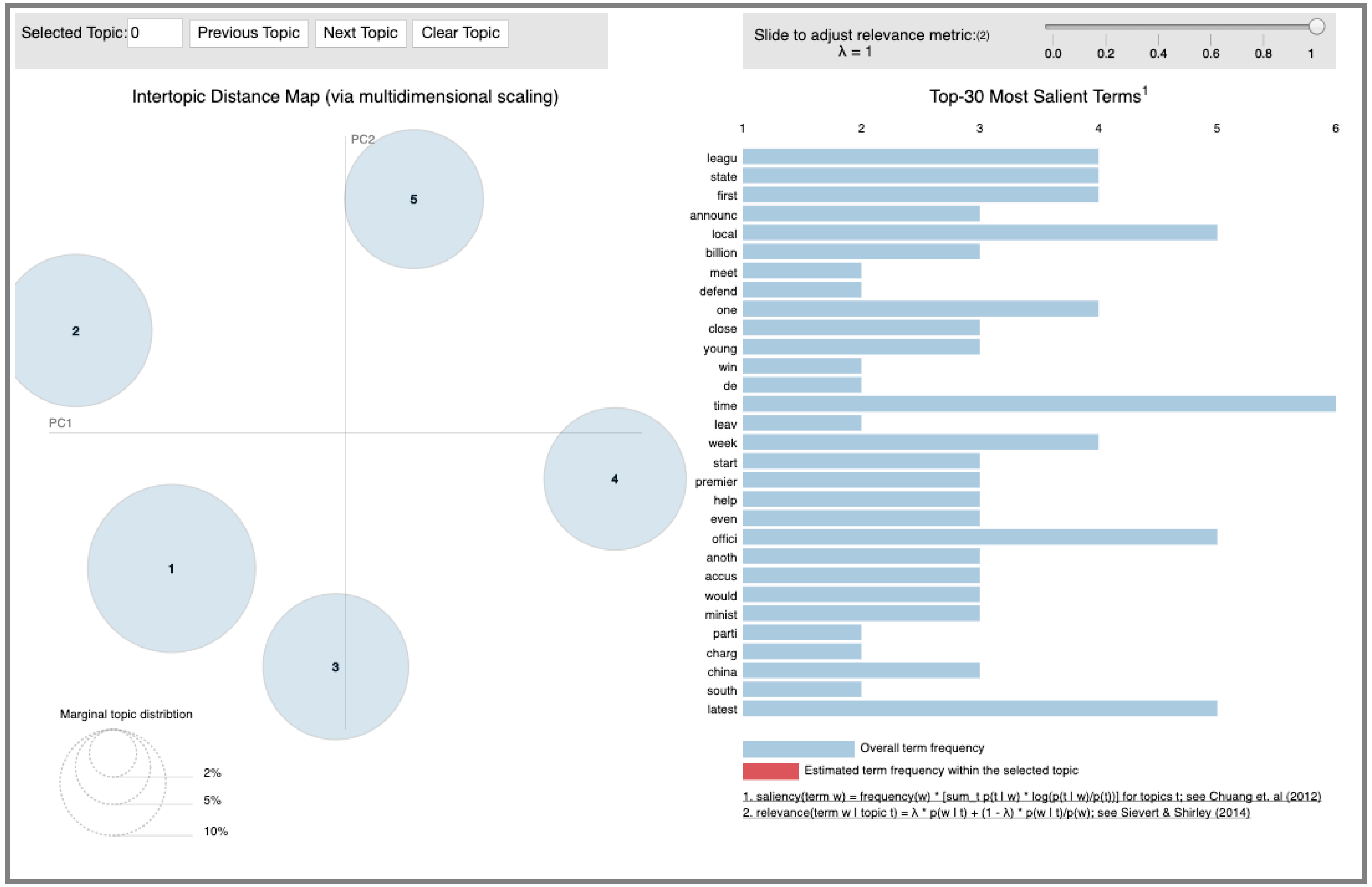
- To display the results in `Jupyter`, you just need to use `pyLDAvis.display()` function

```
# The function takes `vis` object that we prepared above as the main argument.
pyLDAvis.display(vis)
```

- Give the chart a moment to appear and render

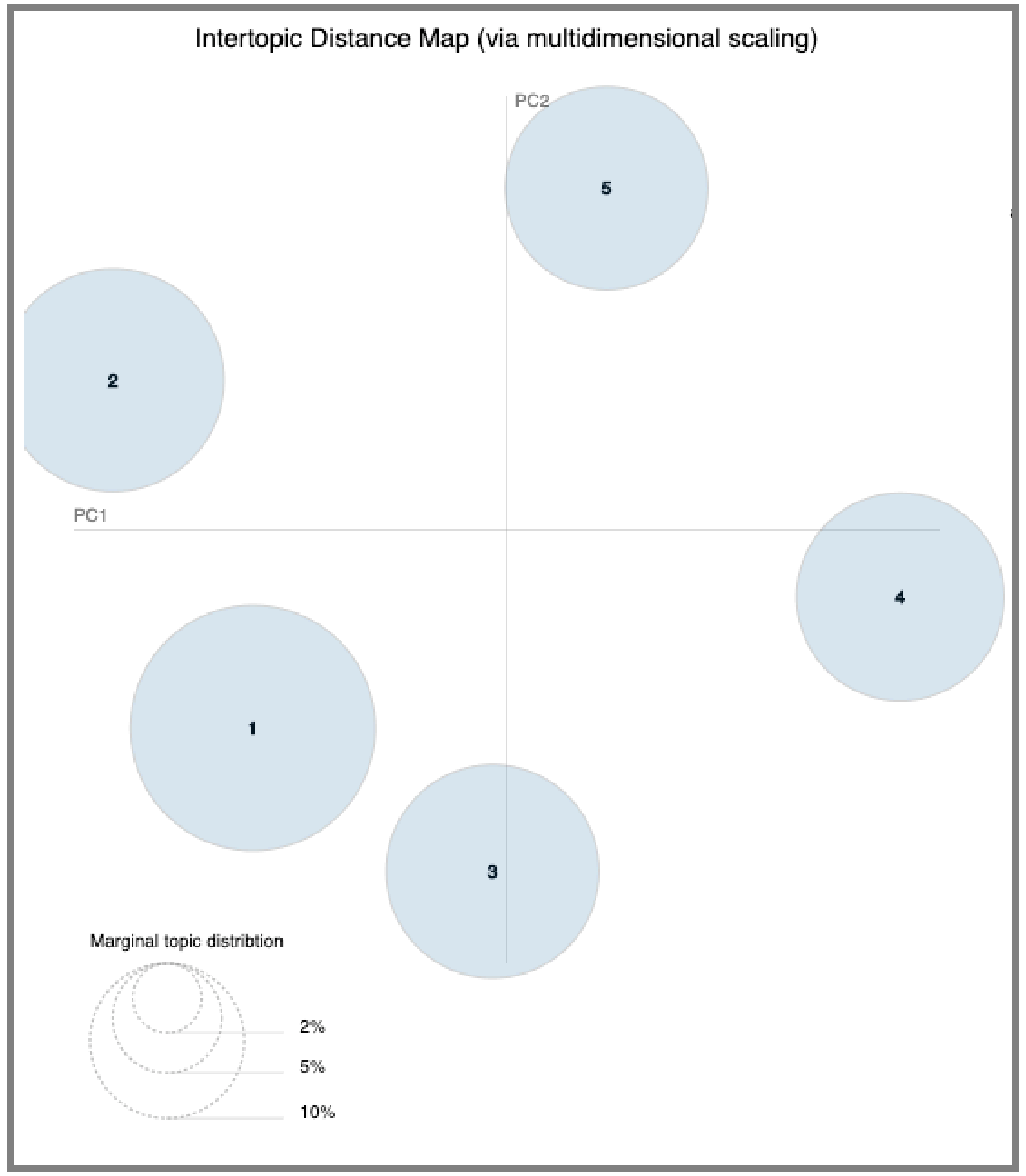
Visualize topics generated with LDA (cont'd)

Note: The results of LDA visualization might differ in the slides and the code notebook and also each time you run the LDA



LDA visualization: topic distribution

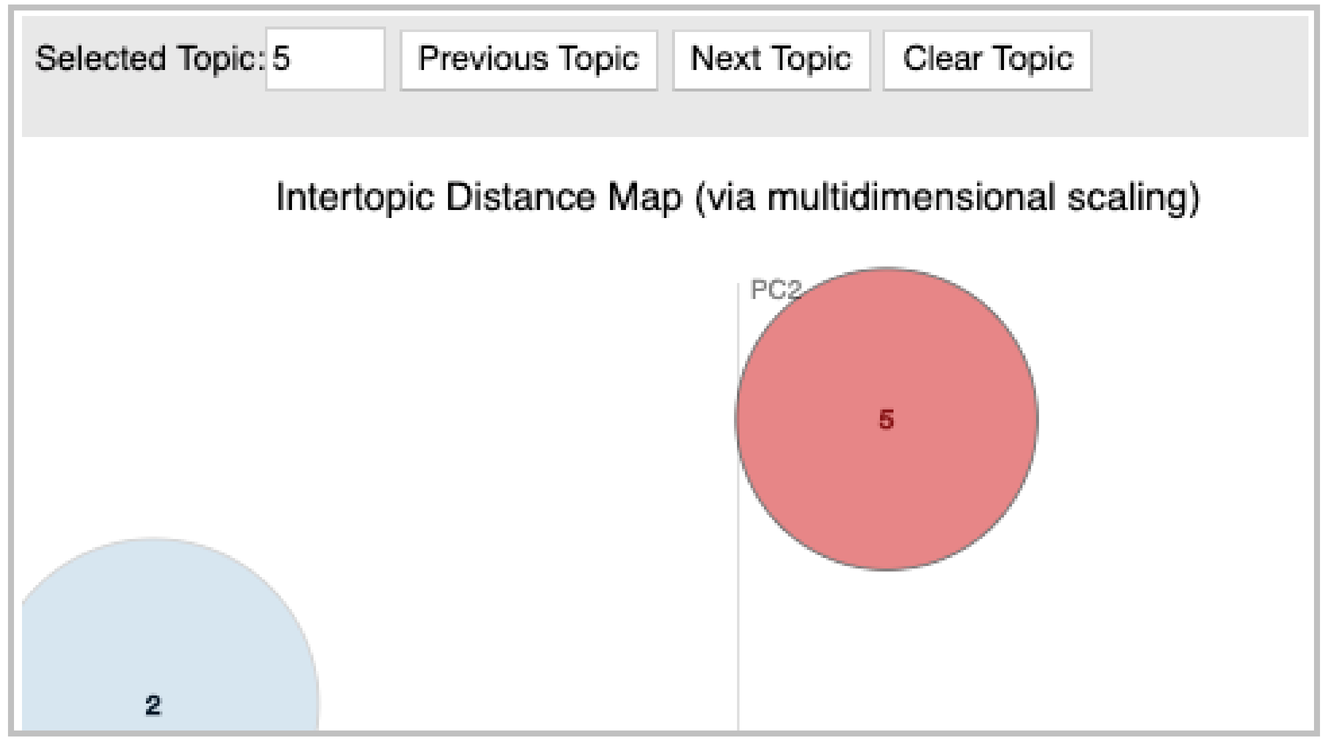
- The left-hand side shows the topics represented by the circles



- Circle **locations** are related to the topic position with respect to one another
  - topics closer to each other in space are closer to each other in meaning
  - topics farther away are more dissimilar in meaning
- Circle **size** is related to the number of documents that contain the topic
  - topics found in more documents are bigger circles
  - topics found in fewer documents are smaller circles
- *By looking at this part of the plot in the code notebook, what can you tell about the topics in our corpus?*

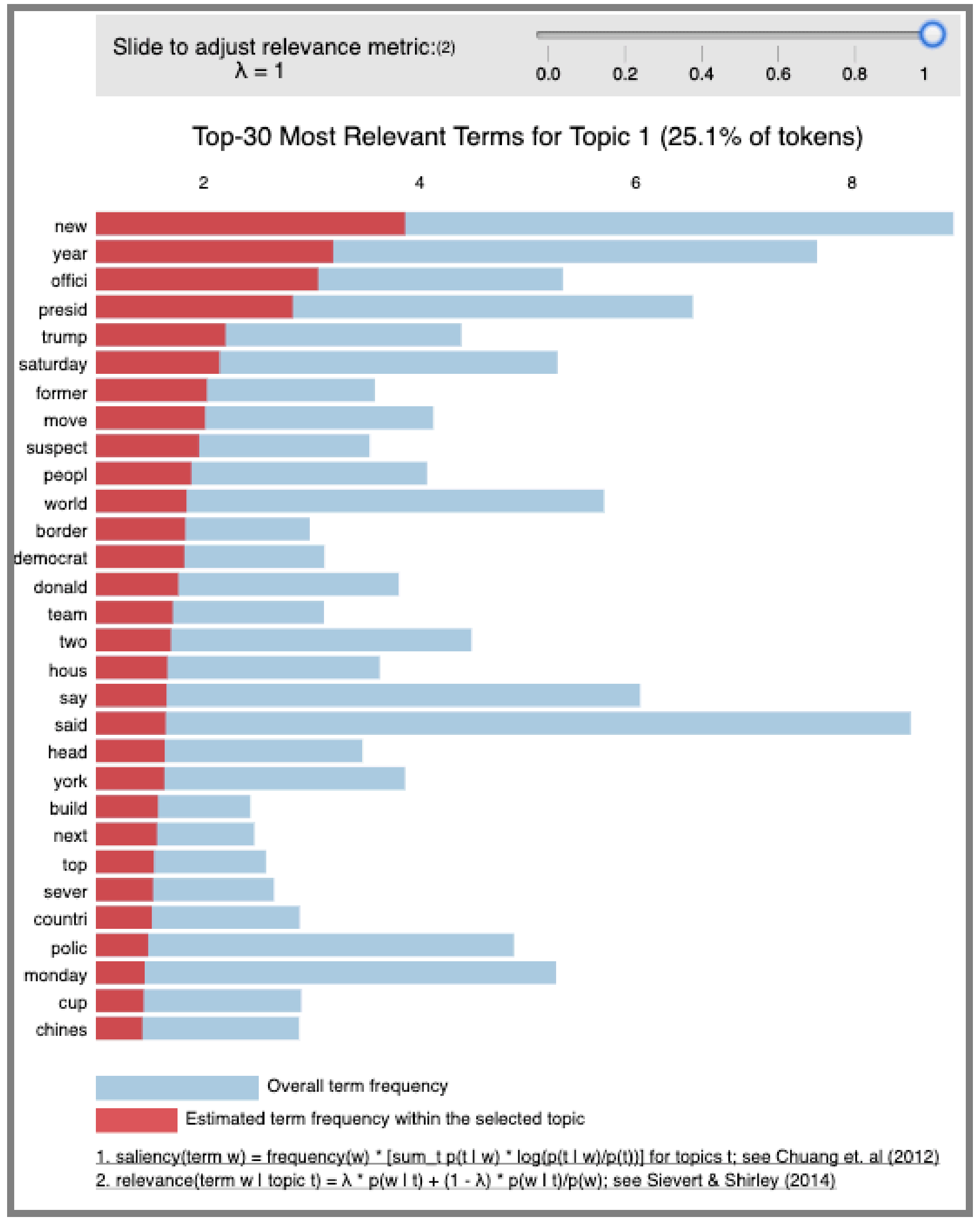
LDA visualization: select topic

- To select a particular topic, you can either
  - click on the respective circle, or
  - enter the topic number in the window in the top left corner



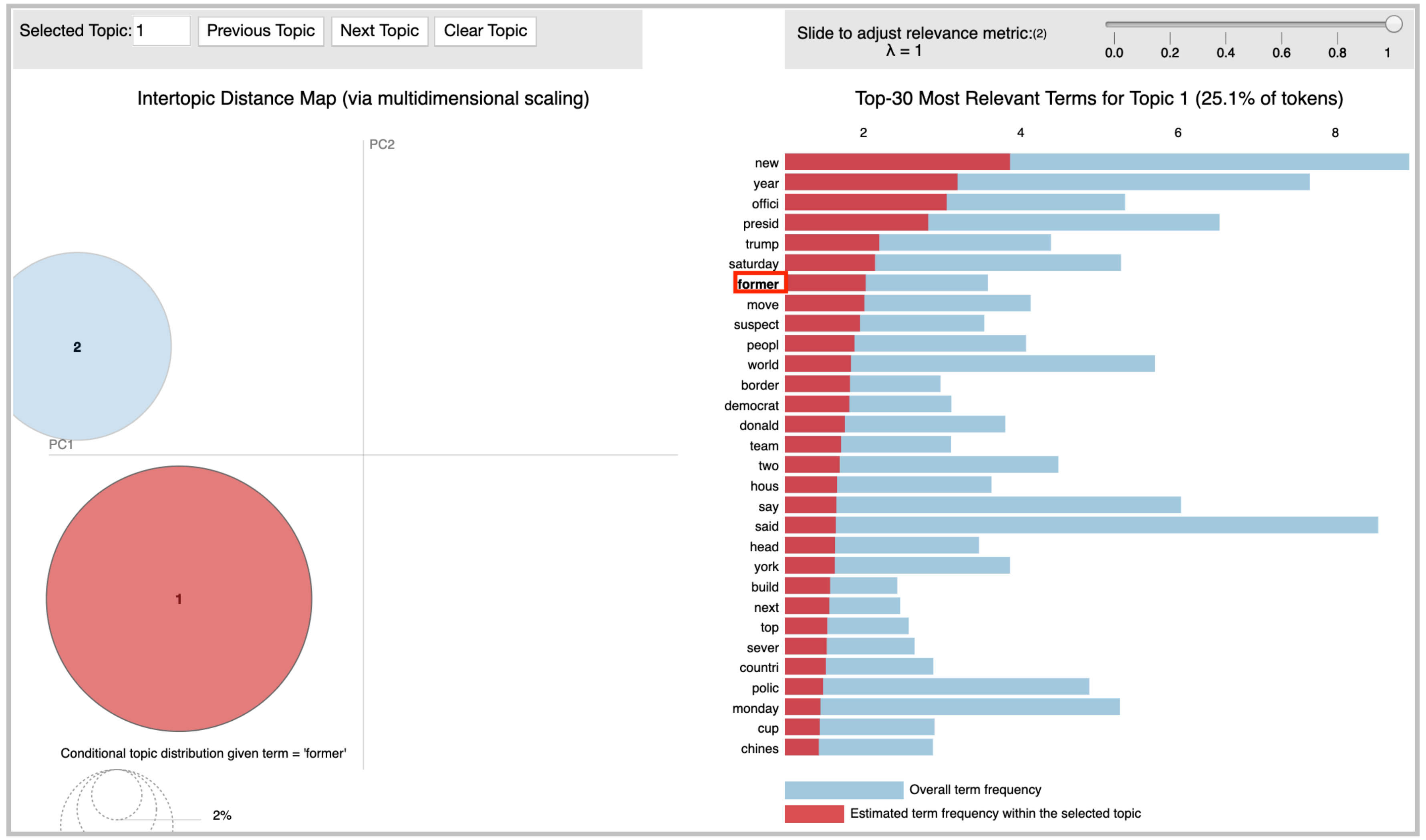
LDA visualization: relevant terms

- The right-hand side shows the most relevant terms in each topic, once the topic is selected



- The **blue bars** represent the overall term frequency in corpus
- The **maroon bars** represent term frequency in selected topic
- The slider at the top represents the value of λ - a relevance metric
  - Default is `1`, which means that the term's place in the relevance ranking below is solely based on its frequency within a selected topic
  - When set to `0`, the ranking re-arranges itself to be based on the term's frequency within topic with respect to its frequency within corpus
  - When set to be between `0` and `1`, the ranking will depend on both of the above

LDA visualization: terms across topics



- When you hover over a term, you will see the term's distribution across topics
- For instance, in the adjacent image, `former` appears in topics 1 and 2, but it is more prominent in topic 1

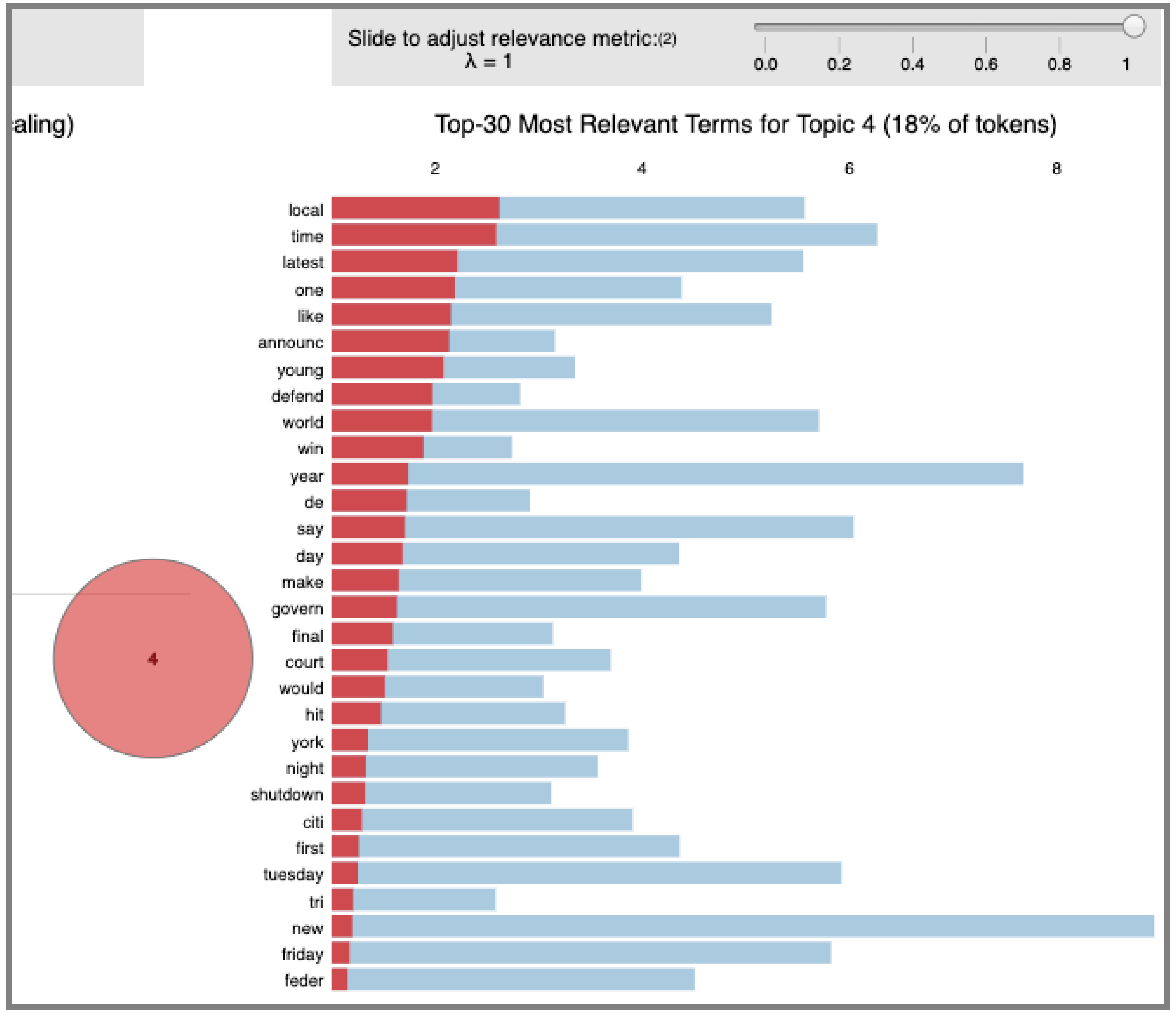
Note: The results of LDA visualization might differ in the slides and the code notebook and also each time you run the LDA

LDA visualization: how to determine topics

- The LDA algorithm does not give us explicit names of topics
- It produces the probability scores associated with *topic distribution within each document* and *term distribution across topics*
- We can **assess** those probabilities and LDA results by interacting with LDA visualization and **inferring the topics**
- LDA visualization allows us to explore and tweak the parameters to **find terms most relevant for each topic**
- These terms will allow us to **infer** the actual topic and to name it
- This process requires the subject matter expertise and some common sense, as the naming conventions of topics are subjective

LDA visualization: name topic 4

- Let's take a look at the most relevant terms in topic 4 in the code notebook:



- Take a look at the words with $\lambda = 1$, what are the top 10?
- Now adjust the slider to $\lambda = 0$, what are the top 10?
- Now adjust the slider to $\lambda = 0.2$, what are the top 10?
- By looking at all relevant terms, what do you think this topic is about?

Note: The results of LDA visualization might differ in the slides and the code notebook and also each time you run the LDA

LDA visualization: name other topics

- Now do the same for all other topics
- Try to answer the following questions:
  - Which topics were the hardest to label?
  - Why do you think that's the case?
  - What can be done to improve the model overall?

Module completion checklist

Objective	Complete
Visualize results of LDA using interactive LDAvis plot	✔
Extract and interpret document-topic information	

Extracting documents for each topic

- LDAvis plot has provided us with a lot of interesting insights, the only thing it is lacking is the documents and their **respective assignments to topics**
- We can easily **extract** that information from our data and model and **supplement** the graph with it
- We have previously loaded the original `df` dataframe and the `word_counts_array`
- We will use them to trace the document IDs in our corpus to documents in the original data and inspect document assignments

Get topic probabilities for a document

- Let's say we would like to get topic probabilities for the first document in our corpus

```
# Select the index of the document in corpus.
doc_num = 0
```

```
# Extract the vector of tf_idf weights for the document.
doc_vec = corpus_tfidf[doc_num]
print(doc_vec)
```

```
[(0, 0.31942373876087665), (1, 0.3549009519669791),
(2, 0.611871856563323 5), (3, 0.3549009519669791), (4,
0.305935928281661 8), (5, 0.22829905152454918), (6,
0.3549009519669791)]
```

```
# Extract topic probabilities for that document.
doc_topics = lda_model_tfidf.get_document_topics(doc_vec)
print(doc_topics)
```

```
[(0, 0.061030492), (1, 0.057895742), (2, 0.057665605),
(3, 0.76453817), (4, 0.058869984)]
```

```
topic, prob = sorted(doc_topics,
key=lambda x: x[1], reverse=True)[0]
```

- We can see that topic 4 has the highest probability for document 1 (remember, it's listed as 3 because the index starts at 0)

Note: The results displayed here on the slides may not match with code notebook because the topics might change each time we run the LDA

Get topic probabilities for a document (cont'd)

- Let's get the best topic and its probability for the document programmatically

```
# Initialize maximum probability score.
max_prob = 0
# Initialize best topic.
best_topic = 0

# Loop over all topics for the document.
for topic in doc_topics:

    if max_prob <= topic[1]:  #<- if current topic's probability is as high as max
        max_prob = topic[1]   #<- make current topic's probability the new max
        best_topic = topic[0] #<- make current topic best

# Create a tuple with information we just got.
doc_topic_pair = (doc_num, best_topic, max_prob)
print(doc_topic_pair)
```

```
(0, 3, 0.76453817)
```

- We can see from this tuple that for document 1 the best topic is 4 and its probability is almost `0.76`
- Let's define a function that will allow us to extract this information for a document given an LDA model

Get topic probabilities for a document (cont'd)

- The function we create here is based upon skills we've already learned, we just wrap the code into a `def` structure and add a `return` statement

```
# Put it all together into a function that returns a tuple
# with the index of the document, the best fit topic, and its probability.
def GetDocTopicPair(doc_num, corpus, lda_model_tfidf):
    # Extract the vector of tf_idf weights for the document.
    doc_vec = corpus[doc_num]
    # Extract topic probabilities for that document.
    doc_topics = lda_model_tfidf.get_document_topics(doc_vec)
    max_prob = 0
    best_topic = 0
    for topic in doc_topics:
        if max_prob <= topic[1]:
            max_prob = topic[1]
            best_topic = topic[0]
    doc_topic_pair = (doc_num, best_topic, max_prob)
    return(doc_topic_pair)
```

Get topic probabilities for all documents

- Now that we have a function that extracts information for a document, we can apply it to each document in our corpus by using a loop

```
# Create an empty list of the same length as the number of documents.
doc_topic_pairs = [None]*dictionary.num_docs

# Loop through a range of document indices.
for i in range(dictionary.num_docs):
    # For each document index, get the document-topic tuple.
    doc_topic_pairs[i] = GetDocTopicPair(i, corpus_tfidf, lda_model_tfidf)

print(doc_topic_pairs[:10])
```

```
[(0, 3, 0.76456594), (1, 3, 0.7874671), (2, 4, 0.7978336), (3, 4, 0.75982654), (4, 4,
0.60156727), (5, 0, 0.6989234), (6, 0, 0.597987), (7, 1, 0.7220774), (8, 0, 0.76217055), (9,
1, 0.7705009)]
```

- We now have a list of tuples that contain:
  - document id
  - topic id
  - topic probability for that document

Create a data frame with doc-topic assignments

- Let's put all of these tuples into a neat dataframe, which will contain the following columns:
  - `doc_id`
  - `best_topic`
  - `best_probability`

```
# Make a dataframe out of a list of tuples.
doc_topic_pairs_df = pd.DataFrame(doc_topic_pairs)

# Assign column names to the dataframe.
doc_topic_pairs_df.columns = ["doc_id", "best_topic", "best_probability"]
print(doc_topic_pairs_df.head())
```

```
   doc_id  best_topic  best_probability
0       0           3          0.764566
1       1           3          0.787467
2       2           4          0.797834
3       3           4          0.759827
4       4           4          0.601567
```

Matching document ids to original data

- When we were cleaning the text, we had to remove all documents with word counts under 5
- That offset our document indexing, let's retrieve it and assign original index from `df` dataframe to our `doc_topic_pairs_df` dataframe

```
# Find indices of documents that we kept.
valid_documents = np.where(word_counts_array >= 5)[0]
```

```
# Now assign the index of the original document to be the index of the dataframe.
doc_topic_pairs_df.index = valid_documents
print(doc_topic_pairs_df.iloc[0:10, ])
```

```
   doc_id  best_topic  best_probability
0       0           3          0.764566
1       1           3          0.787467
2       2           4          0.797834
3       3           4          0.759827
4       4           4          0.601567
5       5           0          0.698923
6       6           0          0.597987
7       7           1          0.722077
8       8           0          0.762171
9       9           1          0.770501
```

Inspect documents for a given topic

- Now that we have all pieces in place, we can retrieve all documents assigned to a topic and inspect them
- For demonstration purposes, we will do this for topic 3 and will only output the top `10` documents in that topic

```
# Filter and sort all documents assigned to topic 3 by probability in descending order.
topic3_docs = doc_topic_pairs_df.query("best_topic==2")
topic3_docs = topic3_docs.sort_values(by = "best_probability", ascending = False)
print(topic3_docs.head())
```

```
     doc_id  best_topic  best_probability
221     219           2          0.806254
145     143           2          0.804139
68       68           2          0.802452
164     162           2          0.799966
97       97           2          0.798225
```

```
# Let's see how many documents were assigned to that topic.
print(topic3_docs.shape)
```

```
(39, 3)
```

Inspect documents for a given topic (cont'd)

- Get the indices of the top `10` documents in the topic and print them

```
# Let's get the top 10 documents that were assigned to that topic.
top_10 = topic3_docs.index[0:10,]
```

```
# Inspect the top 10 documents in topic 3.
df_topic3 = df.loc[top_10, :]
print(df_topic3[['snippet']])
```

```
                                               snippet
221  Job openings are outnumbering unemployed worke...
145  India's central bank, having changed leadershi...
68   Greece's public order minister strongly critic...
164  Britain is testing how its motorway and ferry ...
97   India's Supreme Court is likely to name a pane...
125  Global stocks soared Friday and reversed the b...
203  SoftBank Group Corp will inject another $2 bil...
41   China's population is set to reach a peak of 1...
142  Thousands of demonstrators marched in Hong Kon...
80   An executive at a San Diego television station...
```

Note: the process for topic indexing in the model and in visualization is not same. Be careful, and match them up based on the relevant words instead! In this case, topic 3 corresponds to topic 2 in LDAvis

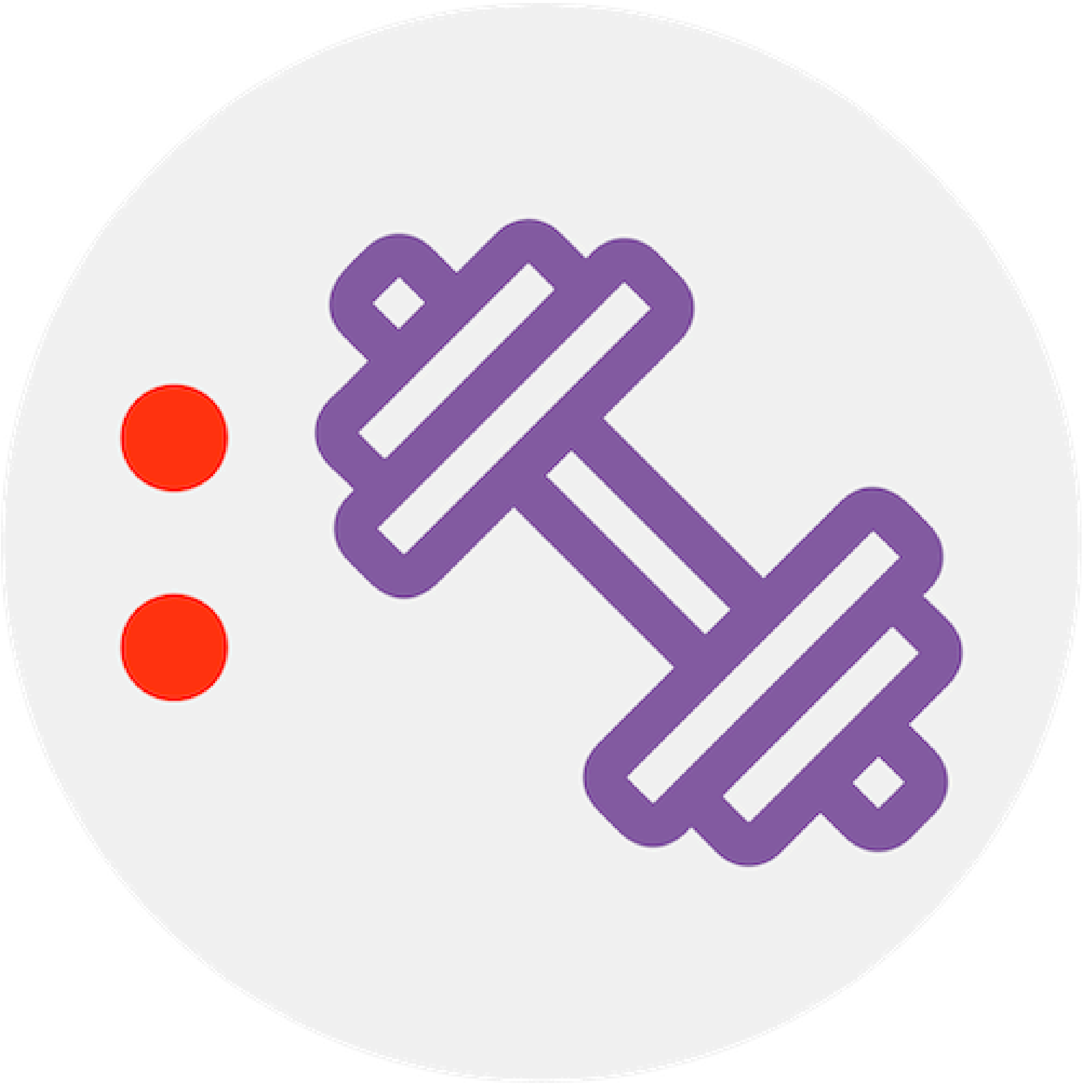
Save LDA visualization to HTML file

- To keep and distribute the visualization as fully-interactive `HTML` file, we can simply use `pyLDAvis.save_html()` method
- We need to provide it with two arguments:
  - visualization object we prepared earlier (i.e. `vis`)
  - file path with name where we would like to save it

```
# Save the plot as a self-contained HTML file.
pyLDAvis.save_html(vis, str(plot_dir) +"/df_LDAvis.html")
```

Knowledge check

Exercise



You are now ready to try Tasks 10-12 in the Exercise for this topic

Module completion checklist

Objective	Complete
Visualize results of LDA using interactive LDAvis plot	✔
Extract and interpret document-topic information	✔

Topic Modeling: Topic summary

In this part of the course, we have covered:

- Topic modeling as an unsupervised method in text analysis
- Latent Dirichlet Allocation as a popular topic modeling algorithm
- Implement LDA on a corpus of documents

Congratulations on completing this module!