Guidelines for Calibrating Nucleus APIs

SumUp Analytics, Proprietary & Confidential

This version 03/30/2019

The purpose of this document is to facilitate the calibration of Nucleus APIs' input arguments by end-users.

Each section is dedicated to an API, and within each section, a subsection is dedicated to a problem an end-user is trying to solve.

0) Dataset Creation and Upload

a. I have a set of documents (txt, docx, csv, pdf, URL, html) that I want to upload to create a dataset off of them. How do I do that?

Please take a look at our use case notebooks to learn how to leverage the Nucleus helper function

<mark>nucleus_helper.upload_files(api_instance, dataset, file_iter, processes)</mark>

For distributed document upload and injection.

For txt, docx, pdf, html: use the API Upload with input args {file, dataset}.

If the documents you work with are either txt, docx or pdf then you will need to proceed with a single document per API call. By specifying the same dataset name in the append_file_to_dataset API, those documents will get inserted into the same dataset.

```
api_instance = nucleus_api.NucleusApi(nucleus_api.ApiClient(configuration))
file = 'quarles20181109a.pdf' # file |
dataset = 'dataset_from_file' # str | Destination dataset where the file will be inserted.
api_response = api_instance.post_upload_file(file, dataset)
```

For URLs: use the API Upload with input args {file_url, dataset, filename, metadata}. The input arguments 'filename' and 'metadata' are optional.

This URL-based upload notably enables users to upload from an AWS S3 bucket.

```
api_instance = nucleus_api.NucleusApi(nucleus_api.ApiClient(configuration))
dataset = 'dataset_from_url'
file_url = 'https://www.federalreserve.gov/newsevents/speech/files/quarles20181109a.pdf'

payload = nucleus_api.UploadURLModel(dataset=dataset, file_url=file_url) # UploadURLModel |
api_response = api_instance.post_upload_url(payload)
```

For csv: If the documents you work with are csv then a csv file will be uploaded as a collection of documents, where each row of the csv file is considered to be a distinct document.

```
api_instance = nucleus_api.NucleusApi(nucleus_api.ApiClient(configuration))
csv_file = 'trump-tweets-100.csv'
dataset = 'dataset_test'
```

```
doc_cnt = 0
with open(csv_file, encoding='utf-8-sig') as csvfile:
    reader = csv.DictReader(csvfile)
    for row in reader:
        if doc_cnt < 1:
            payload = nucleus_api.Appendjsonparams(dataset=dataset,
                                language='english',
                                document={'time': row['time'], 'title': row['title'],
                                        'content':row['content'], 'author': row['author']}
                                )
        api_response = api_instance.post_append_json_to_dataset(payload)
    doc_cnt = doc_cnt + 1
```

b. **I have a database table containing lots of documents. I want to upload it to create a dataset. How do I do that?**

Please take a look at our use case notebooks to learn how to leverage the Nucleus helper function

nucleus_helper.upload_files(api_instance, dataset, file_iter, processes)

For distributed document upload and injection.

Every database has a different schema, and for security reasons, most enterprise databases are only accessible internally. Therefore, in the current release of Nucleus APIs, database tables are not yet supported through a dedicated upload API. Users who wish to connect such data source to the APIs need to write a custom script leveraging the Nucleus library function post_append_json_to_dataset.

The db table content would need to be read line by line. Each line is then inserted in the dataset as per the below example. This is similar to how one would build a dataset from a csv file, where each row is a different document.

```
api_instance = nucleus_api.NucleusApi(nucleus_api.ApiClient(configuration))
# reader a cursor to db table entries
    for row in reader:
        if doc_cnt < 1:
            payload = nucleus_api.Appendjsonparams(dataset=dataset,
                                language='english',
                                document={'time': row['time'], 'title': row['title'],
                                        'content': row['content'], 'author': row['author']}
                                )
        api_response = api_instance.post_append_json_to_dataset(payload)
    doc_cnt = doc_cnt + 1
```

1) **Dataset Deletion**
   a. **I want to delete a dataset. How do I do that?**

   Use the DeleteDataset API. Please note that there is no dataset backup API currently available, so the deletion of a dataset can only be reverted by a SumUp System Admin. You would need to re-upload all the documents from this dataset to reinstate it on your end.

   ```
   api_instance = nucleus_api.NucleusApi(nucleus_api.ApiClient(configuration))
   dataset = 'dataset_from_json'

   payload = nucleus_api.Deletedatasetmodel(dataset=dataset) # Deletedatasetmodel |
    api_response = api_instance.post_delete_dataset(payload)
   ```

   b. **I want to delete specific documents from a dataset. How do I do that?**

   Use the DeleteDocument API. Please note that there is no dataset backup API currently available, so the deletion of a document can only be reverted by a SumUp System Admin. You would need to re-upload that document in the dataset to reinstate it on your end.

   ```
   api_instance = nucleus_api.NucleusApi(nucleus_api.ApiClient(configuration))
   dataset = 'dataset_from_json'
   targetDocid = 'integer_id_of_doc_to_delete' # ID and doc content/title/metadata can be matched
   using  get_doc_info API

   payload = nucleus_api.Deletedocumentmodel(dataset=dataset, docid=targetDocid) #
   Deletedocumentmodel |
   api_response = api_instance.post_delete_document(payload)
   ```

2) **Metadata-based selection of documents**
   a. **I want to retrieve the info of documents that all share a common metadata. How do I do that?**

   Use the DocInfo API and specify a metadata selection using the input argument 'metadata_selection'.

   ```
   api_instance = nucleus_api.NucleusApi(nucleus_api.ApiClient(configuration))
   dataset = 'dataset_example' # str | Dataset name.
   selector = {"field": "value"} # Example {"author": "Albert Einstein"}

   payload = nucleus_api.DocInfo(dataset=dataset, metadata_selection=selector)
   api_response = api_instance.post_doc_info(payload)
   ```

b. I want to display documents that all share a common metadata. How do I do that?

```
api_instance = nucleus_api.NucleusApi(nucleus_api.ApiClient(configuration))
dataset = 'dataset_example' # str | Dataset name.
selector = {"field": "value"} # Example {"author": "Albert Einstein"}

payload = nucleus_api.DocDisplay(dataset= dataset, metadata_selection=selector)
api_response = api_instance.post_doc_display(payload)
```

3) Embedded Data Feeds

a. What are the datafeeds directly available through the Nucleus APIs?

As a preamble, please note that this data is made available as-is and no representation is made regarding its accuracy, completeness and availability.

1. Central Banks
What banks: Federal Reserve, Bank of Canada, Banco de Mexico, Bank of Brazil, ECB, Bundesbank, Bank of England, Bank of France, Bank of Italy, Bank of Spain, Central Bank of Russia, People Bank of China, Bank of Japan, Royal Bank of Australia and all the US Regional Federal Reserves

These banks are encoded with the following names in Nucleus APIs:
```
central_banks_list =
['federal_reserve','bank_of_england','ecb','bank_of_canada','bank_of_australia','bank_of_spain','banc
o_de_mexico','bundesbank','bank_of_japan','people_bank_of_china','bank_of_france','bank_of_italy','
bank_of_brazil','russian_fed']
```

```
regional_feds_list =
['atlanta_fed', 'boston_fed', 'cleveland_fed', 'chicago_fed', 'dallas_fed', 'kansas_city_fed',
'minneapolis_fed', 'new_york_fed', 'philadelphia_fed', 'richmond_fed', 'san_francisco_fed',
'st_louis_fed']
```

You can select any of these banks through a metadata selection, with the key 'bank', within Central Bank datasets that are each language specific. We detail more on this below.

What documents: all speeches, press releases, publications and formal research made available on the central banks' website in both English and native language when different from English.

These document categories are encoded with the following names in Nucleus APIs:
```
document_category = ['speech', 'press release', 'publication', 'formal research']
```

You can select any of these categories through a metadata selection, with the key 'document_category**.**

What history: back to 2008 for all banks, and up to another 10 years back for a few of these Central Banks.

Update frequency: daily

2. **News RSS**
   What Media: 200 RSS feeds grouped into 8 categories: AI, Crypto, Culture, Economics, Finance, General News. The complete list is available at this link

   These categories are encoded with the following names in Nucleus APIs:
   ```
   rss_dataset =
   ["rss_feed_ai","rss_feed_bitcoin","rss_feed_culture","rss_feed_finance","rss
   _feed_news","rss_feed_economics"]
   ```

   They directly exist as standalone datasets. We detail more on this below.

   What documents: all articles shared through RSS, in English.

   What history: back to December 2018

   Update frequency: daily

3. **SEC filings (coming soon)**
   What companies: All public companies listed on a US Exchange

   What documents: all 10K, 10K/A, 10Q, 10Q/A, 8K, S1, S1/A, 13F.

   What history: back to 2000

   Update frequency: daily

b. **How can I use these datasets through the Nucleus APIs?**

1. **Central Banks**
   The datasets are organized per language and can be accessed as follows when building your API payload:

   With LANGUAGE either of {english, chinese, japanese, german, spanish, french, italian, portuguese, russian}

```
dataset = "sumup/central_banks_LANGUAGE"
```

You can then use the metadata_selection argument in your payload to retain specific Central Banks and document types:

```
metadata_selection = {'bank': 'federal_reserve', 'document_category': 'speech'}
```

2. **News RSS**
   The datasets are organized per field and can be accessed as follows when building your API payload:

   With FIELD either of {ai, bitcoin, finance, economics, culture, news}

   ```
   dataset = "sumup/rss_feed_FIELD"
   ```

3. **SEC filings (coming soon)**
   We will update this section once the data becomes available through the APIs

c. **Can I do anything with these embedded datafeeds?**
   No, these datasets are read-only. You cannot remove or add documents to any of these datasets. If you find certain documents are missing or are erroneous, please contact sales@sumup.ai

4) **Multi-Lingual Capabilities**
   a. **What languages are supported by the Nucleus APIs?**
      Currently, 12 languages are fully supported by the Nucleus APIs:
      - English
      - Chinese, both Simplified and Traditional
      - Spanish
      - Japanese
      - Portuguese
      - German
      - Italian
      - French
      - Russian
      - Arabic
      - Farsi

Please note that the sentiment and consensus analyses have been tailored to the fields of finance, economics and geopolitics. If you operate in different fields, nothing will break but the results won't have the same accuracy and relevance. In such situation we strong advocate using your own sentiment dictionary.

If your corpus is in another language than any of the 12 above:
- You will still be able to work for any Latin-alphabet-based language.
- Results won't be as accurate for topics, summarization, content recommendation, but you can do some quick clean-up on the fly since the APIs are so fast and provide transparent results
- Sentiment and consensus will be useless unless you provide your own sentiment dictionary
- For languages that are in a different alphabet than the Latin, Arabic, Russian, Chinese ones, running the APIs will result in errors

b. **How do I query and customize stopwords for languages other than English?**
Any query and any list of custom stopwords that you pass to an API must be in the same language as that of the corpus

Down the line we will support on-the-fly translation so you can query in any language you are comfortable with and the query is translated into the language of the corpus in the background.

5) **Topic Modeling**
a. **The topics I get are too high-level, or they are well-known for someone with domain expertise in my field**

Increase the number of topics 'num_topics' identified and extracted by the algorithm and look at the higher-order topics.

```
api_instance = nucleus_api.NucleusApi(nucleus_api.ApiClient(configuration))
dataset = 'dataset_example' # str | Dataset name.
num_topics = 20 # int | Number of topics to be extracted from dataset. (optional) (default to 8)

payload = nucleus_api.Topics(dataset=dataset, num_topics=num_topics)
api_response = api_instance.post_topic_api(payload)
```

Are the higher-order topics more specific and relevant? Great.

Are all these topics still too high-level? Add their respective keywords to the input argument 'custom_stop_words' and rerun the Topic Model API.

```
api_instance = nucleus_api.NucleusApi(nucleus_api.ApiClient(configuration))
dataset = 'dataset_example' # str | Dataset name.
custom_stop_words = ["word1", "word2" ]
```

```
num_topics = 20 # int | Number of topics to be extracted from dataset. (optional) (default to 8)

payload = nucleus_api.Topics(dataset=dataset, num_topics=num_topics,
                             custom_stop_words=custom_stop_words)
api_response = api_instance.post_topic_api(payload)
```

Do you still find some topics are relevant to you, but they are a bit too broad? In this case, go to the subtopics level.
- Run a first-pass topic model
- Select the topics you identify as relevant
- Extract the keywords from those topics, and run a second-pass topic model that contains these keywords as user-query in the input argument 'query'

```
api_instance = nucleus_api.NucleusApi(nucleus_api.ApiClient(configuration))
dataset = 'dataset_example' # str | Dataset name.
num_topics = 20 # int | Number of topics to be extracted from dataset. (optional) (default to 8)

payload = nucleus_api.Topics(dataset, num_topics=num_topics)
api_response = api_instance.post_topic_api(payload)
pprint(api_response) # Retrieve keywords from topics of interest

num_topics = 6 # int | Number of subtopics to be extracted from dataset. (optional) (default to 8)
query = ' ("word1" OR "word2" OR … OR "wordN") ' | Fulltext query

payload = nucleus_api.Topics(dataset=dataset, query=query, num_topics=num_topics)
api_response = api_instance.post_topic_api(payload)
```

b. The topics I get are too narrowly defined

Increase the number of keywords 'num_keywords' per topic that is identified and extracted. Rerun the Topic Model API.

```
api_instance = nucleus_api.NucleusApi(nucleus_api.ApiClient(configuration))
dataset = 'dataset_example' # str | Dataset name.
num_keywords = 20 # int | Number of keywords per topic that is extracted from the dataset. (optional) (default to 8)

payload = nucleus_api.Topics(dataset=dataset, num_ keywords =num_ keywords)
api_response = api_instance.post_topic_api(payload)
```

c. Several of the topics I get contain some generic words for someone with domain expertise in my field

Add these words as stopwords in the input argument 'custom_stop_words' and rerun the Topic Model API.

```
api_instance = nucleus_api.NucleusApi(nucleus_api.ApiClient(configuration))
dataset = 'dataset_example' # str | Dataset name.
custom_stop_words = ["word1", "word2"]

payload = nucleus_api.Topics(dataset=dataset, custom_stop_words=custom_stop_words)
```

```
api_response = api_instance.post_topic_api(payload)
```

d. I want to exclude certain documents in my dataset from the topic modeling analysis

If you know the document ID of these documents, you can pass them in the input argument 'excluded_docs' of the Topic Model API and rerun.

If you know the titles of these documents, first use the Doc Info API to retrieve the corresponding document IDs, then pass them in the input argument 'excluded_docs' of the Topic Model API and rerun.

```
api_instance = nucleus_api.NucleusApi(nucleus_api.ApiClient(configuration))
dataset = 'dataset_example' # str | Dataset name.
excluded_docs = [ "docid1", "docid2"] | ID of documents to be excluded

payload = nucleus_api.Topics(dataset=dataset, excluded_docs =excluded_docs)
api_response = api_instance.post_topic_api(payload)
```

6) Topic Summarization

a. The summary I get is too dry, I would like a bit more contextual understanding

Increase the value of the input argument 'context_amount' to the Topic Summary API and rerun. Every increment gives you an extra contextual sentence.

```
api_instance = nucleus_api.NucleusApi(nucleus_api.ApiClient(configuration))
dataset = 'dataset_example' # str | Dataset name.
context_amount = 1 # int | The number of sentences surrounding key summary sentences in the
documents that they come from. (optional) (default to 0)

payload = nucleus_api.TopicSummaryModel(dataset=dataset, context_amount=context_amount)
api_response = api_instance.post_topic_summary_api(payload)
```

b. The summary I get covers too much of an overview content of the document. I want it to be about more niche elements, or more contrasted

Identify words characteristic of the overview content:
- Either by reading the sentences currently produced in your summary and deciding
- Or by grouping those sentences from several documents that you are working on into a new dataset and running this dataset through the Topic Model API. In this case, the topics that you identify and extract can be used as the characteristic words of the overview content

Add these words as stopwords through the input argument 'custom_stop_words' to the Topic Summary API and rerun the API.

```
api_instance = nucleus_api.NucleusApi(nucleus_api.ApiClient(configuration))
dataset = 'dataset_example' # str | Dataset name.
payload = nucleus_api.Summary(dataset=dataset)
api_response = api_instance.post_topic_summary_api(payload)
pprint(api_response)
num_topics = 20 # int | Number of topics to be extracted from dataset. (optional) (default to 8)

payload = nucleus_api.Topics(dataset=dataset, num_topics=num_topics)
api_response = api_instance. post_topic _api (payload)
pprint(api_response) # Retrieve keywords from topics of interest

# If you want to determine keywords using a topic modeling approach, pull out every sentence from
#the above pprint response using the API DocDisplay and write a short script with the function
#post_append_json_to_dataset

custom_stop_words = ["word1", "word2"]

payload = nucleus_api.TopicSummaryModel(dataset=dataset,
custom_stop_words=custom_stop_words)
api_response = api_instance. post_topic_summary_api (payload)
```

c. I want to exclude certain documents in my dataset from the topic summarization

If you know the document ID of these documents, you can pass them in the input argument 'excluded_docs' of the Topic Summary API and rerun.

If you know the titles of these documents, first use the Doc Info API to retrieve the corresponding document IDs, then pass them in the input argument 'excluded_docs' of the Topic Summary API and rerun.

```
api_instance = nucleus_api.NucleusApi(nucleus_api.ApiClient(configuration))
dataset = 'dataset_example' # str | Dataset name.
excluded_docs = ["docid1", "docid2"] #| ID of documents to be excluded

payload = nucleus_api.TopicSummaryModel(dataset=dataset, excluded_docs = excluded_docs)
api_response = api_instance.post_topic_summary_api(payload)
```

7) Topic Sentiment Analysis
a. I am surprised by the sentiment on one of my topics, it doesn't look right

The Topic Sentiment API provides a few supplemental outputs in the form of document-level sentiment (on each topic) and weights, for the documents in your corpus.

Take a look at those outputs, to detect whether some specific documents have surprisingly large sentiment (whether positive or negative) and/or weight.

If there are such documents: does their sentiment or weight make sense upon reading the content? If no, you can exclude them from the analysis and rerun your API call.

```
api_instance = nucleus_api.NucleusApi(nucleus_api.ApiClient(configuration))
dataset = 'dataset_example' # str | Dataset name.

payload = nucleus_api.TopicSentimentModel(dataset=dataset)
api_response = api_instance.post_topic_sentiment_api(payload)
for i, res in enumerate(api_response.result):
    print('Topic', i, 'sentiment:')
    print('   Keywords:', res.topic)
    print('   Sentiment:', res.sentiment)
    print('   Strength:', res.strength)

    doc_id_str = ' '.join(str(x) for x in res.doc_id)
    doc_sentiment_str = ' '.join(str(x) for x in res.doc_sentiment)
    doc_score_str = ' '.join(str(x) for x in res.doc_score)
    print('   Document IDs:', doc_id_str)
    print('   Document Sentiments:', doc_sentiment_str)
    print('   Document Scores:', doc_score_str)

    print('---------------')
```

b. I want to exclude certain documents in my dataset from the topic sentiment analysis

If you know the document ID of these documents, you can pass them in the input argument 'excluded_docs' of the Topic Sentiment API and rerun.

If you know the titles of these documents, first use the Doc Info API to retrieve the corresponding document IDs, then pass them in the input argument 'excluded_docs' of the Topic Sentiment API and rerun.

```
api_instance = nucleus_api.NucleusApi(nucleus_api.ApiClient(configuration))
dataset = 'dataset_example' # str | Dataset name.
excluded_docs = ["docid1", "docid2"] #| ID of documents to be excluded

payload = nucleus_api.TopicSentimentModel(dataset=dataset, excluded_docs = excluded_docs)
api_response = api_instance.post_topic_sentiment_api(payload)
```

c. I want to use my own dictionary for sentiment

Upload your sentiment dictionary to the root folder where the rest of your code is located, as a JSON file. The structure of that JSON should be as follows:

```
Custom_dict_file = {"word1": sentiment_value_1, "word2": sentiment_value_2, …, "wordN": sentiment_value_N}
```

You can then pass this custom dictionary to the Topic Sentiment Analysis API.

```
api_instance = nucleus_api.NucleusApi(nucleus_api.ApiClient(configuration))
dataset = 'dataset_example' # str | Dataset name.
custom_dict_file = 'custom-sentiment-dict.json' # file | Custom sentiment dictionary JSON file.
(optional)

payload = nucleus_api.TopicSentimentModel(dataset= dataset, custom_dict_file =
custom_dict_file)
api_response = api_instance.post_topic_sentiment_api(payload)
```

8) Topic Consensus Analysis

a. I am surprised by the consensus on one of my topics, it doesn't look right

This might be the result of some documents in the corpus having an unexpected
sentiment. The Topic Sentiment API provides a few supplemental outputs in the
form of document-level sentiment and weights, for the documents in your corpus.

Take a look at those outputs, to detect whether some specific documents have
surprisingly large sentiment (whether positive or negative) and/or weight.

If there are such documents: does their sentiment or weight make sense based on
reading the content? If no, you can exclude them from the analysis and rerun your
API call.

```
api_instance = nucleus_api.NucleusApi(nucleus_api.ApiClient(configuration))
dataset = 'dataset_example' # str | Dataset name.

payload = nucleus_api.TopicSentimentModel(dataset=dataset)
api_response = api_instance.post_topic_sentiment_api(payload)
for i, res in enumerate(api_response.result):
    print('Topic', i, 'sentiment:')
    print('    Keywords:', res.topic)
    print('    Sentiment:', res.sentiment)
    print('    Strength:', res.strength)

    doc_id_str = ' '.join(str(x) for x in res.doc_id)
    doc_sentiment_str = ' '.join(str(x) for x in res.doc_sentiment)
    doc_score_str = ' '.join(str(x) for x in res.doc_score)
    print('    Document IDs:', doc_id_str)
    print('    Document Sentiments:', doc_sentiment_str)
    print('    Document Scores:', doc_score_str)

    print('---------------')
```

b. I want to exclude certain documents in my dataset from the topic consensus analysis

If you know the document ID of these documents, you can pass them in the input
argument 'excluded_docs' of the Topic Consensus API and rerun.

If you know the titles of these documents, first use the Doc Info API to retrieve the corresponding document IDs, then pass them in the input argument 'excluded_docs' of the Topic Consensus API and rerun.

```
api_instance = nucleus_api.NucleusApi(nucleus_api.ApiClient(configuration))
dataset = 'dataset_example' # str | Dataset name.
excluded_docs = ["docid1", "docid2"] # | ID of documents to be excluded

payload = nucleus_api.TopicConsensusModel (dataset=dataset, excluded_docs = excluded_docs)
api_response = api_instance.post_topic_consensus_api(payload)
```

c. I want to use my own dictionary for sentiment

Upload your sentiment dictionary to the root folder where the rest of your code is located, as a JSON file. The structure of that JSON should be as follows:

```
Custom_dict_file = {"word1": sentiment_value_1, "word2": sentiment_value_2, …, "wordN": sentiment_value_N}
```

You can then pass this custom dictionary to the Topic Consensus Analysis API.

```
api_instance = nucleus_api.NucleusApi(nucleus_api.ApiClient(configuration))
dataset = 'dataset_example' # str | Dataset name.
custom_dict_file = 'custom-sentiment-dict.json' # file | Custom sentiment dictionary JSON file.
(optional)

payload = nucleus_api.TopicConsensusModel(dataset=dataset,
custom_dict_file=custom_dict_file)
api_response = api_instance.post_topic_consensus_api(payload)
```

9) Document Summarization
   a. The summary I get is too dry, I would like a bit more contextual understanding

Increase the value of the input argument 'context_amount' to the Doc Summary API and rerun. Every increment gives you an extra contextual sentence.

```
api_instance = nucleus_api.NucleusApi(nucleus_api.ApiClient(configuration))
dataset = 'dataset_example' # str | Dataset name.
doc_title = 'doc_title_example' # str | The title of the document to be summarized.
context_amount = 1 # int | The number of sentences surrounding key summary sentences in the
documents that they come from. (optional) (default to 0)

payload = nucleus_api.DocumentSummaryModel (dataset=dataset, doc_title=doc_title,
                                            context_amount=context_amount)
api_response = api_instance.post_doc_summary_api(payload)
```

b. The summary I get contains generic content from the document

Identify words characteristic of this generic content:
- Either by reading the generic sentences and deciding
- Or by grouping generic sentences from several documents that you are working on into a new dataset and running this dataset through the Topic Model API. In this case, the topics that you identify and extract can be used as the characteristic words of your generic content

Add these words as stopwords through the parameter 'custom_stop_words' in input to the Document Summary API and rerun the API.

```
api_instance = nucleus_api.NucleusApi(nucleus_api.ApiClient(configuration))
dataset = 'dataset_example' # str | Dataset name.
doc_title = 'doc_title_example' # str | The title of the document to be summarized.

payload = nucleus_api.DocumentSummaryModel (dataset=dataset, doc_title=doc_title)
api_response = api_instance.post_doc_summary_api(payload)
pprint(api_response)

# If you want to determine keywords using a topic modeling approach, pull out every sentence from
#the above pprint response using the API DocDisplay and write a short script with the function
#post_append_json_to_dataset

custom_stop_words = ["word1", "word2"]

payload = nucleus_api.DocumentSummaryModel (dataset=dataset, doc_title=doc_title,
                                    custom_stop_words=custom_stop_words)
api_response = api_instance. post_doc_summary_api (payload)
```

c. My document summary is not specific enough for someone with domain expertise in my field. I want a summary that focuses more on the niche aspects and the contrasting aspects of this document

Identify words characteristic of the overview content:
- Either by reading the sentences currently produced in your summary and deciding
- Or by grouping those sentences into a new dataset and running this dataset through the Topic Model API. In this case, the topics that you identify and extract can be used as the characteristic words of the overview content

Add these words as stopwords through the input argument 'custom_stop_words' to the Document Summary API and rerun the API.

```
api_instance = nucleus_api.NucleusApi(nucleus_api.ApiClient(configuration))
dataset = 'dataset_example' # str | Dataset name.
doc_title = 'doc_title_example' # str | The title of the document to be summarized.

payload = nucleus_api.DocumentSummaryModel (dataset=dataset, doc_title=doc_title)
```

```
api_response = api_instance.post_doc_summary_api(payload)
pprint(api_response)

# If you want to determine keywords using a topic modeling approach, pull out every sentence from
#the above pprint response using the API DocDisplay and write a short script with the function
#post_append_json_to_dataset

custom_stop_words = ["word1", "word2"]

payload = nucleus_api.DocumentSummaryModel (dataset=dataset, doc_title=doc_title,
                                        custom_stop_words=custom_stop_words)
api_response = api_instance. post_doc_summary_api (payload)
```

10) Document Sentiment Analysis

a. I am surprised by the sentiment on one of my documents, it doesn't look right

Leverage the Topic Sentiment API running on a single document to assess whether certain of the key topics extracted from that document and used in evaluating the sentiment of the document are either irrelevant or have a sentiment contribution which is surprising.

```
api_instance = nucleus_api.NucleusApi(nucleus_api.ApiClient(configuration))
dataset = 'dataset_example' # str | Dataset name.
doc_title = 'doc_title_example' # str | The title of the document to be summarized.
metadata_selection = {'title': doc_title}

payload = nucleus_api.TopicSentimentModel(dataset=dataset,
                                        metadata_selection=metadata_selection)
api_response = api_instance.post_topic_sentiment_api(payload)
for i, res in enumerate(api_response.result):
    print('Topic', i, 'sentiment:')
    print('   Keywords:', res.topic)
    print('   Sentiment:', res.sentiment)

    print('---------------')
```

11) Content Recommendations

a. I want to get more recommended documents on my list of topics
Increase the value of the input argument 'doc_num' and rerun the Doc Recommendation API.

```
api_instance = nucleus_api.NucleusApi(nucleus_api.ApiClient(configuration))
dataset = 'dataset_example' # str | Dataset name.
num_docs = 30 # int | Number of desired recommended docs per topic

payload = nucleus_api.DocumentRecommendModel(dataset=dataset, num_docs=num_docs)
api_response = api_instance.post_doc_recommend_api(payload)
```

b. I want to exclude certain documents in my dataset from the content recommendation

If you know the document ID of these documents, you can pass them in the input argument 'excluded_docs' of the Doc Recommendation API and rerun.

If you know the titles of these documents, first use the Doc Info API to retrieve the corresponding document IDs, then pass them in the input argument 'excluded_docs' of the Doc Recommendation API and rerun.

```
api_instance = nucleus_api.NucleusApi(nucleus_api.ApiClient(configuration))
dataset = 'dataset_example' # str | Dataset name.
excluded_docs = ["docid1", "docid2"] # | ID of documents to be excluded

payload = nucleus_api.DocumentRecommendModel(dataset=dataset,
excluded_docs=excluded_docs)
api_response = api_instance.post_doc_recommend_api(payload)
```

12) Topic Historical Analysis

a. The charts of topic strength, sentiment and consensus I get are too granular. How can I get them with less frequent updates?

Adjust the input argument 'update_period' from being a daily frequency ('d') to a lower frequency such as weekly ('w') or monthly ('m').

If you want an adjustment that is a bit less radical, you can use the input argument 'inc_step', which reflects how many units of the update_period are taken in between two historical analyses. E.g.: inc_step = 1 with update_period = 'd' means that the historical analysis will run every day in the time period of your dataset. Increase the value of the input argument 'inc_step'.

```
api_instance = nucleus_api.NucleusApi(nucleus_api.ApiClient(configuration))
dataset = dataset   # str | Dataset name.
time_period = '6M'  # str | Time period selection (default to 1M)
update_period = 'd' # str | Frequency at which the historical anlaysis is performed (default to d)
inc_step = 1 # int | Number of increments of the udpate period in between two historical
computations. (optional) (default to 1)

payload = nucleus_api.TopicHistoryModel(dataset=dataset,
                                        time_period=time_period,
                                        update_period=update_period,
                                        inc_step=inc_step)
api_response = api_instance.post_topic_historical_analysis_api(payload)
```

b. I want to adjust the topic modeling itself before conducting a historical analysis on topics. What is the best way?

Refer to section 2) of this document: Topic Modeling. That API will run faster so you can refine the topics the way you need. Once you found a configuration of the topic model that suits your needs, use it for the Topic Historical Analysis API.

c. I want to use my own dictionary for sentiment

Upload your sentiment dictionary to the root folder where the rest of your code is located, as a JSON file. The structure of that JSON should be as follows:

Custom_dict_file = {"word1": sentiment_value_1, "word2": sentiment_value_2, …, "wordN": sentiment_value_N}

You can then pass this custom dictionary to the Topic Historical Analysis API.

```
api_instance = nucleus_api.NucleusApi(nucleus_api.ApiClient(configuration))
dataset = 'dataset_example' # str | Dataset name.
custom_dict_file = 'custom-sentiment-dict.json' # file | Custom sentiment dictionary JSON file.
(optional)

payload = nucleus_api.TopicHistoryModel(dataset=dataset, custom_dict_file=custom_dict_file)
api_response = api_instance. post_topic_historical_analysis_api (payload)
```

13) Author Connectivity

a. I know the name of an author I am interested in analyzing. How do I use the Author Connectivity API?

```
api_instance = nucleus_api.NucleusApi(nucleus_api.ApiClient(configuration))
dataset = dataset # str | Dataset name.
target_author = 'D_Trump16' # str | Name of the author to be analyzed.

payload = nucleus_api.AuthorConnection(dataset= dataset, target_author=target_author)
api_response = api_instance.post_author_connectivity_api(payload)
```

14) Topic Delta

a. What do I use this API for?

This API allows the user to assess how the exposure of a group of entities to a set of topics has changed between two time periods.

Such measure of the change in exposure can be used as-is for entity monitoring on specific topics or can be used as an input to a supervised predictive model. For instance, one would use this API on SEC filings of a set of companies to determine how each company is exposed to a set of critical topics in those filings on two different filing dates. This change in exposure per company can be used in training a model to predict changes in stock prices between filing dates.

b. How should I build my dataset to use with this API?

The dataset should be comprised of documents that are each attached to an entity (such as a company, a person, a country) that the user wants to analyze. For each entity, there should be multiple documents that are spread through time (such as corporate SEC filings that are published every quarter, emails/chat logs from specific people).

The API enables the user to compute a change in exposure between two time periods of the whole dataset, using a time stamp metadata selection as details below:

```
time_start_t0 = '2018-08-12 00:00:00'
time_end_t0 = '2018-08-15 13:00:00'
time_start_t1 = '2018-08-16 00:00:00'
time_end_t1 = '2018-08-19 00:00:00'

payload = nucleus_api.TopicDeltaModel(dataset=dataset,
                                time_start_t0 = time_start_t0,
                                time_end_t0 = time_end_t0,
                                time_start_t1 = time_start_t1,
                                time_end_t1 = time_end_t1)
api_response = api_instance.post_topic_delta_api(payload)
```

To build a longer historical tracking, there are two ways:
- If you want to see how the latest critical topics have evolved over time, then use the Topic Historical Analysis API
- If you want to determine changes in exposure of some entities to critical topics over time, then use this Topic Delta API and embed it in a For Loop, looping over time stamps

c. I want to adjust the topic modeling itself before analyzing changes in exposures to topics on my dataset. What is the best way?

Refer to section 2) of this document: Topic Modeling. That API will run faster so you can refine the topics the way you need. Once you found a configuration of the topic model that suits your needs, use it for the Topic Delta API.