Algorithm 2: Using BERTopic and GPT2 for Topic Modeling

Topic Modeling With Language Models

View this code on Github

Topic Modeling is the practice of pulling out categorized groups of information from a piece of longer text.

Example: Inferring chapters from a book or segments of a movie. This is a classic data science topic that has been studied for years. Check out examples from previous research like scikit-learn and BERTopic if you want to see these techniques.

In this Algorithm we are going to study the use of Large Language Models in order to make Topic Modeling

In the following steps we are lgoing to put special attention to comprehensiveness and robustness of information and API costs so please be mindful of your expense comfortability.

In this part of the first approach of the algorithm:

- **1st Pass:** Run through the entire document via map reduce and pull out topics as bullet points
- **2nd Pass:** Iterate through your topic bullet points and expand on them with a subset of context that was selected via retrieval

Bonus: As a bonus we are also going to be looking at how to auto generate timestamps for each topic as well. The most common use case of this is YouTube Chapters

The Assumptions for the Transcripts

- You don't have a table of contents. That would definitely help out (since a human likely generated them) The purpose of this method as general as possible so you can apply it
- You want to *learn* the nuts and bolts how to do this. If you wanted a 3rd party tool to do this for you I suggest something like AssemblyAI or PodcastNotes

The application for this can be for the following ones:

- **YouTube Videos** - Auto Chapter Generation
- **Podcasts** - Extract structured information
- **Meeting Notes** - Send topic summaries to participants
- **Town Hall Meetings** - Structured information
- **Earnings Report Calls** - Sell structured data to investment groups
- **Legal Documents** - Quickly summarize by topic
- **Movie Scripts** - Quick bullet points for production recaps
- **Books** - Auto generate table of contents

```
!pip install langchain
```

```
Collecting langchain
  Downloading langchain-0.2.4-py3-none-any.whl (974 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 974.2/974.2 kB 5.3 MB/s eta 0:00:00
Requirement already satisfied: PyYAML>=5.3 in /usr/local/lib/python3.10/dist-packages (from langchain) (6.0.1)
Requirement already satisfied: SQLAlchemy<3,>=1.4 in /usr/local/lib/python3.10/dist-packages (from langchain) (2.0.30)
```

```
Requirement already satisfied: aiohttp<4.0.0,>=3.8.3 in /usr/local/lib/python3.10/dist-packages (from langchain) (3.9.5)
Requirement already satisfied: async-timeout<5.0.0,>=4.0.0 in /usr/local/lib/python3.10/dist-packages (from langchain) (4.0.3)
Collecting langchain-core<0.3.0,>=0.2.6 (from langchain)
  Downloading langchain_core-0.2.6-py3-none-any.whl (315 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 315.5/315.5 kB 4.2 MB/s eta 0:00:00
Collecting langchain-text-splitters<0.3.0,>=0.2.0 (from langchain)
  Downloading langchain_text_splitters-0.2.1-py3-none-any.whl (23 kB)
Collecting langsmith<0.2.0,>=0.1.17 (from langchain)
  Downloading langsmith-0.1.77-py3-none-any.whl (125 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 125.2/125.2 kB 7.0 MB/s eta 0:00:00
Requirement already satisfied: numpy<2,>=1 in /usr/local/lib/python3.10/dist-packages (from langchain) (1.25.2)
Requirement already satisfied: pydantic<3,>=1 in /usr/local/lib/python3.10/dist-packages (from langchain) (2.7.3)
Requirement already satisfied: requests<3,>=2 in /usr/local/lib/python3.10/dist-packages (from langchain) (2.31.0)
Requirement already satisfied: tenacity<9.0.0,>=8.1.0 in /usr/local/lib/python3.10/dist-packages (from langchain) (8.3.0)
Requirement already satisfied: aiosignal>=1.1.2 in /usr/local/lib/python3.10/dist-packages (from aiohttp<4.0.0,>=3.8.3->langchain) (1.3.1)
Requirement already satisfied: attrs>=17.3.0 in /usr/local/lib/python3.10/dist-packages (from aiohttp<4.0.0,>=3.8.3->langchain) (23.2.0)
Requirement already satisfied: frozenlist>=1.1.1 in /usr/local/lib/python3.10/dist-packages (from aiohttp<4.0.0,>=3.8.3->langchain) (1.4.1)
Requirement already satisfied: multidict<7.0,>=4.5 in /usr/local/lib/python3.10/dist-packages (from aiohttp<4.0.0,>=3.8.3->langchain) (6.0.5)
Requirement already satisfied: yarl<2.0,>=1.0 in /usr/local/lib/python3.10/dist-packages (from aiohttp<4.0.0,>=3.8.3->langchain) (1.9.4)
Collecting jsonpatch<2.0,>=1.33 (from langchain-core<0.3.0,>=0.2.6->langchain)
  Downloading jsonpatch-1.33-py2.py3-none-any.whl (12 kB)
Requirement already satisfied: packaging<25,>=23.2 in /usr/local/lib/python3.10/dist-packages (from langchain-core<0.3.0,>=0.2.6->langchain) (24.1)
Collecting orjson<4.0.0,>=3.9.14 (from langsmith<0.2.0,>=0.1.17->langchain)
  Downloading orjson-3.10.5-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (144 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 145.0/145.0 kB 2.5 MB/s eta 0:00:00
Requirement already satisfied: annotated-types>=0.4.0 in /usr/local/lib/python3.10/dist-packages (from pydantic<3,>=1->langchain) (0.7.0)
Requirement already satisfied: pydantic-core==2.18.4 in /usr/local/lib/python3.10/dist-packages (from pydantic<3,>=1->langchain) (2.18.4)
Requirement already satisfied: typing-extensions>=4.6.1 in /usr/local/lib/python3.10/dist-packages (from pydantic<3,>=1->langchain) (4.12.2)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests<3,>=2->langchain) (3.3.2)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests<3,>=2->langchain) (3.7)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests<3,>=2->langchain) (2.0.7)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests<3,>=2->langchain) (2024.6.2)
Requirement already satisfied: greenlet!=0.4.17 in /usr/local/lib/python3.10/dist-packages (from SQLAlchemy<3,>=1.4->langchain) (3.0.3)
Collecting jsonpointer>=1.9 (from jsonpatch<2.0,>=1.33->langchain-core<0.3.0,>=0.2.6->langchain)
  Downloading jsonpointer-3.0.0-py2.py3-none-any.whl (7.6 kB)
Installing collected packages: orjson, jsonpointer, jsonpatch, langsmith, langchain-core, langchain-text-splitters, langchain
Successfully installed jsonpatch-1.33 jsonpointer-3.0.0 langchain-0.2.4 langchain-core-0.2.6 langchain-text-splitters-0.2.1 langsmith-0.1.77 orjson-3.10.5
```

```
!pip install langchain-community
```

```
Collecting langchain-community
  Downloading langchain_community-0.2.4-py3-none-any.whl (2.2 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 2.2/2.2 MB 21.9 MB/s eta 0:00:00
Requirement already satisfied: PyYAML>=5.3 in /usr/local/lib/python3.10/dist-packages (from langchain-community) (6.0.1)
Requirement already satisfied: SQLAlchemy<3,>=1.4 in /usr/local/lib/python3.10/dist-packages (from langchain-community) (2.0.30)
Requirement already satisfied: aiohttp<4.0.0,>=3.8.3 in /usr/local/lib/python3.10/dist-packages (from langchain-community) (3.9.5)
Collecting dataclasses-json<0.7,>=0.5.7 (from langchain-community)
  Downloading dataclasses_json-0.6.7-py3-none-any.whl (28 kB)
Requirement already satisfied: langchain<0.3.0,>=0.2.0 in /usr/local/lib/python3.10/dist-packages (from langchain-community) (0.2.4)
Requirement already satisfied: langchain-core<0.3.0,>=0.2.0 in /usr/local/lib/python3.10/dist-packages (from langchain-community) (0.2.6)
Requirement already satisfied: langsmith<0.2.0,>=0.1.0 in /usr/local/lib/python3.10/dist-packages (from langchain-community) (0.1.77)
Requirement already satisfied: numpy<2,>=1 in /usr/local/lib/python3.10/dist-packages (from langchain-community) (1.25.2)
Requirement already satisfied: requests<3,>=2 in /usr/local/lib/python3.10/dist-packages (from langchain-community) (2.31.0)
Requirement already satisfied: tenacity<9.0.0,>=8.1.0 in /usr/local/lib/python3.10/dist-packages (from langchain-community) (8.3.0)
Requirement already satisfied: aiosignal>=1.1.2 in /usr/local/lib/python3.10/dist-packages (from aiohttp<4.0.0,>=3.8.3->langchain-community) (1.3.1)
Requirement already satisfied: attrs>=17.3.0 in /usr/local/lib/python3.10/dist-packages (from aiohttp<4.0.0,>=3.8.3->langchain-community) (23.2.0)
Requirement already satisfied: frozenlist>=1.1.1 in /usr/local/lib/python3.10/dist-packages (from aiohttp<4.0.0,>=3.8.3->langchain-community) (1.4.1)
Requirement already satisfied: multidict<7.0,>=4.5 in /usr/local/lib/python3.10/dist-packages (from aiohttp<4.0.0,>=3.8.3->langchain-community) (6.0.5)
Requirement already satisfied: yarl<2.0,>=1.0 in /usr/local/lib/python3.10/dist-packages (from aiohttp<4.0.0,>=3.8.3->langchain-community) (1.9.4)
Requirement already satisfied: async-timeout<5.0,>=4.0 in /usr/local/lib/python3.10/dist-packages (from aiohttp<4.0.0,>=3.8.3->langchain-community) (4.0.3)
Collecting marshmallow<4.0.0,>=3.18.0 (from dataclasses-json<0.7,>=0.5.7->langchain-community)
  Downloading marshmallow-3.21.3-py3-none-any.whl (49 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 49.2/49.2 kB 6.7 MB/s eta 0:00:00
```

```
Collecting typing-inspect<1,>=0.4.0 (from dataclasses-json<0.7,>=0.5.7->langchain-community)
  Downloading typing_inspect-0.9.0-py3-none-any.whl (8.8 kB)
Requirement already satisfied: langchain-text-splitters<0.3.0,>=0.2.0 in /usr/local/lib/python3.10/dist-packages (from langchain<0.3.0,>=0.2.0->langchain-community) (0.2.1)
Requirement already satisfied: pydantic<3,>=1 in /usr/local/lib/python3.10/dist-packages (from langchain<0.3.0,>=0.2.0->langchain-community) (2.7.3)
Requirement already satisfied: jsonpatch<2.0,>=1.33 in /usr/local/lib/python3.10/dist-packages (from langchain-core<0.3.0,>=0.2.0->langchain-community) (1.33)
Requirement already satisfied: packaging<25,>=23.2 in /usr/local/lib/python3.10/dist-packages (from langchain-core<0.3.0,>=0.2.0->langchain-community) (24.1)
Requirement already satisfied: orjson<4.0.0,>=3.9.14 in /usr/local/lib/python3.10/dist-packages (from langsmith<0.2.0,>=0.1.0->langchain-community) (3.10.5)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests<3,>=2->langchain-community) (3.3.2)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests<3,>=2->langchain-community) (3.7)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests<3,>=2->langchain-community) (2.0.7)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests<3,>=2->langchain-community) (2024.6.2)
Requirement already satisfied: typing-extensions>=4.6.0 in /usr/local/lib/python3.10/dist-packages (from SQLAlchemy<3,>=1.4->langchain-community) (4.12.2)
Requirement already satisfied: greenlet!=0.4.17 in /usr/local/lib/python3.10/dist-packages (from SQLAlchemy<3,>=1.4->langchain-community) (3.0.3)
Requirement already satisfied: jsonpointer>=1.9 in /usr/local/lib/python3.10/dist-packages (from jsonpatch<2.0,>=1.33->langchain-core<0.3.0,>=0.2.0->langchain-community) (3.0.0)
Requirement already satisfied: annotated-types>=0.4.0 in /usr/local/lib/python3.10/dist-packages (from pydantic<3,>=1->langchain<0.3.0,>=0.2.0->langchain-community) (0.7.0)
Requirement already satisfied: pydantic-core==2.18.4 in /usr/local/lib/python3.10/dist-packages (from pydantic<3,>=1->langchain<0.3.0,>=0.2.0->langchain-community) (2.18.4)
Collecting mypy-extensions>=0.3.0 (from typing-inspect<1,>=0.4.0->dataclasses-json<0.7,>=0.5.7->langchain-community)
  Downloading mypy_extensions-1.0.0-py3-none-any.whl (4.7 kB)
Installing collected packages: mypy-extensions, marshmallow, typing-inspect, dataclasses-json, langchain-community
Successfully installed dataclasses-json-0.6.7 langchain-community-0.2.4 marshmallow-3.21.3 mypy-extensions-1.0.0 typing-inspect-0.9.0
```

```
!pip install pinecone
```

```
Collecting pinecone
  Downloading pinecone-4.0.0-py3-none-any.whl (214 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 214.4/214.4 kB 3.6 MB/s eta 0:00:00
Requirement already satisfied: certifi>=2019.11.17 in /usr/local/lib/python3.10/dist-packages (from pinecone) (2024.6.2)
Requirement already satisfied: tqdm>=4.64.1 in /usr/local/lib/python3.10/dist-packages (from pinecone) (4.66.4)
Requirement already satisfied: typing-extensions>=3.7.4 in /usr/local/lib/python3.10/dist-packages (from pinecone) (4.12.2)
Requirement already satisfied: urllib3>=1.26.0 in /usr/local/lib/python3.10/dist-packages (from pinecone) (2.0.7)
Installing collected packages: pinecone
Successfully installed pinecone-4.0.0
```

```
!pip install dotenv
```

```
Collecting dotenv
  Using cached dotenv-0.0.5.tar.gz (2.4 kB)
  error: subprocess-exited-with-error

  × python setup.py egg_info did not run successfully.
  │ exit code: 1
  ╰─> See above for output.

  note: This error originates from a subprocess, and is likely not a problem with pip.
  Preparing metadata (setup.py) ... error
error: metadata-generation-failed

× Encountered error while generating package metadata.
╰─> See above for output.

note: This is an issue with the package mentioned above, not pip.
hint: See above for details.
```

```
# Make the display a bit wider
# from IPython.display import display, HTML
# display(HTML("<style>.container { width:90% !important; }</style>"))

# LangChain basics
from langchain.chat_models import ChatOpenAI
from langchain.text_splitter import RecursiveCharacterTextSplitter
from langchain.chains.summarize import load_summarize_chain
from langchain.chains import create_extraction_chain

# Vector Store and retrievals
from langchain.embeddings.openai import OpenAIEmbeddings
from langchain.chains import RetrievalQA
from langchain.vectorstores import Chroma, Pinecone
import pinecone

# Chat Prompt templates for dynamic values
from langchain.prompts.chat import (
    ChatPromptTemplate,
    SystemMessagePromptTemplate,
    HumanMessagePromptTemplate
)

# Supporting libraries
import os
from dotenv import load_dotenv

load_dotenv()
```

```
---------------------------------------------------------------------------
ModuleNotFoundError                       Traceback (most recent call last)
<ipython-input-11-5cb3c684c9bb> in <cell line: 26>()
     24 # Supporting libraries
     25 import os
---> 26 from dotenv import load_dotenv
     27
     28 load_dotenv()

ModuleNotFoundError: No module named 'dotenv'

---------------------------------------------------------------------------
NOTE: If your import is failing due to a missing package, you can
manually install dependencies using either !pip or !apt.

To view examples of installing some common dependencies, click the
"Open Examples" button below.
---------------------------------------------------------------------------
```

OPEN EXAMPLES

```
!pip install openai
```

```
Collecting openai
  Downloading openai-1.34.0-py3-none-any.whl (325 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 325.5/325.5 kB 5.4 MB/s eta 0:00:00
Requirement already satisfied: anyio<5,>=3.5.0 in /usr/local/lib/python3.10/dist-packages (from openai) (3.7.1)
Requirement already satisfied: distro<2,>=1.7.0 in /usr/lib/python3/dist-packages (from openai) (1.7.0)
Collecting httpx<1,>=0.23.0 (from openai)
  Downloading httpx-0.27.0-py3-none-any.whl (75 kB)
```

```
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 75.6/75.6 kB 10.1 MB/s eta 0:00:00
Requirement already satisfied: pydantic<3,>=1.9.0 in /usr/local/lib/python3.10/dist-packages (from openai) (2.7.3)
Requirement already satisfied: sniffio in /usr/local/lib/python3.10/dist-packages (from openai) (1.3.1)
Requirement already satisfied: tqdm>4 in /usr/local/lib/python3.10/dist-packages (from openai) (4.66.4)
Requirement already satisfied: typing-extensions<5,>=4.7 in /usr/local/lib/python3.10/dist-packages (from openai) (4.12.2)
Requirement already satisfied: idna>=2.8 in /usr/local/lib/python3.10/dist-packages (from anyio<5,>=3.5.0->openai) (3.7)
Requirement already satisfied: exceptiongroup in /usr/local/lib/python3.10/dist-packages (from anyio<5,>=3.5.0->openai) (1.2.1)
Requirement already satisfied: certifi in /usr/local/lib/python3.10/dist-packages (from httpx<1,>=0.23.0->openai) (2024.6.2)
Collecting httpcore==1.* (from httpx<1,>=0.23.0->openai)
  Downloading httpcore-1.0.5-py3-none-any.whl (77 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 77.9/77.9 kB 9.0 MB/s eta 0:00:00
Collecting h11<0.15,>=0.13 (from httpcore==1.*->httpx<1,>=0.23.0->openai)
  Downloading h11-0.14.0-py3-none-any.whl (58 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 58.3/58.3 kB 7.8 MB/s eta 0:00:00
Requirement already satisfied: annotated-types>=0.4.0 in /usr/local/lib/python3.10/dist-packages (from pydantic<3,>=1.9.0->openai) (0.7.0)
Requirement already satisfied: pydantic-core==2.18.4 in /usr/local/lib/python3.10/dist-packages (from pydantic<3,>=1.9.0->openai) (2.18.4)
Installing collected packages: h11, httpcore, httpx, openai
Successfully installed h11-0.14.0 httpcore-1.0.5 httpx-0.27.0 openai-1.34.0
```

The Set Up - Create your LLMs and get data

```
# Creating two versions of the model so I can swap between gpt3.5 and gpt4
llm3 = ChatOpenAI(temperature=0,
                  openai_api_key=os.getenv('OPENAI_API_KEY', 'YourAPIKeyIfNotSet'),
                  model_name="gpt-3.5-turbo-0613",
                  request_timeout = 180
                 )

llm4 = ChatOpenAI(temperature=0,
                  openai_api_key=os.getenv('OPENAI_API_KEY', 'YourAPIKeyIfNotSet'),
                  model_name="gpt-4-0613",
                  request_timeout = 180
                 )
```

```
/usr/local/lib/python3.10/dist-packages/langchain_core/_api/deprecation.py:119: LangChainDeprecationWarning: The class
  warn_deprecated(
---------------------------------------------------------------------------
ModuleNotFoundError                       Traceback (most recent call last)
/usr/local/lib/python3.10/dist-packages/langchain_community/chat_models/openai.py in validate_environment(cls, values)
    303         try:
--> 304             import openai
    305

ModuleNotFoundError: No module named 'openai'

During handling of the above exception, another exception occurred:

ImportError                               Traceback (most recent call last)
                                          4 frames
/usr/local/lib/python3.10/dist-packages/langchain_community/chat_models/openai.py in validate_environment(cls, values)
    305
    306         except ImportError:
--> 307             raise ImportError(
    308                 "Could not import openai python package. "
    309                 "Please install it with `pip install openai`."

ImportError: Could not import openai python package. Please install it with `pip install openai`.

---------------------------------------------------------------------------
NOTE: If your import is failing due to a missing package, you can
manually install dependencies using either !pip or !apt.

To view examples of installing some common dependencies, click the
"Open Examples" button below.
---------------------------------------------------------------------------
```

OPEN EXAMPLES

First we'll need to get transcripts. I put a few pre-processed transcripts in the data folder of this repo.

If you need transcripts for your own audio I suggest a transcription tool like AssemblyAI. I also tried Steno.ai but the quality and speaker detection wasn't that high.

Reach out if you want me to grab transcripts in bulk for you.

```
# I put three prepared transcripts
transcript_paths = [
    '../data/Transcripts/MFMPod/mfm_pod_steph.txt',
    '../data/Transcripts/MFMPod/mfm_pod_alex.txt',
    '../data/Transcripts/MFMPod/mfm_pod_rob.txt'
]

with open('../data/Transcripts/MFMPod/mfm_pod_steph.txt') as file:
    transcript = file.read()


print(transcript[:280])
```

Then we are going to split our text up into chunks. We do this so:

1. The context size is smaller and the LLM can increase it's attention to context ratio
2. In case the text is too long and it wouldn't fit in the prompt anyway

```
# Load up your text splitter
text_splitter = RecursiveCharacterTextSplitter(separators=["\n\n", "\n", " "], chunk_size=10000, chunk_overlap=2200)

# I'm only doing the first 23250 characters. This to save on costs. When you're doing your exercise you can remove this to let all the data through
transcript_subsection_characters = 23250
docs = text_splitter.create_documents([transcript[:transcript_subsection_characters]])
print (f"You have {len(docs)} docs. First doc is {llm3.get_num_tokens(docs[0].page_content)} tokens")
```

Step 1: Extract Topic Titles & Short Description

The Custom Prompts - Customize your prompt to fit your use case

Next up I'm going to use custom prompts to instruct the LLM on how to pull out the topics I want.

This will be heavily dependent on your domain. You should adjust the prompt below to use descriptions and examples that are relevant to you.

I built these descriptions over many iterations playing with prompts and checking the output. If you ever start a business this will be part of your IP!

I will ask the LLM for a topic title and a short description. I found it was too much for the LLM to ask for a long description in the first pass. Results weren't great and high latency.

Let's start with our map prompt which will iterate over the chunks we just made

```
template="""
You are a helpful assistant that helps retrieve topics talked about in a podcast transcript
- Your goal is to extract the topic names and brief 1-sentence description of the topic
- Topics include:
  - Themes
  - Business Ideas
  - Interesting Stories
  - Money making businesses
  - Quick stories about people
  - Mental Frameworks
  - Stories about an industry
  - Analogies mentioned
  - Advice or words of caution
  - Pieces of news or current events
- Provide a brief description of the topics after the topic name. Example: 'Topic: Brief Description'
- Use the same words and terminology that is said in the podcast
- Do not respond with anything outside of the podcast. If you don't see any topics, say, 'No Topics'
- Do not respond with numbers, just bullet points
- Do not include anything about 'Marketing Against the Grain'
- Only pull topics from the transcript. Do not use the examples
- Make your titles descriptive but concise. Example: 'Shaan's Experience at Twitch' should be 'Shaan's Interesting Projects At Twitch'
- A topic should be substantial, more than just a one-off comment

% START OF EXAMPLES
 - Sam’s Elisabeth Murdoch Story: Sam got a call from Elizabeth Murdoch when he had just launched The Hustle. She wanted to generate video content.
 - Shaan’s Rupert Murdoch Story: When Shaan was running Blab he was invited to an event organized by Rupert Murdoch during CES in Las Vegas.
 - Revenge Against The Spam Calls: A couple of businesses focused on protecting consumers: RoboCall, TrueCaller, DoNotPay, FitIt
 - Wildcard CEOs vs. Prudent CEOs: However, Munger likes to surround himself with prudent CEO’s and says he would never hire Musk.
 - Chess Business: Priyav, a college student, expressed his doubts on the MFM Facebook group about his Chess training business, mychesstutor.com, making $12.5K MRR with 90 enrolled.
 - Restaurant Refiller: An MFM Facebook group member commented on how they pay AirMark $1,000/month for toilet paper and toilet cover refills for their restaurant. Shaan sees an opportuni
 - Collecting: Shaan shared an idea to build a mobile only marketplace for a collectors’ category; similar to what StockX does for premium sneakers.
% END OF EXAMPLES
"""
system_message_prompt_map = SystemMessagePromptTemplate.from_template(template)

human_template="Transcript: {text}" # Simply just pass the text as a human message
human_message_prompt_map = HumanMessagePromptTemplate.from_template(human_template)

chat_prompt_map = ChatPromptTemplate.from_messages(messages=[system_message_prompt_map, human_message_prompt_map])
```

Then we have our combine prompt which will run once over the results of the map prompt above

```
template="""
You are a helpful assistant that helps retrieve topics talked about in a podcast transcript
- You will be given a series of bullet topics of topics vound
- Your goal is to exract the topic names and brief 1-sentence description of the topic
- Deduplicate any bullet points you see
- Only pull topics from the transcript. Do not use the examples

% START OF EXAMPLES
 - Sam’s Elisabeth Murdoch Story: Sam got a call from Elizabeth Murdoch when he had just launched The Hustle. She wanted to generate video content.
 - Shaan’s Rupert Murdoch Story: When Shaan was running Blab he was invited to an event organized by Rupert Murdoch during CES in Las Vegas.
% END OF EXAMPLES
"""
system_message_prompt_map = SystemMessagePromptTemplate.from_template(template)

human_template="Transcript: {text}" # Simply just pass the text as a human message
human_message_prompt_map = HumanMessagePromptTemplate.from_template(human_template)

chat_prompt_combine = ChatPromptTemplate.from_messages(messages=[system_message_prompt_map, human_message_prompt_map])
```

The First Pass - Run through your text and extract the topics per your custom prompts

Then we get our chain ready. This is object that will do the actual processing for us when we call it. I'm using gpt4 because we need the increased reasoning ability to pull out topics. You could use gpt3.5 but results may vary.

```
chain = load_summarize_chain(llm4,
                             chain_type="map_reduce",
                             map_prompt=chat_prompt_map,
                             combine_prompt=chat_prompt_combine,
#                              verbose=True
                            )
```

Then the `.run()` code below will do the actual API calls and work

```
topics_found = chain.run({"input_documents": docs})
```

```
print (topics_found)
```

Structured Data - Turn your LLM output into structured data

The LLM just returned a wall of text to us, I want to convert this into structured data I can more easily use elsewhere.

We might have been able to do add structured output instructions to the pull above but I preferred to do it in two steps for clarity. Plus the cost us super low so we only have latency to worry about, but that isn't a priority for this tutorial.

We will use OpenAI's [function calling](function Calling via ChatGPT API - First Look With LangChain - YouTube) to extract each topic.

```
schema = {
    "properties": {
        # The title of the topic
        "topic_name": {
            "type": "string",
            "description" : "The title of the topic listed"
        },
        # The description
        "description": {
            "type": "string",
            "description" : "The description of the topic listed"
        },
        "tag": {
            "type": "string",
            "description" : "The type of content being described",
            "enum" : ['Business Models', 'Life Advice', 'Health & Wellness', 'Stories']
        }
    },
    "required": ["topic", "description"],
}


# Using gpt3.5 here because this is an easy extraction task and no need to jump to gpt4
chain = create_extraction_chain(schema, llm3)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-2-62dfdcdddf58> in <cell line: 2>()
      1 # Using gpt3.5 here because this is an easy extraction task and no need to jump to gpt4
----> 2 chain = create_extraction_chain(schema, llm3)

NameError: name 'create_extraction_chain' is not defined
```

```
topics_structured = chain.run(topics_found)


topics_structured
```

Great, now we have our structured topics. Let's move into the next step and *expand* on those topics even more.

Step 2: Expand on the topics you found

In order to expand on the topics we found we are going to do the vectorstore dance. We'll chunk our podcast into *small* chunks and then modify the retrieval and qa chain to help us pull out more information.

I want to split into small chunks to hopefully increase the signal to noise ratio. Here I'll only do 4K characters which is less than half of what we did above.

```
text_splitter = RecursiveCharacterTextSplitter(chunk_size=4000, chunk_overlap=800)

docs = text_splitter.create_documents([transcript[:transcript_subsection_characters]])

print (f"You have {len(docs)} docs. First doc is {llm3.get_num_tokens(docs[0].page_content)} tokens")
```

Because I want to do Question & Answer Retrieval, we need to get embeddings for our documents so we can pull out the docs which are similar for context later.

```
embeddings = OpenAIEmbeddings(openai_api_key=os.getenv('OPENAI_API_KEY', 'YourAPIKeyIfNotSet'))
```

Option #1: Pinecone

Use this if you're looking for scale in the cloud

```
# initialize pinecone
pinecone.init(
    api_key=os.getenv('PINECONE_API_KEY', 'YourAPIKeyIfNotSet'),  # find at app.pinecone.io
    environment=os.getenv('PINECONE_ENV', 'YourAPIKeyIfNotSet'),  # next to api key in console
)

index_name = "topic-modeling"

docsearch = Pinecone.from_documents(docs, embeddings, index_name=index_name)

# # If you want to delete your vectors in your index to start over, run the code below!
# index = pinecone.Index(index_name)
# index.delete(delete_all='true')
```

Option #2: Chroma

Use this if you're looking for local and easy to set up

```
# load it into Chroma
docsearch = Chroma.from_documents(docs, embeddings)
```

Then we are going to create a custom prompt for our Retriever. I'm doing this because the out of the out-of-the-box prompt used here isn't bad, but a bit generic for my use case. Plus, I only really want to *answer a question* I want to generated a mini-summary based off of docs.

Let's switch it up!

```
# The system instructions. Notice the 'context' placeholder down below. This is where our relevant docs will go.
# The 'question' in the human message below won't be a question per se, but rather a topic we want to get relevant information on

# I'm using gpt4 for the increased reasoning power.
# I'm also setting k=4 so the number of relevant docs we get back is 4. This parameter should be tuned to your use case
qa = RetrievalQA.from_chain_type(llm=llm4,
                                 chain_type="stuff",
                                 retriever=docsearch.as_retriever(k=4),
                                 chain_type_kwargs = {
#                                       'verbose': True,
                                     'prompt': CHAT_PROMPT
                                 })
```

Then let's iterate through the topics that we found and run our QA query on them.

This will print out our expanded topics. This is the final result you can use wherever you want!

```
# Only doing the first 3 for conciseness
for topic in topics_structured[:5]:
    query = f"""
        {topic['topic_name']}: {topic['description']}
    """

    expanded_topic = qa.run(query)

    print(f"{topic['topic_name']}: {topic['description']}")
    print(expanded_topic)
    print ("\n\n")
```

Bonus: Chapters With Timestamps

Because why not?

We have the timestamps on the transcript so let's pull them out and get timestamp chapters. This is helpful so you can scrub to the topic when