Chatbot for economic topics

Mai Xuan Man 20182014

Why I chose this topic

On big scale, economy plays an important role in our daily life without us being aware.

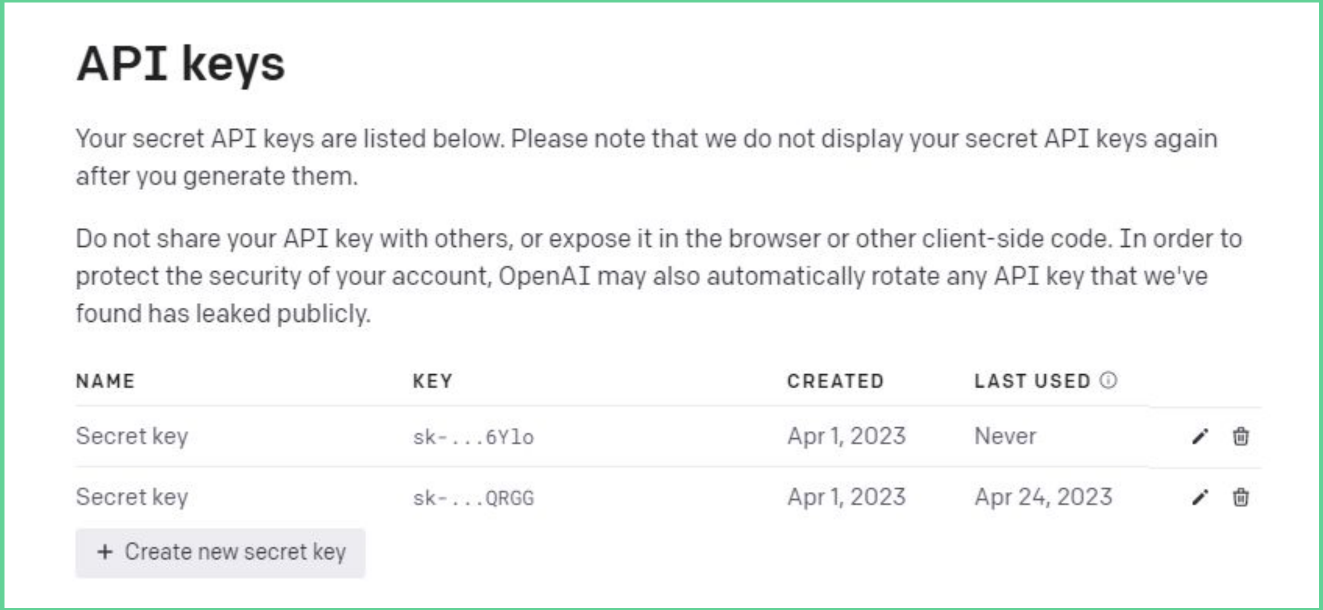
So educating people about economy can help people making good decisions about personal finances or on bigger impact, a nation financial health.

So a chatbot is a feasible solution for this goal

Implementation

Set up OpenAPI key

The key will be used to access to Api free plan



Install environment (optional)

For Linux/MacOS

```
python -m venv env

source env/bin/activate

pip install -r requirements.txt
```

For Window

```
python -m venv env

. env/scripts/activate

pip install -r requirements.txt
```

Set up new virtual environment and install requirement packages.
This serve to avoid any issues during implementation.

Requirement packages

```
aiohttp==3.8.3
aiosignal==1.3.1
appnope==0.1.3
asttokens==2.2.1
async-timeout==4.0.2
attrs==22.2.0
backcall==0.2.0
beautifulsoup4==4.11.1
blobfile==2.0.1
bs4==0.0.1
certifi==2022.12.7
charset-normalizer==2.1.1
comm==0.1.2
contourpy==1.0.7
cycler==0.11.0
debugpy==1.6.5
decorator==5.1.1
docopt==0.6.2
entrypoints==0.4
executing==1.2.0
filelock==3.9.0
fonttools==4.38.0
frozenlist==1.3.3
html==1.13
huggingface-hub==0.11.1
idna==3.4
ipykernel==6.20.1
ipython==8.8.0
jedi==0.18.2
joblib==1.2.0
jupyter_client==7.4.8
jupyter_core==5.1.3
kiwisolver==1.4.4
lxml==4.9.2
```

```
matplotlib==3.6.3
matplotlib-inline==0.1.6
multidict==6.0.4
nest-asyncio==1.5.6
numpy==1.24.1
openai==0.26.1
packaging==23.0
pandas==1.5.2
parso==0.8.3
pexpect==4.8.0
pickleshare==0.7.5
Pillow==9.4.0
pipreqs==0.4.11
platformdirs==2.6.2
plotly==5.12.0
prompt-toolkit==3.0.36
psutil==5.9.4
ptyprocess==0.7.0
pure-eval==0.2.2
pycryptodomex==3.17
Pygments==2.14.0
pyparsing==3.0.9
python-dateutil==2.8.2
pytz==2022.7.1
PyYAML==6.0
pyzmq==24.0.1
regex==2022.10.31
requests==2.28.1
scikit-learn==1.2.0
scipy==1.10.0
six==1.16.0
soupsieve==2.3.2.post1
stack-data==0.6.2
```

scikit-learn==1.2.0
scipy==1.10.0
six==1.16.0
soupsieve==2.3.2.post1
stack-data==0.6.2

```
tenacity==8.1.0
threadpoolctl==3.1.0
tiktoken==0.1.2
tokenizers==0.13.2
tornado==6.2
tqdm==4.64.1
traitlets==5.8.1
transformers==4.25.1
typing_extensions==4.4.0
urllib3==1.26.13
wcwidth==0.2.5
yarg==0.1.9
yarl==1.8.2
```

Create a web crawler

- After setting up environment, we're going to create a web crawler to crawl the information from a desired site and store information into text files:
  - Import the required packages
  - Set up basic URL and create HTMLParser class

Create a web crawler

```python
import requests
import re
import urllib.request
from bs4 import BeautifulSoup
from collections import deque
from html.parser import HTMLParser
from urllib.parse import urlparse
import os
```

```python
# Regex pattern to match a URL
HTTP_URL_PATTERN = r'^http[s]*://.+'

# Define root domain to crawl
domain = "economicprinciples.org"
full_url = "https://economicprinciples.org/"

# Create a class to parse the HTML and get the hyperlinks
class HyperlinkParser(HTMLParser):
    def __init__(self):
        super().__init__()
        # Create a list to store the hyperlinks
        self.hyperlinks = []

    # Override the HTMLParser's handle_starttag method to get the hyperlinks
    def handle_starttag(self, tag, attrs):
        attrs = dict(attrs)

        # If the tag is an anchor tag and it has an href attribute, add the href attribute to the list of hyperlinks
        if tag == "a" and "href" in attrs:
            self.hyperlinks.append(attrs["href"])
```

Create a web crawler

- Create function:
  - that takes URL as an argument
  - opens the URL
  - reads the HTML content.
  - Then, it returns all the hyperlinks found on that page

Create a web crawler

```
# Function to get the hyperlinks from a URL
def get_hyperlinks(url):
    
    # Try to open the URL and read the HTML
    try:
        # Open the URL and read the HTML
        with urllib.request.urlopen(url) as response:

            # If the response is not HTML, return an empty list
            if not response.info().get('Content-Type').startswith("text/html"):
                return []

            # Decode the HTML
            html = response.read().decode('utf-8')
    except Exception as e:
        print(e)
        return []

    # Create the HTML Parser and then Parse the HTML to get hyperlinks
    parser = HyperlinkParser()
    parser.feed(html)

    return parser.hyperlinks
```

Create a web crawler

- We try to crawl through and index only the content that shares same EconomicPrinciples domain. So we create a function:
    - calls the get_hyperlinks function but filters out any URLs that are not part of the specified domain is needed.

Create a web crawler

```python
# Function to get the hyperlinks from a URL that are within the same domain
def get_domain_hyperlinks(local_domain, url):
    clean_links = []
    for link in set(get_hyperlinks(url)):
        clean_link = None

        # If the link is a URL, check if it is within the same domain
        if re.search(HTTP_URL_PATTERN, link):
            # Parse the URL and check if the domain is the same
            url_obj = urlparse(link)
            if url_obj.netloc == local_domain:
                clean_link = link

        # If the link is not a URL, check if it is a relative link
        else:
            if link.startswith("/"):
                link = link[1:]
            elif link.startswith("#") or link.startswith("mailto:"):
                continue
            clean_link = "https://" + local_domain + "/" + link

        if clean_link is not None:
            if clean_link.endswith("/"):
                clean_link = clean_link[:-1]
            clean_links.append(clean_link)

    # Return the list of hyperlinks that are within the same domain
    return list(set(clean_links))
```

Create a web crawler

- Finally we create a crawl function:
  - It keeps track of the visited URLs to avoid repeating the same page, which might be linked across multiple pages on a site.
  - extracts the raw text from a page without the HTML tags
  - writes the text content into a local .txt file specific to the page.

Create a web crawler

```python
def crawl(url):
    # Parse the URL and get the domain
    local_domain = urlparse(url).netloc

    # Create a queue to store the URLs to crawl
    queue = deque([url])

    # Create a set to store the URLs that have already been seen (no duplicates)
    seen = set([url])

    # Create a directory to store the text files
    if not os.path.exists("text/"):
            os.mkdir("text/")

    if not os.path.exists("text/"+local_domain+"/"):
            os.mkdir("text/" + local_domain + "/")

    # Create a directory to store the csv files
    if not os.path.exists("processed"):
            os.mkdir("processed")

    # While the queue is not empty, continue crawling
    while queue:

        # Get the next URL from the queue
        url = queue.pop()
        print(url) # for debugging and to see the progress
```

```python
        # Save text from the url to a <url>.txt file
        with open('text/'+local_domain+'/'+url[8:].replace("/", "_") + ".txt", "w") as f:

            # Get the text from the URL using BeautifulSoup
            soup = BeautifulSoup(requests.get(url).text, "html.parser")

            # Get the text but remove the tags
            text = soup.get_text()

            # If the crawler gets to a page that requires JavaScript, it will stop the crawl
            if ("You need to enable JavaScript to run this app." in text):
                print("Unable to parse page " + url + " due to JavaScript being required")

            # Otherwise, write the text to the file in the text directory
            f.write(text)

        # Get the hyperlinks from the URL and add them to the queue
        for link in get_domain_hyperlinks(local_domain, url):
            if link not in seen:
                queue.append(link)
                seen.add(link)

crawl(full_url)
```

Crawler implementation result

```
https://economicprinciples.org/
https://economicprinciples.org/downloads/MMT_ MP3_MK.pdf
URL can't contain control characters. '/downloads/MMT_ MP3_MK.pdf' (found at least ' ')
https://economicprinciples.org/downloads/cwo-a-deeper-look-at-capital-wars.pdf
https://economicprinciples.org/downloads/cwo-citations-and-bibliography.pdf
https://economicprinciples.org/downloads/cwo-large-drivers-of-life-expectancy-through-time.pdf
https://economicprinciples.org/subscribe
https://economicprinciples.org/Why-and-How-Capitalism-Needs-To-Be-Reformed
https://economicprinciples.org/..
HTTP Error 400: Bad Request
https://economicprinciples.org/downloads/bwam102317.pdf
https://economicprinciples.org/downloads/Paradigm-Shifts.pdf
https://economicprinciples.org/downloads/bw-populism-the-phenomenon.pdf
https://economicprinciples.org/downloads/ray_dalio__how_the_economic_machine_works__leveragings_and_deleveragings.pdf
https://economicprinciples.org
https://economicprinciples.org/downloads/Primer-on-Universal-Basic-Income.pdf
```

The scrapped contents include html file and pdf files, which are then converted to txt file

Crawler implementation result

```
[ ]  from google.colab import drive
     drive.mount('/content/drive')

     Mounted at /content/drive

[ ]  mv /content/text/economicprinciples.org /content/drive/MyDrive/thesis/text
```

- Next we mount to google drive and move the generated text files into the desired folder.
- We need to do this because the file created from Google colab are not saved and will be deleted when colab window are closed.

Building an embeddings index

Building an embeddings index

- Blank empty lines can clutter the text files and make them harder to process. So we will create a function:
  - remove those lines and tidy up the files.

```
def remove_newlines(serie):
    serie = serie.str.replace('\n', ' ')
    serie = serie.str.replace('\\n', ' ')
    serie = serie.str.replace('  ', ' ')
    serie = serie.str.replace('  ', ' ')
    return serie
```

Building an embeddings index

- Now we will convert the text to CSV
    - looping through the text files in the text directory created earlier.
    - After opening each file, remove the extra spacing and append the modified text to a list.
    - Adding the text with the new lines removed to an empty Pandas data frame.
    - Writing the data frame to a CSV file.

Building an embeddings index

```
import pandas as pd

# Create a list to store the text files
texts=[]

# Get all the text files in the text directory
for file in os.listdir("/content/drive/MyDrive/thesis/text/" + domain + "/"):

    # Open the file and read the text
    with open("/content/drive/MyDrive/thesis/text/" + domain + "/" + file, "r") as f:
        text = f.read()

        # Omit the first 11 lines and the last 4 lines, then replace -, _, and #update with spaces.
        texts.append((file[11:-4].replace('-',' ').replace('_', ' ').replace('#update',''), text))

# Create a dataframe from the list of texts
df = pd.DataFrame(texts, columns = ['fname', 'text'])

# Set the text column to be the raw text with the newlines removed
df['text'] = df.fname + ". " + remove_newlines(df.text)
df.to_csv('/content/drive/MyDrive/thesis/processed/scraped.csv')
df.head()
```

Building an embeddings index

```
<ipython-input-2-429202555403>:3: FutureWarning: The default value of regex will change from True to False in a future version.
  serie = serie.str.replace('\\n', ' ')
```

	fname	text
0	nciples.org	nciples.org . Economic Principles ...
1	nciples.org Why and How Capitalism Needs To Be...	nciples.org Why and How Capitalism Needs To Be...
2	nciples.org ..	nciples.org ... Economic Principles ...
3	nciples.org subscribe	nciples.org subscribe. - Get the l...
4	nciples.org	nciples.org. Economic Principles ...

- This is the resulting data frame in CSV file generated from the text files

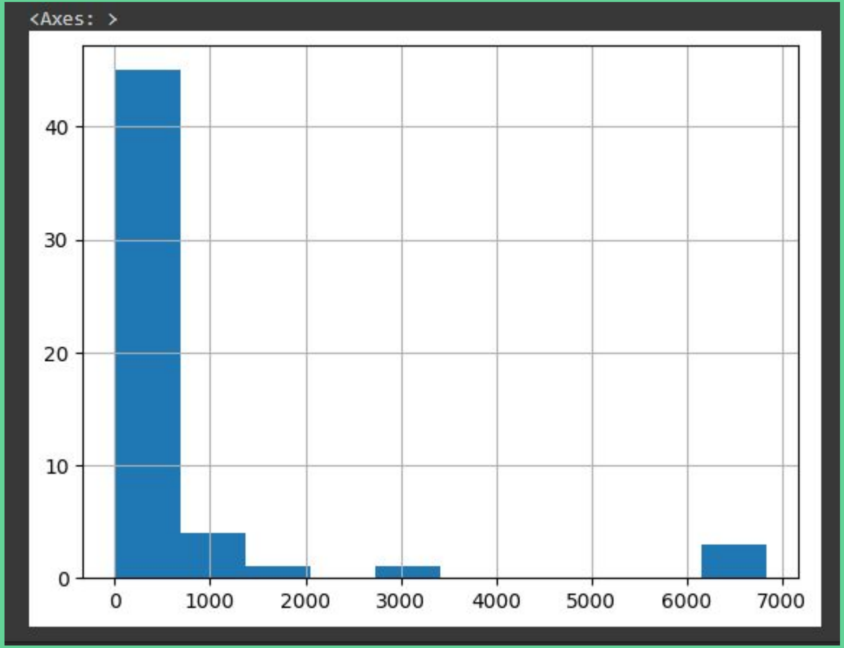
Building an embeddings index

- Next we will tokenize data inside CSV file
    - splitting the input text into tokens by breaking down the sentences and words.
- The OpenAI API has a limit on the maximum number of input tokens for embeddings. To stay below the limit, the text in the CSV file needs to be broken down into multiple rows.
- The existing length of each row will be recorded first to identify which rows need to be split

Building an embeddings index

```
import tiktoken

# Load the cl100k_base tokenizer which is designed to work with the ada-002 model
tokenizer = tiktoken.get_encoding("cl100k_base")

df = pd.read_csv('/content/drive/MyDrive/thesis/processed/scraped.csv', index_col=0)
df.columns = ['title', 'text']

# Tokenize the text and save the number of tokens to a new column
df['n_tokens'] = df.text.apply(lambda x: len(tokenizer.encode(x)))

# Visualize the distribution of the number of tokens per row using a histogram
df.n_tokens.hist()
```

Building an embeddings index



- This shows that the initial text contains very long sequences and thus need to be broken down

Building an embeddings index

- The newest OpenAI embeddings model(**ada-002 model**) can handle inputs with up to 8191 input tokens so most of the rows would not need any chunking
- But this may not be the case for every subpage scraped so we will chunk will split the longer lines into smaller chunks.

Building an embeddings index

```
max_tokens = 500

# Function to split the text into chunks of a maximum number of tokens
def split_into_many(text, max_tokens = max_tokens):

    # Split the text into sentences
    sentences = text.split('. ')

    # Get the number of tokens for each sentence
    n_tokens = [len(tokenizer.encode(" " + sentence)) for sentence in sentences]

    chunks = []
    tokens_so_far = 0
    chunk = []

```

Building an embeddings index

```python
    # Loop through the sentences and tokens joined together in a tuple
    for sentence, token in zip(sentences, n_tokens):

        # If the number of tokens so far plus the number of tokens in the current sentence is greater
        # than the max number of tokens, then add the chunk to the list of chunks and reset
        # the chunk and tokens so far
        if tokens_so_far + token > max_tokens:
            chunks.append(". ".join(chunk) + ".")
            chunk = []
            tokens_so_far = 0

        # If the number of tokens in the current sentence is greater than the max number of
        # tokens, go to the next sentence
        if token > max_tokens:
            continue

        # Otherwise, add the sentence to the chunk and add the number of tokens to the total
        chunk.append(sentence)
        tokens_so_far += token + 1

    return chunks

```

Building an embeddings index

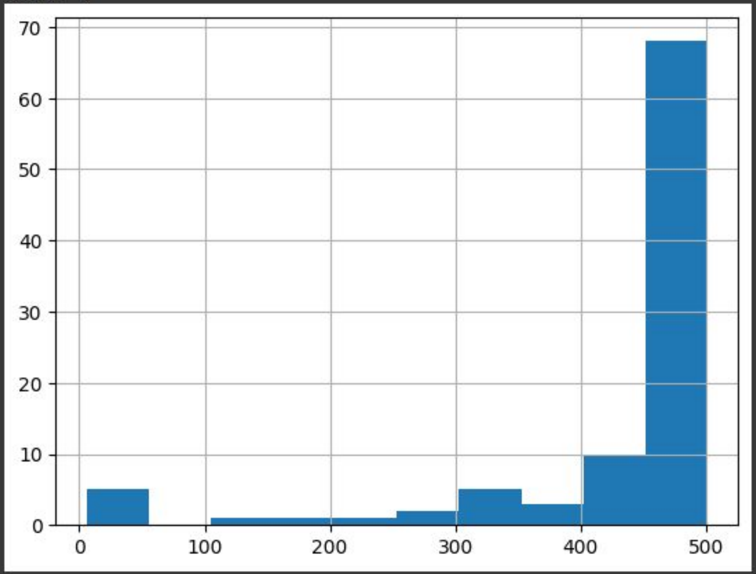
```python

shortened = []

# Loop through the dataframe
for row in df.iterrows():

    # If the text is None, go to the next row
    if row[1]['text'] is None:
        continue

    # If the number of tokens is greater than the max number of tokens, split the text into chunks
    if row[1]['n_tokens'] > max_tokens:
        shortened += split_into_many(row[1]['text'])

    # Otherwise, add the text to the list of shortened texts
    else:
        shortened.append( row[1]['text'] )
```

Building an embeddings index

- The updated histogram confirm that the rows were successfully split into shortened sections.

```
[ ]  df = pd.DataFrame(shortened, columns = ['text'])
     df['n_tokens'] = df.text.apply(lambda x: len(tokenizer.encode(x)))
     df.n_tokens.hist()
```

```
<Axes: >
```



Building an embeddings index

- Now we can send request to the OpenAI API specifically, new text-embedding-ada-002 model, to create the embeddings
- For this, we need to import openai package
- And specify the open ai personal key too
- Since I use the free plan, I will generate the embedding once only

Building an embeddings index

```
[ ]  !pip install openai

    Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
    Collecting openai
      Downloading openai-0.27.6-py3-none-any.whl (71 kB)
         ---------------------------------------- 71.9/71.9 kB 4.2 MB/s eta 0:00:00
    Requirement already satisfied: requests>=2.20 in /usr/local/lib/python3.10/dist-packages (from openai) (2.27.1)
    Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-packages (from openai) (4.65.0)
    Collecting aiohttp (from openai)
      Downloading aiohttp-3.8.4-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (1.0 MB)
         ---------------------------------------- 1.0/1.0 MB 28.1 MB/s eta 0:00:00
    Requirement already satisfied: urllib3<1.27,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests>=2.20->openai) (1.26.15)
    Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests>=2.20->openai) (2022.12.7)
    Requirement already satisfied: charset-normalizer~=2.0.0 in /usr/local/lib/python3.10/dist-packages (from requests>=2.20->openai) (2.0.12)
    Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests>=2.20->openai) (3.4)
    Requirement already satisfied: attrs>=17.3.0 in /usr/local/lib/python3.10/dist-packages (from aiohttp->openai) (23.1.0)
    Collecting multidict<7.0,>=4.5 (from aiohttp->openai)
      Downloading multidict-6.0.4-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (114 kB)
         ---------------------------------------- 114.5/114.5 kB 14.6 MB/s eta 0:00:00
    Collecting async-timeout<5.0,>=4.0.0a3 (from aiohttp->openai)
      Downloading async_timeout-4.0.2-py3-none-any.whl (5.8 kB)
    Collecting yarl<2.0,>=1.0 (from aiohttp->openai)
      Downloading yarl-1.9.2-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (268 kB)
         ---------------------------------------- 268.8/268.8 kB 21.5 MB/s eta 0:00:00
```

```
[ ]  openai.api_key = "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"
```

Building an embeddings index

- This code section will allow us to retrieve embeddings

```
[ ]  import openai

     df['embeddings'] = df.text.apply(lambda x: openai.Embedding.create(input=x, engine='text-embedding-ada-002')['data'][0]['embedding'])
     df.to_csv('/content/drive/MyDrive/thesis/processed/embeddings.csv')
     df.head()
```

Building an embeddings index

	text	n_tokens	embeddings						
0	nciples.or	495	[0.005654186941683292, -0.009135925211012363, 0.010280339978635311, -						
7.17E+13	-0.00015	0.017367	-0.02145	-0.00426	0.032018	-0.01863	0.001162	0.004125	-0.00525
1	I believe t	479	[-0.003091058460995555, -0.00577879510819912, -0.003523419611155987, -						
24033833	-0.00187	-0.00756	-0.0072	0.006924	-0.01924	-0.01074	0.037919	0.003381	-0.00774
2	Because I	486	[-0.016110556200146675, -0.026599613949656487, 0.015779880806803703,						
4.65E+14	0.013915	-0.02369	-0.00907	0.016984	0.008188	-0.00393	-0.00627	-0.01138	0.008822
3	The best r	497	[-0.010994505137205124, -0.01824479177594185, 0.01306507084518671, -0						
1.28E+10	-0.00199	-0.00494	-0.0031	-0.03501	-0.01141	0.032759	0.0021	0.000754	0.001029
4	ThatÃ¢Â€	495	[-5.619485455099493e-05, -0.02316370978951454, 0.010266464203596115,						
0.007418	-0.01983	0.002217	-0.01595	-0.0136	-0.01744	0.026616	0.005573	0.017235	-0.01481
5	So letÃ¢Â	500	[0.0006296807550825179, 0.0020809012930840254, 0.007517880294471979,						
3.73E+13	-0.01903	0.014996	-0.00817	0.010488	-0.01484	0.021788	-0.00457	0.018285	0.006805
6	Here are a	464	[-0.007965067401528358, 0.0060860770754516125, 0.009820940904319286,						
5.94E+08	0.013605	-0.02242	-0.01787	-0.00567	0.018215	0.000982	0.018559	0.016221	-0.01814
7	For examp	462	[0.006393357180058956, -0.0031651596073061228, 0.014186816290020943,						
9821644	-0.00016	-0.00801	0.028878	-0.02334	-0.01202	-0.02464	0.011526	0.000225	-0.00949
8	More spec	444	[0.02807449921965599, 0.013232897035777569, -0.0019443926867097616, -						
51873	-6.59E-05	-0.00303	-0.01349	0.024118	-0.02545	-0.00505	-0.02247	0.033368	0.013544
9	The weak	478	[0.003925706725567579, 0.01203661598265171, 0.010645984672009945, -0.						
0.005323	-0.03125	0.012715	-0.02294	-0.01819	-0.01804	0.031219	-0.01023	0.024552	0.012356
10	xliv The h	453	[-0.0003294514608569443, -0.020283186808228493, 0.02068403922021389,						
53	0.020831	-0.01265	-0.0072	-0.00903	0.033404	0.006791	0.008491	-0.01372	-0.02357

- The generated embeddings

Building a question answer system

- The embeddings are ready and now we can create a simple question and answer system.
- This will take a user's question, create an embedding of it, and compare it with the existing embeddings to retrieve the most relevant text from the scraped website.
- The text-davinci-003 model will then generate a natural sounding answer based on the retrieved text.

Building a question answer system

- First, we will convert the embeddings into a NumPy array
- This will provide more flexibility in how to use the embeddings because many functions available that operate on NumPy arrays.
- The conversion will also flatten the dimension to 1-D, which is the required format for many subsequent operations.

Building a question answer system

```python
import openai
import pandas as pd
import numpy as np
from openai.embeddings_utils import distances_from_embeddings, cosine_similarity

df=pd.read_csv('/content/drive/MyDrive/thesis/processed/embeddings.csv', index_col=0)
df['embeddings'] = df['embeddings'].apply(eval).apply(np.array)

df.head()
```

	text	n_tokens	embeddings
0	nciples.org Why and How Capitalism Needs To Be...	495	[0.005654186941683292, -0.009135925211012363, ...
1	I believe that all good things taken to an ext...	479	[-0.003091058460995555, -0.005778795108199 12, ...
2	Because I loved playing the markets I chose to...	486	[-0.016110556200146675, -0.026599613949656487,...
3	The best results come when there is more rathe...	497	[-0.010994505137205124, -0.018244791775941 85, ...
4	Thatâ□□s for the population as a whole. For mo...	495	[-5.619485455099493e-05, -0.023163709789514 54,...

Building a question answer system

- Since the data for answering is ready, the task left is to convert questions to an embedding.
- This is important because the search with embeddings compares the vector of numbers (which was the conversion of the raw text) using cosine distance.
- The vectors are likely related and might be the answer to the question if they are close in cosine distance.
- The OpenAI python package has a built in distances_from_embeddings function that can be used here.

Building a question answer system

```python
def create_context(
    question, df, max_len=1800, size="ada"
):
    """
    Create a context for a question by finding the most similar context from the dataframe
    """

    # Get the embeddings for the question
    q_embeddings = openai.Embedding.create(input=question, engine='text-embedding-ada-002')['data'][0][

    # Get the distances from the embeddings
    df['distances'] = distances_from_embeddings(q_embeddings, df['embeddings'].values, distance_metric=


    returns = []
    cur_len = 0

```

Building a question answer system

```python

    # Sort by distance and add the text to the context until the context is too long
    for i, row in df.sort_values('distances', ascending=True).iterrows():

        # Add the length of the text to the current length
        cur_len += row['n_tokens'] + 4

        # If the context is too long, break
        if cur_len > max_len:
            break

        # Else add it to the text that is being returned
        returns.append(row["text"])

    # Return the context
    return "\n\n###\n\n".join(returns)
```

Building a question answer system

- In the above create_context method. The text was broken up into smaller sets of tokens, so looping through in ascending order and continuing to add the text is a critical step to ensure a full answer.
- The max_len parameter can also be modified to something smaller, if more content than desired is returned.

Building a question answer system

- The above method only retrieved chunks of texts that are semantically related to the question, so they might contain the answer, but there's no guarantee of it.
- The answering prompt will then try to extract the relevant facts from the retrieved contexts, in order to formulate a coherent answer.
- If there is no relevant answer, the prompt will return “I don’t know”.
- A realistic sounding answer to the question can be created with the completion endpoint using text-davinci-003.

Building a question answer system

```python
def answer_question(
    df,
    model="text-davinci-003",
    question="Am I allowed to publish model outputs to Twitter, without a human review?",
    max_len=1800,
    size="ada",
    debug=False,
    max_tokens=150,
    stop_sequence=None
):
    """
    Answer a question based on the most similar context from the dataframe texts
    """
    context = create_context(
        question,
        df,
        max_len=max_len,
        size=size,
    )
```

Building a question answer system

```
    # If debug, print the raw model response
    if debug:
        print("Context:\n" + context)
        print("\n\n")

    try:
        # Create a completions using the question and context
        response = openai.Completion.create(
            prompt=f"Answer the question based on the context below,
            and if the question can't be answered based on the context,
             say \"I don't know\"\n\nContext: {context}\n\n---\n\nQuestion: {question}\nAnswer:",
            temperature=0,
            max_tokens=max_tokens,
            top_p=1,
            frequency_penalty=0,
            presence_penalty=0,
            stop=stop_sequence,
            model=model,
        )
        return response["choices"][0]["text"].strip()
    except Exception as e:
        print(e)
        return ""
```

Building a question answer system

- The building of a working Q/A system is DONE! It has the knowledge embedded from the OpenAI website
- We can perform several questions to see the quality of the output:

Building a question answer system

```
[ ] answer_question(df, question="What day is it?", debug=False)

    'I don't know.'

    answer_question(df, question="how economic machine works")

    'The economy works by connecting pay to productivity and creating efficient capital markets that facilitate savings and the availability of buying power
    to fuel people's productivity. Poor education, a poor culture, poor infrastructure, and too much debt cause bad economic results. Money is clogged at the
    top because if you have money or good ideas of how to make money you can have more money than you need because lenders will freely lend it to you and inv
    estors will compete to give it to you. Policy makers pay too much attention to budgets relative to returns on investments. Capitalism is a fundamentally
    sound system that is now not working well for the majority of people, so it must be reformed to provide many more equal opportunities and to be more prod
    uctive.'

[ ] answer_question(df, question="why we need economy?")

    'We need an economy to provide basic care, good education, and employment opportunities to people, and to create a system that rewards people for their p
    roductive activities with money that can be used to get all that money can buy. Capitalism is an effective motivator of people and allocator of resources
    that raises people's living standards.'
```

Building a question answer system

```
[ ] answer_question(df, question="how economy knowledge is applied in personal finance?")
```

'Economy knowledge can be applied in personal finance by understanding how different economic systems work and how they can affect personal finances. Thi
s includes understanding how the Federal Reserve and other central banks buy financial assets to put money in the economy in order to stimulate the econo
my, how taxation works, and how to coordinate monetary and fiscal policies. Additionally, understanding the cause and effect relationships and historical
comparisons of different economic systems can help inform personal finance decisions.'

```
[ ] answer_question(df, question="can you give more specific answer?")
```

'I don't know.'

```
[ ] answer_question(df, question="why should i do in economic crisis?")
```

'I don't know.'

```
[ ] answer_question(df, question="what are the richest country nowaday?")
```

'I don't know.'

```
[ ] answer_question(df, question="why currency is strongest?")
```

'I don't know.'

Building a question answer system

```
[12] answer_question(df, question="what currency is strongest?")

'I don't know.'

[13] answer_question(df, question="which country can have more benefit in enocomic crisis?")

'I don't know.'

[14] answer_question(df, question="what activities bring the most finalcial profit?")

'It depends on the activity, but some examples of activities that can bring a high financial return include early childhood education programs, intervent
ions that lead to lower high school dropout rates, school finance reforms, microfinance, infrastructure spending plans, and public health/preventative he
althcare interventions.'

answer_question(df, question="how education make effect on economy of a country?")

'Poor education, a poor culture (one that impedes people from operating effectively together), poor infrastructure, and too much debt cause bad economic
results. The health consequences and economic costs of low education and poverty are terrible, and can lead to an increase in premature deaths, an increa
se in people putting off medical treatment due to cost, and worse health for those who are unemployed or making less than $35,000 per year. These conditi
ons pose an existential risk for the US, as they weaken the US economically, threaten to bring about painful and counterproductive domestic conflict, and
undermine the United States' strength relative to that of its global competitors.'
```

Evaluation and Discussion on result

Evaluation and Discussion on result

- The answers is limited within the range of knowledge provided from the crawled domain only.
- The larger amount of input information, the more informative answer is.
- The answer are not really closely related to the questions.
- The running time for returning an answer varies depending on the content of the question

Potential problems and bugs in running the code

Potential problems and bugs in running the code

- FileNotFoundError: [Errno 2] No such file or directory: '/content/drive/MyDrive/thesis/text/economicprinciples.orgr/‘
- The problem was the additional character “r” that make the machine unable to find the correct path.
- So we can simply remove the “r” character in the domain and the code will work well

```
# Define root domain to crawl
domain = "economicprinciples.orgr"
full_url = "https://economicprinciples.org/"
```

Potential problems and bugs in running the code

- Regenerating the embedding with the OpenAI api key will get error
  - `RateLimitError: You exceeded your current quota, please check your plan and billing details.`
- This is because I use free plan to generate the embedding, so I can only generate several times
- So it’s better to reuse the embedding.

Potential problems and bugs in running the code

```
import openai

df['embeddings'] = df.text.apply(lambda x: openai.Embedding.create(input=x, engine='text-embedding-ada-002')['data'][0]['embedding'])
df.to_csv('/content/drive/MyDrive/thesisprocessed/embeddings.csv')
df.head()
```

```
---------------------------------------------------------------------------
RateLimitError                            Traceback (most recent call last)
<ipython-input-20-7398352f7784> in <cell line: 3>()
      1 import openai
      2
----> 3 df['embeddings'] = df.text.apply(lambda x: openai.Embedding.create(input=x, engine='text-embedding-ada-002')['data'][0]['embedding'])
      4 df.to_csv('/content/drive/MyDrive/thesisprocessed/embeddings.csv')
      5 df.head()

9 frames

/usr/local/lib/python3.9/dist-packages/openai/api_requestor.py in _interpret_response_line(self, rbody, rcode, rheaders, stream)
    681         stream_error = stream and "error" in resp.data
    682         if stream_error or not 200 <= rcode < 300:
--> 683             raise self.handle_error_response(
    684                 rbody, rcode, resp.data, rheaders, stream_error=stream_error
    685             )

RateLimitError: You exceeded your current quota, please check your plan and billing details.
```

SEARCH STACK OVERFLOW

Suggesting order for executing the code

Suggesting order for executing the code

- The reason for this order is that:
    - The text file is already scrapped from domain "economicprinciples.org"
    - The text file is already converted to embeddings via OpenAI api
    - Therefore, we don't need to re-implement these code section.
- And we can ask question after following this suggesting order

Suggesting order for executing the code

- 1/ Mount to Google drive:
  - This allows us to access to the generated embeddings.

```
[1] from google.colab import drive
    drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

Suggesting order for executing the code

- 2/ Install openai package
- We need install this because we will need it to access api key and embeddings

```
[3]  !pip install openai

Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Collecting openai
  Downloading openai-0.27.6-py3-none-any.whl (71 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 71.9/71.9 kB 2.8 MB/s eta 0:00:00
Requirement already satisfied: requests>=2.20 in /usr/local/lib/python3.10/dist-packages (from openai) (2.27.1)
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-packages (from openai) (4.65.0)
Collecting aiohttp (from openai)
  Downloading aiohttp-3.8.4-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (1.0 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.0/1.0 MB 18.5 MB/s eta 0:00:00
Requirement already satisfied: urllib3<1.27,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests>=2.20->openai) (1.26.15)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests>=2.20->openai) (2022.12.7)
Requirement already satisfied: charset-normalizer~=2.0.0 in /usr/local/lib/python3.10/dist-packages (from requests>=2.20->openai) (2.0.12)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests>=2.20->openai) (3.4)
Requirement already satisfied: attrs>=17.3.0 in /usr/local/lib/python3.10/dist-packages (from aiohttp->openai) (23.1.0)
Collecting multidict<7.0,>=4.5 (from aiohttp->openai)
  Downloading multidict-6.0.4-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (114 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 114.5/114.5 kB 11.4 MB/s eta 0:00:00
Collecting async-timeout<5.0,>=4.0.0a3 (from aiohttp->openai)
  Downloading async_timeout-4.0.2-py3-none-any.whl (5.8 kB)
Collecting yarl<2.0,>=1.0 (from aiohttp->openai)
  Downloading yarl-1.9.2-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (268 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 268.8/268.8 kB 19.6 MB/s eta 0:00:00
Collecting frozenlist>=1.1.1 (from aiohttp->openai)
  Downloading frozenlist-1.3.3-cp310-cp310-manylinux_2_5_x86_64.manylinux1_x86_64.manylinux_2_17_x86_64.manylinux2014_x86_64.whl (149 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 149.6/149.6 kB 13.4 MB/s eta 0:00:00
```

Suggesting order for executing the code

- 3/ Share openai api key
- This allow us to directly access to the api and convert the question into embedding
- The question embedding will then be compared with answer embedding to sort out an appropriate answer

```
[4] import openai
    openai.api_key = "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"
```

Suggesting order for executing the code

- 4/ Convert the answering embedding into numpy array
- We do this to utilize the existing function inside numpy library.

```
[9] import openai
    import pandas as pd
    import numpy as np
    from openai.embeddings_utils import distances_from_embeddings, cosine_similarity

    df=pd.read_csv('/content/drive/MyDrive/thesis/processed/embeddings.csv', index_col=0)
    df['embeddings'] = df['embeddings'].apply(eval).apply(np.array)

    df.head()
```

	text	n_tokens	embeddings
0	nciples.org Why and How Capitalism Needs To Be...	495	[0.005654186941683292, -0.009135925211012363, ...
1	I believe that all good things taken to an ext...	479	[-0.003091058460995555, -0.00577879510819912, ...
2	Because I loved playing the markets I chose to...	486	[-0.016110556200146675, -0.026599613949656487,...
3	The best results come when there is more rathe...	497	[-0.010994505137205124, -0.01824479177594185, ...
4	Thatâ□□s for the population as a whole. For mo...	495	[-5.619485455099493e-05, -0.02316370978951454,...

Suggesting order for executing the code

- 5/ Implement create_context function
- We need to run this to retrieve the embedding for question and extract the context for question inside the answer embeddings

```python
def create_context(
    question, df, max_len=1800, size="ada"
):
    """
    Create a context for a question by finding the most similar context from the dataframe
    """

    # Get the embeddings for the question
    q_embeddings = openai.Embedding.create(input=question, engine='text-embedding-ada-002')['data'][0][

    # Get the distances from the embeddings
    df['distances'] = distances_from_embeddings(q_embeddings, df['embeddings'].values, distance_metric=


    returns = []
    cur_len = 0

```

```python

    # Sort by distance and add the text to the context until the context is too long
    for i, row in df.sort_values('distances', ascending=True).iterrows():

        # Add the length of the text to the current length
        cur_len += row['n_tokens'] + 4

        # If the context is too long, break
        if cur_len > max_len:
            break

        # Else add it to the text that is being returned
        returns.append(row["text"])

    # Return the context
    return "\n\n###\n\n".join(returns)
```

Suggesting order for executing the code

- 6/ Implement answer_questionfunction
- We need to run this sort out the final answer from the context

```python
def answer_question(
    df,
    model="text-davinci-003",
    question="Am I allowed to publish model outputs to Twitter, without a human review?",
    max_len=1800,
    size="ada",
    debug=False,
    max_tokens=150,
    stop_sequence=None
):
    """
    Answer a question based on the most similar context from the dataframe texts
    """
    context = create_context(
        question,
        df,
        max_len=max_len,
        size=size,
    )
```

```python
    # If debug, print the raw model response
    if debug:
        print("Context:\n" + context)
        print("\n\n")

    try:
        # Create a completions using the question and context
        response = openai.Completion.create(
            prompt=f"Answer the question based on the context below,
            and if the question can't be answered based on the context,
            say \"I don't know\"\n\nContext: {context}\n\n---\n\nQuestion: {question}\nAnswer:",
            temperature=0,
            max_tokens=max_tokens,
            top_p=1,
            frequency_penalty=0,
            presence_penalty=0,
            stop=stop_sequence,
            model=model,
        )
        return response["choices"][0]["text"].strip()
    except Exception as e:
        print(e)
        return ""
```

Suggesting order for executing the code

- 7/ We are done!
- Now just input the question and ask

Now the process is done! we're ready to ask and receive answer

```
[10] answer_question(df, question="What day is it?", debug=False)

'I don't know.'
```

Preference and source code

- Web Q&A - OpenAI API
- Source code in colab:
- chatbot=1st_trials_)_edited.ipynb - Colaboratory (google.com)


THANK YOU