

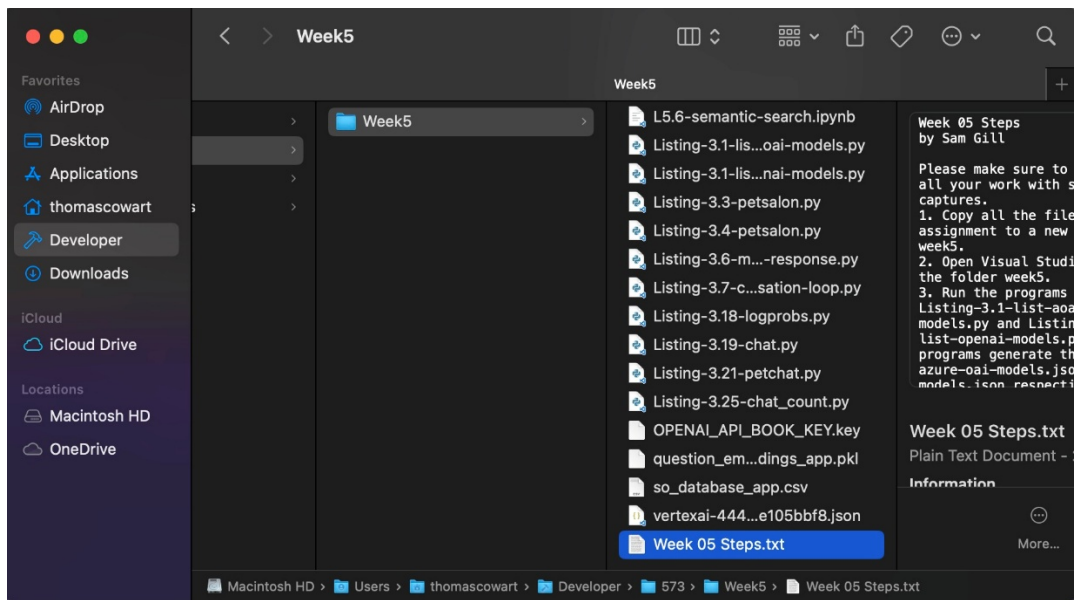
Thomas Cowart

Prof. Gill

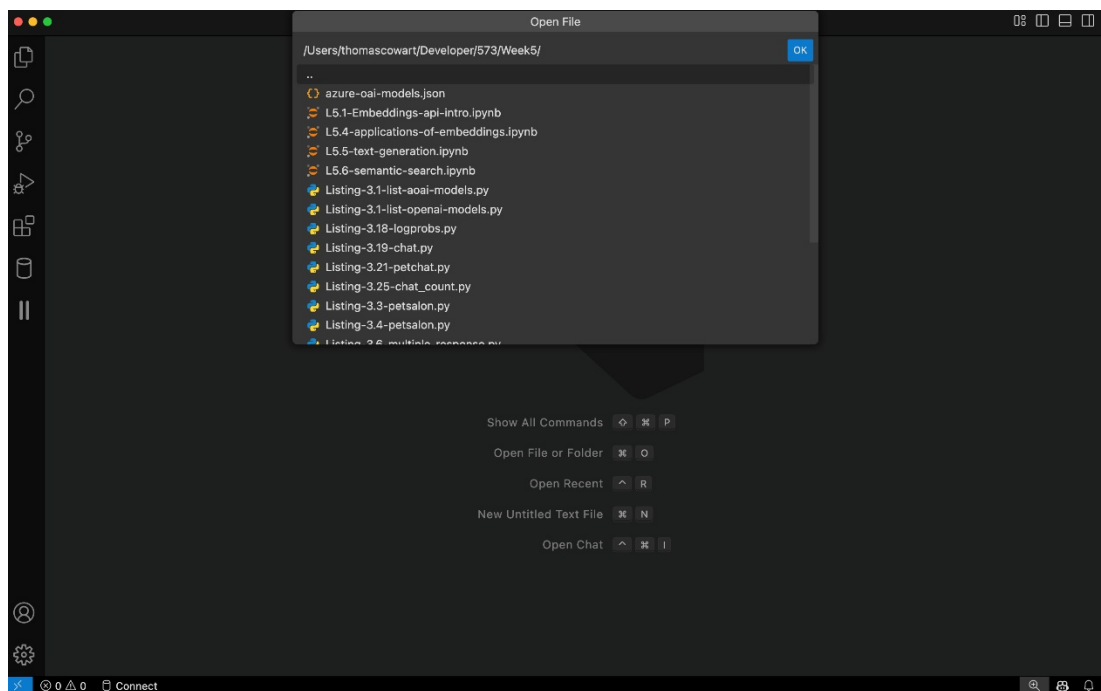
ISYS 573

Week 05 HW

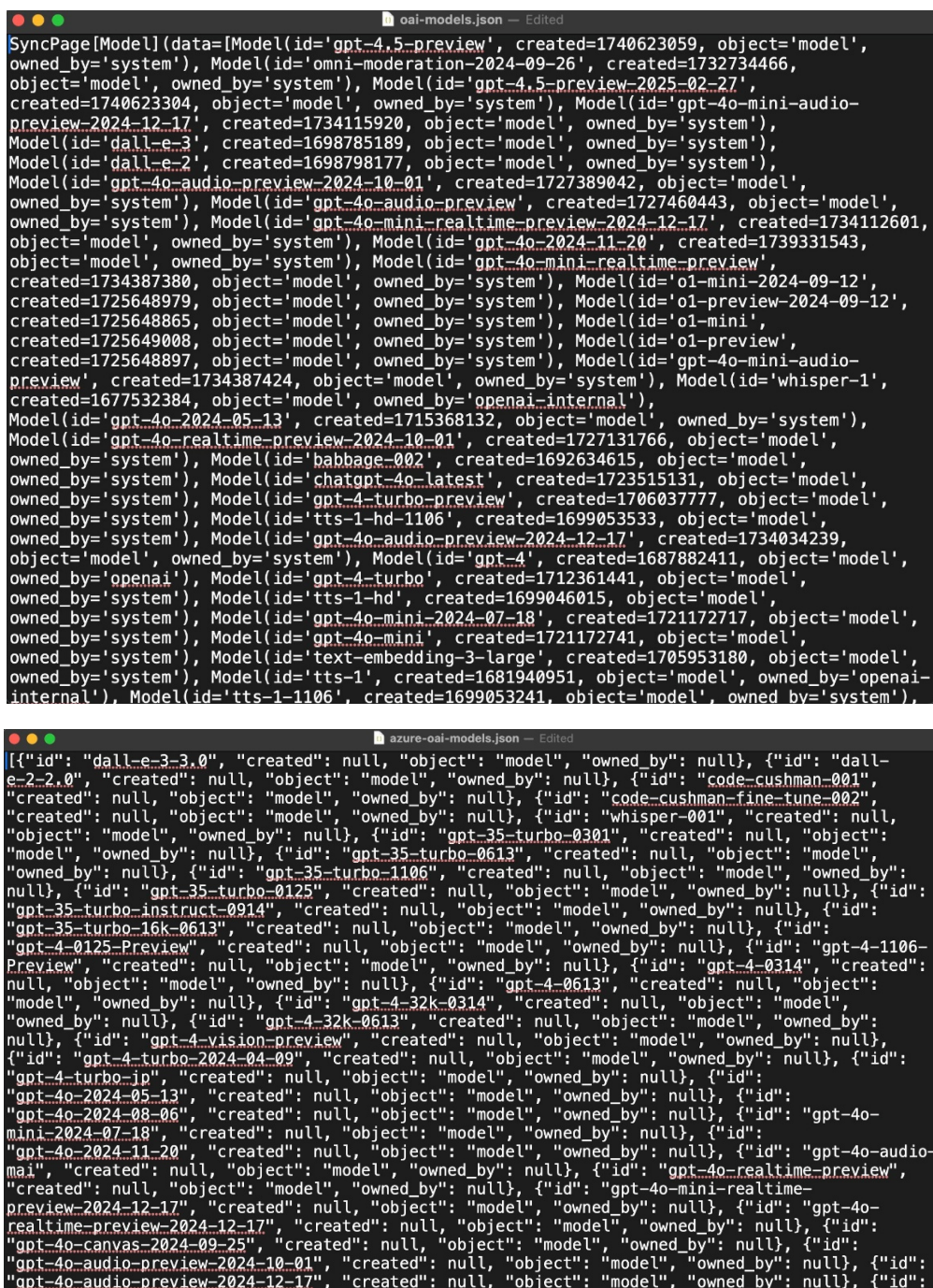
1)



2)



3)



```
oai-models.json - Edited
SyncPage[Model](data=[Model(id='gpt-4.5-preview', created=1740623059, object='model',
owned_by='system'), Model(id='omni-moderation-2024-09-26', created=1732734466,
object='model', owned_by='system'), Model(id='gpt-4.5-preview-2025-02-27',
created=1740623304, object='model', owned_by='system'), Model(id='gpt-4o-mini-audio-
preview-2024-12-17', created=1734115920, object='model', owned_by='system'),
Model(id='dall-e-3', created=1698785189, object='model', owned_by='system'),
Model(id='dall-e-2', created=1698798177, object='model', owned_by='system'),
Model(id='gpt-4o-audio-preview-2024-10-01', created=1727389042, object='model',
owned_by='system'), Model(id='gpt-4o-audio-preview', created=1727460443, object='model',
owned_by='system'), Model(id='gpt-4o-mini-realtime-preview-2024-12-17', created=1734112601,
object='model', owned_by='system'), Model(id='gpt-4o-2024-11-20', created=1739331543,
object='model', owned_by='system'), Model(id='gpt-4o-mini-realtime-preview',
created=1734387380, object='model', owned_by='system'), Model(id='o1-mini-2024-09-12',
created=1725648979, object='model', owned_by='system'), Model(id='o1-preview-2024-09-12',
created=1725648865, object='model', owned_by='system'), Model(id='o1-mini',
created=1725649008, object='model', owned_by='system'), Model(id='o1-preview',
created=1725648897, object='model', owned_by='system'), Model(id='gpt-4o-mini-audio-
preview', created=1734387424, object='model', owned_by='system'), Model(id='whisper-1',
created=1677532384, object='model', owned_by='openai-internal'),
Model(id='gpt-4o-2024-05-13', created=1715368132, object='model', owned_by='system'),
Model(id='gpt-4o-realtime-preview-2024-10-01', created=1727131766, object='model',
owned_by='system'), Model(id='babbage-002', created=1692634615, object='model',
owned_by='system'), Model(id='chatgpt-4o-latest', created=1723515131, object='model',
owned_by='system'), Model(id='gpt-4-turbo-preview', created=1706037777, object='model',
owned_by='system'), Model(id='tts-1-hd-1106', created=1699053533, object='model',
owned_by='system'), Model(id='gpt-4o-audio-preview-2024-12-17', created=1734034239,
object='model', owned_by='system'), Model(id='gpt-4', created=1687882411, object='model',
owned_by='openai'), Model(id='gpt-4-turbo', created=1712361441, object='model',
owned_by='system'), Model(id='tts-1-hd', created=1699046015, object='model',
owned_by='system'), Model(id='gpt-4o-mini-2024-07-18', created=1721172717, object='model',
owned_by='system'), Model(id='gpt-4o-mini', created=1721172741, object='model',
owned_by='system'), Model(id='text-embedding-3-large', created=1705953180, object='model',
owned_by='system'), Model(id='tts-1', created=1681940951, object='model', owned_by='openai-
internal'), Model(id='tts-1-1106', created=1699053241, object='model', owned_by='system'),

azure-oai-models.json - Edited
[{"id": "dall-e-3-3.0", "created": null, "object": "model", "owned_by": null}, {"id": "dall-
e-2.0", "created": null, "object": "model", "owned_by": null}, {"id": "code-cushman-001",
"created": null, "object": "model", "owned_by": null}, {"id": "code-cushman-fine-tune-002",
"created": null, "object": "model", "owned_by": null}, {"id": "whisper-001", "created": null,
"object": "model", "owned_by": null}, {"id": "gpt-35-turbo-0301", "created": null, "object":
"model", "owned_by": null}, {"id": "gpt-35-turbo-0613", "created": null, "object": "model",
"owned_by": null}, {"id": "gpt-35-turbo-1106", "created": null, "object": "model", "owned_by":
null}, {"id": "gpt-35-turbo-0125", "created": null, "object": "model", "owned_by": null}, {"id":
"gpt-35-turbo-instruct-0914", "created": null, "object": "model", "owned_by": null}, {"id":
"gpt-35-turbo-16k-0613", "created": null, "object": "model", "owned_by": null}, {"id":
"gpt-4-0125-Preview", "created": null, "object": "model", "owned_by": null}, {"id": "gpt-4-1106-
Preview", "created": null, "object": "model", "owned_by": null}, {"id": "gpt-4-0314", "created":
null, "object": "model", "owned_by": null}, {"id": "gpt-4-0613", "created": null, "object":
"model", "owned_by": null}, {"id": "gpt-4-32k-0314", "created": null, "object": "model",
"owned_by": null}, {"id": "gpt-4-32k-0613", "created": null, "object": "model", "owned_by":
null}, {"id": "gpt-4-vision-preview", "created": null, "object": "model", "owned_by": null}, {"id":
"gpt-4-turbo-2024-04-09", "created": null, "object": "model", "owned_by": null}, {"id":
"gpt-4-turbo-16k", "created": null, "object": "model", "owned_by": null}, {"id":
"gpt-4o-2024-05-13", "created": null, "object": "model", "owned_by": null}, {"id":
"gpt-4o-2024-08-06", "created": null, "object": "model", "owned_by": null}, {"id": "gpt-4o-
mini-2024-07-18", "created": null, "object": "model", "owned_by": null}, {"id":
"gpt-4o-2024-11-20", "created": null, "object": "model", "owned_by": null}, {"id": "gpt-4o-audio-
mai", "created": null, "object": "model", "owned_by": null}, {"id": "gpt-4o-realtime-preview",
"created": null, "object": "model", "owned_by": null}, {"id": "gpt-4o-mini-realtime-
preview-2024-12-17", "created": null, "object": "model", "owned_by": null}, {"id": "gpt-4o-
realtime-preview-2024-12-17", "created": null, "object": "model", "owned_by": null}, {"id":
"gpt-4o-canvas-2024-09-25", "created": null, "object": "model", "owned_by": null}, {"id":
"gpt-4o-audio-preview-2024-10-01", "created": null, "object": "model", "owned_by": null}, {"id":
"gpt-4o-audio-preview-2024-12-17", "created": null, "object": "model", "owned_by": null}]
```

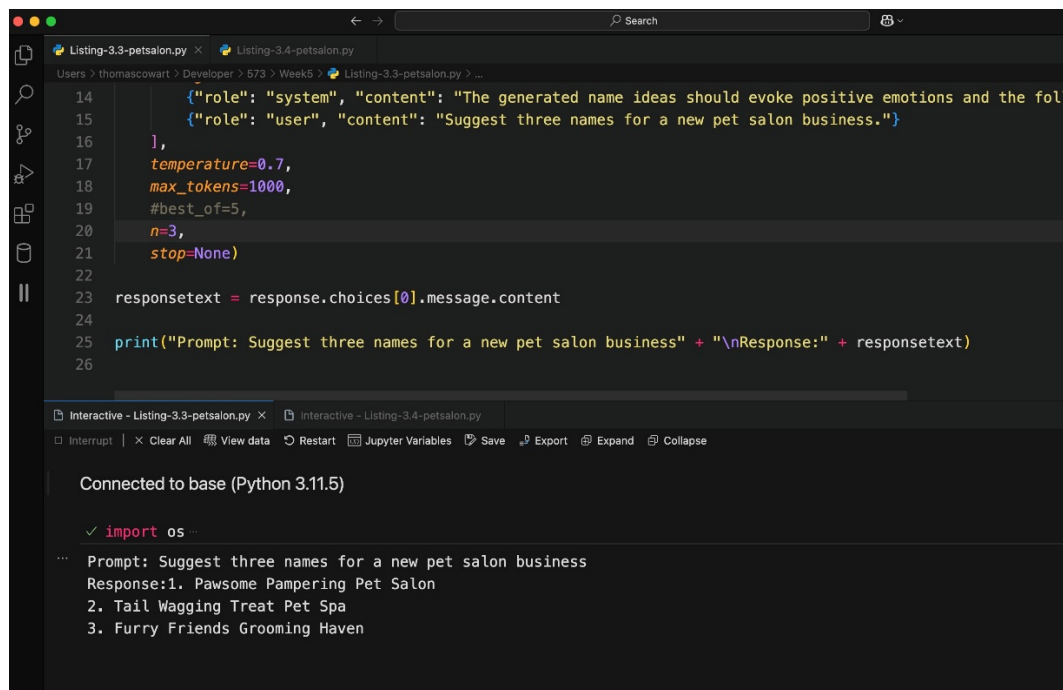
The two JSON files list available models from different sources: Azure OpenAI (azure-oai-models.json) and OpenAI's own API (oai-models.json).

Azure's list includes older models like davinci, curie, and babbage, alongside gpt-4o and dall-e-3, but lacks explicit ownership details. OpenAI's list is more up-to-date, featuring

gpt-4.5-preview, real-time variations, and models labeled with specific dates. It also includes ownership tags (system, openai, openai-internal) and models for moderation (omni-moderation) and text-to-speech (tts-1-hd).

Azure seems to have a more stable, enterprise-focused set of models, while OpenAI's API moves faster, testing and releasing updates more frequently.

4)



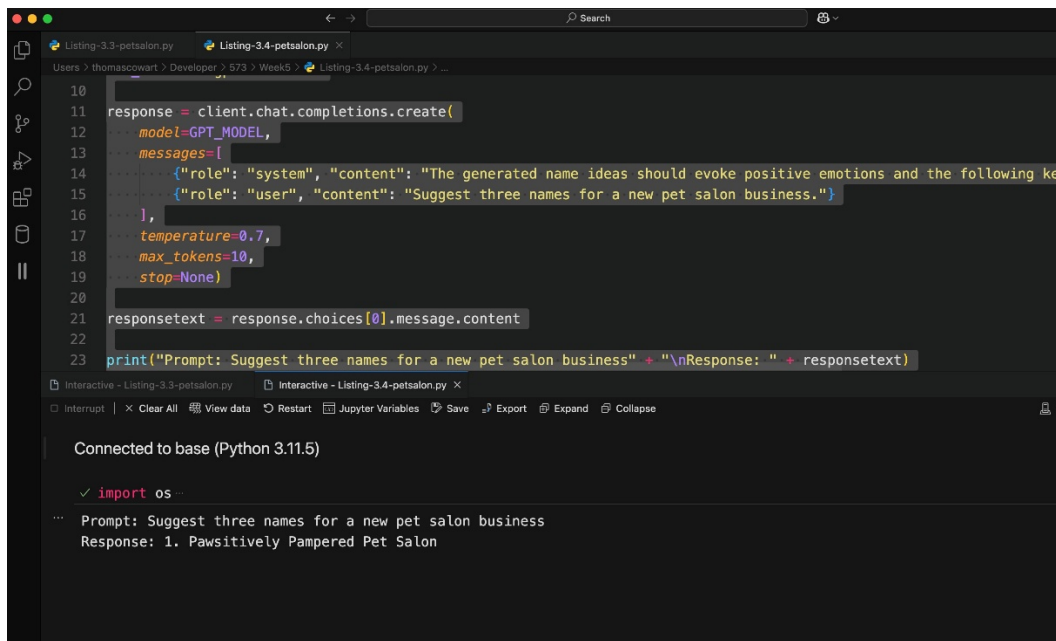
```
14 [{"role": "system", "content": "The generated name ideas should evoke positive emotions and the foll
15 [{"role": "user", "content": "Suggest three names for a new pet salon business."}]
16 ],
17 temperature=0.7,
18 max_tokens=1000,
19 #best_of=5,
20 n=3,
21 stop=None)
22
23 responsetext = response.choices[0].message.content
24
25 print("Prompt: Suggest three names for a new pet salon business" + "\nResponse:" + responsetext)
26
```

Connected to base (Python 3.11.5)

```
✓ import os ...
```

... Prompt: Suggest three names for a new pet salon business
Response:1. Pawsome Pampering Pet Salon
2. Tail Wagging Treat Pet Spa
3. Furry Friends Grooming Haven

As we can see in the screenshot above, the max number of tokens is set to 1000, which gives ample room for the model to successfully complete its assigned task.



```
10
11 response = client.chat.completions.create(
12     model=GPT_MODEL,
13     messages=[
14         {"role": "system", "content": "The generated name ideas should evoke positive emotions and the following key words"},
15         {"role": "user", "content": "Suggest three names for a new pet salon business."}
16     ],
17     temperature=0.7,
18     max_tokens=10,
19     stop=None)
20
21 responsetext = response.choices[0].message.content
22
23 print("Prompt: Suggest three names for a new pet salon business" + "\nResponse: " + responsetext)
```

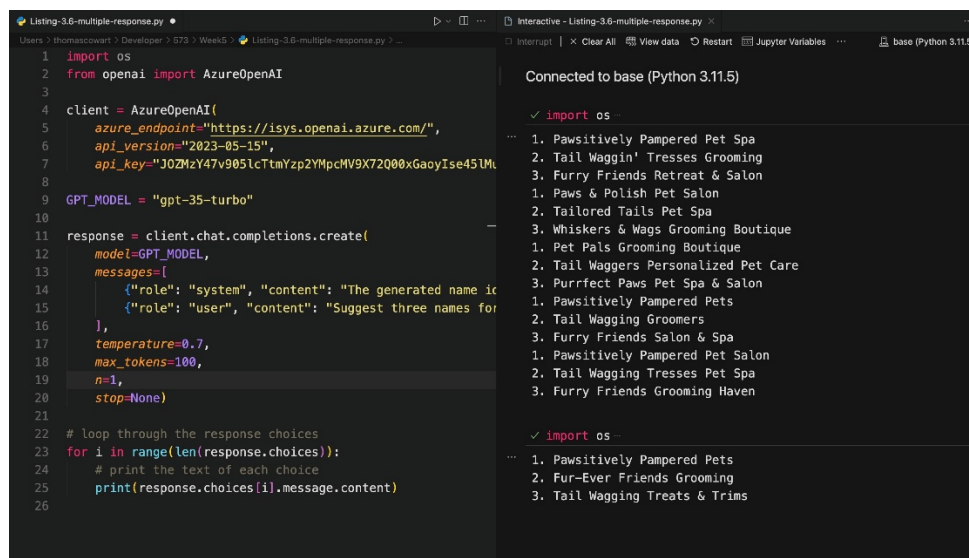
Connected to base (Python 3.11.5)

```
✓ import os
```

Prompt: Suggest three names for a new pet salon business
Response: 1. Pawsitively Pampered Pet Salon

As we can see in the screenshot above, the max number of tokens is set to only 10, which does not give the model enough room to successfully complete its assigned task. Realistically for this task, I would imagine that we would want to set the max number of tokens to a value in-between 10 and 1000 for efficiency sake, while still successfully producing the desired output.

5)



```
1 import os
2 from openai import AzureOpenAI
3
4 client = AzureOpenAI(
5     azure_endpoint="https://isys.openai.azure.com/",
6     api_version="2023-05-15",
7     api_key="J0ZMzY47v9051cTmYzp2YMpcMV9X72Q00xGaoyTse451Wt"
8 )
9
10 GPT_MODEL = "gpt-35-turbo"
11
12 response = client.chat.completions.create(
13     model=GPT_MODEL,
14     messages=[
15         {"role": "system", "content": "The generated name ideas should evoke positive emotions and the following key words"},
16         {"role": "user", "content": "Suggest three names for a new pet salon business."}
17     ],
18     temperature=0.7,
19     max_tokens=100,
20     n=1,
21     stop=None)
22
23 # loop through the response choices
24 for i in range(len(response.choices)):
25     # print the text of each choice
26     print(response.choices[i].message.content)
```

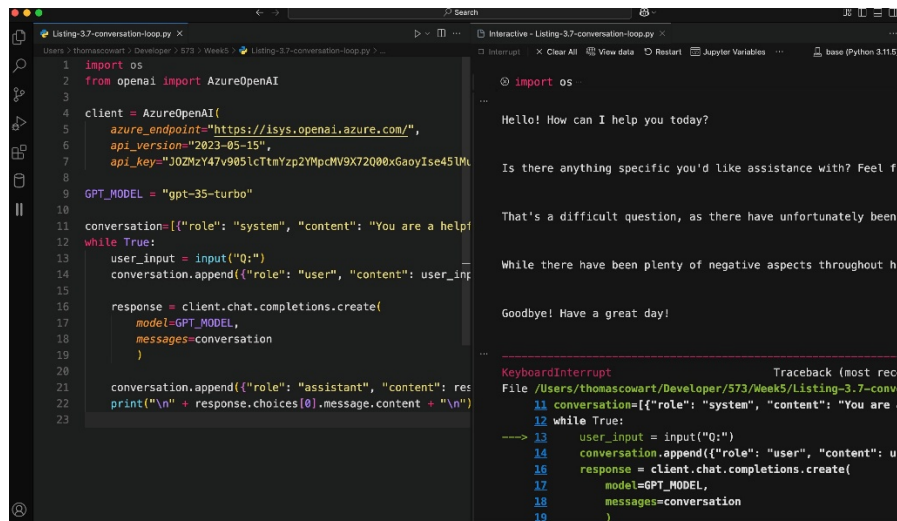
Connected to base (Python 3.11.5)

```
✓ import os
```

1. Pawsitively Pampered Pet Spa
2. Tail Waggin' Tresses Grooming
3. Furry Friends Retreat & Salon

As we can see in the screenshot above, the variable “n” represents the number of times that the prompt is ran. By changing the variable n, we will receive that number or repeated prompts until the maximum number of tokens has been reached.

6)



```
1 import os
2 from openai import AzureOpenAI
3
4 client = AzureOpenAI(
5     azure_endpoint="https://isys.openai.azure.com/",
6     api_version="2023-05-15",
7     api_key="30ZMzY47v9051cTtmYzp2YmPcMV9X72Q00xGaoyIse451Mk"
8 )
9
10 GPT_MODEL = "gpt-35-turbo"
11
12 conversation=[{"role": "system", "content": "You are a helpful assistant."}]
13 while True:
14     user_input = input("0:")
15     conversation.append({"role": "user", "content": user_input})
16
17     response = client.chat.completions.create(
18         model=GPT_MODEL,
19         messages=conversation
20     )
21
22     conversation.append({"role": "assistant", "content": response.choices[0].message.content})
23     print("\n" + response.choices[0].message.content + "\n")
```

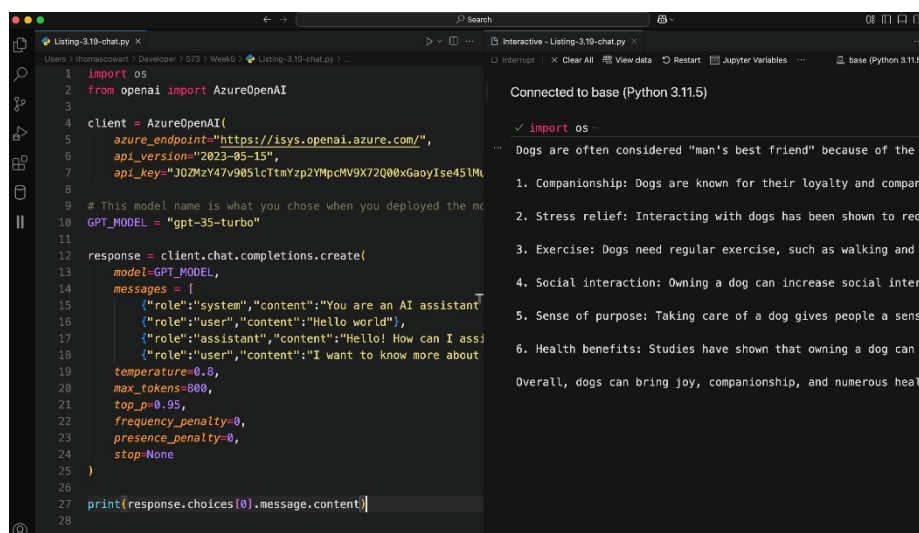
Interactive - Listing-3.7-conversation-loop.py

```
... import os
...
Hello! How can I help you today?
...
Is there anything specific you'd like assistance with? Feel free to ask.
...
That's a difficult question, as there have unfortunately been
...
While there have been plenty of negative aspects throughout history,
...
Goodbye! Have a great day!
```

KeyboardInterrupt Traceback (most recent call last):
File /Users/thomascowart/Developer/573/Week5/Listing-3.7-conversation-loop.py, line 11, in <module>
11 conversation=[{"role": "system", "content": "You are a helpful assistant."}]
12 while True:
----> 13 user_input = input("0:")
14 conversation.append({"role": "user", "content": user_input})
15 response = client.chat.completions.create(
16 model=GPT_MODEL,
17 messages=conversation
18)
19

I had a few back-and-forth conversations with the model. I had to interrupt the script to end the session. After several attempts to try to get the model to hallucinate/innovate, my attempts came to no avail.

7)



```
1 import os
2 from openai import AzureOpenAI
3
4 client = AzureOpenAI(
5     azure_endpoint="https://isys.openai.azure.com/",
6     api_version="2023-05-15",
7     api_key="30ZMzY47v9051cTtmYzp2YmPcMV9X72Q00xGaoyIse451Mk"
8 )
9
10 # This model name is what you chose when you deployed the model
11 GPT_MODEL = "gpt-35-turbo"
12
13 response = client.chat.completions.create(
14     model=GPT_MODEL,
15     messages=[
16         {"role": "system", "content": "You are an AI assistant."},
17         {"role": "user", "content": "Hello world!"},
18         {"role": "assistant", "content": "Hello! How can I assist you?"},
19         {"role": "user", "content": "I want to know more about why dogs are considered 'man's best friend'."}
20     ],
21     temperature=0.8,
22     max_tokens=800,
23     top_p=0.95,
24     frequency_penalty=0,
25     presence_penalty=0,
26     stop=None
27 )
28 print(response.choices[0].message.content)
```

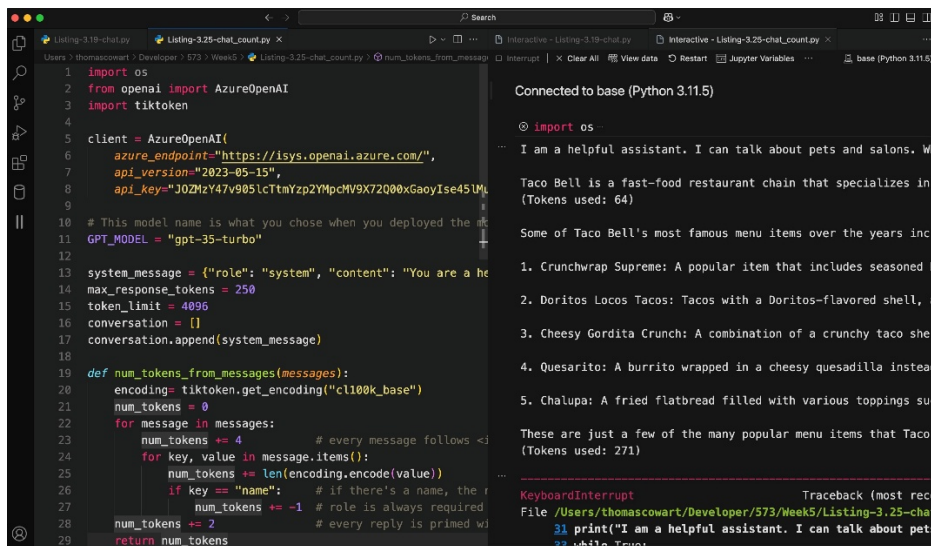
Interactive - Listing-3.10-chat.py

Connected to base (Python 3.11.5)

```
... import os
...
Dogs are often considered "man's best friend" because of the following reasons:
```

1. Companionship: Dogs are known for their loyalty and companionship.
2. Stress relief: Interacting with dogs has been shown to reduce stress and anxiety.
3. Exercise: Dogs need regular exercise, such as walking and playing.
4. Social interaction: Owning a dog can increase social interaction and provide a sense of community.
5. Sense of purpose: Taking care of a dog gives people a sense of purpose and responsibility.
6. Health benefits: Studies have shown that owning a dog can lead to lower blood pressure and improved mental health.

Overall, dogs can bring joy, companionship, and numerous health benefits to their owners.



```
1 import os
2 from openai import AzureOpenAI
3 import tiktoken
4
5 client = AzureOpenAI(
6     azure_endpoint="https://isys.openai.azure.com/",
7     api_version="2023-05-15",
8     api_key="J0ZHzY47v905LcTmYzp2YMpcW9X72Q00xGaoyIse451M..."
9 )
10 # This model name is what you chose when you deployed the model
11 GPT_MODEL = "gpt-35-turbo"
12
13 system_message = {"role": "system", "content": "You are a helpful assistant."}
14 max_response_tokens = 250
15 token_limit = 4096
16 conversation = []
17 conversation.append(system_message)
18
19 def num_tokens_from_messages(messages):
20     encoding = tiktoken.get_encoding("cl100k_base")
21     num_tokens = 0
22     for message in messages:
23         num_tokens += 4 # every message follows a pattern
24         for key, value in message.items():
25             num_tokens += len(encoding.encode(value))
26             if key == "name": # if there's a name, the role is always required
27                 num_tokens += 1
28     num_tokens += 2 # every reply is primed with a stop token
29     return num_tokens
```

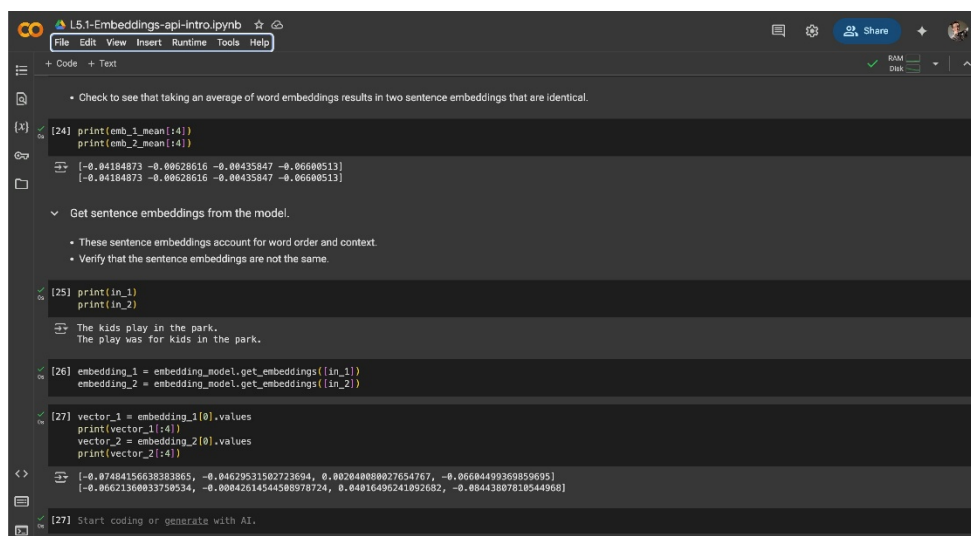
Connected to base (Python 3.11.5)

```
>>> import os
... I am a helpful assistant. I can talk about pets and salons. W
Taco Bell is a fast-food restaurant chain that specializes in
(Tokens used: 64)
Some of Taco Bell's most famous menu items over the years include:
1. Crunchwrap Supreme: A popular item that includes seasoned
2. Doritos Locos Tacos: Tacos with a Doritos-flavored shell,
3. Cheesy Gordita Crunch: A combination of a crunchy taco shell
4. Quesarito: A burrito wrapped in a cheesy quesadilla instead
5. Chalupa: A fried flatbread filled with various toppings such
These are just a few of the many popular menu items that Taco
(Tokens used: 271)
```

KeyboardInterrupt Traceback (most recent call last):
File /Users/thomascowart/Developer/573/Week5/Listing-3.25-chat-count.py:31, in print("I am a helpful assistant. I can talk about pet")
31 print("I am a helpful assistant. I can talk about pet")
KeyboardInterrupt

System defines how the AI behaves, user is me, assistant is the AI replying. Max tokens just limits how long the response can be. In 3.19.py, it's set to 800. In 3.25, it's 250, and old messages get deleted to stay under 4096 total tokens. If the response hits the max tokens limit, it'll stop there, and the finish reason will be "max_tokens." You can check token usage with response.usage.total_tokens.

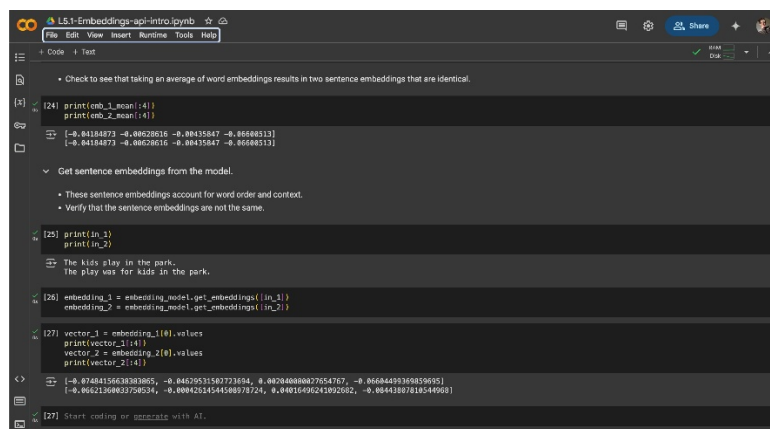
8)



```
File Edit View Insert Runtime Tools Help
+ Code + Text
• Check to see that taking an average of word embeddings results in two sentence embeddings that are identical.
[24] print(emb_1_mean[4])
print(emb_2_mean[4])
[-0.04184873 -0.00628616 -0.00435847 -0.06600513]
[-0.04184873 -0.00628616 -0.00435847 -0.06600513]
Get sentence embeddings from the model.
• These sentence embeddings account for word order and context.
• Verify that the sentence embeddings are not the same.
[25] print(in_1)
print(in_2)
The kids play in the park.
The play was for kids in the park.
[26] embedding_1 = embedding_model.get_embeddings([in_1])
embedding_2 = embedding_model.get_embeddings([in_2])
[27] vector_1 = embedding_1[0].values
print(vector_1[4])
vector_2 = embedding_2[0].values
print(vector_2[4])
[-0.07484156638383865, -0.04629531502723694, 0.002040888027654767, -0.0560449369859695]
[-0.0602136093375934, -0.00842614544380970724, 0.04016496241892602, -0.004430070185449606]
[27] Start coding or generate with AI.
```

Embeddings turn words or data into numbers so a computer can understand relationships between them. Instead of just treating words as separate things, embeddings map them into a multi-dimensional space where similar things are closer together. For example, in an embedding model, “king” and “queen” would be near each other, while “king” and “banana” would be far apart. It’s like plotting words on a graph, but with way more dimensions. This is useful for search, recommendations, or AI understanding context because it helps find patterns and similarities beyond just matching exact words.

9)



```
File Edit View Insert Runtime Tools Help
+ Code + Text

• Check to see that taking an average of word embeddings results in two sentence embeddings that are identical.

[24] print(emb_1_mean[:4])
print(emb_2_mean[:4])

[-0.04184873 -0.00628016 -0.00151847 -0.06608513]
[-0.04184873 -0.00628016 -0.00151847 -0.06608513]

• Get sentence embeddings from the model.
• These sentence embeddings account for word order and context.
• Verify that the sentence embeddings are not the same.

[25] print(in_1)
print(in_2)

The kids play in the park.
The play was for kids in the park.

[26] embedding_1 = embedding_model.get_embeddings([in_1])
embedding_2 = embedding_model.get_embeddings([in_2])

[27] vector_1 = embedding_1[0].values
print(vector_1[:4])
vector_2 = embedding_2[0].values
print(vector_2[:4])

[-0.0748415663838065 -0.04079531587723694 0.04204000007054767 -0.06604409336985905]
[-0.0662150883370534 -0.0004265454588978724 0.00010496241092002 -0.08443807010544068]
```

To use embeddings for classification, you first transform your data (like text) into embeddings, which are numerical representations of the data. Then, you can feed those embeddings into a classification model (like a neural network or a decision tree) to predict categories, since the embeddings capture the important relationships between words or data points. To avoid recalculating embeddings every time, you can store them once, and then reuse the precomputed embeddings whenever you need to classify new data. This saves time, since you don’t need to run the embedding model from scratch each time. Essentially, you preprocess your data, save the embeddings, and then only run the classifier on those pre-saved embeddings.

10)

```

You can check the predictions on a few questions from the test set

[109] y_pred = clf.predict(X_test)

[110] accuracy = accuracy_score(y_test, y_pred) # compute accuracy
print("Accuracy:", accuracy)

Accuracy: 0.6725

Try out the classifier on some questions

[111] # choose a number between 0 and 1999
i = 2
label = so_df.loc[i, 'category']
question = so_df.loc[i, 'input_text']

# get the embedding of this question and predict its category
question_embedding = model.get_embeddings(question)[0].values
pred = clf.predict(question_embedding)

print(f"For question {i}, the prediction is '{pred[0]}'")
print(f"The actual label is '{label}'")
print(f"The question text is:")
print("-" * 50)
print(question)

For question 2, the prediction is 'r'
The actual label is 'python'
The question text is:
How do we test a specific method written in a list of files for functional testing in python? The project has so many modules. There are functional test cases being
but I want to test a specific method in that file. Is there any way to test it like that? /p/

[111] start running or compute with GPU

```

To use embeddings for classification, you first convert your data (like text) into embedding numerical vectors that capture meaning. Then, you feed these vectors into a classification model (like a neural network, SVM, or even KNN) to assign labels based on the patterns in the embedding space. Shortcut: Instead of recalculating embeddings every time, precompute and store them in a database. When new data comes in, look up the closest existing embedding instead of reprocessing everything from scratch. This saves time and computational power.

11)

```

Top k
• The default value for top_k is 40.
• You can set top_k to values between 1 and 40.
• The decoding strategy applies top_k, then top_p, then temperature (in that order).

[23] top_k = 20
top_p = 0.7

[24] response = generation_model.generate_content(prompt,
        generation_config=vertexai.generative_models.GenerationConfig(
            temperature=0.9,
            top_k=top_k,
            top_p=top_p,
        )

[25] print(f"[top_p = {top_p}]")
print(response.text)

[top_p = 0.7]
## Tired of the same old boring jackets?

**Step into a world of playful style with our new line of jackets featuring vibrant blue elephants and adorable avocados!**

These aren't your ordinary jackets. We've combined the fun and funky with the practical and cozy to create a collection that's sure to turn heads wherever you go.

**Imagine:**

* The plush comfort of a premium fleece jacket adorned with a playful blue elephant, its trunk playfully curled upwards.
* A lightweight windbreaker featuring a quirky avocado pattern, perfect for keeping you cool and breezy on sunny days.
* A statement-making bomber jacket with a bold, embroidered design of a blue elephant frolicking amongst avocado trees.

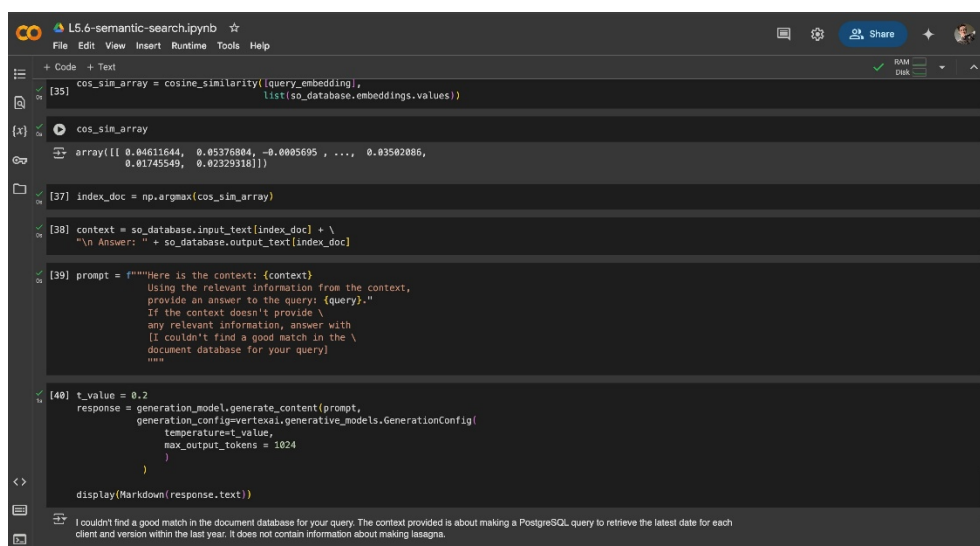
**Our jackets are not just about looking good, they're about feeling good too.** Made with sustainable materials and ethically sourced fabrics, you can feel confident.

**So ditch the dull and embrace the playful!** Our blue elephant and avocado jackets are the perfect way to add a touch of personality and fun to your everyday look.

```


The script demonstrated several features of Vertex AI prompting. It used text-bison@001 for general text generation, showing how to select a model based on the task. Authentication was handled by loading service account credentials with OAuth scopes to grant access to Vertex AI. The script set PROJECT_ID and REGION (us-central1), which are required for sending API requests. It structured a prompt to generate text from the model, demonstrating how to craft inputs for effective responses. Additionally, it integrated with Google Colab by mounting Google Drive to access credentials, making it easier to run in a cloud-based environment.

12)



```
File Edit View Insert Runtime Tools Help
+ Code + Text
[35] cos_sim_array = cosine_similarity(query_embedding,
    list(so_database.embeddings.values))

cos_sim_array
array([[ 0.04611644,  0.85376884, -0.0005695, ...,  0.03502886,
         0.01745549,  0.02329310]])

[37] index_doc = np.argmax(cos_sim_array)

[38] context = so_database.input_text[index_doc] + \
    "\n Answer: " + so_database.output_text[index_doc]

[39] prompt = f"""Here is the context: {context}
Using the relevant information from the context,
provide an answer to the query: {query}."
If the context doesn't provide \
any relevant information, answer with \
'I couldn't find a good match in the \
document database for your query'
"""

[40] t_value = 0.2
response = generation_model.generate_content(prompt,
    generation_config=vertexai.generative_models.GenerationConfig(
        temperature=t_value,
        max_output_tokens = 1024
    )
)

display(Markdown(response.text))

I couldn't find a good match in the document database for your query. The context provided is about making a PostgreSQL query to retrieve the latest date for each client and version within the last year. It does not contain information about making lasagna.
```

Semantic search goes beyond keyword matching by understanding intent and context. It starts with embedding generation, where both queries and documents are converted into vector representations using models like BERT. These embeddings are stored in a vector database, enabling fast similarity searches. When a query is made, it's also embedded and compared to stored vectors using nearest neighbor search to find the closest matches. The results are then ranked and retrieved based on relevance, often incorporating additional factors like metadata or user behavior. This makes search more intuitive, recognizing meaning rather than just exact words.