Vietnam Datathon 2023

Online Submission Round

Topic: CREATE CHATBOT USING CHATGPT API FOR RECOMMENDATIONS WITH EMBEDDINGS

Team: Bot to Top

Vietnam Datathon 2023

General idea

We will create a chatbox where customers can interact and provide information about the products they need. Using the data information obtained, the chatbox will return suitable products with the best recommendations.
The distinct advantage compared to regular sales chatboxes is: Our team's chatbox will respond according to customer requests and interactions are not fixed (rigid interactive answers).
Below is a basic diagram of the idea.

Process Number 01

Receive and store customer information

Process Number 02

Processing customer data

Process Number 03

Compare with available data: then make product recommendations (data comes from dataset.csv file and customer data)

Above is the most general idea of how Chatbox works. Our team will clarify this process in the following pages

About Our Dataset & Library

Dataset 1 (nike_adidas)

We will use Dataset1: Adidas and Nike products. This data was collected previously, specifically in the file adidas_nikes_products_snaphost_data.csv

Dataset 2 (customer_purchases)

We will use Dataset2: Information provided by customers. This data set is collected from customer demographics, previous purchase information and information when customers interact with the chatbox.

Library

We will use library: OpenAI, Pandas

Vietnam Datathon 2023

We have 7 steps to create a chatbox according to the original suggestion

01 Create embeddings for the dataset

02 Create a customer profile dataset and create embeddings for it

03 Create a embeddings for customer chat messages

04 Get previous purchase data similarities

05 Get product database similarities

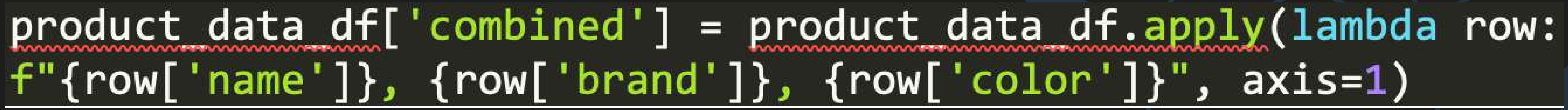
06 Create ChatGPT AI prompt

07 Create ChatGPT product recommendations

Step 1: Create embeddings for the dataset

1. Create a combined attribute for embedding

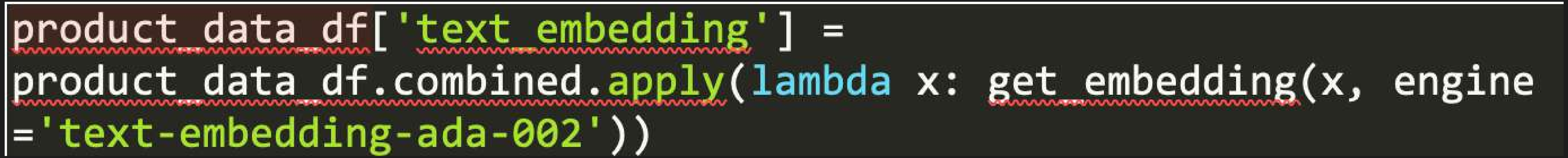
Let's create a new column called combined for the embeddings next, concatenate the name, brand, color, description (extremely important) into the new column combined:



```python
product_data_df['combined'] = product_data_df.apply(lambda row:
f"{row['name']}, {row['brand']}, {row['color']}", axis=1)
```

2. Create embeddings for the dataset

The next step is to create embeddings for the combined column we just created. We'll use get_embedding from OpenAI. This will allow us to represent the product data in the database as vectors in a high-dimensional space, making it easier to calculate their similarity with the user input in the chat later and find the best matches.



```python
product_data_df['text_embedding'] =
product_data_df.combined.apply(lambda x: get_embedding(x, engine
='text-embedding-ada-002'))
```

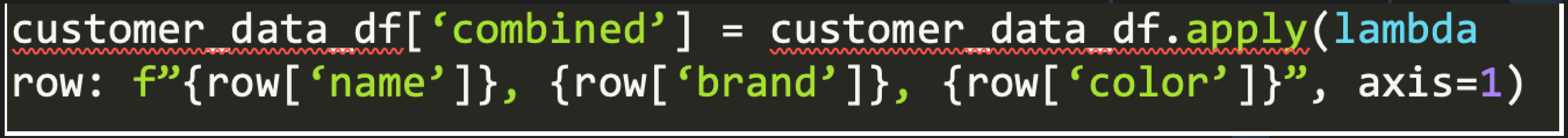
This step can take several minutes depending on the data amount, once finished, you'll have a new column with the numerical representation of the combined column.

Create a column combining the necessary columns to recommend and embed that column into a text_embedding column

Step 2: Create a customer profile dataset and create embeddings for it

1. Create a customer profile dataset

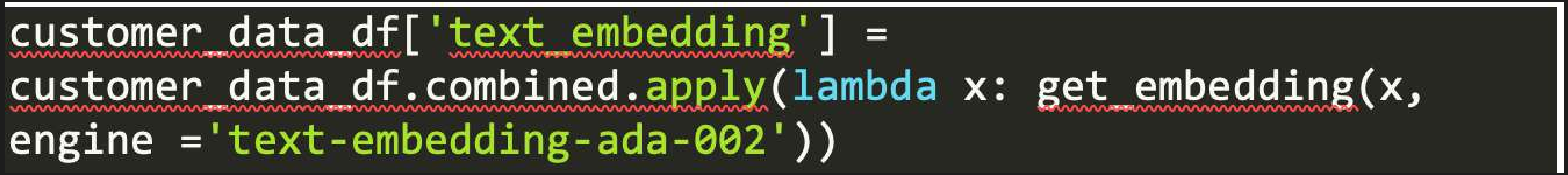
Ideally, this dataset would be past orders or products the customer has previously shown interest in. We name this dataset customer_order_df which has similar attributes as product DataFrame
Next, create a new column for combined purchased product data, just like we did for the product DataFrame:



```
customer_data_df['combined'] = customer_data_df.apply(lambda
row: f"{row['name']}, {row['brand']}, {row['color']}", axis=1)
```

2. Create embeddings for customer profile dataset

Similar to product DataFrame:



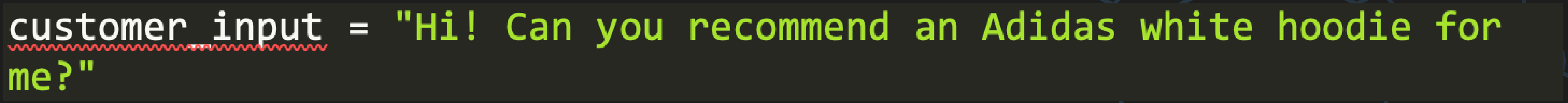
```
customer_data_df['text_embedding'] =
customer_data_df.combined.apply(lambda x: get_embedding(x,
engine ='text-embedding-ada-002'))
```

Create a customer dataset and do the same as the product in step 1. The difference is that the customer has an additional transaction id (prod_id).

Step 3: Create a embeddings for customer chat messages

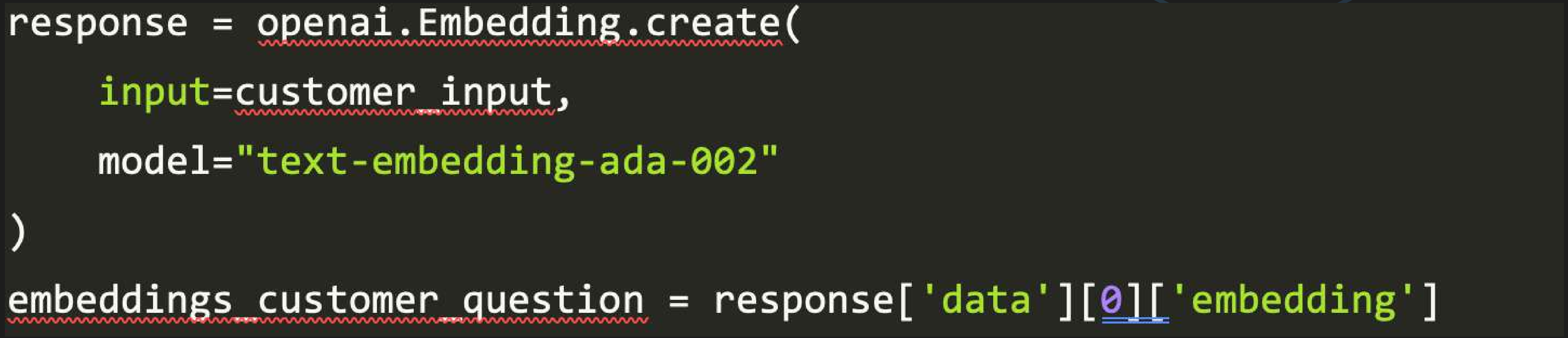
Let's pretend that the customer starts a conversation with your chatbot and asks, "Hi! Can you recommend an Adidas white hoodie for me?"

Start by adding the message as a new customer_input:



```
customer_input = "Hi! Can you recommend an Adidas white hoodie for
me?"
```

Use openai.Embedding.create and make sure to use the same model as for the products:



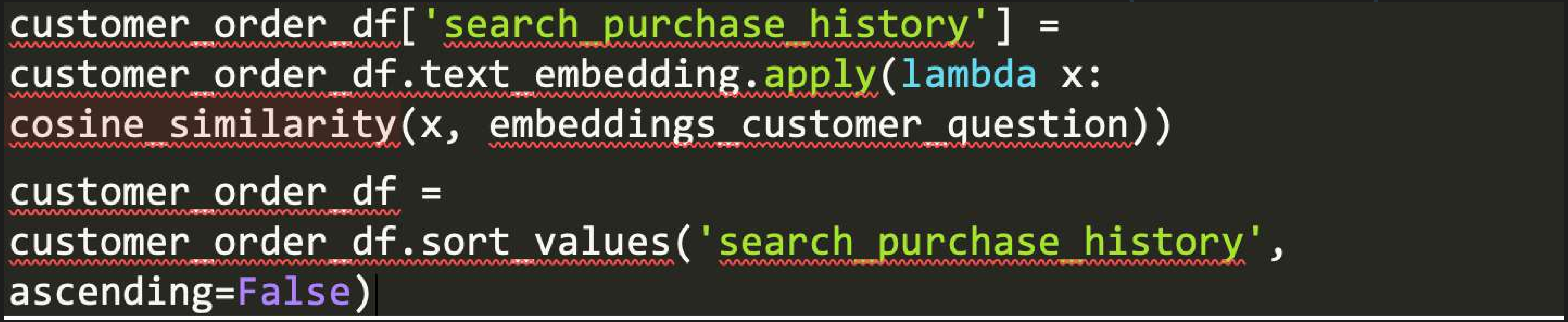
```
response = openai.Embedding.create(
    input=customer_input,
    model="text-embedding-ada-002"
)
embeddings_customer_question = response['data'][0]['embedding']
```

Create an embed for customer messages to compare similarities with product datasets for better recommendations

Step 4: Get previous purchase data similarities

We'll use the endpoint search since we want to find similarities between the user input question: Hi! Can you recommend an Adidas white hoodie for me?with all their previous purchases.
Create a new column in the previous purchase product data DataFrame for the search score and call cosine_similarity for each embedding.

Next, sort the DataFrame in descending order based on the highest score.



```
customer_order_df['search_purchase_history'] =
customer_order_df.text_embedding.apply(lambda x:
cosine_similarity(x, embeddings_customer_question))

customer_order_df =
customer_order_df.sort_values('search_purchase_history',
ascending=False)
```

The previous purchases DataFrame will now have a new column search_purchase_history which is the similarity score between the user question

Create a search_purchase_history column to calculate similarity, then sort in ascending order according to that column to get the top 3 most similar in the next step

Step 5: Get product database similarities

Let's make the same comparison between the user input question and all the products in our product database, and sort the results in descending order based on the highest score:

```
product_data_df['search_products'] =
product_data_df.text_embedding.apply(lambda x: cosine_similarity(x,
embeddings_customer_question))
product_data_df = product_data_df.sort_values('search_products',
ascending=False)
```

The DataFrame should now have a new column search_products, which is the similarity score between the user input question and each product in your database.
Before constructing the ChatGPT API prompt in the next step, let's create two new DataFrames with only the top 3 similarity scores.
One new DataFrame for the previously bought products with the highest similarity scores:

```
top_3_purchases_df = customer_order_df.head(3)
```

And one for the top 3 similarity scores of all the products on our database:

```
top_3_products_df = product_data_df.head(3)
```

Get the top 3 most similar products and deals when getting embedded customer questions

Step 6: Create ChatGPT AI prompt

Start with an empty list:

```
message_objects = []
```

Then append the first message, which is the system message. The system message helps set the behavior of the assistant:

```
message_objects.append({"role":"system", "content":"You're a chatbot helping customers with product recommendations"})
```

After appending the system message, let's add the input message from the customer:

```
message_objects.append({"role":"user", "content": customer_input})
```

Then, let's go ahead and create a string of the previous purchases from our top 3 purchases DataFrame:

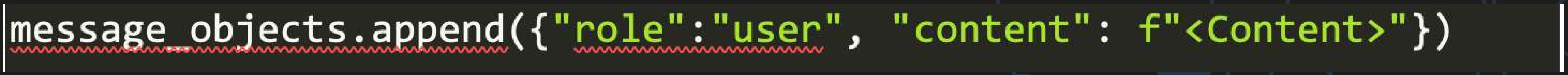
```
prev_purchases = ". ".join([f"{row['combined']}" for index, row in top_3_purchases_df.iterrows()])
```

Add those purchases to a user message and append it to the array of message objects:

```
message_objects.append({"role":"user", "content": f"Here're my latest product orders: {prev_purchases}"})
```

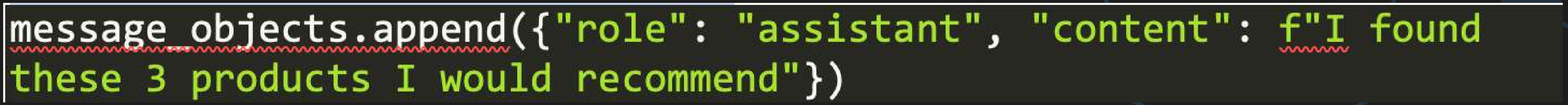
Step 6: Create ChatGPT AI prompt

If we can, add more instruction to help set the assistant's behavior by



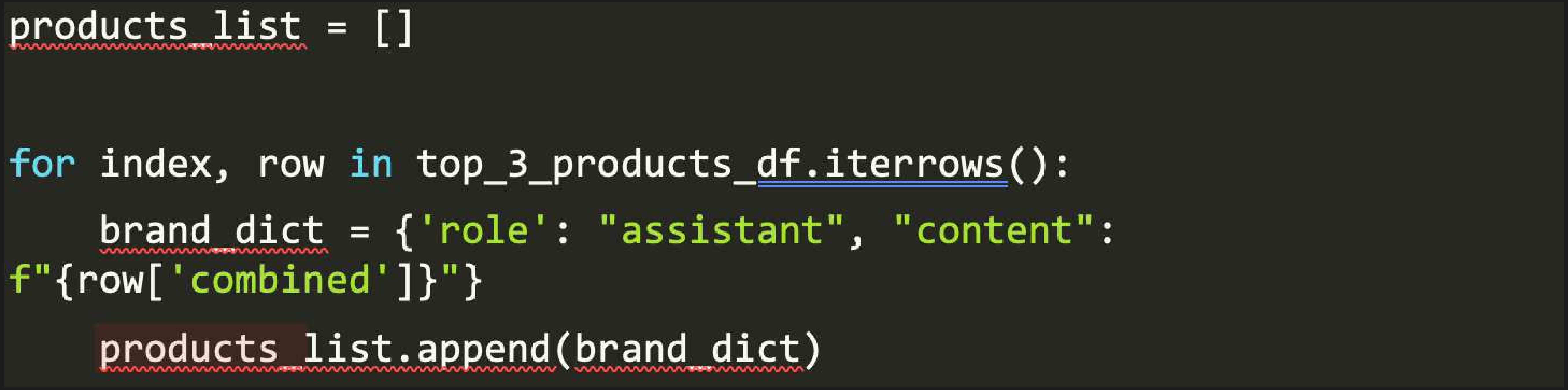
```
message_objects.append({"role":"user", "content": f"<Content>"})
```

After this set of user instructions, I'm adding this assistant content to help give the model an example of desired behavior:



```
message_objects.append({"role": "assistant", "content": f"I found
these 3 products I would recommend"})
```

I'll also go ahead and create a list of the top 3 products we have in our product DataFrame:



```
products_list = []


for index, row in top_3_products_df.iterrows():
    brand_dict = {'role': "assistant", "content":
f"{row['combined']}"}
    products_list.append(brand_dict)
```

Step 6: Create ChatGPT AI prompt

And then add those to our list of message objects with extend:

```
message_objects.extend(products_list)
```

Finally, I'll end the prompt with a last instruction:

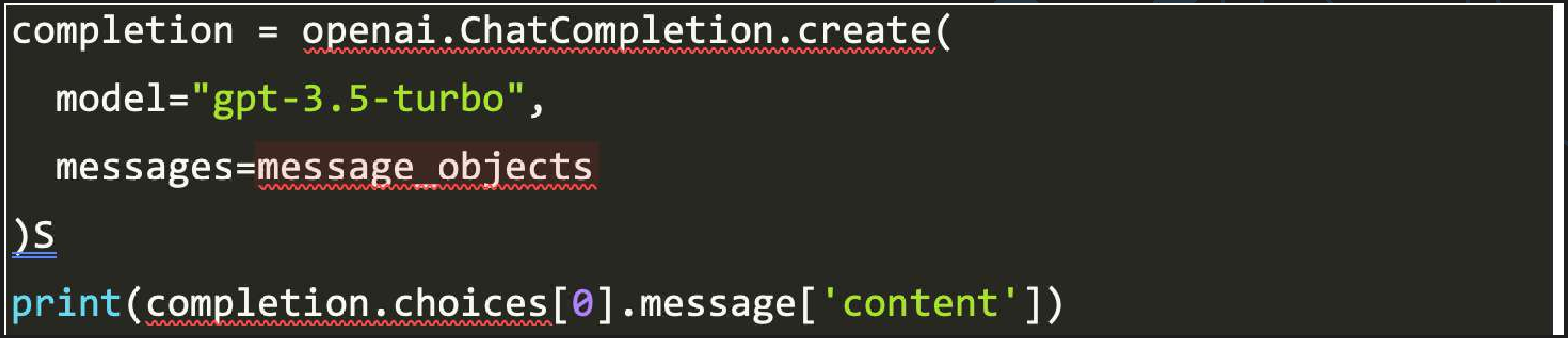
```
message_objects.append({"role": "assistant", "content":"Here's my
summarized recommendation of products, and why it would suit you:"})
```

The last instruction will be continued by the ChatGPT AI generated response.

Create message_objects to store messages between the user and the chatbot (in hardcode) and then use ChatGPT to produce the response.

Step 7: Create ChatGPT product recommendations

The final step is to call the openai.ChatCompletion.create function with our finalized list of message objects:



```
completion = openai.ChatCompletion.create(
  model="gpt-3.5-turbo",
  messages=message_objects
)S
print(completion.choices[0].message['content'])
```

This will give us the AI-generated response to our customer input question based on the previous purchase history and the product database we provided.

Generate ChatGPT response using OpenAI library with message_objects generated from step 6

Dataset Handling using MongoDB and Pandas

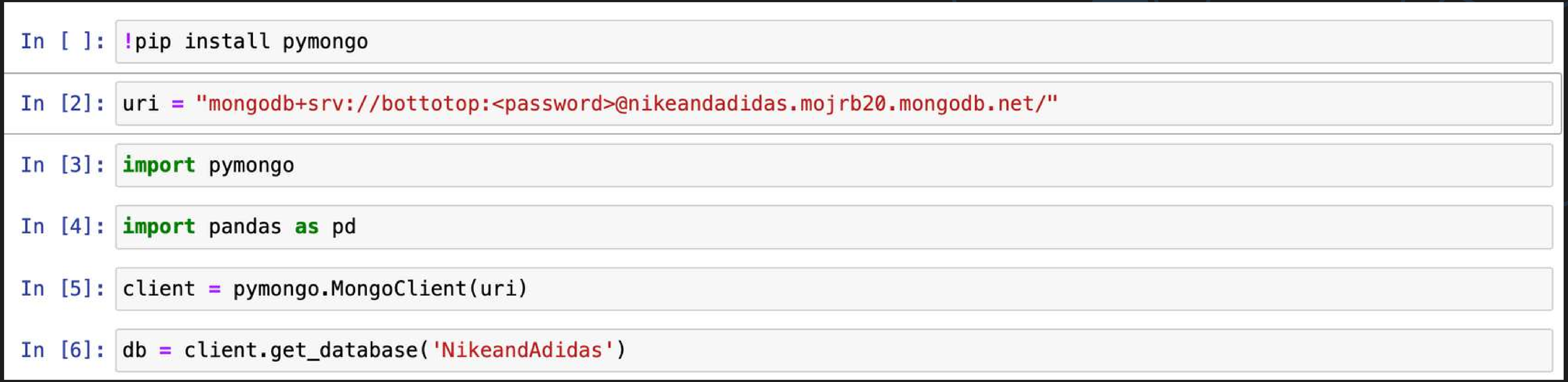
01 Import dataset to MongoDB

02 Use Pandas and PyMongo to handle dataset from MongoDB

03 Export DataFrame back to MongoDB dataset

Step 1: Import dataset to MongoDB

Establish a connection to the MongoDB server using the appropriate connection string or URI.
Verify that the connection is successful and we can interact with the MongoDB database



```
In [ ]: !pip install pymongo

In [2]: uri = "mongodb+srv://bottotop:<password>@nikeandadidas.mojrb20.mongodb.net/"

In [3]: import pymongo

In [4]: import pandas as pd

In [5]: client = pymongo.MongoClient(uri)

In [6]: db = client.get_database('NikeandAdidas')
```

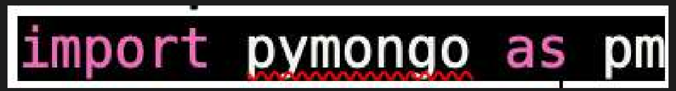
Importing a CSV file (adidas_nikes_products_snaphost_data.csv) with header row into a MongoDB collection:

```
mongoimport --uri <connection-string> --collection products
--file adidas_nikes_products_snaphot_data.csv --type csv --headerline
```

Step 2: Use Pandas and PyMongo to handle dataset from MongoDB

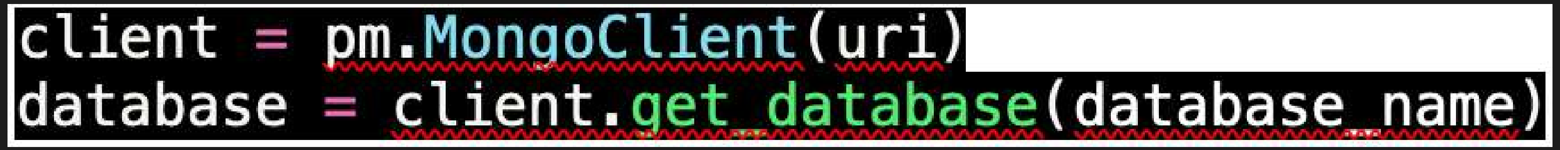
1. Using PyMongo to import dataset from MongoDB

Import APIs:



```
import pymongo as pm
```

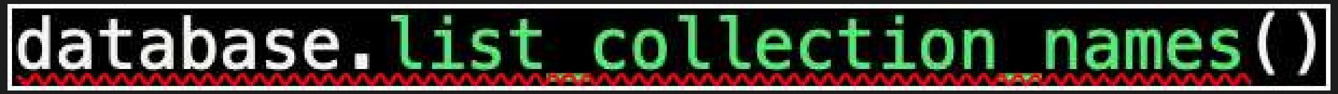
Get database:



```
client = pm.MongoClient(uri)
database = client.get_database(database_name)
```

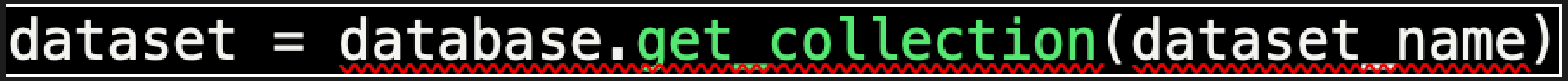
Import uri: The mongoDB client’s uri.

Show all datasets (collections):



```
database.list_collection_names()
```

Get dataset(s):

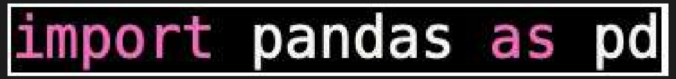


```
dataset = database.get_collection(dataset_name)
```

Step 2: Use Pandas and PyMongo to handle dataset from MongoDB

2. Using Pandas to handle dataset from PyMongo data

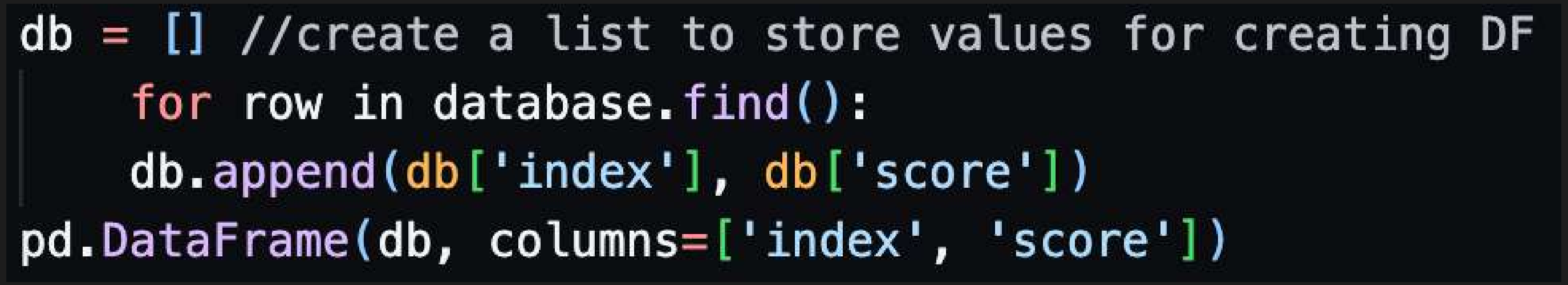
Import APIs:



```
import pandas as pd
```

Creating DataFrame

Assuming this dataset has 2 attributes: Index, score

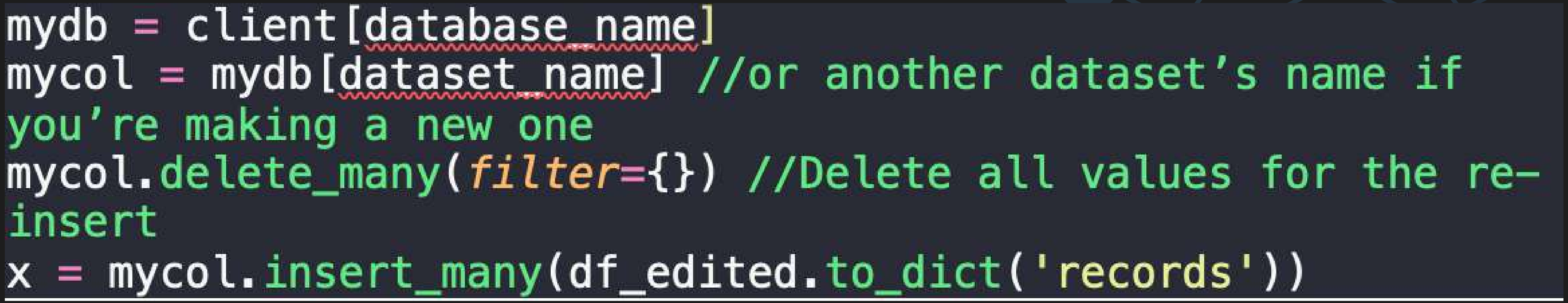


```
db = [] //create a list to store values for creating DF
    for row in database.find():
    db.append(db['index'], db['score'])
pd.DataFrame(db, columns=['index', 'score'])
```

There are lots of Pandas functions that can edit the DataFrame.

Step 3: Export DataFrame back to MongoDB dataset

After you have done editing the DataFrame, you can export that back to MongoDB's dataset by delete all the old values and reassign new values to the dataset.



```
mydb = client[database_name]
mycol = mydb[dataset_name] //or another dataset's name if
you're making a new one
mycol.delete_many(filter={}) //Delete all values for the re-
insert
x = mycol.insert_many(df_edited.to_dict('records'))
```

And that is the minimum viable product (MVP) that leverages real data to propose data-driven solutions in the retail industry by our "Bot to Top" team. We hope that the judges will give us the deepest recognition and evaluation so that we can improve even more in the future. If we join the Team, we will try harder, study hard, and improve our knowledge to achieve greater achievements. Thank you!

The end.