Ask Your Data Questions With Your Own AI Data Scientist Using LangChain's Pandas Data Frame Agent and ChatGPT

If you would prefer a video tutorial click here

Interact with your data via question and answer format as well as generate charts and figures with written commands.

The Rundown:
This tutorial requires Google Colab and Python, HOWEVER:

If you don't know Python, no worries! All of the code is already written, you just need to ask your data questions.

If you know some Python, great! You will most likely understand what is happening under the hood.

If you are a Python expert, even better! You probably understand this better than I do□

In this tutorial, we will explore how to leverage LLMs (Large Language Models) to do Exploratory Data Analysis (EDA), data visualization and interpretation by incorporating Langchain's Pandas Dataframe Agent. EDA is a critical first step in data analysis for data science, academic research and even understanding your own data, personal or business. The process involves exploring the data from multiple angles to begin to see trends that will allow the elucidation of important insights. EDA can include looking at the features (generally the columns in your dataset), maximum and minimums, correlations, building charts and graphs for easy visualization and communication and generating takeaways form your data to inform decisions and take action. Here we will explore the dataset and look at some basic EDA, visualization and statistical techniques using AI.

We will use OpenAI as the LLM and Langchain framework to orchestrate the prompts and the dataset located here

Pre-requisite steps:

1. Sign up for OpenAI and get an API Key
    - Go to platform.openai.com and sign in with an OpenAI account. You get some free tokens just for signing up so no payment necessary!
    - Click on the lock icon in the left hand tray "API keys".
    - Click "Create New Secret Key" to generate a new API key.
    - Copy and paste the key somewhere, once you close this window you will not be able to access the key again and will have to generate a new one
2. Login to Google Drive and have a csv file ready OR feel free to download our sample file from above
3. Click on the "Open in Colab" link to open the notebook! You may need to sign in with Google. Open in Colab

4. Go up to the file menu and select “save a copy in Drive” This will allow you to have a copy of your own forever!
5. Let’s get started!!!

Step 1. Lets install our libraries that we will need

Run the first cell in your notebook. You can either click in the cell and hold shift and press return or click the play button in the top left of the cell.

```
# First, we install langchain and our other dependencies/libraries.
# The -q means quiet so we will see limited output from this cell as it
downloads and installs the libraries
# The -U installs the most updated version of our packages
!pip install -q -U langchain langchain_experimental langchain-openai
typing-extensions cohere
```

Here are the libraries we are installing:

- **Langchain** – This library allows us to set up our chain of AI to enter commands (or wished if you will) in plain English and not only have them turned into code by the LLM but also have the code the LLM creates monitored by our agent to make sure we are getting exactly what we asked for.
- **Langchain-openai** – This library gives us the functions we need in order to make API calls to the OpenAI LLM to generate our code
- **Typing-extension** – enables newer system features of python on older versions to ensure compatibility
- **Cohere**- a natural language processing library that helps simplify human-machine interactions by integrating natural language reducing the need for code.

Step 2. Now we will import the tools we need from the libraries we just downloaded into our notebook

Run the cell just like before, shift+return or hit play.

```
# Now we will import the tools we need from our libraries into the
notebook

import os # A library that allows us to interact with our Operating
System, we will use this for our API key
import pandas as pd # Pandas data frame library for python, the goto for
data analysis and table manipulation
import matplotlib.pyplot as plt # A great graphing library for Python
import seaborn as sns #Another great graphing library, with some extra
features like easy heatmaps
from langchain_experimental.agents import create_pandas_dataframe_agent
#The agent we will be using to explore our data
from langchain.agents.agent_types import AgentType #A tool that allows
for chat history and multi-input tools with OpenAI Functions
from typing_extensions import Iterator # This library provides runtime
support for type hints for the agent
from langchain_openai import ChatOpenAI #Our method of reaching our
OpenAI model of choice
```

Here is what these tools will be used for:
os - A library that allows us to interact with our Operating System, we will use this for our API key. This allows our notebook to "talk" with our computer so that our computer can pass our OpenAI API key onto OpenAI when we make API calls. This prevents us from exposing our API key publicly. A publicly exposed key can be stolen an then someone else can use all of your API credits!

pandas - Pandas data frame library for python, the goto for data analysis and table manipulation. This allows our notebook to construct and manipulate dataframes (tables) as well as do some basic data transformations.

matplotlib.pyplot - A great graphing library for Python. This library will allow our notebook to construct graphs and other data visualizations from our dataframes.

seaborn – Named after Sam Seaborn of the West Wing, this is another great graphing library, with some extra features like easy heatmaps.

create_pandas_dataframe_agent – This is the agent AI that we will use to be the "brain" of our Ai data scientist. It is the central link in the AI chain we are building.

AgentType - A tool that allows for chat history and multi-input tools with OpenAI Functions. This will allow the agent to utilize functions that will allow it to pass on prompts correctly to the OpenAI LLM as well as correct the LLM if it does not send the correct code back for our use case.

Iterator - This library provides runtime support for type hints for the agent

ChatOpenAI – This acts as the method by which our agent can send prompts to and receive code from the OpenAI LLM.

A lot of this may not make sense now but I PROMISE it will once we get to the flowchart when we build our AI chain.

```
# This code simply allows text outputs to wrap so they are easier to
read
from IPython.display import HTML, display

def set_css():
  display(HTML('''
  <style>
    pre {
        white-space: pre-wrap;
    }
  </style>
  '''))
get_ipython().events.register('pre_run_cell', set_css)
```

This cell just formats our text outputs so they are easier to read.

Step 3. Now we need to load our OpenAI API key into the notebook

We can use the secret key in GDrive if you have this set up

```
# If you have your API key stored in your GDrive, you can run this cell
to have your API key passed to OpenAI

from google.colab import userdata
os.environ["OPENAI_API_KEY"] = userdata.get('OPENAI_API_KEY')
```

Or you can paste in your key

```
# Setting our OpenAI API key
os.environ['OPENAI_API_KEY'] = getpass("OpenAI Key:")
```

This will bring up a small window, paste the OpenAI API key you created in OpenAI here.

NOTE- If you are using a Mac you may need to use command+V+shift to paste in the box.

This will allow our agent to make PAI calls to the OpenAI LLM. In plain English, this will allow our agent to talk to the OpenAI LLM because it will allow OpenAI to charge us credits for it.

Step 4. Next up, we need to get some data ready to explore. This can be done one of two ways:

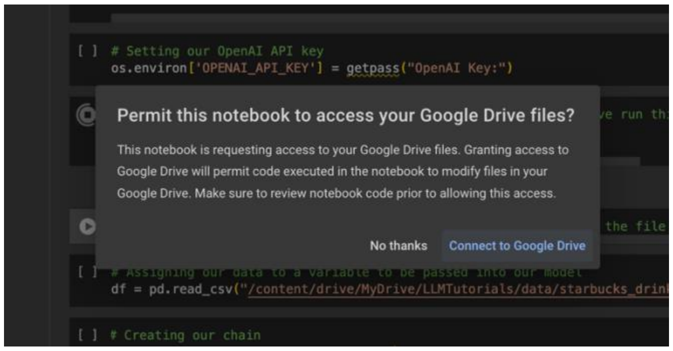
Step 4 Option A. If you would like, you can bring in a CSV file from your GDrive.

First, remove the hashtags from the last two lines of code in this cell (The hashtags tell the notebook not to run whatever comes after them in a line, they are a good way to "comment out" code you do not want to run or simply add comments to line of code so that you can explain what is happening with that code):

```
# To load the CSV file we will be exploring from your Google Drive run
this cell and copy the path and paste it in the cell below where we
assign it a variable
#from google.colab import drive
#drive.mount('/content/drive')
```

```
# To load the CSV file we will be exploring from your Google Drive run
this cell and copy the path and paste it in the cell below where we
assign it a variable
from google.colab import drive
drive.mount('/content/drive')
```

Then click connect to Google Drive and choose your account and click continue two times:

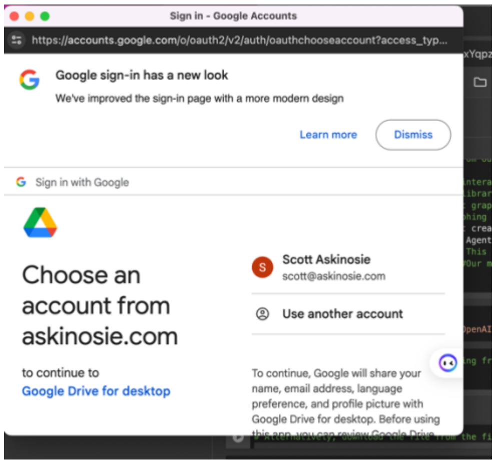


```
[ ] # Setting our OpenAI API key
    os.environ['OPENAI_API_KEY'] = getpass("OpenAI Key:")
```

Permit this notebook to access your Google Drive files?

This notebook is requesting access to your Google Drive files. Granting access to Google Drive will permit code executed in the notebook to modify files in your Google Drive. Make sure to review notebook code prior to allowing this access.

```
[ ] # Assigning our data to a variable to be passed into our model
    df = pd.read_csv("/content/drive/MyDrive/LLMTutorials/data/starbucks_drin

[ ] # Creating our chain
```



Sign in - Google Accounts

https://accounts.google.com/o/oauth2/v2/auth/oauthchooseaccount?access_typ...

Google sign-in has a new look

We've improved the sign-in page with a more modern design

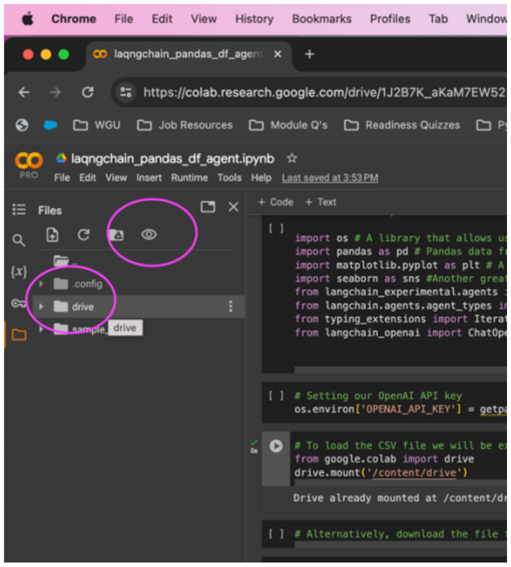
Choose an account from askinosie.com

to continue to Google Drive for desktop

Scott Askinosie
scott@askinosie.com

To continue, Google will share your name, email address, language preference, and profile picture with Google Drive for desktop. Before using

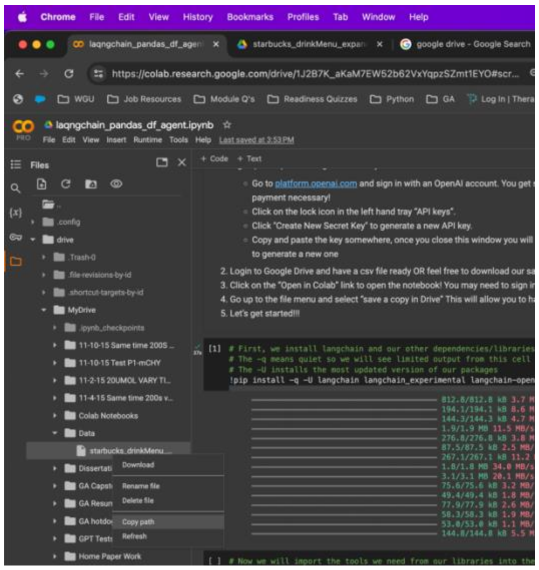
Next, you will need to click on your drive folder to access files stored in your GDrive. If you don't see your Drive folder, no worries, just hit the eyeball once or twice and it will show up.



https://colab.research.google.com/drive/1J2B7K_aKaM7EW52

```
[ ]
    import os # A library that allows us
    import pandas as pd # Pandas data fr
    import matplotlib.pyplot as plt # A
    import seaborn as sns #Another great
    from langchain_experimental.agents i
    from langchain.agents.agent_types im
    from typing_extensions import Iterat
    from langchain_openai import ChatOpe

[ ] # Setting our OpenAI API key
    os.environ['OPENAI_API_KEY'] = getpa

    # To load the CSV file we will be ex
    from google.colab import drive
    drive.mount('/content/drive')

    Drive already mounted at /content/dr

[ ] # Alternatively, download the file f
```

Now you should see the contents of your GDrive, find your CSV file and right click on it and select Copy path.



- Go to platform.openai.com and sign in with an OpenAI account. You get
  payment necessary!
- Click on the lock icon in the left hand tray "API keys".
- Click "Create New Secret Key" to generate a new API key.
- Copy and paste the key somewhere, once you close this window you will
  to generate a new one

2. Login to Google Drive and have a csv file ready OR feel free to download our sa
3. Click on the "Open in Colab" link to open the notebook! You may need to sign in
4. Go up to the file menu and select "save a copy in Drive" This will allow you to ha
5. Let's get started!!!

```
[1] # First, we install langchain and our other dependencies/libraries
    # The -q means quiet so we will see limited output from this cell
    # The -U installs the most updated version of our packages
    !pip install -q -U langchain langchain_experimental langchain-open
```

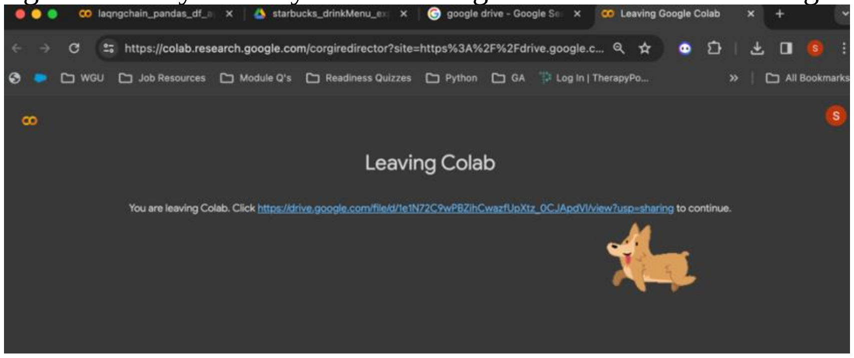
```
812.8/812.8 kB 3.7 M
194.1/194.1 kB 8.6 M
144.3/144.3 kB 4.7 M
1.9/1.9 MB 11.5 MB/s
276.8/276.8 kB 3.8 M
87.5/87.5 kB 2.5 MB/
267.1/267.1 kB 11.2
1.8/1.8 MB 34.0 MB/s
3.1/3.1 MB 20.1 MB/s
75.6/75.6 kB 3.2 MB/
49.4/49.4 kB 1.8 MB/
77.9/77.9 kB 2.6 MB/
58.3/58.3 kB 1.9 MB/
53.0/53.0 kB 1.1 MB/
144.8/144.8 kB 5.5 M
```

```
[ ] # Now we will import the tools we need from our libraries into the
```

You will need the path copied to paste it into the next step.

Step 4 Option B. If you would like to use the CSV I provided, click the link in **The Rundown** above to go to the example dataset.

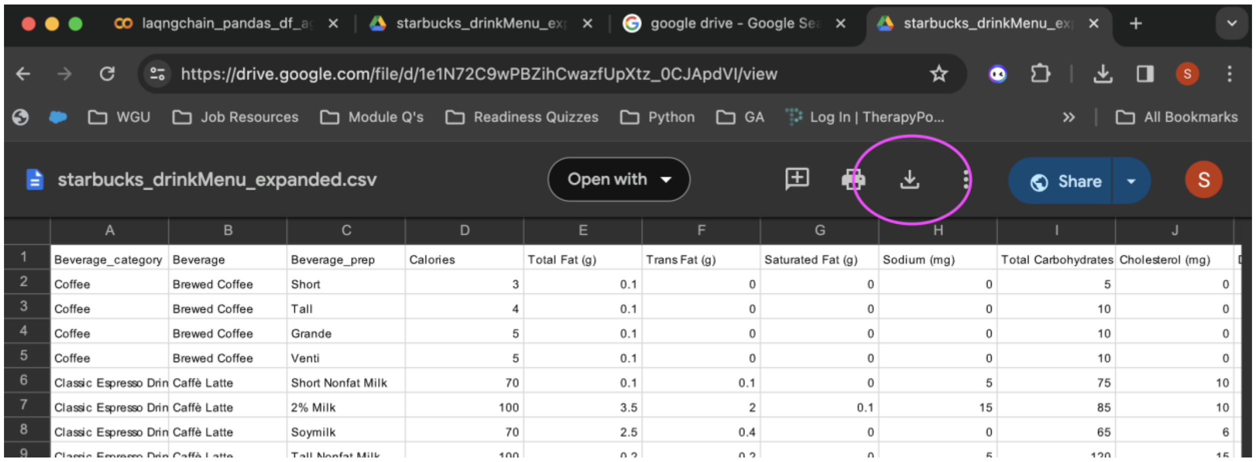
You will get a page that tells you that you are leaving Colab. Click the link to go to the file.



Leaving Colab

You are leaving Colab. Click https://drive.google.com/file/d/1e1N72C9wPBZihCwazfUpXtz_0CJApdVI/view?usp=sharing to continue.

Now you will be in the CSV file, click the download button at the top:

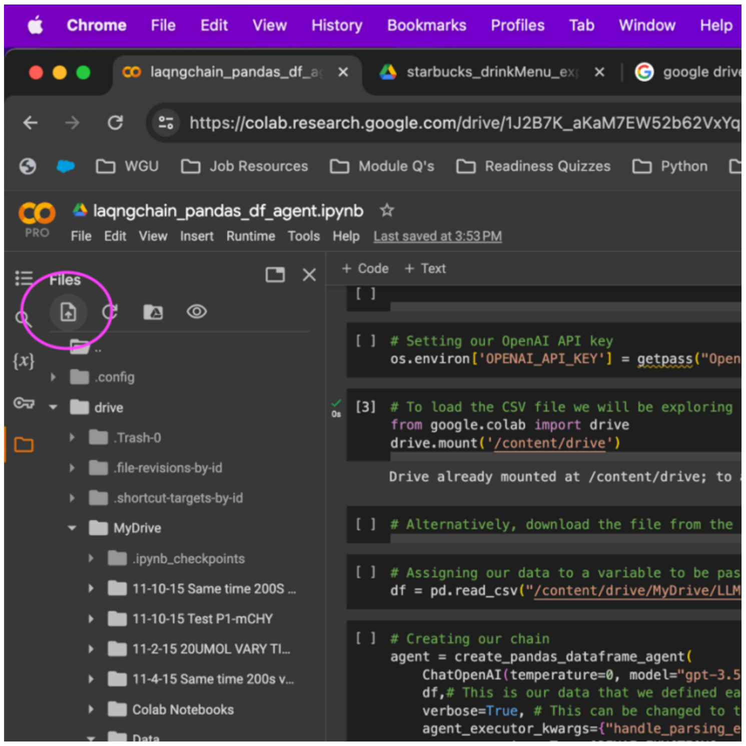


https://drive.google.com/file/d/1e1N72C9wPBZihCwazfUpXtz_0CJApdVI/view

starbucks_drinkMenu_expanded.csv

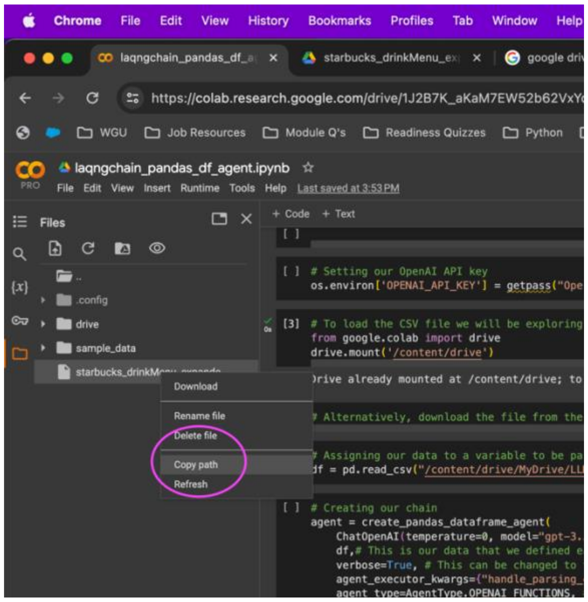
	A	B	C	D	E	F	G	H	I	J
1	Beverage_category	Beverage	Beverage_prep	Calories	Total Fat (g)	Trans Fat (g)	Saturated Fat (g)	Sodium (mg)	Total Carbohydrates	Cholesterol (mg)
2	Coffee	Brewed Coffee	Short	3	0.1	0	0	0	5	0
3	Coffee	Brewed Coffee	Tall	4	0.1	0	0	0	10	0
4	Coffee	Brewed Coffee	Grande	5	0.1	0	0	0	10	0
5	Coffee	Brewed Coffee	Venti	5	0.1	0	0	0	10	0
6	Classic Espresso Drin	Caffè Latte	Short Nonfat Milk	70	0.1	0.1	0	5	75	10
7	Classic Espresso Drin	Caffè Latte	2% Milk	100	3.5	2	0.1	15	85	10
8	Classic Espresso Drin	Caffè Latte	Soymilk	70	2.5	0.4	0	0	65	6

Next go back to your notebook and we will upload the file to our notebook using the upload icon in the left hand pane.


Chrome File Edit View History Bookmarks Profiles Tab Window Help
laqngchain_pandas_df_a
starbucks_drinkMenu_ex
google drive
https://colab.research.google.com/drive/1J2B7K_aKaM7EW52b62VxYq
WGU Job Resources Module Q's Readiness Quizzes Python
laqngchain_pandas_df_agent.ipynb
File Edit View Insert Runtime Tools Help Last saved at 3:53 PM
Files
+ Code + Text
..
.config
drive
.Trash-0
.file-revisions-by-id
.shortcut-targets-by-id
MyDrive
.ipynb_checkpoints
11-10-15 Same time 200S ...
11-10-15 Test P1-mCHY
11-2-15 20UMOL VARY TI...
11-4-15 Same time 200s v...
Colab Notebooks
Data

```
[ ]

[ ] # Setting our OpenAI API key
    os.environ['OPENAI_API_KEY'] = getpass("Open

[3] # To load the CSV file we will be exploring
    from google.colab import drive
    drive.mount('/content/drive')

    Drive already mounted at /content/drive; to

[ ] # Alternatively, download the file from the

[ ] # Assigning our data to a variable to be pas
    df = pd.read_csv("/content/drive/MyDrive/LLM

[ ] # Creating our chain
    agent = create_pandas_dataframe_agent(
        ChatOpenAI(temperature=0, model="gpt-3.5
        df,# This is our data that we defined ea
        verbose=True, # This can be changed to t
        agent_executor_kwargs={"handle_parsing_e
```

Once the file is uploaded, it will appear in your list of files in your notebook. Right click on your CSV file and select copy path:


Chrome File Edit View History Bookmarks Profiles Tab Window Help
laqngchain_pandas_df_a
starbucks_drinkMenu_ex
google dri
https://colab.research.google.com/drive/1J2B7K_aKaM7EW52b62VxY
WGU Job Resources Module Q's Readiness Quizzes Python
laqngchain_pandas_df_agent.ipynb
File Edit View Insert Runtime Tools Help Last saved at 3:53 PM
Files
+ Code + Text
..
.config
drive
sample_data
starbucks_drinkM
Download
Rename file
Delete file
Copy path
Refresh

```
[ ]

[ ] # Setting our OpenAI API key
    os.environ['OPENAI_API_KEY'] = getpass("Ope

[3] # To load the CSV file we will be exploring
    from google.colab import drive
    drive.mount('/content/drive')

    Drive already mounted at /content/drive; to

    # Alternatively, download the file from the

    # Assigning our data to a variable to be pa
    df = pd.read_csv("/content/drive/MyDrive/LL

[ ] # Creating our chain
    agent = create_pandas_dataframe_agent(
        ChatOpenAI(temperature=0, model="gpt-3.
        df,# This is our data that we defined e
        verbose=True, # This can be changed to
        agent_executor_kwargs={"handle_parsing_
        agent_type=AgentType.OPENAI_FUNCTIONS,
```

You will paste the path you just copied into the next cell in the following step.

Step 5. Now we need to set our CSV file as our dataframe variable in our notebook.

Paste the file path you just copied into this line of code in place of paste_file_path_here. You want to make sure that you do not accidentally delete the quotation marks, these are important.

```
# Assigning our data to a variable to be passed into our model
df = pd.read_csv("paste_file_path_here")
```

```
# Assigning our data to a variable to be passed into our model
df = pd.read_csv("/content/drive/MyDrive/Data/starbucks_drinkMenu_expanded.csv")
```

Great! Now run the cell, we have just assigned our CSV file to the variable sf. We will show this to our AI data scientist in just a minute. But first, we need to assemble our AI chain.

Step 6. Creating our AI chain that will become our AI data scientist.

Before we run this code and create our AI chain, lets talk a little bit about what is happening under the hood.

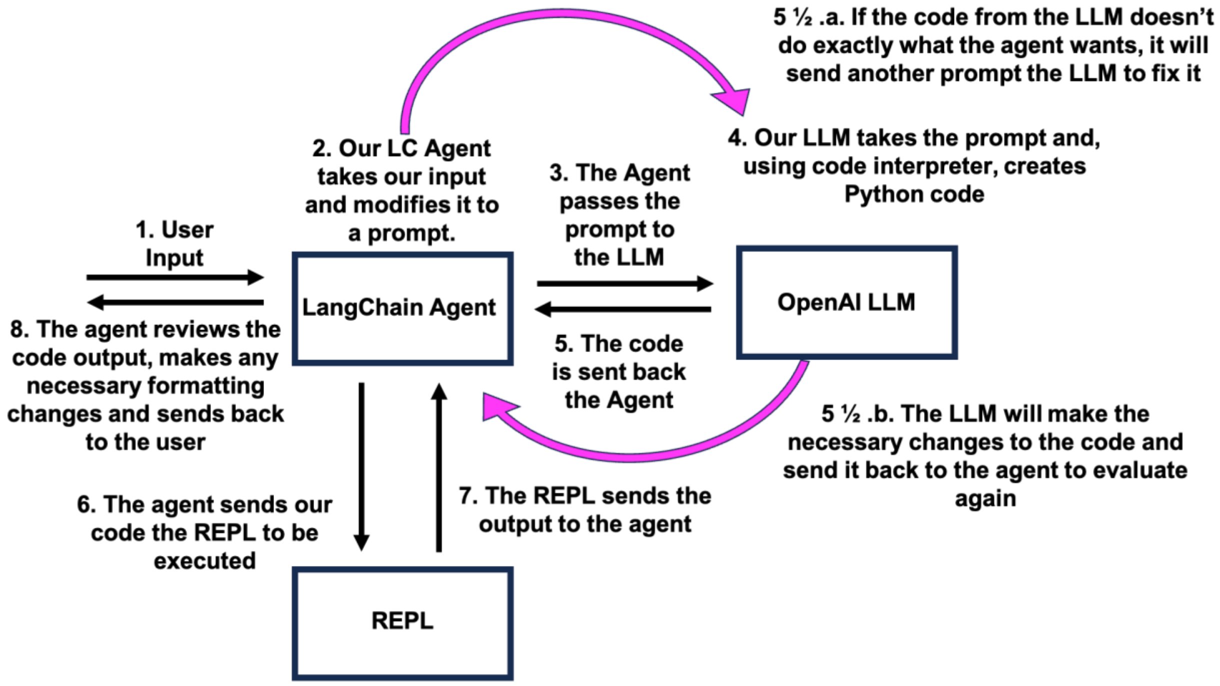

5 ½ .a. If the code from the LLM doesn't do exactly what the agent wants, it will send another prompt the LLM to fix it
2. Our LC Agent takes our input and modifies it to a prompt.
3. The Agent passes the prompt to the LLM
4. Our LLM takes the prompt and, using code interpreter, creates Python code
1. User Input
LangChain Agent
OpenAI LLM
8. The agent reviews the code output, makes any necessary formatting changes and sends back to the user
5. The code is sent back the Agent
5 ½ .b. The LLM will make the necessary changes to the code and send it back to the agent to evaluate again
6. The agent sends our code the REPL to be executed
7. The REPL sends the output to the agent
REPL

Figure 1. AI Chain Including Agent, LLM and REPL

At first glance, this is a pretty busy figure. But lets break it down step by step, it is not that complicated.

1. **User Input** – This is pretty straightforward, this is the question about the data that is being asked. This comes from us and is passed on to our AI agent.

2. **Our LC Agent takes our input and modifies it to a prompt –** Our AI agent is going to take our question and figure out a good prompt to pass on to the OpenAI LLM in order to get the code it needs to return the result that we are asking it for.
3. **The Agent passes the prompt to the LLM** – Our agent then passes the prompt it generated to the LLM to be turned into code.
4. **Our LLM takes the prompt and, using code interpreter, creates Python code** – The LLM takes the prompt and converts it into code that will generate our desired output.
5. **The code is sent back the Agent** – The LLM then passes the code it created back to our agent to review.

5 ½ a. **5 If the code from the LLM doesn't do exactly what the agent wants, it will send another prompt the LLM to fix it** – The agent will review the code and if the code does not yield the desired result it will send another prompt to the LLM to have it try again.

5 ½ b. **The LLM will make the necessary changes to the code and send it back to the agent to evaluate again** – Just like before, the LLM will generate new code and send it back to the agent for review.

6. **The agent sends our code the REPL to be executed** – Once the agent is satisfied with the code generated by the LLM it will pass it to the REPL to be executed.
7. **The REPL sends the output to the agent** – The REPL executes the code and sends the output back to our agent for review.
8. **8. The agent reviews the code output, makes any necessary formatting changes and sends back to the user** – The agent reviews the output from the REPL and makes any changes to the formatting based on the initial input we gave it and then returns the desired output to our notebook!

There it is in a nutshell! Not too bad, right?

So lets look at the code and then run it.

```
# Creating our chain
agent = create_pandas_dataframe_agent(
    ChatOpenAI(temperature=0, model="gpt-3.5-turbo-0613"), # Setting
up our model and how it performs (Temp-How often the model returns a
low probability token, model-The model we are using)
    df,# This is our data that we defined earlier
    verbose=False, # This can be changed to true to see the promts
the agent generats to execute the query
    agent_executor_kwargs={"handle_parsing_errors": True}, #Very
important, this allows the agent to handle errors in the prompts and
returns from GPT
    agent_type=AgentType.OPENAI_FUNCTIONS, # Defining that we are
utilizing the OpenAI Functions agent type
)
```

Here we are creating our AI chain by defining the parameters of our agent, dataframe and LLM:

- **agent** – This is the variable we are defining as our agent. The parameters follow in the parenthesis
- **create_pandas_dataframe_agent** – This is a function within the LangChain libarary that will construct the AI chain for us. There is A LOT happening under the hood with this, I honestly don't understands most of it.

- **ChatIOpenAI –** This defines the parameters with which our OpenAI LLM will use.
    - **temperature** – This defines how "creative" our LLM will behave. Lower numbers, like 0, allow it no creativity. The highest value, 2, allows it to answer with the most creative answers. Since we are asking questions about data we want just the facts, so we assign it a value of 0.
    - **model** – This indicates the model we will be using from OpenAI. We choose `gpt-3.5-turbo-0613`because it is fast, reliable and inexpensive to call.
- **df –** This is our CSV file that we want it to look at. We assigned this variable earlier.
- **verbose** – This determines if our notebook will give us the details of the "conversation" between our agent, LLM and REPL. This will include code from our LLM and return from the REPL before our agent formats the return. We have it set to True so we can see what is happening with the agent AND it makes the outputs easier to read.
- **agent_executor_kwargs-** This is one of the most important setting, this allows the agent to review the code sent back from the LLM before passing it on to the REPL. If the LLM gives the agent bad code and that is passed onto the REPL our cell will fail and we will get an error. Allowing the agent to review the code before passing it on to the REPL is vital for our chain to work.
- **agent_type-** This sets the agent to be able to communicate with OpenAI functions. Again, a lot of complicated stuff happening under the hood here.

Step 7. At long last! Lets start asking questions about our data!!!

We will start by asking our agent what our dataset is all about.

```
# Lets ask our agent about our dataset
agent.invoke('What is our dataset about?')
```

The dataset appears to be about different beverages and their nutritional information. It includes columns such as Beverage_category, Beverage, Beverage_prep, Calories, Total Fat, Trans Fat, Saturated Fat, Sodium, Total Carbohydrates, Cholesterol, Dietary Fiber, Sugars, Protein, Vitamin A (% DV), Vitamin C (% DV), Calcium (% DV), Iron (% DV), and Caffeine.

Lets ask our agent how we should start exploring.

```
# Lets ask our agent how to get started exploring our data
agent.invoke('How should we begin exploring this dataset?')
```

We use the invoke command to activate or agent. Here is the response:

To begin exploring this dataset, we can perform the following steps:

1. Check the shape of the dataframe using `df.shape` to get the number of rows and columns.
2. Use `df.info()` to get a summary of the dataframe, including the data types of each column and the number of non-null values.
3. Check for any missing values in the dataframe using `df.isnull().sum()`.
4. Explore the unique values in categorical columns using `df['column_name'].unique()`.
5. Calculate basic statistics for numerical columns using `df.describe()`.

6. Visualize the data using plots and charts to gain insights.

These steps will provide an initial understanding of the dataset and help identify any data quality issues or patterns in the data.

Okay, lets try what it says!

```
# Lets start with number one
agent.invoke('Check the shape of the dataframe.')
```

The shape of the dataframe is (242, 18), which means it has 242 rows and 18 columns.

Nice! Lets try #2

```
# Now numbe two
agent.invoke('get a summary of the dataframe, including the data types
of each column and the number of non-null values.')
```

The dataframe has a total of 242 entries. Here is a summary of the dataframe:

- `Beverage_category`: object data type, 242 non-null values
- `Beverage`: object data type, 242 non-null values
- `Beverage_prep`: object data type, 242 non-null values
- `Calories`: int64 data type, 242 non-null values
- `Total Fat (g)`: object data type, 242 non-null values
- `Trans Fat (g)`: float64 data type, 242 non-null values
- `Saturated Fat (g)`: float64 data type, 242 non-null values
- `Sodium (mg)`: int64 data type, 242 non-null values
- `Total Carbohydrates (g)`: int64 data type, 242 non-null values
- `Cholesterol (mg)`: int64 data type, 242 non-null values
- `Dietary Fibre (g)`: int64 data type, 242 non-null values
- `Sugars (g)`: int64 data type, 242 non-null values
- `Protein (g)`: float64 data type, 242 non-null values
- `Vitamin A (% DV)`: object data type, 242 non-null values
- `Vitamin C (% DV)`: object data type, 242 non-null values
- `Calcium (% DV)`: object data type, 242 non-null values
- `Iron (% DV)`: object data type, 242 non-null values
- `Caffeine (mg)`: object data type, 241 non-null values

Cool! Now lets jump down the list to something more complex

```
# You get the idea, lets jump to something more fun and go to number
five
agent.invoke('Calculate basic statistics for numerical columns.')
```

```
       Calories  Trans Fat (g)   Saturated Fat (g)   Sodium (mg)  \
count  242.00...  242.00...       242.00...          242.00...
mean   193.87...   1.307025        0.037603           6.363636
std    102.86...   1.640259        0.071377           8.630257
min     0.000000   0.000000        0.000000           0.000000
25%    120.00...   0.100000        0.000000           0.000000
50%    185.00...   0.500000        0.000000           5.000000
75%    260.00...   2.000000        0.100000          10.000000
max    510.00...   9.000000        0.300000          40.000000

       Total Carbohydrates (g)   Cholesterol (mg)   Dietary Fibre (g)  \
count  242.00...                 242.00...          242.00...
mean   128.88...                 35.991736          0.805785
std    82.303223                 20.795186          1.445944
min     0.000000                  0.000000          0.000000
25%    70.000000                 21.000000          0.000000
50%    125.00...                 34.000000          0.000000
75%    170.00...                 50.750000          1.000000
max    340.00...                 90.000000          8.000000

       Sugars (g)   Protein (g)
count  242.00...    242.00...
mean   32.962810    6.978512
std    19.730199    4.871659
min     0.000000    0.000000
25%    18.000000    3.000000
50%    32.000000    6.000000
75%    43.750000   10.000000
max    84.000000   20.000000
```

Nice! That is a good summary of our data.

Lets see what it means by visual insights in number 6.

```python
# Lets ask how to gain visual insights
agent.invoke('How can we visualize the data using plots and charts to gain insights?')
```

To visualize the data using plots and charts, you can use the matplotlib library in Python. Here are a few examples of how you can create different types of plots and charts to gain insights from your dataframe:

1. Bar Chart: A bar chart can be used to compare different categories or groups. You can use it to visualize the total calories for each beverage category.

```python
```

~~~
import matplotlib.pyplot as plt

# Group the data by beverage category and calculate the total calories
calories_by_category = df.groupby('Beverage_category')['Calories'].sum()

# Create a bar chart
plt.figure(figsize=(10, 6))
calories_by_category.plot(kind='bar')
plt.title('Total Calories by Beverage Category')
plt.xlabel('Beverage Category')
plt.ylabel('Total Calories')
plt.xticks(rotation=45)
plt.show()
```
~~~

2. Pie Chart: A pie chart can be used to show the proportion of each category in a dataset. You can use it to visualize the distribution of beverage preparations.

~~~
```python
# Group the data by beverage preparation and calculate the count
prep_counts = df['Beverage_prep'].value_counts()

# Create a pie chart
plt.figure(figsize=(8, 8))
prep_counts.plot(kind='pie', autopct='%1.1f%%')
plt.title('Distribution of Beverage Preparations')
plt.ylabel('')
plt.show()
```
~~~

3. Scatter Plot: A scatter plot can be used to visualize the relationship between two numerical variables. You can use it to explore the relationship between total carbohydrates and sugars in the beverages.

~~~
```python
# Create a scatter plot
plt.figure(figsize=(8, 6))
plt.scatter(df['Total Carbohydrates (g)'], df['Sugars (g)'])
plt.title('Total Carbohydrates vs Sugars')
plt.xlabel('Total Carbohydrates (g)')
plt.ylabel('Sugars (g)')
plt.show()
```
~~~

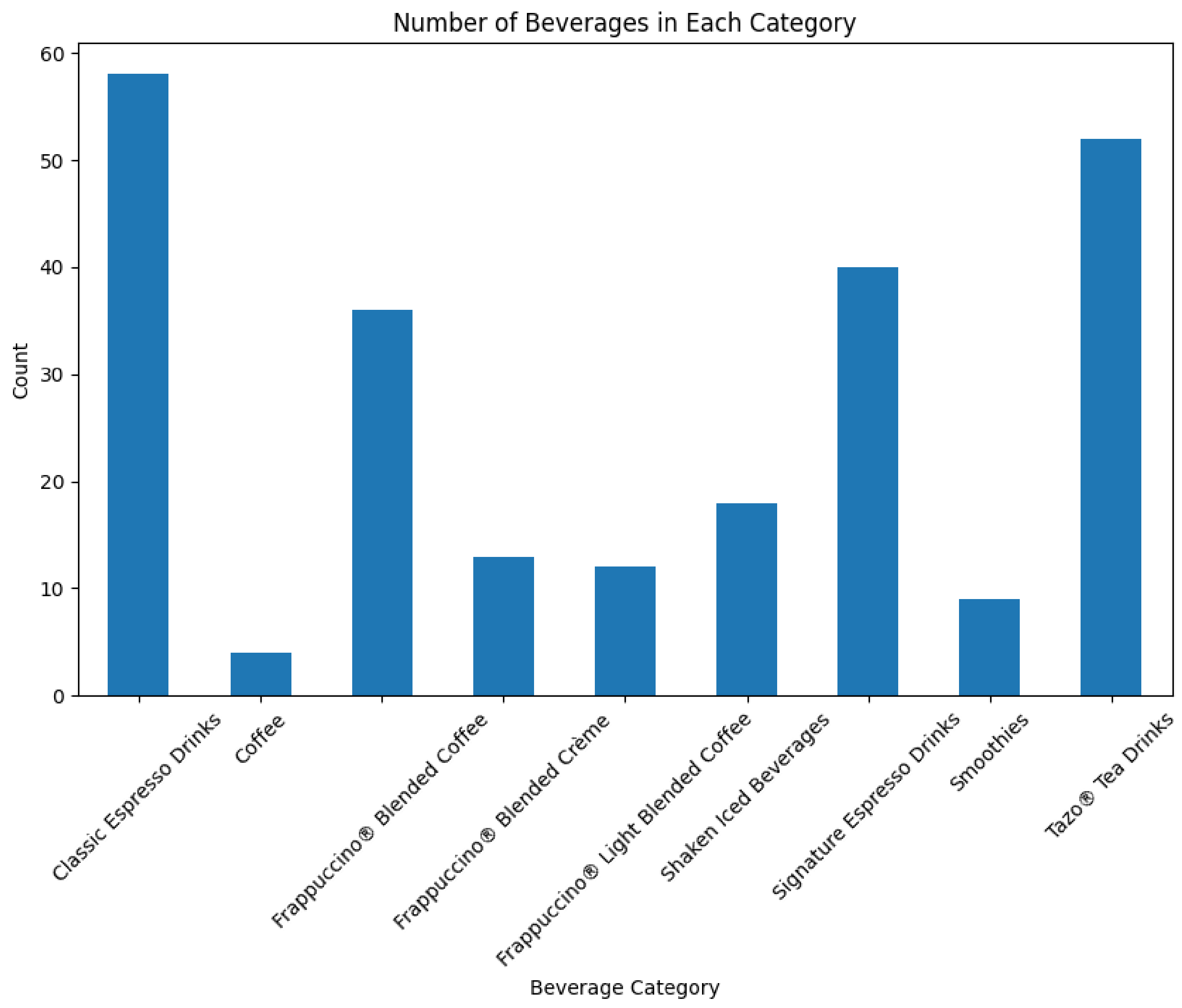
4. Histogram: A histogram can be used to visualize the distribution of a numerical variable. You can use it to explore the distribution of caffeine content in the beverages.

```python
# Create a histogram
plt.figure(figsize=(8, 6))
plt.hist(df['Caffeine (mg)'], bins=10)
plt.title('Distribution of Caffeine Content')
plt.xlabel('Caffeine (mg)')
plt.ylabel('Frequency')
plt.show()
```

These are just a few examples of the types of plots and charts you can create to gain insights from your data. You can customize the plots further by adding labels, titles, legends, and adjusting the plot size according to your requirements.

Lets just do number 1.

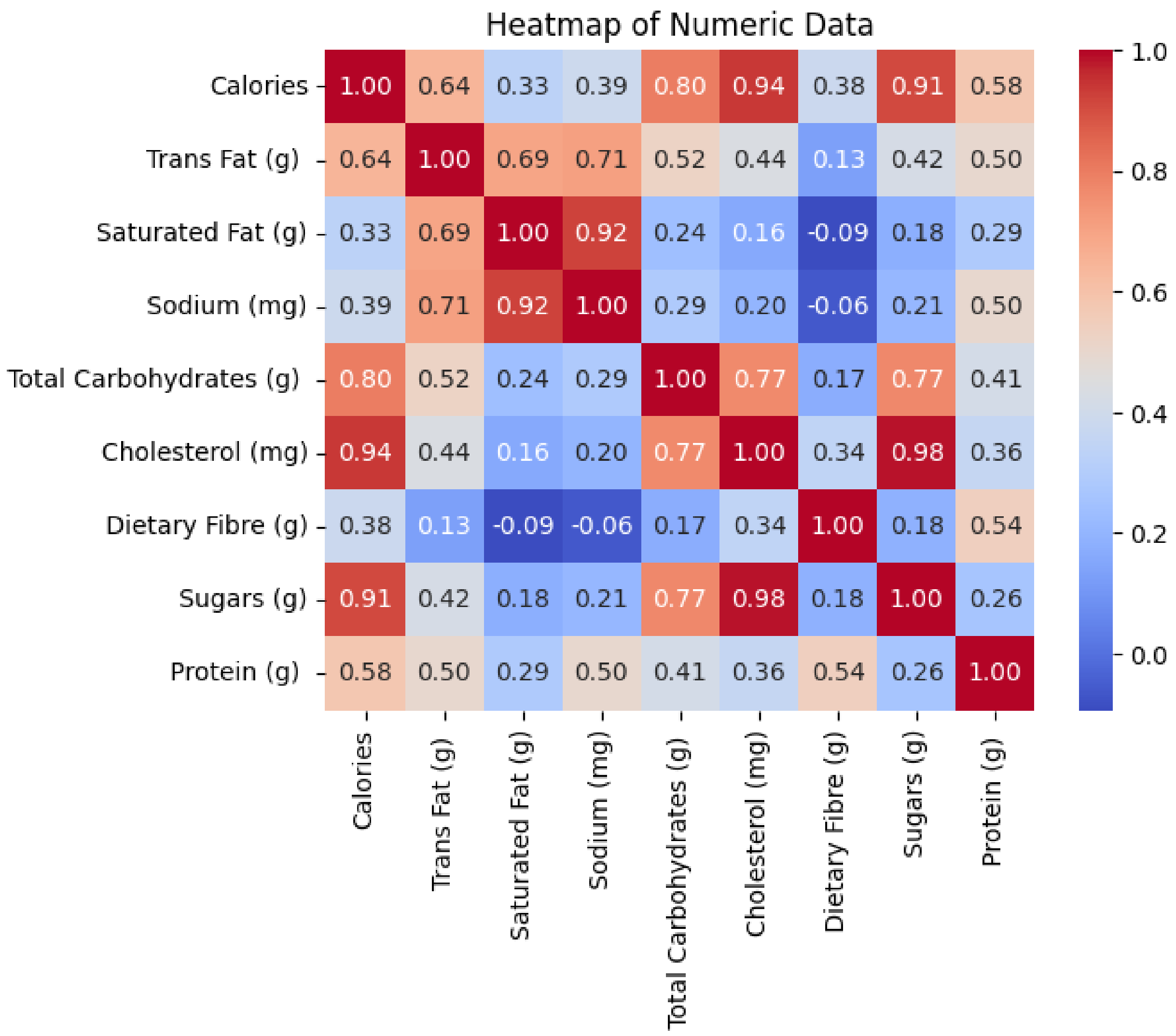
```
# Lets create a bar chart
agent.invoke("Creae a bar chart to compare different categories in the
dataset")
```



The bottom axis labels are a littler misaligned, but that's not a dealbreaker. You can try the other recommended figures on your own!

Lets try a more advanced figure called a heatmap, these show us A LOT of data about correlated numeric values.

```
# Now lets try something more complex, here we will look at a heatmap
(correlation map), we cannot correlate text data
agent.invoke('show me a heatmap of all numeric data')heatmap
```



	Calories	Trans Fat (g)	Saturated Fat (g)	Sodium (mg)	Total Carbohydrates (g)	Cholesterol (mg)	Dietary Fibre (g)	Sugars (g)	Protein (g)
Calories	1.00	0.64	0.33	0.39	0.80	0.94	0.38	0.91	0.58
Trans Fat (g)	0.64	1.00	0.69	0.71	0.52	0.44	0.13	0.42	0.50
Saturated Fat (g)	0.33	0.69	1.00	0.92	0.24	0.16	-0.09	0.18	0.29
Sodium (mg)	0.39	0.71	0.92	1.00	0.29	0.20	-0.06	0.21	0.50
Total Carbohydrates (g)	0.80	0.52	0.24	0.29	1.00	0.77	0.17	0.77	0.41
Cholesterol (mg)	0.94	0.44	0.16	0.20	0.77	1.00	0.34	0.98	0.36
Dietary Fibre (g)	0.38	0.13	-0.09	-0.06	0.17	0.34	1.00	0.18	0.54
Sugars (g)	0.91	0.42	0.18	0.21	0.77	0.98	0.18	1.00	0.26
Protein (g)	0.58	0.50	0.29	0.50	0.41	0.36	0.54	0.26	1.00

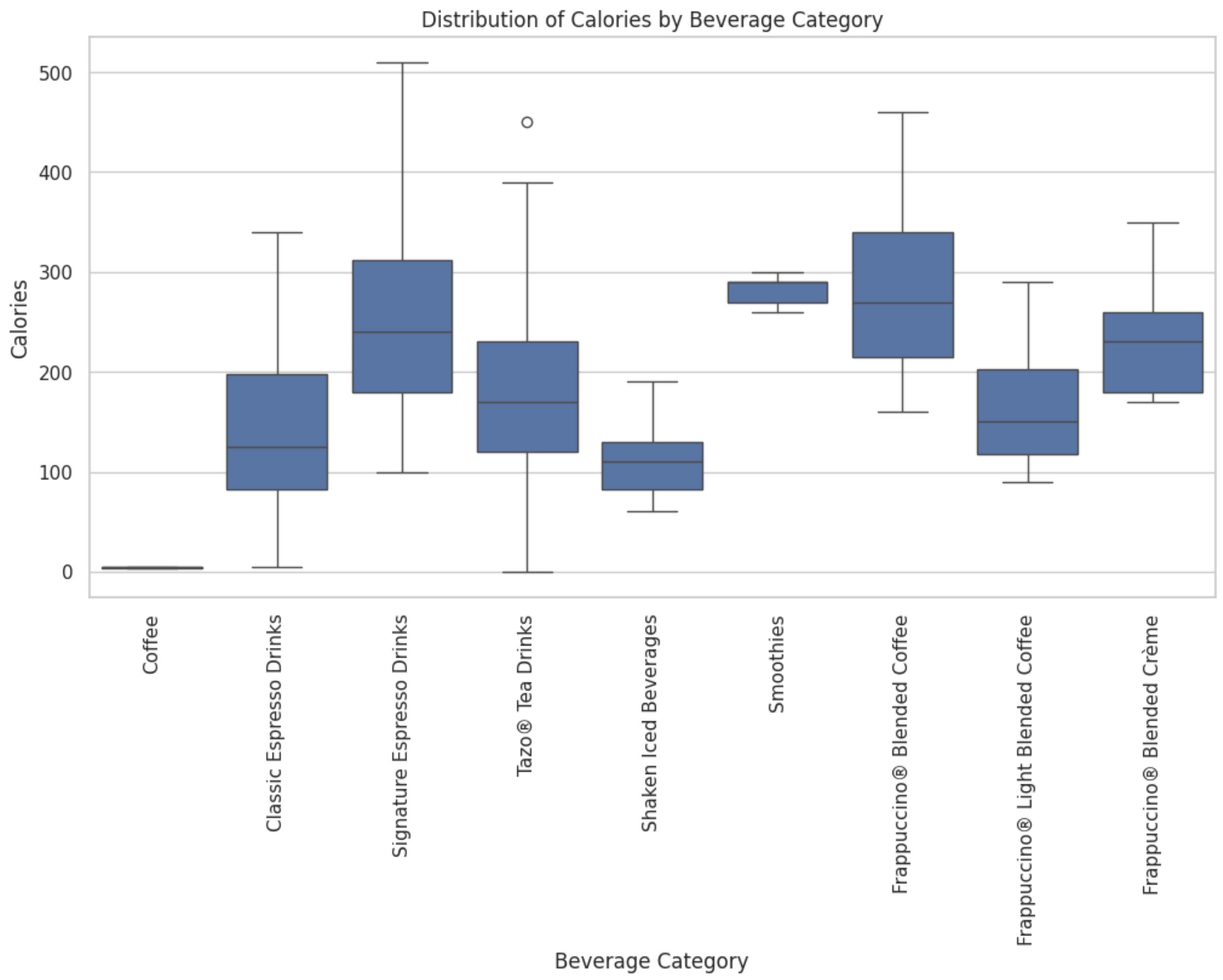
This figure looks great! Nice and clean and the axis are all aligned with labels.

Heatmaps give us correlation values between categories. A number closer to positive 1 shows that those categories are correlated, values closer to 0 tell us they are not correlated and values closer top negative 1 show reverse correlation.

Heatmaps are useful because we can support what we know with data, for instance more sugar is highly correlated to more calories. They can also show us things we may not expect, for instance, saturated fat is highly correlated with sodium and negatively correlated with fiber.

Lets try some stats! We will just do a distribution chart, I know stats are not most people's favorite.

```
# How about some stats?
agent.invoke('show me a distribution of the number of calories for each
category')
```



This is a bar and whisker plot that shows us the mean, standard deviation from the mean and outliers.

One more call, but lets have a little more fun with this one. Lets ask our agent something more subjective. Something that it has to "think" about.

```
# Lets try a subjective question?
agent.invoke('what is the most healthy drink in our data?')
```

The most healthy drink in our data is Tazo® Tea with 0 calories, 0 grams of total fat, 0 grams of trans fat, 0 grams of saturated fat, 0 milligrams of sodium, 0 grams of total carbohydrates, 0 milligrams of cholesterol, 0 grams of dietary fiber, 0 grams of sugars, and 0 grams of protein. It does not contain any vitamin A, vitamin C, calcium, or iron. The caffeine content varies.

Awesome! It knew what we meant and gave us the correct answer!!!

Great work! Now get out there and try this on your own. To ask your own questions, simply copy and paste the invoke code into a new cell and put your own question or command in the quotation marks. And if you have any questions, hit me up anytime.