Bulba Code ICE - Advanced ICE E2E RLHF Instructions

| **IN PROGRESS**

Table of Contents

Task Overview

*The primary objective of these tasks is to generate **R**einforcement **L**earning through **H**uman **F**eedback (RLHF) data to train an AI Model to **break down** a request and **build up** a solution.*

At each step of the conversation, you will be shown two different model responses and asked the following questions in order:

1. You will be shown two model responses and asked to select the “Best model response” of the two.
2. Next answer questions about each model response. For each of the two Model Responses generated, you will be given a set of Single-Sided Ratings to provide information about the content of each Step.

3. Provide a Side-by-Side (SxS) Rating comparing the two model responses. Also provide reasoning about why a response is better than another.
    a. ***NOTE:*** Make sure your reasoning is aligned with your choice in Step 1.
4. If the current Step was unsatisfactory in any way, then you will also have the opportunity to Rewrite the Step and make edits where you see fit.

The Model breaks down a request into multiple steps that follow the order highlighted by the diagram below. Each Step has its own Specialisation & Context, which you can read more about here. In the end, the whole process will be equivalent to one (1) Prompt & Response Pair on an LLM Chatbot in Production like ***Open AI's ChatGPT***, ***Google's Bard***, or ***Meta's LLaMa***.

NOTE: *In other words, this is not a typical conversation with an LLM Chatbot, but the internal mechanisms that go into training these LLM's.*

Key Takeaways

Here are some important notes to grasp from the above excerpt:

- The main objective here is not to write the best, most functional answer. The main objective is to guide the Model and instruct it how to generate an answer on its own, debugging errors alongside it and providing insights will help it generate a functional response.
    - ***NOTE:*** *We are teaching the Model how to solve problems,* ***not*** *answer the specific questions in one shot. The conversation can straight away jump to the answer.*
- The Model's context is akin to being trapped in a Jupyter Notebook with no access to external tools.
    - ***NOTE:*** *Do* ***not*** *tell the Model new information. Provide instructions that will lead the Model to learn & have access to new information on its own.*
- All numbers referred to in any step have to be fully backed by outputs from one of the code executions.
    - *Example:* ***If the response says "Average Price of a Hotdog is $4" then $4 should be the average calculated & printed by the python code in the previous step.***
- If the current Step is Code Generation then you will have the opportunity to run and test the code you're developing in a Code Editor (Sphere Engine).
    - ***NOTE:*** *The Code Editor is* ***not*** *to be used to view the entire dataset and gain knowledge that the Model does* ***not*** *have.*
- Each Sub-Model will have access to its own type of context. Make absolutely sure that you're aware of what each Model is aware of while grading or rewriting responses.

Examples of Model Steps

Below are examples of how each step type will be generated

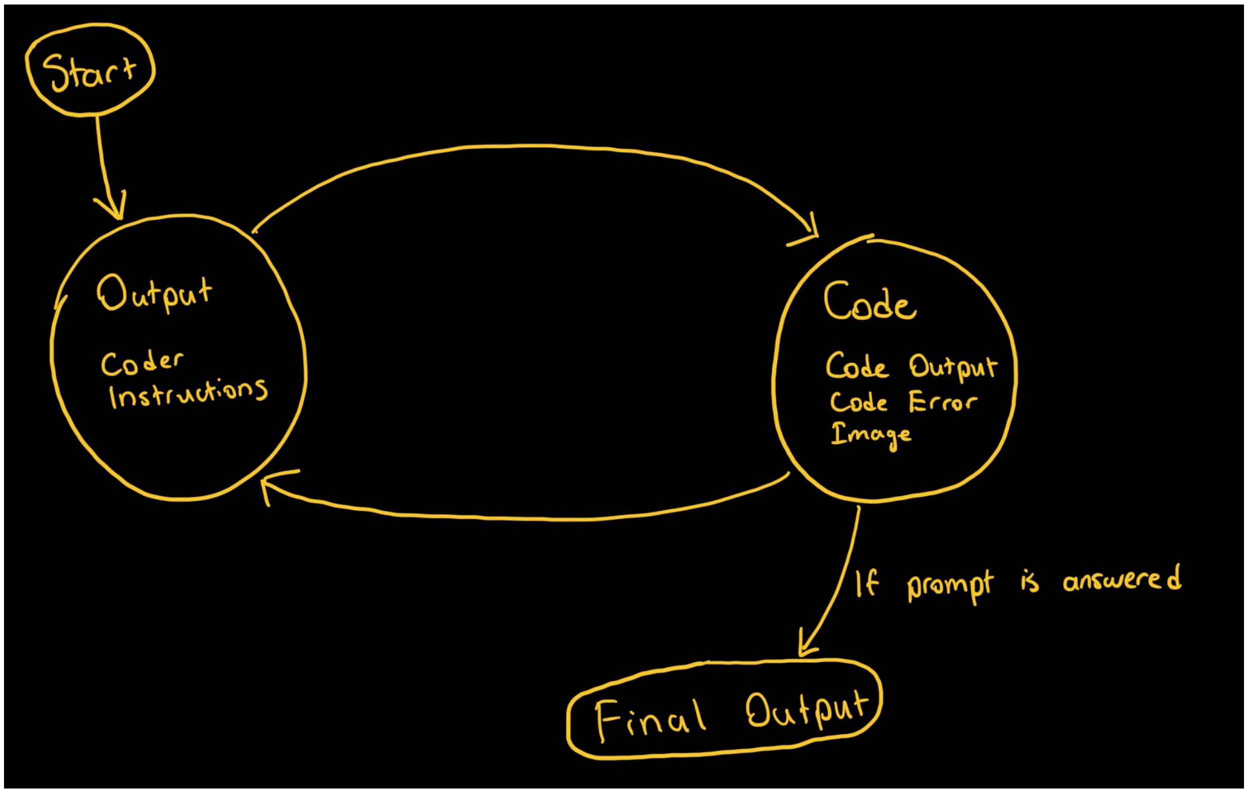
User Prompt	Dataset
How many rows are in this Dataset?	dataset.csv

Planning Step	Code Generation Step	Natural Language Response

Output:
I need to load in `dataset.csv` and grab the number of rows once it's loaded in.
Coder Instruction:
• Import the contents of `dataset.csv` using Pandas.
• Get the number of rows using `len(df)`.
	Code:
	Python
	`import pandas as pd`
	`# Read the contents of the dataset`
	`df = pd.read_csv("dataset.csv")`
	`# Print the number of rows in df`
	`print(len(df))`
	Code Output:
	Unset
	`37`
		Final Output:
		Your dataset `dataset.csv` has 37 rows in it.

Simplified Task Flow Diagram

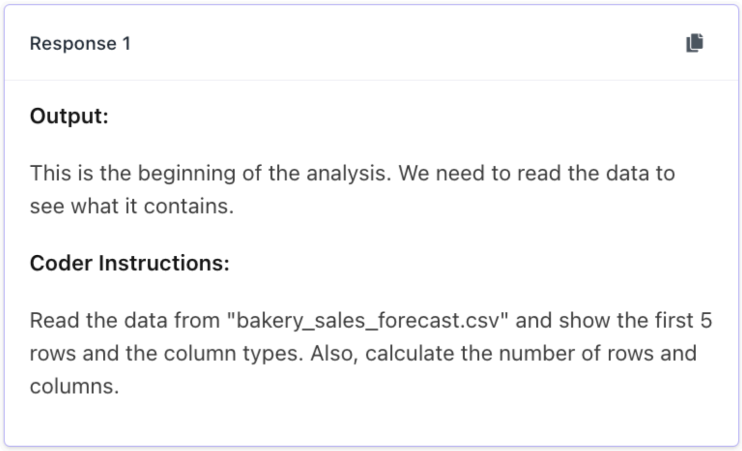
The diagram below illustrates how the Task will be completed as you work through it



Model Step Explanations

The Model we're training is generates Responses in steps, of which there are three types:

1. **The Planning Step** – This model will outline ***how*** Code needs to be generated and what purpose it needs to accomplish. Any Responses from this Model must ***never*** reference previous instructions, and only ever reference the prompt, any previous generated Code or generated Code Execution Output.



Response 1

Output:

This is the beginning of the analysis. We need to read the data to see what it contains.

Coder Instructions:

Read the data from "bakery_sales_forecast.csv" and show the first 5 rows and the column types. Also, calculate the number of rows and columns.

a. **Output + Coder Instructions** – An explanation of what the next steps are & and any insights associated with them **+** Bulleted Instructions for what ***exactly*** each next step is.

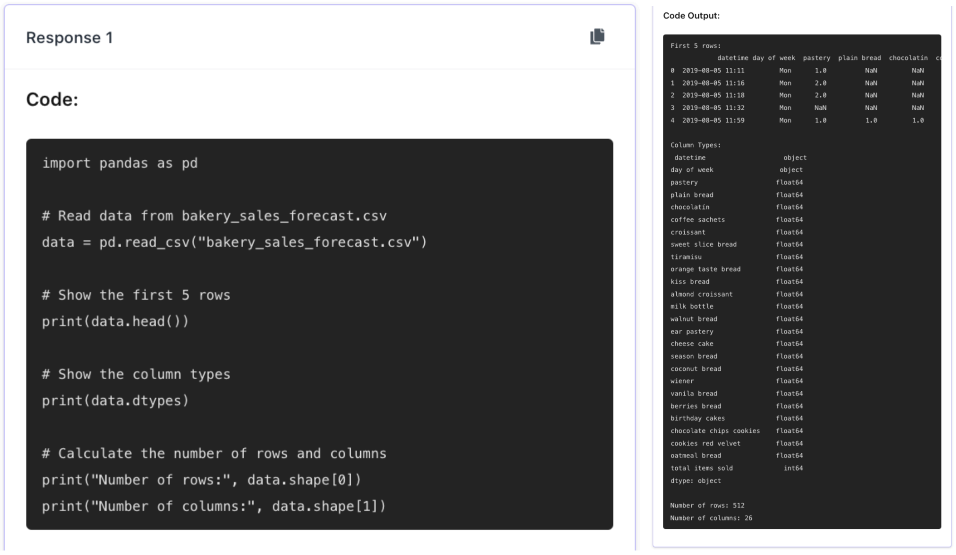
   i. ***NOTE:*** *The **Output** section may be omitted if it was not generated by the Model.*

   ii. ***NOTE:*** *The **Output** section should only contain information relevant to the prompt.*

   iii. ***NOTE:*** *The **Output** section is exposed to the User, so it **needs** to be written concisely, properly, and completely.*

   iv. ***NOTE:*** *The **Coder Instructions** section should **never** have Code snippets in it. Individual small 1-line functions or variable names are an exception.*

2. **The Code Generation & Execution Step** – This step will generate ***Python 3.10*** Code to accomplish whatever task was outlined by the instructions. It does ***not*** have knowledge of the Prompt nor Previous Planning Steps. This Model's code execution will occur by concatenating previous code generation steps together to create one cohesive script.



Response 1

Code:

```
import pandas as pd

# Read data from bakery_sales_forecast.csv
data = pd.read_csv("bakery_sales_forecast.csv")

# Show the first 5 rows
print(data.head())

# Show the column types
print(data.dtypes)

# Calculate the number of rows and columns
print("Number of rows:", data.shape[0])
print("Number of columns:", data.shape[1])
```

Code Output:

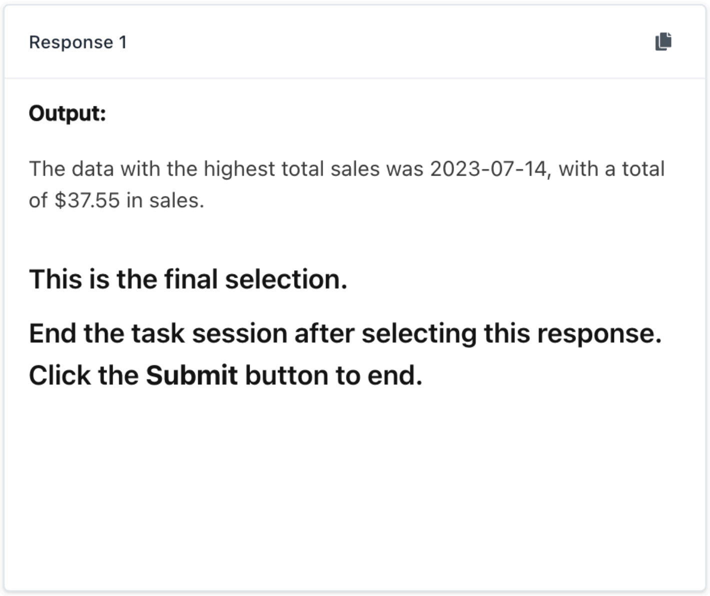
```
First 5 rows:
            datetime day of week  pastery  plain bread  chocolatín  c
0  2019-08-05 11:11         Mon      1.0          NaN         NaN
1  2019-08-05 11:16         Mon      2.0          NaN         NaN
2  2019-08-05 11:18         Mon      2.0          NaN         NaN
3  2019-08-05 11:32         Mon      NaN          NaN         NaN
4  2019-08-05 11:59         Mon      1.0          1.0         1.0

Column Types:
 datetime                    object
day of week                 object
pastery                    float64
plain bread                float64
chocolatín                 float64
coffee sachets             float64
croissant                  float64
sweet slice bread          float64
tiramisu                   float64
orange taste bread         float64
kiss bread                 float64
almond croissant           float64
milk bottle                float64
walnut bread               float64
ear pastery                float64
cheese cake                float64
season bread               float64
coconut bread              float64
wiener                     float64
vanila bread               float64
berries bread              float64
birthday cakes             float64
chocolate chips cookies    float64
cookies red velvet         float64
oatmeal bread              float64
total items sold             int64
dtype: object

Number of rows: 512
Number of columns: 26
```

   a. **Code Generation + Code Execution Output** – Generated Code made to provide a solution to the Previous Coder Instructions **+** The textual output of `stdout` and `stderr` and/or any images generated by matplotlib.

3. **The Final Analysis (Natural Language) Step** – This model will use the context of the Prompt and ***all*** Code Generation & Code Execution Steps to create a response suitable to hand off to a User as if they had asked this question to a LLM Chatbot like ***Open AI's ChatGPT***, ***Google's Bard***, or ***Meta's LLaMa***.


Response 1

Output:

The data with the highest total sales was 2023-07-14, with a total of $37.55 in sales.

This is the final selection.

End the task session after selecting this response. Click the Submit button to end.

a. **Final Output** – A Natural Language response to Prompt's request using the context provided by the Prompt & the previously Generated Code & Code Execution Outputs.
   i. ***NOTE:*** *The **Final Output** section should only contain information relevant to the prompt.*
   ii. ***NOTE:*** *The **Final Output** section is exposed to the User, so it **needs** to be written concisely, properly, and completely.*

Simplified Response Generation Flow

Each Task will receive a Prompt & Dataset. From here, you're expected to follow the Model along as it generates each Step to solve the Prompt's request.

1. **Exploratory Data Analysis (EDA) – Without access to contents of the Dataset, we first need to retrieve and understand a small sample of the dataset before we can decide what code the model needs to be generating next.**
   a. **Planning Step:**
      i. **Output** – An explanation stating that the Model needs to understand the dataset before anything can be done.
      ii. **Coder Instructions** – A bulleted list outlining how the steps in Output are going to be executed.
         - ***NOTE:*** *Each bullet should be simple and clear. Individual Instructions should **not** be high-level requests.*
   b. **Code Generation Step:**

i. **Code Generation** – A short snippet of code in ***Python 3.10*** to pull information from the dataset so that a more complex analysis can begin.
ii. **Code Execution Output** – The Output of Code Generation. If the code runs successfully, then this will be both your window and the Model's window into the contents of the dataset.

2. **Specific Data Analysis (SDA) – Now that we have access to the contents of the dataset, we can work towards providing a solution to the Prompt's request with certainty!**
   a. **Planning Step**:
      i. **Output** – High level plan to solve the data analysis problem.
      ii. **Coder Instructions** – Bulleted steps for how to move forward solving the Prompt's Request.
         - *NOTE: If the **Output** section is generated, then these instructions will need to align with the **Output** section.*

   b. **Code Generation Step**:
      i. **Code Generation** – Continue with another snippet of code in ***Python 3.10*** script that will build on the previously generated **Code Generation** steps to attempt to provide useful information for the final solution to the Prompt's request.
      ii. **Code Execution Output** – The Output of Code Generation. This will contain the Textual Output, Textual Error Log, and any images generated by the generated code.

   c. **Repeat 2a - 2b** – The Model will automatically switch from **SDA** when it considers the output generated by the **Code Generation Step** to be complete.

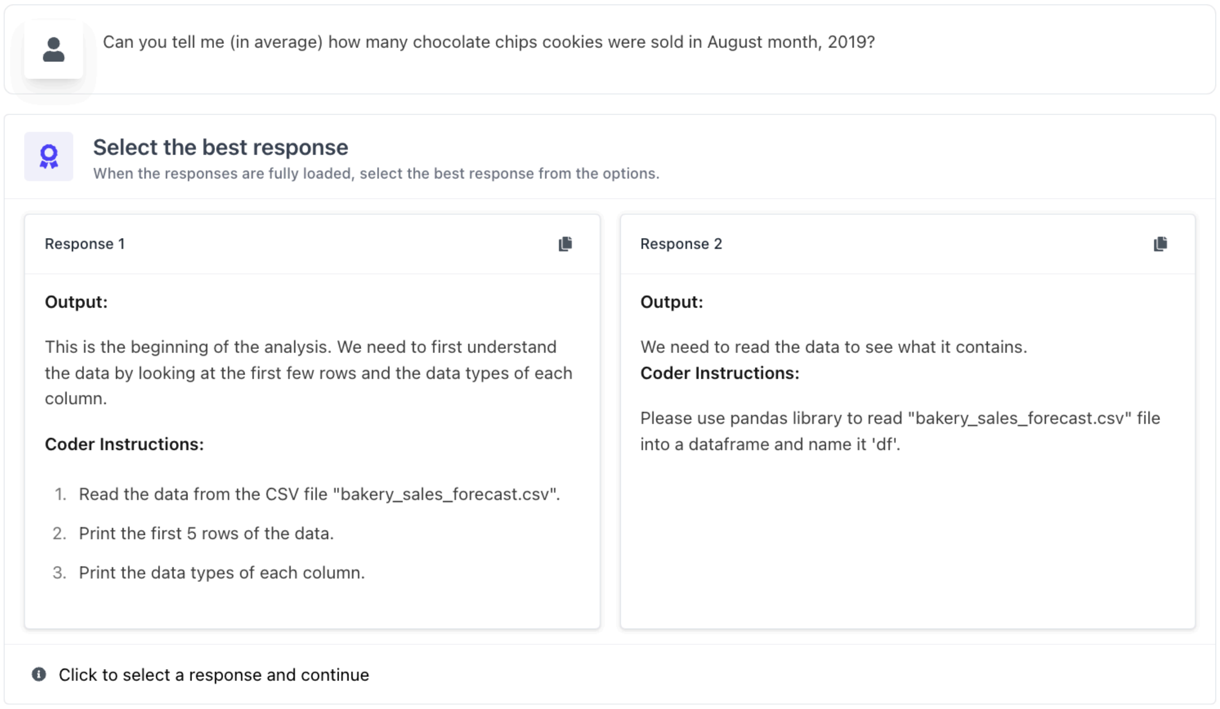
3. **Final Response – The Model's last, confident response that is displayed to the User as a Response to their Prompt.**
   a. **Natural Language Step**:
      i. **Final Output** – Describe the answer to the User. Provide any insights found during the course of the **Code Generation** & **Code Execution Output** Steps.

Instructions

Using the guidelines & details provided above, you should be ready to get started! These instructions below are going to be geared toward the specific of each rating & rewrite on the task as you move through it. If you'd like to propose feedback, we encourage you to ping <name> on the Remotasks or Expert Slack Community.

Model Response Selection

When presented with two Model Responses, you will determine which response is better using the definitions outlined in the ***Simplified Workflow***.



Can you tell me (in average) how many chocolate chips cookies were sold in August month, 2019?

Select the best response

When the responses are fully loaded, select the best response from the options.

Response 1

Output:

This is the beginning of the analysis. We need to first understand the data by looking at the first few rows and the data types of each column.

Coder Instructions:

1. Read the data from the CSV file "bakery_sales_forecast.csv".
2. Print the first 5 rows of the data.
3. Print the data types of each column.

Response 2

Output:

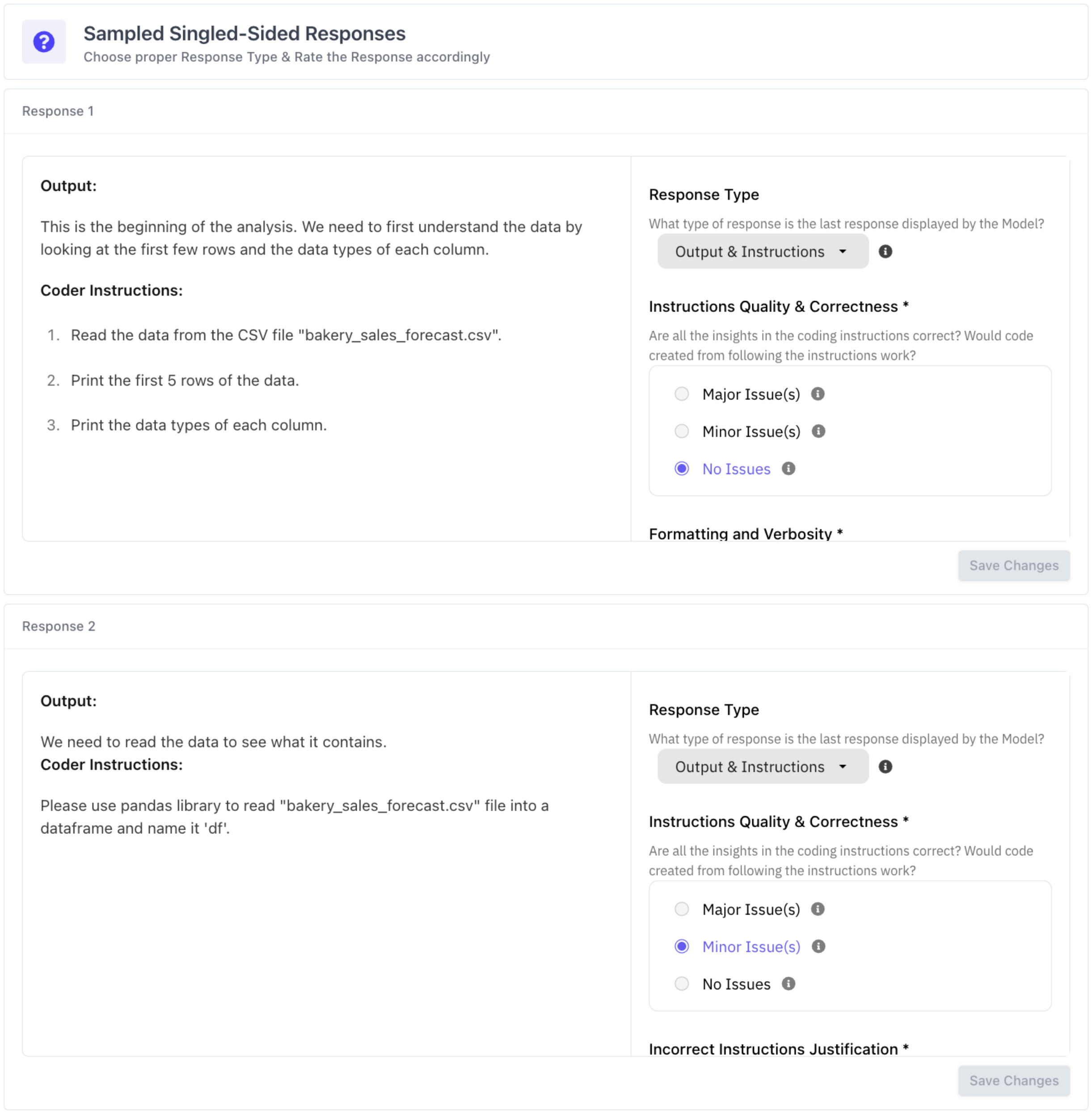
We need to read the data to see what it contains.
Coder Instructions:

Please use pandas library to read "bakery_sales_forecast.csv" file into a dataframe and name it 'df'.

Click to select a response and continue

Sampled Single-Sided Responses

Once you've selected a Response, both Responses need to be rated individually. You will need to make sure that the appropriate "Response Type" is selected, otherwise you'll end up rating the response incorrectly.



Sampled Singled-Sided Responses

Choose proper Response Type & Rate the Response accordingly

Response 1

Output:

This is the beginning of the analysis. We need to first understand the data by looking at the first few rows and the data types of each column.

Coder Instructions:

1. Read the data from the CSV file "bakery_sales_forecast.csv".
2. Print the first 5 rows of the data.
3. Print the data types of each column.

Response Type

What type of response is the last response displayed by the Model?

Output & Instructions

Instructions Quality & Correctness *

Are all the insights in the coding instructions correct? Would code created from following the instructions work?

Major Issue(s) Minor Issue(s) [✓ No Issues]

Formatting and Verbosity *

Save Changes

Response 2

Output:

We need to read the data to see what it contains.

Coder Instructions:

Please use pandas library to read "bakery_sales_forecast.csv" file into a dataframe and name it 'df'.

Response Type

What type of response is the last response displayed by the Model?

Output & Instructions

Instructions Quality & Correctness *

Are all the insights in the coding instructions correct? Would code created from following the instructions work?

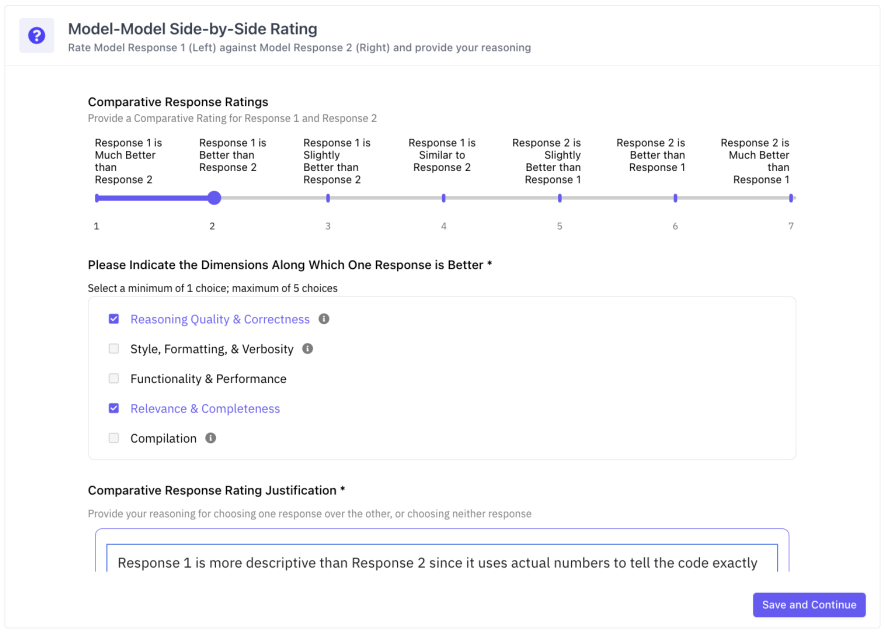
Major Issue(s) [✓ Minor Issue(s)] No Issues

Incorrect Instructions Justification *

Save Changes

Model SxS Rating

After answering rating questions for each model's response, you will compare them side by side. You will rate one response to be better and indicate the dimensions along which the response is better, and then write a justification explaining your reasoning. Be overly critical in your rating, and **ONLY** select that they are equal **if both model's truly are identical in their form/function and quality**.


Model-Model Side-by-Side Rating

Rate Model Response 1 (Left) against Model Response 2 (Right) and provide your reasoning

Comparative Response Ratings

Provide a Comparative Rating for Response 1 and Response 2

Response 1 is Much Better than Response 2	Response 1 is Better than Response 2	Response 1 is Slightly Better than Response 2	Response 1 is Similar to Response 2	Response 2 is Slightly Better than Response 1	Response 2 is Better than Response 1	Response 2 is Much Better than Response 1
1	[✓ 2]	3	4	5	6	7

Please Indicate the Dimensions Along Which One Response is Better *

Select a minimum of 1 choice; maximum of 5 choices

- [x] Reasoning Quality & Correctness
- [ ] Style, Formatting, & Verbosity
- [ ] Functionality & Performance
- [x] Relevance & Completeness
- [ ] Compilation

Comparative Response Rating Justification *

Provide your reasoning for choosing one response over the other, or choosing neither response

Response 1 is more descriptive than Response 2 since it uses actual numbers to tell the code exactly

Coding Responses

The **Code Generation Step** can come in many forms:

1. **Code** and **one of or both - Code Output** and/or **Image**
   a. *print* statements and other methods to display stdout will show **Code Output**
   b. If the prompt asks for a plot or graph, the most common method to display a plot is by using *matplotlib.pyplot.show* which will include an **Image** if compiled successfully.
2. **Code** and **Code Error**
   a. If there are compilation errors due to incorrect syntax or other mistakes, a **Code Error** will be displayed with the Code

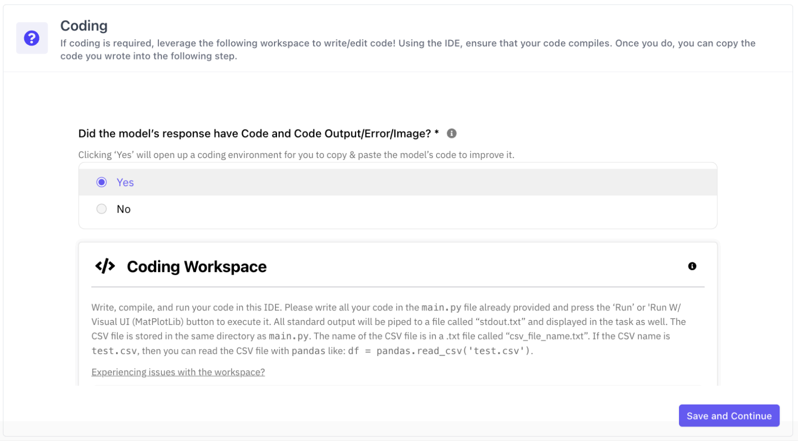
Your goal is to have meaningful outputs to **learn more about the data** to answer the prompt. These code outputs are what the model uses to understand the data and use facts instead of assumptions. When selecting the best code and outputs for the next step, consider these dimensions:

1. **Code must follow Coder Instructions in the previous model step. Code must not do more than what was asked in Coder Instructions.**
2. **Code should be a continuation of Code from all previous steps**
   a. This means that each Code step builds on the previous step, so each new Code step should be executed with the code from all previous steps
   b. This also means that we should avoid re importing previously imported packages and redefining previously defined variables

3. Code should follow proper styling and naming conventions
4. Code should compile successfully
5. Code should have useful outputs for us to learn more and get closer to answering the question

If there is a **Code Error** in both of the model's responses, you will need to fix the error when **rewriting the model's response**. You will be provided with a coding workspace where you can run and execute your code. Follow these steps to properly rewrite code:

1. Copy the Code from the selected model's response.
2. Open the Sphere Engine coding workspace. All code written in the workspace is persisted.
3. Paste the code in the workspace at the bottom. This is because of **Point #3, all Code steps are a continuation of previous code.**
4. Rewrite code to improve quality, adhering to the rating dimensions.
5. Replace the code in the response editor with the current step's rewritten code



Coding

If coding is required, leverage the following workspace to write/edit code! Using the IDE, ensure that your code compiles. Once you do, you can copy the code you wrote into the following step.

Did the model's response have Code and Code Output/Error/Image? *

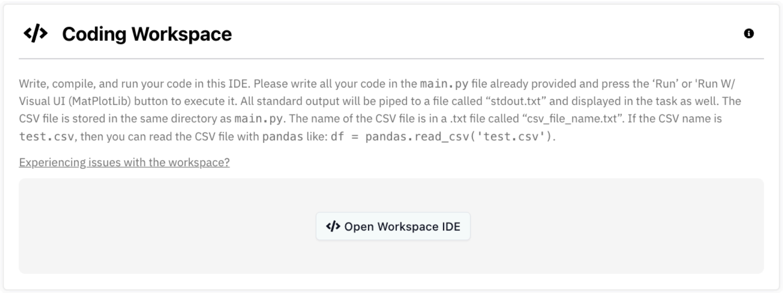
Clicking 'Yes' will open up a coding environment for you to copy & paste the model's code to improve it.

[✓ Yes]
No

Coding Workspace

Write, compile, and run your code in this IDE. Please write all your code in the `main.py` file already provided and press the 'Run' or 'Run W/ Visual UI (MatPlotLib) button to execute it. All standard output will be piped to a file called "stdout.txt" and displayed in the task as well. The CSV file is stored in the same directory as `main.py`. The name of the CSV file is in a .txt file called "csv_file_name.txt". If the CSV name is `test.csv`, then you can read the CSV file with pandas like: `df = pandas.read_csv('test.csv')`.

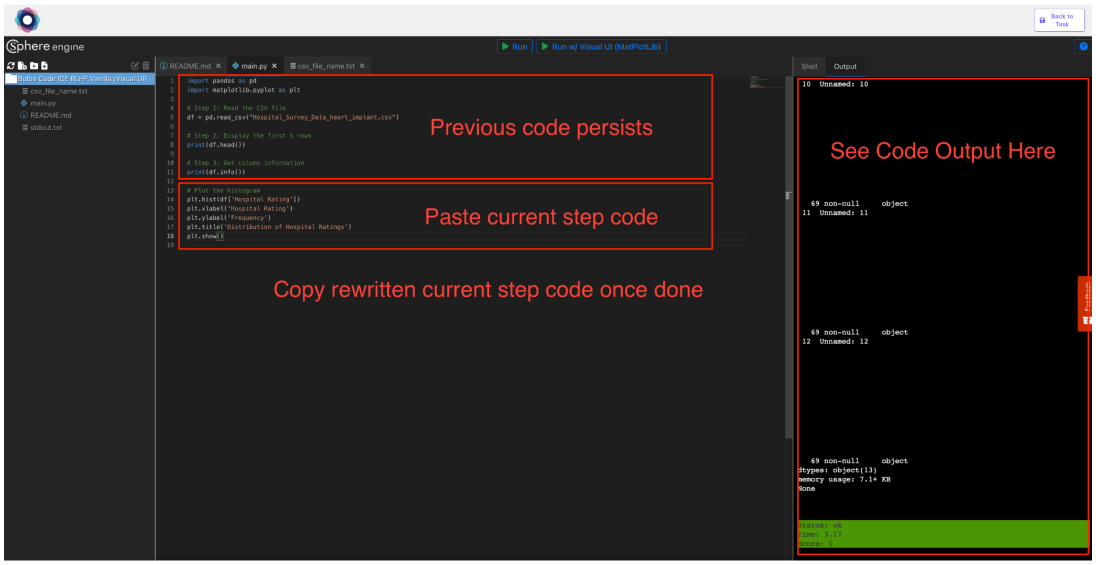
Experiencing issues with the workspace?



Coding Workspace

Write, compile, and run your code in this IDE. Please write all your code in the `main.py` file already provided and press the 'Run' or 'Run W/ Visual UI (MatPlotLib) button to execute it. All standard output will be piped to a file called "stdout.txt" and displayed in the task as well. The CSV file is stored in the same directory as `main.py`. The name of the CSV file is in a .txt file called "csv_file_name.txt". If the CSV name is `test.csv`, then you can read the CSV file with pandas like: `df = pandas.read_csv('test.csv')`.

Experiencing issues with the workspace?



```python
import pandas as pd
import matplotlib.pyplot as plt

# Step 1: Read the CSV file
df = pd.read_csv("Hospital_Survey_Data_heart_implant.csv")

# Step 2: Display the first 5 rows
print(df.head())

# Step 3: Get column information
print(df.info())

# Plot the histogram
plt.hist(df['Hospital Rating'])
plt.xlabel('Hospital Rating')
plt.ylabel('Frequency')
plt.title('Distribution of Hospital Ratings')
plt.show()
```

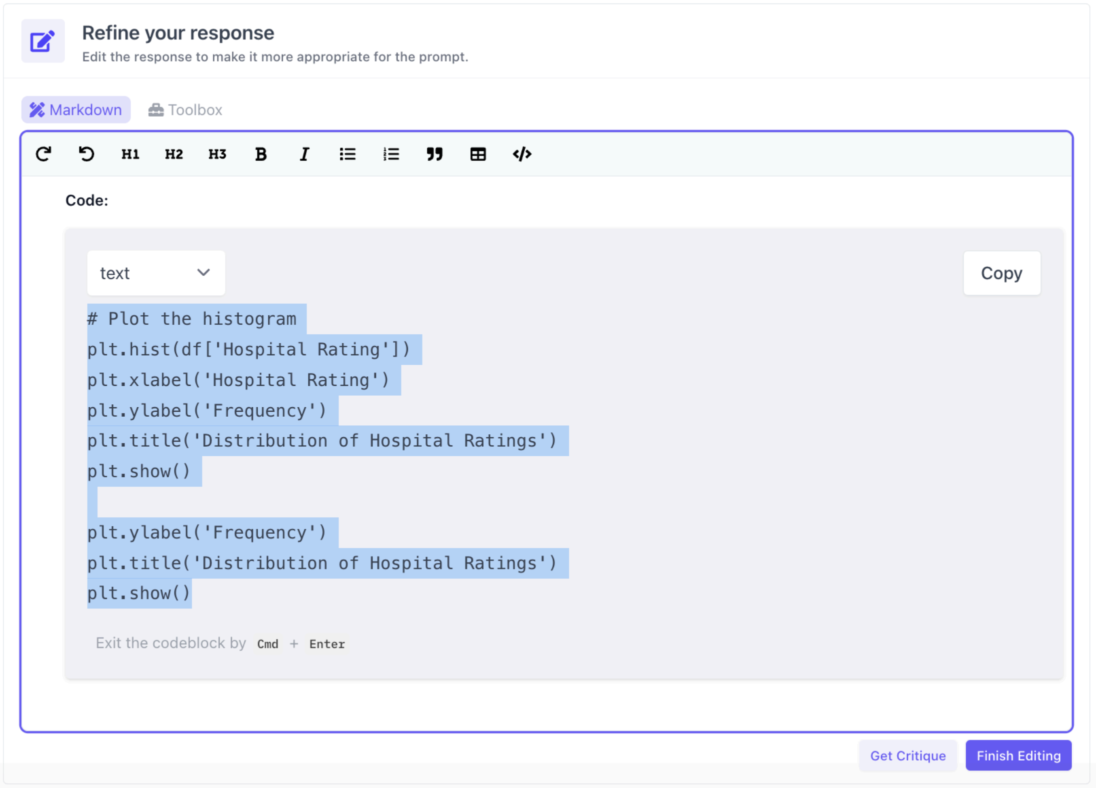
```
10  Unnamed: 10

  69 non-null     object
11  Unnamed: 11

  69 non-null     object
12  Unnamed: 12

  69 non-null     object
dtypes: object(13)
memory usage: 7.1+ KB
None

Status: ok
Time: 3.17
Score: 0
```


Refine your response
Edit the response to make it more appropriate for the prompt.
Markdown
Toolbox
H1
H2
H3
text
Copy
Exit the codeblock by Cmd + Enter
Get Critique
Finish Editing

Code:

```text
# Plot the histogram
plt.hist(df['Hospital Rating'])
plt.xlabel('Hospital Rating')
plt.ylabel('Frequency')
plt.title('Distribution of Hospital Ratings')
plt.show()

plt.ylabel('Frequency')
plt.title('Distribution of Hospital Ratings')
plt.show()
```

Rewriting Model Responses

At the end of each step, before the model transitions into the next step, you will have a chance to **rewrite the chosen model response**. The rewritten response that will be used to generate the next step of the model. Your aim is to refine the model's response for quality and accuracy, and to guide it correctly on the path towards the answer.

If the model is in a code step, access the coding workspace to execute and refine the code. First, copy the code that the model has produced in the current code step and paste it into the workspace. The workspace will include all the code from previous steps, and this should match the code, after rewriting, that has been sent to the model to generate the next step. Rewrite the current code to be the highest quality and produce useful output, and **paste only the current step's rewritten code into the model response editor.**

If the model is not in a code step, you will enhance the English outputs and instructions. Ensure clarity, avoid large code blocks, and guide the model in the correct direction to successfully answer the prompt. Base your assumptions and insights from ground truth in the code outputs from previous steps. **This means to not assume data values and types for columns in the dataset unless it has been seen in previous code outputs.** If you must make assumptions, clearly state them.