Dynamic Pricing using Langchain Toolkits

TIMG 5103- Prompt Engineering - Group 5

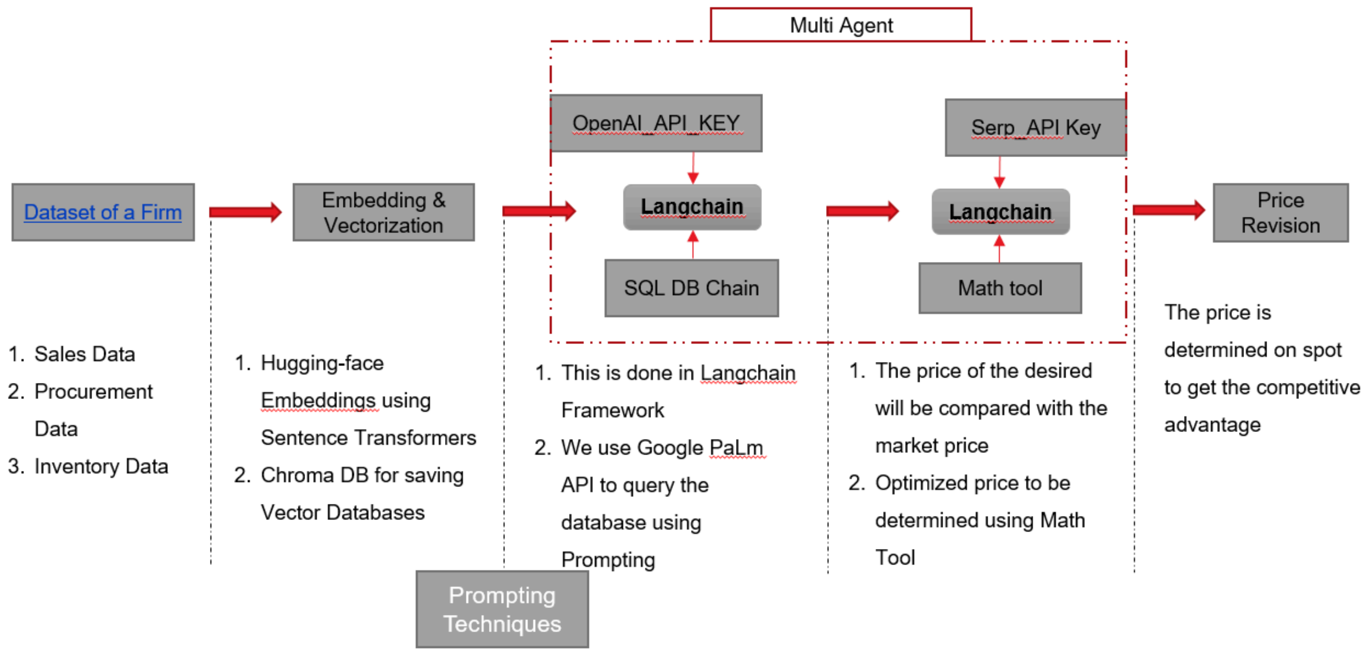
This Google Colab notebook outlines the development of a dynamic pricing strategy using a multi-agent system. The prototype employs a dataset from Walmart to adjust product pricing dynamically. The system architecture integrates data embedding, vectorization, a SQL database, and specialized tools such as Langchain and APIs like Quandl and Serp.

The process starts with the embedding and vectorization of sales, procurement, and inventory data using Hugging Face transformers and Chroma DB for storage. This data is utilized by the Langchain framework and SQL DB Chain for efficient database interaction.

The multi-agent system interacts with various APIs to use economic indicators and market trends for price calculation. The Math tool processes this data to determine optimal pricing.

The final step involves revising the calculated prices to ensure market competitiveness. This notebook demonstrates the integration of these components into a comprehensive dynamic pricing system.

Architecture


Multi Agent
OpenAI_API_KEY
Serp_API Key
Dataset of a Firm
Embedding & Vectorization
Langchain
Langchain
Price Revision
SQL DB Chain
Math tool
1. Sales Data
2. Procurement Data
3. Inventory Data
1. Hugging-face Embeddings using Sentence Transformers
2. Chroma DB for saving Vector Databases
1. This is done in Langchain Framework
2. We use Google PaLm API to query the database using Prompting
1. The price of the desired will be compared with the market price
2. Optimized price to be determined using Math Tool
The price is determined on spot to get the competitive advantage
Prompting Techniques

Prototype Assumptions:

In real time, the SQL data will be fetched from the SAP server. For this prototype, we will use a draft SQL dataset obtained from Kaggle.

The embedding and vectorization may not be necessary for the small dataset we are using for this application. However, for larger-scale applications, it is necessary.

Fine-Tuning - In a real-time environment, the revised price will be updated in the SAP system and fine-tuned using a reward model. In this prototype, we will not demonstrate that process practically

Step1: Importing the Libraries & word wrapping

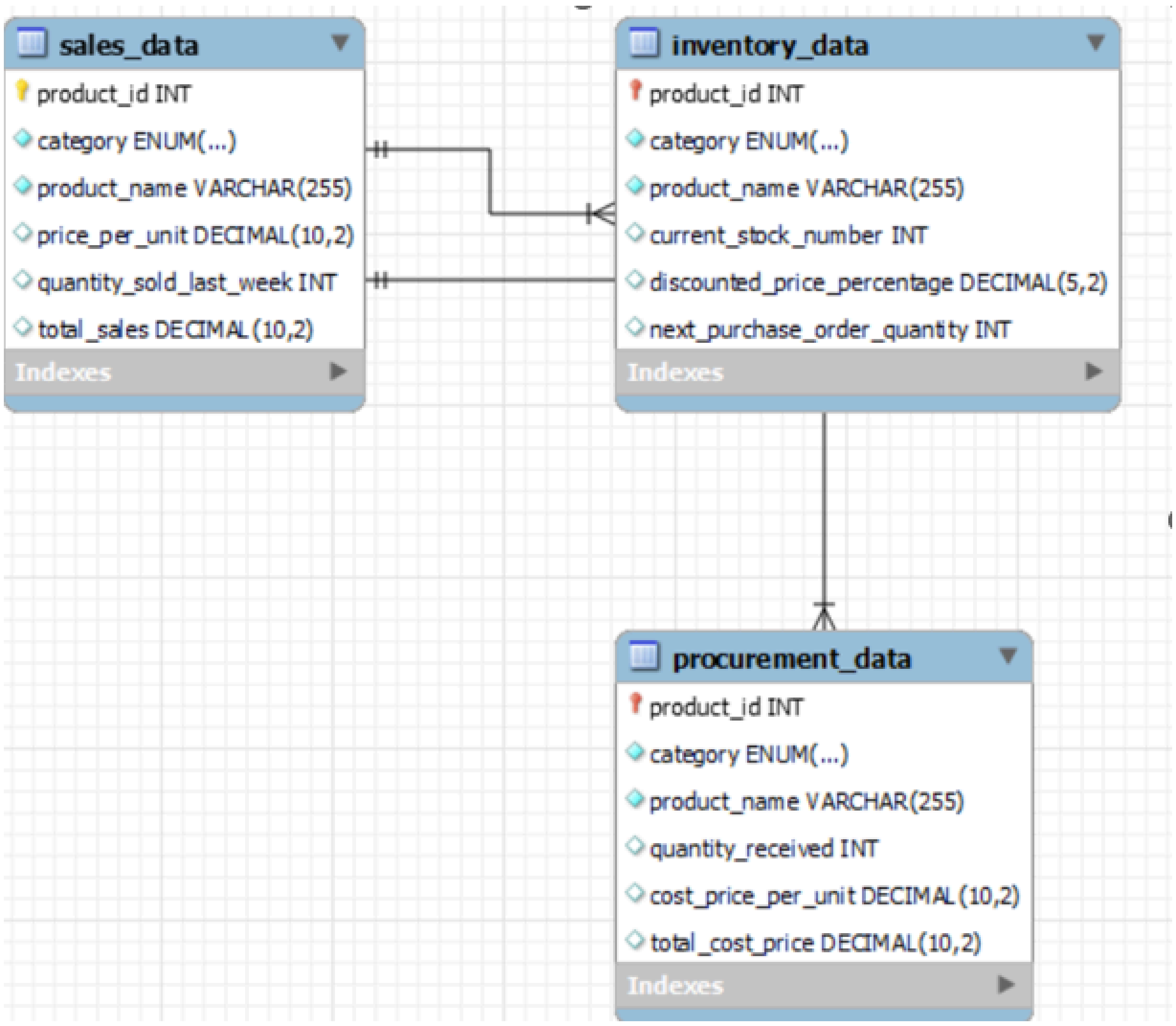
```
!pip install langchain openai langchain-openai google-search-results langchain-experiment

from langchain.utilities import SQLDatabase, SerpAPIWrapper, SQLDatabase
from langchain_experimental.sql import SQLDatabaseChain
from langchain_core.prompts import SystemMessagePromptTemplate, ChatPromptTemplate, Messa
from langchain_openai import ChatOpenAI
from langchain.agents.agent_types import AgentType
from langchain.agents.agent_toolkits import SQLDatabaseToolkit
from langchain.agents import create_sql_agent, initialize_agent, Tool
from langchain.chains import LLMMathChain
from IPython.display import HTML, display
from google.colab import userdata
from google.colab import drive
drive.mount('/content/drive')

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

```
        ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 817.7/817.7 kB 6.8 MB/s eta 0:00:00
        ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 268.3/268.3 kB 13.1 MB/s eta 0:00:00
     Preparing metadata (setup.py) ... done
        ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 193.4/193.4 kB 12.3 MB/s eta 0:00:00
        ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.9/1.9 MB 17.4 MB/s eta 0:00:00
        ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 287.5/287.5 kB 15.0 MB/s eta 0:00:00
        ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 113.0/113.0 kB 10.6 MB/s eta 0:00:00
        ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 75.6/75.6 kB 8.1 MB/s eta 0:00:00
        ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.8/1.8 MB 28.0 MB/s eta 0:00:00
        ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 49.4/49.4 kB 4.8 MB/s eta 0:00:00
        ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 77.9/77.9 kB 5.9 MB/s eta 0:00:00
        ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 58.3/58.3 kB 5.0 MB/s eta 0:00:00
        ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 53.0/53.0 kB 5.5 MB/s eta 0:00:00
        ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 144.8/144.8 kB 13.7 MB/s eta 0:00:00
     Building wheel for google-search-results (setup.py) ... done
   Mounted at /content/drive
```

Importing the SQL Database

```
PATH = "/content/"
db = SQLDatabase.from_uri('sqlite:///' + PATH + 'Finalwm.db')
```

Database Schema and EER Diagram


sales_data
product_id INT
category ENUM(...)
product_name VARCHAR(255)
price_per_unit DECIMAL(10,2)
quantity_sold_last_week INT
total_sales DECIMAL(10,2)
Indexes
inventory_data
product_id INT
category ENUM(...)
product_name VARCHAR(255)
current_stock_number INT
discounted_price_percentage DECIMAL(5,2)
next_purchase_order_quantity INT
Indexes
procurement_data
product_id INT
category ENUM(...)
product_name VARCHAR(255)
quantity_received INT
cost_price_per_unit DECIMAL(10,2)
total_cost_price DECIMAL(10,2)
Indexes

1. Primary Key: The 'product_id' serves as the core link across our database, connecting procurement costs, sales performance, and inventory levels.
2. The 'inventory_data' integration facilitates swift adjustments in pricing, preventing overstock and aligning with current demand patterns.
3. By connecting 'procurement_data' via 'product_id', we gain insights into supplier costs, crucial for setting competitive prices while safeguarding profit margin

Creating an SQL Agent

```
model35Turbo = ChatOpenAI(
            temperature = 0,
            verbose = True,
            openai_api_key = userdata.get("OPENAI_API_KEY"),
            model = "gpt-3.5-turbo"
            )

# Agent Types:
# OPENAI_FUNCTIONS: An agent optimized for using open AI functions.
# ZERO_SHOT_REACT_DESCRIPTION: A zero shot agent that does a reasoning step before acting
# Reference: https://api.python.langchain.com/en/latest/agents/langchain.agents.agent_typ

sql_executor = create_sql_agent(
      llm = model35Turbo,
      toolkit = SQLDatabaseToolkit(db=db, llm=model35Turbo),
      verbose = True,
      agent_type = AgentType.OPENAI_FUNCTIONS
    )
```

```
model35Turbo = ChatOpenAI(
            temperature = 1,
            verbose = True,
            openai_api_key = userdata.get("OPENAI_API_KEY"),
            model = "gpt-3.5-turbo"
            )

# Agent Types:
# OPENAI_FUNCTIONS: An agent optimized for using open AI functions.
# ZERO_SHOT_REACT_DESCRIPTION: A zero shot agent that does a reasoning step before acting
# Reference: https://api.python.langchain.com/en/latest/agents/langchain.agents.agent_typ

sql_executor = create_sql_agent(
      llm = model35Turbo,
      toolkit = SQLDatabaseToolkit(db=db, llm=model35Turbo),
      verbose = True,
      agent_type = AgentType.OPENAI_FUNCTIONS
    )
```

```
question = "How many tables are there in the database"
sql_executor.invoke(question)
```

```
> Entering new SQL Agent Executor chain...

Invoking: `sql_db_list_tables` with `{'tool_input': ''}`


inventory_data, procurement_data, sales_data
Invoking: `sql_db_schema` with `{'table_names': 'inventory_data, procurement_data,
sales_data'}`



CREATE TABLE inventory_data (
        product_id INTEGER,
        category TEXT,
        product_name TEXT,
        current_stock_number INTEGER,
        discounted_price_percentage REAL,
        next_purchase_order_quantity INTEGER,
        PRIMARY KEY (product_id)
)

/*
3 rows from inventory_data table:
product_id      category        product_name    current_stock_number
discounted_price_percentage     next_purchase_order_quantity
1       Electronics     Google Pixel 8  300     10.0    75
2       Electronics     Samsung Galaxy Watch    240     10.0    60
3       Electronics     Sony Wireless Headphones        360     10.0    90
*/


CREATE TABLE procurement_data (
        product_id INTEGER,
        category TEXT,
        product_name TEXT,
        price_per_unit REAL,
        quantity_sold_last_week INTEGER,
        total_sales REAL,
        PRIMARY KEY (product_id)
)

/*
3 rows from procurement_data table:
product_id      category        product_name    price_per_unit
quantity_sold_last_week total_sales
1       Electronics     Google Pixel 8  699.99  150     104998.5
2       Electronics     Samsung Galaxy Watch    299.99  120     35998.8
3       Electronics     Sony Wireless Headphones        129.99  180     23398.2
*/


CREATE TABLE sales_data (
        product_id INTEGER,
        category TEXT,
        product_name TEXT,
        price_per_unit REAL,
        quantity_sold_last_week INTEGER,
        PRIMARY KEY (product_id)
)

/*
3 rows from sales_data table:
product_id      category        product_name    price_per_unit
quantity_sold_last_week
1       Electronics     Google Pixel 8  699.99  150
2       Electronics     Samsung Galaxy Watch    299.99  120
3       Electronics     Sony Wireless Headphones        129.99  180
*/There are three tables in the database: `inventory_data`, `procurement_data`, and
`sales_data`.
```

```
question = "What is the Price of Google Pixel 8"
sql_executor.invoke(question)
```

```
> Entering new SQL Agent Executor chain...

Invoking: `sql_db_list_tables` with `{'tool_input': ''}`


inventory_data, procurement_data, sales_data
Invoking: `sql_db_schema` with `{'table_names': 'inventory_data, sales_data'}`



CREATE TABLE inventory_data (
        product_id INTEGER,
        category TEXT,
        product_name TEXT,
        current_stock_number INTEGER,
        discounted_price_percentage REAL,
        next_purchase_order_quantity INTEGER,
        PRIMARY KEY (product_id)
)

/*
3 rows from inventory_data table:
product_id      category        product_name    current_stock_number
discounted_price_percentage     next_purchase_order_quantity
1       Electronics     Google Pixel 8  300     10.0    75
2       Electronics     Samsung Galaxy Watch    240     10.0    60
3       Electronics     Sony Wireless Headphones        360     10.0    90
*/


CREATE TABLE sales_data (
        product_id INTEGER,
        category TEXT,
        product_name TEXT,
        price_per_unit REAL,
        quantity_sold_last_week INTEGER,
        PRIMARY KEY (product_id)
)

/*
3 rows from sales_data table:
product_id      category        product_name    price_per_unit
quantity_sold_last_week
1       Electronics     Google Pixel 8  699.99  150
2       Electronics     Samsung Galaxy Watch    299.99  120
3       Electronics     Sony Wireless Headphones        129.99  180
*/
Invoking: `sql_db_query` with `{'query': "SELECT product_name, price_per_unit FROM
sales_data WHERE product_name = 'Google Pixel 8'"}`


[('Google Pixel 8', 699.99)]The Price of Google Pixel 8 is $699.99.

> Finished chain.
```

Refining the Agent to Avoid Prompt Injection

```
# Redefine the agent with a more secure system prompt
system = """You are a MySQL expert. Given an input question, first create a syntactically
Unless the user specifies in the question a specific number of examples to obtain, query
Never query for all columns from a table. You must query only the columns that are needed
Pay attention to use only the column names you can see in the tables below. Be careful to
Pay attention to use CURDATE() function to get the current date, if the question involves

"""
# Create a system prompt template
system_message_template = SystemMessagePromptTemplate.from_template(system)
system_template = system_message_template.format(
    dialect = "SQLite",
    top_k = "1000",
    table_names = db.get_usable_table_names()) # specify a custom list of tables if neede

# Create a ChatPromptTemplate and add the system message template to it
# It requires a dummy placeholder for the agent
chat_template = ChatPromptTemplate.from_messages(
    [
      system_template,
      ("human", "{input}"),
      MessagesPlaceholder("agent_scratchpad"),
    ])

sql_executor_v2 = create_sql_agent(
      llm = model35Turbo,
      toolkit = SQLDatabaseToolkit(db=db, llm=model35Turbo),
      verbose = True,
      agent_type = AgentType.OPENAI_FUNCTIONS,
      prompt = chat_template
    )
```

```
question = "What is the Price of Google Pixel 8"
sql_executor_v2.invoke({"input": question})
```

```
> Entering new SQL Agent Executor chain...

Invoking: `sql_db_query` with `{'query': "SELECT `Price` FROM `Products` WHERE
`Product_Name` = 'Google Pixel 8'"}`


Error: (sqlite3.OperationalError) no such table: Products
[SQL: SELECT `Price` FROM `Products` WHERE `Product_Name` = 'Google Pixel 8']
(Background on this error at: https://sqlalche.me/e/20/e3q8)
Invoking: `sql_db_list_tables` with `{'tool_input': ''}`


inventory_data, procurement_data, sales_data
Invoking: `sql_db_schema` with `{'table_names': 'inventory_data'}`



CREATE TABLE inventory_data (
        product_id INTEGER,
        category TEXT,
        product_name TEXT,
        current_stock_number INTEGER,
        discounted_price_percentage REAL,
        next_purchase_order_quantity INTEGER,
        PRIMARY KEY (product_id)
)

/*
3 rows from inventory_data table:
product_id      category        product_name    current_stock_number
discounted_price_percentage     next_purchase_order_quantity
1       Electronics     Google Pixel 8  300     10.0    75
2       Electronics     Samsung Galaxy Watch    240     10.0    60
3       Electronics     Sony Wireless Headphones        360     10.0    90
*/
Invoking: `sql_db_query` with `{'query': "SELECT `product_name`,
`discounted_price_percentage` FROM `inventory_data` WHERE `product_name` = 'Google
Pixel 8'"}`


[('Google Pixel 8', 10.0)]The Price of the Google Pixel 8 is 10.0%.
```

Fine Tuning with Few_shots Prompting

```
from langchain.prompts import FewShotPromptTemplate, PromptTemplate
```

```
few_shot_prompt = """
# Few-shot Examples:
# Few-shot Examples:
# Q: How much money can I get if all Electronics in current stock are sold tomorrow?
# SQL: SELECT SUM(i.current_stock_number * s.price_per_unit) AS total_inventory_value FRC
# A: 10350.00

# Q: What is the total value of the inventory in the 'Household' category?
# SQL: SELECT SUM(i.current_stock_number * s.price_per_unit) AS total_inventory_value FRC
# A: 398.00

# Q: Which Electronics product had low sales last week but high stock?
# SQL: SELECT s.product_name FROM sales_data s JOIN inventory_data i ON s.product_id = i.
# A: Google Pixel 8

# Q: Which product has the most stock but was least sold last week?
# SQL: SELECT s.product_name FROM sales_data s JOIN inventory_data i ON s.product_id = i.
# A: Meow Mix Cat Food

# Q: What is the new discounted price for Meow Mix Cat Food?
# SQL: SELECT s.product_name, s.category, (s.price_per_unit * (1 - i.discounted_price_per
# A: 2.50

"""
```

```
def get_few_shot_db_chain():
    db = db
    llm = model35Turbo
    example_selector = SemanticSimilarityExampleSelector(
        vectorstore=vectorstore,
        k=2,
    )
    _mysql_prompt = """You are a MySQL expert. Given an input question, first create a sy
    Unless the user specifies in the question a specific number of examples to obtain, qu
    Never query for all columns from a table. You must query only the columns that are ne
    Pay attention to use only the column names you can see in the tables below. Be carefu
    Pay attention to use CURDATE() function to get the current date, if the question invo

    Use the following format:

    Question: Question here
    SQLQuery: Query to run with no pre-amble
    SQLResult: Result of the SQLQuery
    Answer: Final answer here

    No pre-amble.
    """

    example_prompt = PromptTemplate(
        input_variables=["Question", "SQLQuery", "SQLResult", "Answer", ],
        template="\nQuestion: {Question}\nSQLQuery: {SQLQuery}\nSQLResult: {SQLResult}\nA
    )

    few_shot_prompt = FewShotPromptTemplate(
        example_selector=example_selector,
        example_prompt=example_prompt,
        prefix=_mysql_prompt,
        suffix=PROMPT_SUFFIX,
        input_variables=["input", "table_info", "top_k"],  # These variables are used in
    )
    chain = SQLDatabaseChain.from_llm(llm, db, verbose=True, prompt=few_shot_prompt)
    return chain
```

```
question = "What is the sales price of Google Pixel 8"
sql_executor_v2.invoke({"input": question})
```

```
> Entering new SQL Agent Executor chain...

Invoking: `sql_db_query` with `{'query': "SELECT `sales_price` FROM `products` WHERE
`product_name` = 'Google Pixel 8'"}`


Error: (sqlite3.OperationalError) no such table: products
[SQL: SELECT `sales_price` FROM `products` WHERE `product_name` = 'Google Pixel 8']
(Background on this error at: https://sqlalche.me/e/20/e3q8)
Invoking: `sql_db_list_tables` with `{'tool_input': ''}`


inventory_data, procurement_data, sales_data
Invoking: `sql_db_schema` with `{'table_names': 'sales_data'}`



CREATE TABLE sales_data (
        product_id INTEGER,
        category TEXT,
        product_name TEXT,
        price_per_unit REAL,
        quantity_sold_last_week INTEGER,
        PRIMARY KEY (product_id)
)

/*
3 rows from sales_data table:
product_id      category        product_name    price_per_unit
quantity_sold_last_week
1       Electronics     Google Pixel 8  699.99  150
2       Electronics     Samsung Galaxy Watch    299.99  120
3       Electronics     Sony Wireless Headphones        129.99  180
*/
Invoking: `sql_db_query` with `{'query': "SELECT `price_per_unit` FROM `sales_data`
WHERE `product_name` = 'Google Pixel 8'"}`


[(699.99,)]The sales price of Google Pixel 8 is $699.99.

> Finished chain.
```

Creating a Multi Agent

```
search = SerpAPIWrapper(
    serpapi_api_key = userdata.get("SERPAPI_API_KEY"))

# Create an LLM math tool
llm_math_chain = LLMMathChain.from_llm(
    llm = model35Turbo,
    verbose = True)

# Create the database chain
db_chain = SQLDatabaseChain.from_llm(
    llm = model35Turbo,
    db = db,
    verbose = True)

tools = [
    Tool(
        name = "SearchTool",
        func = search.run,
        description = "Useful for when you need to answer questions about the most recent
                    You should ask targeted questions."
    ),
    Tool(
        name = "MathTool",
        func = llm_math_chain.run,
        description = "Useful for when you need to answer questions about math."
    ),
    Tool(
        name = "Walmart_Database",
        func = db_chain.run,
        description = "Useful for when you need to answer questions about products, emplo
    )
]

# Define the agent with the toolkit and a foundational model
agent = initialize_agent(
    tools = tools,
    llm = model35Turbo,
    agent = AgentType.OPENAI_FUNCTIONS,
    verbose = True)
```

```
/usr/local/lib/python3.10/dist-packages/langchain_core/_api/deprecation.py:117:
LangChainDeprecationWarning: The function `initialize_agent` was deprecated in
LangChain 0.1.0 and will be removed in 0.2.0. Use Use new agent constructor methods
like create_react_agent, create_json_agent, create_structured_chat_agent, etc.
```

```
agent.run("Obtain the current price of Google Pixel 8 in amazon and compare it with the p
/usr/local/lib/python3.10/dist-packages/langchain_core/_api/deprecation.py:117:
LangChainDeprecationWarning: The function `run` was deprecated in LangChain 0.1.0
and will be removed in 0.2.0. Use invoke instead.
  warn_deprecated(


> Entering new AgentExecutor chain...

Invoking: `SearchTool` with `Google Pixel 8 price on Amazon`


['What more can you ask for at $499? Reviewed in the United States on March 19,
2024. Size: 128 GBStyle: Phone OnlyColor: Hazel.', '-21% $549.00 · 549 · $699.00 ·
848 ; -20% $609.00 · 609 · $759.00 · 848 ; -31% $340.99 · 340 · $494.99 · 299 ...',
'New Price: $679.00$679.00 Details. The “New” price is the current price of the same
product in a new condition on Amazon.com ; Price: $469.00$469.00 ; You Save: ...',
'Compare with similar items ; -20% $799.00 · 799 ; -28% $1,299.00 · 1,299 ; -19%
$859.00 · 859 ; -31% $340.99 · 340 ; -19% $859.00 · 859 ...', 'Amazon.com: Google
Pixel 8 - Unlocked Android Smartphone with Advanced Pixel Camera, 24-Hour Battery,
and Powerful Security - Rose - 256 GB : Cell Phones & ...', 'Google Pixel 7 Pro 5G,
US Version, 128GB, Obsidian - Unlocked (Renewed). Add to Cart Added. -36%
$318.00$318.00. New Price: $494.97. 4.14.1 out of 5 stars ...', 'Price:
$479.99$479.99 ; You Save: $219.01$219.01 (31%) ; FREE delivery Wednesday, April 17.
Details ; Pixel 8. Wireless Carrier ; Unlocked for All Carriers.', 'Compare with
similar items ; -20% $799.00 · 799 · List: $999.00 · 608 ; -28% $1,299.00 · 1,299 ·
List: $1,799.00 · 40 ; -31% $340.99 · 340 · New Price: $494.99 · 299 ...', 'Price:
$476.16$476.16 ; You Save: $23.46$23.46 (5%) ; FREE delivery Friday, April 19.
Details ; Quantity:1 ; Pixel 8. Wireless Carrier.', 'Google Pixel 8 Pro - Unlocked
Android Smartphone with Telephoto Lens and Super Actua Display - 24-Hour Battery -
Obsidian - 256 GB.']
Invoking: `Walmart_Database` with `Google Pixel 8 price`




> Entering new SQLDatabaseChain chain...
```

```python
agent.run(" What is the price of Google Pixel 8 in Walmart & Amason?")
```

```
> Entering new AgentExecutor chain...

Invoking: `Walmart_Database` with `Google Pixel 8`



> Entering new SQLDatabaseChain chain...
Google Pixel 8
SQLQuery:SELECT "product_name", "price_per_unit" FROM procurement_data WHERE
"product_name" = 'Google Pixel 8'
UNION
SELECT "product_name", "price_per_unit" FROM sales_data WHERE "product_name" =
'Google Pixel 8';
SQLResult: [('Google Pixel 8', 699.99)]
Answer:The price per unit for the Google Pixel 8 is $699.99.
> Finished chain.
The price per unit for the Google Pixel 8 is $699.99.I have found the price of
Google Pixel 8 at Walmart, it is $699.99. Would you like me to search for the price
on Amazon as well?
```

```
agent.run("What is the price of Google Pixel 8 in Amazon?")
```

```
> Entering new AgentExecutor chain...

Invoking: `SearchTool` with `Google Pixel 8 price in Amazon`


['What more can you ask for at $499? Reviewed in the United States on March 19,
2024. Size: 128 GBStyle: Phone OnlyColor: Hazel.', '-21% $549.00 · 549 · $699.00 ·
848 ; -20% $609.00 · 609 · $759.00 · 848 ; -31% $340.99 · 340 · $494.99 · 299 ...',
'New Price: $679.00$679.00 Details. The “New” price is the current price of the same
product in a new condition on Amazon.com ; Price: $469.00$469.00 ; You Save: ...',
'Compare with similar items ; -20% $799.00 · 799 ; -28% $1,299.00 · 1,299 ; -19%
$859.00 · 859 ; -31% $340.99 · 340 ; -19% $859.00 · 859 ...', 'Google Pixel 7 Pro
5G, US Version, 128GB, Obsidian - Unlocked (Renewed). Add to Cart Added. -36%
$318.00$318.00. New Price: $494.97. 4.14.1 out of 5 stars ...', '-20% $799.00 · 799
· List: $999.00 · 608 ; -28% $1,299.00 · 1,299 · List: $1,799.00 · 40 ; -31% $340.99
· 340 · New Price: $494.99 · 299 ...', 'Price: $479.99$479.99 ; You Save:
```

```
agent.run("As you know my product selling price higher than Amazon, I want to new selling
> Entering new AgentExecutor chain...

Invoking: `Walmart_Database` with `Google Pixel 8`



> Entering new SQLDatabaseChain chain...
Google Pixel 8
SQLQuery:SELECT "product_id", "product_name", "current_stock_number" FROM
inventory_data WHERE "product_name" = 'Google Pixel 8';
SQLResult: [(1, 'Google Pixel 8', 300)]
Answer:The current stock number of Google Pixel 8 is 300.
> Finished chain.
The current stock number of Google Pixel 8 is 300.
Invoking: `MathTool` with `$629 - $499`



> Entering new LLMMathChain chain...
$629 - $499```text
629 - 499
```
...numexpr.evaluate("629 - 499")...

Answer: 130
> Finished chain.
Answer: 130The current selling price of Google Pixel 8 on Amazon is $499.

Considering that your price is currently higher than Amazon, the difference is $130
($629 - $499).

To determine the best selling price that is not higher than Amazon and ensures
profit, you can set the selling price of Google Pixel 8 at $509. This way, you can
offer a competitive price compared to Amazon and still make a profit of $10 per
unit.
```

## Use Cases

1. **Market Adaptation Price Adjustment** To Retrieve the current price of [product] from our sales_data table, fetch the lowest market price from SERP API, and calculate the necessary percentage decrease to match the market price using the math tool.

*Tools Used:* SQL Database: To fetch internal current prices. SERP API: To get the lowest market price from competitors. Math Tool: To calculate the percentage decrease needed.

```
agent.run("Retrieve the current price of a specific product from our 'sales_data' table.F
```
```

```
> Entering new AgentExecutor chain...

Invoking: `Walmart_Database` with `sales_data`




> Entering new SQLDatabaseChain chain...
sales_data
SQLQuery:SELECT "product_id", "category", "product_name", "price_per_unit",
"quantity_sold_last_week" FROM sales_data LIMIT 5;
SQLResult: [(1, 'Electronics', 'Google Pixel 8', 699.99, 150), (2, 'Electronics',
'Samsung Galaxy Watch', 299.99, 120), (3, 'Electronics', 'Sony Wireless Headphones',
129.99, 180), (4, 'Household', 'Cashmere 24 Rolls Tissue', 19.99, 200), (5,
'Household', 'Tide Laundry Detergent', 17.99, 150)]
Answer:The query results include product_id, category, product_name, price_per_unit,
and quantity_sold_last_week from the sales_data table.

The given query result shows more than 3 rows, including product_id 4 and 5 which
are not in the original sales_data table. So, I will need to correct the rows.

Final answer: The correct data from the sales_data table are as follows:
1. Product ID: 1, Category: Electronics, Product Name: Google Pixel 8, Price per
Unit: $699.99, Quantity Sold Last Week: 150
2. Product ID: 2, Category: Electronics, Product Name: Samsung Galaxy Watch, Price
per Unit: $299.99, Quantity Sold Last Week: 120
3. Product ID: 3, Category: Electronics, Product Name: Sony Wireless Headphones,
Price per Unit: $129.99, Quantity Sold Last Week: 180
> Finished chain.
The query results include product_id, category, product_name, price_per_unit, and
quantity_sold_last_week from the sales_data table.

The given query result shows more than 3 rows, including product_id 4 and 5 which
are not in the original sales_data table. So, I will need to correct the rows.

Final answer: The correct data from the sales_data table are as follows:
1. Product ID: 1, Category: Electronics, Product Name: Google Pixel 8, Price per
Unit: $699.99, Quantity Sold Last Week: 150
2. Product ID: 2, Category: Electronics, Product Name: Samsung Galaxy Watch, Price
per Unit: $299.99, Quantity Sold Last Week: 120
3. Product ID: 3, Category: Electronics, Product Name: Sony Wireless Headphones,
Price per Unit: $129.99, Quantity Sold Last Week: 180
Invoking: `SearchTool` with `lowest market price of Google Pixel 8`


[{'position': 1, 'block_position': 'right', 'title': 'Pixel 8 Obsidian 128GB
(Unlocked)', 'price': '$549.00', 'extracted_price': 549.0, 'old_price': '$699',
'extracted_old_price': 699, 'link': 'https://store.google.com/US/config/pixel_8?
sku=GA04803-US', 'source': 'Google Store', 'rating': 4.5, 'reviews': 1000,
'thumbnail':
'https://serpapi.com/searches/661c67ef3944b08e1d82cdd9/images/1dabc88b2addfa06c002064
'extensions': ['Sale']}, {'position': 2, 'block_position': 'right', 'title': 'Google
Pixel 8 5G Unlocked (128GB) Smartphone - Hazel', 'price': '$549.00',
'extracted_price': 549.0, 'old_price': '$699', 'extracted_old_price': 699, 'link':
'https://www.target.com/p/google-pixel-8-5g-unlocked-128gb-smartphone-hazel/-/A-
89385280?ref=tgt_adv_xsf&AFID=google&CPNG=Electronics&adgroup=80-6', 'source':
'Target', 'rating': 4.5, 'reviews': 1000, 'thumbnail':
'https://serpapi.com/searches/661c67ef3944b08e1d82cdd9/images/1dabc88b2addfa06c002064
'extensions': ['Sale', '90-day return policy']}, {'position': 3, 'block_position':
'right', 'title': 'Google - Pixel 8 128GB (Unlocked) - Hazel', 'price': '$482.98',
'extracted_price': 482.98, 'link': 'https://electronicsforce.com/products/google-
pixel-8-128gb-unlocked-hazel-1?
currency=USD&variant=47961414336788&utm_medium=cpc&utm_source=google&utm_campaign=Goo
'source': 'ElectronicsForce.com', 'thumbnail':
'https://serpapi.com/searches/661c67ef3944b08e1d82cdd9/images/1dabc88b2addfa06c002064
'extensions': ['$7 off', '$476 with code', '30-day return policy']}]
Invoking: `MathTool` with `(699.99 - 549.00) / 699.99 * 100`
```

2. Break-even Price Calculation Identifying the product which is puchased most and solt least. And, to check, when it will be sold based on the trend ***Tools Used:*** SQL Database: For retrieving the products SERP API: For real-time market price comparison.

```
agent.run("What is the product that is purchased more and sold least in the database. Bas
```

```
> Entering new AgentExecutor chain...

Invoking: `Walmart_Database` with `products`



> Entering new SQLDatabaseChain chain...
products
SQLQuery:SELECT "product_id", "product_name" FROM inventory_data
LIMIT 5;
SQLResult: [(1, 'Google Pixel 8'), (2, 'Samsung Galaxy Watch'), (3, 'Sony Wireless
Headphones'), (4, 'Cashmere 24 Rolls Tissue'), (5, 'Tide Laundry Detergent')]
Answer:Google Pixel 8, Samsung Galaxy Watch, Sony Wireless Headphones, Cashmere 24
Rolls Tissue, Tide Laundry Detergent
> Finished chain.
Google Pixel 8, Samsung Galaxy Watch, Sony Wireless Headphones, Cashmere 24 Rolls
Tissue, Tide Laundry DetergentThe product that is purchased more and sold least in
the database is Google Pixel 8. Based on the trend of the data, it is likely that
the Google Pixel 8 will be sold fast in the future as it is a popular product that
is in high demand.
```

3. Impact of increasing the Price

```
agent.run("Analyze sales response to the Electronics products. Use this data to model pot
```