Let's Build A SQL Agent To Automate Reporting!

Let's code an agent to automously explore an advertising dataset using SQL and write a report!

What you'll need!

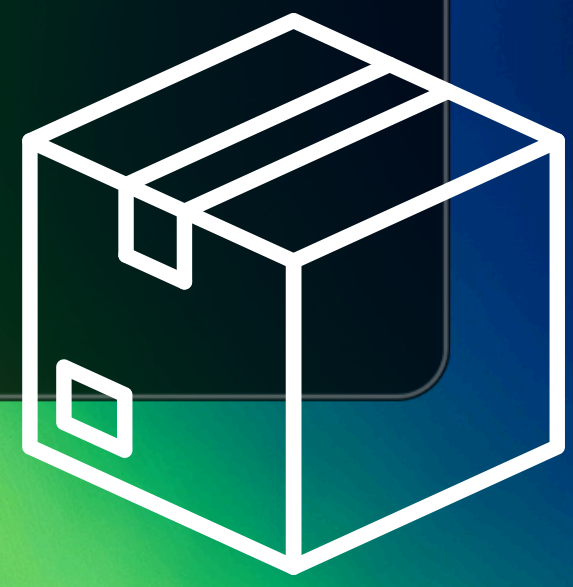
1. A OpenAI or OpenRouter API key.
2. UV, a python package manager, installed on your machine.
3. A dataset, we'll be using an open advertsing dataset.

See my post for the links for all of the above!

Setting up the project

terminal

```
$ uv init sqlagent
$ uv add smolagents duckdb
$ touch main.py
```



This installs smolagents and duckdb, and creates the python file we will be coding in!

Make sure to save the advertising dataset as ads.csv

Setting up the LLM

main.py

```python
from smolagents import OpenAIServerModel

MODEL = "google/gemini-2.5-flash-preview-05-20"
BASE_URL = "https://openrouter.ai/api/v1"
OPENROUTER_API_KEY = # Put your Openrouter API key here

model = OpenAIServerModel(
    model_id=MODEL,
    api_key=OPENROUTER_API_KEY,
    api_base=BASE_URL
)
```

I've chosen to use Openrouter and the gemini flash model, you can use other models and providers. Just check the smolagents docs.

Setting up the database

main.py

```python
import duckdb

def init_db():
    conn = duckdb.connect("ads.db")
    conn.execute("""
        CREATE TABLE IF NOT EXISTS ads AS
        SELECT * FROM 'ads.csv';
    """)

    conn.close()
```

This loads the ads.csv file into the duckdb database. DuckDB here is backed by a file called ads.db.

The Tools

We will need to give our agent some tools to use to explore the data. We will write two tools:

1. A SQL query tool.
2. A report writing tool.

This is allow our agent to explore the dataset and write a final markdown report.

The Query Tool

main.py

```python
from smolagents import tool
from typing import Literal
import pandas as pd

@tool
def duckdb_query(query: str, max_rows: int = 10, output_format: Literal["json",
"pandas", "dict"] = "json") -> str | list[dict] | pd.DataFrame:
    """
    Query the duckdb database using the provided SQL query.

    Args:
        query: The SQL query to execute.
        max_rows: The maximum number of rows to return, max 20.
        output_format: The format to return the results in.
    Returns:
        A JSON string of the results or a list of dictionaries of the result rows.
    """
    conn = duckdb.connect("ads.db")
    df = conn.query(query).df()
    conn.close()

    df_chunk = df.head(max_rows)

    match output_format:
        case "json":
            output = df_chunk.to_json(orient="records")
        case "dict":
            output = df_chunk.to_dict(orient="records")
        case "pandas":
            output = df_chunk

    return output
```

turn for explanation

The Query Tool Explained

The @tool turns this python function into a tool our agent can use.

It's important you include type hints and a docstring with an "Args" section. The @tool decorator uses these.

This tool executes a SQL query and returns the rows to the agent.

The Report Tool

main.py

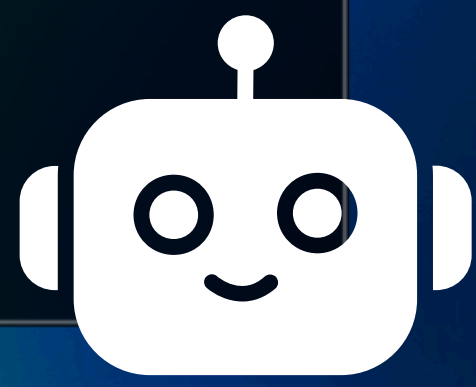
```python
@tool
def write_markdown_report(report: str, title: str) -> None:
    """
    Write a markdown report.

    Args:
        report: The report to write. Use triple quotes to pass in the report.
        title: The title of the report.
    Returns:
        None
    """
    os.makedirs("reports", exist_ok=True)
    with open(f"reports/{title}.md", "w") as f:
        f.write(report)
```

This tool allows the agent to write the report on the dataset, this report will appear in our project folder as a markdown file in a folder called “reports”.

smolagent's CodeAgent

main.py

```
from smolagents import CodeAgent
```



The CodeAgent is very cool!

Instead of using JSON blobs to use tools it will write python code which utilizes the tools you pass it.

You can also allow it to use python imports, as we will show.

Drumroll...The Agent!

main.py

```python
from smolagents import CodeAgent

agent = CodeAgent(
    model=model,
    tools=[duckdb_query, write_markdown_report],
    max_steps=50,
    additional_authorized_imports=["pandas", "duckdb", "json"]
)
```

This puts everything together!

We create a CodeAgent and pass it a list of tools and imports they can use.

The Agent's First Shift

main.py

```
agent.run("""
You are a expert data analyst. You have been given a database of advertising data to
explore.

Explore this data deeply and try to find interesting insights which would be useful
for a business.

When you are done, write a markdown report of your findings.
""")
```

Here's the task!

This is the moment! This agent.run method tells the agent to carry out a task for us given as a prompt.

Now run uv run main.py and wait for the report to be written!

The Results Are In!

Our agent will output a report noting trends and insights from the data.

You can imagine extending the functionality even further!

Insights from Advertising Data Analysis:

1. Gender Distribution:

- The dataset shows more male users (49.86%) than female users (39.62%), with a small percentage categorized as 'Other' (10.52%).
- Ad performance metrics (Clicks, Conversion Rate, CTR) are very similar across all gender categories, suggesting no significant gender bias in overall ad effectiveness.

2. Ad Type, Topic, and Placement Distribution & Performance:

- **Ad Types:** 'Video', 'Native', 'Banner', and 'Text' ad types are fairly balanced in distribution and show similar average performance. 'Video' and 'Native' ads have slightly higher CTRs and conversion rates.
- **Ad Topics:** 'Food' and 'Health' related ad topics demonstrate marginally higher CTRs and conversion rates, indicating a slightly better engagement with these content types.
- **Ad Placement:** 'Social Media' placements show a slight edge in performance metrics (CTR and Conversion Rate) compared to 'Search Engine' and 'Website' placements.

3. Income and Age-based Performance:

- **Income Brackets:** Ad performance metrics (clicks, conversion rate, CTR) are consistent across all income brackets (Low, Medium, High, Very High). There is a slight increase in Conversion Rate for "Very High" income earners. This implies that income level, within the defined brackets, does not significantly influence ad engagement or conversion.
- **Age Groups:** Similarly, performance metrics are generally consistent across 'Young' (18-35), 'Adult' (36-60), and 'Senior' (61+) age groups. While the 'Senior' group shows slightly higher clicks, their conversion rate is a bit lower. The 'Adult' group has a marginally higher CTR.

4. Click Time Analysis:

- The `Click Time` data is limited to the 20th hour of various dates, meaning a detailed hourly trend analysis is not possible.
- **Day of Week:** Wednesday ('DayOfWeek': 3) has the highest total clicks. However, performance metrics across all days of the week are relatively similar, suggesting no strong daily pattern in ad effectiveness based on this dataset.

Summary of Key Insights:

- **Overall Consistency:** The most notable observation is the remarkable consistency in ad performance metrics across almost all demographic segments (gender, age, income) and ad characteristics (type, topic, placement). The differences are very subtle.
- **Marginal Advantages:**
  - **Ad Placement:** Social Media appears to be a marginally better channel for ad placement.
  - **Ad Topic:** Food and Health related topics slightly outperform others.
  - **Ad Type:** Video and Native ads have a slight edge.
  - **Income:** Very High income bracket users show slightly higher conversion rates.
  - **Day of Week:** Wednesday is the busiest day for clicks.

Recommendations:

- **Diversified Strategy:** Given the overall consistency, continue with a diversified advertising strategy across different ad types, topics, placements, and demographics.
- **Optimized Placements:** Prioritize 'Social Media' for ad placements to potentially maximize CTR and conversion.
- **Content Focus:** Consider increasing ad content related to 'Food' and 'Health' topics, as they show slightly better engagement.
- **Target High-Income:** While overall impact is small, ads targeting very high income individuals might see a minor uplift in conversion.
- **Weekday Timing:** No strong temporal pattern but Wednesday sees a higher volume of clicks.

These insights suggest that the advertising campaigns are relatively evenly effective across various segments. Fine-tuning efforts should focus on the marginally better-performing segments and types identified above, but radical shifts are not indicated by this data alone.

Powered by ReadPo

What's Next?

You can download the code using the link in the post!

I recommend writing some additional tools such as web search.

You could even use an agent with a larger, smarter model to evaluate the insights generated by the agent, this would be a basic multi-agent system!