Agentic AI for Multi-Step Data Analysis

Whitepaper by

Wachiravit Modecrua
Director of AI Labs - Amity Solutions

amity solutions

Content

1 Introduction 1

2 Our Multi-Agent Architecture 2

- 2.1 The Planner: The Master Strategist 2
- 2.2 The Executor: The Diligent Worker 3
- 2.3 The Feedbacker: The Quality Inspector 3

3 Performance & Competitive Analysis 3

4 Conclusion and Future Work 5

Appendix A: Example Reasoning Trace 5

1. Introduction

In the rapidly evolving landscape of data analytics, the demand for sophisticated AI agents that can autonomously interpret complex queries, interact with diverse data sources, and generate accurate insights is paramount. Traditional single-agent systems often struggle with multi-step reasoning and dynamic tool utilization.

This Whitepaper introduces a novel, multi-agent architecture designed to overcome these challenges. Our system is composed of three distinct yet collaborative components: a **Planner**, an **Executor**, and a **Feedbacker**. This modular design enables a robust and transparent workflow, from initial query decomposition to final solution validation. We demonstrate the state-of-the-art (SOTA) performance of this architecture, achieving significant scores on benchmark datasets and setting a new standard for AI-driven data analysis.

The Challenge: Moving Beyond Single-Agent Systems

Modern data environments are complex, heterogeneous, and ever-changing. Answering a seemingly simple business question like "What is our top market for fraudulent transactions?" requires a multi-faceted approach. An agent must first understand the user's intent, identify the relevant data tables and fields, formulate a precise query, execute it, and then interpret the results correctly.

Most current agentic systems operate as a single, monolithic block, combining planning, execution, and reasoning. This can lead to a lack of transparency and difficulty in debugging when errors occur. Inspired by the need for specialization and clarity, we have developed a new paradigm based on a division of labor.

2. Our Multi-Agent Architecture

Our architecture deconstructs the analytical process into three specialized agent roles. This separation of concerns allows each agent to excel at its specific task, leading to a more efficient, accurate, and auditable system.

The overall workflow is designed as a pipeline, ensuring a structured and logical progression from query to validated answer.

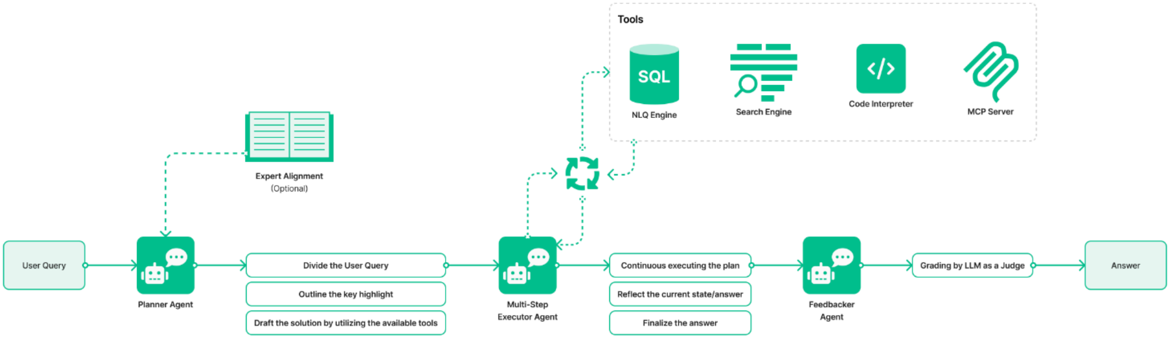


Figure 1: High-level overview of the Planner, Executor, and Feedbacker architecture

2.1 The Planner: The Master Strategist

The first point of contact for any user query is the Planner. Its primary role is not to answer the question directly, but to create a comprehensive, step-by-step blueprint for how to solve it. This involves a four-stage process:

1. **Deconstruct the Question:** The Planner breaks down the user's request into smaller, manageable sub-problems.
2. **Extract Entities:** It identifies key entities, metrics, and filters within the query (e.g., 'ip_country', 'fraud').
3. **Explore and Constrain:** The agent performs a preliminary exploration of the available data schemas to understand their structure, limitations, and constraints. This prevents the formulation of invalid or inefficient queries later on.
4. **Outline the Solution Approach:** The Planner creates an actionable plan that explicitly outlines which tools to use and in what sequence. It selects from a suite of available tools, including:
   - **Text-to-SQL Engine:** For querying structured databases.
   - **Code Interpreter:** For running Python code for complex transformations or calculations.
   - **Search Engine:** For retrieving external information or context.
   - **MCP Server (Model Context Protocol Server):** For interacting with the custom data source.

2.2 The Executor: The Diligent Worker

The Executor's task is to meticulously carry out the blueprint created by the Planner. It operates on a **Reasoning and Action (ReAct)** basis, which involves a continuous loop of thought and execution. This is not a single action but a sequence of them, forming the core of its **multi-step reasoning** capability. For example, the result from a first action (like fetching a list of top-selling products) becomes the context for a second thought process and subsequent action (like analyzing the customer demographics for those specific products). This iterative chain allows the Executor to break down complex analytical problems, handle dependencies between steps, and dynamically adapt its approach based on the output of its actions. If a SQL query returns an error or an unexpected result, the Executor can reason about the cause and attempt a different action, all while staying within the strategic confines of the original plan.

2.3 The Feedbacker: The Quality Inspector

Once the Executor produces a final answer, it is passed to the Feedbacker. This final agent acts as an automated quality assurance layer. The Feedbacker reads the final solution and the original query, then assigns a reward score based on its correctness, format, and relevance. This scoring mechanism is crucial for performance tracking, reinforcement learning, and error analysis.

3. Performance & Competitive Analysis

We evaluated our architecture on the **DABstep benchmark**, which tests an agent's ability to perform multi-step reasoning over diverse datasets. The results demonstrate the effectiveness of our multi-agent approach, not only in absolute terms but also in comparison to other leading industry agents.

AGENT/MODEL	ORGANIZATION	EASY LEVEL ACCURACY	HARD LEVEL ACCURACY
Amity DA Agent v0.1	**Amity**	**80.56%**	**41.01%**
MPHASIS-I2I-AGENTS	Mphasis	80.56%	28.04%
M365 Copilot Analyst	Microsoft	78.00%	32.00%
Gemini Data Science Agent	Google	61.11%	9.79%
OpenAI o4-mini Reasoning Prompt Baseline	Hugging Face	76.39%	14.55%
OpenAI o1 Reasoning Prompt Baseline	Adyen	69.44%	11.11%
Claude 4 Sonnet ReACT Baseline	Hugging Face	81.94%	19.84%

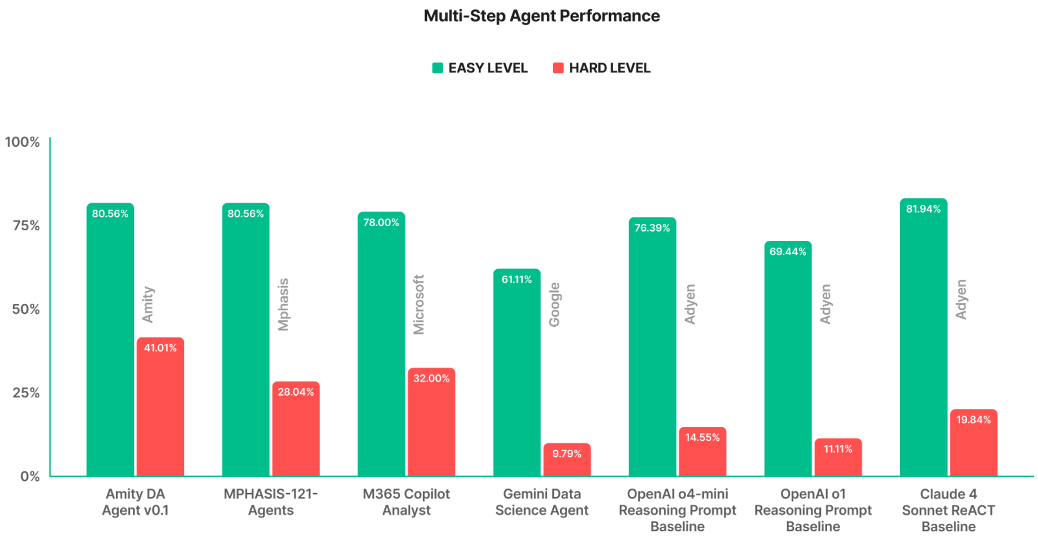
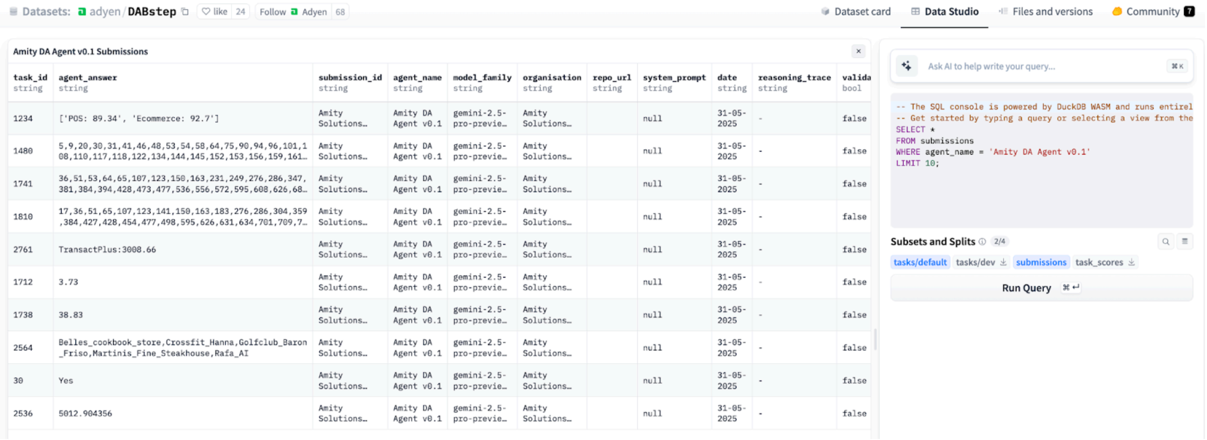


Table 1, Figure 2: Performance comparison on the DABstep benchmark

Our agent achieves the highest score on both Easy and Hard tasks, demonstrating its superior capability. Notably, our agent's **41.01%** accuracy on hard tasks significantly outperforms the next-best competitor, showcasing the robustness of thePlanner-Executor-Feedbacker design for complex, multi-step analytical challenges. While the Mphasis agent matches our performance on easy tasks, our architecture provides a distinct advantage as task difficulty increases.

The reference of our result can be accessed here.



Amity DA Agent v0.1 Submissions

task_id string	agent_answer string	submission_id string	agent_name string	model_family string	organisation string	repo_url string	system_prompt string	date string	reasoning_trace string	valida bool
1234	['POS: 89.34', 'Ecommerce: 92.7']	Amity Solutions…	Amity DA Agent v0.1	gemini-2.5-pro-previe…	Amity Solutions…		null	31-05-2025	-	false
1480	5,9,20,30,31,41,46,48,53,54,58,64,75,90,94,96,101,108,110,117,118,122,134,144,145,152,153,156,159,161…	Amity Solutions…	Amity DA Agent v0.1	gemini-2.5-pro-previe…	Amity Solutions…		null	31-05-2025	-	false
1741	36,51,53,64,65,107,123,150,163,231,249,276,286,347,381,384,394,428,473,477,536,556,572,595,608,626,68…	Amity Solutions…	Amity DA Agent v0.1	gemini-2.5-pro-previe…	Amity Solutions…		null	31-05-2025	-	false
1810	17,36,51,65,107,123,141,150,163,183,276,286,304,359,384,427,428,454,477,498,595,626,631,634,701,709,7…	Amity Solutions…	Amity DA Agent v0.1	gemini-2.5-pro-previe…	Amity Solutions…		null	31-05-2025	-	false
2761	TransactPlus:3008.66	Amity Solutions…	Amity DA Agent v0.1	gemini-2.5-pro-previe…	Amity Solutions…		null	31-05-2025	-	false
1712	3.73	Amity Solutions…	Amity DA Agent v0.1	gemini-2.5-pro-previe…	Amity Solutions…		null	31-05-2025	-	false
1738	38.83	Amity Solutions…	Amity DA Agent v0.1	gemini-2.5-pro-previe…	Amity Solutions…		null	31-05-2025	-	false
2564	Belles_cookbook_store,Crossfit_Hanna,Golfclub_Baron_Friso,Martinis_Fine_Steakhouse,Rafa_AI	Amity Solutions…	Amity DA Agent v0.1	gemini-2.5-pro-previe…	Amity Solutions…		null	31-05-2025	-	false
30	Yes	Amity Solutions…	Amity DA Agent v0.1	gemini-2.5-pro-previe…	Amity Solutions…		null	31-05-2025	-	false
2536	5012.904356	Amity Solutions…	Amity DA Agent v0.1	gemini-2.5-pro-previe…	Amity Solutions…		null	31-05-2025	-	false

```sql
-- The SQL console is powered by DuckDB WASM and runs entirel
-- Get started by typing a query or selecting a view from the
SELECT *
FROM submissions
WHERE agent_name = 'Amity DA Agent v0.1'
LIMIT 10;
```

Figure 3: The reference of our DABStep Result

4. Conclusion and Future Work

Our Planner-Executor-Feedbacker architecture represents a significant step forward in building intelligent, reliable, and transparent data analysis agents. By separating concerns, we empower each component to perform its function optimally, leading to state-of-the-art results that are highly competitive with major industry players.

Future work will focus on:

1. Enhancing the Planner's strategic capabilities with more complex by synthesizing and finetuning using GRPO technique and Test-Time Reinforcement Learning by utilizing the Grader which is built by Human Expert Data Analyst. (Ref1, Ref2)
2. Expanding the Executor's toolset to include more advanced statistical analysis and visualization libraries.

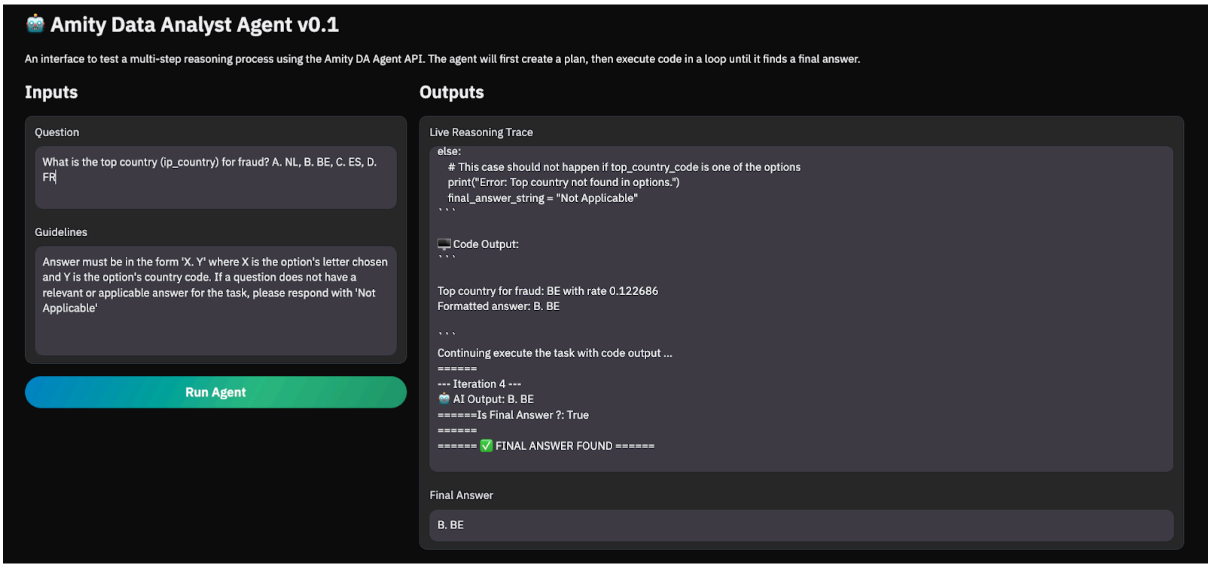
Appendix A: Example Reasoning Trace

To provide transparency into the agent's "thought process," we can examine its reasoning trace. The following is a real example from DABStep Datasets.

Question: What is the top country (ip_country) for fraud? A. NL, B. BE, C. ES, D. FR

Guideline: Answer must be in the form 'X. Y' where:
X is the option's letter chosen
Y is the option's country code.

Reasoning Trace UI



Amity Data Analyst Agent v0.1

An interface to test a multi-step reasoning process using the Amity DA Agent API. The agent will first create a plan, then execute code in a loop until it finds a final answer.

Inputs

Question: What is the top country (ip_country) for fraud? A. NL, B. BE, C. ES, D. FR

Guidelines: Answer must be in the form 'X. Y' where X is the option's letter chosen and Y is the option's country code. If a question does not have a relevant or applicable answer for the task, please respond with 'Not Applicable'

Outputs

Live Reasoning Trace

~~~~
else:
   # This case should not happen if top_country_code is one of the options
   print("Error: Top country not found in options.")
   final_answer_string = "Not Applicable"
```

Code Output:
```

Top country for fraud: BE with rate 0.122686
Formatted answer: B. BE

```
Continuing execute the task with code output ...
======
--- Iteration 4 ---
AI Output: B. BE
======Is Final Answer ?: True
======
====== ✅ FINAL ANSWER FOUND ======
~~~~

Final Answer: B. BE

Figure 4: Reasoning Trace UI

Reasoning Trace Log

~~~
Step 1: Generating a plan to solve the problem...

====== 📝 PLAN ======

Okay, I will break down the user query and provide the requested information based on `manual.md`.

Sub-questions:
- What is the definition of "fraud" or "fraud level" according to the `manual.md`?
- How should "top country for fraud" be determined based on this definition?
- Which data fields from the available data sources (like `payments.csv` described by
`payments-readme.md`) would be used to identify the `ip_country` and measure fraud?
- What are the specific `ip_country` options to evaluate?


Entity Extraction:
- `ip_country`: The country of the shopper at the time of the transaction. Options provided: NL, BE, ES,
FR.
- `fraud`: The metric to be measured. Based on `manual.md`, this is typically a ratio involving fraudulent
volume.
- `top country`: The `ip_country` with the highest calculated fraud metric among the given options.


========== START of Relevant Chunk ==========

Relevant Chunk:
```manual.md
## 5. Understanding Payment Processing Fees


Payment Processing Fees depend on a number of characteristics. These characteristics belong to either the
merchant or the transaction.


Merchant characteritics include
...
~~~
========== END of Relevant Chunk ==========

Constraints:
```
- The user query asks to identify the "top country (ip_country) for fraud" from a specific list of
options: A. NL, B. BE, C. ES, D. FR.
- The `manual.md` (Section 7) defines fraud as "the ratio of fraudulent volume over total volume". Section
5 also refers to "monthly_fraud_level" as a "ratio between monthly total volume and monthly volume
notified as fraud". This definition should be used for calculation.
- The actual calculation requires data from `payments.csv`, specifically the `ip_country`, `eur_amount`
(for volume), and `has_fraudulent_dispute` (to identify fraudulent transactions/volume) columns. The
`manual.md` itself does not contain this transactional data but provides the definition of fraud rate.
- The answer must strictly follow the format 'X. Y', where X is the letter of the chosen option and Y is
the country code.
```

Solution Approach:
- To answer the query, one would need to process the `payments.csv` file.
- For each of the specified `ip_country` options (NL, BE, ES, FR):
   1. Filter the transaction data for the specific `ip_country`.
   2. Calculate the total transaction volume for that country (sum of `eur_amount`).
   3. Calculate the total fraudulent transaction volume for that country (sum of `eur_amount` for
transactions where `has_fraudulent_dispute` is True).
   4. Compute the fraud rate for that country using the definition from `manual.md`: (fraudulent volume /
total volume).
- Compare the calculated fraud rates for NL, BE, ES, and FR.
```
```

~~~
- The country with the highest fraud rate among these options is the "top country for fraud".
- The final answer should be formatted as 'X. Y', where X is the letter corresponding to the top country
and Y is its code (e.g., 'A. NL').
====================


Step 2: Starting execution loop...


--- Iteration 1 ---
🤖 AI Output: **Explore**
I need to determine the top country for fraud based on `ip_country` from the options NL, BE, ES, FR.
Fraud is defined as the ratio of fraudulent volume over total volume.
The relevant data is in `payments.csv`, specifically the columns: `ip_country`, `eur_amount`, and
`has_fraudulent_dispute`.


Let's start by loading the `payments.csv` file and inspecting the relevant columns.


**Plan**
1. Load `payments.csv` into a pandas DataFrame.
2. Filter the DataFrame for transactions where `ip_country` is one of 'NL', 'BE', 'ES', 'FR'.
3. For each of these countries:
   a. Calculate the total volume (sum of `eur_amount`).
   b. Calculate the fraudulent volume (sum of `eur_amount` for transactions where `has_fraudulent_dispute`
is True).
   c. Calculate the fraud rate (fraudulent volume / total volume).
4. Identify the country with the highest fraud rate among the given options.
5. Format the answer as 'X. Y'.


**Execute**


Step 1: Load `payments.csv` and inspect.
Thought: I will load the `payments.csv` file and display its head and info to understand its structure and
data types, especially for the columns `ip_country`, `eur_amount`, and `has_fraudulent_dispute`.
Code:
```python
import pandas as pd


# Load the payments.csv file
payments_df = pd.read_csv("data/context/payments.csv")


# Display the first few rows
print("Payments DataFrame head:")
print(payments_df.head())


# Display DataFrame info
print("\nPayments DataFrame info:")
payments_df.info()


# Check unique values for ip_country to confirm the options are present
print("\nUnique values in ip_country:")
print(payments_df['ip_country'].unique())


# Check unique values for has_fraudulent_dispute
print("\nUnique values in has_fraudulent_dispute:")
print(payments_df['has_fraudulent_dispute'].unique())
~~~

```
# Check data type of eur_amount
print("\nData type of eur_amount:")
print(payments_df['eur_amount'].dtype)
```
Is Final Answer ?: False

🐍 Executing Code:
```python
import pandas as pd


# Load the payments.csv file
payments_df = pd.read_csv("data/context/payments.csv")


# Display the first few rows
print("Payments DataFrame head:")
print(payments_df.head())


# Display DataFrame info
print("\nPayments DataFrame info:")
payments_df.info()


# Check unique values for ip_country to confirm the options are present
print("\nUnique values in ip_country:")
print(payments_df['ip_country'].unique())


# Check unique values for has_fraudulent_dispute
print("\nUnique values in has_fraudulent_dispute:")
print(payments_df['has_fraudulent_dispute'].unique())


# Check data type of eur_amount
print("\nData type of eur_amount:")
print(payments_df['eur_amount'].dtype)
```

🖥️ Code Output:
```
Payments DataFrame head:
  psp_reference               merchant card_scheme  year  hour_of_day  minute_of_hour  ...
shopper_interaction  card_bin  has_fraudulent_dispute is_refused_by_adyen aci acquirer_country
0    20034594130         Crossfit_Hanna      NexPay  2023           16              21  ...
Ecommerce      4802                   False               False   F               NL
1    36926127356         Crossfit_Hanna      NexPay  2023           23              58  ...
Ecommerce      4920                   False               False   F               NL
2    31114608278  Belles_cookbook_store  GlobalCard  2023            4              30  ...
Ecommerce      4571                   False               False   F               US
3    68442235288         Crossfit_Hanna      NexPay  2023            3               5  ...
Ecommerce      4017                   False               False   D               NL
4    81404384199         Crossfit_Hanna      NexPay  2023           17              30  ...
Ecommerce      4532                   False               False   F               NL


[5 rows x 21 columns]


Payments DataFrame info:
<class 'pandas.core.frame.DataFrame'>
```

```
RangeIndex: 138236 entries, 0 to 138235
Data columns (total 21 columns):
#   Column                  Non-Null Count   Dtype
---  ------                  --------------   -----
0   psp_reference           138236 non-null  int64
1   merchant                138236 non-null  object
2   card_scheme             138236 non-null  object
3   year                    138236 non-null  int64
4   hour_of_day             138236 non-null  int64
5   minute_of_hour          138236 non-null  int64
6   day_of_year             138236 non-null  int64
7   is_credit               138236 non-null  bool
8   eur_amount              138236 non-null  float64
9   ip_country              138236 non-null  object
10  issuing_country         138236 non-null  object
11  device_type             138236 non-null  object
12  ip_address              110589 non-null  object
13  email_address           124412 non-null  object
14  card_number             138236 non-null  object
15  shopper_interaction     138236 non-null  object
16  card_bin                138236 non-null  int64
17  has_fraudulent_dispute  138236 non-null  bool
18  is_refused_by_adyen     138236 non-null  bool
19  aci                     138236 non-null  object
20  acquirer_country        138236 non-null  object
dtypes: bool(3), float64(1), int64(6), object(11)
memory usage: 19.4+ MB


Unique values in ip_country:
['SE' 'NL' 'LU' 'IT' 'BE' 'FR' 'GR' 'ES']


Unique values in has_fraudulent_dispute:
[False  True]


Data type of eur_amount:
float64


```
Continuing execute the task with code output ...

--- Iteration 2 ---
🤖 AI Output: Thought: The data types and values are as expected. Now I will proceed to filter the
DataFrame for the specified `ip_country` values and then calculate the fraud rates.
I will perform these calculations in a loop for each country and store the results.
Code:
```python
import pandas as pd


# Load the payments.csv file if not already loaded (it is in this session)
# payments_df = pd.read_csv("data/context/payments.csv")


# List of countries to analyze
countries_of_interest = ['NL', 'BE', 'ES', 'FR']
options_map = {'NL': 'A', 'BE': 'B', 'ES': 'C', 'FR': 'D'}


# Filter the DataFrame for these countries
filtered_payments_df = payments_df[payments_df['ip_country'].isin(countries_of_interest)]
```

~~~
# Calculate fraud rates for each country
fraud_rates = {}


for country in countries_of_interest:
    country_df = filtered_payments_df[filtered_payments_df['ip_country'] == country]

    # Calculate total volume
    total_volume = country_df['eur_amount'].sum()

    # Calculate fraudulent volume
    fraudulent_volume = country_df[country_df['has_fraudulent_dispute'] == True]['eur_amount'].sum()

    # Calculate fraud rate (handle division by zero if total_volume is 0)
    if total_volume > 0:
        fraud_rate = fraudulent_volume / total_volume
    else:
        fraud_rate = 0.0  # Or handle as NaN, but 0.0 is fine for comparison if no transactions

    fraud_rates[country] = fraud_rate
    print(f"Country: {country}")
    print(f"  Total Volume: {total_volume:.2f}")
    print(f"  Fraudulent Volume: {fraudulent_volume:.2f}")
    print(f"  Fraud Rate: {fraud_rate:.4f}")


# Identify the country with the highest fraud rate
top_country_code = None
max_fraud_rate = -1.0


for country, rate in fraud_rates.items():
    if rate > max_fraud_rate:
        max_fraud_rate = rate
        top_country_code = country


print(f"\nFraud rates: {fraud_rates}")
print(f"Top country for fraud: {top_country_code} with rate {max_fraud_rate:.4f}")


# Determine the option letter
top_country_option_letter = options_map[top_country_code]


final_answer_string = f"{top_country_option_letter}. {top_country_code}"
print(f"\nFinal answer string: {final_answer_string}")
```
Is Final Answer ?: False

🐍 Executing Code:
```python
import pandas as pd


# Load the payments.csv file if not already loaded (it is in this session)
# payments_df = pd.read_csv("data/context/payments.csv")


# List of countries to analyze
countries_of_interest = ['NL', 'BE', 'ES', 'FR']
options_map = {'NL': 'A', 'BE': 'B', 'ES': 'C', 'FR': 'D'}
~~~

```
# Filter the DataFrame for these countries
filtered_payments_df = payments_df[payments_df['ip_country'].isin(countries_of_interest)]


# Calculate fraud rates for each country
fraud_rates = {}


for country in countries_of_interest:
    country_df = filtered_payments_df[filtered_payments_df['ip_country'] == country]

    # Calculate total volume
    total_volume = country_df['eur_amount'].sum()

    # Calculate fraudulent volume
    fraudulent_volume = country_df[country_df['has_fraudulent_dispute'] == True]['eur_amount'].sum()

    # Calculate fraud rate (handle division by zero if total_volume is 0)
    if total_volume > 0:
        fraud_rate = fraudulent_volume / total_volume
    else:
        fraud_rate = 0.0  # Or handle as NaN, but 0.0 is fine for comparison if no transactions

    fraud_rates[country] = fraud_rate
    print(f"Country: {country}")
    print(f"  Total Volume: {total_volume:.2f}")
    print(f"  Fraudulent Volume: {fraudulent_volume:.2f}")
    print(f"  Fraud Rate: {fraud_rate:.4f}")


# Identify the country with the highest fraud rate
top_country_code = None
max_fraud_rate = -1.0


for country, rate in fraud_rates.items():
    if rate > max_fraud_rate:
        max_fraud_rate = rate
        top_country_code = country


print(f"\nFraud rates: {fraud_rates}")
print(f"Top country for fraud: {top_country_code} with rate {max_fraud_rate:.4f}")


# Determine the option letter
top_country_option_letter = options_map[top_country_code]


final_answer_string = f"{top_country_option_letter}. {top_country_code}"
print(f"\nFinal answer string: {final_answer_string}")
```

🖥 Code Output:
```
Country: NL
 Total Volume: 2701907.13
 Fraudulent Volume: 329134.08
 Fraud Rate: 0.1218
Country: BE
 Total Volume: 2150473.54
 Fraudulent Volume: 263833.85
 Fraud Rate: 0.1227
Country: ES
```

```
 Total Volume: 644883.17
 Fraudulent Volume: 43531.87
 Fraud Rate: 0.0675
Country: FR
 Total Volume: 1292201.83
 Fraudulent Volume: 89135.03
 Fraud Rate: 0.0690


Fraud rates: {'NL': np.float64(0.12181546743244279), 'BE': np.float64(0.12268639678310111), 'ES':
np.float64(0.06750349834684011), 'FR': np.float64(0.068979185705068899)}
Top country for fraud: BE with rate 0.1227


Final answer string: B. BE


```
Continuing execute the task with code output ...

--- Iteration 3 ---
🤖 AI Output: B. BE
Is Final Answer ?: True

====== ✅ FINAL ANSWER FOUND ======
```
```