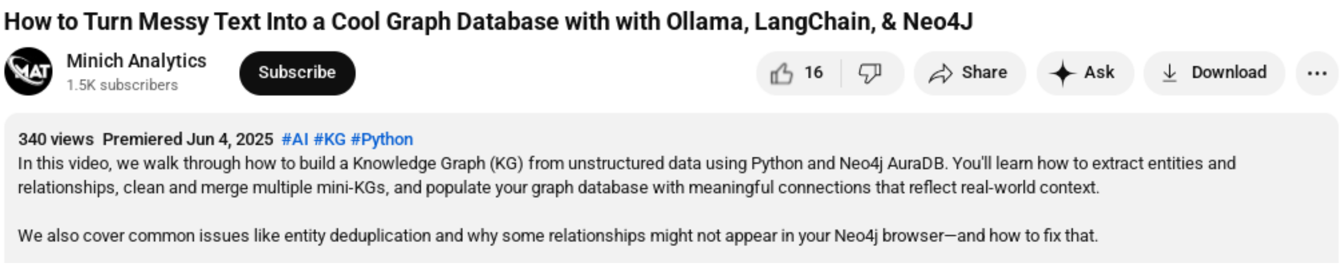
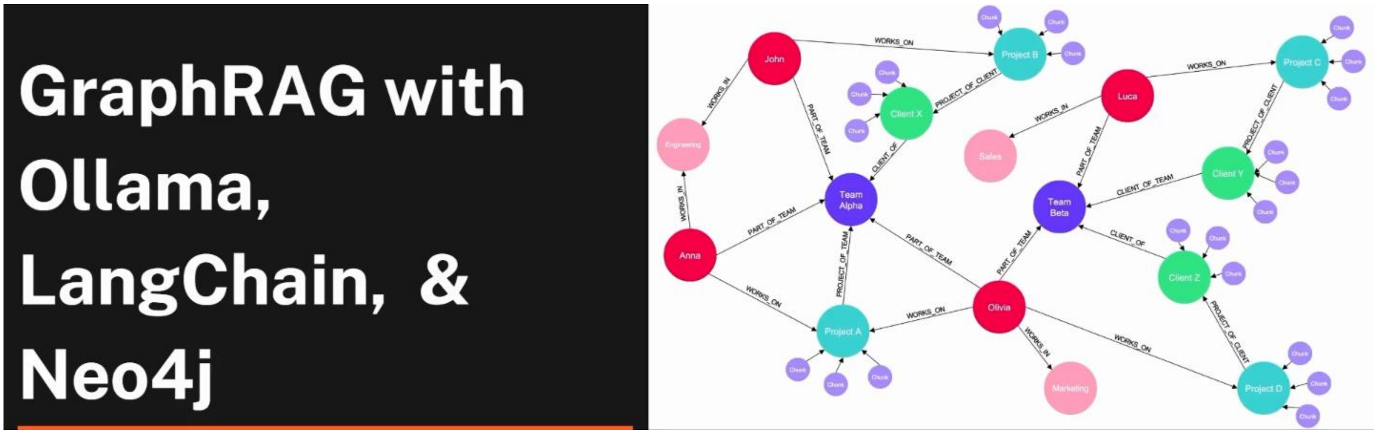


How to Turn Messy Text Into a Cool Graph Database with with Ollama, LangChain, & Neo4J

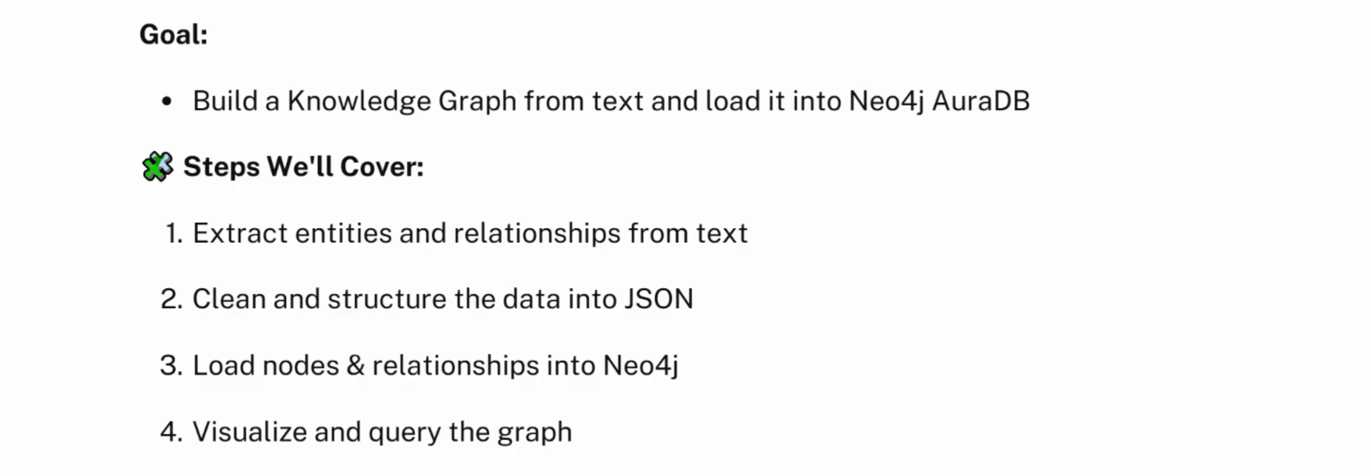
340 views Premiered Jun 4, 2025 #AI #KG #Python

In this video, we walk through how to build a Knowledge Graph (KG) from unstructured data using Python and Neo4j AuraDB. You'll learn how to extract entities and relationships, clean and merge multiple mini-KGs, and populate your graph database with meaningful connections that reflect real-world context.

We also cover common issues like entity deduplication and why some relationships might not appear in your Neo4j browser—and how to fix that.



GraphRAG with Ollama, LangChain, & Neo4j

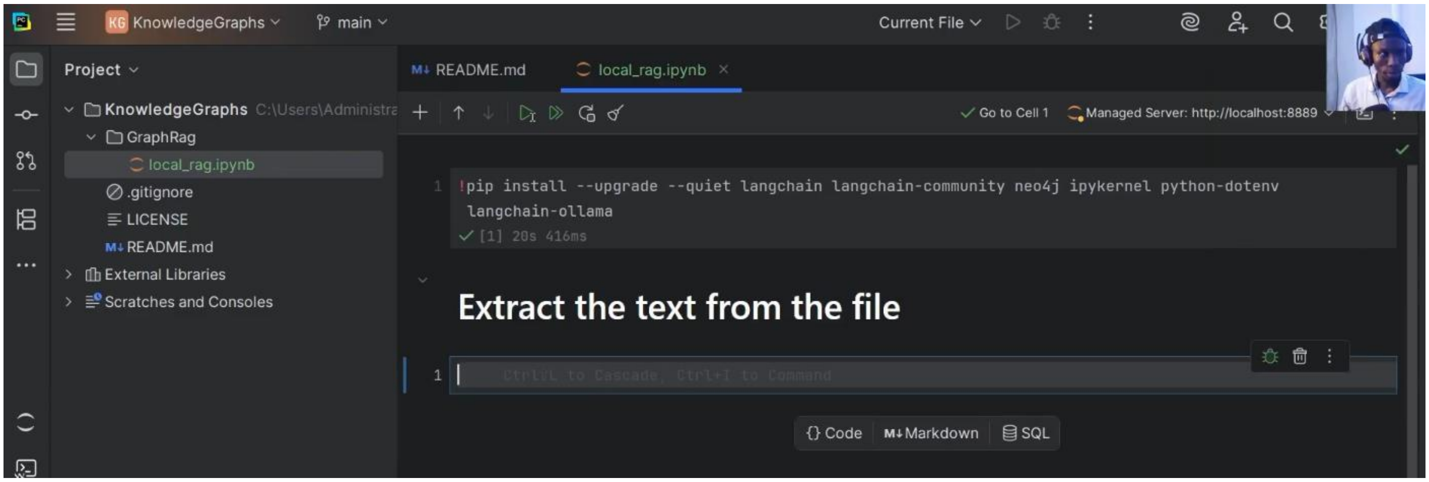


Goal:

- Build a Knowledge Graph from text and load it into Neo4j AuraDB

🧩 **Steps We'll Cover:**

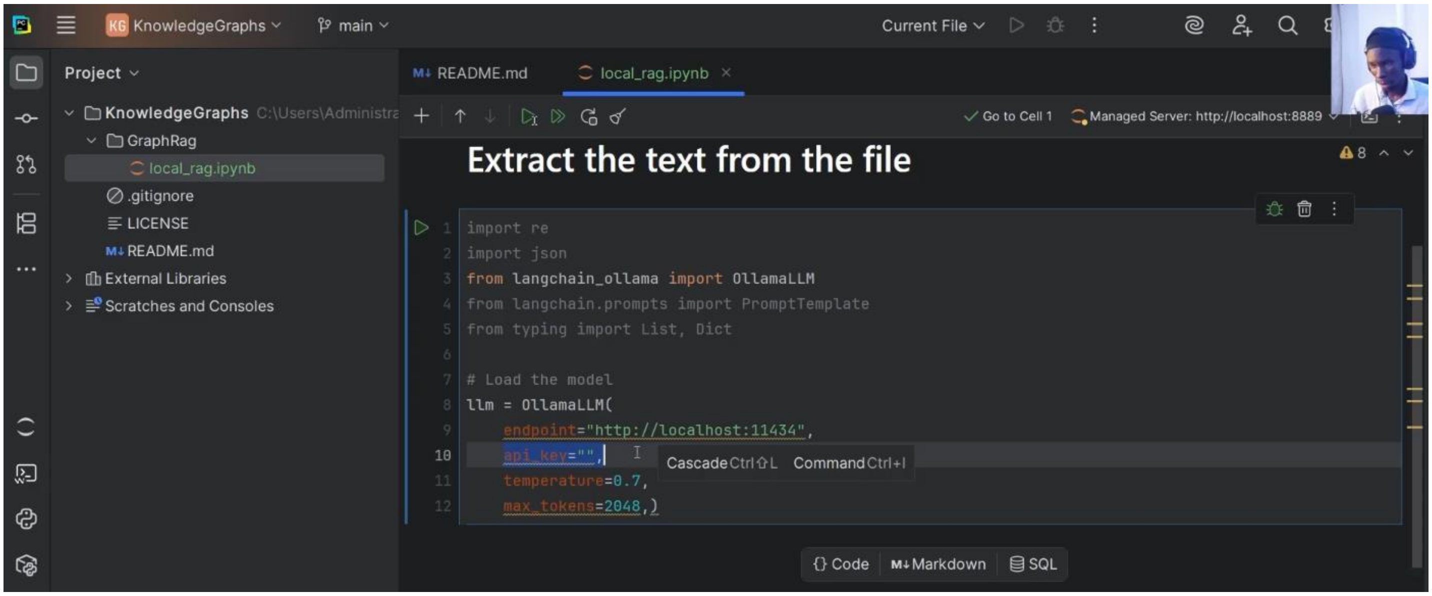
1. Extract entities and relationships from text
2. Clean and structure the data into JSON
3. Load nodes & relationships into Neo4j
4. Visualize and query the graph



```
!pip install --upgrade --quiet langchain langchain-community neo4j ipykernel python-dotenv
 langchain-ollama
✓ [1] 20s 416ms
```

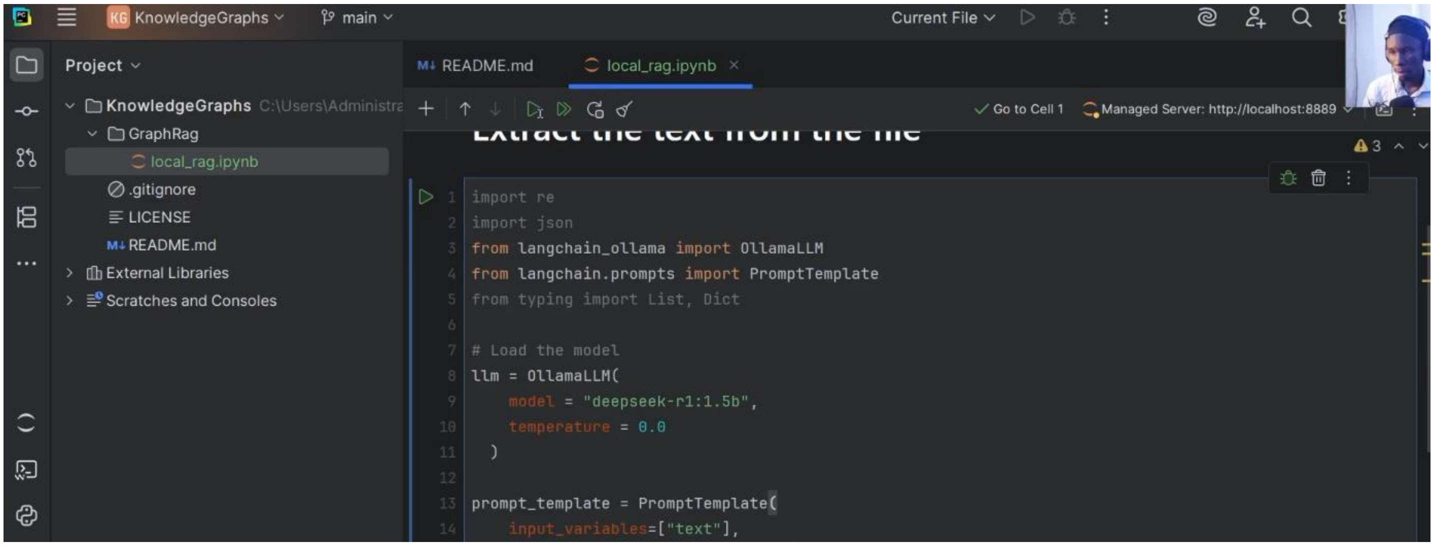
Extract the text from the file

```
Ctrl+L to Cascade, Ctrl+I to Command
```



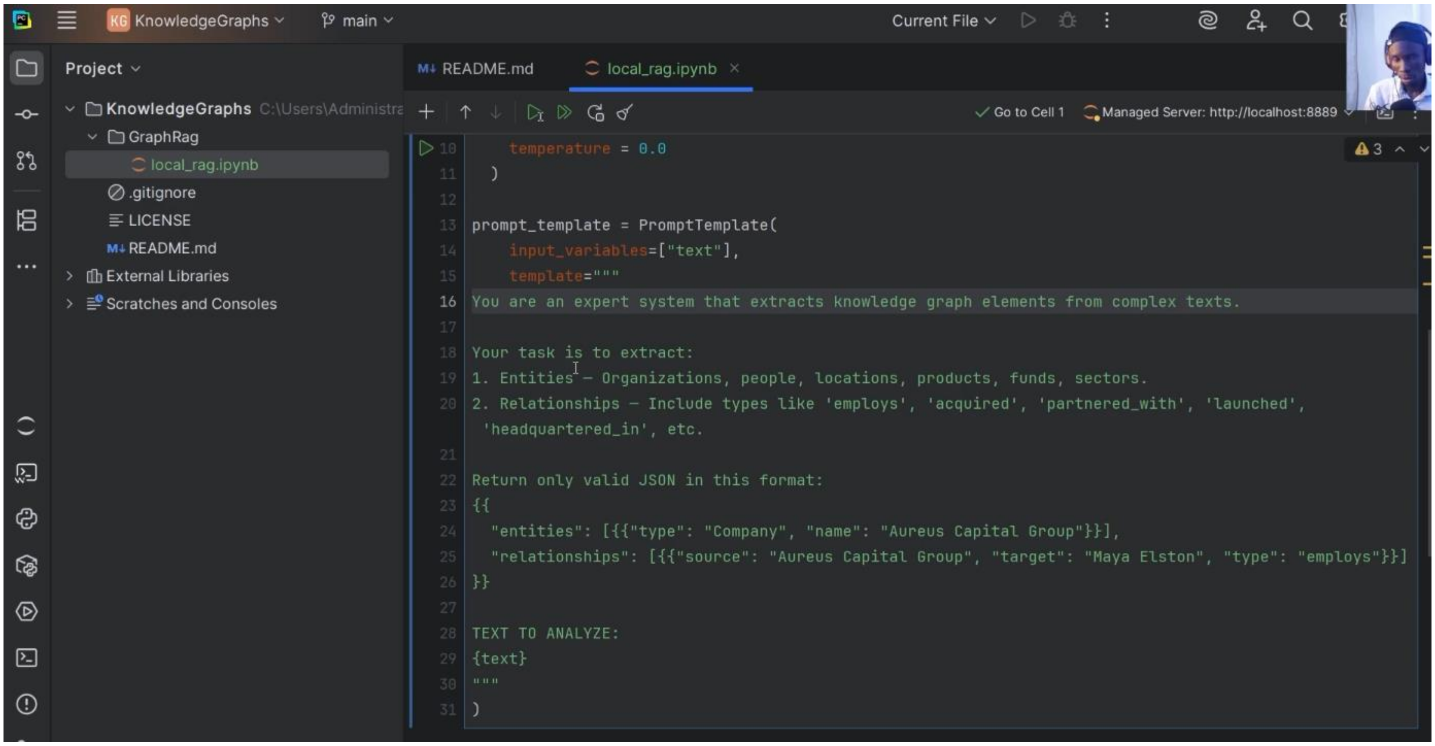
Extract the text from the file

```python
import re
import json
from langchain_ollama import OllamaLLM
from langchain.prompts import PromptTemplate
from typing import List, Dict

# Load the model
llm = OllamaLLM(
    endpoint="http://localhost:11434",
    api_key="",
    temperature=0.7,
    max_tokens=2048,)
```

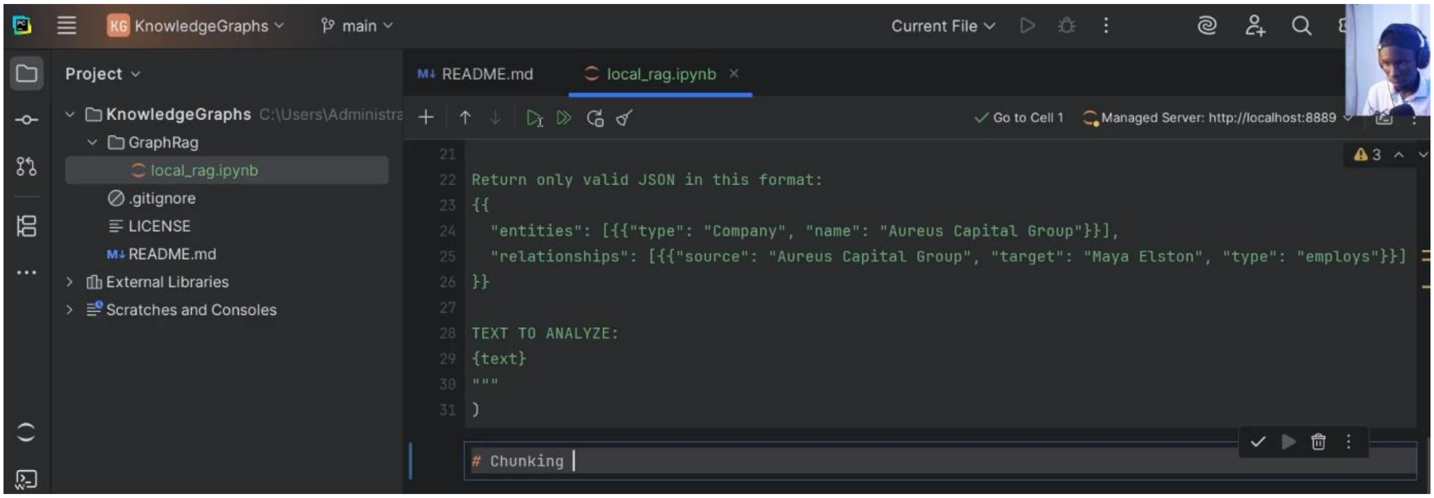


Extract the text from the file

```python
import re
import json
from langchain_ollama import OllamaLLM
from langchain.prompts import PromptTemplate
from typing import List, Dict

# Load the model
llm = OllamaLLM(
    model = "deepseek-r1:1.5b",
    temperature = 0.0
  )

prompt_template = PromptTemplate(
    input_variables=["text"],
```

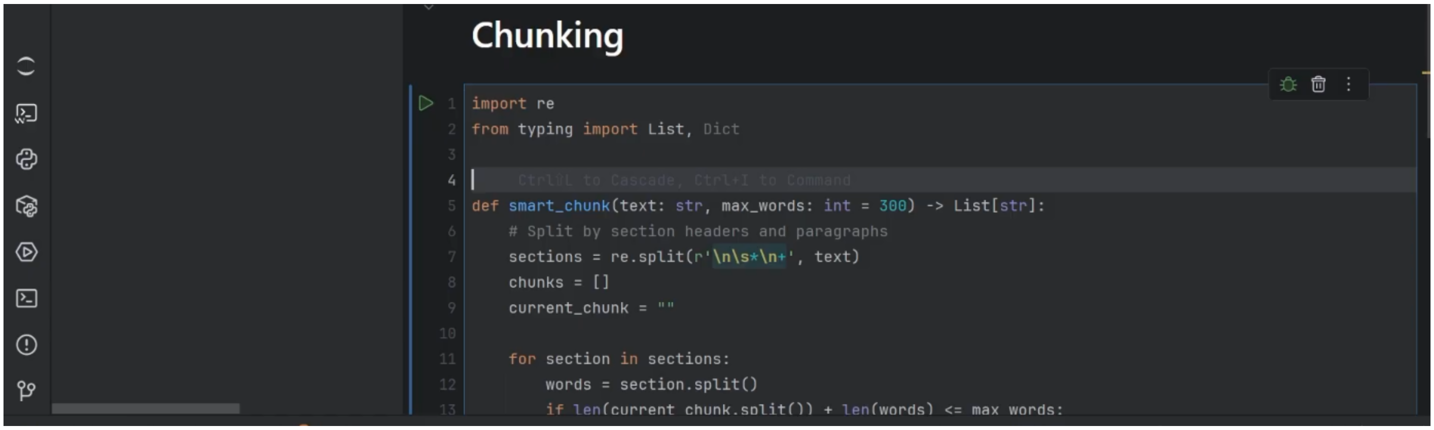


```python
    temperature = 0.0
  )

prompt_template = PromptTemplate(
    input_variables=["text"],
    template="""
You are an expert system that extracts knowledge graph elements from complex texts.

Your task is to extract:
1. Entities – Organizations, people, locations, products, funds, sectors.
2. Relationships – Include types like 'employs', 'acquired', 'partnered_with', 'launched',
 'headquartered_in', etc.

Return only valid JSON in this format:
{{
  "entities": [{{"type": "Company", "name": "Aureus Capital Group"}}],
  "relationships": [{{"source": "Aureus Capital Group", "target": "Maya Elston", "type": "employs"}}]
}}

TEXT TO ANALYZE:
{text}
"""
)
```



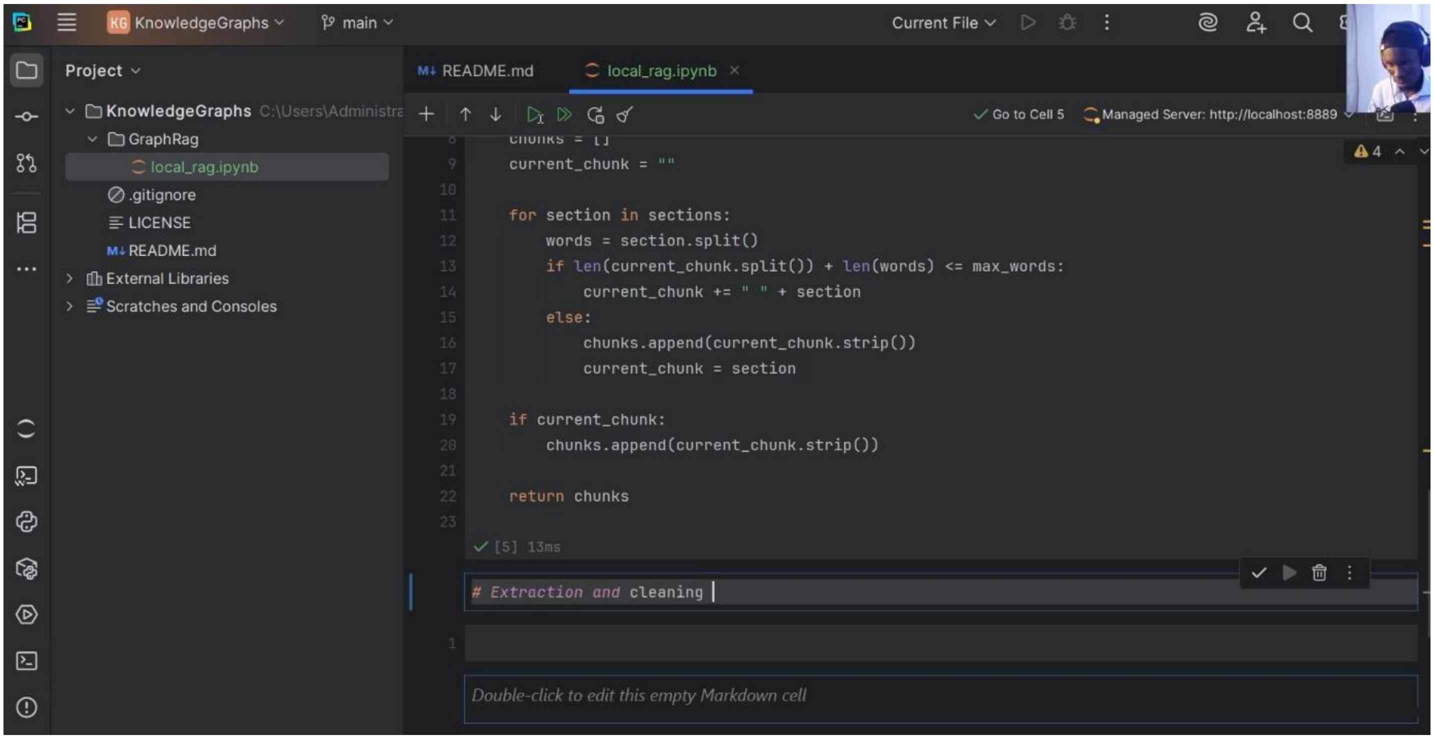
```

Return only valid JSON in this format:
{{
  "entities": [{{"type": "Company", "name": "Aureus Capital Group"}}],
  "relationships": [{{"source": "Aureus Capital Group", "target": "Maya Elston", "type": "employs"}}]
}}

TEXT TO ANALYZE:
{text}
"""
)
```

```
# Chunking
```



Chunking

```python
import re
from typing import List, Dict

      Ctrl⇧L to Cascade, Ctrl+I to Command
def smart_chunk(text: str, max_words: int = 300) -> List[str]:
    # Split by section headers and paragraphs
    sections = re.split(r'\n\s*\n+', text)
    chunks = []
    current_chunk = ""

    for section in sections:
        words = section.split()
        if len(current_chunk.split()) + len(words) <= max_words:
```

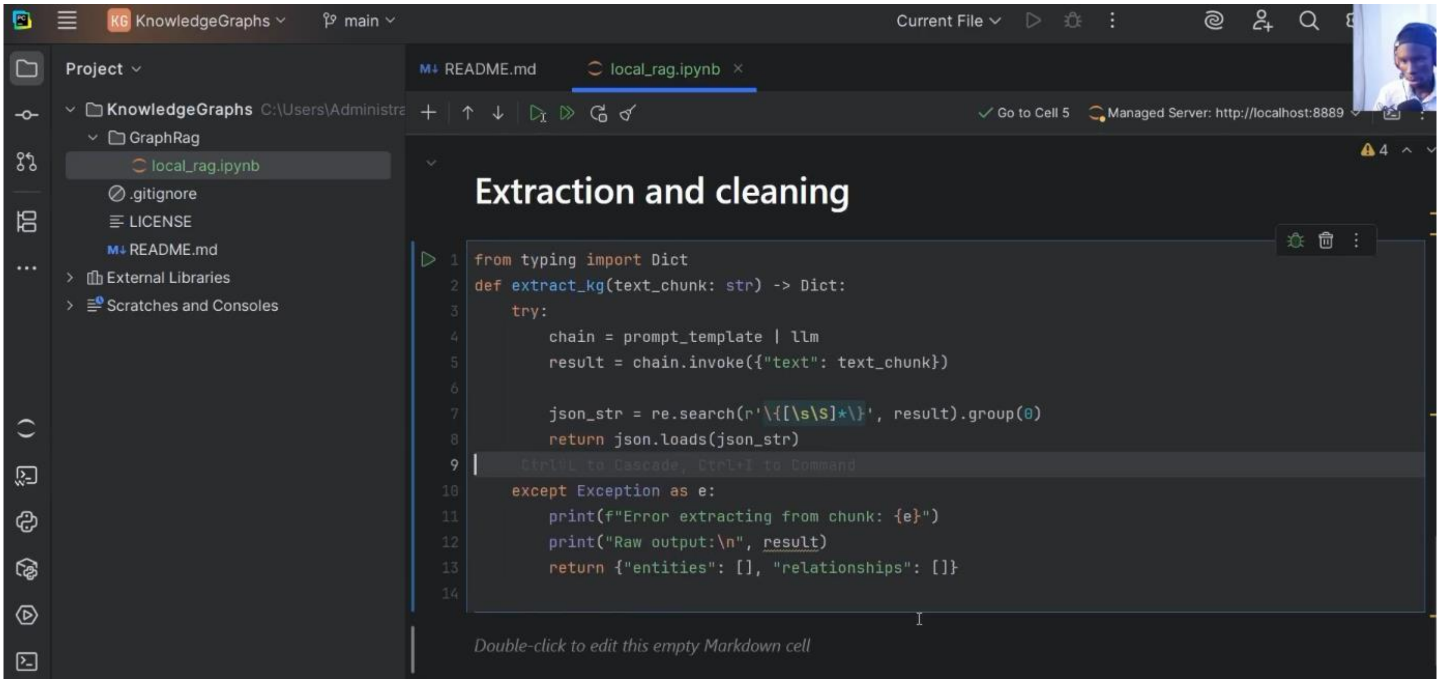


```python
    chunks = []
    current_chunk = ""

    for section in sections:
        words = section.split()
        if len(current_chunk.split()) + len(words) <= max_words:
            current_chunk += " " + section
        else:
            chunks.append(current_chunk.strip())
            current_chunk = section

    if current_chunk:
        chunks.append(current_chunk.strip())

    return chunks

```

✓ [5] 13ms

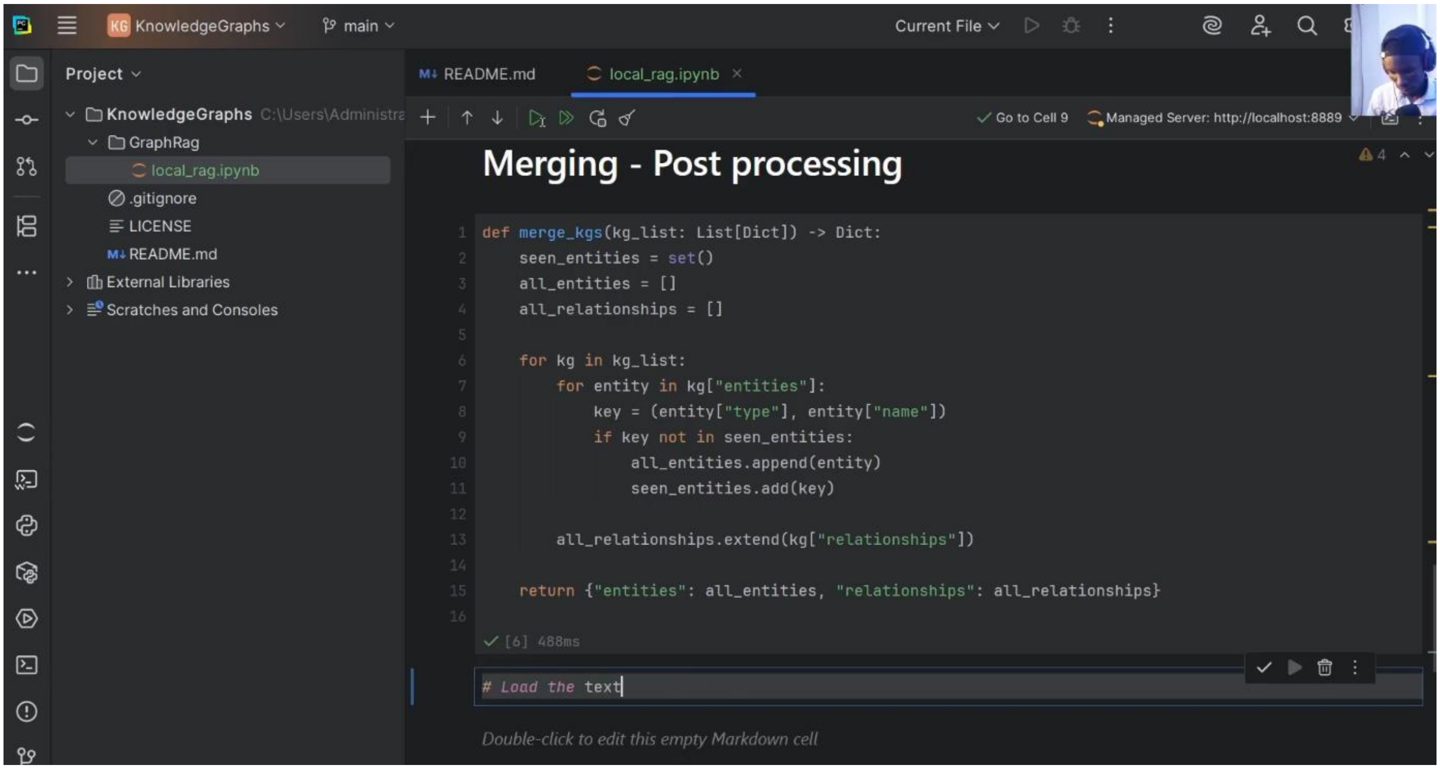
```
# Extraction and cleaning
```

Double-click to edit this empty Markdown cell



Extraction and cleaning

```python
from typing import Dict
def extract_kg(text_chunk: str) -> Dict:
    try:
        chain = prompt_template | llm
        result = chain.invoke({"text": text_chunk})

        json_str = re.search(r'\{[\s\S]*\}', result).group(0)
        return json.loads(json_str)

    except Exception as e:
        print(f"Error extracting from chunk: {e}")
        print("Raw output:\n", result)
        return {"entities": [], "relationships": []}

```

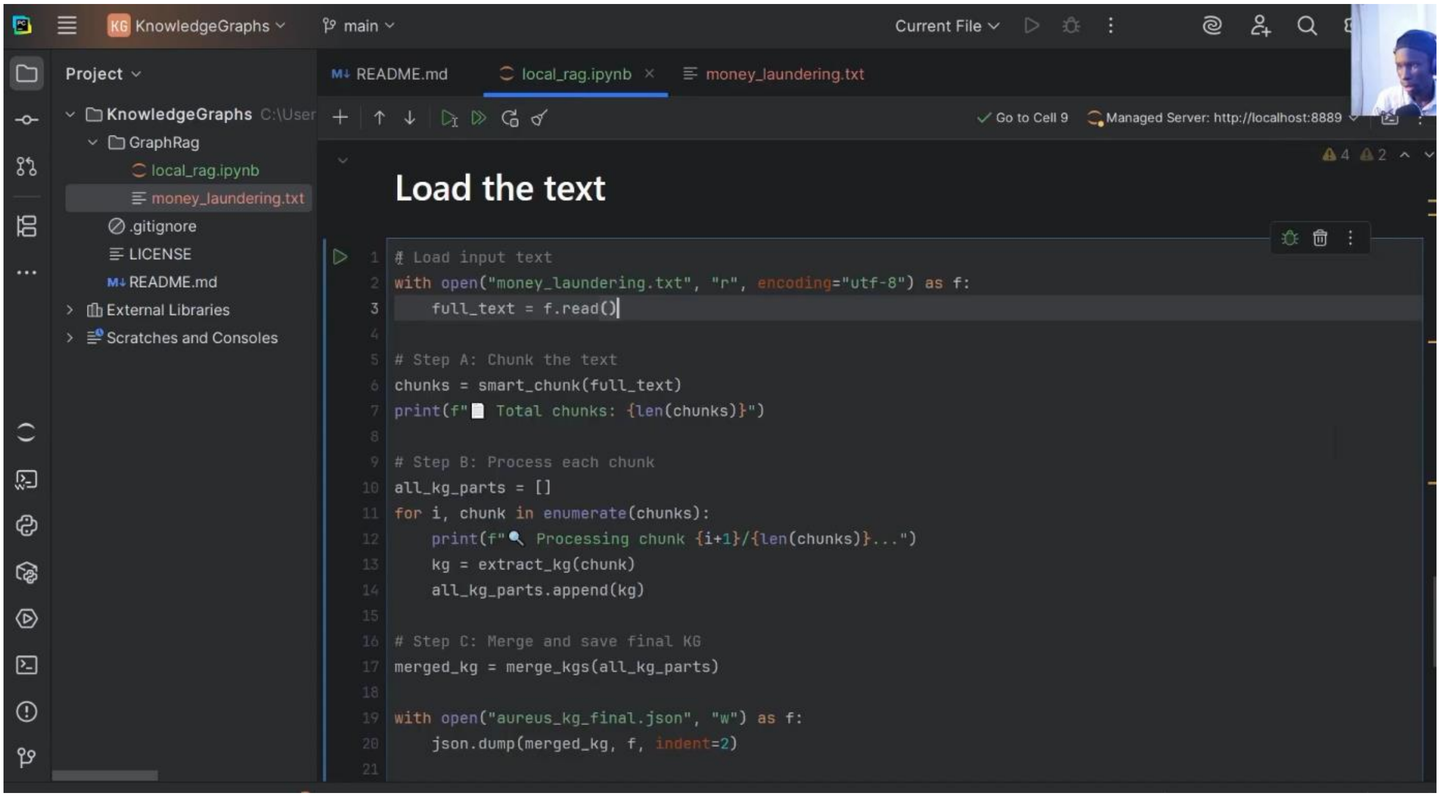


Merging - Post processing

```python
def merge_kgs(kg_list: List[Dict]) -> Dict:
    seen_entities = set()
    all_entities = []
    all_relationships = []

    for kg in kg_list:
        for entity in kg["entities"]:
            key = (entity["type"], entity["name"])
            if key not in seen_entities:
                all_entities.append(entity)
                seen_entities.add(key)

        all_relationships.extend(kg["relationships"])

    return {"entities": all_entities, "relationships": all_relationships}

```

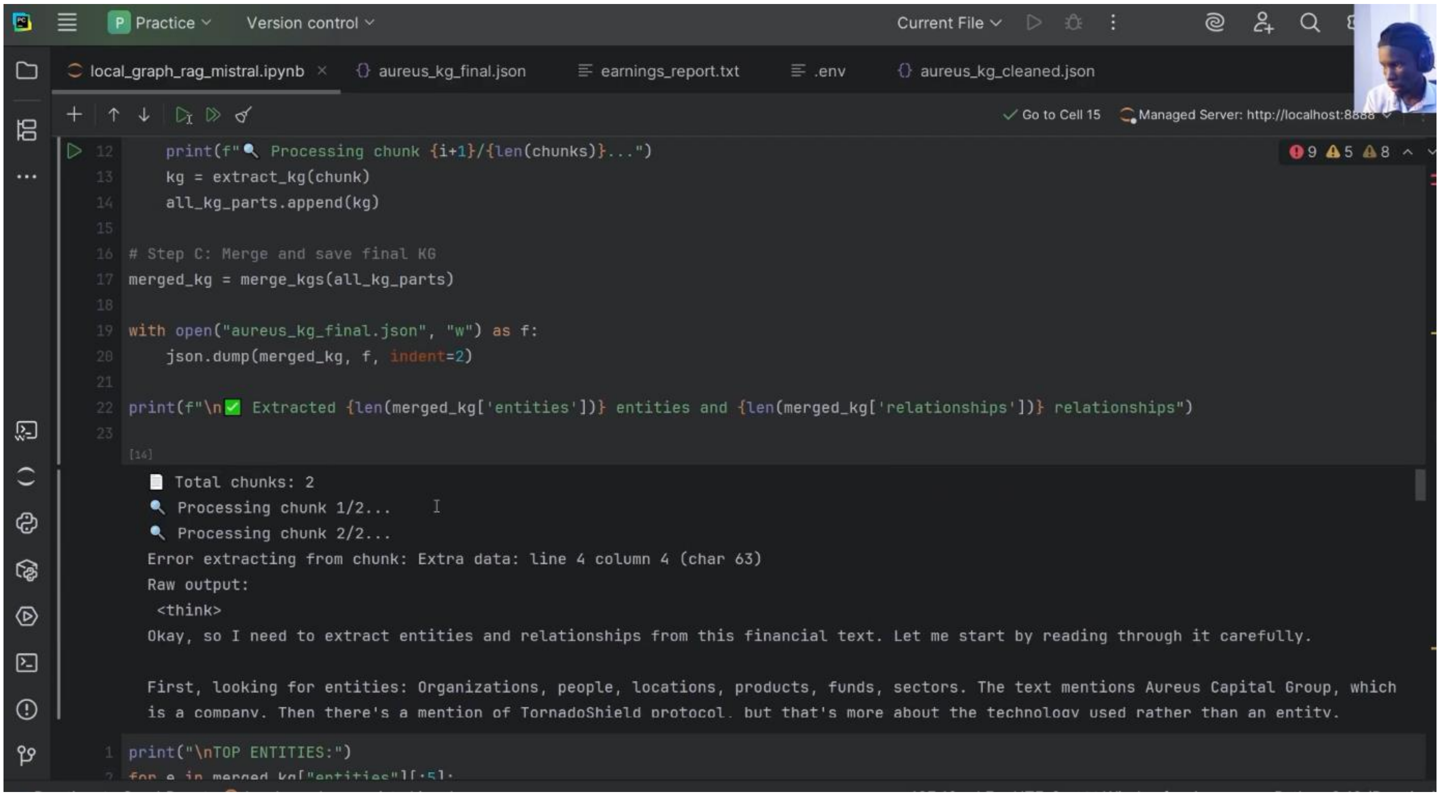
```
✓ [6] 488ms
```

```python
# Load the text
```



Load the text

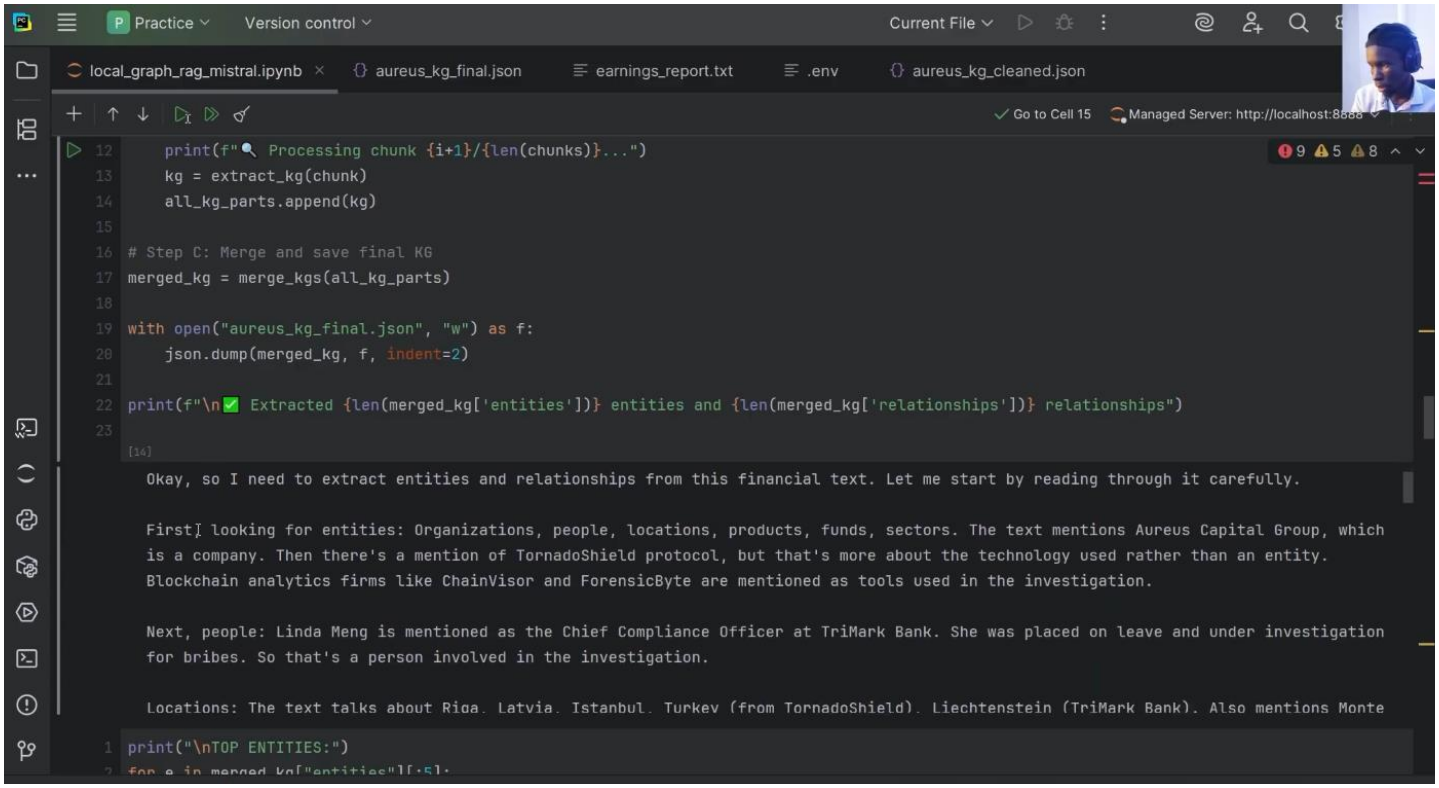
```python
# Load input text
with open("money_laundering.txt", "r", encoding="utf-8") as f:
    full_text = f.read()

# Step A: Chunk the text
chunks = smart_chunk(full_text)
print(f"📄 Total chunks: {len(chunks)}")

# Step B: Process each chunk
all_kg_parts = []
for i, chunk in enumerate(chunks):
    print(f"🔍 Processing chunk {i+1}/{len(chunks)}...")
    kg = extract_kg(chunk)
    all_kg_parts.append(kg)

# Step C: Merge and save final KG
merged_kg = merge_kgs(all_kg_parts)

with open("aureus_kg_final.json", "w") as f:
    json.dump(merged_kg, f, indent=2)
```



```python
    print(f"🔍 Processing chunk {i+1}/{len(chunks)}...")
    kg = extract_kg(chunk)
    all_kg_parts.append(kg)

# Step C: Merge and save final KG
merged_kg = merge_kgs(all_kg_parts)

with open("aureus_kg_final.json", "w") as f:
    json.dump(merged_kg, f, indent=2)

print(f"\n✅ Extracted {len(merged_kg['entities'])} entities and {len(merged_kg['relationships'])} relationships")
```

```
📄 Total chunks: 2
🔍 Processing chunk 1/2...
🔍 Processing chunk 2/2...
Error extracting from chunk: Extra data: line 4 column 4 (char 63)
Raw output:
 <think>
Okay, so I need to extract entities and relationships from this financial text. Let me start by reading through it carefully.

First, looking for entities: Organizations, people, locations, products, funds, sectors. The text mentions Aureus Capital Group, which
is a company. Then there's a mention of TornadoShield protocol, but that's more about the technology used rather than an entity.
```

```python
print("\nTOP ENTITIES:")
for e in merged_kg["entities"][:5]:
```

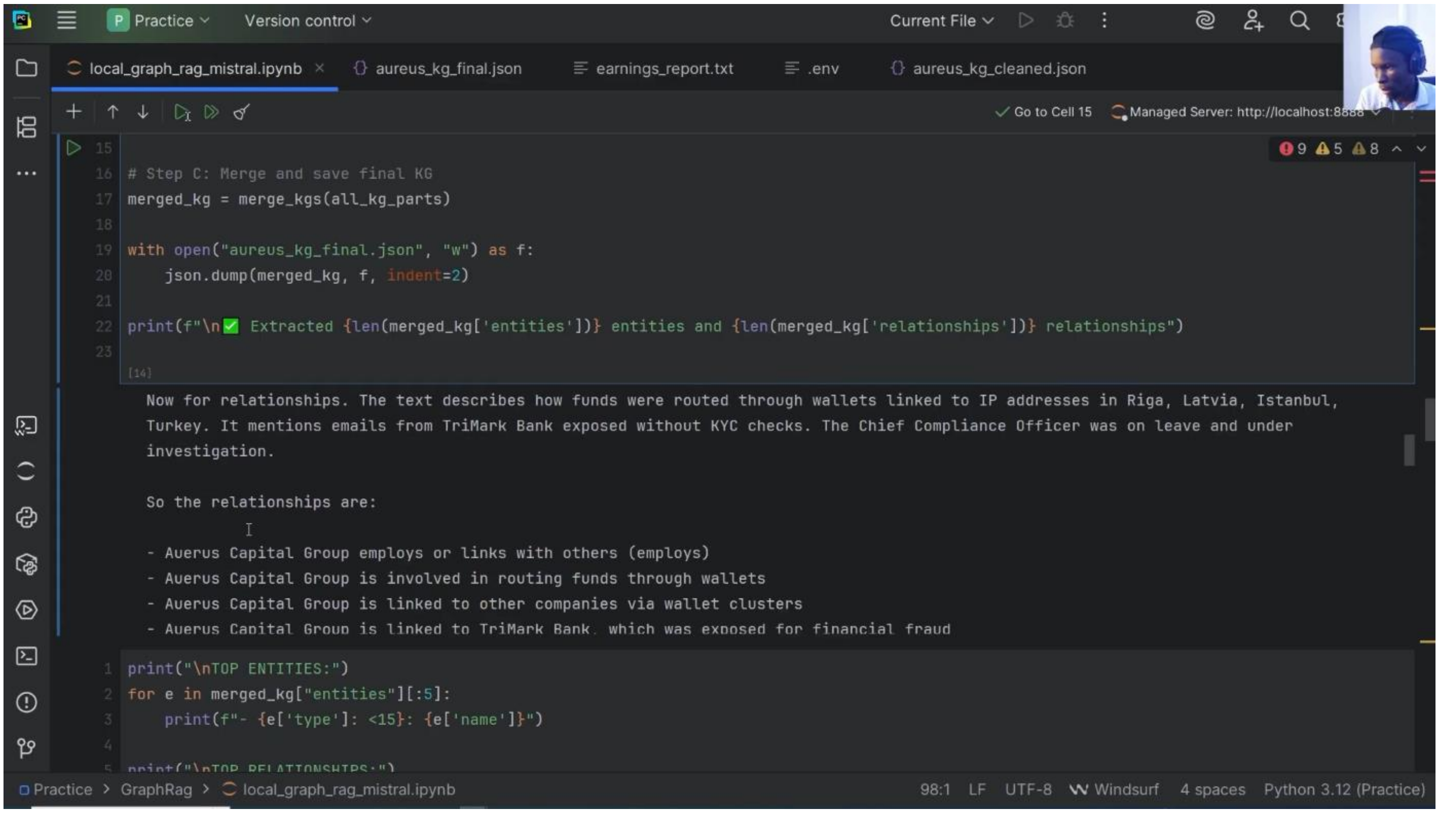


```
    print(f"🔍 Processing chunk {i+1}/{len(chunks)}...")
    kg = extract_kg(chunk)
    all_kg_parts.append(kg)

# Step C: Merge and save final KG
merged_kg = merge_kgs(all_kg_parts)

with open("aureus_kg_final.json", "w") as f:
    json.dump(merged_kg, f, indent=2)

print(f"\n✅ Extracted {len(merged_kg['entities'])} entities and {len(merged_kg['relationships'])} relationships")
```

[14]

```
Okay, so I need to extract entities and relationships from this financial text. Let me start by reading through it carefully.

First, looking for entities: Organizations, people, locations, products, funds, sectors. The text mentions Aureus Capital Group, which
is a company. Then there's a mention of TornadoShield protocol, but that's more about the technology used rather than an entity.
Blockchain analytics firms like ChainVisor and ForensicByte are mentioned as tools used in the investigation.

Next, people: Linda Meng is mentioned as the Chief Compliance Officer at TriMark Bank. She was placed on leave and under investigation
for bribes. So that's a person involved in the investigation.

Locations: The text talks about Riga, Latvia, Istanbul, Turkey (from TornadoShield), Liechtenstein (TriMark Bank). Also mentions Monte
```

```
print("\nTOP ENTITIES:")
for e in merged_kg["entities"][:5]:
```

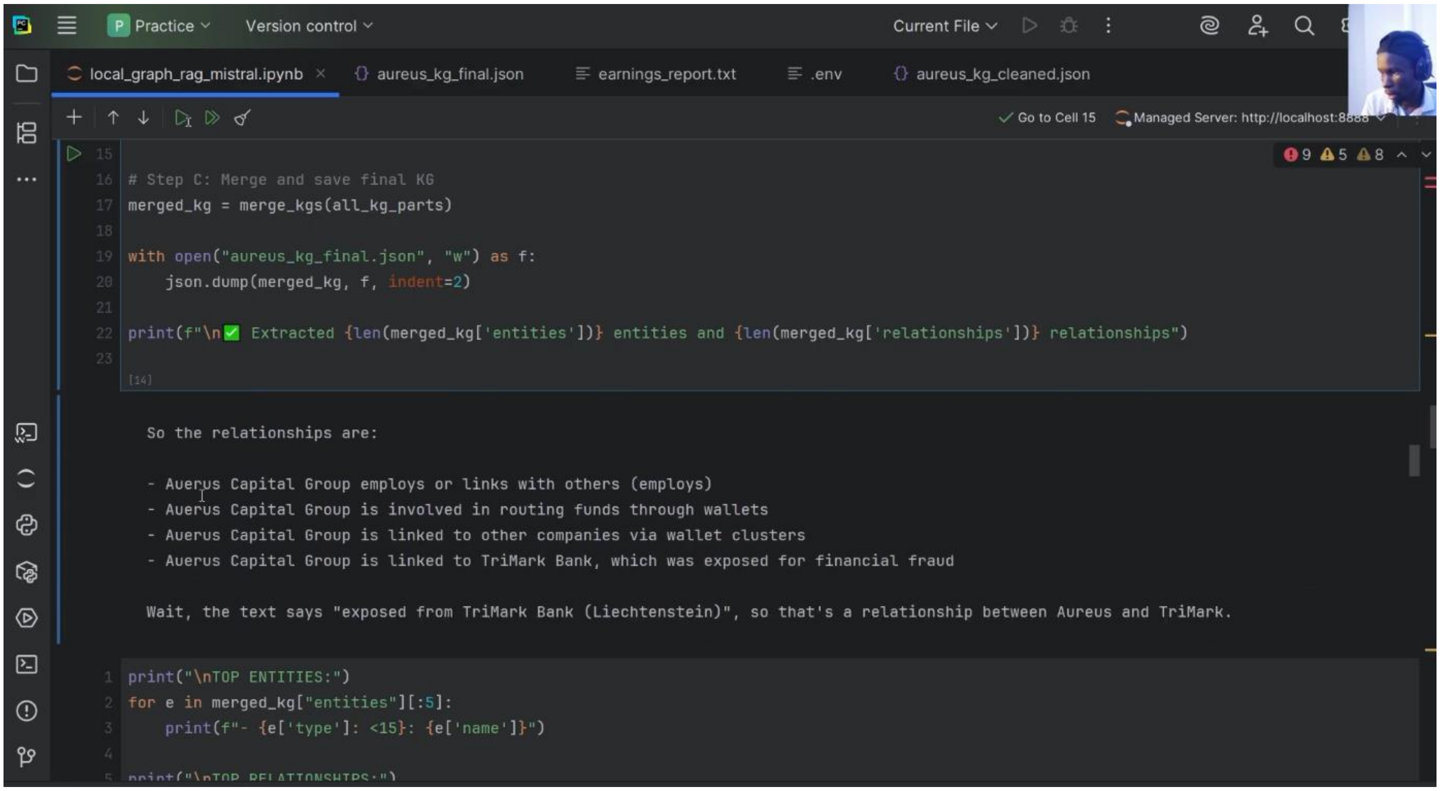

Practice
Version control
Current File
local_graph_rag_mistral.ipynb
aureus_kg_final.json
earnings_report.txt
.env
aureus_kg_cleaned.json
Go to Cell 15
Managed Server: http://localhost:8888
9 5 8
Practice > GraphRag > local_graph_rag_mistral.ipynb
98:1 LF UTF-8 Windsurf 4 spaces Python 3.12 (Practice)

```

# Step C: Merge and save final KG
merged_kg = merge_kgs(all_kg_parts)

with open("aureus_kg_final.json", "w") as f:
    json.dump(merged_kg, f, indent=2)

print(f"\n✅ Extracted {len(merged_kg['entities'])} entities and {len(merged_kg['relationships'])} relationships")
```

[14]

```
Now for relationships. The text describes how funds were routed through wallets linked to IP addresses in Riga, Latvia, Istanbul,
Turkey. It mentions emails from TriMark Bank exposed without KYC checks. The Chief Compliance Officer was on leave and under
investigation.

So the relationships are:

- Auerus Capital Group employs or links with others (employs)
- Auerus Capital Group is involved in routing funds through wallets
- Auerus Capital Group is linked to other companies via wallet clusters
- Auerus Capital Group is linked to TriMark Bank, which was exposed for financial fraud
```

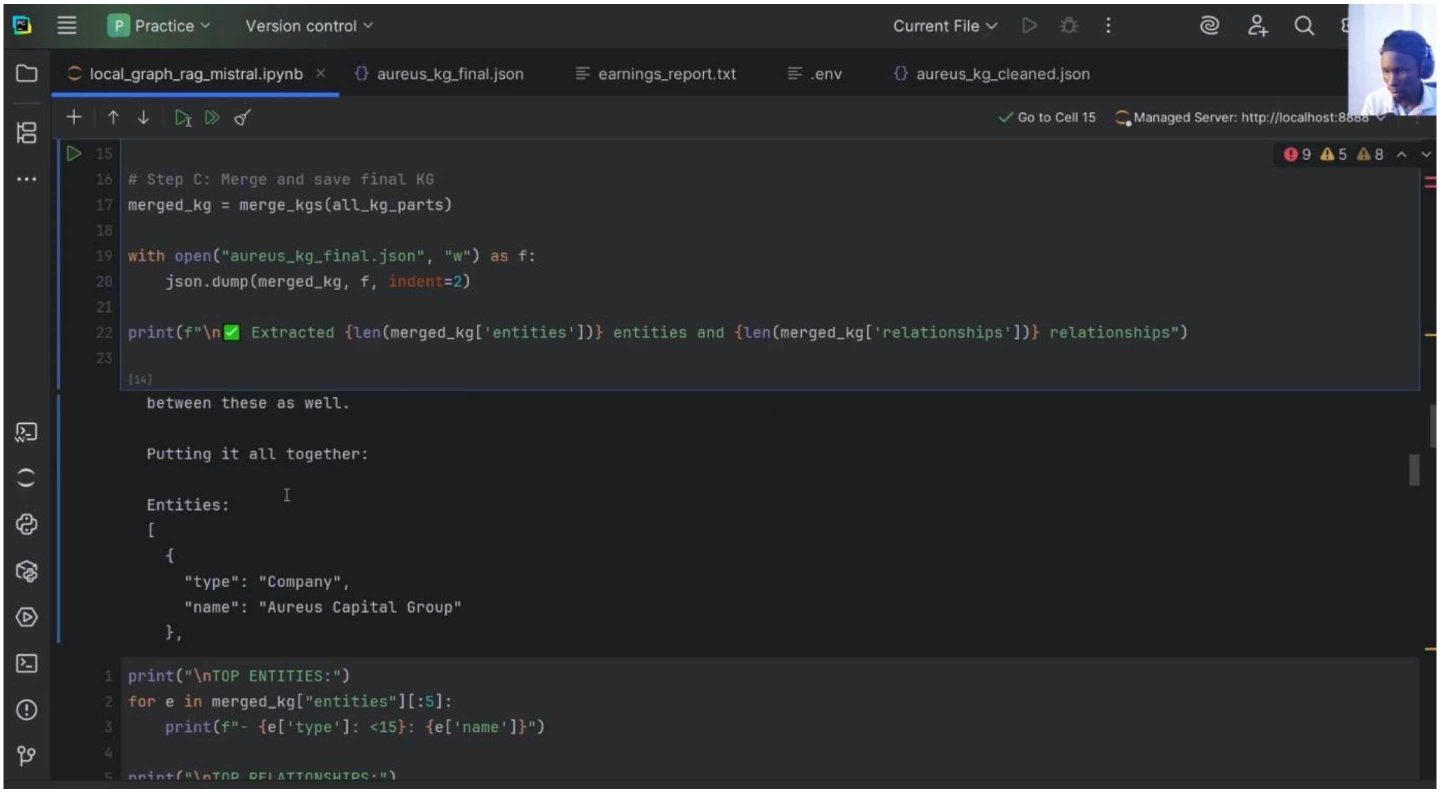
```
print("\nTOP ENTITIES:")
for e in merged_kg["entities"][:5]:
    print(f"- {e['type']: <15}: {e['name']}")

print("\nTOP RELATIONSHIPS:")
```



```python

# Step C: Merge and save final KG
merged_kg = merge_kgs(all_kg_parts)

with open("aureus_kg_final.json", "w") as f:
    json.dump(merged_kg, f, indent=2)

print(f"\n✅ Extracted {len(merged_kg['entities'])} entities and {len(merged_kg['relationships'])} relationships")
```

```
So the relationships are:

- Auerus Capital Group employs or links with others (employs)
- Auerus Capital Group is involved in routing funds through wallets
- Auerus Capital Group is linked to other companies via wallet clusters
- Auerus Capital Group is linked to TriMark Bank, which was exposed for financial fraud

Wait, the text says "exposed from TriMark Bank (Liechtenstein)", so that's a relationship between Aureus and TriMark.
```

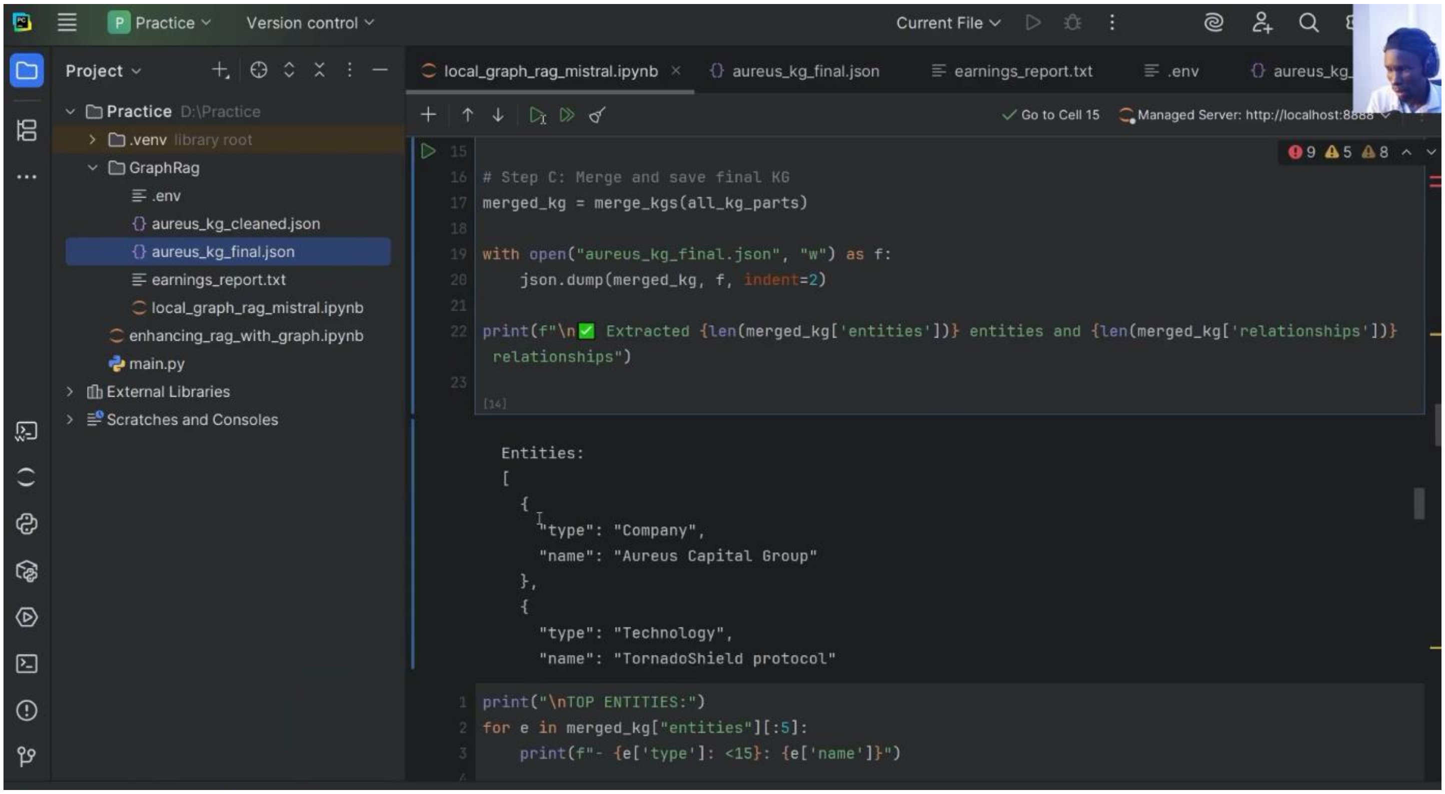
```python
print("\nTOP ENTITIES:")
for e in merged_kg["entities"][:5]:
    print(f"- {e['type']: <15}: {e['name']}")

print("\nTOP RELATIONSHIPS:")
```



```python

# Step C: Merge and save final KG
merged_kg = merge_kgs(all_kg_parts)

with open("aureus_kg_final.json", "w") as f:
    json.dump(merged_kg, f, indent=2)

print(f"\n✅ Extracted {len(merged_kg['entities'])} entities and {len(merged_kg['relationships'])} relationships")
```

```
between these as well.

Putting it all together:

Entities:
[
  {
    "type": "Company",
    "name": "Aureus Capital Group"
  },
```

```python
print("\nTOP ENTITIES:")
for e in merged_kg["entities"][:5]:
    print(f"- {e['type']: <15}: {e['name']}")

print("\nTOP RELATIONSHIPS:")
```

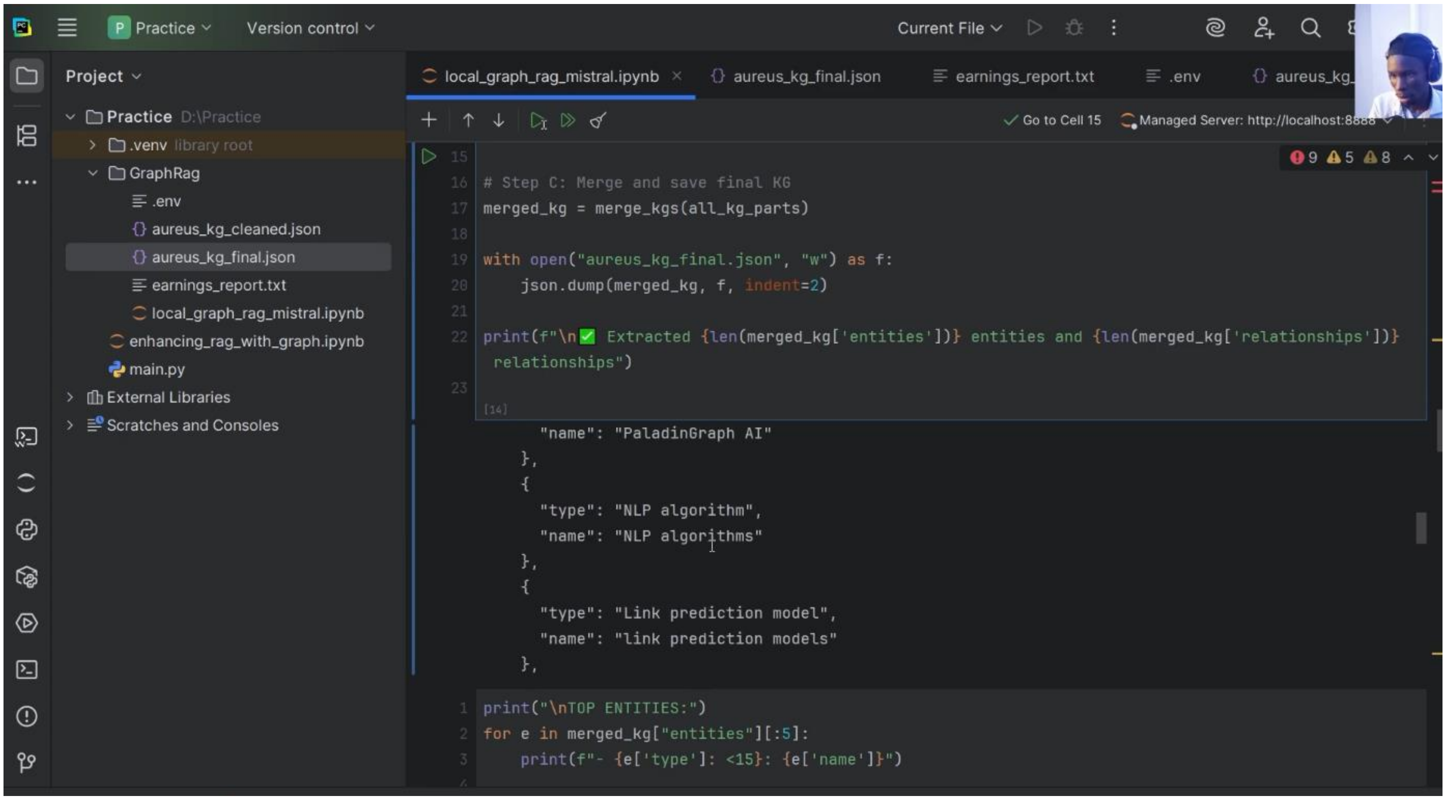


```
# Step C: Merge and save final KG
merged_kg = merge_kgs(all_kg_parts)

with open("aureus_kg_final.json", "w") as f:
    json.dump(merged_kg, f, indent=2)

print(f"\n✅ Extracted {len(merged_kg['entities'])} entities and {len(merged_kg['relationships'])}
 relationships")
```

[14]

```
Entities:
[
  {
    "type": "Company",
    "name": "Aureus Capital Group"
  },
  {
    "type": "Technology",
    "name": "TornadoShield protocol"
```

```
print("\nTOP ENTITIES:")
for e in merged_kg["entities"][:5]:
    print(f"- {e['type']: <15}: {e['name']}")
```

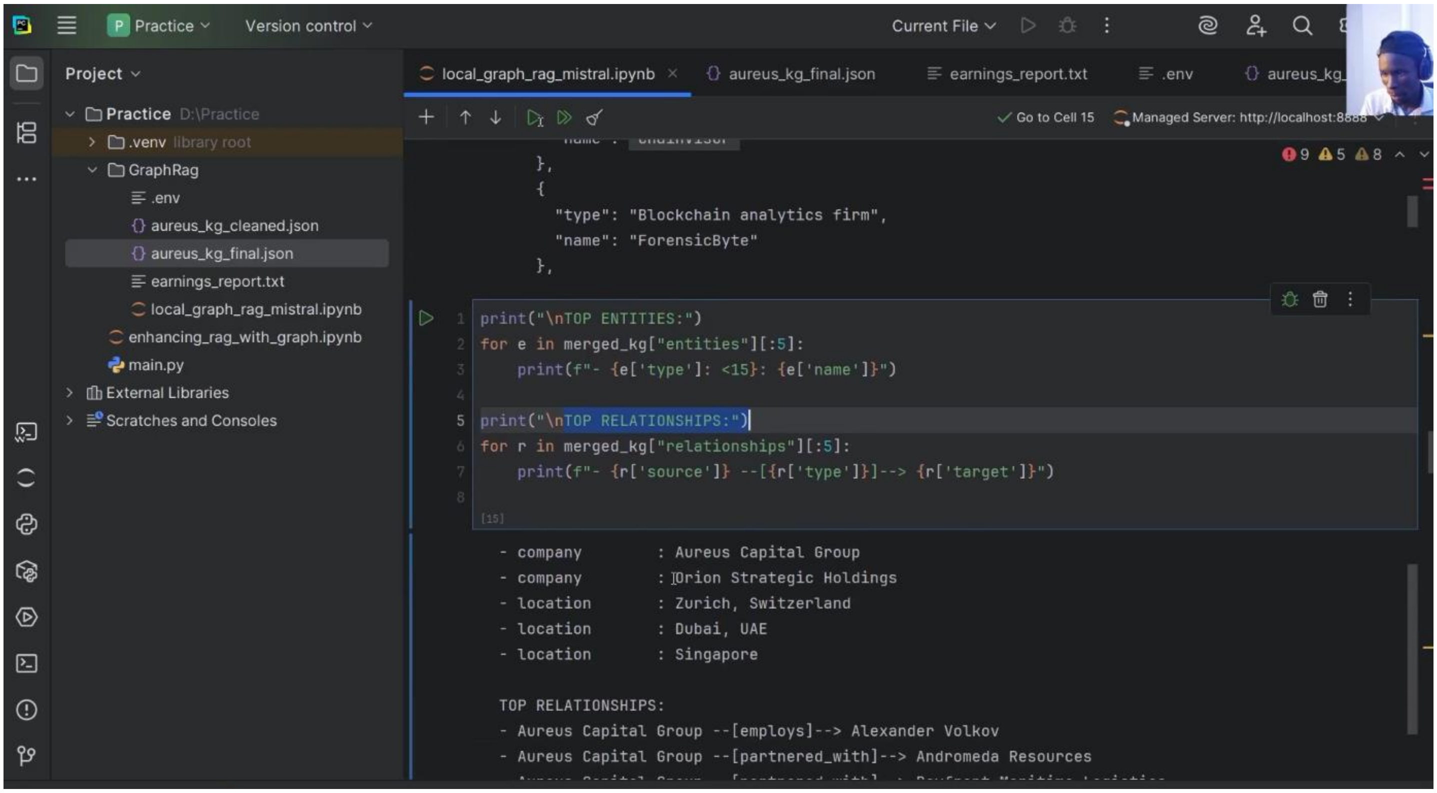


```
# Step C: Merge and save final KG
merged_kg = merge_kgs(all_kg_parts)

with open("aureus_kg_final.json", "w") as f:
    json.dump(merged_kg, f, indent=2)

print(f"\n✅ Extracted {len(merged_kg['entities'])} entities and {len(merged_kg['relationships'])}
 relationships")
```

[14]

```
    "name": "PaladinGraph AI"
  },
  {
    "type": "NLP algorithm",
    "name": "NLP algorithms"
  },
  {
    "type": "Link prediction model",
    "name": "link prediction models"
  },
```

```
print("\nTOP ENTITIES:")
for e in merged_kg["entities"][:5]:
    print(f"- {e['type']: <15}: {e['name']}")
```

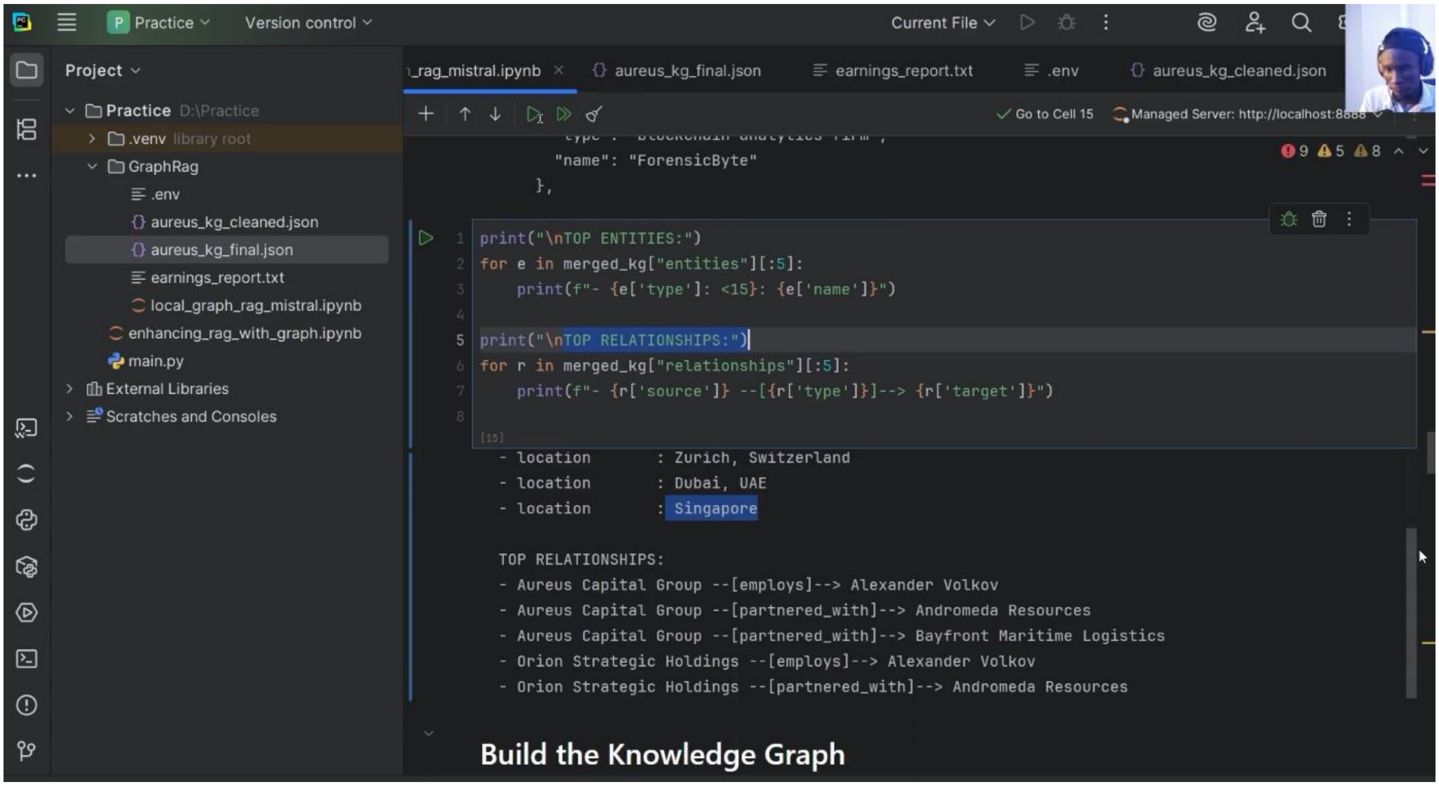


```
        },
        {
          "type": "Blockchain analytics firm",
          "name": "ForensicByte"
        },
```

```python
print("\nTOP ENTITIES:")
for e in merged_kg["entities"][:5]:
    print(f"- {e['type']: <15}: {e['name']}")

print("\nTOP RELATIONSHIPS:")
for r in merged_kg["relationships"][:5]:
    print(f"- {r['source']} --[{r['type']}]--> {r['target']}")
```

[15]

```
- company        : Aureus Capital Group
- company        : Orion Strategic Holdings
- location       : Zurich, Switzerland
- location       : Dubai, UAE
- location       : Singapore

TOP RELATIONSHIPS:
- Aureus Capital Group --[employs]--> Alexander Volkov
- Aureus Capital Group --[partnered_with]--> Andromeda Resources
```



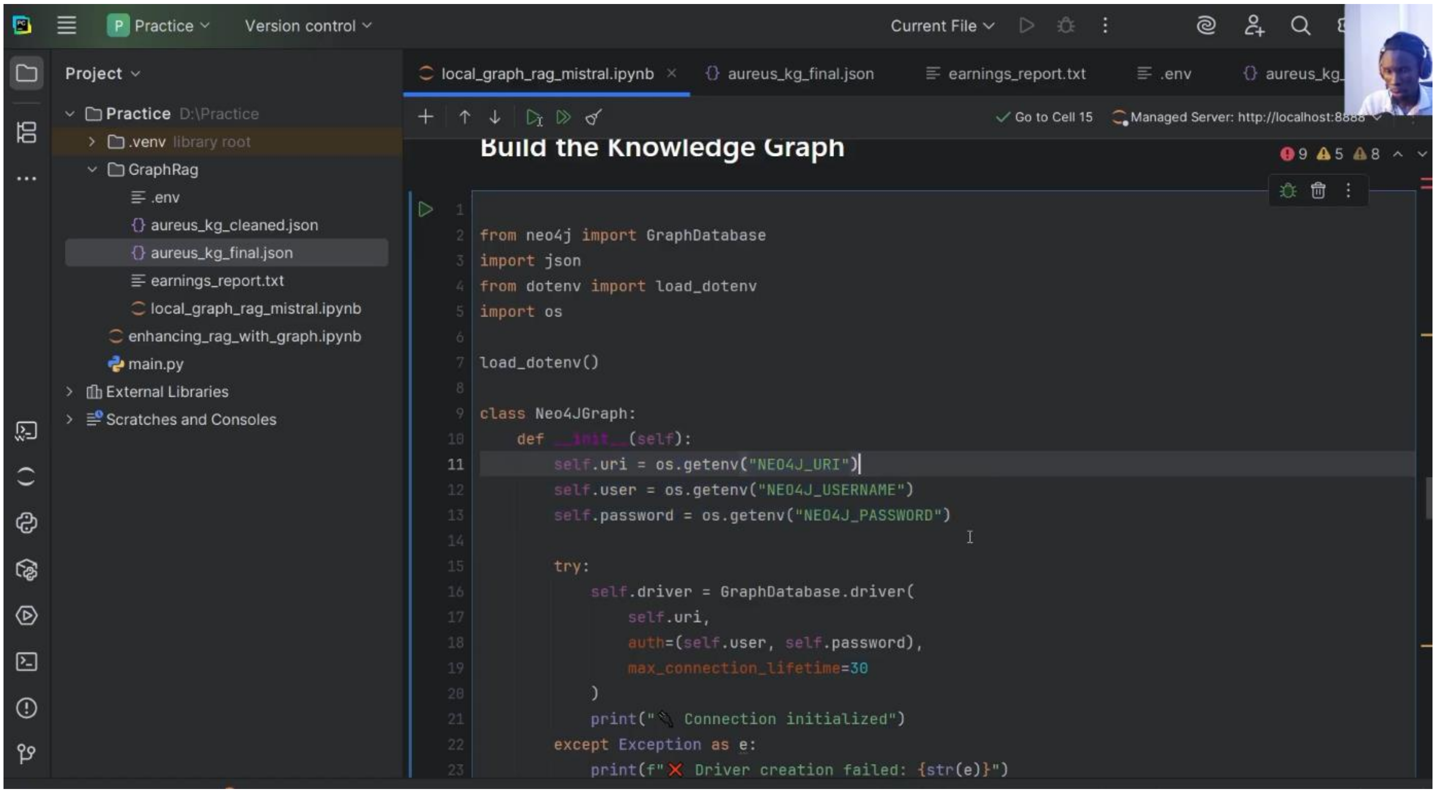
```
          "name": "ForensicByte"
        },
```

```python
print("\nTOP ENTITIES:")
for e in merged_kg["entities"][:5]:
    print(f"- {e['type']: <15}: {e['name']}")

print("\nTOP RELATIONSHIPS:")
for r in merged_kg["relationships"][:5]:
    print(f"- {r['source']} --[{r['type']}]--> {r['target']}")
```

[15]

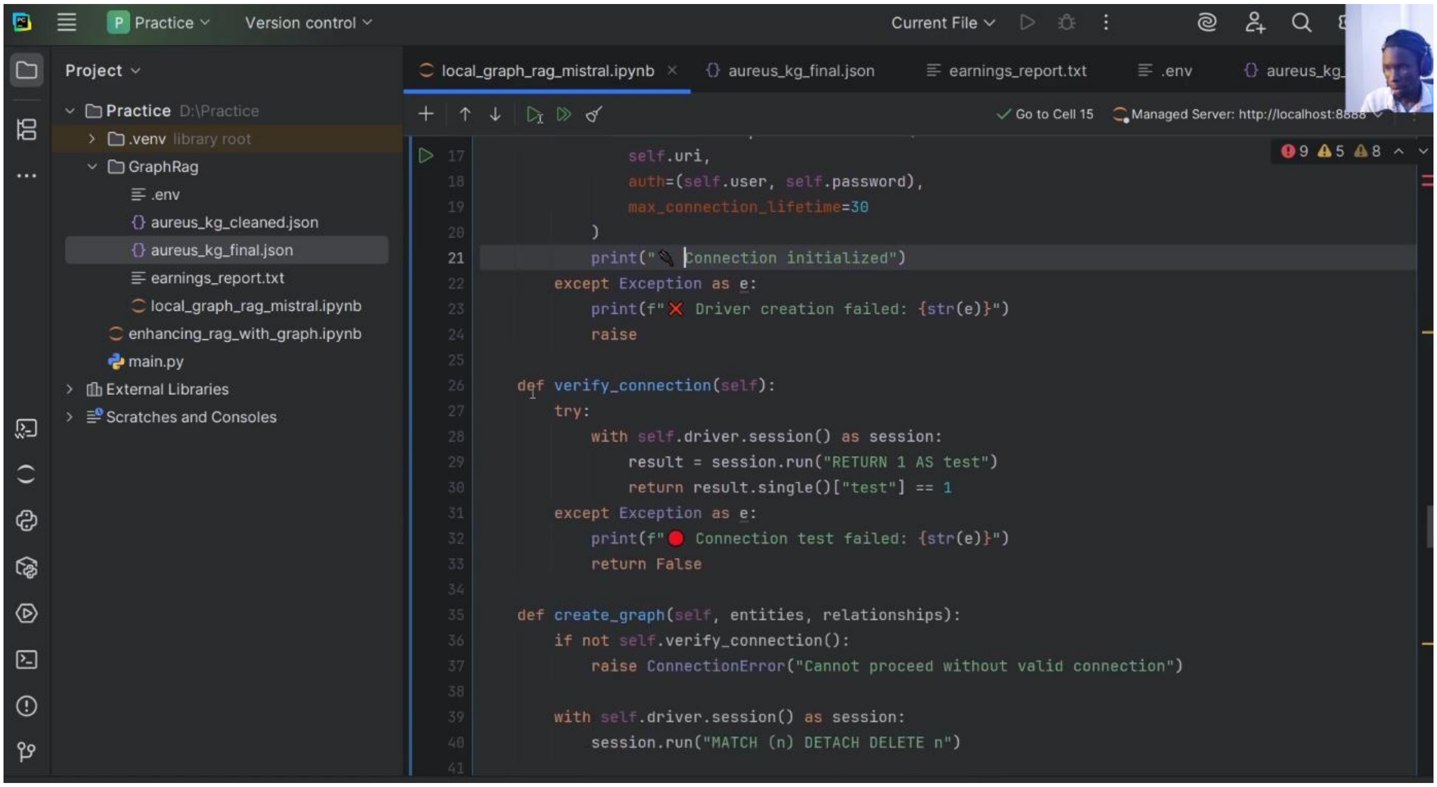
```
- location       : Zurich, Switzerland
- location       : Dubai, UAE
- location       : Singapore

TOP RELATIONSHIPS:
- Aureus Capital Group --[employs]--> Alexander Volkov
- Aureus Capital Group --[partnered_with]--> Andromeda Resources
- Aureus Capital Group --[partnered_with]--> Bayfront Maritime Logistics
- Orion Strategic Holdings --[employs]--> Alexander Volkov
- Orion Strategic Holdings --[partnered_with]--> Andromeda Resources
```

Build the Knowledge Graph

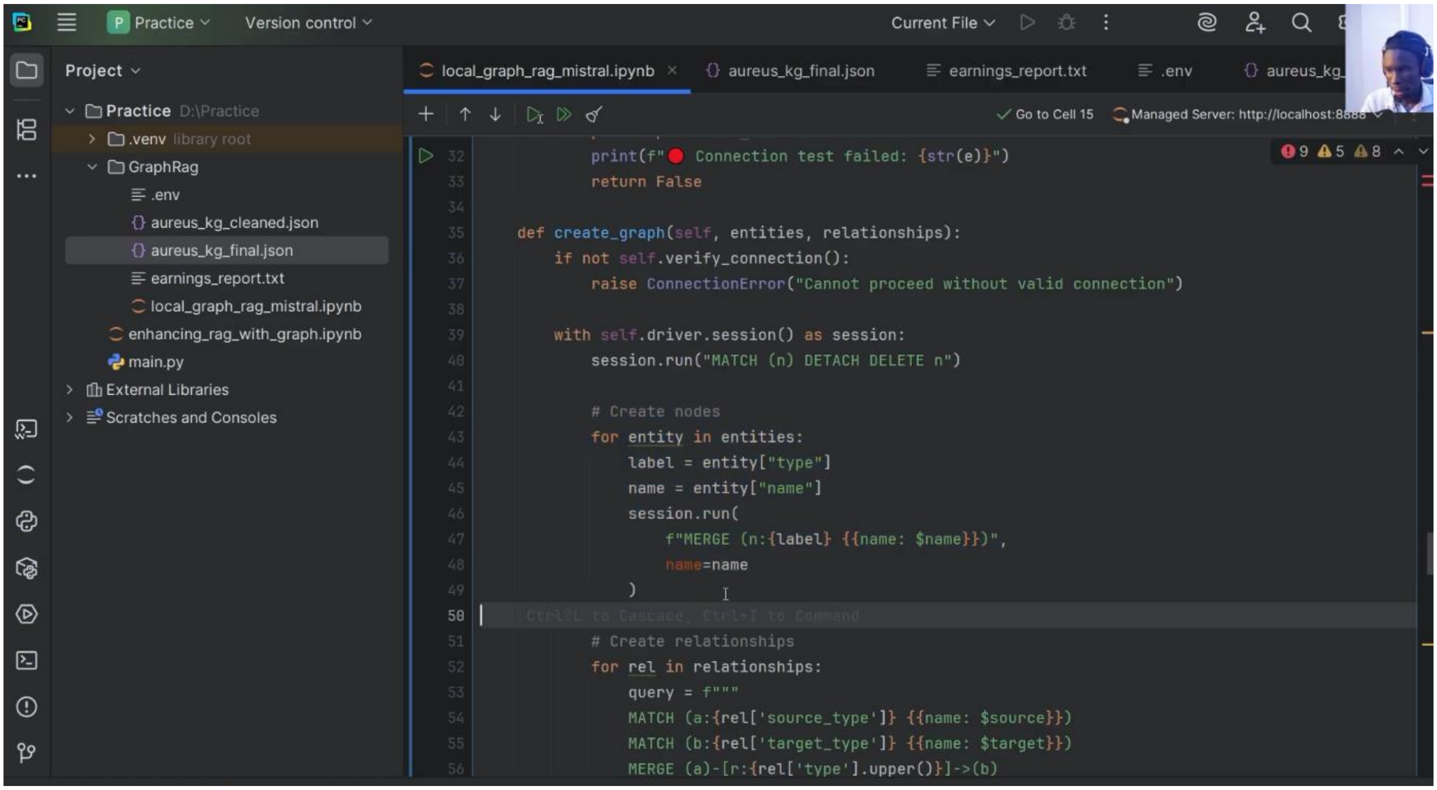


Build the Knowledge Graph

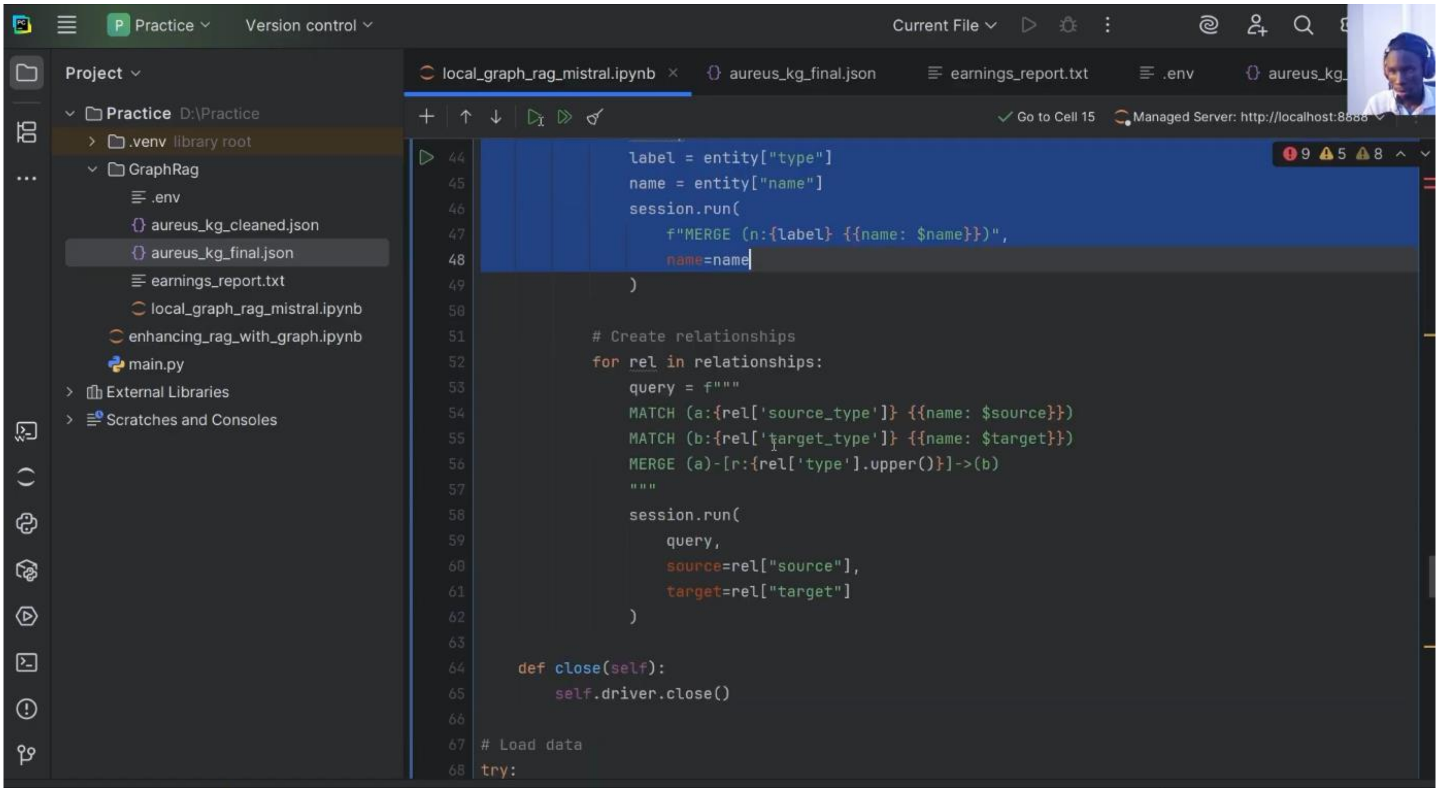
```python

from neo4j import GraphDatabase
import json
from dotenv import load_dotenv
import os

load_dotenv()

class Neo4JGraph:
    def __init__(self):
        self.uri = os.getenv("NEO4J_URI")
        self.user = os.getenv("NEO4J_USERNAME")
        self.password = os.getenv("NEO4J_PASSWORD")

        try:
            self.driver = GraphDatabase.driver(
                self.uri,
                auth=(self.user, self.password),
                max_connection_lifetime=30
            )
            print("✎ Connection initialized")
        except Exception as e:
            print(f"❌ Driver creation failed: {str(e)}")
```



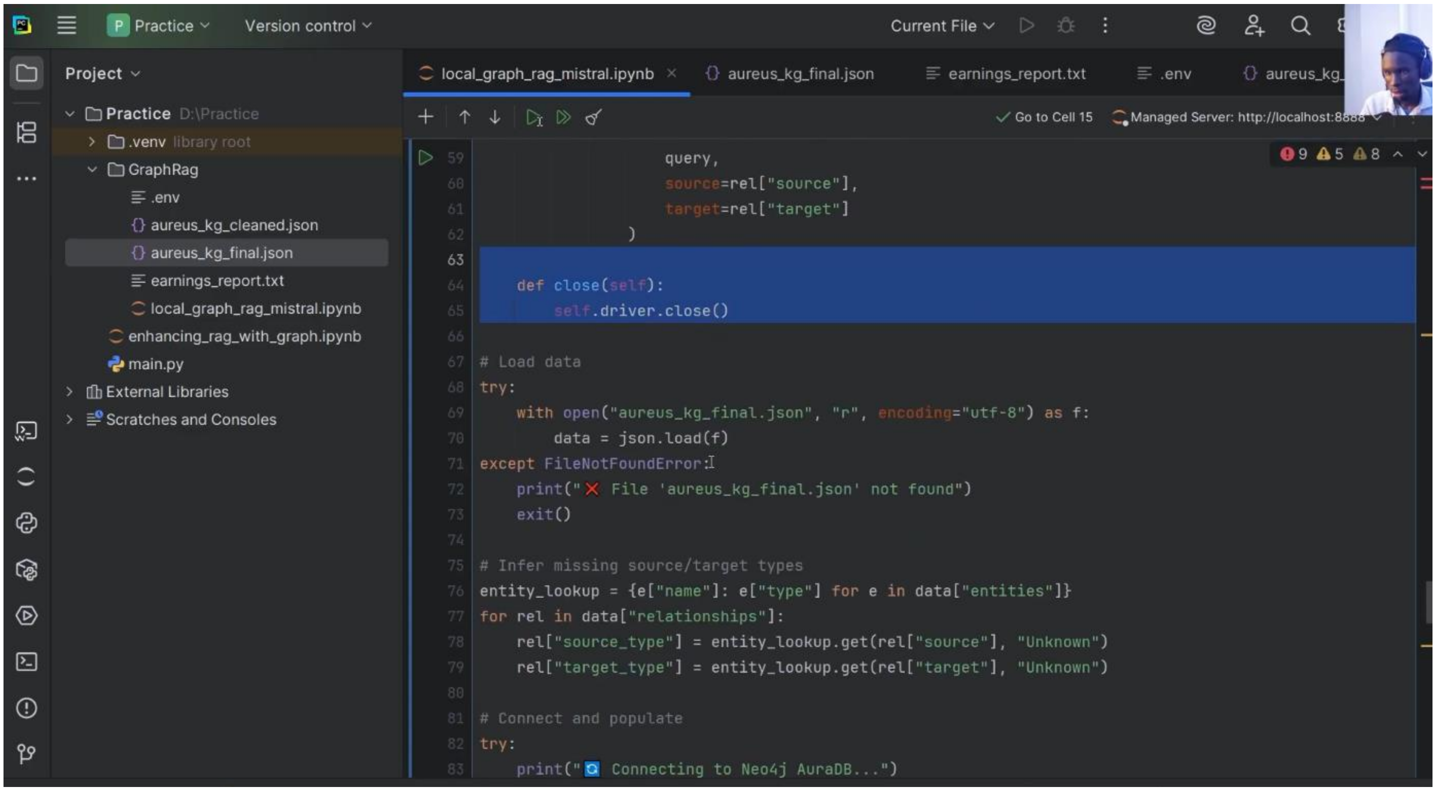
```python
                self.uri,
                auth=(self.user, self.password),
                max_connection_lifetime=30
            )
            print("✎ Connection initialized")
        except Exception as e:
            print(f"❌ Driver creation failed: {str(e)}")
            raise

    def verify_connection(self):
        try:
            with self.driver.session() as session:
                result = session.run("RETURN 1 AS test")
                return result.single()["test"] == 1
        except Exception as e:
            print(f"🔴 Connection test failed: {str(e)}")
            return False

    def create_graph(self, entities, relationships):
        if not self.verify_connection():
            raise ConnectionError("Cannot proceed without valid connection")

        with self.driver.session() as session:
            session.run("MATCH (n) DETACH DELETE n")

```



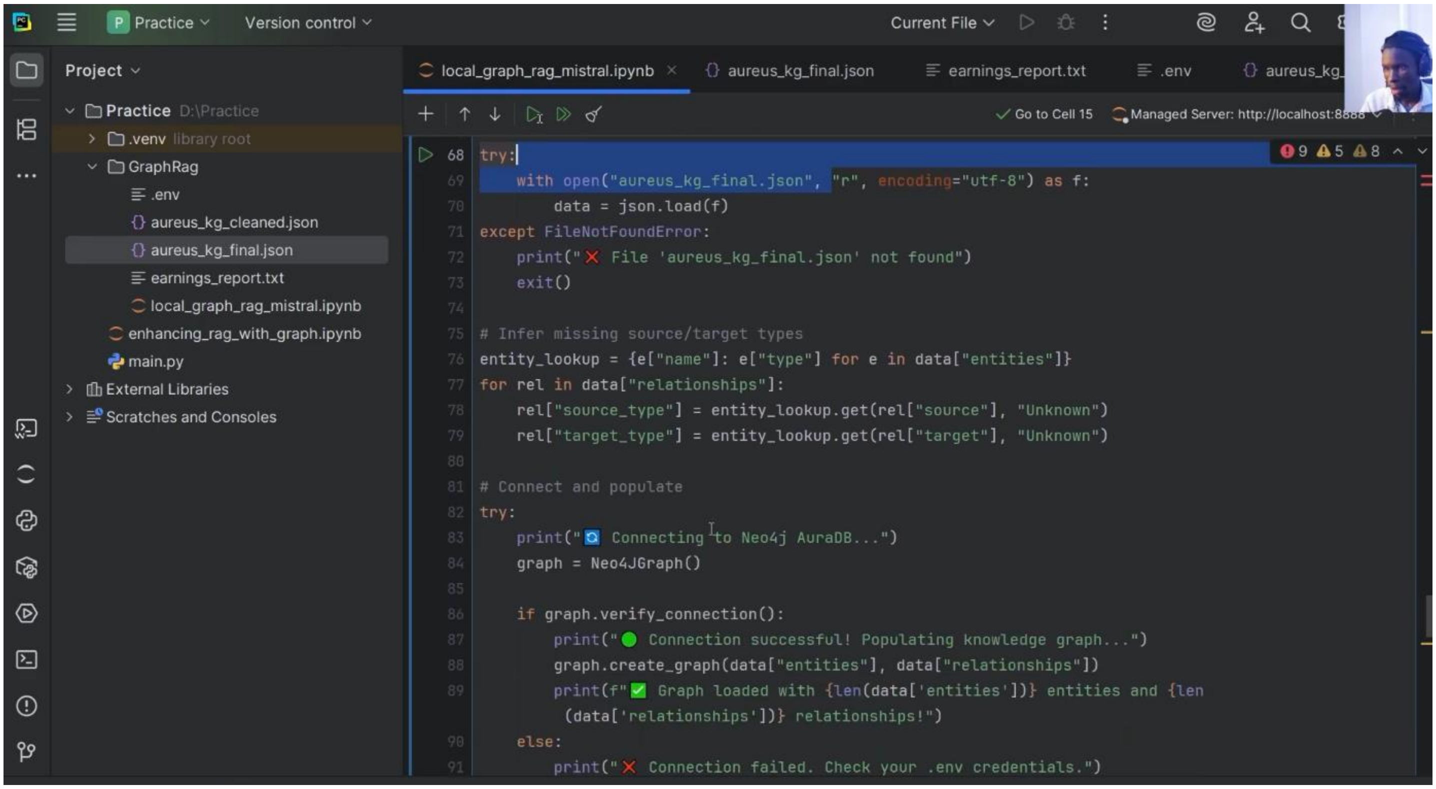
```
            print(f"🔴 Connection test failed: {str(e)}")
            return False

    def create_graph(self, entities, relationships):
        if not self.verify_connection():
            raise ConnectionError("Cannot proceed without valid connection")

        with self.driver.session() as session:
            session.run("MATCH (n) DETACH DELETE n")

            # Create nodes
            for entity in entities:
                label = entity["type"]
                name = entity["name"]
                session.run(
                    f"MERGE (n:{label} {{name: $name}})",
                    name=name
                )
  Ctrl+L to Cascade, Ctrl+I to Command
            # Create relationships
            for rel in relationships:
                query = f"""
                MATCH (a:{rel['source_type']} {{name: $source}})
                MATCH (b:{rel['target_type']} {{name: $target}})
                MERGE (a)-[r:{rel['type'].upper()}]->(b)
```



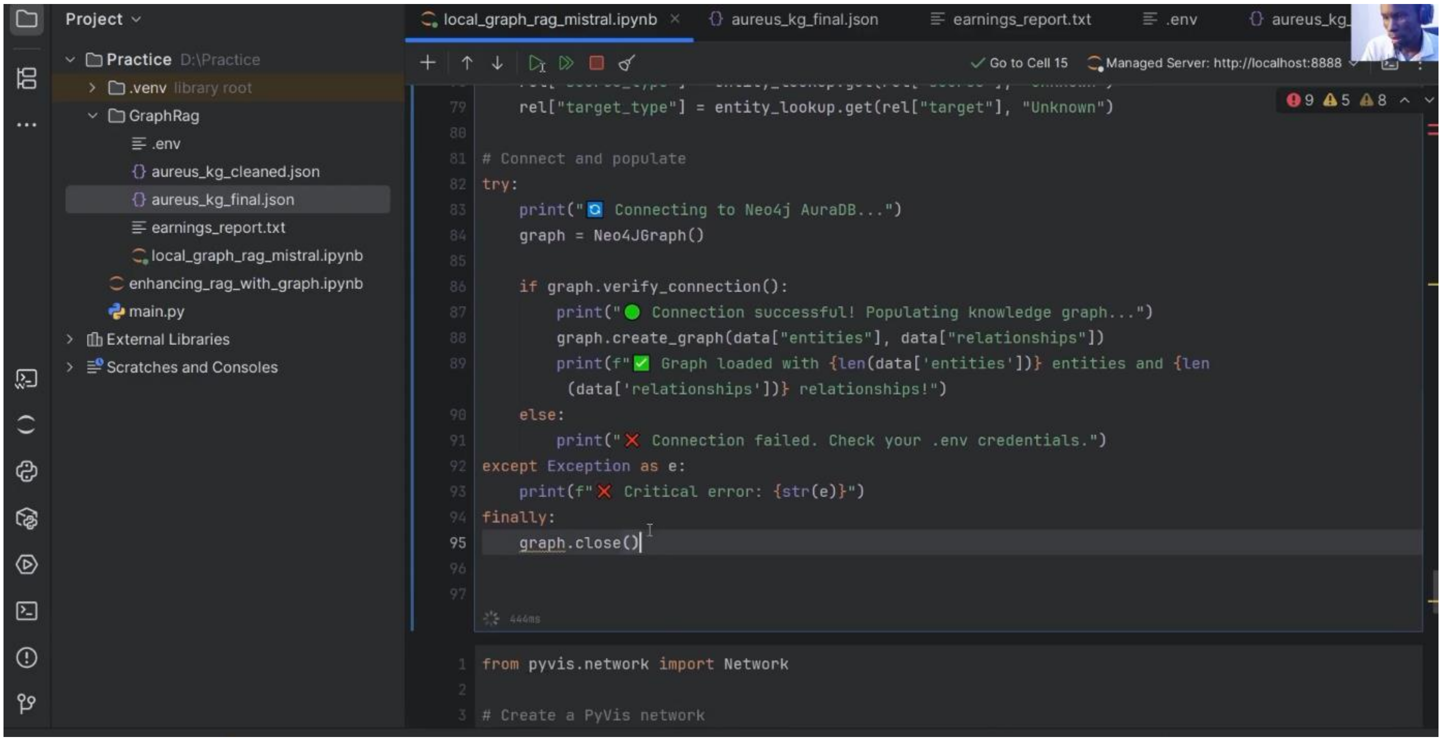
```
                label = entity["type"]
                name = entity["name"]
                session.run(
                    f"MERGE (n:{label} {{name: $name}})",
                    name=name
                )

            # Create relationships
            for rel in relationships:
                query = f"""
                MATCH (a:{rel['source_type']} {{name: $source}})
                MATCH (b:{rel['target_type']} {{name: $target}})
                MERGE (a)-[r:{rel['type'].upper()}]->(b)
                """
                session.run(
                    query,
                    source=rel["source"],
                    target=rel["target"]
                )

    def close(self):
        self.driver.close()

# Load data
try:
```



```
                    query,
                    source=rel["source"],
                    target=rel["target"]
                )

    def close(self):
        self.driver.close()

# Load data
try:
    with open("aureus_kg_final.json", "r", encoding="utf-8") as f:
        data = json.load(f)
except FileNotFoundError:
    print("❌ File 'aureus_kg_final.json' not found")
    exit()

# Infer missing source/target types
entity_lookup = {e["name"]: e["type"] for e in data["entities"]}
for rel in data["relationships"]:
    rel["source_type"] = entity_lookup.get(rel["source"], "Unknown")
    rel["target_type"] = entity_lookup.get(rel["target"], "Unknown")

# Connect and populate
try:
    print("🔄 Connecting to Neo4j AuraDB...")
```



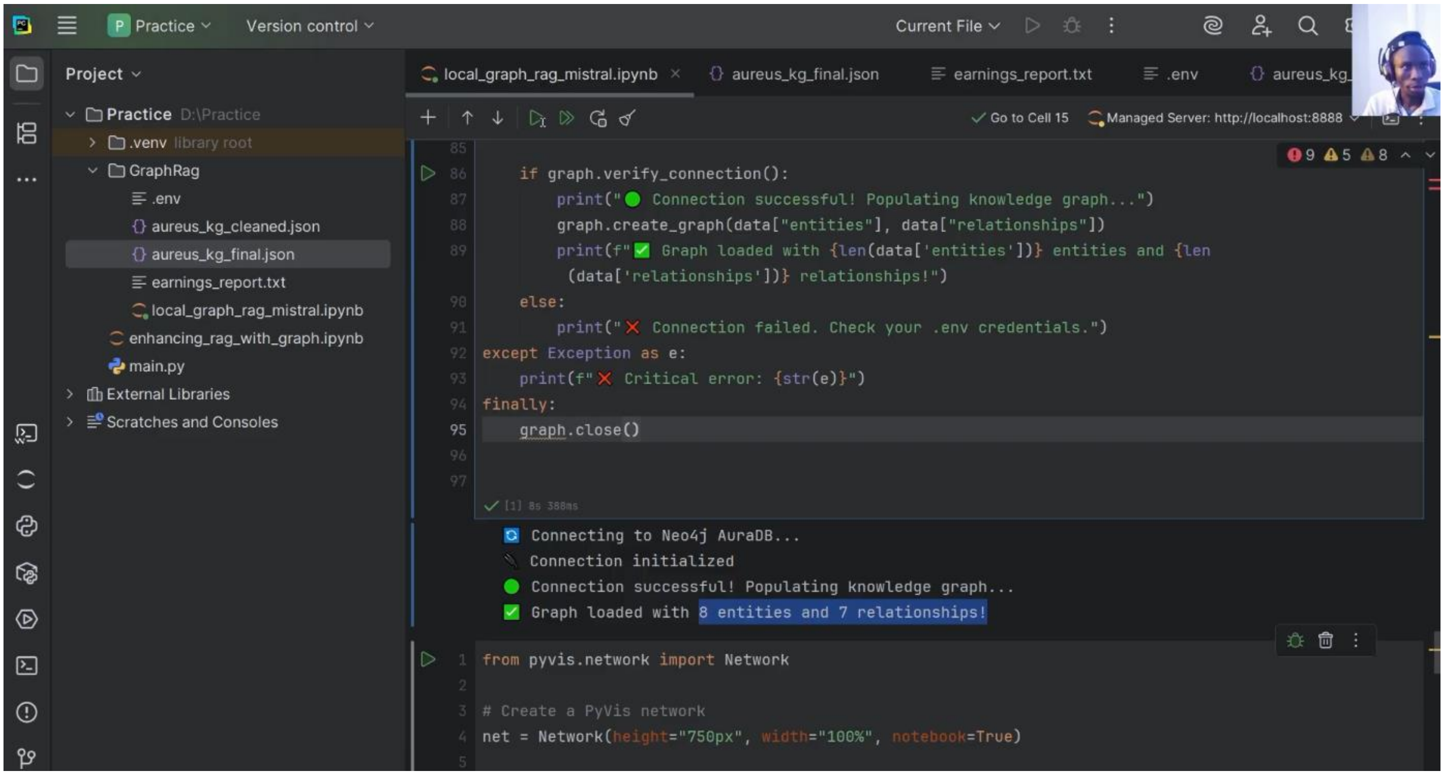
```
try:
    with open("aureus_kg_final.json", "r", encoding="utf-8") as f:
        data = json.load(f)
except FileNotFoundError:
    print("❌ File 'aureus_kg_final.json' not found")
    exit()

# Infer missing source/target types
entity_lookup = {e["name"]: e["type"] for e in data["entities"]}
for rel in data["relationships"]:
    rel["source_type"] = entity_lookup.get(rel["source"], "Unknown")
    rel["target_type"] = entity_lookup.get(rel["target"], "Unknown")

# Connect and populate
try:
    print("🔄 Connecting to Neo4j AuraDB...")
    graph = Neo4JGraph()

    if graph.verify_connection():
        print("🟢 Connection successful! Populating knowledge graph...")
        graph.create_graph(data["entities"], data["relationships"])
        print(f"✅ Graph loaded with {len(data['entities'])} entities and {len(data['relationships'])} relationships!")
    else:
        print("❌ Connection failed. Check your .env credentials.")
```



```
    rel["source_type"] = entity_lookup.get(rel["source"], "Unknown")
    rel["target_type"] = entity_lookup.get(rel["target"], "Unknown")

# Connect and populate
try:
    print("🔄 Connecting to Neo4j AuraDB...")
    graph = Neo4JGraph()

    if graph.verify_connection():
        print("🟢 Connection successful! Populating knowledge graph...")
        graph.create_graph(data["entities"], data["relationships"])
        print(f"✅ Graph loaded with {len(data['entities'])} entities and {len
         (data['relationships'])} relationships!")
    else:
        print("❌ Connection failed. Check your .env credentials.")
except Exception as e:
    print(f"❌ Critical error: {str(e)}")
finally:
    graph.close()
```

446ms

```
from pyvis.network import Network

# Create a PyVis network
```

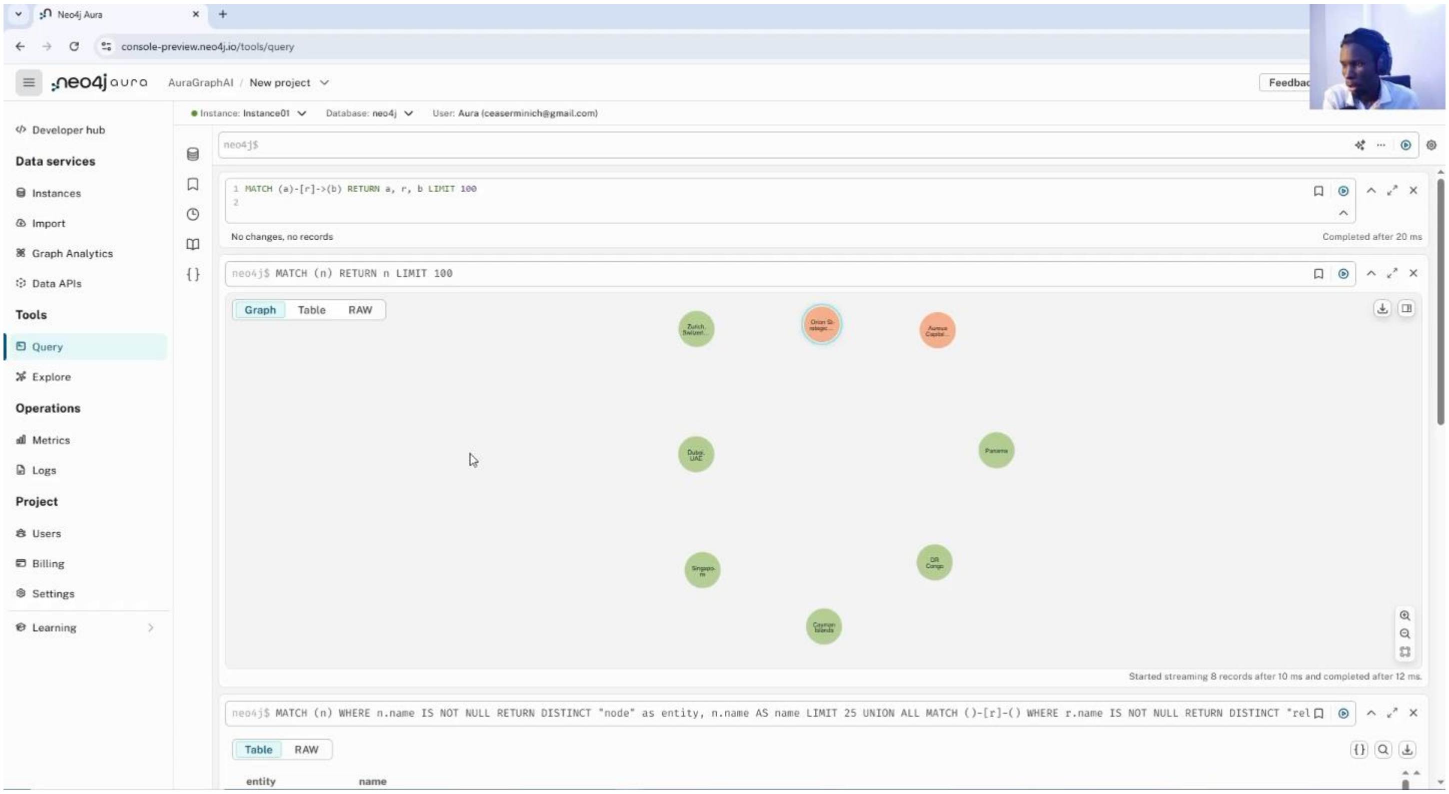


```
    if graph.verify_connection():
        print("🟢 Connection successful! Populating knowledge graph...")
        graph.create_graph(data["entities"], data["relationships"])
        print(f"✅ Graph loaded with {len(data['entities'])} entities and {len
         (data['relationships'])} relationships!")
    else:
        print("❌ Connection failed. Check your .env credentials.")
except Exception as e:
    print(f"❌ Critical error: {str(e)}")
finally:
    graph.close()
```

✓ [1] 8s 388ms

```
🔄 Connecting to Neo4j AuraDB...
  Connection initialized
🟢 Connection successful! Populating knowledge graph...
✅ Graph loaded with 8 entities and 7 relationships!
```

```
from pyvis.network import Network

# Create a PyVis network
net = Network(height="750px", width="100%", notebook=True)
```



```
neo4j$
```

```
1 MATCH (a)-[r]->(b) RETURN a, r, b LIMIT 100
2
```

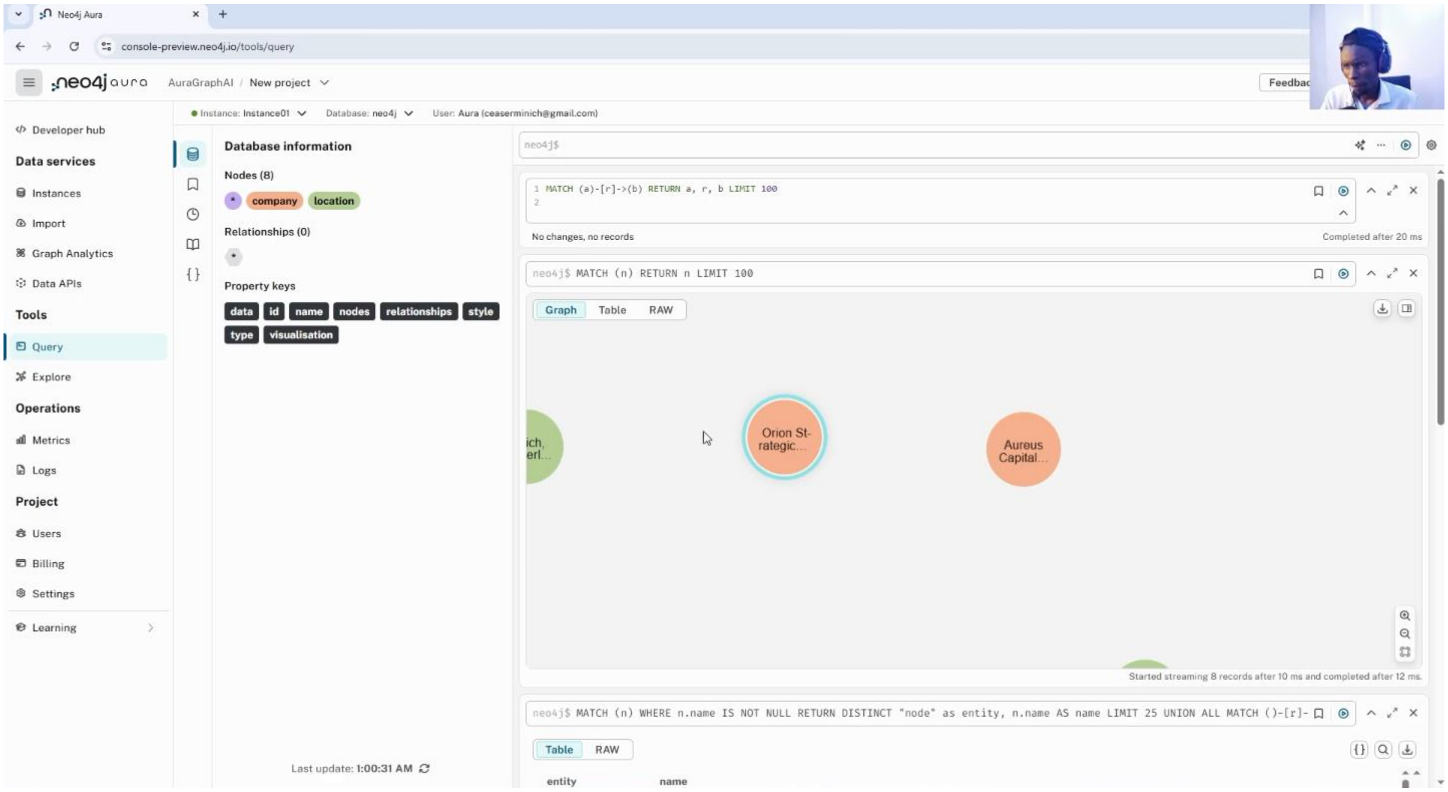
No changes, no records

Completed after 20 ms

```
neo4j$ MATCH (n) RETURN n LIMIT 100
```

Started streaming 8 records after 10 ms and completed after 12 ms.

```
neo4j$ MATCH (n) WHERE n.name IS NOT NULL RETURN DISTINCT "node" as entity, n.name AS name LIMIT 25 UNION ALL MATCH ()-[r]-() WHERE r.name IS NOT NULL RETURN DISTINCT "rel
```



Database information

Nodes (8)

* company location

Relationships (0)

*

Property keys

data id name nodes relationships style type visualisation

Last update: 1:00:31 AM

```
neo4j$
```

```
1 MATCH (a)-[r]->(b) RETURN a, r, b LIMIT 100
2
```

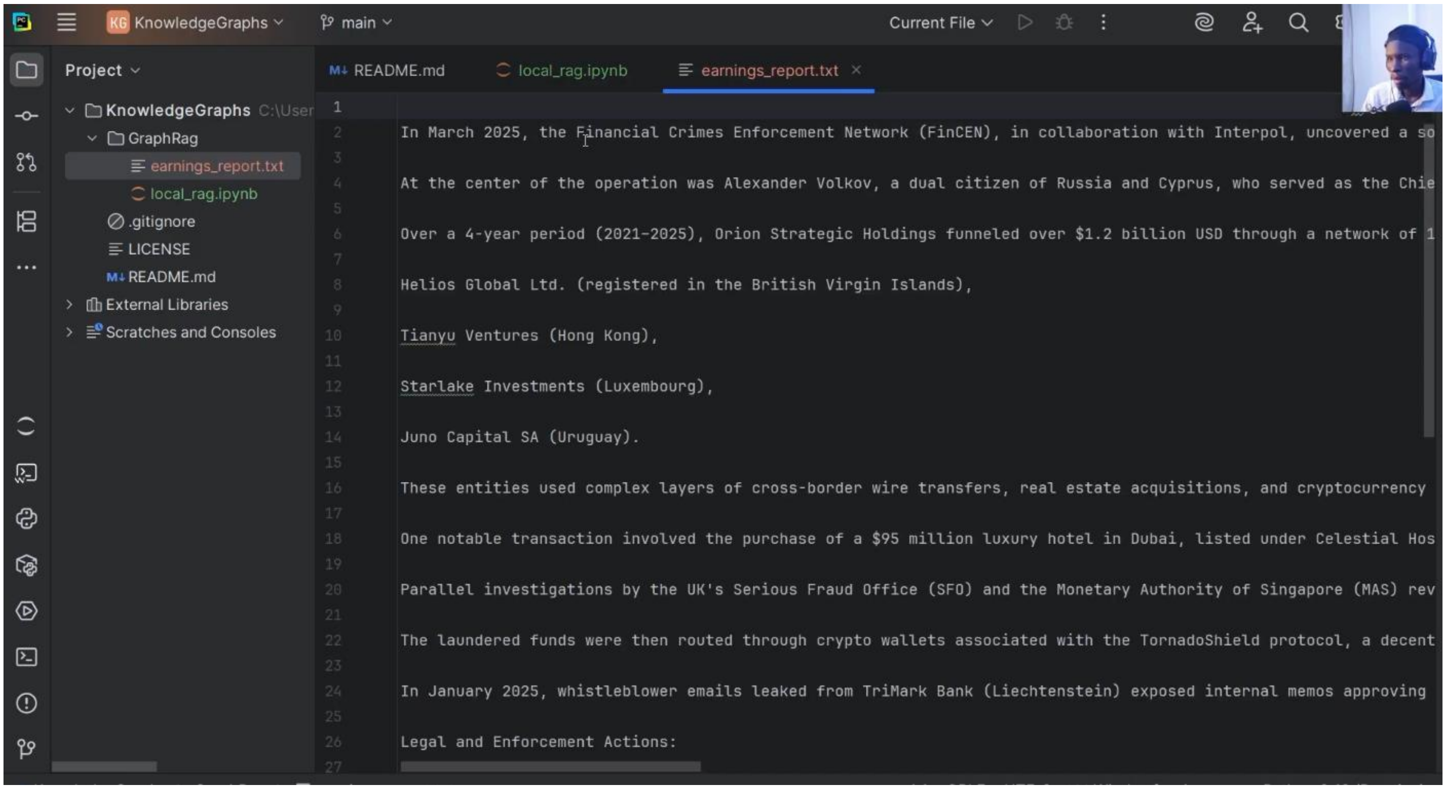
No changes, no records

Completed after 20 ms

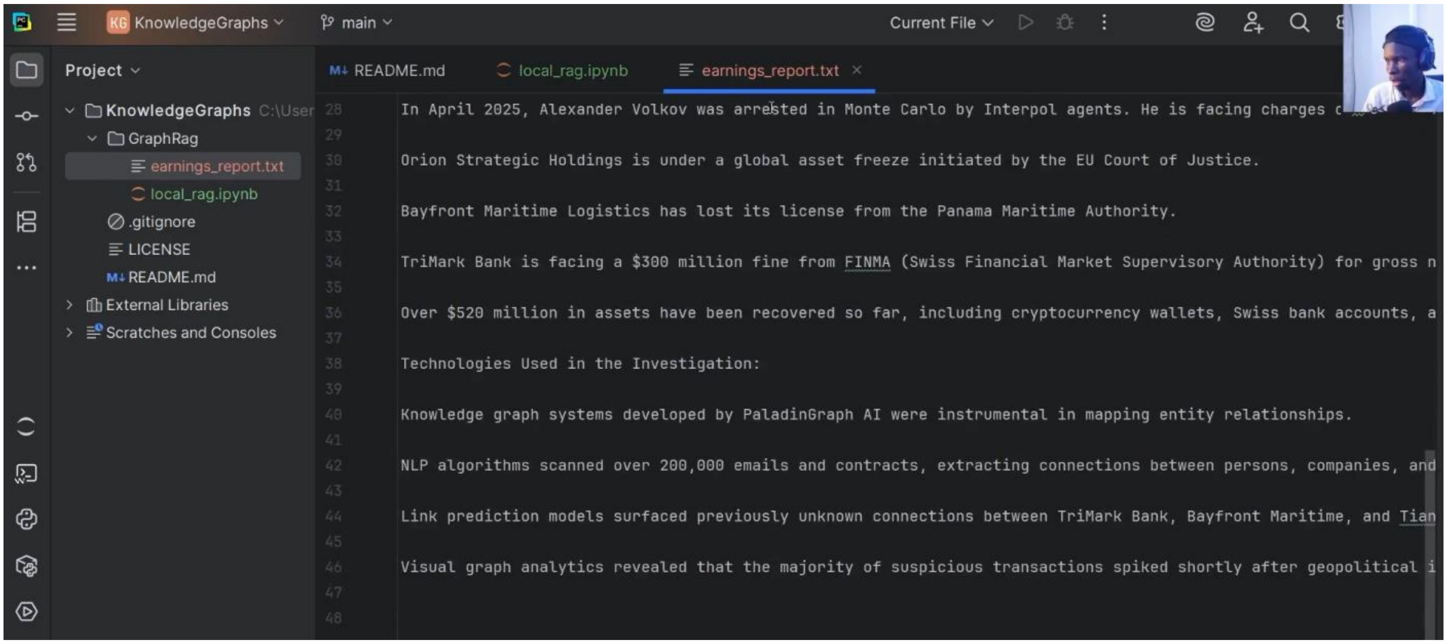
```
neo4j$ MATCH (n) RETURN n LIMIT 100
```

Started streaming 8 records after 10 ms and completed after 12 ms.

```
neo4j$ MATCH (n) WHERE n.name IS NOT NULL RETURN DISTINCT "node" as entity, n.name AS name LIMIT 25 UNION ALL MATCH ()-[r]-
```



```

In March 2025, the Financial Crimes Enforcement Network (FinCEN), in collaboration with Interpol, uncovered a so

At the center of the operation was Alexander Volkov, a dual citizen of Russia and Cyprus, who served as the Chie

Over a 4-year period (2021-2025), Orion Strategic Holdings funneled over $1.2 billion USD through a network of 1

Helios Global Ltd. (registered in the British Virgin Islands),

Tianyu Ventures (Hong Kong),

Starlake Investments (Luxembourg),

Juno Capital SA (Uruguay).

These entities used complex layers of cross-border wire transfers, real estate acquisitions, and cryptocurrency

One notable transaction involved the purchase of a $95 million luxury hotel in Dubai, listed under Celestial Hos

Parallel investigations by the UK's Serious Fraud Office (SFO) and the Monetary Authority of Singapore (MAS) rev

The laundered funds were then routed through crypto wallets associated with the TornadoShield protocol, a decent

In January 2025, whistleblower emails leaked from TriMark Bank (Liechtenstein) exposed internal memos approving

Legal and Enforcement Actions:
```



```
In April 2025, Alexander Volkov was arrested in Monte Carlo by Interpol agents. He is facing charges c

Orion Strategic Holdings is under a global asset freeze initiated by the EU Court of Justice.

Bayfront Maritime Logistics has lost its license from the Panama Maritime Authority.

TriMark Bank is facing a $300 million fine from FINMA (Swiss Financial Market Supervisory Authority) for gross n

Over $520 million in assets have been recovered so far, including cryptocurrency wallets, Swiss bank accounts, a

Technologies Used in the Investigation:

Knowledge graph systems developed by PaladinGraph AI were instrumental in mapping entity relationships.

NLP algorithms scanned over 200,000 emails and contracts, extracting connections between persons, companies, and

Link prediction models surfaced previously unknown connections between TriMark Bank, Bayfront Maritime, and Tian

Visual graph analytics revealed that the majority of suspicious transactions spiked shortly after geopolitical i

```