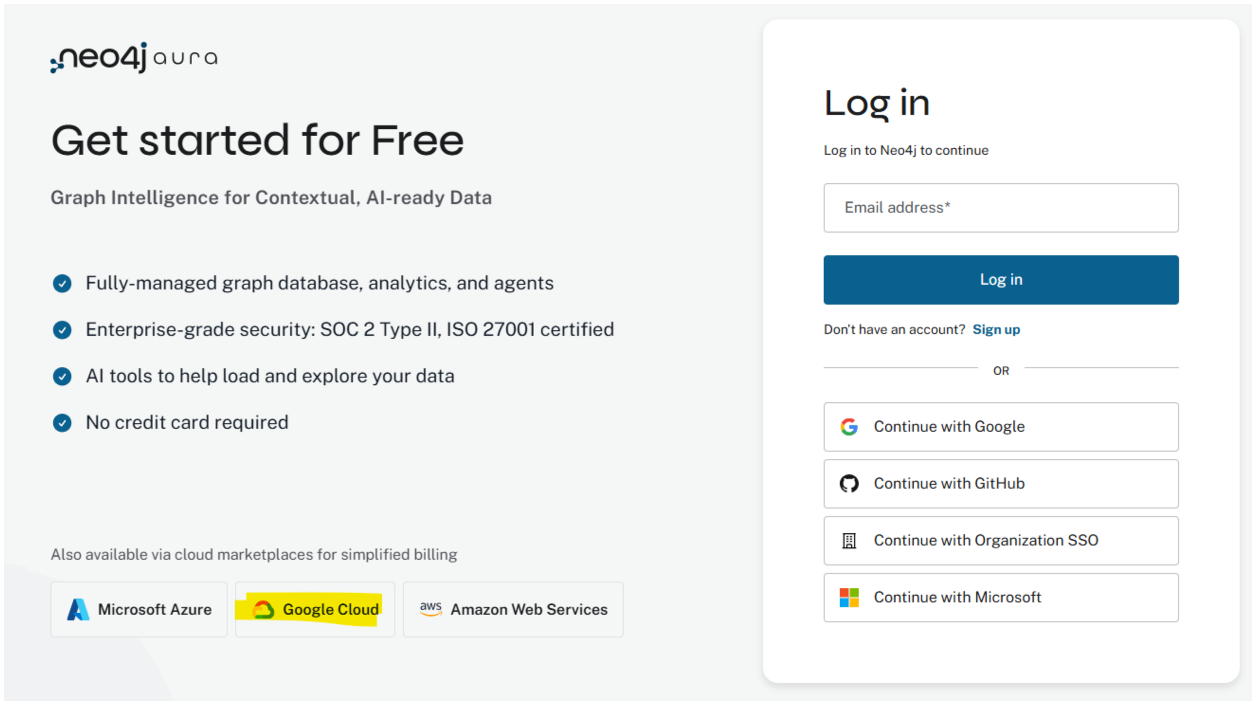
Snowflake Integration With Graph DB + Knowledge Graph + Snowflake Agent & Intelligence

Neo4j Graph DB With Snowflake



neo4j aura

Get started for Free

Graph Intelligence for Contextual, AI-ready Data

- Fully-managed graph database, analytics, and agents
- Enterprise-grade security: SOC 2 Type II, ISO 27001 certified
- AI tools to help load and explore your data
- No credit card required

Also available via cloud marketplaces for simplified billing

Microsoft Azure | Google Cloud | Amazon Web Services

Log in

Log in to Neo4j to continue

Email address*

Log in

Don't have an account? Sign up

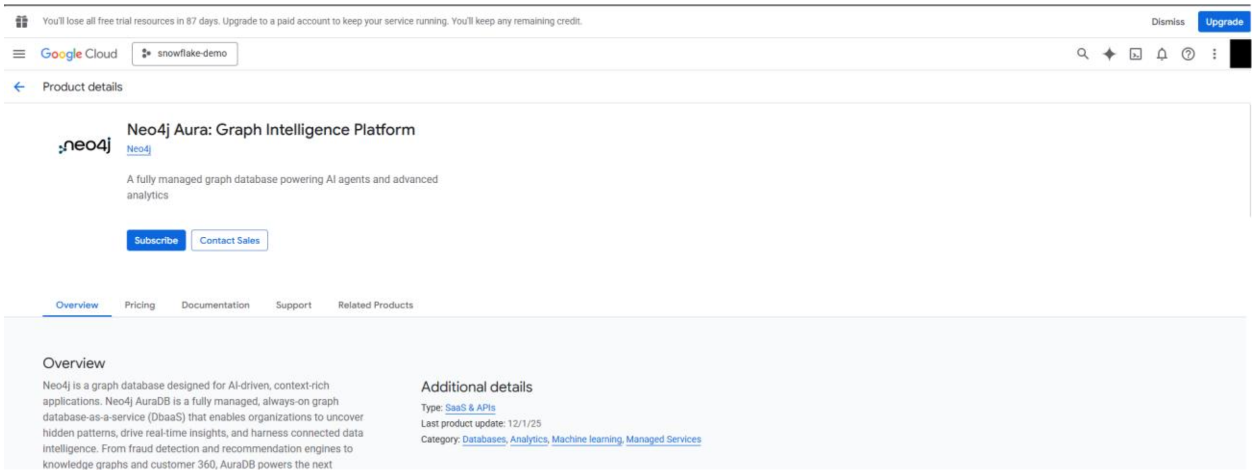
OR

Continue with Google

Continue with GitHub

Continue with Organization SSO

Continue with Microsoft



You'll lose all free trial resources in 87 days. Upgrade to a paid account to keep your service running. You'll keep any remaining credit.

Dismiss Upgrade

Google Cloud snowflake-demo

Product details

Neo4j Aura: Graph Intelligence Platform

Neo4j

A fully managed graph database powering AI agents and advanced analytics

Subscribe Contact Sales

Overview Pricing Documentation Support Related Products

Overview

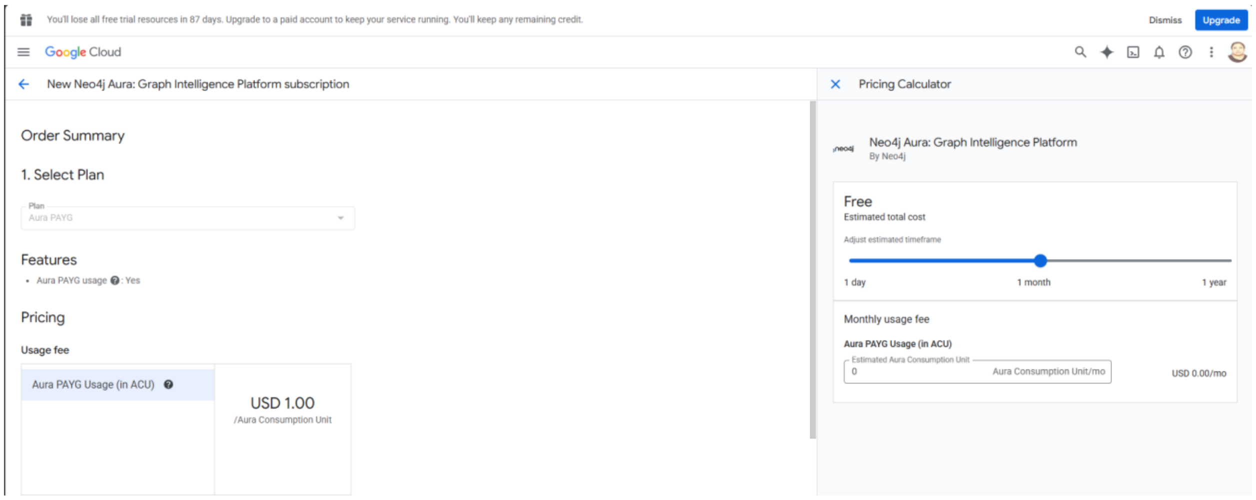
Neo4j is a graph database designed for AI-driven, context-rich applications. Neo4j AuraDB is a fully managed, always-on graph database-as-a-service (DbaaS) that enables organizations to uncover hidden patterns, drive real-time insights, and harness connected data intelligence. From fraud detection and recommendation engines to knowledge graphs and customer 360, AuraDB powers the next

Additional details

Type: SaaS & APIs

Last product update: 12/1/25

Category: Databases, Analytics, Machine learning, Managed Services



You'll lose all free trial resources in 87 days. Upgrade to a paid account to keep your service running. You'll keep any remaining credit.

New Neo4j Aura: Graph Intelligence Platform subscription

Order Summary

1. Select Plan

Plan: Aura PAYG

Features

- Aura PAYG usage: Yes

Pricing

Usage fee

Aura PAYG Usage (in ACU)	USD 1.00 /Aura Consumption Unit

Pricing Calculator

Neo4j Aura: Graph Intelligence Platform
By Neo4j

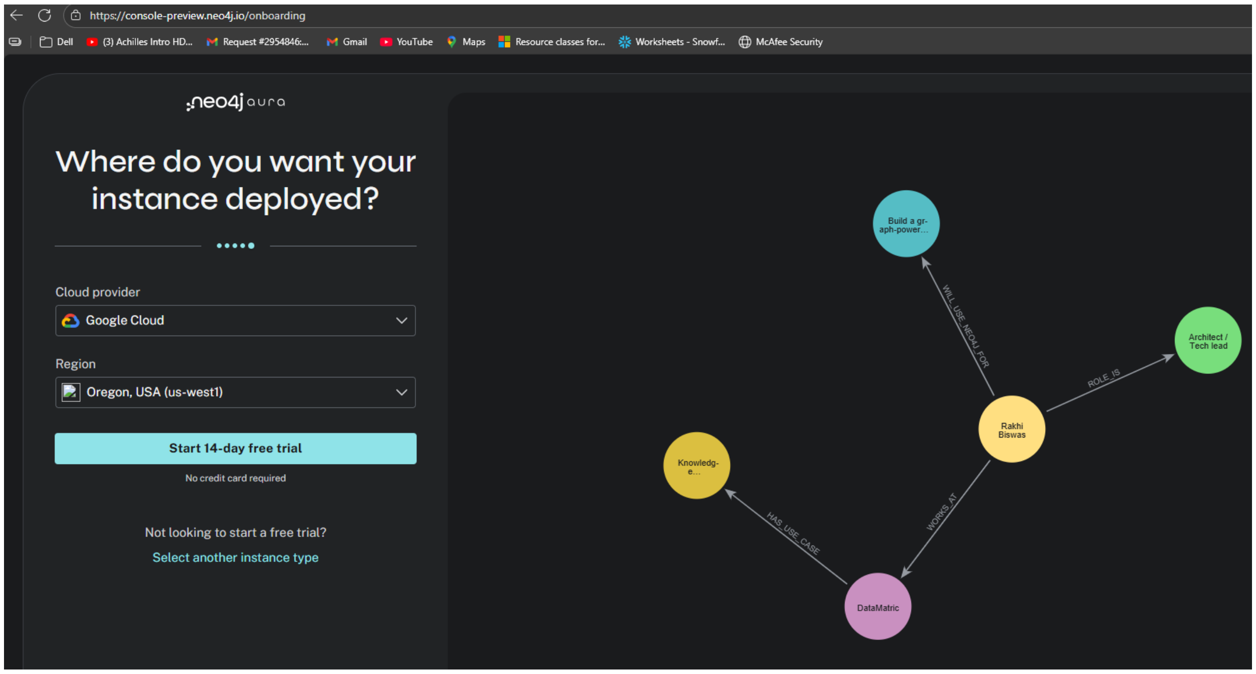
Free
Estimated total cost

Adjust estimated timeframe

Monthly usage fee

Aura PAYG Usage (in ACU)

Estimated Aura Consumption Unit: 0 Aura Consumption Unit/mo — USD 0.00/mo



https://console-preview.neo4j.io/onboarding

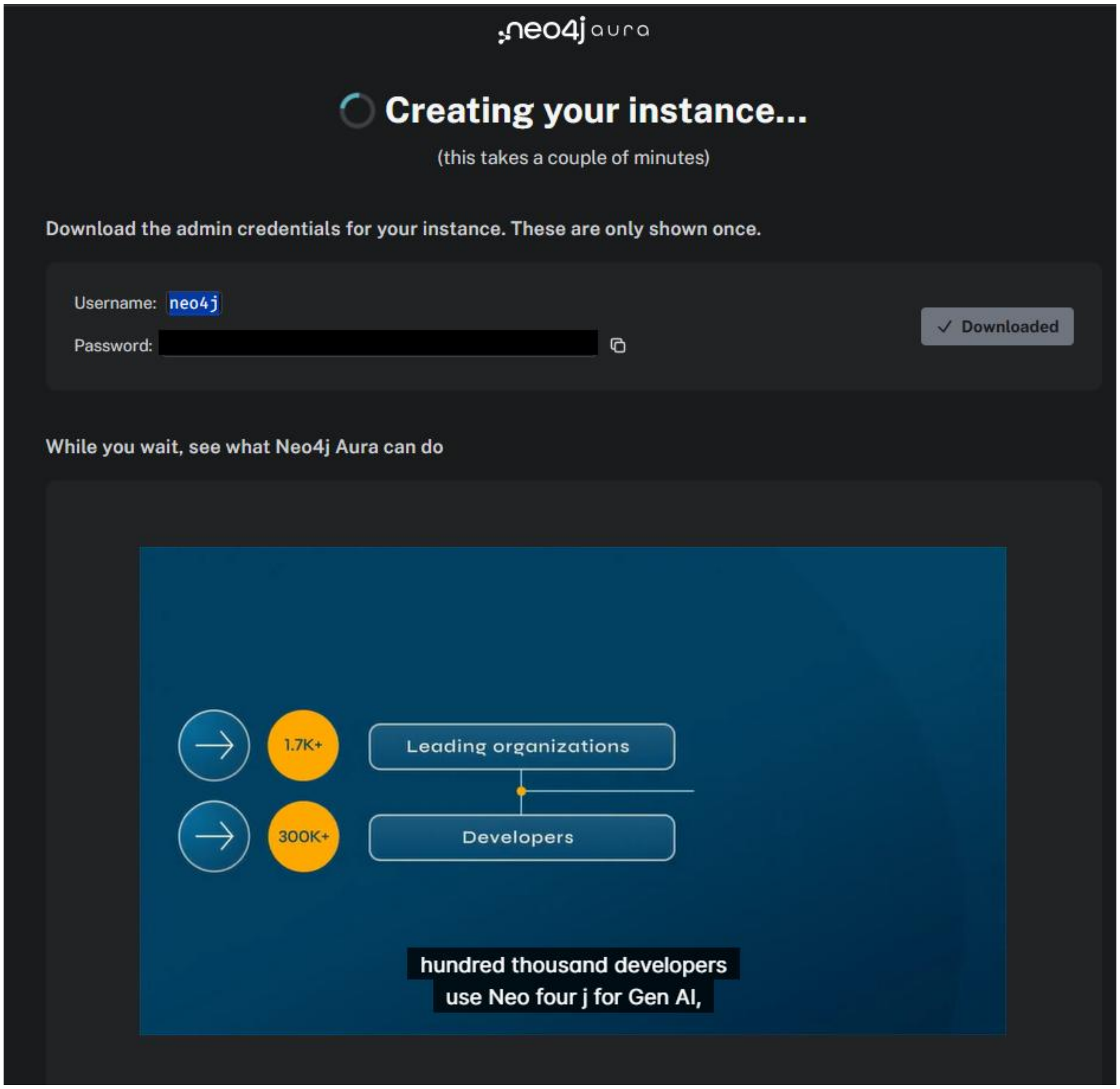
Where do you want your instance deployed?

Cloud provider: Google Cloud

Region: Oregon, USA (us-west1)

No credit card required

Not looking to start a free trial?



Creating your instance...

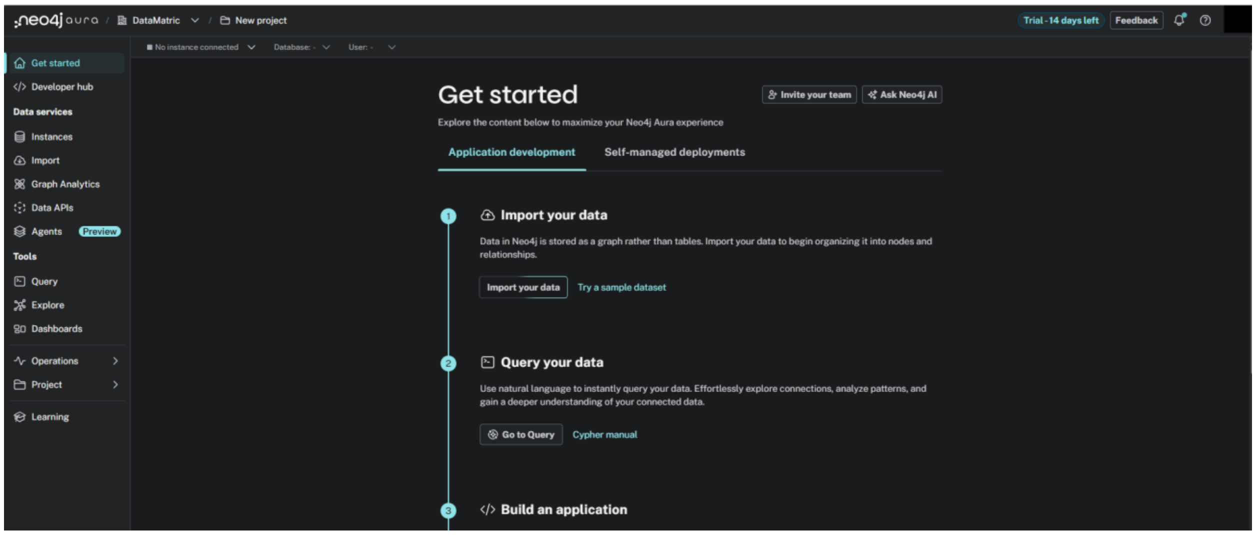
(this takes a couple of minutes)

Download the admin credentials for your instance. These are only shown once.

Username: neo4j

Password: [redacted]

While you wait, see what Neo4j Aura can do



Get started

Explore the content below to maximize your Neo4j Aura experience

1 Import your data

Data in Neo4j is stored as a graph rather than tables. Import your data to begin organizing it into nodes and relationships.

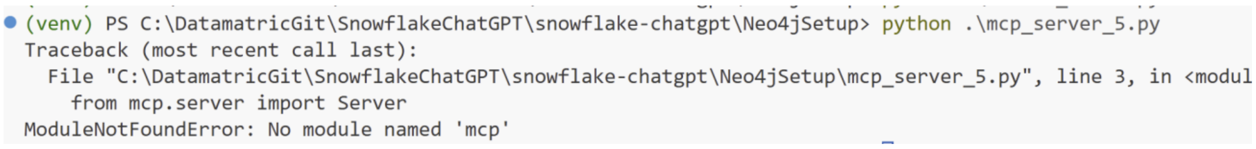
2 Query your data

Use natural language to instantly query your data. Effortlessly explore connections, analyze patterns, and gain a deeper understanding of your connected data.

3 Build an application

Error for MCP module

The MCP (Model Context Protocol) SDK **is NOT installed by default**, and it must be added manually.



```
(venv) PS C:\DatamatricGit\SnowflakeChatGPT\snowflake-chatgpt\Neo4jSetup> python .\mcp_server_5.py
Traceback (most recent call last):
  File "C:\DatamatricGit\SnowflakeChatGPT\snowflake-chatgpt\Neo4jSetup\mcp_server_5.py", line 3, in <modul
    from mcp.server import Server
ModuleNotFoundError: No module named 'mcp'
```

To install MCP server, client and types, we need to install model-context-protocol

pip install model-context-protocol

For MCP server, we need 3.10 or above version

We can have both versions installation in the same server

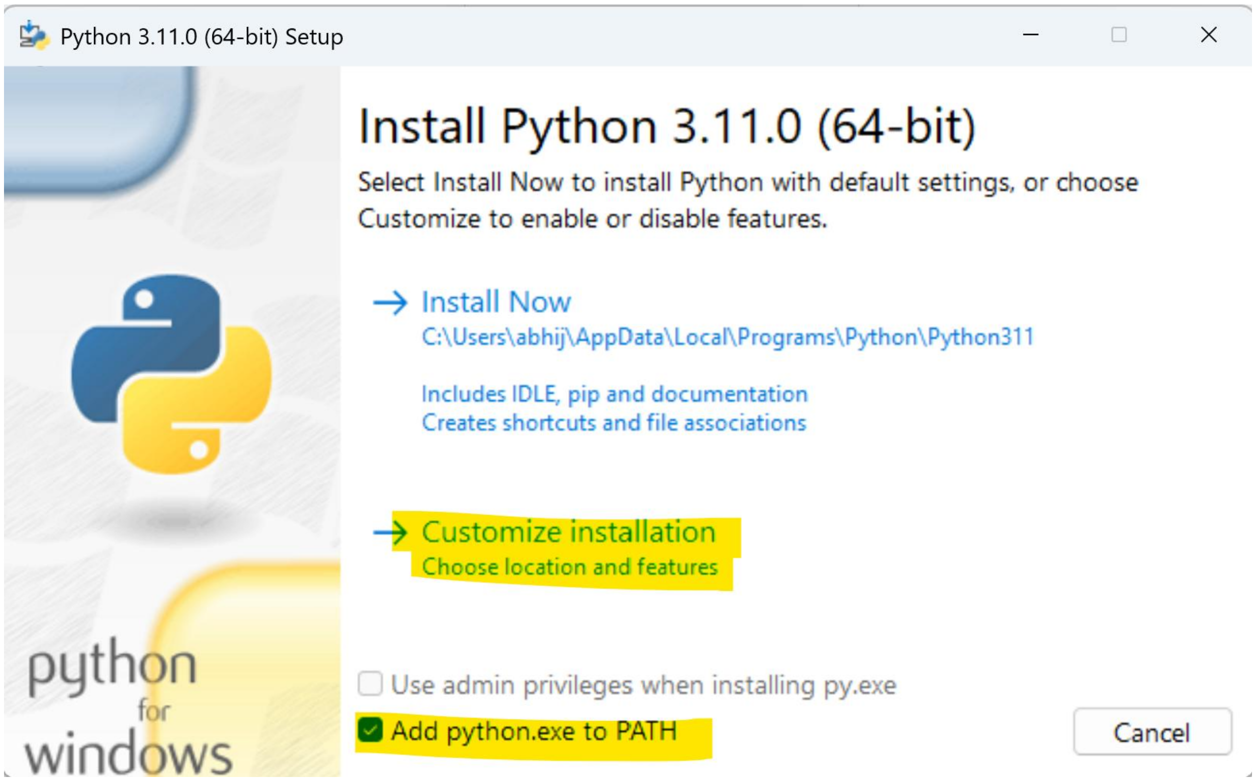
Official Python website:

https://www.python.org/downloads/release/python-3110/

Download the Windows installer (64-bit).

When installing:

✓ KEEP your existing 3.9 installation
✓ CHECK: “Add Python 3.11 to PATH” (optional but convenient)
✓ Choose Customize installation
✓ Click Install for all users (recommended)



Python 3.11.0 (64-bit) Setup

Install Python 3.11.0 (64-bit)

Select Install Now to install Python with default settings, or choose Customize to enable or disable features.

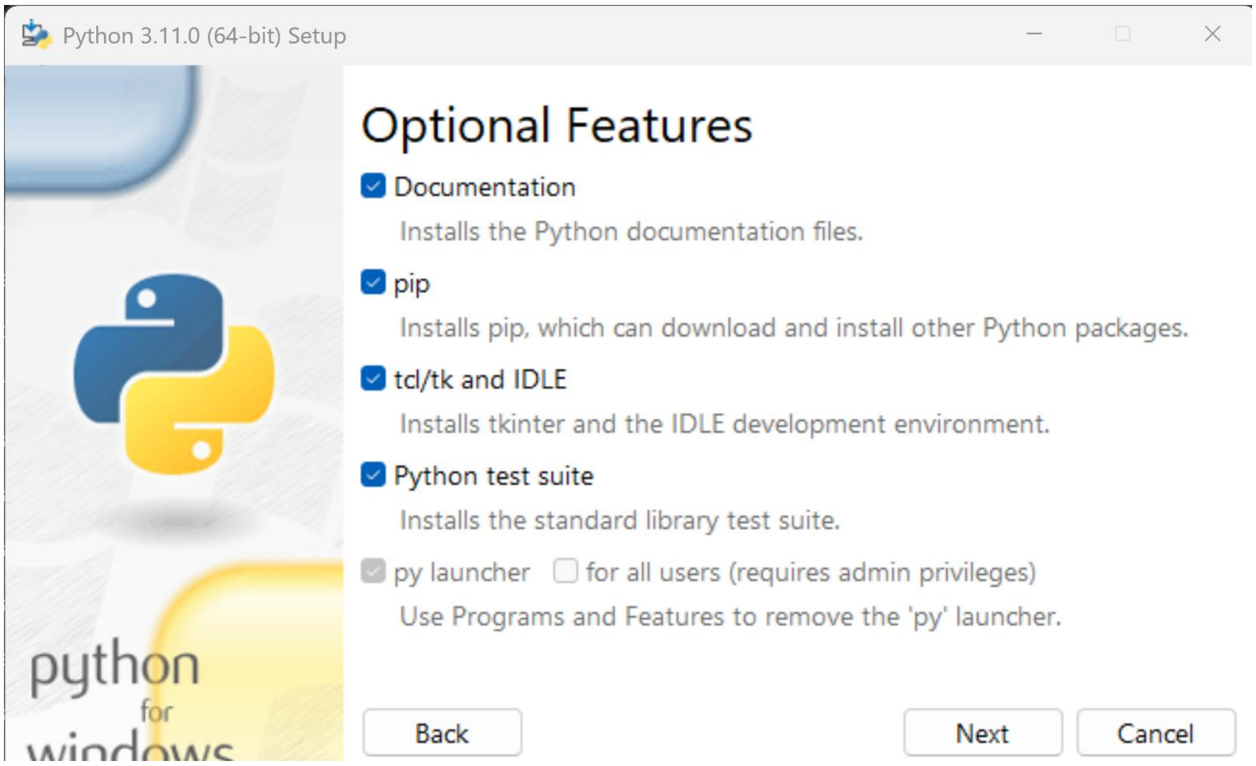
→ Install Now
C:\Users\abhij\AppData\Local\Programs\Python\Python311

Includes IDLE, pip and documentation
Creates shortcuts and file associations

→ Customize installation
Choose location and features

☐ Use admin privileges when installing py.exe
☑ Add python.exe to PATH

Cancel



Python 3.11.0 (64-bit) Setup

Optional Features

☑ Documentation
Installs the Python documentation files.

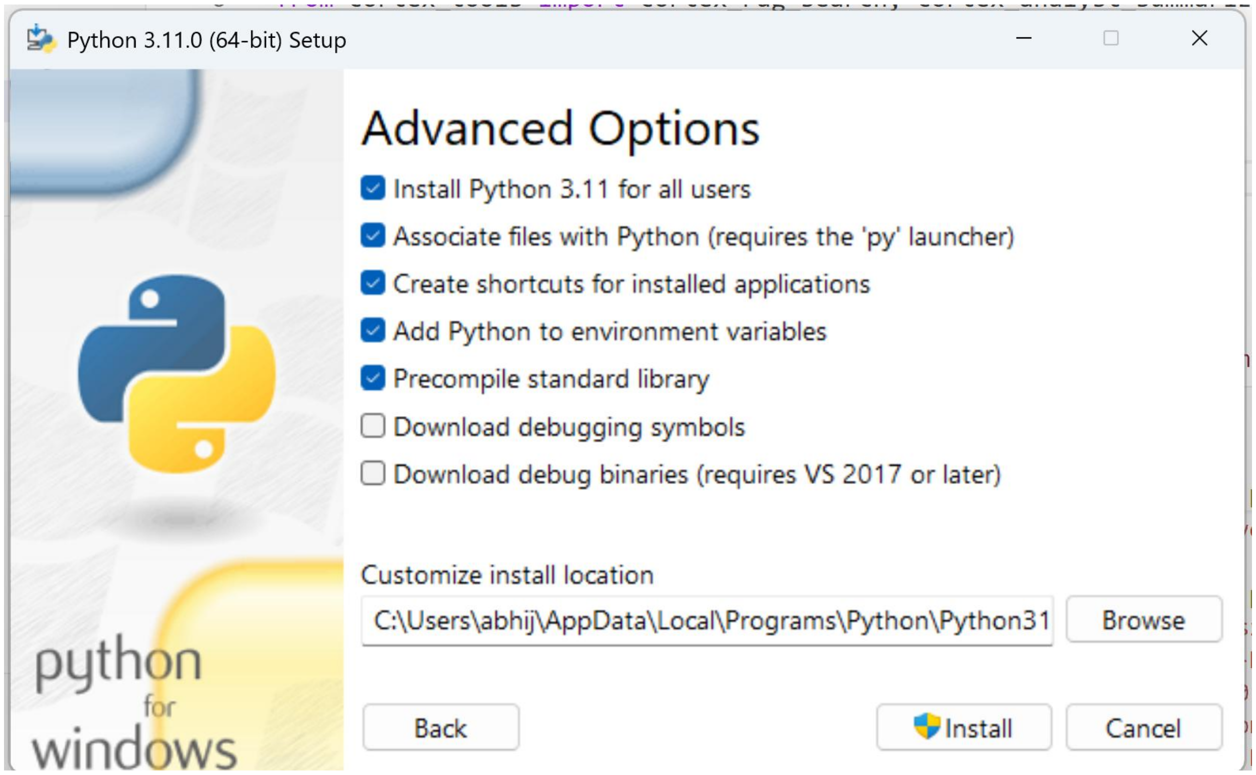
☑ pip
Installs pip, which can download and install other Python packages.

☑ tcl/tk and IDLE
Installs tkinter and the IDLE development environment.

☑ Python test suite
Installs the standard library test suite.

☑ py launcher ☐ for all users (requires admin privileges)
Use Programs and Features to remove the 'py' launcher.

Back Next Cancel


Python 3.11.0 (64-bit) Setup

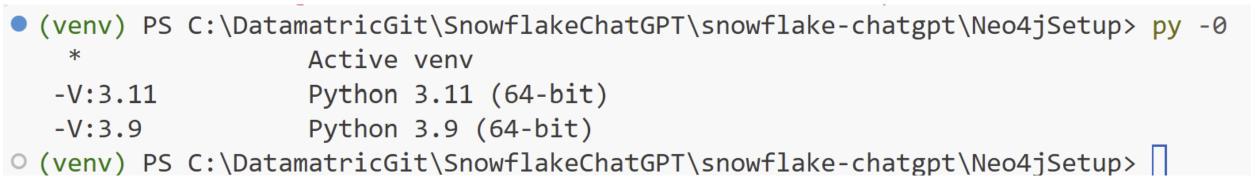
Advanced Options

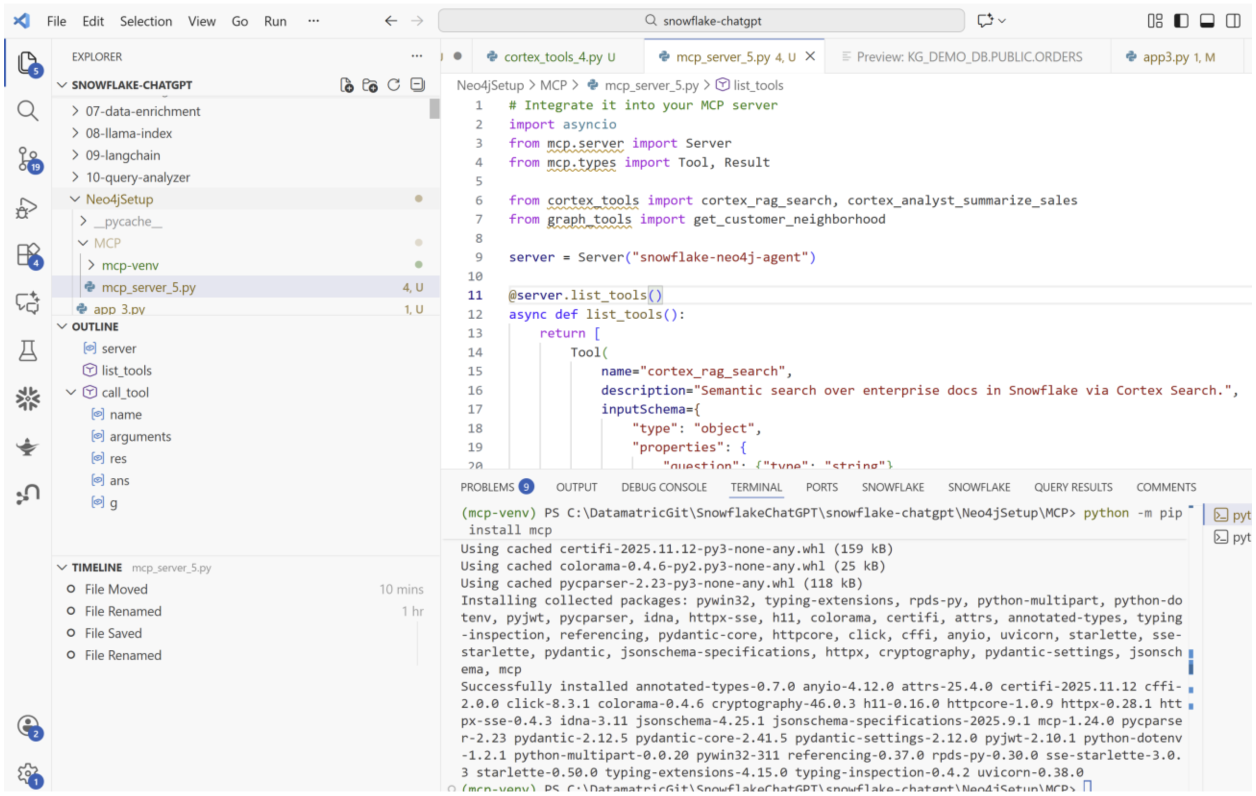
- [✓] Install Python 3.11 for all users
- [✓] Associate files with Python (requires the 'py' launcher)
- [✓] Create shortcuts for installed applications
- [✓] Add Python to environment variables
- [✓] Precompile standard library
- [ ] Download debugging symbols
- [ ] Download debug binaries (requires VS 2017 or later)

Customize install location

C:\Users\abhij\AppData\Local\Programs\Python\Python31

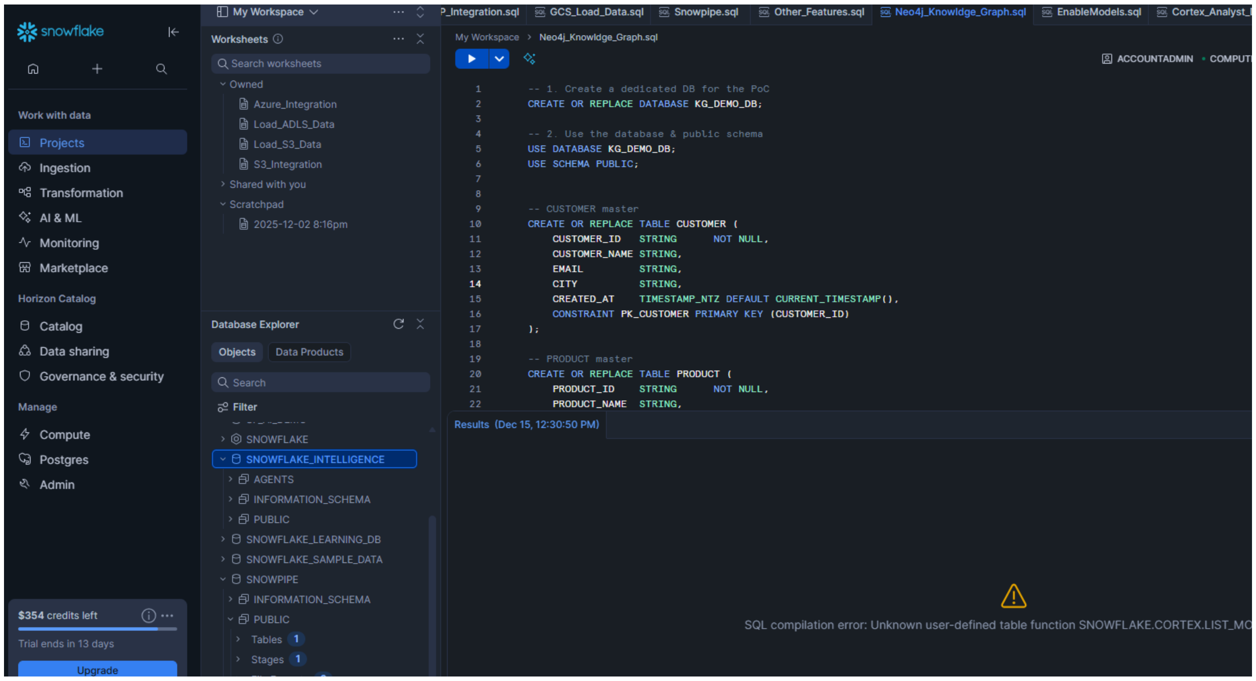
I already have python 3.11 version


```
(venv) PS C:\DatamatricGit\SnowflakeChatGPT\snowflake-chatgpt\Neo4jSetup> py -0
  *               Active venv
 -V:3.11          Python 3.11 (64-bit)
 -V:3.9           Python 3.9 (64-bit)
(venv) PS C:\DatamatricGit\SnowflakeChatGPT\snowflake-chatgpt\Neo4jSetup>
```



```python
# Integrate it into your MCP server
import asyncio
from mcp.server import Server
from mcp.types import Tool, Result

from cortex_tools import cortex_rag_search, cortex_analyst_summarize_sales
from graph_tools import get_customer_neighborhood

server = Server("snowflake-neo4j-agent")

@server.list_tools()
async def list_tools():
    return [
        Tool(
            name="cortex_rag_search",
            description="Semantic search over enterprise docs in Snowflake via Cortex Search.",
            inputSchema={
                "type": "object",
                "properties": {
```

```
(mcp-venv) PS C:\DatamatricGit\SnowflakeChatGPT\snowflake-chatgpt\Neo4jSetup\MCP> python -m pip
 install mcp
Using cached certifi-2025.11.12-py3-none-any.whl (159 kB)
Using cached colorama-0.4.6-py2.py3-none-any.whl (25 kB)
Using cached pycparser-2.23-py3-none-any.whl (118 kB)
Installing collected packages: pywin32, typing-extensions, rpds-py, python-multipart, python-do
tenv, pyjwt, pycparser, idna, httpx-sse, h11, colorama, certifi, attrs, annotated-types, typing
-inspection, referencing, pydantic-core, httpcore, click, cffi, anyio, uvicorn, starlette, sse-
starlette, pydantic, jsonschema-specifications, httpx, cryptography, pydantic-settings, jsonsch
ema, mcp
Successfully installed annotated-types-0.7.0 anyio-4.12.0 attrs-25.4.0 certifi-2025.11.12 cffi-
2.0.0 click-8.3.1 colorama-0.4.6 cryptography-46.0.3 h11-0.16.0 httpcore-1.0.9 httpx-0.28.1 htt
px-sse-0.4.3 idna-3.11 jsonschema-4.25.1 jsonschema-specifications-2025.9.1 mcp-1.24.0 pycparse
r-2.23 pydantic-2.12.5 pydantic-core-2.41.5 pydantic-settings-2.12.0 pyjwt-2.10.1 python-dotenv
-1.2.1 python-multipart-0.0.20 pywin32-311 referencing-0.37.0 rpds-py-0.30.0 sse-starlette-3.0.
3 starlette-0.50.0 typing-extensions-4.15.0 typing-inspection-0.4.2 uvicorn-0.38.0
```



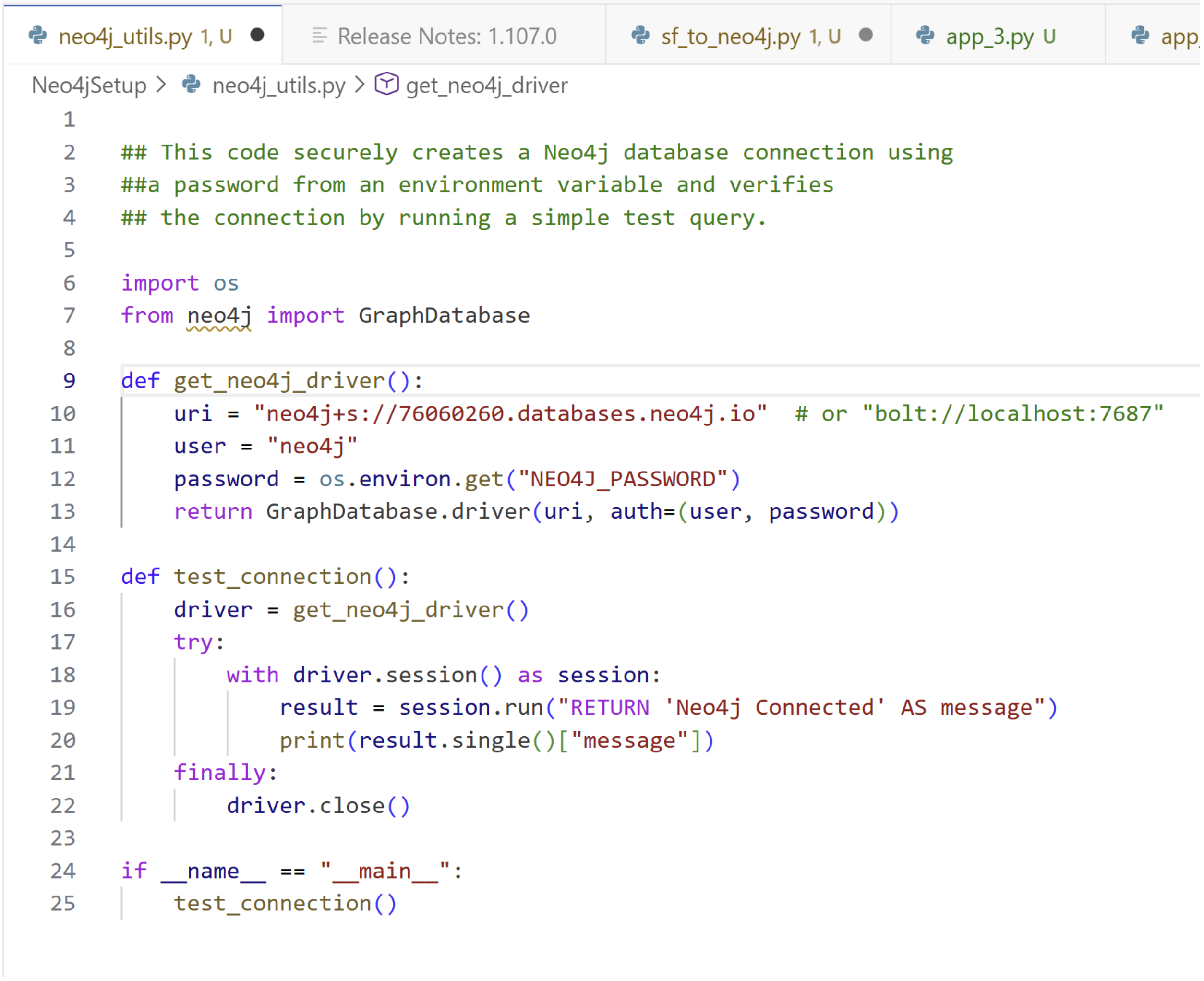
```sql
-- 1. Create a dedicated DB for the PoC
CREATE OR REPLACE DATABASE KG_DEMO_DB;

-- 2. Use the database & public schema
USE DATABASE KG_DEMO_DB;
USE SCHEMA PUBLIC;


-- CUSTOMER master
CREATE OR REPLACE TABLE CUSTOMER (
    CUSTOMER_ID   STRING      NOT NULL,
    CUSTOMER_NAME STRING,
    EMAIL         STRING,
    CITY          STRING,
    CREATED_AT    TIMESTAMP_NTZ DEFAULT CURRENT_TIMESTAMP(),
    CONSTRAINT PK_CUSTOMER PRIMARY KEY (CUSTOMER_ID)
);

-- PRODUCT master
CREATE OR REPLACE TABLE PRODUCT (
    PRODUCT_ID    STRING      NOT NULL,
    PRODUCT_NAME  STRING,
```

Results (Dec 15, 12:30:50 PM)

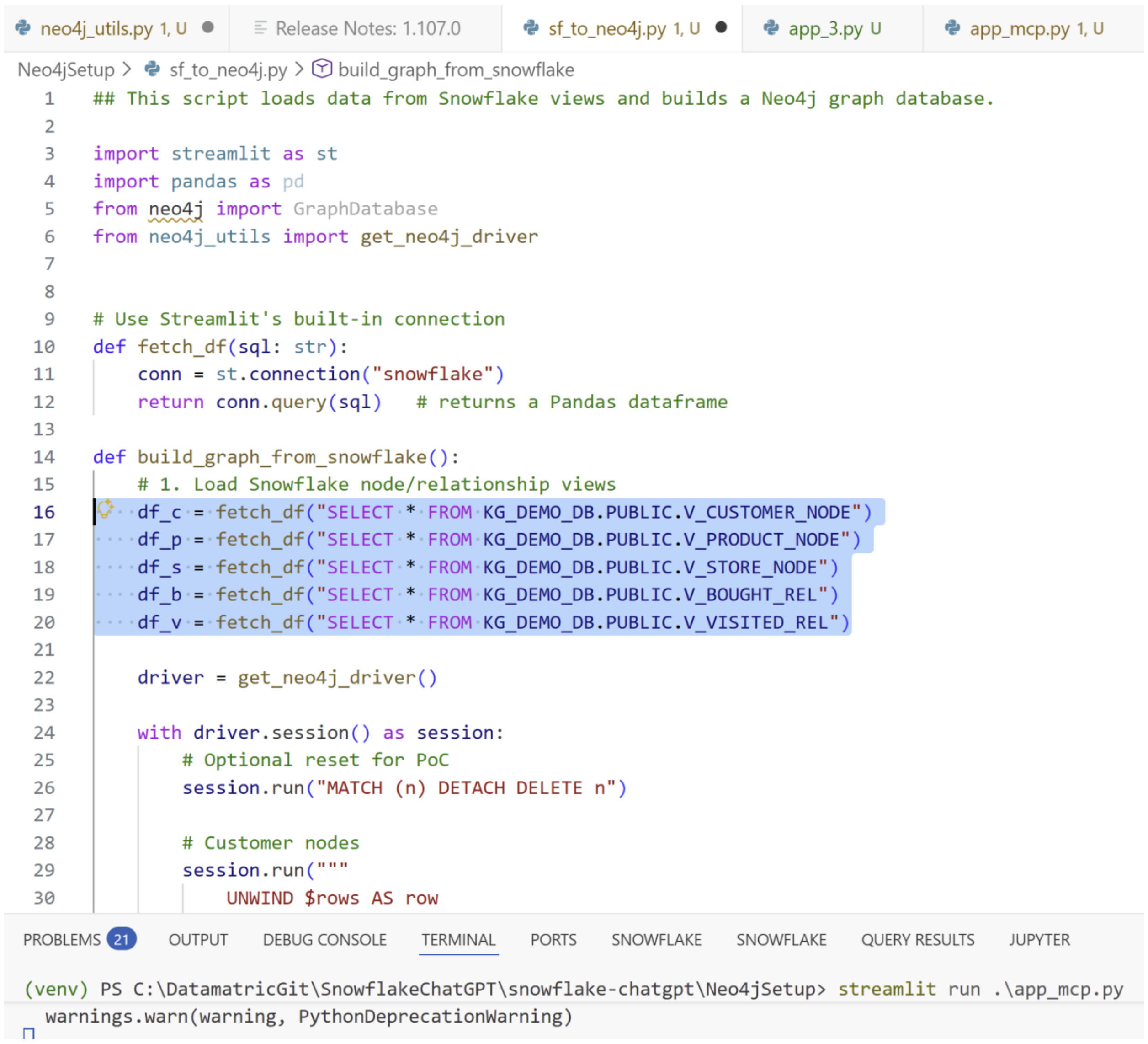
SQL compilation error: Unknown user-defined table function SNOWFLAKE.CORTEX.LIST_MC

Share codes in VS Code

This code securely creates a Neo4j database connection using a password from an environment variable and verifies the connection by running a simple test query.



```python

## This code securely creates a Neo4j database connection using
##a password from an environment variable and verifies
## the connection by running a simple test query.

import os
from neo4j import GraphDatabase

def get_neo4j_driver():
    uri = "neo4j+s://76060260.databases.neo4j.io"  # or "bolt://localhost:7687"
    user = "neo4j"
    password = os.environ.get("NEO4J_PASSWORD")
    return GraphDatabase.driver(uri, auth=(user, password))

def test_connection():
    driver = get_neo4j_driver()
    try:
        with driver.session() as session:
            result = session.run("RETURN 'Neo4j Connected' AS message")
            print(result.single()["message"])
    finally:
        driver.close()

if __name__ == "__main__":
    test_connection()
```



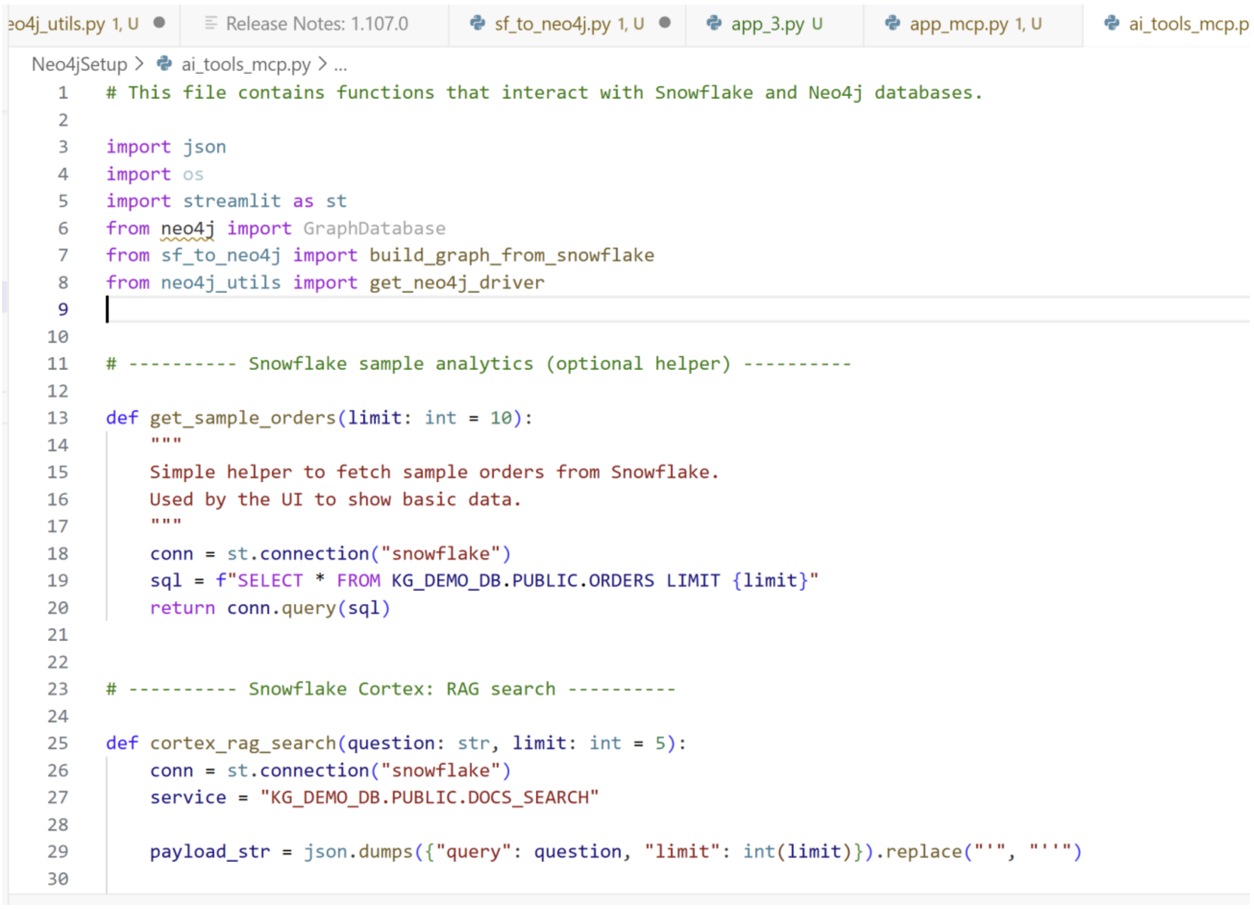
neo4j_utils.py 1, U ● | Release Notes: 1.107.0 | sf_to_neo4j.py 1, U ● | app_3.py U | app_mcp.py 1, U

Neo4jSetup > sf_to_neo4j.py > build_graph_from_snowflake

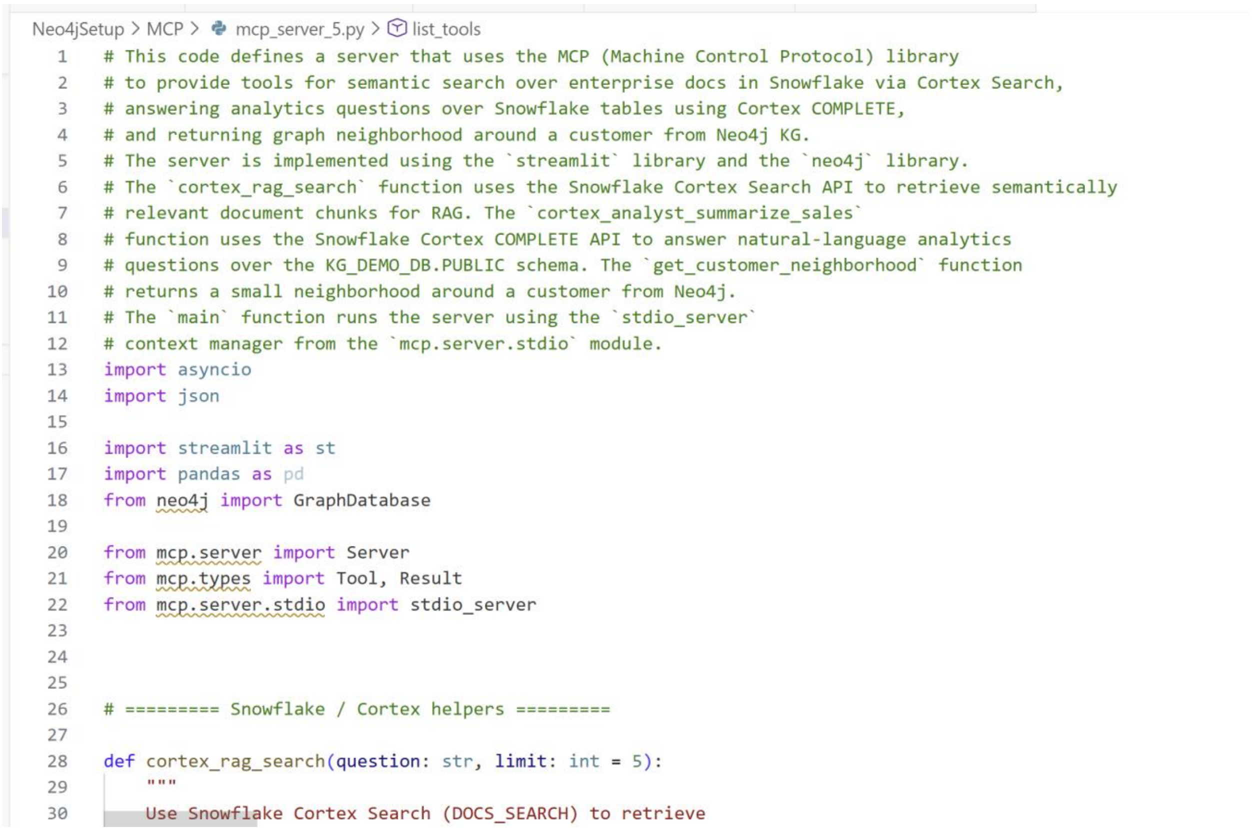
```python
## This script loads data from Snowflake views and builds a Neo4j graph database.

import streamlit as st
import pandas as pd
from neo4j import GraphDatabase
from neo4j_utils import get_neo4j_driver


# Use Streamlit's built-in connection
def fetch_df(sql: str):
    conn = st.connection("snowflake")
    return conn.query(sql)   # returns a Pandas dataframe

def build_graph_from_snowflake():
    # 1. Load Snowflake node/relationship views
    df_c = fetch_df("SELECT * FROM KG_DEMO_DB.PUBLIC.V_CUSTOMER_NODE")
    df_p = fetch_df("SELECT * FROM KG_DEMO_DB.PUBLIC.V_PRODUCT_NODE")
    df_s = fetch_df("SELECT * FROM KG_DEMO_DB.PUBLIC.V_STORE_NODE")
    df_b = fetch_df("SELECT * FROM KG_DEMO_DB.PUBLIC.V_BOUGHT_REL")
    df_v = fetch_df("SELECT * FROM KG_DEMO_DB.PUBLIC.V_VISITED_REL")

    driver = get_neo4j_driver()

    with driver.session() as session:
        # Optional reset for PoC
        session.run("MATCH (n) DETACH DELETE n")

        # Customer nodes
        session.run("""
            UNWIND $rows AS row
```

PROBLEMS 21 | OUTPUT | DEBUG CONSOLE | TERMINAL | PORTS | SNOWFLAKE | SNOWFLAKE | QUERY RESULTS | JUPYTER

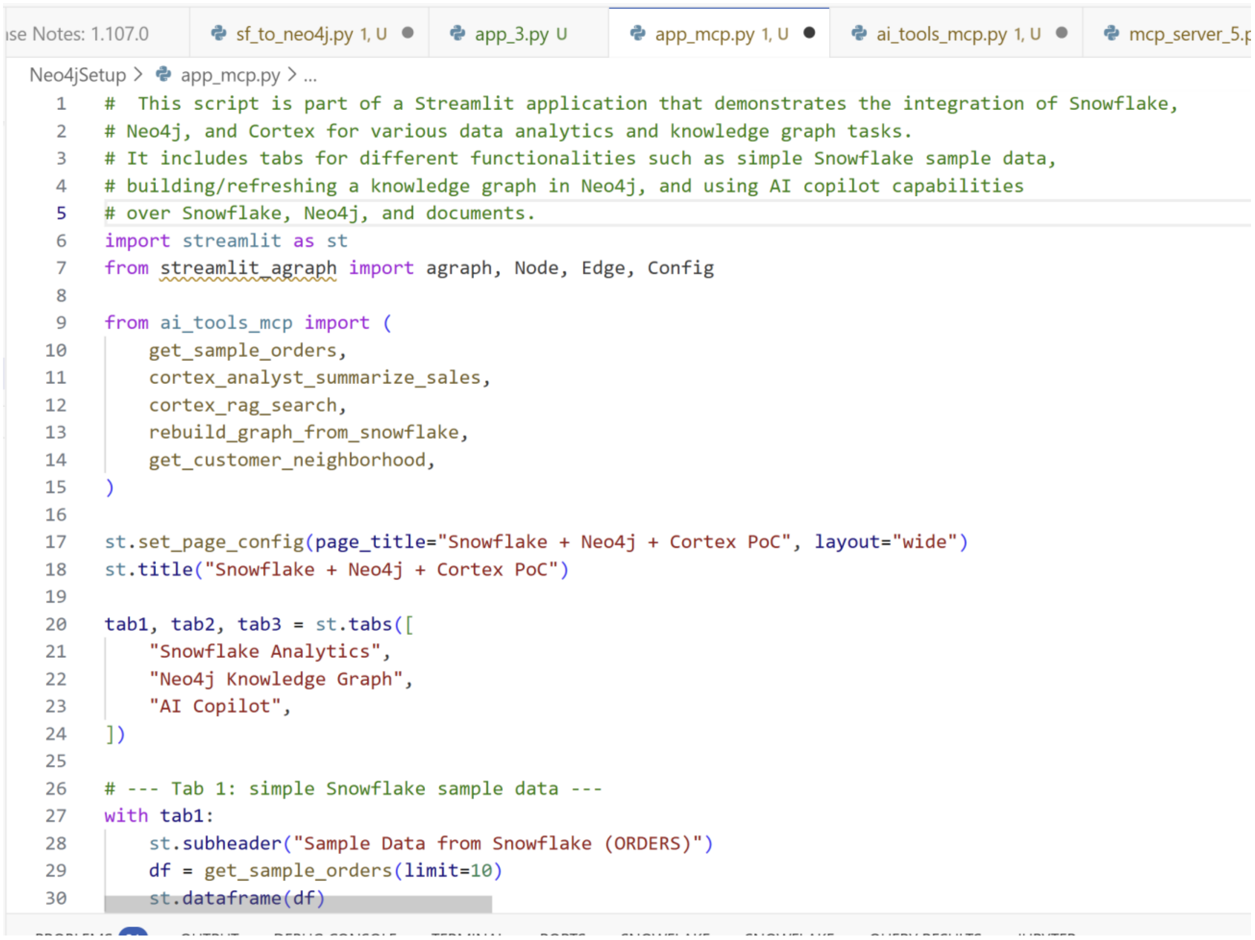
```
(venv) PS C:\DatamatricGit\SnowflakeChatGPT\snowflake-chatgpt\Neo4jSetup> streamlit run .\app_mcp.py
  warnings.warn(warning, PythonDeprecationWarning)
```



```python
# This file contains functions that interact with Snowflake and Neo4j databases.

import json
import os
import streamlit as st
from neo4j import GraphDatabase
from sf_to_neo4j import build_graph_from_snowflake
from neo4j_utils import get_neo4j_driver


# ---------- Snowflake sample analytics (optional helper) ----------

def get_sample_orders(limit: int = 10):
    """
    Simple helper to fetch sample orders from Snowflake.
    Used by the UI to show basic data.
    """
    conn = st.connection("snowflake")
    sql = f"SELECT * FROM KG_DEMO_DB.PUBLIC.ORDERS LIMIT {limit}"
    return conn.query(sql)


# ---------- Snowflake Cortex: RAG search ----------

def cortex_rag_search(question: str, limit: int = 5):
    conn = st.connection("snowflake")
    service = "KG_DEMO_DB.PUBLIC.DOCS_SEARCH"

    payload_str = json.dumps({"query": question, "limit": int(limit)}).replace("'", "''")

```


Neo4jSetup > MCP > mcp_server_5.py > list_tools

```
# This code defines a server that uses the MCP (Machine Control Protocol) library
# to provide tools for semantic search over enterprise docs in Snowflake via Cortex Search,
# answering analytics questions over Snowflake tables using Cortex COMPLETE,
# and returning graph neighborhood around a customer from Neo4j KG.
# The server is implemented using the `streamlit` library and the `neo4j` library.
# The `cortex_rag_search` function uses the Snowflake Cortex Search API to retrieve semantically
# relevant document chunks for RAG. The `cortex_analyst_summarize_sales`
# function uses the Snowflake Cortex COMPLETE API to answer natural-language analytics
# questions over the KG_DEMO_DB.PUBLIC schema. The `get_customer_neighborhood` function
# returns a small neighborhood around a customer from Neo4j.
# The `main` function runs the server using the `stdio_server`
# context manager from the `mcp.server.stdio` module.
import asyncio
import json

import streamlit as st
import pandas as pd
from neo4j import GraphDatabase

from mcp.server import Server
from mcp.types import Tool, Result
from mcp.server.stdio import stdio_server



# ========= Snowflake / Cortex helpers =========

def cortex_rag_search(question: str, limit: int = 5):
    """
    Use Snowflake Cortex Search (DOCS_SEARCH) to retrieve
```

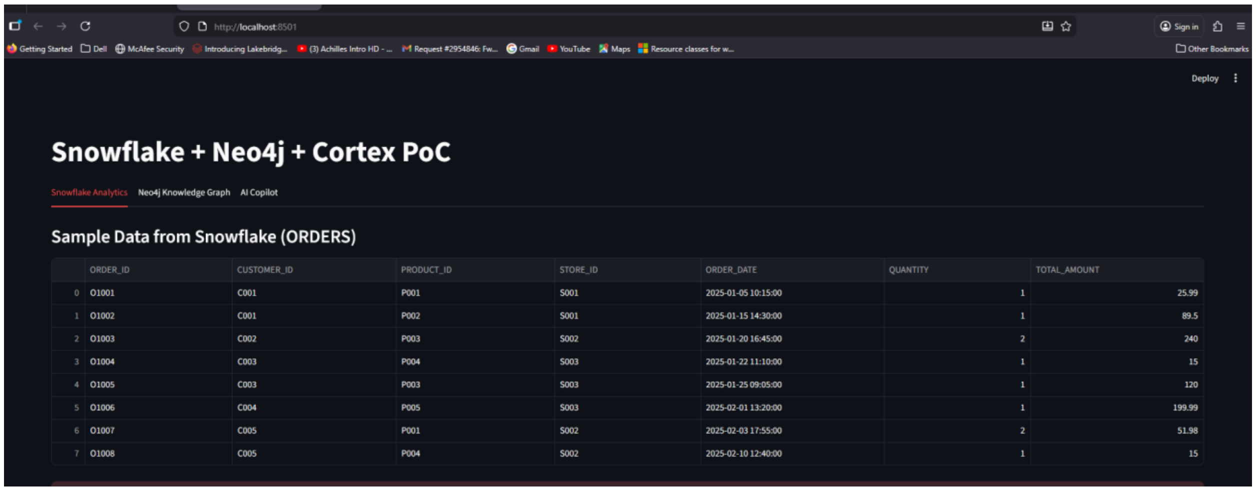


ise Notes: 1.107.0 | sf_to_neo4j.py 1, U | app_3.py U | app_mcp.py 1, U | ai_tools_mcp.py 1, U | mcp_server_5.p

Neo4jSetup > app_mcp.py > ...

```python
#  This script is part of a Streamlit application that demonstrates the integration of Snowflake,
# Neo4j, and Cortex for various data analytics and knowledge graph tasks.
# It includes tabs for different functionalities such as simple Snowflake sample data,
# building/refreshing a knowledge graph in Neo4j, and using AI copilot capabilities
# over Snowflake, Neo4j, and documents.
import streamlit as st
from streamlit_agraph import agraph, Node, Edge, Config

from ai_tools_mcp import (
    get_sample_orders,
    cortex_analyst_summarize_sales,
    cortex_rag_search,
    rebuild_graph_from_snowflake,
    get_customer_neighborhood,
)

st.set_page_config(page_title="Snowflake + Neo4j + Cortex PoC", layout="wide")
st.title("Snowflake + Neo4j + Cortex PoC")

tab1, tab2, tab3 = st.tabs([
    "Snowflake Analytics",
    "Neo4j Knowledge Graph",
    "AI Copilot",
])

# --- Tab 1: simple Snowflake sample data ---
with tab1:
    st.subheader("Sample Data from Snowflake (ORDERS)")
    df = get_sample_orders(limit=10)
    st.dataframe(df)
```

Run Streamlit



http://localhost:8501

Deploy

Snowflake + Neo4j + Cortex PoC

Sample Data from Snowflake (ORDERS)

	ORDER_ID	CUSTOMER_ID	PRODUCT_ID	STORE_ID	ORDER_DATE	QUANTITY	TOTAL_AMOUNT
0	O1001	C001	P001	S001	2025-01-05 10:15:00	1	25.99
1	O1002	C001	P002	S001	2025-01-15 14:30:00	1	89.5
2	O1003	C002	P003	S002	2025-01-20 16:45:00	2	240
3	O1004	C003	P004	S003	2025-01-22 11:10:00	1	15
4	O1005	C003	P003	S003	2025-01-25 09:05:00	1	120
5	O1006	C004	P005	S003	2025-02-01 13:20:00	1	199.99
6	O1007	C005	P001	S002	2025-02-03 17:55:00	2	51.98
7	O1008	C005	P004	S002	2025-02-10 12:40:00	1	15