# PharmApp Suite

## 🧠 AI for Drug Discovery and Development 🧪

Copyright 2025 | RnD Pharma Plus | www.nghiencuuthuoc.com | Zalo: +84888999311 |

## 🧩 Topic: Drugs with most activity data

Okay, I understand. You’re working on a drug discovery project using ChEMBL 35 data, RDKit, and a PostgreSQL database. You’re following an AIMLops template, using Jupyter notebooks for analysis, and need guidance on code execution and error handling. Let’s break this down, addressing your points in both English and Vietnamese.

**I. Analysis & Model Understanding**

Before diving into code, let’s define what we’re aiming for. I’m assuming “Topic\_CheMBL\_35\_42” relates to a specific research question, but without knowing the precise topic, I will generate five examples based on common drug discovery tasks using ChEMBL data. These examples will use SQL to query the database, Python with RDKit for data processing and analysis, and will address the potential errors you’ve identified.

**General Analysis Model:**

The general approach will be to:

1. **Query the ChEMBL database:** Use SQL to extract relevant data (e.g., bioactivity data, compound structures). This involves specifying your selection criteria (e.g., target, activity type, potency).
2. **Data Cleaning & Preprocessing:** Clean the data. This can include removing invalid entries, converting units, and dealing with missing values. Address the “ERROR: operator does not exist: numeric ~ unknown” by correctly casting data types.
3. **Feature Engineering (RDKit):** Use RDKit to calculate molecular descriptors (e.g., molecular weight, LogP, number of hydrogen bond donors/acceptors) from the SMILES strings of the compounds. These descriptors will serve as features for any subsequent modeling.
4. **Analysis & Visualization:** Perform analysis based on the specific topic. This could include calculating descriptive statistics, generating plots (e.g., scatter plots of activity vs. descriptor values), or building predictive models.
5. **Error Handling:** Ensure the code handles potential issues, such as missing data, incorrect data types, or version incompatibilities (e.g., the squared=False issue).

**II. Code Examples (SQL & Python)**

Here are five examples with SQL queries and corresponding Python code. Each example includes a brief explanation of the aim, the SQL query, and the Python code to analyze the results. The SQL queries are designed to return a maximum of 100 rows to limit the load on your machine.

**Example 1: Basic Activity Data Retrieval and Molecular Weight Calculation**

* **Aim:** Retrieve activity data for a specific target (e.g., a specific protein), calculate the molecular weight of the compounds, and plot a simple scatter plot of activity vs. molecular weight.
* **SQL Code (Save as ../data/example1.csv after running in pgAdmin):**

-- Example 1: Activity data and molecular weight  
  
SELECT  
 act.molregno,  
 act.standard\_value,  
 act.standard\_units,  
 md.canonical\_smiles  
FROM  
 activities act  
JOIN  
 target\_dictionary td ON act.tid = td.tid  
JOIN  
 molecule\_dictionary md ON act.molregno = md.molregno  
WHERE  
 td.pref\_name = 'CHEMBL182' -- Example target, replace with your target name  
 AND act.standard\_type = 'IC50'  
 AND act.standard\_relation = '='  
 AND act.standard\_units = 'nM'  
 AND act.standard\_value ~ '^[0-9\.]+$' --Filter for numeric values  
LIMIT 100;

* **Python Code (Topic\_CheMBL\_35\_42\_1\_mw.ipynb):**

import pandas as pd  
from rdkit import Chem  
from rdkit.Chem import Descriptors  
import matplotlib.pyplot as plt  
import os  
  
# Configure base path according to your AIMLops structure  
base\_path = "." # Adjust if your notebook is in a subdirectory  
  
# Construct the path to the CSV file  
csv\_file\_path = os.path.join(base\_path, "data", "example1.csv")  
  
# Load data from CSV  
try:  
 data = pd.read\_csv(csv\_file\_path)  
except FileNotFoundError:  
 print(f"Error: File not found at {csv\_file\_path}")  
 exit()  
  
# Function to calculate molecular weight  
def calculate\_mw(smiles):  
 try:  
 mol = Chem.MolFromSmiles(smiles)  
 if mol:  
 return Descriptors.MolWt(mol)  
 else:  
 return None  
 except:  
 return None  
  
# Apply molecular weight calculation  
data['mol\_weight'] = data['canonical\_smiles'].apply(calculate\_mw)  
  
# Convert standard\_value to numeric, handling errors  
data['standard\_value'] = pd.to\_numeric(data['standard\_value'], errors='coerce')  
  
# Drop rows with missing values in mol\_weight or standard\_value  
data = data.dropna(subset=['mol\_weight', 'standard\_value'])  
  
# Plotting  
plt.figure(figsize=(10, 6))  
plt.scatter(data['mol\_weight'], data['standard\_value'], alpha=0.5)  
plt.xlabel("Molecular Weight")  
plt.ylabel("IC50 (nM)")  
plt.title("IC50 vs. Molecular Weight")  
plt.yscale('log') # Use a logarithmic scale for better visualization  
plt.show()  
  
print(data.head())

**Explanation:**

* The SQL query retrieves molregno, standard\_value, standard\_units, and canonical\_smiles for a specific target and activity type (IC50). The AND act.standard\_value ~ '^[0-9\.]+$' clause attempts to filter the standard value column ensuring it contains numeric data.
* The Python code loads the data, calculates the molecular weight using RDKit, converts the standard\_value column to numeric type, and generates a scatter plot of IC50 vs. molecular weight.
* Error handling is included: try...except blocks in the calculate\_mw function and the pd.read\_csv call. pd.to\_numeric is also used with errors='coerce' to handle non-numeric values.
* The y-axis is plotted on a logarithmic scale because IC50 values often span several orders of magnitude.

**Example 2: LogP Calculation and Distribution Analysis**

* **Aim:** Retrieve compound SMILES strings, calculate LogP (octanol-water partition coefficient), and visualize the distribution of LogP values.
* **SQL Code (Save as ../data/example2.csv):**

-- Example 2: LogP calculation  
  
SELECT DISTINCT  
 md.molregno,  
 md.canonical\_smiles  
FROM  
 molecule\_dictionary md  
WHERE md.canonical\_smiles IS NOT NULL  
LIMIT 100;

* **Python Code (Topic\_CheMBL\_35\_42\_2\_logp.ipynb):**

import pandas as pd  
from rdkit import Chem  
from rdkit.Chem import Descriptors  
import matplotlib.pyplot as plt  
import os  
  
# Configure base path according to your AIMLops structure  
base\_path = "." # Adjust if your notebook is in a subdirectory  
  
# Construct the path to the CSV file  
csv\_file\_path = os.path.join(base\_path, "data", "example2.csv")  
  
# Load data from CSV  
try:  
 data = pd.read\_csv(csv\_file\_path)  
except FileNotFoundError:  
 print(f"Error: File not found at {csv\_file\_path}")  
 exit()  
  
# Function to calculate LogP  
def calculate\_logp(smiles):  
 try:  
 mol = Chem.MolFromSmiles(smiles)  
 if mol:  
 return Descriptors.MolLogP(mol)  
 else:  
 return None  
 except:  
 return None  
  
# Apply LogP calculation  
data['logp'] = data['canonical\_smiles'].apply(calculate\_logp)  
  
# Remove rows where LogP calculation failed  
data = data.dropna(subset=['logp'])  
  
# Plotting the distribution of LogP values  
plt.figure(figsize=(10, 6))  
plt.hist(data['logp'], bins=20, alpha=0.7)  
plt.xlabel("LogP")  
plt.ylabel("Frequency")  
plt.title("Distribution of LogP Values")  
plt.show()  
  
print(data.head())

**Explanation:**

* The SQL query retrieves unique molregno and canonical\_smiles values.
* The Python code calculates LogP using RDKit, removes any rows where the LogP calculation failed (indicated by NaN values), and generates a histogram of the LogP values.
* Error handling is included in the calculate\_logp function.

**Example 3: Hydrogen Bond Donor/Acceptor Analysis and Rule of 5 Compliance**

* **Aim:** Calculate the number of hydrogen bond donors and acceptors and determine how many compounds comply with Lipinski’s Rule of 5.
* **SQL Code (Save as ../data/example3.csv):**

-- Example 3: HBD/HBA calculation and Rule of 5 compliance  
  
SELECT DISTINCT  
 md.molregno,  
 md.canonical\_smiles  
FROM  
 molecule\_dictionary md  
WHERE md.canonical\_smiles IS NOT NULL  
LIMIT 100;

* **Python Code (Topic\_CheMBL\_35\_42\_3\_ruleof5.ipynb):**

import pandas as pd  
from rdkit import Chem  
from rdkit.Chem import Descriptors  
import os  
  
# Configure base path according to your AIMLops structure  
base\_path = "." # Adjust if your notebook is in a subdirectory  
  
# Construct the path to the CSV file  
csv\_file\_path = os.path.join(base\_path, "data", "example3.csv")  
  
# Load data from CSV  
try:  
 data = pd.read\_csv(csv\_file\_path)  
except FileNotFoundError:  
 print(f"Error: File not found at {csv\_file\_path}")  
 exit()  
  
# Function to calculate HBD and HBA  
def calculate\_hbd\_hba(smiles):  
 try:  
 mol = Chem.MolFromSmiles(smiles)  
 if mol:  
 hbd = Descriptors.NumHDonors(mol)  
 hba = Descriptors.NumHAcceptors(mol)  
 mw = Descriptors.MolWt(mol)  
 logp = Descriptors.MolLogP(mol)  
 return hbd, hba, mw, logp  
 else:  
 return None, None, None, None  
 except:  
 return None, None, None, None  
  
# Apply HBD/HBA calculation  
data[['hbd', 'hba', 'mol\_weight', 'logp']] = data['canonical\_smiles'].apply(lambda x: pd.Series(calculate\_hbd\_hba(x)))  
  
# Remove rows where HBD/HBA calculation failed  
data = data.dropna(subset=['hbd', 'hba', 'mol\_weight', 'logp'])  
  
# Rule of 5 compliance  
data['rule\_of\_5\_pass'] = ((data['mol\_weight'] <= 500) & (data['logp'] <= 5) & (data['hbd'] <= 5) & (data['hba'] <= 10))  
  
# Calculate the percentage of compounds that pass the Rule of 5  
percent\_pass = data['rule\_of\_5\_pass'].mean() \* 100  
print(f"Percentage of compounds passing the Rule of 5: {percent\_pass:.2f}%")  
  
print(data.head())

**Explanation:**

* The SQL query retrieves unique molregno and canonical\_smiles values.
* The Python code calculates the number of hydrogen bond donors (HBD) and acceptors (HBA) using RDKit.
* It then checks if each compound complies with Lipinski’s Rule of 5 (MW <= 500, LogP <= 5, HBD <= 5, HBA <= 10).
* Finally, it calculates and prints the percentage of compounds that pass the Rule of 5.

**Example 4: Analyzing Bioactivity Data for a Specific Target Family**

* **Aim:** Retrieve activity data for a target family (e.g., Kinases), and analyze the distribution of activity values.
* **SQL Code (Save as ../data/example4.csv):**

-- Example 4: Bioactivity data for a target family  
  
SELECT  
 act.molregno,  
 act.standard\_value,  
 act.standard\_units,  
 md.canonical\_smiles  
FROM  
 activities act  
JOIN  
 target\_dictionary td ON act.tid = td.tid  
JOIN  
 target\_components tc ON td.tid = tc.tid  
JOIN  
 component\_class cc ON tc.component\_id = cc.component\_id  
JOIN  
 molecule\_dictionary md ON act.molregno = md.molregno  
WHERE  
 cc.protein\_class\_desc = 'Kinase' -- Example target family  
 AND act.standard\_type = 'IC50'  
 AND act.standard\_relation = '='  
 AND act.standard\_units = 'nM'  
 AND act.standard\_value ~ '^[0-9\.]+$'  
LIMIT 100;

* **Python Code (Topic\_CheMBL\_35\_42\_4\_kinase\_activity.ipynb):**

import pandas as pd  
import matplotlib.pyplot as plt  
import numpy as np  
import os  
  
# Configure base path according to your AIMLops structure  
base\_path = "." # Adjust if your notebook is in a subdirectory  
  
# Construct the path to the CSV file  
csv\_file\_path = os.path.join(base\_path, "data", "example4.csv")  
  
# Load data from CSV  
try:  
 data = pd.read\_csv(csv\_file\_path)  
except FileNotFoundError:  
 print(f"Error: File not found at {csv\_file\_path}")  
 exit()  
  
# Convert standard\_value to numeric, handling errors  
data['standard\_value'] = pd.to\_numeric(data['standard\_value'], errors='coerce')  
  
# Remove rows with missing standard\_value  
data = data.dropna(subset=['standard\_value'])  
  
# Plotting the distribution of activity values (IC50)  
plt.figure(figsize=(10, 6))  
plt.hist(np.log10(data['standard\_value']), bins=20, alpha=0.7) # Use log scale  
plt.xlabel("Log10(IC50) (nM)")  
plt.ylabel("Frequency")  
plt.title("Distribution of IC50 Values for Kinases")  
plt.show()  
  
print(data.head())

**Explanation:**

* The SQL query retrieves activity data specifically for targets belonging to the “Kinase” protein class.
* The Python code loads the data, converts the standard\_value to numeric, handles missing values, and plots a histogram of the logarithm (base 10) of the IC50 values. Using the log scale is essential for visualizing activity data, as the values often span several orders of magnitude.

**Example 5: Substructure Search**

* **Aim:** Identify compounds containing a specific chemical substructure (e.g., a benzene ring).
* **SQL Code (Save as ../data/example5.csv):**

-- Example 5: Substructure search - compounds containing a benzene ring  
  
SELECT DISTINCT  
 md.molregno,  
 md.canonical\_smiles  
FROM  
 molecule\_dictionary md  
WHERE md.canonical\_smiles IS NOT NULL  
LIMIT 100;

* **Python Code (Topic\_CheMBL\_35\_42\_5\_substructure.ipynb):**

import pandas as pd  
from rdkit import Chem  
from rdkit.Chem import Descriptors  
from rdkit.Chem import AllChem  
import os  
  
# Configure base path according to your AIMLops structure  
base\_path = "." # Adjust if your notebook is in a subdirectory  
  
# Construct the path to the CSV file  
csv\_file\_path = os.path.join(base\_path, "data", "example5.csv")  
  
# Load data from CSV  
try:  
 data = pd.read\_csv(csv\_file\_path)  
except FileNotFoundError:  
 print(f"Error: File not found at {csv\_file\_path}")  
 exit()  
  
# Define the substructure (benzene ring)  
benzene = Chem.MolFromSmiles('c1ccccc1')  
  
# Function to check for substructure  
def has\_substructure(smiles, substructure):  
 try:  
 mol = Chem.MolFromSmiles(smiles)  
 if mol:  
 return mol.HasSubstructMatch(substructure)  
 else:  
 return False  
 except:  
 return False  
  
# Apply substructure search  
data['has\_benzene'] = data['canonical\_smiles'].apply(lambda x: has\_substructure(x, benzene))  
  
# Filter for compounds containing the substructure  
benzene\_compounds = data[data['has\_benzene'] == True]  
  
print(f"Number of compounds containing a benzene ring: {len(benzene\_compounds)}")  
print(benzene\_compounds.head())

**Explanation:**

* The SQL query retrieves unique molregno and canonical\_smiles values. The filtering based on substructure is done in Python using RDKit.
* The Python code defines a benzene ring substructure using SMILES.
* It then iterates through the SMILES strings in the DataFrame and checks if each molecule contains the benzene ring.
* Finally, it prints the number of compounds containing the benzene ring and displays the first few rows of the filtered DataFrame.

**III. Addressing Your Specific Errors:**

* **ERROR: operator does not exist: numeric ~ unknown, LINE 12: AND act.standard\_value ~ '^[0-9\.]+$':**
* This error occurs because you’re trying to use a regular expression (~) on a column that PostgreSQL doesn’t recognize as text. To fix this, you need to explicitly cast the standard\_value column to text:
* AND CAST(act.standard\_value AS TEXT) ~ '^[0-9\.]+$'
* This tells PostgreSQL to treat the standard\_value as text before applying the regular expression. This is incorporated into the SQL queries above.
* **old scikit-learn version does not support parameters squared=False in the mean\_squared\_error function:**
* This error indicates that you’re using an older version of scikit-learn. You have two options:
  1. **Upgrade scikit-learn:** The easiest solution is to upgrade your scikit-learn version to a more recent one (e.g., 0.22 or later). You can do this using pip:
  + pip install --upgrade scikit-learn
  1. **Remove squared=False:** If upgrading isn’t possible, you can remove the squared=False parameter. The default behavior of mean\_squared\_error *is* to return the mean squared error. If you need the Root Mean Squared Error (RMSE), you’ll have to calculate it manually:
  + from sklearn.metrics import mean\_squared\_error  
    import numpy as np  
      
    mse = mean\_squared\_error(y\_true, y\_predicted)  
    rmse = np.sqrt(mse) # Calculate RMSE manually

**IV. Running the Code**

Here’s a step-by-step guide to running the code:

1. **Database Connection:** Ensure you have PostgreSQL installed and the ChEMBL 35 database loaded. You’ve already got the credentials, so that’s good. Verify you can connect using psql or pgAdmin.
2. **SQL Execution:** Open pgAdmin, connect to your chembl\_35 database, open a query window, paste in *one* of the SQL queries from the examples above (e.g., the SQL query from Example 1), and execute it.
3. **Save CSV:** After the query executes successfully, export the results as a CSV file. In pgAdmin, you can typically right-click on the query results and select “Copy to File.” Save the file to the appropriate location within your ../data/ directory (e.g., ../data/example1.csv). *Make sure the name of the file matches the name referenced in the python script.*
4. **Jupyter Notebook Setup:** Open your Jupyter Notebook environment.
5. **Create Notebook:** Create a new notebook with the appropriate name (e.g., Topic\_CheMBL\_35\_42\_1\_mw.ipynb).
6. **Paste Python Code:** Paste the corresponding Python code from the example (e.g., the Python code from Example 1) into the notebook.
7. **Adjust base\_path:** *Carefully* adjust the base\_path variable in the Python code to reflect the correct location of your data directory relative to the notebook’s location. If your notebook is in the root of your project and your data directory is directly under the project root, base\_path = "." is correct. If your notebook is in a subdirectory (e.g., notebooks), and data is at the project root, then it might be something like base\_path = "..".
8. **Install Dependencies:** Make sure you have the necessary Python libraries installed. You’ll need pandas, rdkit, matplotlib, and potentially scikit-learn. You can install them using pip:

* pip install pandas rdkit matplotlib scikit-learn

1. **Run the Notebook:** Execute the cells in your Jupyter Notebook.
2. **Troubleshooting:** If you encounter any errors, carefully examine the error message and the code. Double-check the file paths, data types, and library versions. Use the error handling (try…except blocks) to help identify the source of the problem.

**V. Vietnamese Translation**

Okay, here is the breakdown and the code translated into Vietnamese:

**I. Phân Tích và Hiểu Mô Hình**

Trước khi đi sâu vào code, chúng ta hãy xác định mục tiêu. Tôi giả định “Topic\_CheMBL\_35\_42” liên quan đến một câu hỏi nghiên cứu cụ thể. Nếu không biết chủ đề chính xác, tôi sẽ tạo ra năm ví dụ dựa trên các tác vụ khám phá thuốc phổ biến sử dụng dữ liệu ChEMBL. Các ví dụ này sẽ sử dụng SQL để truy vấn cơ sở dữ liệu, Python với RDKit để xử lý và phân tích dữ liệu, đồng thời giải quyết các lỗi tiềm ẩn mà bạn đã xác định.

**Mô hình Phân tích Tổng quát:**

Cách tiếp cận chung sẽ là:

1. **Truy vấn Cơ sở Dữ liệu ChEMBL:** Sử dụng SQL để trích xuất dữ liệu liên quan (ví dụ: dữ liệu hoạt tính sinh học, cấu trúc hợp chất). Điều này bao gồm chỉ định các tiêu chí lựa chọn của bạn (ví dụ: mục tiêu, loại hoạt tính, hiệu lực).
2. **Làm sạch và Tiền xử lý Dữ liệu:** Làm sạch dữ liệu. Điều này có thể bao gồm việc loại bỏ các mục không hợp lệ, chuyển đổi đơn vị và xử lý các giá trị bị thiếu. Giải quyết lỗi “ERROR: operator does not exist: numeric ~ unknown” bằng cách truyền đúng kiểu dữ liệu.
3. **Thiết kế Đặc trưng (RDKit):** Sử dụng RDKit để tính toán các mô tả phân tử (ví dụ: trọng lượng phân tử, LogP, số lượng chất cho/nhận liên kết hydro) từ chuỗi SMILES của các hợp chất. Các mô tả này sẽ đóng vai trò là đặc trưng cho bất kỳ mô hình hóa tiếp theo nào.
4. **Phân tích và Trực quan hóa:** Thực hiện phân tích dựa trên chủ đề cụ thể. Điều này có thể bao gồm tính toán thống kê mô tả, tạo đồ thị (ví dụ: đồ thị phân tán của hoạt tính so với giá trị mô tả) hoặc xây dựng các mô hình dự đoán.
5. **Xử lý Lỗi:** Đảm bảo rằng code xử lý các sự cố tiềm ẩn, chẳng hạn như thiếu dữ liệu, kiểu dữ liệu không chính xác hoặc không tương thích phiên bản (ví dụ: sự cố squared=False).

**II. Ví dụ Code (SQL & Python)**

Dưới đây là năm ví dụ với các truy vấn SQL và code Python tương ứng. Mỗi ví dụ bao gồm một lời giải thích ngắn gọn về mục tiêu, truy vấn SQL và code Python để phân tích kết quả. Các truy vấn SQL được thiết kế để trả về tối đa 100 hàng để hạn chế tải cho máy của bạn.

**(Refer to the English section for the SQL and Python code examples. The code remains the same regardless of language.)**

**III. Giải quyết các Lỗi Cụ thể của Bạn:**

* **ERROR: operator does not exist: numeric ~ unknown, LINE 12: AND act.standard\_value ~ '^[0-9\.]+$':**
* Lỗi này xảy ra vì bạn đang cố gắng sử dụng biểu thức chính quy (~) trên một cột mà PostgreSQL không nhận dạng là văn bản. Để khắc phục điều này, bạn cần truyền cột standard\_value thành văn bản một cách rõ ràng:
* AND CAST(act.standard\_value AS TEXT) ~ '^[0-9\.]+$'
* Điều này cho PostgreSQL biết rằng hãy coi standard\_value là văn bản trước khi áp dụng biểu thức chính quy. Điều này đã được tích hợp vào các truy vấn SQL ở trên.
* **old scikit-learn version does not support parameters squared=False in the mean\_squared\_error function:**
* Lỗi này chỉ ra rằng bạn đang sử dụng phiên bản scikit-learn cũ hơn. Bạn có hai lựa chọn:
  1. **Nâng cấp scikit-learn:** Giải pháp dễ nhất là nâng cấp phiên bản scikit-learn của bạn lên phiên bản mới hơn (ví dụ: 0.22 trở lên). Bạn có thể thực hiện việc này bằng pip:
  + pip install --upgrade scikit-learn
  1. **Xóa squared=False:** Nếu không thể nâng cấp, bạn có thể xóa tham số squared=False. Hành vi mặc định của mean\_squared\_error *là* trả về mean squared error. Nếu bạn cần Root Mean Squared Error (RMSE), bạn sẽ phải tính toán thủ công:
  + from sklearn.metrics import mean\_squared\_error  
    import numpy as np  
      
    mse = mean\_squared\_error(y\_true, y\_predicted)  
    rmse = np.sqrt(mse) # Tính RMSE thủ công

**IV. Chạy Code**

Đây là hướng dẫn từng bước để chạy code:

1. **Kết nối Cơ sở Dữ liệu:** Đảm bảo bạn đã cài đặt PostgreSQL và cơ sở dữ liệu ChEMBL 35 đã được tải. Bạn đã có thông tin xác thực, vì vậy điều đó tốt. Xác minh rằng bạn có thể kết nối bằng psql hoặc pgAdmin.
2. **Thực thi SQL:** Mở pgAdmin, kết nối với cơ sở dữ liệu chembl\_35 của bạn, mở một cửa sổ truy vấn, dán *một* trong các truy vấn SQL từ các ví dụ trên (ví dụ: truy vấn SQL từ Ví dụ 1) và thực thi nó.
3. **Lưu CSV:** Sau khi truy vấn được thực thi thành công, hãy xuất kết quả dưới dạng tệp CSV. Trong pgAdmin, bạn thường có thể nhấp chuột phải vào kết quả truy vấn và chọn “Copy to File”. Lưu tệp vào vị trí thích hợp trong thư mục ../data/ của bạn (ví dụ: ../data/example1.csv). *Đảm bảo tên của tệp khớp với tên được tham chiếu trong script python.*
4. **Thiết lập Jupyter Notebook:** Mở môi trường Jupyter Notebook của bạn.
5. **Tạo Notebook:** Tạo một notebook mới với tên thích hợp (ví dụ: Topic\_CheMBL\_35\_42\_1\_mw.ipynb).
6. **Dán Code Python:** Dán code Python tương ứng từ ví dụ (ví dụ: code Python từ Ví dụ 1) vào notebook.
7. **Điều chỉnh base\_path:** *Cẩn thận* điều chỉnh biến base\_path trong code Python để phản ánh vị trí chính xác của thư mục data của bạn so với vị trí của notebook. Nếu notebook của bạn nằm trong thư mục gốc của dự án và thư mục data của bạn nằm trực tiếp bên dưới thư mục gốc của dự án, thì base\_path = "." là chính xác. Nếu notebook của bạn nằm trong một thư mục con (ví dụ: notebooks) và data nằm ở thư mục gốc của dự án, thì nó có thể là base\_path = "..".
8. **Cài đặt Dependencies:** Đảm bảo bạn đã cài đặt các thư viện Python cần thiết. Bạn sẽ cần pandas, rdkit, matplotlib và có thể là scikit-learn. Bạn có thể cài đặt chúng bằng pip:

* pip install pandas rdkit matplotlib scikit-learn

1. **Chạy Notebook:** Thực thi các ô trong Jupyter Notebook của bạn.
2. **Khắc phục Sự cố:** Nếu bạn gặp bất kỳ lỗi nào, hãy kiểm tra cẩn thận thông báo lỗi và code. Kiểm tra kỹ đường dẫn tệp, kiểu dữ liệu và phiên bản thư viện. Sử dụng tính năng xử lý lỗi (các khối try…except) để giúp xác định nguồn gốc của sự cố.

**Key takeaways for the Vietnamese version:**

* I tried to maintain the technical vocabulary as consistent as possible for accurate communication.
* The code samples and instructions are the same as in the English version.
* The main focus is providing clear guidance on the overall workflow. I hope this detailed explanation and code examples will help you progress with your project! Remember to adjust the code and SQL queries to match your specific research question. Good luck!