PharmApp Suite

🧠 AI for Drug Discovery and Development 💉

🧩 Topic: Tìm kiếm hợp chất lead cho bệnh hiếm gặp

Tuyệt vời! Tôi rất vui được hỗ trợ bạn trong dự án phân tích dữ liệu ChEMBL 35 kết hợp RDKit. Với vai trò là một chuyên gia R&D Dược học, tôi sẽ tập trung vào việc đưa ra các phân tích, hướng dẫn và code mẫu có giá trị nhất để giúp bạn khai thác tối đa dữ liệu ChEMBL cho mục đích nghiên cứu và phát triển thuốc.

Dưới đây là phân tích chi tiết, hướng dẫn song ngữ và các ví dụ code SQL và Python mẫu dựa trên thông tin bạn cung cấp.

1. Phân tích mô hình (Analysis Model)

Chúng ta sẽ xây dựng một quy trình phân tích dữ liệu ChEMBL 35 để xác định các hợp chất có hoạt tính sinh học tiềm năng và khám phá mối quan hệ cấu trúc-hoạt tính (SAR). Mô hình này bao gồm các bước chính sau:

- **Trích xuất dữ liệu (Data Extraction):** Sử dụng SQL để truy vấn và trích xuất thông tin cần thiết từ cơ sở dữ liệu ChEMBL 35 (ví dụ: thông tin về hợp chất, hoạt tính sinh học, mục tiêu).
- **Tiền xử lý dữ liệu (Data Preprocessing):** Làm sạch và chuẩn hóa dữ liệu, xử lý các giá trị bị thiếu, loại bỏ các hợp chất trùng lặp hoặc không hợp lệ.
- **Tính toán descriptor phân tử (Molecular Descriptor Calculation):** Sử dụng RDKit để tính toán các descriptor phân tử (ví dụ: trọng lượng phân tử, logP, số lượng liên kết hydro) từ cấu trúc hóa học của các hợp chất.
- **Phân tích thống kê và mô hình hóa (Statistical Analysis and Modeling):** Sử dụng các phương pháp thống kê và học máy để phân tích mối quan hệ giữa các descriptor phân tử và hoạt tính sinh học, xây dựng mô hình dự đoán hoạt tính, xác định các hợp chất tiềm năng.
- **Trực quan hóa dữ liệu (Data Visualization):** Sử dụng các công cụ trực quan hóa để khám phá dữ liệu, trình bày kết quả và giao tiếp với các nhà khoa học khác.

2. Hướng dẫn song ngữ (Bilingual Guidance)

2.1. SQL

- **English:** SQL (Structured Query Language) is used to interact with the ChEMBL 35 database. You can use SQL to extract specific data based on your research question, such as compound properties, bioactivity data, and target information.
- **Tiếng Việt:** SQL (Ngôn ngữ truy vấn cấu trúc) được sử dụng để tương tác với cơ sở dữ liệu ChEMBL 35. Bạn có thể sử dụng SQL để trích xuất dữ liệu cụ thể dựa trên câu hỏi nghiên cứu của bạn, chẳng hạn như thuộc tính hợp chất, dữ liệu hoạt tính sinh học và thông tin mục tiêu.

2.2. Python & RDKit

- **English:** Python is used for data preprocessing, descriptor calculation (using RDKit), statistical analysis, and machine learning. RDKit is a powerful cheminformatics toolkit that allows you to manipulate and analyze chemical structures.
- **Tiếng Việt:** Python được sử dụng để tiền xử lý dữ liệu, tính toán descriptor (sử dụng RDKit), phân tích thống kê và học máy. RDKit là một bộ công cụ tin học hóa học mạnh mẽ cho phép bạn thao tác và phân tích cấu trúc hóa học.

3. Ví dụ Code SQL và Python (SQL and Python Code Examples)

3.1. Ví dụ SQL (SQL Examples)

```sql
-- Ví dụ 1: Lấy 100 hợp chất có hoạt tính ức chế enzyme EGFR (EGFR inhibition activity)
-- Example 1: Get 100 compounds with EGFR inhibition activity
SELECT DISTINCT molregno, act.standard_value, act.standard_units, compounds.canonical_smiles
FROM activities act
JOIN target_dictionary td ON act.tid = td.tid
JOIN assays ass ON act.assay_id = ass.assay_id
JOIN molecule_dictionary md ON act.molregno = md.molregno
JOIN compound_structures compounds ON md.molregno = compounds.molregno
WHERE td.target_name LIKE '%EGFR%'
AND ass.assay_type = 'B' -- 'B' for binding assays
AND act.standard_type = 'IC50'
AND act.standard_units = 'nM'
AND act.standard_value IS NOT NULL
AND act.standard_value < 100 -- IC50 < 100 nM (potent compounds)
AND act.standard_value ~ '^[0-9\.]+$' -- Giá trị standard_value chỉ chứa số và dấu chấm
LIMIT 100;

-- Ví dụ 2: Lấy thông tin cơ bản của 100 hợp chất từ bảng molecule_dictionary
-- Example 2: Get basic information of 100 compounds from molecule_dictionary table
SELECT molregno, chembl_id, pref_name
FROM molecule_dictionary
LIMIT 100;

-- Ví dụ 3: Lấy 100 dòng dữ liệu từ bảng activities với điều kiện lọc
-- Example 3: Retrieve 100 rows from the 'activities' table with filtering conditions.
SELECT act.molregno, md.chembl_id, td.target_name, act.standard_type,
act.standard_value, act.standard_units
FROM activities act
JOIN molecule_dictionary md ON act.molregno = md.molregno
JOIN target_dictionary td ON act.tid = td.tid
WHERE act.standard_type = 'IC50'  -- Lọc theo loại hoạt tính 'IC50'
  AND act.standard_units = 'nM'     -- Lọc theo đơn vị 'nM'
LIMIT 100;

-- Ví dụ 4: Lấy 100 dòng dữ liệu từ bảng compound_structures
-- Example 4: Retrieve 100 rows from the 'compound_structures' table.
SELECT molregno, canonical_smiles
FROM compound_structures
LIMIT 100;

-- Ví dụ 5: Lấy 100 dòng dữ liệu từ bảng target_dictionary
-- Example 5: Retrieve 100 rows from the 'target_dictionary' table.
SELECT tid, target_name, target_type
FROM target_dictionary
LIMIT 100;
```

Giải thích lỗi SQL (Explanation of the SQL error)

Lỗi "ERROR: operator does not exist: numeric ~ unknown, LINE 12: AND act.standard_value ~ '[1]+$'" xảy ra vì toán tử ~ (regular expression match) không được hỗ trợ trực tiếp trên kiểu dữ liệu `numeric` trong PostgreSQL.

Để khắc phục, bạn có thể chuyển đổi cột `standard_value` sang kiểu `text` trước khi so sánh:

```sql
AND CAST(act.standard_value AS TEXT) ~ '^[0-9\.]+$'
```

[1] 0-9.

3.2. Ví dụ Python (Python Examples)

```python
import os
import pandas as pd
from rdkit import Chem
from rdkit.Chem import Descriptors
from rdkit.Chem import AllChem
import psycopg2

# Cấu hình kết nối database (Database connection configuration)
db_ip = "192.168.206.136"
db_user = "rd"
db_pass = "rd"
db_name = "chembl_35"

# Hàm kết nối database (Database connection function)
def connect_to_db(ip, user, password, database):
    conn = None
    try:
        conn = psycopg2.connect(host=ip, user=user, password=password,
database=database)
        print("Connected to PostgreSQL successfully!")
    except psycopg2.Error as e:
        print(f"Error connecting to PostgreSQL: {e}")
    return conn

# Hàm lấy dữ liệu từ database (Function to fetch data from the database)
def fetch_data(conn, query, limit=100):
    try:
        df = pd.read_sql_query(query + f" LIMIT {limit}", conn)
        return df
    except psycopg2.Error as e:
        print(f"Error fetching data: {e}")
        return None

# Hàm tính toán descriptor phân tử (Function to calculate molecular descriptors)
def calculate_descriptors(smiles):
    mol = Chem.MolFromSmiles(smiles)
    if mol is not None:
        # Tính toán các descriptor cơ bản (Calculate basic descriptors)
        mw = Descriptors.MolWt(mol)
        logp = Descriptors.MolLogP(mol)
        num_hba = Descriptors.NumHAcceptors(mol)
        num_hbd = Descriptors.NumHDonors(mol)
        tpsa = Descriptors.TPSA(mol)

        # Tính toán thêm các descriptor khác nếu cần (Calculate other descriptors if
needed)
        # Ví dụ: số lượng vòng, số lượng nguyên tử (e.g., number of rings, number of
atoms)
        num_rings = Chem.GetSSSR(mol)
        num_atoms = mol.GetNumAtoms()

        return mw, logp, num_hba, num_hbd, tpsa, num_rings, num_atoms
    else:
        return None, None, None, None, None, None, None

# Thiết lập đường dẫn cơ sở (Set up the base path)
base_path = "../data"
```

```
# 1. Kết nối đến PostgreSQL (Connect to PostgreSQL)
conn = connect_to_db(db_ip, db_user, db_pass, db_name)

if conn:
    # 2. Lấy dữ liệu từ bảng compound_structures (Fetch data from compound_structures
table)
    query = "SELECT molregno, canonical_smiles FROM compound_structures"
    df = fetch_data(conn, query)

    if df is not None:
        # In ra thông tin DataFrame (Print DataFrame information)
        print("DataFrame info:")
        print(df.info())

        # 3. Tính toán descriptor và thêm vào DataFrame (Calculate descriptors and add
to DataFrame)
        df[['MW', 'LogP', 'NumHBA', 'NumHBD', 'TPSA', 'NumRings', 'NumAtoms']] =
df['canonical_smiles'].apply(
            lambda x: pd.Series(calculate_descriptors(x))
        )

        # In ra 5 dòng đầu của DataFrame với descriptor (Print the first 5 rows of the
DataFrame with descriptors)
        print("\nDataFrame with descriptors (first 5 rows):")
        print(df.head())

        # 4. Lưu DataFrame vào file CSV (Save DataFrame to CSV file)
        output_file = os.path.join(base_path,
"compound_structures_with_descriptors.csv")
        df.to_csv(output_file, index=False)
        print(f"\nDataFrame saved to: {output_file}")

    # Đóng kết nối (Close connection)
    conn.close()
else:
    print("Failed to connect to the database.")
```

Ví dụ 2: Phân tích hoạt tính sinh học (Bioactivity Analysis)

```
import pandas as pd
import psycopg2
import os
from rdkit import Chem
from rdkit.Chem import Descriptors

# Database connection details
db_ip = "192.168.206.136"
db_user = "rd"
db_pass = "rd"
db_name = "chembl_35"

# Function to connect to PostgreSQL
def connect_to_db(ip, user, password, database):
    conn = None
    try:
        conn = psycopg2.connect(host=ip, user=user, password=password,
database=database)
        print("Connected to PostgreSQL successfully!")
    except psycopg2.Error as e:
        print(f"Error connecting to PostgreSQL: {e}")
```

```
    return conn

# Function to fetch data from the database
def fetch_data(conn, query, limit=100):
    try:
        df = pd.read_sql_query(query + f" LIMIT {limit}", conn)
        return df
    except psycopg2.Error as e:
        print(f"Error fetching data: {e}")
        return None

# Function to calculate molecular descriptors
def calculate_descriptors(smiles):
    mol = Chem.MolFromSmiles(smiles)
    if mol is not None:
        mw = Descriptors.MolWt(mol)
        logp = Descriptors.MolLogP(mol)
        num_hba = Descriptors.NumHAcceptors(mol)
        num_hbd = Descriptors.NumHDonors(mol)
        tpsa = Descriptors.TPSA(mol)
        return mw, logp, num_hba, num_hbd, tpsa
    else:
        return None, None, None, None, None

# Base path for saving data
base_path = "../data"

# 1. Connect to PostgreSQL
conn = connect_to_db(db_ip, db_user, db_pass, db_name)

if conn:
    # 2. Fetch bioactivity data
    query = """
    SELECT act.molregno, md.chembl_id, td.target_name, act.standard_type,
act.standard_value, act.standard_units, compounds.canonical_smiles
    FROM activities act
    JOIN molecule_dictionary md ON act.molregno = md.molregno
    JOIN target_dictionary td ON act.tid = td.tid
    JOIN compound_structures compounds ON md.molregno = compounds.molregno
    WHERE act.standard_type = 'IC50'
      AND act.standard_units = 'nM'
      AND compounds.canonical_smiles IS NOT NULL
    """
    df = fetch_data(conn, query)

    if df is not None:
        # 3. Calculate descriptors and add to DataFrame
        df[['MW', 'LogP', 'NumHBA', 'NumHBD', 'TPSA']] = df['canonical_smiles'].apply(
            lambda x: pd.Series(calculate_descriptors(x))
        )

        # Convert standard_value to numeric, handling errors
        df['standard_value'] = pd.to_numeric(df['standard_value'], errors='coerce')

        # Basic statistics
        print("Basic Statistics:")
        print(df[['standard_value', 'MW', 'LogP', 'NumHBA', 'NumHBD',
'TPSA']].describe())

        # Save DataFrame to CSV file
```

```
        output_file = os.path.join(base_path, "bioactivity_data_with_descriptors.csv")
        df.to_csv(output_file, index=False)
        print(f"DataFrame saved to: {output_file}")

    # Close connection
    conn.close()
else:
    print("Failed to connect to the database.")
```

Ví dụ 3: Mô hình hóa SAR (SAR Modeling)

```
import pandas as pd
import psycopg2
import os
from rdkit import Chem
from rdkit.Chem import Descriptors
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error

# Database connection details
db_ip = "192.168.206.136"
db_user = "rd"
db_pass = "rd"
db_name = "chembl_35"

# Function to connect to PostgreSQL
def connect_to_db(ip, user, password, database):
    conn = None
    try:
        conn = psycopg2.connect(host=ip, user=user, password=password,
database=database)
        print("Connected to PostgreSQL successfully!")
    except psycopg2.Error as e:
        print(f"Error connecting to PostgreSQL: {e}")
    return conn

# Function to fetch data from the database
def fetch_data(conn, query, limit=100):
    try:
        df = pd.read_sql_query(query + f" LIMIT {limit}", conn)
        return df
    except psycopg2.Error as e:
        print(f"Error fetching data: {e}")
        return None

# Function to calculate molecular descriptors
def calculate_descriptors(smiles):
    mol = Chem.MolFromSmiles(smiles)
    if mol is not None:
        mw = Descriptors.MolWt(mol)
        logp = Descriptors.MolLogP(mol)
        num_hba = Descriptors.NumHAcceptors(mol)
        num_hbd = Descriptors.NumHDonors(mol)
        tpsa = Descriptors.TPSA(mol)
        return mw, logp, num_hba, num_hbd, tpsa
    else:
        return None, None, None, None, None

# Base path for saving data
```

```
base_path = "../data"

# 1. Connect to PostgreSQL
conn = connect_to_db(db_ip, db_user, db_pass, db_name)

if conn:
    # 2. Fetch bioactivity data
    query = """
    SELECT act.molregno, md.chembl_id, td.target_name, act.standard_type,
act.standard_value, act.standard_units, compounds.canonical_smiles
    FROM activities act
    JOIN molecule_dictionary md ON act.molregno = md.molregno
    JOIN target_dictionary td ON act.tid = td.tid
    JOIN compound_structures compounds ON md.molregno = compounds.molregno
    WHERE act.standard_type = 'IC50'
      AND act.standard_units = 'nM'
      AND compounds.canonical_smiles IS NOT NULL
    """
    df = fetch_data(conn, query)

    if df is not None:
        # 3. Calculate descriptors and add to DataFrame
        df[['MW', 'LogP', 'NumHBA', 'NumHBD', 'TPSA']] = df['canonical_smiles'].apply(
            lambda x: pd.Series(calculate_descriptors(x))
        )

        # Convert standard_value to numeric, handling errors
        df['standard_value'] = pd.to_numeric(df['standard_value'], errors='coerce')

        # Drop rows with NaN values
        df = df.dropna()

        # 4. Prepare data for modeling
        X = df[['MW', 'LogP', 'NumHBA', 'NumHBD', 'TPSA']]
        y = df['standard_value']

        # 5. Split data into training and testing sets
        X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

        # 6. Train a linear regression model
        model = LinearRegression()
        model.fit(X_train, y_train)

        # 7. Make predictions
        y_pred = model.predict(X_test)

        # 8. Evaluate the model
        mse = mean_squared_error(y_test, y_pred)
        print(f"Mean Squared Error: {mse}")

        # Save DataFrame to CSV file
        output_file = os.path.join(base_path, "bioactivity_data_with_descriptors.csv")
        df.to_csv(output_file, index=False)
        print(f"DataFrame saved to: {output_file}")

    # Close connection
    conn.close()
```

```
else:
    print("Failed to connect to the database.")
```

Ví dụ 4: Lọc hợp chất theo mục tiêu (Target-Based Compound Filtering)

```
import pandas as pd
import psycopg2
import os
from rdkit import Chem
from rdkit.Chem import Descriptors

# Database connection details
db_ip = "192.168.206.136"
db_user = "rd"
db_pass = "rd"
db_name = "chembl_35"

# Function to connect to PostgreSQL
def connect_to_db(ip, user, password, database):
    conn = None
    try:
        conn = psycopg2.connect(host=ip, user=user, password=password,
database=database)
        print("Connected to PostgreSQL successfully!")
    except psycopg2.Error as e:
        print(f"Error connecting to PostgreSQL: {e}")
    return conn

# Function to fetch data from the database
def fetch_data(conn, query, limit=100):
    try:
        df = pd.read_sql_query(query + f" LIMIT {limit}", conn)
        return df
    except psycopg2.Error as e:
        print(f"Error fetching data: {e}")
        return None

# Function to calculate molecular descriptors
def calculate_descriptors(smiles):
    mol = Chem.MolFromSmiles(smiles)
    if mol is not None:
        mw = Descriptors.MolWt(mol)
        logp = Descriptors.MolLogP(mol)
        num_hba = Descriptors.NumHAcceptors(mol)
        num_hbd = Descriptors.NumHDonors(mol)
        tpsa = Descriptors.TPSA(mol)
        return mw, logp, num_hba, num_hbd, tpsa
    else:
        return None, None, None, None, None

# Base path for saving data
base_path = "../data"

# 1. Connect to PostgreSQL
conn = connect_to_db(db_ip, db_user, db_pass, db_name)

if conn:
    # 2. Fetch bioactivity data for a specific target (e.g., EGFR)
    target_name = "EGFR"  # Replace with your target of interest
    query = f"""
```

```
    SELECT act.molregno, md.chembl_id, td.target_name, act.standard_type,
act.standard_value, act.standard_units, compounds.canonical_smiles
    FROM activities act
    JOIN molecule_dictionary md ON act.molregno = md.molregno
    JOIN target_dictionary td ON act.tid = td.tid
    JOIN compound_structures compounds ON md.molregno = compounds.molregno
    WHERE td.target_name LIKE '%{target_name}%'
      AND act.standard_type = 'IC50'
      AND act.standard_units = 'nM'
      AND compounds.canonical_smiles IS NOT NULL
    """
    df = fetch_data(conn, query)

    if df is not None:
        # 3. Calculate descriptors and add to DataFrame
        df[['MW', 'LogP', 'NumHBA', 'NumHBD', 'TPSA']] = df['canonical_smiles'].apply(
            lambda x: pd.Series(calculate_descriptors(x))
        )

        # Convert standard_value to numeric, handling errors
        df['standard_value'] = pd.to_numeric(df['standard_value'], errors='coerce')

        # Save DataFrame to CSV file
        output_file = os.path.join(base_path, f"{target_name}
_bioactivity_data_with_descriptors.csv")
        df.to_csv(output_file, index=False)
        print(f"DataFrame saved to: {output_file}")

    # Close connection
    conn.close()
else:
    print("Failed to connect to the database.")
```

Ví dụ 5: Phân tích tương quan descriptor (Descriptor Correlation Analysis)

```
import pandas as pd
import psycopg2
import os
from rdkit import Chem
from rdkit.Chem import Descriptors
import matplotlib.pyplot as plt
import seaborn as sns

# Database connection details
db_ip = "192.168.206.136"
db_user = "rd"
db_pass = "rd"
db_name = "chembl_35"

# Function to connect to PostgreSQL
def connect_to_db(ip, user, password, database):
    conn = None
    try:
        conn = psycopg2.connect(host=ip, user=user, password=password,
database=database)
        print("Connected to PostgreSQL successfully!")
    except psycopg2.Error as e:
        print(f"Error connecting to PostgreSQL: {e}")
    return conn
```

```
# Function to fetch data from the database
def fetch_data(conn, query, limit=100):
    try:
        df = pd.read_sql_query(query + f" LIMIT {limit}", conn)
        return df
    except psycopg2.Error as e:
        print(f"Error fetching data: {e}")
        return None

# Function to calculate molecular descriptors
def calculate_descriptors(smiles):
    mol = Chem.MolFromSmiles(smiles)
    if mol is not None:
        mw = Descriptors.MolWt(mol)
        logp = Descriptors.MolLogP(mol)
        num_hba = Descriptors.NumHAcceptors(mol)
        num_hbd = Descriptors.NumHDonors(mol)
        tpsa = Descriptors.TPSA(mol)
        return mw, logp, num_hba, num_hbd, tpsa
    else:
        return None, None, None, None, None

# Base path for saving data
base_path = "../data"

# 1. Connect to PostgreSQL
conn = connect_to_db(db_ip, db_user, db_pass, db_name)

if conn:
    # 2. Fetch bioactivity data
    query = """
    SELECT act.molregno, md.chembl_id, td.target_name, act.standard_type,
act.standard_value, act.standard_units, compounds.canonical_smiles
    FROM activities act
    JOIN molecule_dictionary md ON act.molregno = md.molregno
    JOIN target_dictionary td ON act.tid = td.tid
    JOIN compound_structures compounds ON md.molregno = compounds.molregno
    WHERE act.standard_type = 'IC50'
      AND act.standard_units = 'nM'
      AND compounds.canonical_smiles IS NOT NULL
    """
    df = fetch_data(conn, query)

    if df is not None:
        # 3. Calculate descriptors and add to DataFrame
        df[['MW', 'LogP', 'NumHBA', 'NumHBD', 'TPSA']] = df['canonical_smiles'].apply(
            lambda x: pd.Series(calculate_descriptors(x))
        )

        # Convert standard_value to numeric, handling errors
        df['standard_value'] = pd.to_numeric(df['standard_value'], errors='coerce')

        # Drop rows with NaN values
        df = df.dropna()

        # 4. Calculate the correlation matrix
        correlation_matrix = df[['standard_value', 'MW', 'LogP', 'NumHBA', 'NumHBD',
'TPSA']].corr()

        # 5. Visualize the correlation matrix using a heatmap
```

```
        plt.figure(figsize=(10, 8))
        sns.heatmap(correlation_matrix, annot=True, cmap="coolwarm", linewidths=.5)
        plt.title("Correlation Matrix of Bioactivity and Descriptors")
        plt.show()

        # Save DataFrame to CSV file
        output_file = os.path.join(base_path, "bioactivity_data_with_descriptors.csv")
        df.to_csv(output_file, index=False)
        print(f"DataFrame saved to: {output_file}")

    # Close connection
    conn.close()
else:
    print("Failed to connect to the database.")
```

Giải quyết lỗi Scikit-learn (Resolving the Scikit-learn Error)

Nếu bạn gặp lỗi "TypeError: mean_squared_error() got an unexpected keyword argument 'squared'", điều này có nghĩa là phiên bản Scikit-learn bạn đang sử dụng quá cũ và không hỗ trợ tham số `squared`.

Để khắc phục, bạn có thể nâng cấp Scikit-learn lên phiên bản mới hơn bằng lệnh:

```
pip install --upgrade scikit-learn
```

Hoặc, bạn có thể tính căn bậc hai của MSE để có được RMSE (Root Mean Squared Error), tương đương với `squared=False`:

```
from sklearn.metrics import mean_squared_error
import numpy as np

mse = mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse)
print(f"Root Mean Squared Error: {rmse}")
```

Lưu ý quan trọng (Important Notes)

- **AIMLOps Template:** Hãy nhớ tuân thủ cấu trúc thư mục AIMLOps Template để đảm bảo tính nhất quán và khả năng tái sử dụng của dự án.
- **Giới hạn dữ liệu:** Vì lý do hiệu suất, chúng ta chỉ lấy 100 dòng dữ liệu trong các ví dụ này. Trong thực tế, bạn có thể cần phải xử lý lượng dữ liệu lớn hơn nhiều.
- **Kiểm tra và xác thực:** Luôn kiểm tra và xác thực dữ liệu của bạn để đảm bảo tính chính xác và độ tin cậy của kết quả.
- **Tùy chỉnh:** Các ví dụ code này chỉ là điểm khởi đầu. Bạn có thể cần phải tùy chỉnh chúng để phù hợp với nhu cầu cụ thể của dự án của bạn.

Chúc bạn thành công với dự án phân tích dữ liệu ChEMBL 35! Nếu bạn có bất kỳ câu hỏi nào khác, đừng ngần ngại hỏi.