PharmApp Suite

🧠 AI for Drug Discovery and Development 💉

🧩 Topic: Filtering rule-of-five compliant molecules

Okay, I understand. You're working on a Chembl 35 data analysis project using RDKit within an AIMLops framework. You're facing issues with SQL syntax and scikit-learn compatibility. We'll address these, build SQL queries to extract data, and construct Python code in Jupyter Notebooks (following your naming convention) with error handling and data manipulation using RDKit. The goal is to filter data down to a manageable size (100 rows) for your machine and provide 5 examples.

Here's the analysis, code, and examples in both English and Vietnamese:

1. Analysis of the Analysis Model

- **Objective:** The project aims to analyze Chembl 35 data for drug research and development. The focus is likely on extracting relevant information about chemical compounds and their biological activities, then using this data to potentially build predictive models or identify promising drug candidates.
- **Data Source:** Chembl 35 (accessed via PostgreSQL)
- **Tools:**
    - **PostgreSQL (psql):** Used to query and extract data from the Chembl 35 database. We'll use SQL to filter the data to the relevant compounds and activity measurements.
    - **RDKit:** A powerful cheminformatics toolkit for manipulating and analyzing chemical structures (SMILES strings). It can be used to generate molecular descriptors, calculate properties, and perform similarity searches.
    - **Jupyter Notebook:** An interactive environment for writing and executing Python code, creating visualizations, and documenting your analysis.
    - **scikit-learn:** a powerful Machine Learning library that offers various model implementation, such as LinearRegression, DecisionTree, RandomForest…
- **Key Steps:**
    1. **Data Extraction:** Use SQL to extract compound information (e.g., SMILES strings, molecule structure) and biological activity data (e.g., IC50, Ki values) from Chembl 35. The SQL will need to handle the `numeric ~ unknown` error.
    2. **Data Preprocessing:** Clean and transform the extracted data. This might involve:
        - Converting SMILES strings to RDKit Mol objects.
        - Handling missing or invalid data.
        - Converting activity values to a consistent unit (e.g., pIC50).
    3. **Feature Generation:** Generate molecular descriptors using RDKit. These descriptors represent the chemical properties of the molecules.
    4. **Analysis & Modeling (Optional):** Use the molecular descriptors and activity data to build predictive models (e.g., regression models to predict activity).
    5. **Visualization:** Create plots and charts to visualize the data and the results of your analysis.

2. SQL and Python Code

SQL Code (SQL file: `Topic_CheMBL_35_4.sql` saved in `../data/`)

```sql
-- Corrected SQL query to avoid "numeric ~ unknown" error and limit to 100 rows
-- and to get Standard Type is IC50
SELECT
```

```
    md.chembl_id,
    cs.canonical_smiles,
    act.standard_type,
    act.standard_value,
    act.standard_units
FROM
    compound_structures cs
JOIN
    molecule_dictionary md ON cs.molregno = md.molregno
JOIN
    activities act ON md.molregno = act.molregno
WHERE act.standard_type = 'IC50'
  AND act.standard_value IS NOT NULL
  AND act.standard_value::TEXT SIMILAR TO '[0-9\.]+' -- Cast to text and use SIMILAR
TO
ORDER BY md.molregno
LIMIT 100;

--Save this output to data/Topic_CheMBL_35_4.csv
```

Explanation:

- `act.standard_value::TEXT SIMILAR TO '[0-9\.]+'`: This line is crucial. The original `act.standard_value ~ '^[0-9\.]+$'` uses a regular expression operator that might not be directly compatible with numeric types in PostgreSQL. We cast `act.standard_value` to `TEXT` and then use the `SIMILAR TO` operator which performs pattern matching on strings. This avoids the `numeric ~ unknown` error.
- `LIMIT 100`: Limits the result set to 100 rows for faster processing.
- `act.standard_type = 'IC50'` : Filter for only IC50 values to make it more specific.
- The output of this SQL query is saved into the data folder, create a file `Topic_CheMBL_35_4.csv`

Python Code (Jupyter Notebook: `Topic_CheMBL_35_4_1_Data_Extraction_and_Preprocessing.ipynb`)

```
import os
import pandas as pd
from rdkit import Chem
from rdkit.Chem import Descriptors
import numpy as np

# Define the base path for your project
base_path = "."  # Assuming the notebook is in the root directory

# Construct the path to the CSV file
csv_file_path = os.path.join(base_path, "data", "Topic_CheMBL_35_4.csv")

# Load the data from the CSV file
try:
    df = pd.read_csv(csv_file_path)
    print("Data loaded successfully.")
except FileNotFoundError:
    print(f"Error: CSV file not found at {csv_file_path}")
    exit()
except Exception as e:
    print(f"Error loading CSV file: {e}")
    exit()

# Display the first few rows of the DataFrame
print("\nFirst 5 rows of the data:")
```

```
print(df.head())

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

# Drop rows with missing standard_value
df = df.dropna(subset=['standard_value'])

# Filter out any standard_value that is zero
df = df[df['standard_value'] != 0]

# Convert IC50 to pIC50 (example transformation)
# pIC50 = -log10(IC50 in Molar)
df['pIC50'] = -np.log10(df['standard_value'] * 1e-9)  # Convert nM to M

# RDKit Mol object creation
def smiles_to_mol(smiles):
    try:
        mol = Chem.MolFromSmiles(smiles)
        if mol is not None:
            return mol
        else:
            return None
    except:
        return None

df['mol'] = df['canonical_smiles'].apply(smiles_to_mol)

# Drop rows where RDKit failed to create a Mol object
df = df.dropna(subset=['mol'])

# Molecular Descriptor Calculation (Example: Molecular Weight)
def calculate_mw(mol):
    try:
        return Descriptors.MolWt(mol)
    except:
        return None

df['molecular_weight'] = df['mol'].apply(calculate_mw)

#Drop rows where molecular weight failed to create a Mol object
df = df.dropna(subset=['molecular_weight'])

print("\nData after preprocessing:")
print(df.head())
```

Explanation:

- **Error Handling:** The code includes `try...except` blocks to handle potential errors during file loading, SMILES parsing, and descriptor calculation.
- **Path Management:** Uses `os.path.join` for constructing file paths, making the code more portable.
- **Data Cleaning:** Converts `standard_value` to numeric and removes rows with missing values. It also filter `standard_value !=0`
- **pIC50 Conversion:** Calculates pIC50 values (a common practice in drug discovery).
- **RDKit Integration:** Creates RDKit `Mol` objects from SMILES strings.

- **Molecular Descriptor Calculation:** Calculates molecular weight as an example. You can add more descriptors as needed.

Python Code (Jupyter Notebook: `Topic_CheMBL_35_4_2_Analysis_and_Visualization.ipynb`)

```
import os
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Define the base path for your project
base_path = "."  # Assuming the notebook is in the root directory

# Construct the path to the CSV file
csv_file_path = os.path.join(base_path, "data", "Topic_CheMBL_35_4.csv")

# Load the data from the CSV file
try:
    df = pd.read_csv(csv_file_path)
    print("Data loaded successfully.")
except FileNotFoundError:
    print(f"Error: CSV file not found at {csv_file_path}")
    exit()
except Exception as e:
    print(f"Error loading CSV file: {e}")
    exit()

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

# Drop rows with missing standard_value
df = df.dropna(subset=['standard_value'])

# Filter out any standard_value that is zero
df = df[df['standard_value'] != 0]

# Convert IC50 to pIC50 (example transformation)
# pIC50 = -log10(IC50 in Molar)
df['pIC50'] = -np.log10(df['standard_value'] * 1e-9)  # Convert nM to M

# --- Visualization Examples ---

# 1. Distribution of pIC50 values
plt.figure(figsize=(8, 6))
sns.histplot(df['pIC50'], kde=True)
plt.title('Distribution of pIC50 Values')
plt.xlabel('pIC50')
plt.ylabel('Frequency')
plt.show()

# 2. Scatter plot of Molecular Weight vs. pIC50 (requires calculation of molecular
weight in previous notebook)
try:
    plt.figure(figsize=(8, 6))
    sns.scatterplot(x='molecular_weight', y='pIC50', data=df)
    plt.title('Molecular Weight vs. pIC50')
    plt.xlabel('Molecular Weight')
    plt.ylabel('pIC50')
    plt.show()
```

```
except KeyError:
    print("Error: 'molecular_weight' column not found.  Make sure to run the first notebook to calculate it.")

# 3. Box plot of pIC50 values for different standard_types (if you have multiple standard_types after filtering)
if len(df['standard_type'].unique()) > 1:
    plt.figure(figsize=(8, 6))
    sns.boxplot(x='standard_type', y='pIC50', data=df)
    plt.title('pIC50 Values by Standard Type')
    plt.xlabel('Standard Type')
    plt.ylabel('pIC50')
    plt.show()
else:
    print("Only one standard type found, skipping boxplot.")

# 4. Correlation heatmap of numerical features (after descriptor calculation)
try:
    numerical_cols = ['pIC50', 'molecular_weight']  # Add other numerical columns you calculate
    corr_matrix = df[numerical_cols].corr()
    plt.figure(figsize=(8, 6))
    sns.heatmap(corr_matrix, annot=True, cmap='coolwarm')
    plt.title('Correlation Heatmap')
    plt.show()
except KeyError:
    print("Error: Not enough numerical columns to calculate correlation.  Calculate more descriptors in the first notebook.")

# 5.  Basic statistics of pIC50
print("\nDescriptive Statistics for pIC50:")
print(df['pIC50'].describe())
```

Explanation:

- **Loads Preprocessed Data:** Assumes you've run the first notebook (`Topic_CheMBL_35_4_1_Data_Extraction_and_Preprocessing.ipynb`) and have a CSV file with the preprocessed data and calculated molecular weight.
- **Visualization with `matplotlib` and `seaborn`:** Creates several common plots:
    - Histogram of pIC50 values
    - Scatter plot of molecular weight vs. pIC50
    - Box plot of pIC50 values by standard type (if applicable)
    - Correlation heatmap
- **Error Handling:** Includes `try...except` blocks to handle cases where the required columns are missing (e.g., `molecular_weight` if you haven't calculated it yet).
- **Conditional Plotting:** The boxplot is only created if there are multiple `standard_type` values in the data.

3. Five Examples (Illustrating Use Cases)

Here are five example scenarios of how you might use this code and the resulting data in a drug discovery project:

1. **Target Prioritization:** You want to prioritize a target for drug discovery based on the availability of high-quality activity data. You extract IC50 values for compounds tested against the target and analyze the distribution of pIC50 values. A target with a wide range of potent inhibitors (high pIC50 values) might be a good starting point.

2. **Lead Identification:** You are looking for lead compounds with specific properties. You can filter the data based on pIC50 values (e.g., find compounds with pIC50 > 7), molecular weight (e.g., MW < 500), and other calculated descriptors to identify promising leads.
3. **Structure-Activity Relationship (SAR) Analysis:** You want to understand how changes in chemical structure affect activity. You can use RDKit to calculate a variety of molecular descriptors and then correlate these descriptors with pIC50 values. This can help you identify key structural features that contribute to activity.
4. **Virtual Screening:** You have a database of compounds and want to predict their activity against a target. You can train a machine learning model using the Chembl 35 data and then use this model to predict the activity of the compounds in your database.
5. **Data Quality Assessment:** You want to assess the quality of the activity data in Chembl 35. You can analyze the distribution of standard values, identify outliers, and check for inconsistencies in the data.

Vietnamese Translation

1. Phân Tích Mô Hình Phân Tích

- **Mục tiêu:** Dự án nhằm phân tích dữ liệu Chembl 35 cho nghiên cứu và phát triển thuốc. Trọng tâm có thể là trích xuất thông tin liên quan về các hợp chất hóa học và hoạt tính sinh học của chúng, sau đó sử dụng dữ liệu này để xây dựng các mô hình dự đoán hoặc xác định các ứng cử viên thuốc đầy triển vọng.
- **Nguồn dữ liệu:** Chembl 35 (được truy cập qua PostgreSQL)
- **Công cụ:**
    - **PostgreSQL (psql):** Được sử dụng để truy vấn và trích xuất dữ liệu từ cơ sở dữ liệu Chembl 35. Chúng ta sẽ sử dụng SQL để lọc dữ liệu thành các hợp chất và phép đo hoạt động có liên quan.
    - **RDKit:** Một bộ công cụ tin học hóa học mạnh mẽ để thao tác và phân tích cấu trúc hóa học (chuỗi SMILES). Nó có thể được sử dụng để tạo các mô tả phân tử, tính toán các thuộc tính và thực hiện các tìm kiếm tương tự.
    - **Jupyter Notebook:** Một môi trường tương tác để viết và thực thi mã Python, tạo trực quan hóa và ghi lại phân tích của bạn.
    - **scikit-learn:** Một thư viện Machine Learning mạnh mẽ cung cấp các mô hình triển khai khác nhau, chẳng hạn như LinearRegression, DecisionTree, RandomForest…
- **Các bước chính:**
    1. **Trích xuất dữ liệu:** Sử dụng SQL để trích xuất thông tin hợp chất (ví dụ: chuỗi SMILES, cấu trúc phân tử) và dữ liệu hoạt tính sinh học (ví dụ: giá trị IC50, Ki) từ Chembl 35. SQL sẽ cần xử lý lỗi `numeric ~ unknown`.
    2. **Tiền xử lý dữ liệu:** Làm sạch và chuyển đổi dữ liệu đã trích xuất. Điều này có thể bao gồm:
        - Chuyển đổi chuỗi SMILES thành đối tượng Mol RDKit.
        - Xử lý dữ liệu bị thiếu hoặc không hợp lệ.
        - Chuyển đổi các giá trị hoạt động thành một đơn vị nhất quán (ví dụ: pIC50).
    3. **Tạo đặc trưng:** Tạo các mô tả phân tử bằng RDKit. Các mô tả này đại diện cho các thuộc tính hóa học của các phân tử.
    4. **Phân tích & Mô hình hóa (Tùy chọn):** Sử dụng các mô tả phân tử và dữ liệu hoạt động để xây dựng các mô hình dự đoán (ví dụ: mô hình hồi quy để dự đoán hoạt động).
    5. **Trực quan hóa:** Tạo các biểu đồ và đồ thị để trực quan hóa dữ liệu và kết quả phân tích của bạn.

2. Mã SQL và Python

Mã SQL (Tệp SQL: `Topic_CheMBL_35_4.sql` được lưu trong `../data/`)

```sql
-- Truy vấn SQL đã sửa để tránh lỗi "numeric ~ unknown" và giới hạn ở 100 hàng
-- và để lấy Standard Type là IC50
SELECT
    md.chembl_id,
    cs.canonical_smiles,
    act.standard_type,
    act.standard_value,
    act.standard_units
FROM
    compound_structures cs
JOIN
    molecule_dictionary md ON cs.molregno = md.molregno
JOIN
    activities act ON md.molregno = act.molregno
WHERE act.standard_type = 'IC50'
  AND act.standard_value IS NOT NULL
  AND act.standard_value::TEXT SIMILAR TO '[0-9\.]+' -- Ép kiểu về text và sử dụng
SIMILAR TO
ORDER BY md.molregno
LIMIT 100;

-- Lưu kết quả này vào data/Topic_CheMBL_35_4.csv
```

Giải thích:

- `act.standard_value::TEXT SIMILAR TO '[0-9\.]+'`: Dòng này rất quan trọng. `act.standard_value ~ '^[0-9\.]+$'` ban đầu sử dụng một toán tử biểu thức chính quy có thể không tương thích trực tiếp với các kiểu số trong PostgreSQL. Chúng ta ép kiểu `act.standard_value` thành `TEXT` và sau đó sử dụng toán tử `SIMILAR TO` để thực hiện so khớp mẫu trên các chuỗi. Điều này tránh lỗi `numeric ~ unknown`.
- `LIMIT 100`: Giới hạn tập kết quả thành 100 hàng để xử lý nhanh hơn.
- `act.standard_type = 'IC50'` : Lọc chỉ các giá trị IC50 để cụ thể hơn.
- Đầu ra của truy vấn SQL này được lưu vào thư mục data, tạo một tệp `Topic_CheMBL_35_4.csv`

Mã Python (Jupyter Notebook: Topic_CheMBL_35_4_1_Data_Extraction_and_Preprocessing.ipynb)

```python
import os
import pandas as pd
from rdkit import Chem
from rdkit.Chem import Descriptors
import numpy as np

# Xác định đường dẫn cơ sở cho dự án của bạn
base_path = "."  # Giả sử notebook nằm trong thư mục gốc

# Xây dựng đường dẫn đến tệp CSV
csv_file_path = os.path.join(base_path, "data", "Topic_CheMBL_35_4.csv")

# Tải dữ liệu từ tệp CSV
try:
    df = pd.read_csv(csv_file_path)
    print("Dữ liệu đã được tải thành công.")
except FileNotFoundError:
    print(f"Lỗi: Không tìm thấy tệp CSV tại {csv_file_path}")
    exit()
except Exception as e:
    print(f"Lỗi khi tải tệp CSV: {e}")
    exit()
```

```
# Hiển thị vài hàng đầu tiên của DataFrame
print("\n5 hàng đầu tiên của dữ liệu:")
print(df.head())

# Làm sạch và tiền xử lý dữ liệu
# Chuyển đổi standard_value thành số, xử lý lỗi
df['standard_value'] = pd.to_numeric(df['standard_value'], errors='coerce')

# Loại bỏ các hàng có standard_value bị thiếu
df = df.dropna(subset=['standard_value'])

# Lọc bỏ bất kỳ standard_value nào bằng không
df = df[df['standard_value'] != 0]

# Chuyển đổi IC50 thành pIC50 (ví dụ về chuyển đổi)
# pIC50 = -log10(IC50 tính bằng Molar)
df['pIC50'] = -np.log10(df['standard_value'] * 1e-9)  # Chuyển đổi nM sang M

# Tạo đối tượng RDKit Mol
def smiles_to_mol(smiles):
    try:
        mol = Chem.MolFromSmiles(smiles)
        if mol is not None:
            return mol
        else:
            return None
    except:
        return None

df['mol'] = df['canonical_smiles'].apply(smiles_to_mol)

# Loại bỏ các hàng mà RDKit không thể tạo đối tượng Mol
df = df.dropna(subset=['mol'])

# Tính toán mô tả phân tử (Ví dụ: Khối lượng phân tử)
def calculate_mw(mol):
    try:
        return Descriptors.MolWt(mol)
    except:
        return None

df['molecular_weight'] = df['mol'].apply(calculate_mw)

# Loại bỏ các hàng mà khối lượng phân tử không thể tạo đối tượng Mol
df = df.dropna(subset=['molecular_weight'])

print("\nDữ liệu sau khi tiền xử lý:")
print(df.head())
```

Giải thích:

- **Xử lý lỗi:** Mã bao gồm các khối `try...except` để xử lý các lỗi tiềm ẩn trong quá trình tải tệp, phân tích cú pháp SMILES và tính toán mô tả.
- **Quản lý đường dẫn:** Sử dụng `os.path.join` để xây dựng đường dẫn tệp, giúp mã dễ di chuyển hơn.
- **Làm sạch dữ liệu:** Chuyển đổi `standard_value` thành số và loại bỏ các hàng có giá trị bị thiếu. Nó cũng lọc `standard_value !=0`

- **Chuyển đổi pIC50:** Tính toán các giá trị pIC50 (một thực hành phổ biến trong khám phá thuốc).
- **Tích hợp RDKit:** Tạo các đối tượng `Mol` RDKit từ chuỗi SMILES.
- **Tính toán mô tả phân tử:** Tính toán khối lượng phân tử làm ví dụ. Bạn có thể thêm nhiều mô tả hơn nếu cần.

Mã Python (Jupyter Notebook: `Topic_CheMBL_35_4_2_Analysis_and_Visualization.ipynb`)

```
import os
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Xác định đường dẫn cơ sở cho dự án của bạn
base_path = "."  # Giả sử notebook nằm trong thư mục gốc

# Xây dựng đường dẫn đến tệp CSV
csv_file_path = os.path.join(base_path, "data", "Topic_CheMBL_35_4.csv")

# Tải dữ liệu từ tệp CSV
try:
    df = pd.read_csv(csv_file_path)
    print("Dữ liệu đã được tải thành công.")
except FileNotFoundError:
    print(f"Lỗi: Không tìm thấy tệp CSV tại {csv_file_path}")
    exit()
except Exception as e:
    print(f"Lỗi khi tải tệp CSV: {e}")
    exit()

# Làm sạch và tiền xử lý dữ liệu
# Chuyển đổi standard_value thành số, xử lý lỗi
df['standard_value'] = pd.to_numeric(df['standard_value'], errors='coerce')

# Loại bỏ các hàng có standard_value bị thiếu
df = df.dropna(subset=['standard_value'])

# Lọc bỏ bất kỳ standard_value nào bằng không
df = df[df['standard_value'] != 0]

# Chuyển đổi IC50 thành pIC50 (ví dụ về chuyển đổi)
# pIC50 = -log10(IC50 tính bằng Molar)
df['pIC50'] = -np.log10(df['standard_value'] * 1e-9)  # Chuyển đổi nM sang M

# --- Ví dụ về trực quan hóa ---

# 1. Phân phối các giá trị pIC50
plt.figure(figsize=(8, 6))
sns.histplot(df['pIC50'], kde=True)
plt.title('Phân phối các giá trị pIC50')
plt.xlabel('pIC50')
plt.ylabel('Tần số')
plt.show()

# 2. Biểu đồ phân tán của Khối lượng phân tử so với pIC50 (yêu cầu tính toán khối
lượng phân tử trong notebook trước)
try:
    plt.figure(figsize=(8, 6))
    sns.scatterplot(x='molecular_weight', y='pIC50', data=df)
```

```
    plt.title('Khối lượng phân tử so với pIC50')
    plt.xlabel('Khối lượng phân tử')
    plt.ylabel('pIC50')
    plt.show()
except KeyError:
    print("Lỗi: Không tìm thấy cột 'molecular_weight'. Đảm bảo chạy notebook đầu tiên
để tính toán nó.")

# 3. Biểu đồ hộp của các giá trị pIC50 cho các standard_types khác nhau (nếu bạn có
nhiều standard_types sau khi lọc)
if len(df['standard_type'].unique()) > 1:
    plt.figure(figsize=(8, 6))
    sns.boxplot(x='standard_type', y='pIC50', data=df)
    plt.title('Giá trị pIC50 theo Standard Type')
    plt.xlabel('Standard Type')
    plt.ylabel('pIC50')
    plt.show()
else:
    print("Chỉ tìm thấy một standard type, bỏ qua biểu đồ hộp.")

# 4. Bản đồ nhiệt tương quan của các đặc trưng số (sau khi tính toán mô tả)
try:
    numerical_cols = ['pIC50', 'molecular_weight']  # Thêm các cột số khác mà bạn tính
toán
    corr_matrix = df[numerical_cols].corr()
    plt.figure(figsize=(8, 6))
    sns.heatmap(corr_matrix, annot=True, cmap='coolwarm')
    plt.title('Bản đồ nhiệt tương quan')
    plt.show()
except KeyError:
    print("Lỗi: Không đủ cột số để tính toán tương quan. Tính toán thêm các mô tả
trong notebook đầu tiên.")

# 5. Thống kê cơ bản của pIC50
print("\nThống kê mô tả cho pIC50:")
print(df['pIC50'].describe())
```

Giải thích:

- **Tải dữ liệu đã tiền xử lý:** Giả định rằng bạn đã chạy notebook đầu tiên (`Topic_CheMBL_35_4_1_Data_Extraction_and_Preprocessing.ipynb`) và có một tệp CSV với dữ liệu đã tiền xử lý và khối lượng phân tử đã tính toán.
- **Trực quan hóa với `matplotlib` và `seaborn`:** Tạo một số biểu đồ phổ biến:
    - Biểu đồ tần suất của các giá trị pIC50
    - Biểu đồ phân tán của khối lượng phân tử so với pIC50
    - Biểu đồ hộp của các giá trị pIC50 theo standard type (nếu có)
    - Bản đồ nhiệt tương quan
- **Xử lý lỗi:** Bao gồm các khối `try...except` để xử lý các trường hợp thiếu các cột cần thiết (ví dụ: `molecular_weight` nếu bạn chưa tính toán).
- **Vẽ biểu đồ có điều kiện:** Biểu đồ hộp chỉ được tạo nếu có nhiều giá trị `standard_type` trong dữ liệu.

3. Năm Ví Dụ (Minh Họa Các Trường Hợp Sử Dụng)

Dưới đây là năm ví dụ về cách bạn có thể sử dụng mã này và dữ liệu thu được trong một dự án khám phá thuốc:

1. **Ưu tiên mục tiêu:** Bạn muốn ưu tiên một mục tiêu cho khám phá thuốc dựa trên tính khả dụng của dữ liệu hoạt động chất lượng cao. Bạn trích xuất các giá trị IC50 cho các hợp chất được thử nghiệm chống lại mục tiêu và phân tích phân phối của các giá trị pIC50. Một mục tiêu có nhiều chất ức chế mạnh (giá trị pIC50 cao) có thể là một điểm khởi đầu tốt.

2. **Xác định hợp chất dẫn đầu:** Bạn đang tìm kiếm các hợp chất dẫn đầu với các thuộc tính cụ thể. Bạn có thể lọc dữ liệu dựa trên các giá trị pIC50 (ví dụ: tìm các hợp chất có pIC50 > 7), khối lượng phân tử (ví dụ: MW < 500) và các mô tả đã tính toán khác để xác định các hợp chất dẫn đầu đầy triển vọng.

3. **Phân tích mối quan hệ cấu trúc-hoạt tính (SAR):** Bạn muốn hiểu cách thay đổi cấu trúc hóa học ảnh hưởng đến hoạt động. Bạn có thể sử dụng RDKit để tính toán một loạt các mô tả phân tử và sau đó tương quan các mô tả này với các giá trị pIC50. Điều này có thể giúp bạn xác định các đặc điểm cấu trúc chính góp phần vào hoạt động.

4. **Sàng lọc ảo:** Bạn có một cơ sở dữ liệu các hợp chất và muốn dự đoán hoạt động của chúng chống lại một mục tiêu. Bạn có thể huấn luyện một mô hình học máy bằng dữ liệu Chembl 35 và sau đó sử dụng mô hình này để dự đoán hoạt động của các hợp chất trong cơ sở dữ liệu của bạn.

5. **Đánh giá chất lượng dữ liệu:** Bạn muốn đánh giá chất lượng của dữ liệu hoạt động trong Chembl 35. Bạn có thể phân tích phân phối các giá trị tiêu chuẩn, xác định các giá trị ngoại lệ và kiểm tra tính nhất quán của dữ liệu.

Important Considerations:

- **Database Credentials:** Make sure to replace `"ip: 192.168.206.136"`, `"user:rd"`, `"pass: rd"`, `"database: chembl_35"` with your actual database credentials. For security, avoid hardcoding these directly into your notebooks. Use environment variables or a configuration file instead.
- **Descriptor Selection:** The molecular weight is just one example. RDKit provides a vast array of descriptors. Choose descriptors that are relevant to the target and the compounds you are studying.
- **Model Building:** If you plan to build predictive models, you'll need to:
    - Split your data into training and testing sets.
    - Select appropriate machine learning algorithms.
    - Tune the model parameters to optimize performance.
    - Evaluate the model's performance on the test set.
- **Data Filtering:** The `LIMIT 100` clause is for demonstration. You'll likely need to work with a larger dataset once your code is working correctly.
- **AIMLops Integration:** Consider how you will integrate this code into your AIMLops pipeline for automated data extraction, model training, and deployment.

This comprehensive guide should get you started with your Chembl 35 analysis project. Let me know if you have any more questions!