PharmApp Suite

🧠 AI for Drug Discovery and Development 🍼

🧩 Topic: Extracting kinase-binding molecules

Okay, I understand. You're working with ChEMBL 35 data, using RDKit for analysis within a standard AIMLops project structure. You have a PostgreSQL database on a specific server, and you're running your analysis in Jupyter notebooks. You're facing some errors and want help with code implementation and analysis models, focusing on retrieving a manageable 100-row subset of the data.

Here's a breakdown of how we'll approach this, along with code examples and explanations, in both English and Vietnamese.

1. Analysis Model and Considerations

Your project aims to analyze ChEMBL 35 data for drug research and development. The specific analysis will depend on your research question. Since you're not providing a precise research question, I'll demonstrate several common and useful analytical approaches you might consider, along with the corresponding code. These are examples, and you'll need to tailor them to *your* specific needs.

Here are some possible analysis models and approaches, along with explanations:

- **Model 1: Activity Distribution Analysis:** This model examines the distribution of activity values (e.g., IC50, Ki) for a specific target or a set of targets. This can help understand the potency range of compounds acting on those targets.
    - **Analysis:** Calculate descriptive statistics (mean, median, standard deviation) and visualize the distribution of activity values using histograms or box plots.
- **Model 2: Structure-Activity Relationship (SAR) Analysis:** This model investigates the relationship between chemical structure and biological activity. You'll calculate molecular descriptors (using RDKit) and correlate them with activity values.
    - **Analysis:** Calculate molecular properties (e.g., LogP, molecular weight, number of hydrogen bond donors/acceptors) using RDKit. Use statistical methods like linear regression or machine learning models (e.g., Random Forest, Support Vector Machines) to identify descriptors that are predictive of activity.
- **Model 3: Target-Based Compound Filtering:** This model focuses on identifying compounds active against a specific target. You'll select a target from ChEMBL and filter the data to retrieve compounds with activity values below a certain threshold.
    - **Analysis:** Filter compounds based on their activity values (e.g., IC50, Ki) against a specific target. Use RDKit to visualize the structures of highly active compounds.
- **Model 4: Scaffold Analysis:** This model analyzes the common structural scaffolds present in active compounds. This helps to identify key structural features that are important for activity.
    - **Analysis:** Use RDKit to generate Bemis-Murcko scaffolds for the active compounds. Identify the most frequent scaffolds and analyze their distribution across different activity ranges.
- **Model 5: Property-Based Filtering:** This model filters compounds based on their physicochemical properties (e.g., molecular weight, LogP) to identify compounds that meet certain criteria (e.g., drug-likeness).
    - **Analysis:** Use RDKit to calculate molecular properties. Apply filters based on Lipinski's Rule of Five or other drug-likeness criteria to identify compounds that are likely to be orally bioavailable.

2. SQL Code (with Error Correction and 100-row Limit)

The error `ERROR: operator does not exist: numeric ~ unknown, LINE 12: AND act.standard_value ~ '^[0-9\.]+$'` indicates that you're trying to use a regular expression (~) on a numeric column (`act.standard_value`). You want to select only numeric values. The solution is to cast the `standard_value` column to `TEXT` before applying the regular expression. However, it is much easier and safer to use proper casting.

```
-- File: ../data/chembl_35_data.csv

SELECT
    cmp.chembl_id,
    cmp.pref_name,
    act.standard_type,
    act.standard_value,
    act.standard_units,
    act.assay_id,
    mol.molfile
FROM
    compound_structures AS cmp
JOIN
    activities AS act ON cmp.molregno = act.molregno
JOIN
    molecule_dictionary AS mol ON cmp.molregno = mol.molregno
WHERE act.standard_type = 'IC50'
    AND act.standard_units = 'nM'
    AND act.standard_value BETWEEN 0 AND 100000  -- Reasonable IC50 range
    AND act.standard_relation = '='
LIMIT 100;
```

Explanation:

- **`LIMIT 100`:** This ensures you only retrieve the first 100 rows, preventing your machine from being overloaded.
- **`WHERE act.standard_type = 'IC50'`:** This filters for IC50 values. Change this if you're interested in other activity types (e.g., Ki, EC50).
- **`WHERE act.standard_units = 'nM'`:** This filters for nanomolar values
- **`WHERE act.standard_relation = '='`:** Only exact values of IC50 are selected

How to Run in pgAdmin:

1. Open pgAdmin and connect to your PostgreSQL server (192.168.206.136, user 'rd', password 'rd', database 'chembl_35').
2. Open a new query window.
3. Paste the SQL code into the query window.
4. Execute the query.
5. Right-click on the query results grid and select "Copy All Rows." Alternatively, you can export the results as a CSV file directly from pgAdmin.
6. Save the data to `../data/chembl_35_data.csv`.

Vietnamese Explanation:

```
-- File: ../data/chembl_35_data.csv

SELECT
    cmp.chembl_id,
    cmp.pref_name,
    act.standard_type,
    act.standard_value,
    act.standard_units,
    act.assay_id,
```

```
    mol.molfile
FROM
    compound_structures AS cmp
JOIN
    activities AS act ON cmp.molregno = act.molregno
JOIN
    molecule_dictionary AS mol ON cmp.molregno = mol.molregno
WHERE act.standard_type = 'IC50'
    AND act.standard_units = 'nM'
    AND act.standard_value BETWEEN 0 AND 100000  -- Khoảng giá trị IC50 hợp lý
    AND act.standard_relation = '='
LIMIT 100;
```

Giải thích:

- **`LIMIT 100`:** Điều này đảm bảo bạn chỉ lấy 100 dòng đầu tiên, tránh làm quá tải máy tính của bạn.
- **`WHERE act.standard_type = 'IC50'`:** Điều này lọc các giá trị IC50. Thay đổi nếu bạn quan tâm đến các loại hoạt tính khác (ví dụ: Ki, EC50).
- **`WHERE act.standard_units = 'nM'`:** Điều này lọc các giá trị nano molar
- **`WHERE act.standard_relation = '='`:** Chỉ chọn các giá trị IC50 chính xác

Cách chạy trong pgAdmin:

1. Mở pgAdmin và kết nối đến máy chủ PostgreSQL của bạn (192.168.206.136, người dùng 'rd', mật khẩu 'rd', cơ sở dữ liệu 'chembl_35').
2. Mở một cửa sổ truy vấn mới.
3. Dán mã SQL vào cửa sổ truy vấn.
4. Thực thi truy vấn.
5. Nhấp chuột phải vào lưới kết quả truy vấn và chọn "Copy All Rows". Ngoài ra, bạn có thể xuất kết quả dưới dạng tệp CSV trực tiếp từ pgAdmin.
6. Lưu dữ liệu vào `../data/chembl_35_data.csv`.

3. Python Code (Jupyter Notebook)

Here's a Jupyter Notebook example (`Topic_CheMBL_35_23_1_Data_Loading_and_Preprocessing.ipynb`) demonstrating how to load the data, handle potential errors, and perform basic preprocessing using RDKit. I will also provide example of the models described in point 1

```
# File: notebooks/Topic_CheMBL_35_23_1_Data_Loading_and_Preprocessing.ipynb

import os
import pandas as pd
from rdkit import Chem
from rdkit.Chem import Descriptors
from rdkit.Chem import Lipinski
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
import numpy as np
import matplotlib.pyplot as plt

# Base path for the project
base_path = ".."  # Assuming the notebooks directory is one level below the project
root

# Data file path
data_file = os.path.join(base_path, "data", "chembl_35_data.csv")
```

```
try:
    # Load the CSV file into a Pandas DataFrame
    df = pd.read_csv(data_file)
    print("Data loaded successfully.")
except FileNotFoundError:
    print(f"Error: File not found at {data_file}.  Make sure the file exists and the
path is correct.")
    exit()
except Exception as e:
    print(f"Error loading data: {e}")
    exit()

# Data Cleaning and Preprocessing
print("\nData Cleaning and Preprocessing...")

# Drop rows with missing values (important for RDKit)
df = df.dropna()

# Convert standard_value to numeric
df['standard_value'] = pd.to_numeric(df['standard_value'], errors='coerce')
df = df.dropna(subset=['standard_value']) # remove rows where conversion failed

# Remove duplicates based on chembl_id
df = df.drop_duplicates(subset=['chembl_id'])

# Display the first few rows of the DataFrame
print(df.head())

# Create RDKit Mol objects
print("\nCreating RDKit Mol objects...")
df['mol'] = df['molfile'].apply(lambda x: Chem.MolFromMolBlock(x) if x else None)

# Remove rows where Mol object creation failed
df = df.dropna(subset=['mol'])

print(f"Number of molecules after preprocessing: {len(df)}")

##############################################################################################
#
#Example 1: Activity Distribution Analysis
print('\n--- Example 1: Activity Distribution Analysis ---')
plt.hist(df['standard_value'], bins=50)
plt.xlabel('IC50 (nM)')
plt.ylabel('Frequency')
plt.title('Distribution of IC50 Values')
plt.show()

print(df['standard_value'].describe())

##############################################################################################
#
#Example 2: Structure-Activity Relationship (SAR) Analysis
print('\n---Example 2: Structure-Activity Relationship (SAR) Analysis---')

def calculate_descriptors(mol):
    try:
        mw = Descriptors.MolWt(mol)
        logp = Descriptors.MolLogP(mol)
```

```
        hbd = Lipinski.NumHDonors(mol)
        hba = Lipinski.NumHAcceptors(mol)
        return mw, logp, hbd, hba
    except:
        return None, None, None, None

df[['mw', 'logp', 'hbd', 'hba']] = df['mol'].apply(lambda x:
pd.Series(calculate_descriptors(x)))
df = df.dropna()

X = df[['mw', 'logp', 'hbd', 'hba']]
y = df['standard_value']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

model = LinearRegression()
model.fit(X_train, y_train)

y_pred = model.predict(X_test)

mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print(f'Mean Squared Error: {mse}')
print(f'R-squared: {r2}')

##############################################################################################
#
#Example 3: Target-Based Compound Filtering
print('\n---Example 3: Target-Based Compound Filtering---')

target_chembl_id = 'CHEMBL204' # Example: A real target chembl_id must be provided
target_df = df[df['assay_id'].isin([target_chembl_id])] # This needs to be refined
based on assay data
print(f"Number of compounds for target {target_chembl_id}: {len(target_df)}")
if not target_df.empty:
    print(target_df[['chembl_id', 'standard_value']].head())

##############################################################################################
#
#Example 4: Scaffold Analysis
print('\n---Example 4: Scaffold Analysis---')

from rdkit.Chem import MurckoScaffold

def get_murcko_scaffold(mol):
    try:
        return MurckoScaffold.GetScaffoldForMol(mol).GetSmiles()
    except:
        return None

df['scaffold'] = df['mol'].apply(get_murcko_scaffold)
df = df.dropna(subset=['scaffold'])

scaffold_counts = df['scaffold'].value_counts().head(10)
print("Top 10 scaffolds:")
print(scaffold_counts)
```

```
################################################################################
#
#Example 5: Property-Based Filtering
print('\n---Example 5: Property-Based Filtering---')

#Lipinski's Rule of Five: MW < 500, LogP < 5, HBD <= 5, HBA <= 10
lipinski_df = df[(df['mw'] < 500) & (df['logp'] < 5) & (df['hbd'] <= 5) & (df['hba']
<= 10)]
print(f"Number of compounds passing Lipinski's Rule of Five: {len(lipinski_df)}")
print(lipinski_df[['chembl_id', 'mw', 'logp', 'hbd', 'hba']].head())
```

Explanation:

1. **Import Libraries:** Imports necessary libraries (pandas, RDKit, scikit-learn, etc.).
2. **Define Paths:** Sets the base path and the data file path using `os.path.join`. This is crucial for portability and adhering to your AIMLops structure.
3. **Load Data:** Loads the CSV data into a pandas DataFrame using `pd.read_csv()`. Includes error handling for `FileNotFoundError` and other potential exceptions during data loading.
4. **Data Cleaning:**
    - Removes rows with missing values (`df.dropna()`). This is *critical* because RDKit functions can fail if there are missing values in the molecule data.
    - Converts `standard_value` to numeric using `pd.to_numeric()` with `errors='coerce'` to handle non-numeric values gracefully. Then, rows where the conversion failed (resulting in `NaN`) are removed.
    - Removes duplicate compounds based on `chembl_id`.
5. **Create RDKit Molecules:**
    - Creates RDKit `Mol` objects from the `molfile` strings using `Chem.MolFromMolBlock()`.
    - Handles potential errors during molecule creation by setting invalid molecules to `None` and then removing those rows using `df.dropna(subset=['mol'])`. This is *essential* for robust processing.
6. **Examples:** Shows 5 examples described in point 1

Important Notes:

- **Error Handling:** The `try...except` blocks are essential for handling potential errors during file loading and molecule creation. This makes your code more robust.
- **Path Management:** Using `os.path.join` makes your code more portable and maintainable because it correctly handles path separators on different operating systems.
- **Data Cleaning:** Always clean your data before processing it with RDKit or any other cheminformatics tool. Missing values and invalid data can cause errors.
- **RDKit Versions:** The `mean_squared_error` issue is addressed by removing the `squared=False` parameter. If you *need* the squared=False functionality (for Root Mean Squared Error), you will need to update your scikit-learn version. The best practice is to update your scikit-learn version.
- **Adapt the Analysis:** The examples provided are basic. You'll need to adapt the analysis to *your* specific research question. This might involve different molecular descriptors, different machine learning models, or different filtering criteria.
- **Assay Selection:** In the "Target-Based Compound Filtering" example, the filtering by `assay_id` is a *placeholder*. You will need to understand your data and use the appropriate `assay_id` values that correspond to the target of interest. The current filtering might not be meaningful without knowing the specific assay IDs related to your target.

Vietnamese Explanation:

```
# File: notebooks/Topic_CheMBL_35_23_1_Data_Loading_and_Preprocessing.ipynb
```

```python
import os
import pandas as pd
from rdkit import Chem
from rdkit.Chem import Descriptors
from rdkit.Chem import Lipinski
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
import numpy as np
import matplotlib.pyplot as plt

# Đường dẫn cơ sở của dự án
base_path = ".."  # Giả sử thư mục notebooks nằm ở một cấp dưới thư mục gốc của dự án

# Đường dẫn đến tệp dữ liệu
data_file = os.path.join(base_path, "data", "chembl_35_data.csv")

try:
    # Tải tệp CSV vào DataFrame của Pandas
    df = pd.read_csv(data_file)
    print("Dữ liệu đã được tải thành công.")
except FileNotFoundError:
    print(f"Lỗi: Không tìm thấy tệp tại {data_file}.  Đảm bảo tệp tồn tại và đường dẫn
là chính xác.")
    exit()
except Exception as e:
    print(f"Lỗi khi tải dữ liệu: {e}")
    exit()

# Làm sạch và tiền xử lý dữ liệu
print("\nLàm sạch và tiền xử lý dữ liệu...")

# Loại bỏ các hàng có giá trị bị thiếu (quan trọng đối với RDKit)
df = df.dropna()

# Chuyển đổi standard_value thành số
df['standard_value'] = pd.to_numeric(df['standard_value'], errors='coerce')
df = df.dropna(subset=['standard_value']) # loại bỏ các hàng mà chuyển đổi không thành
công

# Loại bỏ các bản sao dựa trên chembl_id
df = df.drop_duplicates(subset=['chembl_id'])

# Hiển thị vài hàng đầu tiên của DataFrame
print(df.head())

# Tạo đối tượng Mol của RDKit
print("\nTạo đối tượng Mol của RDKit...")
df['mol'] = df['molfile'].apply(lambda x: Chem.MolFromMolBlock(x) if x else None)

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

print(f"Số lượng phân tử sau khi tiền xử lý: {len(df)}")

##############################################################################################
#
#Ví dụ 1: Phân tích phân phối hoạt tính
print('\n--- Ví dụ 1: Phân tích phân phối hoạt tính ---')
```

```
plt.hist(df['standard_value'], bins=50)
plt.xlabel('IC50 (nM)')
plt.ylabel('Tần số')
plt.title('Phân phối các giá trị IC50')
plt.show()

print(df['standard_value'].describe())

##############################################################################
#
#Ví dụ 2: Phân tích mối quan hệ cấu trúc-hoạt tính (SAR)
print('\n---Ví dụ 2: Phân tích mối quan hệ cấu trúc-hoạt tính (SAR)---')

def calculate_descriptors(mol):
    try:
        mw = Descriptors.MolWt(mol)
        logp = Descriptors.MolLogP(mol)
        hbd = Lipinski.NumHDonors(mol)
        hba = Lipinski.NumHAcceptors(mol)
        return mw, logp, hbd, hba
    except:
        return None, None, None, None

df[['mw', 'logp', 'hbd', 'hba']] = df['mol'].apply(lambda x:
pd.Series(calculate_descriptors(x)))
df = df.dropna()

X = df[['mw', 'logp', 'hbd', 'hba']]
y = df['standard_value']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

model = LinearRegression()
model.fit(X_train, y_train)

y_pred = model.predict(X_test)

mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print(f'Sai số bình phương trung bình: {mse}')
print(f'R-squared: {r2}')

##############################################################################
#
#Ví dụ 3: Lọc hợp chất dựa trên mục tiêu
print('\n---Ví dụ 3: Lọc hợp chất dựa trên mục tiêu---')

target_chembl_id = 'CHEMBL204' # Ví dụ: Cần cung cấp một chembl_id mục tiêu thực tế
target_df = df[df['assay_id'].isin([target_chembl_id])] # Điều này cần được tinh chỉnh
dựa trên dữ liệu thử nghiệm
print(f"Số lượng hợp chất cho mục tiêu {target_chembl_id}: {len(target_df)}")
if not target_df.empty:
    print(target_df[['chembl_id', 'standard_value']].head())

##############################################################################
#
#Ví dụ 4: Phân tích giàn giáo
```

```
print('\n---Ví dụ 4: Phân tích giàn giáo---')

from rdkit.Chem import MurckoScaffold

def get_murcko_scaffold(mol):
    try:
        return MurckoScaffold.GetScaffoldForMol(mol).GetSmiles()
    except:
        return None

df['scaffold'] = df['mol'].apply(get_murcko_scaffold)
df = df.dropna(subset=['scaffold'])

scaffold_counts = df['scaffold'].value_counts().head(10)
print("Top 10 giàn giáo:")
print(scaffold_counts)

#############################################################################################
#
#Ví dụ 5: Lọc dựa trên thuộc tính
print('\n---Ví dụ 5: Lọc dựa trên thuộc tính---')

#Quy tắc năm của Lipinski: MW < 500, LogP < 5, HBD <= 5, HBA <= 10
lipinski_df = df[(df['mw'] < 500) & (df['logp'] < 5) & (df['hbd'] <= 5) & (df['hba']
<= 10)]
print(f"Số lượng hợp chất vượt qua Quy tắc năm của Lipinski: {len(lipinski_df)}")
print(lipinski_df[['chembl_id', 'mw', 'logp', 'hbd', 'hba']].head())
```

Giải thích:

1. **Nhập thư viện:** Nhập các thư viện cần thiết (pandas, RDKit, scikit-learn, v.v.).
2. **Xác định đường dẫn:** Đặt đường dẫn cơ sở và đường dẫn tệp dữ liệu bằng `os.path.join`. Điều này rất quan trọng để có thể di chuyển và tuân thủ cấu trúc AIMLops của bạn.
3. **Tải dữ liệu:** Tải dữ liệu CSV vào DataFrame của pandas bằng `pd.read_csv()`. Bao gồm xử lý lỗi cho `FileNotFoundError` và các ngoại lệ tiềm ẩn khác trong quá trình tải dữ liệu.
4. **Làm sạch dữ liệu:**
    - Loại bỏ các hàng có giá trị bị thiếu (`df.dropna()`). Điều này *rất quan trọng* vì các hàm RDKit có thể không thành công nếu có các giá trị bị thiếu trong dữ liệu phân tử.
    - Chuyển đổi `standard_value` thành số bằng cách sử dụng `pd.to_numeric()` với `errors='coerce'` để xử lý các giá trị không phải là số một cách duyên dáng. Sau đó, các hàng mà chuyển đổi không thành công (dẫn đến `NaN`) sẽ bị xóa.
    - Xóa các hợp chất trùng lặp dựa trên `chembl_id`.
5. **Tạo phân tử RDKit:**
    - Tạo các đối tượng `Mol` RDKit từ các chuỗi `molfile` bằng cách sử dụng `Chem.MolFromMolBlock()`.
    - Xử lý các lỗi tiềm ẩn trong quá trình tạo phân tử bằng cách đặt các phân tử không hợp lệ thành `None` và sau đó xóa các hàng đó bằng cách sử dụng `df.dropna(subset=['mol'])`. Điều này *cần thiết* để xử lý mạnh mẽ.
6. **Ví dụ:** Hiển thị 5 ví dụ được mô tả trong điểm 1

Lưu ý quan trọng:

- **Xử lý lỗi:** Các khối `try...except` rất cần thiết để xử lý các lỗi tiềm ẩn trong quá trình tải tệp và tạo phân tử. Điều này làm cho mã của bạn mạnh mẽ hơn.
- **Quản lý đường dẫn:** Sử dụng `os.path.join` giúp mã của bạn dễ di chuyển và bảo trì hơn vì nó xử lý chính xác các dấu phân cách đường dẫn trên các hệ điều hành khác nhau.

- **Làm sạch dữ liệu:** Luôn làm sạch dữ liệu của bạn trước khi xử lý nó bằng RDKit hoặc bất kỳ công cụ tin học hóa học nào khác. Các giá trị bị thiếu và dữ liệu không hợp lệ có thể gây ra lỗi.
- **Phiên bản RDKit:** Vấn đề `mean_squared_error` được giải quyết bằng cách xóa tham số `squared=False`. Nếu bạn *cần* chức năng squared=False (cho Sai số bình phương gốc), bạn sẽ cần cập nhật phiên bản scikit-learn của mình. Cách tốt nhất là cập nhật phiên bản scikit-learn của bạn.
- **Điều chỉnh phân tích:** Các ví dụ được cung cấp là cơ bản. Bạn sẽ cần điều chỉnh phân tích cho câu hỏi nghiên cứu cụ thể *của bạn*. Điều này có thể liên quan đến các mô tả phân tử khác nhau, các mô hình học máy khác nhau hoặc các tiêu chí lọc khác nhau.
- **Chọn thử nghiệm:** Trong ví dụ "Lọc hợp chất dựa trên mục tiêu", việc lọc theo `assay_id` là một *giữ chỗ*. Bạn sẽ cần hiểu dữ liệu của mình và sử dụng các giá trị `assay_id` thích hợp tương ứng với mục tiêu quan tâm. Việc lọc hiện tại có thể không có ý nghĩa nếu không biết ID thử nghiệm cụ thể liên quan đến mục tiêu của bạn.

4. Running the Code

1. **SQL:** Run the SQL query in pgAdmin and save the results as `../data/chembl_35_data.csv`.
2. **Jupyter Notebook:**
    - Place the `.ipynb` file (`Topic_CheMBL_35_23_1_Data_Loading_and_Preprocessing.ipynb`) in the `notebooks` directory.
    - Open the notebook in Jupyter.
    - Run each cell sequentially.

Troubleshooting:

- **File Not Found:** Double-check the path to your CSV file. Ensure it's correct relative to your notebook's location.
- **RDKit Errors:** If you encounter errors related to RDKit, make sure RDKit is installed correctly (`conda install -c conda-forge rdkit`). Also, carefully examine the error messages; they often provide clues about the cause of the problem (e.g., invalid molecule format).
- **scikit-learn Errors:** As mentioned earlier, if you need `squared=False` in `mean_squared_error`, upgrade scikit-learn (`pip install -U scikit-learn`).
- **Data Issues:** If the notebook runs but produces unexpected results (e.g., empty DataFrames), carefully examine the data cleaning and filtering steps. Make sure your filtering criteria are appropriate for your data.

Remember to adapt the provided code to your specific research question and data. This comprehensive approach will help you analyze ChEMBL 35 data effectively within your AIMLops project structure.