PharmApp Suite

🧠 AI for Drug Discovery and Development 💉

🧩 Topic: Khai thác target kháng khuẩn từ ChEMBL

Tuyệt vời! Tôi rất vui được tham gia vào dự án thú vị này với vai trò Chuyên gia R&D Dược học. Phân tích dữ liệu ChEMBL bằng RDKit là một hướng đi rất tiềm năng để khám phá và phát triển các loại thuốc mới.

Dựa trên thông tin bạn cung cấp, tôi sẽ xây dựng một quy trình làm việc chi tiết, kèm theo các ví dụ code và giải thích cặn kẽ, giúp bạn vượt qua các thách thức và đạt được mục tiêu của dự án.

Tổng quan dự án:

Dự án này tập trung vào việc khai thác dữ liệu từ cơ sở dữ liệu ChEMBL 35, kết hợp với các công cụ tin sinh học như RDKit, để hỗ trợ quá trình nghiên cứu và phát triển thuốc. Chúng ta sẽ sử dụng SQL để truy vấn dữ liệu từ PostgreSQL và Python (với RDKit) để xử lý và phân tích dữ liệu.

Cấu trúc thư mục (theo chuẩn AIMLOps Template):

```
project_root/
├── data/
│   └── *.csv  # Dữ liệu xuất ra từ SQL
├── notebooks/
│   └── Topic_CheMBL_35_37_1_*.ipynb
│   └── Topic_CheMBL_35_37_2_*.ipynb
├── src/
│   └── ... (Nếu có code Python module)
└── ...
```

1. Phân tích mô hình phân tích (Analysis Model):

Chúng ta có thể áp dụng nhiều mô hình phân tích khác nhau, tùy thuộc vào câu hỏi nghiên cứu cụ thể. Dưới đây là một số gợi ý:

- **Phân tích tương quan cấu trúc-hoạt tính (SAR/QSAR):** Xây dựng mô hình dự đoán hoạt tính sinh học của các hợp chất dựa trên cấu trúc hóa học của chúng.
- **Phân tích cụm (Clustering):** Nhóm các hợp chất có đặc điểm tương đồng (ví dụ: cấu trúc, hoạt tính) để xác định các "scaffold" tiềm năng.
- **Phân tích đa dạng hóa:** Chọn một tập hợp con các hợp chất đại diện cho sự đa dạng cấu trúc của toàn bộ tập dữ liệu.
- **Phân tích điểm nóng (Hotspot analysis):** Xác định các vùng quan trọng trên protein đích mà các phân tử thuốc nên tương tác để đạt được hoạt tính cao.
- **Phân tích liên kết cấu trúc-hoạt tính 3D (3D-QSAR):** Mở rộng SAR/QSAR bằng cách xem xét cấu trúc 3D của các phân tử và protein.
- **Học máy (Machine Learning):** Sử dụng các thuật toán học máy để dự đoán hoạt tính, độ tan, độc tính, hoặc các thuộc tính quan trọng khác của thuốc.

2. Hướng dẫn song ngữ (Bilingual Instructions):

- **English:** We will use SQL to extract data from the ChEMBL database, and Python (with RDKit) to process and analyze the data. We will focus on addressing the errors you encountered and providing clear, concise code examples.

- **Tiếng Việt:** Chúng ta sẽ sử dụng SQL để trích xuất dữ liệu từ cơ sở dữ liệu ChEMBL và Python (với RDKit) để xử lý và phân tích dữ liệu. Chúng ta sẽ tập trung vào việc giải quyết các lỗi bạn gặp phải và cung cấp các ví dụ code rõ ràng, dễ hiểu.

3. Code SQL và Python (SQL and Python Code):

3.1. Sửa lỗi SQL (Fixing SQL Error):

Lỗi `ERROR: operator does not exist: numeric ~ unknown` xảy ra do bạn đang cố gắng sử dụng toán tử `~` (regular expression match) trên một cột kiểu số (numeric). Để khắc phục, bạn có thể ép kiểu cột `standard_value` sang kiểu text trước khi so sánh:

```sql
-- English
SELECT *
FROM activities act
WHERE act.standard_type = 'IC50'
  AND act.standard_relation = '='
  AND act.standard_value IS NOT NULL
  AND act.standard_value::TEXT ~ '^[0-9\.]+$' -- Cast to TEXT
LIMIT 100;

-- Vietnamese
-- Chọn tất cả các cột từ bảng activities
-- với điều kiện standard_type là 'IC50', standard_relation là '=',
-- standard_value không null và là một số (chứa chữ số và dấu chấm)
-- Giới hạn kết quả trả về 100 dòng
SELECT *
FROM activities act
WHERE act.standard_type = 'IC50'
  AND act.standard_relation = '='
  AND act.standard_value IS NOT NULL
  AND act.standard_value::TEXT ~ '^[0-9\.]+$' -- Ép kiểu sang TEXT
LIMIT 100;
```

Lưu ý: Bạn nên kiểm tra kỹ kiểu dữ liệu của cột `standard_value` để đảm bảo rằng nó thực sự là kiểu số. Nếu không, có thể cần phải xử lý dữ liệu trước khi sử dụng nó.

3.2. Code Python (Python Code):

```python
# English
import os
import pandas as pd
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, r2_score
import numpy as np

# Base path
base_path = "."  # Adjust if needed

# Load data (assuming you saved the SQL output to a CSV file)
data_file = os.path.join(base_path, "data", "chembl_activities.csv")  # Replace with
your actual file name
try:
    df = pd.read_csv(data_file)
except FileNotFoundError:
    print(f"Error: File not found at {data_file}.  Make sure you have run the SQL
query and saved the results to this file.")
    exit()
```

```
# Function to calculate molecular descriptors using RDKit
def calculate_descriptors(smiles):
    mol = Chem.MolFromSmiles(smiles)
    if mol is None:
        return None
    descriptors = {}
    for name, func in Descriptors.descList:
        try:
            descriptors[name] = func(mol)
        except:
            descriptors[name] = np.nan # Handle errors during descriptor calculation
    return pd.Series(descriptors)


# Apply descriptor calculation
# Assuming you have a column named 'canonical_smiles' in your dataframe
if 'canonical_smiles' not in df.columns:
    print("Error: 'canonical_smiles' column not found in the dataframe.  Please check
your data.")
    exit()

df_descriptors = df['canonical_smiles'].apply(calculate_descriptors)


# Merge descriptors with the original dataframe
df = pd.concat([df, df_descriptors], axis=1)

# Data cleaning: Handle missing values (NaN)
df = df.dropna()  # Remove rows with any NaN values. Consider more sophisticated
imputation methods if needed.

# Prepare data for modeling
X = df.iloc[:, 17:]  # Select descriptor columns (adjust column indices as needed)
y = df['standard_value'] # Target variable


# 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)

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

# Make predictions
y_pred = model.predict(X_test)

# Evaluate the model
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}")

# Vietnamese
# Import các thư viện cần thiết
# Đường dẫn cơ sở
```

```
# Tải dữ liệu (giả sử bạn đã lưu kết quả SQL vào một file CSV)
# Hàm tính toán các descriptor phân tử bằng RDKit
# Áp dụng tính toán descriptor
# Gộp các descriptor vào dataframe gốc
# Làm sạch dữ liệu: Xử lý các giá trị thiếu (NaN)
# Chuẩn bị dữ liệu cho mô hình hóa
# Chia dữ liệu thành tập huấn luyện và tập kiểm tra
# Huấn luyện mô hình hồi quy tuyến tính
# Dự đoán
# Đánh giá mô hình
# In ra các kết quả đánh giá
```

Giải thích code Python:

1. **Import thư viện:** Nhập các thư viện cần thiết (RDKit, pandas, scikit-learn).
2. **Đường dẫn cơ sở:** Xác định đường dẫn cơ sở của dự án.
3. **Tải dữ liệu:** Đọc dữ liệu từ file CSV đã xuất ra từ SQL. **Quan trọng:** Thay đổi `chembl_activities.csv` thành tên file thực tế của bạn.
4. **Hàm tính toán descriptor:** Định nghĩa một hàm để tính toán các descriptor phân tử từ SMILES bằng RDKit. Hàm này xử lý cả trường hợp SMILES không hợp lệ bằng cách trả về `None`.
5. **Áp dụng tính toán descriptor:** Áp dụng hàm tính toán descriptor cho cột SMILES trong dataframe. **Quan trọng:** Giả sử rằng dataframe của bạn có một cột tên là `canonical_smiles`. Nếu không, hãy thay đổi tên cột cho phù hợp.
6. **Gộp descriptor:** Gộp các descriptor mới tính được vào dataframe gốc.
7. **Làm sạch dữ liệu:** Loại bỏ các dòng có giá trị NaN (missing values). Điều này rất quan trọng vì nhiều thuật toán học máy không xử lý được NaN. Bạn có thể sử dụng các phương pháp phức tạp hơn để xử lý missing values nếu cần.
8. **Chuẩn bị dữ liệu:** Chọn các cột descriptor làm biến độc lập (X) và cột `standard_value` làm biến phụ thuộc (y). **Quan trọng:** Bạn cần điều chỉnh các chỉ số cột (column indices) cho phù hợp với dataframe của bạn.
9. **Chia dữ liệu:** Chia dữ liệu thành tập huấn luyện và tập kiểm tra.
10. **Huấn luyện mô hình:** Khởi tạo và huấn luyện một mô hình hồi quy tuyến tính.
11. **Dự đoán:** Dự đoán giá trị trên tập kiểm tra.
12. **Đánh giá mô hình:** Tính toán các độ đo đánh giá mô hình (ví dụ: MSE, R-squared).

3.3. Sửa lỗi scikit-learn (Fixing scikit-learn error):

Nếu bạn đang sử dụng phiên bản scikit-learn cũ, bạn có thể gặp lỗi liên quan đến tham số `squared=False` trong hàm `mean_squared_error`. Trong các phiên bản cũ, tham số này không tồn tại. Để khắc phục, bạn có thể tính căn bậc hai của MSE theo cách thủ công:

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

# Vietnamese
# Tính toán sai số bình phương trung bình
# Tính căn bậc hai của MSE để có RMSE
# In ra RMSE
mse = mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse)  # Tính RMSE thủ công
print(f"Căn bậc hai của sai số bình phương trung bình: {rmse}")
```

Hoặc, bạn nên nâng cấp scikit-learn lên phiên bản mới nhất để sử dụng tham số `squared=False` một cách trực tiếp:

```
pip install -U scikit-learn
```

4. Ví dụ code SQL và Python mẫu (Example SQL and Python Code):

Dưới đây là 5 ví dụ code SQL và Python mẫu, minh họa các thao tác khác nhau mà bạn có thể thực hiện với dữ liệu ChEMBL và RDKit.

Ví dụ 1: Lọc các hợp chất có hoạt tính IC50 dưới 100 nM (SQL & Python)

- **SQL:**

```sql
-- English
SELECT molregno, canonical_smiles, standard_value
FROM activities act
JOIN molecule_dictionary md ON act.molregno = md.molregno
WHERE act.standard_type = 'IC50'
  AND act.standard_relation = '='
  AND act.standard_value <= 100
  AND act.standard_units = 'nM'
LIMIT 100;

-- Vietnamese
-- Lựa chọn molregno, canonical_smiles và standard_value
-- từ bảng activities và molecule_dictionary
-- với điều kiện standard_type là 'IC50', standard_relation là '=',
-- standard_value nhỏ hơn hoặc bằng 100 và đơn vị là 'nM'
-- Giới hạn kết quả trả về 100 dòng
SELECT molregno, canonical_smiles, standard_value
FROM activities act
JOIN molecule_dictionary md ON act.molregno = md.molregno
WHERE act.standard_type = 'IC50'
  AND act.standard_relation = '='
  AND act.standard_value <= 100
  AND act.standard_units = 'nM'
LIMIT 100;
```

- **Python:** (sau khi đã load dữ liệu từ CSV)

```python
# English
import pandas as pd
from rdkit import Chem
from rdkit.Chem import Draw

# Assuming you have a DataFrame named 'df' with 'canonical_smiles' column
# Filter for compounds with IC50 <= 100 nM
active_compounds = df[(df['standard_type'] == 'IC50') & (df['standard_relation'] ==
'=') & (df['standard_value'] <= 100) & (df['standard_units'] == 'nM')]

# Print the first 5 SMILES strings
print(active_compounds['canonical_smiles'].head())

# Visualize the first 3 molecules
smiles_list = active_compounds['canonical_smiles'].head(3).tolist()
mols = [Chem.MolFromSmiles(smiles) for smiles in smiles_list]
img = Draw.MolsToGridImage(mols, molsPerRow=3, subImgSize=(200, 200))
img  # Display the image (in Jupyter Notebook)


# Vietnamese
# Lọc các hợp chất có IC50 nhỏ hơn hoặc bằng 100 nM
# In ra 5 chuỗi SMILES đầu tiên
# Trực quan hóa 3 phân tử đầu tiên
import pandas as pd
from rdkit import Chem
```

```python
from rdkit.Chem import Draw

# Giả sử bạn có một DataFrame tên là 'df' với cột 'canonical_smiles'
# Lọc các hợp chất có IC50 <= 100 nM
active_compounds = df[(df['standard_type'] == 'IC50') & (df['standard_relation'] ==
'=') & (df['standard_value'] <= 100) & (df['standard_units'] == 'nM')]

# In ra 5 chuỗi SMILES đầu tiên
print(active_compounds['canonical_smiles'].head())

# Trực quan hóa 3 phân tử đầu tiên
smiles_list = active_compounds['canonical_smiles'].head(3).tolist()
mols = [Chem.MolFromSmiles(smiles) for smiles in smiles_list]
img = Draw.MolsToGridImage(mols, molsPerRow=3, subImgSize=(200, 200))
img  # Hiển thị hình ảnh (trong Jupyter Notebook)
```

Ví dụ 2: Tính toán trọng lượng phân tử (Molecular Weight) cho các hợp chất (Python)

```python
# English
from rdkit import Chem
from rdkit.Chem import Descriptors
import pandas as pd

# Assuming you have a DataFrame named 'df' with 'canonical_smiles' column
def calculate_mw(smiles):
    mol = Chem.MolFromSmiles(smiles)
    if mol:
        return Descriptors.MolWt(mol)
    else:
        return None

df['molecular_weight'] = df['canonical_smiles'].apply(calculate_mw)
print(df[['canonical_smiles', 'molecular_weight']].head())


# Vietnamese
# Tính toán trọng lượng phân tử
# Giả sử bạn có DataFrame 'df' với cột 'canonical_smiles'
from rdkit import Chem
from rdkit.Chem import Descriptors
import pandas as pd

# Giả sử bạn có một DataFrame tên là 'df' với cột 'canonical_smiles'
def calculate_mw(smiles):
    mol = Chem.MolFromSmiles(smiles)
    if mol:
        return Descriptors.MolWt(mol)
    else:
        return None

df['molecular_weight'] = df['canonical_smiles'].apply(calculate_mw)
print(df[['canonical_smiles', 'molecular_weight']].head())
```

Ví dụ 3: Tính toán LogP và vẽ histogram (Python)

```python
# English
from rdkit import Chem
from rdkit.Chem import Crippen
import matplotlib.pyplot as plt
import pandas as pd
```

```python
# Assuming you have a DataFrame named 'df' with 'canonical_smiles' column
def calculate_logp(smiles):
    mol = Chem.MolFromSmiles(smiles)
    if mol:
        return Crippen.MolLogP(mol)
    else:
        return None

df['logp'] = df['canonical_smiles'].apply(calculate_logp)

# Plot a histogram of LogP values
plt.hist(df['logp'].dropna(), bins=20)  # Drop NaN values before plotting
plt.xlabel('LogP')
plt.ylabel('Frequency')
plt.title('Distribution of LogP Values')
plt.show()


# Vietnamese
# Tính toán LogP và vẽ biểu đồ
# Giả sử bạn có DataFrame 'df' với cột 'canonical_smiles'
from rdkit import Chem
from rdkit.Chem import Crippen
import matplotlib.pyplot as plt
import pandas as pd

# Giả sử bạn có một DataFrame tên là 'df' với cột 'canonical_smiles'
def calculate_logp(smiles):
    mol = Chem.MolFromSmiles(smiles)
    if mol:
        return Crippen.MolLogP(mol)
    else:
        return None

df['logp'] = df['canonical_smiles'].apply(calculate_logp)

# Vẽ biểu đồ histogram của các giá trị LogP
plt.hist(df['logp'].dropna(), bins=20)  # Loại bỏ các giá trị NaN trước khi vẽ
plt.xlabel('LogP')
plt.ylabel('Tần số')
plt.title('Phân phối của các giá trị LogP')
plt.show()
```

Ví dụ 4: Tìm kiếm các hợp chất chứa một motif nhất định (SMARTS query) (Python)

```python
# English
from rdkit import Chem
import pandas as pd

# Assuming you have a DataFrame named 'df' with 'canonical_smiles' column
# Define a SMARTS pattern (e.g., benzene ring)
benzene_smarts = Chem.MolFromSmarts('c1ccccc1')

def has_benzene_ring(smiles):
    mol = Chem.MolFromSmiles(smiles)
    if mol:
        return mol.HasSubstructMatch(benzene_smarts)
    else:
        return False
```

```
df['has_benzene'] = df['canonical_smiles'].apply(has_benzene_ring)

# Filter the DataFrame to show only compounds with a benzene ring
benzene_compounds = df[df['has_benzene'] == True]
print(benzene_compounds[['canonical_smiles', 'has_benzene']].head())


# Vietnamese
# Tìm kiếm các hợp chất chứa vòng benzen
# Giả sử bạn có DataFrame 'df' với cột 'canonical_smiles'
from rdkit import Chem
import pandas as pd

# Giả sử bạn có một DataFrame tên là 'df' với cột 'canonical_smiles'
# Định nghĩa một mẫu SMARTS (ví dụ: vòng benzen)
benzene_smarts = Chem.MolFromSmarts('c1ccccc1')

def has_benzene_ring(smiles):
    mol = Chem.MolFromSmiles(smiles)
    if mol:
        return mol.HasSubstructMatch(benzene_smarts)
    else:
        return False

df['has_benzene'] = df['canonical_smiles'].apply(has_benzene_ring)

# Lọc DataFrame để chỉ hiển thị các hợp chất có vòng benzen
benzene_compounds = df[df['has_benzene'] == True]
print(benzene_compounds[['canonical_smiles', 'has_benzene']].head())
```

Ví dụ 5: Tính TPSA (Topological Polar Surface Area) (Python)

```
# English
from rdkit import Chem
from rdkit.Chem import Descriptors3D
import pandas as pd

# Assuming you have a DataFrame named 'df' with 'canonical_smiles' column
def calculate_tpsa(smiles):
    mol = Chem.MolFromSmiles(smiles)
    if mol:
        return Descriptors3D.TPSA(mol)
    else:
        return None

df['tpsa'] = df['canonical_smiles'].apply(calculate_tpsa)
print(df[['canonical_smiles', 'tpsa']].head())


# Vietnamese
# Tính TPSA (Diện tích bề mặt cực topo)
# Giả sử bạn có DataFrame 'df' với cột 'canonical_smiles'
from rdkit import Chem
from rdkit.Chem import Descriptors3D
import pandas as pd

# Giả sử bạn có một DataFrame tên là 'df' với cột 'canonical_smiles'
def calculate_tpsa(smiles):
    mol = Chem.MolFromSmiles(smiles)
```

```
    if mol:
        return Descriptors3D.TPSA(mol)
    else:
        return None

df['tpsa'] = df['canonical_smiles'].apply(calculate_tpsa)
print(df[['canonical_smiles', 'tpsa']].head())
```

Lưu ý quan trọng:

- **Thay đổi tên file và cột:** Hãy nhớ thay đổi tên file CSV và tên cột trong code cho phù hợp với dữ liệu của bạn.
- **Xử lý lỗi:** Luôn luôn kiểm tra và xử lý các lỗi có thể xảy ra trong quá trình tính toán descriptor và phân tích dữ liệu.
- **Điều chỉnh tham số:** Điều chỉnh các tham số của các thuật toán học máy và các phương pháp phân tích khác để đạt được kết quả tốt nhất.
- **Phân tích kết quả:** Quan trọng nhất là phân tích và diễn giải kết quả của bạn một cách cẩn thận để rút ra những kết luận có ý nghĩa cho dự án nghiên cứu và phát triển thuốc của bạn.
- **Giới hạn dữ liệu:** Các truy vấn SQL đều có `LIMIT 100` để giới hạn số lượng dòng trả về. Bạn có thể bỏ `LIMIT` để lấy toàn bộ dữ liệu, nhưng hãy cẩn thận với hiệu suất.

Tôi hy vọng những hướng dẫn và ví dụ này sẽ giúp bạn bắt đầu dự án phân tích dữ liệu ChEMBL 35 một cách hiệu quả. Nếu bạn có bất kỳ câu hỏi nào, đừng ngần ngại hỏi! Chúc bạn thành công!