# PharmApp Suite

## 🧠 AI for Drug Discovery and Development 🧪

Copyright 2025 | Nghiên Cứu Thuốc | www.nghiencuuthuoc.com | Zalo: +84888999311 |

## 🧩 Topic: Phân tích tính giống thuốc bằng RDKit

Tuyệt vời! Tôi rất vui được hỗ trợ bạn trong dự án phân tích dữ liệu ChEMBL 35 sử dụng RDKit. Với vai trò là một chuyên gia R&D Dược học, tôi sẽ cung cấp cho bạn hướng dẫn chi tiết, code mẫu và giải thích để bạn có thể triển khai dự án một cách hiệu quả.

**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. Bạn sẽ sử dụng SQL để truy vấn và trích xuất dữ liệu từ ChEMBL, sau đó sử dụng Python (với RDKit và các thư viện liên quan) để phân tích và mô hình hóa dữ liệu.

**Cấu trúc thư mục dự án (tuân theo AIMLOps Template):**

├── data/  
│ └── \*.csv # Dữ liệu trích xuất từ SQL  
├── notebooks/  
│ └── Topic\_CheMBL\_35\_5\_1\_\*.ipynb  
│ └── Topic\_CheMBL\_35\_5\_2\_\*.ipynb  
├── src/  
│ └── # (Optional) Các module Python tùy chỉnh  
└── README.md

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

Chủ đề Topic\_CheMBL\_35\_5 có thể tập trung vào nhiều khía cạnh khác nhau của quá trình phát triển thuốc. Dưới đây là một số mô hình phân tích tiềm năng, cùng với hướng dẫn và code mẫu:

* **Mô hình 1: Phân tích mối tương quan giữa cấu trúc hóa học và hoạt tính sinh học (SAR/QSAR):**
  + **Mục tiêu:** Xác định các nhóm chức hoặc đặc điểm cấu trúc nào ảnh hưởng đến hoạt tính của một hợp chất đối với một mục tiêu sinh học cụ thể.
  + **Phương pháp:**
    1. **Trích xuất dữ liệu:** Lấy dữ liệu về cấu trúc hóa học (SMILES) và hoạt tính sinh học (ví dụ: IC50, Ki) của các hợp chất từ ChEMBL.
    2. **Tính toán descriptor:** Sử dụng RDKit để tính toán các descriptor phân tử (ví dụ: trọng lượng phân tử, logP, số lượng liên kết hydro, diện tích bề mặt phân cực).
    3. **Phân tích tương quan:** Sử dụng các phương pháp thống kê (ví dụ: hồi quy tuyến tính, random forest) để tìm mối tương quan giữa các descriptor và hoạt tính sinh học.
  + **Ứng dụng:** Dự đoán hoạt tính của các hợp chất mới, tối ưu hóa cấu trúc hợp chất để cải thiện hoạt tính.
* **Mô hình 2: Phân tích đa dạng hóa học (Chemical Diversity Analysis):**
  + **Mục tiêu:** Đánh giá sự đa dạng của một tập hợp các hợp chất, xác định các hợp chất đại diện cho các vùng khác nhau trong không gian hóa học.
  + **Phương pháp:**
    1. **Tính toán fingerprint:** Sử dụng RDKit để tạo fingerprint cho các hợp chất (ví dụ: Morgan fingerprint, MACCS keys).
    2. **Phân tích thành phần chính (PCA) hoặc t-SNE:** Giảm chiều dữ liệu fingerprint để trực quan hóa không gian hóa học.
    3. **Phân cụm (clustering):** Phân nhóm các hợp chất dựa trên sự tương đồng về cấu trúc.
  + **Ứng dụng:** Lựa chọn các hợp chất đại diện cho thử nghiệm sàng lọc, thiết kế thư viện hợp chất đa dạng.
* **Mô hình 3: Dự đoán tính chất hấp thụ, phân bố, chuyển hóa, thải trừ (ADMET):**
  + **Mục tiêu:** Dự đoán các tính chất dược động học của một hợp chất, giúp đánh giá khả năng trở thành thuốc tiềm năng.
  + **Phương pháp:**
    1. **Tính toán descriptor:** Sử dụng RDKit để tính toán các descriptor phân tử liên quan đến ADMET (ví dụ: logP, diện tích bề mặt phân cực).
    2. **Sử dụng các mô hình dự đoán:** Áp dụng các mô hình máy học (ví dụ: random forest, SVM) hoặc các quy tắc dựa trên cấu trúc để dự đoán các tính chất ADMET (ví dụ: khả năng hấp thụ qua đường uống, khả năng xâm nhập hàng rào máu não).
  + **Ứng dụng:** Lọc bỏ các hợp chất có tính chất ADMET không phù hợp, tối ưu hóa cấu trúc hợp chất để cải thiện tính chất ADMET.

**2. Hướng dẫn song ngữ (Bilingual Instructions):**

* **(English)**: This project focuses on analyzing ChEMBL 35 data using RDKit to support drug discovery and development. You will use SQL to query and extract data, then use Python (with RDKit) to analyze and model the data.
* **(Tiếng Việt)**: Dự án này tập trung vào phân tích dữ liệu ChEMBL 35 sử dụng RDKit để hỗ trợ nghiên cứu và phát triển thuốc. Bạn sẽ sử dụng SQL để truy vấn và trích xuất dữ liệu, sau đó sử dụng Python (với RDKit) để phân tích và mô hình hóa dữ liệu.

**3. Code SQL và Python mẫu (SQL and Python Code Examples):**

**SQL (ví dụ 1): Lấy 100 hợp chất ức chế enzyme có IC50 < 100 nM**

-- English  
SELECT  
 m.chembl\_id,  
 cs.canonical\_smiles,  
 act.standard\_value,  
 act.standard\_units  
FROM  
 activities act  
JOIN  
 molecule\_dictionary m ON act.molregno = m.molregno  
JOIN  
 compound\_structures cs ON m.molregno = cs.molregno  
JOIN  
 target\_dictionary td ON act.tid = td.tid  
WHERE  
 td.target\_type = 'SINGLE PROTEIN'  
 AND act.standard\_type = 'IC50'  
 AND act.standard\_units = 'nM'  
 AND act.standard\_value IS NOT NULL  
 AND act.standard\_value > 0 -- Ensure positive values  
 AND act.standard\_value <= 100  
LIMIT 100;  
  
--Tiếng Việt  
--Lấy 100 hợp chất ức chế enzyme có IC50 < 100 nM  
SELECT  
 m.chembl\_id,  
 cs.canonical\_smiles,  
 act.standard\_value,  
 act.standard\_units  
FROM  
 activities act  
JOIN  
 molecule\_dictionary m ON act.molregno = m.molregno  
JOIN  
 compound\_structures cs ON m.molregno = cs.molregno  
JOIN  
 target\_dictionary td ON act.tid = td.tid  
WHERE  
 td.target\_type = 'SINGLE PROTEIN'  
 AND act.standard\_type = 'IC50'  
 AND act.standard\_units = 'nM'  
 AND act.standard\_value IS NOT NULL  
 AND act.standard\_value > 0 -- Đảm bảo giá trị dương  
 AND act.standard\_value <= 100  
LIMIT 100;

**Python (ví dụ 1): Tính toán descriptor phân tử và vẽ biểu đồ**

# English  
import pandas as pd  
from rdkit import Chem  
from rdkit.Chem import Descriptors  
import matplotlib.pyplot as plt  
  
# Load data from CSV file  
base\_path = "." # Thay đổi nếu cần  
data\_path = os.path.join(base\_path, "data", "your\_data.csv") # Replace "your\_data.csv"  
df = pd.read\_csv(data\_path)  
  
# Function to calculate molecular weight  
def calculate\_mw(smiles):  
 mol = Chem.MolFromSmiles(smiles)  
 if mol:  
 return Descriptors.MolWt(mol)  
 else:  
 return None  
  
# Apply the function to the SMILES column  
df['molecular\_weight'] = df['canonical\_smiles'].apply(calculate\_mw)  
  
# Remove rows with NaN values in 'molecular\_weight'  
df = df.dropna(subset=['molecular\_weight'])  
  
# Plotting the distribution of molecular weights  
plt.figure(figsize=(10, 6))  
plt.hist(df['molecular\_weight'], bins=50, color='skyblue', edgecolor='black')  
plt.title('Distribution of Molecular Weights')  
plt.xlabel('Molecular Weight (Da)')  
plt.ylabel('Frequency')  
plt.grid(True)  
plt.show()  
  
  
#Tiếng Việt  
import pandas as pd  
from rdkit import Chem  
from rdkit.Chem import Descriptors  
import matplotlib.pyplot as plt  
  
# Tải dữ liệu từ file CSV  
base\_path = "." # Thay đổi nếu cần  
data\_path = os.path.join(base\_path, "data", "your\_data.csv") # Thay thế "your\_data.csv"  
df = pd.read\_csv(data\_path)  
  
# Hàm tính toán trọng lượng phân tử  
def calculate\_mw(smiles):  
 mol = Chem.MolFromSmiles(smiles)  
 if mol:  
 return Descriptors.MolWt(mol)  
 else:  
 return None  
  
# Áp dụng hàm cho cột SMILES  
df['molecular\_weight'] = df['canonical\_smiles'].apply(calculate\_mw)  
  
# Loại bỏ các hàng có giá trị NaN trong cột 'molecular\_weight'  
df = df.dropna(subset=['molecular\_weight'])  
  
  
# Vẽ biểu đồ phân bố trọng lượng phân tử  
plt.figure(figsize=(10, 6))  
plt.hist(df['molecular\_weight'], bins=50, color='skyblue', edgecolor='black')  
plt.title('Phân bố trọng lượng phân tử')  
plt.xlabel('Trọng lượng phân tử (Da)')  
plt.ylabel('Tần số')  
plt.grid(True)  
plt.show()

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

**SQL (ví dụ 2): Lấy số lượng hợp chất cho mỗi loại mục tiêu (target type)**

-- English  
SELECT td.target\_type, COUNT(DISTINCT m.molregno) AS num\_compounds  
FROM target\_dictionary td  
JOIN activities act ON td.tid = act.tid  
JOIN molecule\_dictionary m ON act.molregno = m.molregno  
GROUP BY td.target\_type  
ORDER BY num\_compounds DESC  
LIMIT 10;  
  
-- Tiếng Việt  
-- Lấy số lượng hợp chất cho mỗi loại mục tiêu (target type)  
SELECT td.target\_type, COUNT(DISTINCT m.molregno) AS num\_compounds  
FROM target\_dictionary td  
JOIN activities act ON td.tid = act.tid  
JOIN molecule\_dictionary m ON act.molregno = m.molregno  
GROUP BY td.target\_type  
ORDER BY num\_compounds DESC  
LIMIT 10;

**Python (ví dụ 2): Tính toán LogP và vẽ scatter plot với IC50**

# English  
import pandas as pd  
from rdkit import Chem  
from rdkit.Chem import Descriptors  
import matplotlib.pyplot as plt  
import numpy as np  
  
# Load data from CSV (replace 'your\_data.csv' with your actual file)  
base\_path = "." # Thay đổi nếu cần  
data\_path = os.path.join(base\_path, "data", "your\_data.csv") # Replace "your\_data.csv"  
df = pd.read\_csv(data\_path)  
  
# Convert IC50 to numeric, handling potential errors  
df['standard\_value'] = pd.to\_numeric(df['standard\_value'], errors='coerce')  
df = df.dropna(subset=['standard\_value'])  
  
# Function to calculate LogP  
def calculate\_logp(smiles):  
 mol = Chem.MolFromSmiles(smiles)  
 if mol:  
 return Descriptors.MolLogP(mol)  
 else:  
 return None  
  
# Apply the function to the SMILES column  
df['logp'] = df['canonical\_smiles'].apply(calculate\_logp)  
  
# Remove rows with NaN values in 'logp'  
df = df.dropna(subset=['logp'])  
  
# Ensure standard\_value is numeric and convert to pIC50  
df['pIC50'] = -np.log10(df['standard\_value'] \* 1e-9) # Convert nM to M and then to pIC50  
  
# Create the scatter plot  
plt.figure(figsize=(10, 6))  
plt.scatter(df['logp'], df['pIC50'], alpha=0.5)  
plt.xlabel('LogP')  
plt.ylabel('pIC50')  
plt.title('LogP vs pIC50')  
plt.grid(True)  
plt.show()  
  
  
# Tiếng Việt  
import pandas as pd  
from rdkit import Chem  
from rdkit.Chem import Descriptors  
import matplotlib.pyplot as plt  
import numpy as np  
  
# Tải dữ liệu từ file CSV (thay 'your\_data.csv' bằng tên file của bạn)  
base\_path = "." # Thay đổi nếu cần  
data\_path = os.path.join(base\_path, "data", "your\_data.csv") # Thay thế "your\_data.csv"  
df = pd.read\_csv(data\_path)  
  
# Chuyển đổi IC50 sang dạng số, xử lý các lỗi có thể xảy ra  
df['standard\_value'] = pd.to\_numeric(df['standard\_value'], errors='coerce')  
df = df.dropna(subset=['standard\_value'])  
  
# Hàm tính toán LogP  
def calculate\_logp(smiles):  
 mol = Chem.MolFromSmiles(smiles)  
 if mol:  
 return Descriptors.MolLogP(mol)  
 else:  
 return None  
  
# Áp dụng hàm cho cột SMILES  
df['logp'] = df['canonical\_smiles'].apply(calculate\_logp)  
  
# Loại bỏ các hàng có giá trị NaN trong cột 'logp'  
df = df.dropna(subset=['logp'])  
  
# Đảm bảo standard\_value là số và chuyển đổi sang pIC50  
df['pIC50'] = -np.log10(df['standard\_value'] \* 1e-9) # Chuyển đổi nM sang M và sau đó sang pIC50  
  
# Tạo biểu đồ phân tán  
plt.figure(figsize=(10, 6))  
plt.scatter(df['logp'], df['pIC50'], alpha=0.5)  
plt.xlabel('LogP')  
plt.ylabel('pIC50')  
plt.title('LogP so với pIC50')  
plt.grid(True)  
plt.show()

**SQL (ví dụ 3): Lấy các hợp chất có trọng lượng phân tử nằm trong khoảng 200-500 Da**

-- English  
SELECT  
 m.chembl\_id,  
 cs.canonical\_smiles  
FROM  
 molecule\_dictionary m  
JOIN  
 compound\_structures cs ON m.molregno = cs.molregno  
WHERE m.molregno IN (SELECT molregno FROM molecule\_properties WHERE mw\_freebase BETWEEN 200 AND 500)  
LIMIT 100;  
  
-- Tiếng Việt  
-- Lấy các hợp chất có trọng lượng phân tử nằm trong khoảng 200-500 Da  
SELECT  
 m.chembl\_id,  
 cs.canonical\_smiles  
FROM  
 molecule\_dictionary m  
JOIN  
 compound\_structures cs ON m.molregno = m.molregno  
WHERE m.molregno IN (SELECT molregno FROM molecule\_properties WHERE mw\_freebase BETWEEN 200 AND 500)  
LIMIT 100;

**Python (ví dụ 3): Tính toán fingerprint Morgan và phân tích PCA**

# English  
import pandas as pd  
from rdkit import Chem  
from rdkit.Chem import AllChem  
from sklearn.decomposition import PCA  
import matplotlib.pyplot as plt  
import numpy as np  
  
# Load data (replace 'your\_data.csv' with your actual file)  
base\_path = "." # Thay đổi nếu cần  
data\_path = os.path.join(base\_path, "data", "your\_data.csv") # Replace "your\_data.csv"  
df = pd.read\_csv(data\_path)  
  
# Function to generate Morgan Fingerprint  
def generate\_morgan\_fingerprint(smiles, radius=2, nBits=2048):  
 mol = Chem.MolFromSmiles(smiles)  
 if mol:  
 return AllChem.GetMorganFingerprintAsBitVect(mol, radius, nBits=nBits)  
 else:  
 return None  
  
# Apply the function to the SMILES column  
df['morgan\_fp'] = df['canonical\_smiles'].apply(generate\_morgan\_fingerprint)  
  
# Remove rows where fingerprint generation failed  
df = df.dropna(subset=['morgan\_fp'])  
  
# Convert fingerprints to a numpy array  
fingerprint\_array = np.array([list(fp) for fp in df['morgan\_fp']])  
  
# Apply PCA  
pca = PCA(n\_components=2)  
pca\_result = pca.fit\_transform(fingerprint\_array)  
  
# Create a scatter plot of the PCA results  
plt.figure(figsize=(10, 8))  
plt.scatter(pca\_result[:, 0], pca\_result[:, 1], alpha=0.5)  
plt.xlabel('Principal Component 1')  
plt.ylabel('Principal Component 2')  
plt.title('PCA of Morgan Fingerprints')  
plt.grid(True)  
plt.show()  
  
# Tiếng Việt  
import pandas as pd  
from rdkit import Chem  
from rdkit.Chem import AllChem  
from sklearn.decomposition import PCA  
import matplotlib.pyplot as plt  
import numpy as np  
  
# Tải dữ liệu (thay 'your\_data.csv' bằng tên file của bạn)  
base\_path = "." # Thay đổi nếu cần  
data\_path = os.path.join(base\_path, "data", "your\_data.csv") # Thay thế "your\_data.csv"  
df = pd.read\_csv(data\_path)  
  
# Hàm tạo Morgan Fingerprint  
def generate\_morgan\_fingerprint(smiles, radius=2, nBits=2048):  
 mol = Chem.MolFromSmiles(smiles)  
 if mol:  
 return AllChem.GetMorganFingerprintAsBitVect(mol, radius, nBits=nBits)  
 else:  
 return None  
  
# Áp dụng hàm cho cột SMILES  
df['morgan\_fp'] = df['canonical\_smiles'].apply(generate\_morgan\_fingerprint)  
  
# Loại bỏ các hàng mà quá trình tạo fingerprint bị lỗi  
df = df.dropna(subset=['morgan\_fp'])  
  
# Chuyển đổi fingerprint thành một mảng numpy  
fingerprint\_array = np.array([list(fp) for fp in df['morgan\_fp']])  
  
# Áp dụng PCA  
pca = PCA(n\_components=2)  
pca\_result = pca.fit\_transform(fingerprint\_array)  
  
# Tạo biểu đồ phân tán của kết quả PCA  
plt.figure(figsize=(10, 8))  
plt.scatter(pca\_result[:, 0], pca\_result[:, 1], alpha=0.5)  
plt.xlabel('Thành phần chính 1')  
plt.ylabel('Thành phần chính 2')  
plt.title('PCA của Morgan Fingerprints')  
plt.grid(True)  
plt.show()

**SQL (ví dụ 4): Tìm các hợp chất tương tự với một hợp chất cụ thể (dựa trên SubstructureKeys)**

-- English  
SELECT m.chembl\_id, cs.canonical\_smiles  
FROM molecule\_dictionary m  
JOIN compound\_structures cs ON m.molregno = cs.molregno  
WHERE m.molregno IN (  
 SELECT molregno  
 FROM substructure\_sets  
 WHERE substructure IN (  
 SELECT substructure FROM substructure\_sets WHERE molregno = (SELECT molregno FROM molecule\_dictionary WHERE chembl\_id = 'CHEMBL121')  
 )  
 AND molregno != (SELECT molregno FROM molecule\_dictionary WHERE chembl\_id = 'CHEMBL121')  
)  
LIMIT 100;  
  
-- Tiếng Việt  
-- Tìm các hợp chất tương tự với một hợp chất cụ thể (dựa trên SubstructureKeys)  
SELECT m.chembl\_id, cs.canonical\_smiles  
FROM molecule\_dictionary m  
JOIN compound\_structures cs ON m.molregno = cs.molregno  
WHERE m.molregno IN (  
 SELECT molregno  
 FROM substructure\_sets  
 WHERE substructure IN (  
 SELECT substructure FROM substructure\_sets WHERE molregno = (SELECT molregno FROM molecule\_dictionary WHERE chembl\_id = 'CHEMBL121')  
 )  
 AND molregno != (SELECT molregno FROM molecule\_dictionary WHERE chembl\_id = 'CHEMBL121')  
)  
LIMIT 100;

**Python (ví dụ 4): Tính Tanimoto similarity giữa các fingerprint**

# English  
import pandas as pd  
from rdkit import Chem  
from rdkit.Chem import AllChem  
from rdkit import DataStructs  
  
# Load data (replace 'your\_data.csv' with your actual file)  
base\_path = "." # Thay đổi nếu cần  
data\_path = os.path.join(base\_path, "data", "your\_data.csv") # Replace "your\_data.csv"  
df = pd.read\_csv(data\_path)  
  
# Compound to compare against (replace with a valid CHEMBL\_ID from your data)  
reference\_smiles = df['canonical\_smiles'].iloc[0] # lấy smiles đầu tiên làm chuẩn, bạn có thể thay đổi  
reference\_mol = Chem.MolFromSmiles(reference\_smiles)  
reference\_fp = AllChem.GetMorganFingerprintAsBitVect(reference\_mol, 2, nBits=2048)  
  
# Function to calculate Tanimoto similarity  
def calculate\_tanimoto\_similarity(smiles, reference\_fp):  
 mol = Chem.MolFromSmiles(smiles)  
 if mol:  
 fp = AllChem.GetMorganFingerprintAsBitVect(mol, 2, nBits=2048)  
 return DataStructs.TanimotoSimilarity(reference\_fp, fp)  
 else:  
 return None  
  
# Apply the function to the SMILES column  
df['tanimoto\_similarity'] = df['canonical\_smiles'].apply(lambda x: calculate\_tanimoto\_similarity(x, reference\_fp))  
  
# Show the results  
print(df[['chembl\_id', 'tanimoto\_similarity']].head())  
  
# Tiếng Việt  
import pandas as pd  
from rdkit import Chem  
from rdkit.Chem import AllChem  
from rdkit import DataStructs  
  
# Tải dữ liệu (thay 'your\_data.csv' bằng tên file của bạn)  
base\_path = "." # Thay đổi nếu cần  
data\_path = os.path.join(base\_path, "data", "your\_data.csv") # Thay thế "your\_data.csv"  
df = pd.read\_csv(data\_path)  
  
# Hợp chất để so sánh (thay bằng một CHEMBL\_ID hợp lệ từ dữ liệu của bạn)  
reference\_smiles = df['canonical\_smiles'].iloc[0] # lấy smiles đầu tiên làm chuẩn, bạn có thể thay đổi  
reference\_mol = Chem.MolFromSmiles(reference\_smiles)  
reference\_fp = AllChem.GetMorganFingerprintAsBitVect(reference\_mol, 2, nBits=2048)  
  
# Hàm tính độ tương đồng Tanimoto  
def calculate\_tanimoto\_similarity(smiles, reference\_fp):  
 mol = Chem.MolFromSmiles(smiles)  
 if mol:  
 fp = AllChem.GetMorganFingerprintAsBitVect(mol, 2, nBits=2048)  
 return DataStructs.TanimotoSimilarity(reference\_fp, fp)  
 else:  
 return None  
  
# Áp dụng hàm cho cột SMILES  
df['tanimoto\_similarity'] = df['canonical\_smiles'].apply(lambda x: calculate\_tanimoto\_similarity(x, reference\_fp))  
  
# Hiển thị kết quả  
print(df[['chembl\_id', 'tanimoto\_similarity']].head())

**SQL (ví dụ 5): Lấy thông tin về các mục tiêu (targets) liên quan đến một bệnh cụ thể (ví dụ: ung thư)**

-- English  
SELECT td.chembl\_id, td.pref\_name, td.target\_type  
FROM target\_dictionary td  
JOIN target\_components tc ON td.tid = tc.tid  
JOIN component\_sequences cs ON tc.component\_id = cs.component\_id  
WHERE cs.description LIKE '%cancer%'  
LIMIT 100;  
  
-- Tiếng Việt  
-- Lấy thông tin về các mục tiêu (targets) liên quan đến một bệnh cụ thể (ví dụ: ung thư)  
SELECT td.chembl\_id, td.pref\_name, td.target\_type  
FROM target\_dictionary td  
JOIN target\_components tc ON td.tid = tc.tid  
JOIN component\_sequences cs ON tc.component\_id = cs.component\_id  
WHERE cs.description LIKE '%cancer%'  
LIMIT 100;

**Python (ví dụ 5): Sử dụng RDKit để chuẩn hóa SMILES**

# English  
import pandas as pd  
from rdkit import Chem  
from rdkit.Chem import MolStandardize  
  
# Load data (replace 'your\_data.csv' with your actual file)  
base\_path = "." # Thay đổi nếu cần  
data\_path = os.path.join(base\_path, "data", "your\_data.csv") # Replace "your\_data.csv"  
df = pd.read\_csv(data\_path)  
  
def standardize\_smiles(smiles):  
 try:  
 mol = Chem.MolFromSmiles(smiles)  
 if mol is not None:  
 # Initialize the standardization object  
 std = MolStandardize.Standardizer()  
 mol = std.charge\_parent(mol) # Get the parent molecule  
 mol = std.normalize(mol) # Apply normalization  
 mol = std.reionize(mol) # Reionize the molecule  
 return Chem.MolToSmiles(mol) # Convert back to SMILES  
 else:  
 return None  
 except Exception as e:  
 print(f"Error processing SMILES: {smiles} - {e}")  
 return None  
  
# Apply the standardization function  
df['standardized\_smiles'] = df['canonical\_smiles'].apply(standardize\_smiles)  
  
# Display the standardized SMILES  
print(df[['canonical\_smiles', 'standardized\_smiles']].head())  
  
  
# Tiếng Việt  
import pandas as pd  
from rdkit import Chem  
from rdkit.Chem import MolStandardize  
  
# Tải dữ liệu (thay 'your\_data.csv' bằng tên file của bạn)  
base\_path = "." # Thay đổi nếu cần  
data\_path = os.path.join(base\_path, "data", "your\_data.csv") # Thay thế "your\_data.csv"  
df = pd.read\_csv(data\_path)  
  
def standardize\_smiles(smiles):  
 try:  
 mol = Chem.MolFromSmiles(smiles)  
 if mol is not None:  
 # Khởi tạo đối tượng chuẩn hóa  
 std = MolStandardize.Standardizer()  
 mol = std.charge\_parent(mol) # Lấy phân tử gốc  
 mol = std.normalize(mol) # Áp dụng chuẩn hóa  
 mol = std.reionize(mol) # Tái ion hóa phân tử  
 return Chem.MolToSmiles(mol) # Chuyển đổi lại thành SMILES  
 else:  
 return None  
 except Exception as e:  
 print(f"Lỗi xử lý SMILES: {smiles} - {e}")  
 return None  
  
# Áp dụng hàm chuẩn hóa  
df['standardized\_smiles'] = df['canonical\_smiles'].apply(standardize\_smiles)  
  
# Hiển thị SMILES đã được chuẩn hóa  
print(df[['canonical\_smiles', 'standardized\_smiles']].head())

**5. Giải quyết các lỗi thường gặp (Addressing Common Errors):**

* **Lỗi SQL: ERROR: operator does not exist: numeric ~ unknown**
  + **(English)**: This error occurs because you’re trying to use a regular expression operator (~) on a numeric column. To fix this, ensure that the standard\_value column contains only numeric data *before* applying the regular expression. You can use TRY\_CAST to safely convert the column to numeric.
  + **(Tiếng Việt)**: Lỗi này xảy ra vì bạn đang cố gắng sử dụng toán tử biểu thức chính quy (~) trên một cột số. Để khắc phục, hãy đảm bảo rằng cột standard\_value chỉ chứa dữ liệu số *trước khi* áp dụng biểu thức chính quy. Bạn có thể sử dụng TRY\_CAST để chuyển đổi an toàn cột này sang kiểu số.
* -- English  
  SELECT  
   m.chembl\_id,  
   cs.canonical\_smiles,  
   act.standard\_value,  
   act.standard\_units  
  FROM  
   activities act  
  JOIN  
   molecule\_dictionary m ON act.molregno = m.molregno  
  JOIN  
   compound\_structures cs ON m.molregno = cs.molregno  
  WHERE  
   act.standard\_type = 'IC50'  
   AND act.standard\_units = 'nM'  
   AND act.standard\_value IS NOT NULL  
   -- Try to cast to numeric \*before\* filtering by regex  
   AND TRY\_CAST(act.standard\_value AS NUMERIC) IS NOT NULL  
   -- Now that we \*know\* it's numeric, we can compare it as such  
   AND TRY\_CAST(act.standard\_value AS NUMERIC) > 0  
   AND TRY\_CAST(act.standard\_value AS NUMERIC) <= 100  
  LIMIT 100;  
    
  -- Tiếng Việt  
  SELECT  
   m.chembl\_id,  
   cs.canonical\_smiles,  
   act.standard\_value,  
   act.standard\_units  
  FROM  
   activities act  
  JOIN  
   molecule\_dictionary m ON act.molregno = m.molregno  
  JOIN  
   compound\_structures cs ON m.molregno = cs.molregno  
  WHERE  
   act.standard\_type = 'IC50'  
   AND act.standard\_units = 'nM'  
   AND act.standard\_value IS NOT NULL  
   -- Thử chuyển đổi sang kiểu số \*trước khi\* lọc bằng regex  
   AND TRY\_CAST(act.standard\_value AS NUMERIC) IS NOT NULL  
   -- Bây giờ \*chúng ta biết\* nó là số, chúng ta có thể so sánh nó như vậy  
   AND TRY\_CAST(act.standard\_value AS NUMERIC) > 0  
   AND TRY\_CAST(act.standard\_value AS NUMERIC) <= 100  
  LIMIT 100;
* **Lỗi Python: TypeError: mean\_squared\_error() got an unexpected keyword argument 'squared'**
  + **(English)**: This error indicates that your version of scikit-learn is outdated. The squared=False parameter was introduced in a later version. You have two options:
    1. **Upgrade scikit-learn:** pip install scikit-learn --upgrade
    2. **Remove squared=False:** If you can’t upgrade, remove the squared=False argument and take the square root of the result to get the RMSE (Root Mean Squared Error).
  + **(Tiếng Việt)**: Lỗi này chỉ ra rằng phiên bản scikit-learn của bạn đã cũ. Tham số squared=False được giới thiệu trong một phiên bản sau này. Bạn có hai lựa chọn:
    1. **Nâng cấp scikit-learn:** pip install scikit-learn --upgrade
    2. **Xóa squared=False:** Nếu bạn không thể nâng cấp, hãy xóa đối số squared=False và lấy căn bậc hai của kết quả để có được RMSE (Root Mean Squared Error).
* # Ví dụ:  
  from sklearn.metrics import mean\_squared\_error  
  import numpy as np  
    
  y\_true = [3, -0.5, 2, 7]  
  y\_predicted = [2.5, 0.0, 2, 8]  
    
  # Nếu bạn không thể nâng cấp scikit-learn:  
  mse = mean\_squared\_error(y\_true, y\_predicted)  
  rmse = np.sqrt(mse)  
  print(f"RMSE: {rmse}")  
    
  # Nếu bạn đã nâng cấp scikit-learn:  
  rmse = mean\_squared\_error(y\_true, y\_predicted, squared=False)  
  print(f"RMSE: {rmse}")

**Lưu ý quan trọng:**

* Hãy thay thế "your\_data.csv" bằng tên file CSV thực tế của bạn.
* Điều chỉnh các tham số và truy vấn SQL cho phù hợp với mục tiêu nghiên cứu cụ thể của bạn.
* Sử dụng os.path.join(base\_path, "data", "your\_data.csv") để đảm bảo code của bạn hoạt động trên các hệ điều hành khác nhau.
* Hãy nhớ cài đặt các thư viện cần thiết: pip install pandas rdkit scikit-learn matplotlib

Tôi hy vọng hướng dẫn này sẽ giúp bạn bắt đầu dự án của mình một cách suôn sẻ. Nếu bạn có bất kỳ câu hỏi nào khác, đừng ngần ngại hỏi!