# PharmApp Suite

## 🧠 AI for Drug Discovery and Development 🧪

Copyright 2025 | RnD Pharma Plus | www.nghiencuuthuoc.com | Zalo: +84888999311 |

## 🧩 Topic: Linking ChEMBL with clinicaltrials.gov

Okay, I understand. You’re working on a ChEMBL 35 data analysis project using RDKit, following an AIMLops template. You’re connecting to a PostgreSQL database using psql, saving data to CSV files, and running your analysis in Jupyter Notebooks. You’ve encountered a couple of specific errors and want to filter your queries to limit the result set to 100 rows. Let’s break down the process, addressing the errors and providing the requested code.

Here’s the plan, with analysis and code examples in both English and Vietnamese:

**1. Analysis of the Analysis Model & Errors**

**English:**

* **Objective:** Your goal is to analyze ChEMBL 35 data, likely focusing on structure-activity relationships (SAR) or other aspects relevant to drug discovery. The analysis model itself is implied rather than explicitly stated. I’ll assume you’re looking at some property or relationship within the data. Based on the error messages and task description, likely IC50 or similar activity data related to specific targets in Chembl.
* **Error Analysis:**
  + **Error a: ERROR: operator does not exist: numeric ~ unknown, LINE 12: AND act.standard\_value ~ '^[0-9\.]+$'** This error arises because you’re trying to use a regular expression operator (~) on a numeric column (act.standard\_value). The PostgreSQL regular expression operator (~) is designed for text columns, not numeric types. You’re likely trying to filter for values that are purely numeric. The best way to fix it is using WHERE act.standard\_value::text SIMILAR TO '[0-9.]+', meaning you will cast numeric value to text value before matching.
  + **Error b: old scikit-learn version does not support parameters squared=False in the mean\_squared\_error function** This means your scikit-learn version is outdated. You need to upgrade it to a version that supports the squared parameter in mean\_squared\_error. This parameter controls whether you want the Mean Squared Error (MSE) or the Root Mean Squared Error (RMSE). You’ll need to upgrade scikit-learn using pip: pip install -U scikit-learn. If the old scikit-learn cannot be uninstalled (perhaps because of environment incompatibility), you can try another metric that does not require squared=False, e.g. mean\_absolute\_error.
* **Data Filtering:** Filtering to 100 rows is crucial for development to avoid overwhelming your machine. The SQL query will use LIMIT 100 to achieve this.

**Vietnamese:**

* **Mục tiêu:** Mục tiêu của bạn là phân tích dữ liệu ChEMBL 35, có thể tập trung vào mối quan hệ cấu trúc-hoạt tính (SAR) hoặc các khía cạnh khác liên quan đến việc khám phá thuốc. Mô hình phân tích bản thân nó được ngụ ý hơn là được nêu rõ ràng. Tôi sẽ cho rằng bạn đang xem xét một số thuộc tính hoặc mối quan hệ trong dữ liệu. Dựa trên thông báo lỗi và mô tả nhiệm vụ, có khả năng là dữ liệu IC50 hoặc dữ liệu hoạt động tương tự liên quan đến các mục tiêu cụ thể trong Chembl.
* **Phân tích lỗi:**
  + **Lỗi a: ERROR: operator does not exist: numeric ~ unknown, LINE 12: AND act.standard\_value ~ '^[0-9\.]+$'** Lỗi này xảy ra vì bạn đang cố gắng sử dụng toán tử biểu thức chính quy (~) trên một cột số (act.standard\_value). Toán tử biểu thức chính quy PostgreSQL (~) được thiết kế cho các cột văn bản, không phải các kiểu số. Bạn có thể đang cố gắng lọc các giá trị thuần túy là số. Cách tốt nhất để khắc phục là sử dụng WHERE act.standard\_value::text SIMILAR TO '[0-9.]+', có nghĩa là bạn sẽ chuyển đổi giá trị số thành giá trị văn bản trước khi khớp.
  + **Lỗi b: old scikit-learn version does not support parameters squared=False in the mean\_squared\_error function** Điều này có nghĩa là phiên bản scikit-learn của bạn đã lỗi thời. Bạn cần nâng cấp nó lên phiên bản hỗ trợ tham số squared trong mean\_squared\_error. Tham số này kiểm soát xem bạn muốn Mean Squared Error (MSE) hay Root Mean Squared Error (RMSE). Bạn sẽ cần nâng cấp scikit-learn bằng pip: pip install -U scikit-learn. Nếu scikit-learn cũ không thể gỡ cài đặt (có lẽ vì không tương thích môi trường), bạn có thể thử một số liệu khác không yêu cầu squared=False, ví dụ: mean\_absolute\_error.
* **Lọc dữ liệu:** Lọc xuống 100 hàng là rất quan trọng cho quá trình phát triển để tránh làm quá tải máy tính của bạn. Truy vấn SQL sẽ sử dụng LIMIT 100 để đạt được điều này.

**2. SQL Code (Topic\_CheMBL\_35\_48\_1\_SQL.sql)**

-- SQL query to extract data for ChEMBL 35 analysis (Topic\_CheMBL\_35\_48)  
-- Extracts activity data and relevant compound information, limited to 100 rows.  
  
SELECT  
 cmp.chembl\_id,  
 act.standard\_type,  
 act.standard\_value,  
 act.standard\_units,  
 act.activity\_comment,  
 tgt.pref\_name AS target\_name,  
 doc.journal,  
 doc.year  
FROM  
 activities act  
JOIN  
 molecule\_dictionary cmp ON act.molregno = cmp.molregno  
JOIN  
 target\_dictionary tgt ON act.tid = tgt.tid  
JOIN  
 docs doc ON act.doc\_id = doc.doc\_id  
WHERE act.standard\_type = 'IC50' -- Example: Filter for IC50 values  
 AND act.standard\_value IS NOT NULL  
 AND act.standard\_value::text SIMILAR TO '[0-9.]+' -- Filter for numeric values in standard\_value  
LIMIT 100;

**Vietnamese:**

-- Truy vấn SQL để trích xuất dữ liệu cho phân tích ChEMBL 35 (Topic\_CheMBL\_35\_48)  
-- Trích xuất dữ liệu hoạt động và thông tin hợp chất liên quan, giới hạn ở 100 hàng.  
  
SELECT  
 cmp.chembl\_id,  
 act.standard\_type,  
 act.standard\_value,  
 act.standard\_units,  
 act.activity\_comment,  
 tgt.pref\_name AS ten\_muc\_tieu,  
 doc.journal,  
 doc.year  
FROM  
 activities act  
JOIN  
 molecule\_dictionary cmp ON act.molregno = cmp.molregno  
JOIN  
 target\_dictionary tgt ON act.tid = tgt.tid  
JOIN  
 docs doc ON act.doc\_id = doc.doc\_id  
WHERE act.standard\_type = 'IC50' -- Ví dụ: Lọc cho các giá trị IC50  
 AND act.standard\_value IS NOT NULL  
 AND act.standard\_value::text SIMILAR TO '[0-9.]+' -- Lọc cho các giá trị số trong standard\_value  
LIMIT 100;

**Explanation:**

* **SELECT**: Selects relevant columns from different tables.
* **FROM**: Specifies the tables involved: activities, molecule\_dictionary, target\_dictionary, and docs.
* **JOIN**: Connects the tables based on their respective foreign key relationships.
* **WHERE**:
  + act.standard\_type = 'IC50': Filters for IC50 activity data (example). You can change this to other activity types (e.g., Ki, EC50) as needed.
  + act.standard\_value IS NOT NULL: Excludes rows where the standard\_value is missing.
  + act.standard\_value::text SIMILAR TO '[0-9.]+': **Crucially, this line converts the standard\_value to text and then uses the SIMILAR TO operator with a regular expression to ensure it only contains numbers and decimal points.** This addresses the “operator does not exist” error.
* **LIMIT 100**: Limits the result set to the first 100 rows.

**How to Run the SQL:**

1. Open pgAdmin.
2. Connect to your database server (IP: 192.168.206.136, User: rd, Pass: rd, Database: chembl\_35).
3. Open a new query window.
4. Copy and paste the SQL code into the query window.
5. Execute the query.
6. Export the result set to a CSV file (e.g., Topic\_CheMBL\_35\_48\_1.csv) in your ../data/ directory.

**3. Python Code (Topic\_CheMBL\_35\_48\_2\_Analysis.ipynb)**

import pandas as pd  
import numpy as np  
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, mean\_absolute\_error, r2\_score  
import matplotlib.pyplot as plt  
  
# Define the base path  
base\_path = "../data/" # Adjust if your actual path is different  
  
# Construct the full file path to the CSV file  
csv\_file\_path = os.path.join(base\_path, "Topic\_CheMBL\_35\_48\_1.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: File not found at {csv\_file\_path}. Make sure the SQL query was run and the file saved correctly.")  
 exit()  
  
# Data cleaning and preparation  
df = df.dropna(subset=['standard\_value']) # Drop rows with missing standard\_value  
df = df[df['standard\_value'] > 0] # Remove non-positive values  
df['standard\_value'] = df['standard\_value'].astype(float) # Ensure numeric type  
  
# Display the first few rows of the DataFrame  
print(df.head())  
  
# Add dummy SMILES column (replace with actual SMILES retrieval if possible)  
# WARNING: This adds a dummy SMILES string for demonstration only.  
# In a real application, you would need to retrieve SMILES strings from the ChEMBL database  
# based on the chembl\_id.  
df['smiles'] = 'CC(=O)Oc1ccccc1C(=O)O' # Example SMILES for demonstration  
  
# RDKit Feature Calculation  
def calculate\_descriptors(smiles):  
 try:  
 mol = Chem.MolFromSmiles(smiles)  
 if mol is None:  
 return None # Handle invalid SMILES strings  
  
 descriptors = {}  
 descriptors['MW'] = Descriptors.MolWt(mol)  
 descriptors['LogP'] = Descriptors.MolLogP(mol)  
 descriptors['HBD'] = Descriptors.NumHDonors(mol)  
 descriptors['HBA'] = Descriptors.NumHAcceptors(mol)  
 return pd.Series(descriptors)  
 except Exception as e:  
 print(f"Error calculating descriptors for SMILES: {smiles} - {e}")  
 return None  
  
# Apply descriptor calculation to each molecule  
df\_descriptors = df['smiles'].apply(calculate\_descriptors)  
  
# Concatenate descriptors with the main dataframe  
df = pd.concat([df, df\_descriptors], axis=1)  
df = df.dropna() # Remove rows with failed descriptor calculation  
  
# Feature selection and data preparation for the model  
X = df[['MW', 'LogP', 'HBD', 'HBA']] # Using calculated descriptors as features  
y = np.log10(df['standard\_value']) # Using log-transformed IC50 as target  
  
# 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)  
  
# Model training  
model = LinearRegression()  
model.fit(X\_train, y\_train)  
  
# Model evaluation  
y\_pred = model.predict(X\_test)  
  
# Calculate metrics  
mse = mean\_squared\_error(y\_test, y\_pred)  
rmse = np.sqrt(mse)  
mae = mean\_absolute\_error(y\_test, y\_pred)  
r2 = r2\_score(y\_test, y\_pred)  
  
print(f"Mean Squared Error (MSE): {mse}")  
print(f"Root Mean Squared Error (RMSE): {rmse}")  
print(f"Mean Absolute Error (MAE): {mae}")  
print(f"R-squared (R2): {r2}")  
  
# Plotting predicted vs. actual values  
plt.scatter(y\_test, y\_pred)  
plt.xlabel("Actual log(IC50)")  
plt.ylabel("Predicted log(IC50)")  
plt.title("Actual vs. Predicted log(IC50)")  
plt.show()

**Vietnamese:**

import pandas as pd  
import numpy as np  
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, mean\_absolute\_error, r2\_score  
import matplotlib.pyplot as plt  
  
# Xác định đường dẫn cơ sở  
base\_path = "../data/" # Điều chỉnh nếu đường dẫn thực tế của bạn khác  
  
# Xây dựng đường dẫn đầy đủ đến tệp CSV  
csv\_file\_path = os.path.join(base\_path, "Topic\_CheMBL\_35\_48\_1.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 tại {csv\_file\_path}. Đảm bảo truy vấn SQL đã được chạy và tệp đã được lưu chính xác.")  
 exit()  
  
# Làm sạch và chuẩn bị dữ liệu  
df = df.dropna(subset=['standard\_value']) # Loại bỏ các hàng có giá trị standard\_value bị thiếu  
df = df[df['standard\_value'] > 0] # Loại bỏ các giá trị không dương  
df['standard\_value'] = df['standard\_value'].astype(float) # Đảm bảo kiểu số  
  
# Hiển thị một vài hàng đầu tiên của DataFrame  
print(df.head())  
  
# Thêm cột SMILES giả (thay thế bằng cách truy xuất SMILES thực tế nếu có thể)  
# CẢNH BÁO: Điều này chỉ thêm một chuỗi SMILES giả cho mục đích minh họa.  
# Trong một ứng dụng thực tế, bạn sẽ cần truy xuất chuỗi SMILES từ cơ sở dữ liệu ChEMBL  
# dựa trên chembl\_id.  
df['smiles'] = 'CC(=O)Oc1ccccc1C(=O)O' # Ví dụ SMILES để minh họa  
  
# Tính toán các đặc trưng RDKit  
def calculate\_descriptors(smiles):  
 try:  
 mol = Chem.MolFromSmiles(smiles)  
 if mol is None:  
 return None # Xử lý chuỗi SMILES không hợp lệ  
  
 descriptors = {}  
 descriptors['MW'] = Descriptors.MolWt(mol)  
 descriptors['LogP'] = Descriptors.MolLogP(mol)  
 descriptors['HBD'] = Descriptors.NumHDonors(mol)  
 descriptors['HBA'] = Descriptors.NumHAcceptors(mol)  
 return pd.Series(descriptors)  
 except Exception as e:  
 print(f"Lỗi khi tính toán các đặc trưng cho SMILES: {smiles} - {e}")  
 return None  
  
# Áp dụng tính toán đặc trưng cho từng phân tử  
df\_descriptors = df['smiles'].apply(calculate\_descriptors)  
  
# Nối các đặc trưng với dataframe chính  
df = pd.concat([df, df\_descriptors], axis=1)  
df = df.dropna() # Loại bỏ các hàng có tính toán đặc trưng không thành công  
  
# Chọn đặc trưng và chuẩn bị dữ liệu cho mô hình  
X = df[['MW', 'LogP', 'HBD', 'HBA']] # Sử dụng các đặc trưng đã tính toán làm đặc trưng  
y = np.log10(df['standard\_value']) # Sử dụng IC50 đã được chuyển đổi logarit làm mục tiêu  
  
# Chia dữ liệu thành các tập huấn luyện và kiểm tra  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)  
  
# Huấn luyện mô hình  
model = LinearRegression()  
model.fit(X\_train, y\_train)  
  
# Đánh giá mô hình  
y\_pred = model.predict(X\_test)  
  
# Tính toán các số liệu  
mse = mean\_squared\_error(y\_test, y\_pred)  
rmse = np.sqrt(mse)  
mae = mean\_absolute\_error(y\_test, y\_pred)  
r2 = r2\_score(y\_test, y\_pred)  
  
print(f"Mean Squared Error (MSE): {mse}")  
print(f"Root Mean Squared Error (RMSE): {rmse}")  
print(f"Mean Absolute Error (MAE): {mae}")  
print(f"R-squared (R2): {r2}")  
  
# Vẽ biểu đồ giá trị dự đoán so với giá trị thực tế  
plt.scatter(y\_test, y\_pred)  
plt.xlabel("Giá trị log(IC50) thực tế")  
plt.ylabel("Giá trị log(IC50) dự đoán")  
plt.title("Giá trị thực tế so với giá trị dự đoán log(IC50)")  
plt.show()

**Explanation:**

1. **Imports:** Imports necessary libraries (pandas, RDKit, scikit-learn).
2. **File Loading:** Loads the CSV file you created from the SQL query. Includes error handling if the file is not found. *Important:* Change the base\_path variable to match your actual file structure.
3. **Data Cleaning:** Removes rows with missing standard\_value and rows where standard\_value is not a positive number. Also converts the standard\_value column to a float data type.
4. **Dummy SMILES:** **CRITICAL:** The code includes a *placeholder* SMILES string. **You MUST replace this with code to retrieve the actual SMILES strings from the ChEMBL database based on the chembl\_id column.** This example uses CC(=O)Oc1ccccc1C(=O)O (Aspirin) for demonstration purposes only.
5. **RDKit Descriptor Calculation:** Defines a function calculate\_descriptors to calculate molecular weight (MW), LogP, hydrogen bond donors (HBD), and hydrogen bond acceptors (HBA) using RDKit. This function now handles potentially invalid SMILES strings by returning None if Chem.MolFromSmiles() fails.
6. **Descriptor Application:** Applies the descriptor calculation function to each molecule in the DataFrame.
7. **Feature and Target Selection:** Selects the calculated descriptors (MW, LogP, HBD, HBA) as features (X) and the log-transformed standard\_value as the target variable (y). Log transformation is common for IC50 values to normalize the distribution.
8. **Data Splitting:** Splits the data into training and testing sets.
9. **Model Training:** Trains a linear regression model.
10. **Model Evaluation:** Predicts activity values for the test set and calculates Mean Squared Error (MSE), Root Mean Squared Error (RMSE), Mean Absolute Error (MAE), and R-squared (R2) to evaluate the model’s performance.
11. **Plotting:** Creates a scatter plot of predicted vs. actual log(IC50) values.

**Important Notes:**

* **Replace Dummy SMILES:** The most important step is to replace the dummy SMILES assignment with a proper lookup of SMILES strings from the ChEMBL database based on the chembl\_id. This will require another database query (or using the ChEMBL API).
* **Upgrade Scikit-learn:** Make sure you upgrade scikit-learn to a version that supports the squared parameter.
* **Error Handling:** The code includes basic error handling for file loading and descriptor calculation. Consider adding more robust error handling as needed.
* **Feature Engineering:** This is a basic example. You can significantly improve the model by using more sophisticated RDKit descriptors or other relevant features.
* **Model Selection:** Linear Regression is a simple model. Experiment with other machine learning models (e.g., Random Forest, Support Vector Regression) to see if you can improve performance.

**4. Examples**

Here are 5 examples of how you can modify this code:

1. **Change the Activity Type:** Modify the WHERE clause in the SQL query to filter for a different activity type (e.g., act.standard\_type = 'Ki').
2. **Add More Descriptors:** In the Python code, add more RDKit descriptors to the calculate\_descriptors function. There are many descriptors available in rdkit.Chem.Descriptors. For example, you could add Descriptors.TPSA(mol) for Topological Polar Surface Area. Remember to add the new descriptors to the X variable as well.
3. **Change the Machine Learning Model:** Replace the LinearRegression() model with a different model from scikit-learn, such as RandomForestRegressor() or SVR(). You may need to adjust the model’s parameters for optimal performance. For example:

* from sklearn.ensemble import RandomForestRegressor  
  model = RandomForestRegressor(n\_estimators=100, random\_state=42) # Example parameters

1. **Retrieve SMILES from the ChEMBL database:** Create SQL query to get smiles value corresponding to the chembl ID values in current table, combine the value to the current table and continue to run the rest of the code

* # Create a database connection  
  engine = create\_engine('postgresql://rd:rd@192.168.206.136:5432/chembl\_35')  
    
  # Create a SQL query to fetch the SMILES for each chembl\_id  
  smiles\_query = f"""  
   SELECT  
   cmp.chembl\_id,  
   cs.canonical\_smiles  
   FROM  
   molecule\_dictionary cmp  
   JOIN  
   compound\_structures cs ON cmp.molregno = cs.molregno  
   WHERE  
   cmp.chembl\_id IN ({','.join(["'"+i+"'" for i in df['chembl\_id'].tolist()])})  
  """  
  # Execute the query and load the SMILES data into a DataFrame  
  smiles\_df = pd.read\_sql(smiles\_query, engine)  
  # Merge the SMILES data with the original DataFrame  
  df = pd.merge(df, smiles\_df, on='chembl\_id', how='left')  
  # Now use the actual smiles, assign 'Invalid' if smiles value is null  
  df['smiles'] = df['canonical\_smiles'].fillna('Invalid')

1. **Explore Different Evaluation Metrics:** Add other evaluation metrics, such as explained variance score, to evaluate the model’s performance. python from sklearn.metrics import explained\_variance\_score evs = explained\_variance\_score(y\_test, y\_pred) print(f"Explained Variance Score: {evs}")

By adapting and experimenting with these examples, you can explore the ChEMBL 35 data and build more sophisticated predictive models. Remember to focus on replacing the dummy SMILES string with a proper database lookup to get meaningful results. Good luck!