# PharmApp Suite

## 🧠 AI for Drug Discovery and Development 🧪

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

## 🧩 Topic: Comparative analysis of antihypertensive agents

Okay, I understand. You’re working on a drug R&D project using ChEMBL\_35 data, leveraging RDKit in a Jupyter Notebook environment organized according to an AI/MLOps template. You’re encountering a PostgreSQL error during SQL query execution and an incompatibility issue with your scikit-learn version in your Python code. You need guidance on how to proceed, including sample code to extract 100 rows of data while addressing these issues.

Here’s a breakdown, analysis, and guidance in both English and Vietnamese:

**1. Analysis of the Project and Issues**

* **Project Goal:** To use ChEMBL data (version 35) to inform drug research and development using RDKit.
* **Data Source:** ChEMBL\_35 database accessible via PostgreSQL.
* **Tools:** PostgreSQL, RDKit, scikit-learn (presumably for modeling), Jupyter Notebook.
* **Infrastructure:** Specific server details (IP, user, password, database name) provided.
* **Issue 1 (SQL Error):** ERROR: operator does not exist: numeric ~ unknown, LINE 12: AND act.standard\_value ~ '^[0-9\.]+$'
  + **Cause:** This error arises because you’re trying to use a regular expression operator (~) to compare a numeric column (act.standard\_value) with a text pattern ('^[0-9\.]+$'). PostgreSQL’s ~ operator is primarily designed for text matching. You’re trying to check if the standard\_value is a valid number string.
  + **Solution:** You need to cast the standard\_value to text or, preferably, use a different approach to check if the value represents a valid number. A more robust solution would involve checking if the value is not NULL and is greater than or equal to zero (assuming standard\_value should be non-negative).
* **Issue 2 (scikit-learn Error):** “old scikit-learn version does not support parameters squared=False in the mean\_squared\_error function”
  + **Cause:** Your scikit-learn version is older than the one where the squared parameter was introduced in mean\_squared\_error. The default behavior of mean\_squared\_error changed between scikit-learn versions.
  + **Solution:** There are two options:
    1. **Upgrade scikit-learn:** The easiest and recommended approach is to upgrade to a newer version of scikit-learn (e.g., using pip install --upgrade scikit-learn).
    2. **Modify the code:** If upgrading is not feasible (e.g., due to environment constraints), calculate the root mean squared error (RMSE) manually by taking the square root of the output of the mean\_squared\_error function.

**2. SQL Code (to be run on pgAdmin and saved as .csv)**

-- Topic\_CheMBL\_35\_93.sql  
  
-- Extract 100 rows of data from ChEMBL.  
-- Addresses the "operator does not exist" error by avoiding regex on numeric values.  
  
SELECT  
 act.activity\_id,  
 act.standard\_type,  
 act.standard\_relation,  
 act.standard\_value,  
 act.standard\_units,  
 mol.molfile AS molfile  
FROM  
 activities act  
JOIN  
 molecule\_dictionary mol ON act.molregno = mol.molregno  
WHERE  
 act.standard\_type = 'IC50' -- Example filter: Select only IC50 values  
 AND act.standard\_relation = '=' -- Example filter: Select only exact values  
 AND act.standard\_value IS NOT NULL -- Ensure standard\_value is not null  
 AND act.standard\_value >= 0 -- Ensure standard\_value is non-negative  
LIMIT 100;  
  
-- Save the result as a CSV file using pgAdmin's export functionality.

**Explanation:**

* We select a limited set of columns relevant to activity and molecule information. The molfile column (containing the molecule structure in MDL molfile format) is included, which can be read by RDKit.
* We use WHERE clause to filter the data based on standard\_type, standard\_relation and non-null standard\_value that is also non-negative..
* LIMIT 100 restricts the result set to 100 rows. Adjust as needed.
* **Important:** After running this query in pgAdmin, use the export feature to save the result as a CSV file (e.g., ../data/Topic\_CheMBL\_35\_93.csv).

**3. Python Code (in Jupyter Notebook: Topic\_CheMBL\_35\_93\_1\_DataImport.ipynb)**

# Topic\_CheMBL\_35\_93\_1\_DataImport.ipynb  
  
import os  
import pandas as pd  
from rdkit import Chem  
from rdkit.Chem import AllChem  
import numpy as np  
from sklearn.model\_selection import train\_test\_split  
from sklearn.linear\_model import LinearRegression  
from sklearn.metrics import mean\_squared\_error  
  
# Define base path for data directory  
base\_path = os.path.abspath(os.path.join(os.getcwd(), '..'))  
data\_dir = os.path.join(base\_path, 'data')  
file\_path = os.path.join(data\_dir, 'Topic\_CheMBL\_35\_93.csv')  
  
# Load the CSV file into a Pandas DataFrame  
try:  
 df = pd.read\_csv(file\_path)  
 print(f"Data loaded successfully from {file\_path}")  
except FileNotFoundError:  
 print(f"Error: File not found at {file\_path}")  
 exit()  
  
# Display the first few rows of the DataFrame  
print(df.head())  
  
# Function to convert molfile to RDKit Mol object  
def molfile\_to\_mol(molfile\_string):  
 try:  
 mol = Chem.MolFromMolBlock(molfile\_string)  
 if mol is None:  
 print("Warning: Could not read molecule from molfile string.")  
 return None  
 return mol  
 except Exception as e:  
 print(f"Error reading molfile: {e}")  
 return None  
  
# Apply the function to create RDKit Mol objects  
df['ROMol'] = df['molfile'].apply(molfile\_to\_mol)  
  
# Drop rows where ROMol is None  
df = df.dropna(subset=['ROMol'])  
  
# Function to calculate Morgan Fingerprints  
def calculate\_morgan\_fingerprint(mol, radius=2, nBits=2048):  
 try:  
 fingerprint = AllChem.GetMorganFingerprintAsBitVect(mol, radius, nBits=nBits)  
 return np.array(fingerprint)  
 except Exception as e:  
 print(f"Error calculating fingerprint: {e}")  
 return None  
  
# Apply the function to calculate Morgan Fingerprints  
df['Morgan\_Fingerprint'] = df['ROMol'].apply(lambda x: calculate\_morgan\_fingerprint(x))  
  
# Drop rows where Morgan\_Fingerprint is None  
df = df.dropna(subset=['Morgan\_Fingerprint'])  
  
# Convert standard\_value to numeric and handle potential errors  
df['standard\_value'] = pd.to\_numeric(df['standard\_value'], errors='coerce')  
df = df.dropna(subset=['standard\_value'])  
  
# Prepare data for modeling  
X = np.stack(df['Morgan\_Fingerprint'].values)  
y = df['standard\_value'].values  
  
# 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)  
  
# Calculate Mean Squared Error and Root Mean Squared Error  
mse = mean\_squared\_error(y\_test, y\_pred)  
rmse = np.sqrt(mse) # Calculate RMSE manually to avoid potential scikit-learn version issues  
  
print(f"Mean Squared Error: {mse}")  
print(f"Root Mean Squared Error: {rmse}")

**Explanation:**

* **Imports:** Imports necessary libraries: os, pandas, rdkit.Chem, rdkit.Chem.AllChem, numpy, sklearn.model\_selection, sklearn.linear\_model, sklearn.metrics.
* **Path Handling:** Uses os.path.join to construct the path to the CSV file, making the code more portable.
* **Data Loading:** Loads the CSV data using pd.read\_csv and handles potential FileNotFoundError.
* **Molfile Conversion:** Defines a function molfile\_to\_mol to convert the molfile string to an RDKit Mol object, with error handling.
* **Fingerprint Calculation:** Defines a function calculate\_morgan\_fingerprint to compute Morgan fingerprints using AllChem.GetMorganFingerprintAsBitVect.
* **Data Cleaning:** Handles missing data by dropping rows where the ROMol object or fingerprint is None.
* **Numeric Conversion:** Converts standard\_value to numeric, handling any non-numeric values by converting them to NaN and then dropping those rows.
* **Data Preparation:** Prepares the data for modeling by extracting fingerprints and standard\_value.
* **Train/Test Split:** Splits the data into training and testing sets using train\_test\_split.
* **Model Training:** Trains a linear regression model using LinearRegression.
* **Prediction and Evaluation:** Makes predictions on the test set and calculates the Mean Squared Error (MSE). Critically, it calculates the Root Mean Squared Error (RMSE) manually by taking the square root of the MSE. This avoids the potential squared=False error with older scikit-learn versions.

**4. Example Usage and Further Analysis (in Jupyter Notebook: Topic\_CheMBL\_35\_93\_2\_ModelBuilding.ipynb)**

# Topic\_CheMBL\_35\_93\_2\_ModelBuilding.ipynb  
  
import os  
import pandas as pd  
from rdkit import Chem  
from rdkit.Chem import AllChem  
import numpy as np  
from sklearn.model\_selection import train\_test\_split  
from sklearn.linear\_model import LinearRegression  
from sklearn.metrics import mean\_squared\_error  
from sklearn.ensemble import RandomForestRegressor  
  
# Define base path for data directory  
base\_path = os.path.abspath(os.path.join(os.getcwd(), '..'))  
data\_dir = os.path.join(base\_path, 'data')  
file\_path = os.path.join(data\_dir, 'Topic\_CheMBL\_35\_93.csv')  
  
# Load the CSV file into a Pandas DataFrame  
try:  
 df = pd.read\_csv(file\_path)  
 print(f"Data loaded successfully from {file\_path}")  
except FileNotFoundError:  
 print(f"Error: File not found at {file\_path}")  
 exit()  
  
# Function to convert molfile to RDKit Mol object  
def molfile\_to\_mol(molfile\_string):  
 try:  
 mol = Chem.MolFromMolBlock(molfile\_string)  
 if mol is None:  
 print("Warning: Could not read molecule from molfile string.")  
 return None  
 return mol  
 except Exception as e:  
 print(f"Error reading molfile: {e}")  
 return None  
  
# Apply the function to create RDKit Mol objects  
df['ROMol'] = df['molfile'].apply(molfile\_to\_mol)  
  
# Drop rows where ROMol is None  
df = df.dropna(subset=['ROMol'])  
  
# Function to calculate Morgan Fingerprints  
def calculate\_morgan\_fingerprint(mol, radius=2, nBits=2048):  
 try:  
 fingerprint = AllChem.GetMorganFingerprintAsBitVect(mol, radius, nBits=nBits)  
 return np.array(fingerprint)  
 except Exception as e:  
 print(f"Error calculating fingerprint: {e}")  
 return None  
  
# Apply the function to calculate Morgan Fingerprints  
df['Morgan\_Fingerprint'] = df['ROMol'].apply(lambda x: calculate\_morgan\_fingerprint(x))  
  
# Drop rows where Morgan\_Fingerprint is None  
df = df.dropna(subset=['Morgan\_Fingerprint'])  
  
# Convert standard\_value to numeric and handle potential errors  
df['standard\_value'] = pd.to\_numeric(df['standard\_value'], errors='coerce')  
df = df.dropna(subset=['standard\_value'])  
  
# Prepare data for modeling  
X = np.stack(df['Morgan\_Fingerprint'].values)  
y = df['standard\_value'].values  
  
# 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)  
  
# 1. Random Forest Regressor Example  
rf\_model = RandomForestRegressor(n\_estimators=100, random\_state=42) # Example parameters  
rf\_model.fit(X\_train, y\_train)  
rf\_y\_pred = rf\_model.predict(X\_test)  
rf\_mse = mean\_squared\_error(y\_test, rf\_y\_pred)  
rf\_rmse = np.sqrt(rf\_mse)  
  
print("Random Forest Results:")  
print(f" Mean Squared Error: {rf\_mse}")  
print(f" Root Mean Squared Error: {rf\_rmse}")  
  
# 2. Feature Importance (Example with Random Forest)  
if hasattr(rf\_model, 'feature\_importances\_'):  
 importances = rf\_model.feature\_importances\_  
 print("\nFeature Importances (Top 10):")  
 for i in np.argsort(importances)[-10:]:  
 print(f" Feature {i}: {importances[i]}")  
  
# 3. Log Transformation of y (if appropriate)  
# Check if y values are positive and potentially skewed  
if all(y > 0):  
 y\_log = np.log1p(y) #log(1+y)  
 X\_train, X\_test, y\_train\_log, y\_test\_log = train\_test\_split(X, y\_log, test\_size=0.2, random\_state=42)  
  
 # Train a linear regression model  
 model = LinearRegression()  
 model.fit(X\_train, y\_train\_log)  
  
 # Make predictions  
 y\_pred = model.predict(X\_test)  
 y\_pred\_exp = np.expm1(y\_pred) #Back transform  
 # Calculate Mean Squared Error and Root Mean Squared Error  
 mse = mean\_squared\_error(y\_test, y\_pred\_exp)  
 rmse = np.sqrt(mse) # Calculate RMSE manually to avoid potential scikit-learn version issues  
  
 print(f"Mean Squared Error (after log transformation): {mse}")  
 print(f"Root Mean Squared Error (after log transformation): {rmse}")  
  
# 4. Cross-Validation  
from sklearn.model\_selection import cross\_val\_score  
  
# Perform cross-validation  
scores = cross\_val\_score(rf\_model, X, y, cv=5, scoring='neg\_mean\_squared\_error')  
rmse\_scores = np.sqrt(-scores)  
  
print("Cross-validation results:")  
print(f"RMSE scores: {rmse\_scores}")  
print(f"Mean RMSE: {rmse\_scores.mean()}")  
  
# 5. Try Ridge Regression  
from sklearn.linear\_model import Ridge  
  
ridge\_model = Ridge(alpha=1.0) # You can adjust alpha (regularization strength)  
ridge\_model.fit(X\_train, y\_train)  
ridge\_y\_pred = ridge\_model.predict(X\_test)  
ridge\_mse = mean\_squared\_error(y\_test, ridge\_y\_pred)  
ridge\_rmse = np.sqrt(ridge\_mse)  
  
print("Ridge Regression Results:")  
print(f" Mean Squared Error: {ridge\_mse}")  
print(f" Root Mean Squared Error: {ridge\_rmse}")

**Explanation of Examples:**

1. **Random Forest Regressor:** Demonstrates using a RandomForestRegressor as an alternative model.
2. **Feature Importance:** Shows how to access feature importances from a trained Random Forest model (if available). This can help identify which fingerprint bits are most predictive of activity.
3. **Log Transformation:** Illustrates how to apply a log transformation to the standard\_value (if appropriate, e.g., if the values are positively skewed). Back transform the predicted value before calculate MSE. This can sometimes improve model performance if the target variable is not normally distributed.
4. **Cross-Validation:** Demonstrates using cross-validation (cross\_val\_score) to get a more robust estimate of the model’s performance. It uses 5-fold cross-validation and calculates the RMSE for each fold.
5. **Ridge Regression:** Tries another linear model, Ridge Regression, which adds regularization to prevent overfitting. The alpha parameter controls the strength of the regularization.

**Important Considerations:**

* **Data Exploration:** Before jumping into modeling, thoroughly explore your data. Look at the distribution of standard\_value, the types of molecules present, etc.
* **Feature Engineering:** Experiment with different RDKit descriptors and fingerprint types. You might find that other features are more predictive than Morgan fingerprints.
* **Model Selection:** Try a variety of machine learning models (e.g., Support Vector Machines, Gradient Boosting Machines).
* **Hyperparameter Tuning:** Optimize the hyperparameters of your chosen model using techniques like grid search or random search.
* **Validation:** Use a separate validation set to evaluate the final performance of your model. This helps ensure that your model generalizes well to unseen data.
* **ChEMBL License:** Remember to comply with the ChEMBL license agreement when using the data.

**Vietnamese Translation:**

**1. Phân tích dự án và các vấn đề:**

* **Mục tiêu dự án:** Sử dụng dữ liệu ChEMBL (phiên bản 35) để hỗ trợ nghiên cứu và phát triển thuốc bằng cách sử dụng RDKit.
* **Nguồn dữ liệu:** Cơ sở dữ liệu ChEMBL\_35 có thể truy cập thông qua PostgreSQL.
* **Công cụ:** PostgreSQL, RDKit, scikit-learn (có thể để xây dựng mô hình), Jupyter Notebook.
* **Cơ sở hạ tầng:** Thông tin chi tiết về máy chủ (IP, người dùng, mật khẩu, tên cơ sở dữ liệu) đã được cung cấp.
* **Vấn đề 1 (Lỗi SQL):** ERROR: operator does not exist: numeric ~ unknown, LINE 12: AND act.standard\_value ~ '^[0-9\.]+$'
  + **Nguyên nhân:** Lỗi này xảy ra do bạn đang cố gắng sử dụng toán tử biểu thức chính quy (~) để so sánh một cột số (act.standard\_value) với một mẫu văn bản ('^[0-9\.]+$'). Toán tử ~ của PostgreSQL chủ yếu được thiết kế để khớp văn bản. Bạn đang cố gắng kiểm tra xem standard\_value có phải là một chuỗi số hợp lệ hay không.
  + **Giải pháp:** Bạn cần chuyển đổi standard\_value thành văn bản hoặc, tốt hơn, sử dụng một phương pháp khác để kiểm tra xem giá trị có đại diện cho một số hợp lệ hay không. Một giải pháp mạnh mẽ hơn sẽ liên quan đến việc kiểm tra xem giá trị có phải là NULL hay không và có lớn hơn hoặc bằng không hay không (giả sử standard\_value phải không âm).
* **Vấn đề 2 (Lỗi scikit-learn):** “old scikit-learn version does not support parameters squared=False in the mean\_squared\_error function”
  + **Nguyên nhân:** Phiên bản scikit-learn của bạn cũ hơn phiên bản mà tham số squared được giới thiệu trong mean\_squared\_error. Hành vi mặc định của mean\_squared\_error đã thay đổi giữa các phiên bản scikit-learn.
  + **Giải pháp:** Có hai lựa chọn:
    1. **Nâng cấp scikit-learn:** Cách dễ nhất và được khuyến nghị là nâng cấp lên phiên bản scikit-learn mới hơn (ví dụ: sử dụng pip install --upgrade scikit-learn).
    2. **Sửa đổi mã:** Nếu việc nâng cấp không khả thi (ví dụ: do các ràng buộc về môi trường), hãy tính toán sai số bình phương gốc (RMSE) theo cách thủ công bằng cách lấy căn bậc hai của đầu ra của hàm mean\_squared\_error.

**2. Mã SQL (chạy trên pgAdmin và lưu thành .csv):** (Xem phần trên, đã được dịch trong phần mã)

**3. Mã Python (trong Jupyter Notebook: Topic\_CheMBL\_35\_93\_1\_DataImport.ipynb):** (Xem phần trên, đã được dịch trong phần mã)

**4. Ví dụ Sử dụng và Phân tích thêm (trong Jupyter Notebook: Topic\_CheMBL\_35\_93\_2\_ModelBuilding.ipynb):** (Xem phần trên, đã được dịch trong phần mã)

**Các lưu ý quan trọng (Tiếng Việt):**

* **Khám phá dữ liệu:** Trước khi bắt đầu xây dựng mô hình, hãy khám phá kỹ lưỡng dữ liệu của bạn. Xem xét phân phối của standard\_value, các loại phân tử hiện có, v.v.
* **Thiết kế đặc trưng (Feature Engineering):** Thử nghiệm với các mô tả và loại vân tay RDKit khác nhau. Bạn có thể thấy rằng các đặc trưng khác có khả năng dự đoán tốt hơn so với vân tay Morgan.
* **Lựa chọn mô hình:** Thử nhiều mô hình học máy khác nhau (ví dụ: Máy vectơ hỗ trợ, Máy tăng cường độ dốc).
* **Điều chỉnh siêu tham số (Hyperparameter Tuning):** Tối ưu hóa các siêu tham số của mô hình bạn đã chọn bằng cách sử dụng các kỹ thuật như tìm kiếm lưới (grid search) hoặc tìm kiếm ngẫu nhiên (random search).
* **Xác thực (Validation):** Sử dụng một tập hợp xác thực riêng biệt để đánh giá hiệu suất cuối cùng của mô hình của bạn. Điều này giúp đảm bảo rằng mô hình của bạn khái quát hóa tốt cho dữ liệu chưa thấy.
* **Giấy phép ChEMBL:** Hãy nhớ tuân thủ thỏa thuận cấp phép ChEMBL khi sử dụng dữ liệu.

This comprehensive guide should get you started. Remember to adjust the code and analysis based on the specifics of your project and the insights you gain from your data exploration. Good luck!