# PharmApp Suite

## 🧠 AI for Drug Discovery and Development 🧪

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

## 🧩 Topic: Violin plot for multi-target activity spectrum

Okay, I understand. You’re working on a Chembl 35 data analysis project using RDKit, following an AIMLops-inspired folder structure. You’ll be running SQL queries against a PostgreSQL database (Chembl\_35) to extract data, saving it as CSV, and then using Jupyter Notebooks with Python and RDKit to analyze it. You’re encountering some errors and want to focus on extracting a limited dataset (100 rows) for prototyping. You want guidance in both English and Vietnamese on how to proceed.

Let’s break this down step-by-step.

**1. Analysis of the Analysis Model**

* **Goal:** The primary goal is to extract and analyze data from the ChEMBL 35 database using RDKit. The analysis likely aims to:
  + Identify potentially interesting compounds based on specific criteria (activity, properties, etc.).
  + Explore relationships between chemical structures and biological activity.
  + Build predictive models (QSAR, QSPR) for drug discovery.
* **Data Flow:**
  1. **SQL Query:** Extract relevant data from the Chembl\_35 database. You’ll be focusing on activity data and molecular properties (likely SMILES strings for the compounds).
  2. **CSV Export:** Save the extracted data as CSV files.
  3. **Data Loading and Preprocessing:** Load the CSV data into a Pandas DataFrame in your Jupyter Notebook. Clean and prepare the data for analysis. This may involve:
     + Handling missing values.
     + Converting data types.
     + Filtering data based on specific criteria.
  4. **RDKit Integration:** Use RDKit to:
     + Convert SMILES strings to RDKit Mol objects.
     + Calculate molecular descriptors (e.g., LogP, molecular weight, topological polar surface area (TPSA)).
  5. **Analysis and Modeling:**
     + Perform exploratory data analysis (EDA) to visualize relationships between descriptors and activity.
     + Build statistical models (e.g., linear regression, random forests) to predict activity based on molecular descriptors.
* **Addressing Errors:**
  + **SQL Error (numeric ~ unknown):** The ~ operator in SQL is used for regular expression matching. The database is likely not recognizing that the standard\_value column is supposed to be treated as text when applying regular expression matching.
  + **scikit-learn Error (squared=False):** This indicates that you’re using an older version of scikit-learn. The squared=False parameter in mean\_squared\_error was introduced in a later version.

**2. SQL Code (English and Vietnamese)**

Here’s an SQL query to extract data for 100 rows, addressing the regex issue, and including common fields used in drug discovery analysis. This focuses on activity data and SMILES strings. \* **English**

SELECT  
 act.activity\_id,  
 act.standard\_type,  
 act.standard\_relation,  
 act.standard\_value,  
 act.standard\_units,  
 act.assay\_id,  
 mol.molregno,  
 mol.smiles,  
 cmp.chembl\_id  
FROM  
 activities act  
JOIN  
 molecule\_dictionary mol ON act.molregno = mol.molregno  
JOIN  
 compound\_structures cmp ON mol.molregno = cmp.molregno  
WHERE  
 act.standard\_type IN ('IC50', 'Ki', 'EC50') -- Common activity types  
 AND act.standard\_units = 'nM' -- Focus on nanomolar activity  
 AND act.standard\_value IS NOT NULL  
 AND act.standard\_relation = '='  
 AND act.standard\_value::TEXT SIMILAR TO '[0-9.]+' -- Convert to text for regex matching  
ORDER BY  
 act.standard\_value ASC  
LIMIT 100;

* **Vietnamese**

SELECT  
 act.activity\_id,  
 act.standard\_type,  
 act.standard\_relation,  
 act.standard\_value,  
 act.standard\_units,  
 act.assay\_id,  
 mol.molregno,  
 mol.smiles,  
 cmp.chembl\_id  
FROM  
 activities act  
JOIN  
 molecule\_dictionary mol ON act.molregno = mol.molregno  
JOIN  
 compound\_structures cmp ON mol.molregno = cmp.molregno  
WHERE  
 act.standard\_type IN ('IC50', 'Ki', 'EC50') -- Các loại hoạt tính phổ biến  
 AND act.standard\_units = 'nM' -- Tập trung vào hoạt tính nanomolar  
 AND act.standard\_value IS NOT NULL  
 AND act.standard\_relation = '='  
 AND act.standard\_value::TEXT SIMILAR TO '[0-9.]+' -- Chuyển đổi sang text để so khớp regex  
ORDER BY  
 act.standard\_value ASC  
LIMIT 100;

**Explanation:**

* SELECT: Specifies the columns to retrieve.
* FROM: Indicates the tables to use (activities, molecule\_dictionary, compound\_structures).
* JOIN: Connects the tables based on common columns (molregno).
* WHERE: Filters the data based on the following criteria:
  + act.standard\_type IN ('IC50', 'Ki', 'EC50'): Selects only activities with standard types IC50, Ki, or EC50.
  + act.standard\_units = 'nM': Selects only activities with units of nM.
  + act.standard\_value IS NOT NULL: Excludes rows where the standard value is missing.
  + act.standard\_relation = '=': Only include data with the = relation which signifies exact value.
  + act.standard\_value::TEXT SIMILAR TO '[0-9.]+': **Crucially, this converts the standard\_value column to TEXT and uses the SIMILAR TO operator (PostgreSQL’s equivalent of LIKE with regular expressions) to ensure that the value is a number (containing only digits and a decimal point). This addresses the numeric ~ unknown error.** If SIMILAR TO doesn’t work, try CAST(act.standard\_value AS TEXT) ~ '^[0-9\.]+$'.
* ORDER BY act.standard\_value ASC: Order the results from lowest to highest standard value.
* LIMIT 100: Restricts the result set to 100 rows.

**How to Use:**

1. **Run the SQL query:** Execute this query in pgAdmin (or your preferred PostgreSQL client) connected to your Chembl\_35 database.
2. **Export to CSV:** Save the results as a CSV file named something like chembl\_35\_activity\_data.csv in your ../data/ directory.

**3. Python Code (Jupyter Notebook) - English and Vietnamese**

# English  
import os  
import pandas as pd  
from rdkit import Chem  
from rdkit.Chem import Descriptors  
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, r2\_score  
  
# Define the base path for your project  
base\_path = ".." # Assuming your notebook is in the 'notebooks' directory  
  
# File path to your CSV data  
data\_file = os.path.join(base\_path, "data", "chembl\_35\_activity\_data.csv")  
  
# Load the data  
try:  
 df = pd.read\_csv(data\_file)  
except FileNotFoundError:  
 print(f"Error: File not found at {data\_file}")  
 exit()  
  
# --- Data Preprocessing ---  
# Handle missing SMILES (very important!)  
df = df.dropna(subset=['smiles'])  
  
# Convert standard\_value to numeric (important for calculations)  
df['standard\_value'] = pd.to\_numeric(df['standard\_value'], errors='coerce')  
df = df.dropna(subset=['standard\_value'])  
  
# --- RDKit Calculations ---  
def calculate\_descriptors(smiles):  
 mol = Chem.MolFromSmiles(smiles)  
 if mol is None:  
 return None, None # Handle invalid SMILES  
 mw = Descriptors.MolWt(mol)  
 logp = Descriptors.MolLogP(mol)  
 tpsa = Descriptors.TPSA(mol)  
 return mw, logp, tpsa  
  
# Apply the function to each SMILES string  
df[['molecular\_weight', 'logp', 'tpsa']] = df['smiles'].apply(lambda x: pd.Series(calculate\_descriptors(x)))  
  
# Drop rows with NaN values after descriptor calculation  
df = df.dropna()  
  
# --- Feature Selection and Model Building ---  
X = df[['molecular\_weight', 'logp', 'tpsa']] # Features  
y = df['standard\_value'] # Target variable (activity)  
  
# 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}")  
  
# Example of accessing a specific chembl\_id's data  
example\_chembl\_id = df['chembl\_id'].iloc[0] # Access the first chembl\_id  
example\_data = df[df['chembl\_id'] == example\_chembl\_id]  
print(f"\nExample data for ChEMBL ID: {example\_chembl\_id}")  
print(example\_data)

# Vietnamese  
import os  
import pandas as pd  
from rdkit import Chem  
from rdkit.Chem import Descriptors  
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, r2\_score  
  
# Xác định đường dẫn gốc của dự án  
base\_path = ".." # Giả sử notebook nằm trong thư mục 'notebooks'  
  
# Đường dẫn đến file CSV chứa dữ liệu  
data\_file = os.path.join(base\_path, "data", "chembl\_35\_activity\_data.csv")  
  
# Tải dữ liệu  
try:  
 df = pd.read\_csv(data\_file)  
except FileNotFoundError:  
 print(f"Lỗi: Không tìm thấy file tại {data\_file}")  
 exit()  
  
# --- Tiền xử lý dữ liệu ---  
# Xử lý các SMILES bị thiếu (rất quan trọng!)  
df = df.dropna(subset=['smiles'])  
  
# Chuyển đổi standard\_value sang kiểu số (quan trọng cho tính toán)  
df['standard\_value'] = pd.to\_numeric(df['standard\_value'], errors='coerce')  
df = df.dropna(subset=['standard\_value'])  
  
# --- Tính toán các descriptor bằng RDKit ---  
def calculate\_descriptors(smiles):  
 mol = Chem.MolFromSmiles(smiles)  
 if mol is None:  
 return None, None # Xử lý SMILES không hợp lệ  
 mw = Descriptors.MolWt(mol)  
 logp = Descriptors.MolLogP(mol)  
 tpsa = Descriptors.TPSA(mol)  
 return mw, logp, tpsa  
  
# Áp dụng hàm cho mỗi chuỗi SMILES  
df[['molecular\_weight', 'logp', 'tpsa']] = df['smiles'].apply(lambda x: pd.Series(calculate\_descriptors(x)))  
  
# Loại bỏ các hàng có giá trị NaN sau khi tính toán descriptor  
df = df.dropna()  
  
# --- Lựa chọn đặc trưng và xây dựng mô hình ---  
X = df[['molecular\_weight', 'logp', 'tpsa']] # Các đặc trưng  
y = df['standard\_value'] # Biến mục tiêu (hoạt tính)  
  
# Chia dữ liệu thành tập huấn luyện và tập 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 hồi quy tuyến tính  
model = LinearRegression()  
model.fit(X\_train, y\_train)  
  
# Dự đoán  
y\_pred = model.predict(X\_test)  
  
# Đánh giá mô hình  
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"Hệ số R bình phương: {r2}")  
  
# Ví dụ về việc truy cập dữ liệu của một chembl\_id cụ thể  
example\_chembl\_id = df['chembl\_id'].iloc[0] # Truy cập chembl\_id đầu tiên  
example\_data = df[df['chembl\_id'] == example\_chembl\_id]  
print(f"\nVí dụ dữ liệu cho ChEMBL ID: {example\_chembl\_id}")  
print(example\_data)

**Explanation:**

1. **Import Libraries:** Imports necessary libraries (Pandas, RDKit, scikit-learn).
2. **Define base\_path:** Sets the base path to your project root (two levels up, assuming the notebook is in a notebooks directory).
3. **Load Data:** Loads the CSV data into a Pandas DataFrame. Includes error handling for the case where the file is not found.
4. **Data Preprocessing:**
   * df = df.dropna(subset=['smiles']): **Critical:** Removes rows where the smiles column is empty. RDKit *will* crash if you try to process an empty SMILES string.
   * df['standard\_value'] = pd.to\_numeric(df['standard\_value'], errors='coerce'): Converts the standard\_value column to numeric. errors='coerce' will turn any values that can’t be converted to numbers into NaN.
   * df = df.dropna(subset=['standard\_value']): Remove rows with NaN in ‘standard\_value’ column.
5. **RDKit Calculations:**
   * calculate\_descriptors(smiles): A function that takes a SMILES string as input, converts it to an RDKit Mol object, and calculates molecular weight, LogP, and TPSA.
   * Error Handling: The if mol is None: check is **essential** to handle invalid SMILES strings. Invalid SMILES will cause Chem.MolFromSmiles() to return None.
   * df[['molecular\_weight', 'logp', 'tpsa']] = df['smiles'].apply(lambda x: pd.Series(calculate\_descriptors(x))): Applies the calculate\_descriptors function to each SMILES string in the DataFrame and creates new columns for the calculated descriptors. pd.Series is used to properly unpack the multiple return values into separate columns.
   * df = df.dropna(): Removes any rows that now contain NaN values (likely because of invalid SMILES strings that resulted in None values for the descriptors).
6. **Feature Selection and Model Building:**
   * X = df[['molecular\_weight', 'logp', 'tpsa']]: Selects the molecular weight, LogP, and TPSA as features. You can add more descriptors here.
   * y = df['standard\_value']: Sets the standard\_value column as the target variable (the activity you’re trying to predict).
   * X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42): Splits the data into training and testing sets (80% for training, 20% for testing). random\_state=42 ensures that the split is reproducible.
   * model = LinearRegression(): Creates a linear regression model. You can experiment with other models (e.g., RandomForestRegressor).
   * model.fit(X\_train, y\_train): Trains the model on the training data.
   * y\_pred = model.predict(X\_test): Makes predictions on the testing data.
   * mse = mean\_squared\_error(y\_test, y\_pred) and r2 = r2\_score(y\_test, y\_pred): Evaluates the model using mean squared error and R-squared. Since you may have an older scikit-learn version, you can either:
     + Upgrade scikit-learn: pip install --upgrade scikit-learn
     + Remove squared=False from the mean\_squared\_error function (the default is squared=True, which returns the MSE; removing it is fine for this example).
7. **Example Usage:**
   * Shows how to access the data associated with a specific chembl\_id.

**Important Considerations:**

* **Error Handling:** The code includes error handling for file not found and invalid SMILES strings. Robust error handling is crucial in real-world data analysis.
* **Data Cleaning:** Thoroughly cleaning your data is *essential*. Pay close attention to missing values, invalid SMILES strings, and inconsistent units.
* **Feature Engineering:** Experiment with different molecular descriptors. RDKit provides a wide range of descriptors that can be used as features in your models.
* **Model Selection:** Linear regression is a simple model. Consider more complex models (e.g., random forests, support vector machines) for better performance.
* **Hyperparameter Tuning:** Optimize the hyperparameters of your models to improve their accuracy.

**4. AIMLops Folder Structure**

Assuming an AIMLops-inspired structure, your folder structure might look like this:

project\_root/  
 data/ # Contains raw data (chembl\_35\_activity\_data.csv)  
 notebooks/ # Contains Jupyter notebooks (Topic\_CheMBL\_35\_78\_1\_eda.ipynb, Topic\_CheMBL\_35\_78\_2\_modeling.ipynb)  
 models/ # Stores trained models (optional, for saving models)  
 src/ # Contains Python modules (optional, for reusable code)  
 requirements.txt # Lists Python dependencies

**5. Examples (Based on the Code)**

Here are 5 specific examples of tasks you can perform with the code:

1. **Calculate Descriptors for a Specific SMILES:**

# English  
smiles = "CC(=O)Oc1ccccc1C(=O)O" # Aspirin  
mw, logp, tpsa = calculate\_descriptors(smiles)  
print(f"Aspirin: MW = {mw}, LogP = {logp}, TPSA = {tpsa}")  
  
# Vietnamese  
smiles = "CC(=O)Oc1ccccc1C(=O)O" # Aspirin  
mw, logp, tpsa = calculate\_descriptors(smiles)  
print(f"Aspirin: MW = {mw}, LogP = {logp}, TPSA = {tpsa}")

1. **Filter Data by Molecular Weight:**

# English  
filtered\_df = df[(df['molecular\_weight'] > 200) & (df['molecular\_weight'] < 400)]  
print(f"Number of compounds with MW between 200 and 400: {len(filtered\_df)}")  
  
# Vietnamese  
filtered\_df = df[(df['molecular\_weight'] > 200) & (df['molecular\_weight'] < 400)]  
print(f"Số lượng hợp chất có MW từ 200 đến 400: {len(filtered\_df)}")

1. **Calculate the Mean Activity (standard\_value) for a Specific Activity Type (e.g., IC50):**

# English  
ic50\_data = df[df['standard\_type'] == 'IC50']  
mean\_ic50 = ic50\_data['standard\_value'].mean()  
print(f"Mean IC50: {mean\_ic50}")  
  
# Vietnamese  
ic50\_data = df[df['standard\_type'] == 'IC50']  
mean\_ic50 = ic50\_data['standard\_value'].mean()  
print(f"IC50 trung bình: {mean\_ic50}")

1. **Find Compounds with LogP > 3.0 and TPSA < 75:**

# English  
filtered\_df = df[(df['logp'] > 3.0) & (df['tpsa'] < 75)]  
print(f"Number of compounds with LogP > 3.0 and TPSA < 75: {len(filtered\_df)}")  
  
# Vietnamese  
filtered\_df = df[(df['logp'] > 3.0) & (df['tpsa'] < 75)]  
print(f"Số lượng hợp chất có LogP > 3.0 và TPSA < 75: {len(filtered\_df)}")

1. **Save the Processed Data to a New CSV File:**

# English  
output\_file = os.path.join(base\_path, "data", "chembl\_35\_processed\_data.csv")  
df.to\_csv(output\_file, index=False)  
print(f"Processed data saved to: {output\_file}")  
  
# Vietnamese  
output\_file = os.path.join(base\_path, "data", "chembl\_35\_processed\_data.csv")  
df.to\_csv(output\_file, index=False)  
print(f"Dữ liệu đã xử lý được lưu vào: {output\_file}")

Remember to adapt these examples to your specific research questions and data analysis goals. Good luck! Let me know if you have any further questions.