# PharmApp Suite

## 🧠 AI for Drug Discovery and Development 🧪

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

## 🧩 Topic: 3D target structure availability from PDB

Okay, I understand. You’re working on a Chembl 35 data analysis project using RDKit, following an AIMLops template. You’re having some issues with SQL syntax and scikit-learn versioning, and you want to filter the data to 100 rows for faster processing. You want to create 5 different codes following ““Topic\_CheMBL\_35\_31”” Here’s a breakdown of the analysis, code examples, and explanations in both English and Vietnamese.

**Analysis Model Overview**

The general analysis model will revolve around exploring the Chembl 35 data to extract relevant information about drug-like molecules. This will likely involve the following steps:

1. **Data Extraction:** Querying the Chembl 35 database using SQL to retrieve relevant data. This will likely focus on activity data, compound structures (SMILES strings), and potentially target information. We’ll filter this data to a manageable size (100 rows).
2. **Data Preprocessing:** Cleaning and preparing the data for analysis. This includes handling missing values, converting SMILES strings to RDKit molecule objects, and potentially calculating molecular descriptors.
3. **Feature Engineering:** Generating molecular descriptors using RDKit. These descriptors represent various structural and physicochemical properties of the molecules and will be used as features for analysis.
4. **Data Analysis & Modeling (Example Dependent):** The type of analysis will depend on the specific question you’re trying to answer. This could involve:
   * **Exploratory Data Analysis (EDA):** Visualizing the distribution of molecular descriptors and activity values to identify trends and patterns.
   * **Structure-Activity Relationship (SAR) Analysis:** Identifying correlations between molecular descriptors and activity values to understand how changes in molecular structure affect activity.
   * **Machine Learning Modeling:** Building predictive models to predict activity based on molecular descriptors. This could involve regression models (e.g., linear regression, random forests) or classification models (e.g., logistic regression, support vector machines).
5. **Interpretation and Reporting:** Interpreting the results of the analysis and reporting the findings in a clear and concise manner.

**Error Analysis and Solutions**

* **Error a: ERROR: operator does not exist: numeric ~ unknown, LINE 12: AND act.standard\_value ~ '^[0-9\.]+$'**
* This error occurs because the ~ operator in PostgreSQL is used for regular expression matching, and it seems like act.standard\_value is of numeric type. You can’t directly use regular expressions on numeric columns.
* **Solution:** You need to cast the standard\_value to text before using the regular expression.
* **Error b: old scikit-learn version does not support parameters squared=False in the mean\_squared\_error function**
* This error indicates you’re using an older version of scikit-learn. The squared=False parameter was introduced in a later version.
* **Solution:** There are two options:
  1. **Upgrade scikit-learn:** The recommended solution is to upgrade your scikit-learn version to the latest version. You can do this using pip: pip install -U scikit-learn
  2. **Calculate RMSE manually:** If you cannot upgrade scikit-learn, you can calculate the Root Mean Squared Error (RMSE) manually by taking the square root of the mean squared error.

**Topic Code: Topic\_CheMBL\_35\_31**

**Topic Name: Exploration of Activity Data and Molecular Properties for a Subset of Chembl 35**

**Code Examples (5 Examples)**

I’ll provide five code examples focusing on different aspects of data extraction, processing, and analysis. All examples will be limited to 100 rows for efficiency.

**Example 1: Extracting Activity Data and SMILES (SQL and Python)**

* **SQL (extract\_data.sql):** This script extracts activity data (specifically IC50 values) and SMILES strings for a specific target.

-- extract\_data.sql  
SELECT  
 cmp.chembl\_id,  
 cmp.canonical\_smiles,  
 act.standard\_value,  
 act.standard\_units,  
 act.assay\_id  
FROM  
 compound\_structures cmp  
JOIN  
 activities act ON cmp.molregno = act.molregno  
JOIN  
 assays ass ON act.assay\_id = ass.assay\_id  
WHERE  
 ass.target\_chembl\_id = 'CHEMBL182' -- Example: EGFR target  
 AND act.standard\_type = 'IC50'  
 AND act.standard\_units = 'nM'  
 AND act.standard\_value::text ~ '^[0-9\.]+$' -- Corrected line, convert to text for regex  
LIMIT 100;

* **Python (Topic\_CheMBL\_35\_31\_1\_data\_extraction.ipynb):** This notebook reads the CSV file generated by the SQL script, displays the data and performs elementary filtering.

# Topic\_CheMBL\_35\_31\_1\_data\_extraction.ipynb  
  
import pandas as pd  
import os  
  
base\_path = "." # Assuming the notebook is in the base directory  
data\_file = os.path.join(base\_path, "data", "extracted\_data.csv") # Ensure 'extracted\_data.csv' matches your filename  
  
try:  
 df = pd.read\_csv(data\_file)  
 print(df.head())  
 print(df.info())  
except FileNotFoundError:  
 print(f"Error: File not found at {data\_file}. Make sure the SQL script has been run and the data saved to this location.")  
  
#Filter data to where standard\_value is not null  
df = df[df['standard\_value'].notna()]  
  
#Filter data to where standard\_units equals to nM  
df = df[df['standard\_units'] == 'nM']  
print(df.info())

**Explanation (English):**

The SQL script extracts data for a specific target (‘CHEMBL182’ – EGFR is used as an example, replace with your target). It joins the compound\_structures, activities, and assays tables to retrieve the CHEMBL ID, SMILES string, IC50 value, units, and assay ID. The corrected act.standard\_value::text ~ '^[0-9\.]+$' line casts the numeric standard\_value to text so we can use a regular expression to ensure it is a valid number. The LIMIT 100 clause restricts the result set to the first 100 rows. The Python notebook reads the CSV data into a Pandas DataFrame, displays the first few rows, and prints information about the data types and missing values. Elementary data filtering is done to further clean up the data.

**Explanation (Vietnamese):**

Script SQL trích xuất dữ liệu cho một mục tiêu cụ thể (‘CHEMBL182’ – EGFR được sử dụng làm ví dụ, hãy thay thế bằng mục tiêu của bạn). Nó kết hợp các bảng compound\_structures, activities và assays để truy xuất CHEMBL ID, chuỗi SMILES, giá trị IC50, đơn vị và ID xét nghiệm. Dòng act.standard\_value::text ~ '^[0-9\.]+$' đã được sửa đổi chuyển đổi standard\_value số thành văn bản để chúng ta có thể sử dụng biểu thức chính quy để đảm bảo nó là một số hợp lệ. Mệnh đề LIMIT 100 giới hạn tập kết quả chỉ còn 100 hàng đầu tiên. Notebook Python đọc dữ liệu CSV vào DataFrame Pandas, hiển thị một vài hàng đầu tiên và in thông tin về các kiểu dữ liệu và các giá trị bị thiếu. Bộ lọc dữ liệu sơ cấp được thực hiện để làm sạch thêm dữ liệu.

**Example 2: Converting SMILES to RDKit Molecules and Calculating MW (Molecular Weight)**

* **Python (Topic\_CheMBL\_35\_31\_2\_rdkit\_mol\_mw.ipynb):** This notebook converts SMILES strings to RDKit molecule objects and calculates the molecular weight for each molecule.

# Topic\_CheMBL\_35\_31\_2\_rdkit\_mol\_mw.ipynb  
  
import pandas as pd  
import os  
from rdkit import Chem  
from rdkit.Chem import Descriptors  
  
base\_path = "."  
data\_file = os.path.join(base\_path, "data", "extracted\_data.csv")  
  
try:  
 df = pd.read\_csv(data\_file)  
except FileNotFoundError:  
 print(f"Error: File not found at {data\_file}. Make sure you run the SQL script and save the data.")  
 exit()  
  
# Function to convert SMILES to RDKit molecule and calculate MW  
def calculate\_mw(smiles):  
 mol = Chem.MolFromSmiles(smiles)  
 if mol is not None:  
 return Descriptors.MolWt(mol)  
 else:  
 return None  
  
# Apply the function to the 'canonical\_smiles' column  
df['molecular\_weight'] = df['canonical\_smiles'].apply(calculate\_mw)  
  
print(df[['chembl\_id', 'canonical\_smiles', 'molecular\_weight']].head())

**Explanation (English):**

This notebook reads the CSV file containing the SMILES strings. It defines a function calculate\_mw that takes a SMILES string as input, converts it to an RDKit molecule object using Chem.MolFromSmiles(), and then calculates the molecular weight using Descriptors.MolWt(). The function handles cases where the SMILES string is invalid (returns None in that case). Finally, it applies this function to the canonical\_smiles column of the DataFrame and stores the results in a new column called molecular\_weight.

**Explanation (Vietnamese):**

Notebook này đọc tệp CSV chứa các chuỗi SMILES. Nó định nghĩa một hàm calculate\_mw lấy chuỗi SMILES làm đầu vào, chuyển đổi nó thành đối tượng phân tử RDKit bằng Chem.MolFromSmiles() và sau đó tính toán trọng lượng phân tử bằng Descriptors.MolWt(). Hàm xử lý các trường hợp chuỗi SMILES không hợp lệ (trả về None trong trường hợp đó). Cuối cùng, nó áp dụng hàm này cho cột canonical\_smiles của DataFrame và lưu trữ kết quả trong một cột mới có tên là molecular\_weight.

**Example 3: Calculating LogP (Partition Coefficient)**

# Topic\_CheMBL\_35\_31\_3\_rdkit\_logp.ipynb  
  
import pandas as pd  
import os  
from rdkit import Chem  
from rdkit.Chem import Crippen  
  
base\_path = "."  
data\_file = os.path.join(base\_path, "data", "extracted\_data.csv")  
  
try:  
 df = pd.read\_csv(data\_file)  
except FileNotFoundError:  
 print(f"Error: File not found at {data\_file}. Make sure you run the SQL script and save the data.")  
 exit()  
  
# Function to convert SMILES to RDKit molecule and calculate LogP  
def calculate\_logp(smiles):  
 mol = Chem.MolFromSmiles(smiles)  
 if mol is not None:  
 return Crippen.MolLogP(mol)  
 else:  
 return None  
  
# Apply the function to the 'canonical\_smiles' column  
df['logp'] = df['canonical\_smiles'].apply(calculate\_logp)  
  
print(df[['chembl\_id', 'canonical\_smiles', 'logp']].head())

**Explanation (English):**

Very similar to Example 2, this calculates the LogP (partition coefficient) using RDKit’s Crippen.MolLogP function. LogP is a measure of the molecule’s lipophilicity (tendency to dissolve in fats, oils, and nonpolar solvents). The rest of the code follows the same structure as Example 2.

**Explanation (Vietnamese):**

Rất giống với Ví dụ 2, ví dụ này tính toán LogP (hệ số phân vùng) bằng hàm Crippen.MolLogP của RDKit. LogP là một thước đo khả năng ưa lipid của phân tử (xu hướng hòa tan trong chất béo, dầu và dung môi không phân cực). Phần còn lại của mã tuân theo cấu trúc tương tự như Ví dụ 2.

**Example 4: Structure-Activity Relationship (SAR) - Simple Correlation Analysis**

# Topic\_CheMBL\_35\_31\_4\_sar\_correlation.ipynb  
  
import pandas as pd  
import os  
from rdkit import Chem  
from rdkit.Chem import Descriptors  
import numpy as np  
  
base\_path = "."  
data\_file = os.path.join(base\_path, "data", "extracted\_data.csv")  
  
try:  
 df = pd.read\_csv(data\_file)  
except FileNotFoundError:  
 print(f"Error: File not found at {data\_file}. Make sure you run the SQL script and save the data.")  
 exit()  
  
# Remove rows with NaN values  
df = df.dropna()  
  
# Ensure standard\_value is numeric  
df['standard\_value'] = pd.to\_numeric(df['standard\_value'])  
  
# Function to convert SMILES to RDKit molecule and calculate MW  
def calculate\_mw(smiles):  
 mol = Chem.MolFromSmiles(smiles)  
 if mol is not None:  
 return Descriptors.MolWt(mol)  
 else:  
 return None  
  
# Apply the function to the 'canonical\_smiles' column  
df['molecular\_weight'] = df['canonical\_smiles'].apply(calculate\_mw)  
  
#Calculate the correlation between "molecular\_weight" and "standard\_value"  
correlation = df['molecular\_weight'].corr(df['standard\_value'])  
print(f"Correlation between Molecular Weight and IC50: {correlation}")  
  
# Optional: Scatter plot to visualize the relationship  
import matplotlib.pyplot as plt  
plt.scatter(df['molecular\_weight'], df['standard\_value'])  
plt.xlabel("Molecular Weight")  
plt.ylabel("IC50 (nM)")  
plt.title("Molecular Weight vs. IC50")  
plt.show()

**Explanation (English):**

This notebook performs a simple SAR analysis by calculating the correlation between molecular weight and IC50 values. It first reads the data, calculates molecular weight, and then uses the corr() method to calculate the Pearson correlation coefficient between the two columns. A scatter plot is also generated to visualize the relationship. Remember that correlation does not equal causation, but it can provide clues about potential relationships between molecular properties and activity. The code also handles rows with missing values (dropna()) and ensure ‘standard\_value’ column is numeric before calculating the correlation.

**Explanation (Vietnamese):**

Notebook này thực hiện phân tích SAR đơn giản bằng cách tính toán mối tương quan giữa trọng lượng phân tử và giá trị IC50. Đầu tiên, nó đọc dữ liệu, tính toán trọng lượng phân tử, sau đó sử dụng phương thức corr() để tính hệ số tương quan Pearson giữa hai cột. Một biểu đồ phân tán cũng được tạo để trực quan hóa mối quan hệ. Hãy nhớ rằng mối tương quan không tương đương với quan hệ nhân quả, nhưng nó có thể cung cấp manh mối về các mối quan hệ tiềm năng giữa các thuộc tính phân tử và hoạt động. Mã cũng xử lý các hàng có giá trị bị thiếu (dropna()) và đảm bảo cột ‘standard\_value’ là số trước khi tính toán mối tương quan.

**Example 5: Data Visualisation using seaborn library**

# Topic\_CheMBL\_35\_31\_5\_data\_visualisation.ipynb  
  
import pandas as pd  
import os  
import seaborn as sns  
import matplotlib.pyplot as plt  
from rdkit import Chem  
from rdkit.Chem import Descriptors  
  
base\_path = "."  
data\_file = os.path.join(base\_path, "data", "extracted\_data.csv")  
  
try:  
 df = pd.read\_csv(data\_file)  
except FileNotFoundError:  
 print(f"Error: File not found at {data\_file}. Make sure you run the SQL script and save the data.")  
 exit()  
  
# Remove rows with NaN values  
df = df.dropna()  
  
# Ensure standard\_value is numeric  
df['standard\_value'] = pd.to\_numeric(df['standard\_value'])  
  
# Function to convert SMILES to RDKit molecule and calculate MW  
def calculate\_mw(smiles):  
 mol = Chem.MolFromSmiles(smiles)  
 if mol is not None:  
 return Descriptors.MolWt(mol)  
 else:  
 return None  
  
# Apply the function to the 'canonical\_smiles' column  
df['molecular\_weight'] = df['canonical\_smiles'].apply(calculate\_mw)  
  
# Scatter Plot using Seaborn  
sns.scatterplot(x='molecular\_weight', y='standard\_value', data=df)  
plt.title('Molecular Weight vs. Standard Value')  
plt.xlabel('Molecular Weight')  
plt.ylabel('Standard Value')  
plt.show()  
  
# Histogram of Standard Values  
sns.histplot(df['standard\_value'], kde=True) # kde adds a kernel density estimate  
plt.title('Distribution of Standard Values')  
plt.xlabel('Standard Value')  
plt.ylabel('Frequency')  
plt.show()  
  
# Box Plot of Molecular Weight  
sns.boxplot(x=df['molecular\_weight'])  
plt.title('Box Plot of Molecular Weight')  
plt.xlabel('Molecular Weight')  
plt.show()

**Explanation (English):**

This notebook utilizes the seaborn library to create more sophisticated visualizations of the data. It generates a scatter plot (similar to Example 4 but using Seaborn for better aesthetics), a histogram to show the distribution of standard values (IC50), and a box plot to summarize the distribution of molecular weights. Seaborn provides a high-level interface for creating informative and visually appealing statistical graphics.

**Explanation (Vietnamese):**

Notebook này sử dụng thư viện seaborn để tạo ra các hình ảnh trực quan tinh vi hơn về dữ liệu. Nó tạo ra một biểu đồ phân tán (tương tự như Ví dụ 4 nhưng sử dụng Seaborn để có tính thẩm mỹ tốt hơn), một biểu đồ tần suất để hiển thị sự phân bố của các giá trị chuẩn (IC50) và một biểu đồ hộp để tóm tắt sự phân bố của trọng lượng phân tử. Seaborn cung cấp một giao diện cấp cao để tạo ra đồ họa thống kê trực quan hấp dẫn và đầy thông tin.

**Important Considerations:**

* **Database Connection:** These examples assume you have already set up a connection to your Chembl 35 database using psycopg2 or a similar library. You’ll need to adapt the SQL connection code based on your specific setup.
* **File Paths:** Make sure the file paths in the Python notebooks are correct. Adapt base\_path and data\_file as needed.
* **Dependencies:** Ensure you have the necessary libraries installed: pandas, rdkit, matplotlib, and seaborn. You can install them using pip install pandas rdkit matplotlib seaborn.
* **Target Selection:** Remember to replace 'CHEMBL182' in the SQL script with the correct CHEMBL ID for your target of interest.
* **Error Handling:** These examples provide basic error handling, but you should add more robust error handling for production code.
* **Data Cleaning:** Real-world data often requires more extensive cleaning and preprocessing. Be prepared to handle missing values, outliers, and inconsistencies in your data.
* **Units Conversion:** Always be mindful of units. Ensure that activity values are in consistent units before performing any analysis. You may need to convert units if necessary.
* **Statistics understanding:** Make sure you understand the basics of the statistic method to implement

These examples provide a starting point for your Chembl 35 data analysis project. Remember to adapt the code to your specific research question and data. Good luck!