# PharmApp Suite

## 🧠 AI for Drug Discovery and Development 🧪

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

## 🧩 Topic: Time-to-market estimation using ChEMBL history

Okay, I understand. You’re working with ChEMBL 35 data, using PostgreSQL and RDKit within an AI/MLOps framework, and encountering specific errors. Let’s break down the problem and develop the necessary SQL and Python code. We will address the errors, focus on retrieving a small dataset (100 rows), and incorporate RDKit for basic analysis.

**1. Analysis of the Analysis Model**

The goal is to extract data from the ChEMBL 35 database, pre-process it, and perform some preliminary analysis. Based on the topic code, we can infer a general direction:

* **Topic\_CheMBL\_35\_44**: This *likely* suggests an analysis focused around ChEMBL target ID 44 or a process related to target analysis in general within the Chembl 35 database. The specific interpretation would depend on the broader context of your project. We’ll assume it relates to target-related activity data for now.

**Here’s a breakdown of the proposed analysis model:**

1. **Data Extraction (SQL):** We’ll write an SQL query to retrieve relevant data from the ChEMBL database. This will include activity data, compound information (SMILES strings), and target information, focusing on Target ID 44 as the default analysis (or modifying it if necessary based on your specific need). We’ll limit the query to 100 rows for faster prototyping and resource efficiency. The query also include filtering of value ‘~’ to ensure numeric values, which will solve error a.
2. **Data Preprocessing (Python):**
   * Load the data from the CSV file generated by the SQL query.
   * Clean the data: Handle missing values, remove duplicates, and validate SMILES strings.
   * Convert SMILES strings to RDKit Mol objects.
   * Filter valid molecules from invalid molecules.
3. **Basic RDKit Analysis (Python):**
   * Calculate molecular properties using RDKit descriptors (e.g., molecular weight, LogP).
   * **Example 1:** Calculate average molecular weight and LogP for the selected compounds.
   * **Example 2:** Analyze the distribution of a specific descriptor (e.g., LogP) using histograms.
   * **Example 3:** Identify compounds with molecular weights above a certain threshold.
   * **Example 4:** Filter molecules based on Lipinski’s Rule of Five.
   * **Example 5:** Check for specific substructures in the selected compounds.
4. **Error Handling:** We will address the errors you’ve encountered:
   * **Error a (Numeric Operator):** The error operator does not exist: numeric ~ unknown arises because you’re trying to use the ~ operator (likely meaning “contains”) with a numeric column (standard\_value). The ~ operator is designed for text matching (regular expressions). We’ll correct this by using appropriate numeric comparison or type casting if regular expressions are needed on numerical data.
   * **Error b (Scikit-learn):** The squared=False parameter in mean\_squared\_error was introduced in a later version of scikit-learn. We’ll address this by either updating scikit-learn or removing the parameter if MSE is needed or by other appropriate measure. Since we are not using it in the example, we can leave it.

**2. SQL and Python Code**

Here’s the code, structured according to your guidelines:

**SQL Code (ChEMBL\_35\_44.sql):**

-- SQL Query to extract data for ChEMBL Target ID 44 (example)  
-- and limit to 100 rows  
  
SELECT  
 act.activity\_id,  
 cmp.chembl\_id,  
 cmp.canonical\_smiles,  
 act.standard\_type,  
 act.standard\_relation,  
 act.standard\_value,  
 act.standard\_units,  
 act.pchembl\_value,  
 tar.target\_chembl\_id,  
 tar.target\_type  
FROM  
 activities act  
JOIN  
 molecule\_dictionary cmp ON act.molregno = cmp.molregno  
JOIN  
 target\_dictionary tar ON act.tid = tar.tid  
WHERE  
 tar.target\_chembl\_id = 'CHEMBL44' -- Replace with your desired Target ID  
 AND act.standard\_value IS NOT NULL  
 AND act.standard\_type = 'IC50' -- Filtering to get comparable values  
 AND act.standard\_units = 'nM'  
 AND act.standard\_value::text ~ '^[0-9\.]+$' -- Ensure standard\_value is numeric  
ORDER BY  
 act.pchembl\_value DESC  
LIMIT 100;  
  
-- Save the output of this query to a CSV file named chembl\_35\_44.csv  
-- using pgAdmin's export functionality. Save the file to ../data/chembl\_35\_44.csv

**Explanation:**

* The query joins the activities, molecule\_dictionary, and target\_dictionary tables to retrieve activity data, compound information (SMILES), and target information.
* The WHERE clause filters the data based on the target\_chembl\_id (currently set to ‘CHEMBL44’), standard\_type, standard\_units and requires standard\_value to not be NULL. It also uses act.standard\_value::text ~ '^[0-9\.]+$' to filter for numeric values using regular expression matching after casting the numeric values to text.
* The LIMIT 100 clause restricts the result set to 100 rows.
* The comment indicates that you should save the output of this query to a CSV file named chembl\_35\_44.csv and place it in the ../data/ directory.

**Python Code (Topic\_CheMBL\_35\_44\_1\_Data\_Extraction\_and\_Preprocessing.ipynb):**

import os  
import pandas as pd  
from rdkit import Chem  
from rdkit.Chem import Descriptors  
from rdkit.Chem import Lipinski  
  
# Define base path (adjust if needed)  
base\_path = ".." # Assuming notebook is in a subdirectory  
data\_path = os.path.join(base\_path, "data", "chembl\_35\_44.csv")  
  
# Load the data  
try:  
 df = pd.read\_csv(data\_path)  
 print("Data loaded successfully.")  
except FileNotFoundError:  
 print(f"Error: File not found at {data\_path}. Make sure you exported the CSV from pgAdmin to the correct location.")  
 raise # Re-raise the exception to stop execution  
  
# Data Cleaning and Preprocessing  
print("\nData Cleaning and Preprocessing...")  
# 1. Handle missing values (replace with NaN if needed, or drop rows)  
df = df.dropna(subset=['canonical\_smiles']) # Drop rows with missing SMILES  
df = df.fillna(0) # Filling remaining NaN with 0  
# 2. Remove duplicate SMILES  
df = df.drop\_duplicates(subset=['canonical\_smiles'])  
  
# RDKit Conversion and Validation  
print("\nConverting SMILES to RDKit Mol objects...")  
def smiles\_to\_mol(smiles):  
 try:  
 mol = Chem.MolFromSmiles(smiles)  
 if mol is None:  
 return None  
 #Sanitize mol object  
 Chem.SanitizeMol(mol)  
 return mol  
 except:  
 return None  
  
df['mol'] = df['canonical\_smiles'].apply(smiles\_to\_mol)  
  
# Filter out invalid molecules  
df = df[df['mol'].notna()]  
print(f"Number of valid molecules: {len(df)}")  
  
  
# Save the processed dataframe (optional)  
processed\_data\_path = os.path.join(base\_path, "data", "chembl\_35\_44\_processed.csv")  
df.to\_csv(processed\_data\_path, index=False)  
print(f"\nProcessed data saved to: {processed\_data\_path}")  
  
print("\nData Preprocessing Complete.")

**Python Code (Topic\_CheMBL\_35\_44\_2\_RDKit\_Analysis.ipynb):**

import os  
import pandas as pd  
from rdkit import Chem  
from rdkit.Chem import Descriptors  
from rdkit.Chem import Lipinski  
import matplotlib.pyplot as plt  
  
# Define base path (adjust if needed)  
base\_path = ".." # Assuming notebook is in a subdirectory  
processed\_data\_path = os.path.join(base\_path, "data", "chembl\_35\_44\_processed.csv")  
  
# Load the processed data  
try:  
 df = pd.read\_csv(processed\_data\_path)  
 print("Processed data loaded successfully.")  
except FileNotFoundError:  
 print(f"Error: File not found at {processed\_data\_path}. Make sure you ran the data extraction and preprocessing notebook.")  
 raise  
  
# Convert SMILES to RDKit Mol objects (if not already done)  
def smiles\_to\_mol(smiles):  
 try:  
 mol = Chem.MolFromSmiles(smiles)  
 if mol is None:  
 return None  
 #Sanitize mol object  
 Chem.SanitizeMol(mol)  
 return mol  
 except:  
 return None  
  
df['mol'] = df['canonical\_smiles'].apply(smiles\_to\_mol)  
  
# Filter out invalid molecules  
df = df[df['mol'].notna()]  
  
# RDKit Analysis Examples  
print("\nRDKit Analysis Examples:")  
  
# Example 1: Calculate average molecular weight and LogP  
df['MW'] = df['mol'].apply(Descriptors.MolWt)  
df['LogP'] = df['mol'].apply(Descriptors.MolLogP)  
  
avg\_mw = df['MW'].mean()  
avg\_logp = df['LogP'].mean()  
print(f"\nExample 1: Average Molecular Weight: {avg\_mw:.2f}, Average LogP: {avg\_logp:.2f}")  
  
# Example 2: Analyze LogP distribution using a histogram  
plt.hist(df['LogP'], bins=20)  
plt.xlabel("LogP")  
plt.ylabel("Frequency")  
plt.title("LogP Distribution")  
plt.savefig(os.path.join(base\_path, "reports", "logp\_distribution.png")) # Saving the report in report folder, create one if needed  
print("\nExample 2: LogP distribution histogram saved to reports/logp\_distribution.png")  
  
# Example 3: Identify compounds with molecular weights above 500  
heavy\_compounds = df[df['MW'] > 500]  
print(f"\nExample 3: Number of compounds with MW > 500: {len(heavy\_compounds)}")  
print(heavy\_compounds[['chembl\_id', 'canonical\_smiles', 'MW']].head()) # Display first 5  
  
# Example 4: Filter molecules based on Lipinski's Rule of Five  
df['Lipinski\_HBD'] = df['mol'].apply(Lipinski.NumHDonors)  
df['Lipinski\_HBA'] = df['mol'].apply(Lipinski.NumHAcceptors)  
df['Lipinski\_MW'] = df['mol'].apply(Descriptors.MolWt)  
df['Lipinski\_LogP'] = df['mol'].apply(Descriptors.MolLogP)  
  
# Define Lipinski Rule of Five violation  
def lipinski\_rule(row):  
 violations = 0  
 if row['Lipinski\_HBD'] > 5:  
 violations += 1  
 if row['Lipinski\_HBA'] > 10:  
 violations += 1  
 if row['Lipinski\_MW'] > 500:  
 violations += 1  
 if row['Lipinski\_LogP'] > 5:  
 violations += 1  
 return violations  
  
df['Lipinski\_Violations'] = df.apply(lipinski\_rule, axis=1)  
lipinski\_filtered = df[df['Lipinski\_Violations'] <= 1]  
  
print(f"\nExample 4: Number of molecules that pass Lipinski's Rule of Five (<=1 violation): {len(lipinski\_filtered)}")  
  
# Example 5: Check for a specific substructure (e.g., benzene ring)  
benzene = Chem.MolFromSmiles('c1ccccc1')  
def has\_benzene(mol):  
 return mol.HasSubstructMatch(benzene)  
  
df['Has\_Benzene'] = df['mol'].apply(has\_benzene)  
benzene\_containing = df[df['Has\_Benzene'] == True]  
print(f"\nExample 5: Number of molecules containing a benzene ring: {len(benzene\_containing)}")

**Explanation:**

* **Data Loading:** Loads the processed data from the CSV file.
* **RDKit Conversion (if needed):** Converts SMILES strings to RDKit Mol objects. This is included to ensure the ‘mol’ column exists even if it wasn’t saved to the processed CSV.
* **Examples:** Demonstrates several RDKit analysis examples:
  + **Example 1:** Calculates and prints the average molecular weight and LogP.
  + **Example 2:** Generates a histogram of LogP values and saves it to a file.
  + **Example 3:** Identifies and prints the first 5 compounds with molecular weights above 500.
  + **Example 4:** Filters molecules based on Lipinski’s Rule of Five and counts the number of molecules that pass.
  + **Example 5:** Checks for the presence of a benzene ring in each molecule and counts the number of molecules containing it.

**Directory Structure (AIMLops Template):**

ProjectRoot/  
├── data/  
│ ├── chembl\_35\_44.csv (Output from SQL query)  
│ └── chembl\_35\_44\_processed.csv (Output from Python Notebook 1)  
├── notebooks/  
│ ├── Topic\_CheMBL\_35\_44\_1\_Data\_Extraction\_and\_Preprocessing.ipynb  
│ └── Topic\_CheMBL\_35\_44\_2\_RDKit\_Analysis.ipynb  
├── reports/  
│ └── logp\_distribution.png (Output from Example 2)  
├── sql/  
│ └── ChEMBL\_35\_44.sql  
└── ... (other project files)

**Important Considerations and Next Steps:**

* **Target Selection:** I assumed you wanted to analyze CHEMBL44. If your target is different, **modify the WHERE clause in the SQL query** to reflect the correct target\_chembl\_id. This is the *most important* step to tailor the analysis to your specific needs.
* **Error Handling:** The code includes basic error handling (e.g., checking for file existence, invalid SMILES). You should add more robust error handling as needed for production.
* **Data Exploration:** The examples provided are just a starting point. Explore the data further to identify interesting patterns and relationships.
* **Database Connection:** The provided code assumes you’ve already exported the data to a CSV file. For a more direct connection to the database, you would need to use a Python library like psycopg2 to connect to PostgreSQL and execute the SQL query directly from your notebook.
* **Library Versions:** Make sure you have the correct versions of the required libraries (RDKit, Pandas, Matplotlib). You can install them using pip install rdkit pandas matplotlib.

**Two-Language Explanations (English & Chinese):**

**English:**

This analysis pipeline extracts data from ChEMBL 35 related to a specific target (defaulting to CHEMBL44), preprocesses it to remove invalid entries and duplicates, and then performs some basic RDKit-based analysis, including calculating molecular properties, generating histograms, and filtering molecules based on Lipinski’s Rule of Five. The code is structured to fit an AI/MLOps framework with clearly defined data, notebook, and SQL directories. The SQL query is designed to be efficient by limiting the output to 100 rows.

**Chinese (中文):**

这个分析流程从ChEMBL 35数据库中提取与特定靶点（默认为CHEMBL44）相关的数据，对其进行预处理以删除无效条目和重复项，然后执行一些基于RDKit的基本分析，包括计算分子性质、生成直方图以及根据Lipinski规则过滤分子。 代码结构旨在适应AI/MLOps框架，并具有明确定义的数据、notebook和SQL目录。 SQL查询旨在通过将输出限制为100行来提高效率。

I hope this comprehensive explanation and code helps you with your ChEMBL 35 data analysis project! Let me know if you have any more questions.