# PharmApp Suite

## 🧠 AI for Drug Discovery and Development 🧪

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

## 🧩 Topic: Rare disease drug lead identification

Okay, I understand. You’re working on a Chembl 35 data analysis project using RDKit and PostgreSQL, following an AIMLops template. You want to extract data, perform analysis, and you’ve encountered some errors. Let’s break down the problem and create a solution.

**1. Analysis Model and Approach**

The goal here is to leverage Chembl 35 data for drug research and development. Given the errors, we’ll focus on:

* **Data Extraction and Preparation:** Retrieve relevant data from Chembl 35 related to bioactivity data. We’ll limit the data extraction to 100 rows for faster processing as requested. We’ll address the SQL error related to the regular expression matching.
* **Data Transformation:** Convert SMILES strings into RDKit molecules. Handle potential errors that may occur during the conversion.
* **Basic Analysis:** Demonstrate a simple analysis using the data. We will focus on exploring the relationship between molecular properties and bioactivity. Specifically, we’ll calculate molecular weight and logP using RDKit and look at their distribution with respect to activity values. We will also deal with the scikit-learn version incompatibility issue.

**2. Code (SQL and Python)**

Here’s a breakdown of the SQL and Python code, along with explanations:

**SQL (to extract data and save to CSV):**

-- File: ../data/chembl\_35\_100.csv  
-- This SQL script extracts 100 rows of data from ChEMBL related to bioactivity,  
-- addressing the numeric ~ unknown operator error by using a different approach  
-- to filter numeric data.  
  
SELECT  
 act.molregno,  
 act.standard\_value,  
 act.standard\_units,  
 act.standard\_type,  
 md.chembl\_id,  
 cs.canonical\_smiles  
FROM  
 activities act  
JOIN  
 molecule\_dictionary md ON act.molregno = md.molregno  
JOIN  
 compound\_structures cs ON md.molregno = cs.molregno  
WHERE  
 act.standard\_type = 'IC50' -- Filter for IC50 values  
 AND act.standard\_units = 'nM' -- Filter for nM units  
 AND act.standard\_value IS NOT NULL -- Ensure standard\_value is not NULL  
 AND act.standard\_value::text ~ '^[0-9\.]+$' -- Check if standard\_value is a valid number (alternative to numeric ~ unknown)  
ORDER BY  
 act.molregno  
LIMIT 100;

**Explanation:**

* **Error Handling (SQL):** The original error ERROR: operator does not exist: numeric ~ unknown, LINE 12: AND act.standard\_value ~ '^[0-9\.]+$' arises because PostgreSQL’s type checking for regular expression matching can be strict. To avoid type issues, we explicitly cast act.standard\_value to text using act.standard\_value::text before applying the regex check.
* **Data Selection:** We select molregno, standard\_value, standard\_units, standard\_type, chembl\_id, and canonical\_smiles. These fields provide information about the molecule, its activity, and its chemical structure.
* **Filtering:** We filter for IC50 values measured in nM to ensure consistency in the bioactivity data. We also check that standard\_value is not NULL.
* **Limiting Results:** LIMIT 100 ensures that only 100 rows are returned, as requested.

**Python (Jupyter Notebook):**

# File: notebooks/Topic\_CheMBL\_35\_100\_1\_analysis.ipynb  
  
import pandas as pd  
import numpy as np  
from rdkit import Chem  
from rdkit.Chem import Descriptors  
from rdkit.Chem import Lipinski  
import os  
from sklearn.metrics import mean\_squared\_error  
  
# Define base path  
base\_path = ".." # Assuming the notebook is one level deep relative to the project root  
data\_path = os.path.join(base\_path, "data", "chembl\_35\_100.csv")  
  
# Load data from CSV  
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've run the SQL script and saved the CSV.")  
 exit()  
  
# Data Cleaning and Preparation  
df = df.dropna(subset=['canonical\_smiles', 'standard\_value']) # Remove rows with missing SMILES or standard\_value  
  
# Convert standard\_value to numeric, handling potential errors  
df['standard\_value'] = pd.to\_numeric(df['standard\_value'], errors='coerce') # Coerce errors to NaN  
df = df.dropna(subset=['standard\_value']) # Drop rows with NaN standard\_value after conversion  
  
# RDKit Molecule Creation and Feature Calculation  
def calculate\_properties(smiles):  
 mol = Chem.MolFromSmiles(smiles)  
 if mol is not None:  
 mw = Descriptors.MolWt(mol)  
 logp = Descriptors.MolLogP(mol)  
 hbd = Lipinski.NumHDonors(mol)  
 hba = Lipinski.NumHAcceptors(mol)  
 return mw, logp, hbd, hba  
 else:  
 return None, None, None, None  
  
df[['mol\_weight', 'logP', 'HBD', 'HBA']] = df['canonical\_smiles'].apply(lambda x: pd.Series(calculate\_properties(x)))  
df = df.dropna(subset=['mol\_weight', 'logP', 'HBD', 'HBA']) # Drop rows where RDKit failed  
  
# Activity Transformation (pIC50)  
df['pIC50'] = -np.log10(df['standard\_value'] \* 1e-9) # Convert IC50 (nM) to pIC50  
  
# Simple Analysis (Example: Relationship between Molecular Weight and pIC50)  
# Let's try a very basic linear regression. This is just for illustration.  
from sklearn.model\_selection import train\_test\_split  
from sklearn.linear\_model import LinearRegression  
  
X = df[['mol\_weight', 'logP']] # Features  
y = df['pIC50'] # Target  
  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42) # Split data  
  
model = LinearRegression()  
model.fit(X\_train, y\_train)  
y\_pred = model.predict(X\_test)  
  
# Calculate Mean Squared Error  
try:  
 mse = mean\_squared\_error(y\_test, y\_pred)  
except TypeError as e:  
 print(f"Error calculating MSE: {e}")  
 print("This might be due to an old scikit-learn version. Ensure it's up-to-date.")  
 mse = None # Set mse to None if calculation fails  
else:  
 print(f"Mean Squared Error: {mse}")  
  
if mse is not None:  
 # Optionally, you can try to fix the scikit-learn version issue.  
 # However, this often requires a full environment rebuild.  
 # print("Trying to fix scikit-learn version issue (requires restart):")  
 # !pip install scikit-learn --upgrade  
  
 import matplotlib.pyplot as plt  
 plt.scatter(y\_test, y\_pred)  
 plt.xlabel("Actual pIC50")  
 plt.ylabel("Predicted pIC50")  
 plt.title("Actual vs. Predicted pIC50")  
 plt.show()  
  
  
# Display the first few rows of the processed DataFrame  
print("\nProcessed Data (First 5 rows):")  
print(df.head())

**Explanation:**

* **Import Libraries:** Import necessary libraries (pandas, RDKit, scikit-learn, os).
* **File Path:** Construct the correct file path to the CSV file using os.path.join.
* **Load Data:** Load the CSV data into a pandas DataFrame.
* **Data Cleaning:** Drop rows with missing SMILES or standard\_value.
* **RDKit Molecule Conversion:** The calculate\_properties function takes a SMILES string and returns molecular weight, logP, H-bond donors, and H-bond acceptors. It handles potential errors during the RDKit molecule conversion by returning None if the conversion fails.
* **Feature Calculation:** Apply the calculate\_properties function to the canonical\_smiles column to create new columns for molecular weight and logP.
* **Activity Transformation:** Convert IC50 values to pIC50 using the formula: pIC50 = -log10(IC50 \* 1e-9). This transforms the IC50 values (in nM) to a more interpretable scale.
* **Basic Analysis (Linear Regression):**
  + Split the data into training and testing sets.
  + Create a LinearRegression model.
  + Train the model on the training data.
  + Make predictions on the testing data.
  + Calculate the Mean Squared Error (MSE).
* **Error Handling (scikit-learn):** The original code had an issue with the squared=False parameter in mean\_squared\_error, which is not supported in older scikit-learn versions. I have removed this parameter, so the code will now compute the regular MSE (not the root mean squared error). You should upgrade scikit-learn if possible, using !pip install scikit-learn --upgrade. I’ve added a check to see if mse is None, and if so, we do not display the plot.
* **Display Results:** Print the first few rows of the processed DataFrame.

**3. Five Examples of How to Extend the Analysis**

Here are five examples to extend the analysis, building upon the code above:

1. **More Advanced Regression Models:** Instead of a simple linear regression, try more sophisticated models like Random Forest Regression, Support Vector Regression (SVR), or Gradient Boosting Regression. These models can capture non-linear relationships between molecular properties and activity.

* from sklearn.ensemble import RandomForestRegressor  
    
  # ... (Data preparation code from above) ...  
    
  model = RandomForestRegressor(n\_estimators=100, random\_state=42) # Example: Random Forest with 100 trees  
  model.fit(X\_train, y\_train)  
  y\_pred = model.predict(X\_test)  
  mse = mean\_squared\_error(y\_test, y\_pred)  
  print(f"Random Forest MSE: {mse}")

1. **Feature Engineering:** Calculate more molecular descriptors using RDKit. Explore descriptors like:
   * **Topological Polar Surface Area (TPSA):** Related to drug absorption and permeability.
   * **Number of Rotatable Bonds:** Indicates molecular flexibility.
   * **Aromatic Ring Count:** Related to aromaticity.

* from rdkit.Chem import rdMolDescriptors  
    
  def calculate\_more\_properties(mol):  
   if mol is not None:  
   tpsa = rdMolDescriptors.CalcTPSA(mol)  
   rotatable\_bonds = rdMolDescriptors.CalcNumRotatableBonds(mol)  
   aromatic\_rings = rdMolDescriptors.CalcNumAromaticRings(mol)  
   return tpsa, rotatable\_bonds, aromatic\_rings  
   else:  
   return None, None, None  
    
  df[['TPSA', 'RotatableBonds', 'AromaticRings']] = df['canonical\_smiles'].apply(lambda x: pd.Series(calculate\_more\_properties(Chem.MolFromSmiles(x))))  
  df = df.dropna(subset=['TPSA', 'RotatableBonds', 'AromaticRings'])  
    
  # Add the new features to your X matrix for modeling  
  X = df[['mol\_weight', 'logP', 'TPSA', 'RotatableBonds', 'AromaticRings']]

1. **Activity Cliffs Analysis:** Identify pairs of molecules with similar structures but significantly different activities. This can help pinpoint crucial structural features that influence activity. This requires calculating molecular similarity (e.g., Tanimoto similarity) using RDKit.

* from rdkit import DataStructs  
  from rdkit.Chem import AllChem  
    
  def calculate\_fingerprint(mol):  
   if mol is not None:  
   fp = AllChem.GetMorganFingerprintAsBitVect(mol, 2, nBits=2048) # Morgan Fingerprint  
   return fp  
   else:  
   return None  
    
  df['fingerprint'] = df['canonical\_smiles'].apply(lambda x: calculate\_fingerprint(Chem.MolFromSmiles(x)))  
  df = df.dropna(subset=['fingerprint'])  
    
  # Now, iterate through pairs of molecules and calculate Tanimoto similarity  
  # This is just a basic example. A real activity cliff analysis would require more sophisticated methods.  
  from itertools import combinations  
    
  activity\_cliff\_threshold = 1.0 # pIC50 difference threshold  
  similarity\_threshold = 0.8 # Tanimoto similarity threshold  
    
  activity\_cliffs = []  
  for (idx1, idx2) in combinations(df.index, 2):  
   mol1 = df.loc[idx1]  
   mol2 = df.loc[idx2]  
    
   similarity = DataStructs.TanimotoSimilarity(mol1['fingerprint'], mol2['fingerprint'])  
   activity\_difference = abs(mol1['pIC50'] - mol2['pIC50'])  
    
   if similarity >= similarity\_threshold and activity\_difference >= activity\_cliff\_threshold:  
   activity\_cliffs.append((mol1['chembl\_id'], mol2['chembl\_id'], similarity, activity\_difference))  
    
  print(f"Found {len(activity\_cliffs)} potential activity cliffs.")

1. **Clustering:** Group molecules based on their properties (molecular weight, logP, TPSA, etc.) using clustering algorithms like k-means or hierarchical clustering. This can help identify groups of compounds with similar characteristics.

* from sklearn.cluster import KMeans  
  from sklearn.preprocessing import StandardScaler  
    
  # ... (Data preparation code from above) ...  
    
  X = df[['mol\_weight', 'logP', 'TPSA', 'RotatableBonds', 'AromaticRings']]  
    
  # Scale the features  
  scaler = StandardScaler()  
  X\_scaled = scaler.fit\_transform(X)  
    
  # Perform k-means clustering  
  kmeans = KMeans(n\_clusters=5, random\_state=42, n\_init=10) # Example: 5 clusters  
  df['cluster'] = kmeans.fit\_predict(X\_scaled)  
    
  # Analyze the clusters  
  print(df.groupby('cluster')[['mol\_weight', 'logP', 'TPSA']].mean())

1. **Visualization:** Create visualizations to explore the relationships between molecular properties and activity. Use scatter plots, histograms, box plots, and other visualization techniques to gain insights into the data.

* import matplotlib.pyplot as plt  
  import seaborn as sns  
    
  # ... (Data preparation code from above) ...  
    
  # Scatter plot of logP vs. pIC50  
  plt.figure(figsize=(8, 6))  
  sns.scatterplot(x='logP', y='pIC50', data=df)  
  plt.xlabel("LogP")  
  plt.ylabel("pIC50")  
  plt.title("LogP vs. pIC50")  
  plt.show()  
    
  # Histogram of pIC50 values  
  plt.figure(figsize=(8, 6))  
  sns.histplot(df['pIC50'], kde=True)  
  plt.xlabel("pIC50")  
  plt.title("Distribution of pIC50 Values")  
  plt.show()

**4. Important Considerations**

* **Data Quality:** The quality of your analysis depends heavily on the quality of the Chembl 35 data. Pay attention to data cleaning and validation.
* **Statistical Significance:** Be mindful of statistical significance when interpreting results. With only 100 data points, your statistical power will be limited.
* **Domain Knowledge:** Drug discovery benefits from domain expertise. Consult with chemists and biologists to interpret your findings effectively.
* **Environment Consistency:** Ensure that your RDKit and scikit-learn versions are consistent and up-to-date. Use a virtual environment (e.g., conda or venv) to manage dependencies.
* **Error Handling:** Robust error handling is crucial. Anticipate potential issues (e.g., invalid SMILES strings, missing data) and handle them gracefully.

**In Summary (Resumen en Español):**

Este proyecto tiene como objetivo analizar datos de Chembl 35 para la investigación y el desarrollo de fármacos utilizando RDKit y PostgreSQL. Primero, extraemos datos relevantes de Chembl 35, limitando la extracción a 100 filas. Luego, convertimos las cadenas SMILES en moléculas RDKit y calculamos propiedades moleculares como el peso molecular y el logP. Realizamos un análisis básico utilizando regresión lineal para explorar la relación entre las propiedades moleculares y la bioactividad (pIC50). También abordamos errores como problemas de tipos de datos en SQL y la incompatibilidad de versiones de scikit-learn. Finalmente, proporcionamos cinco ejemplos para ampliar el análisis, incluyendo modelos de regresión más avanzados, ingeniería de características, análisis de “activity cliffs”, clustering y visualización de datos. Es crucial prestar atención a la calidad de los datos, la significación estadística y utilizar el conocimiento del dominio para interpretar los resultados de manera efectiva.