# PharmApp Suite

## 🧠 AI for Drug Discovery and Development 🧪

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

## 🧩 Topic: 3D plotting of molecular chemical space

Okay, I understand. You’re working on a Chembl 35 data analysis project using RDKit, and you’re following the AIMLops template. You’ll be using SQL to extract data, saving it to CSV, and then using Jupyter notebooks with Python and RDKit for analysis. You’ve encountered a few errors along the way and need guidance on the code and data analysis approach. Let’s break down the approach and code, keeping in mind your errors and constraints.

**1. Analysis Model and Approach**

Given the errors you’ve encountered and the context of drug research and development, a good starting point would be to focus on analyzing the relationship between chemical structure (SMILES from ChEMBL) and activity (IC50, Ki, etc., from ChEMBL). Here’s a possible analysis model:

* **Data Extraction and Cleaning:** Extract relevant data (SMILES, activity values, units, target information) from ChEMBL using SQL. Clean the data by standardizing units, handling missing values, and filtering for specific activity types (e.g., IC50). The error you’re seeing (numeric ~ unknown) indicates a type mismatch in your SQL query. You’ll need to cast the standard\_value column to a text type before using the regular expression operator ~.
* **Feature Generation:** Generate molecular descriptors from the SMILES strings using RDKit. These descriptors will serve as features in your model. Common descriptors include:
  + Molecular weight (MolWt)
  + LogP (octanol-water partition coefficient)
  + Topological Polar Surface Area (TPSA)
  + Number of hydrogen bond donors/acceptors
  + Rotatable bond count
  + Aromatic ring count
* **Activity Prediction (Regression):** Build a regression model to predict activity values based on the molecular descriptors. Since you’re using an older scikit-learn version, you’ll need to adjust your code accordingly. Common regression models include:
  + Linear Regression
  + Random Forest Regression
  + Support Vector Regression (SVR)
* **Model Evaluation:** Evaluate the performance of the model using metrics such as Mean Squared Error (MSE), Root Mean Squared Error (RMSE), R-squared (coefficient of determination), and Mean Absolute Error (MAE).
* **Example Application:** You can explore Structure-Activity Relationship (SAR) by analyzing how changes in specific descriptors impact activity predictions. You can also identify potential lead compounds based on predicted activity.

**2. SQL Code**

Here’s an SQL query to extract data from ChEMBL 35, addressing the error you encountered and limiting the results to 100 rows:

-- File: ../data/chembl\_35\_data.sql (This file should be saved with .sql extension)  
-- Connect to pgAdmin and execute this query to save results to a CSV file.  
-- Make sure you have write permissions to the ../data directory.  
  
COPY (  
 SELECT  
 md.chembl\_id,  
 cs.canonical\_smiles,  
 act.standard\_type,  
 act.standard\_value,  
 act.standard\_units  
 FROM  
 molecule\_dictionary md  
 JOIN  
 compound\_structures cs ON md.molregno = cs.molregno  
 JOIN  
 activities act ON md.molregno = act.molregno  
 WHERE  
 act.standard\_type IN ('IC50', 'Ki', 'Kd') -- Filter for common activity types  
 AND act.standard\_units = 'nM' -- Filter for nM units  
 AND act.standard\_value IS NOT NULL --exclude null values  
 AND act.standard\_value::text ~ '^[0-9\.]+$' -- Filter for numeric values using regex (cast to text first)  
 AND LENGTH(cs.canonical\_smiles) < 200 -- filter for SMILES with less than 200 length  
 LIMIT 100  
) TO '../data/chembl\_35\_data.csv' WITH CSV HEADER;  
  
-- IMPORTANT:  
-- 1. Ensure that your PostgreSQL user ('rd' in your case) has the necessary permissions to read from the tables in the `chembl\_35` database and write to the `../data/chembl\_35\_data.csv` file. This usually involves granting read permissions on the tables and write permissions on the directory.  
-- 2. The `COPY` command executes on the \*server\*. The path '../data/chembl\_35\_data.csv' is relative to the \*server's\* filesystem, not your local machine. Therefore, you must ensure that the data folder is accessible from your PostgreSQL server.

**Explanation:**

* The query selects the ChEMBL ID, SMILES string, standard type, standard value, and standard units.
* It joins the molecule\_dictionary, compound\_structures, and activities tables to retrieve the necessary information.
* It filters for specific activity types (IC50, Ki, Kd) and units (nM).
* The AND act.standard\_value::text ~ '^[0-9\.]+$' part is crucial. It casts the standard\_value column to text before applying the regular expression. The regular expression ^[0-9\.]+$ ensures that the value contains only numbers and periods. This addresses the error you encountered.
* LIMIT 100 restricts the output to the first 100 rows.
* It uses the COPY command to write the results to a CSV file. The WITH CSV HEADER option adds a header row to the CSV file.

**3. Python Code (Jupyter Notebook)**

Here’s Python code to load the CSV data, generate molecular descriptors, and build a regression model. It addresses the scikit-learn version issue.

# File: notebooks/Topic\_CheMBL\_35\_74\_1\_data\_analysis.ipynb  
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  
from sklearn.preprocessing import StandardScaler  
  
# Define the base path for your project  
base\_path = os.path.abspath(os.path.join(os.getcwd(), "..")) # Assuming notebook is one level deep relative to base  
data\_path = os.path.join(base\_path, "data", "chembl\_35\_data.csv")  
print(f"Data path: {data\_path}")  
  
# 1. 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 ran the SQL script and saved the CSV file correctly.")  
 exit() # Exit the script if the file is not found  
  
# 2. Data Cleaning and Preprocessing  
df = df.dropna(subset=['canonical\_smiles', 'standard\_value']) # Remove rows with missing SMILES or activity values  
  
# Convert standard\_value to numeric, handling potential errors  
df['standard\_value'] = pd.to\_numeric(df['standard\_value'], errors='coerce')  
df = df.dropna(subset=['standard\_value']) # Remove rows where conversion failed  
  
# Optional: Convert IC50 to pIC50 (more common for modeling)  
def ic50\_to\_pic50(ic50\_nM):  
 """Converts IC50 in nM to pIC50."""  
 pIC50 = -np.log10(ic50\_nM \* 1e-9) # Convert nM to M and then to pIC50  
 return pIC50  
  
# Apply pIC50 transformation if standard\_type is IC50  
df['pIC50'] = df.apply(lambda row: ic50\_to\_pic50(row['standard\_value']) if row['standard\_type'] == 'IC50' else None, axis=1)  
  
# Fill NaN values in 'pIC50' with original standard\_value if standard\_type is not IC50  
df['pIC50'] = df['pIC50'].fillna(df['standard\_value'])  
  
  
# 3. Feature Generation (Molecular Descriptors)  
def calculate\_descriptors(smiles):  
 """Calculates a set of RDKit descriptors for a given SMILES string."""  
 mol = Chem.MolFromSmiles(smiles)  
 if mol is None:  
 return None  
 descriptors = {}  
 descriptors['MolWt'] = Descriptors.MolWt(mol)  
 descriptors['LogP'] = Descriptors.MolLogP(mol)  
 descriptors['HBD'] = Descriptors.NumHDonors(mol)  
 descriptors['HBA'] = Descriptors.NumHAcceptors(mol)  
 descriptors['TPSA'] = Descriptors.TPSA(mol)  
  
 # Handle cases where descriptor calculation might fail gracefully:  
 try:  
 descriptors['RotatableBonds'] = Descriptors.NumRotatableBonds(mol)  
 except:  
 descriptors['RotatableBonds'] = 0 # Or some other sensible default  
 return descriptors  
  
# Apply the descriptor calculation function to each SMILES string  
df['descriptors'] = df['canonical\_smiles'].apply(calculate\_descriptors)  
  
# Drop rows where descriptor calculation failed (resulting in None)  
df = df.dropna(subset=['descriptors'])  
  
# Convert the 'descriptors' column (dictionary) into separate columns  
df = pd.concat([df.drop(['descriptors'], axis=1), df['descriptors'].apply(pd.Series)], axis=1)  
  
  
# Drop rows with NaN values after feature generation  
df = df.dropna()  
  
# 4. Model Building  
# Prepare data for the model  
X = df[['MolWt', 'LogP', 'HBD', 'HBA', 'TPSA', 'RotatableBonds']] # Use descriptor columns  
y = df['pIC50'] # Use pIC50 as the target variable  
  
# Data scaling  
scaler = StandardScaler()  
X\_scaled = scaler.fit\_transform(X)  
  
  
# Split data into training and testing sets  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_scaled, y, test\_size=0.2, random\_state=42)  
  
# Create and train the 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)  
rmse = np.sqrt(mse)  
r2 = r2\_score(y\_test, y\_pred)  
  
print(f"Mean Squared Error: {mse}")  
print(f"Root Mean Squared Error: {rmse}")  
print(f"R-squared: {r2}")  
  
# 5. Example Applications  
# Example 1: Predict activity for a new compound  
new\_smiles = 'CC(=O)Oc1ccccc1C(=O)O' # Aspirin  
new\_descriptors = calculate\_descriptors(new\_smiles)  
if new\_descriptors:  
 new\_df = pd.DataFrame([new\_descriptors])  
 new\_scaled = scaler.transform(new\_df)  
 predicted\_activity = model.predict(new\_scaled)  
 print(f"Predicted activity for Aspirin: {predicted\_activity[0]}")  
else:  
 print("Could not calculate descriptors for the new compound.")  
  
# Example 2: Analyze the most important features (for linear regression)  
print("\nFeature Importance (Linear Regression):")  
for i, feature in enumerate(X.columns):  
 print(f"{feature}: {model.coef\_[i]}")  
  
# Example 3: Compare predicted vs. actual values for a few compounds in the test set  
print("\nPredicted vs. Actual (First 5 Compounds):")  
for i in range(min(5, len(y\_test))):  
 print(f"Actual: {y\_test.iloc[i]:.2f}, Predicted: {y\_pred[i]:.2f}")  
  
# Example 4: Identify compounds with high predicted activity  
high\_activity\_threshold = 6 # Example threshold  
high\_activity\_indices = np.where(y\_pred > high\_activity\_threshold)[0]  
print(f"\nNumber of compounds with predicted activity > {high\_activity\_threshold}: {len(high\_activity\_indices)}")  
if len(high\_activity\_indices) > 0:  
 print("ChEMBL IDs of compounds with high predicted activity (example):")  
 for i in high\_activity\_indices[:5]: # Show only first 5  
 print(df['chembl\_id'].iloc[X\_test.index[i]])  
  
# Example 5: Impact of molecular weight on predicted activity  
import matplotlib.pyplot as plt  
plt.scatter(X\_test[:, 0], y\_test, label="Actual", alpha=0.5) # Molecular weight is the first column  
plt.scatter(X\_test[:, 0], y\_pred, label="Predicted", alpha=0.5)  
plt.xlabel("Molecular Weight (Scaled)")  
plt.ylabel("pIC50")  
plt.title("Impact of Molecular Weight on Predicted Activity")  
plt.legend()  
plt.show()

**Explanation:**

1. **Import Libraries:** Imports necessary libraries (pandas, RDKit, scikit-learn).
2. **Load Data:** Loads the CSV file created by the SQL script. Includes error handling if the file is not found.
3. **Data Cleaning:**
   * Removes rows with missing SMILES strings or activity values.
   * Converts standard\_value to numeric.
   * Optionally converts IC50 values to pIC50. This is often preferred because pIC50 values are logarithmically scaled and more suitable for regression models.
4. **Feature Generation:**
   * The calculate\_descriptors function takes a SMILES string as input and calculates a set of molecular descriptors using RDKit.
   * The function is applied to each SMILES string in the DataFrame.
   * Handles cases where descriptor calculation might fail (e.g., invalid SMILES)
   * Expands the dictionary of descriptors into individual columns in the DataFrame.
5. **Model Building:**
   * Selects the descriptor columns as features (X) and the pIC50 column as the target variable (y).
   * Splits the data into training and testing sets.
   * Creates a Linear Regression model and trains it on the training data.
   * Makes predictions on the test data.
   * Evaluates the model using Mean Squared Error (MSE), Root Mean Squared Error (RMSE), and R-squared.
6. **Example Applications:**
   * **Example 1:** Predicts activity for a new compound (Aspirin) based on its SMILES string.
   * **Example 2:** Analyzes the feature importance from the linear regression model, showing the coefficients.
   * **Example 3:** Compares predicted vs. actual activity values for the first 5 compounds in the test set.
   * **Example 4:** Identifies compounds with high predicted activity (above a certain threshold).
   * **Example 5:** Visualizes the relationship between molecular weight and predicted activity.

**Important Considerations:**

* **File Paths:** Double-check that the file paths in your Python code and SQL script are correct. The base\_path calculation is crucial for ensuring that the code works regardless of the current working directory. The paths specified in the SQL COPY command are *relative to the PostgreSQL server’s filesystem*.
* **Permissions:** Ensure that your PostgreSQL user has the necessary permissions to read from the tables in the chembl\_35 database and write to the CSV file.
* **Error Handling:** The Python code includes basic error handling, but you should add more robust error handling as needed. For example, you might want to log errors to a file.
* **Model Selection:** Linear Regression is a good starting point, but you should experiment with other regression models (e.g., Random Forest, SVR) to see if you can improve performance.
* **Feature Selection:** You might want to perform feature selection to identify the most relevant descriptors for predicting activity.
* **Data Size:** While you’re starting with 100 rows, eventually you’ll want to work with a larger dataset. Consider using techniques like data sampling or distributed computing to handle larger datasets.

**Troubleshooting Tips:**

* **SQL Errors:** If you encounter SQL errors, double-check the syntax of your query and ensure that the table and column names are correct. Also, verify that the data types are compatible.
* **Python Errors:** If you encounter Python errors, read the error message carefully and use a debugger to step through your code.
* **RDKit Errors:** If you encounter RDKit errors, make sure that the SMILES strings are valid. You can use RDKit to validate SMILES strings.
* **Model Performance:** If your model performance is poor, try different regression models, feature selection techniques, and data preprocessing steps.

This comprehensive guide should help you get started with your Chembl 35 data analysis project. Good luck!

**Two Languages: (English / Simplified Chinese)**

**English:**

Above is the detailed analysis and code. Follow the instructions carefully. Make sure file paths and permissions are configured correctly. If you encounter problems, debug step by step. Good luck!

**Simplified Chinese:**

以上是详细的分析和代码。 请仔细按照说明进行操作。 确保文件路径和权限配置正确。 如果遇到问题，请逐步调试。 祝你好运！