# PharmApp Suite

## 🧠 AI for Drug Discovery and Development 🧪

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

## 🧩 Topic: Annotating ChEMBL targets with PDB structures

Okay, I understand. You are working on a drug discovery project using ChEMBL 35 data accessed via PostgreSQL, with code structured according to an AIMLops template. You’re encountering issues with data filtering and a deprecated scikit-learn parameter. Let’s break down the analysis, code, and provide examples.

**1. Analysis of the Analysis Model**

Your goal is likely to extract, process, and analyze data from the ChEMBL 35 database to gain insights for drug discovery. This involves:

* **Data Extraction (SQL):** You’ll use SQL queries to retrieve specific activity data from ChEMBL based on criteria such as target, assay type, and activity values. The error ERROR: operator does not exist: numeric ~ unknown, LINE 12: AND act.standard\_value ~ '^[0-9\.]+$' suggests you’re trying to use a regular expression (~) on a numeric column (act.standard\_value). PostgreSQL likely requires you to cast the numeric column to text before applying the regex. The regex '^[0-9\.]+$' aims to filter values that are numbers (including decimals).
* **Data Processing (Python/RDKit):** Once you have the data, you’ll use Python and RDKit to:
  + Clean and standardize the data (e.g., handling missing values, converting units).
  + Generate molecular representations (e.g., SMILES strings to RDKit Mol objects).
  + Calculate molecular descriptors (e.g., physicochemical properties, structural features).
* **Data Analysis (Python/Scikit-learn, etc.):** You’ll use the processed data to perform various analyses, such as:
  + Building predictive models (e.g., activity prediction, toxicity prediction).
  + Performing structure-activity relationship (SAR) analysis.
  + Identifying potential drug candidates.
* **AIMLops Integration:** Packaging your code and workflows so they are repeatable, automatable, and deployable in a consistent manner.

**2. SQL Code (English and Chinese Explanation)**

-- English: SQL query to retrieve activity data for a specific target, filtering for valid numeric standard values, and limiting to 100 rows.  
  
-- Chinese (Simplified): SQL 查询，检索特定靶标的活性数据，过滤有效的数字标准值，并限制为 100 行。  
  
SELECT  
 act.molregno,  
 act.standard\_value,  
 act.standard\_units,  
 act.assay\_id,  
 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' -- Example: Filter for IC50 values. Modify as needed.  
 AND act.standard\_relation = '=' -- Ensure the relationship is an equality  
 AND act.standard\_value IS NOT NULL -- Exclude null values  
 AND act.standard\_value::TEXT ~ '^[0-9\.]+$' -- Cast to TEXT and use regex to check for numeric values.  
LIMIT 100;  
  
--Save result as: ../data/chembl\_activity\_data.csv

**Explanation:**

* SELECT: Specifies the columns to retrieve.
* FROM: Specifies the tables to retrieve data from (activities, molecule\_dictionary, compound\_structures).
* JOIN: Combines data from the tables based on matching columns (molregno). This is crucial to link activity data to molecule information.
* WHERE: Filters the data based on several criteria:
  + act.standard\_type = 'IC50': Filters for activities reported as IC50 values (adjust this to the activity type you’re interested in, e.g., ‘Ki’, ‘EC50’).
  + act.standard\_relation = '=': Only include data where the relationship is equals to.
  + act.standard\_value IS NOT NULL: Excludes rows where standard\_value is NULL.
  + act.standard\_value::TEXT ~ '^[0-9\.]+$': This is the corrected part. It *casts* the standard\_value column to TEXT so that it can be used with the regular expression operator ~. The regular expression '^[0-9\.]+$' ensures that the value consists only of digits and periods (allowing for decimal numbers).
* LIMIT 100: Restricts the output to 100 rows.

**3. Python Code (English and Chinese Explanation)**

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  
import warnings  
warnings.filterwarnings("ignore")  
  
  
# Define paths according to AIMLops template  
base\_path = ".." # Assuming the notebook is one level down from the base  
data\_path = os.path.join(base\_path, "data", "chembl\_activity\_data.csv")  
model\_path = os.path.join(base\_path, "models")  
  
if not os.path.exists(model\_path):  
 os.makedirs(model\_path)  
  
# Load data  
try:  
 df = pd.read\_csv(data\_path)  
except FileNotFoundError:  
 print(f"Error: File not found at {data\_path}. Make sure you have run the SQL query and saved the data.")  
 exit()  
  
# Data Cleaning and Preparation  
df = df.dropna(subset=['canonical\_smiles', 'standard\_value'])  
df = df[df['standard\_value'] > 0] # Remove zero or negative values  
  
# Function to calculate molecular weight (example descriptor)  
def calculate\_mw(smiles):  
 mol = Chem.MolFromSmiles(smiles)  
 if mol:  
 return Descriptors.MolWt(mol)  
 else:  
 return None  
  
# Apply the function to create a new column  
df['molecular\_weight'] = df['canonical\_smiles'].apply(calculate\_mw)  
df = df.dropna(subset=['molecular\_weight']) # remove compounds where MW could not be calculated  
  
  
# Feature Selection and Model Training (Simple Example)  
X = df[['molecular\_weight']] # Use molecular weight as a single feature  
y = np.log10(df['standard\_value']) # Log transform the standard value. Crucial for activity data.  
  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42) #random\_state for reproducibility  
  
# Linear Regression Model  
model = LinearRegression()  
model.fit(X\_train, y\_train)  
  
# Predictions and Evaluation  
y\_pred = model.predict(X\_test)  
  
mse = mean\_squared\_error(y\_test, y\_pred) # No need to specify squared=False, as it's the default. It was deprecated in older scikit-learn versions.  
r2 = r2\_score(y\_test, y\_pred)  
  
print(f"Mean Squared Error: {mse}")  
print(f"R-squared: {r2}")  
  
# Save the model (Optional)  
import joblib  
joblib.dump(model, os.path.join(model\_path, 'linear\_regression\_model.pkl'))  
  
  
# Chinese (Simplified): Python 代码，用于加载、清洗、处理 ChEMBL 数据，计算分子描述符，并训练简单的线性回归模型。  
  
# 导入必要的库  
# 定义文件路径  
# 加载数据  
# 数据清洗：删除缺失值和无效值  
# 定义计算分子量的函数  
# 应用函数创建新列  
# 特征选择和模型训练  
# 划分训练集和测试集  
# 创建线性回归模型  
# 训练模型  
# 预测和评估  
# 保存模型

**Explanation:**

1. **Import Libraries:** Imports necessary libraries (pandas, RDKit, scikit-learn).
2. **Define Paths:** Sets up file paths according to your AIMLops template. This makes your code portable.
3. **Load Data:** Loads the CSV data you saved from the SQL query. Includes error handling if the file is not found.
4. **Data Cleaning:** Removes rows with missing values in the canonical\_smiles and standard\_value columns. Removes rows with activity values of 0 or less.
5. **Molecular Descriptor Calculation:**
   * calculate\_mw(smiles): Defines a function that takes a SMILES string as input, converts it to an RDKit Mol object, and calculates the molecular weight. Returns None if the SMILES string is invalid.
   * Applies the function to the canonical\_smiles column to create a new column called molecular\_weight.
   * Removes rows where the molecular weight could not be calculated (invalid SMILES).
6. **Feature Selection and Model Training:**
   * X = df[['molecular\_weight']]: Selects molecular weight as the independent variable (feature). This is a very simple example; you would typically use many more descriptors.
   * y = np.log10(df['standard\_value']): Selects the standard\_value as the dependent variable (target). **Crucially, it applies a log transformation.** Activity data is often log-normally distributed, and taking the logarithm helps to improve the performance of linear models.
   * train\_test\_split: Splits the data into training and testing sets. random\_state ensures reproducibility.
   * LinearRegression: Creates a linear regression model.
   * model.fit(X\_train, y\_train): Trains the model on the training data.
7. **Predictions and Evaluation:**
   * y\_pred = model.predict(X\_test): Makes predictions on the test data.
   * mse = mean\_squared\_error(y\_test, y\_pred): Calculates the mean squared error. The squared=False argument is *not* needed, as it is deprecated in later versions of scikit-learn. In recent versions, it defaults to calculating the *mean squared error*, not the root mean squared error.
   * r2 = r2\_score(y\_test, y\_pred): Calculates the R-squared value (coefficient of determination).
   * Prints the MSE and R-squared values.
8. **Save Model:** Saves the trained model to a file using joblib.

**Important Notes:**

* **Error Correction:** The SQL query now casts the standard\_value to TEXT before applying the regex. The Python code removes the deprecated squared=False argument from mean\_squared\_error.
* **Log Transformation:** The log transformation of the standard\_value is crucial for activity data.
* **Feature Engineering:** This example uses only molecular weight as a feature. In a real drug discovery project, you would calculate a much larger set of molecular descriptors using RDKit and perform feature selection to identify the most relevant descriptors.
* **Model Selection:** Linear regression is a very simple model. You might consider using more advanced models, such as random forests, support vector machines, or neural networks.
* **Validation:** Proper validation techniques, such as cross-validation, are essential to ensure that your model generalizes well to new data.
* **Units:** Always be mindful of the units of your activity data (e.g., nM, uM, mM). Convert all values to a consistent unit before performing analysis.
* **Data Size:** With only 100 rows, your model will be highly susceptible to overfitting. Increase the data size when possible (but start with a smaller dataset for initial debugging, as you are doing).

**4. Five Examples (Based on Different Activity Types and Descriptors)**

Here are five variations of the code, modifying the activity type and adding other descriptors:

**Example 1: Ki Values and LogP**

* **SQL:** WHERE act.standard\_type = 'Ki' ...
* **Python:** Add a function to calculate LogP (octanol-water partition coefficient) and use both LogP and molecular weight as features.

from rdkit.Chem import AllChem  
  
def calculate\_logp(smiles):  
 mol = Chem.MolFromSmiles(smiles)  
 if mol:  
 return Descriptors.MolLogP(mol)  
 else:  
 return None  
  
df['logp'] = df['canonical\_smiles'].apply(calculate\_logp)  
df = df.dropna(subset=['logp'])  
  
X = df[['molecular\_weight', 'logp']]

**Example 2: EC50 Values and Number of Hydrogen Bond Donors**

* **SQL:** WHERE act.standard\_type = 'EC50' ...
* **Python:** Add a function to calculate the number of hydrogen bond donors and use it as a feature.

def calculate\_hbd(smiles):  
 mol = Chem.MolFromSmiles(smiles)  
 if mol:  
 return Descriptors.NumHDonors(mol)  
 else:  
 return None  
  
df['hbd'] = df['canonical\_smiles'].apply(calculate\_hbd)  
df = df.dropna(subset=['hbd'])  
  
X = df[['molecular\_weight', 'hbd']]

**Example 3: IC50 Values and TPSA (Topological Polar Surface Area)**

* **SQL:** WHERE act.standard\_type = 'IC50' ...
* **Python:** Calculate TPSA.

def calculate\_tpsa(smiles):  
 mol = Chem.MolFromSmiles(smiles)  
 if mol:  
 return Descriptors.TPSA(mol)  
 else:  
 return None  
  
df['tpsa'] = df['canonical\_smiles'].apply(calculate\_tpsa)  
df = df.dropna(subset=['tpsa'])  
  
X = df[['molecular\_weight', 'tpsa']]

**Example 4: Filtering by Target and Using Morgan Fingerprints**

* **SQL:** Add a JOIN to the target\_dictionary table and filter by target\_chembl\_id.

SELECT  
 act.molregno,  
 act.standard\_value,  
 act.standard\_units,  
 act.assay\_id,  
 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  
JOIN  
 assays a ON act.assay\_id = a.assay\_id  
JOIN  
 target\_dictionary td ON a.tid = td.tid  
WHERE  
 act.standard\_type = 'IC50'  
 AND act.standard\_relation = '='  
 AND act.standard\_value IS NOT NULL  
 AND act.standard\_value::TEXT ~ '^[0-9\.]+$'  
 AND td.chembl\_id = 'CHEMBL205' -- Replace with the target ChEMBL ID you want  
LIMIT 100;

* **Python:** Use Morgan fingerprints as features (much more sophisticated than just molecular weight). This requires more extensive changes to the Python code.

def calculate\_morgan\_fingerprint(smiles, radius=2, nBits=2048):  
 mol = Chem.MolFromSmiles(smiles)  
 if mol:  
 fp = AllChem.GetMorganFingerprintAsBitVect(mol, radius, nBits=nBits)  
 return np.array(list(fp.ToBitString()), dtype=int) # Convert to a numpy array of integers  
 else:  
 return None  
  
df['morgan\_fp'] = df['canonical\_smiles'].apply(calculate\_morgan\_fingerprint)  
df = df.dropna(subset=['morgan\_fp'])  
  
# Create a matrix of Morgan fingerprints. This is a crucial step.  
X = np.vstack(df['morgan\_fp'].values) # Stack the arrays vertically  
  
#Now you will have a different shape of X and will need to adjust  
# the test size and random state for your need  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

**Example 5: Filtering by Assay and Using a Different Model (Random Forest)**

* **SQL:** Add a filter for a specific assay\_id.

WHERE  
 act.standard\_type = 'IC50'  
 AND act.standard\_relation = '='  
 AND act.standard\_value IS NOT NULL  
 AND act.standard\_value::TEXT ~ '^[0-9\.]+$'  
 AND act.assay\_id = 669233 -- Replace with your assay ID.  
LIMIT 100;

* **Python:** Use a Random Forest Regressor instead of Linear Regression.

from sklearn.ensemble import RandomForestRegressor  
  
# ... (rest of the code is similar, but replace the LinearRegression part)  
  
model = RandomForestRegressor(n\_estimators=100, random\_state=42) #n\_estimators: number of trees  
model.fit(X\_train, y\_train)  
  
# Predictions and Evaluation  
y\_pred = model.predict(X\_test)  
  
mse = mean\_squared\_error(y\_test, y\_pred)  
r2 = r2\_score(y\_test, y\_pred)

**Key Improvements and Considerations:**

* **Error Handling:** The Python code includes basic error handling for file loading and invalid SMILES strings. Add more robust error handling for a production environment.
* **Data Exploration:** Before building models, perform thorough exploratory data analysis (EDA) to understand the data distribution, identify outliers, and check for data quality issues.
* **Feature Scaling:** Consider scaling your features (e.g., using StandardScaler or MinMaxScaler from scikit-learn) before training your model, especially if you are using models that are sensitive to feature scaling (e.g., Support Vector Machines).
* **Hyperparameter Tuning:** Optimize the hyperparameters of your model using techniques like cross-validation and grid search or random search.
* **Reproducibility:** Use random\_state in your code to ensure that your results are reproducible.
* **Documentation:** Add comments to your code to explain what each part of the code does.
* **AIMLops:** Remember to integrate these scripts into a proper pipeline for repeatability, version control, and automated deployment.

Remember to adapt these examples to your specific research question and the available data in ChEMBL 35. This comprehensive guide will help you get started with your analysis. Good luck!