# PharmApp Suite

## 🧠 AI for Drug Discovery and Development 🧪

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

## 🧩 Topic: Finding natural product-like molecules

Okay, I understand. You’re working with ChEMBL 35 data in a well-organized AIMLops project, leveraging RDKit and PostgreSQL. You’re facing some specific errors and want a clear guide with SQL and Python code examples. Let’s break this down.

**1. Analysis of the Model**

The project aims to analyze ChEMBL 35 data, likely focusing on structure-activity relationships (SAR). You’re extracting data via SQL, processing it with RDKit in Python, and likely building a model (implied by using mean\_squared\_error). The goal is to connect chemical structures (SMILES strings from ChEMBL) to activity data (IC50, Ki, etc.). This allows you to identify key features that drive activity, design better drugs, and predict the activity of new compounds.

* **SQL:** Extracts relevant data from the ChEMBL database. Crucial for filtering and preparing data for analysis.
* **RDKit:** Converts SMILES strings into numerical descriptors (e.g., fingerprints, molecular weight, logP) that machine learning models can understand. This is called featurization.
* **Machine Learning (likely):** The models learn relationships between the RDKit descriptors and the activity values. Regression models (like linear regression, random forest, or neural networks) are typically used to predict activity.

The errors suggest issues with data type handling in SQL and scikit-learn version incompatibilities. Let’s address those.

**2. Code and Guidance**

**Folder Structure (Based on AIMLops Template)**

project\_root/  
├── data/ # Stores data (CSV files extracted from ChEMBL)  
├── notebooks/ # Jupyter notebooks with analysis code  
│ └── Topic\_CheMBL\_35\_16\_1\_data\_extraction\_and\_preparation.ipynb  
│ └── Topic\_CheMBL\_35\_16\_2\_model\_building\_and\_evaluation.ipynb  
├── src/ # (Optional) Python modules for reusable code  
├── models/ # (Optional) Saved machine learning models  
├── requirements.txt # Python dependencies  
└── README.md

**2.1 SQL Code (for PostgreSQL) - ../data/chembl35\_data.csv**

-- Assuming you want activity data (IC50, Ki, etc.) and associated compound structures.  
  
-- Limit the number of results to 100 for demonstration purposes.  
  
SELECT  
 cmp.chembl\_id,  
 cmp.pref\_name,  
 act.standard\_type,  
 act.standard\_value,  
 act.standard\_units,  
 act.pchembl\_value,  
 mol.molfile  
FROM  
 activities act  
JOIN  
 molecule\_dictionary cmp ON act.molregno = cmp.molregno  
JOIN  
 compound\_structures mol ON cmp.molregno = mol.molregno  
WHERE  
 act.standard\_type IN ('IC50', 'Ki', 'EC50') -- Common activity types  
 AND act.standard\_units = 'nM' -- Focus on nanomolar activity  
 AND act.standard\_value IS NOT NULL -- Ensure we have activity values  
 -- Corrected regex for numeric values, casting to TEXT for regex matching  
 AND CAST(act.standard\_value AS TEXT) ~ '^[0-9.]+$'  
 AND mol.molfile IS NOT NULL --Ensure we have structures  
LIMIT 100;

**Explanation of SQL Code:**

* **SELECT ... FROM ... JOIN ... ON ...**: Standard SQL for selecting columns from multiple tables (activities, molecule\_dictionary, compound\_structures) and joining them based on common identifiers (molregno).
* **WHERE ...**: Filters the data:
  + act.standard\_type IN ('IC50', 'Ki', 'EC50'): Selects only common activity types.
  + act.standard\_units = 'nM': Restricts to activities measured in nanomolar units.
  + act.standard\_value IS NOT NULL: Excludes entries with missing activity values.
  + CAST(act.standard\_value AS TEXT) ~ '^[0-9.]+$' **(Fixes Error A)**: This addresses the error ERROR: operator does not exist: numeric ~ unknown. The ~ operator is used for regular expression matching in PostgreSQL. The issue was that it wasn’t implicitly converting the standard\_value (likely a numeric type) to text. CAST(act.standard\_value AS TEXT) explicitly converts the numeric column to text *before* applying the regular expression. The regular expression '^[0-9.]+$' checks if the value consists only of digits and periods.
  + mol.molfile IS NOT NULL: Ensure we have structure
* **LIMIT 100**: Restricts the result set to 100 rows. Remove this line for the full dataset.

**Important:**

* **molfile vs. smiles:** I used molfile because it’s guaranteed to be present, but you might prefer mol.smiles. Adjust the SELECT statement accordingly if you switch to mol.smiles.
* **Target Selection:** You likely want to add another WHERE clause to filter by a specific ChEMBL target (e.g., a specific protein). You’ll need to identify the target\_chembl\_id from the target\_dictionary table and join it to the activities table via target\_id.

**How to Run the SQL:**

1. Open pgAdmin and connect to your database (192.168.206.136, user rd, password rd, database chembl\_35).
2. Open a new query window.
3. Paste the SQL code into the query window.
4. Execute the query.
5. Export the results as a CSV file named chembl35\_data.csv and save it to the data/ directory of your project. pgAdmin has a convenient “Copy with Headers” option which you can then paste into a text file and save as CSV.

**2.2 Python Code (Jupyter Notebooks)**

Here’s the Python code, split into two notebooks as per your AIMLops structure.

**Notebook 1: Topic\_CheMBL\_35\_16\_1\_data\_extraction\_and\_preparation.ipynb**

import os  
import pandas as pd  
from rdkit import Chem  
from rdkit.Chem import AllChem  
from rdkit.Chem import Descriptors  
import numpy as np  
  
# Define the base path for the project (adjust if needed)  
base\_path = "./" # Assuming the notebook is in the notebooks/ directory  
  
# Construct the path to the CSV file  
data\_file\_path = os.path.join(base\_path, "data", "chembl35\_data.csv")  
  
# Load the data  
try:  
 df = pd.read\_csv(data\_file\_path)  
 print(f"Data loaded successfully from: {data\_file\_path}")  
except FileNotFoundError:  
 print(f"Error: File not found at {data\_file\_path}. Make sure the CSV file exists.")  
 exit()  
  
# Display the first few rows of the DataFrame  
print(df.head())  
  
# Function to convert molfile to smiles  
def molfile\_to\_smiles(molfile):  
 try:  
 mol = Chem.MolFromMolBlock(molfile)  
 if mol is not None:  
 return Chem.MolToSmiles(mol)  
 else:  
 return None  
 except:  
 return None  
  
# Apply the molfile to smiles function to the dataframe  
df['smiles'] = df['molfile'].apply(molfile\_to\_smiles)  
  
# Drop rows where SMILES conversion failed  
df = df.dropna(subset=['smiles'])  
  
# Basic Data Cleaning and Preparation  
# Convert standard\_value to numeric (handling potential errors)  
df['standard\_value'] = pd.to\_numeric(df['standard\_value'], errors='coerce')  
  
# Drop rows where standard\_value is NaN after conversion  
df = df.dropna(subset=['standard\_value'])  
  
# Apply pIC50 transformation function  
def calculate\_pic50(standard\_value):  
 pIC50 = 9 - np.log10(standard\_value)  
 return pIC50  
  
df['pIC50'] = df['standard\_value'].apply(calculate\_pic50)  
  
# Display updated DataFrame information  
print(df.info())  
print(df.head())  
  
# Define a function to calculate RDKit descriptors  
def calculate\_rdkit\_descriptors(smiles):  
 try:  
 mol = Chem.MolFromSmiles(smiles)  
 if mol is None:  
 return None  
 descriptors = {}  
 descriptors['MW'] = Descriptors.MolWt(mol)  
 descriptors['LogP'] = Descriptors.MolLogP(mol)  
 descriptors['HBA'] = Descriptors.NumHAcceptors(mol)  
 descriptors['HBD'] = Descriptors.NumHDonors(mol)  
 descriptors['TPSA'] = Descriptors.TPSA(mol)  
 return descriptors  
 except:  
 return None  
  
# Apply the descriptor calculation function to each SMILES string  
df['descriptors'] = df['smiles'].apply(calculate\_rdkit\_descriptors)  
  
# Handle missing descriptors  
df = df.dropna(subset=['descriptors'])  
  
# Convert descriptor dictionary to columns  
df = pd.concat([df.drop(['descriptors'], axis=1), df['descriptors'].apply(pd.Series)], axis=1)  
  
# Save processed data to a new CSV file  
processed\_data\_path = os.path.join(base\_path, "data", "chembl35\_processed\_data.csv")  
df.to\_csv(processed\_data\_path, index=False)  
  
print(f"Processed data saved to: {processed\_data\_path}")

**Notebook 2: Topic\_CheMBL\_35\_16\_2\_model\_building\_and\_evaluation.ipynb**

import os  
import pandas as pd  
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  
import matplotlib.pyplot as plt  
  
# Define the base path  
base\_path = "./"  
  
# Construct the path to the processed CSV file  
processed\_data\_path = os.path.join(base\_path, "data", "chembl35\_processed\_data.csv")  
  
# Load the processed data  
try:  
 df = pd.read\_csv(processed\_data\_path)  
 print(f"Processed data loaded successfully from: {processed\_data\_path}")  
except FileNotFoundError:  
 print(f"Error: File not found at {processed\_data\_path}. Ensure the previous notebook was run successfully.")  
 exit()  
  
# Data Preparation for Modeling  
# Select features (descriptors) and target (pIC50)  
features = ['MW', 'LogP', 'HBA', 'HBD', 'TPSA'] # Example descriptors  
target = 'pIC50'  
  
# Drop rows with NaN values in selected features and target  
df = df.dropna(subset=features + [target])  
  
X = df[features]  
y = df[target]  
  
# 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)  
  
# Model Building and Training (Linear Regression)  
model = LinearRegression()  
model.fit(X\_train, y\_train)  
  
# Model Evaluation  
y\_pred = model.predict(X\_test)  
  
# Calculate performance metrics  
mse = mean\_squared\_error(y\_test, y\_pred)  
r2 = r2\_score(y\_test, y\_pred)  
  
print(f"Mean Squared Error: {mse}")  
print(f"R-squared: {r2}")  
  
# Scatter plot of predictions vs. actual values  
plt.scatter(y\_test, y\_pred)  
plt.xlabel("Actual pIC50")  
plt.ylabel("Predicted pIC50")  
plt.title("Actual vs. Predicted pIC50")  
plt.show()  
  
# Feature Importance (for Linear Regression)  
coefficients = model.coef\_  
feature\_importance = pd.DataFrame({'Feature': features, 'Coefficient': coefficients})  
feature\_importance = feature\_importance.sort\_values('Coefficient', ascending=False)  
print("\nFeature Importance:")  
print(feature\_importance)

**Explanation of Python Code:**

* **os.path.join(base\_path, ...)**: Constructs file paths correctly, relative to the project’s base directory.
* **pandas.read\_csv()**: Loads the CSV data into a Pandas DataFrame.
* **RDKit:** Used to process the SMILES strings and calculate molecular descriptors.
* **sklearn.model\_selection.train\_test\_split()**: Splits the data into training and testing sets for model evaluation.
* **sklearn.linear\_model.LinearRegression()**: Creates a linear regression model. You can easily swap this out for other models.
* **sklearn.metrics.mean\_squared\_error()**, **sklearn.metrics.r2\_score()**: Evaluates the model’s performance using Mean Squared Error (MSE) and R-squared.
* **sklearn.preprocessing.StandardScaler()**: Standardizes the features (important for linear models and some other models).
* **Data Cleaning:** Removing any cases that are none (nan values).

**Important Considerations:**

* **Error Handling:** The code includes try...except blocks to handle potential errors during file loading and SMILES processing.
* **Feature Selection:** I’ve chosen some basic RDKit descriptors (MW, LogP, HBA, HBD, TPSA) as examples. You’ll likely want to experiment with different descriptors and feature selection techniques.
* **Model Selection:** Linear regression is a simple model. Consider more complex models like Random Forest, Gradient Boosting, or Neural Networks.
* **Cross-Validation:** Use cross-validation techniques (e.g., sklearn.model\_selection.cross\_val\_score) for more robust model evaluation.
* **Data Scaling:** The StandardScaler is important for many machine learning algorithms, especially those that use distance-based calculations.
* **Reproducibility:** Use random\_state in train\_test\_split for reproducible results.
* **Install Libraries**: Make sure you install the libraries via pip install pandas rdkit scikit-learn matplotlib. You can save these packages in a requirements.txt for future use.

**3. Addressing the Errors**

* **Error A (SQL): ERROR: operator does not exist: numeric ~ unknown** - Fixed by casting the standard\_value to TEXT before using the regular expression operator ~. See the corrected SQL code above.
* **Error B (Python - Old scikit-learn): squared=False in mean\_squared\_error** - This is no longer a current error. You can remove this parameter. If it persists, upgrade your scikit-learn version (pip install -U scikit-learn).

**4. Example Results (Illustrative)**

After running the code, you might see output like this in your Jupyter Notebook:

**Notebook 1 Output (Data Preparation):**

Data loaded successfully from: ./data/chembl35\_data.csv  
 chembl\_id pref\_name standard\_type standard\_value standard\_units pchembl\_value \  
0 CHEMBL1953617 NaN Ki 50.000000 nM 7.30 ...  
1 CHEMBL422687 NaN Ki 50.000000 nM 7.30 ...  
2 CHEMBL422687 NaN IC50 70.794578 nM 7.15 ...  
3 CHEMBL1953617 NaN IC50 70.794578 nM 7.15 ...  
4 CHEMBL1953617 NaN IC50 10.000000 nM 8.00 ...  
  
 molfile  
0 \n RDKit 2D\n\n 9 9 0 0 0 0 ...  
1 \n RDKit 2D\n\n 9 9 0 0 0 0 ...  
2 \n RDKit 2D\n\n 9 9 0 0 0 0 ...  
3 \n RDKit 2D\n\n 9 9 0 0 0 0 ...  
4 \n RDKit 2D\n\n 9 9 0 0 0 0 ...  
  
<class 'pandas.core.frame.DataFrame'>  
RangeIndex: 100 entries, 0 to 99  
Data columns (total 8 columns):  
 chembl\_id 100 non-null object  
 pref\_name 19 non-null object  
 standard\_type 100 non-null object  
 standard\_value 100 non-null float64  
 standard\_units 100 non-null object  
 pchembl\_value 92 non-null float64  
 molfile 100 non-null object  
 smiles 100 non-null object  
dtypes: float64(2), object(6)  
memory usage: 6.3+ KB  
 chembl\_id pref\_name standard\_type standard\_value standard\_units pchembl\_value \  
0 CHEMBL1953617 NaN Ki 50.000000 nM 7.30 ...  
1 CHEMBL422687 NaN Ki 50.000000 nM 7.30 ...  
2 CHEMBL422687 NaN IC50 70.794578 nM 7.15 ...  
3 CHEMBL1953617 NaN IC50 70.794578 nM 7.15 ...  
4 CHEMBL1953617 NaN IC50 10.000000 nM 8.00 ...  
  
 molfile \  
0 \n RDKit 2D\n\n 9 9 0 0 0 0 ...  
1 \n RDKit 2D\n\n 9 9 0 0 0 0 ...  
2 \n RDKit 2D\n\n 9 9 0 0 0 0 ...  
3 \n RDKit 2D\n\n 9 9 0 0 0 0 ...  
4 \n RDKit 2D\n\n 9 9 0 0 0 0 ...  
  
 smiles  
0 Cc1ccccc1CN(Cc1ccccc1)C(=O)N1CCOCC1  
1 Cc1ccccc1CN(Cc1ccccc1)C(=O)N1CCOCC1  
2 Cc1ccccc1CN(Cc1ccccc1)C(=O)N1CCOCC1  
3 Cc1ccccc1CN(Cc1ccccc1)C(=O)N1CCOCC1  
4 Cc1ccccc1CN(Cc1ccccc1)C(=O)N1CCOCC1  
Processed data saved to: ./data/chembl35\_processed\_data.csv

**Notebook 2 Output (Model Building and Evaluation):**

Processed data loaded successfully from: ./data/chembl35\_processed\_data.csv  
Mean Squared Error: 0.8542176587247836  
R-squared: 0.3912347890564484  
  
Feature Importance:  
 Feature Coefficient  
2 HBA 0.448862  
4 TPSA 0.393045  
1 LogP -0.124559  
3 HBD -0.292158  
0 MW -0.351017

(And a scatter plot visualizing predicted vs. actual pIC50 values)

**5. Five Example Use Cases:**

1. **Lead Optimization:** Identify compounds with similar structures but differing activities to understand which modifications improve potency.
2. **Virtual Screening:** Predict the activity of a large library of virtual compounds before synthesizing them, prioritizing those with the highest predicted activity.
3. **Target Identification:** If you have activity data for a compound but don’t know its target, analyze its structural features to predict likely protein targets.
4. **Drug Repurposing:** Identify existing drugs that might be effective against a new target based on their structural similarity to known active compounds.
5. **ADMET Prediction:** Build models to predict Absorption, Distribution, Metabolism, Excretion, and Toxicity (ADMET) properties based on molecular structure, helping to filter out problematic compounds early in the drug discovery process.

Let me know if you’d like more detailed examples, different code snippets, or help with any specific part of the process. Good luck!