# PharmApp Suite

## 🧠 AI for Drug Discovery and Development 🧪

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

## 🧩 Topic: Mapping ChEMBL drugs to WHO ATC codes

Okay, I understand. You’re working on a Chembl 35 data analysis project using RDKit within an AIMLops framework, using a PostgreSQL database, and running your code in Jupyter notebooks. Let’s break down how to approach this, address the errors, and get you running with a focused workflow.

**1. Analysis Model and Goal**

Based on your description, it seems your primary goal is to extract and analyze data from the Chembl 35 database to support drug research and development. This likely involves:

* **Data Extraction (SQL):** Querying the database to retrieve relevant information about compounds, targets, activities, and other relevant data points.
* **Data Transformation and Cleaning (Python/RDKit):** Converting the data into usable formats, handling missing values, and cleaning up inconsistencies.
* **Feature Engineering (Python/RDKit):** Calculating relevant molecular properties and features using RDKit. This could include things like molecular weight, logP, topological polar surface area (TPSA), number of hydrogen bond donors/acceptors, and more.
* **Data Analysis and Modeling (Python/Scikit-learn, etc.):** Exploring relationships between features and activity, building predictive models for activity, and identifying promising compounds.
* **Visualization (Python/Matplotlib, Seaborn):** Creating plots and charts to visualize data distributions, relationships, and model performance.

**The focus on getting only 100 rows and fixing the errors suggests you’re aiming for a proof-of-concept or initial exploration phase.**

**2. Addressing the Errors**

* **Error a: ERROR: operator does not exist: numeric ~ unknown, LINE 12: AND act.standard\_value ~ '^[0-9\.]+$'**
* This error arises because you’re trying to use the regular expression operator ~ to match a pattern against a numeric data type in PostgreSQL. ~ is designed for string matching. The issue is act.standard\_value is probably defined as a numeric type (like integer or numeric).
* **Solution:** Cast the standard\_value to text before applying the regex.
* **Error b: old scikit-learn version does not support parameters squared=False in the mean\_squared\_error function**
* This is a version compatibility issue. The squared=False parameter was introduced in a later version of scikit-learn.
* **Solution:** There are two main approaches:
  1. **Upgrade Scikit-learn:** This is the best long-term solution. In your Jupyter notebook, try: !pip install scikit-learn --upgrade
  2. **Calculate RMSE Manually:** If upgrading isn’t feasible (due to environment constraints), you can calculate the Root Mean Squared Error (RMSE) manually:
  + from sklearn.metrics import mean\_squared\_error  
    import numpy as np  
      
    mse = mean\_squared\_error(y\_true, y\_pred)  
    rmse = np.sqrt(mse)

**3. Code Examples (SQL and Python)**

Here’s a structured approach with code examples. I will aim for clarity, efficiency, and addressing the errors.

import os  
import pandas as pd  
import psycopg2 # For PostgreSQL connection  
from rdkit import Chem  
from rdkit.Chem import Descriptors  
from sklearn.model\_selection import train\_test\_split  
from sklearn.linear\_model import LinearRegression  
from sklearn.metrics import mean\_squared\_error  
import numpy as np  
  
# --- DEFINE PATHS (AIMLops Style) ---  
base\_path = os.getcwd() # Or your root directory  
data\_path = os.path.join(base\_path, "data")  
notebook\_path = os.path.join(base\_path, "notebooks") # Assuming 'notebooks' dir  
model\_path = os.path.join(base\_path, "models")  
  
# Ensure directories exist (or create them)  
os.makedirs(data\_path, exist\_ok=True)  
os.makedirs(notebook\_path, exist\_ok=True)  
os.makedirs(model\_path, exist\_ok=True)  
  
# --- DATABASE CONNECTION DETAILS ---  
db\_host = "192.168.206.136"  
db\_user = "rd"  
db\_pass = "rd"  
db\_name = "chembl\_35"  
  
# --- SQL QUERY ---  
sql\_query = """  
SELECT  
 act.molregno,  
 act.standard\_value,  
 act.standard\_units,  
 act.pchembl\_value,  
 md.chembl\_id,  
 md.canonical\_smiles  
FROM  
 activities act  
JOIN  
 molecule\_dictionary md ON act.molregno = md.molregno  
WHERE  
 act.standard\_type = 'IC50'  
 AND act.standard\_units = 'nM'  
 AND act.standard\_value IS NOT NULL  
 AND act.standard\_value::text ~ '^[0-9\.]+$' -- Corrected Regex Error  
LIMIT 100;  
"""  
  
# --- PYTHON CODE (Jupyter Notebook - Topic\_CheMBL\_35\_45\_1\_Data\_Extraction.ipynb) ---  
try:  
 conn = psycopg2.connect(host=db\_host, user=db\_user, password=db\_pass, database=db\_name)  
 df = pd.read\_sql\_query(sql\_query, conn)  
 conn.close()  
  
 print("Data extracted successfully:")  
 print(df.head())  
  
 # Save to CSV  
 csv\_file\_path = os.path.join(data\_path, "chembl\_ic50\_data.csv")  
 df.to\_csv(csv\_file\_path, index=False)  
 print(f"Data saved to: {csv\_file\_path}")  
  
  
except Exception as e:  
 print(f"Error connecting to the database or extracting data: {e}")

# --- PYTHON CODE (Jupyter Notebook - Topic\_CheMBL\_35\_45\_2\_Feature\_Engineering\_Modeling.ipynb) ---  
# Feature Engineering and Modeling  
import pandas as pd  
import os  
from rdkit import Chem  
from rdkit.Chem import Descriptors  
from sklearn.model\_selection import train\_test\_split  
from sklearn.linear\_model import LinearRegression  
from sklearn.metrics import mean\_squared\_error  
import numpy as np  
  
# --- DEFINE PATHS (AIMLops Style) ---  
base\_path = os.getcwd() # Or your root directory  
data\_path = os.path.join(base\_path, "data")  
notebook\_path = os.path.join(base\_path, "notebooks") # Assuming 'notebooks' dir  
model\_path = os.path.join(base\_path, "models")  
  
# --- Load the data ---  
csv\_file\_path = os.path.join(data\_path, "chembl\_ic50\_data.csv")  
try:  
 df = pd.read\_csv(csv\_file\_path)  
 print("Data loaded successfully.")  
except FileNotFoundError:  
 print(f"Error: File not found at {csv\_file\_path}")  
 exit()  
  
# --- Feature Engineering (RDKit) ---  
def calculate\_descriptors(smiles):  
 mol = Chem.MolFromSmiles(smiles)  
 if mol is None:  
 return None # Handle invalid SMILES  
 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)  
 return descriptors  
  
# Apply the function to create new columns  
df['descriptors'] = df['canonical\_smiles'].apply(calculate\_descriptors)  
  
# Convert descriptors to separate columns  
df = pd.concat([df, df['descriptors'].apply(pd.Series)], axis=1)  
df.drop('descriptors', axis=1, inplace=True)  
  
# Drop rows with NaN values in the descriptor columns (due to invalid SMILES)  
df = df.dropna(subset=['MolWt', 'LogP', 'HBD', 'HBA', 'TPSA'])  
  
# Convert IC50 to pIC50 (if pIC50 is not already present)  
if 'pchembl\_value' not in df.columns:  
 df['pIC50'] = -np.log10(df['standard\_value'] \* 1e-9) # Convert nM to M  
  
# --- Data Preparation for Modeling ---  
X = df[['MolWt', 'LogP', 'HBD', 'HBA', 'TPSA']] # Feature matrix  
y = df['pIC50'] # Target variable  
  
# Split data into training and testing sets  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)  
  
# --- Model Training ---  
model = LinearRegression()  
model.fit(X\_train, y\_train)  
  
# --- Model Evaluation ---  
y\_pred = model.predict(X\_test)  
  
# Calculate RMSE (handling the potential scikit-learn version issue)  
mse = mean\_squared\_error(y\_test, y\_pred)  
rmse = np.sqrt(mse) #Calculate RMSE manually  
  
print(f"Root Mean Squared Error (RMSE): {rmse}")  
  
# --- Save the Model (Optional) ---  
import joblib  
model\_file\_path = os.path.join(model\_path, "linear\_regression\_model.pkl")  
joblib.dump(model, model\_file\_path)  
print(f"Model saved to: {model\_file\_path}")

**Explanation:**

* **File Organization:** The code uses os.path.join to create paths according to your AIMLops structure, ensuring portability and reproducibility.
* **SQL Query:** The SQL query selects relevant data, filters for IC50 values in nM, and addresses the regex error by casting standard\_value to TEXT before applying the regular expression. The LIMIT 100 clause ensures you only retrieve 100 rows.
* **Data Loading and Cleaning:** The Python code loads the data, handles potential errors (like FileNotFoundError), and performs initial cleaning (dropping rows with missing descriptor values).
* **Feature Engineering (RDKit):** The calculate\_descriptors function uses RDKit to calculate key molecular properties. Error handling is included for invalid SMILES strings.
* **pIC50 Conversion:** If pchembl\_value is not available, the code converts IC50 values to pIC50.
* **Model Training and Evaluation:** A simple linear regression model is trained and evaluated using RMSE. The RMSE calculation is done manually to avoid the scikit-learn squared=False issue.
* **Model Persistence (Optional):** The trained model is saved using joblib for later use.

**4. 5 Examples with Variations**

Here are 5 examples demonstrating how you can modify the code:

**Example 1: Different Activity Type**

* **Goal:** Extract data for Ki (inhibition constant) instead of IC50.
* **SQL Modification:**
* SELECT  
   act.molregno,  
   act.standard\_value,  
   act.standard\_units,  
   act.pchembl\_value,  
   md.chembl\_id,  
   md.canonical\_smiles  
  FROM  
   activities act  
  JOIN  
   molecule\_dictionary md ON act.molregno = md.molregno  
  WHERE  
   act.standard\_type = 'Ki' -- Changed to Ki  
   AND act.standard\_units = 'nM'  
   AND act.standard\_value IS NOT NULL  
   AND act.standard\_value::text ~ '^[0-9\.]+$'  
  LIMIT 100;
* **No Python code changes are needed if only data is being extracted.**

**Example 2: Different Unit**

* **Goal:** Extract data for IC50 in uM (micromolar)
* **SQL Modification:** sql SELECT act.molregno, act.standard\_value, act.standard\_units, act.pchembl\_value, md.chembl\_id, md.canonical\_smiles FROM activities act JOIN molecule\_dictionary md ON act.molregno = md.molregno WHERE act.standard\_type = 'IC50' AND act.standard\_units = 'uM' -- Changed to uM AND act.standard\_value IS NOT NULL AND act.standard\_value::text ~ '^[0-9\.]+$' LIMIT 100;
* **Python Modification:** The conversion to pIC50 needs to be adjusted to account for the unit change. python if 'pchembl\_value' not in df.columns: df['pIC50'] = -np.log10(df['standard\_value'] \* 1e-6) # Convert uM to M

**Example 3: Adding More Descriptors**

* **Goal:** Calculate more molecular descriptors.
* **Python Modification (in calculate\_descriptors function):**
* def calculate\_descriptors(smiles):  
   mol = Chem.MolFromSmiles(smiles)  
   if mol is None:  
   return None # Handle invalid SMILES  
   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)  
   descriptors["RotatableBonds"] = Descriptors.NumRotatableBonds(mol) #Added  
   descriptors["AromaticRings"] = Descriptors.NumAromaticRings(mol) #Added  
   return descriptors
* You’ll also need to update the X variable to include the new descriptors:
* X = df[['MolWt', 'LogP', 'HBD', 'HBA', 'TPSA', 'RotatableBonds', 'AromaticRings']]

**Example 4: Using a Different Model**

* **Goal:** Use a Random Forest Regressor instead of Linear Regression.
* **Python Modification:**
* from sklearn.ensemble import RandomForestRegressor  
    
  # --- Model Training ---  
  model = RandomForestRegressor(n\_estimators=100, random\_state=42) #Example parameters  
  model.fit(X\_train, y\_train)

**Example 5: Target Specificity by adding target information**

* **Goal:** Include target name and accession in your data
* **SQL Modification:** sql SELECT act.molregno, act.standard\_value, act.standard\_units, act.pchembl\_value, md.chembl\_id, md.canonical\_smiles, td.target\_name, /\* added \*/ td.accession /\* added \*/ FROM activities act JOIN molecule\_dictionary md ON act.molregno = md.molregno JOIN target\_dictionary td ON act.tid = td.tid /\* added \*/ WHERE act.standard\_type = 'IC50' AND act.standard\_units = 'nM' AND act.standard\_value IS NOT NULL AND act.standard\_value::text ~ '^[0-9\.]+$' LIMIT 100;

**In summary:**

I have provided a complete walkthrough of your project, from SQL data extraction to Python-based feature engineering, model building, and error handling. I’ve also given you 5 examples to help you adapt the code to your specific needs. Let me know if you have any more questions.

**Traducción al Español (Spanish Translation):**

**1. Análisis del Modelo y Objetivo:**

Basado en su descripción, parece que su objetivo principal es extraer y analizar datos de la base de datos Chembl 35 para respaldar la investigación y el desarrollo de fármacos. Esto probablemente implica:

* **Extracción de datos (SQL):** Consultar la base de datos para recuperar información relevante sobre compuestos, objetivos, actividades y otros puntos de datos relevantes.
* **Transformación y limpieza de datos (Python/RDKit):** Convertir los datos en formatos utilizables, manejar los valores faltantes y limpiar las inconsistencias.
* **Ingeniería de características (Python/RDKit):** Calcular propiedades y características moleculares relevantes utilizando RDKit. Esto podría incluir cosas como el peso molecular, logP, el área de superficie polar topológica (TPSA), el número de donantes/aceptores de enlaces de hidrógeno y más.
* **Análisis y modelado de datos (Python/Scikit-learn, etc.):** Explorar las relaciones entre las características y la actividad, construir modelos predictivos para la actividad e identificar compuestos prometedores.
* **Visualización (Python/Matplotlib, Seaborn):** Crear gráficos y diagramas para visualizar las distribuciones de datos, las relaciones y el rendimiento del modelo.

**El enfoque en obtener solo 100 filas y corregir los errores sugiere que está apuntando a una prueba de concepto o una fase de exploración inicial.**

**2. Abordar los errores:**

* **Error a: ERROR: operator does not exist: numeric ~ unknown, LINE 12: AND act.standard\_value ~ '^[0-9\.]+$'**
* Este error surge porque está intentando utilizar el operador de expresión regular ~ para comparar un patrón con un tipo de datos numeric en PostgreSQL. ~ está diseñado para la coincidencia de cadenas. El problema es que act.standard\_value probablemente se defina como un tipo numérico (como integer o numeric).
* **Solución:** Convierta standard\_value a texto antes de aplicar la expresión regular.
* **Error b: old scikit-learn version does not support parameters squared=False in the mean\_squared\_error function**
* Este es un problema de compatibilidad de versiones. El parámetro squared=False se introdujo en una versión posterior de scikit-learn.
* **Solución:** Hay dos enfoques principales:
  1. **Actualizar Scikit-learn:** Esta es la mejor solución a largo plazo. En su cuaderno Jupyter, pruebe: !pip install scikit-learn --upgrade
  2. **Calcular RMSE manualmente:** Si la actualización no es factible (debido a las limitaciones del entorno), puede calcular el error cuadrático medio de la raíz (RMSE) manualmente:
  + from sklearn.metrics import mean\_squared\_error  
    import numpy as np  
      
    mse = mean\_squared\_error(y\_true, y\_pred)  
    rmse = np.sqrt(mse)

**3. Ejemplos de código (SQL y Python)**

Aquí hay un enfoque estructurado con ejemplos de código. Apunto a la claridad, la eficiencia y la resolución de los errores. (Ve el codigo completo arriba)

**4. 5 Ejemplos con Variaciones**

Aquí hay 5 ejemplos que demuestran cómo puede modificar el código: (Ve los ejemplos completos arriba)

En resumen, he proporcionado un recorrido completo de su proyecto, desde la extracción de datos SQL hasta la ingeniería de características basada en Python, la construcción de modelos y el manejo de errores. También le he dado 5 ejemplos para ayudarle a adaptar el código a sus necesidades específicas. Avísame si tienes más preguntas.