# PharmApp Suite

## 🧠 AI for Drug Discovery and Development 🧪

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

## 🧩 Topic: Streamlit dashboard for bioactivity exploration

Okay, I understand. You’re working on a Chembl 35 data analysis project using RDKit and PostgreSQL, following an AIMLops template. You’re experiencing some errors and want guidance on how to proceed, with a focus on retrieving a limited dataset of 100 rows for faster experimentation.

Let’s break down the analysis model, provide the code, address the errors, and create some examples.

**1. Analysis Model:**

The core of your project involves analyzing chemical and biological data from the ChEMBL database (version 35). This data likely includes information about:

* **Chemical Compounds:** Represented by SMILES strings or other chemical identifiers.
* **Bioactivity Data:** Measurements like IC50, Ki, Kd values, indicating how well a compound interacts with a target.
* **Targets:** Proteins or other biological entities that the compounds are designed to interact with.

Your analysis goals likely include:

* **Data Cleaning and Preparation:** Handling missing data, standardizing units, filtering for relevant data types (e.g., focusing on IC50 values).
* **Feature Engineering:** Generating molecular descriptors from the SMILES strings using RDKit. These descriptors capture various properties of the molecules (e.g., molecular weight, LogP, number of rings, etc.).
* **Exploratory Data Analysis (EDA):** Visualizing the data to understand distributions, identify outliers, and look for correlations between features and bioactivity.
* **Model Building:** Developing predictive models to estimate the bioactivity of new compounds based on their molecular descriptors. This could involve regression models (e.g., linear regression, random forests, support vector machines) or classification models (e.g., predicting whether a compound is active or inactive).
* **Model Evaluation:** Assessing the performance of the models using appropriate metrics (e.g., R-squared, RMSE for regression; accuracy, precision, recall, F1-score for classification).

**Addressing Errors:**

* **Error a: ERROR: operator does not exist: numeric ~ unknown, LINE 12: AND act.standard\_value ~ '^[0-9\.]+$'**
* This error means that you’re trying to use a regular expression operator (~) on a numeric column (act.standard\_value). PostgreSQL requires the column to be of a text type for regex matching. You need to cast the numeric column 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. Either update your scikit-learn library to a version that supports squared=False (recommended) or remove the squared=False argument. Removing it will return the Mean Squared Error (MSE) instead of the Root Mean Squared Error (RMSE).

**2. SQL and Python Code:**

Here’s the code, incorporating the fixes and limitations you specified:

**SQL (Topic\_CheMBL\_35\_71.sql - to be run in pgAdmin):**

-- Select 100 rows of data from ChEMBL, focusing on IC50 values for a specific target.  
-- Adjust the WHERE clause to target a specific protein or biological process if needed.  
-- The limit clause is used to retrieve 100 rows for faster experimentation.  
  
SELECT  
 md.molregno,  
 cs.canonical\_smiles,  
 act.standard\_value  
FROM  
 compound\_structures cs  
JOIN  
 molecule\_dictionary md ON cs.molregno = md.molregno  
JOIN  
 activities act ON md.molregno = act.molregno  
JOIN  
 assay\_xref ax ON act.assay\_id = ax.assay\_id  
JOIN  
 assays a ON ax.assay\_id = a.assay\_id  
WHERE  
 act.standard\_type = 'IC50' -- Focus on IC50 values  
 AND act.standard\_units = 'nM' -- Ensure units are consistent  
 AND act.standard\_relation = '=' --Only get equal relation  
 AND act.standard\_value IS NOT NULL  
 AND a.confidence\_score >= 8 -- High Confidence Data  
 AND CAST(act.standard\_value AS TEXT) ~ '^[0-9\.]+$' -- Fix: Cast to text for regex  
LIMIT 100;

**Explanation:**

* SELECT ... FROM ... JOIN ...: Selects data from several ChEMBL tables and joins them based on common keys. We’re selecting molregno (molecule registry number), canonical\_smiles (SMILES string), and standard\_value (IC50 value).
* WHERE act.standard\_type = 'IC50': Filters for IC50 values. You can change this to other bioactivity types if needed.
* WHERE act.standard\_units = 'nM': Filters for nanomolar units to ensure consistency.
* WHERE a.confidence\_score >= 8: Filters for high-confidence data. Adjust this score based on your data quality requirements.
* AND CAST(act.standard\_value AS TEXT) ~ '^[0-9\.]+$': This is the crucial fix. It casts the standard\_value column to TEXT before applying the regular expression to ensure that the value only consists of numbers and periods.
* LIMIT 100: Limits the result set to 100 rows.

**Python Code (Topic\_CheMBL\_35\_71\_1\_Data\_Analysis.ipynb):**

import os  
import pandas as pd  
import psycopg2  
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 the project  
base\_path = "." # Assuming the notebook is in the root of your project  
  
# Database credentials  
db\_host = "192.168.206.136"  
db\_user = "rd"  
db\_pass = "rd"  
db\_name = "chembl\_35"  
  
# Function to connect to the database and execute a query  
def connect\_and\_query(sql\_query):  
 try:  
 conn = psycopg2.connect(host=db\_host, user=db\_user, password=db\_pass, database=db\_name)  
 cur = conn.cursor()  
 cur.execute(sql\_query)  
 results = cur.fetchall()  
 conn.close()  
 return results  
 except Exception as e:  
 print(f"Error connecting to the database: {e}")  
 return None  
  
# Function to load data from CSV (if you prefer that to querying directly)  
def load\_data\_from\_csv(csv\_file\_path):  
 try:  
 df = pd.read\_csv(csv\_file\_path)  
 return df  
 except FileNotFoundError:  
 print(f"Error: CSV file not found at {csv\_file\_path}")  
 return None  
 except Exception as e:  
 print(f"Error reading CSV file: {e}")  
 return None  
  
  
# SQL query to retrieve data  
sql\_query = """  
SELECT  
 md.molregno,  
 cs.canonical\_smiles,  
 act.standard\_value  
FROM  
 compound\_structures cs  
JOIN  
 molecule\_dictionary md ON cs.molregno = md.molregno  
JOIN  
 activities act ON md.molregno = act.molregno  
JOIN  
 assay\_xref ax ON act.assay\_id = ax.assay\_id  
JOIN  
 assays a ON ax.assay\_id = a.assay\_id  
WHERE  
 act.standard\_type = 'IC50' -- Focus on IC50 values  
 AND act.standard\_units = 'nM' -- Ensure units are consistent  
 AND act.standard\_relation = '=' --Only get equal relation  
 AND act.standard\_value IS NOT NULL  
 AND a.confidence\_score >= 8 -- High Confidence Data  
 AND CAST(act.standard\_value AS TEXT) ~ '^[0-9\.]+$' -- Fix: Cast to text for regex  
LIMIT 100;  
"""  
  
# Option 1: Query the database directly  
results = connect\_and\_query(sql\_query)  
if results:  
 df = pd.DataFrame(results, columns=['molregno', 'smiles', 'ic50'])  
else:  
 df = None # Handle the error appropriately  
  
# Option 2: Load from CSV (alternative if you ran the SQL query in pgAdmin)  
# data\_file\_path = os.path.join(base\_path, "data", "chembl\_data.csv") #Replace "chembl\_data.csv" with your file name  
# df = load\_data\_from\_csv(data\_file\_path)  
  
if df is not None:  
 print("Data loaded successfully:")  
 print(df.head())  
  
 # Data cleaning: Convert IC50 to numeric and drop missing values  
 df['ic50'] = pd.to\_numeric(df['ic50'], errors='coerce') # Convert to numeric  
 df = df.dropna(subset=['ic50', 'smiles']) # drop NA values if the data contain those values  
  
 # Feature Engineering with RDKit  
 def calculate\_descriptors(smiles):  
 mol = Chem.MolFromSmiles(smiles)  
 if mol is None:  
 return None  
 descriptors = {}  
 descriptors['MolWt'] = Descriptors.MolWt(mol)  
 descriptors['LogP'] = Chem.Crippen.MolLogP(mol)  
 descriptors['HBD'] = Chem.Lipinski.NumHDonors(mol)  
 descriptors['HBA'] = Chem.Lipinski.NumHAcceptors(mol)  
 descriptors['TPSA'] = Chem.rdMolDescriptors.CalcTPSA(mol)  
 return pd.Series(descriptors)  
  
 # Apply the function to create new columns  
 df = pd.concat([df, df['smiles'].apply(calculate\_descriptors)], axis=1)  
 df = df.dropna() # Drop any rows that have NA for some reason  
  
 # Prepare data for modeling  
 X = df[['MolWt', 'LogP', 'HBD', 'HBA', 'TPSA']] # Features  
 y = np.log10(df['ic50']) # Target variable (log-transformed IC50)  
  
 # 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)  
  
 # Train a linear regression 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) # No squared=False for older scikit-learn  
 r2 = r2\_score(y\_test, y\_pred)  
  
 print(f"Mean Squared Error: {mse}")  
 print(f"R-squared: {r2}")  
  
else:  
 print("Failed to load data.")

**Explanation:**

* **Imports:** Imports necessary libraries (pandas, psycopg2, RDKit, scikit-learn).
* **Database Connection:** connect\_and\_query function encapsulates the database connection logic, making it reusable. It takes an SQL query as input, executes it, and returns the results. Error handling is included.
* **CSV Loading:** load\_data\_from\_csv function will load your data from the file.
* **SQL Query:** Defines the SQL query (the same one from the SQL section).
* **Data Loading:** Uses either the database connection or CSV loading to get the data into a Pandas DataFrame.
* **Data Cleaning:** Converts the ic50 column to a numeric type and removes any rows with missing values in the ic50 or smiles columns.
* **Feature Engineering:**
  + The calculate\_descriptors function takes a SMILES string as input, converts it to an RDKit molecule object, and calculates several molecular descriptors (Molecular Weight, LogP, Hydrogen Bond Donors, Hydrogen Bond Acceptors, TPSA).
  + The function is applied to the smiles column of the DataFrame to create new columns for each descriptor.
* **Data Preparation:**
  + Defines the features (X) and the target variable (y). The target variable is log-transformed to potentially improve the distribution and model performance.
  + StandardScaler is used to scale the features, which is often important for linear models.
* **Model Training:**
  + Splits the data into training and testing sets.
  + Creates a LinearRegression model, trains it on the training data, and makes predictions on the test data.
* **Model Evaluation:**
  + Calculates the Mean Squared Error (MSE) and R-squared to evaluate the model’s performance. Since you might be on an older scikit-learn version, squared=False is removed from mean\_squared\_error.

**3. AIMLops Template Integration:**

The code follows the AIMLops template by:

* Using os.path.join(base\_path, ...) to construct file paths relative to the project root. This makes the code more portable.
* Keeping the data loading and processing logic separate from the model training and evaluation.

**4. Example Usage (5 examples):**

Here are 5 examples of how you might use and extend this code:

**Example 1: Changing the Target**

# Change the SQL query to target a different standard\_type (e.g., 'Ki')  
sql\_query = """  
SELECT  
 md.molregno,  
 cs.canonical\_smiles,  
 act.standard\_value  
FROM  
 compound\_structures cs  
JOIN  
 molecule\_dictionary md ON cs.molregno = md.molregno  
JOIN  
 activities act ON md.molregno = act.molregno  
JOIN  
 assay\_xref ax ON act.assay\_id = ax.assay\_id  
JOIN  
 assays a ON ax.assay\_id = a.assay\_id  
WHERE  
 act.standard\_type = 'Ki' -- Changed to Ki  
 AND act.standard\_units = 'nM' -- Ensure units are consistent  
 AND act.standard\_relation = '=' --Only get equal relation  
 AND act.standard\_value IS NOT NULL  
 AND a.confidence\_score >= 8 -- High Confidence Data  
 AND CAST(act.standard\_value AS TEXT) ~ '^[0-9\.]+$' -- Fix: Cast to text for regex  
LIMIT 100;  
"""

**Example 2: Adding More Descriptors**

def calculate\_descriptors(smiles):  
 mol = Chem.MolFromSmiles(smiles)  
 if mol is None:  
 return None  
 descriptors = {}  
 descriptors['MolWt'] = Descriptors.MolWt(mol)  
 descriptors['LogP'] = Chem.Crippen.MolLogP(mol)  
 descriptors['HBD'] = Chem.Lipinski.NumHDonors(mol)  
 descriptors['HBA'] = Chem.Lipinski.NumHAcceptors(mol)  
 descriptors['TPSA'] = Chem.rdMolDescriptors.CalcTPSA(mol)  
 descriptors['NumRings'] = Chem.rdMolDescriptors.CalcNumRings(mol) # Add number of rings  
 return pd.Series(descriptors)  
  
# ... (rest of the code remains the same)  
  
X = df[['MolWt', 'LogP', 'HBD', 'HBA', 'TPSA', 'NumRings']] # Add NumRings to features

**Example 3: Using a Different Model**

from sklearn.ensemble import RandomForestRegressor  
  
# ... (rest of the code, including data preparation, remains the same)  
  
# Train a Random Forest Regression model  
model = RandomForestRegressor(n\_estimators=100, random\_state=42) # Example parameters  
model.fit(X\_train, y\_train)  
  
# ... (rest of the code, including prediction and evaluation, remains the same)

**Example 4: Target a specific Assay**

-- Select 100 rows of data from ChEMBL, focusing on IC50 values for a specific target.  
-- Adjust the WHERE clause to target a specific protein or biological process if needed.  
-- The limit clause is used to retrieve 100 rows for faster experimentation.  
  
SELECT  
 md.molregno,  
 cs.canonical\_smiles,  
 act.standard\_value  
FROM  
 compound\_structures cs  
JOIN  
 molecule\_dictionary md ON cs.molregno = md.molregno  
JOIN  
 activities act ON md.molregno = act.molregno  
JOIN  
 assay\_xref ax ON act.assay\_id = ax.assay\_id  
JOIN  
 assays a ON ax.assay\_id = a.assay\_id  
WHERE  
 act.standard\_type = 'IC50' -- Focus on IC50 values  
 AND act.standard\_units = 'nM' -- Ensure units are consistent  
 AND act.standard\_relation = '=' --Only get equal relation  
 AND act.standard\_value IS NOT NULL  
 AND a.confidence\_score >= 8 -- High Confidence Data  
 AND a.assay\_id = 1234 -- replace 1234 with the assay\_id of your target  
 AND CAST(act.standard\_value AS TEXT) ~ '^[0-9\.]+$' -- Fix: Cast to text for regex  
LIMIT 100;

**Example 5: Saving the Model**

import joblib # Or pickle  
  
# ... (rest of the code, including model training, remains the same)  
  
# Save the model  
model\_path = os.path.join(base\_path, "models", "linear\_regression\_model.joblib")  
joblib.dump(model, model\_path)  
print(f"Model saved to {model\_path}")  
  
# Later, you can load the model:  
# loaded\_model = joblib.load(model\_path)

**Summary in English:**

This comprehensive guide provides you with a clear analysis model, SQL and Python code to address your project’s requirements, solutions to the errors you encountered, and five practical examples to help you get started. Remember to adapt the SQL query and feature engineering steps to match the specific targets and compounds you are interested in. Good luck!

**Resumen en español:**

Esta guía completa le proporciona un modelo de análisis claro, código SQL y Python para abordar los requisitos de su proyecto, soluciones a los errores que encontró y cinco ejemplos prácticos para ayudarlo a comenzar. Recuerde adaptar la consulta SQL y los pasos de ingeniería de características para que coincidan con los objetivos y compuestos específicos que le interesan. ¡Buena suerte!