# PharmApp Suite

## 🧠 AI for Drug Discovery and Development 🧪

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

## 🧩 Topic: Correlating bioactivity with target taxonomy

Okay, I understand. You’re working with ChEMBL 35 data, using RDKit for analysis, and adhering to an AIMLops template. You’re encountering SQL and Python errors and need help structuring your code and analysis for the Topic\_CheMBL\_35\_26 project. Let’s break this down.

**1. Analysis and Model (Analyse et Modèle)**

The overall goal appears to be extracting and analyzing data from ChEMBL 35, likely focusing on a specific aspect within Topic\_CheMBL\_35\_26. To suggest a more specific analysis model, I need more information about the topic itself. However, let’s assume you’re interested in building a simple QSAR (Quantitative Structure-Activity Relationship) model. This involves:

* **Data Extraction:** Fetching relevant activity data (e.g., IC50, Ki values) and corresponding molecule SMILES strings from ChEMBL.
* **Data Cleaning and Preparation:**
  + Handling missing values and inconsistencies.
  + Converting SMILES strings to RDKit Mol objects.
  + Calculating molecular descriptors using RDKit (e.g., molecular weight, LogP, number of hydrogen bond donors/acceptors).
* **Model Building:**
  + Splitting the data into training and testing sets.
  + Training a regression model (e.g., Linear Regression, Random Forest) to predict activity based on the descriptors.
* **Model Evaluation:**
  + Evaluating the model’s performance on the test set using metrics like R-squared, RMSE (Root Mean Squared Error), or MAE (Mean Absolute Error).

**Here’s a more detailed breakdown, tailored to a QSAR task:**

1. **Target Selection:** Identify a specific target (e.g., a protein or enzyme) you’re interested in. ChEMBL has targets identified by ChEMBL ID.
2. **Activity Selection:** Choose the activity type you’ll be modeling (e.g., IC50). Standardize activity units (e.g., convert all IC50s to nM).
3. **Data Filtering:** Filter out data points that are unreliable or outside a reasonable activity range (e.g., very large or very small IC50 values).
4. **Feature Generation (Descriptors):** Use RDKit to generate molecular descriptors. Consider using a diverse set of descriptors to capture different aspects of molecular structure.
5. **Model Selection:** Start with a simple model (e.g., Linear Regression) and then try more complex models (e.g., Random Forest, Support Vector Regression) if needed.
6. **Cross-Validation:** Use cross-validation (e.g., k-fold cross-validation) to evaluate the model’s performance more robustly.
7. **Model Interpretation:** Analyze the model to understand which descriptors are most important for predicting activity. This can provide insights into the structure-activity relationship.

**2. SQL Code (Code SQL)**

Here’s SQL code to extract data, addressing the error you mentioned and limiting the data to 100 rows. The key is to correctly filter the standard\_value column. The initial error ERROR: operator does not exist: numeric ~ unknown, LINE 12: AND act.standard\_value ~ '^[0-9\.]+$' arises because you’re trying to use a regular expression-like operator (~) on a numeric column. We’ll address this by casting to text when using regex. Also, we’ll use a combination of WHERE and LIKE to accomplish filtering for numerical standard values.

-- SQL Query to extract data from ChEMBL 35 (limited to 100 rows)  
-- and addressing standard\_value filtering.  
-- Please replace 'CHEMBL\_TARGET\_ID' with the actual target ID.  
  
SELECT  
 md.chembl\_id AS molecule\_chembl\_id,  
 cs.canonical\_smiles,  
 act.standard\_type,  
 act.standard\_value,  
 act.standard\_units  
FROM  
 activities act  
JOIN  
 molecule\_dictionary md ON act.molregno = md.molregno  
JOIN  
 compound\_structures cs ON md.molregno = cs.molregno  
JOIN  
 target\_dictionary td ON act.tid = td.tid  
WHERE  
 td.chembl\_id = 'CHEMBL\_TARGET\_ID' -- Replace with the target CHEMBL ID  
 AND act.standard\_type = 'IC50' -- Or other activity type  
 AND act.standard\_units = 'nM' -- Or other units  
 AND act.standard\_value IS NOT NULL  
 AND act.standard\_value::TEXT LIKE '%.%' -- Ensure standard\_value contains a decimal point  
 AND act.standard\_value::TEXT NOT LIKE '%-%' -- Ensure standard\_value doesn't contain a minus sign  
 AND act.standard\_value::TEXT ~ '^[0-9\.]+$' -- Check it's a number  
LIMIT 100;

**Explanation:**

* **SELECT Clause:** Selects the molecule ChEMBL ID, SMILES string, activity type, activity value, and units.
* **FROM Clause:** Specifies the tables to join (activities, molecule\_dictionary, compound\_structures, and target\_dictionary).
* **JOIN Clause:** Connects the tables based on relevant foreign keys.
* **WHERE Clause:**
  + td.chembl\_id = 'CHEMBL\_TARGET\_ID' : Filters for a specific ChEMBL target (replace 'CHEMBL\_TARGET\_ID' with the actual ChEMBL ID). **Crucial: Find the ChEMBL ID for your target of interest.**
  + act.standard\_type = 'IC50' : Filters for a specific activity type (e.g., IC50). Adjust this if you want Ki, Kd, etc.
  + act.standard\_units = 'nM' : Filters for a specific unit (e.g., nM). Standardize units if needed.
  + act.standard\_value IS NOT NULL : Excludes entries with missing standard values.
  + act.standard\_value::TEXT LIKE '%.%': Checks if the standard value contains a decimal point to ensure it’s a floating-point number. We cast the numeric column to TEXT for the LIKE operator.
  + act.standard\_value::TEXT NOT LIKE '%-%': Makes sure it’s not a negative number.
  + act.standard\_value::TEXT ~ '^[0-9\.]+$': Ensures that the string contains only numbers and periods. This regular expression check is now applied to a text representation of the value.
* **LIMIT 100 Clause:** Limits the results to the first 100 rows.

**How to use the SQL code:**

1. **Replace 'CHEMBL\_TARGET\_ID':** Find the correct ChEMBL ID for your target and substitute it in the query. You can browse ChEMBL using their web interface to find the ID.
2. **Execute in pgAdmin:** Run this query in your pgAdmin interface connected to your ChEMBL 35 database.
3. **Save to CSV:** Export the results of the query to a CSV file named something like chembl\_data.csv and save it to the ../data/ directory, as you specified.

**3. Python Code (Code Python)**

Here’s the Python code (using Jupyter Notebook format) to read the data, perform descriptor calculation, and build a basic QSAR model. This addresses the error you mentioned regarding the squared=False parameter in mean\_squared\_error by checking scikit-learn version before use.

# -\*- coding: utf-8 -\*-  
"""Topic\_CheMBL\_35\_26\_1\_Data\_Preparation\_and\_QSAR\_Modeling.ipynb  
  
Automatically generated by Colaboratory.  
  
Original file is located at  
 https://colab.research.google.com/drive/1i\_g6u8P7U9G1j\_yWfWq9w5oY2\_n2Wl3N  
"""  
  
import os  
import pandas as pd  
from rdkit import Chem  
from rdkit.Chem import Descriptors  
from rdkit.Chem import Lipinski  
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  
import sklearn  
import warnings  
warnings.filterwarnings("ignore")  
# Define the base path according to your AIMLops structure  
base\_path = ".." # Assuming the notebook is one level deep from the base  
  
# Construct the data file path  
data\_file\_path = os.path.join(base\_path, "data", "chembl\_data.csv") # Adjust filename if needed  
print(f"Loading data from: {data\_file\_path}")  
  
# Load the data using pandas  
try:  
 df = pd.read\_csv(data\_file\_path)  
 print("Data loaded successfully.")  
except FileNotFoundError:  
 print(f"Error: File not found at {data\_file\_path}. Make sure you've run the SQL query and saved the data correctly.")  
 exit()  
  
# Data Cleaning and Preparation  
print("\nData Cleaning and Preparation...")  
df = df.dropna(subset=['canonical\_smiles', 'standard\_value']) # Drop rows with missing SMILES or activity values  
df = df[df['standard\_value'] > 0] # Filter out non-positive standard values to avoid log transform errors  
df['standard\_value'] = pd.to\_numeric(df['standard\_value'], errors='coerce') # Ensure numeric type after loading from CSV  
df = df.dropna(subset=['standard\_value']) # Clean up any rows that failed conversion to numeric  
  
# RDKit Descriptor Calculation  
print("\nCalculating RDKit Descriptors...")  
  
def lipinski\_descriptors(smiles):  
 """Calculates Lipinski descriptors using RDKit."""  
 mol = Chem.MolFromSmiles(smiles)  
 if mol is None:  
 return None  
 mw = Descriptors.MolWt(mol)  
 logp = Descriptors.MolLogP(mol)  
 hbd = Lipinski.NumHDonors(mol)  
 hba = Lipinski.NumHAcceptors(mol)  
 return mw, logp, hbd, hba  
  
# Apply the function to create new columns  
df[['MW', 'LogP', 'HBD', 'HBA']] = df['canonical\_smiles'].apply(lambda x: pd.Series(lipinski\_descriptors(x)) if lipinski\_descriptors(x) is not None else pd.Series([None]\*4))  
  
df = df.dropna() # Remove rows where descriptor calculation failed (invalid SMILES)  
print(df.head())  
  
# Feature Scaling (Important for many models)  
print("\nFeature Scaling...")  
X = df[['MW', 'LogP', 'HBD', 'HBA']]  
y = np.log10(df['standard\_value']) # Log transform activity values (often improves model performance)  
  
scaler = StandardScaler()  
X\_scaled = scaler.fit\_transform(X)  
  
# Data Splitting  
print("\nData Splitting...")  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_scaled, y, test\_size=0.2, random\_state=42)  
  
# Model Training  
print("\nModel Training...")  
model = LinearRegression()  
model.fit(X\_train, y\_train)  
  
# Model Evaluation  
print("\nModel Evaluation...")  
y\_pred = model.predict(X\_test)  
  
# Check scikit-learn version before using squared=False  
if sklearn.\_\_version\_\_ < '1.4':  
 mse = mean\_squared\_error(y\_test, y\_pred)  
else:  
 mse = mean\_squared\_error(y\_test, y\_pred, squared=False)  
  
r2 = r2\_score(y\_test, y\_pred)  
  
print(f"Mean Squared Error: {mse}")  
print(f"R-squared: {r2}")  
  
# Print the first 5 rows of the dataframe for inspection  
print("\nFirst 5 rows of the processed dataframe:")  
print(df.head())

**Explanation:**

1. **Imports:** Imports necessary libraries.
2. **File Path:** Defines the correct file path based on your AIMLops structure using os.path.join. **Important:** Double-check that this path is correct relative to where you run the notebook. I’ve assumed the notebook is in a subfolder.
3. **Data Loading:** Loads the CSV data using pandas. Includes error handling for the file not found.
4. **Data Cleaning:**
   * Removes rows with missing SMILES or activity values.
   * Filters out non-positive standard\_value to avoid issues with the log transform.
   * Converts standard\_value to numeric, handling potential errors during conversion.
5. **Descriptor Calculation:**
   * Defines a function lipinski\_descriptors to calculate Lipinski’s Rule of Five descriptors using RDKit.
   * Applies this function to the canonical\_smiles column to create new columns for each descriptor (MW, LogP, HBD, HBA).
   * Handles cases where RDKit fails to process a SMILES string.
6. **Feature Scaling:** Scales the descriptor values using StandardScaler. This is *crucial* for many machine learning algorithms.
7. **Data Splitting:** Splits the data into training and testing sets.
8. **Model Training:** Creates a Linear Regression model and trains it on the training data.
9. **Model Evaluation:**
   * Predicts activity values for the test set.
   * Calculates Mean Squared Error (MSE) and R-squared to evaluate the model’s performance.
   * **Addresses the squared=False error:** The code now checks the scikit-learn version and uses the squared=False parameter only if the version is 1.4 or greater.

**How to use the Python code:**

1. **Save the code:** Save the code above as a .ipynb file named Topic\_CheMBL\_35\_26\_1\_Data\_Preparation\_and\_QSAR\_Modeling.ipynb in your notebook directory.
2. **Adjust the base\_path:** Make sure the base\_path variable in the Python code is correctly set to reflect the location of your data directory.
3. **Run in Jupyter Notebook:** Open the .ipynb file in Jupyter Notebook and run each cell sequentially.
4. **Interpret the results:** Examine the MSE and R-squared values to assess the model’s performance. A higher R-squared (closer to 1) indicates a better fit. A lower MSE indicates lower prediction error.

**4. Examples (Exemples)**

Here are 5 examples of how you could adapt and extend this basic workflow:

1. **Different Target:** Modify the SQL query to use a different CHEMBL\_TARGET\_ID to analyze activity data for a different protein or enzyme.
2. **Different Activity Type:** Change the act.standard\_type filter in the SQL query to analyze a different activity type (e.g., Ki, Kd). Also remember to change the unit act.standard\_units.
3. **More Descriptors:** Add more molecular descriptors to the Python code using other RDKit functions. Explore different descriptor types to potentially improve model performance. Examples include:

* from rdkit.Chem import AllChem  
  # Calculate Morgan Fingerprint (ECFP4)  
  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(fp)  
   else:  
   return None  
    
  # Example usage (add to the descriptor calculation section)  
  df['MorganFP'] = df['canonical\_smiles'].apply(calculate\_morgan\_fingerprint)  
  df = df.dropna() # Drop rows where fingerprint calculation failed
* Remember to adapt the X variable to include the new fingerprint data:
* X = np.stack(df['MorganFP'].values)

1. **Different Model:** Experiment with different machine learning models in the Python code, such as Random Forest, Support Vector Regression, or Neural Networks. This might improve prediction accuracy.

* from sklearn.ensemble import RandomForestRegressor  
    
  model = RandomForestRegressor(n\_estimators=100, random\_state=42) # Example  
  model.fit(X\_train, y\_train)

1. **Feature Selection:** Implement feature selection techniques (e.g., using SelectKBest or Recursive Feature Elimination) to identify the most relevant descriptors and potentially simplify the model.

* from sklearn.feature\_selection import SelectKBest, f\_regression  
    
  selector = SelectKBest(score\_func=f\_regression, k=3) # Select the top 3 features  
  X\_new = selector.fit\_transform(X\_scaled, y)  
    
  # Use X\_new for training and testing instead of X\_scaled  
  X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_new, y, test\_size=0.2, random\_state=42)

**Key Considerations and Troubleshooting:**

* **ChEMBL ID:** The most important thing is to use the correct ChEMBL ID for your target. Double-check this!
* **File Paths:** Make sure the file paths in your Python code are correct. Use os.path.join to construct paths dynamically.
* **RDKit Installation:** Ensure RDKit is correctly installed in your Jupyter Notebook environment.
* **Memory Usage:** If you still encounter memory issues, consider processing the data in smaller chunks (e.g., using pandas chunksize parameter when reading the CSV). However, with a limit of 100 rows, this shouldn’t be necessary.
* **Data Exploration:** Spend time exploring the data to understand its characteristics. Look for outliers, missing values, and inconsistencies.

By following these steps and addressing the potential issues, you should be able to successfully run your ChEMBL 35 data analysis project and build a basic QSAR model using RDKit. Remember to adapt the code and analysis to your specific research question and the nature of the Topic\_CheMBL\_35\_26.