# PharmApp Suite

## 🧠 AI for Drug Discovery and Development 🧪

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

## 🧩 Topic: Finding selective COX-2 inhibitors

Okay, I understand. You’re working on a Chembl 35 data analysis project using RDKit and need guidance on how to structure your code and address specific errors. I’ll help you set up your analysis following an AIMLops template, incorporating best practices and addressing the issues you’ve encountered. We will focus on retrieving a limited dataset (100 rows) for efficiency and clarity.

Here’s a breakdown of the analysis, the code, and some example use cases:

**1. Analysis Model**

The core of this analysis likely involves extracting bioactivity data from the ChEMBL database, processing it using RDKit for chemical structure manipulation and feature generation, and then applying machine learning techniques for modeling and prediction. Here’s a general framework:

* **Data Extraction (SQL):** Query the ChEMBL database to retrieve relevant bioactivity data. This will likely involve joining tables like activities, assays, molecule\_dictionary, and potentially others to get IC50 values, target information, and compound structures.
* **Data Cleaning and Filtering (Python):**
  + Remove entries with missing or invalid values (e.g., IC50 values that are NULL or outside a reasonable range).
  + Standardize units (e.g., convert all IC50 values to nM).
  + Handle duplicate entries.
  + Convert SMILES strings to RDKit Mol objects.
* **Feature Generation (Python/RDKit):** Calculate molecular descriptors using RDKit. These descriptors represent various physicochemical properties of the molecules and can be used as features in your machine learning models. Examples include:
  + Molecular weight
  + LogP (octanol-water partition coefficient)
  + Topological Polar Surface Area (TPSA)
  + Number of hydrogen bond donors/acceptors
  + Rings count
* **Machine Learning (Python/Scikit-learn):**
  + Split the data into training and testing sets.
  + Choose a suitable machine learning model (e.g., Random Forest, Support Vector Machine, Linear Regression). The choice of model depends on the specific problem you’re trying to solve (e.g., predicting IC50 values, classifying compounds as active/inactive).
  + Train the model on the training data.
  + Evaluate the model’s performance on the testing data using appropriate metrics (e.g., R-squared, RMSE, AUC).

**2. Code (SQL & Python)**

**Folder Structure (AIMLops)**

Topic\_CheMBL\_35\_92/  
├── data/  
│ └── chembl\_bioactivity\_100.csv # CSV file exported from SQL  
├── notebooks/  
│ ├── Topic\_CheMBL\_35\_92\_1\_data\_extraction\_and\_cleaning.ipynb  
│ ├── Topic\_CheMBL\_35\_92\_2\_feature\_generation\_and\_modeling.ipynb  
├── src/ #optional, for reusable python functions  
│ └── utils.py  
└── README.md

**SQL Code (Save as Topic\_CheMBL\_35\_92/data/chembl\_bioactivity\_sql.sql)**

-- Query to extract bioactivity data from ChEMBL  
-- limiting to 100 rows for demonstration  
SELECT  
 md.chembl\_id,  
 md.pref\_name,  
 act.standard\_type,  
 act.standard\_relation,  
 act.standard\_value,  
 act.standard\_units,  
 act.pchembl\_value,  
 acts.assay\_id,  
 assays.description,  
 assays.assay\_type,  
 assays.confidence\_score,  
 mol.canonical\_smiles  
FROM  
 activities act  
JOIN  
 assays ON act.assay\_id = assays.assay\_id  
JOIN  
 target\_dictionary td ON assays.tid = td.tid  
JOIN  
 molecule\_dictionary md ON act.molregno = md.molregno  
JOIN  
 compound\_structures mol ON md.molregno = mol.molregno  
JOIN  
 activity\_properties acts ON act.activity\_id = acts.activity\_id  
  
WHERE  
 --Filtering to get IC50 values  
 act.standard\_type = 'IC50'  
 AND act.standard\_relation = '='  
 --Fix the ERROR: operator does not exist: numeric ~ unknown, LINE 12: AND act.standard\_value ~ '^[0-9\.]+$'  
 AND act.standard\_value::text ~ '^[0-9.]+$'  
 AND act.standard\_units = 'nM'  
LIMIT 100;  
  
  
-- Save the output to a CSV file using pgAdmin's export functionality.  
-- Right-click on the query result in pgAdmin, select "Copy", and then "Copy with Headers".  
-- Paste this into a text editor and save as chembl\_bioactivity\_100.csv in the data folder

**Explanation of SQL:**

* **SELECT Clause:** Selects the necessary columns from various tables. I’ve included chembl\_id, standard\_type, standard\_value, standard\_units, and canonical\_smiles as core features.
* **JOIN Clauses:** Joins tables based on their relationships (e.g., activities.assay\_id = assays.assay\_id). This is crucial to combine information from different parts of the ChEMBL database.
* **WHERE Clause:** Filters the data to include only IC50 values with specific criteria:
  + act.standard\_type = 'IC50': Ensures we’re only looking at IC50 measurements.
  + act.standard\_relation = '=': Selects only exact IC50 values (not “>” or “<”).
  + act.standard\_value::text ~ '^[0-9.]+$': **Addresses the error**. This ensures that the standard\_value is a numeric value (allowing for decimal points). We cast standard\_value to text type before using the regular expression operator ~.
  + act.standard\_units = 'nM': Ensures values are in nanomolar units.
* **LIMIT 100 Clause:** Restricts the output to the first 100 rows.

**Important:**

1. **Run this SQL query in pgAdmin.**
2. **Export the results to a CSV file:** The easiest way is to run the query, then right-click on the results grid in pgAdmin, choose “Copy”, and then “Copy with Headers”. Paste the copied data into a text editor (e.g., Notepad++, VS Code) and save it as chembl\_bioactivity\_100.csv in your data directory (Topic\_CheMBL\_35\_92/data/). Ensure it’s saved as a CSV file.

**Python Code (Notebook: Topic\_CheMBL\_35\_92/notebooks/Topic\_CheMBL\_35\_92\_1\_data\_extraction\_and\_cleaning.ipynb)**

import pandas as pd  
import numpy as np  
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, r2\_score  
import matplotlib.pyplot as plt  
  
# Define the base path  
base\_path = os.path.dirname(os.getcwd()) # Assuming the notebook is one level down from the project root  
data\_path = os.path.join(base\_path, 'data', 'chembl\_bioactivity\_100.csv')  
notebook\_path = os.path.join(base\_path, 'notebooks')  
  
print(f"Base Path: {base\_path}")  
print(f"Data Path: {data\_path}")  
print(f"Notebook Path: {notebook\_path}")  
  
# Load the data from the CSV file  
try:  
 df = pd.read\_csv(data\_path)  
 print("Data loaded successfully.")  
except FileNotFoundError:  
 print(f"Error: File not found at {data\_path}")  
 exit()  
  
# Display the first few rows of the DataFrame  
print(df.head())  
  
# Data Cleaning and Preprocessing  
# Convert standard\_value to numeric, handling potential errors  
df['standard\_value'] = pd.to\_numeric(df['standard\_value'], errors='coerce')  
  
# Remove rows with missing values in 'standard\_value' or 'canonical\_smiles'  
df = df.dropna(subset=['standard\_value', 'canonical\_smiles'])  
  
# Convert IC50 to pIC50  
# Function to convert IC50 to pIC50  
def ic50\_to\_pIC50(ic50\_nM):  
 """Converts IC50 in nM to pIC50."""  
 pIC50 = -np.log10(ic50\_nM \* 1e-9) # Convert nM to Molar  
 return pIC50  
  
df['pIC50'] = df['standard\_value'].apply(ic50\_to\_pIC50)  
  
# RDKit Mol object creation  
df['mol'] = df['canonical\_smiles'].apply(lambda x: Chem.MolFromSmiles(x))  
df = df[df['mol'].notna()] # Remove rows where Mol object creation failed  
print(df.head())  
  
#Feature generation  
def calculate\_lipinski\_descriptors(mol):  
 """Calculates Lipinski descriptors for a given molecule."""  
 mw = Descriptors.MolWt(mol)  
 logp = Chem.Crippen.MolLogP(mol)  
 hbd = Descriptors.NumHDonors(mol)  
 hba = Descriptors.NumHAcceptors(mol)  
 return mw, logp, hbd, hba  
  
# Apply descriptor calculation  
df[['mw', 'logp', 'hbd', 'hba']] = df['mol'].apply(lambda x: pd.Series(calculate\_lipinski\_descriptors(x)))  
print(df.head())  
  
# Select features and target variable  
X = df[['mw', 'logp', 'hbd', 'hba']] # Features  
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)  
  
# 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)  
r2 = r2\_score(y\_test, y\_pred)  
  
print(f'Mean Squared Error: {mse}')  
print(f'R-squared: {r2}')  
  
# Visualization (optional)  
plt.scatter(y\_test, y\_pred)  
plt.xlabel("Actual pIC50")  
plt.ylabel("Predicted pIC50")  
plt.title("Actual vs. Predicted pIC50")  
plt.show()

**Explanation of Python Code:**

1. **Import Libraries:** Imports necessary libraries like pandas, RDKit, and scikit-learn.
2. **Define Paths:** Uses os.path.join to create paths to the data file. **Crucially, it dynamically calculates the base\_path assuming the notebook is one level deep in your AIMLops structure.** This makes the code more portable.
3. **Load Data:** Loads the CSV data into a Pandas DataFrame.
4. **Data Cleaning:**
   * Converts standard\_value to numeric, handling potential non-numeric values by converting them to NaN.
   * Removes rows with NaN in standard\_value or canonical\_smiles. This ensures that only valid data is used for further analysis.
5. **IC50 to pIC50 Conversion:** Converts IC50 values to pIC50 values, which are often preferred for modeling.
6. **RDKit Mol Object Creation:** Creates RDKit Mol objects from the SMILES strings. This is necessary for calculating molecular descriptors. It also removes any rows where the Mol object could not be created (invalid SMILES).
7. **Feature Generation:** Calculates Lipinski descriptors (molecular weight, LogP, H-bond donors, H-bond acceptors) using RDKit. These are simple but useful descriptors.
8. **Data Splitting:** Splits the data into training and testing sets.
9. **Model Training:** Trains a linear regression model (you can experiment with other models).
10. **Model Evaluation:** Evaluates the model using Mean Squared Error (MSE) and R-squared.
11. **Visualization (Optional):** Creates a scatter plot of predicted vs. actual pIC50 values.

**Addressing the squared=False Error:**

The error “old scikit-learn version does not support parameters squared=False in the mean\_squared\_error function” indicates that you’re using an older version of scikit-learn. The squared=False parameter was introduced in a later version. **The easiest solution is to remove the squared=False argument**. The default behavior (returning the MSE) is perfectly acceptable for most use cases. If you need RMSE, just take the square root of the MSE.

**Modified Code (No squared=False):**

# Evaluate the model  
mse = mean\_squared\_error(y\_test, y\_pred) # Remove squared=False  
rmse = np.sqrt(mse) #calculate RMSE  
r2 = r2\_score(y\_test, y\_pred)  
  
print(f'Mean Squared Error: {mse}')  
print(f'Root Mean Squared Error: {rmse}')  
print(f'R-squared: {r2}')

If you *really* need to use squared=False (which is unlikely), you’ll need to upgrade your scikit-learn version. You can do this using pip:

pip install --upgrade scikit-learn

**3. Examples**

Here are 5 examples of how this code might be used and extended:

1. **Target-Specific Analysis:** Modify the SQL query to focus on a specific protein target (e.g., by filtering on the target\_dictionary.target\_name). This allows you to build models for individual targets, which is often more accurate than a general model.

* -- Example: Target-specific analysis (e.g., EGFR)  
  WHERE  
   act.standard\_type = 'IC50'  
   AND act.standard\_relation = '='  
   AND act.standard\_value::text ~ '^[0-9.]+$'  
   AND act.standard\_units = 'nM'  
   AND td.target\_name = 'Epidermal growth factor receptor'; -- EGFR

1. **Different Bioactivity Type:** Instead of IC50, analyze Ki, Kd, or EC50 values. Change the act.standard\_type filter in the SQL query accordingly.

* -- Example: Analyzing Ki values  
  WHERE  
   act.standard\_type = 'Ki'  
   AND act.standard\_relation = '='  
   AND act.standard\_value::text ~ '^[0-9.]+$'  
   AND act.standard\_units = 'nM';

1. **More Complex Feature Generation:** Use RDKit to calculate a wider range of molecular descriptors, such as:
   * **Morgan Fingerprints (ECFPs):** Capture structural features and are very powerful for machine learning.
   * **Physicochemical Properties:** TPSA, number of rotatable bonds, etc.

* from rdkit.Chem import AllChem  
    
  def calculate\_morgan\_fingerprint(mol, radius=2, nBits=2048):  
   """Calculates Morgan fingerprint for a given molecule."""  
   try:  
   fp = AllChem.GetMorganFingerprintAsBitVect(mol, radius, nBits=nBits)  
   fp\_array = np.zeros((1,), dtype=int)  
   Chem.DataStructs.ConvertToNumpyArray(fp, fp\_array)  
   return fp\_array  
   except Exception as e:  
   print(f"Error calculating fingerprint: {e}")  
   return None  
    
  # Apply fingerprint calculation  
  df['morgan\_fp'] = df['mol'].apply(calculate\_morgan\_fingerprint)  
  df = df.dropna(subset=['morgan\_fp'])  
    
  # Prepare data for machine learning using Morgan fingerprints  
  X = np.vstack(df['morgan\_fp'].to\_numpy()) # Stack the fingerprints into a numpy array  
    
  y = df['pIC50']  
  X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)  
    
  #Example with RandomForest  
  from sklearn.ensemble import RandomForestRegressor  
  model = RandomForestRegressor(n\_estimators=100, random\_state=42)  
  model.fit(X\_train, y\_train)  
  y\_pred = model.predict(X\_test)  
    
  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}')

1. **Different Machine Learning Models:** Experiment with different machine learning algorithms. Random Forest, SVM, and deep learning models (e.g., using TensorFlow or PyTorch) can provide better performance than linear regression. Consider hyperparameter tuning to optimize model performance.
2. **QSAR Model Building:** Build a Quantitative Structure-Activity Relationship (QSAR) model to predict the bioactivity of new compounds based on their chemical structure. This is a common application of this type of analysis.

**In Summary (En Français):**

Votre projet consiste à analyser des données ChEMBL 35 en utilisant RDKit pour la recherche et le développement de médicaments. Voici une analyse et un guide pour vous aider :

1. **Analyse du modèle :**
   * Extraction des données de bioactivité de ChEMBL via SQL.
   * Nettoyage et filtrage des données en Python (gestion des valeurs manquantes, standardisation des unités).
   * Génération de descripteurs moléculaires avec RDKit (poids moléculaire, LogP, etc.).
   * Application de techniques d’apprentissage automatique (Random Forest, SVM) pour la modélisation et la prédiction.
2. **Code :**
   * SQL pour extraire les données de ChEMBL (limité à 100 lignes pour la démonstration). Important de corriger l’erreur avec act.standard\_value::text ~ '^[0-9.]+$'.
   * Python pour le nettoyage des données, la génération de descripteurs, la division des données en ensembles d’entraînement et de test, l’entraînement du modèle et l’évaluation. Corriger l’erreur squared=False en supprimant cet argument.
3. **Exemples :**
   * Analyse spécifique à une cible.
   * Analyse de différents types de bioactivité (Ki, Kd).
   * Génération de descripteurs plus complexes (empreintes digitales de Morgan).
   * Utilisation de différents modèles d’apprentissage automatique.
   * Construction d’un modèle QSAR.

N’oubliez pas de bien organiser votre code selon la structure AIMLops, et d’utiliser os.path.join pour gérer les chemins de fichiers de manière portable. Bon courage!

Remember to adapt the code and examples to your specific research question and dataset. Good luck!