# PharmApp Suite

## 🧠 AI for Drug Discovery and Development 🧪

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

## 🧩 Topic: Ortholog mapping of targets (human → mouse)

Okay, I understand. You’re working on a Chembl 35 data analysis project using RDKit, following an AIMLops-style folder structure, and you’ve encountered a couple of errors. Let’s break down the analysis, provide corrected code snippets, and address your specific needs.

**Analysis Model**

The overall goal is to analyze Chembl 35 data to support drug research and development. Given your request, it sounds like you want to extract and process bioactivity data, potentially building a predictive model. Here’s a basic analysis model that we will create:

1. **Data Extraction:** We’ll use SQL to query the Chembl 35 database, specifically targeting bioactivity data related to specific targets or assays. The SQL query will be refined to filter for standard values (IC50, Ki, etc.) and activity outcomes.
2. **Data Preprocessing:**
   * **Filtering:** Remove entries with missing or inconsistent data.
   * **Data Type Conversion:** Convert standard values to numeric and handle units.
   * **Molecular Representation:** Use RDKit to generate molecular fingerprints (e.g., Morgan fingerprints) from SMILES strings.
3. **Feature Engineering:** Combine molecular fingerprints with other relevant features (e.g., physicochemical properties).
4. **Model Building (Simplified):** For demonstration purposes, let’s focus on a simple Regression model to predict bioactivity values from molecular fingerprints.
5. **Model Evaluation (Simplified):** Split data into training and test sets, train the model, and evaluate its performance using Mean Squared Error (MSE).
6. **Example Generation:** Demonstrate how to apply this analysis process to various aspects such as activity distribution or target activity profile.

**Error Analysis & Correction**

* **Error (a): ERROR: operator does not exist: numeric ~ unknown, LINE 12: AND act.standard\_value ~ '^[0-9\.]+$'**
  + **Explanation:** This error arises because the ~ operator (regular expression matching) is not directly applicable to numeric columns in PostgreSQL. You’re trying to check if act.standard\_value (which is probably numeric) matches a regular expression.
  + **Solution:** Cast the standard\_value to a text type before applying the regular expression. Alternatively, avoid regex altogether if you simply want to check if the value is a number. The better solution is to directly check if it’s numeric which the code below does.
* **Error (b): old scikit-learn version does not support parameters squared=False in the mean\_squared\_error function**
  + **Explanation:** The squared=False parameter in mean\_squared\_error was introduced in a later version of scikit-learn.
  + **Solution:**
    - **Upgrade scikit-learn:** The best solution is to upgrade your scikit-learn version. pip install -U scikit-learn
    - **Calculate RMSE manually:** If you can’t upgrade, calculate the Root Mean Squared Error (RMSE) manually by taking the square root of the MSE: rmse = np.sqrt(mean\_squared\_error(y\_true, y\_pred))

**Folder Structure (Based on AIMLops)**

Let’s assume a basic AIMLops-style structure:

my\_project/  
├── data/ # Stores raw data (CSV files from SQL)  
├── notebooks/ # Jupyter notebooks with analysis code  
├── models/ # Stores trained models (optional for this example)  
├── src/ # Python modules (optional for this example)  
├── .env # Environment variables (database credentials)  
└── README.md

**Code Implementation**

**1. SQL Code (../data/chembl\_data.sql)**

-- Extract bioactivity data for a specific target (e.g., a specific protein)  
-- Limiting to 100 rows for demonstration. Adjust WHERE clause as needed.  
  
SELECT  
 cmp.chembl\_id,  
 act.standard\_type,  
 act.standard\_value,  
 act.standard\_units,  
 act.activity\_comment,  
 mol.canonical\_smiles  
FROM  
 activities act  
JOIN  
 assays ass ON act.assay\_id = ass.assay\_id  
JOIN  
 target\_dictionary td ON ass.tid = td.tid  
JOIN  
 molecule\_dictionary mol ON act.molregno = mol.molregno  
JOIN  
 component\_sequences cs ON td.tid = cs.tid  
JOIN  
 component\_molecules cm ON cs.component\_id = cm.component\_id  
JOIN  
 molecule\_dictionary cmp ON cm.molregno = cmp.molregno  
WHERE  
 td.chembl\_id = 'CHEMBL205' -- Example: Replace with your target of interest  
 AND act.standard\_type IN ('IC50', 'Ki', 'EC50') -- Filter for common activity types  
 AND act.standard\_value IS NOT NULL  
 AND act.standard\_units = 'nM'  
 AND act.standard\_value > 0 -- Avoid zero values  
 --AND act.standard\_value ~ '^[0-9\.]+$' -- Removed regex. Handled in Python after conversion.  
ORDER BY act.standard\_value ASC  
LIMIT 100;

**Important:**

* Replace 'CHEMBL205' with the Chembl ID of the target you want to analyze.
* Adjust the standard\_type filter to include the types of bioactivity you’re interested in.

**Save the SQL query result to a CSV file (e.g., ../data/chembl\_bioactivity\_data.csv) using pgAdmin.**

**2. Python Code (notebooks/Topic\_CheMBL\_35\_28\_1\_Data\_Analysis.ipynb)**

import os  
import pandas as pd  
import numpy as np  
from rdkit import Chem  
from rdkit.Chem import AllChem  
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  
from sklearn.preprocessing import StandardScaler  
  
# Define the base path  
base\_path = os.path.dirname(os.getcwd()) #goes to the parent directory  
data\_path = os.path.join(base\_path, 'data')  
model\_path = os.path.join(base\_path, 'models') #create a model path to save models later  
  
# Load the data from the CSV file  
data\_file = os.path.join(data\_path, 'chembl\_bioactivity\_data.csv')  
try:  
 df = pd.read\_csv(data\_file)  
 print(f"Data loaded successfully from {data\_file}")  
except FileNotFoundError:  
 print(f"Error: File not found at {data\_file}")  
 exit()  
  
# Data Cleaning and Preprocessing  
print("\nData Cleaning and Preprocessing...")  
df = df.dropna(subset=['canonical\_smiles', 'standard\_value']) # Drop rows with missing SMILES or standard\_value  
df = df[df['standard\_value'] > 0] # Remove non-positive standard\_value  
  
# Convert standard\_value to numeric, handling potential errors  
df['standard\_value'] = pd.to\_numeric(df['standard\_value'], errors='coerce')  
df = df.dropna(subset=['standard\_value']) #remove NaN created by the coercion.  
  
# Molecular Feature Generation (RDKit)  
print("\nGenerating Molecular Features...")  
def generate\_fingerprint(smiles):  
 mol = Chem.MolFromSmiles(smiles)  
 if mol:  
 fp = AllChem.GetMorganFingerprintAsBitVect(mol, 2, nBits=2048) # Increased nBits to 2048  
 return np.array(fp)  
 else:  
 return None  
  
df['fingerprint'] = df['canonical\_smiles'].apply(generate\_fingerprint)  
df = df.dropna(subset=['fingerprint'])  
  
# Example: Calculate LogP (another molecular descriptor)  
df['logp'] = df['canonical\_smiles'].apply(lambda x: Descriptors.MolLogP(Chem.MolFromSmiles(x)) if Chem.MolFromSmiles(x) else None)  
df = df.dropna(subset=['logp'])  
  
# Prepare Data for Modeling  
print("\nPreparing Data for Modeling...")  
X = np.stack(df['fingerprint'].to\_numpy()) # Stack fingerprints into a NumPy array  
y = np.log10(df['standard\_value'].astype(float)) # Log transform of standard\_value. Important for bioactivity data.  
  
# Data Scaling  
scaler = StandardScaler()  
X\_scaled = scaler.fit\_transform(X)  
  
# Split Data  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_scaled, y, test\_size=0.2, random\_state=42)  
  
# Model Training  
print("\nTraining the Model...")  
model = LinearRegression()  
model.fit(X\_train, y\_train)  
  
# Model Evaluation  
print("\nEvaluating the Model...")  
y\_pred = model.predict(X\_test)  
mse = mean\_squared\_error(y\_test, y\_pred)  
rmse = np.sqrt(mse) # Calculate RMSE manually  
  
print(f"Mean Squared Error: {mse:.3f}")  
print(f"Root Mean Squared Error: {rmse:.3f}")  
  
# Example Usages (Illustrative)  
print("\nExample Usages:")  
  
# 1. Distribution of Activity Values  
print("\n1. Distribution of Activity Values (Log Transformed):")  
import matplotlib.pyplot as plt  
import seaborn as sns  
sns.histplot(y)  
plt.xlabel("Log10(Standard Value)")  
plt.ylabel("Frequency")  
plt.title("Distribution of Log10(Standard Value)")  
plt.show()  
  
# 2. Predicted vs. Actual Values  
print("\n2. Predicted vs. Actual Values:")  
plt.scatter(y\_test, y\_pred)  
plt.xlabel("Actual Log10(Standard Value)")  
plt.ylabel("Predicted Log10(Standard Value)")  
plt.title("Predicted vs. Actual Log10(Standard Value)")  
plt.show()  
  
# 3. Most Active Compounds  
print("\n3. Most Active Compounds (Top 5):")  
top\_5 = df.sort\_values('standard\_value').head(5)  
print(top\_5[['chembl\_id', 'standard\_value', 'canonical\_smiles']])  
  
# 4. Impact of LogP on Activity  
print("\n4. Impact of LogP on Activity:")  
plt.scatter(df['logp'], y)  
plt.xlabel("LogP")  
plt.ylabel("Log10(Standard Value)")  
plt.title("LogP vs. Log10(Standard Value)")  
plt.show()  
  
# 5. Model Prediction for a specific molecule  
print("\n5. Model Prediction for a Specific Molecule:")  
example\_smiles = 'CC(=O)Oc1ccccc1C(=O)O' # Aspirin  
mol = Chem.MolFromSmiles(example\_smiles)  
if mol:  
 fp = AllChem.GetMorganFingerprintAsBitVect(mol, 2, nBits=2048)  
 fp\_array = np.array(fp).reshape(1, -1) # Reshape for single sample prediction  
 fp\_scaled = scaler.transform(fp\_array) # Scale using the SAME scaler  
 predicted\_log\_value = model.predict(fp\_scaled)[0]  
 print(f"Predicted Log10(Standard Value) for {example\_smiles}: {predicted\_log\_value:.2f}")  
else:  
 print(f"Invalid SMILES: {example\_smiles}")  
  
print("\nAnalysis Complete.")

**Key Improvements and Explanations:**

* **Robust Data Loading:** Uses os.path.join for path construction and includes error handling for file loading.
* **Data Cleaning:** Handles missing data and ensures standard\_value is numeric.
* **Log Transformation:** Applies a log transformation to standard\_value. This is *crucial* for bioactivity data because it often follows a log-normal distribution. Linear models work much better on normally distributed data.
* **Molecular Fingerprint Generation:** Uses AllChem.GetMorganFingerprintAsBitVect to generate Morgan fingerprints. **Increased nBits to 2048** for better representation.
* **Data Scaling:** **StandardScaler is used to scale the fingerprint data.** This is essential for linear models and can significantly improve performance. The scaler MUST be fit on the TRAINING data only, and then used to transform both the training and test data. This prevents data leakage.
* **Model Training and Evaluation:** Trains a LinearRegression model and calculates RMSE (to avoid the squared=False error).
* **Example Usages:** Includes five examples to demonstrate the kinds of analyses you can perform with the data and model.
* **Clear Comments:** The code is extensively commented to explain each step.
* **Error Handling:** Adds error handling for SMILES conversion.

**Important Considerations:**

* **Database Credentials:** Store your database credentials (user, password, IP address, database name) in a .env file and use a library like python-dotenv to load them. **Never hardcode credentials in your scripts.**
* **Model Selection:** Linear Regression is a very basic model. For more accurate predictions, consider using more sophisticated models like Random Forests, Gradient Boosting, or Support Vector Machines.
* **Cross-Validation:** Use cross-validation to get a more robust estimate of model performance.
* **Feature Engineering:** Experiment with different molecular descriptors and feature combinations to improve model accuracy. Consider adding more properties with RDKit, such as Molecular Weight, Number of Hydrogen Bond Donors/Acceptors, etc.
* **Target Specificity:** Focus your analysis on specific targets or assay types to get more meaningful results.

**How to Run:**

1. **Install Dependencies:** bash pip install pandas rdkit scikit-learn numpy matplotlib seaborn
2. **Create the .env file:** (Optional but highly recommended) DATABASE\_USER=your\_db\_user DATABASE\_PASSWORD=your\_db\_password DATABASE\_HOST=your\_db\_host DATABASE\_NAME=your\_db\_name
3. **Run the SQL query in pgAdmin and save the result to ../data/chembl\_bioactivity\_data.csv.**
4. **Open the Jupyter notebook (notebooks/Topic\_CheMBL\_35\_28\_1\_Data\_Analysis.ipynb) and run all cells.**

**Two Languages (English & Simplified Chinese)**

**English:**

The analysis model involves extracting bioactivity data from ChEMBL, preprocessing it with RDKit for feature generation (molecular fingerprints), and building a simple regression model. The code provides error handling for common issues, includes data cleaning steps, and demonstrates various ways to visualize and interpret the results. Remember to replace the example target ID with your target of interest and consider using more advanced modeling techniques for better accuracy.

**Simplified Chinese:**

分析模型包括从ChEMBL提取生物活性数据，使用RDKit进行预处理以生成特征（分子指纹），并构建一个简单的回归模型。 代码提供了针对常见问题的错误处理，包括数据清理步骤，并演示了可视化和解释结果的各种方法。 请记住将示例目标ID替换为您感兴趣的目标，并考虑使用更高级的建模技术以获得更好的准确性。