# PharmApp Suite

## 🧠 AI for Drug Discovery and Development 🧪

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

## 🧩 Topic: Target pathway analysis via UniProt/KEGG

Okay, I understand. You’re working with ChEMBL 35 data in PostgreSQL, using RDKit for analysis within a Jupyter notebook following an AIMLops template. You need to address specific errors, limit the data to 100 rows for speed, and are looking for SQL and Python code examples, along with detailed analysis. Let’s break this down.

**1. Analysis Model Overview**

The underlying analysis model is data-driven, focusing on extracting and processing information from the ChEMBL database. You are using SQL to query and filter the data and Python (RDKit, scikit-learn) to further process and analyze the retrieved information. The following are likely areas of focus:

* **Data Retrieval and Cleaning (SQL):** SQL will be used to select specific activity data from the ChEMBL database based on certain criteria (e.g., activity type, target, potency range). It will also be used to clean the data by filtering out entries with invalid or missing values. This is a crucial step as the quality of the downstream analysis depends on the quality of the input data.
* **Data Preprocessing (Python, RDKit):** Once the data is retrieved, RDKit will be used to process the SMILES strings, generating molecular descriptors that can be used as input features for machine learning models.
* **Data Analysis & Modelling (Python, scikit-learn):** Scikit-learn or other machine learning libraries may be used for tasks such as:
  + **Regression:** Predicting activity values (e.g., IC50, Ki) based on molecular descriptors.
  + **Classification:** Predicting whether a molecule is active or inactive against a specific target.
  + **Clustering:** Grouping molecules based on their structural similarity or activity profiles.
  + **Feature Selection:** Identifying the most important molecular descriptors that correlate with activity.

**2. Code Implementation & Error Resolution**

Here’s a breakdown of the code, including error correction and example usage, focusing on a hypothetical regression task.

**Folder Structure (Based on AIMLops Template - Example)**

chembl\_35\_project/  
├── data/ # Contains data files (.csv from SQL queries)  
├── notebooks/ # Contains Jupyter notebooks (.ipynb files)  
│ └── Topic\_CheMBL\_35\_29\_1\_data\_extraction.ipynb  
│ └── Topic\_CheMBL\_35\_29\_2\_analysis.ipynb  
├── src/ # (Optional) Python modules, custom functions  
├── models/ # (Optional) Saved model files  
├── reports/ # (Optional) Analysis reports  
└── README.md

**SQL Code (data/chembl\_35\_data.csv)**

-- chembl\_35\_data.sql  
  
SELECT  
 act.molregno,  
 md.chembl\_id,  
 act.standard\_type,  
 act.standard\_relation,  
 act.standard\_value,  
 act.standard\_units,  
 act.assay\_id,  
 cmp.canonical\_smiles  
FROM  
 activities act  
JOIN molecule\_dictionary md ON act.molregno = md.molregno  
JOIN compound\_structures cmp ON md.molregno = cmp.molregno  
WHERE  
 act.standard\_type = 'IC50' -- Example: Filter for IC50 values  
 AND act.standard\_relation = '='  
 AND act.standard\_units = 'nM'  
 AND act.standard\_value IS NOT NULL  
 AND act.standard\_value::text ~ '^[0-9\.]+$' --Corrected Line  
LIMIT 100;  
  
-- Explanation:  
-- 1. corrected the error of 'operator does not exist: numeric ~ unknown': 'act.standard\_value ~ '^[0-9\.]+$'' to 'act.standard\_value::text ~ '^[0-9\.]+$''  
-- The error occurs because the 'standard\_value' column is likely numeric, and the '~' operator (regular expression matching) typically works with text data. The fix is to explicitly cast the 'standard\_value' to text using '::text'.  
-- 2. `LIMIT 100` limits the result to 100 rows. Adjust as needed.  
-- 3. I also added standard\_relation = '=' to filter to avoid different IC50 cases

**How to Run SQL in pgAdmin and Save to CSV**

1. **Connect to your ChEMBL 35 database in pgAdmin** using the provided credentials (IP, user, pass, database).
2. **Open a query window** and paste the SQL code above.
3. **Execute the query.**
4. **Export the results to a CSV file:**
   * In pgAdmin, after the query executes, you should see the results in a grid.
   * Right-click on the grid, select “Copy All Rows”.
   * Open a text editor (like Notepad or VS Code), paste the copied data.
   * Save the file as chembl\_35\_data.csv in your data/ directory. Make sure to select “All Files” as the save type to avoid a .txt extension. Alternatively, you can use pgAdmin’s “Export…” feature and choose CSV format directly.

**Python Code (notebooks/Topic\_CheMBL\_35\_29\_2\_analysis.ipynb)**

# notebooks/Topic\_CheMBL\_35\_29\_2\_analysis.ipynb  
  
import os  
import pandas as pd  
from rdkit import Chem  
from rdkit.Chem import AllChem  
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 base path  
base\_path = os.path.dirname(os.getcwd()) # Assumes notebook is one level deep  
  
# Data Loading and Preprocessing  
data\_path = os.path.join(base\_path, 'data', 'chembl\_35\_data.csv')  
df = pd.read\_csv(data\_path)  
  
# Function to convert SMILES to Morgan Fingerprints (ECFP4)  
def smiles\_to\_fingerprint(smiles, radius=2, nBits=2048):  
 mol = Chem.MolFromSmiles(smiles)  
 if mol is None:  
 return None  
 fp = AllChem.GetMorganFingerprintAsBitVect(mol, radius, nBits=nBits)  
 return np.array(fp)  
  
# Apply the fingerprint generation  
df['fingerprint'] = df['canonical\_smiles'].apply(smiles\_to\_fingerprint)  
df = df.dropna(subset=['fingerprint', 'standard\_value']) # Drop rows where fingerprint generation failed  
  
# Convert IC50 to pIC50 (example, adjust as needed)  
df['pIC50'] = -np.log10(df['standard\_value'].astype(float) / 1e9) # Convert nM to M, then -log10  
  
# Prepare data for machine learning  
X = np.vstack(df['fingerprint'].values)  
y = df['pIC50'].values  
  
# Data Scaling (important for linear models)  
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)  
  
# Model Training  
model = LinearRegression()  
model.fit(X\_train, y\_train)  
  
# Model Evaluation  
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}")  
  
# Optional: Visualize predictions  
import matplotlib.pyplot as plt  
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. **Imports:** Imports necessary libraries (pandas, RDKit, scikit-learn).
2. **Path Definition:** Defines the path to the CSV data file using os.path.join.
3. **Data Loading:** Loads the CSV data using pandas.
4. **SMILES to Fingerprint Conversion:**
   * Defines a function smiles\_to\_fingerprint that takes a SMILES string as input and generates a Morgan fingerprint (ECFP4) using RDKit.
   * Handles potential errors during SMILES parsing (returns None if the SMILES is invalid).
   * Applies the function to the canonical\_smiles column of the DataFrame to create a new fingerprint column.
   * Drops rows where fingerprint generation failed.
5. **pIC50 Conversion (Example):** Converts IC50 values to pIC50 values. This is a common transformation in drug discovery to work with a logarithmic scale. *Adjust the calculation based on the standard\_units and standard\_type you are working with.*
6. **Data Preparation for Machine Learning:**
   * Extracts the fingerprint data and pIC50 values into NumPy arrays.
   * Splits the data into training and testing sets using train\_test\_split.
7. **Model Training:**
   * Creates a LinearRegression model.
   * Trains the model on the training data using model.fit.
8. **Model Evaluation:**
   * Makes predictions on the test data using model.predict.
   * Calculates the mean squared error (MSE) and R-squared (R2) to evaluate the model’s performance.
   * Prints the MSE and R2 values.
9. **Visualization (Optional):** Creates a scatter plot of actual vs. predicted pIC50 values.

**Addressing the Scikit-Learn Error**

The error “old scikit-learn version does not support parameters squared=False in the mean\_squared\_error function” indicates that you are using an older version of scikit-learn. The squared=False parameter was introduced in a later version to allow you to get the root mean squared error (RMSE) directly.

**Solution:** Remove the squared=False parameter if you have an older version of scikit-learn. If you want the RMSE, calculate it manually:

# If squared=False is not supported:  
mse = mean\_squared\_error(y\_test, y\_pred)  
rmse = np.sqrt(mse)  
print(f"Root Mean Squared Error: {rmse}")  
  
# If you can, it is recommended that you use the latest scikit-learn version  
# pip install -U scikit-learn

**Important Considerations:**

* **RDKit Installation:** Make sure you have RDKit installed in your environment. You can install it using conda install -c conda-forge rdkit or pip install rdkit.
* **Error Handling:** Add more robust error handling in the smiles\_to\_fingerprint function to catch and log potential errors during SMILES parsing.
* **Feature Engineering:** Experiment with different types of molecular descriptors (e.g., physicochemical properties, other fingerprint types) to see if you can improve the model’s performance.
* **Model Selection:** Consider trying different machine learning models (e.g., Random Forest, Support Vector Machines) to see which one performs best for your data.
* **Hyperparameter Tuning:** Optimize the hyperparameters of your chosen model using techniques like cross-validation and grid search.
* **Validation:** Always validate your model on an independent test set to ensure that it generalizes well to new data.

**3. Five Examples of Different Analyses**

Here are five examples of different analyses you could perform using this data and code:

1. **Target-Specific Activity Prediction:** Modify the SQL query to filter for activities against a specific protein target (using target\_dictionary and target\_components tables to link to activities). Then, train a model to predict the activity of compounds against *that specific target*. This allows you to build target-focused models.
   * **SQL Change:** Add JOIN target\_dictionary td ON act.tid = td.tid WHERE td.chembl\_id = 'CHEMBL205' -- Example Target (Replace ‘CHEMBL205’ with the actual Chembl ID of the target).
2. **Classification (Active vs. Inactive):** Instead of predicting a continuous activity value (pIC50), convert the problem to a classification problem. Define a threshold for activity (e.g., pIC50 > 6 means “active”). Then, train a classifier (e.g., Logistic Regression, Random Forest) to predict whether a compound is active or inactive.
   * **Python Change:** Add a threshold to determine active vs inactive: df['active'] = (df['pIC50'] > 6).astype(int)
   * Change LinearRegression() to LogisticRegression() or RandomForestClassifier()
   * Change evaluation metrics to accuracy\_score, precision\_score, recall\_score, and f1\_score
3. **Structure-Activity Relationship (SAR) Analysis:** Use RDKit to calculate various physicochemical properties (e.g., LogP, molecular weight, number of hydrogen bond donors/acceptors) and correlate these properties with activity. This can help identify key structural features that contribute to activity.
   * **Python Change:** Calculate additional properties:
   * from rdkit.Chem import Descriptors  
     df['logP'] = df['canonical\_smiles'].apply(lambda x: Descriptors.MolLogP(Chem.MolFromSmiles(x)) if Chem.MolFromSmiles(x) else None)  
     #... other properties ...
   * Analyze correlations between properties and activity.
4. **Clustering Analysis:** Use clustering algorithms (e.g., k-means, hierarchical clustering) to group molecules based on their structural similarity (fingerprints) or activity profiles. This can help identify clusters of compounds with similar activity.
   * **Python Change:** Use KMeans for clustering.
   * from sklearn.cluster import KMeans  
     kmeans = KMeans(n\_clusters=5, random\_state=0, n\_init='auto') #Specify number of clusters  
     df['cluster'] = kmeans.fit\_predict(X)
5. **Activity Cliff Detection:** Identify pairs of compounds that are structurally very similar but have significantly different activities (activity cliffs). These compounds can provide valuable insights into the SAR.
   * **Python Change:** Requires calculating Tanimoto similarity between fingerprints. This is more involved but provides specific SAR insights. There are RDKit examples online for this.

**Important Notes (En Français et Anglais)**

* **Adjustments:** The provided code is a starting point. You’ll need to adjust it based on the specific activity types, targets, and data characteristics you are working with.
* **Data Understanding:** Thoroughly understand the ChEMBL database schema and the meaning of the different fields to ensure that you are extracting and interpreting the data correctly.
* **Experimentation:** Don’t be afraid to experiment with different approaches and parameters to find the best solution for your specific problem.
* **Code Readability:** Write clear and well-documented code to make it easier to understand and maintain.

**En Français:**

* **Ajustements:** Le code fourni est un point de départ. Vous devrez l’ajuster en fonction des types d’activités, des cibles et des caractéristiques des données spécifiques avec lesquelles vous travaillez.
* **Compréhension des données:** Comprenez en profondeur le schéma de la base de données ChEMBL et la signification des différents champs pour vous assurer que vous extrayez et interprétez correctement les données.
* **Expérimentation:** N’ayez pas peur d’expérimenter différentes approches et paramètres pour trouver la meilleure solution à votre problème spécifique.
* **Lisibilité du code:** Écrivez un code clair et bien documenté pour le rendre plus facile à comprendre et à maintenir.

Let me know if you have any other questions or would like me to elaborate on any of these points! Good luck with your project!