# PharmApp Suite

## 🧠 AI for Drug Discovery and Development 🧪

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

## 🧩 Topic: Case study: JAK inhibitors from ChEMBL

Okay, I understand. You’re working on a Chembl 35 data analysis project using RDKit, following an AIMLops-style directory structure, and encountering specific issues related to SQL and scikit-learn versions. Your goal is to extract a manageable subset (100 rows) of data and perform some analysis.

Here’s a breakdown of the analysis, code examples (SQL and Python), and solutions to your errors, all tailored to the “Topic\_CheMBL\_35\_91” convention:

**1. Analysis of the Analysis Model**

The project aims to extract and analyze data from the Chembl 35 database related to bioactivity. A typical workflow includes:

* **Data Extraction (SQL):** This involves querying the Chembl database to retrieve relevant information like compound structures (SMILES), target information, and bioactivity data (e.g., IC50, Ki values). The query should filter for relevant data, likely based on specific targets, assays, or activity types.
* **Data Preprocessing (Python/RDKit):** RDKit is used to process the SMILES strings, generate molecular fingerprints, calculate descriptors, and prepare the data for machine learning models. This step often includes handling missing values, outliers, and data normalization.
* **Exploratory Data Analysis (Python):** Understanding dataset structure. This helps to uncover patterns, anomalies, or relationships.
* **Model Building (Python/Scikit-learn):** Machine learning models (e.g., regression, classification) are used to predict bioactivity based on the calculated descriptors. Common models include linear regression, random forests, and support vector machines.
* **Model Evaluation (Python/Scikit-learn):** The models are evaluated using appropriate metrics (e.g., R-squared, RMSE for regression; accuracy, AUC for classification) on a held-out test set.
* **Interpretation and Visualization (Python):** Results are visualized and interpreted to understand the factors driving bioactivity. This might involve generating scatter plots, histograms, or visualizing molecular structures with activity information.

The analysis will likely involve establishing relationships between chemical structures and their biological activity.

**2. SQL Code (Topic\_CheMBL\_35\_91\_1.sql)**

This SQL script extracts a limited number of data (100 rows) from the Chembl database, specifically focusing on activities with standard values.

-- Topic\_CheMBL\_35\_91\_1.sql  
  
-- Connect to the chembl\_35 database  
\c chembl\_35 rd rd 192.168.206.136  
  
-- Extract relevant data with a limit of 100 rows  
SELECT  
 md.molregno,  
 cs.canonical\_smiles,  
 act.standard\_type,  
 act.standard\_value,  
 act.standard\_units,  
 act.assay\_id  
FROM  
 molecule\_dictionary md  
JOIN  
 compound\_structures cs ON md.molregno = cs.molregno  
JOIN  
 activities act ON md.molregno = act.molregno  
WHERE  
 act.standard\_type IN ('IC50', 'Ki', 'Kd') -- Filter for relevant activity types  
 AND act.standard\_value IS NOT NULL -- Ensure standard value exists  
 AND act.standard\_units = 'nM' -- Filter for standard units  
 AND act.standard\_value::text ~ '^[0-9\.]+$' -- Corrected numeric check  
LIMIT 100;  
  
-- Save the output to a CSV file  
\copy (SELECT  
 md.molregno,  
 cs.canonical\_smiles,  
 act.standard\_type,  
 act.standard\_value,  
 act.standard\_units,  
 act.assay\_id  
FROM  
 molecule\_dictionary md  
JOIN  
 compound\_structures cs ON md.molregno = cs.molregno  
JOIN  
 activities act ON md.molregno = act.molregno  
WHERE  
 act.standard\_type IN ('IC50', 'Ki', 'Kd') -- Filter for relevant activity types  
 AND act.standard\_value IS NOT NULL -- Ensure standard value exists  
 AND act.standard\_units = 'nM' -- Filter for standard units  
 AND act.standard\_value::text ~ '^[0-9\.]+$' -- Corrected numeric check  
LIMIT 100) TO '../data/chembl\_35\_subset.csv' WITH CSV HEADER;  
  
  
-- Explanation:  
-- \* `\c chembl\_35 rd rd 192.168.206.136`: Connects to the database.  
-- \* `SELECT ... FROM ...`: Selects the desired columns from the tables.  
-- \* `JOIN`: Joins tables based on `molregno` (molecule registration number).  
-- \* `WHERE`: Filters the data based on activity type, existence of standard value, and units.  
-- \* `LIMIT 100`: Restricts the output to 100 rows.  
-- \* `\copy ... TO ...`: Saves the query result to a CSV file. Make sure the path `../data/chembl\_35\_subset.csv` is valid relative to where you run the `psql` command.

**Correction for SQL Error (a): ERROR: operator does not exist: numeric ~ unknown, LINE 12: AND act.standard\_value ~ '^[0-9\.]+$'**

The issue is that the ~ operator (regular expression match) cannot be directly applied to a numeric type. To fix this, cast the standard\_value to text before applying the regular expression.

AND act.standard\_value::text ~ '^[0-9\.]+$'

This casts the standard\_value to a text string, allowing the regular expression to match. The regex ^[0-9\.]+$ ensures that the value consists only of digits and periods.

**Run this script in pgAdmin.** This will create the chembl\_35\_subset.csv file in your ../data/ directory.

**3. Python Code (Topic\_CheMBL\_35\_91\_2.ipynb)**

This Jupyter Notebook code reads the CSV file, processes the data using RDKit, calculates descriptors, and performs a simple analysis.

# Topic\_CheMBL\_35\_91\_2.ipynb  
  
import pandas as pd  
import numpy as np  
import os  
from rdkit import Chem  
from rdkit.Chem import Descriptors  
from rdkit.Chem import AllChem  
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 (important for AIMLops structure)  
base\_path = os.path.dirname(os.getcwd()) # Go up one level to the project root  
data\_path = os.path.join(base\_path, 'data', 'chembl\_35\_subset.csv') # Full path to the CSV  
  
# Load the data  
try:  
 df = pd.read\_csv(data\_path)  
 print(f"Data loaded successfully from: {data\_path}")  
except FileNotFoundError:  
 print(f"Error: File not found at {data\_path}. Make sure the SQL script has been run and the file exists.")  
 exit()  
  
# Data Cleaning and Preprocessing  
df = df.dropna(subset=['canonical\_smiles', 'standard\_value']) # Remove rows with missing SMILES or values  
df = df[df['standard\_value'] > 0] # Remove non-positive values (important for log transform)  
df = df.drop\_duplicates(subset=['canonical\_smiles']) # Remove duplicate SMILES entries  
  
# Convert standard\_value to pIC50 (or pKi, pKd)  
df['pActivity'] = -np.log10(df['standard\_value'] \* 1e-9) # Convert nM to M, then to pActivity  
  
# RDKit Descriptor Calculation  
def calculate\_descriptors(smiles):  
 mol = Chem.MolFromSmiles(smiles)  
 if mol is None:  
 return None  
 try:  
 AllChem.Compute2DCoords(mol) # Ensure 2D coordinates are computed  
 descriptors = {desc\_name: desc\_func(mol) for desc\_name, desc\_func in Descriptors.descList}  
 return descriptors  
 except Exception as e:  
 print(f"Error calculating descriptors for {smiles}: {e}")  
 return None  
  
df['descriptors'] = df['canonical\_smiles'].apply(calculate\_descriptors)  
  
# Handle missing descriptors  
df = df.dropna(subset=['descriptors'])  
  
# Convert descriptors to DataFrame  
descriptors\_df = pd.DataFrame(df['descriptors'].tolist(), index=df.index)  
df = pd.concat([df, descriptors\_df], axis=1)  
  
# Select features and target  
X = df.iloc[:, 12:] # Select all descriptor columns (adjust index if needed)  
y = df['pActivity']  
  
# 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)  
  
# 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}")  
  
# Visualization (Example)  
plt.scatter(y\_test, y\_pred)  
plt.xlabel("Actual pActivity")  
plt.ylabel("Predicted pActivity")  
plt.title("Actual vs. Predicted pActivity")  
plt.show()

**Explanation:**

1. **Import Libraries:** Imports necessary libraries.
2. **File Path:** Constructs the full file path using os.path.join to adhere to the AIMLops directory structure. This is crucial for portability and maintainability.
3. **Data Loading:** Loads the CSV data using pandas. Includes error handling for the case where the file is not found.
4. **Data Cleaning:**
   * Removes rows with missing SMILES strings or standard values.
   * Removes duplicate SMILES to avoid bias in the model.
   * Removes non-positive standard\_value to prepare for logarithmic transformation.
5. **pActivity Calculation:** Converts standard\_value (assumed to be in nM) to pActivity (e.g., pIC50) using -log10(value \* 1e-9).
6. **Descriptor Calculation:**
   * Defines a function calculate\_descriptors that takes a SMILES string as input and returns a dictionary of RDKit descriptors.
   * Applies the function to the canonical\_smiles column to create a new ‘descriptors’ column.
7. **Handle Missing Descriptors:** \*Remove rows with missing descriptors
8. **Descriptor DataFrame:**
   * Extracts the calculated descriptors from the dictionary format into a Pandas DataFrame (descriptors\_df).
   * Concatenates the descriptor DataFrame with the original DataFrame.
9. **Feature and Target Selection:** Selects the descriptor columns (adjust index if necessary) as features (X) and pActivity as the target variable (y).
10. **Data Splitting:** Splits the data into training and testing sets.
11. **Model Training:** Creates a linear regression model and trains it on the training data.
12. **Model Evaluation:** Predicts pActivity values for the test set and calculates the mean squared error (MSE) and R-squared.
13. **Visualization:** Creates a scatter plot of actual vs. predicted pActivity values.

**Correction for Python Error (b): old scikit-learn version does not support parameters squared=False in the mean\_squared\_error function**

The squared=False parameter was added to mean\_squared\_error in a later version of scikit-learn. Since you can’t upgrade (as per your instructions), the solution is to take the square root of the MSE to get the RMSE (Root Mean Squared Error):

mse = mean\_squared\_error(y\_test, y\_pred)  
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}")

**Important Notes:**

* **Directory Structure:** Double-check that the paths used in os.path.join are correct for your AIMLops project structure. This is critical for reproducibility. The example assumes that the notebook is located one level deep within the project directory.
* **Descriptor Selection:** You might want to experiment with different sets of descriptors. Not all descriptors are relevant for every problem.
* **Model Selection:** Linear regression is a simple starting point. Try other models like Random Forest, Gradient Boosting, or Support Vector Machines, which may perform better.
* **Feature Scaling:** Consider scaling your features (e.g., using StandardScaler from scikit-learn) before training the model. This can improve the performance of some machine-learning algorithms.
* **Error Handling:** The code includes basic error handling for file loading and descriptor calculation. Add more robust error handling as needed.
* **Virtual Environment:** It is *highly* recommended to use a virtual environment (e.g., venv, conda) to manage your Python dependencies. This will prevent conflicts between different projects.

**4. Code in English and Chinese:**

**English:**

* The SQL script extracts compound data and bioactivity information from the ChEMBL database.
* The Python script calculates molecular descriptors using RDKit.
* A Linear Regression model is trained to predict bioactivity based on the descriptors.
* The model’s performance is evaluated using Mean Squared Error and R-squared.

**Chinese:**

* SQL脚本从ChEMBL数据库提取化合物数据和生物活性信息。
* Python脚本使用RDKit计算分子描述符。
* 训练线性回归模型，以基于描述符预测生物活性。
* 使用均方误差和R平方评估模型的性能。

**5. 5 Examples of Analysis/Tasks you can perform**

Here are 5 examples of more specific tasks you can perform with this data:

1. **Predict pIC50 of a novel compound:** Given a SMILES string of a new compound, calculate its descriptors and use your trained model to predict its pIC50 value. This is a classic virtual screening application.

* new\_smiles = 'Cc1ccccc1C(=O)O' # Example SMILES  
  new\_mol = Chem.MolFromSmiles(new\_smiles)  
  if new\_mol is not None:  
   new\_descriptors = calculate\_descriptors(new\_smiles)  
   if new\_descriptors is not None:  
   new\_descriptors\_df = pd.DataFrame([new\_descriptors])  
   # Ensure the new descriptors DataFrame has the same columns as the training data  
   missing\_cols = set(X\_train.columns) - set(new\_descriptors\_df.columns)  
   for c in missing\_cols:  
   new\_descriptors\_df[c] = 0 # Or use a more appropriate imputation method  
   new\_descriptors\_df = new\_descriptors\_df[X\_train.columns] # Ensure correct column order  
   predicted\_pIC50 = model.predict(new\_descriptors\_df)[0]  
   print(f"Predicted pIC50 for {new\_smiles}: {predicted\_pIC50}")  
   else:  
   print(f"Could not calculate descriptors for {new\_smiles}")  
  else:  
   print(f"Invalid SMILES string: {new\_smiles}")

1. **Identify the most important descriptors for bioactivity:** Examine the coefficients of your linear regression model to determine which descriptors have the largest impact on the predicted pIC50. This can provide insights into the structural features that drive activity.

* coefficients = pd.DataFrame({'Descriptor': X\_train.columns, 'Coefficient': model.coef\_})  
  coefficients['Absolute\_Coefficient'] = abs(coefficients['Coefficient'])  
  coefficients = coefficients.sort\_values('Absolute\_Coefficient', ascending=False)  
  print("Top 10 Most Important Descriptors:")  
  print(coefficients.head(10))

1. **Build a QSAR model for a specific target:** Filter your data to only include compounds that target a specific protein or enzyme. Then, build a QSAR (Quantitative Structure-Activity Relationship) model specifically for that target. This will likely result in a more accurate model than a general QSAR model. The act.assay\_id field in the SQL query can be used for this filtering.
2. **Compare different machine learning models:** Evaluate the performance of different machine learning algorithms (e.g., Random Forest, SVM) on your data. Compare their performance metrics (e.g., MSE, R-squared) to determine which model performs best. Remember to tune hyperparameters using cross-validation.
3. **Visualize the chemical space:** Use dimensionality reduction techniques (e.g., PCA, t-SNE) to visualize the chemical space of your compounds based on their descriptors. Color-code the points by pIC50 value to see if there are any clusters of active compounds.

By following these steps and examples, you should be able to successfully perform your Chembl 35 data analysis project, address the errors you were encountering, and gain valuable insights from the data. Remember to adapt the code to your specific needs and project requirements. Good luck!