# PharmApp Suite

## 🧠 AI for Drug Discovery and Development 🧪

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

## 🧩 Topic: Scaffold hopping analysis

Okay, I understand. You’re working with ChEMBL 35 data, using RDKit in Jupyter notebooks, and following an AIMLops template. You’re encountering specific errors in your SQL and Python code. Your goal is to extract a small dataset (100 rows) for testing and development.

Here’s a breakdown, along with SQL and Python code examples, and a guide to addressing your errors.

**1. Analysis Model Overview**

Your project seems to involve analyzing bioactivity data from ChEMBL 35. A typical analysis pipeline would include these steps:

* **Data Extraction:** Pulling relevant information from the ChEMBL database using SQL. This typically involves selecting specific activity types (e.g., IC50, Ki), targets, and compounds.
* **Data Cleaning & Transformation:** Cleaning up the extracted data. This can involve filtering for specific activity units (e.g., nM), converting activity values to a consistent scale (e.g., pChEMBL), and handling missing data. This also will include data for Machine Learning processing in the future.
* **Feature Generation:** Generating molecular descriptors using RDKit. These descriptors represent the chemical properties of the molecules and can be used as input for machine learning models.
* **Data Analysis & Modeling:** Performing statistical analysis, building predictive models (e.g., regression, classification), or visualizing the data.

**2. Addressing Errors**

* **SQL Error:** ERROR: operator does not exist: numeric ~ unknown, LINE 12: AND act.standard\_value ~ '^[0-9\.]+$'
* This error means PostgreSQL doesn’t know how to use the regular expression operator ~ (for matching) directly with a numeric column. You need to explicitly cast the standard\_value column to text before applying the regex.
* **Python Error:** old scikit-learn version does not support parameters squared=False in the mean\_squared\_error function
* This means you are using an older version of scikit-learn. You should either upgrade scikit-learn or remove the squared=False parameter if you only need the Mean Squared Error (MSE), not the Root Mean Squared Error (RMSE).

**3. Code Examples (SQL & Python)**

Here’s the code, incorporating error fixes and adhering to your file structure requirements. I will create a simplified example to get you started and then add more complex examples to show case some advanced usages.

# Assumes you have a base directory for your project  
import os  
import sys  
import pandas as pd  
from rdkit import Chem  
from rdkit.Chem import Descriptors  
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 sqlalchemy import create\_engine  
import warnings  
warnings.filterwarnings("ignore")  
  
# Define base path  
base\_path = "." # Or your project's root directory  
  
# Create data directory if it doesn't exist  
data\_dir = os.path.join(base\_path, "data")  
os.makedirs(data\_dir, exist\_ok=True)  
  
notebook\_dir = os.path.join(base\_path, "notebooks")  
os.makedirs(notebook\_dir, exist\_ok=True)  
  
def create\_db\_connection(user, password, host, database):  
 engine = create\_engine(f'postgresql://{user}:{password}@{host}/{database}')  
 return engine.connect()  
  
# Database credentials (replace with your actual credentials)  
db\_user = "rd"  
db\_password = "rd"  
db\_host = "192.168.206.136"  
db\_name = "chembl\_35"  
  
# Establish database connection  
try:  
 connection = create\_db\_connection(db\_user, db\_password, db\_host, db\_name)  
 print("Database connection successful")  
except Exception as e:  
 print(f"Error connecting to database: {e}")  
 sys.exit(1) # Exit if connection fails

**SQL (Save as Topic\_CheMBL\_35\_13\_extract\_data.sql in your data directory)**

-- Topic\_CheMBL\_35\_13\_extract\_data.sql  
-- Extracts activity data for a specific target (e.g., a specific protein)  
-- and filters for a specific activity type (e.g., IC50).  
-- Limits the result to 100 rows for testing.  
  
SELECT  
 cmp.chembl\_id,  
 act.standard\_type,  
 act.standard\_value,  
 act.standard\_units,  
 mol.molfile  
FROM  
 activities act  
JOIN  
 molecule\_dictionary cmp ON act.molregno = cmp.molregno  
JOIN  
 target\_dictionary tgt ON act.tid = tgt.tid  
JOIN  
 compound\_structures mol ON cmp.molregno = mol.molregno  
WHERE  
 tgt.pref\_name = 'CHEMBL205' -- Example target (replace with your target of interest)  
 AND act.standard\_type = 'IC50'  
 AND act.standard\_units = 'nM'  
 AND act.standard\_value IS NOT NULL  
 AND act.standard\_value::text ~ '^[0-9\.]+$' -- Cast to text for regex matching  
LIMIT 100;

**Python (Save as Topic\_CheMBL\_35\_13\_1\_data\_extraction.ipynb in your notebooks directory)**

# Topic\_CheMBL\_35\_13\_1\_data\_extraction.ipynb  
import os  
import pandas as pd  
from rdkit import Chem  
from rdkit.Chem import Descriptors  
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 sqlalchemy import create\_engine  
import warnings  
warnings.filterwarnings("ignore")  
  
# Define base path  
base\_path = "." # Or your project's root directory  
  
# Create data directory if it doesn't exist  
data\_dir = os.path.join(base\_path, "data")  
os.makedirs(data\_dir, exist\_ok=True)  
  
notebook\_dir = os.path.join(base\_path, "notebooks")  
os.makedirs(notebook\_dir, exist\_ok=True)  
  
# Database credentials (replace with your actual credentials)  
db\_user = "rd"  
db\_password = "rd"  
db\_host = "192.168.206.136"  
db\_name = "chembl\_35"  
  
# Output CSV file path  
output\_csv = os.path.join(data\_dir, "chembl\_data.csv")  
  
# SQL query to extract data (assuming you ran the .sql and have the output)  
#sql\_query = "SELECT \* FROM activities LIMIT 100" # Example, replace with your actual query  
  
  
sql\_query = """  
SELECT  
 cmp.chembl\_id,  
 act.standard\_type,  
 act.standard\_value,  
 act.standard\_units,  
 mol.molfile  
FROM  
 activities act  
JOIN  
 molecule\_dictionary cmp ON act.molregno = cmp.molregno  
JOIN  
 target\_dictionary tgt ON act.tid = tgt.tid  
JOIN  
 compound\_structures mol ON cmp.molregno = mol.molregno  
WHERE  
 tgt.pref\_name = 'CHEMBL205' -- Example target (replace with your target of interest)  
 AND act.standard\_type = 'IC50'  
 AND act.standard\_units = 'nM'  
 AND act.standard\_value IS NOT NULL  
 AND act.standard\_value::text ~ '^[0-9\.]+$' -- Cast to text for regex matching  
LIMIT 100;  
"""  
  
# Establish database connection  
try:  
 engine = create\_engine(f'postgresql://{db\_user}:{db\_password}@{db\_host}/{db\_name}')  
 connection = engine.connect()  
 print("Database connection successful")  
except Exception as e:  
 print(f"Error connecting to database: {e}")  
 sys.exit(1) # Exit if connection fails  
  
  
# Read data from the database using pandas  
try:  
 df = pd.read\_sql(sql\_query, connection)  
 print("Data read successfully from the database")  
except Exception as e:  
 print(f"Error reading data from database: {e}")  
 sys.exit(1)  
  
# Save the data to a CSV file  
try:  
 df.to\_csv(output\_csv, index=False)  
 print(f"Data saved to {output\_csv}")  
except Exception as e:  
 print(f"Error saving data to CSV: {e}")  
 sys.exit(1)  
  
# Print the first few rows of the DataFrame to verify  
print(df.head())  
  
# Close the database connection  
connection.close()  
print("Database connection closed")

**Python (Save as Topic\_CheMBL\_35\_13\_2\_feature\_generation\_and\_model.ipynb in your notebooks directory)**

# Topic\_CheMBL\_35\_13\_2\_feature\_generation\_and\_model.ipynb  
import os  
import pandas as pd  
from rdkit import Chem  
from rdkit.Chem import Descriptors  
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  
import warnings  
warnings.filterwarnings("ignore")  
  
# Define base path  
base\_path = "." # Or your project's root directory  
  
# Create data directory if it doesn't exist  
data\_dir = os.path.join(base\_path, "data")  
os.makedirs(data\_dir, exist\_ok=True)  
  
notebook\_dir = os.path.join(base\_path, "notebooks")  
os.makedirs(notebook\_dir, exist\_ok=True)  
  
# Input CSV file path  
input\_csv = os.path.join(data\_dir, "chembl\_data.csv")  
  
# Load the data from the CSV file  
try:  
 df = pd.read\_csv(input\_csv)  
 print(f"Data loaded successfully from {input\_csv}")  
except Exception as e:  
 print(f"Error loading data from CSV: {e}")  
 exit(1)  
  
# Print the first few rows of the DataFrame to verify  
print(df.head())  
  
# Function to calculate molecular weight  
def calculate\_mw(mol):  
 return Descriptors.MolWt(mol)  
  
# Function to calculate LogP  
def calculate\_logp(mol):  
 return Descriptors.MolLogP(mol)  
  
# Function to calculate Hydrogen Bond Donors  
def calculate\_hbd(mol):  
 return Descriptors.NumHDonors(mol)  
  
# Function to calculate Hydrogen Bond Acceptors  
def calculate\_hba(mol):  
 return Descriptors.NumHAcceptors(mol)  
  
# Convert molfile strings to RDKit Mol objects and handle potential errors  
def create\_mol\_object(molfile):  
 try:  
 mol = Chem.MolFromMolBlock(molfile)  
 if mol is None:  
 return None # Handle cases where MolFromMolBlock fails  
 return mol  
 except Exception as e:  
 print(f"Error creating Mol object: {e}")  
 return None  
  
# Apply the function to create Mol objects  
df['ROMol'] = df['molfile'].apply(create\_mol\_object)  
  
# Drop rows where ROMol is None (invalid structures)  
df = df.dropna(subset=['ROMol'])  
  
# Calculate molecular descriptors  
df['Molecular\_Weight'] = df['ROMol'].apply(calculate\_mw)  
df['LogP'] = df['ROMol'].apply(calculate\_logp)  
df['HBD'] = df['ROMol'].apply(calculate\_hbd)  
df['HBA'] = df['ROMol'].apply(calculate\_hba)  
  
# Convert IC50 values to pIC50  
df['pIC50'] = -np.log10(df['standard\_value'] / 1e9) # Convert nM to Molar and then to pIC50  
  
# Display the DataFrame with calculated descriptors  
print(df.head())  
  
# Prepare data for modeling  
X = df[['Molecular\_Weight', 'LogP', 'HBD', 'HBA']]  
y = df['pIC50']  
  
# 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 on the test set  
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}")

**4. Running the Code**

1. **Database Setup:** Ensure your PostgreSQL database is running and accessible at the specified IP address and port.
2. **SQL Execution:** Run the Topic\_CheMBL\_35\_13\_extract\_data.sql script in pgAdmin. This will *not* create a file, but populate the data to pandas dataframe.
3. **Jupyter Notebook:** Open the Topic\_CheMBL\_35\_13\_1\_data\_extraction.ipynb notebook in Jupyter. Run the cells sequentially. This will connect to your database, extract the data, and save it as chembl\_data.csv in the data directory.
4. **Feature Generation and Modeling:** Open the Topic\_CheMBL\_35\_13\_2\_feature\_generation\_and\_model.ipynb notebook. Run the cells sequentially. This will load the CSV file, calculate molecular descriptors, train a linear regression model, and evaluate its performance.

**5. Example Expansions**

Here are 5 example expansions based on your code to show case some advanced usages:

**Example 1: Feature Selection**

* **Goal:** Use feature selection to improve the linear regression model.
* **Changes to Topic\_CheMBL\_35\_13\_2\_feature\_generation\_and\_model.ipynb:**

from sklearn.feature\_selection import SelectKBest, f\_regression  
  
# Feature selection using SelectKBest  
selector = SelectKBest(score\_func=f\_regression, k=3) # Select top 3 features  
X\_new = selector.fit\_transform(X, y)  
  
# Get the indices of the selected features  
selected\_indices = selector.get\_support(indices=True)  
selected\_features = X.columns[selected\_indices]  
  
print("Selected features:", selected\_features)  
  
# Split data into training and testing sets  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_new, y, test\_size=0.2, random\_state=42)  
  
# Train a linear regression model  
model = LinearRegression()  
model.fit(X\_train, y\_train)  
  
# Make predictions on the test set  
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}")

**Example 2: Using Different Regression Models**

* **Goal:** Compare the performance of linear regression with a Random Forest Regressor.
* **Changes to Topic\_CheMBL\_35\_13\_2\_feature\_generation\_and\_model.ipynb:**

from sklearn.ensemble import RandomForestRegressor  
  
# Train a Random Forest Regressor model  
rf\_model = RandomForestRegressor(n\_estimators=100, random\_state=42) # Adjust parameters as needed  
rf\_model.fit(X\_train, y\_train)  
  
# Make predictions on the test set  
y\_pred\_rf = rf\_model.predict(X\_test)  
  
# Evaluate the Random Forest model  
mse\_rf = mean\_squared\_error(y\_test, y\_pred\_rf)  
r2\_rf = r2\_score(y\_test, y\_pred\_rf)  
  
print("Random Forest Results:")  
print(f"Mean Squared Error: {mse\_rf}")  
print(f"R-squared: {r2\_rf}")

**Example 3: Activity Cliff Detection**

* **Goal:** Identify pairs of compounds with similar structures but significantly different activities (activity cliffs). Requires a structural similarity calculation.
* **Changes to Topic\_CheMBL\_35\_13\_2\_feature\_generation\_and\_model.ipynb:**

from rdkit.Chem import AllChem  
from rdkit.DataStructs import FingerprintSimilarity  
  
# Calculate Morgan fingerprints (ECFP4)  
def calculate\_fingerprint(mol):  
 return AllChem.GetMorganFingerprintAsBitVect(mol, 2, nBits=2048)  
  
df['Fingerprint'] = df['ROMol'].apply(calculate\_fingerprint)  
  
# Function to calculate Tanimoto similarity  
def calculate\_tanimoto\_similarity(fp1, fp2):  
 return FingerprintSimilarity(fp1, fp2)  
  
# Calculate similarity between all pairs of compounds (can be slow for large datasets)  
similarity\_matrix = np.zeros((len(df), len(df)))  
for i in range(len(df)):  
 for j in range(i + 1, len(df)):  
 similarity = calculate\_tanimoto\_similarity(df['Fingerprint'][i], df['Fingerprint'][j])  
 similarity\_matrix[i, j] = similarity  
 similarity\_matrix[j, i] = similarity  
  
# Define a similarity threshold and a activity difference threshold  
similarity\_threshold = 0.8  
activity\_difference\_threshold = 1 # pIC50 units  
  
# Identify potential activity cliffs  
activity\_cliffs = []  
for i in range(len(df)):  
 for j in range(i + 1, len(df)):  
 similarity = similarity\_matrix[i, j]  
 activity\_difference = abs(df['pIC50'][i] - df['pIC50'][j])  
 if similarity >= similarity\_threshold and activity\_difference >= activity\_difference\_threshold:  
 activity\_cliffs.append(((df['chembl\_id'][i], df['pIC50'][i]), (df['chembl\_id'][j], df['pIC50'][j]), similarity))  
  
print("Potential Activity Cliffs:")  
for cliff in activity\_cliffs:  
 print(f"Compound 1: {cliff[0]}, Compound 2: {cliff[1]}, Similarity: {cliff[2]}")

**Example 4: Substructure Searching**

* **Goal:** Identify compounds containing a specific substructure (e.g., a common scaffold).
* **Changes to Topic\_CheMBL\_35\_13\_2\_feature\_generation\_and\_model.ipynb:**

from rdkit.Chem import MolFromSmarts  
  
# Define the SMARTS pattern for the substructure  
substructure\_smarts = 'c1ccccc1' # Example: Benzene ring  
substructure = MolFromSmarts(substructure\_smarts)  
  
# Function to check if a molecule contains the substructure  
def contains\_substructure(mol, substructure):  
 if mol is None or substructure is None:  
 return False  
 return mol.HasSubstructMatch(substructure)  
  
# Identify compounds containing the substructure  
df['Contains\_Substructure'] = df['ROMol'].apply(lambda mol: contains\_substructure(mol, substructure))  
  
# Filter the DataFrame to show only compounds containing the substructure  
substructure\_compounds = df[df['Contains\_Substructure']]  
  
print("Compounds containing the substructure:")  
print(substructure\_compounds[['chembl\_id', 'pIC50']])

**Example 5: ADMET Property Prediction (using a pre-trained model)**

* **Goal:** Predict ADMET (Absorption, Distribution, Metabolism, Excretion, Toxicity) properties for the compounds. This requires a pre-trained model (e.g., from DeepChem or a custom model). This example shows a placeholder for where you would load and use the model.
* **Changes to Topic\_CheMBL\_35\_13\_2\_feature\_generation\_and\_model.ipynb:**

# Placeholder for ADMET property prediction (replace with your actual model and data)  
#from deepchem.models import Model # Example using DeepChem  
  
# Load a pre-trained ADMET model (replace with the actual path to your model)  
#admet\_model\_path = 'path/to/your/admet\_model'  
#admet\_model = Model.load(admet\_model\_path)  
  
# Function to predict ADMET properties  
def predict\_admet\_properties(mol):  
 # Convert RDKit Mol object to input format required by the ADMET model  
 # Example: Convert to SMILES and then featurize using the model's featurizer  
 smiles = Chem.MolToSmiles(mol)  
 #features = admet\_model.featurizer([smiles]) # Assuming the model has a featurizer attribute  
  
 # Make predictions using the ADMET model  
 #admet\_predictions = admet\_model.predict(features) #Model.predict requires Numpy array-like inputs, where each element is a sample  
  
 # Return the predicted ADMET properties as a dictionary  
 #return admet\_predictions  
 return None #added to be error-free  
  
# Apply the ADMET property prediction function to each molecule  
df['ADMET\_Predictions'] = df['ROMol'].apply(predict\_admet\_properties)  
  
print(df.head())

**Important Considerations**

* **RDKit Installation:** Make sure you have RDKit installed in your Jupyter notebook environment (conda install -c conda-forge rdkit).
* **Scikit-learn:** Update scikit-learn (pip install --upgrade scikit-learn) to avoid the squared=False error. If upgrading is not feasible, remove the squared=False argument from mean\_squared\_error.
* **Database Connection:** Double-check your database credentials and ensure the database server is running and accessible from your notebook environment.
* **Error Handling:** Add more robust error handling to your code, especially when dealing with external data sources or complex calculations.
* **Performance:** For larger datasets, consider using more efficient data structures and algorithms. Pandas can become slow for very large datasets.
* **Virtual Environments:** Use virtual environments (e.g., conda env create -f environment.yml) to manage dependencies and ensure reproducibility.
* **Molfile Parsing:** Ensure that the molfile data in your ChEMBL database is correctly formatted. Invalid molfile entries can cause errors in RDKit.

**In Summary (En Français):**

Ce plan vous guide à travers l’analyse des données ChEMBL 35 en utilisant RDKit et en suivant un modèle AIMLops. Il comprend :

1. **Analyse du Modèle:** Une vue d’ensemble du pipeline d’analyse de bioactivité typique.
2. **Correction des Erreurs:** Solutions pour les erreurs SQL (conversion du type de données) et Python (mise à jour de scikit-learn ou suppression du paramètre squared=False).
3. **Exemples de Code:** Des exemples SQL (pour extraire les données) et Python (pour le calcul des descripteurs moléculaires et la modélisation).
4. **Instructions d’Exécution:** Les étapes pour exécuter le code dans un environnement Jupyter Notebook.
5. **Exemples d’Extension:** Cinq exemples de code supplémentaires pour montrer l’usage de feature selection, de modèles de régression différents, la détection de “activity cliff”, la recherche de sous-structures et la prédiction des propriétés ADMET.

N’oubliez pas de remplacer les identifiants de base de données et les noms de fichiers par vos propres valeurs. Assurez-vous que RDKit et scikit-learn sont correctement installés dans votre environnement. Ajoutez une gestion d’erreur robuste pour gérer les problèmes potentiels.

Good luck with your project! Let me know if you have more questions.