# PharmApp Suite

## 🧠 AI for Drug Discovery and Development 🧪

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

## 🧩 Topic: FDA-approved ChEMBL drugs analysis

Okay, I understand. You’re working on a Chembl 35 data analysis project using RDKit, following an AIMLops-style structure, and encountering some errors. You want to extract 100 rows of data, and need guidance with SQL and Python code. Let’s get started.

**1. Analysis of the Analysis Model:**

This project likely aims to explore the ChEMBL database for drug discovery purposes. A common workflow would involve:

* **Data Extraction:** Fetching relevant data from the ChEMBL database using SQL queries. This would include information about compounds (structures, properties) and bioactivities (e.g., IC50 values against specific targets).
* **Data Cleaning & Transformation:** Processing the extracted data to handle missing values, convert units, standardize compound representations (e.g., using SMILES), and filter out irrelevant entries.
* **Feature Engineering:** Calculating molecular descriptors using RDKit (e.g., molecular weight, logP, number of hydrogen bond donors/acceptors) that can be used as features for machine learning models.
* **Modeling (Potentially):** Building predictive models to relate compound structure to bioactivity. This could involve tasks like QSAR (Quantitative Structure-Activity Relationship) modeling or virtual screening.

Your current setup suggests you’re focusing on data extraction and initial cleaning/transformation. The error messages indicate issues with your SQL query and the scikit-learn version you’re using.

**2. SQL Code:**

Here’s a SQL query tailored to your needs, addressing the numeric ~ unknown error and limiting the result to 100 rows. The core of the error stems from using the regular expression operator ~ on a numeric column (standard\_value). We’ll cast the standard\_value to TEXT before applying the regex.

-- File: ../data/Topic\_CheMBL\_35\_49.sql  
  
SELECT  
 act.activity\_id,  
 act.standard\_value,  
 act.standard\_units,  
 act.standard\_type,  
 cmp.canonical\_smiles,  
 tgt.target\_chembl\_id  
FROM  
 activities act  
JOIN  
 component\_sequences cs ON act.component\_id = cs.component\_id  
JOIN  
 compound\_structures cmp ON act.molregno = cmp.molregno  
JOIN  
 target\_dictionary tgt ON act.tid = tgt.tid  
WHERE  
 act.standard\_type = 'IC50' -- Filter for IC50 values  
 AND act.standard\_units = 'nM' -- Filter for nM units  
 AND act.standard\_value IS NOT NULL -- Exclude null values  
 AND act.standard\_value::TEXT ~ '^[0-9\.]+$' -- Ensure standard\_value is numeric using TEXT casting and regex  
LIMIT 100; -- Limit to 100 rows

**Explanation:**

* The query joins several ChEMBL tables (activities, component\_sequences, compound\_structures, target\_dictionary) to retrieve activity data, compound SMILES, and target information.
* It filters for IC50 values reported in nM units.
* act.standard\_value::TEXT ~ '^[0-9\.]+$' This is the crucial part. It explicitly casts standard\_value to TEXT before applying the regular expression. The regular expression ^[0-9\.]+$ checks if the value contains only digits and periods, ensuring it’s a numeric value.
* LIMIT 100 limits the result set to 100 rows.

**How to Run:**

1. Open pgAdmin and connect to your database (IP: 192.168.206.136, user: rd, pass: rd, database: chembl\_35).
2. Open a query window.
3. Paste the SQL code into the query window.
4. Execute the query.
5. **Important:** After the query executes successfully, export the result set to a CSV file named Topic\_CheMBL\_35\_49.csv and save it in the ../data/ directory relative to your notebook. Make sure the CSV file includes a header row.

**3. Python Code:**

Here’s the Python code to read the CSV, perform some basic data cleaning, and calculate molecular descriptors using RDKit.

# File: notebook/Topic\_CheMBL\_35\_49\_1\_data\_processing.ipynb  
  
import pandas as pd  
import os  
from rdkit import Chem  
from rdkit.Chem import Descriptors  
import numpy as np  
from sklearn.metrics import mean\_squared\_error # Import mean\_squared\_error  
  
# Define the base path (adjust if needed)  
base\_path = ".." # Assuming the notebook is one level down from the base  
data\_path = os.path.join(base\_path, "data", "Topic\_CheMBL\_35\_49.csv")  
  
# Load the data  
try:  
 df = pd.read\_csv(data\_path)  
except FileNotFoundError:  
 print(f"Error: File not found at {data\_path}. Make sure you saved the CSV correctly.")  
 exit()  
  
# Data Cleaning  
df = df.dropna(subset=['canonical\_smiles', 'standard\_value']) # Remove rows with missing SMILES or standard\_value  
df['standard\_value'] = pd.to\_numeric(df['standard\_value'], errors='coerce') # Convert to numeric, handle errors  
df = df.dropna(subset=['standard\_value']) # Remove rows where conversion to numeric failed  
df = df[df['standard\_value'] > 0] # Remove standard\_value <= 0, log transform will cause error.  
  
# RDKit Descriptor Calculation  
def calculate\_descriptors(smiles):  
 mol = Chem.MolFromSmiles(smiles)  
 if mol is None:  
 return None  
 descriptors = {}  
 descriptors['MW'] = Descriptors.MolWt(mol)  
 descriptors['LogP'] = Descriptors.MolLogP(mol)  
 descriptors['HBD'] = Descriptors.NumHDonors(mol)  
 descriptors['HBA'] = Descriptors.NumHAcceptors(mol)  
 return descriptors  
  
# Apply descriptor calculation  
df['descriptors'] = df['canonical\_smiles'].apply(calculate\_descriptors)  
df = df.dropna(subset=['descriptors']) # Remove rows where descriptor calculation failed  
df = pd.concat([df.drop(['descriptors'], axis=1), df['descriptors'].apply(pd.Series)], axis=1)  
  
# Example usage of mean\_squared\_error (if you had predicted values)  
# Assuming you have 'predicted\_value' column  
# from sklearn.metrics import mean\_squared\_error  
  
# Example data  
# df['predicted\_value'] = np.random.rand(len(df)) \* 1000 # Replace with your actual predicted values  
  
# Check if 'predicted\_value' exists  
#if 'predicted\_value' in df.columns:  
# mse = mean\_squared\_error(df['standard\_value'], df['predicted\_value']) # Default squared=True  
# print(f"Mean Squared Error: {mse}")  
#  
# # If you need RMSE (Root Mean Squared Error), take the square root  
# rmse = np.sqrt(mse)  
# print(f"Root Mean Squared Error: {rmse}")  
#else:  
# print("No predicted\_value column found. Skipping MSE/RMSE calculation.")  
  
  
# Display the first few rows of the processed DataFrame  
print(df.head())

**Explanation:**

1. **Imports:** Imports necessary libraries (pandas, os, RDKit, numpy, scikit-learn).
2. **File Path:** Constructs the full path to your CSV file using os.path.join.
3. **Data Loading:** Loads the CSV file into a pandas DataFrame. Includes error handling in case the file is not found.
4. **Data Cleaning:**
   * Removes rows with missing canonical\_smiles or standard\_value.
   * Converts the standard\_value column to numeric, handling potential errors.
   * Removes rows where the conversion to numeric failed.
   * Remove rows where standard\_value is less or equal to 0.
5. **Descriptor Calculation:**
   * Defines a function calculate\_descriptors to calculate RDKit descriptors (Molecular Weight, LogP, H-bond donors, H-bond acceptors) from a SMILES string.
   * Applies this function to the canonical\_smiles column to create a new ‘descriptors’ column.
   * Handles cases where descriptor calculation might fail (invalid SMILES).
   * Expand the descriptors to different column
6. **MSE Example:**
   * Demonstrates how to use mean\_squared\_error *without* the squared=False parameter (to be compatible with older scikit-learn versions).
   * It calculates both MSE and RMSE (Root Mean Squared Error).
   * Includes a check to see if the predicted\_value column exists before trying to calculate the error. This is important if you’re not always running a prediction step.
7. **Output:** Prints the first few rows of the processed DataFrame to verify the results.

**Addressing the squared=False Error:**

The squared parameter was introduced in a later version of scikit-learn. To fix this with your older version, calculate the MSE and then take the square root to get the RMSE (Root Mean Squared Error). The example in the code above demonstrates this approach.

**Folder Structure and base\_path:**

The base\_path = ".." is crucial. It assumes your Jupyter Notebook (.ipynb file) is located in a notebook directory, which is one level below the root of your AIMLops project directory (where the data directory resides). Adjust the base\_path if your folder structure is different.

**4. Example Runs (5 Examples):**

These examples illustrate how you can use the code above to process different subsets of your data or perform different analyses.

**Example 1: Filter by Target**

# Inside the notebook, after loading the data:  
  
# Filter for a specific target (replace with a real target\_chembl\_id)  
target\_id = 'CHEMBL205' # Example Target  
df\_filtered = df[df['target\_chembl\_id'] == target\_id].copy() # Create a copy to avoid SettingWithCopyWarning  
  
# Calculate descriptors for the filtered data  
df\_filtered['descriptors'] = df\_filtered['canonical\_smiles'].apply(calculate\_descriptors)  
df\_filtered = df\_filtered.dropna(subset=['descriptors']) # Remove rows where descriptor calculation failed  
df\_filtered = pd.concat([df\_filtered.drop(['descriptors'], axis=1), df\_filtered['descriptors'].apply(pd.Series)], axis=1)  
  
print(f"Number of compounds for target {target\_id}: {len(df\_filtered)}")  
print(df\_filtered.head())

**Example 2: Filter by Activity Value Range**

# Inside the notebook, after loading the data:  
  
# Filter for compounds with IC50 values within a specific range  
min\_ic50 = 10  
max\_ic50 = 100  
df\_filtered = df[(df['standard\_value'] >= min\_ic50) & (df['standard\_value'] <= max\_ic50)].copy() # Create a copy  
  
# Calculate descriptors for the filtered data  
df\_filtered['descriptors'] = df\_filtered['canonical\_smiles'].apply(calculate\_descriptors)  
df\_filtered = df\_filtered.dropna(subset=['descriptors']) # Remove rows where descriptor calculation failed  
df\_filtered = pd.concat([df\_filtered.drop(['descriptors'], axis=1), df\_filtered['descriptors'].apply(pd.Series)], axis=1)  
  
print(f"Number of compounds with IC50 between {min\_ic50} and {max\_ic50} nM: {len(df\_filtered)}")  
print(df\_filtered.head())

**Example 3: Calculate More RDKit Descriptors**

# Modify the calculate\_descriptors function:  
  
from rdkit.Chem import Lipinski  
  
def calculate\_descriptors(smiles):  
 mol = Chem.MolFromSmiles(smiles)  
 if mol is None:  
 return None  
 descriptors = {}  
 descriptors['MW'] = Descriptors.MolWt(mol)  
 descriptors['LogP'] = Descriptors.MolLogP(mol)  
 descriptors['HBD'] = Descriptors.NumHDonors(mol)  
 descriptors['HBA'] = Descriptors.NumHAcceptors(mol)  
 descriptors['TPSA'] = Descriptors.TPSA(mol)  
 descriptors['RotatableBonds'] = Descriptors.NumRotatableBonds(mol)  
 descriptors['Lipinski\_HBA'] = Lipinski.NumHAcceptors(mol)  
 descriptors['Lipinski\_HBD'] = Lipinski.NumHDonors(mol)  
 return descriptors

Then, run the descriptor calculation as before:

# Apply descriptor calculation  
df['descriptors'] = df['canonical\_smiles'].apply(calculate\_descriptors)  
df = df.dropna(subset=['descriptors']) # Remove rows where descriptor calculation failed  
df = pd.concat([df.drop(['descriptors'], axis=1), df['descriptors'].apply(pd.Series)], axis=1)  
print(df.head())

**Example 4: Save the Processed Data**

# After processing the data (including descriptor calculation):  
  
output\_path = os.path.join(base\_path, "data", "Topic\_CheMBL\_35\_49\_processed.csv")  
df.to\_csv(output\_path, index=False)  
print(f"Processed data saved to: {output\_path}")

**Example 5: Basic Statistics of Descriptors**

# After descriptor calculation:  
  
print("Descriptive Statistics for Molecular Weight (MW):")  
print(df['MW'].describe())  
  
print("\nDescriptive Statistics for LogP:")  
print(df['LogP'].describe())  
  
# Visualize the distribution of LogP (requires matplotlib or seaborn)  
import matplotlib.pyplot as plt  
plt.hist(df['LogP'], bins=20)  
plt.xlabel("LogP")  
plt.ylabel("Frequency")  
plt.title("Distribution of LogP Values")  
plt.show()

**Important Considerations:**

* **Error Handling:** The code includes basic error handling (e.g., checking for file existence, handling invalid SMILES). Expand this error handling to make your code more robust.
* **Memory Management:** When working with larger datasets, be mindful of memory usage. Consider using techniques like chunking to process the data in smaller pieces.
* **Data Validation:** Always validate your data to ensure its quality and consistency.
* **Documentation:** Add comments to your code to explain what it does.

**In Summary (En Español):**

Has estado trabajando en un proyecto de análisis de datos de ChEMBL 35 usando RDKit, siguiendo una estructura tipo AIMLops. Has encontrado algunos errores con tus consultas SQL y la versión de scikit-learn. Para arreglar el error de SQL, necesitas convertir el valor standard\_value a texto antes de aplicar la expresión regular. Para el error de scikit-learn, usa la función mean\_squared\_error y luego calcula la raíz cuadrada para obtener el RMSE. Los ejemplos te muestran cómo filtrar los datos por objetivo, rango de valores de actividad y cómo calcular descriptores moleculares adicionales. También se te muestra cómo guardar los datos procesados y obtener estadísticas básicas de los descriptores. Recuerda ajustar las rutas de los archivos (base\_path) según tu estructura de carpetas.

**In English:**

You’ve been working on a ChEMBL 35 data analysis project using RDKit, following an AIMLops-style structure. You’ve encountered some errors with your SQL queries and scikit-learn version. To fix the SQL error, you need to cast the standard\_value to text before applying the regular expression. For the scikit-learn error, use the mean\_squared\_error function and then calculate the square root to get the RMSE. The examples show you how to filter the data by target, activity value range, and how to calculate additional molecular descriptors. You’re also shown how to save the processed data and obtain basic statistics of the descriptors. Remember to adjust the file paths (base\_path) according to your folder structure.

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