PharmApp Suite

🧠 AI for Drug Discovery and Development 🍼

🧩 Topic: Pharmacovigilance risk signals in ChEMBL

Okay, I understand. You're working with Chembl 35 data, using PostgreSQL and RDKit, and following an AIMLops-like directory structure. You need help structuring your code, addressing errors, and implementing a specific workflow. Let's break down the process, incorporating the "Topic_CheMBL_35_57" code and limiting the data to 100 rows.

Here's a comprehensive guide in both English and Vietnamese:

English Version

1. Analysis Model & Workflow

The overall analysis involves extracting relevant data from the Chembl 35 database using SQL, cleaning and transforming it using Python (RDKit and scikit-learn), and then performing your desired analysis. Given the "Topic_CheMBL_35_57," I assume you are investigating a specific area within Chembl, but I will give a set of diverse examples. Here's the proposed workflow:

- **Data Extraction (SQL):**
  - Connect to the Chembl 35 database using the provided credentials.
  - Write SQL queries to extract the necessary data (e.g., molecules, activities, targets). Limit the results to 100 rows for faster processing and testing.
  - Save the extracted data as CSV files in the `../data/` directory.
- **Data Loading and Processing (Python):**
  - Use Python to load the CSV files into Pandas DataFrames.
  - Clean and pre-process the data. This might involve:
    - Handling missing values.
    - Converting data types.
    - Filtering data based on specific criteria.
  - Use RDKit to generate molecular descriptors (e.g., fingerprints, physicochemical properties).
  - Prepare the data for analysis (e.g., feature scaling, train-test split).
- **Analysis (Python):**
  - Perform the analysis relevant to your specific research question (Topic_CheMBL_35_57). Examples below.
  - Use scikit-learn or other libraries to build and evaluate your models.
  - Visualize the results.

2. Code Implementation

Directory Structure (AIMLops-Inspired):

```
project_root/
├── data/
│   └── extracted_data_1.csv
│   └── extracted_data_2.csv
│   └── ...
├── notebooks/
│   └── Topic_CheMBL_35_57_1_data_extraction_and_preprocessing.ipynb
│   └── Topic_CheMBL_35_57_2_analysis.ipynb
```

```
├── src/  # Optional, for reusable modules
│   └── utils.py
└── README.md
```

2.1. SQL Code (for pgAdmin and saving to CSV)

```sql
-- File: Topic_CheMBL_35_57_data_extraction.sql
-- Get only 100 rows

-- Example 1: Extract target, molecule, and activity data for a specific target
SELECT
    act.standard_value,
    act.standard_units,
    mol.molecule_structures,
    target.pref_name
FROM
    activities act
JOIN
    molecule_dictionary mol ON act.molregno = mol.molregno
JOIN
    target_dictionary target ON act.tid = target.tid
WHERE target.pref_name LIKE '%Kinase%'  -- Filtering for Kinases as an example
AND act.standard_type = 'IC50'
AND act.standard_relation = '='
AND act.standard_value IS NOT NULL
AND act.standard_units = 'nM'
AND act.standard_value::text ~ '^[0-9\.]+$' -- Check for numeric values
LIMIT 100;

-- Save this result as a CSV file named extracted_data_1.csv

-- Example 2: Extract data related to a specific assay
SELECT
    act.standard_value,
    act.standard_units,
    mol.molecule_structures,
    assay.description
FROM
    activities act
JOIN
    molecule_dictionary mol ON act.molregno = mol.molregno
JOIN
    assays assay ON act.assay_id = assay.assay_id
WHERE assay.description LIKE '%acetylcholinesterase%' -- Filtering for
Acetylcholinesterase assays
AND act.standard_type = 'IC50'
AND act.standard_relation = '='
AND act.standard_value IS NOT NULL
AND act.standard_units = 'nM'
AND act.standard_value::text ~ '^[0-9\.]+$'  -- Check for numeric values
LIMIT 100;

-- Save this result as a CSV file named extracted_data_2.csv
```

Explanation:

- The `WHERE` clause filters for specific targets (kinases and acetylcholinesterase) or assays. Adjust this based on your research focus (Topic_CheMBL_35_57).
- `act.standard_type = 'IC50'` ensures we're working with IC50 values.
- `act.standard_relation = '='` ensures we only get exact IC50 measurements.

- `act.standard_value IS NOT NULL` excludes entries with missing values.
- `act.standard_units = 'nM'` filters for data reported in nanomolar units.
- `act.standard_value::text ~ '^[0-9\.]+$'` This is important! It casts the `standard_value` to text and then uses a regular expression to check if it contains only numbers and decimal points, addressing the error message you received. This helps ensure that you're only trying to convert valid numeric strings to numbers in your Python code.
- `LIMIT 100` restricts the output to 100 rows.
- The comments indicate how to save the results as CSV files. In pgAdmin, you can usually right-click on the query results and choose "Copy with Headers" and then paste into a CSV file. Alternatively, you can use the `\copy` command in `psql`.

2.2. Python Code (Jupyter Notebook - `Topic_CheMBL_35_57_1_data_extraction_and_preprocessing.ipynb`)

```
import os
import pandas as pd
from rdkit import Chem
from rdkit.Chem import AllChem
from rdkit.Chem import Descriptors
import numpy as np
from sklearn.preprocessing import StandardScaler

# Define base path
base_path = os.path.abspath(os.path.join(os.getcwd(), ".."))  # Go up one level

# Load the data (replace with your actual file names)
data_file_1 = os.path.join(base_path, "data", "extracted_data_1.csv")
data_file_2 = os.path.join(base_path, "data", "extracted_data_2.csv")

try:
    df1 = pd.read_csv(data_file_1)
    df2 = pd.read_csv(data_file_2)
    print("Data loaded successfully!")
except FileNotFoundError:
    print(f"Error: One or both data files not found in the specified directory:
{os.path.join(base_path, 'data')}")
    raise  # Re-raise the exception to halt execution


# Data Cleaning and Preprocessing
# Handle missing values (example: drop rows with missing molecule structures)
df1 = df1.dropna(subset=['molecule_structures'])
df2 = df2.dropna(subset=['molecule_structures'])


# Function to calculate RDKit descriptors
def calculate_descriptors(mol):
    try:
        return Descriptors.CalcMolDescriptors(mol)
    except:
        return None

def calculate_morgan_fingerprint(mol, radius=2, nBits=2048):
    try:
        return AllChem.GetMorganFingerprintAsBitVect(mol, radius, nBits=nBits)
    except:
        return None
```

```
# Generate RDKit molecules and calculate descriptors
def process_dataframe(df):
    df['ROMol'] = df['molecule_structures'].apply(lambda x: Chem.MolFromSmiles(x))
    df = df.dropna(subset=['ROMol']) # remove rows where smiles cannot be parsed
    df['descriptors'] = df['ROMol'].apply(calculate_descriptors)
    df['fingerprint'] = df['ROMol'].apply(calculate_morgan_fingerprint)
    df = df.dropna(subset=['descriptors', 'fingerprint']) # remove rows where
descriptor generation failed
    return df

df1 = process_dataframe(df1)
df2 = process_dataframe(df2)

# Display the first few rows of the processed dataframes
print("Processed Dataframe 1:")
print(df1.head())
print("\nProcessed Dataframe 2:")
print(df2.head())

#Save processed dataframes to new csv files

df1.to_csv(os.path.join(base_path, "data", "processed_data_1.csv"), index=False)
df2.to_csv(os.path.join(base_path, "data", "processed_data_2.csv"), index=False)
```

Explanation:

- **Import Libraries:** Imports necessary libraries (os, pandas, RDKit, scikit-learn).
- **Define `base_path`:** Uses `os.path.join` to construct the correct path to the data directory, adhering to your AIMLops structure.
- **Load Data:** Loads the CSV files into Pandas DataFrames. Includes error handling for `FileNotFoundError`.
- **Data Cleaning:** Handles missing values.
- **RDKit Processing:**
    - Converts SMILES strings to RDKit `ROMol` objects.
    - Calculates molecular descriptors and Morgan fingerprints.
    - Handles potential errors during descriptor calculation (important for robustness).
- **Feature Scaling (Optional):** Scales the descriptors using `StandardScaler` if needed for your analysis.
- **Display and Save:** Prints the first few rows of the processed DataFrames and saves the processed data to new CSV files.

2.3. Python Code (Jupyter Notebook - `Topic_CheMBL_35_57_2_analysis.ipynb`)

```
import os
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.impute import SimpleImputer
from joblib import dump

# Define base path
base_path = os.path.abspath(os.path.join(os.getcwd(), ".."))

# Load the processed data (replace with your actual file names)
data_file_1 = os.path.join(base_path, "data", "processed_data_1.csv")
data_file_2 = os.path.join(base_path, "data", "processed_data_2.csv")
```

```
try:
    df1 = pd.read_csv(data_file_1)
    df2 = pd.read_csv(data_file_2)
    print("Processed data loaded successfully!")
except FileNotFoundError:
    print(f"Error: One or both processed data files not found in the specified
directory: {os.path.join(base_path, 'data')}")
    raise

#Rename columns to avoid conflicts after concat
df1 = df1.rename(columns={'standard_value': 'standard_value_1'})
df2 = df2.rename(columns={'standard_value': 'standard_value_2'})

#Concatenate both dataframes
df = pd.concat([df1, df2], ignore_index=True)

# Prepare data for modeling
# Assuming 'descriptors' and 'standard_value' are in your DataFrames
# Convert descriptors from dictionary to list of values
def extract_descriptor_values(descriptors):
    if isinstance(descriptors, str):
        try:
            descriptors = eval(descriptors) # Convert string representation of dict to
actual dict
            return list(descriptors.values())
        except (SyntaxError, NameError):
            return None  # Handle cases where descriptor string is invalid
    elif isinstance(descriptors, dict):
        return list(descriptors.values())
    else:
        return None

df['descriptor_values'] = df['descriptors'].apply(extract_descriptor_values)
df = df.dropna(subset=['descriptor_values'])

# Convert descriptor values to numeric and create feature matrix X
X = df['descriptor_values'].apply(lambda x: pd.Series(x))
y = df['standard_value_1']  # Using IC50 as target variable;  handle missing values

# Impute missing values (if any)
imputer = SimpleImputer(strategy='mean')
X = imputer.fit_transform(X)

# 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 Random Forest Regressor model
model = RandomForestRegressor(n_estimators=100, random_state=42)
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}")

# Save the trained model
model_path = os.path.join(base_path, "models", "random_forest_model.joblib")
dump(model, model_path)
print(f"Model saved to {model_path}")
```

Explanation:

- **Load Data:** Loads the processed DataFrames.
- **Prepare Data:**
  - Extracts descriptor values from the `descriptors` column.
  - Creates feature matrix `X` from the descriptors.
  - Uses the first `standard_value` as the target `y`
  - Imputes missing values using `SimpleImputer`.
- **Train-Test Split:** Splits the data into training and testing sets.
- **Model Training:**
  - Creates a `RandomForestRegressor` model.
  - Trains the model on the training data.
- **Model Evaluation:**
  - Makes predictions on the test data.
  - Calculates Mean Squared Error (MSE) and R-squared.
- **Model Saving:** Saves the trained model using `joblib`.

3. Addressing the Errors

- **Error a: `ERROR: operator does not exist: numeric ~ unknown, LINE 12: AND act.standard_value ~ '^[0-9\.]+$'`**
  - **Cause:** PostgreSQL might not be able to directly use the `~` (regular expression match) operator on a `numeric` column.
  - **Solution:** Cast the `standard_value` column to `text` before using the regular expression:

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

  - **Explanation:** This converts the numeric value to a string, allowing the regular expression to work correctly. The `^[0-9\.]+$` regular expression checks if the string contains only digits and periods.
- **Error b: `old scikit-learn version does not support parameters squared=False in the mean_squared_error function`**
  - **Cause:** You're using an older version of scikit-learn where the `squared` parameter isn't available in `mean_squared_error`.
  - **Solution:** Remove the `squared=False` parameter. The default behavior is to return the MSE (mean squared error), which is the square root of the RMSE (root mean squared error). If you specifically need RMSE, take the square root of the MSE.

    ```
    mse = mean_squared_error(y_test, y_pred)  # Remove squared=False
    ```

    If you need RMSE, calculate it separately:

    ```
    mse = mean_squared_error(y_test, y_pred)
    rmse = mse**0.5
    print(f"Root Mean Squared Error: {rmse}")
    ```

4. Examples (5 Scenarios)

Here are five different examples of how you might adapt this workflow for different research questions related to Topic_CheMBL_35_57. You'll need to modify the SQL queries and the Python analysis based on the specific question. I will provide example topics so you can use it.

- **Example 1: Activity Prediction for Kinase Inhibitors**
    - **Topic:** Predicting IC50 values for novel kinase inhibitors based on molecular descriptors.
    - **SQL:** Extract data for molecules that inhibit kinase activity. Focus on a specific kinase or a family of kinases.

```sql
SELECT
    act.standard_value,
    mol.molecule_structures
FROM
    activities act
JOIN
    molecule_dictionary mol ON act.molregno = mol.molregno
JOIN
    target_dictionary target ON act.tid = target.tid
WHERE target.pref_name LIKE '%EGFR%'  -- Example: Epidermal Growth Factor Receptor
AND act.standard_type = 'IC50'
AND act.standard_relation = '='
AND act.standard_value IS NOT NULL
AND act.standard_units = 'nM'
AND act.standard_value::text ~ '^[0-9\.]+$'
LIMIT 100;
```

    - **Python (Analysis):** Use the molecular descriptors as features (X) and the IC50 values as the target variable (y). Train a regression model (e.g., Random Forest, Support Vector Regression) to predict IC50 values.
    - **Relevant Code Blocks to Modify:** SQL `WHERE` clause, `y = df['standard_value_1']` in the analysis notebook.

- **Example 2: Structure-Activity Relationship (SAR) Analysis for Acetylcholinesterase Inhibitors**
    - **Topic:** Identifying key molecular features that influence the activity of acetylcholinesterase inhibitors.
    - **SQL:** Extract data for molecules that inhibit acetylcholinesterase.

```sql
SELECT
    act.standard_value,
    mol.molecule_structures
FROM
    activities act
JOIN
    molecule_dictionary mol ON act.molregno = mol.molregno
JOIN
    target_dictionary target ON act.tid = target.tid
WHERE target.pref_name LIKE '%acetylcholinesterase%'
AND act.standard_type = 'IC50'
AND act.standard_relation = '='
AND act.standard_value IS NOT NULL
AND act.standard_units = 'nM'
```

```
AND act.standard_value::text ~ '^[0-9\.]+$'
LIMIT 100;
```

- **Python (Analysis):** Calculate various molecular descriptors. Use feature selection techniques (e.g., SelectKBest, Recursive Feature Elimination) to identify the descriptors that are most strongly correlated with IC50 values. Visualize the relationship between key descriptors and activity.
- **Relevant Code Blocks to Modify:** SQL `WHERE` clause, feature selection in the analysis notebook.

- **Example 3: Comparing Activity Profiles Across Different Assays for the Same Target**
  - **Topic:** Investigating how the activity of a compound against a specific target varies depending on the assay conditions.
  - **SQL:** Extract data for the same target (e.g., a specific kinase) but from different assays.

```
SELECT
    act.standard_value,
    mol.molecule_structures,
    assay.assay_id,
    assay.description
FROM
    activities act
JOIN
    molecule_dictionary mol ON act.molregno = mol.molregno
JOIN
    target_dictionary target ON act.tid = target.tid
JOIN
    assays assay ON act.assay_id = assay.assay_id
WHERE target.pref_name LIKE '%MAPK14%'  -- Example: MAPK14 Kinase
AND act.standard_type = 'IC50'
AND act.standard_relation = '='
AND act.standard_value IS NOT NULL
AND act.standard_units = 'nM'
AND act.standard_value::text ~ '^[0-9\.]+$'
LIMIT 100;
```

  - **Python (Analysis):** Group the data by assay ID. Compare the distribution of IC50 values for each assay. Perform statistical tests (e.g., t-tests, ANOVA) to determine if there are significant differences in activity across different assays.
  - **Relevant Code Blocks to Modify:** SQL `WHERE` clause, grouping and statistical analysis in the analysis notebook.

- **Example 4: Building a Classification Model to Distinguish Active vs. Inactive Compounds**
  - **Topic:** Developing a model to classify compounds as active or inactive against a specific target.
  - **SQL:** Extract data for a specific target (e.g., a protease). Define a threshold for activity (e.g., IC50 < 1000 nM = active, IC50 > 10000 nM = inactive).

```
SELECT
    act.standard_value,
    mol.molecule_structures,
    CASE
        WHEN act.standard_value < 1000 THEN 1  -- Active
        WHEN act.standard_value > 10000 THEN 0 -- Inactive
```

```
            ELSE NULL  -- Exclude intermediate values
        END AS activity_class
    FROM
        activities act
    JOIN
        molecule_dictionary mol ON act.molregno = mol.molregno
    JOIN
        target_dictionary target ON act.tid = target.tid
    WHERE target.pref_name LIKE '%Thrombin%'  -- Example: Thrombin
    AND act.standard_type = 'IC50'
    AND act.standard_relation = '='
    AND act.standard_value IS NOT NULL
    AND act.standard_units = 'nM'
    AND act.standard_value::text ~ '^[0-9\.]+$'
    AND act.standard_value < 100000 -- Limit to values less than 100000
    LIMIT 100;
```

- **Python (Analysis):** Use the molecular descriptors as features (X) and the `activity_class` as the target variable (y). Train a classification model (e.g., Random Forest, Support Vector Machine, Logistic Regression) to predict activity.
- **Relevant Code Blocks to Modify:** SQL `WHERE` clause and `CASE` statement, model type in the analysis notebook.

- **Example 5: Identifying Potential Off-Target Effects**
  - **Topic:** Investigating whether compounds designed for one target also show activity against other targets.
  - **SQL:** Extract data for a set of compounds and their activity against multiple targets. This requires a more complex SQL query.

```
SELECT
    mol.molecule_structures,
    target.pref_name,
    act.standard_value
FROM
    activities act
JOIN
    molecule_dictionary mol ON act.molregno = mol.molregno
JOIN
    target_dictionary target ON act.tid = target.tid
WHERE mol.molregno IN (SELECT molregno FROM activities WHERE tid = (SELECT
tid FROM target_dictionary WHERE pref_name LIKE '%EGFR%') AND
act.standard_value < 1000) -- Select molecules active against EGFR
AND act.standard_type = 'IC50'
AND act.standard_relation = '='
AND act.standard_value IS NOT NULL
AND act.standard_units = 'nM'
AND act.standard_value::text ~ '^[0-9\.]+$'
LIMIT 100;
```

  - **Python (Analysis):** Analyze the activity profiles of the compounds across different targets. Identify compounds that show significant activity against multiple targets. This requires data manipulation to pivot the table so each target becomes a column.
  - **Relevant Code Blocks to Modify:** SQL query, data manipulation and analysis in the analysis notebook.

Key Considerations:

- **Adapt the SQL queries and Python code to match your specific research question (Topic_CheMBL_35_57).** The examples are starting points.
- **Choose appropriate molecular descriptors and machine learning models based on the nature of your data and research question.**
- **Pay close attention to data cleaning and preprocessing.** Missing values and inconsistencies can significantly impact the results.
- **Evaluate the performance of your models using appropriate metrics.**
- **Document your code and analysis thoroughly.**

Vietnamese Version

1. Phân Tích Mô Hình & Quy Trình Làm Việc

Phân tích tổng thể bao gồm trích xuất dữ liệu liên quan từ cơ sở dữ liệu Chembl 35 bằng SQL, làm sạch và chuyển đổi nó bằng Python (RDKit và scikit-learn), và sau đó thực hiện phân tích mong muốn của bạn. Với "Topic_CheMBL_35_57," tôi cho rằng bạn đang điều tra một lĩnh vực cụ thể trong Chembl, nhưng tôi sẽ cung cấp một tập hợp các ví dụ đa dạng. Đây là quy trình làm việc được đề xuất:

- **Trích Xuất Dữ Liệu (SQL):**
  - Kết nối với cơ sở dữ liệu Chembl 35 bằng thông tin đăng nhập được cung cấp.
  - Viết các truy vấn SQL để trích xuất dữ liệu cần thiết (ví dụ: phân tử, hoạt động, mục tiêu). Giới hạn kết quả thành 100 hàng để xử lý và kiểm tra nhanh hơn.
  - Lưu dữ liệu đã trích xuất dưới dạng tệp CSV trong thư mục `../data/`.
- **Tải và Xử Lý Dữ Liệu (Python):**
  - Sử dụng Python để tải các tệp CSV vào Pandas DataFrames.
  - Làm sạch và tiền xử lý dữ liệu. Điều này có thể bao gồm:
    - Xử lý các giá trị bị thiếu.
    - Chuyển đổi các loại dữ liệu.
    - Lọc dữ liệu dựa trên các tiêu chí cụ thể.
  - Sử dụng RDKit để tạo các mô tả phân tử (ví dụ: dấu vân tay, các thuộc tính vật lý hóa học).
  - Chuẩn bị dữ liệu để phân tích (ví dụ: chia tỷ lệ đặc trưng, chia tập huấn luyện-kiểm tra).
- **Phân Tích (Python):**
  - Thực hiện phân tích liên quan đến câu hỏi nghiên cứu cụ thể của bạn (Topic_CheMBL_35_57). Ví dụ bên dưới.
  - Sử dụng scikit-learn hoặc các thư viện khác để xây dựng và đánh giá các mô hình của bạn.
  - Trực quan hóa kết quả.

2. Triển Khai Mã

Cấu Trúc Thư Mục (Lấy Cảm Hứng từ AIMLops):

```
project_root/
├── data/
│   └── extracted_data_1.csv
│   └── extracted_data_2.csv
│   └── ...
├── notebooks/
│   └── Topic_CheMBL_35_57_1_data_extraction_and_preprocessing.ipynb
│   └── Topic_CheMBL_35_57_2_analysis.ipynb
├── src/  # Tùy chọn, cho các mô-đun có thể tái sử dụng
│   └── utils.py
└── README.md
```

2.1. Mã SQL (cho pgAdmin và lưu vào CSV)

```sql
-- File: Topic_CheMBL_35_57_data_extraction.sql
-- Chỉ lấy 100 hàng

-- Ví dụ 1: Trích xuất dữ liệu mục tiêu, phân tử và hoạt động cho một mục tiêu cụ thể
SELECT
    act.standard_value,
    act.standard_units,
    mol.molecule_structures,
    target.pref_name
FROM
    activities act
JOIN
    molecule_dictionary mol ON act.molregno = mol.molregno
JOIN
    target_dictionary target ON act.tid = target.tid
WHERE target.pref_name LIKE '%Kinase%'  -- Lọc cho Kinases như một ví dụ
AND act.standard_type = 'IC50'
AND act.standard_relation = '='
AND act.standard_value IS NOT NULL
AND act.standard_units = 'nM'
AND act.standard_value::text ~ '^[0-9\.]+$' -- Kiểm tra giá trị số
LIMIT 100;

-- Lưu kết quả này dưới dạng tệp CSV có tên extracted_data_1.csv

-- Ví dụ 2: Trích xuất dữ liệu liên quan đến một xét nghiệm cụ thể
SELECT
    act.standard_value,
    act.standard_units,
    mol.molecule_structures,
    assay.description
FROM
    activities act
JOIN
    molecule_dictionary mol ON act.molregno = mol.molregno
JOIN
    assays assay ON act.assay_id = assay.assay_id
WHERE assay.description LIKE '%acetylcholinesterase%' -- Lọc cho xét nghiệm
Acetylcholinesterase
AND act.standard_type = 'IC50'
AND act.standard_relation = '='
AND act.standard_value IS NOT NULL
AND act.standard_units = 'nM'
AND act.standard_value::text ~ '^[0-9\.]+$'  -- Kiểm tra giá trị số
LIMIT 100;

-- Lưu kết quả này dưới dạng tệp CSV có tên extracted_data_2.csv
```

Giải thích:

- Mệnh đề `WHERE` lọc cho các mục tiêu (kinases và acetylcholinesterase) hoặc xét nghiệm cụ thể. Điều chỉnh điều này dựa trên trọng tâm nghiên cứu của bạn (Topic_CheMBL_35_57).
- `act.standard_type = 'IC50'` đảm bảo rằng chúng ta đang làm việc với các giá trị IC50.
- `act.standard_relation = '='` đảm bảo rằng chúng ta chỉ nhận được các phép đo IC50 chính xác.
- `act.standard_value IS NOT NULL` loại trừ các mục có giá trị bị thiếu.
- `act.standard_units = 'nM'` lọc dữ liệu được báo cáo bằng đơn vị nanomolar.
- `act.standard_value::text ~ '^[0-9\.]+$'` Điều này rất quan trọng! Nó chuyển đổi `standard_value` thành văn bản và sau đó sử dụng một biểu thức chính quy để kiểm tra xem nó

chỉ chứa số và dấu thập phân hay không, giải quyết thông báo lỗi bạn nhận được. Điều này giúp đảm bảo rằng bạn chỉ đang cố gắng chuyển đổi các chuỗi số hợp lệ thành số trong mã Python của bạn.

- `LIMIT 100` giới hạn đầu ra thành 100 hàng.
- Các nhận xét cho biết cách lưu kết quả dưới dạng tệp CSV. Trong pgAdmin, bạn thường có thể nhấp chuột phải vào kết quả truy vấn và chọn "Copy with Headers", sau đó dán vào tệp CSV. Ngoài ra, bạn có thể sử dụng lệnh `\copy` trong `psql`.

2.2. Mã Python (Jupyter Notebook - Topic_CheMBL_35_57_1_data_extraction_and_preprocessing.ipynb)

```
import os
import pandas as pd
from rdkit import Chem
from rdkit.Chem import AllChem
from rdkit.Chem import Descriptors
import numpy as np
from sklearn.preprocessing import StandardScaler

# Xác định đường dẫn gốc
base_path = os.path.abspath(os.path.join(os.getcwd(), ".."))  # Đi lên một cấp

# Tải dữ liệu (thay thế bằng tên tệp thực tế của bạn)
data_file_1 = os.path.join(base_path, "data", "extracted_data_1.csv")
data_file_2 = os.path.join(base_path, "data", "extracted_data_2.csv")

try:
    df1 = pd.read_csv(data_file_1)
    df2 = pd.read_csv(data_file_2)
    print("Dữ liệu đã được tải thành công!")
except FileNotFoundError:
    print(f"Lỗi: Một hoặc cả hai tệp dữ liệu không được tìm thấy trong thư mục được
chỉ định: {os.path.join(base_path, 'data')}")
    raise  # Gây ra lại ngoại lệ để dừng thực thi


# Làm sạch và Tiền xử lý Dữ liệu
# Xử lý các giá trị bị thiếu (ví dụ: loại bỏ các hàng có cấu trúc phân tử bị thiếu)
df1 = df1.dropna(subset=['molecule_structures'])
df2 = df2.dropna(subset=['molecule_structures'])


# Hàm tính toán các mô tả RDKit
def calculate_descriptors(mol):
    try:
        return Descriptors.CalcMolDescriptors(mol)
    except:
        return None

def calculate_morgan_fingerprint(mol, radius=2, nBits=2048):
    try:
        return AllChem.GetMorganFingerprintAsBitVect(mol, radius, nBits=nBits)
    except:
        return None

# Tạo các phân tử RDKit và tính toán các mô tả
def process_dataframe(df):
    df['ROMol'] = df['molecule_structures'].apply(lambda x: Chem.MolFromSmiles(x))
    df = df.dropna(subset=['ROMol']) # Xóa các hàng mà smiles không thể được phân tích
```

```
cú pháp
    df['descriptors'] = df['ROMol'].apply(calculate_descriptors)
    df['fingerprint'] = df['ROMol'].apply(calculate_morgan_fingerprint)
    df = df.dropna(subset=['descriptors', 'fingerprint']) # Xóa các hàng mà việc tạo
mô tả thất bại
    return df

df1 = process_dataframe(df1)
df2 = process_dataframe(df2)

# Hiển thị một vài hàng đầu tiên của các dataframe đã xử lý
print("Dataframe đã xử lý 1:")
print(df1.head())
print("\nDataframe đã xử lý 2:")
print(df2.head())

#Lưu dataframe đã xử lý vào tệp csv mới

df1.to_csv(os.path.join(base_path, "data", "processed_data_1.csv"), index=False)
df2.to_csv(os.path.join(base_path, "data", "processed_data_2.csv"), index=False)
```

Giải thích:

- **Nhập Thư Viện:** Nhập các thư viện cần thiết (os, pandas, RDKit, scikit-learn).
- **Xác định `base_path`:** Sử dụng `os.path.join` để xây dựng đường dẫn chính xác đến thư mục dữ liệu, tuân thủ cấu trúc AIMLops của bạn.
- **Tải Dữ Liệu:** Tải các tệp CSV vào Pandas DataFrames. Bao gồm xử lý lỗi cho `FileNotFoundError`.
- **Làm Sạch Dữ Liệu:** Xử lý các giá trị bị thiếu.
- **Xử Lý RDKit:**
    - Chuyển đổi chuỗi SMILES thành các đối tượng `ROMol` RDKit.