PharmApp Suite

🧠 AI for Drug Discovery and Development 💉

🧩 Topic: ChEMBL → Real-world evidence mapping

Okay, I understand. You're working with ChEMBL 35 data in PostgreSQL, using RDKit for analysis, and following an AIMLops-inspired folder structure. You're encountering errors related to SQL syntax and scikit-learn version compatibility, and you need guidance on how to structure your project and code, limited to 100 rows of data for demonstration.

Let's break this down step-by-step, providing code examples in both SQL and Python, and addressing the errors you're facing. I will provide the analysis and solutions in both English and Vietnamese.

1. Project Setup and Folder Structure

Following the AIMLops principle, let's assume a simplified project structure like this:

```
project_root/
├── data/           # Stores CSV files extracted from the database
├── notebooks/       # Jupyter notebooks with your analysis code
├── src/            # (Optional) Python modules for reusable code
├── models/        #  (Optional) Saved model files
├── README.md
```

This structure helps in organizing your data, code, and models.

2. Addressing the SQL Error: `ERROR: operator does not exist: numeric ~ unknown`

This error occurs because the ~ operator in PostgreSQL is used for regular expression matching on *text* data, not *numeric* data. You are trying to use it on `act.standard_value`, which is likely a numeric column. The correct approach is to either cast the numeric column to text or, more preferably, use a numerical comparison (e.g., >=, <=). However, your intent is to filter for values that look like valid numbers. A better approach to filter invalid data is to use `standard_value IS NOT NULL`

3. SQL Code (to be run in pgAdmin)

This SQL query will extract 100 rows from the ChEMBL database, focusing on activities and molecules. This will generate a CSV file named `Topic_CheMBL_35_60.csv` in your `data/` folder.

```sql
-- SQL Code (Topic_CheMBL_35_60.sql)
-- Retrieve data and save as Topic_CheMBL_35_60.csv
COPY (
  SELECT
    act.activity_id,
    act.standard_type,
    act.standard_value,
    act.standard_units,
    act.assay_id,
    mol.molregno,
    mol.smiles
  FROM activities act
  JOIN assays ass ON act.assay_id = ass.assay_id
  JOIN target_dictionary td ON ass.tid = td.tid
  JOIN molecule_dictionary mol ON act.molregno = mol.molregno
  WHERE act.standard_type = 'IC50'  -- Example: Filter for IC50 values
  AND act.standard_relation = '='    -- Example: Filter for exact values
  AND act.standard_value IS NOT NULL
  AND act.standard_units = 'nM' -- Filter for values in nM
```

```
  LIMIT 100
)
TO '/tmp/Topic_CheMBL_35_60.csv'  -- Adjust path to your server's /tmp directory.
Important for permissions!
WITH CSV HEADER DELIMITER ',';
```

Important notes about the SQL code:

- **Path:** Change `/tmp/Topic_CheMBL_35_60.csv` to a directory where PostgreSQL has write permissions on the server. `/tmp` is often a good choice for testing. If you don't have permissions, you'll need to grant them or choose a different location.
- **Error Handling:** This SQL code assumes a smooth execution. In a real-world scenario, you'd add error handling (e.g., `TRY...CATCH` blocks) and logging.
- **Filtering:** Adjust the `WHERE` clause to select the specific data you need. The provided example filters for `IC50` values, exact relations (=), only valid standard values (NOT NULL) and for values in `nM`. You can adjust this based on your research question.
- **`LIMIT` Clause:** The `LIMIT 100` clause restricts the output to 100 rows. Remove or adjust this if needed.

Running the SQL:

1. Open pgAdmin.
2. Connect to your database (ip: 192.168.206.136, user: rd, pass: rd, database: chembl_35).
3. Open a new query window.
4. Paste the SQL code into the query window.
5. Execute the query.
6. The CSV file `Topic_CheMBL_35_60.csv` will be created on the *server* at the location you specified (e.g., `/tmp`). You'll need to copy it to your local `data/` folder. A common way to do this is using `scp` from your local machine:

```
scp rd@192.168.206.136:/tmp/Topic_CheMBL_35_60.csv data/
```

4. Python Code (Jupyter Notebook)

Now, let's create a Jupyter Notebook (`notebooks/Topic_CheMBL_35_60_1_Data_Loading.ipynb`) to load and inspect the data.

```
# Topic_CheMBL_35_60_1_Data_Loading.ipynb
import pandas as pd
import os
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 matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import StandardScaler

# Define the base path to your project
base_path = os.getcwd() # Gets the current working directory
data_path = os.path.join(base_path, "data")
csv_file = os.path.join(data_path, "Topic_CheMBL_35_60.csv")

# Load the data
try:
    df = pd.read_csv(csv_file)
    print("Data loaded successfully.")
```

```
except FileNotFoundError:
    print(f"Error: File not found at {csv_file}.  Make sure you've copied the file from the server.")
    df = None  # Or raise the exception if you want to stop execution

if df is not None:
    print(df.head())
    print(df.info())

    # Data Cleaning
    # Handle missing values (example: fill with the mean)
    df['standard_value'] = pd.to_numeric(df['standard_value'], errors='coerce')  # Convert to numeric, coerce errors to NaN
    df['standard_value'].fillna(df['standard_value'].mean(), inplace=True) # Fill missing values with mean

    # Remove rows with missing SMILES
    df = df.dropna(subset=['smiles'])
    print("Data cleaning completed.")

    # Feature Engineering with RDKit
    def calculate_molecular_weight(smiles):
        mol = Chem.MolFromSmiles(smiles)
        if mol:
            return Descriptors.MolWt(mol)
        else:
            return None

    df['molecular_weight'] = df['smiles'].apply(calculate_molecular_weight)
    df = df.dropna(subset=['molecular_weight']) # Remove rows where molecular weight calculation failed
    print("Feature engineering completed.")

    print(df.head())
```

Explanation of the Python Code:

1. **Imports:** Imports necessary libraries (pandas, os, RDKit, scikit-learn).
2. **Path Handling:** Uses `os.path.join` to construct file paths robustly.
3. **Data Loading:** Loads the CSV file into a pandas DataFrame. Includes a `try...except` block to handle the `FileNotFoundError`.
4. **Data Cleaning:**
    - **Missing Values:** Converts 'standard_value' to numeric and fills missing values with the mean.
    - **SMILES Handling:** Removes rows with missing SMILES strings to prevent errors in RDKit.
5. **Feature Engineering (RDKit):**
    - **Molecular Weight:** Calculates the molecular weight using RDKit and adds it as a new column.
    - **Error Handling:** The `calculate_molecular_weight` function includes a check to handle invalid SMILES strings. The code also removes rows where the molecular weight calculation failed.
6. **Prints:** Prints the first few rows and info to verify the data loading and processing.

Example 2: Further Analysis and Modeling (Topic_CheMBL_35_60_2_Modeling.ipynb)

```
# Topic_CheMBL_35_60_2_Modeling.ipynb
import pandas as pd
```

```
import os
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 matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import StandardScaler

# Define the base path to your project
base_path = os.getcwd() # Gets the current working directory
data_path = os.path.join(base_path, "data")
csv_file = os.path.join(data_path, "Topic_CheMBL_35_60.csv")


# Load the data (same as before, but wrapped in a function)
def load_and_preprocess_data(csv_file):
    try:
        df = pd.read_csv(csv_file)
        print("Data loaded successfully.")
    except FileNotFoundError:
        print(f"Error: File not found at {csv_file}.  Make sure you've copied the file
from the server.")
        return None

    # Data Cleaning
    df['standard_value'] = pd.to_numeric(df['standard_value'], errors='coerce')
    df['standard_value'].fillna(df['standard_value'].mean(), inplace=True)
    df = df.dropna(subset=['smiles'])

    # Feature Engineering with RDKit
    def calculate_molecular_weight(smiles):
        mol = Chem.MolFromSmiles(smiles)
        if mol:
            return Descriptors.MolWt(mol)
        else:
            return None

    df['molecular_weight'] = df['smiles'].apply(calculate_molecular_weight)
    df = df.dropna(subset=['molecular_weight'])
    print("Data loading and preprocessing completed.")
    return df

df = load_and_preprocess_data(csv_file)

if df is not None:
    # Prepare data for modeling
    X = df[['molecular_weight']] # Features
    y = df['standard_value']      # Target variable

    # Data Scaling
    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)
```

```
    # Train a linear regression model
    model = LinearRegression()
    model.fit(X_train, y_train)

    # Make predictions
    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}")

    # Visualization (Scatter plot of actual vs. predicted values)
    plt.figure(figsize=(8, 6))
    plt.scatter(y_test, y_pred)
    plt.xlabel("Actual Values")
    plt.ylabel("Predicted Values")
    plt.title("Actual vs. Predicted Values")
    plt.show()
```

Explanation:

1. **Modularization:** The data loading and preprocessing steps are now encapsulated in a function `load_and_preprocess_data`. This promotes code reusability.
2. **Feature Selection:** The code selects `molecular_weight` as the feature (X) and `standard_value` as the target variable (y).
3. **Data Scaling:** Uses `StandardScaler` to scale the features. This is important for linear regression and other models that are sensitive to feature scaling.
4. **Train/Test Split:** Splits the data into training and testing sets using `train_test_split`.
5. **Model Training:** Trains a linear regression model using `LinearRegression`.
6. **Model Evaluation:**
    - Calculates the Mean Squared Error (MSE) and R-squared (R2) to evaluate the model's performance.
    - Prints the evaluation metrics.
7. **Visualization:** Creates a scatter plot of actual vs. predicted values to visualize the model's performance.

Addressing the `squared=False` Error

The error "old scikit-learn version does not support parameters squared=False in the mean_squared_error function" indicates you are likely 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:

The simplest solution is to upgrade your scikit-learn version:

```
pip install --upgrade scikit-learn
```

If upgrading is not an option (e.g., due to environment constraints), you can calculate the RMSE manually:

```
from math import sqrt

mse = mean_squared_error(y_test, y_pred)
```

```
rmse = sqrt(mse)  # Calculate RMSE manually
print(f"Root Mean Squared Error: {rmse}")
```

5. Example Usage and Next Steps

You now have a basic framework for analyzing ChEMBL data using RDKit and scikit-learn. Here are some ideas for expanding this:

Example 3: Adding More Features

Expand the feature engineering to include other molecular descriptors from RDKit (e.g., LogP, number of hydrogen bond donors/acceptors).

```
def calculate_descriptors(smiles):
    mol = Chem.MolFromSmiles(smiles)
    if mol:
        mw = Descriptors.MolWt(mol)
        logp = Chem.Crippen.MolLogP(mol)
        hbd = Descriptors.NumHDonors(mol)
        hba = Descriptors.NumHAcceptors(mol)
        return mw, logp, hbd, hba
    else:
        return None, None, None, None

df[['molecular_weight', 'logp', 'hbd', 'hba']] = df['smiles'].apply(lambda x:
pd.Series(calculate_descriptors(x)))
df = df.dropna() # Remove rows with NaN values after descriptor calculation
```

Example 4: Trying Different Models

Experiment with different machine learning models, such as Random Forest Regressor or Support Vector Regressor.

```
from sklearn.ensemble import RandomForestRegressor

model = RandomForestRegressor(n_estimators=100, random_state=42) # Example: Random
Forest
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
print(f"Random Forest - Mean Squared Error: {mse}")
print(f"Random Forest - R-squared: {r2}")
```

Example 5: Hyperparameter Tuning

Use techniques like GridSearchCV or RandomizedSearchCV to optimize the hyperparameters of your chosen model.

```
from sklearn.model_selection import GridSearchCV

param_grid = {
    'n_estimators': [50, 100, 200],
    'max_depth': [None, 5, 10],
    'min_samples_split': [2, 5, 10]
}

grid_search = GridSearchCV(RandomForestRegressor(random_state=42), param_grid, cv=3,
scoring='neg_mean_squared_error')
grid_search.fit(X_train, y_train)
best_model = grid_search.best_estimator_
y_pred = best_model.predict(X_test)
```

```
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
print(f"Tuned Random Forest - Mean Squared Error: {mse}")
print(f"Tuned Random Forest - R-squared: {r2}")
```

Further Steps:

- **Data Exploration:** Spend more time exploring the data to understand its distributions and relationships.
- **Feature Selection:** Use more sophisticated feature selection techniques to identify the most relevant features.
- **Model Validation:** Use cross-validation to get a more robust estimate of your model's performance.
- **Deployment:** Consider how you might deploy your model to make predictions on new compounds.

Vietnamese Translation:

1. Phân tích và Hướng dẫn

Chào bạn, tôi hiểu rằng bạn đang làm việc với dữ liệu ChEMBL 35 trong PostgreSQL, sử dụng RDKit để phân tích, và tuân theo cấu trúc thư mục kiểu AIMLops. Bạn đang gặp lỗi liên quan đến cú pháp SQL và khả năng tương thích phiên bản của scikit-learn, và bạn cần hướng dẫn về cách cấu trúc dự án và mã của mình, giới hạn ở 100 hàng dữ liệu để minh họa.

Chúng ta sẽ phân tích từng bước, cung cấp các ví dụ mã bằng cả SQL và Python, và giải quyết các lỗi bạn đang gặp phải.

2. Thiết lập Dự án và Cấu trúc Thư mục

Theo nguyên tắc AIMLops, hãy giả định một cấu trúc dự án đơn giản như sau:

```
project_root/
├── data/            # Lưu trữ các tệp CSV được trích xuất từ cơ sở dữ liệu
├── notebooks/        # Sổ tay Jupyter với mã phân tích của bạn
├── src/             # (Tùy chọn) Các mô-đun Python cho mã có thể tái sử dụng
├── models/         #  (Tùy chọn) Các tệp mô hình đã lưu
├── README.md
```

Cấu trúc này giúp tổ chức dữ liệu, mã và mô hình của bạn.

3. Giải quyết Lỗi SQL: `ERROR: operator does not exist: numeric ~ unknown`

Lỗi này xảy ra vì toán tử ~ trong PostgreSQL được sử dụng để so khớp biểu thức chính quy trên dữ liệu *văn bản*, không phải dữ liệu *số*. Bạn đang cố gắng sử dụng nó trên `act.standard_value`, có khả năng là một cột số. Cách tiếp cận chính xác là chuyển đổi cột số thành văn bản hoặc, tốt hơn là, sử dụng so sánh số (ví dụ: >=, <=). Tuy nhiên, mục đích của bạn là lọc các giá trị trông giống như số hợp lệ. Một cách tiếp cận tốt hơn để lọc dữ liệu không hợp lệ là sử dụng `standard_value IS NOT NULL`

4. Mã SQL (chạy trong pgAdmin)

Truy vấn SQL này sẽ trích xuất 100 hàng từ cơ sở dữ liệu ChEMBL, tập trung vào các hoạt động và phân tử. Thao tác này sẽ tạo ra một tệp CSV có tên `Topic_CheMBL_35_60.csv` trong thư mục `data/` của bạn.

```
-- Mã SQL (Topic_CheMBL_35_60.sql)
-- Truy xuất dữ liệu và lưu dưới dạng Topic_CheMBL_35_60.csv
COPY (
  SELECT
    act.activity_id,
    act.standard_type,
```

```
    act.standard_value,
    act.standard_units,
    act.assay_id,
    mol.molregno,
    mol.smiles
  FROM activities act
  JOIN assays ass ON act.assay_id = ass.assay_id
  JOIN target_dictionary td ON ass.tid = td.tid
  JOIN molecule_dictionary mol ON act.molregno = mol.molregno
  WHERE act.standard_type = 'IC50'  -- Ví dụ: Lọc cho các giá trị IC50
  AND act.standard_relation = '='    -- Ví dụ: Lọc cho các giá trị chính xác
  AND act.standard_value IS NOT NULL
  AND act.standard_units = 'nM' -- Lọc cho các giá trị tính bằng nM
  LIMIT 100
)
TO '/tmp/Topic_CheMBL_35_60.csv'  -- Điều chỉnh đường dẫn đến thư mục /tmp của máy chủ của bạn. Quan trọng đối với quyền!
WITH CSV HEADER DELIMITER ',';
```

Lưu ý quan trọng về mã SQL:

- **Đường dẫn:** Thay đổi `/tmp/Topic_CheMBL_35_60.csv` thành một thư mục mà PostgreSQL có quyền ghi trên máy chủ. `/tmp` thường là một lựa chọn tốt để thử nghiệm. Nếu bạn không có quyền, bạn sẽ cần cấp chúng hoặc chọn một vị trí khác.
- **Xử lý Lỗi:** Mã SQL này giả định một quá trình thực thi suôn sẻ. Trong một tình huống thực tế, bạn sẽ thêm xử lý lỗi (ví dụ: khối `TRY...CATCH`) và ghi nhật ký.
- **Lọc:** Điều chỉnh mệnh đề `WHERE` để chọn dữ liệu cụ thể bạn cần. Ví dụ được cung cấp lọc cho các giá trị `IC50`, quan hệ chính xác (=), chỉ các giá trị tiêu chuẩn hợp lệ (NOT NULL) và cho các giá trị tính bằng `nM`. Bạn có thể điều chỉnh điều này dựa trên câu hỏi nghiên cứu của bạn.
- **Mệnh đề `LIMIT`:** Mệnh đề `LIMIT 100` giới hạn đầu ra thành 100 hàng. Xóa hoặc điều chỉnh điều này nếu cần.

Chạy SQL:

1. Mở pgAdmin.
2. Kết nối với cơ sở dữ liệu của bạn (ip: 192.168.206.136, user: rd, pass: rd, database: chembl_35).
3. Mở một cửa sổ truy vấn mới.
4. Dán mã SQL vào cửa sổ truy vấn.
5. Thực thi truy vấn.
6. Tệp CSV `Topic_CheMBL_35_60.csv` sẽ được tạo trên *máy chủ* tại vị trí bạn đã chỉ định (ví dụ: `/tmp`). Bạn sẽ cần sao chép nó vào thư mục `data/` cục bộ của bạn. Một cách phổ biến để thực hiện việc này là sử dụng `scp` từ máy cục bộ của bạn:

```
scp rd@192.168.206.136:/tmp/Topic_CheMBL_35_60.csv data/
```

5. Mã Python (Sổ tay Jupyter)

Bây giờ, hãy tạo một Sổ tay Jupyter (`notebooks/Topic_CheMBL_35_60_1_Data_Loading.ipynb`) để tải và kiểm tra dữ liệu.

```
# Topic_CheMBL_35_60_1_Data_Loading.ipynb
import pandas as pd
import os
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
```

```python
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import StandardScaler

# Xác định đường dẫn cơ sở đến dự án của bạn
base_path = os.getcwd()  # Lấy thư mục làm việc hiện tại
data_path = os.path.join(base_path, "data")
csv_file = os.path.join(data_path, "Topic_CheMBL_35_60.csv")

# Tải dữ liệu
try:
    df = pd.read_csv(csv_file)
    print("Dữ liệu đã được tải thành công.")
except FileNotFoundError:
    print(f"Lỗi: Không tìm thấy tệp tại {csv_file}. Đảm bảo bạn đã sao chép tệp từ máy chủ.")
    df = None  # Hoặc đưa ra ngoại lệ nếu bạn muốn dừng thực thi

if df is not None:
    print(df.head())
    print(df.info())

    # Làm sạch dữ liệu
    # Xử lý các giá trị bị thiếu (ví dụ: điền vào bằng giá trị trung bình)
    df['standard_value'] = pd.to_numeric(df['standard_value'], errors='coerce')  # Chuyển đổi sang số, ép các lỗi thành NaN
    df['standard_value'].fillna(df['standard_value'].mean(), inplace=True)  # Điền các giá trị bị thiếu bằng giá trị trung bình

    # Xóa các hàng có SMILES bị thiếu
    df = df.dropna(subset=['smiles'])
    print("Đã hoàn thành làm sạch dữ liệu.")

    # Kỹ thuật đặc trưng với RDKit
    def calculate_molecular_weight(smiles):
        mol = Chem.MolFromSmiles(smiles)
        if mol:
            return Descriptors.MolWt(mol)
        else:
            return None

    df['molecular_weight'] = df['smiles'].apply(calculate_molecular_weight)
    df = df.dropna(subset=['molecular_weight'])  # Xóa các hàng mà tính toán trọng lượng phân tử không thành công
    print("Đã hoàn thành kỹ thuật đặc trưng.")

    print(df.head())
```

Giải thích về Mã Python:

1. **Nhập:** Nhập các thư viện cần thiết (pandas, os, RDKit, scikit-learn).
2. **Xử lý Đường dẫn:** Sử dụng `os.path.join` để xây dựng các đường dẫn tệp một cách mạnh mẽ.
3. **Tải Dữ liệu:** Tải tệp CSV vào một DataFrame pandas. Bao gồm một khối `try...except` để xử lý `FileNotFoundError`.
4. **Làm sạch Dữ liệu:**
    - **Giá trị Bị thiếu:** Chuyển đổi 'standard_value' thành số và điền các giá trị bị thiếu bằng giá trị trung bình.
    - **Xử lý SMILES:** Xóa các hàng có chuỗi SMILES bị thiếu để ngăn chặn lỗi trong RDKit.

5. **Kỹ thuật Đặc trưng (RDKit):**
    - **Trọng lượng Phân tử:** Tính trọng lượng phân tử bằng RDKit và thêm nó làm một cột mới.
    - **Xử lý Lỗi:** Hàm `calculate_molecular_weight` bao gồm một kiểm tra để xử lý các chuỗi SMILES không hợp lệ. Mã này cũng xóa các hàng mà tính toán trọng lượng phân tử không thành công.
6. **In:** In một vài hàng đầu tiên và thông tin để xác minh việc tải và xử lý dữ liệu.

Ví dụ 2: Phân tích và Mô hình hóa Thêm (Topic_CheMBL_35_60_2_Modeling.ipynb)

```
# Topic_CheMBL_35_60_2_Modeling.ipynb
import pandas as pd
import os
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 matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import StandardScaler

# Xác định đường dẫn cơ sở đến dự án của bạn
base_path = os.getcwd()  # Lấy thư mục làm việc hiện tại
data_path = os.path.join(base_path, "data")
csv_file = os.path.join(data_path, "Topic_CheMBL_35_60.csv")


# Tải dữ liệu (giống như trước đây, nhưng được gói trong một hàm)
def load_and_preprocess_data(csv_file):
    try:
        df = pd.read_csv(csv_file)
        print("Dữ liệu đã được tải thành công.")
    except FileNotFoundError:
        print(f"Lỗi: Không tìm thấy tệp tại {csv_file}. Đảm bảo bạn đã sao chép tệp từ
máy chủ.")
        return None

    # Làm sạch dữ liệu
    df['standard_value'] = pd.to_numeric(df['standard_value'], errors='coerce')
    df['standard_value'].fillna(df['standard_value'].mean(), inplace=True)
    df = df.dropna(subset=['smiles'])

    # Kỹ thuật đặc trưng với RDKit
    def calculate_molecular_weight(smiles):
        mol = Chem.MolFromSmiles(smiles)
        if mol:
            return Descriptors.MolWt(mol)
        else:
            return None

    df['molecular_weight'] = df['smiles'].apply(calculate_molecular_weight)
    df = df.dropna(subset=['molecular_weight'])
    print("Đã hoàn thành tải và tiền xử lý dữ liệu.")
    return df

df = load_and_preprocess_data(csv_file)
```

```
if df is not None:
    # Chuẩn bị dữ liệu cho mô hình hóa
    X = df[['molecular_weight']]  # Các đặc trưng
    y = df['standard_value']       # Biến mục tiêu

    # Chia tỷ lệ Dữ liệu
    scaler = StandardScaler()
    X_scaled = scaler.fit_transform(X)

    # Chia dữ liệu thành các tập huấn luyện và kiểm tra
    X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2,
random_state=42)

    # Huấn luyện một mô hình hồi quy tuyến tính
    model = LinearRegression()
    model.fit(X_train, y_train)

    # Đưa ra dự đoán
    y_pred = model.predict(X_test)

    # Đánh giá mô hình
    mse = mean_squared_error(y_test, y_pred)
    r2 = r2_score(y_test, y_pred)

    print(f"Lỗi Bình phương Trung bình: {mse}")
    print(f"R-squared: {r2}")

    # Trực quan hóa (Biểu đồ phân tán của các giá trị thực tế so với giá trị dự đoán)
    plt.figure(figsize=(8, 6))
    plt.scatter(y_test, y_pred)
    plt.xlabel("Giá trị Thực tế")
    plt.ylabel("Giá trị Dự đoán")
    plt.title("Giá trị Thực tế so với Giá trị Dự đoán")
    plt.show()
```

Giải thích:

1. **Mô-đun hóa:** Các bước tải và tiền xử lý dữ liệu giờ đây được đóng gói trong một hàm `load_and_preprocess_data`. Điều này thúc đẩy khả năng tái sử dụng mã.
2. **Chọn Đặc trưng:** Mã chọn `molecular_weight` làm đặc trưng (X) và `standard_value` làm biến mục tiêu (y).
3. **Chia tỷ lệ Dữ liệu:** Sử dụng `StandardScaler` để chia tỷ lệ các đặc trưng. Điều này quan trọng đối với hồi quy tuyến tính và các mô hình khác nhạy cảm với việc chia tỷ lệ đặc trưng.
4. **Chia Tập Huấn luyện/Kiểm tra:** Chia dữ liệu thành các tập huấn luyện và kiểm tra bằng cách sử dụng `train_test_split`.
5. **Huấn luyện Mô hình:** Huấn luyện một mô hình hồi quy tuyến tính bằng cách sử dụng `LinearRegression`.
6. **Đánh giá Mô hình:**
    - Tính toán Lỗi Bình phương Trung bình (MSE) và R-squared (R2) để đánh giá hiệu suất của mô hình.
    - In các số liệu đánh giá.
7. **Trực quan hóa:** Tạo một biểu đồ phân tán của các giá trị thực tế so với các giá trị dự đoán để trực quan hóa hiệu suất của mô hình.

Giải quyết Lỗi `squared=False`

Lỗi "phiên bản scikit-learn cũ không hỗ trợ tham số squared=False trong hàm mean_squared_error" chỉ ra rằng bạn có thể đang sử dụng một phiên bản scikit-learn cũ hơn. Tham số `squared=False` đã

được giới thiệu trong một phiên bản sau này để cho phép bạn nhận trực tiếp Lỗi Bình phương Trung bình Gốc (RMSE).

Giải pháp:

Giải pháp đơn giản nhất là nâng cấp phiên bản scikit-learn của bạn:

```
pip install --upgrade scikit-learn
```

Nếu việc nâng cấp không phải là một tùy chọn (ví dụ: do các ràng buộc về môi trường), bạn có thể tính RMSE thủ công:

```
from math import sqrt

mse = mean_squared_error(y_test, y_pred)
rmse = sqrt(mse)  # Tính RMSE thủ công
print(f"Lỗi Bình phương Trung bình Gốc: {rmse}")
```

6. Ví dụ Sử dụng và Các Bước Tiếp theo

Bạn hiện có một khuôn khổ cơ bản để phân tích dữ liệu ChEMBL bằng RDKit và scikit-learn. Dưới đây là một số ý tưởng để mở rộng điều này:

Ví dụ 3: Thêm Nhiều Đặc trưng Hơn

Mở rộng kỹ thuật đặc trưng để bao gồm các bộ mô tả phân tử khác từ RDKit (ví dụ: LogP, số lượng người cho/nhận liên kết hydro).

```
def calculate_descriptors(smiles):
    mol = Chem.MolFromSmiles(smiles)
    if mol:
        mw = Descriptors.MolWt(mol)
        logp = Chem.Crippen.MolLogP(mol)
        hbd = Descriptors.NumHDonors(mol)
        hba = Descriptors.NumHAcceptors(mol)
        return mw, logp, hbd, hba
    else:
        return None, None, None, None

df[['molecular_weight', 'logp', 'hbd', 'hba']] = df['smiles'].apply(lambda x:
pd.Series(calculate_descriptors(x)))
df = df.dropna()  # Xóa các hàng có giá trị NaN sau khi tính toán bộ mô tả
```

Ví dụ 4: Thử Các Mô hình Khác Nhau

Thử nghiệm với các mô hình máy học khác nhau, chẳng hạn như Random Forest Regressor hoặc Support Vector Regressor.

```
from sklearn.ensemble import RandomForestRegressor

model = RandomForestRegressor(n_estimators=100, random_state=42)  # Ví dụ: Random
Forest
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
print(f"Random Forest - Lỗi Bình phương Trung bình: {mse}")
print(f"Random Forest - R-squared: {r2}")
```

Ví dụ 5: Điều chỉnh Siêu tham số

Sử dụng các kỹ thuật như GridSearchCV hoặc RandomizedSearchCV để tối ưu hóa các siêu tham số của mô hình bạn đã chọn.

```python from sklearn.model_selection import
```