PharmApp Suite

🧠 AI for Drug Discovery and Development 💉

🧩 Topic: Gene–compound mapping with bioactivity overlays

Okay, I understand. You're working on a Chembl 35 data analysis project, aiming to extract information relevant to drug research and development, and you're following an AIMLops template structure. You want to use RDKit within Jupyter notebooks to analyze the data extracted from a PostgreSQL database. You've encountered some errors and want help structuring your code, specifically around the topic "Topic_CheMBL_35_87". You need SQL and Python code, limited to the first 100 rows, along with error handling and solutions. You need the solutions in both English and Vietnamese. Here's a breakdown of the analysis, SQL code, Python code, and examples, along with explanations and error solutions, presented in both English and Vietnamese.

1. Analysis Model (Phân tích Mô hình)

The goal is to extract relevant data from the Chembl 35 database, perform data cleaning and transformation (e.g., converting SMILES strings to RDKit molecule objects, standardizing activity values), and then conduct exploratory data analysis (EDA) and potentially build predictive models based on chemical properties and activity data. The specific details of the analysis will depend on what "Topic_CheMBL_35_87" refers to, but a common workflow includes:

- **Data Extraction (Trích xuất dữ liệu):** Query the Chembl 35 database to retrieve relevant information based on a specific criteria. This often involves joining multiple tables to get compound structures (SMILES), target information, and activity data.
- **Data Cleaning and Preparation (Làm sạch và chuẩn bị dữ liệu):**
  - Handle missing values.
  - Standardize activity units (e.g., convert all IC50 values to nM).
  - Convert SMILES strings to RDKit `Mol` objects, handling invalid SMILES.
  - Calculate molecular descriptors using RDKit.
- **Exploratory Data Analysis (Phân tích dữ liệu thăm dò):**
  - Visualize the distribution of activity values.
  - Examine the correlation between molecular descriptors and activity.
  - Identify potential outliers.
- **Modeling (Xây dựng mô hình):**
  - Split the data into training and test sets.
  - Train a machine learning model (e.g., Random Forest, Support Vector Machine) to predict activity based on molecular descriptors.
  - Evaluate the model's performance on the test set.

2. SQL Code (Mã SQL)

Here's SQL code to extract data, addressing your error and limiting the output to 100 rows.

```
-- ../data/chembl_35_topic_87.csv

SELECT
    cmp.chembl_id,
    cs.canonical_smiles,
    act.standard_type,
    act.standard_value,
    act.standard_units,
```

```
    act.relation,
    t.target_type,
    t.pref_name AS target_name
FROM
    compound_structures cs
JOIN
    molecule_dictionary cmp ON cs.molregno = cmp.molregno
JOIN
    activities act ON cmp.molregno = act.molregno
JOIN
    target_dictionary t ON act.tid = t.tid
WHERE act.standard_type IN ('IC50', 'Ki', 'EC50') -- Filter for common activity types
  AND act.standard_units = 'nM' -- Focus on nM for consistency
  AND act.standard_value IS NOT NULL  --Avoid null values
  AND act.standard_value::text ~ '^[0-9\.]+$'  --Check values are numbers
LIMIT 100;
```

Explanation (Giải thích):

- **`SELECT ... FROM ... JOIN ...`**: This part of the query selects the desired columns and joins the necessary tables (compound_structures, molecule_dictionary, activities, and target_dictionary) based on their relationships (molregno, tid).
- **`WHERE` Clause**: This is crucial for filtering the data:
    - **`act.standard_type IN ('IC50', 'Ki', 'EC50')`**: Limits the results to common activity types.
    - **`act.standard_units = 'nM'`**: Ensures consistency by focusing on activities measured in nanomolars.
    - **`act.standard_value IS NOT NULL`**: Avoids potential errors by excluding rows with missing activity values.
    - **`act.standard_value::text ~ '^[0-9\.]+$'`**: **Error Solution (Giải pháp lỗi):** This line addresses the error you encountered. The `~` operator in PostgreSQL is for regular expression matching. The `^[0-9\.]+$` is a regular expression that checks if the `standard_value` consists only of digits and periods. The key is to cast `standard_value` to text before applying the regular expression, using `standard_value::text`.
- **`LIMIT 100`**: Limits the number of returned rows to 100.

Instructions (Hướng dẫn):

1. Open pgAdmin.
2. Connect to your PostgreSQL database (192.168.206.136, user: rd, pass: rd, database: chembl_35).
3. Open a query window.
4. Paste this SQL code into the query window.
5. Execute the query.
6. Save the results to a CSV file named `chembl_35_topic_87.csv` in your `../data/` directory.

3. Python Code (Mã Python)

```
import pandas as pd
import numpy as np
import os
from rdkit import Chem
from rdkit.Chem import Descriptors
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, r2_score

# Base path (assuming you are running the notebook from the 'notebooks' directory)
```

```
base_path = os.path.abspath(os.path.join(os.getcwd(), ".."))  #Go up one directory
from current dir(notebook)

# Construct the path to the CSV file
data_path = os.path.join(base_path, "data", "chembl_35_topic_87.csv")

# Load the data
try:
    df = pd.read_csv(data_path)
    print("Data loaded successfully.")
except FileNotFoundError:
    print(f"Error: File not found at {data_path}")
    exit()
except Exception as e:
    print(f"Error loading data: {e}")
    exit()

# Data Cleaning and Preparation
def process_data(df):
    # Drop rows with missing SMILES
    df = df.dropna(subset=['canonical_smiles'])

    # Convert standard_value to numeric, handling errors
    df['standard_value'] = pd.to_numeric(df['standard_value'], errors='coerce')
    df = df.dropna(subset=['standard_value'])

    # Function to convert SMILES to RDKit Mol object and handle errors
    def smiles_to_mol(smiles):
        try:
            mol = Chem.MolFromSmiles(smiles)
            if mol is None:
                return None
            return mol
        except:
            return None  # Handle any exceptions during SMILES parsing


    # Apply SMILES conversion
    df['mol'] = df['canonical_smiles'].apply(smiles_to_mol)
    df = df.dropna(subset=['mol']) # Remove rows where mol is None (invalid SMILES)

    return df

df = process_data(df.copy()) #Work on a copy to avoid modification in-place

# Feature Engineering (Molecular Descriptors)
def calculate_descriptors(mol):
    try:
        descriptors = {}
        descriptors['MW'] = Descriptors.MolWt(mol)
        descriptors['LogP'] = Descriptors.MolLogP(mol)
        descriptors['HBD'] = Descriptors.NumHDonors(mol)
        descriptors['HBA'] = Descriptors.NumHAcceptors(mol)
        return pd.Series(descriptors)
    except:
        return pd.Series([None] * 4, index=['MW', 'LogP', 'HBD', 'HBA'])

df[['MW', 'LogP', 'HBD', 'HBA']] = df['mol'].apply(calculate_descriptors)
df = df.dropna(subset=['MW', 'LogP', 'HBD', 'HBA']) #Drop NAs generated from
```

```
descriptor calculation

# Prepare data for modeling
X = df[['MW', 'LogP', 'HBD', 'HBA']]
y = df['standard_value'] #Activity values

# Split data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Model training (Random Forest)
model = RandomForestRegressor(n_estimators=100, random_state=42)  # You can tune
hyperparameters
model.fit(X_train, y_train)

# Model evaluation
y_pred = model.predict(X_test)

#Check if mean_squared_error has the squared parameter
import inspect
squared_param = 'squared' in inspect.getfullargspec(mean_squared_error).args

if squared_param:
    mse = mean_squared_error(y_test, y_pred, squared=False) #if available, use
squared=False
else:
    mse = mean_squared_error(y_test, y_pred)**0.5 #Calculate manually

r2 = r2_score(y_test, y_pred)

print(f"Root Mean Squared Error: {mse}")
print(f"R-squared: {r2}")
```

Explanation (Giải thích):

1. **Import Libraries**: Imports necessary libraries like pandas for data manipulation, RDKit for cheminformatics, and scikit-learn for machine learning.
2. **File Path Management**: Uses `os.path.join` to construct the correct file path based on your project structure. The `base_path` is calculated assuming your notebook is in the "notebooks" folder, and the data is in the "data" folder one level up.
3. **Data Loading**: Loads the CSV file into a pandas DataFrame, including error handling for file not found and other potential issues.
4. **Data Cleaning & Preparation**:
    - `process_data` function: Drops rows with missing SMILES strings and rows with invalid standard activity values. Converts activity values to numeric types. Converts SMILES to RDKit molecule objects, handling errors when the SMILES string is invalid using `Chem.MolFromSmiles`. Invalid SMILES will result in `None` which are then dropped.
5. **Feature Engineering**:
    - `calculate_descriptors` function: Calculates molecular descriptors using RDKit (Molecular Weight, LogP, H-bond donors, H-bond acceptors) which are then used as features for the model. The function also handles errors during descriptor calculation by returning `None` values if any error occurs during calculation which are later dropped.
6. **Data Splitting**: Splits the data into training and test sets using `train_test_split`.
7. **Model Training**: Trains a Random Forest Regressor model using the training data.
8. **Model Evaluation**: Predicts activity values for the test set and evaluates the model's performance using Root Mean Squared Error (RMSE) and R-squared.

9. **Error Handling for scikit-learn Version:** Checks if the installed scikit-learn version supports `squared=False` in `mean_squared_error` and calculates RMSE accordingly. If the `squared` parameter is not available, it calculates the square root of the MSE manually.

Instructions (Hướng dẫn):

1. Ensure you have RDKit and scikit-learn installed. If not, install them using `pip install rdkit scikit-learn`.
2. Create a Jupyter Notebook in your `notebooks` directory.
3. Copy and paste the Python code into the notebook.
4. Run the notebook.

4. Example Uses (Ví dụ sử dụng)

Here are five examples of how you might use this code to analyze the Chembl 35 data for drug research and development:

1. **Identifying Compounds with High Activity (Xác định các hợp chất có hoạt tính cao):** After loading and cleaning the data, you can filter the DataFrame to identify compounds with a `standard_value` below a certain threshold. This helps identify potential lead compounds.

```
active_compounds = df[df['standard_value'] < 100]  # IC50 < 100 nM
print(active_compounds[['chembl_id', 'standard_value']])
```

2. **Investigating Structure-Activity Relationships (Nghiên cứu mối quan hệ cấu trúc-hoạt tính):** You can explore the correlation between molecular descriptors (calculated using RDKit) and activity values. This helps understand which chemical properties are important for activity.

```
correlation = df[['standard_value', 'MW', 'LogP', 'HBD', 'HBA']].corr()
print(correlation)
import seaborn as sns
import matplotlib.pyplot as plt

sns.heatmap(correlation, annot=True, cmap='coolwarm')
plt.show()
```

3. **Predicting Activity for New Compounds (Dự đoán hoạt tính cho các hợp chất mới):** The trained machine learning model can be used to predict the activity of new compounds based on their molecular descriptors. This is useful for virtual screening and lead optimization.

```
# Example: predicting activity for a new compound
new_compound_smiles = 'Cc1ccccc1'  # Example SMILES string
new_mol = Chem.MolFromSmiles(new_compound_smiles)
if new_mol:
    new_descriptors = calculate_descriptors(new_mol)
    new_descriptors = new_descriptors.fillna(0) #Handle potential NAs
    new_descriptors_df = pd.DataFrame([new_descriptors])
    predicted_activity = model.predict(new_descriptors_df[['MW', 'LogP', 'HBD',
'HBA']])[0]
    print(f"Predicted activity for {new_compound_smiles}: {predicted_activity}")
else:
    print("Invalid SMILES string")
```

4. **Target Analysis (Phân tích mục tiêu):** You can group the data by target name (`target_name`) and analyze the activity distribution for each target. This helps understand which targets are most promising for drug development.

```
target_activity = df.groupby('target_name')['standard_value'].describe()
print(target_activity)
```

5. **Filtering by Target Type (Lọc theo loại mục tiêu):** You can filter the data based on target type (e.g., 'PROTEIN', 'ORGANISM') to focus on specific types of targets.

```
protein_targets = df[df['target_type'] == 'PROTEIN']
print(protein_targets.head())
```

Vietnamese Translation (Dịch tiếng Việt)

1. Phân tích Mô hình

Mục tiêu là trích xuất dữ liệu liên quan từ cơ sở dữ liệu Chembl 35, thực hiện làm sạch và chuyển đổi dữ liệu (ví dụ: chuyển đổi chuỗi SMILES thành đối tượng phân tử RDKit, chuẩn hóa các giá trị hoạt tính), sau đó tiến hành phân tích dữ liệu thăm dò (EDA) và có khả năng xây dựng các mô hình dự đoán dựa trên các thuộc tính hóa học và dữ liệu hoạt tính. Chi tiết cụ thể của phân tích sẽ phụ thuộc vào ý nghĩa của "Topic_CheMBL_35_87", nhưng quy trình làm việc phổ biến bao gồm:

- **Trích xuất dữ liệu:** Truy vấn cơ sở dữ liệu Chembl 35 để truy xuất thông tin liên quan dựa trên một tiêu chí cụ thể. Điều này thường liên quan đến việc kết hợp nhiều bảng để lấy cấu trúc hợp chất (SMILES), thông tin mục tiêu và dữ liệu hoạt tính.
- **Làm sạch và chuẩn bị dữ liệu:**
  - Xử lý các giá trị bị thiếu.
  - Chuẩn hóa các đơn vị hoạt tính (ví dụ: chuyển đổi tất cả các giá trị IC50 thành nM).
  - Chuyển đổi chuỗi SMILES thành đối tượng `Mol` của RDKit, xử lý các SMILES không hợp lệ.
  - Tính toán các mô tả phân tử bằng RDKit.
- **Phân tích dữ liệu thăm dò:**
  - Trực quan hóa sự phân bố của các giá trị hoạt tính.
  - Kiểm tra mối tương quan giữa các mô tả phân tử và hoạt tính.
  - Xác định các giá trị ngoại lệ tiềm năng.
- **Xây dựng mô hình:**
  - Chia dữ liệu thành bộ huấn luyện và bộ kiểm tra.
  - Huấn luyện một mô hình học máy (ví dụ: Random Forest, Support Vector Machine) để dự đoán hoạt tính dựa trên các mô tả phân tử.
  - Đánh giá hiệu suất của mô hình trên bộ kiểm tra.

2. Mã SQL

Đây là mã SQL để trích xuất dữ liệu, giải quyết lỗi của bạn và giới hạn đầu ra thành 100 hàng.

```
-- ../data/chembl_35_topic_87.csv

SELECT
    cmp.chembl_id,
    cs.canonical_smiles,
    act.standard_type,
    act.standard_value,
    act.standard_units,
    act.relation,
    t.target_type,
    t.pref_name AS target_name
FROM
    compound_structures cs
JOIN
    molecule_dictionary cmp ON cs.molregno = cmp.molregno
JOIN
    activities act ON cmp.molregno = act.molregno
JOIN
```

```
    target_dictionary t ON act.tid = t.tid
WHERE act.standard_type IN ('IC50', 'Ki', 'EC50') -- Lọc theo các loại hoạt tính phổ
biến
  AND act.standard_units = 'nM' -- Tập trung vào nM để nhất quán
  AND act.standard_value IS NOT NULL  -- Tránh các giá trị null
  AND act.standard_value::text ~ '^[0-9\.]+$'  --Kiểm tra giá trị là số
LIMIT 100;
```

Giải thích:

- **`SELECT ... FROM ... JOIN ...`**: Phần này của truy vấn chọn các cột mong muốn và kết hợp các bảng cần thiết (compound_structures, molecule_dictionary, activities và target_dictionary) dựa trên các mối quan hệ của chúng (molregno, tid).
- **Mệnh đề `WHERE`**: Điều này rất quan trọng để lọc dữ liệu:
  - **`act.standard_type IN ('IC50', 'Ki', 'EC50')`**: Giới hạn kết quả cho các loại hoạt tính phổ biến.
  - **`act.standard_units = 'nM'`**: Đảm bảo tính nhất quán bằng cách tập trung vào các hoạt tính được đo bằng nanomol.
  - **`act.standard_value IS NOT NULL`**: Tránh các lỗi tiềm ẩn bằng cách loại trừ các hàng có giá trị hoạt tính bị thiếu.
  - **`act.standard_value::text ~ '^[0-9\.]+$'`**: **Giải pháp lỗi:** Dòng này giải quyết lỗi bạn gặp phải. Toán tử `~` trong PostgreSQL dành cho so khớp biểu thức chính quy. `^[0-9\.]+$` là một biểu thức chính quy kiểm tra xem `standard_value` chỉ bao gồm các chữ số và dấu chấm hay không. Quan trọng là phải ép kiểu `standard_value` thành văn bản trước khi áp dụng biểu thức chính quy, bằng cách sử dụng `standard_value::text`.
- **`LIMIT 100`**: Giới hạn số lượng hàng trả về thành 100.

Hướng dẫn:

1. Mở pgAdmin.
2. Kết nối với cơ sở dữ liệu PostgreSQL của bạn (192.168.206.136, user: rd, pass: rd, database: chembl_35).
3. Mở một cửa sổ truy vấn.
4. Dán mã SQL này vào cửa sổ truy vấn.
5. Thực thi truy vấn.
6. Lưu kết quả vào một tệp CSV có tên `chembl_35_topic_87.csv` trong thư mục `../data/` của bạn.

3. Mã Python

```
import pandas as pd
import numpy as np
import os
from rdkit import Chem
from rdkit.Chem import Descriptors
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, r2_score

# Đường dẫn cơ sở (giả sử bạn đang chạy notebook từ thư mục 'notebooks')
base_path = os.path.abspath(os.path.join(os.getcwd(), ".."))  # Đi lên một thư mục từ
thư mục hiện tại (notebook)

# Xây dựng đường dẫn đến tệp CSV
data_path = os.path.join(base_path, "data", "chembl_35_topic_87.csv")

# Tải dữ liệu
```

```
try:
    df = pd.read_csv(data_path)
    print("Dữ liệu đã được tải thành công.")
except FileNotFoundError:
    print(f"Lỗi: Không tìm thấy tệp tại {data_path}")
    exit()
except Exception as e:
    print(f"Lỗi khi tải dữ liệu: {e}")
    exit()

# Làm sạch và chuẩn bị dữ liệu
def process_data(df):
    # Xóa các hàng có SMILES bị thiếu
    df = df.dropna(subset=['canonical_smiles'])

    # Chuyển đổi standard_value thành số, xử lý lỗi
    df['standard_value'] = pd.to_numeric(df['standard_value'], errors='coerce')
    df = df.dropna(subset=['standard_value'])

    # Hàm để chuyển đổi SMILES thành đối tượng RDKit Mol và xử lý lỗi
    def smiles_to_mol(smiles):
        try:
            mol = Chem.MolFromSmiles(smiles)
            if mol is None:
                return None
            return mol
        except:
            return None  # Xử lý mọi ngoại lệ trong quá trình phân tích cú pháp SMILES

    # Áp dụng chuyển đổi SMILES
    df['mol'] = df['canonical_smiles'].apply(smiles_to_mol)
    df = df.dropna(subset=['mol']) # Xóa các hàng mà mol là None (SMILES không hợp lệ)

    return df

df = process_data(df.copy()) # Làm việc trên một bản sao để tránh sửa đổi tại chỗ

# Kỹ thuật tính năng (Mô tả phân tử)
def calculate_descriptors(mol):
    try:
        descriptors = {}
        descriptors['MW'] = Descriptors.MolWt(mol)
        descriptors['LogP'] = Descriptors.MolLogP(mol)
        descriptors['HBD'] = Descriptors.NumHDonors(mol)
        descriptors['HBA'] = Descriptors.NumHAcceptors(mol)
        return pd.Series(descriptors)
    except:
        return pd.Series([None] * 4, index=['MW', 'LogP', 'HBD', 'HBA'])

df[['MW', 'LogP', 'HBD', 'HBA']] = df['mol'].apply(calculate_descriptors)
df = df.dropna(subset=['MW', 'LogP', 'HBD', 'HBA']) # Loại bỏ các giá trị NA được tạo
từ tính toán mô tả

# Chuẩn bị dữ liệu cho mô hình hóa
X = df[['MW', 'LogP', 'HBD', 'HBA']]
y = df['standard_value'] # Giá trị hoạt tính

# Chia dữ liệu
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
```

```
random_state=42)

# Huấn luyện mô hình (Random Forest)
model = RandomForestRegressor(n_estimators=100, random_state=42)  # Bạn có thể điều chỉnh các siêu tham số

model.fit(X_train, y_train)

# Đánh giá mô hình
y_pred = model.predict(X_test)

# Kiểm tra xem mean_squared_error có tham số squared hay không
import inspect
squared_param = 'squared' in inspect.getfullargspec(mean_squared_error).args

if squared_param:
    mse = mean_squared_error(y_test, y_pred, squared=False) # Nếu có, sử dụng squared=False
else:
    mse = mean_squared_error(y_test, y_pred)**0.5 # Tính toán thủ công

r2 = r2_score(y_test, y_pred)

print(f"Root Mean Squared Error: {mse}")
print(f"R-squared: {r2}")
```

Giải thích:

1. **Nhập thư viện**: Nhập các thư viện cần thiết như pandas để thao tác dữ liệu, RDKit cho tin học hóa học và scikit-learn cho học máy.
2. **Quản lý đường dẫn tệp**: Sử dụng `os.path.join` để xây dựng đường dẫn tệp chính xác dựa trên cấu trúc dự án của bạn. `base_path` được tính toán giả sử notebook của bạn nằm trong thư mục "notebooks" và dữ liệu nằm trong thư mục "data" ở một cấp cao hơn.
3. **Tải dữ liệu**: Tải tệp CSV vào DataFrame của pandas, bao gồm xử lý lỗi cho tệp không tìm thấy và các sự cố tiềm ẩn khác.
4. **Làm sạch & Chuẩn bị dữ liệu**:
    - Hàm `process_data`: Loại bỏ các hàng có chuỗi SMILES bị thiếu và các hàng có giá trị hoạt tính tiêu chuẩn không hợp lệ. Chuyển đổi giá trị hoạt tính sang kiểu số. Chuyển đổi SMILES thành đối tượng phân tử RDKit, xử lý lỗi khi chuỗi SMILES không hợp lệ bằng cách sử dụng `Chem.MolFromSmiles`. SMILES không hợp lệ sẽ trả về `None` và sau đó sẽ bị loại bỏ.
5. **Kỹ thuật tính năng**:
    - Hàm `calculate_descriptors`: Tính toán các mô tả phân tử bằng RDKit (Khối lượng phân tử, LogP, Số lượng liên kết hydro, Số lượng chất nhận liên kết hydro) sau đó được sử dụng làm tính năng cho mô hình. Hàm này cũng xử lý lỗi trong quá trình tính toán mô tả bằng cách trả về giá trị `None` nếu có lỗi xảy ra trong quá trình tính toán, sau đó sẽ bị loại bỏ.
6. **Chia dữ liệu**: Chia dữ liệu thành bộ huấn luyện và bộ kiểm tra bằng cách sử dụng `train_test_split`.
7. **Huấn luyện mô hình**: Huấn luyện mô hình Random Forest Regressor bằng cách sử dụng dữ liệu huấn luyện.
8. **Đánh giá mô hình**: Dự đoán các giá trị hoạt tính cho bộ kiểm tra và đánh giá hiệu suất của mô hình bằng cách sử dụng Root Mean Squared Error (RMSE) và R-squared.
9. **Xử lý lỗi cho phiên bản scikit-learn:** Kiểm tra xem phiên bản scikit-learn đã cài đặt có hỗ trợ `squared=False` trong `mean_squared_error` hay không và tính RMSE tương ứng. Nếu tham số `squared` không khả dụng, nó sẽ tính căn bậc hai của MSE theo cách thủ công.

Hướng dẫn:

1. Đảm bảo bạn đã cài đặt RDKit và scikit-learn. Nếu không, hãy cài đặt chúng bằng cách sử dụng `pip install rdkit scikit-learn`.
2. Tạo một Jupyter Notebook trong thư mục `notebooks` của bạn.
3. Sao chép và dán mã Python vào notebook.
4. Chạy notebook.

4. Ví dụ sử dụng

Đây là năm ví dụ về cách bạn có thể sử dụng mã này để phân tích dữ liệu Chembl 35 cho nghiên cứu và phát triển thuốc:

1. **Xác định các hợp chất có hoạt tính cao:** Sau khi tải và làm sạch dữ liệu, bạn có thể lọc DataFrame để xác định các hợp chất có `standard_value` dưới một ngưỡng nhất định. Điều này giúp xác định các hợp chất dẫn đầu tiềm năng.

```
active_compounds = df[df['standard_value'] < 100]  # IC50 < 100 nM
print(active_compounds[['chembl_id', 'standard_value']])
```

2. **Nghiên cứu mối quan hệ cấu trúc-hoạt tính:** Bạn có thể khám phá mối tương quan giữa các mô tả phân tử (được tính toán bằng RDKit) và các giá trị hoạt tính. Điều này giúp hiểu được các thuộc tính hóa học nào là quan trọng đối với hoạt tính.

```
correlation = df[['standard_value', 'MW', 'LogP', 'HBD', 'HBA']].corr()
print(correlation)
import seaborn as sns
import matplotlib.pyplot as plt

sns.heatmap(correlation, annot=True, cmap='coolwarm')
plt.show()
```

3. **Dự đoán hoạt tính cho các hợp chất mới:** Mô hình học máy đã được huấn luyện có thể được sử dụng để dự đoán hoạt tính của các hợp chất mới dựa trên các mô tả phân tử của chúng. Điều này hữu ích cho sàng lọc ảo và tối ưu hóa dẫn đầu.

```
# Ví dụ: dự đoán hoạt tính cho một hợp chất mới
new_compound_smiles = 'Cc1ccccc1'  # Ví dụ chuỗi SMILES
new_mol = Chem.MolFromSmiles(new_compound_smiles)
if new_mol:
    new_descriptors = calculate_descriptors(new_mol)
    new_descriptors = new_descriptors.fillna(0) # Xử lý các giá trị NA tiềm ẩn
    new_descriptors_df = pd.DataFrame([new_descriptors])
    predicted_activity = model.predict(new_descriptors_df[['MW', 'LogP', 'HBD',
'HBA']])[0]
    print(f"Hoạt tính dự đoán cho {new_compound_smiles}: {predicted_activity}")
else:
    print("Chuỗi SMILES không hợp lệ")
```

4. **Phân tích mục tiêu:** Bạn có thể nhóm dữ liệu theo tên mục tiêu (`target_name`) và phân tích sự phân bố hoạt tính cho từng mục tiêu. Điều này giúp hiểu được các mục tiêu nào là hứa hẹn nhất cho phát triển thuốc.

```
target_activity = df.groupby('target_name')['standard_value'].describe()
print(target_activity)
```

5. **Lọc theo loại mục tiêu:** Bạn có thể lọc dữ liệu dựa trên loại mục tiêu (ví dụ: 'PROTEIN', 'ORGANISM') để tập trung vào các loại mục tiêu cụ thể.

```
protein_targets = df[df['target_type'] == 'PROTEIN']
print(protein_targets.head())
```

This comprehensive breakdown, including the SQL and Python code, explanations, error handling, example use cases, and Vietnamese translations, should provide a solid foundation for your Chembl 35 data analysis project. Remember to adapt the code and examples to your specific needs and the precise goals of "Topic_CheMBL_35_87." Good luck!