PharmApp Suite

🧠 AI for Drug Discovery and Development 🧪

🧩 Topic: Target class distribution in ChEMBL

Okay, I understand. You're working with Chembl 35 data, using PostgreSQL with rdkit in a Jupyter Notebook environment following an AIMLops template. You've encountered SQL and Python errors and need guidance with corrected code, analysis, and examples.

Here's a breakdown of the analysis, SQL, Python code (including error fixes), and examples. I'll provide explanations in both English and Vietnamese.

1. Analysis Model (Phân tích Mô hình)

- **Objective (Mục tiêu):** To extract and analyze activity data from Chembl 35 for a specific target or compound class. We'll filter the data to ensure it contains relevant and valid activity measurements (IC50, Ki, etc.) and then perform some basic analysis. This will involve filtering for activity values that are numeric and then exploring the distribution of these values.
- **Data Source (Nguồn Dữ liệu):** Chembl 35 database, specifically activity data related to compounds.
- **Key Tables (Bảng chính):**
  - `activities`: Contains activity measurements (IC50, Ki, etc.).
  - `activity_properties`: Details about the activity, like the standard type (e.g., IC50).
  - `assays`: Information about the assay used to measure the activity.
  - `target_dictionary`: Details about the target the compound is acting on.
  - `molecule_dictionary`: Information about the molecule (SMILES, etc.).
- **Data Preprocessing (Xử lý dữ liệu):**
  - Filtering for specific activity types (e.g., IC50).
  - Filtering for numeric activity values (addresses SQL error).
  - Converting activity values to a logarithmic scale (optional, often useful for potency).
- **Analysis (Phân tích):**
  - Descriptive statistics (mean, median, standard deviation) of activity values.
  - Visualization of activity distributions (histograms, boxplots).
  - Potential correlation analysis with molecular properties (e.g., molecular weight, LogP). This would require RDKit to calculate the molecular properties.

2. SQL Code (Mã SQL)

Here's the SQL code, addressing the error you encountered and limiting the results to 100 rows.

```sql
-- Topic_CheMBL_35_24 - Extracting Activity Data

-- Ensure the rdkit extension is installed
CREATE EXTENSION IF NOT EXISTS rdkit;

-- Limit the results to 100 rows using LIMIT
SELECT
    md.chembl_id AS compound_chembl_id,
    ad.standard_type,
    ad.standard_value,
    ad.standard_units,
    td.chembl_id AS target_chembl_id,
```

```
    td.target_type,
    assays.description AS assay_description
FROM
    activities ad
JOIN
    activity_properties ap ON ad.activity_id = ap.activity_id
JOIN
    assays ON ad.assay_id = assays.assay_id
JOIN
    target_dictionary td ON assays.tid = td.tid
JOIN
    molecule_dictionary md ON ad.molregno = md.molregno
WHERE
    ad.standard_type IN ('IC50', 'Ki', 'EC50') -- Filter for common activity types
    AND ad.standard_relation = '=' -- Filter for direct relationships
    AND ad.standard_value IS NOT NULL -- Ensure value is not null
    AND ad.standard_value::text ~ '^[0-9\.]+$'  -- Corrected: Filter for numeric
values using explicit casting to text
    AND ad.standard_units = 'nM'  -- Filter for consistent units (nanomolar)
    AND td.target_type = 'SINGLE PROTEIN' -- Filter for single protein targets
LIMIT 100;


-- Save the results to a CSV file (using pgAdmin's export functionality or psql's
\copy command)
```

Explanation (Giải thích):

- `CREATE EXTENSION IF NOT EXISTS rdkit;`: Enables the RDKit extension. Important for cheminformatics functionality.
- `WHERE ad.standard_value::text ~ '^[0-9\.]+$'`: **Corrected the error.** The original error occurred because PostgreSQL couldn't directly compare a numeric type to a regular expression. The `::text` casts the `standard_value` to text, allowing the regular expression `'^[0-9\.]+$'` to check if it contains only numbers and dots (allowing for decimal values).
- `LIMIT 100`: Restricts the output to the first 100 rows.
- The query joins multiple tables to retrieve compound IDs, activity data, target information, and assay details.
- Filters are applied to select specific activity types (IC50, Ki, EC50), a direct relationship (=), non-null values, numeric values, nM units, and single protein targets.

Vietnamese Explanation (Giải thích tiếng Việt):

- `CREATE EXTENSION IF NOT EXISTS rdkit;`: Kích hoạt tiện ích RDKit. Quan trọng cho các chức năng tin sinh học hóa học.
- `WHERE ad.standard_value::text ~ '^[0-9\.]+$'`: **Đã sửa lỗi.** Lỗi ban đầu xảy ra vì PostgreSQL không thể so sánh trực tiếp kiểu số với biểu thức chính quy. `::text` chuyển `standard_value` thành văn bản, cho phép biểu thức chính quy `'^[0-9\.]+$'` kiểm tra xem nó chỉ chứa số và dấu chấm (cho phép giá trị thập phân).
- `LIMIT 100`: Giới hạn đầu ra thành 100 hàng đầu tiên.
- Truy vấn này kết hợp nhiều bảng để lấy ID hợp chất, dữ liệu hoạt động, thông tin mục tiêu và chi tiết thử nghiệm.
- Bộ lọc được áp dụng để chọn các loại hoạt động cụ thể (IC50, Ki, EC50), mối quan hệ trực tiếp (=), giá trị không rỗng, giá trị số, đơn vị nM và mục tiêu protein đơn lẻ.

To Save to CSV (Để Lưu vào CSV):

1. **pgAdmin:** After running the query in pgAdmin, use the "Export" functionality (usually found in the query result window) to save the results as a CSV file to your `../data/` directory.

2. **psql:** Alternatively, from the command line using `psql`:

   ```
   psql -h 192.168.206.136 -U rd -d chembl_35 -c "COPY (SELECT ... your SQL query here ...) TO '/path/to/your/data/chembl_data.csv' WITH (FORMAT CSV, HEADER);"
   ```

   Replace `/path/to/your/data/` with the actual path. You might need to adjust permissions to allow the `postgres` user to write to that directory.

3. Python Code (Mã Python)

```python
# -*- coding: utf-8 -*-
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.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
import matplotlib.pyplot as plt

# Define the base path according to AIMLops template
base_path = "../data"  # Adjust if your base path is different
csv_file_path = os.path.join(base_path, "chembl_data.csv")  # Assuming you saved the SQL results as chembl_data.csv

# Load the data
try:
    df = pd.read_csv(csv_file_path)
except FileNotFoundError:
    print(f"Error: File not found at {csv_file_path}. Make sure you've saved the CSV file from your SQL query.")
    exit()

# Data Cleaning and Preprocessing
df = df.dropna(subset=['standard_value'])  # Remove rows with missing standard_value
df = df[pd.to_numeric(df['standard_value'], errors='coerce').notna()] # Ensure standard_value is numeric after reading from csv

# Convert standard_value to numeric
df['standard_value'] = pd.to_numeric(df['standard_value'])

# Filter for IC50 values only, you can adjust this based on your needs
df = df[df['standard_type'] == 'IC50']

# Convert IC50 to pIC50 (optional but common)
df['pIC50'] = -np.log10(df['standard_value'] * 1e-9) # Convert nM to M and then to pIC50

# RDKit - Calculate Molecular Descriptors (example)
def calculate_descriptors(smiles):
    try:
        mol = Chem.MolFromSmiles(smiles)
        if mol is None:
            return None
        descriptors = {desc[0]: desc[1](mol) for desc in Descriptors.descList}
        return descriptors
    except:
        return None
```

```
# Fetch canonical smiles from molecule_dictionary.
# You will need another query to fetch smiles and merge it into the dataframe
# For the example, let's assume you have a 'smiles' column in your df

# Example only - Generating random SMILES (replace with your actual smiles data)
import random
df['smiles'] = [Chem.MolToSmiles(Chem.MolFromSmiles('C' * random.randint(1, 10))) for
_ in range(len(df))]

df['descriptors'] = df['smiles'].apply(calculate_descriptors)

# Handle cases where descriptor calculation failed
df = df.dropna(subset=['descriptors'])

# Convert the 'descriptors' column to a DataFrame
descriptors_df = pd.DataFrame(df['descriptors'].tolist())

# Remove columns with NaN values (if any)
descriptors_df = descriptors_df.dropna(axis=1, how='any')

# Merge descriptors with main dataframe
df = pd.concat([df, descriptors_df], axis=1)

# Drop the original 'descriptors' and 'smiles' column
df = df.drop(columns=['descriptors', 'smiles'])

# Prepare data for modeling
X = df.select_dtypes(include=np.number).drop(columns=['standard_value', 'pIC50'])  #
Select numerical features, remove target
y = df['pIC50'] # Target variable

# Handle missing values by imputation
from sklearn.impute import SimpleImputer

imputer = SimpleImputer(strategy='mean')  # Replace missing values with the 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 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 (example)
plt.scatter(y_test, y_pred)
```

```
plt.xlabel("Actual pIC50")
plt.ylabel("Predicted pIC50")
plt.title("Actual vs. Predicted pIC50 Values")
plt.show()
```

Explanation (Giải thích):

- **Imports:** Imports necessary libraries (pandas, numpy, rdkit, sklearn).
- **File Loading:** Loads the CSV file you created from the SQL query. Handles potential `FileNotFoundError`.
- **Data Cleaning:** Removes rows with missing or non-numeric `standard_value`. Specifically handles cases where `standard_value` is read as object after loading csv and needs to be converted to numeric and removes rows where this conversion fails.
- **Unit Conversion (Optional):** Converts `IC50` values to `pIC50` (a common transformation).
- **RDKit Integration:**
  - The `calculate_descriptors` function calculates molecular descriptors using RDKit, converting SMILES strings into molecular objects and extracting relevant properties.
  - It handles potential errors during SMILES parsing.
- **Feature Engineering:** Converts calculated descriptors to columns and concatenates to the main dataframe
- **Data Preparation:**
  - Selects numerical features from the DataFrame to use as input for the model (X).
  - Defines the target variable (y) as the 'pIC50' column.
  - Handles missing values using `SimpleImputer`.
- **Model Training:** Trains a Linear Regression model using the training data.
- **Model Evaluation:** Evaluates the model using Mean Squared Error (MSE) and R-squared (R2).
- **Visualization:** Creates a scatter plot of actual vs. predicted pIC50 values.

Vietnamese Explanation (Giải thích tiếng Việt):

- **Nhập thư viện:** Nhập các thư viện cần thiết (pandas, numpy, rdkit, sklearn).
- **Tải tệp:** Tải tệp CSV bạn đã tạo từ truy vấn SQL. Xử lý `FileNotFoundError` tiềm năng.
- **Làm sạch dữ liệu:** Loại bỏ các hàng có `standard_value` bị thiếu hoặc không phải là số.
- **Chuyển đổi đơn vị (Tùy chọn):** Chuyển đổi các giá trị `IC50` thành `pIC50` (một biến đổi phổ biến).
- **Tích hợp RDKit:**
  - Hàm `calculate_descriptors` tính toán các mô tả phân tử bằng RDKit, chuyển đổi chuỗi SMILES thành các đối tượng phân tử và trích xuất các thuộc tính có liên quan.
  - Nó xử lý các lỗi tiềm ẩn trong quá trình phân tích cú pháp SMILES.
- **Chuẩn bị dữ liệu:**
  - Chọn các tính năng số từ DataFrame để sử dụng làm đầu vào cho mô hình (X).
  - Xác định biến mục tiêu (y) là cột 'pIC50'.
- **Huấn luyện mô hình:** Huấn luyện một mô hình Hồi quy tuyến tính bằng dữ liệu huấn luyện.
- **Đánh giá mô hình:** Đánh giá mô hình bằng Lỗi bình phương trung bình (MSE) và R-squared (R2).
- **Trực quan hóa:** Tạo một biểu đồ phân tán của các giá trị pIC50 thực tế so với giá trị dự đoán.

4. Error Correction (Sửa Lỗi)

- **SQL Error:** The SQL error `ERROR: operator does not exist: numeric ~ unknown` was fixed by explicitly casting the `standard_value` to text: `ad.standard_value::text ~ '^[0-9\.]+$'`.
- **Old scikit-learn version:** You mentioned an issue with `squared=False` in `mean_squared_error`. This parameter was added in a later version of scikit-learn. The

standard MSE calculation *is* squared, so if you're using an older version, just omit the `squared=False` argument.

- **Ensure standard_value is numeric after reading from csv.** After loading the csv, `standard_value` may be of `object` type, so the conversion and filtering are performed to avoid any type errors.
- **Handle missing values**: Missing values from the imported data will cause errors in the prediction process, so they should be handled with imputation.

5. Examples (Ví dụ)

Here are five examples of how you can use this code, focusing on different aspects of the analysis. Each example builds on the previous code blocks. These assume you've run the SQL query, saved the data to `chembl_data.csv`, and have the necessary libraries installed.

Example 1: Basic Data Loading and Inspection (Ví dụ 1: Tải và Kiểm tra Dữ liệu Cơ bản)

```
import pandas as pd
import os

base_path = "../data"
csv_file_path = os.path.join(base_path, "chembl_data.csv")

try:
    df = pd.read_csv(csv_file_path)
except FileNotFoundError:
    print(f"Error: File not found at {csv_file_path}.")
    exit()

print(df.head())  # Display the first few rows
print(df.info())  # Get information about the DataFrame (data types, missing values)
```

Explanation: This example simply loads the data and prints the first few rows and information about the DataFrame. This is useful for quickly verifying that the data has been loaded correctly.

Vietnamese: Ví dụ này chỉ đơn giản là tải dữ liệu và in ra một vài hàng đầu tiên và thông tin về DataFrame. Điều này hữu ích để nhanh chóng xác minh rằng dữ liệu đã được tải chính xác.

Example 2: Filtering and Descriptive Statistics (Ví dụ 2: Lọc và Thống kê Mô tả)

```
import pandas as pd
import os

base_path = "../data"
csv_file_path = os.path.join(base_path, "chembl_data.csv")

try:
    df = pd.read_csv(csv_file_path)
except FileNotFoundError:
    print(f"Error: File not found at {csv_file_path}.")
    exit()

df = df.dropna(subset=['standard_value']) # Drop rows with missing standard values
df = df[pd.to_numeric(df['standard_value'], errors='coerce').notna()] # Ensure
standard_value is numeric after reading from csv
df['standard_value'] = pd.to_numeric(df['standard_value'])

df = df[df['standard_type'] == 'IC50'] # Filter for IC50 values
print(df['standard_value'].describe())  # Calculate and print descriptive statistics
for IC50 values
```

Explanation: This example filters the data to include only IC50 values and then calculates descriptive statistics (mean, standard deviation, min, max, etc.) for the `standard_value` column.

Vietnamese: Ví dụ này lọc dữ liệu để chỉ bao gồm các giá trị IC50, sau đó tính toán thống kê mô tả (giá trị trung bình, độ lệch chuẩn, min, max, v.v.) cho cột `standard_value`.

Example 3: pIC50 Conversion and Visualization (Ví dụ 3: Chuyển đổi pIC50 và Trực quan hóa)

```
import pandas as pd
import numpy as np
import os
import matplotlib.pyplot as plt

base_path = "../data"
csv_file_path = os.path.join(base_path, "chembl_data.csv")

try:
    df = pd.read_csv(csv_file_path)
except FileNotFoundError:
    print(f"Error: File not found at {csv_file_path}.")
    exit()

df = df.dropna(subset=['standard_value']) # Drop rows with missing standard values
df = df[pd.to_numeric(df['standard_value'], errors='coerce').notna()] # Ensure
standard_value is numeric after reading from csv
df['standard_value'] = pd.to_numeric(df['standard_value'])
df = df[df['standard_type'] == 'IC50']
df['pIC50'] = -np.log10(df['standard_value'] * 1e-9)  # Convert IC50 to pIC50

plt.hist(df['pIC50'], bins=20) # Create a histogram of pIC50 values
plt.xlabel("pIC50")
plt.ylabel("Frequency")
plt.title("Distribution of pIC50 Values")
plt.show()
```

Explanation: This example converts IC50 values to pIC50 and then creates a histogram to visualize the distribution of pIC50 values.

Vietnamese: Ví dụ này chuyển đổi các giá trị IC50 thành pIC50 và sau đó tạo một biểu đồ để trực quan hóa sự phân phối của các giá trị pIC50.

Example 4: Calculating a Single Molecular Descriptor (Ví dụ 4: Tính toán Mô tả Phân tử Đơn)

```
import pandas as pd
import numpy as np
import os
from rdkit import Chem
from rdkit.Chem import Descriptors

base_path = "../data"
csv_file_path = os.path.join(base_path, "chembl_data.csv")

try:
    df = pd.read_csv(csv_file_path)
except FileNotFoundError:
    print(f"Error: File not found at {csv_file_path}.")
    exit()

df = df.dropna(subset=['standard_value']) # Drop rows with missing standard values
df = df[pd.to_numeric(df['standard_value'], errors='coerce').notna()] # Ensure
standard_value is numeric after reading from csv
```

```
df['standard_value'] = pd.to_numeric(df['standard_value'])
df = df[df['standard_type'] == 'IC50']
df['pIC50'] = -np.log10(df['standard_value'] * 1e-9)

# Fetch canonical smiles from molecule_dictionary.
# You will need another query to fetch smiles and merge it into the dataframe
# For the example, let's assume you have a 'smiles' column in your df

# Example only - Generating random SMILES (replace with your actual smiles data)
import random
df['smiles'] = [Chem.MolToSmiles(Chem.MolFromSmiles('C' * random.randint(1, 10))) for
_ in range(len(df))]

def calculate_molecular_weight(smiles):
    try:
        mol = Chem.MolFromSmiles(smiles)
        if mol is None:
            return np.nan
        return Descriptors.MolWt(mol)
    except:
        return np.nan

df['molecular_weight'] = df['smiles'].apply(calculate_molecular_weight)
print(df[['smiles', 'molecular_weight']].head())
```

Explanation: This example demonstrates how to use RDKit to calculate a single molecular descriptor (molecular weight) from SMILES strings. It includes error handling for invalid SMILES. Remember to replace the example SMILES generation with *your actual SMILES data* from the database.

Vietnamese: Ví dụ này minh họa cách sử dụng RDKit để tính toán một mô tả phân tử duy nhất (trọng lượng phân tử) từ chuỗi SMILES. Nó bao gồm xử lý lỗi cho SMILES không hợp lệ. Hãy nhớ thay thế việc tạo SMILES mẫu bằng *dữ liệu SMILES thực tế của bạn* từ cơ sở dữ liệu.

Example 5: Correlation Analysis (Ví dụ 5: Phân tích Tương quan)

```
import pandas as pd
import numpy as np
import os
from rdkit import Chem
from rdkit.Chem import Descriptors

base_path = "../data"
csv_file_path = os.path.join(base_path, "chembl_data.csv")

try:
    df = pd.read_csv(csv_file_path)
except FileNotFoundError:
    print(f"Error: File not found at {csv_file_path}.")
    exit()

df = df.dropna(subset=['standard_value']) # Drop rows with missing standard values
df = df[pd.to_numeric(df['standard_value'], errors='coerce').notna()] # Ensure
standard_value is numeric after reading from csv
df['standard_value'] = pd.to_numeric(df['standard_value'])
df = df[df['standard_type'] == 'IC50']
df['pIC50'] = -np.log10(df['standard_value'] * 1e-9)

# Fetch canonical smiles from molecule_dictionary.
# You will need another query to fetch smiles and merge it into the dataframe
# For the example, let's assume you have a 'smiles' column in your df
```

```
# Example only - Generating random SMILES (replace with your actual smiles data)
import random
df['smiles'] = [Chem.MolToSmiles(Chem.MolFromSmiles('C' * random.randint(1, 10))) for
_ in range(len(df))]


def calculate_molecular_weight(smiles):
    try:
        mol = Chem.MolFromSmiles(smiles)
        if mol is None:
            return np.nan
        return Descriptors.MolWt(mol)
    except:
        return np.nan

df['molecular_weight'] = df['smiles'].apply(calculate_molecular_weight)
df = df.dropna(subset=['molecular_weight'])  # Drop rows where molecular weight
calculation failed

correlation = df['pIC50'].corr(df['molecular_weight']) # Calculate the correlation
between pIC50 and molecular weight

print(f"Correlation between pIC50 and Molecular Weight: {correlation}")
```

Explanation: This example calculates the Pearson correlation coefficient between pIC50 and molecular weight. It first calculates the molecular weight using RDKit and then uses the `.corr()` method to find the correlation.

Vietnamese: Ví dụ này tính toán hệ số tương quan Pearson giữa pIC50 và trọng lượng phân tử. Đầu tiên, nó tính toán trọng lượng phân tử bằng RDKit, sau đó sử dụng phương thức `.corr()` để tìm tương quan.

Important Considerations:

- **SMILES Data:** The most crucial part is getting the SMILES strings for your compounds. You'll need a separate SQL query to retrieve the SMILES from the `molecule_dictionary` table (using `md.molregno` to join) and then merge that data into your main DataFrame. The example uses random SMILES *only* to make the code runnable without the full dataset.
- **Data Validation:** Thoroughly validate your data, especially the SMILES strings. Invalid SMILES will cause errors in RDKit.
- **Feature Selection:** When building models, carefully select the most relevant molecular descriptors. Not all descriptors are useful or informative.
- **Model Selection:** Linear Regression is a simple starting point, but more complex models (e.g., Random Forest, Support Vector Machines) might provide better performance.
- **Units:** Always pay close attention to units (e.g., nM for IC50). Ensure consistency before performing calculations.

This comprehensive guide should help you analyze your Chembl 35 data effectively. Remember to adapt the code to your specific needs and data. Good luck with your project!