PharmApp Suite

🧠 AI for Drug Discovery and Development 💉

🧩 Topic: Integrating adverse event data (SIDER/FAERS)

Okay, I understand. You're working with ChEMBL 35 data in a structured project environment using the AIMLops template. You're using PostgreSQL for data storage, RDKit for cheminformatics, and Jupyter notebooks for code execution. You want guidance on running your `Topic_CheMBL_35_86` analysis, dealing with specific errors, and limiting data to 100 rows. Let's break this down.

Analysis and Guidance (English & Vietnamese)

1. Analysis Model (Phân tích Mô hình)

Your project likely involves analyzing bioactivity data from ChEMBL 35. A typical workflow might include:

- **Data Extraction and Cleaning:** Using SQL to extract relevant information from the ChEMBL database, focusing on specific target proteins, assays, or activity types. This often involves filtering data based on quality criteria (e.g., activity type, standard units, confidence scores). Cleaning involves handling missing values and converting data into a consistent format.
- **Feature Engineering:** Using RDKit to generate molecular descriptors (e.g., fingerprints, physicochemical properties) from the SMILES strings of the compounds. These descriptors become the features used in your models.
- **Data Analysis/Modeling:** Using machine learning to build predictive models. This could involve regression (e.g., predicting pIC50 values) or classification (e.g., predicting active vs. inactive compounds). Common algorithms include linear regression, random forests, support vector machines, and neural networks.
- **Model Evaluation:** Evaluating the performance of your models using appropriate metrics (e.g., R-squared, RMSE for regression; accuracy, precision, recall, F1-score for classification).
- **Visualization and Interpretation:** Visualizing data and model results to gain insights into the relationships between molecular features and bioactivity.

Vietnamese Translation:

- **Trích xuất và làm sạch dữ liệu:** Sử dụng SQL để trích xuất thông tin liên quan từ cơ sở dữ liệu ChEMBL, tập trung vào các protein mục tiêu, xét nghiệm hoặc loại hoạt động cụ thể. Điều này thường bao gồm lọc dữ liệu dựa trên các tiêu chí chất lượng (ví dụ: loại hoạt động, đơn vị tiêu chuẩn, điểm tin cậy). Làm sạch bao gồm xử lý các giá trị bị thiếu và chuyển đổi dữ liệu thành một định dạng nhất quán.
- **Thiết kế đặc trưng:** Sử dụng RDKit để tạo các mô tả phân tử (ví dụ: dấu vân tay, thuộc tính lý hóa) từ chuỗi SMILES của các hợp chất. Các mô tả này trở thành các đặc trưng được sử dụng trong mô hình của bạn.
- **Phân tích/Mô hình hóa dữ liệu:** Sử dụng máy học để xây dựng các mô hình dự đoán. Điều này có thể bao gồm hồi quy (ví dụ: dự đoán giá trị pIC50) hoặc phân loại (ví dụ: dự đoán các hợp chất hoạt động so với không hoạt động). Các thuật toán phổ biến bao gồm hồi quy tuyến tính, rừng ngẫu nhiên, máy vectơ hỗ trợ và mạng nơ-ron.
- **Đánh giá mô hình:** Đánh giá hiệu suất của mô hình của bạn bằng các số liệu phù hợp (ví dụ: R-squared, RMSE cho hồi quy; độ chính xác, độ thu hồi, điểm F1 cho phân loại).
- **Trực quan hóa và diễn giải:** Trực quan hóa dữ liệu và kết quả mô hình để hiểu rõ hơn về mối quan hệ giữa các đặc điểm phân tử và hoạt tính sinh học.

2. SQL and Python Code (Mã SQL và Python)

Here's an example of SQL code to extract data, addressing the error you mentioned, and then Python code to process it.

SQL Code (extract_chembl_data.sql):

```sql
-- SQL Code to extract ChEMBL data (limited to 100 rows)

SELECT
    md.chembl_id,
    cs.canonical_smiles,
    act.standard_type,
    act.standard_value,
    act.standard_units,
    act.pchembl_value
FROM
    compound_structures cs
JOIN
    molecule_dictionary md ON cs.molregno = md.molregno
JOIN
    activities act ON md.molregno = act.molregno
JOIN
    assays ass ON act.assay_id = ass.assay_id
JOIN
    target_dictionary td ON ass.tid = td.tid
WHERE
    td.target_name = 'Your Target of Interest'  -- Replace with your target
    AND act.standard_type = 'IC50'            -- Example bioactivity type
    AND act.standard_units = 'nM'
    AND act.standard_value IS NOT NULL
    AND act.standard_value::text ~ '^[0-9.]+$'  -- Corrected regex for numeric check
    AND act.pchembl_value IS NOT NULL
LIMIT 100;

-- Save the result to a CSV file
\copy (SELECT * FROM (
SELECT
    md.chembl_id,
    cs.canonical_smiles,
    act.standard_type,
    act.standard_value,
    act.standard_units,
    act.pchembl_value
FROM
    compound_structures cs
JOIN
    molecule_dictionary md ON cs.molregno = md.molregno
JOIN
    activities act ON md.molregno = act.molregno
JOIN
    assays ass ON act.assay_id = ass.assay_id
JOIN
    target_dictionary td ON ass.tid = td.tid
WHERE
    td.target_name = 'Your Target of Interest'  -- Replace with your target
    AND act.standard_type = 'IC50'            -- Example bioactivity type
    AND act.standard_units = 'nM'
    AND act.standard_value IS NOT NULL
    AND act.standard_value::text ~ '^[0-9.]+$'  -- Corrected regex for numeric check
    AND act.pchembl_value IS NOT NULL
LIMIT 100
) as subquery) TO '../data/chembl_data.csv' DELIMITER ',' CSV HEADER;
```

Explanation of the SQL:

- **JOINs:** This query joins several ChEMBL tables to retrieve compound structures, activities, assay information, and target information.
- **`WHERE` clause:** This clause filters the data:
  - `td.target_name = 'Your Target of Interest'` : Replace this with the name of your target protein.
  - `act.standard_type = 'IC50'`: Specifies the type of activity you're interested in. Change this if you're looking for Ki, EC50, etc.
  - `act.standard_units = 'nM'`: Only include activities measured in nanomolars.
  - `act.standard_value IS NOT NULL`: Excludes rows with missing standard values.
  - `act.standard_value::text ~ '^[0-9.]+$'`: **Important Correction:** This line addresses the error you encountered. It first casts the `standard_value` to text, then uses a regular expression (~) to check if the value contains only numbers and periods. This ensures that you're only including numeric values in your data.
  - `act.pchembl_value IS NOT NULL`: Excludes rows with missing pChEMBL values.
- **`LIMIT 100`:** This limits the results to the first 100 rows.
- **`\copy`**: Save the result to a CSV file

Important:

1. **Replace `'Your Target of Interest'` with the actual target name you're interested in.**
2. Run this SQL in pgAdmin. It will save a file named `chembl_data.csv` in your `../data/` directory.
3. Make sure your user has the right permissions to write to this directory.

Python Code (Topic_CheMBL_35_86_1_data_processing.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.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score

# Define the base path
base_path = ".."  # Assuming your notebook is in the 'notebooks' directory

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

# Load the data using pandas
try:
    df = pd.read_csv(data_file)
    print("Data loaded successfully.")
except FileNotFoundError:
    print(f"Error: File not found at {data_file}")
    exit()

# Display the first few rows of the DataFrame
print(df.head())

# Basic Data Cleaning
df = df.dropna(subset=['canonical_smiles', 'pchembl_value']) # Drop rows with missing
```

```
SMILES or pChEMBL values

# RDKit Feature Generation (Example: Calculate Molecular Weight and LogP)
def calculate_descriptors(smiles):
    mol = Chem.MolFromSmiles(smiles)
    if mol is None:
        return None, None  # Handle invalid SMILES
    mw = Descriptors.MolWt(mol)
    logp = Descriptors.MolLogP(mol)
    return mw, logp

df[['molecular_weight', 'logp']] = df['canonical_smiles'].apply(lambda x:
pd.Series(calculate_descriptors(x)))
df = df.dropna() #drop rows with invalid smiles

# Display the DataFrame with new features
print(df.head())

# Prepare data for machine Learning

# Define features and target
X = df[['molecular_weight', 'logp']]  # Features: Molecular Weight and LogP
y = df['pchembl_value'] # Target: pChEMBL value

# Split the 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)

# Create a Linear Regression model
model = LinearRegression()

# Train the model
model.fit(X_train, y_train)

# Make predictions on the test set
y_pred = model.predict(X_test)

# Evaluate the model

# Old scikit-learn version does not support parameters squared=False in the
mean_squared_error function
# Calculate Mean Squared Error (MSE)
mse = mean_squared_error(y_test, y_pred)

# Calculate R-squared
r2 = r2_score(y_test, y_pred)

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

Explanation of the Python Code:

- **Import Libraries:** Imports necessary libraries (pandas, RDKit, scikit-learn).
- **Define Paths:** Defines the path to the CSV file using `os.path.join`. This ensures that the code works regardless of your current working directory.
- **Load Data:** Loads the CSV data into a pandas DataFrame.
- **Data Cleaning:** Removes rows with missing SMILES strings or pChEMBL values.

- **Feature Generation (RDKit):** Defines a function `calculate_descriptors` to calculate molecular weight (MW) and LogP using RDKit. These are simple example features. You can add more complex descriptors as needed. The code handles `None` returns from RDKit which indicate invalid SMILES strings.
- **Feature Selection:** Selects molecular weight and LogP as features (`X`) and pChEMBL value as the target variable (`y`).
- **Data Splitting:** Splits the data into training and testing sets.
- **Model Training:** Creates a Linear Regression model and trains it using the training data.
- **Model Evaluation:** Predicts pChEMBL values for the test set and evaluates the model using Mean Squared Error (MSE) and R-squared.
- **Error Handling:** Include `try...except` for file handling and `if mol is None` in the `calculate_descriptors` function to handle invalid SMILES strings, this prevents your program from crashing.

Vietnamese Translation (Giải thích Mã Python):

- **Nhập thư viện:** Nhập các thư viện cần thiết (pandas, RDKit, scikit-learn).
- **Xác định đường dẫn:** Xác định đường dẫn đến tệp CSV bằng `os.path.join`. Điều này đảm bảo rằng mã hoạt động bất kể thư mục làm việc hiện tại của bạn.
- **Tải dữ liệu:** Tải dữ liệu CSV vào một DataFrame pandas.
- **Làm sạch dữ liệu:** Loại bỏ các hàng có chuỗi SMILES hoặc giá trị pChEMBL bị thiếu.
- **Tạo đặc trưng (RDKit):** Xác định một hàm `calculate_descriptors` để tính toán trọng lượng phân tử (MW) và LogP bằng RDKit. Đây là những đặc trưng ví dụ đơn giản. Bạn có thể thêm các mô tả phức tạp hơn nếu cần. Mã xử lý các giá trị `None` trả về từ RDKit, cho biết các chuỗi SMILES không hợp lệ.
- **Chọn đặc trưng:** Chọn trọng lượng phân tử và LogP làm đặc trưng (`X`) và giá trị pChEMBL làm biến mục tiêu (`y`).
- **Chia dữ liệu:** Chia dữ liệu thành các tập huấn luyện và kiểm tra.
- **Huấn luyện mô hình:** Tạo mô hình Hồi quy tuyến tính và huấn luyện nó bằng dữ liệu huấn luyện.
- **Đánh giá mô hình:** Dự đoán các giá trị pChEMBL cho tập kiểm tra và đánh giá mô hình bằng Lỗi bình phương trung bình (MSE) và R-squared.
- **Xử lý lỗi:** Bao gồm `try...except` để xử lý tệp và `if mol is None` trong hàm `calculate_descriptors` để xử lý các chuỗi SMILES không hợp lệ, điều này ngăn chương trình của bạn bị sập.

3. Addressing the `mean_squared_error` Error (Giải quyết lỗi `mean_squared_error`)

The error `old scikit-learn version does not support parameters squared=False in the mean_squared_error function` indicates that you're using an older version of scikit-learn. The `squared=False` parameter was introduced in a later version of the library, which is used to return the Root Mean Squared Error (RMSE) instead of the MSE.

Solution: Since you have an older version, remove the `squared=False` parameter from the `mean_squared_error` function call.

```
#Calculate Mean Squared Error (MSE)
mse = mean_squared_error(y_test, y_pred)
```

If you need RMSE you would need to calculate it manually:

```
# Calculate Mean Squared Error (MSE)
mse = mean_squared_error(y_test, y_pred)

# Calculate Root Mean Squared Error (RMSE)
rmse = np.sqrt(mse)
```

```
print(f"Mean Squared Error: {mse}")
print(f"Root Mean Squared Error: {rmse}")
```

Vietnamese Translation (Giải pháp cho lỗi):

Lỗi `old scikit-learn version does not support parameters squared=False in the mean_squared_error function` chỉ ra rằng bạn đang sử dụng phiên bản cũ hơn của scikit-learn. Tham số `squared=False` đã được giới thiệu trong một phiên bản thư viện sau này, được sử dụng để trả về Lỗi bình phương gốc trung bình (RMSE) thay vì MSE.

Giải pháp: Vì bạn có một phiên bản cũ hơn, hãy xóa tham số `squared=False` khỏi lệnh gọi hàm `mean_squared_error`. Nếu bạn cần RMSE, bạn sẽ cần phải tính toán nó bằng cách thủ công:

```
# Tính Lỗi bình phương trung bình (MSE)
mse = mean_squared_error(y_test, y_pred)

# Tính Lỗi bình phương gốc trung bình (RMSE)
rmse = np.sqrt(mse)

print(f"Mean Squared Error: {mse}")
print(f"Root Mean Squared Error: {rmse}")
```

4. AIMLops Structure (Cấu trúc AIMLops)

You've correctly identified the use of `os.path.join` to handle file paths. Here's how your project structure might look and how it relates to the code:

```
my_chembl_project/
├── data/                   # Contains the chembl_data.csv file
│   └── chembl_data.csv
├── notebooks/              # Contains your Jupyter notebooks
│   └── Topic_CheMBL_35_86_1_data_processing.ipynb
│   └── Topic_CheMBL_35_86_2_model_building.ipynb  # (Example - for a second notebook)
├── scripts/                # (Optional) For reusable Python scripts
│   └── data_extraction.py   # (Example)
├── models/                 # (Optional) For saving trained models
│   └── linear_regression_model.pkl
├── requirements.txt      # List of Python dependencies
└── README.md
```

Your `base_path` in the notebook is set to `".."`. This means your notebook is located one level deeper than the root of your project.

5. Five Examples (Năm ví dụ)

Here are five examples extending the base code:

Example 1: Calculate More RDKit Descriptors (Ví dụ 1: Tính toán nhiều mô tả RDKit hơn)

```
from rdkit.Chem import Lipinski

def calculate_more_descriptors(smiles):
    mol = Chem.MolFromSmiles(smiles)
    if mol is None:
        return None, None, None, None

    # Calculate more descriptors
    mw = Descriptors.MolWt(mol)
    logp = Descriptors.MolLogP(mol)
    hbd = Lipinski.NumHDonors(mol) #Hydrogen Bond Donors
    hba = Lipinski. NumHAcceptors(mol) #Hydrogen Bond Acceptors
```

```
    return mw, logp, hbd, hba

df[['molecular_weight', 'logp', 'hbd', 'hba']] = df['canonical_smiles'].apply(lambda
x: pd.Series(calculate_more_descriptors(x)))

df = df.dropna()
X = df[['molecular_weight', 'logp', 'hbd', 'hba']]
```

Example 2: Different Machine Learning Model (Ví dụ 2: Mô hình học máy khác)

```
from sklearn.ensemble import RandomForestRegressor

#Create a Random Forest Regressor model
model = RandomForestRegressor(n_estimators=100, random_state=42) #You can tune
n_estimators

#Train the model
model.fit(X_train, y_train)
```

Example 3: Using pIC50 instead of pChEMBLValue (Ví dụ 3: Sử dụng pIC50 thay vì pChEMBLValue)

First, make sure your SQL includes the `pchembl_value` . Then, in your python script:

```
y = df['pchembl_value'] # Target: pChEMBL value
```

Example 4: Filtering by a Specific Target (Ví dụ 4: Lọc theo mục tiêu cụ thể)

Modify your SQL query to target a particular protein, for example:

```
WHERE
    td.target_name = 'Tyrosine-protein kinase SYK'
    AND act.standard_type = 'IC50'
```

Example 5: Saving the Trained Model (Ví dụ 5: Lưu mô hình đã huấn luyện)

```
import pickle

#After training your model:
model_file = os.path.join(base_path, "models", "linear_regression_model.pkl")
pickle.dump(model, open(model_file, 'wb'))

#To load the model later:
loaded_model = pickle.load(open(model_file, 'rb'))
```

Key Takeaways:

- **Address the SQL error** by casting `act.standard_value` to text and using a regular expression to check for numeric values.
- **Remove `squared=False`** from `mean_squared_error` or upgrade your scikit-learn version.
- **Pay attention to file paths** and use `os.path.join` to ensure your code works consistently.
- **Adapt the examples** to your specific research question and the data you're working with.

Remember to install the necessary libraries: `pip install pandas rdkit-pypi scikit-learn`. Good luck with your analysis! I am here if you have more questions.