PharmApp Suite

🧠 AI for Drug Discovery and Development 💉

🧩 Topic: Multi-target ligand profiling

Okay, I understand. You're working on a Chembl 35 data analysis project using RDKit within an MLOps-structured environment. You have a PostgreSQL database, and you're facing some challenges in your SQL and Python code. I'll help you analyze the problem, provide corrected code snippets, and offer guidance.

1. Analysis of the Analysis Model and Challenges

Your project aims to leverage the Chembl 35 database for drug research and development. This is a common and valuable use case. You're using RDKit for chemical informatics, which is excellent. The MLOps structure is good for organization and reproducibility.

The errors you're encountering are:

- **SQL Error (a):** **`ERROR: operator does not exist: numeric ~ unknown, LINE 12: AND act.standard_value ~ '^[0-9\.]+$'`** This means PostgreSQL is having trouble using the regular expression operator `~` on a numeric column (`act.standard_value`). This likely means the column is either not a text type, or that your regular expression is trying to match values that are not compatible with the way PostgreSQL handles numeric columns.
- **Python Error (b):** **`old scikit-learn version does not support parameters squared=False in the mean_squared_error function`** This indicates you're using an older version of scikit-learn that doesn't support the `squared=False` argument in `mean_squared_error`. This argument is used to calculate the Root Mean Squared Error (RMSE) instead of Mean Squared Error (MSE).

Analysis Model Approach:

Your analysis approach should ideally involve the following steps:

1. **Data Extraction (SQL):** Extract relevant data from the Chembl 35 database using SQL queries. This will likely include compound IDs, activity values (IC50, Ki, etc.), and target information. Limit to 100 rows for development.
2. **Data Preprocessing (Python):**
    - Clean and filter the extracted data.
    - Convert SMILES strings to RDKit Mol objects.
    - Calculate molecular descriptors using RDKit (e.g., molecular weight, logP, number of hydrogen bond donors/acceptors).
    - Handle missing values or outliers.
3. **Feature Engineering (Python):** Create new features based on the existing data or domain knowledge. This could involve combining descriptors, creating interaction terms, or generating fingerprints.
4. **Model Building (Python):** Choose appropriate machine learning models for your task (e.g., regression for predicting activity, classification for predicting binding affinity).
5. **Model Evaluation (Python):** Evaluate the model's performance using appropriate metrics (e.g., RMSE, R-squared for regression; accuracy, precision, recall, F1-score for classification).

2. Code (SQL and Python)

Here's the code with the corrections and examples, presented in English and then translated into Vietnamese.

English:

SQL Code (save as `../data/Topic_CheMBL_35_25.csv`):

```sql
-- SQL query to extract data from ChEMBL 35 (limited to 100 rows)
-- Ensure you've loaded the ChEMBL data into your PostgreSQL database.

SELECT
    cmp.chembl_id,
    act.standard_value,
    act.standard_units,
    act.standard_type,
    mol.molfile
FROM
    compound_structures mol
JOIN
    activities act ON mol.molregno = act.molregno
JOIN
    molecule_dictionary cmp ON act.molregno = cmp.molregno
WHERE act.standard_type = 'IC50'  -- Filter for IC50 values
    AND act.standard_units = 'nM'  -- Filter for nM units
    AND act.standard_value IS NOT NULL
    AND mol.molfile IS NOT NULL
    AND act.standard_value::text ~ '^[0-9\.]+$'  -- Ensure standard_value is a number
ORDER BY act.standard_value ASC
LIMIT 100;
```

Explanation:

- **`act.standard_value::text ~ '^[0-9\.]+$'`:** This is the crucial fix. We explicitly cast the `standard_value` to text before applying the regular expression. This ensures that the regular expression operator works correctly.
- The query now filters to only include rows where `standard_value` is numeric.
- The `LIMIT 100` clause ensures you only get 100 rows, as requested.
- `mol.molfile IS NOT NULL` is added to ensure no null values.

Python Code (save as `notebook/Topic_CheMBL_35_25_1_data_preprocessing.ipynb`):

```python
import pandas as pd
import numpy as np
from rdkit import Chem
from rdkit.Chem import Descriptors
import os
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 for your project
base_path = ".."  # Assuming the notebook is in a 'notebook' subdirectory

# Construct the full path to your CSV file
csv_file_path = os.path.join(base_path, "data", "Topic_CheMBL_35_25.csv")

# Load the data from the CSV file
try:
```

```
    df = pd.read_csv(csv_file_path)
    print("Data loaded successfully!")
except FileNotFoundError:
    print(f"Error: File not found at {csv_file_path}")
    df = None # Or handle the error as appropriate

if df is not None:
    # Data Cleaning and Preprocessing
    df = df.dropna(subset=['molfile', 'standard_value'])  # Remove rows with missing
values
    df = df[df['standard_value'] > 0]  # Remove zero or negative activity values
    df['standard_value'] = pd.to_numeric(df['standard_value'])
    df['pIC50'] = -np.log10(df['standard_value'] * 1e-9)  # Convert IC50 to pIC50

    # RDKit processing
    def smiles_from_molfile(molfile):
        try:
            mol = Chem.MolFromMolBlock(molfile)
            if mol is not None:
                return Chem.MolToSmiles(mol)
            else:
                return None
        except:
            return None

    df['SMILES'] = df['molfile'].apply(smiles_from_molfile)
    df = df.dropna(subset=['SMILES'])

    def calculate_descriptors(smiles):
        mol = Chem.MolFromSmiles(smiles)
        if mol is not None:
            descriptors = {}
            descriptors['MolWt'] = Descriptors.MolWt(mol)
            descriptors['LogP'] = Descriptors.MolLogP(mol)
            descriptors['HBD'] = Descriptors.NumHDonors(mol)
            descriptors['HBA'] = Descriptors.NumHAcceptors(mol)
            return pd.Series(descriptors)
        else:
            return pd.Series([None] * 4, index=['MolWt', 'LogP', 'HBD', 'HBA']) #
return None Series

    df[['MolWt', 'LogP', 'HBD', 'HBA']] = df['SMILES'].apply(calculate_descriptors)
    df = df.dropna()

    # Feature Selection
    features = ['MolWt', 'LogP', 'HBD', 'HBA']
    X = df[features]
    y = df['pIC50']

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

    # Model Training
    model = LinearRegression()
    model.fit(X_train, y_train)

    # Model Evaluation
    y_pred = model.predict(X_test)
    mse = mean_squared_error(y_test, y_pred)
```

```
rmse = np.sqrt(mse)  # Calculate RMSE manually
r2 = r2_score(y_test, y_pred)

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

Explanation:

- **File Path Handling:** Uses `os.path.join` to construct the file path.
- **Data Loading Error Handling:** Includes a `try...except` block to handle the case where the CSV file is not found.
- **Data Cleaning:** Drops rows with missing `molfile` or `standard_value`.
- **pIC50 Calculation:** Converts IC50 values to pIC50 for a more suitable scale for modeling.
- **RDKit Integration:**
  - Converts SMILES strings to RDKit Mol objects.
  - Calculates molecular descriptors (MolWt, LogP, HBD, HBA) using RDKit. Handles cases where SMILES conversion fails.
- **Feature Selection and Splitting:** Selects the calculated descriptors as features and splits the data into training and testing sets.
- **Model Training:** Trains a simple linear regression model.
- **Model Evaluation:** Calculates MSE, RMSE, and R-squared. Note that `squared=False` is *not* used in `mean_squared_error` because you're on an older version of scikit-learn. Instead, the RMSE is calculated manually.
- Error handling is implemented when creating SMILES from molfile, if can't create SMILES, return None
- `descriptors = {}` in def calculate_descriptors(smiles): is to avoid error
- return pd.Series([None] * 4, index=['MolWt', 'LogP', 'HBD', 'HBA']) to keep the number of columns consistent.
- df = df.dropna() to delete rows containing missing values, such as calculating molecular descriptors.

3. Five Examples

Here are five small modifications you can make to this code to experiment with different aspects of the analysis.

1. **Different Descriptors:** Change the list of descriptors being calculated in the `calculate_descriptors` function. Try adding TPSA (Topological Polar Surface Area), number of rotatable bonds, or other common descriptors.

```
def calculate_descriptors(smiles):
    mol = Chem.MolFromSmiles(smiles)
    if mol is not None:
        descriptors = {}
        descriptors['MolWt'] = Descriptors.MolWt(mol)
        descriptors['LogP'] = Descriptors.MolLogP(mol)
        descriptors['HBD'] = Descriptors.NumHDonors(mol)
        descriptors['HBA'] = Descriptors.NumHAcceptors(mol)
        descriptors['TPSA'] = Descriptors.TPSA(mol)  # Added TPSA
        return pd.Series(descriptors)
    else:
        return pd.Series([None] * 5, index=['MolWt', 'LogP', 'HBD', 'HBA',
'TPSA']) # return None Series
```

2. **Different Model:** Swap out the `LinearRegression` model for another model, such as `RandomForestRegressor` or `GradientBoostingRegressor`. You'll need to import the new model.

```
from sklearn.ensemble import RandomForestRegressor

# ... (rest of the code)

# Model Training
model = RandomForestRegressor(n_estimators=100, random_state=42)  # Example parameters
model.fit(X_train, y_train)
```

3. **Different Activity Type:** Modify the SQL query to use a different `standard_type`, such as 'Ki' or 'EC50', instead of 'IC50'.

```
WHERE act.standard_type = 'Ki'
```

4. **Different Splitting Strategy:** Change the `test_size` parameter in `train_test_split` to adjust the ratio of training to testing data.

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)  # 30% test data
```

5. **Data Scaling:** Apply feature scaling to the descriptor data before training the model. This can often improve model performance.

```
from sklearn.preprocessing import StandardScaler

# ... (rest of the code)

# Data Scaling
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Model Training (using scaled data)
model = LinearRegression()
model.fit(X_train, y_train)
```

Vietnamese Translation:

1. Phân tích Mô hình Phân tích và Thách thức

Dự án của bạn nhằm mục đích tận dụng cơ sở dữ liệu Chembl 35 cho nghiên cứu và phát triển thuốc. Đây là một trường hợp sử dụng phổ biến và có giá trị. Bạn đang sử dụng RDKit cho tin học hóa học, điều này rất tuyệt vời. Cấu trúc MLOps rất tốt cho việc tổ chức và khả năng tái tạo.

Các lỗi bạn đang gặp phải là:

- **Lỗi SQL (a):** **`ERROR: operator does not exist: numeric ~ unknown, LINE 12: AND act.standard_value ~ '^[0-9\.]+$'`** Điều này có nghĩa là PostgreSQL gặp khó khăn khi sử dụng toán tử biểu thức chính quy ~ trên cột số (`act.standard_value`). Điều này có nghĩa là cột không phải là kiểu văn bản hoặc biểu thức chính quy của bạn đang cố gắng khớp các giá trị không tương thích với cách PostgreSQL xử lý các cột số.
- **Lỗi Python (b):** **`old scikit-learn version does not support parameters squared=False in the mean_squared_error function`** Điều này chỉ ra rằng bạn đang sử dụng phiên bản cũ của scikit-learn không hỗ trợ đối số `squared=False` trong `mean_squared_error`. Đối số này được sử dụng để tính Root Mean Squared Error (RMSE) thay vì Mean Squared Error (MSE).

Cách tiếp cận Mô hình Phân tích:

Cách tiếp cận phân tích của bạn nên lý tưởng bao gồm các bước sau:

1. **Trích xuất Dữ liệu (SQL):** Trích xuất dữ liệu liên quan từ cơ sở dữ liệu Chembl 35 bằng các truy vấn SQL. Điều này có thể bao gồm ID hợp chất, giá trị hoạt động (IC50, Ki, v.v.) và thông tin mục tiêu. Giới hạn ở 100 hàng để phát triển.
2. **Tiền xử lý Dữ liệu (Python):**
    - Làm sạch và lọc dữ liệu đã trích xuất.
    - Chuyển đổi chuỗi SMILES thành đối tượng RDKit Mol.
    - Tính toán các descriptor phân tử bằng RDKit (ví dụ: trọng lượng phân tử, logP, số lượng người cho/nhận liên kết hydro).
    - Xử lý các giá trị bị thiếu hoặc ngoại lệ.
3. **Thiết kế Tính năng (Python):** Tạo các tính năng mới dựa trên dữ liệu hiện có hoặc kiến thức miền. Điều này có thể bao gồm kết hợp các descriptor, tạo các số hạng tương tác hoặc tạo dấu vân tay.
4. **Xây dựng Mô hình (Python):** Chọn các mô hình học máy phù hợp cho tác vụ của bạn (ví dụ: hồi quy để dự đoán hoạt động, phân loại để dự đoán ái lực liên kết).
5. **Đánh giá Mô hình (Python):** Đánh giá hiệu suất của mô hình bằng các chỉ số phù hợp (ví dụ: RMSE, R-squared cho hồi quy; độ chính xác, độ chính xác, độ thu hồi, F1-score cho phân loại).

2. Mã (SQL và Python)

Đây là mã với các chỉnh sửa và ví dụ, được trình bày bằng tiếng Anh và sau đó được dịch sang tiếng Việt.

SQL Code (lưu thành `../data/Topic_CheMBL_35_25.csv`):

```sql
-- Truy vấn SQL để trích xuất dữ liệu từ ChEMBL 35 (giới hạn 100 hàng)
-- Đảm bảo bạn đã tải dữ liệu ChEMBL vào cơ sở dữ liệu PostgreSQL của mình.

SELECT
    cmp.chembl_id,
    act.standard_value,
    act.standard_units,
    act.standard_type,
    mol.molfile
FROM
    compound_structures mol
JOIN
    activities act ON mol.molregno = act.molregno
JOIN
    molecule_dictionary cmp ON act.molregno = cmp.molregno
WHERE act.standard_type = 'IC50'  -- Lọc cho các giá trị IC50
    AND act.standard_units = 'nM'  -- Lọc cho các đơn vị nM
    AND act.standard_value IS NOT NULL
    AND mol.molfile IS NOT NULL
    AND act.standard_value::text ~ '^[0-9\.]+$'  -- Đảm bảo standard_value là một số
ORDER BY act.standard_value ASC
LIMIT 100;
```

Giải thích:

- **`act.standard_value::text ~ '^[0-9\.]+$'`:** Đây là bản sửa lỗi quan trọng. Chúng tôi chuyển đổi rõ ràng `standard_value` thành văn bản trước khi áp dụng biểu thức chính quy. Điều này đảm bảo rằng toán tử biểu thức chính quy hoạt động chính xác.

- Truy vấn hiện lọc để chỉ bao gồm các hàng trong đó `standard_value` là số.
- Mệnh đề `LIMIT 100` đảm bảo bạn chỉ nhận được 100 hàng, như đã yêu cầu.
- `mol.molfile IS NOT NULL` được thêm vào để đảm bảo không có giá trị null.

Python Code (lưu thành `notebook/Topic_CheMBL_35_25_1_data_preprocessing.ipynb`):

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

# Xác định đường dẫn cơ sở cho dự án của bạn
base_path = ".."  # Giả sử notebook nằm trong thư mục con 'notebook'

# Xây dựng đường dẫn đầy đủ đến tệp CSV của bạn
csv_file_path = os.path.join(base_path, "data", "Topic_CheMBL_35_25.csv")

# Tải dữ liệu từ tệp CSV
try:
    df = pd.read_csv(csv_file_path)
    print("Dữ liệu đã được tải thành công!")
except FileNotFoundError:
    print(f"Lỗi: Không tìm thấy tệp tại {csv_file_path}")
    df = None # Hoặc xử lý lỗi cho phù hợp

if df is not None:
    # Làm sạch và Tiền xử lý Dữ liệu
    df = df.dropna(subset=['molfile', 'standard_value'])  # Loại bỏ các hàng có giá
trị bị thiếu
    df = df[df['standard_value'] > 0]  # Loại bỏ các giá trị hoạt động bằng không hoặc
âm
    df['standard_value'] = pd.to_numeric(df['standard_value'])
    df['pIC50'] = -np.log10(df['standard_value'] * 1e-9)  # Chuyển đổi IC50 thành
pIC50

    # Xử lý RDKit
    def smiles_from_molfile(molfile):
        try:
            mol = Chem.MolFromMolBlock(molfile)
            if mol is not None:
                return Chem.MolToSmiles(mol)
            else:
                return None
        except:
            return None

    df['SMILES'] = df['molfile'].apply(smiles_from_molfile)
    df = df.dropna(subset=['SMILES'])

    def calculate_descriptors(smiles):
        mol = Chem.MolFromSmiles(smiles)
        if mol is not None:
            descriptors = {}
            descriptors['MolWt'] = Descriptors.MolWt(mol)
            descriptors['LogP'] = Descriptors.MolLogP(mol)
```

```
            descriptors['HBD'] = Descriptors.NumHDonors(mol)
            descriptors['HBA'] = Descriptors.NumHAcceptors(mol)
            return pd.Series(descriptors)
        else:
            return pd.Series([None] * 4, index=['MolWt', 'LogP', 'HBD', 'HBA']) # return None Series

    df[['MolWt', 'LogP', 'HBD', 'HBA']] = df['SMILES'].apply(calculate_descriptors)
    df = df.dropna()

    # Lựa chọn Tính năng
    features = ['MolWt', 'LogP', 'HBD', 'HBA']
    X = df[features]
    y = df['pIC50']

    # 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
    model = LinearRegression()
    model.fit(X_train, y_train)

    # Đánh giá Mô hình
    y_pred = model.predict(X_test)
    mse = mean_squared_error(y_test, y_pred)
    rmse = np.sqrt(mse)  # Tính RMSE thủ công
    r2 = r2_score(y_test, y_pred)

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

Giải thích:

- **Xử lý Đường dẫn Tệp:** Sử dụng `os.path.join` để xây dựng đường dẫn tệp.
- **Xử lý Lỗi Tải Dữ liệu:** Bao gồm một khối `try...except` để xử lý trường hợp không tìm thấy tệp CSV.
- **Làm sạch Dữ liệu:** Loại bỏ các hàng có `molfile` hoặc `standard_value` bị thiếu.
- **Tính toán pIC50:** Chuyển đổi các giá trị IC50 thành pIC50 để có thang đo phù hợp hơn cho mô hình hóa.
- **Tích hợp RDKit:**
    - Chuyển đổi chuỗi SMILES thành đối tượng RDKit Mol.
    - Tính toán các descriptor phân tử (MolWt, LogP, HBD, HBA) bằng RDKit. Xử lý các trường hợp chuyển đổi SMILES không thành công.
- **Lựa chọn và Chia Tính năng:** Chọn các descriptor đã tính làm tính năng và chia dữ liệu thành bộ đào tạo và bộ kiểm tra.
- **Huấn luyện Mô hình:** Huấn luyện một mô hình hồi quy tuyến tính đơn giản.
- **Đánh giá Mô hình:** Tính toán MSE, RMSE và R-squared. Lưu ý rằng `squared=False` *không* được sử dụng trong `mean_squared_error` vì bạn đang sử dụng phiên bản cũ hơn của scikit-learn. Thay vào đó, RMSE được tính thủ công.
- Xử lý lỗi được thực hiện khi tạo SMILES từ molfile, nếu không thể tạo SMILES, hãy trả về None
- `descriptors = {}` trong def calculate_descriptors(smiles): để tránh lỗi
- return pd.Series([None] * 4, index=['MolWt', 'LogP', 'HBD', 'HBA']) để giữ cho số lượng cột nhất quán.

- df = df.dropna() để xóa các hàng chứa giá trị bị thiếu, chẳng hạn như tính toán các descriptor phân tử.

3. Năm Ví dụ

Dưới đây là năm sửa đổi nhỏ mà bạn có thể thực hiện đối với mã này để thử nghiệm các khía cạnh khác nhau của phân tích.

1. **Các Descriptor Khác nhau:** Thay đổi danh sách các descriptor được tính trong hàm `calculate_descriptors`. Hãy thử thêm TPSA (Topological Polar Surface Area), số lượng liên kết có thể xoay hoặc các descriptor phổ biến khác.

```
def calculate_descriptors(smiles):
    mol = Chem.MolFromSmiles(smiles)
    if mol is not None:
        descriptors = {}
        descriptors['MolWt'] = Descriptors.MolWt(mol)
        descriptors['LogP'] = Descriptors.MolLogP(mol)
        descriptors['HBD'] = Descriptors.NumHDonors(mol)
        descriptors['HBA'] = Descriptors.NumHAcceptors(mol)
        descriptors['TPSA'] = Descriptors.TPSA(mol)  # Đã thêm TPSA
        return pd.Series(descriptors)
    else:
        return pd.Series([None] * 5, index=['MolWt', 'LogP', 'HBD', 'HBA',
'TPSA']) # return None Series
```

2. **Mô hình Khác nhau:** Thay thế mô hình `LinearRegression` bằng một mô hình khác, chẳng hạn như `RandomForestRegressor` hoặc `GradientBoostingRegressor`. Bạn cần nhập mô hình mới.

```
from sklearn.ensemble import RandomForestRegressor

# ... (phần còn lại của mã)

# Huấn Luyện Mô hình
model = RandomForestRegressor(n_estimators=100, random_state=42)  # Ví dụ về các
tham số
model.fit(X_train, y_train)
```

3. **Loại Hoạt động Khác nhau:** Sửa đổi truy vấn SQL để sử dụng `standard_type` khác, chẳng hạn như 'Ki' hoặc 'EC50', thay vì 'IC50'.

```
WHERE act.standard_type = 'Ki'
```

4. **Chiến lược Phân chia Khác nhau:** Thay đổi tham số `test_size` trong `train_test_split` để điều chỉnh tỷ lệ dữ liệu đào tạo và kiểm tra.

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=42)  # 30% dữ liệu kiểm tra
```

5. **Chia Tỷ lệ Dữ liệu:** Áp dụng chia tỷ lệ tính năng cho dữ liệu descriptor trước khi huấn luyện mô hình. Điều này thường có thể cải thiện hiệu suất của mô hình.

```
from sklearn.preprocessing import StandardScaler

# ... (phần còn lại của mã)

# Chia Tỷ lệ Dữ liệu
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

```
# Huấn luyện Mô hình (sử dụng dữ liệu đã chia tỷ lệ)
model = LinearRegression()
model.fit(X_train, y_train)
```

I hope this comprehensive analysis and code helps you move forward with your project! Let me know if you have any further questions.