PharmApp Suite

🧠 AI for Drug Discovery and Development 💉

🧩 Topic: Comparative analysis of antihypertensive agents

Okay, I understand. You're working on a drug R&D project using ChEMBL_35 data, leveraging RDKit in a Jupyter Notebook environment organized according to an AI/MLOps template. You're encountering a PostgreSQL error during SQL query execution and an incompatibility issue with your scikit-learn version in your Python code. You need guidance on how to proceed, including sample code to extract 100 rows of data while addressing these issues.

Here's a breakdown, analysis, and guidance in both English and Vietnamese:

1. Analysis of the Project and Issues

- **Project Goal:** To use ChEMBL data (version 35) to inform drug research and development using RDKit.
- **Data Source:** ChEMBL_35 database accessible via PostgreSQL.
- **Tools:** PostgreSQL, RDKit, scikit-learn (presumably for modeling), Jupyter Notebook.
- **Infrastructure:** Specific server details (IP, user, password, database name) provided.
- **Issue 1 (SQL Error):** `ERROR: operator does not exist: numeric ~ unknown, LINE 12: AND act.standard_value ~ '^[0-9\.]+$'`
    - **Cause:** This error arises because you're trying to use a regular expression operator (`~`) to compare a numeric column (`act.standard_value`) with a text pattern (`'^[0-9\.]+$'`). PostgreSQL's `~` operator is primarily designed for text matching. You're trying to check if the `standard_value` is a valid number string.
    - **Solution:** You need to cast the `standard_value` to text or, preferably, use a different approach to check if the value represents a valid number. A more robust solution would involve checking if the value is not NULL and is greater than or equal to zero (assuming `standard_value` should be non-negative).
- **Issue 2 (scikit-learn Error):** "old scikit-learn version does not support parameters squared=False in the mean_squared_error function"
    - **Cause:** Your scikit-learn version is older than the one where the `squared` parameter was introduced in `mean_squared_error`. The default behavior of `mean_squared_error` changed between scikit-learn versions.
    - **Solution:** There are two options:
        1. **Upgrade scikit-learn:** The easiest and recommended approach is to upgrade to a newer version of scikit-learn (e.g., using `pip install --upgrade scikit-learn`).
        2. **Modify the code:** If upgrading is not feasible (e.g., due to environment constraints), calculate the root mean squared error (RMSE) manually by taking the square root of the output of the `mean_squared_error` function.

2. SQL Code (to be run on pgAdmin and saved as .csv)

```sql
-- Topic_CheMBL_35_93.sql

-- Extract 100 rows of data from ChEMBL.
-- Addresses the "operator does not exist" error by avoiding regex on numeric values.

SELECT
    act.activity_id,
```

```
    act.standard_type,
    act.standard_relation,
    act.standard_value,
    act.standard_units,
    mol.molfile AS molfile
FROM
    activities act
JOIN
    molecule_dictionary mol ON act.molregno = mol.molregno
WHERE
    act.standard_type = 'IC50'  -- Example filter: Select only IC50 values
    AND act.standard_relation = '='  -- Example filter: Select only exact values
    AND act.standard_value IS NOT NULL  -- Ensure standard_value is not null
    AND act.standard_value >= 0  -- Ensure standard_value is non-negative
LIMIT 100;

-- Save the result as a CSV file using pgAdmin's export functionality.
```

Explanation:

- We select a limited set of columns relevant to activity and molecule information. The `molfile` column (containing the molecule structure in MDL molfile format) is included, which can be read by RDKit.
- We use `WHERE` clause to filter the data based on `standard_type`, `standard_relation` and non-null `standard_value` that is also non-negative..
- `LIMIT 100` restricts the result set to 100 rows. Adjust as needed.
- **Important:** After running this query in pgAdmin, use the export feature to save the result as a CSV file (e.g., `../data/Topic_CheMBL_35_93.csv`).

3. Python Code (in Jupyter Notebook: `Topic_CheMBL_35_93_1_DataImport.ipynb`)

```
# Topic_CheMBL_35_93_1_DataImport.ipynb

import os
import pandas as pd
from rdkit import Chem
from rdkit.Chem import AllChem
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

# Define base path for data directory
base_path = os.path.abspath(os.path.join(os.getcwd(), '..'))
data_dir = os.path.join(base_path, 'data')
file_path = os.path.join(data_dir, 'Topic_CheMBL_35_93.csv')

# Load the CSV file into a Pandas DataFrame
try:
    df = pd.read_csv(file_path)
    print(f"Data loaded successfully from {file_path}")
except FileNotFoundError:
    print(f"Error: File not found at {file_path}")
    exit()

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

# Function to convert molfile to RDKit Mol object
```

```
def molfile_to_mol(molfile_string):
    try:
        mol = Chem.MolFromMolBlock(molfile_string)
        if mol is None:
            print("Warning: Could not read molecule from molfile string.")
            return None
        return mol
    except Exception as e:
        print(f"Error reading molfile: {e}")
        return None

# Apply the function to create RDKit Mol objects
df['ROMol'] = df['molfile'].apply(molfile_to_mol)

# Drop rows where ROMol is None
df = df.dropna(subset=['ROMol'])

# Function to calculate Morgan Fingerprints
def calculate_morgan_fingerprint(mol, radius=2, nBits=2048):
    try:
        fingerprint = AllChem.GetMorganFingerprintAsBitVect(mol, radius, nBits=nBits)
        return np.array(fingerprint)
    except Exception as e:
        print(f"Error calculating fingerprint: {e}")
        return None

# Apply the function to calculate Morgan Fingerprints
df['Morgan_Fingerprint'] = df['ROMol'].apply(lambda x:
calculate_morgan_fingerprint(x))

# Drop rows where Morgan_Fingerprint is None
df = df.dropna(subset=['Morgan_Fingerprint'])

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

# Prepare data for modeling
X = np.stack(df['Morgan_Fingerprint'].values)
y = df['standard_value'].values

# 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)

# Calculate Mean Squared Error and Root Mean Squared Error
mse = mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse)  # Calculate RMSE manually to avoid potential scikit-learn version
issues

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

Explanation:

- **Imports:** Imports necessary libraries: `os`, `pandas`, `rdkit.Chem`, `rdkit.Chem.AllChem`, `numpy`, `sklearn.model_selection`, `sklearn.linear_model`, `sklearn.metrics`.
- **Path Handling:** Uses `os.path.join` to construct the path to the CSV file, making the code more portable.
- **Data Loading:** Loads the CSV data using `pd.read_csv` and handles potential `FileNotFoundError`.
- **Molfile Conversion:** Defines a function `molfile_to_mol` to convert the `molfile` string to an RDKit `Mol` object, with error handling.
- **Fingerprint Calculation:** Defines a function `calculate_morgan_fingerprint` to compute Morgan fingerprints using `AllChem.GetMorganFingerprintAsBitVect`.
- **Data Cleaning:** Handles missing data by dropping rows where the `ROMol` object or fingerprint is `None`.
- **Numeric Conversion:** Converts `standard_value` to numeric, handling any non-numeric values by converting them to NaN and then dropping those rows.
- **Data Preparation:** Prepares the data for modeling by extracting fingerprints and `standard_value`.
- **Train/Test Split:** Splits the data into training and testing sets using `train_test_split`.
- **Model Training:** Trains a linear regression model using `LinearRegression`.
- **Prediction and Evaluation:** Makes predictions on the test set and calculates the Mean Squared Error (MSE). Critically, it calculates the Root Mean Squared Error (RMSE) manually by taking the square root of the MSE. This avoids the potential `squared=False` error with older scikit-learn versions.

4. Example Usage and Further Analysis (in Jupyter Notebook: `Topic_CheMBL_35_93_2_ModelBuilding.ipynb`)

```
# Topic_CheMBL_35_93_2_ModelBuilding.ipynb

import os
import pandas as pd
from rdkit import Chem
from rdkit.Chem import AllChem
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
from sklearn.ensemble import RandomForestRegressor

# Define base path for data directory
base_path = os.path.abspath(os.path.join(os.getcwd(), '..'))
data_dir = os.path.join(base_path, 'data')
file_path = os.path.join(data_dir, 'Topic_CheMBL_35_93.csv')

# Load the CSV file into a Pandas DataFrame
try:
    df = pd.read_csv(file_path)
    print(f"Data loaded successfully from {file_path}")
except FileNotFoundError:
    print(f"Error: File not found at {file_path}")
    exit()

# Function to convert molfile to RDKit Mol object
def molfile_to_mol(molfile_string):
    try:
```

```
        mol = Chem.MolFromMolBlock(molfile_string)
        if mol is None:
            print("Warning: Could not read molecule from molfile string.")
            return None
        return mol
    except Exception as e:
        print(f"Error reading molfile: {e}")
        return None

# Apply the function to create RDKit Mol objects
df['ROMol'] = df['molfile'].apply(molfile_to_mol)

# Drop rows where ROMol is None
df = df.dropna(subset=['ROMol'])

# Function to calculate Morgan Fingerprints
def calculate_morgan_fingerprint(mol, radius=2, nBits=2048):
    try:
        fingerprint = AllChem.GetMorganFingerprintAsBitVect(mol, radius, nBits=nBits)
        return np.array(fingerprint)
    except Exception as e:
        print(f"Error calculating fingerprint: {e}")
        return None

# Apply the function to calculate Morgan Fingerprints
df['Morgan_Fingerprint'] = df['ROMol'].apply(lambda x:
calculate_morgan_fingerprint(x))

# Drop rows where Morgan_Fingerprint is None
df = df.dropna(subset=['Morgan_Fingerprint'])

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

# Prepare data for modeling
X = np.stack(df['Morgan_Fingerprint'].values)
y = df['standard_value'].values

# 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)

# 1. Random Forest Regressor Example
rf_model = RandomForestRegressor(n_estimators=100, random_state=42)  # Example
parameters
rf_model.fit(X_train, y_train)
rf_y_pred = rf_model.predict(X_test)
rf_mse = mean_squared_error(y_test, rf_y_pred)
rf_rmse = np.sqrt(rf_mse)

print("Random Forest Results:")
print(f"  Mean Squared Error: {rf_mse}")
print(f"  Root Mean Squared Error: {rf_rmse}")

# 2. Feature Importance (Example with Random Forest)
if hasattr(rf_model, 'feature_importances_'):
    importances = rf_model.feature_importances_
    print("\nFeature Importances (Top 10):")
```

```
    for i in np.argsort(importances)[-10:]:
        print(f"  Feature {i}: {importances[i]}")

# 3. Log Transformation of y (if appropriate)
# Check if y values are positive and potentially skewed
if all(y > 0):
    y_log = np.log1p(y) #log(1+y)
    X_train, X_test, y_train_log, y_test_log = train_test_split(X, y_log,
test_size=0.2, random_state=42)

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

    # Make predictions
    y_pred = model.predict(X_test)
    y_pred_exp = np.expm1(y_pred)  #Back transform
    # Calculate Mean Squared Error and Root Mean Squared Error
    mse = mean_squared_error(y_test, y_pred_exp)
    rmse = np.sqrt(mse)  # Calculate RMSE manually to avoid potential scikit-learn
version issues

    print(f"Mean Squared Error (after log transformation): {mse}")
    print(f"Root Mean Squared Error (after log transformation): {rmse}")

# 4. Cross-Validation
from sklearn.model_selection import cross_val_score

# Perform cross-validation
scores = cross_val_score(rf_model, X, y, cv=5, scoring='neg_mean_squared_error')
rmse_scores = np.sqrt(-scores)

print("Cross-validation results:")
print(f"RMSE scores: {rmse_scores}")
print(f"Mean RMSE: {rmse_scores.mean()}")

# 5. Try Ridge Regression
from sklearn.linear_model import Ridge

ridge_model = Ridge(alpha=1.0)  # You can adjust alpha (regularization strength)
ridge_model.fit(X_train, y_train)
ridge_y_pred = ridge_model.predict(X_test)
ridge_mse = mean_squared_error(y_test, ridge_y_pred)
ridge_rmse = np.sqrt(ridge_mse)

print("Ridge Regression Results:")
print(f"  Mean Squared Error: {ridge_mse}")
print(f"  Root Mean Squared Error: {ridge_rmse}")
```

Explanation of Examples:

1. **Random Forest Regressor:** Demonstrates using a `RandomForestRegressor` as an alternative model.
2. **Feature Importance:** Shows how to access feature importances from a trained Random Forest model (if available). This can help identify which fingerprint bits are most predictive of activity.
3. **Log Transformation:** Illustrates how to apply a log transformation to the `standard_value` (if appropriate, e.g., if the values are positively skewed). Back transform the predicted value

before calculate MSE. This can sometimes improve model performance if the target variable is not normally distributed.

4. **Cross-Validation:** Demonstrates using cross-validation (`cross_val_score`) to get a more robust estimate of the model's performance. It uses 5-fold cross-validation and calculates the RMSE for each fold.
5. **Ridge Regression:** Tries another linear model, Ridge Regression, which adds regularization to prevent overfitting. The `alpha` parameter controls the strength of the regularization.

Important Considerations:

- **Data Exploration:** Before jumping into modeling, thoroughly explore your data. Look at the distribution of `standard_value`, the types of molecules present, etc.
- **Feature Engineering:** Experiment with different RDKit descriptors and fingerprint types. You might find that other features are more predictive than Morgan fingerprints.
- **Model Selection:** Try a variety of machine learning models (e.g., Support Vector Machines, Gradient Boosting Machines).
- **Hyperparameter Tuning:** Optimize the hyperparameters of your chosen model using techniques like grid search or random search.
- **Validation:** Use a separate validation set to evaluate the final performance of your model. This helps ensure that your model generalizes well to unseen data.
- **ChEMBL License:** Remember to comply with the ChEMBL license agreement when using the data.

Vietnamese Translation:

1. Phân tích dự án và các vấn đề:

- **Mục tiêu dự án:** Sử dụng dữ liệu ChEMBL (phiên bản 35) để hỗ trợ nghiên cứu và phát triển thuốc bằng cách sử dụng RDKit.
- **Nguồn dữ liệu:** Cơ sở dữ liệu ChEMBL_35 có thể truy cập thông qua PostgreSQL.
- **Công cụ:** PostgreSQL, RDKit, scikit-learn (có thể để xây dựng mô hình), Jupyter Notebook.
- **Cơ sở hạ tầng:** Thông tin chi tiết về máy chủ (IP, người dùng, mật khẩu, tên cơ sở dữ liệu) đã được cung cấp.
- **Vấn đề 1 (Lỗi SQL):** `ERROR: operator does not exist: numeric ~ unknown, LINE 12: AND act.standard_value ~ '^[0-9\.]+$'`
    - **Nguyên nhân:** Lỗi này xảy ra do bạn đang cố gắng sử dụng toán tử biểu thức chính quy (~) để so sánh một cột số (`act.standard_value`) với một mẫu văn bản (`'^[0-9\.]+$'`). Toán tử ~ của PostgreSQL chủ yếu được thiết kế để khớp văn bản. Bạn đang cố gắng kiểm tra xem `standard_value` có phải là một chuỗi số hợp lệ hay không.
    - **Giải pháp:** Bạn cần chuyển đổi `standard_value` thành văn bản hoặc, tốt hơn, sử dụng một phương pháp khác để kiểm tra xem giá trị có đại diện cho một số hợp lệ hay không. Một giải pháp mạnh mẽ hơn sẽ liên quan đến việc kiểm tra xem giá trị có phải là NULL hay không và có lớn hơn hoặc bằng không hay không (giả sử `standard_value` phải không âm).
- **Vấn đề 2 (Lỗi scikit-learn):** "old scikit-learn version does not support parameters squared=False in the mean_squared_error function"
    - **Nguyên nhân:** Phiên bản scikit-learn của bạn cũ hơn phiên bản mà tham số `squared` được giới thiệu trong `mean_squared_error`. Hành vi mặc định của `mean_squared_error` đã thay đổi giữa các phiên bản scikit-learn.
    - **Giải pháp:** Có hai lựa chọn:
        1. **Nâng cấp scikit-learn:** Cách dễ nhất và được khuyến nghị là nâng cấp lên phiên bản scikit-learn mới hơn (ví dụ: sử dụng `pip install --upgrade scikit-learn`).

2. **Sửa đổi mã:** Nếu việc nâng cấp không khả thi (ví dụ: do các ràng buộc về môi trường), hãy tính toán sai số bình phương gốc (RMSE) theo cách thủ công bằng cách lấy căn bậc hai của đầu ra của hàm `mean_squared_error`.

2. Mã SQL (chạy trên pgAdmin và lưu thành .csv): (Xem phần trên, đã được dịch trong phần mã)

3. Mã Python (trong Jupyter Notebook: `Topic_CheMBL_35_93_1_DataImport.ipynb`): (Xem phần trên, đã được dịch trong phần mã)

4. Ví dụ Sử dụng và Phân tích thêm (trong Jupyter Notebook: `Topic_CheMBL_35_93_2_ModelBuilding.ipynb`): (Xem phần trên, đã được dịch trong phần mã)

Các lưu ý quan trọng (Tiếng Việt):

- **Khám phá dữ liệu:** Trước khi bắt đầu xây dựng mô hình, hãy khám phá kỹ lưỡng dữ liệu của bạn. Xem xét phân phối của `standard_value`, các loại phân tử hiện có, v.v.
- **Thiết kế đặc trưng (Feature Engineering):** Thử nghiệm với các mô tả và loại vân tay RDKit khác nhau. Bạn có thể thấy rằng các đặc trưng khác có khả năng dự đoán tốt hơn so với vân tay Morgan.
- **Lựa chọn mô hình:** Thử nhiều mô hình học máy khác nhau (ví dụ: Máy vectơ hỗ trợ, Máy tăng cường độ dốc).
- **Điều chỉnh siêu tham số (Hyperparameter Tuning):** Tối ưu hóa các siêu tham số của mô hình bạn đã chọn bằng cách sử dụng các kỹ thuật như tìm kiếm lưới (grid search) hoặc tìm kiếm ngẫu nhiên (random search).
- **Xác thực (Validation):** Sử dụng một tập hợp xác thực riêng biệt để đánh giá hiệu suất cuối cùng của mô hình của bạn. Điều này giúp đảm bảo rằng mô hình của bạn khái quát hóa tốt cho dữ liệu chưa thấy.
- **Giấy phép ChEMBL:** Hãy nhớ tuân thủ thỏa thuận cấp phép ChEMBL khi sử dụng dữ liệu.

This comprehensive guide should get you started. Remember to adjust the code and analysis based on the specifics of your project and the insights you gain from your data exploration. Good luck!