PharmApp Suite

🧠 AI for Drug Discovery and Development 🧪

🧩 Topic: Pediatric drug insights from ChEMBL

Okay, I understand. You're working on a Chembl 35 data analysis project using RDKit within an AIMLops framework. You have a PostgreSQL database instance and want to run SQL queries to extract data, then analyze it using Python in Jupyter notebooks. You've also encountered a few specific errors. Let's break down the approach, code, and examples.

Here's how we'll proceed:

1. Analysis Model:

The core of your project is likely to involve:

- **Data Extraction (SQL):** Retrieving activity data from the Chembl 35 database based on specific criteria (e.g., target, activity type). You'll need to filter the data for relevant assays and activity values. A key goal is to prepare the data for subsequent analysis in Python.
- **Data Preparation (Python/RDKit):** Cleaning, transforming, and preparing the extracted data. This involves:
  - Reading the CSV file created by the SQL query.
  - Converting SMILES strings to RDKit molecules.
  - Handling missing values (e.g., `standard_value`, `standard_units`).
  - Potentially applying transformations to activity values (e.g., converting IC50 to pIC50).
- **Analysis (Python/RDKit/SciKit-Learn):** Performing the desired analysis. This could include:
  - **Descriptive Statistics:** Calculating summary statistics of activity values, molecular properties.
  - **Structure-Activity Relationship (SAR) Analysis:** Examining relationships between molecular structure and activity. This may involve:
    - Calculating molecular descriptors using RDKit.
    - Using machine learning models (e.g., linear regression, random forest) to predict activity from descriptors.
    - Visualizing SAR using scatter plots, heatmaps, etc.
  - **Target-Based Analysis:** Focusing on specific protein targets and analyzing the activities of compounds against those targets.
  - **Clustering:** Grouping compounds based on structural similarity or activity profiles.

2. Addressing the Errors:

- **Error a:** **`ERROR: operator does not exist: numeric ~ unknown, LINE 12: AND act.standard_value ~ '^[0-9\.]+$'`** This error indicates that PostgreSQL doesn't know how to use the regular expression matching operator (~) with a numeric column (`act.standard_value`). You need to cast the `standard_value` column to text.
- **Error b:** **`old scikit-learn version does not support parameters squared=False in the mean_squared_error function`** This means you are using an older version of scikit-learn that does not support `squared=False` in `mean_squared_error`. Either upgrade your scikit-learn version (recommended) or remove the `squared=False` argument. Removing it will return the Mean Squared Error (MSE), and you can take the square root in your code to get the Root Mean Squared Error (RMSE) if you need it.

3. Code (English and Chinese Annotations):

```
-- SQL Code (Topic_CheMBL_35_58.sql)
-- This SQL query retrieves activity data from the Chembl 35 database
-- and saves it to a CSV file.
-- 这个SQL 查询从Chembl 35 数据库中检索活性数据,
-- 并将其保存到CSV 文件中。

-- Connect to the chembl_35 database.
-- 连接到chembl_35 数据库。
\c chembl_35

-- Extract data and save to a CSV file.
-- 提取数据并保存到CSV 文件。
COPY (
    SELECT
        act.activity_id,
        mol.molecule_id,
        mol.chembl_id,
        mol.pref_name,
        mol.canonical_smiles,
        act.assay_id,
        ass.assay_type,
        ass.description AS assay_description,
        act.standard_type,
        act.standard_relation,
        act.standard_value,
        act.standard_units,
        act.pchembl_value,
        t.target_chembl_id,
        t.target_type,
        t.pref_name AS target_name
    FROM
        activities act
    JOIN
        assays ass ON act.assay_id = ass.assay_id
    JOIN
        target_dictionary t ON ass.tid = t.tid
    JOIN
        molecule_dictionary mol ON act.molregno = mol.molregno
    WHERE
        t.target_type = 'SINGLE PROTEIN' -- Filter for single protein targets (筛选单
蛋白靶标)
        AND act.standard_type = 'IC50'      -- Filter for IC50 activity type (筛选IC50
活性类型)
        AND act.standard_relation = '='     -- Filter for exact values (筛选精确值)
        AND act.standard_value IS NOT NULL -- Filter out null activity values (过滤空
活性值)
        AND mol.canonical_smiles IS NOT NULL -- Ensure SMILES is not null (确保SMILES
不为空)
        AND act.standard_value::TEXT ~ '^[0-9\.]+$' -- Corrected line: Cast to TEXT
for regex (已更正的行: 转换为TEXT 以进行正则表达式匹配)
    LIMIT 100 -- Limit to 100 rows (限制为100 行)
) TO '/tmp/chembl_35_ic50_data.csv' WITH CSV HEADER;
-- Save the data to '/tmp/chembl_35_ic50_data.csv' in CSV format with a header.
-- 将数据以CSV 格式保存到'/tmp/chembl_35_ic50_data.csv', 并带有标题。

# Python Code (Topic_CheMBL_35_58_1_Data_Preparation.ipynb)
# This notebook prepares the Chembl 35 data for analysis.
# 这个notebook 准备Chembl 35 数据以进行分析。

import os
```

```python
import pandas as pd
from rdkit import Chem
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
import matplotlib.pyplot as plt

# Define base path (定义基本路径)
base_path = "../data"

# Define the CSV file path (定义CSV 文件路径)
csv_file_path = "/tmp/chembl_35_ic50_data.csv" #os.path.join(base_path,
"chembl_35_ic50_data.csv")

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

# Data Cleaning and Preparation (数据清洗和准备)

# Remove rows with missing SMILES strings (删除SMILES 字符串缺失的行)
df = df.dropna(subset=['canonical_smiles'])

# Convert IC50 to pIC50 (将IC50 转换为pIC50)
# Function to convert IC50 to pIC50 (将IC50 转换为pIC50 的函数)
def ic50_to_pic50(ic50_nM):
    """Converts IC50 (nM) to pIC50."""
    if pd.isna(ic50_nM) or ic50_nM <= 0:
        return np.nan
    pIC50 = -np.log10(ic50_nM / 1e9)  # Convert nM to M and take negative log
    return pIC50

df['pIC50'] = df['standard_value'].apply(ic50_to_pic50)

# Drop rows where pIC50 is NaN (删除pIC50 为NaN 的行)
df = df.dropna(subset=['pIC50'])

# Calculate molecular descriptors (计算分子描述符)
def calculate_descriptors(smiles):
    """Calculates molecular descriptors using RDKit."""
    mol = Chem.MolFromSmiles(smiles)
    if mol is None:
        return 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)
    return descriptors

# Apply the descriptor calculation to each SMILES string (将描述符计算应用于每个SMILES
字符串)
```

```
df['Descriptors'] = df['canonical_smiles'].apply(calculate_descriptors)

# Remove rows where descriptor calculation failed (删除描述符计算失败的行)
df = df.dropna(subset=['Descriptors'])

# Convert the dictionary of descriptors into separate columns (将描述符字典转换为单独的
列)
df = pd.concat([df.drop(['Descriptors'], axis=1), df['Descriptors'].apply(pd.Series)],
axis=1)

# Display the first few rows of the processed data (显示处理后的数据的前几行)
print(df.head())

# Python Code (Topic_CheMBL_35_58_2_SAR_Analysis.ipynb)
# This notebook performs structure-activity relationship (SAR) analysis on the
prepared data.
# 这个notebook 对准备好的数据进行结构-活性关系(SAR) 分析。

import pandas as pd
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

# Load the data (加载数据)
# Assuming the data preparation notebook was run and the DataFrame 'df' is available
# 假设已运行数据准备notebook 并且DataFrame 'df' 可用
try:
    df = pd.read_csv("/tmp/chembl_35_ic50_data.csv")
    print("Data loaded successfully.")
except FileNotFoundError:
    print(f"Error: File not found at /tmp/chembl_35_ic50_data.csv")
    exit()

# Preprocess the data (数据预处理)
df = df.dropna(subset=['canonical_smiles'])

# Convert IC50 to pIC50 (将IC50 转换为pIC50)
# Function to convert IC50 to pIC50 (将IC50 转换为pIC50 的函数)
def ic50_to_pic50(ic50_nM):
    """Converts IC50 (nM) to pIC50."""
    if pd.isna(ic50_nM) or ic50_nM <= 0:
        return np.nan
    pIC50 = -np.log10(ic50_nM / 1e9)  # Convert nM to M and take negative log
    return pIC50

df['pIC50'] = df['standard_value'].apply(ic50_to_pic50)

# Drop rows where pIC50 is NaN (删除pIC50 为NaN 的行)
df = df.dropna(subset=['pIC50'])

# Calculate molecular descriptors (计算分子描述符)
def calculate_descriptors(smiles):
    """Calculates molecular descriptors using RDKit."""
    mol = Chem.MolFromSmiles(smiles)
    if mol is None:
        return 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)
    return descriptors

# Apply the descriptor calculation to each SMILES string (将描述符计算应用于每个SMILES
字符串)
df['Descriptors'] = df['canonical_smiles'].apply(calculate_descriptors)

# Remove rows where descriptor calculation failed (删除描述符计算失败的行)
df = df.dropna(subset=['Descriptors'])

# Convert the dictionary of descriptors into separate columns (将描述符字典转换为单独的
列)
df = pd.concat([df.drop(['Descriptors'], axis=1), df['Descriptors'].apply(pd.Series)],
axis=1)

# Select features (descriptors) and target (pIC50)
# 选择特征（描述符）和目标（pIC50）
features = ['MolWt', 'LogP', 'HBD', 'HBA', 'TPSA']
target = 'pIC50'

# Handle missing values (处理缺失值)
df = df.dropna(subset=features + [target])

X = df[features]
y = df[target]

# Split the data into training and testing sets (将数据分为训练集和测试集)
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 (训练线性回归模型)
model = LinearRegression()
model.fit(X_train, y_train)

# Make predictions on the test set (在测试集上进行预测)
y_pred = model.predict(X_test)

# Evaluate the model (评估模型)
mse = mean_squared_error(y_test, y_pred)
rmse = mse**0.5
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}")

# Plot predicted vs. actual values (绘制预测值与实际值)
plt.scatter(y_test, y_pred)
plt.xlabel("Actual pIC50")
plt.ylabel("Predicted pIC50")
plt.title("Actual vs. Predicted pIC50")
plt.show()
```

4. Five Example Analyses:

Here are five example analyses you could perform using this data, along with the relevant concepts:

- **Example 1: Basic Descriptor Statistics**
    - **Concept:** Descriptive statistics.
    - **Code (add to notebook):** `python     print(df[['MolWt', 'LogP', 'HBD', 'HBA', 'TPSA', 'pIC50']].describe())`
    - **Explanation:** This calculates and prints the mean, standard deviation, min, max, and quartiles for the molecular descriptors and pIC50 values. This provides a basic overview of the distribution of these properties in your dataset.
- **Example 2: Correlation Analysis**
    - **Concept:** Feature correlation.
    - **Code (add to notebook):** `python     import seaborn as sns     correlation_matrix = df[['MolWt', 'LogP', 'HBD', 'HBA', 'TPSA', 'pIC50']].corr()     sns.heatmap(correlation_matrix, annot=True, cmap="coolwarm")     plt.title("Correlation Matrix")     plt.show()`
    - **Explanation:** This calculates the correlation matrix between the descriptors and pIC50. The heatmap visualizes these correlations, showing which descriptors are positively or negatively correlated with activity. This can help identify potentially important descriptors.
- **Example 3: Target-Specific Activity Distribution**
    - **Concept:** Grouped analysis.
    - **Code (add to notebook):** `python     import seaborn as sns     plt.figure(figsize=(12, 6))     sns.boxplot(x='target_name', y='pIC50', data=df)     plt.xticks(rotation=90)     plt.title("pIC50 Distribution by Target")     plt.show()`
    - **Explanation:** This creates boxplots of pIC50 values for each target in your dataset. This allows you to see if certain targets tend to have compounds with higher or lower activity.
- **Example 4: Scatter Plot of Descriptor vs. pIC50**
    - **Concept:** Visualization of SAR.
    - **Code (add to notebook):** `python     plt.scatter(df['LogP'], df['pIC50'])     plt.xlabel("LogP")     plt.ylabel("pIC50")     plt.title("LogP vs. pIC50")     plt.show()`
    - **Explanation:** This creates a scatter plot of LogP vs. pIC50. This allows you to visually inspect the relationship between lipophilicity (LogP) and activity. You can create similar plots for other descriptors.
- **Example 5: Refine the Linear Regression Model with Feature Selection**
- **Concept:** Feature Selection and model improvement.
- **Code (add to notebook):** ```python from sklearn.feature_selection import SelectKBest, f_regression # Select the top 3 features selector = SelectKBest(score_func=f_regression, k=3) selector.fit(X_train, y_train)

  # Get the indices of the selected features selected_features_indices = selector.get_support(indices=True)

  # Get the names of the selected features selected_features = X.columns[selected_features_indices].tolist()

  print(“Selected Features:”, selected_features)

  # Use only the selected features to train the model X_train_selected = X_train[selected_features] X_test_selected = X_test[selected_features]

# Train a linear regression model model_selected = LinearRegression() model_selected.fit(X_train_selected, y_train)

# Make predictions on the test set y_pred_selected = model_selected.predict(X_test_selected)

# Evaluate the model mse_selected = mean_squared_error(y_test, y_pred_selected) rmse_selected = mse_selected**0.5 r2_selected = r2_score(y_test, y_pred_selected)

print(f"Mean Squared Error (Selected Features): {mse_selected}") print(f"Root Mean Squared Error (Selected Features): {rmse_selected}") print(f"R-squared (Selected Features): {r2_selected}")

# Plot predicted vs. actual values plt.scatter(y_test, y_pred_selected) plt.xlabel("Actual pIC50") plt.ylabel("Predicted pIC50") plt.title("Actual vs. Predicted pIC50 (Selected Features)") plt.show() ``* **Explanation:** This usesSelectKBest` to select the top 3 features (descriptors) that are most correlated with pIC50. A linear regression model is then trained using only these selected features, potentially improving model performance and interpretability.

Important Considerations:

- **Database Credentials:** *Never* hardcode your database credentials directly in your code. Use environment variables or a configuration file to store these securely.
- **Error Handling:** Add more robust error handling to your code. For example, handle cases where the SMILES string is invalid and cannot be converted to an RDKit molecule.
- **Data Exploration:** Before building complex models, spend time exploring your data. Look at distributions of activity values, relationships between descriptors, and any potential biases or errors in the data.
- **Model Validation:** Use appropriate model validation techniques (e.g., cross-validation) to ensure your models are robust and generalizable.
- **Feature Engineering:** Consider creating new features from the existing data that might improve your models. For example, you could calculate more complex molecular descriptors or create interaction terms between descriptors.
- **Documentation:** Document your code thoroughly so that others can understand and use it. Include comments explaining the purpose of each section of the code.
- **Unit Testing:** Write unit tests to verify the correctness of your code, especially the functions for calculating descriptors and converting IC50 to pIC50.

I hope this comprehensive guide helps you with your project! Remember to adapt the code and examples to your specific research questions and goals. Good luck!