

Topic: ChEMBL target network visualization

Okay, I understand. You're working on a ChEMBL 35 data analysis project using RDKit and need guidance on organizing your code within an AIMLops template, addressing specific errors, and limiting the data to 100 rows for efficiency. I'll provide a breakdown, code examples, and analysis in both English and Vietnamese.

1. Analysis of the Analysis Model

Your project involves analyzing data from ChEMBL 35 to support drug research and development. A typical analysis model often involves the following steps:

- **Data Extraction and Preparation:** This involves querying the ChEMBL database (using SQL), extracting relevant information about molecules and their activities, and cleaning the data (handling missing values, standardizing units).
- **Feature Engineering:** Using RDKit, you'll generate molecular descriptors (e.g., molecular weight, logP, number of hydrogen bond donors/acceptors, topological polar surface area (TPSA), etc.) that represent the chemical properties of the molecules.
- **Data Analysis:** This includes calculating the following
 - Distribution of molecules: Molecular Weight, LogP, TPSA.
 - Activity Analysis: Identify and analyze compounds with significant activities (e.g., IC50, Ki)
- **Modelling (Optional):** You *could* build a predictive model (e.g., regression, classification) to predict the activity of new molecules based on their descriptors. This is not explicitly requested, so I will leave it out for now.
- **Visualization:** Plotting data distributions, relationships between descriptors and activity, and visualizing molecules.

Key Considerations:

- **Data Integrity:** Ensure the SQL queries accurately extract the required information. Double-check the filters and joins.
- **Descriptor Selection:** Choose RDKit descriptors that are relevant to the biological activity you are investigating.
- **Data Preprocessing:** Properly handle missing values, outliers, and standardize/normalize data before modeling (if you were doing modelling).
- **Error Handling:** Address the SQL error you encountered and version compatibility issues with scikit-learn.

2. Code Examples

Here's how the code could be structured. Remember to replace "TOPIC_CHEMBL_35_35" with the actual topic code.

```
import os
import pandas as pd
from rdkit import Chem
from rdkit.Chem import Descriptors
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
```

```

#Base Path
base_path = os.getcwd()

# Set display options for pandas (optional, but useful)
pd.set_option('display.max_columns', None)
pd.set_option('display.width', 1000)

# Define functions for generating molecular descriptors
def calculate_descriptors(smiles):
    """Calculates RDKit descriptors for a given SMILES string.

    Args:
        smiles (str): SMILES string of the molecule.

    Returns:
        dict: A dictionary of calculated descriptors. Returns None if the SMILES is
        invalid.
    """
    mol = Chem.MolFromSmiles(smiles)
    if mol is None:
        return None # Handle invalid SMILES

    descriptors = {}
    descriptors["MolWt"] = Descriptors.MolWt(mol)
    descriptors["LogP"] = Descriptors.MolLogP(mol)
    descriptors["TPSA"] = Descriptors.TPSA(mol)
    descriptors["HBD"] = Descriptors.NumHDonors(mol)
    descriptors["HBA"] = Descriptors.NumHAcceptors(mol)
    return descriptors

# SQL Code (saved as data/chembl_data.sql)
# This SQL query is designed to retrieve data from the ChEMBL database,
# specifically targeting activity data and relevant compound information.
# The query selects a limited number of rows (100) for efficiency.
# It includes error handling to ensure only valid numeric standard_values are
# selected.
# Make sure to execute this in pgAdmin and export the result as a CSV file

sql_query = """
SELECT
    compd.chembl_id,
    compd.pref_name,
    act.standard_type,
    act.standard_value,
    act.standard_units,
    mol.molfile
FROM
    activities act
JOIN
    molecule_dictionary compd ON act.molregno = compd.molregno
JOIN
    compound_structures mol ON compd.molregno = mol.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 act.standard_value::text ~ '^\\d+(\\.\\d+)?$' -- Ensure standard_value is
numeric
LIMIT 100;

```

```

"""

# Python Code (notebooks/Topic_CheMBL_35_35_1_data_extraction.ipynb)

# Data Loading and Preparation
csv_file_path = os.path.join(base_path, "data", "chembl_data.csv")

try:
    df = pd.read_csv(csv_file_path)
except FileNotFoundError:
    print(f"Error: CSV file not found at {csv_file_path}. Make sure you ran the SQL query and saved the CSV.")
    exit()

print("Original DataFrame:")
print(df.head())
print(df.info())

# Data Cleaning (Handle Missing Values)
df = df.dropna(subset=['molfile']) #Drop if there is no molfile to work with

# Feature Engineering (notebooks/Topic_CheMBL_35_35_2_feature_engineering.ipynb)

# Apply the descriptor calculation to each row
df['descriptors'] = df['molfile'].apply(lambda x:
calculate_descriptors(Chem.MolToSmiles(Chem.MolFromMolBlock(x))))

# Drop rows where descriptor calculation failed (invalid SMILES)
df = df.dropna(subset=['descriptors'])

# Convert the 'descriptors' column (which contains dictionaries) into separate columns
df = pd.concat([df.drop(['descriptors'], axis=1), df['descriptors'].apply(pd.Series)],
axis=1)

print("\nDataFrame with Descriptors:")
print(df.head())

# Data Analysis and Visualization (notebooks/Topic_CheMBL_35_35_3_analysis.ipynb)

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

# Descriptive Statistics
print("\nDescriptive Statistics of Molecular Weight:")
print(df['MolWt'].describe())

# Distribution of Molecular Weight
plt.figure(figsize=(8, 6))
sns.histplot(df['MolWt'], kde=True)
plt.title('Distribution of Molecular Weight')
plt.xlabel('Molecular Weight (g/mol)')
plt.ylabel('Frequency')
plt.show()

# Descriptive Statistics
print("\nDescriptive Statistics of LogP:")
print(df['LogP'].describe())

```

```

# Distribution of LogP
plt.figure(figsize=(8, 6))
sns.histplot(df['LogP'], kde=True)
plt.title('Distribution of LogP')
plt.xlabel('LogP')
plt.ylabel('Frequency')
plt.show()

# Descriptive Statistics
print("\nDescriptive Statistics of TPSA:")
print(df['TPSA'].describe())

# Distribution of TPSA
plt.figure(figsize=(8, 6))
sns.histplot(df['TPSA'], kde=True)
plt.title('Distribution of TPSA')
plt.xlabel('TPSA')
plt.ylabel('Frequency')
plt.show()

# Activity Analysis
print("\nActivity Analysis (First 10 rows):")
print(df[['chembl_id', 'standard_type', 'standard_value', 'standard_units']].head(10))

# Example: Compounds with highest activity (lowest IC50)
print("\nTop 10 Most Active Compounds (Lowest IC50):")
print(df.sort_values('standard_value').head(10)[['chembl_id', 'standard_value']])

```

Error Resolution

- SQL Error:** ERROR: operator does not exist: numeric ~ unknown, LINE 12: AND
act.standard_value ~ '^[\d-\d\.\d]+\d\$'
 - Explanation:** The ~ operator in PostgreSQL is for regular expression matching, but you're trying to use it directly on a numeric column. PostgreSQL needs to explicitly cast the numeric value to text for regex matching.
 - Solution:** Cast the standard_value column to text before using the ~ operator: AND
act.standard_value::text ~ '^[\d+(\.\d+)]?\d\$' The \d+(\.\d+)? is a regular expression that matches an integer with an optional decimal part. The double backslashes \d are needed to escape the backslash for regular expressions within the SQL string. ^ and \$ anchor the regex to the beginning and end of the string respectively.
- Scikit-learn Error:** old scikit-learn version does not support parameters squared=False in the mean_squared_error function
 - Explanation:** The squared=False parameter in mean_squared_error was introduced in a later version of scikit-learn.
 - Solution:** Either upgrade your scikit-learn version (recommended): `pip install --upgrade scikit-learn` OR remove the squared=False argument and take the square root of the result manually: `rmse = np.sqrt(mean_squared_error(y_true, y_pred))`

3. Folder Structure

```

.
├── data
│   └── chembl_data.csv
└── notebooks

```

```

├── Topic_CheMBL_35_35_1_data_extraction.ipynb
├── Topic_CheMBL_35_35_2_feature_engineering.ipynb
├── Topic_CheMBL_35_35_3_analysis.ipynb
└── data
    └── chembl_data.sql

```

4. 5 Examples of Analysis

Here are 5 example analyses you can perform and visualization:

1. **Distribution of LogP for Active vs. Inactive Compounds:** Create two histograms of LogP, one for compounds with IC50 below a certain threshold (active) and one for compounds with IC50 above the threshold (inactive). This helps visualize if there's a relationship between lipophilicity and activity.
2. **Scatter Plot of Molecular Weight vs. LogP:** This helps identify the chemical space covered by your compounds. You can color-code the points by activity to see if certain regions of the chemical space are associated with higher activity.
3. **Correlation Matrix of Descriptors:** Calculate the correlation matrix between all the RDKit descriptors you generated. This helps identify highly correlated descriptors, which might indicate redundancy.
4. **Box Plots of Activity (IC50) for Different Chembl ID:** Compare the range of IC50 values across different Chembl ID. This can help in visualizing the activities.
5. **Tanimoto Similarity Search:** Given a specific molecule, search for similar molecules in your dataset based on Tanimoto similarity of their Morgan fingerprints. This can help identify potential lead compounds. (Note: This requires generating Morgan fingerprints using RDKit.)

5. Vietnamese Translation

1. Phân tích Mô hình Phân tích

Dự án của bạn liên quan đến việc phân tích dữ liệu từ ChEMBL 35 để hỗ trợ nghiên cứu và phát triển thuốc. Một mô hình phân tích điển hình thường bao gồm các bước sau:

- **Trích xuất và Chuẩn bị Dữ liệu:** Điều này bao gồm truy vấn cơ sở dữ liệu ChEMBL (sử dụng SQL), trích xuất thông tin liên quan về các phân tử và hoạt động của chúng, và làm sạch dữ liệu (xử lý các giá trị bị thiếu, chuẩn hóa đơn vị).
- **Thiết kế Đặc trưng:** Sử dụng RDKit, bạn sẽ tạo ra các mô tả phân tử (ví dụ: trọng lượng phân tử, logP, số lượng người cho/nhận liên kết hydro, diện tích bề mặt cực topo (TPSA), v.v.) đại diện cho các đặc tính hóa học của các phân tử.
- **Phân tích Dữ liệu:**
 - Phân phối các phân tử: Trọng lượng phân tử, LogP, TPSA.
 - Phân tích Hoạt động: Xác định và phân tích các hợp chất có hoạt động đáng kể (ví dụ: IC50, Ki)
- **Mô hình hóa (Tùy chọn):** Bạn có thể xây dựng một mô hình dự đoán (ví dụ: hồi quy, phân loại) để dự đoán hoạt động của các phân tử mới dựa trên các mô tả của chúng. Điều này không được yêu cầu rõ ràng, vì vậy tôi sẽ bỏ qua nó.
- **Trực quan hóa:** Vẽ đồ thị phân phối dữ liệu, mối quan hệ giữa các mô tả và hoạt động, và trực quan hóa các phân tử.

Cần nhắc Quan trọng:

- **Tính Toàn vẹn của Dữ liệu:** Đảm bảo các truy vấn SQL trích xuất chính xác thông tin cần thiết. Kiểm tra kỹ các bộ lọc và kết nối.
- **Lựa chọn Mô tả:** Chọn các mô tả RDKit có liên quan đến hoạt động sinh học mà bạn đang điều tra.
- **Tiền Xử lý Dữ liệu:** Xử lý đúng cách các giá trị bị thiếu, các giá trị ngoại lệ và chuẩn hóa/bình thường hóa dữ liệu trước khi mô hình hóa (nếu bạn đang thực hiện mô hình hóa).

- **Xử lý Lỗi:** Giải quyết lỗi SQL bạn gặp phải và các vấn đề về khả năng tương thích phiên bản với scikit-learn.

2. Ví dụ về Mã

(Xem các ví dụ về mã SQL và Python ở trên. Chúng hoạt động như nhau trong ngữ cảnh của dự án của bạn.)

3. Cấu trúc thư mục

(Xem cấu trúc thư mục ở trên.)

4. 5 Ví dụ về Phân tích

Đây là 5 ví dụ phân tích bạn có thể thực hiện:

1. **Phân phối LogP cho các Hợp chất Hoạt động so với Không Hoạt động:** Tạo hai biểu đồ LogP, một cho các hợp chất có IC50 dưới một ngưỡng nhất định (hoạt động) và một cho các hợp chất có IC50 trên ngưỡng (không hoạt động). Điều này giúp trực quan hóa xem có mối quan hệ giữa tính ưa mỡ và hoạt động hay không.
2. **Biểu đồ Phân tán của Trọng lượng Phân tử so với LogP:** Điều này giúp xác định không gian hóa học được bao phủ bởi các hợp chất của bạn. Bạn có thể mã hóa màu cho các điểm theo hoạt động để xem liệu các vùng nhất định của không gian hóa học có liên quan đến hoạt động cao hơn hay không.
3. **Ma trận Tương quan của các Mô tả:** Tính toán ma trận tương quan giữa tất cả các mô tả RDKit bạn đã tạo. Điều này giúp xác định các mô tả tương quan cao, có thể chỉ ra sự dư thừa.
4. **Biểu đồ Hộp của Hoạt động (IC50) cho các ChEMBL ID khác nhau:** So sánh phạm vi giá trị IC50 trên các ChEMBL ID khác nhau. Điều này có thể giúp trực quan hóa các hoạt động.
5. **Tìm kiếm Tương tự Tanimoto:** Cho một phân tử cụ thể, tìm kiếm các phân tử tương tự trong bộ dữ liệu của bạn dựa trên sự tương tự Tanimoto của dấu vân tay Morgan của chúng. Điều này có thể giúp xác định các hợp chất dẫn đầu tiềm năng. (Lưu ý: Điều này yêu cầu tạo dấu vân tay Morgan bằng RDKit.)

Let me know if you have any other questions. Good luck with your project!