PharmApp Suite

🧠 AI for Drug Discovery and Development 💉

🧩 Topic: Lập hồ sơ ligand đa đích

Tuyệt vời! Tôi rất vui được hỗ trợ bạn trong dự án phân tích dữ liệu ChEMBL 35 này. Với kinh nghiệm trong lĩnh vực nghiên cứu và phát triển dược học, tôi sẽ giúp bạn xây dựng một quy trình làm việc hiệu quả, từ truy vấn SQL đến phân tích dữ liệu bằng Python và RDKit.

1. Phân tích Mô hình Phân tích (Analysis of the Analysis Model)

Dựa trên thông tin bạn cung cấp, chúng ta sẽ thực hiện một quy trình phân tích dữ liệu ChEMBL 35 theo các bước sau:

- **Thu thập dữ liệu (Data Acquisition):** Sử dụng SQL để truy vấn cơ sở dữ liệu ChEMBL 35 và trích xuất thông tin cần thiết, lưu trữ dưới dạng file CSV.
- **Tiền xử lý dữ liệu (Data Preprocessing):**
    - Làm sạch dữ liệu: Loại bỏ các giá trị null, xử lý các giá trị ngoại lệ.
    - Chuyển đổi dữ liệu: Chuyển đổi các giá trị hoạt tính (activity values) về dạng số, chuẩn hóa dữ liệu (nếu cần).
- **Phân tích dữ liệu (Data Analysis):**
    - Tính toán các thuộc tính phân tử (Molecular Properties): Sử dụng RDKit để tính toán các thuộc tính vật lý hóa học của các phân tử (ví dụ: LogP, MW, TPSA).
    - Phân tích tương quan (Correlation Analysis): Xác định mối tương quan giữa các thuộc tính phân tử và hoạt tính sinh học.
    - Xây dựng mô hình (Model Building): Sử dụng các thuật toán học máy (ví dụ: Linear Regression, Random Forest) để xây dựng mô hình dự đoán hoạt tính sinh học dựa trên các thuộc tính phân tử.
- **Đánh giá mô hình (Model Evaluation):** Đánh giá hiệu năng của mô hình bằng các chỉ số phù hợp (ví dụ: RMSE, R-squared).
- **Trực quan hóa dữ liệu (Data Visualization):** Sử dụng các biểu đồ để trực quan hóa kết quả phân tích, giúp dễ dàng nhận biết các xu hướng và mối quan hệ quan trọng.

2. Hướng dẫn song ngữ (Bilingual Instructions)

Dưới đây là hướng dẫn chi tiết bằng cả tiếng Anh và tiếng Việt cho từng bước trong quy trình phân tích:

Step 1: Data Acquisition (Bước 1: Thu thập dữ liệu)

- **English:** Use SQL queries to extract relevant data from the ChEMBL 35 database (e.g., target information, compound structures, activity data). Save the results as CSV files.
- **Tiếng Việt:** Sử dụng các truy vấn SQL để trích xuất dữ liệu liên quan từ cơ sở dữ liệu ChEMBL 35 (ví dụ: thông tin về đích tác động, cấu trúc hợp chất, dữ liệu hoạt tính). Lưu kết quả dưới dạng file CSV.

Step 2: Data Preprocessing (Bước 2: Tiền xử lý dữ liệu)

- **English:** Clean the data by removing null values, handling outliers, and converting activity values to numerical format. Consider data normalization if necessary.
- **Tiếng Việt:** Làm sạch dữ liệu bằng cách loại bỏ các giá trị null, xử lý các giá trị ngoại lệ và chuyển đổi các giá trị hoạt tính về dạng số. Cân nhắc chuẩn hóa dữ liệu nếu cần thiết.

Step 3: Data Analysis (Bước 3: Phân tích dữ liệu)

- **English:**
  - Calculate molecular properties using RDKit (e.g., LogP, MW, TPSA).
  - Perform correlation analysis to identify relationships between molecular properties and biological activity.
  - Build predictive models using machine learning algorithms (e.g., Linear Regression, Random Forest).
- **Tiếng Việt:**
  - Tính toán các thuộc tính phân tử bằng RDKit (ví dụ: LogP, MW, TPSA).
  - Thực hiện phân tích tương quan để xác định mối quan hệ giữa các thuộc tính phân tử và hoạt tính sinh học.
  - Xây dựng các mô hình dự đoán bằng cách sử dụng các thuật toán học máy (ví dụ: Linear Regression, Random Forest).

Step 4: Model Evaluation (Bước 4: Đánh giá mô hình)

- **English:** Evaluate the performance of the models using appropriate metrics (e.g., RMSE, R-squared).
- **Tiếng Việt:** Đánh giá hiệu năng của các mô hình bằng các chỉ số phù hợp (ví dụ: RMSE, R-squared).

Step 5: Data Visualization (Bước 5: Trực quan hóa dữ liệu)

- **English:** Use plots and charts to visualize the analysis results and identify key trends and relationships.
- **Tiếng Việt:** Sử dụng các biểu đồ để trực quan hóa kết quả phân tích và xác định các xu hướng và mối quan hệ quan trọng.

3. Code SQL, Python (English)

SQL (to extract 100 rows of data):

```sql
-- SQL query to extract 100 rows of data from ChEMBL
SELECT
    md.molregno,
    cs.canonical_smiles,
    act.standard_type,
    act.standard_value,
    act.standard_units
FROM
    molecule_dictionary md
JOIN
    compound_structures cs ON md.molregno = cs.molregno
JOIN
    activities act ON md.molregno = act.molregno
WHERE act.standard_type = 'IC50'  -- You can change activity type
LIMIT 100;
```

Python (using RDKit and scikit-learn):

```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

# Define base path
base_path = "." # Replace with your actual base path
```

```
# Load data from CSV
data_path = os.path.join(base_path, 'data', 'chembl_data.csv')  # Replace
'chembl_data.csv' with your file name
df = pd.read_csv(data_path)

# Data cleaning and conversion
df = df.dropna(subset=['canonical_smiles', 'standard_value'])
df = df[df['standard_value'].astype(str).str.match(r'^[0-9\.]+$')]  # Corrected regex
df['standard_value'] = pd.to_numeric(df['standard_value'], errors='coerce')
df = df.dropna(subset=['standard_value'])

# RDKit molecular descriptors calculation
def calculate_descriptors(smiles):
    mol = Chem.MolFromSmiles(smiles)
    if mol is None:
        return None
    descriptors = {}
    descriptors['MolLogP'] = Descriptors.MolLogP(mol)
    descriptors['MolWt'] = Descriptors.MolWt(mol)
    descriptors['TPSA'] = Descriptors.TPSA(mol)
    return descriptors

df['descriptors'] = df['canonical_smiles'].apply(calculate_descriptors)
df = df.dropna(subset=['descriptors'])
df = df[df['descriptors'].apply(lambda x: isinstance(x, dict))]

# Convert descriptors to columns
df = pd.concat([df.drop(['descriptors'], axis=1), df['descriptors'].apply(pd.Series)],
axis=1)

# Prepare data for machine learning
X = df[['MolLogP', 'MolWt', 'TPSA']]
y = np.log10(df['standard_value'])  # log transform IC50 values

# 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 prediction
y_pred = model.predict(X_test)

# Model evaluation
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

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

Important Notes:

- **Error Handling (Xử lý lỗi):** The `dropna` and `astype(str).str.match(r'^[0-9\.]+$')` lines in the Python code address the potential issues with missing data and non-numeric activity values. The regular expression `^[0-9\.]+$` ensures that only strings containing numbers and periods are considered valid.

- **Scikit-learn Version:** If you encounter the error related to `squared=False` in `mean_squared_error`, either upgrade your scikit-learn version or remove the `squared=False` argument (the default behavior is `squared=True`).
- **File Paths (Đường dẫn tệp):** Remember to replace `"chembl_data.csv"` with the actual name of your CSV file and update the `base_path` variable to reflect your project directory structure.
- **Activity Type (Loại hoạt tính):** The SQL query currently filters for `standard_type = 'IC50'`. Adjust this condition based on the specific activity type you are interested in.
- **Data Exploration (Khám phá dữ liệu):** Before building the model, it's highly recommended to explore the data using descriptive statistics and visualizations to gain a better understanding of the data distribution and potential relationships.

4. Ví dụ Code SQL và Python (SQL and Python Code Examples)

Here are 5 examples demonstrating various SQL queries and Python data analysis techniques:

SQL Examples:

1. **Extract compounds with specific substructure:**

```sql
-- Find compounds containing a specific substructure (using SMARTS pattern)
SELECT md.molregno, cs.canonical_smiles
FROM molecule_dictionary md
JOIN compound_structures cs ON md.molregno = cs.molregno
WHERE cs.canonical_smiles LIKE '%[CH3][OH]%';  -- Example: finds compounds containing methyl alcohol
```

2. **Find target information for a specific protein:**

```sql
-- Find target information for a specific protein (e.g., by target name)
SELECT td.tid, td.pref_name, td.target_type
FROM target_dictionary td
WHERE td.pref_name LIKE '%Kinase%';
```

3. **Get activity data for a specific compound:**

```sql
-- Get activity data for a specific compound (by molregno)
SELECT act.standard_type, act.standard_value, act.standard_units
FROM activities act
WHERE act.molregno = 727278; -- Replace with a valid molregno
```

4. **Extract compounds with activity against a specific target:**

```sql
-- Extract compounds with activity against a specific target (using target_id)
SELECT md.molregno, cs.canonical_smiles, act.standard_type, act.standard_value
FROM molecule_dictionary md
JOIN compound_structures cs ON md.molregno = cs.molregno
JOIN activities act ON md.molregno = act.molregno
WHERE act.tid = 214; -- Replace with a valid target_id
```

5. **Count number of compounds for each activity type:**

```sql
-- Count the number of compounds for each activity type
SELECT standard_type, COUNT(*) AS compound_count
FROM activities
GROUP BY standard_type
ORDER BY compound_count DESC;
```

Python Examples:

1. **Calculate and visualize the distribution of molecular weight:**

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

```
# Load data (replace with your actual data loading)
data_path = os.path.join(base_path, 'data', 'chembl_data.csv')
df = pd.read_csv(data_path)
df = df.dropna(subset=['canonical_smiles']) # Drop rows with missing smiles

def calculate_mw(smiles):
    mol = Chem.MolFromSmiles(smiles)
    if mol is None:
        return None
    return Descriptors.MolWt(mol)

df['mol_weight'] = df['canonical_smiles'].apply(calculate_mw)
df = df.dropna(subset=['mol_weight']) # Drop rows where MW could not be calculated

plt.hist(df['mol_weight'], bins=50)
plt.xlabel('Molecular Weight')
plt.ylabel('Frequency')
plt.title('Distribution of Molecular Weight')
plt.show()
```

2. **Calculate LogP and plot against activity:**

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

# Load data (replace with your actual data loading)
data_path = os.path.join(base_path, 'data', 'chembl_data.csv')
df = pd.read_csv(data_path)
df = df.dropna(subset=['canonical_smiles', 'standard_value']) # Handle missing data
df = df[df['standard_value'].astype(str).str.match(r'^[0-9\.]+$')] #Handle non numeric
df['standard_value'] = pd.to_numeric(df['standard_value'], errors='coerce') #Convert
to numeric
df = df.dropna(subset=['standard_value']) #Drop NA values
def calculate_logp(smiles):
    mol = Chem.MolFromSmiles(smiles)
    if mol is None:
        return None
    return Descriptors.MolLogP(mol)

df['logp'] = df['canonical_smiles'].apply(calculate_logp)
df = df.dropna(subset=['logp'])

plt.scatter(df['logp'], np.log10(df['standard_value'])) # Log transform activity for
better visualization
plt.xlabel('LogP')
plt.ylabel('Log10(Activity)')
plt.title('LogP vs. Activity')
plt.show()
```

3. **Calculate TPSA and filter compounds based on TPSA value:**

```
import pandas as pd
from rdkit import Chem
from rdkit.Chem import Descriptors

# Load data (replace with your actual data loading)
data_path = os.path.join(base_path, 'data', 'chembl_data.csv')
df = pd.read_csv(data_path)
df = df.dropna(subset=['canonical_smiles']) # Handle missing data
```

```
def calculate_tpsa(smiles):
    mol = Chem.MolFromSmiles(smiles)
    if mol is None:
        return None
    return Descriptors.TPSA(mol)

df['tpsa'] = df['canonical_smiles'].apply(calculate_tpsa)
df = df.dropna(subset=['tpsa'])

# Filter compounds with TPSA less than 100
filtered_df = df[df['tpsa'] < 100]
print(f"Number of compounds with TPSA < 100: {len(filtered_df)}")
```

4. **Create a function to convert SMILES to Mol object and handle errors:**

```
from rdkit import Chem

def smiles_to_mol(smiles):
    """
    Converts a SMILES string to an RDKit Mol object, handling potential errors.

    Args:
        smiles (str): The SMILES string.

    Returns:
        Chem.Mol: The RDKit Mol object, or None if an error occurred.
    """
    try:
        mol = Chem.MolFromSmiles(smiles)
        if mol is None:
            print(f"Warning: Could not create Mol object from SMILES: {smiles}")
            return None
        return mol
    except Exception as e:
        print(f"Error processing SMILES {smiles}: {e}")
        return None

# Example usage:
smiles_list = ['CC(=O)Oc1ccccc1C(=O)O', 'Invalid SMILES']
mol_objects = [smiles_to_mol(s) for s in smiles_list]
print(mol_objects)
```

5. **Calculate multiple descriptors at once:**

```
import pandas as pd
from rdkit import Chem
from rdkit.Chem import Descriptors
import numpy as np

# Load data (replace with your actual data loading)
data_path = os.path.join(base_path, 'data', 'chembl_data.csv')
df = pd.read_csv(data_path)
df = df.dropna(subset=['canonical_smiles']) # Handle missing data

def calculate_descriptors(smiles):
    mol = Chem.MolFromSmiles(smiles)
    if mol is None:
        return None
    return pd.Series({
        'MolLogP': Descriptors.MolLogP(mol),
        'MolWt': Descriptors.MolWt(mol),
```

```
        'TPSA': Descriptors.TPSA(mol),
        'HBA': Descriptors.NumHAcceptors(mol),
        'HBD': Descriptors.NumHDonors(mol)
    })

df = pd.concat([df, df['canonical_smiles'].apply(calculate_descriptors)], axis=1)
print(df.head())
```

Remember to install the necessary libraries (`pandas`, `rdkit`, `scikit-learn`, `matplotlib`) using `pip install pandas rdkit scikit-learn matplotlib`.

This comprehensive guide, along with the code examples, should provide a solid foundation for your ChEMBL 35 data analysis project. Please let me know if you have any further questions or need more specific assistance. Good luck!