PharmApp Suite

🧠 AI for Drug Discovery and Development 💉

🧩 Topic: Violin plot for multi-target activity spectrum

Okay, I understand. You're working on a Chembl 35 data analysis project using RDKit, following an AIMLops-inspired folder structure. You'll be running SQL queries against a PostgreSQL database (Chembl_35) to extract data, saving it as CSV, and then using Jupyter Notebooks with Python and RDKit to analyze it. You're encountering some errors and want to focus on extracting a limited dataset (100 rows) for prototyping. You want guidance in both English and Vietnamese on how to proceed.

Let's break this down step-by-step.

1. Analysis of the Analysis Model

- **Goal:** The primary goal is to extract and analyze data from the ChEMBL 35 database using RDKit. The analysis likely aims to:
    - Identify potentially interesting compounds based on specific criteria (activity, properties, etc.).
    - Explore relationships between chemical structures and biological activity.
    - Build predictive models (QSAR, QSPR) for drug discovery.
- **Data Flow:**
    1. **SQL Query:** Extract relevant data from the Chembl_35 database. You'll be focusing on activity data and molecular properties (likely SMILES strings for the compounds).
    2. **CSV Export:** Save the extracted data as CSV files.
    3. **Data Loading and Preprocessing:** Load the CSV data into a Pandas DataFrame in your Jupyter Notebook. Clean and prepare the data for analysis. This may involve:
        - Handling missing values.
        - Converting data types.
        - Filtering data based on specific criteria.
    4. **RDKit Integration:** Use RDKit to:
        - Convert SMILES strings to RDKit Mol objects.
        - Calculate molecular descriptors (e.g., LogP, molecular weight, topological polar surface area (TPSA)).
    5. **Analysis and Modeling:**
        - Perform exploratory data analysis (EDA) to visualize relationships between descriptors and activity.
        - Build statistical models (e.g., linear regression, random forests) to predict activity based on molecular descriptors.
- **Addressing Errors:**
    - **SQL Error (numeric ~ unknown):** The ~ operator in SQL is used for regular expression matching. The database is likely not recognizing that the `standard_value` column is supposed to be treated as text when applying regular expression matching.
    - **scikit-learn Error (squared=False):** This indicates that you're using an older version of scikit-learn. The `squared=False` parameter in `mean_squared_error` was introduced in a later version.

2. SQL Code (English and Vietnamese)

Here's an SQL query to extract data for 100 rows, addressing the regex issue, and including common fields used in drug discovery analysis. This focuses on activity data and SMILES strings. * **English**

```sql
SELECT
    act.activity_id,
    act.standard_type,
    act.standard_relation,
    act.standard_value,
    act.standard_units,
    act.assay_id,
    mol.molregno,
    mol.smiles,
    cmp.chembl_id
FROM
    activities act
JOIN
    molecule_dictionary mol ON act.molregno = mol.molregno
JOIN
    compound_structures cmp ON mol.molregno = cmp.molregno
WHERE
    act.standard_type IN ('IC50', 'Ki', 'EC50')  -- Common activity types
    AND act.standard_units = 'nM'  -- Focus on nanomolar activity
    AND act.standard_value IS NOT NULL
    AND act.standard_relation = '='
    AND act.standard_value::TEXT SIMILAR TO '[0-9.]+' -- Convert to text for regex
matching
ORDER BY
    act.standard_value ASC
LIMIT 100;
```

- **Vietnamese**

```sql
SELECT
    act.activity_id,
    act.standard_type,
    act.standard_relation,
    act.standard_value,
    act.standard_units,
    act.assay_id,
    mol.molregno,
    mol.smiles,
    cmp.chembl_id
FROM
    activities act
JOIN
    molecule_dictionary mol ON act.molregno = mol.molregno
JOIN
    compound_structures cmp ON mol.molregno = cmp.molregno
WHERE
    act.standard_type IN ('IC50', 'Ki', 'EC50')  -- Các loại hoạt tính phổ biến
    AND act.standard_units = 'nM'  -- Tập trung vào hoạt tính nanomolar
    AND act.standard_value IS NOT NULL
    AND act.standard_relation = '='
    AND act.standard_value::TEXT SIMILAR TO '[0-9.]+' -- Chuyển đổi sang text để so
khớp regex
ORDER BY
    act.standard_value ASC
LIMIT 100;
```

Explanation:

- `SELECT`: Specifies the columns to retrieve.
- `FROM`: Indicates the tables to use (`activities`, `molecule_dictionary`, `compound_structures`).
- `JOIN`: Connects the tables based on common columns (`molregno`).

- `WHERE`: Filters the data based on the following criteria:
    - `act.standard_type IN ('IC50', 'Ki', 'EC50')`: Selects only activities with standard types IC50, Ki, or EC50.
    - `act.standard_units = 'nM'`: Selects only activities with units of nM.
    - `act.standard_value IS NOT NULL`: Excludes rows where the standard value is missing.
    - `act.standard_relation = '='`: Only include data with the = relation which signifies exact value.
    - `act.standard_value::TEXT SIMILAR TO '[0-9.]+'`: **Crucially, this converts the `standard_value` column to TEXT and uses the `SIMILAR TO` operator (PostgreSQL's equivalent of `LIKE` with regular expressions) to ensure that the value is a number (containing only digits and a decimal point). This addresses the `numeric ~ unknown` error.** If `SIMILAR TO` doesn't work, try `CAST(act.standard_value AS TEXT) ~ '^[0-9\.]+$'`.
- `ORDER BY act.standard_value ASC`: Order the results from lowest to highest standard value.
- `LIMIT 100`: Restricts the result set to 100 rows.

How to Use:

1. **Run the SQL query:** Execute this query in pgAdmin (or your preferred PostgreSQL client) connected to your Chembl_35 database.
2. **Export to CSV:** Save the results as a CSV file named something like `chembl_35_activity_data.csv` in your `../data/` directory.

3. Python Code (Jupyter Notebook) - English and Vietnamese

```
# English
import os
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 the base path for your project
base_path = ".."  # Assuming your notebook is in the 'notebooks' directory

# File path to your CSV data
data_file = os.path.join(base_path, "data", "chembl_35_activity_data.csv")

# Load the data
try:
    df = pd.read_csv(data_file)
except FileNotFoundError:
    print(f"Error: File not found at {data_file}")
    exit()

# --- Data Preprocessing ---
# Handle missing SMILES (very important!)
df = df.dropna(subset=['smiles'])

# Convert standard_value to numeric (important for calculations)
df['standard_value'] = pd.to_numeric(df['standard_value'], errors='coerce')
df = df.dropna(subset=['standard_value'])
```

```
# --- RDKit Calculations ---
def calculate_descriptors(smiles):
    mol = Chem.MolFromSmiles(smiles)
    if mol is None:
        return None, None  # Handle invalid SMILES
    mw = Descriptors.MolWt(mol)
    logp = Descriptors.MolLogP(mol)
    tpsa = Descriptors.TPSA(mol)
    return mw, logp, tpsa

# Apply the function to each SMILES string
df[['molecular_weight', 'logp', 'tpsa']] = df['smiles'].apply(lambda x:
pd.Series(calculate_descriptors(x)))

# Drop rows with NaN values after descriptor calculation
df = df.dropna()

# --- Feature Selection and Model Building ---
X = df[['molecular_weight', 'logp', 'tpsa']]  # Features
y = df['standard_value']  # Target variable (activity)

# 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)

# Evaluate the model
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}")

# Example of accessing a specific chembl_id's data
example_chembl_id = df['chembl_id'].iloc[0] # Access the first chembl_id
example_data = df[df['chembl_id'] == example_chembl_id]
print(f"\nExample data for ChEMBL ID: {example_chembl_id}")
print(example_data)

# Vietnamese
import os
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

# Xác định đường dẫn gốc của dự án
base_path = ".."  # Giả sử notebook nằm trong thư mục 'notebooks'

# Đường dẫn đến file CSV chứa dữ liệu
data_file = os.path.join(base_path, "data", "chembl_35_activity_data.csv")
```

```
# Tải dữ liệu
try:
    df = pd.read_csv(data_file)
except FileNotFoundError:
    print(f"Lỗi: Không tìm thấy file tại {data_file}")
    exit()

# --- Tiền xử lý dữ liệu ---
# Xử lý các SMILES bị thiếu (rất quan trọng!)
df = df.dropna(subset=['smiles'])

# Chuyển đổi standard_value sang kiểu số (quan trọng cho tính toán)
df['standard_value'] = pd.to_numeric(df['standard_value'], errors='coerce')
df = df.dropna(subset=['standard_value'])

# --- Tính toán các descriptor bằng RDKit ---
def calculate_descriptors(smiles):
    mol = Chem.MolFromSmiles(smiles)
    if mol is None:
        return None, None  # Xử lý SMILES không hợp lệ
    mw = Descriptors.MolWt(mol)
    logp = Descriptors.MolLogP(mol)
    tpsa = Descriptors.TPSA(mol)
    return mw, logp, tpsa

# Áp dụng hàm cho mỗi chuỗi SMILES
df[['molecular_weight', 'logp', 'tpsa']] = df['smiles'].apply(lambda x:
pd.Series(calculate_descriptors(x)))

# Loại bỏ các hàng có giá trị NaN sau khi tính toán descriptor
df = df.dropna()

# --- Lựa chọn đặc trưng và xây dựng mô hình ---
X = df[['molecular_weight', 'logp', 'tpsa']]  # Các đặc trưng
y = df['standard_value']  # Biến mục tiêu (hoạt tính)

# Chia dữ liệu thành tập huấn luyện và tập kiểm tra
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Huấn luyện mô hình hồi quy tuyến tính
model = LinearRegression()
model.fit(X_train, y_train)

# Dự đoán
y_pred = model.predict(X_test)

# Đánh giá mô hình
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print(f"Sai số bình phương trung bình: {mse}")
print(f"Hệ số R bình phương: {r2}")

# Ví dụ về việc truy cập dữ liệu của một chembl_id cụ thể
example_chembl_id = df['chembl_id'].iloc[0] # Truy cập chembl_id đầu tiên
example_data = df[df['chembl_id'] == example_chembl_id]
```

```
print(f"\nVí dụ dữ liệu cho ChEMBL ID: {example_chembl_id}")
print(example_data)
```

Explanation:

1. **Import Libraries:** Imports necessary libraries (Pandas, RDKit, scikit-learn).
2. **Define `base_path`:** Sets the base path to your project root (two levels up, assuming the notebook is in a `notebooks` directory).
3. **Load Data:** Loads the CSV data into a Pandas DataFrame. Includes error handling for the case where the file is not found.
4. **Data Preprocessing:**
   - `df = df.dropna(subset=['smiles'])`: **Critical:** Removes rows where the `smiles` column is empty. RDKit *will* crash if you try to process an empty SMILES string.
   - `df['standard_value'] = pd.to_numeric(df['standard_value'], errors='coerce')`: Converts the `standard_value` column to numeric. `errors='coerce'` will turn any values that can't be converted to numbers into `NaN`.
   - `df = df.dropna(subset=['standard_value'])`: Remove rows with NaN in 'standard_value' column.
5. **RDKit Calculations:**
   - `calculate_descriptors(smiles)`: A function that takes a SMILES string as input, converts it to an RDKit Mol object, and calculates molecular weight, LogP, and TPSA.
   - Error Handling: The `if mol is None:` check is **essential** to handle invalid SMILES strings. Invalid SMILES will cause `Chem.MolFromSmiles()` to return `None`.
   - `df[['molecular_weight', 'logp', 'tpsa']] = df['smiles'].apply(lambda x: pd.Series(calculate_descriptors(x)))`: Applies the `calculate_descriptors` function to each SMILES string in the DataFrame and creates new columns for the calculated descriptors. `pd.Series` is used to properly unpack the multiple return values into separate columns.
   - `df = df.dropna()`: Removes any rows that now contain `NaN` values (likely because of invalid SMILES strings that resulted in `None` values for the descriptors).
6. **Feature Selection and Model Building:**
   - `X = df[['molecular_weight', 'logp', 'tpsa']]`: Selects the molecular weight, LogP, and TPSA as features. You can add more descriptors here.
   - `y = df['standard_value']`: Sets the `standard_value` column as the target variable (the activity you're trying to predict).
   - `X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)`: Splits the data into training and testing sets (80% for training, 20% for testing). `random_state=42` ensures that the split is reproducible.
   - `model = LinearRegression()`: Creates a linear regression model. You can experiment with other models (e.g., RandomForestRegressor).
   - `model.fit(X_train, y_train)`: Trains the model on the training data.
   - `y_pred = model.predict(X_test)`: Makes predictions on the testing data.
   - `mse = mean_squared_error(y_test, y_pred)` and `r2 = r2_score(y_test, y_pred)`: Evaluates the model using mean squared error and R-squared. Since you may have an older scikit-learn version, you can either:
     - Upgrade scikit-learn: `pip install --upgrade scikit-learn`
     - Remove `squared=False` from the `mean_squared_error` function (the default is `squared=True`, which returns the MSE; removing it is fine for this example).
7. **Example Usage:**
   - Shows how to access the data associated with a specific `chembl_id`.

Important Considerations:

- **Error Handling:** The code includes error handling for file not found and invalid SMILES strings. Robust error handling is crucial in real-world data analysis.
- **Data Cleaning:** Thoroughly cleaning your data is *essential*. Pay close attention to missing values, invalid SMILES strings, and inconsistent units.
- **Feature Engineering:** Experiment with different molecular descriptors. RDKit provides a wide range of descriptors that can be used as features in your models.
- **Model Selection:** Linear regression is a simple model. Consider more complex models (e.g., random forests, support vector machines) for better performance.
- **Hyperparameter Tuning:** Optimize the hyperparameters of your models to improve their accuracy.

4. AIMLops Folder Structure

Assuming an AIMLops-inspired structure, your folder structure might look like this:

```
project_root/
    data/               # Contains raw data (chembl_35_activity_data.csv)
    notebooks/          # Contains Jupyter notebooks (Topic_CheMBL_35_78_1_eda.ipynb,
Topic_CheMBL_35_78_2_modeling.ipynb)
    models/             # Stores trained models (optional, for saving models)
    src/                # Contains Python modules (optional, for reusable code)
    requirements.txt    # Lists Python dependencies
```

5. Examples (Based on the Code)

Here are 5 specific examples of tasks you can perform with the code:

1. **Calculate Descriptors for a Specific SMILES:**

```
# English
smiles = "CC(=O)Oc1ccccc1C(=O)O"  # Aspirin
mw, logp, tpsa = calculate_descriptors(smiles)
print(f"Aspirin: MW = {mw}, LogP = {logp}, TPSA = {tpsa}")

# Vietnamese
smiles = "CC(=O)Oc1ccccc1C(=O)O"  # Aspirin
mw, logp, tpsa = calculate_descriptors(smiles)
print(f"Aspirin: MW = {mw}, LogP = {logp}, TPSA = {tpsa}")
```

2. **Filter Data by Molecular Weight:**

```
# English
filtered_df = df[(df['molecular_weight'] > 200) & (df['molecular_weight'] < 400)]
print(f"Number of compounds with MW between 200 and 400: {len(filtered_df)}")

# Vietnamese
filtered_df = df[(df['molecular_weight'] > 200) & (df['molecular_weight'] < 400)]
print(f"Số lượng hợp chất có MW từ 200 đến 400: {len(filtered_df)}")
```

3. **Calculate the Mean Activity (standard_value) for a Specific Activity Type (e.g., IC50):**

```
# English
ic50_data = df[df['standard_type'] == 'IC50']
mean_ic50 = ic50_data['standard_value'].mean()
print(f"Mean IC50: {mean_ic50}")

# Vietnamese
ic50_data = df[df['standard_type'] == 'IC50']
mean_ic50 = ic50_data['standard_value'].mean()
print(f"IC50 trung bình: {mean_ic50}")
```

4. **Find Compounds with LogP > 3.0 and TPSA < 75:**

```
# English
filtered_df = df[(df['logp'] > 3.0) & (df['tpsa'] < 75)]
print(f"Number of compounds with LogP > 3.0 and TPSA < 75: {len(filtered_df)}")

# Vietnamese
filtered_df = df[(df['logp'] > 3.0) & (df['tpsa'] < 75)]
print(f"Số lượng hợp chất có LogP > 3.0 và TPSA < 75: {len(filtered_df)}")
```

5. **Save the Processed Data to a New CSV File:**

```
# English
output_file = os.path.join(base_path, "data", "chembl_35_processed_data.csv")
df.to_csv(output_file, index=False)
print(f"Processed data saved to: {output_file}")

# Vietnamese
output_file = os.path.join(base_path, "data", "chembl_35_processed_data.csv")
df.to_csv(output_file, index=False)
print(f"Dữ liệu đã xử lý được lưu vào: {output_file}")
```

Remember to adapt these examples to your specific research questions and data analysis goals. Good luck! Let me know if you have any further questions.