PharmApp Suite

🧠 AI for Drug Discovery and Development 🧪

🧩 Topic: Target enrichment from gene sets

Okay, I understand. You're working on a Chembl 35 data analysis project using RDKit and need help structuring your code and addressing some errors you're encountering. Let's break down the analysis, provide the necessary code (SQL and Python), and address the specific issues you've highlighted.

Analysis Model Overview

The overall analysis model will follow these steps:

1. **Data Extraction (SQL):** Query the Chembl 35 database to extract relevant bioactivity data, focusing on specific activity types (e.g., IC50, Ki). We'll filter the data to extract only 100 rows to manage memory. The extracted data will include compound structures (SMILES) and corresponding activity values.
2. **Data Preprocessing (Python):**
    - Load the extracted data into a Pandas DataFrame.
    - Clean the data: Handle missing values, remove duplicates.
    - Convert SMILES strings to RDKit Mol objects.
    - Calculate molecular descriptors using RDKit (e.g., molecular weight, logP, number of hydrogen bond donors/acceptors).
3. **Exploratory Data Analysis (EDA) (Python):**
    - Explore the distribution of activity values.
    - Visualize the relationship between molecular descriptors and activity.
    - Potentially identify clusters of compounds with similar activity profiles.
4. **Model Building (Python):** (Optional, depending on your specific goal)
    - Split the data into training and testing sets.
    - Train a machine learning model to predict activity based on molecular descriptors (e.g., Linear Regression, Random Forest).
    - Evaluate the model's performance on the test set.

File Structure (AIMLops Template)

Assuming an AIMLops-like project structure, it might look something like this:

```
Topic_CheMBL_35_36/
├── data/
│   └── chembl_35_bioactivity_100.csv  # Extracted data from SQL
├── notebooks/
│   ├── Topic_CheMBL_35_36_1_data_extraction.ipynb
│   ├── Topic_CheMBL_35_36_2_data_preprocessing.ipynb
│   ├── Topic_CheMBL_35_36_3_eda.ipynb
│   └── Topic_CheMBL_35_36_4_model_building.ipynb (Optional)
├── sql/
│   └── chembl_35_bioactivity_query.sql
└── README.md
```

1. SQL Code (chembl_35_bioactivity_query.sql)

This SQL code extracts bioactivity data, focusing on IC50 values, filtering for human targets, and limiting the results to 100 rows. It also addresses your error `ERROR: operator does not exist: numeric ~ unknown, LINE 12: AND act.standard_value ~ '^[0-9\.]+$'` by using `REGEXP_MATCHES` instead of `~`.

```
-- Connect to the chembl_35 database
\c chembl_35

-- Extract bioactivity data
SELECT
    act.molregno,
    cmp.smiles,
    act.standard_type,
    act.standard_value,
    act.standard_units
FROM
    activities act
JOIN
    molecule_dictionary cmp ON act.molregno = cmp.molregno
JOIN
    target_dictionary tgt ON act.tid = tgt.tid
WHERE
    tgt.organism = 'Homo sapiens'  -- Filter for human targets
    AND act.standard_type = 'IC50'   -- Filter for IC50 values
    AND act.standard_units = 'nM'  -- Filter for nM units
    AND act.standard_value::text SIMILAR TO '[0-9\.]+'  -- Ensure standard_value is
numeric using SIMILAR TO for numeric check
LIMIT 100;

-- Save the output to a CSV file:
-- \copy (SELECT ...) TO 'path/to/your/data/chembl_35_bioactivity_100.csv' WITH CSV
HEADER;
```

Important: After running this SQL in pgAdmin, replace `'path/to/your/data/chembl_35_bioactivity_100.csv'` with the *actual* path within your `data/` folder. For example, `'./data/chembl_35_bioactivity_100.csv'` if you execute the `\copy` command from the project root and assuming `data/` is in the current directory. Make sure the user you're running pgAdmin as has write permissions to that directory.

2. Python Code (Jupyter Notebooks)

Here's the Python code, split into the notebooks as suggested.

2.1: Topic_CheMBL_35_36_1_data_extraction.ipynb

This notebook documents the SQL query used for data extraction.

```
#Topic_CheMBL_35_36_1_data_extraction.ipynb
import os

base_path = "."  # Assuming the notebook is run from the project root
data_path = os.path.join(base_path, "data")
sql_path = os.path.join(base_path, "sql")

print(f"Data path: {data_path}")
print(f"SQL path: {sql_path}")

# You would typically include the SQL query as a multiline string here for
documentation:
sql_query = """
-- Connect to the chembl_35 database
```

```
\\c chembl_35

-- Extract bioactivity data
SELECT
    act.molregno,
    cmp.smiles,
    act.standard_type,
    act.standard_value,
    act.standard_units
FROM
    activities act
JOIN
    molecule_dictionary cmp ON act.molregno = cmp.molregno
JOIN
    target_dictionary tgt ON act.tid = tgt.tid
WHERE
    tgt.organism = 'Homo sapiens'  -- Filter for human targets
    AND act.standard_type = 'IC50'   -- Filter for IC50 values
    AND act.standard_units = 'nM'  -- Filter for nM units
    AND act.standard_value::text SIMILAR TO '[0-9\\.]+'  -- Ensure standard_value is
numeric using SIMILAR TO for numeric check
LIMIT 100;

-- Save the output to a CSV file:
-- \\copy (SELECT ...) TO 'path/to/your/data/chembl_35_bioactivity_100.csv' WITH CSV
HEADER;
"""
print("\nSQL Query Used:\n", sql_query)

#This notebook is primarily for documentation.  The actual data extraction happens via
pgAdmin using the SQL query.
```

2.2: Topic_CheMBL_35_36_2_data_preprocessing.ipynb

```
# Topic_CheMBL_35_36_2_data_preprocessing.ipynb

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

base_path = "."  # Assuming the notebook is run from the project root
data_path = os.path.join(base_path, "data")
data_file = os.path.join(data_path, "chembl_35_bioactivity_100.csv")

print(f"Data file path: {data_file}")

# Load the data
try:
    df = pd.read_csv(data_file)
except FileNotFoundError:
    print(f"Error: File not found at {data_file}.  Make sure you've run the SQL query
and saved the CSV correctly.")
    exit()

print(f"Shape of the dataframe: {df.shape}")
print(df.head())
```

```
# Data Cleaning
df = df.dropna(subset=['smiles', 'standard_value']) #drop nan value
df = df.drop_duplicates(subset=['smiles', 'standard_value']) #drop duplicate value

# RDKit Conversion and descriptor calculation
def calculate_descriptors(smiles):
    mol = Chem.MolFromSmiles(smiles)
    if mol is not None:
        mw = Descriptors.MolWt(mol)
        logp = Chem.Crippen.MolLogP(mol)
        hbd = Descriptors.NumHDonors(mol)
        hba = Descriptors.NumHAcceptors(mol)
        return mw, logp, hbd, hba
    else:
        return None, None, None, None

df[['MW', 'LogP', 'HBD', 'HBA']] = df['smiles'].apply(lambda x:
pd.Series(calculate_descriptors(x)))

df = df.dropna()  # remove rows where descriptor calculation failed (invalid SMILES)

print(df.head())


# Save the processed data (optional)
processed_data_file = os.path.join(data_path,
"chembl_35_bioactivity_100_processed.csv")
df.to_csv(processed_data_file, index=False)
print(f"Processed data saved to: {processed_data_file}")
```

2.3: Topic_CheMBL_35_36_3_eda.ipynb

```
# Topic_CheMBL_35_36_3_eda.ipynb

import os
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

base_path = "."  # Assuming the notebook is run from the project root
data_path = os.path.join(base_path, "data")
processed_data_file = os.path.join(data_path,
"chembl_35_bioactivity_100_processed.csv")

print(f"Loading processed data from: {processed_data_file}")

try:
    df = pd.read_csv(processed_data_file)
except FileNotFoundError:
    print(f"Error: File not found at {processed_data_file}.  Make sure you've run the
preprocessing notebook.")
    exit()

# EDA
print(df.describe())

# Distribution of IC50 values (log scale)
plt.figure(figsize=(8, 6))
sns.histplot(np.log10(df['standard_value']), kde=True)
plt.title('Distribution of log10(IC50) values')
```

```
plt.xlabel('log10(IC50) (nM)')
plt.ylabel('Frequency')
plt.show()

# Scatter plots of descriptors vs. IC50
plt.figure(figsize=(12, 8))
plt.subplot(2, 2, 1)
sns.scatterplot(x='MW', y=np.log10(df['standard_value']), data=df)
plt.title('MW vs. log10(IC50)')

plt.subplot(2, 2, 2)
sns.scatterplot(x='LogP', y=np.log10(df['standard_value']), data=df)
plt.title('LogP vs. log10(IC50)')

plt.subplot(2, 2, 3)
sns.scatterplot(x='HBD', y=np.log10(df['standard_value']), data=df)
plt.title('HBD vs. log10(IC50)')

plt.subplot(2, 2, 4)
sns.scatterplot(x='HBA', y=np.log10(df['standard_value']), data=df)
plt.title('HBA vs. log10(IC50)')

plt.tight_layout()
plt.show()
```

2.4: Topic_CheMBL_35_36_4_model_building.ipynb (Optional)

This is an *optional* notebook if you want to build a predictive model. It provides a basic example using Linear Regression.

```
# Topic_CheMBL_35_36_4_model_building.ipynb

import os
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
from sklearn.preprocessing import StandardScaler

base_path = "."  # Assuming the notebook is run from the project root
data_path = os.path.join(base_path, "data")
processed_data_file = os.path.join(data_path,
"chembl_35_bioactivity_100_processed.csv")

print(f"Loading processed data from: {processed_data_file}")

try:
    df = pd.read_csv(processed_data_file)
except FileNotFoundError:
    print(f"Error: File not found at {processed_data_file}.  Make sure you've run the
preprocessing notebook.")
    exit()

# Data Preparation
X = df[['MW', 'LogP', 'HBD', 'HBA']]
y = np.log10(df['standard_value'])  # Use log-transformed IC50

# Data Scaling
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

```
# Train-Test Split
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2,
random_state=42)

# Model Training
model = LinearRegression()
model.fit(X_train, y_train)

# Model Evaluation
y_pred = model.predict(X_test)

# Addressing the scikit-learn version issue:  The squared=False parameter was
introduced in later versions of scikit-learn.
# If you have an older version, you can calculate the RMSE manually:
mse = mean_squared_error(y_test, y_pred)
rmse = mse**0.5  # Calculate RMSE manually

# If you have a newer version of scikit-learn (>= 0.22), you can use squared=False:
# rmse = mean_squared_error(y_test, y_pred, squared=False)

r2 = r2_score(y_test, y_pred)

print(f"Root Mean Squared Error (RMSE): {rmse}")
print(f"R-squared: {r2}")

# Feature Importance (for Linear Regression)
print("\nFeature Importance:")
for i, feature in enumerate(['MW', 'LogP', 'HBD', 'HBA']):
    print(f"{feature}: {model.coef_[i]}")
```

3. Addressing the Errors

- **Error a: `ERROR: operator does not exist: numeric ~ unknown, LINE 12: AND act.standard_value ~ '^[0-9\.]+$'`**

  This error occurs because the `~` operator (POSIX regular expression match) might not work directly with numeric columns in some PostgreSQL configurations. Also, using regex on numeric columns is usually inefficient. The fix is to:

  1. **Cast the `standard_value` to text**: `act.standard_value::text`
  2. **Use `SIMILAR TO` instead of `~`**: `act.standard_value::text SIMILAR TO '[0-9\.]+'` `SIMILAR TO` provides a simpler pattern matching syntax for checking numeric values. I've updated the SQL code above.
  3. **Type Safety**: The best solution (if possible) is to ensure that the `standard_value` column is actually a numeric type in the database (e.g., `NUMERIC` or `DOUBLE PRECISION`). If it's stored as text, strongly consider changing the column type. You can then remove the pattern matching entirely.

- **Error b: `old scikit-learn version does not support parameters squared=False in the mean_squared_error function`**

  This indicates that you're using an older version of scikit-learn (`< 0.22`). The `squared` parameter was added in scikit-learn version 0.22. The fix is:

  1. **Upgrade scikit-learn:** The recommended approach is to upgrade to a more recent version: `pip install -U scikit-learn`

  2. **Calculate RMSE manually:** If you cannot upgrade scikit-learn, calculate the RMSE (Root Mean Squared Error) manually:

```
mse = mean_squared_error(y_test, y_pred)
rmse = mse**0.5
```

I've included this manual calculation in the `Topic_CheMBL_35_36_4_model_building.ipynb` notebook.

4. Example Execution

1. **SQL (pgAdmin):**
    - Connect to your Chembl 35 database (192.168.206.136, rd/rd, chembl_35).
    - Open a query window.
    - Paste the SQL code from `chembl_35_bioactivity_query.sql`.
    - **Modify the `\copy` path** to the correct location on your server's filesystem (e.g., `'/home/rd/Topic_CheMBL_35_36/data/chembl_35_bioactivity_100.csv'`). Double-check permissions!
    - Execute the query. This will create the CSV file.
2. **Jupyter Notebooks:**
    - Navigate to the `notebooks/` directory in your project.
    - Open each notebook (`Topic_CheMBL_35_36_1_data_extraction.ipynb`, `Topic_CheMBL_35_36_2_data_preprocessing.ipynb`, etc.) in order.
    - Run each cell in the notebook sequentially. Make sure the paths defined in the notebooks are correct relative to where you launch the Jupyter notebook.

5. Examples

Here are 5 examples to illustrate how to use the code.

1. **Basic Data Extraction:** Execute the SQL query in pgAdmin *exactly* as provided, ensuring the `\copy` path points to your `data/` folder. Then, open `Topic_CheMBL_35_36_1_data_extraction.ipynb` and verify the SQL query documented there matches what you executed.

2. **Data Preprocessing and Descriptor Calculation:** After extracting the data (example 1), run `Topic_CheMBL_35_36_2_data_preprocessing.ipynb`. Check the output of the `print(df.head())` statements to see the calculated molecular descriptors (MW, LogP, HBD, HBA).

3. **Exploratory Data Analysis (EDA):** After preprocessing (example 2), run `Topic_CheMBL_35_36_3_eda.ipynb`. Observe the generated histograms and scatter plots to understand the data's distribution and relationships between descriptors and IC50.

4. **Model Building (Linear Regression):** After preprocessing (example 2), run `Topic_CheMBL_35_36_4_model_building.ipynb`. Note the RMSE and R-squared values. Experiment with different `random_state` values in `train_test_split` to see how it affects the model's performance.

5. **Troubleshooting File Paths:** If you get a `FileNotFoundError` in any of the notebooks, double-check the `base_path`, `data_path`, and file paths. Ensure that the CSV files exist in the expected locations and that the notebook is being run from the correct directory (project root). The `print(f"Data file path: {data_file}")` statements are your friends here!

Important Considerations:

- **CheMBL License:** Remember that the CheMBL database has licensing terms. Make sure you adhere to them.

- **Data Size:** Working with the full CheMBL database requires significant computational resources. The 100-row limit is suitable for testing, but you'll need more powerful hardware and optimized code for larger-scale analyses.
- **More Sophisticated Models:** Linear Regression is a simple starting point. Consider more advanced machine learning algorithms (e.g., Random Forest, Support Vector Machines, Neural Networks) for improved prediction accuracy.
- **Feature Engineering:** Experiment with different molecular descriptors beyond the basic ones provided (MW, LogP, HBD, HBA). RDKit provides a vast library of descriptors. Feature selection techniques can also help improve model performance.
- **Units:** Ensure that all activity values are in consistent units (e.g., nM). You might need to perform unit conversions if the data contains mixed units.

In Summary (Türkçe):

Bu proje, Chembl 35 veritabanından ilaç geliştirme verilerini analiz etmeyi amaçlamaktadır. Proje, RDKit kütüphanesini kullanarak moleküler tanımlayıcılar hesaplayacak ve makine öğrenimi modelleri oluşturacaktır.

1. **Veri Çekme (SQL):** Chembl 35 veritabanından gerekli bioaktivite verilerini (IC50 değerleri gibi) çekmek için SQL sorgusu kullanılacaktır. İnsan hedeflerine odaklanılacak ve veri boyutu sınırlı tutulacaktır (100 satır). Bu, sunucunuzun aşırı yüklenmesini önleyecektir.
2. **Veri Ön İşleme (Python):** Çekilen veriler Pandas DataFrame'e yüklenecek, temizlenecek (eksik değerler ve tekrarlar giderilecek) ve SMILES stringleri RDKit Mol objelerine dönüştürülecektir. Ardından, RDKit kullanarak moleküler ağırlık, logP gibi moleküler tanımlayıcılar hesaplanacaktır.
3. **Keşifsel Veri Analizi (EDA) (Python):** Aktivite değerlerinin dağılımı incelenecek, moleküler tanımlayıcılar ile aktivite arasındaki ilişkiler görselleştirilecektir.
4. **Model Oluşturma (Python):** Veriler eğitim ve test kümelerine ayrılacak ve aktiviteyi tahmin etmek için bir makine öğrenimi modeli eğitilecektir (örneğin, Lineer Regresyon). Modelin performansı test kümesinde değerlendirilecektir.
5. **Hataları Giderme:** SQL sorgusundaki ~ operatörü hatası, `SIMILAR TO` ile çözülmüştür. Scikit-learn sürümünüzün eski olması durumunda, RMSE değeri manuel olarak hesaplanmıştır.

Bu adımları takip ederek, Chembl 35 verilerini analiz edebilir ve ilaç geliştirme süreçlerinize katkıda bulunabilirsiniz. Projenizi başlatırken ve ilerletirken, hata mesajlarını dikkatlice okuyun ve her adımda çıktıları kontrol edin. Başarılar!