PharmApp Suite

🧠 AI for Drug Discovery and Development 💉

🧩 Topic: Orphan drug discovery via ChEMBL

Okay, I understand. You're working on a ChEMBL 35 data analysis project using RDKit, and you want to follow an AIMLops-inspired structure with Jupyter notebooks. You're using a PostgreSQL database hosted on 192.168.206.136 and need help with SQL queries and Python code to extract and analyze the data, keeping in mind the errors you've encountered. You specifically want to explore "Topic_CheMBL_35_59" and need 5 example analyses after the initial setup.

Here's a breakdown of how we can approach this, including SQL and Python code snippets, error handling, and considerations for your AIMLops structure:

1. Project Structure and AIMLops Adherence:

Following the AIMLops template:

```
project_root/
├── data/           # Stores the extracted CSV data from ChEMBL
├── notebooks/      # Jupyter notebooks for analysis and experimentation
│   ├── Topic_CheMBL_35_59_1_Data_Extraction.ipynb
│   ├── Topic_CheMBL_35_59_2_Analysis.ipynb
├── src/            # (Optional) Python modules for reusable functions (e.g., data
cleaning, model building)
├── models/         # (Optional) Saved machine learning models
├── README.md       # Project description, instructions, and dependencies
```

- **`data/`**: This will hold the CSV files extracted from your ChEMBL database using SQL queries.
- **`notebooks/`**: This directory contains your Jupyter notebooks. `Topic_CheMBL_35_59_1_Data_Extraction.ipynb` will focus on data retrieval, and `Topic_CheMBL_35_59_2_Analysis.ipynb` will contain the analysis.
- **`src/`**: If you have reusable Python code, put it here in modules. This helps keep your notebooks clean.
- **`models/`**: If you build and train any machine learning models, save them here.

2. Addressing the Errors:

- **SQL Error: `ERROR: operator does not exist: numeric ~ unknown, LINE 12: AND act.standard_value ~ '^[0-9\.]+$'`**: This error occurs because the `~` operator is used for regular expression matching with text data. `standard_value` is likely a numeric column (e.g., `INTEGER`, `REAL`, `NUMERIC`). You should use numeric comparisons instead. If the column is sometimes a string, you may need to cast it to a numeric type first or filter out non-numeric values in a safer way.
- **Scikit-learn Error: `old scikit-learn version does not support parameters squared=False in the mean_squared_error function`**: This indicates you are using an older version of scikit-learn. You have two options:
  - **Upgrade scikit-learn:** This is the preferred solution. Run `pip install scikit-learn --upgrade` in your terminal.
  - **Remove `squared=False`:** If you *cannot* upgrade, you can remove the `squared=False` argument. This will return the Mean Squared Error (MSE) instead of the Root Mean Squared Error (RMSE). Remember to adjust your interpretation accordingly.

3. SQL Code (for `Topic_CheMBL_35_59_1_Data_Extraction.ipynb`):

This SQL query retrieves data from the ChEMBL database, focusing on bioactivities and molecules. It's designed to be limited to 100 rows for initial testing.

```sql
-- data/chembl_35_59_bioactivities.csv

SELECT
    cmp.chembl_id AS molecule_chembl_id,
    cmp.pref_name AS molecule_name,
    act.standard_type,
    act.standard_value,
    act.standard_units,
    act.activity_comment,
    targ.target_type,
    targ.pref_name AS target_name,
    act.pchembl_value,
    mol.structure AS smiles
FROM
    activities act
JOIN
    molecule_dictionary cmp ON act.molregno = cmp.molregno
JOIN
    target_dictionary targ ON act.tid = targ.tid
JOIN
    compound_structures mol ON cmp.molregno = mol.molregno

WHERE act.standard_type IN ('IC50', 'Ki', 'EC50')  -- Common activity types
    AND act.standard_units = 'nM'                   -- Focus on nM values
    AND act.standard_value IS NOT NULL
    AND act.standard_value > 0                      -- Exclude zero or negative values
    AND act.pchembl_value IS NOT NULL   -- Require pChEMBL value
    AND targ.target_type = 'SINGLE PROTEIN'         -- Filter for single protein
targets
ORDER BY act.pchembl_value DESC
LIMIT 100;
```

Explanation:

- **`SELECT ... FROM ... JOIN ...`**: This structure joins the necessary tables (activities, molecule_dictionary, target_dictionary, compound_structures) to retrieve molecule information, activity data, target information, and SMILES strings.
- **`WHERE ...`**: This clause filters the data:
    - `act.standard_type IN ('IC50', 'Ki', 'EC50')`: Selects only common and well-defined activity types. Adjust this if you have a specific activity type related to Topic 59.
    - `act.standard_units = 'nM'`: Ensures that the activity values are in nanomolar units for consistency.
    - `act.standard_value IS NOT NULL AND act.standard_value > 0`: Excludes missing or non-positive activity values.
    - `act.pchembl_value IS NOT NULL`: Only includes entries where the pChEMBL value is available (a more processed and comparable activity measure).
    - `targ.target_type = 'SINGLE PROTEIN'`: Filters results to a target that is a single protein.
- **`ORDER BY act.pchembl_value DESC`**: Orders results by potency (pChEMBL value) in descending order, putting the most potent compounds at the top.
- **`LIMIT 100`**: Limits the result set to the first 100 rows, as requested.

Saving the Data (using `psql` from the command line):

```
psql -h 192.168.206.136 -U rd -d chembl_35 -c "\copy (SELECT cmp.chembl_id AS
molecule_chembl_id, cmp.pref_name AS molecule_name, act.standard_type,
act.standard_value, act.standard_units, act.activity_comment, targ.target_type,
targ.pref_name AS target_name, act.pchembl_value, mol.structure AS smiles FROM
activities act JOIN molecule_dictionary cmp ON act.molregno = cmp.molregno JOIN
target_dictionary targ ON act.tid = targ.tid JOIN compound_structures mol ON
cmp.molregno = mol.molregno WHERE act.standard_type IN ('IC50', 'Ki', 'EC50') AND
act.standard_units = 'nM' AND act.standard_value IS NOT NULL AND act.standard_value >
0  AND act.pchembl_value IS NOT NULL AND targ.target_type = 'SINGLE PROTEIN' ORDER BY
act.pchembl_value DESC LIMIT 100) TO 'data/chembl_35_59_bioactivities.csv' DELIMITER
',' CSV HEADER;"
```

- Replace `-h`, `-U`, and `-d` with your correct server, username, and database information.

4. Python Code (for `Topic_CheMBL_35_59_2_Analysis.ipynb`):

```python
import os
import pandas as pd
from rdkit import Chem
from rdkit.Chem import Descriptors
from rdkit.Chem import Lipinski
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
import numpy as np  # needed for safe log calculations

# Base path for the project (adapt to your actual project root)
base_path = "."  # Assuming the notebook is in the 'notebooks' directory

# Load the data
data_path = os.path.join(base_path, "data", "chembl_35_59_bioactivities.csv")
try:
    df = pd.read_csv(data_path)
    print("Data loaded successfully.")
except FileNotFoundError:
    print(f"Error: File not found at {data_path}")
    df = None # Handle the error appropriately

if df is not None: # Only proceed if the DataFrame was loaded successfully

    # Data Cleaning and Preprocessing
    df = df.dropna(subset=['smiles', 'pchembl_value'])  # Handle missing SMILES or
pChEMBL values

    # Function to calculate molecular weight
    def calculate_mw(smiles):
        mol = Chem.MolFromSmiles(smiles)
        if mol:
            return Descriptors.MolWt(mol)
        else:
            return None

    # Function to calculate LogP
    def calculate_logp(smiles):
        mol = Chem.MolFromSmiles(smiles)
        if mol:
            return Descriptors.MolLogP(mol)
        else:
```

```
            return None

    # Apply the functions to create new columns
    df['mol_weight'] = df['smiles'].apply(calculate_mw)
    df['log_p'] = df['smiles'].apply(calculate_logp)

    # Basic statistics and visualization (example)
    print(df.describe())

    # Plotting pChEMBL vs. Molecular Weight
    plt.figure(figsize=(8, 6))
    sns.scatterplot(x='mol_weight', y='pchembl_value', data=df)
    plt.xlabel('Molecular Weight')
    plt.ylabel('pChEMBL Value')
    plt.title('pChEMBL Value vs. Molecular Weight')
    plt.show()

    # Linear Regression Example (predicting pChEMBL from molecular weight)
    X = df[['mol_weight']].fillna(df['mol_weight'].mean())  # Handle potential NaN
values
    y = df['pchembl_value']

    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

    model = LinearRegression()
    model.fit(X_train, y_train)

    y_pred = model.predict(X_test)

    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}")

    # Visualize the regression line
    plt.figure(figsize=(8, 6))
    plt.scatter(X_test, y_test, color='blue', label='Actual')
    plt.plot(X_test, y_pred, color='red', linewidth=2, label='Predicted')
    plt.xlabel('Molecular Weight')
    plt.ylabel('pChEMBL Value')
    plt.title('Linear Regression: pChEMBL vs. Molecular Weight')
    plt.legend()
    plt.show()
```

Explanation:

1. **Import Libraries**: Imports necessary libraries for data manipulation, cheminformatics (RDKit), visualization, and machine learning (scikit-learn).
2. **Define Base Path**: Sets the base path for accessing files.
3. **Load Data**: Loads the CSV file into a Pandas DataFrame. Includes error handling for the case where the file is not found.
4. **Data Cleaning**:
    - Removes rows with missing SMILES strings or pChEMBL values. This ensures that the RDKit functions and the regression model will work correctly.
5. **Feature Calculation (RDKit)**:

- `calculate_mw(smiles)`: Calculates the molecular weight of a molecule from its SMILES string using RDKit. Handles cases where the SMILES string is invalid.
- `calculate_logp(smiles)`: Calculates the LogP (octanol-water partition coefficient) of a molecule from its SMILES string using RDKit. Handles cases where the SMILES string is invalid.

6. **Visualization**:
   - Creates a scatter plot of pChEMBL value versus molecular weight using Seaborn.
7. **Linear Regression**:
   - Prepares the data for linear regression. `X` is the molecular weight (independent variable), and `y` is the pChEMBL value (dependent variable). Handles any remaining NaN values by filling them with the mean.
   - Splits the data into training and testing sets.
   - Creates a `LinearRegression` model, trains it on the training data, and makes predictions on the test data.
   - Calculates the Mean Squared Error (MSE) and R-squared (R2) to evaluate the model's performance.
   - Visualizes the regression line along with the actual data points.

5. Five Example Analyses (Expand on these in `Topic_CheMBL_35_59_2_Analysis.ipynb`):

Here are five analysis ideas you can implement, building on the basic code:

1. **Lipinski's Rule of Five Analysis:**
   - Calculate Lipinski's Rule of Five properties (molecular weight, LogP, number of hydrogen bond donors, number of hydrogen bond acceptors) using RDKit.
   - Create a new column indicating whether a compound violates any of Lipinski's rules.
   - Analyze the distribution of compounds that violate the rules and see if there's a correlation with activity (pChEMBL value).

```
def lipinski_rule(smiles):
    mol = Chem.MolFromSmiles(smiles)
    if mol:
        mw = Descriptors.MolWt(mol)
        logp = Descriptors.MolLogP(mol)
        hbd = Lipinski.NumHDonors(mol)
        hba = Lipinski.NumHAcceptors(mol)
        violations = 0
        if mw > 500:
            violations += 1
        if logp > 5:
            violations += 1
        if hbd > 5:
            violations += 1
        if hba > 10:
            violations += 1
        return violations
    else:
        return None

df['lipinski_violations'] = df['smiles'].apply(lipinski_rule)

# Analysis
print(df['lipinski_violations'].value_counts())
sns.boxplot(x='lipinski_violations', y='pchembl_value', data=df)
plt.show()
```

2. **Activity Distribution by Target:**

- Group the data by `target_name`.
- Calculate the mean and standard deviation of pChEMBL values for each target.
- Visualize the distribution of activities for the top N targets (e.g., top 10).

```
target_activity = df.groupby('target_name')['pchembl_value'].agg(['mean', 'std',
'count'])
target_activity = target_activity.sort_values(by='mean', ascending=False)
print(target_activity.head(10)) # Top 10 targets by average pChEMBL

# Example visualization
top_targets = target_activity.head(5).index  # Select top 5 targets
sns.boxplot(x='target_name', y='pchembl_value',
data=df[df['target_name'].isin(top_targets)])
plt.xticks(rotation=45, ha='right')
plt.show()
```

3. **Correlation Analysis:**
   - Calculate the correlation matrix between molecular properties (molecular weight, LogP) and activity (pChEMBL value).
   - Visualize the correlation matrix using a heatmap.

```
correlation_matrix = df[['pchembl_value', 'mol_weight', 'log_p']].corr()
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm')
plt.show()
```

4. **pChEMBL Value Prediction with Multiple Features (Multiple Linear Regression):**
   - Extend the linear regression model to use both molecular weight and LogP as predictors.
   - Evaluate the model's performance using MSE and R-squared.

```
X = df[['mol_weight', 'log_p']].fillna(df[['mol_weight', 'log_p']].mean())
y = df['pchembl_value']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

model = LinearRegression()
model.fit(X_train, y_train)

y_pred = model.predict(X_test)

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}")
```

5. **Activity Cliff Analysis:**
   - This requires more complex calculations. The idea is to find pairs of molecules that are structurally very similar but have significantly different activities. You'd need to:
     - Calculate molecular similarity using RDKit (e.g., Tanimoto similarity based on Morgan fingerprints).
     - Define a threshold for similarity and a threshold for activity difference.
     - Identify molecule pairs that meet both criteria. This helps identify structural features that are critical for activity.

```
from rdkit.Chem import AllChem
from rdkit import DataStructs

def calculate_morgan_fingerprint(smiles, radius=2):
```

```
        mol = Chem.MolFromSmiles(smiles)
        if mol:
            fp = AllChem.GetMorganFingerprint(mol, radius)
            return fp
        else:
            return None

    df['morgan_fp'] = df['smiles'].apply(calculate_morgan_fingerprint)
    df = df.dropna(subset=['morgan_fp'])  # Remove molecules with invalid
    fingerprints

    # Now you need to compare all pairs of molecules.  This is computationally
    expensive
    # For a smaller dataset, you can iterate through all pairs.  For larger
    datasets, consider using a more efficient data structure like a BallTree.

    # This is a very simplified example for illustration only
    similarity_threshold = 0.8
    activity_difference_threshold = 2.0  # pChEMBL units

    for i in range(len(df)):
        for j in range(i + 1, len(df)):  # Avoid comparing a molecule to itself and
    avoid duplicate comparisons
            fp1 = df['morgan_fp'].iloc[i]
            fp2 = df['morgan_fp'].iloc[j]
            similarity = DataStructs.TanimotoSimilarity(fp1, fp2)
            activity_difference = abs(df['pchembl_value'].iloc[i] -
    df['pchembl_value'].iloc[j])

            if similarity > similarity_threshold and activity_difference >
    activity_difference_threshold:
                print(f"Activity Cliff Found: Molecule 1:
    {df['molecule_chembl_id'].iloc[i]}, pChEMBL: {df['pchembl_value'].iloc[i]},
    Molecule 2: {df['molecule_chembl_id'].iloc[j]}, pChEMBL:
    {df['pchembl_value'].iloc[j]}, Similarity: {similarity}")
                # Further analysis:  Examine the structural differences between
    these molecules
```

6. Languages:

English:

This approach provides a solid foundation for your ChEMBL 35 data analysis project. Remember to adapt the SQL queries and Python code to your specific research questions and the characteristics of "Topic_CheMBL_35_59". Good luck!

Chinese (Simplified):

这个方法为你的 ChEMBL 35 数据分析项目提供了一个坚实的基础。请记住根据你具体的研究问题和 "Topic_CheMBL_35_59" 的特点来调整 SQL 查询和 Python 代码。祝你好运!