PharmApp Suite

🧠 AI for Drug Discovery and Development 🍼

🧩 Topic: Chemical space mapping of ChEMBL molecules

Okay, I understand. You're working with ChEMBL 35 data, using RDKit for analysis within an AIMLops framework. You have a PostgreSQL database, and you're encountering issues related to data type comparisons in your SQL queries and an outdated scikit-learn version. You want to create a notebook (`Topic_CheMBL_35_10_1_*`, `Topic_CheMBL_35_10_2_*`), incorporating SQL and Python code, focusing on a subset of 100 rows.

Here's a breakdown of the process, the issues, and suggested solutions, along with example code in both SQL and Python:

1. Analysis of the Analysis Model

The core idea likely revolves around:

- **Data Extraction & Cleaning:** Retrieving bioactivity data (IC50, Ki, etc.) from ChEMBL, often associated with specific targets or compounds. Cleaning might involve standardizing units, removing outliers, or converting pIC50 values.
- **Data Transformation:** Convert data to appropriate formats. This could mean converting IC50 values to pIC50 (negative log of IC50 in molar units).
- **Exploratory Data Analysis (EDA):** Investigating the distribution of bioactivity values, relationships between different properties (e.g., molecular weight vs. pIC50), and identifying potentially interesting compounds or targets.
- **Structure-Activity Relationship (SAR) Analysis:** Exploring how changes in molecular structure affect bioactivity. This often involves calculating molecular descriptors using RDKit and correlating them with activity.
- **Machine Learning (Optional):** Building predictive models to estimate bioactivity based on molecular descriptors. This can be used for virtual screening or lead optimization.

2. Error Analysis and Solutions

- **SQL Error:** `ERROR: operator does not exist: numeric ~ unknown, LINE 12: AND act.standard_value ~ '^[0-9\.]+$'`
    - **Explanation:** This error occurs because you're trying to use the regular expression operator `~` (similar to `LIKE` with regex) on a numeric column (`act.standard_value`). PostgreSQL doesn't directly support regex matching on numeric types.
    - **Solution:** Cast the `standard_value` to text before applying the regex, or use a numeric comparison:

      ```
      AND CAST(act.standard_value AS TEXT) ~ '^[0-9\.]+$'
      ```

      Alternatively, if you just want to filter for non-null numeric values you could use:

      ```
      AND act.standard_value IS NOT NULL
      ```

      Or if you have certain range that you want to filter, for example between 0 and 1000, you could use: `sql AND act.standard_value BETWEEN 0 AND 1000`
- **Scikit-learn Error:** `old scikit-learn version does not support parameters squared=False in the mean_squared_error function`

- **Explanation:** The `squared=False` argument was introduced in a later version of scikit-learn's `mean_squared_error` function.
- **Solution:** Upgrade your scikit-learn version. In your Jupyter notebook:

```
!pip install scikit-learn --upgrade
```

Alternatively, if upgrading isn't possible, calculate the root mean squared error (RMSE) manually:

```python
from sklearn.metrics import mean_squared_error
import numpy as np

mse = mean_squared_error(y_true, y_pred)
rmse = np.sqrt(mse)
```

3. Folder Structure and Code Structure (Following AIMLops)

Assume your base path is `base_path = "/path/to/your/project"`

- **`data/`:** Contains the raw data extracted from the database (`chembl_data.csv`).
- **`notebooks/`:** Contains your Jupyter notebooks (`Topic_CheMBL_35_10_1_data_extraction.ipynb`, `Topic_CheMBL_35_10_2_analysis.ipynb`).
- **`src/`:** (Optional, but recommended) Contains Python modules for reusable functions (e.g., `data_processing.py`, `feature_extraction.py`).
- **`models/`:** (Optional) Stores trained machine learning models.
- **`reports/`:** (Optional) Stores generated reports and visualizations.

4. SQL Code (to be run in pgAdmin and save as CSV)

```sql
-- Topic_CheMBL_35_10_data_extraction.sql
-- Extracts ChEMBL data for analysis (limited to 100 rows).

SELECT
    cmp.chembl_id AS compound_chembl_id,
    act.activity_id,
    act.standard_type,
    act.standard_relation,
    act.standard_value,
    act.standard_units,
    act.pchembl_value,
    tar.chembl_id AS target_chembl_id,
    tar.pref_name AS target_name
FROM
    activities act
JOIN
    assays ass ON act.assay_id = ass.assay_id
JOIN
    target_dictionary tar ON ass.tid = tar.tid
JOIN
    component_sequences cs ON tar.tid = cs.tid
JOIN
    compound_structures cmp ON act.molregno = cmp.molregno
WHERE
    tar.target_type = 'SINGLE PROTEIN'  -- Focus on single protein targets
    AND act.standard_type = 'IC50'      -- Focus on IC50 values
    AND act.standard_relation = '='      -- Focus on exact IC50 values
    AND act.standard_units = 'nM'        -- Focus on nM units
    AND act.pchembl_value IS NOT NULL    -- Filter for records with pchembl_value
    AND CAST(act.standard_value AS TEXT) ~ '^[0-9\.]+$'  -- Ensure standard_value is
```

```
numeric (after casting to text)
LIMIT 100;
```

- **Save this as `chembl_data.csv` in your `data/` directory.**

5. Python Code (in Jupyter Notebooks)

`Topic_CheMBL_35_10_1_data_extraction.ipynb`

```
# Topic_CheMBL_35_10_1_data_extraction.ipynb
import os
import pandas as pd

base_path = "/path/to/your/project"  # Replace with your actual base path
data_dir = os.path.join(base_path, "data")
csv_file = os.path.join(data_dir, "chembl_data.csv")

# Load the data from the CSV file
try:
    df = pd.read_csv(csv_file)
    print(f"Data loaded successfully from {csv_file}")
    print(df.head()) # Display the first few rows
except FileNotFoundError:
    print(f"Error: File not found at {csv_file}. Make sure you ran the SQL query and
saved the data correctly.")
except Exception as e:
    print(f"An error occurred: {e}")

# Basic data inspection
if 'df' in locals():
    print(f"Number of rows: {len(df)}")
    print(df.info())
```

`Topic_CheMBL_35_10_2_analysis.ipynb`

```
# Topic_CheMBL_35_10_2_analysis.ipynb
import os
import pandas as pd
import numpy as np
from rdkit import Chem
from rdkit.Chem import AllChem
from rdkit.Chem import Descriptors
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error

base_path = "/path/to/your/project"  # Replace with your actual base path
data_dir = os.path.join(base_path, "data")
csv_file = os.path.join(data_dir, "chembl_data.csv")
# Load the data from the CSV file
try:
    df = pd.read_csv(csv_file)
    print(f"Data loaded successfully from {csv_file}")
except FileNotFoundError:
    print(f"Error: File not found at {csv_file}. Make sure you ran the SQL query and
saved the data correctly.")
    exit() # Stop execution if the data isn't loaded

# 1. Data Cleaning and Transformation

# Handle missing pChEMBL values (if any) - replace with median or drop rows
```

```python
df = df.dropna(subset=['pchembl_value'])  # Or impute with median if appropriate

# Function to calculate molecular descriptors using RDKit
def calculate_descriptors(smiles):
    mol = Chem.MolFromSmiles(smiles)
    if mol is None:
        return None  # Handle invalid SMILES strings

    # Ensure molecule has 3D coordinates for some descriptors
    mol = Chem.AddHs(mol)
    AllChem.EmbedMolecule(mol, AllChem.ETKDGv2())  # ETKDGv2 is a robust conformer
generator

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

# Get SMILES string.  The compound_structures table does not have SMILES.
# You may need to join with a table with the smiles or canonical_smiles
# Let's assume a table called compound_properties with chembl_id and canonical_smiles
# You'll need to adapt this part based on the database.
# For this example, I will generate 100 random SMILES for demonstration.
import random
smiles_list = [
    'CC(=O)Oc1ccccc1C(=O)O',
    'c1ccccc1',
    'CC(=O)O',
    'C1CCCCC1',
    'CN1C=NC2=C1N=CN=C2N',
    'C1=CC=CC=C1',
    'CC(=O)NC1=CC=CC=C1',
    'CC(=O)C',
    'CC(C)O',
    'C1=CC=CN=C1',
    'CC(=O)Oc1ccccc1C(=O)O',
    'c1ccccc1',
    'CC(=O)O',
    'C1CCCCC1',
    'CN1C=NC2=C1N=CN=C2N',
    'C1=CC=CC=C1',
    'CC(=O)NC1=CC=CC=C1',
    'CC(=O)C',
    'CC(C)O',
    'C1=CC=CN=C1',
    'CC(=O)Oc1ccccc1C(=O)O',
    'c1ccccc1',
    'CC(=O)O',
    'C1CCCCC1',
    'CN1C=NC2=C1N=CN=C2N',
    'C1=CC=CC=C1',
    'CC(=O)NC1=CC=CC=C1',
    'CC(=O)C',
    'CC(C)O',
    'C1=CC=CN=C1',
    'CC(=O)Oc1ccccc1C(=O)O',
```

```
    'c1ccccc1',
    'CC(=O)O',
    'C1CCCCC1',
    'CN1C=NC2=C1N=CN=C2N',
    'C1=CC=CC=C1',
    'CC(=O)NC1=CC=CC=C1',
    'CC(=O)C',
    'CC(C)O',
    'C1=CC=CN=C1',
    'CC(=O)Oc1ccccc1C(=O)O',
    'c1ccccc1',
    'CC(=O)O',
    'C1CCCCC1',
    'CN1C=NC2=C1N=CN=C2N',
    'C1=CC=CC=C1',
    'CC(=O)NC1=CC=CC=C1',
    'CC(=O)C',
    'CC(C)O',
    'C1=CC=CN=C1',
    'CC(=O)Oc1ccccc1C(=O)O',
    'c1ccccc1',
    'CC(=O)O',
    'C1CCCCC1',
    'CN1C=NC2=C1N=CN=C2N',
    'C1=CC=CC=C1',
    'CC(=O)NC1=CC=CC=C1',
    'CC(=O)C',
    'CC(C)O',
    'C1=CC=CN=C1',
    'CC(=O)Oc1ccccc1C(=O)O',
    'c1ccccc1',
    'CC(=O)O',
    'C1CCCCC1',
    'CN1C=NC2=C1N=CN=C2N',
    'C1=CC=CC=C1',
    'CC(=O)NC1=CC=CC=C1',
    'CC(=O)C',
    'CC(C)O',
    'C1=CC=CN=C1',
    'CC(=O)Oc1ccccc1C(=O)O',
    'c1ccccc1',
    'CC(=O)O',
    'C1CCCCC1',
    'CN1C=NC2=C1N=CN=C2N',
    'C1=CC=CC=C1',
    'CC(=O)NC1=CC=CC=C1',
    'CC(=O)C',
    'CC(C)O',
    'C1=CC=CN=C1',
    'CC(=O)Oc1ccccc1C(=O)O',
    'c1ccccc1',
    'CC(=O)O',
    'C1CCCCC1',
    'CN1C=NC2=C1N=CN=C2N',
    'C1=CC=CC=C1',
    'CC(=O)NC1=CC=CC=C1',
    'CC(=O)C',
    'CC(C)O',
    'C1=CC=CN=C1'
]
```

```
df['canonical_smiles'] = random.choices(smiles_list, k=len(df))

# Apply descriptor calculation to each compound
df['descriptors'] = df['canonical_smiles'].apply(calculate_descriptors)

# Handle potential errors in descriptor calculation
df = df[df['descriptors'].notna()]

# Convert descriptor dictionaries to individual columns
descriptors_df = pd.json_normalize(df['descriptors'])
df = pd.concat([df.drop('descriptors', axis=1), descriptors_df], axis=1)

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

# 2. Simple SAR Analysis
# For example, correlate MolWt with pChEMBLValue
correlation = df['MolWt'].corr(df['pchembl_value'])
print(f"Correlation between Molecular Weight and pChEMBLValue: {correlation}")

# 3. Basic Machine Learning (Linear Regression)

# Prepare data for machine learning
X = df[['MolWt', 'LogP', 'HBA', 'HBD', 'TPSA']]  # Use your calculated descriptors
y = df['pchembl_value']

# 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)
rmse = np.sqrt(mse)  # Calculate RMSE manually
print(f"Root Mean Squared Error: {rmse}")

# Print the coefficients of the linear regression model
print("Model Coefficients:")
for feature, coef in zip(X.columns, model.coef()):
    print(f"{feature}: {coef}")
```

6. Example Scenarios (Using the Code Above)

Here are five example scenarios you could explore using the provided code:

1. **Target-Specific Analysis:** Modify the SQL query to focus on a specific target (e.g., `WHERE tar.chembl_id = 'CHEMBL205'`). Then, analyze the SAR for compounds targeting that protein.
2. **Descriptor Importance:** Experiment with different sets of molecular descriptors (using RDKit) in the linear regression model to see which ones are most predictive of activity. Add more descriptors from `rdkit.Chem.Descriptors`.
3. **Data Visualization:** Create scatter plots of molecular weight vs. pIC50, or other descriptor combinations, to visually explore SAR trends. Use `matplotlib` or `seaborn` in your notebook.

4. **Activity Cliffs:** Identify activity cliffs – pairs of compounds with very similar structures but significantly different activities. This requires more sophisticated SAR analysis.
5. **Different Machine Learning Models:** Replace the linear regression model with a more complex model like Random Forest or Support Vector Regression to see if you can improve prediction accuracy. You'll likely need to tune hyperparameters.

Key Considerations and Best Practices:

- **Error Handling:** The provided code includes basic error handling, but you should add more robust error checking (e.g., handling invalid SMILES strings more gracefully).
- **Modularity:** Refactor the code into functions to improve readability and reusability.
- **Documentation:** Add comments to explain the purpose of each code block.
- **Version Control:** Use Git to track changes to your code.
- **Data Validation:** Check the integrity of the data loaded from the CSV file.
- **Logging:** Use a logging library (e.g., `logging`) to record important events and errors.

Summary (in two languages):

English:

This comprehensive guide provides a detailed analysis plan for your ChEMBL 35 data analysis project, including solutions to encountered errors, structured SQL and Python code examples, and suggestions for further exploration. Remember to adapt the file paths and SMILES data based on your specific project setup.

Vietnamese:

Hướng dẫn toàn diện này cung cấp một kế hoạch phân tích chi tiết cho dự án phân tích dữ liệu ChEMBL 35 của bạn, bao gồm các giải pháp cho các lỗi gặp phải, các ví dụ về mã SQL và Python có cấu trúc và các đề xuất để khám phá thêm. Hãy nhớ điều chỉnh các đường dẫn tệp và dữ liệu SMILES dựa trên thiết lập dự án cụ thể của bạn.

Remember to replace `/path/to/your/project` with the actual path to your project directory. This detailed guide should provide a solid foundation for your ChEMBL 35 analysis project. Good luck!