PharmApp Suite

🧠 AI for Drug Discovery and Development 🧪

🧩 Topic: Plotting IC50 distribution across targets

Okay, I understand. You're working on a Chembl 35 data analysis project using RDKit within an AIMLops-structured environment, and you're encountering some issues with your SQL queries and potentially your scikit-learn version. Let's break down the analysis, provide the code (SQL and Python), and address your errors, keeping in mind the limited data constraint (100 rows).

Here's a breakdown of the analysis and the code:

1. Analysis Model

The goal seems to be to extract relevant data from the ChEMBL database, likely to analyze activity data (IC50, Ki, etc.) for specific targets or compounds. A typical analysis might involve:

- **Data Extraction:** Querying the ChEMBL database to retrieve activity data, compound structures (SMILES strings), and target information.
- **Data Cleaning and Transformation:** Handling missing values, converting activity units, and transforming SMILES strings into RDKit molecules.
- **Feature Engineering:** Generating molecular descriptors using RDKit (e.g., topological polar surface area, number of rotatable bonds, etc.).
- **Data Analysis and Modeling:** Exploring the relationship between molecular descriptors and activity values, potentially using machine learning models (e.g., regression) to predict activity.

2. Addressing Errors

- **ERROR: operator does not exist: numeric ~ unknown, LINE 12: AND act.standard_value ~ '[1]+$'**: This error arises because the ~ operator is typically used for regular expression matching against text data. However, the `standard_value` column is likely of a numeric data type (e.g., `numeric`, `float`). You need to use a numeric comparison or cast the column to text if you want to use regular expressions.

- **old scikit-learn version does not support parameters squared=False in the mean_squared_error function**: This simply means you are using an older version of scikit-learn. You'll need to either update scikit-learn (`pip install --upgrade scikit-learn`) or remove the `squared=False` argument. Since you are using the AIMLops structure, updating the environment might be preferable. But here's how to proceed without updating. If you want the Root Mean Squared Error (RMSE), take the square root of the Mean Squared Error (MSE) result.

3. Code (SQL and Python)

Here's the code, following your specified file structure and addressing the errors.

```sql
-- SQL: Extracting Data from ChEMBL (limit to 100 rows)
-- File: ../data/chembl_data.csv

SELECT
    cmp.chembl_id,
    cmp.molecule_structures,
    act.standard_type,
```

[1] 0-9.

```
    act.standard_value,
    act.standard_units,
    act.target_chembl_id
FROM
    compound_structures cmp
JOIN
    activities act ON cmp.molregno = act.molregno
WHERE
    act.standard_type IN ('IC50', 'Ki', 'Kd')  -- Example activity types
    AND act.standard_units = 'nM'
    AND act.standard_value IS NOT NULL
    AND act.standard_value > 0  --Avoid log transform of 0
    AND act.standard_value::text ~ '^[0-9\.]+$' -- Ensure standard_value is numeric
LIMIT 100;
```

Explanation:

- `cmp.chembl_id`: The ChEMBL ID of the compound.
- `cmp.molecule_structures`: Contains SMILES strings of the molecule.
- `act.standard_type`: The type of activity (e.g., IC50, Ki).
- `act.standard_value`: The activity value.
- `act.standard_units`: The units of the activity value.
- `act.target_chembl_id`: The ChEMBL ID of the target.
- `WHERE` clause filters the data to include only IC50, Ki, and Kd values in nM. It also ensures that `standard_value` is not null and contains a non-zero positive numeric value. The `act.standard_value::text ~ '^[0-9\.]+$'` casts the numeric value to text before using the regex.
- `LIMIT 100` limits the output to 100 rows.

```
# Python: Analyzing ChEMBL Data using RDKit
# File: notebooks/Topic_CheMBL_35_72_1_data_analysis.ipynb

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

# Define base path
base_path = "../data"  # Adjust if your data directory is different

# Load data from CSV
data_file = os.path.join(base_path, "chembl_data.csv")
try:
    df = pd.read_csv(data_file)
except FileNotFoundError:
    print(f"Error: File not found at {data_file}.  Make sure you ran the SQL query and
saved the result to this location.")
    exit()

# Data Cleaning and Transformation
df = df.dropna(subset=['molecule_structures'])  # Remove rows with missing SMILES
df = df[df['standard_value'].notna()] #drop NA values

# Convert standard_value to numeric
df['standard_value'] = pd.to_numeric(df['standard_value'], errors='coerce')
```

```python
# Convert SMILES to RDKit molecules
def smiles_to_mol(smiles):
    try:
        mol = Chem.MolFromSmiles(smiles)
        if mol is not None:
            return mol
        else:
            return None
    except:
        return None


df['molecule'] = df['molecule_structures'].apply(smiles_to_mol)
df = df.dropna(subset=['molecule']) #remove rows that failed smiles conversion


# Activity Transformation (pIC50) - common for modeling
df['pIC50'] = -np.log10(df['standard_value'] * 1e-9)  # Convert nM to M and calculate
pIC50


# Feature Engineering (Molecular Descriptors)
def calculate_descriptors(mol):
    try:
        descriptors = {
            "MW": Descriptors.MolWt(mol),
            "LogP": Descriptors.MolLogP(mol),
            "HBD": Descriptors.NumHDonors(mol),
            "HBA": Descriptors.NumHAcceptors(mol),
            "TPSA": Descriptors.TPSA(mol),
            "RotatableBonds": Descriptors.NumRotatableBonds(mol)
        }
        return descriptors
    except:
        return None

df['descriptors'] = df['molecule'].apply(calculate_descriptors)
df = df.dropna(subset=['descriptors'])
df = pd.concat([df.drop(['descriptors'], axis=1), df['descriptors'].apply(pd.Series)],
axis=1)


# Data Splitting
X = df[['MW', 'LogP', 'HBD', 'HBA', 'TPSA', 'RotatableBonds']]
y = df['pIC50']
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 Evaluation
y_pred = model.predict(X_test)
mse = mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse) #Calculating RMSE because squared=False is unavailable
```

```
print(f"Mean Squared Error: {mse}")
print(f"Root Mean Squared Error: {rmse}")
```

Explanation:

- **Imports:** Imports necessary libraries (pandas, RDKit, scikit-learn).
- **Data Loading:** Loads the CSV file you created from the SQL query.
- **Data Cleaning:** Removes rows with missing SMILES strings or activity values.
- **SMILES Conversion:** Converts SMILES strings to RDKit `Mol` objects. Includes error handling for invalid SMILES.
- **Activity Transformation:** Converts `standard_value` (assumed to be in nM) to pIC50 values (a logarithmic scale, which is often better for modeling). Important: Adjust the transformation if your `standard_units` are different.
- **Feature Engineering:** Calculates molecular descriptors using RDKit. This example calculates molecular weight (MW), LogP, hydrogen bond donors (HBD), hydrogen bond acceptors (HBA), topological polar surface area (TPSA), and the number of rotatable bonds.
- **Data Splitting:** Splits the data into training and testing sets.
- **Model Training:** Trains a linear regression model.
- **Model Evaluation:** Predicts activity values for the test set and calculates the Mean Squared Error (MSE) and Root Mean Squared Error (RMSE).
- **Error Handling:** Includes `try...except` blocks for error handling during SMILES conversion and descriptor calculation. Rows that cause errors are removed from the DataFrame.

Important Considerations:

- **Target Selection:** In the SQL query, you may want to add a `WHERE` clause to filter by a specific `target_chembl_id` to focus your analysis on a particular target.
- **Activity Type:** The code assumes you are primarily interested in IC50, Ki, and Kd values. Adjust the `WHERE` clause in the SQL query if you need to include other activity types.
- **Units:** The code assumes activity values are in nM. Adjust the activity transformation step if your data uses different units.
- **Descriptor Selection:** The code calculates a basic set of molecular descriptors. You may want to experiment with different descriptors based on the specific target and compounds you are analyzing.
- **Model Selection:** Linear regression is a simple model. You may want to explore other machine learning models (e.g., random forests, support vector machines) if linear regression doesn't provide satisfactory results.
- **Error Handling:** The error handling is basic. You might want to add more sophisticated error handling and logging to your code.

4. Five Examples (Based on Possible Analysis Scenarios)

Here are five examples of how you could use this code in different analysis scenarios:

1. **Target-Specific Analysis:**
    - Modify the SQL query to filter for a specific `target_chembl_id`. For example, if you are interested in analyzing data for EGFR (Epidermal Growth Factor Receptor), find its ChEMBL ID and add `AND act.target_chembl_id = 'CHEMBL203'` to your SQL query.
    - The Python code will then analyze the activity data and molecular descriptors for compounds active against EGFR.
2. **Activity Type Comparison:**
    - Modify the SQL query to include only IC50 values (`AND act.standard_type = 'IC50'`) and run the Python code. Then, repeat the process for Ki values (`AND act.standard_type = 'Ki'`).

    - Compare the performance of the linear regression model for predicting IC50 versus Ki values. This might reveal differences in the structure-activity relationship for different activity types.
3. **Descriptor Importance Analysis:**
    - After training the linear regression model, examine the coefficients of the model to determine which molecular descriptors have the greatest impact on predicted activity.
    - This can provide insights into the structural features that are most important for activity against the target.
4. **Outlier Analysis:**
    - Identify compounds in the test set for which the predicted activity values are significantly different from the actual activity values (i.e., outliers).
    - Examine the structures of these outlier compounds to see if they have any unusual features that might explain why they are not well-predicted by the model.
5. **Virtual Screening:**
    - After training the model, use it to predict the activity of new compounds (e.g., compounds from a virtual library).
    - Select the compounds with the highest predicted activity values for further experimental testing.

In Summary (En Resumen)

This comprehensive guide provides you with the SQL and Python code to start your ChEMBL 35 data analysis project. It addresses the errors you encountered and gives you a clear framework for data extraction, cleaning, transformation, feature engineering, and modeling. Remember to adapt the code to your specific analysis goals, and experiment with different parameters, descriptors, and models to achieve the best results.

Español (Spanish)

Esta guía completa te proporciona el código SQL y Python para comenzar tu proyecto de análisis de datos de ChEMBL 35. Aborda los errores que encontraste y te brinda un marco claro para la extracción, limpieza, transformación, ingeniería de características y modelado de datos. Recuerda adaptar el código a tus objetivos de análisis específicos y experimentar con diferentes parámetros, descriptores y modelos para lograr los mejores resultados. ¡Buena suerte!