PharmApp Suite

🧠 AI for Drug Discovery and Development 🧪

🧩 Topic: Visualization of protein families

Okay, I understand. You're working on a Chembl 35 data analysis project using RDKit, aiming to extract and analyze data relevant to drug research and development. You have a database server running PostgreSQL with the Chembl 35 database on `192.168.206.136`. You're using the AIMLops template, storing SQL/CSV data in `../data/`, and running Jupyter Notebooks in the `notebook` directory. Your notebook filenames follow the format `Topic_CheMBL_35_38_1_*` and `Topic_CheMBL_35_38_2_*`. You've also encountered two errors: a PostgreSQL error related to the ~ operator and a scikit-learn version issue. You want guidance, SQL and Python code examples (limited to 100 rows), and a clear explanation of the analysis.

Let's break down the process, address the errors, and create the necessary code.

1. Analysis Model

The goal is to extract relevant data from ChEMBL 35, filter it based on specific criteria (e.g., activity types, standard values), and then perform some initial analysis using RDKit and scikit-learn. Here's a potential model:

- **Data Extraction & Filtering (SQL):** Retrieve data from the `activities`, `assays`, `molecule_dictionary`, `compound_structures`, and other relevant tables. Filter based on activity type (e.g., 'IC50'), confidence score, and standard values that are numeric. Limit the results to 100 rows for efficiency.
- **Data Preparation (Python/RDKit):**
    - Load the data from the CSV file into a Pandas DataFrame.
    - Clean the data: Handle missing values, convert data types.
    - Convert SMILES strings to RDKit molecules.
    - Calculate molecular descriptors using RDKit (e.g., molecular weight, LogP, number of hydrogen bond donors/acceptors).
- **Data Analysis & Modeling (Python/scikit-learn):**
    - **Exploratory Data Analysis (EDA):** Visualize the distribution of molecular descriptors and activity values. Look for correlations.
    - **Predictive Modeling (Example):** Build a simple regression model to predict activity (e.g., pIC50) from molecular descriptors. This is just an example; other models (classification, clustering) could be used.
    - **Model Evaluation:** Evaluate the performance of the model using appropriate metrics (e.g., R-squared, RMSE).
- **Interpretation:** Draw conclusions about the relationship between molecular structure and activity.

2. SQL Code (data/Topic_CheMBL_35_38_data_extraction.sql)

```sql
--  Filter for IC50 values, confidence score above 8, and numeric standard values
SELECT
    md.chembl_id,
    cs.canonical_smiles,
    act.standard_type,
    act.standard_value,
    act.standard_units,
    act.pchembl_value,
```

```
    assays.confidence_score
FROM
    activities act
JOIN
    assays ON act.assay_id = assays.assay_id
JOIN
    molecule_dictionary md ON act.molregno = md.molregno
JOIN
    compound_structures cs ON md.molregno = cs.molregno
WHERE
    act.standard_type = 'IC50'
    AND assays.confidence_score >= 8
    AND act.standard_value::text ~ '^[0-9.]+$'  -- Corrected line:  explicitly cast to
text for regex matching
    AND act.standard_units = 'nM'
LIMIT 100;
```

Explanation of the SQL Code:

- **`SELECT` Clause:** Selects the ChEMBL ID, SMILES string, standard type, standard value, standard units, pChEMBL value, and confidence score.
- **`FROM` Clause:** Specifies the tables to retrieve data from: `activities`, `assays`, `molecule_dictionary`, and `compound_structures`.
- **`JOIN` Clauses:** Connects the tables based on the appropriate foreign keys.
- **`WHERE` Clause:** Filters the data based on the following criteria:
    - `act.standard_type = 'IC50'`: Filters for IC50 activity values.
    - `assays.confidence_score >= 8`: Filters for assays with a confidence score of 8 or higher.
    - `act.standard_value::text ~ '^[0-9.]+$'`: This is the corrected line to fix your error. The problem was that PostgreSQL's numeric types don't directly support regular expression matching with the ~ operator. The solution is to explicitly cast the `standard_value` to text (`::text`) before applying the regex. The regex `'^[0-9.]+$'` ensures that the `standard_value` consists only of digits and periods (to allow for decimal values).
    - `act.standard_units = 'nM'`: Filters for activities with standard units in nM (nanomolar).
- **`LIMIT 100`:** Limits the result set to 100 rows.

Important: Run this SQL query in pgAdmin and save the result as a CSV file in your `../data/` directory. Name the file something like `Topic_CheMBL_35_38_data.csv`.

3. Python Code (notebook/Topic_CheMBL_35_38_1_data_preparation.ipynb)

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

# Define the base path
base_path = os.path.abspath(os.path.join(os.getcwd(), ".."))  # Go up one level from
the notebook directory
data_path = os.path.join(base_path, "data", "Topic_CheMBL_35_38_data.csv")

# Load the data
try:
    df = pd.read_csv(data_path)
except FileNotFoundError:
```

```
    print(f"Error: File not found at {data_path}.  Make sure you've run the SQL query
and saved the CSV.")
    exit()

# Data Cleaning and Preparation
print("Original DataFrame:")
print(df.head())

# Handle missing values (e.g., fill with 0 or drop)
df = df.dropna(subset=['canonical_smiles', 'standard_value'])  # Drop rows with
missing SMILES or standard_value
df['standard_value'] = pd.to_numeric(df['standard_value'], errors='coerce') # Convert
to numeric, coercing errors to NaN
df = df.dropna(subset=['standard_value']) # Drop newly created NaNs

# Convert IC50 to pIC50
df['pIC50'] = -np.log10(df['standard_value'] / 1e9)  # Convert nM to Molar, then to
pIC50

# Create RDKit molecules
df['molecule'] = df['canonical_smiles'].apply(lambda x: Chem.MolFromSmiles(x))
df = df.dropna(subset=['molecule']) # Remove rows where molecule creation failed
(invalid SMILES)

# Calculate Molecular Descriptors (Example: Molecular Weight and LogP)
def calculate_mw(mol):
    return Descriptors.MolWt(mol)

def calculate_logp(mol):
    return Descriptors.MolLogP(mol)

df['mol_weight'] = df['molecule'].apply(calculate_mw)
df['log_p'] = df['molecule'].apply(calculate_logp)

print("\nDataFrame with RDKit molecules and descriptors:")
print(df.head())

# Save the processed dataframe (optional)
processed_data_path = os.path.join(base_path, "data",
"Topic_CheMBL_35_38_processed_data.csv")
df.to_csv(processed_data_path, index=False)
print(f"\nProcessed data saved to: {processed_data_path}")
```

Explanation of the Python Code (Data Preparation):

- **Import Libraries:** Imports `os`, `pandas`, RDKit modules (`Chem`, `Descriptors`), and `numpy`.
- **Define Paths:** Constructs the paths to the data file using `os.path.join`. Crucially, the `base_path` is calculated to go up one level from the notebook directory to the root of your project.
- **Load Data:** Loads the CSV data into a Pandas DataFrame using `pd.read_csv`. Includes error handling for the case where the CSV file is not found.
- **Data Cleaning:**
    - Handles missing values: `dropna` is used to remove rows with missing SMILES or standard values. Missing values can cause problems with RDKit and numerical calculations.
    - Converts 'standard_value' to numeric: `pd.to_numeric` is used to ensure the standard values are numeric, with error handling to convert non-numeric values to NaN, which are then dropped.

- **pIC50 Conversion:** Calculates pIC50 values from IC50 values using the formula: `pIC50 = -log10(IC50 / 1e9)`. The IC50 is divided by 1e9 to convert from nM to Molar units.
- **RDKit Molecule Creation:** Converts SMILES strings to RDKit molecule objects using `Chem.MolFromSmiles`. Rows where molecule creation fails are removed.
- **Molecular Descriptor Calculation:**
    - Defines functions `calculate_mw` and `calculate_logp` to calculate molecular weight and LogP using RDKit's `Descriptors` module.
    - Applies these functions to the 'molecule' column to create new 'mol_weight' and 'log_p' columns in the DataFrame.
- **Saving Processed Data:** Saves the processed DataFrame to a new CSV file.

4. Python Code (notebook/Topic_CheMBL_35_38_2_data_analysis.ipynb)

```
import os
import pandas as pd
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
from sklearn.preprocessing import StandardScaler

# Define the base path
base_path = os.path.abspath(os.path.join(os.getcwd(), ".."))
data_path = os.path.join(base_path, "data", "Topic_CheMBL_35_38_processed_data.csv")

# Load the processed data
try:
    df = pd.read_csv(data_path)
except FileNotFoundError:
    print(f"Error: File not found at {data_path}.  Make sure you've run the data
preparation notebook.")
    exit()

# Data Analysis and Modeling
print("DataFrame for Analysis:")
print(df.head())

# 1. Exploratory Data Analysis (EDA)
sns.histplot(df['pIC50'], kde=True)
plt.title('Distribution of pIC50 Values')
plt.xlabel('pIC50')
plt.ylabel('Frequency')
plt.show()

sns.scatterplot(x='mol_weight', y='pIC50', data=df)
plt.title('pIC50 vs. Molecular Weight')
plt.xlabel('Molecular Weight')
plt.ylabel('pIC50')
plt.show()

# 2. Predictive Modeling (Linear Regression Example)
# Select features (descriptors) and target (pIC50)
X = df[['mol_weight', 'log_p']]
y = df['pIC50']

# Handle missing values (important before modeling!)
X = X.fillna(X.mean())  # Impute missing values with the column's average
```

```
# 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)

# Feature scaling (important for linear models)
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)


# Train a linear regression model
model = LinearRegression()
model.fit(X_train, y_train)

# Make predictions on the test set
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}')

# Plot predicted vs. actual values
plt.scatter(y_test, y_pred)
plt.xlabel('Actual pIC50')
plt.ylabel('Predicted pIC50')
plt.title('Actual vs. Predicted pIC50')
plt.show()
```

Explanation of the Python Code (Data Analysis and Modeling):

- **Import Libraries:** Imports necessary libraries for data analysis, visualization, and machine learning.
- **Load Data:** Loads the processed data from the CSV file.
- **Exploratory Data Analysis (EDA):**
    - **Histograms:** Creates a histogram of pIC50 values to visualize their distribution.
    - **Scatter Plots:** Creates a scatter plot of pIC50 vs. molecular weight to explore the relationship between these variables.
- **Predictive Modeling (Linear Regression):**
    - **Select Features and Target:** Selects 'mol_weight' and 'log_p' as features (independent variables) and 'pIC50' as the target (dependent variable).
    - **Handle Missing Values:** `X = X.fillna(X.mean())` imputes any remaining missing values in the feature matrix with the mean of the respective column. This is crucial for avoiding errors in the model training.
    - **Split Data:** Splits the data into training and testing sets using `train_test_split`.
    - **Feature Scaling:** Scales the features using `StandardScaler`. This is important for linear models because it ensures that features with larger values don't dominate the model. `StandardScaler` centers the data around zero and scales it to have unit variance.
    - **Train Model:** Creates a `LinearRegression` model and trains it on the training data using `model.fit`.
    - **Make Predictions:** Makes predictions on the test set using `model.predict`.
    - **Evaluate Model:** Calculates the mean squared error (MSE) and R-squared to evaluate the model's performance.

- **Plot Results:** Creates a scatter plot of actual vs. predicted pIC50 values to visualize the model's performance.

Addressing the `squared=False` Error:

The error "old scikit-learn version does not support parameters squared=False in the mean_squared_error function" indicates that you're using an older version of scikit-learn. The `squared=False` parameter was added to `mean_squared_error` in a later version to allow you to directly obtain the Root Mean Squared Error (RMSE).

Solution: The easiest solution is to upgrade your scikit-learn version. Run the following command in your terminal or Jupyter Notebook:

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

After upgrading, the `squared=False` parameter will be available. However, for backwards compatibility, you can also calculate the RMSE manually:

```
mse = mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse)  # Calculate RMSE manually
print(f'Mean Squared Error: {mse}')
print(f'Root Mean Squared Error: {rmse}') # print results
print(f'R-squared: {r2}')
```

5. Examples

Here are five examples illustrating how the code can be used and adapted:

- **Example 1: Using Different Descriptors:** Instead of just molecular weight and LogP, calculate a wider range of RDKit descriptors (e.g., number of hydrogen bond donors/acceptors, topological polar surface area (TPSA), rotatable bond count). Add these to the `X` matrix for the linear regression model and see how they affect the model's performance.

  ```
  # Add more descriptors
  df['num_hbd'] = df['molecule'].apply(Descriptors.NumHDonors)
  df['num_hba'] = df['molecule'].apply(Descriptors.NumHAcceptors)
  df['tpsa'] = df['molecule'].apply(Descriptors.TPSA)
  df['rotatable_bonds'] = df['molecule'].apply(Descriptors.NumRotatableBonds)

  X = df[['mol_weight', 'log_p', 'num_hbd', 'num_hba', 'tpsa', 'rotatable_bonds']]
  ```

- **Example 2: Using a Different Model:** Try a different regression model, such as Random Forest Regression or Support Vector Regression (SVR). These models can often capture non-linear relationships in the data better than linear regression.

  ```
  from sklearn.ensemble import RandomForestRegressor

  model = RandomForestRegressor(n_estimators=100, random_state=42)  # Example with
  100 trees
  model.fit(X_train, y_train)
  y_pred = model.predict(X_test)
  ```

- **Example 3: Using Different Activity Types:** Modify the SQL query to extract data for a different activity type (e.g., Ki, EC50). You'll need to adjust the `WHERE` clause in the SQL query accordingly. Then, rerun the Python notebooks to analyze the data for the new activity type.

  ```
  -- Modify the SQL query
  WHERE
      act.standard_type = 'Ki'
      AND assays.confidence_score >= 8
  ```

```
    AND act.standard_value::text ~ '^[0-9.]+$'
    AND act.standard_units = 'nM'
```

- **Example 4: Filtering by Specific Targets:** Add a filter to the SQL query to only retrieve data for compounds that target a specific protein or enzyme. This requires joining with the `target_dictionary` table and filtering by the `target_chembl_id`.

```
-- Modify the SQL query
JOIN
    target_dictionary td ON assays.tid = td.tid
WHERE
    act.standard_type = 'IC50'
    AND assays.confidence_score >= 8
    AND act.standard_value::text ~ '^[0-9.]+$'
    AND act.standard_units = 'nM'
    AND td.target_chembl_id = 'CHEMBL205'  -- Example target
```

- **Example 5: Analyzing a Specific Subset of Compounds:** Add a filter to the SQL query to retrieve data for compounds with specific substructures or properties. For example, you could filter for compounds containing a particular chemical moiety. This might require using more advanced SQL techniques or external cheminformatics tools.

```
-- More advanced filtering might require external tools or database functions
-- Example (conceptually - this might need adaptation depending on your DB
setup):
-- WHERE ... AND some_function_to_check_substructure(cs.canonical_smiles,
'C(=O)N')  -- Compounds with amide group
```

Summary (English):

This provides a comprehensive guide to your ChEMBL 35 data analysis project. It includes corrected SQL code for data extraction, Python code for data preparation and analysis, and solutions to the errors you encountered. The five examples demonstrate how to adapt the code for different analyses. Remember to upgrade scikit-learn to avoid errors related to the `squared=False` parameter in `mean_squared_error`. Carefully follow the instructions and adapt the code to your specific research questions. Good luck!

Resumen (Español):

Esto proporciona una guía completa para su proyecto de análisis de datos de ChEMBL 35. Incluye código SQL corregido para la extracción de datos, código Python para la preparación y el análisis de datos y soluciones a los errores que encontró. Los cinco ejemplos demuestran cómo adaptar el código para diferentes análisis. Recuerde actualizar scikit-learn para evitar errores relacionados con el parámetro `squared=False` en `mean_squared_error`. Siga cuidadosamente las instrucciones y adapte el código a sus preguntas de investigación específicas. ¡Buena suerte!