PharmApp Suite

🧠 AI for Drug Discovery and Development 🧪

🧩 Topic: Mechanism-of-action clustering of ChEMBL drugs

Okay, I understand. You're working on a Chembl 35 data analysis project using RDKit within an MLOps framework, and you need guidance on running your code, specifically with a focus on the errors you're encountering. I will provide a step-by-step guide with SQL and Python code examples, keeping the data size small (100 rows) for efficient execution.

Here's a breakdown and solution:

1. Analysis Model & Strategy

Your project involves extracting and analyzing data from the ChEMBL database to support drug research and development. Here's a general analysis model you'll likely follow:

- **Data Extraction (SQL):** Query the ChEMBL database (running on PostgreSQL) to retrieve relevant data, focusing on activity data. This will likely involve joining multiple tables (e.g., activities, assays, molecules) to get information about compounds, their biological activities, and assay details. You'll filter this data based on specific criteria (e.g., activity type, target organism, value ranges).
- **Data Cleaning & Preprocessing (Python):** Load the extracted data (CSV) into a Pandas DataFrame. Clean the data by handling missing values, converting data types, and removing duplicates. The `standard_value` column needs to be appropriately processed (e.g., converting to numeric).
- **Feature Engineering (Python/RDKit):** Generate molecular descriptors using RDKit. These descriptors are numerical representations of molecular structures and can be used as features in machine learning models. Common descriptors include molecular weight, logP, topological polar surface area (TPSA), and number of rotatable bonds.
- **Exploratory Data Analysis (EDA) (Python):** Explore the data using visualizations and statistical summaries. This helps identify trends, outliers, and relationships between variables. Examples include histograms of activity values, scatter plots of molecular descriptors, and box plots comparing activity across different target classes.
- **Modeling (Python):** (Optional - depending on the task) Build machine learning models to predict activity, classify compounds, or identify structure-activity relationships. Common algorithms include Random Forest, Support Vector Machines (SVMs), and linear regression.
- **Evaluation (Python):** Evaluate the performance of the models using appropriate metrics (e.g., RMSE, R-squared for regression; accuracy, precision, recall, F1-score for classification).

Error Analysis & Solutions

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

  This error indicates that you're trying to use the regular expression operator ~ on a column that is already defined as a `numeric` type. PostgreSQL doesn't directly support regular expression matching on numeric columns. The solution is to cast the `standard_value` column to `TEXT` before applying the regular expression.

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

  This means you're using an older version of scikit-learn. There are two ways to resolve this:

1. **Upgrade Scikit-learn:** The best option is to upgrade your scikit-learn version to the latest stable release using `pip install -U scikit-learn`.
2. **Remove `squared=False`:** If upgrading is not possible, remove the `squared=False` parameter from the `mean_squared_error` function. Note that this will return the Mean Squared Error (MSE) instead of the Root Mean Squared Error (RMSE). You'll need to take the square root of the result manually to get the RMSE if that's what you need.

2. Code Implementation

Folder Structure (Based on AIMLops template)

```
.
├── data
│   └── chembl_activity_data.csv
├── notebooks
│   ├── Topic_CheMBL_35_94_1_data_extraction_preprocessing.ipynb
│   └── Topic_CheMBL_35_94_2_feature_engineering_analysis.ipynb
└── ... (other directories like models, src, etc.)
```

SQL Code (data/extract_chembl_data.sql)

```sql
-- extract_chembl_data.sql
-- Extracts activity data from ChEMBL for the specified target.
-- Limits to 100 rows for demonstration purposes.

SELECT
    act.molregno,  -- Molecule Registry Number
    act.standard_value,  -- Standard Value of Activity
    act.standard_units,  -- Standard Units
    act.standard_type,   -- Standard Type (e.g., IC50, Ki)
    act.assay_id,
    md.chembl_id,    -- ChEMBL ID of the compound
    cs.canonical_smiles -- SMILES string of the molecule
FROM
    activities act
JOIN
    molecule_dictionary md ON act.molregno = md.molregno
JOIN
    compound_structures cs ON md.molregno = cs.molregno
WHERE
    act.standard_type = 'IC50'  -- Filter for IC50 values
    AND act.standard_units = 'nM'  -- Filter for nM units
    AND act.standard_relation = '=' -- Filter for direct equality
    AND act.standard_value IS NOT NULL
    AND CAST(act.standard_value AS TEXT) ~ '^[0-9\.]+$'  -- REGEX ON TEXT
LIMIT 100;
```

Run on pgAdmin:

1. Open pgAdmin and connect to your PostgreSQL server (192.168.206.136, rd/rd, chembl_35).
2. Open a new query window and paste the SQL code.
3. Execute the query.
4. **Export to CSV:** Right-click on the query result and choose "Copy with Headers". Paste this into a text editor and save as `../data/chembl_activity_data.csv`. Ensure the CSV uses commas as delimiters. (Alternatively, you can use `\copy` command in `psql` to export directly to a file.)

Python Code (notebooks/Topic_CheMBL_35_94_1_data_extraction_preprocessing.ipynb)

```python
# notebooks/Topic_CheMBL_35_94_1_data_extraction_preprocessing.ipynb
import pandas as pd
import os
```

```python
# Define paths using os.path.join
base_path = os.getcwd()  # Current working directory (where notebook is)
data_path = os.path.join(base_path, 'data', 'chembl_activity_data.csv')

# Load the data
try:
    df = pd.read_csv(data_path)
except FileNotFoundError:
    print(f"Error: File not found at {data_path}")
    raise  # Re-raise the exception to stop execution

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

# Handle missing values (if any)
df = df.dropna()  # Remove rows with any missing values. Consider more sophisticated
imputation.

# Convert standard_value to numeric
df['standard_value'] = pd.to_numeric(df['standard_value'], errors='coerce') # coerce
turns invalid parsing into NaN

# Remove any remaining NaN values created by conversion
df = df.dropna(subset=['standard_value'])

# Remove duplicate entries based on chembl_id (or other relevant columns)
df = df.drop_duplicates(subset=['chembl_id'])

# Convert IC50 to pIC50
# Function to convert IC50 to pIC50
import numpy as np

def ic50_to_pic50(ic50_nM):
    """Converts IC50 in nM to pIC50."""
    pIC50 = -np.log10(ic50_nM * 1e-9)  # Convert nM to Molar
    return pIC50

df['pIC50'] = df['standard_value'].apply(ic50_to_pic50)
print("\nCleaned DataFrame:")
print(df.head())

# Save the cleaned dataframe
cleaned_data_path = os.path.join(base_path, 'data', 'chembl_activity_cleaned.csv')
df.to_csv(cleaned_data_path, index=False)
print(f"\nCleaned data saved to {cleaned_data_path}")
```

Python Code (notebooks/Topic_CheMBL_35_94_2_feature_engineering_analysis.ipynb)

```python
# notebooks/Topic_CheMBL_35_94_2_feature_engineering_analysis.ipynb
import pandas as pd
import os
from rdkit import Chem
from rdkit.Chem import Descriptors
import matplotlib.pyplot as plt
import seaborn as sns

# Define paths
base_path = os.getcwd()
```

```python
cleaned_data_path = os.path.join(base_path, 'data', 'chembl_activity_cleaned.csv')

# Load the cleaned data
try:
    df = pd.read_csv(cleaned_data_path)
except FileNotFoundError:
    print(f"Error: File not found at {cleaned_data_path}")
    raise

# Feature Engineering with RDKit

def calculate_descriptors(smiles):
    """Calculates molecular descriptors using RDKit."""
    mol = Chem.MolFromSmiles(smiles)
    if mol is None:
        return None, None, None, None #Handle Invalid SMILES
    mw = Descriptors.MolWt(mol)
    logp = Chem.Crippen.MolLogP(mol)
    tpsa = Chem.rdMolDescriptors.CalcTPSA(mol)
    rotatable_bonds = Chem.rdMolDescriptors.CalcNumRotatableBonds(mol)
    return mw, logp, tpsa, rotatable_bonds

# Apply the function to create new columns
df[['molecular_weight', 'logp', 'tpsa', 'rotatable_bonds']] =
df['canonical_smiles'].apply(lambda x: pd.Series(calculate_descriptors(x)))

# Handle cases with invalid SMILES
df = df.dropna(subset=['molecular_weight', 'logp', 'tpsa', 'rotatable_bonds'])

print("\nDataFrame with Descriptors:")
print(df.head())

# Exploratory Data Analysis (EDA)

# Histogram of pIC50 values
plt.figure(figsize=(8, 6))
sns.histplot(df['pIC50'], bins=20, kde=True)
plt.title('Distribution of pIC50 Values')
plt.xlabel('pIC50')
plt.ylabel('Frequency')
plt.show()

# Scatter plot of LogP vs. pIC50
plt.figure(figsize=(8, 6))
sns.scatterplot(x='logp', y='pIC50', data=df)
plt.title('LogP vs. pIC50')
plt.xlabel('LogP')
plt.ylabel('pIC50')
plt.show()

# Boxplot of pIC50 for different standard_types (e.g., IC50, Ki) - if there are
multiple
if len(df['standard_type'].unique()) > 1:
    plt.figure(figsize=(8, 6))
    sns.boxplot(x='standard_type', y='pIC50', data=df)
    plt.title('pIC50 by Standard Type')
    plt.xlabel('Standard Type')
    plt.ylabel('pIC50')
    plt.show()
```

```
# Correlation matrix (optional)
correlation_matrix = df[['pIC50', 'molecular_weight', 'tpsa', 'logp',
'rotatable_bonds']].corr()
plt.figure(figsize=(8,6))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm')
plt.title("Correlation Matrix of pIC50 and Descriptors")
plt.show()

print("\nEDA Completed")

# Example Modeling (Simple Linear Regression) - Optional

from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score

# Prepare data for modeling
X = df[['molecular_weight', 'logp', 'tpsa', 'rotatable_bonds']]
y = df['pIC50']

# 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 the 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 = mse**0.5  # Calculate RMSE manually

r2 = r2_score(y_test, y_pred)

print("\nModel Evaluation:")
print(f"Mean Squared Error: {mse}")
print(f"Root Mean Squared Error: {rmse}")
print(f"R-squared: {r2}")
```

3. Code Explanation (In two languages: English and Vietnamese)

English:

- **SQL:** The SQL code extracts relevant data from the ChEMBL database, specifically activity data for IC50 values. It joins tables to retrieve molecule information (SMILES strings) and filters the data to include only records with valid IC50 values in nM. The `LIMIT 100` clause ensures that only 100 rows are retrieved for faster processing. The critical fix here is `CAST(act.standard_value AS TEXT) ~ '^[0-9\.]+$'`, which converts the numeric column to text before applying the regex.
- **Python:**
  - The Python code loads the extracted CSV data into a Pandas DataFrame.
  - It performs data cleaning steps, including handling missing values and converting the `standard_value` column to a numeric type.

    - It generates molecular descriptors using RDKit (molecular weight, LogP, TPSA, rotatable bonds). Proper error handling for invalid SMILES strings is included.
    - It performs exploratory data analysis (EDA) to visualize the data and identify trends. Examples include histograms of pIC50 values and scatter plots of molecular descriptors.
    - A simple linear regression model is included as an example of how to build a model. The code addresses the potential scikit-learn version issue by calculating RMSE manually. This modeling section is optional.
    - The code uses `os.path.join` to ensure that file paths are platform-independent.

Vietnamese:

- **SQL:** Mã SQL trích xuất dữ liệu liên quan từ cơ sở dữ liệu ChEMBL, đặc biệt là dữ liệu hoạt động cho các giá trị IC50. Nó kết hợp các bảng để truy xuất thông tin phân tử (chuỗi SMILES) và lọc dữ liệu để chỉ bao gồm các bản ghi có giá trị IC50 hợp lệ tính bằng nM. Mệnh đề `LIMIT 100` đảm bảo rằng chỉ có 100 hàng được truy xuất để xử lý nhanh hơn. Sửa chữa quan trọng ở đây là `CAST(act.standard_value AS TEXT) ~ '^[0-9\.]+$'`, chuyển đổi cột số thành văn bản trước khi áp dụng biểu thức chính quy.
- **Python:**
    - Mã Python tải dữ liệu CSV đã trích xuất vào một Pandas DataFrame.
    - Nó thực hiện các bước làm sạch dữ liệu, bao gồm xử lý các giá trị bị thiếu và chuyển đổi cột `standard_value` thành kiểu số.
    - Nó tạo ra các mô tả phân tử bằng RDKit (khối lượng phân tử, LogP, TPSA, số liên kết có thể xoay). Xử lý lỗi thích hợp cho các chuỗi SMILES không hợp lệ được bao gồm.
    - Nó thực hiện phân tích khám phá dữ liệu (EDA) để trực quan hóa dữ liệu và xác định xu hướng. Ví dụ bao gồm biểu đồ tần suất của các giá trị pIC50 và biểu đồ phân tán của các mô tả phân tử.
    - Một mô hình hồi quy tuyến tính đơn giản được bao gồm như một ví dụ về cách xây dựng mô hình. Mã giải quyết vấn đề phiên bản scikit-learn tiềm năng bằng cách tính toán RMSE thủ công. Phần mô hình hóa này là tùy chọn.
    - Mã sử dụng `os.path.join` để đảm bảo rằng đường dẫn tệp độc lập với nền tảng.

4. Example Runs & Expected Output

Here's what you might expect as output from running these scripts (the exact output will vary depending on the specific data in your ChEMBL database):

SQL Output (chembl_activity_data.csv):

The CSV file will contain 100 rows of data, with columns like `molregno`, `standard_value`, `standard_units`, `standard_type`, `assay_id`, `chembl_id`, and `canonical_smiles`.

Python Notebook 1 Output:

```
Original DataFrame:
   molregno  standard_value standard_units standard_type  assay_id  \
0    716255            50.0             nM          IC50     36571
1    716255           100.0             nM          IC50     36571
2    716255            25.0             nM          IC50     36571
...

Cleaned DataFrame:
   molregno  standard_value standard_units standard_type  assay_id  \
0    716255            50.0             nM          IC50     36571
1    716255           100.0             nM          IC50     36571
2    716255            25.0             nM          IC50     36571
...
pIC50
```

```
0  7.301030
1  7.000000
2  7.602060
...

Cleaned data saved to .../data/chembl_activity_cleaned.csv
```

Python Notebook 2 Output (including EDA and Modeling):

You'll see the first few rows of the DataFrame with the newly added descriptor columns. You'll also see the plots generated by the EDA code (histograms, scatter plots, box plots). Finally, the model evaluation output will show the MSE, RMSE, and R-squared values for the linear regression model.

```
DataFrame with Descriptors:
   molregno  standard_value standard_units standard_type  assay_id  \
0    716255            50.0             nM          IC50     36571
1    716255           100.0             nM          IC50     36571
2    716255            25.0             nM          IC50     36571
...

molecular_weight       logp      tpsa  rotatable_bonds
0         339.413  3.48380   60.83   4
1         339.413  3.48380   60.83   4
2         339.413  3.48380   60.83   4
...

EDA Completed

Model Evaluation:
Mean Squared Error: 1.2345
Root Mean Squared Error: 1.1111
R-squared: 0.5678
```

5. Examples

Here are 5 specific examples demonstrating the usage of the code:

1. **Extracting Data for a Specific Target:** Modify the SQL query to filter for activities related to a specific protein target. You'll need to find the appropriate target ID in the ChEMBL database and add a `JOIN` clause with the `target_dictionary` table and a `WHERE` clause filtering on the `target_chembl_id`.
2. **Calculating Additional Descriptors:** Add more molecular descriptors to the `calculate_descriptors` function in the second Python notebook. RDKit provides a wide range of descriptors (e.g., number of hydrogen bond donors, number of hydrogen bond acceptors).
3. **Visualizing Structure-Activity Relationships:** Create more sophisticated visualizations to explore the relationship between molecular descriptors and activity. For example, you could create a 3D scatter plot of LogP, TPSA, and pIC50, or a heatmap of the correlation matrix between all descriptors and pIC50.
4. **Building a More Complex Model:** Replace the simple linear regression model with a more advanced machine learning model, such as Random Forest or Support Vector Machine. You'll need to import the appropriate classes from scikit-learn and tune the model's hyperparameters.
5. **Implementing Cross-Validation:** Implement k-fold cross-validation to obtain more robust estimates of model performance. This involves splitting the data into multiple folds and training and evaluating the model on different combinations of folds. Use scikit-learn's `KFold` or `cross_val_score` functions.

Important Notes:

- **ChEMBL License:** Ensure you comply with the ChEMBL database license agreement when using this data.
- **Database Performance:** For larger datasets, consider optimizing your SQL queries and using database indexing to improve performance.
- **Data Validation:** Thoroughly validate your data at each step of the process to ensure data quality and accuracy. This is especially important when dealing with large, complex datasets like ChEMBL.

This comprehensive guide should provide a solid foundation for your ChEMBL 35 data analysis project. Remember to adapt the code and analysis to your specific research questions. Good luck!