PharmApp Suite

🧠 AI for Drug Discovery and Development 🍼

🧩 Topic: Target prediction for orphan molecules

Okay, I understand. You're working with ChEMBL 35 data using RDKit and want to analyze it within an AIMLops framework. You have a database connection, a file structure, and some error conditions to address. I will provide SQL and Python code, focusing on extracting 100 rows and addressing the specific errors you've encountered.

Analysis Model:

The core idea is to query the ChEMBL database using SQL to extract relevant chemical and bioactivity data. We'll specifically target activity data (`activities`), compounds (`molecule_dictionary`), and potentially target information if needed. The extracted data will then be loaded into a Pandas DataFrame in Python, pre-processed, and analyzed. Since you haven't specified a particular analysis type, I'll focus on basic filtering, cleaning, and some initial exploration using RDKit for molecule representation and property calculation. The overall flow is:

1. **SQL Query:** Retrieve data (molecule structures and activity data) from ChEMBL.
2. **Data Export:** Save the SQL query result as CSV file.
3. **Data Loading:** Load CSV data into a Pandas DataFrame in Python.
4. **Data Cleaning:** Handle missing values, filter based on standard values, and ensure data types are correct.
5. **RDKit Integration:** Create RDKit molecule objects from SMILES strings.
6. **Exploratory Data Analysis (EDA):** Calculate molecular properties using RDKit, visualize data, and gain insights.

Addressing the Errors:

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

  This error arises because you're trying to use the regular expression operator ~ with a numeric column (`act.standard_value`). PostgreSQL requires explicit casting when using regular expressions on numeric columns. We will 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 indicates you have an older version of scikit-learn installed. The `squared=False` parameter (to get Root Mean Squared Error - RMSE) was introduced in later versions. I'll avoid using it and manually calculate the RMSE to ensure compatibility or suggest an upgrade if RMSE calculation is needed.

File Structure (AIMLops Template):

Assuming a basic structure like this:

```
Project_Root/
├── data/
│   └── chembl_data.csv  # SQL export saved here
├── notebooks/
│   └── Topic_CheMBL_35_18_1_data_extraction.ipynb
│   └── Topic_CheMBL_35_18_2_data_analysis.ipynb
```

```
├── src/            # (optional, for reusable code)
└── models/         # (optional, for saved models)
```

1. SQL Code (to run on pgAdmin and save as `chembl_data.csv` in the `data/` folder):

```sql
-- Topic_CheMBL_35_18.sql
SELECT
    md.chembl_id,
    md.canonical_smiles,
    act.standard_type,
    act.standard_value,
    act.standard_units
FROM
    activities act
JOIN
    molecule_dictionary md ON act.molregno = md.molregno
WHERE
    act.standard_type = 'IC50'  -- Example: Filter for IC50 values
    AND act.standard_units = 'nM' -- Example: Filter for nM units
    AND act.standard_value IS NOT NULL
    AND act.standard_value::TEXT ~ '^[0-9\.]+$' -- Corrected regex for numeric columns
LIMIT 100;
```

Explanation:

- `SELECT`: Retrieves the ChEMBL ID, SMILES string, standard type, standard value, and standard units.
- `FROM activities act JOIN molecule_dictionary md`: Joins the `activities` and `molecule_dictionary` tables based on the `molregno` (molecule registry number).
- `WHERE`: Filters the data based on standard type (e.g., IC50), standard units (e.g., nM), ensures standard value is not NULL, and uses a regular expression to ensure `standard_value` is a number. **Important:** The `act.standard_value::TEXT ~ '^[0-9\.]+$'` part casts the numeric `standard_value` to TEXT before applying the regex.
- `LIMIT 100`: Restricts the result set to 100 rows.

2. Python Code (Jupyter Notebook: `Topic_CheMBL_35_18_1_data_extraction.ipynb` - Data Extraction and Cleaning):

```python
# Topic_CheMBL_35_18_1_data_extraction.ipynb
import os
import pandas as pd
from rdkit import Chem
from rdkit.Chem import Descriptors

# Define the base path (important for AIMLops)
base_path = os.path.dirname(os.getcwd())  # Assumes notebook is in /notebooks, Project root
data_path = os.path.join(base_path, 'data', 'chembl_data.csv')
print(f"Data path: {data_path}")

# Load the data
try:
    df = pd.read_csv(data_path)
    print("Data loaded successfully.")
except FileNotFoundError:
    print(f"Error: File not found at {data_path}. Make sure you ran the SQL query and saved the data.")
    exit()

# Data Cleaning and Preprocessing
```

```
print("\nOriginal DataFrame:")
print(df.head())
print(df.info())

# Handle missing values (if any) - fill with mean, median or drop
df = df.dropna(subset=['canonical_smiles', 'standard_value']) #drop the NA rows
print("\nDataFrame after handling missing values:")
print(df.head())
print(df.info())

# Convert standard_value to numeric (important)
df['standard_value'] = pd.to_numeric(df['standard_value'], errors='coerce')
df = df.dropna(subset=['standard_value'])  # Remove rows where conversion failed
print("\nDataFrame after converting standard_value to numeric:")
print(df.head())
print(df.info())

# Filter for specific standard_type and standard_units (if needed, you did this in
SQL)
#df = df[(df['standard_type'] == 'IC50') & (df['standard_units'] == 'nM')]

# Basic Data Analysis (Descriptive Stats)
print("\nDescriptive Statistics of standard_value:")
print(df['standard_value'].describe())

# Create RDKit Mol objects
df['mol'] = df['canonical_smiles'].apply(lambda x: Chem.MolFromSmiles(x))
df = df[df['mol'].notna()] #drop invalid smiles
print("\nDataFrame with RDKit Mol objects:")
print(df.head())
print(df.info())

# Example: Calculate Molecular Weight
df['mol_wt'] = df['mol'].apply(lambda x: Descriptors.MolWt(x))
print("\nDataFrame with Molecular Weight:")
print(df.head())
print(df.info())

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

print("\nCompleted Data Extraction and Basic Processing")
```

Explanation:

- **Import Libraries:** Imports necessary libraries (os, pandas, RDKit).
- **Define Paths:** Uses `os.path.join` to construct file paths based on the `base_path`, adhering to your AIMLops structure. This makes the code portable.
- **Load Data:** Loads the CSV data into a Pandas DataFrame. Includes error handling if the file is not found.
- **Data Cleaning:**
    - Handles missing values (demonstrates dropping rows with missing `canonical_smiles` or `standard_value`).
    - Converts the `standard_value` column to numeric, handling potential errors by coercing invalid values to `NaN` and then removing those rows. This is *crucial* for further analysis.
- **RDKit Integration:**

    - Creates RDKit molecule objects from the SMILES strings using `Chem.MolFromSmiles()`.
    - Handles potential errors where SMILES strings are invalid (e.g., dropping rows where `Chem.MolFromSmiles()` returns `None`).
- **Example Molecular Property Calculation:** Calculates Molecular Weight using `Descriptors.MolWt()`.
- **Saving Processed Data:** Saves the cleaned and processed data to a new CSV file.

3. Python Code (Jupyter Notebook: `Topic_CheMBL_35_18_2_data_analysis.ipynb` - Data Analysis and Visualization):

```
# Topic_CheMBL_35_18_2_data_analysis.ipynb
import os
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from rdkit import Chem
from rdkit.Chem import Draw
from rdkit.Chem import Descriptors

# Define the base path
base_path = os.path.dirname(os.getcwd())
processed_data_path = os.path.join(base_path, 'data', 'chembl_data_processed.csv')

# Load the processed data
try:
    df = pd.read_csv(processed_data_path)
    print("Data loaded successfully.")
except FileNotFoundError:
    print(f"Error: File not found at {processed_data_path}.  Run the previous notebook
first.")
    exit()

# Basic EDA and Visualization
print("\nFirst 5 rows of the processed DataFrame:")
print(df.head())

# Distribution of standard_value
plt.figure(figsize=(10, 6))
sns.histplot(df['standard_value'], kde=True)
plt.title('Distribution of Standard Value (IC50 in nM)')
plt.xlabel('Standard Value (nM)')
plt.ylabel('Frequency')
plt.show()

# Scatter plot of Molecular Weight vs. Standard Value
plt.figure(figsize=(10, 6))
sns.scatterplot(x='mol_wt', y='standard_value', data=df)
plt.title('Molecular Weight vs. Standard Value')
plt.xlabel('Molecular Weight (Da)')
plt.ylabel('Standard Value (nM)')
plt.show()

# Convert SMILES to Mol objects (again, if needed, since it wasn't saved)
df['mol'] = df['canonical_smiles'].apply(lambda x: Chem.MolFromSmiles(x))
df = df[df['mol'].notna()]

# Visualize a few molecules
mols = df['mol'].tolist()[:4]  # First 4 molecules
img = Draw.MolsToGridImage(mols, molsPerRow=2, subImgSize=(200, 200))
```

```
img  # Display the image in the notebook

# More advanced analysis could go here, like:
# - Calculating more molecular properties
# - Building SAR models (QSAR)
# - Clustering compounds based on properties
# - Target prediction

print("\nCompleted Data Analysis and Visualization")
```

Explanation:

- **Loading Processed Data:** Loads the processed data from the CSV file saved in the previous notebook.
- **Basic EDA:**
  - Prints the first few rows of the DataFrame.
  - Plots the distribution of `standard_value` using a histogram.
  - Creates a scatter plot of `mol_wt` vs. `standard_value`.
- **Visualization:**
  - Converts SMILES strings back to RDKit Mol objects (if not already in the saved data).
  - Visualizes the first few molecules using `Draw.MolsToGridImage()`.
- **Placeholder for Advanced Analysis:** Includes comments indicating where you could add more advanced analyses, such as QSAR modeling, clustering, or target prediction.

4. Example Scenarios / Uses Cases:

Here are five examples of how you can use this code and the data:

1. **Identify Potent Compounds:** Filter the DataFrame to identify compounds with low `standard_value` (e.g., `df[df['standard_value'] < 100]`). These are the most potent compounds in your dataset.
2. **Explore Structure-Activity Relationships (SAR):** Group compounds by structural similarity (e.g., using RDKit's MCS - Maximum Common Substructure) and compare their activity profiles.
3. **Develop a QSAR Model:** Use molecular descriptors (calculated with RDKit) as features to predict the `standard_value` of new compounds. This requires splitting the data into training and test sets.
4. **Visualize the Chemical Space:** Use dimensionality reduction techniques (e.g., PCA, t-SNE) to project the compounds onto a 2D or 3D space based on their molecular properties. This allows you to visualize the diversity of your compound library.
5. **Calculate Druglikeness:** Use RDKit to calculate properties like Lipinski's Rule of Five and assess the druglikeness of the compounds. (`Descriptors.MolLogP(mol)`, `Descriptors.NumHAcceptors(mol)`, `Descriptors.NumHDonors(mol)`, `Descriptors.MolWt(mol)`)

Key Considerations:

- **Database Credentials:** Make sure to keep your database username and password secure. Consider using environment variables to store them instead of hardcoding them in the script.
- **Error Handling:** Add more robust error handling to your code. For example, you could use `try...except` blocks to catch exceptions during RDKit molecule creation or property calculation.
- **Data Validation:** Always validate your data after loading it to ensure that it is in the expected format and range.
- **Reproducibility:** Use a consistent environment (e.g., a Conda environment or Docker container) to ensure that your code is reproducible.

In summary (en français):

Vous travaillez avec les données ChEMBL 35 en utilisant RDKit dans un cadre AIMLops. L'objectif est d'extraire des données de la base de données, de les nettoyer et de les analyser. J'ai fourni du code SQL et Python pour accomplir cela, en tenant compte des erreurs spécifiques que vous avez rencontrées.

- **SQL :** Extrait des données de la base de données ChEMBL, en filtrant pour obtenir 100 lignes et en gérant le problème de typage des données pour la comparaison avec les expressions régulières.
- **Python :**
    - Charge les données extraites.
    - Nettoie les données (gestion des valeurs manquantes, conversion des types).
    - Intègre RDKit pour créer des objets moléculaires à partir des chaînes SMILES.
    - Calcule des propriétés moléculaires de base (par exemple, le poids moléculaire).
    - Visualise les données (distribution des valeurs d'activité, nuages de points).
- **Exemples d'utilisation :** Identification de composés puissants, exploration des relations structure-activité (SAR), développement de modèles QSAR, visualisation de l'espace chimique, calcul de la druglikeness.

N'oubliez pas de sécuriser vos informations d'identification de base de données, de valider vos données et d'utiliser un environnement cohérent pour assurer la reproductibilité de votre code. Bonne analyse !