PharmApp Suite

🧠 AI for Drug Discovery and Development 💉

🧩 Topic: Merging ChEMBL with PubChem CID data

Okay, I understand. You're working on a ChEMBL 35 data analysis project using RDKit, following an AIMLops-inspired directory structure. You're running PostgreSQL (psql) on a remote server (192.168.206.136) and want guidance on executing your code within the defined framework, specifically addressing the errors you've encountered. I'll provide a clear analysis, SQL and Python code snippets, and examples, focusing on extracting a manageable 100-row sample.

Let's break down the problem and create the solution.

1. Analysis of the Analysis Model

Your analysis likely involves querying the ChEMBL database, extracting relevant information (e.g., activity data), cleaning/transforming the data, and potentially building predictive models (though you haven't explicitly stated this). Key aspects include:

- **Data Extraction:** This involves SQL queries to select specific tables and columns from the ChEMBL database (e.g., `activities`, `molecule_dictionary`, `assays`).
- **Filtering:** You need to filter the data based on relevant criteria. This includes filtering based on activity type (e.g., IC50), target, assay type, and potency.
- **Data Cleaning and Transformation:** This includes handling missing values, converting units, and standardizing data formats. The error you encountered suggests issues with data type inconsistencies (e.g., trying to use a regular expression on a numeric column).
- **RDKit Integration:** This involves using RDKit to manipulate molecule structures (e.g., generate fingerprints, calculate descriptors).
- **Modeling (Optional):** If you're building a predictive model, you'll need to select appropriate algorithms, train the model, and evaluate its performance. The scikit-learn error indicates you need to handle version compatibility.

2. Addressing the Errors

- **SQL Error:** `"ERROR: operator does not exist: numeric ~ unknown, LINE 12: AND act.standard_value ~ '^[0-9\.]+$'"`
  - **Cause:** This error occurs because you're trying to use a regular expression operator (~) on a column (`act.standard_value`) that PostgreSQL has identified as a numeric type (likely `integer` or `numeric`). Regular expressions are for text-based data.
  - **Solution:** You need to either:
    1. Cast the column to text: `CAST(act.standard_value AS TEXT) ~ '^[0-9\.]+$'`
    2. Use numeric comparison operators: `act.standard_value >= 0` (if you're just checking for non-negative values)
- **Python/Scikit-learn Error:** `"old scikit-learn version does not support parameters squared=False in the mean_squared_error function"`
  - **Cause:** You're using an older version of scikit-learn that doesn't support the `squared=False` parameter in `mean_squared_error`. This parameter controls whether the function returns the Mean Squared Error (MSE) or the Root Mean Squared Error (RMSE).
  - **Solution:**
    1. **Upgrade Scikit-learn (Recommended):** `pip install -U scikit-learn`

2. **Calculate RMSE Manually (Workaround):** If upgrading isn't an option, calculate the RMSE by taking the square root of the MSE.

3. SQL Code (Topic_CheMBL_35_82.sql)

```sql
-- SQL query to extract data and save as CSV
-- Get 100 random activities related to specific target
-- Modify the WHERE clause based on your topic of interest

SELECT
    md.chembl_id,
    act.standard_type,
    act.standard_value,
    act.standard_units,
    act.pchembl_value
FROM
    activities act
JOIN
    molecule_dictionary md ON act.molregno = md.molregno
JOIN
    target_dictionary td ON act.tid = td.tid
WHERE
    td.target_name LIKE '%Kinase%' -- Example: Filter by target name
    AND act.standard_type = 'IC50'   -- Example: Filter by activity type
    AND act.standard_units = 'nM'     -- Example: Filter by activity units
    AND CAST(act.standard_value AS TEXT) ~ '^[0-9\.]+$'  -- Filter numeric values, fix
SQL error here. Cast to text for regex or use numeric comparison.
ORDER BY random()
LIMIT 100;

-- Save the result of the above query to a CSV file
-- In pgAdmin, right-click on the query result and select "Copy with Headers"
-- Paste the data into a text file and save it as Topic_CheMBL_35_82.csv in the
../data/ folder.
```

Important: Execute this SQL code in pgAdmin, then *manually* copy the results (including headers) and paste them into a file named `Topic_CheMBL_35_82.csv` located in your `../data/` directory. pgAdmin doesn't directly save to a file; you must copy and paste.

4. Python Code (Topic_CheMBL_35_82_1_Data_Loading.ipynb)

```python
# Topic_CheMBL_35_82_1_Data_Loading.ipynb

import os
import pandas as pd

# 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")
csv_file = os.path.join(data_path, "Topic_CheMBL_35_82.csv")

print(f"Base path: {base_path}")
print(f"Data path: {data_path}")
print(f"CSV file path: {csv_file}")

# Load the data using pandas
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 saved the CSV from
pgAdmin to the correct location.")
except Exception as e:
    print(f"An error occurred while loading the data: {e}")
```

5. Python Code (Topic_CheMBL_35_82_2_Data_Analysis.ipynb)

```
# Topic_CheMBL_35_82_2_Data_Analysis.ipynb

import os
import pandas as pd
from rdkit import Chem
from rdkit.Chem import AllChem
import numpy as np
#from sklearn.model_selection import train_test_split # Remove if not modeling
#from sklearn.linear_model import LinearRegression # Remove if not modeling
#from sklearn.metrics import mean_squared_error # Remove if not modeling

# Define the base path and data path
base_path = os.path.abspath(os.path.join(os.getcwd(), ".."))
data_path = os.path.join(base_path, "data")
csv_file = os.path.join(data_path, "Topic_CheMBL_35_82.csv")

# Load the data
try:
    df = pd.read_csv(csv_file)
except FileNotFoundError:
    print(f"Error: File not found at {csv_file}.  Make sure you saved the CSV from
pgAdmin to the correct location.")
    exit() # Stop execution
except Exception as e:
    print(f"An error occurred while loading the data: {e}")
    exit() # Stop execution

# Display data info
print("Dataframe Info:")
print(df.info())

# Handle missing pchembl_value (replace NaN with 0 for demonstration)
df['pchembl_value'] = df['pchembl_value'].fillna(0) # Or remove rows with NaN

# Example: Create RDKit Mol objects
def create_mol(chembl_id):
  """Dummy function to simulate fetching molecules using chembl_id"""
  # Normally, you'd query ChEMBL API here
  # For the sake of demonstration, we create a simple molecule
  smiles = 'CC(=O)Oc1ccccc1C(=O)O' # Aspirin SMILES
  mol = Chem.MolFromSmiles(smiles)
  return mol

df['ROMol'] = [create_mol(chembl_id) for chembl_id in df['chembl_id']]
df = df[df['ROMol'].notna()]

# Example: Calculate Morgan Fingerprints
def calculate_morgan_fingerprint(mol):
    try:
        info = {}
        fp = AllChem.GetMorganFingerprint(mol, 2, bitInfo=info) # Radius of 2
        return fp
```

```
    except:
        return None

df['MorganFP'] = df['ROMol'].apply(calculate_morgan_fingerprint)
df = df[df['MorganFP'].notna()]

print(df.head())

# Example of regression calculation (if you have proper features and target variables)
# Make sure there are at least 2 unique non-NaN values in the target column to split
it
#if df['pchembl_value'].nunique() > 1:
#   X = df['MorganFP'].tolist() # Example Feature (Morgan fingerprints)
#   y = df['pchembl_value'].tolist()  # Target variable
#
#   X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
#
#   # Model Building
#   model = LinearRegression()
#   model.fit(X_train, y_train)
#   y_pred = model.predict(X_test)
#
#   # Calculate Mean Squared Error (MSE) or Root Mean Squared Error (RMSE)
#   mse = mean_squared_error(y_test, y_pred)
#   rmse = np.sqrt(mse) # Calculate RMSE if MSE is not available

# Example Output
print("Completed data analysis")
```

Explanation and Key Points:

- **AIMLops Compliance:** The `os.path.join(base_path, ...)` calls ensure your code works correctly regardless of the current working directory, which is crucial for a robust AIMLops setup. The notebook names follow your specified pattern.
- **Error Handling:** The `try...except` blocks are essential for catching potential errors, like the CSV file not being found. This makes your code more resilient.
- **SQL Error Correction:** The `CAST(act.standard_value AS TEXT) ~ '^[0-9\.]+$'` line in the SQL query addresses the `numeric ~ unknown` error by explicitly casting the `standard_value` column to text before applying the regular expression. Alternatively, you can use numeric comparisons (e.g., `act.standard_value >= 0`).
- **Scikit-learn Error:** If you cannot upgrade scikit-learn, uncomment the calculation of RMSE manually and replace the function parameter squared = False.
- **RDKit Integration:** The example code demonstrates how to load molecules using RDKit and calculate Morgan fingerprints. **Important:** The `create_mol` function is a placeholder. You'll need to replace it with code that actually fetches molecules from the ChEMBL database based on the `chembl_id`. This might involve using the ChEMBL API. If you cannot access the ChEMBL database, you can replace the dummy molecule "Aspirin" with other known molecules.
- **Fingerprint generation** is included to show that the code is capable of processing data with rdkit
- **Data Filtering** Filtering is included to ensure that molecules are valid and usable
- **Regression Example:** The commented-out regression code shows how you *could* build a simple model. However, **you will need to adapt this based on your actual features and target variable.** The code also includes an example of how to calculate the RMSE manually if your scikit-learn version is old. Remember to uncomment the import statements (`from sklearn.model_selection...`) if you use the regression example.

- **CSV Loading:** Pay very close attention to the `csv_file` path. Double-check that the CSV file is actually located in that directory.
- **Install RDKit:** `conda install -c conda-forge rdkit`
- **SQL Manual Copy:** You **must** copy the data from pgAdmin and save it as a CSV manually. pgAdmin doesn't have a built-in "save to CSV" feature like some other database tools.
- **Install pandas**`pip install pandas`
- **Install scikit-learn**`pip install scikit-learn`

6. Examples

Here are five examples of how you might adapt the SQL query and Python code to focus on different research areas within ChEMBL. Remember to modify the `WHERE` clause in the SQL query and adapt the Python code accordingly.

1. **Example 1: Specific Target Family (Kinases)**
    - **SQL:**

```
WHERE
    td.target_name LIKE '%Kinase%'
    AND act.standard_type = 'IC50'
    AND act.standard_units = 'nM'
    AND CAST(act.standard_value AS TEXT) ~ '^[0-9\.]+$'
```

    - **Python:** Analyze the activities of compounds against kinases. You might calculate descriptors specific to kinase inhibitors.
2. **Example 2: Specific Protein Target (e.g., EGFR)**
    - **SQL:**

```
WHERE
    td.target_name LIKE '%EGFR%'
    AND act.standard_type = 'IC50'
    AND act.standard_units = 'nM'
    AND CAST(act.standard_value AS TEXT) ~ '^[0-9\.]+$'
```

    - **Python:** Focus on compounds targeting EGFR (Epidermal Growth Factor Receptor). You could explore structure-activity relationships (SAR) around known EGFR inhibitors.
3. **Example 3: Specific Activity Type (e.g., Ki)**
    - **SQL:**

```
WHERE
    act.standard_type = 'Ki'
    AND act.standard_units = 'nM'
    AND CAST(act.standard_value AS TEXT) ~ '^[0-9\.]+$'
```

    - **Python:** Analyze compounds with Ki (inhibition constant) data. You might compare Ki values to IC50 values for the same compounds.
4. **Example 4: Specific Assay Type (e.g., Binding Assay)**
    - **SQL:**

```
JOIN assays a ON act.assay_id = a.assay_id
WHERE
    a.assay_type = 'B' -- 'B' likely represents binding assay
    AND act.standard_type = 'IC50'
    AND act.standard_units = 'nM'
    AND CAST(act.standard_value AS TEXT) ~ '^[0-9\.]+$'
```

- **Python:** Analyze data from binding assays. You could investigate the correlation between binding affinity and cellular activity.

5. **Example 5: Focus on High-Potency Compounds (IC50 < 100 nM)**

   - **SQL:**

```
WHERE
    act.standard_type = 'IC50'
    AND act.standard_units = 'nM'
    AND act.standard_value < 100
```

   - **Python:** Identify and analyze the structures of highly potent compounds. You could look for common structural features or pharmacophores.

Remember to adapt the Python code within the notebooks to perform the specific analysis relevant to each example.

Important Considerations:

- **ChEMBL API:** For more robust and scalable data retrieval, consider using the ChEMBL API instead of directly querying the database. This is especially important if you need to access more than just a small subset of the data.
- **Data Validation:** Always validate the data you extract from ChEMBL. Check for inconsistencies, errors, and potential biases.
- **Units Conversion:** Be very careful with units. Ensure that all activity values are in the same units before performing any calculations.
- **Reproducibility:** Document your code and analysis thoroughly so that your results can be easily reproduced.

Let me know if you have any more questions or need help refining your code.