PharmApp Suite

🧠 AI for Drug Discovery and Development 🧪

🧩 Topic: Interactive similarity search app

Okay, I understand. You want me to guide you through analyzing ChEMBL 35 data using RDKit and PostgreSQL, following your defined project structure and addressing specific errors. Let's break this down step-by-step.

Project Analysis and Model Overview

The core of your project revolves around extracting, transforming, and analyzing chemical data from the ChEMBL 35 database. Here's a general analysis model:

1. **Data Extraction (SQL):** You'll use SQL queries to retrieve relevant information from the ChEMBL 35 database. This includes activity data (IC50, Ki, etc.), compound structures (SMILES strings), and potentially target information. You'll save this data as CSV files within your `../data/` directory.
2. **Data Transformation (Python/RDKit):** Using RDKit, you'll transform the SMILES strings into molecular representations. This involves:
    - **Parsing SMILES:** Converting SMILES strings to RDKit molecule objects.
    - **Calculating Descriptors:** Generating a set of molecular descriptors. These are numerical features that characterize the molecule's properties (e.g., molecular weight, logP, number of hydrogen bond donors/acceptors, topological polar surface area, etc.). You'll choose a relevant set of descriptors based on your research question.
    - **Handling Activity Data:** You might need to convert activity values (IC50, Ki) to a pChEMBL scale (-log10(activity)) for more convenient analysis. You also want to filter for values that are numeric and above 0.
3. **Data Analysis (Python/Scikit-learn, etc.):** Once you have your molecular descriptors and activity data, you can perform various analyses:
    - **Regression Modeling:** Predict activity based on molecular descriptors. Common algorithms include:
        - Linear Regression
        - Random Forest
        - Support Vector Regression (SVR)
        - Neural Networks
    - **Clustering:** Group molecules with similar properties based on their descriptors.
    - **Classification:** Predict whether a molecule is active or inactive based on a threshold.
4. **Model Evaluation:** You'll need to evaluate the performance of your models using appropriate metrics (e.g., R-squared, RMSE, MAE for regression; accuracy, precision, recall, F1-score for classification). Use cross-validation techniques to get robust performance estimates.

Addressing Your Errors

- **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 ~ (which is for string matching) on a numeric column (`act.standard_value`). PostgreSQL probably stores `standard_value` as a numeric type (e.g., `numeric`, `float`, `integer`).

Solution: You need to cast the numeric value to text before applying the regular expression. You also need to change the regular expression to use proper numeric comparison operators instead of regular expressions for filtering numeric values.

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

  This indicates you are using an older version of scikit-learn.

  Solution: There are a couple of options:

  - **Upgrade scikit-learn:** This is the preferred solution. Use `pip install --upgrade scikit-learn` in your terminal or Jupyter Notebook.
  - **Remove `squared=False`:** If you can't upgrade, remove the `squared=False` argument. The function will return the Mean Squared Error (MSE) instead of the Root Mean Squared Error (RMSE). You can then take the square root of the MSE to get the RMSE.

Code Examples (SQL and Python)

Here's a breakdown with SQL and Python examples, tailored to your requirements and addressing the errors. I'll use `Topic_CheMBL_35_73` as the base for file names. Remember to adjust the SQL query based on *your specific research question* and the columns available in your ChEMBL 35 database.

1. SQL Code (Extracting Data - `Topic_CheMBL_35_73_1_extract_data.sql`)

```sql
-- Connect to the chembl_35 database using psql or pgAdmin

-- Extract data for a specific target (replace 'CHEMBL205' with a relevant target ID)
SELECT
    cmp.chembl_id,  -- Compound ChEMBL ID
    md.molregno,     -- Molecule Registry Number
    cs.canonical_smiles, -- Canonical SMILES string
    act.standard_type, -- Activity type (e.g., IC50, Ki)
    act.standard_value, -- Activity value
    act.standard_units   -- Activity units (e.g., nM)
FROM
    compound_structures cs
JOIN
    molecule_dictionary md ON cs.molregno = md.molregno
JOIN
    activities act ON md.molregno = act.molregno
JOIN
    assays ass ON act.assay_id = ass.assay_id
JOIN
    target_dictionary td ON ass.tid = td.tid
JOIN
    component_sequences tcs ON td.tid = tcs.tid
WHERE
    td.chembl_id = 'CHEMBL205' -- Replace with the target you are interested in
    AND act.standard_type IN ('IC50', 'Ki')  -- Limit to specific activity types
    AND act.standard_relation = '=' -- Ensure the relationship is direct '='
    AND act.standard_value IS NOT NULL  -- Avoid NULL activity values
    AND act.standard_value > 0        -- Keep positive values
    AND cs.canonical_smiles IS NOT NULL -- Ensure we have SMILES strings
    AND act.standard_units = 'nM'
ORDER BY
    act.standard_value ASC
LIMIT 100;

-- Save the results to a CSV file using pgAdmin's export feature or the \copy command
```

```
in psql:
-- \copy (SELECT ...) TO 'path/to/your/data/Topic_CheMBL_35_73_data.csv' WITH CSV
HEADER;

--Important: Replace 'path/to/your/data/' with the actual path to your data directory
```

Explanation:

- **Target Selection:** The `WHERE td.chembl_id = 'CHEMBL205'` clause is *crucial*. Replace `'CHEMBL205'` with the actual ChEMBL ID of the target you want to study. Without this, you'll get data from all targets, which is not what you want for a targeted analysis. You can find target IDs in the ChEMBL database.
- **Activity Types:** The query filters for `IC50` and `Ki` values. Adjust this based on the activity types you are interested in.
- **Data Validation:** The query checks for `NULL` values and ensures that `standard_value` is greater than 0 and the relationship is `=`, because you cant take the logarithm of a non-positive number.
- **LIMIT 100:** This limits the output to 100 rows, as requested.
- **Saving to CSV:** The commented-out `\copy` command shows how to save the results to a CSV file directly from `psql`. In pgAdmin, you can right-click on the query results and choose "Save as CSV." Remember to replace `'path/to/your/data/'` with the actual path to your data directory.

How to Run:

1. Open pgAdmin and connect to your `chembl_35` database.
2. Open the `Topic_CheMBL_35_73_1_extract_data.sql` file in the query editor.
3. **Crucially:** Edit the `td.chembl_id` value to the target you are interested in analyzing.
4. Execute the query.
5. Save the results as `Topic_CheMBL_35_73_data.csv` in your `../data/` directory. Use pgAdmin's export functionality or the `\copy` command.

2. Python Code (Data Processing and Analysis - `Topic_CheMBL_35_73_2_analyze_data.ipynb`)

```python
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, r2_score
from sklearn.preprocessing import StandardScaler  # Import StandardScaler
import math

# Define base path
base_path = "../data"  # Corrected path

# Load the data
data_file = os.path.join(base_path, "Topic_CheMBL_35_73_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 script
and saved the data to the correct location.")
    exit()

# Data Cleaning and Preprocessing
```

```
df = df.dropna(subset=['canonical_smiles', 'standard_value']) # Drop rows with missing
SMILES or activity
df = df[df['standard_value'] > 0] # remove standard_value <= 0

# Convert activity to pChEMBL (negative log scale)
df['pChEMBL'] = -np.log10(df['standard_value'] / 1e9)  # Convert nM to Molar, then
-log10

# RDKit Descriptor Calculation
def calculate_descriptors(smiles):
    try:
        mol = Chem.MolFromSmiles(smiles)
        if mol is None:
            return None  # Handle invalid SMILES
        descriptors = {}
        descriptors['MolWt'] = Descriptors.MolWt(mol)
        descriptors['LogP'] = Descriptors.MolLogP(mol)
        descriptors['HBD'] = Descriptors.NumHDonors(mol)
        descriptors['HBA'] = Descriptors.NumHAcceptors(mol)
        descriptors['TPSA'] = Descriptors.TPSA(mol)
        return descriptors
    except:
        return None

# Apply descriptor calculation to each molecule
df['descriptors'] = df['canonical_smiles'].apply(calculate_descriptors)

# Handle rows where descriptor calculation failed
df = df.dropna(subset=['descriptors'])

# Convert descriptor dictionaries to DataFrame columns
df = pd.concat([df.drop(['descriptors'], axis=1), df['descriptors'].apply(pd.Series)],
axis=1)

# Data Splitting
X = df[['MolWt', 'LogP', 'HBD', 'HBA', 'TPSA']]
y = df['pChEMBL']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Feature Scaling
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Model Training
model = LinearRegression()
model.fit(X_train, y_train)

# Model Evaluation
y_pred = model.predict(X_test)

# Check scikit-learn version
import sklearn
sklearn_version = sklearn.__version__

# Convert version string to tuple of integers
version_tuple = tuple(map(int, sklearn_version.split('.')))
```

```
if version_tuple < (1, 2):
    mse = mean_squared_error(y_test, y_pred)
    rmse = math.sqrt(mse)
else:
    rmse = mean_squared_error(y_test, y_pred, squared=False) # Recommended, if
possible

r2 = r2_score(y_test, y_pred)

print(f"RMSE: {rmse}")
print(f"R-squared: {r2}")

# Displaying the results
print("\nFirst 5 rows of the processed DataFrame:")
print(df.head())
```

Explanation:

- **Path Handling:** Uses `os.path.join` for robust path construction, as requested.
- **Error Handling:** The code now includes `try...except` blocks to catch `FileNotFoundError` when loading the data. This makes the code more robust. Also, it handles invalid SMILES strings with `mol is None`.
- **pChEMBL Calculation:** Converts `standard_value` (assumed to be in nM) to pChEMBL values. **Important:** Adjust the units conversion (`/ 1e9`) if your data is in different units.
- **Descriptor Calculation:** Calculates a set of commonly used molecular descriptors using RDKit. You can customize these based on your needs. The `try...except` block handles potential errors during SMILES parsing or descriptor calculation.
- **Data Cleaning:** Removes rows with missing SMILES strings or activity values.
- **Data Splitting:** Splits the data into training and testing sets.
- **Feature Scaling:** Applies `StandardScaler` to standardize the features (important for many machine learning algorithms).
- **Model Training:** Trains a simple Linear Regression model. You can replace this with more sophisticated models.
- **Model Evaluation:** Calculates RMSE and R-squared to evaluate the model's performance. It addresses the `squared=False` error by checking scikit-learn version.
- **Printing Results:** Prints the RMSE and R-squared values, and the first 5 rows of the processed DataFrame.

How to Run:

1. Make sure you have RDKit, pandas, scikit-learn, and numpy installed: `pip install rdkit pandas scikit-learn numpy`
2. Open the `Topic_CheMBL_35_73_2_analyze_data.ipynb` file in Jupyter Notebook.
3. **Edit `base_path`:** Make sure the `base_path` variable is correctly set to `../data`.
4. Run the notebook cells sequentially.

3. Example Uses (Adjust the Code for these scenarios)

Here are 5 examples of how you can adapt this code for different analyses:

1. **Different Target:** Change the `td.chembl_id` in the SQL query to analyze activity data for a different protein target.
2. **Different Activity Type:** Modify the `act.standard_type IN ('IC50', 'Ki')` in the SQL query to include or exclude specific activity types (e.g., EC50, Kd).

3. **Different Descriptors:** Add or remove descriptors in the `calculate_descriptors` function. RDKit offers a wide range of descriptors. Be mindful of multicollinearity (highly correlated descriptors).
4. **Different Model:** Replace the `LinearRegression` model with a more complex model like `RandomForestRegressor` or `SVR` from scikit-learn. Remember to tune the hyperparameters of the model using techniques like cross-validation.
5. **Classification:** Convert the problem to a classification task by defining an activity threshold. For example, you could classify compounds as "active" (pChEMBL > 6) or "inactive" (pChEMBL <= 6). Then, use classification algorithms like `LogisticRegression` or `RandomForestClassifier`. Modify the evaluation metrics accordingly (accuracy, precision, recall, F1-score).

Important Considerations:

- **Data Quality:** ChEMBL data can be noisy. Careful data cleaning and validation are essential. Look for outliers, inconsistencies in units, and potential errors in the SMILES strings.
- **Feature Selection:** Choosing the right molecular descriptors is crucial for building accurate models. Consider using feature selection techniques to identify the most relevant descriptors.
- **Model Validation:** Use appropriate cross-validation techniques to assess the generalization performance of your models.
- **Domain Applicability:** Be aware that models are only valid within the chemical space they were trained on. Extrapolating to very different molecules can lead to inaccurate predictions.
- **Regular updates:** ChEMBL database is updated regularly. Remember to update your local copy to stay current.

I hope this detailed guide helps you with your project! Good luck!