PharmApp Suite

🧠 AI for Drug Discovery and Development 🧪

🧩 Topic: Linking ChEMBL with DrugBank for repurposing

Okay, I understand. You're working with ChEMBL 35 data to drive drug discovery, and you need guidance on running your analysis within an AIMLops framework. Let's break down the process, focusing on your specific needs and potential errors.

1. Analysis Model and Workflow

The core idea is to leverage ChEMBL 35 data (specifically activity data) and RDKit to extract meaningful information for drug research. Here's a breakdown of a potential analysis model and workflow:

- **Data Extraction & Preparation:**
  - Use SQL to query ChEMBL 35 and extract relevant activity data (e.g., IC50, Ki, EC50) for specific targets or compounds.
  - Filter the data based on standard values and units to ensure consistency.
  - Save the extracted data as CSV files for further processing.
- **Molecule Representation (RDKit):**
  - Read the CSV data in your Jupyter Notebook.
  - Use RDKit to convert SMILES strings (from ChEMBL) into molecular objects.
  - Calculate molecular descriptors (e.g., molecular weight, LogP, number of hydrogen bond donors/acceptors, topological polar surface area - TPSA). These are numerical representations of the molecule's properties.
- **Data Analysis & Modeling:**
  - Explore the data distribution of both activity and calculated descriptors (histograms, scatter plots).
  - Correlate descriptors with activity values.
  - Build predictive models (e.g., linear regression, random forest, SVM) to predict activity based on molecular descriptors.
  - Evaluate model performance using appropriate metrics (e.g., R-squared, RMSE, MAE).
- **Interpretation & Iteration:**
  - Analyze the model to identify key descriptors that influence activity.
  - Use this information to guide the design of new compounds with improved activity.
  - Repeat the process (extract, represent, analyze, interpret) to refine your understanding and models.

2. Code Implementation (SQL & Python)

Let's provide the code you need. We'll address the error messages you mentioned.

SQL (To be run in pgAdmin - save as a .sql file, then run. After running the script, export the output to a CSV file.)

```sql
-- Topic_CheMBL_35_19.sql

-- Extract activity data for a specific target (replace 'CHEMBL205' with your target
of interest)
-- Limiting to 100 rows for demonstration purposes.
SELECT
```

```
    act.molregno,
    act.standard_type,
    act.standard_value,
    act.standard_units,
    act.assay_id,
    md.chembl_id AS compound_chembl_id,
    cs.canonical_smiles
FROM
    activities act
JOIN
    molecule_dictionary md ON act.molregno = md.molregno
JOIN
    compound_structures cs ON md.molregno = cs.molregno
WHERE
    act.target_chembl_id = 'CHEMBL205'  -- Replace with your desired target CHEMBL ID
    AND act.standard_type IN ('IC50', 'Ki', 'EC50') -- Filter for common activity
types
    AND act.standard_units = 'nM' -- Standardize units (adjust as needed)
    AND act.standard_value IS NOT NULL
    AND act.standard_value::TEXT ~ '^[0-9.]+$' -- Corrected regex for numeric check
ORDER BY
    act.standard_value ASC
LIMIT 100;

-- Save the results to a CSV file (e.g., chembl_activity_data.csv)
-- Use pgAdmin's export functionality or \copy command.
```

- **Explanation:**
  - The query retrieves activity data (standard_type, standard_value, standard_units), compound ChEMBL ID, and SMILES string for a specific target.
  - It filters based on common activity types (IC50, Ki, EC50) and standardizes the units to ‘nM’ (nanomolar). Adjust these filters based on your research question.
  - `AND act.standard_value::TEXT ~ '^[0-9.]+$'` This is the crucial fix for your first error. It ensures that `standard_value` only contains numeric values. We cast the numeric column to text, and then check using regex.
  - `LIMIT 100` restricts the output to 100 rows.
- **Important:** Replace `'CHEMBL205'` with the actual ChEMBL ID of the target you are interested in. You can find these IDs in the ChEMBL database.

Python (Jupyter Notebook - “Topic_CheMBL_35_19_1_data_analysis.ipynb”)

```
# Topic_CheMBL_35_19_1_data_analysis.ipynb

import pandas as pd
from rdkit import Chem
from rdkit.Chem import Descriptors
import numpy as np
import os
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
import warnings
warnings.filterwarnings("ignore") # Suppress warnings

# Define the base path for your project
base_path = "../data"  # Assuming your data folder is one level up

# Construct the full path to your CSV file
csv_file_path = os.path.join(base_path, "chembl_activity_data.csv") # Make sure file
```

```
name matches what you saved the .csv file as!

# Load the data from the CSV file
try:
    df = pd.read_csv(csv_file_path)
except FileNotFoundError:
    print(f"Error: File not found at {csv_file_path}.  Make sure the file exists and
the path is correct.")
    exit()

# Print the first few rows of the DataFrame to verify the data
print(df.head())

# Function to calculate molecular descriptors using RDKit
def calculate_descriptors(smiles):
    mol = Chem.MolFromSmiles(smiles)
    if mol is None:
        return None  # Handle invalid SMILES strings

    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)  # Topological Polar Surface Area
    return descriptors

# Apply the descriptor calculation to each SMILES string in the DataFrame
df['descriptors'] = df['canonical_smiles'].apply(calculate_descriptors)

# Handle cases where descriptor calculation failed (e.g., invalid SMILES)
df = df.dropna(subset=['descriptors'])

# Convert the 'descriptors' column to a DataFrame
descriptors_df = pd.DataFrame(df['descriptors'].tolist())

# Concatenate the descriptors DataFrame with the original DataFrame
df = pd.concat([df, descriptors_df], axis=1)

# Convert standard_value to numeric and handle errors
df['standard_value'] = pd.to_numeric(df['standard_value'], errors='coerce')
df = df.dropna(subset=['standard_value'])

# Prepare data for modeling (example: using LogP and MolWt to predict pIC50)
# Convert standard_value to pIC50 (optional but often useful)
df['pIC50'] = -np.log10(df['standard_value'] / 1e9)  # Convert nM to Molar, then
-log10

# Select features and target
X = df[['LogP', 'MolWt']] # You can add more descriptors here
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 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}")

# Print the coefficients of the linear regression model
print(f"Coefficients: {model.coef_}")
print(f"Intercept: {model.intercept_}")

# Example: Print some predictions
print("\nExample Predictions:")
for i in range(min(5, len(y_test))):  # Print up to 5 examples
    print(f"Actual pIC50: {y_test.iloc[i]:.2f}, Predicted pIC50: {y_pred[i]:.2f}")
```

- **Explanation:**
    - **Import Libraries:** Imports necessary libraries.
    - **File Handling:** Uses `os.path.join` to construct the correct file path to your CSV file, adhering to your AIMLops structure. Includes error handling if the file isn't found.
    - **Data Loading:** Loads the CSV data into a pandas DataFrame.
    - **Descriptor Calculation:** Defines a function `calculate_descriptors` that takes a SMILES string as input and calculates a set of common molecular descriptors using RDKit. It includes error handling for invalid SMILES strings.
    - **Descriptor Application:** Applies the `calculate_descriptors` function to the `canonical_smiles` column of the DataFrame.
    - **Data Cleaning:** Handles missing descriptor values (due to potentially invalid SMILES strings).
    - **Data Preparation:** Prepares the data for modeling by selecting features (molecular descriptors) and the target variable (pIC50). **You'll likely want to add more descriptors to the `X` variable.**
    - **Train/Test Split:** Splits the data into training and testing sets.
    - **Model Training:** Trains a linear regression model.
    - **Model Evaluation:** Evaluates the model using Mean Squared Error (MSE) and R-squared.
    - **Example Predictions:** Prints a few example predictions to illustrate the model's performance.
- **Key Improvements & Error Handling:**
    - **`os.path.join`:** Correctly uses `os.path.join` for path construction, ensuring portability.
    - **Error Handling (File Not Found):** Includes a `try...except` block to handle the case where the CSV file is not found at the specified path. This prevents the script from crashing.
    - **Error Handling (Invalid SMILES):** The `calculate_descriptors` function now returns `None` if it encounters an invalid SMILES string. The code then removes rows with missing descriptors. This is crucial for robustness.
    - **`warnings.filterwarnings("ignore")`:** Suppresses warnings that might arise from certain operations. This is mostly for cleaner output, but you should still be aware of the warnings you're suppressing.

    - **`pd.to_numeric(errors='coerce')`:** This is important! It converts the `standard_value` column to a numeric type. `errors='coerce'` tells pandas to replace any values that can't be converted to numbers with `NaN` (Not a Number). This allows you to handle non-numeric values in your data. Then, we drop the rows with NaN in the standard_value.
    - **pIC50 Calculation:** Includes a common conversion of the `standard_value` (assumed to be IC50 in nM) to pIC50. This is a more common and often more useful representation of activity. Make sure your `standard_value` is in nM before using this calculation. Adjust as needed.
- **Addressing the second error, old scikit-learn version does not support parameters squared=False in the mean_squared_error function**: The code uses `mean_squared_error` without the `squared=False` parameter, which is compatible with older scikit-learn versions.

3. Running the Code (AIMLops Workflow)

1. **Prepare your environment:** Make sure you have Python 3.6+ installed, along with the necessary libraries (pandas, RDKit, scikit-learn). Use `pip install pandas rdkit scikit-learn` to install them if you don't already. If scikit-learn is old, upgrade with `pip install scikit-learn --upgrade`.
2. **Database Setup:** Ensure your PostgreSQL database is running on `192.168.206.136` and that you can connect to it using the credentials `rd/rd` and the database name `chembl_35`.
3. **SQL Execution:** Run the SQL script in pgAdmin. Then, export the resulting table to a CSV file (e.g., `chembl_activity_data.csv`) and save it in the `../data` directory.
4. **Jupyter Notebook:** Open the Jupyter Notebook (`Topic_CheMBL_35_19_1_data_analysis.ipynb`).
5. **Run the Notebook:** Execute each cell in the notebook sequentially.
6. **Interpret the Results:** Analyze the output of the notebook, including the model evaluation metrics (MSE, R-squared), coefficients, and example predictions.

4. Example Scenarios (5 Examples)

Here are five examples of how you could use this analysis in a drug discovery context:

1. **Target-Based Activity Prediction:** You are interested in a specific protein target (e.g., a kinase) and want to predict the activity of new compounds against that target. You would replace `'CHEMBL205'` in the SQL query with the CHEMBL ID of your target kinase. You'd then run the Python notebook to build a model that predicts activity based on molecular descriptors. You could use this model to screen virtual libraries of compounds and identify promising candidates for experimental testing.

2. **Ligand-Based Virtual Screening:** You have a set of known active compounds for a target and want to identify other compounds with similar activity. You would extract activity data for those compounds from ChEMBL. You'd calculate descriptors, build a model, and then use the model to score a large database of commercially available compounds.

3. **Structure-Activity Relationship (SAR) Analysis:** You want to understand the relationship between the structure of a compound and its activity. By analyzing the coefficients of the linear regression model, you can identify which descriptors are most strongly correlated with activity. This helps you understand which structural features are important for binding to the target.

4. **Lead Optimization:** You have a lead compound with moderate activity and want to improve its potency. You can analyze the SAR of related compounds in ChEMBL to identify modifications that might increase activity. You can then use the model to predict the activity of the modified compounds before synthesizing them.

5. **Off-Target Prediction:** You want to predict the activity of a compound against a panel of targets to assess its potential for off-target effects. You would repeat the analysis for multiple

targets and build separate models for each target. This can help you identify potential safety concerns early in the drug discovery process.

5. Further Development

- **More Descriptors:** Experiment with a wider range of molecular descriptors. RDKit provides a rich set of descriptors.
- **More Complex Models:** Explore more sophisticated machine learning models (e.g., Random Forest, Support Vector Machines, Neural Networks).
- **Feature Selection:** Use feature selection techniques to identify the most relevant descriptors for your model.
- **Cross-Validation:** Use cross-validation to obtain a more robust estimate of model performance.
- **Model Deployment:** Consider deploying your model as a web service so that it can be easily used by other researchers.
- **Use other libraries:** Explore other libraries such as DeepChem for deep learning based virtual screening, or use different fingerprint types to encode structures.

Important Considerations:

- **Data Quality:** ChEMBL data is curated, but it's essential to understand its limitations. Activity data can be variable depending on assay conditions.
- **Model Validation:** Rigorously validate your models using independent datasets to ensure they generalize well.
- **Domain of Applicability:** Be aware of the domain of applicability of your model. It may not be accurate for compounds that are very different from those used to train the model.

Let me know if you have any other questions or want to explore more specific aspects of your project.