PharmApp Suite

🧠 AI for Drug Discovery and Development 🍼

🧩 Topic: Correlating bioactivity with target taxonomy

Okay, I understand. You're working with ChEMBL 35 data, using RDKit for analysis, and adhering to an AIMLops template. You're encountering SQL and Python errors and need help structuring your code and analysis for the `Topic_CheMBL_35_26` project. Let's break this down.

1. Analysis and Model (Analyse et Modèle)

The overall goal appears to be extracting and analyzing data from ChEMBL 35, likely focusing on a specific aspect within `Topic_CheMBL_35_26`. To suggest a more specific analysis model, I need more information about the topic itself. However, let's assume you're interested in building a simple QSAR (Quantitative Structure-Activity Relationship) model. This involves:

- **Data Extraction:** Fetching relevant activity data (e.g., IC50, Ki values) and corresponding molecule SMILES strings from ChEMBL.
- **Data Cleaning and Preparation:**
    - Handling missing values and inconsistencies.
    - Converting SMILES strings to RDKit Mol objects.
    - Calculating molecular descriptors using RDKit (e.g., molecular weight, LogP, number of hydrogen bond donors/acceptors).
- **Model Building:**
    - Splitting the data into training and testing sets.
    - Training a regression model (e.g., Linear Regression, Random Forest) to predict activity based on the descriptors.
- **Model Evaluation:**
    - Evaluating the model's performance on the test set using metrics like R-squared, RMSE (Root Mean Squared Error), or MAE (Mean Absolute Error).

Here's a more detailed breakdown, tailored to a QSAR task:

1. **Target Selection:** Identify a specific target (e.g., a protein or enzyme) you're interested in. ChEMBL has targets identified by ChEMBL ID.
2. **Activity Selection:** Choose the activity type you'll be modeling (e.g., IC50). Standardize activity units (e.g., convert all IC50s to nM).
3. **Data Filtering:** Filter out data points that are unreliable or outside a reasonable activity range (e.g., very large or very small IC50 values).
4. **Feature Generation (Descriptors):** Use RDKit to generate molecular descriptors. Consider using a diverse set of descriptors to capture different aspects of molecular structure.
5. **Model Selection:** Start with a simple model (e.g., Linear Regression) and then try more complex models (e.g., Random Forest, Support Vector Regression) if needed.
6. **Cross-Validation:** Use cross-validation (e.g., k-fold cross-validation) to evaluate the model's performance more robustly.
7. **Model Interpretation:** Analyze the model to understand which descriptors are most important for predicting activity. This can provide insights into the structure-activity relationship.

2. SQL Code (Code SQL)

Here's SQL code to extract data, addressing the error you mentioned and limiting the data to 100 rows. The key is to correctly filter the `standard_value` column. The initial error `ERROR: operator does not exist: numeric ~ unknown, LINE 12: AND act.standard_value ~ '^[0-9\.]+$'` arises because you're trying to use a regular expression-like operator (~) on a `numeric` column. We'll address this by casting to text when using regex. Also, we'll use a combination of `WHERE` and `LIKE` to accomplish filtering for numerical standard values.

```
-- SQL Query to extract data from ChEMBL 35 (limited to 100 rows)
-- and addressing standard_value filtering.
-- Please replace 'CHEMBL_TARGET_ID' with the actual target ID.

SELECT
    md.chembl_id AS molecule_chembl_id,
    cs.canonical_smiles,
    act.standard_type,
    act.standard_value,
    act.standard_units
FROM
    activities act
JOIN
    molecule_dictionary md ON act.molregno = md.molregno
JOIN
    compound_structures cs ON md.molregno = cs.molregno
JOIN
    target_dictionary td ON act.tid = td.tid
WHERE
    td.chembl_id = 'CHEMBL_TARGET_ID'  -- Replace with the target CHEMBL ID
    AND act.standard_type = 'IC50'      -- Or other activity type
    AND act.standard_units = 'nM'       -- Or other units
    AND act.standard_value IS NOT NULL
    AND act.standard_value::TEXT LIKE '%.%'  -- Ensure standard_value contains a
decimal point
    AND act.standard_value::TEXT NOT LIKE '%-%' -- Ensure standard_value doesn't
contain a minus sign
    AND act.standard_value::TEXT ~ '^[0-9\.]+$' -- Check it's a number
LIMIT 100;
```

Explanation:

- **`SELECT` Clause:** Selects the molecule ChEMBL ID, SMILES string, activity type, activity value, and units.
- **`FROM` Clause:** Specifies the tables to join (`activities`, `molecule_dictionary`, `compound_structures`, and `target_dictionary`).
- **`JOIN` Clause:** Connects the tables based on relevant foreign keys.
- **`WHERE` Clause:**
    - `td.chembl_id = 'CHEMBL_TARGET_ID'` : Filters for a specific ChEMBL target (replace `'CHEMBL_TARGET_ID'` with the actual ChEMBL ID). **Crucial: Find the ChEMBL ID for your target of interest.**
    - `act.standard_type = 'IC50'` : Filters for a specific activity type (e.g., IC50). Adjust this if you want Ki, Kd, etc.
    - `act.standard_units = 'nM'` : Filters for a specific unit (e.g., nM). Standardize units if needed.
    - `act.standard_value IS NOT NULL` : Excludes entries with missing standard values.

    - `act.standard_value::TEXT LIKE '%.%'`: Checks if the standard value contains a decimal point to ensure it's a floating-point number. We cast the `numeric` column to `TEXT` for the `LIKE` operator.
    - `act.standard_value::TEXT NOT LIKE '%-%'`: Makes sure it's not a negative number.
    - `act.standard_value::TEXT ~ '^[0-9\.]+$'`: Ensures that the string contains only numbers and periods. This regular expression check is now applied to a text representation of the value.
- **`LIMIT 100` Clause:** Limits the results to the first 100 rows.

How to use the SQL code:

1. **Replace `'CHEMBL_TARGET_ID'`:** Find the correct ChEMBL ID for your target and substitute it in the query. You can browse ChEMBL using their web interface to find the ID.
2. **Execute in pgAdmin:** Run this query in your pgAdmin interface connected to your ChEMBL 35 database.
3. **Save to CSV:** Export the results of the query to a CSV file named something like `chembl_data.csv` and save it to the `../data/` directory, as you specified.

3. Python Code (Code Python)

Here's the Python code (using Jupyter Notebook format) to read the data, perform descriptor calculation, and build a basic QSAR model. This addresses the error you mentioned regarding the `squared=False` parameter in `mean_squared_error` by checking scikit-learn version before use.

```
# -*- coding: utf-8 -*-
"""Topic_CheMBL_35_26_1_Data_Preparation_and_QSAR_Modeling.ipynb

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/1i_g6u8P7U9G1j_yWfWq9w5oY2_n2Wl3N
"""

import os
import pandas as pd
from rdkit import Chem
from rdkit.Chem import Descriptors
from rdkit.Chem import Lipinski
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 sklearn
import warnings
warnings.filterwarnings("ignore")
# Define the base path according to your AIMLops structure
base_path = ".."  # Assuming the notebook is one level deep from the base

# Construct the data file path
data_file_path = os.path.join(base_path, "data", "chembl_data.csv")  # Adjust filename
if needed
print(f"Loading data from: {data_file_path}")

# Load the data using pandas
try:
    df = pd.read_csv(data_file_path)
    print("Data loaded successfully.")
except FileNotFoundError:
```

```
    print(f"Error: File not found at {data_file_path}.  Make sure you've run the SQL
query and saved the data correctly.")
    exit()

# Data Cleaning and Preparation
print("\nData Cleaning and Preparation...")
df = df.dropna(subset=['canonical_smiles', 'standard_value'])  # Drop rows with
missing SMILES or activity values
df = df[df['standard_value'] > 0] # Filter out non-positive standard values to avoid
log transform errors
df['standard_value'] = pd.to_numeric(df['standard_value'], errors='coerce') # Ensure
numeric type after loading from CSV
df = df.dropna(subset=['standard_value']) # Clean up any rows that failed conversion
to numeric

# RDKit Descriptor Calculation
print("\nCalculating RDKit Descriptors...")

def lipinski_descriptors(smiles):
    """Calculates Lipinski descriptors using RDKit."""
    mol = Chem.MolFromSmiles(smiles)
    if mol is None:
        return None
    mw = Descriptors.MolWt(mol)
    logp = Descriptors.MolLogP(mol)
    hbd = Lipinski.NumHDonors(mol)
    hba = Lipinski.NumHAcceptors(mol)
    return mw, logp, hbd, hba

# Apply the function to create new columns
df[['MW', 'LogP', 'HBD', 'HBA']] = df['canonical_smiles'].apply(lambda x:
pd.Series(lipinski_descriptors(x)) if lipinski_descriptors(x) is not None else
pd.Series([None]*4))

df = df.dropna()  # Remove rows where descriptor calculation failed (invalid SMILES)
print(df.head())

# Feature Scaling (Important for many models)
print("\nFeature Scaling...")
X = df[['MW', 'LogP', 'HBD', 'HBA']]
y = np.log10(df['standard_value'])  # Log transform activity values (often improves
model performance)

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Data Splitting
print("\nData Splitting...")
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2,
random_state=42)

# Model Training
print("\nModel Training...")
model = LinearRegression()
model.fit(X_train, y_train)

# Model Evaluation
print("\nModel Evaluation...")
y_pred = model.predict(X_test)
```

```
# Check scikit-learn version before using squared=False
if sklearn.__version__ < '1.4':
    mse = mean_squared_error(y_test, y_pred)
else:
    mse = mean_squared_error(y_test, y_pred, squared=False)

r2 = r2_score(y_test, y_pred)

print(f"Mean Squared Error: {mse}")
print(f"R-squared: {r2}")

# Print the first 5 rows of the dataframe for inspection
print("\nFirst 5 rows of the processed dataframe:")
print(df.head())
```

Explanation:

1. **Imports:** Imports necessary libraries.
2. **File Path:** Defines the correct file path based on your AIMLops structure using `os.path.join`. **Important:** Double-check that this path is correct relative to where you run the notebook. I've assumed the notebook is in a subfolder.
3. **Data Loading:** Loads the CSV data using pandas. Includes error handling for the file not found.
4. **Data Cleaning:**
    - Removes rows with missing SMILES or activity values.
    - Filters out non-positive `standard_value` to avoid issues with the log transform.
    - Converts `standard_value` to numeric, handling potential errors during conversion.
5. **Descriptor Calculation:**
    - Defines a function `lipinski_descriptors` to calculate Lipinski's Rule of Five descriptors using RDKit.
    - Applies this function to the `canonical_smiles` column to create new columns for each descriptor (MW, LogP, HBD, HBA).
    - Handles cases where RDKit fails to process a SMILES string.
6. **Feature Scaling:** Scales the descriptor values using `StandardScaler`. This is *crucial* for many machine learning algorithms.
7. **Data Splitting:** Splits the data into training and testing sets.
8. **Model Training:** Creates a Linear Regression model and trains it on the training data.
9. **Model Evaluation:**
    - Predicts activity values for the test set.
    - Calculates Mean Squared Error (MSE) and R-squared to evaluate the model's performance.
    - **Addresses the `squared=False` error:** The code now checks the scikit-learn version and uses the `squared=False` parameter only if the version is 1.4 or greater.

How to use the Python code:

1. **Save the code:** Save the code above as a `.ipynb` file named `Topic_CheMBL_35_26_1_Data_Preparation_and_QSAR_Modeling.ipynb` in your `notebook` directory.
2. **Adjust the `base_path`:** Make sure the `base_path` variable in the Python code is correctly set to reflect the location of your `data` directory.
3. **Run in Jupyter Notebook:** Open the `.ipynb` file in Jupyter Notebook and run each cell sequentially.

4. **Interpret the results:** Examine the MSE and R-squared values to assess the model's performance. A higher R-squared (closer to 1) indicates a better fit. A lower MSE indicates lower prediction error.

4. Examples (Exemples)

Here are 5 examples of how you could adapt and extend this basic workflow:

1. **Different Target:** Modify the SQL query to use a different `CHEMBL_TARGET_ID` to analyze activity data for a different protein or enzyme.

2. **Different Activity Type:** Change the `act.standard_type` filter in the SQL query to analyze a different activity type (e.g., Ki, Kd). Also remember to change the unit `act.standard_units`.

3. **More Descriptors:** Add more molecular descriptors to the Python code using other RDKit functions. Explore different descriptor types to potentially improve model performance. Examples include:

```
from rdkit.Chem import AllChem
# Calculate Morgan Fingerprint (ECFP4)
def calculate_morgan_fingerprint(smiles, radius=2, nBits=2048):
    mol = Chem.MolFromSmiles(smiles)
    if mol:
        fp = AllChem.GetMorganFingerprintAsBitVect(mol, radius, nBits=nBits)
        return np.array(fp)
    else:
        return None

# Example usage (add to the descriptor calculation section)
df['MorganFP'] = df['canonical_smiles'].apply(calculate_morgan_fingerprint)
df = df.dropna() # Drop rows where fingerprint calculation failed
```

   Remember to adapt the `X` variable to include the new fingerprint data:

```
X = np.stack(df['MorganFP'].values)
```

4. **Different Model:** Experiment with different machine learning models in the Python code, such as Random Forest, Support Vector Regression, or Neural Networks. This might improve prediction accuracy.

```
from sklearn.ensemble import RandomForestRegressor

model = RandomForestRegressor(n_estimators=100, random_state=42)  # Example
model.fit(X_train, y_train)
```

5. **Feature Selection:** Implement feature selection techniques (e.g., using `SelectKBest` or Recursive Feature Elimination) to identify the most relevant descriptors and potentially simplify the model.

```
from sklearn.feature_selection import SelectKBest, f_regression

selector = SelectKBest(score_func=f_regression, k=3)  # Select the top 3
features
X_new = selector.fit_transform(X_scaled, y)

# Use X_new for training and testing instead of X_scaled
X_train, X_test, y_train, y_test = train_test_split(X_new, y, test_size=0.2,
random_state=42)
```

Key Considerations and Troubleshooting:

- **ChEMBL ID:** The most important thing is to use the correct ChEMBL ID for your target. Double-check this!
- **File Paths:** Make sure the file paths in your Python code are correct. Use `os.path.join` to construct paths dynamically.
- **RDKit Installation:** Ensure RDKit is correctly installed in your Jupyter Notebook environment.
- **Memory Usage:** If you still encounter memory issues, consider processing the data in smaller chunks (e.g., using pandas `chunksize` parameter when reading the CSV). However, with a limit of 100 rows, this shouldn't be necessary.
- **Data Exploration:** Spend time exploring the data to understand its characteristics. Look for outliers, missing values, and inconsistencies.

By following these steps and addressing the potential issues, you should be able to successfully run your ChEMBL 35 data analysis project and build a basic QSAR model using RDKit. Remember to adapt the code and analysis to your specific research question and the nature of the `Topic_CheMBL_35_26`.