PharmApp Suite

🧠 AI for Drug Discovery and Development 💉

🧩 Topic: Comparing clinical vs. preclinical molecules

Okay, I understand. You're working on a Chembl 35 data analysis project using RDKit and need help structuring your code within an AIMLops template, addressing specific errors, and limiting the data size for development.

Let's break down the process and provide code snippets to address your needs. I will provide both English and (where applicable) Bahasa Indonesia explanations.

1. Analysis Model Overview

The general flow of your project will be:

- **Data Extraction (SQL):** Extract relevant data from the Chembl 35 database, specifically focusing on activity data. The SQL query will filter for compounds with activity values, limit the results to 100 rows, and save the results to a CSV file.
- **Data Loading and Cleaning (Python):** Load the CSV file into a Pandas DataFrame.
- **Feature Engineering (Python):** Use RDKit to generate molecular descriptors (features) from the SMILES strings.
- **Data Preprocessing (Python):** Handle missing values, scale features (if necessary), and split the data into training and testing sets.
- **Model Building and Evaluation (Python):** Choose a regression model (e.g., Linear Regression, Random Forest Regression) to predict activity values. Evaluate the model's performance using metrics like Mean Squared Error (MSE) or R-squared.
- **Model Refinement (Python):** Fine-tune model parameters for better performance

2. Code Implementation (SQL & Python)

2.1 SQL Code (for pgAdmin and CSV Export)

This SQL query addresses the error you're seeing: `ERROR: operator does not exist: numeric ~ unknown, LINE 12: AND act.standard_value ~ '^[0-9\.]+$'` by casting `standard_value` to TEXT before using the regular expression. We're also adding a `LIMIT` clause to restrict the number of rows.

```sql
-- File: ../data/chembl_35_activity_data.csv

SELECT
    md.molregno,
    cs.canonical_smiles,
    act.standard_type,
    act.standard_value,
    act.standard_units
FROM
    molecule_dictionary md
JOIN
    compound_structures cs ON md.molregno = cs.molregno
JOIN
    activities act ON md.molregno = act.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\.]+$' -- Cast to TEXT to allow regex match
LIMIT 100;
```

- **Explanation (English):**
    - Selects relevant columns (molecule ID, SMILES, activity type, activity value, units).
    - Joins tables to link molecules, structures, and activity data.
    - Filters for a specific activity type (e.g., IC50) and units (e.g., nM).
    - Ensures `standard_value` is not NULL and is a number using a regular expression check *after* explicitly casting to TEXT to allow it to work.
    - Limits the result set to 100 rows.
- **Penjelasan (Bahasa Indonesia):**
    - Memilih kolom-kolom yang relevan (ID molekul, SMILES, tipe aktivitas, nilai aktivitas, satuan).
    - Menggabungkan tabel-tabel untuk menghubungkan molekul, struktur, dan data aktivitas.
    - Memfilter untuk tipe aktivitas tertentu (contohnya, IC50) dan satuan (contohnya, nM).
    - Memastikan bahwa `standard_value` tidak NULL dan merupakan angka menggunakan pemeriksaan ekspresi reguler *setelah* dikonversi secara eksplisit ke TEXT agar berfungsi.
    - Membatasi hasil menjadi 100 baris.

Steps to run the SQL code:

1. Open pgAdmin and connect to your `chembl_35` database using the provided credentials (ip: 192.168.206.136, user: rd, pass: rd).
2. Open a new query window.
3. Paste the SQL code into the query window.
4. Execute the query.
5. Right-click on the result set and select “Copy All Rows”.
6. Paste the data into a text file and save it as `chembl_35_activity_data.csv` in your `../data/` directory. Alternatively, you can use pgAdmin’s export functionality if available.

2.2 Python Code (Jupyter Notebook)

```
# File: notebook/Topic_CheMBL_35_47_1_data_preparation.ipynb

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

# Define the base path according to AIMLops structure
base_path = ".."  # Assuming the notebook is in the 'notebook' directory
data_path = os.path.join(base_path, "data", "chembl_35_activity_data.csv")
model_path = os.path.join(base_path, "model")

# Create model directory if it doesn't exist
os.makedirs(model_path, exist_ok=True)


# Load the data
```

```
try:
    df = pd.read_csv(data_path)
except FileNotFoundError:
    print(f"Error: File not found at {data_path}.  Make sure the CSV file exists.")
    exit()

# Display the first few rows of the DataFrame
print(df.head())

# Data Cleaning and Preprocessing
df = df.dropna(subset=['canonical_smiles', 'standard_value'])

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

# Keep only needed columns
df = df[['canonical_smiles', 'standard_value']]

# RDKit Feature Generation
def generate_features(smiles):
    mol = Chem.MolFromSmiles(smiles)
    if mol is not None:
        return [Descriptors.MolWt(mol), Descriptors.MolLogP(mol),
Descriptors.NumHAcceptors(mol), Descriptors.NumHDonors(mol)] # Example features
    else:
        return None

df['features'] = df['canonical_smiles'].apply(generate_features)
df = df.dropna(subset=['features'])

# Convert features to a DataFrame
feature_names = ['MolWt', 'MolLogP', 'NumHAcceptors', 'NumHDonors']
features_df = pd.DataFrame(df['features'].tolist(), columns=feature_names)
df = pd.concat([df, features_df], axis=1)
df = df.drop(columns=['features', 'canonical_smiles'])

# Splitting the data
X = df[['MolWt', 'MolLogP', 'NumHAcceptors', 'NumHDonors']]
y = df['standard_value']

# Data Scaling (important for some models like Linear Regression)
scaler = StandardScaler()
X = scaler.fit_transform(X)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

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

# Model Evaluation
y_pred = model.predict(X_test)
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}")
```

```
# Save the model (optional)
import joblib
joblib.dump(model, os.path.join(model_path, 'linear_regression_model.pkl'))


# Load the model (optional)
loaded_model = joblib.load(os.path.join(model_path, 'linear_regression_model.pkl'))
print("Loaded model:", loaded_model)
```

- **Explanation (English):**
  - **Import Libraries:** Imports necessary libraries (Pandas, RDKit, scikit-learn).
  - **Define Paths:** Defines file paths using `os.path.join` for better portability and adherence to your AIMLops structure.
  - **Load Data:** Loads the CSV data into a Pandas DataFrame. Handles the case where the file might not be found.
  - **Data Cleaning:** Removes rows with missing SMILES strings or standard values. Converts the standard value to numeric.
  - **Feature Generation:** Uses RDKit to generate molecular descriptors (features) from SMILES strings. This example uses `MolWt`, `MolLogP`, `NumHAcceptors`, and `NumHDonors`. Add more features as needed.
  - **Data Splitting:** Splits the data into training and testing sets.
  - **Data Scaling:** Scales the input features using `StandardScaler` to have zero mean and unit variance. This is crucial for algorithms sensitive to feature scaling (like Linear Regression, SVMs, etc.).
  - **Model Training:** Trains a Linear Regression model.
  - **Model Evaluation:** Evaluates the model using Mean Squared Error (MSE) and R-squared.
  - **Save and Load Model:** Using `joblib` to save the model.
- **Penjelasan (Bahasa Indonesia):**
  - **Impor Pustaka:** Mengimpor pustaka yang diperlukan (Pandas, RDKit, scikit-learn).
  - **Definisikan Path:** Mendefinisikan path file menggunakan `os.path.join` untuk portabilitas yang lebih baik dan kepatuhan terhadap struktur AIMLops Anda.
  - **Muat Data:** Memuat data CSV ke dalam Pandas DataFrame. Menangani kasus di mana file mungkin tidak ditemukan.
  - **Pembersihan Data:** Menghapus baris dengan string SMILES atau nilai standar yang hilang. Mengonversi nilai standar menjadi numerik.
  - **Pembuatan Fitur:** Menggunakan RDKit untuk menghasilkan deskriptor molekul (fitur) dari string SMILES. Contoh ini menggunakan `MolWt`, `MolLogP`, `NumHAcceptors`, dan `NumHDonors`. Tambahkan lebih banyak fitur sesuai kebutuhan.
  - **Pembagian Data:** Membagi data menjadi set pelatihan dan pengujian.
  - **Penskalaan Data:** Menggunakan `StandardScaler` untuk menskalakan fitur input supaya memiliki mean nol dan varians satuan. Ini sangat penting untuk algoritma yang sensitif terhadap penskalaan fitur (seperti Regresi Linier, SVM, dll.).
  - **Pelatihan Model:** Melatih model Regresi Linier.
  - **Evaluasi Model:** Mengevaluasi model menggunakan Mean Squared Error (MSE) dan R-squared.
  - **Simpan dan Muat Model:** Menggunakan `joblib` untuk menyimpan model.

2.3 Addressing the `squared=False` Error:

The error `old scikit-learn version does not support parameters squared=False in the mean_squared_error function` arises if you are using an older version of scikit-learn. The `squared=False` parameter was introduced in a later version.

The easiest solution is to *remove* the `squared=False` argument entirely. The default behavior is to return the MSE (Mean Squared Error), which is the square root of the RMSE (Root Mean Squared Error). If you specifically need RMSE, you can calculate it manually:

```
# Model Evaluation
y_pred = model.predict(X_test)
mse = mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse)  # Calculate RMSE manually
r2 = r2_score(y_test, y_pred)

print(f"Mean Squared Error: {mse}")
print(f"Root Mean Squared Error: {rmse}") # Show RMSE
print(f"R-squared: {r2}")
```

Alternatively, update your scikit-learn version:

```
pip install --upgrade scikit-learn
```

3. AIMLops Folder Structure

Your code already reflects the suggested AIMLops structure:

- `../data/`: Contains the `chembl_35_activity_data.csv` file.
- `../notebook/`: Contains the Jupyter Notebook files (`Topic_CheMBL_35_47_1_data_preparation.ipynb`, `Topic_CheMBL_35_47_2_model_training.ipynb`, etc.).
- `../model/`: Will contain the saved model file (`linear_regression_model.pkl`).

4. Example Notebooks

Here are five example notebook files with their intended focus:

1. **`Topic_CheMBL_35_47_1_data_preparation.ipynb`:** (As provided above) Focuses on data loading, cleaning, and feature engineering using RDKit. Saves the processed data (optional).
2. **`Topic_CheMBL_35_47_2_model_training.ipynb`:** Loads the processed data, trains a regression model (Linear Regression, Random Forest, etc.), and evaluates its performance. Saves the trained model.
3. **`Topic_CheMBL_35_47_3_model_evaluation.ipynb`:** Loads the trained model, evaluates it on a held-out test set, and generates performance metrics (MSE, R-squared, plots).
4. **`Topic_CheMBL_35_47_4_feature_selection.ipynb`:** Explores different feature selection techniques (e.g., SelectKBest, Recursive Feature Elimination) to identify the most relevant features for the model.
5. **`Topic_CheMBL_35_47_5_parameter_tuning.ipynb`:** Uses techniques like GridSearchCV or RandomizedSearchCV to optimize the hyperparameters of the chosen model.

Example content for `Topic_CheMBL_35_47_2_model_training.ipynb`:

```
# File: notebook/Topic_CheMBL_35_47_2_model_training.ipynb

import os
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor  # Example model
from sklearn.metrics import mean_squared_error, r2_score
import joblib
from sklearn.preprocessing import StandardScaler
```

```
# Define paths
base_path = ".."
data_path = os.path.join(base_path, "data", "chembl_35_activity_data.csv")
model_path = os.path.join(base_path, "model")

# Load the data (assuming data_preparation notebook saved a processed CSV)
df = pd.read_csv(data_path)

# Data Cleaning and Preprocessing
df = df.dropna(subset=['canonical_smiles', 'standard_value'])

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

# Keep only needed columns
df = df[['canonical_smiles', 'standard_value']]

# RDKit Feature Generation
def generate_features(smiles):
    mol = Chem.MolFromSmiles(smiles)
    if mol is not None:
        return [Descriptors.MolWt(mol), Descriptors.MolLogP(mol),
Descriptors.NumHAcceptors(mol), Descriptors.NumHDonors(mol)] # Example features
    else:
        return None

df['features'] = df['canonical_smiles'].apply(generate_features)
df = df.dropna(subset=['features'])

# Convert features to a DataFrame
feature_names = ['MolWt', 'MolLogP', 'NumHAcceptors', 'NumHDonors']
features_df = pd.DataFrame(df['features'].tolist(), columns=feature_names)
df = pd.concat([df, features_df], axis=1)
df = df.drop(columns=['features', 'canonical_smiles'])

# Splitting the data
X = df[['MolWt', 'MolLogP', 'NumHAcceptors', 'NumHDonors']]
y = df['standard_value']

# Data Scaling (important for some models like Linear Regression)
scaler = StandardScaler()
X = scaler.fit_transform(X)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)


# Model Training (Random Forest Example)
model = RandomForestRegressor(n_estimators=100, random_state=42)  # Example: 100 trees
model.fit(X_train, y_train)

# Model Evaluation
y_pred = model.predict(X_test)
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}")

# Save the model
joblib.dump(model, os.path.join(model_path, 'random_forest_model.pkl'))
```

5. Important Considerations

- **Error Handling:** Implement robust error handling in your code (e.g., `try...except` blocks) to catch potential issues during data loading, feature generation, or model training.
- **Feature Selection:** Experiment with different feature selection techniques to improve model performance and reduce overfitting.
- **Model Selection:** Try different regression models (e.g., Linear Regression, Random Forest, Support Vector Regression) and compare their performance.
- **Hyperparameter Tuning:** Optimize the hyperparameters of your chosen model using techniques like GridSearchCV or RandomizedSearchCV.
- **Data Visualization:** Use visualizations (e.g., scatter plots, histograms) to explore the data and gain insights into relationships between features and activity values.
- **Logging:** Implement logging to track the progress of your experiments and debug any issues.
- **Version Control:** Use a version control system like Git to track changes to your code and collaborate with others.

By following these guidelines and adapting the code snippets to your specific needs, you should be able to successfully run your Chembl 35 data analysis project within the AIMLops framework. Remember to adjust the code and techniques based on your specific research goals and the characteristics of your data. Good luck!