PharmApp Suite

🧠 AI for Drug Discovery and Development 🧪

🧩 Topic: Predicting clinical success from early bioactivity

Okay, I understand. You're working on a ChEMBL 35 data analysis project using RDKit, following an AIMLops template, and you're encountering some errors. Let's break down the task and address the issues. I will provide SQL and Python code snippets in both English and Indonesian, along with explanations, focusing on getting a manageable dataset (100 rows) and fixing the errors you're encountering.

1. Analysis Model:

The core idea is to extract relevant chemical and bioactivity data from the ChEMBL database, perform data cleaning and transformation, and then use RDKit to calculate molecular descriptors. These descriptors can then be used for various downstream tasks like building predictive models for activity, analyzing structure-activity relationships (SAR), or simply understanding the chemical space represented in your dataset.

Your current model appears to be focusing on filtering bioactivity data based on standard values and calculating some basic descriptors.

2. Addressing the Errors:

- **Error a: `ERROR: operator does not exist: numeric ~ unknown, LINE 12: AND act.standard_value ~ '^[0-9\.]+$'`**

  This error arises because you're trying to use a regular expression (~) on a numeric column (`act.standard_value`). PostgreSQL doesn't allow direct regex matching on numeric types. Instead, we can cast the value to text or use alternative methods to check if it's a valid number.

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

  This error indicates that your `scikit-learn` version is outdated. You can either upgrade `scikit-learn` (`pip install scikit-learn --upgrade`) or remove the `squared=False` argument from your `mean_squared_error` function call if you're not using the root mean squared error.

3. Code and Explanation:

English:

Folder Structure (Based on AIMLops):

```
project_root/
├── data/
│   └── chembl_activities_100.csv   # Your extracted data will be saved here
├── notebooks/
│   └── Topic_CheMBL_35_43_1_data_extraction.ipynb
│   └── Topic_CheMBL_35_43_2_data_analysis.ipynb
├── scripts/
│   └── (Optional: Any standalone scripts)
├── models/
│   └── (Placeholder for model saving later)
└── README.md
```

SQL Code (to be run in pgAdmin and save the output to `data/chembl_activities_100.csv`):

```sql
-- Save as: ../data/chembl_activities_100.csv
COPY (
    SELECT
        act.molregno,
        act.standard_type,
        act.standard_value,
        act.standard_units,
        act.assay_id,
        cmp.chembl_id,
        cmp.canonical_smiles
    FROM
        activities act
    JOIN
        assays ass ON act.assay_id = ass.assay_id
    JOIN
        target_dictionary td ON ass.tid = td.tid
    JOIN
        molecule_dictionary md ON act.molregno = md.molregno
    JOIN
        compound_structures cmp ON md.molregno = cmp.molregno
    WHERE
        td.target_type = 'SINGLE PROTEIN'
        AND act.standard_type = 'IC50'
        AND act.standard_units = 'nM'
        AND act.standard_value IS NOT NULL
        AND act.standard_value::text ~ '^[0-9\.]+$' -- Safe numeric check
    LIMIT 100
) TO '/tmp/chembl_activities_100.csv' WITH CSV HEADER;
```

Explanation:

- **`COPY ... TO '/tmp/chembl_activities_100.csv' WITH CSV HEADER;`**: This copies the result of the SQL query to a CSV file. Make sure PostgreSQL user has permissions to write to this directory. Adjust `/tmp/chembl_activities_100.csv` if necessary. Remember to move the created file from `/tmp/chembl_activities_100.csv` to `./data/` folder after execution.
- **`SELECT ... FROM ... JOIN ...`**: This selects the desired columns and joins the relevant tables (activities, assays, target_dictionary, molecule_dictionary, compound_structures) to gather the data.
- **`WHERE ...`**: This filters the data based on:
    - `td.target_type = 'SINGLE PROTEIN'`: Targets a specific type (single protein).
    - `act.standard_type = 'IC50'`: Filters for IC50 values.
    - `act.standard_units = 'nM'`: Filters for nanomolar units.
    - `act.standard_value IS NOT NULL`: Excludes missing values.
    - `act.standard_value::text ~ '^[0-9\.]+$'`: This ensures that the `standard_value` can be safely converted to a number by casting it to text and checking that it only contains digits and periods. This fixes your regex error.
- **`LIMIT 100`**: This limits the result to 100 rows.

Python Code (`notebooks/Topic_CheMBL_35_43_2_data_analysis.ipynb`):

```python
import os
import pandas as pd
from rdkit import Chem
from rdkit.Chem import Descriptors
import numpy as np
from sklearn.metrics import mean_squared_error
```

```python
# Define base path
base_path = os.getcwd()  # Current working directory where the notebook is
data_path = os.path.join(base_path, 'data', 'chembl_activities_100.csv')

# Load the data
try:
    df = pd.read_csv(data_path)
except FileNotFoundError:
    print(f"Error: The file {data_path} was not found. Please ensure it exists and the
path is correct.")
    exit()

# Handle missing values (more robust)
df = df.dropna(subset=['canonical_smiles', 'standard_value']) #Dropping NA on both
columns.

# Convert standard_value to numeric (after cleaning)
df['standard_value'] = pd.to_numeric(df['standard_value'], errors='coerce')
df = df.dropna(subset=['standard_value']) # Drop NaN produced by numeric conversion

# RDKit Molecular Descriptor Calculation
def calculate_descriptors(smiles):
    try:
        mol = Chem.MolFromSmiles(smiles)
        if mol is None:
            return None
        descriptors = {desc[0]: desc[1](mol) for desc in Descriptors.descList}
        return descriptors
    except:
        return None

df['descriptors'] = df['canonical_smiles'].apply(calculate_descriptors)
df = df.dropna(subset=['descriptors'])  # Remove rows where descriptor calculation
failed

# Expand the descriptor column into separate columns
df = pd.concat([df.drop(['descriptors'], axis=1), df['descriptors'].apply(pd.Series)],
axis=1)


# Example: Calculate LogP and Molecular Weight
df['LogP'] = df['canonical_smiles'].apply(lambda x: Chem.MolFromSmiles(x).GetMolWt()
if Chem.MolFromSmiles(x) else None)
df['MolWt'] = df['canonical_smiles'].apply(lambda x:
Descriptors.MolLogP(Chem.MolFromSmiles(x)) if Chem.MolFromSmiles(x) else None)

# Simple Example of using the molecular descriptors
X = df[['LogP', 'MolWt']].fillna(0) #Fills nan values with 0s
y = np.log10(df['standard_value']) #Transforming standard value

# Basic model (for demonstration)
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression

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

model = LinearRegression()
```

```python
model.fit(X_train, y_train)

y_pred = model.predict(X_test)

# Calculate MSE (fix the scikit-learn error)
mse = mean_squared_error(y_test, y_pred) # Remove squared=False if needed for old
scikit-learn
print(f"Mean Squared Error: {mse}")

# Print the first 5 rows of the dataframe with calculated features
print(df.head())
```

Explanation:

- **Import Libraries:** Imports necessary libraries like `os`, `pandas`, `rdkit`, `numpy`, and `sklearn`.
- **Define Paths:** Uses `os.path.join` to construct the correct file path to your CSV data, making it robust to different environments.
- **Load Data:** Loads the CSV data into a Pandas DataFrame. Includes error handling for the case where the file is not found.
- **Data Cleaning:** Drops rows with missing `canonical_smiles` or `standard_value`. Converts `standard_value` to numeric type and handles potential conversion errors. Crucially, missing values can cause the whole thing to blow up.
- **RDKit Descriptor Calculation:** Defines a function `calculate_descriptors` to compute molecular descriptors from SMILES strings using RDKit. Handles cases where SMILES parsing fails by returning `None`.
- **Expand Descriptors:** Expands the dictionary of descriptors into individual columns in the DataFrame.
- **Example Descriptors & Model:** Calculates `LogP` and `MolWt` as examples. Creates a simple linear regression model to predict log-transformed IC50 values using these descriptors. Calculates and prints the Mean Squared Error (MSE). The transformation of `standard_value` (IC50) via `np.log10` is a common practice as activity data often follows a log-normal distribution.
- **Prints the first 5 rows of the Dataframe** Displays the first five rows of the dataframe, showcasing calculated features and data.

Indonesian:

Struktur Folder (Berdasarkan AIMLops):

```
project_root/
├── data/
│   └── chembl_activities_100.csv   # Data hasil ekstraksi Anda akan disimpan di sini
├── notebooks/
│   └── Topic_CheMBL_35_43_1_data_extraction.ipynb
│   └── Topic_CheMBL_35_43_2_data_analysis.ipynb
├── scripts/
│   └── (Opsional: Skrip mandiri lainnya)
├── models/
│   └── (Placeholder untuk penyimpanan model nanti)
└── README.md
```

Kode SQL (dijalankan di pgAdmin dan simpan output ke `data/chembl_activities_100.csv`):

```sql
-- Simpan sebagai: ../data/chembl_activities_100.csv
COPY (
    SELECT
        act.molregno,
        act.standard_type,
        act.standard_value,
        act.standard_units,
        act.assay_id,
        cmp.chembl_id,
        cmp.canonical_smiles
    FROM
        activities act
    JOIN
        assays ass ON act.assay_id = ass.assay_id
    JOIN
        target_dictionary td ON ass.tid = td.tid
    JOIN
        molecule_dictionary md ON act.molregno = md.molregno
    JOIN
        compound_structures cmp ON md.molregno = cmp.molregno
    WHERE
        td.target_type = 'SINGLE PROTEIN'
        AND act.standard_type = 'IC50'
        AND act.standard_units = 'nM'
        AND act.standard_value IS NOT NULL
        AND act.standard_value::text ~ '^[0-9\.]+$' -- Pemeriksaan numerik yang aman
    LIMIT 100
) TO '/tmp/chembl_activities_100.csv' WITH CSV HEADER;
```

Penjelasan:

- **`COPY ... TO '/tmp/chembl_activities_100.csv' WITH CSV HEADER;`**: Ini menyalin hasil query SQL ke file CSV. Pastikan pengguna PostgreSQL memiliki izin untuk menulis ke direktori ini. Sesuaikan `/tmp/chembl_activities_100.csv` jika diperlukan. Ingat untuk memindahkan file yang dibuat dari `/tmp/chembl_activities_100.csv` ke folder `./data/` setelah eksekusi.
- **`SELECT ... FROM ... JOIN ...`**: Ini memilih kolom yang diinginkan dan menggabungkan tabel yang relevan (activities, assays, target_dictionary, molecule_dictionary, compound_structures) untuk mengumpulkan data.
- **`WHERE ...`**: Ini menyaring data berdasarkan:
  - `td.target_type = 'SINGLE PROTEIN'`: Menargetkan tipe tertentu (protein tunggal).
  - `act.standard_type = 'IC50'`: Menyaring untuk nilai IC50.
  - `act.standard_units = 'nM'`: Menyaring untuk unit nanomolar.
  - `act.standard_value IS NOT NULL`: Mengecualikan nilai yang hilang.
  - `act.standard_value::text ~ '^[0-9\.]+$'`: Ini memastikan bahwa `standard_value` dapat dikonversi dengan aman menjadi angka dengan mengubahnya menjadi teks dan memeriksa apakah hanya berisi angka dan titik. Ini memperbaiki kesalahan regex Anda.
- **`LIMIT 100`**: Ini membatasi hasilnya menjadi 100 baris.

Kode Python (`notebooks/Topic_CheMBL_35_43_2_data_analysis.ipynb`):

```python
import os
import pandas as pd
from rdkit import Chem
from rdkit.Chem import Descriptors
import numpy as np
from sklearn.metrics import mean_squared_error

# Definisikan path dasar
base_path = os.getcwd()  # Direktori kerja saat ini di mana notebook berada
data_path = os.path.join(base_path, 'data', 'chembl_activities_100.csv')

# Muat data
try:
    df = pd.read_csv(data_path)
except FileNotFoundError:
    print(f"Error: File {data_path} tidak ditemukan. Pastikan file tersebut ada dan
path-nya benar.")
    exit()

# Tangani nilai yang hilang (lebih kuat)
df = df.dropna(subset=['canonical_smiles', 'standard_value']) #Menghapus NA di kedua
kolom.

# Konversi standard_value ke numerik (setelah dibersihkan)
df['standard_value'] = pd.to_numeric(df['standard_value'], errors='coerce')
df = df.dropna(subset=['standard_value']) # Menghapus NaN yang dihasilkan oleh
konversi numerik

# Perhitungan Deskriptor Molekuler RDKit
def hitung_deskriptor(smiles):
    try:
        mol = Chem.MolFromSmiles(smiles)
        if mol is None:
            return None
        descriptors = {desc[0]: desc[1](mol) for desc in Descriptors.descList}
        return descriptors
    except:
        return None

df['descriptors'] = df['canonical_smiles'].apply(hitung_deskriptor)
df = df.dropna(subset=['descriptors'])  # Hapus baris di mana perhitungan deskriptor
gagal

# Perluas kolom deskriptor menjadi kolom terpisah
df = pd.concat([df.drop(['descriptors'], axis=1), df['descriptors'].apply(pd.Series)],
axis=1)


# Contoh: Hitung LogP dan Berat Molekul
df['LogP'] = df['canonical_smiles'].apply(lambda x: Chem.MolFromSmiles(x).GetMolWt()
if Chem.MolFromSmiles(x) else None)
df['MolWt'] = df['canonical_smiles'].apply(lambda x:
Descriptors.MolLogP(Chem.MolFromSmiles(x)) if Chem.MolFromSmiles(x) else None)

# Contoh Sederhana penggunaan deskriptor molekul
X = df[['LogP', 'MolWt']].fillna(0) #Mengisi nilai nan dengan 0
y = np.log10(df['standard_value']) #Transformasi nilai standar

# Model dasar (untuk demonstrasi)
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression

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

model = LinearRegression()
model.fit(X_train, y_train)

y_pred = model.predict(X_test)

# Hitung MSE (perbaiki kesalahan scikit-learn)
mse = mean_squared_error(y_test, y_pred) # Hapus squared=False jika diperlukan untuk
```

```
scikit-learn lama
print(f"Mean Squared Error: {mse}")

# Cetak 5 baris pertama dari dataframe dengan fitur yang dihitung
print(df.head())
```

Penjelasan:

- **Impor Pustaka:** Mengimpor pustaka yang diperlukan seperti `os`, `pandas`, `rdkit`, `numpy`, dan `sklearn`.
- **Tentukan Path:** Menggunakan `os.path.join` untuk membangun path file yang benar ke data CSV Anda, membuatnya kuat terhadap lingkungan yang berbeda.
- **Muat Data:** Memuat data CSV ke dalam Pandas DataFrame. Menyertakan penanganan kesalahan untuk kasus di mana file tidak ditemukan.
- **Pembersihan Data:** Menghapus baris dengan `canonical_smiles` atau `standard_value` yang hilang. Mengonversi `standard_value` ke tipe numerik dan menangani potensi kesalahan konversi. Penting, nilai yang hilang dapat menyebabkan semuanya gagal.
- **Perhitungan Deskriptor RDKit:** Mendefinisikan fungsi `hitung_deskriptor` untuk menghitung deskriptor molekuler dari string SMILES menggunakan RDKit. Menangani kasus di mana penguraian SMILES gagal dengan mengembalikan `None`.
- **Perluas Deskriptor:** Memperluas kamus deskriptor menjadi kolom individual dalam DataFrame.
- **Contoh Deskriptor & Model:** Menghitung `LogP` dan `MolWt` sebagai contoh. Membuat model regresi linier sederhana untuk memprediksi nilai IC50 yang ditransformasikan log menggunakan deskriptor ini. Menghitung dan mencetak Mean Squared Error (MSE). Transformasi `standard_value` (IC50) melalui `np.log10` adalah praktik umum karena data aktivitas sering mengikuti distribusi log-normal.
- **Mencetak 5 baris pertama dari Dataframe:** Menampilkan lima baris pertama dari dataframe, menampilkan fitur dan data yang dihitung.

4. Example Usages:

Here are 5 examples on what else you can do with the code:

1. **Calculate and Visualize Descriptor Correlations:**

   ```python
   import seaborn as sns
   import matplotlib.pyplot as plt

   # Calculate the correlation matrix for descriptors
   descriptor_cols = df.columns[df.columns.get_loc('MolWt')+1:]
   corr_matrix = df[descriptor_cols].corr()

   # Visualize the correlation matrix using a heatmap
   plt.figure(figsize=(12, 10))
   sns.heatmap(corr_matrix, annot=False, cmap='coolwarm')
   plt.title('Descriptor Correlation Matrix')
   plt.show()
   ```

   This example calculates the correlation between the various calculated descriptors and visualizes it as a heatmap. This helps identify highly correlated descriptors, which might be redundant in modeling.

2. **Building a more Complex Regression Model:**

   ```python
   from sklearn.ensemble import RandomForestRegressor

   # Choose descriptors as features
   ```

```python
X = df[descriptor_cols].fillna(0)
y = np.log10(df['standard_value'])

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

# Train a Random Forest model
model = RandomForestRegressor(n_estimators=100, random_state=42)  # Example
parameters
model.fit(X_train, y_train)

# Evaluate the model
y_pred = model.predict(X_test)
mse = mean_squared_error(y_test, y_pred)
print(f"Random Forest MSE: {mse}")
```

This example demonstrates building a more sophisticated regression model (Random Forest) using all the calculated descriptors.

3. **Calculate and Visualize Activity Distribution:**

```python
import matplotlib.pyplot as plt

# Plot the distribution of log-transformed IC50 values
plt.hist(np.log10(df['standard_value']), bins=30)
plt.xlabel('Log10(IC50)')
plt.ylabel('Frequency')
plt.title('Distribution of Log10(IC50) Values')
plt.show()
```

This example creates a histogram showing the distribution of the log-transformed IC50 values, providing insights into the range and distribution of activity values.

4. **Filtering Data based on Descriptor Values:**

```python
# Filter compounds based on a LogP range
filtered_df = df[(df['LogP'] >= 2) & (df['LogP'] <= 4)]
print(f"Number of compounds with LogP between 2 and 4: {len(filtered_df)}")
```

This example shows how to filter the DataFrame based on the calculated descriptor values, allowing you to focus on subsets of compounds with specific properties.

5. **Saving the Processed Data:**

```python
# Save the DataFrame with calculated descriptors to a new CSV file
output_path = os.path.join(base_path, 'data', 'chembl_activities_processed.csv')
df.to_csv(output_path, index=False)
print(f"Processed data saved to: {output_path}")
```

This example saves the DataFrame, including all the calculated descriptors, to a new CSV file for later use.

Important Considerations:

- **File Paths:** Double-check that the file paths are correct in your environment. Use absolute paths if necessary, especially for the SQL `COPY` command.
- **Permissions:** Ensure that the PostgreSQL user has the necessary permissions to write to the specified file path.
- **RDKit Installation:** Make sure RDKit is correctly installed in your Python environment.

- **Data Integrity:** Always validate your data after loading and cleaning it. Check for unexpected values or inconsistencies.

By following these steps and addressing the errors, you should be able to successfully extract, process, and analyze ChEMBL data using RDKit in your AIMLops project. Remember to adapt the code and examples to your specific research questions and data. Good luck!