PharmApp Suite

🧠 AI for Drug Discovery and Development 💉

🧩 Topic: Structure-Activity Relationship (SAR) mining

Okay, I understand. You're working on a Chembl 35 data analysis project using RDKit within an AIMLops framework. You're facing issues with SQL syntax and scikit-learn compatibility. You need help structuring your code and providing example analysis.

Here's a breakdown of how we can approach this, keeping the data size manageable and addressing your specific errors.

1. Analysis Model & Workflow

The overall goal is to extract relevant chemical and bioactivity data from the Chembl 35 database, analyze it using RDKit for cheminformatics tasks, and potentially build predictive models. Here's a suggested workflow:

- **Data Extraction (SQL):** Extract relevant data (compounds and their bioactivities) from ChEMBL. This will be saved as CSV files.
- **Data Loading and Preprocessing (Python):** Load the CSV files into a Pandas DataFrame, clean the data (handle missing values, convert data types), and filter for relevant bioactivity types (e.g., IC50).
- **Feature Generation (Python/RDKit):** Use RDKit to generate molecular descriptors (e.g., molecular weight, logP, number of hydrogen bond donors/acceptors) for each compound. These will be your features.
- **Exploratory Data Analysis (EDA) (Python):** Explore the relationships between the molecular descriptors and bioactivity values using visualization and statistical analysis.
- **Modeling (Python/Scikit-learn):** Build predictive models to relate the molecular descriptors to bioactivity (e.g., regression models to predict IC50 values).
- **Evaluation (Python):** Evaluate the performance of the models using appropriate metrics (e.g., Mean Squared Error, R-squared).

2. Addressing the Errors

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

  This error occurs because you're trying to use a regular expression (~) to match a numeric column (`act.standard_value`). The ~ operator is typically used for string matching. To fix this, we'll use the `SIMILAR TO` operator (if supported by your PostgreSQL version) or a `CASE` statement or cast the column to TEXT before applying the regular expression.

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

  This means you're using an older version of scikit-learn. The simplest solution is to either:

  - **Upgrade scikit-learn:** `pip install --upgrade scikit-learn`
  - **Remove `squared=False`:** If upgrading isn't an option, remove the `squared=False` argument from `mean_squared_error`. This will return the Mean Squared Error (MSE) instead of the Root Mean Squared Error (RMSE), which is often what you want anyway.

3. Folder Structure (Based on AIMLops Template)

Assuming a simplified AIMLops structure:

```
project_root/
├── data/         # CSV files extracted from ChEMBL
├── notebooks/    # Jupyter notebooks with your analysis
├── sql/          # SQL scripts for data extraction
├── src/          # (Optional) Python modules for reusable functions
├── README.md     # Project documentation
```

4. Code Examples (English & French)

Here are 5 examples, following your specified structure and limitations (100 rows, addressing errors, proper path construction):

Example 1: Data Extraction (SQL)

```sql
-- File: project_root/sql/extract_chembl_data.sql
-- Extracts compound and bioactivity data for a specific target (e.g., CHEMBL205 - Dopamine D4 receptor)

SELECT
    md.chembl_id,
    cs.canonical_smiles,
    act.standard_type,
    act.standard_value,
    act.standard_units
FROM
    compound_structures cs
JOIN
    molecule_dictionary md ON cs.molregno = md.molregno
JOIN
    activities act ON md.molregno = act.molregno
JOIN
    target_dictionary td ON act.tid = td.tid
WHERE
    td.chembl_id = 'CHEMBL205'  -- Example: Dopamine D4 receptor
    AND act.standard_type = 'IC50'
    AND act.standard_units = 'nM'
    AND act.standard_value IS NOT NULL
    -- Convert standard_value to TEXT for regex matching or use SIMILAR TO or CASE statement
    AND CAST(act.standard_value AS TEXT) SIMILAR TO '[0-9.]+' -- PostgreSQL specific
LIMIT 100;
```

(French):

```sql
-- Fichier : project_root/sql/extract_chembl_data.sql
-- Extrait les données de composés et de bioactivité pour une cible spécifique (par exemple, CHEMBL205 - Récepteur Dopamine D4)

SELECT
    md.chembl_id,
    cs.canonical_smiles,
    act.standard_type,
    act.standard_value,
    act.standard_units
FROM
    compound_structures cs
JOIN
    molecule_dictionary md ON cs.molregno = md.molregno
JOIN
    activities act ON md.molregno = act.molregno
```

```sql
JOIN
    target_dictionary td ON act.tid = td.tid
WHERE
    td.chembl_id = 'CHEMBL205'  -- Exemple : Récepteur Dopamine D4
    AND act.standard_type = 'IC50'
    AND act.standard_units = 'nM'
    AND act.standard_value IS NOT NULL
    -- Convertit standard_value en TEXT pour la correspondance regex ou utilise
SIMILAR TO ou CASE
    AND CAST(act.standard_value AS TEXT) SIMILAR TO '[0-9.]+' -- Spécifique à
PostgreSQL
LIMIT 100;
```

Steps to run in pgAdmin:

1. Connect to your PostgreSQL server (192.168.206.136, user: rd, pass: rd, database: chembl_35).
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 grid and select "Copy All Rows".
6. Paste the data into a text editor and save it as a CSV file (e.g., `chembl_205_ic50.csv`) in the `project_root/data/` directory. Make sure to adjust the delimiters and quoting as needed.

Example 2: Loading Data and Basic Statistics (Python)

```python
# File: project_root/notebooks/Topic_CheMBL_35_8_1_data_loading.ipynb
import pandas as pd
import os

base_path = "project_root"  # Replace with your actual project root path (absolute)
data_path = os.path.join(base_path, "data", "chembl_205_ic50.csv")

try:
    df = pd.read_csv(data_path)
except FileNotFoundError:
    print(f"Error: File not found at {data_path}.  Make sure you ran the SQL script
and saved the CSV.")
    exit()

print(df.head())
print(df.describe())
```

(French):

```python
# Fichier : project_root/notebooks/Topic_CheMBL_35_8_1_data_loading.ipynb
import pandas as pd
import os

base_path = "project_root"  # Remplacez par votre chemin de base de projet réel
(absolu)
data_path = os.path.join(base_path, "data", "chembl_205_ic50.csv")

try:
    df = pd.read_csv(data_path)
except FileNotFoundError:
    print(f"Erreur : Fichier introuvable à {data_path}. Assurez-vous d'avoir exécuté
le script SQL et enregistré le CSV.")
    exit()
```

```
print(df.head())
print(df.describe())
```

Important: Replace `"project_root"` with the *actual* absolute path to your project root directory. This is crucial for `os.path.join` to work correctly.

Example 3: RDKit Feature Generation (Python)

```
# File: project_root/notebooks/Topic_CheMBL_35_8_2_feature_generation.ipynb
import pandas as pd
import os
from rdkit import Chem
from rdkit.Chem import Descriptors

base_path = "project_root" # Replace with your actual project root path (absolute)
data_path = os.path.join(base_path, "data", "chembl_205_ic50.csv")

try:
    df = pd.read_csv(data_path)
except FileNotFoundError:
    print(f"Error: File not found at {data_path}.  Make sure you ran the SQL script
and saved the CSV.")
    exit()

def calculate_descriptors(smiles):
    mol = Chem.MolFromSmiles(smiles)
    if mol is None:
        return None, None, None  # Handle invalid SMILES strings
    mw = Descriptors.MolWt(mol)
    logp = Descriptors.MolLogP(mol)
    hbd = Descriptors.NumHDonors(mol)
    return mw, logp, hbd

df[['MW', 'LogP', 'HBD']] = df['canonical_smiles'].apply(lambda x:
pd.Series(calculate_descriptors(x)))

# Handle potential None values (invalid SMILES)
df = df.dropna(subset=['MW', 'LogP', 'HBD'])

print(df.head())
```

(French):

```
# Fichier : project_root/notebooks/Topic_CheMBL_35_8_2_feature_generation.ipynb
import pandas as pd
import os
from rdkit import Chem
from rdkit.Chem import Descriptors

base_path = "project_root" # Remplacez par votre chemin de base de projet réel
(absolu)
data_path = os.path.join(base_path, "data", "chembl_205_ic50.csv")

try:
    df = pd.read_csv(data_path)
except FileNotFoundError:
    print(f"Erreur : Fichier introuvable à {data_path}. Assurez-vous d'avoir exécuté
le script SQL et enregistré le CSV.")
    exit()

def calculer_descripteurs(smiles):
```

```
    mol = Chem.MolFromSmiles(smiles)
    if mol is None:
        return None, None, None  # Gérer les chaînes SMILES invalides
    mw = Descriptors.MolWt(mol)
    logp = Descriptors.MolLogP(mol)
    hbd = Descriptors.NumHDonors(mol)
    return mw, logp, hbd

df[['MW', 'LogP', 'HBD']] = df['canonical_smiles'].apply(lambda x:
pd.Series(calculer_descripteurs(x)))

# Gérer les valeurs None potentielles (SMILES invalides)
df = df.dropna(subset=['MW', 'LogP', 'HBD'])

print(df.head())
```

Example 4: Basic Visualization (Python)

```
# File: project_root/notebooks/Topic_CheMBL_35_8_3_visualization.ipynb
import pandas as pd
import os
import matplotlib.pyplot as plt
import seaborn as sns

base_path = "project_root" # Replace with your actual project root path (absolute)
data_path = os.path.join(base_path, "data", "chembl_205_ic50.csv")

try:
    df = pd.read_csv(data_path)
except FileNotFoundError:
    print(f"Error: File not found at {data_path}.  Make sure you ran the SQL script
and saved the CSV.")
    exit()

def calculate_descriptors(smiles):
    mol = Chem.MolFromSmiles(smiles)
    if mol is None:
        return None, None, None
    mw = Descriptors.MolWt(mol)
    logp = Descriptors.MolLogP(mol)
    hbd = Descriptors.NumHDonors(mol)
    return mw, logp, hbd

df[['MW', 'LogP', 'HBD']] = df['canonical_smiles'].apply(lambda x:
pd.Series(calculate_descriptors(x)))
df = df.dropna(subset=['MW', 'LogP', 'HBD'])

# Convert IC50 to numeric and take the log
df['IC50'] = pd.to_numeric(df['standard_value'], errors='coerce')
df['pIC50'] = -np.log10(df['IC50'] * 1e-9) # Convert nM to Molar and take -log10
df = df.dropna(subset=['pIC50'])

# Scatter plot
plt.figure(figsize=(8, 6))
sns.scatterplot(x='LogP', y='pIC50', data=df)
plt.xlabel('LogP')
plt.ylabel('pIC50')
plt.title('LogP vs. pIC50')
plt.show()
```

```
# Histogram
plt.figure(figsize=(8, 6))
sns.histplot(df['pIC50'])
plt.xlabel('pIC50')
plt.title('Distribution of pIC50 values')
plt.show()
```

(French):

```
# Fichier : project_root/notebooks/Topic_CheMBL_35_8_3_visualization.ipynb
import pandas as pd
import os
import matplotlib.pyplot as plt
import seaborn as sns

base_path = "project_root" # Remplacez par votre chemin de base de projet réel
(absolu)
data_path = os.path.join(base_path, "data", "chembl_205_ic50.csv")

try:
    df = pd.read_csv(data_path)
except FileNotFoundError:
    print(f"Erreur : Fichier introuvable à {data_path}. Assurez-vous d'avoir exécuté
le script SQL et enregistré le CSV.")
    exit()

def calculer_descripteurs(smiles):
    mol = Chem.MolFromSmiles(smiles)
    if mol is None:
        return None, None, None
    mw = Descriptors.MolWt(mol)
    logp = Descriptors.MolLogP(mol)
    hbd = Descriptors.NumHDonors(mol)
    return mw, logp, hbd

df[['MW', 'LogP', 'HBD']] = df['canonical_smiles'].apply(lambda x:
pd.Series(calculer_descripteurs(x)))
df = df.dropna(subset=['MW', 'LogP', 'HBD'])

# Convertir IC50 en numérique et prendre le log
df['IC50'] = pd.to_numeric(df['standard_value'], errors='coerce')
df['pIC50'] = -np.log10(df['IC50'] * 1e-9) # Convertir nM en Molaire et prendre -log10
df = df.dropna(subset=['pIC50'])

# Diagramme de dispersion
plt.figure(figsize=(8, 6))
sns.scatterplot(x='LogP', y='pIC50', data=df)
plt.xlabel('LogP')
plt.ylabel('pIC50')
plt.title('LogP vs. pIC50')
plt.show()

# Histogramme
plt.figure(figsize=(8, 6))
sns.histplot(df['pIC50'])
plt.xlabel('pIC50')
plt.title('Distribution des valeurs pIC50')
plt.show()
```

Example 5: Simple Linear Regression (Python)

```python
# File: project_root/notebooks/Topic_CheMBL_35_8_4_regression.ipynb
import pandas as pd
import os
import numpy as np
from rdkit import Chem
from rdkit.Chem import Descriptors
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score

base_path = "project_root" # Replace with your actual project root path (absolute)
data_path = os.path.join(base_path, "data", "chembl_205_ic50.csv")

try:
    df = pd.read_csv(data_path)
except FileNotFoundError:
    print(f"Error: File not found at {data_path}.  Make sure you ran the SQL script
and saved the CSV.")
    exit()

def calculate_descriptors(smiles):
    mol = Chem.MolFromSmiles(smiles)
    if mol is None:
        return None, None, None
    mw = Descriptors.MolWt(mol)
    logp = Descriptors.MolLogP(mol)
    hbd = Descriptors.NumHDonors(mol)
    return mw, logp, hbd

df[['MW', 'LogP', 'HBD']] = df['canonical_smiles'].apply(lambda x:
pd.Series(calculate_descriptors(x)))
df = df.dropna(subset=['MW', 'LogP', 'HBD'])

# Convert IC50 to numeric and take the log
df['IC50'] = pd.to_numeric(df['standard_value'], errors='coerce')
df['pIC50'] = -np.log10(df['IC50'] * 1e-9) # Convert nM to Molar and take -log10
df = df.dropna(subset=['pIC50'])


# Prepare data for modeling
X = df[['MW', 'LogP', 'HBD']]
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
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}")
```

(French):

```
# Fichier : project_root/notebooks/Topic_CheMBL_35_8_4_regression.ipynb
import pandas as pd
import os
import numpy as np
from rdkit import Chem
from rdkit.Chem import Descriptors
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score

base_path = "project_root" # Remplacez par votre chemin de base de projet réel
(absolu)
data_path = os.path.join(base_path, "data", "chembl_205_ic50.csv")

try:
    df = pd.read_csv(data_path)
except FileNotFoundError:
    print(f"Erreur : Fichier introuvable à {data_path}. Assurez-vous d'avoir exécuté
le script SQL et enregistré le CSV.")
    exit()

def calculer_descripteurs(smiles):
    mol = Chem.MolFromSmiles(smiles)
    if mol is None:
        return None, None, None
    mw = Descriptors.MolWt(mol)
    logp = Descriptors.MolLogP(mol)
    hbd = Descriptors.NumHDonors(mol)
    return mw, logp, hbd

df[['MW', 'LogP', 'HBD']] = df['canonical_smiles'].apply(lambda x:
pd.Series(calculer_descripteurs(x)))
df = df.dropna(subset=['MW', 'LogP', 'HBD'])

# Convertir IC50 en numérique et prendre le log
df['IC50'] = pd.to_numeric(df['standard_value'], errors='coerce')
df['pIC50'] = -np.log10(df['IC50'] * 1e-9) # Convertir nM en Molaire et prendre -log10
df = df.dropna(subset=['pIC50'])

# Préparer les données pour la modélisation
X = df[['MW', 'LogP', 'HBD']]
y = df['pIC50']

# Diviser les données en ensembles d'entraînement et de test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Entraîner un modèle de régression linéaire
model = LinearRegression()
model.fit(X_train, y_train)

# Faire des prédictions
y_pred = model.predict(X_test)

# Évaluer le modèle
```

```
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print(f"Erreur quadratique moyenne (MSE) : {mse}")
print(f"Coefficient de détermination (R-squared) : {r2}")
```

Key Improvements and Considerations:

- **Error Handling:** The Python code includes `try...except` blocks to handle the `FileNotFoundError` and `None` values from invalid SMILES
- **Absolute Paths:** Using absolute paths for `base_path` makes the code more robust and independent of the current working directory. **THIS IS VERY IMPORTANT.**
- **SQL Regular Expression:** The SQL query uses `SIMILAR TO` for numeric checking. If `SIMILAR TO` is not available, you can use a `CASE` statement or cast the value to text before matching.
- **Data Cleaning:** The Python code cleans up any rows with invalid SMILES strings by using `df.dropna()`.
- **pIC50 Calculation:** The code converts IC50 values from nM to pIC50, which is a more common scale for analyzing bioactivity data.
- **Modularization:** Consider moving the `calculate_descriptors` function into a separate `src` directory module for better organization and reusability.
- **Dependencies:** Make sure you have all the necessary libraries installed (`pandas`, `rdkit`, `scikit-learn`, `matplotlib`, `seaborn`). You can install them using `pip install pandas rdkit scikit-learn matplotlib seaborn`.

Remember to replace `"project_root"` with the correct path. Good luck! Let me know if you have more questions.