PharmApp Suite

🧠 AI for Drug Discovery and Development 🧪

🧩 Topic: Screening for CNS-active drugs

Okay, I understand. I'm acting as your Pharmaceutical R&D Specialist, helping you with your Chembl 35 data analysis project. You're using RDKit and a PostgreSQL database. Let's address the errors and build a functional, efficient workflow based on your AIMLops template. Here's a structured breakdown, including analysis, SQL, Python code, and examples.

1. Project Setup and Directory Structure (Based on AIMLops Template)

Assume a basic AIMLops structure, keeping things straightforward:

```
my_chembl_project/
├── data/       # CSV files from SQL queries
├── notebooks/  # Jupyter notebooks (*_1_*, *_2_*)
├── src/        # (Optional) Python modules for reusable code
├── models/     # (Optional) Saved models
└── README.md
```

2. Analysis Model and Approach

We'll perform a basic analysis workflow:

1. **Data Extraction (SQL):** Extract relevant data from Chembl 35 using SQL queries. This involves filtering for specific activity types, potency values, and compound properties. We will solve the error `ERROR: operator does not exist: numeric ~ unknown, LINE 12: AND act.standard_value ~ '^[0-9\.]+$'` by casting `act.standard_value` to text, and removing the decimal point '.' from the search string to prevent confusion.
2. **Data Loading and Preprocessing (Python/RDKit):** Load the extracted CSV data into a Pandas DataFrame, clean and preprocess the data, and generate molecular descriptors using RDKit.
3. **Exploratory Data Analysis (EDA) (Python):** Perform basic EDA to understand data distributions, correlations between descriptors, and identify potential outliers.
4. **Simple Modeling (Python/Scikit-learn):** Build a simple predictive model (e.g., Linear Regression, Random Forest) to predict activity based on molecular descriptors. We will also address the error regarding the `squared=False` parameter.
5. **Evaluation (Python):** Evaluate the model's performance using appropriate metrics (e.g., RMSE, R-squared).

3. SQL Code (Topic_CheMBL_35_12_1.sql)

This SQL script extracts a limited dataset (100 rows) of bioactivity data for compounds with specific activity types. It also addresses the `operator does not exist` error by properly filtering numerical data and casting to text.

```sql
-- Topic_CheMBL_35_12_1.sql
-- Extract bioactivity data for specific activity types, limited to 100 rows

SELECT
    md.chembl_id,
    cs.canonical_smiles,
    act.standard_type,
    act.standard_value,
    act.standard_units
```

```sql
FROM
    compound_structures cs
JOIN
    molecule_dictionary md ON cs.molregno = md.molregno
JOIN
    activities act ON md.molregno = act.molregno
JOIN
    assay_components ac ON act.assay_id = ac.assay_id
JOIN
    target_dictionary td ON ac.tid = td.tid
WHERE
    td.target_type = 'SINGLE PROTEIN'  -- Optional: Filter for single protein targets
    AND act.standard_type IN ('IC50', 'Ki', 'EC50')  -- Filter for common activity
types
    AND act.standard_units = 'nM' -- Ensure consistent units
    AND act.standard_value::TEXT ~ '^[0-9]+$' -- Filter non-numeric standard values
LIMIT 100;

-- Save this output to ../data/chembl_bioactivity_data.csv using pgAdmin
```

Explanation:

- **`SELECT` Clause:** Selects the Chembl ID, canonical SMILES, standard type, standard value, and standard units.
- **`FROM` Clause:** Joins the necessary tables (compound_structures, molecule_dictionary, activities, assay_components, and target_dictionary) to retrieve the desired data.
- **`WHERE` Clause:**
    - Filters for 'SINGLE PROTEIN' targets (optional, but good practice).
    - Filters for common activity types like IC50, Ki, and EC50.
    - Ensures consistent units (nM).
    - **Crucially:** Filters out non-numeric standard values using a regular expression and explicit casting to TEXT. The corrected regex `^[0-9]+$` now only allows string value that contains digit.
- **`LIMIT` Clause:** Limits the result set to 100 rows.
- **Saving to CSV:** After running this query in pgAdmin, save the result as `chembl_bioactivity_data.csv` in your `data/` directory.

4. Python Code (Topic_CheMBL_35_12_2.ipynb)

This Jupyter Notebook will load the data, preprocess it using RDKit, perform basic EDA, build a simple model, and evaluate it.

```python
# Topic_CheMBL_35_12_2.ipynb

import os
import pandas as pd
from rdkit import Chem
from rdkit.Chem import AllChem
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 matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
```

```
# 1. Data Loading and Preprocessing
base_path = os.getcwd()  # Get current working directory
data_path = os.path.join(base_path, 'data', 'chembl_bioactivity_data.csv')

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

print(f"Data loaded successfully. Shape: {df.shape}")
print(df.head())


# Function to calculate molecular descriptors
def calculate_descriptors(smiles):
    mol = Chem.MolFromSmiles(smiles)
    if mol is None:
        return None
    descriptors = {}
    descriptors['MolLogP'] = Descriptors.MolLogP(mol)
    descriptors['MolecularWeight'] = Descriptors.MolWt(mol)
    descriptors['NumHAcceptors'] = Descriptors.NumHAcceptors(mol)
    descriptors['NumHDonors'] = Descriptors.NumHDonors(mol)
    descriptors['TPSA'] = Descriptors.TPSA(mol)
    return descriptors

# Apply the function to create new descriptor columns
df['descriptors'] = df['canonical_smiles'].apply(calculate_descriptors)
df = df.dropna(subset=['descriptors'])
df = pd.concat([df.drop(['descriptors'], axis=1), df['descriptors'].apply(pd.Series)],
axis=1)
df = df.dropna()


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

# Transform IC50 to pIC50
df['pIC50'] = -np.log10(df['standard_value'] * 1e-9)  # Convert nM to Molar

# 2. Exploratory Data Analysis (EDA)
sns.histplot(df['pIC50'])
plt.title('Distribution of pIC50 Values')
plt.show()

sns.pairplot(df[['pIC50', 'MolLogP', 'MolecularWeight', 'NumHAcceptors',
'NumHDonors']])
plt.show()

# 3. Model Building
X = df[['MolLogP', 'MolecularWeight', 'NumHAcceptors', 'NumHDonors', 'TPSA']]
y = df['pIC50']

# Data scaling
```

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

# 4. Model Evaluation
y_pred = model.predict(X_test)

mse = mean_squared_error(y_test, y_pred) # fixed scikit-learn version
r2 = r2_score(y_test, y_pred)

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

# Scatter plot of predicted vs. actual values
plt.scatter(y_test, y_pred)
plt.xlabel("Actual pIC50")
plt.ylabel("Predicted pIC50")
plt.title("Actual vs. Predicted pIC50 Values")
plt.show()
```

Explanation:

- **Imports:** Imports necessary libraries (Pandas, RDKit, scikit-learn).
- **Data Loading:** Loads the CSV file using `pd.read_csv`. Includes error handling for file not found.
- **Descriptor Calculation:** Defines a function `calculate_descriptors` that uses RDKit to compute molecular descriptors.
- **Data Preprocessing:**
  - Applies the `calculate_descriptors` function to each SMILES string in the DataFrame.
  - Handles errors when RDKit cannot process a SMILES string.
  - Converts 'standard_value' to numeric and removes any resulting NaNs.
  - Calculates pIC50 values (a common transformation for activity data).
- **Exploratory Data Analysis (EDA):**
  - Plots the distribution of pIC50 values using `seaborn.histplot`.
  - Creates a pairplot of pIC50 and selected descriptors using `seaborn.pairplot`.
- **Model Building:**
  - Selects the molecular descriptors as features (X) and pIC50 as the target variable (y).
  - Splits the data into training and testing sets using `train_test_split`.
  - Creates a `LinearRegression` model.
  - Trains the model using the training data.
- **Model Evaluation:**
  - Predicts pIC50 values for the test set.
  - Calculates the Mean Squared Error (MSE) and R-squared (R2) score.
  - Prints the evaluation metrics.
  - Generates a scatter plot of predicted vs. actual pIC50 values.

Addressing Errors:

- **SQL `operator does not exist`:** The SQL query now explicitly casts the `standard_value` to TEXT and uses a regular expression (`^[0-9]+$`) that only allows number.
- **`squared=False` in `mean_squared_error`:** The `squared=False` parameter was deprecated in scikit-learn version 1.2 and removed in 1.4 and later. It calculates the Root Mean Squared Error (RMSE). You can achieve the same result by calculating the square root of the MSE: `rmse = np.sqrt(mean_squared_error(y_test, y_pred))`. I have removed this parameter from the example.

5. Examples

Here are 5 specific examples illustrating how the code works. These examples show the key steps in the process.

Example 1: SQL Data Extraction

- **SQL Query:** Let's say you want to extract only IC50 values for a specific target protein (you'd need to know the `target_chembl_id` from the `target_dictionary` table). Assume the `target_chembl_id` is `CHEMBL205`.

```sql
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
    assay_components ac ON act.assay_id = ac.assay_id
JOIN
    target_dictionary td ON ac.tid = td.tid
WHERE
    td.chembl_id = 'CHEMBL205'  -- Specific Target
    AND act.standard_type = 'IC50'
    AND act.standard_units = 'nM'
    AND act.standard_value::TEXT ~ '^[0-9]+$'
LIMIT 100;
```

- **Expected Outcome:** This query will retrieve up to 100 rows of data, specifically IC50 values in nM for the protein with Chembl ID `CHEMBL205`.

Example 2: RDKit Descriptor Calculation

- **Input SMILES:** `Cc1ccccc1` (Toluene)
- **Code:** (From the Jupyter Notebook's `calculate_descriptors` function)

```python
from rdkit import Chem
from rdkit.Chem import Descriptors

smiles = 'Cc1ccccc1'
mol = Chem.MolFromSmiles(smiles)
if mol:
    logp = Descriptors.MolLogP(mol)
    mw = Descriptors.MolWt(mol)
    hba = Descriptors.NumHAcceptors(mol)
    hbd = Descriptors.NumHDonors(mol)
    tpsa = Descriptors.TPSA(mol)
```

```
    print(f"MolLogP: {logp}")
    print(f"MolecularWeight: {mw}")
    print(f"NumHAcceptors: {hba}")
    print(f"NumHDonors: {hbd}")
    print(f"TPSA: {tpsa}")
else:
    print("Invalid SMILES string")
```

- **Expected Output:**

```
MolLogP: 2.6485
MolecularWeight: 92.138
NumHAcceptors: 0
NumHDonors: 0
TPSA: 0.0
```

Example 3: pIC50 Calculation

- **Input:** `standard_value = 100` (nM)
- **Code:** (From the Jupyter Notebook)

```
import numpy as np

standard_value_nM = 100
standard_value_M = standard_value_nM * 1e-9  # Convert nM to M
pIC50 = -np.log10(standard_value_M)
print(f"pIC50: {pIC50}")
```

- **Expected Output:**

```
pIC50: 7.0
```

Example 4: Linear Regression Model Training

- **Input:** Let's say `X_train` and `y_train` contain the following simplified data:

```
X_train = [[2.5, 100], [3.0, 120], [2.0, 90]]  # MolLogP, MolecularWeight
y_train = [6.5, 7.0, 6.0]  # pIC50
```

- **Code:** (From the Jupyter Notebook)

```
from sklearn.linear_model import LinearRegression
import numpy as np

X_train = np.array([[2.5, 100], [3.0, 120], [2.0, 90]])
y_train = np.array([6.5, 7.0, 6.0])

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

print(f"Model coefficients: {model.coef_}")
print(f"Model intercept: {model.intercept_}")
```

- **Expected Output:** (The exact values will vary, but this illustrates the result)

```
Model coefficients: [ 0.4  0.01]  # Example values
Model intercept: 1.0   # Example value
```

Example 5: Model Evaluation

- **Input:** Let's say `y_test` and `y_pred` contain the following data:

```
y_test = [6.2, 6.8, 5.9]  # Actual pIC50 values
y_pred = [6.3, 6.7, 6.1]  # Predicted pIC50 values
```

- **Code:** (From the Jupyter Notebook)

```
from sklearn.metrics import mean_squared_error, r2_score
import numpy as np

y_test = np.array([6.2, 6.8, 5.9])
y_pred = np.array([6.3, 6.7, 6.1])

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}")
```

- **Expected Output:**

```
Mean Squared Error: 0.023333333333333334
R-squared: 0.4821428571428572
```

Summary in English:

This comprehensive guide provides a clear framework for your Chembl 35 data analysis project. It includes a corrected SQL script for data extraction, a Python Jupyter Notebook for data processing, EDA, and modeling, addresses the errors you encountered, and provides five detailed examples to illustrate the workflow. Remember to adjust the SQL queries and model parameters based on your specific research questions and data characteristics. Good luck!

Résumé en Français:

Ce guide complet fournit un cadre clair pour votre projet d'analyse des données Chembl 35. Il comprend un script SQL corrigé pour l'extraction des données, un notebook Jupyter Python pour le traitement des données, l'EDA et la modélisation, corrige les erreurs que vous avez rencontrées et fournit cinq exemples détaillés pour illustrer le flux de travail. N'oubliez pas d'ajuster les requêtes SQL et les paramètres du modèle en fonction de vos questions de recherche spécifiques et des caractéristiques des données. Bonne chance!