PharmApp Suite

🧠 AI for Drug Discovery and Development 💉

🧩 Topic: Finding targets with dual antagonist/agonist activity

Okay, let's break down this Chembl 35 data analysis project using RDKit within your AIMLops template. I'll provide a structured approach with SQL queries, Python code (with RDKit), error resolution, and five examples.

Analysis Model Overview

The overall aim is to extract relevant data from the ChEMBL 35 database, process it using RDKit (most likely for structure-based analysis or featurization), and then perform some form of data analysis or modeling. This analysis will be conducted on a subset of 100 rows for efficiency.

Here's a breakdown of the key steps:

1. **Data Extraction (SQL):** We'll use SQL to connect to your PostgreSQL database, extract the necessary information from the ChEMBL 35 database tables (likely involving `activities`, `molecule_dictionary`, `assays`, and potentially others), filter it based on activity data, and save it to a CSV file. The error `ERROR: operator does not exist: numeric ~ unknown, LINE 12: AND act.standard_value ~ '^[0-9\.]+$'` suggests you're trying to use a regular expression operator (~) on a numeric column. We will use `CAST` method to convert the numeric column to type `TEXT`
2. **Data Loading and Processing (Python/RDKit):** The Python code in your Jupyter Notebook will load the data from the CSV file, clean it (handle missing values, inconsistencies, etc.), use RDKit to process the chemical structures (converting SMILES strings to RDKit molecules, calculating descriptors, etc.), and prepare the data for analysis.
3. **Analysis/Modeling (Python/Scikit-learn):** Depending on the goal of your analysis, you will use scikit-learn (or other libraries) to build a model (e.g., regression, classification, clustering). Because the version of your Scikit-learn doesn't support parameters squared=False in the mean_squared_error function. We will remove the parameters squared=False in the mean_squared_error function
4. **Example Scenarios:** I'll provide five examples of potential analyses you could perform.

Directory Structure (Based on AIMLops):

Assuming a structure something like this:

```
project_root/
├── data/             # CSV files extracted from the database
├── notebooks/        # Jupyter notebooks
│   ├── Topic_CheMBL_35_39_1_data_extraction_and_processing.ipynb
│   ├── Topic_CheMBL_35_39_2_analysis_and_modeling.ipynb
├── sql/              # SQL scripts
├── models/           # Saved models
├── ...
```

1. SQL Code (sql/Topic_CheMBL_35_39_extract_data.sql):

```sql
-- Extracting data from ChEMBL 35 for Topic_CheMBL_35_39
-- Limiting to 100 rows for demonstration purposes

SELECT
    md.chembl_id,
    md.pref_name,
```

```
    act.standard_type,
    act.standard_relation,
    act.standard_value,
    act.standard_units,
    act.pchembl_value,
    assays.assay_type,
    assays.assay_class,
    md.mol_weight,
    md.alogp,
    md.num_hba,
    md.num_hbd,
    md.psa,
    m.canonical_smiles

FROM
    activities act
JOIN
    molecule_dictionary md ON act.molregno = md.molregno
JOIN
    assays ON act.assay_id = assays.assay_id
JOIN
    compound_structures m ON md.molregno = m.molregno

WHERE
    act.standard_type IN ('IC50', 'Ki', 'EC50')  -- Focus on common activity types
    AND act.standard_relation = '='   -- Focus on direct measurements
    AND act.standard_units = 'nM'     -- Standardize units
    AND CAST(act.standard_value AS TEXT) ~ '^[0-9\.]+$'  -- Filter for numeric
standard_value
    AND act.pchembl_value IS NOT NULL  -- Requires pChEMBL value for further filtering

ORDER BY
    act.pchembl_value DESC -- Example Ordering by pChEMBL Value

LIMIT 100;
```

Explanation:

- **`SELECT` Clause:** Specifies the columns you want to retrieve. I've included compound identifiers, activity data, assay information, and some basic molecular properties. The canonical SMILES structure is very important.
- **`FROM` and `JOIN` Clauses:** Connect the necessary tables (`activities`, `molecule_dictionary`, `assays`, `compound_structures`) based on their foreign key relationships.
- **`WHERE` Clause:**
    - Filters for specific `standard_type` values (IC50, Ki, EC50). You can adjust this based on your research question.
    - Filters for `standard_relation = '='` to ensure direct measurements.
    - Filters for `standard_units = 'nM'` for consistency.
    - **`CAST(act.standard_value AS TEXT) ~ '^[0-9\.]+$'`**: This is the fix for the error you encountered. It casts the `standard_value` column to `TEXT` before applying the regular expression match (`~`). The regular expression `^[0-9\.]+$` ensures that the `standard_value` consists only of numbers and periods (for decimal values).
    - `act.pchembl_value IS NOT NULL`: requires the pChembl value so that there are no null values.
- **`ORDER BY` Clause:** Orders the results by `pchembl_value` in descending order.
- **`LIMIT 100` Clause:** Restricts the output to the first 100 rows. **Important:** Remove or increase this limit once you've verified the code is working correctly.

How to run the SQL and save to CSV:

1. Connect to your PostgreSQL database using pgAdmin (or your preferred database client).
2. Open the `Topic_CheMBL_35_39_extract_data.sql` file in pgAdmin.
3. Execute the query.
4. Export the result set as a CSV file. Save it to `data/Topic_CheMBL_35_39_chembl_data.csv` (or a similar name).

2. Python Code (Notebook: notebooks/Topic_CheMBL_35_39_1_data_extraction_and_processing.ipynb):

```
import os
import pandas as pd
from rdkit import Chem
from rdkit.Chem import Descriptors
import numpy as np  # Import NumPy

# Define base path
base_path = os.getcwd()  # Or manually set if needed

# Data loading
data_file = os.path.join(base_path, 'data', 'Topic_CheMBL_35_39_chembl_data.csv')
try:
    df = pd.read_csv(data_file)
except FileNotFoundError:
    print(f"Error: File not found at {data_file}.  Make sure the SQL query was run and
the CSV file was saved correctly.")
    exit()

print(f"Dataframe loaded with {len(df)} rows")

# Data Cleaning and Preprocessing
# Handle missing values (replace with NaN and then impute or drop)
df = df.replace([np.inf, -np.inf], np.nan)  # Replace inf with NaN
df = df.dropna(subset=['canonical_smiles', 'standard_value', 'pchembl_value']) #
Remove rows with missing SMILES, standard_value, or pchembl_value
df['standard_value'] = pd.to_numeric(df['standard_value'], errors='coerce') # ensure
numeric
df['pchembl_value'] = pd.to_numeric(df['pchembl_value'], errors='coerce') # ensure
numeric
df = df.dropna(subset=['standard_value', 'pchembl_value']) # Remove rows where
conversion failed

# RDKit Processing: Create RDKit molecules and calculate descriptors
def smiles_to_mol(smiles):
    try:
        mol = Chem.MolFromSmiles(smiles)
        if mol is None:
            return None
        #Sanitize the molecule to fix aromatic bonds and Kekulize structure
        Chem.SanitizeMol(mol)
        return mol
    except Exception as e:
        print(f"Error processing SMILES: {smiles}, error: {e}")
        return None

df['molecule'] = df['canonical_smiles'].apply(smiles_to_mol)
df = df.dropna(subset=['molecule'])  # Remove rows where molecule creation failed
```

```
def calculate_descriptors(mol):
    try:
        mol_wt = Descriptors.MolWt(mol)
        logp = Descriptors.MolLogP(mol)
        num_hba = Descriptors.NumHAcceptors(mol)
        num_hbd = Descriptors.NumHDonors(mol)
        tpsa = Descriptors.TPSA(mol)

        return pd.Series([mol_wt, logp, num_hba, num_hbd, tpsa])
    except:
        return pd.Series([None, None, None, None, None]) # or np.nan

df[['rdkit_MW', 'rdkit_LogP', 'rdkit_HBA', 'rdkit_HBD', 'rdkit_TPSA']] =
df['molecule'].apply(calculate_descriptors)

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

# Save the processed data to a new CSV (optional)
processed_data_file = os.path.join(base_path, 'data',
'Topic_CheMBL_35_39_chembl_data_processed.csv')
df.to_csv(processed_data_file, index=False)
print(f"Processed data saved to {processed_data_file}")
```

Explanation:

- **Imports:** Imports necessary libraries (os, pandas, RDKit).
- **`base_path`:** Sets the base path for file operations.
- **Data Loading:** Loads the CSV file into a Pandas DataFrame. Includes error handling for the case where the CSV file is missing.
- **Data Cleaning:**
    - Handles missing values using `df.dropna()`. Adjust the columns based on your data and analysis. This is *crucial*.
    - Converts 'standard_value' and 'pchembl_value' to numeric using `pd.to_numeric`. This is important for calculations.
    - Removes missing values again just to make sure
- **RDKit Processing:**
    - **`smiles_to_mol` function:** Converts SMILES strings to RDKit molecule objects. Includes error handling. `Chem.SanitizeMol(mol)` function is added to sanitize the molecule and to fix aromatic bonds and Kekulize structure. This addresses potential issues with SMILES strings that may not be perfectly well-formed.
    - **`calculate_descriptors` function:** Calculates a set of common molecular descriptors using RDKit. You can add or remove descriptors as needed.
    - Applies the functions to the DataFrame.
- **Saving Processed Data:** Saves the processed DataFrame to a new CSV file.

3. Analysis and Modeling (Notebook: notebooks/Topic_CheMBL_35_39_2_analysis_and_modeling.ipynb):

```
import os
import pandas as pd
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

# Define base path
```

```
base_path = os.getcwd()

# Load the processed data
processed_data_file = os.path.join(base_path, 'data',
'Topic_CheMBL_35_39_chembl_data_processed.csv')
try:
    df = pd.read_csv(processed_data_file)
except FileNotFoundError:
    print(f"Error: File not found at {processed_data_file}.  Make sure the processing
notebook was run.")
    exit()

# Select features and target variable
features = ['rdkit_MW', 'rdkit_LogP', 'rdkit_HBA', 'rdkit_HBD', 'rdkit_TPSA']
target = 'pchembl_value'

# Drop rows with missing values in selected features
df = df.dropna(subset=features + [target])

# Prepare data
X = df[features]
y = df[target]

# Data scaling (important for many models)
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_scaled, 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) # Remove the squared=False
r2 = r2_score(y_test, y_pred)

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

# Optional: Save the model
import joblib
model_file = os.path.join(base_path, 'models',
'Topic_CheMBL_35_39_linear_regression_model.pkl')
joblib.dump(model, model_file)
print(f"Model saved to {model_file}")
```

Explanation:

- **Imports:** Imports necessary scikit-learn modules.
- **Data Loading:** Loads the processed data.
- **Feature Selection:** Selects the features (RDKit descriptors) and the target variable (`pchembl_value`). *Adapt this to your specific research question!*
- **Data Preprocessing:**

    - Handles missing values (again, ensure no NaNs).
    - **Data Scaling:** Uses `StandardScaler` to scale the features. This is crucial for many machine learning algorithms.
- **Train/Test Split:** Splits the data into training and testing sets.
- **Model Training:** Creates a `LinearRegression` model and trains it on the training data.
- **Prediction and Evaluation:** Makes predictions on the test set and evaluates the model using Mean Squared Error (MSE) and R-squared.
- **Model Saving (Optional):** Saves the trained model to a file using `joblib`.

4. Error Explanation:

- **`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 operator (~) on a numeric column. The ~ operator is for string matching. The fix is to cast the `standard_value` column to `TEXT` using `CAST(act.standard_value AS TEXT)` before applying the regular expression.
- **`old scikit-learn version does not support parameters squared=False in the mean_squared_error function`:** This means your scikit-learn version is older, and the `squared` parameter is not available. The fix is simply to remove the `squared=False` argument from the `mean_squared_error` function call. The default behavior without the parameter is the same as setting `squared=True`, which returns the MSE.

5. Example Analyses (Based on the Code):

Here are five examples of how you could extend this analysis, building on the code provided:

1. **QSAR Modeling (Quantitative Structure-Activity Relationship):**
    - **Goal:** Predict `pchembl_value` based on RDKit descriptors.
    - **Method:** Use more sophisticated regression models from scikit-learn, such as Random Forest Regressor, Gradient Boosting Regressor, or Support Vector Regression. Experiment with different feature sets and hyperparameter tuning. Use cross-validation for robust model evaluation.
    - **Notebook:** Extend `Topic_CheMBL_35_39_2_analysis_and_modeling.ipynb`.
2. **Activity Classification:**
    - **Goal:** Classify compounds as "active" or "inactive" based on a threshold for `pchembl_value`.
    - **Method:** Create a binary classification target variable (e.g., `active = 1 if pchembl_value >= 6 else 0`). Use classification models like Logistic Regression, Random Forest Classifier, or Support Vector Machines. Evaluate performance using metrics like accuracy, precision, recall, and F1-score.
    - **Notebook:** Create a new notebook, `Topic_CheMBL_35_39_3_activity_classification.ipynb`.
3. **Clustering and Chemical Space Analysis:**
    - **Goal:** Identify clusters of compounds with similar properties.
    - **Method:** Use clustering algorithms like k-means or hierarchical clustering based on RDKit descriptors or other molecular properties. Visualize the clusters using dimensionality reduction techniques like PCA or t-SNE.
    - **Notebook:** Create a new notebook, `Topic_CheMBL_35_39_4_clustering_and_chemical_space.ipynb`.
4. **Feature Importance Analysis:**
    - **Goal:** Determine which RDKit descriptors are most important for predicting activity.

    - **Method:** Use feature importance methods available in scikit-learn models (e.g., `feature_importances_` attribute in Random Forest). Alternatively, use permutation importance techniques.
    - **Notebook:** Extend `Topic_CheMBL_35_39_2_analysis_and_modeling.ipynb`.
5. **Scaffold Analysis:**
    - **Goal:** Identify and analyze the prevalence of different molecular scaffolds within the dataset.
    - **Method:** Use RDKit's `GetScaffoldForMol` function to extract the Bemis-Murcko scaffold from each molecule. Analyze the frequency of different scaffolds and investigate their relationship to activity. This can help identify promising core structures for drug development.
    - **Notebook:** Create a new notebook, `Topic_CheMBL_35_39_5_scaffold_analysis.ipynb`.

Example Code Snippet (Scaffold Analysis - Add to notebooks/Topic_CheMBL_35_39_5_scaffold_analysis.ipynb):

```
import os
import pandas as pd
from rdkit import Chem
from rdkit.Chem import MurckoDecomposition

# Load the processed data
base_path = os.getcwd()
processed_data_file = os.path.join(base_path, 'data',
'Topic_CheMBL_35_39_chembl_data_processed.csv')
df = pd.read_csv(processed_data_file)

def get_scaffold(mol):
    if mol is None:
        return None
    try:
        core = MurckoDecomposition.GetScaffoldForMol(mol)
        return Chem.MolToSmiles(core)
    except:
        return None

df['scaffold'] = df['molecule'].apply(get_scaffold)
df = df.dropna(subset=['scaffold'])

scaffold_counts = df['scaffold'].value_counts()
print(scaffold_counts.head(10)) # Show the top 10 most frequent scaffolds
```

Important Considerations:

- **Data Exploration:** Spend time exploring your data before building models. Look at distributions of variables, correlations, and potential outliers.
- **Feature Engineering:** Experiment with different RDKit descriptors and combinations of features.
- **Model Selection:** Try different machine learning models and compare their performance.
- **Hyperparameter Tuning:** Optimize the hyperparameters of your chosen model using techniques like grid search or random search.
- **Validation:** Use proper validation techniques (e.g., cross-validation) to ensure your model generalizes well to unseen data.
- **Interpretability:** Strive to build models that are interpretable, so you can understand why they are making certain predictions.

En Français:

Analyse et Guide en Français

Ce projet vise à analyser les données de ChEMBL 35 en utilisant RDKit pour la recherche et le développement de médicaments.

1. Analyse Détaillée du Modèle d'Analyse

L'objectif principal est d'extraire des données pertinentes de la base de données ChEMBL 35, de les traiter avec RDKit (pour l'analyse basée sur la structure ou la vectorisation), puis d'effectuer une analyse ou une modélisation des données. Cette analyse sera effectuée sur un sous-ensemble de 100 lignes pour plus d'efficacité.

Étapes clés :

1. **Extraction des Données (SQL) :** Nous utiliserons SQL pour nous connecter à votre base de données PostgreSQL, extraire les informations nécessaires des tables de ChEMBL 35 (impliquant probablement `activities`, `molecule_dictionary`, `assays`, etc.), filtrer les données en fonction des données d'activité et les enregistrer dans un fichier CSV.
2. **Chargement et Traitement des Données (Python/RDKit) :** Le code Python dans votre Jupyter Notebook chargera les données à partir du fichier CSV, les nettoiera (gestion des valeurs manquantes, des incohérences, etc.), utilisera RDKit pour traiter les structures chimiques (conversion des chaînes SMILES en molécules RDKit, calcul des descripteurs, etc.) et préparera les données pour l'analyse.
3. **Analyse/Modélisation (Python/Scikit-learn) :** En fonction de l'objectif de votre analyse, vous utiliserez scikit-learn (ou d'autres bibliothèques) pour construire un modèle (par exemple, régression, classification, clustering).
4. **Scénarios d'Exemples :** Je fournirai cinq exemples d'analyses potentielles que vous pourriez effectuer.

2. Code SQL, Python

(Voir le code SQL et Python ci-dessus. Les commentaires expliquent le fonctionnement.)

3. Exemples

(Voir les cinq exemples d'analyses ci-dessus.)

N'hésitez pas à poser d'autres questions au fur et à mesure que vous avancez dans le projet! Good luck!