PharmApp Suite

🧠 AI for Drug Discovery and Development 💉

🧩 Topic: Streamlit dashboard for bioactivity exploration

Okay, I understand. You're working on a Chembl 35 data analysis project using RDKit and PostgreSQL, following an AIMLops template. You're experiencing some errors and want guidance on how to proceed, with a focus on retrieving a limited dataset of 100 rows for faster experimentation.

Let's break down the analysis model, provide the code, address the errors, and create some examples.

1. Analysis Model:

The core of your project involves analyzing chemical and biological data from the ChEMBL database (version 35). This data likely includes information about:

- **Chemical Compounds:** Represented by SMILES strings or other chemical identifiers.
- **Bioactivity Data:** Measurements like IC50, Ki, Kd values, indicating how well a compound interacts with a target.
- **Targets:** Proteins or other biological entities that the compounds are designed to interact with.

Your analysis goals likely include:

- **Data Cleaning and Preparation:** Handling missing data, standardizing units, filtering for relevant data types (e.g., focusing on IC50 values).
- **Feature Engineering:** Generating molecular descriptors from the SMILES strings using RDKit. These descriptors capture various properties of the molecules (e.g., molecular weight, LogP, number of rings, etc.).
- **Exploratory Data Analysis (EDA):** Visualizing the data to understand distributions, identify outliers, and look for correlations between features and bioactivity.
- **Model Building:** Developing predictive models to estimate the bioactivity of new compounds based on their molecular descriptors. This could involve regression models (e.g., linear regression, random forests, support vector machines) or classification models (e.g., predicting whether a compound is active or inactive).
- **Model Evaluation:** Assessing the performance of the models using appropriate metrics (e.g., R-squared, RMSE for regression; accuracy, precision, recall, F1-score for classification).

Addressing Errors:

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

  This error means that you're trying to use a regular expression operator (~) on a numeric column (`act.standard_value`). PostgreSQL requires the column to be of a text type for regex matching. You need to cast the numeric column to text before applying the regex.

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

  This is a version compatibility issue. Either update your scikit-learn library to a version that supports `squared=False` (recommended) or remove the `squared=False` argument. Removing it will return the Mean Squared Error (MSE) instead of the Root Mean Squared Error (RMSE).

2. SQL and Python Code:

Here's the code, incorporating the fixes and limitations you specified:

SQL (`Topic_CheMBL_35_71.sql` - to be run in pgAdmin):

```sql
-- Select 100 rows of data from ChEMBL, focusing on IC50 values for a specific target.
-- Adjust the WHERE clause to target a specific protein or biological process if
needed.
-- The limit clause is used to retrieve 100 rows for faster experimentation.

SELECT
    md.molregno,
    cs.canonical_smiles,
    act.standard_value
FROM
    compound_structures cs
JOIN
    molecule_dictionary md ON cs.molregno = md.molregno
JOIN
    activities act ON md.molregno = act.molregno
JOIN
    assay_xref ax ON act.assay_id = ax.assay_id
JOIN
    assays a ON ax.assay_id = a.assay_id
WHERE
    act.standard_type = 'IC50'  -- Focus on IC50 values
    AND act.standard_units = 'nM'  -- Ensure units are consistent
    AND act.standard_relation = '=' --Only get equal relation
    AND act.standard_value IS NOT NULL
    AND a.confidence_score >= 8    -- High Confidence Data
    AND CAST(act.standard_value AS TEXT) ~ '^[0-9\.]+$'  -- Fix: Cast to text for
regex
LIMIT 100;
```

Explanation:

- `SELECT ... FROM ... JOIN ...`: Selects data from several ChEMBL tables and joins them based on common keys. We're selecting `molregno` (molecule registry number), `canonical_smiles` (SMILES string), and `standard_value` (IC50 value).
- `WHERE act.standard_type = 'IC50'`: Filters for IC50 values. You can change this to other bioactivity types if needed.
- `WHERE act.standard_units = 'nM'`: Filters for nanomolar units to ensure consistency.
- `WHERE a.confidence_score >= 8`: Filters for high-confidence data. Adjust this score based on your data quality requirements.
- `AND CAST(act.standard_value AS TEXT) ~ '^[0-9\.]+$'`: This is the crucial fix. It casts the `standard_value` column to `TEXT` before applying the regular expression to ensure that the value only consists of numbers and periods.
- `LIMIT 100`: Limits the result set to 100 rows.

Python Code (`Topic_CheMBL_35_71_1_Data_Analysis.ipynb`):

```python
import os
import pandas as pd
import psycopg2
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
```

```python
from sklearn.preprocessing import StandardScaler

# Define the base path for the project
base_path = "."  # Assuming the notebook is in the root of your project

# Database credentials
db_host = "192.168.206.136"
db_user = "rd"
db_pass = "rd"
db_name = "chembl_35"

# Function to connect to the database and execute a query
def connect_and_query(sql_query):
    try:
        conn = psycopg2.connect(host=db_host, user=db_user, password=db_pass,
database=db_name)
        cur = conn.cursor()
        cur.execute(sql_query)
        results = cur.fetchall()
        conn.close()
        return results
    except Exception as e:
        print(f"Error connecting to the database: {e}")
        return None

# Function to load data from CSV (if you prefer that to querying directly)
def load_data_from_csv(csv_file_path):
    try:
        df = pd.read_csv(csv_file_path)
        return df
    except FileNotFoundError:
        print(f"Error: CSV file not found at {csv_file_path}")
        return None
    except Exception as e:
        print(f"Error reading CSV file: {e}")
        return None


# SQL query to retrieve data
sql_query = """
SELECT
    md.molregno,
    cs.canonical_smiles,
    act.standard_value
FROM
    compound_structures cs
JOIN
    molecule_dictionary md ON cs.molregno = md.molregno
JOIN
    activities act ON md.molregno = act.molregno
JOIN
    assay_xref ax ON act.assay_id = ax.assay_id
JOIN
    assays a ON ax.assay_id = a.assay_id
WHERE
    act.standard_type = 'IC50'  -- Focus on IC50 values
    AND act.standard_units = 'nM'  -- Ensure units are consistent
    AND act.standard_relation = '=' --Only get equal relation
    AND act.standard_value IS NOT NULL
    AND a.confidence_score >= 8    -- High Confidence Data
```

```
    AND CAST(act.standard_value AS TEXT) ~ '^[0-9\.]+$'  -- Fix: Cast to text for
regex
LIMIT 100;
"""

# Option 1: Query the database directly
results = connect_and_query(sql_query)
if results:
    df = pd.DataFrame(results, columns=['molregno', 'smiles', 'ic50'])
else:
    df = None  # Handle the error appropriately

# Option 2: Load from CSV (alternative if you ran the SQL query in pgAdmin)
# data_file_path = os.path.join(base_path, "data", "chembl_data.csv") #Replace
"chembl_data.csv" with your file name
# df = load_data_from_csv(data_file_path)

if df is not None:
    print("Data loaded successfully:")
    print(df.head())

    # Data cleaning: Convert IC50 to numeric and drop missing values
    df['ic50'] = pd.to_numeric(df['ic50'], errors='coerce')  # Convert to numeric
    df = df.dropna(subset=['ic50', 'smiles']) # drop NA values if the data contain
those values

    # Feature Engineering with RDKit
    def calculate_descriptors(smiles):
        mol = Chem.MolFromSmiles(smiles)
        if mol is None:
            return None
        descriptors = {}
        descriptors['MolWt'] = Descriptors.MolWt(mol)
        descriptors['LogP'] = Chem.Crippen.MolLogP(mol)
        descriptors['HBD'] = Chem.Lipinski.NumHDonors(mol)
        descriptors['HBA'] = Chem.Lipinski.NumHAcceptors(mol)
        descriptors['TPSA'] = Chem.rdMolDescriptors.CalcTPSA(mol)
        return pd.Series(descriptors)

    # Apply the function to create new columns
    df = pd.concat([df, df['smiles'].apply(calculate_descriptors)], axis=1)
    df = df.dropna() # Drop any rows that have NA for some reason

    # Prepare data for modeling
    X = df[['MolWt', 'LogP', 'HBD', 'HBA', 'TPSA']]  # Features
    y = np.log10(df['ic50'])  # Target variable (log-transformed IC50)

    # Data scaling
    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)  # No squared=False for older scikit-
learn
    r2 = r2_score(y_test, y_pred)

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

else:
    print("Failed to load data.")
```

Explanation:

- **Imports:** Imports necessary libraries (pandas, psycopg2, RDKit, scikit-learn).
- **Database Connection:** `connect_and_query` function encapsulates the database connection logic, making it reusable. It takes an SQL query as input, executes it, and returns the results. Error handling is included.
- **CSV Loading:** `load_data_from_csv` function will load your data from the file.
- **SQL Query:** Defines the SQL query (the same one from the SQL section).
- **Data Loading:** Uses either the database connection or CSV loading to get the data into a Pandas DataFrame.
- **Data Cleaning:** Converts the `ic50` column to a numeric type and removes any rows with missing values in the `ic50` or `smiles` columns.
- **Feature Engineering:**
    - The `calculate_descriptors` function takes a SMILES string as input, converts it to an RDKit molecule object, and calculates several molecular descriptors (Molecular Weight, LogP, Hydrogen Bond Donors, Hydrogen Bond Acceptors, TPSA).
    - The function is applied to the `smiles` column of the DataFrame to create new columns for each descriptor.
- **Data Preparation:**
    - Defines the features (X) and the target variable (y). The target variable is log-transformed to potentially improve the distribution and model performance.
    - `StandardScaler` is used to scale the features, which is often important for linear models.
- **Model Training:**
    - Splits the data into training and testing sets.
    - Creates a `LinearRegression` model, trains it on the training data, and makes predictions on the test data.
- **Model Evaluation:**
    - Calculates the Mean Squared Error (MSE) and R-squared to evaluate the model's performance. Since you might be on an older scikit-learn version, `squared=False` is removed from `mean_squared_error`.

3. AIMLops Template Integration:

The code follows the AIMLops template by:

- Using `os.path.join(base_path, ...)` to construct file paths relative to the project root. This makes the code more portable.
- Keeping the data loading and processing logic separate from the model training and evaluation.

4. Example Usage (5 examples):

Here are 5 examples of how you might use and extend this code:

Example 1: Changing the Target

```
# Change the SQL query to target a different standard_type (e.g., 'Ki')
sql_query = """
SELECT
    md.molregno,
    cs.canonical_smiles,
    act.standard_value
FROM
    compound_structures cs
JOIN
    molecule_dictionary md ON cs.molregno = md.molregno
JOIN
    activities act ON md.molregno = act.molregno
JOIN
    assay_xref ax ON act.assay_id = ax.assay_id
JOIN
    assays a ON ax.assay_id = a.assay_id
WHERE
    act.standard_type = 'Ki'  -- Changed to Ki
    AND act.standard_units = 'nM'  -- Ensure units are consistent
    AND act.standard_relation = '=' --Only get equal relation
    AND act.standard_value IS NOT NULL
    AND a.confidence_score >= 8    -- High Confidence Data
    AND CAST(act.standard_value AS TEXT) ~ '^[0-9\.]+$'  -- Fix: Cast to text for
regex
LIMIT 100;
"""
```

Example 2: Adding More Descriptors

```
def calculate_descriptors(smiles):
    mol = Chem.MolFromSmiles(smiles)
    if mol is None:
        return None
    descriptors = {}
    descriptors['MolWt'] = Descriptors.MolWt(mol)
    descriptors['LogP'] = Chem.Crippen.MolLogP(mol)
    descriptors['HBD'] = Chem.Lipinski.NumHDonors(mol)
    descriptors['HBA'] = Chem.Lipinski.NumHAcceptors(mol)
    descriptors['TPSA'] = Chem.rdMolDescriptors.CalcTPSA(mol)
    descriptors['NumRings'] = Chem.rdMolDescriptors.CalcNumRings(mol)  # Add number of
rings
    return pd.Series(descriptors)

# ... (rest of the code remains the same)

X = df[['MolWt', 'LogP', 'HBD', 'HBA', 'TPSA', 'NumRings']]  # Add NumRings to
features
```

Example 3: Using a Different Model

```
from sklearn.ensemble import RandomForestRegressor

# ... (rest of the code, including data preparation, remains the same)

# Train a Random Forest Regression model
model = RandomForestRegressor(n_estimators=100, random_state=42) # Example parameters
```

```
model.fit(X_train, y_train)

# ... (rest of the code, including prediction and evaluation, remains the same)
```

Example 4: Target a specific Assay

```
-- Select 100 rows of data from ChEMBL, focusing on IC50 values for a specific target.
-- Adjust the WHERE clause to target a specific protein or biological process if
needed.
-- The limit clause is used to retrieve 100 rows for faster experimentation.

SELECT
    md.molregno,
    cs.canonical_smiles,
    act.standard_value
FROM
    compound_structures cs
JOIN
    molecule_dictionary md ON cs.molregno = md.molregno
JOIN
    activities act ON md.molregno = act.molregno
JOIN
    assay_xref ax ON act.assay_id = ax.assay_id
JOIN
    assays a ON ax.assay_id = a.assay_id
WHERE
    act.standard_type = 'IC50'  -- Focus on IC50 values
    AND act.standard_units = 'nM'  -- Ensure units are consistent
    AND act.standard_relation = '=' --Only get equal relation
    AND act.standard_value IS NOT NULL
    AND a.confidence_score >= 8    -- High Confidence Data
    AND a.assay_id = 1234          -- replace 1234 with the assay_id of your target
    AND CAST(act.standard_value AS TEXT) ~ '^[0-9\.]+$'  -- Fix: Cast to text for
regex
LIMIT 100;
```

Example 5: Saving the Model

```
import joblib  # Or pickle

# ... (rest of the code, including model training, remains the same)

# Save the model
model_path = os.path.join(base_path, "models", "linear_regression_model.joblib")
joblib.dump(model, model_path)
print(f"Model saved to {model_path}")

# Later, you can load the model:
# loaded_model = joblib.load(model_path)
```

Summary in English:

This comprehensive guide provides you with a clear analysis model, SQL and Python code to address your project's requirements, solutions to the errors you encountered, and five practical examples to help you get started. Remember to adapt the SQL query and feature engineering steps to match the specific targets and compounds you are interested in. Good luck!

Resumen en español:

Esta guía completa le proporciona un modelo de análisis claro, código SQL y Python para abordar los requisitos de su proyecto, soluciones a los errores que encontró y cinco ejemplos prácticos para

ayudarlo a comenzar. Recuerde adaptar la consulta SQL y los pasos de ingeniería de características para que coincidan con los objetivos y compuestos específicos que le interesan. ¡Buena suerte!