PharmApp Suite

🧠 AI for Drug Discovery and Development 🧪

🧩 Topic: Neuropsychiatric drug insights from ChEMBL

Okay, I understand. You're working with Chembl 35 data using RDKit and PostgreSQL, following an AIMLops template, and encountering some errors. Let's break down the analysis model, address the errors, and provide code examples.

1. Analysis Model Overview

The goal is to analyze Chembl 35 data, likely focusing on specific chemical properties or biological activities. Given the use of RDKit, you're probably interested in featurizing molecules (converting them into numerical representations) and then applying machine learning or statistical methods to understand relationships between molecular structure and activity. Here's a general analysis pipeline we can build on:

- **Data Extraction (SQL):** Select relevant data from the Chembl 35 database based on the Topic. This includes molecule structures (SMILES strings or other formats), activity data (e.g., IC50, Ki values), target information (protein targets), and any relevant metadata. The SQL query should be optimized to retrieve only the necessary columns and filter to a manageable number of rows (100 in your case).
- **Data Preprocessing (Python):**
    - **Loading Data:** Read the extracted data from the CSV file into a Pandas DataFrame.
    - **Data Cleaning:** Handle missing values, remove duplicates, and filter based on data quality criteria (e.g., activity type, units). The error `ERROR: operator does not exist: numeric ~ unknown, LINE 12: AND act.standard_value ~ '^[0-9\.]+$'` suggests an issue with how you're filtering numeric values in your SQL query. It's likely trying to use a regular expression operator (~) on a numeric column.
    - **Unit Conversion:** Standardize activity values to a common unit (e.g., uM).
    - **Molecule Featurization:** Use RDKit to generate molecular descriptors or fingerprints (e.g., Morgan fingerprints, physicochemical properties) from the SMILES strings. This converts molecules into numerical vectors that can be used in machine learning models.
- **Exploratory Data Analysis (EDA):**
    - **Descriptive Statistics:** Calculate summary statistics for activity values and molecular descriptors.
    - **Visualization:** Create scatter plots, histograms, and other visualizations to explore relationships between features and activity.
    - **Correlation Analysis:** Identify highly correlated features.
- **Model Building (Python):**
    - **Feature Selection:** Select relevant features based on EDA or feature importance scores from machine learning models.
    - **Model Selection:** Choose an appropriate machine learning model based on the type of data and the research question (e.g., linear regression, random forest, support vector machine).
    - **Model Training:** Train the model on the prepared data.
    - **Model Evaluation:** Evaluate the model's performance using appropriate metrics (e.g., R-squared, RMSE, AUC). The error `old scikit-learn version does not support`

parameters `squared=False` in the `mean_squared_error` function means you need to either upgrade your scikit-learn version or remove the `squared=False` parameter. Upgrading is generally the best option.

- **Interpretation and Reporting:**
    - **Feature Importance Analysis:** Determine which features are most important for predicting activity.
    - **Model Interpretation:** Interpret the model's predictions and draw conclusions about the relationships between molecular structure and activity.
    - **Reporting:** Document the entire analysis process, including data cleaning, preprocessing, model building, and evaluation.

2. SQL Code (for data extraction - example for Topic_CheMBL_35_54)

```sql
-- Extract data for a specific target (replace with your target of interest)
-- and limit the results to 100 rows.
SELECT
    md.chembl_id,
    cs.canonical_smiles,
    act.standard_type,
    act.standard_value,
    act.standard_units
FROM
    molecule_dictionary md
JOIN
    compound_structures cs ON md.molregno = cs.molregno
JOIN
    activities act ON md.molregno = act.molregno
JOIN
    target_dictionary td ON act.tid = td.tid
WHERE
    td.chembl_id = 'CHEMBL205'  -- Replace with your target of interest related to
Topic_CheMBL_35_54
    AND act.standard_type = 'IC50' -- Or other activity type of interest
    AND act.standard_relation = '='
    AND act.standard_value IS NOT NULL  -- Ensure numeric columns are checked for null
*before* applying numeric constraints
    AND act.standard_value BETWEEN 0 AND 100000 -- Filter out extreme values
    AND act.standard_units = 'nM'  -- Ensure consistent units
LIMIT 100;

-- Save this to ../data/chembl_35_54_data.csv (using pgAdmin's export feature)
```

Explanation:

- **Target Selection:** I've used `td.chembl_id = 'CHEMBL205'` as an example. You need to replace this with the specific Chembl ID related to your `Topic_CheMBL_35_54`. You'll need to investigate the Chembl database to find a relevant target ID.
- **Activity Type:** I've used `act.standard_type = 'IC50'`. Adjust this to the specific activity type you're interested in.
- **Value Filtering:** `act.standard_value BETWEEN 0 AND 100000` filters the activity values to a reasonable range. Adjust the upper and lower bounds as needed. This is crucial because attempting to use regex (~) on non-text fields will cause an error. The `IS NOT NULL` check is *essential* before any numerical comparison.
- **Unit Consistency:** `act.standard_units = 'nM'` ensures you're working with consistent units.
- **Limiting Results:** `LIMIT 100` restricts the number of rows returned.
- **Saving to CSV:** Save the output of this query to `../data/chembl_35_54_data.csv` using pgAdmin's export functionality.

Addressing the SQL Error:

The error `ERROR: operator does not exist: numeric ~ unknown, LINE 12: AND act.standard_value ~ '^[0-9\.]+$'` occurs because you're trying to use the regular expression operator (~) on a numeric column (`act.standard_value`). Instead of using a regular expression, use numeric comparisons like:

- `act.standard_value IS NOT NULL` (Check for null values FIRST!)
- `act.standard_value > 0`
- `act.standard_value < 1000`
- `act.standard_value BETWEEN 10 AND 100`

The corrected SQL query above implements this.

3. Python Code (Jupyter Notebook - `Topic_CheMBL_35_54_1_Data_Loading_and_Preprocessing.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  # Feature Scaling

# Define base path
base_path = os.path.abspath(os.path.join(os.getcwd(), ".."))  # Assumes notebook is in
a subfolder
data_path = os.path.join(base_path, "data", "chembl_35_54_data.csv")

print(f"Base path: {base_path}")
print(f"Data path: {data_path}")

# Load data
try:
    df = pd.read_csv(data_path)
    print("Data loaded successfully.")
except FileNotFoundError:
    print(f"Error: File not found at {data_path}. Make sure the SQL query was run and
the file was saved correctly.")
    exit()

# Data Cleaning and Preprocessing
print("\nData Cleaning and Preprocessing...")
# Handle missing values (replace with median for numeric columns)
for col in df.select_dtypes(include=np.number).columns:
    df[col] = df[col].fillna(df[col].median())

# Remove duplicates based on molecule ID
df = df.drop_duplicates(subset=['chembl_id'])

# Standardize activity values (convert to pIC50)
def convert_to_pic50(ic50_nM):
    """Converts IC50 in nM to pIC50."""
    if ic50_nM is None or pd.isna(ic50_nM):
        return None
    try:
        pIC50 = -np.log10(ic50_nM * 1e-9)  # Convert nM to M and then to pIC50
        return pIC50
    except:
        return None

df['pIC50'] = df['standard_value'].apply(convert_to_pic50)
df = df.dropna(subset=['pIC50', 'canonical_smiles']) # Drop rows with missing pIC50 or
SMILES

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


# Molecule Featurization
print("\nMolecule Featurization...")

def generate_descriptors(smiles):
    """Generates RDKit descriptors for a given SMILES string."""
    mol = Chem.MolFromSmiles(smiles)
    if mol is None:
        return None
    descriptors = {}
    descriptors['MW'] = Descriptors.MolWt(mol)
    descriptors['LogP'] = Descriptors.MolLogP(mol)
    descriptors['HBD'] = Descriptors.NumHDonors(mol)
    descriptors['HBA'] = Descriptors.NumHAcceptors(mol)
    descriptors['TPSA'] = Descriptors.TPSA(mol)

    # Calculate Morgan Fingerprint
    info = {}
    fp = AllChem.GetMorganFingerprint(mol, radius=2, bitInfo=info)
    descriptors['MorganFP'] = fp  # Store the fingerprint object

    return descriptors


df['descriptors'] = df['canonical_smiles'].apply(generate_descriptors)
df = df.dropna(subset=['descriptors']) # Drop rows where descriptor generation failed
df = df[df['descriptors'].apply(lambda x: isinstance(x, dict))] # ensure descriptors
are dictionaries

# Convert descriptors to columns, handling fingerprints
def unpack_descriptors(row):
    descriptors = row['descriptors']
    if isinstance(descriptors, dict):
      return pd.Series(descriptors)
    else:
      return pd.Series([None] * 6, index=['MW', 'LogP', 'HBD', 'HBA', 'TPSA',
'MorganFP']) # Return series of Nones
df = pd.concat([df, df.apply(unpack_descriptors, axis=1)], axis=1)

# Drop rows with missing descriptor values after expansion
df = df.dropna(subset=['MW', 'LogP', 'HBD', 'HBA', 'TPSA', 'MorganFP'])

# Function to convert Morgan Fingerprints to sparse arrays
def fingerprint_to_array(fp):
    if fp is None:
        return None
    info = {}
    fp_vect = np.zeros((2048,), dtype=int) # set vector size
    for bit, count in fp.GetNonzeroElements().items():
        fp_vect[bit] = count
    return fp_vect

df['MorganFP'] = df['MorganFP'].apply(fingerprint_to_array)

# Filter out rows where Morgan Fingerprint conversion resulted in None
df = df.dropna(subset=['MorganFP'])

# Convert Morgan Fingerprints to columns
morgan_fingerprint_df = pd.DataFrame(df['MorganFP'].tolist())
morgan_fingerprint_df = morgan_fingerprint_df.add_prefix('MorganFP_')
df = pd.concat([df, morgan_fingerprint_df], axis=1)
df = df.drop(columns=['descriptors', 'MorganFP'])

# Feature Scaling
numerical_cols = df.select_dtypes(include=np.number).columns.tolist()
numerical_cols.remove('pIC50') # Don't scale the target variable
scaler = StandardScaler()
df[numerical_cols] = scaler.fit_transform(df[numerical_cols])


print("\nData Preprocessing Complete.")
print(df.head())

# Save the preprocessed data
preprocessed_data_path = os.path.join(base_path, "data",
"chembl_35_54_preprocessed_data.csv")
df.to_csv(preprocessed_data_path, index=False)
print(f"Preprocessed data saved to: {preprocessed_data_path}")
```

Explanation:

- **Base Path:** The `base_path` is correctly set to move one level up from the notebook directory.
- **Data Loading:** Loads the CSV data using Pandas. The `try...except` block handles the `FileNotFoundError`.
- **Data Cleaning:**
  - Handles missing values by filling with the median.
  - Removes duplicate molecules based on `chembl_id`.
  - Converts activity values to pIC50. This is a crucial step for standardizing the data and making it suitable for modeling.
  - Drops rows with missing pIC50 or SMILES values.
- **Molecule Featurization:**
  - The `generate_descriptors` function calculates a set of basic RDKit descriptors (MW, LogP, HBD, HBA, TPSA). **Important:** I've added Morgan Fingerprint calculation as well, since they are commonly used.
  - The function stores the Morgan fingerprint object directly.
  - The `.apply` function applies the `generate_descriptors` function to each SMILES string in the DataFrame.
  - The `unpack_descriptors` function transforms the descriptors (dictionary) into separate columns in the DataFrame. This makes the descriptors usable in machine learning models.
  - Rows where descriptor generation fails are dropped.

    - The code converts Morgan Fingerprints to sparse arrays and create individual columns for each bit using the `fingerprint_to_array` function.
- **Feature Scaling:** StandardScaler is used to scale numerical features.
- **Saving Preprocessed Data:** The preprocessed data is saved to a new CSV file.

4. Python Code (Jupyter Notebook - `Topic_CheMBL_35_54_2_Model_Building_and_Evaluation.ipynb`)

```python
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
import numpy as np

# Define base path
base_path = os.path.abspath(os.path.join(os.getcwd(), ".."))  # Assumes notebook is in
a subfolder
preprocessed_data_path = os.path.join(base_path, "data",
"chembl_35_54_preprocessed_data.csv")

# Load preprocessed data
try:
    df = pd.read_csv(preprocessed_data_path)
    print("Preprocessed data loaded successfully.")
except FileNotFoundError:
    print(f"Error: File not found at {preprocessed_data_path}. Make sure the
preprocessing notebook was run.")
    exit()


# Prepare data for modeling
X = df.drop(['chembl_id', 'canonical_smiles', 'standard_type', 'standard_value',
'standard_units', 'pIC50'], axis=1, errors='ignore') # Drop non-feature columns
y = df['pIC50']

# Handle any non-numeric values that might have slipped in
X = X.apply(pd.to_numeric, errors='coerce')  # Convert all columns to numeric,
coercing errors to NaN
X = X.fillna(X.mean())  # Fill remaining NaNs with the column mean

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

# Model Training
model = LinearRegression()
model.fit(X_train, y_train)

# Model Evaluation
y_pred = model.predict(X_test)
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}")

# Feature Importance (for Linear Regression)
if hasattr(model, 'coef_'):
    feature_importance = pd.DataFrame({'Feature': X.columns, 'Importance': 
model.coef_})
    feature_importance = feature_importance.sort_values('Importance', ascending=False)
    print("\nFeature Importance:")
    print(feature_importance)
else:
    print("\nFeature importance not available for this model.")


# Save the model (optional)
# import joblib
# model_path = os.path.join(base_path, "models", 
"chembl_35_54_linear_regression_model.joblib")
# joblib.dump(model, model_path)
# print(f"Model saved to: {model_path}")
```

Explanation:

- **Data Loading:** Loads the preprocessed data.
- **Data Preparation:**
    - Splits the data into features (X) and target (y). It's crucial to drop non-feature columns like IDs, SMILES strings, and the original activity values. The `errors='ignore'` prevents errors if some of these columns don't exist.
    - **Handling Non-Numeric Values:** This is a *critical* addition. Even after preprocessing, there's a chance that some non-numeric values might have slipped in. The `X = X.apply(pd.to_numeric, errors='coerce')` line attempts to convert all columns in the feature matrix `X` to numeric values. If a value cannot be converted (e.g., it's a string), it's replaced with `NaN`. Then, `X = X.fillna(X.mean())` fills any remaining `NaN` values with the mean of the column. This ensures that the model receives numeric input.
- **Train/Test Split:** Splits the data into training and testing sets.
- **Model Training:** Trains a Linear Regression model. You can easily substitute this with other models.
- **Model Evaluation:** Evaluates the model using Mean Squared Error (MSE) and R-squared.
- **Feature Importance:** Prints the feature importance scores (only applicable to linear models).
- **Model Saving (Optional):** Shows how to save the trained model using `joblib`. Uncomment the lines to save the model. You'll also need to `import joblib`.

Addressing the scikit-learn Error:

The error `old scikit-learn version does not support parameters squared=False in the mean_squared_error function` means one of two things:

1. **Upgrade scikit-learn:** This is the *recommended* solution. In your Jupyter Notebook, run `!pip install --upgrade scikit-learn`. Then, restart the kernel.
2. **Remove `squared=False`:** If you cannot upgrade scikit-learn, simply remove the `squared=False` argument from the `mean_squared_error` function. However, this will return the Mean Squared Error (MSE) instead of the Root Mean Squared Error (RMSE).

5. Examples

Here are five examples focusing on different aspects of the analysis:

1. **Target Selection and Activity Type Exploration:** Focus on exploring the different target IDs and activity types available in your dataset. Modify the SQL query to select different target IDs and activity types (e.g., Ki, EC50) and then analyze the distribution of activity values for each combination.

2. **Descriptor Selection and Correlation Analysis:** Explore different RDKit descriptors beyond the basic ones I included. Calculate a wider range of descriptors and then perform a correlation analysis to identify highly correlated descriptors. Remove redundant descriptors before model building.

3. **Model Comparison:** Experiment with different machine learning models (e.g., Random Forest, Support Vector Machine) and compare their performance on the same dataset. Use cross-validation to obtain robust estimates of model performance.

4. **Feature Selection Techniques:** Implement different feature selection techniques, such as Recursive Feature Elimination (RFE) or SelectKBest, to identify the most relevant features for predicting activity. Compare the performance of models trained with and without feature selection.

5. **Handling Different Activity Units:** Extend the code to handle different activity units (e.g., uM, nM, pM). Implement a unit conversion function to standardize all activity values to a common unit before modeling.

Important Considerations:

- **Chembl Version:** Ensure you are using the correct Chembl version (35) consistently throughout your analysis.
- **Database Connection:** Double-check your database connection parameters (IP address, username, password, database name) and ensure that you have the necessary permissions to access the database.
- **Error Handling:** Implement robust error handling throughout your code to catch potential issues and provide informative error messages.
- **Documentation:** Document your code thoroughly, including comments explaining each step of the analysis.

By following this guide and adapting the code examples to your specific research question, you should be able to successfully analyze Chembl 35 data using RDKit and PostgreSQL. Remember to address the errors I highlighted and adjust the code based on your specific needs. Good luck!