PharmApp Suite

🧠 AI for Drug Discovery and Development 💉

🧩 Topic: Analyzing drug-likeness using RDKit

Okay, I understand. You're working on a Chembl 35 data analysis project within an AIMLops framework, focusing on a specific topic (Topic_CheMBL_35_5) using PostgreSQL, RDKit, and Jupyter Notebook. You're encountering errors related to SQL operator compatibility and scikit-learn versioning. Let's break down the analysis, provide corrected code snippets, and address the errors.

1. Analysis of the Analysis Model

The project aims to leverage Chembl 35 data for drug research and development. The specific focus (Topic_CheMBL_35_5) isn't defined, so I'll assume it relates to identifying potent inhibitors for a specific target. The model will likely involve:

- **Data Extraction (SQL):** Querying the Chembl 35 database to retrieve relevant bioactivity data (e.g., IC50, Ki, EC50 values) for compounds against a specific target of interest. This requires filtering based on assay type, target ID, and data quality (e.g., confidence scores). Crucially, the SQL must be robust to handle variations in data formatting. The extracted data will be saved as CSV files.
- **Data Preprocessing (Python/RDKit):**
    - Loading the data from the CSV files.
    - Filtering and cleaning the data (e.g., removing entries with missing values or incorrect units).
    - Generating molecular descriptors using RDKit. These descriptors represent the physicochemical properties of the molecules and are essential for building predictive models. Common descriptors include:
        - Molecular Weight
        - LogP (octanol-water partition coefficient)
        - Hydrogen Bond Donors/Acceptors
        - Topological Polar Surface Area (TPSA)
        - Number of Rotatable Bonds
- **Model Building (Python/Scikit-learn):** Using the molecular descriptors as features and the bioactivity values (e.g., pIC50) as the target variable, build a machine learning model to predict the activity of new compounds. Possible model choices include:
    - Linear Regression
    - Support Vector Regression (SVR)
    - Random Forest
    - Gradient Boosting
- **Model Evaluation:** Splitting the data into training and test sets. Training the model on the training set and evaluating its performance on the test set using metrics such as:
    - Mean Squared Error (MSE)
    - R-squared (coefficient of determination)
    - Root Mean Squared Error (RMSE)

Addressing the Errors

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

  This error occurs in PostgreSQL when you're trying to use the regular expression operator ~ (similar to LIKE) on a numeric column. PostgreSQL doesn't implicitly convert numeric types to text for regular expression matching. The solution is to explicitly cast the `standard_value` column to text.

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

  This indicates that you are using an older version of scikit-learn. The `squared=False` argument was introduced in a later version of scikit-learn's `mean_squared_error` function to directly return the Root Mean Squared Error (RMSE). You have two options:

  1. **Upgrade scikit-learn:** The easiest solution is to upgrade your scikit-learn version to the latest stable release. In your Jupyter Notebook, run:

     ```
     !pip install -U scikit-learn
     ```

  2. **Calculate RMSE Manually:** If upgrading isn't feasible, you can calculate the RMSE by taking the square root of the MSE:

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

     mse = mean_squared_error(y_true, y_predicted)
     rmse = np.sqrt(mse)
     print(f"RMSE: {rmse}")
     ```

2. SQL and Python Code (with corrections and examples)

Here's the SQL and Python code, incorporating the error corrections and aiming for a 100-row sample. This example assumes that `Topic_CheMBL_35_5` relates to finding inhibitors for a specific target. I'll use target ID `CHEMBL205` (a common target) as an example. **Adjust the WHERE clauses to match your specific research question and target.**

English:

SQL (query.sql):

```
-- Adjust the WHERE clauses to match your target and criteria
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
    component_sequences cs2 ON ac.component_id = cs2.component_id
WHERE
    cs2.accession = 'P00533'  -- Example: EGFR target (UniProt accession) -- Adjust
this to YOUR target
```

```
    AND act.standard_type = 'IC50'  -- Example: IC50 values
    AND act.standard_units = 'nM'
    AND act.standard_value IS NOT NULL
    AND act.standard_value::TEXT ~ '^[0-9\.]+$'  -- Corrected: Casting to TEXT for
regex
    AND act.confidence_score >= 7 -- High Confidence data
LIMIT 100;
```

Explanation:

1. **Target Filtering:** The `WHERE cs2.accession = 'P00533'` line filters for activities related to a specific target (EGFR in this example, identified by its UniProt accession number). **Change this to the UniProt accession number of YOUR target.**
2. **Activity Filtering:** `act.standard_type = 'IC50'` filters for IC50 values. You might need to adjust this to other activity types (e.g., Ki, EC50) based on your research question.
3. **Unit Filtering:** `act.standard_units = 'nM'` ensures that the values are in nanomolars. This is important for consistency.
4. **NULL Handling:** `act.standard_value IS NOT NULL` removes entries with missing activity values.
5. **Regular Expression for Numeric Values:** `act.standard_value::TEXT ~ '^[0-9\.]+$'` filters to keep rows where `standard_value` is a number. The crucial change is the `::TEXT` which *casts* the numeric value to text before applying the regex.
6. **Confidence Score:** `act.confidence_score >= 7` ensures you're using relatively high-quality data. Adjust as needed.
7. **LIMIT:** `LIMIT 100` restricts the result set to 100 rows.

How to Run SQL:

1. Open pgAdmin and connect to your `chembl_35` database (using the provided credentials).
2. Open a query window.
3. Paste the SQL code into the query window.
4. Execute the query.
5. Export the results to a CSV file named `chembl_data.csv` (or any name you choose) and save it to the `../data/` directory in your project. Remember to set the appropriate separator (usually a comma).

Python (Topic_CheMBL_35_5_1_data_prep.ipynb):

```
import os
import pandas as pd
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
import matplotlib.pyplot as plt

# Define the base path
base_path = ".."  # Assuming notebook is in the 'notebooks' directory

# Construct the file path to the CSV data
data_file = os.path.join(base_path, "data", "chembl_data.csv")  # Change filename if
different

# Load the data
try:
    df = pd.read_csv(data_file)
```

```
    print("Data loaded successfully.")
except FileNotFoundError:
    print(f"Error: The file '{data_file}' was not found.")
    exit()

# Data Cleaning and Preprocessing
df = df.dropna(subset=['canonical_smiles', 'standard_value'])  # Remove rows with
missing SMILES or IC50
df = df[df['standard_units'] == 'nM']  # Ensure units are nM
df['standard_value'] = pd.to_numeric(df['standard_value'], errors='coerce') #Convert
standard value to number
df = df.dropna(subset=['standard_value']) #Remove NaN values after conversion

# Calculate pIC50
df['pIC50'] = -np.log10(df['standard_value'] / 1e9)  # Convert IC50 to pIC50

# RDKit Molecular Descriptor Calculation
def calculate_descriptors(smiles):
    mol = Chem.MolFromSmiles(smiles)
    if mol is None:
        return None
    descriptors = {}
    descriptors['MolWt'] = Descriptors.MolWt(mol)
    descriptors['LogP'] = Descriptors.MolLogP(mol)
    descriptors['HBD'] = Descriptors.NumHDonors(mol)
    descriptors['HBA'] = Descriptors.NumHAcceptors(mol)
    descriptors['TPSA'] = Descriptors.TPSA(mol)
    descriptors['RotatableBonds'] = Descriptors.NumRotatableBonds(mol)
    return pd.Series(descriptors)

# Apply Descriptor Calculation
df = pd.concat([df, df['canonical_smiles'].apply(calculate_descriptors)], axis=1)
df = df.dropna()  # Remove any rows where descriptor calculation failed

# Data Splitting
X = df[['MolWt', 'LogP', 'HBD', 'HBA', 'TPSA', 'RotatableBonds']]
y = df['pIC50']
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)
rmse = np.sqrt(mse)  # Calculate RMSE manually
r2 = r2_score(y_test, y_pred)

print(f"Mean Squared Error (MSE): {mse}")
print(f"Root Mean Squared Error (RMSE): {rmse}")
print(f"R-squared (R2): {r2}")

# Visualization (Example)
plt.scatter(y_test, y_pred)
plt.xlabel("Actual pIC50")
plt.ylabel("Predicted pIC50")
plt.title("Actual vs. Predicted pIC50 Values")
plt.show()
```

Explanation:

1. **Path Handling:** The code uses `os.path.join` to construct the file path to the CSV data, making it more portable and robust.
2. **Data Loading:** Loads the CSV data using `pd.read_csv`.
3. **Data Cleaning:**
    - Removes rows with missing SMILES strings or IC50 values.
    - Filters for entries where the standard units are 'nM'.
    - Converts `standard_value` to numeric, handling potential errors.
    - Drops any rows that resulted in NaN after converting to a number.
4. **pIC50 Calculation:** Converts IC50 values to pIC50 (a more common and convenient scale).
5. **RDKit Descriptor Calculation:** The `calculate_descriptors` function takes a SMILES string as input, creates an RDKit molecule object, and calculates a set of common molecular descriptors. The `apply` method is used to apply this function to each SMILES string in the DataFrame. Includes error handling for invalid SMILES.
6. **Data Splitting:** Splits the data into training and test sets.
7. **Model Training:** Trains a Linear Regression model. You can easily experiment with other models.
8. **Model Evaluation:** Calculates the MSE, RMSE (manually), and R-squared to assess the model's performance.
9. **Visualization:** Creates a scatter plot of actual vs. predicted pIC50 values.

Important Considerations:

- **Target Selection:** The SQL query is currently filtering for EGFR (UniProt accession P00533). **Change this to the appropriate UniProt accession number for your target.**
- **Activity Type:** The SQL query is filtering for IC50 values. Adjust this if you're interested in other activity types (e.g., Ki, EC50).
- **Model Choice:** Linear Regression is a simple model. Consider more complex models like Random Forest or Gradient Boosting for better performance. You may need to tune the hyperparameters of these models.
- **Feature Selection:** You can experiment with different sets of molecular descriptors to see which ones are most predictive for your target.
- **Data Quality:** Always be mindful of data quality. Chembl data can have inconsistencies and errors. Careful data cleaning is crucial.
- **CheMBL Version:** Make sure you are aligned with the CheMBL 35 schema. Check the official documentation.

3. Five Examples (Topic_CheMBL_35_5_2_examples.ipynb):

Here are five examples of what you can do with this data, focusing on different aspects:

```
import os
import pandas as pd
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
import matplotlib.pyplot as plt
import seaborn as sns

# Define the base path
base_path = ".."  # Assuming notebook is in the 'notebooks' directory
```

```
# Construct the file path to the CSV data
data_file = os.path.join(base_path, "data", "chembl_data.csv")  # Change filename if
different

# Load the data
try:
    df = pd.read_csv(data_file)
    print("Data loaded successfully.")
except FileNotFoundError:
    print(f"Error: The file '{data_file}' was not found.")
    exit()

# Data Cleaning and Preprocessing
df = df.dropna(subset=['canonical_smiles', 'standard_value'])  # Remove rows with
missing SMILES or IC50
df = df[df['standard_units'] == 'nM']  # Ensure units are nM
df['standard_value'] = pd.to_numeric(df['standard_value'], errors='coerce') #Convert
standard value to number
df = df.dropna(subset=['standard_value']) #Remove NaN values after conversion

# Calculate pIC50
df['pIC50'] = -np.log10(df['standard_value'] / 1e9)  # Convert IC50 to pIC50

# RDKit Molecular Descriptor Calculation
def calculate_descriptors(smiles):
    mol = Chem.MolFromSmiles(smiles)
    if mol is None:
        return None
    descriptors = {}
    descriptors['MolWt'] = Descriptors.MolWt(mol)
    descriptors['LogP'] = Descriptors.MolLogP(mol)
    descriptors['HBD'] = Descriptors.NumHDonors(mol)
    descriptors['HBA'] = Descriptors.NumHAcceptors(mol)
    descriptors['TPSA'] = Descriptors.TPSA(mol)
    descriptors['RotatableBonds'] = Descriptors.NumRotatableBonds(mol)
    return pd.Series(descriptors)

# Apply Descriptor Calculation
df = pd.concat([df, df['canonical_smiles'].apply(calculate_descriptors)], axis=1)
df = df.dropna()  # Remove any rows where descriptor calculation failed

# EXAMPLE 1: Distribution of pIC50 Values
plt.figure(figsize=(8, 6))
sns.histplot(df['pIC50'], kde=True)
plt.title("Distribution of pIC50 Values")
plt.xlabel("pIC50")
plt.ylabel("Frequency")
plt.show()


# EXAMPLE 2: Scatter Plot of LogP vs. pIC50
plt.figure(figsize=(8, 6))
plt.scatter(df['LogP'], df['pIC50'])
plt.title("LogP vs. pIC50")
plt.xlabel("LogP")
plt.ylabel("pIC50")
plt.show()

# EXAMPLE 3: Boxplot of pIC50 for Different Standard Types (If you have multiple
activity types)
```

```
# (This example assumes you have multiple standard types - adapt the SQL query if
needed)
if 'standard_type' in df.columns and len(df['standard_type'].unique()) > 1:
    plt.figure(figsize=(8, 6))
    sns.boxplot(x='standard_type', y='pIC50', data=df)
    plt.title("pIC50 by Standard Type")
    plt.xlabel("Standard Type")
    plt.ylabel("pIC50")
    plt.show()
else:
    print("Skipping Example 3: Only one standard type found.")

# EXAMPLE 4:  Correlation Matrix of Descriptors
descriptors = ['MolWt', 'LogP', 'HBD', 'HBA', 'TPSA', 'RotatableBonds', 'pIC50']
if all(col in df.columns for col in descriptors):
    correlation_matrix = df[descriptors].corr()
    plt.figure(figsize=(10, 8))
    sns.heatmap(correlation_matrix, annot=True, cmap="coolwarm")
    plt.title("Correlation Matrix of Molecular Descriptors and pIC50")
    plt.show()
else:
    print("Skipping Example 4: Not all descriptors are present.")

# EXAMPLE 5: Train a Random Forest Regressor and Compare to Linear Regression
from sklearn.ensemble import RandomForestRegressor

# Data Splitting
X = df[['MolWt', 'LogP', 'HBD', 'HBA', 'TPSA', 'RotatableBonds']]
y = df['pIC50']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Train Linear Regression
linear_model = LinearRegression()
linear_model.fit(X_train, y_train)
linear_predictions = linear_model.predict(X_test)
linear_rmse = np.sqrt(mean_squared_error(y_test, linear_predictions))

# Train Random Forest Regressor
rf_model = RandomForestRegressor(n_estimators=100, random_state=42) # Adjust
n_estimators as needed
rf_model.fit(X_train, y_train)
rf_predictions = rf_model.predict(X_test)
rf_rmse = np.sqrt(mean_squared_error(y_test, rf_predictions))

print(f"Linear Regression RMSE: {linear_rmse}")
print(f"Random Forest RMSE: {rf_rmse}")
```

Explanation of Examples:

- **Example 1:** Visualizes the distribution of pIC50 values to understand the range of activity in your dataset.
- **Example 2:** Examines the relationship between LogP (a measure of lipophilicity) and pIC50. This can help identify trends in activity related to lipophilicity.
- **Example 3:** Compares pIC50 values for different assay types (e.g., IC50 vs. Ki) if your dataset includes multiple assay types. This helps assess if different assay types give consistent results.
- **Example 4:** Shows the correlation matrix between the various molecular descriptors and pIC50. This will highlight which descriptors are most strongly correlated with the target variable.

- **Example 5:** Trains a Random Forest model, a more advanced machine learning model than linear regression, and compares its performance (RMSE) to linear regression. This helps determine if a more complex model improves predictive accuracy.

Important Notes about Examples:

- These examples are starting points. Customize them to explore your specific research questions.
- Adapt the SQL queries to retrieve the data needed for each example. For Example 3, adapt the SQL to get other types.
- Consider more advanced visualizations (e.g., 3D scatter plots, heatmaps of descriptor values).
- Experiment with different machine learning models and hyperparameter tuning to improve model performance.

In summary, I have provided a comprehensive guide to address your Chembl 35 data analysis project. This includes the analysis of the model, corrected SQL and Python code, and five illustrative examples. Remember to adapt the code and examples to match your specific target, research question, and the specific content of Topic_CheMBL_35_5.