PharmApp Suite

🧠 AI for Drug Discovery and Development 🍼

🧩 Topic: Top viral/bacterial targets in ChEMBL

Okay, I understand. You're working on a Chembl 35 data analysis project using RDKit and facing a few challenges. You want to follow an AIMLops template, use Jupyter notebooks, and address the specific errors you're encountering. You also need guidance on building SQL queries and Python code for data filtering and analysis, aiming for a sample size of 100 rows. Let's break this down:

1. Analysis and Model Explanation

The project involves analyzing chemical data from ChEMBL 35 to support drug research and development. A typical analysis model would involve the following steps:

- **Data Extraction:** Extract relevant data from the ChEMBL database using SQL queries. This will involve selecting specific tables (e.g., activities, molecules, targets) and filtering the data based on criteria such as activity types, standard values, and target organism.
- **Data Cleaning and Preprocessing:** Clean the data to handle missing values, remove duplicates, and standardize units. This may also involve converting data types and formatting the data for analysis.
- **Feature Engineering:** Generate relevant features from the chemical structures using RDKit. These features can include molecular weight, logP, topological polar surface area (TPSA), number of hydrogen bond donors/acceptors, and various other descriptors.
- **Exploratory Data Analysis (EDA):** Explore the data to identify patterns and relationships. This may involve generating histograms, scatter plots, and correlation matrices to visualize the data and identify potential predictors of activity.
- **Model Building:** Build predictive models to predict the activity of compounds based on their chemical features. Common machine learning algorithms used for this purpose include linear regression, random forests, support vector machines, and neural networks.
- **Model Evaluation:** Evaluate the performance of the models using appropriate metrics such as R-squared, mean squared error, and area under the ROC curve (AUC).
- **Interpretation and Validation:** Interpret the results of the models and validate them using external data or experimental results.

Addressing specific error

- **ERROR: operator does not exist: numeric ~ unknown, LINE 12: AND act.standard_value ~ '[1]+$'**
    - This error occurs because the ~ operator is being used for regular expression matching on a numeric column (`act.standard_value`). PostgreSQL doesn't directly support this. You need to cast the numeric column to text or use a different approach.
- **old scikit-learn version does not support parameters squared=False in the mean_squared_error function**
    - This means you're using an older version of scikit-learn. You have two options:
        1. **Upgrade scikit-learn:** The easiest solution is to upgrade to a newer version. Run `pip install --upgrade scikit-learn`.
        2. **Remove `squared=False`:** If upgrading isn't possible, calculate the RMSE manually by taking the square root of the MSE.

[1] 0-9.

2. Code Examples (SQL and Python)

English:

SQL (Extracting data and addressing the first error)

```sql
-- SQL query to extract data from ChEMBL, addressing the numeric ~ unknown error.
-- Limiting to 100 rows for demonstration purposes.

SELECT
    mol.molregno,
    mol.chembl_id,
    act.standard_type,
    act.standard_value,
    act.standard_units,
    act.assay_chembl_id,
    cs.canonical_smiles
FROM
    activities act
JOIN
    molecule_dictionary mol ON act.molregno = mol.molregno
JOIN
    compound_structures cs ON mol.molregno = cs.molregno
WHERE
    act.standard_type IN ('IC50', 'Ki', 'EC50')  -- Common activity types
    AND act.standard_units = 'nM'  -- Focus on nanomolar activities
    AND act.standard_value IS NOT NULL
    AND act.standard_value::TEXT ~ '^[0-9\.]+$'  -- Cast to text for regex matching,
ensuring numeric values
    AND act.standard_value < 100000 --activity value < 100000
LIMIT 100;
```

Explanation:

- `SELECT`: Specifies the columns you want to retrieve.
- `FROM`: Specifies the tables to retrieve data from (`activities`, `molecule_dictionary`, `compound_structures`).
- `JOIN`: Connects the tables based on the `molregno` (molecule registration number).
- `WHERE`: Filters the data based on the following criteria:
    - `act.standard_type IN ('IC50', 'Ki', 'EC50')`: Selects only activity types that are commonly used in drug discovery.
    - `act.standard_units = 'nM'`: Selects only activities that are measured in nanomolars.
    - `act.standard_value IS NOT NULL`: Ensures that only activities with a value are kept.
    - `act.standard_value::TEXT ~ '^[0-9\.]+$'`: This is the key part for fixing your error. It casts `act.standard_value` to `TEXT` before applying the regular expression `^[0-9\.]+$`, which checks if the value consists only of digits and periods. This effectively filters out non-numeric values.
    - `act.standard_value < 100000`: add this condiction to get appropriate activity values
- `LIMIT 100`: Limits the result set to 100 rows.

Python (Jupyter Notebook: `Topic_CheMBL_35_40_1_Data_Loading_and_Preprocessing.ipynb`)

```python
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

# Define the base path for your project
base_path = "."  # Assuming the notebook is in the root of your project directory
data_path = os.path.join(base_path, "data")

# Ensure the data directory exists
if not os.path.exists(data_path):
    os.makedirs(data_path)

# CSV file name (adjust if needed)
csv_file = os.path.join(data_path, "chembl_data.csv")

# Function to calculate molecular descriptors using 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 descriptors

# Load the data from the CSV file
try:
    df = pd.read_csv(csv_file)
except FileNotFoundError:
    print(f"Error: File not found at {csv_file}.  Make sure you've run the SQL query
and saved the data.")
    exit()  # Exit the script if the file is not found

# Data Cleaning and Preprocessing
df = df.dropna(subset=['canonical_smiles', 'standard_value'])  # Drop rows with
missing SMILES or activity values
df['standard_value'] = pd.to_numeric(df['standard_value'], errors='coerce') #convert
standard_value to numeric
df = df.dropna(subset=['standard_value']) #drop if standard_value can not convert to
numeric
df = df[df['standard_value'] > 0] #remove activities < 0
df = df[df['standard_value'] < 100000] #remove activities > 100000, filter data a
second time to avoid the string errors


# Feature Engineering using RDKit
descriptors_list = []
for smiles in df['canonical_smiles']:
    descriptors = calculate_descriptors(smiles)
    descriptors_list.append(descriptors)

descriptors_df = pd.DataFrame(descriptors_list)

# Merge descriptors with the main DataFrame
df = pd.concat([df.reset_index(drop=True), descriptors_df.reset_index(drop=True)],
axis=1)
df = df.dropna()  # Remove rows where RDKit failed to calculate descriptors
```

```
# Prepare data for modeling
X = df[['MolWt', 'LogP', 'HBD', 'HBA', 'TPSA']]
y = np.log10(df['standard_value'])  # Log transform the activity values
# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Model Building (Linear Regression)
model = LinearRegression()
model.fit(X_train, y_train)

# Model Evaluation
y_pred = model.predict(X_test)

# Calculate MSE and RMSE
mse = mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse)  # Calculate RMSE manually

print(f"Mean Squared Error: {mse}")
print(f"Root Mean Squared Error: {rmse}")
```

Explanation:

1. **Imports:** Imports necessary libraries (pandas, RDKit, scikit-learn).
2. **Path Definition:** Defines the base path for the project and constructs the path to the CSV file using `os.path.join`. This makes the code more portable.
3. **Data Loading:** Loads the data from the CSV file into a pandas DataFrame. Includes error handling to gracefully exit if the file isn't found.
4. **Data Cleaning:**
    - Removes rows with missing `canonical_smiles` or `standard_value`.
    - Converts `standard_value` to numeric, coercing errors to NaN and removing them.
5. **Feature Engineering:**
    - Defines a function `calculate_descriptors` to calculate molecular descriptors using RDKit. It handles cases where RDKit fails to process a SMILES string by returning `None`.
    - Iterates through the SMILES strings in the DataFrame, calculates the descriptors, and stores them in a list.
    - Creates a new DataFrame from the list of descriptors and merges it with the original DataFrame.
    - Removes rows where RDKit failed to calculate descriptors.
6. **Data Preparation for Modeling:**
    - Selects the features (descriptors) and the target variable (activity).
    - Log-transforms the activity values. This is a common practice in drug discovery as activity data is often log-normally distributed.
    - Splits the data into training and testing sets.
7. **Model Building:**
    - Creates a Linear Regression model.
    - Trains the model on the training data.
8. **Model Evaluation:**
    - Makes predictions on the test data.
    - Calculates the Mean Squared Error (MSE) and Root Mean Squared Error (RMSE). The RMSE is calculated manually to avoid the `squared=False` error if you're using an older version of scikit-learn.

3. Examples

Here are 5 examples illustrating potential analyses you could perform using this data, each fitting a different notebook. Assume that the initial data loading and preprocessing (from `Topic_CheMBL_35_40_1_Data_Loading_and_Preprocessing.ipynb`) are done and the DataFrame `df` is available.

- **Example 1: `Topic_CheMBL_35_40_2_Activity_Distribution.ipynb` (Activity Distribution Analysis)**

```
import matplotlib.pyplot as plt
import seaborn as sns

plt.figure(figsize=(10, 6))
sns.histplot(df['standard_value'], bins=50, kde=True)
plt.title('Distribution of Standard Values')
plt.xlabel('Standard Value (nM)')
plt.ylabel('Frequency')
plt.xscale('log')  # Use a logarithmic scale for better visualization
plt.show()

print(df['standard_value'].describe())
```

  This notebook analyzes the distribution of the `standard_value` (activity) data. It generates a histogram and provides descriptive statistics. The logarithmic scale is helpful for visualizing the distribution when the data spans several orders of magnitude.

- **Example 2: `Topic_CheMBL_35_40_3_Descriptor_Correlation.ipynb` (Descriptor Correlation Analysis)**

```
import matplotlib.pyplot as plt
import seaborn as sns

# Select descriptor columns
descriptor_cols = ['MolWt', 'LogP', 'HBD', 'HBA', 'TPSA']
correlation_matrix = df[descriptor_cols].corr()

plt.figure(figsize=(8, 6))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', center=0)
plt.title('Correlation Matrix of Molecular Descriptors')
plt.show()
```

  This notebook calculates and visualizes the correlation matrix between the molecular descriptors. This helps identify multicollinearity, which can affect model performance.

- **Example 3: `Topic_CheMBL_35_40_4_Activity_vs_Descriptors.ipynb` (Activity vs. Descriptors Scatter Plots)**

```
import matplotlib.pyplot as plt
import seaborn as sns

descriptor_cols = ['MolWt', 'LogP', 'HBD', 'HBA', 'TPSA']

for descriptor in descriptor_cols:
    plt.figure(figsize=(8, 6))
    sns.scatterplot(x=df[descriptor], y=np.log10(df['standard_value']))
    plt.title(f'Log10(Standard Value) vs. {descriptor}')
    plt.xlabel(descriptor)
    plt.ylabel('Log10(Standard Value)')
    plt.show()
```

This notebook generates scatter plots of each descriptor against the log-transformed activity. This helps visualize the relationship between the descriptors and activity and can inform feature selection.

- **Example 4: `Topic_CheMBL_35_40_5_Target_Activity_Analysis.ipynb` (Target Activity Analysis)**

```
import matplotlib.pyplot as plt
import seaborn as sns

# Assuming you have target information in a column named 'target_chembl_id'
if 'target_chembl_id' in df.columns:
    top_targets = df['target_chembl_id'].value_counts().head(10)
    df_top_targets = df[df['target_chembl_id'].isin(top_targets.index)]

    plt.figure(figsize=(12, 6))
    sns.boxplot(x='target_chembl_id',
y=np.log10(df_top_targets['standard_value']))
    plt.title('Activity Distribution for Top 10 Targets')
    plt.xlabel('Target ChEMBL ID')
    plt.ylabel('Log10(Standard Value)')
    plt.xticks(rotation=45, ha='right')
    plt.tight_layout()
    plt.show()
else:
    print("No 'target_chembl_id' column found.  This example requires target
information.")
```

This notebook analyzes the activity distribution for different targets. It identifies the top 10 targets with the most activity data and generates box plots to compare their activity distributions. You need to make sure your SQL query includes the `target_chembl_id` for this to work.

- **Example 5: `Topic_CheMBL_35_40_6_Advanced_Modeling.ipynb` (Advanced Modeling)**

```
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import GridSearchCV

# Assuming X_train, X_test, y_train, y_test are already defined
# Define a grid of hyperparameters to search
param_grid = {
    'n_estimators': [100, 200, 300],
    'max_depth': [4, 6, 8],
    'min_samples_split': [2, 4],
    'min_samples_leaf': [1, 2]
}

# Create a Random Forest Regressor
rf = RandomForestRegressor(random_state=42)

# Use GridSearchCV to find the best hyperparameters
grid_search = GridSearchCV(estimator=rf, param_grid=param_grid, cv=3,
scoring='neg_mean_squared_error')
grid_search.fit(X_train, y_train)

# Print the best hyperparameters
print("Best Hyperparameters:", grid_search.best_params_)

# Get the best model
best_rf = grid_search.best_estimator_
```

```
# Evaluate the best model on the test set
y_pred = best_rf.predict(X_test)
mse = mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse)

print(f"Mean Squared Error on Test Set: {mse}")
print(f"Root Mean Squared Error on Test Set: {rmse}")
```

This notebook demonstrates advanced modeling using a Random Forest Regressor and GridSearchCV for hyperparameter tuning. It searches for the best combination of hyperparameters to optimize model performance.

Important Considerations:

- **Error Handling:** Add more robust error handling to your code, especially when dealing with external data sources and RDKit calculations.
- **Data Validation:** Validate your data and results throughout the process. Check for inconsistencies and outliers.
- **Model Selection:** Experiment with different machine learning algorithms to find the best model for your data.
- **Feature Selection:** Consider using feature selection techniques to reduce the number of features and improve model performance.
- **Hyperparameter Tuning:** Optimize the hyperparameters of your models using techniques like GridSearchCV or RandomizedSearchCV.
- **Documentation:** Document your code and analysis thoroughly.

In summary:

1. Correct your SQL query to handle the numeric column and regex matching problem.
2. Update or adjust your `mean_squared_error` call in Python based on your scikit-learn version.
3. Follow the AIMLops structure for organizing your project.
4. Use the provided code examples as a starting point and expand them to perform more in-depth analyses.

Let me know if you have any more specific questions or need further assistance with any of these steps.