PharmApp Suite

🧠 AI for Drug Discovery and Development 💉

🧩 Topic: Extracting structure alerts for toxicity

Okay, I understand. You're working with Chembl 35 data using PostgreSQL, RDKit, and Jupyter Notebooks in an AIMLops structure. You're encountering SQL errors and scikit-learn compatibility issues, and you need guidance on running your analysis, specifically focusing on extracting 100 rows and creating examples.

Here's a breakdown, analysis, code examples, and explanations, addressing your concerns.

1. Analysis and Model Overview

Your project likely involves analyzing chemical data from ChEMBL 35 to identify trends, patterns, or relationships relevant to drug discovery. This might include:

- **Data Extraction & Filtering:** You're extracting data from the ChEMBL database based on specific criteria (e.g., activity values, target information). You'll use SQL to filter and retrieve relevant data.
- **Data Preprocessing:** You'll clean and prepare the data. This often involves:
    - Handling missing values.
    - Converting data types.
    - Filtering out invalid or noisy data.
    - Feature Engineering (using RDKit): This is where you'll use RDKit to calculate molecular descriptors (e.g., molecular weight, LogP, number of hydrogen bond donors/acceptors) from the SMILES strings of the compounds. These descriptors serve as features for your analysis.
- **Data Analysis & Modeling:** You'll use machine learning techniques (potentially regression or classification) to model the relationship between the molecular descriptors and the activity values. Common techniques include:
    - **Regression:** Predicting continuous activity values (e.g., IC50, Ki).
    - **Classification:** Predicting whether a compound is active or inactive.

Addressing the Errors

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

  This error indicates that you're trying to use the regular expression operator ~ on a numeric column (`act.standard_value`). ~ is for string matching. The column is likely defined as `numeric` or `double precision` (or similar). You need to either:

  1. **Cast to TEXT:** Explicitly convert the numeric column to text before applying the regular expression: `AND text(act.standard_value) ~ '^[0-9\.]+$'`
  2. **Use Numeric Comparison:** If possible, avoid the regular expression entirely. If you're trying to check if the value is a positive number, you can directly compare it: `AND act.standard_value > 0`
- **Scikit-learn Error:** "old scikit-learn version does not support parameters squared=False in the mean_squared_error function"

  This is a version incompatibility issue. The `squared` parameter was introduced in a later version of scikit-learn. You have a few options:

1. **Upgrade scikit-learn:** The best solution is to upgrade your scikit-learn version to the latest version, or at least one that supports the `squared` parameter. Use `pip install -U scikit-learn`
2. **Remove `squared=False`:** If upgrading isn't feasible, remove the `squared=False` parameter. The default behavior of `mean_squared_error` *without* the `squared` parameter is to return the mean squared error (MSE), which is what you'd usually want. So, just use `mean_squared_error(y_true, y_pred)`. If you specifically need the Root Mean Squared Error (RMSE), take the square root of the result: `rmse = np.sqrt(mean_squared_error(y_true, y_pred))`

2. Code Examples

Here's the code, formatted for your AIMLops structure, and tailored to extract a limited set of 100 rows.

Project Structure:

```
my_chembl_project/
├── data/
│   └── chembl_activity_data.csv  # Output from SQL query
├── notebooks/
│   └── Topic_CheMBL_35_95_1_data_extraction.ipynb
│   └── Topic_CheMBL_35_95_2_analysis_and_modeling.ipynb
├── src/
│   └── utils.py  # (Optional) For reusable functions
└── README.md
```

SQL (Save to `data/chembl_activity_data.csv`)

```sql
-- data/chembl_activity_data.sql
COPY (
    SELECT
        act.molregno,
        act.standard_value,
        act.standard_units,
        act.standard_type,
        md.chembl_id,
        cs.canonical_smiles
    FROM
        activities act
    JOIN
        molecule_dictionary md ON act.molregno = md.molregno
    JOIN
        compound_structures cs ON md.molregno = cs.molregno
    WHERE
        act.standard_type = 'IC50'
        AND act.standard_units = 'nM'
        AND act.standard_value IS NOT NULL
        AND act.standard_value > 0 -- Use direct numeric comparison
        AND cs.canonical_smiles IS NOT NULL
    LIMIT 100 -- Limit to 100 rows
) TO '/tmp/chembl_activity_data.csv' WITH CSV HEADER;
```

Explanation:

- **`COPY ... TO '/tmp/chembl_activity_data.csv'`**: This command copies the results of the query to a CSV file. **Important:** You may need to adjust the path `/tmp/chembl_activity_data.csv` depending on your PostgreSQL server's configuration and permissions. Also, you need to make sure the user `rd` has the permission to write to the file.
- **`WITH CSV HEADER`**: Includes the column names in the first row of the CSV file.

- **`LIMIT 100`**: Restricts the result set to 100 rows.
- The problematic `act.standard_value ~ '^[0-9\.]+$'` has been replaced with `act.standard_value > 0`. This is generally a safer and more efficient way to ensure the value is a positive number.

Running the SQL in pgAdmin:

1. Open pgAdmin and connect to your `chembl_35` database.
2. Open a query window.
3. Paste the SQL code into the query window.
4. Execute the query. The CSV file will be created on the *server* at the specified path (`/tmp/chembl_activity_data.csv`).
5. **Important:** You'll need to *transfer* the CSV file from the server (`192.168.206.136`) to your local machine (where your Jupyter Notebook is running). You can use `scp` (Secure Copy) for this:

   ```
   scp rd@192.168.206.136:/tmp/chembl_activity_data.csv
   ./data/chembl_activity_data.csv
   ```

   Replace `./data/chembl_activity_data.csv` with the correct path on your local machine within your `my_chembl_project` directory.

Python (Jupyter Notebook - `notebooks/Topic_CheMBL_35_95_1_data_extraction.ipynb`)

```
# notebooks/Topic_CheMBL_35_95_1_data_extraction.ipynb
import os
import pandas as pd

base_path = os.getcwd()  # Get current working directory (project root)
data_path = os.path.join(base_path, 'data', 'chembl_activity_data.csv')

try:
    df = pd.read_csv(data_path)
    print("Data loaded successfully:")
    print(df.head())
    print(f"Shape of the dataframe: {df.shape}")
except FileNotFoundError:
    print(f"Error: File not found at {data_path}.  Make sure you've transferred the
CSV from the server.")
except Exception as e:
    print(f"An error occurred: {e}")
```

Explanation:

- **`os.getcwd()`**: Gets the current working directory, which is the root of your project.
- **`os.path.join()`**: Safely joins the path components to create the full path to the CSV file. This ensures platform compatibility.
- **Error Handling:** The `try...except` block handles potential errors like the file not being found or other issues during CSV loading. This is crucial for robustness.
- **Shape Check**: print shape of the dataframe to check if the number of rows is as expected.

Python (Jupyter Notebook - `notebooks/Topic_CheMBL_35_95_2_analysis_and_modeling.ipynb`)

```
# notebooks/Topic_CheMBL_35_95_2_analysis_and_modeling.ipynb
import os
import pandas as pd
```

```
from rdkit import Chem
from rdkit.Chem import AllChem
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


base_path = os.getcwd()
data_path = os.path.join(base_path, 'data', 'chembl_activity_data.csv')

try:
    df = pd.read_csv(data_path)
except FileNotFoundError:
    print(f"Error: File not found at {data_path}")
    exit()

# 1. Data Preprocessing
# Handle missing values (if any) - simple imputation with the mean
df = df.dropna(subset=['canonical_smiles', 'standard_value'])
df['standard_value'] = pd.to_numeric(df['standard_value'], errors='coerce')
df = df.dropna(subset=['standard_value']) # Drop rows where conversion failed
df = df[df['standard_value'] > 0]  # Keep only positive activity values
df['pIC50'] = -np.log10(df['standard_value'] / 1e9)  # Convert IC50 to pIC50

# 2. Feature Engineering (RDKit)
def calculate_descriptors(smiles):
    mol = Chem.MolFromSmiles(smiles)
    if mol is not None:
        AllChem.Compute2DCoords(mol)  # Ensure coordinates are computed for 2D
descriptors
        return [desc[1](mol) for desc in AllChem.CalcMolDescriptors()]
    else:
        return None

# Create a list of descriptor names (ensure it's defined only once)
descriptor_names = [desc[0] for desc in AllChem.CalcMolDescriptors()]

df['descriptors'] = df['canonical_smiles'].apply(calculate_descriptors)
df = df.dropna(subset=['descriptors']) # Drop rows where descriptor calculation failed


# Convert list of descriptors to separate columns
df[descriptor_names] = pd.DataFrame(df['descriptors'].tolist(), index=df.index)
df = df.drop(columns=['descriptors'])

# 3. Data Splitting and Scaling
X = df[descriptor_names]
y = df['pIC50']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Scaling
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

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

# 5. 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}")
print(f"Root Mean Squared Error: {rmse}")
print(f"R-squared: {r2}")

# Example predictions (first 5 test samples)
print("\nExample Predictions:")
for i in range(min(5, len(y_test))):
    print(f"Actual pIC50: {y_test.iloc[i]:.2f}, Predicted pIC50: {y_pred[i]:.2f}")
```

Explanation:

- **RDKit Feature Engineering:**
  - `Chem.MolFromSmiles(smiles)`: Converts a SMILES string to an RDKit molecule object.
  - `AllChem.Compute2DCoords(mol)`: Compute 2D coordinates, necessary for some descriptors.
  - `AllChem.CalcMolDescriptors()`: Calculates a list of molecular descriptors.
  - The code applies this function to each SMILES string in your DataFrame to generate features.
- **Data Scaling:** `StandardScaler` is used to standardize the features (important for many machine learning algorithms).
- **Linear Regression:** A simple linear regression model is used as an example. You can replace this with more sophisticated models.
- **Evaluation:** The code calculates the Mean Squared Error (MSE), Root Mean Squared Error (RMSE), and R-squared to evaluate the model's performance.
- **Example Predictions:** Shows the actual and predicted pIC50 values for the first 5 test samples.

3. Examples

Here are 5 examples of things you could do with this data, building on the code above:

1. **Change the Model:** Instead of Linear Regression, try a different model like Random Forest Regressor, Support Vector Regression (SVR), or a neural network. You'll need to import the appropriate class from `sklearn` and adjust the training and prediction steps.

   ```
   from sklearn.ensemble import RandomForestRegressor

   model = RandomForestRegressor(n_estimators=100, random_state=42)  # Example parameters
   model.fit(X_train, y_train)
   y_pred = model.predict(X_test)
   ```

2. **Add More Descriptors:** RDKit has many more descriptors available. Explore the `rdkit.Chem.Descriptors` module to calculate different types of descriptors and add them to your feature set. Consider things like topological indices, electronic properties, or fragment counts.

3. **Feature Selection:** Use feature selection techniques (e.g., SelectKBest, Recursive Feature Elimination) to identify the most important descriptors for predicting activity. This can improve model performance and reduce complexity.

```
from sklearn.feature_selection import SelectKBest, f_regression

selector = SelectKBest(score_func=f_regression, k=20)  # Select top 20 features
X_train_selected = selector.fit_transform(X_train, y_train)
X_test_selected = selector.transform(X_test)

model = LinearRegression()
model.fit(X_train_selected, y_train)
y_pred = model.predict(X_test_selected)
```

4. **Investigate Outliers:** Identify and investigate outliers in your data. These could be errors in the data, or they could be particularly interesting compounds with unexpected activity. Visualize the data (e.g., scatter plots of predicted vs. actual activity) to help identify outliers.

5. **Substructure Search:** Use RDKit to perform substructure searches. For example, you could search for compounds containing a specific functional group and analyze their activity compared to compounds without that group.

```
from rdkit.Chem import MolFromSmarts

# Define a SMARTS pattern for a substructure (e.g., a benzene ring)
benzene = MolFromSmarts('c1ccccc1')

def has_substructure(smiles, substructure):
    mol = Chem.MolFromSmiles(smiles)
    if mol is not None:
        return mol.HasSubstructure(substructure)
    else:
        return False

df['has_benzene'] = df['canonical_smiles'].apply(lambda x: has_substructure(x,
benzene))

# Analyze the activity of compounds with and without the benzene ring
print(df.groupby('has_benzene')['pIC50'].mean())
```

Important Considerations:

- **Database Permissions:** Make sure the PostgreSQL user (`rd`) has the necessary permissions to read the ChEMBL data and write to the specified file path (`/tmp/chembl_activity_data.csv` or wherever you choose).
- **Data Transfer:** Remember to transfer the CSV file from the PostgreSQL server to your local machine where you're running your Jupyter Notebook.
- **RDKit Installation:** Ensure RDKit is installed in your Jupyter Notebook environment: `conda install -c conda-forge rdkit` or `pip install rdkit`.
- **Error Handling:** Robust error handling is crucial, especially when working with external data sources and complex libraries like RDKit. Use `try...except` blocks to catch potential exceptions and provide informative error messages.
- **Virtual Environments:** Use virtual environments to isolate your project dependencies and avoid conflicts with other projects.

In Summary (En Français):

1. **Analyse du modèle :** Votre projet consiste à analyser les données de ChEMBL 35 pour identifier des tendances en utilisant SQL, RDKit et potentiellement des modèles de machine

learning. Il est important de gérer les erreurs SQL et les problèmes de compatibilité de scikit-learn.

2. **Code SQL :** Le code SQL fourni extrait des données de ChEMBL en limitant le nombre de lignes à 100 et corrige l'erreur liée à la comparaison de valeurs numériques.
3. **Code Python :** Le code Python utilise RDKit pour calculer des descripteurs moléculaires, divise les données en ensembles d'entraînement et de test, et entraîne un modèle de régression linéaire. Il inclut également des exemples d'évaluation du modèle.
4. **Exemples :** Cinq exemples sont fournis pour étendre votre analyse, tels que le changement de modèle, l'ajout de descripteurs, la sélection de caractéristiques, l'investigation des valeurs aberrantes et la recherche de sous-structures.
5. **Considérations importantes :** Vérifiez les permissions de la base de données, le transfert des données, l'installation de RDKit, la gestion des erreurs et l'utilisation d'environnements virtuels.

By following this guide and adapting the code examples to your specific needs, you should be able to effectively analyze Chembl 35 data using RDKit and address the errors you've encountered. Remember to carefully adjust file paths, database credentials, and model parameters to match your environment and research goals. Good luck!