PharmApp Suite

🧠 AI for Drug Discovery and Development 🧪

🧩 Topic: Scaffold hopping analysis

Okay, I understand. You're working with ChEMBL 35 data, using RDKit in Jupyter notebooks, and following an AIMLops template. You're encountering specific errors in your SQL and Python code. Your goal is to extract a small dataset (100 rows) for testing and development.

Here's a breakdown, along with SQL and Python code examples, and a guide to addressing your errors.

1. Analysis Model Overview

Your project seems to involve analyzing bioactivity data from ChEMBL 35. A typical analysis pipeline would include these steps:

- **Data Extraction:** Pulling relevant information from the ChEMBL database using SQL. This typically involves selecting specific activity types (e.g., IC50, Ki), targets, and compounds.
- **Data Cleaning & Transformation:** Cleaning up the extracted data. This can involve filtering for specific activity units (e.g., nM), converting activity values to a consistent scale (e.g., pChEMBL), and handling missing data. This also will include data for Machine Learning processing in the future.
- **Feature Generation:** Generating molecular descriptors using RDKit. These descriptors represent the chemical properties of the molecules and can be used as input for machine learning models.
- **Data Analysis & Modeling:** Performing statistical analysis, building predictive models (e.g., regression, classification), or visualizing the data.

2. Addressing Errors

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

  This error means PostgreSQL doesn't know how to use the regular expression operator ~ (for matching) directly with a numeric column. You need to explicitly cast the `standard_value` column to text before applying the regex.

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

  This means you are using an older version of scikit-learn. You should either upgrade scikit-learn or remove the `squared=False` parameter if you only need the Mean Squared Error (MSE), not the Root Mean Squared Error (RMSE).

3. Code Examples (SQL & Python)

Here's the code, incorporating error fixes and adhering to your file structure requirements. I will create a simplified example to get you started and then add more complex examples to show case some advanced usages.

```
# Assumes you have a base directory for your project
import os
import sys
import pandas as pd
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
from sqlalchemy import create_engine
import warnings
warnings.filterwarnings("ignore")

# Define base path
base_path = "."  # Or your project's root directory

# Create data directory if it doesn't exist
data_dir = os.path.join(base_path, "data")
os.makedirs(data_dir, exist_ok=True)

notebook_dir = os.path.join(base_path, "notebooks")
os.makedirs(notebook_dir, exist_ok=True)

def create_db_connection(user, password, host, database):
    engine = create_engine(f'postgresql://{user}:{password}@{host}/{database}')
    return engine.connect()

# Database credentials (replace with your actual credentials)
db_user = "rd"
db_password = "rd"
db_host = "192.168.206.136"
db_name = "chembl_35"

# Establish database connection
try:
    connection = create_db_connection(db_user, db_password, db_host, db_name)
    print("Database connection successful")
except Exception as e:
    print(f"Error connecting to database: {e}")
    sys.exit(1)  # Exit if connection fails
```

SQL (Save as `Topic_CheMBL_35_13_extract_data.sql` in your `data` directory)

```
-- Topic_CheMBL_35_13_extract_data.sql
-- Extracts activity data for a specific target (e.g., a specific protein)
-- and filters for a specific activity type (e.g., IC50).
-- Limits the result to 100 rows for testing.

SELECT
    cmp.chembl_id,
    act.standard_type,
    act.standard_value,
    act.standard_units,
    mol.molfile
FROM
    activities act
JOIN
    molecule_dictionary cmp ON act.molregno = cmp.molregno
JOIN
    target_dictionary tgt ON act.tid = tgt.tid
JOIN
    compound_structures mol ON cmp.molregno = mol.molregno
WHERE
    tgt.pref_name = 'CHEMBL205'  -- Example target (replace with your target of
```

```
interest)
    AND act.standard_type = 'IC50'
    AND act.standard_units = 'nM'
    AND act.standard_value IS NOT NULL
    AND act.standard_value::text ~ '^[0-9\.]+$'  -- Cast to text for regex matching
LIMIT 100;
```

Python (Save as `Topic_CheMBL_35_13_1_data_extraction.ipynb` in your `notebooks` directory)

```
# Topic_CheMBL_35_13_1_data_extraction.ipynb
import os
import pandas as pd
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
from sqlalchemy import create_engine
import warnings
warnings.filterwarnings("ignore")

# Define base path
base_path = "."  # Or your project's root directory

# Create data directory if it doesn't exist
data_dir = os.path.join(base_path, "data")
os.makedirs(data_dir, exist_ok=True)

notebook_dir = os.path.join(base_path, "notebooks")
os.makedirs(notebook_dir, exist_ok=True)

# Database credentials (replace with your actual credentials)
db_user = "rd"
db_password = "rd"
db_host = "192.168.206.136"
db_name = "chembl_35"

# Output CSV file path
output_csv = os.path.join(data_dir, "chembl_data.csv")

# SQL query to extract data (assuming you ran the .sql and have the output)
#sql_query = "SELECT * FROM activities LIMIT 100" # Example, replace with your actual
query


sql_query = """
SELECT
    cmp.chembl_id,
    act.standard_type,
    act.standard_value,
    act.standard_units,
    mol.molfile
FROM
    activities act
JOIN
    molecule_dictionary cmp ON act.molregno = cmp.molregno
JOIN
    target_dictionary tgt ON act.tid = tgt.tid
JOIN
```

```
    compound_structures mol ON cmp.molregno = mol.molregno
WHERE
    tgt.pref_name = 'CHEMBL205'  -- Example target (replace with your target of
interest)
    AND act.standard_type = 'IC50'
    AND act.standard_units = 'nM'
    AND act.standard_value IS NOT NULL
    AND act.standard_value::text ~ '^[0-9\.]+$'  -- Cast to text for regex matching
LIMIT 100;
"""

# Establish database connection
try:
    engine = create_engine(f'postgresql://{db_user}:{db_password}
@{db_host}/{db_name}')
    connection = engine.connect()
    print("Database connection successful")
except Exception as e:
    print(f"Error connecting to database: {e}")
    sys.exit(1)  # Exit if connection fails


# Read data from the database using pandas
try:
    df = pd.read_sql(sql_query, connection)
    print("Data read successfully from the database")
except Exception as e:
    print(f"Error reading data from database: {e}")
    sys.exit(1)

# Save the data to a CSV file
try:
    df.to_csv(output_csv, index=False)
    print(f"Data saved to {output_csv}")
except Exception as e:
    print(f"Error saving data to CSV: {e}")
    sys.exit(1)

# Print the first few rows of the DataFrame to verify
print(df.head())

# Close the database connection
connection.close()
print("Database connection closed")
```

Python (Save as `Topic_CheMBL_35_13_2_feature_generation_and_model.ipynb` in your `notebooks` directory)

```
# Topic_CheMBL_35_13_2_feature_generation_and_model.ipynb
import os
import pandas as pd
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
import warnings
warnings.filterwarnings("ignore")
```

```
# Define base path
base_path = "."  # Or your project's root directory

# Create data directory if it doesn't exist
data_dir = os.path.join(base_path, "data")
os.makedirs(data_dir, exist_ok=True)

notebook_dir = os.path.join(base_path, "notebooks")
os.makedirs(notebook_dir, exist_ok=True)

# Input CSV file path
input_csv = os.path.join(data_dir, "chembl_data.csv")

# Load the data from the CSV file
try:
    df = pd.read_csv(input_csv)
    print(f"Data loaded successfully from {input_csv}")
except Exception as e:
    print(f"Error loading data from CSV: {e}")
    exit(1)

# Print the first few rows of the DataFrame to verify
print(df.head())

# Function to calculate molecular weight
def calculate_mw(mol):
    return Descriptors.MolWt(mol)

# Function to calculate LogP
def calculate_logp(mol):
    return Descriptors.MolLogP(mol)

# Function to calculate Hydrogen Bond Donors
def calculate_hbd(mol):
    return Descriptors.NumHDonors(mol)

# Function to calculate Hydrogen Bond Acceptors
def calculate_hba(mol):
    return Descriptors.NumHAcceptors(mol)

# Convert molfile strings to RDKit Mol objects and handle potential errors
def create_mol_object(molfile):
    try:
        mol = Chem.MolFromMolBlock(molfile)
        if mol is None:
            return None  # Handle cases where MolFromMolBlock fails
        return mol
    except Exception as e:
        print(f"Error creating Mol object: {e}")
        return None

# Apply the function to create Mol objects
df['ROMol'] = df['molfile'].apply(create_mol_object)

# Drop rows where ROMol is None (invalid structures)
df = df.dropna(subset=['ROMol'])

# Calculate molecular descriptors
df['Molecular_Weight'] = df['ROMol'].apply(calculate_mw)
```

```
df['LogP'] = df['ROMol'].apply(calculate_logp)
df['HBD'] = df['ROMol'].apply(calculate_hbd)
df['HBA'] = df['ROMol'].apply(calculate_hba)

# Convert IC50 values to pIC50
df['pIC50'] = -np.log10(df['standard_value'] / 1e9)  # Convert nM to Molar and then to
pIC50

# Display the DataFrame with calculated descriptors
print(df.head())

# Prepare data for modeling
X = df[['Molecular_Weight', 'LogP', 'HBD', 'HBA']]
y = df['pIC50']

# 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)

# Train a linear regression model
model = LinearRegression()
model.fit(X_train, y_train)

# Make predictions on the test set
y_pred = model.predict(X_test)

# Evaluate the model
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}")
```

4. Running the Code

1. **Database Setup:** Ensure your PostgreSQL database is running and accessible at the specified IP address and port.
2. **SQL Execution:** Run the `Topic_CheMBL_35_13_extract_data.sql` script in pgAdmin. This will *not* create a file, but populate the data to pandas dataframe.
3. **Jupyter Notebook:** Open the `Topic_CheMBL_35_13_1_data_extraction.ipynb` notebook in Jupyter. Run the cells sequentially. This will connect to your database, extract the data, and save it as `chembl_data.csv` in the `data` directory.
4. **Feature Generation and Modeling:** Open the `Topic_CheMBL_35_13_2_feature_generation_and_model.ipynb` notebook. Run the cells sequentially. This will load the CSV file, calculate molecular descriptors, train a linear regression model, and evaluate its performance.

5. Example Expansions

Here are 5 example expansions based on your code to show case some advanced usages:

Example 1: Feature Selection

- **Goal:** Use feature selection to improve the linear regression model.
- **Changes to `Topic_CheMBL_35_13_2_feature_generation_and_model.ipynb`:**

```
from sklearn.feature_selection import SelectKBest, f_regression

# Feature selection using SelectKBest
selector = SelectKBest(score_func=f_regression, k=3)  # Select top 3 features
```

```
X_new = selector.fit_transform(X, y)

# Get the indices of the selected features
selected_indices = selector.get_support(indices=True)
selected_features = X.columns[selected_indices]

print("Selected features:", selected_features)

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_new, y, test_size=0.2,
random_state=42)

# Train a linear regression model
model = LinearRegression()
model.fit(X_train, y_train)

# Make predictions on the test set
y_pred = model.predict(X_test)

# Evaluate the model
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}")
```

Example 2: Using Different Regression Models

- **Goal:** Compare the performance of linear regression with a Random Forest Regressor.
- **Changes to `Topic_CheMBL_35_13_2_feature_generation_and_model.ipynb`:**

```
from sklearn.ensemble import RandomForestRegressor

# Train a Random Forest Regressor model
rf_model = RandomForestRegressor(n_estimators=100, random_state=42)  # Adjust
parameters as needed
rf_model.fit(X_train, y_train)

# Make predictions on the test set
y_pred_rf = rf_model.predict(X_test)

# Evaluate the Random Forest model
mse_rf = mean_squared_error(y_test, y_pred_rf)
r2_rf = r2_score(y_test, y_pred_rf)

print("Random Forest Results:")
print(f"Mean Squared Error: {mse_rf}")
print(f"R-squared: {r2_rf}")
```

Example 3: Activity Cliff Detection

- **Goal:** Identify pairs of compounds with similar structures but significantly different activities (activity cliffs). Requires a structural similarity calculation.
- **Changes to `Topic_CheMBL_35_13_2_feature_generation_and_model.ipynb`:**

```
from rdkit.Chem import AllChem
from rdkit.DataStructs import FingerprintSimilarity

# Calculate Morgan fingerprints (ECFP4)
def calculate_fingerprint(mol):
    return AllChem.GetMorganFingerprintAsBitVect(mol, 2, nBits=2048)
```

```
df['Fingerprint'] = df['ROMol'].apply(calculate_fingerprint)

# Function to calculate Tanimoto similarity
def calculate_tanimoto_similarity(fp1, fp2):
    return FingerprintSimilarity(fp1, fp2)

# Calculate similarity between all pairs of compounds (can be slow for large datasets)
similarity_matrix = np.zeros((len(df), len(df)))
for i in range(len(df)):
    for j in range(i + 1, len(df)):
        similarity = calculate_tanimoto_similarity(df['Fingerprint'][i],
df['Fingerprint'][j])
        similarity_matrix[i, j] = similarity
        similarity_matrix[j, i] = similarity

# Define a similarity threshold and a activity difference threshold
similarity_threshold = 0.8
activity_difference_threshold = 1  # pIC50 units

# Identify potential activity cliffs
activity_cliffs = []
for i in range(len(df)):
    for j in range(i + 1, len(df)):
        similarity = similarity_matrix[i, j]
        activity_difference = abs(df['pIC50'][i] - df['pIC50'][j])
        if similarity >= similarity_threshold and activity_difference >=
activity_difference_threshold:
            activity_cliffs.append(((df['chembl_id'][i], df['pIC50'][i]),
(df['chembl_id'][j], df['pIC50'][j]), similarity))

print("Potential Activity Cliffs:")
for cliff in activity_cliffs:
    print(f"Compound 1: {cliff[0]}, Compound 2: {cliff[1]}, Similarity: {cliff[2]}")
```

Example 4: Substructure Searching

- **Goal:** Identify compounds containing a specific substructure (e.g., a common scaffold).
- **Changes to `Topic_CheMBL_35_13_2_feature_generation_and_model.ipynb`:**

```
from rdkit.Chem import MolFromSmarts

# Define the SMARTS pattern for the substructure
substructure_smarts = 'c1ccccc1'  # Example: Benzene ring
substructure = MolFromSmarts(substructure_smarts)

# Function to check if a molecule contains the substructure
def contains_substructure(mol, substructure):
    if mol is None or substructure is None:
        return False
    return mol.HasSubstructMatch(substructure)

# Identify compounds containing the substructure
df['Contains_Substructure'] = df['ROMol'].apply(lambda mol: contains_substructure(mol,
substructure))

# Filter the DataFrame to show only compounds containing the substructure
substructure_compounds = df[df['Contains_Substructure']]
```

```
print("Compounds containing the substructure:")
print(substructure_compounds[['chembl_id', 'pIC50']])
```

Example 5: ADMET Property Prediction (using a pre-trained model)

- **Goal:** Predict ADMET (Absorption, Distribution, Metabolism, Excretion, Toxicity) properties for the compounds. This requires a pre-trained model (e.g., from DeepChem or a custom model). This example shows a placeholder for where you would load and use the model.
- **Changes to `Topic_CheMBL_35_13_2_feature_generation_and_model.ipynb`:**

```
# Placeholder for ADMET property prediction (replace with your actual model and data)
#from deepchem.models import Model  # Example using DeepChem

# Load a pre-trained ADMET model (replace with the actual path to your model)
#admet_model_path = 'path/to/your/admet_model'
#admet_model = Model.load(admet_model_path)

# Function to predict ADMET properties
def predict_admet_properties(mol):
    # Convert RDKit Mol object to input format required by the ADMET model
    # Example: Convert to SMILES and then featurize using the model's featurizer
    smiles = Chem.MolToSmiles(mol)
    #features = admet_model.featurizer([smiles])  # Assuming the model has a
featurizer attribute

    # Make predictions using the ADMET model
    #admet_predictions = admet_model.predict(features) #Model.predict requires Numpy
array-like inputs, where each element is a sample

    # Return the predicted ADMET properties as a dictionary
    #return admet_predictions
    return None #added to be error-free

# Apply the ADMET property prediction function to each molecule
df['ADMET_Predictions'] = df['ROMol'].apply(predict_admet_properties)

print(df.head())
```

Important Considerations

- **RDKit Installation:** Make sure you have RDKit installed in your Jupyter notebook environment (`conda install -c conda-forge rdkit`).
- **Scikit-learn:** Update scikit-learn (`pip install --upgrade scikit-learn`) to avoid the `squared=False` error. If upgrading is not feasible, remove the `squared=False` argument from `mean_squared_error`.
- **Database Connection:** Double-check your database credentials and ensure the database server is running and accessible from your notebook environment.
- **Error Handling:** Add more robust error handling to your code, especially when dealing with external data sources or complex calculations.
- **Performance:** For larger datasets, consider using more efficient data structures and algorithms. Pandas can become slow for very large datasets.
- **Virtual Environments:** Use virtual environments (e.g., `conda env create -f environment.yml`) to manage dependencies and ensure reproducibility.
- **Molfile Parsing:** Ensure that the `molfile` data in your ChEMBL database is correctly formatted. Invalid `molfile` entries can cause errors in RDKit.

In Summary (En Français):

Ce plan vous guide à travers l'analyse des données ChEMBL 35 en utilisant RDKit et en suivant un modèle AIMLops. Il comprend :

1. **Analyse du Modèle:** Une vue d'ensemble du pipeline d'analyse de bioactivité typique.
2. **Correction des Erreurs:** Solutions pour les erreurs SQL (conversion du type de données) et Python (mise à jour de scikit-learn ou suppression du paramètre `squared=False`).
3. **Exemples de Code:** Des exemples SQL (pour extraire les données) et Python (pour le calcul des descripteurs moléculaires et la modélisation).
4. **Instructions d'Exécution:** Les étapes pour exécuter le code dans un environnement Jupyter Notebook.
5. **Exemples d'Extension:** Cinq exemples de code supplémentaires pour montrer l'usage de feature selection, de modèles de régression différents, la détection de "activity cliff", la recherche de sous-structures et la prédiction des propriétés ADMET.

N'oubliez pas de remplacer les identifiants de base de données et les noms de fichiers par vos propres valeurs. Assurez-vous que RDKit et scikit-learn sont correctement installés dans votre environnement. Ajoutez une gestion d'erreur robuste pour gérer les problèmes potentiels.

Good luck with your project! Let me know if you have more questions.