PharmApp Suite

🧠 AI for Drug Discovery and Development 🧪

🧩 Topic: Case study: JAK inhibitors from ChEMBL

Okay, I understand. You're working on a Chembl 35 data analysis project using RDKit, following an AIMLops-style directory structure, and encountering specific issues related to SQL and scikit-learn versions. Your goal is to extract a manageable subset (100 rows) of data and perform some analysis.

Here's a breakdown of the analysis, code examples (SQL and Python), and solutions to your errors, all tailored to the "Topic_CheMBL_35_91" convention:

1. Analysis of the Analysis Model

The project aims to extract and analyze data from the Chembl 35 database related to bioactivity. A typical workflow includes:

- **Data Extraction (SQL):** This involves querying the Chembl database to retrieve relevant information like compound structures (SMILES), target information, and bioactivity data (e.g., IC50, Ki values). The query should filter for relevant data, likely based on specific targets, assays, or activity types.
- **Data Preprocessing (Python/RDKit):** RDKit is used to process the SMILES strings, generate molecular fingerprints, calculate descriptors, and prepare the data for machine learning models. This step often includes handling missing values, outliers, and data normalization.
- **Exploratory Data Analysis (Python):** Understanding dataset structure. This helps to uncover patterns, anomalies, or relationships.
- **Model Building (Python/Scikit-learn):** Machine learning models (e.g., regression, classification) are used to predict bioactivity based on the calculated descriptors. Common models include linear regression, random forests, and support vector machines.
- **Model Evaluation (Python/Scikit-learn):** The models are evaluated using appropriate metrics (e.g., R-squared, RMSE for regression; accuracy, AUC for classification) on a held-out test set.
- **Interpretation and Visualization (Python):** Results are visualized and interpreted to understand the factors driving bioactivity. This might involve generating scatter plots, histograms, or visualizing molecular structures with activity information.

The analysis will likely involve establishing relationships between chemical structures and their biological activity.

2. SQL Code (Topic_CheMBL_35_91_1.sql)

This SQL script extracts a limited number of data (100 rows) from the Chembl database, specifically focusing on activities with standard values.

```sql
-- Topic_CheMBL_35_91_1.sql

-- Connect to the chembl_35 database
\c chembl_35 rd rd 192.168.206.136

-- Extract relevant data with a limit of 100 rows
SELECT
    md.molregno,
    cs.canonical_smiles,
    act.standard_type,
    act.standard_value,
```

```
    act.standard_units,
    act.assay_id
FROM
    molecule_dictionary md
JOIN
    compound_structures cs ON md.molregno = cs.molregno
JOIN
    activities act ON md.molregno = act.molregno
WHERE
    act.standard_type IN ('IC50', 'Ki', 'Kd') -- Filter for relevant activity types
    AND act.standard_value IS NOT NULL -- Ensure standard value exists
    AND act.standard_units = 'nM' -- Filter for standard units
    AND act.standard_value::text ~ '^[0-9\.]+$' -- Corrected numeric check
LIMIT 100;

-- Save the output to a CSV file
\copy (SELECT
    md.molregno,
    cs.canonical_smiles,
    act.standard_type,
    act.standard_value,
    act.standard_units,
    act.assay_id
FROM
    molecule_dictionary md
JOIN
    compound_structures cs ON md.molregno = cs.molregno
JOIN
    activities act ON md.molregno = act.molregno
WHERE
    act.standard_type IN ('IC50', 'Ki', 'Kd') -- Filter for relevant activity types
    AND act.standard_value IS NOT NULL -- Ensure standard value exists
    AND act.standard_units = 'nM' -- Filter for standard units
    AND act.standard_value::text ~ '^[0-9\.]+$' -- Corrected numeric check
LIMIT 100) TO '../data/chembl_35_subset.csv' WITH CSV HEADER;


-- Explanation:
-- *   `\c chembl_35 rd rd 192.168.206.136`: Connects to the database.
-- *   `SELECT ... FROM ...`: Selects the desired columns from the tables.
-- *   `JOIN`: Joins tables based on `molregno` (molecule registration number).
-- *   `WHERE`: Filters the data based on activity type, existence of standard value,
and units.
-- *   `LIMIT 100`: Restricts the output to 100 rows.
-- *   `\copy ... TO ...`: Saves the query result to a CSV file.  Make sure the path
`../data/chembl_35_subset.csv` is valid relative to where you run the `psql` command.
```

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

The issue is that the `~` operator (regular expression match) cannot be directly applied to a `numeric` type. To fix this, cast the `standard_value` to `text` before applying the regular expression.

```
AND act.standard_value::text ~ '^[0-9\.]+$'
```

This casts the `standard_value` to a text string, allowing the regular expression to match. The regex `^[0-9\.]+$` ensures that the value consists only of digits and periods.

Run this script in pgAdmin. This will create the `chembl_35_subset.csv` file in your `../data/` directory.

3. Python Code (Topic_CheMBL_35_91_2.ipynb)

This Jupyter Notebook code reads the CSV file, processes the data using RDKit, calculates descriptors, and performs a simple analysis.

```
# Topic_CheMBL_35_91_2.ipynb

import pandas as pd
import numpy as np
import os
from rdkit import Chem
from rdkit.Chem import Descriptors
from rdkit.Chem import AllChem
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 (important for AIMLops structure)
base_path = os.path.dirname(os.getcwd()) # Go up one level to the project root
data_path = os.path.join(base_path, 'data', 'chembl_35_subset.csv') # Full path to the
CSV

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

# Data Cleaning and Preprocessing
df = df.dropna(subset=['canonical_smiles', 'standard_value']) # Remove rows with
missing SMILES or values
df = df[df['standard_value'] > 0] # Remove non-positive values (important for log
transform)
df = df.drop_duplicates(subset=['canonical_smiles']) # Remove duplicate SMILES entries

# Convert standard_value to pIC50 (or pKi, pKd)
df['pActivity'] = -np.log10(df['standard_value'] * 1e-9)  # Convert nM to M, then to
pActivity

# RDKit Descriptor Calculation
def calculate_descriptors(smiles):
    mol = Chem.MolFromSmiles(smiles)
    if mol is None:
        return None
    try:
        AllChem.Compute2DCoords(mol)  # Ensure 2D coordinates are computed
        descriptors = {desc_name: desc_func(mol) for desc_name, desc_func in
Descriptors.descList}
        return descriptors
    except Exception as e:
        print(f"Error calculating descriptors for {smiles}: {e}")
        return None

df['descriptors'] = df['canonical_smiles'].apply(calculate_descriptors)

# Handle missing descriptors
```

```
df = df.dropna(subset=['descriptors'])

# Convert descriptors to DataFrame
descriptors_df = pd.DataFrame(df['descriptors'].tolist(), index=df.index)
df = pd.concat([df, descriptors_df], axis=1)

# Select features and target
X = df.iloc[:, 12:]  # Select all descriptor columns (adjust index if needed)
y = df['pActivity']

# 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 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}")

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

Explanation:

1. **Import Libraries:** Imports necessary libraries.
2. **File Path:** Constructs the full file path using `os.path.join` to adhere to the AIMLops directory structure. This is crucial for portability and maintainability.
3. **Data Loading:** Loads the CSV data using `pandas`. Includes error handling for the case where the file is not found.
4. **Data Cleaning:**
    - Removes rows with missing SMILES strings or standard values.
    - Removes duplicate SMILES to avoid bias in the model.
    - Removes non-positive standard_value to prepare for logarithmic transformation.
5. **pActivity Calculation:** Converts `standard_value` (assumed to be in nM) to pActivity (e.g., pIC50) using `-log10(value * 1e-9)`.
6. **Descriptor Calculation:**
    - Defines a function `calculate_descriptors` that takes a SMILES string as input and returns a dictionary of RDKit descriptors.
    - Applies the function to the `canonical_smiles` column to create a new 'descriptors' column.
7. **Handle Missing Descriptors:** *Remove rows with missing descriptors
8. **Descriptor DataFrame:**
    - Extracts the calculated descriptors from the dictionary format into a Pandas DataFrame (`descriptors_df`).
    - Concatenates the descriptor DataFrame with the original DataFrame.

9. **Feature and Target Selection:** Selects the descriptor columns (adjust index if necessary) as features (X) and `pActivity` as the target variable (y).
10. **Data Splitting:** Splits the data into training and testing sets.
11. **Model Training:** Creates a linear regression model and trains it on the training data.
12. **Model Evaluation:** Predicts pActivity values for the test set and calculates the mean squared error (MSE) and R-squared.
13. **Visualization:** Creates a scatter plot of actual vs. predicted pActivity values.

Correction for Python Error (b): `old scikit-learn version does not support parameters squared=False in the mean_squared_error function`

The `squared=False` parameter was added to `mean_squared_error` in a later version of scikit-learn. Since you can't upgrade (as per your instructions), the solution is to take the square root of the MSE to get the RMSE (Root Mean Squared Error):

```
mse = mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse)  # Calculate RMSE
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}")
```

Important Notes:

- **Directory Structure:** Double-check that the paths used in `os.path.join` are correct for your AIMLops project structure. This is critical for reproducibility. The example assumes that the notebook is located one level deep within the project directory.
- **Descriptor Selection:** You might want to experiment with different sets of descriptors. Not all descriptors are relevant for every problem.
- **Model Selection:** Linear regression is a simple starting point. Try other models like Random Forest, Gradient Boosting, or Support Vector Machines, which may perform better.
- **Feature Scaling:** Consider scaling your features (e.g., using `StandardScaler` from scikit-learn) before training the model. This can improve the performance of some machine-learning algorithms.
- **Error Handling:** The code includes basic error handling for file loading and descriptor calculation. Add more robust error handling as needed.
- **Virtual Environment:** It is *highly* recommended to use a virtual environment (e.g., `venv`, `conda`) to manage your Python dependencies. This will prevent conflicts between different projects.

4. Code in English and Chinese:

English:

- The SQL script extracts compound data and bioactivity information from the ChEMBL database.
- The Python script calculates molecular descriptors using RDKit.
- A Linear Regression model is trained to predict bioactivity based on the descriptors.
- The model's performance is evaluated using Mean Squared Error and R-squared.

Chinese:

- SQL 脚本从 ChEMBL 数据库提取化合物数据和生物活性信息。
- Python 脚本使用 RDKit 计算分子描述符。
- 训练线性回归模型，以基于描述符预测生物活性。
- 使用均方误差和 R 平方评估模型的性能。

5. 5 Examples of Analysis/Tasks you can perform

Here are 5 examples of more specific tasks you can perform with this data:

1. **Predict pIC50 of a novel compound:** Given a SMILES string of a new compound, calculate its descriptors and use your trained model to predict its pIC50 value. This is a classic virtual screening application.

```
new_smiles = 'Cc1ccccc1C(=O)O' # Example SMILES
new_mol = Chem.MolFromSmiles(new_smiles)
if new_mol is not None:
    new_descriptors = calculate_descriptors(new_smiles)
    if new_descriptors is not None:
        new_descriptors_df = pd.DataFrame([new_descriptors])
        # Ensure the new descriptors DataFrame has the same columns as the
training data
        missing_cols = set(X_train.columns) - set(new_descriptors_df.columns)
        for c in missing_cols:
            new_descriptors_df[c] = 0  # Or use a more appropriate imputation
method
        new_descriptors_df = new_descriptors_df[X_train.columns]  # Ensure
correct column order
        predicted_pIC50 = model.predict(new_descriptors_df)[0]
        print(f"Predicted pIC50 for {new_smiles}: {predicted_pIC50}")
    else:
        print(f"Could not calculate descriptors for {new_smiles}")
else:
    print(f"Invalid SMILES string: {new_smiles}")
```

2. **Identify the most important descriptors for bioactivity:** Examine the coefficients of your linear regression model to determine which descriptors have the largest impact on the predicted pIC50. This can provide insights into the structural features that drive activity.

```
coefficients = pd.DataFrame({'Descriptor': X_train.columns, 'Coefficient':
model.coef_})
coefficients['Absolute_Coefficient'] = abs(coefficients['Coefficient'])
coefficients = coefficients.sort_values('Absolute_Coefficient', ascending=False)
print("Top 10 Most Important Descriptors:")
print(coefficients.head(10))
```

3. **Build a QSAR model for a specific target:** Filter your data to only include compounds that target a specific protein or enzyme. Then, build a QSAR (Quantitative Structure-Activity Relationship) model specifically for that target. This will likely result in a more accurate model than a general QSAR model. The `act.assay_id` field in the SQL query can be used for this filtering.

4. **Compare different machine learning models:** Evaluate the performance of different machine learning algorithms (e.g., Random Forest, SVM) on your data. Compare their performance metrics (e.g., MSE, R-squared) to determine which model performs best. Remember to tune hyperparameters using cross-validation.

5. **Visualize the chemical space:** Use dimensionality reduction techniques (e.g., PCA, t-SNE) to visualize the chemical space of your compounds based on their descriptors. Color-code the points by pIC50 value to see if there are any clusters of active compounds.

By following these steps and examples, you should be able to successfully perform your Chembl 35 data analysis project, address the errors you were encountering, and gain valuable insights from the data. Remember to adapt the code to your specific needs and project requirements. Good luck!