PharmApp Suite

🧠 AI for Drug Discovery and Development 💉

🧩 Topic: Designing focused libraries from ChEMBL

Okay, I understand. I'll help you set up your ChEMBL 35 data analysis project using the AIMLops template, address the errors you've encountered, and provide example code for your "Topic_CheMBL_35_15" topic.

Analysis Model:

The goal is to leverage ChEMBL 35 data, accessed via PostgreSQL using `rdkit`, to contribute to drug research and development. Your workflow involves:

1. **Data Extraction (SQL):** Querying the ChEMBL 35 database using SQL via pgAdmin to retrieve relevant data and saving it as CSV files in the `../data/` directory. This allows you to pre-process and filter data directly at the database level, improving efficiency.
2. **Data Analysis (Python/Jupyter Notebook):** Loading the extracted CSV data into a Jupyter Notebook environment. The `rdkit` library will be used to work with chemical structures, and you'll likely be performing analyses such as:
    - **Structure-Activity Relationship (SAR) Analysis:** Relating chemical structure features (e.g., molecular descriptors, fingerprints) to activity values (e.g., IC50, Ki).
    - **Data Visualization:** Creating plots and charts to explore relationships between chemical features and activity.
    - **Model Building:** Developing predictive models to estimate the activity of new compounds based on their chemical structure.

Addressing the Errors:

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

  This error indicates that you are trying to use a regular expression operator (~) on a numeric column (`act.standard_value`). PostgreSQL likely does not allow regular expression matching directly on numeric types. You'll need to cast the numeric column to text before applying the regular expression.

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

  This means that you are using an older version of scikit-learn. The `squared=False` parameter was added to `mean_squared_error` in a later version. You have two solutions: 1) upgrade your scikit-learn installation (recommended), or 2) remove the `squared=False` argument and take the square root of the result manually to get the Root Mean Squared Error (RMSE).

Folder Structure (Based on AIMLops Template):

Assuming a typical AIMLops structure, here's how the folder structure might look:

```
my_chembl_project/
├── data/           # Contains the extracted CSV files from the database.  e.g.,
activity_data.csv
├── notebooks/      # Contains your Jupyter Notebooks.  e.g.,
Topic_CheMBL_35_15_1_data_exploration.ipynb
```

```
├── models/         # (Optional)  Stores saved model files (e.g., pickled scikit-learn models).
├── src/            # (Optional) Python modules containing reusable code.
├── reports/        # (Optional) Generated reports and figures.
└── README.md
```

Topic: `Topic_CheMBL_35_15` (Let's assume this topic focuses on a specific target or assay)

For demonstration purposes, let's assume `Topic_CheMBL_35_15` is focused on **"Inhibitors of EGFR Kinase"**.

SQL Code (to extract data):

```sql
-- File: ../data/egfr_inhibitors.csv

SELECT
    cmp.chembl_id AS compound_chembl_id,
    cmp.pref_name AS compound_name,
    act.standard_type,
    act.standard_value,
    act.standard_units,
    act.activity_comment,
    mol.molfile AS molfile -- Include molfile
FROM
    activities act
JOIN
    molecule_dictionary cmp ON act.molregno = cmp.molregno
JOIN
    assays ass ON act.assay_id = ass.assay_id
JOIN
    target_dictionary tgt ON ass.tid = tgt.tid
LEFT JOIN
    mols mol ON cmp.molregno = mol.molregno
WHERE
    tgt.pref_name = 'Epidermal Growth Factor Receptor' -- EGFR Kinase
    AND act.standard_type IN ('IC50', 'Ki', 'Kd')  -- Common activity types
    AND act.standard_units = 'nM'
    AND act.standard_value IS NOT NULL
    AND act.standard_value::text ~ '^[0-9\.]+$'   -- Corrected line: Cast to text for regex
ORDER BY
    act.standard_value ASC
LIMIT 100;  -- Limit to 100 rows
```

Explanation:

- **`cmp.chembl_id, cmp.pref_name`**: Selects the ChEMBL ID and preferred name of the compound.
- **`act.standard_type, act.standard_value, act.standard_units`**: Selects the activity type (e.g., IC50), value, and units.
- **`tgt.pref_name = 'Epidermal Growth Factor Receptor'`**: Filters for activities against EGFR.
- **`act.standard_type IN ('IC50', 'Ki', 'Kd')`**: Filters for common activity types.
- **`act.standard_units = 'nM'`**: Ensures activity values are in nM.
- **`act.standard_value IS NOT NULL`**: Excludes rows with missing standard values.
- **`act.standard_value::text ~ '^[0-9\.]+$'`**: **Important Correction:** This line filters `standard_value` to ensure it contains only positive numeric values (integers or decimals). It *also* addresses the PostgreSQL error by explicitly casting `act.standard_value` to `text` before applying the regular expression. This allows the regular expression operator to work correctly.

- **`LIMIT 100`**: Restricts the number of returned rows to 100. This is important for performance during development.
- **`mol.molfile AS molfile`**: includes the molfile in the data

How to Run:

1. Open pgAdmin.
2. Connect to your PostgreSQL server (IP: 192.168.206.136, user: rd, password: rd, database: chembl_35).
3. Open a new query window.
4. Paste the SQL code into the query window.
5. Execute the query.
6. Save the results as a CSV file named `egfr_inhibitors.csv` in the `../data/` directory of your project. (Make sure you choose CSV format when saving).

Python Code (Jupyter Notebook: `notebooks/Topic_CheMBL_35_15_1_data_exploration.ipynb`):

```
import os
import pandas as pd
from rdkit import Chem
from rdkit.Chem import Descriptors
from rdkit.Chem import Lipinski
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
import numpy as np

# Define base path for project
base_path = os.path.abspath(os.path.join(os.getcwd(), ".."))  # Assuming notebook is
in notebooks/

# Construct the data path
data_path = os.path.join(base_path, "data", "egfr_inhibitors.csv")

# Load the data
try:
    df = pd.read_csv(data_path)
    print("Data loaded successfully from:", data_path)
except FileNotFoundError:
    print(f"Error: File not found at {data_path}.  Make sure you ran the SQL query and
saved the CSV file.")
    exit()

# Data Cleaning and Preprocessing
# Convert standard_value to numeric, handling potential errors
df['standard_value'] = pd.to_numeric(df['standard_value'], errors='coerce')
df = df.dropna(subset=['standard_value'])  # Remove rows with invalid standard_value

# Convert IC50/Ki/Kd to pIC50 (or pKi/pKd)
def convert_to_pActivity(standard_value):
  return -np.log10(standard_value/10**9)

df['pActivity'] = df['standard_value'].apply(convert_to_pActivity)


# RDKit Feature Calculation
def calculate_descriptors(mol):
```

```
    try:
        mol = Chem.MolFromSmiles(mol)
        if mol is None:
            return None
        descriptors = {
            'MW': Descriptors.MolWt(mol),
            'LogP': Descriptors.MolLogP(mol),
            'HBD': Lipinski.NumHDonors(mol),
            'HBA': Lipinski.NumHAcceptors(mol),
            'TPSA': Descriptors.TPSA(mol)
        }
        return descriptors
    except Exception as e:
        print(f"Error calculating descriptors: {e}")
        return None

# Create smiles from molfile
def molfile_to_smiles(molfile):
    try:
        mol = Chem.MolFromMolBlock(molfile)
        if mol:
            smiles = Chem.MolToSmiles(mol)
            return smiles
        else:
            return None
    except Exception as e:
        print(f"Error converting molfile to SMILES: {e}")
        return None

df['smiles'] = df['molfile'].apply(molfile_to_smiles)
df = df.dropna(subset=['smiles'])  # Remove rows with invalid smiles

#Apply descriptior calculation
df['descriptors'] = df['smiles'].apply(calculate_descriptors)

# Convert the dictionary to individual columns in the DataFrame
df = pd.concat([df, df['descriptors'].apply(pd.Series)], axis=1)

# Drop rows where descriptor calculation failed
df = df.dropna(subset=['MW', 'LogP', 'HBD', 'HBA', 'TPSA'])

# Simple Linear Regression Model
X = df[['MW', 'LogP', 'HBD', 'HBA', 'TPSA']]
y = df['pActivity']

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

model = LinearRegression()
model.fit(X_train, y_train)

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

Explanation:

- **Import Libraries:** Imports necessary libraries.
- **Define Paths:** Defines the base path and data path using `os.path.join` for platform independence.
- **Load Data:** Loads the CSV data into a Pandas DataFrame. Includes error handling for file not found.
- **Data Cleaning:**
  - Convert `standard_value` to numeric and handles errors, dropping rows with non-numeric values.
  - Converts standard values (IC50, Ki, Kd) to pActivity (e.g., pIC50). This is a common practice in SAR analysis.
- **RDKit Feature Calculation:**
  - The `calculate_descriptors` function calculates molecular weight (MW), LogP, hydrogen bond donors (HBD), hydrogen bond acceptors (HBA), and topological polar surface area (TPSA) using RDKit. It handles potential errors during SMILES parsing.
- **SMILES from molfile:**
  - The `molfile_to_smiles` function converts the molfile to SMILES using RDKit.
- **Model Building (Linear Regression):**
  - Selects features (MW, LogP, HBD, HBA, TPSA) and the target variable (pActivity).
  - Splits the data into training and testing sets.
  - Creates a Linear Regression model, trains it on the training data, and makes predictions on the testing data.
  - Calculates Mean Squared Error (MSE), Root Mean Squared Error (RMSE), and R-squared (R2) to evaluate the model's performance.
- **Visualization:** Creates a scatter plot of actual vs. predicted pActivity values.

How to Run:

1. Open the Jupyter Notebook (`notebooks/Topic_CheMBL_35_15_1_data_exploration.ipynb`).
2. Run each cell in the notebook sequentially.
3. Examine the output, including the model performance metrics and the visualization.

5 Examples

Here are 5 example analyses you can perform with this data:

1. **Correlation Analysis:** Calculate the correlation matrix between the molecular descriptors (MW, LogP, HBD, HBA, TPSA) and pActivity to identify the descriptors that are most strongly correlated with activity. Visualize the correlation matrix using a heatmap.

   ```
   correlation_matrix = df[['pActivity', 'MW', 'LogP', 'HBD', 'HBA',
   'TPSA']].corr()
   sns.heatmap(correlation_matrix, annot=True, cmap="coolwarm")
   plt.title("Correlation Matrix")
   plt.show()
   ```

2. **Distribution Plots:** Create distribution plots (histograms or kernel density estimates) for the molecular descriptors and pActivity to understand their distributions.

```
sns.histplot(df['pActivity'], kde=True)
plt.title("Distribution of pActivity")
plt.show()

sns.histplot(df['LogP'], kde=True)
plt.title("Distribution of LogP")
plt.show()
```

3. **Scatter Plots with Regression Lines:** Create scatter plots of each molecular descriptor against pActivity, with a regression line overlaid to visualize the linear relationship.

```
sns.regplot(x='LogP', y='pActivity', data=df)
plt.title("LogP vs. pActivity")
plt.show()
```

4. **Lipinski's Rule of Five Analysis:** Assess how many compounds in your dataset violate Lipinski's Rule of Five (MW <= 500, LogP <= 5, HBD <= 5, HBA <= 10).

```
def lipinski_violations(row):
    violations = 0
    if row['MW'] > 500:
        violations += 1
    if row['LogP'] > 5:
        violations += 1
    if row['HBD'] > 5:
        violations += 1
    if row['HBA'] > 10:
        violations += 1
    return violations

df['Lipinski_Violations'] = df.apply(lipinski_violations, axis=1)
print(df['Lipinski_Violations'].value_counts())
```

5. **Substructure Search:** Use RDKit to search for compounds containing a specific substructure (e.g., a common pharmacophore).

```
from rdkit.Chem.Draw import IPythonConsole
from rdkit.Chem import Draw
# Define the SMARTS pattern for the substructure (example: a pyridine ring)
substructure_smarts = "c1ccncc1"
substructure_pattern = Chem.MolFromSmarts(substructure_smarts)

# Find compounds containing the substructure
def contains_substructure(smiles, pattern):
    try:
        mol = Chem.MolFromSmiles(smiles)
        if mol:
            return mol.HasSubstructMatch(pattern)
        else:
            return False
    except:
        return False

df['Has_Substructure'] = df['smiles'].apply(lambda x: contains_substructure(x,
substructure_pattern))
print(df['Has_Substructure'].value_counts())
```

```
# Visualize a few compounds containing the substructure
subset = df[df['Has_Substructure'] == True].head(5)
mols = [Chem.MolFromSmiles(s) for s in subset['smiles']]
img = Draw.MolsToGridImage(mols,
highlightAtomLists=[mols[i].GetSubstructMatches(substructure_pattern) for i in
range(len(mols))], molsPerRow=5)
img  # Display the image in the notebook
```

Important Considerations:

- **Error Handling:** The code includes basic error handling (e.g., checking for file existence, handling potential errors during SMILES parsing). You should add more robust error handling as needed.
- **Data Validation:** Always validate your data to ensure its quality. Check for missing values, outliers, and inconsistencies.
- **Statistical Significance:** Be mindful of statistical significance when interpreting results, especially with a small dataset (100 rows).
- **Model Selection:** Linear Regression is a simple starting point. Explore other machine learning models (e.g., Random Forest, Support Vector Machines) to see if you can improve prediction accuracy.
- **Feature Engineering:** Experiment with different molecular descriptors and feature engineering techniques to improve model performance.
- **Cross-Validation:** Use cross-validation to get a more reliable estimate of model performance.
- **Project Structure:** Organize your code into reusable modules (in the `src/` directory) to improve maintainability.

In Summary:

This comprehensive guide provides you with the SQL and Python code, addresses the errors you encountered, sets up your project structure, and provides examples to get you started with analyzing ChEMBL 35 data for EGFR kinase inhibitors. Remember to adapt the code and analyses to your specific research questions and data characteristics. Good luck!

中文翻译:

好的，我明白了。我将帮助您使用 AIMLops 模板设置 ChEMBL 35 数据分析项目，解决您遇到的错误，并为您提供“Topic_CheMBL_35_15”主题的示例代码。

分析模型:

目标是利用 ChEMBL 35 数据，通过 `rdkit` 通过 PostgreSQL 访问，为药物研究和开发做出贡献。您的工作流程包括:

1. 数据提取 **(SQL)**：使用 SQL 通过 pgAdmin 查询 ChEMBL 35 数据库以检索相关数据，并将其另存为 `../data/` 目录中的 CSV 文件。这允许您直接在数据库级别预处理和过滤数据，从而提高效率。
2. 数据分析 **(Python/Jupyter Notebook)**：将提取的 CSV 数据加载到 Jupyter Notebook 环境中。`rdkit` 库将用于处理化学结构，您可能会执行以下分析:
    - 结构-活性关系 **(SAR)** 分析：将化学结构特征（例如，分子描述符、指纹）与活性值（例如，IC50、Ki）相关联。
    - 数据可视化：创建绘图和图表以探索化学特征和活性之间的关系。
    - 模型构建：开发预测模型以根据化学结构估算新化合物的活性。

解决错误:

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

  此错误表明您正在尝试对数字列 (`act.standard_value`) 使用正则表达式运算符 (`~`)。 PostgreSQL 可能不允许直接对数字类型进行正则表达式匹配。 您需要在应用正则表达式之前将数字列转换为文本。

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

  这意味着您正在使用旧版本的 scikit-learn。 `squared=False` 参数是在更高版本中添加到 `mean_squared_error` 的。 您有两种解决方案：1) 升级您的 scikit-learn 安装（推荐），或 2) 删除 `squared=False` 参数并手动获取结果的平方根以获得均方根误差 (RMSE)。

文件夹结构（基于 **AIMLops** 模板）：

假设一个典型的 AIMLops 结构，这是文件夹结构的样子：

```
my_chembl_project/
├── data/            # 包含从数据库中提取的 CSV 文件。 例如，activity_data.csv
├── notebooks/       # 包含您的 Jupyter Notebook。 例如，
Topic_CheMBL_35_15_1_data_exploration.ipynb
├── models/          #（可选）存储保存的模型文件（例如，pickle 的 scikit-learn 模型）。
├── src/             #（可选）包含可重用代码的 Python 模块。
├── reports/         #（可选）生成的报告和图形。
└── README.md
```

主题：**`Topic_CheMBL_35_15`**（假设此主题侧重于特定目标或测定）

为了演示，让我们假设 `Topic_CheMBL_35_15` 侧重于 **“EGFR** 激酶的抑制剂”。

SQL 代码（用于提取数据）：

```sql
-- File: ../data/egfr_inhibitors.csv

SELECT
    cmp.chembl_id AS compound_chembl_id,
    cmp.pref_name AS compound_name,
    act.standard_type,
    act.standard_value,
    act.standard_units,
    act.activity_comment,
    mol.molfile AS molfile -- 包含molfile
FROM
    activities act
JOIN
    molecule_dictionary cmp ON act.molregno = cmp.molregno
JOIN
    assays ass ON act.assay_id = ass.assay_id
JOIN
    target_dictionary tgt ON ass.tid = tgt.tid
LEFT JOIN
    mols mol ON cmp.molregno = mol.molregno
WHERE
    tgt.pref_name = 'Epidermal Growth Factor Receptor' -- EGFR 激酶
    AND act.standard_type IN ('IC50', 'Ki', 'Kd')  -- 常见活性类型
    AND act.standard_units = 'nM'
    AND act.standard_value IS NOT NULL
    AND act.standard_value::text ~ '^[0-9\.]+$'   -- 更正的行：转换为文本以进行正则表达式
ORDER BY
```

```
    act.standard_value ASC
LIMIT 100;  -- 限制为100 行
```

说明：

- **cmp.chembl_id, cmp.pref_name**：选择化合物的 ChEMBL ID 和首选名称。
- **act.standard_type, act.standard_value, act.standard_units**：选择活性类型（例如，IC50）、值和单位。
- **tgt.pref_name = 'Epidermal Growth Factor Receptor'**：过滤针对 EGFR 的活性。
- **act.standard_type IN ('IC50', 'Ki', 'Kd')**：过滤常见活性类型。
- **act.standard_units = 'nM'**：确保活性值以 nM 为单位。
- **act.standard_value IS NOT NULL**：排除缺少标准值的行。
- **act.standard_value::text ~ '^[0-9\.]+$'**：重要更正：此行过滤 standard_value 以确保它仅包含正数数值（整数或小数）。它*还*通过在应用正则表达式之前显式地将 act.standard_value 转换为 text 来解决 PostgreSQL 错误。这允许正则表达式运算符正常工作。
- **LIMIT 100**：将返回的行数限制为 100。这对于开发期间的性能很重要。
- **mol.molfile AS molfile**: 在数据中包含 molfile

如何运行：

1. 打开 pgAdmin。
2. 连接到您的 PostgreSQL 服务器（IP：192.168.206.136，用户：rd，密码：rd，数据库：chembl_35）。
3. 打开一个新的查询窗口。
4. 将 SQL 代码粘贴到查询窗口中。
5. 执行查询。
6. 将结果另存为 CSV 文件，命名为 egfr_inhibitors.csv，位于项目的 ../data/ 目录中。（确保在保存时选择 CSV 格式）。

Python 代码（Jupyter Notebook：notebooks/Topic_CheMBL_35_15_1_data_exploration.ipynb）：

```
import os
import pandas as pd
from rdkit import Chem
from rdkit.Chem import Descriptors
from rdkit.Chem import Lipinski
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
import numpy as np

# 定义项目的基础路径
base_path = os.path.abspath(os.path.join(os.getcwd(), ".."))  # 假设notebook 位于 notebooks/ 中

# 构建数据路径
data_path = os.path.join(base_path, "data", "egfr_inhibitors.csv")

# 加载数据
try:
    df = pd.read_csv(data_path)
    print("数据从以下位置成功加载：", data_path)
except FileNotFoundError:
```

```
    print(f"错误：在 {data_path} 找不到文件。请确保您运行了 SQL 查询并保存了 CSV 文件。")
    exit()

# 数据清洗和预处理
# 将 standard_value 转换为数值，处理潜在的错误
df['standard_value'] = pd.to_numeric(df['standard_value'], errors='coerce')
df = df.dropna(subset=['standard_value'])  # 删除具有无效 standard_value 的行

# 将 IC50/Ki/Kd 转换为 pIC50（或 pKi/pKd）
def convert_to_pActivity(standard_value):
  return -np.log10(standard_value/10**9)

df['pActivity'] = df['standard_value'].apply(convert_to_pActivity)


# RDKit 特征计算
def calculate_descriptors(mol):
    try:
        mol = Chem.MolFromSmiles(mol)
        if mol is None:
            return None
        descriptors = {
            'MW': Descriptors.MolWt(mol),
            'LogP': Descriptors.MolLogP(mol),
            'HBD': Lipinski.NumHDonors(mol),
            'HBA': Lipinski.NumHAcceptors(mol),
            'TPSA': Descriptors.TPSA(mol)
        }
        return descriptors
    except Exception as e:
        print(f"计算描述符时出错：{e}")
        return None

# 从 molfile 创建 smiles
def molfile_to_smiles(molfile):
    try:
        mol = Chem.MolFromMolBlock(molfile)
        if mol:
            smiles = Chem.MolToSmiles(mol)
            return smiles
        else:
            return None
    except Exception as e:
        print(f"将 molfile 转换为 SMILES 时出错：{e}")
        return None

df['smiles'] = df['molfile'].apply(molfile_to_smiles)
df = df.dropna(subset=['smiles'])  # 删除具有无效 smiles 的行

#应用描述符计算
df['descriptors'] = df['smiles'].apply(calculate_descriptors)

# 将字典转换为 DataFrame 中的单个列
df = pd.concat([df, df['descriptors'].apply(pd.Series)], axis=1)

# 删除描述符计算失败的行
df = df.dropna(subset=['MW', 'LogP', 'HBD', 'HBA', 'TPSA'])

# 简单的线性回归模型
```

```
X = df[['MW', 'LogP', 'HBD', 'HBA', 'TPSA']]
y = df['pActivity']

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

model = LinearRegression()
model.fit(X_train, y_train)

y_pred = model.predict(X_test)

mse = mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse)  # 手动计算 RMSE
r2 = r2_score(y_test, y_pred)

print(f"均方误差: {mse}")
print(f"均方根误差: {rmse}")
print(f"R 平方: {r2}")

# 示例可视化
plt.scatter(y_test, y_pred)
plt.xlabel("实际 pActivity")
plt.ylabel("预测 pActivity")
plt.title("实际与预测 pActivity")
plt.show()
```

说明:

- 导入库: 导入必要的库。
- 定义路径: 使用 `os.path.join` 定义基础路径和数据路径以实现平台独立性。
- 加载数据: 将 CSV 数据加载到 Pandas DataFrame 中。 包括文件未找到的错误处理。
- 数据清洗:
  - 将 `standard_value` 转换为数值并处理错误，删除具有非数值的行。
  - 将标准值（IC50、Ki、Kd）转换为 pActivity（例如，pIC50）。 这是 SAR 分析中的常见做法。
- **RDKit** 特征计算:
  - `calculate_descriptors` 函数使用 RDKit 计算分子量 (MW)、LogP、氢键供体 (HBD)、氢键受体 (HBA) 和拓扑极性表面积 (TPSA)。 它处理 SMILES 解析期间的潜在错误。
- 从 **molfile** 获取 **SMILES**:
  - `molfile_to_smiles` 函数使用 RDKit 将 molfile 转换为 SMILES。
- 模型构建（线性回归）:
  - 选择特征（MW、LogP、HBD、HBA、TPSA）和目标变量（pActivity）。
  - 将数据拆分为训练集和测试集。
  - 创建一个线性回归模型，在训练数据上对其进行训练，并对测试数据进行预测。
  - 计算均方误差 (MSE)、均方根误差 (RMSE) 和 R 平方 (R2) 以评估模型的性能。
- 可视化: 创建实际与预测 pActivity 值的散点图。

如何运行:

1. 打开 Jupyter Notebook (`notebooks/Topic_CheMBL_35_15_1_data_exploration.ipynb`)。
2. 依次运行 notebook 中的每个单元格。
3. 检查输出，包括模型性能指标和可视化。

5 个例子

以下是您可以使用此数据执行的 5 个示例分析：

1. 相关性分析：计算分子描述符（MW、LogP、HBD、HBA、TPSA）和 pActivity 之间的相关性矩阵，以识别与活性最密切相关的描述符。使用热图可视化相关性矩阵。

```
correlation_matrix = df[['pActivity', 'MW', 'LogP', 'HBD', 'HBA',
'TPSA']].corr()
sns.heatmap(correlation_matrix, annot=True, cmap="coolwarm")
plt.title("相关性矩阵")
plt.show()
```

2. 分布图：为分子描述符和 pActivity 创建分布图（直方图或核密度估计），以了解它们的分布。

```
sns.histplot(df['pActivity'], kde=True)
plt.title("pActivity 的分布")
plt.show()

sns.histplot(df['LogP'], kde=True)
plt.title("LogP 的分布")
plt.show()
```

3. 带有回归线的散点图：创建每个分子描述符与 pActivity 之间的散点图，并覆盖回归线以可视化线性关系。

```
sns.regplot(x='LogP', y='pActivity', data=df)
plt.title("LogP vs. pActivity")
plt.show()
```

4. **Lipinski** 五规则分析：评估数据集中有多少化合物违反了 Lipinski 五规则（MW <= 500、LogP <= 5、HBD <= 5、HBA <= 10）。

```
def lipinski_violations(row):
    violations = 0
    if row['MW'] > 500:
        violations += 1
    if row['LogP'] > 5:
        violations += 1
    if row['HBD'] > 5:
        violations += 1
    if row['HBA'] > 10:
        violations += 1
    return violations

df['Lipinski_Violations'] = df.apply(lipinski_violations, axis=1)
print(df['Lipinski_Violations'].value_counts())
```

5. 子结构搜索：使用 RDKit 搜索包含特定子结构（例如，常见药效团）的化合物。

```
from rdkit.Chem.Draw import IPythonConsole
from rdkit.Chem import Draw
# 定义子结构的SMARTS 模式（示例：吡啶环）
substructure_smarts = "c1ccncc1"
substructure_pattern = Chem.MolFromSmarts(substructure_smarts)

# 查找包含子结构的化合物
def contains_substructure(smiles, pattern):
    try:
        mol = Chem.MolFromSmiles(smiles)
        if mol:
            return mol.HasSubstructMatch(pattern)
```

```
            else:
                return False
        except:
            return False

    df['Has_Substructure'] = df['smiles'].apply(lambda x: contains_substructure(x,
    substructure_pattern))
    print(df['Has_Substructure'].value_counts())

    # 可视化一些包含子结构的化合物
    subset = df[df['Has_Substructure'] == True].head(5)
    mols = [Chem.MolFromSmiles(s) for s in subset['smiles']]
    img = Draw.MolsToGridImage(mols,
    highlightAtomLists=[mols[i].GetSubstructMatches(substructure_pattern) for i in
    range(len(mols))], molsPerRow=5)
    img  # 在notebook 中显示图像
```

重要注意事项：

- 错误处理：该代码包括基本的错误处理（例如，检查文件是否存在，处理 SMILES 解析期间的潜在错误）。您应该根据需要添加更强大的错误处理。
- 数据验证：始终验证您的数据以确保其质量。检查缺失值、异常值和不一致之处。
- 统计显着性：在解释结果时，请注意统计显着性，尤其是在小型数据集（100 行）中。
- **模型