PharmApp Suite

🧠 AI for Drug Discovery and Development 🍼

🧩 Topic: AutoML pipelines for ChEMBL data

Okay, I understand. I'll help you structure your Chembl 35 data analysis project, address the errors you've encountered, and provide example code.

1. Analysis Model & Workflow

Your project aims to analyze ChEMBL 35 data using RDKit and scikit-learn. Here's a breakdown of a typical workflow and analysis models you might consider:

- **Data Extraction & Preparation:**
  - Connect to the ChEMBL 35 database (PostgreSQL using `psycopg2` in Python).
  - Extract relevant data: bioactivities, compounds (SMILES strings), targets, etc. Your SQL query needs to be tailored to your specific research question.
  - Clean and filter the data (removing duplicates, invalid SMILES, etc.).
  - Convert SMILES strings to RDKit Mol objects.
- **Feature Generation:**
  - Use RDKit to generate molecular descriptors (physicochemical properties, fingerprints, etc.). Common descriptors include:
    - Morgan fingerprints (ECFP/FCFP)
    - Physicochemical properties (molecular weight, logP, H-bond donors/acceptors, etc.)
    - Other descriptors like topological polar surface area (TPSA), number of rotatable bonds, etc.
- **Model Building & Evaluation:**
  - **Regression Models:** If you're predicting a continuous variable (e.g., IC50, Ki):
    - Linear Regression
    - Ridge Regression
    - Lasso Regression
    - Random Forest Regression
    - Support Vector Regression (SVR)
  - **Classification Models:** If you're predicting a categorical variable (e.g., active/inactive):
    - Logistic Regression
    - Random Forest Classification
    - Support Vector Classification (SVC)
    - Naive Bayes
  - **Clustering:** If you want to group similar compounds:
    - K-Means Clustering
    - Hierarchical Clustering
    - DBSCAN
- **Model Evaluation:**
  - Split your data into training and test sets (e.g., 80/20 split).
  - Train your model on the training set.
  - Evaluate your model on the test set using appropriate metrics:

- **Regression:** Mean Squared Error (MSE), Root Mean Squared Error (RMSE), R-squared.
- **Classification:** Accuracy, Precision, Recall, F1-score, AUC-ROC.

2. 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're trying to use a regular expression operator (~) on a `numeric` column (`act.standard_value`). Regular expressions are for text, not numbers. To fix this, you should use numeric comparisons. If you want to filter values to be positive numbers only, use `act.standard_value > 0`.

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

  This indicates you are using an older version of scikit-learn. Either upgrade your scikit-learn version (`pip install -U scikit-learn`) or remove the `squared=False` argument (which will return MSE instead of RMSE). RMSE is just `sqrt(MSE)`.

3. SQL and Python Code (with examples)

SQL (Save to `../data/Topic_CheMBL_35_64.csv`)

```sql
-- Topic_CheMBL_35_64.sql
-- Get 100 rows of activity data with standard values, target information, and compound SMILES

SELECT
    act.activity_id,
    act.standard_value,
    act.standard_units,
    act.standard_type,
    cmp.canonical_smiles,
    td.pref_name AS target_name
FROM activities act
JOIN assays ass ON act.assay_id = ass.assay_id
JOIN target_dictionary td ON ass.tid = td.tid
JOIN molecule_dictionary md ON act.molregno = md.molregno
JOIN compound_structures cmp ON md.molregno = cmp.molregno
WHERE act.standard_type = 'IC50'  -- Filter for IC50 values
  AND act.standard_units = 'nM'    -- Filter for nM units
  AND act.standard_value > 0       -- standard_value is a positive number
LIMIT 100;  -- Limit to 100 rows
```

Explanation:

- **`SELECT ... FROM ...`**: Selects the columns you need from different tables.
- **`JOIN ... ON ...`**: Connects tables based on related columns. This is crucial to link activities to compounds, targets, and assays.
- **`WHERE ...`**: Filters the data based on specific criteria:
  - `act.standard_type = 'IC50'` : Selects only IC50 activity values.
  - `act.standard_units = 'nM'` : Selects only activities reported in nanomolar (nM).
  - `act.standard_value > 0` : Ensures only activities with values greater than zero are included. This addresses the previous error of attempting regex on numeric data.
- **`LIMIT 100`**: Limits the result set to 100 rows. This is important to keep the data manageable, as you requested.

Python (Jupyter Notebook: `notebook/Topic_CheMBL_35_64_1_Data_Prep.ipynb`)

```python
# notebook/Topic_CheMBL_35_64_1_Data_Prep.ipynb

import os
import pandas as pd
from rdkit import Chem
from rdkit.Chem import AllChem
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 math

# Define the base path for your project
base_path = os.path.abspath(os.path.join(os.getcwd(), ".."))  # Go up one level from 'notebooks'
data_path = os.path.join(base_path, "data", "Topic_CheMBL_35_64.csv")

# 1. Load the data
try:
    df = pd.read_csv(data_path)
    print("Data loaded successfully.")
except FileNotFoundError:
    print(f"Error: File not found at {data_path}")
    exit()

# 2. Data Cleaning and Preparation
print("\nData Cleaning and Preparation...")
df.dropna(subset=['canonical_smiles', 'standard_value'], inplace=True) #drop rows with NaN value
df = df[df['standard_value'] > 0] # Remove non-positive activity values
df = df.drop_duplicates(subset=['canonical_smiles', 'standard_value'])  # Remove duplicate molecules with same activity

# 3. RDKit Mol Object Creation
print("\nCreating RDKit Mol objects...")
df['mol'] = df['canonical_smiles'].apply(lambda x: Chem.MolFromSmiles(x))
df = df.dropna(subset=['mol']) #remove invalid Smiles

# 4. Feature Generation (Example: Molecular Weight)
print("\nGenerating Molecular Weight feature...")
df['mol_weight'] = df['mol'].apply(Descriptors.MolWt)

# 5. Feature Generation (Example: Morgan Fingerprints)
print("\nGenerating Morgan Fingerprints...")
def generate_morgan_fingerprint(mol, radius=2, nBits=2048):
    try:
        return AllChem.GetMorganFingerprintAsBitVect(mol, radius, nBits=nBits)
    except:
        return None  # Handle cases where fingerprint generation fails

df['morgan_fp'] = df['mol'].apply(generate_morgan_fingerprint)
df = df.dropna(subset=['morgan_fp'])

def fp_to_numpy(fp):
    arr = np.zeros((1,), dtype=np.int32)
    AllChem.DataStructs.ConvertToNumpyArray(fp, arr)
    return arr
```

```
df['morgan_fp_array'] = df['morgan_fp'].apply(fp_to_numpy)


# 6. Prepare data for modeling (example: using molecular weight as feature, IC50 as
target)
print("\nPreparing data for modeling...")

X = df[['mol_weight']]
y = df['standard_value']

# 7. Train-Test Split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# 8. Model Training
print("\nTraining Linear Regression model...")
model = LinearRegression()
model.fit(X_train, y_train)

# 9. Model Evaluation
print("\nEvaluating the model...")
y_pred = model.predict(X_test)

mse = mean_squared_error(y_test, y_pred)
rmse = math.sqrt(mse)
r2 = r2_score(y_test, y_pred) # R-squared

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


print("\nDone!")
```

Explanation:

1. **Import Libraries:** Import necessary libraries.
2. **Define Paths:** Constructs the path to the CSV file using `os.path.join` and `base_path` to ensure portability.
3. **Load Data:** Loads the CSV data into a Pandas DataFrame using `pd.read_csv`. Includes error handling if the file is not found.
4. **Data Cleaning:**
    - Handles missing values using `dropna`.
    - Removes non-positive activity values based on the standard value column.
    - Removes duplicate entries based on SMILES and standard value.
5. **RDKit Mol Objects:** Creates RDKit Mol objects from the SMILES strings using `Chem.MolFromSmiles`. Handles potential errors by removing rows where the SMILES string is invalid.
6. **Feature Generation:**
    - Calculates molecular weight using `Descriptors.MolWt`.
    - Calculates Morgan fingerprints. Includes error handling. Converts RDKit fingerprints to NumPy arrays to be compatible with scikit-learn.
7. **Data Preparation for Modeling:** Selects 'mol_weight' as the feature and 'standard_value' as the target variable.
8. **Train-Test Split:** Splits the data into training and testing sets.

9. **Model Training:** Trains a Linear Regression model.
10. **Model Evaluation:** Evaluates the model using Mean Squared Error (MSE), Root Mean Squared Error (RMSE), and R-squared.

Example Usage:

1. **Run the SQL script** in pgAdmin to create the `Topic_CheMBL_35_64.csv` file in the `data` directory.
2. **Open the `Topic_CheMBL_35_64_1_Data_Prep.ipynb`** notebook in Jupyter.
3. **Run all the cells** in the notebook. The output will show the data loading, cleaning, feature generation, model training, and evaluation results.

Example 2: Using Morgan Fingerprints as Features for Regression

```
# notebook/Topic_CheMBL_35_64_2_Morgan_Regression.ipynb
import os
import pandas as pd
from rdkit import Chem
from rdkit.Chem import AllChem
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 math

# Define the base path for your project
base_path = os.path.abspath(os.path.join(os.getcwd(), ".."))  # Go up one level from
'notebooks'
data_path = os.path.join(base_path, "data", "Topic_CheMBL_35_64.csv")

# 1. Load the data
try:
    df = pd.read_csv(data_path)
    print("Data loaded successfully.")
except FileNotFoundError:
    print(f"Error: File not found at {data_path}")
    exit()

# 2. Data Cleaning and Preparation
print("\nData Cleaning and Preparation...")
df.dropna(subset=['canonical_smiles', 'standard_value'], inplace=True) #drop rows with
NaN value
df = df[df['standard_value'] > 0] # Remove non-positive activity values
df = df.drop_duplicates(subset=['canonical_smiles', 'standard_value'])  # Remove
duplicate molecules with same activity

# 3. RDKit Mol Object Creation
print("\nCreating RDKit Mol objects...")
df['mol'] = df['canonical_smiles'].apply(lambda x: Chem.MolFromSmiles(x))
df = df.dropna(subset=['mol']) #remove invalid Smiles

# 4. Feature Generation (Example: Morgan Fingerprints)
print("\nGenerating Morgan Fingerprints...")
def generate_morgan_fingerprint(mol, radius=2, nBits=2048):
    try:
        return AllChem.GetMorganFingerprintAsBitVect(mol, radius, nBits=nBits)
    except:
        return None  # Handle cases where fingerprint generation fails
```

```
df['morgan_fp'] = df['mol'].apply(generate_morgan_fingerprint)
df = df.dropna(subset=['morgan_fp'])

def fp_to_numpy(fp):
    arr = np.zeros((1,), dtype=np.int32)
    AllChem.DataStructs.ConvertToNumpyArray(fp, arr)
    return arr

df['morgan_fp_array'] = df['morgan_fp'].apply(fp_to_numpy)

# Prepare data for modeling (example: using Morgan fingerprint as feature, IC50 as
target)
print("\nPreparing data for modeling...")

# Stack the fingerprint arrays into a single NumPy array
X = np.stack(df['morgan_fp_array'].values)
y = df['standard_value'].values

# Train-Test Split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Model Training
print("\nTraining Linear Regression model...")
model = LinearRegression()
model.fit(X_train, y_train)

# Model Evaluation
print("\nEvaluating the model...")
y_pred = model.predict(X_test)

mse = mean_squared_error(y_test, y_pred)
rmse = math.sqrt(mse)
r2 = r2_score(y_test, y_pred) # R-squared

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

print("\nDone!")
```

Explanation: This example performs regression using Morgan fingerprints as features. It extracts the fingerprint arrays, stacks them into a NumPy array, and uses that as the input to the linear regression model. This is more representative of typical QSAR/QSPR modeling.

Example 3: Using Morgan Fingerprints for Classification (Activity Prediction)

```
# notebook/Topic_CheMBL_35_64_3_Morgan_Classification.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 LogisticRegression
from sklearn.metrics import accuracy_score, classification_report

# Define the base path for your project
base_path = os.path.abspath(os.path.join(os.getcwd(), ".."))  # Go up one level from
'notebooks'
```

```
data_path = os.path.join(base_path, "data", "Topic_CheMBL_35_64.csv")

# 1. Load the data
try:
    df = pd.read_csv(data_path)
    print("Data loaded successfully.")
except FileNotFoundError:
    print(f"Error: File not found at {data_path}")
    exit()

# 2. Data Cleaning and Preparation
print("\nData Cleaning and Preparation...")
df.dropna(subset=['canonical_smiles', 'standard_value'], inplace=True) #drop rows with
NaN value
df = df[df['standard_value'] > 0] # Remove non-positive activity values
df = df.drop_duplicates(subset=['canonical_smiles', 'standard_value'])  # Remove
duplicate molecules with same activity

# 3. RDKit Mol Object Creation
print("\nCreating RDKit Mol objects...")
df['mol'] = df['canonical_smiles'].apply(lambda x: Chem.MolFromSmiles(x))
df = df.dropna(subset=['mol']) #remove invalid Smiles

# 4. Feature Generation (Example: Morgan Fingerprints)
print("\nGenerating Morgan Fingerprints...")
def generate_morgan_fingerprint(mol, radius=2, nBits=2048):
    try:
        return AllChem.GetMorganFingerprintAsBitVect(mol, radius, nBits=nBits)
    except:
        return None  # Handle cases where fingerprint generation fails

df['morgan_fp'] = df['mol'].apply(generate_morgan_fingerprint)
df = df.dropna(subset=['morgan_fp'])

def fp_to_numpy(fp):
    arr = np.zeros((1,), dtype=np.int32)
    AllChem.DataStructs.ConvertToNumpyArray(fp, arr)
    return arr

df['morgan_fp_array'] = df['morgan_fp'].apply(fp_to_numpy)

# 5. Define Activity Threshold (e.g., IC50 < 1000 nM is active)
activity_threshold = 1000  # nM

# 6. Create Activity Labels
df['active'] = (df['standard_value'] <= activity_threshold).astype(int)

# 7. Prepare data for modeling (example: using Morgan fingerprint as feature, activity
label as target)
print("\nPreparing data for modeling...")
X = np.stack(df['morgan_fp_array'].values)
y = df['active'].values

# 8. Train-Test Split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# 9. Model Training
print("\nTraining Logistic Regression model...")
```

```
model = LogisticRegression(max_iter=1000)  # Increase max_iter if it doesn't converge
model.fit(X_train, y_train)

# 10. Model Evaluation
print("\nEvaluating the model...")
y_pred = model.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
report = classification_report(y_test, y_pred)

print(f"Accuracy: {accuracy}")
print(f"Classification Report:\n{report}")

print("\nDone!")
```

Explanation: This example demonstrates a binary classification task. It defines an activity threshold based on IC50 values (e.g., IC50 < 1000 nM is considered active). It then trains a Logistic Regression model to predict activity based on Morgan fingerprints.

Example 4: Calculation of QED (Drug-Likeness)

```
# notebook/Topic_CheMBL_35_64_4_QED_Calculation.ipynb

import os
import pandas as pd
from rdkit import Chem
from rdkit.Chem import QED

# Define the base path for your project
base_path = os.path.abspath(os.path.join(os.getcwd(), ".."))  # Go up one level from
'notebooks'
data_path = os.path.join(base_path, "data", "Topic_CheMBL_35_64.csv")

# 1. Load the data
try:
    df = pd.read_csv(data_path)
    print("Data loaded successfully.")
except FileNotFoundError:
    print(f"Error: File not found at {data_path}")
    exit()

# 2. Data Cleaning and Preparation
print("\nData Cleaning and Preparation...")
df.dropna(subset=['canonical_smiles', 'standard_value'], inplace=True) #drop rows with
NaN value
df = df[df['standard_value'] > 0] # Remove non-positive activity values
df = df.drop_duplicates(subset=['canonical_smiles', 'standard_value'])  # Remove
duplicate molecules with same activity

# 3. RDKit Mol Object Creation
print("\nCreating RDKit Mol objects...")
df['mol'] = df['canonical_smiles'].apply(lambda x: Chem.MolFromSmiles(x))
df = df.dropna(subset=['mol']) #remove invalid Smiles

# 4. Calculate QED
print("\nCalculating QED...")
df['QED'] = df['mol'].apply(QED.qed)

# 5. Display some results
print("\nSample of QED values:")
```

```
print(df[['canonical_smiles', 'QED']].head())

# 6. Basic Statistics
print("\nQED Statistics:")
print(df['QED'].describe())

print("\nDone!")
```

Explanation: This notebook calculates the Quantitative Estimate of Drug-likeness (QED) using the RDKit. QED is a metric that reflects the overall drug-likeness of a molecule based on a combination of properties.

Example 5: Using a different Machine learning model, Random Forest Regression

```
# notebook/Topic_CheMBL_35_64_5_RandomForest_Regression.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.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, r2_score
import math

# Define the base path for your project
base_path = os.path.abspath(os.path.join(os.getcwd(), ".."))  # Go up one level from
'notebooks'
data_path = os.path.join(base_path, "data", "Topic_CheMBL_35_64.csv")

# 1. Load the data
try:
    df = pd.read_csv(data_path)
    print("Data loaded successfully.")
except FileNotFoundError:
    print(f"Error: File not found at {data_path}")
    exit()

# 2. Data Cleaning and Preparation
print("\nData Cleaning and Preparation...")
df.dropna(subset=['canonical_smiles', 'standard_value'], inplace=True) #drop rows with
NaN value
df = df[df['standard_value'] > 0] # Remove non-positive activity values
df = df.drop_duplicates(subset=['canonical_smiles', 'standard_value'])  # Remove
duplicate molecules with same activity

# 3. RDKit Mol Object Creation
print("\nCreating RDKit Mol objects...")
df['mol'] = df['canonical_smiles'].apply(lambda x: Chem.MolFromSmiles(x))
df = df.dropna(subset=['mol']) #remove invalid Smiles

# 4. Feature Generation (Example: Morgan Fingerprints)
print("\nGenerating Morgan Fingerprints...")
def generate_morgan_fingerprint(mol, radius=2, nBits=2048):
    try:
        return AllChem.GetMorganFingerprintAsBitVect(mol, radius, nBits=nBits)
    except:
        return None  # Handle cases where fingerprint generation fails

df['morgan_fp'] = df['mol'].apply(generate_morgan_fingerprint)
```

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

def fp_to_numpy(fp):
    arr = np.zeros((1,), dtype=np.int32)
    AllChem.DataStructs.ConvertToNumpyArray(fp, arr)
    return arr

df['morgan_fp_array'] = df['morgan_fp'].apply(fp_to_numpy)

# Prepare data for modeling (example: using Morgan fingerprint as feature, IC50 as
target)
print("\nPreparing data for modeling...")
X = np.stack(df['morgan_fp_array'].values)
y = df['standard_value'].values

# Train-Test Split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Model Training
print("\nTraining Random Forest Regression model...")
model = RandomForestRegressor(n_estimators=100, random_state=42)  # You can adjust
hyperparameters like n_estimators
model.fit(X_train, y_train)

# Model Evaluation
print("\nEvaluating the model...")
y_pred = model.predict(X_test)

mse = mean_squared_error(y_test, y_pred)
rmse = math.sqrt(mse)
r2 = r2_score(y_test, y_pred)  # R-squared

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

print("\nDone!")
```

Explanation: This example uses a Random Forest Regressor, a more complex model than linear regression, which can often capture non-linear relationships in the data.

Important Considerations:

- **Hyperparameter Tuning:** For more advanced modeling, explore hyperparameter tuning techniques (e.g., GridSearchCV, RandomizedSearchCV) to optimize your model's performance.
- **Feature Selection:** Consider feature selection methods to identify the most relevant descriptors for your model.
- **Cross-Validation:** Use cross-validation (e.g., k-fold cross-validation) for more robust model evaluation.
- **Data Scaling/Normalization:** Scaling or normalizing your features (especially when using models like SVR or Linear Regression) can improve performance.
- **Error Handling:** Add more robust error handling to your code (e.g., try-except blocks) to gracefully handle potential issues during data processing and model training.

Language translation

1. 分析模型和工作流程

您的项目旨在分析 ChEMBL 35 数据，使用 RDKit 和 scikit-learn。以下是一个典型的工作流程和您可以考虑的分析模型：

- 数据提取与准备：
  - 连接到 ChEMBL 35 数据库（PostgreSQL，使用 Python 中的 psycopg2）。
  - 提取相关数据：生物活性、化合物（SMILES 字符串）、靶标等。您的 SQL 查询需要根据您的具体研究问题进行定制。
  - 清理和过滤数据（删除重复项、无效的 SMILES 等）。
  - 将 SMILES 字符串转换为 RDKit Mol 对象。
- 特征生成：
  - 使用 RDKit 生成分子描述符（理化性质、指纹图谱等）。常见的描述符包括：
    - Morgan 指纹图谱（ECFP/FCFP）
    - 理化性质（分子量、logP、氢键供体/受体等）
    - 其他描述符，如拓扑极性表面积 (TPSA)、可旋转键数等。
- 模型构建与评估：
  - 回归模型：如果您要预测一个连续变量（例如，IC50、Ki）：
    - 线性回归
    - 岭回归
    - Lasso 回归
    - 随机森林回归
    - 支持向量回归 (SVR)
  - 分类模型：如果您要预测一个分类变量（例如，活性/非活性）：
    - 逻辑回归
    - 随机森林分类
    - 支持向量分类 (SVC)
    - 朴素贝叶斯
  - 聚类：如果您想对相似的化合物进行分组：
    - K-Means 聚类
    - 层次聚类
    - DBSCAN
- 模型评估：
  - 将您的数据分成训练集和测试集（例如，80/20 分割）。
  - 在训练集上训练您的模型。
  - 使用适当的指标在测试集上评估您的模型：
    - 回归：均方误差 (MSE)、均方根误差 (RMSE)、R 平方。
    - 分类：准确率、精确率、召回率、F1 分数、AUC-ROC。

2. 解决错误

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

  这个错误表明您正在尝试对 numeric 列 (act.standard_value) 使用正则表达式运算符 (~)。正则表达式用于文本，而不是数字。要解决这个问题，您应该使用数值比较。如果您只想过滤值为正数的数值，请使用 act.standard_value > 0。

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

  这表明您正在使用旧版本的 scikit-learn。升级您的 scikit-learn 版本 (pip install -U scikit-learn) 或删除 squared=False 参数（这将返回 MSE 而不是 RMSE）。RMSE 只是 sqrt(MSE)。

3. SQL 和 **Python** 代码 **(**含示例**)**

请参考上面的英文代码和解释.

I hope this helps! Let me know if you have any more questions.