PharmApp Suite

🧠 AI for Drug Discovery and Development 🍼

🧩 Topic: Multi-task learning for pan-target inhibition

Okay, I understand. As a Pharmaceutical Research and Development Specialist, I can help you set up your Chembl 35 data analysis project using RDKit, following the AIMLops template, and addressing the specific errors you encountered.

Here's a breakdown of the process, incorporating SQL, Python code snippets, explanations, and example usage, along with addressing the identified errors.

1. Project Setup and Understanding the Analysis Model

You're using ChEMBL 35 data to explore drug properties and relationships. The aim is likely to perform exploratory data analysis (EDA), feature engineering, and possibly build predictive models based on chemical structure and activity data. The key components are:

- **Database:** ChEMBL 35 hosted on PostgreSQL (psql).
- **Data Extraction:** SQL queries to retrieve specific data from the database.
- **Data Processing:** RDKit for handling molecular structures, feature generation, and data manipulation.
- **Modeling:** Scikit-learn for potential machine learning tasks (regression, classification, etc.).
- **AIMLops Structure:** Organizing your project according to AIMLops best practices (data, notebooks, models, etc.).

Analysis Model Considerations:

1. **Data Exploration:** Understand the distributions of key variables like `standard_value`, `standard_type`, `pCHEMBL_value`, and the molecular properties. Look for missing data and outliers.
2. **Feature Engineering:** RDKit allows you to generate a vast array of molecular descriptors (e.g., topological polar surface area (TPSA), number of hydrogen bond donors/acceptors, molecular weight). These descriptors can be used as features in your models.
3. **Activity Data:** `standard_value` and `standard_type` are crucial for understanding the activity of compounds. You will need to standardize this data (e.g., convert all IC50 values to pIC50).
4. **Regression/Classification:** You can build regression models to predict activity values based on molecular descriptors. Or, you can create classification models to predict whether a compound is active or inactive based on a threshold.
5. **Model Evaluation:** Choose appropriate evaluation metrics (e.g., R-squared, RMSE for regression; accuracy, precision, recall, F1-score for classification).

2. Code (SQL and Python) and Error Correction

a) SQL (for data extraction and saving to CSV):

```sql
-- Topic_CheMBL_35_96.sql
-- Extracting 100 rows of data for Topic_CheMBL_35_96

SELECT
    act.molregno,
    act.standard_type,
    act.standard_value,
    act.standard_units,
    act.pchembl_value,
```

```
    md.chembl_id,
    md.pref_name,
    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'  -- Filter for IC50 values (example)
    AND act.standard_units = 'nM' -- Filter for nM units (example)
    AND act.standard_value IS NOT NULL -- Ensure standard_value is not null
    AND act.standard_value::text ~ '^[0-9\.]+$' -- Filter to only include valid values
in column standard_value
LIMIT 100;
```

Explanation:

- **`SELECT` Clause:** Retrieves the necessary columns from the `activities`, `molecule_dictionary`, and `compound_structures` tables.
- **`JOIN` Clause:** Connects the tables based on the `molregno` (molecule registration number).
- **`WHERE` Clause:** Filters the data based on:
    - `act.standard_type = 'IC50'`: Only retrieves data where the activity type is IC50. You can modify this to other activity types as needed (e.g., 'Ki', 'EC50').
    - `act.standard_units = 'nM'`: Filters for results reported in nanomolar (nM) units.
    - `act.standard_value IS NOT NULL`: Excludes entries with missing activity values.
    - `act.standard_value::text ~ '^[0-9\.]+$'`: **Addressing Error a)** This line filters `standard_value` to only include valid values using a regular expression to only include valid integers or floats to exclude values such as ">500".
- **`LIMIT 100` Clause:** Restricts the result set to 100 rows.

How to Run:

1. Open pgAdmin.
2. Connect to your `chembl_35` database (using the credentials you provided).
3. Open a new query window.
4. Paste the SQL code into the query window.
5. Execute the query.
6. Export the results to a CSV file. You can usually do this from pgAdmin's query result view by right-clicking and selecting "Copy with Headers" (or a similar option) and then pasting it into a text file and saving as a CSV.

b) Python (for data processing, RDKit usage, and example analysis):

```
# Topic_CheMBL_35_96_1_Data_Processing.ipynb

import os
import pandas as pd
from rdkit import Chem
from rdkit.Chem import Descriptors
import numpy as np
from sklearn.metrics import mean_squared_error
from sklearn.preprocessing import MinMaxScaler

# Define the base path for your project (replace with your actual path)
base_path = "/path/to/your/project"  # Replace this!
data_path = os.path.join(base_path, "data")
```

```
notebook_path = os.path.join(base_path, "notebooks")
model_path = os.path.join(base_path, "models")

# Ensure directories exist (if they don't, create them)
os.makedirs(data_path, exist_ok=True)
os.makedirs(notebook_path, exist_ok=True)
os.makedirs(model_path, exist_ok=True)

# 1. Load the data from the CSV file
csv_file_path = os.path.join(data_path, "Topic_CheMBL_35_96.csv")  # Adjust filename
if needed
try:
    df = pd.read_csv(csv_file_path)
except FileNotFoundError:
    print(f"Error: File not found at {csv_file_path}. Make sure you saved the CSV
correctly from pgAdmin.")
    exit()

print(f"Data loaded successfully. Shape: {df.shape}")
print(df.head())

# 2. Data Cleaning and Preprocessing
# Drop rows with missing SMILES strings
df = df.dropna(subset=['canonical_smiles'])

# Convert IC50 to pIC50
def ic50_to_pic50(ic50_nM):
    """Converts IC50 (nM) to pIC50."""
    pIC50 = 9 - np.log10(ic50_nM)
    return pIC50

df = df.dropna(subset=['standard_value'])

# Function to try converting to float
def safe_float(x):
    try:
        return float(x)
    except ValueError:
        return np.nan  # Or another suitable replacement like None

# Apply safe_float to 'standard_value'
df['standard_value'] = df['standard_value'].apply(safe_float)
df = df.dropna(subset=['standard_value'])

df['pIC50'] = df['standard_value'].apply(ic50_to_pic50)  # Convert IC50 to pIC50
df = df[np.isfinite(df['pIC50'])] #Remove inf value

# 3. RDKit: Create Molecules and Generate Descriptors
def generate_descriptors(smiles):
    """Generates a list of RDKit descriptors for a given SMILES string."""
    mol = Chem.MolFromSmiles(smiles)
    if mol is None:
        return None  # Handle invalid SMILES strings

    descriptors = {}
    descriptors['MolWt'] = Descriptors.MolWt(mol)
    descriptors['LogP'] = Descriptors.MolLogP(mol)
    descriptors['HBD'] = Descriptors.NumHDonors(mol)
    descriptors['HBA'] = Descriptors.NumHAcceptors(mol)
```

```
    descriptors['TPSA'] = Descriptors.TPSA(mol)
    descriptors['RotatableBonds'] = Descriptors.NumRotatableBonds(mol)
    # Add more descriptors as needed

    return descriptors

df['descriptors'] = df['canonical_smiles'].apply(generate_descriptors)
df = df.dropna(subset=['descriptors']) #Drop rows with missing descriptors
df = df[df['descriptors'].map(lambda d: isinstance(d, dict))] #Making sure all rows
are dictionary and not None

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

# 4. Basic Analysis (Example)
print("\nDescriptive Statistics:")
print(df[['pIC50', 'MolWt', 'LogP', 'HBD', 'HBA', 'TPSA']].describe())

# 5. Example: Simple Linear Regression (Illustrative)
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression

# Select features and target variable
features = ['MolWt', 'LogP', 'HBD', 'HBA', 'TPSA']
target = 'pIC50'

#Handle infinities in features
df = df.replace([np.inf, -np.inf], np.nan)
df = df.dropna(subset=features + [target])

X = df[features]
y = df[target]

# Scale the data
scaler = MinMaxScaler()
X = scaler.fit_transform(X)

# 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)

# Create and train a linear regression model
model = LinearRegression()
model.fit(X_train, y_train)

# Make predictions
y_pred = model.predict(X_test)

# Evaluate the model
rmse = np.sqrt(mean_squared_error(y_test, y_pred)) #Remove squared=False

print(f"\nRoot Mean Squared Error (RMSE): {rmse}")
# Save the model (optional)
import joblib
model_filename = os.path.join(model_path, "linear_regression_model.joblib")
joblib.dump(model, model_filename)
print(f"Model saved to {model_filename}")
```

Explanation:

1. **Import Libraries:** Imports necessary libraries (pandas, RDKit, scikit-learn).
2. **Define Paths:** Defines the paths for data, notebooks, and models based on your AIMLops structure. **Important:** Replace `/path/to/your/project` with the actual path to your project directory.
3. **Load Data:** Loads the CSV file into a pandas DataFrame. Includes error handling if the file isn't found.
4. **Data Cleaning and Preprocessing:**
    - Handles missing SMILES strings.
    - **pIC50 Conversion:** Converts IC50 values (in nM) to pIC50 values using the formula `pIC50 = 9 - log10(IC50)`. This is a common transformation in drug discovery.
    - **Dealing with potentially incorrect standard_values:** Added a layer to handle invalid float values which are saved in the `standard_value` column.
5. **RDKit Descriptor Generation:**
    - **`generate_descriptors(smiles)` Function:** Takes a SMILES string as input and calculates several RDKit descriptors (Molecular Weight, LogP, H-bond donors, H-bond acceptors, TPSA, Rotatable Bonds). You can easily add more descriptors here.
    - Applies the function to the `canonical_smiles` column to create a new `descriptors` column.
    - **Error Handling for Invalid SMILES:** The `generate_descriptors` function now returns `None` if the SMILES string is invalid. The code then removes rows where the `descriptors` column is `None`.
    - Converts the dictionary of descriptors into separate columns in the DataFrame.
6. **Basic Analysis:** Calculates descriptive statistics (mean, std, min, max, etc.) for `pIC50` and the generated descriptors. This helps you understand the data distribution.
7. **Simple Linear Regression (Example):**
    - **Feature and Target Selection:** Selects a few of the generated descriptors as features and `pIC50` as the target variable.
    - **Data Splitting:** Splits the data into training and testing sets.
    - **Model Training:** Creates a linear regression model and trains it on the training data.
    - **Prediction and Evaluation:** Makes predictions on the test data and calculates the Root Mean Squared Error (RMSE).
    - **Model Saving:** Saves the trained model to a file using `joblib`.

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

The `squared=False` parameter was introduced in a later version of scikit-learn. To fix this:

1. **Upgrade scikit-learn:** The best solution is to upgrade your scikit-learn version to the latest stable version:

    ```
    pip install --upgrade scikit-learn
    ```

2. **Remove `squared=False`:** If you cannot upgrade scikit-learn, you can remove the `squared=False` parameter from the `mean_squared_error` function and then take the square root of the result manually:

    ```
    from sklearn.metrics import mean_squared_error
    mse = mean_squared_error(y_test, y_pred)
    rmse = np.sqrt(mse)
    ```

I've corrected this in the code above using the square root function.

3. Examples (5 Examples of Usage)

Here are 5 examples of how you might extend this code to perform more in-depth analysis:

Example 1: Exploring the Correlation between Molecular Weight and pIC50

```
import matplotlib.pyplot as plt
import seaborn as sns

#Create the graph
sns.scatterplot(data=df, x='MolWt', y='pIC50')
plt.title('Molecular Weight vs. pIC50')
plt.xlabel('Molecular Weight')
plt.ylabel('pIC50')
plt.show()

# Calculate the Pearson correlation coefficient
correlation = df['MolWt'].corr(df['pIC50'])
print(f"Pearson correlation between Molecular Weight and pIC50: {correlation}")
```

This example creates a scatter plot to visualize the relationship between molecular weight and pIC50 and calculates the Pearson correlation coefficient to quantify the strength and direction of the linear relationship.

Example 2: Filtering Data by Lipinski's Rule of Five

Lipinski's Rule of Five is a guideline for drug-likeness. It states that a drug candidate should generally have:

- Molecular weight < 500 Da
- LogP < 5
- H-bond donors <= 5
- H-bond acceptors <= 10

```
# Filter data based on Lipinski's Rule of Five
lipinski_df = df[
    (df['MolWt'] < 500) &
    (df['LogP'] < 5) &
    (df['HBD'] <= 5) &
    (df['HBA'] <= 10)
]

print(f"Number of compounds that satisfy Lipinski's Rule of Five: {len(lipinski_df)}")
print(lipinski_df.head())
```

This example filters the DataFrame to keep only compounds that satisfy Lipinski's Rule of Five.

Example 3: Building a Random Forest Regression Model

```
from sklearn.ensemble import RandomForestRegressor

# Features and target (same as before)
features = ['MolWt', 'LogP', 'HBD', 'HBA', 'TPSA']
target = 'pIC50'

#Handle infinities in features
df = df.replace([np.inf, -np.inf], np.nan)
df = df.dropna(subset=features + [target])

X = df[features]
y = df[target]
```

```
# Scale the data
scaler = MinMaxScaler()
X = scaler.fit_transform(X)

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

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

# Make predictions
y_pred_rf = rf_model.predict(X_test)

# Evaluate the model
rmse_rf = np.sqrt(mean_squared_error(y_test, y_pred_rf)) #Remove squared=False
print(f"Random Forest RMSE: {rmse_rf}")

# Save the model
rf_model_filename = os.path.join(model_path, "random_forest_model.joblib")
joblib.dump(rf_model, rf_model_filename)
print(f"Random Forest model saved to {rf_model_filename}")
```

This example demonstrates building a more complex model (Random Forest) for predicting pIC50 values.

Example 4: Visualizing Descriptor Distributions

```
import matplotlib.pyplot as plt
import seaborn as sns

# Create histograms of molecular weight and LogP
plt.figure(figsize=(12, 6))

plt.subplot(1, 2, 1)
sns.histplot(df['MolWt'], kde=True)
plt.title('Distribution of Molecular Weight')

plt.subplot(1, 2, 2)
sns.histplot(df['LogP'], kde=True)
plt.title('Distribution of LogP')

plt.tight_layout()
plt.show()
```

This example generates histograms to visualize the distributions of Molecular Weight and LogP. This helps you understand the range and spread of these properties in your dataset.

Example 5: Calculating and Visualizing TPSA vs pIC50

```
import matplotlib.pyplot as plt
import seaborn as sns

# Create a scatter plot of TPSA vs. pIC50
plt.figure(figsize=(8, 6))
sns.scatterplot(x='TPSA', y='pIC50', data=df)
plt.title('TPSA vs. pIC50')
plt.xlabel('TPSA (Topological Polar Surface Area)')
plt.ylabel('pIC50')
plt.show()
```

```
# Calculate the Pearson correlation coefficient
correlation = df['TPSA'].corr(df['pIC50'])
print(f"Pearson correlation between TPSA and pIC50: {correlation}")
```

This code generates a scatterplot of TPSA vs. pIC50 and calculates their correlation coefficient. TPPSA is a measure of the polar surface area of a molecule, which is related to its ability to penetrate cell membranes.

4. Conclusion

Remember to replace `/path/to/your/project` with your actual project directory. This comprehensive guide, including the corrected code and examples, should help you get started with your Chembl 35 data analysis project. Remember to adapt the SQL queries and Python code to suit your specific research questions and data exploration goals. Good luck!

Traducción al Español:

1. Configuración del Proyecto y Comprensión del Modelo de Análisis

Estás utilizando datos de ChEMBL 35 para explorar propiedades y relaciones de fármacos. El objetivo probablemente sea realizar análisis exploratorio de datos (EDA), ingeniería de características y, posiblemente, construir modelos predictivos basados en la estructura química y los datos de actividad. Los componentes clave son:

- **Base de Datos:** ChEMBL 35 alojada en PostgreSQL (psql).
- **Extracción de Datos:** Consultas SQL para recuperar datos específicos de la base de datos.
- **Procesamiento de Datos:** RDKit para manejar estructuras moleculares, generación de características y manipulación de datos.
- **Modelado:** Scikit-learn para posibles tareas de aprendizaje automático (regresión, clasificación, etc.).
- **Estructura AIMLops:** Organización de su proyecto según las mejores prácticas de AIMLops (datos, notebooks, modelos, etc.).

Consideraciones del Modelo de Análisis:

1. **Exploración de Datos:** Comprender las distribuciones de variables clave como `standard_value`, `standard_type`, `pCHEMBL_value` y las propiedades moleculares. Buscar datos faltantes y valores atípicos.
2. **Ingeniería de Características:** RDKit te permite generar una gran variedad de descriptores moleculares (por ejemplo, área de superficie polar topológica (TPSA), número de donantes/aceptores de enlaces de hidrógeno, peso molecular). Estos descriptores pueden utilizarse como características en tus modelos.
3. **Datos de Actividad:** `standard_value` y `standard_type` son cruciales para comprender la actividad de los compuestos. Tendrás que estandarizar estos datos (por ejemplo, convertir todos los valores de IC50 a pIC50).
4. **Regresión/Clasificación:** Puedes construir modelos de regresión para predecir valores de actividad basados en descriptores moleculares. O bien, puedes crear modelos de clasificación para predecir si un compuesto está activo o inactivo en función de un umbral.
5. **Evaluación del Modelo:** Elegir métricas de evaluación adecuadas (por ejemplo, R-cuadrado, RMSE para regresión; precisión, exhaustividad, puntuación F1 para clasificación).

2. Código (SQL y Python) y Corrección de Errores

a) SQL (para extracción de datos y guardado en CSV): (Ver código SQL en la respuesta en inglés)

b) Python (para procesamiento de datos, uso de RDKit y análisis de ejemplo): (Ver código Python en la respuesta en inglés)

Addressing Error b): `old scikit-learn version does not support parameters squared=False in the mean_squared_error function` (Ver la solución del error en la respuesta en inglés)

3. Ejemplos (5 Ejemplos de Uso)

(Ver los 5 ejemplos de uso en la respuesta en inglés)

4. Conclusión

Recuerda reemplazar `/path/to/your/project` con tu directorio de proyecto real. Esta guía completa, incluido el código corregido y los ejemplos, debería ayudarte a comenzar con tu proyecto de análisis de datos de Chembl 35. Recuerda adaptar las consultas SQL y el código Python para que se ajusten a tus preguntas de investigación específicas y a tus objetivos de exploración de datos. ¡Buena suerte!