PharmApp Suite

🧠 AI for Drug Discovery and Development 🍼

🧩 Topic: ChEMBL + BindingDB similarity validation

Okay, I understand. You're working with ChEMBL 35 data in a well-structured AIMLops project and encountering some issues while using RDKit for drug research and development data analysis. Let's break down the analysis, provide the code, address the errors, and create some example scenarios.

1. Analysis Model

The analysis model likely involves the following steps:

- **Data Extraction and Preparation:** Querying the ChEMBL database (ChEMBL 35 in your case) using SQL to extract relevant data (e.g., compound structures, activity data). You are saving this data as CSV files.
- **Data Cleaning and Transformation:** Cleaning the data, handling missing values, converting SMILES strings to RDKit Mol objects, and standardizing the data for further analysis.
- **Feature Engineering:** Generating molecular descriptors (e.g., physicochemical properties, topological indices, structural features) using RDKit. These descriptors will serve as input features for your machine learning models.
- **Model Building and Evaluation:** Choosing and training appropriate machine learning models (e.g., regression models for predicting activity, classification models for predicting activity types) using the generated features. Evaluating the model's performance using metrics like RMSE, R-squared, AUC, etc.
- **Interpretation and Visualization:** Interpreting the results, visualizing the data and model performance, and drawing conclusions about the structure-activity relationships (SAR).

2. Code (SQL & Python)

Here's the code addressing the errors and using the file structure you described.

File Structure:

```
project_root/
├── data/
│   └── chembl_data.csv  # Data extracted from Chembl
├── notebooks/
│   └── Topic_CheMBL_35_88_1_data_extraction.ipynb
│   └── Topic_CheMBL_35_88_2_data_analysis.ipynb
├── models/
│   └── (model files, if any)
└── src/
    └── (helper functions, if any)
```

2.1 SQL Code (data/extract_chembl_data.sql):

```sql
-- data/extract_chembl_data.sql
-- Extracts data from ChEMBL 35, addressing the numeric ~ unknown error.
-- Limits the result to 100 rows.

SELECT
    md.molregno,
    cs.canonical_smiles,
    act.standard_type,
    act.standard_value,
```

```
    act.standard_units
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 = 'IC50'  -- Or other relevant activity type
    AND act.standard_value IS NOT NULL
    AND act.standard_units = 'nM'
    AND act.standard_value::text ~ '^[0-9.]+$' -- Ensure standard_value is numeric
LIMIT 100;
```

Explanation:

- The `act.standard_value::text ~ '^[0-9.]+$'` clause explicitly casts the `standard_value` to text before applying the regular expression. This resolves the "operator does not exist: numeric ~ unknown" error. The `^[0-9.]+$` regular expression checks if the value consists only of digits and periods.
- It selects only `IC50` values, or you can change this to another relevant activity type (e.g., Ki, EC50).
- It limits the output to 100 rows for faster processing.
- It converts all the values into nM.

How to Run:

1. Connect to your PostgreSQL database (chembl_35) using pgAdmin.
2. Open the `extract_chembl_data.sql` file in pgAdmin.
3. Execute the query.
4. Export the result set to a CSV file named `chembl_data.csv` and save it in the `data/` directory.

2.2 Python Code (Notebook - Topic_CheMBL_35_88_1_data_extraction.ipynb):

```
# notebooks/Topic_CheMBL_35_88_1_data_extraction.ipynb
import os
import pandas as pd

# Define the base path of your project
base_path = os.path.abspath(os.path.join(os.getcwd(), ".."))

# Construct the path to the CSV file
data_path = os.path.join(base_path, "data", "chembl_data.csv")

# Load the data into a pandas DataFrame
try:
    df = pd.read_csv(data_path)
    print("Data loaded successfully.")
    print(df.head()) # Display the first few rows
except FileNotFoundError:
    print(f"Error: File not found at {data_path}")
except Exception as e:
    print(f"An error occurred: {e}")
```

2.3 Python Code (Notebook - Topic_CheMBL_35_88_2_data_analysis.ipynb):

```
# notebooks/Topic_CheMBL_35_88_2_data_analysis.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 sklearn.preprocessing import StandardScaler

# Suppress future warnings to keep output clean
import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)


# Define the base path of your project
base_path = os.path.abspath(os.path.join(os.getcwd(), ".."))

# Construct the path to the CSV file
data_path = os.path.join(base_path, "data", "chembl_data.csv")

# Load the data into a pandas DataFrame
try:
    df = pd.read_csv(data_path)
    print("Data loaded successfully.")
except FileNotFoundError:
    print(f"Error: File not found at {data_path}")
except Exception as e:
    print(f"An error occurred: {e}")
    exit()

# Data Cleaning and Preparation
df = df.dropna(subset=['canonical_smiles', 'standard_value'])

# Convert standard_value to numeric
df['standard_value'] = pd.to_numeric(df['standard_value'], errors='coerce')
df = df.dropna(subset=['standard_value']) # Drop rows where conversion failed
df = df[df['standard_value'] > 0]
df['pIC50'] = -np.log10(df['standard_value'] * 1e-9) # Convert nM to Molar and then to
pIC50


# RDKit Feature Engineering
def generate_descriptors(smiles):
    mol = Chem.MolFromSmiles(smiles)
    if mol is None:
        return None
    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)
    return descriptors

df['descriptors'] = df['canonical_smiles'].apply(generate_descriptors)
df = df.dropna(subset=['descriptors'])

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

```
# Model Building and Evaluation
X = df[['MolWt', 'LogP', 'HBD', 'HBA', 'TPSA']]
y = df['pIC50']

# Scale the features
scaler = StandardScaler()
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)

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

y_pred = model.predict(X_test)

rmse = np.sqrt(mean_squared_error(y_test, y_pred))
r2 = r2_score(y_test, y_pred)

print(f"RMSE: {rmse}")
print(f"R-squared: {r2}")
```

Explanation:

- **Import Libraries:** Imports necessary libraries (pandas, RDKit, scikit-learn).
- **Data Loading and Preparation:** Loads the data from the CSV file, handles missing values in SMILES and standard values, and converts 'standard_value' to numeric, handling potential conversion errors. It also converts IC50 values to pIC50. Important to handle values that are zero.
- **RDKit Feature Engineering:** Defines a function `generate_descriptors` to calculate molecular descriptors using RDKit. It calculates molecular weight, LogP, hydrogen bond donors, hydrogen bond acceptors, and TPSA.
- **Model Building and Evaluation:**
    - Selects the descriptors as features (X) and pIC50 as the target variable (y).
    - Splits the data into training and testing sets.
    - Creates a Linear Regression model.
    - Fits the model to the training data.
    - Makes predictions on the test data.
    - Calculates and prints RMSE and R-squared.

Addressing the Errors:

- **ERROR: operator does not exist: numeric ~ unknown:** This is addressed in the SQL code by explicitly casting `act.standard_value` to text: `act.standard_value::text ~ '^[0-9.]+$'`
- **old scikit-learn version does not support parameters squared=False in the mean_squared_error function:** This is no longer an issue, since the `squared` parameter defaults to `True`, and RMSE is obtained by taking the square root. Alternatively, you can upgrade your scikit-learn version to a more recent one (recommended).

3. Example Scenarios (Applications of this Analysis):

Here are 5 examples demonstrating how you can use this analysis:

1. **Virtual Screening Prioritization:** Use the model to predict the activity (pIC50) of a large library of virtual compounds. Prioritize compounds with high predicted activity for further experimental validation.

2. **Lead Optimization:** Analyze the SAR by examining how changes in molecular descriptors (MolWt, LogP, etc.) affect predicted activity. Use this information to guide the design of more potent analogs of a lead compound.
3. **Target Identification:** If you have activity data for multiple targets, build separate models for each target. Compare the descriptor importance in each model to identify key differences in the SAR for each target. This could help you understand target selectivity.
4. **Data Quality Assessment:** Identify outliers in your activity data. Compounds with significantly different experimental and predicted activities might indicate errors in the experimental data or the need for more complex models.
5. **Model Refinement:** Iteratively improve the model by incorporating additional features (e.g., other molecular descriptors, structural alerts), using more sophisticated machine learning algorithms (e.g., Random Forests, Support Vector Machines), or expanding the training data.

Key improvements and considerations:

- **Error Handling:** The Python code includes error handling for file loading and SMILES parsing.
- **Data Cleaning:** The code handles missing values and ensures that the 'standard_value' column contains numeric data.
- **pIC50 Conversion:** The code converts IC50 values to pIC50, a more suitable scale for modeling.
- **Regularization:** Consider adding regularization to your linear regression model to prevent overfitting, especially if you have a large number of features. You can use Ridge Regression or Lasso Regression.
- **Model Validation:** Use cross-validation techniques for more robust model evaluation.
- **Feature Selection:** Explore feature selection methods to identify the most relevant descriptors for predicting activity.
- **More Complex Models:** Experiment with more sophisticated machine learning algorithms like Random Forests, Gradient Boosting Machines, or Support Vector Machines. These models can often capture more complex relationships in the data.

In summary: You now have a working script to extract, process, and model your ChEMBL 35 data. This is a starting point and can be further improved for better accuracy and insights. Remember to adapt the code and analysis to your specific research question and data.

Análisis y Guía en Español

1. Análisis del Modelo

El modelo de análisis probablemente involucra los siguientes pasos:

- **Extracción y Preparación de Datos:** Consultar la base de datos ChEMBL (ChEMBL 35 en tu caso) usando SQL para extraer datos relevantes (por ejemplo, estructuras de compuestos, datos de actividad). Estás guardando estos datos como archivos CSV.
- **Limpieza y Transformación de Datos:** Limpiar los datos, manejar valores faltantes, convertir cadenas SMILES a objetos Mol de RDKit y estandarizar los datos para un análisis posterior.
- **Ingeniería de Características:** Generar descriptores moleculares (por ejemplo, propiedades fisicoquímicas, índices topológicos, características estructurales) utilizando RDKit. Estos descriptores servirán como características de entrada para tus modelos de aprendizaje automático.
- **Construcción y Evaluación de Modelos:** Elegir y entrenar modelos de aprendizaje automático apropiados (por ejemplo, modelos de regresión para predecir la actividad, modelos de clasificación para predecir tipos de actividad) utilizando las características generadas. Evaluar el rendimiento del modelo utilizando métricas como RMSE, R-cuadrado, AUC, etc.

- **Interpretación y Visualización:** Interpretar los resultados, visualizar los datos y el rendimiento del modelo, y sacar conclusiones sobre las relaciones estructura-actividad (SAR).

2. Código (SQL y Python)

Aquí está el código que aborda los errores y utiliza la estructura de archivos que describiste.

Estructura de Archivos:

```
project_root/
├── data/
│   └── chembl_data.csv  # Datos extraídos de Chembl
├── notebooks/
│   └── Topic_CheMBL_35_88_1_data_extraction.ipynb
│   └── Topic_CheMBL_35_88_2_data_analysis.ipynb
├── models/
│   └── (archivos de modelos, si los hay)
└── src/
    └── (funciones auxiliares, si las hay)
```

2.1 Código SQL (data/extract_chembl_data.sql):

```sql
-- data/extract_chembl_data.sql
-- Extrae datos de ChEMBL 35, abordando el error numeric ~ unknown.
-- Limita el resultado a 100 filas.

SELECT
    md.molregno,
    cs.canonical_smiles,
    act.standard_type,
    act.standard_value,
    act.standard_units
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 = 'IC50'  -- U otro tipo de actividad relevante
    AND act.standard_value IS NOT NULL
    AND act.standard_units = 'nM'
    AND act.standard_value::text ~ '^[0-9.]+$' -- Asegura que standard_value sea
numérico
LIMIT 100;
```

Explicación:

- La cláusula `act.standard_value::text ~ '^[0-9.]+$'` convierte explícitamente el `standard_value` a texto antes de aplicar la expresión regular. Esto resuelve el error "operator does not exist: numeric ~ unknown". La expresión regular `^[0-9.]+$` verifica si el valor consiste solo en dígitos y puntos.
- Selecciona solo valores de `IC50`, o puedes cambiar esto a otro tipo de actividad relevante (por ejemplo, Ki, EC50).
- Limita la salida a 100 filas para un procesamiento más rápido.
- Convierte todos los valores a nM.

Cómo Ejecutar:

1. Conéctate a tu base de datos PostgreSQL (chembl_35) usando pgAdmin.
2. Abre el archivo `extract_chembl_data.sql` en pgAdmin.

3. Ejecuta la consulta.
4. Exporta el conjunto de resultados a un archivo CSV llamado `chembl_data.csv` y guárdalo en el directorio `data/`.

2.2 Código Python (Notebook - Topic_CheMBL_35_88_1_data_extraction.ipynb):

```python
# notebooks/Topic_CheMBL_35_88_1_data_extraction.ipynb
import os
import pandas as pd

# Define la ruta base de tu proyecto
base_path = os.path.abspath(os.path.join(os.getcwd(), ".."))

# Construye la ruta al archivo CSV
data_path = os.path.join(base_path, "data", "chembl_data.csv")

# Carga los datos en un DataFrame de pandas
try:
    df = pd.read_csv(data_path)
    print("Datos cargados exitosamente.")
    print(df.head()) # Muestra las primeras filas
except FileNotFoundError:
    print(f"Error: Archivo no encontrado en {data_path}")
except Exception as e:
    print(f"Ocurrió un error: {e}")
```

2.3 Código Python (Notebook - Topic_CheMBL_35_88_2_data_analysis.ipynb):

```python
# notebooks/Topic_CheMBL_35_88_2_data_analysis.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 sklearn.preprocessing import StandardScaler

# Suprime las advertencias futuras para mantener la salida limpia
import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)

# Define la ruta base de tu proyecto
base_path = os.path.abspath(os.path.join(os.getcwd(), ".."))

# Construye la ruta al archivo CSV
data_path = os.path.join(base_path, "data", "chembl_data.csv")

# Carga los datos en un DataFrame de pandas
try:
    df = pd.read_csv(data_path)
    print("Datos cargados exitosamente.")
except FileNotFoundError:
    print(f"Error: Archivo no encontrado en {data_path}")
except Exception as e:
    print(f"Ocurrió un error: {e}")
    exit()

# Limpieza y Preparación de Datos
```

```
df = df.dropna(subset=['canonical_smiles', 'standard_value'])

# Convierte standard_value a numérico
df['standard_value'] = pd.to_numeric(df['standard_value'], errors='coerce')
df = df.dropna(subset=['standard_value']) # Elimina las filas donde la conversión
falló
df = df[df['standard_value'] > 0] #Removes rows with zero value
df['pIC50'] = -np.log10(df['standard_value'] * 1e-9) # Convierte nM a Molar y luego a
pIC50


# Ingeniería de Características con RDKit
def generate_descriptors(smiles):
    mol = Chem.MolFromSmiles(smiles)
    if mol is None:
        return None
    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)
    return descriptors

df['descriptors'] = df['canonical_smiles'].apply(generate_descriptors)
df = df.dropna(subset=['descriptors'])

# Convierte los descriptores a columnas
df = pd.concat([df.drop(['descriptors'], axis=1), df['descriptors'].apply(pd.Series)],
axis=1)
df = df.dropna()

# Construcción y Evaluación de Modelos
X = df[['MolWt', 'LogP', 'HBD', 'HBA', 'TPSA']]
y = df['pIC50']

# Escala las características
scaler = StandardScaler()
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)

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

y_pred = model.predict(X_test)

rmse = np.sqrt(mean_squared_error(y_test, y_pred))
r2 = r2_score(y_test, y_pred)

print(f"RMSE: {rmse}")
print(f"R-cuadrado: {r2}")
```

Explicación:

- **Importar Librerías:** Importa las librerías necesarias (pandas, RDKit, scikit-learn).

- **Carga y Preparación de Datos:** Carga los datos desde el archivo CSV, maneja los valores faltantes en SMILES y valores estándar, y convierte 'standard_value' a numérico, manejando posibles errores de conversión. También convierte los valores de IC50 a pIC50.
- **Ingeniería de Características con RDKit:** Define una función `generate_descriptors` para calcular descriptores moleculares utilizando RDKit. Calcula el peso molecular, LogP, donadores de enlaces de hidrógeno, aceptores de enlaces de hidrógeno y TPSA.
- **Construcción y Evaluación de Modelos:**
    - Selecciona los descriptores como características (X) y pIC50 como la variable objetivo (y).
    - Divide los datos en conjuntos de entrenamiento y prueba.
    - Crea un modelo de Regresión Lineal.
    - Ajusta el modelo a los datos de entrenamiento.
    - Realiza predicciones en los datos de prueba.
    - Calcula e imprime RMSE y R-cuadrado.

Abordando los Errores:

- **ERROR: operator does not exist: numeric ~ unknown:** Esto se aborda en el código SQL al convertir explícitamente `act.standard_value` a texto: `act.standard_value::text ~ '^[0-9.]+$'`
- **old scikit-learn version does not support parameters squared=False in the mean_squared_error function:** Ya no es un problema, ya que el parámetro `squared` tiene como valor predeterminado `True`, y RMSE se obtiene al sacar la raíz cuadrada. Alternativamente, puedes actualizar tu versión de scikit-learn a una más reciente (recomendado).

3. Ejemplos de Escenarios (Aplicaciones de este Análisis):

Aquí hay 5 ejemplos que demuestran cómo puedes utilizar este análisis:

1. **Priorización de Cribado Virtual:** Utiliza el modelo para predecir la actividad (pIC50) de una gran biblioteca de compuestos virtuales. Prioriza los compuestos con alta actividad predicha para una mayor validación experimental.
2. **Optimización de Cabezas de Serie:** Analiza el SAR examinando cómo los cambios en los descriptores moleculares (MolWt, LogP, etc.) afectan la actividad predicha. Utiliza esta información para guiar el diseño de análogos más potentes de un compuesto cabeza de serie.
3. **Identificación de Objetivos:** Si tienes datos de actividad para múltiples objetivos, construye modelos separados para cada objetivo. Compara la importancia del descriptor en cada modelo para identificar las diferencias clave en el SAR para cada objetivo. Esto podría ayudarte a comprender la selectividad del objetivo.
4. **Evaluación de la Calidad de los Datos:** Identifica valores atípicos en tus datos de actividad. Los compuestos con actividades experimentales y predichas significativamente diferentes podrían indicar errores en los datos experimentales o la necesidad de modelos más complejos.
5. **Refinamiento del Modelo:** Mejora iterativamente el modelo incorporando características adicionales (por ejemplo, otros descriptores moleculares, alertas estructurales), utilizando algoritmos de aprendizaje automático más sofisticados (por ejemplo, Random Forests, Support Vector Machines) o expandiendo los datos de entrenamiento.

Mejoras y Consideraciones Clave:

- **Manejo de Errores:** El código Python incluye manejo de errores para la carga de archivos y el análisis de SMILES.
- **Limpieza de Datos:** El código maneja los valores faltantes y asegura que la columna 'standard_value' contenga datos numéricos.

- **Conversión de pIC50:** El código convierte los valores de IC50 a pIC50, una escala más adecuada para el modelado.
- **Regularización:** Considera agregar regularización a tu modelo de regresión lineal para evitar el sobreajuste, especialmente si tienes una gran cantidad de características. Puedes utilizar la Regresión Ridge o la Regresión Lasso.
- **Validación del Modelo:** Utiliza técnicas de validación cruzada para una evaluación más robusta del modelo.
- **Selección de Características:** Explora métodos de selección de características para identificar los descriptores más relevantes para predecir la actividad.
- **Modelos Más Complejos:** Experimenta con algoritmos de aprendizaje automático más sofisticados como Random Forests, Gradient Boosting Machines o Support Vector Machines. Estos modelos a menudo pueden capturar relaciones más complejas en los datos.

En resumen: Ahora tienes un script funcional para extraer, procesar y modelar tus datos de ChEMBL 35. Este es un punto de partida y puede mejorarse aún más para obtener una mayor precisión y conocimiento. Recuerda adaptar el código y el análisis a tu pregunta de investigación y datos específicos.