PharmApp Suite

🧠 AI for Drug Discovery and Development 🧪

🧩 Topic: Linking ChEMBL drugs to clinical phases

Okay, I understand. You're working on a Chembl 35 data analysis project using RDKit, following an AIMLops template, and encountering some errors. Let's break down the analysis, provide code (SQL and Python), and address the errors you're seeing.

1. Analysis Model:

Your project focuses on Chembl 35 data, likely aiming to extract insights relevant to drug discovery and development. The specific topic "Topic_CheMBL_35_41" suggests you're investigating a particular area within the vast Chembl dataset.

The analysis involves these steps:

- **Data Extraction (SQL):** Retrieve relevant data from the Chembl 35 database using SQL queries. This will involve selecting specific tables (e.g., activities, molecules, targets) and filtering them based on your research question. You want to limit the initial dataset to 100 rows for faster prototyping.
- **Data Processing (Python/RDKit):** Load the extracted data into a Pandas DataFrame using Python. Then, use RDKit to process the molecule data. This may include:
    - **SMILES Parsing:** Converting SMILES strings to RDKit molecule objects.
    - **Molecular Property Calculation:** Calculating descriptors like molecular weight, logP, number of hydrogen bond donors/acceptors, etc.
    - **Fingerprint Generation:** Generating molecular fingerprints (e.g., Morgan fingerprints) for similarity searches and machine learning models.
- **Data Analysis & Modeling (Python/Scikit-learn):** Perform statistical analysis and/or build predictive models using scikit-learn. This could involve:
    - **Regression:** Predicting activity values (e.g., IC50) based on molecular descriptors.
    - **Classification:** Classifying molecules as active or inactive based on a threshold.
    - **Clustering:** Grouping molecules based on their properties.

Addressing Errors:

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

  This error occurs in your SQL query. The ~ operator represents regular expression matching in PostgreSQL. However, it seems the `act.standard_value` column is a numeric type (e.g., integer or float), not a text type. You cannot directly use regular expressions on numeric columns.

  Solution: Convert the `standard_value` column to text before applying the regular expression. Use `CAST(act.standard_value AS TEXT)` or `act.standard_value::TEXT`.

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

  This error indicates you are using an older version of scikit-learn (likely pre-0.20). The `squared=False` parameter was introduced in scikit-learn 0.20 to return the Root Mean Squared Error (RMSE) directly from `mean_squared_error`.

Solution:

1. **Update scikit-learn:** The best solution is to update scikit-learn to the latest version (or at least 0.20 or later) using `pip install -U scikit-learn`.
2. **Calculate RMSE manually:** If updating is not possible, calculate the RMSE manually by taking the square root of the mean squared error:

```
from sklearn.metrics import mean_squared_error
import numpy as np

mse = mean_squared_error(y_true, y_pred)
rmse = np.sqrt(mse)
print(f"RMSE: {rmse}")
```

2. SQL and Python Code:

Here's the SQL and Python code, incorporating the fixes and considering your project structure. I'll assume your "Topic_CheMBL_35_41" relates to a specific target or a specific range of activity values. I'll provide a generic example; adapt the `WHERE` clause to match your actual topic.

SQL (data/Topic_CheMBL_35_41.sql):

```
-- Topic_CheMBL_35_41.sql
-- Extracts activity data for a specific target (e.g., CHEMBL205) and filters for
standard_type 'IC50'.
-- Limits to 100 rows.  Adjust the WHERE clause to fit your specific
"Topic_CheMBL_35_41".

SELECT
    act.activity_id,
    act.molregno,
    act.standard_type,
    act.standard_value,
    act.standard_units,
    md.chembl_id,
    md.canonical_smiles
FROM
    activities act
JOIN
    molecule_dictionary md ON act.molregno = md.molregno
WHERE
    act.standard_type = 'IC50'
    AND act.standard_units = 'nM'
    --AND act.standard_value BETWEEN 10 AND 1000 -- Example: Filter for IC50 values
between 10 and 1000 nM
    --AND md.chembl_id = 'CHEMBL205' -- Example: Filter for molecules targeting
CHEMBL205 (a specific target)
    AND act.standard_value IS NOT NULL
ORDER BY act.standard_value ASC
LIMIT 100;

-- Save the output as Topic_CheMBL_35_41.csv using pgAdmin
```

Important: Run this SQL query in pgAdmin. Then, use pgAdmin's export functionality to save the results as a CSV file named `Topic_CheMBL_35_41.csv` in the `../data/` directory.

Python (notebooks/Topic_CheMBL_35_41_1_Data_Loading_and_Processing.ipynb):

```
# notebooks/Topic_CheMBL_35_41_1_Data_Loading_and_Processing.ipynb
```

```python
import os
import pandas as pd
from rdkit import Chem
from rdkit.Chem import Descriptors
import numpy as np  # Import numpy

# Define paths according to AIMLops template
base_path = os.path.dirname(os.getcwd())  # Go up one level to the project root
data_path = os.path.join(base_path, 'data')
csv_file = os.path.join(data_path, 'Topic_CheMBL_35_41.csv')
print(f"Data path: {csv_file}")

# Load the data
try:
    df = pd.read_csv(csv_file)
    print(f"Data loaded successfully. Shape: {df.shape}")
except FileNotFoundError:
    print(f"Error: File not found at {csv_file}.  Make sure you ran the SQL and
exported the CSV.")
    exit()

# Basic Data Cleaning and RDKit Processing
df = df.dropna(subset=['canonical_smiles']) # Drop rows with missing SMILES
df = df[df['canonical_smiles'] != ''] # Drop rows with empty SMILES strings

# Create RDKit molecule objects
df['molecule'] = df['canonical_smiles'].apply(lambda x: Chem.MolFromSmiles(x))
df = df.dropna(subset=['molecule']) # Remove rows where RDKit failed to parse SMILES

print(f"Number of valid molecules: {len(df)}")

# Example: Calculate Molecular Weight
df['mol_weight'] = df['molecule'].apply(lambda x: Descriptors.MolWt(x))

# Example: Function to calculate LogP (Octanol-water partition coefficient)
def calculate_logp(mol):
    try:
        return Descriptors.MolLogP(mol)
    except:
        return None

df['logp'] = df['molecule'].apply(calculate_logp)

#Clean the dataframe after all operations
df = df.dropna(subset=['logp'])

print(df[['chembl_id', 'canonical_smiles', 'mol_weight', 'logp']].head())
```

Python (notebooks/Topic_CheMBL_35_41_2_Analysis_and_Modeling.ipynb):

```python
# notebooks/Topic_CheMBL_35_41_2_Analysis_and_Modeling.ipynb

import os
import pandas as pd
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
import numpy as np
```

```python
# Define paths
base_path = os.path.dirname(os.getcwd())
data_path = os.path.join(base_path, 'data')
csv_file = os.path.join(data_path, 'Topic_CheMBL_35_41.csv')

# Load the data
df = pd.read_csv(csv_file)

# Data Cleaning and Preprocessing (same as previous notebook, but make sure these steps are consistent)
df = df.dropna(subset=['canonical_smiles', 'standard_value'])
df = df[df['canonical_smiles'] != '']
df['molecule'] = df['canonical_smiles'].apply(lambda x: Chem.MolFromSmiles(x))
df = df.dropna(subset=['molecule'])
df['mol_weight'] = df['molecule'].apply(lambda x: Descriptors.MolWt(x))
df['logp'] = df['molecule'].apply(calculate_logp) #Make sure to define or import calculate_logp function in this notebook as well
df = df.dropna(subset=['logp'])

# Prepare data for modeling (example: predicting standard_value from mol_weight and logp)
X = df[['mol_weight', 'logp']]
y = df['standard_value']

# Handle potential infinite values in X by replacing them with a large number
X = X.replace([np.inf, -np.inf], np.nan)  # Replace inf with NaN
X = X.fillna(X.max())  # Fill NaN with the maximum value in the column

# Scale the features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random_state=42)

# Train a linear regression model
model = LinearRegression()
model.fit(X_train, y_train)

# Make predictions
y_pred = model.predict(X_test)

# Evaluate the model
mse = mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse)  # Calculate RMSE manually (compatible with older scikit-learn)
r2 = r2_score(y_test, y_pred)

print(f"Mean Squared Error (MSE): {mse}")
print(f"Root Mean Squared Error (RMSE): {rmse}")
print(f"R-squared (R2): {r2}")
```

3. Five Examples of Adapted Code (Varying the Analysis):

Here are five examples of how you can adapt the above code to explore different aspects of the Chembl data. Remember to modify the SQL queries and Python code accordingly.

1. **Target-Specific Activity Prediction:** Focus on a single protein target (e.g., CHEMBL205 for the beta-2 adrenergic receptor) and try to predict the IC50 values.

    - **SQL:** Add `AND md.chembl_id = 'CHEMBL205'` to the `WHERE` clause.
    - **Python:** Use molecular descriptors and machine learning to predict IC50. Consider feature selection techniques to identify the most relevant descriptors.
2. **Activity Cliff Analysis:** Identify pairs of molecules with very similar structures but significantly different activities ("activity cliffs").
    - **SQL:** Retrieve SMILES and activity values.
    - **Python:** Calculate molecular similarity (e.g., using Tanimoto coefficient on Morgan fingerprints). Identify molecule pairs with high similarity and large activity differences.
3. **SAR (Structure-Activity Relationship) Exploration:** Investigate how changes in molecular structure affect activity.
    - **SQL:** Retrieve SMILES, activity values, and potentially substructure information (if available in Chembl).
    - **Python:** Visualize the relationship between specific structural features (e.g., the presence of a particular substituent) and activity. Consider using matched molecular pair analysis (MMPA).
4. **LogP and Activity Relationship:** Examine the correlation between lipophilicity (LogP) and activity.
    - **SQL:** Retrieve SMILES and activity values.
    - **Python:** Calculate LogP using RDKit. Create scatter plots of LogP vs. activity. Calculate the Pearson correlation coefficient.
5. **Molecular Weight and Activity Relationship:** Investigate the relationship between molecular weight and activity. This is often a constraint in drug design.
    - **SQL:** Retrieve SMILES and activity values.
    - **Python:** Calculate molecular weight using RDKit. Create scatter plots of molecular weight vs. activity. Calculate the Pearson correlation coefficient. Filter data for molecules with a specific molecular weight range.

Important Considerations:

- **Error Handling:** Add more comprehensive error handling to your Python code to gracefully handle potential issues (e.g., invalid SMILES strings, missing data).
- **Data Validation:** Validate the data you extract from Chembl. Check for inconsistencies or errors.
- **Feature Engineering:** Experiment with different molecular descriptors and feature engineering techniques to improve the performance of your models.
- **Model Evaluation:** Use appropriate metrics to evaluate the performance of your machine learning models (e.g., RMSE, R2 score, AUC).
- **Documentation:** Document your code clearly, explaining the purpose of each step and the rationale behind your choices.
- **Virtual Environments:** Use virtual environments (e.g., `venv` or `conda`) to manage your project dependencies and avoid conflicts with other Python projects.

By following these steps, addressing the errors, and adapting the examples, you should be able to effectively analyze the Chembl 35 data and extract valuable insights for drug research and development. Good luck!

Traducción al español:

Ok, entiendo. Estás trabajando en un proyecto de análisis de datos de Chembl 35 usando RDKit, siguiendo una plantilla AIMLops, y te encuentras con algunos errores. Vamos a desglosar el análisis, proporcionar código (SQL y Python), y abordar los errores que estás viendo.

1. Modelo de Análisis:

Tu proyecto se centra en los datos de Chembl 35, probablemente con el objetivo de extraer información relevante para el descubrimiento y desarrollo de fármacos. El tema específico "Topic_CheMBL_35_41" sugiere que estás investigando un área particular dentro del vasto conjunto de datos de Chembl.

El análisis implica estos pasos:

- **Extracción de Datos (SQL):** Recuperar datos relevantes de la base de datos de Chembl 35 usando consultas SQL. Esto implicará seleccionar tablas específicas (por ejemplo, actividades, moléculas, objetivos) y filtrarlas en función de tu pregunta de investigación. Quieres limitar el conjunto de datos inicial a 100 filas para una creación de prototipos más rápida.
- **Procesamiento de Datos (Python/RDKit):** Cargar los datos extraídos en un DataFrame de Pandas usando Python. Luego, usar RDKit para procesar los datos de las moléculas. Esto podría incluir:
    - **Análisis de SMILES:** Convertir cadenas SMILES en objetos moleculares de RDKit.
    - **Cálculo de Propiedades Moleculares:** Calcular descriptores como el peso molecular, logP, número de donantes/aceptores de enlaces de hidrógeno, etc.
    - **Generación de Huellas Digitales:** Generar huellas digitales moleculares (por ejemplo, huellas digitales de Morgan) para búsquedas de similitud y modelos de aprendizaje automático.
- **Análisis de Datos y Modelado (Python/Scikit-learn):** Realizar análisis estadísticos y/o construir modelos predictivos usando scikit-learn. Esto podría implicar:
    - **Regresión:** Predecir valores de actividad (por ejemplo, IC50) basados en descriptores moleculares.
    - **Clasificación:** Clasificar moléculas como activas o inactivas basándose en un umbral.
    - **Agrupamiento:** Agrupar moléculas basándose en sus propiedades.

Abordando los Errores:

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

  Este error ocurre en tu consulta SQL. El operador ~ representa la coincidencia de expresiones regulares en PostgreSQL. Sin embargo, parece que la columna `act.standard_value` es un tipo numérico (por ejemplo, entero o flotante), no un tipo de texto. No puedes usar directamente expresiones regulares en columnas numéricas.

  Solución: Convierte la columna `standard_value` a texto antes de aplicar la expresión regular. Usa `CAST(act.standard_value AS TEXT)` o `act.standard_value::TEXT`.

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

  Este error indica que estás usando una versión antigua de scikit-learn (probablemente anterior a 0.20). El parámetro `squared=False` se introdujo en scikit-learn 0.20 para devolver el Error Cuadrático Medio Raíz (RMSE) directamente desde `mean_squared_error`.

  Solución:

  1. **Actualizar scikit-learn:** La mejor solución es actualizar scikit-learn a la última versión (o al menos 0.20 o posterior) usando `pip install -U scikit-learn`.
  2. **Calcular RMSE manualmente:** Si la actualización no es posible, calcula el RMSE manualmente tomando la raíz cuadrada del error cuadrático medio:

```
from sklearn.metrics import mean_squared_error
import numpy as np

mse = mean_squared_error(y_true, y_pred)
rmse = np.sqrt(mse)
print(f"RMSE: {rmse}")
```

2. Código SQL y Python:

Aquí está el código SQL y Python, incorporando las correcciones y considerando la estructura de tu proyecto. Asumiré que tu "Topic_CheMBL_35_41" se relaciona con un objetivo específico o un rango específico de valores de actividad. Proporcionaré un ejemplo genérico; adapta la cláusula `WHERE` para que coincida con tu tema real.

SQL (data/Topic_CheMBL_35_41.sql):

```
-- Topic_CheMBL_35_41.sql
-- Extrae datos de actividad para un objetivo específico (ej., CHEMBL205) y filtra por standard_type 'IC50'.
-- Limita a 100 filas. Ajusta la cláusula WHERE para que se ajuste a tu "Topic_CheMBL_35_41" específico.

SELECT
    act.activity_id,
    act.molregno,
    act.standard_type,
    act.standard_value,
    act.standard_units,
    md.chembl_id,
    md.canonical_smiles
FROM
    activities act
JOIN
    molecule_dictionary md ON act.molregno = md.molregno
WHERE
    act.standard_type = 'IC50'
    AND act.standard_units = 'nM'
    --AND act.standard_value BETWEEN 10 AND 1000 -- Ejemplo: Filtrar por valores de IC50 entre 10 y 1000 nM
    --AND md.chembl_id = 'CHEMBL205' -- Ejemplo: Filtrar por moléculas dirigidas a CHEMBL205 (un objetivo específico)
    AND act.standard_value IS NOT NULL
ORDER BY act.standard_value ASC
LIMIT 100;

-- Guarda la salida como Topic_CheMBL_35_41.csv usando pgAdmin
```

Importante: Ejecuta esta consulta SQL en pgAdmin. Luego, usa la funcionalidad de exportación de pgAdmin para guardar los resultados como un archivo CSV llamado `Topic_CheMBL_35_41.csv` en el directorio `../data/`.

Python (notebooks/Topic_CheMBL_35_41_1_Data_Loading_and_Processing.ipynb):

```
# notebooks/Topic_CheMBL_35_41_1_Data_Loading_and_Processing.ipynb

import os
import pandas as pd
from rdkit import Chem
from rdkit.Chem import Descriptors
import numpy as np  # Import numpy
```

```
# Define rutas de acuerdo con la plantilla AIMLops
base_path = os.path.dirname(os.getcwd())  # Sube un nivel al directorio raíz del proyecto
data_path = os.path.join(base_path, 'data')
csv_file = os.path.join(data_path, 'Topic_CheMBL_35_41.csv')
print(f"Data path: {csv_file}")

# Cargar los datos
try:
    df = pd.read_csv(csv_file)
    print(f"Data loaded successfully. Shape: {df.shape}")
except FileNotFoundError:
    print(f"Error: File not found at {csv_file}.  Asegúrate de haber ejecutado el SQL y exportado el CSV.")
    exit()

# Limpieza básica de datos y procesamiento con RDKit
df = df.dropna(subset=['canonical_smiles']) # Eliminar filas con SMILES faltantes
df = df[df['canonical_smiles'] != ''] # Eliminar filas con cadenas SMILES vacías

# Crear objetos moleculares de RDKit
df['molecule'] = df['canonical_smiles'].apply(lambda x: Chem.MolFromSmiles(x))
df = df.dropna(subset=['molecule']) # Eliminar filas donde RDKit no pudo analizar SMILES

print(f"Number of valid molecules: {len(df)}")

# Ejemplo: Calcular el Peso Molecular
df['mol_weight'] = df['molecule'].apply(lambda x: Descriptors.MolWt(x))

# Ejemplo: Función para calcular LogP (Coeficiente de partición Octanol-agua)
def calculate_logp(mol):
    try:
        return Descriptors.MolLogP(mol)
    except:
        return None

df['logp'] = df['molecule'].apply(calculate_logp)

#Limpiar el dataframe después de todas las operaciones
df = df.dropna(subset=['logp'])

print(df[['chembl_id', 'canonical_smiles', 'mol_weight', 'logp']].head())
```

Python (notebooks/Topic_CheMBL_35_41_2_Analysis_and_Modeling.ipynb):

```
# notebooks/Topic_CheMBL_35_41_2_Analysis_and_Modeling.ipynb

import os
import pandas as pd
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
import numpy as np

# Definir rutas
base_path = os.path.dirname(os.getcwd())
data_path = os.path.join(base_path, 'data')
csv_file = os.path.join(data_path, 'Topic_CheMBL_35_41.csv')
```

```python
# Cargar los datos
df = pd.read_csv(csv_file)

# Limpieza y preprocesamiento de datos (igual que el notebook anterior, pero asegúrate de que estos pasos sean consistentes)
df = df.dropna(subset=['canonical_smiles', 'standard_value'])
df = df[df['canonical_smiles'] != '']
df['molecule'] = df['canonical_smiles'].apply(lambda x: Chem.MolFromSmiles(x))
df = df.dropna(subset=['molecule'])
df['mol_weight'] = df['molecule'].apply(lambda x: Descriptors.MolWt(x))
df['logp'] = df['molecule'].apply(calculate_logp) #Asegúrate de definir o importar la función calculate_logp también en este notebook
df = df.dropna(subset=['logp'])

# Preparar los datos para el modelado (ejemplo: predecir standard_value a partir de mol_weight y logp)
X = df[['mol_weight', 'logp']]
y = df['standard_value']

# Manejar posibles valores infinitos en X reemplazándolos con un número grande
X = X.replace([np.inf, -np.inf], np.nan)  # Reemplazar inf con NaN
X = X.fillna(X.max())  # Llenar NaN con el valor máximo en la columna

# Escalar las características
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Dividir los datos en conjuntos de entrenamiento y prueba
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random_state=42)

# Entrenar un modelo de regresión lineal
model = LinearRegression()
model.fit(X_train, y_train)

# Hacer predicciones
y_pred = model.predict(X_test)

# Evaluar el modelo
mse = mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse)  # Calcular RMSE manualmente (compatible con scikit-learn antiguo)
r2 = r2_score(y_test, y_pred)

print(f"Mean Squared Error (MSE): {mse}")
print(f"Root Mean Squared Error (RMSE): {rmse}")
print(f"R-squared (R2): {r2}")
```

3. Cinco Ejemplos de Código Adaptado (Variando el Análisis):

Aquí hay cinco ejemplos de cómo puedes adaptar el código anterior para explorar diferentes aspectos de los datos de Chembl. Recuerda modificar las consultas SQL y el código Python en consecuencia.

1. **Predicción de Actividad Específica del Objetivo:** Centrarse en un solo objetivo proteico (por ejemplo, CHEMBL205 para el receptor adrenérgico beta-2) e intentar predecir los valores de IC50.
    - **SQL:** Agregar `AND md.chembl_id = 'CHEMBL205'` a la cláusula `WHERE`.

    - **Python:** Usar descriptores moleculares y aprendizaje automático para predecir IC50. Considerar técnicas de selección de características para identificar los descriptores más relevantes.
2. **Análisis de Acantilados de Actividad:** Identificar pares de moléculas con estructuras muy similares pero actividades significativamente diferentes ("acantilados de actividad").
    - **SQL:** Recuperar SMILES y valores de actividad.
    - **Python:** Calcular la similitud molecular (por ejemplo, usando el coeficiente de Tanimoto en las huellas digitales de Morgan). Identificar pares de moléculas con alta similitud y grandes diferencias de actividad.
3. **Exploración SAR (Relación Estructura-Actividad):** Investigar cómo los cambios en la estructura molecular afectan la actividad.
    - **SQL:** Recuperar SMILES, valores de actividad y, potencialmente, información de subestructura (si está disponible en Chembl).
    - **Python:** Visualizar la relación entre características estructurales específicas (por ejemplo, la presencia de un sustituyente particular) y la actividad. Considerar el uso del análisis de pares moleculares emparejados (MMPA).
4. **Relación LogP y Actividad:** Examinar la correlación entre la lipofilicidad (LogP) y la actividad.
    - **SQL:** Recuperar SMILES y valores de actividad.
    - **Python:** Calcular LogP usando RDKit. Crear diagramas de dispersión de LogP vs. actividad. Calcular el coeficiente de correlación de Pearson.
5. **Relación Peso Molecular y Actividad:** Investigar la relación entre el peso molecular y la actividad. Esto suele ser una limitación en el diseño de fármacos.
    - **SQL:** Recuperar SMILES y valores de actividad.
    - **Python:** Calcular el peso molecular usando RDKit. Crear diagramas de dispersión del peso molecular vs. actividad. Calcular el coeficiente de correlación de Pearson. Filtrar datos para moléculas con un rango de peso molecular específico.

Consideraciones Importantes:

- **Manejo de Errores:** Agregar un manejo de errores más completo a tu código Python para manejar con elegancia posibles problemas (por ejemplo, cadenas SMILES inválidas, datos faltantes).
- **Validación de Datos:** Validar los datos que extraes de Chembl. Verificar si hay inconsistencias o errores.
- **Ingeniería de Características:** Experimentar con diferentes descriptores moleculares y técnicas de ingeniería de características para mejorar el rendimiento de tus modelos.
- **Evaluación del Modelo:** Usar métricas apropiadas para evaluar el rendimiento de tus modelos de aprendizaje automático (por ejemplo, RMSE, puntuación R2, AUC).
- **Documentación:** Documentar tu código claramente, explicando el propósito de cada paso y la justificación detrás de tus decisiones.
- **Entornos Virtuales:** Usar entornos virtuales (por ejemplo, `venv` o `conda`) para administrar las dependencias de tu proyecto y evitar conflictos con otros proyectos de Python.

Siguiendo estos pasos, abordando los errores y adaptando los ejemplos, deberías ser capaz de analizar eficazmente los datos de Chembl 35 y extraer información valiosa para la investigación y el desarrollo de fármacos. ¡Buena suerte!