# PharmApp Suite

## 🧠 AI for Drug Discovery and Development 🧪

Copyright 2025 | RnD Pharma Plus | www.nghiencuuthuoc.com | Zalo: +84888999311 |

## 🧩 Topic: Top 50 protein targets by activity count

Okay, I understand. You’re working with ChEMBL 35 data, using RDKit for analysis, and facing some errors while running SQL and Python code within an AIMLops folder structure. Let’s break down the problem, provide solutions, and craft the necessary code.

**1. Analysis of the Analysis Model & Approach**

Your project seems to be centered around analyzing ChEMBL data. A typical drug R&D analysis model would involve:

* **Data Extraction and Cleaning:** Retrieving relevant data from the ChEMBL database and addressing issues like missing values, incorrect data types, and inconsistent units. This is where your SQL scripts come in.
* **Feature Engineering:** Generating molecular descriptors using RDKit. These descriptors represent various physicochemical properties of the molecules (e.g., molecular weight, logP, hydrogen bond donors/acceptors).
* **Exploratory Data Analysis (EDA):** Understanding the data distribution, identifying potential correlations between features, and visualizing the data.
* **Model Building (Optional):** If your goal is predictive, you might build models to predict activity based on the molecular descriptors. Common models include linear regression, random forests, or neural networks. However, let’s focus on simpler descriptive analysis for this demonstration.
* **Interpretation and Reporting:** Drawing conclusions from the analysis and presenting them in a clear and concise manner.

**2. Addressing 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 (~) on a numeric column (act.standard\_value) in PostgreSQL. The ~ operator in PostgreSQL is used for pattern matching on strings, not numbers.
* **Solution:** Instead of using a regular expression, use numeric comparison operators (>, <, =, etc.) or cast the value to text if you absolutely need regex matching.
* **Error b: old scikit-learn version does not support parameters squared=False in the mean\_squared\_error function**
* This means you’re using an older version of scikit-learn.
* **Solution:**
  + **Option 1 (Recommended):** Update your scikit-learn version to the latest stable release using pip install -U scikit-learn.
  + **Option 2 (If updating is not possible):** Remove the squared=False argument from the mean\_squared\_error function call. This will return the Mean Squared Error (MSE) instead of the Root Mean Squared Error (RMSE). If you need RMSE, calculate it manually by taking the square root of the MSE.

**3. AIMLops Folder Structure & Code Execution**

Assuming a basic AIMLops structure like this:

project\_root/  
├── data/  
├── notebooks/  
│ ├── Topic\_CheMBL\_35\_21\_1\_data\_extraction.ipynb  
│ ├── Topic\_CheMBL\_35\_21\_2\_data\_analysis.ipynb  
├── scripts/  
├── models/  
├── README.md

**4. Code (SQL & Python)**

* **SQL (data/chembl\_35\_21\_activity\_data.csv)**

-- File: data/chembl\_35\_21\_activity\_data.sql  
  
SELECT  
 cmp.chembl\_id,  
 act.standard\_type,  
 act.standard\_value,  
 act.standard\_units,  
 act.assay\_chembl\_id  
FROM  
 activities act  
JOIN  
 molecule\_dictionary cmp ON act.molregno = cmp.molregno  
WHERE  
 act.standard\_type = 'IC50' -- Focus on IC50 values  
 AND act.standard\_units = 'nM' -- Focus on nM units  
 AND act.standard\_value > 0 -- Ensure positive values  
 AND act.standard\_value < 100000 --Filter out extremely high values  
LIMIT 100;  
COPY (  
SELECT  
 cmp.chembl\_id,  
 act.standard\_type,  
 act.standard\_value,  
 act.standard\_units,  
 act.assay\_chembl\_id  
FROM  
 activities act  
JOIN  
 molecule\_dictionary cmp ON act.molregno = cmp.molregno  
WHERE  
 act.standard\_type = 'IC50' -- Focus on IC50 values  
 AND act.standard\_units = 'nM' -- Focus on nM units  
 AND act.standard\_value > 0 -- Ensure positive values  
 AND act.standard\_value < 100000 --Filter out extremely high values  
LIMIT 100  
) TO '/tmp/chembl\_35\_21\_activity\_data.csv' WITH CSV HEADER;

**Explanation:**

* This SQL script retrieves activity data (specifically IC50 values in nM) along with the ChEMBL ID of the compound.
* It joins the activities and molecule\_dictionary tables.
* It filters for positive standard\_value and values less than 100000 to exclude unreasonably high values that are often errors or inactive compounds.
* LIMIT 100 ensures only 100 rows are retrieved.
* The COPY command saves the result to a CSV file. **Important:** Adjust the path (/tmp/chembl\_35\_21\_activity\_data.csv) to a location accessible by your PostgreSQL server user. You might need to grant permissions to the PostgreSQL user to write to that directory. Also, ensure the directory exists. After the file is created, move it to the data/ directory within your project.

**Steps to run the .sql script:**

1. Open pgAdmin and connect to your chembl\_35 database on 192.168.206.136.
2. Open a new query window.
3. Copy and paste the SQL script into the query window.
4. Execute the query.
5. Move the generated /tmp/chembl\_35\_21\_activity\_data.csv file (or the location you specified in the COPY command) to your data/ directory.

* **Python (notebooks/Topic\_CheMBL\_35\_21\_1\_data\_extraction.ipynb)**

This notebook is responsible for reading the SQL output.

# File: notebooks/Topic\_CheMBL\_35\_21\_1\_data\_extraction.ipynb  
  
import os  
import pandas as pd  
  
# Define the base path  
base\_path = os.getcwd() # Assuming the notebook is run from the project root.  
data\_path = os.path.join(base\_path, 'data')  
csv\_file\_path = os.path.join(data\_path, 'chembl\_35\_21\_activity\_data.csv')  
  
# Read the CSV file into a Pandas DataFrame  
try:  
 df = pd.read\_csv(csv\_file\_path)  
 print("Data loaded successfully!")  
 print(df.head()) # Display the first few rows  
except FileNotFoundError:  
 print(f"Error: CSV file not found at {csv\_file\_path}")  
except Exception as e:  
 print(f"An error occurred: {e}")

* **Python (notebooks/Topic\_CheMBL\_35\_21\_2\_data\_analysis.ipynb)**

This notebook performs the analysis.

# File: notebooks/Topic\_CheMBL\_35\_21\_2\_data\_analysis.ipynb  
  
import os  
import pandas as pd  
from rdkit import Chem  
from rdkit.Chem import Descriptors  
import matplotlib.pyplot as plt  
import seaborn as sns  
  
# Define the base path  
base\_path = os.getcwd() # Assuming the notebook is run from the project root.  
data\_path = os.path.join(base\_path, 'data')  
csv\_file\_path = os.path.join(data\_path, 'chembl\_35\_21\_activity\_data.csv')  
  
# Load data  
try:  
 df = pd.read\_csv(csv\_file\_path)  
except FileNotFoundError:  
 print(f"Error: CSV file not found at {csv\_file\_path}")  
 exit() # Stop execution if the file is not found  
  
#Convert IC50 to pIC50  
df['pIC50'] = -np.log10(df['standard\_value']/(10\*\*9)) # Standard value is in nM, convert to Molar.  
  
#RDKit Functions  
  
def smiles\_from\_chembl\_id(chembl\_id):  
 """Fetch SMILES from ChEMBL ID"""  
 try:  
 from chembl\_webresource\_client.new\_client import new\_client  
 molecule = new\_client.molecule  
 res = molecule.get(chembl\_id).to\_dict()  
 return res['molecule\_structures']['canonical\_smiles']  
 except:  
 return None  
  
def generate\_descriptors(smiles):  
 """Generate molecular descriptors using RDKit."""  
 mol = Chem.MolFromSmiles(smiles)  
 if mol:  
 descriptors = {  
 "MolWt": Descriptors.MolWt(mol),  
 "LogP": Descriptors.MolLogP(mol),  
 "HBD": Descriptors.NumHDonors(mol),  
 "HBA": Descriptors.NumHAcceptors(mol),  
 "TPSA": Descriptors.TPSA(mol),  
 }  
 return descriptors  
 else:  
 return None  
  
# Get SMILES and generate descriptors  
  
df['SMILES'] = df['chembl\_id'].apply(smiles\_from\_chembl\_id)  
df = df.dropna(subset=['SMILES']) #drop rows with no SMILES  
  
df['descriptors'] = df['SMILES'].apply(generate\_descriptors)  
df = df.dropna(subset=['descriptors']) #drop rows with no descriptors calculated  
  
#Expand the descriptors dictionary into individual columns.  
  
df = pd.concat([df.drop(['descriptors'], axis=1), df['descriptors'].apply(pd.Series)], axis=1)  
  
# Basic EDA:  
print(df.describe()) #Statistical description  
  
# Visualization (example: pIC50 vs. Molecular Weight)  
plt.figure(figsize=(8, 6))  
sns.scatterplot(x='MolWt', y='pIC50', data=df)  
plt.title('pIC50 vs. Molecular Weight')  
plt.xlabel('Molecular Weight')  
plt.ylabel('pIC50')  
plt.show()  
  
  
#Correlation Heatmap (Example)  
correlation\_matrix = df[['pIC50', 'MolWt', 'LogP', 'HBD', 'HBA', 'TPSA']].corr()  
plt.figure(figsize=(10, 8))  
sns.heatmap(correlation\_matrix, annot=True, cmap="coolwarm")  
plt.title("Correlation Heatmap")  
plt.show()

**Explanation:**

1. **Import Libraries:** Imports necessary libraries (pandas, RDKit, matplotlib, seaborn).
2. **File Path:** Defines the path to the CSV file using os.path.join.
3. **Load Data:** Loads the data from the CSV file using pd.read\_csv.
4. **Convert IC50 to pIC50**: Transforms the IC50 into pIC50 as a measure of potency.
5. **Fetch SMILES and Generate Descriptors:**
   * Defines functions to fetch SMILES strings from ChEMBL IDs using chembl\_webresource\_client. **You’ll need to install this: pip install chembl\_webresource\_client**.
   * Defines a function generate\_descriptors that takes a SMILES string as input and calculates molecular descriptors using RDKit.
   * Applies these functions to create new columns in the DataFrame.
6. **Data Cleaning**: Removes rows where the smiles string could not be found
7. **Expand Descriptors**: Expand the descriptors dictionary into separate columns
8. **Basic EDA and Visualization:**
   * Calculates descriptive statistics using df.describe().
   * Creates a scatter plot of pIC50 vs. Molecular Weight using Seaborn.
   * Calculates and visualizes a correlation heatmap.

**5. Running the Code**

1. **Install Dependencies:** Make sure you have all the necessary libraries installed:

* pip install pandas rdkit matplotlib seaborn chembl\_webresource\_client

1. **Execute SQL:** Run the SQL script in pgAdmin and move the resulting CSV file to the data/ directory. Remember to adjust the path in the COPY command if needed.
2. **Run Notebooks:** Open the notebooks/Topic\_CheMBL\_35\_21\_1\_data\_extraction.ipynb and notebooks/Topic\_CheMBL\_35\_21\_2\_data\_analysis.ipynb notebooks in Jupyter Notebook and execute the cells.

**6. Example Analysis and Output**

The Topic\_CheMBL\_35\_21\_2\_data\_analysis.ipynb notebook, when run, will output:

* **Descriptive Statistics:** A table showing the mean, standard deviation, min, max, etc., for each numerical column (pIC50, MolWt, LogP, etc.).
* **Scatter Plot:** A scatter plot showing the relationship between pIC50 and Molecular Weight.
* **Correlation Heatmap:** A visualization of the correlation between different molecular descriptors and pIC50.

**7. Five Examples**

Here are five examples of analyses you can perform using the code and data:

1. **Distribution of pIC50 Values:** Create a histogram of the pIC50 column to visualize the distribution of activity values. This helps understand the potency range of the compounds in your dataset.

* plt.figure(figsize=(8, 6))  
  sns.histplot(df['pIC50'], kde=True) # kde=True adds a kernel density estimate  
  plt.title('Distribution of pIC50 Values')  
  plt.xlabel('pIC50')  
  plt.ylabel('Frequency')  
  plt.show()

1. **Relationship Between LogP and pIC50:** Generate a scatter plot of LogP (octanol-water partition coefficient, a measure of lipophilicity) vs. pIC50. This can reveal whether more lipophilic compounds tend to be more or less active.

* plt.figure(figsize=(8, 6))  
  sns.scatterplot(x='LogP', y='pIC50', data=df)  
  plt.title('pIC50 vs. LogP')  
  plt.xlabel('LogP')  
  plt.ylabel('pIC50')  
  plt.show()

1. **Box Plots of pIC50 for Different Assay Types:** If your data contains different assay\_chembl\_id, you can create box plots to compare the distribution of pIC50 values across different assays. This can help identify assays where compounds tend to be more potent. (You’ll need to keep the assay\_chembl\_id column in your SQL query and load it into the DataFrame.)

* plt.figure(figsize=(12, 6))  
  sns.boxplot(x='assay\_chembl\_id', y='pIC50', data=df)  
  plt.title('pIC50 Distribution by Assay')  
  plt.xlabel('Assay ChEMBL ID')  
  plt.ylabel('pIC50')  
  plt.xticks(rotation=45, ha='right') # Rotate x-axis labels for readability  
  plt.tight\_layout() #Adjusts plot parameters for a tight layout  
  plt.show()

1. **Calculate and Display Summary Statistics for Key Descriptors:** Calculate the mean, median, and standard deviation for descriptors like Molecular Weight, LogP, HBD, and HBA. This provides a concise summary of the physicochemical properties of the compounds in your dataset.

* descriptor\_stats = df[['MolWt', 'LogP', 'HBD', 'HBA', 'TPSA']].describe().loc[['mean', 'std', '50%']] # 50% is the median  
  print(descriptor\_stats)

1. **Correlation Analysis with Visual Annotations:** Extend the correlation heatmap by adding annotations to show the actual correlation values and highlighting statistically significant correlations.

* import numpy as np  
  from scipy.stats import pearsonr  
    
  def calculate\_p\_values(df):  
   df = df.dropna().\_get\_numeric\_data()  
   dfcols = pd.DataFrame(columns=df.columns)  
   pvalues = dfcols.transpose().join(dfcols, how='outer')  
   for r in df.columns:  
   for c in df.columns:  
   if r == c:  
   pvalues[r][c] = np.nan  
   else:  
   pvalues[r][c] = pearsonr(df[r], df[c])[1]  
   return pvalues  
    
  correlation\_matrix = df[['pIC50', 'MolWt', 'LogP', 'HBD', 'HBA', 'TPSA']].corr()  
  p\_values = calculate\_p\_values(df[['pIC50', 'MolWt', 'LogP', 'HBD', 'HBA', 'TPSA']])  
    
  plt.figure(figsize=(12, 10))  
  sns.heatmap(correlation\_matrix, annot=True, fmt=".2f", cmap="coolwarm", mask=np.triu(np.ones\_like(correlation\_matrix, dtype=bool)))  
    
  for text in plt.gca().texts:  
   value = float(text.get\_text())  
   if abs(value) >= 0.5:  
   text.set\_fontsize(12) #Increase font size for strong correlations  
   else:  
   text.set\_fontsize(8) # Decrease font size for weak correlations  
    
  plt.title("Correlation Heatmap with Annotations")  
  plt.show()

**Important Considerations:**

* **ChEMBL Web Services:** The chembl\_webresource\_client relies on the ChEMBL web services. Ensure these services are available and that your network configuration allows access.
* **Error Handling:** The code includes basic error handling (e.g., checking for file existence). Add more robust error handling as needed.
* **Data Cleaning:** The data cleaning steps in the SQL script are minimal. You might need to add more cleaning steps depending on the nature of your data (e.g., handling missing values, converting units).
* **RDKit Installation:** Make sure RDKit is installed correctly and accessible in your Jupyter Notebook environment.
* **Performance:** For larger datasets, consider using more efficient data processing techniques (e.g., using Dask or Spark for parallel processing).
* **Security:** Be mindful of security best practices when handling database credentials. Avoid hardcoding passwords directly in your code. Use environment variables or a configuration file to store sensitive information.
* **Units**: Pay special attention to units in ChEMBL. Always confirm that you are operating on the correct scale. The code converts nM to Molar during pIC50 calculation, but ensure the units are consistent throughout your analysis.

**Traducción al español:**

Entiendo. Estás trabajando con datos de ChEMBL 35, usando RDKit para el análisis y enfrentando algunos errores al ejecutar código SQL y Python dentro de una estructura de carpetas AIMLops. Analicemos el problema, proporcionemos soluciones y creemos el código necesario.

**1. Análisis del modelo de análisis y enfoque**

Tu proyecto parece centrarse en el análisis de datos de ChEMBL. Un modelo típico de análisis de I+D de fármacos implicaría:

* **Extracción y limpieza de datos:** Recuperar datos relevantes de la base de datos ChEMBL y abordar problemas como valores faltantes, tipos de datos incorrectos y unidades inconsistentes. Aquí es donde entran en juego tus scripts SQL.
* **Ingeniería de características:** Generar descriptores moleculares utilizando RDKit. Estos descriptores representan varias propiedades fisicoquímicas de las moléculas (por ejemplo, peso molecular, logP, donantes/aceptores de enlaces de hidrógeno).
* **Análisis exploratorio de datos (EDA):** Comprender la distribución de los datos, identificar posibles correlaciones entre las características y visualizar los datos.
* **Construcción de modelos (opcional):** Si tu objetivo es predictivo, podrías construir modelos para predecir la actividad basándote en los descriptores moleculares. Los modelos comunes incluyen la regresión lineal, los bosques aleatorios o las redes neuronales. Sin embargo, centrémonos en un análisis descriptivo más sencillo para esta demostración.
* **Interpretación e informes:** Sacar conclusiones del análisis y presentarlas de forma clara y concisa.

**2. Abordar los errores**

* **Error a: ERROR: operator does not exist: numeric ~ unknown, LINE 12: AND act.standard\_value ~ '^[0-9\.]+$'**
* Este error indica que estás intentando utilizar una expresión regular (~) en una columna numeric (act.standard\_value) en PostgreSQL. El operador ~ en PostgreSQL se utiliza para la coincidencia de patrones en cadenas, no en números.
* **Solución:** En lugar de utilizar una expresión regular, utiliza operadores de comparación numérica (>, <, =, etc.) o convierte el valor a texto si realmente necesitas la coincidencia regex.
* **Error b: old scikit-learn version does not support parameters squared=False in the mean\_squared\_error function**
* Esto significa que estás utilizando una versión anterior de scikit-learn.
* **Solución:**
  + **Opción 1 (Recomendada):** Actualiza tu versión de scikit-learn a la última versión estable utilizando pip install -U scikit-learn.
  + **Opción 2 (Si la actualización no es posible):** Elimina el argumento squared=False de la llamada a la función mean\_squared\_error. Esto devolverá el Error Cuadrático Medio (MSE) en lugar de la Raíz del Error Cuadrático Medio (RMSE). Si necesitas RMSE, calcúlalo manualmente tomando la raíz cuadrada del MSE.

**3. Estructura de carpetas AIMLops y ejecución de código**

Suponiendo una estructura AIMLops básica como esta:

project\_root/  
├── data/  
├── notebooks/  
│ ├── Topic\_CheMBL\_35\_21\_1\_data\_extraction.ipynb  
│ ├── Topic\_CheMBL\_35\_21\_2\_data\_analysis.ipynb  
├── scripts/  
├── models/  
├── README.md

**4. Código (SQL y Python)**

* **SQL (data/chembl\_35\_21\_activity\_data.csv)**

-- Archivo: data/chembl\_35\_21\_activity\_data.sql  
  
SELECT  
 cmp.chembl\_id,  
 act.standard\_type,  
 act.standard\_value,  
 act.standard\_units,  
 act.assay\_chembl\_id  
FROM  
 activities act  
JOIN  
 molecule\_dictionary cmp ON act.molregno = cmp.molregno  
WHERE  
 act.standard\_type = 'IC50' -- Centrarse en los valores de IC50  
 AND act.standard\_units = 'nM' -- Centrarse en las unidades de nM  
 AND act.standard\_value > 0 -- Asegurarse de que los valores sean positivos  
 AND act.standard\_value < 100000 --Filtrar los valores extremadamente altos  
LIMIT 100;  
  
COPY (  
SELECT  
 cmp.chembl\_id,  
 act.standard\_type,  
 act.standard\_value,  
 act.standard\_units,  
 act.assay\_chembl\_id  
FROM  
 activities act  
JOIN  
 molecule\_dictionary cmp ON act.molregno = cmp.molregno  
WHERE  
 act.standard\_type = 'IC50' -- Centrarse en los valores de IC50  
 AND act.standard\_units = 'nM' -- Centrarse en las unidades de nM  
 AND act.standard\_value > 0 -- Asegurarse de que los valores sean positivos  
 AND act.standard\_value < 100000 --Filtrar los valores extremadamente altos  
LIMIT 100  
) TO '/tmp/chembl\_35\_21\_activity\_data.csv' WITH CSV HEADER;

**Explicación:**

* Este script SQL recupera datos de actividad (específicamente valores de IC50 en nM) junto con el ID de ChEMBL del compuesto.
* Une las tablas activities y molecule\_dictionary.
* Filtra por standard\_value positivos y valores menores de 100000 para excluir valores irrazonablemente altos que a menudo son errores o compuestos inactivos.
* LIMIT 100 garantiza que solo se recuperen 100 filas.
* El comando COPY guarda el resultado en un archivo CSV. **Importante:** Ajusta la ruta (/tmp/chembl\_35\_21\_activity\_data.csv) a una ubicación accesible para el usuario de tu servidor PostgreSQL. Es posible que debas otorgar permisos al usuario de PostgreSQL para escribir en ese directorio. Además, asegúrate de que el directorio exista. Después de crear el archivo, muévelo al directorio data/ dentro de tu proyecto.

**Pasos para ejecutar el script .sql:**

1. Abre pgAdmin y conéctate a tu base de datos chembl\_35 en 192.168.206.136.
2. Abre una nueva ventana de consulta.
3. Copia y pega el script SQL en la ventana de consulta.
4. Ejecuta la consulta.
5. Mueve el archivo generado /tmp/chembl\_35\_21\_activity\_data.csv (o la ubicación que especificaste en el comando COPY) a tu directorio data/.

* **Python (notebooks/Topic\_CheMBL\_35\_21\_1\_data\_extraction.ipynb)**

Este notebook es responsable de leer la salida de SQL.

# Archivo: notebooks/Topic\_CheMBL\_35\_21\_1\_data\_extraction.ipynb  
  
import os  
import pandas as pd  
  
# Define la ruta base  
base\_path = os.getcwd() # Suponiendo que el notebook se ejecuta desde la raíz del proyecto.  
data\_path = os.path.join(base\_path, 'data')  
csv\_file\_path = os.path.join(data\_path, 'chembl\_35\_21\_activity\_data.csv')  
  
# Lee el archivo CSV en un DataFrame de Pandas  
try:  
 df = pd.read\_csv(csv\_file\_path)  
 print("¡Datos cargados correctamente!")  
 print(df.head()) # Muestra las primeras filas  
except FileNotFoundError:  
 print(f"Error: No se encontró el archivo CSV en {csv\_file\_path}")  
except Exception as e:  
 print(f"Se produjo un error: {e}")

* **Python (notebooks/Topic\_CheMBL\_35\_21\_2\_data\_analysis.ipynb)**

Este notebook realiza el análisis.

# Archivo: notebooks/Topic\_CheMBL\_35\_21\_2\_data\_analysis.ipynb  
  
import os  
import pandas as pd  
from rdkit import Chem  
from rdkit.Chem import Descriptors  
import matplotlib.pyplot as plt  
import seaborn as sns  
import numpy as np #Import numpy  
  
# Define la ruta base  
base\_path = os.getcwd() # Suponiendo que el notebook se ejecuta desde la raíz del proyecto.  
data\_path = os.path.join(base\_path, 'data')  
csv\_file\_path = os.path.join(data\_path, 'chembl\_35\_21\_activity\_data.csv')  
  
# Carga los datos  
try:  
 df = pd.read\_csv(csv\_file\_path)  
except FileNotFoundError:  
 print(f"Error: No se encontró el archivo CSV en {csv\_file\_path}")  
 exit() # Detiene la ejecución si no se encuentra el archivo  
  
# Convierte IC50 a pIC50  
df['pIC50'] = -np.log10(df['standard\_value']/(10\*\*9)) # El valor estándar está en nM, se convierte a Molar.  
  
# Funciones de RDKit  
  
def smiles\_from\_chembl\_id(chembl\_id):  
 """Obtener SMILES del ID de ChEMBL"""  
 try:  
 from chembl\_webresource\_client.new\_client import new\_client  
 molecule = new\_client.molecule  
 res = molecule.get(chembl\_id).to\_dict()  
 return res['molecule\_structures']['canonical\_smiles']  
 except:  
 return None  
  
def generate\_descriptors(smiles):  
 """Genera descriptores moleculares utilizando RDKit."""  
 mol = Chem.MolFromSmiles(smiles)  
 if mol:  
 descriptors = {  
 "MolWt": Descriptors.MolWt(mol),  
 "LogP": Descriptors.MolLogP(mol),  
 "HBD": Descriptors.NumHDonors(mol),  
 "HBA": Descriptors.NumHAcceptors(mol),  
 "TPSA": Descriptors.TPSA(mol),  
 }  
 return descriptors  
 else:  
 return None  
  
# Obtiene SMILES y genera descriptores  
  
df['SMILES'] = df['chembl\_id'].apply(smiles\_from\_chembl\_id)  
df = df.dropna(subset=['SMILES']) #Elimina las filas sin SMILES  
  
df['descriptors'] = df['SMILES'].apply(generate\_descriptors)  
df = df.dropna(subset=['descriptors']) #Elimina las filas sin descriptores calculados  
  
#Expande el diccionario de descriptores en columnas individuales.  
  
df = pd.concat([df.drop(['descriptors'], axis=1), df['descriptors'].apply(pd.Series)], axis=1)  
  
# EDA básico:  
print(df.describe()) # Descripción estadística  
  
# Visualización (ejemplo: pIC50 vs. Peso Molecular)  
plt.figure(figsize=(8, 6))  
sns.scatterplot(x='MolWt', y='pIC50', data=df)  
plt.title('pIC50 vs. Peso Molecular')  
plt.xlabel('Peso Molecular')  
plt.ylabel('pIC50')  
plt.show()  
  
# Mapa de calor de correlación (Ejemplo)  
correlation\_matrix = df[['pIC50', 'MolWt', 'LogP', 'HBD', 'HBA', 'TPSA']].corr()  
plt.figure(figsize=(10, 8))  
sns.heatmap(correlation\_matrix, annot=True, cmap="coolwarm")  
plt.title("Mapa de calor de correlación")  
plt.show()

**Explicación:**

1. **Importa bibliotecas:** Importa las bibliotecas necesarias (pandas, RDKit, matplotlib, seaborn).
2. **Ruta del archivo:** Define la ruta al archivo CSV utilizando os.path.join.
3. **Carga datos:** Carga los datos desde el archivo CSV utilizando pd.read\_csv.
4. **Convierte IC50 a pIC50:** Transforma el IC50 en pIC50 como una medida de la potencia.
5. **Obtiene SMILES y genera descriptores:**
   * Define funciones para obtener cadenas SMILES de los ID de ChEMBL utilizando chembl\_webresource\_client. **Deberás instalar esto: pip install chembl\_webresource\_client**.
   * Define una función generate\_descriptors que toma una cadena SMILES como entrada y calcula descriptores moleculares utilizando RDKit.
   * Aplica estas funciones para crear nuevas columnas en el DataFrame.
6. **Limpieza de datos**: Elimina las filas donde no se pudo encontrar la cadena SMILES
7. **Expande descriptores**: Expande el diccionario de descriptores en columnas separadas.
8. **EDA básico y visualización:**
   * Calcula estadísticas descriptivas utilizando df.describe().
   * Crea un diagrama de dispersión de pIC50 vs. Peso Molecular utilizando Seaborn.
   * Calcula y visualiza un mapa de calor de correlación.

**5. Ejecutar el código**

1. **Instala dependencias:** Asegúrate de tener todas las bibliotecas necesarias instaladas:

* pip install pandas rdkit matplotlib seaborn chembl\_webresource\_client

1. **Ejecuta SQL:** Ejecuta el script SQL en pgAdmin y mueve el archivo CSV resultante al directorio data/. Recuerda ajustar la ruta en el comando COPY si es necesario.
2. **Ejecuta los notebooks:** Abre los notebooks notebooks/Topic\_CheMBL\_35\_21\_1\_data\_extraction.ipynb y notebooks/Topic\_CheMBL\_35\_21\_2\_data\_analysis.ipynb en Jupyter Notebook y ejecuta las celdas.

**6. Ejemplo de análisis y salida**

El notebook Topic\_CheMBL\_35\_21\_2\_data\_analysis.ipynb, cuando se ejecuta, mostrará:

* **Estadísticas descriptivas:** Una tabla que muestra la media, la desviación estándar, el mínimo, el máximo, etc., para cada columna numérica (pIC50, MolWt, LogP, etc.).
* **Diagrama de dispersión:** Un diagrama de dispersión que muestra la relación entre pIC50 y el peso molecular.
* **Mapa de calor de correlación:** Una visualización de la correlación entre diferentes descriptores moleculares y pIC50.

**7. Cinco ejemplos**

Aquí hay cinco ejemplos de análisis que puedes realizar utilizando el código y los datos:

1. **Distribución de valores de pIC50:** Crea un histograma de la columna pIC50 para visualizar la distribución de los valores de actividad. Esto ayuda a comprender el rango de potencia de los compuestos en tu conjunto de datos.

* plt.figure(figsize=(8, 6))  
  sns.histplot(df['pIC50'], kde=True) # kde=True añade una estimación de densidad del núcleo  
  plt.title('Distribución de los valores de pIC50')  
  plt.xlabel('pIC50')  
  plt.ylabel('Frecuencia')  
  plt.show()

1. **Relación entre LogP y pIC50:** Genera un diagrama de dispersión de LogP (coeficiente de partición octanol-agua, una medida de la lipofilicidad) vs. pIC50. Esto puede revelar si los compuestos más lipofílicos tienden a ser más o menos activos.

* plt.figure(figsize=(8, 6))  
  sns.scatterplot(x='LogP', y='pIC50', data=df)  
  plt.title('pIC50 vs. LogP')  
  plt.xlabel('LogP')  
  plt.ylabel('pIC50')  
  plt.show()

1. **Diagramas de caja de pIC50 para diferentes tipos de ensayo:** Si tus datos contienen