PharmApp Suite

🧠 AI for Drug Discovery and Development 💉

🧩 Topic: QSAR model building from ChEMBL IC50 data

Okay, I understand. You're working with ChEMBL data to develop drug research insights using RDKit and are following an AIMLops template structure. You have a PostgreSQL database on a remote server and want to perform analysis using Jupyter notebooks. You've encountered a few errors and need help with generating SQL queries and Python code, specifically keeping the data volume small (100 rows) for demonstration purposes.

Here's a breakdown of how to approach this, addressing your points:

1. Analysis Model (Clear Explanation):

We'll focus on basic but essential data exploration and analysis within the ChEMBL dataset. The core idea is to extract relevant information about compounds and their activities, perform initial data cleaning, and explore relationships between compound properties and activity measurements. We will cover the following key areas:

- **Data Extraction:** Querying the ChEMBL database to retrieve compound structures (SMILES) and activity data (e.g., IC50 values) for specific targets or activity types.
- **Data Cleaning:** Filtering and standardizing the extracted data. This includes handling missing values, converting activity units, and filtering out data with low confidence scores. We'll specifically address the error you encountered related to numeric type matching.
- **Data Conversion:** Convert SMILES to Molecular objects with RDKit.
- **Basic Analysis:** This includes calculating simple statistical summaries of activity data, visualizing activity distributions, and looking for correlations between different compound properties.

2. Code (SQL & Python):

SQL (To be run in pgAdmin to generate `Topic_CheMBL_35_1.csv`):

```sql
-- File: Topic_CheMBL_35_1.sql
-- Purpose: Extract ChEMBL data for analysis (limited to 100 rows)
-- Server IP: 192.168.206.136
-- User: rd
-- Pass: rd
-- Database: chembl_35

-- Modified query to address the numeric ~ unknown error and limit the result

SELECT
    md.chembl_id,
    cs.canonical_smiles,
    act.standard_type,
    act.standard_value,
    act.standard_units,
    act.assay_id
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 IN ('IC50', 'Ki', 'EC50')  -- Example activity types
    AND act.standard_relation = '='  -- Filter for direct measurements
    AND act.standard_value IS NOT NULL
    AND act.standard_value::text ~ '^[0-9\.]+$' -- Check for numeric values using
explicit type casting
    AND act.standard_units = 'nM'
LIMIT 100;

-- Save the result as Topic_CheMBL_35_1.csv (using pgAdmin's export feature)
```

Explanation:

- **`SELECT ... FROM ...`**: Selects relevant columns from the `molecule_dictionary`, `compound_structures`, and `activities` tables.
- **`JOIN ... ON ...`**: Joins the tables based on common molecule identifiers (`molregno`).
- **`WHERE ...`**:
    - `act.standard_type IN ('IC50', 'Ki', 'EC50')`: Filters for specific activity types. You can customize this list.
    - `act.standard_relation = '='`: Ensures we only retrieve data where the relationship is a direct equality. This is more reliable than inequalities.
    - `act.standard_value IS NOT NULL`: Excludes rows with missing activity values.
    - `act.standard_value::text ~ '^[0-9\.]+$'`: **Crucially, this addresses your error**. It explicitly casts the `standard_value` to text (`::text`) before applying the regular expression `^[0-9\.]+$`. This ensures that the comparison is performed on a text representation of the value.
    - `act.standard_units = 'nM'`: Filters for nanomolar units (a common unit for activity).
- **`LIMIT 100`**: Limits the result set to 100 rows. **Important for keeping the data size small.**

Python (Jupyter Notebook: `Topic_CheMBL_35_1_DataAnalysis.ipynb`):

```
# File: Topic_CheMBL_35_1_DataAnalysis.ipynb
import os
import pandas as pd
from rdkit import Chem
from rdkit.Chem import Descriptors
import numpy as np  # Import NumPy
from sklearn.metrics import mean_squared_error
import matplotlib.pyplot as plt
import seaborn as sns

# Define the base path (adjust this to your actual base path)
base_path = "../data"  # Example: Assuming 'data' folder is one level above the
notebook

# Construct the file path to your CSV file
csv_file_path = os.path.join(base_path, "Topic_CheMBL_35_1.csv")

# Load the data from the CSV file
try:
    df = pd.read_csv(csv_file_path)
    print("Data loaded successfully!")
except FileNotFoundError:
    print(f"Error: File not found at {csv_file_path}.  Please check the path.")
    exit()

# Data Cleaning and Preprocessing
print("\nData Cleaning and Preprocessing...")
```

```
# Remove rows with missing SMILES strings
df = df.dropna(subset=['canonical_smiles'])
df = df.dropna(subset=['standard_value'])
print("Number of compounds remaining after cleaning:", len(df))

# Convert SMILES to RDKit Mol objects
df['mol'] = df['canonical_smiles'].apply(lambda x: Chem.MolFromSmiles(x))
df = df.dropna(subset=['mol'])

# Calculate Molecular Weight
df['mol_weight'] = df['mol'].apply(Descriptors.MolWt)

# Display the first few rows of the DataFrame
print("\nFirst 5 rows of the processed data:")
print(df.head())

# Basic Analysis: Distribution of Molecular Weights and Activities
print("\nBasic Analysis: Distribution of Molecular Weights and Activities...")

# Plot Molecular Weight Distribution
plt.figure(figsize=(8, 6))
sns.histplot(df['mol_weight'], kde=True)
plt.title('Molecular Weight Distribution')
plt.xlabel('Molecular Weight')
plt.ylabel('Frequency')
plt.show()

# Plot Activity Distribution (log scale for better visualization)
plt.figure(figsize=(8, 6))
sns.histplot(np.log10(df['standard_value'].astype(float)), kde=True)
plt.title('Distribution of log10(IC50/Ki/EC50) (nM)')  # Explicitly state units
plt.xlabel('log10(Activity (nM))')
plt.ylabel('Frequency')
plt.show()

# Example: Simple Linear Regression (Molecular Weight vs. Activity)
print("\nExample: Simple Linear Regression (Molecular Weight vs. Activity)...")
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression

# Prepare data for regression
X = df[['mol_weight']]
y = np.log10(df['standard_value'].astype(float))

# Handle infinite values and NaN values correctly
X = X[np.isfinite(y)]
y = y[np.isfinite(y)]

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

# Make predictions on the test set
y_pred = model.predict(X_test)
```

```
# Evaluate the model
mse = mean_squared_error(y_test, y_pred)  # No squared=False needed in newer scikit-learn
print(f'Mean Squared Error: {mse}')

# Plot predicted vs. actual values
plt.figure(figsize=(8, 6))
plt.scatter(y_test, y_pred)
plt.xlabel('Actual log10(Activity)')
plt.ylabel('Predicted log10(Activity)')
plt.title('Actual vs. Predicted log10(Activity)')
plt.show()


print("\nAnalysis complete.")
```

Explanation:

- **Import Libraries:** Imports necessary libraries (pandas, RDKit, scikit-learn, matplotlib, seaborn).
- **File Path Handling:** Uses `os.path.join` to create the file path. **This is essential for the AIMLops structure.** Make sure `base_path` is correctly set.
- **Data Loading:** Loads the CSV data into a pandas DataFrame.
- **Data Cleaning:** Removes rows with missing SMILES or activity values using `dropna()`.
- **SMILES Conversion:** Converts SMILES strings to RDKit `Mol` objects. Rows where conversion fails are dropped.
- **Feature Calculation:** Calculates molecular weight using `Descriptors.MolWt`.
- **Basic Analysis:**
  - Plots the distribution of molecular weights using `sns.histplot`.
  - Plots the distribution of activity values (log-transformed) for better visualization.
- **Simple Linear Regression:**
  - Splits the data into training and testing sets.
  - Creates and trains a `LinearRegression` model.
  - Makes predictions on the test set.
  - Evaluates the model using `mean_squared_error`. Note that the `squared=False` parameter is **no longer required** in newer versions of scikit-learn. Remove it.
  - Plots predicted vs. actual values.
- **Error Handling:** Uses `try...except` to handle potential `FileNotFoundError` if the CSV file is not found.
- **NumPy for Data Handling:** Added import of `numpy as np` and usage of `np.isfinite()` function for handling infinite values and NaN values in the standard_value column during the regression calculation.

Addressing Your Errors:

- **Error a (Operator does not exist: numeric ~ unknown):** The SQL query includes `act.standard_value::text ~ '^[0-9\.]+$'` to convert the numeric column to text for the regex comparison. This should resolve the issue.
- **Error b (old scikit-learn version):** Remove the `squared=False` parameter from `mean_squared_error`. Newer versions of scikit-learn default to returning the mean squared error (MSE) without the square root.

3. Examples (5 Examples of Usage):

Here are 5 examples of how you can extend this code for more detailed analysis. Each example adds a specific functionality:

Example 1: Filtering by Target Organism:

```
# (Add this to the Python code, after loading the data)
# Example 1: Filtering by Target Organism
from rdkit import Chem
from rdkit.Chem import Descriptors
from rdkit.Chem import Lipinski

# Assuming you have target information in the 'assay_id' column
# In a real ChEMBL workflow, you would typically join the activities table with the
target_dictionary table

target_organism = 'Homo sapiens' # Set your desired target organism

#Create a mock to map assay id to target organisms
assay_to_target = {
    1: 'Homo sapiens',
    2: 'Mus musculus',
    3: 'Rattus norvegicus',
    4: 'Homo sapiens',
    5: 'Other',
}
# Create a new column called 'target_organism' using the mock
df['target_organism'] = df['assay_id'].map(assay_to_target)

df_filtered = df[df['target_organism'] == target_organism].copy()
print(f"Number of compounds targeting {target_organism}: {len(df_filtered)}")

# Proceed with analysis using df_filtered
# For example, calculate the average molecular weight for compounds targeting Homo
sapiens:
avg_mol_weight = df_filtered['mol_weight'].mean()
print(f"Average molecular weight of compounds targeting {target_organism}:
{avg_mol_weight}")
```

Explanation: This example demonstrates how to filter the DataFrame based on the `target_organism`. In a real ChEMBL database, you'd perform a SQL join between the `activities` and `target_dictionary` tables to get target organism information. This example uses a mock mapping for demonstration.

Example 2: Lipinski's Rule of Five Analysis:

```
# (Add this to the Python code, after loading the data and converting SMILES)
# Example 2: Lipinski's Rule of Five Analysis

def lipinski_properties(mol):
    """Calculates Lipinski's Rule of Five properties."""
    mw = Descriptors.MolWt(mol)
    logp = Descriptors.MolLogP(mol)
    hbd = Descriptors.NumHDonors(mol)
    hba = Descriptors.NumHAcceptors(mol)
    return mw, logp, hbd, hba

df[['mol_weight', 'logP', 'HBD', 'HBA']] = df['mol'].apply(lambda x:
pd.Series(lipinski_properties(x)))

def lipinski_rule(row):
```

```
    """Checks if a molecule violates Lipinski's Rule of Five."""
    violations = 0
    if row['mol_weight'] > 500:
        violations += 1
    if row['logP'] > 5:
        violations += 1
    if row['HBD'] > 5:
        violations += 1
    if row['HBA'] > 10:
        violations += 1
    return violations

df['Lipinski_Violations'] = df.apply(lipinski_rule, axis=1)

print(df[['canonical_smiles', 'mol_weight', 'logP', 'HBD', 'HBA',
'Lipinski_Violations']].head())

# Analyze the distribution of Lipinski violations
violation_counts = df['Lipinski_Violations'].value_counts().sort_index()
print("\nDistribution of Lipinski Violations:")
print(violation_counts)

plt.figure(figsize=(8, 6))
violation_counts.plot(kind='bar')
plt.title('Distribution of Lipinski Rule Violations')
plt.xlabel('Number of Violations')
plt.ylabel('Number of Compounds')
plt.show()
```

Explanation: This example calculates Lipinski's Rule of Five properties (molecular weight, logP, H-bond donors, H-bond acceptors) and determines the number of violations for each compound. It then analyzes the distribution of violations.

Example 3: Activity Cliff Detection (Requires more data for meaningful results):

```
# (Add this to the Python code, after loading the data and converting SMILES)
# Example 3: Activity Cliff Detection (Requires more data for meaningful results)

from rdkit import DataStructs
from rdkit.Chem.Fingerprints import FingerprintMols

# Generate Morgan fingerprints (ECFP4)
df['fingerprint'] = df['mol'].apply(lambda x: FingerprintMols.FingerprintMol(x))

def calculate_tanimoto_coefficient(fp1, fp2):
    """Calculates the Tanimoto coefficient between two fingerprints."""
    return DataStructs.TanimotoSimilarity(fp1, fp2)

# Activity cliff detection (simplified)
# Requires a larger dataset for robust results!

activity_cliff_cutoff = 1  # Example: log10 activity difference cutoff (adjust as
needed)
tanimoto_cutoff = 0.8 # Example: Tanimoto coefficient cutoff (adjust as needed)

activity_cliffs = []
for i in range(len(df)):
    for j in range(i + 1, len(df)):
        tanimoto_similarity =
calculate_tanimoto_coefficient(df['fingerprint'].iloc[i], df['fingerprint'].iloc[j])
```

```
        activity_difference = abs(np.log10(df['standard_value'].iloc[i]) -
np.log10(df['standard_value'].iloc[j]))
        if tanimoto_similarity >= tanimoto_cutoff and activity_difference >=
activity_cliff_cutoff:
            activity_cliffs.append((df['chembl_id'].iloc[i], df['chembl_id'].iloc[j],
tanimoto_similarity, activity_difference))

if activity_cliffs:
    print("\nPotential Activity Cliffs:")
    for cliff in activity_cliffs:
        print(f"Compound Pair: {cliff[0]}, {cliff[1]}, Tanimoto Similarity:
{cliff[2]:.2f}, Activity Difference: {cliff[3]:.2f}")
else:
    print("\nNo activity cliffs found (with current cutoffs and data).  Try increasing
the dataset size or adjusting the cutoffs.")
```

Explanation: This example demonstrates a simplified approach to activity cliff detection. It calculates Tanimoto similarity between compound fingerprints and compares the activity difference. Pairs with high similarity and significant activity differences are identified as potential activity cliffs. **Important:** This example requires a much larger dataset for meaningful results. With only 100 rows, you're unlikely to find significant activity cliffs.

Example 4: Substructure Searching:

```
# (Add this to the Python code, after loading the data and converting SMILES)
# Example 4: Substructure Searching

from rdkit.Chem import AllChem

# Define the SMARTS pattern for the substructure you want to search for
substructure_smarts = 'c1ccccc1[N+](=O)[O-]'  # Example: Nitrobenzene

# Create a Mol object from the SMARTS pattern
substructure = Chem.MolFromSmarts(substructure_smarts)

if substructure is None:
    print(f"Invalid SMARTS pattern: {substructure_smarts}")
else:
    # Check if each molecule contains the substructure
    df['contains_substructure'] = df['mol'].apply(lambda x:
x.HasSubstructMatch(substructure))

    # Filter the DataFrame to show only compounds containing the substructure
    substructure_compounds = df[df['contains_substructure']]

    print(f"\nNumber of compounds containing the substructure '{substructure_smarts}':
{len(substructure_compounds)}")

    if not substructure_compounds.empty:
        print("\nCompounds containing the substructure:")
        print(substructure_compounds[['chembl_id', 'canonical_smiles']].head())  #
Show ChEMBL ID and SMILES
```

Explanation: This example demonstrates how to search for a specific substructure within your compounds. You define a SMARTS pattern, create a Mol object from it, and then use `HasSubstructMatch` to check if each molecule contains the substructure.

Example 5: Calculating QED (Quantitative Estimate of Drug-likeness):

```python
# (Add this to the Python code, after loading the data and converting SMILES)
# Example 5: Calculating QED (Quantitative Estimate of Drug-likeness)

from rdkit.Chem import QED

# Calculate QED for each molecule
df['QED'] = df['mol'].apply(QED.qed)

print("\nQED values for the first 5 compounds:")
print(df[['chembl_id', 'canonical_smiles', 'QED']].head())

# Analyze the distribution of QED values
plt.figure(figsize=(8, 6))
sns.histplot(df['QED'], kde=True)
plt.title('Distribution of QED (Quantitative Estimate of Drug-likeness)')
plt.xlabel('QED Value')
plt.ylabel('Frequency')
plt.show()

# Calculate average QED
average_qed = df['QED'].mean()
print(f"\nAverage QED value: {average_qed:.3f}")
```

Explanation: This example demonstrates how to calculate the QED (Quantitative Estimate of Drug-likeness) for each molecule using the `rdkit.Chem.QED` module. It then analyzes the distribution of QED values.

4. Important Considerations:

- **Data Volume:** You're intentionally limiting the data to 100 rows. This is good for initial development, but keep in mind that many analyses (like activity cliff detection) require significantly more data to be meaningful. Scale up the `LIMIT` in your SQL query as your system allows.
- **Error Handling:** Include comprehensive error handling in your Python code (e.g., using `try...except` blocks) to gracefully handle potential issues like invalid SMILES strings, missing data, or database connection errors.
- **Units:** Be very careful about units when working with activity data. Always explicitly state the units (e.g., nM) in your plots and calculations. Convert all activities to a consistent unit before performing comparisons.
- **ChEMBL Version:** Your SQL assumes ChEMBL 35 schema. Adapt the queries if you use a different version.
- **AIMLops Compliance:** Make sure your data, code, and models are version-controlled (e.g., using Git) and follow your organization's AIMLops standards for reproducibility and deployment. The `os.path.join` usage is a good start for directory structure.

In summary, this comprehensive response provides you with a starting point for your ChEMBL 35 data analysis project. Remember to adapt the code and examples to your specific research questions and data. Good luck!

Spanish Translation:

1. Modelo de Análisis (Explicación Clara):

Nos centraremos en la exploración y el análisis de datos básicos pero esenciales dentro del conjunto de datos ChEMBL. La idea central es extraer información relevante sobre los compuestos y sus actividades, realizar una limpieza de datos inicial y explorar las relaciones entre las propiedades de los compuestos y las mediciones de actividad. Cubriremos las siguientes áreas clave:

- **Extracción de Datos:** Consultar la base de datos ChEMBL para recuperar estructuras de compuestos (SMILES) y datos de actividad (por ejemplo, valores de IC50) para objetivos o tipos de actividad específicos.
- **Limpieza de Datos:** Filtrar y estandarizar los datos extraídos. Esto incluye el manejo de valores faltantes, la conversión de unidades de actividad y el filtrado de datos con puntajes de baja confianza. Abordaremos específicamente el error que encontró relacionado con la coincidencia de tipos numéricos.
- **Conversión de Datos:** Convertir SMILES en objetos Moleculares con RDKit.
- **Análisis Básico:** Esto incluye el cálculo de resúmenes estadísticos simples de los datos de actividad, la visualización de distribuciones de actividad y la búsqueda de correlaciones entre diferentes propiedades de los compuestos.

2. Código (SQL & Python):

SQL (Para ejecutar en pgAdmin para generar `Topic_CheMBL_35_1.csv`):

```sql
-- Archivo: Topic_CheMBL_35_1.sql
-- Propósito: Extraer datos de ChEMBL para análisis (limitado a 100 filas)
-- IP del servidor: 192.168.206.136
-- Usuario: rd
-- Contraseña: rd
-- Base de datos: chembl_35

-- Consulta modificada para abordar el error numeric ~ unknown y limitar el resultado

SELECT
    md.chembl_id,
    cs.canonical_smiles,
    act.standard_type,
    act.standard_value,
    act.standard_units,
    act.assay_id
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 IN ('IC50', 'Ki', 'EC50')  -- Tipos de actividad de ejemplo
    AND act.standard_relation = '='  -- Filtrar para mediciones directas
    AND act.standard_value IS NOT NULL
    AND act.standard_value::text ~ '^[0-9\.]+$' -- Verificar valores numéricos usando
conversión de tipo explícita
    AND act.standard_units = 'nM'
LIMIT 100;

-- Guardar el resultado como Topic_CheMBL_35_1.csv (usando la función de exportación
de pgAdmin)
```

Python (Jupyter Notebook: `Topic_CheMBL_35_1_DataAnalysis.ipynb`):

```python
# Archivo: Topic_CheMBL_35_1_DataAnalysis.ipynb
import os
import pandas as pd
from rdkit import Chem
from rdkit.Chem import Descriptors
import numpy as np  # Importar NumPy
from sklearn.metrics import mean_squared_error
import matplotlib.pyplot as plt
```

```
import seaborn as sns

# Define la ruta base (ajusta esto a tu ruta base real)
base_path = "../data"  # Ejemplo: Asumiendo que la carpeta 'data' está un nivel por
encima del notebook

# Construye la ruta del archivo a tu archivo CSV
csv_file_path = os.path.join(base_path, "Topic_CheMBL_35_1.csv")

# Carga los datos desde el archivo CSV
try:
    df = pd.read_csv(csv_file_path)
    print("¡Datos cargados con éxito!")
except FileNotFoundError:
    print(f"Error: No se encontró el archivo en {csv_file_path}. Por favor, verifica
la ruta.")
    exit()

# Limpieza y Preprocesamiento de Datos
print("\nLimpieza y Preprocesamiento de Datos...")

# Elimina filas con cadenas SMILES faltantes
df = df.dropna(subset=['canonical_smiles'])
df = df.dropna(subset=['standard_value'])
print("Número de compuestos restantes después de la limpieza:", len(df))

# Convierte SMILES a objetos Mol de RDKit
df['mol'] = df['canonical_smiles'].apply(lambda x: Chem.MolFromSmiles(x))
df = df.dropna(subset=['mol'])

# Calcula el Peso Molecular
df['mol_weight'] = df['mol'].apply(Descriptors.MolWt)

# Muestra las primeras filas del DataFrame
print("\nPrimeras 5 filas de los datos procesados:")
print(df.head())

# Análisis Básico: Distribución de Pesos Moleculares y Actividades
print("\nAnálisis Básico: Distribución de Pesos Moleculares y Actividades...")

# Grafica la Distribución del Peso Molecular
plt.figure(figsize=(8, 6))
sns.histplot(df['mol_weight'], kde=True)
plt.title('Distribución del Peso Molecular')
plt.xlabel('Peso Molecular')
plt.ylabel('Frecuencia')
plt.show()

# Grafica la Distribución de la Actividad (escala logarítmica para una mejor
visualización)
plt.figure(figsize=(8, 6))
sns.histplot(np.log10(df['standard_value'].astype(float)), kde=True)
plt.title('Distribución de log10(IC50/Ki/EC50) (nM)')  # Indica explícitamente las
unidades
plt.xlabel('log10(Actividad (nM))')
plt.ylabel('Frecuencia')
plt.show()

# Ejemplo: Regresión Lineal Simple (Peso Molecular vs. Actividad)
```

```
print("\nEjemplo: Regresión Lineal Simple (Peso Molecular vs. Actividad)...")
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression

# Prepara los datos para la regresión
X = df[['mol_weight']]
y = np.log10(df['standard_value'].astype(float))

# Maneja valores infinitos y NaN correctamente
X = X[np.isfinite(y)]
y = y[np.isfinite(y)]

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

# Crea y entrena el modelo de regresión lineal
model = LinearRegression()
model.fit(X_train, y_train)

# Realiza predicciones en el conjunto de prueba
y_pred = model.predict(X_test)

# Evalúa el modelo
mse = mean_squared_error(y_test, y_pred)  # No se necesita squared=False en scikit-
learn más reciente
print(f'Error Cuadrático Medio: {mse}')

# Grafica los valores predichos vs. reales
plt.figure(figsize=(8, 6))
plt.scatter(y_test, y_pred)
plt.xlabel('log10(Actividad) Real')
plt.ylabel('log10(Actividad) Predicha')
plt.title('log10(Actividad) Real vs. Predicha')
plt.show()

print("\nAnálisis completo.")
```

3. Ejemplos (5 Ejemplos de Uso):

[Se incluye la traducción al español de los 5 ejemplos de Python, adaptando el texto explicativo cuando sea necesario.]

Example 1: Filtrado por Organismo Objetivo:

```
# (Agrega esto al código Python, después de cargar los datos)
# Ejemplo 1: Filtrado por Organismo Objetivo
from rdkit import Chem
from rdkit.Chem import Descriptors
from rdkit.Chem import Lipinski

# Asumiendo que tienes información del objetivo en la columna 'assay_id'
# En un flujo de trabajo real de ChEMBL, normalmente unirías la tabla activities con
la tabla target_dictionary

target_organism = 'Homo sapiens' # Establece tu organismo objetivo deseado

#Crea una simulación para mapear el id del ensayo a los organismos objetivo
assay_to_target = {
    1: 'Homo sapiens',
```

```
    2: 'Mus musculus',
    3: 'Rattus norvegicus',
    4: 'Homo sapiens',
    5: 'Other',
}
# Crea una nueva columna llamada 'target_organism' usando la simulación
df['target_organism'] = df['assay_id'].map(assay_to_target)


df_filtered = df[df['target_organism'] == target_organism].copy()
print(f"Número de compuestos dirigidos a {target_organism}: {len(df_filtered)}")

# Procede con el análisis utilizando df_filtered
# Por ejemplo, calcula el peso molecular promedio para los compuestos dirigidos a Homo
sapiens:
avg_mol_weight = df_filtered['mol_weight'].mean()
print(f"Peso molecular promedio de los compuestos dirigidos a {target_organism}:
{avg_mol_weight}")
```

Example 2: Análisis de la Regla de los Cinco de Lipinski:

```
# (Agrega esto al código Python, después de cargar los datos y convertir los SMILES)
# Ejemplo 2: Análisis de la Regla de los Cinco de Lipinski

def lipinski_properties(mol):
    """Calcula las propiedades de la Regla de los Cinco de Lipinski."""
    mw = Descriptors.MolWt(mol)
    logp = Descriptors.MolLogP(mol)
    hbd = Descriptors.NumHDonors(mol)
    hba = Descriptors.NumHAcceptors(mol)
    return mw, logp, hbd, hba

df[['mol_weight', 'logP', 'HBD', 'HBA']] = df['mol'].apply(lambda x:
pd.Series(lipinski_properties(x)))

def lipinski_rule(row):
    """Verifica si una molécula viola la Regla de los Cinco de Lipinski."""
    violations = 0
    if row['mol_weight'] > 500:
        violations += 1
    if row['logP'] > 5:
        violations += 1
    if row['HBD'] > 5:
        violations += 1
    if row['HBA'] > 10:
        violations += 1
    return violations

df['Lipinski_Violations'] = df.apply(lipinski_rule, axis=1)

print(df[['canonical_smiles', 'mol_weight', 'logP', 'HBD', 'HBA',
'Lipinski_Violations']].head())

# Analiza la distribución de las violaciones de Lipinski
violation_counts = df['Lipinski_Violations'].value_counts().sort_index()
print("\nDistribución de las Violaciones de Lipinski:")
print(violation_counts)

plt.figure(figsize=(8, 6))
violation_counts.plot(kind='bar')
```

```
plt.title('Distribución de las Violaciones de la Regla de Lipinski')
plt.xlabel('Número de Violaciones')
plt.ylabel('Número de Compuestos')
plt.show()
```

Example 3: Detección de Acantilados de Actividad (Requiere más datos para resultados significativos):

```
# (Agrega esto al código Python, después de cargar los datos y convertir los SMILES)
# Ejemplo 3: Detección de Acantilados de Actividad (Requiere más datos para resultados significativos)

from rdkit import DataStructs
from rdkit.Chem.Fingerprints import FingerprintMols

# Genera huellas dactilares de Morgan (ECFP4)
df['fingerprint'] = df['mol'].apply(lambda x: FingerprintMols.FingerprintMol(x))

def calculate_tanimoto_coefficient(fp1, fp2):
    """Calcula el coeficiente de Tanimoto entre dos huellas dactilares."""
    return DataStructs.TanimotoSimilarity(fp1, fp2)

# Detección de acantilados de actividad (simplificada)
# ¡Requiere un conjunto de datos más grande para obtener resultados robustos!

activity_cliff_cutoff = 1  # Ejemplo: umbral de diferencia de actividad log10 (ajusta según sea necesario)
tanimoto_cutoff = 0.8 # Ejemplo: umbral del coeficiente de Tanimoto (ajusta según sea necesario)

activity_cliffs = []
for i in range(len(df)):
    for j in range(i + 1, len(df)):
        tanimoto_similarity =
calculate_tanimoto_coefficient(df['fingerprint'].iloc[i], df['fingerprint'].iloc[j])
        activity_difference = abs(np.log10(df['standard_value'].iloc[i]) -
np.log10(df['standard_value'].iloc[j]))
        if tanimoto_similarity >= tanimoto_cutoff and activity_difference >=
activity_cliff_cutoff:
            activity_cliffs.append((df['chembl_id'].iloc[i], df['chembl_id'].iloc[j],
tanimoto_similarity, activity_difference))

if activity_cliffs:
    print("\nPosibles Acantilados de Actividad:")
    for cliff in activity_cliffs:
        print(f"Par de Compuestos: {cliff[0]}, {cliff[1]}, Similitud de Tanimoto:
{cliff[2]:.2f}, Diferencia de Actividad: {cliff[3]:.2f}")
else:
    print("\nNo se encontraron acantilados de actividad (con los umbrales y datos
actuales). Intenta aumentar el tamaño del conjunto de datos o ajustar los umbrales.")
```

Example 4: Búsqueda de Subestructuras:

```python # (Agrega esto al código Python, después de cargar los datos y convertir
```