

### Topic: Matched molecular pair analysis from ChEMBL

Okay, I understand. You are working with ChEMBL 35 data using RDKit and are following an AIMLops-inspired directory structure. You need help crafting SQL and Python code to extract data, perform basic analysis, and address the errors you've encountered. I will generate the code and explain the analysis steps. Let's get started!

#### 1. Analysis Model & Goals

The project aims to leverage ChEMBL 35 data, using RDKit for cheminformatics tasks. You'll use SQL to extract relevant information (bioactivities, compounds, etc.) from the database. This extracted data will be loaded into a Pandas DataFrame within a Jupyter notebook. In the notebook, you will then use RDKit to perform operations like generating molecular fingerprints. The analysis model is as follows:

- **Data Extraction (SQL):** Query the ChEMBL 35 database to retrieve a relevant subset of bioactivity data, focusing on compounds with activity data for a specific target or bioassay. Crucially, limit the data to the first 100 rows to manage computational resources.
- **Data Loading & Preprocessing (Python):** Load the CSV data extracted from the SQL query into a Pandas DataFrame.
- **RDKit Integration (Python):** Use RDKit to process the compounds, generating molecular representations (e.g., SMILES strings converted into RDKit molecules).
- **Feature Generation (Python):** Derive features from the molecular representations (e.g., Morgan fingerprints).
- **Basic Analysis (Python):** Perform some example analysis such as:
  - Show number of compounds, number of unique compounds
  - Calculate some statistics of standard values

#### 2. Directory Structure (Implied from AIMLops)

Based on your description, I'm assuming a simplified directory structure like this:

```
project_root/
├── data/
│   └── chembl_bioactivities.csv # Example CSV file
├── notebooks/
│   ├── Topic_CheMBL_35_97_1_data_extraction.ipynb
│   └── Topic_CheMBL_35_97_2_analysis.ipynb
└── src/ # (Optional) Might contain reusable Python modules
```

#### 3. SQL Code (Extracting Bioactivity Data)

Here's the SQL code to extract bioactivity data and save it to a CSV file. This query pulls data from several ChEMBL tables and filters for a specific activity type and reasonable standard value. It also limits the results to the first 100 entries.

-- File: ../data/chembl\_bioactivities.sql

```
SELECT
    act.molregno,
    compd.chembl_id,
    act.standard_type,
    act.standard_value,
```

```

act.standard_units,
act.assay_id,
assays.description,
compd.canonical_smiles
FROM
activities act
JOIN
molecule_dictionary compd ON act.molregno = compd.molregno
JOIN
assays ON act.assay_id = assays.assay_id
WHERE
act.standard_type = 'IC50' -- Example: Filter for IC50 values
AND act.standard_units = 'nM' -- Example: Filter for nM units
AND act.standard_value IS NOT NULL
AND act.standard_value BETWEEN 0 AND 100000 -- Reasonable range for IC50 (adjust
as needed)
--AND act.standard_value ~ '^[\0-9\.]+' -- Removed problematic regex
LIMIT 100;

```

### Explanation of Changes & Addressing Errors:

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

The problem is with the line `AND act.standard_value ~ '^[\0-9\.]+'`. The `~` operator in PostgreSQL is for regular expression matching on *text* data. `act.standard_value` is likely a numeric type. The simplest and most robust solution is to remove this line entirely, as we are already filtering for `NOT NULL` and `BETWEEN` a reasonable range of values. The `BETWEEN` clause effectively handles the numeric validation.

- **Data Selection:** The SQL query now selects the `molregno`, `chembl_id`, `standard_type`, `standard_value`, `standard_units`, `assay_id`, `description` and `canonical_smiles` columns. This provides a good starting point for analysis. The `LIMIT 100` clause ensures you only retrieve the first 100 rows.
- **Data Filtering:** Added `act.standard_units = 'nM'` to select activities measured in nanomolars. Adjust as needed.
- **Saving to CSV:** After running this SQL in pgAdmin, export the result set as a CSV file named `chembl_bioactivities.csv` and save it to the `../data/` directory.

## 4. Python Code (Jupyter Notebook - Analysis)

Create two Jupyter notebooks: `Topic_CheMBL_35_97_1_data_extraction.ipynb` and `Topic_CheMBL_35_97_2_analysis.ipynb`.

### 4.1. Topic\_CheMBL\_35\_97\_1\_data\_extraction.ipynb

This notebook will focus on loading data and converting to a desired format.

```

# Topic_CheMBL_35_97_1_data_extraction.ipynb
import os
import pandas as pd
from rdkit import Chem
from rdkit.Chem import AllChem
import numpy as np

# Define the base path
base_path = os.path.abspath(os.path.join(os.getcwd(), "..")) # Assuming notebooks are
in project_root/notebooks

```

```

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

# Load the CSV data into a Pandas DataFrame
try:
    df = pd.read_csv(csv_file_path)
    print("Data loaded successfully.")
except FileNotFoundError:
    print(f"Error: File not found at {csv_file_path}")
    df = None # or handle the error appropriately

if df is not None:
    print(df.head())
    print(df.info())

```

## 4.2. Topic\_CheMBL\_35\_97\_2\_analysis.ipynb

This notebook will focus on analyzing the loaded data.

```

# Topic_CheMBL_35_97_2_analysis.ipynb
import os
import pandas as pd
from rdkit import Chem
from rdkit.Chem import AllChem
import numpy as np

# Define the base path
base_path = os.path.abspath(os.path.join(os.getcwd(), "..")) # Assuming notebooks are
in project_root/notebooks

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

# Load the CSV data into a Pandas DataFrame
try:
    df = pd.read_csv(csv_file_path)
    print("Data loaded successfully.")
except FileNotFoundError:
    print(f"Error: File not found at {csv_file_path}")
    df = None # or handle the error appropriately

# Drop NULL value
df.dropna(subset=['canonical_smiles'], inplace=True)

# Remove duplicate compounds by canonical_smiles
df = df.drop_duplicates(subset=['canonical_smiles'])

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

# Generate Morgan fingerprints (ECFP4)
df['Morgan_FP'] = df['ROMol'].apply(lambda x: AllChem.GetMorganFingerprintAsBitVect(x,
radius=2, nBits=2048))

if df is not None:
    # Basic Analysis Examples:
    print(f"Number of compounds: {len(df)}")
    print(f"Number of unique compounds: {df['chembl_id'].nunique()}")

```

```
# Standard Value Statistics
print("\nStandard Value Statistics:")
print(df['standard_value'].describe())

#Show number of each IC50 value
print("\nIC50 value distribution:")
print(df['standard_value'].value_counts())
```

## Explanation of the Python Code:

- **Import Libraries:** Imports necessary libraries: os, pandas, rdkit.Chem, rdkit.Chem.AllChem.
- **Path Handling:** Uses os.path.join to construct the full path to the CSV file, making the code more portable. The base\_path is dynamically determined.
- **Data Loading:** Loads the CSV data into a Pandas DataFrame using pd.read\_csv(). Includes error handling for the case where the file is not found.
- **RDKit Integration:**
  - Creates a new column ROMol in the DataFrame.
  - Applies a lambda function to the canonical\_smiles column to convert each SMILES string into an RDKit Mol object using Chem.MolFromSmiles().
  - Filters out rows where the SMILES conversion failed (resulting in None values in ROMol).
- **Feature Generation:**
  - Generates Morgan fingerprints (ECFP4) using AllChem.GetMorganFingerprintAsBitVect(). You can adjust the radius and nBits parameters as needed. The fingerprints are stored in the Morgan\_FP column.
- **Basic Analysis:**
  - Calculates and prints the number of compounds and unique compounds (based on chembl\_id).
  - Calculates and prints descriptive statistics (mean, std, min, max, etc.) of the standard\_value column.
- **Error Handling:** try...except block handles the FileNotFoundError.
- **Data Cleaning:** added data cleaning to delete duplicate and null values.

## 5. Example Analysis (5 Examples based on the above code):

Here are five concrete examples of the analysis you can perform after running the code above:

1. **Distribution of IC50 values:** python import matplotlib.pyplot as plt  
df['standard\_value'].hist(bins=50) # Adjust 'bins' as needed  
plt.xlabel("IC50 Value (nM)") plt.ylabel("Frequency")  
plt.title("Distribution of IC50 Values") plt.show() This will generate a histogram showing the distribution of IC50 values in your dataset.
2. **Most frequent assay descriptions:** python print("\nMost Frequent Assay Descriptions:") print(df['description'].value\_counts().head(10)) # Show the top 10 This will print the 10 most frequent assay descriptions in your dataset, giving you an idea of the types of assays represented.
3. **Calculate the Tanimoto similarity between the first two compounds:**

```
from rdkit import DataStructs

if len(df) >= 2:
    fp1 = df['Morgan_FP'].iloc[0]
    fp2 = df['Morgan_FP'].iloc[1]
    similarity = DataStructs.TanimotoSimilarity(fp1, fp2)
    print(f"Tanimoto Similarity between compound 1 and 2: {similarity}")
```

```
else:
    print("Not enough compounds to calculate similarity.")
```

This calculates the Tanimoto similarity between the Morgan fingerprints of the first two compounds in your dataset. This is a basic measure of structural similarity.

4. **Filter compounds based on IC50 value:**
- ```
python active_compounds =
df[df['standard_value'] <= 100] # Example: IC50 <= 100 nM print(f"\nNumber
of active compounds (IC50 <= 100 nM): {len(active_compounds)}")
print(active_compounds[['chembl_id', 'standard_value']].head()) # Show chembl_id
and standard_value of first 5 This filters the DataFrame to select compounds with IC50
values below a threshold (e.g., 100 nM) and then prints the number of active compounds and
the first few chembl_id and standard_value entries.
```

5. **Find average IC50 for the same target:**

```
average_ic50 = df.groupby('description')['standard_value'].mean()
print(average_ic50)
```

This code groups the DataFrame by assay description (description) and calculates the mean of the IC50 values (standard\_value) for each group. This provides insights into the average activity level for each specific target or assay.

### Important Considerations:

- **Database Credentials:** Make sure your database credentials (user, password, host) are correctly configured in your pgAdmin connection. *Do not hardcode credentials directly into your Python code.* Instead, use environment variables or a configuration file.
- **RDKit Installation:** Ensure that RDKit is properly installed in your Python environment (conda install -c conda-forge rdkit).
- **Adjustments:** You will likely need to adjust the SQL query and Python code based on the specific target or bioassay you are interested in and the specific questions you want to answer.
- **Memory:** Even with a limited number of rows, keep an eye on memory usage, especially when generating fingerprints for a large number of compounds.
- **Error Handling:** Implement more robust error handling in your Python code, especially when dealing with external data sources like databases.
- **Data Validation:** Always validate your data to ensure that it is consistent and accurate.
- **AIMLops Principles:** Consider how you can further structure your project to align with AIMLops principles, such as version control, automated testing, and reproducible workflows.

### French Translation:

Okay, je comprends. Vous travaillez avec les données ChEMBL 35 en utilisant RDKit et vous suivez une structure de répertoires inspirée d'AIMLops. Vous avez besoin d'aide pour créer du code SQL et Python pour extraire des données, effectuer une analyse de base et corriger les erreurs que vous avez rencontrées. Je vais générer le code et expliquer les étapes de l'analyse. Commençons !

#### 1. Modèle d'analyse et objectifs

Le projet vise à exploiter les données ChEMBL 35 en utilisant RDKit pour les tâches de chimioinformatique. Vous utiliserez SQL pour extraire les informations pertinentes (bioactivités, composés, etc.) de la base de données. Ces données extraites seront chargées dans un DataFrame Pandas dans un notebook Jupyter. Dans le notebook, vous utiliserez ensuite RDKit pour effectuer des opérations telles que la génération d'empreintes moléculaires. Le modèle d'analyse est le suivant :

- **Extraction des données (SQL) :** Interroger la base de données ChEMBL 35 pour récupérer un sous-ensemble pertinent de données de bioactivité, en se concentrant sur les composés ayant des données d'activité pour une cible ou un bioessai spécifique. Il est essentiel de limiter les données aux 100 premières lignes pour gérer les ressources de calcul.

- **Chargement et prétraitement des données (Python)** : Charger les données CSV extraites de la requête SQL dans un DataFrame Pandas.
- **Intégration de RDKit (Python)** : Utiliser RDKit pour traiter les composés, en générant des représentations moléculaires (par exemple, des chaînes SMILES converties en molécules RDKit).
- **Génération de caractéristiques (Python)** : Dériver des caractéristiques des représentations moléculaires (par exemple, les empreintes digitales de Morgan).
- **Analyse de base (Python)** : Effectuer quelques analyses d'exemple telles que :
  - Afficher le nombre de composés, le nombre de composés uniques
  - Calculer quelques statistiques des valeurs standard

## 2. Structure des répertoires (implicite d'AIMLops)

D'après votre description, je suppose une structure de répertoires simplifiée comme celle-ci :

```
project_root/
├── data/
│   └── chembl_bioactivities.csv # Exemple de fichier CSV
├── notebooks/
│   ├── Topic_CheMBL_35_97_1_data_extraction.ipynb
│   └── Topic_CheMBL_35_97_2_analysis.ipynb
└── src/ # (Optionnel) Peut contenir des modules Python réutilisables
```

## 3. Code SQL (Extraction des données de bioactivité)

Voici le code SQL pour extraire les données de bioactivité et les enregistrer dans un fichier CSV. Cette requête extrait des données de plusieurs tables ChEMBL et filtre pour un type d'activité spécifique et une valeur standard raisonnable. Elle limite également les résultats aux 100 premières entrées.

-- Fichier : ../data/chembl\_bioactivities.sql

```
SELECT
    act.molregno,
    compd.chembl_id,
    act.standard_type,
    act.standard_value,
    act.standard_units,
    act.assay_id,
    assays.description,
    compd.canonical_smiles
FROM
    activities act
JOIN
    molecule_dictionary compd ON act.molregno = compd.molregno
JOIN
    assays ON act.assay_id = assays.assay_id
WHERE
    act.standard_type = 'IC50' -- Exemple : filtrer pour les valeurs IC50
    AND act.standard_units = 'nM' -- Exemple : Filtrer pour les unités nM
    AND act.standard_value IS NOT NULL
    AND act.standard_value BETWEEN 0 AND 100000 -- Plage raisonnable pour IC50 (à
ajuster si nécessaire)
    --AND act.standard_value ~ '^[\d-9\.]+\d' -- Suppression de l'expression régulière
problématique
LIMIT 100;
```

### Explication des modifications et correction des erreurs :

- Erreur a:ERROR: operator does not exist: numeric ~ unknown, LINE 12: AND act.standard\_value ~ '^[\d-9\.]+\d'



Le problème vient de la ligne `AND act.standard_value ~ '^[0-9\\.]+$'`. L'opérateur `~` dans PostgreSQL est utilisé pour la correspondance d'expressions régulières sur les données *textuelles*. `act.standard_value` est probablement de type `numeric`. La solution la plus simple et la plus robuste est de supprimer complètement cette ligne, car nous filtrons déjà pour `NOT NULL` et `BETWEEN` une plage de valeurs raisonnable. La clause `BETWEEN` gère efficacement la validation numérique.

- **Sélection des données :** La requête SQL sélectionne maintenant les colonnes `molregno`, `chembl_id`, `standard_type`, `standard_value`, `standard_units`, `assay_id`, `description` et `canonical_smiles`. Ceci fournit un bon point de départ pour l'analyse. La clause `LIMIT 100` garantit que vous ne récupérez que les 100 premières lignes.
- **Filtrage des données :** Ajout de `act.standard_units = 'nM'` pour sélectionner les activités mesurées en nanomolaires. Ajustez au besoin.
- **Enregistrement au format CSV :** Après avoir exécuté ce SQL dans pgAdmin, exportez l'ensemble de résultats sous forme de fichier CSV nommé `chembl_bioactivities.csv` et enregistrez-le dans le répertoire `../data/`.

## 4. Code Python (Jupyter Notebook - Analyse)

Créez deux notebooks Jupyter : `Topic_CheMBL_35_97_1_data_extraction.ipynb` et `Topic_CheMBL_35_97_2_analysis.ipynb`.

### 4.1. Topic\_CheMBL\_35\_97\_1\_data\_extraction.ipynb

Ce notebook se concentrera sur le chargement des données et la conversion dans un format souhaité.

```
# Topic_CheMBL_35_97_1_data_extraction.ipynb
import os
import pandas as pd
from rdkit import Chem
from rdkit.Chem import AllChem
import numpy as np

# Définir le chemin de base
base_path = os.path.abspath(os.path.join(os.getcwd(), "..")) # En supposant que Les
notebooks sont dans project_root/notebooks

# Construire le chemin vers le fichier CSV
csv_file_path = os.path.join(base_path, "data", "chembl_bioactivities.csv")

# Charger Les données CSV dans un DataFrame Pandas
try:
    df = pd.read_csv(csv_file_path)
    print("Données chargées avec succès.")
except FileNotFoundError:
    print(f"Erreur : Fichier non trouvé à {csv_file_path}")
    df = None # ou gérer l'erreur de manière appropriée

if df is not None:
    print(df.head())
    print(df.info())
```

### 4.2. Topic\_CheMBL\_35\_97\_2\_analysis.ipynb

Ce notebook se concentrera sur l'analyse des données chargées.

```
# Topic_CheMBL_35_97_2_analysis.ipynb
import os
import pandas as pd
```

```

from rdkit import Chem
from rdkit.Chem import AllChem
import numpy as np

# Définir Le chemin de base
base_path = os.path.abspath(os.path.join(os.getcwd(), "..")) # En supposant que Les
notebooks sont dans project_root/notebooks

# Construire Le chemin vers Le fichier CSV
csv_file_path = os.path.join(base_path, "data", "chembl_bioactivities.csv")

# Charger Les données CSV dans un DataFrame Pandas
try:
    df = pd.read_csv(csv_file_path)
    print("Données chargées avec succès.")
except FileNotFoundError:
    print(f"Erreur : Fichier non trouvé à {csv_file_path}")
    df = None # ou gérer l'erreur de manière appropriée

#Supprimer Les valeurs NULL
df.dropna(subset=['canonical_smiles'], inplace=True)

# Supprimer Les composés dupliqués par canonical_smiles
df = df.drop_duplicates(subset=['canonical_smiles'])

# Convertir SMILES en objets Mol RDKit
df['ROMol'] = df['canonical_smiles'].apply(lambda x: Chem.MolFromSmiles(x))
df = df[df['ROMol'].notna()]

# Générer des empreintes digitales Morgan (ECFP4)
df['Morgan_FP'] = df['ROMol'].apply(lambda x: AllChem.GetMorganFingerprintAsBitVect(x,
radius=2, nBits=2048))

if df is not None:
    # Exemples d'analyse de base :
    print(f"Nombre de composés : {len(df)}")
    print(f"Nombre de composés uniques : {df['chembl_id'].nunique()}")

    # Statistiques des valeurs standard
    print("\nStatistiques des valeurs standard :")
    print(df['standard_value'].describe())

    #Afficher Le nombre de chaque valeur IC50
    print("\nDistribution des valeurs IC50 :")
    print(df['standard_value'].value_counts())

```

### Explication du code Python :

- **Importer les bibliothèques :** Importe les bibliothèques nécessaires : `os`, `pandas`, `rdkit.Chem`, `rdkit.Chem.AllChem`.
- **Gestion des chemins :** Utilise `os.path.join` pour construire le chemin complet vers le fichier CSV, ce qui rend le code plus portable. Le `base_path` est déterminé dynamiquement.
- **Chargement des données :** Charge les données CSV dans un DataFrame Pandas à l'aide de `pd.read_csv()`. Inclut la gestion des erreurs dans le cas où le fichier est introuvable.
- **Intégration de RDKit :**
  - Crée une nouvelle colonne `ROMol` dans le DataFrame.
  - Applique une fonction lambda à la colonne `canonical_smiles` pour convertir chaque chaîne SMILES en un objet `Mol` RDKit à l'aide de `Chem.MolFromSmiles()`.



- Filtre les lignes où la conversion SMILES a échoué (ce qui entraîne des valeurs None dans ROMol).
- **Génération de caractéristiques :**
  - Génère des empreintes digitales Morgan (ECFP4) à l'aide de `AllChem.GetMorganFingerprintAsBitVect()`. Vous pouvez ajuster les paramètres `radius` et `nBits` selon vos besoins. Les empreintes digitales sont stockées dans la colonne `Morgan_FP`.
- **Analyse de base :**
  - Calcule et affiche le nombre de composés et de composés uniques (basé sur `chembl_id`).
  - Calcule et affiche les statistiques descriptives (moyenne, écart type, min, max, etc.) de la colonne `standard_value`.
- **Gestion des erreurs :** Le bloc `try...except` gère le `FileNotFoundError`.
- **Nettoyage des données :** Ajout d'un nettoyage des données pour supprimer les valeurs en double et nulles.

## 5. Exemples d'analyse (5 exemples basés sur le code ci-dessus) :

Voici cinq exemples concrets de l'analyse que vous pouvez effectuer après avoir exécuté le code ci-dessus :

1. **Distribution des valeurs IC50 :** `python` `import matplotlib.pyplot as plt`  
`df['standard_value'].hist(bins=50)` # Ajuster 'bins' au besoin  
`plt.xlabel("Valeur IC50 (nM)")` `plt.ylabel("Fréquence")`  
`plt.title("Distribution des valeurs IC50")` `plt.show()` Ceci générera un histogramme montrant la distribution des valeurs IC50 dans votre ensemble de données.
2. **Descriptions des essais les plus fréquentes :** `python` `print("\nDescriptions des essais les plus fréquentes :")`  
`print(df['description'].value_counts().head(10))` # Afficher les 10 premiers Ceci affichera les 10 descriptions d'essais les plus fréquentes dans votre ensemble de données, vous donnant une idée des types d'essais représentés.

### 3. Calculer la similarité de Tanimoto entre les deux premiers composés :

```
from rdkit import DataStructs

if len(df) >= 2:
    fp1 = df['Morgan_FP'].iloc[0]
    fp2 = df['Morgan_FP'].iloc[1]
    similarity = DataStructs.TanimotoSimilarity(fp1, fp2)
    print(f"Similarité de Tanimoto entre le composé 1 et 2 : {similarity}")
else:
    print("Pas assez de composés pour calculer la similarité.")
```

Ceci calcule la similarité de Tanimoto entre les empreintes digitales Morgan des deux premiers composés de votre ensemble de données. C'est une mesure de base de la similarité structurelle.

4. **Filtrer les composés en fonction de la valeur IC50 :** `python` `active_compounds =`  
`df[df['standard_value'] <= 100]` # Exemple : `IC50 <= 100 nM` `print(f"\nNombre de composés actifs (IC50 <= 100 nM) : {len(active_compounds)}")`  
`print(active_compounds[['chembl_id', 'standard_value']].head())` # Afficher `chembl_id` et `standard_value` des 5 premiers Ceci filtre le DataFrame pour sélectionner les composés avec des valeurs IC50 inférieures à un seuil (par exemple, 100 nM), puis affiche le nombre de composés actifs et les premières entrées `chembl_id` et `standard_value`.
5. **Trouver la moyenne des IC50 pour la même cible :**

```
average_ic50 = df.groupby('description')['standard_value'].mean()  
print(average_ic50)
```

Ce code regroupe le DataFrame par description d'essai (description) et calcule la moyenne des valeurs IC50 (standard\_value) pour chaque groupe. Cela donne un aperçu du niveau d'activité moyen pour chaque cible ou essai spécifique.

### Considérations importantes :

- **Informations d'identification de la base de données :** Assurez-vous que les informations d'identification de votre base de données (utilisateur, mot de passe, hôte) sont correctement configurées dans votre connexion pgAdmin. *Ne codez pas en dur les informations d'identification directement dans votre code Python.* Utilisez plutôt des variables d'environnement ou un fichier de configuration.
- **Installation de RDKit :** Assurez-vous que RDKit est correctement installé dans votre environnement Python (`conda install -c conda-forge rdkit`).
- **Ajustements :** Vous devrez probablement ajuster la requête SQL et le code Python en fonction de la cible ou du bioessai spécifique qui vous intéresse et des questions spécifiques auxquelles vous souhaitez répondre.
- **Mémoire :** Même avec un nombre limité de lignes, surveillez l'utilisation de la mémoire, en particulier lors de la génération d'empreintes digitales pour un grand nombre de composés.
- **Gestion des erreurs :** Mettez en œuvre une gestion des erreurs plus robuste dans votre code Python, en particulier lors du traitement de sources de données externes telles que les bases de données.
- **Validation des données :** Validez toujours vos données pour vous assurer qu'elles sont cohérentes et exactes.
- **Principes AIMLops :** Réfléchissez à la manière dont vous pouvez structurer davantage votre projet pour l'aligner sur les principes AIMLops, tels que le contrôle de version, les tests automatisés et les flux de travail reproductibles.

I hope this helps you with your ChEMBL 35 data analysis project! Good luck!