PharmApp Suite

🧠 AI for Drug Discovery and Development 🧪

🧩 Topic: Prioritizing compounds for synthesis

Okay, I understand. You're working on a Chembl 35 data analysis project, aiming to extract and analyze information relevant to drug research and development. You're using PostgreSQL (psql) with a Chembl_35 database, RDKit for cheminformatics, and Jupyter Notebooks for your analysis. You want to keep the data subset small (100 rows) due to resource constraints and are encountering specific errors. Let's break this down.

1. Analysis Model and Objectives:

The overall objective is to leverage the Chembl 35 database to extract relevant data, likely focusing on activity information (e.g., IC50, Ki) against specific targets. You'll use SQL to query the database and filter the data. Then, you'll use RDKit in Python to process the molecular structures and potentially perform structure-activity relationship (SAR) analysis.

Specific objectives might include:

- **Target Identification:** Identify compounds active against a specific target protein.
- **Activity Range Filtering:** Focus on compounds with activity within a certain range (e.g., IC50 < 100 nM).
- **Structure Similarity Analysis:** Identify compounds structurally similar to a known active compound.
- **Basic SAR:** Investigate the relationship between molecular properties and activity.

2. Addressing Errors:

- **ERROR: operator does not exist: numeric ~ unknown, LINE 12: AND act.standard_value ~ '[1]+$'**: This error indicates that you're trying to use the regular expression matching operator (~) on a numeric column (`act.standard_value`). PostgreSQL's ~ operator is for string matching. The column `act.standard_value` is likely of type `numeric`, so it needs to be cast to text before being used with the regular expression.

- **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. You have two options:

    - **Update scikit-learn:** The recommended approach is to update to a newer version of scikit-learn using `pip install -U scikit-learn`.
    - **Remove `squared=False`:** If updating isn't feasible, remove the `squared=False` argument. This will return the Mean Squared Error (MSE) instead of the Root Mean Squared Error (RMSE).

3. Recommended File Structure:

```
Project/
├── data/
│   └── chembl_data.csv  # Output from SQL query
├── notebooks/
│   └── Topic_CheMBL_35_14_1_data_extraction.ipynb
│   └── Topic_CheMBL_35_14_2_data_analysis.ipynb
```

[1] 0-9.

```
├── sql/
│   └── chembl_query.sql
└── README.md
```

4. Code Examples:

a. SQL (chembl_query.sql): This example retrieves data for a specific target (e.g., 'CHEMBL205' - Dopamine D4 receptor) and filters for activity values that are numeric. It also limits the results to 100 rows.

```sql
-- sql/chembl_query.sql
SELECT
    cmp.chembl_id,
    act.standard_type,
    act.standard_value,
    act.standard_units,
    mol.molfile
FROM
    activities act
JOIN
    molecule_dictionary cmp ON act.molregno = cmp.molregno
JOIN
    target_dictionary tgt ON act.tid = tgt.tid
JOIN
    compound_structures mol ON cmp.molregno = mol.molregno
WHERE
    tgt.chembl_id = 'CHEMBL205'  -- Dopamine D4 receptor
    AND act.standard_type = 'IC50'
    AND act.standard_units = 'nM'
    AND act.standard_value::text ~ '^[0-9\.]+$'  -- FIX: Cast to text for regex
ORDER BY
    act.standard_value
LIMIT 100;
```

How to run:

1. Open pgAdmin.
2. Connect to your database (192.168.206.136, rd, rd, chembl_35).
3. Open a new query window.
4. Paste the SQL code.
5. Execute the query.
6. Export the results as a CSV file (e.g., `chembl_data.csv`) to your `data/` directory.

b. Python (Topic_CheMBL_35_14_1_data_extraction.ipynb): This notebook reads the CSV file generated by the SQL query.

```python
# notebooks/Topic_CheMBL_35_14_1_data_extraction.ipynb
import os
import pandas as pd

base_path = ".."  # Assuming notebooks are one level below the project root
data_path = os.path.join(base_path, "data", "chembl_data.csv")

try:
    df = pd.read_csv(data_path)
    print(df.head())
    print(f"Data loaded successfully from {data_path}")
except FileNotFoundError:
    print(f"Error: File not found at {data_path}. Make sure you have run the SQL query
and saved the data.")
```

```
except Exception as e:
    print(f"An error occurred: {e}")
```

c. Python (Topic_CheMBL_35_14_2_data_analysis.ipynb): This notebook performs basic analysis using RDKit.

```
# notebooks/Topic_CheMBL_35_14_2_data_analysis.ipynb
import os
import pandas as pd
from rdkit import Chem
from rdkit.Chem import Descriptors
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error

base_path = ".."  # Assuming notebooks are one level below the project root
data_path = os.path.join(base_path, "data", "chembl_data.csv")

try:
    df = pd.read_csv(data_path)
except FileNotFoundError:
    print(f"Error: File not found at {data_path}. Make sure you have run the SQL query
and saved the data.")
    exit()
except Exception as e:
    print(f"An error occurred: {e}")
    exit()

# Convert molfile to RDKit Mol objects
df['ROMol'] = df['molfile'].apply(lambda x: Chem.MolFromMolBlock(x) if pd.notnull(x)
else None)
df = df.dropna(subset=['ROMol'])  # Remove rows with invalid molecules

# Calculate Molecular Weight
df['MW'] = df['ROMol'].apply(Descriptors.MolWt)

# Convert IC50 to pIC50
df['pIC50'] = -np.log10(df['standard_value'] * 1e-9)  # Convert nM to Molar and then
to pIC50

# Prepare data for regression
X = df[['MW']]  # Feature: Molecular Weight
y = df['pIC50'] # Target: pIC50

# Handle potential missing values (replace with the median)
X = X.fillna(X.median())
y = y.fillna(y.median())

# 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)

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

# Make predictions
y_pred = model.predict(X_test)
```

```
# Evaluate the model. Check your scikit-learn version before using squared=False
try:
    rmse = np.sqrt(mean_squared_error(y_test, y_pred, squared=False)) # Requires
scikit-learn >= 0.22
except TypeError:
    rmse = np.sqrt(mean_squared_error(y_test, y_pred)) # For older scikit-learn
versions
print(f"Root Mean Squared Error: {rmse}")

# Example: Print predicted vs. actual pIC50 for a few test samples
for i in range(5):
    print(f"Predicted pIC50: {y_pred[i]:.2f}, Actual pIC50: {y_test.iloc[i]:.2f}")
```

5. Five Examples of Analysis You Could Perform:

1. **Basic Activity Distribution:** Create a histogram of the `pIC50` values. This will show the range of activity and identify the most common activity levels.

   ```
   import matplotlib.pyplot as plt

   plt.hist(df['pIC50'], bins=20)
   plt.xlabel('pIC50')
   plt.ylabel('Frequency')
   plt.title('Distribution of pIC50 Values')
   plt.show()
   ```

2. **Molecular Weight vs. Activity Scatter Plot:** Plot molecular weight against `pIC50` to see if there's any correlation.

   ```
   plt.scatter(df['MW'], df['pIC50'])
   plt.xlabel('Molecular Weight (Da)')
   plt.ylabel('pIC50')
   plt.title('Molecular Weight vs. pIC50')
   plt.show()
   ```

3. **Tanimoto Similarity Search:** Given a known active molecule, search for similar molecules in your dataset based on Tanimoto similarity. This requires calculating fingerprints (e.g., Morgan fingerprints) for the molecules.

   ```
   from rdkit.Chem import AllChem
   from rdkit import DataStructs

   # Choose a reference molecule (replace with a ChEMBL ID from your data)
   reference_chembl_id = df['chembl_id'].iloc[0]  # Example: Take the first
   molecule
   reference_mol = df['ROMol'].iloc[0]
   reference_fp = AllChem.GetMorganFingerprint(reference_mol, radius=2) # radius of
   2 is common

   # Calculate Tanimoto similarity for all molecules
   similarities = []
   for mol in df['ROMol']:
       if mol is not None:
           fp = AllChem.GetMorganFingerprint(mol, radius=2)
           similarity = DataStructs.TanimotoSimilarity(reference_fp, fp)
           similarities.append(similarity)
       else:
           similarities.append(0.0)  # Handle missing molecules

   df['Tanimoto'] = similarities
   ```

```
# Sort by similarity and display the top 5 most similar compounds
top_similar = df.sort_values('Tanimoto', ascending=False).head(5)
print(f"Compounds most similar to {reference_chembl_id}:")
print(top_similar[['chembl_id', 'Tanimoto', 'pIC50']])
```

4. **Calculate and Visualize a Different Molecular Descriptor:** Calculate LogP (partition coefficient) and plot it against activity.

```
df['LogP'] = df['ROMol'].apply(Descriptors.MolLogP)

plt.scatter(df['LogP'], df['pIC50'])
plt.xlabel('LogP')
plt.ylabel('pIC50')
plt.title('LogP vs. pIC50')
plt.show()
```

5. **Basic SAR Analysis - Grouping by Structural Alerts (Example):** This is a simplified example. Real structural alerts require more sophisticated analysis. This demonstrates the *idea* of associating structural features with activity. **Important:** This is a *highly* simplified example. True SAR analysis requires domain knowledge and careful consideration of structural alerts.

```
from rdkit.Chem import Lipinski

# Define a simple "structural alert" (e.g., high LogP)
df['HighLogP'] = df['LogP'] > 5  # Arbitrary threshold

# Group by the "structural alert" and calculate the mean pIC50
activity_by_alert = df.groupby('HighLogP')['pIC50'].mean()
print("Mean pIC50 by High LogP Alert:")
print(activity_by_alert)
```

Important Considerations:

- **Error Handling:** The Python code includes `try...except` blocks for file loading. Extend this to handle potential errors when creating RDKit Mol objects (e.g., if a molfile is malformed).
- **Data Cleaning:** Thoroughly clean your data. Check for missing values, invalid molecules, and outliers.
- **Units:** Be *extremely* careful with units. Ensure all activity values are in consistent units (e.g., nM) before performing calculations.
- **Database Performance:** If you need to analyze a significantly larger dataset, consider using database indexing to improve query performance.
- **Virtual Environments:** Use virtual environments (`venv` or `conda`) to manage your project dependencies and avoid conflicts with other Python projects.

In Summary:

This guide provides a solid foundation for your Chembl 35 data analysis project. Remember to address the errors, organize your code, and focus on asking specific research questions that you can answer using the data. Good luck!

Traduction en français:

D'accord, je comprends. Vous travaillez sur un projet d'analyse des données Chembl 35, visant à extraire et à analyser des informations pertinentes pour la recherche et le développement de médicaments. Vous utilisez PostgreSQL (psql) avec une base de données Chembl_35, RDKit pour la chemoinformatique et des notebooks Jupyter pour votre analyse. Vous souhaitez que le sous-

ensemble de données reste petit (100 lignes) en raison de contraintes de ressources et vous rencontrez des erreurs spécifiques. Décomposons cela.

1. Modèle d'analyse et objectifs :

L'objectif général est d'exploiter la base de données Chembl 35 pour extraire des données pertinentes, en se concentrant probablement sur les informations d'activité (par exemple, IC50, Ki) contre des cibles spécifiques. Vous utiliserez SQL pour interroger la base de données et filtrer les données. Ensuite, vous utiliserez RDKit en Python pour traiter les structures moléculaires et potentiellement effectuer une analyse de la relation structure-activité (SAR).

Les objectifs spécifiques pourraient inclure :

- **Identification de la cible :** Identifier les composés actifs contre une protéine cible spécifique.
- **Filtrage de la plage d'activité :** Se concentrer sur les composés ayant une activité dans une certaine plage (par exemple, IC50 < 100 nM).
- **Analyse de la similarité de la structure :** Identifier les composés structurellement similaires à un composé actif connu.
- **SAR de base :** Étudier la relation entre les propriétés moléculaires et l'activité.

2. Résolution des erreurs :

- **ERROR: operator does not exist: numeric ~ unknown, LINE 12: AND act.standard_value ~ '[2]+$'**: Cette erreur indique que vous essayez d'utiliser l'opérateur de correspondance d'expression régulière (~) sur une colonne numérique (`act.standard_value`). L'opérateur ~ de PostgreSQL est destiné à la correspondance de chaînes. La colonne `act.standard_value` est probablement de type `numeric`, elle doit donc être convertie en texte avant d'être utilisée avec l'expression régulière.

- **old scikit-learn version does not support parameters squared=False in the mean_squared_error function**: Cela signifie que vous utilisez une ancienne version de scikit-learn. Vous avez deux options :

    - **Mettre à jour scikit-learn :** L'approche recommandée est de mettre à jour vers une version plus récente de scikit-learn en utilisant `pip install -U scikit-learn`.
    - **Supprimer `squared=False` :** Si la mise à jour n'est pas possible, supprimez l'argument `squared=False`. Cela renverra l'erreur quadratique moyenne (MSE) au lieu de l'erreur quadratique moyenne (RMSE).

3. Structure de fichiers recommandée :

```
Projet/
├── data/
│   └── chembl_data.csv  # Sortie de la requête SQL
├── notebooks/
│   └── Topic_CheMBL_35_14_1_extraction_des_données.ipynb
│   └── Topic_CheMBL_35_14_2_analyse_des_données.ipynb
├── sql/
│   └── chembl_query.sql
└── README.md
```

4. Exemples de code :

a. SQL (chembl_query.sql) : Cet exemple récupère les données d'une cible spécifique (par exemple, 'CHEMBL205' - Récepteur D4 de la dopamine) et filtre les valeurs d'activité qui sont numériques. Il limite également les résultats à 100 lignes.

[2] 0-9.

```sql
-- sql/chembl_query.sql
SELECT
    cmp.chembl_id,
    act.standard_type,
    act.standard_value,
    act.standard_units,
    mol.molfile
FROM
    activities act
JOIN
    molecule_dictionary cmp ON act.molregno = cmp.molregno
JOIN
    target_dictionary tgt ON act.tid = tgt.tid
JOIN
    compound_structures mol ON cmp.molregno = mol.molregno
WHERE
    tgt.chembl_id = 'CHEMBL205'  -- Récepteur D4 de la dopamine
    AND act.standard_type = 'IC50'
    AND act.standard_units = 'nM'
    AND act.standard_value::text ~ '^[0-9\.]+$'  -- FIX: Conversion en texte pour
regex
ORDER BY
    act.standard_value
LIMIT 100;
```

Comment exécuter :

1. Ouvrez pgAdmin.
2. Connectez-vous à votre base de données (192.168.206.136, rd, rd, chembl_35).
3. Ouvrez une nouvelle fenêtre de requête.
4. Collez le code SQL.
5. Exécutez la requête.
6. Exportez les résultats sous forme de fichier CSV (par exemple, `chembl_data.csv`) dans votre répertoire `data/`.

b. Python (Topic_CheMBL_35_14_1_data_extraction.ipynb) : Ce notebook lit le fichier CSV généré par la requête SQL.

```python
# notebooks/Topic_CheMBL_35_14_1_data_extraction.ipynb
import os
import pandas as pd

base_path = ".."  # En supposant que les notebooks sont un niveau en dessous de la
racine du projet
data_path = os.path.join(base_path, "data", "chembl_data.csv")

try:
    df = pd.read_csv(data_path)
    print(df.head())
    print(f"Données chargées avec succès depuis {data_path}")
except FileNotFoundError:
    print(f"Erreur : Fichier introuvable à {data_path}. Assurez-vous d'avoir exécuté
la requête SQL et enregistré les données.")
except Exception as e:
    print(f"Une erreur s'est produite : {e}")
```

c. Python (Topic_CheMBL_35_14_2_data_analysis.ipynb) : Ce notebook effectue une analyse de base à l'aide de RDKit.

```python
# notebooks/Topic_CheMBL_35_14_2_data_analysis.ipynb
import os
import pandas as pd
from rdkit import Chem
from rdkit.Chem import Descriptors
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error

base_path = ".."  # En supposant que les notebooks sont un niveau en dessous de la racine du projet
data_path = os.path.join(base_path, "data", "chembl_data.csv")

try:
    df = pd.read_csv(data_path)
except FileNotFoundError:
    print(f"Erreur : Fichier introuvable à {data_path}. Assurez-vous d'avoir exécuté la requête SQL et enregistré les données.")
    exit()
except Exception as e:
    print(f"Une erreur s'est produite : {e}")
    exit()

# Convertir molfile en objets Mol RDKit
df['ROMol'] = df['molfile'].apply(lambda x: Chem.MolFromMolBlock(x) if pd.notnull(x) else None)
df = df.dropna(subset=['ROMol'])  # Supprimer les lignes avec des molécules invalides

# Calculer le poids moléculaire
df['MW'] = df['ROMol'].apply(Descriptors.MolWt)

# Convertir IC50 en pIC50
df['pIC50'] = -np.log10(df['standard_value'] * 1e-9)  # Convertir nM en Molaire puis en pIC50

# Préparer les données pour la régression
X = df[['MW']]  # Caractéristique : Poids moléculaire
y = df['pIC50'] # Cible : pIC50

# Gérer les valeurs manquantes potentielles (remplacer par la médiane)
X = X.fillna(X.median())
y = y.fillna(y.median())

# Diviser les données en ensembles d'entraînement et de test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Entraîner un modèle de régression linéaire
model = LinearRegression()
model.fit(X_train, y_train)

# Faire des prédictions
y_pred = model.predict(X_test)

# Évaluer le modèle. Vérifiez votre version de scikit-learn avant d'utiliser squared=False
try:
    rmse = np.sqrt(mean_squared_error(y_test, y_pred, squared=False)) # Nécessite
```

```
scikit-learn >= 0.22
except TypeError:
    rmse = np.sqrt(mean_squared_error(y_test, y_pred)) # Pour les anciennes versions
de scikit-learn
print(f"Racine carrée de l'erreur quadratique moyenne : {rmse}")

# Exemple : Afficher la pIC50 prédite par rapport à la pIC50 réelle pour quelques
échantillons de test
for i in range(5):
    print(f"pIC50 prédite : {y_pred[i]:.2f}, pIC50 réelle : {y_test.iloc[i]:.2f}")
```

5. Cinq exemples d'analyses que vous pourriez effectuer :

1. **Distribution de l'activité de base :** Créez un histogramme des valeurs `pIC50`. Cela montrera la plage d'activité et identifiera les niveaux d'activité les plus courants.

   ```
   import matplotlib.pyplot as plt

   plt.hist(df['pIC50'], bins=20)
   plt.xlabel('pIC50')
   plt.ylabel('Fréquence')
   plt.title('Distribution des valeurs de pIC50')
   plt.show()
   ```

2. **Diagramme de dispersion du poids moléculaire par rapport à l'activité :** Tracez le poids moléculaire par rapport à `pIC50` pour voir s'il existe une corrélation.

   ```
   plt.scatter(df['MW'], df['pIC50'])
   plt.xlabel('Poids moléculaire (Da)')
   plt.ylabel('pIC50')
   plt.title('Poids moléculaire vs. pIC50')
   plt.show()
   ```

3. **Recherche de similarité de Tanimoto :** Étant donné une molécule active connue, recherchez des molécules similaires dans votre ensemble de données en fonction de la similarité de Tanimoto. Cela nécessite le calcul des empreintes digitales (par exemple, les empreintes digitales de Morgan) pour les molécules.

   ```
   from rdkit.Chem import AllChem
   from rdkit import DataStructs

   # Choisir une molécule de référence (remplacer par un identifiant ChEMBL de vos
   données)
   reference_chembl_id = df['chembl_id'].iloc[0]  # Exemple : Prendre la première
   molécule
   reference_mol = df['ROMol'].iloc[0]
   reference_fp = AllChem.GetMorganFingerprint(reference_mol, radius=2) # rayon de
   2 est courant

   # Calculer la similarité de Tanimoto pour toutes les molécules
   similarities = []
   for mol in df['ROMol']:
       if mol is not None:
           fp = AllChem.GetMorganFingerprint(mol, radius=2)
           similarity = DataStructs.TanimotoSimilarity(reference_fp, fp)
           similarities.append(similarity)
       else:
           similarities.append(0.0)  # Gérer les molécules manquantes

   df['Tanimoto'] = similarities
   ```

```
# Trier par similarité et afficher les 5 composés les plus similaires
top_similar = df.sort_values('Tanimoto', ascending=False).head(5)
print(f"Composés les plus similaires à {reference_chembl_id} :")
print(top_similar[['chembl_id', 'Tanimoto', 'pIC50']])
```

4. **Calculer et visualiser un autre descripteur moléculaire :** Calculer LogP (coefficient de partage) et le tracer par rapport à l'activité.

```
df['LogP'] = df['ROMol'].apply(Descriptors.MolLogP)

plt.scatter(df['LogP'], df['pIC50'])
plt.xlabel('LogP')
plt.ylabel('pIC50')
plt.title('LogP vs. pIC50')
plt.show()
```

5. **Analyse SAR de base - Regroupement par alertes structurelles (Exemple) :** Ceci est un exemple simplifié. Les véritables alertes structurelles nécessitent une analyse plus sophistiquée. Ceci démontre l'*idée* d'associer des caractéristiques structurelles à l'activité. **Important :** Il s'agit d'un exemple *très* simplifié. Une véritable analyse SAR nécessite une connaissance du domaine et une prise en compte attentive des alertes structurelles.

```
from rdkit.Chem import Lipinski

# Définir une simple "alerte structurelle" (par exemple, LogP élevé)
df['HighLogP'] = df['LogP'] > 5  # Seuil arbitraire

# Grouper par "alerte structurelle" et calculer la pIC50 moyenne
activity_by_alert = df.groupby('HighLogP')['pIC50'].mean()
print("pIC50 moyenne par alerte LogP élevé :")
print(activity_by_alert)
```

Considérations importantes :

- **Gestion des erreurs :** Le code Python inclut des blocs `try...except` pour le chargement des fichiers. Étendez cela pour gérer les erreurs potentielles lors de la création d'objets Mol RDKit (par exemple, si un molfile est mal formé).
- **Nettoyage des données :** Nettoyez soigneusement vos données. Vérifiez les valeurs manquantes, les molécules invalides et les valeurs aberrantes.
- **Unités :** Soyez *extrêmement* prudent avec les unités. Assurez-vous que toutes les valeurs d'activité sont dans des unités cohérentes (par exemple, nM) avant d'effectuer des calculs.
- **Performances de la base de données :** Si vous devez analyser un ensemble de données beaucoup plus volumineux, envisagez d'utiliser l'indexation de la base de données pour améliorer les performances des requêtes.
- **Environnements virtuels :** Utilisez des environnements virtuels (`venv` ou `conda`) pour gérer les dépendances de votre projet et éviter les conflits avec d'autres projets Python.

En résumé :

Ce guide fournit une base solide pour votre projet d'analyse des données Chembl 35. N'oubliez pas de corriger les erreurs, d'organiser votre code et de vous concentrer sur les questions de recherche spécifiques auxquelles vous pouvez répondre à l'aide des données. Bonne chance !