PharmApp Suite

🧠 AI for Drug Discovery and Development 💉

🧩 Topic: Binding site comparison across targets

Okay, I understand. You're working on a Chembl 35 data analysis project using RDKit, with a specific focus on a topic (Topic_CheMBL_35_30) and want to integrate it into an AIMLops-style folder structure. You need help with SQL queries, Python code, error handling, and generating relevant examples.

Let's break this down.

1. Analysis Model

The core of your analysis likely involves:

- **Data Extraction and Preparation:** Pulling relevant data from the ChEMBL database using SQL. This involves identifying the tables containing activity data, compound information, and any relevant target details.
- **Data Cleaning and Filtering:** Addressing issues like invalid or missing data, filtering activities based on confidence scores or assay types, and selecting a representative subset (your 100 rows). This is crucial for reliable downstream analysis. The error you're seeing (ERROR: operator does not exist: numeric ~ unknown) suggests you are using a regular expression operator (~) on a numeric column. You'll need to cast or convert the numeric column to text if you intend to use regular expressions, or use numeric comparisons if appropriate.
- **Chemical Structure Handling with RDKit:** Using RDKit to process the SMILES strings from ChEMBL, generate molecular descriptors (e.g., molecular weight, LogP, topological polar surface area (TPSA)), and potentially perform substructure searches or similarity calculations.
- **Data Analysis and Modeling:** Applying machine learning techniques (regression, classification, clustering, etc.) to relate the molecular descriptors to activity data. This might involve building predictive models for activity, identifying structure-activity relationships (SAR), or clustering compounds based on their properties.
- **Visualization:** Creating plots and visualizations to explore the data, understand model results, and communicate findings.

2. Folder Structure (AIMLops Style)

While you haven't provided the exact structure, an AIMLops-compliant structure often looks something like this:

```
Topic_CheMBL_35_30/
├── data/           # Raw and processed data (CSV files, etc.)
├── notebooks/      # Jupyter notebooks for exploration and analysis
│   ├── Topic_CheMBL_35_30_1_Data_Extraction.ipynb
│   ├── Topic_CheMBL_35_30_2_Descriptor_Calculation.ipynb
│   ├── Topic_CheMBL_35_30_3_Model_Building.ipynb
│   ├── Topic_CheMBL_35_30_4_Model_Evaluation.ipynb
│   ├── Topic_CheMBL_35_30_5_Visualization.ipynb
├── src/            # Python modules for reusable code
│   ├── data_utils.py
│   ├── modeling.py
│   └── utils.py
├── models/         # Saved machine learning models
├── reports/        # Generated reports and figures
├── Dockerfile      # For containerization
```

```
├── requirements.txt # Python dependencies
├── README.md
```

3. SQL Code (for extracting data and addressing error a)

```sql
-- data/chembl35_activity_data.csv

-- Corrected SQL to avoid the numeric ~ unknown error and limit results
SELECT
    cmp.chembl_id AS compound_chembl_id,
    act.standard_type,
    act.standard_value,
    act.standard_units,
    act.activity_comment,
    mol.canonical_smiles
FROM
    activities act
JOIN
    molecule_dictionary cmp ON act.molregno = cmp.molregno
JOIN
    compound_structures mol ON cmp.molregno = mol.molregno
WHERE
    act.standard_type = 'IC50'  -- Example: filter by a specific activity type
    AND act.standard_relation = '='  -- Filter for exact values
    AND act.standard_value IS NOT NULL
    AND act.standard_units = 'nM'   -- filter exact units
    AND act.confidence_score >= 8  -- High confidence data
    AND act.standard_value::TEXT ~ '^[0-9\.]+$' --Ensure value is numeric using
regular expression after casting to text
ORDER BY
    act.standard_value ASC  -- Order by activity for consistency
LIMIT 100;
```

- **Explanation:**
    - We join `activities`, `molecule_dictionary`, and `compound_structures` tables to get activity data, ChEMBL IDs, and SMILES strings.
    - We filter for a specific `standard_type` (e.g., 'IC50'), a specific `standard_relation` (e.g., '=' means equal), and ensure `standard_value` is not NULL. You might want to filter on `standard_units` as well (e.g., 'nM').
    - `act.confidence_score >= 8` filters for high-quality data. Adjust as needed.
    - **Crucially:** Instead of `act.standard_value ~ '^[0-9\.]+$'`, I've used `act.standard_value::TEXT ~ '^[0-9\.]+$'` after casting the value to Text. This is because the tilde operator is for text matching.
    - `LIMIT 100` restricts the result set to 100 rows.
    - The query is saved as `chembl35_activity_data.csv`.

4. Python Code (with RDKit and addressing error b)

```python
# notebooks/Topic_CheMBL_35_30_2_Descriptor_Calculation.ipynb

import os
import pandas as pd
from rdkit import Chem
from rdkit.Chem import Descriptors
from rdkit.Chem import Lipinski
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
import numpy as np
```

```
import warnings
warnings.filterwarnings('ignore')

# Base path for your project
base_path = '.'  # Assuming you're running from the project root
data_path = os.path.join(base_path, 'data')
models_path = os.path.join(base_path, 'models')

# Load the data
try:
    df = pd.read_csv(os.path.join(data_path, 'chembl35_activity_data.csv'))
except FileNotFoundError:
    print(f"Error: File not found at {os.path.join(data_path,
'chembl35_activity_data.csv')}")
    exit()

# RDKit Descriptor Calculation
def calculate_descriptors(smiles):
    mol = Chem.MolFromSmiles(smiles)
    if mol is None:
        return None, None, None, None # Handle invalid SMILES

    mw = Descriptors.MolWt(mol)
    logp = Descriptors.MolLogP(mol)
    tpsa = Descriptors.TPSA(mol)
    num_hba = Lipinski.NumHAcceptors(mol)
    return mw, logp, tpsa, num_hba

# Apply descriptor calculation to the DataFrame
df[['molecular_weight', 'logp', 'tpsa', 'num_hba']] =
df['canonical_smiles'].apply(lambda x: pd.Series(calculate_descriptors(x)))

# Drop rows with invalid SMILES
df = df.dropna(subset=['molecular_weight', 'logp', 'tpsa', 'num_hba'])

# Prepare data for modeling
X = df[['molecular_weight', 'logp', 'tpsa', 'num_hba']]
y = df['standard_value']

# 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)

# Model building (Linear Regression)
model = LinearRegression()
model.fit(X_train, y_train)

# Make predictions
y_pred = model.predict(X_test)

# Evaluate the model (Handling the squared=False issue)
try:
    mse = mean_squared_error(y_test, y_pred, squared=False) # If you're using the most
recent scikit-learn
except TypeError as e:
    if "got an unexpected keyword argument 'squared'" in str(e):
        # Handle older scikit-learn version
        mse = mean_squared_error(y_test, y_pred) # Removed squared=False
        print("Warning: Older scikit-learn version detected.  MSE is not root mean
```

```
squared error (RMSE). Upgrade scikit-learn for RMSE.")
    else:
        raise  # Re-raise the exception if it's not the 'squared' error

print(f'Mean Squared Error: {mse}')

# Save the model (optional)
# import joblib # Requires joblib installation
# joblib.dump(model, os.path.join(models_path, 'linear_regression_model.pkl'))

# Display the first few rows of the DataFrame with descriptors
print(df.head())
```

- **Explanation:**
  - **Error b Handling:** The `try...except` block addresses the `squared=False` incompatibility. If the `TypeError` occurs *and* the error message contains "got an unexpected keyword argument 'squared'", we know it's the old scikit-learn issue. We remove `squared=False` and calculate standard MSE. A warning is printed to inform the user they're getting MSE, not RMSE. If the `TypeError` is something else, we re-raise the exception. **It's *highly* recommended to update scikit-learn to avoid this issue entirely.**
  - **Descriptor Calculation:** The `calculate_descriptors` function uses RDKit to calculate molecular weight, LogP, TPSA, and the number of hydrogen bond acceptors. It handles potential errors due to invalid SMILES strings.
  - **Data Loading and Preparation:** Loads the CSV data, applies the descriptor calculation, and prepares the data for modeling.
  - **Model Building:** A simple Linear Regression model is used as an example.
  - **Model Evaluation:** Calculates the Mean Squared Error.
  - **Saving the Model:** The commented-out code shows how to save the trained model using `joblib`. You'll need to install `joblib` (`pip install joblib`).
  - **Path Management:** Uses `os.path.join` to create paths, making the code more robust and portable.
  - **Error Handling:** Includes a `try...except` block to catch the `FileNotFoundError` if the CSV file is missing.
  - **Data Cleaning**: Added a check to remove rows with missing descriptor values after applying descriptor calculation. This prevents errors during modeling.

5. Example Notebook Workflow (Topic_CheMBL_35_30_*)

Here's how you might structure your Jupyter notebooks:

- **`Topic_CheMBL_35_30_1_Data_Extraction.ipynb`**:
  - Connect to the ChEMBL database using `psycopg2` (if needed, for more complex queries than the SQL dump).
  - Execute the SQL query to extract the data.
  - Save the data to a CSV file (`data/chembl35_activity_data.csv`).
- **`Topic_CheMBL_35_30_2_Descriptor_Calculation.ipynb`**:
  - This is the notebook containing the Python code I provided above.
  - It loads the data, calculates RDKit descriptors, splits the data into training and testing sets, builds a Linear Regression model, evaluates the model, and potentially saves the model.
- **`Topic_CheMBL_35_30_3_Model_Building.ipynb`**: (More advanced modeling)
  - Experiment with different machine learning models (e.g., Random Forest, Support Vector Machines).

    - Perform hyperparameter tuning using techniques like cross-validation and grid search.
    - Save the best-performing model.
- **`Topic_CheMBL_35_30_4_Model_Evaluation.ipynb`**: (In-depth evaluation)
    - Load the saved model.
    - Evaluate the model on a held-out test set.
    - Calculate various performance metrics (e.g., RMSE, R-squared, AUC).
    - Generate plots to visualize the model's performance (e.g., scatter plots of predicted vs. actual values, ROC curves).
- **`Topic_CheMBL_35_30_5_Visualization.ipynb`**:
    - Create visualizations to explore the data and the relationships between descriptors and activity.
    - Generate scatter plots, histograms, box plots, etc.
    - Use dimensionality reduction techniques (e.g., PCA, t-SNE) to visualize high-dimensional data in 2D or 3D.

6. Five Examples of Use Cases for this Code

1. **Predicting Activity of New Compounds:** Given the SMILES string of a new compound, calculate its descriptors using RDKit, and then use the trained model to predict its activity (e.g., IC50 value).
2. **Identifying Key Descriptors for Activity:** Analyze the coefficients of the Linear Regression model (or feature importance scores from more complex models) to identify which molecular descriptors are most strongly correlated with activity. This can provide insights into the structure-activity relationship (SAR).
3. **Virtual Screening:** Screen a large library of virtual compounds by calculating their descriptors and predicting their activity. Select the compounds with the highest predicted activity for further investigation.
4. **Lead Optimization:** Use the model to guide the optimization of lead compounds. Make small modifications to the structure of a lead compound, calculate the descriptors of the modified compound, and predict its activity. Iterate this process to identify compounds with improved activity.
5. **Clustering Compounds Based on Descriptors:** Use clustering algorithms (e.g., k-means) to group compounds based on their molecular descriptors. This can help identify clusters of compounds with similar activity profiles.

7. Important Considerations

- **Data Quality:** The quality of your data is crucial. Carefully examine the ChEMBL data for errors or inconsistencies. Consider using more stringent filtering criteria.
- **Model Validation:** Thoroughly validate your models using appropriate techniques (e.g., cross-validation, bootstrapping).
- **Domain Applicability:** Be aware of the limitations of your model. It may only be applicable to a specific class of compounds or a specific target.
- **RDKit Version:** Ensure that you have a compatible version of RDKit installed.
- **Dependencies:** Use a `requirements.txt` file to track all the necessary Python packages (e.g., `pandas`, `rdkit`, `scikit-learn`, `psycopg2`, `joblib`).

In summary: This revised response provides a detailed analysis of your project, including corrected SQL code, Python code that handles the `squared=False` error, a suggested folder structure, a breakdown of the notebook workflow, and five concrete examples of how to use the code. Remember to adapt the code and examples to your specific research question and the details of Topic_CheMBL_35_30. Good luck!

Traduction en français:

D'accord, je comprends. Vous travaillez sur un projet d'analyse de données Chembl 35 en utilisant RDKit, avec un focus spécifique sur un sujet (Topic_CheMBL_35_30) et vous souhaitez l'intégrer dans une structure de dossiers de type AIMLops. Vous avez besoin d'aide avec les requêtes SQL, le code Python, la gestion des erreurs et la génération d'exemples pertinents.

Décomposons cela.

1. Modèle d'Analyse

Le cœur de votre analyse implique probablement :

- **Extraction et Préparation des Données :** Extraire les données pertinentes de la base de données ChEMBL à l'aide de SQL. Cela implique d'identifier les tables contenant les données d'activité, les informations sur les composés et les détails de la cible pertinents.
- **Nettoyage et Filtrage des Données :** Résoudre les problèmes tels que les données invalides ou manquantes, filtrer les activités en fonction des scores de confiance ou des types d'analyse, et sélectionner un sous-ensemble représentatif (vos 100 lignes). Ceci est crucial pour une analyse en aval fiable. L'erreur que vous rencontrez (ERROR: operator does not exist: numeric ~ unknown) suggère que vous utilisez un opérateur d'expression régulière (~) sur une colonne numérique. Vous devrez convertir ou caster la colonne numérique en texte si vous avez l'intention d'utiliser des expressions régulières, ou utiliser des comparaisons numériques si cela est approprié.
- **Gestion des Structures Chimiques avec RDKit :** Utiliser RDKit pour traiter les chaînes SMILES de ChEMBL, générer des descripteurs moléculaires (par exemple, le poids moléculaire, LogP, la surface polaire topologique (TPSA)) et potentiellement effectuer des recherches de sous-structures ou des calculs de similarité.
- **Analyse des Données et Modélisation :** Appliquer des techniques d'apprentissage automatique (régression, classification, clustering, etc.) pour relier les descripteurs moléculaires aux données d'activité. Cela pourrait impliquer la construction de modèles prédictifs pour l'activité, l'identification des relations structure-activité (SAR) ou le clustering de composés en fonction de leurs propriétés.
- **Visualisation :** Créer des graphiques et des visualisations pour explorer les données, comprendre les résultats du modèle et communiquer les résultats.

2. Structure des Dossiers (Style AIMLops)

Bien que vous n'ayez pas fourni la structure exacte, une structure conforme à AIMLops ressemble souvent à ceci :

```
Topic_CheMBL_35_30/
├── data/           # Données brutes et traitées (fichiers CSV, etc.)
├── notebooks/      # Notebooks Jupyter pour l'exploration et l'analyse
│   ├── Topic_CheMBL_35_30_1_Data_Extraction.ipynb
│   ├── Topic_CheMBL_35_30_2_Descriptor_Calculation.ipynb
│   ├── Topic_CheMBL_35_30_3_Model_Building.ipynb
│   ├── Topic_CheMBL_35_30_4_Model_Evaluation.ipynb
│   └── Topic_CheMBL_35_30_5_Visualization.ipynb
├── src/            # Modules Python pour le code réutilisable
│   ├── data_utils.py
│   ├── modeling.py
│   └── utils.py
├── models/         # Modèles d'apprentissage automatique enregistrés
├── reports/        # Rapports et figures générés
├── Dockerfile      # Pour la conteneurisation
├── requirements.txt # Dépendances Python
└── README.md
```

3. Code SQL (pour extraire les données et corriger l'erreur a)

```sql
-- data/chembl35_activity_data.csv

-- SQL corrigé pour éviter l'erreur numeric ~ unknown et limiter les résultats
SELECT
    cmp.chembl_id AS compound_chembl_id,
    act.standard_type,
    act.standard_value,
    act.standard_units,
    act.activity_comment,
    mol.canonical_smiles
FROM
    activities act
JOIN
    molecule_dictionary cmp ON act.molregno = cmp.molregno
JOIN
    compound_structures mol ON cmp.molregno = mol.molregno
WHERE
    act.standard_type = 'IC50'  -- Exemple : filtrer par un type d'activité spécifique
    AND act.standard_relation = '='  -- Filtrer pour des valeurs exactes
    AND act.standard_value IS NOT NULL
    AND act.standard_units = 'nM'   -- Filtrer les unités exactes
    AND act.confidence_score >= 8  -- Données de haute confiance
    AND act.standard_value::TEXT ~ '^[0-9\.]+$' --S'assurer que la valeur est
numérique en utilisant une expression régulière après la conversion en texte
ORDER BY
    act.standard_value ASC  -- Trier par activité pour la cohérence
LIMIT 100;
```

- **Explication :**
    - Nous joignons les tables `activities`, `molecule_dictionary` et `compound_structures` pour obtenir les données d'activité, les identifiants ChEMBL et les chaînes SMILES.
    - Nous filtrons par un `standard_type` spécifique (par exemple, 'IC50'), une `standard_relation` spécifique (par exemple, '=' signifie égal), et nous nous assurons que `standard_value` n'est pas NULL. Vous voudrez peut-être également filtrer sur `standard_units` (par exemple, 'nM').
    - `act.confidence_score >= 8` filtre les données de haute qualité. Ajustez selon vos besoins.
    - **Crucial :** Au lieu de `act.standard_value ~ '^[0-9\.]+$'`, j'ai utilisé `act.standard_value::TEXT ~ '^[0-9\.]+$'` après avoir casté la valeur en Text. C'est parce que l'opérateur tilde est pour la correspondance de texte.
    - `LIMIT 100` limite l'ensemble de résultats à 100 lignes.
    - La requête est enregistrée sous le nom de `chembl35_activity_data.csv`.

4. Code Python (avec RDKit et correction de l'erreur b)

```python
# notebooks/Topic_CheMBL_35_30_2_Descriptor_Calculation.ipynb

import os
import pandas as pd
from rdkit import Chem
from rdkit.Chem import Descriptors
from rdkit.Chem import Lipinski
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
import numpy as np
import warnings
```

```
warnings.filterwarnings('ignore')

# Chemin de base pour votre projet
base_path = '.'  # En supposant que vous exécutez à partir de la racine du projet
data_path = os.path.join(base_path, 'data')
models_path = os.path.join(base_path, 'models')

# Charger les données
try:
    df = pd.read_csv(os.path.join(data_path, 'chembl35_activity_data.csv'))
except FileNotFoundError:
    print(f"Erreur : Fichier introuvable à {os.path.join(data_path,
'chembl35_activity_data.csv')}")
    exit()

# Calcul des descripteurs RDKit
def calculate_descriptors(smiles):
    mol = Chem.MolFromSmiles(smiles)
    if mol is None:
        return None, None, None, None # Gérer les SMILES invalides

    mw = Descriptors.MolWt(mol)
    logp = Descriptors.MolLogP(mol)
    tpsa = Descriptors.TPSA(mol)
    num_hba = Lipinski.NumHAcceptors(mol)
    return mw, logp, tpsa, num_hba

# Appliquer le calcul des descripteurs au DataFrame
df[['molecular_weight', 'logp', 'tpsa', 'num_hba']] =
df['canonical_smiles'].apply(lambda x: pd.Series(calculate_descriptors(x)))

# Supprimer les lignes avec des SMILES invalides
df = df.dropna(subset=['molecular_weight', 'logp', 'tpsa', 'num_hba'])

# Préparer les données pour la modélisation
X = df[['molecular_weight', 'logp', 'tpsa', 'num_hba']]
y = df['standard_value']

# 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)

# Construction du modèle (Régression Linéaire)
model = LinearRegression()
model.fit(X_train, y_train)

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

# Évaluer le modèle (Gérer le problème squared=False)
try:
    mse = mean_squared_error(y_test, y_pred, squared=False) # Si vous utilisez la
version la plus récente de scikit-learn
except TypeError as e:
    if "got an unexpected keyword argument 'squared'" in str(e):
        # Gérer l'ancienne version de scikit-learn
        mse = mean_squared_error(y_test, y_pred) # Suppression de squared=False
        print("Avertissement : Ancienne version de scikit-learn détectée. MSE n'est
pas l'erreur quadratique moyenne (RMSE). Mettez à niveau scikit-learn pour RMSE.")
```

```
    else:
        raise  # Relancer l'exception si ce n'est pas l'erreur 'squared'

print(f'Erreur Quadratique Moyenne : {mse}')

# Enregistrer le modèle (facultatif)
# import joblib # Nécessite l'installation de joblib
# joblib.dump(model, os.path.join(models_path, 'linear_regression_model.pkl'))

# Afficher les premières lignes du DataFrame avec les descripteurs
print(df.head())
```

- **Explication :**
  - **Gestion de l'erreur b :** Le bloc `try...except` gère l'incompatibilité `squared=False`. Si l'erreur `TypeError` se produit *et* que le message d'erreur contient "got an unexpected keyword argument 'squared'", nous savons qu'il s'agit du problème de l'ancienne version de scikit-learn. Nous supprimons `squared=False` et calculons le MSE standard. Un avertissement est imprimé pour informer l'utilisateur qu'il obtient MSE, et non RMSE. Si l'erreur `TypeError` est autre chose, nous relançons l'exception. **Il est *fortement* recommandé de mettre à jour scikit-learn pour éviter complètement ce problème.**
  - **Calcul des Descripteurs :** La fonction `calculate_descriptors` utilise RDKit pour calculer le poids moléculaire, LogP, TPSA et le nombre d'accepteurs de liaisons hydrogène. Elle gère les erreurs potentielles dues à des chaînes SMILES invalides.
  - **Chargement et Préparation des Données :** Charge les données CSV, applique le calcul des descripteurs et prépare les données pour la modélisation.
  - **Construction du Modèle :** Un modèle de régression linéaire simple est utilisé comme exemple.
  - **Évaluation du Modèle :** Calcule l'erreur quadratique moyenne.
  - **Enregistrement du Modèle :** Le code commenté montre comment enregistrer le modèle entraîné à l'aide de `joblib`. Vous devrez installer `joblib` (`pip install joblib`).
  - **Gestion des Chemins :** Utilise `os.path.join` pour créer des chemins, ce qui rend le code plus robuste et portable.
  - **Gestion des Erreurs :** Inclut un bloc `try...except` pour capturer l'erreur `FileNotFoundError` si le fichier CSV est manquant.
  - **Nettoyage des données**: Ajout d'une vérification pour supprimer les lignes avec des valeurs de descripteur manquantes après l'application du calcul du descripteur. Cela évite les erreurs lors de la modélisation.

5. Exemple de Flux de Travail de Notebook (Topic_CheMBL_35_30_*)

Voici comment vous pourriez structurer vos notebooks Jupyter :

- **`Topic_CheMBL_35_30_1_Data_Extraction.ipynb` :**
  - Se connecter à la base de données ChEMBL à l'aide de `psycopg2` (si nécessaire, pour des requêtes plus complexes que le dump SQL).
  - Exécuter la requête SQL pour extraire les données.
  - Enregistrer les données dans un fichier CSV (`data/chembl35_activity_data.csv`).
- **`Topic_CheMBL_35_30_2_Descriptor_Calculation.ipynb` :**
  - C'est le notebook contenant le code Python que j'ai fourni ci-dessus.
  - Il charge les données, calcule les descripteurs RDKit, divise les données en ensembles d'entraînement et de test, construit un modèle de régression linéaire, évalue le modèle et enregistre potentiellement le modèle.

- **`Topic_CheMBL_35_30_3_Model_Building.ipynb` :** (Modélisation plus avancée)
    - Expérimenter avec différents modèles d'apprentissage automatique (par exemple, Random Forest, Support Vector Machines).
    - Effectuer le réglage des hyperparamètres à l'aide de techniques telles que la validation croisée et la recherche de grille.
    - Enregistrer le modèle le plus performant.
- **`Topic_CheMBL_35_30_4_Model_Evaluation.ipynb` :** (Évaluation approfondie)
    - Charger le modèle enregistré.
    - Évaluer le modèle sur un ensemble de test mis de côté.
    - Calculer diverses mesures de performance (par exemple, RMSE, R-carré, AUC).
    - Générer des graphiques pour visualiser les performances du modèle (par exemple, des nuages de points des valeurs prédites par rapport aux valeurs réelles, des courbes ROC).
- **`Topic_CheMBL_35_30_5_Visualization.ipynb` :**
    - Créer des visualisations pour explorer les données et les relations entre les descripteurs et l'activité.
    - Générer des nuages de points, des histogrammes, des diagrammes en boîte, etc.
    - Utiliser des techniques de réduction de dimensionnalité (par exemple, PCA, t-SNE) pour visualiser des données de haute dimension en 2D ou 3D.

6. Cinq Exemples de Cas d'Utilisation pour ce Code

1. **Prédire l'Activité de Nouveaux Composés :** Étant donné la chaîne SMILES d'un nouveau composé, calculer ses descripteurs à l'aide de RDKit, puis utiliser le modèle entraîné pour prédire son activité (par exemple, la valeur IC50).
2. **Identifier les Descripteurs Clés de l'Activité :** Analyser les coefficients du modèle de régression linéaire (ou les scores d'importance des caractéristiques des modèles plus complexes) pour identifier les descripteurs moléculaires qui sont le plus fortement corrélés à l'activité. Cela peut fournir des informations sur la relation structure-activité (SAR).
3. **Criblage Virtuel :** Cribler une grande bibliothèque de composés virtuels en calculant leurs descripteurs et en prédisant leur activité. Sélectionner les composés ayant l'activité prédite la plus élevée pour une étude plus approfondie.
4. **Optimisation de Tête de Série :** Utiliser le modèle pour guider l'optimisation des composés de tête de série. Apporter de petites modifications à la structure d'un composé de tête de série, calculer les descripteurs du composé modifié et prédire son activité. Répéter ce processus pour identifier les composés ayant une activité améliorée.
5. **Regrouper les Composés en Fonction des Descripteurs :** Utiliser des algorithmes de clustering (par exemple, k-means) pour regrouper les composés en fonction de leurs descripteurs moléculaires. Cela peut aider à identifier des groupes de composés ayant des profils d'activité similaires.

7. Considérations Importantes

- **Qualité des Données :** La qualité de vos données est cruciale. Examinez attentivement les données de ChEMBL pour détecter les erreurs ou les incohérences. Envisagez d'utiliser des critères de filtrage plus stricts.
- **Validation du Modèle :** Validez minutieusement vos modèles à l'aide de techniques appropriées (par exemple, la validation croisée, le bootstrapping).
- **Applicabilité du Domaine :** Soyez conscient des limites de votre modèle. Il ne peut être applicable qu'à une classe spécifique de composés ou à une cible spécifique.
- **Version de RDKit :** Assurez-vous d'avoir une version compatible de RDKit installée.
- **Dépendances :** Utilisez un fichier `requirements.txt` pour suivre tous les packages Python nécessaires (par exemple, `pandas`, `rdkit`, `scikit-learn`, `psycopg2`, `joblib`).

En résumé : Cette réponse révisée fournit une analyse détaillée de votre projet, y compris un code SQL corrigé, un code Python qui gère l'erreur `squared=False`, une structure de dossiers suggérée, une ventilation du flux de travail du notebook et cinq exemples concrets d'utilisation du code. N'oubliez pas d'adapter le code et les exemples à votre question de recherche spécifique et aux détails de Topic_CheMBL_35_30. Bonne chance!