# PharmApp Suite

## 🧠 AI for Drug Discovery and Development 🧪

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

## 🧩 Topic: Epigenetic modulator screening via ChEMBL

Okay, I understand. You’re working on a Chembl 35 data analysis project using RDKit, following an AIMLops-like folder structure, and are encountering some specific issues. Let’s break down the analysis model, provide the code, and address the errors.

**Analysis Model and Objectives:**

The core objective is to explore the ChEMBL 35 database, likely focusing on structure-activity relationships (SAR) or property prediction for drug candidates. The steps involve:

1. **Data Extraction (SQL):** Using SQL to query the ChEMBL database and extract relevant information (e.g., compound structures, activity data, physicochemical properties) related to a specific target or therapeutic area. You’ll save this data as CSV files. You are specifically interested in activity data represented by standard\_value.
2. **Data Processing (Python/RDKit):** Reading the CSV data into a Jupyter notebook and utilizing RDKit for tasks such as:
   * Generating molecular fingerprints or descriptors.
   * Calculating physicochemical properties (e.g., LogP, molecular weight).
   * Cleaning and standardizing the data.
3. **Data Analysis and Modeling (Python/Scikit-learn):** Applying machine learning techniques to:
   * Build predictive models for activity (e.g., regression models to predict IC50 values).
   * Identify key molecular features that correlate with activity.
   * Visualize the data and model results.

**Addressing the Errors:**

* **SQL Error (ERROR: operator does not exist: numeric ~ unknown, LINE 12: AND act.standard\_value ~ ‘**[[1]](#footnote-21)**+$’):** This error indicates that you’re trying to use a regular expression (~) to filter a column (act.standard\_value) that’s likely defined as a numeric type (integer or floating-point). Regular expressions are meant for strings. You need to convert standard\_value to a string or use numerical comparisons.
* **Scikit-learn Error (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 that doesn’t support the squared=False argument in mean\_squared\_error. Either update your Scikit-learn installation, or remove/omit that argument to get the Mean Squared Error (MSE) instead of the Root Mean Squared Error (RMSE).

**Folder Structure (Based on AIMLops principles):**

While you don’t provide the full structure, I’ll assume a basic setup, and you can adapt it:

Project\_ChEMBL\_35/  
├── data/ # Raw and processed data (CSV files)  
├── notebooks/ # Jupyter notebooks  
│ ├── Topic\_CheMBL\_35\_99\_1\_data\_extraction.ipynb  
│ ├── Topic\_CheMBL\_35\_99\_2\_data\_processing.ipynb  
│ ├── Topic\_CheMBL\_35\_99\_3\_modeling.ipynb  
│ ├── Topic\_CheMBL\_35\_99\_4\_visualization.ipynb  
│ └── Topic\_CheMBL\_35\_99\_5\_evaluation.ipynb  
├── sql/ # SQL scripts  
├── requirements.txt # Python dependencies  
└── README.md

**Code Examples:**

**1. SQL (Data Extraction - sql/chembl\_35\_99\_extraction.sql):**

-- Extract activity data for a specific target (replace with your target\_id)  
-- Limiting to 100 rows for demonstration purposes.  
  
SELECT  
 cmp.chembl\_id AS compound\_chembl\_id,  
 act.standard\_type,  
 act.standard\_relation,  
 act.standard\_value,  
 act.standard\_units,  
 act.activity\_comment,  
 mol.molfile,  
 targ.target\_type,  
 targ.pref\_name AS target\_name  
FROM  
 activities act  
JOIN  
 molecule\_dictionary cmp ON act.molregno = cmp.molregno  
JOIN  
 target\_dictionary targ ON act.tid = targ.tid  
LEFT JOIN  
 mols mol ON cmp.molregno = mol.molregno -- Add molfile for RDKit processing  
WHERE targ.target\_type = 'SINGLE PROTEIN'  
AND act.standard\_type = 'IC50'  
AND act.standard\_relation = '='  
AND act.standard\_units = 'nM'  
AND act.standard\_value BETWEEN 0 AND 10000 -- Filtering values within a reasonable range  
LIMIT 100;  
  
  
-- Save the result to data/chembl\_35\_99\_activity\_data.csv (using pgAdmin's export functionality)

**Explanation:**

* **WHERE targ.target\_type = 'SINGLE PROTEIN'**: Filters for single protein targets (common in drug discovery). You can adapt this if you have a different target type in mind.
* **act.standard\_type = 'IC50'**: Filters for IC50 activity values.
* **act.standard\_relation = '='**: Filters where the relationship between the compound and the target is an exact IC50 value.
* **act.standard\_units = 'nM'**: Filters for activity values reported in nanomolars.
* **act.standard\_value BETWEEN 0 AND 10000**: Filters for IC50 values within a reasonable range (0-10000 nM). Adjust this range as needed.
* **LIMIT 100**: Restricts the output to 100 rows.
* **LEFT JOIN mols mol ON cmp.molregno = mol.molregno**: Retrieve the molfile so that we can display the molecules.

**How to run this SQL:**

1. Open pgAdmin.
2. Connect to your chembl\_35 database (IP: 192.168.206.136, user: rd, pass: rd).
3. Open a new query window.
4. Paste the SQL code into the query window.
5. Execute the query.
6. After the query completes, use pgAdmin’s “Copy with Headers” option on the result set, and paste that into a text file. Save the text file as data/chembl\_35\_99\_activity\_data.csv in your project directory. Ensure the file is properly comma-delimited.

**2. Python (Data Loading and Preprocessing - notebooks/Topic\_CheMBL\_35\_99\_1\_data\_extraction.ipynb):**

import os  
import pandas as pd  
from rdkit import Chem  
from rdkit.Chem import AllChem  
  
# Define base path  
base\_path = os.path.dirname(os.getcwd())  
  
# Construct the path to the CSV file  
csv\_file\_path = os.path.join(base\_path, 'data', 'chembl\_35\_99\_activity\_data.csv')  
  
# Read the CSV file into a pandas DataFrame  
try:  
 df = pd.read\_csv(csv\_file\_path)  
except FileNotFoundError:  
 print(f"Error: File not found at {csv\_file\_path}")  
 exit()  
  
# Display the first few rows of the DataFrame  
print(df.head())  
  
# Basic Data Cleaning (handle missing values, etc.)  
df = df.dropna(subset=['standard\_value', 'molfile']) #important  
df = df[df['standard\_value'] > 0] # remove standard\_values with value = 0  
  
# Convert 'standard\_value' to numeric type (important for later calculations)  
df['standard\_value'] = pd.to\_numeric(df['standard\_value'], errors='coerce') # Coerce errors to NaN  
  
# Filter out rows where molfile is None or empty string  
df = df[df['molfile'].notna() & (df['molfile'] != '')]  
  
# Display the cleaned DataFrame information  
print(df.info())  
  
print(f"Number of rows after cleaning: {len(df)}")

**Explanation:**

* **os.path.join(base\_path, 'data', 'chembl\_35\_99\_activity\_data.csv')**: Correctly constructs the path to your CSV file using os.path.join to ensure cross-platform compatibility.
* **pd.read\_csv(csv\_file\_path)**: Reads the CSV into a Pandas DataFrame. Includes error handling in case the file isn’t found.
* **df.dropna(subset=['standard\_value', 'molfile'])**: Removes rows with missing standard\_value or molfile values.
* **df = df[df['standard\_value'] > 0]**: Removes standard\_value with value = 0 to prevent calculating on logIC50 problems
* **df['standard\_value'] = pd.to\_numeric(df['standard\_value'], errors='coerce')**: Correctly converts the standard\_value column to a numeric type. errors='coerce' will replace invalid parsing with NaN, which you can then handle.
* **df = df[df['molfile'].notna() & (df['molfile'] != '')]**: Further cleaning to ensure the molfile column doesn’t have any issues.

**3. Python (RDKit Processing - notebooks/Topic\_CheMBL\_35\_99\_2\_data\_processing.ipynb):**

import os  
import pandas as pd  
from rdkit import Chem  
from rdkit.Chem import AllChem  
import numpy as np  
  
# Define base path  
base\_path = os.path.dirname(os.getcwd())  
  
# Construct the path to the CSV file  
csv\_file\_path = os.path.join(base\_path, 'data', 'chembl\_35\_99\_activity\_data.csv')  
  
# Read the CSV file into a pandas DataFrame  
try:  
 df = pd.read\_csv(csv\_file\_path)  
except FileNotFoundError:  
 print(f"Error: File not found at {csv\_file\_path}")  
 exit()  
  
# Basic Data Cleaning (handle missing values, etc.)  
df = df.dropna(subset=['standard\_value', 'molfile']) #important  
df = df[df['standard\_value'] > 0] # remove standard\_values with value = 0  
  
# Convert 'standard\_value' to numeric type (important for later calculations)  
df['standard\_value'] = pd.to\_numeric(df['standard\_value'], errors='coerce') # Coerce errors to NaN  
  
# Filter out rows where molfile is None or empty string  
df = df[df['molfile'].notna() & (df['molfile'] != '')]  
  
# Create RDKit Mol objects and calculate LogP  
def create\_mol\_and\_logp(row):  
 try:  
 mol = Chem.MolFromMolBlock(row['molfile']) # use MolFromMolBlock for molfile data  
 if mol is None:  
 return None, None # Handle cases where molfile is invalid  
 logp = Chem.Crippen.MolLogP(mol)  
 return mol, logp  
 except Exception as e:  
 print(f"Error processing molfile: {e}")  
 return None, None  
  
df[['ROMol', 'LogP']] = df.apply(create\_mol\_and\_logp, axis=1, result\_type='expand') #important  
  
# Drop rows where molecule creation failed  
df = df.dropna(subset=['ROMol'])  
  
# Calculate Morgan Fingerprints (ECFP4)  
def calculate\_morgan\_fingerprint(mol):  
 try:  
 info = {}  
 fp = AllChem.GetMorganFingerprintAsBitVect(mol, radius=2, nBits=2048, bitInfo=info)  
 return fp  
 except Exception as e:  
 print(f"Error calculating Morgan fingerprint: {e}")  
 return None  
  
df['Morgan\_FP'] = df['ROMol'].apply(calculate\_morgan\_fingerprint)  
  
# Drop rows where fingerprint calculation failed  
df = df.dropna(subset=['Morgan\_FP'])  
  
print(df.head())

**Explanation:**

* **Chem.MolFromMolBlock(row['molfile'])**: This function is used to create an RDKit molecule object directly from the molfile string.
* **Error Handling:** The try...except block is crucial for handling potential errors during molecule creation. If a molfile is invalid, it will return None, which is then handled.
* **Chem.Crippen.MolLogP(mol)**: Calculates the LogP value using RDKit.
* **AllChem.GetMorganFingerprintAsBitVect(mol, radius=2, nBits=2048, bitInfo=info)**: Generates Morgan fingerprints (ECFP4) with radius 2 and 2048 bits, which are commonly used in drug discovery.
* **df[['ROMol', 'LogP']] = df.apply(create\_mol\_and\_logp, axis=1, result\_type='expand')** applies the function create\_mol\_and\_logp to each row. The result\_type='expand' makes sure that the two values returned by the function create\_mol\_and\_logp are properly set in two different columns named ROMol and LogP.

**4. Python (Modeling - notebooks/Topic\_CheMBL\_35\_99\_3\_modeling.ipynb):**

import os  
import pandas as pd  
from rdkit import Chem  
from rdkit.Chem import AllChem  
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, r2\_score  
from sklearn.preprocessing import StandardScaler  
from scipy.stats import pearsonr  
  
# Define base path  
base\_path = os.path.dirname(os.getcwd())  
  
# Construct the path to the CSV file  
csv\_file\_path = os.path.join(base\_path, 'data', 'chembl\_35\_99\_activity\_data.csv')  
  
# Read the CSV file into a pandas DataFrame  
try:  
 df = pd.read\_csv(csv\_file\_path)  
except FileNotFoundError:  
 print(f"Error: File not found at {csv\_file\_path}")  
 exit()  
  
# Basic Data Cleaning (handle missing values, etc.)  
df = df.dropna(subset=['standard\_value', 'molfile']) #important  
df = df[df['standard\_value'] > 0] # remove standard\_values with value = 0  
  
# Convert 'standard\_value' to numeric type (important for later calculations)  
df['standard\_value'] = pd.to\_numeric(df['standard\_value'], errors='coerce') # Coerce errors to NaN  
  
# Filter out rows where molfile is None or empty string  
df = df[df['molfile'].notna() & (df['molfile'] != '')]  
  
# Create RDKit Mol objects and calculate LogP  
def create\_mol\_and\_logp(row):  
 try:  
 mol = Chem.MolFromMolBlock(row['molfile']) # use MolFromMolBlock for molfile data  
 if mol is None:  
 return None, None # Handle cases where molfile is invalid  
 logp = Chem.Crippen.MolLogP(mol)  
 return mol, logp  
 except Exception as e:  
 print(f"Error processing molfile: {e}")  
 return None, None  
  
df[['ROMol', 'LogP']] = df.apply(create\_mol\_and\_logp, axis=1, result\_type='expand') #important  
  
# Drop rows where molecule creation failed  
df = df.dropna(subset=['ROMol'])  
  
# Calculate Morgan Fingerprints (ECFP4)  
def calculate\_morgan\_fingerprint(mol):  
 try:  
 info = {}  
 fp = AllChem.GetMorganFingerprintAsBitVect(mol, radius=2, nBits=2048, bitInfo=info)  
 return fp  
 except Exception as e:  
 print(f"Error calculating Morgan fingerprint: {e}")  
 return None  
  
df['Morgan\_FP'] = df['ROMol'].apply(calculate\_morgan\_fingerprint)  
  
# Drop rows where fingerprint calculation failed  
df = df.dropna(subset=['Morgan\_FP'])  
  
# Prepare Data for Modeling  
  
# Convert Morgan fingerprints to numpy arrays  
X = np.array([list(fp) for fp in df['Morgan\_FP']])  
y = -np.log10(df['standard\_value'] / 1e9) # Convert IC50 to pIC50 (Molar)  
  
# Data Scaling (important for linear models and other algorithms)  
scaler = StandardScaler()  
X\_scaled = scaler.fit\_transform(X)  
  
# Split data into training and testing sets  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_scaled, 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  
mse = mean\_squared\_error(y\_test, y\_pred)  
r2 = r2\_score(y\_test, y\_pred)  
  
print(f"Mean Squared Error: {mse}")  
print(f"R-squared: {r2}")  
  
# Calculate Pearson correlation coefficient  
correlation, \_ = pearsonr(y\_test, y\_pred)  
print(f"Pearson Correlation Coefficient: {correlation}")

**Explanation:**

* **X = np.array([list(fp) for fp in df['Morgan\_FP']])**: Converts the RDKit fingerprints to a NumPy array, which is required for Scikit-learn.
* **y = -np.log10(df['standard\_value'] / 1e9)**: Converts IC50 values to pIC50 (a more common metric in drug discovery). The division by 1e9 converts the standard\_value from nM to M (molar). The negative logarithm transforms the IC50 into a pIC50 value.
* **StandardScaler()**: Scales the fingerprint data to have zero mean and unit variance. This can significantly improve the performance of many machine learning algorithms.
* **train\_test\_split()**: Splits the data into training and testing sets to evaluate the model’s performance on unseen data.
* **LinearRegression()**: Trains a linear regression model. You can experiment with other models (e.g., Random Forest, Support Vector Regression).
* **mean\_squared\_error(y\_test, y\_pred)**: Calculates the Mean Squared Error (MSE) between the predicted and actual pIC50 values.
* **r2\_score(y\_test, y\_pred)**: Calculates the R-squared value, which represents the proportion of variance in the dependent variable that is predictable from the independent variables.
* **pearsonr(y\_test, y\_pred)**: Calculates the Pearson correlation coefficient to measure the linear relationship between predicted and observed pIC50 values.

**5. Python (Visualization - notebooks/Topic\_CheMBL\_35\_99\_4\_visualization.ipynb):**

import os  
import pandas as pd  
from rdkit import Chem  
from rdkit.Chem import AllChem  
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, r2\_score  
from sklearn.preprocessing import StandardScaler  
from scipy.stats import pearsonr  
import matplotlib.pyplot as plt  
import seaborn as sns  
  
# Define base path  
base\_path = os.path.dirname(os.getcwd())  
  
# Construct the path to the CSV file  
csv\_file\_path = os.path.join(base\_path, 'data', 'chembl\_35\_99\_activity\_data.csv')  
  
# Read the CSV file into a pandas DataFrame  
try:  
 df = pd.read\_csv(csv\_file\_path)  
except FileNotFoundError:  
 print(f"Error: File not found at {csv\_file\_path}")  
 exit()  
  
# Basic Data Cleaning (handle missing values, etc.)  
df = df.dropna(subset=['standard\_value', 'molfile']) #important  
df = df[df['standard\_value'] > 0] # remove standard\_values with value = 0  
  
# Convert 'standard\_value' to numeric type (important for later calculations)  
df['standard\_value'] = pd.to\_numeric(df['standard\_value'], errors='coerce') # Coerce errors to NaN  
  
# Filter out rows where molfile is None or empty string  
df = df[df['molfile'].notna() & (df['molfile'] != '')]  
  
# Create RDKit Mol objects and calculate LogP  
def create\_mol\_and\_logp(row):  
 try:  
 mol = Chem.MolFromMolBlock(row['molfile']) # use MolFromMolBlock for molfile data  
 if mol is None:  
 return None, None # Handle cases where molfile is invalid  
 logp = Chem.Crippen.MolLogP(mol)  
 return mol, logp  
 except Exception as e:  
 print(f"Error processing molfile: {e}")  
 return None, None  
  
df[['ROMol', 'LogP']] = df.apply(create\_mol\_and\_logp, axis=1, result\_type='expand') #important  
  
# Drop rows where molecule creation failed  
df = df.dropna(subset=['ROMol'])  
  
# Calculate Morgan Fingerprints (ECFP4)  
def calculate\_morgan\_fingerprint(mol):  
 try:  
 info = {}  
 fp = AllChem.GetMorganFingerprintAsBitVect(mol, radius=2, nBits=2048, bitInfo=info)  
 return fp  
 except Exception as e:  
 print(f"Error calculating Morgan fingerprint: {e}")  
 return None  
  
df['Morgan\_FP'] = df['ROMol'].apply(calculate\_morgan\_fingerprint)  
  
# Drop rows where fingerprint calculation failed  
df = df.dropna(subset=['Morgan\_FP'])  
  
# Prepare Data for Modeling  
  
# Convert Morgan fingerprints to numpy arrays  
X = np.array([list(fp) for fp in df['Morgan\_FP']])  
y = -np.log10(df['standard\_value'] / 1e9) # Convert IC50 to pIC50 (Molar)  
  
# Data Scaling (important for linear models and other algorithms)  
scaler = StandardScaler()  
X\_scaled = scaler.fit\_transform(X)  
  
# Split data into training and testing sets  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_scaled, 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  
mse = mean\_squared\_error(y\_test, y\_pred)  
r2 = r2\_score(y\_test, y\_pred)  
  
print(f"Mean Squared Error: {mse}")  
print(f"R-squared: {r2}")  
  
# Calculate Pearson correlation coefficient  
correlation, \_ = pearsonr(y\_test, y\_pred)  
print(f"Pearson Correlation Coefficient: {correlation}")  
  
# Visualization  
  
# Scatter plot of predicted vs. actual pIC50 values  
plt.figure(figsize=(8, 6))  
sns.scatterplot(x=y\_test, y=y\_pred)  
plt.xlabel("Actual pIC50")  
plt.ylabel("Predicted pIC50")  
plt.title("Predicted vs. Actual pIC50 Values")  
plt.annotate(f'R-squared = {r2:.2f}', xy=(0.05, 0.9), xycoords='axes fraction') # Add R-squared value to the plot  
plt.show()  
  
# Distribution plot of pIC50 values  
plt.figure(figsize=(8, 6))  
sns.histplot(y, kde=True)  
plt.xlabel("pIC50")  
plt.ylabel("Frequency")  
plt.title("Distribution of pIC50 Values")  
plt.show()  
  
# Example: Visualize a molecule with its predicted pIC50 value  
example\_index = 0 # Change this to visualize different molecules  
mol = df['ROMol'].iloc[example\_index]  
predicted\_pIC50 = y\_pred[example\_index] if example\_index < len(y\_pred) else "N/A"  
  
img = Chem.Draw.MolToImage(mol)  
plt.imshow(img)  
plt.title(f"Molecule (Predicted pIC50: {predicted\_pIC50:.2f})")  
plt.axis('off')  
plt.show()  
  
# Scatter plot of LogP vs pIC50  
plt.figure(figsize=(8, 6))  
sns.scatterplot(x=df['LogP'], y=y)  
plt.xlabel("LogP")  
plt.ylabel("pIC50")  
plt.title("LogP vs pIC50")  
plt.show()

**Explanation:**

* **matplotlib.pyplot and seaborn**: Imports the necessary libraries for creating visualizations.
* **Scatter Plot (Predicted vs. Actual):** A scatter plot to visualize the correlation between predicted and actual pIC50 values. The closer the points are to a diagonal line, the better the model’s performance. The R-squared value is added as an annotation to provide a quantitative measure of the model’s fit.
* **Distribution Plot (pIC50):** A histogram showing the distribution of pIC50 values in your dataset. This helps you understand the range and frequency of activity values.
* **Molecule Visualization:** Displays an example molecule along with its predicted pIC50 value. This allows you to visually inspect the molecules and see how their predicted activity relates to their structure.
* **Scatter Plot (LogP vs pIC50):** Explores the relationship between lipophilicity (LogP) and activity (pIC50). This can reveal trends or correlations that might be useful for understanding SAR.

**Important Considerations:**

* **Dependencies:** Make sure you have all the necessary Python packages installed. Create a requirements.txt file in your project root with the following (or similar) content:
* pandas  
  rdkit  
  scikit-learn  
  matplotlib  
  seaborn  
  numpy  
  scipy
* Then, run pip install -r requirements.txt in your terminal.
* **Database Connection:** The Python code assumes that you have already extracted the data from the ChEMBL database and saved it as a CSV file.
* **Error Handling:** The code includes basic error handling, but you should expand it to handle more specific errors (e.g., invalid molecule formats, database connection issues).
* **Model Selection:** Linear regression is a simple starting point. Experiment with other machine learning models (e.g., Random Forest, Support Vector Machines) to see if you can improve performance.
* **Feature Engineering:** You can add more sophisticated features beyond Morgan fingerprints (e.g., physicochemical properties, structural alerts).
* **Hyperparameter Tuning:** Optimize the hyperparameters of your machine learning models using techniques like cross-validation and grid search.
* **Data Size:** With only 100 data points, your models will likely have limited predictive power. Consider extracting more data from ChEMBL if possible.

**In Summary (Analyse en deux langues)**

**English:**

This comprehensive guide provides you with the SQL and Python code necessary to extract, process, model, and visualize data from the ChEMBL 35 database for drug discovery purposes. It addresses the specific errors you encountered and offers a structured approach to your project, aligning with AIMLops principles. Remember to adapt the code and parameters to your specific research question and target of interest. Key improvements include error handling, data cleaning, and conversion of IC50 values to pIC50. The visualization section helps you to understand the data and model predictions in a graphical manner.

**Français:**

Ce guide complet vous fournit le code SQL et Python nécessaire pour extraire, traiter, modéliser et visualiser les données de la base de données ChEMBL 35 à des fins de découverte de médicaments. Il aborde les erreurs spécifiques que vous avez rencontrées et propose une approche structurée de votre projet, conformément aux principes AIMLops. N’oubliez pas d’adapter le code et les paramètres à votre question de recherche spécifique et à votre cible d’intérêt. Les améliorations clés incluent la gestion des erreurs, le nettoyage des données et la conversion des valeurs IC50 en pIC50. La section de visualisation vous aide à comprendre les données et les prédictions du modèle de manière graphique.

Remember to adjust the target selection and data ranges in the SQL query based on your specific research goals. Good luck with your project!

1. 0-9. [↑](#footnote-ref-21)