# PharmApp Suite

## 🧠 AI for Drug Discovery and Development 🧪

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

## 🧩 Topic: Timeline of drug approval dates

Okay, I understand. You’re working with ChEMBL 35 data, using RDKit and PostgreSQL, and you need help structuring your code within an AIMLops framework, debugging errors, and creating a focused analysis for drug discovery. Let’s break this down.

**1. Analysis Model & Overall Strategy**

The core of your project likely involves exploring relationships within ChEMBL data. A common approach is to:

* **Data Extraction & Cleaning:** Use SQL queries to extract relevant activity data (IC50, Ki, etc.) and compound information (SMILES strings) from ChEMBL. This step will also filter data to ensure consistent units and data types. Addressing the "numeric ~ unknown" error will also fall into this category.
* **Molecular Feature Generation:** Use RDKit to convert SMILES strings into numerical representations of molecular properties (e.g., molecular weight, logP, number of hydrogen bond donors/acceptors, topological polar surface area - TPSA, fingerprints like Morgan fingerprints).
* **Data Exploration and Visualization:** Visualize the distribution of activity values, molecular properties, and relationships between them using libraries like Matplotlib or Seaborn.
* **Model Building (Optional):** If you have a specific target in mind, you could build predictive models relating molecular features to activity values (e.g., quantitative structure-activity relationship - QSAR models) using machine learning algorithms from scikit-learn.
* **Interpretation:** Translate your analysis into meaningful insights for drug discovery, like identifying key structural features associated with activity.

**2. AIMLops Structure & Code Location**

Following the AIMLops template, here’s how your project should be organized:

* **data/:** Contains raw data (CSV files generated from SQL queries) and potentially processed data (e.g., featurized data).
* **notebooks/:** Contains Jupyter notebooks with your code. File names should be like Topic\_CheMBL\_35\_80\_1\_Data\_Extraction.ipynb, Topic\_CheMBL\_35\_80\_2\_Feature\_Engineering.ipynb, etc.
* **src/ (Optional, but recommended for more complex projects):** Contains reusable Python modules for functions like data cleaning, feature generation, or model training. This promotes modularity and reusability.
* **models/ (Optional):** Stores trained machine learning models if you build any.

**3. Code (SQL & Python) with Error Handling**

Here’s the code addressing your issues:

**SQL (PostgreSQL)**

-- File: ../data/chembl\_35\_data\_100.csv (This is where the output will be saved)  
-- Adapted for Chembl 35. Important: Adjust table names if needed.  
-- Limiting to 100 results for demonstration.  
  
SELECT  
 cmp.chembl\_id,  
 cmp.pref\_name,  
 act.standard\_type,  
 act.standard\_relation,  
 act.standard\_value,  
 act.standard\_units,  
 act.activity\_comment,  
 mol.molecule\_structures  
FROM  
 compound\_structures mol  
JOIN  
 molecule\_dictionary cmp ON mol.molregno = cmp.molregno  
JOIN  
 activities act ON cmp.molregno = act.molregno  
WHERE  
 --Filtering to only grab rows with defined values and type  
 act.standard\_type IN ('IC50', 'Ki', 'Kd')  
 AND act.standard\_relation = '='  
 AND act.standard\_units = 'nM'  
 AND act.standard\_value IS NOT NULL  
 AND mol.molecule\_structures IS NOT NULL -- Get only rows where molecule\_structures are not empty  
 AND act.standard\_value::text ~ '^([0-9]+\\.?[0-9]\*|\\.[0-9]+)$' -- Ensure standard\_value is numeric, and converts before applying the check  
LIMIT 100;

**Explanation of SQL code:**

1. **Join Tables:** Joins compound\_structures, molecule\_dictionary, and activities tables based on molregno to retrieve molecule structures and activity data.
2. **Filter Activity Types:** Filters for activity types like IC50, Ki, and Kd, and standard relation ‘=’.
3. **Filter Units:** Filters for standard units in ‘nM’.
4. **Handle NULL Values:** Ensures that standard\_value and molecule\_structures are not NULL.
5. **Fix Numeric Check:** act.standard\_value::text ~ '^([0-9]+\\.?[0-9]\*|\\.[0-9]+)$'
   * This converts the standard\_value to text (::text).
   * Then, it uses regular expression matching (~) to check if the string consists only of digits and optionally a decimal point (allowing for values like “10”, “10.5”, “.5”). The regular expression ^([0-9]+\\.?[0-9]\*|\\.[0-9]+)$ means:
     + ^: Start of the string
     + ([0-9]+\\.?[0-9]\*|\\.[0-9]+): Either:
       - [0-9]+: One or more digits
       - \\.?: An optional decimal point (escaped with \)
       - [0-9]\*: Zero or more digits
     + |: OR
     + \\.[0-9]+: A decimal point followed by one or more digits
     + $: End of the string
6. **Limit Results:** Limits the output to 100 rows for demonstration purposes. Remove this line for the full dataset.

**How to run SQL and save to CSV:**

1. Open pgAdmin and connect to your database (chembl\_35 on 192.168.206.136 with user rd and password rd).
2. Open a new query window and paste the SQL code.
3. Execute the query.
4. Right-click on the results grid and choose “Copy All Rows”. Alternatively, use the “Export” function (if available in your pgAdmin version) to save directly to a CSV file.
5. Save the file as ../data/chembl\_35\_data\_100.csv. Ensure the path is correct relative to where you’re running pgAdmin.

**Python (Jupyter Notebook - Topic\_CheMBL\_35\_80\_1\_Data\_Loading.ipynb)**

import os  
import pandas as pd  
from rdkit import Chem  
from rdkit.Chem import Descriptors  
import numpy as np  
  
# Define the base path for your project  
base\_path = ".." # Assuming the notebook is in the 'notebooks' directory  
  
# Construct the path to the CSV file  
csv\_file\_path = os.path.join(base\_path, "data", "chembl\_35\_data\_100.csv")  
  
# Load the data using pandas  
try:  
 df = pd.read\_csv(csv\_file\_path)  
 print("Data loaded successfully.")  
 print(df.head()) # Display the first few rows  
except FileNotFoundError:  
 print(f"Error: File not found at {csv\_file\_path}")  
except Exception as e:  
 print(f"An error occurred while loading the data: {e}")  
  
# Function to calculate LogP (example feature)  
def calculate\_logp(smiles):  
 try:  
 mol = Chem.MolFromSmiles(smiles)  
 if mol:  
 logp = Descriptors.MolLogP(mol)  
 return logp  
 else:  
 return np.nan # Handle invalid SMILES  
 except:  
 return np.nan  
  
# Apply the function to the 'molecule\_structures' column (assuming this column has the SMILES)  
if 'molecule\_structures' in df.columns:  
 df['logp'] = df['molecule\_structures'].apply(calculate\_logp)  
 print(df[['molecule\_structures', 'logp']].head()) # Display SMILES and LogP  
else:  
 print("Error: 'molecule\_structures' column not found in the DataFrame.")

**Python (Jupyter Notebook - Topic\_CheMBL\_35\_80\_2\_Data\_Analysis.ipynb)**

import os  
import pandas as pd  
import matplotlib.pyplot as plt  
import seaborn as sns  
from sklearn.metrics import mean\_squared\_error  
from sklearn.preprocessing import MinMaxScaler  
from sklearn.model\_selection import train\_test\_split  
from sklearn.linear\_model import LinearRegression # Example model  
import numpy as np # Added import  
  
# Define the base path for your project  
base\_path = ".." # Assuming the notebook is in the 'notebooks' directory  
  
# Construct the path to the CSV file  
csv\_file\_path = os.path.join(base\_path, "data", "chembl\_35\_data\_100.csv")  
  
# Load the data using pandas  
try:  
 df = pd.read\_csv(csv\_file\_path)  
 print("Data loaded successfully.")  
 print(df.head()) # Display the first few rows  
except FileNotFoundError:  
 print(f"Error: File not found at {csv\_file\_path}")  
except Exception as e:  
 print(f"An error occurred while loading the data: {e}")  
  
# Convert standard\_value to numeric and handle errors  
df['standard\_value'] = pd.to\_numeric(df['standard\_value'], errors='coerce')  
df.dropna(subset=['standard\_value'], inplace=True) # Drop rows with NaN in standard\_value  
  
# Convert IC50, Ki, Kd to pIC50  
def convert\_to\_pIC50(value):  
 return -np.log10(value \* 1e-9) # Convert nM to M and then to pIC50  
  
df['pIC50'] = df['standard\_value'].apply(convert\_to\_pIC50)  
  
# Simple Data Analysis Example: Distribution of pIC50  
plt.figure(figsize=(8, 6))  
sns.histplot(df['pIC50'], kde=True)  
plt.title('Distribution of pIC50 values')  
plt.xlabel('pIC50')  
plt.ylabel('Frequency')  
plt.show()

**Explanation of Python Code:**

1. **Import Libraries:** Imports necessary libraries (pandas, RDKit, etc.).
2. **Define Paths:** Defines the base path and constructs the path to the CSV file using os.path.join. This is important for reproducibility and portability.
3. **Load Data:** Loads the CSV data into a pandas DataFrame. Includes error handling for FileNotFoundError and other potential exceptions.
4. **calculate\_logp(smiles) Function:**
   * Takes a SMILES string as input.
   * Uses Chem.MolFromSmiles() to create an RDKit molecule object.
   * If the molecule is successfully created, it calculates LogP using Descriptors.MolLogP().
   * Returns the LogP value. If there’s an error (e.g., invalid SMILES), it returns np.nan. This is important for handling problematic molecules.
5. **Apply the Function:** Applies the calculate\_logp function to the molecule\_structures column using df['molecule\_structures'].apply(calculate\_logp). This creates a new column called logp in your DataFrame.
6. **Error Handling:** Includes a check to ensure that the ‘molecule\_structures’ column exists before attempting to apply the function.
7. **Data Cleaning:** Uses pd.to\_numeric with errors='coerce' to convert the standard\_value column to numeric. This will convert any non-numeric values to NaN. Then, dropna removes rows with NaN in the standard\_value column.
8. **pIC50 conversion:** The provided code will now correctly convert standard\_value to pIC50.
9. **Visualization:** Creates a histogram of the pIC50 values using Seaborn.

**4. Error Resolution**

* **ERROR: operator does not exist: numeric ~ unknown:** This is due to a type mismatch in your SQL query. The ~ operator (regular expression matching) expects a string on the right-hand side, but act.standard\_value is likely a numeric type. The corrected SQL query above converts the numeric value to text before applying the regular expression.
* **old scikit-learn version does not support parameters squared=False in the mean\_squared\_error function:** This indicates that you’re using an older version of scikit-learn. There are two solutions:
  + **Upgrade scikit-learn:** The easiest solution is to upgrade your scikit-learn version to a more recent one (>=0.20). You can do this using pip: pip install --upgrade scikit-learn
  + **Remove squared=False:** If you can’t upgrade scikit-learn, simply remove the squared=False parameter from the mean\_squared\_error function call. The function will then return the Mean Squared Error (MSE) instead of the Root Mean Squared Error (RMSE).

**5. Examples (Analysis)**

Here are five analysis examples, building on the code above, demonstrating different aspects of drug discovery data analysis with ChEMBL data:

**Example 1: Basic Statistics of Activity Values**

import os  
import pandas as pd  
import numpy as np  
  
base\_path = ".."  
csv\_file\_path = os.path.join(base\_path, "data", "chembl\_35\_data\_100.csv")  
df = pd.read\_csv(csv\_file\_path)  
  
# Convert standard\_value to numeric and handle errors  
df['standard\_value'] = pd.to\_numeric(df['standard\_value'], errors='coerce')  
df.dropna(subset=['standard\_value'], inplace=True)  
  
# Calculate descriptive statistics  
print("Descriptive Statistics for Standard Value:")  
print(df['standard\_value'].describe())

This example calculates and prints descriptive statistics (mean, standard deviation, min, max, etc.) for the standard\_value column. This gives you a quick overview of the distribution of activity values.

**Example 2: Activity Type Distribution**

import os  
import pandas as pd  
import matplotlib.pyplot as plt  
import seaborn as sns  
  
base\_path = ".."  
csv\_file\_path = os.path.join(base\_path, "data", "chembl\_35\_data\_100.csv")  
df = pd.read\_csv(csv\_file\_path)  
  
# Plot the distribution of activity types (IC50, Ki, Kd)  
plt.figure(figsize=(8, 6))  
sns.countplot(x='standard\_type', data=df)  
plt.title('Distribution of Activity Types')  
plt.xlabel('Activity Type')  
plt.ylabel('Count')  
plt.show()

This example creates a bar plot showing the distribution of different activity types (IC50, Ki, Kd) in your dataset. This helps you understand the composition of your data.

**Example 3: LogP vs. Activity Scatter Plot**

import os  
import pandas as pd  
from rdkit import Chem  
from rdkit.Chem import Descriptors  
import matplotlib.pyplot as plt  
import seaborn as sns  
import numpy as np  
  
base\_path = ".."  
csv\_file\_path = os.path.join(base\_path, "data", "chembl\_35\_data\_100.csv")  
df = pd.read\_csv(csv\_file\_path)  
  
# Convert standard\_value to numeric and handle errors  
df['standard\_value'] = pd.to\_numeric(df['standard\_value'], errors='coerce')  
df.dropna(subset=['standard\_value'], inplace=True)  
  
# Calculate LogP (same function as before)  
def calculate\_logp(smiles):  
 try:  
 mol = Chem.MolFromSmiles(smiles)  
 if mol:  
 logp = Descriptors.MolLogP(mol)  
 return logp  
 else:  
 return np.nan # Handle invalid SMILES  
 except:  
 return np.nan  
  
if 'molecule\_structures' in df.columns:  
 df['logp'] = df['molecule\_structures'].apply(calculate\_logp)  
else:  
 print("Error: 'molecule\_structures' column not found in the DataFrame.")  
  
df.dropna(subset=['logp'], inplace=True) # Remove rows with NaN LogP  
  
# Convert IC50, Ki, Kd to pIC50  
def convert\_to\_pIC50(value):  
 return -np.log10(value \* 1e-9) # Convert nM to M and then to pIC50  
  
df['pIC50'] = df['standard\_value'].apply(convert\_to\_pIC50)  
  
# Scatter plot of LogP vs. pIC50  
plt.figure(figsize=(8, 6))  
sns.scatterplot(x='logp', y='pIC50', data=df)  
plt.title('LogP vs. pIC50')  
plt.xlabel('LogP')  
plt.ylabel('pIC50')  
plt.show()

This example calculates LogP for each molecule and then creates a scatter plot of LogP vs. pIC50. This helps you visualize the relationship between lipophilicity (LogP) and activity.

**Example 4: Distribution of Molecular Weight**

import os  
import pandas as pd  
from rdkit import Chem  
from rdkit.Chem import Descriptors  
import matplotlib.pyplot as plt  
import seaborn as sns  
import numpy as np  
  
base\_path = ".."  
csv\_file\_path = os.path.join(base\_path, "data", "chembl\_35\_data\_100.csv")  
df = pd.read\_csv(csv\_file\_path)  
  
def calculate\_mol\_wt(smiles):  
 try:  
 mol = Chem.MolFromSmiles(smiles)  
 if mol:  
 mol\_wt = Descriptors.MolWt(mol)  
 return mol\_wt  
 else:  
 return np.nan # Handle invalid SMILES  
 except:  
 return np.nan  
  
if 'molecule\_structures' in df.columns:  
 df['mol\_wt'] = df['molecule\_structures'].apply(calculate\_mol\_wt)  
else:  
 print("Error: 'molecule\_structures' column not found in the DataFrame.")  
  
df.dropna(subset=['mol\_wt'], inplace=True)  
  
# Histogram of Molecular Weight  
plt.figure(figsize=(8, 6))  
sns.histplot(df['mol\_wt'], kde=True)  
plt.title('Distribution of Molecular Weight')  
plt.xlabel('Molecular Weight (g/mol)')  
plt.ylabel('Frequency')  
plt.show()

This example calculates the molecular weight of each molecule and creates a histogram to visualize the distribution of molecular weights. This is important for understanding the physical properties of your compounds.

**Example 5: Comparing Activity Across Different Pref\_Names (Targets)**

import os  
import pandas as pd  
import matplotlib.pyplot as plt  
import seaborn as sns  
import numpy as np  
  
base\_path = ".."  
csv\_file\_path = os.path.join(base\_path, "data", "chembl\_35\_data\_100.csv")  
df = pd.read\_csv(csv\_file\_path)  
  
# Convert standard\_value to numeric and handle errors  
df['standard\_value'] = pd.to\_numeric(df['standard\_value'], errors='coerce')  
df.dropna(subset=['standard\_value'], inplace=True)  
  
# Convert IC50, Ki, Kd to pIC50  
def convert\_to\_pIC50(value):  
 return -np.log10(value \* 1e-9) # Convert nM to M and then to pIC50  
  
df['pIC50'] = df['standard\_value'].apply(convert\_to\_pIC50)  
  
# Boxplot of pIC50 for different target names (pref\_name)  
plt.figure(figsize=(12, 6)) # Adjust figure size for readability  
  
# Group by 'pref\_name' and calculate median pIC50, then sort  
median\_pIC50 = df.groupby('pref\_name')['pIC50'].median().sort\_values(ascending=False)  
  
# Use the sorted index to maintain order in the boxplot  
sns.boxplot(x='pref\_name', y='pIC50', data=df, order=median\_pIC50.index)  
plt.title('pIC50 by Target Name (Pref\_Name)')  
plt.xlabel('Target Name (Pref\_Name)')  
plt.ylabel('pIC50')  
plt.xticks(rotation=90) # Rotate x-axis labels for readability  
plt.tight\_layout() # Adjust layout to prevent labels from overlapping  
plt.show()

This example creates a boxplot comparing the distribution of pIC50 values for different target names (pref\_name). This can help you identify targets with high or low activity. The code is modified to properly sort the boxplots by the median pIC50 to make it more readable.

**Important Considerations:**

* **Data Scaling:** If you’re building machine learning models, consider scaling your molecular features (e.g., using MinMaxScaler or StandardScaler from scikit-learn). This can improve model performance.
* **Feature Selection:** Experiment with different sets of molecular features. Some features may be more predictive than others.
* **Model Validation:** Use proper cross-validation techniques to evaluate the performance of your machine learning models.
* **Domain Knowledge:** Always use your domain knowledge (understanding of chemistry, biology, and pharmacology) to guide your analysis and interpret your results.

**In summary (En résumé):**

You’re on the right track! Follow the AIMLops structure, use the corrected SQL and Python code, upgrade or adapt to your scikit-learn version, and explore the data with the provided examples. Remember to adapt the code to your specific research questions and ChEMBL 35 table names if they differ. Good luck!

**1. Modèle d’analyse et stratégie globale**

Le cœur de votre projet consiste probablement à explorer les relations au sein des données ChEMBL. Une approche courante consiste à :

* **Extraction et nettoyage des données :** Utilisez des requêtes SQL pour extraire les données d’activité pertinentes (IC50, Ki, etc.) et les informations sur les composés (chaînes SMILES) de ChEMBL. Cette étape permettra également de filtrer les données pour garantir des unités et des types de données cohérents. La résolution de l’erreur "numeric ~ unknown" relèvera également de cette catégorie.
* **Génération de caractéristiques moléculaires :** Utilisez RDKit pour convertir les chaînes SMILES en représentations numériques des propriétés moléculaires (par exemple, le poids moléculaire, le logP, le nombre de donneurs/accepteurs de liaisons hydrogène, la surface polaire topologique - TPSA, les empreintes digitales comme les empreintes digitales Morgan).
* **Exploration et visualisation des données :** Visualisez la distribution des valeurs d’activité, des propriétés moléculaires et des relations entre elles à l’aide de bibliothèques telles que Matplotlib ou Seaborn.
* **Construction de modèles (facultatif) :** Si vous avez une cible spécifique en tête, vous pouvez construire des modèles prédictifs reliant les caractéristiques moléculaires aux valeurs d’activité (par exemple, des modèles quantitatifs de relation structure-activité - QSAR) en utilisant des algorithmes d’apprentissage automatique de scikit-learn.
* **Interprétation :** Traduisez votre analyse en informations significatives pour la découverte de médicaments, comme l’identification des principales caractéristiques structurelles associées à l’activité.

**2. Structure AIMLops et emplacement du code**

En suivant le modèle AIMLops, voici comment votre projet doit être organisé :

* **data/ :** Contient les données brutes (fichiers CSV générés à partir de requêtes SQL) et potentiellement les données traitées (par exemple, les données de caractéristiques).
* **notebooks/ :** Contient les notebooks Jupyter avec votre code. Les noms de fichiers doivent ressembler à Topic\_CheMBL\_35\_80\_1\_Data\_Extraction.ipynb, Topic\_CheMBL\_35\_80\_2\_Feature\_Engineering.ipynb, etc.
* **src/ (Facultatif, mais recommandé pour les projets plus complexes) :** Contient des modules Python réutilisables pour des fonctions telles que le nettoyage des données, la génération de caractéristiques ou l’entraînement des modèles. Cela favorise la modularité et la réutilisabilité.
* **models/ (Facultatif) :** Stocke les modèles d’apprentissage automatique entraînés si vous en construisez.

**3. Code (SQL et Python) avec gestion des erreurs**

Le code ci-dessus corrige vos problèmes.

**4. Résolution des erreurs**

* **ERROR: operator does not exist: numeric ~ unknown :** Cela est dû à une incompatibilité de type dans votre requête SQL. L’opérateur ~ (correspondance d’expression régulière) attend une chaîne de caractères du côté droit, mais act.standard\_value est probablement un type numérique. La requête SQL corrigée ci-dessus convertit la valeur numérique en texte avant d’appliquer l’expression régulière.
* **old scikit-learn version does not support parameters squared=False in the mean\_squared\_error function :** Cela indique que vous utilisez une ancienne version de scikit-learn. Il existe deux solutions :
  + **Mettre à niveau scikit-learn :** La solution la plus simple est de mettre à niveau votre version de scikit-learn vers une version plus récente (>=0.20). Vous pouvez le faire en utilisant pip : pip install --upgrade scikit-learn
  + **Supprimer squared=False :** Si vous ne pouvez pas mettre à niveau scikit-learn, supprimez simplement le paramètre squared=False de l’appel de la fonction mean\_squared\_error. La fonction renverra alors l’erreur quadratique moyenne (MSE) au lieu de la racine de l’erreur quadratique moyenne (RMSE).

**5. Exemples (Analyse)**

Les cinq exemples d’analyse ci-dessus montrent différents aspects de l’analyse des données de découverte de médicaments avec les données ChEMBL.

Suivez la structure AIMLops, utilisez le code SQL et Python corrigé, mettez à niveau ou adaptez-vous à votre version de scikit-learn et explorez les données avec les exemples fournis. N’oubliez pas d’adapter le code à vos questions de recherche spécifiques et aux noms des tables ChEMBL 35 s’ils diffèrent. Bonne chance !