# PharmApp Suite

## 🧠 AI for Drug Discovery and Development 🧪

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

## 🧩 Topic: Comparative barplots for molecule classes

Okay, I understand. I will act as your Pharmaceutical Research and Development Specialist and guide you through this Chembl 35 data analysis project. We’ll focus on getting a working pipeline with small, manageable data subsets.

**1. Analysis Model Explanation**

The goal of your project is likely to extract relevant data from the ChEMBL 35 database, process it using RDKit, and potentially perform some machine learning analysis. Here’s a breakdown of the assumed analysis model, along with solutions to your identified errors:

* **Data Extraction (SQL):**
  + We need to extract relevant data like compound structures (SMILES), activity data (IC50, Ki, etc.), and target information.
  + The SQL query should be efficient and filter the data to a manageable size (100 rows as requested).
  + **Addressing Error (a): ERROR: operator does not exist: numeric ~ unknown, LINE 12: AND act.standard\_value ~ '^[0-9\.]+$':** This error means you’re trying to use a regular expression operator (~) on a numeric column (act.standard\_value). The column likely contains values that are not purely numeric. We need to either:
    - Convert the column to text first before using ~: AND CAST(act.standard\_value AS TEXT) ~ '^[0-9\.]+$'
    - Or, more robustly, filter using a numeric comparison: AND act.standard\_value IS NOT NULL AND act.standard\_value >= 0 (assuming activity values can’t be negative and null values shouldn’t be included)
* **Data Processing (Python/RDKit):**
  + Load the extracted data into a pandas DataFrame.
  + Use RDKit to generate molecular fingerprints or other descriptors from the SMILES strings.
  + Handle potential errors in SMILES parsing.
* **Data Analysis/Modeling (Python/Scikit-learn):**
  + This part is more open-ended. You could explore relationships between molecular descriptors and activity values. You might want to:
    - Build regression models to predict activity.
    - Perform clustering to identify similar compounds.
    - Visualize the data to identify trends.
  + **Addressing Error (b): old scikit-learn version does not support parameters squared=False in the mean\_squared\_error function:** This is straightforward. Either:
    - Upgrade your scikit-learn version: pip install -U scikit-learn
    - Or, if upgrading is not possible (due to dependency constraints), remove the squared=False argument. The default behavior of mean\_squared\_error is to return the Mean Squared Error (MSE), which is the squared error. If you want the Root Mean Squared Error (RMSE), you’ll need to take the square root manually: rmse = np.sqrt(mean\_squared\_error(y\_true, y\_pred))

**2. SQL and Python Code (Topic\_CheMBL\_35\_75)**

Here’s example code to get you started. I’ll incorporate error handling and address the points above.

**SQL (save as Topic\_CheMBL\_35\_75.sql in your ../data/ directory):**

-- Get 100 rows, example table  
SELECT DISTINCT  
 md.molregno,  
 cs.canonical\_smiles,  
 act.standard\_value,  
 act.standard\_units,  
 act.standard\_type,  
 act.assay\_id  
FROM  
 molecule\_dictionary md  
JOIN  
 compound\_structures cs ON md.molregno = cs.molregno  
JOIN  
 activities act ON md.molregno = act.molregno  
WHERE act.standard\_type = 'IC50'  
AND act.standard\_units = 'nM'  
AND act.standard\_value IS NOT NULL -- Handle null values  
AND act.standard\_value >= 0 -- Filter only positive values  
LIMIT 100;

**Explanation:**

* This query retrieves 100 rows of IC50 values in nM units.
* It joins the molecule\_dictionary, compound\_structures, and activities tables based on molregno.
* AND act.standard\_value IS NOT NULL AND act.standard\_value >= 0 ensures we are only taking numeric values

**Python (Jupyter Notebook Topic\_CheMBL\_35\_75\_1\_Data\_Extraction\_and\_Processing.ipynb):**

import os  
import pandas as pd  
import psycopg2  
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  
import matplotlib.pyplot as plt  
  
# Define paths  
base\_path = os.getcwd() # Current working directory (where the notebook is)  
data\_path = os.path.join(base\_path, '../data')  
sql\_file = os.path.join(data\_path, 'Topic\_CheMBL\_35\_75.sql')  
csv\_file = os.path.join(data\_path, 'Topic\_CheMBL\_35\_75.csv')  
  
# Database credentials  
db\_host = '192.168.206.136'  
db\_user = 'rd'  
db\_pass = 'rd'  
db\_name = 'chembl\_35'  
  
# Function to execute SQL and save to CSV  
def execute\_sql\_and\_save\_csv(sql\_file, csv\_file, db\_host, db\_user, db\_pass, db\_name):  
 try:  
 conn = psycopg2.connect(host=db\_host, user=db\_user, password=db\_pass, database=db\_name)  
 cur = conn.cursor()  
  
 # Read SQL from file  
 with open(sql\_file, 'r') as f:  
 sql\_query = f.read()  
  
 # Execute SQL query  
 cur.execute(sql\_query)  
  
 # Fetch data and column names  
 data = cur.fetchall()  
 column\_names = [desc[0] for desc in cur.description]  
  
 # Create Pandas DataFrame  
 df = pd.DataFrame(data, columns=column\_names)  
  
 # Save to CSV  
 df.to\_csv(csv\_file, index=False)  
  
 print(f"Data successfully extracted and saved to {csv\_file}")  
  
 except psycopg2.Error as e:  
 print(f"Error connecting to database or executing query: {e}")  
 finally:  
 if conn:  
 cur.close()  
 conn.close()  
# Run data extraction  
execute\_sql\_and\_save\_csv(sql\_file, csv\_file, db\_host, db\_user, db\_pass, db\_name)  
  
# Function to generate Morgan fingerprints  
def generate\_morgan\_fingerprints(smiles, radius=2, n\_bits=2048):  
 try:  
 mol = Chem.MolFromSmiles(smiles)  
 if mol is None:  
 return None # Handle invalid SMILES  
 fp = AllChem.GetMorganFingerprintAsBitVect(mol, radius, nBits=n\_bits)  
 return np.array(fp)  
 except Exception as e:  
 print(f"Error generating fingerprint for {smiles}: {e}")  
 return None  
  
# Load data from CSV  
try:  
 df = pd.read\_csv(csv\_file)  
except FileNotFoundError:  
 print(f"Error: CSV file not found at {csv\_file}")  
 exit()  
  
# Generate Morgan fingerprints and add them to the DataFrame  
df['fingerprint'] = df['canonical\_smiles'].apply(generate\_morgan\_fingerprints)  
  
# Drop rows where fingerprint generation failed  
df = df.dropna(subset=['fingerprint'])  
  
# Convert standard\_value to numeric (handle potential errors)  
df['standard\_value'] = pd.to\_numeric(df['standard\_value'], errors='coerce')  
df = df.dropna(subset=['standard\_value']) # Remove rows with invalid standard\_value  
  
print(df.head())  
print(df.shape)

**Explanation:**

1. **Imports:** Imports necessary libraries (pandas, psycopg2, RDKit, scikit-learn).
2. **Paths:** Defines file paths using os.path.join for cross-platform compatibility.
3. **execute\_sql\_and\_save\_csv Function:**
   * Connects to the PostgreSQL database using psycopg2.
   * Reads the SQL query from the specified file.
   * Executes the SQL query and fetches the results.
   * Creates a pandas DataFrame from the results and saves it to a CSV file.
   * Includes comprehensive error handling for database connection and query execution.
4. **generate\_morgan\_fingerprints Function:**
   * Takes a SMILES string as input.
   * Uses RDKit to convert the SMILES string to a molecule object.
   * Generates a Morgan fingerprint (ECFP4) as a bit vector.
   * Returns the fingerprint as a NumPy array.
   * Includes error handling to catch invalid SMILES strings.
5. **Data Loading:** Loads the CSV file into a pandas DataFrame.
6. **Fingerprint Generation:** Applies the generate\_morgan\_fingerprints function to the canonical\_smiles column to generate fingerprints. Stores the fingerprints in a new ‘fingerprint’ column.
7. **Data Cleaning:** Removes rows where fingerprint generation failed (indicated by NaN in the ‘fingerprint’ column).
8. **Display Data:** Prints the first few rows of the DataFrame and its shape.

**Python (Jupyter Notebook Topic\_CheMBL\_35\_75\_2\_Model\_Building.ipynb):**

import os  
import pandas as pd  
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  
import matplotlib.pyplot as plt  
  
# Define paths  
base\_path = os.getcwd()  
data\_path = os.path.join(base\_path, '../data')  
csv\_file = os.path.join(data\_path, 'Topic\_CheMBL\_35\_75.csv') # Use the same CSV from the previous notebook  
  
# Load data from CSV  
try:  
 df = pd.read\_csv(csv\_file)  
except FileNotFoundError:  
 print(f"Error: CSV file not found at {csv\_file}")  
 exit()  
  
# Convert standard\_value to pIC50  
df['pIC50'] = -np.log10(df['standard\_value'] \* 1e-9) # nM to Molar  
  
# Prepare data for modeling  
X = np.stack(df['fingerprint'].values) # Stack fingerprints into a feature matrix  
y = df['pIC50'].values # Target variable (pIC50)  
  
# 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 on the test set  
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}")  
  
# Optional: Plot predictions vs. actual values  
plt.scatter(y\_test, y\_pred)  
plt.xlabel("Actual pIC50")  
plt.ylabel("Predicted pIC50")  
plt.title("Actual vs. Predicted pIC50")  
plt.show()

**Explanation:**

1. **Imports:** Imports necessary libraries.
2. **Paths:** Defines the path to the CSV file (important to ensure it matches the previous notebook).
3. **Data Loading:** Loads the CSV file into a pandas DataFrame.
4. **pIC50 Conversion:** Converts the IC50 values to pIC50, a more common scale for drug discovery.
5. **Data Preparation:**
   * Creates the feature matrix X by stacking the fingerprint arrays. Critically, this assumes your fingerprint column contains *arrays*, not strings representing arrays.
   * Creates the target variable y from the ‘pIC50’ column.
6. **Train/Test Split:** Splits the data into training and testing sets.
7. **Model Training:** Trains a linear regression model.
8. **Prediction:** Makes predictions on the test set.
9. **Evaluation:** Calculates and prints the Mean Squared Error and R-squared.
10. **Visualization (Optional):** Creates a scatter plot of predicted vs. actual pIC50 values.

**3. Five Examples of Further Analysis:**

Here are five examples of what you could do to further analyze your data:

1. **Different Molecular Descriptors:** Instead of Morgan fingerprints, explore other descriptors like physicochemical properties (e.g., LogP, molecular weight, number of hydrogen bond donors/acceptors) using RDKit. See the RDKit documentation for Descriptors and Chem.Lipinski.
2. **Different Machine Learning Models:** Try other regression models, such as Random Forest Regressor, Support Vector Regression, or Gradient Boosting Regressor. Compare their performance. You will need to import these from sklearn.ensemble or sklearn.svm accordingly.
3. **Substructure Search:** Use RDKit to search for specific substructures within your molecules. See if the presence of these substructures correlates with activity. Use Chem.MolFromSmarts and mol.HasSubstructMatch.
4. **Clustering:** Perform clustering on the molecular descriptors to identify groups of similar compounds. Use algorithms like K-Means or hierarchical clustering from sklearn.cluster.
5. **Data Visualization:** Create more sophisticated visualizations to explore relationships between variables. Use libraries like Seaborn or Plotly to create interactive plots. Explore scatter plots, histograms, and box plots.

**Important Considerations:**

* **Error Handling:** The code includes basic error handling, but you should add more robust error handling to catch potential issues during data processing and modeling.
* **Data Scaling/Normalization:** For many machine learning algorithms, scaling or normalizing your data can improve performance. Consider using StandardScaler or MinMaxScaler from sklearn.preprocessing.
* **Hyperparameter Tuning:** Optimize the hyperparameters of your machine learning models using techniques like cross-validation and grid search. Use GridSearchCV or RandomizedSearchCV from sklearn.model\_selection.
* **Validation:** Always validate your models on an independent test set to ensure they generalize well to new data.

**Before Running:**

1. **Install Dependencies:** Make sure you have all the necessary libraries installed: bash pip install pandas psycopg2 rdkit scikit-learn matplotlib
2. **PostgreSQL Connection:** Verify that you can connect to your PostgreSQL database using the provided credentials.
3. **Directory Structure:** Confirm that your directory structure matches the example: your\_project\_directory/ notebooks/ Topic\_CheMBL\_35\_75\_1\_Data\_Extraction\_and\_Processing.ipynb Topic\_CheMBL\_35\_75\_2\_Model\_Building.ipynb data/ Topic\_CheMBL\_35\_75.sql

This should give you a solid foundation for your Chembl 35 analysis. Let me know if you have any more questions. Good luck!