# PharmApp Suite

## 🧠 AI for Drug Discovery and Development 🧪

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

## 🧩 Topic: Deep learning models for activity prediction

Okay, I understand. You’re working with ChEMBL 35 data, using PostgreSQL, RDKit, and Jupyter Notebooks in an AIMLops-structured project. You want to analyze the data, create some models, and address potential errors, limiting the data to 100 rows for efficiency. Let’s break this down.

**1. Analysis Model**

The analysis will focus on exploring the ChEMBL data related to bioactivity and molecular properties. We’ll perform the following steps:

* **Data Retrieval:** Query the ChEMBL database to extract bioactivity data (e.g., IC50 values) and associated compound information (e.g., SMILES strings). We’ll focus on a specific target or a set of targets to narrow down the scope.
* **Data Cleaning and Preprocessing:** Clean the data, handling missing values, converting units, and standardizing activity values. We’ll address the numeric ~ unknown error in the SQL query.
* **Molecular Representation:** Use RDKit to generate molecular fingerprints or descriptors from the SMILES strings. These fingerprints represent the molecules numerically and are used as input for machine learning models.
* **Model Building:** Build a machine learning model to predict bioactivity based on the molecular fingerprints. This could be a regression model (e.g., Linear Regression, Random Forest Regression) or a classification model (e.g., Logistic Regression, Random Forest Classifier) depending on how the bioactivity is represented (continuous or categorical).
* **Model Evaluation:** Evaluate the performance of the model using appropriate metrics (e.g., Mean Squared Error (MSE) for regression, Accuracy, Precision, Recall, F1-score for classification).
* **Visualization and Interpretation:** Visualize the data and the model results to gain insights into the relationship between molecular properties and bioactivity.

**2. Code (SQL & Python)**

**SQL (Topic\_CheMBL\_35\_62.sql):**

-- data/Topic\_CheMBL\_35\_62.csv  
  
SELECT  
 act.activity\_id,  
 act.standard\_inchi\_key,  
 act.standard\_relation,  
 act.standard\_value,  
 act.standard\_units,  
 act.assay\_id,  
 act.target\_id,  
 cp.canonical\_smiles  
FROM  
 activities act  
JOIN  
 compound\_structures cp ON act.molregno = cp.molregno  
WHERE  
 act.standard\_type = 'IC50' -- Filter for IC50 values  
 AND act.standard\_relation = '=' -- Filter for direct measurements  
 AND act.standard\_value IS NOT NULL  
 AND cp.canonical\_smiles IS NOT NULL  
 AND act.standard\_value::text ~ '^[0-9\.]+$' -- Corrected line: Cast to text and use regex  
LIMIT 100; -- Limit to 100 rows  
  
-- Explanation:  
-- 1. `act.standard\_value::text ~ '^[0-9\.]+$'`: The original error occurs because you're trying to use a regex operator (`~`) directly on a `numeric` type. The corrected code casts `act.standard\_value` to `text` before applying the regex. This ensures that the regular expression matching can be performed. The regex `^[0-9\.]+$` checks if the value consists only of digits and periods.  
-- 2. `LIMIT 100`: Limits the result set to the first 100 rows.  
-- 3. Add filter Standard\_inchi\_key, cp.canonical\_smiles is not null

**Steps to run the SQL and save the results:**

1. **Connect to your PostgreSQL database:** Open pgAdmin and connect to the database using the IP address, username, password, and database name you provided.
2. **Create the SQL file:** Copy the SQL code above and save it as Topic\_CheMBL\_35\_62.sql in your project directory (likely under a data subdirectory).
3. **Execute the SQL query:** Open the SQL file in pgAdmin and execute it.
4. **Export the results to CSV:** After the query runs successfully, export the results as a CSV file named Topic\_CheMBL\_35\_62.csv and save it in the data directory of your project. Make sure the header row is included.

**Python (Topic\_CheMBL\_35\_62\_1\_Data\_Preprocessing.ipynb):**

# Topic\_CheMBL\_35\_62\_1\_Data\_Preprocessing.ipynb  
  
import os  
import pandas as pd  
from rdkit import Chem  
from rdkit.Chem import AllChem  
import numpy as np  
  
# Define the base path for your project  
base\_path = "." # Assuming the notebook is run from the project root  
  
# Path to the CSV file  
csv\_file\_path = os.path.join(base\_path, "data", "Topic\_CheMBL\_35\_62.csv")  
  
# Load the data  
try:  
 df = pd.read\_csv(csv\_file\_path)  
 print("Data loaded successfully.")  
except FileNotFoundError:  
 print(f"Error: File not found at {csv\_file\_path}")  
 exit()  
  
# Data Cleaning and Preprocessing  
print("\nData Cleaning and Preprocessing...")  
  
# Handle missing values (replace with NaN and then drop rows with NaNs)  
df.replace([np.inf, -np.inf], np.nan, inplace=True)  
df.dropna(subset=['canonical\_smiles', 'standard\_value'], inplace=True) # Drop rows with missing SMILES or IC50 values  
  
# Convert IC50 values to pIC50 (negative log)  
def ic50\_to\_pic50(ic50\_nM):  
 """Converts IC50 (nM) to pIC50."""  
 pIC50 = -np.log10(ic50\_nM / 1e9) # Convert nM to Molar  
 return pIC50  
  
# Only process rows where standard\_units == "nM"  
df\_nM = df[df['standard\_units'] == 'nM'].copy()  
  
if not df\_nM.empty:  
 df\_nM['pIC50'] = df\_nM['standard\_value'].apply(ic50\_to\_pic50)  
 print("pIC50 values calculated for nM data.")  
  
 # Add RDKit Mol objects  
 df\_nM['ROMol'] = df\_nM['canonical\_smiles'].apply(lambda x: Chem.MolFromSmiles(x))  
 df\_nM = df\_nM[df\_nM['ROMol'].notna()] # Remove rows where Mol object creation failed  
 print("RDKit Mol objects created.")  
  
 # Display the first few rows of the processed DataFrame  
 print("\nFirst 5 rows of processed data:")  
 print(df\_nM.head())  
else:  
 print("No data with standard\_units 'nM' found.")  
  
# Save the processed DataFrame (optional)  
processed\_csv\_file\_path = os.path.join(base\_path, "data", "Topic\_CheMBL\_35\_62\_processed.csv")  
if not df\_nM.empty:  
 df\_nM.to\_csv(processed\_csv\_file\_path, index=False)  
 print(f"\nProcessed data saved to {processed\_csv\_file\_path}")  
else:  
 print("\nNo processed data to save.")

**Python (Topic\_CheMBL\_35\_62\_2\_Model\_Building.ipynb):**

# Topic\_CheMBL\_35\_62\_2\_Model\_Building.ipynb  
  
import os  
import pandas as pd  
from rdkit import Chem  
from rdkit.Chem import AllChem  
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  
import numpy as np  
  
# Define the base path for your project  
base\_path = "." # Assuming the notebook is run from the project root  
  
# Path to the processed CSV file  
processed\_csv\_file\_path = os.path.join(base\_path, "data", "Topic\_CheMBL\_35\_62\_processed.csv")  
  
# Load the processed data  
try:  
 df = pd.read\_csv(processed\_csv\_file\_path)  
 print("Processed data loaded successfully.")  
except FileNotFoundError:  
 print(f"Error: File not found at {processed\_csv\_file\_path}")  
 exit()  
  
# Molecular Fingerprint Generation  
print("\nGenerating Molecular Fingerprints...")  
  
def generate\_fingerprint(mol):  
 """Generates Morgan fingerprints (ECFP4)."""  
 try:  
 return AllChem.GetMorganFingerprintAsBitVect(mol, 2, nBits=2048) #Returns SparseBitVect  
 except Exception as e:  
 print(f"Error generating fingerprint: {e}")  
 return None  
  
df['fingerprint'] = df['ROMol'].apply(generate\_fingerprint)  
df.dropna(subset=['fingerprint'], inplace=True) # Drop rows where fingerprint generation failed  
print("Molecular fingerprints generated.")  
  
# Data Preparation for Modeling  
print("\nPreparing data for modeling...")  
  
# Convert fingerprints to numpy arrays  
X = np.array([np.array(x) for x in df['fingerprint']]) #Convert SparseBitVect to numpy array  
y = df['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)  
  
# Standardize the data (important for linear models)  
scaler = StandardScaler()  
X\_train = scaler.fit\_transform(X\_train)  
X\_test = scaler.transform(X\_test)  
  
# Model Building  
print("\nBuilding and training the Linear Regression model...")  
  
model = LinearRegression()  
model.fit(X\_train, y\_train)  
  
# Model Evaluation  
print("\nEvaluating the model...")  
  
y\_pred = model.predict(X\_test)  
  
mse = mean\_squared\_error(y\_test, y\_pred)  
r2 = r2\_score(y\_test, y\_pred)  
  
print(f"Mean Squared Error: {mse:.2f}")  
print(f"R-squared: {r2:.2f}")  
  
# Interpretation (Example)  
print("\nModel Interpretation (Example):")  
print("Coefficients for the first 5 features:", model.coef\_[:5])  
  
# Visualization (Example - requires matplotlib)  
# import matplotlib.pyplot as plt  
# plt.scatter(y\_test, y\_pred)  
# plt.xlabel("Actual pIC50")  
# plt.ylabel("Predicted pIC50")  
# plt.title("Actual vs. Predicted pIC50")  
# plt.show()

**Explanation of Python Code:**

* **Data Loading:** Loads the CSV file generated from the SQL query using pandas. Handles potential FileNotFoundError.
* **Data Cleaning:** Handles missing values by dropping rows with missing SMILES or pIC50 values.
* **pIC50 Conversion:** Converts IC50 values (in nM) to pIC50 values. This is a common transformation in drug discovery, as pIC50 values are more normally distributed and easier to work with.
* **RDKit Mol Object Creation:** Creates RDKit Mol objects from the SMILES strings. These Mol objects are required for generating molecular fingerprints.
* **Molecular Fingerprint Generation:** Uses RDKit to generate Morgan fingerprints (ECFP4). These fingerprints are numerical representations of the molecules and are used as input features for the machine learning model.
* **Data Splitting:** Splits the data into training and testing sets.
* **Model Building:** Creates and trains a Linear Regression model.
* **Model Evaluation:** Evaluates the model using Mean Squared Error (MSE) and R-squared.
* **Model Interpretation:** Prints the coefficients of the first 5 features of the linear regression model. This provides some insight into which features are most important for predicting pIC50.
* **Visualization (Commented Out):** Includes example code for visualizing the results using a scatter plot of actual vs. predicted pIC50 values. This requires matplotlib.
* **Error Handling:** The code includes try...except blocks to handle potential errors, such as FileNotFoundError when loading the CSV file or errors during fingerprint generation. If an error occurs, the code prints an informative message and continues to the next row.

**Addressing the Errors:**

* **SQL Error (numeric ~ unknown):** The fix is in the SQL code above. Cast the standard\_value to text before using the regex operator: act.standard\_value::text ~ '^[0-9\.]+$'
* **Scikit-learn Error (squared=False):** The squared=False parameter in mean\_squared\_error was introduced in a later version of scikit-learn. If you’re using an older version, remove the squared=False parameter. Alternatively, upgrade your scikit-learn version to the latest one. Since MSE is a popular metric, it is not useful.

**Important Considerations:**

* **Dependencies:** Make sure you have the necessary Python packages installed: pandas, rdkit, scikit-learn, numpy. You can install them using pip: bash pip install pandas rdkit scikit-learn numpy
* **RDKit Installation:** RDKit can be a bit tricky to install. It’s recommended to use conda if possible: bash conda install -c conda-forge rdkit
* **File Paths:** Double-check the file paths in the Python code to make sure they are correct for your project structure.
* **Data Exploration:** Before building the model, it’s crucial to explore the data. Look at the distribution of pIC50 values, the range of fingerprint values, and any potential biases in the data. This will help you choose the right model and interpret the results.
* **Model Selection:** Linear Regression is a simple model, but it may not be the best choice for all datasets. Consider trying other models, such as Random Forest Regression or Support Vector Regression.
* **Hyperparameter Tuning:** Tune the hyperparameters of the machine learning model to improve its performance. You can use techniques like cross-validation and grid search to find the optimal hyperparameters.
* **Data Filtering:** Filter by inchi\_key, canonical\_smiles to reduce error handling.

**3. Five Examples (Scenario-Based)**

Here are five examples of how you could use this analysis in a pharmaceutical R&D context:

1. **Target Identification:** Analyze the bioactivity data for a specific target protein (e.g., a kinase involved in cancer). The model can help identify compounds that are likely to be active against that target.
2. **Lead Optimization:** You have a lead compound and want to improve its potency. Use the model to predict the pIC50 of various analogs of the lead compound, based on their molecular fingerprints. This can help you prioritize which analogs to synthesize and test.
3. **Off-Target Prediction:** You want to assess the potential for off-target effects of a drug candidate. Build models for multiple targets and use them to predict the activity of the drug candidate against each target. This can help identify potential safety concerns.
4. **Virtual Screening:** You have a large database of compounds and want to identify potential hits for a target of interest. Use the model to predict the pIC50 of each compound in the database. This can help you prioritize which compounds to screen experimentally.
5. **Structure-Activity Relationship (SAR) Analysis:** Use the model to understand the relationship between the structure of a molecule and its activity. Analyze the coefficients of the linear regression model or the feature importance of a random forest model to identify which molecular features are most important for activity.

**In Summary (En résumé):**

You’re taking on a well-defined project. The SQL and Python code provides a solid foundation. Remember to adjust file paths, install dependencies, address errors, and explore your data thoroughly. By following these steps, you can build a predictive model and gain valuable insights from the ChEMBL 35 data.

**Analyse (Analyse):**

Vous vous lancez dans un projet bien défini. Le code SQL et Python fournit une base solide. N’oubliez pas d’ajuster les chemins d’accès aux fichiers, d’installer les dépendances, de corriger les erreurs et d’explorer vos données en profondeur. En suivant ces étapes, vous pouvez créer un modèle prédictif et obtenir des informations précieuses à partir des données ChEMBL 35.