# PharmApp Suite

## 🧠 AI for Drug Discovery and Development 🧪

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

## 🧩 Topic: Antiviral compounds mapping to COVID-19 targets

Okay, I understand. You’re working with ChEMBL 35 data using RDKit, following an AIMLops template structure. You need guidance on structuring your project, running SQL queries, and implementing Python code for analysis, addressing specific errors encountered. You want to focus on retrieving a small subset of data (100 rows) for faster prototyping and provide 5 examples.

Here’s a breakdown of the approach, including SQL and Python code snippets, and addressing the errors you’ve encountered.

**1. Project Structure & AIMLops Template Adherence**

Following the AIMLops pattern, a typical project structure looks like this:

Topic\_CheMBL\_35\_53/ # Root directory  
├── data/ # Raw data, CSV files extracted from ChEMBL  
├── notebooks/ # Jupyter notebooks for exploration and analysis  
│ ├── Topic\_CheMBL\_35\_53\_1\_EDA.ipynb # Exploratory Data Analysis  
│ ├── Topic\_CheMBL\_35\_53\_2\_Model.ipynb # Model building, e.g., regression  
│ └── ...  
├── src/ # Python modules for reusable functions  
│ ├── data\_processing.py  
│ ├── model\_training.py  
│ └── utils.py  
├── models/ # Serialized models (e.g., pickle files)  
├── reports/ # Generated reports (e.g., HTML, PDFs)  
├── config/ # Configuration files (e.g., database connection details)  
├── README.md # Project documentation  
└── requirements.txt # Python dependencies

This organized structure makes your project maintainable, reproducible, and scalable.

**2. Analysis Model & Strategy**

Since you’re not specifying a specific task beyond working with ChEMBL data, let’s outline a simple, common drug discovery workflow that you can adapt:

* **Data Extraction & Preparation:** Fetch data from ChEMBL using SQL, focusing on bioactivity data (e.g., IC50 values) and compound structures (SMILES). Convert SMILES strings to RDKit molecules.
* **Exploratory Data Analysis (EDA):** Calculate molecular properties using RDKit (e.g., molecular weight, LogP, number of hydrogen bond donors/acceptors). Visualize the distributions of these properties.
* **Feature Engineering:** Generate more complex molecular descriptors or fingerprints using RDKit (e.g., Morgan fingerprints).
* **Modeling (Regression):** Build a regression model to predict bioactivity (e.g., IC50) based on the calculated molecular descriptors. Common choices are linear regression, Random Forest, or Support Vector Regression.
* **Model Evaluation:** Assess the model’s performance using metrics like Mean Squared Error (MSE), R-squared, and other relevant measures.
* **Iterate:** refine the analysis steps above

**3. SQL Code (Extracting Data)**

-- File: data/chembl\_data.sql  
  
SELECT  
 md.chembl\_id,  
 cs.canonical\_smiles,  
 act.standard\_type,  
 act.standard\_value,  
 act.standard\_units,  
 act.relation  
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' -- Focus on IC50 values  
 AND act.standard\_units = 'nM' -- Focus on nM units  
 AND act.relation = '=' --Only select the exactly value (=)  
 AND act.standard\_value ~ '^[0-9\.]+$' -- ensure `standard\_value` contains only numeric value  
LIMIT 100; -- Limit to 100 rows  
-- Ensure you have created an output to save the result as a CSV file.  
-- For example, in pgAdmin, right-click on the query result and select "Copy with Headers" or "Save as CSV".

**Explanation:**

* **molecule\_dictionary, compound\_structures, activities**: These are core ChEMBL tables.
* **JOIN**: Connects the tables based on molecule identifiers (molregno).
* **WHERE**: Filters the data to IC50 values in nM, and retrieves the standard value.
* **LIMIT 100**: Restricts the output to the first 100 rows for faster testing.
* **act.standard\_value ~ '^[0-9\.]+$'**: This regular expression filter makes sure that the standard\_value column contains only numbers and periods, preventing errors during numeric conversion in Python. **This addresses error (a) in your question.**

**How to Run (using psql or pgAdmin):**

1. **psql:** bash psql -h 192.168.206.136 -U rd -d chembl\_35 -f data/chembl\_data.sql -o data/chembl\_data.csv -F ',' -A -t
   * -h: Host IP address
   * -U: User
   * -d: Database name
   * -f: SQL file
   * -o: Output file
   * -F ',': Set the field separator to a comma for CSV format.
   * -A: Turn off unaligned output.
   * -t: Turn off table header and row count output.
2. **pgAdmin:**
   * Connect to your PostgreSQL server.
   * Open a query window for the chembl\_35 database.
   * Paste the SQL code into the query window.
   * Execute the query.
   * Right-click on the query result grid.
   * Select “Copy with Headers” or “Save as CSV” to save the result to data/chembl\_data.csv.

**4. Python Code (Notebook Example - Topic\_CheMBL\_35\_53\_1\_EDA.ipynb)**

# Topic\_CheMBL\_35\_53\_1\_EDA.ipynb  
  
import os  
import pandas as pd  
from rdkit import Chem  
from rdkit.Chem import Descriptors  
import matplotlib.pyplot as plt  
import numpy as np  
import seaborn as sns  
  
  
# Define the base path based on your project structure  
base\_path = os.getcwd() # Assuming notebook is in the "notebooks" directory  
  
  
# Load the data  
data\_path = os.path.join(base\_path, 'data', 'chembl\_data.csv') # Fix: Correct path  
print(f"Loading data from: {data\_path}")  
try:  
 df = pd.read\_csv(data\_path)  
except FileNotFoundError:  
 print(f"Error: File not found at {data\_path}. Make sure you ran the SQL query and saved the CSV correctly.")  
 raise # Re-raise the exception to stop execution  
  
print(df.head())  
print(df.shape)  
  
  
# Data Cleaning and Conversion  
df = df.dropna(subset=['canonical\_smiles', 'standard\_value'])  
df['standard\_value'] = pd.to\_numeric(df['standard\_value'], errors='coerce') # Convert to numeric  
df = df.dropna(subset=['standard\_value']) # Remove any rows where conversion failed  
df = df[df['relation'] == '='] # Filter the records with relation  
df = df[df['standard\_units'] == 'nM'] # Filter the records with units  
print(df.shape)  
  
# RDKit Molecule Creation  
df['molecule'] = df['canonical\_smiles'].apply(lambda x: Chem.MolFromSmiles(x))  
df = df.dropna(subset=['molecule']) # Remove invalid SMILES  
print(df.shape)  
  
# Calculate Molecular Properties (Example)  
def calculate\_mw(mol):  
 try:  
 return Descriptors.MolWt(mol)  
 except:  
 return None  
  
df['molecular\_weight'] = df['molecule'].apply(calculate\_mw)  
df = df.dropna(subset=['molecular\_weight'])  
  
# Basic EDA (Example)  
plt.figure(figsize=(8, 6))  
sns.histplot(df['molecular\_weight'], kde=True)  
plt.title('Distribution of Molecular Weight')  
plt.xlabel('Molecular Weight')  
plt.ylabel('Frequency')  
plt.show()  
  
#Display data structure  
print(df.info())

**Explanation:**

1. **Path Handling:** Uses os.path.join to construct the correct file path. Includes error handling for file not found.
2. **Data Loading:** Reads the CSV file into a Pandas DataFrame.
3. **Data Cleaning:** Handles missing values (dropna), converts standard\_value to numeric, and removes rows with invalid SMILES.
4. **RDKit Integration:** Creates RDKit molecule objects from SMILES strings.
5. **Molecular Property Calculation:** Demonstrates calculating molecular weight using RDKit.
6. **EDA:** Creates a simple histogram of molecular weights.
7. **Error handling**: Added try and except block

**5. Python Code (Notebook Example - Topic\_CheMBL\_35\_53\_2\_Model.ipynb)**

# Topic\_CheMBL\_35\_53\_2\_Model.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  
import warnings  
warnings.filterwarnings('ignore')  
  
  
# Define the base path  
base\_path = os.getcwd()  
  
# Load the data  
data\_path = os.path.join(base\_path, 'data', 'chembl\_data.csv')  
df = pd.read\_csv(data\_path)  
  
# Data Cleaning and Conversion (same as in EDA notebook)  
df = df.dropna(subset=['canonical\_smiles', 'standard\_value'])  
df['standard\_value'] = pd.to\_numeric(df['standard\_value'], errors='coerce') # Convert to numeric  
df = df.dropna(subset=['standard\_value']) # Remove any rows where conversion failed  
df = df[df['relation'] == '='] # Filter the records with relation  
df = df[df['standard\_units'] == 'nM'] # Filter the records with units  
  
# RDKit Molecule Creation  
df['molecule'] = df['canonical\_smiles'].apply(lambda x: Chem.MolFromSmiles(x))  
df = df.dropna(subset=['molecule'])  
  
  
# Feature Engineering (Morgan Fingerprints)  
def calculate\_morgan\_fingerprint(mol, radius=2, nBits=2048):  
 try:  
 return AllChem.GetMorganFingerprintAsBitVect(mol, radius, nBits=nBits)  
 except:  
 return None  
  
df['fingerprint'] = df['molecule'].apply(calculate\_morgan\_fingerprint)  
df = df.dropna(subset=['fingerprint'])  
  
# Prepare Data for Modeling  
X = np.array([list(fp) for fp in df['fingerprint']]) # Convert fingerprints to numpy array  
y = np.log10(df['standard\_value']) # Log transform IC50 values (important for skewed data)  
  
  
# Data Scaling  
scaler = StandardScaler()  
X\_scaled = scaler.fit\_transform(X) # Scale the features  
  
# Split Data  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_scaled, y, test\_size=0.2, random\_state=42)  
  
# Model Training  
model = LinearRegression()  
model.fit(X\_train, y\_train)  
  
# Model Evaluation  
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}")  
print(f"R-squared: {r2}")  
  
# Save the model (optional)  
import joblib  
model\_path = os.path.join(base\_path, 'models', 'linear\_regression\_model.pkl')  
joblib.dump(model, model\_path)  
print(f"Model saved to {model\_path}")

**Explanation:**

1. **Feature Engineering:** Calculates Morgan fingerprints (ECFP4) using RDKit. This converts the molecule structure into numerical representation.
2. **Data Preparation:** Converts fingerprints to a NumPy array suitable for scikit-learn. Log-transforms the standard\_value (IC50) to handle skewed data, which is very common in bioactivity data.
3. **Data Scaling**: Apply feature scaling with StandardScaler
4. **Data Splitting:** Splits the data into training and testing sets.
5. **Model Training:** Trains a Linear Regression model. This is a simple example; you can experiment with other models like Random Forest or Support Vector Regression.
6. **Model Evaluation:** Calculates MSE and R-squared to evaluate the model’s performance.
7. **Model Saving (Optional):** Demonstrates how to save the trained model using joblib.

**Addressing Error (b): Old scikit-learn version**

The error “old scikit-learn version does not support parameters squared=False in the mean\_squared\_error function” means your scikit-learn version is too old. The squared=False parameter was introduced in a later version of scikit-learn.

**Solution:**

1. **Upgrade scikit-learn:**

* pip install --upgrade scikit-learn

1. **Remove squared=False (Alternative if upgrading isn’t possible):**

* If you *cannot* upgrade scikit-learn for some reason, remove the squared=False argument from the mean\_squared\_error function. The default behavior (without squared=False) is to return the *mean squared error*. If you want the root mean squared error (RMSE), take the square root of the MSE:
* mse = mean\_squared\_error(y\_test, y\_pred) # Remove squared=False  
  rmse = np.sqrt(mse) # Calculate the RMSE manually  
  print(f"Root Mean Squared Error: {rmse}")

**6. Five Examples**

Here are 5 example modifications or extensions you can make to the above code:

1. **Different Regression Model:** Replace LinearRegression with RandomForestRegressor or SVR (Support Vector Regression) from scikit-learn. You’ll need to import the appropriate class from sklearn.ensemble or sklearn.svm. Experiment with hyperparameter tuning for better performance.

* from sklearn.ensemble import RandomForestRegressor  
  model = RandomForestRegressor(n\_estimators=100, random\_state=42) # Example hyperparameters

1. **More Molecular Descriptors:** Add more RDKit descriptors. Explore the rdkit.Chem.Descriptors module for properties like LogP, number of hydrogen bond donors/acceptors, TPSA, etc. Add these as columns to your DataFrame. Be careful of multicollinearity (highly correlated descriptors).

* df['logp'] = df['molecule'].apply(lambda x: Descriptors.MolLogP(x))  
  df['hbd'] = df['molecule'].apply(lambda x: Descriptors.NumHDonors(x))

1. **Different Fingerprint Type:** Experiment with different types of RDKit fingerprints, such as AtomPairFP or TopologicalTorsion. You’ll need to adjust the code to use the appropriate RDKit functions.

* from rdkit.Chem import AtomPairs  
  def calculate\_atom\_pair\_fingerprint(mol):  
   try:  
   return AtomPairs.GetAtomPairFingerprint(mol)  
   except:  
   return None  
    
  df['fingerprint'] = df['molecule'].apply(calculate\_atom\_pair\_fingerprint)
* **Important:** AtomPairFingerprint and TopologicalTorsionFingerprint are sparse fingerprints. You might need to convert them to dense vectors before using them in scikit-learn:
* from rdkit.Chem import AtomPairs  
  from scipy.sparse import csr\_matrix  
  def calculate\_atom\_pair\_fingerprint(mol):  
   try:  
   fp = AtomPairs.GetAtomPairFingerprint(mol)  
   return fp  
   except:  
   return None  
  df['fingerprint'] = df['molecule'].apply(calculate\_atom\_pair\_fingerprint)  
  df = df.dropna(subset=['fingerprint'])  
  # Convert sparse fingerprints to dense vectors  
  X = []  
  for fp in df['fingerprint']:  
   X.append(fp.ToList()) # Convert to a list of integers  
  X = np.array(X)

1. **Data Visualization:** Create more informative visualizations using matplotlib or seaborn. Examples include scatter plots of predicted vs. actual values, box plots of bioactivity values for different compound classes, or heatmaps of descriptor correlations.

* plt.figure(figsize=(8, 6))  
  plt.scatter(y\_test, y\_pred, alpha=0.5)  
  plt.xlabel('Actual Log IC50')  
  plt.ylabel('Predicted Log IC50')  
  plt.title('Predicted vs. Actual Log IC50')  
  plt.show()

1. **Cross-Validation:** Implement cross-validation to get a more robust estimate of model performance. Use KFold or cross\_val\_score from sklearn.model\_selection.

* from sklearn.model\_selection import cross\_val\_score, KFold  
  model = LinearRegression()  
  cv = KFold(n\_splits=5, shuffle=True, random\_state=42) # 5-fold cross-validation  
  scores = cross\_val\_score(model, X\_scaled, y, cv=cv, scoring='neg\_mean\_squared\_error')  
  mse\_scores = -scores # Convert negative MSE to positive MSE  
  print("Cross-validation MSE:", mse\_scores)  
  print("Mean MSE:", mse\_scores.mean())

**7. Error Handling and Logging**

Incorporate comprehensive error handling (using try...except blocks) and logging to make your code more robust and easier to debug. Use the logging module in Python.

**Summary (In summary)**

You now have a solid foundation for your ChEMBL 35 data analysis project. Remember to adapt the code snippets and examples to your specific research questions and data. Good luck!

**Résumé (French Translation)**

Vous avez maintenant une base solide pour votre projet d’analyse de données ChEMBL 35. N’oubliez pas d’adapter les extraits de code et les exemples à vos questions de recherche spécifiques et à vos données. Bonne chance !

**Key Considerations:**

* **Database Connection Security:** Avoid hardcoding database credentials directly in your notebooks or code. Use environment variables or a configuration file (config/) to store sensitive information securely.
* **Reproducibility:** Use a requirements.txt file to specify all Python dependencies. This ensures that your project can be easily reproduced by others. Use a consistent random seed (random\_state) in your code for reproducibility.
* **Documentation:** Write clear and concise comments in your code and maintain a README.md file to document your project.

This comprehensive guide should help you get started and address the issues you’ve encountered. Let me know if you have any more questions.