# PharmApp Suite

## 🧠 AI for Drug Discovery and Development 🧪

Copyright 2025 | Nghiên Cứu Thuốc | www.nghiencuuthuoc.com | Zalo: +84888999311 |

## 🧩 Topic: Phân tích thuốc dành cho trẻ em

Tuyệt vời! Dự án phân tích dữ liệu ChEMBL 35 bằng RDKit là một hướng đi rất tiềm năng trong nghiên cứu và phát triển thuốc. Với vai trò là chuyên gia R&D Dược học, tôi sẽ hỗ trợ bạn xây dựng quy trình phân tích hiệu quả, khắc phục các lỗi và đưa ra các ví dụ code cụ thể.

**Tổng quan về chủ đề Topic\_CheMBL\_35\_58**

Chủ đề này có vẻ tập trung vào một khía cạnh cụ thể trong ChEMBL 35, có thể là một nhóm hợp chất, một loại mục tiêu sinh học (target), hoặc một tính chất hóa lý nào đó. Để đưa ra phân tích chính xác hơn, tôi cần biết tên chủ đề đầy đủ hoặc mô tả chi tiết hơn về mục tiêu của bạn. Tuy nhiên, dựa trên thông tin bạn cung cấp, tôi sẽ xây dựng một quy trình phân tích tổng quát, có thể áp dụng và điều chỉnh cho nhiều chủ đề khác nhau trong ChEMBL 35.

**1. Phân tích mô hình phân tích (Analysis Model)**

Mô hình phân tích của chúng ta sẽ bao gồm các bước sau:

1. **Data Extraction (Trích xuất dữ liệu):** Sử dụng SQL để truy vấn dữ liệu từ cơ sở dữ liệu ChEMBL 35, tập trung vào các bảng quan trọng như activities, molecule\_dictionary, compound\_structures, và target\_dictionary.
2. **Data Cleaning and Preprocessing (Làm sạch và tiền xử lý dữ liệu):**
   * Loại bỏ các giá trị null hoặc không hợp lệ.
   * Chuẩn hóa dữ liệu (ví dụ: chuyển đổi IC50, Ki về một đơn vị chung).
   * Lọc dữ liệu dựa trên các tiêu chí cụ thể (ví dụ: chỉ giữ lại các hoạt chất có IC50 < 10µM).
3. **Feature Engineering (Xây dựng đặc trưng):** Sử dụng RDKit để tính toán các đặc trưng phân tử (molecular descriptors) từ cấu trúc hóa học của các hợp chất. Các đặc trưng này có thể bao gồm:
   * Tính chất vật lý (ví dụ: LogP, MW, TPSA).
   * Hình dạng phân tử (ví dụ: số vòng, số nhánh).
   * Các đặc trưng dựa trên fingerprint (ví dụ: Morgan fingerprint, MACCS keys).
4. **Data Analysis and Modeling (Phân tích dữ liệu và xây dựng mô hình):**
   * **Exploratory Data Analysis (EDA):** Thống kê mô tả, trực quan hóa dữ liệu để hiểu rõ hơn về phân bố và mối quan hệ giữa các biến.
   * **Machine Learning (ML):** Xây dựng các mô hình dự đoán hoạt tính sinh học (ví dụ: sử dụng Random Forest, Support Vector Machines) hoặc phân loại hợp chất (ví dụ: dựa trên cấu trúc hoặc hoạt tính).
5. **Model Evaluation and Interpretation (Đánh giá và giải thích mô hình):** Đánh giá hiệu năng của mô hình bằng các metrics phù hợp (ví dụ: AUC, ROC, RMSE) và giải thích các yếu tố quan trọng ảnh hưởng đến hoạt tính.

**2. Hướng dẫn song ngữ (Bilingual Instructions)**

Dưới đây là hướng dẫn chi tiết cho từng bước, kèm theo giải thích bằng cả tiếng Anh và tiếng Việt:

**Step 1: Data Extraction (Trích xuất dữ liệu)**

* **English:** Use SQL to query data from the ChEMBL 35 database, focusing on tables like activities, molecule\_dictionary, compound\_structures, and target\_dictionary.
* **Tiếng Việt:** Sử dụng SQL để truy vấn dữ liệu từ cơ sở dữ liệu ChEMBL 35, tập trung vào các bảng như activities, molecule\_dictionary, compound\_structures, và target\_dictionary.

**Step 2: Data Cleaning and Preprocessing (Làm sạch và tiền xử lý dữ liệu)**

* **English:**
  + Remove null or invalid values.
  + Standardize data (e.g., convert IC50, Ki to a common unit).
  + Filter data based on specific criteria (e.g., keep only compounds with IC50 < 10µM).
* **Tiếng Việt:**
  + Loại bỏ các giá trị null hoặc không hợp lệ.
  + Chuẩn hóa dữ liệu (ví dụ: chuyển đổi IC50, Ki về một đơn vị chung).
  + Lọc dữ liệu dựa trên các tiêu chí cụ thể (ví dụ: chỉ giữ lại các hợp chất có IC50 < 10µM).

**Step 3: Feature Engineering (Xây dựng đặc trưng)**

* **English:** Use RDKit to calculate molecular descriptors from the chemical structures of the compounds. These descriptors can include:
  + Physicochemical properties (e.g., LogP, MW, TPSA).
  + Molecular shape (e.g., number of rings, number of branches).
  + Fingerprint-based features (e.g., Morgan fingerprint, MACCS keys).
* **Tiếng Việt:** Sử dụng RDKit để tính toán các đặc trưng phân tử từ cấu trúc hóa học của các hợp chất. Các đặc trưng này có thể bao gồm:
  + Tính chất vật lý (ví dụ: LogP, MW, TPSA).
  + Hình dạng phân tử (ví dụ: số vòng, số nhánh).
  + Các đặc trưng dựa trên fingerprint (ví dụ: Morgan fingerprint, MACCS keys).

**Step 4: Data Analysis and Modeling (Phân tích dữ liệu và xây dựng mô hình)**

* **English:**
  + **Exploratory Data Analysis (EDA):** Descriptive statistics, data visualization to understand the distribution and relationships between variables.
  + **Machine Learning (ML):** Build models to predict biological activity (e.g., using Random Forest, Support Vector Machines) or classify compounds (e.g., based on structure or activity).
* **Tiếng Việt:**
  + **Exploratory Data Analysis (EDA):** Thống kê mô tả, trực quan hóa dữ liệu để hiểu rõ hơn về phân bố và mối quan hệ giữa các biến.
  + **Machine Learning (ML):** Xây dựng các mô hình dự đoán hoạt tính sinh học (ví dụ: sử dụng Random Forest, Support Vector Machines) hoặc phân loại hợp chất (ví dụ: dựa trên cấu trúc hoặc hoạt tính).

**Step 5: Model Evaluation and Interpretation (Đánh giá và giải thích mô hình)**

* **English:** Evaluate the performance of the model using appropriate metrics (e.g., AUC, ROC, RMSE) and interpret the important factors affecting activity.
* **Tiếng Việt:** Đánh giá hiệu năng của mô hình bằng các metrics phù hợp (ví dụ: AUC, ROC, RMSE) và giải thích các yếu tố quan trọng ảnh hưởng đến hoạt tính.

**3. Code SQL, Python (English)**

Dưới đây là các ví dụ code SQL và Python, tập trung vào việc trích xuất dữ liệu, xây dựng đặc trưng và phân tích cơ bản.

**SQL (PostgreSQL)**

-- Example 1: Extracting data for a specific target (e.g., CHEMBL205)  
SELECT  
 act.standard\_value,  
 act.standard\_units,  
 md.chembl\_id,  
 cs.canonical\_smiles  
FROM  
 activities act  
JOIN  
 molecule\_dictionary md ON act.molregno = md.molregno  
JOIN  
 compound\_structures cs ON md.molregno = cs.molregno  
WHERE  
 act.target\_chembl\_id = 'CHEMBL205' -- Replace with your target CHEMBL\_ID  
 AND act.standard\_type = 'IC50'  
 AND act.standard\_relation = '='  
 AND act.standard\_value IS NOT NULL  
 AND act.standard\_value > 0  
 AND act.standard\_value < 10000 -- Filter IC50 values  
 AND cs.canonical\_smiles IS NOT NULL  
LIMIT 100;  
  
-- Example 2: Corrected SQL to filter numeric values for standard\_value  
SELECT  
 act.standard\_value,  
 act.standard\_units,  
 md.chembl\_id,  
 cs.canonical\_smiles  
FROM  
 activities act  
JOIN  
 molecule\_dictionary md ON act.molregno = md.molregno  
JOIN  
 compound\_structures cs ON md.molregno = cs.molregno  
WHERE  
 act.target\_chembl\_id = 'CHEMBL205'  
 AND act.standard\_type = 'IC50'  
 AND act.standard\_relation = '='  
 AND act.standard\_value IS NOT NULL  
 AND act.standard\_value::TEXT ~ '^[0-9\.]+$' -- Ensure standard\_value is numeric  
 AND act.standard\_value::NUMERIC > 0  
 AND act.standard\_value::NUMERIC < 10000  
 AND cs.canonical\_smiles IS NOT NULL  
LIMIT 100;  
  
-- Example 3: Extracting data for a specific activity type (e.g., Ki)  
SELECT  
 act.standard\_value,  
 act.standard\_units,  
 md.chembl\_id,  
 cs.canonical\_smiles  
FROM  
 activities act  
JOIN  
 molecule\_dictionary md ON act.molregno = md.molregno  
JOIN  
 compound\_structures cs ON md.molregno = cs.molregno  
WHERE  
 act.standard\_type = 'Ki'  
 AND act.standard\_relation = '='  
 AND act.standard\_value IS NOT NULL  
 AND act.standard\_value::TEXT ~ '^[0-9\.]+$'  
 AND act.standard\_value::NUMERIC > 0  
 AND act.standard\_value::NUMERIC < 10000  
 AND cs.canonical\_smiles IS NOT NULL  
LIMIT 100;  
  
-- Example 4: Extracting data with specific units (e.g., nM)  
SELECT  
 act.standard\_value,  
 act.standard\_units,  
 md.chembl\_id,  
 cs.canonical\_smiles  
FROM  
 activities act  
JOIN  
 molecule\_dictionary md ON act.molregno = md.molregno  
JOIN  
 compound\_structures cs ON md.molregno = cs.molregno  
WHERE  
 act.standard\_type = 'IC50'  
 AND act.standard\_units = 'nM'  
 AND act.standard\_relation = '='  
 AND act.standard\_value IS NOT NULL  
 AND act.standard\_value::TEXT ~ '^[0-9\.]+$'  
 AND act.standard\_value::NUMERIC > 0  
 AND act.standard\_value::NUMERIC < 10000  
 AND cs.canonical\_smiles IS NOT NULL  
LIMIT 100;  
  
-- Example 5: Extracting data for a specific protein target type  
SELECT  
 act.standard\_value,  
 act.standard\_units,  
 md.chembl\_id,  
 cs.canonical\_smiles  
FROM  
 activities act  
JOIN  
 molecule\_dictionary md ON act.molregno = md.molregno  
JOIN  
 compound\_structures cs ON md.molregno = cs.molregno  
JOIN  
 target\_dictionary td ON act.tid = td.tid  
WHERE  
 td.target\_type = 'PROTEIN'  
 AND act.standard\_type = 'IC50'  
 AND act.standard\_relation = '='  
 AND act.standard\_value IS NOT NULL  
 AND act.standard\_value::TEXT ~ '^[0-9\.]+$'  
 AND act.standard\_value::NUMERIC > 0  
 AND act.standard\_value::NUMERIC < 10000  
 AND cs.canonical\_smiles IS NOT NULL  
LIMIT 100;

**Python (Jupyter Notebook)**

import os  
import pandas as pd  
import psycopg2  
from rdkit import Chem  
from rdkit.Chem import Descriptors  
import numpy as np  
from sklearn.model\_selection import train\_test\_split  
from sklearn.ensemble import RandomForestRegressor  
from sklearn.metrics import mean\_squared\_error, r2\_score  
  
# Database credentials  
DB\_HOST = "192.168.206.136"  
DB\_USER = "rd"  
DB\_PASS = "rd"  
DB\_NAME = "chembl\_35"  
base\_path = "." # Assuming the notebook is in the root directory  
  
# Function to connect to the database  
def connect\_to\_db(host, user, password, database):  
 conn = psycopg2.connect(host=host, user=user, password=password, database=database)  
 return conn  
  
# Function to execute SQL query and return a Pandas DataFrame  
def execute\_sql\_query(conn, query):  
 df = pd.read\_sql\_query(query, conn)  
 return df  
  
# Function to calculate molecular descriptors using RDKit  
def calculate\_descriptors(smiles):  
 mol = Chem.MolFromSmiles(smiles)  
 if mol is None:  
 return None  
 descriptors = {}  
 descriptors['MW'] = Descriptors.MolWt(mol)  
 descriptors['LogP'] = Descriptors.MolLogP(mol)  
 # Add more descriptors as needed  
 return descriptors  
  
# Main function  
def main():  
 # Connect to the database  
 conn = connect\_to\_db(DB\_HOST, DB\_USER, DB\_PASS, DB\_NAME)  
  
 # SQL query to extract data  
 sql\_query = """  
 SELECT  
 act.standard\_value,  
 act.standard\_units,  
 md.chembl\_id,  
 cs.canonical\_smiles  
 FROM  
 activities act  
 JOIN  
 molecule\_dictionary md ON act.molregno = md.molregno  
 JOIN  
 compound\_structures cs ON md.molregno = cs.molregno  
 WHERE  
 act.target\_chembl\_id = 'CHEMBL205' -- Replace with your target CHEMBL\_ID  
 AND act.standard\_type = 'IC50'  
 AND act.standard\_relation = '='  
 AND act.standard\_value IS NOT NULL  
 AND act.standard\_value::TEXT ~ '^[0-9\.]+$' -- Ensure standard\_value is numeric  
 AND act.standard\_value::NUMERIC > 0  
 AND act.standard\_value::NUMERIC < 10000  
 AND cs.canonical\_smiles IS NOT NULL  
 LIMIT 100;  
 """  
  
 # Execute the SQL query  
 df = execute\_sql\_query(conn, sql\_query)  
 conn.close()  
  
 # Data Cleaning and Preprocessing  
 df = df.dropna(subset=['canonical\_smiles', 'standard\_value']) # Drop rows with missing SMILES or standard\_value  
 df['standard\_value'] = pd.to\_numeric(df['standard\_value']) # Ensure standard\_value is numeric  
  
 # Feature Engineering  
 descriptors\_list = []  
 for smiles in df['canonical\_smiles']:  
 descriptors = calculate\_descriptors(smiles)  
 descriptors\_list.append(descriptors)  
  
 descriptors\_df = pd.DataFrame(descriptors\_list)  
 df = pd.concat([df, descriptors\_df], axis=1)  
 df = df.dropna() # Drop rows with missing descriptors  
  
 # Data Analysis and Modeling  
 X = df[['MW', 'LogP']] # Features  
 y = np.log10(df['standard\_value']) # Target variable (log transform for better distribution)  
  
 X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)  
  
 model = RandomForestRegressor(n\_estimators=100, random\_state=42)  
 model.fit(X\_train, y\_train)  
  
 y\_pred = model.predict(X\_test)  
  
 # Model Evaluation  
 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 DataFrame to a CSV file (optional)  
 output\_file = os.path.join(base\_path, "data", "chembl\_data.csv")  
 df.to\_csv(output\_file, index=False)  
 print(f"Data saved to {output\_file}")  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()

**Explanation:**

* **SQL:** The SQL query extracts standard\_value, standard\_units, chembl\_id, and canonical\_smiles for a specific target from the ChEMBL database. The WHERE clause filters the data based on the target ID, activity type (IC50), relation (=), and ensures that the standard\_value is numeric, not null, and within a reasonable range. The LIMIT clause restricts the output to the first 100 rows. The corrected SQL uses ::TEXT ~ '^[0-9\.]+$' and ::NUMERIC to properly handle numeric filtering.
* **Python:**
  + The Python code connects to the ChEMBL database using the provided credentials.
  + It executes the SQL query and loads the data into a Pandas DataFrame.
  + It calculates molecular descriptors (MW and LogP) using RDKit.
  + It builds a Random Forest Regression model to predict the standard\_value based on the calculated descriptors.
  + It evaluates the model using Mean Squared Error (MSE) and R-squared.
  + Finally, it saves the processed DataFrame to a CSV file.

**Fixing the Errors:**

* **Error a: ERROR: operator does not exist: numeric ~ unknown LINE 12: AND act.standard\_value ~ '^[0-9\.]+$'**
* This error occurs because PostgreSQL’s ~ operator (regular expression match) is not directly applicable to numeric columns. To fix this, you need to cast the standard\_value to text before using the ~ operator and cast to numeric for comparison:
* AND act.standard\_value::TEXT ~ '^[0-9\.]+$' -- Ensure standard\_value is numeric  
  AND act.standard\_value::NUMERIC > 0  
  AND act.standard\_value::NUMERIC < 10000
* **Error b: mean\_squared\_error does not support squared=False**
* This error indicates that you are using an older version of scikit-learn. Either upgrade scikit-learn to a version that supports squared=False (which returns the root mean squared error (RMSE) directly) or calculate the RMSE manually:
* mse = mean\_squared\_error(y\_test, y\_pred)  
  rmse = np.sqrt(mse) # Calculate RMSE manually  
  print(f"Mean Squared Error: {mse}")  
  print(f"Root Mean Squared Error: {rmse}") # Print RMSE

**4. Five Examples of SQL and Python Code**

Here are five more examples demonstrating different data extraction and analysis techniques:

**Example 1: Filtering by Molecular Weight (SQL & Python)**

* **SQL:** Extract compounds with molecular weight between 200 and 500. This requires joining with a table that contains molecular weight information (which may require creating a custom table or function if not directly available in ChEMBL). The example assumes a table called molecule\_properties with a molregno and molecular\_weight column.

SELECT  
 act.standard\_value,  
 act.standard\_units,  
 md.chembl\_id,  
 cs.canonical\_smiles  
FROM  
 activities act  
JOIN  
 molecule\_dictionary md ON act.molregno = md.molregno  
JOIN  
 compound\_structures cs ON md.molregno = cs.molregno  
LEFT JOIN  
 molecule\_properties mp ON md.molregno = mp.molregno -- Assuming a table with MW info  
WHERE  
 act.target\_chembl\_id = 'CHEMBL205'  
 AND act.standard\_type = 'IC50'  
 AND act.standard\_relation = '='  
 AND act.standard\_value IS NOT NULL  
 AND act.standard\_value::TEXT ~ '^[0-9\.]+$'  
 AND act.standard\_value::NUMERIC > 0  
 AND act.standard\_value::NUMERIC < 10000  
 AND cs.canonical\_smiles IS NOT NULL  
 AND mp.molecular\_weight BETWEEN 200 AND 500 -- Filter by molecular weight  
LIMIT 100;

* **Python:** Add molecular weight filtering after extracting the data using RDKit.

# ... (previous code) ...  
  
 # Feature Engineering (including MW calculation)  
 descriptors\_list = []  
 for smiles in df['canonical\_smiles']:  
 mol = Chem.MolFromSmiles(smiles)  
 if mol is None:  
 descriptors\_list.append(None)  
 continue  
 descriptors = {}  
 descriptors['MW'] = Descriptors.MolWt(mol)  
 descriptors['LogP'] = Descriptors.MolLogP(mol)  
 descriptors\_list.append(descriptors)  
  
 descriptors\_df = pd.DataFrame(descriptors\_list)  
 df = pd.concat([df, descriptors\_df], axis=1)  
 df = df.dropna()  
  
 # Filter by molecular weight  
 df = df[(df['MW'] >= 200) & (df['MW'] <= 500)]  
  
 # ... (rest of the code) ...

**Example 2: Using Morgan Fingerprints (Python)**

* **Python:** Calculate Morgan fingerprints and use them as features in the model.

from rdkit.Chem import AllChem  
  
# Function to calculate Morgan Fingerprint  
def calculate\_morgan\_fingerprint(smiles, radius=2, nBits=2048):  
 mol = Chem.MolFromSmiles(smiles)  
 if mol is None:  
 return None  
 fp = AllChem.GetMorganFingerprintAsBitVect(mol, radius, nBits=nBits)  
 return fp  
  
# ... (previous code) ...  
  
 # Feature Engineering (including Morgan Fingerprint)  
 fingerprints = []  
 for smiles in df['canonical\_smiles']:  
 fp = calculate\_morgan\_fingerprint(smiles)  
 if fp is None:  
 fingerprints.append(None)  
 else:  
 fingerprints.append(fp)  
  
 df['fingerprint'] = fingerprints  
 df = df.dropna(subset=['fingerprint'])  
  
 # Convert fingerprints to a matrix for machine learning  
 X = np.array([list(fp) for fp in df['fingerprint']])  
 y = np.log10(df['standard\_value'])  
  
 X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)  
  
 model = RandomForestRegressor(n\_estimators=100, random\_state=42)  
 model.fit(X\_train, y\_train)  
  
 # ... (rest of the code) ...

**Example 3: Extracting Data for Multiple Target CHEMBL IDs (SQL & Python)**

* **SQL:** Extract data for several targets at once.

SELECT  
 act.standard\_value,  
 act.standard\_units,  
 md.chembl\_id,  
 cs.canonical\_smiles  
FROM  
 activities act  
JOIN  
 molecule\_dictionary md ON act.molregno = md.molregno  
JOIN  
 compound\_structures cs ON md.molregno = cs.molregno  
WHERE  
 act.target\_chembl\_id IN ('CHEMBL205', 'CHEMBL206', 'CHEMBL207') -- Multiple targets  
 AND act.standard\_type = 'IC50'  
 AND act.standard\_relation = '='  
 AND act.standard\_value IS NOT NULL  
 AND act.standard\_value::TEXT ~ '^[0-9\.]+$'  
 AND act.standard\_value::NUMERIC > 0  
 AND act.standard\_value::NUMERIC < 10000  
 AND cs.canonical\_smiles IS NOT NULL  
LIMIT 100;

* **Python:** No significant changes needed in the Python code, as it processes the data returned by the SQL query.

**Example 4: Calculating TPSA (Topological Polar Surface Area) (Python)**

* **Python:** Add TPSA calculation to the molecular descriptor calculation.

from rdkit.Chem import rdMolDescriptors  
  
# Function to calculate molecular descriptors including TPSA  
def calculate\_descriptors(smiles):  
 mol = Chem.MolFromSmiles(smiles)  
 if mol is None:  
 return None  
 descriptors = {}  
 descriptors['MW'] = Descriptors.MolWt(mol)  
 descriptors['LogP'] = Descriptors.MolLogP(mol)  
 descriptors['TPSA'] = rdMolDescriptors.CalcTPSA(mol) # Add TPSA  
 return descriptors  
  
# ... (rest of the code) ...

**Example 5: Using a Different Machine Learning Model (Python)**

* **Python:** Replace the Random Forest Regressor with a Support Vector Regressor (SVR).

from sklearn.svm import SVR  
from sklearn.preprocessing import StandardScaler  
from sklearn.pipeline import make\_pipeline  
  
# ... (previous code) ...  
  
 # Data Analysis and Modeling  
 X = df[['MW', 'LogP']]  
 y = np.log10(df['standard\_value'])  
  
 X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)  
  
 # Feature scaling is often important for SVR  
 scaler = StandardScaler()  
 X\_train = scaler.fit\_transform(X\_train)  
 X\_test = scaler.transform(X\_test)  
  
 # Create an SVR model  
 model = make\_pipeline(SVR(kernel='rbf')) # Use a pipeline for scaling and SVR  
 model.fit(X\_train, y\_train)  
  
 y\_pred = model.predict(X\_test)  
  
 # Model Evaluation  
 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}")  
  
 # ... (rest of the code) ...

**Important Considerations:**

* **Database Connection:** Ensure your database connection details (IP address, username, password, database name) are correct. It’s best practice to store these credentials in environment variables rather than hardcoding them in your script.
* **Error Handling:** Add more robust error handling to your code, especially when dealing with database connections, file I/O, and RDKit calculations.
* **Data Scaling:** For machine learning models like SVR and linear regression, scaling your data (e.g., using StandardScaler) can significantly improve performance.
* **Feature Selection:** Experiment with different sets of molecular descriptors to find the features that best predict activity.
* **Model Tuning:** Tune the hyperparameters of your machine learning models (e.g., n\_estimators for Random Forest, C and gamma for SVR) using techniques like cross-validation to optimize performance.
* **Virtual Environments:** Use virtual environments (e.g., venv or conda) to manage your Python dependencies and ensure reproducibility.
* **Logging:** Implement logging to track the progress of your script and debug any issues.
* **File Paths:** Double-check all file paths to ensure they are correct and that you have the necessary permissions to read and write files.

By following these guidelines and adapting the examples to your specific needs, you can build a powerful and effective data analysis pipeline for your ChEMBL 35 project. Let me know if you have any further questions or need more specific assistance!