# PharmApp Suite

## 🧠 AI for Drug Discovery and Development 🧪

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

## 🧩 Topic: Tinh chỉnh siêu tham số bằng Optuna cho mô hình thuốc

Tuyệt vời! Tôi rất vui được hỗ trợ bạn trong dự án phân tích dữ liệu ChEMBL 35 này. Với vai trò là một chuyên gia nghiên cứu và phát triển dược học, tôi sẽ cung cấp cho bạn các phân tích, hướng dẫn, code mẫu (SQL và Python) và các ví dụ cụ thể để bạn có thể triển khai dự án một cách hiệu quả nhất.

**Tổng quan về dự án:**

Dự án của bạn tập trung vào việc phân tích dữ liệu từ cơ sở dữ liệu ChEMBL 35, kết hợp với thư viện RDKit để hỗ trợ các nghiên cứu và phát triển thuốc. Việc sử dụng cấu trúc thư mục theo chuẩn MLOps sẽ giúp bạn tổ chức và quản lý dự án một cách chuyên nghiệp.

**Phân tích mô hình:**

Mục tiêu chính của bạn có thể là:

1. **Khám phá dữ liệu (Exploratory Data Analysis - EDA):**
   * Hiểu rõ cấu trúc và nội dung của dữ liệu ChEMBL 35.
   * Xác định các thuộc tính quan trọng và mối quan hệ giữa chúng.
   * Tìm kiếm các xu hướng và mẫu tiềm năng có thể giúp ích cho việc phát triển thuốc.
2. **Xây dựng mô hình dự đoán (Predictive Modeling):**
   * Dự đoán hoạt tính sinh học của các phân tử dựa trên cấu trúc hóa học của chúng.
   * Xác định các đặc điểm cấu trúc quan trọng ảnh hưởng đến hoạt tính.
   * Tối ưu hóa cấu trúc phân tử để cải thiện hoạt tính.
3. **Phân tích QSAR/QSPR (Quantitative Structure-Activity Relationship/Quantitative Structure-Property Relationship):**
   * Xây dựng mô hình liên hệ giữa cấu trúc hóa học và hoạt tính sinh học/tính chất vật lý của phân tử.
   * Sử dụng mô hình để dự đoán hoạt tính/tính chất của các phân tử mới.
   * Tìm kiếm các phân tử có tiềm năng trở thành thuốc.

**Hướng dẫn song ngữ:**

**1. Kết nối đến cơ sở dữ liệu ChEMBL 35 (Connecting to ChEMBL 35 Database):**

* **Tiếng Việt:** Sử dụng thư viện psycopg2 trong Python để kết nối đến cơ sở dữ liệu PostgreSQL.
* **English:** Use the psycopg2 library in Python to connect to the PostgreSQL database.

import psycopg2  
  
# Database credentials  
db\_params = {  
 'host': '192.168.206.136',  
 'user': 'rd',  
 'password': 'rd',  
 'database': 'chembl\_35'  
}  
  
# Function to connect to the database  
def connect\_to\_db(params):  
 try:  
 conn = psycopg2.connect(\*\*params)  
 print("Connected to the database successfully!")  
 return conn  
 except psycopg2.Error as e:  
 print(f"Error connecting to the database: {e}")  
 return None  
  
# Establish connection  
conn = connect\_to\_db(db\_params)

**2. Truy vấn dữ liệu từ cơ sở dữ liệu (Querying Data from the Database):**

* **Tiếng Việt:** Sử dụng SQL để truy vấn dữ liệu từ các bảng trong cơ sở dữ liệu ChEMBL 35.
* **English:** Use SQL to query data from the tables in the ChEMBL 35 database.

-- SQL query to retrieve data from the 'activities' and 'molecule\_dictionary' tables  
-- Retrieving only 100 records for demonstration purposes  
SELECT act.molregno, md.chembl\_id, act.standard\_type, act.standard\_value, act.standard\_units  
FROM activities act  
JOIN molecule\_dictionary md ON act.molregno = md.molregno  
WHERE act.standard\_type = 'IC50'  
 AND act.standard\_units = 'nM'  
 AND act.standard\_value IS NOT NULL  
 AND act.standard\_value > 0  
LIMIT 100;

**3. Tiền xử lý dữ liệu (Data Preprocessing):**

* **Tiếng Việt:** Làm sạch và chuẩn hóa dữ liệu để đảm bảo chất lượng và tính nhất quán. Loại bỏ các giá trị thiếu, xử lý các giá trị ngoại lệ và chuyển đổi dữ liệu về định dạng phù hợp.
* **English:** Clean and normalize the data to ensure quality and consistency. Remove missing values, handle outliers, and convert data to the appropriate format.

import pandas as pd  
import numpy as np  
  
# Function to load data from a CSV file  
def load\_data(file\_path):  
 try:  
 data = pd.read\_csv(file\_path)  
 print(f"Data loaded successfully from {file\_path}")  
 return data  
 except FileNotFoundError:  
 print(f"Error: File not found at {file\_path}")  
 return None  
  
# Function to clean the data  
def clean\_data(df):  
 # Remove missing values  
 df = df.dropna()  
 # Remove duplicate rows  
 df = df.drop\_duplicates()  
 # Convert 'standard\_value' to numeric, handling errors  
 df['standard\_value'] = pd.to\_numeric(df['standard\_value'], errors='coerce')  
 # Remove rows where 'standard\_value' is NaN after conversion  
 df = df.dropna(subset=['standard\_value'])  
 # Filter out non-positive values  
 df = df[df['standard\_value'] > 0]  
 print("Data cleaning complete.")  
 return df  
  
# Example usage:  
# Assuming base\_path is defined and the CSV file is in the data directory  
base\_path = '.' # Replace with your actual base path  
csv\_file\_path = os.path.join(base\_path, 'data', 'your\_data\_file.csv') # Replace with your actual file name  
data = load\_data(csv\_file\_path)  
  
if data is not None:  
 cleaned\_data = clean\_data(data)  
 print(cleaned\_data.head())

**4. Tính toán đặc trưng phân tử (Molecular Feature Calculation):**

* **Tiếng Việt:** Sử dụng RDKit để tính toán các đặc trưng hóa học của phân tử, như trọng lượng phân tử, độ tan, số lượng liên kết, v.v.
* **English:** Use RDKit to calculate molecular features such as molecular weight, solubility, number of bonds, etc.

from rdkit import Chem  
from rdkit.Chem import Descriptors  
  
# Function to calculate molecular descriptors using RDKit  
def calculate\_descriptors(smiles):  
 try:  
 mol = Chem.MolFromSmiles(smiles)  
 if mol is None:  
 return None # Handle invalid SMILES strings  
  
 descriptors = {}  
 descriptors['MolWt'] = Descriptors.MolWt(mol)  
 descriptors['LogP'] = Descriptors.MolLogP(mol)  
 descriptors['NumHAcceptors'] = Descriptors.NumHAcceptors(mol)  
 descriptors['NumHDonors'] = Descriptors.NumHDonors(mol)  
 return descriptors  
 except Exception as e:  
 print(f"Error calculating descriptors for SMILES {smiles}: {e}")  
 return None  
  
# Example usage:  
# Assuming you have a DataFrame with a 'smiles' column  
def add\_descriptors\_to\_df(df, smiles\_column='smiles'):  
 # Apply the descriptor calculation to each SMILES in the DataFrame  
 df['descriptors'] = df[smiles\_column].apply(calculate\_descriptors)  
  
 # Expand the 'descriptors' column into separate columns  
 df = pd.concat([df, df['descriptors'].apply(pd.Series)], axis=1)  
  
 # Remove the original 'descriptors' column  
 df = df.drop(columns=['descriptors'])  
  
 return df  
  
# Example of how to use this function (assuming 'cleaned\_data' DataFrame is available)  
# Ensure that 'cleaned\_data' DataFrame has a 'smiles' column  
# cleaned\_data = add\_descriptors\_to\_df(cleaned\_data, smiles\_column='smiles')  
# print(cleaned\_data.head())

**5. Xây dựng mô hình học máy (Machine Learning Model Building):**

* **Tiếng Việt:** Sử dụng các thuật toán học máy để xây dựng mô hình dự đoán hoạt tính sinh học của phân tử.
* **English:** Use machine learning algorithms to build a model for predicting the biological activity of molecules.

from sklearn.model\_selection import train\_test\_split  
from sklearn.ensemble import RandomForestRegressor  
from sklearn.metrics import mean\_squared\_error, r2\_score  
  
# Function to train a machine learning model  
def train\_model(df, features, target):  
 # Split the data into training and testing sets  
 X\_train, X\_test, y\_train, y\_test = train\_test\_split(df[features], df[target], test\_size=0.2, random\_state=42)  
  
 # Initialize the model  
 model = RandomForestRegressor(n\_estimators=100, random\_state=42)  
  
 # Train the model  
 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^2 Score: {r2}")  
  
 return model, X\_test, y\_test, y\_pred  
  
# Example usage:  
# Assuming you have a DataFrame with calculated descriptors and activity values  
# Ensure that 'cleaned\_data' DataFrame has the necessary columns for features and target  
# Define the features and target  
# features = ['MolWt', 'LogP', 'NumHAcceptors', 'NumHDonors'] # Replace with your feature columns  
# target = 'standard\_value' # Replace with your target column  
  
# Train the model  
# model, X\_test, y\_test, y\_pred = train\_model(cleaned\_data, features, target)

**Sửa lỗi:**

**a. ERROR: operator does not exist: numeric ~ unknown:**

* **Tiếng Việt:** Lỗi này xảy ra do bạn đang cố gắng sử dụng toán tử ~ (tương tự như LIKE trong SQL) để so sánh một cột số (numeric) với một chuỗi (unknown). Để khắc phục, bạn cần đảm bảo rằng cả hai vế của toán tử ~ đều là chuỗi, hoặc sử dụng các toán tử so sánh số học phù hợp (ví dụ: =, >, <). Trong trường hợp này, bạn có thể loại bỏ điều kiện act.standard\_value ~ '^[0-9\.]+$' vì nó không cần thiết.
* **English:** This error occurs because you are trying to use the ~ operator (similar to LIKE in SQL) to compare a numeric column with a string. To fix this, ensure that both sides of the ~ operator are strings, or use appropriate numeric comparison operators (e.g., =, >, <). In this case, you can remove the condition act.standard\_value ~ '^[0-9\.]+$' as it is not necessary.

-- Corrected SQL query  
SELECT act.molregno, md.chembl\_id, act.standard\_type, act.standard\_value, act.standard\_units  
FROM activities act  
JOIN molecule\_dictionary md ON act.molregno = md.molregno  
WHERE act.standard\_type = 'IC50'  
 AND act.standard\_units = 'nM'  
 AND act.standard\_value IS NOT NULL  
 AND act.standard\_value > 0  
LIMIT 100;

**b. Phiên bản scikit-learn cũ không hỗ trợ tham số squared=False trong hàm mean\_squared\_error:**

* **Tiếng Việt:** Nếu bạn đang sử dụng phiên bản scikit-learn cũ, hãy loại bỏ tham số squared=False khỏi hàm mean\_squared\_error. Nếu bạn muốn tính căn bậc hai của MSE (RMSE), bạn có thể sử dụng hàm np.sqrt() để tính toán sau.
* **English:** If you are using an older version of scikit-learn, remove the squared=False parameter from the mean\_squared\_error function. If you want to calculate the square root of MSE (RMSE), you can use the np.sqrt() function to calculate it afterwards.

from sklearn.metrics import mean\_squared\_error  
import numpy as np  
  
# Calculate Mean Squared Error  
mse = mean\_squared\_error(y\_test, y\_pred)  
  
# Calculate Root Mean Squared Error (RMSE)  
rmse = np.sqrt(mse)  
  
print(f"Mean Squared Error: {mse}")  
print(f"Root Mean Squared Error: {rmse}")

**5 Ví dụ code SQL và Python mẫu:**

**Ví dụ 1: Lọc các phân tử có hoạt tính IC50 dưới 100nM (Filtering molecules with IC50 activity below 100nM):**

* **SQL:**

SELECT md.chembl\_id, act.standard\_value  
FROM activities act  
JOIN molecule\_dictionary md ON act.molregno = md.molregno  
WHERE act.standard\_type = 'IC50'  
 AND act.standard\_units = 'nM'  
 AND act.standard\_value <= 100  
LIMIT 100;

* **Python:**

import pandas as pd  
import psycopg2  
  
# Database credentials  
db\_params = {  
 'host': '192.168.206.136',  
 'user': 'rd',  
 'password': 'rd',  
 'database': 'chembl\_35'  
}  
  
# SQL query to retrieve data from the 'activities' and 'molecule\_dictionary' tables  
sql\_query = """  
SELECT md.chembl\_id, act.standard\_value  
FROM activities act  
JOIN molecule\_dictionary md ON act.molregno = md.molregno  
WHERE act.standard\_type = 'IC50'  
 AND act.standard\_units = 'nM'  
 AND act.standard\_value <= 100  
LIMIT 100;  
"""  
  
# Function to execute the SQL query and return the data as a DataFrame  
def execute\_query(db\_params, sql\_query):  
 try:  
 # Establish a connection to the PostgreSQL database  
 conn = psycopg2.connect(\*\*db\_params)  
   
 # Create a cursor object  
 cur = conn.cursor()  
   
 # Execute the SQL query  
 cur.execute(sql\_query)  
   
 # Fetch all the results  
 results = cur.fetchall()  
   
 # Get column names from the cursor description  
 column\_names = [desc[0] for desc in cur.description]  
   
 # Convert the results into a pandas DataFrame  
 df = pd.DataFrame(results, columns=column\_names)  
   
 # Close the cursor and connection  
 cur.close()  
 conn.close()  
   
 print("Query executed successfully!")  
 return df  
 except psycopg2.Error as e:  
 print(f"Error executing query: {e}")  
 return None  
  
# Execute the query and store the results in a DataFrame  
df = execute\_query(db\_params, sql\_query)  
  
# Print the DataFrame  
if df is not None:  
 print(df.head())

**Ví dụ 2: Tính trọng lượng phân tử trung bình (Calculating average molecular weight):**

* **SQL:**

SELECT AVG(md.molecular\_weight) AS average\_molecular\_weight  
FROM molecule\_dictionary md  
LIMIT 100;

* **Python:**

import pandas as pd  
import psycopg2  
  
# Database credentials  
db\_params = {  
 'host': '192.168.206.136',  
 'user': 'rd',  
 'password': 'rd',  
 'database': 'chembl\_35'  
}  
  
# SQL query to retrieve data from the 'activities' and 'molecule\_dictionary' tables  
sql\_query = """  
SELECT md.molecular\_weight  
FROM molecule\_dictionary md  
LIMIT 100;  
"""  
  
# Function to execute the SQL query and return the data as a DataFrame  
def execute\_query(db\_params, sql\_query):  
 try:  
 # Establish a connection to the PostgreSQL database  
 conn = psycopg2.connect(\*\*db\_params)  
   
 # Create a cursor object  
 cur = conn.cursor()  
   
 # Execute the SQL query  
 cur.execute(sql\_query)  
   
 # Fetch all the results  
 results = cur.fetchall()  
   
 # Get column names from the cursor description  
 column\_names = [desc[0] for desc in cur.description]  
   
 # Convert the results into a pandas DataFrame  
 df = pd.DataFrame(results, columns=column\_names)  
   
 # Close the cursor and connection  
 cur.close()  
 conn.close()  
   
 print("Query executed successfully!")  
 return df  
 except psycopg2.Error as e:  
 print(f"Error executing query: {e}")  
 return None  
  
# Execute the query and store the results in a DataFrame  
df = execute\_query(db\_params, sql\_query)  
if df is not None:  
 average\_molecular\_weight = df['molecular\_weight'].mean()  
 print(f"Average Molecular Weight: {average\_molecular\_weight}")

**Ví dụ 3: Tìm các phân tử có chứa vòng benzen (Finding molecules containing a benzene ring):**

* **SQL:**

-- This SQL query will not directly identify molecules with a benzene ring.  
-- You would typically use substructure searching capabilities within ChEMBL's interface or RDKit.  
-- This is a placeholder and might require a different approach using ChEMBL's API or RDKit integration.  
SELECT md.chembl\_id  
FROM molecule\_dictionary md  
WHERE md.molecule\_structures LIKE '%c1ccccc1%' -- This is a simplistic approach and might not be accurate.  
LIMIT 100;

* **Python:**

from rdkit import Chem  
from rdkit.Chem import AllChem  
import pandas as pd  
import psycopg2  
  
# Database credentials  
db\_params = {  
 'host': '192.168.206.136',  
 'user': 'rd',  
 'password': 'rd',  
 'database': 'chembl\_35'  
}  
  
# Function to execute the SQL query and return the data as a DataFrame  
def execute\_query(db\_params, sql\_query):  
 try:  
 # Establish a connection to the PostgreSQL database  
 conn = psycopg2.connect(\*\*db\_params)  
   
 # Create a cursor object  
 cur = conn.cursor()  
   
 # Execute the SQL query  
 cur.execute(sql\_query)  
   
 # Fetch all the results  
 results = cur.fetchall()  
   
 # Get column names from the cursor description  
 column\_names = [desc[0] for desc in cur.description]  
   
 # Convert the results into a pandas DataFrame  
 df = pd.DataFrame(results, columns=column\_names)  
   
 # Close the cursor and connection  
 cur.close()  
 conn.close()  
   
 print("Query executed successfully!")  
 return df  
 except psycopg2.Error as e:  
 print(f"Error executing query: {e}")  
 return None  
  
# SQL query to retrieve data from the 'activities' and 'molecule\_dictionary' tables  
sql\_query = """  
SELECT md.chembl\_id, md.molecule\_structures  
FROM molecule\_dictionary md  
LIMIT 100;  
"""  
  
# Define the SMILES pattern for a benzene ring  
benzene\_smiles = 'c1ccccc1'  
benzene\_mol = Chem.MolFromSmiles(benzene\_smiles)  
  
# Function to check if a molecule contains a benzene ring  
def contains\_benzene(smiles):  
 mol = Chem.MolFromSmiles(smiles)  
 if mol is None:  
 return False  
 return mol.HasSubstructMatch(benzene\_mol)  
  
# Execute the query and store the results in a DataFrame  
df = execute\_query(db\_params, sql\_query)  
  
# Apply the function to check for benzene ring  
if df is not None:  
 df['has\_benzene'] = df['molecule\_structures'].apply(contains\_benzene)  
  
 # Filter the DataFrame to only include molecules with a benzene ring  
 benzene\_molecules = df[df['has\_benzene']]  
  
 # Print the DataFrame  
 print(benzene\_molecules.head())

**Ví dụ 4: Phân tích mối tương quan giữa LogP và hoạt tính (Analyzing the correlation between LogP and activity):**

* **SQL:** (Không thể thực hiện trực tiếp trong SQL, cần kết hợp với Python)
* **Python:**

import pandas as pd  
import psycopg2  
from rdkit import Chem  
from rdkit.Chem import Descriptors  
  
# Database credentials  
db\_params = {  
 'host': '192.168.206.136',  
 'user': 'rd',  
 'password': 'rd',  
 'database': 'chembl\_35'  
}  
  
# SQL query to retrieve data from the 'activities' and 'molecule\_dictionary' tables  
sql\_query = """  
SELECT md.chembl\_id, md.molecule\_structures, act.standard\_value  
FROM activities act  
JOIN molecule\_dictionary md ON act.molregno = md.molregno  
WHERE act.standard\_type = 'IC50'  
 AND act.standard\_units = 'nM'  
 AND act.standard\_value IS NOT NULL  
 AND act.standard\_value > 0  
LIMIT 100;  
"""  
  
# Function to execute the SQL query and return the data as a DataFrame  
def execute\_query(db\_params, sql\_query):  
 try:  
 # Establish a connection to the PostgreSQL database  
 conn = psycopg2.connect(\*\*db\_params)  
   
 # Create a cursor object  
 cur = conn.cursor()  
   
 # Execute the SQL query  
 cur.execute(sql\_query)  
   
 # Fetch all the results  
 results = cur.fetchall()  
   
 # Get column names from the cursor description  
 column\_names = [desc[0] for desc in cur.description]  
   
 # Convert the results into a pandas DataFrame  
 df = pd.DataFrame(results, columns=column\_names)  
   
 # Close the cursor and connection  
 cur.close()  
 conn.close()  
   
 print("Query executed successfully!")  
 return df  
 except psycopg2.Error as e:  
 print(f"Error executing query: {e}")  
 return None  
  
# Function to calculate LogP using RDKit  
def calculate\_logp(smiles):  
 try:  
 mol = Chem.MolFromSmiles(smiles)  
 if mol is None:  
 return None  
 return Descriptors.MolLogP(mol)  
 except:  
 return None  
  
# Execute the query and store the results in a DataFrame  
df = execute\_query(db\_params, sql\_query)  
  
# Calculate LogP for each molecule  
if df is not None:  
 df['logp'] = df['molecule\_structures'].apply(calculate\_logp)  
 df = df.dropna(subset=['logp', 'standard\_value'])  
  
 # Calculate the correlation between LogP and activity  
 correlation = df['logp'].corr(df['standard\_value'])  
 print(f"Correlation between LogP and IC50: {correlation}")

**Ví dụ 5: Tạo biểu đồ phân tán giữa trọng lượng phân tử và LogP (Creating a scatter plot between molecular weight and LogP):**

* **SQL:** (Không thể thực hiện trực tiếp trong SQL, cần kết hợp với Python)
* **Python:**

import pandas as pd  
import psycopg2  
from rdkit import Chem  
from rdkit.Chem import Descriptors  
import matplotlib.pyplot as plt  
  
# Database credentials  
db\_params = {  
 'host': '192.168.206.136',  
 'user': 'rd',  
 'password': 'rd',  
 'database': 'chembl\_35'  
}  
  
# SQL query to retrieve data from the 'activities' and 'molecule\_dictionary' tables  
sql\_query = """  
SELECT md.chembl\_id, md.molecule\_structures  
FROM molecule\_dictionary md  
LIMIT 100;  
"""  
  
# Function to execute the SQL query and return the data as a DataFrame  
def execute\_query(db\_params, sql\_query):  
 try:  
 # Establish a connection to the PostgreSQL database  
 conn = psycopg2.connect(\*\*db\_params)  
   
 # Create a cursor object  
 cur = conn.cursor()  
   
 # Execute the SQL query  
 cur.execute(sql\_query)  
   
 # Fetch all the results  
 results = cur.fetchall()  
   
 # Get column names from the cursor description  
 column\_names = [desc[0] for desc in cur.description]  
   
 # Convert the results into a pandas DataFrame  
 df = pd.DataFrame(results, columns=column\_names)  
   
 # Close the cursor and connection  
 cur.close()  
 conn.close()  
   
 print("Query executed successfully!")  
 return df  
 except psycopg2.Error as e:  
 print(f"Error executing query: {e}")  
 return None  
  
# Function to calculate molecular weight and LogP using RDKit  
def calculate\_mw\_logp(smiles):  
 try:  
 mol = Chem.MolFromSmiles(smiles)  
 if mol is None:  
 return None, None  
 mw = Descriptors.MolWt(mol)  
 logp = Descriptors.MolLogP(mol)  
 return mw, logp  
 except:  
 return None, None  
  
# Execute the query and store the results in a DataFrame  
df = execute\_query(db\_params, sql\_query)  
  
# Calculate molecular weight and LogP for each molecule  
if df is not None:  
 df['mw'], df['logp'] = zip(\*df['molecule\_structures'].apply(calculate\_mw\_logp))  
 df = df.dropna(subset=['mw', 'logp'])  
  
 # Create a scatter plot  
 plt.figure(figsize=(10, 6))  
 plt.scatter(df['mw'], df['logp'], alpha=0.5)  
 plt.xlabel('Molecular Weight')  
 plt.ylabel('LogP')  
 plt.title('Scatter Plot of Molecular Weight vs LogP')  
 plt.grid(True)  
 plt.show()

**Lưu ý quan trọng:**

* **SMILES Strings:** Đảm bảo rằng cột molecule\_structures chứa chuỗi SMILES hợp lệ.
* **Error Handling:** Xử lý các lỗi có thể xảy ra trong quá trình tính toán đặc trưng phân tử.
* **Data Scaling:** Cân nhắc việc chuẩn hóa hoặc масштабирование dữ liệu trước khi xây dựng mô hình học máy.
* **Feature Selection:** Lựa chọn các đặc trưng quan trọng nhất để cải thiện hiệu suất của mô hình.
* **Model Evaluation:** Đánh giá mô hình bằng các mét đo phù hợp (ví dụ: MSE, RMSE, R^2).

Tôi hy vọng những hướng dẫn và ví dụ này sẽ giúp bạn triển khai dự án của mình một cách hiệu quả. Nếu bạn có bất kỳ câu hỏi nào khác, đừng ngần ngại hỏi!