# PharmApp Suite

## 🧠 AI for Drug Discovery and Development 🧪

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

## 🧩 Topic: Converting ChEMBL SMILES to InChIKeys

Okay, I understand. You’re working on a drug discovery project using ChEMBL 35 data, RDKit, and PostgreSQL. You want to follow an MLOps-inspired structure, run your code in Jupyter notebooks, and need help addressing specific errors and creating a small, workable dataset (100 rows) for initial analysis.

Here’s a breakdown of the analysis, code, and examples in both English and Vietnamese, addressing your specific concerns.

**1. Analysis Model & Project Structure**

**English:**

Your project seems to be following a standard data analysis workflow for drug discovery:

* **Data Extraction & Preparation (SQL):** You’re using SQL to query the ChEMBL 35 database and extract relevant data. This likely involves filtering for specific activity types, potency values, and molecule properties. You’re saving this data to CSV files for easier access in Python.
* **Data Loading & Processing (Python):** You’re using Jupyter notebooks to load the data from the CSV files, process it using RDKit (for molecular representation and feature generation), and potentially perform machine learning tasks.
* **Machine Learning (Python):** The end goal is likely to build a model that can predict activity or other properties of molecules based on their structure and features. This could involve regression (predicting potency) or classification (predicting activity/inactivity).

**Vietnamese:**

Dự án của bạn dường như đang tuân theo quy trình phân tích dữ liệu tiêu chuẩn cho việc khám phá thuốc:

* **Trích xuất và Chuẩn bị Dữ liệu (SQL):** Bạn đang sử dụng SQL để truy vấn cơ sở dữ liệu ChEMBL 35 và trích xuất dữ liệu liên quan. Điều này có thể liên quan đến việc lọc các loại hoạt động cụ thể, giá trị hiệu lực và thuộc tính phân tử. Bạn đang lưu dữ liệu này vào các tệp CSV để dễ dàng truy cập hơn trong Python.
* **Tải và Xử lý Dữ liệu (Python):** Bạn đang sử dụng Jupyter Notebook để tải dữ liệu từ các tệp CSV, xử lý nó bằng RDKit (để biểu diễn phân tử và tạo đặc trưng), và có khả năng thực hiện các tác vụ học máy.
* **Học Máy (Python):** Mục tiêu cuối cùng có thể là xây dựng một mô hình có thể dự đoán hoạt động hoặc các thuộc tính khác của phân tử dựa trên cấu trúc và đặc trưng của chúng. Điều này có thể liên quan đến hồi quy (dự đoán hiệu lực) hoặc phân loại (dự đoán hoạt động/không hoạt động).

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

Here’s the code, addressing the errors and following your folder structure guidelines.

**a) SQL (../data/chembl\_35\_89\_data.csv)**

-- Query to extract ChEMBL data for analysis (limited to 100 rows)  
-- Save this as chembl\_35\_89\_data.csv  
  
SELECT  
 cmp.chembl\_id,  
 md.molfile,  
 act.standard\_type,  
 act.standard\_value,  
 act.standard\_units,  
 act.assay\_chembl\_id  
FROM  
 compound\_structures cmp  
JOIN  
 molecule\_dictionary md ON cmp.molregno = md.molregno  
JOIN  
 activities act ON md.molregno = act.molregno  
WHERE  
 act.standard\_type IN ('IC50', 'Ki', 'EC50') -- Consider relevant activity types  
 AND act.standard\_units = 'nM' -- Focus on nM units for consistency  
 AND act.standard\_value IS NOT NULL  
 AND act.standard\_value::text ~ '^[0-9.]+$' -- Ensure standard\_value is numeric  
LIMIT 100; -- Limit to 100 rows

**Explanation (SQL):**

* **Error Fix:** The original error ERROR: operator does not exist: numeric ~ unknown occurred because you were trying to use the ~ operator (regular expression matching) on a numeric column. PostgreSQL needs an explicit cast to text for that. The act.standard\_value::text ~ '^[0-9.]+$' line now correctly casts the standard\_value to text before applying the regex.
* **Data Selection:** Selects the molecule’s ChEMBL ID, molfile (SDF string), activity type, standard value, units, and assay ID.
* **Filtering:** Filters for common activity types (IC50, Ki, EC50) and standardizes units to ‘nM’. It also makes sure the standard\_value is not null and can be represented as a number.
* **Limiting:** Crucially limits the result to 100 rows to keep your machine from being overloaded.

**Vietnamese (SQL):**

-- Truy vấn để trích xuất dữ liệu ChEMBL để phân tích (giới hạn ở 100 hàng)  
-- Lưu truy vấn này thành chembl\_35\_89\_data.csv  
  
SELECT  
 cmp.chembl\_id,  
 md.molfile,  
 act.standard\_type,  
 act.standard\_value,  
 act.standard\_units,  
 act.assay\_chembl\_id  
FROM  
 compound\_structures cmp  
JOIN  
 molecule\_dictionary md ON cmp.molregno = md.molregno  
JOIN  
 activities act ON md.molregno = act.molregno  
WHERE  
 act.standard\_type IN ('IC50', 'Ki', 'EC50') -- Xem xét các loại hoạt động liên quan  
 AND act.standard\_units = 'nM' -- Tập trung vào đơn vị nM để nhất quán  
 AND act.standard\_value IS NOT NULL  
 AND act.standard\_value::text ~ '^[0-9.]+$' -- Đảm bảo standard\_value là số  
LIMIT 100; -- Giới hạn ở 100 hàng

**Giải thích (SQL):**

* **Sửa Lỗi:** Lỗi ban đầu ERROR: operator does not exist: numeric ~ unknown xảy ra vì bạn đang cố gắng sử dụng toán tử ~ (khớp biểu thức chính quy) trên cột numeric. PostgreSQL cần một kiểu chuyển đổi rõ ràng sang văn bản cho việc đó. Dòng act.standard\_value::text ~ '^[0-9.]+$' hiện chuyển đổi chính xác standard\_value thành văn bản trước khi áp dụng biểu thức chính quy.
* **Chọn Dữ liệu:** Chọn ID ChEMBL của phân tử, molfile (chuỗi SDF), loại hoạt động, giá trị tiêu chuẩn, đơn vị và ID xét nghiệm.
* **Lọc:** Lọc theo các loại hoạt động phổ biến (IC50, Ki, EC50) và chuẩn hóa các đơn vị thành ‘nM’. Nó cũng đảm bảo rằng standard\_value không phải là null và có thể được biểu diễn dưới dạng một số.
* **Giới hạn:** Quan trọng là giới hạn kết quả ở 100 hàng để tránh máy của bạn bị quá tải.

**b) Python (notebook/Topic\_CheMBL\_35\_89\_1\_data\_loading.ipynb - Data Loading & Processing)**

import os  
import pandas as pd  
from rdkit import Chem  
from rdkit.Chem import Descriptors  
import numpy as np # Import numpy  
  
# Define base path (adjust as needed based on your project structure)  
base\_path = ".." # Assuming the notebook is in the 'notebook' folder one level up from base  
data\_path = os.path.join(base\_path, "data", "chembl\_35\_89\_data.csv")  
  
# Load the data  
try:  
 df = pd.read\_csv(data\_path)  
except FileNotFoundError:  
 print(f"Error: CSV file not found at {data\_path}. Make sure you ran the SQL script and saved the CSV.")  
 exit()  
  
print(f"Data loaded successfully. Shape: {df.shape}")  
print(df.head())  
  
# Function to calculate molecular weight using RDKit  
def calculate\_molecular\_weight(molfile):  
 try:  
 mol = Chem.MolFromMolBlock(molfile)  
 if mol is not None:  
 return Descriptors.MolWt(mol)  
 else:  
 return np.nan # Handle cases where MolBlock parsing fails  
 except Exception as e:  
 print(f"Error processing MolBlock: {e}")  
 return np.nan  
  
# Apply the function to the dataframe  
df['molecular\_weight'] = df['molfile'].apply(calculate\_molecular\_weight)  
  
# Display the dataframe with molecular weight  
print(df.head())  
  
# Handle missing values (NaN) if any  
df = df.dropna()  
  
print(f"Dataframe shape after removing NaN values: {df.shape}")  
  
  
# Example of further processing: Convert standard\_value to numeric and take -log10(IC50)  
df = df[df['standard\_type'] == 'IC50'] # Filter for IC50 values only for this example  
df['standard\_value'] = pd.to\_numeric(df['standard\_value'], errors='coerce') # Convert to numeric, handling errors  
df = df.dropna(subset=['standard\_value']) # Remove rows with NaN standard\_value after conversion  
df['pIC50'] = -np.log10(df['standard\_value'] \* 1e-9) # Convert nM to M and take -log10  
  
print(df.head())

**Explanation (Python - Data Loading & Processing):**

* **Import Libraries:** Imports necessary libraries: os, pandas, RDKit (Chem, Descriptors).
* **Path Handling:** Uses os.path.join to construct the correct path to your CSV file. This is important for reproducibility.
* **Data Loading:** Loads the CSV file into a Pandas DataFrame using pd.read\_csv. Includes error handling for the case where the file is not found.
* **Molecular Weight Calculation:** Defines a function calculate\_molecular\_weight that takes a MolBlock string (from the molfile column), parses it using RDKit, and calculates the molecular weight. It also includes error handling to deal with potentially invalid MolBlocks.
* **Apply Function:** Applies the calculate\_molecular\_weight function to the molfile column to create a new column called molecular\_weight.
* **Handle Missing Values:** Removes rows with missing values (NaN) that might result from errors in MolBlock parsing.
* **Example Processing (pIC50):** Demonstrates further data processing:
  + Filters the DataFrame to include only rows where standard\_type is ‘IC50’.
  + Converts the standard\_value column to numeric using pd.to\_numeric, handling potential errors by setting invalid values to NaN.
  + Removes rows with NaN values in the standard\_value column (resulting from the conversion).
  + Calculates pIC50 values from IC50 (nM) values using the formula: pIC50 = -log10(IC50 \* 1e-9) (converting nM to M).

**Vietnamese (Python - Tải và Xử lý Dữ liệu):**

import os  
import pandas as pd  
from rdkit import Chem  
from rdkit.Chem import Descriptors  
import numpy as np # Nhập numpy  
  
# Xác định đường dẫn cơ sở (điều chỉnh theo cấu trúc dự án của bạn)  
base\_path = ".." # Giả sử notebook nằm trong thư mục 'notebook' một cấp trên cơ sở  
data\_path = os.path.join(base\_path, "data", "chembl\_35\_89\_data.csv")  
  
# Tải dữ liệu  
try:  
 df = pd.read\_csv(data\_path)  
except FileNotFoundError:  
 print(f"Lỗi: Không tìm thấy tệp CSV tại {data\_path}. Đảm bảo bạn đã chạy script SQL và lưu CSV.")  
 exit()  
  
print(f"Dữ liệu đã được tải thành công. Hình dạng: {df.shape}")  
print(df.head())  
  
# Hàm tính toán trọng lượng phân tử bằng RDKit  
def calculate\_molecular\_weight(molfile):  
 try:  
 mol = Chem.MolFromMolBlock(molfile)  
 if mol is not None:  
 return Descriptors.MolWt(mol)  
 else:  
 return np.nan # Xử lý các trường hợp phân tích cú pháp MolBlock không thành công  
 except Exception as e:  
 print(f"Lỗi khi xử lý MolBlock: {e}")  
 return np.nan  
  
# Áp dụng hàm cho dataframe  
df['molecular\_weight'] = df['molfile'].apply(calculate\_molecular\_weight)  
  
# Hiển thị dataframe với trọng lượng phân tử  
print(df.head())  
  
# Xử lý các giá trị bị thiếu (NaN) nếu có  
df = df.dropna()  
  
print(f"Hình dạng của Dataframe sau khi loại bỏ các giá trị NaN: {df.shape}")  
  
# Ví dụ về xử lý thêm: Chuyển đổi standard\_value thành số và lấy -log10(IC50)  
df = df[df['standard\_type'] == 'IC50'] # Lọc chỉ các giá trị IC50 cho ví dụ này  
df['standard\_value'] = pd.to\_numeric(df['standard\_value'], errors='coerce') # Chuyển đổi thành số, xử lý lỗi  
df = df.dropna(subset=['standard\_value']) # Xóa các hàng có standard\_value NaN sau khi chuyển đổi  
df['pIC50'] = -np.log10(df['standard\_value'] \* 1e-9) # Chuyển đổi nM thành M và lấy -log10  
  
print(df.head())

**Giải thích (Python - Tải và Xử lý Dữ liệu):**

* **Nhập Thư viện:** Nhập các thư viện cần thiết: os, pandas, RDKit (Chem, Descriptors).
* **Xử lý Đường dẫn:** Sử dụng os.path.join để xây dựng đường dẫn chính xác đến tệp CSV của bạn. Điều này rất quan trọng để tái tạo.
* **Tải Dữ liệu:** Tải tệp CSV vào Pandas DataFrame bằng pd.read\_csv. Bao gồm xử lý lỗi cho trường hợp không tìm thấy tệp.
* **Tính Trọng lượng Phân tử:** Định nghĩa một hàm calculate\_molecular\_weight lấy một chuỗi MolBlock (từ cột molfile), phân tích cú pháp nó bằng RDKit và tính toán trọng lượng phân tử. Nó cũng bao gồm xử lý lỗi để đối phó với các MolBlock có khả năng không hợp lệ.
* **Áp dụng Hàm:** Áp dụng hàm calculate\_molecular\_weight cho cột molfile để tạo một cột mới có tên là molecular\_weight.
* **Xử lý Giá trị Thiếu:** Loại bỏ các hàng có giá trị bị thiếu (NaN) có thể phát sinh từ lỗi trong phân tích cú pháp MolBlock.
* **Ví dụ Xử lý (pIC50):** Thể hiện xử lý dữ liệu thêm:
  + Lọc DataFrame để chỉ bao gồm các hàng có standard\_type là ‘IC50’.
  + Chuyển đổi cột standard\_value thành số bằng pd.to\_numeric, xử lý các lỗi tiềm ẩn bằng cách đặt các giá trị không hợp lệ thành NaN.
  + Xóa các hàng có giá trị NaN trong cột standard\_value (do chuyển đổi).
  + Tính giá trị pIC50 từ các giá trị IC50 (nM) bằng công thức: pIC50 = -log10(IC50 \* 1e-9) (chuyển đổi nM thành M).

**c) Python (notebook/Topic\_CheMBL\_35\_89\_2\_feature\_generation.ipynb - Feature Generation)**

import os  
import pandas as pd  
from rdkit import Chem  
from rdkit.Chem import AllChem  
from rdkit.Chem import Descriptors  
  
# Define base path  
base\_path = ".."  
data\_path = os.path.join(base\_path, "data", "chembl\_35\_89\_data.csv")  
  
# Load the data  
try:  
 df = pd.read\_csv(data\_path)  
except FileNotFoundError:  
 print(f"Error: CSV file not found at {data\_path}.")  
 exit()  
  
# Function to generate Morgan fingerprints (ECFP4)  
def generate\_morgan\_fingerprint(molfile, radius=2, nBits=2048):  
 try:  
 mol = Chem.MolFromMolBlock(molfile)  
 if mol is not None:  
 fp = AllChem.GetMorganFingerprintAsBitVect(mol, radius, nBits=nBits)  
 return fp  
 else:  
 return None  
 except Exception as e:  
 print(f"Error generating fingerprint: {e}")  
 return None  
  
# Generate Morgan fingerprints  
df['morgan\_fp'] = df['molfile'].apply(generate\_morgan\_fingerprint)  
  
# Convert fingerprints to a list of integers (if fingerprint generation was successful)  
df['morgan\_fp\_list'] = df['morgan\_fp'].apply(lambda fp: list(fp) if fp is not None else None)  
  
# Drop rows where fingerprint generation failed  
df = df.dropna(subset=['morgan\_fp\_list'])  
  
# Expand the fingerprint list into separate columns  
fingerprint\_df = pd.DataFrame(df['morgan\_fp\_list'].tolist(), index=df.index)  
df = pd.concat([df, fingerprint\_df], axis=1)  
  
# Remove the original 'molfile' and intermediate fingerprint columns  
df = df.drop(columns=['molfile', 'morgan\_fp', 'morgan\_fp\_list'])  
  
print(df.head())  
print(f"Shape of the dataframe after feature generation: {df.shape}")

**Explanation (Python - Feature Generation):**

* **Import Libraries:** Imports necessary libraries, including AllChem for fingerprint generation.
* **Load Data:** Loads the data from the CSV file (you might want to load the processed data from the previous notebook, depending on your workflow).
* **generate\_morgan\_fingerprint function:**
  + Takes a MolBlock string as input.
  + Parses the MolBlock into an RDKit molecule object using Chem.MolFromMolBlock().
  + Generates a Morgan fingerprint (also known as ECFP4) using AllChem.GetMorganFingerprintAsBitVect(). The radius and nBits parameters control the fingerprint’s characteristics.
  + Returns the fingerprint object. Returns None if the molecule parsing fails.
* **Apply Fingerprint Generation:** Applies the generate\_morgan\_fingerprint function to the molfile column to create a new column called morgan\_fp.
* **Convert to List:** Converts the fingerprint object to a list of integers (0 or 1) for each bit. This makes it easier to use the fingerprint as features in machine learning models.
* **Expand into Columns:** The core part of turning the fingerprints into usable features:
  + Creates a new DataFrame (fingerprint\_df) from the list of fingerprints. Each fingerprint bit becomes a separate column in this DataFrame.
  + Concatenates the fingerprint\_df with the original DataFrame df.
* **Clean Up:** Removes the original molfile and intermediate fingerprint columns (morgan\_fp, morgan\_fp\_list) since they are no longer needed.

**Vietnamese (Python - Tạo Đặc trưng):**

import os  
import pandas as pd  
from rdkit import Chem  
from rdkit.Chem import AllChem  
from rdkit.Chem import Descriptors  
  
# Xác định đường dẫn cơ sở  
base\_path = ".."  
data\_path = os.path.join(base\_path, "data", "chembl\_35\_89\_data.csv")  
  
# Tải dữ liệu  
try:  
 df = pd.read\_csv(data\_path)  
except FileNotFoundError:  
 print(f"Lỗi: Không tìm thấy tệp CSV tại {data\_path}.")  
 exit()  
  
# Hàm tạo vân tay Morgan (ECFP4)  
def generate\_morgan\_fingerprint(molfile, radius=2, nBits=2048):  
 try:  
 mol = Chem.MolFromMolBlock(molfile)  
 if mol is not None:  
 fp = AllChem.GetMorganFingerprintAsBitVect(mol, radius, nBits=nBits)  
 return fp  
 else:  
 return None  
 except Exception as e:  
 print(f"Lỗi khi tạo vân tay: {e}")  
 return None  
  
# Tạo vân tay Morgan  
df['morgan\_fp'] = df['molfile'].apply(generate\_morgan\_fingerprint)  
  
# Chuyển đổi vân tay thành danh sách các số nguyên (nếu tạo vân tay thành công)  
df['morgan\_fp\_list'] = df['morgan\_fp'].apply(lambda fp: list(fp) if fp is not None else None)  
  
# Loại bỏ các hàng mà việc tạo vân tay không thành công  
df = df.dropna(subset=['morgan\_fp\_list'])  
  
# Mở rộng danh sách vân tay thành các cột riêng biệt  
fingerprint\_df = pd.DataFrame(df['morgan\_fp\_list'].tolist(), index=df.index)  
df = pd.concat([df, fingerprint\_df], axis=1)  
  
# Loại bỏ cột 'molfile' ban đầu và các cột vân tay trung gian  
df = df.drop(columns=['molfile', 'morgan\_fp', 'morgan\_fp\_list'])  
  
print(df.head())  
print(f"Hình dạng của dataframe sau khi tạo đặc trưng: {df.shape}")

**Giải thích (Python - Tạo Đặc trưng):**

* **Nhập Thư viện:** Nhập các thư viện cần thiết, bao gồm AllChem để tạo vân tay.
* **Tải Dữ liệu:** Tải dữ liệu từ tệp CSV (bạn có thể muốn tải dữ liệu đã xử lý từ notebook trước đó, tùy thuộc vào quy trình làm việc của bạn).
* **Hàm generate\_morgan\_fingerprint:**
  + Lấy một chuỗi MolBlock làm đầu vào.
  + Phân tích cú pháp MolBlock thành một đối tượng phân tử RDKit bằng cách sử dụng Chem.MolFromMolBlock().
  + Tạo vân tay Morgan (còn được gọi là ECFP4) bằng cách sử dụng AllChem.GetMorganFingerprintAsBitVect(). Các tham số radius và nBits kiểm soát các đặc điểm của vân tay.
  + Trả về đối tượng vân tay. Trả về None nếu việc phân tích cú pháp phân tử không thành công.
* **Áp dụng Tạo Vân tay:** Áp dụng hàm generate\_morgan\_fingerprint cho cột molfile để tạo một cột mới có tên là morgan\_fp.
* **Chuyển đổi thành Danh sách:** Chuyển đổi đối tượng vân tay thành một danh sách các số nguyên (0 hoặc 1) cho mỗi bit. Điều này giúp bạn dễ dàng sử dụng vân tay làm đặc trưng trong các mô hình học máy.
* **Mở rộng thành Cột:** Phần cốt lõi của việc biến vân tay thành các đặc trưng có thể sử dụng được:
  + Tạo một DataFrame mới (fingerprint\_df) từ danh sách vân tay. Mỗi bit vân tay trở thành mộtột riêng biệt trong DataFrame này.
  + Nối fingerprint\_df với DataFrame ban đầu df.
* **Dọn dẹp:** Loại bỏ cột molfile ban đầu và các cột vân tay trung gian (morgan\_fp, morgan\_fp\_list) vì chúng không còn cần thiết nữa.

**d) Python (notebook/Topic\_CheMBL\_35\_89\_3\_model\_building.ipynb - Model Building)**

import os  
import pandas as pd  
from sklearn.model\_selection import train\_test\_split  
from sklearn.ensemble import RandomForestRegressor  
from sklearn.metrics import mean\_squared\_error, r2\_score  
import joblib  
  
# Define base path  
base\_path = ".."  
data\_path = os.path.join(base\_path, "data", "chembl\_35\_89\_data.csv") # Or load the processed data file from previous notebook  
  
# Load the data  
try:  
 df = pd.read\_csv(data\_path) # Consider loading the processed data here if using pIC50 values  
except FileNotFoundError:  
 print(f"Error: CSV file not found at {data\_path}.")  
 exit()  
  
# Drop rows with missing values  
df = df.dropna()  
  
# Prepare the data  
X = df.iloc[:, 6:] # Features: Assuming fingerprint columns start from the 7th column onwards  
y = df['standard\_value'] # Target variable: IC50 values  
  
# Split the 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)  
  
# Initialize and train the Random Forest Regressor model  
model = RandomForestRegressor(n\_estimators=100, random\_state=42)  
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}')  
  
# Save the model  
model\_path = os.path.join(base\_path, "models", "random\_forest\_model.joblib")  
os.makedirs(os.path.dirname(model\_path), exist\_ok=True)  
joblib.dump(model, model\_path)  
  
print(f"Model saved to: {model\_path}")

**Explanation (Python - Model Building):**

* **Import Libraries:** Imports necessary libraries for model building, evaluation, and saving.
* **Load Data:** Loads the data from the CSV file (or the processed data from the previous step). **Important:** Make sure this data includes the features you generated (e.g., the fingerprint columns).
* **Prepare Data:**
  + Defines X (the features) and y (the target variable). **Crucially, adjust the column indexing for X to match the columns where your features (fingerprints) are located.** The example assumes they start from the 7th column onwards.
  + y is set to 'standard\_value' in this example. If you created and want to use pIC50, change this line to y = df['pIC50'].
* **Split Data:** Splits the data into training and testing sets using train\_test\_split.
* **Train Model:**
  + Initializes a RandomForestRegressor model. You can experiment with different models and hyperparameters.
  + Trains the model using the training data (model.fit).
* **Evaluate Model:**
  + Makes predictions on the test set (model.predict).
  + Calculates the Mean Squared Error (MSE) and R-squared (R2) to evaluate the model’s performance.
* **Save Model:** Saves the trained model to a file using joblib.dump. This allows you to load the model later without retraining it. The model is saved in a models directory.

**Vietnamese (Python - Xây dựng Mô hình):**

import os  
import pandas as pd  
from sklearn.model\_selection import train\_test\_split  
from sklearn.ensemble import RandomForestRegressor  
from sklearn.metrics import mean\_squared\_error, r2\_score  
import joblib  
  
# Xác định đường dẫn cơ sở  
base\_path = ".."  
data\_path = os.path.join(base\_path, "data", "chembl\_35\_89\_data.csv") # Hoặc tải tệp dữ liệu đã xử lý từ notebook trước đó  
  
# Tải dữ liệu  
try:  
 df = pd.read\_csv(data\_path) # Cân nhắc tải dữ liệu đã xử lý ở đây nếu sử dụng giá trị pIC50  
except FileNotFoundError:  
 print(f"Lỗi: Không tìm thấy tệp CSV tại {data\_path}.")  
 exit()  
  
# Loại bỏ các hàng có giá trị bị thiếu  
df = df.dropna()  
  
# Chuẩn bị dữ liệu  
X = df.iloc[:, 6:] # Đặc trưng: Giả sử các cột vân tay bắt đầu từ cột thứ 7 trở đi  
y = df['standard\_value'] # Biến mục tiêu: Giá trị IC50  
  
# Chia dữ liệu thành các tập huấn luyện và kiểm tra  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)  
  
# Khởi tạo và huấn luyện mô hình Random Forest Regressor  
model = RandomForestRegressor(n\_estimators=100, random\_state=42)  
model.fit(X\_train, y\_train)  
  
# Đưa ra dự đoán trên tập kiểm tra  
y\_pred = model.predict(X\_test)  
  
# Đánh giá mô hình  
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}')  
  
# Lưu mô hình  
model\_path = os.path.join(base\_path, "models", "random\_forest\_model.joblib")  
os.makedirs(os.path.dirname(model\_path), exist\_ok=True)  
joblib.dump(model, model\_path)  
  
print(f"Mô hình đã được lưu vào: {model\_path}")

**Giải thích (Python - Xây dựng Mô hình):**

* **Nhập Thư viện:** Nhập các thư viện cần thiết cho việc xây dựng, đánh giá và lưu mô hình.
* **Tải Dữ liệu:** Tải dữ liệu từ tệp CSV (hoặc dữ liệu đã xử lý từ bước trước). **Quan trọng:** Đảm bảo dữ liệu này bao gồm các đặc trưng bạn đã tạo (ví dụ: các cột vân tay).
* **Chuẩn bị Dữ liệu:**
  + Xác định X (các đặc trưng) và y (biến mục tiêu). **Quan trọng, điều chỉnh chỉ mục cột cho X để khớp với các cột nơi các đặc trưng (vân tay) của bạn được đặt.** Ví dụ giả định chúng bắt đầu từ cột thứ 7 trở đi.
  + y được đặt thành 'standard\_value' trong ví dụ này. Nếu bạn đã tạo và muốn sử dụng pIC50, hãy thay đổi dòng này thành y = df['pIC50'].
* **Chia Dữ liệu:** Chia dữ liệu thành các tập huấn luyện và kiểm tra bằng cách sử dụng train\_test\_split.
* **Huấn luyện Mô hình:**
  + Khởi tạo một mô hình RandomForestRegressor. Bạn có thể thử nghiệm với các mô hình và siêu tham số khác nhau.
  + Huấn luyện mô hình bằng cách sử dụng dữ liệu huấn luyện (model.fit).
* **Đánh giá Mô hình:**
  + Đưa ra dự đoán trên tập kiểm tra (model.predict).
  + Tính toán Mean Squared Error (MSE) và R-squared (R2) để đánh giá hiệu suất của mô hình.
* **Lưu Mô hình:** Lưu mô hình đã được huấn luyện vào một tệp bằng cách sử dụng joblib.dump. Điều này cho phép bạn tải mô hình sau này mà không cần huấn luyện lại. Mô hình được lưu trong thư mục models.

**3. Addressing the squared=False error:**

The error old scikit-learn version does not support parameters squared=False in the mean\_squared\_error function indicates that you are using an older version of scikit-learn. The squared=False parameter was introduced in a later version.

**Solution:**

1. **Update scikit-learn:** The best solution is to update your scikit-learn version. In your Jupyter notebook, run:

* !pip install --upgrade scikit-learn
* Restart your Jupyter kernel after upgrading.

1. **Alternative (if upgrading is not possible):** If you absolutely cannot upgrade, you can manually calculate the Root Mean Squared Error (RMSE) instead of relying on the squared=False parameter:

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

**4. Project Folder Structure**

Topic\_CheMBL\_35\_89/  
├── data/  
│ └── chembl\_35\_89\_data.csv (Output from SQL query)  
├── notebooks/  
│ ├── Topic\_CheMBL\_35\_89\_1\_data\_loading.ipynb  
│ ├── Topic\_CheMBL\_35\_89\_2\_feature\_generation.ipynb  
│ └── Topic\_CheMBL\_35\_89\_3\_model\_building.ipynb  
├── models/  
│ └── random\_forest\_model.joblib  
└── README.md (Optional: Project description and instructions)

**5. Five Examples of Using the Code (Example Use Cases)**

**Example 1: Basic Molecular Weight Calculation and Saving to a New CSV**

* **Goal:** Calculate molecular weight for the 100 compounds and save the data with the calculated molecular weights to a new CSV file.

# In notebook/Topic\_CheMBL\_35\_89\_1\_data\_loading.ipynb  
  
import os  
import pandas as pd  
from rdkit import Chem  
from rdkit.Chem import Descriptors  
import numpy as np  
  
# ... (Data loading and molecular weight calculation code from above) ...  
  
# Save the updated DataFrame to a new CSV file  
output\_path = os.path.join(base\_path, "data", "chembl\_35\_89\_data\_with\_mw.csv")  
df.to\_csv(output\_path, index=False) # index=False prevents writing the DataFrame index to the CSV  
print(f"Data with molecular weights saved to: {output\_path}")

**Vietnamese:**

* **Mục tiêu:** Tính trọng lượng phân tử cho 100 hợp chất và lưu dữ liệu cùng với trọng lượng phân tử đã tính vào một tệp CSV mới.

# Trong notebook/Topic\_CheMBL\_35\_89\_1\_data\_loading.ipynb  
  
import os  
import pandas as pd  
from rdkit import Chem  
from rdkit.Chem import Descriptors  
import numpy as np  
  
# ... (Mã tải dữ liệu và tính trọng lượng phân tử từ trên) ...  
  
# Lưu DataFrame đã cập nhật vào một tệp CSV mới  
output\_path = os.path.join(base\_path, "data", "chembl\_35\_89\_data\_with\_mw.csv")  
df.to\_csv(output\_path, index=False) # index=False ngăn không cho ghi chỉ mục DataFrame vào CSV  
print(f"Dữ liệu có trọng lượng phân tử đã được lưu vào: {output\_path}")

**Example 2: Generating and Visualizing a Single Molecular Fingerprint**

* **Goal:** Generate a Morgan fingerprint for the first molecule in your dataset and visualize its bit representation.

# In notebook/Topic\_CheMBL\_35\_89\_2\_feature\_generation.ipynb  
  
import os  
import pandas as pd  
from rdkit import Chem  
from rdkit.Chem import AllChem  
from rdkit.Chem.Draw import DrawMorganBit  
  
# ... (Data loading and fingerprint generation code from above) ...  
  
# Get the first molecule and its Morgan fingerprint  
first\_molfile = df['molfile'].iloc[0]  
mol = Chem.MolFromMolBlock(first\_molfile)  
fp = AllChem.GetMorganFingerprint(mol, 2) #Different for visualization  
  
# Find the most significant bit(s)  
info = {}  
bits = fp.GetNonzeroElements()  
print(bits)  
  
#Draw Significant bits  
for bit in bits:  
 img = DrawMorganBit(mol,bit,info)  
 img.save(f"bit\_{bit}.png") #save each bit

**Vietnamese:**

* **Mục tiêu:** Tạo vân tay Morgan cho phân tử đầu tiên trong tập dữ liệu của bạn và trực quan hóa biểu diễn bit của nó.

# Trong notebook/Topic\_CheMBL\_35\_89\_2\_feature\_generation.ipynb  
  
import os  
import pandas as pd  
from rdkit import Chem  
from rdkit.Chem import AllChem  
from rdkit.Chem.Draw import DrawMorganBit  
  
# ... (Mã tải dữ liệu và tạo vân tay từ trên) ...  
  
# Lấy phân tử đầu tiên và vân tay Morgan của nó  
first\_molfile = df['molfile'].iloc[0]  
mol = Chem.MolFromMolBlock(first\_molfile)  
fp = AllChem.GetMorganFingerprint(mol, 2) #Khác nhau để trực quan hóa  
  
# Tìm (các) bit quan trọng nhất  
info = {}  
bits = fp.GetNonzeroElements()  
print(bits)  
  
#Vẽ các bit quan trọng  
for bit in bits:  
 img = DrawMorganBit(mol,bit,info)  
 img.save(f"bit\_{bit}.png") #lưu từng bit

**Example 3: Filtering by Activity Value**

* **Goal:** Filter the DataFrame to only include compounds with a standard value less than 100 nM.

# In notebook/Topic\_CheMBL\_35\_89\_1\_data\_loading.ipynb (or a new notebook)  
  
import os  
import pandas as pd  
  
# ... (Data loading code from above) ...  
  
# Filter for compounds with standard\_value < 100 nM  
df\_filtered = df[df['standard\_value'] < 100]  
print(f"Shape of filtered DataFrame: {df\_filtered.shape}")  
print(df\_filtered.head())

**Vietnamese:**

* **Mục tiêu:** Lọc DataFrame để chỉ bao gồm các hợp chất có giá trị tiêu chuẩn nhỏ hơn 100 nM.

# Trong notebook/Topic\_CheMBL\_35\_89\_1\_data\_loading.ipynb (hoặc một notebook mới)  
  
import os  
import pandas as pd  
  
# ... (Mã tải dữ liệu từ trên) ...  
  
# Lọc các hợp chất có standard\_value < 100 nM  
df\_filtered = df[df['standard\_value'] < 100]  
print(f"Hình dạng của DataFrame đã lọc: {df\_filtered.shape}")  
print(df\_filtered.head())

**Example 4: Train a model with pIC50 values and evaluate with RMSE**

* **Goal:** Calculate the pIC50 values, train and evaluate the mode, report the RMSE

# In notebook/Topic\_CheMBL\_35\_89\_3\_model\_building.ipynb  
  
import os  
import pandas as pd  
from sklearn.model\_selection import train\_test\_split  
from sklearn.ensemble import RandomForestRegressor  
from sklearn.metrics import mean\_squared\_error, r2\_score  
import joblib  
import numpy as np  
  
# Define base path  
base\_path = ".."  
data\_path = os.path.join(base\_path, "data", "chembl\_35\_89\_data.csv") # Or load the processed data file from previous notebook  
  
# Load the data  
try:  
 df = pd.read\_csv(data\_path) # Consider loading the processed data here if using pIC50 values  
except FileNotFoundError:  
 print(f"Error: CSV file not found at {data\_path}.")  
 exit()  
  
# Drop rows with missing values  
df = df.dropna()  
  
# Prepare the data  
X = df.iloc[:, 6:] # Features: Assuming fingerprint columns start from the 7th column onwards  
y = df['pIC50'] # Target variable: pIC50 values  
  
# Split the 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)  
  
# Initialize and train the Random Forest Regressor model  
model = RandomForestRegressor(n\_estimators=100, random\_state=42)  
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)  
rmse = np.sqrt(mse) #Calculate RMSE  
r2 = r2\_score(y\_test, y\_pred)  
  
print(f'Root Mean Squared Error: {rmse}')  
print(f'R-squared: {r2}')  
  
# Save the model  
model\_path = os.path.join(base\_path, "models", "random\_forest\_model.joblib")  
os.makedirs(os.path.dirname(model\_path), exist\_ok=True)  
joblib.dump(model, model\_path)  
  
print(f"Model saved to: {model\_path}")

**Vietnamese:**

* **Mục tiêu:** Tính giá trị pIC50, huấn luyện và đánh giá mô hình, báo cáo RMSE

# Trong notebook/Topic\_CheMBL\_35\_89\_3\_model\_building.ipynb  
  
import os  
import pandas as pd  
from sklearn.model\_selection import train\_test\_split  
from sklearn.ensemble import RandomForestRegressor  
from sklearn.metrics import mean\_squared\_error, r2\_score  
import joblib  
import numpy as np  
  
# Xác định đường dẫn cơ sở  
base\_path = ".."  
data\_path = os.path.join(base\_path, "data", "chembl\_35\_89\_data.csv") # Hoặc tải tệp dữ liệu đã xử lý từ notebook trước đó  
  
# Tải dữ liệu  
try:  
 df = pd.read\_csv(data\_path) # Cân nhắc tải dữ liệu đã xử lý ở đây nếu sử dụng giá trị pIC50  
except FileNotFoundError:  
 print(f"Lỗi: Không tìm thấy tệp CSV tại {data\_path}.")  
 exit()  
  
# Loại bỏ các hàng có giá trị bị thiếu  
df = df.dropna()  
  
# Chuẩn bị dữ liệu  
X = df.iloc[:, 6:] # Đặc trưng: Giả sử các cột vân tay bắt đầu từ cột thứ 7 trở đi  
y = df['pIC50'] # Biến mục tiêu: Giá trị pIC50  
  
# Chia dữ liệu thành các tập huấn luyện và kiểm tra  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)  
  
# Khởi tạo và huấn luyện mô hình Random Forest Regressor  
model = RandomForestRegressor(n\_estimators=100, random\_state=42)  
model.fit(X\_train, y\_train)  
  
# Đưa ra dự đoán trên tập kiểm tra  
y\_pred = model.predict(X\_test)  
  
# Đánh giá mô hình  
mse = mean\_squared\_error(y\_test, y\_pred)  
rmse = np.sqrt(mse) # Tính RMSE  
r2 = r2\_score(y\_test, y\_pred)  
  
print(f'Root Mean Squared Error: {rmse}')  
print(f'R-squared: {r2}')  
  
# Lưu mô hình  
model\_path = os.path.join(base\_path, "models", "random\_forest\_model.joblib")  
os.makedirs(os.path.dirname(model\_path), exist\_ok=True)  
joblib.dump(model, model\_path)  
  
print(f"Mô hình đã được lưu vào: {model\_path}")

**Example 5: Loading a Saved Model and Making Predictions on New Data**

* **Goal:** Load the saved Random Forest model and use it to predict activity values for a small set of new molecules (assuming you have their MolBlocks).

# In a new notebook (or the model building notebook)  
  
import os  
import pandas as pd  
import joblib  
from rdkit import Chem  
from rdkit.Chem import AllChem  
import numpy as np  
  
# Define base path  
base\_path = ".."  
model\_path = os.path.join(base\_path, "models", "random\_forest\_model.joblib")  
  
# Load the model  
try:  
 model = joblib.load(model\_path)  
 print("Model loaded successfully.")  
except FileNotFoundError:  
 print(f"Error: Model file not found at {model\_path}")  
 exit()  
  
# Sample new data (replace with your actual data)  
new\_data = {  
 'molfile': [  
 """  
 MolBlock for molecule 1...  
 """,  
 """  
 MolBlock for molecule 2...  
 """,  
 ]  
}  
new\_df = pd.DataFrame(new\_data)  
  
def generate\_morgan\_fingerprint(molfile, radius=2, nBits=2048):  
 try:  
 mol = Chem.MolFromMolBlock(molfile)  
 if mol is not None:  
 fp = AllChem.GetMorganFingerprintAsBitVect(mol, radius, nBits=nBits)  
 return fp  
 else:  
 return None  
 except Exception as e:  
 print(f"Error generating fingerprint: {e}")  
 return None  
  
# Generate Morgan fingerprints  
new\_df['morgan\_fp'] = new\_df['molfile'].apply(generate\_morgan\_fingerprint)  
  
# Convert fingerprints to a list of integers (if fingerprint generation was successful)  
new\_df['morgan\_fp\_list'] = new\_df['morgan\_fp'].apply(lambda fp: list(fp) if fp is not None else None)  
  
# Drop rows where fingerprint generation failed  
new\_df = new\_df.dropna(subset=['morgan\_fp\_list'])  
  
# Expand the fingerprint list into separate columns  
fingerprint\_df = pd.DataFrame(new\_df['morgan\_fp\_list'].tolist(), index=new\_df.index)  
new\_df = pd.concat([new\_df, fingerprint\_df], axis=1)  
  
# Prepare the data  
X\_new = new\_df.iloc[:, 2:] # Features: Adjust indexing based on where features start  
  
# Make predictions  
predictions = model.predict(X\_new)  
  
print("Predictions:")  
for i, pred in enumerate(predictions):  
 print(f"Molecule {i+1}: Predicted activity = {pred}")

**Vietnamese:**

* **Mục tiêu:** Tải mô hình Random Forest đã lưu và sử dụng nó để dự đoán giá trị hoạt động cho một tập hợp nhỏ các phân tử mới (giả sử bạn có MolBlock của chúng).

# Trong một notebook mới (hoặc notebook xây dựng mô hình)  
  
import os  
import pandas as pd  
import joblib  
from rdkit import Chem  
from rdkit.Chem import AllChem  
import numpy as np  
  
# Xác định đường dẫn cơ sở  
base\_path = ".."  
model\_path = os.path.join(base\_path, "models", "random\_forest\_model.joblib")  
  
# Tải mô hình  
try:  
 model = joblib.load(model\_path)  
 print("Mô hình đã được tải thành công.")  
except FileNotFoundError:  
 print(f"Lỗi: Không tìm thấy tệp mô hình tại {model\_path}")  
 exit()  
  
# Dữ liệu mới mẫu (thay thế bằng dữ liệu thực tế của bạn)  
new\_data = {  
 'molfile': [  
 """  
 MolBlock cho phân tử 1...  
 """,  
 """  
 MolBlock cho phân tử 2...  
 """,  
 ]  
}  
new\_df = pd.DataFrame(new\_data)  
  
def generate\_morgan\_fingerprint(molfile, radius=2, nBits=2048):  
 try:  
 mol = Chem.MolFromMolBlock(molfile)  
 if mol is not None:  
 fp = AllChem.GetMorganFingerprintAsBitVect(mol, radius, nBits=nBits)  
 return fp  
 else:  
 return None  
 except Exception as e:  
 print(f"Lỗi khi tạo vân tay: {e}")  
 return None  
  
# Tạo vân tay Morgan  
new\_df['morgan\_fp'] = new\_df['molfile'].apply(generate\_morgan\_fingerprint)  
  
# Chuyển đổi vân tay thành danh sách các số nguyên (nếu tạo vân tay thành công)  
new\_df['morgan\_fp\_list'] = new\_df['morgan\_fp'].apply(lambda fp: list(fp) if fp is not None else None)  
  
# Loại bỏ các hàng mà việc tạo vân tay không thành công  
new\_df = new\_df.dropna(subset=['morgan\_fp\_list'])  
  
# Mở rộng danh sách vân tay thành các cột riêng biệt  
fingerprint\_df = pd.DataFrame(new\_df['morgan\_fp\_list'].tolist(), index=new\_df.index)  
new\_df = pd.concat([new\_df, fingerprint\_df], axis=1)  
  
  
# Chuẩn bị dữ liệu  
X\_new = new\_df.iloc[:, 2:] # Đặc trưng: Điều chỉnh chỉ mục dựa trên vị trí bắt đầu của đặc trưng  
  
# Đưa ra dự đoán  
predictions = model.predict(X\_new)  
  
print("Dự đoán:")  
for i, pred in enumerate(predictions):  
 print(f"Phân tử {i+1}: Hoạt động dự đoán = {pred}")

**Key Considerations & Best Practices:**

* **Error Handling:** I’ve included basic error handling (e.g., try...except blocks for file loading and MolBlock parsing). You should expand this to handle other potential issues.
* **Data Validation:** Thoroughly validate your data after loading it. Check for inconsistencies, outliers, and unexpected values.
* **Feature Selection:** Consider using feature selection techniques to identify the most relevant features for your model. This can improve performance and reduce overfitting.
* **Hyperparameter Tuning:** Experiment with different hyperparameters for your machine learning models to optimize their performance. Techniques like grid search or randomized search can be helpful.
* **Cross-Validation:** Use cross-validation to get a more robust estimate of your model’s performance.
* **Logging:** Implement logging to track the progress of your analysis and to record any errors or warnings.
* **Reproducibility:** Use a consistent project structure, version control (e.g., Git), and a virtual environment to ensure that your analysis is reproducible.
* **Workflow Management:** Consider using a workflow management system (e.g., Airflow, Prefect) to automate your data analysis pipeline.

Remember to adapt the code to your specific needs and experiment with different models, features, and parameters. Good luck with your project!