# PharmApp Suite

## 🧠 AI for Drug Discovery and Development 🧪

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

## 🧩 Topic: Hyperparameter tuning with Optuna for drug models

Okay, I understand. I’m acting as your Pharmaceutical R&D Specialist, assisting you with your ChEMBL 35 data analysis project. You’re using PostgreSQL, RDKit, and Jupyter notebooks following the AIMLops template, and you need guidance on data extraction, analysis, and code implementation, specifically addressing the errors you’ve encountered.

Here’s a detailed breakdown, guidance, and code examples in both English and Indonesian:

**1. Analysis Model Overview**

The overall analysis model will likely involve these steps:

* **Data Extraction:** Extract relevant data from the ChEMBL 35 database using SQL queries. This involves filtering for specific activities, molecules, and properties of interest. You’ll save this data into CSV files.
* **Data Preprocessing:** Load the CSV data into pandas DataFrames within your Jupyter notebook. Clean and preprocess the data, handling missing values, converting data types, and preparing features for analysis. This will likely involve RDKit for molecular feature generation.
* **Feature Engineering (RDKit):** Utilize RDKit to calculate relevant molecular descriptors (e.g., molecular weight, logP, number of hydrogen bond donors/acceptors, topological polar surface area). These descriptors will serve as features for your analysis.
* **Activity Filtering:** Filter out activity records with standard values not representing numbers.
* **Exploratory Data Analysis (EDA):** Perform EDA to visualize data distributions, identify outliers, and uncover relationships between molecular descriptors and activity values.
* **Modeling (Example: Regression):** Build a regression model (e.g., linear regression, random forest) to predict activity values based on the calculated molecular descriptors.
* **Model Evaluation:** Evaluate the performance of the model using appropriate metrics (e.g., Mean Squared Error - MSE, R-squared).
* **Interpretation:** Interpret the results of the model to gain insights into the relationship between molecular structure and activity.

**Analisis Model Overview (Indonesian)**

Model analisis secara keseluruhan akan melibatkan langkah-langkah berikut:

* **Ekstraksi Data:** Ekstrak data yang relevan dari database ChEMBL 35 menggunakan query SQL. Ini melibatkan penyaringan aktivitas, molekul, dan properti tertentu yang menarik. Anda akan menyimpan data ini ke dalam file CSV.
* **Pra-pemrosesan Data:** Muat data CSV ke dalam pandas DataFrames di dalam Jupyter notebook Anda. Bersihkan dan pra-proses data, tangani nilai yang hilang, konversi tipe data, dan siapkan fitur untuk analisis. Ini kemungkinan akan melibatkan RDKit untuk pembuatan fitur molekuler.
* **Rekayasa Fitur (RDKit):** Manfaatkan RDKit untuk menghitung deskriptor molekuler yang relevan (misalnya, berat molekul, logP, jumlah donor/akseptor ikatan hidrogen, area permukaan polar topologi). Deskriptor ini akan berfungsi sebagai fitur untuk analisis Anda.
* **Penyaringan Aktivitas:** Saring catatan aktivitas dengan nilai standar yang tidak mewakili angka.
* **Analisis Data Eksplorasi (EDA):** Lakukan EDA untuk memvisualisasikan distribusi data, mengidentifikasi outlier, dan mengungkap hubungan antara deskriptor molekuler dan nilai aktivitas.
* **Pemodelan (Contoh: Regresi):** Bangun model regresi (misalnya, regresi linier, random forest) untuk memprediksi nilai aktivitas berdasarkan deskriptor molekuler yang dihitung.
* **Evaluasi Model:** Evaluasi kinerja model menggunakan metrik yang sesuai (misalnya, Mean Squared Error - MSE, R-squared).
* **Interpretasi:** Interpretasikan hasil model untuk mendapatkan wawasan tentang hubungan antara struktur molekul dan aktivitas.

**2. Addressing Errors and Code**

**Error A: ERROR: operator does not exist: numeric ~ unknown, LINE 12: AND act.standard\_value ~ '^[0-9\.]+$'**

This error indicates that PostgreSQL is having trouble applying the regular expression ^[0-9\.]+$ to a numeric column (likely act.standard\_value). The ~ operator in PostgreSQL is used for regular expression matching on text data, not numeric data.

**Solution:**

Instead of using regular expressions, you should directly cast the column to text and then use the regular expression or, more efficiently, filter out rows where standard\_value is not a number using IS NOT NULL and ensuring the value is greater than or equal to zero if required. Since we are retrieving the top 100, the impact of doing this is minimal, and will prevent errors. This SQL has been updated below to reflect this.

**Error B: old scikit-learn version does not support parameters squared=False in the mean\_squared\_error function**

This error means that your scikit-learn version is older than 0.20, where the squared parameter was introduced in the mean\_squared\_error function.

**Solution:**

1. **Upgrade Scikit-learn:** The best solution is to upgrade your scikit-learn library: bash pip install -U scikit-learn
2. **Conditional Code (if upgrade is not possible):** If you absolutely cannot upgrade scikit-learn (not recommended), you can write conditional code:

* from sklearn.metrics import mean\_squared\_error  
  import sklearn  
    
  if sklearn.\_\_version\_\_ >= '0.20':  
   mse = mean\_squared\_error(y\_true, y\_pred, squared=False)  
  else:  
   mse = mean\_squared\_error(y\_true, y\_pred)\*\*0.5 # Calculate RMSE manually

**3. SQL Code (chembl\_35\_66.sql)**

-- Select 100 rows from the activities table and related tables  
-- Filter by standard\_type = 'IC50' and ensure standard\_value is numeric.  
-- Use a safe numeric check instead of regular expressions.  
  
SELECT  
 mol.chembl\_id,  
 act.standard\_value,  
 act.standard\_units,  
 act.standard\_type,  
 cmp.structure\_text AS smiles  
FROM  
 activities act  
JOIN  
 molecule\_dictionary mol ON act.molregno = mol.molregno  
JOIN  
 compound\_structures cmp ON mol.molregno = cmp.molregno  
WHERE  
 act.standard\_type = 'IC50'  
 AND act.standard\_value IS NOT NULL -- Ensure standard\_value is not null  
 AND act.standard\_units = 'nM'  
ORDER BY  
 act.standard\_value ASC --order to grab the lowest values first.  
LIMIT 100;  
  
-- Save the results to a CSV file (../data/chembl\_35\_66.csv) using pgAdmin or psql.

**SQL Code (Indonesian)**

-- Pilih 100 baris dari tabel activities dan tabel terkait  
-- Filter berdasarkan standard\_type = 'IC50' dan pastikan standard\_value adalah numerik.  
-- Gunakan pemeriksaan numerik yang aman alih-alih ekspresi reguler.  
  
SELECT  
 mol.chembl\_id,  
 act.standard\_value,  
 act.standard\_units,  
 act.standard\_type,  
 cmp.structure\_text AS smiles  
FROM  
 activities act  
JOIN  
 molecule\_dictionary mol ON act.molregno = mol.molregno  
JOIN  
 compound\_structures cmp ON mol.molregno = cmp.molregno  
WHERE  
 act.standard\_type = 'IC50'  
 AND act.standard\_value IS NOT NULL -- Pastikan standard\_value tidak null  
 AND act.standard\_units = 'nM'  
ORDER BY  
 act.standard\_value ASC --Urutkan untuk mengambil nilai terendah terlebih dahulu.  
LIMIT 100;  
  
-- Simpan hasil ke file CSV (../data/chembl\_35\_66.csv) menggunakan pgAdmin atau psql.

**Important:** Execute this SQL query in pgAdmin and save the result as ../data/chembl\_35\_66.csv. Make sure the path ../data exists.

**4. Python Code (Topic\_CheMBL\_35\_66\_1\_Data\_Loading\_and\_Preprocessing.ipynb)**

import os  
import pandas as pd  
from rdkit import Chem  
from rdkit.Chem import Descriptors  
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  
import sklearn  
  
# Define base path  
base\_path = ".." # Assuming your notebook is in the 'notebooks' directory  
  
# Load data from CSV  
data\_path = os.path.join(base\_path, "data", "chembl\_35\_66.csv")  
df = pd.read\_csv(data\_path)  
  
# Data Cleaning and Preprocessing  
df = df.dropna(subset=['standard\_value', 'smiles']) # Remove rows with missing values in essential columns  
  
# Convert standard\_value to numeric (explicitly)  
df['standard\_value'] = pd.to\_numeric(df['standard\_value'], errors='coerce')  
df = df.dropna(subset=['standard\_value']) #Remove rows that cannot be converted to numeric  
  
#Handle duplicate Smiles. Keep the one with the lowest IC50 value  
df = df.sort\_values('standard\_value').drop\_duplicates(subset='smiles', keep='first')  
  
# RDKit Feature Generation  
def calculate\_descriptors(smiles):  
 mol = Chem.MolFromSmiles(smiles)  
 if mol is None:  
 return None # Handle invalid SMILES  
 descriptors = {}  
 descriptors['MolWt'] = Descriptors.MolWt(mol)  
 descriptors['LogP'] = Descriptors.MolLogP(mol)  
 descriptors['HBD'] = Descriptors.NumHDonors(mol)  
 descriptors['HBA'] = Descriptors.NumHAcceptors(mol)  
 descriptors['TPSA'] = Descriptors.TPSA(mol)  
 return descriptors  
  
# Apply descriptor calculation to each molecule  
df['descriptors'] = df['smiles'].apply(calculate\_descriptors)  
  
# Remove rows where descriptor calculation failed  
df = df.dropna(subset=['descriptors'])  
  
# Expand the descriptors column into individual columns  
df = pd.concat([df.drop(['descriptors'], axis=1), df['descriptors'].apply(pd.Series)], axis=1)  
  
# Display the first few rows of the processed data  
print(df.head())  
  
# Prepare data for modeling  
X = df[['MolWt', 'LogP', 'HBD', 'HBA', 'TPSA']] # Features  
y = np.log10(df['standard\_value']) # Target (log transformed IC50) - useful for normalizing data  
# Split data into training and testing sets  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)  
  
# Model Training (Linear Regression)  
model = LinearRegression()  
model.fit(X\_train, y\_train)  
  
# Model Prediction  
y\_pred = model.predict(X\_test)  
  
# Model Evaluation  
mse = mean\_squared\_error(y\_test, y\_pred) # Calculate MSE. This will work even in older sklearn versions  
r2 = r2\_score(y\_test, y\_pred)  
  
print(f"Mean Squared Error: {mse}")  
print(f"R-squared: {r2}")

**Python Code (Indonesian)**

import os  
import pandas as pd  
from rdkit import Chem  
from rdkit.Chem import Descriptors  
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  
import sklearn  
  
# Tentukan base path  
base\_path = ".." # Asumsi notebook Anda berada di direktori 'notebooks'  
  
# Muat data dari CSV  
data\_path = os.path.join(base\_path, "data", "chembl\_35\_66.csv")  
df = pd.read\_csv(data\_path)  
  
# Pembersihan dan Pra-pemrosesan Data  
df = df.dropna(subset=['standard\_value', 'smiles']) # Hapus baris dengan nilai yang hilang di kolom penting  
  
# Konversi standard\_value ke numerik (secara eksplisit)  
df['standard\_value'] = pd.to\_numeric(df['standard\_value'], errors='coerce')  
df = df.dropna(subset=['standard\_value']) #Hapus baris yang tidak dapat dikonversi ke numerik  
  
#Tangani duplikat Smiles. Simpan yang memiliki nilai IC50 terendah  
df = df.sort\_values('standard\_value').drop\_duplicates(subset='smiles', keep='first')  
  
# Pembuatan Fitur RDKit  
def calculate\_descriptors(smiles):  
 mol = Chem.MolFromSmiles(smiles)  
 if mol is None:  
 return None # Tangani SMILES yang tidak valid  
 descriptors = {}  
 descriptors['MolWt'] = Descriptors.MolWt(mol)  
 descriptors['LogP'] = Descriptors.MolLogP(mol)  
 descriptors['HBD'] = Descriptors.NumHDonors(mol)  
 descriptors['HBA'] = Descriptors.NumHAcceptors(mol)  
 descriptors['TPSA'] = Descriptors.TPSA(mol)  
 return descriptors  
  
# Terapkan perhitungan deskriptor ke setiap molekul  
df['descriptors'] = df['smiles'].apply(calculate\_descriptors)  
  
# Hapus baris di mana perhitungan deskriptor gagal  
df = df.dropna(subset=['descriptors'])  
  
# Perluas kolom deskriptor menjadi kolom individual  
df = pd.concat([df.drop(['descriptors'], axis=1), df['descriptors'].apply(pd.Series)], axis=1)  
  
# Tampilkan beberapa baris pertama dari data yang diproses  
print(df.head())  
  
# Siapkan data untuk pemodelan  
X = df[['MolWt', 'LogP', 'HBD', 'HBA', 'TPSA']] # Fitur  
y = np.log10(df['standard\_value']) # Target (IC50 yang ditransformasi log) - berguna untuk menormalkan data  
# Pisahkan data menjadi set pelatihan dan pengujian  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)  
  
# Pelatihan Model (Regresi Linier)  
model = LinearRegression()  
model.fit(X\_train, y\_train)  
  
# Prediksi Model  
y\_pred = model.predict(X\_test)  
  
# Evaluasi Model  
mse = mean\_squared\_error(y\_test, y\_pred) # Hitung MSE. Ini akan berfungsi bahkan dalam versi sklearn yang lebih lama  
r2 = r2\_score(y\_test, y\_pred)  
  
print(f"Mean Squared Error: {mse}")  
print(f"R-squared: {r2}")

**5. Example Analysis and Code (Additional Notebooks)**

Here are 5 additional example notebooks you can create, building upon the initial data loading and preprocessing:

* **Topic\_CheMBL\_35\_66\_2\_EDA\_and\_Visualization.ipynb:** Focuses on Exploratory Data Analysis.
* **Topic\_CheMBL\_35\_66\_3\_RandomForest\_Regression.ipynb:** Implements a Random Forest Regression model.
* **Topic\_CheMBL\_35\_66\_4\_Feature\_Importance.ipynb:** Analyzes feature importance from the Random Forest model.
* **Topic\_CheMBL\_35\_66\_5\_Model\_Evaluation\_and\_Comparison.ipynb:** Compares the performance of Linear Regression and Random Forest.
* **Topic\_CheMBL\_35\_66\_6\_Outlier\_Analysis.ipynb:** Explores outliers in the data and their impact on the model.

I’ll provide code snippets for each of these examples. Remember to run the first notebook (Topic\_CheMBL\_35\_66\_1\_Data\_Loading\_and\_Preprocessing.ipynb) *first* and ensure the df DataFrame is correctly created before running the other notebooks.

**Example 1: Topic\_CheMBL\_35\_66\_2\_EDA\_and\_Visualization.ipynb**

#EDA and Visualization  
  
import matplotlib.pyplot as plt  
import seaborn as sns  
import os  
import pandas as pd  
import numpy as np  
  
# Load data from CSV (assuming the first notebook has been run and the dataframe exists)  
base\_path = ".."  
data\_path = os.path.join(base\_path, "data", "chembl\_35\_66.csv")  
df = pd.read\_csv(data\_path)  
  
# Data Cleaning and Preprocessing  
df = df.dropna(subset=['standard\_value', 'smiles']) # Remove rows with missing values in essential columns  
  
# Convert standard\_value to numeric (explicitly)  
df['standard\_value'] = pd.to\_numeric(df['standard\_value'], errors='coerce')  
df = df.dropna(subset=['standard\_value']) #Remove rows that cannot be converted to numeric  
  
#Handle duplicate Smiles. Keep the one with the lowest IC50 value  
df = df.sort\_values('standard\_value').drop\_duplicates(subset='smiles', keep='first')  
  
  
# RDKit Feature Generation  
from rdkit import Chem  
from rdkit.Chem import Descriptors  
def calculate\_descriptors(smiles):  
 mol = Chem.MolFromSmiles(smiles)  
 if mol is None:  
 return None # Handle invalid SMILES  
 descriptors = {}  
 descriptors['MolWt'] = Descriptors.MolWt(mol)  
 descriptors['LogP'] = Descriptors.MolLogP(mol)  
 descriptors['HBD'] = Descriptors.NumHDonors(mol)  
 descriptors['HBA'] = Descriptors.NumHAcceptors(mol)  
 descriptors['TPSA'] = Descriptors.TPSA(mol)  
 return descriptors  
  
# Apply descriptor calculation to each molecule  
df['descriptors'] = df['smiles'].apply(calculate\_descriptors)  
  
# Remove rows where descriptor calculation failed  
df = df.dropna(subset=['descriptors'])  
  
# Expand the descriptors column into individual columns  
df = pd.concat([df.drop(['descriptors'], axis=1), df['descriptors'].apply(pd.Series)], axis=1)  
  
  
  
# Distribution of IC50 values  
plt.figure(figsize=(10, 6))  
sns.histplot(np.log10(df['standard\_value']), kde=True)  
plt.title('Distribution of Log10(IC50)')  
plt.xlabel('Log10(IC50)')  
plt.ylabel('Frequency')  
plt.show()  
  
# Scatter plots of descriptors vs. IC50  
descriptors = ['MolWt', 'LogP', 'HBD', 'HBA', 'TPSA']  
for descriptor in descriptors:  
 plt.figure(figsize=(8, 6))  
 sns.scatterplot(x=df[descriptor], y=np.log10(df['standard\_value']))  
 plt.title(f'{descriptor} vs. Log10(IC50)')  
 plt.xlabel(descriptor)  
 plt.ylabel('Log10(IC50)')  
 plt.show()  
  
# Correlation heatmap  
correlation\_matrix = df[descriptors + ['standard\_value']].corr()  
plt.figure(figsize=(10, 8))  
sns.heatmap(correlation\_matrix, annot=True, cmap='coolwarm')  
plt.title('Correlation Heatmap')  
plt.show()

**Example 2: Topic\_CheMBL\_35\_66\_3\_RandomForest\_Regression.ipynb**

#RandomForest Regression Model  
import os  
import pandas as pd  
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  
import matplotlib.pyplot as plt  
  
  
# Load data from CSV  
base\_path = ".."  
data\_path = os.path.join(base\_path, "data", "chembl\_35\_66.csv")  
df = pd.read\_csv(data\_path)  
  
# Data Cleaning and Preprocessing  
df = df.dropna(subset=['standard\_value', 'smiles']) # Remove rows with missing values in essential columns  
  
# Convert standard\_value to numeric (explicitly)  
df['standard\_value'] = pd.to\_numeric(df['standard\_value'], errors='coerce')  
df = df.dropna(subset=['standard\_value']) #Remove rows that cannot be converted to numeric  
  
#Handle duplicate Smiles. Keep the one with the lowest IC50 value  
df = df.sort\_values('standard\_value').drop\_duplicates(subset='smiles', keep='first')  
  
  
# RDKit Feature Generation  
from rdkit import Chem  
from rdkit.Chem import Descriptors  
def calculate\_descriptors(smiles):  
 mol = Chem.MolFromSmiles(smiles)  
 if mol is None:  
 return None # Handle invalid SMILES  
 descriptors = {}  
 descriptors['MolWt'] = Descriptors.MolWt(mol)  
 descriptors['LogP'] = Descriptors.MolLogP(mol)  
 descriptors['HBD'] = Descriptors.NumHDonors(mol)  
 descriptors['HBA'] = Descriptors.NumHAcceptors(mol)  
 descriptors['TPSA'] = Descriptors.TPSA(mol)  
 return descriptors  
  
# Apply descriptor calculation to each molecule  
df['descriptors'] = df['smiles'].apply(calculate\_descriptors)  
  
# Remove rows where descriptor calculation failed  
df = df.dropna(subset=['descriptors'])  
  
# Expand the descriptors column into individual columns  
df = pd.concat([df.drop(['descriptors'], axis=1), df['descriptors'].apply(pd.Series)], axis=1)  
  
  
# Prepare data for modeling  
X = df[['MolWt', 'LogP', 'HBD', 'HBA', 'TPSA']]  
y = np.log10(df['standard\_value'])  
  
# Split data into training and testing sets  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)  
  
# Model Training (Random Forest Regression)  
rf\_model = RandomForestRegressor(n\_estimators=100, random\_state=42) # Adjust hyperparameters as needed  
rf\_model.fit(X\_train, y\_train)  
  
# Model Prediction  
y\_pred = rf\_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}")  
  
# Scatter plot of predicted vs. actual values  
plt.figure(figsize=(8, 6))  
plt.scatter(y\_test, y\_pred)  
plt.xlabel("Actual Log10(IC50)")  
plt.ylabel("Predicted Log10(IC50)")  
plt.title("Actual vs. Predicted Log10(IC50) (Random Forest)")  
plt.show()

**Example 3: Topic\_CheMBL\_35\_66\_4\_Feature\_Importance.ipynb**

#Feature Importance  
import os  
import pandas as pd  
import numpy as np  
from sklearn.model\_selection import train\_test\_split  
from sklearn.ensemble import RandomForestRegressor  
import matplotlib.pyplot as plt  
  
  
# Load data from CSV  
base\_path = ".."  
data\_path = os.path.join(base\_path, "data", "chembl\_35\_66.csv")  
df = pd.read\_csv(data\_path)  
  
# Data Cleaning and Preprocessing  
df = df.dropna(subset=['standard\_value', 'smiles']) # Remove rows with missing values in essential columns  
  
# Convert standard\_value to numeric (explicitly)  
df['standard\_value'] = pd.to\_numeric(df['standard\_value'], errors='coerce')  
df = df.dropna(subset=['standard\_value']) #Remove rows that cannot be converted to numeric  
  
#Handle duplicate Smiles. Keep the one with the lowest IC50 value  
df = df.sort\_values('standard\_value').drop\_duplicates(subset='smiles', keep='first')  
  
  
# RDKit Feature Generation  
from rdkit import Chem  
from rdkit.Chem import Descriptors  
def calculate\_descriptors(smiles):  
 mol = Chem.MolFromSmiles(smiles)  
 if mol is None:  
 return None # Handle invalid SMILES  
 descriptors = {}  
 descriptors['MolWt'] = Descriptors.MolWt(mol)  
 descriptors['LogP'] = Descriptors.MolLogP(mol)  
 descriptors['HBD'] = Descriptors.NumHDonors(mol)  
 descriptors['HBA'] = Descriptors.NumHAcceptors(mol)  
 descriptors['TPSA'] = Descriptors.TPSA(mol)  
 return descriptors  
  
# Apply descriptor calculation to each molecule  
df['descriptors'] = df['smiles'].apply(calculate\_descriptors)  
  
# Remove rows where descriptor calculation failed  
df = df.dropna(subset=['descriptors'])  
  
# Expand the descriptors column into individual columns  
df = pd.concat([df.drop(['descriptors'], axis=1), df['descriptors'].apply(pd.Series)], axis=1)  
  
# Prepare data for modeling  
X = df[['MolWt', 'LogP', 'HBD', 'HBA', 'TPSA']]  
y = np.log10(df['standard\_value'])  
  
# Split data into training and testing sets  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)  
  
  
# Model Training (Random Forest Regression)  
rf\_model = RandomForestRegressor(n\_estimators=100, random\_state=42) # Same model as before  
rf\_model.fit(X\_train, y\_train)  
  
  
# Feature Importance  
importances = rf\_model.feature\_importances\_  
feature\_names = X.columns  
indices = np.argsort(importances)  
  
plt.figure(figsize=(10, 6))  
plt.title('Feature Importances (Random Forest)')  
plt.barh(range(len(indices)), importances[indices], align='center')  
plt.yticks(range(len(indices)), [feature\_names[i] for i in indices])  
plt.xlabel('Relative Importance')  
plt.show()

**Example 4: Topic\_CheMBL\_35\_66\_5\_Model\_Evaluation\_and\_Comparison.ipynb**

#Model Evaluation and Comparison  
import os  
import pandas as pd  
import numpy as np  
from sklearn.model\_selection import train\_test\_split  
from sklearn.linear\_model import LinearRegression  
from sklearn.ensemble import RandomForestRegressor  
from sklearn.metrics import mean\_squared\_error, r2\_score  
import matplotlib.pyplot as plt  
  
# Load data from CSV  
base\_path = ".."  
data\_path = os.path.join(base\_path, "data", "chembl\_35\_66.csv")  
df = pd.read\_csv(data\_path)  
  
# Data Cleaning and Preprocessing  
df = df.dropna(subset=['standard\_value', 'smiles']) # Remove rows with missing values in essential columns  
  
# Convert standard\_value to numeric (explicitly)  
df['standard\_value'] = pd.to\_numeric(df['standard\_value'], errors='coerce')  
df = df.dropna(subset=['standard\_value']) #Remove rows that cannot be converted to numeric  
  
#Handle duplicate Smiles. Keep the one with the lowest IC50 value  
df = df.sort\_values('standard\_value').drop\_duplicates(subset='smiles', keep='first')  
  
  
# RDKit Feature Generation  
from rdkit import Chem  
from rdkit.Chem import Descriptors  
def calculate\_descriptors(smiles):  
 mol = Chem.MolFromSmiles(smiles)  
 if mol is None:  
 return None # Handle invalid SMILES  
 descriptors = {}  
 descriptors['MolWt'] = Descriptors.MolWt(mol)  
 descriptors['LogP'] = Descriptors.MolLogP(mol)  
 descriptors['HBD'] = Descriptors.NumHDonors(mol)  
 descriptors['HBA'] = Descriptors.NumHAcceptors(mol)  
 descriptors['TPSA'] = Descriptors.TPSA(mol)  
 return descriptors  
  
# Apply descriptor calculation to each molecule  
df['descriptors'] = df['smiles'].apply(calculate\_descriptors)  
  
# Remove rows where descriptor calculation failed  
df = df.dropna(subset=['descriptors'])  
  
# Expand the descriptors column into individual columns  
df = pd.concat([df.drop(['descriptors'], axis=1), df['descriptors'].apply(pd.Series)], axis=1)  
  
# Prepare data for modeling  
X = df[['MolWt', 'LogP', 'HBD', 'HBA', 'TPSA']]  
y = np.log10(df['standard\_value'])  
  
# Split data into training and testing sets  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)  
  
# Linear Regression  
lr\_model = LinearRegression()  
lr\_model.fit(X\_train, y\_train)  
lr\_pred = lr\_model.predict(X\_test)  
lr\_mse = mean\_squared\_error(y\_test, lr\_pred)  
lr\_r2 = r2\_score(y\_test, lr\_pred)  
  
# Random Forest Regression  
rf\_model = RandomForestRegressor(n\_estimators=100, random\_state=42)  
rf\_model.fit(X\_train, y\_train)  
rf\_pred = rf\_model.predict(X\_test)  
rf\_mse = mean\_squared\_error(y\_test, rf\_pred)  
rf\_r2 = r2\_score(y\_test, rf\_pred)  
  
print("Linear Regression:")  
print(f" Mean Squared Error: {lr\_mse}")  
print(f" R-squared: {lr\_r2}")  
  
print("\nRandom Forest Regression:")  
print(f" Mean Squared Error: {rf\_mse}")  
print(f" R-squared: {rf\_r2}")  
  
# Visualize predictions  
plt.figure(figsize=(12, 6))  
plt.subplot(1, 2, 1)  
plt.scatter(y\_test, lr\_pred)  
plt.xlabel("Actual Log10(IC50)")  
plt.ylabel("Predicted Log10(IC50)")  
plt.title("Linear Regression: Actual vs. Predicted")  
  
plt.subplot(1, 2, 2)  
plt.scatter(y\_test, rf\_pred)  
plt.xlabel("Actual Log10(IC50)")  
plt.ylabel("Predicted Log10(IC50)")  
plt.title("Random Forest: Actual vs. Predicted")  
  
plt.tight\_layout()  
plt.show()

**Example 5: Topic\_CheMBL\_35\_66\_6\_Outlier\_Analysis.ipynb**

#Outlier Analysis  
import os  
import pandas as pd  
import numpy as np  
from sklearn.model\_selection import train\_test\_split  
from sklearn.linear\_model import LinearRegression  
import matplotlib.pyplot as plt  
import seaborn as sns  
  
  
# Load data from CSV  
base\_path = ".."  
data\_path = os.path.join(base\_path, "data", "chembl\_35\_66.csv")  
df = pd.read\_csv(data\_path)  
  
# Data Cleaning and Preprocessing  
df = df.dropna(subset=['standard\_value', 'smiles']) # Remove rows with missing values in essential columns  
  
# Convert standard\_value to numeric (explicitly)  
df['standard\_value'] = pd.to\_numeric(df['standard\_value'], errors='coerce')  
df = df.dropna(subset=['standard\_value']) #Remove rows that cannot be converted to numeric  
  
#Handle duplicate Smiles. Keep the one with the lowest IC50 value  
df = df.sort\_values('standard\_value').drop\_duplicates(subset='smiles', keep='first')  
  
  
# RDKit Feature Generation  
from rdkit import Chem  
from rdkit.Chem import Descriptors  
def calculate\_descriptors(smiles):  
 mol = Chem.MolFromSmiles(smiles)  
 if mol is None:  
 return None # Handle invalid SMILES  
 descriptors = {}  
 descriptors['MolWt'] = Descriptors.MolWt(mol)  
 descriptors['LogP'] = Descriptors.MolLogP(mol)  
 descriptors['HBD'] = Descriptors.NumHDonors(mol)  
 descriptors['HBA'] = Descriptors.NumHAcceptors(mol)  
 descriptors['TPSA'] = Descriptors.TPSA(mol)  
 return descriptors  
  
# Apply descriptor calculation to each molecule  
df['descriptors'] = df['smiles'].apply(calculate\_descriptors)  
  
# Remove rows where descriptor calculation failed  
df = df.dropna(subset=['descriptors'])  
  
# Expand the descriptors column into individual columns  
df = pd.concat([df.drop(['descriptors'], axis=1), df['descriptors'].apply(pd.Series)], axis=1)  
  
# Prepare data for modeling  
X = df[['MolWt', 'LogP', 'HBD', 'HBA', 'TPSA']]  
y = np.log10(df['standard\_value'])  
  
# Split data into training and testing sets  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)  
  
  
# Train a Linear Regression model (or any model)  
model = LinearRegression()  
model.fit(X\_train, y\_train)  
y\_pred = model.predict(X\_test)  
  
# Calculate residuals  
residuals = y\_test - y\_pred  
  
# Visualize residuals  
plt.figure(figsize=(10, 6))  
sns.scatterplot(x=y\_pred, y=residuals)  
plt.axhline(y=0, color='red', linestyle='--')  
plt.xlabel("Predicted Values")  
plt.ylabel("Residuals")  
plt.title("Residual Plot")  
plt.show()  
  
# Identify potential outliers based on residuals (e.g., beyond 2 standard deviations)  
residual\_std = residuals.std()  
outlier\_threshold = 2 \* residual\_std  
outlier\_indices = residuals[abs(residuals) > outlier\_threshold].index  
  
print(f"Number of potential outliers: {len(outlier\_indices)}")  
print(df.loc[outlier\_indices]) # Display the outlier data points

**Important Considerations:**

* **Install RDKit:** conda install -c conda-forge rdkit or pip install rdkit
* **File Paths:** Double-check that the file paths in your code are correct relative to your project structure.
* **Error Handling:** Add more robust error handling (e.g., using try...except blocks) to handle potential issues with SMILES parsing or descriptor calculation.
* **Hyperparameter Tuning:** Experiment with different hyperparameters for your models (e.g., n\_estimators in Random Forest) to optimize performance.
* **Data Scaling:** Consider scaling your features (e.g., using StandardScaler or MinMaxScaler) before training your models, especially for algorithms sensitive to feature scaling like Linear Regression or Support Vector Machines.