PharmApp Suite

🧠 AI for Drug Discovery and Development 💉

🧩 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
```

```sql
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)

```sql
-- 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)

```python
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.