PharmApp Suite

🧠 AI for Drug Discovery and Development 💉

🧩 Topic: Drug safety flags and bioactivity profile

Okay, I understand. I'm playing the role of a Pharmaceutical Research and Development Specialist, and I will help you set up your Chembl 35 data analysis project using RDKit, focusing on the "Topic_CheMBL_35_55" code. I will provide SQL and Python code snippets, address the identified errors, limit the dataset to 100 rows, and follow the requested AIMLops structure.

1. Analysis of the Analysis Model

This project likely involves exploring structure-activity relationships (SAR) within the ChEMBL 35 database. You are querying the database to extract bioactivity data (likely IC50, Ki, EC50 values) and molecular properties. RDKit will then be used to generate molecular descriptors, and machine learning models will be trained to predict bioactivity based on these descriptors. By focusing on a limited dataset of 100 rows, we are essentially creating a proof-of-concept. This will allows to test the workflow, address errors and adjust parameters before scaling up the analysis to a larger dataset. **Key Analysis steps** * **Data Extraction (SQL):** Retrieve relevant data (molecular structures, bioactivity data) from ChEMBL. * **Data Preprocessing (Python):** Clean and filter the data (e.g., handle missing values, standardize units). Convert SMILES strings to RDKit molecules. * **Feature Generation (Python/RDKit):** Calculate molecular descriptors using RDKit (e.g., molecular weight, logP, topological polar surface area (TPSA), number of rotatable bonds). * **Model Building (Python/Scikit-learn):** Train a machine learning model (e.g., linear regression, random forest, support vector machine) to predict bioactivity based on the generated descriptors. * **Model Evaluation (Python):** Evaluate the model's performance using appropriate metrics (e.g., R-squared, RMSE, MAE) on a held-out test set.

2. SQL and Python Code (with Corrections)

File Structure: Let's assume your project structure looks like this:

```
chembl_35_project/
├── data/
│   └── chembl35_activity_data.csv  # CSV file generated from SQL
├── notebooks/
│   └── Topic_CheMBL_35_55_1_data_extraction.ipynb
│   └── Topic_CheMBL_35_55_2_model_building.ipynb
└── README.md
```

2.1 SQL Code (save as `extract_chembl_data.sql` and run via pgAdmin to generate `chembl35_activity_data.csv`):

```sql
-- Extracting data from ChEMBL 35 for Topic_CheMBL_35_55.
-- Limited to 100 rows for testing.

SELECT
    act.molregno,
    act.standard_type,
    act.standard_relation,
    act.standard_value,
    act.standard_units,
    cmp.canonical_smiles
FROM
    activities act
JOIN
```

```
    molecule_dictionary md ON act.molregno = md.molregno
JOIN
    compound_structures cmp ON md.molregno = cmp.molregno
WHERE
    act.standard_type IN ('IC50', 'Ki', 'EC50')  -- Focus on relevant activity types
    AND act.standard_relation = '='              -- Only consider exact values
    AND act.standard_value IS NOT NULL           -- Exclude missing values
    AND cmp.canonical_smiles IS NOT NULL        -- Exclude missing SMILES
    AND act.standard_value::text ~ '^[0-9\.]+$'  -- Corrected regex for numeric
values. Cast to text before regex.
LIMIT 100;

-- Save the results to a CSV file using pgAdmin's export feature.
```

Explanation of SQL Changes:

- **Error 4.a Fix:** The error `ERROR: operator does not exist: numeric ~ unknown` arises because PostgreSQL's ~ operator (regex match) doesn't directly work on numeric types. To fix this, I've explicitly cast `act.standard_value` to `text` using `act.standard_value::text`. This allows the regex `^[0-9\.]+$` (which matches strings consisting of only digits and periods) to work correctly. This ensures that the `standard_value` is indeed a numeric representation.
- **Limit:** The `LIMIT 100` clause restricts the query to the first 100 rows to avoid overwhelming your machine during testing.
- **Clarity:** Added comments to explain each section of the SQL query.
- **Data Integrity:** Added `AND act.standard_relation = '='` to only get exact values of activities.
- **Missing Values:** Added `AND act.standard_value IS NOT NULL` to avoid errors in downstream analysis.

After running the SQL query in pgAdmin, export the result as a CSV file named `chembl35_activity_data.csv` and save it in the `data/` directory.

2.2 Python Code (in `notebooks/Topic_CheMBL_35_55_1_data_extraction.ipynb`):

```
import os
import pandas as pd
from rdkit import Chem
from rdkit.Chem import Descriptors
import numpy as np #add for handle nan error
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score

# Define base path
base_path = os.path.dirname(os.getcwd()) #Go up to the directory above notebooks
data_path = os.path.join(base_path, 'data', 'chembl35_activity_data.csv')
print(f"Data path: {data_path}")

# Load the data
try:
    df = pd.read_csv(data_path)
    print("Data loaded successfully.")
except FileNotFoundError:
    print(f"Error: File not found at {data_path}.  Make sure you ran the SQL script
and saved the CSV file in the correct location.")
    exit()

# Data Cleaning and Preprocessing
df = df.dropna(subset=['canonical_smiles', 'standard_value']) # Drop rows with missing
```

```
SMILES or activity values
df = df[df['standard_value'].astype(str).str.match(r'^[0-9\.]+$')] # Keep only numeric
activity values as strings
df['standard_value'] = pd.to_numeric(df['standard_value'], errors='coerce')  # Convert
to numeric, handle errors, convert unconvertible values to NaN

# Handle NaN values by removing them
df = df.dropna(subset=['standard_value'])

print(f"Number of rows after cleaning: {len(df)}")
print(df.head())
```

Explanation of Python Code (`Topic_CheMBL_35_55_1_data_extraction.ipynb`):

- **Path Handling:** Uses `os.path.join` to create platform-independent file paths, adhering to AIMLops standards.
- **Error Handling:** Includes a `try-except` block to handle `FileNotFoundError` in case the CSV file is not found.
- **Data Loading:** Loads the CSV file into a Pandas DataFrame.
- **Data Cleaning:**
    - Removes rows with missing SMILES strings or `standard_values`.
    - **Ensures Numeric `standard_values`:** This part is CRUCIAL. It uses `.astype(str).str.match(r'^[0-9\.]+$')` to *explicitly* filter `standard_value` to *only* those strings that represent numeric values (digits and periods). It addresses cases where the values are of mixed types due to issues with the original data. The explicit check with string-based regex avoids potential issues during the numeric conversion in the next step.
    - Converts the `standard_value` column to numeric type using `pd.to_numeric`, handling potential conversion errors by setting invalid values to `NaN`.
    - Removes any rows that resulted in `NaN` in the `standard_value` column after the conversion.

2.3 Python Code (in `notebooks/Topic_CheMBL_35_55_2_model_building.ipynb`):

```
import os
import pandas as pd
from rdkit import Chem
from rdkit.Chem import Descriptors
import numpy as np #add for handle nan error
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score

# Define base path
base_path = os.path.dirname(os.getcwd()) #Go up to the directory above notebooks
data_path = os.path.join(base_path, 'data', 'chembl35_activity_data.csv')

# Load the data (same as before, but crucial to be included in this notebook too!)
try:
    df = pd.read_csv(data_path)
    print("Data loaded successfully.")
except FileNotFoundError:
    print(f"Error: File not found at {data_path}.  Make sure you ran the SQL script
and saved the CSV file in the correct location.")
    exit()

# Data Cleaning and Preprocessing (same as before, crucial to be included in this
notebook too!)
```

```python
df = df.dropna(subset=['canonical_smiles', 'standard_value']) # Drop rows with missing SMILES or activity values
df = df[df['standard_value'].astype(str).str.match(r'^[0-9\.]+$')] # Keep only numeric activity values as strings
df['standard_value'] = pd.to_numeric(df['standard_value'], errors='coerce')  # Convert to numeric, handle errors, convert unconvertible values to NaN

# Handle NaN values by removing them
df = df.dropna(subset=['standard_value'])

# Feature Engineering (RDKit 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['TPSA'] = Descriptors.TPSA(mol)
    descriptors['HBD'] = Descriptors.NumHDonors(mol)
    descriptors['HBA'] = Descriptors.NumHAcceptors(mol)
    descriptors['RotBonds'] = Descriptors.NumRotatableBonds(mol)
    return descriptors

df['descriptors'] = df['canonical_smiles'].apply(calculate_descriptors)
df = df.dropna(subset=['descriptors'])  # Remove rows where descriptor calculation failed (invalid SMILES)
df = df[df['descriptors'].apply(lambda x: isinstance(x, dict))] #Ensure descriptors is a dictionary, removes rows where the lambda function returns none

# Expand the descriptor column into separate columns
df = pd.concat([df.drop(['descriptors'], axis=1), df['descriptors'].apply(pd.Series)], axis=1)

# Model Building
X = df[['MolWt', 'LogP', 'TPSA', 'HBD', 'HBA', 'RotBonds']]
y = df['standard_value']

# Handle any potential NaN values that might have been introduced in descriptor calculation
X = X.fillna(X.mean()) #Impute missing values with the mean

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

model = LinearRegression()
model.fit(X_train, y_train)

# Prediction and Evaluation
y_pred = model.predict(X_test)

#Error 4.b Fix: Remove squared = False
mse = mean_squared_error(y_test, y_pred) #Remove squared = False
r2 = r2_score(y_test, y_pred)

print(f"Mean Squared Error: {mse}")
print(f"R-squared: {r2}")
```

Explanation of Python Code (`Topic_CheMBL_35_55_2_model_building.ipynb`):

- **Dependencies:** Imports necessary libraries.
- **Path Handling:** Same as before. Uses `os.path.join` for robust path construction.
- **Data Loading and Cleaning: CRITICAL:** The data loading and *identical* cleaning steps from `Topic_CheMBL_35_55_1_data_extraction.ipynb` are REPEATED here. This is essential because this notebook runs independently. It cannot rely on the state of the previous notebook. This prevents errors that occur if these processes are not done.
- **Feature Engineering:**
    - Defines a function `calculate_descriptors` that takes a SMILES string, converts it to an RDKit molecule, and calculates a set of molecular descriptors. Returns `None` if the SMILES is invalid.
    - Applies this function to the `canonical_smiles` column to create a new `descriptors` column.
    - Handles the cases where the SMILES strings are invalid by:
        - Removing rows where the `descriptors` calculation failed (SMILES strings were invalid). `df = df.dropna(subset=['descriptors'])`
        - Ensuring descriptors is a dictionary and that rows are removed where the lambda function returns none with `df = df[df['descriptors'].apply(lambda x: isinstance(x, dict))]`
    - Expands the `descriptors` column into separate columns for each descriptor.
- **Model Building:**
    - Selects the descriptor columns as features (X) and the `standard_value` column as the target variable (y).
    - Splits the data into training and testing sets.
    - Creates a linear regression model, trains it on the training data, and makes predictions on the test data.
- **Model Evaluation:**
    - Calculates the mean squared error (MSE) and R-squared (R2) to evaluate the model's performance.
- **Error 4.b Fix:** Removes the `squared=False` parameter from the `mean_squared_error` function. This is because older versions of scikit-learn do not support this parameter.

Important Considerations and Best Practices:

- **Data Exploration:** Before running the model, perform exploratory data analysis (EDA) to understand the distributions of the variables, identify potential outliers, and look for correlations between features.
- **Feature Selection/Engineering:** The choice of descriptors can significantly impact model performance. Consider using feature selection techniques to identify the most relevant descriptors or engineering new features based on domain knowledge.
- **Model Selection:** Linear regression is a simple model but may not be appropriate for all datasets. Experiment with other machine learning models, such as random forests, support vector machines, or neural networks.
- **Hyperparameter Tuning:** Optimize the hyperparameters of the chosen model using techniques like cross-validation and grid search.
- **Scaling/Normalization:** Consider scaling or normalizing the features before training the model, especially if you are using models that are sensitive to feature scaling (e.g., support vector machines).
- **Logging:** Use a logging library to track the progress of your experiments, record important parameters and metrics, and facilitate reproducibility.

3. Five Examples of How to Run the Code

These examples show how to integrate the code in different scenarios. They assume you have the correct folder structure and that the `chembl35_activity_data.csv` file is in the `data` directory.

Example 1: Running the entire pipeline from start to finish (CLI)

1. **Execute SQL:** Using your terminal or command prompt (assuming `psql` is in your PATH) navigate to the `chembl_35_project` directory. `bash psql -h 192.168.206.136 -U rd -d chembl_35 -f extract_chembl_data.sql -o data/chembl35_activity_data.csv -P "header=on" -P "delimiter=," -P "format=csv"` *(This assumes you have `extract_chembl_data.sql` in the project root.) Note: The specific command-line options for exporting to CSV depend on the database system; you may need to adjust them based on your psql version and preferences.*

2. **Run the Notebooks:**

    - Open a terminal and navigate to the `chembl_35_project` directory.
    - Activate your Conda environment (if you are using one): `conda activate <your_env_name>`
    - Run the notebooks in order: `bash jupyter notebook notebooks/Topic_CheMBL_35_55_1_data_extraction.ipynb jupyter notebook notebooks/Topic_CheMBL_35_55_2_model_building.ipynb`
        - This will open the notebooks in your web browser. Run all cells in each notebook sequentially.

Example 2: Running from a single Python script (after SQL execution):

Create a new Python file (e.g., `run_analysis.py`) in your project directory with the following content:

```
import os
import pandas as pd
from rdkit import Chem
from rdkit.Chem import Descriptors
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score

# --- Data Loading and Cleaning (Combined from both notebooks) ---
base_path = os.path.dirname(os.path.abspath(__file__))  # Get the directory of the
script
data_path = os.path.join(base_path, 'data', 'chembl35_activity_data.csv')

try:
    df = pd.read_csv(data_path)
    print("Data loaded successfully.")
except FileNotFoundError:
    print(f"Error: File not found at {data_path}.  Make sure you ran the SQL script
and saved the CSV file in the correct location.")
    exit()

df = df.dropna(subset=['canonical_smiles', 'standard_value'])
df = df[df['standard_value'].astype(str).str.match(r'^[0-9\.]+$')]
df['standard_value'] = pd.to_numeric(df['standard_value'], errors='coerce')
df = df.dropna(subset=['standard_value'])

# --- Feature Engineering ---
def calculate_descriptors(smiles):
    mol = Chem.MolFromSmiles(smiles)
    if mol is None:
        return None
    descriptors = {}
```

```
    descriptors['MolWt'] = Descriptors.MolWt(mol)
    descriptors['LogP'] = Descriptors.MolLogP(mol)
    descriptors['TPSA'] = Descriptors.TPSA(mol)
    descriptors['HBD'] = Descriptors.NumHDonors(mol)
    descriptors['HBA'] = Descriptors.NumHAcceptors(mol)
    descriptors['RotBonds'] = Descriptors.NumRotatableBonds(mol)
    return descriptors

df['descriptors'] = df['canonical_smiles'].apply(calculate_descriptors)
df = df.dropna(subset=['descriptors'])
df = df[df['descriptors'].apply(lambda x: isinstance(x, dict))]
df = pd.concat([df.drop(['descriptors'], axis=1), df['descriptors'].apply(pd.Series)],
axis=1)

# --- Model Building and Evaluation ---
X = df[['MolWt', 'LogP', 'TPSA', 'HBD', 'HBA', 'RotBonds']]
y = df['standard_value']
X = X.fillna(X.mean())

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

model = LinearRegression()
model.fit(X_train, y_train)

y_pred = model.predict(X_test)

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}")
```

- **Run from Terminal:** `python run_analysis.py`

Example 3: Using a Configuration File:

This is a more sophisticated approach for managing parameters.

1. **Create a Config File (e.g., `config.yaml`):**

```
database:
  host: 192.168.206.136
  user: rd
  password: rd
  name: chembl_35
data_path: data/chembl35_activity_data.csv
model:
  test_size: 0.2
  random_state: 42
features:
  - MolWt
  - LogP
  - TPSA
  - HBD
  - HBA
  - RotBonds
```

2. **Modify the Python Script (e.g., `run_analysis_config.py`):**

```python
import os
import pandas as pd
from rdkit import Chem
from rdkit.Chem import Descriptors
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
import yaml  # Import the YAML library

# Load configuration from YAML file
with open("config.yaml", "r") as f:
    config = yaml.safe_load(f)

# --- Data Loading and Cleaning ---
base_path = os.path.dirname(os.path.abspath(__file__))  # Get the directory of
the script
data_path = os.path.join(base_path, config['data_path'])

try:
    df = pd.read_csv(data_path)
    print("Data loaded successfully.")
except FileNotFoundError:
    print(f"Error: File not found at {data_path}.")
    exit()

df = df.dropna(subset=['canonical_smiles', 'standard_value'])
df = df[df['standard_value'].astype(str).str.match(r'^[0-9\.]+$')]
df['standard_value'] = pd.to_numeric(df['standard_value'], errors='coerce')
df = df.dropna(subset=['standard_value'])

# --- Feature Engineering ---
def calculate_descriptors(smiles):
    mol = Chem.MolFromSmiles(smiles)
    if mol is None:
        return None
    descriptors = {}
    descriptors['MolWt'] = Descriptors.MolWt(mol)
    descriptors['LogP'] = Descriptors.MolLogP(mol)
    descriptors['TPSA'] = Descriptors.TPSA(mol)
    descriptors['HBD'] = Descriptors.NumHDonors(mol)
    descriptors['HBA'] = Descriptors.NumHAcceptors(mol)
    descriptors['RotBonds'] = Descriptors.NumRotatableBonds(mol)
    return descriptors

df['descriptors'] = df['canonical_smiles'].apply(calculate_descriptors)
df = df.dropna(subset=['descriptors'])
df = df[df['descriptors'].apply(lambda x: isinstance(x, dict))]
df = pd.concat([df.drop(['descriptors'], axis=1),
df['descriptors'].apply(pd.Series)], axis=1)

# --- Model Building and Evaluation ---
X = df[config['features']]
y = df['standard_value']
X = X.fillna(X.mean())

X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=config['model']['test_size'], random_state=config['model']
['random_state'])
```

```
model = LinearRegression()
model.fit(X_train, y_train)

y_pred = model.predict(X_test)

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}")
```

3. **Run from Terminal:** `python run_analysis_config.py`

Example 4: Modularizing the code into functions

This is a good way to reuse functions.

```
import os
import pandas as pd
from rdkit import Chem
from rdkit.Chem import Descriptors
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score

def load_and_clean_data(data_path):
    """Loads data from CSV, cleans it, and returns a Pandas DataFrame."""
    try:
        df = pd.read_csv(data_path)
        print("Data loaded successfully.")
    except FileNotFoundError:
        print(f"Error: File not found at {data_path}.")
        exit()

    df = df.dropna(subset=['canonical_smiles', 'standard_value'])
    df = df[df['standard_value'].astype(str).str.match(r'^[0-9\.]+$')]
    df['standard_value'] = pd.to_numeric(df['standard_value'], errors='coerce')
    df = df.dropna(subset=['standard_value'])
    return df

def calculate_descriptors(smiles):
    """Calculates RDKit descriptors for a given SMILES string."""
    mol = Chem.MolFromSmiles(smiles)
    if mol is None:
        return None
    descriptors = {}
    descriptors['MolWt'] = Descriptors.MolWt(mol)
    descriptors['LogP'] = Descriptors.MolLogP(mol)
    descriptors['TPSA'] = Descriptors.TPSA(mol)
    descriptors['HBD'] = Descriptors.NumHDonors(mol)
    descriptors['HBA'] = Descriptors.NumHAcceptors(mol)
    descriptors['RotBonds'] = Descriptors.NumRotatableBonds(mol)
    return descriptors

def feature_engineering(df):
  """Calculate the descriptors and return a dataframe with the new features"""
  df['descriptors'] = df['canonical_smiles'].apply(calculate_descriptors)
  df = df.dropna(subset=['descriptors'])
  df = df[df['descriptors'].apply(lambda x: isinstance(x, dict))]
  df = pd.concat([df.drop(['descriptors'], axis=1),
df['descriptors'].apply(pd.Series)], axis=1)
```

```
    return df

def build_and_evaluate_model(X, y, test_size=0.2, random_state=42):
    """Builds and evaluates a linear regression model."""
    X = X.fillna(X.mean()) # Handle NaN values

    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=test_size,
random_state=random_state)

    model = LinearRegression()
    model.fit(X_train, y_train)

    y_pred = model.predict(X_test)

    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}")
    return model #Return the model for later use if needed

if __name__ == "__main__":
    base_path = os.path.dirname(os.path.abspath(__file__))
    data_path = os.path.join(base_path, 'data', 'chembl35_activity_data.csv')

    # Load and clean the data
    df = load_and_clean_data(data_path)

    # Feature Engineering
    df = feature_engineering(df)

    # Define features and target
    X = df[['MolWt', 'LogP', 'TPSA', 'HBD', 'HBA', 'RotBonds']]
    y = df['standard_value']

    # Build and evaluate the model
    model = build_and_evaluate_model(X, y)
```

Example 5: Creating a class for better organization

```
import os
import pandas as pd
from rdkit import Chem
from rdkit.Chem import Descriptors
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score

class BioactivityModel:
    def __init__(self, data_path):
        self.data_path = data_path
        self.df = None #Initialize dataframe

    def load_and_clean_data(self):
        """Loads data from CSV, cleans it, and stores it in self.df."""
        try:
            self.df = pd.read_csv(self.data_path)
            print("Data loaded successfully.")
        except FileNotFoundError:
            print(f"Error: File not found at {self.data_path}.")
```

```
            exit()

        self.df = self.df.dropna(subset=['canonical_smiles', 'standard_value'])
        self.df = self.df[self.df['standard_value'].astype(str).str.match(r'^[0-9\.]+
$')]
        self.df['standard_value'] = pd.to_numeric(self.df['standard_value'],
errors='coerce')
        self.df = self.df.dropna(subset=['standard_value'])

    def calculate_descriptors(self, smiles):
        """Calculates RDKit descriptors for a given SMILES string."""
        mol = Chem.MolFromSmiles(smiles)
        if mol is None:
            return None
        descriptors = {}
        descriptors['MolWt'] = Descriptors.MolWt(mol)
        descriptors['LogP'] = Descriptors.MolLogP(mol)
        descriptors['TPSA'] = Descriptors.TPSA(mol)
        descriptors['HBD'] = Descriptors.NumHDonors(mol)
        descriptors['HBA'] = Descriptors.NumHAcceptors(mol)
        descriptors['RotBonds'] = Descriptors.NumRotatableBonds(mol)
        return descriptors

    def feature_engineering(self):
        """Calculate the descriptors and add them to the dataframe."""
        if self.df is None:
            print("Error: Dataframe is not loaded. Call load_and_clean_data() first.")
            return

        self.df['descriptors'] =
self.df['canonical_smiles'].apply(self.calculate_descriptors)
        self.df = self.df.dropna(subset=['descriptors'])
        self.df = self.df[self.df['descriptors'].apply(lambda x: isinstance(x, dict))]
        self.df = pd.concat([self.df.drop(['descriptors'], axis=1),
self.df['descriptors'].apply(pd.Series)], axis=1)

    def build_and_evaluate_model(self, features, target, test_size=0.2,
random_state=42):
        """Builds and evaluates a linear regression model."""
        if self.df is None:
            print("Error: Dataframe is not loaded. Call load_and_clean_data() first.")
            return

        X = self.df[features]
        y = self.df[target]

        X = X.fillna(X.mean()) # Handle NaN values

        X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=test_size,
random_state=random_state)

        model = LinearRegression()
        model.fit(X_train, y_train)

        y_pred = model.predict(X_test)

        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}")
        return model

if __name__ == "__main__":
    base_path = os.path.dirname(os.path.abspath(__file__))
    data_path = os.path.join(base_path, 'data', 'chembl35_activity_data.csv')

    # Create an instance of the BioactivityModel class
    model = BioactivityModel(data_path)

    # Load and clean the data
    model.load_and_clean_data()

    # Feature Engineering
    model.feature_engineering()

    # Define features and target
    features = ['MolWt', 'LogP', 'TPSA', 'HBD', 'HBA', 'RotBonds']
    target = 'standard_value'

    # Build and evaluate the model
    linear_model = model.build_and_evaluate_model(features, target)
```

Important Notes for All Examples:

- **Environment:** Ensure you have the necessary Python libraries installed (pandas, RDKit, scikit-learn). Use `pip install pandas rdkit scikit-learn pyyaml` to install them if you don't.
- **Adapt Paths:** Adjust the file paths (especially `data_path`) to match your actual project structure.
- **Data Validation:** Thoroughly inspect your data after each step (loading, cleaning, feature engineering) to ensure it's in the expected format and that no unexpected data loss has occurred.
- **Error Handling:** Implement more robust error handling (e.g., logging exceptions) in production code.
- **Experiment:** These examples are a starting point. Experiment with different machine learning models, feature engineering techniques, and hyperparameter settings to improve the performance of your model.

I hope this helps! Let me know if you have any other questions.

Analyse en Français (French Analysis):

Ce projet vise à explorer les relations structure-activité (SAR) dans la base de données ChEMBL 35, en se concentrant sur le code "Topic_CheMBL_35_55". Vous interrogez la base de données pour extraire des données de bioactivité (probablement les valeurs IC50, Ki, EC50) et les propriétés moléculaires. RDKit sera ensuite utilisé pour générer des descripteurs moléculaires, et des modèles d'apprentissage automatique seront entraînés pour prédire la bioactivité en fonction de ces descripteurs. En nous concentrant sur un ensemble de données limité à 100 lignes, nous créons essentiellement une preuve de concept. Cela nous permettra de tester le flux de travail, de corriger les erreurs et d'ajuster les paramètres avant de mettre à l'échelle l'analyse sur un ensemble de données plus important.

Étapes Clés d'Analyse:

- **Extraction de données (SQL) :** Récupérer les données pertinentes (structures moléculaires, données de bioactivité) de ChEMBL.

- **Prétraitement des données (Python) :** Nettoyer et filtrer les données (par exemple, gérer les valeurs manquantes, standardiser les unités). Convertir les chaînes SMILES en molécules RDKit