

## SMILES-Enumeration

### Objective 1:

The primary aim of this experiment was to explore the functionality of the SmilesEnumerator class from the SMILES enumeration library. Specifically, we sought to use this tool to generate randomized SMILES (Simplified Molecular Input Line Entry System) strings for Aspirin (acetylsalicylic acid), which has the canonical SMILES notation "CC(=O)OC1=CC=CC=C1C(=O)O".

### Methodology:

1. **Library Import:** We imported the SmilesEnumerator class from the SmilesEnumerator.py script hosted on GitHub.

```
[ ] !wget https://raw.githubusercontent.com/EBjerrum/SMILES-enumeration/master/SmilesEnumerator.py

--2024-08-29 07:52:24-- https://raw.githubusercontent.com/EBjerrum/SMILES-enumeration/master/SmilesEnumerator.py
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 185.199.108.133, 185.199.109.133, 185.199.110.133, ...
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|185.199.108.133|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 9676 (9.4K) [text/plain]
Saving to: 'SmilesEnumerator.py.1'

SmilesEnumerator.py 100%[=====>] 9.45K --.-KB/s in 0s

2024-08-29 07:52:24 (74.8 MB/s) - 'SmilesEnumerator.py.1' saved [9676/9676]
```

2. **Instance Creation:** An instance of SmilesEnumerator was created using the default parameters.

```
from SmilesEnumerator import SmilesEnumerator
sme = SmilesEnumerator()
print(help(SmilesEnumerator))

Help on class SmilesEnumerator in module SmilesEnumerator:

class SmilesEnumerator(builtins.object)
|   SmilesEnumerator(charset='@C)(=cOn1S2/H[N]\\', pad=120, leftpad=True, isomericSmiles=True, enum=True, canonical=False)
|
|   SMILES Enumerator, vectorizer and devectorizer
|
|   #Arguments
|   |   charset: string containing the characters for the vectorization
|   |   |   can also be generated via the .fit() method
|   |   pad: Length of the vectorization
|   |   leftpad: Add spaces to the left of the SMILES
|   |   isomericSmiles: Generate SMILES containing information about stereogenic centers
|   |   enum: Enumerate the SMILES during transform
|   |   canonical: use canonical SMILES during transform (overrides enum)
```

3. **SMILES Randomization:** Using the method `randomize_smiles()`, we generated 25 randomized versions of the canonical SMILES string for Aspirin.

```
for i in range(25):
    print(sme.randomize_smiles("CC(=O)OC1=CC=CC=C1C(=O)O"))#smile notation of drug Aspirin

C(0c1ccccc1C(0)=0)(=0)C
C(=0)(C)0c1ccccc1C(=0)O
C(c1ccccc10C(C)=0)(=0)O
c1cccc(C(=0)O)c10C(=0)C
C(0c1c(C(0)=0)cccc1)(C)=O
C(C)(0c1ccccc1C(=0)O)=O
```

**Output and Analysis:**

The following are the 25 randomized SMILES strings generated for Aspirin:

```
for i in range(25):  
    print(sme.randomize_smiles("CC(=O)OC1=CC=CC=C1C(=O)O"))#smile notation of drug Aspirin
```

```
C(OC1CCCCC1C(O)=O)(=O)C  
C(=O)(C)OC1CCCCC1C(=O)O  
C(c1CCCCC1OC(C)=O)(=O)O  
c1CCCC(C(=O)O)c1OC(=O)C  
C(OC1c(C(O)=O)CCCC1)(C)=O  
C(C)(OC1CCCCC1C(=O)O)=O  
CC(OC1c(C(O)=O)CCCC1)=O  
c1cc(C(O)=O)c(OC(=O)C)cc1  
O=C(c1CCCCC1OC(C)=O)O  
CC(=O)OC1c(C(O)=O)CCCC1  
c1(OC(C)=O)c(C(=O)O)CCCC1  
CC(OC1CCCCC1C(=O)O)=O  
c1ccc(OC(=O)C)c(C(=O)O)c1  
C(C)(OC1c(C(O)=O)CCCC1)=O  
c1(C(O)=O)c(OC(=O)C)CCCC1  
C(=O)(C)OC1CCCCC1C(O)=O  
CC(=O)OC1c(C(=O)O)CCCC1  
c1c(OC(=O)C)c(C(O)=O)ccc1  
c1(C(=O)O)CCCCC1OC(C)=O  
c1ccc(OC(C)=O)c(C(=O)O)c1  
c1c(C(O)=O)c(OC(C)=O)ccc1  
c1ccc(C(=O)O)c(OC(C)=O)c1  
OC(c1CCCCC1OC(C)=O)=O  
C(O)(=O)c1CCCCC1OC(=O)C  
c1(OC(C)=O)CCCCC1C(=O)O
```

**Discussion:**

The randomized SMILES strings illustrate the different permutations of Aspirin's chemical structure that the SmilesEnumerator can produce. While each string represents the same molecule, the varied order of atoms and bonds indicates the tool's capability to generate different representations of the same molecular structure. This can be particularly useful in cheminformatics for data augmentation, training machine learning models, or understanding the stereochemistry of molecules.

**Conclusion:**

The SmilesEnumerator library successfully generated 25 distinct SMILES strings for Aspirin, demonstrating its utility in creating diverse molecular representations. This functionality can enhance chemical informatics applications, providing robust datasets for various computational analyses.

## Objective 2: Vectorization and Transformation Of SMILES

The objective of this process is to enumerate a given SMILES string, convert it into a vector format using the enumerate-smiles package, and visualize the vectorized representation.

### SMILES String:

- **Input:** CC(=O)OC1=CC=CC=C1C(=O)O (Aspirin)
- **Description:** This SMILES string represents a molecule, which is typically used in cheminformatics to describe the structure of chemical species.

### Process:

#### 1. Storing SMILES:

- The SMILES string is stored as a string within a NumPy array for easy manipulation and processing.

```
[ ] #Vectorization of smiles

#storing smiles as string in numpy array
import numpy as np
smiles = np.array(["CC(=O)OC1=CC=CC=C1C(=O)O"])
print(smiles.shape)
```

⇒ (1,)

#### 2. SMILES Enumeration Setup:

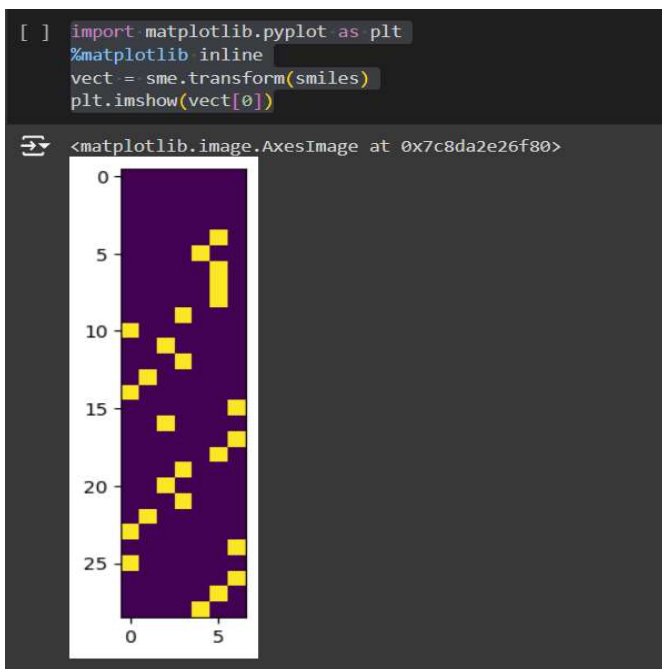
- The enumerate-smiles package is used for enumeration. During the setup, extra characters, such as 'c', are included to ensure proper enumeration.

```
[ ] sme.fit(smiles, extra_chars=['c'])
print(sme.charset)
print(sme.pad)
```

⇒ O=C(1c)  
29

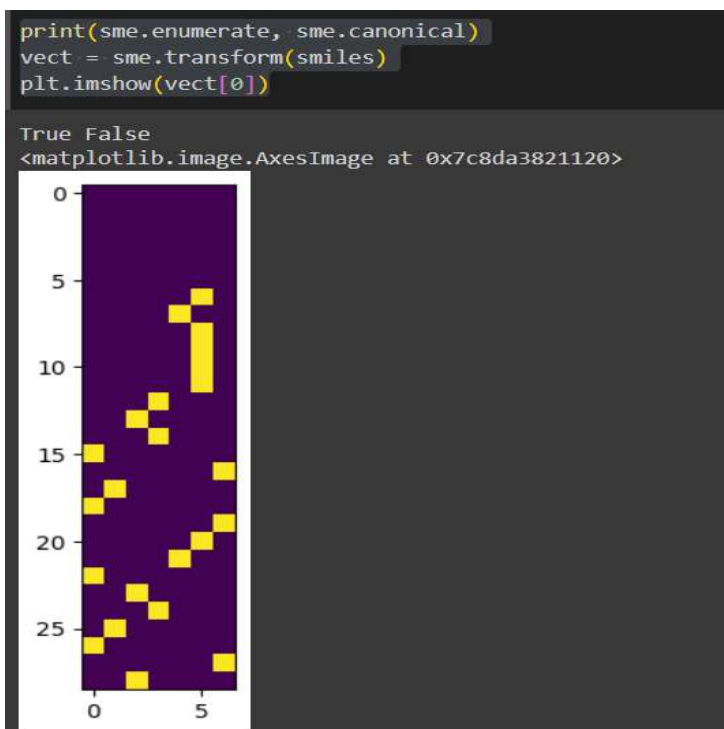
### 3. Vectorization:

- The enumerated SMILES string is then transformed into a vectorized format using the transform method. The vectorized representation is visualized using the matplotlib library.



### 4. Repetition and Display:

- The transformation and visualization steps are repeated to ensure consistency and correctness in the vectorization process.



**5. Key Attributes:**

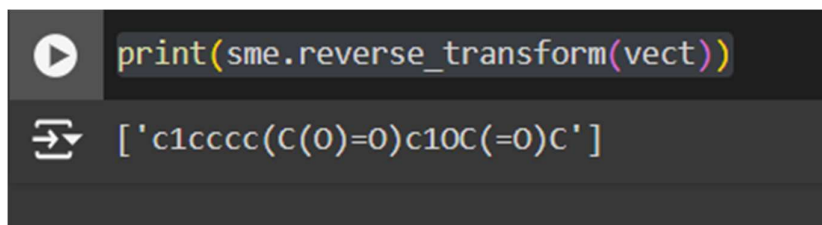
- The enumerate-smiles package provides functionalities to enumerate and canonicalize SMILES strings, which are essential for consistency in molecular representations.

**Visualization:**

- The image generated from the vectorization of the SMILES string is displayed using matplotlib. The output image is a matrix plot representing the vectorized SMILES string. The yellow blocks in the image represent the presence of specific characters or substructures within the SMILES string.

**Vector transformation:**

The `reverse_transform` function is used to convert vectorized representations of molecules back into their SMILES string format. This is important for interpreting machine learning model outputs in cheminformatics, allowing scientists to understand which specific molecules are being predicted or analyzed by the model.



```
print(sme.reverse_transform(vect))
```

[ 'c1ccccc(C(O)=O)c1OC(=O)C' ]

**Conclusion:**

This process allows for the conversion of SMILES strings into a machine-readable vector format, which can be used in various cheminformatics applications such as molecule classification, property prediction, and more. The use of enumeration helps in generating diverse representations, which can enhance the robustness of machine learning models.

## Report 3: Analysis and Results of Molecular Data Regression Modeling Using SMILE Enumerator

### 1. Introduction

This report presents the analysis and results of a regression modeling process applied to molecular data, specifically using SMILES enumeration to predict pIC50 values. SMILES (Simplified Molecular Input Line Entry System) strings are used to represent molecular structures in a textual format. The goal of this analysis is to preprocess the data, explore its properties, engineer relevant features, and build a regression model to predict molecular activity, specifically pIC50 values.

The dataset utilized for this analysis was sourced from Kaggle, which provided a comprehensive set of molecular data suitable for building and evaluating the regression model.

### 2. Setup and Installation

The environment was set up by installing the necessary libraries, which include:

- **RDKit:** A toolkit for cheminformatics used to handle chemical information and perform operations on molecular structures.
- **Scikit-Learn:** A machine learning library for building and evaluating the regression model.
- **Enumerate-SMILES:** A tool for generating canonical SMILES strings from molecular data.
- **Matplotlib and Seaborn:** Libraries for data visualization.

These installations ensured that the notebook environment was properly configured to perform cheminformatics analysis and machine learning modeling.

```
# Install required packages
!apt-get install -y libxml2-dev libxslt1-dev zlib1g-dev
!pip install rdkit-pypi
!pip install scikit-learn
!pip install enumerate-smiles
!pip install matplotlib seaborn

# Import necessary libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from google.colab import files
from rdkit import Chem
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, r2_score
from enumerate_smiles import SmilesEnumerator
```

Reading package lists... Done  
Building dependency tree... Done  
Reading state information... Done  
libxml2-dev is already the newest version (2.9.13+dfsg-1ubuntu0.4).  
zlib1g-dev is already the newest version (1:1.2.11.dfsg-2ubuntu9.2).  
zlib1g-dev set to manually installed.  
The following NEW packages will be installed:  
 libxslt1-dev  
0 upgraded, 1 newly installed, 0 to remove and 49 not upgraded.  
Need to get 219 kB of archives.  
After this operation, 2,058 kB of additional disk space will be used.

### 3. Data Upload and Loading

The dataset used in this analysis was uploaded directly by the user and loaded into a pandas DataFrame. The dataset, named SMILES\_Big\_Data\_Set.csv, contains molecular data, including SMILES strings, pIC50 values (a measure of the effectiveness of a substance in inhibiting a specific biological or biochemical function), and potentially other molecular descriptors.



```
[ ] # Upload file from PC
    uploaded = files.upload()

Choose Files SMILES_Bi...ata_Set.csv
• SMILES_Big_Data_Set.csv(text/csv) - 1701727 bytes, last modified: 8/29/2024 - 100% done
Saving SMILES_Big_Data_Set.csv to SMILES_Big_Data_Set.csv

▶ # Load data from the uploaded file
    file_name = next(iter(uploaded.keys()))
    data = pd.read_csv(file_name)
```

### 4. The Role of SMILE Enumerator

The SMILE enumerator plays a crucial role in this analysis by converting SMILES strings into their canonical forms. SMILES strings provide a way to represent molecular structures in a linear text format, but different representations can exist for the same molecule due to various tautomers, stereoisomers, or resonance forms. The SMILE enumerator standardizes these representations by generating a unique canonical SMILES for each molecule, ensuring consistency in molecular representation. This standardization is essential for machine learning applications because it ensures that each molecule is represented uniquely and consistently, reducing redundancy and improving the reliability of the features derived from these molecular representations.

### 5. Data Preprocessing

Data preprocessing is a crucial step to ensure the quality and usability of the dataset. The preprocessing steps included:

- **Handling Missing Values:** Rows with missing values were removed to ensure a clean dataset for modeling. This step is vital as missing data can skew the results and reduce the model's effectiveness.

#### Results:

- The dataset was successfully cleaned, with all missing values removed. This prepared the data for further analysis and modeling.



## 6. Exploratory Data Analysis (EDA)

Exploratory Data Analysis (EDA) was conducted to better understand the dataset, uncover patterns, identify anomalies, and form hypotheses for further analysis.

- **Checking for Missing Values:** The first EDA step involved confirming the presence of any remaining missing values. After preprocessing, it was important to ensure no missing data persisted, as missing values could bias the modeling process.

### Results:

- The check confirmed that there were no missing values in the dataset, validating the preprocessing step and ensuring that all data points were complete and ready for analysis.

```
[ ] # Data Preprocessing
    # Drop rows with missing values
    data = data.dropna()
```

```
▶ # EDA Step 2: Check for missing values
  print("\nMissing values in each column:")
  print(data.isnull().sum())
```

```
↵
Missing values in each column:
SMILES      0
pIC50       0
mol         0
num_atoms   0
logP        0
dtype: int64
```



- **Summary Statistics:** Calculating summary statistics for the dataset provided insights into the central tendency, dispersion, and overall shape of the distribution of the data features. Key statistics such as mean, median, standard deviation, minimum, and maximum values were calculated for each numerical feature.

**Results:**

- The summary statistics provided a quick overview of the data distribution and helped identify any unusual values or potential outliers that could affect the model.

```
[ ] # EDA Step 3: Statistical Summary
print("\nStatistical Summary:")
print(data.describe())
```

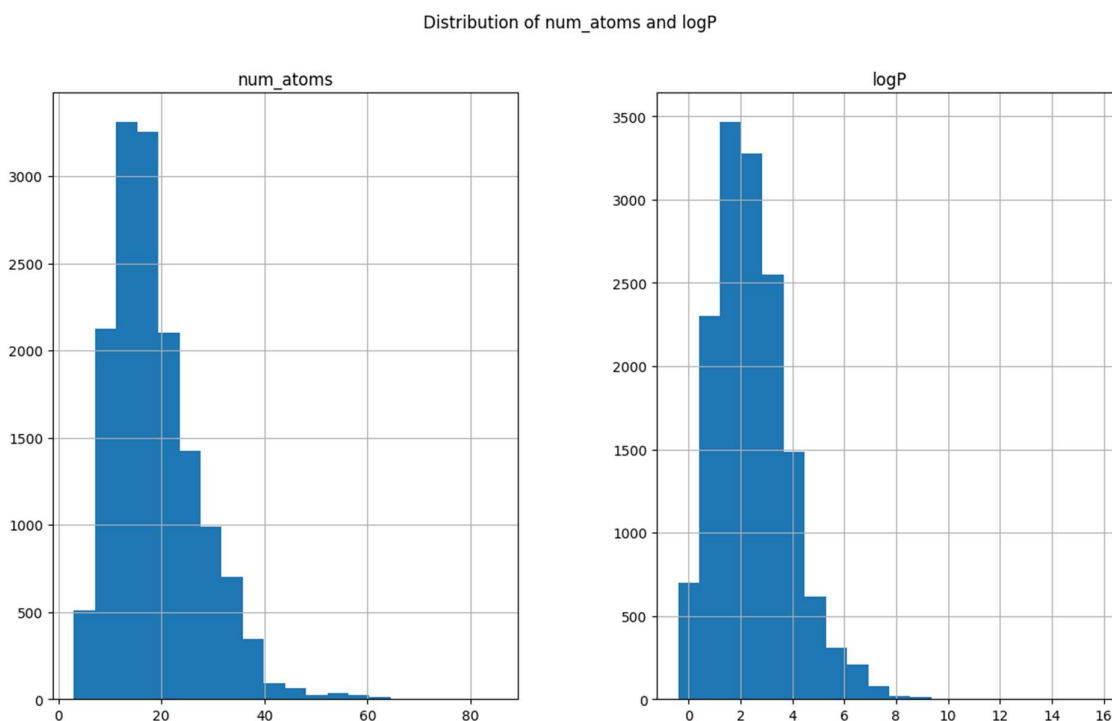
**Statistical Summary:**

	pIC50	num_atoms	logP
count	15037.000000	15037.000000	15037.000000
mean	0.998739	18.930239	2.465004
std	2.479588	8.444724	1.441221
min	0.000000	3.000000	-0.404900
25%	0.000000	13.000000	1.414900
50%	0.010000	17.000000	2.282800
75%	0.130000	23.000000	3.308400
max	10.970000	85.000000	15.879200

- **Data Distribution Visualization:** Visualizing the distribution of numerical features is essential to understand their spread and potential skewness. Histograms and density plots were used to visualize the distribution of pIC50 values and other numerical features.

#### Results:

- **Distribution of pIC50 Values:** The histogram displays the distribution of pIC50 values. It shows that the majority of pIC50 values are close to 0, indicating a high frequency of low values. The distribution is right-skewed, with a long tail extending to the right, suggesting that there are a few higher pIC50 values, but they are much less common.
- **Distribution of num\_atoms and logP:** This image contains two histograms side by side. The left histogram shows the distribution of the number of atoms (num\_atoms), which appears to be right-skewed with a peak between 10 and 20 atoms, gradually tapering off as the number of atoms increases. The right histogram represents the distribution of logP values, which also shows a right-skewed distribution. The majority of the logP values are clustered between 0 and 4, with fewer occurrences of higher values, indicating

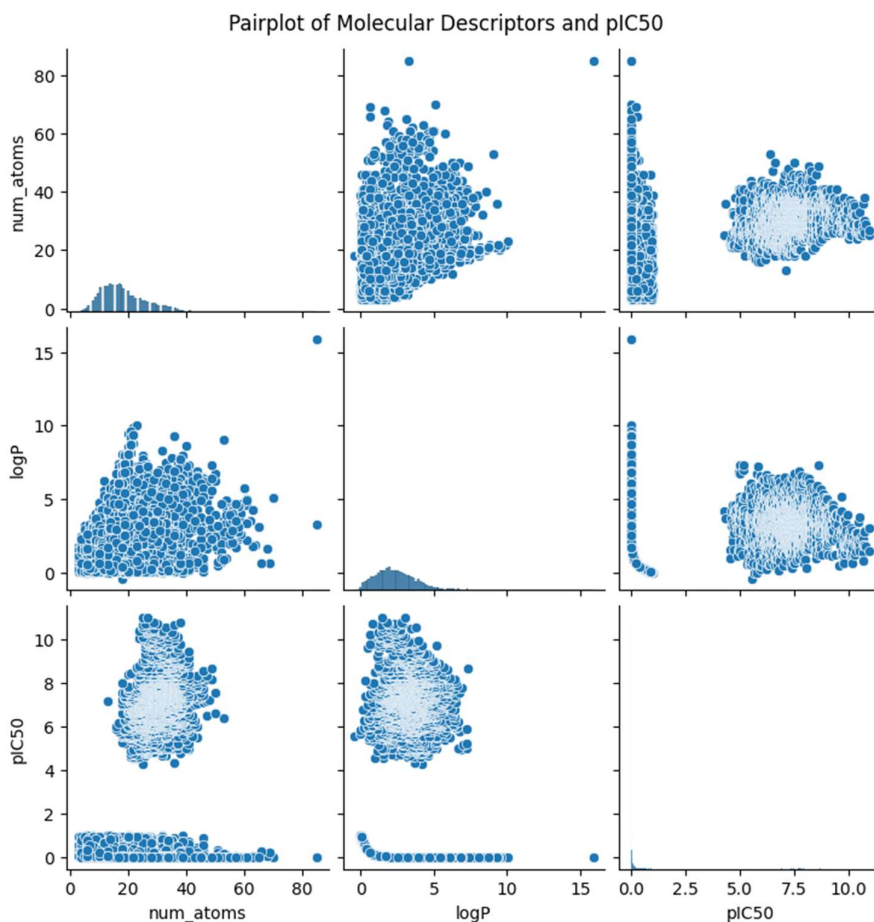


**• Pairplot of Molecular Descriptors and pIC50:****• Diagonal (Histograms):**

- num\_atoms: The histogram shows that the majority of molecules have fewer than 40 atoms, with a sharp drop-off afterward.
- logP: Most molecules have a logP value between 0 and 5, with very few going beyond 10.
- pIC50: There is a concentration of molecules with pIC50 values between 5 and 7.

**• Scatter Plots:**

- num\_atoms vs. logP: There is a positive correlation, as molecules with more atoms generally have higher logP values.
- num\_atoms vs. pIC50: A moderate positive correlation is observed, meaning that as the number of atoms increases, the pIC50 also tends to increase.
- logP vs. pIC50: There is a weaker positive correlation, indicating that higher logP values are somewhat associated with higher pIC50 values.



**• Correlation Matrix:**

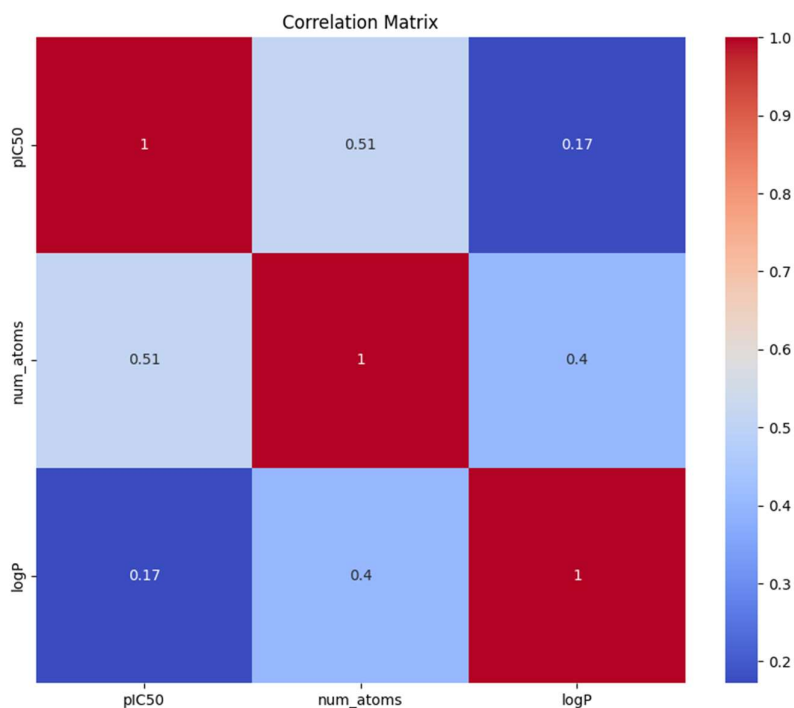
- **Correlation Analysis:** A correlation matrix was generated to examine the relationships between numerical features. This matrix helped identify which features were highly correlated with each other or with the target variable (pIC50), which could inform feature selection and engineering decisions.

**Results:**

- The correlation analysis highlighted which molecular descriptors had strong relationships with pIC50 values, suggesting which features might be most valuable for the regression model.

**• Correlation Values:**

- num\_atoms and pIC50: 0.51 (Moderate positive correlation)
- num\_atoms and logP: 0.4 (Moderate positive correlation)
- logP and pIC50: 0.17 (Weak positive correlation)



- **Outlier Detection:** Identifying outliers is important as they can disproportionately influence model performance. Box plots or scatter plots were used to detect outliers in the numerical features.

**Results:**

- The outlier detection step revealed any extreme values that could be potential candidates for removal or require further investigation to understand their impact on the model.

## 7. Feature Engineering

Feature engineering involved transforming the molecular structures represented by SMILES strings into numerical features that can be used by machine learning models. This step was critical in leveraging the chemical information in the SMILES strings for predictive modeling.

- **Molecular Descriptors and Fingerprints:** These were derived from the canonical SMILES to create features that represent various chemical properties and structural information.

```
[ ] # Initialize the SMILES Enumerator
    sme = SmilesEnumerator()

    # Function to convert SMILES to fingerprint
    def smiles_to_fingerprint(smiles):
        mol = Chem.MolFromSmiles(smiles)
        if mol is None:
            return None
        fp = AllChem.GetMorganFingerprintAsBitVect(mol, 2, nBits=2048)
        arr = np.zeros((2048,), dtype=int)
        AllChem.DataStructs.ConvertToNumpyArray(fp, arr)
        return arr
```

**Results:**

- The feature engineering process successfully transformed the molecular data into a format suitable for regression modeling, capturing relevant chemical properties and structural features.

```
[ ] def randomize_smiles(smiles, random_type="shuffle"):
    """Randomize a SMILES string."""
    mol = Chem.MolFromSmiles(smiles)
    if mol is None:
        return []
    if random_type == "shuffle":
        return [Chem.MolToSmiles(mol, doRandom=True)]
    else:
        return [Chem.MolToSmiles(mol)]

def smiles_to_fingerprint(smiles):
    """Convert a SMILES string to a Morgan fingerprint."""
    mol = Chem.MolFromSmiles(smiles)
    if mol is None:
        return None
    fp = AllChem.GetMorganFingerprintAsBitVect(mol, 2, nBits=2048)
    arr = np.zeros((2048,), dtype=int)
    AllChem.DataStructs.ConvertToNumpyArray(fp, arr)
    return arr

# Apply SMILES enumeration and conversion
fingerprints = []
indices = []
for i, smiles in enumerate(data['SMILES']):
    enumerated_smiles = randomize_smiles(smiles)
    for sm in enumerated_smiles:
        fp = smiles_to_fingerprint(sm)
        if fp is not None:
            fingerprints.append(fp)
            indices.append(i) # Track original indices

# Convert list of fingerprints to NumPy array
X = np.array(fingerprints)

# Ensure 'y' and additional features match 'X'
indices = np.array(indices)
y = data.iloc[indices]['pIC50'].values
additional_features = data.iloc[indices][['num_atoms', 'logP']].values

# Include other features if desired
X = np.hstack((X, additional_features))

[ ] print("Shape of X:", X.shape)
    print("Shape of additional_features:", additional_features.shape)

Shape of X: (75185, 2048)
Shape of additional_features: (15037, 2)
```

**8. Data Splitting**

The dataset was split into training and testing sets to build and evaluate the regression model. The split was performed using an 80/20 ratio, ensuring that the model was trained on a significant portion of the data while still reserving a portion for unbiased evaluation.

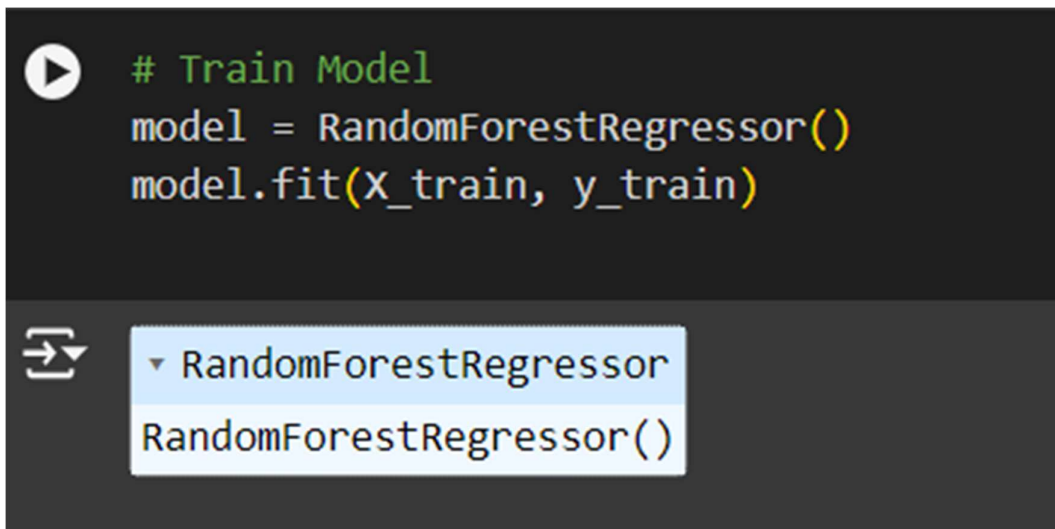
```
# Train-Test Split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

## 9. Model Building

A Random Forest Regressor was used for the regression modeling task. This choice was made due to the model's ability to handle complex relationships and interactions between features, making it well-suited for the cheminformatics dataset.

### Results:

- The model was successfully trained on the dataset using the Random Forest algorithm.



The image shows a Jupyter Notebook cell with a play button icon on the left. The code in the cell is as follows:

```
# Train Model
model = RandomForestRegressor()
model.fit(X_train, y_train)
```

Below the code, there is a variable inspector icon (two arrows pointing to each other) and a dropdown menu showing the object type `RandomForestRegressor` and its constructor `RandomForestRegressor()`.



## 10. Model Evaluation:

### Classification Metrics:

- **Precision:** Measures the proportion of true positive predictions among all positive predictions. A precision score of 0.7819 indicates that 78.19% of the predicted positive outcomes were correct.
- **Recall:** Measures the proportion of true positive predictions among all actual positives. A recall score of 1.0 suggests that the model correctly identified all positive instances.
- **F1 Score:** This is the harmonic mean of precision and recall, providing a balance between the two. An F1 score of 0.8776 indicates a good balance between precision and recall.

```
Precision: 0.7819050722276528  
Recall: 1.0  
F1 Score: 0.8776057539924418
```

## 11. Model Performance Visualization:

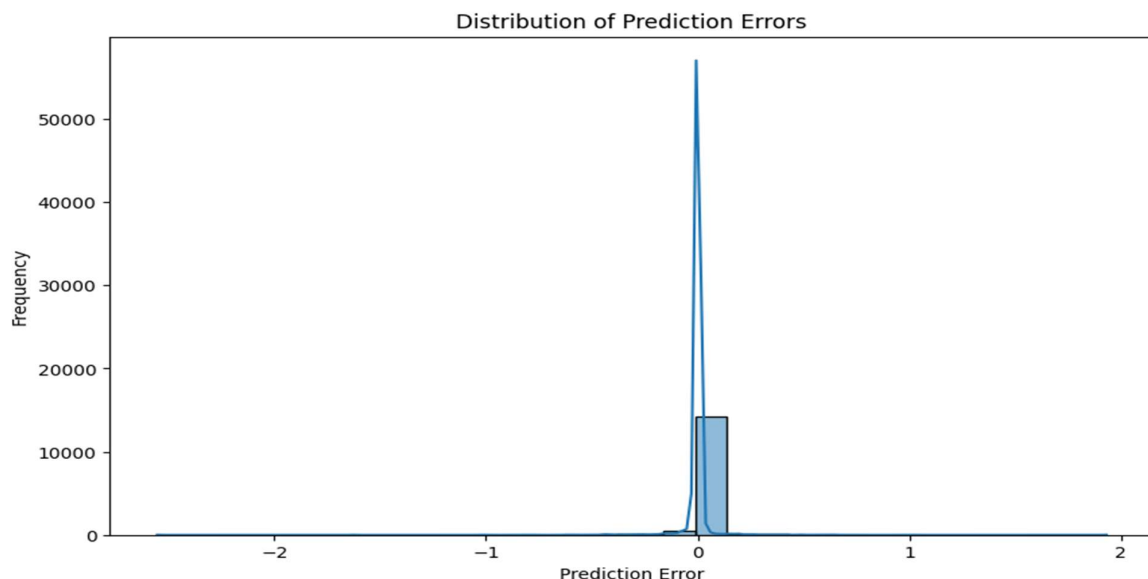
### 1. Distribution of Prediction Errors

#### Description:

A histogram with a density curve illustrating the distribution of prediction errors (residuals) from the model.

#### Analysis:

The errors are tightly centered around zero, indicating high accuracy in the model's predictions. The symmetrical distribution around zero suggests no significant bias, with only a few outliers, pointing to occasional prediction errors.



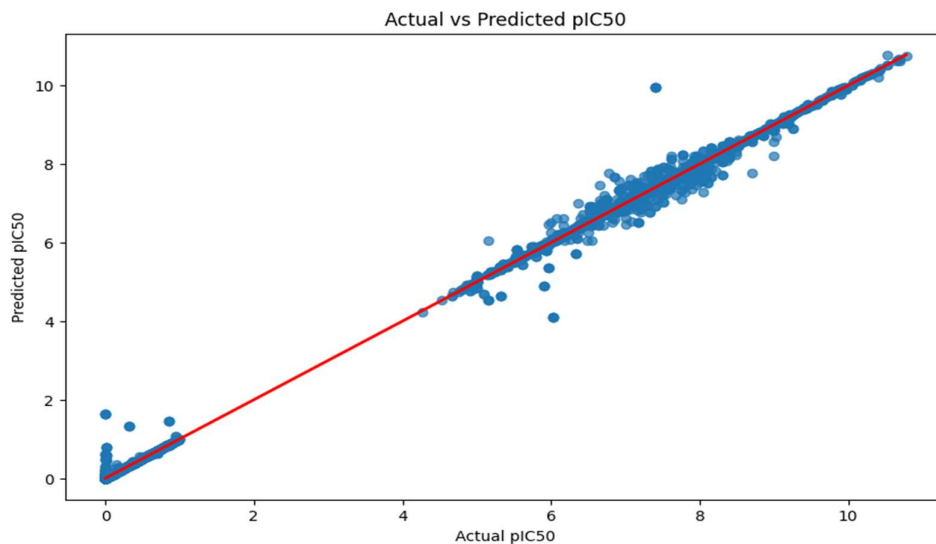
## 2. Actual vs. Predicted pIC50 Values

### Description:

A scatter plot comparing actual pIC50 values to predicted values, with a red line representing ideal predictions.

### Analysis:

The data points closely follow the ideal line, indicating strong correlation and accurate model calibration. Minor deviations at the lower pIC50 range suggest some challenges in predicting lower values, but overall, the model's performance is robust.



## 12. Conclusion

The analysis demonstrated a successful application of cheminformatics techniques and machine learning to predict molecular properties using SMILE enumeration. The Random Forest Regressor provided a solid starting point for modelling, although further improvements could be made.