CHEMICALS IN COSEMETICS

Introduction ¶

The aim of the project is to identify the amount of chemical substances present in a Cosmetic product.The California Safe Cosmetics Program (CSCP), administered by the California Department of Public Health (CDPH), is a vital initiative designed to protect public health by monitoring and disclosing hazardous ingredients in cosmetic products sold in California. The program, established under the California Safe Cosmetics Act, requires cosmetic manufacturers, packers, and distributors to report products containing ingredients that are known or suspected to cause cancer, birth defects, or other developmental or reproductive harm.

Goal of the Project¶

The goal of this project is to analyze and assess the data reported to the California Safe Cosmetics Program (CSCP) to better understand the presence and prevalence of hazardous ingredients in cosmetic products sold in California. By examining this data, the project aims to: 1)Identify trends in the use of chemicals known or suspected to cause cancer, birth defects, or other developmental or reproductive harm within the cosmetic industry. 2)Evaluate the compliance of manufacturers with the reporting requirements set by the California Safe Cosmetics Act, and assess the completeness of the data collected by the CSCP. 3)Increase awareness about the potential health risks associated with certain cosmetic ingredients, and provide insights into how these chemicals are distributed across different product categories and brands. 4)Highlight gaps in the reporting process, including missing data or products that may not be included, and suggest ways to improve transparency and safety within the cosmetics industry. 5)Support consumer education by presenting the data in a user-friendly format, helping consumers make more informed decisions about the personal care products they use.

Importing the libraraies

Load the data using libraries like Pandas. The dataset is typically stored in a CSV or Excel file.

In [1]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings("ignore")
from sklearn.preprocessing import LabelEncoder
from sklearn.feature_selection import SelectKBest,chi2
from sklearn.feature_selection import f_classif
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from imblearn.over_sampling import SMOTE
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, classification_report, confusion_matrix
```

1.Importing the Data Set

In [2]:
```
df=pd.read_csv("C:/Users/user/Downloads/cscpopendata.csv")
df
```

Out[2]:

	CDPHId	ProductName	CSFId	CSF	CompanyId	CompanyName	BrandName	PrimaryCategoryId	PrimaryCategory	SubCategoryId	...	CasNumbe
0	2	ULTRA COLOR RICH EXTRA PLUMP LIPSTICK-ALL SHADES	NaN	NaN	4	New Avon LLC	AVON	44	Makeup Products (non-permanent)	53	...	13463-67-
1	3	Glover's Medicated Shampoo	NaN	NaN	338	J. Strickland & Co.	Glover's	18	Hair Care Products (non-coloring)	25	...	65996-92-
2	3	Glover's Medicated Shampoo	NaN	NaN	338	J. Strickland & Co.	Glover's	18	Hair Care Products (non-coloring)	25	...	140-67-
3	4	PRECISION GLIMMER EYE LINER-ALL SHADES	NaN	NaN	4	New Avon LLC	AVON	44	Makeup Products (non-permanent)	46	...	13463-67-
4	5	AVON BRILLIANT SHINE LIP GLOSS-ALL SHADES	NaN	NaN	4	New Avon LLC	AVON	44	Makeup Products (non-permanent)	52	...	13463-67-
...	...	...	...	...	...	...	...	...	...	...	...	.
114630	41523	HYDRA-LIP TRANSLUCENT COLOR LIPSTICK	65001.0	Rosa Soft	1259	Yanbal USA, Inc	YANBAL	44	Makeup Products (non-permanent)	53	...	13463-67-
114631	41523	HYDRA-LIP TRANSLUCENT COLOR LIPSTICK	65002.0	Malva Spirit	1259	Yanbal USA, Inc	YANBAL	44	Makeup Products (non-permanent)	53	...	13463-67-
114632	41523	HYDRA-LIP TRANSLUCENT COLOR LIPSTICK	65003.0	Rojo Fashion	1259	Yanbal USA, Inc	YANBAL	44	Makeup Products (non-permanent)	53	...	13463-67-
114633	41523	HYDRA-LIP TRANSLUCENT COLOR LIPSTICK	65004.0	Terra Mystic	1259	Yanbal USA, Inc	YANBAL	44	Makeup Products (non-permanent)	53	...	13463-67-
114634	41524	OLD SPICE GENTLEMENS BLEND ALOE AND WILD SAGE ...	NaN	NaN	86	The Procter & Gamble Company	Old Spice	6	Bath Products	159	...	13463-67-

114635 rows × 22 columns

Column Description

1)CDPHId: Likely an ID related to the California Department of Public Health (CDPH).

2)ProductName: The name of the product.

3)CSFId: Likely a unique ID for a CSF (could be related to a specific certification or program).

4)CSF: Information about the CSF, possibly indicating a certification, status, or category.

5)CompanyId: Unique identifier for the company.

6)CompanyName: Name of the company.

7)BrandName: Name of the brand associated with the product.

8)PrimaryCategoryId: ID for the primary category of the product.

9)PrimaryCategory: The main category of the product.

10)SubCategoryId: ID for the subcategory of the product.

11)SubCategory: The subcategory under which the product is listed.

12)CasId: Likely referring to an ID related to the Chemical Abstracts Service (CAS).

13)CasNumber: The CAS number associated with the product or chemical.

14)ChemicalId: Unique identifier for the chemical in the product. 15)ChemicalName: The name of the chemical.

16)InitialDateReported: The first date this product/chemical was reported.

17)MostRecentDateReported: The most recent date the product/chemical was reported.

18)DiscontinuedDate: The date when the product or chemical was discontinued.

19)ChemicalCreatedAt: The creation date of the chemical record.

20)ChemicalUpdatedAt: The last update date of the chemical record.

21)ChemicalDateRemoved: Date when the chemical was removed (from a database or list).

22)ChemicalCount: Likely the number of occurrences or items related to the chemical

2. Data Exploration

In [3]:
```
df.head()# View the first 5 rows
```

Out[3]:

	CDPHId	ProductName	CSFId	CSF	CompanyId	CompanyName	BrandName	PrimaryCategoryId	PrimaryCategory	SubCategoryId	...	CasNumber	Chemical
0	2	ULTRA COLOR RICH EXTRA PLUMP LIPSTICK-ALL SHADES	NaN	NaN	4	New Avon LLC	AVON	44	Makeup Products (non-permanent)	53	...	13463-67-7	
1	3	Glover's Medicated Shampoo	NaN	NaN	338	J. Strickland & Co.	Glover's	18	Hair Care Products (non-coloring)	25	...	65996-92-1	
2	3	Glover's Medicated Shampoo	NaN	NaN	338	J. Strickland & Co.	Glover's	18	Hair Care Products (non-coloring)	25	...	140-67-0	
3	4	PRECISION GLIMMER EYE LINER-ALL SHADES	NaN	NaN	4	New Avon LLC	AVON	44	Makeup Products (non-permanent)	46	...	13463-67-7	
4	5	AVON BRILLIANT SHINE LIP GLOSS-ALL SHADES	NaN	NaN	4	New Avon LLC	AVON	44	Makeup Products (non-permanent)	52	...	13463-67-7	

5 rows × 22 columns

In [4]:
```
df.tail()# View the last 5 rows
```

Out[4]:

	CDPHId	ProductName	CSFId	CSF	CompanyId	CompanyName	BrandName	PrimaryCategoryId	PrimaryCategory	SubCategoryId	...	CasNumbe
114630	41523	HYDRA-LIP TRANSLUCENT COLOR LIPSTICK	65001.0	Rosa Soft	1259	Yanbal USA, Inc	YANBAL	44	Makeup Products (non-permanent)	53	...	13463-67-
114631	41523	HYDRA-LIP TRANSLUCENT COLOR LIPSTICK	65002.0	Malva Spirit	1259	Yanbal USA, Inc	YANBAL	44	Makeup Products (non-permanent)	53	...	13463-67-
114632	41523	HYDRA-LIP TRANSLUCENT COLOR LIPSTICK	65003.0	Rojo Fashion	1259	Yanbal USA, Inc	YANBAL	44	Makeup Products (non-permanent)	53	...	13463-67-
114633	41523	HYDRA-LIP TRANSLUCENT COLOR LIPSTICK	65004.0	Terra Mystic	1259	Yanbal USA, Inc	YANBAL	44	Makeup Products (non-permanent)	53	...	13463-67-
114634	41524	OLD SPICE GENTLEMENS BLEND ALOE AND WILD SAGE ...	NaN	NaN	86	The Procter & Gamble Company	Old Spice	6	Bath Products	159	...	13463-67-

5 rows × 22 columns

In [5]:
```
df.shape # To find the number of rows and coloumns
```

Out[5]: (114635, 22)

In [6]:
```
df.info()# Get basic info about the dataset (data types, non-null counts)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 114635 entries, 0 to 114634
Data columns (total 22 columns):
 #   Column                  Non-Null Count   Dtype
---  ------                  --------------   -----
 0   CDPHId                  114635 non-null  int64
 1   ProductName             114635 non-null  object
 2   CSFId                   80662 non-null   float64
 3   CSF                     80237 non-null   object
 4   CompanyId               114635 non-null  int64
 5   CompanyName             114635 non-null  object
 6   BrandName               114408 non-null  object
 7   PrimaryCategoryId       114635 non-null  int64
 8   PrimaryCategory         114635 non-null  object
 9   SubCategoryId           114635 non-null  int64
 10  SubCategory             114635 non-null  object
 11  CasId                   114635 non-null  int64
 12  CasNumber               108159 non-null  object
 13  ChemicalId              114635 non-null  int64
 14  ChemicalName            114635 non-null  object
 15  InitialDateReported     114635 non-null  object
 16  MostRecentDateReported  114635 non-null  object
 17  DiscontinuedDate        12920 non-null   object
 18  ChemicalCreatedAt       114635 non-null  object
 19  ChemicalUpdatedAt       114635 non-null  object
 20  ChemicalDateRemoved     2985 non-null    object
 21  ChemicalCount           114635 non-null  int64
dtypes: float64(1), int64(7), object(14)
memory usage: 19.2+ MB
```

In [170]:
```
df.columns# Identify the columns in the dataset
```

Out[170]:
```
Index(['ProductName', 'CompanyName', 'BrandName', 'PrimaryCategoryId',
       'PrimaryCategory', 'SubCategoryId', 'SubCategory', 'CasId', 'CasNumber',
       'ChemicalId', 'ChemicalName', 'InitialDateReported',
       'MostRecentDateReported', 'ChemicalCreatedAt', 'ChemicalUpdatedAt',
       'ChemicalCount'],
      dtype='object')
```

In [9]:
```
df.describe().T # Summary statistics for numerical columns
```

Out[9]:

	count	mean	std	min	25%	50%	75%	max
CDPHId	114635.0	20304.858987	12489.052554	2.0	8717.0	20895.0	31338.50	41524.0
CSFId	80662.0	32608.658377	19089.443910	1.0	15789.0	32541.0	48717.75	65009.0
CompanyId	114635.0	450.641532	409.533093	4.0	86.0	297.0	798.00	1391.0
PrimaryCategoryId	114635.0	51.076294	20.474341	1.0	44.0	44.0	59.00	111.0
SubCategoryId	114635.0	66.819252	35.822097	3.0	48.0	52.0	65.00	172.0
CasId	114635.0	674.094107	149.214101	2.0	656.0	656.0	656.00	1242.0
ChemicalId	114635.0	32837.556959	20439.412299	0.0	13990.0	32055.0	51578.50	68074.0
ChemicalCount	114635.0	1.288359	0.636418	0.0	1.0	1.0	1.00	9.0

In [44]:
```
df.describe(include="object")# Summary statistics for categorical columns
```

Out[44]:

	ProductName	CSF	CompanyName	BrandName	PrimaryCategory	SubCategory	CasNumber	ChemicalName	InitialDateReported	MostRecentDateRepo
count	114635	80237	114635	114408	114635	114635	108159	114635	114635	114
unique	33716	34326	606	2713	13	89	125	123	2274	2
top	Eyecolor	Black	L'Oreal USA	SEPHORA	Makeup Products (non-permanent)	Lip Color - Lipsticks, Liners, and Pencils	13463-67-7	Titanium dioxide	10/13/2009	12/30/2
freq	766	247	5747	3394	75827	16555	93049	93480	2557	18

3.Data Cleaning

This step focuses on fixing issues in the data to prepare it for analysis. Common issues include missing values, duplicate rows, and incorrect data types.

1. Handling Missing Values Identify columns with missing values and decide how to handle them:
2. Drop rows/columns with too many missing values. Fill missing values with a default (e.g., mean, median, or mode) or forward/backward fill.
3. Remove Duplicates Duplicates can appear if the same product is reported multiple times with different chemical variations. Depending on your goal, you might need to either aggregate them or remove exact duplicates.

In [4]:
```
df.duplicated().sum()
```

Out[4]:
```
254
```

In [3]:
```
df.drop_duplicates()
```

Out[3]:

	CDPHId	ProductName	CSFId	CSF	CompanyId	CompanyName	BrandName	PrimaryCategoryId	PrimaryCategory	SubCategoryId	...	CasNumbe
0	2	ULTRA COLOR RICH EXTRA PLUMP LIPSTICK-ALL SHADES	NaN	NaN	4	New Avon LLC	AVON	44	Makeup Products (non-permanent)	53	...	13463-67-
1	3	Glover's Medicated Shampoo	NaN	NaN	338	J. Strickland & Co.	Glover's	18	Hair Care Products (non-coloring)	25	...	65996-92-
2	3	Glover's Medicated Shampoo	NaN	NaN	338	J. Strickland & Co.	Glover's	18	Hair Care Products (non-coloring)	25	...	140-67-
3	4	PRECISION GLIMMER EYE LINER-ALL SHADES ◆	NaN	NaN	4	New Avon LLC	AVON	44	Makeup Products (non-permanent)	46	...	13463-67-
4	5	AVON BRILLIANT SHINE LIP GLOSS-ALL SHADES ◆	NaN	NaN	4	New Avon LLC	AVON	44	Makeup Products (non-permanent)	52	...	13463-67-
...	...	...	...	...	...	...	...	...	...	...	...	.
114630	41523	HYDRA-LIP TRANSLUCENT COLOR LIPSTICK	65001.0	Rosa Soft	1259	Yanbal USA, Inc	YANBAL	44	Makeup Products (non-permanent)	53	...	13463-67-
114631	41523	HYDRA-LIP TRANSLUCENT COLOR LIPSTICK	65002.0	Malva Spirit	1259	Yanbal USA, Inc	YANBAL	44	Makeup Products (non-permanent)	53	...	13463-67-
114632	41523	HYDRA-LIP TRANSLUCENT COLOR LIPSTICK	65003.0	Rojo Fashion	1259	Yanbal USA, Inc	YANBAL	44	Makeup Products (non-permanent)	53	...	13463-67-
114633	41523	HYDRA-LIP TRANSLUCENT COLOR LIPSTICK	65004.0	Terra Mystic	1259	Yanbal USA, Inc	YANBAL	44	Makeup Products (non-permanent)	53	...	13463-67-
114634	41524	OLD SPICE GENTLEMENS BLEND ALOE AND WILD SAGE ...	NaN	NaN	86	The Procter & Gamble Company	Old Spice	6	Bath Products	159	...	13463-67-

114381 rows × 22 columns

In [5]:
```
df.value_counts("ChemicalCount")# Target Columns
```

Out[5]:
```
ChemicalCount
1    87267
2    21266
3     3528
4     1481
0      869
5      105
8       41
7       36
6       33
9        9
Name: count, dtype: int64
```

In [4]:
```
df.isnull().sum()
```

Out[4]:
```
CDPHId                      0
ProductName                 0
CSFId                   33973
CSF                     34398
CompanyId                   0
CompanyName                 0
BrandName                 227
PrimaryCategoryId           0
PrimaryCategory             0
SubCategoryId               0
SubCategory                 0
CasId                       0
CasNumber                6476
ChemicalId                  0
ChemicalName                0
InitialDateReported         0
MostRecentDateReported      0
DiscontinuedDate       101715
ChemicalCreatedAt           0
ChemicalUpdatedAt           0
ChemicalDateRemoved    111650
ChemicalCount               0
dtype: int64
```

In [4]:
```
df.drop(columns=["CDPHId","CSFId","CSF","DiscontinuedDate","ChemicalDateRemoved","CompanyId"],inplace=True,errors="ignore")
df
```

Out[4]:

	ProductName	CompanyName	BrandName	PrimaryCategoryId	PrimaryCategory	SubCategoryId	SubCategory	CasId	CasNumber	ChemicalId	Che
0	ULTRA COLOR RICH EXTRA PLUMP LIPSTICK-ALL SHADES	New Avon LLC	AVON	44	Makeup Products (non-permanent)	53	Lip Color - Lipsticks, Liners, and Pencils	656	13463-67-7	6	
1	Glover's Medicated Shampoo	J. Strickland & Co.	Glover's	18	Hair Care Products (non-coloring)	25	Hair Shampoos (making a cosmetic claim)	889	65996-92-1	4	Dis
2	Glover's Medicated Shampoo	J. Strickland & Co.	Glover's	18	Hair Care Products (non-coloring)	25	Hair Shampoos (making a cosmetic claim)	293	140-67-0	5	
3	PRECISION GLIMMER EYE LINER-ALL SHADES ♦	New Avon LLC	AVON	44	Makeup Products (non-permanent)	46	Eyeliner/Eyebrow Pencils	656	13463-67-7	7	
4	AVON BRILLIANT SHINE LIP GLOSS-ALL SHADES ♦	New Avon LLC	AVON	44	Makeup Products (non-permanent)	52	Lip Gloss/Shine	656	13463-67-7	8	
...	...	...	...	...	...	...	...	...	...	...	
114630	HYDRA-LIP TRANSLUCENT COLOR LIPSTICK	Yanbal USA, Inc	YANBAL	44	Makeup Products (non-permanent)	53	Lip Color - Lipsticks, Liners, and Pencils	656	13463-67-7	68059	
114631	HYDRA-LIP TRANSLUCENT COLOR LIPSTICK	Yanbal USA, Inc	YANBAL	44	Makeup Products (non-permanent)	53	Lip Color - Lipsticks, Liners, and Pencils	656	13463-67-7	68060	
114632	HYDRA-LIP TRANSLUCENT COLOR LIPSTICK	Yanbal USA, Inc	YANBAL	44	Makeup Products (non-permanent)	53	Lip Color - Lipsticks, Liners, and Pencils	656	13463-67-7	68061	
114633	HYDRA-LIP TRANSLUCENT COLOR LIPSTICK	Yanbal USA, Inc	YANBAL	44	Makeup Products (non-permanent)	53	Lip Color - Lipsticks, Liners, and Pencils	656	13463-67-7	68062	
114634	OLD SPICE GENTLEMENS BLEND ALOE AND WILD SAGE ...	The Procter & Gamble Company	Old Spice	6	Bath Products	159	Body Washes and Soaps	656	13463-67-7	68074	

114635 rows × 16 columns

In [ ]:
```
# Replacing null values with corresponding names
```

In [5]:
```
df["BrandName"].fillna("Merle Norman",inplace=True)
```

In [6]:
```
df['CasNumber'].fillna('Unknown',inplace=True)
```

In [7]:
```
df.isnull().sum()
```

Out[7]:
```
ProductName              0
CompanyName              0
BrandName                0
PrimaryCategoryId        0
PrimaryCategory          0
SubCategoryId            0
SubCategory              0
CasId                    0
CasNumber                0
ChemicalId               0
ChemicalName             0
InitialDateReported      0
MostRecentDateReported   0
ChemicalCreatedAt        0
ChemicalUpdatedAt        0
ChemicalCount            0
dtype: int64
```

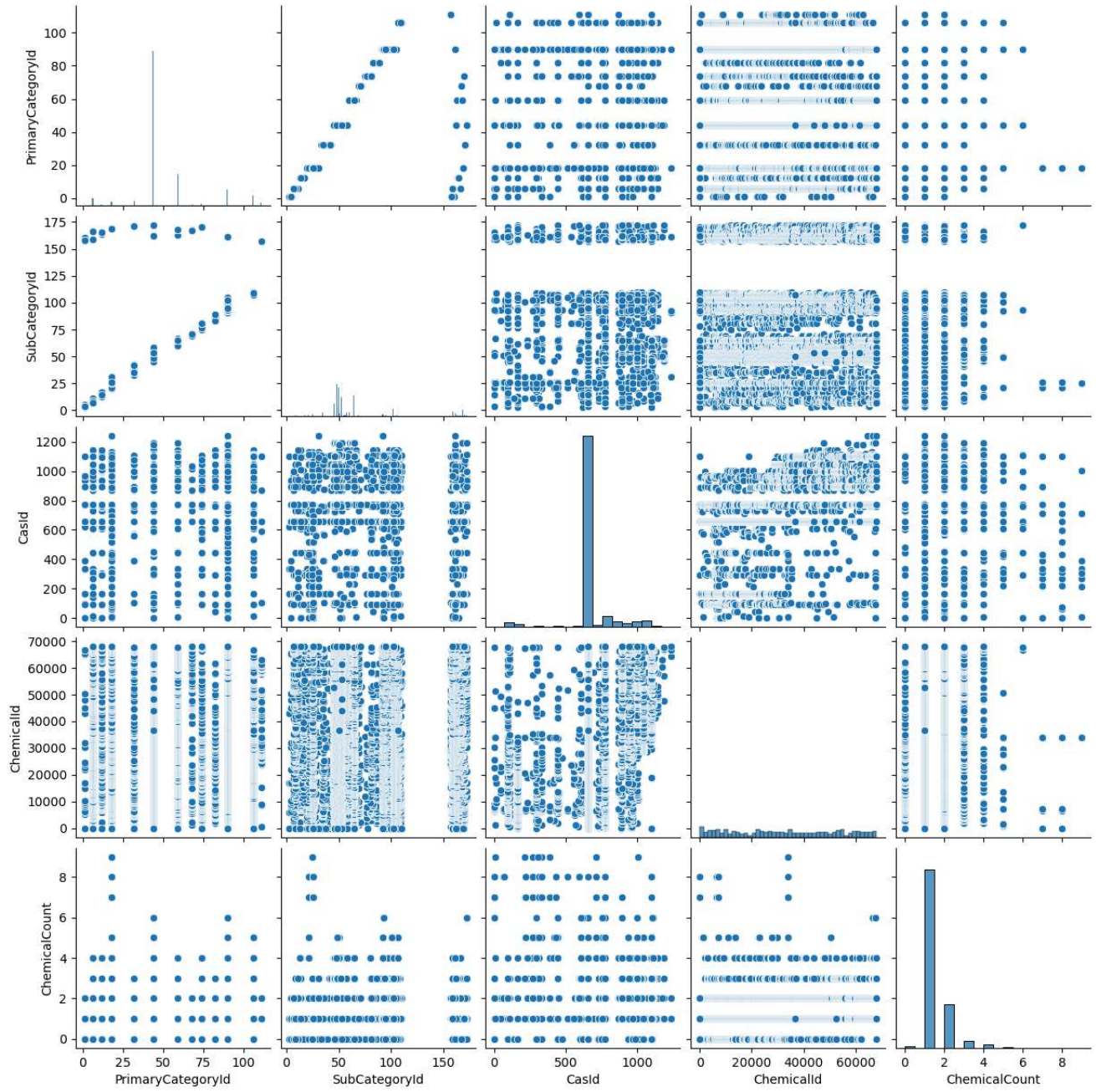
In [10]:
```
date_columns = ['InitialDateReported','MostRecentDateReported','ChemicalCreatedAt','ChemicalUpdatedAt']
for col in date_columns:
    df[col] = pd.to_datetime(df[col])
df["MostRecentDateReported"]
```

Out[10]:
```
0        2013-08-28
1        2009-07-01
2        2009-07-01
3        2013-08-28
4        2013-08-28
            ...
114630   2020-06-19
114631   2020-06-19
114632   2020-06-19
114633   2020-06-19
114634   2020-06-23
Name: MostRecentDateReported, Length: 114635, dtype: datetime64[ns]
```

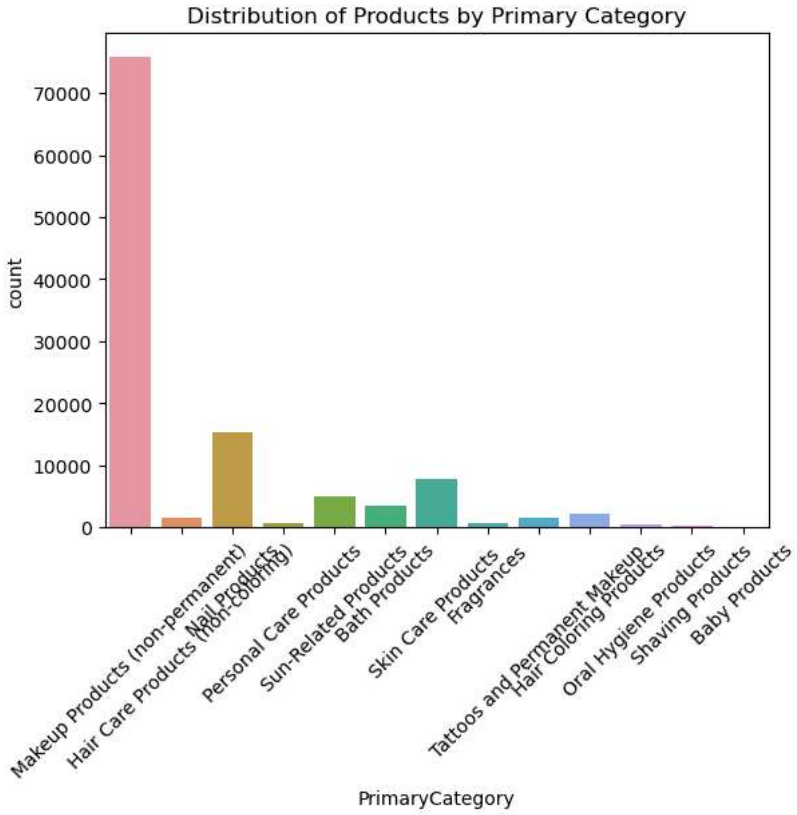
4.Exploratory Data Analysis

Once the data is cleaned and aggregated, the next step is to conduct an Exploratory Data Analysis to find patterns, trends, and relationships in the data.

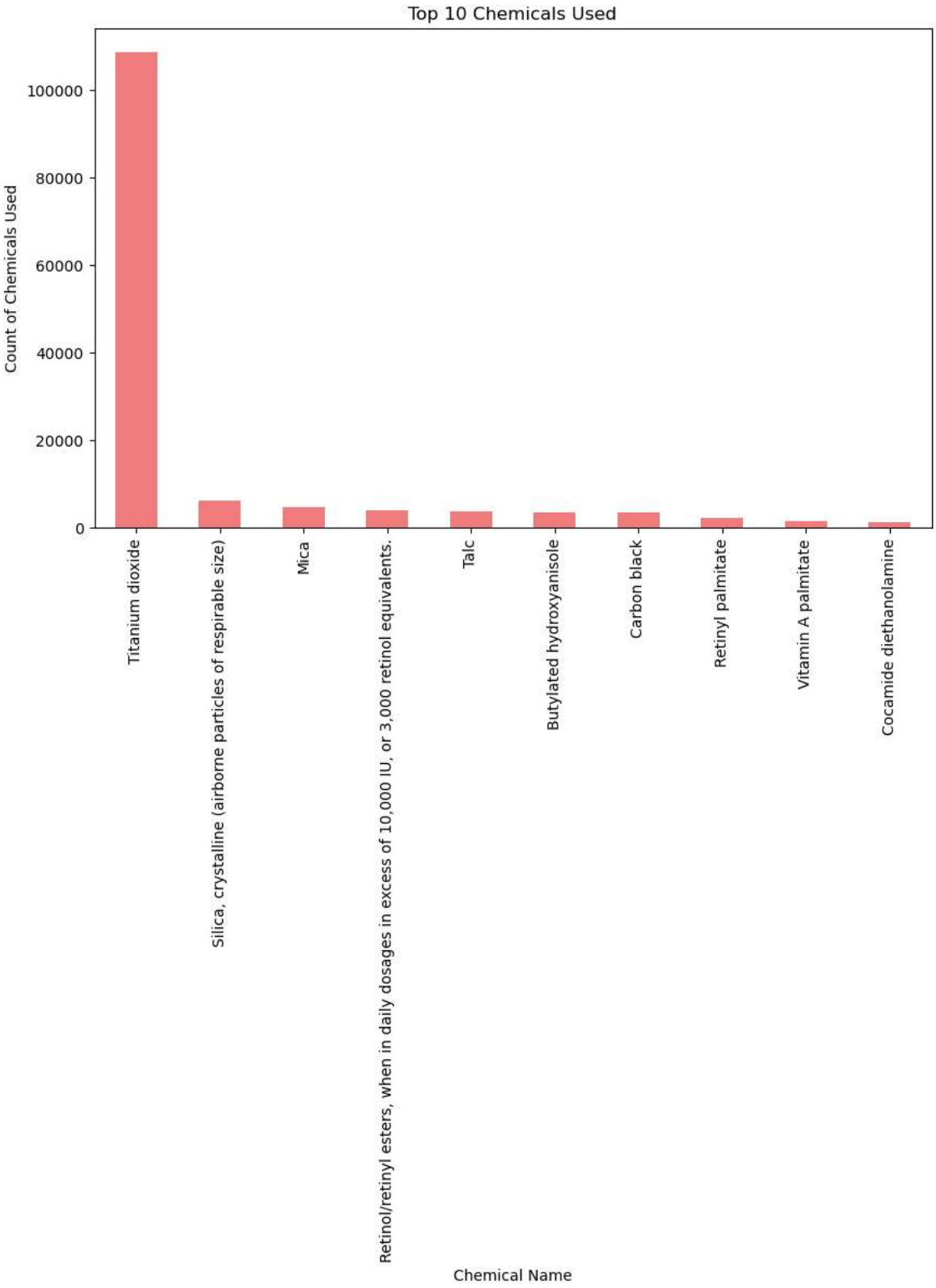
In [81]:
```
sns.pairplot(df)
plt.show()
```



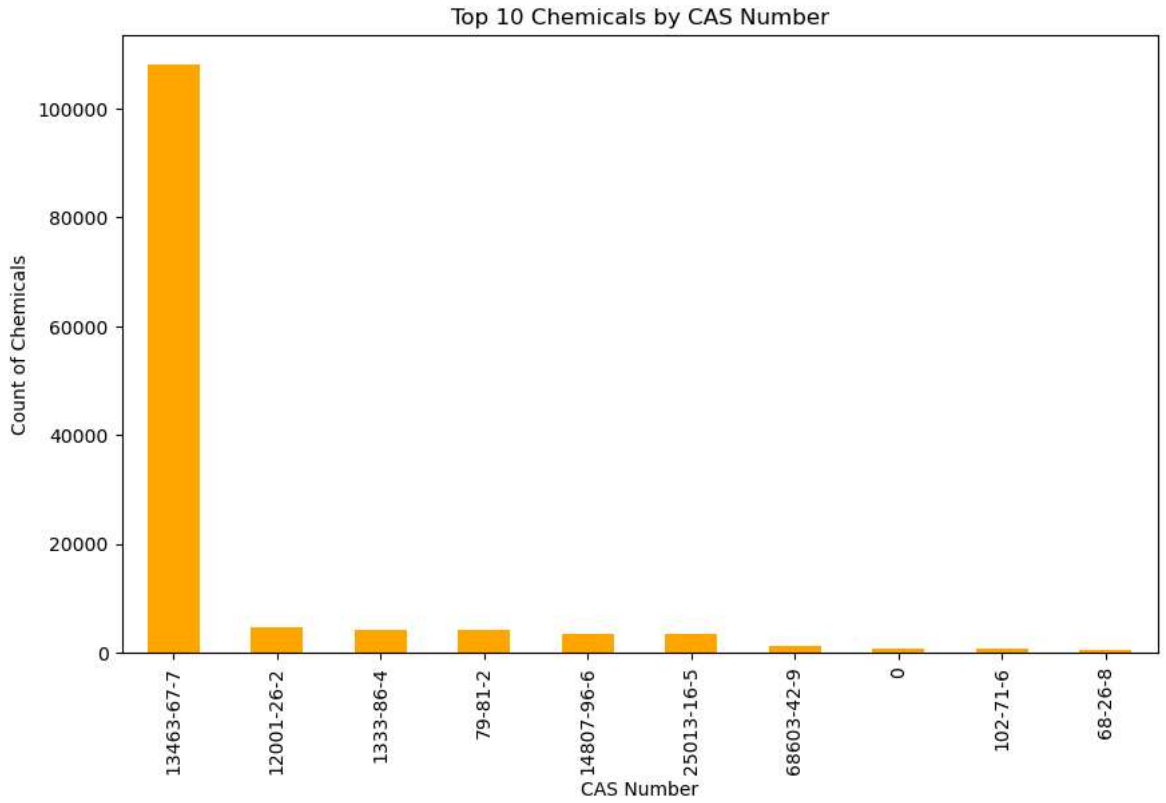
In [33]:
```
sns.countplot(x='PrimaryCategory', data=df)
plt.xticks(rotation=45)
plt.title('Distribution of Products by Primary Category')
plt.show()
```



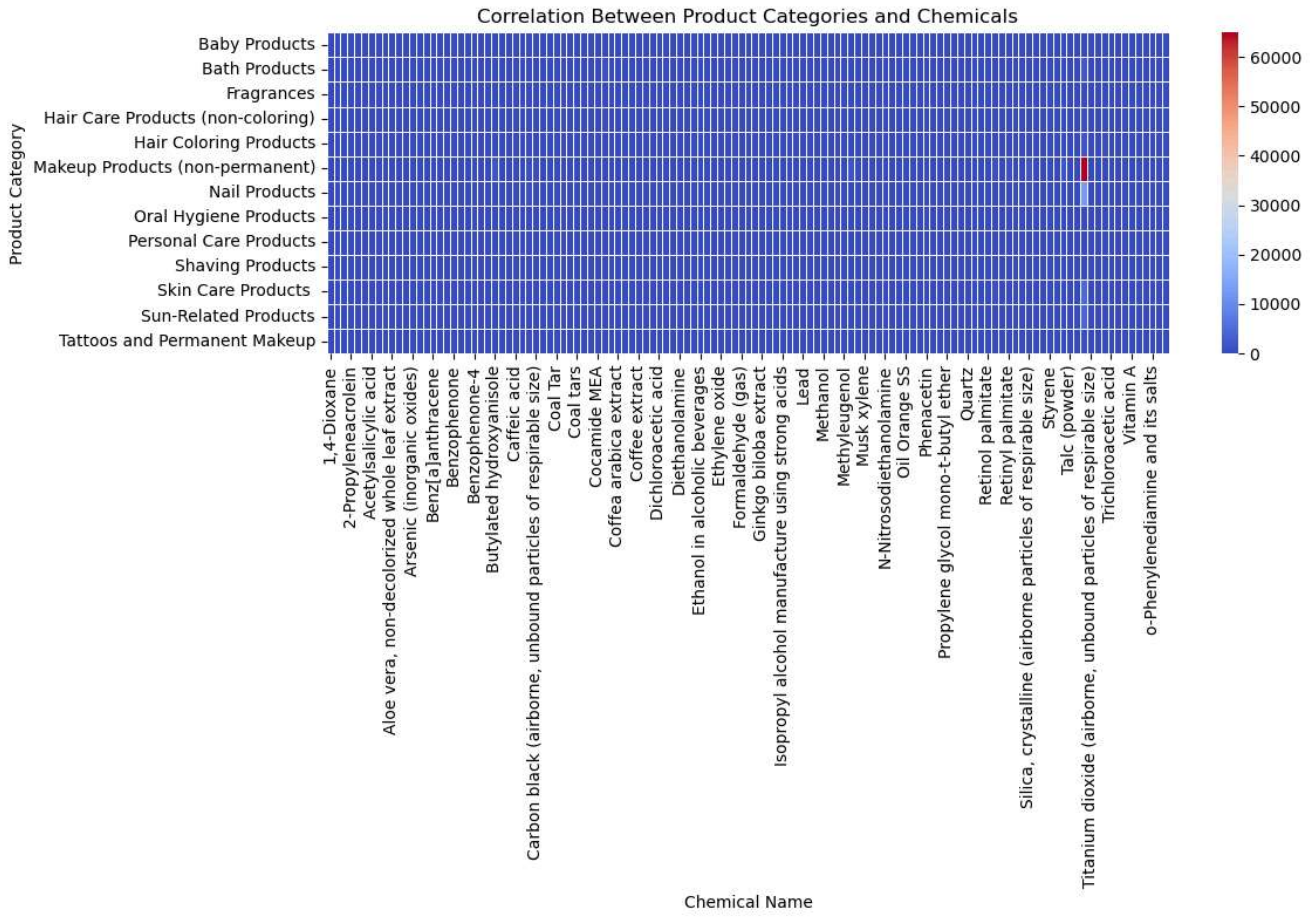
In [3]:
```
plt.figure(figsize=(10, 6))
chemical_counts = df.groupby('ChemicalName')['ChemicalCount'].sum().sort_values(ascending=False).head(10)
chemical_counts.plot(kind='bar', color='lightcoral')
plt.title('Top 10 Chemicals Used')
plt.xlabel('Chemical Name')
plt.ylabel('Count of Chemicals Used')
plt.show()
```



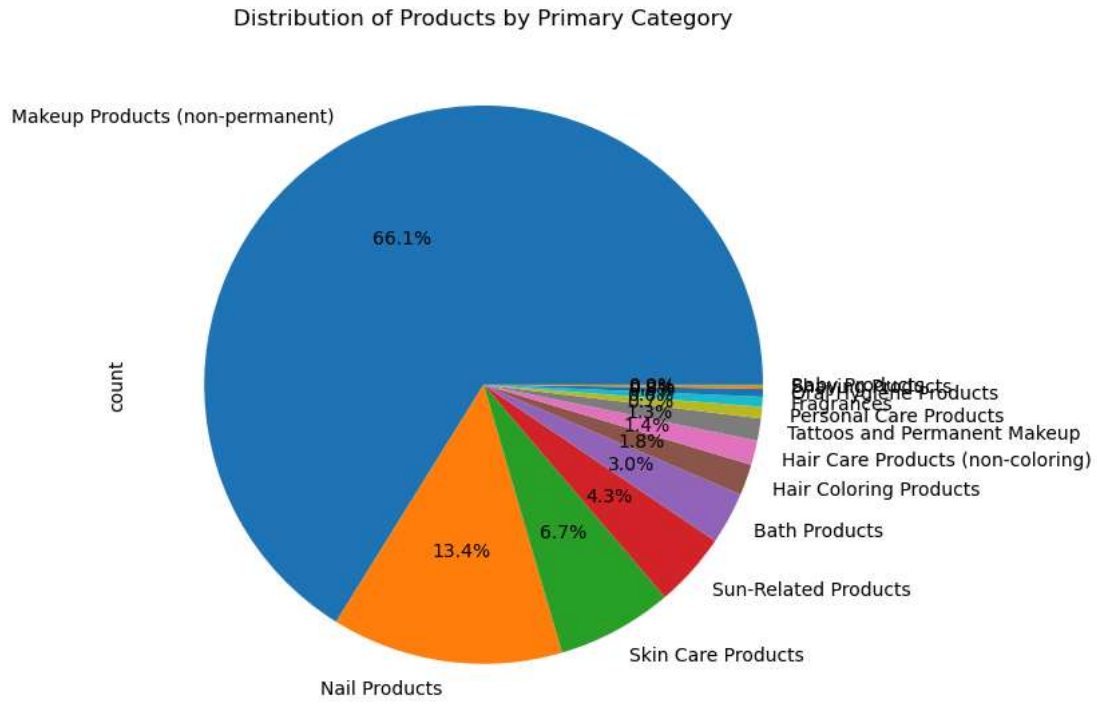
In [4]:
```
plt.figure(figsize=(10, 6))
cas_counts = df.groupby('CasNumber')['ChemicalCount'].sum().sort_values(ascending=False).head(10)
cas_counts.plot(kind='bar', color='orange')
plt.title('Top 10 Chemicals by CAS Number')
plt.xlabel('CAS Number')
plt.ylabel('Count of Chemicals')
plt.show()
```



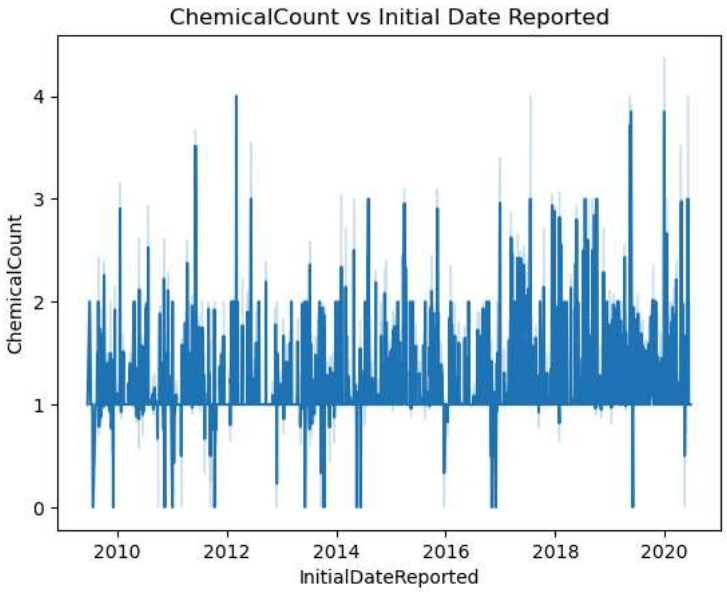
In [6]:
```
category_chemical_counts = df.groupby(['PrimaryCategory', 'ChemicalName']).size().unstack(fill_value=0)
plt.figure(figsize=(12, 8))
sns.heatmap(category_chemical_counts, cmap="coolwarm", annot=False, fmt="d", linewidths=0.5)
plt.title('Correlation Between Product Categories and Chemicals')
plt.xlabel('Chemical Name')
plt.ylabel('Product Category')
plt.tight_layout()
plt.show()
```



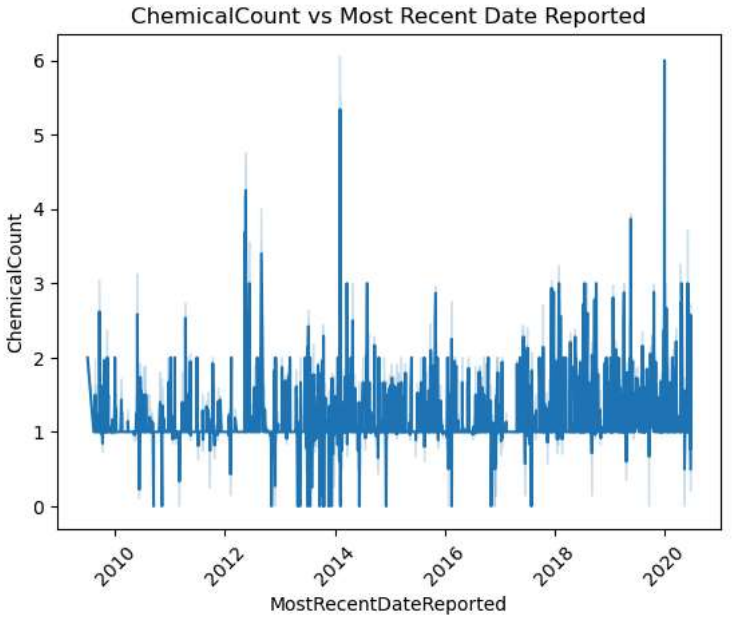
In [104]:
```
plt.figure(figsize=(8,8))
df['PrimaryCategory'].value_counts().plot(kind='pie', autopct='%1.1f%%')
plt.title('Distribution of Products by Primary Category')
plt.tight_layout()
plt.show()
```



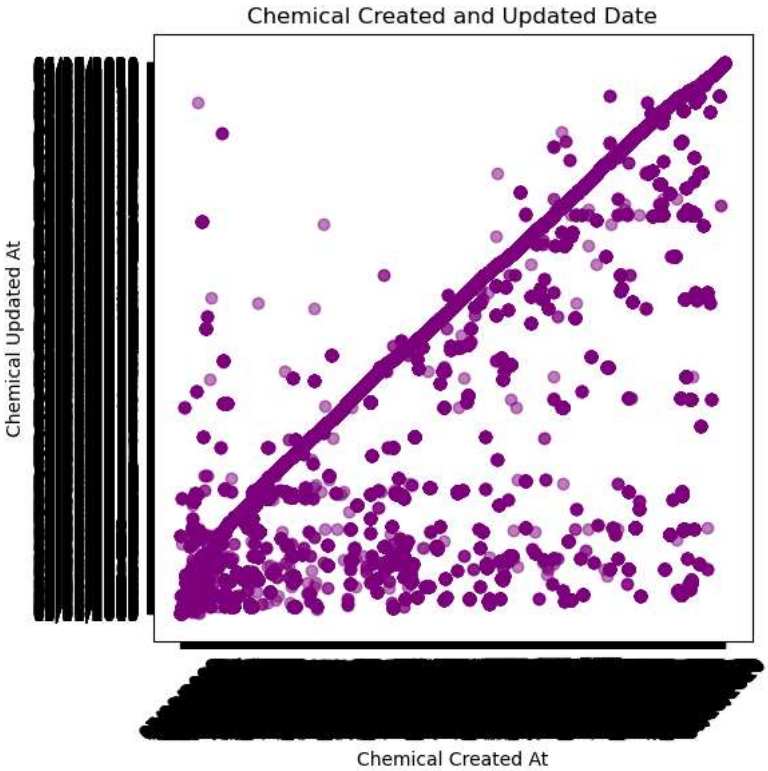
In [84]:
```
sns.lineplot(x='InitialDateReported', y='ChemicalCount', data=df)
plt.title('ChemicalCount vs Initial Date Reported')
plt.show()
```



In [86]:
```
sns.lineplot(x='MostRecentDateReported', y='ChemicalCount', data=df)
plt.title('ChemicalCount vs Most Recent Date Reported')
plt.xticks(rotation=45)
plt.show()
```



In [17]:
```
plt.figure(figsize=(6, 6))
plt.scatter(df['ChemicalCreatedAt'], df['ChemicalUpdatedAt'], alpha=0.5, c='purple', marker='o')
plt.title('Chemical Created and Updated Date')
plt.xlabel('Chemical Created At')
plt.ylabel('Chemical Updated At')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```



In [98]:
```
df.select_dtypes(include="number").columns
```

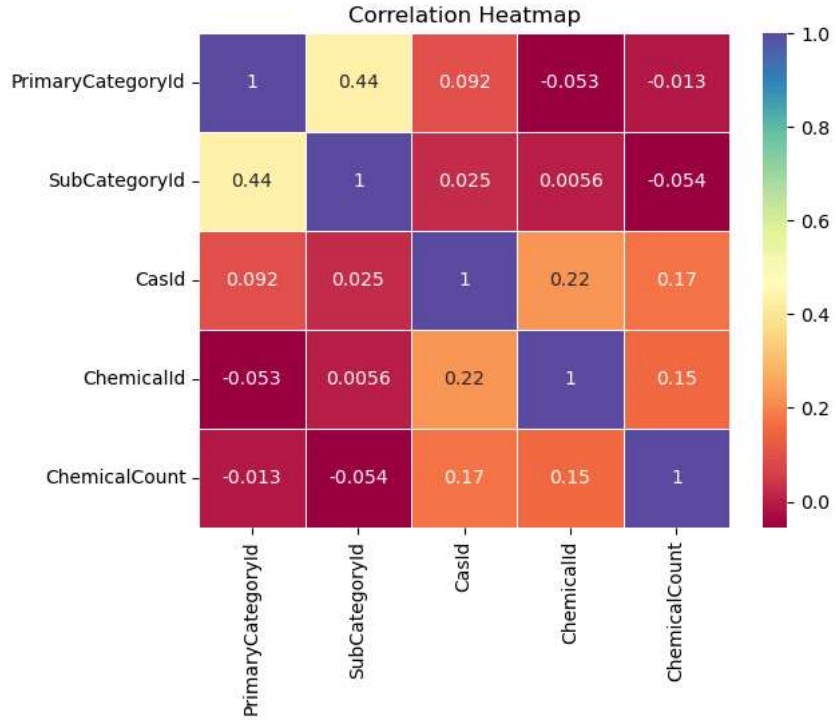
Out[98]:
```
Index(['PrimaryCategoryId', 'SubCategoryId', 'CasId', 'ChemicalId',
       'ChemicalCount'],
      dtype='object')
```

In [97]:
```
for i in df.select_dtypes(include="number").columns:
    sns.histplot(data=df,x=i)
    plt.show()
```

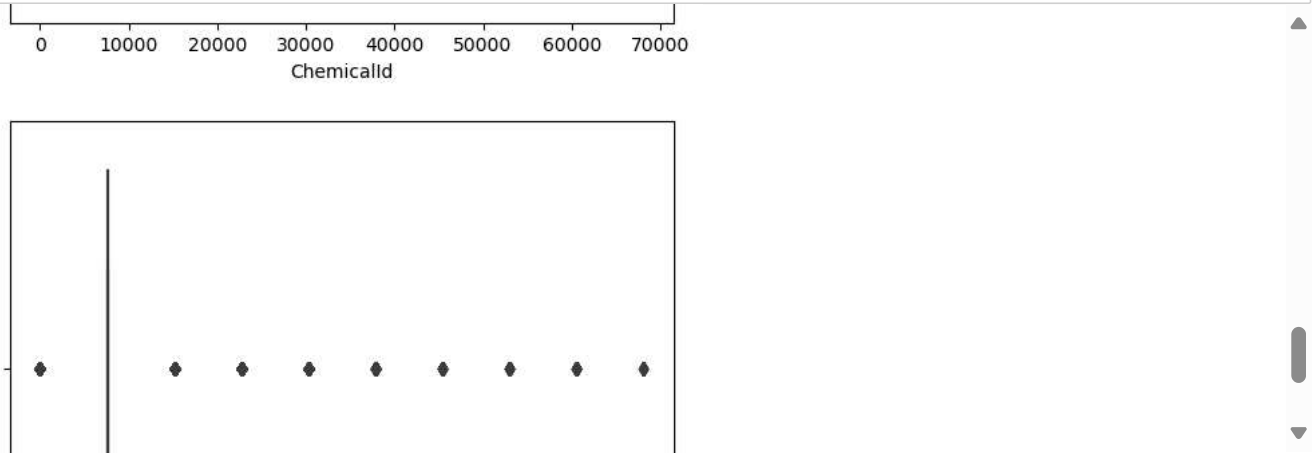
In [166]:
```
s=df.select_dtypes(include="number").corr()["ChemicalCount"]
s
```

Out[166]:
```
PrimaryCategoryId   -0.013065
SubCategoryId       -0.054375
CasId                0.170746
ChemicalId           0.154053
ChemicalCount        1.000000
Name: ChemicalCount, dtype: float64
```

In [102]:
```
s=df.select_dtypes(include="number").corr()
sns.heatmap(s,annot=True, cmap='Spectral', linewidths=0.5)
plt.title('Correlation Heatmap')
plt.show()
```



In [105]:
```
for i in df.select_dtypes(include="number").columns:
    sns.boxplot(data=df,x=i)
    plt.show()
```



In [ ]:
```
# There are outliers, but they have to be preserved, since the there are many rows and values  start from 1 and that cause a chan
```

In [129]:
```
df.select_dtypes(include=['object']).columns
```

Out[129]:
```
Index(['ProductName', 'CompanyName', 'BrandName', 'PrimaryCategory',
       'SubCategory', 'CasNumber', 'ChemicalName'],
      dtype='object')
```

In [51]:
```
df.select_dtypes(include="number").columns
```

Out[51]:
```
Index(['PrimaryCategoryId', 'SubCategoryId', 'CasId', 'ChemicalId',
       'ChemicalCount'],
      dtype='object')
```

5.Encoding

Label Encoding transforms categorical variables into numeric format by assigning each category a unique integer. For example, if you have a column like BrandName with categories ["AVON", "L'Oreal", "Maybelline"], LabelEncoder will assign integers such as: AVON = 0 L'Oreal = 1 Maybelline = 2 Label encoding is typically used for ordinal categorical variables, where the categories have a meaningful order (like Low, Medium, High), but it can also be applied to nominal variables (like product names, company names) as long as you don't need to capture the relationship between categories in the encoding.

In [7]:
```
encoder=LabelEncoder()
```

In [8]:
```
data=df.copy()
data['ProductName']=encoder.fit_transform(data['ProductName'])
data['CompanyName']=encoder.fit_transform(data['CompanyName'])
data['BrandName']=encoder.fit_transform(data['BrandName'])
data['PrimaryCategory']=encoder.fit_transform(data['PrimaryCategory'])
data['SubCategory']=encoder.fit_transform(data['SubCategory'])
data['ChemicalName']=encoder.fit_transform(data['ChemicalName'])
data['CasNumber']=encoder.fit_transform(data['CasNumber'])
```

In [9]:
```
data.drop(columns=['InitialDateReported','MostRecentDateReported','ChemicalCreatedAt','ChemicalUpdatedAt',"ChemicalId",'SubCatego
data
```

Out[9]:

	ProductName	CompanyName	BrandName	PrimaryCategoryId	PrimaryCategory	SubCategory	CasId	CasNumber	ChemicalName	ChemicalCount
0	30981	372	82	44	5	40	656	30	110	1
1	13104	252	1023	18	3	32	889	69	52	2
2	13104	252	1023	18	3	32	293	35	53	2
3	22843	372	82	44	5	19	656	30	110	1
4	1328	372	82	44	5	41	656	30	110	1
...	...	...	...	...	...	...	...	...	...	...
114630	13606	579	2578	44	5	40	656	30	110	1
114631	13606	579	2578	44	5	40	656	30	110	1
114632	13606	579	2578	44	5	40	656	30	110	1
114633	13606	579	2578	44	5	40	656	30	110	1
114634	21965	527	1729	6	1	11	656	30	110	1

114635 rows × 10 columns

In [ ]:
```
# converting the target columns to a new column name chemical_intesnsity, where chemical count is more than 6 its treated as high
#and within 3 is treated as low
```

In [10]:
```
def classify_chemical_count(count):
    if count > 6:
        return "High"
    elif count > 3:
        return "Medium"
    else:
        return "Low"
data['chemical_intensity'] = data['ChemicalCount'].apply(classify_chemical_count)
print(data)
```

```
        ProductName  CompanyName  BrandName  PrimaryCategoryId  \
0             30981          372         82                 44
1             13104          252       1023                 18
2             13104          252       1023                 18
3             22843          372         82                 44
4              1328          372         82                 44
...             ...          ...        ...                ...
114630        13606          579       2578                 44
114631        13606          579       2578                 44
114632        13606          579       2578                 44
114633        13606          579       2578                 44
114634        21965          527       1729                  6

        PrimaryCategory  SubCategory  CasId  CasNumber  ChemicalName  \
0                     5           40    656         30           110
1                     3           32    889         69            52
2                     3           32    293         35            53
3                     5           19    656         30           110
4                     5           41    656         30           110
...                 ...          ...    ...        ...           ...
114630                5           40    656         30           110
114631                5           40    656         30           110
114632                5           40    656         30           110
114633                5           40    656         30           110
114634                1           11    656         30           110

        ChemicalCount chemical_intensity
0                   1               Low
1                   2               Low
2                   2               Low
3                   1               Low
4                   1               Low
...               ...               ...
114630              1               Low
114631              1               Low
114632              1               Low
114633              1               Low
114634              1               Low

[114635 rows x 11 columns]
```

In [14]:
```
data.drop(["ChemicalCount"],axis=1,inplace=True)
```

In [15]:
```
data
```

Out[15]:

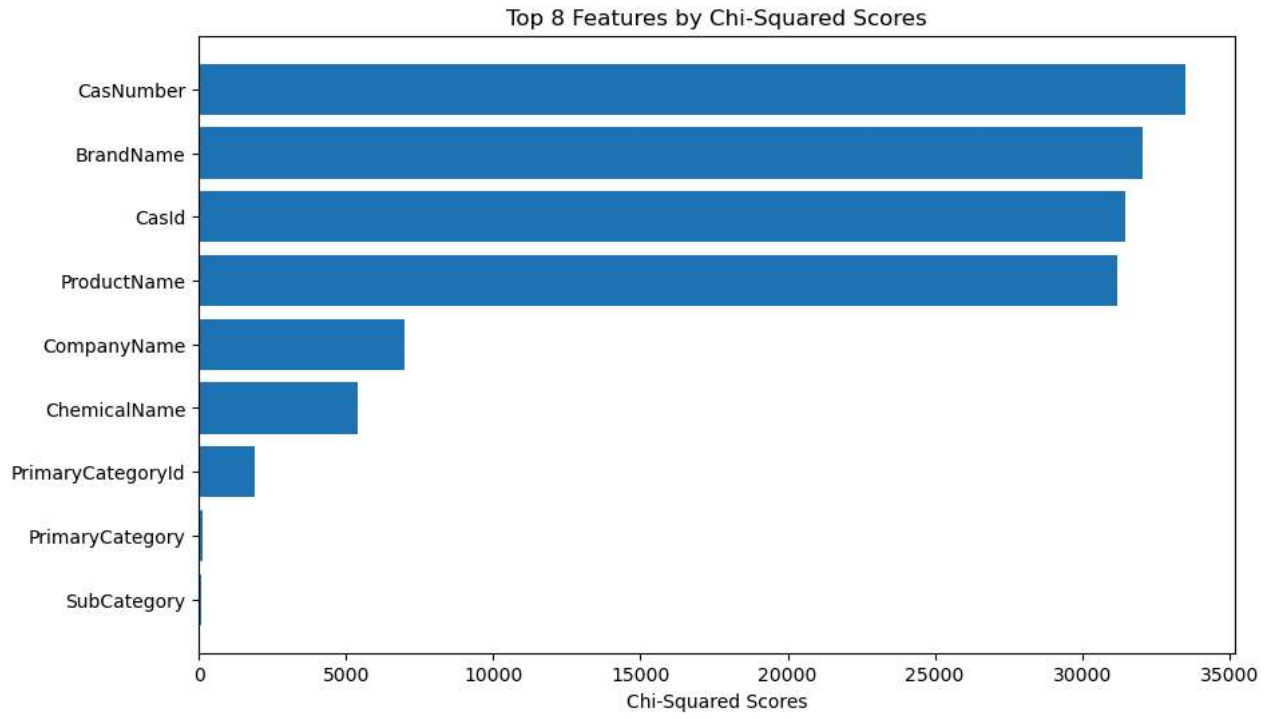
	ProductName	CompanyName	BrandName	PrimaryCategoryId	PrimaryCategory	SubCategory	CasId	CasNumber	ChemicalName	chemical_intensity
0	30981	372	82	44	5	40	656	30	110	Low
1	13104	252	1023	18	3	32	889	69	52	Low
2	13104	252	1023	18	3	32	293	35	53	Low
3	22843	372	82	44	5	19	656	30	110	Low
4	1328	372	82	44	5	41	656	30	110	Low
...	...	...	...	...	...	...	...	...	...	...
114630	13606	579	2578	44	5	40	656	30	110	Low
114631	13606	579	2578	44	5	40	656	30	110	Low
114632	13606	579	2578	44	5	40	656	30	110	Low
114633	13606	579	2578	44	5	40	656	30	110	Low
114634	21965	527	1729	6	1	11	656	30	110	Low

114635 rows × 10 columns

6.Selecting best features

In [16]:
```
x = data.drop(columns=['chemical_intensity'])
y = data["chemical_intensity"]
selector = SelectKBest(score_func=chi2, k=9)
selector.fit_transform(x, y)
datascores=pd.DataFrame(selector.scores_,columns=["scores"])
datacolumns=pd.DataFrame(x.columns.tolist(),columns=["features"])
result=pd.concat([datacolumns,datascores,],axis=1)
sorted_result=result.sort_values(by="scores", ascending=False)
```

In [22]:
```
plt.figure(figsize=(10, 6))
plt.barh(sorted_result['features'][:9], sorted_result['scores'][:9])
plt.xlabel('Chi-Squared Scores')
plt.title('Top 8 Features by Chi-Squared Scores')
plt.gca().invert_yaxis()  # Reverse the order for better readability
plt.show()
```



7.Spliting the Dataset

Data need to be split for both training and testing purposes, here 80% data is used for training purpose and 20% data is used for testing purpose.

In [17]:
```
x=data.drop(columns="chemical_intensity")
y=data["chemical_intensity"]
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,random_state=2)
```

In [24]:
```
print(x)
```

```
        ProductName  CompanyName  BrandName  PrimaryCategoryId  \
0             30981          372         82                 44
1             13104          252       1023                 18
2             13104          252       1023                 18
3             22843          372         82                 44
4              1328          372         82                 44
...             ...          ...        ...                ...
114630        13606          579       2578                 44
114631        13606          579       2578                 44
114632        13606          579       2578                 44
114633        13606          579       2578                 44
114634        21965          527       1729                  6

        PrimaryCategory  SubCategory  CasId  CasNumber  ChemicalName
0                     5           40    656         30           110
1                     3           32    889         69            52
2                     3           32    293         35            53
3                     5           19    656         30           110
4                     5           41    656         30           110
...                 ...          ...    ...        ...           ...
114630                5           40    656         30           110
114631                5           40    656         30           110
114632                5           40    656         30           110
114633                5           40    656         30           110
114634                1           11    656         30           110

[114635 rows x 9 columns]
```

8. Scaling

Scaling in the context of machine learning refers to the process of adjusting the range of feature values in your dataset so that they are comparable.one of the most commonly used scaler is standard scaler. Standardization transforms the data to have a mean of 0 and a standard deviation of 1.

In [18]:
```
scaler = StandardScaler()
```

In [19]:
```
x_train_scaled = scaler.fit_transform(x_train)
x_test_scaled = scaler.transform(x_test)
```

8. Smote Analysis

SMOTE is an advanced technique used to handle the issue of class imbalance in machine learning datasets, particularly for binary classification problems. Class imbalance occurs when one class (usually the minority class) is underrepresented compared to the other (majority class), which can lead to poor model performance, as the classifier may become biased towards the majority class.

In [20]:
```
smote = SMOTE(sampling_strategy='auto', random_state=42)
```

In [21]:
```
x_train_resampled, y_train_resampled = smote.fit_resample(x_train_scaled, y_train)
```

In [30]:
```
print("Original y_train class distribution:")
print(y_train.value_counts())
```

```
Original y_train class distribution:
chemical_intensity
Low       90348
Medium     1284
High         76
Name: count, dtype: int64
```

In [29]:
```
print("\nResampled y_train class distribution:")
print(y_train_resampled.value_counts())
```

```
Resampled y_train class distribution:
chemical_intensity
Low       90348
Medium    90348
High      90348
Name: count, dtype: int64
```

1. Logistic Regression

Linear Regression is a statistical method and a supervised machine learning algorithm used to model the relationship between a dependent (target) variable and one or more independent (predictor) variables. It assumes that the relationship between the variables can be represented as a straight line.

In [31]:
```
model = LogisticRegression()
```

In [32]:
```
model.fit(x_train_resampled, y_train_resampled)
```

Out[32]:
```
LogisticRegression()
```
In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [ ]:
```
# Accuracy on training data
```

In [33]:
```
y_train_pred = model.predict(x_train_resampled)
y_test_pred = model.predict(x_test_scaled)
```

In [34]:
```
train_accuracy = accuracy_score(y_train_resampled, y_train_pred)
test_accuracy = accuracy_score(y_test, y_test_pred)
print(f"Training Data Accuracy: {train_accuracy:.4f}")
print(f"Test Data Accuracy: {test_accuracy:.4f}")
```

```
Training Data Accuracy: 0.8028
Test Data Accuracy: 0.8483
```

In [35]:
```
precision = precision_score(y_test, y_test_pred, average='weighted')
recall = recall_score(y_test, y_test_pred, average='weighted')
f1 = f1_score(y_test, y_test_pred, average='weighted')
```

In [36]:
```
print(f"Precision: {precision:.4f}")
print(f"Recall: {recall:.4f}")
print(f"F1-Score: {f1:.4f}")
```

```
Precision: 0.9798
Recall: 0.8483
F1-Score: 0.9050
```

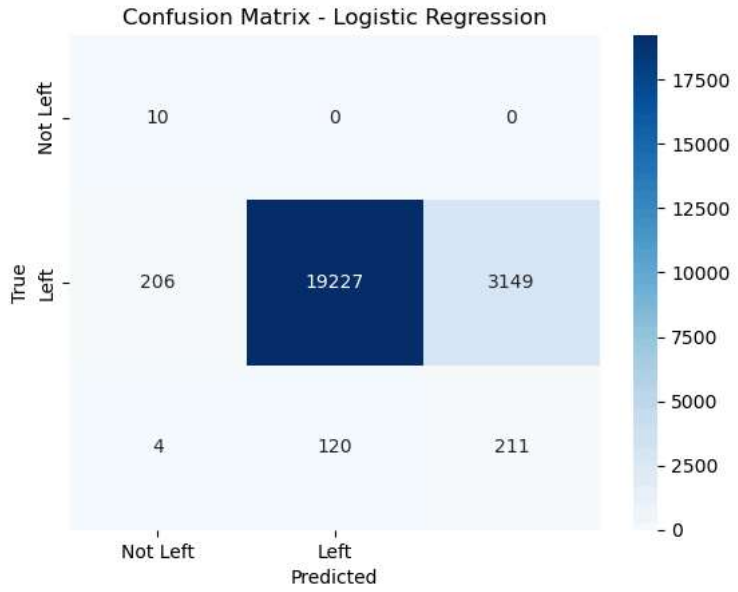
In [37]:
```
cm = confusion_matrix(y_test, y_test_pred)
print("Confusion Matrix:\n", cm)
```

```
Confusion Matrix:
 [[   10     0     0]
 [  206 19227  3149]
 [    4   120   211]]
```

Matrix Explanation:

The matrix is a 10x10 grid, likely indicating a multi-class classification problem where there are 10 classes (possibly 10 labels or categories). Each row represents the true class. Each column represents the predicted class True Positives (TP) are located on the diagonal of the matrix. These represent the correctly predicted instances for each class. For instance, the value at position (0, 0) is 64, meaning that 64 instances from class 0 were correctly classified as class 0. False Positives (FP) are the values off the diagonal, where the model predicted a class incorrectly. For example: The value at position (0, 1) is 20, which means that 20 instances from class 0 were misclassified as class 1. False Negatives (FN) are the values where the true class is misclassified by the model into another class. For example: The value at position (1, 0) is 359, which means that 359 instances from class 1 were misclassified as class 0. Overall Performance: The diagonal elements represent the correctly predicted instances, and the non-diagonal elements represent misclassifications. T

In [38]:
```
# Visualize Confusion Matrix
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=['Not Left', 'Left'], yticklabels=['Not Left', 'Left'])
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix - Logistic Regression')
plt.show()
```



2. Random Forest Classifier

Random Forest is an ensemble learning method that combines multiple decision trees to improve classification accuracy and robustness. It is widely used for both classification and regression tasks.t combines the predictions of several individual models (decision trees) to make a more accurate overall prediction. This helps to reduce overfitting and increases the model's generalization ability.

In [22]:
```
rf_model = RandomForestClassifier(random_state=42)
rf_model.fit(x_train_resampled, y_train_resampled)
```

Out[22]:
```
RandomForestClassifier(random_state=42)
```
In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [23]:
```
y_train_pred_rf = rf_model.predict(x_train_resampled)
y_test_pred_rf = rf_model.predict(x_test_scaled)
```

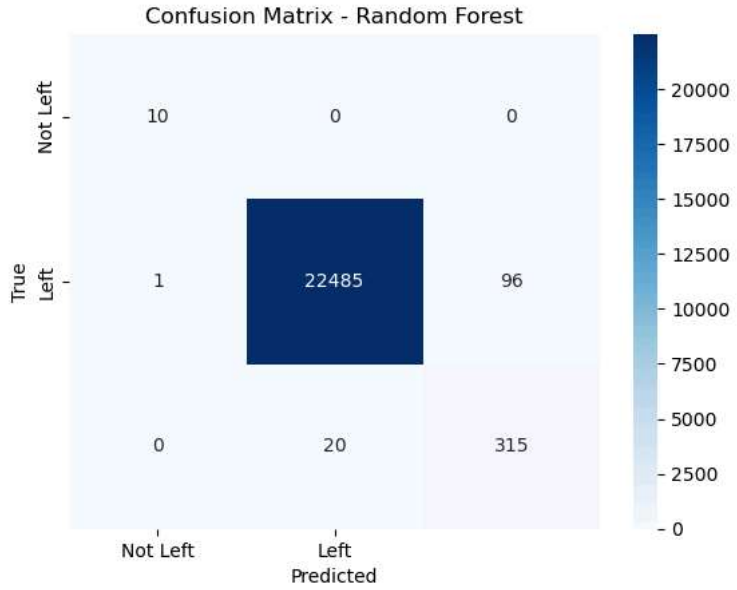
In [24]:
```
train_accuracy_rf = accuracy_score(y_train_resampled, y_train_pred_rf)
test_accuracy_rf = accuracy_score(y_test, y_test_pred_rf)
precision_rf = precision_score(y_test, y_test_pred_rf, average='weighted')
recall_rf = recall_score(y_test, y_test_pred_rf, average='weighted')
f1_rf = f1_score(y_test, y_test_pred_rf, average='weighted')
```

In [25]:
```
print(f"Random Forest - Training Accuracy: {train_accuracy_rf:.4f}")
print(f"Random Forest - Test Accuracy: {test_accuracy_rf:.4f}")
print(f"Random Forest - Precision (weighted): {precision_rf:.4f}")
print(f"Random Forest - Recall (weighted): {recall_rf:.4f}")
print(f"Random Forest - F1 Score (weighted): {f1_rf:.4f}")
```

```
Random Forest - Training Accuracy: 0.9986
Random Forest - Test Accuracy: 0.9949
Random Forest - Precision (weighted): 0.9957
Random Forest - Recall (weighted): 0.9949
Random Forest - F1 Score (weighted): 0.9952
```

In [26]:
```
cm_rf = confusion_matrix(y_test,y_test_pred_rf)
```

In [27]:
```
sns.heatmap(cm_rf, annot=True, fmt='d', cmap='Blues', xticklabels=['Not Left', 'Left'], yticklabels=['Not Left', 'Left'])
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix - Random Forest')
plt.show()
```



3.K-Nearest Neighbors (KNN)

KNN is an instance-based learning algorithm, meaning that it doesn't build an explicit model or function during training. Instead, it memorizes the training data and makes predictions based on how close new data points are to the stored data.The algorithm uses a parameter K which represents the number of nearest neighbors to consider when making predictions.

In [28]:
```
knn_model = KNeighborsClassifier(n_neighbors=5)
knn_model.fit(x_train_resampled, y_train_resampled)
```

Out[28]:
```
KNeighborsClassifier()
```
In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [29]:
```
y_train_pred_knn = knn_model.predict(x_train_resampled)
y_test_pred_knn = knn_model.predict(x_test_scaled)
```

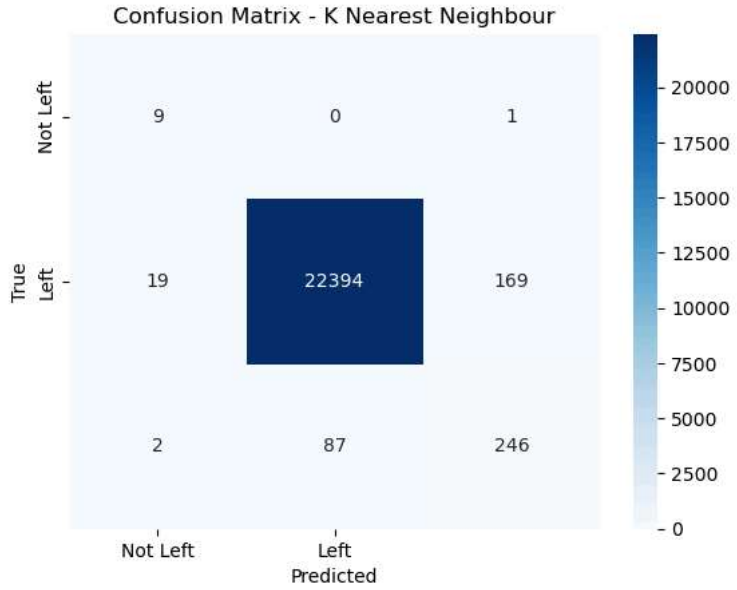
In [30]:
```
train_accuracy_knn = accuracy_score(y_train_resampled, y_train_pred_knn)
test_accuracy_knn = accuracy_score(y_test, y_test_pred_knn)
precision_knn = precision_score(y_test, y_test_pred_knn, average='weighted')
recall_knn = recall_score(y_test, y_test_pred_knn, average='weighted')
f1_knn = f1_score(y_test, y_test_pred_knn, average='weighted')
```

In [31]:
```
print(f"KNN - Training Accuracy: {train_accuracy_knn:.4f}")
print(f"KNN - Test Accuracy: {test_accuracy_knn:.4f}")
print(f"KNN - Precision (weighted): {precision_knn:.4f}")
print(f"KNN - Recall (weighted): {recall_knn:.4f}")
print(f"KNN - F1 Score (weighted): {f1_knn:.4f}")
```

```
KNN - Training Accuracy: 0.9396
KNN - Test Accuracy: 0.9879
KNN - Precision (weighted): 0.9899
KNN - Recall (weighted): 0.9879
KNN - F1 Score (weighted): 0.9887
```

In [32]:
```
cm_knn = confusion_matrix(y_test, y_test_pred_knn)
```

In [33]:
```
sns.heatmap(cm_knn, annot=True, fmt='d', cmap='Blues', xticklabels=['Not Left', 'Left'], yticklabels=['Not Left', 'Left'])
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix - K Nearest Neighbour')
plt.show()
```



4. Decision Tree Classifier

A Decision Tree Classifier is a supervised machine learning algorithm used for classification tasks. It works by splitting the dataset into subsets based on feature values, forming a tree-like structure where each internal node represents a decision based on a feature, and each leaf node represents a class label or outcome. It is a non-parametric, interpretable model that can handle both numerical and categorical data.

In [34]:
```
dt_model = DecisionTreeClassifier(random_state=42)
dt_model.fit(x_train_resampled, y_train_resampled)
```

Out[34]:
```
DecisionTreeClassifier(random_state=42)
```
In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [35]:
```
y_train_pred_dt = dt_model.predict(x_train_resampled)
y_test_pred_dt = dt_model.predict(x_test_scaled)
```

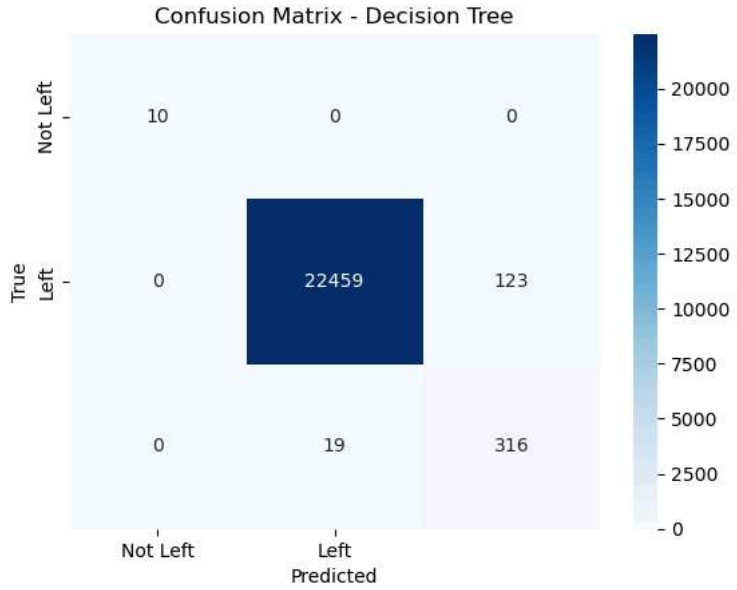
In [36]:
```
train_accuracy_dt = accuracy_score(y_train_resampled, y_train_pred_dt)
test_accuracy_dt = accuracy_score(y_test, y_test_pred_dt)
precision_dt = precision_score(y_test, y_test_pred_dt, average='weighted')
recall_dt = recall_score(y_test, y_test_pred_dt, average='weighted')
f1_dt = f1_score(y_test, y_test_pred_dt, average='weighted')
```

In [37]:
```
print(f"Decision Tree - Training Accuracy: {train_accuracy_dt:.4f}")
print(f"Decision Tree - Test Accuracy: {test_accuracy_dt:.4f}")
print(f"Decision Tree - Precision (weighted): {precision_dt:.4f}")
print(f"Decision Tree - Recall (weighted): {recall_dt:.4f}")
print(f"Decision Tree - F1 Score (weighted): {f1_dt:.4f}")
```

```
Decision Tree - Training Accuracy: 0.9986
Decision Tree - Test Accuracy: 0.9938
Decision Tree - Precision (weighted): 0.9951
Decision Tree - Recall (weighted): 0.9938
Decision Tree - F1 Score (weighted): 0.9942
```

In [38]:
```
cm_dt = confusion_matrix(y_test, y_test_pred_dt)
```

In [39]:
```
sns.heatmap(cm_dt, annot=True, fmt='d', cmap='Blues', xticklabels=['Not Left', 'Left'], yticklabels=['Not Left', 'Left'])
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix - Decision Tree')
plt.show()
```



5. Gradient Boosting Classifier

Gradient Boosting is an ensemble learning method used for classification and regression tasks. It builds a model in a stage-wise manner by combining the predictions of several weak models (typically decision trees) to create a strong model. The key idea behind gradient boosting is to iteratively train a series of models, each one correcting the errors (residuals) of the previous one, using a technique called gradient descent.Boosting is an ensemble method that combines multiple weak learners (typically shallow decision trees) to form a strong learner. The main idea is to give more weight to the misclassified instances in each iteration so that subsequent models focus on the hard-to-predict samples.

In [40]:
```
from sklearn.ensemble import GradientBoostingClassifier
```

In [41]:
```
gb_model = GradientBoostingClassifier(n_estimators=100, learning_rate=0.1, max_depth=3)
```

In [42]:
```
gb_model.fit(x_train_resampled, y_train_resampled)
```

Out[42]:
```
GradientBoostingClassifier()
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [43]:
```
y_train_pred_gb = gb_model.predict(x_train_resampled)
y_test_pred_gb = gb_model.predict(x_test_scaled)
```

In [44]:
```
train_accuracy_gb = accuracy_score(y_train_resampled, y_train_pred_gb)
test_accuracy_gb = accuracy_score(y_test, y_test_pred_gb)
```

In [45]:
```
precision_gb = precision_score(y_test, y_test_pred_gb, average='weighted')
recall_gb = recall_score(y_test, y_test_pred_gb, average='weighted')
f1_gb = f1_score(y_test, y_test_pred_gb, average='weighted')
```

In [46]:
```
print(f"Gradient Booster - Training Accuracy: {train_accuracy_gb:.4f}")
print(f"Gradient Booster - Test Accuracy: {test_accuracy_gb:.4f}")
print(f"Gradient Booster - Precision : {precision_gb:.4f}")
print(f"Gradient Booster - Recall : {recall_gb:.4f}")
print(f"Gradient Booster - F1 Score: {f1_gb:.4f}")
```

```
Gradient Booster - Training Accuracy: 0.9696
Gradient Booster - Test Accuracy: 0.9481
Gradient Booster - Precision : 0.9874
Gradient Booster - Recall : 0.9481
Gradient Booster - F1 Score: 0.9638
```

In [47]:
```
cm_gb = confusion_matrix(y_test, y_test_pred_gb)
```

In [48]:
```
sns.heatmap(cm_gb, annot=True, fmt='d', cmap='Blues', xticklabels=['Not Left', 'Left'], yticklabels=['Not Left', 'Left'])
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix - Gradient Booster')
plt.show()
```

CONCLUSION

In [ ]:
```
1)Gradient Boosting shows a strong performance, with high precision and a solid recall, indicating it balances both false positiv
 The relatively small gap between training and test accuracy suggests that the model is generalizing well to new data.
2)The Decision Tree performs exceptionally well, with near-perfect accuracy both on the training and test datasets.
The weighted precision, recall, and F1 scores further demonstrate its ability to classify the data effectively, though it may be
3)KNN shows a solid performance with high accuracy on the test set. The model has a good balance between precision, recall, and F
but it might not be as strong as the Decision Tree or Random Forest in terms of raw accuracy, though it's still highly effective
4)Forest achieves excellent performance, similar to the Decision Tree, but with slightly better generalization to the test set (1
The precision, recall, and F1 scores are very high, confirming the robustness and reliability of the model.
5)Linear regression shows decent precision, but its recall is relatively lower, meaning it might miss some positive cases.
The F1 score is reasonable, but it's evident that linear regression isn't as effective as the other classification algorithms, es
```

In [ ]:
```
Best Performers: The Decision Tree and Random Forest models perform the best in terms of overall accuracy and classification metr
Linear Regression is not as suited for this classification task, given the lower recall and F1 score, suggesting it's less effect
KNN offers a good balance of performance, though it lags slightly behind the top models in terms of raw accuracy.
Gradient Boosting also performs well, with competitive metrics, particularly in terms of precision.
```