REGRESSION ANALYSIS

CHEMICALS IN

COSMETICS

Importing Libraries and Loading the Dataset

In [2]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from pandas.plotting import table
import seaborn as sns

from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import OneHotEncoder
from sklearn.model_selection import train_test_split
from scipy import stats

from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, mean_absolute_error
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report, roc_auc_score, roc_
from sklearn.linear_model import Lasso, Ridge
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import Ridge
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import ElasticNet
from sklearn.model_selection import cross_val_score
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score



from sklearn.preprocessing import StandardScaler
from imblearn.over_sampling import SMOTE

import warnings
warnings.filterwarnings("ignore")


from sklearn.model_selection import GridSearchCV
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, classification_report
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler

from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from xgboost import XGBClassifier
from catboost import CatBoostClassifier, Pool, cv
from sklearn.ensemble import GradientBoostingClassifier
from lightgbm import LGBMClassifier

from sklearn.model_selection import GridSearchCV, cross_val_score
```

```
/opt/conda/lib/python3.10/site-packages/scipy/__init__.py:146: UserWarning: A NumPy v
ersion >=1.16.5 and <1.23.0 is required for this version of SciPy (detected version
1.24.3
  warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversion}"
```

In [3]:
```python
df = pd.read_csv('/kaggle/input/chemicals-in-cosmetics/Planilha sem ttulo - cscpopenda
```

In [4]:
```python
df.head()
```

Out[4]:

	CDPHId	ProductName	CSFId	CSF	CompanyId	CompanyName	BrandName	PrimaryCategoryId
0	41524	ULTRA COLOR RICH EXTRA PLUMP LIPSTICK-ALL SHADES	NaN	NaN	4	New Avon LLC	AVON	44
1	41523	Glover's Medicated Shampoo	NaN	NaN	338	J. Strickland & Co.	Glover's	18
2	41523	Glover's Medicated Shampoo	NaN	NaN	338	J. Strickland & Co.	Glover's	18
3	41523	PRECISION GLIMMER EYE LINER-ALL SHADES �	NaN	NaN	4	New Avon LLC	AVON	44
4	41523	AVON BRILLIANT SHINE LIP GLOSS-ALL SHADES �	NaN	NaN	4	New Avon LLC	AVON	44

5 rows × 22 columns

In [5]:
```python
df.info()
```

```
<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 [6]:
```
df.describe().transpose()
```

Out[6]:

	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 [7]:
```
df.shape
```

Out[7]:
```
(114635, 22)
```

In [8]:
```
#Check for null values in the dataset
df.isnull().sum()
```

Out[8]:
```
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 [9]:
```
#Checking the number of unique values
df.select_dtypes(include='int64').nunique()
```

Out[9]:
```
CDPHId               36972
CompanyId              635
PrimaryCategoryId       13
SubCategoryId           92
CasId                  134
ChemicalId           58079
ChemicalCount           10
dtype: int64
```

In [10]:
```
#Checking the number of unique values
df.select_dtypes(include='object').nunique()
```

Out[10]:
```
ProductName               33716
CSF                       34243
CompanyName                 606
BrandName                  2713
PrimaryCategory              13
SubCategory                  89
CasNumber                   125
ChemicalName                123
InitialDateReported        2274
MostRecentDateReported     2178
DiscontinuedDate            991
ChemicalCreatedAt          2320
ChemicalUpdatedAt          2326
ChemicalDateRemoved         524
dtype: int64
```

In [11]:
```
#check duplicate values
df.duplicated().sum()
```

Out[11]:
```
215
```

In [12]:
```python
#drop the duplicated values
df.drop_duplicates()
```

Out[12]:

	CDPHId	ProductName	CSFId	CSF	CompanyId	CompanyName	BrandName	PrimaryC
0	41524	ULTRA COLOR RICH EXTRA PLUMP LIPSTICK-ALL SHADES	NaN	NaN	4	New Avon LLC	AVON	
1	41523	Glover's Medicated Shampoo	NaN	NaN	338	J. Strickland & Co.	Glover's	
2	41523	Glover's Medicated Shampoo	NaN	NaN	338	J. Strickland & Co.	Glover's	
3	41523	PRECISION GLIMMER EYE LINER-ALL SHADES �	NaN	NaN	4	New Avon LLC	AVON	
4	41523	AVON BRILLIANT SHINE LIP GLOSS-ALL SHADES �	NaN	NaN	4	New Avon LLC	AVON	
...	...	...	...	...	...	...	...	...
114630	5	HYDRA-LIP TRANSLUCENT COLOR LIPSTICK	65001.0	Rosa Soft	1259	Yanbal USA, Inc	YANBAL	
114631	4	HYDRA-LIP TRANSLUCENT COLOR LIPSTICK	65002.0	Malva Spirit	1259	Yanbal USA, Inc	YANBAL	
114632	3	HYDRA-LIP TRANSLUCENT COLOR LIPSTICK	65003.0	Rojo Fashion	1259	Yanbal USA, Inc	YANBAL	
114633	3	HYDRA-LIP TRANSLUCENT COLOR LIPSTICK	65004.0	Terra Mystic	1259	Yanbal USA, Inc	YANBAL	
114634	2	OLD SPICE GENTLEMENS BLEND ALOE AND WILD SAGE ...	NaN	NaN	86	The Procter & Gamble Company	Old Spice	

114420 rows × 22 columns

In [13]:
```
column_names = df.columns.tolist()
print("Column Names:")
print(column_names)
```

```
Column Names:
['CDPHId', 'ProductName', 'CSFId', 'CSF', 'CompanyId', 'CompanyName', 'BrandName', 'P
rimaryCategoryId', 'PrimaryCategory', 'SubCategoryId', 'SubCategory', 'CasId', 'CasNu
mber', 'ChemicalId', 'ChemicalName', 'InitialDateReported', 'MostRecentDateReported',
'DiscontinuedDate', 'ChemicalCreatedAt', 'ChemicalUpdatedAt', 'ChemicalDateRemoved',
'ChemicalCount']
```

In [14]:
```
# Calculate the percentage of missing values for each column
na_percentage = (df.isnull().sum() / len(df)) * 100

# Create a DataFrame to store the results
na_percentage_df = pd.DataFrame({'Column': na_percentage.index, 'Percentage': na_perce

# Sort the DataFrame in descending order based on the percentage of missing values
na_percentage_df = na_percentage_df.sort_values(by='Percentage', ascending=False)

# Print the results
print("Missing Value Counts (Percentage-wise, Descending Order):\n")
print(na_percentage_df)
```

```
Missing Value Counts (Percentage-wise, Descending Order):

                    Column  Percentage
20     ChemicalDateRemoved   97.396083
17        DiscontinuedDate   88.729446
3                      CSF   30.006543
2                    CSFId   29.635801
12               CasNumber    5.649235
6                BrandName    0.198020
0                   CDPHId    0.000000
13              ChemicalId    0.000000
19       ChemicalUpdatedAt    0.000000
18       ChemicalCreatedAt    0.000000
16  MostRecentDateReported    0.000000
15     InitialDateReported    0.000000
14            ChemicalName    0.000000
11                   CasId    0.000000
1              ProductName    0.000000
10             SubCategory    0.000000
9            SubCategoryId    0.000000
8          PrimaryCategory    0.000000
7        PrimaryCategoryId    0.000000
5              CompanyName    0.000000
4                CompanyId    0.000000
21           ChemicalCount    0.000000
```

In [15]:
```
# Drop the not needed columns from the original DataFrame
df = df.drop(['ChemicalDateRemoved', 'DiscontinuedDate'], axis=1)

df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 114635 entries, 0 to 114634
Data columns (total 20 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  ChemicalCreatedAt       114635 non-null  object 
 18  ChemicalUpdatedAt       114635 non-null  object 
 19  ChemicalCount           114635 non-null  int64  
dtypes: float64(1), int64(7), object(12)
memory usage: 17.5+ MB
```

In [16]:
```python
# Drop all rows with null values from the original DataFrame
df.dropna(inplace=True)
```

In [17]:
```python
# Print null value counts for each column in the DataFrame
print("Null Value Counts for Each Column in df:\n")
print(df.isnull().sum())
```

```
Null Value Counts for Each Column in df:

CDPHId                    0
ProductName               0
CSFId                     0
CSF                       0
CompanyId                 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 [18]:
```
df.shape
```

Out[18]:
```
(76595, 20)
```

In [19]:
```
df.head()
```

Out[19]:

	CDPHId	ProductName	CSFId	CSF	CompanyId	CompanyName	BrandName	PrimaryCatego
6	41522	ABSOLUTE Precision Color Powder System - All S...	310.0	5858-81-1	11	OPI PRODUCTS INC.	OPI	
7	41522	ABSOLUTE Precision Color Powder System - All S...	311.0	D&C RED 7 CALCIUM LAKE	11	OPI PRODUCTS INC.	OPI	
8	41522	ABSOLUTE Precision Color Powder System - All S...	312.0	D&C RED 28	11	OPI PRODUCTS INC.	OPI	
9	41521	ABSOLUTE Precision Color Powder System Opaque ...	313.0	D&C RED 7 CALCIUM LAKE	11	OPI PRODUCTS INC.	ABSOLUTE	
11	41521	ABSOLUTE Precision Color Powder System Translu...	314.0	D&C RED 28	11	OPI PRODUCTS INC.	ABSOLUTE	

In [20]:
```
df.tail()
```

Out[20]:

	CDPHId	ProductName	CSFId	CSF	CompanyId	CompanyName	BrandName	PrimaryC
114629	6	EYESHADOW / ATARDECER NARANJA	65000.0	Crema T1	1259	Yanbal USA, Inc	YANBAL	
114630	5	HYDRA-LIP TRANSLUCENT COLOR LIPSTICK	65001.0	Rosa Soft	1259	Yanbal USA, Inc	YANBAL	
114631	4	HYDRA-LIP TRANSLUCENT COLOR LIPSTICK	65002.0	Malva Spirit	1259	Yanbal USA, Inc	YANBAL	
114632	3	HYDRA-LIP TRANSLUCENT COLOR LIPSTICK	65003.0	Rojo Fashion	1259	Yanbal USA, Inc	YANBAL	
114633	3	HYDRA-LIP TRANSLUCENT COLOR LIPSTICK	65004.0	Terra Mystic	1259	Yanbal USA, Inc	YANBAL	

In [21]:

```
#remove the columns which are the same, (cdphID & product name), (csfid &csf), (compan
df = df.drop(['CDPHId', 'CSFId','CompanyId','PrimaryCategoryId','ChemicalId','SubCateg

# Display the updated DataFrame
df.head()
```

Out[21]:

	ProductName	CSF	CompanyName	BrandName	PrimaryCategory	SubCategory	ChemicalNa
6	ABSOLUTE Precision Color Powder System - All S...	5858-81-1	OPI PRODUCTS INC.	OPI	Nail Products	Artificial Nails and Related Products	Titan dio:
7	ABSOLUTE Precision Color Powder System - All S...	D&C RED 7 CALCIUM LAKE	OPI PRODUCTS INC.	OPI	Nail Products	Artificial Nails and Related Products	Titan dio:
8	ABSOLUTE Precision Color Powder System - All S...	D&C RED 28	OPI PRODUCTS INC.	OPI	Nail Products	Artificial Nails and Related Products	Titan dio:
9	ABSOLUTE Precision Color Powder System Opaque ...	D&C RED 7 CALCIUM LAKE	OPI PRODUCTS INC.	ABSOLUTE	Nail Products	Artificial Nails and Related Products	Titan dio:
11	ABSOLUTE Precision Color Powder System Translu...	D&C RED 28	OPI PRODUCTS INC.	ABSOLUTE	Nail Products	Artificial Nails and Related Products	Titan dio:

In [22]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 76595 entries, 6 to 114633
Data columns (total 12 columns):
 #   Column                Non-Null Count  Dtype 
---  ------                --------------  ----- 
 0   ProductName           76595 non-null  object
 1   CSF                   76595 non-null  object
 2   CompanyName           76595 non-null  object
 3   BrandName             76595 non-null  object
 4   PrimaryCategory       76595 non-null  object
 5   SubCategory           76595 non-null  object
 6   ChemicalName          76595 non-null  object
 7   InitialDateReported   76595 non-null  object
 8   MostRecentDateReported  76595 non-null  object
 9   ChemicalCreatedAt     76595 non-null  object
 10  ChemicalUpdatedAt     76595 non-null  object
 11  ChemicalCount         76595 non-null  int64 
dtypes: int64(1), object(11)
memory usage: 7.6+ MB
```

In [23]:

```
#Checking the number of unique values
df.select_dtypes(include='object').nunique()
```

Out[23]:

```
ProductName                8179
CSF                       33952
CompanyName                 276
BrandName                   746
PrimaryCategory              13
SubCategory                  77
ChemicalName                 51
InitialDateReported        1350
MostRecentDateReported     1125
ChemicalCreatedAt          1424
ChemicalUpdatedAt          1441
dtype: int64
```

In [24]:

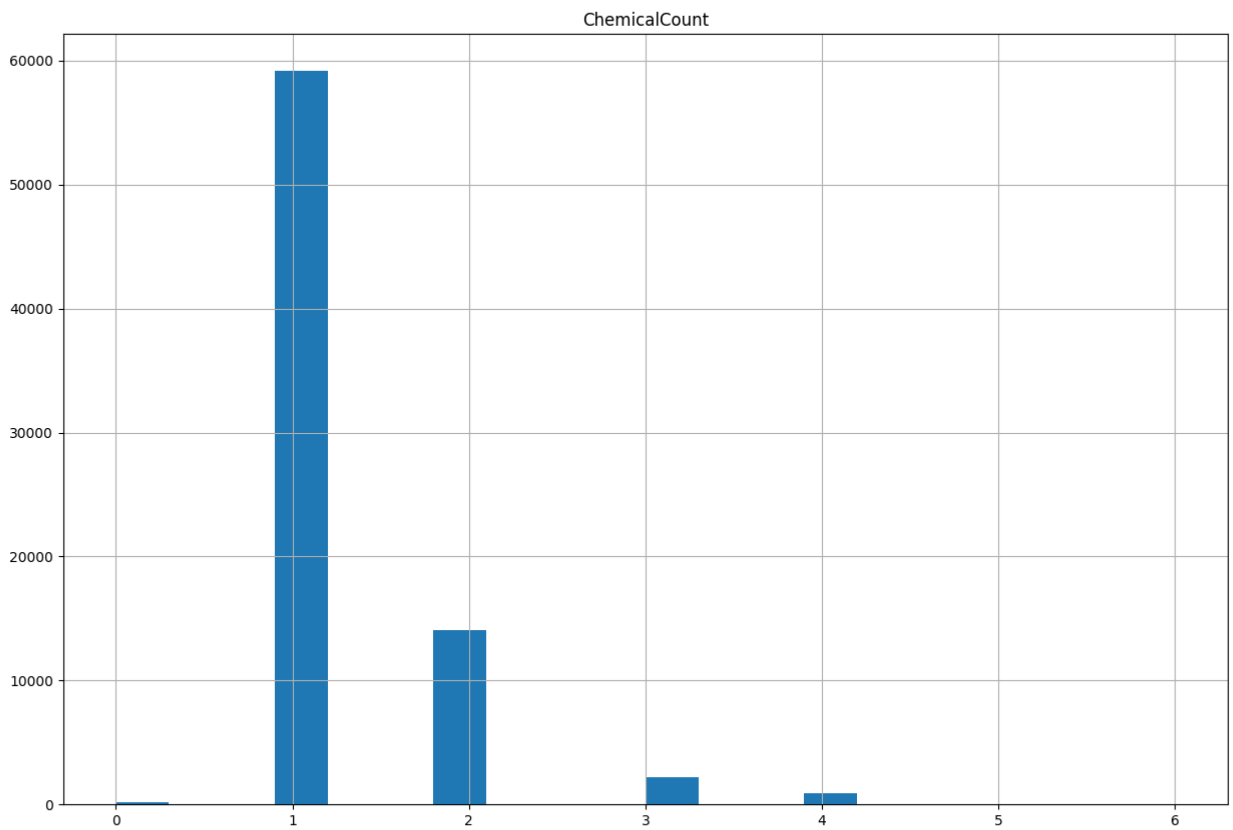
```
#Checking the number of unique values
df.select_dtypes(include='int64').nunique()
```

Out[24]:

```
ChemicalCount    7
dtype: int64
```

In [25]:

```
numeric_columns = df.select_dtypes(include=['int64','float64'])
numeric_columns.hist(bins=20, figsize=(15, 10))
plt.show()
```



In [26]:

```
# Convert date columns to datetime format
date_columns = ['InitialDateReported', 'MostRecentDateReported', 'ChemicalCreatedAt',
for column in date_columns:
    df[column] = pd.to_datetime(df[column], errors='coerce')

# Extract year and month from date columns
for column in date_columns:
    df[column + '_Year'] = df[column].dt.year
    df[column + '_Month'] = df[column].dt.month
```

```python
# Drop the original date columns
df.drop(date_columns, axis=1, inplace=True)
```

In [27]:
```python
df.head()
```

Out[27]:

	ProductName	CSF	CompanyName	BrandName	PrimaryCategory	SubCategory	ChemicalNa
6	ABSOLUTE Precision Color Powder System - All S...	5858-81-1	OPI PRODUCTS INC.	OPI	Nail Products	Artificial Nails and Related Products	Titan dio:
7	ABSOLUTE Precision Color Powder System - All S...	D&C RED 7 CALCIUM LAKE	OPI PRODUCTS INC.	OPI	Nail Products	Artificial Nails and Related Products	Titan dio:
8	ABSOLUTE Precision Color Powder System - All S...	D&C RED 28	OPI PRODUCTS INC.	OPI	Nail Products	Artificial Nails and Related Products	Titan dio:
9	ABSOLUTE Precision Color Powder System Opaque ...	D&C RED 7 CALCIUM LAKE	OPI PRODUCTS INC.	ABSOLUTE	Nail Products	Artificial Nails and Related Products	Titan dio:
11	ABSOLUTE Precision Color Powder System Translu...	D&C RED 28	OPI PRODUCTS INC.	ABSOLUTE	Nail Products	Artificial Nails and Related Products	Titan dio:

In [28]:
```python
# Drop the month columns
month_columns = [col for col in df.columns if '_Month' in col]
df.drop(month_columns, axis=1, inplace=True)
# Drop the 'CSF' column
df.drop('CSF', axis=1, inplace=True)
```

In [29]:
```python
# Check the number of unique values for each column
print(df.nunique())
```

```
ProductName                  8179
CompanyName                   276
BrandName                     746
PrimaryCategory                13
SubCategory                    77
ChemicalName                   51
ChemicalCount                   7
InitialDateReported_Year       12
MostRecentDateReported_Year    11
ChemicalCreatedAt_Year         12
ChemicalUpdatedAt_Year         12
dtype: int64
```

In [30]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 76595 entries, 6 to 114633
Data columns (total 11 columns):
 #   Column                      Non-Null Count  Dtype 
---  ------                      --------------  ----- 
 0   ProductName                 76595 non-null  object
 1   CompanyName                 76595 non-null  object
 2   BrandName                   76595 non-null  object
 3   PrimaryCategory             76595 non-null  object
 4   SubCategory                 76595 non-null  object
 5   ChemicalName                76595 non-null  object
 6   ChemicalCount               76595 non-null  int64 
 7   InitialDateReported_Year    76595 non-null  int32 
 8   MostRecentDateReported_Year 76595 non-null  int32 
 9   ChemicalCreatedAt_Year      76595 non-null  int32 
 10  ChemicalUpdatedAt_Year      76595 non-null  int32 
dtypes: int32(4), int64(1), object(6)
memory usage: 5.8+ MB
```

In [31]:
```
df.head()
```

Out[31]:

	ProductName	CompanyName	BrandName	PrimaryCategory	SubCategory	ChemicalName	Chem
6	ABSOLUTE Precision Color Powder System - All S...	OPI PRODUCTS INC.	OPI	Nail Products	Artificial Nails and Related Products	Titanium dioxide	
7	ABSOLUTE Precision Color Powder System - All S...	OPI PRODUCTS INC.	OPI	Nail Products	Artificial Nails and Related Products	Titanium dioxide	
8	ABSOLUTE Precision Color Powder System - All S...	OPI PRODUCTS INC.	OPI	Nail Products	Artificial Nails and Related Products	Titanium dioxide	
9	ABSOLUTE Precision Color Powder System Opaque ...	OPI PRODUCTS INC.	ABSOLUTE	Nail Products	Artificial Nails and Related Products	Titanium dioxide	
11	ABSOLUTE Precision Color Powder System Translu...	OPI PRODUCTS INC.	ABSOLUTE	Nail Products	Artificial Nails and Related Products	Titanium dioxide	

In [32]:
```
# Create a label encoder
label_encoder = LabelEncoder()

# Apply label encoding to each column in the DataFrame
```

```python
for column in df.columns:
    if df[column].dtype == 'object':
        df[column] = label_encoder.fit_transform(df[column])
```

In [33]:
```python
df.head()
```

Out[33]:

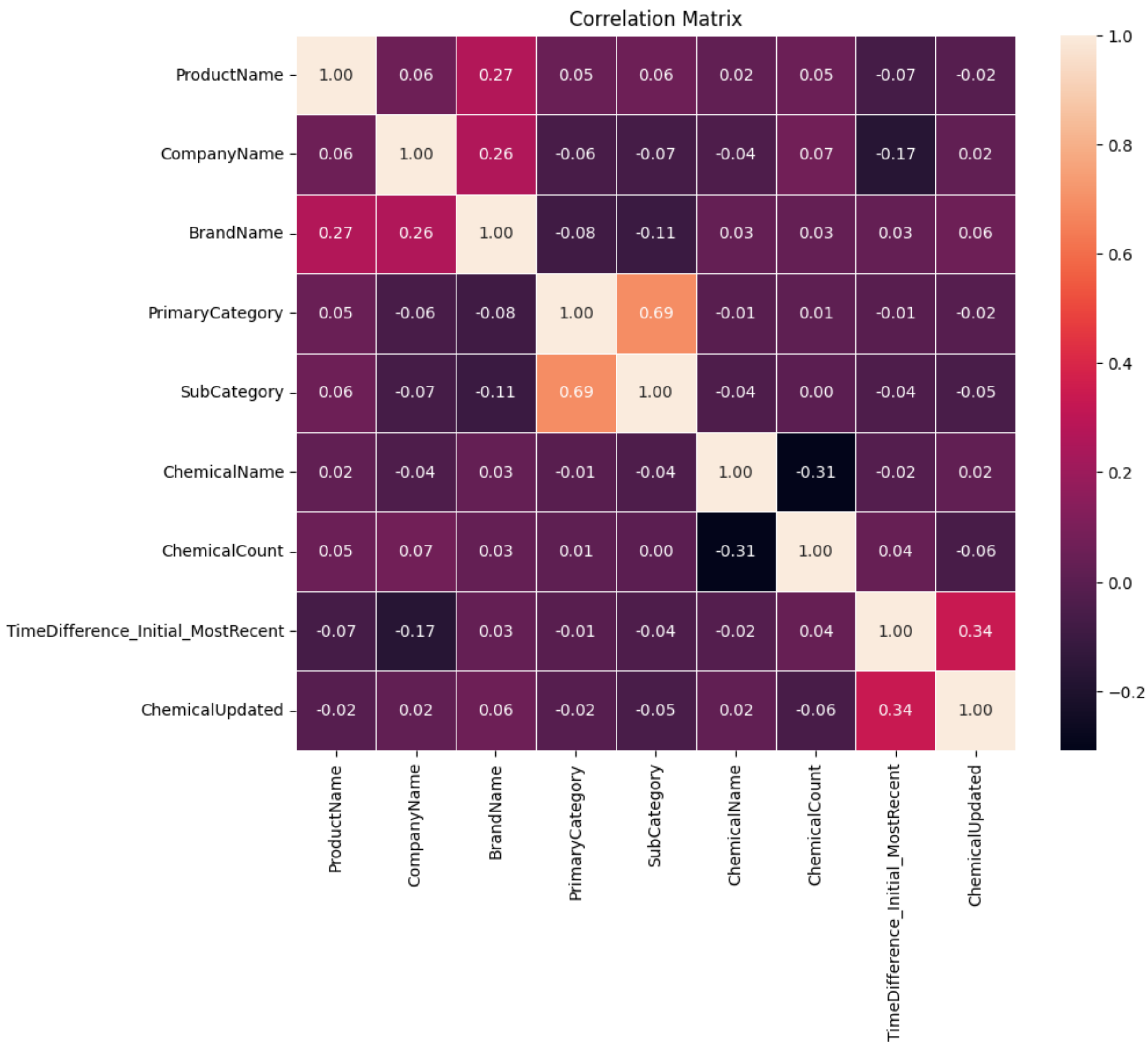
	ProductName	CompanyName	BrandName	PrimaryCategory	SubCategory	ChemicalName	Chem
6	138	176	501	6	3	42	
7	138	176	501	6	3	42	
8	138	176	501	6	3	42	
9	139	176	11	6	3	42	
11	140	176	11	6	3	42	

In [34]:
```python
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 76595 entries, 6 to 114633
Data columns (total 11 columns):
 #   Column                        Non-Null Count  Dtype
---  ------                        --------------  -----
 0   ProductName                   76595 non-null  int64
 1   CompanyName                   76595 non-null  int64
 2   BrandName                     76595 non-null  int64
 3   PrimaryCategory               76595 non-null  int64
 4   SubCategory                   76595 non-null  int64
 5   ChemicalName                  76595 non-null  int64
 6   ChemicalCount                 76595 non-null  int64
 7   InitialDateReported_Year      76595 non-null  int32
 8   MostRecentDateReported_Year   76595 non-null  int32
 9   ChemicalCreatedAt_Year        76595 non-null  int32
 10  ChemicalUpdatedAt_Year        76595 non-null  int32
dtypes: int32(4), int64(7)
memory usage: 5.8 MB
```

In [35]:
```python
# Create a new column for time difference
df['TimeDifference_Initial_MostRecent'] = df['MostRecentDateReported_Year'] - df['Init
df['ChemicalUpdated'] =df['ChemicalUpdatedAt_Year'] - df['ChemicalCreatedAt_Year']
# Drop the original date columns
df.drop(['InitialDateReported_Year', 'MostRecentDateReported_Year','ChemicalCreatedAt_
```

In [36]:
```python
# Plot the correlation matrix
correlation_matrix = df.corr()
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, fmt='.2f', linewidths=0.5)
plt.title('Correlation Matrix')
plt.show()
```



In [37]:
```
df.head()
```

Out[37]:

	ProductName	CompanyName	BrandName	PrimaryCategory	SubCategory	ChemicalName	Chem
6	138	176	501	6	3	42	
7	138	176	501	6	3	42	
8	138	176	501	6	3	42	
9	139	176	11	6	3	42	
11	140	176	11	6	3	42	

In [38]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 76595 entries, 6 to 114633
Data columns (total 9 columns):
 #   Column                           Non-Null Count  Dtype
---  ------                           --------------  -----
 0   ProductName                      76595 non-null  int64
 1   CompanyName                      76595 non-null  int64
 2   BrandName                        76595 non-null  int64
 3   PrimaryCategory                  76595 non-null  int64
 4   SubCategory                      76595 non-null  int64
 5   ChemicalName                     76595 non-null  int64
 6   ChemicalCount                    76595 non-null  int64
 7   TimeDifference_Initial_MostRecent  76595 non-null  int32
 8   ChemicalUpdated                  76595 non-null  int32
dtypes: int32(2), int64(7)
memory usage: 5.3 MB
```

In [39]:

```
df.shape
```

Out[39]:

```
(76595, 9)
```

In [40]:

```
# Calculate Z-scores for each column
z_scores = stats.zscore(df)

# Define a threshold for Z-scores (here: 3 standard deviations)
threshold = 3
outliers = (abs(z_scores) > threshold).all(axis=1)

# Remove outliers from the dataset
df_no_outliers = df[~outliers]

# Verify the shape of the new dataset
print("Original shape:", df.shape)
print("Shape after removing outliers:", df_no_outliers.shape)
```

```
Original shape: (76595, 9)
Shape after removing outliers: (76595, 9)
```

In [41]:

```
# 'ChemicalCount' is the target variable
X = df.drop('ChemicalCount', axis=1)  # Features
y = df['ChemicalCount']  # Target variable
```

In [42]:

```
from sklearn.model_selection import train_test_split

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

In [43]:

```
model = LinearRegression()
model.fit(X_train, y_train)
y_pred = model.predict(X_test)

mae = mean_absolute_error(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print(f'MAE: {mae}')
print(f'MSE: {mse}')
print(f'R-squared: {r2}')
```

```
MAE: 0.38114818709064274
MSE: 0.31050637157629807
R-squared: 0.11015402707992583
```

INITIAL TESTING

In [44]:
```
lr_model = LinearRegression()
lr_model.fit(X_train, y_train)

# Make predictions on the test set
lr_predictions = lr_model.predict(X_test)

# Evaluate the model's performance
mse = mean_squared_error(y_test, lr_predictions)
mae = mean_absolute_error(y_test, lr_predictions)
r2 = r2_score(y_test, lr_predictions)
print(f"Linear Regression Mean Squared Error: {mse:.2f}")
print(f"Linear Regression Mean Absolute Error: {mae:.2f}")
print(f'R-squared: {r2}')
```

```
Linear Regression Mean Squared Error: 0.31
Linear Regression Mean Absolute Error: 0.38
R-squared: 0.11015402707992583
```

In [45]:
```
# Initialize the Lasso Regression model
lasso_model = Lasso(alpha=1.0)

# Fit the model to the training data
lasso_model.fit(X_train, y_train)

# Make predictions on the test set
lasso_predictions = lasso_model.predict(X_test)

# Evaluate the model's performance
mse = mean_squared_error(y_test, lasso_predictions)
mae = mean_absolute_error(y_test, lasso_predictions)
r2 = r2_score(y_test, lasso_predictions)

print(f"Lasso Regression Mean Squared Error: {mse:.2f}")
print(f"Lasso Regression Mean Absolute Error: {mae:.2f}")
print(f"Lasso Regression R-squared: {r2:.2f}")
```

```
Lasso Regression Mean Squared Error: 0.33
Lasso Regression Mean Absolute Error: 0.42
Lasso Regression R-squared: 0.05
```

In [46]:
```
# Initialize the Ridge Regression model
ridge_model = Ridge(alpha=1.0)

# Fit the model to the training data
ridge_model.fit(X_train, y_train)

# Make predictions on the test set
ridge_predictions = ridge_model.predict(X_test)

# Evaluate the model's performance
mse = mean_squared_error(y_test, ridge_predictions)
mae = mean_absolute_error(y_test, ridge_predictions)
```

```
r2 = r2_score(y_test, ridge_predictions)

print(f"Ridge Regression Mean Squared Error: {mse:.2f}")
print(f"Ridge Regression Mean Absolute Error: {mae:.2f}")
print(f"Ridge Regression R-squared: {r2:.2f}")
```

```
Ridge Regression Mean Squared Error: 0.31
Ridge Regression Mean Absolute Error: 0.38
Ridge Regression R-squared: 0.11
```

In [47]:

```
# Initialize the Elastic Net model
elasticnet_model = ElasticNet(alpha=1.0, l1_ratio=0.5)

# Fit the model to the training data
elasticnet_model.fit(X_train, y_train)

# Make predictions on the test set
elasticnet_predictions = elasticnet_model.predict(X_test)

# Evaluate the model's performance
mse = mean_squared_error(y_test, elasticnet_predictions)
mae = mean_absolute_error(y_test, elasticnet_predictions)
r2 = r2_score(y_test, elasticnet_predictions)

print(f"Elastic Net Mean Squared Error: {mse:.2f}")
print(f"Elastic Net Mean Absolute Error: {mae:.2f}")
print(f"Elastic Net R-squared: {r2:.2f}")
```

```
Elastic Net Mean Squared Error: 0.32
Elastic Net Mean Absolute Error: 0.40
Elastic Net R-squared: 0.09
```

In [48]:

```
from sklearn.linear_model import LinearRegression, Lasso, Ridge, ElasticNet
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
from sklearn.model_selection import train_test_split

def evaluate_regression_model(model, X_train, X_test, y_train, y_test):
    # Fit the model to the training data
    model.fit(X_train, y_train)

    # Make predictions on the test set
    predictions = model.predict(X_test)

    # Evaluate the model's performance
    mse = mean_squared_error(y_test, predictions)
    mae = mean_absolute_error(y_test, predictions)
    r2 = r2_score(y_test, predictions)

    # Return the evaluation metrics
    return {'Mean Squared Error': mse, 'Mean Absolute Error': mae, 'R-squared': r2}


# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(df.drop('ChemicalCount', axis=1),

# Initialize models
lr_model = LinearRegression()
lasso_model = Lasso(alpha=1.0)
ridge_model = Ridge(alpha=1.0)
elasticnet_model = ElasticNet(alpha=1.0, l1_ratio=0.5)
```

```
# Evaluate models
lr_metrics = evaluate_regression_model(lr_model, X_train, X_test, y_train, y_test)
lasso_metrics = evaluate_regression_model(lasso_model, X_train, X_test, y_train, y_tes
ridge_metrics = evaluate_regression_model(ridge_model, X_train, X_test, y_train, y_tes
elasticnet_metrics = evaluate_regression_model(elasticnet_model, X_train, X_test, y_tr

# Print the results
print("Linear Regression Metrics:", lr_metrics)
print("Lasso Regression Metrics:", lasso_metrics)
print("Ridge Regression Metrics:", ridge_metrics)
print("Elastic Net Metrics:", elasticnet_metrics)
```

```
Linear Regression Metrics: {'Mean Squared Error': 0.31050637157629807, 'Mean Absolute
Error': 0.38114818709064274, 'R-squared': 0.11015402707992583}
Lasso Regression Metrics: {'Mean Squared Error': 0.3302049523037424, 'Mean Absolute E
rror': 0.4209245718931422, 'R-squared': 0.05370203659878947}
Ridge Regression Metrics: {'Mean Squared Error': 0.31050637030109834, 'Mean Absolute
Error': 0.38114818885372487, 'R-squared': 0.11015403073438013}
Elastic Net Metrics: {'Mean Squared Error': 0.31776296433535456, 'Mean Absolute Erro
r': 0.4014086668788687, 'R-squared': 0.08935815802581626}
```

LASSO Regression

In [49]:

```
# Define the parameter grid
param_grid = {'alpha': [0.001, 0.01, 0.1, 1, 10, 100]}

# Initialize Lasso Regression model
lasso_model = Lasso()

# Initialize GridSearchCV
lasso_grid_search = GridSearchCV(lasso_model, param_grid, cv=5, scoring='neg_mean_squa
lasso_grid_search.fit(X_train, y_train)

# Get the best hyperparameters
best_lasso_params = lasso_grid_search.best_params_

# Initialize Lasso model with best hyperparameters
best_lasso_model = Lasso(alpha=best_lasso_params['alpha'])
```

In [50]:

```
# Fit the Lasso model with the best hyperparameters
best_lasso_model.fit(X_train, y_train)

# Perform k-fold cross-validation
lasso_cv_scores = cross_val_score(best_lasso_model, X_train, y_train, cv=5, scoring='r
lasso_rmse_scores = np.sqrt(-lasso_cv_scores)

# Print the metrics
print("Lasso Regression Cross-Validation RMSE Scores:", lasso_rmse_scores)
print("Mean RMSE:", lasso_rmse_scores.mean())


y_pred = best_lasso_model.predict(X_test)
print("Test RMSE:", np.sqrt(mean_squared_error(y_test, y_pred)))
print("Test MAE:", mean_absolute_error(y_test, y_pred))
print("Test R-squared:", r2_score(y_test, y_pred))
```

```
Lasso Regression Cross-Validation RMSE Scores: [0.56160198 0.56785035 0.5557676  0.55
349208 0.55417529]
Mean RMSE: 0.558577457308336
Test RMSE: 0.5572406715293753
Test MAE: 0.38120646860219454
Test R-squared: 0.11012309251261077
```

RIDGE Regression

In [51]:

```
# Define the parameter grid
param_grid = {'alpha': [0.001, 0.01, 0.1, 1, 10, 100]}

# Initialize Ridge Regression model
ridge_model = Ridge()

# Initialize GridSearchCV
ridge_grid_search = GridSearchCV(ridge_model, param_grid, cv=5, scoring='neg_mean_squa
ridge_grid_search.fit(X_train, y_train)

# Get the best hyperparameters
best_ridge_params = ridge_grid_search.best_params_

# Initialize Ridge model with best hyperparameters
best_ridge_model = Ridge(alpha=best_ridge_params['alpha'])
```

In [52]:

```
# Fit the Ridge model with the best hyperparameters
best_ridge_model.fit(X_train, y_train)

# Perform k-fold cross-validation
ridge_cv_scores = cross_val_score(best_ridge_model, X_train, y_train, cv=5, scoring='r
ridge_rmse_scores = np.sqrt(-ridge_cv_scores)

# Print the metrics
print("Ridge Regression Cross-Validation RMSE Scores:", ridge_rmse_scores)
print("Mean RMSE:", ridge_rmse_scores.mean())


y_pred = best_ridge_model.predict(X_test)
print("Test RMSE:", np.sqrt(mean_squared_error(y_test, y_pred)))
print("Test MAE:", mean_absolute_error(y_test, y_pred))
print("Test R-squared:", r2_score(y_test, y_pred))
```

```
Ridge Regression Cross-Validation RMSE Scores: [0.56159427 0.5678207  0.55578352 0.55
34885  0.55418859]
Mean RMSE: 0.5585751167177138
Test RMSE: 0.5572309744650078
Test MAE: 0.38114820473940825
Test R-squared: 0.11015406339829659
```

ELASTIC NET Regression

In [53]:

```
# Define the parameter grid
param_grid = {'alpha': [0.001, 0.01, 0.1, 1, 10, 100], 'l1_ratio': [0.1, 0.3, 0.5, 0.7

# Initialize Elastic Net model
```

```
elasticnet_model = ElasticNet()

# Initialize GridSearchCV
elasticnet_grid_search = GridSearchCV(elasticnet_model, param_grid, cv=5, scoring='neg
elasticnet_grid_search.fit(X_train, y_train)

# Get the best hyperparameters
best_elasticnet_params = elasticnet_grid_search.best_params_

# Initialize Elastic Net model with best hyperparameters
best_elasticnet_model = ElasticNet(alpha=best_elasticnet_params['alpha'], l1_ratio=bes
```

In [54]:

```
# Fit the Elastic Net model with the best hyperparameters
best_elasticnet_model.fit(X_train, y_train)

# Perform k-fold cross-validation
elasticnet_cv_scores = cross_val_score(best_elasticnet_model, X_train, y_train, cv=5,
elasticnet_rmse_scores = np.sqrt(-elasticnet_cv_scores)

# Print the metrics
print("Elasticnet Regression Cross-Validation RMSE Scores:", elasticnet_rmse_scores)
print("Mean RMSE:", elasticnet_rmse_scores.mean())


y_pred = best_elasticnet_model.predict(X_test)
print("Test RMSE:", np.sqrt(mean_squared_error(y_test, y_pred)))
print("Test MAE:", mean_absolute_error(y_test, y_pred))
print("Test R-squared:", r2_score(y_test, y_pred))
```

```
Elasticnet Regression Cross-Validation RMSE Scores: [0.56159484 0.56782402 0.55578131
0.55348858 0.55418701]
Mean RMSE: 0.5585751524830123
Test RMSE: 0.5572316973846079
Test MAE: 0.3811541018349057
Test R-squared: 0.11015175452633719
```

Random Forest Regressor

In [55]:

```
from sklearn.ensemble import RandomForestRegressor
# Initialize Random Forest model
rf_model = RandomForestRegressor(n_estimators=50, max_depth=10, min_samples_split= 5)

# Fit the model to the training data
rf_model.fit(X_train, y_train)

# Make predictions on the test set
rf_predictions = rf_model.predict(X_test)

# Evaluate the model's performance
mse = mean_squared_error(y_test, rf_predictions)
mae = mean_absolute_error(y_test, rf_predictions)
r2 = r2_score(y_test, rf_predictions)

# Perform k-fold cross-validation
rf_cv_scores = cross_val_score(rf_model, X_train, y_train, cv=5, scoring='neg_mean_squ
rf_rmse_scores = np.sqrt(-rf_cv_scores)

# Print the metrics
```

```python
print("Random Forest Regression Cross-Validation RMSE Scores:", rf_rmse_scores)
print("Mean RMSE:", rf_rmse_scores.mean())


y_pred_rf = rf_model.predict(X_test)
print("Test RMSE:", np.sqrt(mean_squared_error(y_test, y_pred_rf)))
print("Test MAE:", mean_absolute_error(y_test, y_pred_rf))
print("Test R-squared:", r2_score(y_test, y_pred_rf))
```

```
Random Forest Regression Cross-Validation RMSE Scores: [0.32051682 0.33473696 0.32854
181 0.32496124 0.33299434]
Mean RMSE: 0.3283502351063231
Test RMSE: 0.33287594116083535
Test MAE: 0.1699433359047322
Test R-squared: 0.6824521784344542
```

LGBM Regressor

In [56]:

```python
from lightgbm import LGBMRegressor
# Initialize LGBM model
lgbm_model = LGBMRegressor(n_estimators=200, max_depth=50, learning_rate=  0.1)

# Fit the model to the training data
lgbm_model.fit(X_train, y_train)

# Make predictions on the test set
lgbm_predictions = lgbm_model.predict(X_test)

# Evaluate the model's performance
mse = mean_squared_error(y_test, rf_predictions)
mae = mean_absolute_error(y_test, rf_predictions)
r2 = r2_score(y_test, rf_predictions)

# Perform k-fold cross-validation
lgbm_cv_scores = cross_val_score(lgbm_model, X_train, y_train, cv=5, scoring='neg_mear
lgbm_rmse_scores = np.sqrt(-lgbm_cv_scores)

# Print the metrics
print("LGBM Regression Cross-Validation RMSE Scores:", lgbm_rmse_scores)
print("Mean RMSE:", lgbm_rmse_scores.mean())


y_pred_lgbm = lgbm_model.predict(X_test)
print("Test RMSE:", np.sqrt(mean_squared_error(y_test, y_pred_lgbm)))
print("Test MAE:", mean_absolute_error(y_test, y_pred_lgbm))
print("Test R-squared:", r2_score(y_test, y_pred_lgbm))
```

```
LGBM Regression Cross-Validation RMSE Scores: [0.26426246 0.26744224 0.25995353 0.259
33286 0.26805805]
Mean RMSE: 0.2638098285726663
Test RMSE: 0.25732805711459683
Test MAE: 0.12947584052342934
Test R-squared: 0.8102339118898256
```

CatBoost Regressor

In [57]:

```
from catboost import CatBoostRegressor

# Define the parameter grid
#param_grid_catboost = {'iterations': [50, 100, 200], 'depth': [4, 6, 8], 'learning_ra

# Initialize CatBoost model
catboost_model = CatBoostRegressor(iterations=100,depth = 8 , learning_rate = 0.1)

# Fit the model with the best hyperparameters
catboost_model.fit(X_train, y_train)

# Perform k-fold cross-validation
catboost_cv_scores = cross_val_score(catboost_model, X_train, y_train, cv=5, scoring='
catboost_rmse_scores = np.sqrt(-catboost_cv_scores)

# Print the metrics
print("CatBoost Regression Cross-Validation RMSE Scores:", catboost_rmse_scores)
print("Mean RMSE:", catboost_rmse_scores.mean())


y_pred_catboost = catboost_model.predict(X_test)
print("Test RMSE:", np.sqrt(mean_squared_error(y_test, y_pred_catboost)))
print("Test MAE:", mean_absolute_error(y_test, y_pred_catboost))
print("Test R-squared:", r2_score(y_test, y_pred_catboost))
```

```
0:	learn: 0.5684330	total: 62.9ms	remaining: 6.23s
1:	learn: 0.5520901	total: 71ms	remaining: 3.48s
2:	learn: 0.5381373	total: 78.3ms	remaining: 2.53s
3:	learn: 0.5240120	total: 85.7ms	remaining: 2.06s
4:	learn: 0.5103243	total: 92.7ms	remaining: 1.76s
5:	learn: 0.4971489	total: 100ms	remaining: 1.57s
6:	learn: 0.4889165	total: 108ms	remaining: 1.44s
7:	learn: 0.4788177	total: 116ms	remaining: 1.33s
8:	learn: 0.4704789	total: 124ms	remaining: 1.25s
9:	learn: 0.4630636	total: 132ms	remaining: 1.19s
10:	learn: 0.4570805	total: 140ms	remaining: 1.13s
11:	learn: 0.4503325	total: 147ms	remaining: 1.08s
12:	learn: 0.4455834	total: 154ms	remaining: 1.03s
13:	learn: 0.4425158	total: 161ms	remaining: 991ms
14:	learn: 0.4378039	total: 169ms	remaining: 957ms
15:	learn: 0.4330359	total: 176ms	remaining: 926ms
16:	learn: 0.4296463	total: 184ms	remaining: 897ms
17:	learn: 0.4271568	total: 191ms	remaining: 869ms
18:	learn: 0.4242989	total: 198ms	remaining: 844ms
19:	learn: 0.4204402	total: 207ms	remaining: 826ms
20:	learn: 0.4179421	total: 215ms	remaining: 809ms
21:	learn: 0.4139306	total: 223ms	remaining: 791ms
22:	learn: 0.4106899	total: 231ms	remaining: 775ms
23:	learn: 0.4090447	total: 239ms	remaining: 755ms
24:	learn: 0.4066953	total: 246ms	remaining: 738ms
25:	learn: 0.4052043	total: 254ms	remaining: 722ms
26:	learn: 0.4026892	total: 262ms	remaining: 709ms
27:	learn: 0.4010437	total: 271ms	remaining: 697ms
28:	learn: 0.3994518	total: 278ms	remaining: 681ms
29:	learn: 0.3976824	total: 287ms	remaining: 669ms
30:	learn: 0.3953338	total: 295ms	remaining: 657ms
31:	learn: 0.3933505	total: 303ms	remaining: 644ms
32:	learn: 0.3913089	total: 311ms	remaining: 631ms
33:	learn: 0.3902898	total: 319ms	remaining: 620ms
34:	learn: 0.3890837	total: 329ms	remaining: 611ms
35:	learn: 0.3873399	total: 337ms	remaining: 599ms
36:	learn: 0.3849644	total: 345ms	remaining: 588ms
37:	learn: 0.3837125	total: 353ms	remaining: 576ms
38:	learn: 0.3819181	total: 361ms	remaining: 564ms
39:	learn: 0.3813497	total: 367ms	remaining: 551ms
40:	learn: 0.3803891	total: 375ms	remaining: 540ms
41:	learn: 0.3781717	total: 383ms	remaining: 529ms
42:	learn: 0.3772107	total: 390ms	remaining: 517ms
43:	learn: 0.3758278	total: 398ms	remaining: 507ms
44:	learn: 0.3754159	total: 405ms	remaining: 495ms
45:	learn: 0.3738076	total: 412ms	remaining: 484ms
46:	learn: 0.3731487	total: 419ms	remaining: 472ms
47:	learn: 0.3723391	total: 425ms	remaining: 461ms
48:	learn: 0.3706657	total: 433ms	remaining: 451ms
49:	learn: 0.3697661	total: 441ms	remaining: 441ms
50:	learn: 0.3692560	total: 448ms	remaining: 430ms
51:	learn: 0.3678558	total: 455ms	remaining: 420ms
52:	learn: 0.3672839	total: 464ms	remaining: 412ms
53:	learn: 0.3665824	total: 472ms	remaining: 402ms
54:	learn: 0.3653153	total: 480ms	remaining: 393ms
55:	learn: 0.3643711	total: 488ms	remaining: 383ms
56:	learn: 0.3636370	total: 495ms	remaining: 373ms
57:	learn: 0.3620553	total: 503ms	remaining: 364ms
58:	learn: 0.3612416	total: 510ms	remaining: 354ms
59:	learn: 0.3606254	total: 517ms	remaining: 345ms
```

```
40:	learn: 0.3851573	total: 286ms	remaining: 412ms
41:	learn: 0.3830812	total: 293ms	remaining: 405ms
42:	learn: 0.3817961	total: 300ms	remaining: 398ms
43:	learn: 0.3793966	total: 307ms	remaining: 391ms
44:	learn: 0.3784993	total: 314ms	remaining: 383ms
45:	learn: 0.3764445	total: 321ms	remaining: 377ms
46:	learn: 0.3758205	total: 328ms	remaining: 370ms
47:	learn: 0.3738338	total: 336ms	remaining: 364ms
48:	learn: 0.3724735	total: 343ms	remaining: 357ms
49:	learn: 0.3716736	total: 351ms	remaining: 351ms
50:	learn: 0.3709611	total: 357ms	remaining: 343ms
51:	learn: 0.3696846	total: 365ms	remaining: 337ms
52:	learn: 0.3689307	total: 372ms	remaining: 329ms
53:	learn: 0.3681446	total: 380ms	remaining: 324ms
54:	learn: 0.3668783	total: 388ms	remaining: 317ms
55:	learn: 0.3658837	total: 395ms	remaining: 310ms
56:	learn: 0.3653936	total: 401ms	remaining: 303ms
57:	learn: 0.3644950	total: 409ms	remaining: 296ms
58:	learn: 0.3637285	total: 417ms	remaining: 290ms
59:	learn: 0.3628107	total: 425ms	remaining: 283ms
60:	learn: 0.3623787	total: 431ms	remaining: 276ms
61:	learn: 0.3618818	total: 437ms	remaining: 268ms
62:	learn: 0.3610949	total: 444ms	remaining: 260ms
63:	learn: 0.3599156	total: 450ms	remaining: 253ms
64:	learn: 0.3590027	total: 457ms	remaining: 246ms
65:	learn: 0.3583034	total: 464ms	remaining: 239ms
66:	learn: 0.3575285	total: 470ms	remaining: 232ms
67:	learn: 0.3567331	total: 478ms	remaining: 225ms
68:	learn: 0.3553793	total: 485ms	remaining: 218ms
69:	learn: 0.3543115	total: 493ms	remaining: 211ms
70:	learn: 0.3533886	total: 499ms	remaining: 204ms
71:	learn: 0.3523799	total: 507ms	remaining: 197ms
72:	learn: 0.3516349	total: 514ms	remaining: 190ms
73:	learn: 0.3507537	total: 521ms	remaining: 183ms
74:	learn: 0.3491128	total: 529ms	remaining: 176ms
75:	learn: 0.3484902	total: 536ms	remaining: 169ms
76:	learn: 0.3475737	total: 543ms	remaining: 162ms
77:	learn: 0.3469779	total: 550ms	remaining: 155ms
78:	learn: 0.3464335	total: 557ms	remaining: 148ms
79:	learn: 0.3457873	total: 564ms	remaining: 141ms
80:	learn: 0.3448457	total: 572ms	remaining: 134ms
81:	learn: 0.3444011	total: 578ms	remaining: 127ms
82:	learn: 0.3431087	total: 586ms	remaining: 120ms
83:	learn: 0.3424805	total: 593ms	remaining: 113ms
84:	learn: 0.3413953	total: 601ms	remaining: 106ms
85:	learn: 0.3406845	total: 608ms	remaining: 99ms
86:	learn: 0.3402693	total: 616ms	remaining: 92.1ms
87:	learn: 0.3399528	total: 625ms	remaining: 85.2ms
88:	learn: 0.3392806	total: 632ms	remaining: 78.1ms
89:	learn: 0.3382398	total: 640ms	remaining: 71.1ms
90:	learn: 0.3377996	total: 647ms	remaining: 64ms
91:	learn: 0.3368222	total: 653ms	remaining: 56.8ms
92:	learn: 0.3361037	total: 660ms	remaining: 49.7ms
93:	learn: 0.3353724	total: 667ms	remaining: 42.6ms
94:	learn: 0.3349562	total: 674ms	remaining: 35.5ms
95:	learn: 0.3344982	total: 682ms	remaining: 28.4ms
96:	learn: 0.3339837	total: 689ms	remaining: 21.3ms
97:	learn: 0.3331642	total: 696ms	remaining: 14.2ms
98:	learn: 0.3326931	total: 704ms	remaining: 7.11ms
99:	learn: 0.3319979	total: 711ms	remaining: 0us
```

```
CatBoost Regression Cross-Validation RMSE Scores: [0.3348018  0.34303145 0.34079943
0.33935277 0.34120893]
Mean RMSE: 0.3398388765054262
Test RMSE: 0.3395217014473761
Test MAE: 0.19573309082166934
Test R-squared: 0.6696461289002491
```

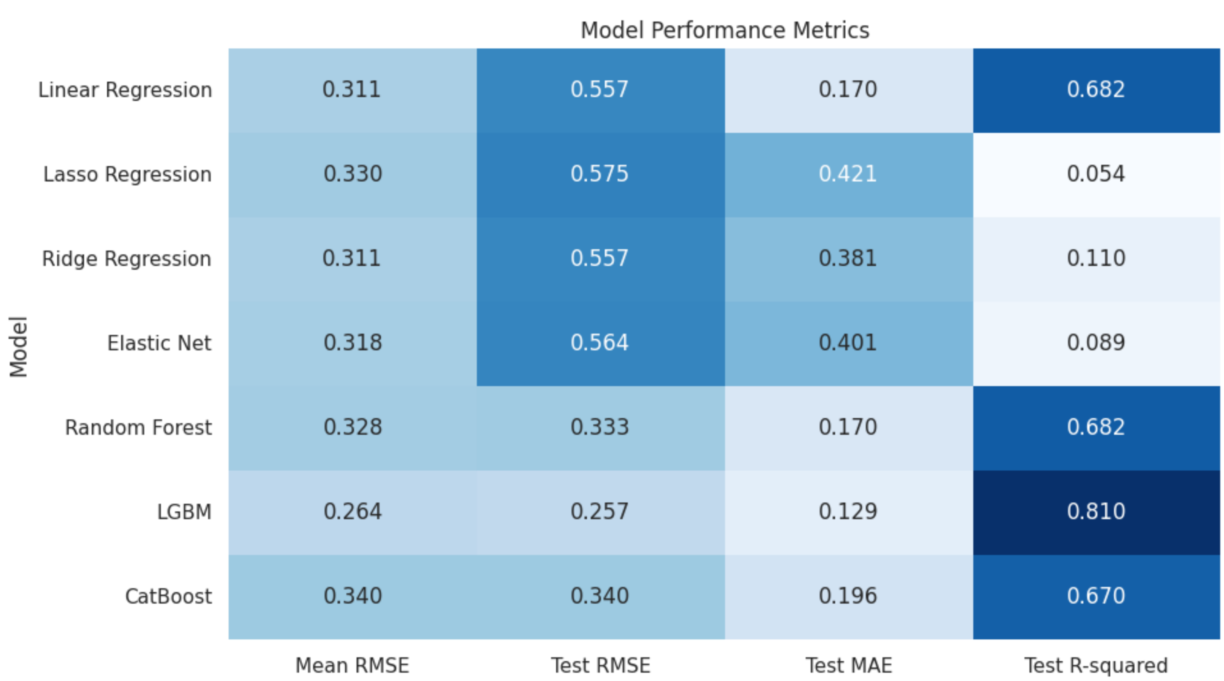
Model Performance Metrics

In [58]:

```python
# Create a DataFrame with the metrics for each model
data = {
    'Model': ['Linear Regression', 'Lasso Regression', 'Ridge Regression', 'Elastic Ne
    'Mean RMSE': [lr_metrics['Mean Squared Error'], lasso_metrics['Mean Squared Error'
                  elasticnet_metrics['Mean Squared Error'], rf_rmse_scores.mean(), lgb
    'Test RMSE': [np.sqrt(mean_squared_error(y_test, lr_predictions)), np.sqrt(mean_sq
                  np.sqrt(mean_squared_error(y_test, ridge_predictions)), np.sqrt(mear
                  np.sqrt(mean_squared_error(y_test, y_pred_rf)), np.sqrt(mean_squarec
                  np.sqrt(mean_squared_error(y_test, y_pred_catboost))],
    'Test MAE': [mae, lasso_metrics['Mean Absolute Error'], ridge_metrics['Mean Absolu
                 elasticnet_metrics['Mean Absolute Error'], mean_absolute_error(y_test
                 mean_absolute_error(y_test, y_pred_lgbm), mean_absolute_error(y_test,
    'Test R-squared': [r2, lasso_metrics['R-squared'], ridge_metrics['R-squared'],
                       elasticnet_metrics['R-squared'], r2_score(y_test, y_pred_rf),
                       r2_score(y_test, y_pred_lgbm), r2_score(y_test, y_pred_catboost
}

df_metrics = pd.DataFrame(data)

# Plot a table using seaborn
plt.figure(figsize=(10, 6))
sns.set_theme(style="whitegrid")
table = sns.heatmap(df_metrics.set_index('Model'), annot=True, cmap="Blues", fmt=".3f"
plt.title('Model Performance Metrics')
plt.show()
```



Prediction on Random 10 rows in the dataset

In [59]:
```
#  Select random 10 rows
random_rows = df.sample(10, random_state=42)  # You can adjust the random_state for re

#  Use trained models to predict Chemical Count
random_rows_X = random_rows.drop('ChemicalCount', axis=1)

# Linear Regression
lr_predictions = lr_model.predict(random_rows_X)

# Lasso Regression
lasso_predictions = best_lasso_model.predict(random_rows_X)

# Ridge Regression
ridge_predictions = best_ridge_model.predict(random_rows_X)

# Elastic Net Regression
elasticnet_predictions = best_elasticnet_model.predict(random_rows_X)

# Random Forest Regression
rf_predictions = rf_model.predict(random_rows_X)

# LGBM Regression
lgbm_predictions = lgbm_model.predict(random_rows_X)

# CatBoost Regression
catboost_predictions = catboost_model.predict(random_rows_X)

#Create a table
prediction_table = pd.DataFrame({
    'Actual_ChemicalCount': random_rows['ChemicalCount'].values,
    'Linear_Regression': lr_predictions,
    'Lasso_Regression': lasso_predictions,
    'Ridge_Regression': ridge_predictions,
    'ElasticNet_Regression': elasticnet_predictions,
    'RandomForest_Regression': rf_predictions,
    'LGBM_Regression': lgbm_predictions,
    'CatBoost_Regression': catboost_predictions
})

# Display the prediction table
print(prediction_table)
```

```
   Actual_ChemicalCount  Linear_Regression  Lasso_Regression  \
0                     4           1.411662          1.408694
1                     1           1.267941          1.265423
2                     2           1.250786          1.250171
3                     2           1.238902          1.238902
4                     1           1.204234          1.204573
5                     1           1.258559          1.258744
6                     1           1.316727          1.316377
7                     2           1.269267          1.270015
8                     2           1.246874          1.247107
9                     1           1.295899          1.295113

   Ridge_Regression  ElasticNet_Regression  RandomForest_Regression  \
0          1.411647               1.411286                 3.143386
1          1.267928               1.267621                 1.001180
2          1.250785               1.250719                 1.057445
3          1.238902               1.238906                 1.175989
4          1.204234               1.204267                 1.057445
5          1.258557               1.258567                 1.057445
6          1.316726               1.316685                 1.130517
7          1.269268               1.269348                 1.369610
8          1.246874               1.246894                 1.861840
9          1.295898               1.295810                 1.150812

   LGBM_Regression  CatBoost_Regression
0         3.140672             2.455530
1         1.007706             1.008220
2         1.133689             1.120932
3         1.596924             1.280145
4         1.006736             1.060738
5         1.129186             1.234258
6         1.047561             1.158851
7         1.288512             1.200647
8         1.901961             1.771320
9         1.114100             1.236704
```

In [60]:
```python
from pandas.plotting import table
# Plotting the table
fig, ax = plt.subplots(figsize=(12, 4))
ax.axis('off')  # Turn off the axis

# Create a table and add it to the plot
tab = table(ax, prediction_table, loc='center', colWidths=[0.2]*len(prediction_table.c
tab.auto_set_font_size(False)
tab.set_fontsize(10)
tab.scale(1.2, 1.2)  # Adjust the table size

# Display the plot
plt.show()
```

	Actual_ChemicalCount	Linear_Regression	Lasso_Regression	Ridge_Regression	ElasticNet_Regression	RandomForest_Regression	LGBM_Regression	CatBoost_Regression
0	4.0	1.4116616187291728	1.4086936463860478	1.4116472270659977	1.4112858270084958	3.1433864384215298	3.1406719681023003	2.455529639496131
1	1.0	1.2679409018013925	1.2654232376752241	1.2679284620518405	1.2676208974895673	1.0011802668278993	1.0077058118184237	1.0082196098104383
2	2.0	1.2507862620656187	1.2501711238360391	1.2507851595068575	1.2507188755466236	1.057445089955324	1.1336887588682136	1.1209321581421674
3	2.0	1.2389017884296498	1.238902491671153	1.238902464402034	1.2389056353997243	1.1759887795831885	1.5969238275116808	1.2801445781756497
4	1.0	1.2042343449464585	1.2045726381780746	1.2042341618801684	1.2042670961941377	1.057445089955324	1.0067358504169592	1.0607379173057203
5	1.0	1.258558727824879	1.2587439722494438	1.2585568714918307	1.258566893068	1.057445089955324	1.1291859472322925	1.2342579613771205
6	1.0	1.3167269374602517	1.316377325951768	1.316725674351368	1.316685118387062	1.1305173023697206	1.0475607189877334	1.1588508908432107
7	2.0	1.2692674582375765	1.2700147840334202	1.2692684674890988	1.2693475901224167	1.3696101626935002	1.2885116834853785	1.200646882377074
8	2.0	1.2468743108371794	1.2471072353159331	1.2468735876799806	1.2468935552498126	1.8618398232833437	1.9019608194162305	1.771320248995839
9	1.0	1.2958994379017832	1.2951133415159064	1.2958975075042702	1.2958104046967032	1.1508121294200364	1.114100362700154	1.2367037041208226

Performance Summary

**The LightGBM Regression model is the best-performing model among the ones that have been tested.

**Here's why:

**Lower Cross-Validation RMSE: The mean cross-validation RMSE for the LGBM model is 0.2638, which is the lowest among all the models. This indicates that, on average, the LGBM model has the smallest error when predicting the target variable across different folds.

**Lower Test RMSE: The test RMSE for the LGBM model is 0.2573, again the lowest compared to other models. This means that the LGBM model performs well not only in cross-validation but also on unseen data, providing accurate predictions.

**Higher R-squared Value: The test R-squared value for the LGBM model is 0.8102, which is the highest. R-squared measures the proportion of the variance in the dependent variable that is predictable from the independent variables. A higher R-squared value indicates a better fit of the model to the data.

**Consistency Across Metrics: The LGBM model consistently performs well across different evaluation metrics, including cross-validation scores and test set scores.

In summary, the LightGBM Regression model outperforms other models in terms of both cross-validation and test set performance, making it the best choice for predicting the Chemical Count in this dataset.