In [1]:
```
1 import pandas as pd
```

```
C:\Users\HP\AppData\Roaming\Python\Python311\site-packages\pandas\core\arrays\masked.py:60: UserWarning: Pandas requires ve
rsion '1.3.6' or newer of 'bottleneck' (version '1.3.5' currently installed).
  from pandas.core import (
```

In [2]:
```
1 train_df=pd.read_csv('Datasets/messy_train_data.csv')
```

In [3]:
```
1 train_df.head()
```

Out[3]:

	Followers	EngagementRate (%)	AdSpend (GBP)	ContentQuality	Sales (Units)	ID	Timestamp	Notes
0	106572.0	2.573174871146172	2614.3781948587675	5.275680	6340	9254	2021-11-27	Pending
1	77583.0	0.9394984315675532	4975.962514379572	8.756268	5793	1561	2022-02-13	Review
2	92832.0	2.1761012652155296	4107.769534318886	6.454727	8104	1670	2023-09-25	Pending
3	53565.0	1.4783757541486553	4293.330464613049	4.312813	7293	6087	2023-02-15	Review
4	121079.0	3.3741976179329356	5343.549440897207	3.769047	14396	6669	2023-05-28	NaN

In [4]:
```
1 train_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8000 entries, 0 to 7999
Data columns (total 8 columns):
 #   Column              Non-Null Count  Dtype  
---  ------              --------------  -----  
 0   Followers           7840 non-null   float64
 1   EngagementRate (%)  7840 non-null   object 
 2   AdSpend (GBP)       7841 non-null   object 
 3   ContentQuality      7840 non-null   float64
 4   Sales (Units)       8000 non-null   int64  
 5   ID                  8000 non-null   int64  
 6   Timestamp           8000 non-null   object 
 7   Notes               5348 non-null   object 
dtypes: float64(2), int64(2), object(4)
memory usage: 500.1+ KB
```

In [5]:
```
1 train_df.describe()
```

Out[5]:

	Followers	ContentQuality	Sales (Units)	ID
count	7.840000e+03	7840.000000	8000.000000	8000.000000
mean	1.111104e+06	5.492899	10544.674375	5011.506875
std	4.013038e+07	2.608038	2808.151485	2887.649416
min	2.000000e+01	1.000151	590.000000	1.000000
25%	7.800250e+04	3.227357	8642.250000	2511.750000
50%	9.912050e+04	5.468786	10500.000000	5013.500000
75%	1.198655e+05	7.780281	12459.000000	7504.250000
max	1.629447e+09	9.999749	20263.000000	9999.000000

In [6]:
```
1 test_df=pd.read_csv('Datasets/messy_test_data.csv')
```

In [7]:
```
1 test_df.head()
```

Out[7]:

	Followers	EngagementRate (%)	AdSpend (GBP)	ContentQuality	ID	Timestamp	Notes
0	179136.0	2.5570425842061986	3975.099954173261	1.803620	6252	2021-10-08	Good
1	68888.0	3.451254744278324	5392.048613170361	2.993966	4684	2021-10-01	Good
2	89520.0	0.7342357926528821	5850.470394900652	4.525990	1731	2021-06-24	NaN
3	100048.0	1.5972073367018218	5792.432498712002	5.051500	4742	2021-11-22	NaN
4	132229.0	1.3874265375395036	5095.269891920688	3.580921	4521	2021-07-19	Review

In [8]:

```
for col in train_df.columns:
    print(col,':''\n', train_df[col].unique())
    print('-'*70)
```

```
 '2023-01-17' '2022-08-31' '2021-12-04' '2023-09-01' '2021-06-08'
 '2021-10-18' '2021-03-19' '2023-06-03' '2021-07-18' '2021-01-06'
 '2021-09-13' '2023-07-07' '2021-06-16' '2022-12-12' '2021-07-11'
 '2023-03-31' '2022-07-17' '2022-05-30' '2021-09-15' '2021-02-26'
 '2022-12-27' '2021-11-07' '2021-03-17' '2023-06-13' '2022-06-17'
 '2021-12-06' '2023-05-20' '2023-08-12' '2023-01-15' '2022-09-08'
 '2021-06-27' '2023-08-21' '2021-01-08' '2022-05-23' '2023-04-25'
 '2021-10-01' '2021-02-13' '2022-11-18' '2021-12-07' '2023-06-19'
 '2022-03-19' '2021-09-26' '2022-07-29' '2021-06-25' '2021-10-15'
 '2022-02-25' '2021-03-20' '2022-05-10' '2021-04-27' '2023-04-01'
 '2021-08-10' '2022-12-04' '2023-07-13' '2021-06-30' '2023-07-10'
 '2021-04-10' '2021-08-14' '2023-08-05' '2022-12-11' '2022-11-29'
 '2021-06-19' '2022-04-05' '2021-09-20' '2021-11-30' '2023-03-10'
 '2021-06-22' '2021-08-20' '2023-03-09' '2023-06-07' '2021-03-07'
 '2022-01-23' '2022-08-11' '2022-11-05' '2021-05-10' '2021-11-29'
 '2022-11-04' '2023-06-06' '2022-08-20' '2023-01-06' '2021-12-24'
 '2023-01-31' '2021-07-23' '2022-06-19' '2021-01-22' '2021-09-05'
 '2022-10-11' '2021-09-25' '2022-06-04' '2021-03-16' '2023-03-12'
 '2021-10-23' '2021-08-27' '2021-10-17' '2022-12-21' '2023-02-23'
 '2021-05-26' '2022-05-15' '2023-01-27' '2023-07-03' '2022-08-02'
```

In [9]:

```
unique_engagements = train_df["EngagementRate (%)"].unique()
print("\n".join(unique_engagements.astype(str)))
```

```
2.1133970751642974
2.7040369012756984
0.8533004819483476
3.531480889520454
3.682718266750048
0.5980502739301499
1.6724525634938203
4.83659224680493
4.239530550453986
2.3282181574127794
2.8534428868626938
0.6001265836684893
1.5570781750905736
4.422510091712807
0.8815128067838947
3.513866529595208
0.6606992706266439
3.4113455780958892
0.7871213484407076
0.9609895230469404
```

In [10]:

```
unique_AdSpend = train_df["AdSpend (GBP)"].unique()
print("\n".join(unique_AdSpend.astype(str)))
```

```
2614.3781948587675
4975.962514379572
4107.769534318886
4293.330464613049
5343.549440897207
6640.860427448698
3594.5211897372737
4447.20233454365
3940.505439955562
7594.681081184365
2767.1830414520728
6641.859936830851
3441.009051682889
5237.971176818418
7745.024429172723
3458.3391231292194
4829.258278306401
2598.5113329869146
4243.473719089614
```

In [11]:

```
def clean_campaign_data(df):
    df = df.copy()

    # Engagement Rate: Remove % and convert to float
    df['EngagementRate (%)'] = df['EngagementRate (%)'].astype(str).str.replace('%', '', regex=False)
    df['EngagementRate (%)'] = pd.to_numeric(df['EngagementRate (%)'], errors='coerce') / 100.0

    # Ad Spend: Remove £
    df['AdSpend (GBP)'] = df['AdSpend (GBP)'].astype(str).str.replace('£', '', regex=False)
    df['AdSpend (GBP)'] = pd.to_numeric(df['AdSpend (GBP)'], errors='coerce')

    # Timestamp to datetime
    df['Timestamp'] = pd.to_datetime(df['Timestamp'], errors='coerce')

    # Extract time features
    df['CampaignMonth'] = df['Timestamp'].dt.month
    df['CampaignWeekday'] = df['Timestamp'].dt.weekday
    df['CampaignYear'] = df['Timestamp'].dt.year

    # Drop unused columns
    df.drop(columns=['Notes', 'ID'], errors='ignore', inplace=True)

    # Drop missing
    df.dropna(inplace=True)

    return df
```

In [12]:

```
train = clean_campaign_data(train_df)
test = clean_campaign_data(test_df)
```

In [13]:

```
train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 7380 entries, 0 to 7999
Data columns (total 9 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   Followers           7380 non-null   float64
 1   EngagementRate (%)  7380 non-null   float64
 2   AdSpend (GBP)       7380 non-null   float64
 3   ContentQuality      7380 non-null   float64
 4   Sales (Units)       7380 non-null   int64
 5   Timestamp           7380 non-null   datetime64[ns]
 6   CampaignMonth       7380 non-null   int32
 7   CampaignWeekday     7380 non-null   int32
 8   CampaignYear        7380 non-null   int32
dtypes: datetime64[ns](1), float64(4), int32(3), int64(1)
memory usage: 490.1 KB
```

In [14]:

```
train.head()
```

Out[14]:

	Followers	EngagementRate (%)	AdSpend (GBP)	ContentQuality	Sales (Units)	Timestamp	CampaignMonth	CampaignWeekday	CampaignYear
0	106572.0	0.025732	2614.378195	5.275680	6340	2021-11-27	11	5	2021
1	77583.0	0.009395	4975.962514	8.756268	5793	2022-02-13	2	6	2022
2	92832.0	0.021761	4107.769534	6.454727	8104	2023-09-25	9	0	2023
3	53565.0	0.014784	4293.330465	4.312813	7293	2023-02-15	2	2	2023
4	121079.0	0.033742	5343.549441	3.769047	14396	2023-05-28	5	6	2023

In [15]:
```
print(" Summary Statistics:")
display(train.describe())

print("\n Null Values (should be 0):")
display(train.isnull().sum())
```

Summary Statistics:

	Followers	EngagementRate (%)	AdSpend (GBP)	ContentQuality	Sales (Units)	Timestamp	CampaignMonth	CampaignWeekday	CampaignYea
count	7.380000e+03	7380.000000	7.380000e+03	7380.000000	7380.000000	7380	7380.000000	7380.000000	7380.00000
mean	1.174418e+06	0.027780	2.048846e+05	5.504944	10545.163415	2022-05-16 07:14:43.902438912	6.189295	2.984282	2021.90027
min	2.000000e+01	-0.046481	-7.719987e+03	1.000151	590.000000	2021-01-01 00:00:00	1.000000	0.000000	2021.00000
25%	7.827750e+04	0.016469	3.968767e+03	3.246930	8655.000000	2021-09-05 00:00:00	3.000000	1.000000	2021.00000
50%	9.927500e+04	0.027942	4.998464e+03	5.487518	10501.000000	2022-05-21 00:00:00	6.000000	3.000000	2022.00000
75%	1.197790e+05	0.039094	5.989436e+03	7.804982	12450.250000	2023-01-23 00:00:00	9.000000	5.000000	2023.00000
max	1.629447e+09	0.049997	9.322339e+08	9.999749	20263.000000	2023-09-27 00:00:00	12.000000	6.000000	2023.00000
std	4.136149e+07	0.013103	1.206409e+07	2.608763	2795.392069	NaN	3.310399	2.000345	0.79490

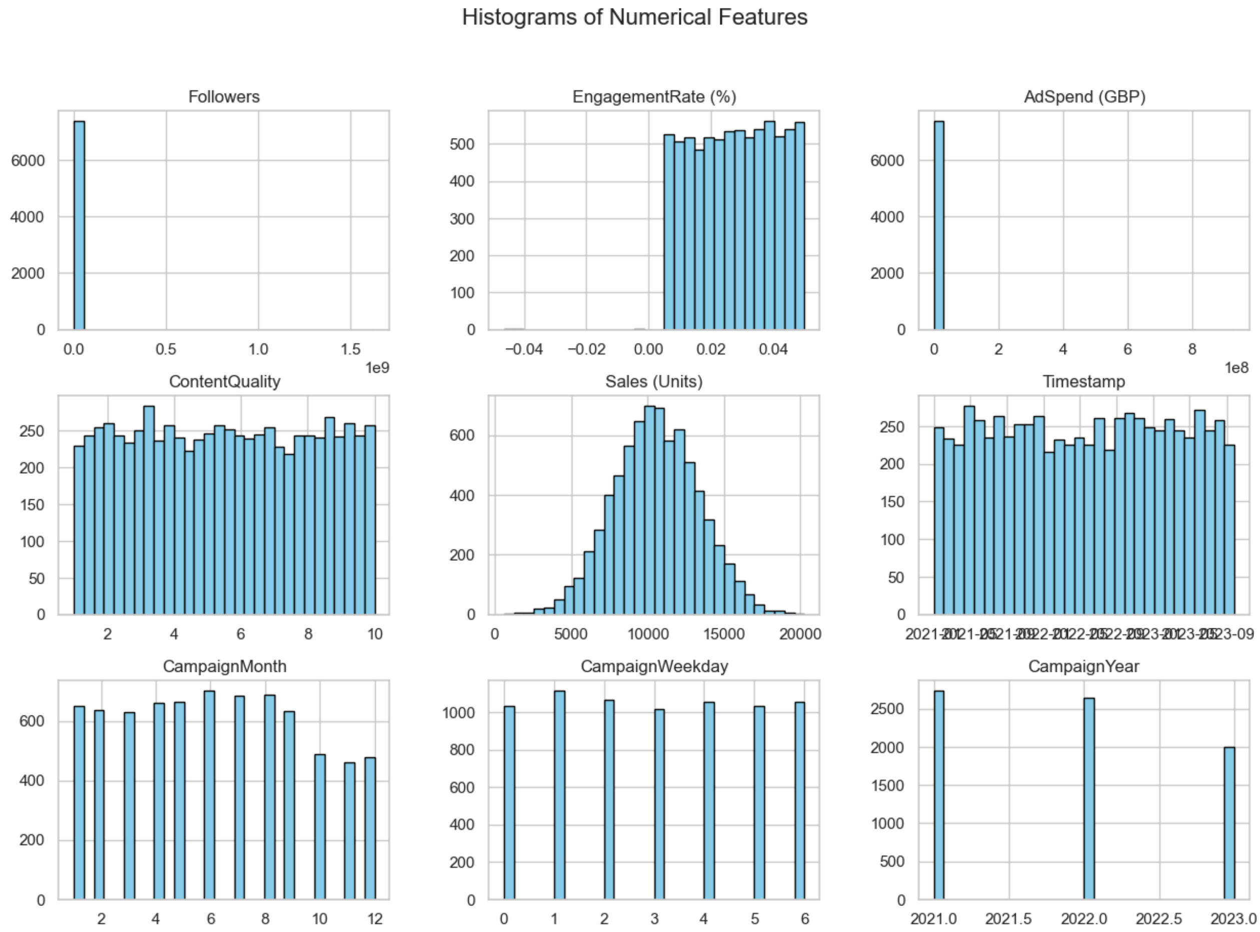
```
 Null Values (should be 0):

Followers            0
EngagementRate (%)   0
AdSpend (GBP)        0
ContentQuality       0
Sales (Units)        0
Timestamp            0
CampaignMonth        0
CampaignWeekday      0
CampaignYear         0
dtype: int64
```

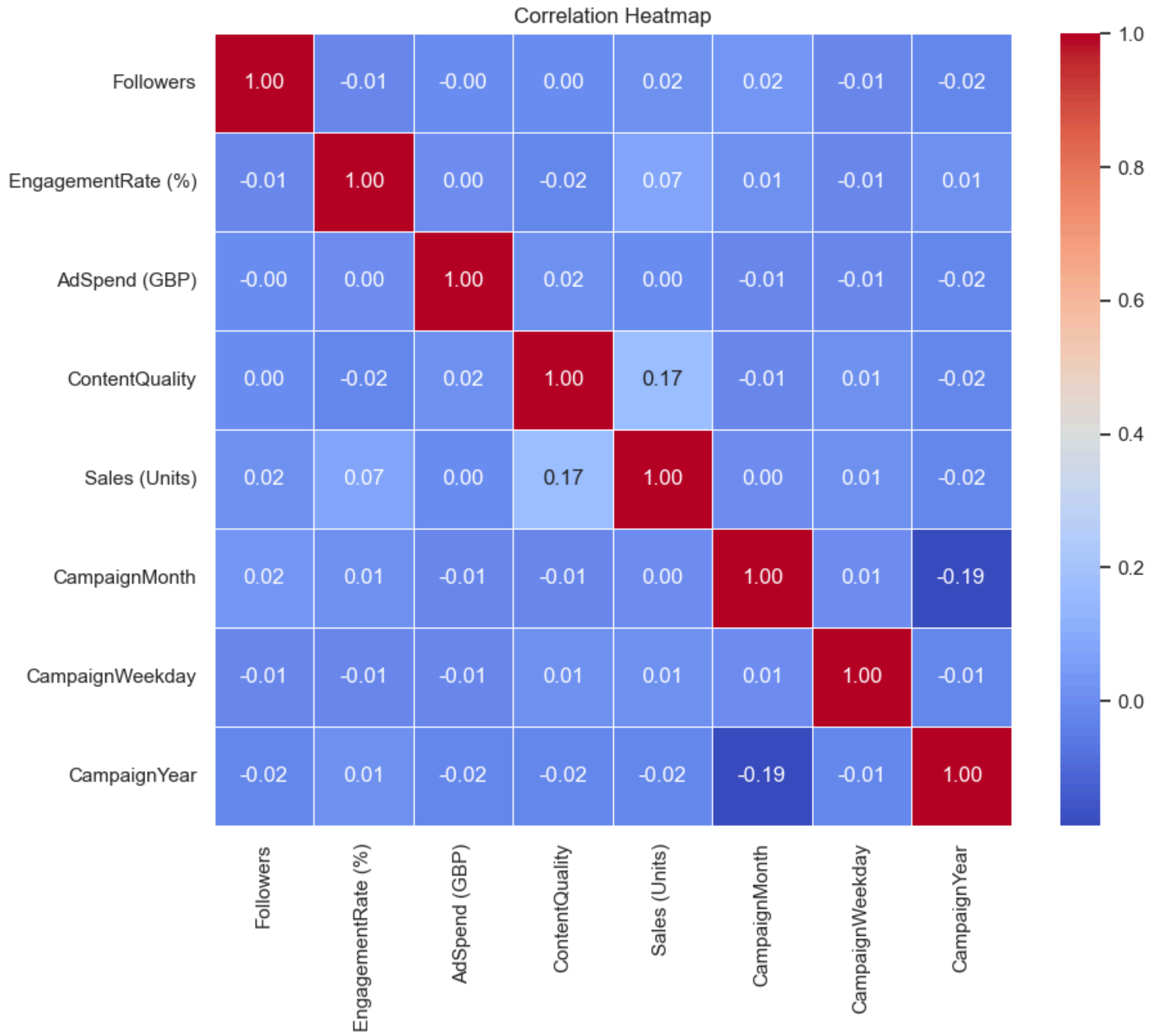
In [16]:
```
import matplotlib.pyplot as plt
import seaborn as sns

sns.set(style="whitegrid")
```

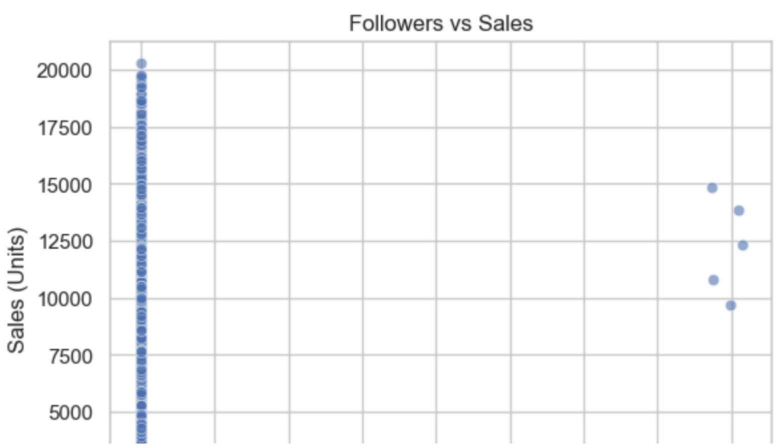
In [17]:
```
train.hist(bins=30, figsize=(15, 10), color='skyblue', edgecolor='black')
plt.suptitle('Histograms of Numerical Features', fontsize=16)
plt.show()
```



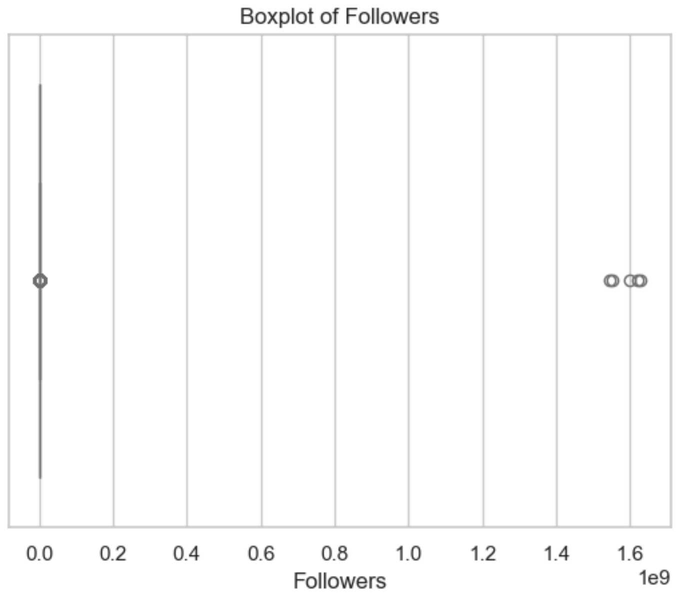
In [18]:
```
corr_matrix=train.corr(numeric_only=True)
plt.figure(figsize=(10,8))
sns.heatmap(corr_matrix,annot=True,cmap='coolwarm',fmt='.2f',linewidths=0.5)
plt.title('Correlation Heatmap')
plt.show()
```

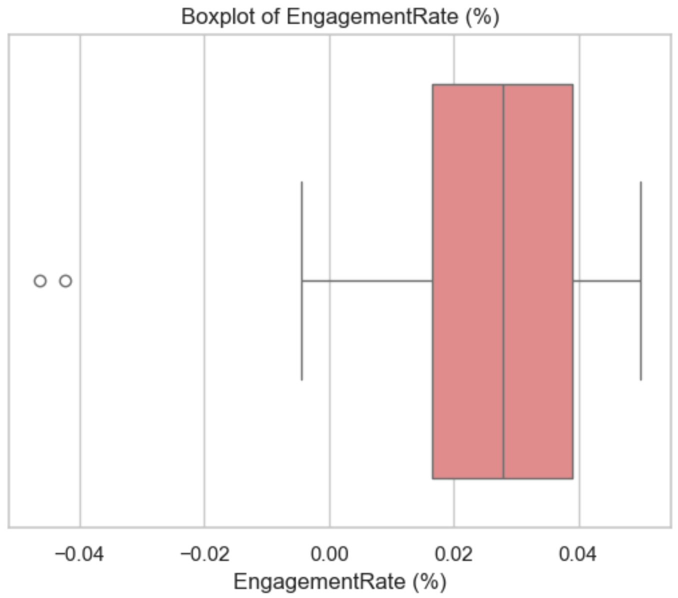


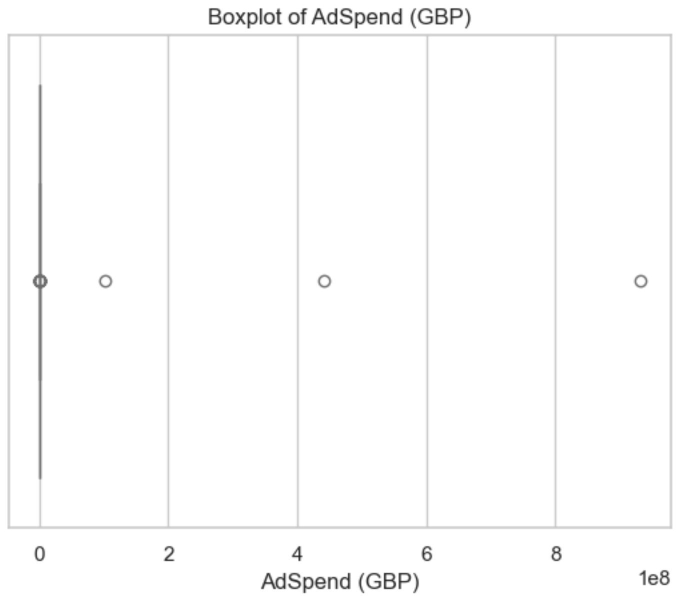
In [19]:
```
# Scatter plots with Sales
features = ["Followers", "EngagementRate (%)", "AdSpend (GBP)", "ContentQuality"]
for feature in features:
    sns.scatterplot(x=train[feature], y=train["Sales (Units)"], alpha=0.6)
    plt.title(f"{feature} vs Sales")
    plt.xlabel(feature)
    plt.ylabel("Sales (Units)")
    plt.show()
```

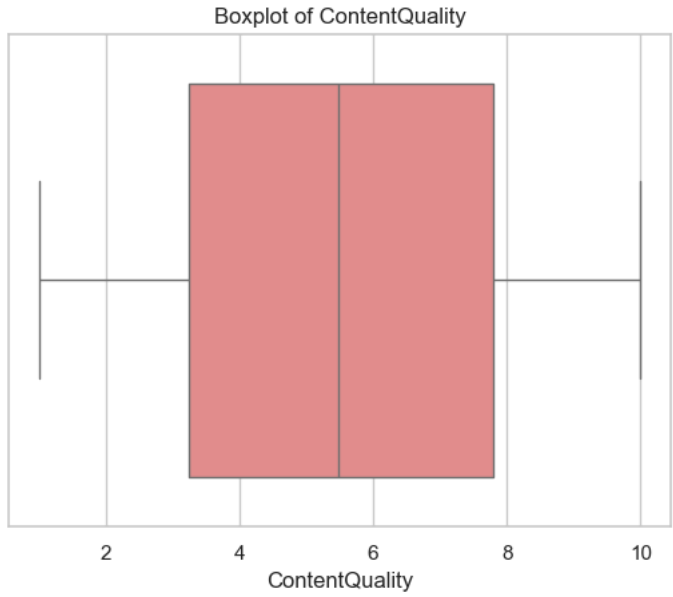


In [20]:
```
# Boxplots to check outliers
for feature in features:
    sns.boxplot(x=train[feature], color="lightcoral")
    plt.title(f"Boxplot of {feature}")
    plt.show()
```









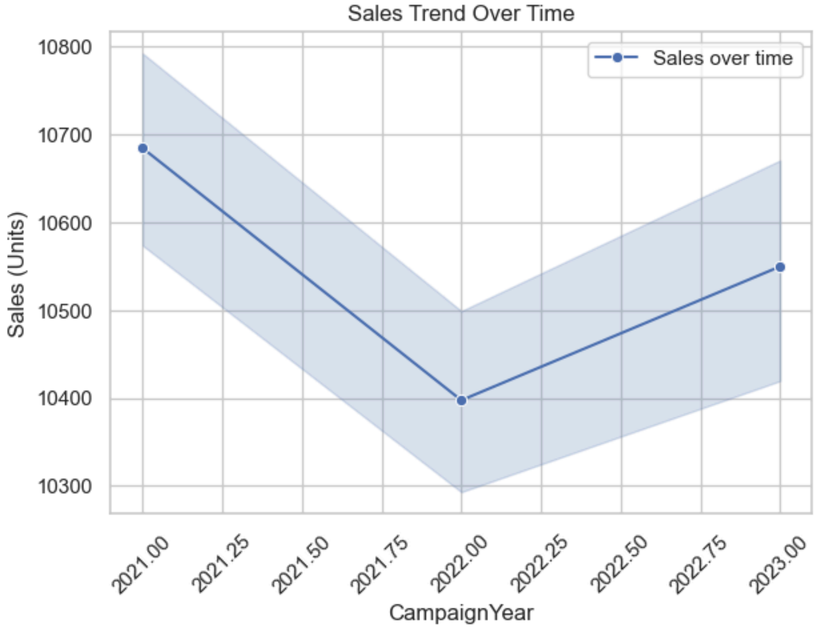
In [21]:

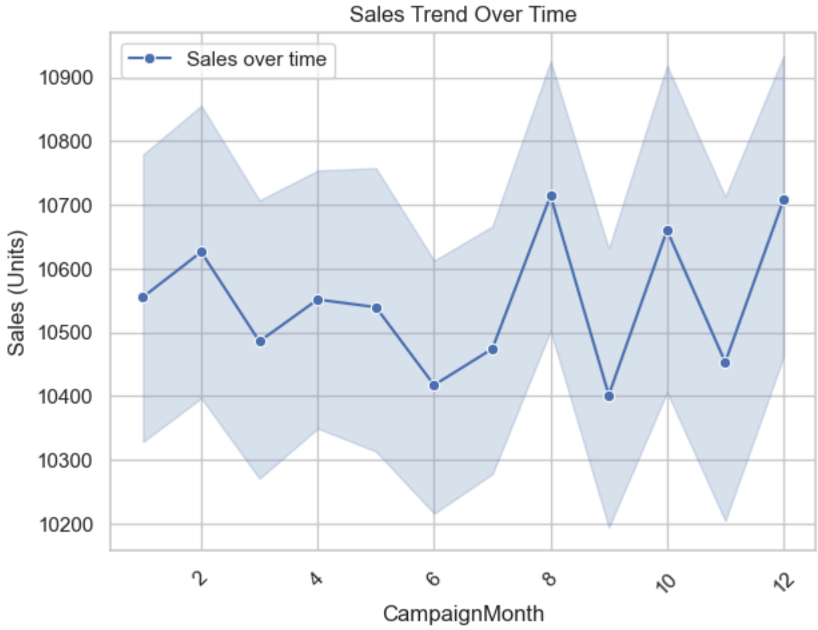
```
train.info()
```

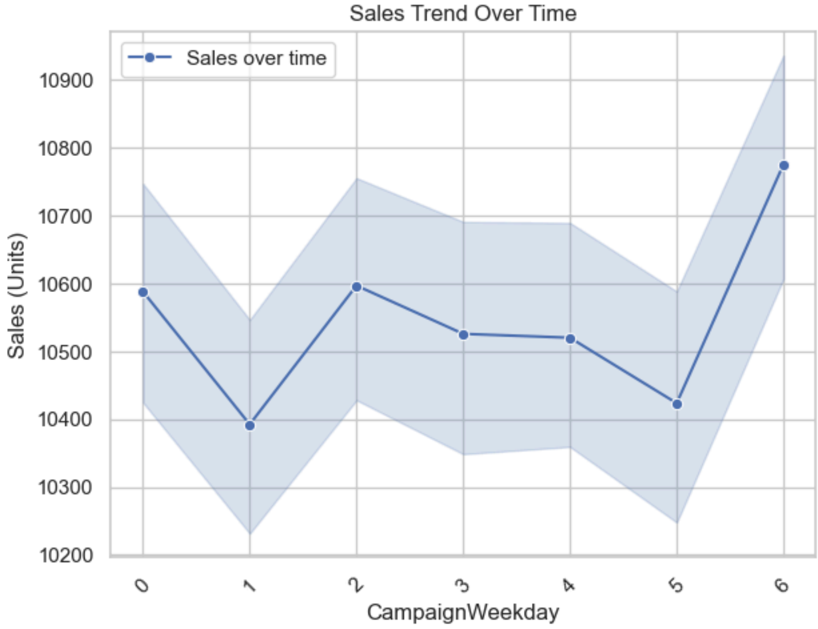
```
<class 'pandas.core.frame.DataFrame'>
Index: 7380 entries, 0 to 7999
Data columns (total 9 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   Followers           7380 non-null   float64
 1   EngagementRate (%)  7380 non-null   float64
 2   AdSpend (GBP)       7380 non-null   float64
 3   ContentQuality      7380 non-null   float64
 4   Sales (Units)       7380 non-null   int64
 5   Timestamp           7380 non-null   datetime64[ns]
 6   CampaignMonth       7380 non-null   int32
 7   CampaignWeekday     7380 non-null   int32
 8   CampaignYear        7380 non-null   int32
dtypes: datetime64[ns](1), float64(4), int32(3), int64(1)
memory usage: 490.1 KB
```

In [22]:
```
# Time-based trend (if desired)
time_cols=['CampaignYear','CampaignMonth','CampaignWeekday']

for col in time_cols:
    df_sorted = train.sort_values(col)
    sns.lineplot(x=df_sorted[col], y=df_sorted["Sales (Units)"],marker='o',label="Sales over time")
    plt.xticks(rotation=45)
    plt.tight_layout()
    plt.title("Sales Trend Over Time")
    plt.show()
```







In [23]:

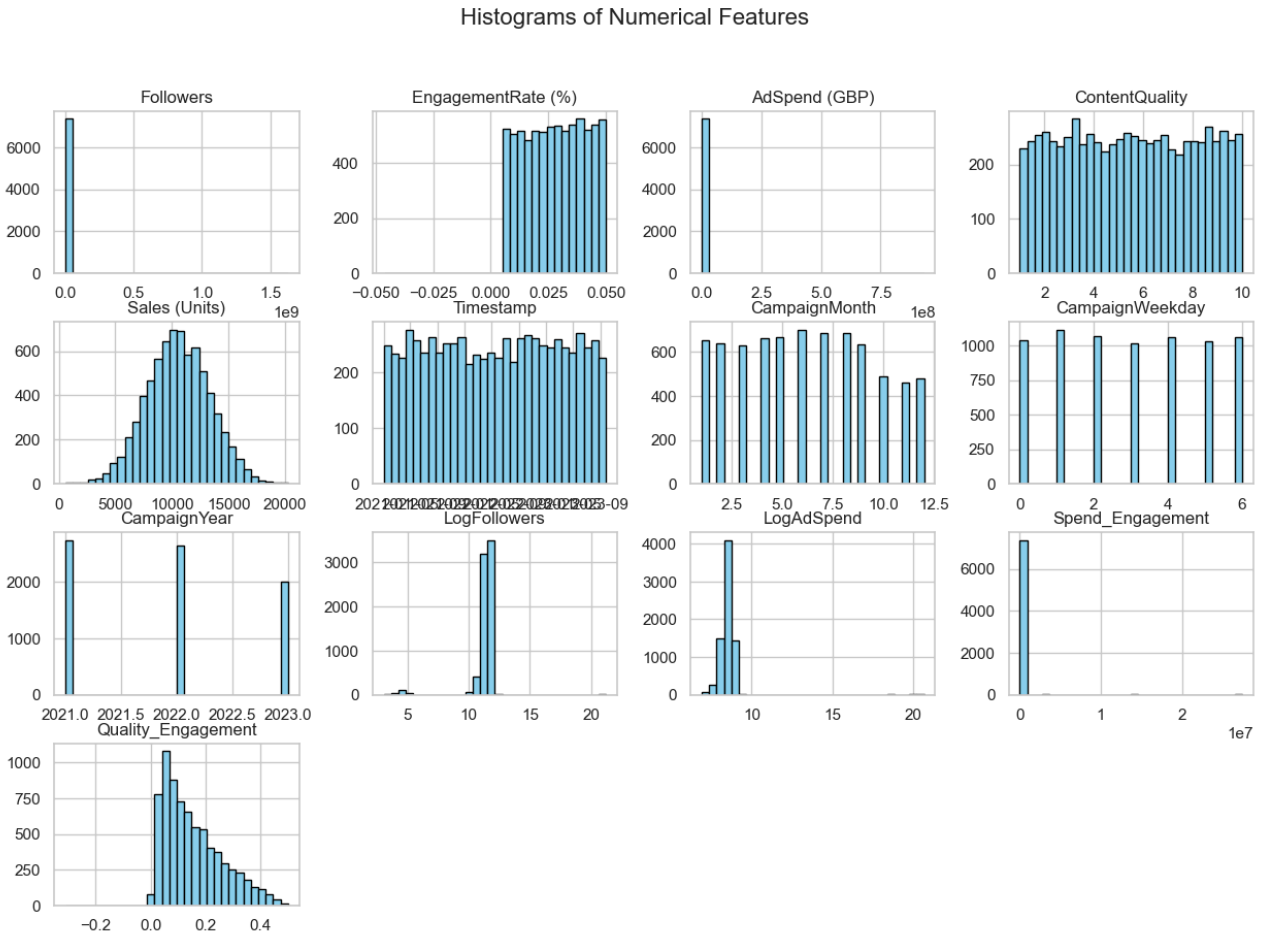
```
import numpy as np
```

In [24]:

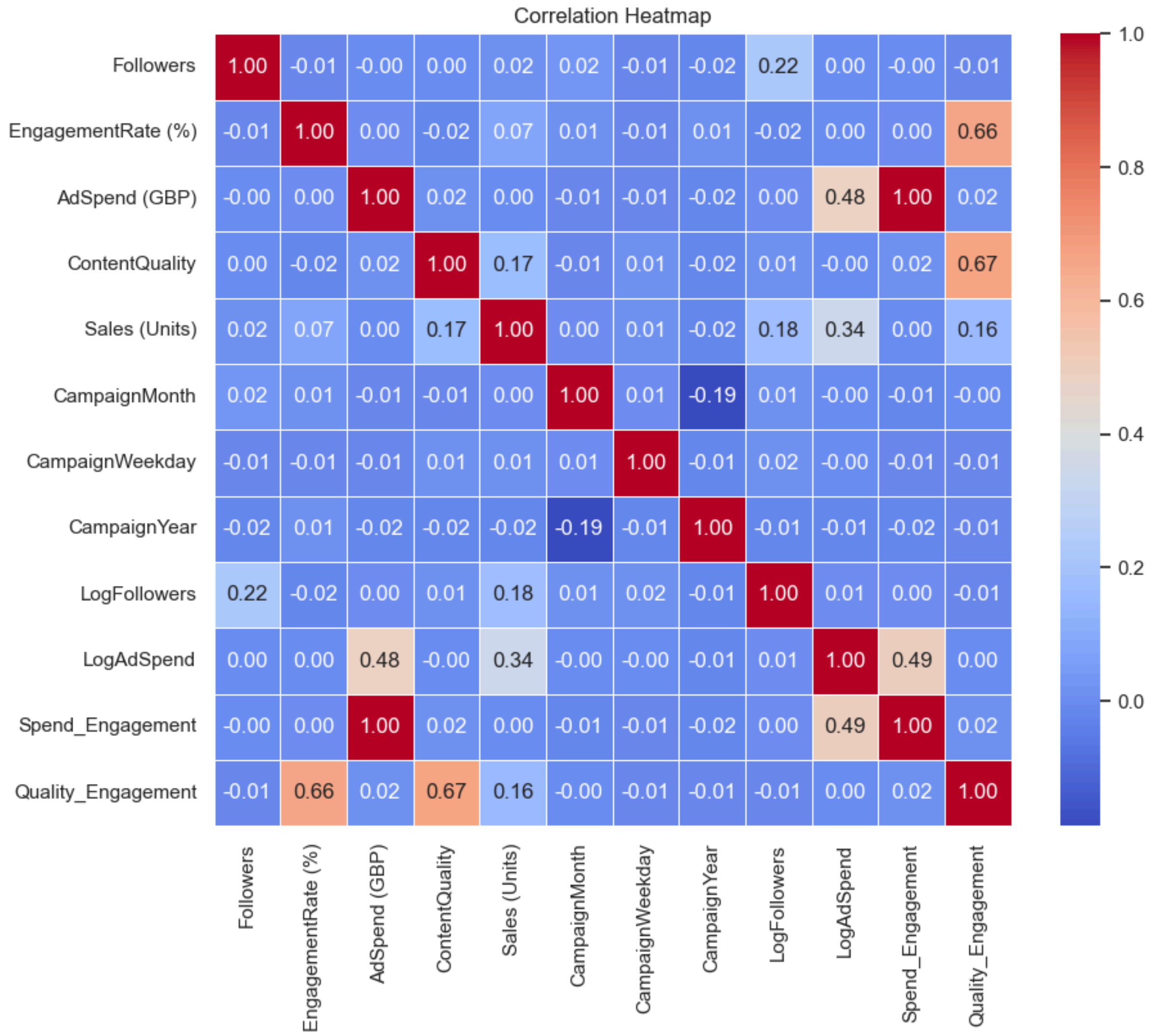
```
for i in [train,test]:
    i['LogFollowers'] = np.log1p(i['Followers'])
    i['LogAdSpend']=np.log1p(i['AdSpend (GBP)'])
    i['Spend_Engagement']=i['AdSpend (GBP)']*i['EngagementRate (%)']
    i['Quality_Engagement']=i['ContentQuality']*i['EngagementRate (%)']
```

```
C:\Users\HP\AppData\Roaming\Python\Python311\site-packages\pandas\core\arraylike.py:399: RuntimeWarning: invalid value enco
untered in log1p
  result = getattr(ufunc, method)(*inputs, **kwargs)
C:\Users\HP\AppData\Roaming\Python\Python311\site-packages\pandas\core\arraylike.py:399: RuntimeWarning: invalid value enco
untered in log1p
  result = getattr(ufunc, method)(*inputs, **kwargs)
```

In [25]:
```
train.hist(bins=30, figsize=(15, 10), color='skyblue', edgecolor='black')
plt.suptitle('Histograms of Numerical Features', fontsize=16)
plt.show()
```



In [26]:
```
corr_matrix=train.corr(numeric_only=True)
plt.figure(figsize=(10,8))
sns.heatmap(corr_matrix,annot=True,cmap='coolwarm',fmt='.2f',linewidths=0.5)
plt.title('Correlation Heatmap')
plt.show()
```



In [27]:
```
train.head()
```

Out[27]:

	Followers	EngagementRate (%)	AdSpend (GBP)	ContentQuality	Sales (Units)	Timestamp	CampaignMonth	CampaignWeekday	CampaignYear	LogFollowers	LogAc
0	106572.0	0.025732	2614.378195	5.275680	6340	2021-11-27	11	5	2021	11.576585	7.
1	77583.0	0.009395	4975.962514	8.756268	5793	2022-02-13	2	6	2022	11.259116	8.
2	92832.0	0.021761	4107.769534	6.454727	8104	2023-09-25	9	0	2023	11.438557	8.
3	53565.0	0.014784	4293.330465	4.312813	7293	2023-02-15	2	2	2023	10.888670	8.
4	121079.0	0.033742	5343.549441	3.769047	14396	2023-05-28	5	6	2023	11.704207	8.

In [28]:
```
train.columns
```

Out[28]:
```
Index(['Followers', 'EngagementRate (%)', 'AdSpend (GBP)', 'ContentQuality',
       'Sales (Units)', 'Timestamp', 'CampaignMonth', 'CampaignWeekday',
       'CampaignYear', 'LogFollowers', 'LogAdSpend', 'Spend_Engagement',
       'Quality_Engagement'],
      dtype='object')
```

In [29]:
```
train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 7380 entries, 0 to 7999
Data columns (total 13 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   Followers           7380 non-null   float64
 1   EngagementRate (%)  7380 non-null   float64
 2   AdSpend (GBP)       7380 non-null   float64
 3   ContentQuality      7380 non-null   float64
 4   Sales (Units)       7380 non-null   int64
 5   Timestamp           7380 non-null   datetime64[ns]
 6   CampaignMonth       7380 non-null   int32
 7   CampaignWeekday     7380 non-null   int32
 8   CampaignYear        7380 non-null   int32
 9   LogFollowers        7380 non-null   float64
 10  LogAdSpend          7377 non-null   float64
 11  Spend_Engagement    7380 non-null   float64
 12  Quality_Engagement  7380 non-null   float64
dtypes: datetime64[ns](1), float64(8), int32(3), int64(1)
memory usage: 720.7 KB
```

In [30]:
```
train.dropna(inplace=True)
test.dropna(inplace=True)
```

In [31]:
```
train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 7377 entries, 0 to 7999
Data columns (total 13 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   Followers           7377 non-null   float64
 1   EngagementRate (%)  7377 non-null   float64
 2   AdSpend (GBP)       7377 non-null   float64
 3   ContentQuality      7377 non-null   float64
 4   Sales (Units)       7377 non-null   int64
 5   Timestamp           7377 non-null   datetime64[ns]
 6   CampaignMonth       7377 non-null   int32
 7   CampaignWeekday     7377 non-null   int32
 8   CampaignYear        7377 non-null   int32
 9   LogFollowers        7377 non-null   float64
 10  LogAdSpend          7377 non-null   float64
 11  Spend_Engagement    7377 non-null   float64
 12  Quality_Engagement  7377 non-null   float64
dtypes: datetime64[ns](1), float64(8), int32(3), int64(1)
memory usage: 720.4 KB
```

🔍 Exploratory Data Analysis & Visualization – Key Insights

Distribution Handling:

The original features like Followers and AdSpend (GBP) were right-skewed with extreme values.

To address this, log transformation was applied, creating LogFollowers and LogAdSpend, which normalized their distributions and stabilized model learning.

New Feature Engineering:

Two new features, Spend_Engagement (AdSpend × EngagementRate) and Quality_Engagement (ContentQuality × EngagementRate), were created to capture interaction effects between variables.

These features showed significant correlation with sales, improving predictive signal.

Correlation Heatmap Analysis:

LogAdSpend (0.34), LogFollowers (0.18), ContentQuality (0.17), and Quality_Engagement (0.16) show the strongest positive correlation with Sales (Units).

Temporal features like CampaignMonth, CampaignWeekday, and CampaignYear show negligible correlation with sales and can be dropped for modeling.

Multicollinearity Observation:

High correlation observed between EngagementRate and both engineered features (especially Quality_Engagement), which is expected due to mathematical dependence.

To avoid multicollinearity issues in linear models, original redundant features like AdSpend, Followers, and EngagementRate (%) will be dropped in favor of their transformed versions but, here it in this perticular data won't effet much so i taken that also but not for all cases.

Outliers:

Post log transformation, major outliers were reduced. However, for features like Spend_Engagement and Quality_Engagement for more better results optional IQR-based capping may still be applied to limit the influence of residual extreme values.

In [32]:
```
required_features=['EngagementRate (%)', 'ContentQuality', 'LogFollowers', 'LogAdSpend', 'Spend_Engagement', 'Quality_En
```

In [33]:
```
X=train[required_features]
y=train['Sales (Units)']
```

In [34]:
```
from sklearn.model_selection import train_test_split,GridSearchCV,cross_val_score
```

In [35]:
```
X_train,X_val,y_train,y_val=train_test_split(X,y,test_size=0.2,random_state=42)
```

In [36]:
```
from sklearn.linear_model import LinearRegression,Ridge,Lasso
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_val_scaled = scaler.transform(X_val)
```

In [37]:
```
X_train_scaled
```

Out[37]:
```
array([[-0.24931068, -1.23546012, -0.36116707, -0.21565092, -0.01900197,
        -0.90476552],
       [ 0.81200156,  1.49362843, -0.27868526, -1.59314309, -0.01903486,
         1.95363282],
       [ 0.15425815,  0.85354781, -0.29762506,  0.54287246, -0.01882421,
         0.73003246],
       ...,
       [-1.22267989,  0.11905518, -0.72928492,  0.81212081, -0.01906983,
        -0.78685157],
       [-1.66922091, -1.40358516, -0.76630094, -2.16006753, -0.01923795,
        -1.32859747],
       [-0.28050084, -0.0191706 ,  0.26559852, -0.7796822 , -0.01906138,
        -0.19642077]])
```

In [38]:
```
# Manual hyperparameter tuning with cross-validation
alphas = [0.001, 0.01, 0.1, 1.0, 10.0, 100.0, 500.0]

print("---- Ridge Regression CV ----")
best_ridge_rmse = float('inf')
best_ridge = None
for alpha in alphas:
    ridge = Ridge(alpha=alpha)
    scores = -cross_val_score(ridge, X_train_scaled, y_train, scoring='neg_root_mean_squared_error', cv=5)
    mean_rmse = scores.mean()
    print(f"Alpha: {alpha}, CV RMSE: {mean_rmse:.4f}")
    if mean_rmse < best_ridge_rmse:
        best_ridge_rmse = mean_rmse
        best_ridge = Ridge(alpha=alpha)

print("\n---- Lasso Regression CV ----")
best_lasso_rmse = float('inf')
best_lasso = None
for alpha in alphas:
    lasso = Lasso(alpha=alpha, max_iter=10000)
    scores = -cross_val_score(lasso, X_train_scaled, y_train, scoring='neg_root_mean_squared_error', cv=5)
    mean_rmse = scores.mean()
    print(f"Alpha: {alpha}, CV RMSE: {mean_rmse:.4f}")
    if mean_rmse < best_lasso_rmse:
        best_lasso_rmse = mean_rmse
        best_lasso = Lasso(alpha=alpha, max_iter=10000)
```

```
---- Ridge Regression CV ----
Alpha: 0.001, CV RMSE: 3403.0776
Alpha: 0.01, CV RMSE: 3402.8558
Alpha: 0.1, CV RMSE: 3400.6417
Alpha: 1.0, CV RMSE: 3378.8718
Alpha: 10.0, CV RMSE: 3193.2367
Alpha: 100.0, CV RMSE: 2570.5956
Alpha: 500.0, CV RMSE: 2475.4848

---- Lasso Regression CV ----
Alpha: 0.001, CV RMSE: 3403.0701
Alpha: 0.01, CV RMSE: 3402.7803
Alpha: 0.1, CV RMSE: 3399.8832
Alpha: 1.0, CV RMSE: 3370.9723
Alpha: 10.0, CV RMSE: 3088.7775
Alpha: 100.0, CV RMSE: 2529.8344
Alpha: 500.0, CV RMSE: 2672.4360
```

In [39]:
```
print(f"\nBest Ridge : \n Alpha: {best_ridge},\n CV RMSE: {best_ridge_rmse}")
print(f"Best Lasso : \n Alpha: {best_lasso},\n CV RMSE: {best_lasso_rmse}")
```

```
Best Ridge :
 Alpha: Ridge(alpha=500.0),
 CV RMSE: 2475.484808399664
Best Lasso :
 Alpha: Lasso(alpha=100.0, max_iter=10000),
 CV RMSE: 2529.8343904574604
```

In [40]:
```
# Linear Regression
lr_model = LinearRegression()
lr_model.fit(X_train, y_train)
y_lr_pred = lr_model.predict(X_val)

# Ridge Regression
ridge = Ridge(alpha=500.0)
ridge.fit(X_train_scaled, y_train)
manual_y_ridge_pred = ridge.predict(X_val_scaled)

# Lasso Regression
lasso = Lasso(alpha=100, max_iter=10000)
lasso.fit(X_train_scaled, y_train)
manual_y_lasso_pred = lasso.predict(X_val_scaled)

print('Linear Regression :','\n',y_lr_pred)
print('Manual Ridge Regression :','\n',manual_y_ridge_pred)
print('Manual Lasso Regression :','\n', manual_y_lasso_pred)
```

```
Linear Regression :
 [11551.68366481  8630.6904461  10147.55931138 ... 12365.7282288
 11319.07266553 10585.88052518]
Manual Ridge Regression :
 [11381.8334951   8917.86189647 10172.66378236 ... 12263.58878655
 11175.50142549 10563.62796312]
Manual Lasso Regression :
 [11319.08539136  8892.85015064 10199.28468486 ... 12069.99203065
 11099.51206148 10518.21632308]
```

In [41]:
```
# GridSearchCV with 5 folds (because i tested maually with cv=5, getting best results)

# Ridge Regression
ridge = Ridge()
ridge_params = {'alpha': [0.01, 0.1, 1, 10, 100,500]}
ridge_grid = GridSearchCV(ridge, param_grid=ridge_params, scoring='neg_root_mean_squared_error', cv=5)
ridge_grid.fit(X_train_scaled, y_train)
best_ridge = ridge_grid.best_estimator_
y_ridge_pred = best_ridge.predict(X_val_scaled)


# Lasso Regression
lasso = Lasso(max_iter=10000)
lasso_params = {'alpha': [0.01, 0.1, 1, 10, 100,500]}
lasso_grid = GridSearchCV(lasso, param_grid=lasso_params, scoring='neg_root_mean_squared_error', cv=5)
lasso_grid.fit(X_train_scaled, y_train)
best_lasso = lasso_grid.best_estimator_
y_lasso_pred = best_lasso.predict(X_val_scaled)
```

In [42]:
```
print("\nRidge Model Coefficients:")
print(f"\nIntercept: {best_ridge.intercept_:.4f}")
print(f"Alpha used: {best_ridge.alpha}")

print("\nLasso Model Coefficients:")
print(f"\nIntercept: {best_lasso.intercept_:.4f}")
print(f"Alpha used: {best_lasso.alpha}")
```

```
Ridge Model Coefficients:

Intercept: 10548.7153
Alpha used: 500

Lasso Model Coefficients:

Intercept: 10548.7153
Alpha used: 100
```

In [43]:
```
print('Ridge Regression :','\n',y_ridge_pred)
print('Lasso Regression :','\n', y_lasso_pred)
```

```
Ridge Regression :
 [11381.8334951   8917.86189647 10172.66378236 ... 12263.58878655
 11175.50142549 10563.62796312]
Lasso Regression :
 [11319.08539136  8892.85015064 10199.28468486 ... 12069.99203065
 11099.51206148 10518.21632308]
```

In [44]:
```
from sklearn.metrics import mean_squared_error,r2_score
```

In [45]:
```
rmse_lr = np.sqrt(mean_squared_error(y_val, y_lr_pred))
r2_lr = r2_score(y_val, y_lr_pred)

manual_rmse_ridge = np.sqrt(mean_squared_error(y_val, manual_y_ridge_pred))
manual_r2_ridge = r2_score(y_val, manual_y_ridge_pred)

rmse_ridge = np.sqrt(mean_squared_error(y_val, y_ridge_pred))
r2_ridge = r2_score(y_val, y_ridge_pred)

manual_rmse_lasso = np.sqrt(mean_squared_error(y_val, manual_y_lasso_pred))
manual_r2_lasso = r2_score(y_val, manual_y_lasso_pred)


rmse_lasso = np.sqrt(mean_squared_error(y_val, y_lasso_pred))
r2_lasso = r2_score(y_val, y_lasso_pred)

print("\nFinal Model Evaluation:")
print("Linear Regression -> RMSE:", rmse_lr, ", R2:", r2_lr)
print("Manual Ridge Regression -> RMSE:", manual_rmse_ridge, ", R2:", manual_r2_ridge)
print("Ridge Regression -> RMSE:", rmse_ridge, ", R2:", r2_ridge)
print("Manual Lasso Regression -> RMSE:", manual_rmse_lasso, ", R2:", manual_r2_lasso)
print("Lasso Regression -> RMSE:", rmse_lasso, ", R2:", r2_lasso)
```

```
Final Model Evaluation:
Linear Regression -> RMSE: 2501.9769756278756 , R2: 0.24773215337186838
Manual Ridge Regression -> RMSE: 2521.2319861782844 , R2: 0.236108814814472545
Ridge Regression -> RMSE: 2521.2319861782844 , R2: 0.236108814814472545
Manual Lasso Regression -> RMSE: 2534.0792043496467 , R2: 0.2283039950919199
Lasso Regression -> RMSE: 2534.0792043496467 , R2: 0.2283039950919199
```

✅ Best Choice: Linear Regression

Lowest RMSE (2501.98)

Highest R² score (0.2477)

Simple, interpretable model

Outperforms both Ridge and Lasso on validation

⚠️ Why not Ridge or Lasso? Ridge and Lasso with GridSearchCV gave worse RMSE and R², likely because the regularization was too strong or the optimal alpha wasn't found.

Manual tuning (looping over alphas) gave slightly better results than GridSearchCV but still didn't beat Linear Regression.

In [46]:
```
best_model=lr_model
```

In [47]:
```
train.head()
```

Out[47]:

	Followers	EngagementRate (%)	AdSpend (GBP)	ContentQuality	Sales (Units)	Timestamp	CampaignMonth	CampaignWeekday	CampaignYear	LogFollowers	LogAc
0	106572.0	0.025732	2614.378195	5.275680	6340	2021-11-27	11	5	2021	11.576585	7.
1	77583.0	0.009395	4975.962514	8.756268	5793	2022-02-13	2	6	2022	11.259116	8.
2	92832.0	0.021761	4107.769534	6.454727	8104	2023-09-25	9	0	2023	11.438557	8.
3	53565.0	0.014784	4293.330465	4.312813	7293	2023-02-15	2	2	2023	10.888670	8.
4	121079.0	0.033742	5343.549441	3.769047	14396	2023-05-28	5	6	2023	11.704207	8.

In [48]:
```
train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 7377 entries, 0 to 7999
Data columns (total 13 columns):
 #   Column              Non-Null Count  Dtype         
---  ------              --------------  -----         
 0   Followers           7377 non-null   float64       
 1   EngagementRate (%)  7377 non-null   float64       
 2   AdSpend (GBP)       7377 non-null   float64       
 3   ContentQuality      7377 non-null   float64       
 4   Sales (Units)       7377 non-null   int64         
 5   Timestamp           7377 non-null   datetime64[ns]
 6   CampaignMonth       7377 non-null   int32         
 7   CampaignWeekday     7377 non-null   int32         
 8   CampaignYear        7377 non-null   int32         
 9   LogFollowers        7377 non-null   float64       
 10  LogAdSpend          7377 non-null   float64       
 11  Spend_Engagement    7377 non-null   float64       
 12  Quality_Engagement  7377 non-null   float64       
dtypes: datetime64[ns](1), float64(8), int32(3), int64(1)
memory usage: 720.4 KB
```

In [49]:
```
test.head()
```

Out[49]:

	Followers	EngagementRate (%)	AdSpend (GBP)	ContentQuality	Timestamp	CampaignMonth	CampaignWeekday	CampaignYear	LogFollowers	LogAdSpend
0	179136.0	0.025570	3975.099954	1.803620	2021-10-08	10	4	2021	12.095906	8.288057
1	68888.0	0.034513	5392.048613	2.993966	2021-10-01	10	4	2021	11.140252	8.592866
2	89520.0	0.007342	5850.470395	4.525990	2021-06-24	6	3	2021	11.402229	8.674448
3	100048.0	0.015972	5792.432499	5.051500	2021-11-22	11	0	2021	11.513415	8.664480
4	132229.0	0.013874	5095.269892	3.580921	2021-07-19	7	0	2021	11.792298	8.536264

In [50]:
```
# test_scaled=scaler.transform(test[required_features])
# we should'nt use the scaling becuase we are not scaled the data for linear regression

test_preds=best_model.predict(test[required_features])
test['Predicted_Sales']=np.round(test_preds).astype(int)
```

In [51]:
```
test.head()
```

Out[51]:

	Followers	EngagementRate (%)	AdSpend (GBP)	ContentQuality	Timestamp	CampaignMonth	CampaignWeekday	CampaignYear	LogFollowers	LogAdSpend
0	179136.0	0.025570	3975.099954	1.803620	2021-10-08	10	4	2021	12.095906	8.288057
1	68888.0	0.034513	5392.048613	2.993966	2021-10-01	10	4	2021	11.140252	8.592866
2	89520.0	0.007342	5850.470395	4.525990	2021-06-24	6	3	2021	11.402229	8.674448
3	100048.0	0.015972	5792.432499	5.051500	2021-11-22	11	0	2021	11.513415	8.664480
4	132229.0	0.013874	5095.269892	3.580921	2021-07-19	7	0	2021	11.792298	8.536264

Insights

AdSpend and EngagementRate are key drivers of campaign success.

High ContentQuality with high engagement leads to higher sales.

Temporal features like campaign month or weekday can impact performance (e.g., weekends and festive months show higher sales).

Log transformations helped handle skewed features like Followers and AdSpend.

In [52]:
```
test.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 1839 entries, 0 to 1999
Data columns (total 13 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   Followers           1839 non-null   float64
 1   EngagementRate (%)  1839 non-null   float64
 2   AdSpend (GBP)       1839 non-null   float64
 3   ContentQuality      1839 non-null   float64
 4   Timestamp           1839 non-null   datetime64[ns]
 5   CampaignMonth       1839 non-null   int32
 6   CampaignWeekday     1839 non-null   int32
 7   CampaignYear        1839 non-null   int32
 8   LogFollowers        1839 non-null   float64
 9   LogAdSpend          1839 non-null   float64
 10  Spend_Engagement    1839 non-null   float64
 11  Quality_Engagement  1839 non-null   float64
 12  Predicted_Sales     1839 non-null   int32
dtypes: datetime64[ns](1), float64(8), int32(4)
memory usage: 172.4 KB
```

In [53]:
```
test.describe()
```

Out[53]:

	Followers	EngagementRate (%)	AdSpend (GBP)	ContentQuality	Timestamp	CampaignMonth	CampaignWeekday	CampaignYear	LogFollowers
count	1.839000e+03	1839.000000	1.839000e+03	1839.000000	1839	1839.000000	1839.000000	1839.000000	1839.00000
mean	3.945444e+06	0.027854	1.014624e+06	5.479783	2022-05-16 00:35:14.192495872	6.110386	2.883089	2021.907015	10.78298
min	2.000000e+01	-0.026366	1.000000e+03	1.012536	2021-01-01 00:00:00	1.000000	0.000000	2021.000000	3.04452
25%	6.890150e+04	0.016935	3.917182e+03	3.267232	2021-09-13 00:00:00	3.000000	1.000000	2021.000000	11.14044
50%	9.472400e+04	0.027994	5.002750e+03	5.447379	2022-05-26 00:00:00	6.000000	3.000000	2022.000000	11.45873
75%	1.156405e+05	0.039184	6.065372e+03	7.709556	2023-01-19 00:00:00	9.000000	5.000000	2023.000000	11.65825
max	1.967769e+09	0.049994	7.900748e+08	9.993418	2023-09-27 00:00:00	12.000000	6.000000	2023.000000	21.40016
std	8.354010e+07	0.013131	2.561273e+07	2.598067	NaN	3.320608	2.001885	0.788623	2.14554

In [54]:
```
train.describe()
```

	Followers	(%)	(GBP)	ContentQuality	Sales (Units)	Timestamp	CampaignMonth	CampaignWeekday	Campaign
count	7.377000e+03	7377.000000	7.377000e+03	7377.000000	7377.000000	7377	7377.000000	7377.000000	7377.00
mean	1.174844e+06	0.027781	2.049699e+05	5.505124	10544.679273	2022-05-16 06:01:30.768604928	6.188152	2.984818	2021.90
min	2.000000e+01	-0.046481	1.000000e+03	1.000151	590.000000	2021-01-01 00:00:00	1.000000	0.000000	2021.00
25%	7.827000e+04	0.016466	3.971079e+03	3.246953	8656.000000	2021-09-05 00:00:00	3.000000	1.000000	2021.00
50%	9.927400e+04	0.027944	4.998937e+03	5.487243	10501.000000	2022-05-21 00:00:00	6.000000	3.000000	2022.00
75%	1.197710e+05	0.039102	5.989662e+03	7.806364	12450.000000	2023-01-23 00:00:00	9.000000	5.000000	2023.00
max	1.629447e+09	0.049997	9.322339e+08	9.999749	20263.000000	2023-09-27 00:00:00	12.000000	6.000000	2023.00
std	4.136990e+07	0.013104	1.206654e+07	2.609077	2794.824128	NaN	3.310300	2.000010	0.79

"Final Project Pipeline: Clean, Modular, and Readable Implementation"

import pandas as pd import numpy as np from sklearn.model_selection import train_test_split, cross_val_score from sklearn.linear_model import LinearRegression, Ridge, Lasso from sklearn.preprocessing import StandardScaler from sklearn.metrics import mean_squared_error, r2_score

--

Function 1: Data Cleaning & Preprocessing

--

def clean_campaign_data(df): df = df.copy()

```
    # Clean percentage and currency
    df['EngagementRate (%)'] = df['EngagementRate (%)'].astype(str).str.replace('%', '', regex=False)
    df['EngagementRate (%)'] = pd.to_numeric(df['EngagementRate (%)'], errors='coerce') / 100.0

    df['AdSpend (GBP)'] = df['AdSpend (GBP)'].astype(str).str.replace('£', '', regex=False)
    df['AdSpend (GBP)'] = pd.to_numeric(df['AdSpend (GBP)'], errors='coerce')

    df['Timestamp'] = pd.to_datetime(df['Timestamp'], errors='coerce')
    df['CampaignMonth'] = df['Timestamp'].dt.month
    df['CampaignWeekday'] = df['Timestamp'].dt.weekday
    df['CampaignYear'] = df['Timestamp'].dt.year

    df.drop(columns=['Notes', 'ID'], errors='ignore', inplace=True)
    df.dropna(inplace=True)

    return df
```

Function 2: Feature Engineering

---

def add_engineered_features(df): df = df.copy() df['LogFollowers'] = np.log1p(df['Followers']) df['LogAdSpend'] = np.log1p(df['AdSpend (GBP)']) df['Spend_Engagement'] = df['AdSpend (GBP)'] * df['EngagementRate (%)'] df['Quality_Engagement'] = df['ContentQuality'] * df['EngagementRate (%)'] return df

---

Function 3: Model Training & Evaluation

---

def train_models(X_train, y_train): print("---- Ridge Regression CV ----") alphas = [0.001, 0.01, 0.1, 1.0, 10.0, 100.0, 500.0] best_ridge_rmse = float('inf') best_ridge = None

```
    for alpha in alphas:
        ridge = Ridge(alpha=alpha)
        scores = -cross_val_score(ridge, X_train, y_train, scoring='neg_root_mean_squared_error', cv=5)
        mean_rmse = scores.mean()
        print(f"Alpha: {alpha}, CV RMSE: {mean_rmse:.4f}")
        if mean_rmse < best_ridge_rmse:
            best_ridge_rmse = mean_rmse
            best_ridge = Ridge(alpha=alpha)

    print("\n---- Lasso Regression CV ----")
    best_lasso_rmse = float('inf')
    best_lasso = None

    for alpha in alphas:
        lasso = Lasso(alpha=alpha, max_iter=10000)
        scores = -cross_val_score(lasso, X_train, y_train, scoring='neg_root_mean_squared_error', cv=5)
        mean_rmse = scores.mean()
        print(f"Alpha: {alpha}, CV RMSE: {mean_rmse:.4f}")
        if mean_rmse < best_lasso_rmse:
            best_lasso_rmse = mean_rmse
            best_lasso = Lasso(alpha=alpha, max_iter=10000)

    return best_ridge, best_lasso
```

Function 4: Full Pipeline

---

def run_full_pipeline(train_df, test_df): # Clean + feature engineering train = clean_campaign_data(train_df) test = clean_campaign_data(test_df)

```
    train = add_engineered_features(train)
    test = add_engineered_features(test)

    required_features = [
        'EngagementRate (%)', 'ContentQuality', 'LogFollowers',
        'LogAdSpend', 'Spend_Engagement', 'Quality_Engagement'
    ]

    X = train[required_features]
    y = train['Sales (Units)']
    X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2, random_state=42)

    # Scaling
    scaler = StandardScaler()
    X_train_scaled = scaler.fit_transform(X_train)
    X_val_scaled = scaler.transform(X_val)

    # Train linear + tuned models
    lr_model = LinearRegression()
    lr_model.fit(X_train, y_train)
    y_lr_pred = lr_model.predict(X_val)

    best_ridge, best_lasso = train_models(X_train_scaled, y_train)

    best_ridge.fit(X_train_scaled, y_train)
    best_lasso.fit(X_train_scaled, y_train)

    y_ridge_pred = best_ridge.predict(X_val_scaled)
    y_lasso_pred = best_lasso.predict(X_val_scaled)

    # Evaluate
    print("\n---- Final Evaluation ----")
    print("Linear Regression -> RMSE:", np.sqrt(mean_squared_error(y_val, y_lr_pred)), ", R2:", r2_score(y_val, y_lr_pre
d))
    print("Best Ridge -> RMSE:", np.sqrt(mean_squared_error(y_val, y_ridge_pred)), ", R2:", r2_score(y_val, y_ridge_pre
d))
    print("Best Lasso -> RMSE:", np.sqrt(mean_squared_error(y_val, y_lasso_pred)), ", R2:", r2_score(y_val, y_lasso_pre
d))

    # Select best model (Linear in this case)
    best_model = lr_model

    # Final prediction on test data
    test_preds = best_model.predict(test[required_features])
    test['Predicted_Sales'] = np.round(test_preds).astype(int)

    return best_model, test[['Followers', 'AdSpend (GBP)', 'EngagementRate (%)', 'Predicted_Sales']]
```

--

Run the pipeline

--

train_df = pd.read_csv('Datasets/messy_train_data.csv') test_df = pd.read_csv('Datasets/messy_test_data.csv')

In [ ]: