In [2]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import datetime as dt
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
import seaborn as sns
# Load dataset
df = pd.read_csv('C:/Users/Adamin/OneDrive/Desktop/sales_data_sample.csv', encod
df.head()
```

Out[2]:

	ORDERNUMBER	QUANTITYORDERED	PRICEEACH	ORDERLINENUMBER	SALES	OR
0	10107	30	95.70	2	2871.00	
1	10121	34	81.35	5	2765.90	
2	10134	41	94.74	2	3884.34	
3	10145	45	83.26	6	3746.70	
4	10159	49	100.00	14	5205.27	1(

5 rows × 25 columns

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

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2823 entries, 0 to 2822
Data columns (total 25 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   ORDERNUMBER       2823 non-null   int64  
 1   QUANTITYORDERED   2823 non-null   int64  
 2   PRICEEACH         2823 non-null   float64
 3   ORDERLINENUMBER   2823 non-null   int64  
 4   SALES             2823 non-null   float64
 5   ORDERDATE         2823 non-null   object 
 6   STATUS            2823 non-null   object 
 7   QTR_ID            2823 non-null   int64  
 8   MONTH_ID          2823 non-null   int64  
 9   YEAR_ID           2823 non-null   int64  
 10  PRODUCTLINE       2823 non-null   object 
 11  MSRP              2823 non-null   int64  
 12  PRODUCTCODE       2823 non-null   object 
 13  CUSTOMERNAME      2823 non-null   object 
 14  PHONE             2823 non-null   object 
 15  ADDRESSLINE1      2823 non-null   object 
 16  ADDRESSLINE2      302 non-null    object 
 17  CITY              2823 non-null   object 
 18  STATE             1337 non-null   object 
 19  POSTALCODE        2747 non-null   object 
 20  COUNTRY           2823 non-null   object 
 21  TERRITORY         1749 non-null   object 
 22  CONTACTLASTNAME   2823 non-null   object 
 23  CONTACTFIRSTNAME  2823 non-null   object 
 24  DEALSIZE          2823 non-null   object 
dtypes: float64(2), int64(7), object(16)
memory usage: 551.5+ KB
```

In [4]:

```
# Drop unnecessary columns
to_drop = ['ADDRESSLINE1', 'ADDRESSLINE2', 'STATE', 'POSTALCODE', 'PHONE']
df = df.drop(to_drop, axis=1)
#Check for null values
df.isnull().sum()
```

Out[4]:

```
ORDERNUMBER            0
QUANTITYORDERED        0
PRICEEACH              0
ORDERLINENUMBER        0
SALES                  0
ORDERDATE              0
STATUS                 0
QTR_ID                 0
MONTH_ID               0
YEAR_ID                0
PRODUCTLINE            0
MSRP                   0
PRODUCTCODE            0
CUSTOMERNAME           0
CITY                   0
COUNTRY                0
TERRITORY           1074
CONTACTLASTNAME        0
CONTACTFIRSTNAME       0
DEALSIZE               0
dtype: int64
```

In [5]:
```
df.dtypes
```

Out[5]:
```
ORDERNUMBER            int64
QUANTITYORDERED        int64
PRICEEACH            float64
ORDERLINENUMBER        int64
SALES                float64
ORDERDATE             object
STATUS                object
QTR_ID                 int64
MONTH_ID               int64
YEAR_ID                int64
PRODUCTLINE           object
MSRP                   int64
PRODUCTCODE           object
CUSTOMERNAME          object
CITY                  object
COUNTRY               object
TERRITORY             object
CONTACTLASTNAME       object
CONTACTFIRSTNAME      object
DEALSIZE              object
dtype: object
```

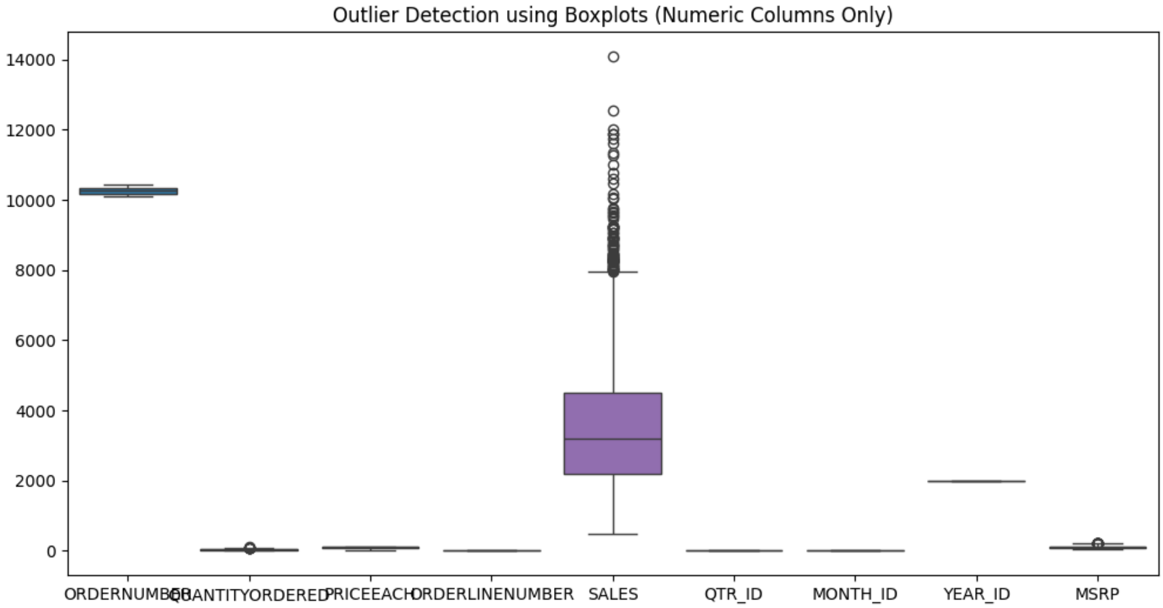
In [9]:
```
# Select numeric columns only
df_numeric = df.select_dtypes(include=['int64', 'float64'])

# Visualize outliers
import matplotlib.pyplot as plt
import seaborn as sns

plt.figure(figsize=(12, 6))
sns.boxplot(data=df_numeric)
plt.title("Outlier Detection using Boxplots (Numeric Columns Only)")
plt.show()

# Identify outliers using IQR
Q1 = df_numeric.quantile(0.25)
Q3 = df_numeric.quantile(0.75)
IQR = Q3 - Q1

# Count outliers per column
outliers = ((df_numeric < (Q1 - 1.5 * IQR)) | (df_numeric > (Q3 + 1.5 * IQR))).s
print("\nNumber of Outliers per Numeric Feature:\n", outliers)
```




```
Number of Outliers per Numeric Feature:
 ORDERNUMBER         0
QUANTITYORDERED     8
PRICEEACH           0
ORDERLINENUMBER     0
SALES              81
QTR_ID              0
MONTH_ID            0
YEAR_ID             0
MSRP               28
dtype: int64
```

In [13]:

```
#normilization data
scaler = StandardScaler()
X_scaled = scaler.fit_transform(df_numeric)
print(" Data normalized using StandardScaler.")

df_normalized = pd.DataFrame(X_scaled, columns=df_numeric.columns)

print("\nSample of Normalized Data:")
display(df_normalized.head())

print("\nMean of each feature after normalization:\n", df_normalized.mean())
print("\nStandard deviation of each feature after normalization:\n", df_normaliz
```

```
 Data normalized using StandardScaler.

Sample of Normalized Data:
```

	ORDERNUMBER	QUANTITYORDERED	PRICEEACH	ORDERLINENUMBER	SALES	(
0	-1.647947	-0.522891	0.596978	-1.057059	-0.370825	-1.4
1	-1.495888	-0.112201	-0.114450	-0.347015	-0.427897	-0.5
2	-1.354689	0.606505	0.549384	-1.057059	0.179443	0.2
3	-1.235214	1.017195	-0.019759	-0.110334	0.104701	0.2
4	-1.083154	1.427884	0.810158	1.783116	0.896740	1.0

```
Mean of each feature after normalization:
 ORDERNUMBER        -5.927482e-15
QUANTITYORDERED     3.347580e-16
PRICEEACH          -3.221731e-16
ORDERLINENUMBER     9.312817e-17
SALES               1.812224e-16
QTR_ID              7.550932e-17
MONTH_ID           -1.761884e-17
YEAR_ID            -1.988412e-15
MSRP                8.054328e-17
dtype: float64

Standard deviation of each feature after normalization:
 ORDERNUMBER         1.000177
QUANTITYORDERED     1.000177
PRICEEACH           1.000177
ORDERLINENUMBER     1.000177
SALES               1.000177
QTR_ID              1.000177
MONTH_ID            1.000177
YEAR_ID             1.000177
MSRP                1.000177
dtype: float64
```

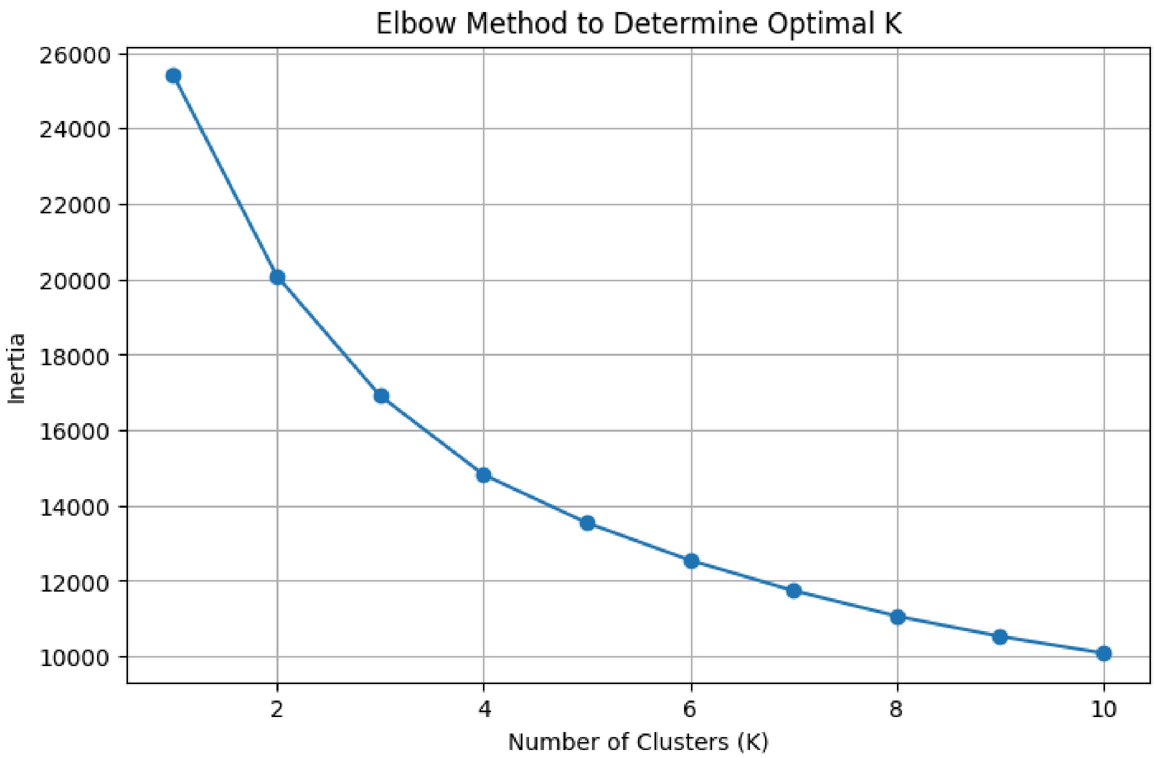
In [14]:

```python
inertia = []
K = range(1, 11)

for k in K:
    kmeans = KMeans(n_clusters=k, random_state=42, n_init=10)
    kmeans.fit(X_scaled)
    inertia.append(kmeans.inertia_)

# Plot the Elbow curve
plt.figure(figsize=(8, 5))
plt.plot(K, inertia, marker='o')
plt.title("Elbow Method to Determine Optimal K")
plt.xlabel("Number of Clusters (K)")
plt.ylabel("Inertia")
plt.grid(True)
plt.show()
```

```
C:\Users\Adamin\AppData\Roaming\Python\Python313\site-packages\joblib\externals\l
oky\backend\context.py:136: UserWarning: Could not find the number of physical co
res for the following reason:
[WinError 2] The system cannot find the file specified
Returning the number of logical cores instead. You can silence this warning by se
tting LOKY_MAX_CPU_COUNT to the number of cores you want to use.
  warnings.warn(
  File "C:\Users\Adamin\AppData\Roaming\Python\Python313\site-packages\joblib\ext
ernals\loky\backend\context.py", line 257, in _count_physical_cores
    cpu_info = subprocess.run(
        "wmic CPU Get NumberOfCores /Format:csv".split(),
        capture_output=True,
        text=True,
    )
  File "C:\Program Files\Python313\Lib\subprocess.py", line 554, in run
    with Popen(*popenargs, **kwargs) as process:
         ~~~~~^^^^^^^^^^^^^^^^^^^^^^
  File "C:\Program Files\Python313\Lib\subprocess.py", line 1039, in __init__
    self._execute_child(args, executable, preexec_fn, close_fds,
    ~~~~~~~~~~~~~~~~~~~^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
                        pass_fds, cwd, env,
                        ^^^^^^^^^^^^^^^^^^^
    ...<5 lines>...
                        gid, gids, uid, umask,
                        ^^^^^^^^^^^^^^^^^^^^^^
                        start_new_session, process_group)
                        ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "C:\Program Files\Python313\Lib\subprocess.py", line 1554, in _execute_chi
ld
    hp, ht, pid, tid = _winapi.CreateProcess(executable, args,
                       ~~~~~~~~~~~~~~~~~~~~~^^^^^^^^^^^^^^^^^^
                             # no special security
                             ^^^^^^^^^^^^^^^^^^^^^
    ...<4 lines>...
                             cwd,
                             ^^^^
                             startupinfo)
                             ^^^^^^^^^^^^
```



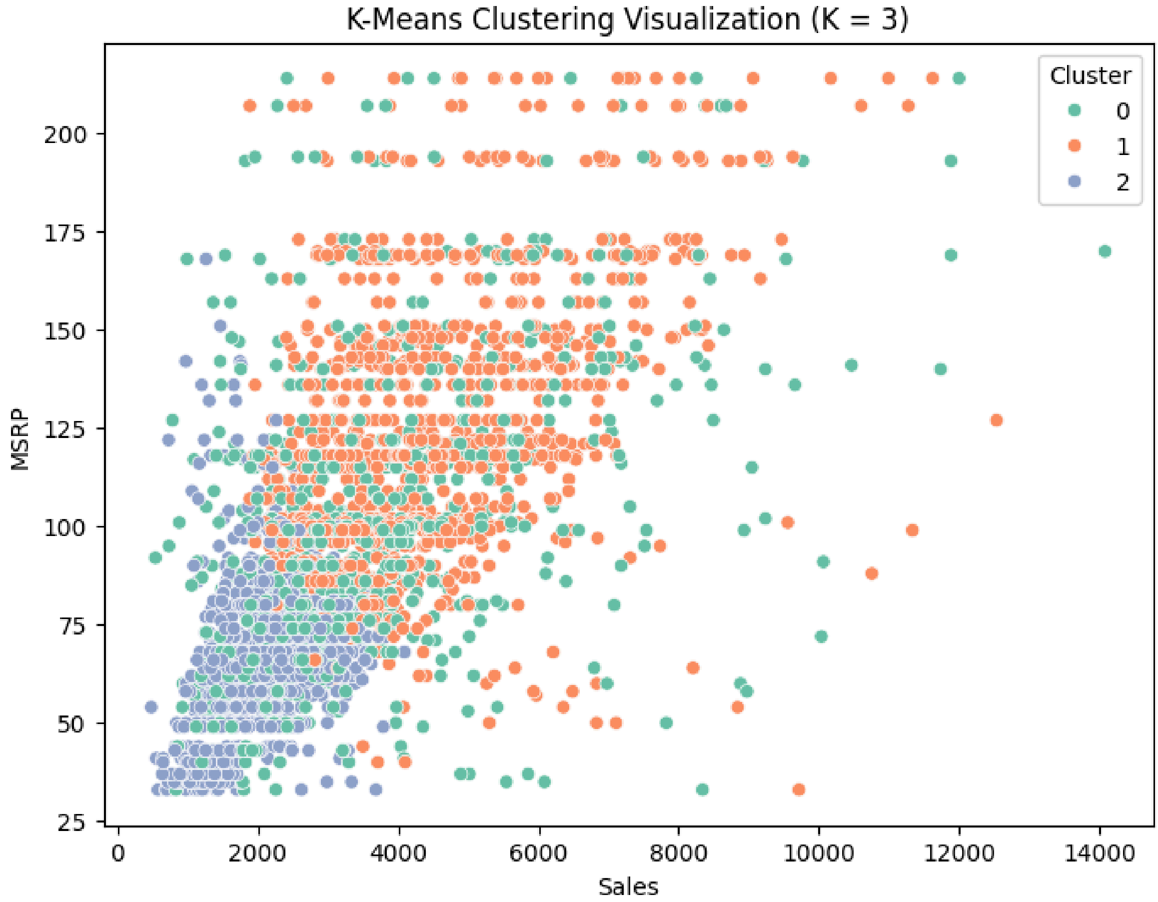
In [15]:
```
from sklearn.metrics import silhouette_score

for k in range (2, 11):
    kmeans = KMeans(n_clusters=k, random_state=42, n_init=10)
    labels = kmeans.fit_predict(X_scaled)
    sil_score = silhouette_score(X_scaled, labels)
    print(f"K = {k} → Silhouette Score = {sil_score:.4f}")
```

```
K = 2 → Silhouette Score = 0.2169
K = 3 → Silhouette Score = 0.1960
K = 4 → Silhouette Score = 0.2059
K = 5 → Silhouette Score = 0.1872
K = 6 → Silhouette Score = 0.1949
K = 7 → Silhouette Score = 0.1922
K = 8 → Silhouette Score = 0.1837
K = 9 → Silhouette Score = 0.1832
K = 10 → Silhouette Score = 0.1842
```

In [17]:
```
#Visualize Clusters for K = 3
kmeans = KMeans(n_clusters=3, random_state=42, n_init=10)
labels = kmeans.fit_predict(X_scaled)

plt.figure(figsize=(8,6))
sns.scatterplot(
    x=df_numeric['SALES'],
    y=df_numeric['MSRP'],
    hue=labels,
    palette='Set2'
)
plt.title("K-Means Clustering Visualization (K = 3)")
plt.xlabel("Sales")
plt.ylabel("MSRP")
plt.legend(title='Cluster')
plt.show()
```


K-Means Clustering Visualization (K = 3)
Cluster
0
1
2
MSRP
Sales
25
50
75
100
125
150
175
200
0
2000
4000
6000
8000
10000
12000
14000

In [ ]: