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

import matplotlib.pyplot as plt
import seaborn as sns
```

Level 1: Data Loading and Basic Analysis

1. Load the data into pandas dataframes

In [2]:
```
df_sales=pd.read_csv("given_sales.csv")

df_customer=pd.read_csv("given_customers.csv")

df_regions=pd.read_csv("given_regions.csv")

df_products=pd.read_csv("given_products.csv")
```

In [3]:
```
df_sales
```

Out[3]:

	sales_id	product_id	customer_id	sales_amount	date	region_id
0	1	101	201	1000	01-01-2022	1
1	2	102	202	1500	02-01-2022	2
2	3	103	203	800	03-01-2022	1
3	4	104	204	1200	04-01-2022	3
4	5	105	205	2000	05-01-2022	1
...	...	...	...	...	...	...
95	96	196	296	1250	06-04-2022	9
96	97	197	297	850	07-04-2022	10
97	98	198	298	1950	08-04-2022	1
98	99	199	299	1300	09-04-2022	2
99	100	200	300	1750	10-04-2022	3

100 rows × 6 columns

In [4]:
```
df_products
```

Out[4]:

	product_id	product_name	category	price
0	101	Product A	Electronics	500
1	102	Product B	Clothing	800
2	103	Product C	Home Decor	300
3	104	Product D	Electronics	1000
4	105	Product E	Beauty	1500
...	...	...	...	...
95	196	Product CR	Sports	850
96	197	Product CS	Clothing	650
97	198	Product CT	Electronics	1400
98	199	Product CU	Home Decor	1900
99	200	Product CV	Beauty	2400

100 rows × 4 columns

In [5]:
```
df_customer
```

Out[5]:

	customer_id	customer_name	email	address
0	201	John Doe	john@example.com	123 Main St, Anytown, USA
1	202	Jane Smith	jane@example.com	456 Elm St, Othertown, USA
2	203	Robert Brown	robert@example.com	789 Oak St, Anycity, USA
3	204	Lisa Johnson	lisa@example.com	321 Maple St, Somewhere, USA
4	205	Michael Wilson	michael@example.com	654 Pine St, Nowhere, USA
...	...	...	...	...
95	296	Amelia Foster	amelia@example.com	366 Oak St, Nowhere, USA
96	297	Oliver Reyes	oliver@example.com	586 Maple St, Anytown, USA
97	298	Sophia Gray	sophia@example.com	747 Pine St, Othertown, USA
98	299	Elijah Bryant	elijah@example.com	983 Cedar St, Somewhere, USA
99	300	Harper Watson	harper@example.com	624 Elm St, Nowhere, USA

100 rows × 4 columns

In [6]:
```
df_regions
```

Out[6]:

	region_id	region_name
0	1	East Coast
1	2	West Coast
2	3	Midwest
3	4	South
4	5	Northeast
...	...	...
95	96	West Coast
96	97	Midwest
97	98	South
98	99	Northeast
99	100	Southwest

100 rows × 2 columns

In [7]:
```
merged_df = pd.merge(df_sales, df_products, on='product_id')

merged_df=pd.merge(merged_df, df_customer, on='customer_id')

merged_df=pd.merge(merged_df, df_regions, on='region_id')
```

In [8]:
```
merged_df
```

Out[8]:

	sales_id	product_id	customer_id	sales_amount	date	region_id	product_name	cate
0	1	101	201	1000	01-01-2022	1	Product A	Electro
1	3	103	203	800	03-01-2022	1	Product C	H D
2	5	105	205	2000	05-01-2022	1	Product E	Be
3	8	108	208	3000	08-01-2022	1	Product H	H D
4	11	111	211	1300	11-01-2022	1	Product K	Sp
...	...	...	...	...	...	...	...	
95	57	157	257	800	26-02-2022	10	Product BE	Clot
96	67	167	267	850	08-03-2022	10	Product BO	Clot
97	77	177	277	850	18-03-2022	10	Product BY	Clot
98	87	187	287	850	28-03-2022	10	Product CI	Clot
99	97	197	297	850	07-04-2022	10	Product CS	Clot

100 rows × 13 columns

2. Display the first 5 rows of the database

In [9]:
```
merged_df.head(5)
```

Out[9]:

	sales_id	product_id	customer_id	sales_amount	date	region_id	product_name	categ
0	1	101	201	1000	01-01-2022	1	Product A	Electron
1	3	103	203	800	03-01-2022	1	Product C	Ho De
2	5	105	205	2000	05-01-2022	1	Product E	Bea
3	8	108	208	3000	08-01-2022	1	Product H	Ho De
4	11	111	211	1300	11-01-2022	1	Product K	Spo

3. Check the shape of the database

In [10]:
```
merged_df.shape
```

Out[10]:
```
(100, 13)
```

4. Display basic statistics (mean, median, min, max, etc.) for Sales

In [11]:
```
merged_df.describe(include='all')
```

Out[11]:

	sales_id	product_id	customer_id	sales_amount	date	region_id	product_nam
count	100.000000	100.000000	100.000000	100.000000	100	100.000000	10
unique	NaN	NaN	NaN	NaN	100	NaN	9
top	NaN	NaN	NaN	NaN	01-01-2022	NaN	Product C
freq	NaN	NaN	NaN	NaN	1	NaN	
mean	50.500000	150.500000	250.500000	1499.000000	NaN	4.780000	Na
std	29.011492	29.011492	29.011492	510.593831	NaN	2.983388	Na
min	1.000000	101.000000	201.000000	600.000000	NaN	1.000000	Na
25%	25.750000	125.750000	225.750000	1100.000000	NaN	2.000000	Na
50%	50.500000	150.500000	250.500000	1500.000000	NaN	4.000000	Na
75%	75.250000	175.250000	275.250000	1850.000000	NaN	7.000000	Na
max	100.000000	200.000000	300.000000	3000.000000	NaN	10.000000	Na

In [12]:
```
merged_df.describe()
```

Out[12]:

	sales_id	product_id	customer_id	sales_amount	region_id	price
count	100.000000	100.000000	100.000000	100.000000	100.000000	100.00000
mean	50.500000	150.500000	250.500000	1499.000000	4.780000	1366.00000
std	29.011492	29.011492	29.011492	510.593831	2.983388	591.68822
min	1.000000	101.000000	201.000000	600.000000	1.000000	300.00000
25%	25.750000	125.750000	225.750000	1100.000000	2.000000	887.50000
50%	50.500000	150.500000	250.500000	1500.000000	4.000000	1350.00000
75%	75.250000	175.250000	275.250000	1850.000000	7.000000	1800.00000
max	100.000000	200.000000	300.000000	3000.000000	10.000000	2500.00000

5. Determine the number of unique products sold

In [13]:
```
# From above observation we can see that sales_amount,region_id,price have almost same m
# data is normallized and not skewed
```

In [14]:
```
merged_df.product_name.unique()
```

Out[14]:
```
array(['Product A', 'Product C', 'Product E', 'Product H', 'Product K',
       'Product N', 'Product Q', 'Product T', 'Product BB', 'Product LL',
       'Product AV', 'Product BF', 'Product BP', 'Product BZ',
       'Product CJ', 'Product CT', 'Product B', 'Product F', 'Product I',
       'Product L', 'Product O', 'Product R', 'Product CC', 'Product MM',
       'Product AW', 'Product BG', 'Product BQ', 'Product CA',
       'Product CK', 'Product CU', 'Product D', 'Product G', 'Product J',
       'Product M', 'Product P', 'Product S', 'Product DD', 'Product NN',
       'Product AX', 'Product BH', 'Product BR', 'Product CB',
       'Product CL', 'Product CV', 'Product U', 'Product EE',
       'Product AO', 'Product AY', 'Product BI', 'Product BS',
       'Product CM', 'Product V', 'Product FF', 'Product AP',
       'Product AZ', 'Product BJ', 'Product BT', 'Product CD',
       'Product CN', 'Product W', 'Product GG', 'Product AQ',
       'Product BA', 'Product BK', 'Product BU', 'Product CE',
       'Product CO', 'Product X', 'Product HH', 'Product AR',
       'Product BL', 'Product BV', 'Product CF', 'Product CP',
       'Product Y', 'Product II', 'Product AS', 'Product BC',
       'Product BM', 'Product BW', 'Product CG', 'Product CQ',
       'Product Z', 'Product JJ', 'Product AT', 'Product BD',
       'Product BN', 'Product BX', 'Product CH', 'Product CR',
       'Product AA', 'Product KK', 'Product AU', 'Product BE',
       'Product BO', 'Product BY', 'Product CI', 'Product CS'],
      dtype=object)
```

In [15]:
```
merged_df['product_name'].value_counts()
```

Out[15]:
```
product_name
Product CC    2
Product BB    2
Product A     1
Product BK    1
Product CF    1
              ..
Product G     1
Product D     1
Product CU    1
Product CK    1
Product CS    1
Name: count, Length: 98, dtype: int64
```

Level 2: Data Cleaning and Pre-processing

1. Handle missing values by filling hem wih appropriae values (e.g., mean or median)

In [16]:
```
merged_df.isnull().sum()
```

Out[16]:
```
sales_id         0
product_id       0
customer_id      0
sales_amount     0
date             0
region_id        0
product_name     0
category         0
price            0
customer_name    0
email            0
address          0
region_name      0
dtype: int64
```

2. Convert categorical variables to numerical representation using one-hot encoding or label encoding.

In [17]:
```
from sklearn.preprocessing import LabelEncoder

encoder=LabelEncoder()
```

In [18]:
```
df1=merged_df.copy()
```

In [19]:
```
columns=['product_name','category','region_name']
for col in columns:
    encoded=encoder.fit_transform(df1[col])
    df1[col+'_encoded']=encoded
```

In [20]:
```
df1.head()
```

Out[20]:

	sales_id	product_id	customer_id	sales_amount	date	region_id	product_name	categ
0	1	101	201	1000	01-01-2022	1	Product A	Electron
1	3	103	203	800	03-01-2022	1	Product C	Ho De
2	5	105	205	2000	05-01-2022	1	Product E	Bea
3	8	108	208	3000	08-01-2022	1	Product H	Ho De
4	11	111	211	1300	11-01-2022	1	Product K	Spo

3. Check for and remove any duplicate rows in the database

In [21]:
```
duplicate_val=df1.duplicated()
duplicate_val.value_counts()
#no duplicate
#incase dulicate is there we can remove  using following
# df.drop_duplicates(inplace=True)
```

Out[21]:
```
False    100
Name: count, dtype: int64
```

In [22]:
```
df1
```

Out[22]:

	sales_id	product_id	customer_id	sales_amount	date	region_id	product_name	cate
0	1	101	201	1000	01-01-2022	1	Product A	Electr
1	3	103	203	800	03-01-2022	1	Product C	H D
2	5	105	205	2000	05-01-2022	1	Product E	Be
3	8	108	208	3000	08-01-2022	1	Product H	H D
4	11	111	211	1300	11-01-2022	1	Product K	Sp
...	...	...	...	...	...	...	...	
95	57	157	257	800	26-02-2022	10	Product BE	Clot
96	67	167	267	850	08-03-2022	10	Product BO	Clot
97	77	177	277	850	18-03-2022	10	Product BY	Clot
98	87	187	287	850	28-03-2022	10	Product CI	Clot
99	97	197	297	850	07-04-2022	10	Product CS	Clot

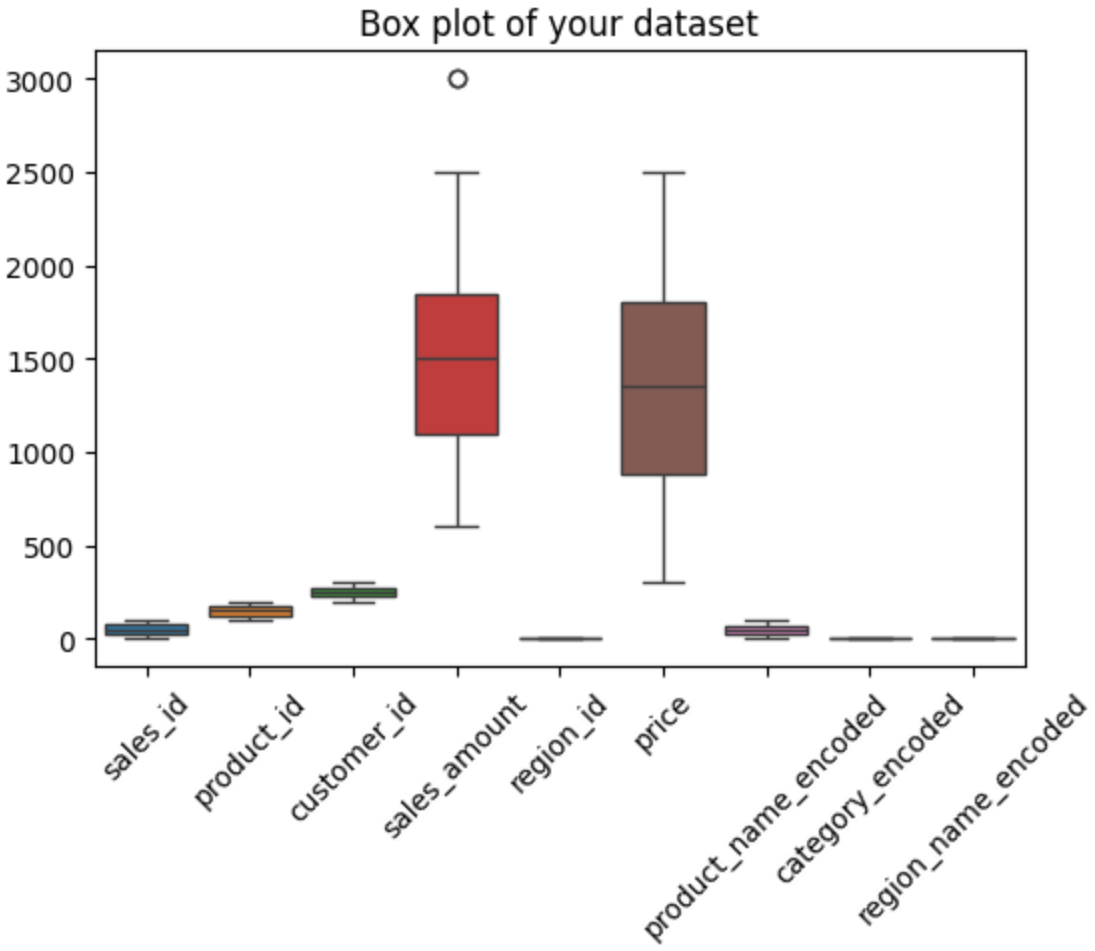
100 rows × 16 columns

4. Normalize the 'Sales' column to a scale between 0 and 1.

5. Identify and remove outliers from the database using appropriate techniques.

In [23]:

```
plt.figure(figsize=(6, 4))
sns.boxplot(data=df1)
plt.title('Box plot of your dataset')
plt.xticks(rotation=45)  # Rotate x-axis labels for better readability
plt.show()
```



In [24]:
```
# we can see that there is outlier present n the sales amount  column
```

In [25]:
```
Q1=df1['sales_amount'].quantile(0.25)

Q3=df1['sales_amount'].quantile(0.75)

IQR=Q3-Q1
```

In [26]:
```
lower_bound = 0
upper_bound = Q3 + 1.5 * IQR
```

In [27]:
```
upper_bound
```

Out[27]:
```
2975.0
```

In [28]:
```
cleaned_sales=df1[(df1['sales_amount']>lower_bound )& (df1['sales_amount']<upper_bound)]
cleaned_sales
```

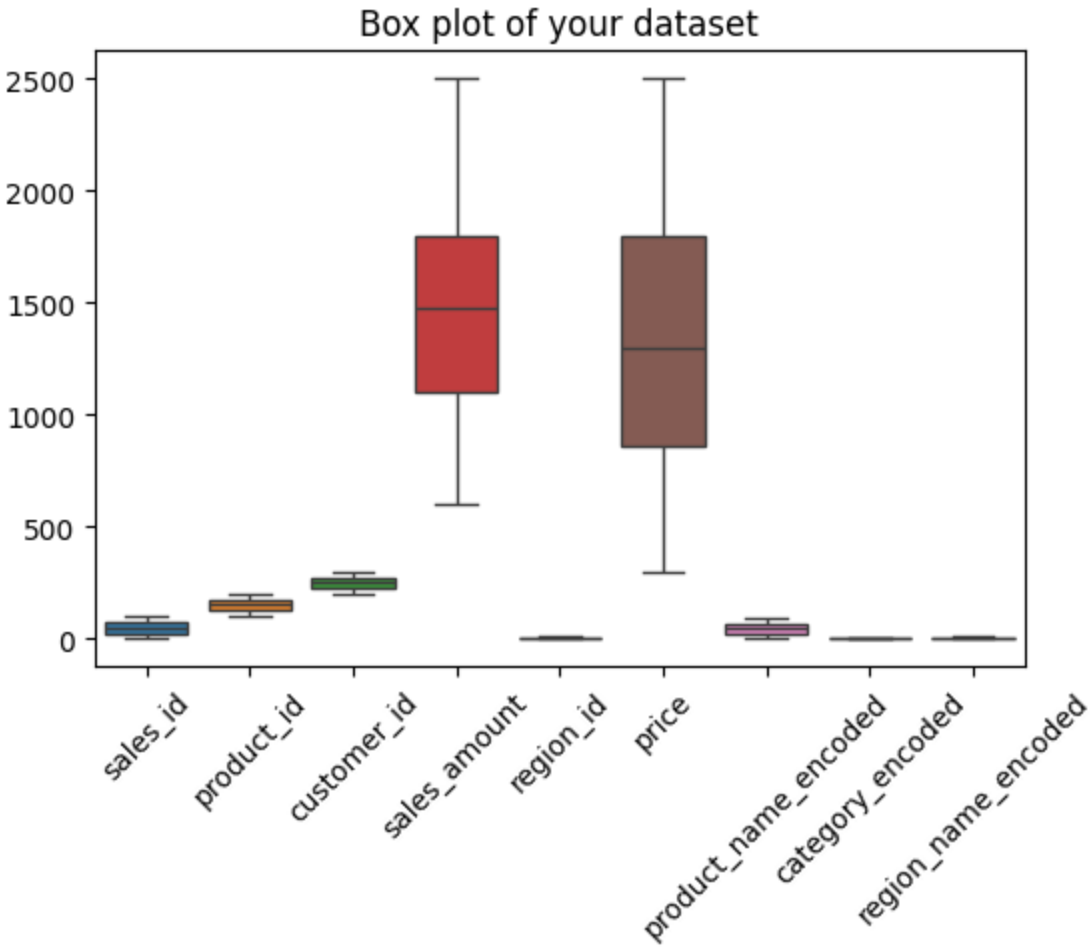
Out[28]:

	sales_id	product_id	customer_id	sales_amount	date	region_id	product_name	cate
0	1	101	201	1000	01-01-2022	1	Product A	Electro
1	3	103	203	800	03-01-2022	1	Product C	H D
2	5	105	205	2000	05-01-2022	1	Product E	Be
4	11	111	211	1300	11-01-2022	1	Product K	Sp
5	14	114	214	1700	14-01-2022	1	Product N	H D
...	...	...	...	...	...	...	...	
95	57	157	257	800	26-02-2022	10	Product BE	Clot
96	67	167	267	850	08-03-2022	10	Product BO	Clot
97	77	177	277	850	18-03-2022	10	Product BY	Clot
98	87	187	287	850	28-03-2022	10	Product CI	Clot
99	97	197	297	850	07-04-2022	10	Product CS	Clot

98 rows × 16 columns

In [29]:

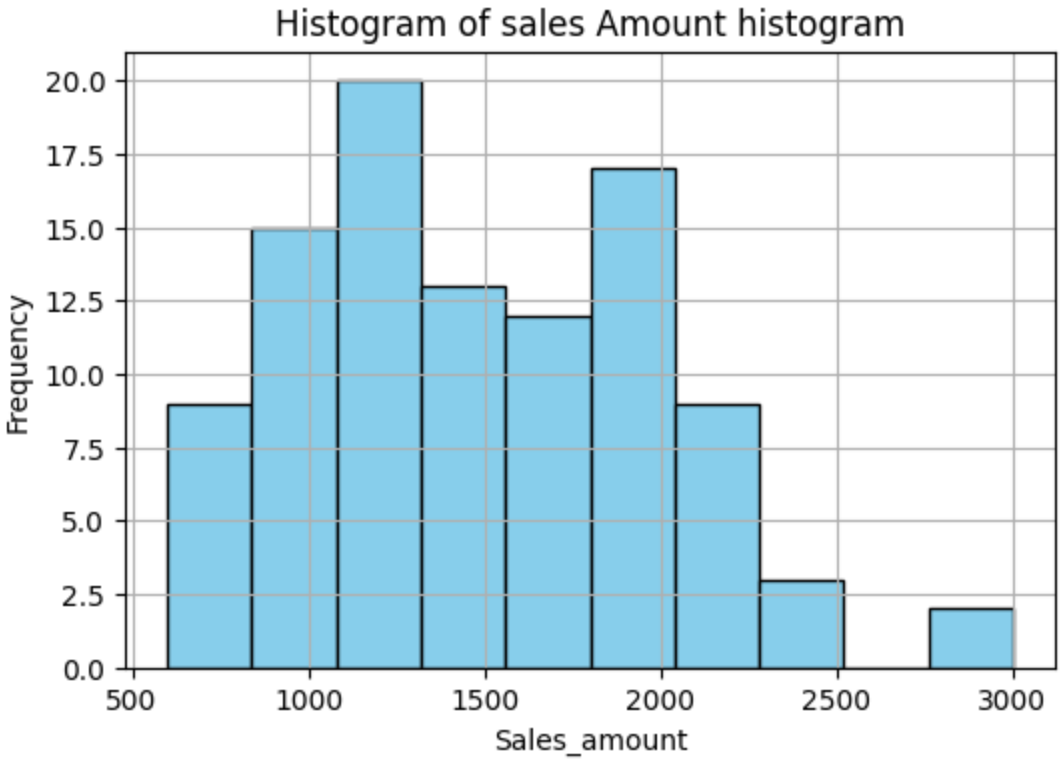
```
import matplotlib.pyplot as plt
import seaborn as sns
plt.figure(figsize=(6, 4))
sns.boxplot(data=cleaned_sales)
plt.title('Box plot of your dataset')
plt.xticks(rotation=45)  # Rotate x-axis labels for better readability
plt.show()

# we can see that two rows has been removed and the two outlier values has been removed
```



Level 3: Exploratory Data Analysis (EDA)

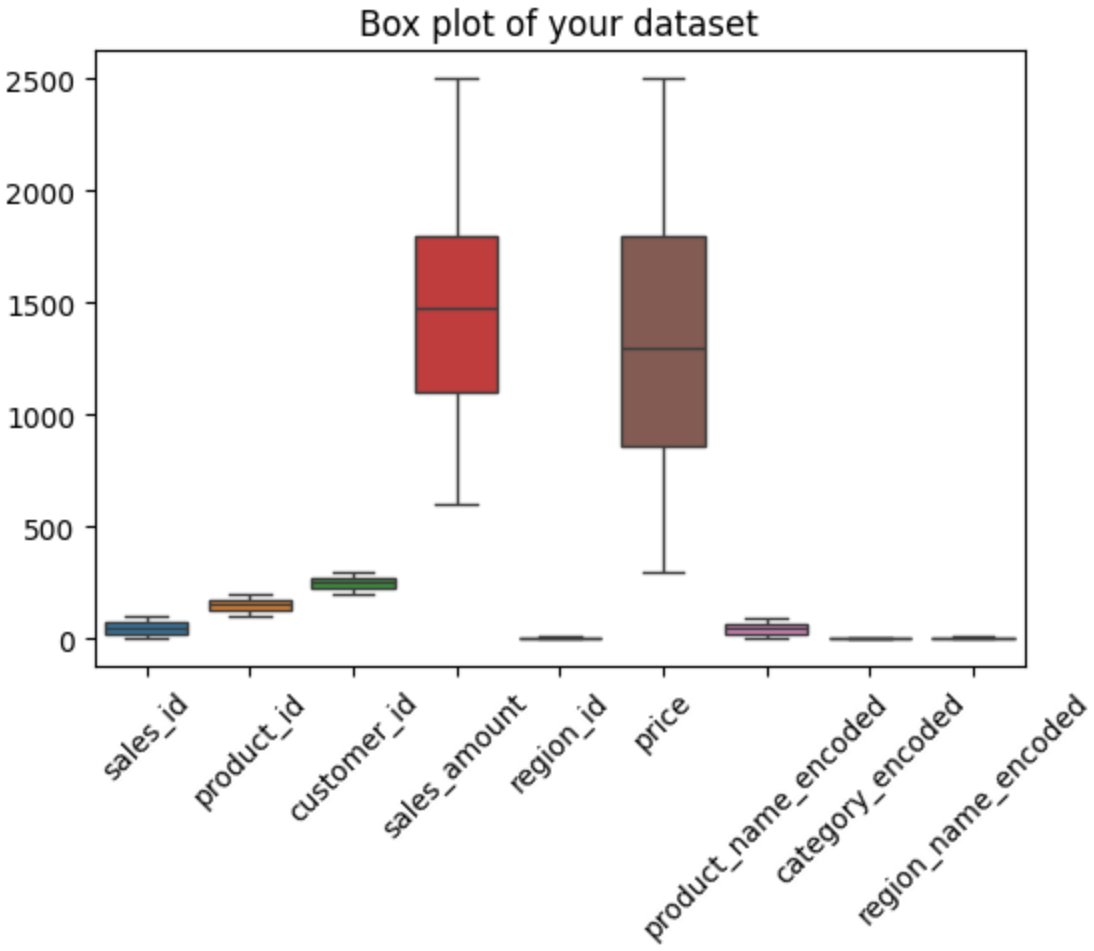
1. Visualize the distribution of sales amounts using a histogram

In [30]:
```
plt.figure(figsize=(6, 4))
plt.hist(df1['sales_amount'], bins=10, color='skyblue', edgecolor='black')  # Adjust the
plt.title('Histogram of {}'.format('sales Amount histogram'))
plt.xlabel('Sales_amount')
plt.ylabel('Frequency')
plt.grid(True)
plt.show()
```



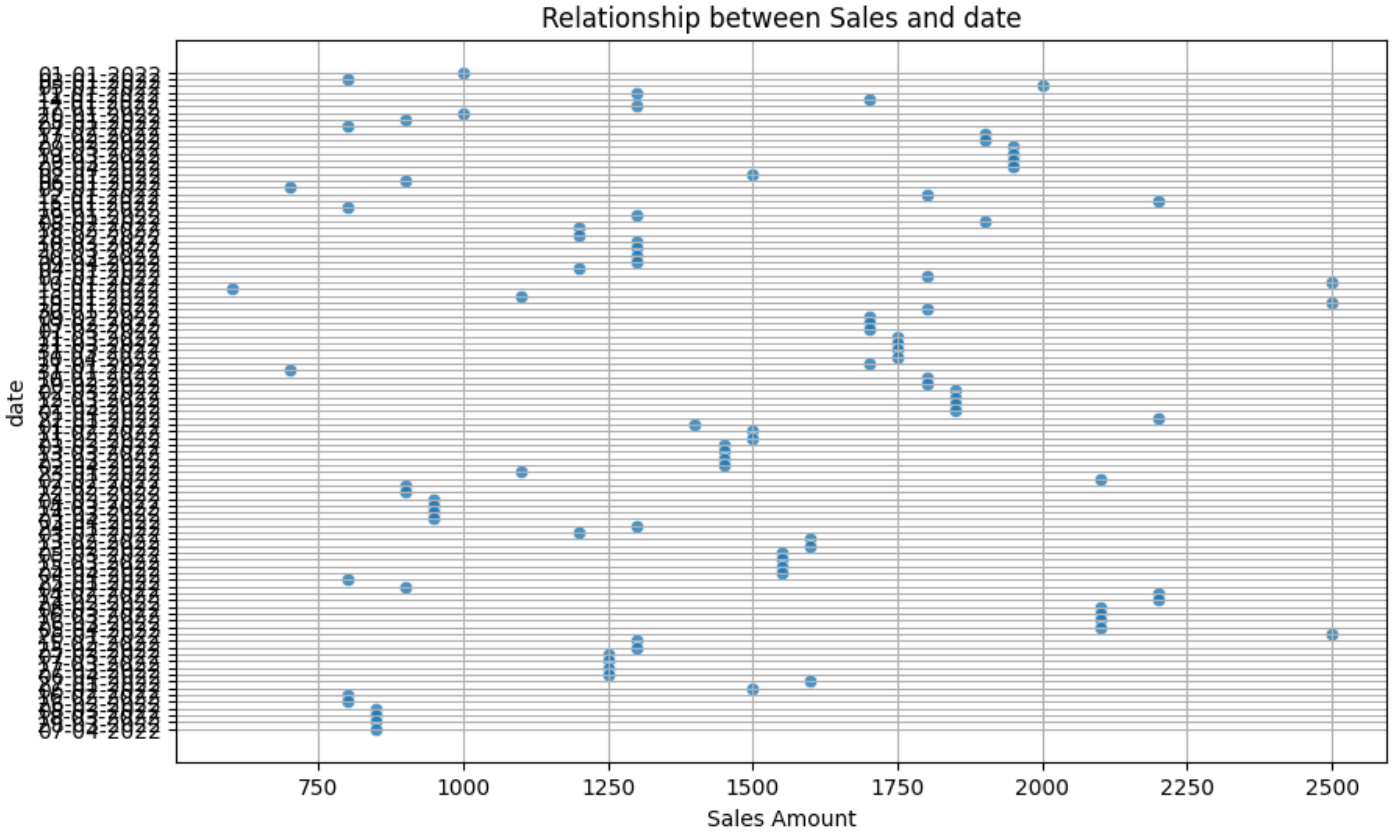
2. Create a boxplot to identify any outliers in the sales data.

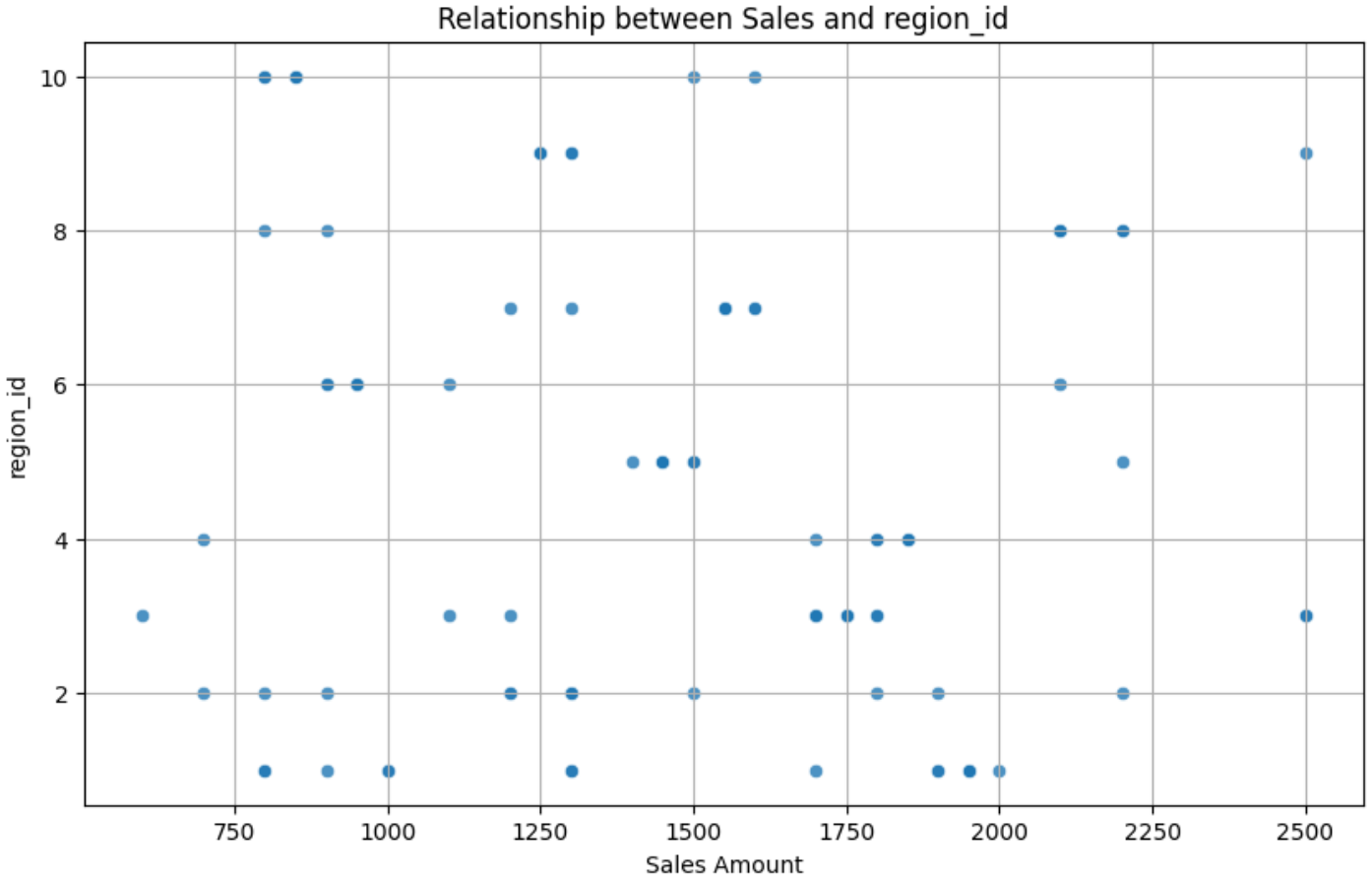
In [31]:
```
import matplotlib.pyplot as plt
import seaborn as sns
plt.figure(figsize=(6, 4))
sns.boxplot(data=cleaned_sales)
plt.title('Box plot of your dataset')
plt.xticks(rotation=45)  # Rotate x-axis labels for better readability
plt.show()
```

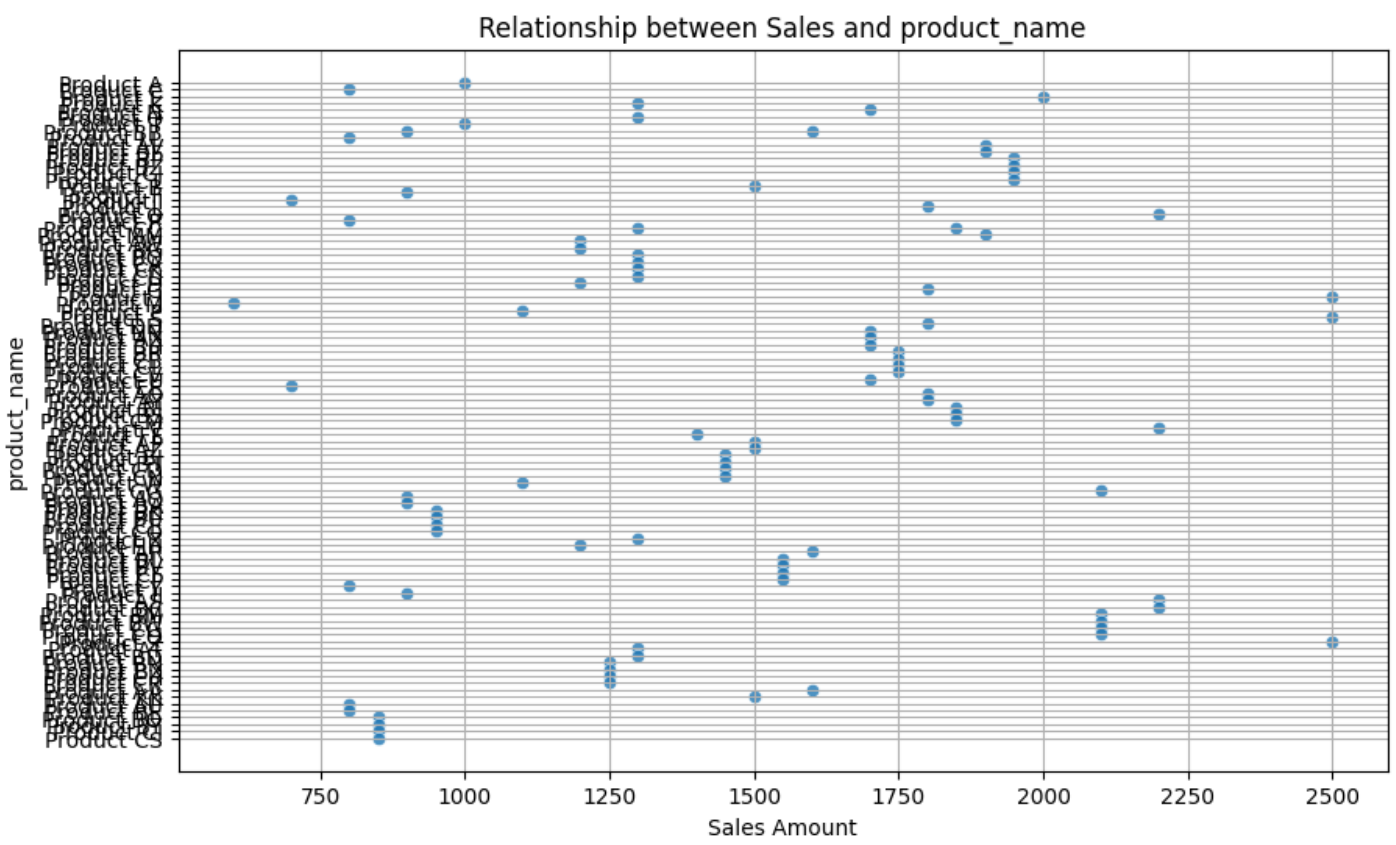


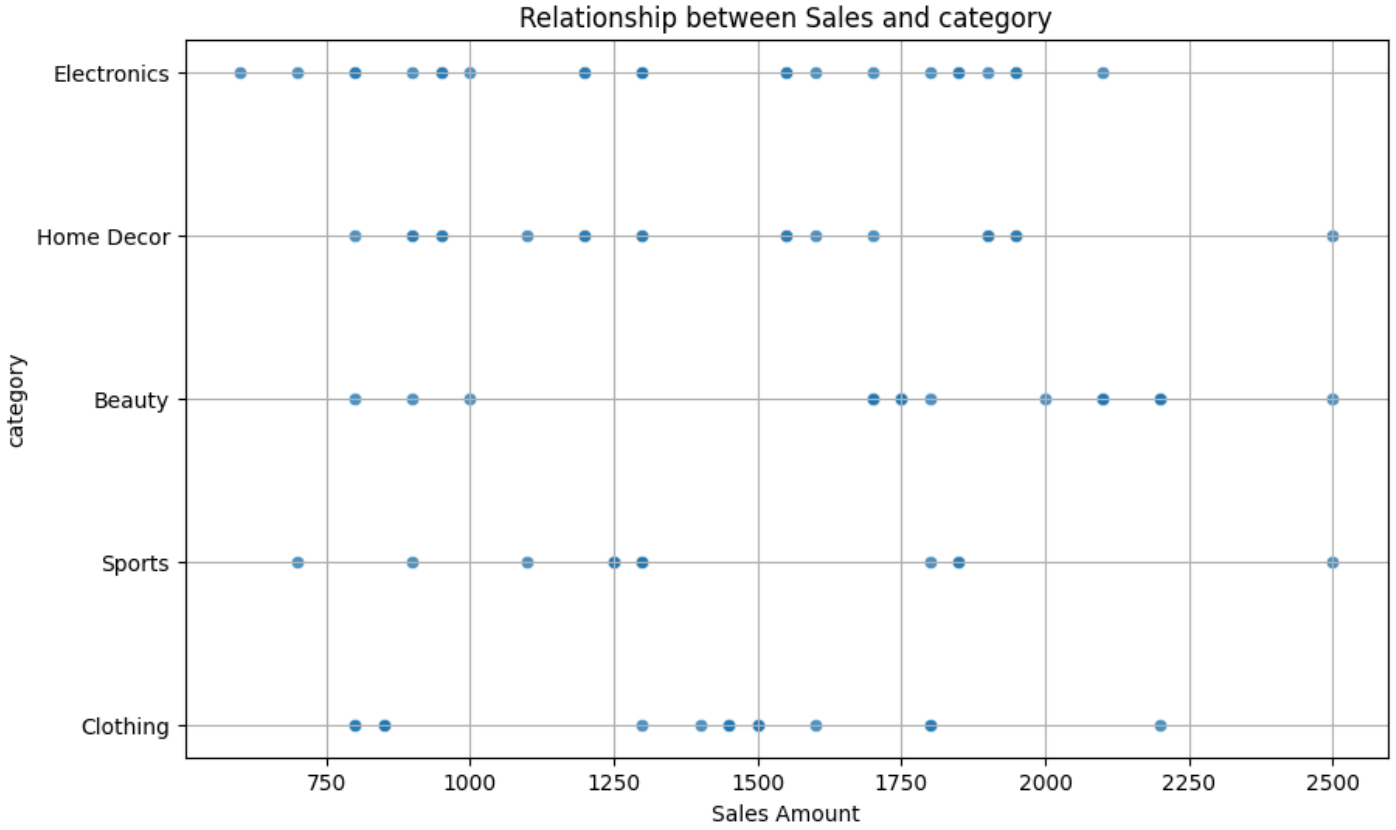
3. Explore the relationship between sales and other variables using scatter plots.

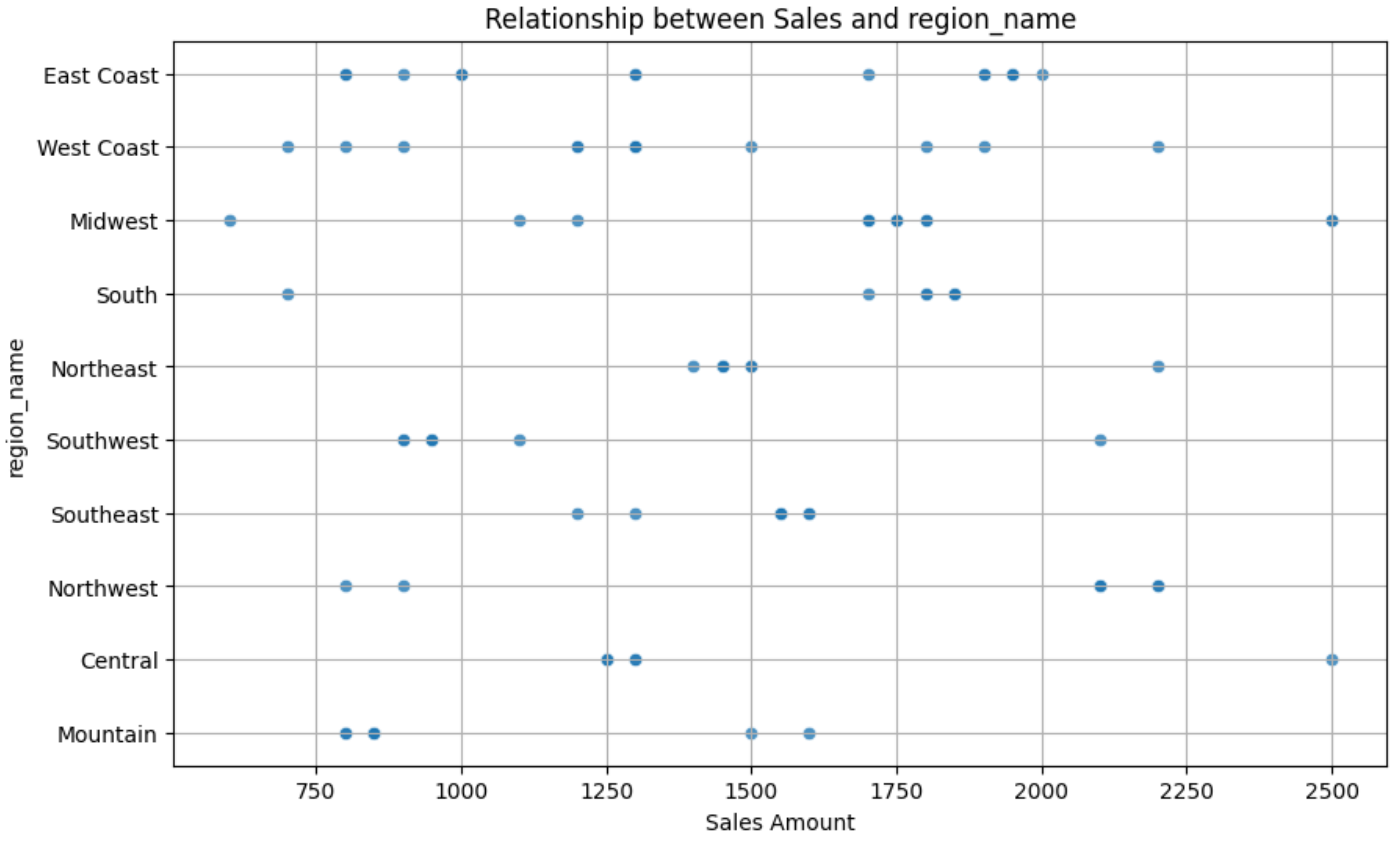
In [32]:
```
# Define the required columns for scatter plots
required_columns = ['date', 'region_id', 'product_name','category','region_name']

# Create scatter plots for each combination of sales and other variables
for column in required_columns:
    plt.figure(figsize=(10, 6))

    # Scatter plot between sales and the current variable
    sns.scatterplot(data=cleaned_sales, x='sales_amount', y=column, alpha=0.8)

    # Set plot labels and title
    plt.title(f'Relationship between Sales and {column}')
    plt.xlabel('Sales Amount')
    plt.ylabel(column)

    # Show grid
    plt.grid(True)

    # Show plot
    plt.show()
```


Relationship between Sales and date
date
Sales Amount
750
1000
1250
1500
1750
2000
2250
2500


Relationship between Sales and region_id
region_id
Sales Amount
10
8
6
4
2
750
1000
1250
1500
1750
2000
2250
2500

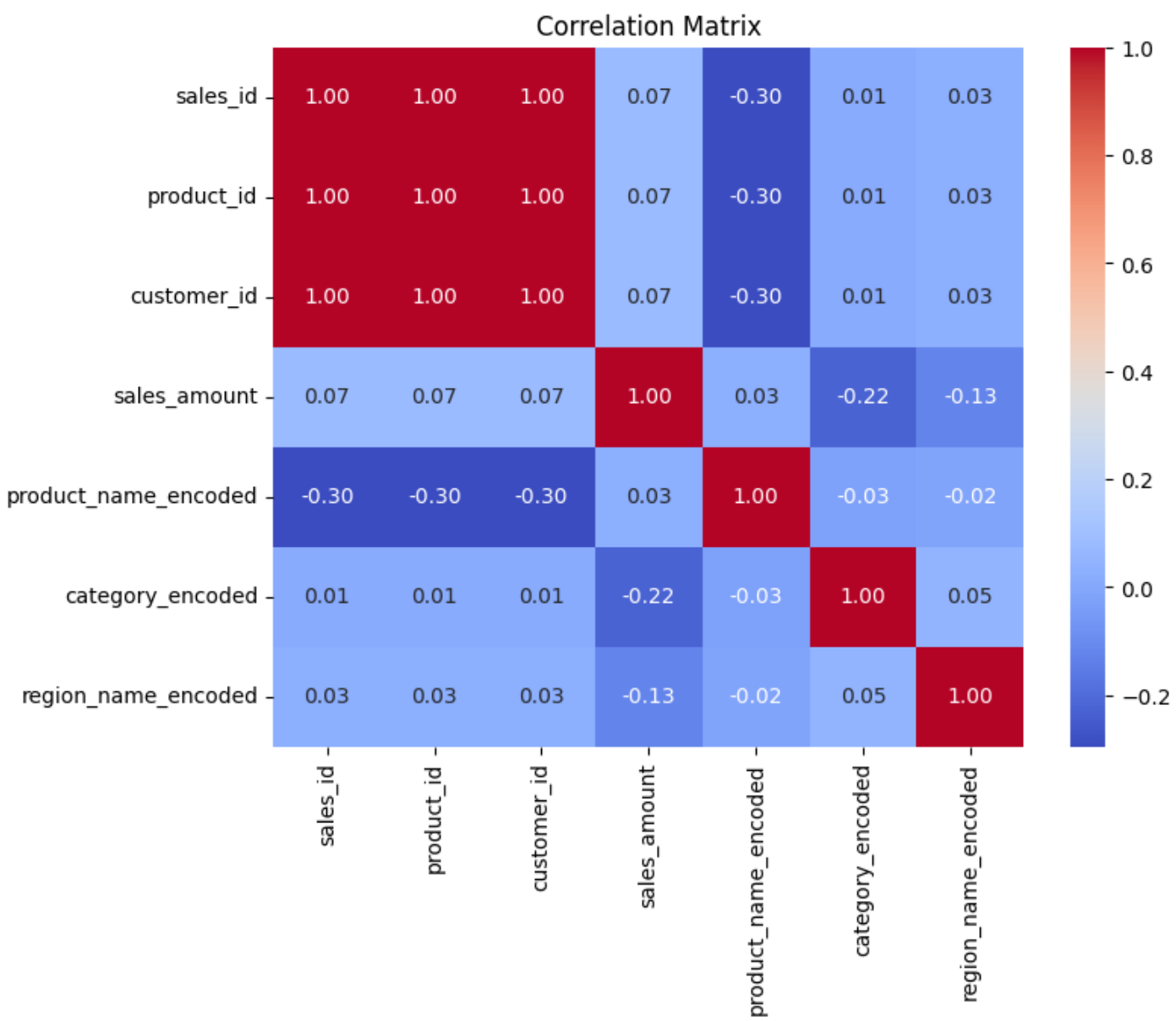

Relationship between Sales and product_name
product_name
Sales Amount
750
1000
1250
1500
1750
2000
2250
2500


Relationship between Sales and category
category
Electronics
Home Decor
Beauty
Sports
Clothing
Sales Amount
750
1000
1250
1500
1750
2000
2250
2500



4. Calculate and visualize the correlation matrix between numerical variables.

In [33]:
```
# Calculate the correlation matrix
correlation_matrix = pd.concat([cleaned_sales.iloc[:,:4], cleaned_sales.iloc[:,13:]], ax

# Visualize the correlation matrix using a heatmap
import seaborn as sns
plt.figure(figsize=(8, 6))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Correlation Matrix')
plt.show()
```


Correlation Matrix
sales_id
product_id
customer_id
sales_amount
product_name_encoded
category_encoded
region_name_encoded
1.00 1.00 1.00 0.07 -0.30 0.01 0.03
1.00 1.00 1.00 0.07 -0.30 0.01 0.03
1.00 1.00 1.00 0.07 -0.30 0.01 0.03
0.07 0.07 0.07 1.00 0.03 -0.22 -0.13
-0.30 -0.30 -0.30 0.03 1.00 -0.03 -0.02
0.01 0.01 0.01 -0.22 -0.03 1.00 0.05
0.03 0.03 0.03 -0.13 -0.02 0.05 1.00
1.0
0.8
0.6
0.4
0.2
0.0
−0.2

In [34]:
```
cleaned_sales.head()
```

Out[34]:

	sales_id	product_id	customer_id	sales_amount	date	region_id	product_name	categ
0	1	101	201	1000	01-01-2022	1	Product A	Electron
1	3	103	203	800	03-01-2022	1	Product C	Ho De
2	5	105	205	2000	05-01-2022	1	Product E	Bea
4	11	111	211	1300	11-01-2022	1	Product K	Spo
5	14	114	214	1700	14-01-2022	1	Product N	Ho De

5. Analyze the sales trend over time using line plots or time series plots

Level 4: Data Aggretation and Grouping

1. Group the sales data by product category and calculate the total sales amount for each category.

In [35]:
```
category_sales = cleaned_sales.groupby('category')['sales_amount'].sum()
category_sales
```

Out[35]:
```
category
Beauty         36100
Clothing       26900
Electronics    34100
Home Decor     27200
Sports         19600
Name: sales_amount, dtype: int64
```

2. Group the sales data by month and year and calculate the average sales amount for each month.

In [36]:
```
sales_data=cleaned_sales.copy()
sales_data['date'] = pd.to_datetime(sales_data['date'], format="mixed")
sales_data['month'] = sales_data['date'].dt.to_period('M')
sales_data['year']=sales_data['date'].dt.to_period('Y')
monthly_average_sales = sales_data.groupby('month')['sales_amount'].mean()
yearly_average_sales = sales_data.groupby('year')['sales_amount'].mean()
```

In [37]:
```
print(monthly_average_sales,yearly_average_sales)
```

```
month
2022-01    1436.956522
2022-02    1540.000000
2022-03    1415.217391
2022-04    1150.000000
2022-05    1883.333333
2022-06    1437.500000
2022-07    1175.000000
2022-08    1566.666667
2022-09    1412.500000
2022-10    1837.500000
2022-11    1516.666667
2022-12    1516.666667
Freq: M, Name: sales_amount, dtype: float64 year
2022    1468.367347
Freq: A-DEC, Name: sales_amount, dtype: float64
```

3. Aggregate the sales data by region and calculate the total sales amount for each region.

In [38]:
```
region_sales = sales_data.groupby('region_name')['sales_amount'].sum()
region_sales
```

Out[38]:
```
region_name
Central       10100
East Coast    22400
Midwest       23600
Mountain       8100
Northeast     12400
Northwest     14500
South         13400
Southeast     11900
Southwest      8800
West Coast    18700
Name: sales_amount, dtype: int64
```

4. Group the sales data by customer segment and calculate the average sales amount for each segment.

In [39]:
```
sales_data['customer_segment'] = sales_data['customer_name'].apply(lambda x: x.split()[0
segment_average_sales = sales_data.groupby('customer_segment')['sales_amount'].mean()
```

5. Aggregate the sales data by sales representative and calculate the total sales amount for each customer

In [40]:
```
sales_data['customer_segment'] = sales_data['customer_name'].apply(lambda x: x.split()[0
segment_average_sales = sales_data.groupby('customer_segment')['sales_amount'].mean()
segment_average_sales
```

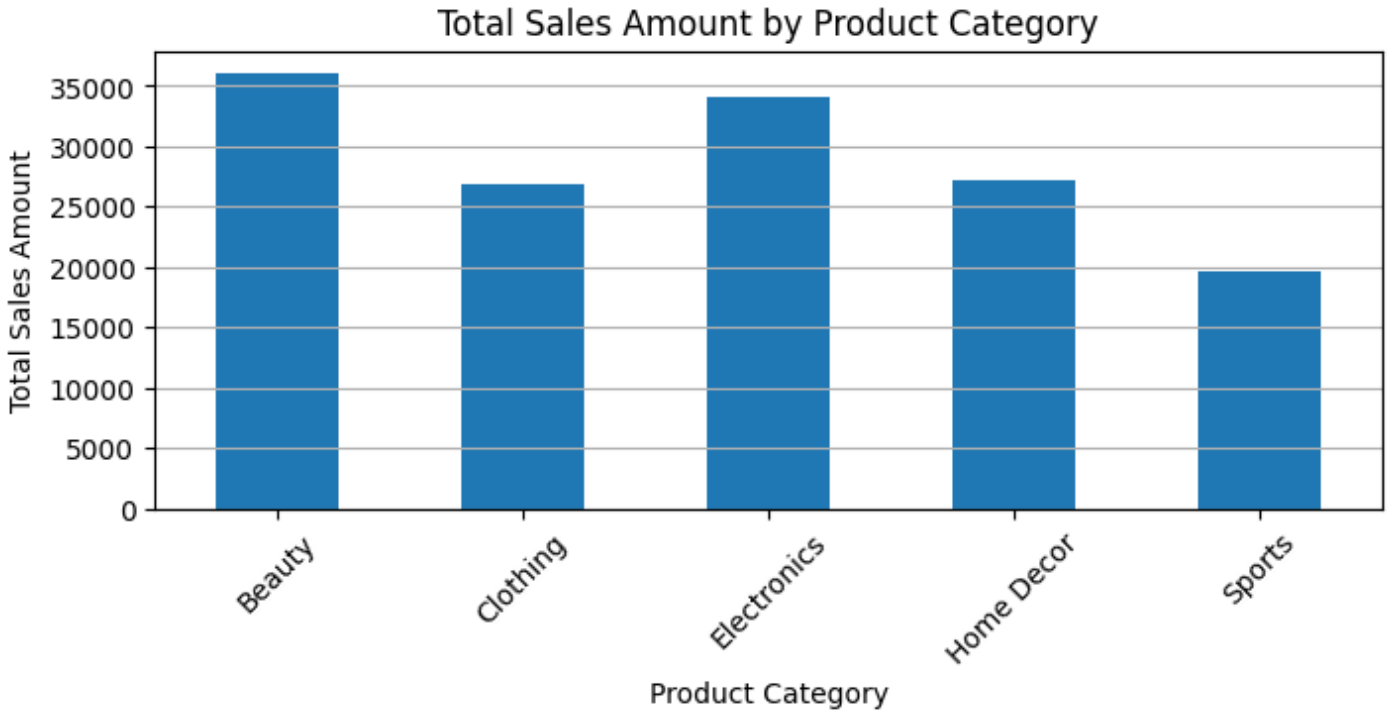
Out[40]:
```
customer_segment
Abigail          800.000000
Addison         1500.000000
Alexander       1900.000000
Amanda          1800.000000
Amelia          1390.000000
Andrew           700.000000
Ava             1433.333333
Benjamin         900.000000
Brian           1000.000000
Charlotte       1700.000000
Christopher      800.000000
Daniel          1800.000000
David           1800.000000
Elijah          1575.000000
Emily            900.000000
Emma             700.000000
Ethan           1400.000000
Evelyn          1900.000000
Grace            800.000000
Harper          1210.000000
Henry           1700.000000
Isabella        1330.000000
Jackson         1600.000000
Jacob           1650.000000
James            900.000000
Jane            1500.000000
Jason           2200.000000
Jessica         2500.000000
John            1000.000000
Kevin           1300.000000
Laura           1100.000000
Liam            1400.000000
Lily            2200.000000
Lisa            1200.000000
Logan           1400.000000
Lucas           1250.000000
Mason           1662.500000
Megan           2500.000000
Mia             1730.000000
Michael         1850.000000
Michelle        1300.000000
Mila            1625.000000
Noah            1812.500000
Oliver          1300.000000
Olivia          1800.000000
Penelope        2200.000000
Rachel          1700.000000
Robert           800.000000
Ryan            1700.000000
Samuel          1900.000000
Scarlett        1200.000000
Sophia          1475.000000
Steven           600.000000
William         1400.000000
Name: sales_amount, dtype: float64
```

Level 5: Data Visualization with Matplotlib

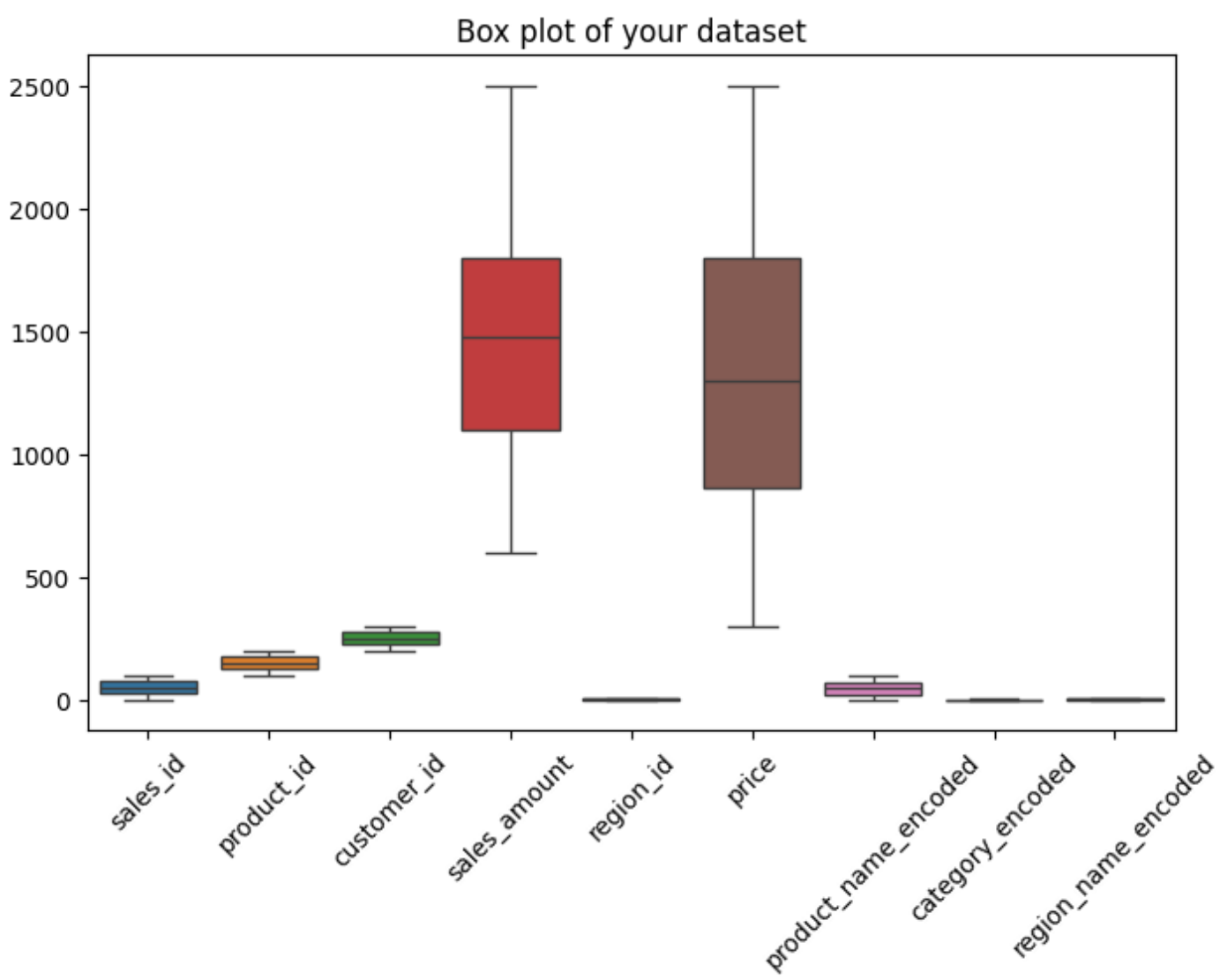
1. Create a bar chart to visualize the total sales amount by product category.

In [41]:
```
df=cleaned_sales.copy()
```

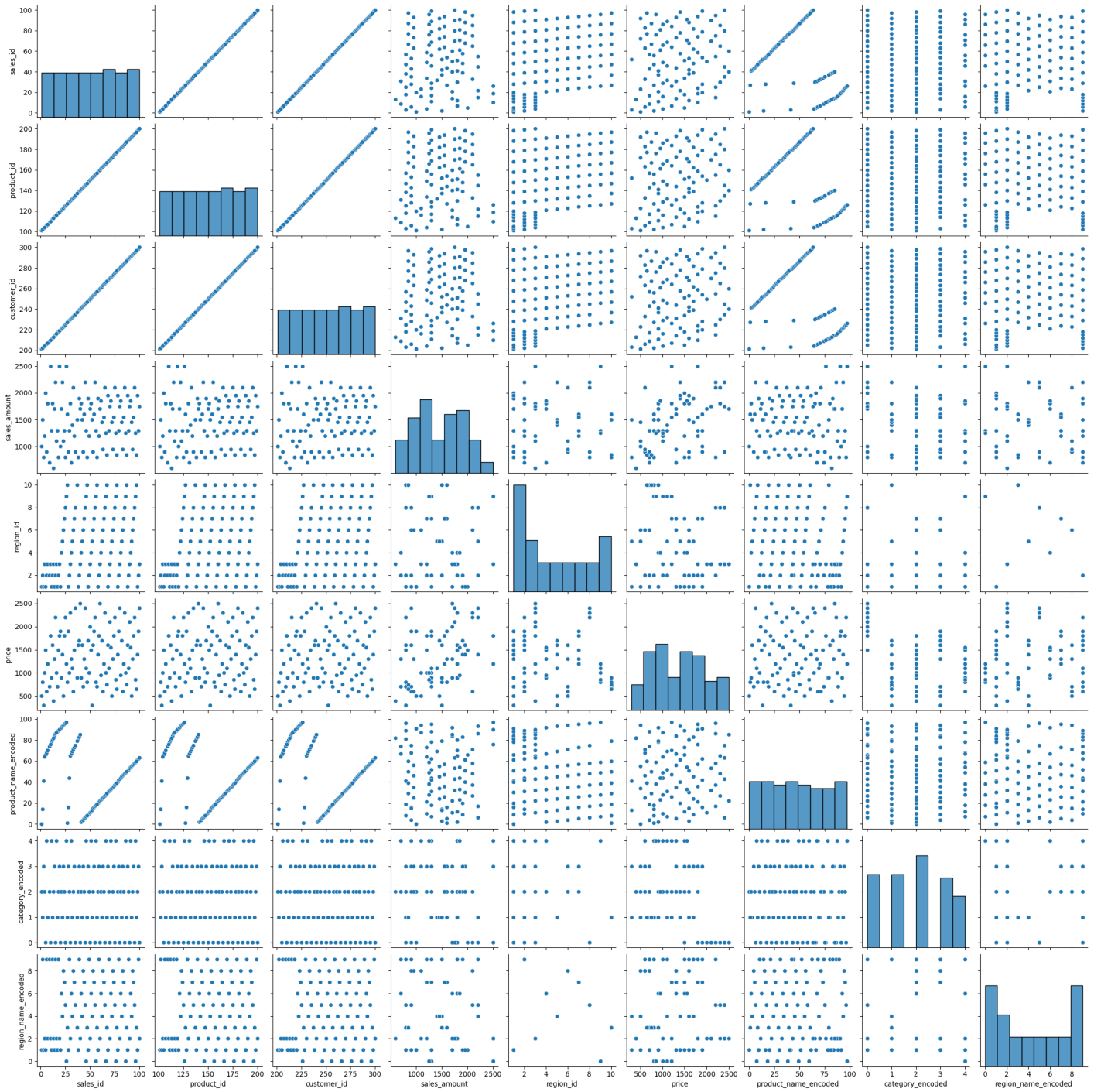
In [42]:
```
sales_by_category=df.groupby('category')['sales_amount'].sum()
plt.figure(figsize=(8, 3))
sales_by_category.plot(kind='bar')
plt.title('Total Sales Amount by Product Category')
plt.xlabel('Product Category')
plt.ylabel('Total Sales Amount')
plt.xticks(rotation=45)
plt.grid(axis='y')
plt.show()
```



In [43]:
```
import matplotlib.pyplot as plt
import seaborn as sns
plt.figure(figsize=(8, 5))
sns.boxplot(data=cleaned_sales)
plt.title('Box plot of your dataset')
plt.xticks(rotation=45)  # Rotate x-axis labels for better readability
plt.show()
```



In [44]:
```
sns.pairplot(df)
plt.show()
```



2. Generate a line plot to show the sales trend over time

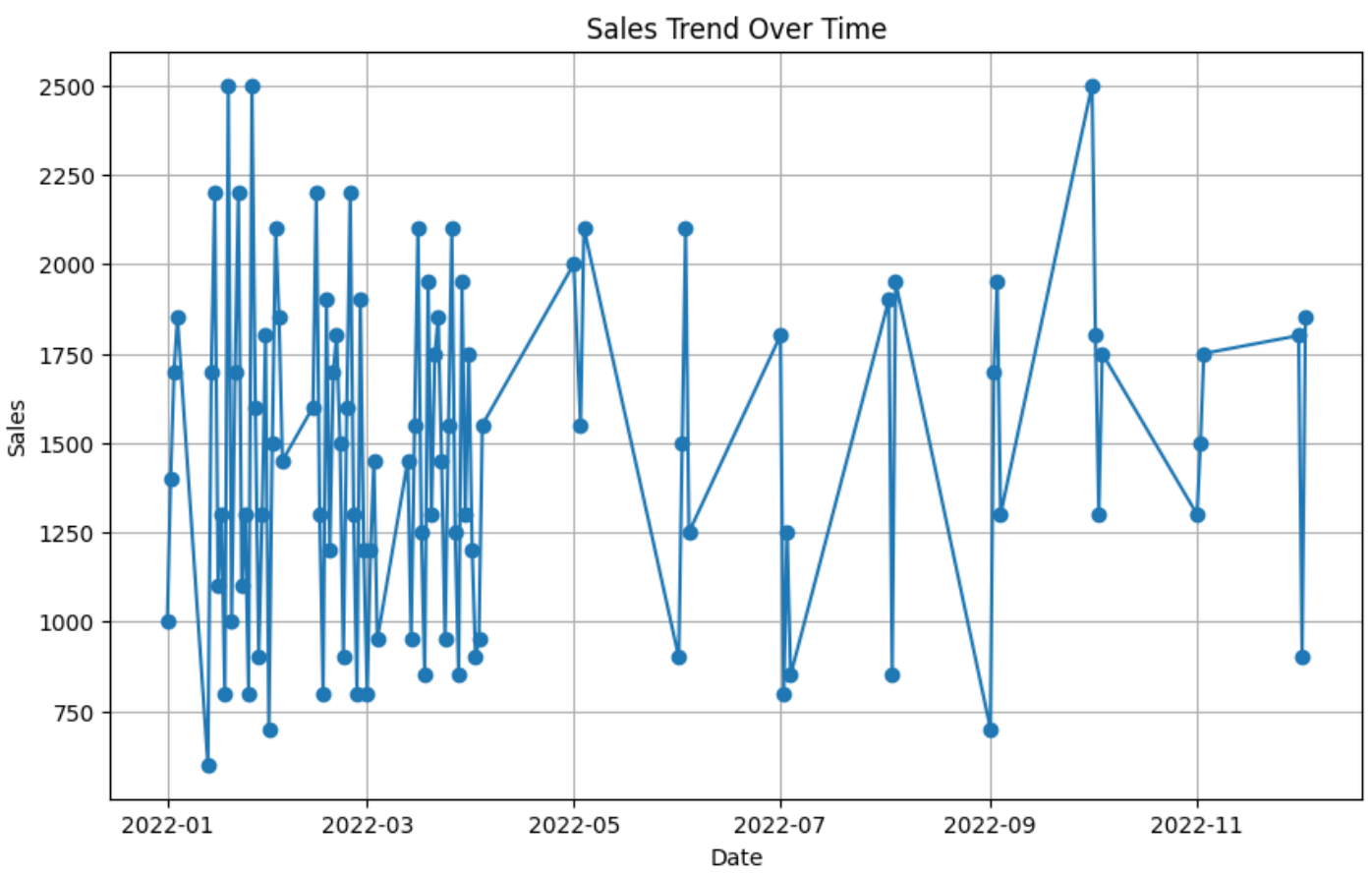
In [45]:
```
df['date'] = pd.to_datetime(df['date'], format="mixed")
# converted the date column so that we can use for time series analysis
df
```

Out[45]:

	sales_id	product_id	customer_id	sales_amount	date	region_id	product_name	cate
0	1	101	201	1000	2022-01-01	1	Product A	Electro
1	3	103	203	800	2022-03-01	1	Product C	H D
2	5	105	205	2000	2022-05-01	1	Product E	Be
4	11	111	211	1300	2022-11-01	1	Product K	Sp
5	14	114	214	1700	2022-01-14	1	Product N	H D
...	...	...	...	...	...	...	...	
95	57	157	257	800	2022-02-26	10	Product BE	Clot
96	67	167	267	850	2022-08-03	10	Product BO	Clot
97	77	177	277	850	2022-03-18	10	Product BY	Clot
98	87	187	287	850	2022-03-28	10	Product CI	Clot
99	97	197	297	850	2022-07-04	10	Product CS	Clot

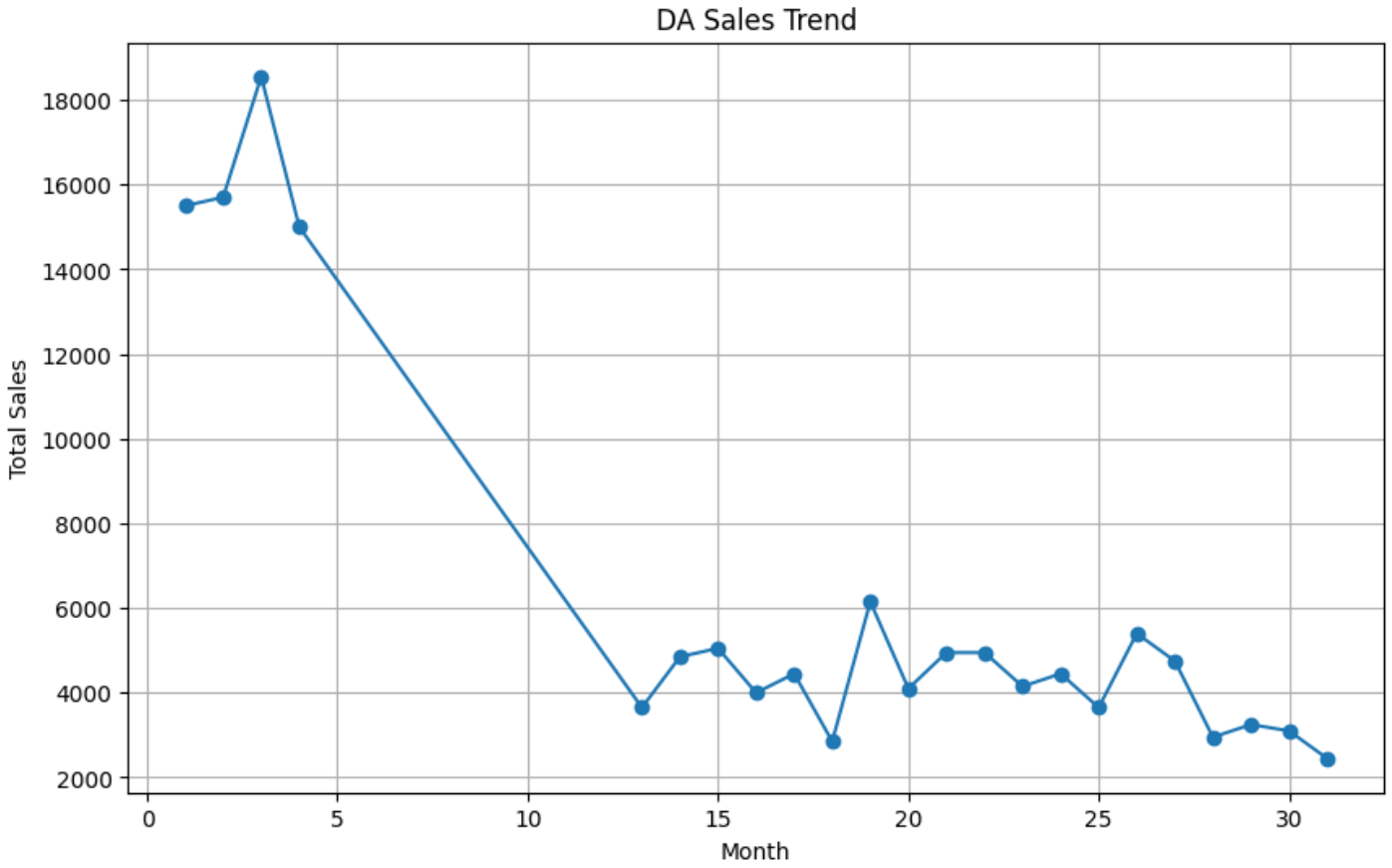
98 rows × 16 columns

In [46]:
```
#sorted the date data for time series analysis
df_sort=df.sort_values('date')
plt.figure(figsize=(10, 6))
plt.plot(df_sort['date'], df_sort['sales_amount'], marker='o', linestyle='-')
plt.title('Sales Trend Over Time')
plt.xlabel('Date')
plt.ylabel('Sales')
plt.grid(True)
plt.show()
```

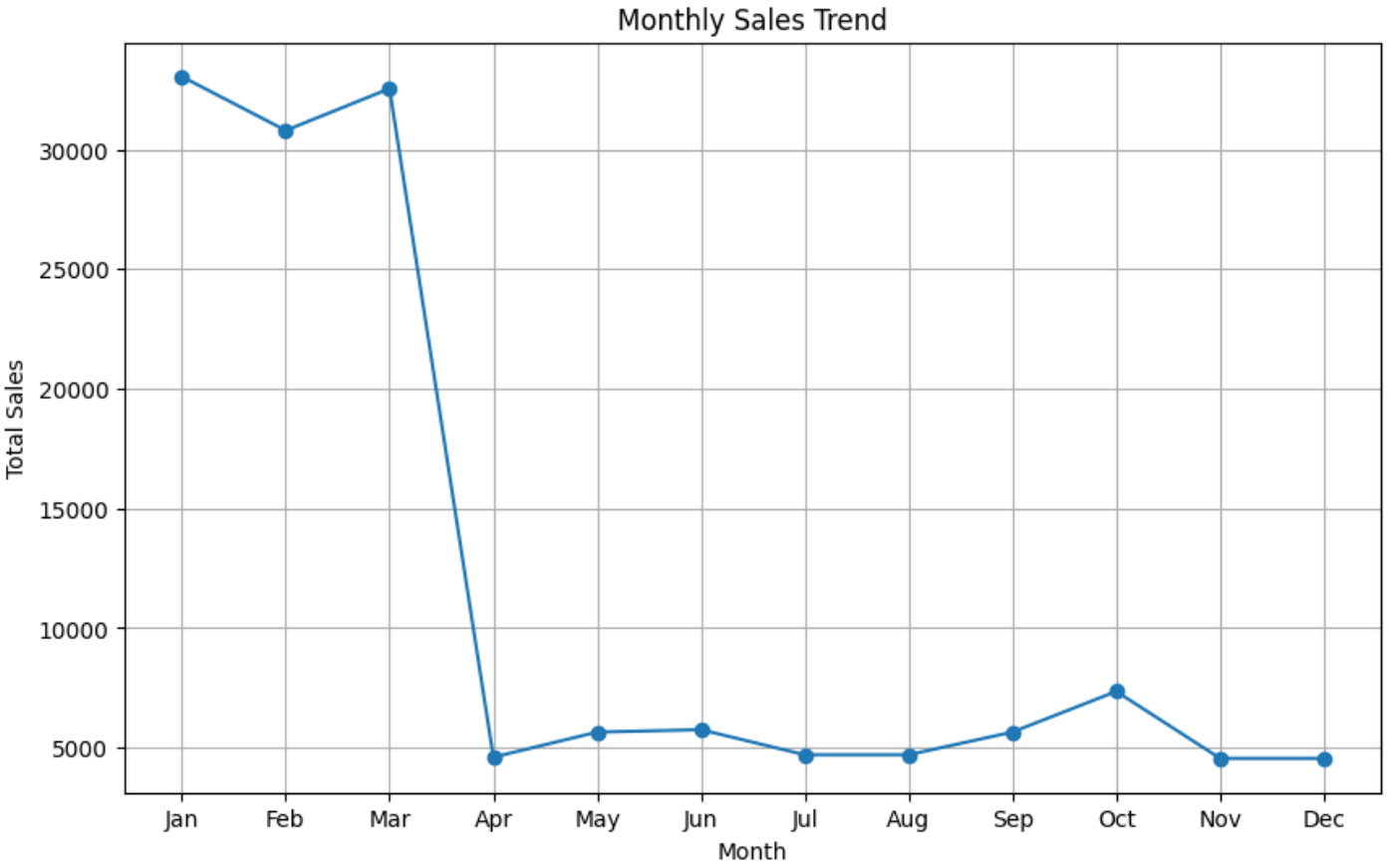


In [47]:
```
#df_sort['month'] = df_sort['date'].dt.month
```

In [48]:
```
df_sort['day'] = df_sort['date'].dt.day

daily_sales = df_sort.groupby('day')['sales_amount'].sum()


# Create a line plot for sales trend over months
plt.figure(figsize=(10, 6))
plt.plot(daily_sales.index, daily_sales.values, marker='o', linestyle='-')
plt.title('DA Sales Trend')
plt.xlabel('Month')
plt.ylabel('Total Sales')
plt.grid(True)
plt.show()
```

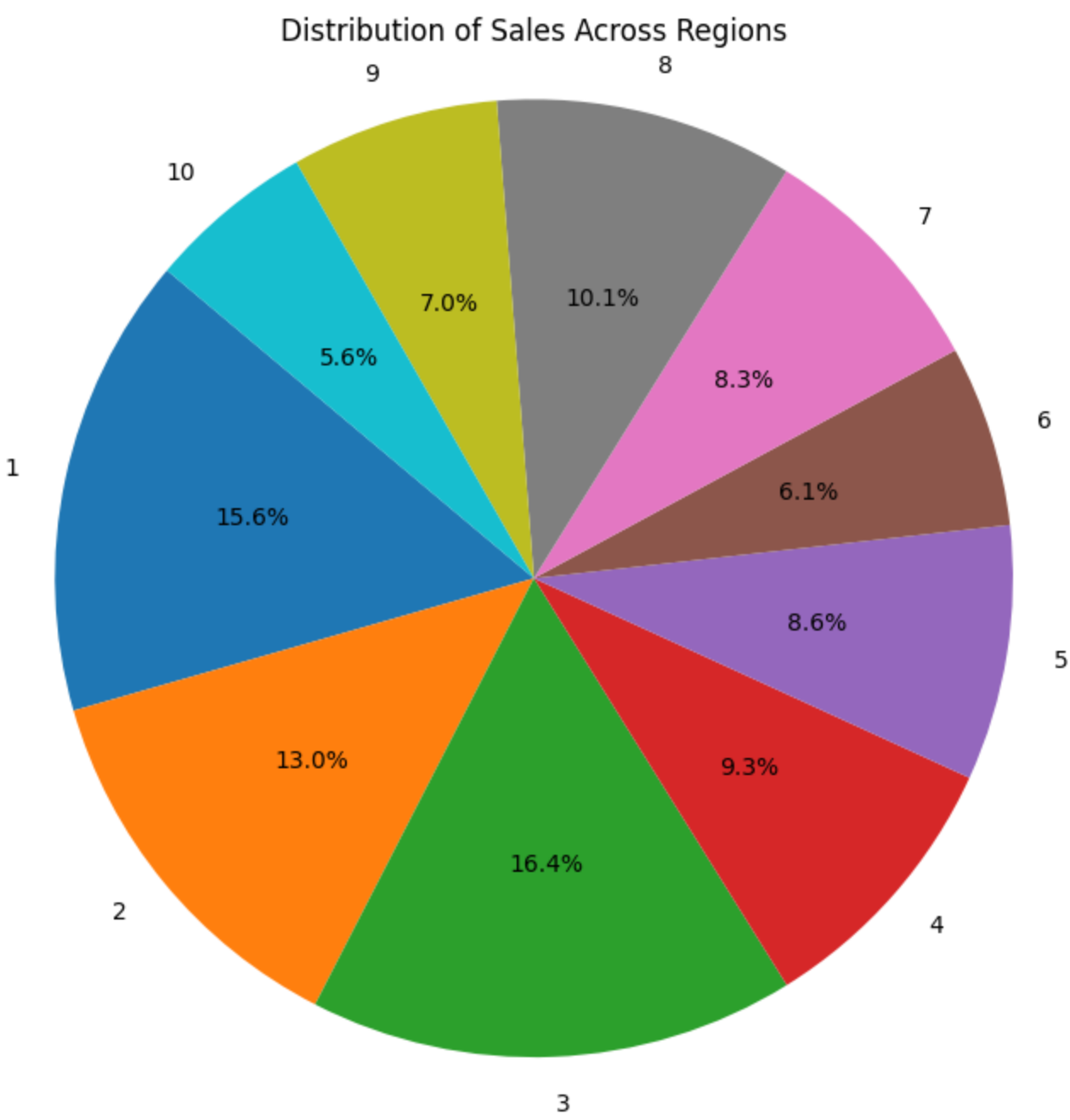


In [49]:
```
df_sort['month'] = df_sort['date'].dt.month

monthly_sales = df_sort.groupby('month')['sales_amount'].sum()

monthly_sales
# Create a line plot for sales trend over months
plt.figure(figsize=(10, 6))
plt.plot(monthly_sales.index, monthly_sales.values, marker='o', linestyle='-')
plt.title('Monthly Sales Trend')
plt.xlabel('Month')
plt.ylabel('Total Sales')
plt.xticks(range(1, 13), ['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul', 'Aug', 'Sep',
plt.grid(True)
plt.show()
```



3. Create a pie chart to represent the distribution of sales across different regions.

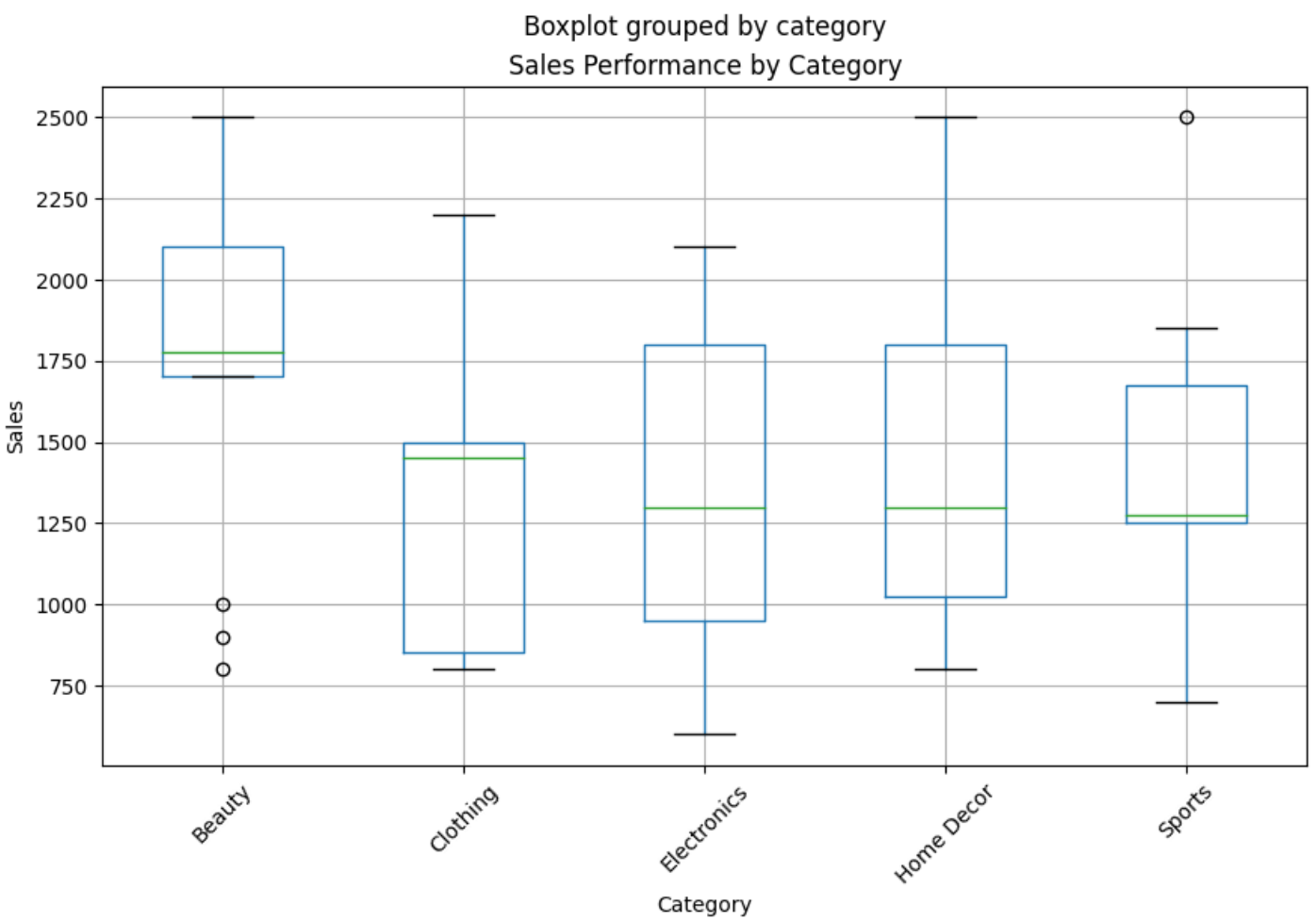
In [50]:
```
sales_by_region=df_sort.groupby('region_id')['sales_amount'].sum()

plt.figure(figsize=(8, 8))
plt.pie(sales_by_region, labels=sales_by_region.index, autopct='%1.1f%%', startangle=140
plt.title('Distribution of Sales Across Regions')
plt.axis('equal')  # Equal aspect ratio ensures that pie is drawn as a circle.
plt.show()
```



4. Generate a boxplot to compare the sales performance of different customer segments.

In [51]:
```
plt.figure(figsize=(8, 6))
df.boxplot(column='sales_amount', by='category', figsize=(10, 6))
plt.title('Sales Performance by Category')
plt.xlabel('Category')
plt.ylabel('Sales')
plt.xticks(rotation=45)
plt.grid(True)
plt.show()
```

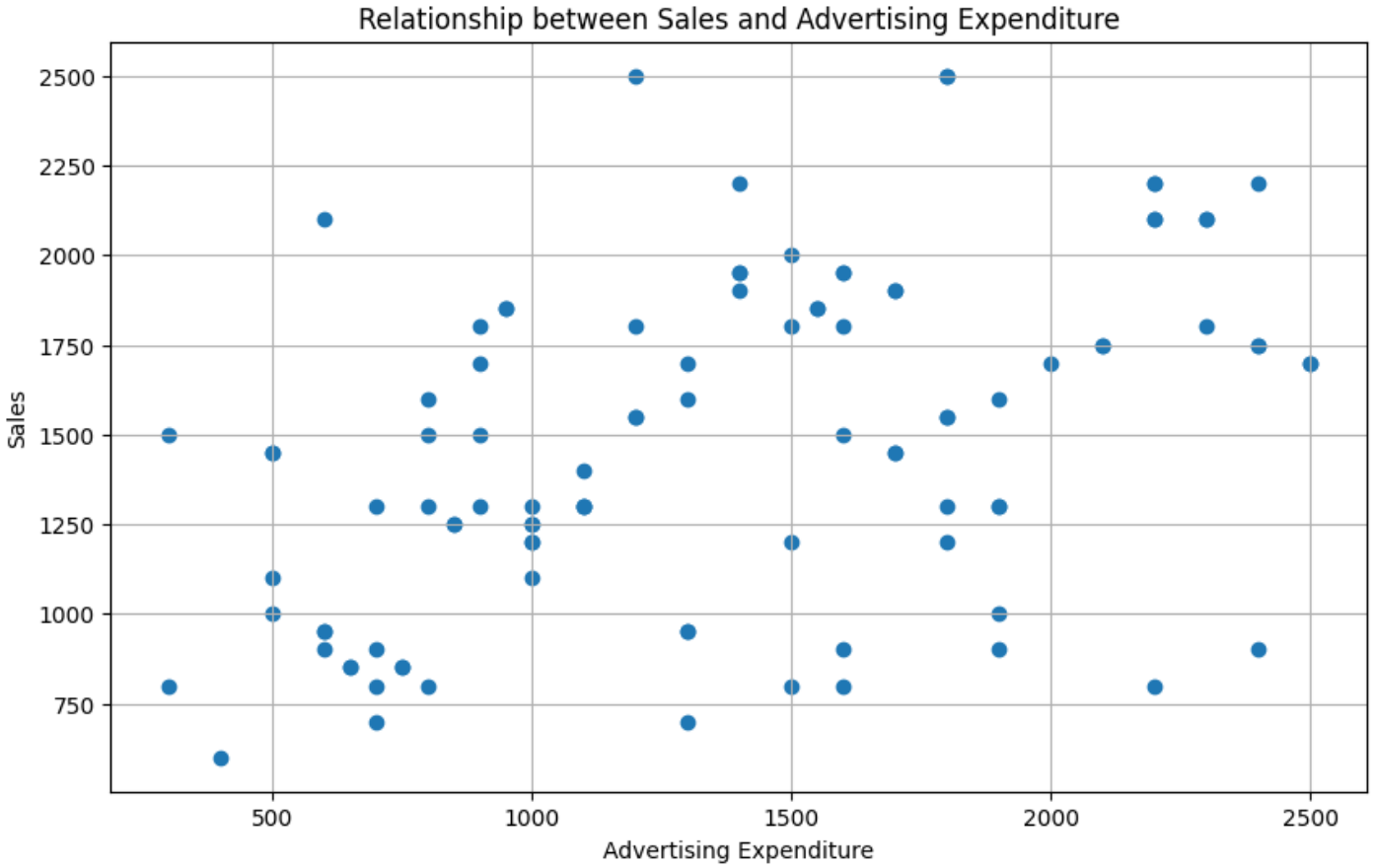
```
<Figure size 800x600 with 0 Axes>
```



In [52]:
```
#Beauty product data is heavily skewed and mostly has high end products but there are so
#low valued products, whereas Electronics,Home Decor , and sports has average value prod
```

5. Visualize the relationship between sales and advertising expenditure using a scatter plot.

In [53]:
```
plt.figure(figsize=(10, 6))
plt.scatter(df_sort['price'], df_sort['sales_amount'])
plt.title('Relationship between Sales and Advertising Expenditure')
plt.xlabel('Advertising Expenditure')
plt.ylabel('Sales')
plt.grid(True)
plt.show()
```



In [54]:
```
# there is no some sort of relationship between price and the sales_amount, but it has w
```

In [55]:
```
df
```

Out[55]:

	sales_id	product_id	customer_id	sales_amount	date	region_id	product_name	cate
0	1	101	201	1000	2022-01-01	1	Product A	Electr
1	3	103	203	800	2022-03-01	1	Product C	H D
2	5	105	205	2000	2022-05-01	1	Product E	Be
4	11	111	211	1300	2022-11-01	1	Product K	Sp
5	14	114	214	1700	2022-01-14	1	Product N	H D
...	...	...	...	...	...	...	...	
95	57	157	257	800	2022-02-26	10	Product BE	Clot
96	67	167	267	850	2022-08-03	10	Product BO	Clot
97	77	177	277	850	2022-03-18	10	Product BY	Clot
98	87	187	287	850	2022-03-28	10	Product CI	Clot
99	97	197	297	850	2022-07-04	10	Product CS	Clot

98 rows × 16 columns

Level 6: Advanced Data Manipulation with Numpy

1. Use Numpy to calculate the mean, median, and standard deviation of the sales data.

In [56]:
```
np.median(df.sales_id),np.median(df.product_id),np.median(df.sales_amount),np.median(df.
```

Out[56]:
```
(51.5, 151.5, 1475.0, 4.0, 1300.0)
```

In [57]:
```
np.std(df.sales_amount)
```

Out[57]: 465.2401045182422

2. Perform element-wise arithmetic operations on the sales data (e.g., addition, subtraction, multiplication).

In [58]:
```
# Convert the sales data to a NumPy array
sales_array = np.array(df_sort['sales_amount'])

# Example values for operations
value_to_add = 10
value_to_subtract = 5
value_to_multiply = 2

# Perform element-wise arithmetic operations
sales_added = sales_array + value_to_add
sales_subtracted = sales_array - value_to_subtract
sales_multiplied = sales_array* value_to_multiply
```

In [59]:
```
print(sales_added,sales_subtracted,sales_multiplied)
```

```
[1010 1410 1710 1860  610 1710 2210 1110 1310  810 2510 1010 1710 2210
 1110 1310  810 2510 1610  910 1310 1810  710 1510 2110 1860 1460 1610
 2210 1310  810 1910 1210 1710 1810 1510  910 1610 2210 1310  810 1910
 1210  810 1210 1460  960 1460  960 1560 2110 1260  860 1960 1310 1760
 1860 1460  960 1560 2110 1260  860 1960 1310 1760 1210  910  960 1560
 2010 1560 2110  910 1510 2110 1260 1810  810 1260  860 1910  860 1960
  710 1710 1960 1310 2510 1810 1310 1760 1310 1510 1760 1810  910 1860] [ 995 1395 1695 1
845  595 1695 2195 1095 1295  795 2495  995 1695 2195
 1095 1295  795 2495 1595  895 1295 1795  695 1495 2095 1845 1445 1595
 2195 1295  795 1895 1195 1695 1795 1495  895 1595 2195 1295  795 1895
 1195  795 1195 1445  945 1445  945 1545 2095 1245  845 1945 1295 1745
 1845 1445  945 1545 2095 1245  845 1945 1295 1745 1195  895  945 1545
 1995 1545 2095  895 1495 2095 1245 1795  795 1245  845 1895  845 1945
  695 1695 1945 1295 2495 1795 1295 1745 1295 1495 1745 1795  895 1845] [2000 2800 3400 3
700 1200 3400 4400 2200 2600 1600 5000 2000 3400 4400
 2200 2600 1600 5000 3200 1800 2600 3600 1400 3000 4200 3700 2900 3200
 4400 2600 1600 3800 2400 3400 3600 3000 1800 3200 4400 2600 1600 3800
 2400 1600 2400 2900 1900 2900 1900 3100 4200 2500 1700 3900 2600 3500
 3700 2900 1900 3100 4200 2500 1700 3900 2600 3500 2400 1800 1900 3100
 4000 3100 4200 1800 3000 4200 2500 3600 1600 2500 1700 3800 1700 3900
 1400 3400 3900 2600 5000 3600 2600 3500 2600 3000 3500 3600 1800 3700]
```

3. Use Numpy to reshape the sales data into a different dimension.

In [60]:
```
sales_reshaped = sales_array.reshape(1, 1, 1, -1)
sales_reshaped
```

Out[60]:
```
array([[[[1000, 1400, 1700, 1850,  600, 1700, 2200, 1100, 1300,  800,
          2500, 1000, 1700, 2200, 1100, 1300,  800, 2500, 1600,  900,
          1300, 1800,  700, 1500, 2100, 1850, 1450, 1600, 2200, 1300,
           800, 1900, 1200, 1700, 1800, 1500,  900, 1600, 2200, 1300,
           800, 1900, 1200,  800, 1200, 1450,  950, 1450,  950, 1550,
          2100, 1250,  850, 1950, 1300, 1750, 1850, 1450,  950, 1550,
          2100, 1250,  850, 1950, 1300, 1750, 1200,  900,  950, 1550,
          2000, 1550, 2100,  900, 1500, 2100, 1250, 1800,  800, 1250,
           850, 1900,  850, 1950,  700, 1700, 1950, 1300, 2500, 1800,
          1300, 1750, 1300, 1500, 1750, 1800,  900, 1850]]]])
```

4. Apply broadcasting to perform operations on arrays with different shapes.

In [61]:
```
four_dim=sales_reshaped
one_dim=sales_array

result=four_dim+one_dim
result.shape
```

Out[61]:
```
(1, 1, 1, 98)
```

5. Use Numpy to perform matrix multiplication on sales data arrays.

In [62]:
```
mult_dim=np.dot(four_dim,one_dim)
mult_dim.shape
```

Out[62]:
```
(1, 1, 1)
```

Level 7: Advanced Pandas Queries

1. Use Pandas to filter the sales data for a specifc time period (e.g., quarter or year).

In [63]:
```
df.columns
```

Out[63]:
```
Index(['sales_id', 'product_id', 'customer_id', 'sales_amount', 'date',
       'region_id', 'product_name', 'category', 'price', 'customer_name',
       'email', 'address', 'region_name', 'product_name_encoded',
       'category_encoded', 'region_name_encoded'],
      dtype='object')
```

In [64]:
```
# Extract the quarter and year from the 'date' column
df['quarter'] = df['date'].dt.quarter
df['year'] = df['date'].dt.year

# Filter the sales data for a specific quarter and year
target_quarter = 1
target_year = 2022

filtered_sales_data = df[(df['quarter'] == target_quarter) & (df['year'] == target_year)

# Display the filtered sales data
print("Filtered Sales Data for Quarter {} of Year {}:".format(target_quarter, target_yea
print(filtered_sales_data)
```

```
Filtered Sales Data for Quarter 1 of Year 2022:
    sales_id  product_id  customer_id  sales_amount       date  region_id  \
0          1         101          201          1000 2022-01-01          1
1          3         103          203           800 2022-03-01          1
5         14         114          214          1700 2022-01-14          1
6         17         117          217          1300 2022-01-17          1
7         20         120          220          1000 2022-01-20          1
..       ...         ...          ...           ...        ...        ...
92        27         127          227          1600 2022-01-27         10
94        47         147          247           800 2022-02-16         10
95        57         157          257           800 2022-02-26         10
97        77         177          277           850 2022-03-18         10
98        87         187          287           850 2022-03-28         10

   product_name     category  price      customer_name                email  \
0     Product A  Electronics    500           John Doe      john@example.com
1     Product C   Home Decor    300       Robert Brown    robert@example.com
5     Product N   Home Decor   1300        Rachel Hall    rachel@example.com
6     Product Q     Clothing    700     Michelle Perez  michelle@example.com
7     Product T       Beauty   1900         Brian Hall     brian@example.com
..          ...          ...    ...                ...                  ...
92   Product AA     Clothing    800  Isabella Martinez  isabella@example.com
94   Product AU     Clothing    800       Abigail King   abigail@example.com
95   Product BE     Clothing    700      Grace Ramirez     grace@example.com
97   Product BY     Clothing    650       Sophia Reyes    sophia@example.com
98   Product CI     Clothing    750     Harper Coleman    harper@example.com

                         address region_name  product_name_encoded  \
0       123 Main St, Anytown, USA  East Coast                     0
1        789 Oak St, Anycity, USA  East Coast                    41
5      852 Cedar St, Nowhere, USA  East Coast                    84
6   357 Spruce St, Othertown, USA  East Coast                    88
7        159 Elm St, Nowhere, USA  East Coast                    91
..                           ...        ...                   ...
92   587 Maple St, Somewhere, USA    Mountain                     1
94   587 Maple St, Somewhere, USA    Mountain                     8
95     586 Maple St, Anytown, USA    Mountain                    19
97     586 Maple St, Anytown, USA    Mountain                    39
98   587 Maple St, Somewhere, USA    Mountain                    50

    category_encoded  region_name_encoded  quarter  year
0                  2                    1        1  2022
1                  3                    1        1  2022
5                  3                    1        1  2022
6                  1                    1        1  2022
7                  0                    1        1  2022
..               ...                  ...      ...   ...
92                 1                    3        1  2022
94                 1                    3        1  2022
95                 1                    3        1  2022
97                 1                    3        1  2022
98                 1                    3        1  2022

[66 rows x 18 columns]
```

2. Apply boolean indexing to select rows based on multiple conditions.

In [65]:
```
[(df['region_name']=='East Coast') & (df['price']>500)]
```

Out[65]:
```
[0     False
 1     False
 2      True
 4      True
 5      True
       ...
 95    False
 96    False
 97    False
 98    False
 99    False
 Length: 98, dtype: bool]
```

3. Use Pandas groupby and aggregate functions to calculate custom metrics.

In [66]:
```
# Define custom aggregate functions
def total_sales_amount(series):
    return series.sum()

def average_price(series):
    return series.mean()

# Group by 'category' and apply custom aggregate functions
custom_metrics = df.groupby('category').agg(
    total_sales=('sales_amount', total_sales_amount),
    average_price=('price', average_price)
)
```

In [67]:
```
custom_metrics
```

Out[67]:

category	total_sales	average_price
Beauty	36100	2195.000000
Clothing	26900	955.000000
Electronics	34100	1124.000000
Home Decor	27200	1415.789474
Sports	19600	1085.714286

4. Combine multiple DataFrames using merge or join operations.

In [68]:
```
# I am goin to join the custom metrics with the original data frame using join

# Reset index of 'custom_metrics' to make 'category' a column
custom_metrics.reset_index(inplace=True)

# Merge the original DataFrame with the DataFrame containing custom metrics on the 'cate
merged_df = pd.merge(df, custom_metrics, on='category', how='left')

# Display the merged DataFrame
print("Merged DataFrame:")
merged_df
```

```
Merged DataFrame:
```

Out[68]:

	sales_id	product_id	customer_id	sales_amount	date	region_id	product_name	cate
0	1	101	201	1000	2022-01-01	1	Product A	Electro
1	3	103	203	800	2022-03-01	1	Product C	H D
2	5	105	205	2000	2022-05-01	1	Product E	Be
3	11	111	211	1300	2022-11-01	1	Product K	Sp
4	14	114	214	1700	2022-01-14	1	Product N	H D
...	...	...	...	...	...	...	...	
93	57	157	257	800	2022-02-26	10	Product BE	Clot
94	67	167	267	850	2022-08-03	10	Product BO	Clot
95	77	177	277	850	2022-03-18	10	Product BY	Clot
96	87	187	287	850	2022-03-28	10	Product CI	Clot
97	97	197	297	850	2022-07-04	10	Product CS	Clot

98 rows × 20 columns

In [69]:
```
merged_df.address
```

Out[69]:
```
0        123 Main St, Anytown, USA
1         789 Oak St, Anycity, USA
2        654 Pine St, Nowhere, USA
3       147 Oak St, Othertown, USA
4       852 Cedar St, Nowhere, USA
                  ...
93      586 Maple St, Anytown, USA
94    587 Maple St, Somewhere, USA
95      586 Maple St, Anytown, USA
96    587 Maple St, Somewhere, USA
97      586 Maple St, Anytown, USA
Name: address, Length: 98, dtype: object
```

Level 10: Advanced Data Analysis Techniques with NumPy and Matplotlib

1. Use historical sales data to create a time series array.

In [70]:
```
# performed in earlier section as part of answer
```

2. Visualize the original and smoothed time series using Matplotlib line plot

In [71]:
```
# performed in earlier section as part of answer
```

3. Create word frequency arrays to identify common themes in customer reviews

In [72]:
```
# Created a dataframe of 98 customer reviews with 5 types of reviews
num_reviews = 98
num_review_types = 5
review_types = np.random.randint(1, num_review_types+1, num_reviews)
review_texts = [
    'This product exceeded my expectations. Highly recommended!',
    'Not satisfied with the product. Poor quality and late delivery.',
    'I absolutely love this product! It has made my life so much easier.',
    'Average product. Not bad, but not great either.',
    'Terrible experience. Product arrived damaged and customer service was unhelpful.'
]

reviews = [review_texts[i-1] for i in review_types]
customer_review_df = pd.DataFrame({
    'review_text': reviews
})

# Merge the customer review dataframe with the existing dataframe
review_df = pd.concat([merged_df, customer_review_df], axis=1)

# Display the merged dataframe
review_df.head()
```

Out[72]:

	sales_id	product_id	customer_id	sales_amount	date	region_id	product_name	categ
0	1	101	201	1000	2022-01-01	1	Product A	Electron
1	3	103	203	800	2022-03-01	1	Product C	Ho De
2	5	105	205	2000	2022-05-01	1	Product E	Bea
3	11	111	211	1300	2022-11-01	1	Product K	Spo
4	14	114	214	1700	2022-01-14	1	Product N	Ho De

5 rows × 21 columns

In [73]:

```
from nltk.tokenize import word_tokenize
from collections import Counter

# Assuming df is your merged dataframe
customer_reviews = review_df['review_text'].dropna().tolist()

# Tokenize words and count frequencies
word_freq = Counter()
for review in customer_reviews:
    words = word_tokenize(review.lower())  # Convert to lowercase for consistency
    word_freq.update(words)

# Convert word frequencies to dataframe
word_freq_df = pd.DataFrame(word_freq.items(), columns=['Word', 'Frequency'])

# Sort dataframe by frequency
word_freq_df = word_freq_df.sort_values(by='Frequency', ascending=False)

# Display the dataframe
word_freq_df.reset_index()
```

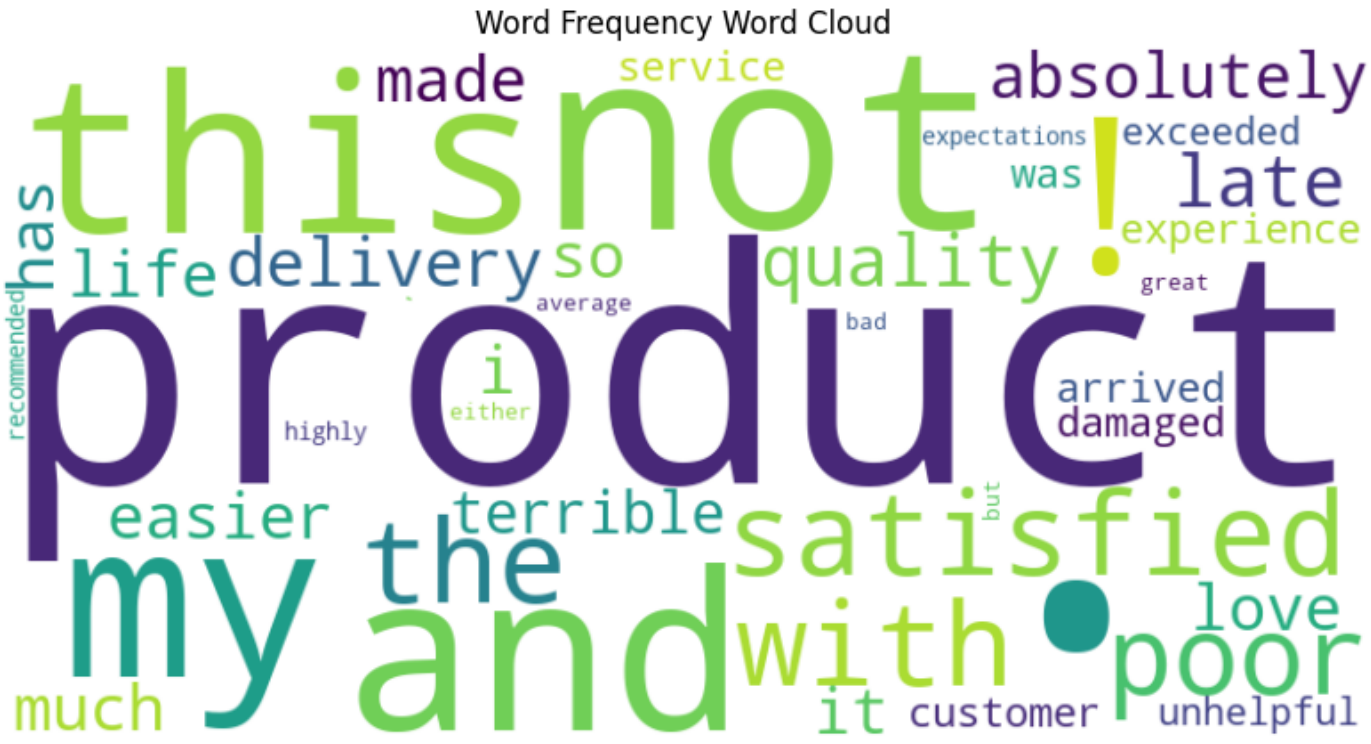
Out[73]:

	index	Word	Frequency
0	14	.	156
1	4	product	98
2	24	not	56
3	19	and	41
4	3	this	40
5	5	!	40
6	9	my	40
7	26	with	22
8	31	delivery	22
9	25	satisfied	22
10	28	poor	22
11	29	quality	22
12	30	late	22
13	27	the	22
14	1	absolutely	21
15	0	i	21
16	12	much	21
17	11	so	21
18	10	life	21
19	8	made	21
20	7	has	21
21	6	it	21
22	2	love	21
23	13	easier	21
24	39	expectations	19
25	38	exceeded	19
26	40	highly	19
27	21	service	19
28	23	unhelpful	19
29	22	was	19
30	20	customer	19
31	18	damaged	19
32	17	arrived	19
33	16	experience	19
34	15	terrible	19

	index	Word	Frequency
35	41	recommended	19
36	32	average	17
37	33	bad	17
38	34	,	17
39	35	but	17
40	36	great	17
41	37	either	17

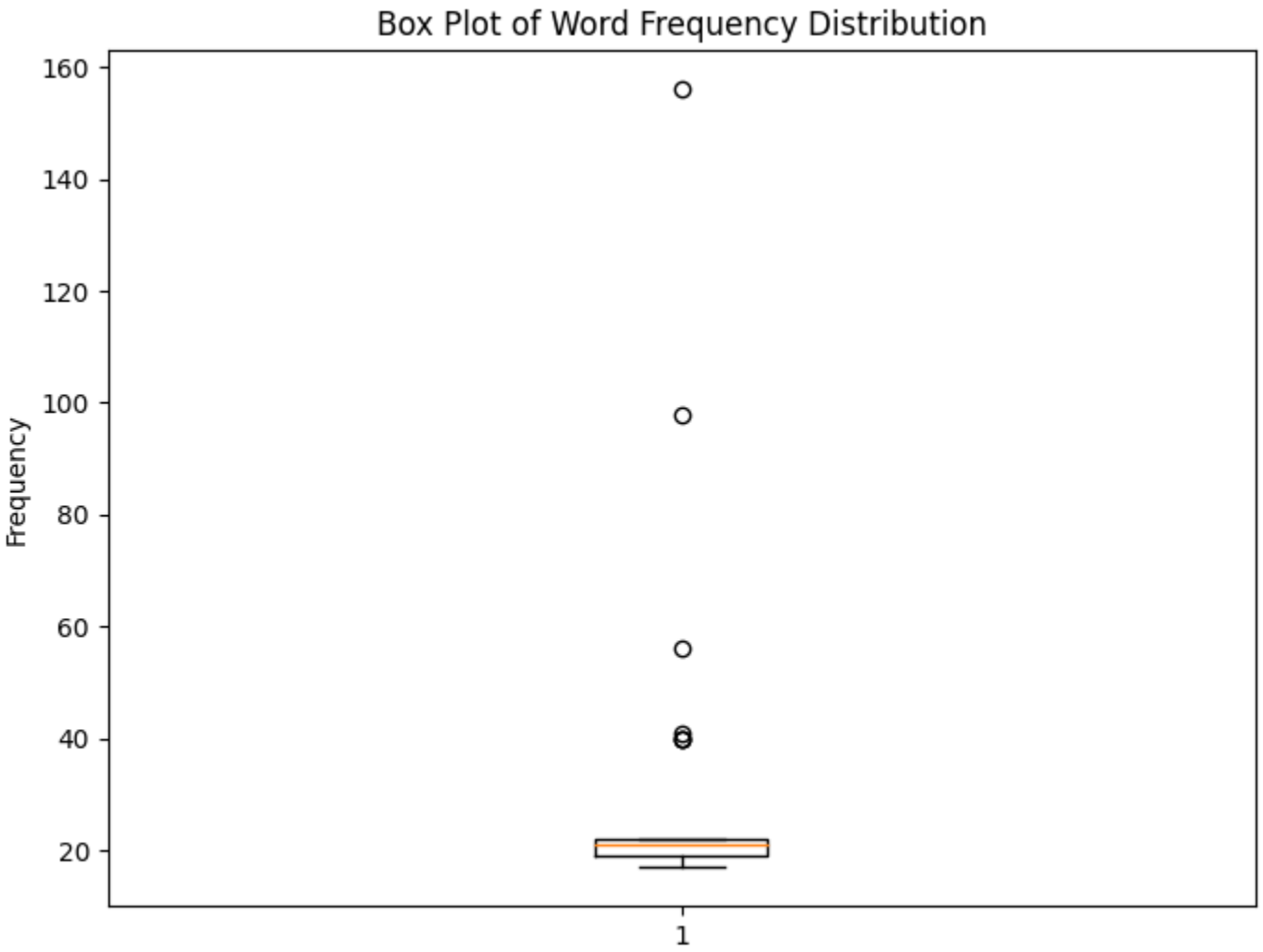
4. Visualize the word frequency distribution using Matplotlib histograms or word clouds.

In [74]:
```
from wordcloud import WordCloud

# Generate word cloud
wordcloud = WordCloud(width=800, height=400, background_color='white').generate_from_fre

# Plot word cloud
plt.figure(figsize=(10, 6))
plt.imshow(wordcloud, interpolation='bilinear')
plt.title('Word Frequency Word Cloud')
plt.axis('off')
plt.show()
```

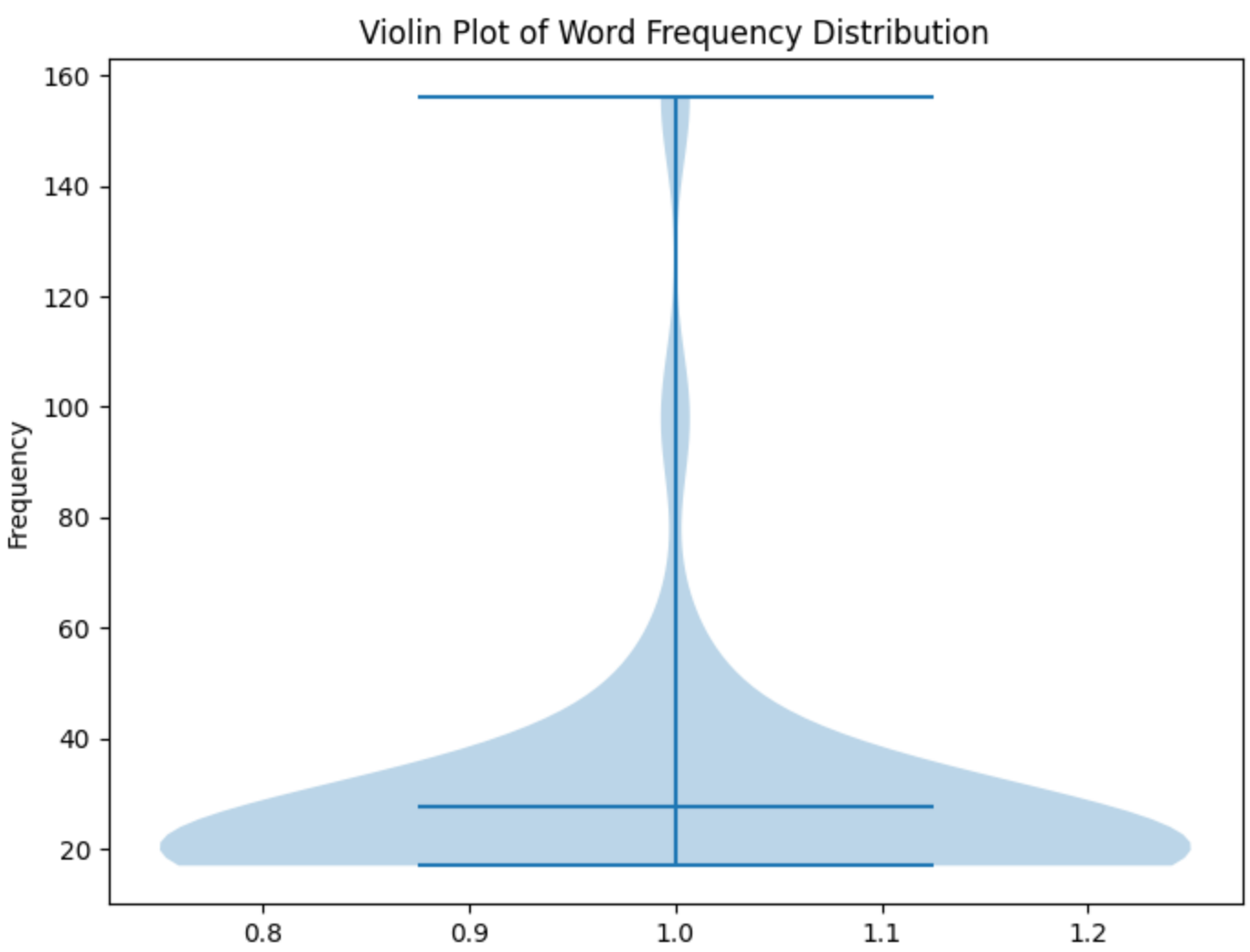


5. Visualize statistical results using Matplotlib box plots, violin plots, or histograms

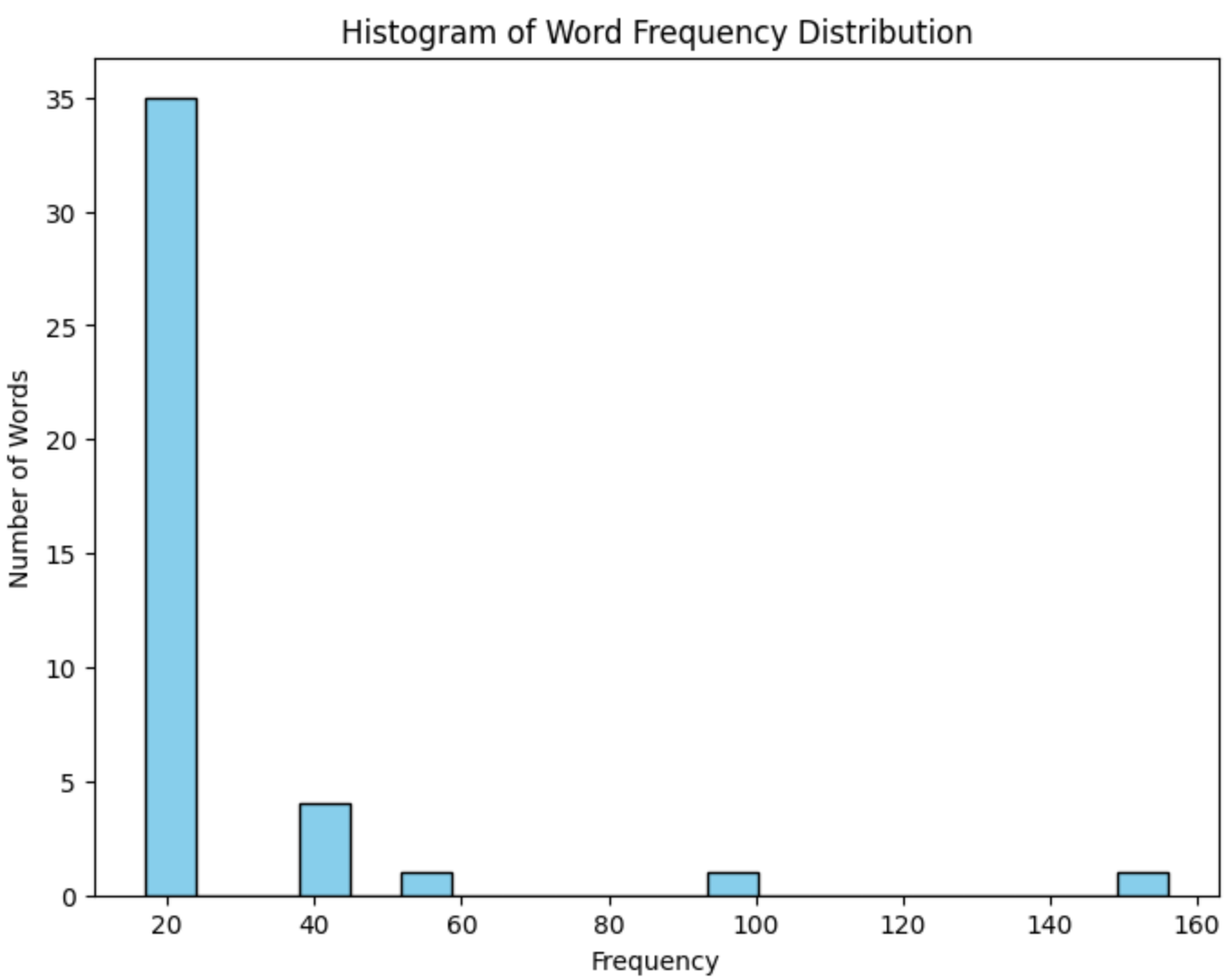
In [75]:
```
# Assuming word_freq_df is your word frequency dataframe
# Extract word frequencies
data = word_freq_df['Frequency']

# Create a box plot
plt.figure(figsize=(8, 6))
plt.boxplot(data)
plt.title('Box Plot of Word Frequency Distribution')
plt.ylabel('Frequency')
plt.show()
```



In [76]:
```
# Assuming word_freq_df is your word frequency dataframe
# Extract word frequencies
data = word_freq_df['Frequency']

# Create a violin plot
plt.figure(figsize=(8, 6))
plt.violinplot(data, showmeans=True)
plt.title('Violin Plot of Word Frequency Distribution')
plt.ylabel('Frequency')
plt.show()
```



In [77]:
```
# Assuming word_freq_df is your word frequency dataframe
# Extract word frequencies
data = word_freq_df['Frequency']

# Create a histogram
plt.figure(figsize=(8, 6))
plt.hist(data, bins=20, color='skyblue', edgecolor='black')
plt.title('Histogram of Word Frequency Distribution')
plt.xlabel('Frequency')
plt.ylabel('Number of Words')
plt.show()
```


Histogram of Word Frequency Distribution
Number of Words
Frequency
0
5
10
15
20
25
30
35
20
40
60
80
100
120
140
160