In [1]:

```
import numpy as np
import pandas as pd
df=pd.read_csv('rawdata.csv')
df.columns
```

Out[1]:

```
Index(['Customer_ID', 'Age', 'Gender', 'Income', 'Spending_Score',
       'Credit_Score', 'Loan_Amount', 'Previous_Defaults', 'Marketing_Spend',
       'Purchase_Frequency', 'Seasonality', 'Sales', 'Customer_Churn',
       'Defaulted', 'seasonality_date'],
      dtype='object')
```

In [5]:

```
df.describe()
```

Out[5]:

	Customer_ID	Age	Income	Spending_Score	Credit_Score	Loan_Amount	Previous_Defaults	Marketing_Spend	Purchase_Frequency	Sales	Customer_Churn	Defaulted
count	500.000000	500.000000	500.000000	500.000000	500.000000	500.000000	500.00000	500.000000	500.000000	500.000000	500.000000	500.000000
mean	250.500000	44.220000	84398.055556	50.862000	573.411111	28456.928889	0.97400	10558.128000	15.350000	54378.954000	0.254000	0.190000
std	144.481833	15.036082	38049.398377	29.125101	149.302942	11788.254534	0.82625	5508.219008	8.475327	27263.106468	0.435734	0.392694
min	1.000000	18.000000	20055.000000	1.000000	300.000000	5163.000000	0.00000	1024.000000	1.000000	5203.000000	0.000000	0.000000
25%	125.750000	32.000000	51746.250000	25.750000	453.000000	19052.500000	0.00000	6041.500000	8.000000	30507.500000	0.000000	0.000000
50%	250.500000	45.000000	84398.055556	51.000000	573.411111	28456.928889	1.00000	10754.000000	16.000000	54032.500000	0.000000	0.000000
75%	375.250000	57.000000	117492.000000	77.000000	682.250000	37371.500000	2.00000	15099.750000	23.000000	78523.750000	1.000000	0.000000
max	500.000000	69.000000	149922.000000	99.000000	848.000000	49936.000000	2.00000	19990.000000	29.000000	99835.000000	1.000000	1.000000

In [2]:

```
df.isnull().sum()
```

Out[2]:

```
Customer_ID            0
Age                    0
Gender                 0
Income                50
Spending_Score         0
Credit_Score          50
Loan_Amount           50
Previous_Defaults      0
Marketing_Spend        0
Purchase_Frequency     0
Seasonality            0
Sales                  0
Customer_Churn         0
Defaulted              0
seasonality_date       0
dtype: int64
```

In [3]:

```
# fill missing value by mean and double checked
df[['Income','Credit_Score','Loan_Amount']] = df[['Income','Credit_Score','Loan_Amount']].fillna(df[['Income','Credit_Score','Loan_Amount']].mean())
df.isnull().sum()
```

Out[3]:

```
Customer_ID           0
Age                   0
Gender                0
Income                0
Spending_Score        0
Credit_Score          0
Loan_Amount           0
Previous_Defaults     0
Marketing_Spend       0
Purchase_Frequency    0
Seasonality           0
Sales                 0
Customer_Churn        0
Defaulted             0
seasonality_date      0
dtype: int64
```

In [4]:

```
df.duplicated().sum()
```

Out[4]:

```
np.int64(0)
```

In [6]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 500 entries, 0 to 499
Data columns (total 15 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   Customer_ID         500 non-null    int64
 1   Age                 500 non-null    int64
 2   Gender              500 non-null    object
 3   Income              500 non-null    float64
 4   Spending_Score      500 non-null    int64
 5   Credit_Score        500 non-null    float64
 6   Loan_Amount         500 non-null    float64
 7   Previous_Defaults   500 non-null    int64
 8   Marketing_Spend     500 non-null    int64
 9   Purchase_Frequency  500 non-null    int64
 10  Seasonality         500 non-null    object
 11  Sales               500 non-null    int64
 12  Customer_Churn      500 non-null    int64
 13  Defaulted           500 non-null    int64
 14  seasonality_date    500 non-null    object
dtypes: float64(3), int64(9), object(3)
memory usage: 58.7+ KB
```

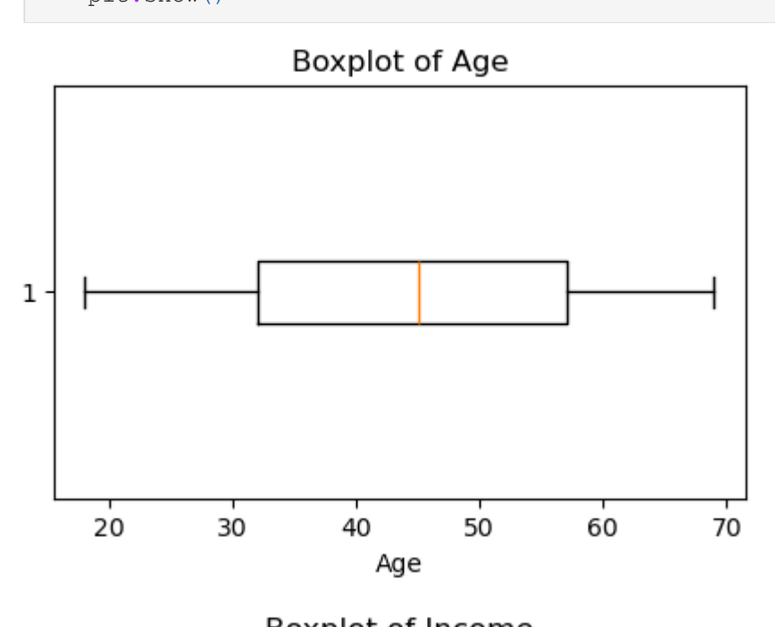
In [7]:

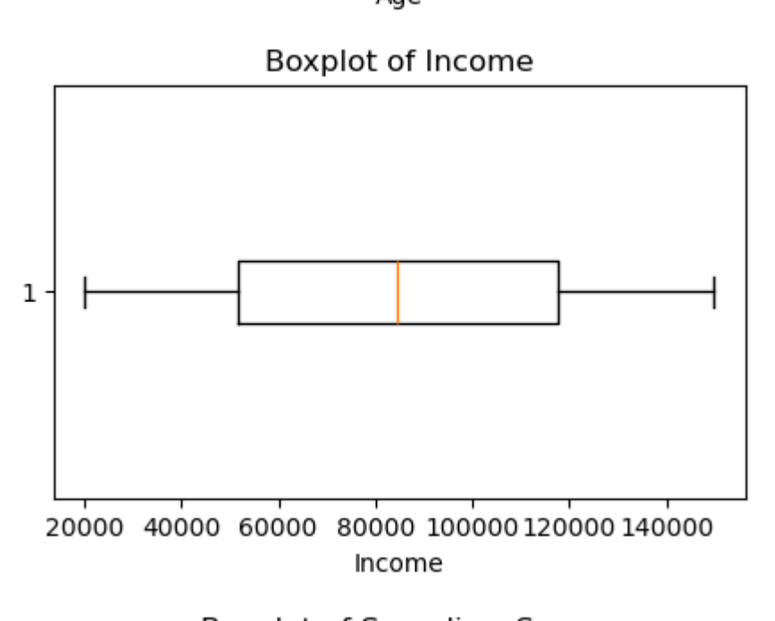
```
df['seasonality_date'] = pd.to_datetime(df['seasonality_date'], format='%d/%m/%Y')
print(df.dtypes)
```

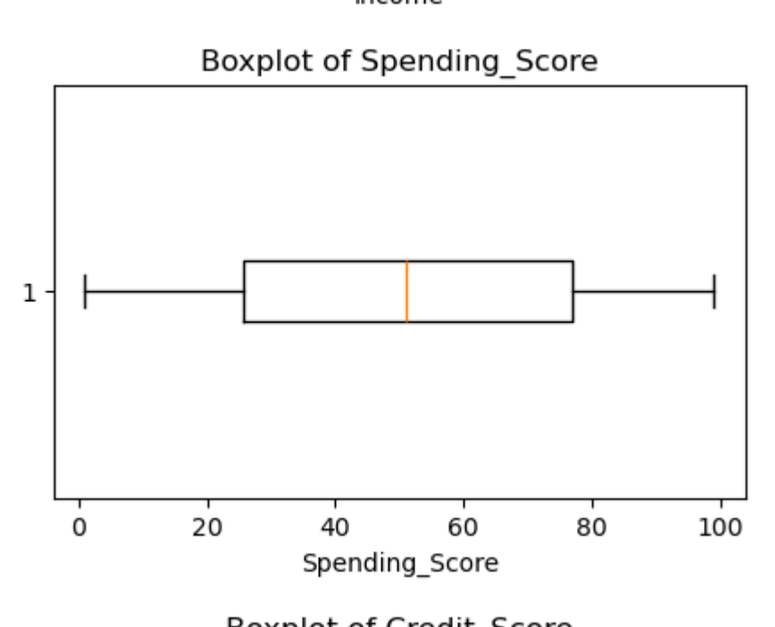
```
Customer_ID                    int64
Age                            int64
Gender                        object
Income                       float64
Spending_Score                 int64
Credit_Score                 float64
Loan_Amount                  float64
Previous_Defaults              int64
Marketing_Spend                int64
Purchase_Frequency             int64
Seasonality                   object
Sales                          int64
Customer_Churn                 int64
Defaulted                      int64
seasonality_date      datetime64[ns]
dtype: object
```

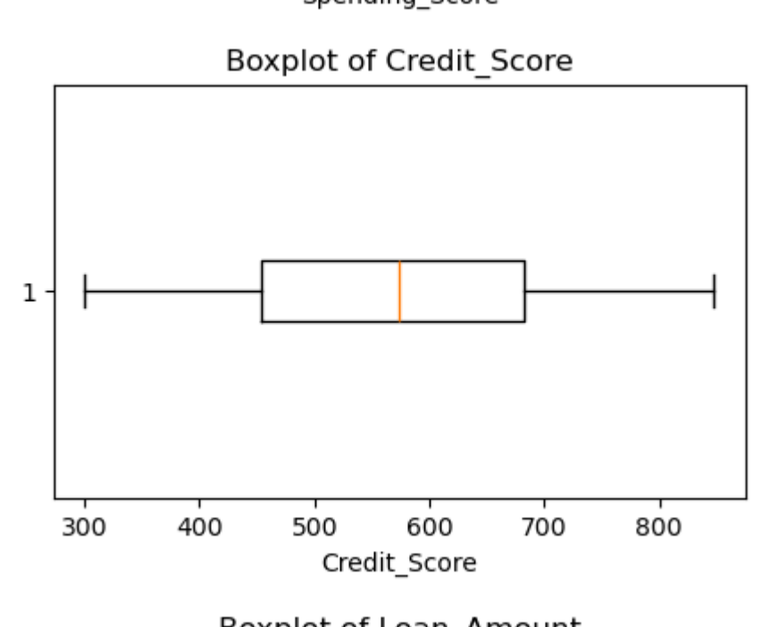
In [11]:

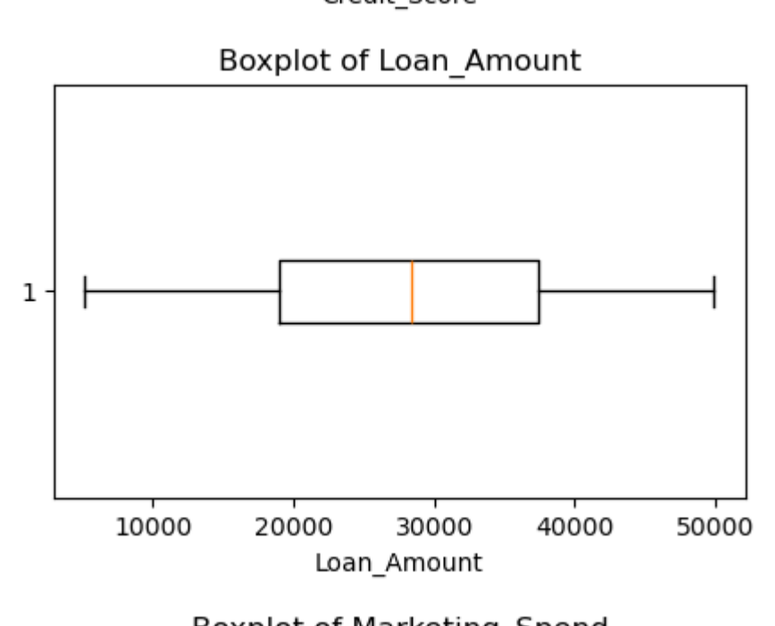
```
import matplotlib.pyplot as plt
numeric_cols = ['Age', 'Income', 'Spending_Score', 'Credit_Score', 'Loan_Amount','Marketing_Spend', 'Sales']
for col in numeric_cols:
    plt.figure(figsize=(5, 3))
    plt.boxplot(df[col], vert=False)
    plt.title(f'Boxplot of {col}')
    plt.xlabel(col)
    plt.show()
```

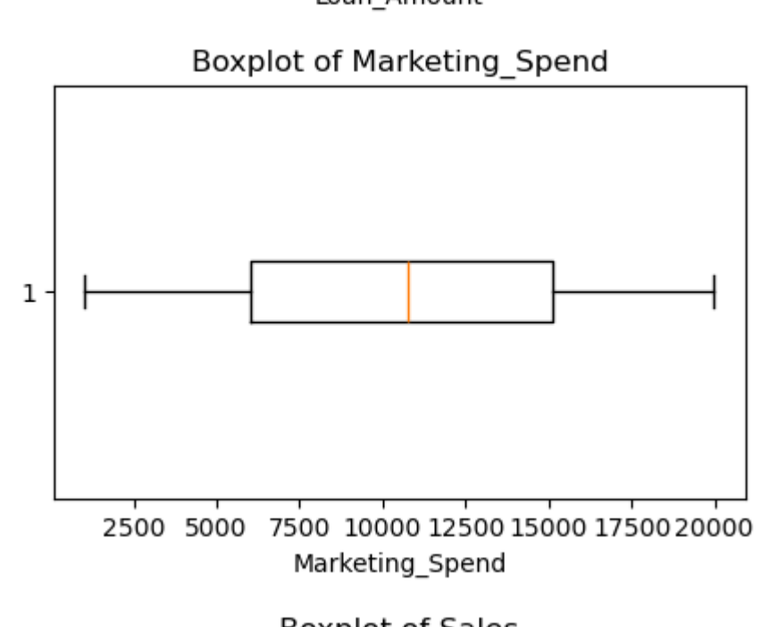


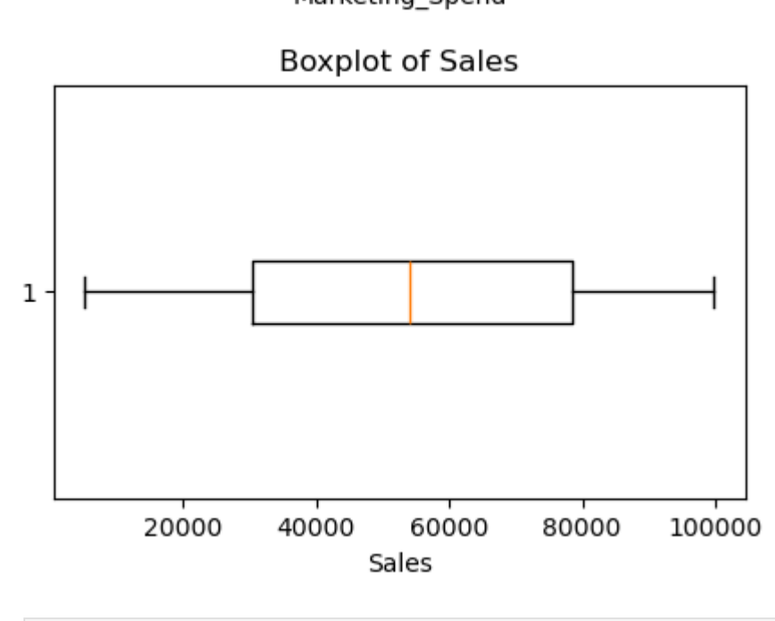












In [12]:

```
numeric_df = df.select_dtypes(include=['number'])
Q1 = numeric_df.quantile(0.25)
Q3 = numeric_df.quantile(0.75)
IQR = Q3 - Q1
df = df[~((numeric_df < (Q1 - 1.5 * IQR)) | (numeric_df > (Q3 + 1.5 * IQR))).any(axis=1)]
print("After removing outliers, rows remaining:", df.shape[0])
```

```
After removing outliers, rows remaining: 405
```

In [14]:

```
df.to_csv("cleaned_data.csv", index=False)
print("Data cleaning complete and saved as 'cleaned_data.csv'")
```

```
Data cleaning complete and saved as 'cleaned_data.csv'
```

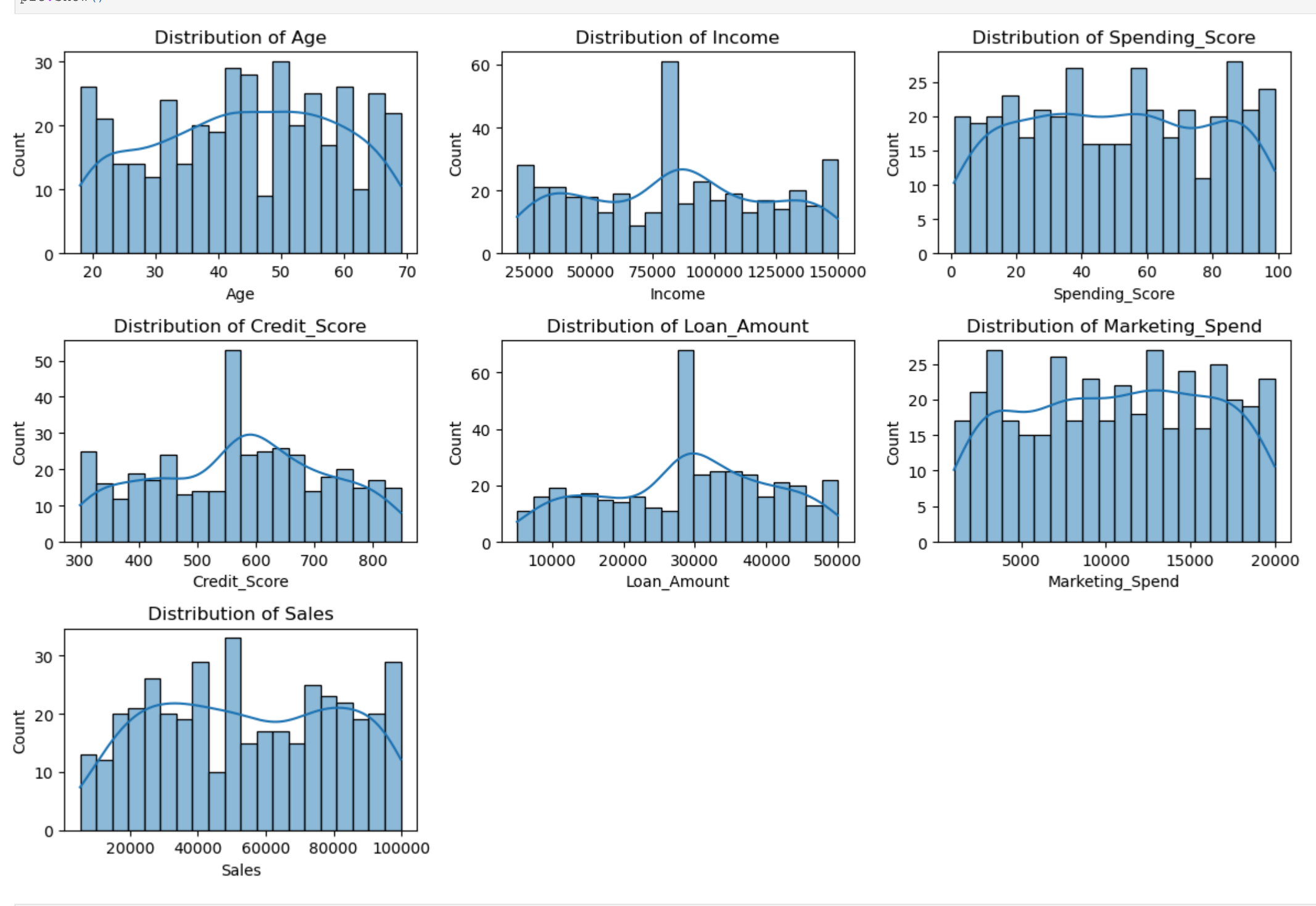
In [2]:

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
df=pd.read_csv("cleaned_data.csv")
print("Shape of data:", df.shape)
print("\nBasic information:")
print(df.info())
print("\nSummary statistics:")
print(df.describe())
```

```
Shape of data: (405, 15)

Basic information:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 405 entries, 0 to 404
Data columns (total 15 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   Customer_ID         405 non-null    int64
 1   Age                 405 non-null    int64
 2   Gender              405 non-null    object
 3   Income              405 non-null    float64
 4   Spending_Score      405 non-null    int64
 5   Credit_Score        405 non-null    float64
 6   Loan_Amount         405 non-null    float64
 7   Previous_Defaults   405 non-null    int64
 8   Marketing_Spend     405 non-null    int64
 9   Purchase_Frequency  405 non-null    int64
 10  Seasonality         405 non-null    object
 11  Sales               405 non-null    int64
 12  Customer_Churn      405 non-null    int64
 13  Defaulted           405 non-null    int64
 14  seasonality_date    405 non-null    object
dtypes: float64(3), int64(9), object(3)
memory usage: 47.6+ KB
None

Summary statistics:
       Customer_ID         Age         Income  Spending_Score  Credit_Score  \
count   405.000000  405.000000     405.000000      405.000000    405.000000
mean    249.153086   44.148148   84311.467490       50.765432    572.757531
std     141.444637   14.727148   37835.082256       29.044806    148.549349
min       1.000000   18.000000   20055.000000        1.000000    300.000000
25%     127.000000   32.000000   50105.000000       26.000000    453.000000
50%     252.000000   45.000000   84398.055556       51.000000    573.411111
75%     368.000000   56.000000  113229.000000       77.000000    681.000000
max     500.000000   69.000000  149922.000000       99.000000    848.000000

        Loan_Amount  Previous_Defaults  Marketing_Spend  Purchase_Frequency  \
count    405.000000         405.000000       405.000000          405.000000
mean   28937.427193           0.987654     10668.259259           15.214815
std    11849.042234           0.824949      5495.941548            8.452299
min     5163.000000           0.000000      1024.000000            1.000000
25%    19566.000000           0.000000      6347.000000            8.000000
50%    28563.000000           1.000000     10915.000000           16.000000
75%    38069.000000           2.000000     15376.000000           23.000000
max    49936.000000           2.000000     19990.000000           29.000000

              Sales  Customer_Churn  Defaulted
count    405.000000      405.000000      405.0
mean   54692.958025        0.241975        0.0
std    27125.289994        0.428809        0.0
min     5336.000000        0.000000        0.0
25%    30805.000000        0.000000        0.0
50%    52574.000000        0.000000        0.0
75%    78616.000000        0.000000        0.0
max    99835.000000        1.000000        0.0
```

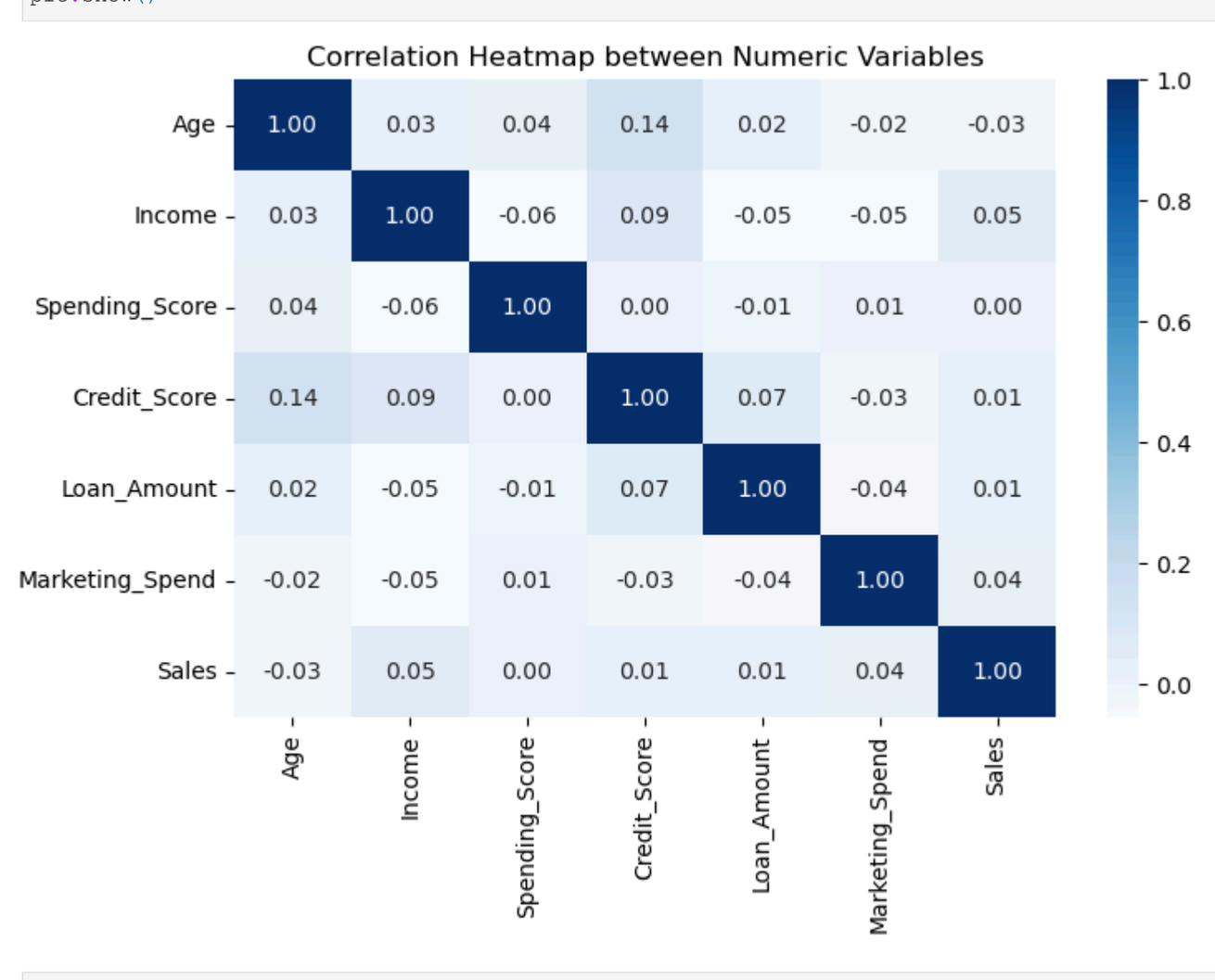
In [3]:

```
numeric_cols = ['Age', 'Income', 'Spending_Score', 'Credit_Score', 'Loan_Amount', 'Marketing_Spend', 'Sales']

plt.figure(figsize=(12, 8))
for i, col in enumerate(numeric_cols, 1):
    plt.subplot(3, 3, i)
    sns.histplot(df[col], kde=True, bins=20)
    plt.title(f"Distribution of {col}")
plt.tight_layout()
plt.show()
```



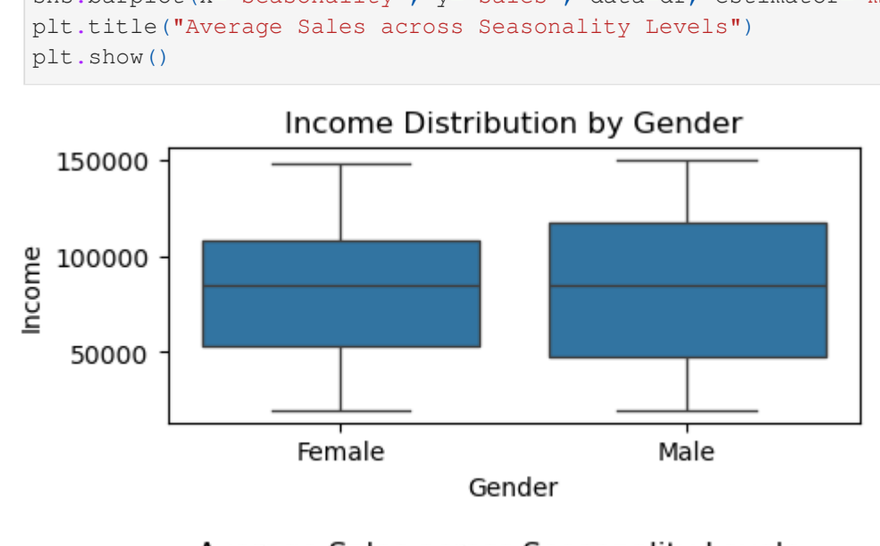
In [10]:

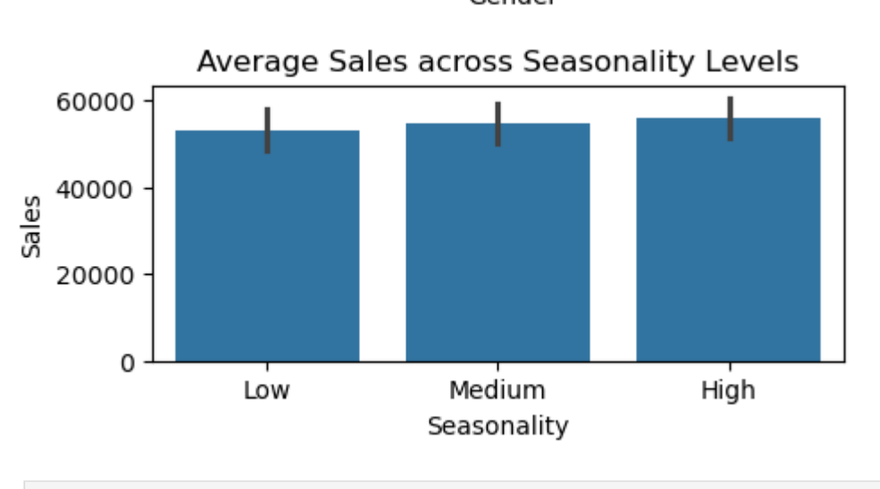
```
# Correlation Heatmap
plt.figure(figsize=(8,5))
corr = df[numeric_cols].corr()
sns.heatmap(corr, annot=True, cmap='Blues', fmt=".2f")
plt.title("Correlation Heatmap between Numeric Variables")
plt.show()
```



In [9]:

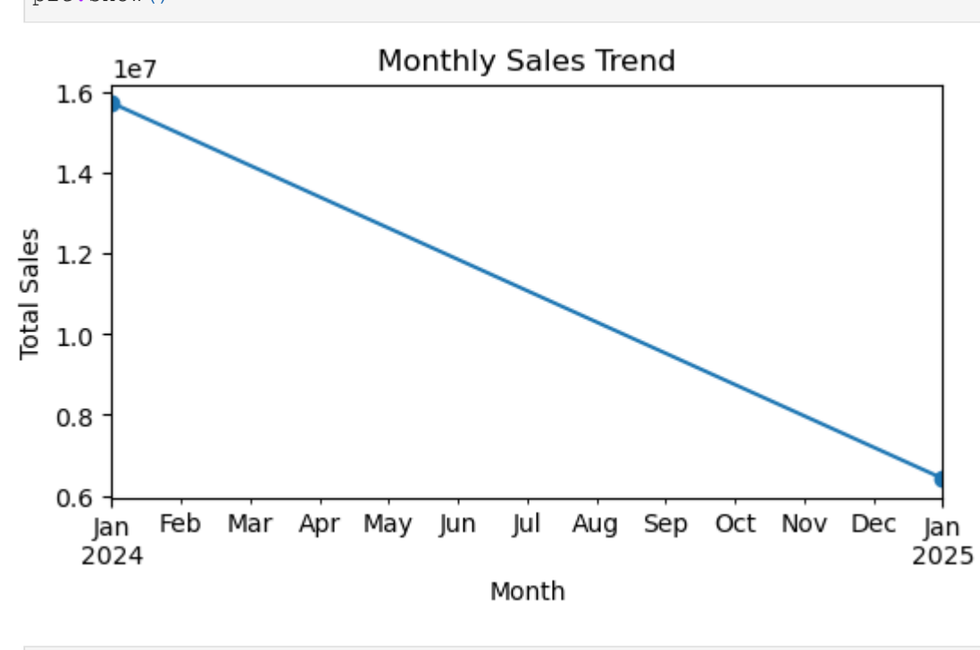
```
# Gender and Seasonality Insights
plt.figure(figsize=(5,2))
sns.boxplot(x='Gender', y='Income', data=df)
plt.title("Income Distribution by Gender")
plt.show()

plt.figure(figsize=(5,2))
sns.barplot(x='Seasonality', y='Sales', data=df, estimator='mean')
plt.title("Average Sales across Seasonality Levels")
plt.show()
```





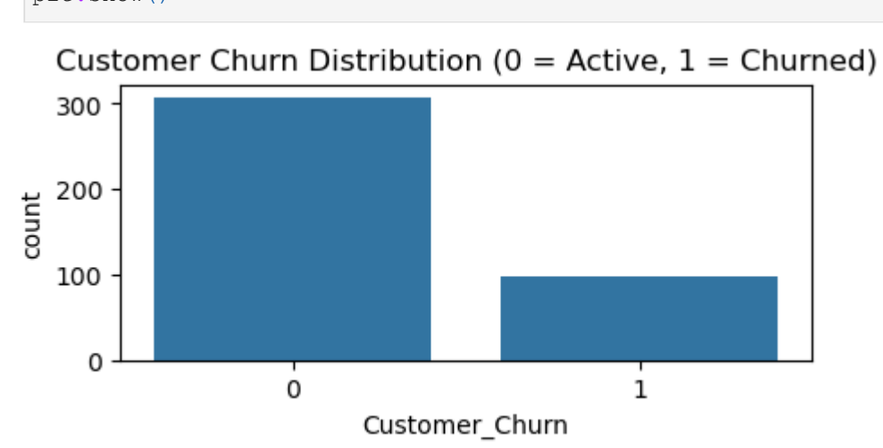
In [15]:

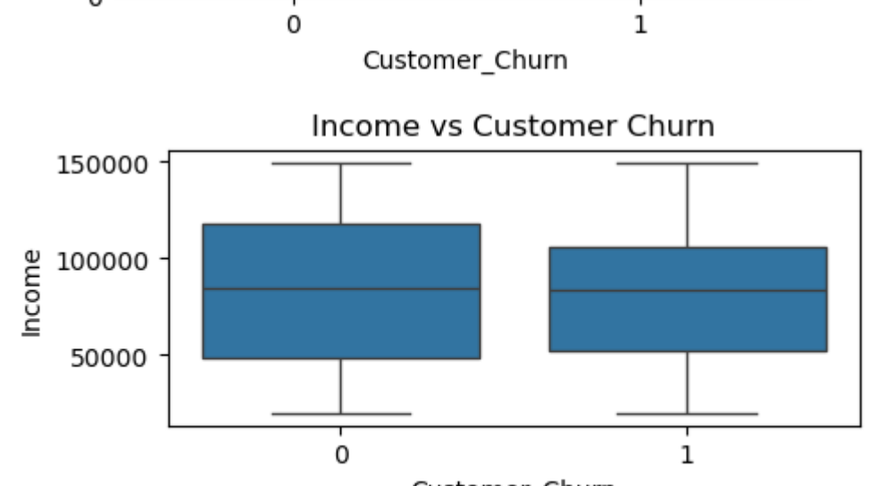
```
# Trend Analysis Over Time
df['seasonality_date'] = pd.to_datetime(df['seasonality_date'], dayfirst=True)

monthly_sales = df.groupby(df['seasonality_date'].dt.to_period("M"))['Sales'].sum()
monthly_sales.plot(figsize=(6,3), marker='o', title='Monthly Sales Trend')
plt.xlabel('Month')
plt.ylabel('Total Sales')
plt.show()
```

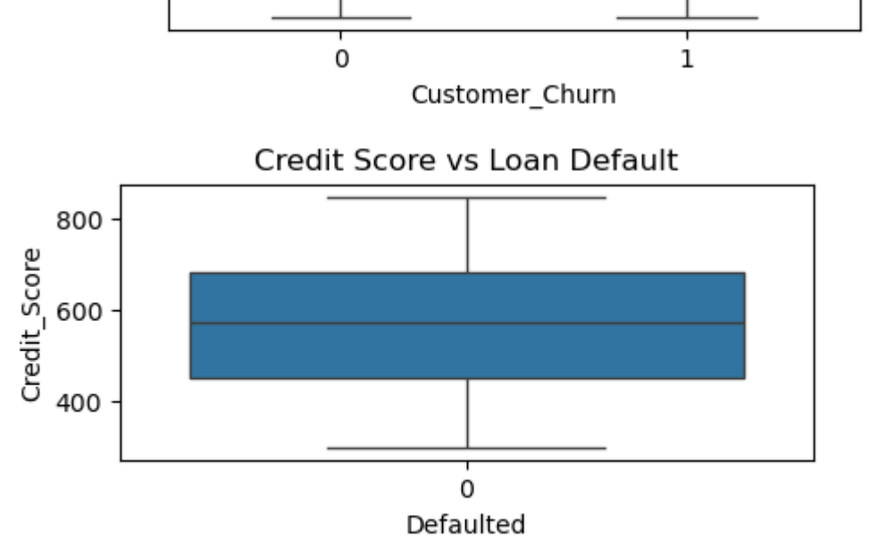


In [27]:

```
# Customer Churn Analysis
plt.figure(figsize=(5,2))
sns.countplot(x='Customer_Churn', data=df)
plt.title("Customer Churn Distribution (0 = Active, 1 = Churned)")
plt.show()

plt.figure(figsize=(5,2))
sns.boxplot(x='Customer_Churn', y='Income', data=df)
plt.title("Income vs Customer Churn")
plt.show()

plt.figure(figsize=(5,2))
sns.boxplot(x='Defaulted', y='Credit_Score', data=df)
plt.title("Credit Score vs Loan Default")
plt.show()
```

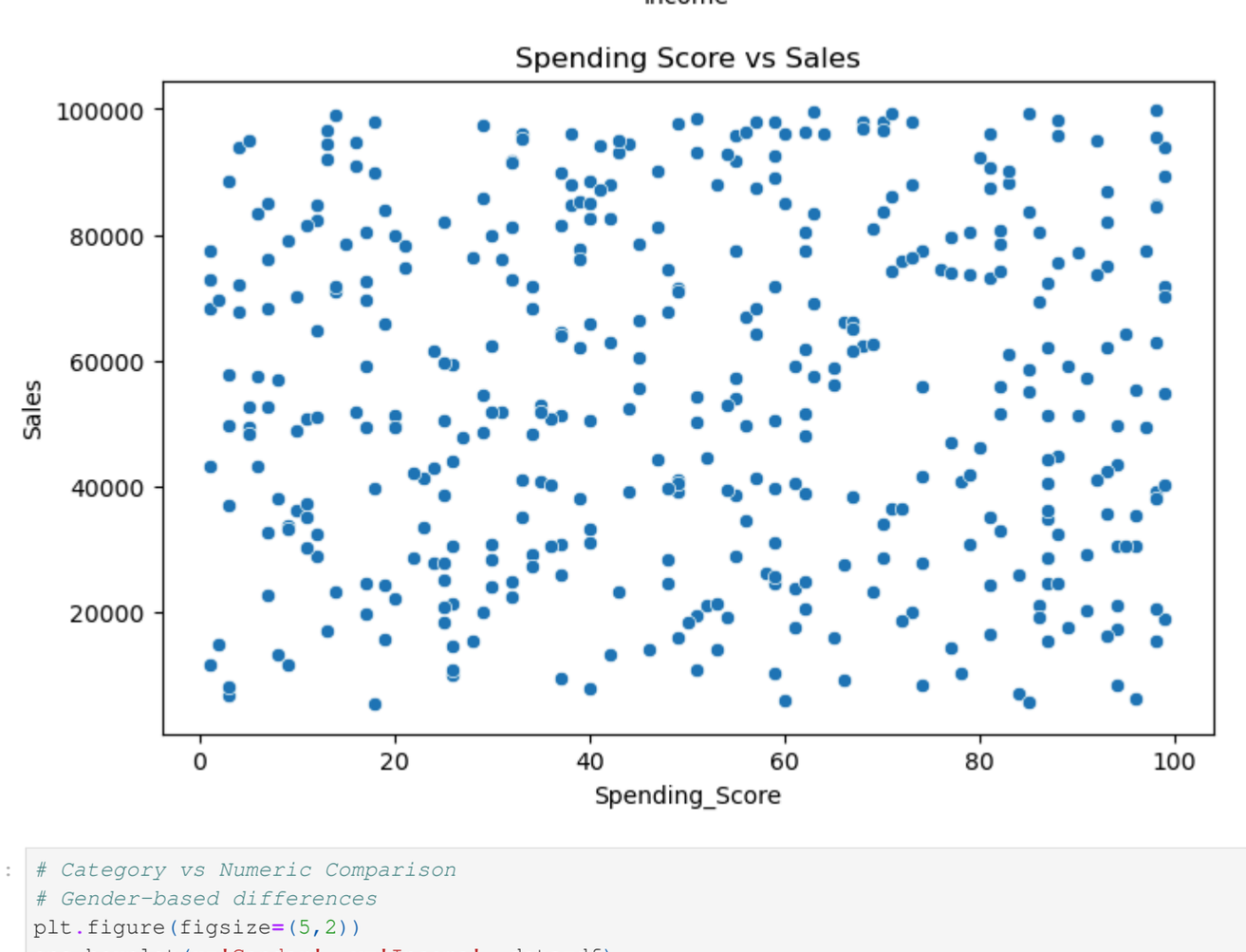






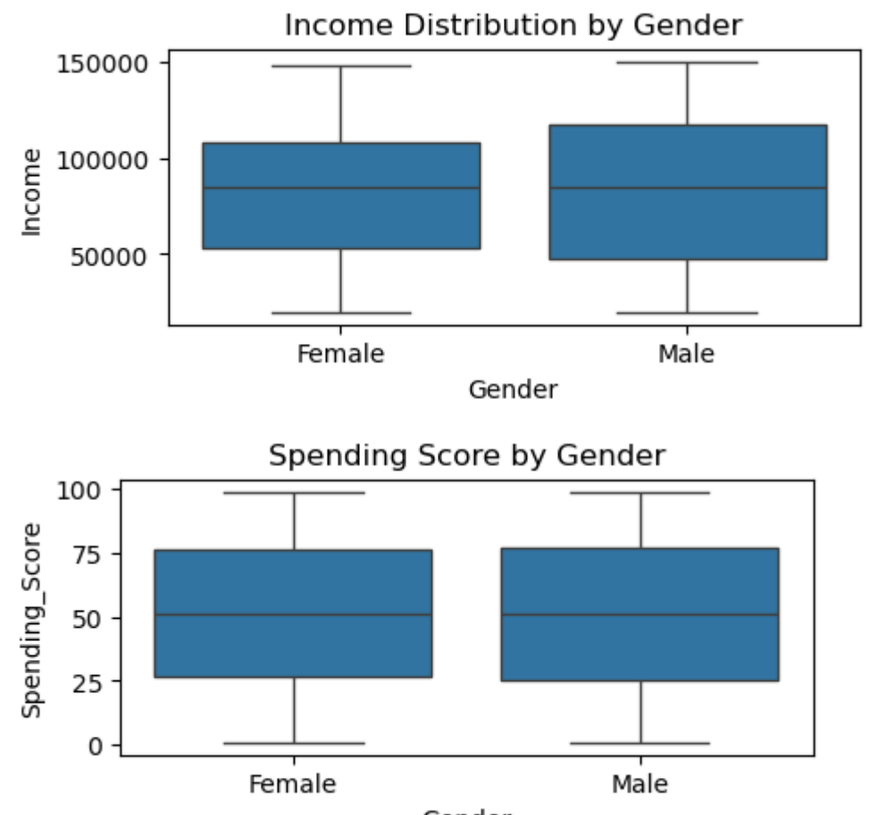
In [20]:

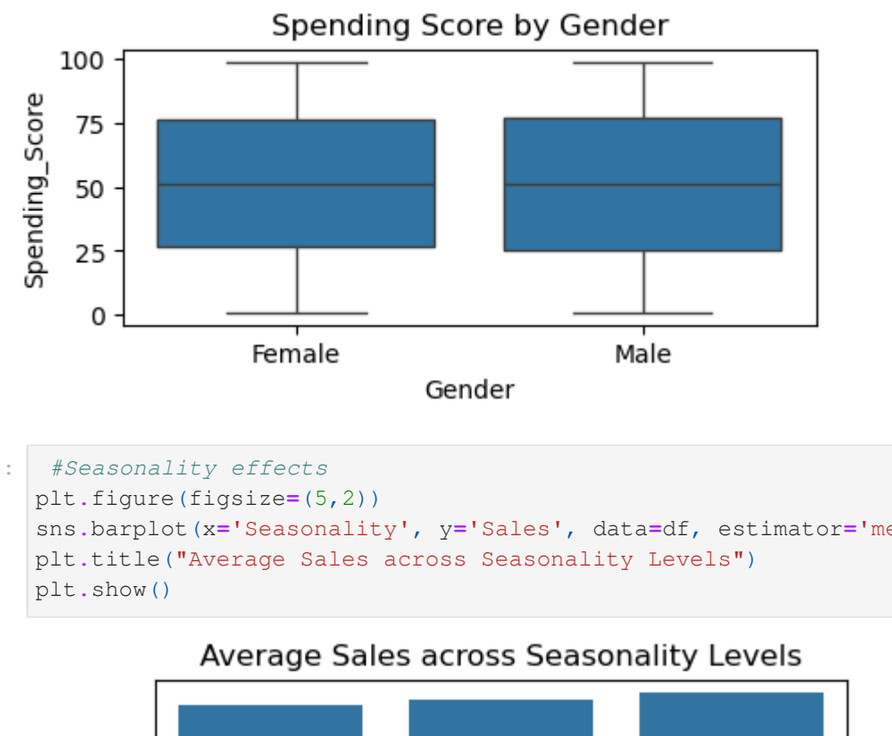
```
# scatter plots for strongest correlations
plt.figure(figsize=(8,5))
sns.scatterplot(x='Income', y='Sales', data=df)
plt.title("Income vs Sales")
plt.show()

plt.figure(figsize=(8,5))
sns.scatterplot(x='Spending_Score', y='Sales', data=df)
plt.title("Spending Score vs Sales")
plt.show()
```



In [22]:

```
# Category vs Numeric Comparison
# Gender-based differences
plt.figure(figsize=(5,2))
sns.boxplot(x='Gender', y='Income', data=df)
plt.title("Income Distribution by Gender")
plt.show()

plt.figure(figsize=(5,2))
sns.boxplot(x='Gender', y='Spending_Score', data=df)
plt.title("Spending Score by Gender")
plt.show()
```





In [24]:

```
 #Seasonality effects
plt.figure(figsize=(5,2))
sns.barplot(x='Seasonality', y='Sales', data=df, estimator='mean', errorbar=None)
plt.title("Average Sales across Seasonality Levels")
plt.show()
```

In [7]:

```
# Import libraries
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score

# Select features and target
X = df[['Income', 'Marketing_Spend']]
y = df['Sales']

# Split data into training and testing
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Create and train the model
lin_model = LinearRegression()
lin_model.fit(X_train, y_train)

# Predict
y_pred = lin_model.predict(X_test)

# Evaluate model
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print("Linear Regression - Sales Prediction")
print("Mean Squared Error:", mse)
print("R² Score:", r2)
```

```
Linear Regression - Sales Prediction
Mean Squared Error: 759564757.0598772
R² Score: -0.03374006282217201
```

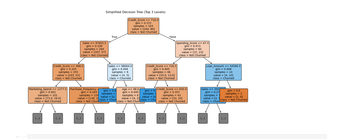
In [22]:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
X = df[['Sales', 'Purchase_Frequency', 'Marketing_Spend','Income']]
y = df['Customer_Churn']  # 1 = Yes, 0 = No
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
model = LogisticRegression(class_weight='balanced')
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
print("Accuracy:", accuracy_score(y_test, y_pred))
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

```
Accuracy: 0.5555555555555556
[[35 28]
 [ 8 10]]
              precision    recall  f1-score   support

           0       0.81      0.56      0.66        63
           1       0.26      0.56      0.36        18

    accuracy                           0.56        81
   macro avg       0.54      0.56      0.51        81
weighted avg       0.69      0.56      0.59        81
```

In [24]:

```
plt.figure(figsize=(15, 8))
plot_tree(model,
          feature_names=X.columns,
          class_names=['Not Churned', 'Churned'],
          filled=True,
          rounded=True,
          fontsize=10,
          max_depth=3)
plt.title("Simplified Decision Tree (Top 3 Levels)")
plt.show()
```


Simplified Decision Tree (Top 3 Levels)
True
False