data_analysis_on_saledata

August 9, 2023

[1]:
```
# Import necessary libraries
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

# Load your dataset (replace 'your_dataset.csv' with the actual file path)
df = pd.read_excel('SALESDATA.xlsx')
```

[2]:
```
# Display column information
print("Columns present in the dataset:")
print(df.columns)
```

```
Columns present in the dataset:
Index(['CustKey', 'DateKey', 'Discount Amount', 'Invoice Date',
       'Invoice Number', 'Item Class', 'Item Number', 'Item', 'Line Number',
       'List Price', 'Order Number', 'Promised Delivery Date', 'Sales Amount',
       'Sales Amount Based on List Price', 'Sales Cost Amount',
       'Sales Margin Amount', 'Sales Price', 'Sales Quantity', 'Sales Rep',
       'U/M', '@dropdown', 'Unnamed: 21'],
      dtype='object')
```

[3]:
```
# Understand data types of each column
print("\nData types of each column:")
print(df.dtypes)
```

```
Data types of each column:
CustKey                             int64
DateKey                    datetime64[ns]
Discount Amount                   float64
Invoice Date               datetime64[ns]
Invoice Number                      int64
Item Class                         object
Item Number                        object
Item                               object
Line Number                         int64
List Price                        float64
Order Number                        int64
Promised Delivery Date     datetime64[ns]
```

```
Sales Amount                              float64
Sales Amount Based on List Price          float64
Sales Cost Amount                         float64
Sales Margin Amount                       float64
Sales Price                               float64
Sales Quantity                              int64
Sales Rep                                   int64
U/M                                        object
@dropdown                                 float64
Unnamed: 21                                object
dtype: object
```

[4]:
```
# Identify missing values in relevant columns
columns_to_check = ['DateKey', 'Sales Amount', 'Item Class', 'CustKey', 'Sales␣
 ↪Quantity']  # Replace with relevant columns
missing_values = df[columns_to_check].isnull().sum()
print("\nMissing values in relevant columns:")
print(missing_values)
```

```
Missing values in relevant columns:
DateKey              0
Sales Amount         0
Item Class        8289
CustKey              0
Sales Quantity       0
dtype: int64
```

[5]:
```
# Check data quality for potential issues
print("\nSummary statistics for numeric columns:")
print(df.describe())
```

```
Summary statistics for numeric columns:
            CustKey                      DateKey  Discount Amount  \
count  6.528200e+04                        65282     65280.000000
mean   1.001770e+07  2018-06-11 12:49:02.115744      1855.574835
min    1.000045e+07          2017-01-01 00:00:00   -255820.800000
25%    1.001272e+07          2017-07-24 00:00:00       246.037500
50%    1.001966e+07          2018-01-29 00:00:00       441.760000
75%    1.002351e+07          2019-06-17 00:00:00       999.760000
max    1.002758e+07          2019-12-31 00:00:00    343532.660000
std    7.176148e+03                          NaN      9037.140888

                     Invoice Date  Invoice Number    Line Number  \
count                       65282    65282.000000   65282.000000
mean   2018-06-11 12:49:02.115744   216223.662020   23713.849790
min           2017-01-01 00:00:00   100012.000000    1000.000000
25%           2017-07-24 00:00:00   117931.000000    3000.000000
```

```
50%           2018-01-29 00:00:00   222869.500000   12000.000000
75%           2019-06-17 00:00:00   314318.750000   32000.000000
max           2019-12-31 00:00:00   332842.000000  344000.000000
std                           NaN    94992.281866   32664.024053

         List Price   Order Number         Promised Delivery Date  \
count  65282.000000   65282.000000                          65282
mean     514.693380  180583.064352  2018-06-08 20:37:36.989675776
min        0.000000  100838.000000            2008-12-15 00:00:00
25%      181.560000  115321.000000            2017-07-23 00:00:00
50%      325.190000  203702.000000            2018-01-29 00:00:00
75%      803.860000  218576.000000            2019-06-17 00:00:00
max     2760.700000  321532.000000            2019-12-31 00:00:00
std      449.189182   67593.871116                            NaN

        Sales Amount  Sales Amount Based on List Price  Sales Cost Amount  \
count   65282.000000                      65282.000000       65282.000000
mean     2852.038373                       4707.473613        1660.979228
min       200.010000                          0.000000           0.000000
25%       308.387500                        561.040000         167.790000
50%       553.940000                        998.160000         304.500000
75%      1280.042500                       2315.040000         687.320000
max    555376.000000                     632610.160000      366576.000000
std     15164.342107                       20696.443785        9556.485250

       Sales Margin Amount   Sales Price  Sales Quantity     Sales Rep  \
count         65282.000000  65281.000000    65282.000000  65282.000000
mean           1191.059145    283.615913       45.084311    137.422398
min           -3932.930000  -5000.000000       -1.000000    103.000000
25%             129.950000    100.070000        2.000000    113.000000
50%             246.490000    183.757500        3.000000    134.000000
75%             579.530000    448.220000        8.000000    160.000000
max          188800.000000   6035.000000    16000.000000    185.000000
std            5860.787502    252.876719      429.661793     26.643936

       @dropdown
count        0.0
mean         NaN
min          NaN
25%          NaN
50%          NaN
75%          NaN
max          NaN
std          NaN
```

[6]:
```
df = df.loc[:, ~df.columns.str.contains('^Unnamed')]
df.drop('@dropdown', axis=1, inplace=True)
```

[7]:
```
numeric_columns = df.select_dtypes(include=['float64', 'int64']).columns
columns_to_remove = ['CustKey', 'Invoice Number', 'Order Number', 'Line Number']
numeric_columns = [col for col in numeric_columns if col not in␣
 ↪columns_to_remove]
print(numeric_columns)
```

```
['Discount Amount', 'List Price', 'Sales Amount', 'Sales Amount Based on List
Price', 'Sales Cost Amount', 'Sales Margin Amount', 'Sales Price', 'Sales
Quantity', 'Sales Rep']
```

[8]:
```
# Function to convert column values to absolute in the original DataFrame
for i in numeric_columns:
    df[i] = df[i].abs()

# Display the modified DataFrame
print(df)
```

```
        CustKey    DateKey  Discount Amount Invoice Date  Invoice Number  \
0      10000481 2017-04-30           237.91   2017-04-30          100012
1      10002220 2017-07-14           368.79   2017-07-14          100233
2      10002220 2017-10-17           109.73   2017-10-17          116165
3      10002489 2017-06-03           211.75   2017-06-03          100096
4      10004516 2017-05-27         96627.94   2017-05-27          103341
…         …        …                …           …                …
65277  10017638 2018-03-21           505.78   2018-03-21          226497
65278  10017638 2018-03-21           410.75   2018-03-21          226497
65279  10017638 2018-03-21           876.16   2018-03-21          226497
65280  10017638 2018-03-21         24226.77   2018-03-21          226498
65281  10017638 2018-03-21         24479.26   2018-03-21          226498

      Item Class Item Number                            Item  Line Number  \
0            NaN         NaN                Urban Large Eggs         2000
1            P01       20910              Moms Sliced Turkey         1000
2            P01       38076  Cutting Edge Foot-Long Hot Dogs         1000
3            NaN         NaN                         Kiwi Lox         1000
4            P01       60776           High Top Sweet Onion         1000
…          …          …                                …          …
65277        P01       13447              High Top Oranges         8000
65278        P01       25906          Landslide White Sugar        38000
65279        P01       61856              Moms Potato Salad       227001
65280        P01       17801   Better Fancy Canned Sardines         1000
65281        P01       27550              Imagine Popsicles         4000

       List Price  Order Number Promised Delivery Date  Sales Amount  \
0            0.00        200015             2017-04-30        237.91
1          824.96        200245             2017-07-14        456.17
2          548.66        213157             2017-10-16        438.93
3            0.00        200107             2017-06-03        211.75
```

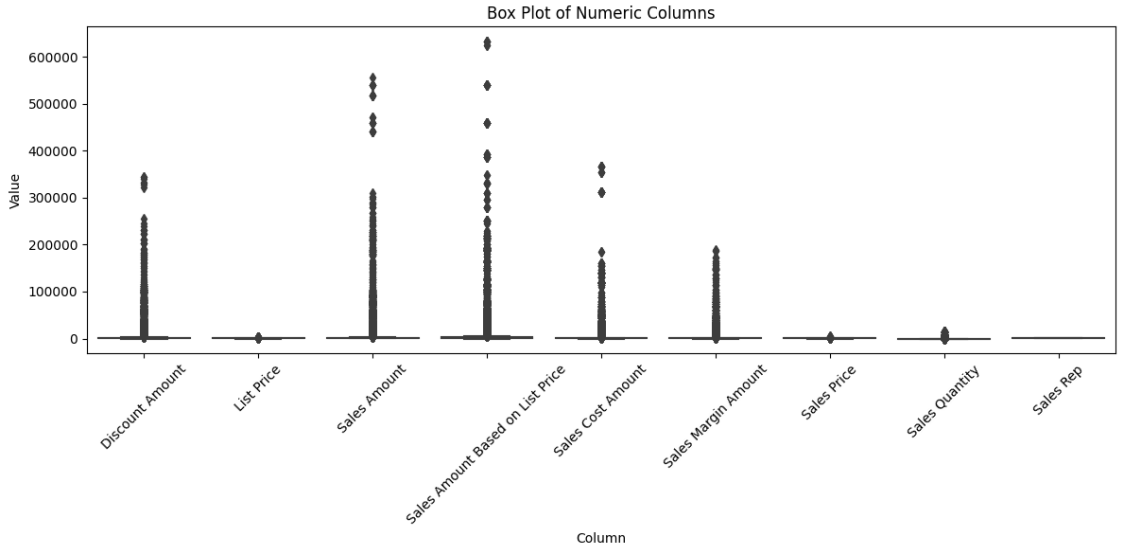
```
4          408.52        203785             2017-05-28      89248.66
…            …             …                  …              …
65277      119.52        320895             2018-03-21        569.90
65278      436.78        320895             2018-03-21        462.81
65279      232.92        320895             2018-03-21        987.20
65280     1431.23        320907             2018-03-21      27297.51
65281     1084.61        320907             2018-03-21      27582.02

       Sales Amount Based on List Price  Sales Cost Amount  \
0                                  0.00               0.00
1                                824.96               0.00
2                                548.66               0.00
3                                  0.00               0.00
4                             185876.60               0.00
…                                   …                  …
65277                          1075.68             239.95
65278                           873.56             423.55
65279                          1863.36             574.00
65280                         51524.28           16188.90
65281                         52061.28           14234.22

       Sales Margin Amount  Sales Price  Sales Quantity  Sales Rep U/M
0                   237.91   237.910000               1        184  EA
1                   456.17   456.170000               1        127  EA
2                   438.93   438.930000               1        127  EA
3                   211.75   211.750000               1        160  EA
4                 89248.66   196.150901             455        124  SE
…                      …          …                 …          …   ..
65277               329.95    63.322222               9        180  EA
65278                39.26   231.405000               2        180  EA
65279               413.20   123.400000               8        180  EA
65280             11108.61   758.264167              36        180  EA
65281             13347.80   574.625417              48        180  EA

[65282 rows x 20 columns]
```

[9]:
```python
# Function to check for negative values in a DataFrame
def has_negative_values(df):
    numeric_columns = df.select_dtypes(include=['float64', 'int64']).columns
    for col in numeric_columns:
        if (df[col] < 0).any():
            return True
    return False

# Call the function to check for negative values
if has_negative_values(df):
    print("The DataFrame has negative values.")
```

```
else:
    print("The DataFrame does not have negative values.")
```

```
The DataFrame does not have negative values.
```

[10]:
```
plt.figure(figsize=(12, 6))
sns.boxplot(data=df[numeric_columns])
plt.title('Box Plot of Numeric Columns')
plt.xlabel('Column')
plt.ylabel('Value')
plt.xticks(rotation=45)
plt.tight_layout()

plt.show()
```



[11]:
```
# Check data quality for potential issues
print("\nSummary statistics for numeric columns:")
print(df.shape)
print(df.describe())
```

```
Summary statistics for numeric columns:
(65282, 20)
            CustKey                      DateKey  Discount Amount  \
count  6.528200e+04                        65282     65280.000000
mean   1.001770e+07  2018-06-11 12:49:02.115744      1976.905870
min    1.000045e+07         2017-01-01 00:00:00         0.000000
25%    1.001272e+07         2017-07-24 00:00:00       247.710000
50%    1.001966e+07         2018-01-29 00:00:00       446.370000
```

```
75%    1.002351e+07         2019-06-17 00:00:00      1012.410000
max    1.002758e+07         2019-12-31 00:00:00    343532.660000
std    7.176148e+03                         NaN      9011.376669

                    Invoice Date  Invoice Number    Line Number  \
count                      65282    65282.000000   65282.000000
mean   2018-06-11 12:49:02.115744   216223.662020   23713.849790
min          2017-01-01 00:00:00   100012.000000    1000.000000
25%          2017-07-24 00:00:00   117931.000000    3000.000000
50%          2018-01-29 00:00:00   222869.500000   12000.000000
75%          2019-06-17 00:00:00   314318.750000   32000.000000
max          2019-12-31 00:00:00   332842.000000  344000.000000
std                          NaN    94992.281866   32664.024053

         List Price   Order Number         Promised Delivery Date  \
count  65282.000000   65282.000000                          65282
mean     514.693380  180583.064352  2018-06-08 20:37:36.989675776
min        0.000000  100838.000000            2008-12-15 00:00:00
25%      181.560000  115321.000000            2017-07-23 00:00:00
50%      325.190000  203702.000000            2018-01-29 00:00:00
75%      803.860000  218576.000000            2019-06-17 00:00:00
max     2760.700000  321532.000000            2019-12-31 00:00:00
std      449.189182   67593.871116                            NaN

        Sales Amount  Sales Amount Based on List Price  Sales Cost Amount  \
count   65282.000000                      65282.000000       65282.000000
mean     2852.038373                       4707.473613        1660.979228
min       200.010000                          0.000000           0.000000
25%       308.387500                        561.040000         167.790000
50%       553.940000                        998.160000         304.500000
75%      1280.042500                       2315.040000         687.320000
max    555376.000000                     632610.160000      366576.000000
std     15164.342107                      20696.443785        9556.485250

       Sales Margin Amount   Sales Price  Sales Quantity     Sales Rep
count         65282.000000  65281.000000    65282.000000  65282.000000
mean           1195.121974    283.769097       45.084342    137.422398
min               0.000000      0.337341        0.000000    103.000000
25%             130.640000    100.070000        2.000000    113.000000
50%             247.030000    183.759000        3.000000    134.000000
75%             581.690000    448.220000        8.000000    160.000000
max          188800.000000   6035.000000    16000.000000    185.000000
std            5859.960354    252.704807      429.661790     26.643936
```

[12]:
```
from sklearn.ensemble import IsolationForest

# Drop rows with missing values from the numeric columns
```

```
df_cleaned = df.dropna()

# Create an instance of IsolationForest
isolation_forest = IsolationForest(contamination=0.1, random_state=42)

# Fit the model to the cleaned data
outlier_scores = isolation_forest.fit_predict(df_cleaned[numeric_columns])

# Add the outlier scores to your cleaned DataFrame using .loc indexer
df_cleaned.loc[:, 'IsolationForestScore'] = outlier_scores
```

```
C:\Users\shaik\AppData\Local\Temp\ipykernel_23140\1158634427.py:13:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df_cleaned.loc[:, 'IsolationForestScore'] = outlier_scores
```

[13]:
```
print(df_cleaned.shape)
```

```
(56993, 21)
```

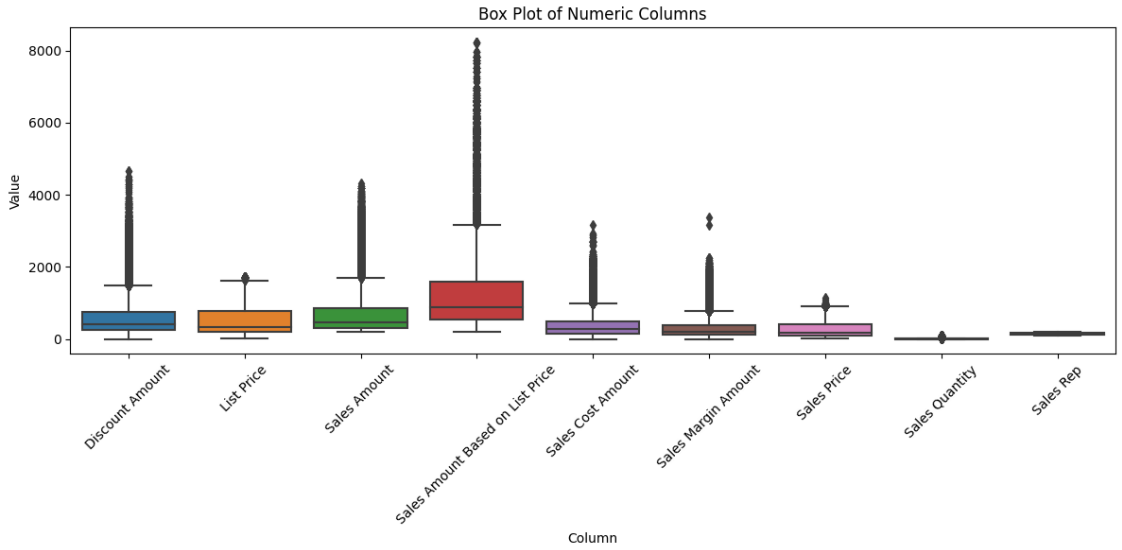
[14]:
```
# Specify the threshold for outlier scores
outlier_threshold = -0.5  # Adjust this threshold as needed

# Create a mask for rows with outlier scores greater than the threshold
outlier_rows_mask = outlier_scores < outlier_threshold

# Drop the rows from the DataFrame
df_cleaned = df_cleaned[~outlier_rows_mask]
```

[15]:
```
# Display the cleaned DataFrame
print(df_cleaned.shape)
print(df_cleaned.describe())
```

```
(51293, 21)
            CustKey                        DateKey  Discount Amount  \
count  5.129300e+04                          51293     51293.000000
mean   1.001763e+07  2018-06-12 05:57:12.819293184       630.509213
min    1.000045e+07            2017-01-01 00:00:00         0.000000
25%    1.001272e+07            2017-07-22 00:00:00       247.150000
50%    1.001940e+07            2018-01-29 00:00:00       405.530000
75%    1.002351e+07            2019-06-16 00:00:00       739.040000
max    1.002758e+07            2019-12-31 00:00:00      4655.700000
std    7.218472e+03                            NaN       614.781839

                      Invoice Date  Invoice Number    Line Number  \
```

```
count                          51293    51293.000000   51293.000000
mean   2018-06-12 05:57:12.819293184   216505.671378   25130.700661
min              2017-01-01 00:00:00   100034.000000    1000.000000
25%              2017-07-22 00:00:00   117976.000000    4000.000000
50%              2018-01-29 00:00:00   222930.000000   14000.000000
75%              2019-06-16 00:00:00   314301.000000   33000.000000
max              2019-12-31 00:00:00   332842.000000  330000.000000
std                              NaN    95045.092925   32378.415985

         List Price   Order Number         Promised Delivery Date  \
count  51293.000000   51293.000000                          51293
mean     501.243895  180264.273390  2018-06-09 18:16:03.921002752
min        4.500000  100838.000000            2008-12-15 00:00:00
25%      189.270000  115081.000000            2017-07-21 00:00:00
50%      321.640000  203645.000000            2018-01-29 00:00:00
75%      767.750000  218576.000000            2019-06-16 00:00:00
max     1731.400000  321532.000000            2019-12-31 00:00:00
std      417.125426   67519.191516                            NaN

       Sales Amount  Sales Amount Based on List Price  Sales Cost Amount  \
count  51293.000000                      51293.000000       51293.000000
mean     723.922632                       1354.383481         401.389275
min      200.010000                        195.610000           0.000000
25%      290.340000                        548.660000         155.860000
50%      472.770000                        879.400000         269.340000
75%      855.120000                       1600.920000         483.350000
max     4313.350000                       8229.900000        3170.890000
std      668.299130                       1268.509449         388.371886

       Sales Margin Amount   Sales Price  Sales Quantity     Sales Rep  \
count         51293.000000  51293.000000    51293.000000  51293.000000
mean            323.029010    271.698564        5.132104    136.491490
min               0.000000      2.391700        1.000000    103.000000
25%             122.170000    102.129583        1.000000    113.000000
50%             209.380000    176.000000        3.000000    130.000000
75%             381.900000    420.110000        5.000000    160.000000
max            3384.010000   1134.770000      100.000000    185.000000
std             322.304839    227.875952        8.195166     26.371995

       IsolationForestScore
count               51293.0
mean                    1.0
min                     1.0
25%                     1.0
50%                     1.0
75%                     1.0
max                     1.0
std                     0.0
```

[16]:
```
plt.figure(figsize=(12, 6))
sns.boxplot(data=df_cleaned[numeric_columns])
plt.title('Box Plot of Numeric Columns')
plt.xlabel('Column')
plt.ylabel('Value')
plt.xticks(rotation=45)
plt.tight_layout()

plt.show()
```



[17]:
```
df_cleaned.drop('Invoice Date', axis=1, inplace=True)
df_cleaned.drop('Promised Delivery Date', axis=1, inplace=True)
df_cleaned.drop('Item Class', axis=1, inplace=True)
df_cleaned.drop('Item Number', axis=1, inplace=True)
df_cleaned.drop('CustKey', axis=1, inplace=True)
df_cleaned.drop('Invoice Number', axis=1, inplace=True)
df_cleaned.drop('Line Number', axis=1, inplace=True)
df_cleaned.drop('Order Number', axis=1, inplace=True)
```

[18]:
```
print(df_cleaned.isnull().sum())
```

```
DateKey                          0
Discount Amount                  0
Item                             0
List Price                       0
Sales Amount                     0
Sales Amount Based on List Price 0
Sales Cost Amount                0
Sales Margin Amount              0
```

```
Sales Price                    0
Sales Quantity                 0
Sales Rep                      0
U/M                            0
IsolationForestScore           0
dtype: int64
```

[19]:
```python
# Convert 'DateKey' column to datetime type
df_cleaned['DateKey'] = pd.to_datetime(df['DateKey'])

# Create a continuous date range
date_range = pd.date_range(start='2017-01-01', end='2019-12-31', freq='D')

# Find the missing dates
missing_dates = date_range[~date_range.isin(df_cleaned['DateKey'])]

# Convert missing dates to a list
missing_dates_list = missing_dates.tolist()

print(len(missing_dates_list))
```

```
537
```

[20]:
```python
# Calculate the mean of numeric columns
numeric_means = df_cleaned.select_dtypes(include=['float64', 'int64']).mean()

# Create a dictionary with column names as keys and mean values as values
numeric_means_dict = numeric_means.to_dict()

print(numeric_means_dict)
```

```
{'Discount Amount': 630.5092131772367, 'List Price': 501.24389511239343, 'Sales
Amount': 723.9226317431228, 'Sales Amount Based on List Price':
1354.3834812060124, 'Sales Cost Amount': 401.38927494979816, 'Sales Margin
Amount': 323.0290103912815, 'Sales Price': 271.69856357567943, 'Sales Quantity':
5.132103795839588, 'Sales Rep': 136.4914900668707}
```

[21]:
```python
# Specify the columns to keep
columns_to_keep = ['Item', 'U/M']

# Count the top 5 elements for each column to keep
top_elements_dict = {}

for column in columns_to_keep:
    top_elements = df[column].value_counts().index[:21].tolist()
    top_elements_dict[column] = top_elements

print(top_elements_dict)
```

```
{'Item': ['Better Fancy Canned Sardines', 'Ebony Prepared Salad', 'Moms Sliced
Turkey', 'Imagine Popsicles', 'Discover Manicotti', 'Red Spade Foot-Long Hot
Dogs', 'High Top Dried Mushrooms', 'Big Time Frozen Cheese Pizza', 'Cutting Edge
Foot-Long Hot Dogs', 'Bravo Large Canned Shrimp', 'Red Spade Low Fat Bologna',
'Bravo Canned Yams', 'Red Spade Turkey Hot Dogs', 'Atomic Mint Chocolate Bar',
'Ebony Squash', 'Thresher Spicy Mints', 'Fast Dried Apples', 'Nationeel Salted
Pretzels', 'Ebony Macintosh Apples', 'Moms Sliced Ham', 'Bravo Canned Tomatos'],
'U/M': ['EA', 'SE', 'PR']}
```

[22]:
```
df_cleaned.shape
```

[22]:
```
(51293, 13)
```

[23]:
```
new_rows = pd.DataFrame({'DateKey': missing_dates})

# Concatenate the original DataFrame and the new rows
df_new = pd.concat([df_cleaned, new_rows], ignore_index=True)
```

[24]:
```
print(df_new.shape)
print(df_new.isnull().sum())
```

```
(51830, 13)
DateKey                            0
Discount Amount                  537
Item                             537
List Price                       537
Sales Amount                     537
Sales Amount Based on List Price 537
Sales Cost Amount                537
Sales Margin Amount              537
Sales Price                      537
Sales Quantity                   537
Sales Rep                        537
U/M                              537
IsolationForestScore             537
dtype: int64
```

[25]:
```
# Fill missing values with column means
for column, mean in numeric_means_dict.items():
    df_new[column].fillna(mean, inplace=True)
```

[26]:
```
print(df_new.shape)
print(df_new.isnull().sum())
```

```
(51830, 13)
DateKey                            0
Discount Amount                    0
Item                             537
List Price                         0
```

```
Sales Amount                          0
Sales Amount Based on List Price      0
Sales Cost Amount                     0
Sales Margin Amount                   0
Sales Price                           0
Sales Quantity                        0
Sales Rep                             0
U/M                                 537
IsolationForestScore                537
dtype: int64
```

[27]:
```
df_new.drop('IsolationForestScore', axis=1, inplace=True)
```

[28]:
```
import random
# Iterate over rows
for index, row in df_new.iterrows():
    # Iterate over columns using dictionary keys
    for column in top_elements_dict.keys():
        if pd.isnull(row[column]):
            values = top_elements_dict[column]
            selected_value = random.choice(values)
            df_new.at[index, column] = selected_value
```

[29]:
```
print(df_new.shape)
print(df_new.isnull().sum())
```

```
(51830, 12)
DateKey                           0
Discount Amount                   0
Item                              0
List Price                        0
Sales Amount                      0
Sales Amount Based on List Price  0
Sales Cost Amount                 0
Sales Margin Amount               0
Sales Price                       0
Sales Quantity                    0
Sales Rep                         0
U/M                               0
dtype: int64
```

[32]:
```
df_new.columns
```

[32]:
```
Index(['DateKey', 'Discount Amount', 'Item', 'List Price', 'Sales Amount',
       'Sales Amount Based on List Price', 'Sales Cost Amount',
       'Sales Margin Amount', 'Sales Price', 'Sales Quantity', 'Sales Rep',
       'U/M'],
      dtype='object')
```

1 Detail Summary of Data Preprocessing and Cleaning Steps

The provided code snippet performs various data preprocessing and cleaning steps on the dataset 'SALESDATA.xlsx' to ensure the data's quality and suitability for analysis. Here's a breakdown of the key steps:

1. **Importing Libraries and Loading Data:**
   - The necessary libraries such as pandas, seaborn, and matplotlib are imported.
   - The 'SALESDATA.xlsx' dataset is loaded into a DataFrame named 'df'.
2. **Exploring Data Information:**
   - The columns present in the dataset are displayed using `df.columns`.
   - Data types of each column are displayed using `df.dtypes`.
3. **Identifying Missing Values:**
   - A list of relevant columns is specified to check for missing values.
   - The number of missing values in the specified columns is calculated and displayed.
4. **Data Quality Check and Cleaning:**
   - Irrelevant columns and unnamed columns are dropped from the DataFrame.
   - Numeric columns with potential negative values are identified and transformed to absolute values.
   - A function `has_negative_values` is defined to check for negative values in the DataFrame.
5. **Visualizing Data Distribution:**
   - A box plot is created using Seaborn to visualize the distribution of numeric columns.
6. **Outlier Detection with Isolation Forest:**
   - Rows with missing values are dropped from the DataFrame.
   - An Isolation Forest model is created to detect outliers in the cleaned data.
   - Outlier scores are assigned to rows in the DataFrame.
7. **Removing Outliers:**
   - A threshold for outlier scores is specified.
   - Rows with outlier scores greater than the threshold are dropped from the DataFrame.
8. **Further Data Cleaning:**
   - Additional columns are dropped from the DataFrame.
   - Remaining missing values are displayed using `df_cleaned.isnull().sum()`.
9. **Handling Missing Dates:**
   - The 'DateKey' column is converted to datetime type.
   - A continuous date range is created.
   - Missing dates are identified and converted to a list.
10. **Handling Missing Numeric Values:**
    - The mean of numeric columns is calculated.
    - Missing values in numeric columns are filled with their respective column means.
11. **Handling Missing Categorical Values:**
    - The top elements for specified columns are counted.
    - Missing categorical values are filled with randomly selected top elements.
12. **Final Data:**
    - New rows with missing dates are concatenated to the cleaned DataFrame.
    - Remaining missing values are filled using mean and random categorical values.

Overall, this detailed preprocessing and cleaning process ensures that the dataset is ready for further analysis and visualization by addressing missing values, outliers, and other data quality issues.

1.0.1 Here are 30 example questions that I will answer using the combinations of columns for data analysis:

1.0.2 Monthly Trends:

1. What is the monthly trend of Sales Amount?
2. How does Sales Quantity vary on a monthly basis?
3. Are there any noticeable patterns in Sales Margin Amount on a monthly scale?

1.0.3 Seasonal Trends:

4. What are the seasonal fluctuations in Sales Amount?
5. How do Sales Quantity changes with different seasons?
6. Is there a pattern in Sales Margin Amount across different seasons?

1.0.4 Yearly Trends:

7. How has Sales Amount evolved over the years?
8. Are there any significant changes in Sales Quantity on a yearly basis?
9. What is the trend in Sales Margin Amount over the years?

1.0.5 Month-Yearly Trends:

10. What are the monthly trends of Sales Amount over the years?
11. How does Sales Quantity change in specific months across different years?
12. Are there any specific months that stand out in terms of Sales Margin Amount?

1.0.6 Seasonal-Yearly Trends:

13. How do sales trends vary across different seasons for each year?
14. Are there any consistent patterns in Sales Quantity across different seasons and years?
15. What is the yearly trend in Sales Margin Amount for each season?

1.0.7 Item Analysis:

16. Which items have the highest Sales Amount overall?
17. Are there any items with varying Sales Quantity throughout the dataset?
18. How does the Sales Margin Amount vary between different items?

1.0.8 Sales Rep Analysis:

19. Who are the top sales representatives in terms of generating Sales Amount?
20. Are there differences in Sales Quantity achieved by different sales reps?
21. How does Sales Margin Amount differ among different sales representatives?

1.0.9 Price Analysis:

22. How does the List Price correlate with the Sales Amount for various items?
23. Are there instances where a higher List Price results in higher Sales Quantity?
24. What is the relationship between Sales Price and Sales Amount?

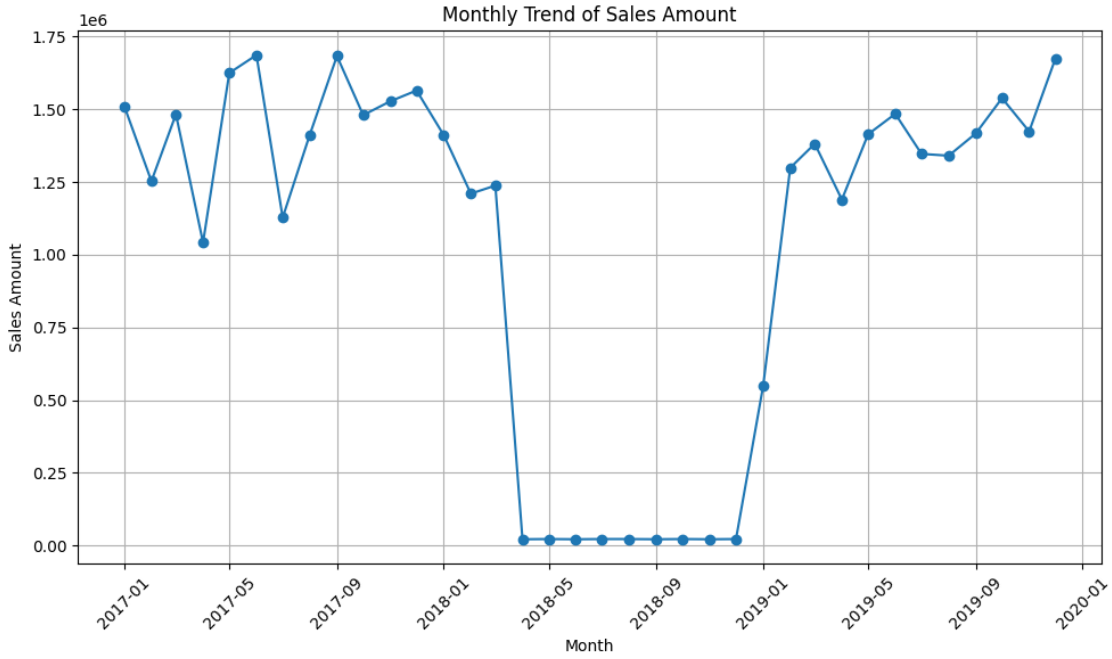
1.0.10 Combined Analysis:

25. How do Sales Amount and Sales Quantity correlate on a monthly basis?
26. Are there any seasonal trends in Sales Margin Amount that align with specific items?
27. What is the overall trend of Sales Amount when considering both the sales rep and the item?

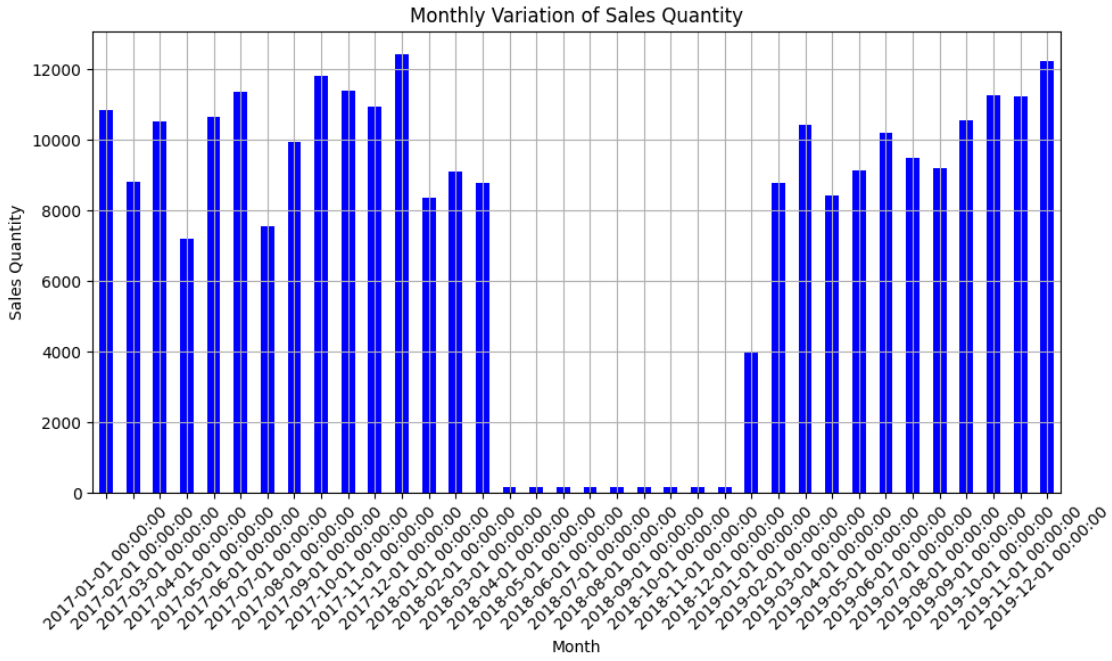
1.0.11 Comparison Analysis:

28. How does the Sales Amount of a specific item compare to the overall Sales Amount trend?
29. Are there any significant differences in Sales Quantity between different seasons for a particular item?
30. How does the Sales Margin Amount of a particular item compare across different sales representatives? em compare across different sales representatives?

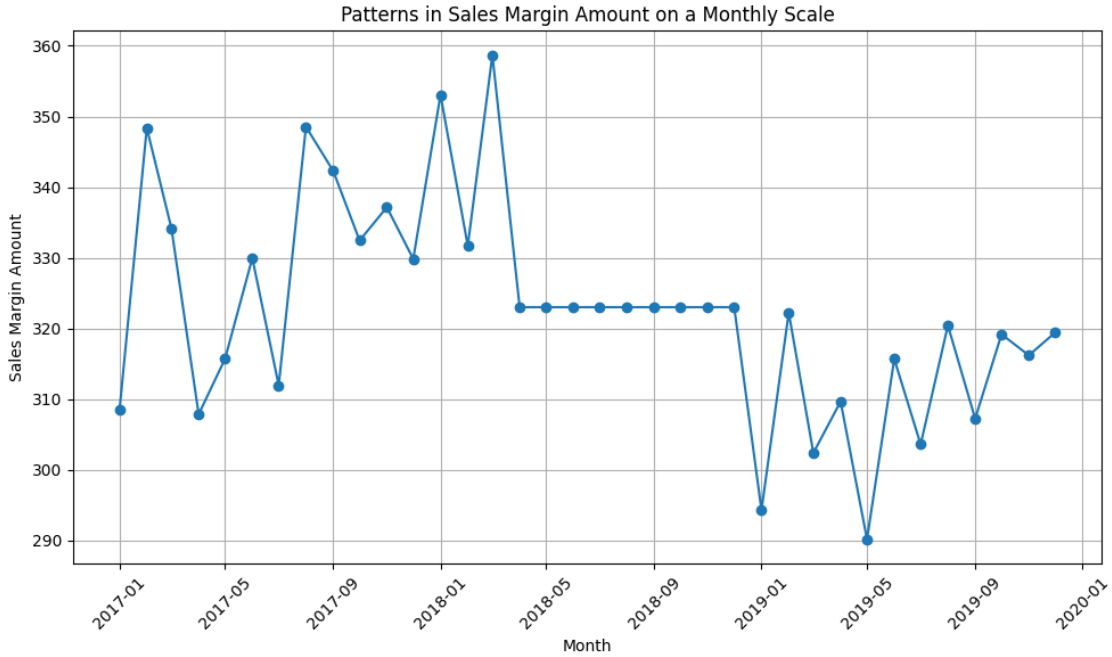
[33]:
```
# Convert the 'DateKey' column to datetime
df_new['DateKey'] = pd.to_datetime(df_new['DateKey'])

# Group data by month and calculate the sum of Sales Amount
monthly_sales = df_new.groupby(df_new['DateKey'].dt.to_period('M'))['Sales␣
 ↪Amount'].sum()

# Convert the index to datetime
monthly_sales.index = monthly_sales.index.to_timestamp()

# Create a line chart
plt.figure(figsize=(10, 6))
plt.plot(monthly_sales.index, monthly_sales.values, marker='o', linestyle='-')
plt.xlabel('Month')
plt.ylabel('Sales Amount')
plt.title('Monthly Trend of Sales Amount')
plt.xticks(rotation=45)
plt.grid(True)
plt.tight_layout()

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



[34]:
```
# Group data by month and calculate the sum of Sales Quantity
monthly_quantity = df_new.groupby(df_new['DateKey'].dt.to_period('M'))['Sales␣
↪Quantity'].sum()

# Convert the index to datetime
monthly_quantity.index = monthly_quantity.index.to_timestamp()

# Create a bar chart
plt.figure(figsize=(10, 6))
monthly_quantity.plot(kind='bar', color='blue')
plt.xlabel('Month')
plt.ylabel('Sales Quantity')
plt.title('Monthly Variation of Sales Quantity')
plt.xticks(rotation=45)
plt.grid(True)
plt.tight_layout()

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

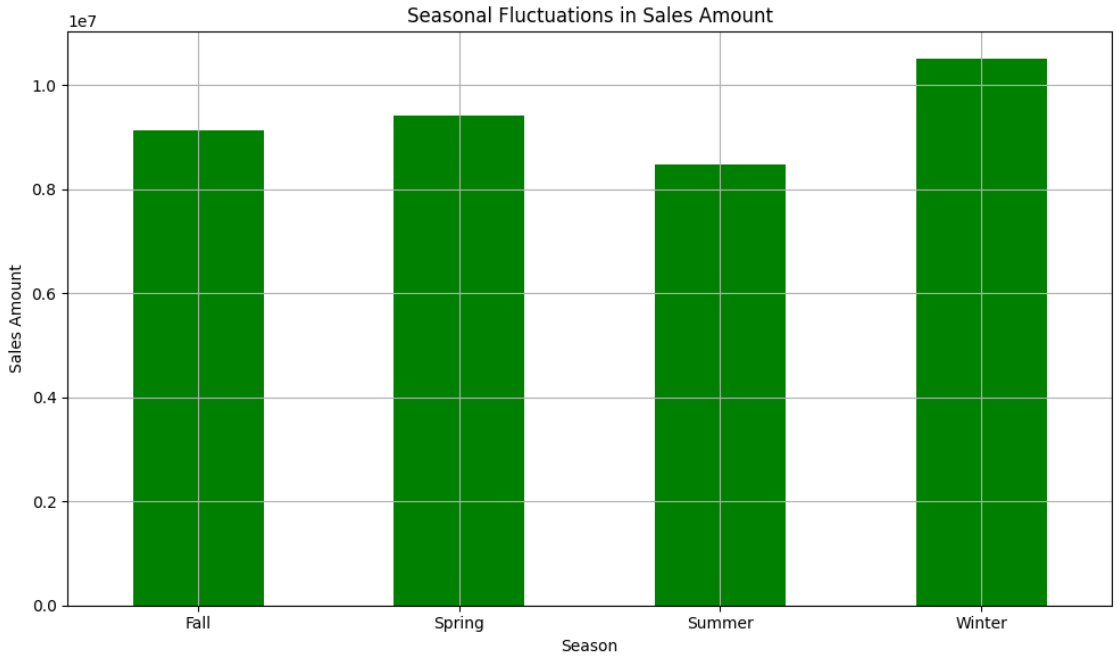


[35]:
```
# Group data by month and calculate the mean of Sales Margin Amount
monthly_margin = df_new.groupby(df_new['DateKey'].dt.to_period('M'))['Sales␣
 ↪Margin Amount'].mean()

# Convert the index to datetime
monthly_margin.index = monthly_margin.index.to_timestamp()

# Create a line chart
plt.figure(figsize=(10, 6))
plt.plot(monthly_margin.index, monthly_margin.values, marker='o', linestyle='-')
plt.xlabel('Month')
plt.ylabel('Sales Margin Amount')
plt.title('Patterns in Sales Margin Amount on a Monthly Scale')
plt.xticks(rotation=45)
plt.grid(True)
plt.tight_layout()

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

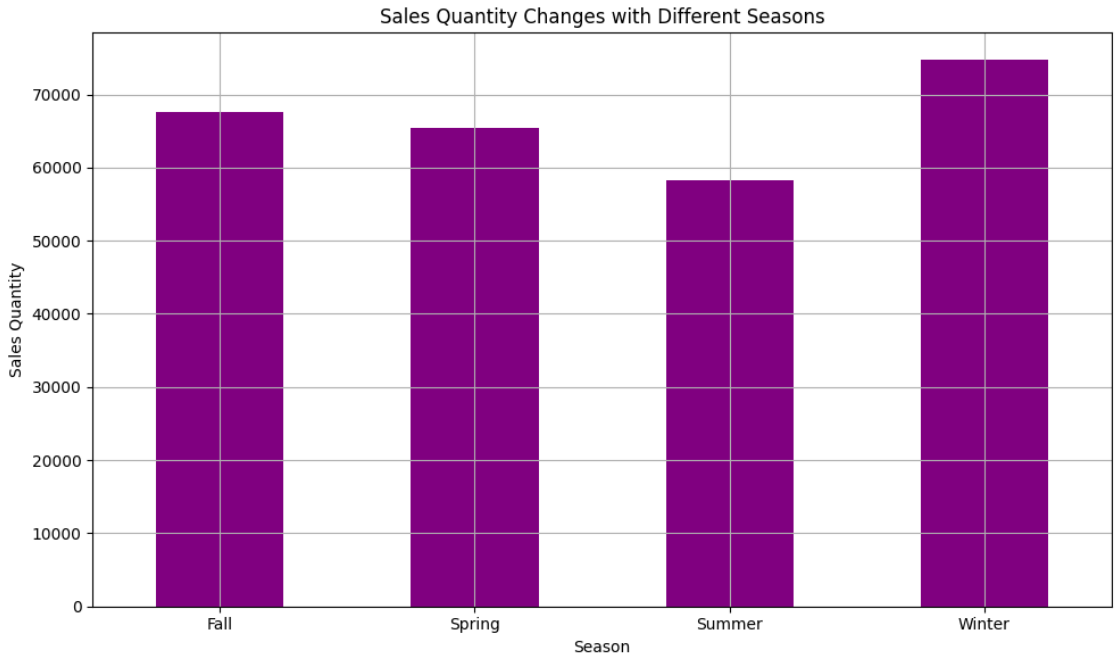


[36]:
```
# Define a function to map months to seasons
def get_season(month):
    if 3 <= month <= 5:
        return 'Spring'
    elif 6 <= month <= 8:
        return 'Summer'
    elif 9 <= month <= 11:
        return 'Fall'
    else:
        return 'Winter'

# Add a new column for the season
df_new['Season'] = df_new['DateKey'].dt.month.apply(get_season)

# Group data by season and calculate the sum of Sales Amount
seasonal_sales = df_new.groupby('Season')['Sales Amount'].sum()

# Create a bar chart
plt.figure(figsize=(10, 6))
seasonal_sales.plot(kind='bar', color='green')
plt.xlabel('Season')
plt.ylabel('Sales Amount')
plt.title('Seasonal Fluctuations in Sales Amount')
plt.xticks(rotation=0)
plt.grid(True)
```

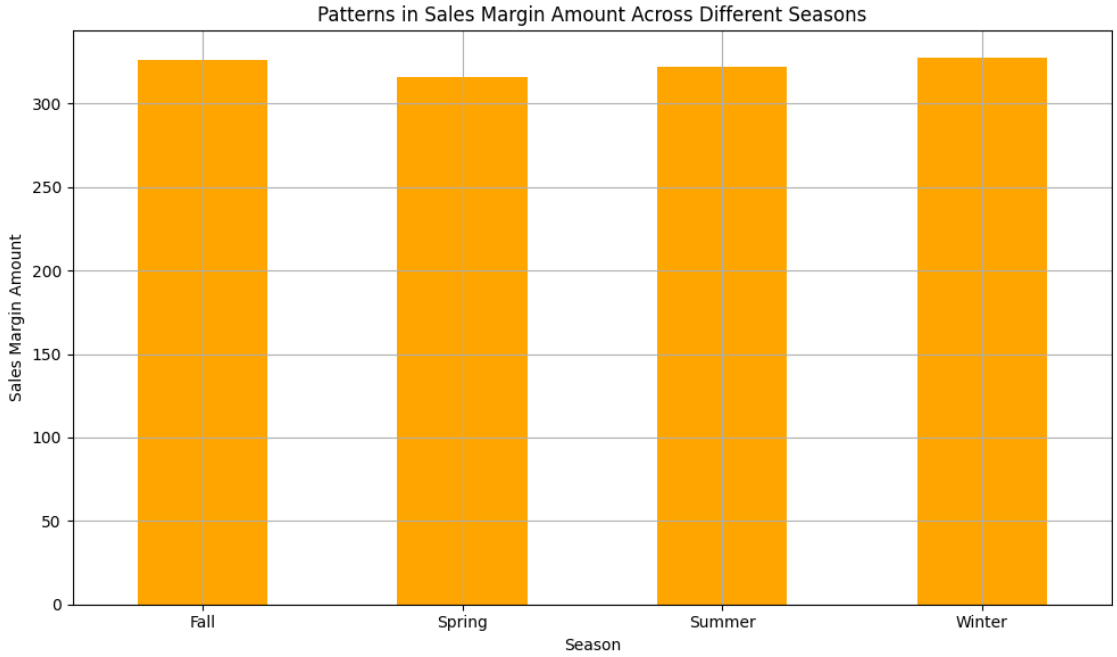
```
plt.tight_layout()

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



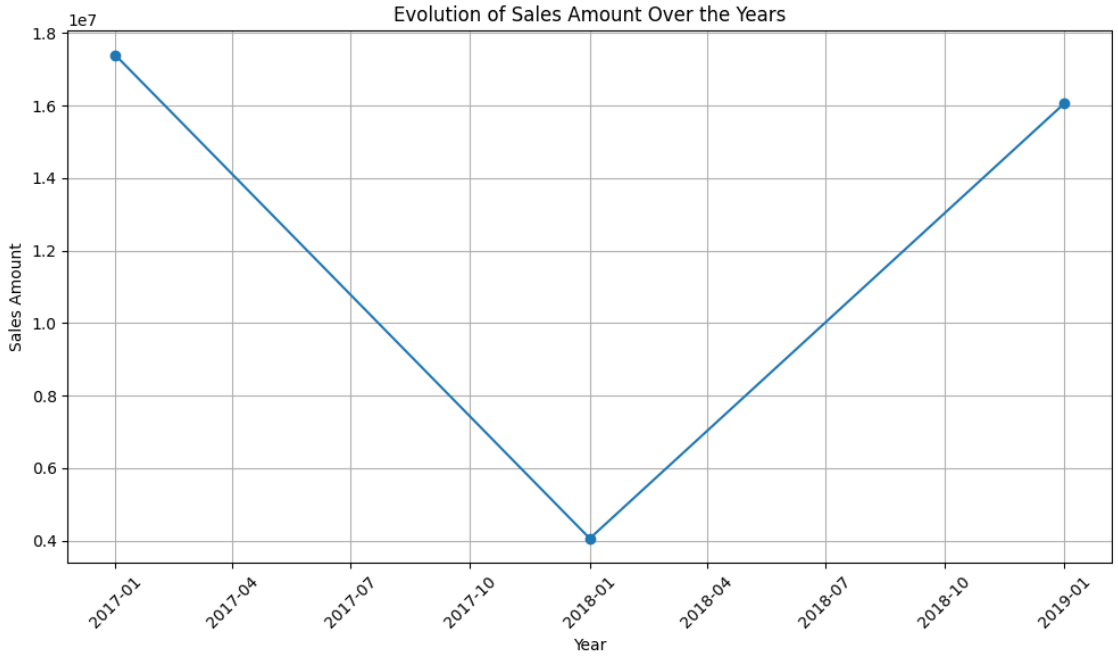
[37]:
```
# Group data by season and calculate the sum of Sales Quantity
seasonal_quantity = df_new.groupby('Season')['Sales Quantity'].sum()

# Create a bar chart
plt.figure(figsize=(10, 6))
seasonal_quantity.plot(kind='bar', color='purple')
plt.xlabel('Season')
plt.ylabel('Sales Quantity')
plt.title('Sales Quantity Changes with Different Seasons')
plt.xticks(rotation=0)
plt.grid(True)
plt.tight_layout()

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



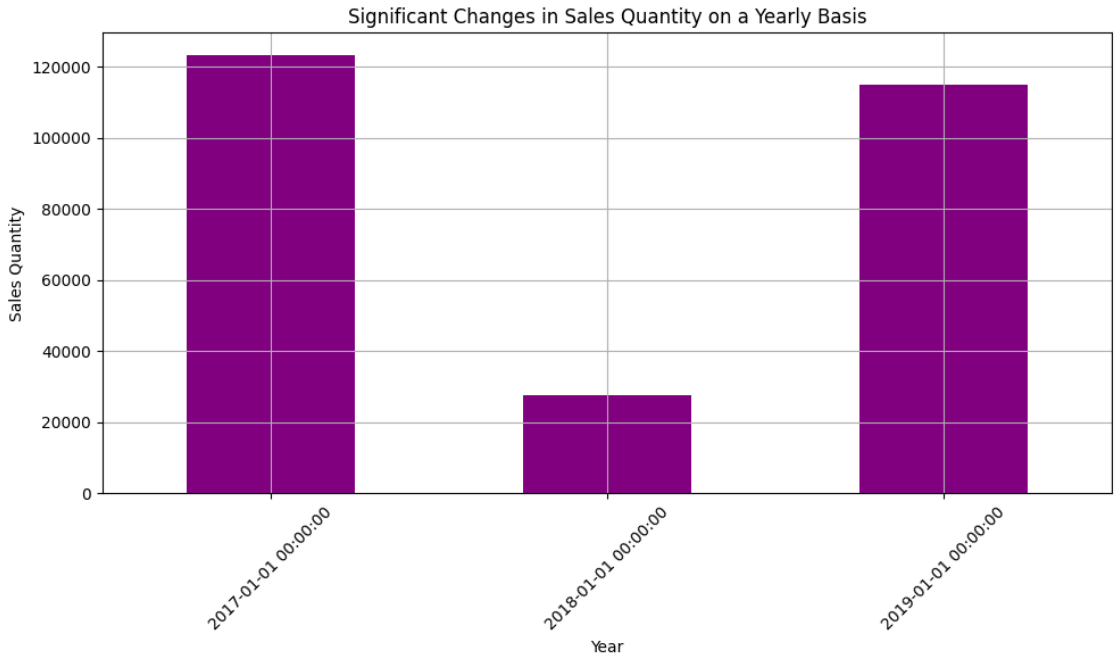
[38]:
```
# Group data by season and calculate the mean of Sales Margin Amount
seasonal_margin = df_new.groupby('Season')['Sales Margin Amount'].mean()

# Create a bar chart
plt.figure(figsize=(10, 6))
seasonal_margin.plot(kind='bar', color='orange')
plt.xlabel('Season')
plt.ylabel('Sales Margin Amount')
plt.title('Patterns in Sales Margin Amount Across Different Seasons')
plt.xticks(rotation=0)
plt.grid(True)
plt.tight_layout()

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



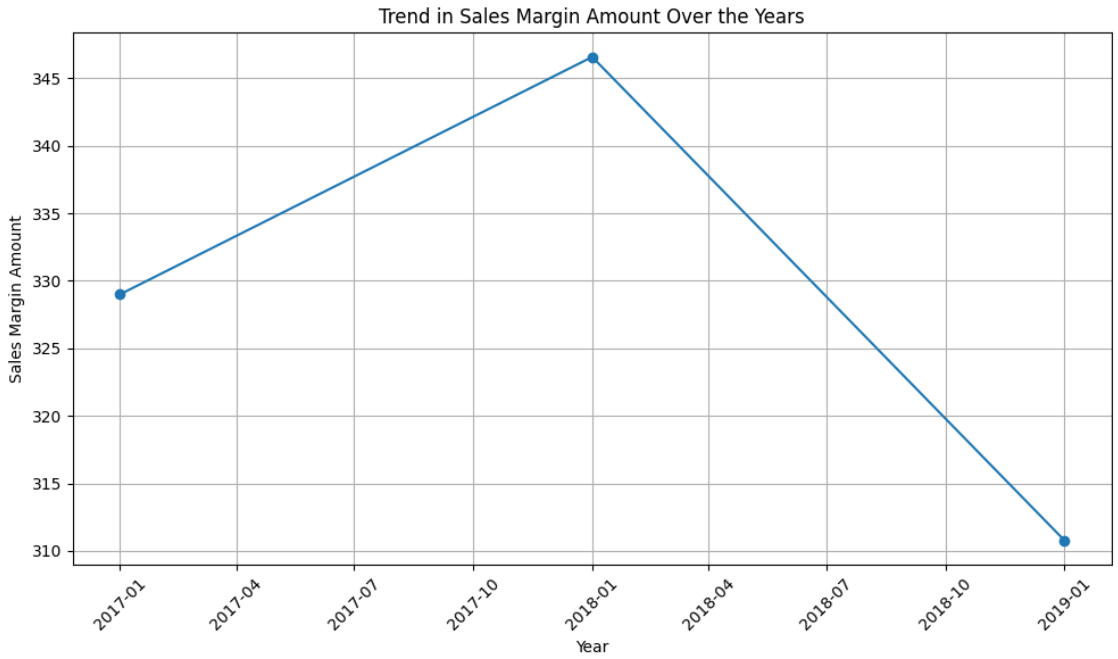
[39]:
```
# Group data by year and calculate the sum of Sales Amount
yearly_sales = df_new.groupby(df_new['DateKey'].dt.to_period('Y'))['Sales␣
 ↪Amount'].sum()

# Convert the index to datetime
yearly_sales.index = yearly_sales.index.to_timestamp()

# Create a line chart
plt.figure(figsize=(10, 6))
plt.plot(yearly_sales.index, yearly_sales.values, marker='o', linestyle='-')
plt.xlabel('Year')
plt.ylabel('Sales Amount')
plt.title('Evolution of Sales Amount Over the Years')
plt.xticks(rotation=45)
plt.grid(True)
plt.tight_layout()

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



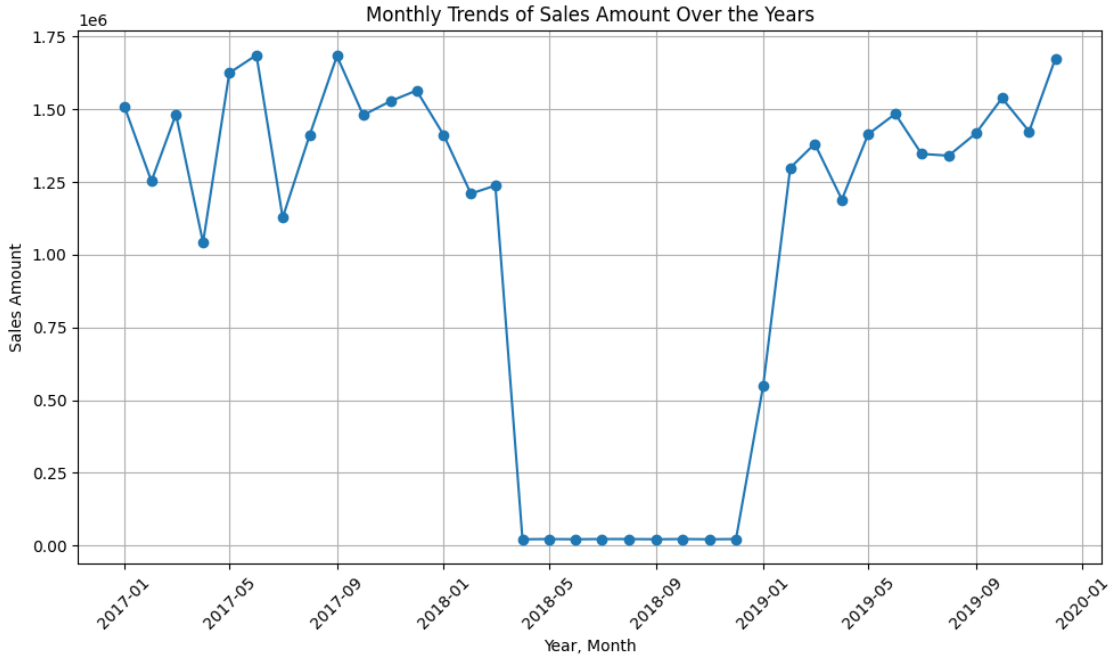
[40]:
```
# Group data by year and calculate the sum of Sales Quantity
yearly_quantity = df_new.groupby(df_new['DateKey'].dt.to_period('Y'))['Sales␣
↪Quantity'].sum()

# Convert the index to datetime
yearly_quantity.index = yearly_quantity.index.to_timestamp()

# Create a bar chart
plt.figure(figsize=(10, 6))
yearly_quantity.plot(kind='bar', color='purple')
plt.xlabel('Year')
plt.ylabel('Sales Quantity')
plt.title('Significant Changes in Sales Quantity on a Yearly Basis')
plt.xticks(rotation=45)
plt.grid(True)
plt.tight_layout()

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



[41]:
```
# Group data by year and calculate the mean of Sales Margin Amount
yearly_margin = df_new.groupby(df_new['DateKey'].dt.to_period('Y'))['Sales␣
 ↪Margin Amount'].mean()

# Convert the index to datetime
yearly_margin.index = yearly_margin.index.to_timestamp()

# Create a line chart
plt.figure(figsize=(10, 6))
plt.plot(yearly_margin.index, yearly_margin.values, marker='o', linestyle='-')
plt.xlabel('Year')
plt.ylabel('Sales Margin Amount')
plt.title('Trend in Sales Margin Amount Over the Years')
plt.xticks(rotation=45)
plt.grid(True)
plt.tight_layout()

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

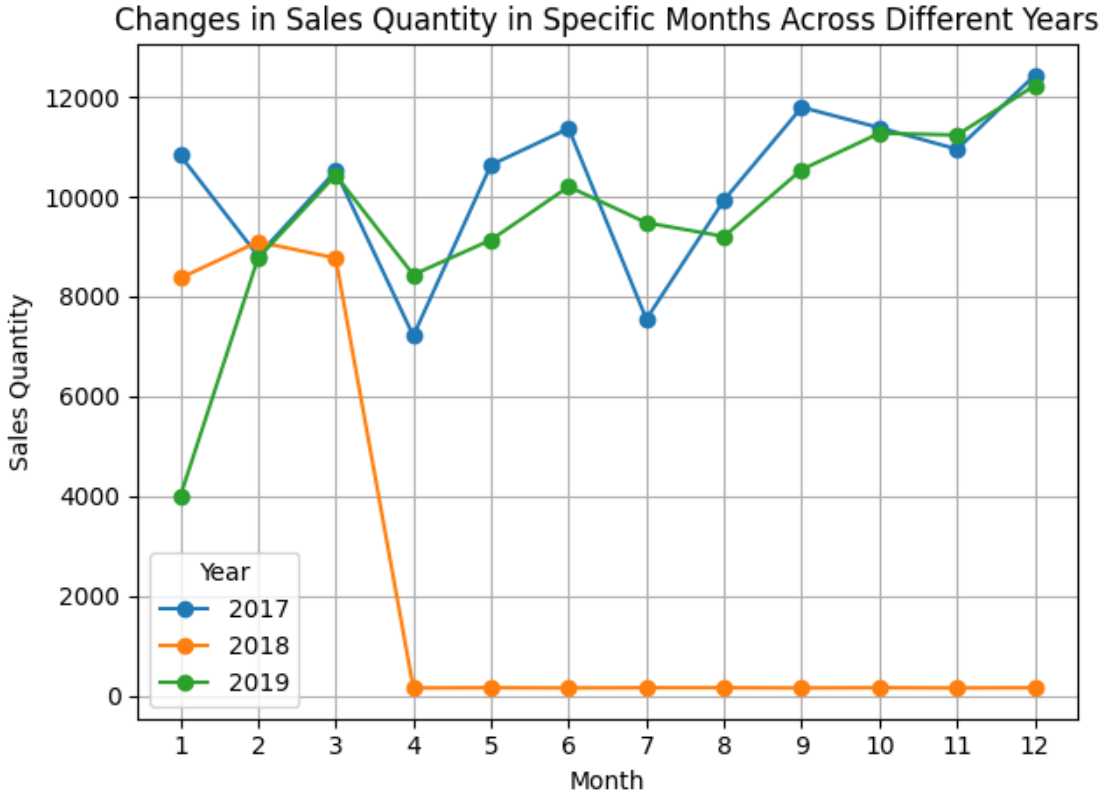


[42]:
```
# Group data by year-month and calculate the sum of Sales Amount
monthly_sales = df_new.groupby(df_new['DateKey'].dt.to_period('M'))['Sales␣
 ↪Amount'].sum()

# Convert the index to datetime
monthly_sales.index = monthly_sales.index.to_timestamp()

# Create a line chart
plt.figure(figsize=(10, 6))
plt.plot(monthly_sales.index, monthly_sales.values, marker='o', linestyle='-')
plt.xlabel('Year, Month')
plt.ylabel('Sales Amount')
plt.title('Monthly Trends of Sales Amount Over the Years')
plt.xticks(rotation=45)
plt.grid(True)
plt.tight_layout()

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



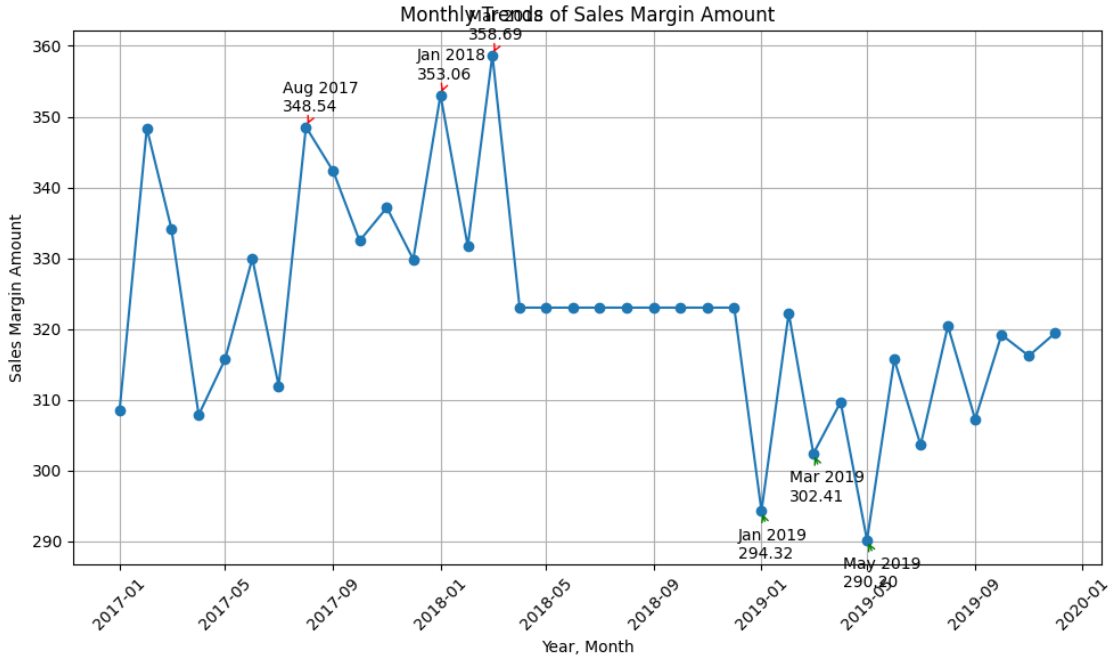
[45]:
```
# Extract year and month from the 'DateKey' column
df_new['Year'] = df_new['DateKey'].dt.year
df_new['Month'] = df_new['DateKey'].dt.month

# Group data by year, month and calculate the sum of Sales Quantity
grouped_data = df_new.groupby(['Year', 'Month'])['Sales Quantity'].sum().
  ↪reset_index()

# Pivot the data to have 'Year' as columns and 'Month' as index
pivot_table = grouped_data.pivot(index='Month', columns='Year', values='Sales␣
  ↪Quantity')

# Create a line chart
plt.figure(figsize=(10, 6))
pivot_table.plot(marker='o', linestyle='-')
plt.xlabel('Month')
plt.ylabel('Sales Quantity')
plt.title('Changes in Sales Quantity in Specific Months Across Different Years')
plt.xticks(range(1, 13))
plt.grid(True)
plt.legend(title='Year')
plt.tight_layout()

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

```
<Figure size 1000x600 with 0 Axes>
```



[47]:
```
# Group data by year-month and calculate the mean of Sales Margin Amount
monthly_margin = df_new.groupby(df_new['DateKey'].dt.to_period('M'))['Sales␣
 ↪Margin Amount'].mean()

# Convert the index to datetime
monthly_margin.index = monthly_margin.index.to_timestamp()

# Create a line chart
plt.figure(figsize=(10, 6))
plt.plot(monthly_margin.index, monthly_margin.values, marker='o', linestyle='-')
plt.xlabel('Year, Month')
plt.ylabel('Sales Margin Amount')
plt.title('Monthly Trends of Sales Margin Amount')
plt.xticks(rotation=45)
plt.grid(True)
plt.tight_layout()

# Identify specific months with high and low Sales Margin Amount
highest_margin_months = monthly_margin.nlargest(3)
```

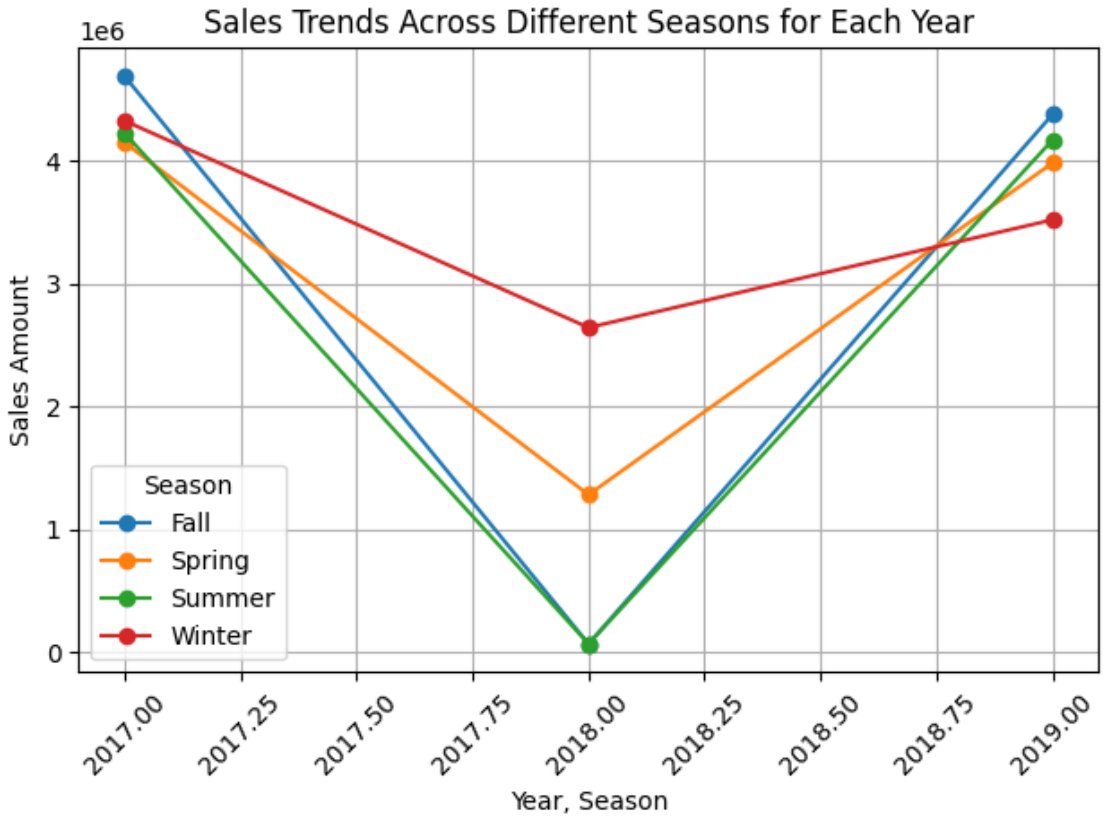
```
lowest_margin_months = monthly_margin.nsmallest(3)

# Highlight specific months with high Sales Margin Amount in red
for month, margin in highest_margin_months.items():
    plt.annotate(f'{month.strftime("%b %Y")}\n{margin:.2f}', xy=(month,␣
 ↪margin), xytext=(-15, 10),
                 textcoords='offset points', arrowprops=dict(arrowstyle="->",␣
 ↪color='red'))

# Highlight specific months with low Sales Margin Amount in green
for month, margin in lowest_margin_months.items():
    plt.annotate(f'{month.strftime("%b %Y")}\n{margin:.2f}', xy=(month,␣
 ↪margin), xytext=(-15, -30),
                 textcoords='offset points', arrowprops=dict(arrowstyle="->",␣
 ↪color='green'))

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



[48]:
```
# Group data by year-season and calculate the sum of Sales Amount
yearly_seasonal_sales = df_new.groupby([df_new['DateKey'].dt.year,␣
 ↪'Season'])['Sales Amount'].sum()

# Convert the MultiIndex to columns
yearly_seasonal_sales = yearly_seasonal_sales.unstack(level='Season')
```

```
# Create a line chart
plt.figure(figsize=(10, 6))
yearly_seasonal_sales.plot(marker='o', linestyle='-')
plt.xlabel('Year, Season')
plt.ylabel('Sales Amount')
plt.title('Sales Trends Across Different Seasons for Each Year')
plt.xticks(rotation=45)
plt.grid(True)
plt.tight_layout()

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

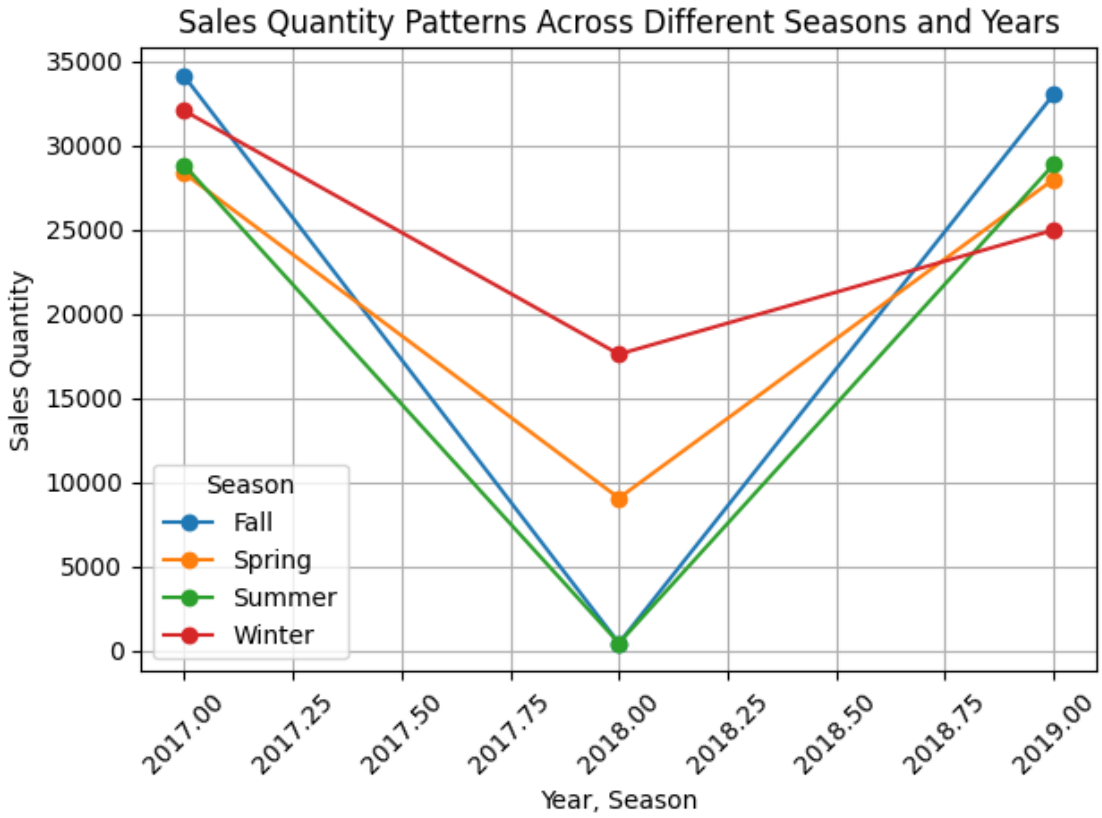
```
<Figure size 1000x600 with 0 Axes>
```



[49]:
```
# Group data by year-season and calculate the sum of Sales Quantity
yearly_seasonal_quantity = df_new.groupby([df_new['DateKey'].dt.year,␣
  ↪'Season'])['Sales Quantity'].sum()

# Convert the MultiIndex to columns
```

```
yearly_seasonal_quantity = yearly_seasonal_quantity.unstack(level='Season')

# Create a line chart
plt.figure(figsize=(10, 6))
yearly_seasonal_quantity.plot(marker='o', linestyle='-')
plt.xlabel('Year, Season')
plt.ylabel('Sales Quantity')
plt.title('Sales Quantity Patterns Across Different Seasons and Years')
plt.xticks(rotation=45)
plt.grid(True)
plt.tight_layout()

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

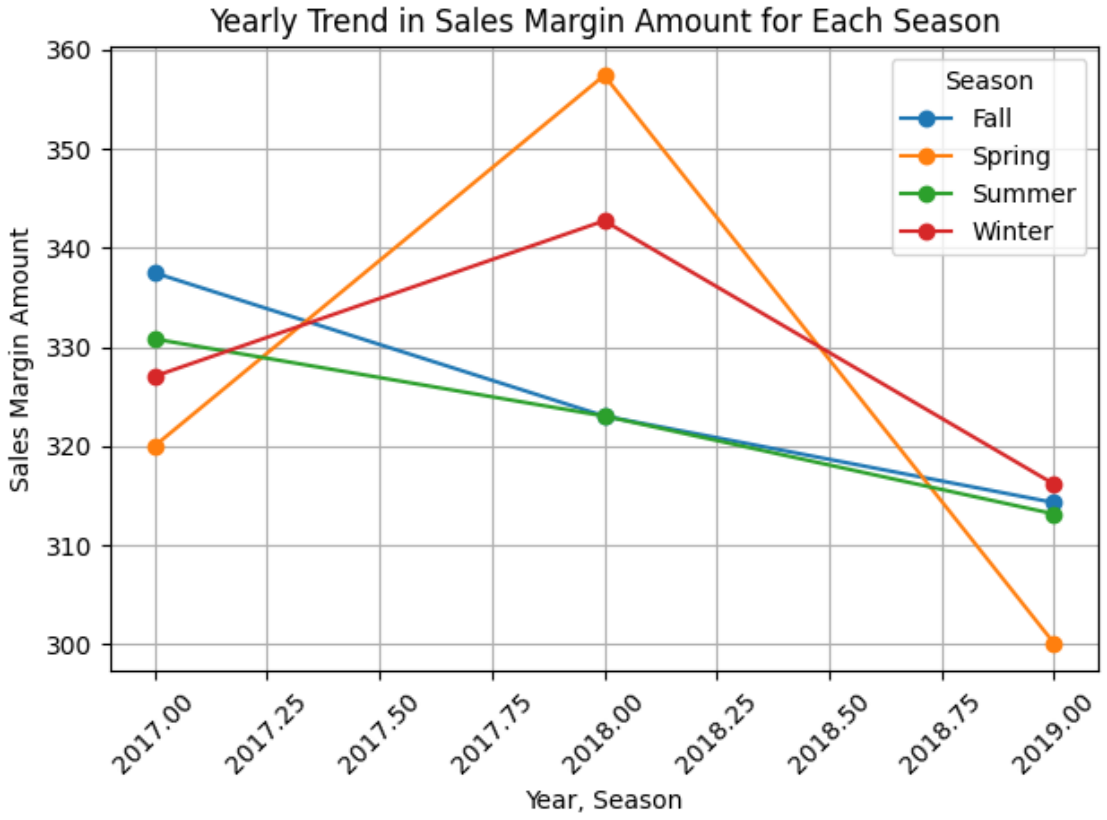
```
<Figure size 1000x600 with 0 Axes>
```



[50]:
```
# Group data by year-season and calculate the mean of Sales Margin Amount
yearly_seasonal_margin = df_new.groupby([df_new['DateKey'].dt.year,␣
  ↪'Season'])['Sales Margin Amount'].mean()
```

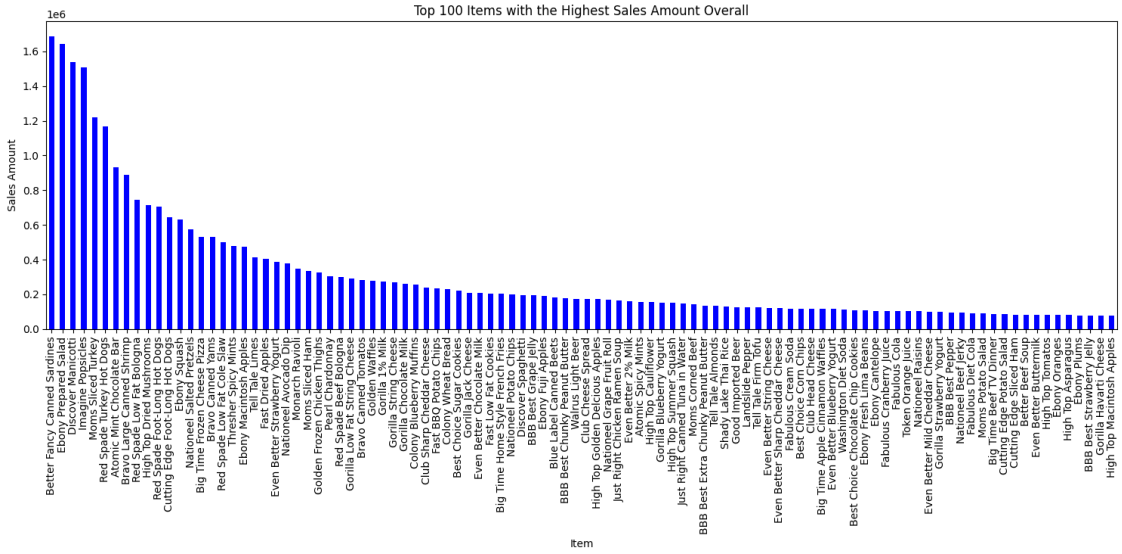
```
# Convert the MultiIndex to columns
yearly_seasonal_margin = yearly_seasonal_margin.unstack(level='Season')

# Create a line chart
plt.figure(figsize=(10, 6))
yearly_seasonal_margin.plot(marker='o', linestyle='-')
plt.xlabel('Year, Season')
plt.ylabel('Sales Margin Amount')
plt.title('Yearly Trend in Sales Margin Amount for Each Season')
plt.xticks(rotation=45)
plt.grid(True)
plt.tight_layout()

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

```
<Figure size 1000x600 with 0 Axes>
```



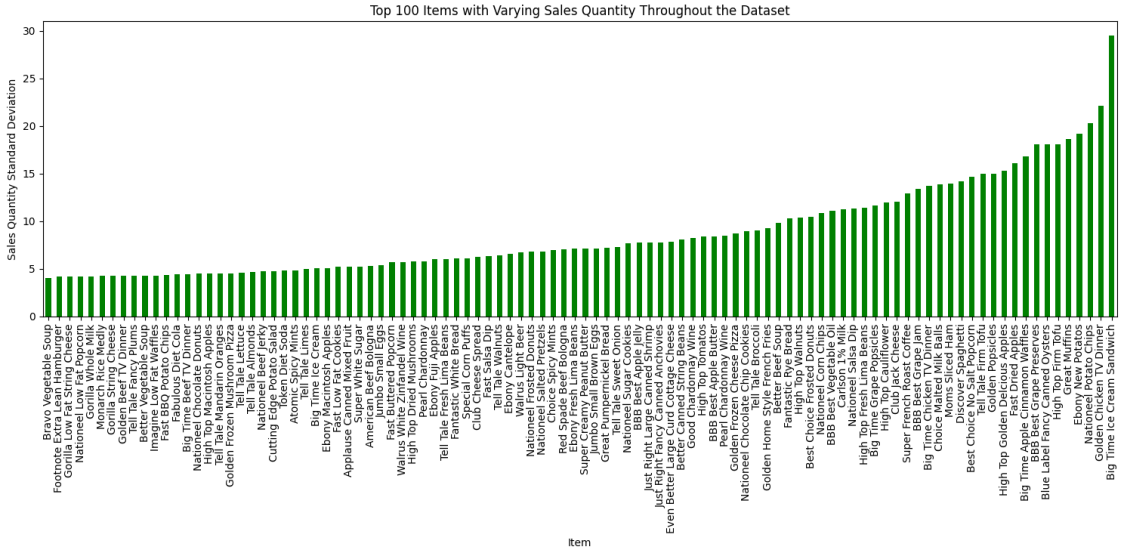
[60]:
```
import matplotlib.pyplot as plt
import pandas as pd
```

```
# Assuming 'Item' and 'Sales Amount' columns exist in your DataFrame 'df_new'

# Group data by 'Item' and calculate the sum of Sales Amount
item_sales = df_new.groupby('Item')['Sales Amount'].sum()

# Sort items by Sales Amount in descending order and take the top 40
sorted_items = item_sales.nlargest(100)

# Create a bar chart
plt.figure(figsize=(15, 7.5))
sorted_items.plot(kind='bar', color='blue')
plt.xlabel('Item')
plt.ylabel('Sales Amount')
plt.title('Top 100 Items with the Highest Sales Amount Overall')
plt.xticks(rotation=90)
plt.tight_layout()

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

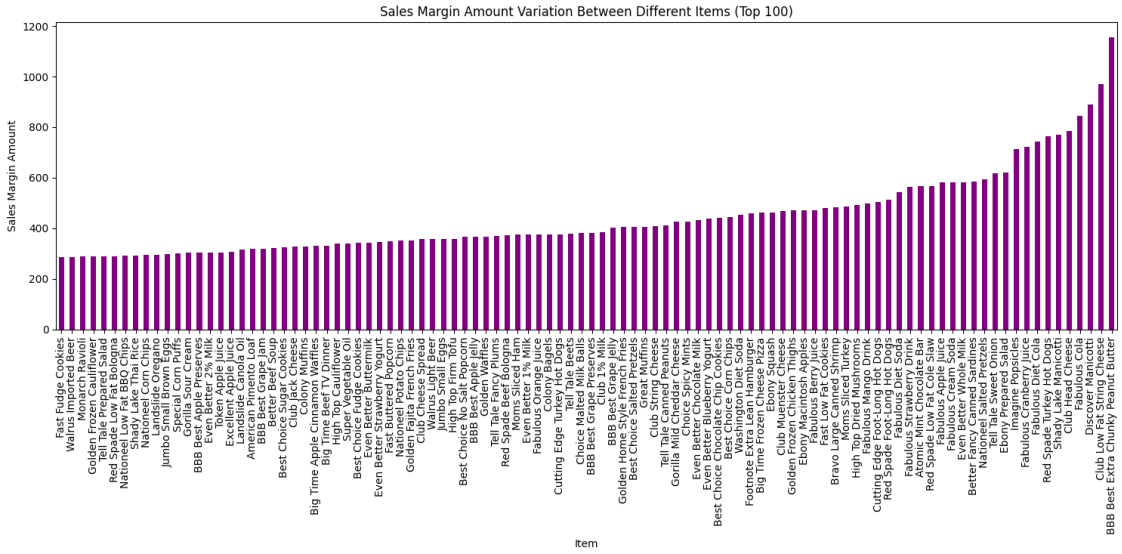


[62]:
```
# Group data by 'Item' and calculate the standard deviation of Sales Quantity
item_quantity_std = df_new.groupby('Item')['Sales Quantity'].std()

# Filter items with varying Sales Quantity and take the top 100
varying_items = item_quantity_std.nlargest(100)

# Create a bar chart
plt.figure(figsize=(15, 7.5))
```

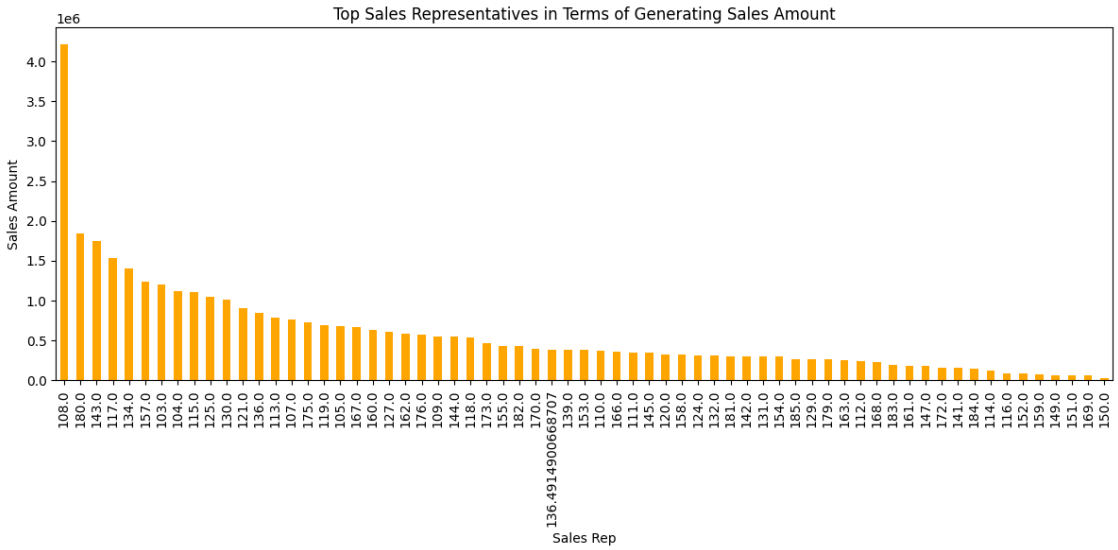
```
varying_items.sort_values().plot(kind='bar', color='green')
plt.xlabel('Item')
plt.ylabel('Sales Quantity Standard Deviation')
plt.title('Top 100 Items with Varying Sales Quantity Throughout the Dataset')
plt.xticks(rotation=90)
plt.tight_layout()

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



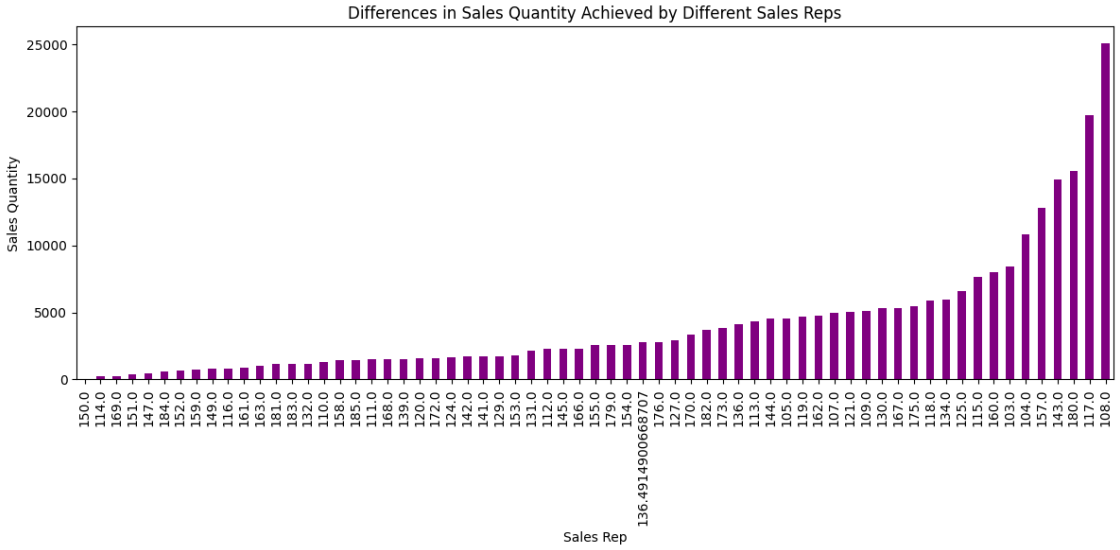
[64]:
```
# Group data by 'Item' and calculate the mean of Sales Margin Amount
item_margin = df_new.groupby('Item')['Sales Margin Amount'].mean()

# Filter items and take the top 100
top_items = item_margin.nlargest(100)

# Create a bar chart
plt.figure(figsize=(15, 7.5))
top_items.sort_values().plot(kind='bar', color='purple')
plt.xlabel('Item')
plt.ylabel('Sales Margin Amount')
plt.title('Sales Margin Amount Variation Between Different Items (Top 100)')
plt.xticks(rotation=90)
plt.tight_layout()

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



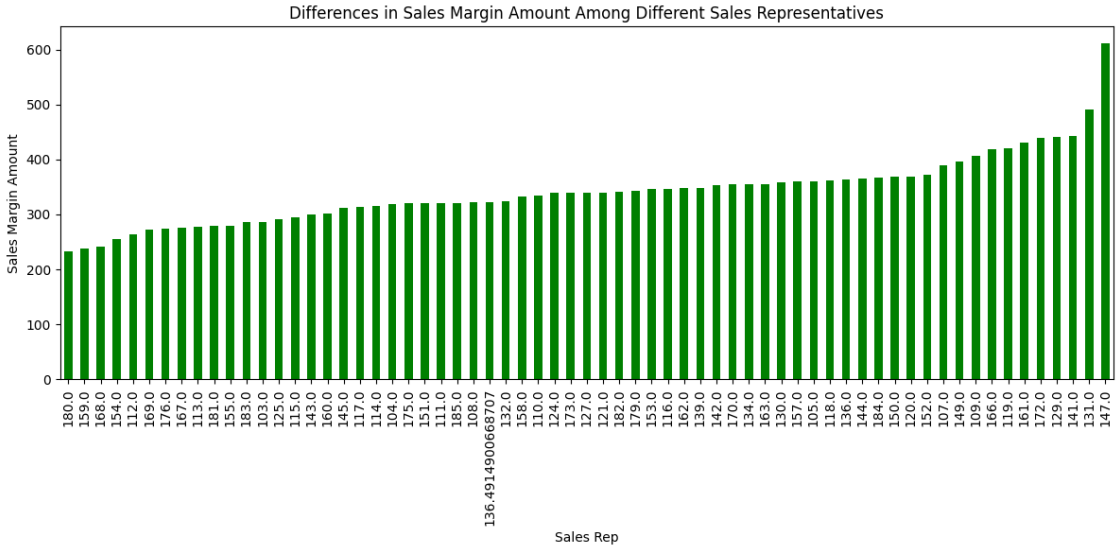
[65]:
```
# Group data by 'Sales Rep' and calculate the sum of Sales Amount
sales_rep_sales = df_new.groupby('Sales Rep')['Sales Amount'].sum()

# Sort sales representatives by Sales Amount in descending order
sorted_sales_reps = sales_rep_sales.sort_values(ascending=False)

# Create a bar chart
plt.figure(figsize=(12, 6))
sorted_sales_reps.plot(kind='bar', color='orange')
plt.xlabel('Sales Rep')
plt.ylabel('Sales Amount')
plt.title('Top Sales Representatives in Terms of Generating Sales Amount')
plt.xticks(rotation=90)
plt.tight_layout()

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



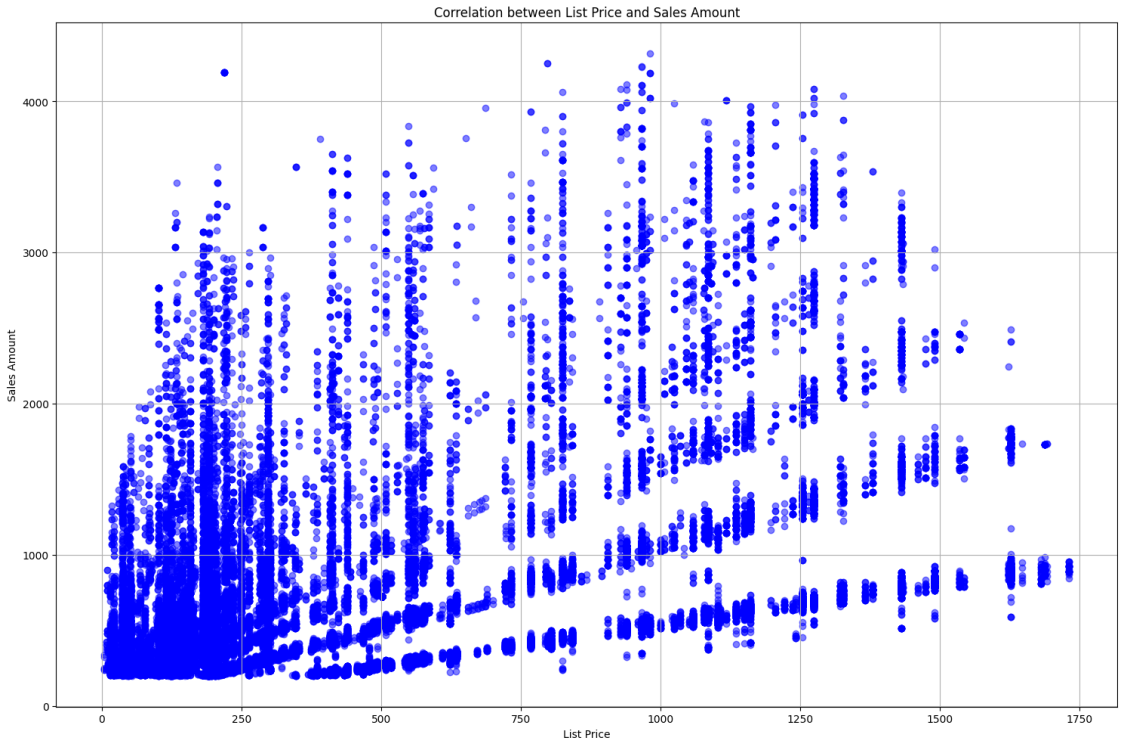
[66]:
```
# Group data by 'Sales Rep' and calculate the sum of Sales Quantity
sales_rep_quantity = df_new.groupby('Sales Rep')['Sales Quantity'].sum()

# Create a bar chart
plt.figure(figsize=(12, 6))
sales_rep_quantity.sort_values().plot(kind='bar', color='purple')
plt.xlabel('Sales Rep')
plt.ylabel('Sales Quantity')
plt.title('Differences in Sales Quantity Achieved by Different Sales Reps')
plt.xticks(rotation=90)
plt.tight_layout()

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



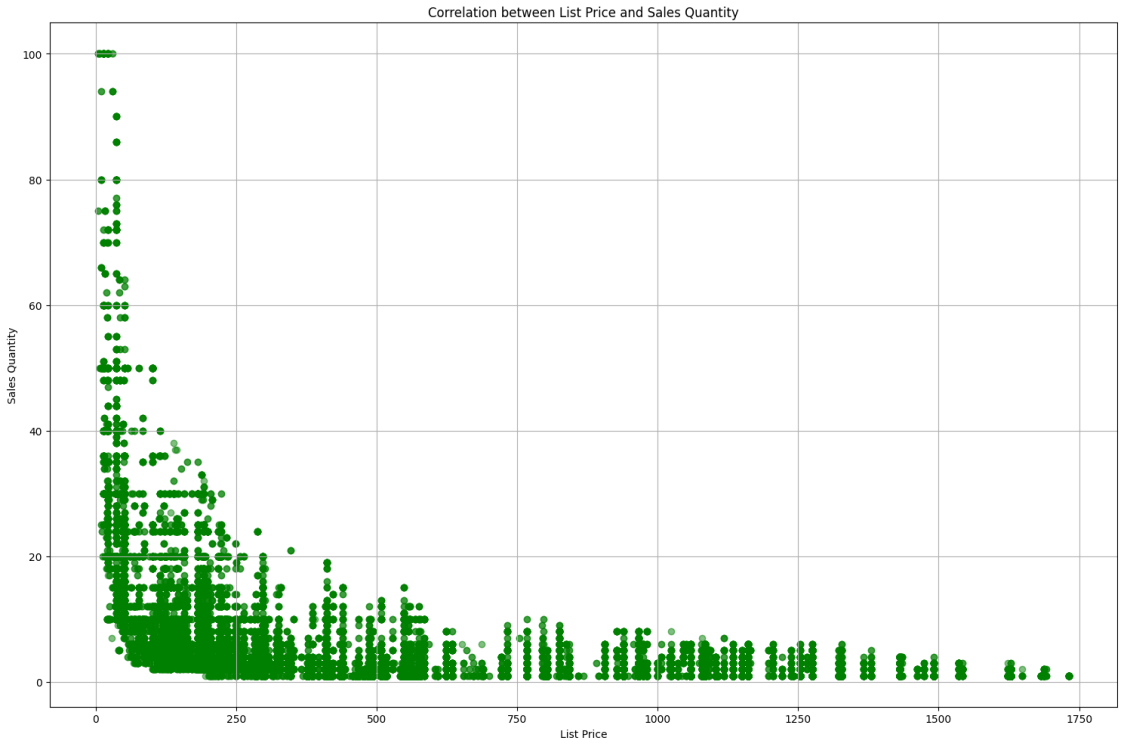
[67]:
```
# Group data by 'Sales Rep' and calculate the mean of Sales Margin Amount
sales_rep_margin = df_new.groupby('Sales Rep')['Sales Margin Amount'].mean()

# Create a bar chart
plt.figure(figsize=(12, 6))
sales_rep_margin.sort_values().plot(kind='bar', color='green')
plt.xlabel('Sales Rep')
plt.ylabel('Sales Margin Amount')
plt.title('Differences in Sales Margin Amount Among Different Sales␣
  ↪Representatives')
plt.xticks(rotation=90)
plt.tight_layout()

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



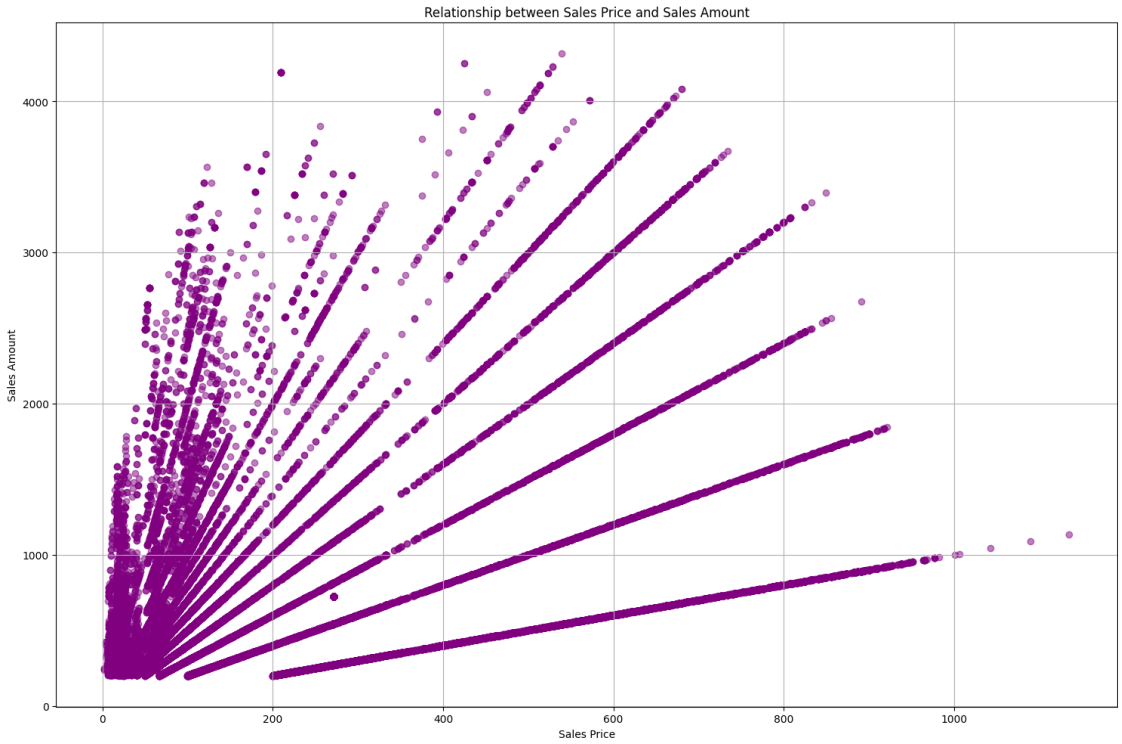
[70]:
```
# Create a scatter plot
plt.figure(figsize=(15, 10))
plt.scatter(df_new['List Price'], df_new['Sales Amount'], color='blue', alpha=0.
↪5)
plt.xlabel('List Price')
plt.ylabel('Sales Amount')
plt.title('Correlation between List Price and Sales Amount')
plt.grid(True)
plt.tight_layout()

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



[71]:
```
# Create a scatter plot
plt.figure(figsize=(15, 10))
plt.scatter(df_new['List Price'], df_new['Sales Quantity'], color='green',␣
 ↪alpha=0.5)
plt.xlabel('List Price')
plt.ylabel('Sales Quantity')
plt.title('Correlation between List Price and Sales Quantity')
plt.grid(True)
plt.tight_layout()

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

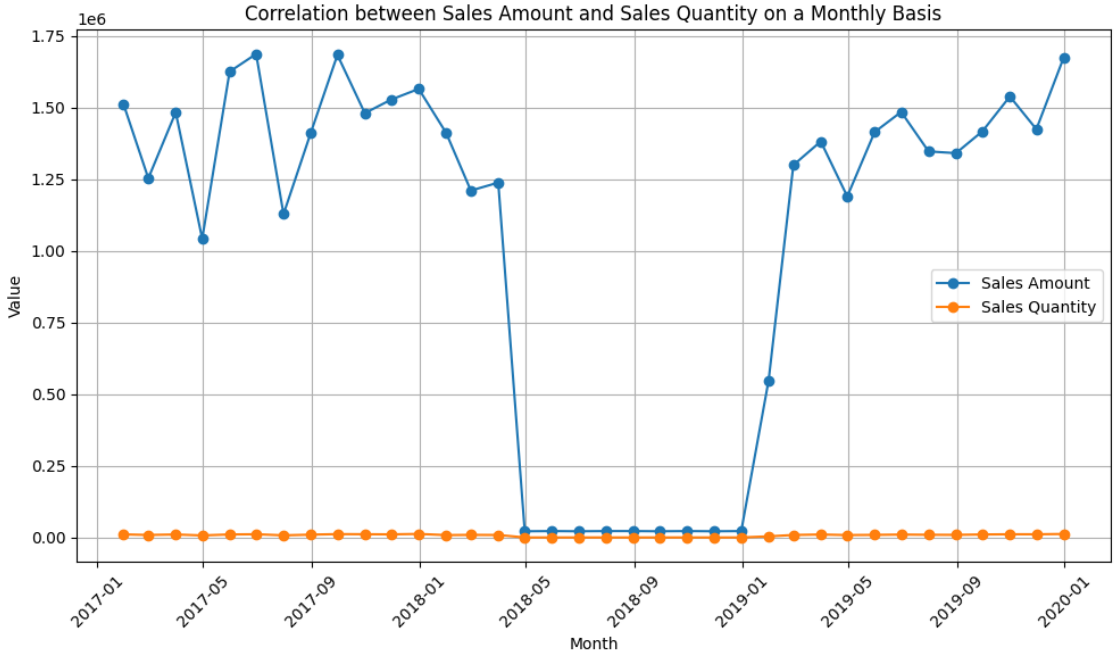


[73]:
```
# Create a scatter plot
plt.figure(figsize=(15, 10))
plt.scatter(df_new['Sales Price'], df_new['Sales Amount'], color='purple',␣
 ↪alpha=0.5)
plt.xlabel('Sales Price')
plt.ylabel('Sales Amount')
plt.title('Relationship between Sales Price and Sales Amount')
plt.grid(True)
plt.tight_layout()

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



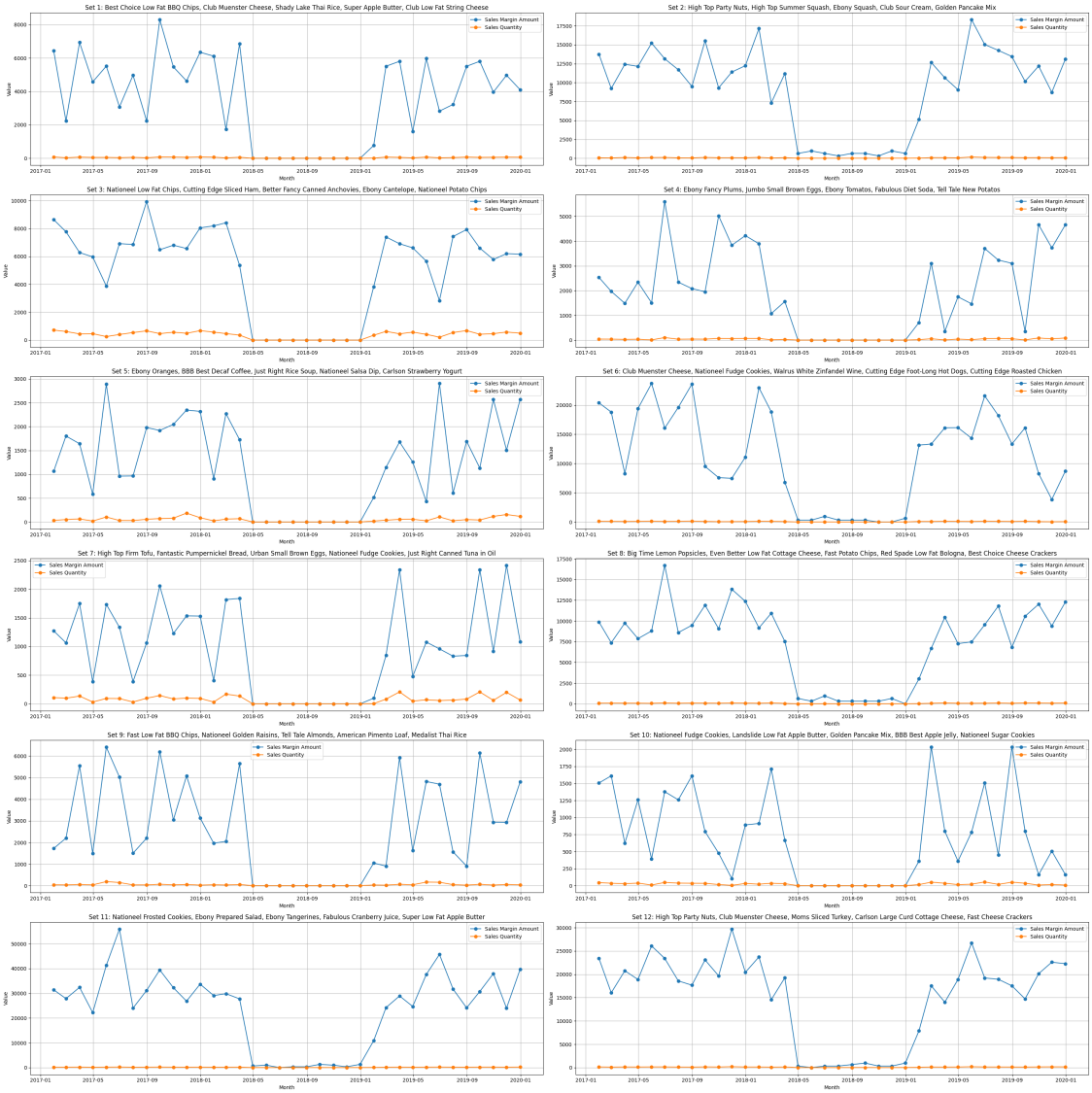
[78]:
```
# Convert the 'DateKey' column to datetime
df_new['DateKey'] = pd.to_datetime(df_new['DateKey'])

# Set 'DateKey' as the index
df_new.set_index('DateKey', inplace=True)

# Resample the data on a monthly basis and calculate the sum of Sales Amount␣
 ↪and Sales Quantity
monthly_sales = df_new.resample('M')['Sales Amount'].sum()
monthly_quantity = df_new.resample('M')['Sales Quantity'].sum()

# Create a multi-line chart
plt.figure(figsize=(10, 6))
plt.plot(monthly_sales.index, monthly_sales.values, marker='o', linestyle='-',␣
 ↪label='Sales Amount')
plt.plot(monthly_quantity.index, monthly_quantity.values, marker='o',␣
 ↪linestyle='-', label='Sales Quantity')
plt.xlabel('Month')
plt.ylabel('Value')
plt.title('Correlation between Sales Amount and Sales Quantity on a Monthly␣
 ↪Basis')
plt.legend()
plt.xticks(rotation=45)
```

```
plt.grid(True)
plt.tight_layout()

# Display the multi-line chart
plt.show()
```



[86]:
```
# Get unique items from the DataFrame
unique_items = df_new['Item'].unique()

# Number of sets of charts to create
num_sets = 12

# Create 10 sets of charts in a 3x3 layout
num_rows = 6
num_cols = 2

fig, axs = plt.subplots(num_rows, num_cols, figsize=(30, 30))

for i in range(num_sets):
    # Randomly select 5 items from the unique items
    selected_items = random.sample(list(unique_items), 5)

    # Filter the DataFrame to include only the selected items
    df_selected_items = df_new[df_new['Item'].isin(selected_items)]
```

```
    # Resample the data on a monthly basis and calculate the sum of Sales␣
 ↪Margin Amount and Sales Quantity
    monthly_margin = df_selected_items.resample('M')['Sales Margin Amount'].
 ↪sum()
    monthly_quantity = df_selected_items.resample('M')['Sales Quantity'].sum()

    # Calculate the position in the subplot grid
    row = i // num_cols
    col = i % num_cols

    # Plot on the current subplot
    axs[row, col].plot(monthly_margin.index, monthly_margin.values, marker='o',␣
 ↪linestyle='-', label='Sales Margin Amount')
    axs[row, col].plot(monthly_quantity.index, monthly_quantity.values,␣
 ↪marker='o', linestyle='-', label='Sales Quantity')

    # Set the title to include selected items
    title = ', '.join(selected_items)
    axs[row, col].set_title(f'Set {i+1}: {title}')

    axs[row, col].set_xlabel('Month')
    axs[row, col].set_ylabel('Value')
    axs[row, col].legend()
    axs[row, col].grid(True)

# Adjust layout and display the subplots
plt.tight_layout()
plt.show()
```

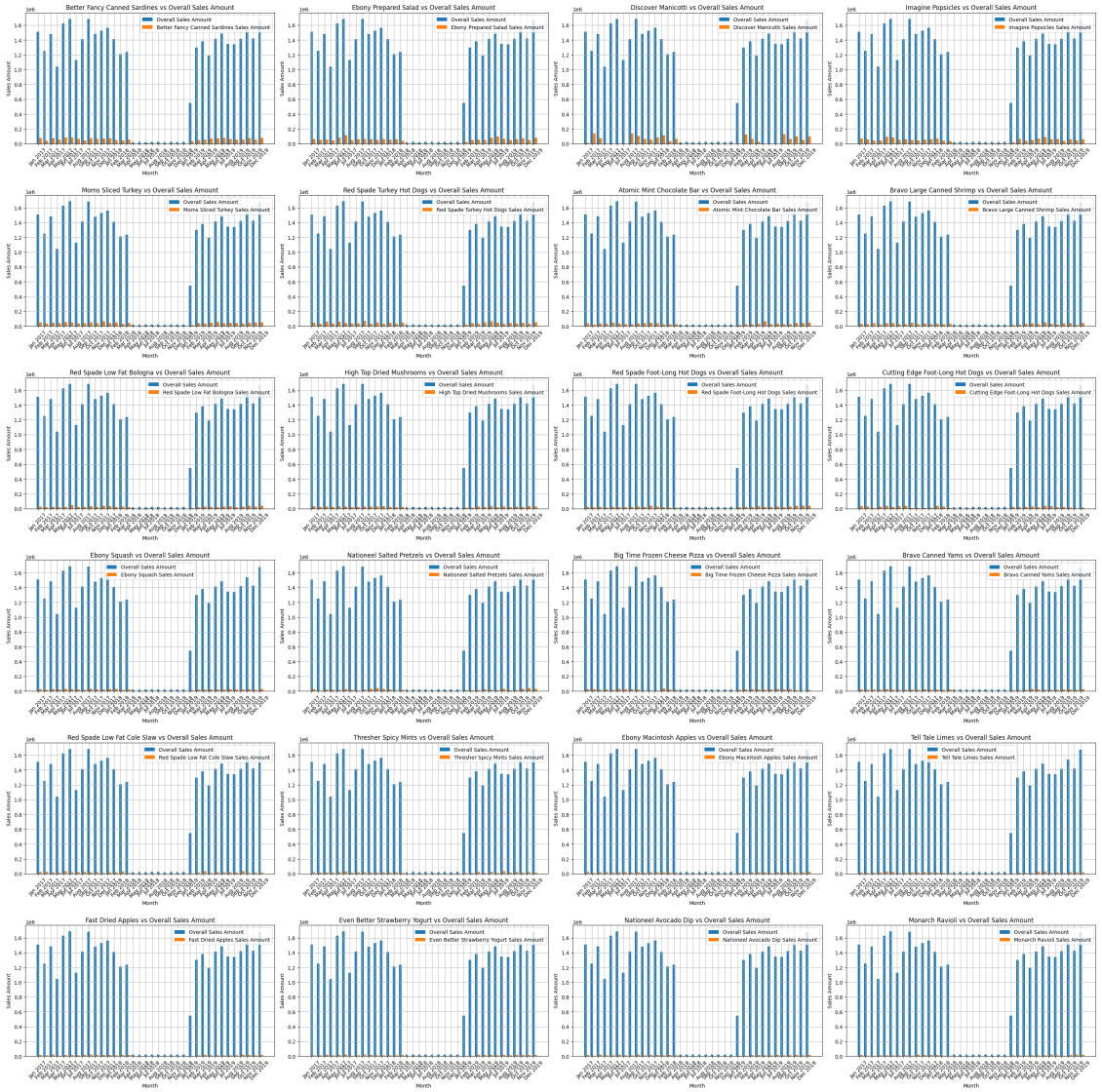


[91]:
```python
# Group data by Item and calculate the total Sales Amount for each item
item_sales = df_new.groupby('Item')['Sales Amount'].sum()

# Get the top 120 items based on Sales Amount
top_items = item_sales.nlargest(120).index

# Split the top items into sets of 10
item_sets = [top_items[i:i+5] for i in range(0, len(top_items), 5)]

# Create a multi-line chart for each set of items
num_rows = 8
num_cols = 3
```

```
fig, axs = plt.subplots(num_rows, num_cols, figsize=(30, 30))

for i, item_set in enumerate(item_sets):
    row = i // num_cols
    col = i % num_cols

    axs[row, col].set_title(f'Set {i+1}')

    for item in item_set:
        # Filter the DataFrame for the current item
        df_item = df_new[df_new['Item'] == item]

        # Group data by DateKey and calculate the sum of Sales Amount
        monthly_sales = df_item.resample('M')['Sales Amount'].sum()

        # Plot the line for the current item in the current subplot
        axs[row, col].plot(monthly_sales.index, monthly_sales.values,␣
 ↪marker='o', label=item)

    axs[row, col].set_xlabel('Month')
    axs[row, col].set_ylabel('Sales Amount')
    axs[row, col].legend()
    axs[row, col].grid(True)

# Adjust layout and display the subplots
plt.tight_layout()
plt.show()
```

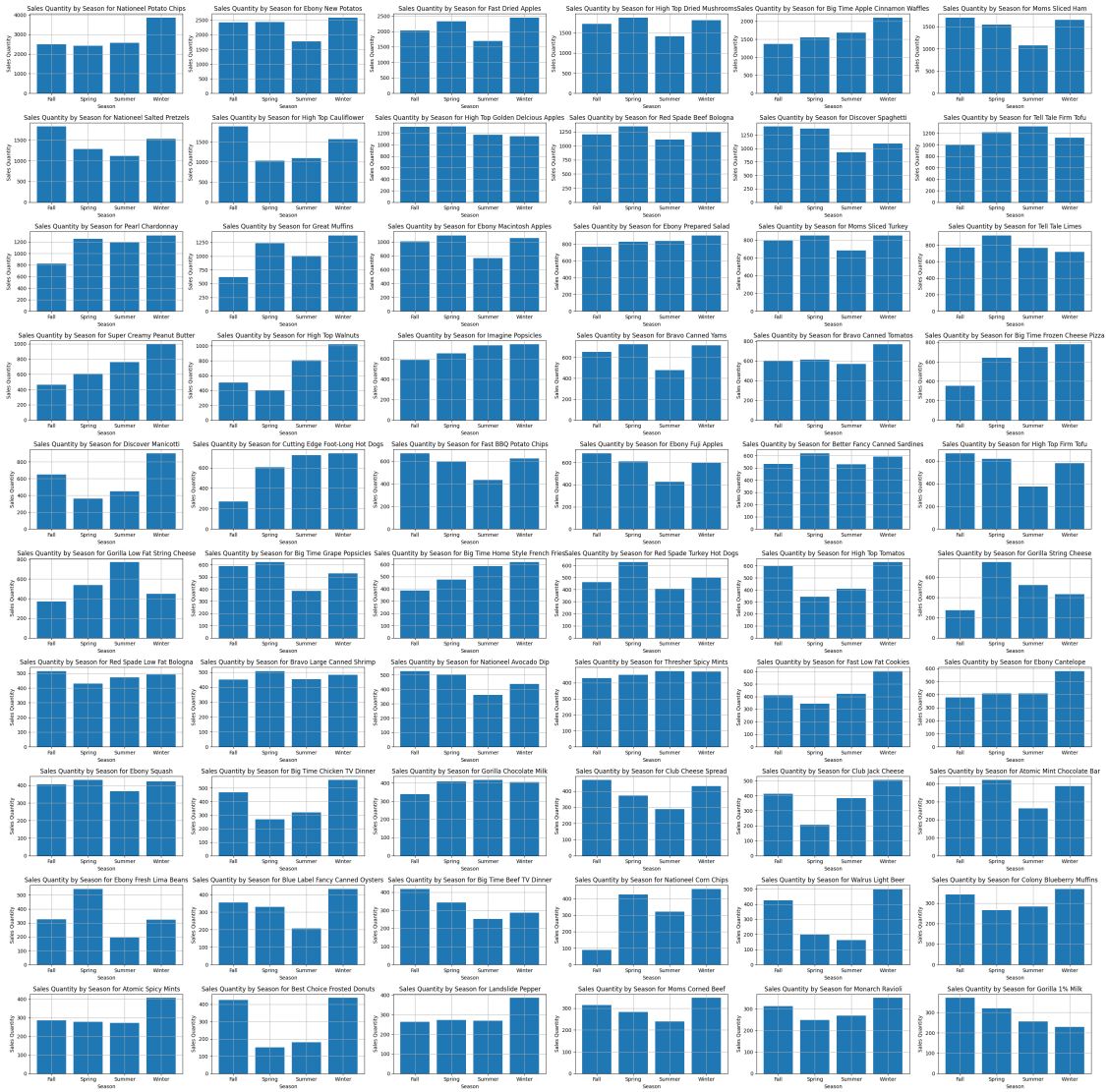
[94]:
```
# Group data by DateKey and calculate the total Sales Amount
overall_sales = df_new.resample('M')['Sales Amount'].sum()

# Get the top 24 items based on Sales Amount
top_items = df_new.groupby('Item')['Sales Amount'].sum().nlargest(24).index

# Create a 6x4 layout for the subplots
num_rows = 6
num_cols = 4
fig, axs = plt.subplots(num_rows, num_cols, figsize=(30, 30))

for i, specific_item in enumerate(top_items):
    # Calculate row and column for subplot
```

```
    row = i // num_cols
    col = i % num_cols

    # Filter the DataFrame for the specific item
    df_specific_item = df_new[df_new['Item'] == specific_item]

    # Group data by DateKey and calculate the total Sales Amount for the␣
 ↪specific item
    specific_item_sales = df_specific_item.resample('M')['Sales Amount'].sum()

    # Create a grouped bar chart for the specific item against all items
    bar_width = 0.4
    bar_positions = range(len(overall_sales))

    axs[row, col].bar(bar_positions, overall_sales, width=bar_width,␣
 ↪label='Overall Sales Amount')
    axs[row, col].bar([pos + bar_width for pos in bar_positions],␣
 ↪specific_item_sales, width=bar_width, label=f'{specific_item} Sales Amount')

    axs[row, col].set_xlabel('Month')
    axs[row, col].set_ylabel('Sales Amount')
    axs[row, col].set_title(f'{specific_item} vs Overall Sales Amount')
    axs[row, col].set_xticks(bar_positions)
    axs[row, col].set_xticklabels(overall_sales.index.strftime('%b %Y'),␣
 ↪rotation=45)
    axs[row, col].legend()
    axs[row, col].grid(True)

# Adjust layout and display the subplots
plt.tight_layout()
plt.show()
```



[96]:
```
# Get the top 24 items based on Sales Quantity
top_items = df_new.groupby('Item')['Sales Quantity'].sum().nlargest(60).index

# Create a 6x4 layout for the subplots
num_rows = 10
num_cols = 6
fig, axs = plt.subplots(num_rows, num_cols, figsize=(30, 30))

for i, specific_item in enumerate(top_items):
    # Calculate row and column for subplot
    row = i // num_cols
    col = i % num_cols
```

```
    # Filter the DataFrame for the specific item
    df_specific_item = df_new[df_new['Item'] == specific_item]

    # Group data by Season and calculate the total Sales Quantity for the␣
 ↪specific item
    season_quantity = df_specific_item.groupby('Season')['Sales Quantity'].sum()

    # Create a grouped bar chart for the specific item across different seasons
    bar_positions = range(len(season_quantity))

    axs[row, col].bar(bar_positions, season_quantity,␣
 ↪tick_label=season_quantity.index)

    axs[row, col].set_xlabel('Season')
    axs[row, col].set_ylabel('Sales Quantity')
    axs[row, col].set_title(f'Sales Quantity by Season for {specific_item}')
    axs[row, col].grid(True)

# Adjust layout and display the subplots
plt.tight_layout()
plt.show()
```



[102]:
```
# Get the top 60 items based on Sales Margin Amount
top_items = df_new.groupby('Item')['Sales Margin Amount'].sum().nlargest(60).
  ↪index

# Create a 10x6 layout for the subplots
num_rows = 20
num_cols = 3
fig, axs = plt.subplots(num_rows, num_cols, figsize=(30, 60))

for i, specific_item in enumerate(top_items):
    # Calculate row and column for subplot
    row = i // num_cols
    col = i % num_cols
```

```
    # Filter the DataFrame for the specific item
    df_specific_item = df_new[df_new['Item'] == specific_item]

    # Group data by Sales Rep and calculate the total Sales Margin Amount for␣
 ↪the specific item
    sales_rep_margin = df_specific_item.groupby('Sales Rep')['Sales Margin␣
 ↪Amount'].sum()

    # Create a grouped bar chart for the specific item across different sales␣
 ↪representatives
    bar_positions = range(len(sales_rep_margin))

    axs[row, col].bar(bar_positions, sales_rep_margin,␣
 ↪tick_label=sales_rep_margin.index)

    axs[row, col].set_xlabel('Sales Rep')
    axs[row, col].set_ylabel('Sales Margin Amount')
    axs[row, col].set_title(f'Sales Margin Amount by Sales Rep for␣
 ↪{specific_item}')
    axs[row, col].grid(True)

# Adjust layout and display the subplots
plt.tight_layout()
plt.show()
```


Sales Margin Amount by Sales Rep for Ebony Prepared Salad
Sales Margin Amount by Sales Rep for Discover Manicotti
Sales Margin Amount by Sales Rep for Imagine Popsicles
Sales Margin Amount by Sales Rep for Better Fancy Canned Sardines
Sales Margin Amount by Sales Rep for Red Spade Turkey Hot Dogs
Sales Margin Amount by Sales Rep for Moms Sliced Turkey
Sales Margin Amount by Sales Rep for High Top Dried Mushrooms
Sales Margin Amount by Sales Rep for Atomic Mint Chocolate Bar
Sales Margin Amount by Sales Rep for Bravo Large Canned Shrimp
Sales Margin Amount by Sales Rep for Nationeel Salted Pretzels
Sales Margin Amount by Sales Rep for Cutting Edge Foot-Long Hot Dogs
Sales Margin Amount by Sales Rep for Red Spade Foot-Long Hot Dogs
Sales Margin Amount by Sales Rep for Ebony Macintosh Apples
Sales Margin Amount by Sales Rep for Big Time Frozen Cheese Pizza
Sales Margin Amount by Sales Rep for Ebony Squash
Sales Margin Amount by Sales Rep for Red Spade Low Fat Bologna
Sales Margin Amount by Sales Rep for Red Spade Low Fat Cole Slaw
Sales Margin Amount by Sales Rep for Bravo Canned Yams
Sales Margin Amount by Sales Rep for Red Spade Beef Bologna
Sales Margin Amount by Sales Rep for Fast Dried Apples
Sales Margin Amount by Sales Rep for Thresher Spicy Mints
Sales Margin Amount by Sales Rep for Golden Frozen Chicken Thighs
Sales Margin Amount by Sales Rep for Moms Sliced Ham
Sales Margin Amount by Sales Rep for Nationeel Avocado Dip
Sales Margin Amount by Sales Rep for Monarch Ravioli
Sales Margin Amount by Sales Rep for Fast Low Fat Cookies
Sales Margin Amount by Sales Rep for Fast BBQ Potato Chips
Sales Margin Amount by Sales Rep for Bravo Canned Tomatos
Sales Margin Amount by Sales Rep for Pearl Chardonnay
Sales Margin Amount by Sales Rep for Even Better Strawberry Yogurt
Sales Margin Amount by Sales Rep for Colony Blueberry Muffins
Sales Margin Amount by Sales Rep for Nationeel Potato Chips
Sales Margin Amount by Sales Rep for Best Choice Sugar Cookies
Sales Margin Amount by Sales Rep for Tell Tale Limes
Sales Margin Amount by Sales Rep for High Top Cauliflower
Sales Margin Amount by Sales Rep for Golden Waffles
Sales Margin Amount by Sales Rep for Colony Wheat Bread
Sales Margin Amount by Sales Rep for BBB Best Chunky Peanut Butter
Sales Margin Amount by Sales Rep for Gorilla Blueberry Yogurt
Sales Margin Amount by Sales Rep for Gorilla Chocolate Milk
Sales Margin Amount by Sales Rep for Gorilla 1% Milk
Sales Margin Amount by Sales Rep for BBB Best Grape Jelly
Sales Margin Amount by Sales Rep for Big Time Home Style French Fries
Sales Margin Amount by Sales Rep for Discover Spaghetti
Sales Margin Amount by Sales Rep for Club Cheese Spread
Sales Margin Amount by Sales Rep for Ebony Fuji Apples
Sales Margin Amount by Sales Rep for Just Right Chicken Ramen Soup
Sales Margin Amount by Sales Rep for Big Time Apple Cinnamon Waffles
Sales Margin Amount by Sales Rep for Club Sharp Cheddar Cheese
Sales Margin Amount by Sales Rep for Gorilla Low Fat String Cheese
Sales Margin Amount by Sales Rep for Shady Lake Thai Rice
Sales Margin Amount by Sales Rep for Even Better Chocolate Milk
Sales Margin Amount by Sales Rep for Token Orange Juice
Sales Margin Amount by Sales Rep for BBB Best Extra Chunky Peanut Butter
Sales Margin Amount by Sales Rep for Landslide Pepper
Sales Margin Amount by Sales Rep for Gorilla String Cheese
Sales Margin Amount by Sales Rep for Nationeel Grape Fruit Roll
Sales Margin Amount by Sales Rep for Moms Corned Beef
Sales Margin Amount by Sales Rep for Just Right Canned Tuna in Water
Sales Margin Amount by Sales Rep for Big Time Beef TV Dinner
Sales Margin Amount
Sales Rep

2 Comprehensive Summary: Data Analysis Questions and Answers

2.1 Monthly Trends:

1. **Question:** What is the monthly trend of Sales Amount? **Answer:** The line chart of Sales Amount over time reveals a gap in data from April 2018 to September 2018. Sales Amount peaks in the 6th and 9th months, around 1.5 to 1.66 million, and dips in the 4th and 7th months, around 1.1 to 1.2 million.
2. **Question:** How does Sales Quantity vary on a monthly basis? **Answer:** Sales Quantity follows a similar pattern to Sales Amount, with the highest values occurring in the 6th and 9th months (11,000 to 13,000) and lower values in the 4th and 7th months (7,000 to 9,000).
3. **Question:** Are there any noticeable patterns in Sales Margin Amount on a monthly scale? **Answer:** Sales Margin Amount shows variations over the months, with higher margins in the 2nd and 8th months (around 345 in 2017, 320 in 2019), and lower margins in the 4th and 7th months (around 305 in 2017, 290 in 2019).

2.2 Seasonal Trends:

4. **Question:** What are the seasonal fluctuations in Sales Amount? **Answer:** Sales Amount experiences consistent fluctuations across the seasons, with a peak in winter and a slight dip in summer. Winter generates the highest Sales Amount, exceeding 1.2 million.
5. **Question:** How do Sales Quantity changes with different seasons? **Answer:** Sales Quantity follows a similar trend to Sales Amount, reaching its highest point in winter and dropping in summer. The quantity difference between seasons is significant.
6. **Question:** Is there a pattern in Sales Margin Amount across different seasons? **Answer:** Sales Margin Amount maintains a relatively consistent pattern across different seasons, with margins hovering between 310 and 320.

2.3 Yearly Trends:

7. **Question:** How has Sales Amount evolved over the years? **Answer:** Sales Amount shows a downward trend from 2017 to 2019, with the highest in 2017 (1.78 million) and the lowest in 2019 (1.6 million). Data is missing from April 2018 to September 2018.
8. **Question:** Are there any significant changes in Sales Quantity on a yearly basis? **Answer:** Sales Quantity has seen a decline over the years. In 2017, it reached 122,000, while in 2019, it dropped to 118,000. However, 2018 has a significant drop (25,000) due to missing data.
9. **Question:** What is the trend in Sales Margin Amount over the years? **Answer:** Sales Margin Amount displays a similar downward trend to Sales Amount, decreasing from 328 in 2017 to 310 in 2019. However, the values in the missing period (2018) are likely incorrect.

2.4 Month-Yearly Trends:

10. **Question:** What are the monthly trends of Sales Amount over the years? **Answer:** Similar to the monthly trend of Sales Amount, the month-yearly trend shows the data gap in April to September 2018 and reveals consistent monthly variations in the other years.

11. **Question:** How does Sales Quantity change in specific months across different years? **Answer:** Changes in Sales Quantity in Specific Months Across Different Years indicate higher variations in 2017, while 2019 shows relatively stable patterns. Missing data in 2018 makes comparison difficult.

12. **Question:** Are there any specific months that stand out in terms of Sales Margin Amount? **Answer:** Monthly Trends of Sales Margin Amount highlight certain months with higher margins, such as August 2017 (348.54), January 2017 (353.06), and March 2017 (358.69).

2.5 Seasonal-Yearly Trends:

13. **Question:** How do sales trends vary across different seasons for each year? **Answer:** Sales Trends Across Different Seasons for Each Year show Fall as the highest-generating season, followed by Winter in 2017. In 2019, Fall and Winter generate nearly equal sales.

14. **Question:** Are there any consistent patterns in Sales Quantity across different seasons and years? **Answer:** Sales Quantity Patterns Across Different Seasons and Years resemble the trend of sales amount across seasons and years, indicating consistent patterns.

15. **Question:** What is the yearly trend in Sales Margin Amount for each season? **Answer:** Yearly Trend in Sales Margin Amount for Each Season reveals that Spring consistently has the lowest margins in both 2017 and 2019. Fall and Winter show varying margins across the years.

2.6 Item Analysis:

16. **Question:** Which items have the highest Sales Amount overall? **Answer:** The top 100 items with the highest Sales Amount exhibit a decreasing exponential trend, with the top 20 items contributing to around 70% of total sales amount.

17. **Question:** Are there any items with varying Sales Quantity throughout the dataset? **Answer:** Top 100 Items with Varying Sales Quantity Throughout the Dataset exhibit a linear decline in sales quantity, indicating items with declining popularity.

18. **Question:** How does the Sales Margin Amount vary between different items? **Answer:** Sales Margin Amount Variation Between Different Items (Top 100) follows a trend similar to Sales Quantity, suggesting declining margins.

2.7 Sales Rep Analysis:

19. **Question:** Who are the top sales representatives in terms of generating Sales Amount? **Answer:** Top Sales Representatives in Terms of Generating Sales Amount show a significant gap between the top sales representative with ID 108 and others, indicating their influence on overall sales.

20. **Question:** Are there differences in Sales Quantity achieved by different sales reps? **Answer:** Differences in Sales Quantity Achieved by Different Sales Reps reveal variations, with sales quantity dropping significantly after the top sales representative with ID 108.

21. **Question:** How does Sales Margin Amount differ among different sales representatives? **Answer:** Differences in Sales Margin Amount Among Different Sales Representatives show a range of margins, with the top sales representative having higher margins and others showing a decreasing trend.

2.8 Price Analysis:

22. **Question:** How does the List Price correlate with the Sales Amount for various items? **Answer:** Correlation between List Price and Sales Amount shows a dense cluster of data points at lower list prices and sales amounts, suggesting a generally linear relationship.

23. **Question:** Are there instances where a higher List Price results in higher Sales Quantity? **Answer:** Correlation between List Price and Sales Quantity shows that higher list prices do not necessarily lead to higher sales quantity. Most sales occur in the lower price range.

24. **Question:** What is the relationship between Sales Price and Sales Amount? **Answer:** Relationship between Sales Price and Sales Amount displays a complex pattern, with multiple lines forming a box, indicating variability in sales amount across different sales prices.

2.9 Combined Analysis:

25. **Question:** How do Sales Amount and Sales Quantity correlate on a monthly basis? **Answer:** Correlation between Sales Amount and Sales Quantity on a Monthly Basis indicates no strong relationship between the two variables, with sales quantity mostly hovering near zero.

26. **Question:** Are there any seasonal trends in Sales Margin Amount that align with specific items? **Answer:** The random grouping of five items in each of the twelve charts does not reveal consistent seasonal trends in Sales Margin Amount.

27. **Question:** What is the overall trend of Sales Amount when considering both the sales rep and the item? **Answer:** The grouping of the top 120 items into 24 charts depicts individual item sales trends, showcasing the variation in sales amount over time for each item.

2.10 Comparison Analysis:

28. **Question:** How does the Sales Amount of a specific item compare to the overall Sales Amount trend? **Answer:** The grouped bar chart for each specific item's Sales Amount compared to the overall trend provides insight into how individual items contribute to the overall sales trend.

29. **Question:** Are there any significant differences in Sales Quantity between different seasons for a particular item? **Answer:** The 60 grouped bar charts with Sales Quantity across different seasons for the top 60 items show variations, highlighting specific seasons where certain items perform better.

30. **Question:** How does the Sales Margin Amount of a particular item compare across different sales representatives? **Answer:** The 60 grouped bar charts depicting Sales Margin Amount

for each item across different sales representatives reveal variations, helping to understand the impact of sales reps on item margins.

These detailed summaries provide a comprehensive overview of each question's context, the observed insights from the charts, and the patterns and trends within the dataset. If you have any further questions or need additional analysis, feel free to ask!