Sales Analysis

Import necessary libralies

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

Task 1: Merge 12 Months of Sales Data Into Single CSV

In [2]:
```
data = pd.read_csv("C:/Users/ZJU/Desktop/Thesis Projects Norman + Mongolia/Pandas-Data
```

In [3]:
```
data.head()
```

Out[3]:

	Order ID	Product	Quantity Ordered	Price Each	Order Date	Purchase Address
0	176558	USB-C Charging Cable	2	11.95	04/19/19 08:46	917 1st St, Dallas, TX 75001
1	NaN	NaN	NaN	NaN	NaN	NaN
2	176559	Bose SoundSport Headphones	1	99.99	04/07/19 22:30	682 Chestnut St, Boston, MA 02215
3	176560	Google Phone	1	600	04/12/19 14:38	669 Spruce St, Los Angeles, CA 90001
4	176560	Wired Headphones	1	11.99	04/12/19 14:38	669 Spruce St, Los Angeles, CA 90001

In [4]:
```
data_1 = pd.read_csv("C:/Users/ZJU/Desktop/Thesis Projects Norman + Mongolia/Pandas-Da
data_2  = pd.read_csv("C:/Users/ZJU/Desktop/Thesis Projects Norman + Mongolia/Pandas-D
data_3  = pd.read_csv("C:/Users/ZJU/Desktop/Thesis Projects Norman + Mongolia/Pandas-D
data_4  = pd.read_csv("C:/Users/ZJU/Desktop/Thesis Projects Norman + Mongolia/Pandas-D
data_5  = pd.read_csv("C:/Users/ZJU/Desktop/Thesis Projects Norman + Mongolia/Pandas-D
data_6  = pd.read_csv("C:/Users/ZJU/Desktop/Thesis Projects Norman + Mongolia/Pandas-D
data_7  = pd.read_csv("C:/Users/ZJU/Desktop/Thesis Projects Norman + Mongolia/Pandas-D
data_8  = pd.read_csv("C:/Users/ZJU/Desktop/Thesis Projects Norman + Mongolia/Pandas-D
data_9  = pd.read_csv("C:/Users/ZJU/Desktop/Thesis Projects Norman + Mongolia/Pandas-D
data_10  = pd.read_csv("C:/Users/ZJU/Desktop/Thesis Projects Norman + Mongolia/Pandas-
data_11  = pd.read_csv("C:/Users/ZJU/Desktop/Thesis Projects Norman + Mongolia/Pandas-
data_12  = pd.read_csv("C:/Users/ZJU/Desktop/Thesis Projects Norman + Mongolia/Pandas-
```

In [5]:
```
dfs = [data_1, data_2, data_3, data_4, data_5, data_6, data_7, data_8, data_9, data_1
merged_df = dfs[0]
for data in dfs[1:]:
    merged_df = merged_df.merge(data, on='Order ID')
    merged_df = pd.concat(dfs, ignore_index=True)
```

```
# View the first few rows of the merged DataFrame
merged_df.head()
```

Out[5]:

	Order ID	Product	Quantity Ordered	Price Each	Order Date	Purchase Address
0	176558	USB-C Charging Cable	2	11.95	04/19/19 08:46	917 1st St, Dallas, TX 75001
1	NaN	NaN	NaN	NaN	NaN	NaN
2	176559	Bose SoundSport Headphones	1	99.99	04/07/19 22:30	682 Chestnut St, Boston, MA 02215
3	176560	Google Phone	1	600	04/12/19 14:38	669 Spruce St, Los Angeles, CA 90001
4	176560	Wired Headphones	1	11.99	04/12/19 14:38	669 Spruce St, Los Angeles, CA 90001

In [6]:
```
merged_df.tail()
```

Out[6]:

	Order ID	Product	Quantity Ordered	Price Each	Order Date	Purchase Address
186845	259353	AAA Batteries (4-pack)	3	2.99	09/17/19 20:56	840 Highland St, Los Angeles, CA 90001
186846	259354	iPhone	1	700	09/01/19 16:00	216 Dogwood St, San Francisco, CA 94016
186847	259355	iPhone	1	700	09/23/19 07:39	220 12th St, San Francisco, CA 94016
186848	259356	34in Ultrawide Monitor	1	379.99	09/19/19 17:30	511 Forest St, San Francisco, CA 94016
186849	259357	USB-C Charging Cable	1	11.95	09/30/19 00:18	250 Meadow St, San Francisco, CA 94016

Step 2 To merge multiple datasets

In [7]:
```
import os
import pandas as pd

# Specify the directory containing the CSV files
directory = "C:/Users/ZJU/Desktop/Thesis Projects Norman + Mongolia/Pandas-Data-Scienc

# List all CSV files in the directory
files = [file for file in os.listdir(directory) if file.endswith('.csv')]

# Create an empty DataFrame to store all the data
all_data_sets = pd.DataFrame()

# Loop through each file and concatenate the data
for file in files:

    file_path = os.path.join(directory, file)  # Construct the full file path as a str
```

```
    data = pd.read_csv(file_path)

    all_data_sets = pd.concat([all_data_sets, data], ignore_index=True)

#emerged datasets was saved in the directory folder called output

    all_data_sets.to_csv("all_data.csv",index=False)

# View the first few rows of the merged DataFrame

all_data_sets.head(5)
```

Out[7]:

	Order ID	Product	Quantity Ordered	Price Each	Order Date	Purchase Address
0	176558	USB-C Charging Cable	2	11.95	04/19/19 08:46	917 1st St, Dallas, TX 75001
1	NaN	NaN	NaN	NaN	NaN	NaN
2	176559	Bose SoundSport Headphones	1	99.99	04/07/19 22:30	682 Chestnut St, Boston, MA 02215
3	176560	Google Phone	1	600	04/12/19 14:38	669 Spruce St, Los Angeles, CA 90001
4	176560	Wired Headphones	1	11.99	04/12/19 14:38	669 Spruce St, Los Angeles, CA 90001

In [8]:
```
all_data_sets.tail()
```

Out[8]:

	Order ID	Product	Quantity Ordered	Price Each	Order Date	Purchase Address
186845	259353	AAA Batteries (4-pack)	3	2.99	09/17/19 20:56	840 Highland St, Los Angeles, CA 90001
186846	259354	iPhone	1	700	09/01/19 16:00	216 Dogwood St, San Francisco, CA 94016
186847	259355	iPhone	1	700	09/23/19 07:39	220 12th St, San Francisco, CA 94016
186848	259356	34in Ultrawide Monitor	1	379.99	09/19/19 17:30	511 Forest St, San Francisco, CA 94016
186849	259357	USB-C Charging Cable	1	11.95	09/30/19 00:18	250 Meadow St, San Francisco, CA 94016

Read In Update DataFrame

In [9]:
```
all_data = pd.read_csv("C:/Users\ZJU\Desktop/Thesis Projects Norman + Mongolia/Pandas-
all_data.head()
```

Out[9]:

	Order ID	Product	Quantity Ordered	Price Each	Order Date	Purchase Address
0	176558	USB-C Charging Cable	2	11.95	04/19/19 08:46	917 1st St, Dallas, TX 75001
1	NaN	NaN	NaN	NaN	NaN	NaN
2	176559	Bose SoundSport Headphones	1	99.99	04/07/19 22:30	682 Chestnut St, Boston, MA 02215
3	176560	Google Phone	1	600	04/12/19 14:38	669 Spruce St, Los Angeles, CA 90001
4	176560	Wired Headphones	1	11.99	04/12/19 14:38	669 Spruce St, Los Angeles, CA 90001

In [10]:
```
all_data.tail()
```

Out[10]:

	Order ID	Product	Quantity Ordered	Price Each	Order Date	Purchase Address
186845	259353	AAA Batteries (4-pack)	3	2.99	09/17/19 20:56	840 Highland St, Los Angeles, CA 90001
186846	259354	iPhone	1	700	09/01/19 16:00	216 Dogwood St, San Francisco, CA 94016
186847	259355	iPhone	1	700	09/23/19 07:39	220 12th St, San Francisco, CA 94016
186848	259356	34in Ultrawide Monitor	1	379.99	09/19/19 17:30	511 Forest St, San Francisco, CA 94016
186849	259357	USB-C Charging Cable	1	11.95	09/30/19 00:18	250 Meadow St, San Francisco, CA 94016

Clean Up Data

Drop Rows with NaN

In [11]:
```
nan_df = all_data[all_data.isna().any(axis=1)]

nan_df.head()

# Drop rows with NaN values

all_data = all_data.dropna(how='any')

all_data.head()
```

Out[11]:

	Order ID	Product	Quantity Ordered	Price Each	Order Date	Purchase Address
0	176558	USB-C Charging Cable	2	11.95	04/19/19 08:46	917 1st St, Dallas, TX 75001
2	176559	Bose SoundSport Headphones	1	99.99	04/07/19 22:30	682 Chestnut St, Boston, MA 02215
3	176560	Google Phone	1	600	04/12/19 14:38	669 Spruce St, Los Angeles, CA 90001
4	176560	Wired Headphones	1	11.99	04/12/19 14:38	669 Spruce St, Los Angeles, CA 90001
5	176561	Wired Headphones	1	11.99	04/30/19 09:27	333 8th St, Los Angeles, CA 90001

Cleaned _Data

In [12]:
```
all_data.head()
```

Out[12]:

	Order ID	Product	Quantity Ordered	Price Each	Order Date	Purchase Address
0	176558	USB-C Charging Cable	2	11.95	04/19/19 08:46	917 1st St, Dallas, TX 75001
2	176559	Bose SoundSport Headphones	1	99.99	04/07/19 22:30	682 Chestnut St, Boston, MA 02215
3	176560	Google Phone	1	600	04/12/19 14:38	669 Spruce St, Los Angeles, CA 90001
4	176560	Wired Headphones	1	11.99	04/12/19 14:38	669 Spruce St, Los Angeles, CA 90001
5	176561	Wired Headphones	1	11.99	04/30/19 09:27	333 8th St, Los Angeles, CA 90001

Augmented data with additional Columns

TASK 2 : Add Month Colum

Fingd OR Delete it

In [13]:
```
all_data = all_data [all_data ['Order Date'].str[0:2] != 'OR']
all_data.head()
```

Out[13]:

	Order ID	Product	Quantity Ordered	Price Each	Order Date	Purchase Address
0	176558	USB-C Charging Cable	2	11.95	04/19/19 08:46	917 1st St, Dallas, TX 75001
2	176559	Bose SoundSport Headphones	1	99.99	04/07/19 22:30	682 Chestnut St, Boston, MA 02215
3	176560	Google Phone	1	600	04/12/19 14:38	669 Spruce St, Los Angeles, CA 90001
4	176560	Wired Headphones	1	11.99	04/12/19 14:38	669 Spruce St, Los Angeles, CA 90001
5	176561	Wired Headphones	1	11.99	04/30/19 09:27	333 8th St, Los Angeles, CA 90001

In [14]:
```
print(all_data.columns)
```

```
Index(['Order ID', 'Product', 'Quantity Ordered', 'Price Each', 'Order Date',
       'Purchase Address'],
      dtype='object')
```

In [15]:
```
# Exclude rows with 'OR' in the 'Order Date' column
all_data = all_data[~all_data['Order Date'].str.startswith('OR')]

# Create the 'Month' column
all_data['Month'] = all_data['Order Date'].str[0:2]

# Convert 'Month' column to integers, ignoring any non-numeric values
all_data['Month'] = pd.to_numeric(all_data['Month'], errors='coerce')

all_data.head()
```

Out[15]:

	Order ID	Product	Quantity Ordered	Price Each	Order Date	Purchase Address	Month
0	176558	USB-C Charging Cable	2	11.95	04/19/19 08:46	917 1st St, Dallas, TX 75001	4.0
2	176559	Bose SoundSport Headphones	1	99.99	04/07/19 22:30	682 Chestnut St, Boston, MA 02215	4.0
3	176560	Google Phone	1	600	04/12/19 14:38	669 Spruce St, Los Angeles, CA 90001	4.0
4	176560	Wired Headphones	1	11.99	04/12/19 14:38	669 Spruce St, Los Angeles, CA 90001	4.0
5	176561	Wired Headphones	1	11.99	04/30/19 09:27	333 8th St, Los Angeles, CA 90001	4.0

Convert Column into correct datatype

In [16]:
```
all_data['Quantity Ordered'] = pd.to_numeric(all_data['Quantity Ordered'], errors='coe

all_data['Price Each'] = pd.to_numeric(all_data['Price Each'], errors='coerce')
```

```
all_data.head()
```

Out[16]:

	Order ID	Product	Quantity Ordered	Price Each	Order Date	Purchase Address	Month
0	176558	USB-C Charging Cable	2.0	11.95	04/19/19 08:46	917 1st St, Dallas, TX 75001	4.0
2	176559	Bose SoundSport Headphones	1.0	99.99	04/07/19 22:30	682 Chestnut St, Boston, MA 02215	4.0
3	176560	Google Phone	1.0	600.00	04/12/19 14:38	669 Spruce St, Los Angeles, CA 90001	4.0
4	176560	Wired Headphones	1.0	11.99	04/12/19 14:38	669 Spruce St, Los Angeles, CA 90001	4.0
5	176561	Wired Headphones	1.0	11.99	04/30/19 09:27	333 8th St, Los Angeles, CA 90001	4.0

In [17]:
```
print(all_data.columns)
```

```
Index(['Order ID', 'Product', 'Quantity Ordered', 'Price Each', 'Order Date',
       'Purchase Address', 'Month'],
      dtype='object')
```

In [18]:
```
#using apply methods to run any function

def get_city(Address):

    return Address.split(',')[1:]

all_data ['City']= all_data['Purchase Address'].apply(lambda x: get_city(x))

all_data.head()
```

Out[18]:

	Order ID	Product	Quantity Ordered	Price Each	Order Date	Purchase Address	Month	City
0	176558	USB-C Charging Cable	2.0	11.95	04/19/19 08:46	917 1st St, Dallas, TX 75001	4.0	[Dallas, TX 75001]
2	176559	Bose SoundSport Headphones	1.0	99.99	04/07/19 22:30	682 Chestnut St, Boston, MA 02215	4.0	[Boston, MA 02215]
3	176560	Google Phone	1.0	600.00	04/12/19 14:38	669 Spruce St, Los Angeles, CA 90001	4.0	[Los Angeles, CA 90001]
4	176560	Wired Headphones	1.0	11.99	04/12/19 14:38	669 Spruce St, Los Angeles, CA 90001	4.0	[Los Angeles, CA 90001]
5	176561	Wired Headphones	1.0	11.99	04/30/19 09:27	333 8th St, Los Angeles, CA 90001	4.0	[Los Angeles, CA 90001]

In [19]:
```
print(all_data.columns)
```

```
Index(['Order ID', 'Product', 'Quantity Ordered', 'Price Each', 'Order Date',
       'Purchase Address', 'Month', 'City'],
      dtype='object')
```

In [20]:
```
def get_city(address):

    split_address = address.split(',')

    if len(split_address) >= 2:

        return split_address[1].strip()

    else:

        return ''

all_data['City'] = all_data['Purchase Address'].apply(lambda x: get_city(x))

all_data.head()

#In this updated code, I've added a condition if len(split_address) >= 2
#to check if the split result has at least two elements. If it does,
#it retrieves the second element using split_address[1], which corresponds to the city
#If the split result does not have enough elements, it returns an empty string '' as t
```

Out[20]:

	Order ID	Product	Quantity Ordered	Price Each	Order Date	Purchase Address	Month	City
0	176558	USB-C Charging Cable	2.0	11.95	04/19/19 08:46	917 1st St, Dallas, TX 75001	4.0	Dallas
2	176559	Bose SoundSport Headphones	1.0	99.99	04/07/19 22:30	682 Chestnut St, Boston, MA 02215	4.0	Boston
3	176560	Google Phone	1.0	600.00	04/12/19 14:38	669 Spruce St, Los Angeles, CA 90001	4.0	Los Angeles
4	176560	Wired Headphones	1.0	11.99	04/12/19 14:38	669 Spruce St, Los Angeles, CA 90001	4.0	Los Angeles
5	176561	Wired Headphones	1.0	11.99	04/30/19 09:27	333 8th St, Los Angeles, CA 90001	4.0	Los Angeles

In [21]:
```
def get_city(address):

    split_address = address.split(',')

    if len(split_address) >= 2:

        return split_address[1].strip()

    else:

        return ''

all_data['City'] = all_data['Purchase Address'].apply(lambda x: get_city(x))
```

```
all_data.head()
```

Out[21]:

	Order ID	Product	Quantity Ordered	Price Each	Order Date	Purchase Address	Month	City
0	176558	USB-C Charging Cable	2.0	11.95	04/19/19 08:46	917 1st St, Dallas, TX 75001	4.0	Dallas
2	176559	Bose SoundSport Headphones	1.0	99.99	04/07/19 22:30	682 Chestnut St, Boston, MA 02215	4.0	Boston
3	176560	Google Phone	1.0	600.00	04/12/19 14:38	669 Spruce St, Los Angeles, CA 90001	4.0	Los Angeles
4	176560	Wired Headphones	1.0	11.99	04/12/19 14:38	669 Spruce St, Los Angeles, CA 90001	4.0	Los Angeles
5	176561	Wired Headphones	1.0	11.99	04/30/19 09:27	333 8th St, Los Angeles, CA 90001	4.0	Los Angeles

Task 4: Add city column

In [22]:

```
def get_city(address):
    split_address = address.split(',')

    if len(split_address) >= 3:
        city_state = split_address[1:3]
        return ', '.join(city_state).strip()
    else:
        return ''

all_data['City'] = all_data['Purchase Address'].apply(lambda x: get_city(x))

all_data.head()
```

Out[22]:

	Order ID	Product	Quantity Ordered	Price Each	Order Date	Purchase Address	Month	City
0	176558	USB-C Charging Cable	2.0	11.95	04/19/19 08:46	917 1st St, Dallas, TX 75001	4.0	Dallas, TX 75001
2	176559	Bose SoundSport Headphones	1.0	99.99	04/07/19 22:30	682 Chestnut St, Boston, MA 02215	4.0	Boston, MA 02215
3	176560	Google Phone	1.0	600.00	04/12/19 14:38	669 Spruce St, Los Angeles, CA 90001	4.0	Los Angeles, CA 90001
4	176560	Wired Headphones	1.0	11.99	04/12/19 14:38	669 Spruce St, Los Angeles, CA 90001	4.0	Los Angeles, CA 90001
5	176561	Wired Headphones	1.0	11.99	04/30/19 09:27	333 8th St, Los Angeles, CA 90001	4.0	Los Angeles, CA 90001

In [23]:
```
print(all_data.columns)
```

```
Index(['Order ID', 'Product', 'Quantity Ordered', 'Price Each', 'Order Date',
       'Purchase Address', 'Month', 'City'],
      dtype='object')
```

Question 3: What time should we advertizements products to maximize the likelyhood of customers to buying products ?

In [24]:
```
import pandas as pd

all_data['Order Date'] = pd.to_datetime(all_data['Order Date'], errors='coerce')
```

In [25]:
```
# Add hour column
all_data['Hour'] = pd.to_datetime(all_data['Order Date']).dt.hour

all_data['Minute']= pd.to_datetime(all_data['Order Date']).dt.minute

all_data.head()
```

Out[25]:

	Order ID	Product	Quantity Ordered	Price Each	Order Date	Purchase Address	Month	City	Hour	Minute
0	176558	USB-C Charging Cable	2.0	11.95	2019-04-19 08:46:00	917 1st St, Dallas, TX 75001	4.0	Dallas, TX 75001	8.0	46.0
2	176559	Bose SoundSport Headphones	1.0	99.99	2019-04-07 22:30:00	682 Chestnut St, Boston, MA 02215	4.0	Boston, MA 02215	22.0	30.0
3	176560	Google Phone	1.0	600.00	2019-04-12 14:38:00	669 Spruce St, Los Angeles, CA 90001	4.0	Los Angeles, CA 90001	14.0	38.0
4	176560	Wired Headphones	1.0	11.99	2019-04-12 14:38:00	669 Spruce St, Los Angeles, CA 90001	4.0	Los Angeles, CA 90001	14.0	38.0
5	176561	Wired Headphones	1.0	11.99	2019-04-30 09:27:00	333 8th St, Los Angeles, CA 90001	4.0	Los Angeles, CA 90001	9.0	27.0

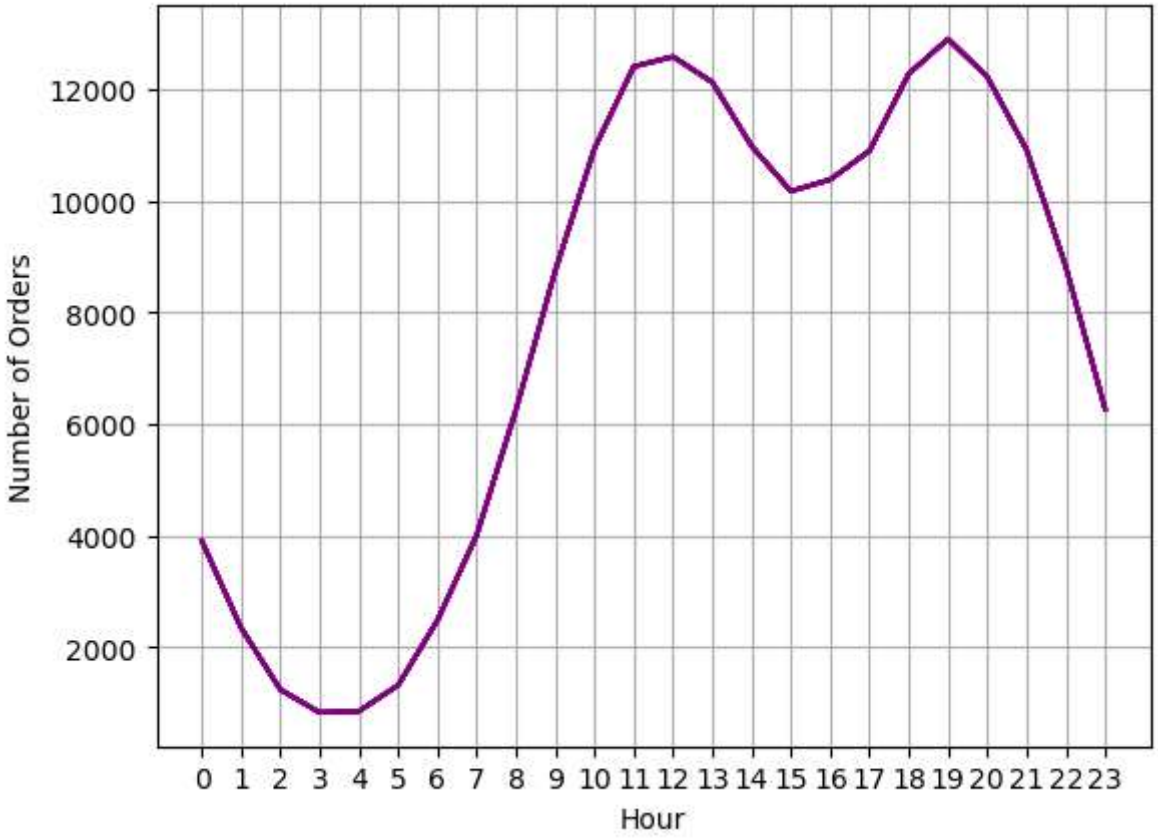
In [26]:
```
print(all_data.columns)
```

```
Index(['Order ID', 'Product', 'Quantity Ordered', 'Price Each', 'Order Date',
       'Purchase Address', 'Month', 'City', 'Hour', 'Minute'],
      dtype='object')
```

In [27]:
```
import matplotlib.pyplot as plt
```

In [28]:
```
best_hour = [hour for hour, df in all_data.groupby(['Hour'])]

plt.plot(best_hour, all_data.groupby(['Hour']).count(), color='purple')
plt.xticks(best_hour)
plt.xlabel('Hour')
plt.ylabel('Number of Orders')
plt.grid()
plt.show()

# My recommendation is slightly before 11am or 7pm
```

```
C:\Users\ZJU\AppData\Local\Temp\ipykernel_18004\1239377100.py:1: FutureWarning: In a future version of pandas, a length 1 tuple will be returned when iterating over a groupby with a grouper equal to a list of length 1. Don't supply a list with a single grouper to avoid this warning.
  best_hour = [hour for hour, df in all_data.groupby(['Hour'])]
```



Question 4: What products are most often sold together

In [29]:
```
all_data.head()
```

Out[29]:

	Order ID	Product	Quantity Ordered	Price Each	Order Date	Purchase Address	Month	City	Hour	Minute
0	176558	USB-C Charging Cable	2.0	11.95	2019-04-19 08:46:00	917 1st St, Dallas, TX 75001	4.0	Dallas, TX 75001	8.0	46.0
2	176559	Bose SoundSport Headphones	1.0	99.99	2019-04-07 22:30:00	682 Chestnut St, Boston, MA 02215	4.0	Boston, MA 02215	22.0	30.0
3	176560	Google Phone	1.0	600.00	2019-04-12 14:38:00	669 Spruce St, Los Angeles, CA 90001	4.0	Los Angeles, CA 90001	14.0	38.0
4	176560	Wired Headphones	1.0	11.99	2019-04-12 14:38:00	669 Spruce St, Los Angeles, CA 90001	4.0	Los Angeles, CA 90001	14.0	38.0
5	176561	Wired Headphones	1.0	11.99	2019-04-30 09:27:00	333 8th St, Los Angeles, CA 90001	4.0	Los Angeles, CA 90001	9.0	27.0

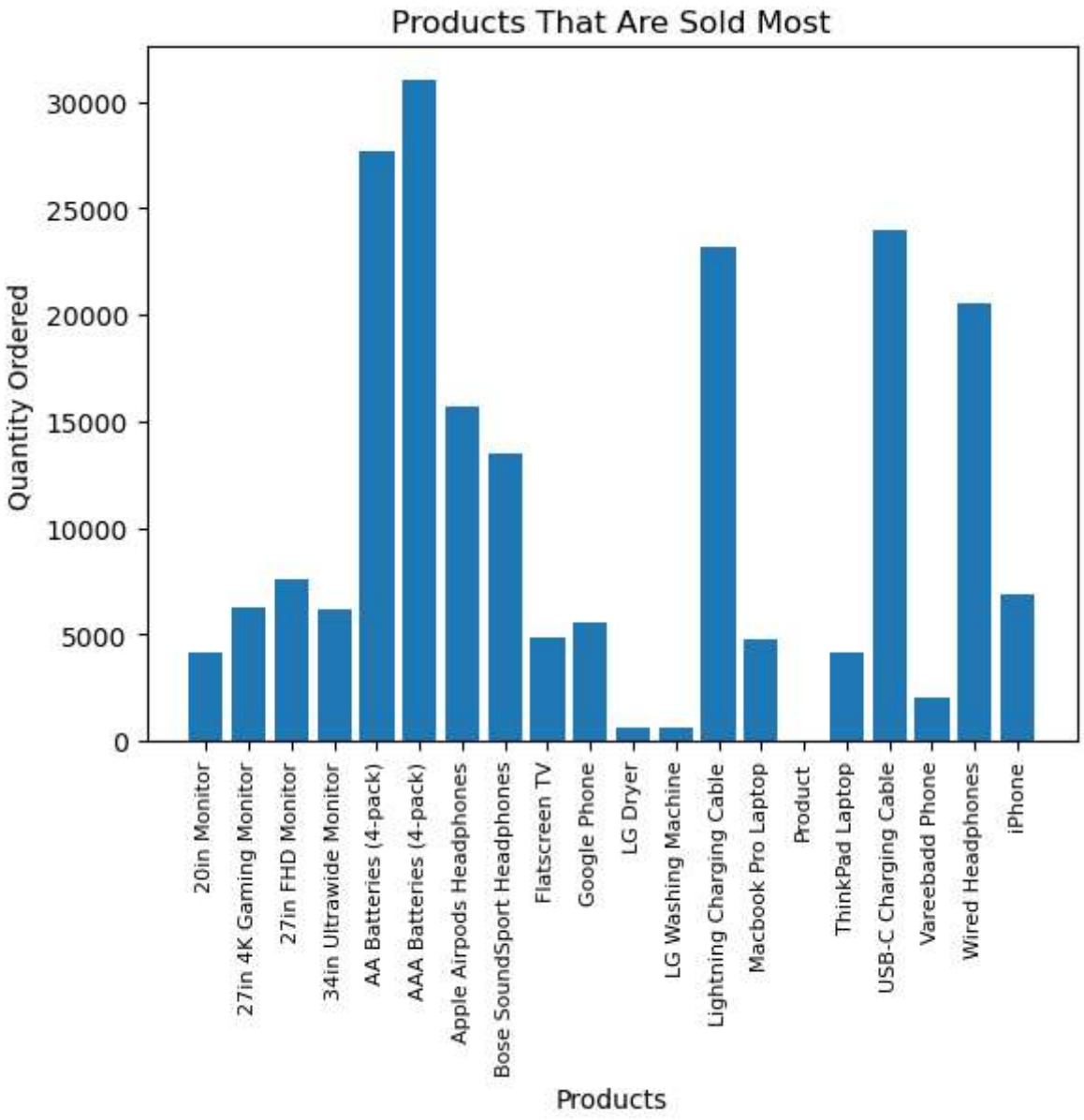
In [30]:

```
import pandas as pd
from itertools import combinations
from collections import Counter

# Assuming you have a DataFrame named 'df' with columns 'Order ID' and 'Product'
# Replace 'df' with the actual name of your DataFrame

# Step 1: Group rows based on 'Order ID' and aggregate products into a list
grouped_products = all_data.groupby('Order ID')['Product'].apply(list)

# Step 2: Create a new DataFrame with 'Order ID' and list of products
product_combinations = pd.DataFrame({'Order ID': grouped_products.index, 'Products': g

# Step 3: Count the occurrence of product combinations
combinations_count = Counter()

for products in product_combinations['Products']:

    combinations_count.update(combinations(products, 3))  # Change the '2' to the numb

# Step 4: Determine the most frequently occurring product combinations
most_common_combinations = combinations_count.most_common(10)  # Change '10' to the de

# Display the most frequently occurring product combinations
for combination, count in most_common_combinations:

    print(f"Products: {', '.join(combination)}, Count: {count}")
```

```
Products: Product, Product, Product, Count: 7393585
Products: Google Phone, USB-C Charging Cable, Wired Headphones, Count: 87
Products: iPhone, Lightning Charging Cable, Wired Headphones, Count: 62
Products: iPhone, Lightning Charging Cable, Apple Airpods Headphones, Count: 47
Products: Google Phone, USB-C Charging Cable, Bose SoundSport Headphones, Count: 35
Products: Vareebadd Phone, USB-C Charging Cable, Wired Headphones, Count: 33
Products: iPhone, Apple Airpods Headphones, Wired Headphones, Count: 27
Products: Google Phone, Bose SoundSport Headphones, Wired Headphones, Count: 24
Products: Vareebadd Phone, USB-C Charging Cable, Bose SoundSport Headphones, Count: 1
6
Products: Vareebadd Phone, Bose SoundSport Headphones, Wired Headphones, Count: 5
```

Question 5: What are product are sold all_data ? and why do you reasons behind leads sold most

In [31]:
```
product_group = all_data.groupby('Product')
Quantity_Ordered = product_group['Quantity Ordered'].sum()
products = [product for product, _ in product_group]

plt.bar(products, Quantity_Ordered)

plt.xticks(products, rotation='vertical', size=8)
plt.xlabel('Products')
plt.ylabel('Quantity Ordered')
plt.title('Products That Are Sold Most')

plt.show()
```

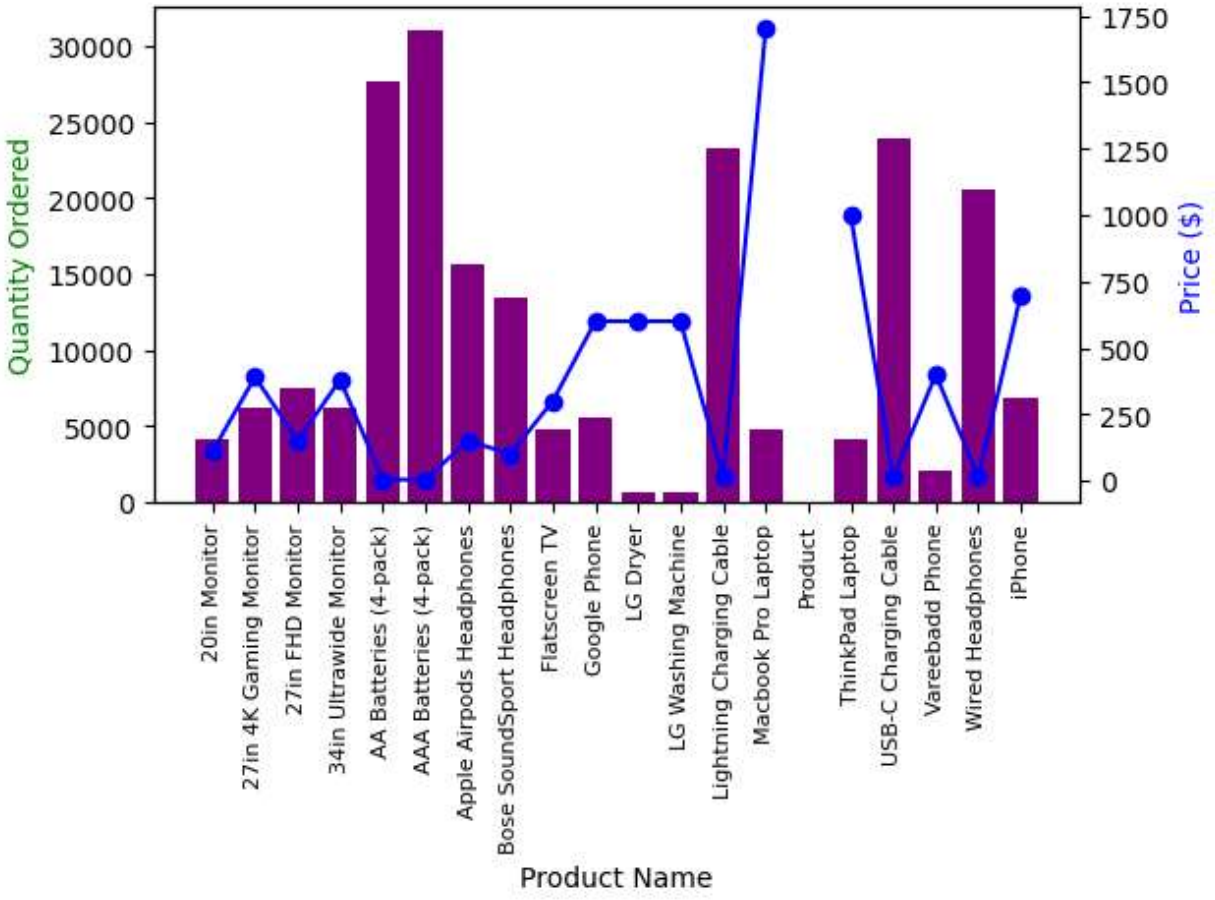


why do AAA Batteries(4-pack) is sold Most than others

In [32]:
```
import pandas as pd
import matplotlib.pyplot as plt

# Assuming you have a DataFrame named 'all_data' with columns 'Product', 'Price Each',
# Replace 'all_data' with the actual name of your DataFrame

# Calculate the average price for each product
product_prices = all_data.groupby('Product').mean()['Price Each']

# Group the data by product and calculate the sum of quantity ordered
product_quantity = all_data.groupby('Product').sum()['Quantity Ordered']

# Get the products and convert them to a list
products = list(product_prices.index)
```

```
# Create a figure and axes
fig, ax1 = plt.subplots()

# Create a twin axes
ax2 = ax1.twinx()

# Plot the bar chart for quantity ordered
ax1.bar(products, product_quantity, color='purple')

# Plot the line graph for average price
ax2.plot(products, product_prices, 'b-',marker='o', linestyle='-', label='Prices')

# Set x-axis tick locations and labels
ax1.set_xticks(range(len(products)))
ax1.set_xticklabels(products, rotation='vertical', fontsize=8)

# Set labels and colors for y-axes
ax1.set_xlabel('Product Name')
ax1.set_ylabel('Quantity Ordered', color='g')
ax2.set_ylabel('Price ($)', color='b')

plt.tight_layout()  # Adjust the spacing between subplots, labels, and titles

plt.show()
```

```
C:\Users\ZJU\AppData\Local\Temp\ipykernel_18004\620898814.py:8: FutureWarning: The de
fault value of numeric_only in DataFrameGroupBy.mean is deprecated. In a future versi
on, numeric_only will default to False. Either specify numeric_only or select only co
lumns which should be valid for the function.
  product_prices = all_data.groupby('Product').mean()['Price Each']
C:\Users\ZJU\AppData\Local\Temp\ipykernel_18004\620898814.py:11: FutureWarning: The d
efault value of numeric_only in DataFrameGroupBy.sum is deprecated. In a future versi
on, numeric_only will default to False. Either specify numeric_only or select only co
lumns which should be valid for the function.
  product_quantity = all_data.groupby('Product').sum()['Quantity Ordered']
C:\Users\ZJU\AppData\Local\Temp\ipykernel_18004\620898814.py:26: UserWarning: linesty
le is redundantly defined by the 'linestyle' keyword argument and the fmt string "b-"
(-> linestyle='-'). The keyword argument will take precedence.
  ax2.plot(products, product_prices, 'b-',marker='o', linestyle='-', label='Prices')
```



In [33]:
```
all_data.head()
```

Out[33]:

	Order ID	Product	Quantity Ordered	Price Each	Order Date	Purchase Address	Month	City	Hour	Minute
0	176558	USB-C Charging Cable	2.0	11.95	2019-04-19 08:46:00	917 1st St, Dallas, TX 75001	4.0	Dallas, TX 75001	8.0	46.0
2	176559	Bose SoundSport Headphones	1.0	99.99	2019-04-07 22:30:00	682 Chestnut St, Boston, MA 02215	4.0	Boston, MA 02215	22.0	30.0
3	176560	Google Phone	1.0	600.00	2019-04-12 14:38:00	669 Spruce St, Los Angeles, CA 90001	4.0	Los Angeles, CA 90001	14.0	38.0
4	176560	Wired Headphones	1.0	11.99	2019-04-12 14:38:00	669 Spruce St, Los Angeles, CA 90001	4.0	Los Angeles, CA 90001	14.0	38.0
5	176561	Wired Headphones	1.0	11.99	2019-04-30 09:27:00	333 8th St, Los Angeles, CA 90001	4.0	Los Angeles, CA 90001	9.0	27.0

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

Out[34]:

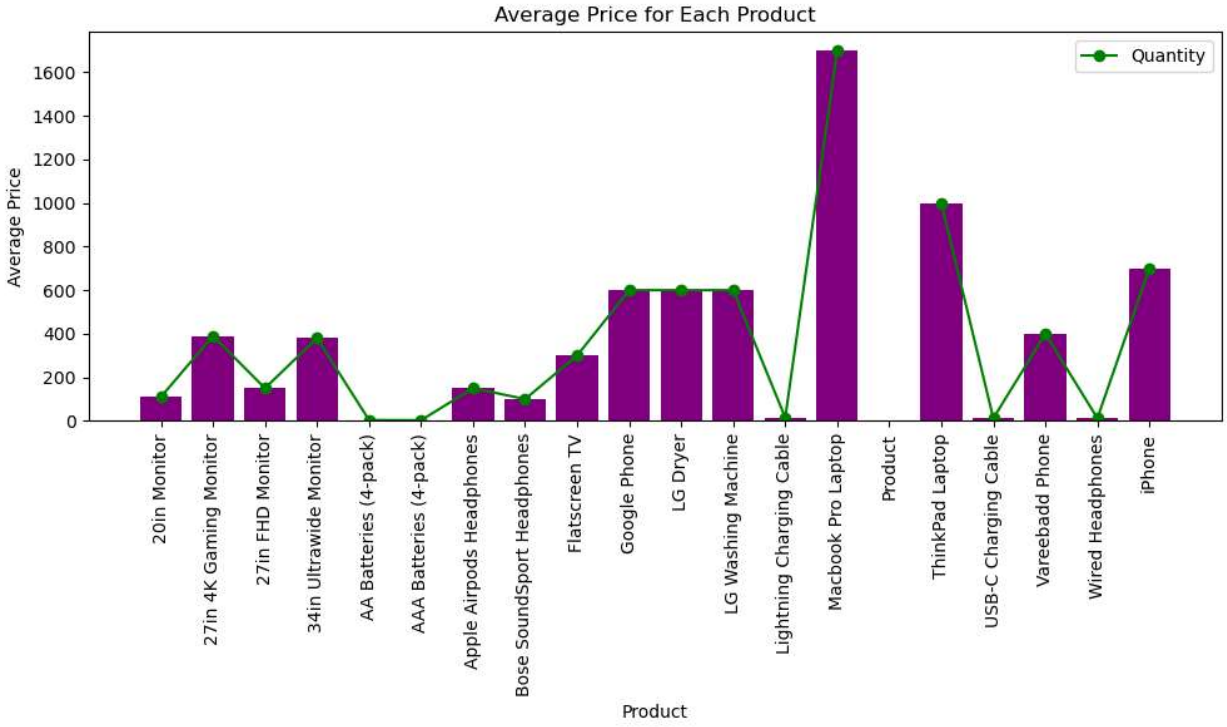
	Order ID	Product	Quantity Ordered	Price Each	Order Date	Purchase Address	Month	City	Hour	Minute
0	176558	USB-C Charging Cable	2.0	11.95	2019-04-19 08:46:00	917 1st St, Dallas, TX 75001	4.0	Dallas, TX 75001	8.0	46.0
2	176559	Bose SoundSport Headphones	1.0	99.99	2019-04-07 22:30:00	682 Chestnut St, Boston, MA 02215	4.0	Boston, MA 02215	22.0	30.0
3	176560	Google Phone	1.0	600.00	2019-04-12 14:38:00	669 Spruce St, Los Angeles, CA 90001	4.0	Los Angeles, CA 90001	14.0	38.0
4	176560	Wired Headphones	1.0	11.99	2019-04-12 14:38:00	669 Spruce St, Los Angeles, CA 90001	4.0	Los Angeles, CA 90001	14.0	38.0
5	176561	Wired Headphones	1.0	11.99	2019-04-30 09:27:00	333 8th St, Los Angeles, CA 90001	4.0	Los Angeles, CA 90001	9.0	27.0

In [35]:
```
product_prices = all_data.groupby('Product')['Price Each'].mean()
print(product_prices)
```

```
Product
20in Monitor                  109.99
27in 4K Gaming Monitor        389.99
27in FHD Monitor              149.99
34in Ultrawide Monitor        379.99
AA Batteries (4-pack)           3.84
AAA Batteries (4-pack)          2.99
Apple Airpods Headphones      150.00
Bose SoundSport Headphones     99.99
Flatscreen TV                 300.00
Google Phone                  600.00
LG Dryer                      600.00
LG Washing Machine            600.00
Lightning Charging Cable       14.95
Macbook Pro Laptop           1700.00
Product                         NaN
ThinkPad Laptop               999.99
USB-C Charging Cable           11.95
Vareebadd Phone               400.00
Wired Headphones               11.99
iPhone                        700.00
Name: Price Each, dtype: float64
```

In [36]:
```
import matplotlib.pyplot as plt

# Assuming you have the 'product_prices' Series with the average prices for each produ
```

```
# Plotting the bar chart
plt.figure(figsize=(10, 6))  # Adjust the figure size as needed
plt.bar(product_prices.index, product_prices.values,color='purple')
plt.plot(product_prices.index, product_prices.values, color='green', marker='o', lines

plt.xticks(rotation='vertical')  # Rotate x-axis labels if needed

plt.xlabel('Product')

plt.ylabel('Average Price')

plt.title('Average Price for Each Product')

plt.legend()  # Display the legend

plt.tight_layout()  # Adjust the spacing between subplots, labels, and titles

plt.show()
```



In [37]:
```
Result= all_data.groupby('City').sum()

Result.head(12)
```

```
C:\Users\ZJU\AppData\Local\Temp\ipykernel_18004\1040525327.py:1: FutureWarning: The d
efault value of numeric_only in DataFrameGroupBy.sum is deprecated. In a future versi
on, numeric_only will default to False. Either specify numeric_only or select only co
lumns which should be valid for the function.
  Result= all_data.groupby('City').sum()
```

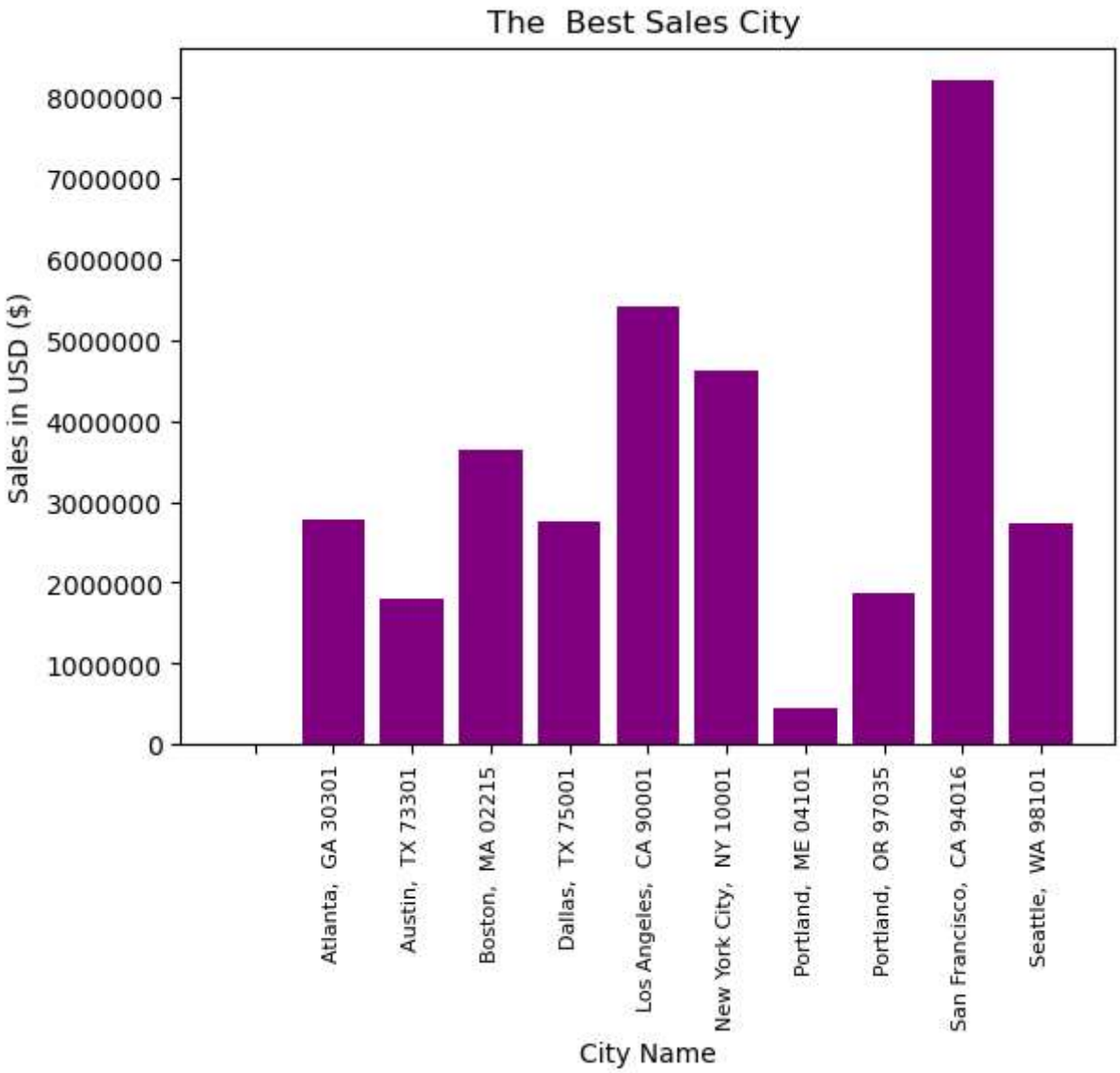
Out[37]:

City	Quantity Ordered	Price Each	Month	Hour	Minute
	0.0	0.00	0.0	0.0	0.0
Atlanta, GA 30301	16602.0	2779908.20	104794.0	214264.0	442932.0
Austin, TX 73301	11153.0	1809873.61	69829.0	141946.0	289060.0
Boston, MA 02215	22528.0	3637409.77	141112.0	288225.0	590442.0
Dallas, TX 75001	16730.0	2752627.82	104620.0	214390.0	435155.0
Los Angeles, CA 90001	33289.0	5421435.23	208325.0	427444.0	866638.0
New York City, NY 10001	27932.0	4635370.83	175741.0	357696.0	733598.0
Portland, ME 04101	2750.0	447189.25	17144.0	35211.0	72856.0
Portland, OR 97035	11303.0	1860558.22	70621.0	144421.0	295533.0
San Francisco, CA 94016	50239.0	8211461.74	315520.0	643265.0	1319477.0
Seattle, WA 98101	16553.0	2733296.01	104941.0	213292.0	436368.0

In [38]:
```
import matplotlib.pyplot as plt
```

Question 2: What a city has the highest number of sales

In [39]:
```
Cities = [city for city , df in all_data.groupby('City')] # Assuming you have 12 month

plt.bar(Cities, Result['Price Each'], color='purple')

plt.ticklabel_format(style='plain', axis='y')  # Display y-axis labels in actual numbe

plt.xticks(Cities,rotation='vertical',size=8)  # Use Months instead of months

plt.ylabel('Sales in USD ($)')

plt.xlabel('City Name')

plt.title('The  Best Sales City ')

plt.show()
```



Data Exploration!

Question 1: What was the best month for sales? How much was earned that month?

Add Sales Column

In [40]:
```
all_data['Quantity Ordered'] = pd.to_numeric(all_data['Quantity Ordered'], errors='coe
all_data['Price Each'] = pd.to_numeric(all_data['Price Each'], errors='coerce')

all_data['Sales'] = all_data['Quantity Ordered'] * all_data['Price Each']

all_data['Sales'] = all_data['Sales'].fillna(0)  # Replace NaN values with 0

all_data.groupby(['Month']).sum()

#I've removed the explicit casting to 'int' and 'float' for the 'Quantity Ordered' and
```

```
#To handle the NaN values in the 'Sales' column, I've added fillna(0) to replace the N
```

```
C:\Users\ZJU\AppData\Local\Temp\ipykernel_18004\1607201213.py:8: FutureWarning: The d
efault value of numeric_only in DataFrameGroupBy.sum is deprecated. In a future versi
on, numeric_only will default to False. Either specify numeric_only or select only co
lumns which should be valid for the function.
  all_data.groupby(['Month']).sum()
```

Out[40]:

Month	Quantity Ordered	Price Each	Hour	Minute	Sales
1.0	10903.0	1811768.38	139485.0	282440.0	1822256.73
2.0	13449.0	2188884.72	172669.0	354885.0	2202022.42
3.0	17005.0	2791207.83	218969.0	447559.0	2807100.38
4.0	20558.0	3367671.02	262259.0	544186.0	3390670.24
5.0	18667.0	3135125.13	238780.0	487899.0	3152606.75
6.0	15253.0	2562025.61	195528.0	402436.0	2577802.26
7.0	16072.0	2632539.56	206169.0	417349.0	2647775.76
8.0	13448.0	2230345.42	172289.0	353857.0	2244467.88
9.0	13109.0	2084992.09	168513.0	341698.0	2097560.13
10.0	22703.0	3715554.83	290650.0	598437.0	3736726.88
11.0	19798.0	3180600.68	254865.0	518231.0	3199603.20
12.0	28114.0	4588415.41	359978.0	733082.0	4613443.34

What is the best sales for a months

Q1. What was the Best a Month for Sales? How much was earned in that Month

In [41]:
```
all_data= all_data.groupby('Month').sum()

all_data.head(15)
```

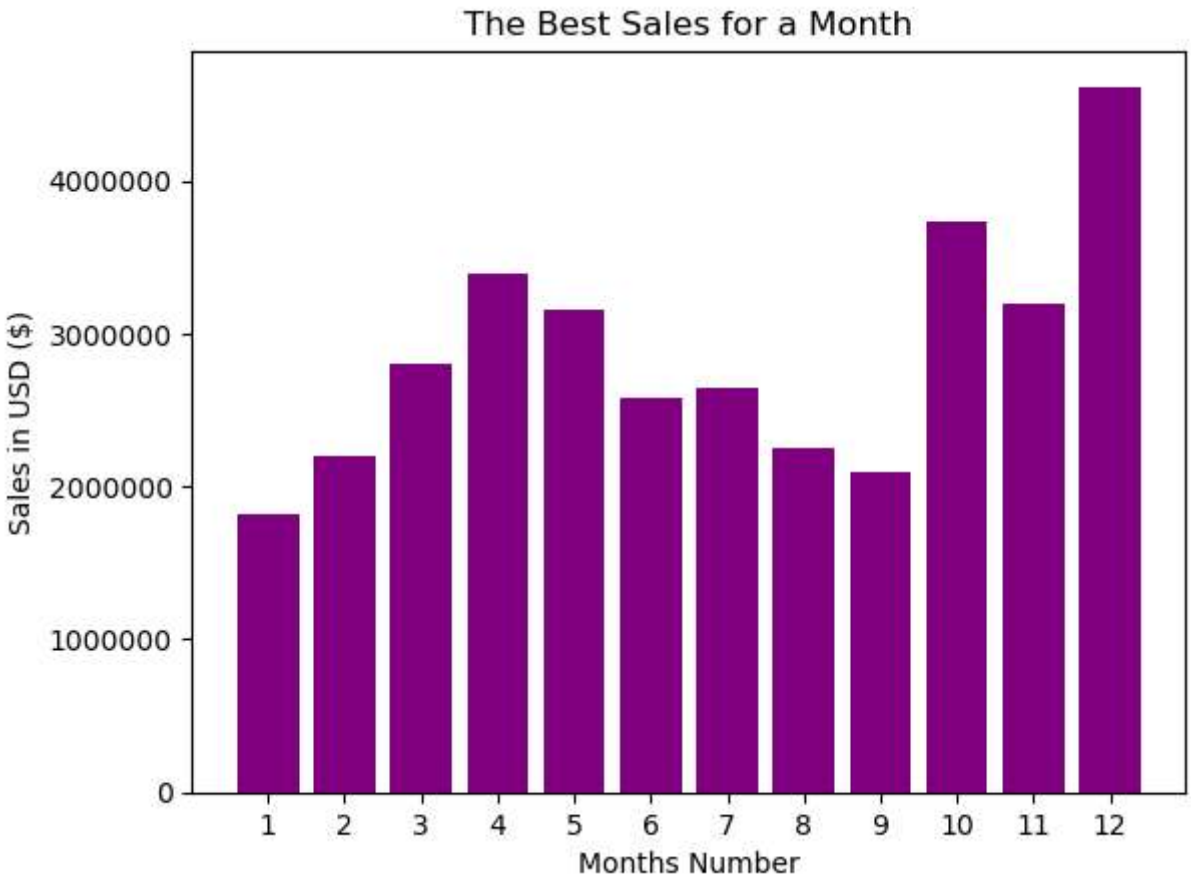
```
C:\Users\ZJU\AppData\Local\Temp\ipykernel_18004\4050269954.py:1: FutureWarning: The d
efault value of numeric_only in DataFrameGroupBy.sum is deprecated. In a future versi
on, numeric_only will default to False. Either specify numeric_only or select only co
lumns which should be valid for the function.
  all_data= all_data.groupby('Month').sum()
```

Out[41]:

Month	Quantity Ordered	Price Each	Hour	Minute	Sales
1.0	10903.0	1811768.38	139485.0	282440.0	1822256.73
2.0	13449.0	2188884.72	172669.0	354885.0	2202022.42
3.0	17005.0	2791207.83	218969.0	447559.0	2807100.38
4.0	20558.0	3367671.02	262259.0	544186.0	3390670.24
5.0	18667.0	3135125.13	238780.0	487899.0	3152606.75
6.0	15253.0	2562025.61	195528.0	402436.0	2577802.26
7.0	16072.0	2632539.56	206169.0	417349.0	2647775.76
8.0	13448.0	2230345.42	172289.0	353857.0	2244467.88
9.0	13109.0	2084992.09	168513.0	341698.0	2097560.13
10.0	22703.0	3715554.83	290650.0	598437.0	3736726.88
11.0	19798.0	3180600.68	254865.0	518231.0	3199603.20
12.0	28114.0	4588415.41	359978.0	733082.0	4613443.34

In [42]:
```
import matplotlib.pyplot as plt
```

In [43]:
```
Months = range(1, 13)  # Assuming you have 12 months of sales data

plt.bar(Months, all_data['Sales'], color='purple')

plt.ticklabel_format(style='plain', axis='y')  # Display y-axis labels in actual numbe

plt.xticks(Months)  # Use Months instead of months

plt.ylabel('Sales in USD ($)')
plt.xlabel('Months Number')
plt.title('The Best Sales for a Month')

plt.show()
```


The Best Sales for a Month
Sales in USD ($)
0
1000000
2000000
3000000
4000000
1
2
3
4
5
6
7
8
9
10
11
12
Months Number