Product Sales Analysis Using Python

TEAM MEMBER

621521104098 : PraveenKumar .A

Phase-5 submission document

Project Title: Product Sales Analysis Using Python

Phase 5: Project Documentation & Submission

Topic:

Outline the project's objective, design thinking process, and development phases. Describe the analysis objectives, data collection process, data visualization using IBM Cognos, and derived actionable insights. Explain how the insights from the analysis can guide inventory management and marketing strategies.

Product Sales Analysis Using Python

To perform a product sales analysis using Python, we'll walk through a basic example of how to analyze sales data, calculate key metrics, and visualize the results. We'll use libraries such as Pandas for data manipulation, Matplotlib for visualization, and NumPy for numerical operations.

Assuming you have sales data in a CSV file named "sales_data.csv" with columns like 'Product', 'Date', 'Revenue', and 'Quantity', here's a step-by-step approach

OVERVIEW

In this post, I use Python Pandas & Python Matplotlib to analyze and answer business questions about 12 months worth of sales data. The data contains hundreds of thousands of electronics store purchases broken down by month, product type, cost, purchase address, etc. The dataset can be downloaded here. In this analysis, I'm using jupyter notebook.


PLAN!

Introduction:

Product sales analysis is a crucial process for businesses to gain insights into their sales performance and customer behavior. It involves the systematic examination of sales data to identify trends, patterns, and key performance indicators that help businesses make informed decisions and improve their overall strategies. This analysis provides valuable information for optimizing product offerings, enhancing marketing efforts, and maximizing profitability.

Problem Statement:

Analyzing 12 months of electronics store sales data to address key business questions, such as understanding monthly revenue trends, identifying top-selling products, and assessing regional sales performance. The goal is to provide actionable insights for optimizing product offerings and marketing strategies.

Design Thinking Approach:

1. **Empathize: Understand the Stakeholders**

    - Identify key stakeholders, such as business owners, marketing teams, and customers.
    - Conduct interviews or surveys to understand their needs, goals, and pain points related to sales data analysis.
    - Gain empathy for their perspectives and challenges in making data-driven decisions.

2. **Define: Problem Statements and Objectives**

- Synthesize the information gathered in the empathy phase to define clear problem statements.
- Formulate specific objectives for the analysis, aligning them with the business goals.
- Prioritize the most pressing questions and issues to address.

3. Ideate: Generate Insights and Solutions

- Brainstorm various analysis approaches and potential visualizations.
- Encourage creativity within the project team to explore different angles and insights.
- Consider innovative ways to present findings that resonate with stakeholders.

4. Prototype: Create Data Visualizations

- Develop initial data visualizations and analysis prototypes using Python, Pandas, and Matplotlib.
- Experiment with different chart types, layouts, and color schemes to effectively convey insights.
- Share these prototypes with stakeholders for feedback and validation.

5. Test: Gather Feedback

- Present the initial analysis to stakeholders and gather their feedback.
- Ensure that the analysis addresses their needs and provides actionable insights.
- Be open to making iterations and improvements based on feedback.

6. Implement: Refine and Finalize

- Refine the analysis and visualizations based on feedback and testing results.
- Create a finalized report or dashboard that provides a clear narrative and actionable recommendations.
- Ensure that the output aligns with the project's objectives and solves the defined problems.

7. Evaluate: Measure Impact

- Implement any recommended changes in business strategies or processes.

- Continuously monitor and measure the impact of these changes on sales and business performance.
- Iterate and refine the analysis as new data becomes available.

Key elements of a product sales analysis include:

1.Data Collection:

Gathering comprehensive sales data, which typically includes information about the products sold, sales revenue, quantity sold, customer demographics, and sales channels.

2.Data Preprocessing:

Cleaning and organizing the data to ensure accuracy and consistency. This step involves handling missing values, removing duplicates, and formatting data for analysis.

3.Key Metrics Calculation:

Determining essential sales metrics such as total revenue, profit margins, customer acquisition costs, and customer lifetime value. These metrics provide a clear overview of the financial health of the business.

4.Product Performance Analysis:

Evaluating the performance of individual products or product categories by analyzing sales volume, revenue generated, and profit margins. This analysis helps identify top-performing and underperforming products.

5.Customer Segmentation:

Segmenting customers based on their demographics, behaviors, and preferences. Understanding different customer groups helps tailor marketing strategies and product offerings to specific audiences.

6.Market Basket Analysis:

Examining customer purchasing patterns to identify which products are frequently bought together. This insight is valuable for cross-selling and bundling strategies.

7.Price Sensitivity Analysis:

Evaluating how changes in product pricing affect sales and revenue. This information guides pricing strategies for maximizing profits.

8.Inventory Management:

Analyzing inventory turnover rates to optimize stock levels and inventory management strategies. This ensures that products are neither overstocked nor out of stock.

9.Sales Channel Effectiveness:

Assessing the performance of different sales channels (e.g., online, offline, partnerships) to allocate resources effectively and reach target markets.

10.Data Visualization:

Creating clear and informative reports and visualizations, such as charts, graphs, and dashboards, to present the analysis findings in an understandable format.

11.Continuous Monitoring and Iteration:

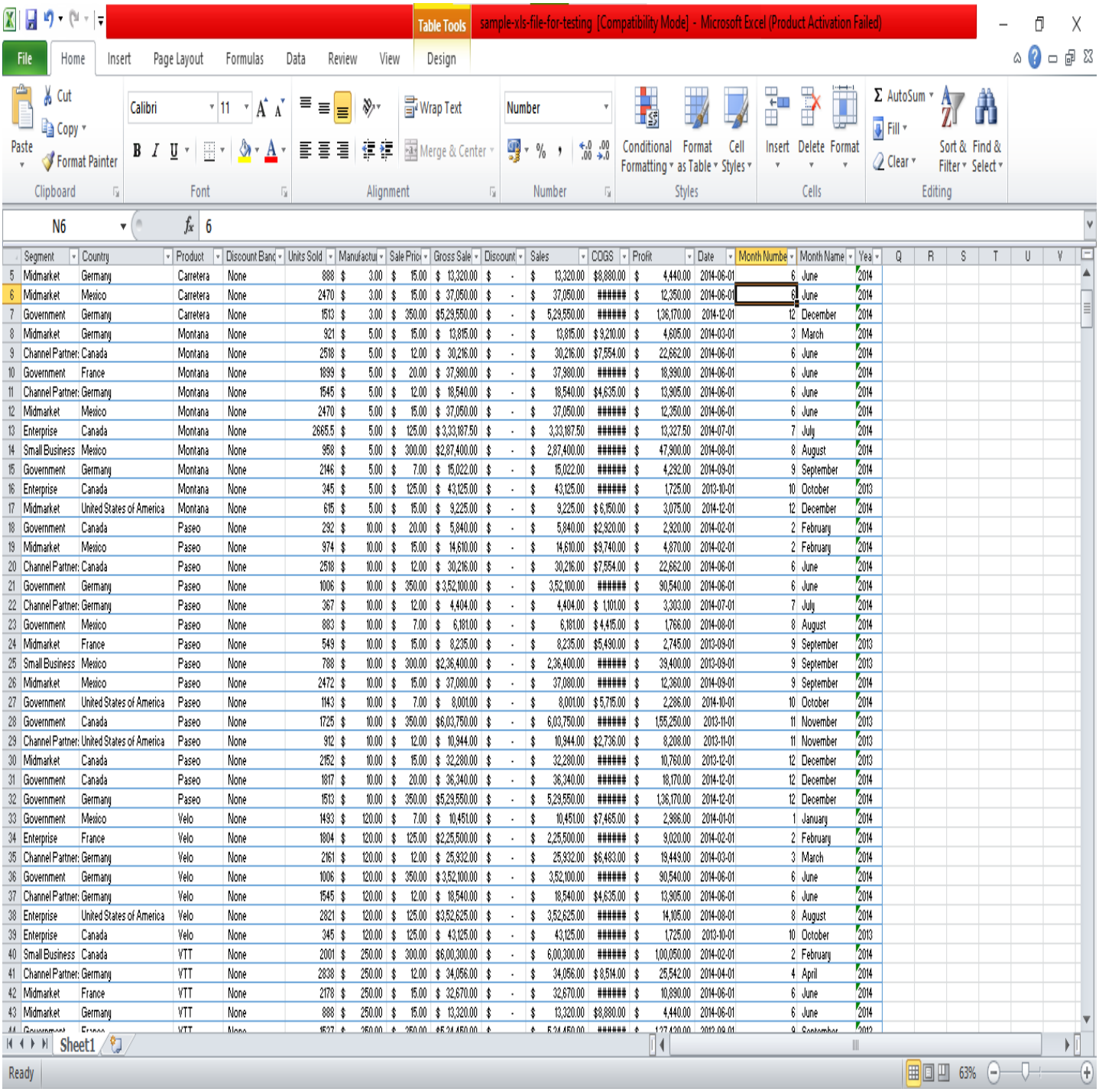
Regularly monitoring sales data and revising strategies based on changing market conditions, customer preferences, and business objectives.
Product sales analysis is a dynamic and ongoing process that empowers businesses to make data-driven decisions. By understanding sales trends, identifying customer preferences, and optimizing product offerings, businesses can enhance their competitiveness and profitability in today's ever-evolving market. It is a fundamental practice for data-informed

decision-making, helping businesses thrive and adapt to changing market dynamics.

Dataset Link:

(https://www.kaggle.com/datasets/ksabishek/product-sales-data)

Sample Data Base:



To conduct a more comprehensive product sales analysis in Python, we'll cover various aspects such as data preprocessing, exploratory data analysis (EDA), key metrics calculation, and visualization. We'll use sample sales data for demonstration purposes.

1. Import Necessary Libraries:

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

2. Load and Explore the Data:

Assuming you have a CSV file named "sales_data.csv" containing relevant sales data.

```
# Load the sales data into a DataFrame
sales_data = pd.read_csv('sales_data.csv')
# Display basic information about the data
print(sales_data.info())
# Display the first few rows of the DataFrame
print(sales_data.head())
```

3. Data Preprocessing:

Ensure the data is in the appropriate format and handle any missing or incorrect values.

```
# Convert the 'Date' column to datetime format
sales_data['Date'] = pd.to_datetime(sales_data['Date'])
# Check for missing values
print('Missing values:\n', sales_data.isnull().sum())
# Drop rows with missing values
sales_data.dropna(inplace=True)
```

4. Key Metrics Calculation:

Calculate key metrics such as total revenue, total quantity sold, and average selling price.

```
# Total revenue
```

```
total_revenue = sales_data['Revenue'].sum()
# Total quantity sold
total_quantity_sold = sales_data['Quantity'].sum()
# Average selling price
average_selling_price = total_revenue / total_quantity_sold
 print('Total Revenue:', total_revenue)
print('Total Quantity Sold:', total_quantity_sold)
print('Average Selling Price:', average_selling_price)
```

5. Exploratory Data Analysis (EDA):

Explore the data to understand the distribution and relationships between variables.

```
# Summary statistics
print(sales_data.describe())
# Visualize the distribution of revenue and quantity sold
plt.figure(figsize=(12, 6))
sns.histplot(sales_data['Revenue'], bins=30, kde=True)
plt.title('Distribution of Revenue')
plt.xlabel('Revenue')
plt.ylabel('Frequency')
plt.show()
plt.figure(figsize=(12, 6))
sns.histplot(sales_data['Quantity'], bins=30, kde=True)
plt.title('Distribution of Quantity Sold')
plt.xlabel('Quantity Sold')
plt.ylabel('Frequency')
plt.show()
```

6. Product Performance Analysis:

Analyze the performance of products based on revenue and quantity sold.

```
# Group data by product and calculate total revenue and total quantity sold for each product
```

```
product_performance = sales_data.groupby('Product').agg
({'Revenue': 'sum', 'Quantity': 'sum'}).reset_index()
# Sort products by revenue in descending order
product_performance =
product_performance.sort_values(by='Revenue', ascending=False)
# Display the top-performing products
print('Top Performing Products:')
print(product_performance.head())
# Visualize top performing products
plt.figure(figsize=(12, 6))
sns.barplot(x='Product', y='Revenue',
data=product_performance.head(10))
plt.xticks(rotation=45)
plt.title('Top Performing Products by Revenue')
plt.xlabel('Product')
plt.ylabel('Total Revenue')
plt.show()
```

You can further extend this analysis to include customer segmentation, market basket analysis, seasonality analysis, and other advanced techniques to derive valuable insights from your sales data. Modify and customize the analysis based on the specific requirements of your dataset and business needs.

Feature Selection :

To create interactive dashboards and reports for your product sales analysis in Python, you can use libraries like Plotly and Dash. Dash is a web application framework for building interactive, web-based data dashboards. It's ideal for creating reports and interactive visualizations. Here's a step-bystep guide to creating a basic interactive product sales dashboard:

1.Install Required Libraries:

If you haven't already, you'll need to install the Dash library and other necessary packages.

```
pip install dash pandas
```

2.Import Libraries and Load Data:

```
import dash from dash
import dcc, html
import pandas as pd
# Load your product sales data
sales_data = pd.read_csv('product_sales_data.csv')
```

3.Initialize the Dash App:

```
app = dash.Dash(__name__)
```

4.Create Layout for the Dashboard:

Define the layout of the dashboard using HTML and Dash components. For example:

```
app.layout = html.Div([
html.H1('Product Sales Analysis Dashboard'),
 dcc.Graph(id='top-products-bar-chart'),
dcc.Graph(id='sales-trends-line-chart'),
dcc.Graph(id='customer-preferences-pie-chart')
])
```

5.Define Callbacks:

Callbacks are functions that specify how the content of the dashboard should change in response to user interactions. For instance, when a user selects a particular product, the dashboard updates to show relevant information.

```
@app.callback(
```

```
dash.dependencies.Output('top-products-bar-chart', 'figure'),
dash.dependencies.Output('sales-trends-line-chart', 'figure'),
dash.dependencies.Output('customer-preferences-pie-chart',
'figure'),
dash.dependencies.Input('product-dropdown', 'value')
)
 def update_charts(selected_product):
        # Write code to update charts based on user input
# For example, filter data and create charts based on the selected
product
```

6.Run the Dash App:

```
if __name__ == '__main__':
        app.run_server(debug=True)
```

This is a simplified example to get you started. You'll need to add the following functionalities:

- Add more details to your dashboard layout.
- Create functions to update the charts based on user interactions.
- Customize the charts to display the desired insights such as top-selling products, sales trends, and customer preferences.

For identifying products with the highest sales, you can create a bar chart showing product sales. For peak sales periods, you can plot a time series chart. For customer preferences, a pie chart or a bar chart could display the distribution of product purchases.

Remember to adapt the code to your specific dataset and analysis goals. Dash offers extensive customization options, and you can create complex dashboards with various components to visualize your product sales analysis effectively.

Data preprocessing is a crucial step in product sales analysis

1. **Data Collection**:

Gather data from various sources, such as sales records, databases, or online platforms.

2. **Data Cleaning**:

Handle missing values by imputing or removing them, Remove duplicates, Correct errors or inconsistencies in the data.

3. **Data Transformation**:

Convert data types (e.g., dates, categorical variables). Normalize or scale numerical features. Create new features if necessary (e.g., calculate total sales per product).

4. **Data Integration:**

Merge data from different sources if needed (e.g., combining sales data with product information).

5. **Feature Engineering**:

Create relevant features like calculating sales per day, week, or month.Use domain knowledge to engineer features that may impact sales.

Machine learning Modeling:

Decision Tree:

A decision tree is a supervised machine learning algorithm that can be used for both classification and regression tasks. It creates a tree-like structure where each internal node represents a feature, each branch represents a decision rule, and each leaf node represents an outcome or class.

Random Forest:

A random forest is an ensemble learning method that builds multiple decision trees during training and combinestheir predictionsto improve accuracy and reduce overfitting. It operates by creating a forest of decision trees, where each tree is trained on a different subset of the data and uses a random selection of features.

Algorithm:

Sure! Decision of trees are a popular algorithm for classification and regression tasks. Random forests are an ensemble method that combines multiple decision treesfor better performance.

EVLOVINH PROBLEMS

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

First of all, we need to see what kind of data we are trying to analyze. we can simply do this

Import Necessary Libraries

```
In [3]: 1 import pandas as pd
```

Look at the first 5 datas

```
In [5]: 1 df = pd.read_csv("D:\Self\Online Course\Solve real world Data science task\Pandas-Data-Science-Tasks-master\Pandas-Data
        2
        3 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

Figure 1. Code to show the top 5 data in Sales_April_2019.csv

We use panda to read the csv file and create a dataframe from it. The file's directory can be put anywhere. I personally put it in D. You can copy the directory and paste it in the syntax. At Figure 1, we can see that we have 6 columns. Now the first task is to merge all 12 months worth of sales data (12 csv files) into a single csv file. To do that, we need to import new library called os.

In [3]:
```
import pandas as pd
import os
```

Task 1: Merging 12 months of sales data into a single csv file.

In [7]:
```
df = pd.read_csv("D:\Self\Online Course\Solve real world Data science task\Pandas-Data-Science-Tasks-master\Pandas-Data

files = [file for file in os.listdir("D:\Self\Online Course\Solve real world Data science task\Pandas-Data-Science-Task
for file in files:
    print(file)
```

```
Sales_April_2019.csv
Sales_August_2019.csv
Sales_December_2019.csv
Sales_February_2019.csv
Sales_January_2019.csv
Sales_July_2019.csv
Sales_June_2019.csv
Sales_March_2019.csv
Sales_May_2019.csv
Sales_November_2019.csv
Sales_October_2019.csv
Sales_September_2019.csv
```

Figure 2. Import new library os.

We need os library to read all csv files' title and call it using *for loop*. As you can see from Figure 2, we successfully read all the csv files' title and we're ready to merge it. To do that, we can simply do

Import Necessary Libraries

In [3]:
```
import pandas as pd
import os
```

Task 1: Merging 12 months of sales data into a single csv file.

In [13]:
```
df = pd.read_csv("D:\Self\Online Course\Solve real world Data science task\Pandas-Data-Science-Tasks-master\Pandas-Data

files = [file for file in os.listdir("D:\Self\Online Course\Solve real world Data science task\Pandas-Data-Science-Task

all_months_data = pd.DataFrame() #Creating empty dataframe called 'all_month_data'

for file in files:
    df = pd.read_csv("D:\Self\Online Course\Solve real world Data science task\Pandas-Data-Science-Tasks-master\Pandas-
    all_months_data = pd.concat([all_months_data, df]) #Merging to the previous empty dataframe

#Checking the result
all_months_data.to_csv("all_data.csv", index=False) #single csv file contain 12 months data.
```

In [ ]:
```

```

Figure 3. Creating a new file contains all 12 months data.

DETAIL CODE :

```
import pandas as pd import os

df = pd.read_csv("D:\Self\Online Course\Solve real world Data science task\Pandas-Data-Science-Tasks-master\Pandas-Data- Science-Tasks-master\SalesAnalysis\Sales_Data\Sales_April_2019.csv")

files = [file for file in os.listdir("D:\Self\Online Course\Solve real world Data science task\Pandas-Data-Science-Tasks- master\Pandas-Data-Science-Tasks- master\SalesAnalysis\Sales_Data")]

all_months_data = pd.DataFrame() #Creating empty dataframe called 'all_month_data'

for file in files:
df = pd.read_csv("D:\Self\Online Course\Solve real world Data science task\Pandas-Data-Science-Tasks-master\Pandas-Data- Science-Tasks-master\SalesAnalysis\Sales_Data/"+file) all_months_data = pd.concat([all_months_data, df]) #Merging to the previous empty dataframe

#Checking the result
all_months_data.to_csv("all_data.csv", index=False) #single csv file contain 12 months data.
```

It will take a little longer time because of heavy computation. But once it's done, you can open the same directory folder and check the folder called "Output". You will see a new csv file contains all 12 months data.

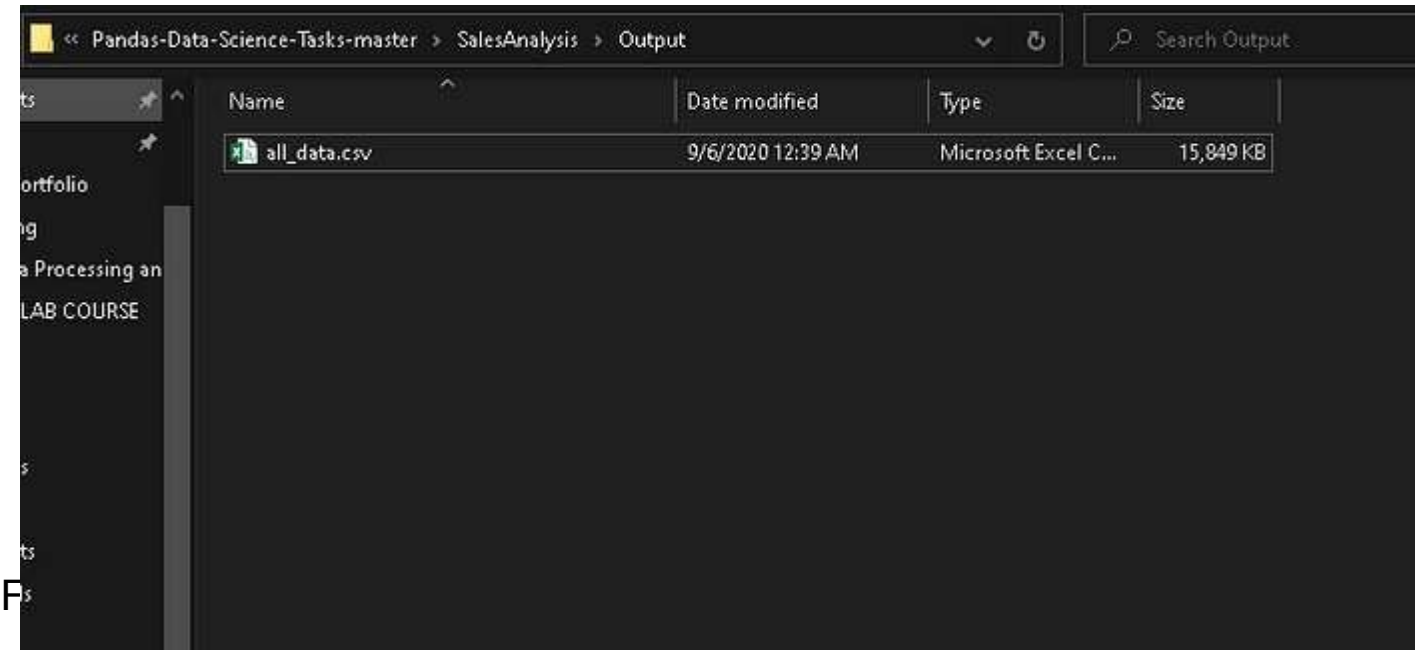


Figure 4. Output folder

After we create this new csv file, you can delete the previous code (if you want) and we will use this file to answer all of the problems.

Now we only use this code to read all of 12 months data.

Reading an updated dataframe

```
In [14]:
1 all_data=pd.read_csv("all_data.csv")
2 all_data.head()
```

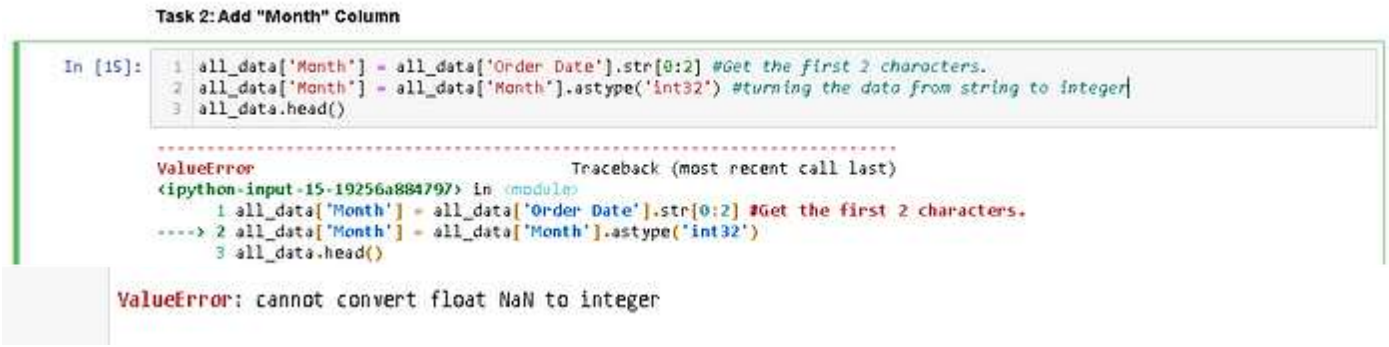
Out[14]:

	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

Figure 5. Reading an updated dataframe

Now we're ready to answer the problem number 1. To remind you, the question is: **What was the best month for sales? How much was earned that month?**

To answer this problem, obviously we need an additional column called "Month". If you look carefully at Figure 5, you will see the first 2 characters in "Order Date" values represent months. So the next task we will do is to add "Month" Column.



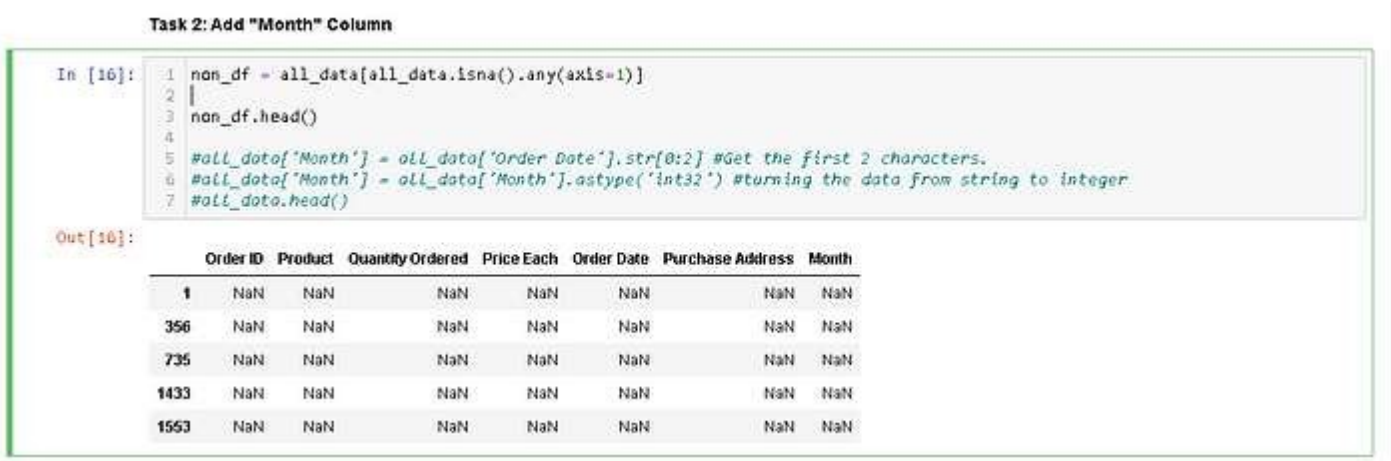
Task 2: Add "Month" Column

```
In [15]:
all_data['Month'] = all_data['Order Date'].str[0:2] #Get the first 2 characters.
all_data['Month'] = all_data['Month'].astype('int32') #turning the data from string to integer
all_data.head()
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-15-19256a884797> in <module>
      1 all_data['Month'] = all_data['Order Date'].str[0:2] #Get the first 2 characters.
----> 2 all_data['Month'] = all_data['Month'].astype('int32')
      3 all_data.head()

ValueError: cannot convert float NaN to integer
```

Figure 6. Adding "Month" Column

Now, we get an issue here. There are NaN values in our data. You could spot on of NaN value at Figure 1 or Figure 5 in index 1. Now we need to clean up the data by dropping rows of NaN. Let's spot more NaN value here. You don't have to do this, I am just curious.



Task 2: Add "Month" Column

```
In [16]:
non_df = all_data[all_data.isna().any(axis=1)]

non_df.head()

#all_data['Month'] = all_data['Order Date'].str[0:2] #Get the first 2 characters.
#all_data['Month'] = all_data['Month'].astype('int32') #turning the data from string to integer
#all_data.head()
```

Out[16]:

	Order ID	Product	Quantity Ordered	Price Each	Order Date	Purchase Address	Month
1	NaN	NaN	NaN	NaN	NaN	NaN	NaN
356	NaN	NaN	NaN	NaN	NaN	NaN	NaN
735	NaN	NaN	NaN	NaN	NaN	NaN	NaN
1433	NaN	NaN	NaN	NaN	NaN	NaN	NaN
1553	NaN	NaN	NaN	NaN	NaN	NaN	NaN

Figure 7. Spotting the NaN Values in our data.

We use *.isna().any(axis=1)* to spot rows containing the NaN values (axis = 0 to spot column containing NaN values). Now, we're gonna remove it from our dataframe using *.dropna()* method.

Task 2: Add "Month" Column

```
In [17]:
all_data=all_data.dropna(how='all')

all_data['Month'] = all_data['Order Date'].str[0:2] #Get the first 2 characters.
all_data['Month'] = all_data['Month'].astype('int32') #turning the data from string to integer
all_data.head()
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-17-dcd59b77aa46> in <module>
      2
      3 all_data['Month'] = all_data['Order Date'].str[0:2] #Get the first 2 characters.
----> 4 all_data['Month'] = all_data['Month'].astype('int32') #turning the data from string to integer
      5 all_data.head()

ValueError: invalid literal for int() with base 10: 'Or'
```

Figure 8. Dropping the NaN values from our dataframe.

.dropna() method is successful, but we get a new issue here. There are values "Or" in our data. Let's find it first.

Task 2: Add "Month" Column

```
In [19]:
#Removing Nan Values in our data
all_data=all_data.dropna(how='all')

#Removing rows based on condition, finding 'Or' and delete it
df_dummy = all_data[all_data['Order Date'].str[0:2]=='Or']
df_dummy.head()

#all_data['Month'] = all_data['Order Date'].str[0:2] #Get the first 2 characters.
#all_data['Month'] = all_data['Month'].astype('int32') #turning the data from string to integer
#all_data.head()
```

Out[19]:

	Order ID	Product	Quantity Ordered	Price Each	Order Date	Purchase Address	Month
519	Order ID	Product	Quantity Ordered	Price Each	Order Date	Purchase Address	Or
1149	Order ID	Product	Quantity Ordered	Price Each	Order Date	Purchase Address	Or
1155	Order ID	Product	Quantity Ordered	Price Each	Order Date	Purchase Address	Or
2878	Order ID	Product	Quantity Ordered	Price Each	Order Date	Purchase Address	Or
2893	Order ID	Product	Quantity Ordered	Price Each	Order Date	Purchase Address	Or

Figure 9. Finding "Or" values in our dataframe.

We can see clearly from Figure 9 that the issue is the rows contain the same words as the title rows. So clearly 'Or' is coming from 'Order Date'. We need to drop this "Or" rows just simply change the equal sign ("==") to not equal sign ("!=").

Task 2: Add "Month" Column

In [20]:
```
#Removing Nan Values in our data
all_data=all_data.dropna(how='all')

#Removing rows based on condition, finding 'Or' and delete it
all_data = all_data[all_data['Order Date'].str[0:2]!='Or']

all_data['Month'] = all_data['Order Date'].str[0:2] #Get the first 2 characters.
all_data['Month'] = all_data['Month'].astype('int32') #turning the data from string to integer
all_data.head()
```

Out[20]:

	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
2	176559	Bose SoundSport Headphones	1	99.99	04/07/19 22:30	682 Chestnut St, Boston, MA 02215	4
3	176560	Google Phone	1	600	04/12/19 14:38	669 Spruce St, Los Angeles, CA 90001	4
4	176560	Wired Headphones	1	11.99	04/12/19 14:38	669 Spruce St, Los Angeles, CA 90001	4
5	176561	Wired Headphones	1	11.99	04/30/19 09:27	333 9th St, Los Angeles, CA 90001	4

Figure 10. Dropping all unnecesary values in our dataframe.

We can see clearly from Figure 10 that we successfully created “Month” column and make its data type to integer.

Now, are we ready to answer the question? Not yet, we need obviously one more column called “Sales” Column. How can we get that? We get “Sales” by multiplying “Quantity Ordered” and “Price Each” values. Let’s create it.

Task 2: Add "Month" Column

In [21]:
```
#Removing Nan Values in our data
all_data=all_data.dropna(how='all')

#Removing rows based on condition, finding 'Or' and delete it
all_data = all_data[all_data['Order Date'].str[0:2]!='Or']

#Add "Month" Column
all_data['Month'] = all_data['Order Date'].str[0:2] #Get the first 2 characters.
all_data['Month'] = all_data['Month'].astype('int32') #turning the data from string to integer

#Add "Sales" Column
all_data['Sales'] = all_data['Quantity Ordered']*all_data['Price Each']

all_data.head()
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
~\anaconda3\lib\site-packages\pandas\core\ops\array_ops.py in na_arithmetic_op(left, right, op, str_rep)
    148     try:
--> 149         result = expressions.evaluate(op, str_rep, left, right)
    150     except TypeError:

TypeError: can't multiply sequence by non-int of type 'str'
```

Figure 11. Adding “Sales” Column in our dataframe.

Now we encounter a new issue. The values of the column “Quantity Ordered” and “Price Each” are strings. So the next task is to convert these columns to the correct type (“Quantity Ordered” is integer and “Price Each” is float). We’re gonna use *pd.to_numeric()* method to convert them to numeric.

Task 2: Add "Month" Column

In [22]:

```
#Removing Nan Values in our data
all_data=all_data.dropna(how='all')

#Removing rows based on condition, finding 'Or' and delete it
all_data = all_data[all_data['Order Date'].str[0:2]!='Or']

#Add "Month" Column
all_data['Month'] = all_data['Order Date'].str[0:2] #Get the first 2 characters.
all_data['Month'] = all_data['Month'].astype('int32') #turning the data from string to integer

#Convert 'Quantity Ordered' and 'Price Each' to numeric
all_data['Quantity Ordered'] = pd.to_numeric(all_data['Quantity Ordered']) #Becoming integer
all_data['Price Each'] = pd.to_numeric(all_data['Price Each']) #Becoming float

#Add "Sales" Column
all_data['Sales'] = all_data['Quantity Ordered']*all_data['Price Each']
all_data.head()
```

Out[22]:

	Order ID	Product	Quantity Ordered	Price Each	Order Date	Purchase Address	Month	Sales
0	176558	USB-C Charging Cable	2	11.95	04/19/19 08:46	917 1st St, Dallas, TX 75001	4	23.90
2	176559	Bose SoundSport Headphones	1	99.99	04/07/19 22:30	682 Chestnut St, Boston, MA 02215	4	99.99
3	176560	Google Phone	1	600.00	04/12/19 14:38	669 Spruce St, Los Angeles, CA 90001	4	600.00
4	176560	Wired Headphones	1	11.99	04/12/19 14:38	669 Spruce St, Los Angeles, CA 90001	4	11.99
5	176561	Wired Headphones	1	11.99	04/30/19 09:27	333 8th St, Los Angeles, CA 90001	4	11.99

Figure 12. Converting “Quantity Ordered” and “Price Each” Columns to Numeric

Now the “Sales” Column is successfully created, we can answer the first question. What was the best month for sales? How much was earned that month? We can easily answer it by using *groupby(‘Month’).sum()* method.

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

```
In [23]: 1 all_data.groupby('Month').sum()
```

Out[23]:

Month	Quantity Ordered	Price Each	Sales
1	10903	1.811768e+06	1.822257e+06
2	13449	2.188885e+06	2.202022e+06
3	17005	2.791208e+06	2.807100e+06
4	20558	3.367671e+06	3.390670e+06
5	18667	3.135125e+06	3.152607e+06
6	15253	2.562026e+06	2.577802e+06
7	16072	2.632540e+06	2.647776e+06
8	13448	2.230345e+06	2.244468e+06
9	13109	2.084992e+06	2.097560e+06
10	22703	3.715555e+06	3.736727e+06
11	19798	3.180601e+06	3.199603e+06
12	28114	4.588415e+06	4.613443e+06

Figure 13. Grouping by month and summing the Sales.

Look carefully at Figure 13. We can clearly see that month 12 (December) is the highest sales in 2019 with approximately $4,810,000. But we need to visualize it to make our bussiness partner easier to understand. So we're gonna import matplotlib and visualizing our results with bar chart.

Importing Matplotlib

```
In [27]: 1 import matplotlib.pyplot as plt
```

Visualizing our results

```
In [28]: 1 months = range(1,13) #For x axes
         2 results = all_data.groupby('Month').sum()
         3
         4 plt.bar(months, results['Sales'])
         5 plt.show()
```

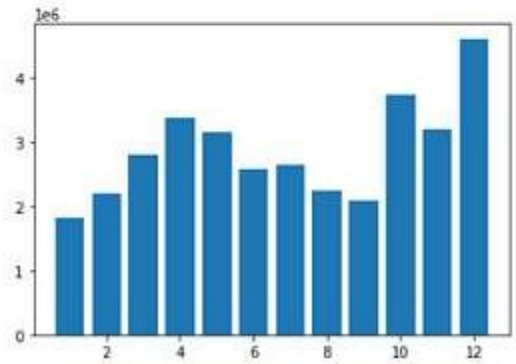


Figure 14. Visualizing our results using matplotlib library.

It’s good. But we need to make it looks neater. So we’re just gonna add a little code.

Importing Matplotlib

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

Visualizing our results

In [35]:
```
months = range(1,13) #For x axes
results = all_data.groupby('Month').sum()

plt.bar(months, results['Sales'])
plt.xticks(months)
labels, location = plt.yticks()
plt.yticks(labels, (labels/1000000).astype(int)) #Scaling in million USD
plt.ylabel('Sales in million USD')
plt.xlabel('Month Number')
plt.show()
```

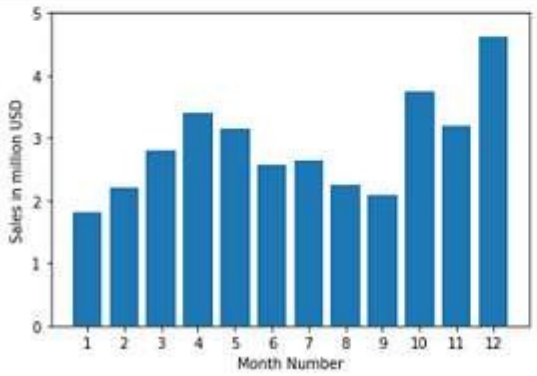


Figure 15. Improving the visualization.

Now, not only we can get the highest sales, but we can also get the lowest sales just by looking it for a few seconds. As a data scientist, we have to figure out why a certain month is better than others.
Maybe the company spend more money on April so the product sales are increasing. Maybe the best product sales are on December because it’s holiday and Christmas. Those are just my hyphothesis, right now we don’t have enough data to prove that hypothesis. But we can take these as a consideration if you want to decide something that relates to product sales.

What city sold the most product?

To answer this question, obviously we need to create a new column called “City” column. How do we get that? As usual, we’re gonna check the top 5 data in our dataframe to figure out where can we get our “City” column using *.head()* method.

Question 2: What city sold the most product?

Task 3: Add a "City" Column

```
In [11]: 1 all_data.head()
```

Out[11]:

	Order ID	Product	Quantity Ordered	Price Each	Order Date	Purchase Address	Month	Sales
0	176558	USB-C Charging Cable	2	11.95	04/19/19 08:46	917 1st St, Dallas, TX 75001	4	23.90
2	176559	Bose SoundSport Headphones	1	99.99	04/07/19 22:30	682 Chestnut St, Boston, MA 02215	4	99.99
3	176560	Google Phone	1	600.00	04/12/19 14:38	669 Spruce St, Los Angeles, CA 90001	4	600.00
4	176560	Wired Headphones	1	11.99	04/12/19 14:38	669 Spruce St, Los Angeles, CA 90001	4	11.99
5	176561	Wired Headphones	1	11.99	04/30/19 09:27	333 8th St, Los Angeles, CA 90001	4	11.99

Figure 16. Showing Our Top 5 Updated Dataframe

As you can see at Figure 16, the “Purchase Address” Column contain the city. We can’t get it directly, we need to extract the data. We can use one of most useful function in pandas, *.apply()* method.

Question 2: What city sold the most product?

Task 3: Add a "City" Column

```
In [15]: 1 all_data['City'] = all_data['Purchase Address'].apply(lambda x: x.split(',')[1])
         2
         3 all_data.head()
```

Out[15]:

	Order ID	Product	Quantity Ordered	Price Each	Order Date	Purchase Address	Month	Sales	City
0	176558	USB-C Charging Cable	2	11.95	04/19/19 08:46	917 1st St, Dallas, TX 75001	4	23.90	Dallas
2	176559	Bose SoundSport Headphones	1	99.99	04/07/19 22:30	682 Chestnut St, Boston, MA 02215	4	99.99	Boston
3	176560	Google Phone	1	600.00	04/12/19 14:38	669 Spruce St, Los Angeles, CA 90001	4	600.00	Los Angeles
4	176560	Wired Headphones	1	11.99	04/12/19 14:38	669 Spruce St, Los Angeles, CA 90001	4	11.99	Los Angeles
5	176561	Wired Headphones	1	11.99	04/30/19 09:27	333 8th St, Los Angeles, CA 90001	4	11.99	Los Angeles

Figure 17. Using .apply() Method to Extract The Data

To make it neater, we can use this

Question 2: What city sold the most product?

Task 3: Add a "City" Column

In [17]:
```
#Function
def get_city(address):
    return address.split(',')[1]

#Extract the city and the state
all_data['City'] = all_data['Purchase Address'].apply(lambda x: get_city(x))

all_data.head()
```

Out[17]:

	Order ID	Product	Quantity Ordered	Price Each	Order Date	Purchase Address	Month	Sales	City
0	176558	USB-C Charging Cable	2	11.95	04/19/19 08:46	917 1st St, Dallas, TX 75001	4	23.90	Dallas
2	176559	Bose SoundSport Headphones	1	99.99	04/07/19 22:30	682 Chestnut St, Boston, MA 02215	4	99.99	Boston
3	176560	Google Phone	1	600.00	04/12/19 14:38	669 Spruce St, Los Angeles, CA 90001	4	600.00	Los Angeles
4	176560	Wired Headphones	1	11.99	04/12/19 14:38	669 Spruce St, Los Angeles, CA 90001	4	11.99	Los Angeles
5	176561	Wired Headphones	1	11.99	04/30/19 09:27	333 8th St, Los Angeles, CA 90001	4	11.99	Los Angeles

Figure 18. Using def function to Make The Code Neater

apply and lambda usually used to create new column based on other column (example .apply(lambda x: x*2. It means every input x in other column will be changed to x*2 in a new column). In this case we create “City” column based on “Purchase Address” column and we split the data into 3 part. The first one is before the first comma (index = 0), the second one is between the commas (index = 1), and the third on is after the last comma (index = 2). As we need to extract the city data, we use [1] to state it to index 1.

As you can see at Figure 17, we successfully created a “City” column. So are we ready to answer the second question? Not yet. We get an issue here. It’s not error, it’s the value of the “City” Column. This is just a rare case when there are 2 cities are named exactly the same. Example someone in New England and someone in West Coast would think Portland in different way. Someone in New England thinks Portland as Portland Maine and someone in West Coast thinks Portland as Portland Oregon. So in our dataset we actually had the overlapping cities between these two. So, we should also grab the state.

Question 2: What city sold the most product?

Task 3: Add a "City" Column

In [18]:
```
#Function
def get_city(address):
    return address.split(',')[1]

def get_state(address):
    return address.split(',')[2].split(' ')[1]

#Extract the city and the state
all_data['City'] = all_data['Purchase Address'].apply(lambda x: get_city(x) + ' ' + get_state(x))

all_data.head()
```

Out[18]:

	Order ID	Product	Quantity Ordered	Price Each	Order Date	Purchase Address	Month	Sales	City
0	176558	USB-C Charging Cable	2	11.95	04/19/19 08:46	917 1st St, Dallas, TX 75001	4	23.90	Dallas TX
2	176559	Bose SoundSport Headphones	1	99.99	04/07/19 22:30	682 Chestnut St, Boston, MA 02215	4	99.99	Boston MA
3	176560	Google Phone	1	600.00	04/12/19 14:38	669 Spruce St, Los Angeles, CA 90001	4	600.00	Los Angeles CA
4	176560	Wired Headphones	1	11.99	04/12/19 14:38	669 Spruce St, Los Angeles, CA 90001	4	11.99	Los Angeles CA
5	176561	Wired Headphones	1	11.99	04/30/19 09:27	333 8th St, Los Angeles, CA 90001	4	11.99	Los Angeles CA

Figure 19. Extracting the state to “City” Column

The function get_state() basically works as explained before. But in this function, we seperate the data again become three parts. The first one is before the whitespace (index = 0), the second one is between the whitespaces (index = 1), and the third one is after the last whitespace (index = 2). So that’s why we use *.split(‘ ’)[1*] in the second split.

Now we’re ready to answer the second question, **what city sold the most product**? As we did before, we’re gonna group it by the city and summing all the values based on the group.

```
In [19]:
1 results2 = all_data.groupby('City').sum()
2 results2
```

Out[19]:

City	Quantity Ordered	Price Each	Month	Sales
Atlanta GA	16602	2.779908e+06	104794	2.795499e+06
Austin TX	11153	1.809874e+06	69829	1.819582e+06
Boston MA	22528	3.637410e+06	141112	3.661642e+06
Dallas TX	16730	2.752628e+06	104620	2.767975e+06
Los Angeles CA	33289	5.421435e+06	208325	5.452571e+06
New York City NY	27932	4.635371e+06	175741	4.664317e+06
Portland ME	2750	4.471893e+05	17144	4.497583e+05
Portland OR	11303	1.860558e+06	70621	1.870732e+06
San Francisco CA	50239	8.211462e+06	315520	8.262204e+06
Seattle WA	16553	2.733296e+06	104941	2.747755e+06

Figure 20. Grouping The Dataframe by The City

It's too messy, but if you look carefully you can see that San Fransisco is the highest sold product of all cities with approximately $8,200,000. We clearly need to visualize it because it's so hard to conclude anything just based on that numbers and also it will make our bussiness partner easier to understand.

```
In [22]:
#We've already import the matplotlib

cities = all_data['City'].unique()

plt.bar(cities, results2['Sales'])
plt.xticks(cities, rotation='vertical', size = 8)
labels, location = plt.yticks()
plt.yticks(labels, (labels/1000000).astype(int)) #Scaling in million USD
plt.ylabel('Sales in million USD')
plt.xlabel('City Name')
plt.show()
```

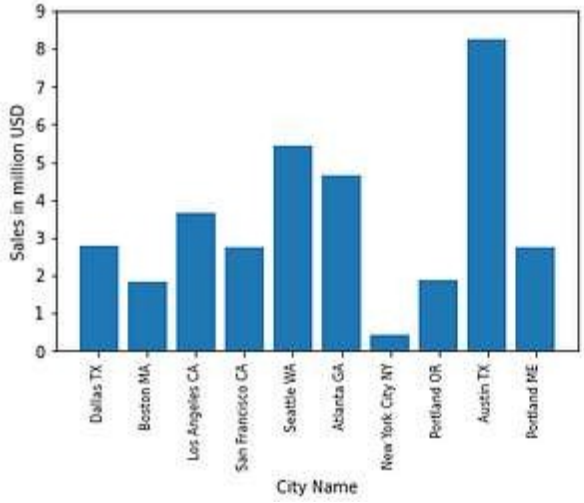


Figure 21. Plotting the Sales Grouped by Cities.

Now we successfully plot it. But there is a big issue here. If you notice that the values (Figure 20) and the plot (Figure 21) are not synchronized. The highest sales should be San Fransisco. What's wrong with our code?

There's an issue between *.unique()* method and *plt.bar()*. Their cities order are different. we're gonna syncronized the order by simply fixing the variable 'cities'.

```
In [23]:
#We've already import the matplotlib

#Fixing the cities order
#cities = all_data['City'].unique()
cities = [city for city, df in all_data.groupby('City')]

plt.bar(cities, results2['Sales'])
plt.xticks(cities, rotation='vertical', size = 8)
labels, location = plt.yticks()
plt.yticks(labels, (labels/1000000).astype(int)) #Scaling in million USD
plt.ylabel('Sales in million USD')
plt.xlabel('City Name')
plt.show()
```

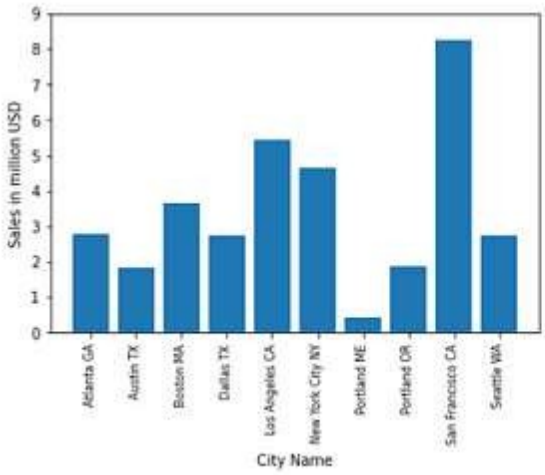


Figure 22. Fixing The Code

Now, we fix the issue and successfully plot it. As a data scientist, we need to figure out why San Fransisco is the highest sale compare to other cities. Maybe Sillicon Valley need more electronic products.
Maybe the advertisement is better in San Fransisco. We can use this data to improve the sales of bussiness.

What Time Should We Display Advertisements to Maximize Likelihood of Customer's Buying Product?
As usual, to remind what our data look like, we use the .head() method to show the top 5 of our updated dataframe.

Question 3: What time should we display advertisements to maximize likelihood of customer's buying product?

```
In [24]: 1 all_data.head()
```

Out[24]:

	Order ID	Product	Quantity Ordered	Price Each	Order Date	Purchase Address	Month	Sales	City
0	176558	USB-C Charging Cable	2	11.95	04/19/19 08:46	917 1st St, Dallas, TX 75001	4	23.90	Dallas TX
2	176559	Bose SoundSport Headphones	1	99.99	04/07/19 22:30	682 Chestnut St, Boston, MA 02215	4	99.99	Boston MA
3	176560	Google Phone	1	600.00	04/12/19 14:38	669 Spruce St, Los Angeles, CA 90001	4	600.00	Los Angeles CA
4	176560	Wired Headphones	1	11.99	04/12/19 14:38	669 Spruce St, Los Angeles, CA 90001	4	11.99	Los Angeles CA
5	176561	Wired Headphones	1	11.99	04/30/19 09:27	333 8th St, Los Angeles, CA 90001	4	11.99	Los Angeles CA

Figure 23. Showing our Top 5 updated dataframe

If we're gonna use our data to answer this question, we need to aggregate the period in 24 hours distribution. Look carefully at Figure 23. In "Order Date" column, there are times data. We could extract it like we did before. But to make it more consistent, we need to convert the "Order Date" Column into date time object. We're gonna use *pd.to_datetime()* method.

Question 3: What time should we display advertisements to maximize likelihood of customer's buying product?

Task 4: Aggregate the period in 24-hours distribution

```
In [29]: 1 #Create new column in date-time Object (DTO)
         2 all_data['Order_Date_DTO'] = pd.to_datetime(all_data['Order Date'])
         3 all_data.head()
```

Out[29]:

	Order ID	Product	Quantity Ordered	Price Each	Order Date	Purchase Address	Month	Sales	City	Order_Date_DTO
0	176558	USB-C Charging Cable	2	11.95	04/19/19 08:46	917 1st St, Dallas, TX 75001	4	23.90	Dallas TX	2019-04-19 08:46:00
2	176559	Bose SoundSport Headphones	1	99.99	04/07/19 22:30	682 Chestnut St, Boston, MA 02215	4	99.99	Boston MA	2019-04-07 22:30:00
3	176560	Google Phone	1	600.00	04/12/19 14:38	669 Spruce St, Los Angeles, CA 90001	4	600.00	Los Angeles CA	2019-04-12 14:38:00
4	176560	Wired Headphones	1	11.99	04/12/19 14:38	669 Spruce St, Los Angeles, CA 90001	4	11.99	Los Angeles CA	2019-04-12 14:38:00
5	176561	Wired Headphones	1	11.99	04/30/19 09:27	333 8th St, Los Angeles, CA 90001	4	11.99	Los Angeles CA	2019-04-30 09:27:00

Figure 24. Converting the "Order Date" Column into Date-Time Object

It will take a little bit longer because of the heavy calculation. Now we can create a new column called “Hour” contain the extraction of “Order_Date_DTO” data. We only need the hours data, so we can extract them by doing this.

Task 4: Aggregate the period in 24-hours distribution

In [30]:
```
#Create new column in date-time Object (DTO)
all_data['Order_Date_DTO'] = pd.to_datetime(all_data['Order Date'])

#Extraction the hours data
all_data['Hour'] = all_data['Order_Date_DTO'].dt.hour

all_data.head()
```

Out[30]:

	Order ID	Product	Quantity Ordered	Price Each	Order Date	Purchase Address	Month	Sales	City	Order_Date_DTO	Hour
0	176558	USB-C Charging Cable	2	11.95	04/19/19 08:46	917 1st St, Dallas, TX 75001	4	23.90	Dallas TX	2019-04-19 08:46:00	8
2	176559	Bose SoundSport Headphones	1	99.99	04/07/19 22:30	682 Chestnut St, Boston, MA 02215	4	99.99	Boston MA	2019-04-07 22:30:00	22
3	176560	Google Phone	1	600.00	04/12/19 14:38	669 Spruce St, Los Angeles, CA 90001	4	600.00	Los Angeles CA	2019-04-12 14:38:00	14
4	176560	Wired Headphones	1	11.99	04/12/19 14:38	669 Spruce St, Los Angeles, CA 90001	4	11.99	Los Angeles CA	2019-04-12 14:38:00	14
5	176561	Wired Headphones	1	11.99	04/30/19 09:27	333 8th St, Los Angeles, CA 90001	4	11.99	Los Angeles CA	2019-04-30 09:27:00	9

Figure 25. Extracting The Hours Data Into The New Column

Now we can answer the third question, **what time should we display advertisements to maximize likelihood of customer’s buying product?** To answer this, we’re gonna group it by the hours and counting all of the orders.

In [40]:
```
results3 = all_data.groupby(['Hour']).count()
results3
```

Out[40]:

Hour	Order ID	Product	Quantity Ordered	Price Each	Order Date	Purchase Address	Month	Sales	City	Order_Date_DTO
0	3910	3910	3910	3910	3910	3910	3910	3910	3910	3910
1	2350	2350	2350	2350	2350	2350	2350	2350	2350	2350
2	1243	1243	1243	1243	1243	1243	1243	1243	1243	1243
3	831	831	831	831	831	831	831	831	831	831
4	854	854	854	854	854	854	854	854	854	854
5	1321	1321	1321	1321	1321	1321	1321	1321	1321	1321
6	2482	2482	2482	2482	2482	2482	2482	2482	2482	2482
7	4011	4011	4011	4011	4011	4011	4011	4011	4011	4011
8	6256	6256	6256	6256	6256	6256	6256	6256	6256	6256
9	8748	8748	8748	8748	8748	8748	8748	8748	8748	8748
10	10944	10944	10944	10944	10944	10944	10944	10944	10944	10944
11	12411	12411	12411	12411	12411	12411	12411	12411	12411	12411
12	12587	12587	12587	12587	12587	12587	12587	12587	12587	12587

13	12129	12129	12129	12129	12129	12129	12129	12129	12129	12129
14	10984	10984	10984	10984	10984	10984	10984	10984	10984	10984
15	10175	10175	10175	10175	10175	10175	10175	10175	10175	10175
16	10384	10384	10384	10384	10384	10384	10384	10384	10384	10384
17	10899	10899	10899	10899	10899	10899	10899	10899	10899	10899
18	12280	12280	12280	12280	12280	12280	12280	12280	12280	12280
19	12905	12905	12905	12905	12905	12905	12905	12905	12905	12905
20	12228	12228	12228	12228	12228	12228	12228	12228	12228	12228
21	10921	10921	10921	10921	10921	10921	10921	10921	10921	10921
22	8822	8822	8822	8822	8822	8822	8822	8822	8822	8822
23	6275	6275	6275	6275	6275	6275	6275	6275	6275	6275

Figure 26. Grouping the data by the hours

If we want to answer the third question, we only need the “Quantity Ordered” column. Now let’s visualize it. We want it to be the line chart because this spesific data (hours) are more logical to show using line chart than bar chart because the data has to be continue.

```
In [41]:
#Plotting
results3 = all_data.groupby(['Hour'])['Quantity Ordered'].count()
hours = [hour for hour, df in all_data.groupby('Hour')]

plt.plot(hours, results3)
plt.xticks(hours)
plt.xlabel('Hour')
plt.ylabel('Number of Orders')
plt.grid()
plt.show()
```

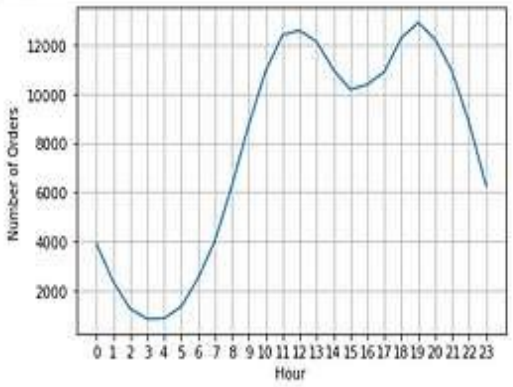


Figure 27. Visualizing the Number of Orders in 24 hours format

As you can see from Figure 27, there are approximately 2 peaks at the data. They are 12 (12 PM) and 19 (7 PM). It makes sense since most people shopping during the day. From this data, we can suggest to our bussiness partner to advertise their product right before 12 PM and/or 7 PM. It could be 11.30 AM and/or 6.30 PM.
Remember, this chart is the total orders of **all cities**. Maybe you could make a spesific chart for a spesific city and planning the advertisement better for that city.

What Products Are Most Often Sold Together?
We're gonna take a look to our top 5 data as usual.

Question 4: What products are most often sold together?

```
In [42]: 1 all_data.head()
```

Out[42]:

	Order ID	Product	Quantity Ordered	Price Each	Order Date	Purchase Address	Month	Sales	City	Order_Date_DTO	Hour
0	176558	USB-C Charging Cable	2	11.95	04/19/19 08:46	917 1st St, Dallas, TX 75001	4	23.90	Dallas TX	2019-04-19 08:46:00	8
2	176559	Bose SoundSport Headphones	1	99.99	04/07/19 22:30	682 Chestnut St, Boston, MA 02215	4	99.99	Boston MA	2019-04-07 22:30:00	22
3	176560	Google Phone	1	600.00	04/12/19 14:38	669 Spruce St, Los Angeles, CA 90001	4	600.00	Los Angeles CA	2019-04-12 14:38:00	14
4	176560	Wired Headphones	1	11.99	04/12/19 14:38	669 Spruce St, Los Angeles, CA 90001	4	11.99	Los Angeles CA	2019-04-12 14:38:00	14
5	176561	Wired Headphones	1	11.99	04/30/19 09:27	333 8th St, Los Angeles, CA 90001	4	11.99	Los Angeles CA	2019-04-30 09:27:00	9

Figure 28. Showing Our Top 5 Updated Dataframe

Look carefully at Figure 28. We can see that "Order ID" indicate the transaction. So by grouping the product by the Order ID, we are able to know what products are often sold together. We're gonna use .duplicated() method to find a duplicate values of "Order ID".

Task 5: Make a new column called "Product Bundle"

In [43]:
```
#Make a new dataframe to seperate the duplicated values of Order ID
new_all = all_data[all_data['Order ID'].duplicated(keep=False)]
new_all.head(20)
```

Out[43]:

	Order ID	Product	Quantity Ordered	Price Each	Order Date	Purchase Address	Month	Sales	City	Order_Date_DTO	Hour
3	176560	Google Phone	1	600.00	04/12/19 14:38	669 Spruce St, Los Angeles, CA 90001	4	600.00	Los Angeles CA	2019-04-12 14:38:00	14
4	176560	Wired Headphones	1	11.99	04/12/19 14:38	669 Spruce St, Los Angeles, CA 90001	4	11.99	Los Angeles CA	2019-04-12 14:38:00	14
18	176574	Google Phone	1	600.00	04/03/19 19:42	20 Hill St, Los Angeles, CA 90001	4	600.00	Los Angeles CA	2019-04-03 19:42:00	19
19	176574	USB-C Charging Cable	1	11.95	04/03/19 19:42	20 Hill St, Los Angeles, CA 90001	4	11.95	Los Angeles CA	2019-04-03 19:42:00	19
30	176585	Bose SoundSport Headphones	1	99.99	04/07/19 11:31	823 Highland St, Boston, MA 02215	4	99.99	Boston MA	2019-04-07 11:31:00	11
31	176585	Bose SoundSport Headphones	1	99.99	04/07/19 11:31	823 Highland St, Boston, MA 02215	4	99.99	Boston MA	2019-04-07 11:31:00	11
32	176586	AAA Batteries (4-pack)	2	2.99	04/10/19 17:00	365 Center St, San Francisco, CA 94016	4	5.98	San Francisco CA	2019-04-10 17:00:00	17
33	176586	Google Phone	1	600.00	04/10/19 17:00	365 Center St, San Francisco, CA 94016	4	600.00	San Francisco CA	2019-04-10 17:00:00	17
119	176672	Lightning Charging Cable	1	14.95	04/12/19 11:07	778 Maple St, New York City, NY 10001	4	14.95	New York City NY	2019-04-12 11:07:00	11
120	176672	USB-C Charging Cable	1	11.95	04/12/19 11:07	778 Maple St, New York City, NY 10001	4	11.95	New York City NY	2019-04-12 11:07:00	11
129	176681	Apple Airpods Headphones	1	150.00	04/20/19 10:39	331 Cherry St, Seattle, WA 98101	4	150.00	Seattle WA	2019-04-20 10:39:00	10
130	176681	ThinkPad Laptop	1	999.99	04/20/19 10:39	331 Cherry St, Seattle, WA 98101	4	999.99	Seattle WA	2019-04-20 10:39:00	10
138	176689	Bose SoundSport Headphones	1	99.99	04/24/19 17:15	659 Lincoln St, New York City, NY 10001	4	99.99	New York City NY	2019-04-24 17:15:00	17
139	176689	AAA Batteries (4-pack)	2	2.99	04/24/19 17:15	659 Lincoln St, New York City, NY 10001	4	5.98	New York City NY	2019-04-24 17:15:00	17
189	176739	34in Ultrawide Monitor	1	379.99	04/05/19 17:38	730 6th St, Austin, TX 73301	4	379.99	Austin TX	2019-04-05 17:38:00	17
190	176739	Google Phone	1	600.00	04/05/19 17:38	730 6th St, Austin, TX 73301	4	600.00	Austin TX	2019-04-05 17:38:00	17
225	176774	Lightning Charging Cable	1	14.95	04/25/19 15:06	372 Church St, Los Angeles, CA 90001	4	14.95	Los Angeles CA	2019-04-25 15:06:00	15
226	176774	USB-C Charging Cable	1	11.95	04/25/19 15:06	372 Church St, Los Angeles, CA 90001	4	11.95	Los Angeles CA	2019-04-25 15:06:00	15

Figure 29. Showing Our Top 20 Duplicated Dataframe

Now we want to create a column called “Product Bundle” that contain example *Google Phone* and *Wired Headphone* (transaction 17650) at the same line. We’re gonna use the *.transform()* method to join values from two rows into a single row.

Question 4: What products are most often sold together?

Task 5: Make a new column called "Product Bundle"

In [44]:
```
#Make a new dataframe to seperate the duplicated values of Order ID
new_all = all_data[all_data['Order ID'].duplicated(keep=False)]

#Joining few products with the same Order ID into the same line.
new_all['Product_Bundle'] = new_all.groupby('Order ID')['Product'].transform(lambda x: ','.join(x))

new_all.head()
```

Out[44]:

	Order ID	Product	Quantity Ordered	Price Each	Order Date	Purchase Address	Month	Sales	City	Order_Date_DTO	Hour	Product_Bundle
3	176560	Google Phone	1	600.00	04/12/19 14:38	669 Spruce St, Los Angeles, CA 90001	4	600.00	Los Angeles CA	2019-04-12 14:38:00	14	Google Phone,Wired Headphones
4	176560	Wired Headphones	1	11.99	04/12/19 14:38	669 Spruce St, Los Angeles, CA 90001	4	11.99	Los Angeles CA	2019-04-12 14:38:00	14	Google Phone,Wired Headphones
18	176574	Google Phone	1	600.00	04/03/19 19:42	20 Hill St, Los Angeles, CA 90001	4	600.00	Los Angeles CA	2019-04-03 19:42:00	19	Google Phone,USB-C Charging Cable
19	176574	USB-C Charging Cable	1	11.95	04/03/19 19:42	20 Hill St, Los Angeles, CA 90001	4	11.95	Los Angeles CA	2019-04-03 19:42:00	19	Google Phone,USB-C Charging Cable
30	176585	Bose SoundSport Headphones	1	99.99	04/07/19 11:31	823 Highland St, Boston, MA 02215	4	99.99	Boston MA	2019-04-07 11:31:00	11	Bose SoundSport Headphones,Bose SoundSport Hea...

Figure 30. Joining Few Products With The Same Order ID Into The Same Line

It's good, but we have an issue here. We have the same order at least twice because we did merge them in every situation in groupby without dropping the duplicate values. Now let's drop the rows with duplicate values.

Question 4: What products are most often sold together? ¶

Task 5: Make a new column called "Product Bundle"

In [18]:
```
#Make a new dataframe to seperate the duplicated values of Order ID
new_all = all_data[all_data['Order ID'].duplicated(keep=False)]

#Joining few products with the same Order ID into the same line.
new_all['Product_Bundle'] = new_all.groupby('Order ID')['Product'].transform(lambda x: ','.join(x))

#Dropping the duplicate values
new_all = new_all[['Order ID','Product_Bundle']].drop_duplicates()

new_all.head()
```

Out[18]:

	Order ID	Product_Bundle
3	176560	Google Phone,Wired Headphones
18	176574	Google Phone,USB-C Charging Cable
30	176585	Bose SoundSport Headphones,Bose SoundSport Hea...
32	176586	AAA Batteries (4-pack),Google Phone
119	176672	Lightning Charging Cable,USB-C Charging Cable

Figure 31. Dropping rows with duplicate values

Now, we need to count the pair of products. We need new libraries because they have all we need to count all the combination of products bundle. We're gonna use *itertools* and *collections* libraries.

Task 6: Counting the Product bundles

In [21]:
```
#Importing libraries
from itertools import combinations
from collections import Counter

count = Counter()

for row in new_all['Product_Bundle']:
    row_list = row.split(',')
    count.update(Counter(combinations(row_list,2))) #Counting all the 2 products bundle
print(count)
```

```
Counter({('iPhone', 'Lightning Charging Cable'): 1005, ('Google Phone', 'USB-C Charging Cable'): 987, ('iPhone', 'Wired Hea
dphones'): 447, ('Google Phone', 'Wired Headphones'): 414, ('Vareebadd Phone', 'USB-C Charging Cable'): 361, ('iPhone', 'Ap
ple Airpods Headphones'): 360, ('Google Phone', 'Bose SoundSport Headphones'): 220, ('USB-C Charging Cable', 'Wired Headpho
nes'): 160, ('Vareebadd Phone', 'Wired Headphones'): 143, ('Lightning Charging Cable', 'Wired Headphones'): 92, ('Lightning
Charging Cable', 'Apple Airpods Headphones'): 81, ('Vareebadd Phone', 'Bose SoundSport Headphones'): 80, ('USB-C Charging C
able', 'Bose SoundSport Headphones'): 77, ('Apple Airpods Headphones', 'Wired Headphones'): 69, ('Lightning Charging Cabl
e', 'USB-C Charging Cable'): 58, ('Lightning Charging Cable', 'AA Batteries (4-pack)'): 55, ('Lightning Charging Cable', 'L
ightning Charging Cable'): 54, ('Bose SoundSport Headphones', 'Wired Headphones'): 53, ('AA Batteries (4-pack)', 'Lightning
Charging Cable'): 51, ('AAA Batteries (4-pack)', 'USB-C Charging Cable'): 50, ('Apple Airpods Headphones', 'AAA Batteries
(4-pack)'): 48, ('AA Batteries (4-pack)', 'AAA Batteries (4-pack)'): 48, ('USB-C Charging Cable', 'USB-C Charging Cable'):
48, ('AAA Batteries (4-pack)', 'AAA Batteries (4-pack)'): 48, ('USB-C Charging Cable', 'AAA Batteries (4-pack)'): 45, ('Wir
ed Headphones', 'USB-C Charging Cable'): 45, ('AA Batteries (4-pack)', 'Wired Headphones'): 44, ('AAA Batteries (4-pack)',
'Lightning Charging Cable'): 44, ('AAA Batteries (4-pack)', 'Wired Headphones'): 43, ('Wired Headphones', 'AAA Batteries (4
-pack)'): 43, ('USB-C Charging Cable', 'Lightning Charging Cable'): 42, ('AA Batteries (4-pack)', 'Apple Airpods Headphone
s'): 41, ('AAA Batteries (4-pack)', 'AA Batteries (4-pack)'): 39, ('Wired Headphones', 'AA Batteries (4-pack)'): 39, ('Ligh
tning Charging Cable', 'Bose SoundSport Headphones'): 39, ('USB-C Charging Cable', 'AA Batteries (4-pack)'): 38, ('Bose Sou
ndSport Headphones', 'AAA Batteries (4-pack)'): 37, ('AA Batteries (4-pack)', 'USB-C Charging Cable'): 37, ('Wired Headphon
es', 'Lightning Charging Cable'): 37, ('Lightning Charging Cable', 'AAA Batteries (4-pack)'): 36, ('Apple Airpods Headphone
s', 'Lightning Charging Cable'): 35, ('Wired Headphones', 'Wired Headphones'): 35, ('AA Batteries (4-pack)', 'AA Batteries
```

Figure 32. Counting the Product Bundle

It's too messy, let's just showing the top 10 data

Task 6: Counting the Product bundles

In [23]:
```
#Importing libraries
from itertools import combinations
from collections import Counter

count = Counter()

for row in new_all['Product_Bundle']:
    row_list = row.split(',')
    count.update(Counter(combinations(row_list,2))) #Counting all the 2 products bundle

count.most_common(10)
```

Out[23]:
```
[(('iPhone', 'Lightning Charging Cable'), 1005),
 (('Google Phone', 'USB-C Charging Cable'), 987),
 (('iPhone', 'Wired Headphones'), 447),
 (('Google Phone', 'Wired Headphones'), 414),
 (('Vareebadd Phone', 'USB-C Charging Cable'), 361),
 (('iPhone', 'Apple Airpods Headphones'), 360),
 (('Google Phone', 'Bose SoundSport Headphones'), 220),
 (('USB-C Charging Cable', 'Wired Headphones'), 160),
 (('Vareebadd Phone', 'Wired Headphones'), 143),
 (('Lightning Charging Cable', 'Wired Headphones'), 92)]
```

Figure 33. Showing The Top 10 2-Product Bundles

Now we can clearly see that the most often products sold together are iPhone and Lightning Charging Cable with 1005 transactions.
We could count the 3 product bundles by just changing the *count.update* index into 3.

Task 6: Counting the Product bundles

In [24]:
```
#Importing Libraries
from itertools import combinations
from collections import Counter

count = Counter()

for row in new_all['Product_Bundle']:
    row_list = row.split(',')
    #count.update(Counter(combinations(row_list,2))) #Counting all the 2 products bundle
    count.update(Counter(combinations(row_list,3))) #Counting all the 3 products bundle

count.most_common(10)

```

Out[24]:
```
[(('Google Phone', 'USB-C Charging Cable', 'Wired Headphones'), 87),
 (('iPhone', 'Lightning Charging Cable', 'Wired Headphones'), 62),
 (('iPhone', 'Lightning Charging Cable', 'Apple Airpods Headphones'), 47),
 (('Google Phone', 'USB-C Charging Cable', 'Bose SoundSport Headphones'), 35),
 (('Vareebadd Phone', 'USB-C Charging Cable', 'Wired Headphones'), 33),
 (('iPhone', 'Apple Airpods Headphones', 'Wired Headphones'), 27),
 (('Google Phone', 'Bose SoundSport Headphones', 'Wired Headphones'), 24),
 (('Vareebadd Phone', 'USB-C Charging Cable', 'Bose SoundSport Headphones'),
  16),
 (('USB-C Charging Cable', 'Bose SoundSport Headphones', 'Wired Headphones'),
  5),
 (('Vareebadd Phone', 'Bose SoundSport Headphones', 'Wired Headphones'), 5)]
```

Figure 34. Showing The Top 10 3-Product Bundles

We can see the most often sold products (3 products) together are Google Phone, USB-C Charging Cable, and Wired Headphones with 87 transactions. It's not really significant compare to the 2-Product Bundle. So we're gonna ignore the 3-Product bundle.

What would we do with this data? Well, we could offer a smart deal to the customer that buy iPhone, you could recommend the charging cable with discount. That's one of the possibility and you can bundle the remaining products if you need to.

What product sold the most? Why do you think it did?
As usual, let's see our data looks like again.

Question 5: What Product sold the most? Why do you think it did?

```
In [25]: 1 all_data.head()
```

Out[25]:

	Order ID	Product	Quantity Ordered	Price Each	Order Date	Purchase Address	Month	Sales	City	Order_Date_DTO	Hour
0	176558	USB-C Charging Cable	2	11.95	04/19/19 08:46	917 1st St, Dallas, TX 75001	4	23.90	Dallas TX	2019-04-19 08:46:00	8
2	176559	Bose SoundSport Headphones	1	99.99	04/07/19 22:30	682 Chestnut St, Boston, MA 02215	4	99.99	Boston MA	2019-04-07 22:30:00	22
3	176560	Google Phone	1	600.00	04/12/19 14:38	669 Spruce St, Los Angeles, CA 90001	4	600.00	Los Angeles CA	2019-04-12 14:38:00	14
4	176560	Wired Headphones	1	11.99	04/12/19 14:38	669 Spruce St, Los Angeles, CA 90001	4	11.99	Los Angeles CA	2019-04-12 14:38:00	14
5	176561	Wired Headphones	1	11.99	04/30/19 09:27	333 8th St, Los Angeles, CA 90001	4	11.99	Los Angeles CA	2019-04-30 09:27:00	9

Figure 35. Showing Out Top 5 Updated Dataframe

We need to sum up the "Quantity Ordered" based on grouping the Product. So let's do it.

Question 5: What Product sold the most? Why do you think it did?

Task 7: Grouping by the product

In [27]:
```
product_group = all_data.groupby('Product')
product_group.sum()
```

Out[27]:

Product	Quantity Ordered	Price Each	Month	Sales	Hour
20in Monitor	4129	451068.99	29336	454148.71	58764
27in 4K Gaming Monitor	6244	2429637.70	44440	2435097.56	90916
27in FHD Monitor	7550	1125974.93	52658	1132424.50	107540
34in Ultrawide Monitor	6199	2348718.19	43304	2355558.01	89076
AA Batteries (4-pack)	27635	79015.68	145558	106118.40	298342
AAA Batteries (4-pack)	31017	61716.59	140370	92740.83	297332
Apple Airpods Headphones	15661	2332350.00	109477	2349150.00	223304
Bose SoundSport Headphones	13457	1332366.75	94113	1345565.43	192445
Flatscreen TV	4819	1440000.00	34224	1445700.00	68815
Google Phone	5532	3315000.00	38305	3319200.00	79479
LG Dryer	646	387600.00	4383	387600.00	9326
LG Washing Machine	666	399600.00	4523	399600.00	9785
Lightning Charging Cable	23217	323787.10	153092	347094.15	312529
Macbook Pro Laptop	4728	8030800.00	33548	8037600.00	68261
ThinkPad Laptop	4130	4127958.72	28950	4129958.70	59746
USB-C Charging Cable	23975	261740.85	154819	286501.25	314645
Vareebadd Phone	2068	826000.00	14309	827200.00	29472
Wired Headphones	20557	226395.18	133397	246478.43	271720
iPhone	6849	4789400.00	47941	4794300.00	98657

Figure 36. Grouping by the Product

To make it easier to understand, let’s visualize it.

Question 5: What Product sold the most? Why do you think it did?

Task 7: Grouping by the product

In [28]:
```
product_group = all_data.groupby('Product')

#Visualizing
quantity_ordered = product_group.sum()['Quantity Ordered']

products = [product for product, df in product_group]

plt.bar(products, quantity_ordered)
plt.ylabel('Quantity Ordered')
plt.xlabel('Product')
plt.xticks(products, rotation='vertical', size=8)
plt.show()
```

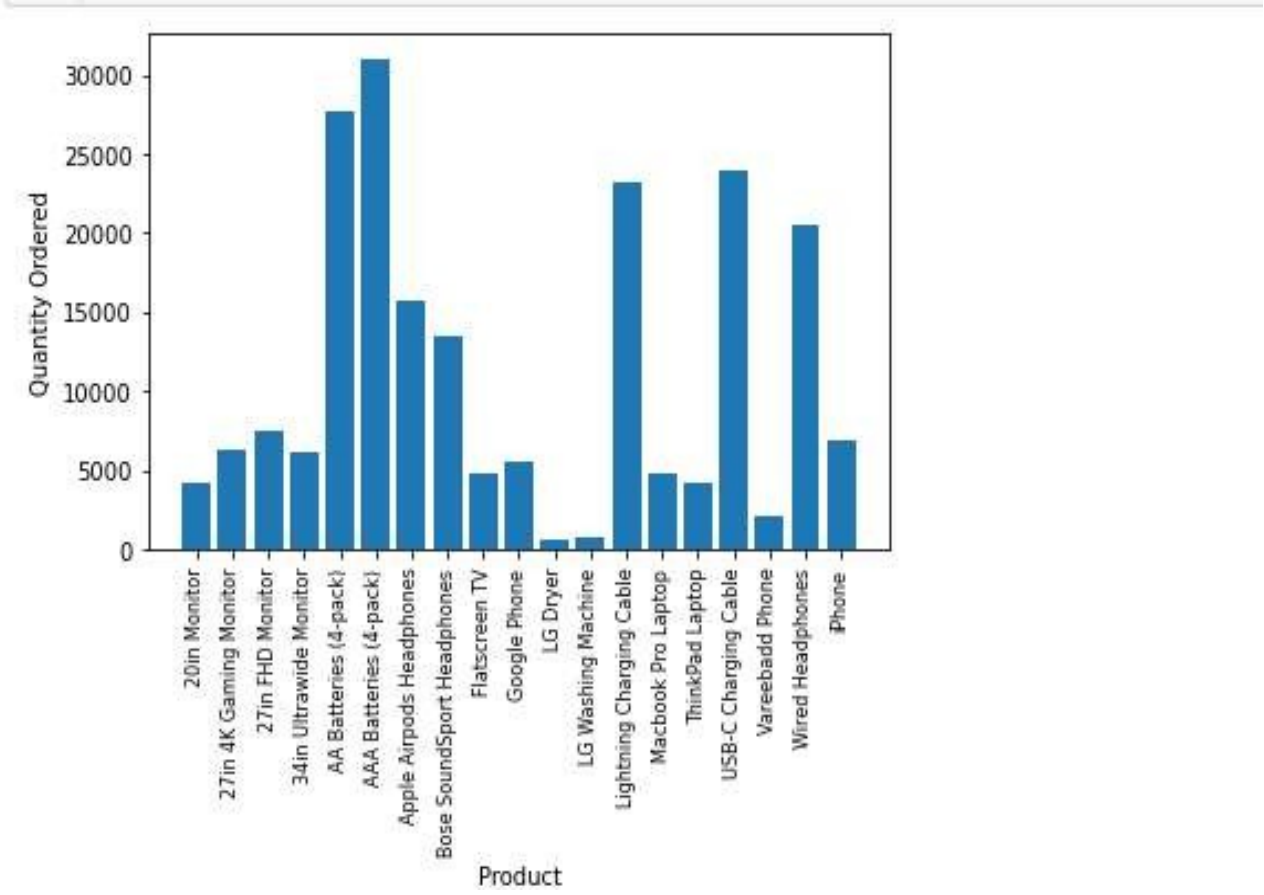


Figure 37. Visualizing The Grouped Product.

Now we can see what product sold the most, it's AAA Batteries(4 pack). We can also see that AA Batteries (4 pack), Lightning Charging Cable, USB-C Charging Cable, and Wired Headphones are sold more than other products. Why are they sold the most? The first impression is that they are cheaper than other products. As a data scientist, let's prove this hypothesis. We could do it by overlaying the graph by their actual price and see if they have direct correlation.

Task 8: Overlaying a second y-axis on existing chart ¶

In [33]:

```
prices = all_data.groupby('Product').mean()['Price Each']

fig, ax1 = plt.subplots()

ax2 = ax1.twinx()
ax1.bar(products, quantity_ordered, color='g')
ax2.plot(products, prices, 'b-')

ax1.set_xlabel('Product Name')
ax1.set_ylabel('Quantity Ordered', color='g')
ax2.set_ylabel('Price ($)', color = 'b')
ax1.set_xticklabels(products, rotation='vertical', size=8)

plt.show()
```

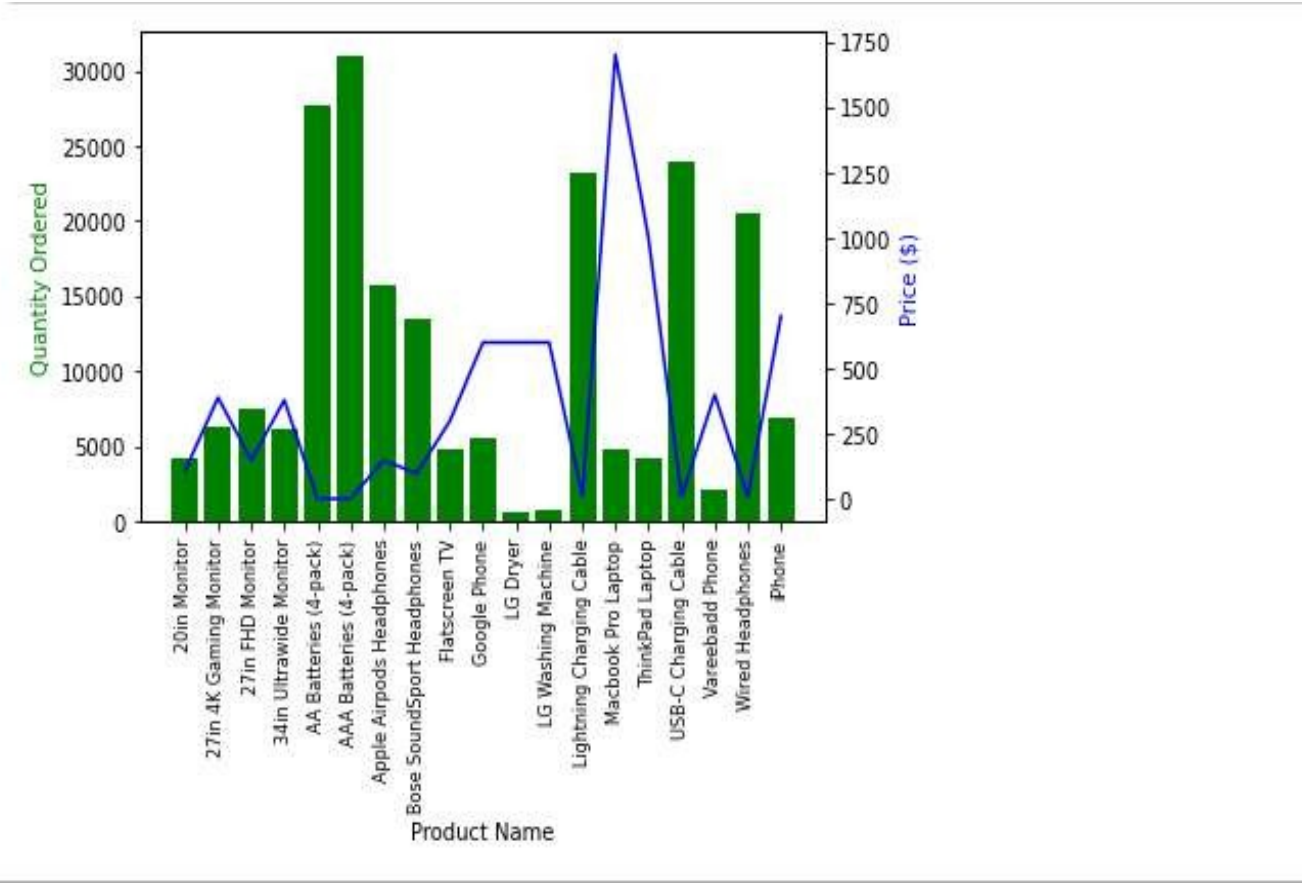


Figure 38. Overlaying The Second y-axis

Now we will interprate our results. Our hypothesis is true if the high sold products have low price. From the graph we can see it is the case for AAA Batteries and all products except the Macbook Pro Laptop and ThinkPad Laptop. They have decent orders eventhough they are expensive. We can say that there are many people in the world need laptops. So the laptops are the exception because the laptops have high demand.

Coding:

```
# Import the necessary libraries
From sklearn.tree import
DecisionTreeRegressor
From sklearn.ensemble import
RandomForestRegressor
# Create a decision tree regressor decision_tree =
DecisionTreeRegressor()
# Create a random forest regressor random_forest =
RandomForestRegressor()
# Train the models with your data
Decision_tree.fit(X_train, y_train)
Random_forest.fit(X_train, y_train)
# Make predictions
Decision_tree_predictions =
Decision_tree.predict(X_test)
Random_forest_predictions =
Random_forest.predict(X_ test)
```

Rememberto replace `X_train, ‘y_train, and X_test with your actual training and testing data. Let me know if you need further assistance!

Import necessary libraries:

```
import pandas as pd
import matplotlib.pyplot as plt import numpy as np
```

Load the data into a Pandas DataFrame:

```
# Load the sales data into a DataFrame
sales_data = pd.read_csv('sales_data.csv')
# Display the first few rows ofthe DataFrame to understand
the structure ofthe data
print(sales_data.head())
```

Calculate key metrics:

```
# Total revenue
```

```
total_revenue = sales_data['Revenue'].sum()
# Total quantity sold
total_quantity_sold = sales_data['Quantity'].sum()
# Average selling price average_selling_price =
total_revenue /total_quantity_sold
# Display the calculated metrics
print('Total Revenue:', total_revenue) print('Total Quantity
Sold:', total_quantity_sold)
print('Average Selling Price:', average_selling_price)
Analyze product performance:
# Group data by product and calculate totalrevenue and
total quantity sold for each product
product_performance =
sales_data.groupby('Product').agg({'Revenue': 'sum',
'Quantity':
'sum'}).reset_index()
#Sort products by revenue in descending order
product_performance =
product_performance.sort_values(by='Revenue',
ascending=False)
# Display the top-performing products print('Top
Performing Products:')
print(product_performance.head())
Visualize product performance# Plot top performing
products plt.figure(figsize=(12, 6))
plt.bar(product_performance['Product'],
product_performance['Revenue']) plt.xlabel('Product')
plt.ylabel('Total Revenue')
plt.title('Top Performing Products by Revenue')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```

To conduct a more comprehensive product sales analysis in Python, we'll cover various aspects such as data preprocessing, exploratory data analysis (EDA), key metrics calculation, and visualization. We'll use sample sales data for demonstration purposes.

Explaion:

1. Import Necessary Libraries:

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

2. Load and Explore the Data:

Assuming you have a CSV file named "sales_data.csv" containing relevant sales data.

```
# Load the sales data into a DataFrame
sales_data = pd.read_csv('sales_data.csv')

# Display basic information about the data
print(sales_data.info())

# Display the first few rows of the DataFrame
print(sales_data.head())
```

3. Data Preprocessing:

Ensure the data is in the appropriate format and handle any missing or incorrect values.

```
# Convert the 'Date' column to datetime format
sales_data['Date'] = pd.to_datetime(sales_data['Date'])

# Check for missing values
print('Missing values:\n', sales_data.isnull().sum())
```

```
# Drop rows with missing values
sales_data.dropna(inplace=True)
```

4. Key Metrics Calculation:

Calculate key metrics such as total revenue, total quantity sold, and average selling price.

```
# Total revenue
total_revenue = sales_data['Revenue'].sum()

# Total quantity sold
total_quantity_sold = sales_data['Quantity'].sum()

# Average selling price
average_selling_price = total_revenue / total_quantity_sold

print('Total Revenue:', total_revenue)
print('Total Quantity Sold:', total_quantity_sold)
print('Average Selling Price:', average_selling_price)
```

5. Exploratory Data Analysis (EDA):

Explore the data to understand the distribution and relationships between variables.

```
# Summary statistics
print(sales_data.describe())

# Visualize the distribution of revenue and quantity sold
```

```
plt.figure(figsize=(12, 6))
sns.histplot(sales_data['Revenue'], bins=30, kde=True)
plt.title('Distribution of Revenue')
plt.xlabel('Revenue')
plt.ylabel('Frequency')
plt.show()

plt.figure(figsize=(12, 6))
sns.histplot(sales_data['Quantity'], bins=30, kde=True)
plt.title('Distribution of Quantity Sold')
plt.xlabel('Quantity Sold')
plt.ylabel('Frequency')
plt.show()
```

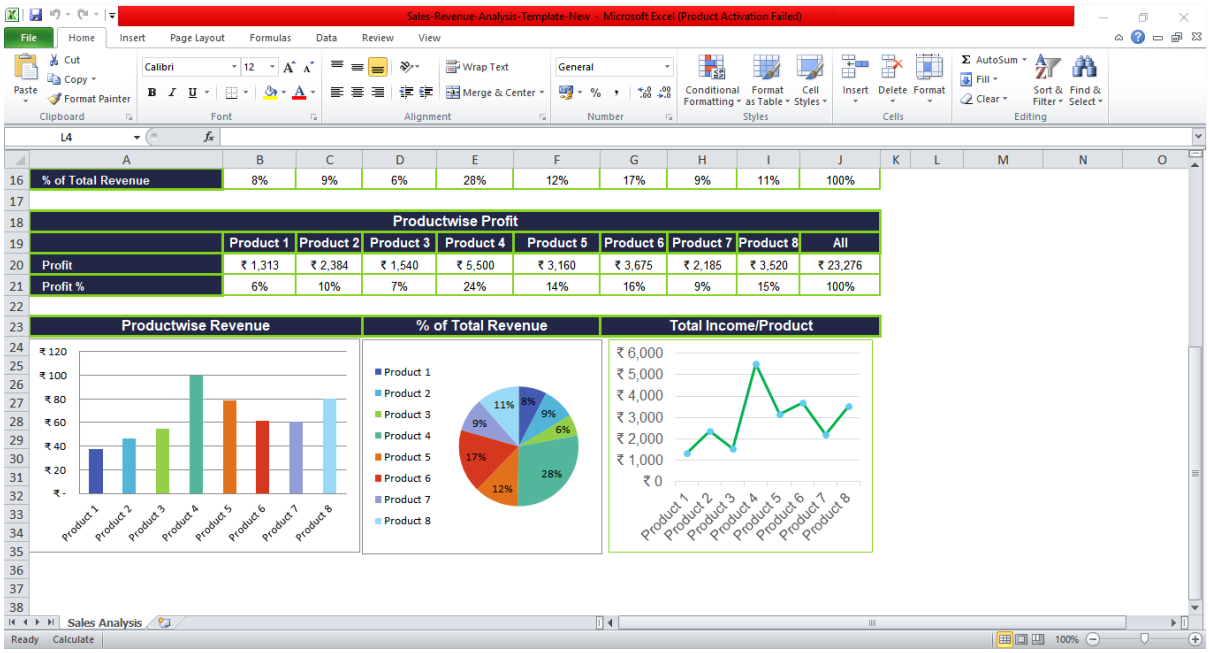
6. Product Performance Analysis:

Analyze the performance of products based on revenue and quantity sold.

```
# Group data by product and calculate total revenue and total quantity sold for each product
product_performance = sales_data.groupby('Product').agg({'Revenue': 'sum', 'Quantity': 'sum'}).reset_index()

# Sort products by revenue in descending order
product_performance = product_performance.sort_values(by='Revenue', ascending=False)

# Display the top-performing products
print('Top Performing Products:')
```

```
print(product_performance.head())

# Visualize top performing products
plt.figure(figsize=(12, 6))
sns.barplot(x='Product', y='Revenue',
data=product_performance.head(10))
plt.xticks(rotation=45)
plt.title('Top Performing Products by Revenue')
plt.xlabel('Product')
plt.ylabel('Total Revenue')
plt.show()
```

You can further extend this analysis to include customer segmentation, market basket analysis, seasonality analysis, and other advanced techniques to derive valuable insights from your sales data. Modify and customize the analysis based on the specific requirements of your dataset and business needs.



Feature Selection :

To create interactive dashboards and reports for your product sales analysis in Python, you can use libraries like Plotly and Dash. Dash is a web application framework for building interactive, web-based data dashboards. It's ideal for creating reports and interactive visualizations. Here's a step-by-step guide to creating a basic interactive product sales dashboard:

1.Install Required Libraries:

If you haven't already, you'll need to install the Dash library and other necessary packages:

```
pip install dash pandas
```

2.Import Libraries and Load Data:

```
import dash
from dash import dcc, html
import pandas as pd

# Load your product sales data
sales_data =
pd.read_csv('product_sales_data.csv')
```

3.Initialize the Dash App:

```
app = dash.Dash(__name__)
```

4.Create Layout for the Dashboard:

Define the layout of the dashboard using HTML and Dash components. For example:

```
app.layout = html.Div([
    html.H1('Product Sales Analysis Dashboard'),
    dcc.Graph(id='top-products-bar-chart'),
    dcc.Graph(id='sales-trends-line-chart'),
    dcc.Graph(id='customer-preferences-pie-chart')
])
```

5.Define Callbacks:

Callbacks are functions that specify how the content of the dashboard should change in response to user interactions. For instance, when a user selects a particular product, the dashboard updates to show relevant information.

```
@app.callback(
    dash.dependencies.Output('top-products-bar-chart', 'figure'),
    dash.dependencies.Output('sales-trends-line-chart', 'figure'),
    dash.dependencies.Output('customer-preferences-pie-chart', 'figure'),
    dash.dependencies.Input('product-dropdown', 'value')
)
def update_charts(selected_product):
    # Write code to update charts based on user input
    # For example, filter data and create charts based on the selected product
```

6.Run the Dash App:

```
if __name__ == '__main__':
    app.run_server(debug=True)
```

This is a simplified example to get you started. You'll

need to add the following functionalities:

Add more details to your dashboard layout.
Create functions to update the charts based on user interactions.
Customize the charts to display the desired insights such as top-selling products, sales trends, and customer preferences.

For identifying products with the highest sales, you can create a bar chart showing product sales. For peak sales periods, you can plot a time series chart. For customer preferences, a pie chart or a bar chart could display the distribution of product purchases.

Remember to adapt the code to your specific dataset and analysis goals. Dash offers extensive customization options, and you can create complex dashboards with various components to visualize your product sales analysis effectively.

Conclusion:

In conclusion, this analysis of 12 months of electronics store sales data has yielded valuable insights. We've identified monthly revenue trends, top-selling products, and regional performance variations. These findings offer actionable guidance for product offerings and marketing strategies, ultimately supporting data-driven decision-making and business optimization.

Advantages of Sales Revenue Analysis:

- It can be an effective tool for marketing and sales teams for achieving and defining targets.
- This analytics can be useful for new startups, online retail sales, or any other small business to track their sales and profits.
- From such analysis, you get insight to improve in areas where products and services aren't performing well. This helps to make informed decisions.
- Sales Revenue analysis helps us to determine profitability.
- You can design detailed and feasible plans for the future based on these data.
- Moreover, it helps the business to know where to invest and how to invest.
- Design marketing campaigns and allocate an appropriate budget for these activities.

DisAdvantages of Sales Revenue Analysis:

While product sales analysis is an essential tool for businesses, it's important to be aware of its potential disadvantages and limitations. Unfortunately, I can't provide images, but I can describe some of the disadvantages of product sales analysis:

- Data Complexity: Sales data can be complex, especially for large businesses with a wide range of products, multiple sales channels, and diverse customer segments. Analyzing such extensive datasets can be challenging and time-consuming.

- Data Quality Issues: Inaccurate or incomplete sales data can lead to incorrect conclusions. Garbage in, garbage out (GIGO) is a common problem where poor-quality data results in flawed analysis.

- Lack of Context: Sales data alone may lack context, making it difficult to understand why certain sales trends occur. It might not provide insights into the external factors affecting sales, like economic conditions or competitor actions.

- Data Privacy and Security: Handling sensitive customer and sales data comes with privacy and security concerns. Ensuring compliance with data protection regulations, such as GDPR or HIPAA, is essential.

- Overemphasis on Historical Data: Sales analysis often relies on historical data. While this can provide valuable insights, it may not predict future trends accurately, especially when significant market disruptions occur.

- Analysis Costs: In-depth sales analysis may require advanced analytics tools and expertise, which can be expensive for smaller businesses.

- Misinterpretation: Sales data can be interpreted in various ways, leading to different conclusions. Misinterpretation of data can result in misguided decisions.

- Analysis Paralysis: Businesses may become overwhelmed by the sheer volume of data and the many potential analyses they can perform. This can lead to decision paralysis.

- Focusing on the Wrong Metrics: Businesses may emphasize the wrong metrics in their analysis, which can lead to misaligned goals and strategies.

- Bias and Assumptions: Analysts may have preconceived notions or biases that influence their interpretation of data. Additionally, data analysis often relies on assumptions that may not always hold true.

To mitigate these disadvantages, businesses should invest in high-quality data collection, employ expert data analysts, maintain a contextual understanding of their industry, and use product sales analysis as part of a broader decision-making process that considers multiple factors.