# CIS 5270 Python Project Topic:

# Kicking It Up a Notch: A Data-Driven Study of Adidas' Sales Performance in America

**Authors: Aishwarya Gupta, Rahul Bhogale**

**Professor: Dr. Shilpa Balan**

## A. INTRODUCTION:

The Adidas sales dataset provides valuable insights into the retail operations of the popular sportswear brand Adidas. This dataset encompasses various key aspects of Adidas sales, including the retailer information, invoice date, geographical details, product details, pricing, units sold, total sales, operating profit, operating margin, and sales method. Analyzing this dataset can help identify trends, patterns, and opportunities for optimizing sales strategies and operational efficiency.

### I. Objective of Python Project Visualization:

The objective of this Python project is to perform comprehensive data visualization and analysis on the Adidas sales dataset. By leveraging Python's powerful libraries and tools for data visualization, we aim to gain a deep understanding of the sales trends, identify potential correlations, and create insightful visualizations that facilitate decision-making processes for stakeholders in the retail industry.

### II. Python Libraries Used:

***Pandas:*** Pandas is a widely used data manipulation library in Python. It provides powerful data structures such as Data Frames to efficiently oversee and analyze structured data.

***Matplotlib:*** Matplotlib is a versatile plotting library in Python that enables the creation of a wide range of static, animated, and interactive visualizations.

***Seaborn:*** Seaborn is a high-level statistical plotting library built on top of Matplotlib. It provides a more streamlined interface for creating attractive statistical visualizations.

***Plotly:*** Plotly is a library that offers interactive and web-based visualizations. It allows for creating dynamic and visually appealing charts, graphs, and dashboards.

***NumPy:*** NumPy is a fundamental library for scientific computing in Python. It provides support for large, multi-dimensional arrays and an extensive collection of mathematical functions.

## III. Project Workflow:

***Data Loading and Preparation:*** We will start by loading the Adidas sales dataset into a pandas Data Frame. This step involves cleaning the data, managing missing values, and ensuring proper data types for analysis.

***Exploratory Data Analysis:*** We will conduct exploratory data analysis to gain initial insights into the dataset. This includes examining summary statistics, identifying outliers, and visualizing the distribution of variables [1].

***Sales Trends and Patterns:*** We will analyze the sales trends over time, examining how sales vary across different regions, states, and cities. Visualizations such as line charts, bar plots, and geographic maps will be employed to highlight the patterns and trends.

***Product Analysis:*** We will delve into the product details, exploring the pricing, units sold, and total sales for various products. Visualizations such as scatter plots, box plots, and histograms will help understand the distribution and performance of various products.

***Operating Profit and Margin Analysis:*** We will investigate the operating profit and operating margin, examining how they vary across retailers, regions, and sales methods. Visualizations such as stacked bar plots and grouped bar charts will aid in comparing these metrics.

***Interactive Visualizations:*** Using Plotly, we will create interactive visualizations that allow users to interact with the data, zoom in on specific details, and obtain additional information on demand.

## IV. Conclusion:

This Python project aims to provide a comprehensive visualization and analysis of the Adidas sales dataset. By leveraging the power of Python libraries such as Pandas, Matplotlib, Seaborn, and Plotly, we will generate meaningful insights and visually appealing representations of the data. The project's outcome will be a valuable resource for retailers, sales teams, and stakeholders in the retail industry to make data-driven decisions, identify areas for improvement, and optimize sales strategies.

By examining sales trends, product performance, and operating metrics, this project will enable a deeper understanding of the factors influencing sales success. The interactive dashboard will empower users [2].

## V. Citations:

[1] https://www.statista.com/topics/1257/adidas/#topicOverview

[2] https://www.grandviewresearch.com/industry-analysis/footwear-market

## B. DATA SET URL's & DESCRIPTION:

### I. URL: https://www.kaggle.com/datasets/heemalichaudhari/adidas-sales-dataset

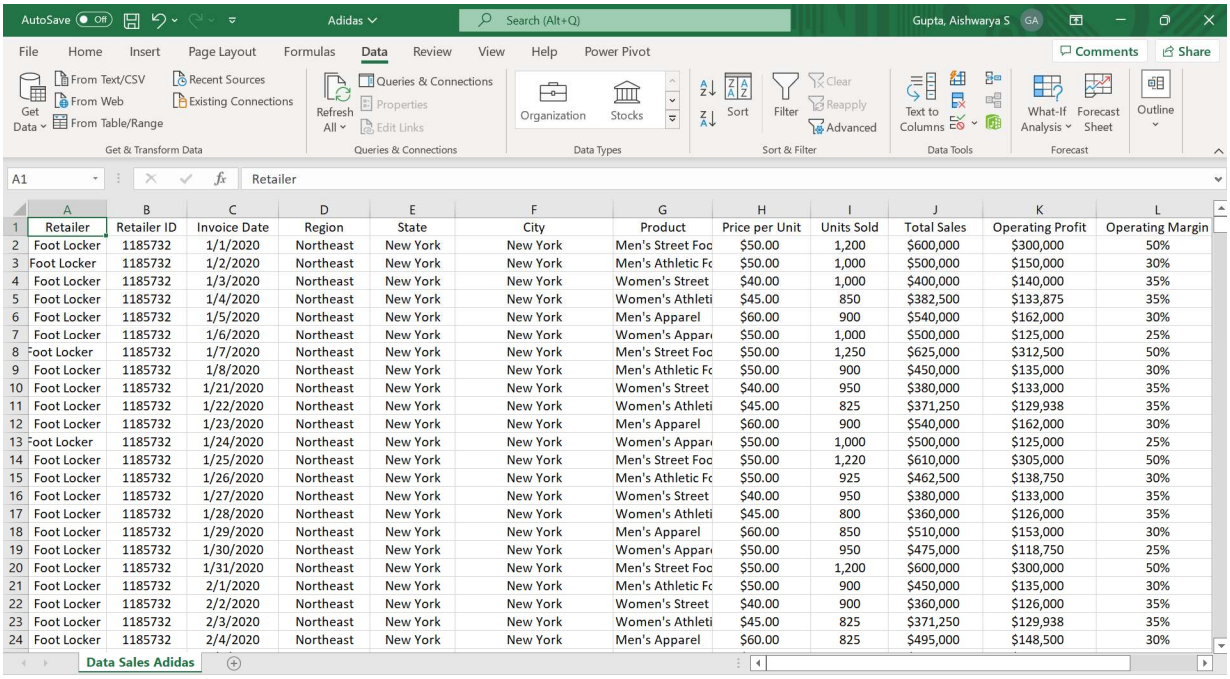
### II. Description:

An Adidas sales dataset consists of information about the sales of Adidas products, including sales revenue, units sold, product types, and location data. It is used for analyzing sales trends, evaluating product performance, and developing sales strategies. The dataset can be obtained from sources such as Adidas, market research firms, or government agencies.

The dataset you provided contains several columns. Here is a description of each column.

| Fieldname | Description | Example |
|---|---|---|
| Retailer | This column represents the name or identifier of the retailer. It is a categorical variable that helps identify different retailers in the dataset. | Foot Locker |
| | | |
| Retailer ID | This column contains a unique identifier for each retailer. It is often a numerical value assigned to each retailer for identification purposes. | 1185732 |
| | | |
| Invoice Date | This column represents the date on which the invoice or sales transaction took place. It typically includes the day, month, and year of the transaction. | 1/1/2020 |
| | | |
| Region | This column indicates the geographic region where the retailer operates. It is a categorical variable that helps categorize retailers based on their operating locations. | Northeast |
| | | |
| State | This column represents the state in which the retailer is located. It is a categorical variable that provides more specific information about the retailer's operating location within a region. | New York |
| | | |
| City | This column indicates the city in which the retailer is located. It is a categorical variable that provides further granularity about the retailer's operating location within a state. | New York |
| | | |
| Product | This column specifies the name or identifier of the product sold by the retailer. It is a categorical variable that helps identify the specific product associated with each transaction. | Men's Street Footwear |
| | | |

| | | |
|---|---|---|
| Price per Unit | This column represents the price of a single unit of the product sold by the retailer. It is a numerical variable typically denoted in the currency used in the dataset. | $50.00 |
| | | |
| Units Sold | This column indicates the number of units of the product sold in a specific transaction. It is a numerical variable that represents the quantity of the product sold. | 1,200 |
| | | |
| Total Sales | This column calculates the total sales generated by multiplying the price per unit by the number of units sold in a transaction. It is a numerical variable that represents the overall revenue generated by the retailer. | $600,000 |
| | | |
| Operating Profit | This column represents the profit earned by the retailer from the sale of the product. It is a numerical variable that reflects the difference between the total sales and the retailer's operating costs. | $300,000 |
| | | |
| Operating Margin | This column calculates the operating profit as a percentage of the total sales. It is a numerical variable that represents the profitability of the retailer's operations. | 50% |
| | | |
| Sales Method | This column indicates the method or channel through which the sales transaction took place. It is a categorical variable that helps identify the sales channel used by the retailer, such as in-store, online, or wholesale. | In-store |



| Retailer | Retailer ID | Invoice Date | Region | State | City | Product | Price per Unit | Units Sold | Total Sales | Operating Profit | Operating Margin |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Foot Locker | 1185732 | 1/1/2020 | Northeast | New York | New York | Men's Street Foc | $50.00 | 1,200 | $600,000 | $300,000 | 50% |
| Foot Locker | 1185732 | 1/2/2020 | Northeast | New York | New York | Men's Athletic Fc | $50.00 | 1,000 | $500,000 | $150,000 | 30% |
| Foot Locker | 1185732 | 1/3/2020 | Northeast | New York | New York | Women's Street | $40.00 | 1,000 | $400,000 | $140,000 | 35% |
| Foot Locker | 1185732 | 1/4/2020 | Northeast | New York | New York | Women's Athleti | $45.00 | 850 | $382,500 | $133,875 | 35% |
| Foot Locker | 1185732 | 1/5/2020 | Northeast | New York | New York | Men's Apparel | $60.00 | 900 | $540,000 | $162,000 | 30% |
| Foot Locker | 1185732 | 1/6/2020 | Northeast | New York | New York | Women's Appar | $50.00 | 1,000 | $500,000 | $125,000 | 25% |
| Foot Locker | 1185732 | 1/7/2020 | Northeast | New York | New York | Men's Street Foc | $50.00 | 1,250 | $625,000 | $312,500 | 50% |
| Foot Locker | 1185732 | 1/8/2020 | Northeast | New York | New York | Men's Athletic Fc | $50.00 | 900 | $450,000 | $135,000 | 30% |
| Foot Locker | 1185732 | 1/21/2020 | Northeast | New York | New York | Women's Street | $40.00 | 950 | $380,000 | $133,000 | 35% |
| Foot Locker | 1185732 | 1/22/2020 | Northeast | New York | New York | Women's Athleti | $45.00 | 825 | $371,250 | $129,938 | 35% |
| Foot Locker | 1185732 | 1/23/2020 | Northeast | New York | New York | Men's Apparel | $60.00 | 900 | $540,000 | $162,000 | 30% |
| Foot Locker | 1185732 | 1/24/2020 | Northeast | New York | New York | Women's Appar | $50.00 | 1,000 | $500,000 | $125,000 | 25% |
| Foot Locker | 1185732 | 1/25/2020 | Northeast | New York | New York | Men's Street Foc | $50.00 | 1,220 | $610,000 | $305,000 | 50% |
| Foot Locker | 1185732 | 1/26/2020 | Northeast | New York | New York | Men's Athletic Fc | $50.00 | 925 | $462,500 | $138,750 | 30% |
| Foot Locker | 1185732 | 1/27/2020 | Northeast | New York | New York | Women's Street | $40.00 | 950 | $380,000 | $133,000 | 35% |
| Foot Locker | 1185732 | 1/28/2020 | Northeast | New York | New York | Women's Athleti | $45.00 | 800 | $360,000 | $126,000 | 35% |
| Foot Locker | 1185732 | 1/29/2020 | Northeast | New York | New York | Men's Apparel | $60.00 | 850 | $510,000 | $153,000 | 30% |
| Foot Locker | 1185732 | 1/30/2020 | Northeast | New York | New York | Women's Appar | $50.00 | 950 | $475,000 | $118,750 | 25% |
| Foot Locker | 1185732 | 1/31/2020 | Northeast | New York | New York | Men's Street Foc | $50.00 | 1,200 | $600,000 | $300,000 | 50% |
| Foot Locker | 1185732 | 2/1/2020 | Northeast | New York | New York | Men's Athletic Fc | $50.00 | 900 | $450,000 | $135,000 | 30% |
| Foot Locker | 1185732 | 2/2/2020 | Northeast | New York | New York | Women's Street | $40.00 | 900 | $360,000 | $126,000 | 35% |
| Foot Locker | 1185732 | 2/3/2020 | Northeast | New York | New York | Women's Athleti | $45.00 | 825 | $371,250 | $129,938 | 35% |
| Foot Locker | 1185732 | 2/4/2020 | Northeast | New York | New York | Men's Apparel | $60.00 | 825 | $495,000 | $148,500 | 30% |

*(Dataset screenshot)*

## C. DATA CLEANING:

### I. Handling Missing Values:

As we can see from the below screenshot, the dataset was having ***null values*** for column ***'Price per Unit.'***

Thus, the ***null values*** are being replaced by the ***mean value*** of the entire column.

| F | G | H | I | J | K | L |
|---|---|---|---|---|---|---|
| City | Product | Price per Un | Units Sol | Total Sales | Operating Prof | Operating Margi |
| Milwaukee | Women's Street Footwear | | 150 | $15,000 | $4,500 | 30% |
| Birmingham | Men's Street Footwear | | 203 | $1,827 | $1,078 | 59% |
| Salt Lake City | Women's Street Footwear | | 91 | $819 | $410 | 50% |
| Salt Lake City | Women's Athletic Footwear | | 88 | $880 | $449 | 51% |
| Little Rock | Women's Street Footwear | | 88 | $880 | $475 | 54% |
| Oklahoma City | Women's Street Footwear | | 80 | $720 | $360 | 50% |
| Wichita | Women's Street Footwear | | 75 | $675 | $311 | 46% |
| Des Moines | Women's Street Footwear | | 65 | $585 | $234 | 40% |
| Des Moines | Women's Street Footwear | | 53 | $530 | $233 | 44% |
| Milwaukee | Women's Street Footwear | | 59 | $531 | $234 | 44% |
| Milwaukee | Women's Street Footwear | | 41 | $369 | $159 | 43% |
| Birmingham | Men's Street Footwear | | 224 | $1,568 | $737 | 47% |
| Salt Lake City | Women's Street Footwear | | 107 | $1,070 | $439 | 41% |

The below figure shows the code to fill null or NA values in python using ***'df.fillna()'*** function and here is the code mentioned below.

```
# 1. fill NaN values with mean values for price per unit columns
mean_price = df['Price per Unit'].mean()  # Calculate the mean of the 'Price per Unit' column
df['Price per Unit'].fillna(mean_price, inplace=True)  # Fill missing values with the mean value
df.head()
```

```
In [4]: # 1.fill na with mean values for price per unit columns
        mean_price = df['Price per Unit'].mean()  # Calculate the mean of the 'Price per Unit' column
        df['Price per Unit'].fillna(mean_price, inplace=True)  # Fill missing values with the mean value
        df.head()
```

Out[4]:

| | Retailer | Retailer ID | Invoice Date | Region | State | City | Product | Price per Unit | Units Sold | Total Sales | Operating Profit | Operating Margin | Sales Method |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | Foot Locker | 1185732 | 2020-01-01 | Northeast | New York | New York | Men's Street Footwear | 50.0 | 1200 | 600000.0 | 300000.0 | 0.50 | In-store |
| 1 | Foot Locker | 1185732 | 2020-01-02 | Northeast | New York | New York | Men's Athletic Footwear | 50.0 | 1000 | 500000.0 | 150000.0 | 0.30 | In-store |
| 2 | Foot Locker | 1185732 | 2020-01-03 | Northeast | New York | New York | Women's Street Footwear | 40.0 | 1000 | 400000.0 | 140000.0 | 0.35 | In-store |
| 3 | Foot Locker | 1185732 | 2020-01-04 | Northeast | New York | New York | Women's Athletic Footwear | 45.0 | 850 | 382500.0 | 133875.0 | 0.35 | In-store |
| 4 | Foot Locker | 1185732 | 2020-01-05 | Northeast | New York | New York | Men's Apparel | 60.0 | 900 | 540000.0 | 162000.0 | 0.30 | In-store |

## II. Replacing and Formatting values:

As we can see from the below figure, ***'Total Sales' and 'Operating Profit'*** values are shown incorrectly. i.e., ***$0.***

| H | I | J | K | L |
|---|---|---|---|---|
| Price per Un | Units Sol | Total Sales | Operating Prof | Operating Margi |
| $35.00 | 3 | $0 | $0 | 40% |
| $30.00 | 3 | $0 | $0 | 40% |
| $33.00 | 3 | $0 | $0 | 55% |
| $27.00 | 3 | $0 | $0 | 53% |

To rectify this, we replaced the values using formula and formatted the data type of these two columns as integer ***using function: astype(int)*** shown from the python code below.

| **# 2. replace zero values with formula and formatting into int** |
|---|
| df['Total Sales'] = (df['Price per Unit'] * df['Units Sold']).astype(int) |
| df['Operating Profit'] = (df['Operating Margin'] * df['Total Sales']).astype(int) |
| df.head() |

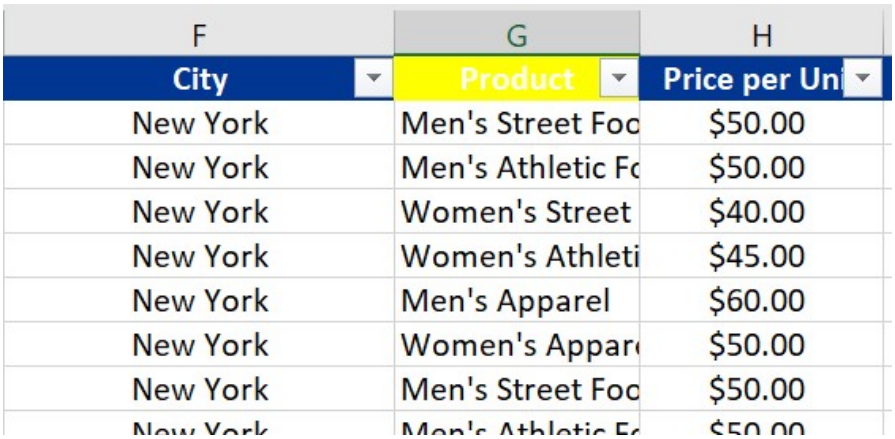
```
In [5]: # 2.replace 0 values with formula and formatting into int
        df['Total Sales'] = (df['Price per Unit'] * df['Units Sold']).astype(int)
        df['Operating Profit'] = (df['Operating Margin'] * df['Total Sales']).astype(int)

        df.head()
```

Out[5]:

| | Retailer | Retailer ID | Invoice Date | Region | State | City | Product | Price per Unit | Units Sold | Total Sales | Operating Profit | Operating Margin | Sales Method |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | Foot Locker | 1185732 | 2020-01-01 | Northeast | New York | New York | Men's Street Footwear | 50.0 | 1200 | 60000 | 30000 | 0.50 | In-store |
| 1 | Foot Locker | 1185732 | 2020-01-02 | Northeast | New York | New York | Men's Athletic Footwear | 50.0 | 1000 | 50000 | 15000 | 0.30 | In-store |
| 2 | Foot Locker | 1185732 | 2020-01-03 | Northeast | New York | New York | Women's Street Footwear | 40.0 | 1000 | 40000 | 14000 | 0.35 | In-store |
| 3 | Foot Locker | 1185732 | 2020-01-04 | Northeast | New York | New York | Women's Athletic Footwear | 45.0 | 850 | 38250 | 13387 | 0.35 | In-store |
| 4 | Foot Locker | 1185732 | 2020-01-05 | Northeast | New York | New York | Men's Apparel | 60.0 | 900 | 54000 | 16200 | 0.30 | In-store |

## III. Splitting Columns:

From the below figure, column ***'Product' has distinct categories for Men and Women***. Thus, we have ***split this column into 'Gender' using delimiter ''s'*** using ***string function str.split(" 's ")*** as shown below from the python code.



| F | G | H |
|---|---|---|
| City | Product | Price per Uni |
| New York | Men's Street Foc | $50.00 |
| New York | Men's Athletic Fc | $50.00 |
| New York | Women's Street | $40.00 |
| New York | Women's Athleti | $45.00 |
| New York | Men's Apparel | $60.00 |
| New York | Women's Appar | $50.00 |
| New York | Men's Street Foc | $50.00 |
| New York | Men's Athletic Fc | $50.00 |

(1)

| **# 3. split the columns** |
|---|
| df['Gender'] = df['Product'].str.split("'s").str[0] |
| df.head() |

```
# 3.split the columns
df['Gender'] = df['Product'].str.split("'s").str[0]
df.head()
```

| | Retailer | Retailer ID | Invoice Date | Region | State | City | Product | Price per Unit | Units Sold | Total Sales | Operating Profit | Operating Margin | Sales Method | Gender |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | Foot Locker | 1185732 | 2020-01-01 | Northeast | New York | New York | Men's Street Footwear | 50.0 | 1200 | 60000 | 30000 | 0.50 | In-store | Men |
| 1 | Foot Locker | 1185732 | 2020-01-02 | Northeast | New York | New York | Men's Athletic Footwear | 50.0 | 1000 | 50000 | 15000 | 0.30 | In-store | Men |
| 2 | Foot Locker | 1185732 | 2020-01-03 | Northeast | New York | New York | Women's Street Footwear | 40.0 | 1000 | 40000 | 14000 | 0.35 | In-store | Women |
| 3 | Foot Locker | 1185732 | 2020-01-04 | Northeast | New York | New York | Women's Athletic Footwear | 45.0 | 850 | 38250 | 13387 | 0.35 | In-store | Women |
| 4 | Foot Locker | 1185732 | 2020-01-05 | Northeast | New York | New York | Men's Apparel | 60.0 | 900 | 54000 | 16200 | 0.30 | In-store | Men |

(2)

## IV. Value Formatting:

From the below snapshot, we observe that ***'Invoice Date' column has string values***.

Thus, we have extracted year from this value by converting the current datatype ***to_datetime*** format and later extract year from it using ***dt.year()*** function.

| A | B | C | D |
|---|---|---|---|
| Retailer | Retailer I | Invoice Dat | Region |
| Foot Locker | 1185732 | 1/1/2020 | Northeast |
| Foot Locker | 1185732 | 1/2/2020 | Northeast |
| Foot Locker | 1185732 | 1/3/2020 | Northeast |
| Foot Locker | 1185732 | 1/4/2020 | Northeast |
| Foot Locker | 1185732 | 1/5/2020 | Northeast |

Python code to extract year from date timestamp as follows:

| **# 4. Data values formatting** |
|---|
| df['Invoice Date'] = pd.to_datetime(df['Invoice Date'])  # Convert 'Invoice Date' to datetime format |
| df['Year'] = df['Invoice Date'].dt.year  # Extract the year |
| df.head() |

```
In [7]: # 4.Data values formatting
        df['Invoice Date'] = pd.to_datetime(df['Invoice Date'])  # Convert 'Invoice Date' to datetime format
        df['Year'] = df['Invoice Date'].dt.year  # Extract the year

        df.head()
```

Out[7]:

| | Retailer | Retailer ID | Invoice Date | Region | State | City | Product | Price per Unit | Units Sold | Total Sales | Operating Profit | Operating Margin | Sales Method | Gender | Year |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | Foot Locker | 1185732 | 2020-01-01 | Northeast | New York | New York | Men's Street Footwear | 50.0 | 1200 | 60000 | 30000 | 0.50 | In-store | Men | 2020 |
| 1 | Foot Locker | 1185732 | 2020-01-02 | Northeast | New York | New York | Men's Athletic Footwear | 50.0 | 1000 | 50000 | 15000 | 0.30 | In-store | Men | 2020 |
| 2 | Foot Locker | 1185732 | 2020-01-03 | Northeast | New York | New York | Women's Street Footwear | 40.0 | 1000 | 40000 | 14000 | 0.35 | In-store | Women | 2020 |
| 3 | Foot Locker | 1185732 | 2020-01-04 | Northeast | New York | New York | Women's Athletic Footwear | 45.0 | 850 | 38250 | 13387 | 0.35 | In-store | Women | 2020 |
| 4 | Foot Locker | 1185732 | 2020-01-05 | Northeast | New York | New York | Men's Apparel | 60.0 | 900 | 54000 | 16200 | 0.30 | In-store | Men | 2020 |

## V. Trim Spaces:

From the below figure, we observe that, the values in ***'Retailer' column have leading or trailing spaces.***

Thus, to remove such unwanted spaces, we used ***strip ()*** function in python code and screenshot of before trimming and after trimming has been show below.

***Before Trimming***: The ***len ()*** function is used to identify if we have any extra spaces and could be seen in the figure (2) with length =***14***.

***After Trimming***: The length of data value should be ***11***. Thus, we have achieved it using strip () func and snapshots are attached below.

| A | B | C | D |
|---|---|---|---|
| Retailer | Retailer I | Invoice Dat | Region |
| Foot Locker | 1185732 | 1/1/2020 | Northeast |
| Foot Locker | 1185732 | 1/2/2020 | Northeast |
| Foot Locker | 1185732 | 1/3/2020 | Northeast |
| Foot Locker | 1185732 | 1/4/2020 | Northeast |
| Foot Locker | 1185732 | 1/5/2020 | Northeast |
| Foot Locker | 1185732 | 1/6/2020 | Northeast |
| Foot Locker | 1185732 | 1/7/2020 | Northeast |

```
In [8]: # 5. Trim spaces
        #before trimming
        df['length of retailer'] = df['Retailer'].str.len()
        df.head()
```

Out[8]:

| | Retailer | Retailer ID | Invoice Date | Region | State | City | Product | Price per Unit | Units Sold | Total Sales | Operating Profit | Operating Margin | Sales Method | Gender | Year | length of retailer |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | Foot Locker | 1185732 | 2020-01-01 | Northeast | New York | New York | Men's Street Footwear | 50.0 | 1200 | 60000 | 30000 | 0.50 | In-store | Men | 2020 | 11 |
| 1 | Foot Locker | 1185732 | 2020-01-02 | Northeast | New York | New York | Men's Athletic Footwear | 50.0 | 1000 | 50000 | 15000 | 0.30 | In-store | Men | 2020 | 14 |
| 2 | Foot Locker | 1185732 | 2020-01-03 | Northeast | New York | New York | Women's Street Footwear | 40.0 | 1000 | 40000 | 14000 | 0.35 | In-store | Women | 2020 | 11 |
| 3 | Foot Locker | 1185732 | 2020-01-04 | Northeast | New York | New York | Women's Athletic Footwear | 45.0 | 850 | 38250 | 13387 | 0.35 | In-store | Women | 2020 | 11 |
| 4 | Foot Locker | 1185732 | 2020-01-05 | Northeast | New York | New York | Men's Apparel | 60.0 | 900 | 54000 | 16200 | 0.30 | In-store | Men | 2020 | 11 |

**(2)**

| # 5. After trimming spaces |
| --- |
| df['Retailer1'] = df['Retailer'].str.strip() |
| df['length of retailer1'] = df['Retailer1'].str.len() |
| df.head() |

In [9]:
```
#After trimming spaces
df['Retailer1'] = df['Retailer'].str.strip()
df['length of retailer1'] = df['Retailer1'].str.len()
df.head()
```

Out[9]:

| | Retailer | Retailer ID | Invoice Date | Region | State | City | Product | Price per Unit | Units Sold | Total Sales | Operating Profit | Operating Margin | Sales Method | Gender | Year | length of retailer | Retailer1 | length of retailer1 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | Foot Locker | 1185732 | 2020-01-01 | Northeast | New York | New York | Men's Street Footwear | 50.0 | 1200 | 60000 | 30000 | 0.50 | In-store | Men | 2020 | 11 | Foot Locker | 11 |
| 1 | Foot Locker | 1185732 | 2020-01-02 | Northeast | New York | New York | Men's Athletic Footwear | 50.0 | 1000 | 50000 | 15000 | 0.30 | In-store | Men | 2020 | 14 | Foot Locker | 11 |
| 2 | Foot Locker | 1185732 | 2020-01-03 | Northeast | New York | New York | Women's Street Footwear | 40.0 | 1000 | 40000 | 14000 | 0.35 | In-store | Women | 2020 | 11 | Foot Locker | 11 |
| 3 | Foot Locker | 1185732 | 2020-01-04 | Northeast | New York | New York | Women's Athletic Footwear | 45.0 | 850 | 38250 | 13387 | 0.35 | In-store | Women | 2020 | 11 | Foot Locker | 11 |
| 4 | Foot Locker | 1185732 | 2020-01-05 | Northeast | New York | New York | Men's Apparel | 60.0 | 900 | 54000 | 16200 | 0.30 | In-store | Men | 2020 | 11 | Foot Locker | 11 |

## D. STATISTICAL SUMMARY:

### I. Statistical information of dataset.

In the below figure, we are describing the statistical metrics such as ***count, min, max, std, mean, quartiles*** etc. of all the numerical columns present in entire dataset.

This is achieved by using ***df.describe()*** func as shown below.

| **# Statistical information** |
|---|
| df.describe() |

```
In [10]: # Statistical information
         df.describe()
```

Out[10]:

| | Retailer ID | Price per Unit | Units Sold | Total Sales | Operating Profit | Operating Margin | Year | length of retailer | length of retailer1 |
|---|---|---|---|---|---|---|---|---|---|
| count | 9.648000e+03 | 9648.000000 | 9648.000000 | 9648.000000 | 9648.000000 | 9648.000000 | 9648.000000 | 9648.000000 | 9648.000000 |
| mean | 1.173850e+06 | 45.265179 | 256.931281 | 12460.060531 | 4896.768346 | 0.422991 | 2020.865050 | 9.645315 | 9.643968 |
| std | 2.636038e+04 | 14.645833 | 214.250547 | 12712.861367 | 4865.008221 | 0.097197 | 0.341688 | 2.482827 | 2.480859 |
| min | 1.128299e+06 | 7.000000 | 3.000000 | 81.000000 | 36.000000 | 0.100000 | 2020.000000 | 6.000000 | 6.000000 |
| 25% | 1.185732e+06 | 35.000000 | 106.000000 | 4077.750000 | 1759.500000 | 0.350000 | 2021.000000 | 7.000000 | 7.000000 |
| 50% | 1.185732e+06 | 45.000000 | 176.000000 | 7812.000000 | 3264.000000 | 0.410000 | 2021.000000 | 9.000000 | 9.000000 |
| 75% | 1.185732e+06 | 55.000000 | 350.000000 | 15864.500000 | 6192.250000 | 0.490000 | 2021.000000 | 11.000000 | 11.000000 |
| max | 1.197831e+06 | 110.000000 | 1275.000000 | 82500.000000 | 39000.000000 | 0.800000 | 2021.000000 | 16.000000 | 13.000000 |

### II. Statistical Q1:

Here, we are analyzing the statistical metrics of ***'Price per Unit' column for State as 'New York.'***

For this, we have performed ***aggregation using 'group by' command for 'State'*** column to get ***Average price per unit of shoes sold across entire New York state.***

| **#Statistical func 1: col: Price per Unit** |
|---|
| from scipy import stats |

```
# Filter dataframe for New York state
ny_df = df[df['State'] == 'New York']
# Group by state and calculate the descriptive statistics for 'Price per Unit'
grouped_stats = ny_df.groupby('State')['Price per Unit'].describe()
# Print the descriptive statistics
print(grouped_stats)
# Calculate mode for 'Price per Unit'
mode = stats.mode(ny_df['Price per Unit'])
print("Mode:", mode.mode[0])
# Calculate median for 'Price per Unit'
median = ny_df['Price per Unit'].median()
print("Median:", median)
```

In [13]:
```
#statistical func 1: col: Price per Unit
from scipy import stats

# Filter dataframe for New York state
ny_df = df[df['State'] == 'New York']

# Group by state and calculate the descriptive statistics for 'Price per Unit'
grouped_stats = ny_df.groupby('State')['Price per Unit'].describe()

# Print the descriptive statistics
print(grouped_stats)

# Calculate mode for 'Price per Unit'
mode = stats.mode(ny_df['Price per Unit'])
print("Mode:", mode.mode[0])

# Calculate median for 'Price per Unit'
median = ny_df['Price per Unit'].median()
print("Median:", median)
```

```
          count       mean        std   min    25%   50%   75%   max
State
New York  360.0  49.111111  10.247026  20.0  41.75  50.0  56.0  70.0
Mode: 50.0
Median: 50.0
```

## III. Statistical Q2:

Here, we are analyzing ***Average Operating Margin performed during Online shopping method*** for Adidas Sales shoes.

This is achieved by using filter on ***Sales Method column = 'Online'*** and calculating different metrics such as mean, median, mode, min, max, std, count, quartiles etc.

| **#Statistical func 2:** |
|---|
| # Filter dataframe for Sales Method = Online |
| online_df = df[df['Sales Method'] == 'Online'] |
| # Calculate operating margin |
| operating_margin = online_df['Operating Margin'] |
| # Describe operating margin |
| operating_margin_stats = operating_margin.describe() |
| print("Operating Margin Description:") |
| print(operating_margin_stats) |
| # Calculate mode for operating margin |
| operating_margin_mode = stats.mode(operating_margin) |
| print("Operating Margin Mode:", operating_margin_mode.mode[0]) |
| # Calculate median for operating margin |
| operating_margin_median = operating_margin.median() |
| print("Operating Margin Median:", operating_margin_median) |

```
In [15]: #statistical func 2:

         # Filter dataframe for Sales Method = Online
         online_df = df[df['Sales Method'] == 'Online']

         # Calculate operating margin
         operating_margin = online_df['Operating Margin']

         # Describe operating margin
         operating_margin_stats = operating_margin.describe()
         print("Operating Margin Description:")
         print(operating_margin_stats)

         # Calculate mode for operating margin
         operating_margin_mode = stats.mode(operating_margin)
         print("Operating Margin Mode:", operating_margin_mode.mode[0])

         # Calculate median for operating margin
         operating_margin_median = operating_margin.median()
         print("Operating Margin Median:", operating_margin_median)
```

```
Operating Margin Description:
count    4889.000000
mean        0.464152
std         0.088536
min         0.210000
25%         0.400000
50%         0.470000
75%         0.520000
max         0.800000
Name: Operating Margin, dtype: float64
Operating Margin Mode: 0.4
Operating Margin Median: 0.47
```

## IV. STATISTICAL Q3:

In this, we are analyzing ***'Total Sales' for the Year =2021*** for different Adidas products that are sold.

We are using ***describe ()*** func to calculate the following metrics such as min, max, count, mean, median, mode, std, quartiles etc.

| **#Statistical func 3:** |
| --- |
| # Filter dataframe for Year = 2021 |
| df_2021 = df[df['Year'] == 2021] |

```
# Calculate total sales for 2021
total_sales_2021 = df_2021['Total Sales']
# Describe total sales for 2021
total_sales_stats = total_sales_2021.describe()
print("Total Sales Description for 2021:")
print(total_sales_stats)
# Calculate mode for total sales in 2021
total_sales_mode = stats.mode(total_sales_2021)
print("Total Sales Mode for 2021:", total_sales_mode.mode[0])
# Calculate median for total sales in 2021
total_sales_median = total_sales_2021.median()
print("Total Sales Median for 2021:", total_sales_median)
```

```
In [16]: #statistical func 3:

         # Filter dataframe for Year = 2021
         df_2021 = df[df['Year'] == 2021]

         # Calculate total sales for 2021
         total_sales_2021 = df_2021['Total Sales']

         # Describe total sales for 2021
         total_sales_stats = total_sales_2021.describe()
         print("Total Sales Description for 2021:")
         print(total_sales_stats)

         # Calculate mode for total sales in 2021
         total_sales_mode = stats.mode(total_sales_2021)
         print("Total Sales Mode for 2021:", total_sales_mode.mode[0])

         # Calculate median for total sales in 2021
         total_sales_median = total_sales_2021.median()
         print("Total Sales Median for 2021:", total_sales_median)
```

```
Total Sales Description for 2021:
count     8346.000000
mean     11499.803618
std      11777.430554
min         81.000000
25%       3780.000000
50%       7135.000000
75%      14778.000000
max      82500.000000
Name: Total Sales, dtype: float64
Total Sales Mode for 2021: 10000
Total Sales Median for 2021: 7135.0
```

## E. ANALYSIS AND VISUALIZATION

### 1. Which region has the highest revenue and operating profit, and which region has the lowest?

- Here we are analyzing an Adidas Sales dataset to determine which region has the highest revenue and operating profit, as well as the region with the lowest revenue and operating profit.
- To do this, we first filter the dataset to exclude any missing values in the 'Total Sales' and 'Operating Profit' columns.
- Then, we calculate the sum of total sales and operating profit for each region using the group by function.
- Next, we sort the data in descending order based on total sales and operating profit.
- We visualize the results using bar plots to compare the total sales and operating profit for each region.
- Finally, we identify the region with the highest and lowest revenue and operating profit based on the sorted data.

***Conclusion:***

- The West region has the highest revenue and operating profit, with approximately $3.5 million in revenue and $1.2 million in operating profit.
- On the other hand, the Midwest region has the lowest revenue and operating profit, with approximately $1.6 million in revenue and $0.7 million in operating profit.

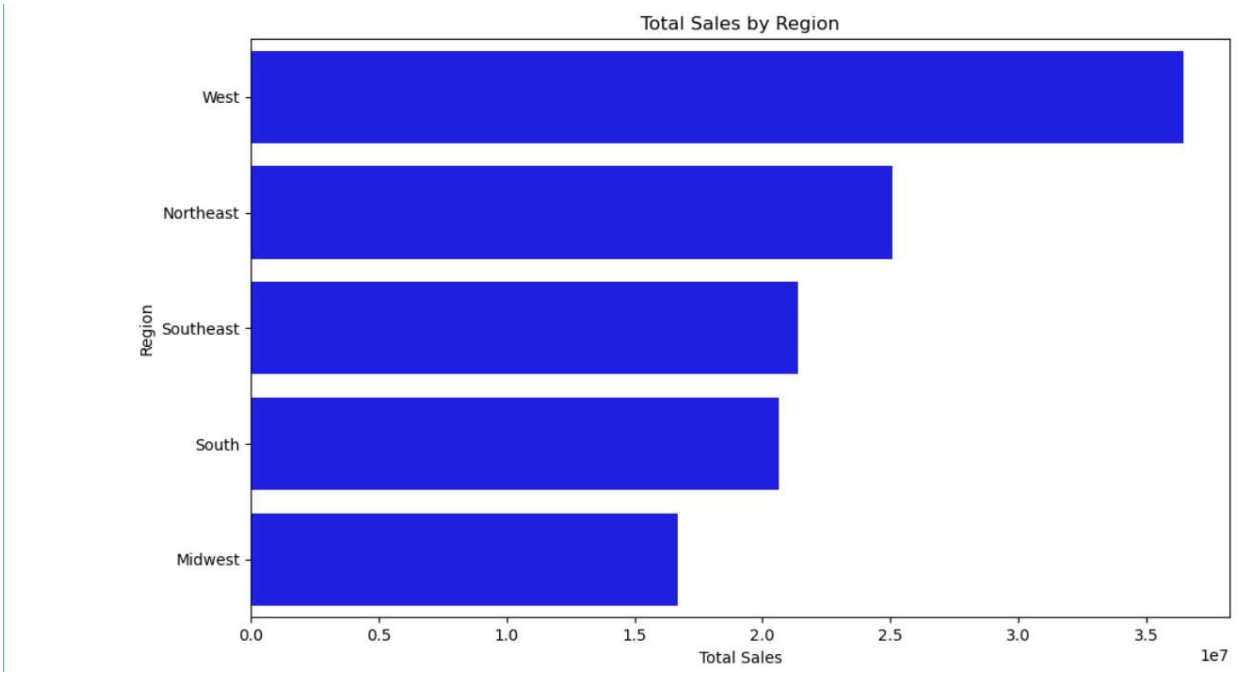
```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
# Assuming you have a DataFrame named df with the necessary columns
# Filter the data to exclude any missing values in Total Sales and Operating Profit columns
filtered_df = df.dropna(subset=['Total Sales', 'Operating Profit'])
# Calculate the sum of Total Sales and Operating Profit by Region
revenue_profit_by_region = filtered_df.groupby('Region').agg({'Total Sales': 'sum', 'Operating Profit': 'sum'}).reset_index()
# Sort the data by Total Sales and Operating Profit in descending order
revenue_profit_by_region = revenue_profit_by_region.sort_values(by=['Total Sales', 'Operating Profit'], ascending=False)
# Create bar plots using Seaborn
fig, axes = plt.subplots(2, 1, figsize=(10, 12))
# Total Sales by Region
sns.barplot(data=revenue_profit_by_region, x='Total Sales', y='Region', color='blue', ax=axes[0])
axes[0].set_xlabel('Total Sales')
axes[0].set_ylabel('Region')
axes[0].set_title('Total Sales by Region')
# Operating Profit by Region
sns.barplot(data=revenue_profit_by_region, x='Operating Profit', y='Region', color='green', ax=axes[1])
```

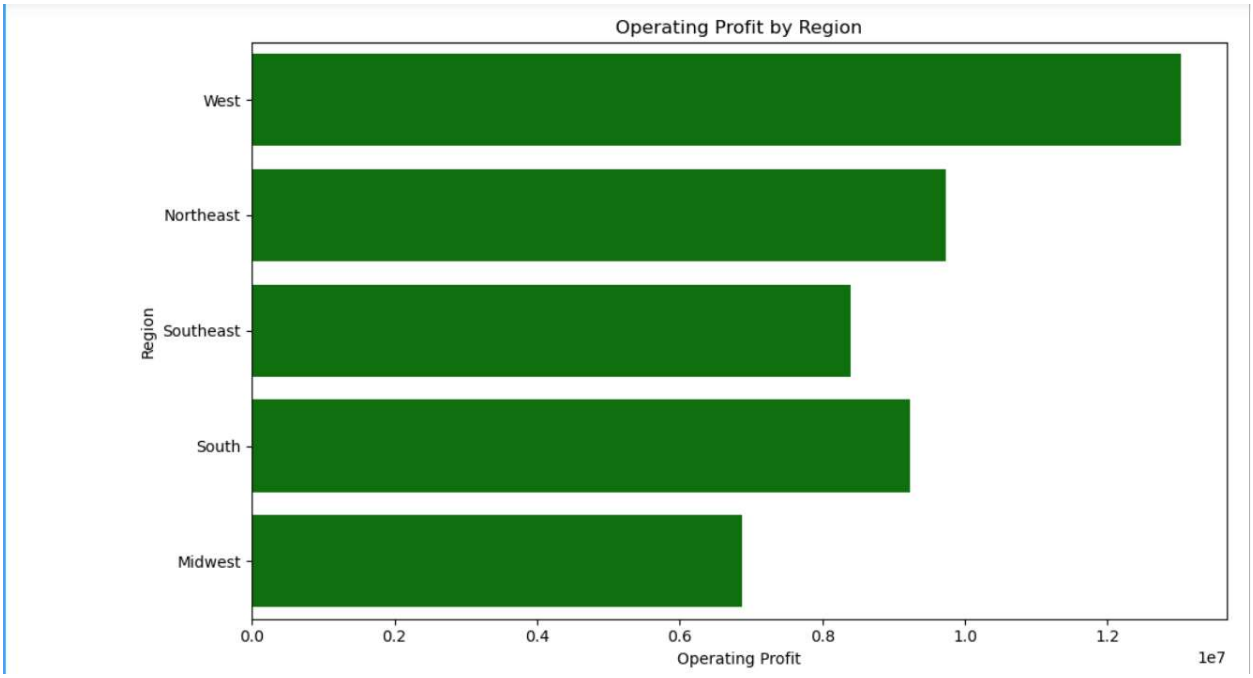
```
axes[1].set_xlabel('Operating Profit')
axes[1].set_ylabel('Region')
axes[1].set_title('Operating Profit by Region')
# Region with highest revenue and operating profit
highest_revenue_region = revenue_profit_by_region.iloc[0]['Region']
highest_profit_region = revenue_profit_by_region.iloc[0]['Region']
# Region with lowest revenue and operating profit
lowest_revenue_region = revenue_profit_by_region.iloc[-1]['Region']
lowest_profit_region = revenue_profit_by_region.iloc[-1]['Region']
plt.tight_layout()
plt.show()
print('Region with highest revenue:', highest_revenue_region)
print('Region with highest operating profit:', highest_profit_region)
print('Region with lowest revenue:', lowest_revenue_region)
print('Region with lowest operating profit:', lowest_profit_region)
```

In [18]:

```
#Which region has the highest revenue and operating profit, and which region has the lowest?
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

# Assuming you have a DataFrame named df with the necessary columns

# Filter the data to exclude any missing values in Total Sales and Operating Profit columns
filtered_df = df.dropna(subset=['Total Sales', 'Operating Profit'])

# Calculate the sum of Total Sales and Operating Profit by Region
revenue_profit_by_region = filtered_df.groupby('Region').agg({'Total Sales': 'sum', 'Operating Profit': 'sum'}).reset_index()

# Sort the data by Total Sales and Operating Profit in descending order
revenue_profit_by_region = revenue_profit_by_region.sort_values(by=['Total Sales', 'Operating Profit'], ascending=False)

# Create bar plots using Seaborn
fig, axes = plt.subplots(2, 1, figsize=(10, 12))

# Total Sales by Region
sns.barplot(data=revenue_profit_by_region, x='Total Sales', y='Region', color='blue', ax=axes[0])
axes[0].set_xlabel('Total Sales')
axes[0].set_ylabel('Region')
axes[0].set_title('Total Sales by Region')

# Operating Profit by Region
sns.barplot(data=revenue_profit_by_region, x='Operating Profit', y='Region', color='green', ax=axes[1])
axes[1].set_xlabel('Operating Profit')
axes[1].set_ylabel('Region')
axes[1].set_title('Operating Profit by Region')
```

```
# Region with highest revenue and operating profit
highest_revenue_region = revenue_profit_by_region.iloc[0]['Region']
highest_profit_region = revenue_profit_by_region.iloc[0]['Region']

# Region with lowest revenue and operating profit
lowest_revenue_region = revenue_profit_by_region.iloc[-1]['Region']
lowest_profit_region = revenue_profit_by_region.iloc[-1]['Region']

plt.tight_layout()
plt.show()

print('Region with highest revenue:', highest_revenue_region)
print('Region with highest operating profit:', highest_profit_region)
print('Region with lowest revenue:', lowest_revenue_region)
print('Region with lowest operating profit:', lowest_profit_region)
```




```
Region with highest revenue: West
Region with highest operating profit: West
Region with lowest revenue: Midwest
Region with lowest operating profit: Midwest
```

**2. How does the number of units sold vary across various products?**

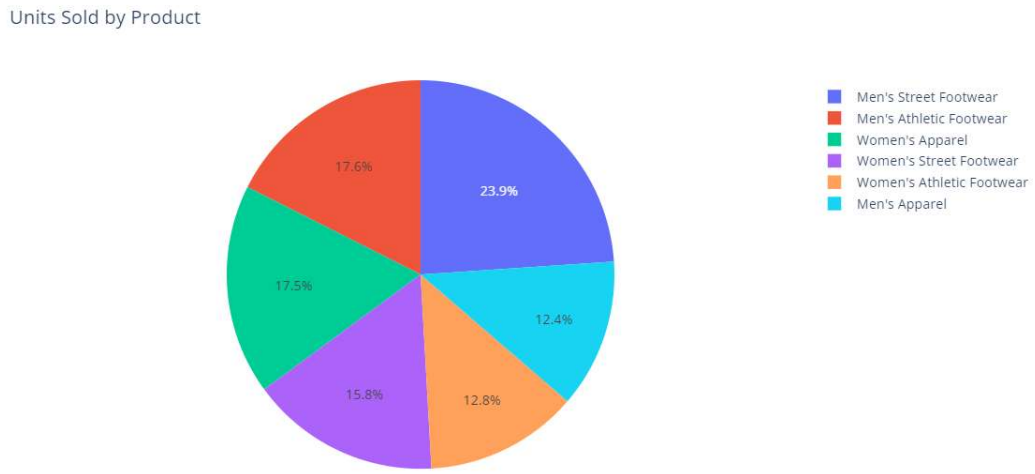
- In this, we used the Plotly Express library to create a pie chart visualizing the variation in the number of units sold across various products.
- Each product is represented by a slice of the pie, and the size of each slice corresponds to the proportion of units sold for that product.
- By examining the pie chart, we can gain insights into the distribution of unit sales among the various products.
- Products with larger slices indicate higher sales volumes, while smaller slices represent lower sales volumes.
- This visualization allows us to easily compare the sales performance of various products and identify which products contribute the most or least to the overall sales.
- It provides a visual summary of the relative popularity or demand for each product based on the number of units sold.

***Conclusion:***

- In the men's product category, the highest selling item is Street/Casual Footwear, accounting for 23.9% of sales, while the least sold items are Apparel/Clothing, comprising only 12.4% of sales in the USA.
- On the other hand, in the women's product category, the most popular item is Women's Apparel, representing 17.5% of sales, followed by athletic footwear, which is the least sold item with 12.8% of sales.

```
import pandas as pd
import plotly.express as px
# Assuming you have a DataFrame named df with the necessary columns
fig = px.pie(df, values='Units Sold', names='Product', title='Units Sold by Product')
fig.show()
```

In [19]:
```
#How does the number of units sold vary across different products?

import pandas as pd
import plotly.express as px

# Assuming you have a DataFrame named df with the necessary columns
fig = px.pie(df, values='Units Sold', names='Product', title='Units Sold by Product')
fig.show()
```

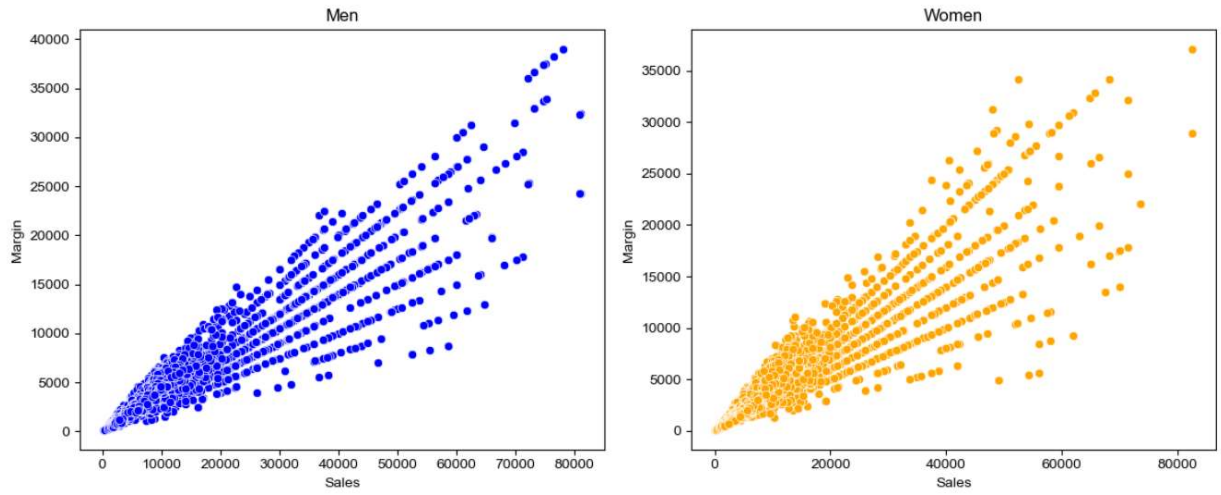


**3. How does the relationship between total sales and operating profit differ between men and women?**

- The figure displays two subplots comparing the relationship between total sales and operating profit for men and women. The scatterplots visualize the data points for each gender, with the x-axis representing total sales and the y-axis representing operating profit. The left subplot focuses on men, while the right subplot focuses on women.
- The following visualization shows the ***direct proportion*** with Total Sales compared to Operating Margin for both the subplots.
- The two figures are subplots created using the plt.subplots() function.
- The DataFrame filters creates a separate df for men and women, named df_men and df_women, respectively.
- The style of the plots is set using sns.set() with the "darkgrid" style and these subplots are a scatter plot that is created using sns.scatterplot().
- For the left subplot (representing men), it uses the data from the df_men DataFrame, with the 'Total Sales' column as the x-axis and the 'Operating Profit' column as the y-axis. The scatter plot is colored in blue.
- For the right subplot (representing women), it uses the df_women DataFrame. The scatter plot is colored in orange. The subplot is given a title ('Women'), x-axis label ('Sales'), and y-axis label ('Margin').
- The spacing between the subplots is adjusted using plt.tight_layout() to avoid overlap.
- Finally, the plot is displayed using plt.show().

```
# Create a figure with two subplots
fig, axes = plt.subplots(1, 2, figsize=(12, 5))
# Filtered DataFrame for men and women
df_men = df[df['Gender'] == 'Men']
df_women = df[df['Gender'] == 'Women']
sns.set(style="darkgrid")  # Set the style of the plots
# Plot for men (left subplot)
sns.scatterplot(data=df_men, x='Total Sales', y='Operating Profit', ax=axes[0], color='blue')
axes[0].set_title('Men')
axes[0].set_xlabel('Sales')
axes[0].set_ylabel('Margin')
# Plot for women (right subplot)
sns.scatterplot(data=df_women, x='Total Sales', y='Operating Profit', ax=axes[1], color='orange')
axes[1].set_title('Women')
axes[1].set_xlabel('Sales')
axes[1].set_ylabel('Margin')
plt.tight_layout() # Adjust the spacing between subplots
plt.show() # Display the plot
```

In [17]:
```
# How does the relationship between total sales and operating profit differ between men and women?
# Create a figure with two subplots
fig, axes = plt.subplots(1, 2, figsize=(12, 5))

# Filtered DataFrame for men and women
df_men = df[df['Gender'] == 'Men']
df_women = df[df['Gender'] == 'Women']

# Set the style of the plots
sns.set(style="darkgrid")

# Plot for men (left subplot)
sns.scatterplot(data=df_men, x='Total Sales', y='Operating Profit', ax=axes[0], color='blue')
axes[0].set_title('Men')
axes[0].set_xlabel('Sales')
axes[0].set_ylabel('Margin')

# Plot for women (right subplot)
sns.scatterplot(data=df_women, x='Total Sales', y='Operating Profit', ax=axes[1], color='orange')
axes[1].set_title('Women')
axes[1].set_xlabel('Sales')
axes[1].set_ylabel('Margin')

# Adjust the spacing between subplots
plt.tight_layout()

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



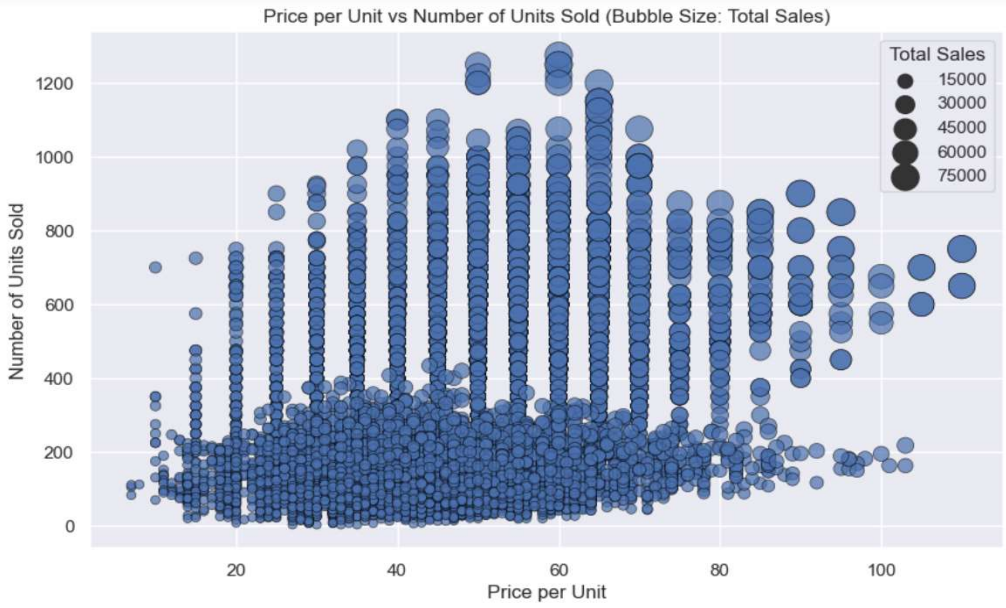
**4. How does the price per unit affect the number of units sold and total sales?**

- The below visualization shows the relationship between the price per unit and the number of units sold, where the size of the bubbles represents the total sales. It allows for an overview of the distribution of sales based on price and quantity, with larger bubbles indicating higher total sales.
- The provided code below creates a bubble chart using the Seaborn library. The style of the plot is set to "darkgrid" using sns.set().
- The bubble chart is created using sns.scatterplot(). The 'Price per Unit' column is used for the x-axis, the 'Units Sold' column for the y-axis, and the 'Total Sales' column is mapped to the size of the bubbles. The sizes parameter is set to (30, 300), specifying the range of sizes for the bubbles.
- The bubbles are slightly transparent (alpha=0.7) and have black edges (edgecolor='black').
- The plot is given a title ('Price per Unit vs Number of Units Sold (Bubble Size: Total Sales)'), and the x-axis and y-axis are labeled accordingly ('Price per Unit' and 'Number of Units Sold').
- Finally, the plot is displayed using plt.show().

```
# Set the style of the plot
sns.set(style="darkgrid")
# Create the bubble chart
plt.figure(figsize=(10, 6))
```

| |
|---|
| sns.scatterplot(data=df, x='Price per Unit', y='Units Sold', size='Total Sales', sizes=(30, 300), alpha=0.7, edgecolor='black') |
| # Set the title and labels for the plot |
| plt.title('Price per Unit vs Number of Units Sold (Bubble Size: Total Sales)') |
| plt.xlabel('Price per Unit') |
| plt.ylabel('Number of Units Sold') |
| # Display the plot |
| plt.show() |

In [18]:
```
#How does the price per unit affect the number of units sold and total sales?
# Set the style of the plot
sns.set(style="darkgrid")

# Create the bubble chart
plt.figure(figsize=(10, 6))
sns.scatterplot(data=df, x='Price per Unit', y='Units Sold', size='Total Sales', sizes=(30, 300),
                alpha=0.7, edgecolor='black')

# Set the title and labels for the plot
plt.title('Price per Unit vs Number of Units Sold (Bubble Size: Total Sales)')
plt.xlabel('Price per Unit')
plt.ylabel('Number of Units Sold')

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



**5. What is the total revenue generated by the retailer during a specific period?**

- The following provided code filters df for the retailer 'Amazon' and the year '2021', creating a new dataframe called filtered_df.
- From the filtered data, a subset of columns ('Retailer', 'Year', 'Region', 'Product', 'Total Sales') is selected into a new DataFrame called df1.
- The code then groups the data in df1 by 'Region' and 'Product' and calculates the average total sales for each group. The result is stored in an average_sales.
- A pivot table is created from average_sales, with 'Region' as the index, 'Product' as the columns, and 'Total Sales' as the values.
- Using Seaborn, a heatmap is created with the pivot table data. The heatmap visualizes the average total sales for each combination of product and region. The values in the heatmap cells are annotated, the color map used is 'YlGnBu', and the numbers are formatted with two decimal places. The plot is given a title ('Average Total Sales by Product and Region'), and the x-axis and y-axis are labeled accordingly ('Product' and 'Region').
- Finally, the plot is displayed using plt.show().
- In summary, the visualization shows the average total sales for various products and regions, using a heatmap to represent the values. The darker shades indicate higher average sales, allowing for a quick comparison of sales performance across products and regions.

```
# Filter the DataFrame for retailer='Amazon' and year='2021'
filtered_df = df[(df['Retailer'] == 'Amazon') & (df['Year'] == 2021)]
df1 = filtered_df[['Retailer', 'Year', 'Region', 'Product', 'Total Sales']]
# Group the filtered data by product and region and calculate the average total sales
average_sales = df1.groupby(['Region', 'Product'])['Total Sales'].mean().reset_index()
# Create the pivot table
pivot_table = average_sales.pivot(index='Region', columns='Product', values='Total Sales')
# Create the heatmap using Seaborn
plt.figure(figsize=(10, 6))
sns.heatmap(data=pivot_table, annot=True, cmap='YlGnBu', fmt='.2f', linewidths=0.5)
plt.title('Average Total Sales by Product and Region')
plt.xlabel('Product')
plt.ylabel('Region')
plt.show()
```

In [19]:
```
#What is the total revenue generated by the retailer during a specific period?
# Filter the DataFrame for retailer='Amazon' and year='2021'
filtered_df = df[(df['Retailer'] == 'Amazon') & (df['Year'] == 2021)]

df1 = filtered_df[['Retailer', 'Year', 'Region', 'Product', 'Total Sales']]

# Group the filtered data by product and region and calculate the average total sales
average_sales = df1.groupby(['Region', 'Product'])['Total Sales'].mean().reset_index()

# Create the pivot table
pivot_table = average_sales.pivot(index='Region', columns='Product', values='Total Sales')

# Create the heatmap using Seaborn
plt.figure(figsize=(10, 6))
sns.heatmap(data=pivot_table, annot=True, cmap='YlGnBu', fmt='.2f', linewidths=0.5)
plt.title('Average Total Sales by Product and Region')
plt.xlabel('Product')
plt.ylabel('Region')
plt.show()
```