**Project Title: Retail Data Analysis**

**Background**

The retail industry generates vast amounts of data related to product sales, inventory, store performance, and customer behaviour. Leveraging this data for analysis provides valuable insights that can help businesses make informed decisions regarding product stocking, marketing strategies, and operational efficiency. For this project, several datasets were made available, including details on stores, inventory, product information, and sales records, along with a calendar table for time-based analysis.

The project is centered around utilizing this raw retail data to generate insights related to sales performance, inventory management, and product profitability. By transforming this data into dimension tables, the project aims to structure the data into a more analytical and efficient format that can be further utilized for reporting and visualization.

**Objectives**

The primary objectives of this project are:

1. **Data Integration and Cleaning**: To load, clean, and preprocess data from multiple CSV files (inventory, products, sales, store, and calendar data).
2. **Sales and Inventory Analysis**: To analyse sales trends, identify top and bottom-performing products and stores, and explore inventory levels across different stores.
3. **Profitability Assessment**: To evaluate the profitability of different products by calculating profit margins and identifying key drivers of revenue.
4. **Creation of Dimension Tables**: To transform raw data into well-structured dimension tables for easier querying and reporting, which can then be used in data warehousing or business intelligence tools.
5. **Time Series Analysis**: To explore time-based sales trends, including seasonal patterns and monthly performance, using the calendar dimension table.

**End Goals**

1. **Informed Decision-Making**: Provide a framework for store managers and retail analysts to make data-driven decisions regarding inventory restocking, product placement, and sales strategies.
2. **Optimized Inventory Management**: Use insights from inventory and sales data to reduce stockouts and overstock situations by better understanding store-specific demands.
3. **Profit Maximization**: Identify which products generate the most profit and which are underperforming to optimize marketing strategies.
4. **Data-Driven Reporting**: Prepare clean, structured dimension tables that can be utilized for reporting purposes in tools like Power BI for continuous and scalable analysis.

**Final Deliverables**

1. **Cleaned and Pre-processed Datasets**: The provided raw datasets will be transformed into clean and structured formats, ready for analysis.
2. **Insights and Visualizations**:
   * Store-wise performance: the top-performing and low-performing stores based on sales.
   * Sales trends over time: Analyze sales across months or seasons (using the calendar table) to spot patterns.
   * Inventory vs Sales: Identify stores with overstock or stockouts and correlate that with their sales performance.
   * Category-based analysis: Explore how different product categories contribute to sales.
   * Top Performing Products: unit wise and profit wise
   * Worst performing Products: units and Profit wise.
3. **Dimension Tables**: Structured dimension tables for stores, products,category and the calendar, which can be used for further analysis or integration into a data warehouse.
4. **Documentation**: A detailed report covering the methodology, key findings, insights generated, and recommendations based on the analysis.
5. **Python Scripts and Notebooks**: Well-commented Python scripts used for data processing, analysis, and visualization.

**Data Collection:**

www.mavenanalytic.io

**Data specification:**

We have five datasets:

1. **Inventory Data**:
   * Store\_ID: The store identifier.
   * Product\_ID: The product identifier.
   * Stock\_On\_Hand: The current stock levels.
2. **Products Data**:
   * Product\_ID: The product identifier.
   * Product\_Name: Name of the product.
   * Product\_Category: Category of the product.
   * Product\_Cost: Cost of the product to the store.
   * Product\_Price: Selling price of the product.
3. **Sales Data**:
   * Sale\_ID: The sales transaction identifier.
   * Date: Date of sale.
   * Store\_ID: The store where the sale happened.
   * Product\_ID: The product sold.
   * Units: Number of units sold.
4. **Stores:**
   * Store\_ID,
   * Store\_Name
   * Store\_City
   * Store\_Location,
   * Store\_Open\_Date
5. **Calender:**
   * Date

**Tools: Colab**

**Codes:**

**Part 1**

**Importing Libraries**

**import pandas as pd**

**import numpy as np**

**import matplotlib.pyplot as plt**

**import seaborn as sns**

**#sns.set(style='whitegrid')**

**Setting Background**

# Set the background color to black

plt.style.use('dark\_background')

# Define colors

primary\_colors = ['#FDE64B', '#00B4D8','#008ECC',

#0E4D92','#1E90FF','#FA8072', '#C21807']

grid\_color = '#D3D3D3'

text\_color = '#FFFFFF'

subtext\_color = '#F0E68C'

**Load Datasets:**

stores = pd.read\_csv('drive/MyDrive/Maven Toys Data/stores.csv')

sales = pd.read\_csv("drive/MyDrive/Maven Toys Data/sales.csv")

products = pd.read\_csv("drive/MyDrive/Maven Toys Data/products.csv")

inventory = pd.read\_csv("drive/MyDrive/Maven Toys

Data/inventory.csv")

calender = pd.read\_csv("drive/MyDrive/Maven Toys Data/calendar.csv")

**Part 2**

**Data Cleaning**

1. **Basic Data Info**

print(stores.head())

print(sales.head())

print(products.head())

print(inventory.head())

print(calender.head())

print(stores.info())

print(sales.info())

print(products.info())

print(inventory.info())

print(calender.info())

**2. Check for missing value**

print(stores.isnull().sum())

print(sales.isnull().sum())

print(products.isnull().sum())

print(inventory.isnull().sum())

print(calender.isnull().sum())

**3. Check for Duplicates**

print(stores.duplicated().sum())

print(sales.duplicated().sum())

print(products.duplicated().sum())

print(inventory.duplicated().sum())

print(calender.duplicated().sum())

1. **Check for outliers**

print(stores.describe())

print(sales.describe())

print(products.describe())

print(inventory.describe())

print(calender.describe())

**5. Validate Data Range in Sales table**

print(sales['Date'].min())

print(sales['Date'].max())

print(calender.duplicated().sum())

**Data Transformation**

**1.Extracting Date Attributes**

calender['Date'] = pd.to\_datetime(calender['Date'])

calender['Year'] = calender['Date'].dt.year

calender['Quarter'] = calender['Date'].dt.quarter

calender['Month'] = calender['Date'].dt.month

calender['Day'] = calender['Date'].dt.day

# Calculate the Week of the Month (start from 1 for each month)

calender['Week\_of\_Month'] = calender['Date'].apply(lambda d: (d.day - 1) // 7 + 1)

1. **Creating Dimension Tables**

# Store Dimension

store\_dim = stores[['Store\_ID', 'Store\_Name', 'Store\_City', 'Store\_Location', 'Store\_Open\_Date']].drop\_duplicates().reset\_index(drop=True)

# Calendar Dimension Table

date\_dim = calender[['Date', 'Year', 'Month', 'Day', 'Week\_of\_Month', 'Quarter']].drop\_duplicates().reset\_index(drop=True)

**# Category Dimension**

category\_dim = products[['Product\_Category']].drop\_duplicates().reset\_index(drop=True)

# Add unique Category\_ID

category\_dim['Category\_ID'] = category\_dim.index + 1

**# Product Dimension Table**

**# Merge Category Dimension with Product Dimension**

product\_dim = pd.merge(products[['Product\_ID', 'Product\_Name', 'Product\_Category', 'Product\_Cost', 'Product\_Price']],

category\_dim, on='Product\_Category', how='left')

# Drop the 'Product\_Category' column and retain 'Category\_ID'

product\_dim.drop('Product\_Category', axis=1, inplace=True)

# Inventory Dimension Table

#Merge Category\_ID into the Inventory Table

inventory\_dim = pd.merge(inventory, product\_dim[['Product\_ID', 'Category\_ID']], on='Product\_ID', how='left'

**Saving Dimension Tables**

# Saving Dimension Tables as csv

store\_dim.to\_csv('store\_dim.csv', index=False)

inventory\_dim.to\_csv('inventory\_dim.csv', index=False)

product\_dim.to\_csv('product\_dim.csv', index=False)

date\_dim.to\_csv('date\_dim.csv', index=False)

category\_dim.to\_csv('category\_dim.csv', index=False)

**Merging tables and more transformation for Analysis**

# Convert 'Date' in sales to datetime format

sales['Date'] = pd.to\_datetime(sales['Date'])

# Merge sales with product and date dimensions to get product names and year information

sales\_with\_product = pd.merge(sales, product\_dim, on='Product\_ID', how='left')

sales\_with\_date = pd.merge(sales\_with\_product, date\_dim, on='Date', how='left')

# Removing $ sign from Product\_Cost and Product\_Price columns

sales\_with\_date['Product\_Price'] = sales\_with\_date['Product\_Price'].replace({'\$': ''}, regex=True)

sales\_with\_date['Product\_Cost'] = sales\_with\_date['Product\_Cost'].replace({'\$': ''}, regex=True)

sales\_with\_date['Units'] = pd.to\_numeric(sales\_with\_date['Units'], errors='coerce')

sales\_with\_date['Product\_Price'] = pd.to\_numeric(sales\_with\_date['Product\_Price'], errors='coerce')

# Merge inventory with product and category tables to get category and product info

inventory\_with\_product = pd.merge(inventory\_dim, product\_dim[['Product\_ID', 'Product\_Name']], on='Product\_ID', how='left')

# Merge inventory with category dimension to get category information

inventory\_with\_category = pd.merge(inventory\_with\_product, category\_dim, on='Category\_ID', how='left')

# Merge with store\_dim to get store information

inventory\_with\_store = pd.merge(inventory\_with\_category, store\_dim, on='Store\_ID', how='left')

inventory\_with\_store['Stock\_On\_Hand'] = pd.to\_numeric(inventory\_with\_store['Stock\_On\_Hand'], errors='coerce')

**Visualization**

**Sala Analysis**

**Sale Trend**

# Aggregate sales by Year

yearly\_sales = sales\_with\_price.groupby('Year').agg({'Total\_Sales': 'sum'}).reset\_index()

# Plot Yearly Sales

plt.figure(figsize=(10, 6))

plt.plot(yearly\_sales['Year'], yearly\_sales['Total\_Sales'], marker='o', color=primary\_colors[0], label='Yearly Sales')

plt.title('Yearly Sales Trend', color= subtext\_color)

plt.xlabel('Year', color=text\_color)

plt.ylabel('Total Sales', color=text\_color)

plt.xticks(yearly\_sales['Year'], rotation=45)

plt.grid(False)

plt.legend(loc='upper right')

plt.tight\_layout()

plt.show()

A screen shot of a graph

Description automatically generated

Fig: Sale trend for year 2022 and 2023

# Aggregate sales by Year and Quarter for Quarterly Sales

quarterly\_sales = sales\_with\_price.groupby(['Year', 'Quarter']).agg({'Total\_Sales': 'sum'}).reset\_index()

quarterly\_sales['Year\_Quarter'] = quarterly\_sales['Year'].astype(str) + ' Q' + quarterly\_sales['Quarter'].astype(str)

# Plot Quarterly Sales

plt.figure(figsize=(10, 6))

plt.plot(quarterly\_sales['Year\_Quarter'], quarterly\_sales['Total\_Sales'], marker='o', color=primary\_colors[0])

plt.title('Quarterly Total Sales', color= subtext\_color)

plt.xlabel('Year Quarter', color = text\_color)

plt.ylabel('Total Sales', color = text\_color)

plt.xticks(rotation=45)

plt.grid(False)

plt.tight\_layout()

plt.show()

A graph with a line and a dotted line

Description automatically generated

Fig: Quarterly Sale trend for year 2022 and 2023

year = sales\_with\_price['Year']

month = sales\_with\_price['Month']

week\_of\_month = sales\_with\_price['Week\_of\_Month']

 #Aggregate sales by Year, Month, and Week\_of\_Month for accurate weekly sales

weekly\_sales = sales\_with\_price.groupby(['Year', 'Month', 'Week\_of\_Month']).agg({'Total\_Sales': 'sum'}).reset\_index()

# Create a 'Year\_Week' column to use as the x-axis label in the format 'Jan-22-W1', 'Feb-23-W2', etc.

weekly\_sales['Year\_Week'] = weekly\_sales['Year'].astype(str) + '-' + weekly\_sales['Month'].astype(str) + '-W' + weekly\_sales['Week\_of\_Month'].astype(str)

# Plot Weekly Sales

plt.figure(figsize=(14, 6))

plt.plot(weekly\_sales['Year\_Week'], weekly\_sales['Total\_Sales'], color=primary\_colors[0])

plt.title('Weekly Total Sales')

plt.xlabel('Year-Month-Week')

plt.ylabel('Total Sales')

# Rotate the labels for better readability

plt.xticks(rotation=60, ha='right', fontsize=8)  # Rotate labels and adjust alignment and font size

# Show only every nth label (e.g., every 4th label)

plt.gca().set\_xticks(plt.gca().get\_xticks()[::4])  # Show every 4th label

plt.xticks(rotation=45)

plt.grid(False)

plt.tight\_layout()

plt.show()

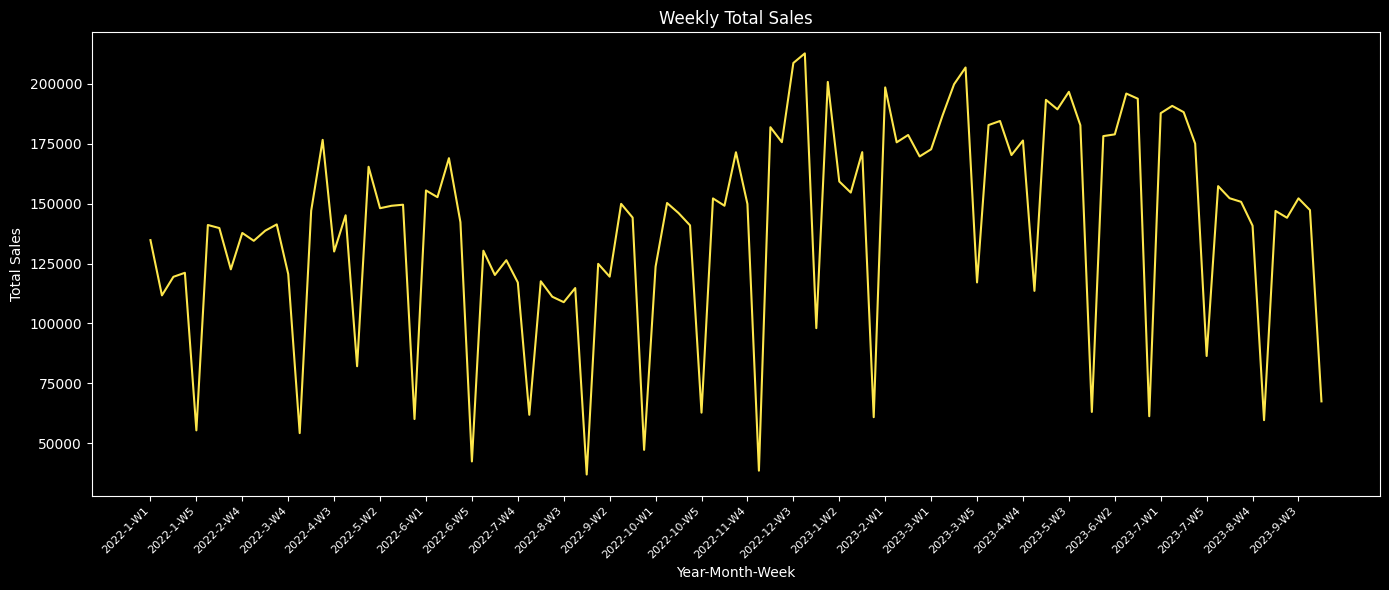


Fig: Weekly Sale Trend for year 2022 and 2023

# Aggregate sales by Date for Daily Sales

daily\_sales = sales\_with\_price.groupby('Date').agg({'Total\_Sales': 'sum'}).reset\_index()

# Plot Daily Sales

plt.figure(figsize=(10, 6))

plt.plot(daily\_sales['Date'], daily\_sales['Total\_Sales'], color=primary\_colors[0])

plt.title('Daily Total Sales')

plt.xlabel('Date')

plt.ylabel('Total Sales')

plt.xticks(rotation=45)

plt.grid(False)

plt.tight\_layout()

plt.show()

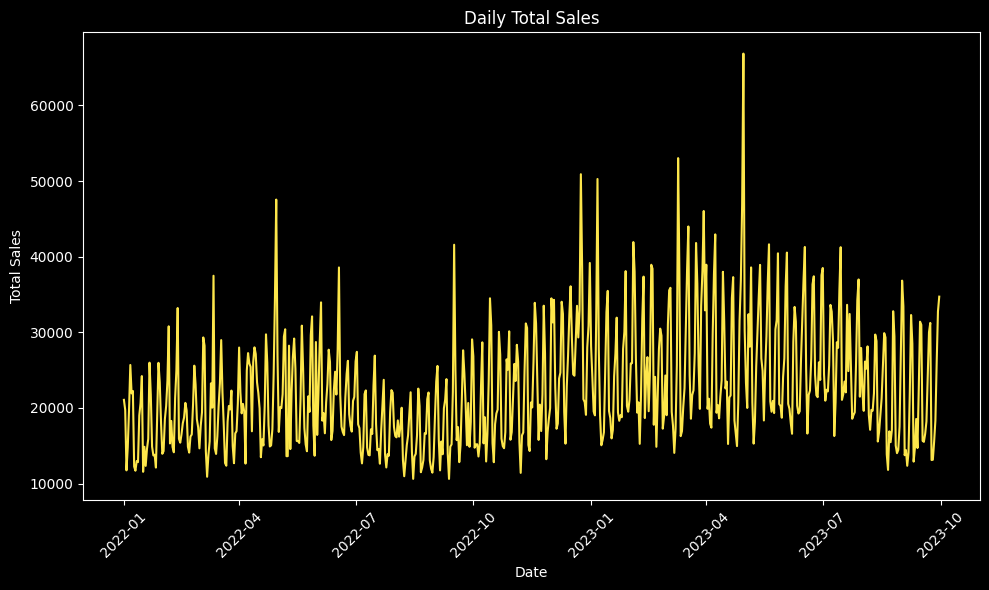


Fig: Daily Sale Trend for year 2022-2023

# Convert the Date column to datetime if not already

sales\_with\_price['Date'] = pd.to\_datetime(sales\_with\_price['Date'])

# Aggregate sales by Year and Month

monthly\_sales = sales\_with\_price.groupby(['Year', 'Month']).agg({'Total\_Sales': 'sum'}).reset\_index()

monthly\_sales['Date'] = pd.to\_datetime(monthly\_sales['Year'].astype(str) + '-' + monthly\_sales['Month'].astype(str) + '-01')

# Dictionary of holidays with corresponding months to highlight

holiday\_months = {

    "New Year": ["2022-01", "2023-01"],

    "Valentine": ["2022-02", "2023-02"],

    "Easter": ["2022-04", "2023-04"],

    "Back to School": ["2022-08", "2023-08"],

    "Thanksgiving": ["2022-11"],

    " Christmas": ["2022-12"]

}

# Plot Monthly Sales

plt.figure(figsize=(10, 6))

plt.plot(monthly\_sales['Date'], monthly\_sales['Total\_Sales'], color=primary\_colors[0], label='Monthly Sales')

plt.title('Monthly Total Sales with Holiday Months Highlighted', color= subtext\_color)

plt.xlabel('Year-Month', color= text\_color)

plt.ylabel('Total Sales', color= text\_color)

plt.xticks(rotation=45)

# Highlight holiday months and add annotations

for holiday, date\_str\_list in holiday\_months.items():

    for date\_str in date\_str\_list:

# Extract start and end of the month

 start\_date = pd.to\_datetime(date\_str + "-01")

 end\_date = start\_date + pd.offsets.MonthEnd(0)

 plt.axvspan(start\_date, end\_date, color='grey', alpha=0.3)

 # Annotate the holiday label at the midpoint of the month

 mid\_date = start\_date + (end\_date - start\_date) / 2

 plt.annotate(holiday, (mid\_date, monthly\_sales['Total\_Sales'].max() \* 0.95),

textcoords="offset points", xytext=(0, 9), ha='center',

fontsize=6, color=subtext\_color)

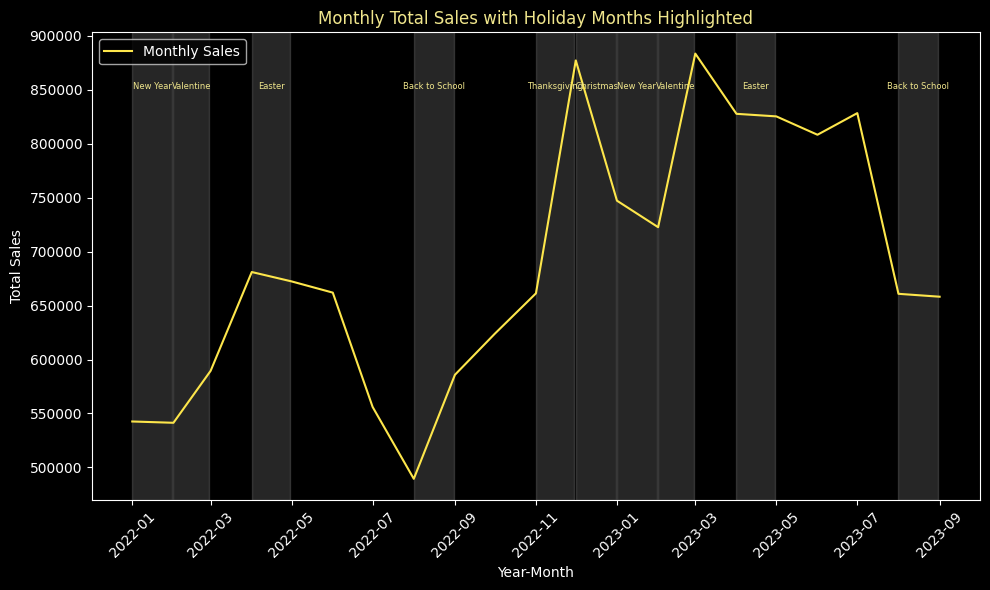
# Configure the legend

plt.legend(loc='upper left')

plt.grid(False)

plt.tight\_layout()

plt.show()



**Sales forecasting**

from statsmodels.tsa.statespace.sarimax import SARIMAX

import matplotlib.dates as mdates

# Assuming 'sales\_with\_price' DataFrame has a 'Date' and 'Total\_Sales' column

# Aggregate sales monthly

sales\_with\_price['Date'] = pd.to\_datetime(sales\_with\_price['Date'])

monthly\_sales = sales\_with\_price.groupby(pd.Grouper(key='Date', freq='M'))['Total\_Sales'].sum().reset\_index()

# Fit the SARIMA model (Seasonal ARIMA)

# Here, the order parameters (p, d, q) and seasonal order (P, D, Q, m) are chosen for simplicity

# You may need to tune these for better accuracy

sarima\_model = SARIMAX(monthly\_sales['Total\_Sales'], order=(1, 1, 1), seasonal\_order=(1, 1, 1, 12))

sarima\_results = sarima\_model.fit()

# Forecasting the last quarter of 2023 (next 3 months)

forecast\_steps = 3

forecasted\_values = sarima\_results.get\_forecast(steps=forecast\_steps).predicted\_mean

forecasted\_dates = pd.date\_range(start=monthly\_sales['Date'].iloc[-1] + pd.DateOffset(months=1), periods=forecast\_steps, freq='M')

# Create a DataFrame for forecasted sales

forecast\_df = pd.DataFrame({'Date': forecasted\_dates, 'Total\_Sales': forecasted\_values})

# Combine the actual and forecasted sales

combined\_sales = pd.concat([monthly\_sales, forecast\_df], ignore\_index=True)

# Plot monthly sales and forecast

plt.figure(figsize=(14, 8))

plt.plot(combined\_sales['Date'], combined\_sales['Total\_Sales'], marker='o', color=primary\_colors[0], label='Monthly Sales')

plt.axvline(x=monthly\_sales['Date'].iloc[-1], color=primary\_colors[5], linestyle='--', label='Forecast Start')

plt.title('Monthly Total Sales with Forecast (2022 & 2023)')

plt.xlabel('Date')

plt.ylabel('Total Sales')

plt.xticks(rotation=45)

plt.legend()

plt.grid(False)

plt.tight\_layout()

plt.show()

# Quarterly and yearly aggregation

quarterly\_sales = combined\_sales.groupby(combined\_sales['Date'].dt.to\_period('Q'))['Total\_Sales'].sum().reset\_index()

yearly\_sales = combined\_sales.groupby(combined\_sales['Date'].dt.year)['Total\_Sales'].sum().reset\_index()

# Plot quarterly sales

plt.figure(figsize=(10, 6))

plt.plot(quarterly\_sales['Date'].astype(str), quarterly\_sales['Total\_Sales'], color=primary\_colors[0], marker='o', label='Quarterly Sales')

plt.axvline(x=str(monthly\_sales['Date'].iloc[-1].to\_period('Q')), color=primary\_colors[5], linestyle='--', label='Forecast Start')

plt.title('Quarterly Total Sales with Forecast')

plt.xlabel('Quarter')

plt.ylabel('Total Sales')

plt.xticks(rotation=45)

plt.legend()

plt.grid(False)

plt.tight\_layout()

plt.show()

# Plot yearly sales

plt.figure(figsize=(10, 6))

plt.plot(yearly\_sales['Date'].astype(str), yearly\_sales['Total\_Sales'], color=primary\_colors[0], marker='o', label='Yearly Sales')

plt.title('Yearly Total Sales')

plt.xlabel('Year')

plt.ylabel('Total Sales')

plt.legend()

plt.grid(False)

plt.tight\_layout()

plt.show()

A graph with yellow dots

Description automatically generated

A graph with yellow lines

Description automatically generated

A screen shot of a graph

Description automatically generated

**Category Contribution**

# Group by Category and Year to calculate total sales (Units) per category

category\_unit\_sales\_year = sales\_with\_category.groupby(['Year', 'Product\_Category'])['Units'].sum().reset\_index()

# Get sales data by category for 2022 and 2023

category\_unit\_sales\_2022 = category\_unit\_sales\_year[category\_unit\_sales\_year['Year'] == 2022]

category\_unit\_sales\_2023 = category\_unit\_sales\_year[category\_unit\_sales\_year['Year'] == 2023]

# Plot pie charts for 2022 and 2023

fig, ax = plt.subplots(1, 2, figsize=(14, 7))

# Pie chart for 2022

ax[0].pie(category\_unit\_sales\_2022['Units'],explode= (0,0.1,0,0.1,0), labels=category\_unit\_sales\_2022['Product\_Category'], autopct='%1.1f%%', startangle=90,colors=['#0e98ba', '#00B4D8','#008ECC', '#0E4D92','#1E90FF','#00CED1'],wedgeprops={'edgecolor':'k'})

ax[0].set\_title('Unit Sold by Category in 2022', color = subtext\_color)

# Pie chart for 2023

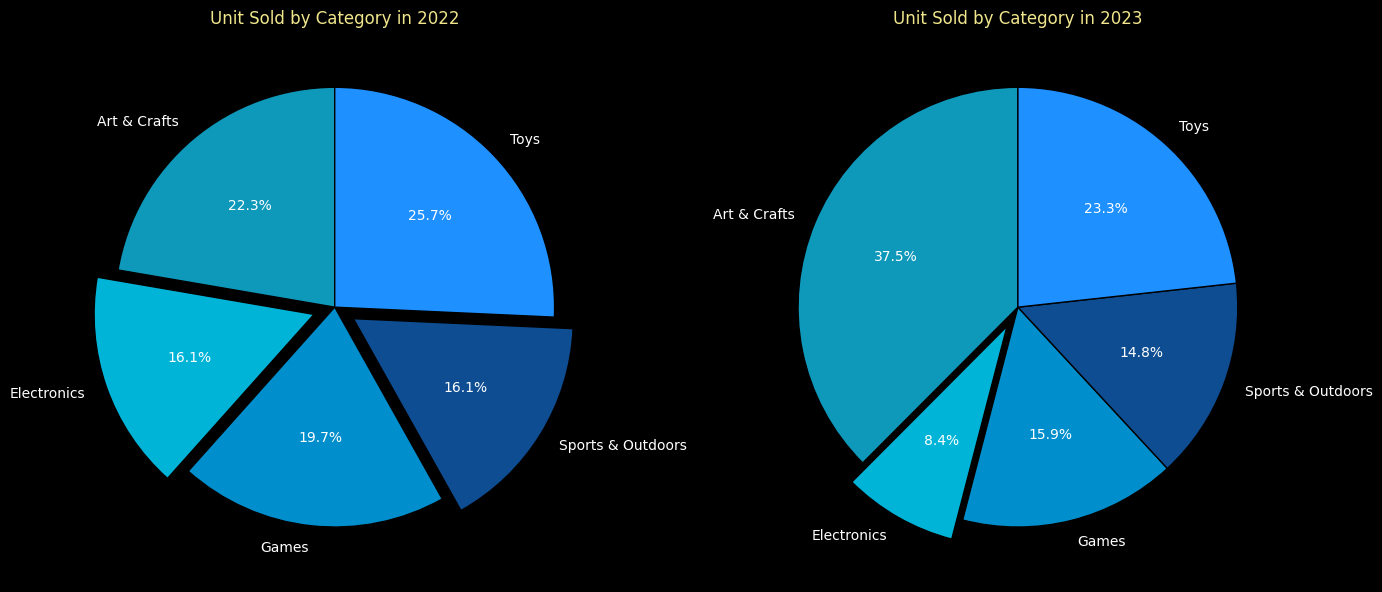
ax[1].pie(category\_unit\_sales\_2023['Units'],explode= (0,0.1,0,0,0), labels=category\_unit\_sales\_2023['Product\_Category'], autopct='%1.1f%%', startangle=90,colors=['#0e98ba', '#00B4D8','#008ECC', '#0E4D92','#1E90FF','#00CED1'],wedgeprops={'edgecolor':'k'})

ax[1].set\_title('Unit Sold by Category in 2023', color= subtext\_color)

# Display the charts

plt.tight\_layout()

plt.show()



**Product Performance (total Sales)**

sales\_with\_price = sales\_with\_category

# Calculate Total Sales (Units \* Price)

sales\_with\_price['Total\_Sales'] = sales\_with\_price['Units'] \* sales\_with\_price['Product\_Price']

# Filter for 2022 and 2023

sales\_2022 = sales\_with\_price[sales\_with\_price['Year'] == 2022]

sales\_2023 = sales\_with\_price[sales\_with\_price['Year'] == 2023]

# Top 5 products by sales (Units \* Price) in 2022

top\_5\_products\_sales\_2022 = sales\_2022.groupby('Product\_Name')['Total\_Sales'].sum().nlargest(5).reset\_index()

# Bottom 5 products by sales (Units \* Price) in 2022

bottom\_5\_products\_sales\_2022 = sales\_2022.groupby('Product\_Name')['Total\_Sales'].sum().nsmallest(5).reset\_index()

# Top 5 products by sales (Units \* Price) in 2023

top\_5\_products\_sales\_2023 = sales\_2023.groupby('Product\_Name')['Total\_Sales'].sum().nlargest(5).reset\_index()

# Bottom 5 products by sales (Units \* Price) in 2023

bottom\_5\_products\_sales\_2023 = sales\_2023.groupby('Product\_Name')['Total\_Sales'].sum().nsmallest(5).reset\_index()

# Create the first plot for Top 5 Products in 2022 and 2023

fig1, ax1 = plt.subplots(1, 2, figsize=(16, 6))

# Bar chart for Top 5 Best-Selling Products in 2022

ax1[0].barh(top\_5\_products\_sales\_2022['Product\_Name'], top\_5\_products\_sales\_2022['Total\_Sales'], color=primary\_colors[3], edgecolor='none')

ax1[0].set\_title('Top 5 Best-Selling Products in 2022', color=subtext\_color)

ax1[0].set\_xlabel('Total Sales', color=text\_color)

ax1[0].set\_ylabel('Product Name', color=text\_color)

ax1[0].grid(False)

# Bar chart for Top 5 Best-Selling Products in 2023

ax1[1].barh(top\_5\_products\_sales\_2023['Product\_Name'], top\_5\_products\_sales\_2023['Total\_Sales'], color=primary\_colors[1], edgecolor='none')

ax1[1].set\_title('Top 5 Best-Selling Products in 2023', color=subtext\_color)

ax1[1].set\_xlabel('Total Sales', color=text\_color)

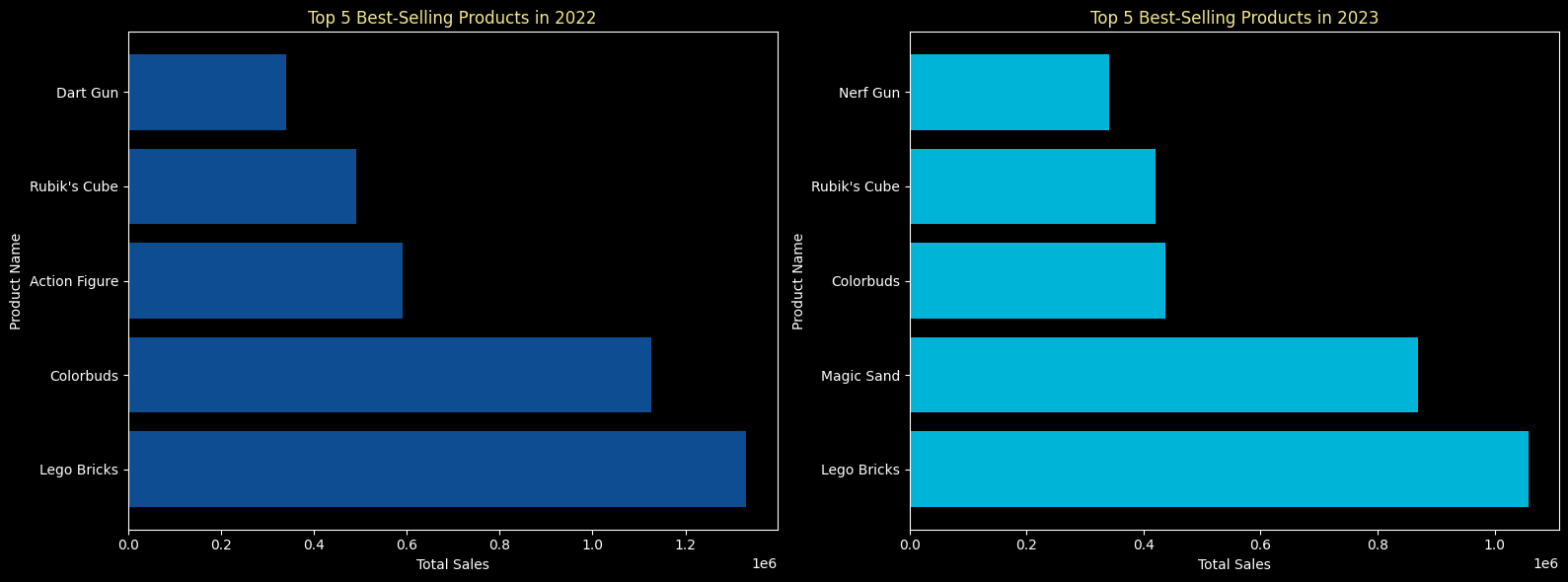
ax1[1].set\_ylabel('Product Name', color=text\_color)

ax1[1].grid(False)

# Adjust layout for better visibility

fig1.tight\_layout()

plt.show()



# Create the second plot for Bottom 5 Products in 2022 and 2023

fig2, ax2 = plt.subplots(1, 2, figsize=(16, 6))

# Bar chart for Bottom 5 Worst-Selling Products in 2022

ax2[0].barh(bottom\_5\_products\_sales\_2022['Product\_Name'], bottom\_5\_products\_sales\_2022['Total\_Sales'], color=primary\_colors[5], edgecolor='none')

ax2[0].set\_title('Bottom 5 Worst-Selling Products in 2022', color=subtext\_color)

ax2[0].set\_xlabel('Total Sales', color=text\_color)

ax2[0].set\_ylabel('Product Name', color=text\_color)

ax2[0].grid(False)

# Bar chart for Bottom 5 Worst-Selling Products in 2023

ax2[1].barh(bottom\_5\_products\_sales\_2023['Product\_Name'], bottom\_5\_products\_sales\_2023['Total\_Sales'], color=primary\_colors[6], edgecolor='none')

ax2[1].set\_title('Bottom 5 Worst-Selling Products in 2023', color=subtext\_color)

ax2[1].set\_xlabel('Total Sales', color=text\_color)

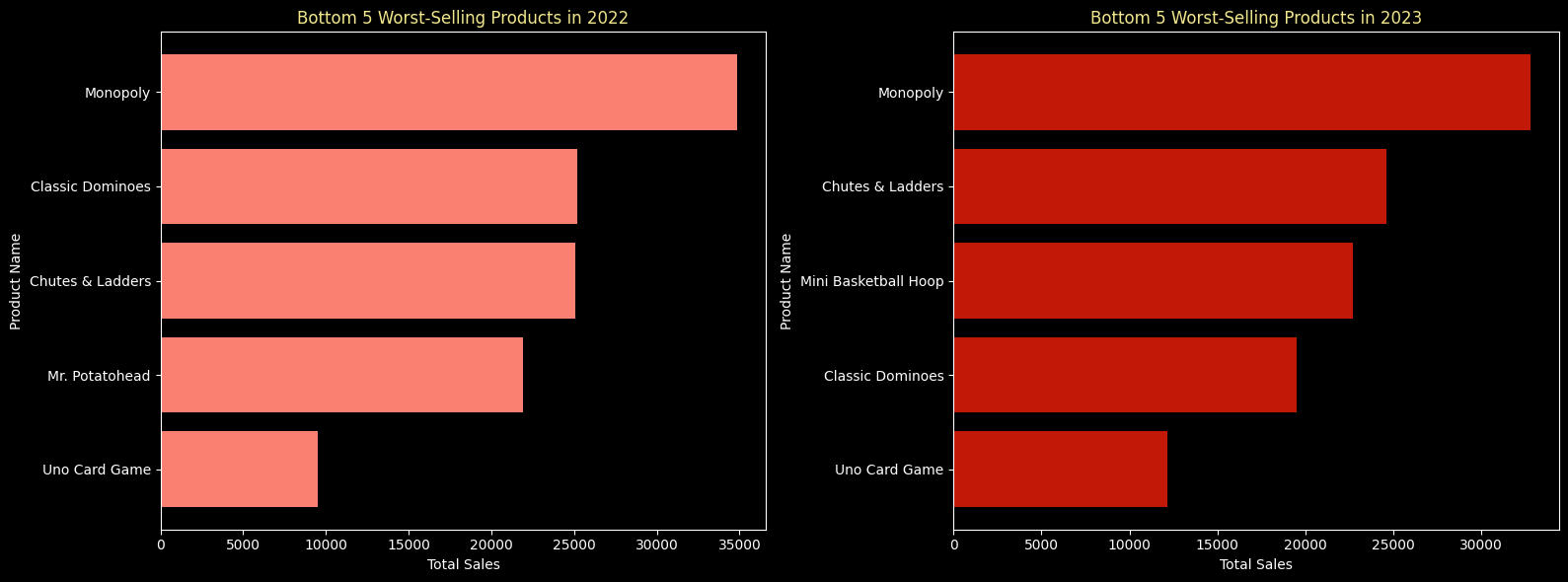
ax2[1].set\_ylabel('Product Name', color=text\_color)

ax2[1].grid(False)

# Adjust layout for better visibility

fig2.tight\_layout()

plt.show()



**Weekly vs Weekday Sale Analysis**

**# Create a new column to identify whether the sale occurred on a weekday or weekend**

**sales\_with\_price['Day\_of\_Week'] = sales\_with\_price['Date'].dt.day\_name()**

**sales\_with\_price['Is\_Weekend'] = sales\_with\_price['Day\_of\_Week'].isin(['Saturday', 'Sunday'])**

**# Calculate total units sold on weekends and weekdays**

**units\_weekend = sales\_with\_price[sales\_with\_price['Is\_Weekend'] == True]['Units'].sum()**

**units\_weekday = sales\_with\_price[sales\_with\_price['Is\_Weekend'] == False]['Units'].sum()**

**units\_per\_day\_weekday = units\_weekday/5**

**units\_per\_day\_weekend = units\_weekend/2**

**units\_per\_day\_total = units\_per\_day\_weekday + units\_per\_day\_weekend**

**perday\_weekday = (units\_per\_day\_weekday/units\_per\_day\_total)\*100**

**perday\_weekend = (units\_per\_day\_weekend/units\_per\_day\_total)\*100**

**# Calculate total units sold**

**total\_units = sales\_with\_price['Units'].sum()**

**# Calculate percentages**

**percent\_weekend = (units\_weekend / total\_units) \* 100**

**percent\_weekday = (units\_weekday / total\_units) \* 100**

**# Data for visualization**

labels = ['Weekends', 'Weekdays']

sizes = [percent\_weekend, percent\_weekday]

colors = ['#0e98ba','#00CED1']

explode = (0.1, 0)  # Explode the weekend slice for emphasis

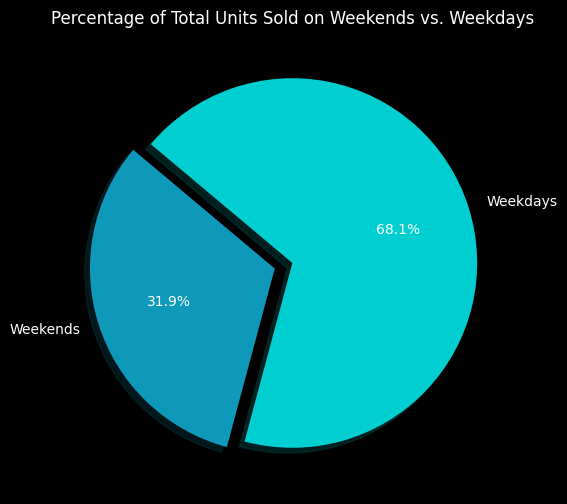
**# Create a pie chart**

plt.figure(figsize=(8, 6))

plt.pie(sizes, explode=explode, labels=labels, colors=colors, autopct='%1.1f%%', shadow=True, startangle=140, wedgeprops={'edgecolor':'none'})

plt.title('Percentage of Total Units Sold on Weekends vs. Weekdays')

plt.show()



**# Per day units sold percentage of weekday and weekend**

labels = ['Weekends', 'Weekdays']

sizes = [perday\_weekend, perday\_weekday]

colors = ['#00B4D8', '#0E4D92']

explode = (0, 0)  # Explode the weekend slice for emphasis

**# Create a pie chart**

plt.figure(figsize=(8, 6))

wedges, texts, autotexts = plt.pie(sizes, explode=explode, labels=labels, colors=colors, autopct='%1.1f%%', shadow=True, startangle=140, wedgeprops={'edgecolor':'black'})

**# Draw a circle at the center of the pie chart to create a donut shape**

centre\_circle = plt.Circle((0, 0), 0.70, fc='black')  # Adjust the radius as needed

fig = plt.gcf()

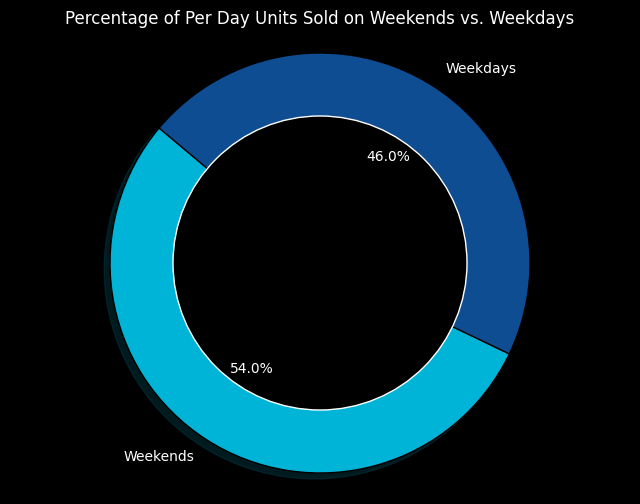
fig.gca().add\_artist(centre\_circle)

**# Equal aspect ratio ensures that pie is drawn as a circle**

plt.axis('equal')

plt.title('Percentage of Per Day Units Sold on Weekends vs. Weekdays')

plt.show()



**Store Analysis**

**# Filter data for years 2022 and 2023**

**#filtered\_sales\_2022\_2023 = sales\_with\_price[sales\_with\_price['Year'].isin([2022, 2023])]**

**#print(sales\_with\_price.head())**

**# Aggregate total sales and total units sold by store and year**

**store\_performance = sales\_with\_price.groupby(['Store\_ID', 'Year']).agg({**

**'Total\_Sales': 'sum',**

**'Units': 'sum'**

**}).reset\_index()**

**# Merge with store\_dim to get store information**

**store\_performance = pd.merge(store\_performance, store\_dim[['Store\_ID', 'Store\_Name']], on='Store\_ID', how='left')**

**# Function to find top 5 stores for a given year**

**def get\_top\_5\_stores(year):**

**# Filter data for the given year**

**top\_5\_sales = store\_performance[store\_performance['Year'] == year].sort\_values(by='Total\_Sales', ascending=False).head(5)**

**top\_5\_units\_sold = store\_performance[store\_performance['Year'] == year].sort\_values(by='Units', ascending=False).head(5)**

**return top\_5\_sales, top\_5\_units\_sold**

**# Get top 5 stores for 2022**

**top\_5\_sales\_2022, top\_5\_units\_sold\_2022 = get\_top\_5\_stores(2022)**

**# Get top 5 stores for 2023**

**top\_5\_sales\_2023, top\_5\_units\_sold\_2023 = get\_top\_5\_stores(2023)**

**def plot\_top\_bottom\_performance\_separate(top\_5\_sales\_2022, top\_5\_sales\_2023, bottom\_5\_sales\_2022, bottom\_5\_sales\_2023):**

**# Create the first figure for Top 5 Stores for 2022 and 2023**

**fig1, ax1 = plt.subplots(1, 2, figsize=(16, 6))**

**# Bar chart for Top 5 Stores by Sales in 2022**

**ax1[0].barh(top\_5\_sales\_2022['Store\_Name'], top\_5\_sales\_2022['Total\_Sales'], color=primary\_colors[1], edgecolor='none')**

**ax1[0].set\_title('Top 5 Stores by Sales in 2022', fontsize=14)**

**ax1[0].set\_xlabel('Total Sales ($)', fontsize=12)**

**ax1[0].set\_ylabel('Store Name', fontsize=12)**

**ax1[0].invert\_yaxis()**

**ax1[0].grid(False)**

**# Bar chart for Top 5 Stores by Sales in 2023**

**ax1[1].barh(top\_5\_sales\_2023['Store\_Name'], top\_5\_sales\_2023['Total\_Sales'], color=primary\_colors[3], edgecolor='none')**

**ax1[1].set\_title('Top 5 Stores by Sales in 2023', fontsize=14)**

**ax1[1].set\_xlabel('Total Sales ($)', fontsize=12)**

**ax1[1].set\_ylabel('Store Name', fontsize=12)**

**ax1[1].invert\_yaxis()**

**ax1[1].grid(False)**

**# Adjust layout for better readability**

**fig1.tight\_layout()**

**plt.show()**

**# Create the second figure for Bottom 5 Stores for 2022 and 2023**

**fig2, ax2 = plt.subplots(1, 2, figsize=(16, 6))**

**# Bar chart for Bottom 5 Stores by Sales in 2022**

**ax2[0].barh(bottom\_5\_sales\_2022['Store\_Name'], bottom\_5\_sales\_2022['Total\_Sales'], color=primary\_colors[5], edgecolor='none')**

**ax2[0].set\_title('Bottom 5 Stores by Sales in 2022', fontsize=14)**

**ax2[0].set\_xlabel('Total Sales ($)', fontsize=12)**

**ax2[0].set\_ylabel('Store Name', fontsize=12)**

**ax2[0].invert\_yaxis()**

**ax2[0].grid(False)**

**# Bar chart for Bottom 5 Stores by Sales in 2023**

**ax2[1].barh(bottom\_5\_sales\_2023['Store\_Name'], bottom\_5\_sales\_2023['Total\_Sales'], color=primary\_colors[6], edgecolor='none')**

**ax2[1].set\_title('Bottom 5 Stores by Sales in 2023', fontsize=14)**

**ax2[1].set\_xlabel('Total Sales ($)', fontsize=12)**

**ax2[1].set\_ylabel('Store Name', fontsize=12)**

**ax2[1].invert\_yaxis()**

**ax2[1].grid(False)**

**# Adjust layout for better readability**

**fig2.tight\_layout()**

**plt.show()**

**# Get top and bottom 5 stores for 2022 and 2023**

**top\_5\_sales\_2022 = store\_performance[store\_performance['Year'] == 2022].sort\_values(by='Total\_Sales', ascending=False).head(5)**

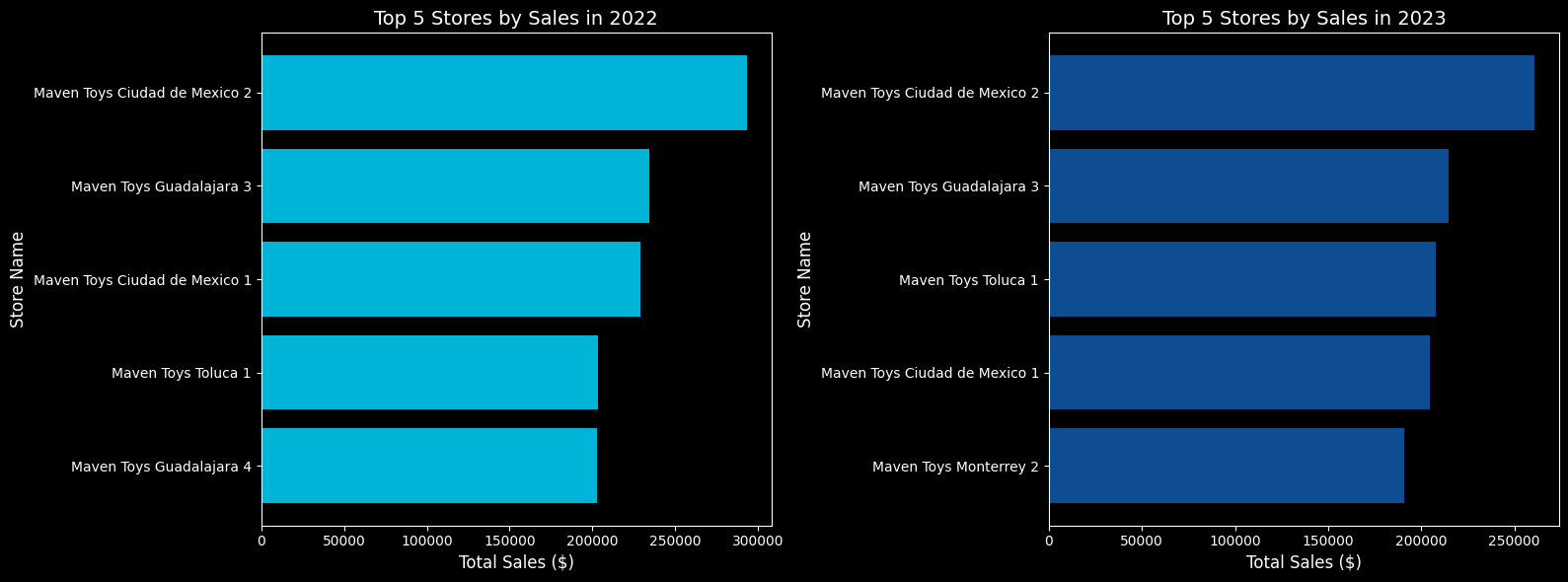
**top\_5\_sales\_2023 = store\_performance[store\_performance['Year'] == 2023].sort\_values(by='Total\_Sales', ascending=False).head(5)**

**bottom\_5\_sales\_2022 = store\_performance[store\_performance['Year'] == 2022].sort\_values(by='Total\_Sales', ascending=True).head(5)**

**bottom\_5\_sales\_2023 = store\_performance[store\_performance['Year'] == 2023].sort\_values(by='Total\_Sales', ascending=True).head(5)**

**# Plot the top and bottom performing stores separately**

**plot\_top\_bottom\_performance\_separate(top\_5\_sales\_2022, top\_5\_sales\_2023, bottom\_5\_sales\_2022, bottom\_5\_sales\_2023)**



A screenshot of a computer screen

Description automatically generated

**Product Analysis**

a) Top 5 Products

# Group by Product Name and Year to calculate total units sold

units\_sold\_by\_product\_year = sales\_with\_date.groupby(['Year', 'Product\_Name'])['Units'].sum().reset\_index()

# Get top 5 products by units sold for 2022

top\_5\_products\_unit\_2022 = units\_sold\_by\_product\_year[units\_sold\_by\_product\_year['Year'] == 2022].nlargest(5, 'Units')

# Get top 5 products by units sold for 2023

top\_5\_products\_unit\_2023 = units\_sold\_by\_product\_year[units\_sold\_by\_product\_year['Year'] == 2023].nlargest(5, 'Units')

# Create subplots for the two bar charts (one for each year)

fig, ax = plt.subplots(1, 2, figsize=(14, 6))

# Bar chart for 2022

ax[0].barh(top\_10\_products\_unit\_2022['Product\_Name'], top\_10\_products\_unit\_2022['Units'], color=primary\_colors[3], edgecolor='none')

ax[0].set\_title('Top 10 Best-Selling Products(Unit Wise) in 2022', color = subtext\_color)

ax[0].set\_xlabel('Units Sold',color= text\_color)

ax[0].set\_ylabel('Product Name', color= text\_color)

ax[0].grid(False)

# Bar chart for 2023

ax[1].barh(top\_10\_products\_unit\_2023['Product\_Name'], top\_10\_products\_unit\_2023['Units'], color=primary\_colors[1], edgecolor='none')

ax[1].set\_title('Top 10 Best-Selling Products(Unit Wise) in 2023',color = subtext\_color)

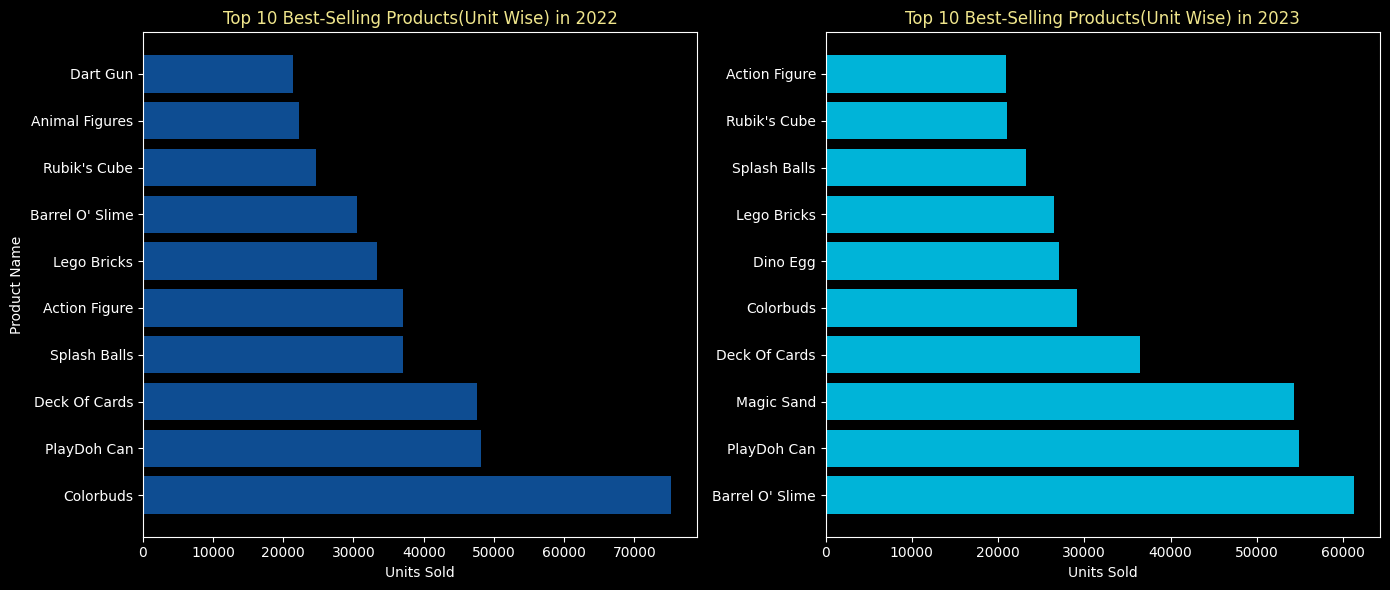
ax[1].set\_xlabel('Units Sold', color= text\_color)

ax[1].grid(False)

# Adjust layout for better visibility

plt.tight\_layout()

plt.show()



# Get bottom 10 products by units sold for 2022

bottom\_10\_products\_unit\_2022 = units\_sold\_by\_product\_year[units\_sold\_by\_product\_year['Year'] == 2022].nsmallest(10, 'Units')

bottom\_10\_products\_unit\_2023 = units\_sold\_by\_product\_year[units\_sold\_by\_product\_year['Year'] == 2023].nsmallest(10, 'Units')

# Worset Performing Products Two bar charts (one for each year)

fig, ax = plt.subplots(1, 2, figsize=(14, 6))

# Bar chart for 2022

ax[0].barh(bottom\_10\_products\_unit\_2022['Product\_Name'], bottom\_10\_products\_unit\_2022['Units'], color=primary\_colors[5], edgecolor='none')

ax[0].set\_title('Bottom 10 Worst-Selling Products(Unit Wise) in 2022', color= subtext\_color)

ax[0].set\_xlabel('Units Sold', color=text\_color)

ax[0].set\_ylabel('Product Name', color=text\_color)

ax[0].grid(False)

# Bar chart for 2023

ax[1].barh(bottom\_10\_products\_unit\_2023['Product\_Name'], bottom\_10\_products\_unit\_2023['Units'], color=primary\_colors[6], edgecolor='none')

ax[1].set\_title('Bottom 10 Worst-Selling Products(Unit Wise) in 2023', color = subtext\_color)

ax[1].set\_xlabel('Units Sold', color= text\_color)

ax[1].grid(False)

# Adjust layout for better visibility

plt.tight\_layout()

plt.show()

A screenshot of a graph

Description automatically generated

**Top 10 Costly and least Costly Products**

**sales\_with\_price['Product\_Cost'] = pd.to\_numeric(sales\_with\_price['Product\_Cost'], errors='coerce')**

**# Drop rows where 'Product\_Cost' is NaN**

**sales\_with\_price = sales\_with\_price.dropna(subset=['Product\_Cost'])**

**# Find top 10 most expensive and top 10 cheapest products**

**top\_10\_costliest = sales\_with\_price.groupby('Product\_ID').agg({'Product\_Cost': 'mean'}).nlargest(10, 'Product\_Cost').reset\_index()**

**top\_10\_cheapest = sales\_with\_price.groupby('Product\_ID').agg({'Product\_Cost': 'mean'}).nsmallest(10, 'Product\_Cost').reset\_index()**

**# Merge with product\_dim to get product names**

**top\_10\_costliest = pd.merge(top\_10\_costliest, product\_dim[['Product\_ID', 'Product\_Name']], on='Product\_ID', how='left')**

**top\_10\_cheapest = pd.merge(top\_10\_cheapest, product\_dim[['Product\_ID', 'Product\_Name']], on='Product\_ID', how='left')**

**# Plot the data**

**fig, ax = plt.subplots(1, 2, figsize=(16, 8))**

**# Plot for the 10 costliest products**

**ax[0].barh(top\_10\_costliest['Product\_Name'], top\_10\_costliest['Product\_Cost'], color=primary\_colors[3], edgecolor='none')**

**ax[0].set\_title('Top 10 Costliest Products')**

**ax[0].set\_xlabel('Product Cost ($)')**

**ax[0].invert\_yaxis()  # Invert y-axis for better readability**

**ax[0].grid(False)**

**# Plot for the 10 cheapest products**

**ax[1].barh(top\_10\_cheapest['Product\_Name'], top\_10\_cheapest['Product\_Cost'], color=primary\_colors[1], edgecolor='none')**

**ax[1].set\_title('Top 10 Cheapest Products')**

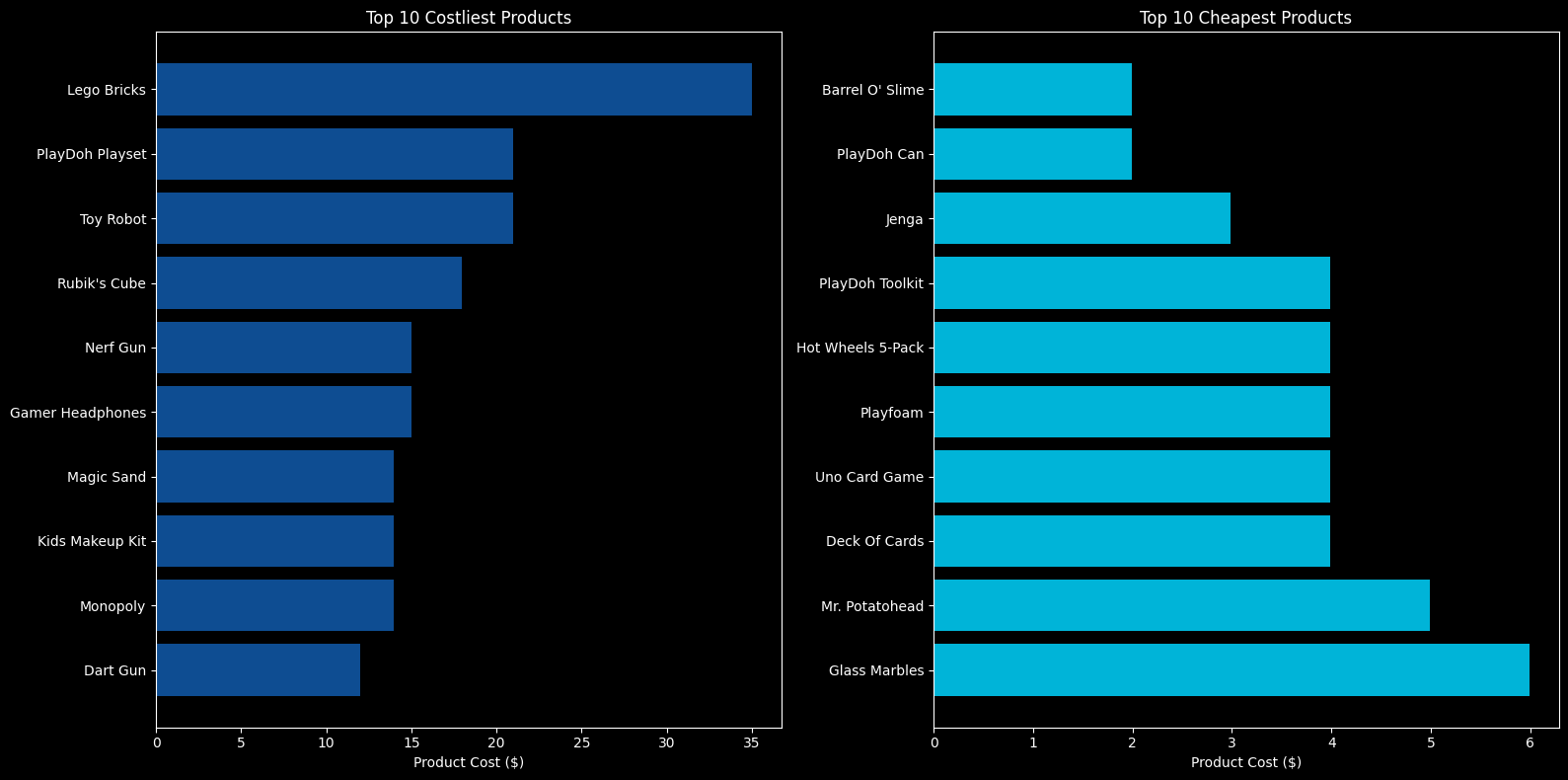
**ax[1].set\_xlabel('Product Cost ($)')**

**ax[1].invert\_yaxis()  # Invert y-axis for better readability**

**ax[1].grid(False)**

**plt.tight\_layout()**

**plt.show()**



**Profit Margin**

**sales\_with\_price['Total\_Sales'] = pd.to\_numeric(sales\_with\_price['Total\_Sales'], errors='coerce')**

**sales\_with\_price['Product\_Cost'] = pd.to\_numeric(sales\_with\_price['Product\_Cost'], errors='coerce')**

**# Drop rows where 'Total\_Sales' or 'Product\_Cost' is NaN (if any)**

**sales\_with\_price = sales\_with\_price.dropna(subset=['Total\_Sales', 'Product\_Cost'])**

**# Calculate profit margin for each product**

**sales\_with\_price['Profit\_Margin'] = (sales\_with\_price['Total\_Sales'] - sales\_with\_price['Product\_Cost']) / sales\_with\_price['Total\_Sales'] \* 100**

**# Aggregate total sales and units sold for each product**

**product\_performance = sales\_with\_price.groupby('Product\_ID').agg({**

**'Total\_Sales': 'sum',**

**'Units': 'sum’, # Use 'Units' instead of 'Units\_Sold'**

**'Profit\_Margin': 'mean'}). reset\_index()**

**# Merge with product\_dim to get product names (if available)**

**product\_performance = pd.merge(product\_performance, product\_dim[['Product\_ID', 'Product\_Name']], on='Product\_ID', how='left')**

**# Scatter plot for Sales Volume vs. Profit Margin**

**plt.figure(figsize=(12, 8))**

**scatter = plt.scatter(product\_performance['Units'], product\_performance['Profit\_Margin'], color=primary\_colors[0], s=100, alpha=0.8, edgecolors='grey')**

**plt.title('Sales Volume vs. Profit Margin Analysis', color = subtext\_color)**

**plt.xlabel('Total Units Sold', color=text\_color)**

**plt.ylabel('Profit Margin (%)', color = text\_color)**

**plt.colorbar(scatter, label='Total Sales ($)')**

**plt.grid(False)**

**# Highlight some specific products if necessary**

**for i, row in product\_performance.iterrows():**

**if row['Units'] > 1000 or row['Profit\_Margin'] < 5: # Example condition to highlight products**

**plt.annotate(row['Product\_Name'], (row['Units'], row['Profit\_Margin']), fontsize=8, alpha=0.75)**

**plt.show()**

A screenshot of a computer screen

Description automatically generated

**Inventory Analysis**

# Group the inventory by Product\_ID to sum the Stock\_On\_Hand for each product

product\_stock\_summary = inventory\_with\_category.groupby('Product\_ID').agg({'Stock\_On\_Hand': 'sum'}).reset\_index()

# Merge with inventory\_with\_category to get unique product names and categories

product\_stock\_summary = pd.merge(

    product\_stock\_summary,

    inventory\_with\_category[['Product\_ID', 'Product\_Name', 'Product\_Category']].drop\_duplicates(),

    on='Product\_ID',

    how='left'

)

# Define thresholds for overstocked and critically low stock

overstocked\_threshold = 1200

low\_stock\_threshold = 250

# Apply thresholds to categorize stock levels

def categorize\_stock(stock):

    if stock > overstocked\_threshold:

        return 'Overstocked'

    elif stock < low\_stock\_threshold:

        return 'Critically Low'

    else:

        return 'Normal Stock'

# Categorize stock levels based on thresholds

product\_stock\_summary['Stock\_Status'] = product\_stock\_summary['Stock\_On\_Hand'].apply(categorize\_stock)

# Group by Stock\_Status to count the number of products in each category

stock\_status\_counts = product\_stock\_summary['Stock\_Status'].value\_counts().reset\_index()

stock\_status\_counts.columns = ['Stock\_Status', 'Count']

# Visualization using Matplotlib

plt.figure(figsize=(10, 6))

# Plotting the bar chart for stock status counts

plt.bar(stock\_status\_counts['Stock\_Status'], stock\_status\_counts['Count'], color=['#00B4D8', '#0E4D92', '#FA8072'], edgecolor='none')

# Adding title and labels

plt.title('Inventory Stock Status Overview', fontsize=16, color='white')

plt.xlabel('Stock Status', fontsize=14, color='white')

plt.ylabel('Number of Products', fontsize=14, color='white')

# Setting black background

plt.gca().set\_facecolor('black')

plt.gcf().patch.set\_facecolor('black')

# Making axis labels white

plt.tick\_params(colors='white')

# Display the plot

plt.tight\_layout()

plt.show()

# Generate lists of critically low and overstocked products

critically\_low\_products = product\_stock\_summary[product\_stock\_summary['Stock\_Status'] == 'Critically Low'][['Product\_Name', 'Product\_Category', 'Stock\_On\_Hand']]

overstocked\_products = product\_stock\_summary[product\_stock\_summary['Stock\_Status'] == 'Overstocked'][['Product\_Name', 'Product\_Category', 'Stock\_On\_Hand']]

# Print the list of critically low products

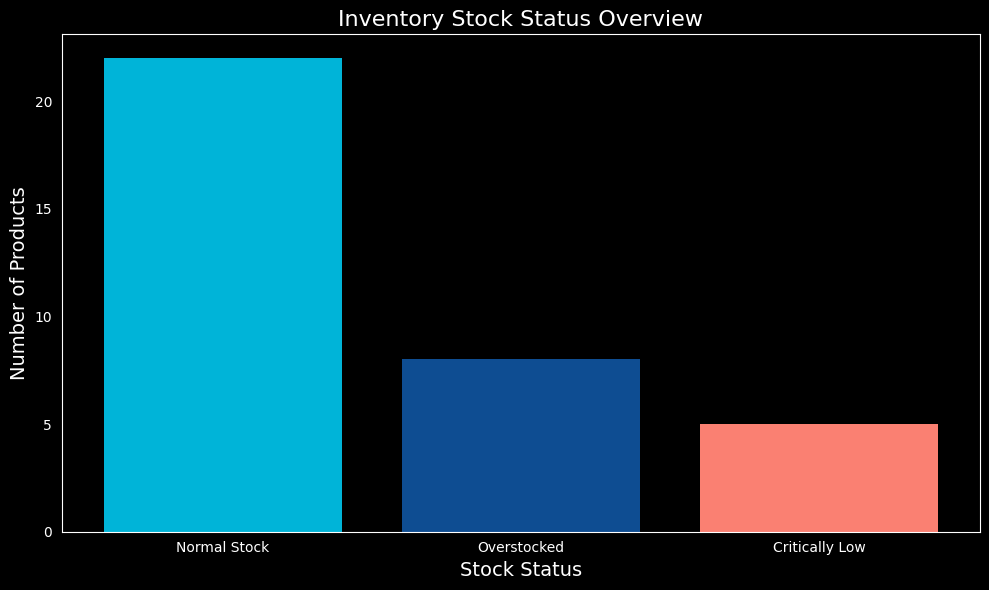
print("Critically Low Products:")

print(critically\_low\_products.to\_string(index=False))

# Print the list of overstocked products

print("\nOverstocked Products:")

print(overstocked\_products.to\_string(index=False))





**Product Demand vs Stock**

# Filter data for the years 2022 and 2023

sales\_filtered = sales\_with\_price[sales\_with\_price['Year']. isin([2022, 2023])]

# Group by Product\_ID to calculate the average units sold in 2022 and 2023

avg\_units\_sold = sales\_filtered.groupby('Product\_ID').agg({

    'Units': 'sum'  # Taking the average units sold

}).reset\_index()

avg\_units\_sold['Units'] = avg\_units\_sold['Units'] / 2  # Divide by 2 to get the average

avg\_units\_sold['Units'] = avg\_units\_sold['Units']/12 # Divide by 12 to get average /month units sold

avg\_units\_sold['Units'] = avg\_units\_sold['Units'].round(0)

# Rename the column for clarity

avg\_units\_sold.rename(columns={'Units': 'Avg\_Units\_Sold\_Monthly'}, inplace=True)

# Merge with product\_dim to get product names

avg\_units\_sold = pd.merge(avg\_units\_sold, product\_dim[['Product\_ID', 'Product\_Name']], on='Product\_ID', how='left')

#  Sort by Avg\_Units\_Sold to find the top 10 popular products

top\_10\_products = avg\_units\_sold.sort\_values(by='Avg\_Units\_Sold\_Monthly', ascending=False).head(10)

# Get the stock availability from the inventory table

product\_stock = inventory.groupby('Product\_ID').agg({'Stock\_On\_Hand': 'sum'}).reset\_index()

#  Merge the top 10 products with their stock availability

top\_10\_products\_with\_stock = pd.merge(top\_10\_products, product\_stock, on='Product\_ID', how='left')

#  Sort based on Avg\_Units\_Sold to get the final data for analysis

top\_10\_products\_with\_stock = top\_10\_products\_with\_stock.sort\_values(by='Avg\_Units\_Sold\_Monthly', ascending=False)

# Visualize the analysis

plt.figure(figsize=(12, 6))

sns.barplot(x='Product\_Name', y='Avg\_Units\_Sold\_Monthly', data=top\_10\_products\_with\_stock, color=primary\_colors[3], label='Avg Units Sold', edgecolor='none')

sns.barplot(x='Product\_Name', y='Stock\_On\_Hand', data=top\_10\_products\_with\_stock, color=primary\_colors[1], alpha=0.5, label='Stock On Hand', edgecolor='none')

plt.title('Top 10 Popular Products (Avg Units Sold\_Monthly vs Stock Availability)')

plt.xlabel('Product Name')

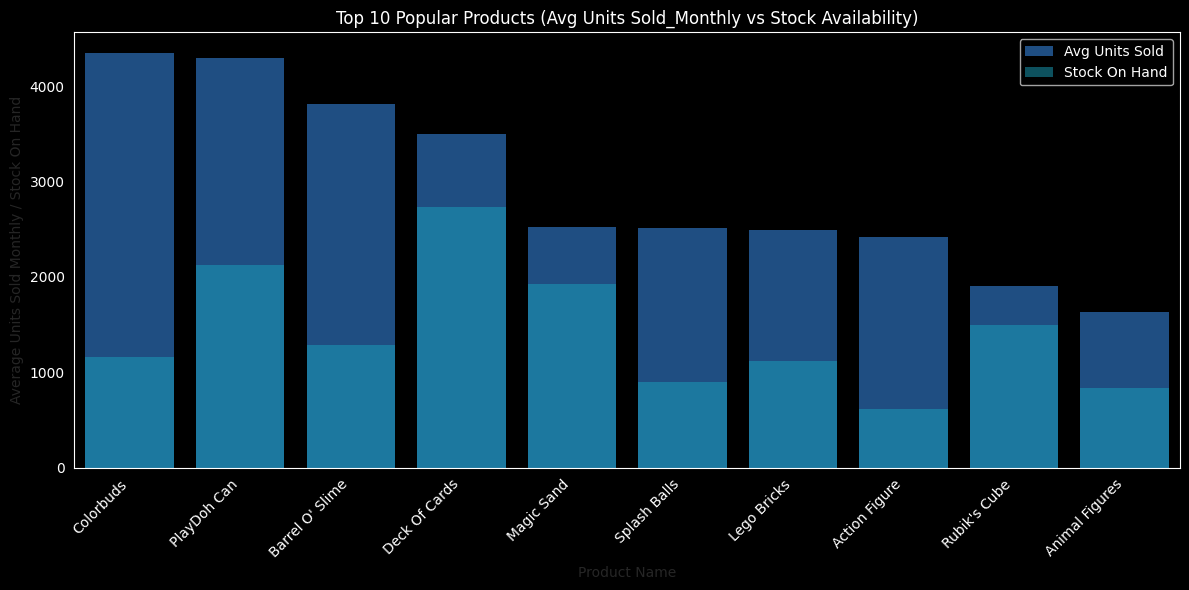
plt.ylabel('Average Units Sold Monthly / Stock On Hand')

plt.xticks(rotation=45, ha='right')

plt.legend(loc='upper right')

plt.tight\_layout()

plt.show()



Bottom\_10\_products = avg\_units\_sold.sort\_values(by='Avg\_Units\_Sold\_Monthly', ascending=True).head(10)

#  Merge the Bottom 10 products with their stock availability

Bottom\_10\_products\_with\_stock = pd.merge(Bottom\_10\_products, product\_stock, on='Product\_ID', how='left')

#  Sort based on Avg\_Units\_Sold to get the final data for analysis

Bottom\_10\_products\_with\_stock = Bottom\_10\_products\_with\_stock.sort\_values(by='Avg\_Units\_Sold\_Monthly', ascending=False)

# Visualize the analysis

plt.figure(figsize=(12, 6))

sns.barplot(x='Product\_Name', y='Avg\_Units\_Sold\_Monthly', data=Bottom\_10\_products\_with\_stock, color=primary\_colors[6], label='Avg Units Sold', edgecolor='none')

sns.barplot(x='Product\_Name', y='Stock\_On\_Hand', data=Bottom\_10\_products\_with\_stock, color=primary\_colors[5], alpha=0.5, label='Stock On Hand',edgecolor='none')

plt.title('Bottom 10 Popular Products (Avg Units Sold\_Monthly vs Stock Availability)')

plt.xlabel('Product Name')

plt.ylabel('Average Units Sold Monthly / Stock On Hand')

plt.xticks(rotation=45, ha='right')

plt.legend(loc='upper right')

plt.tight\_layout()

plt.show()

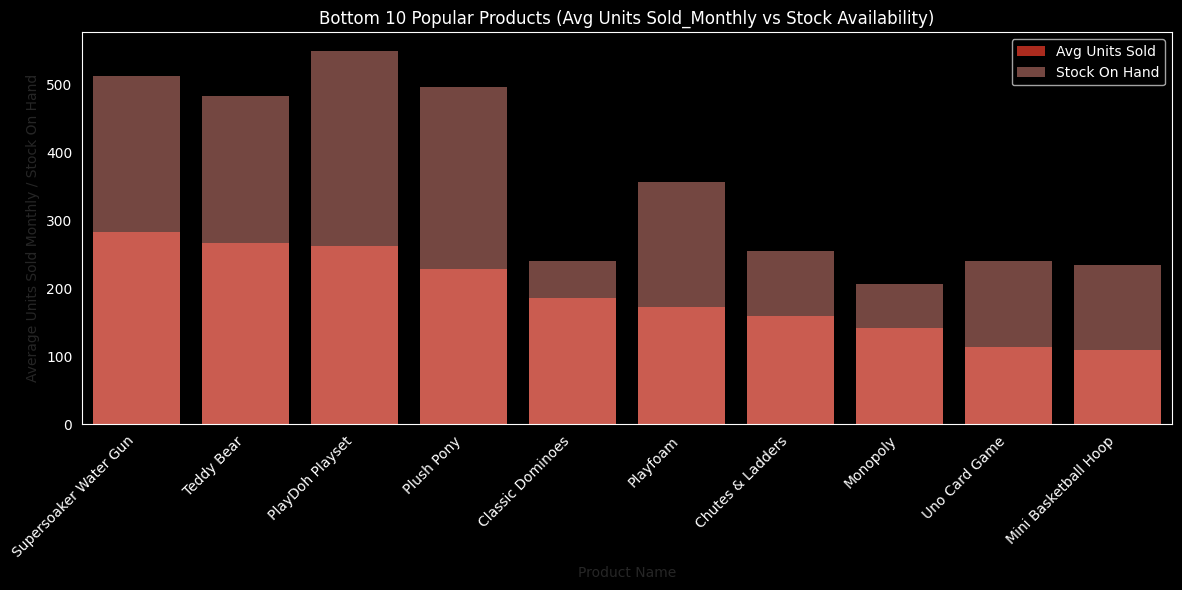


Fig:

**Inventory Distribution**

**# Group by City and Category to calculate total stock**

**city\_category\_stock = inventory\_with\_store.groupby(['Store\_City', 'Product\_Category'])['Stock\_On\_Hand'].sum().reset\_index()**

**# Create a pivot table for better visualization (Cities as rows, Categories as columns)**

**pivot\_city\_stock = city\_category\_stock.pivot(index='Store\_City', columns='Product\_Category', values='Stock\_On\_Hand')**

**# Plot the heatmap for City-wise and Category-wise stock distribution**

**plt.figure(figsize=(10, 8))**

**sns.set\_style("dark")  # Set seaborn style to dark for better appearance on black background**

**# Set background and text colors manually**

**plt.rcParams['axes.facecolor'] = 'black'  # Set background color to black**

**plt.rcParams['figure.facecolor'] = 'black'  # Set figure background color to black**

**plt.rcParams['text.color'] = 'white'  # Set text color to white**

**plt.rcParams['xtick.color'] = 'white'  # Set x-tick label color to white**

**plt.rcParams['ytick.color'] = 'white'  # Set y-tick label color to white**

**# Plot the heatmap**

**sns.heatmap(pivot\_city\_stock, annot=True, cmap=primary\_colors, fmt='.0f', linewidths=.5,**

**cbar\_kws={'label': 'Stock On Hand'})  # Set color bar label**

**# Customize title and labels with white color**

**plt.title('City-wise and Category-wise Stock Distribution', fontsize=14, color='white')**

**plt.xlabel('Category', fontsize=12, color='white')**

**plt.ylabel('City', fontsize=12, color='white')**

**plt.show()**

A screen shot of a chart

Description automatically generated