**TechSphere Global Electronics Retailer**

Today, we own a large global electronics retail company, **TechSphere**, which operates through physical and online stores. We sell a wide variety of electronics products from different categories and brands.

We will work with multiple tables containing information about customers, products, sales, stores, and exchange rates. We have also been provided with a data dictionary.

Here is a look at our data dictionary, which gives us an idea of the tables we will work with.

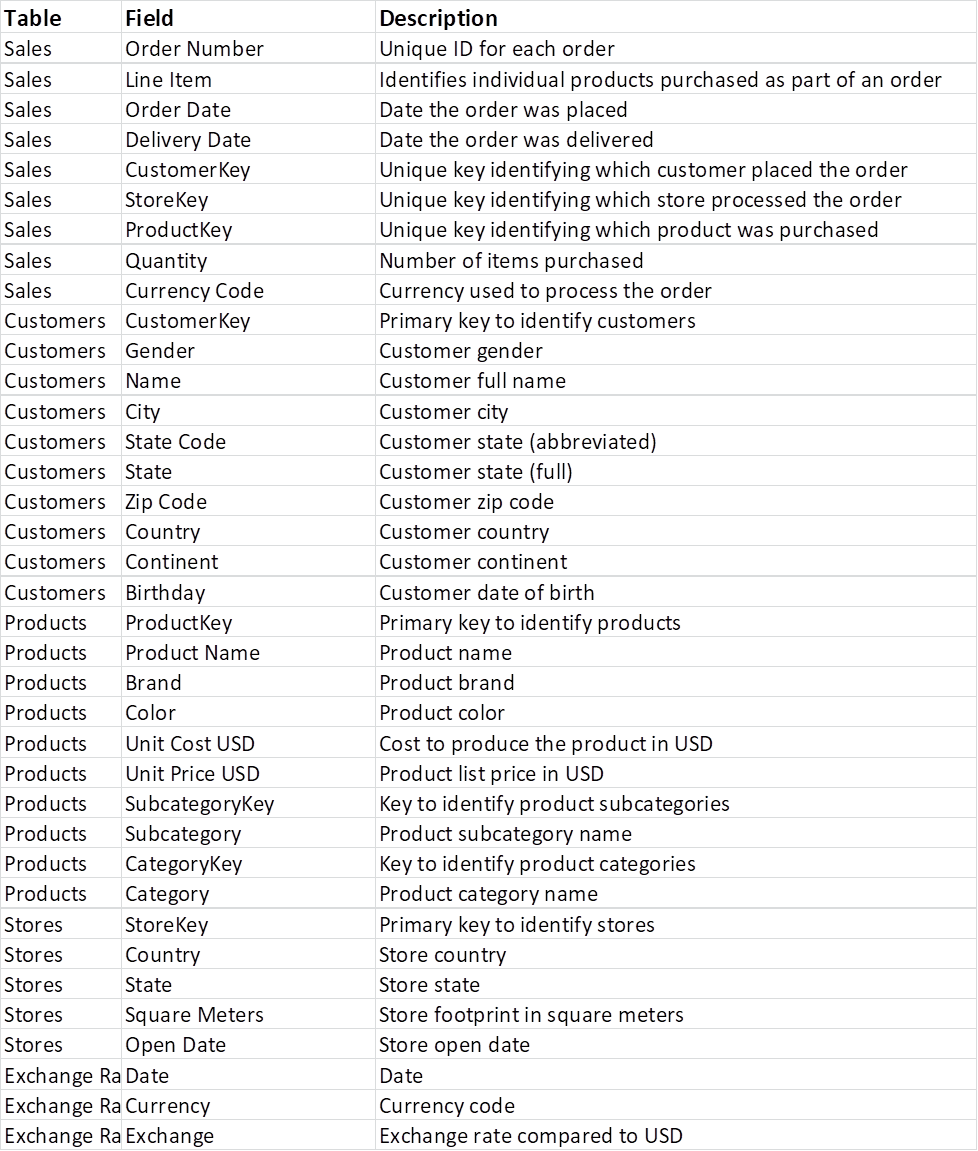


Table of Contents

* [Objectives](#_9ur8zxkzqxh)
* [Technical Methods Used](#_otvtlpm2vopw)
* [Loading our datasets](#_v5ibj09grfs7)
* [Counting rows of each table](#_7qwr7ehrrafa)
* [Count of Distinct Records in Each Table](#_xvvr2qaurrcw)
* [Transforming Data in Products Table](#_qvqy4yckusrd)
* [Finding Total Profit](#_gecp3hfdpiax)
* [Products with No Sales](#_af0fae8g5cx7)
* [Total profit of each product category](#_ok65xji7vrr)
* [Total profit of each product subcategory](#_musk5h58x0u6)
* [Total Profit Brought in by Different Brands](#_9ufu8gatjmao)
* [Profit Generated by Stores](#_pwr8nsgvcjgc)
* [Stores with No Sales](#_cxlro2nnxhya)
* [Profit Made from Customers](#_j1h8bnmzcr6p)
* [Customer Distribution by Gender](#_1rpl4m45o2ze)
* [Sales Information Over Time](#_b7e5atotmrw3)

## **Objectives**

The primary objective of this analysis is to extract key insights that can drive strategic decision-making within the company. This project will lead us to the following information:

**Basic Descriptive Analysis**

* **Total Number of Orders: 62884**
* **Total Sales Quantity: 26326**
* **Number of Unique Customers: 15266**
* **Number of Unique Products: 2517**
* **Number of Stores: 67**

**Customer Analysis**

* **Top Customers by Sales:** Which customers profited most from their purchases?
* **Customer Demographics;** Analyzing customer distribution by gender, country, and continent.

**Product Analysis**

* **Top-Selling Products:** Top-selling products by quantity sold.
* **Revenue by Product:** Total revenue generated by each product.
* **Product Performance by Category:** Analyze which product categories (Category) generate the most sales.
* **Product Subcategory Performance**:

**Store Performance Analysis**

* **Top Performing Stores:** Identify which stores have the highest total sales.
* **Store Sales by Country:** Distribution of sales by country where the stores are located.
* **Sales per Square Meter:** Sales per square meter for each store.

**Sales over time**

* **Sales over time**: Amount of orders made and monthly profit generated every year.

We will see how these questions were answered as we go on with the project.

* Sales Trend Over Time
* Product Performance
* Store Performance
* Customer Lifetime Value (CLV) and Segmentation
* Churn Analysis
* Market and Geographic Analysis
* Impact of Exchange Rates on Sales

## **Technical Methods Used**

We will be making use of **PostgreSQL** with the pgAdmin interface to carry out our analysis and **Python** for visualizations, we will make use of Python alongside **SQL** code and output to help us with more complex analysis. We will use  **Chatgpt**, a generative AI tool to brainstorm information that can be gotten from the data dictionary above, it can also be used to improve the code we’ve written.

LET’S BEGIN!

The datasets used in this project were added to the [**Maven analytics data playground**](https://mavenanalytics.io/data-playground?dataStructure=Multiple%20tables&order=date_added%2Cdesc) on April 1st, 2024.

Right at the start we already have some thinking to do; our dataset consists of 5 tables with some having up to 10 columns. We could easily CREATE each table and INSERT columns either through SQL code or from our pgAdmin tool, but I believe that’s a bit too manual and the major question I asked myself is *“What if we have up to 40 columns?”.*

## **Loading our datasets**

We can use Python to make this easier for us with fewer lines of code using the pandas and sqlalchemy libraries to bring in the files and automatically create and store them in our created database.

| import pandas as pd from sqlalchemy import create\_engine  # Local file paths to the datasets files = [  "C:/Users/HP/Documents/DATA\_PRACTICE/datasets/global\_electronics\_retailer/Customers.csv",  "C:/Users/HP/Documents/DATA\_PRACTICE/datasets/global\_electronics\_retailer/Exchange\_Rates.csv",  "C:/Users/HP/Documents/DATA\_PRACTICE/datasets/global\_electronics\_retailer/Products.csv",  "C:/Users/HP/Documents/DATA\_PRACTICE/datasets/global\_electronics\_retailer/Sales.csv",  "C:/Users/HP/Documents/DATA\_PRACTICE/datasets/global\_electronics\_retailer/Stores.csv" ] # naming the tables that will be created in our database tables = [  "customers",  "exchange\_rates",  "products",  "sales",  "stores" ] |
| --- |

Next, we will make use of a loop to automatically connect to our database server and load the tables straight from our file location into a data frame to our SQL server.

| for i in range(len(files)):  df = pd.read\_csv(files[i], encoding="ISO-8859-1")  engine = create\_engine("postgresql://postgres:mydatabase@localhost:5432/global\_electronics\_retailer")   df.head(0).to\_sql(tables[i], engine, if\_exists="replace", index=False)    df.to\_sql(tables[i], engine, if\_exists="append", index=False) |
| --- |

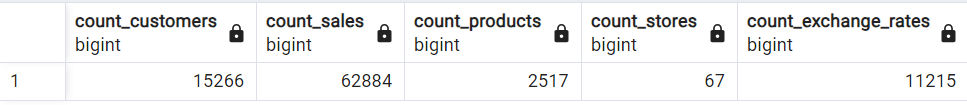
The code reads the CSV files, creates tables in the SQL database, and loads the data from their CSV files into respective tables.

## **Counting rows of each table**

Let’s run some exploratory analysis to understand our data better. We will start by counting the rows of each table.

| SELECT (SELECT COUNT (\*) FROM customers) AS count\_customers,   (SELECT COUNT (\*) FROM sales) AS count\_sales,  (SELECT COUNT(\*) FROM products) AS count\_products,  (SELECT COUNT(\*) FROM stores) AS count\_stores,  (SELECT COUNT(\*) FROM exchange\_rates) AS count\_exchange\_rates |
| --- |

Output:

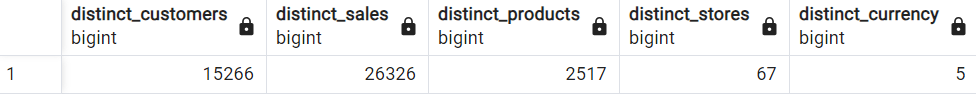


The tables have different numbers of rows, meaning there are multiple occurrences of table records; a single customer might have made multiple sales, and one product may have been purchased multiple times.

## **Count of Distinct Records in Each Table**

We can find out how many unique records there are for each table using the DISTINCT clause.

| SELECT   (SELECT COUNT( DISTINCT "CustomerKey") FROM customers) AS distinct\_customers,  (SELECT COUNT( DISTINCT "Order Number") FROM sales) AS distinct\_sales,  (SELECT COUNT( DISTINCT "ProductKey") FROM products) AS distinct\_products,  (SELECT COUNT( DISTINCT "StoreKey") FROM stores) AS distinct\_stores,  (SELECT COUNT( DISTINCT "Currency") FROM exchange\_rates) AS distinct\_currency; |
| --- |



We notice that the customers, products, and stores tables have their count of distinct values to be equal to their unique identifiers whereas sales and currencies are much less.

This means a sale occurred multiple times, and 5 currencies were used to make different payments for our products. *For this project, all monetary values will be in* ***USD($)***.

Let’s try to find out more about our products table:

* Total profit of each product category
* Which products are the top performers in terms of sales volume and revenue?
* Total profit of each product subcategory
* Total profit from every brand of product
* Products with the most number of orders

But first, it will be more convenient for us to carry out arithmetic operations if we convert the data types of necessary columns. We want to convert the **“Unit Price USD”** and **“Unit Cost USD”** columns to decimal.

## **Transforming Data in Products Table**

| UPDATE products SET "Unit Price USD" = REPLACE(REPLACE("Unit Price USD", '$', ''), ',', ''); |
| --- |

This replaces the “$” and “,” symbols with whitespace to make it a fully numerical format.

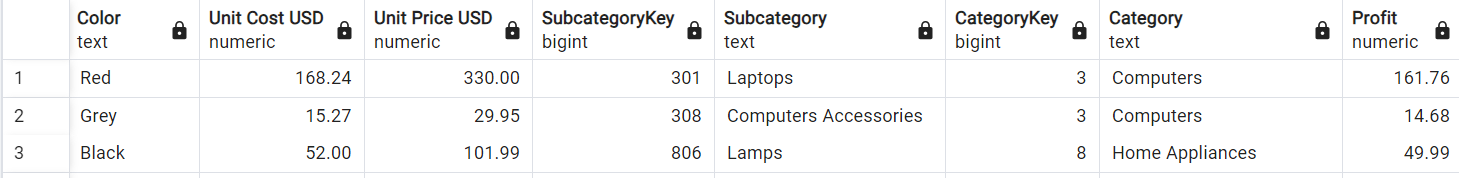
| UPDATE products  SET "Unit Cost USD" = REPLACE(REPLACE("Unit Cost USD", '$', ''), ',', ''); |
| --- |

Now, we can change the data types to DECIMAL and add a “Profit” column to calculate the difference between the price and cost of making our products.

| ALTER TABLE products ALTER COLUMN "Unit Price USD" TYPE DECIMAL USING "Unit Price USD"::NUMERIC, ALTER COLUMN "Unit Cost USD" TYPE DECIMAL USING "Unit Cost USD"::NUMERIC; |
| --- |

| ALTER TABLE products ADD "Profit" DECIMAL;  UPDATE products SET "Profit" = "Unit Price USD" - "Unit Cost USD"; |
| --- |

Now the table looks better, we can now work with values more easily.



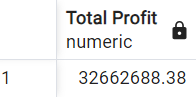
We have transformed the table into a much more suitable format for further analysis. It will be better for us to group product profitability by category or subcategory since our products in the “Product Name” column are all unique.

## **Finding Total Profit**

Before we proceed, let's actually get our total profit for all sales we have made. We can do this by joining our sales and products table.

| SELECT SUM((products."Unit Price USD" - products."Unit Cost USD") \* sales."Quantity") AS "Total Profit" FROM   sales JOIN   products ON sales."ProductKey" = products."ProductKey"; |
| --- |

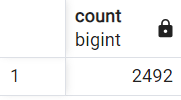
Here, we multiply the profit we made off each individual product and multiply it by the number of times it was bought from the “Quantity”, the output gives us the total amount of profit we have generated.



We have generated a total of over $ 32 million **in profit**.

We can find out the products that were sold by counting the products that appeared on the sales table

| WITH count\_product\_sales AS (SELECT   p."Product Name",   SUM((p."Profit") \* s."Quantity") AS "Total Profit" FROM   sales s JOIN   products p ON s."ProductKey" = p."ProductKey" GROUP BY   p."Product Name" ORDER BY   "Total Profit" DESC) SELECT COUNT("Product Name") FROM count\_product\_sales; |
| --- |

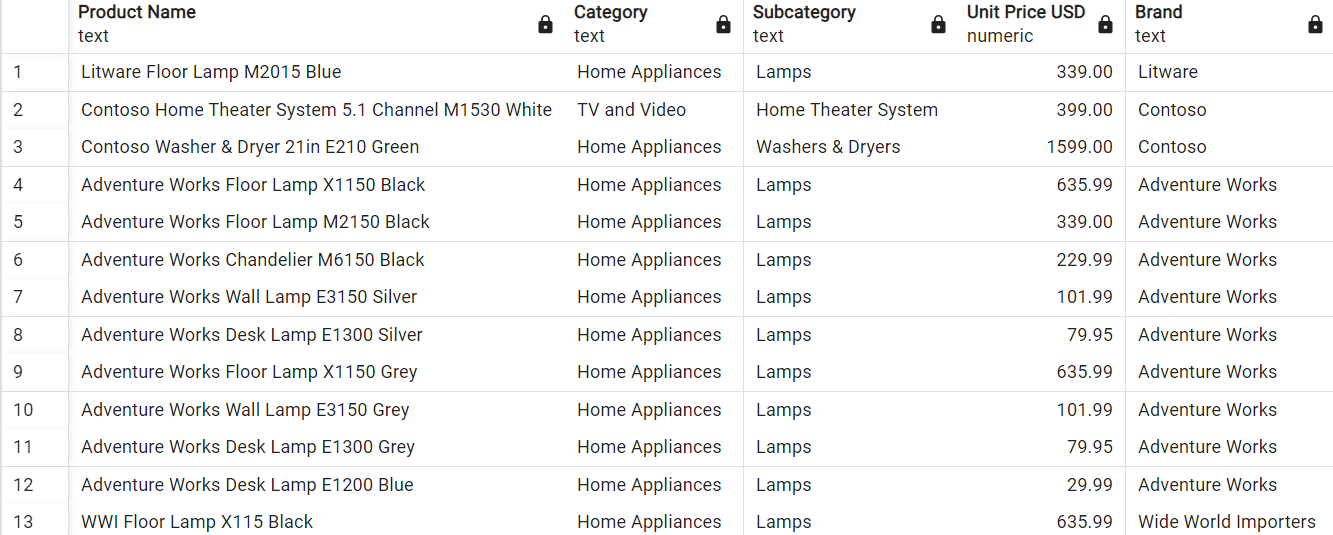


## **Products with No Sales**

We notice that from our above count of distinct products using their unique keys, there are 2,517 products but only 2,492 in our sales table, this means that there are 25 products that did not sell; we can find them by using:

| SELECT p."Product Name", "Category", "Subcategory", "Unit Price USD", "Brand" FROM products p LEFT JOIN sales ON p."ProductKey" = sales."ProductKey" WHERE sales."ProductKey" IS NULL; |
| --- |

Output of first 13 rows:

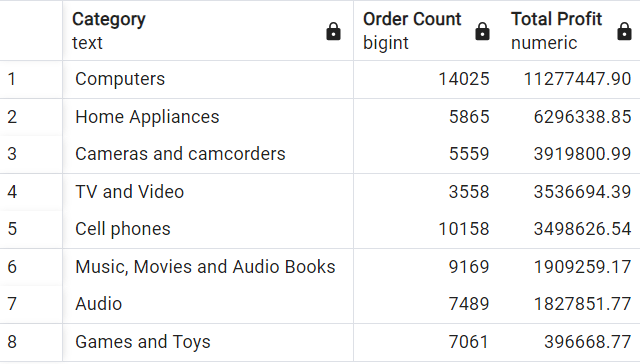


One thing that stood out to me about the output is that **23 out of the 25 products** that did not sell were lamps.

We can group profit got by product category to see our top performers.

## **Total profit of each product category**

| SELECT   p."Category",   COUNT(sales."Order Number") AS "Order Count",   SUM(p."Profit" \* sales."Quantity") AS "Total Profit" FROM   sales JOIN   products p ON sales."ProductKey" = p."ProductKey" GROUP BY   p."Category" ORDER BY   "Total Profit" DESC; |
| --- |



The Computers category is at the top of the list, with almost **$ 5 million** more than home appliances. The games and toys sections have generated the least.

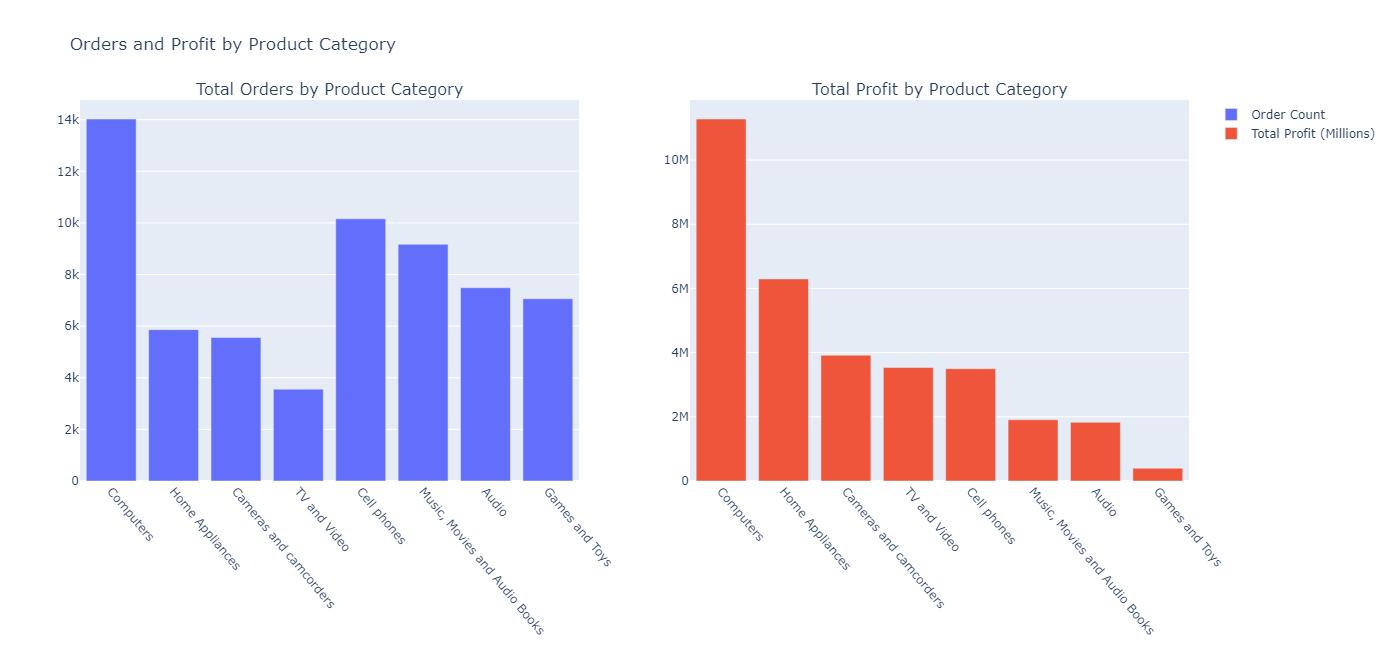
We’ll import the Python libraries that we will use in this project for our visualizations.

| import pandas as pd import plotly.express as px from plotly.subplots import make\_subplots import plotly.graph\_objects as go |
| --- |

| category\_sales\_df = pd.read\_csv("C:/Users/HP/Downloads/Global+Electronics+Retailer/category\_total\_profit.csv") |
| --- |

| fig = make\_subplots(rows=1, cols=2, subplot\_titles=['Total Orders by Product Category', 'Total Profit by Product Category'])  orders\_fig = px.bar(category\_sales\_df, x='Category', y='Order Count')  profit\_fig = px.bar(category\_sales\_df, x='Category', y='Total Profit')  fig.add\_trace(go.Bar(x=category\_sales\_df['Category'], y=category\_sales\_df['Order Count'].sort\_values(ascending=False), name="Order Count"), row=1, col=1) fig.add\_trace(go.Bar(x=category\_sales\_df['Category'], y=category\_sales\_df['Total Profit'], name="Total Profit (Millions)"), row=1, col=2)  fig.update\_layout(height=650, title\_text="Orders and Profit by Product Category") fig.update\_xaxes(tickangle=50)  fig.show() |
| --- |

It gives us bar chart plots of our total orders and profit for each product category.

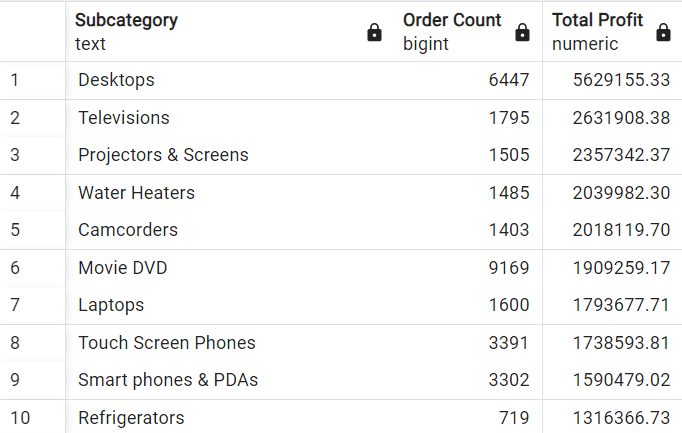
[](https://drive.google.com/file/d/1fthw961IlttwW0vul1GZK-anQTutYQV4/view?usp=drive_link)

We can do the same for our subcategory table to better understand products performing well in each category.

## **Total profit of each product subcategory**

| SELECT   p."Subcategory",   COUNT(sales."Order Number") AS "Order Count",   SUM(p."Profit" \* sales."Quantity") AS "Total Profit" FROM   sales JOIN   products p ON sales."ProductKey" = p."ProductKey" GROUP BY   p."Subcategory" ORDER BY   "Total Profit" DESC; |
| --- |

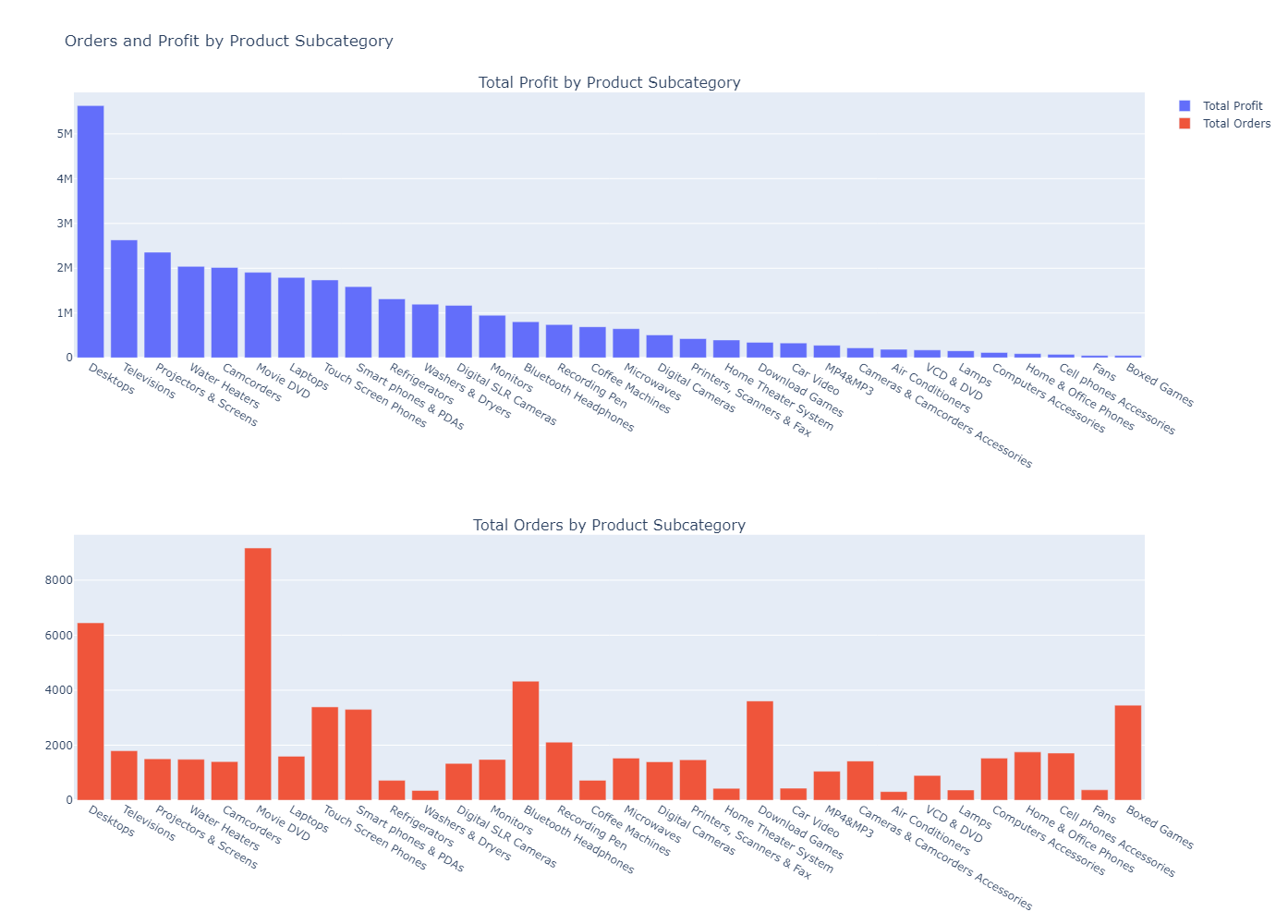
There are 32 unique subcategories but we will limit this output to 10.



We will also import and visualize the saved table with Python.

| subcategory\_sales\_df = pd.read\_csv("C:/Users/HP/Downloads/Global+Electronics+Retailer/subcat\_profit\_and\_orders.csv") |
| --- |

| fig = make\_subplots(rows=2, cols=1, subplot\_titles=["Total Profit by Product Subcategory", "Total Orders by Product Subcategory"])  profit\_fig = px.bar(subcategory\_sales\_df, x="Subcategory", y="Total Profit") orders\_fig = px.bar(subcategory\_sales\_df, x="Subcategory", y="Order Count")  fig.add\_trace(go.Bar(x=subcategory\_sales\_df["Subcategory"], y = subcategory\_sales\_df["Total Profit"], name = "Total Profit"), row=1, col=1) fig.add\_trace(go.Bar(x=subcategory\_sales\_df["Subcategory"], y = subcategory\_sales\_df["Order Count"].sort\_values(ascending=False), name = "Total Orders"), row=2, col=1)  fig.update\_layout(height=1000, title\_text="Orders and Profit by Product Subcategory")  fig.show() |
| --- |

[](https://drive.google.com/file/d/1POtGpbdJ70se7OUYqjSFfRVQQ_eXqIc2/view?usp=drive_link)

We now have a better idea of the types of products that are bringing in the most profit. Next, we will find out the product brands bringing us the most money.

## **Total Profit Brought in by Different Brands**

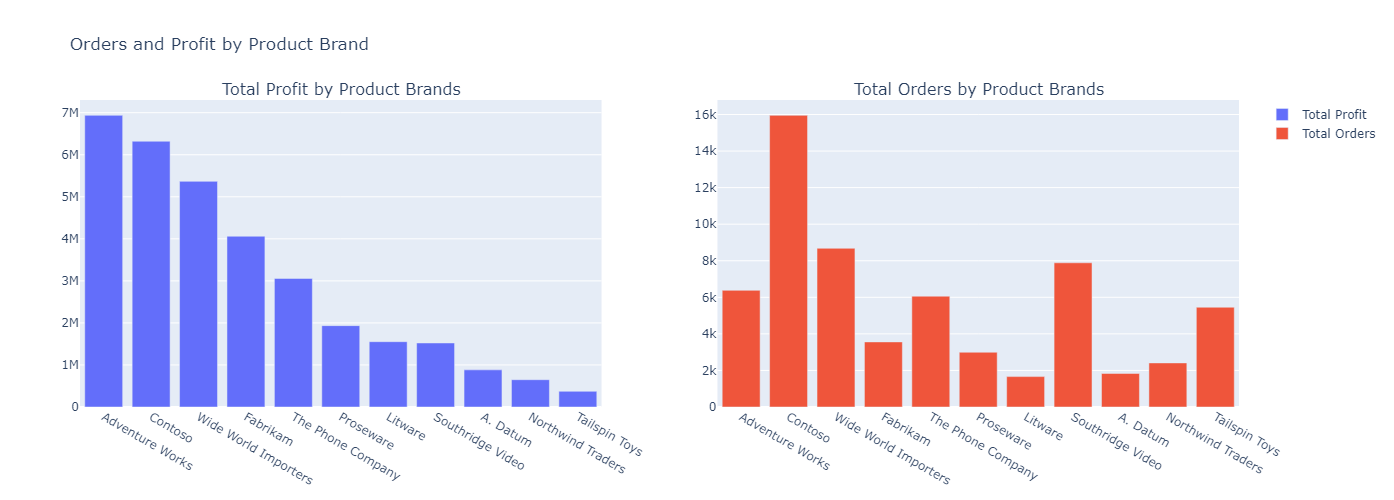
| SELECT   p."Brand",   COUNT(sales."Order Number") AS "Order Count",   SUM(p."Profit" \* sales."Quantity") AS "Total Profit" FROM   sales JOIN   products p ON sales."ProductKey" = p."ProductKey" GROUP BY   p."Brand" ORDER BY   "Total Profit" DESC; |
| --- |

The output is 11 rows of all brand names, the number of orders placed, and their total profit generated



We will use plotly subplots to create visualizations in Python to clearly display brand information.

| brand\_sales\_df = pd.read\_csv("C:/Users/HP/Downloads/Global+Electronics+Retailer/brand\_profit\_and\_orders.csv")  fig = make\_subplots(rows=1, cols=2, subplot\_titles=["Total Profit by Product Brands", "Total Orders by Product Brands"])   fig.add\_trace(go.Bar(x=brand\_sales\_df["Brand"], y = brand\_sales\_df["Total Profit"], name = "Total Profit"), row=1, col=1) fig.add\_trace(go.Bar(x=brand\_sales\_df["Brand"], y = brand\_sales\_df["Order Count"], name = "Total Orders"), row=1, col=2)  fig.update\_layout(height=500, title\_text="Orders and Profit by Product Brand")  fig.show() |
| --- |

[](https://drive.google.com/file/d/1JLp-6TKZcvPv7IweX9LeDZlMS0diIEZZ/view?usp=drive_link)

We have ordered specific products by number of orders and profit generated. We can notice that even with less than half the number of orders than Contoso, Adventure Works is our most profitable brand. The chart also tells us that Contoso is the brand our customers are most attracted to based on orders.

Alright! We have performed a decent analysis of our products table. Of course, we could go further to get more insights like:

* What categories of products do various brands produce?
* Grouping different product categories and subcategories by brands.
* Does the color of lamps determine the sales that were not gotten?
* If a particular color of product or Category makes a difference in sales.
* The categories and subcategories top product orders belong to.

This is a good point we have reached for this stage of the product table analysis. Now let’s get a better understanding of our customers.

| SELECT DISTINCT "Country" FROM customers; |
| --- |



| SELECT DISTINCT "Country" FROM stores; |
| --- |

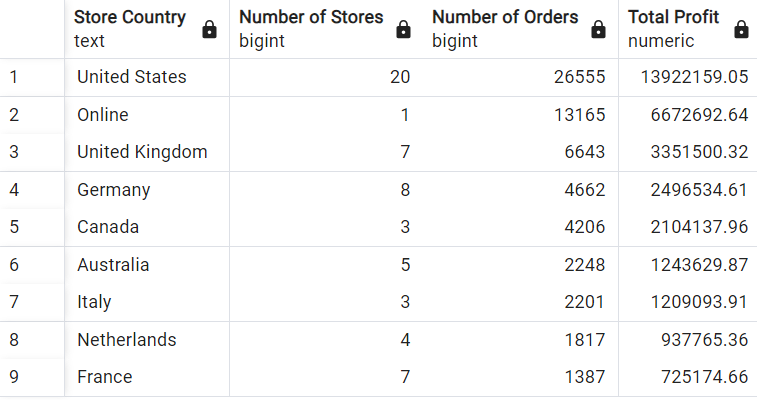


We don’t have stores in Canada but we have customers there, this most likely means all Canadian customers order online

## **Profit Generated by Stores**

| SELECT   st."Country" AS "Store Country",   COUNT(DISTINCT st."StoreKey") AS "Number of Stores",   COUNT(s."Order Number") AS "Number of Orders",   SUM(p."Profit" \* s."Quantity") AS "Total Profit" FROM   sales s JOIN   stores st ON s."StoreKey" = st."StoreKey" JOIN   products p ON s."ProductKey" = p."ProductKey" GROUP BY   st."Country" ORDER BY   "Total Profit" DESC; |
| --- |

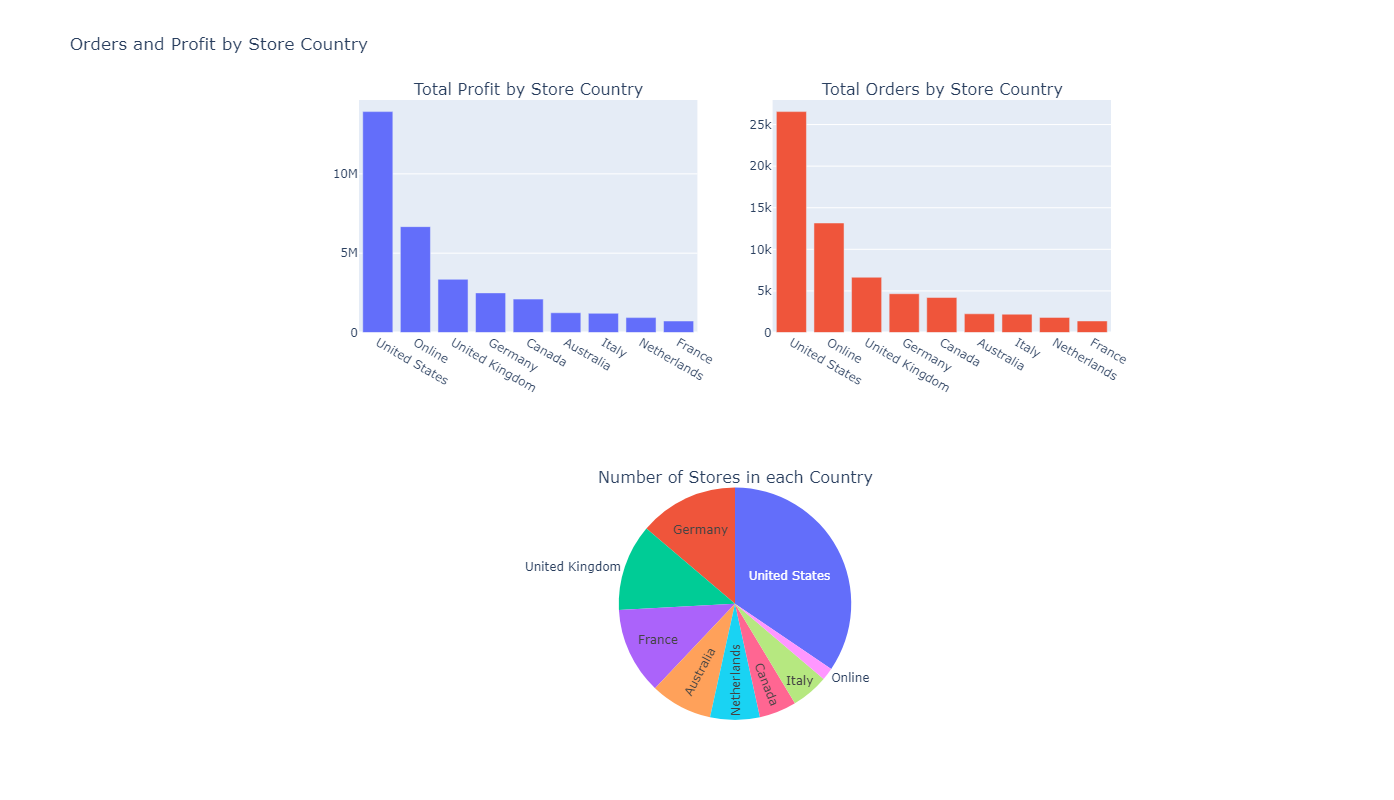
Let’s find out more about our stores’ performance in each country as well as online.



We can visualize this for a clearer picture using Plotly.

| stores\_sales\_df = pd.read\_csv("C:/Users/HP/Downloads/Global+Electronics+Retailer/stores\_with\_sales.csv")    fig = make\_subplots(  rows=2, cols=2,  subplot\_titles=[  "Total Profit by Store Country",  "Total Orders by Store Country",  "Profit Distribution by Store Country"  ],  specs=[[{'type': 'xy'}, {'type': 'xy'}], [{'colspan': 2, 'type': 'domain'}, None]]  )  fig.add\_trace(go.Bar(x=stores\_sales\_df["Store Country"], y=stores\_sales\_df["Total Profit"], name="Total Profit"), row=1, col=1) fig.add\_trace(go.Bar(x=stores\_sales\_df["Store Country"], y=stores\_sales\_df["Number of Orders"], name="Total Orders"), row=1, col=2)  fig.add\_trace(go.Pie(labels=stores\_sales\_df["Store Country"], values=stores\_sales\_df["Number of Stores"], name="Number of Stores in each Country"), row=2, col=1)  fig.update\_layout(  height=800,  title\_text="Orders and Profit by Store Country",  showlegend=False )  fig.show() |
| --- |

The output:

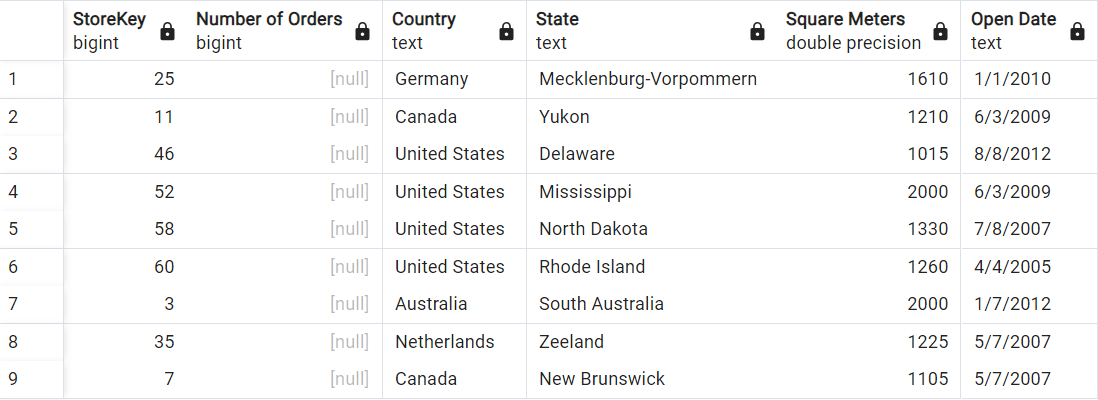
[](https://drive.google.com/file/d/1z_qjm23RL_ZzNWgV3-XSa88dIlcZq9DA/view?usp=drive_link)

The total number of orders is 62,884, which corresponds with the number from our sales table. The total number of stores we have is 67(from our store key count from earlier), however, if we add up the number of stores from our above output, the result is 58.

This means that some stores had no orders, we will get these stores by using the IS NULL method, similar to how we had previously done.

## **Stores with No Sales**

| SELECT   st."StoreKey",   "Order Number" AS "Number of Orders",   st."Country",   st."State",   st."Square Meters",   st."Open Date" FROM   stores st LEFT JOIN   sales s ON st."StoreKey" = s."StoreKey" WHERE   s."Order Number" IS NULL; |
| --- |

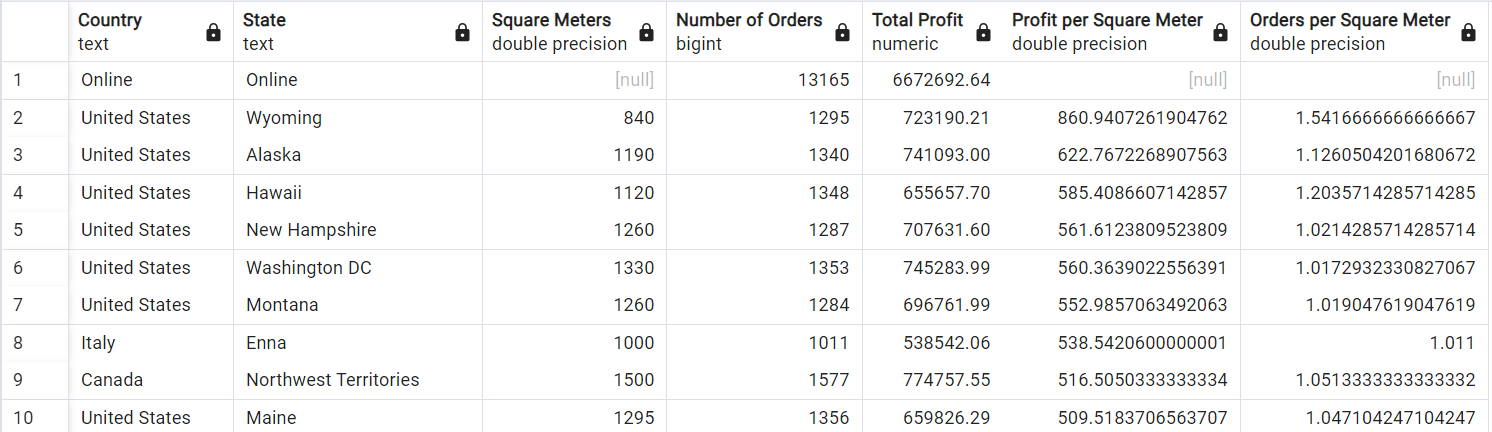


9 physical stores did not make any sales.

## **Profit per Square Meter**

We can run further analysis to find the total profit and orders we made per square meter in different states where our stores are located.

| SELECT   st."StoreKey",  st."Country",   st."State",  st."Square Meters",  COUNT(s."Order Number") AS "Number of Orders",  SUM(p."Profit" \* s."Quantity") AS "Total Profit",  (SUM(p."Profit" \* s."Quantity") / st."Square Meters") AS "Profit per Square Meter",  (COUNT(s."Order Number") / st."Square Meters") AS "Orders per Square Meter" FROM   sales s JOIN   stores st ON s."StoreKey" = st."StoreKey" JOIN   products p ON s."ProductKey" = p."ProductKey" GROUP BY   st."StoreKey",   st."Country",   st."State",   st."Square Meters" ORDER BY   "Profit per Square Meter" DESC; |
| --- |



The output gives us the details of all **58** stores that generated profit.

We will make use of pandas dataframe in python to select our physical stores and visualize different store information across different states; for this, we will be using plotly dash to create an interactive dashboard.

| profit\_per\_sq\_metres\_df = pd.read\_csv("C:/Users/HP/Downloads/Global+Electronics+Retailer/profit\_per\_square\_meters.csv") profit\_per\_sq\_metres\_df = profit\_per\_sq\_metres\_df.drop(0).reset\_index(drop=True) |
| --- |

We imported and removed the first row of the data frame, which was the information for our online store since we only wanted to find out more about our physical stores.

Let’s now bring in the dash library

| from dash import Dash, dcc, html, callback from dash.dependencies import Input, Output app = Dash(\_\_name\_\_) |
| --- |

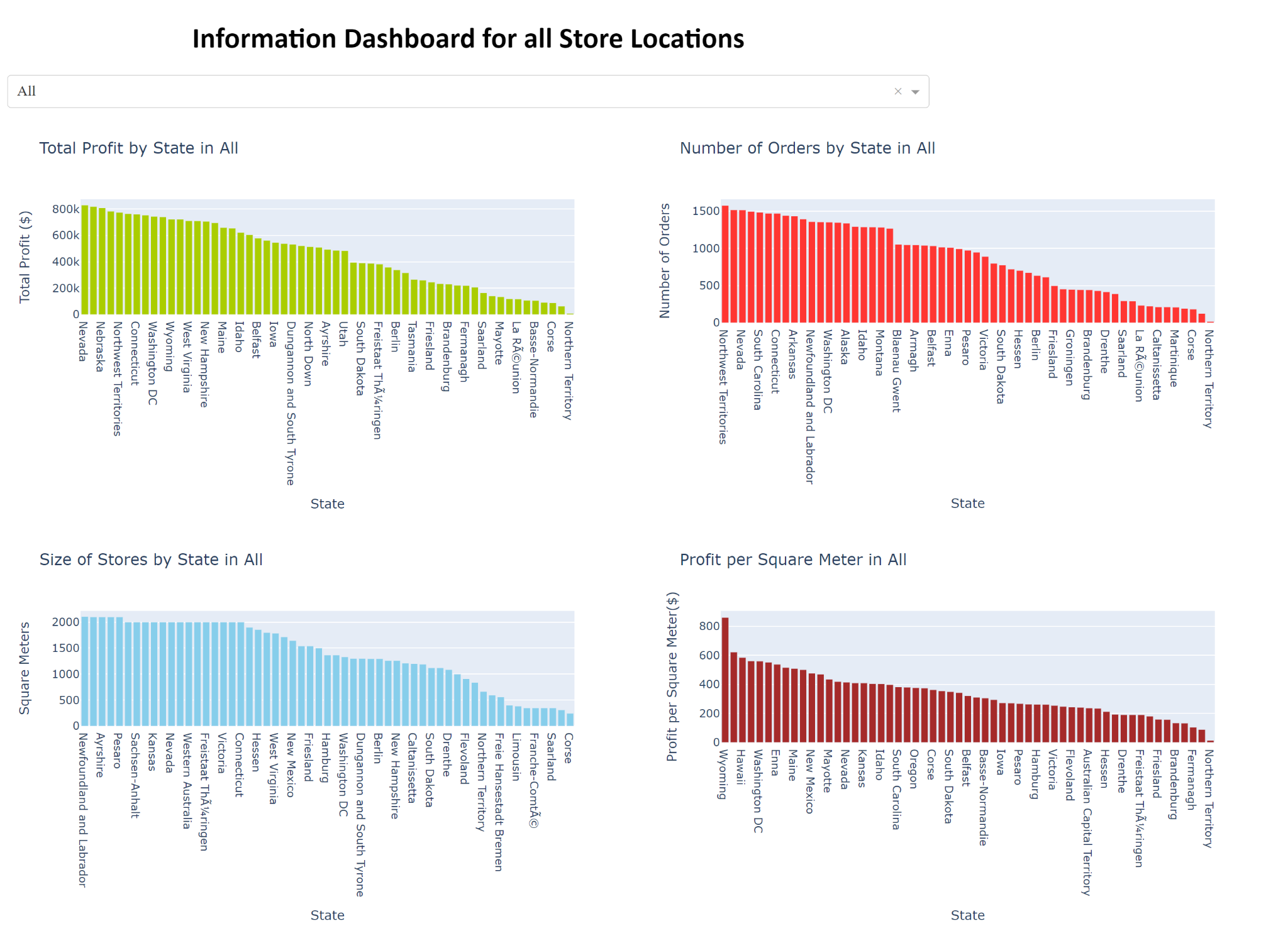
We will set up the layout of our dash web app, making use of dash dropdown to make it easy to navigate information about different states in different countries.

| app.layout = html.Div([ html.H1("Information Dashboard for all Store Locations", style={"text-align" : "center", "font-family" : "calibri"}),  html.Div([    dcc.Dropdown( id="country-dropdown",  options= [{'label': 'All', 'value': 'All'}] +  [{'label': country, 'value': country} for country in profit\_per\_sq\_metres\_df["Country"].unique()],  *# value=profit\_per\_sq\_metres\_df['Country'].unique()[1],*  value="All",  ), ]),  html.Div([   html.Div([  dcc.Graph(id='state-profit'),  dcc.Graph(id="store-size"),  ]),   html.Div([  dcc.Graph(id='state-orders'),  dcc.Graph(id="profit-per-sq-meter")  ])   ], id="graph-plots", style={"display" : "flex"})  ]) |
| --- |

Next, we will include our callback function to add interactivity and change the display depending on our dropdown selection.

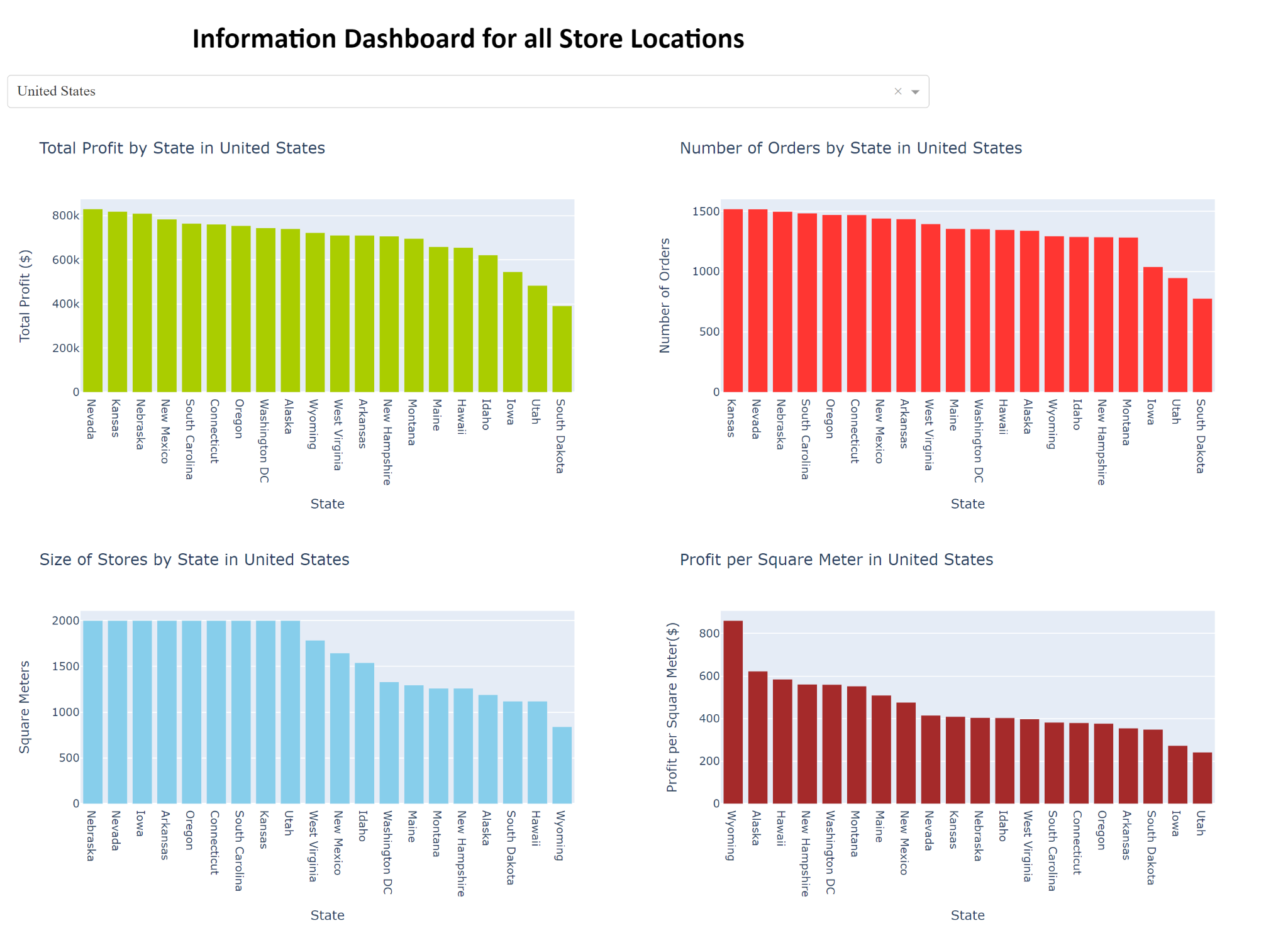
| @app.callback(  [Output(component\_id="state-profit", component\_property="figure"),  Output(component\_id="state-orders", component\_property="figure"),  Output(component\_id="store-size", component\_property="figure"),  Output(component\_id="profit-per-sq-meter", component\_property="figure")],  Input(component\_id="country-dropdown", component\_property="value") )  def country\_plots(selected\_country):   if selected\_country == "All":    filtered\_df = profit\_per\_sq\_metres\_df   else:  filtered\_df = profit\_per\_sq\_metres\_df[profit\_per\_sq\_metres\_df['Country'] == selected\_country]   profit\_fig = px.bar(  filtered\_df.sort\_values(by="Total Profit", ascending=False),  x='State',  y= 'Total Profit',  title=f'Total Profit by State in {selected\_country} States',  labels={'Total Profit': 'Total Profit ($)', 'State': 'State'},  color\_discrete\_sequence=['#AACD00'],  hover\_data=["Country"]    )   orders\_fig = px.bar(  filtered\_df.sort\_values(by="Number of Orders", ascending=False),  x="State",  y = "Number of Orders",  title=f'Number of Orders by State in {selected\_country} States',  labels={'Total Orders': 'Total Orders', 'State': 'State'},  color\_discrete\_sequence=['#FF3632'],  hover\_data=["Country"]    )   sq\_metres\_fig = px.bar(  filtered\_df.sort\_values(by="Square Meters", ascending=False),  x="State",  y = "Square Meters",  title=f'Size of Stores by State in {selected\_country} States',  labels={'Store Size': 'Store Size(m²)', 'State': 'State'},  color\_discrete\_sequence=['#87CEEB'],  hover\_data=["Country"]    )   profit\_per\_sq\_fig = px.bar(  filtered\_df.sort\_values(by="Profit per Square Meter", ascending=False),  x="State",  y = "Profit per Square Meter",  title=f'Profit per Square Meter in {selected\_country} States',  labels={'Profit per Square Meter': 'Profit per Square Meter($)', 'State': 'State'},  color\_discrete\_sequence=['brown'],  hover\_data=["Country"]    )    return profit\_fig, orders\_fig, sq\_metres\_fig, profit\_per\_sq\_fig  *#can be run locally* if \_\_name\_\_ == ("\_\_main\_\_"):  app.run\_server( host="127.0.0.1", port="8053") |
| --- |
|  |

The result of the code:

[](https://drive.google.com/file/d/10qgORcQOAz6zL8GmK7Co_AKbwGyXWdM4/view?usp=drive_link)

The above dashboard shows data for all states from all countries, we can also select and specify data of different countries with the dropdown

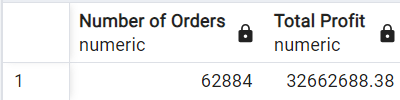
Data for all stores located in the United States.

[](https://drive.google.com/file/d/1jqhPLQEBP7YD3CsPOPDukTV6BfmBxa6o/view?usp=drive_link)

We have made use of just the bar chart for this, it seems simple but it’s one of the best ways to compare different categories.

I like to confirm that output is on the right track by summing up the number of orders and the total profit to see if it corresponds with our total profit and orders we have found previously.

| WITH summation AS (SELECT   EXTRACT(YEAR FROM "Order Date") AS "Year",  -- Get the text form of the month in order date  TO\_CHAR("Order Date", 'Month') AS "Month",  COUNT( "Order Number") AS "Number of Orders",  SUM(p."Profit" \* s."Quantity") AS "Total Profit" FROM   sales s JOIN   products p ON s."ProductKey" = p."ProductKey" GROUP BY   EXTRACT(YEAR FROM "Order Date"),   TO\_CHAR("Order Date", 'Month'),  EXTRACT(MONTH FROM "Order Date") ORDER BY   "Year" ASC,   EXTRACT(MONTH FROM "Order Date") ASC)  SELECT SUM("Number of Orders"), SUM("Total Profit")  FROM summation; |
| --- |

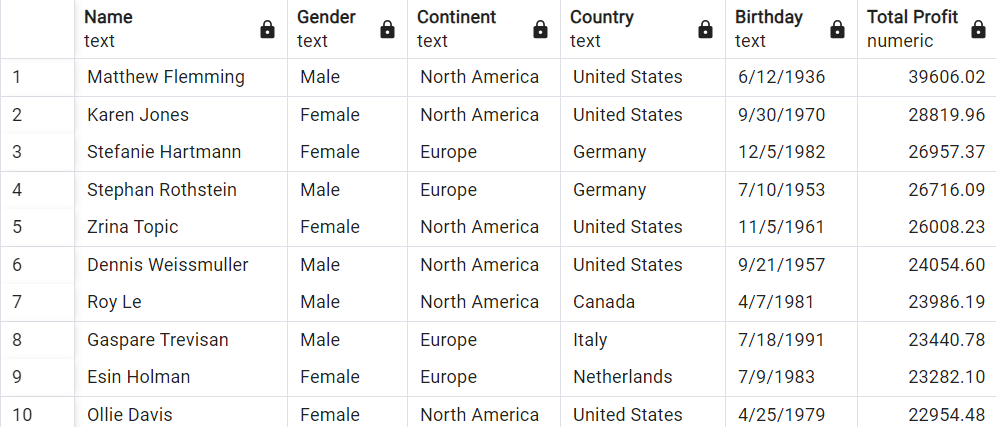


## **Profit Made from Customers**

We’ll gather information about our customers as well, this can be useful in grouping different types of customers.

| SELECT   c."Name",   c."Gender",   c."Continent",   c."Country",   c."Birthday",   SUM(p."Profit" \* s."Quantity") AS "Total Profit" FROM   sales s JOIN   customers c ON s."CustomerKey" = c."CustomerKey" JOIN   products p ON s."ProductKey" = p."ProductKey" GROUP BY   c."Name",   c."Gender",   c."Continent",   c."Country",   c."Birthday" ORDER BY   "Total Profit" DESC; |
| --- |

We will display our top 10 customers by how much they have spent on our store.



We now have a table of 11,887 rows containing details of different customers and the amount of money we have made in profit from their purchases.

Let’s find out more about the gender distribution of our customers, we’ll start by importing the saved CSV file.

| customer\_profit\_df = pd.read\_csv("C:/Users/HP/Downloads/Global+Electronics+Retailer/profit\_from\_customers.csv") customer\_profit\_df.head() |
| --- |



Knowing customer ages is a great way to know the age group of our buyers, we will create a new column “Customer Age”.

| customer\_profit\_df["Birthday"] = pd.to\_datetime(customer\_profit\_df["Birthday"], format="%m-%d-%Y")  current\_date = dt.now()  customer\_profit\_df["Customer Age"] = current\_date.year - pd.to\_datetime(customer\_profit\_df["Birthday"]).dt.year   *# moving 'Customer Age' column* column\_popped = customer\_profit\_df.pop("Customer Age") customer\_profit\_df.insert(5, column\_popped.name, column\_popped)   customer\_profit\_df.head() |
| --- |

This gives us our new table with customer ages included.



## **Customer Distribution by Gender**

We will use plotly subplots again to show the distribution of male and female customers across different countries.

We will first get the data frame data we would want to plot.

| gender\_count = customer\_profit\_df["Gender"].value\_counts().reset\_index() gender\_count.columns = ['Gender', 'Count']  profit\_by\_gender = customer\_profit\_df.groupby("Gender")["Total Profit"].sum().reset\_index()  country\_gender\_distribution = customer\_profit\_df.groupby(["Country", "Gender"]).size().reset\_index(name="Count")  country\_gender\_profit\_distribution = customer\_profit\_df.groupby(["Country", "Gender"])["Total Profit"].sum().reset\_index() |
| --- |

Let’s add subplots

| fig = make\_subplots(  rows=2, cols=2,  subplot\_titles=[  "Distribution of Genders in Customer Data",  "Total Profit by Customer Gender",  "Gender Distribution by Country",  "Profit by Gender and Country"  ],  specs=[[{'type': 'domain'}, {'type': 'domain'}], [{'type': 'xy'}, {'type': 'xy'}]]  ) |
| --- |

Now we can plot the figure traces that would be shown as subplots, for Plotly graph objects, we can’t make use of the barmode = "group" as you can in Plotly express(because we would want a grouped bar chart plot), so we will use multiple traces to serve as a second bar in the subplot; it will be clearer when we are done.

| # First plot - Pie chart for gender distribution fig.add\_trace(go.Pie(showlegend=False, labels=gender\_count["Gender"], values=gender\_count["Count"],  marker=dict(colors=['blue', '#FF1493'])), row=1, col=1)  # Second plot - Pie chart for profit by gender  fig.add\_trace(go.Pie(showlegend=False, labels=profit\_by\_gender["Gender"], values=profit\_by\_gender["Total Profit"],  marker=dict(colors=[ '#FF1493','blue'])), row=1, col=2)   # Third plot - Barplots for gender count in different countries fig.add\_trace(go.Bar(  x=country\_gender\_distribution[country\_gender\_distribution['Gender'] == 'Male']["Country"],  y=country\_gender\_distribution[country\_gender\_distribution['Gender'] == 'Male']["Count"],  name="Male", marker\_color='blue'), row=2, col=1)  fig.add\_trace(go.Bar(showlegend=False,  x=country\_gender\_distribution[country\_gender\_distribution['Gender'] == 'Female']["Country"],  y=country\_gender\_distribution[country\_gender\_distribution['Gender'] == 'Female']["Count"],  name="Female", marker\_color='#FF1493'), row=2, col=1)   # Fourth plot - Barplots for for total profit by gender in different countries fig.add\_trace(go.Bar( showlegend=False,  x=country\_gender\_profit\_distribution[country\_gender\_profit\_distribution['Gender'] == 'Male']["Country"],  y=country\_gender\_profit\_distribution[country\_gender\_profit\_distribution['Gender'] == 'Male']["Total Profit"],  name="Male", marker\_color='blue'), row=2, col=2)  fig.add\_trace(go.Bar(  x=country\_gender\_profit\_distribution[country\_gender\_profit\_distribution['Gender'] == 'Female']["Country"],  y=country\_gender\_profit\_distribution[country\_gender\_profit\_distribution['Gender'] == 'Female']["Total Profit"],  name="Female", marker\_color='#FF1493'), row=2, col=2)  fig.update\_layout(  height=800,  width = 1200 ,  title\_text="Customer Data Analysis by Gender and Country" )  fig.show() |
| --- |

Our results:

[](https://drive.google.com/file/d/1zG9AjnLaZmutHSXvy7LWY4lnFIZEFKHK/view?usp=drive_link)

There are slightly more males than females, and they generate more profit.

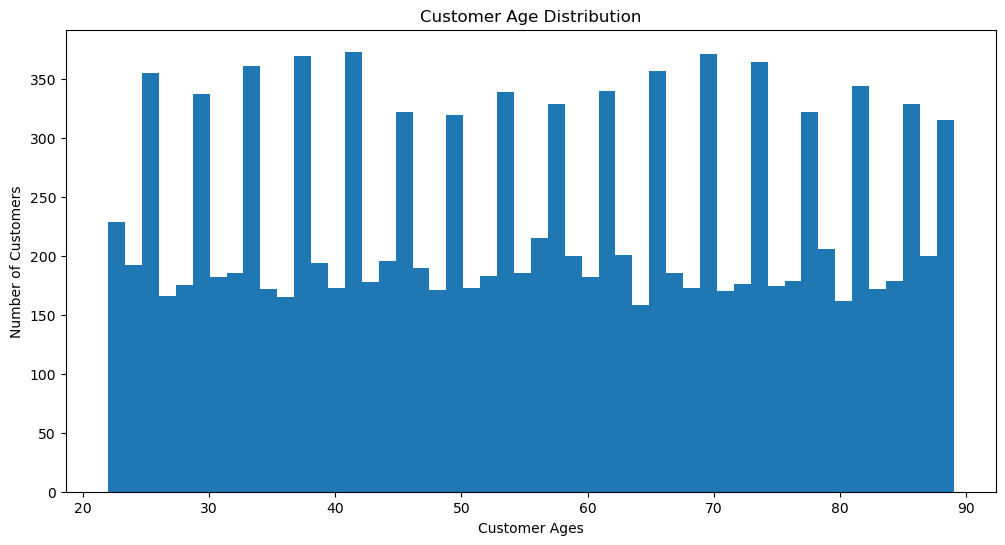
Most of our customers are from the United States, so increased marketing and spreading awareness in other countries would benefit our store’s profitability.

Total profit is highly dependent on the number of customers in each country.

## **Customer Age Distribution**

With the help of a histogram, we will be able to see the distribution of our customers across different age groups.

| plt.figure(figsize=(12, 6))  plt.title("Customer Age Distribution") plt.xlabel("Customer Ages") plt.ylabel("Number of Customers")  plt.hist(customer\_profit\_df["Customer Age"], bins=50) plt.show() |
| --- |



Our customers are spread fairly evenly across all age groups, with spikes at certain age brackets. We can attract customers from the mid-20s to late 80s.

## **Sales Information Over Time**

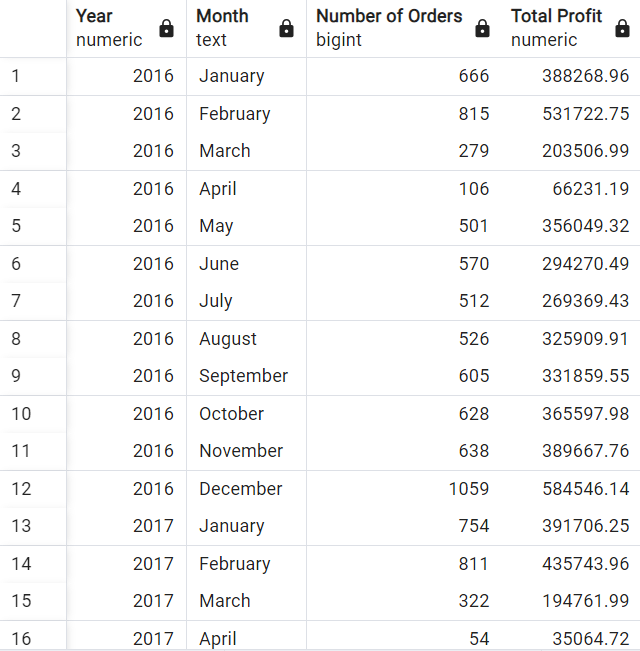
We want to find out information about our sales at different time intervals. We will need to format our order and delivery date to a format that we can work with.

| ALTER TABLE sales ALTER COLUMN "Order Date" TYPE DATE USING TO\_DATE("Order Date", 'MM-DD-YYYY');  ALTER TABLE sales ALTER COLUMN "Delivery Date" TYPE DATE USING TO\_DATE("Delivery Date", 'MM-DD-YYYY'); |
| --- |

We want to find out the number of orders and profit generated every month per year.

We can get our monthly performance for every year by extracting and grouping by the months and years from the date format. We will count the number of orders and get the sum of profit for each grouping.

| SELECT   EXTRACT(YEAR FROM "Order Date") AS "Year",  -- get the text form of the month in order date  TO\_CHAR("Order Date", 'Month') AS "Month",  COUNT( "Order Number") AS "Number of Orders",  SUM(p."Profit" \* s."Quantity") AS "Total Profit" FROM   sales s JOIN   products p ON s."ProductKey" = p."ProductKey" GROUP BY   EXTRACT(YEAR FROM "Order Date"),   TO\_CHAR("Order Date", 'Month'),  EXTRACT(MONTH FROM "Order Date") ORDER BY   "Year" ASC,   EXTRACT(MONTH FROM "Order Date") ASC; |
| --- |
|  |

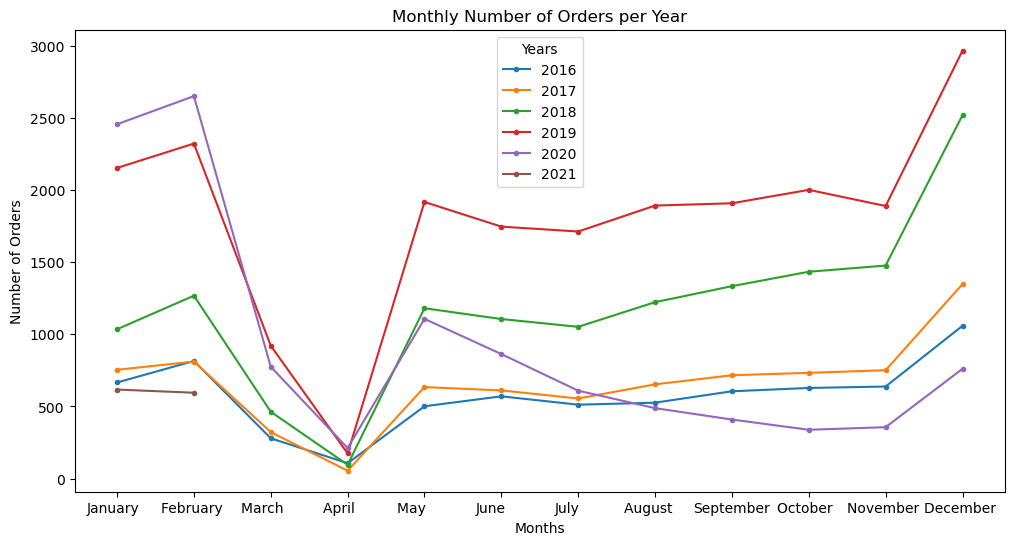


It gives us **62 rows** of different sales duration.

We can also visualize this using Python, we’ll be making use of matplotlib this time to have a clearer representation of our monthly profit and orders per year.

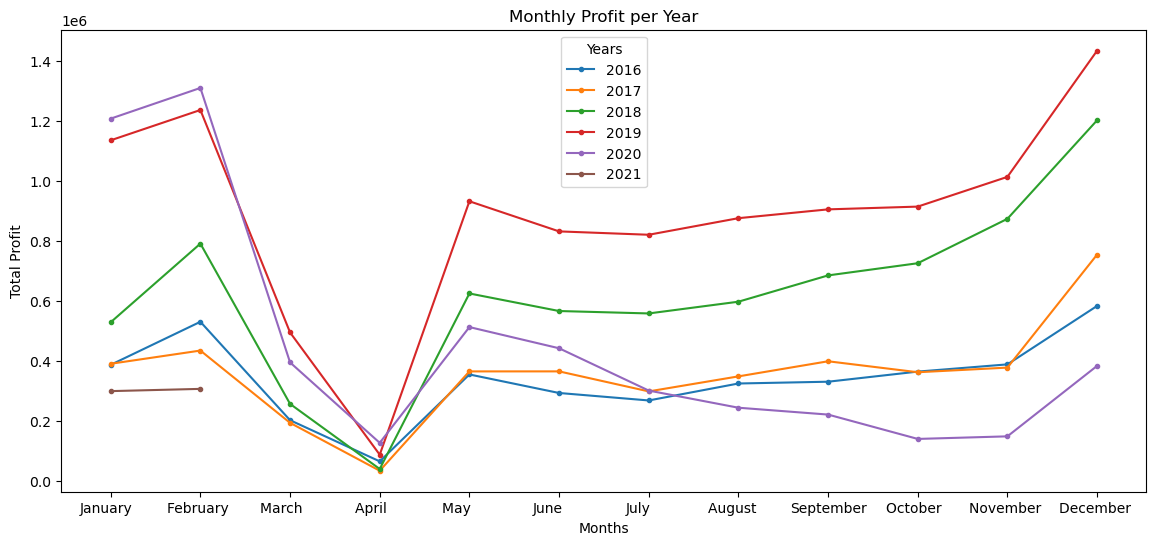
| plt.figure(figsize=(12, 6)) plt.title("Monthly Number of Orders per Year") plt.xlabel("Months") plt.ylabel("Number of Orders")  for year in sale\_years:   year\_trend = sales\_trend[sales\_trend["Year"] == year]   plt.plot(year\_trend["Month"],  year\_trend["Number of Orders"],  label = year,  marker = "o",  markersize = 3  )    plt.legend(title = "Years") plt.show() |
| --- |

And from the output plot, profit and orders are at an all-time low in the month of April for every year with sales in that month. April has always been a bad month for TechSphere.

[](https://drive.google.com/file/d/1YrkcvNitRdlbnU3wH6BLBdj_cCkuBvL0/view?usp=drive_link)

For our profit visualization, we just copy the same code from above and change it a bit.

| plt.figure(figsize=(14, 6)) plt.title("Monthly Profit per Year") plt.xlabel("Months") plt.ylabel("Total Profit")  for year in sale\_years:   year\_trend = sales\_trend[sales\_trend["Year"] == year]   plt.plot(year\_trend["Month"],  year\_trend["Total Profit"],  label = year,  marker = "o",  markersize = 3  )  plt.legend(title = "Years") plt.show() |
| --- |

[](https://drive.google.com/file/d/1rKpKjlL01QkLtUD7Dd06ybZGkDO-FAPD/view?usp=drive_link)

The delivery date column in our sales table has a lot of missing values. It would have been useful in finding average delivery time across different time periods. We don’t want to make guesses or fill it in with imaginary values that were never there; so we’ll just leave it blank and not include it in the analysis.

Who knows? Maybe the orders were not delivered or were returned, but we’ll leave that out for the sake of this project.