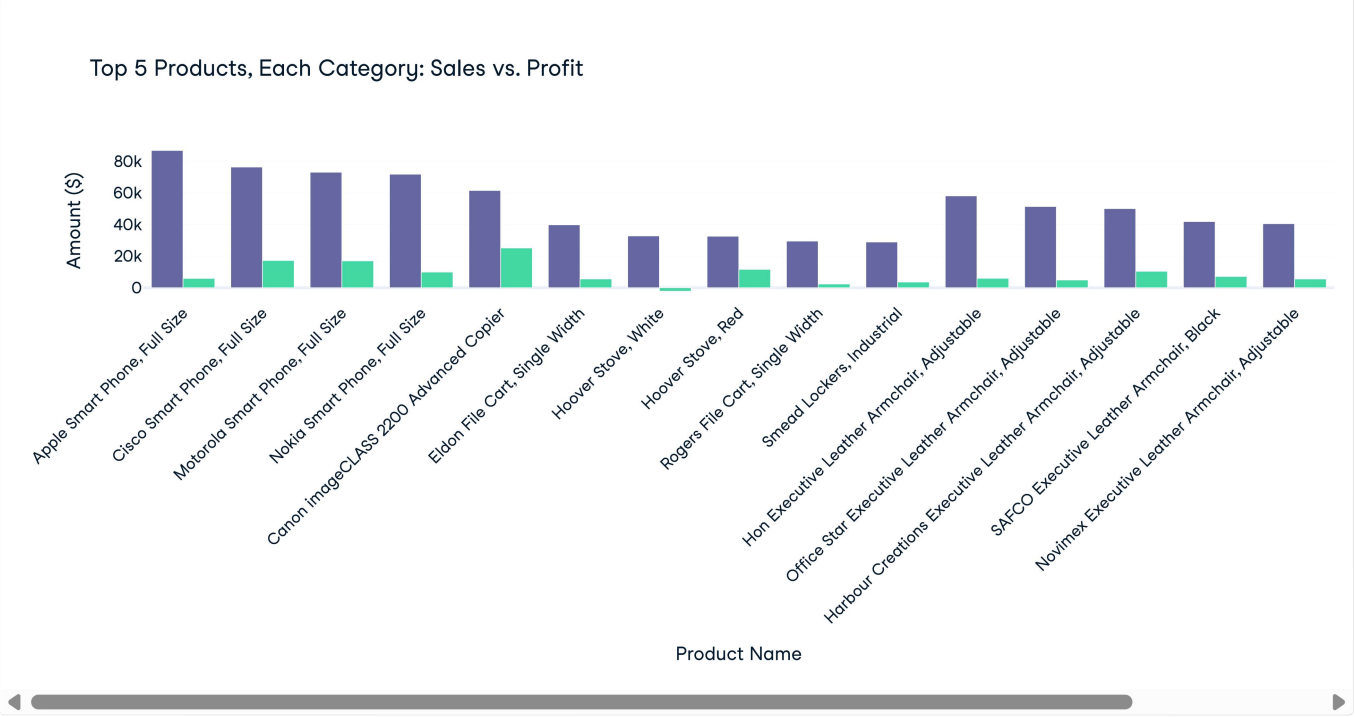
Projects Data DataFrame as top_five_products_each_category

```sql
-- Use a common table expression (CTE) to aggregate total sales and profit for each product.
WITH product_aggregates AS (
    SELECT
        p.category,                                 -- Retrieves the product category from the products table.
        p.product_name,                             -- Retrieves the product name.
        ROUND(SUM(o.sales)::numeric, 2) AS product_total_sales,   -- Sums the sales for each product, casts the result to
numeric, and rounds it to two decimal places.
        ROUND(SUM(o.profit)::numeric, 2) AS product_total_profit  -- Sums the profit for each product, casts the result to
numeric, and rounds it to two decimal places.
    FROM orders o
    JOIN products p
        ON o.product_id = p.product_id              -- Joins orders and products tables based on product_id.
    GROUP BY p.category, p.product_name            -- Groups the data by category and product name to perform the aggregations.
),

-- Use another CTE to rank products within each category by total sales in descending order.
ranked_products AS (
    SELECT
        category,
        product_name,
        product_total_sales,
        product_total_profit,
        RANK() OVER (PARTITION BY category ORDER BY product_total_sales DESC) AS product_rank
        -- RANK() calculates the rank of each product within its category, ordering by product_total_sales in descending order.
    FROM product_aggregates
)

-- Select the top 5 products per category.
SELECT
    category,
    product_name,
    product_total_sales,
    product_total_profit,
    product_rank
FROM ranked_products
WHERE product_rank <= 5                    -- Filters the results to include only the top 5 ranked products per category.
ORDER BY category DESC, product_total_sales DESC;
-- Orders the final output by category in descending order and then by product_total_sales (highest first within each
category).
```

	category	product_name	product_total_sa...	product_total_profit	prod...
0	Technology	Apple Smart Phone, Full Size	86935.78	5921.58	
1	Technology	Cisco Smart Phone, Full Size	76441.53	17238.52	
2	Technology	Motorola Smart Phone, Full Size	73156.3	17027.11	
3	Technology	Nokia Smart Phone, Full Size	71904.56	9938.2	
4	Technology	Canon imageCLASS 2200 Advanced Copier	61599.82	25199.93	
5	Office Supplies	Eldon File Cart, Single Width	39873.23	5571.26	
6	Office Supplies	Hoover Stove, White	32842.6	-2180.63	
7	Office Supplies	Hoover Stove, Red	32644.13	11651.68	
8	Office Supplies	Rogers File Cart, Single Width	29558.82	2368.82	
9	Office Supplies	Smead Lockers, Industrial	28991.66	3630.44	
10	Furniture	Hon Executive Leather Armchair, Adjustable	58193.48	5997.25	
11	Furniture	Office Star Executive Leather Armchair, Adju...	51449.8	4925.8	
12	Furniture	Harbour Creations Executive Leather Armch...	50121.52	10427.33	
13	Furniture	SAFCO Executive Leather Armchair, Black	41923.53	7154.28	
14	Furniture	Novimex Executive Leather Armchair, Adjust...	40585.13	5562.35	

Rows: 15 Expand Table

```python
import plotly.express as px
fig = px.bar(
    top_five_products_each_category,
    x="product_name",
    y=["product_total_sales", "product_total_profit"],
    barmode='group',
    title="Top 5 Products, Each Category: Sales vs. Profit",
    labels={
        "product_name": "Product Name",
        "value": "Amount ($)",
        "variable": "Metric"
    },
    height=500
)

fig.update_layout(xaxis_tickangle=-45)
fig.show()
```



Projects Data DataFrame as i

```sql
-- impute_missing_values
-- First, compute the average unit price for each product_id, discount, market, and region.
-- Unit price is defined as sales divided by quantity (from rows where quantity is known and > 0).
WITH unit_prices AS (
    SELECT
        product_id,                             -- Product identifier
        discount,                               -- Discount applied
        market,                                 -- Market identifier
        region,                                 -- Region of the customer/salesperson
        AVG(sales / quantity) AS unit_price       -- Average unit price computed from known quantities
    FROM orders
    WHERE quantity IS NOT NULL AND quantity > 0  -- Use only orders with valid quantity
    GROUP BY product_id, discount, market, region  -- Group by product and pricing factors
)

-- Next, join the orders with missing quantity to the computed unit_prices.
SELECT
    o.product_id,                                 -- Product identifier from orders table
    o.discount,                                    -- Discount applied in the order table
    o.market,                                     -- Market identifier in the orders table
    o.region,                                      -- Region in the orders table
    o.sales,                                    -- Total sales amount from the orders table
    o.quantity,                       -- Original (missing) quantity value from orders table
    -- Calculate the missing quantity by dividing sales by the unit price and round to 0 decimals.
    ROUND(o.sales / up.unit_price) AS calculated_quantity
FROM orders o
JOIN unit_prices up
    ON o.product_id = up.product_id                -- Join on product_id
   AND o.discount = up.discount                    -- Join on discount
   AND o.market = up.market                        -- Join on market
   AND o.region = up.region                        -- Join on region
WHERE o.quantity IS NULL;                     -- Only process orders with missing quantity
```

	product_id						calculated_quan...
0	TEC-STA-10003330	0	Africa	Africa	506.64		2
1	FUR-ADV-10000571	0	EMEA	EMEA	438.96		4
2	FUR-BO-10001337	0.15	US	West	308.499		3
3	TEC-STA-10004542	0	Africa	Africa	160.32		4
4	FUR-ADV-10004395	0	EMEA	EMEA	84.12		2

Rows: 5 Expand Table

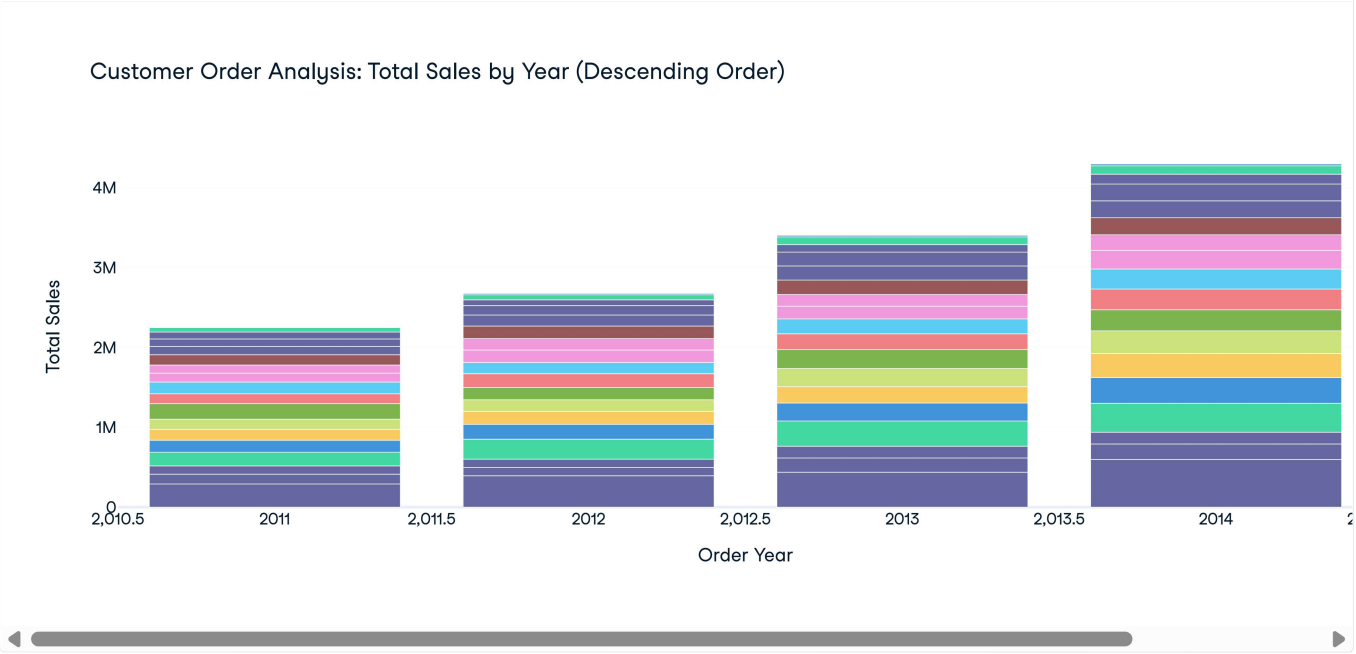
Projects Data DataFrame as C

```sql
-- Select key metrics to analyze customer purchase behavior and sales trends.
SELECT
    EXTRACT(YEAR FROM TO_DATE(order_date, 'YYYY-MM-DD')) AS order_year,
        -- Extracts the year from the order_date (assumes order_date is in 'YYYY-MM-DD' format).
    region,
        -- Returns the region from the orders table.
    market,
        -- Returns the market from the orders table.
    COUNT(DISTINCT order_id) AS num_orders,
        -- Counts the number of distinct orders for that year, region, and market.
    COUNT(*) AS total_line_items,
        -- Counts the total number of line items (each row represents an order line).
    SUM(sales) AS total_sales,
        -- Sums up the total sales amount for those orders.
    ROUND(AVG(sales)::numeric, 2) AS avg_sales_per_order,
        -- Calculates the average sales per order and rounds it to two decimal places.
    SUM(quantity) AS total_quantity
        -- Sums up the total quantity sold for those orders.
FROM orders

GROUP BY order_year, region, market
        -- Groups the data by extracted order year, region, and market.
ORDER BY order_year DESC, total_sales DESC;
        -- Orders the results by year, then region, then market in ascending order.
```

	o…	region		n…	total_line_it…	total_sales	avg_sales_per_o…	t
0	2014	Central	EU	843	2008	597898.9454999991	297.76	
18	2013	Central	EU	698	1535	438140.1570000001	285.43	
36	2012	Central	EU	581	1314	393744.6150000002	299.65	
1	2014	Oceania	APAC	625	1193	362429.8859999998	303.8	
2	2014	Southeast Asia	APAC	534	1112	323068.2122999993	290.53	
19	2013	Oceania	APAC	442	957	316270.4940000001	330.48	
3	2014	EMEA	EMEA	888	1811	301685.9430000006	166.59	
54	2011	Central	EU	454	965	290768.9009999997	301.31	
4	2014	Africa	Africa	813	1624	283036.4399999998	174.28	
5	2014	North Asia	APAC	377	754	264560.3310000004	350.88	
6	2014	Central Asia	APAC	349	737	259140.2579999999	351.62	
7	2014	West	US	541	1099	250632.5255	228.06	
37	2012	Oceania	APAC	384	792	250065.672	315.74	
20	2013	North Asia	APAC	312	629	236865.4560000001	376.57	
8	2014	North	EU	359	814	233684.073	287.08	
21	2013	Africa	Africa	608	1294	229068.7920000002	177.02	

Rows: 72 Expand Table

```python
# Sort the Customer_Order dataframe by order_year (descending) and total_sales (descending)
Customer_Order_sorted = Customer_Order.sort_values(by=['order_year', 'total_sales'], ascending=[False, False])
# Now the data is ordered so that the most recent years with the highest sales come first.

# Create a bar chart with Plotly Express:
# - x-axis: order_year
# - y-axis: total_sales
# - color: region (to add another dimension to the visualization)
# - The title describes the analysis.
fig = px.bar(
    Customer_Order_sorted,
    x='order_year',
    y='total_sales',
    color='region',
    title='Customer Order Analysis: Total Sales by Year (Descending Order)',
    labels={'order_year': 'Order Year', 'total_sales': 'Total Sales'}
)

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



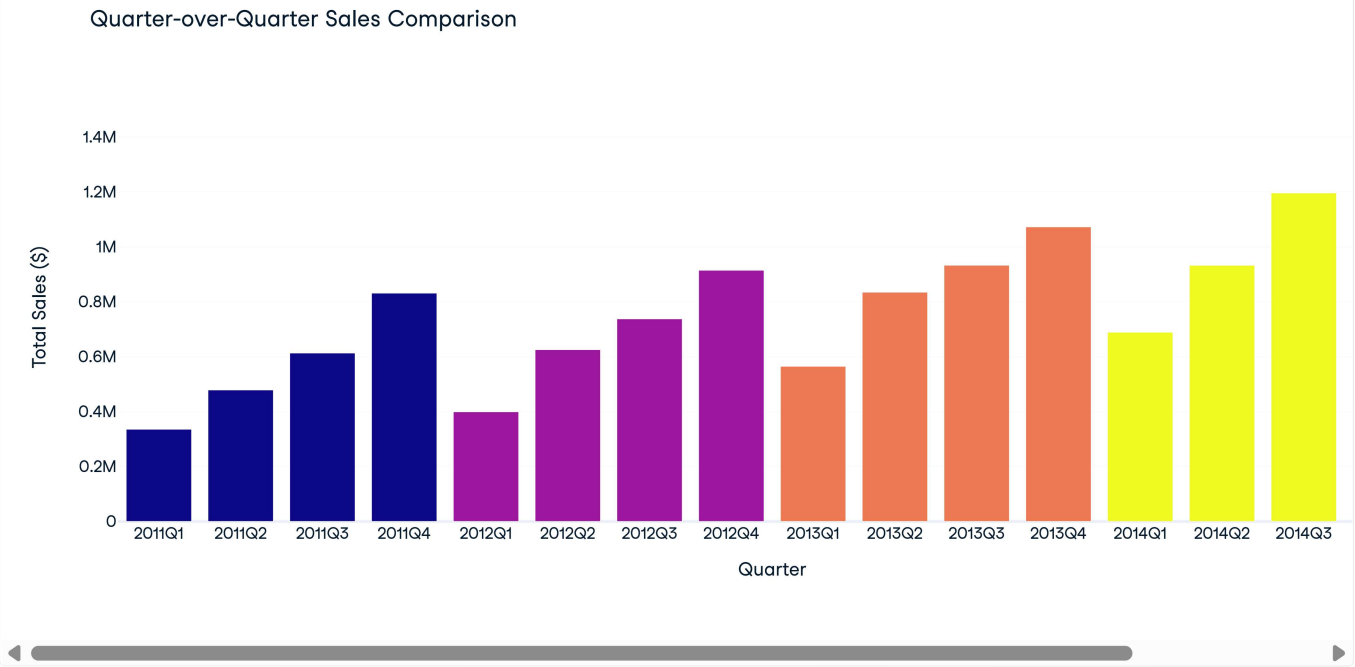
Projects Data DataFrame as

```sql
-- First, aggregate order data by month. We assume order_date is stored as TEXT in 'YYYY-MM-DD' format.
WITH monthly_aggregates AS (
    SELECT
        -- Truncate the order_date to the first day of the month to group by month
        DATE_TRUNC('month', TO_DATE(order_date, 'YYYY-MM-DD'))::date AS order_month,
        -- Sum total sales for each month
        SUM(sales) AS total_sales,
        -- Sum total profit for each month
        SUM(profit) AS total_profit,
        -- Count the number of distinct orders (as a proxy for order volume)
        COUNT(DISTINCT order_id) AS order_volume
    FROM orders
    GROUP BY order_month
)

-- Now, use window functions on the monthly aggregates to calculate moving averages and growth rates
SELECT
    order_month,                            -- The month of the orders
    total_sales,                            -- Total sales for that month
    total_profit,                           -- Total profit for that month
    order_volume,                           -- Number of orders in that month
    -- Calculate a 3-month moving average of total sales (current month and two preceding months)
    ROUND(AVG(total_sales) OVER (
        ORDER BY order_month
        ROWS BETWEEN 2 PRECEDING AND CURRENT ROW
    )::numeric, 2) AS moving_avg_sales,
    -- Use LAG() to get the total sales from the previous month
    LAG(total_sales) OVER (ORDER BY order_month) AS previous_month_sales,
    -- Calculate the percentage growth in sales from the previous month, rounded to 2 decimals.
    ROUND(
        CASE
            WHEN LAG(total_sales) OVER (ORDER BY order_month) IS NULL THEN NULL
            WHEN LAG(total_sales) OVER (ORDER BY order_month) = 0 THEN NULL
            ELSE ((total_sales - LAG(total_sales) OVER (ORDER BY order_month)) * 100.0
                  / LAG(total_sales) OVER (ORDER BY order_month))
        END::numeric,
        2
    ) AS sales_growth_percent
FROM monthly_aggregates
ORDER BY order_month DESC, total_sales DESC;
```

	order_month	total_sales	total_profit	orde...	moving_avg...	previous_month_
3	2014-09-01T00:00:00.000	481157.2437000002	67979.4510999999	1017	398827.62	45661
1	2014-11-01T00:00:00.000	555279.0270000007	62856.5879	1077	486400.97	42276
2	2014-10-01T00:00:00.000	422766.6291599997	58209.83476	810	453514.61	48115
4	2014-08-01T00:00:00.000	456619.9423600005	53542.89496	843	372379.9	25870
12	2013-12-01T00:00:00.000	405454.3780199995	50202.8711199999	825	357616.79	37398
13	2013-11-01T00:00:00.000	373989.3601000009	48062.9967000001	789	348005.08	29340
0	2014-12-01T00:00:00.000	503143.6934799998	46916.52068	1093	493729.78	55527
18	2013-06-01T00:00:00.000	396519.6118999995	45478.4134	724	278279.83	26049
6	2014-06-01T00:00:00.000	401814.0630999999	43778.6028	899	310995.66	28840
28	2012-08-01T00:00:00.000	303142.9423800004	43573.87858	522	234851.81	
14	2013-10-01T00:00:00.000	293406.6428800002	42433.22258	580	332171.56	37661
36	2011-12-01T00:00:00.000	333925.7346	40647.984	620	277164.51	29849
15	2013-09-01T00:00:00.000	376619.2456800007	38905.66778	836	311012.33	32648
9	2014-03-01T00:00:00.000	263100.7726200007	37357.26052	530	229735.56	18483
39	2011-09-01T00:00:00.000	290214.4553400001	35776.88394	527	204435.46	20758
30	2012-06-01T00:00:00.000	256175.6984200001	34407.15362	571	208530.95	20836

Rows: 48 Expand Table

```python
import plotly.express as px
# Extract year and quarter for grouping
order_month_sales['year'] = order_month_sales['order_month'].dt.year
order_month_sales['quarter'] = order_month_sales['order_month'].dt.to_period('Q').astype(str)

# Group by year and quarter to get total sales, profit, and order volume
quarterly_summary = order_month_sales.groupby(['year', 'quarter'], as_index=False).agg({
    'total_sales': 'sum',
    'total_profit': 'sum',
    'order_volume': 'sum'
})

# Create a bar chart to show quarter-over-quarter sales
fig = px.bar(
    quarterly_summary,
    x='quarter',
    y='total_sales',
    color='year',
    title='Quarter-over-Quarter Sales Comparison',
    labels={'total_sales': 'Total Sales ($)', 'quarter': 'Quarter'},
    barmode='group',
    height=500
)

fig.show()
```



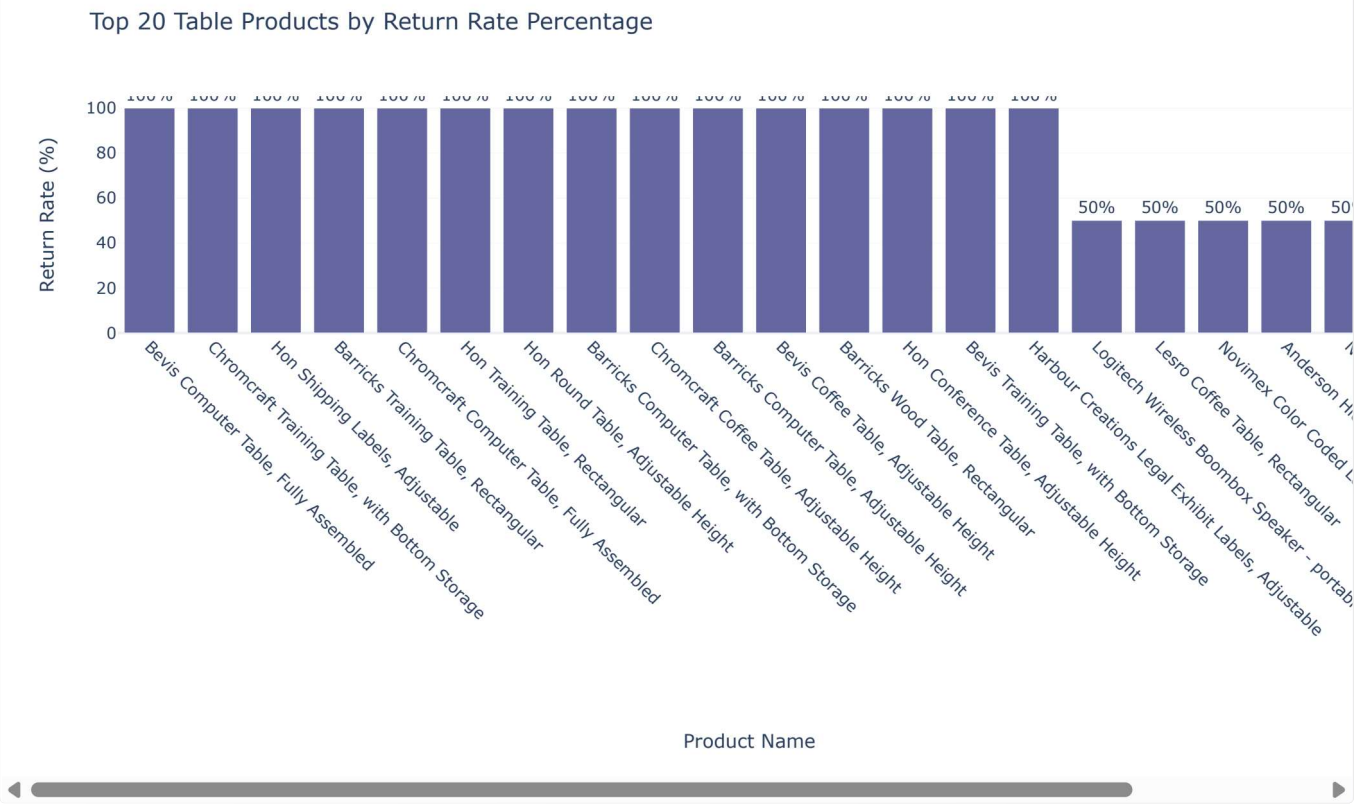
Projects Data DataFrame as

```sql
-- First, aggregate overall order data per product from the orders table.
WITH product_orders AS (
    SELECT
        o.product_id,                           -- Product identifier from orders table
        COUNT(*) AS total_orders,               -- Total orders for this product
        SUM(o.sales) AS total_sales,            -- Total sales for this product
        SUM(o.profit) AS total_profit           -- Total profit for this product
    FROM orders o
    GROUP BY o.product_id                       -- Group by product_id to aggregate per product
),

-- Next, aggregate returned orders data by joining orders with returned_orders.
product_returns AS (
    SELECT
        o.product_id,                           -- Product identifier
        COUNT(*) AS returned_orders,            -- Number of returned orders for this product
        SUM(o.sales) AS returned_sales,         -- Total sales for returned orders
        SUM(o.profit) AS returned_profit        -- Total profit for returned orders
    FROM orders o
    INNER JOIN returned_orders r                -- Join orders with returned_orders using order_id
        ON o.order_id = r.order_id
    GROUP BY o.product_id                       -- Group by product_id to aggregate per product
)

-- Finally, join the aggregated order data with the products table to get product details.
SELECT
    p.product_id,                               -- Product identifier from products table
    p.product_name,                             -- Product name from products table
    po.total_orders,                            -- Total number of orders for the product
    pr.returned_orders,                         -- Total number of returned orders for the product
    ROUND((pr.returned_orders::numeric / po.total_orders) * 100, 2) AS return_rate_percentage,
                                                -- Calculate and round return rate as percentage
    po.total_sales,                             -- Total sales for the product
    pr.returned_sales,                          -- Sales from returned orders
    po.total_profit,                            -- Total profit for the product
    pr.returned_profit                          -- Profit lost from returned orders
FROM products p
JOIN product_orders po ON p.product_id = po.product_id    -- Join aggregated orders to products
JOIN product_returns pr ON p.product_id = pr.product_id   -- Join aggregated returns to products
ORDER BY returned_orders DESC, returned_profit DESC;                    -- Order by return rate (highest first)
```

	product_id	product_name	total...	returned_...	return_rate_percentage
62	OFF-EN-10002613	Ames Clasp Envelope, Security-Tint	3	3	100
223	TEC-PH-10002119	Samsung Headset, Full Size	2	2	100
278	OFF-LA-10003077	Avery 500	2	2	100
282	OFF-SU-10000153	Acme Shears, High Speed	2	2	100
322	OFF-FA-10000146	Stockwell Rubber Bands, Assorted Sizes	2	2	100
351	OFF-PA-10003296	Xerox Message Books, Recycled	2	2	100
492	OFF-AP-10002321	Hoover Microwave, Silver	1	1	100
493	FUR-TA-10000022	Hon Conference Table, Adjustable Height	1	1	100
501	OFF-AP-10000003	KitchenAid Refrigerator, White	1	1	100
518	TEC-AC-10004018	SanDisk Memory Card, USB	1	1	100
551	FUR-TA-10000945	Bevis Coffee Table, Adjustable Height	1	1	100
557	TEC-CO-10003406	HP Fax and Copier, Laser	1	1	100
568	TEC-CO-10002696	Hewlett Personal Copier, Color	1	1	100
594	FUR-TA-10004810	Barricks Computer Table, Adjustable Height	1	1	100
638	FUR-CH-10000042	Harbour Creations Swivel Stool, Black	1	1	100
646	FUR-TA-10000670	Chromcraft Coffee Table, Adjustable Height	1	1	100

Rows: 2,484 Expand Table

```python
# Step 1: Filter for products that have 'Table' in the name (case-insensitive)
table_products = returns_df[
    returns_df["product_name"].str.contains("Table", case=False, na=False)
]

# Step 2: Sort by return rate percentage and select the top 20
top_20_tables = table_products.sort_values(by="return_rate_percentage", ascending=False).head(20)

# Step 3: Create the bar chart
fig = px.bar(
    top_20_tables,
    x="product_name",
    y="return_rate_percentage",
    title="Top 20 Table Products by Return Rate Percentage",
    labels={
        "product_name": "Product Name",
        "return_rate_percentage": "Return Rate (%)"
    },
    text="return_rate_percentage",
    height=600
)

# Step 4: Update layout and show the plot
fig.update_traces(texttemplate="%{text}%", textposition="outside")
fig.update_layout(
    xaxis_tickangle=45,
    template="plotly_white"
)

fig.show()
```



Projects Data DataFrame as

```sql
-- Step 1: Aggregate sales and profit by category and sub-category
WITH category_profitability AS (
    SELECT
        p.category,                                    -- Product category (e.g., Technology, Furniture)
        p.sub_category,                                -- Product sub-category (e.g., Chairs, Phones)
        SUM(o.sales) AS total_sales,                   -- Total sales amount for the sub-category
        SUM(o.profit) AS total_profit,                 -- Total profit earned from that sub-category
        -- Calculate profit margin: profit divided by sales, cast to numeric, rounded to 2 decimal places
        ROUND((SUM(o.profit) / NULLIF(SUM(o.sales), 0))::numeric, 2) AS profit_margin
    FROM orders o
    JOIN products p ON o.product_id = p.product_id     -- Join orders with product info to access category data
    GROUP BY p.category, p.sub_category                -- Group the data by category and sub-category
),

-- Step 2: Rank sub-categories by profit margin within each category
ranked_profitability AS (
    SELECT
        category,
        sub_category,
        total_sales,
        total_profit,
        profit_margin,
        -- Assign rank within each category based on highest to lowest profit margin
        RANK() OVER (
            PARTITION BY category
            ORDER BY profit_margin DESC
        ) AS margin_rank
    FROM category_profitability
)

-- Step 3: Final result selection and ordering
SELECT
    category,                       -- Product category
    sub_category,                   -- Product sub-category
    total_sales,                    -- Total sales in dollars
    total_profit,                   -- Total profit in dollars
    profit_margin,                  -- Profit as a percentage of sales
    margin_rank                     -- Rank within the category by profit margin
FROM ranked_profitability
ORDER BY category ASC, profit_margin DESC;  -- Order first by category, then by highest profit margin
```

	category	sub_...	total_sales	total_profit	profit_...	ma...
3	Furniture	Tables	757041.9243999979	-64083.3887	-0.08	4
9	Office Supplies	Fasteners	83242.3159	11525.4241	0.14	6
5	Office Supplies	Labels	73384.3739999997	15008.8559999999	0.2	2
12	Office Supplies	Supplies	243074.2205999997	22583.2631	0.09	9
6	Office Supplies	Envelopes	170904.3016000002	29601.1163	0.17	3
0	Furniture	Furnishings	385578.2559000014	46967.4254999999	0.12	1
8	Office Supplies	Art	372091.9659000034	57953.9108999998	0.16	4
16	Technology	Machines	779060.0671000002	58867.8729999999	0.08	4
4	Office Supplies	Paper	244291.7193999999	59207.6827	0.24	1
7	Office Supplies	Binders	461931.1616999998	72451.5020000001	0.16	4
11	Office Supplies	Storage	1127085.8613999898	108461.4898	0.1	8
14	Technology	Accessories	749237.0184999978	129626.3061999995	0.17	1
2	Furniture	Chairs	1501681.7641999938	140396.2675	0.09	3
10	Office Supplies	Appliances	1011064.3049999982	141680.5893999997	0.14	6
1	Furniture	Bookcases	1466572.2417999967	161924.4195000004	0.11	2
15	Technology	Phones	1706824.1391999812	216717.0058000002	0.13	3

Rows: 17 Expand Table

Projects Data DataFrame as

```sql
SELECT *
FROM people;
```

	region	person
0	Central	Anna Andreadi
1	South	Chuck Magee
2	East	Kelly Williams
3	West	Matt Collister
4	Africa	Deborah Brumfield
5	EMEA	Larry Hughes
6	Canada	Nicole Hansen
7	Caribbean	Giulietta Dortch
8	Central Asia	Nora Preis
9	North	Jack Lebron
10	North Asia	Shirley Daniels
11	Oceania	Anthony Jacobs
12	Southeast Asia	Alejandro Ballentine

Rows: 13 Expand Table

Projects Data DataFrame as

```sql
-- Step 1: Join orders with the people table using the region column
SELECT
    o.region,                                   -- Region of the customer
    o.market,                                   -- Market where the order was placed
    p.person AS salesperson,                    -- Salesperson responsible, from the people table
    SUM(o.sales) AS total_sales,                -- Total sales amount in this region-market-person group
    SUM(o.profit) AS total_profit,              -- Total profit earned
    -- Calculate average profit margin (profit / sales), rounded to 2 decimals
    ROUND((SUM(o.profit) / NULLIF(SUM(o.sales), 0))::numeric, 2) AS profit_margin
FROM orders o
JOIN people p ON o.region = p.region            -- Join on region (associates salesperson with order region)
GROUP BY o.region, o.market, p.person           -- Group by region, market, and salesperson
ORDER BY total_sales DESC;                      -- Sort by highest performing areas in terms of total sales
```

	region		salesperson	total_sales	total_profit	profit_...
2	Southeast Asia	APAC	Alejandro Ballentine	884423.1690000044	17852.329	0.02
0	Central	EU	Anna Andreadi	1720552.6184999964	215534.0685000008	0.13
12	Central	LATAM	Anna Andreadi	600510.0106400028	56163.5506399999	0.09
14	Central	US	Anna Andreadi	501239.8908000016	39706.3624999999	0.08
1	Oceania	APAC	Anthony Jacobs	1100184.6119999948	120089.1120000002	0.11
11	South	LATAM	Chuck Magee	617223.6778800004	28090.51788	0.05
13	South	EU	Chuck Magee	591961.4580000007	65515.8179999998	0.11
15	South	US	Chuck Magee	391721.9050000008	46749.4303000001	0.12
5	Africa	Africa	Deborah Brumfield	783773.2109999972	88871.631	0.11
16	Caribbean	LATAM	Giulietta Dortch	324280.8610400009	34571.32104	0.11
9	North	EU	Jack Lebron	625574.9849999968	91779.8549999997	0.15
10	North	LATAM	Jack Lebron	622590.617519996	102818.0975200001	0.17
8	East	US	Kelly Williams	678781.2399999935	91522.7800000001	0.13
4	EMEA	EMEA	Larry Hughes	806161.3109999954	43897.971	0.05
7	West	US	Matt Collister	725457.8244999961	108418.4489000003	0.15
17	Canada	Canada	Nicole Hansen	66928.17	17817.39	0.27

Rows: 18 Expand Table

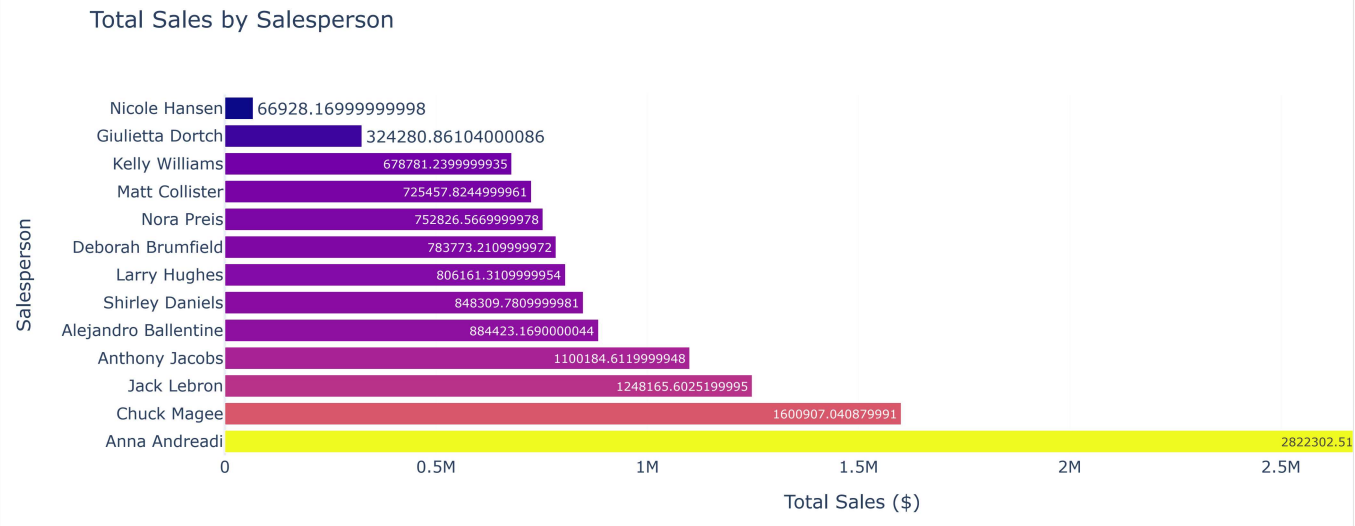
Projects Data DataFrame as

```sql
-- Step 1: Join orders with people and calculate total sales and profit per salesperson
SELECT
    p.person AS salesperson,                        -- Salesperson's name
    SUM(o.sales) AS total_sales,                    -- Total sales amount handled by this salesperson
    SUM(o.profit) AS total_profit,                  -- Total profit generated by this salesperson
    -- Calculate profit margin as (profit / sales), rounded to 2 decimal places
    ROUND((SUM(o.profit) / NULLIF(SUM(o.sales), 0))::numeric, 2) AS profit_margin
FROM orders o
JOIN people p ON o.region = p.region                -- Join using region (associates salesperson to order)
GROUP BY p.person                                   -- Group results by salesperson
ORDER BY total_sales DESC;                          -- Sort by highest total sales
```

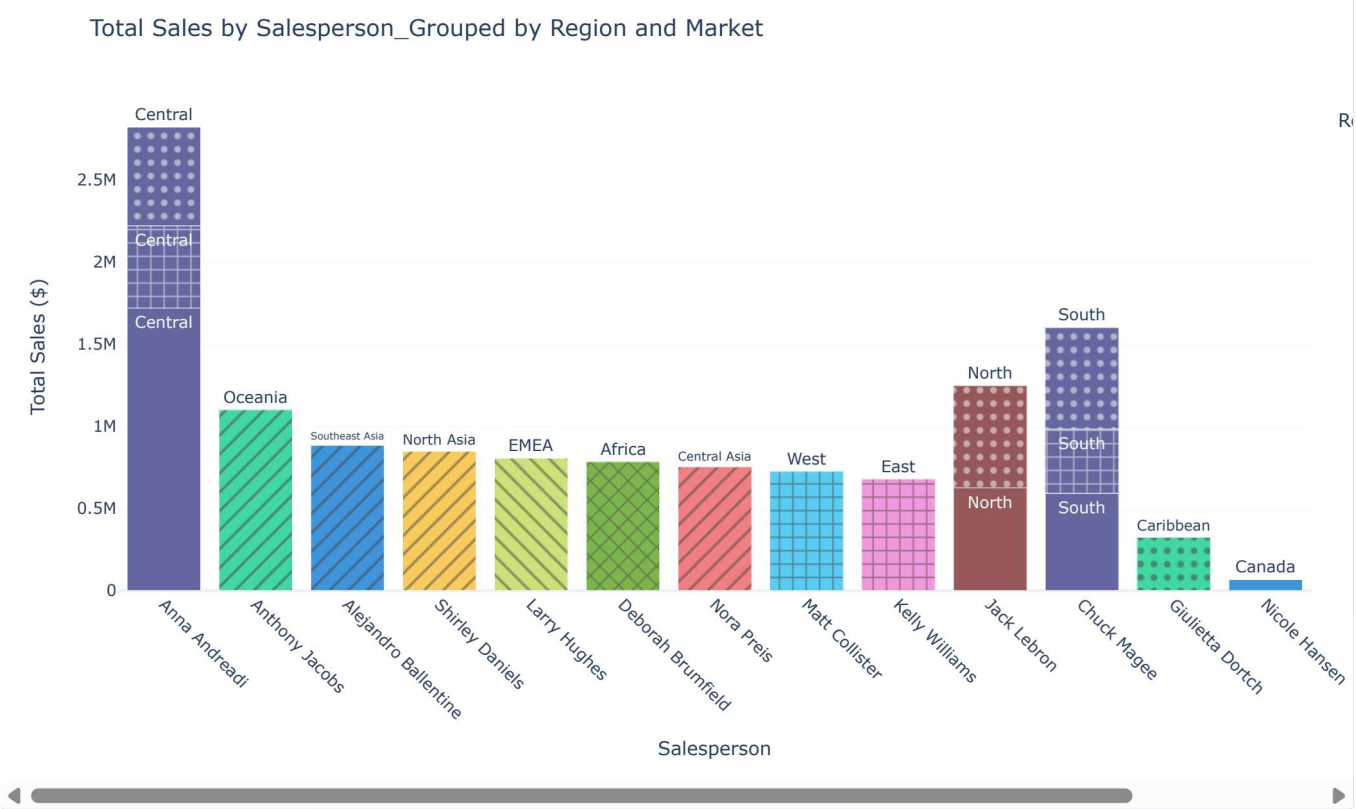
	salesperson	total_sales	total_profit	profit_...
0	Anna Andreadi	2822302.5199399963	311403.9816399976	0.11
1	Chuck Magee	1600907.040879991	140355.7661799999	0.09
2	Jack Lebron	1248165.6025199995	194597.9525200008	0.16
3	Anthony Jacobs	1100184.6119999948	120089.1120000002	0.11
4	Alejandro Ballentine	884423.1690000044	17852.329	0.02
5	Shirley Daniels	848309.7809999981	165578.4210000004	0.2
6	Larry Hughes	806161.3109999954	43897.971	0.05
7	Deborah Brumfield	783773.2109999972	88871.631	0.11
8	Nora Preis	752826.5669999978	132480.1870000003	0.18
9	Matt Collister	725457.8244999961	108418.4489000003	0.15
10	Kelly Williams	678781.2399999935	91522.7800000001	0.13
11	Giulietta Dortch	324280.8610400009	34571.32104	0.11
12	Nicole Hansen	66928.17	17817.39	0.27

Rows: 13 Expand Table

```python
#import plotly.express as px

# Round the profit margin to 4 decimal places
salesperson_df['profit_margin'] = salesperson_df['profit_margin'].round(4)

# Create a horizontal bar chart
fig = px.bar(
    salesperson_df,
    x='total_sales',               # Total sales on the x-axis
    y='salesperson',               # Salesperson names on the y-axis
    orientation='h',               # Horizontal bars
    color='total_sales',           # Color bars based on total sales
    text='total_sales',            # Show sales values on bars
    hover_data={
        'total_profit': True,      # Show total profit in hover
        'profit_margin': True,     # Show profit margin (rounded)
        'total_sales': ':.2f',     # Format sales nicely in hover
        'salesperson': False       # Already on y-axis, no need to repeat in hover
    },
    title='Total Sales by Salesperson'
)

# Customize layout
fig.update_layout(
    xaxis_title='Total Sales ($)',
    yaxis_title='Salesperson',
    coloraxis_showscale=False,  # Hide color scale legend
    template='plotly_white'
)

# Show the figure
fig.show()
```



```python
# Create a grouped bar chart showing total sales by salesperson
fig = px.bar(
    people_df,
    x="salesperson",
    y="total_sales",
    color="region",                  # Use region to color bars
    pattern_shape="market",          # Use market to differentiate with patterns
    title="Total Sales by Salesperson_Grouped by Region and Market",
    labels={
        "salesperson": "Salesperson",
        "total_sales": "Total Sales ($)",
        "region": "Region",
        "market": "Market"
    },
    text="region",                   # Display region on the bars
    height=600
)

# Format layout
fig.update_traces(textposition="outside")
fig.update_layout(
    xaxis_tickangle=45,
    template="plotly_white",
    legend_title="Region / Market"
)

fig.show()
```



Projects Data DataFrame as

```
SELECT *
FROM orders;
```

		order_id	order_date	ship_date	ship_mo...	cus...	
331	38123	CA-2013-118689	2013-10-03T00:00:00.000	2013-10-10T00:00:00.000	Standard Class	TC-20980	T
292	35487	CA-2014-166709	2014-11-18T00:00:00.000	2014-11-23T00:00:00.000	Standard Class	HL-15040	H
124	40336	CA-2013-117121	2013-12-18T00:00:00.000	2013-12-22T00:00:00.000	Standard Class	AB-10105	A
47	35395	CA-2011-116904	2011-09-23T00:00:00.000	2011-09-28T00:00:00.000	Standard Class	SC-20095	S
19	12069	ES-2014-1651774	2014-09-08T00:00:00.000	2014-09-14T00:00:00.000	Standard Class	PJ-18835	P
78	31806	CA-2012-145352	2012-03-16T00:00:00.000	2012-03-22T00:00:00.000	Standard Class	CM-12385	C
217	27720	ID-2011-64599	2011-02-10T00:00:00.000	2011-02-15T00:00:00.000	Standard Class	CA-11965	C
315	23212	IN-2013-50809	2013-06-12T00:00:00.000	2013-06-17T00:00:00.000	Standard Class	CA-12775	C
83	50788	MO-2014-2000	2014-10-28T00:00:00.000	2014-10-30T00:00:00.000	Second Class	DP-3105	D
652	38963	US-2013-140158	2013-10-05T00:00:00.000	2013-10-09T00:00:00.000	Standard Class	DR-12940	D
84	37817	CA-2014-138289	2014-01-17T00:00:00.000	2014-01-19T00:00:00.000	Second Class	AR-10540	A
644	30958	IN-2013-83191	2013-12-16T00:00:00.000	2013-12-20T00:00:00.000	Standard Class	AS-10630	A
160	32382	US-2013-143819	2013-03-02T00:00:00.000	2013-03-06T00:00:00.000	Standard Class	KD-16270	K
112	35574	US-2013-107440	2013-04-17T00:00:00.000	2013-04-21T00:00:00.000	Standard Class	BS-11365	B
336	21263	IN-2014-56206	2014-06-24T00:00:00.000	2014-06-28T00:00:00.000	Standard Class	MB-17305	M
328	40287	US-2012-128587	2012-12-24T00:00:00.000	2012-12-30T00:00:00.000	Standard Class	HM-14860	H

Rows: 4,761 ⚠ truncated from 51,290 rows Expand Table

```python
# Step 1: Count the number of orders per customer
top_customers = (
    Orders_df.groupby("customer_name")["order_id"]
    .nunique()                          # count unique orders
    .sort_values(ascending=False)       # sort in descending order
    .head(10)                           # get top 10
    .reset_index(name="total_orders")   # convert to DataFrame
)

# Step 2: Plot the bar chart
fig = px.bar(
    top_customers,
    x="customer_name",
    y="total_orders",
    title="Top 10 Frequent Customers by Number of Orders",
    labels={"customer_name": "Customer", "total_orders": "Number of Orders"},
    text="total_orders",
    height=500
)

fig.update_traces(textposition="outside")
fig.update_layout(xaxis_tickangle=45, template="plotly_white")

fig.show()
```


```python
# Drop rows where quantity is missing
# exclude rows where quantity is missing so the analysis remains accurate.
clean_orders_df = Orders_df.dropna(subset=['quantity'])

# Calculate correlation matrix
correlation_matrix = clean_orders_df[['discount', 'sales', 'quantity', 'profit']].corr()

print("Correlation Matrix:")
print(correlation_matrix)
```

```
Correlation Matrix:
          discount     sales  quantity    profit
discount  1.000000 -0.086718 -0.019886 -0.316477
sales    -0.086718  1.000000  0.313584  0.484914
quantity -0.019886  0.313584  1.000000  0.104379
profit   -0.316477  0.484914  0.104379  1.000000
```