Project Introduction:

In a real-world business scenario, I analyzed sales data from a Berlin store for September 2025, extracted from cloud-based business management software. To share the project publicly, I converted the dataset into a fully generic format, while preserving the original structure and realistic characteristics of the data, so that it reflects a retail store scenario and retains educational and analytical value without exposing sensitive information. Although this is a small yet in-depth analysis, the structure, workflow, and techniques applied here can easily scale to larger datasets, additional stores, multiple regions, and longer time periods.

Objective:

The task was to generate a monthly sales report with a focus on uncovering trends among the top 20 items by quantity sold and by total sales value.

Data Preparation:

Starting with a dataset comprising 988 rows and 23 columns, I focused on six critical fields:

- Item: Representing the name of each product.
- Quantity: The number of units sold per transaction.
- Discount %: The percentage discount applied to a specific item during the purchase.
- Price Unit Including VAT: The unit price of each item, inclusive of tax.
- Total discount: The monetary value of the discount applied to the item or transaction.
- Net price unit including VAT: The actual amount paid by the customer for each unit, including VAT, after any discounts. This reflects the real revenue received per item.

The data was then saved into an Excel file, making it easy to work with and analyze using Python. This step helped to explore the data thoroughly, ensuring the analysis stayed on track with the project goals.

Exploratory Data Analysis:

I observed that some items were purchased multiple times in a single transaction, so I created an 'Occurrences' column to track how often each item appears in the dataset.

This is the code that supported the operation:
df['Occurrences']=df.groupby('Item')['Item'].transform('size')

Explanation: The data gets grouped by each item name found in (Item) column. The transform('size') function then counts the number of entries for each item and takes that count back into a new column in the dataset. Each row now shows the count of occurrences per item.

```
df['Occurrences'] = df.groupby('Item')['Item'].transform('size')
df
```

	Item	Quantity	Discount %	Price unit including VAT	Total discount	Price unit netto including VAT	Occurrences
1	ITEM 2	1	0	29.9	0.0	30	4
2	ITEM 3	1	0	22.9	0.0	23	4
3	ITEM 4	1	0	36.9	0.0	37	4
4	ITEM 5	1	0	21.9	0.0	22	7
5	ITEM 6	1	0	21.9	0.0	22	7
...	...	...	...	...	...	...	...
982	ITEM 983	1	0	28.9	0.0	29	1
983	ITEM 220	1	0	23.9	0.0	24	8
984	ITEM 329	1	0	26.9	0.0	27	9
985	ITEM 29	3	0	21.9	0.0	22	15
986	ITEM 34	1	0	24.9	0.0	25	4

934 rows × 7 columns

Yet, recognizing that occurrences alone didn't provide the complete context, I added a 'Total Quantity' column to show total units sold per item.

This is the code that supported the operation:
df['Total Quantity'] = df.groupby('Item')['Quantity'].transform('sum')

Explanation: The data gets grouped by each item name found in [Item] column. And ['Quantity'].transform('sum') adds up all quantities within each group, calculating the total number of units sold for each item. The sum for each group is then applied to a new column, "Total Quantity," in every row, allowing each row for an item to show its total sales number.

```
df['Total Quantity'] = df.groupby('Item')['Quantity'].transform('sum')
df
```

	Item	Quantity	Discount %	Price unit including VAT	Total discount	Price unit netto including VAT	Occurrences	Total Quantity
1	ITEM 2	1	0	29.9	0.0	30	4	4
2	ITEM 3	1	0	22.9	0.0	23	4	4
3	ITEM 4	1	0	36.9	0.0	37	4	2
4	ITEM 5	1	0	21.9	0.0	22	7	7
5	ITEM 6	1	0	21.9	0.0	22	7	7
...	...	...	...	...	...	...	...	...
982	ITEM 983	1	0	28.9	0.0	29	1	1
983	ITEM 220	1	0	23.9	0.0	24	8	8
984	ITEM 329	1	0	26.9	0.0	27	9	9
985	ITEM 29	3	0	21.9	0.0	22	15	17
986	ITEM 34	1	0	24.9	0.0	25	4	4

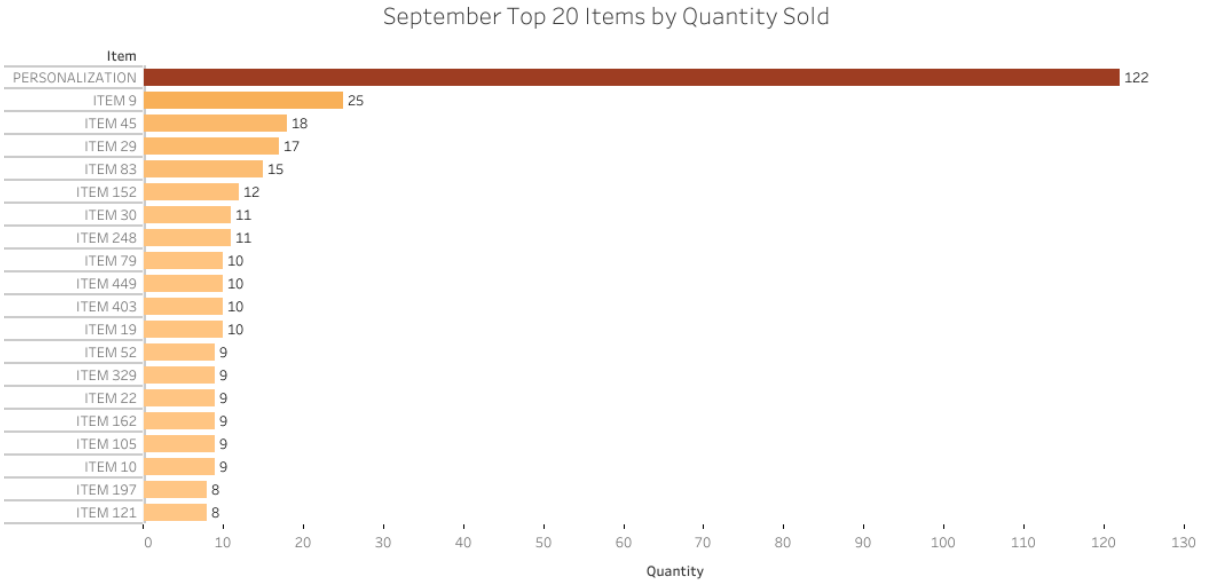
934 rows × 8 columns

I was assigned to identify the top 20 items sold by both value and quantity. After carefully cleaning and preparing the data, I successfully determined them. The steps and operations that made this possible include:

Top 20 Items by Quantity Sold:

```
top_20_items_quantity = (df.groupby('Item')['Quantity'].sum().reset_index().sort_values(by='Quantity',
                         ascending=False).head(20).reset_index(drop=True))
top_20_items_quantity.index = top_20_items_quantity.index + 1
top_20_items_quantity
```

This code groups the DataFrame by the Item column, gathering all records related to each product. For every group, it sums the Quantity column to calculate the total number of units sold per item. The result is a Series where Item serves as the index and the values represent the total quantities sold. Finally, using reset_index() converts this Series back into a DataFrame, turning the Item index into a regular column for easier data manipulation. Following this, the DataFrame is sorted in descending order by the total quantities. This step ensures that the items with the highest sales volumes appear at the top. The head(20) function is used to select the top 20 products, ensuring only the best-selling items are captured. After extracting these top 20 products, reset_index(drop=True) is employed to organize the index in a clean sequence, starting from 1. This adjustment provides a clear and intuitive ranking, making it easier for users to interpret and navigate the data. As seen in the image below.



This horizontal bar chart displays the top 20 best-selling items for September, ranked by quantity sold. The length of each bar represents the number of units sold, with the highest-selling items at the top. The visualization highlights key products to help quickly identify trends and opportunities.

Top 20 Items by Net Value:

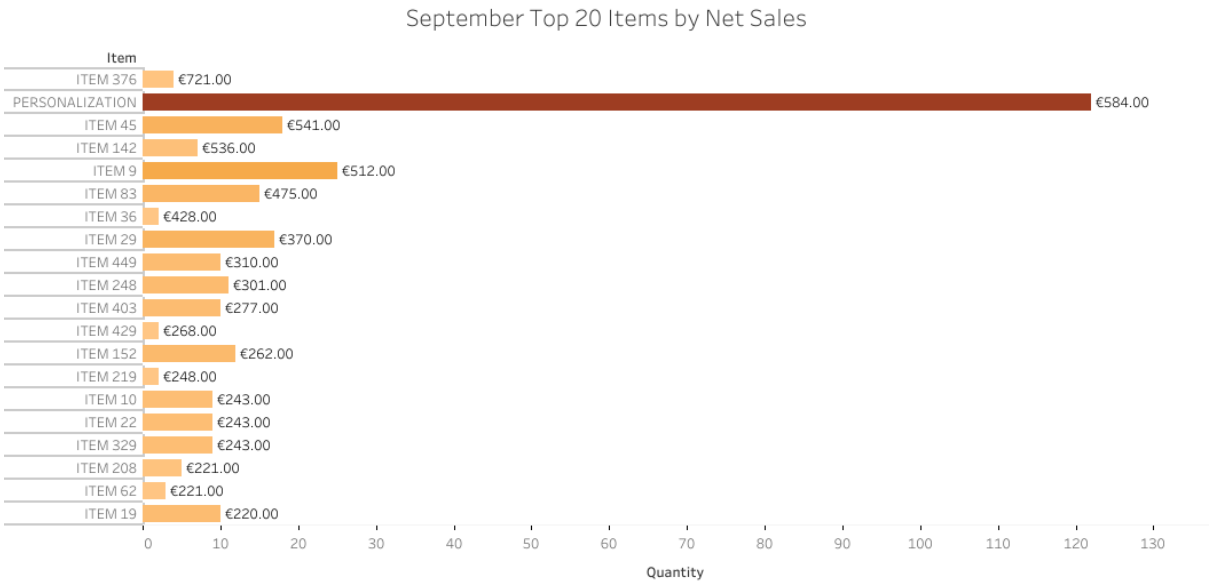
```
top_20_items_value_netto = (df.assign(IndividualTotalValue=lambda x: x['Quantity'] * x['Price unit net including VAT'])
    .groupby('Item', as_index=False)
    .agg({'Quantity': 'sum', 'IndividualTotalValue': 'sum'})
    .rename(columns={'Quantity': 'Quantity', 'IndividualTotalValue': 'Total Net Value'})
    .sort_values(by='Total Net Value', ascending=False)
    .reset_index(drop=True)
    .head(20))

top_20_items_value_netto['Total Net Value'] = top_20_items_value_netto['Total Net Value'].apply(lambda x: f'€{x:,.2f}')
top_20_items_value_netto.index = top_20_items_value_netto.index + 1
top_20_items_value_netto
```

Using the assign() method, I introduced a new column called IndividualTotalValue to our DataFrame. This was achieved through a lambda function, which efficiently processed each row. It multiplied the Quantity (Quantity) by the Price unit netto including VAT (Price unit netto Including VAT) to determine the sales net value for each transaction. This step helped provide a clear and precise accounting of each item's contribution to total net sales, enhancing our data's depth and detail.

The data is then grouped by Item, consolidating all transactions related to each unique product. Within these groups, aggregation is applied to calculate two key metrics: the total quantity sold (Quantity) and the total sales value (IndividualTotalValue). This step provides a clear overview of each item’s performance across all sales metrics. The columns are renamed for clarity, ensuring they accurately reflect the data they hold. After this, the items are sorted by "Total Net Value" in descending order, which brings the top earners right into the spotlight.
I then selected the top 20 entries to highlight the leading products. The index is reset to start cleanly from 0, ensuring the data remains well-organized and easy to read.

The "Total Net Value" column is formatted to include the Euro symbol (€) and ensure each amount is displayed with two decimal places. The apply() function, paired with a lambda, is key here. It works through each value in the "Total Net Value" column, formatting numbers to include commas as thousand separators and ensuring they display with two decimal places for precision. This formatting ensures a consistent, professional look that makes financial figures clear and easy to read during reviews. To finalize, the index is adjusted to start from 1, creating a user-friendly ranking display.



This horizontal bar chart shows the top 20 items by net sales value for September, ranked from highest to lowest. Each bar's length represents the quantity sold, while the label shows the total revenue generated, highlighting the most valuable products.

Conclusion:

A well-executed sales data analysis plays a key role in supporting the strategic goals of any retail business. It helps reveal trends, identify best-selling products, and understand customer preferences. These insights guide decisions in areas such as storytelling, marketing, and customer communication. When teams understand which items perform well and the reasons behind it, they can refine their approach and stay aligned with the overall direction of the business.

In this example, the analysis shows a particularly strong performance in product personalization. Adding a personal touch to an item that the customer already wants creates extra value with only a small additional budget. This simple upgrade is having a noticeable impact, generating net revenue that is close to some of the store's highest-priced products. With this information, both management and store teams can strengthen their storytelling, engage customers more meaningfully, and offer added value that benefits everyone.

Overall, this type of analysis helps keep people, products, and business goals connected. It supports a more informed, responsive, and value-driven retail experience.