Sales Data: Exploratory Data Analysis report

By Assim Alharbi

Contents

Introduction........1
Collect........2
Clean........2
Analysis........9
Findings........16
Conclusion........17

Figures

Figure 1: Import dataset to Power Bi........11
Figure 2 Number of orders over time........12
Figure 3 Total revenue across cities and branches........12
Figure 4 Product price boxplot........13
Figure 5 Orders per time of day........14
Figure 6 Orders per day of week........15

Introduction

- Purpose:

This is an assessment by the **SDC App** as part of the recruitment process for the role of data analyst intern, the goal is to assess my abilities in data analysis.

- Objective:

The objective of this assignment is to assess your ability to perform exploratory data analysis (EDA) on a given dataset and derive meaningful insights. This task will evaluate your skills in data manipulation, visualization, and interpretation.

- Dataset:

You will be provided with a dataset containing information about sales transactions. The dataset includes columns such as order ID, date, product ID, quantity, etc.

- Tasks:

1. Load the dataset into your preferred data analysis environment.

2. Explore the data and perform any necessary data cleaning.

3. Conduct an exploratory data analysis (EDA) to uncover patterns, trends, or insights in the dataset.

4. Use visualizations (e.g., histograms, box plots, scatter plots) to enhance your understanding of the data.

5. Feel free to perform any additional analyses or create visualizations that you believe will provide valuable insights into the dataset. You are encouraged to use any other data or resources you find relevant for a more comprehensive analysis.

6. Summarize your findings and any actionable insights you have derived from the analysis.

- Submission:

Submit your analysis in a well-documented report. Include code, visualizations, and explanations for each step. Make sure to clearly state your findings and any assumptions made during the analysis.

- Evaluation Criteria

You will be evaluated on the completeness of your analysis, the accuracy of your insights, the quality of visualizations, and your ability to communicate your findings effectively.

Collect

The dataset was already provided with the assessment, and I decided not to collect any more data from external sources. Therefore, no more actions will be taken in this phase, and I will be moving to the next process immediately.

Clean

I'll be using SQL to clean the data, but before that, I noticed that the data had merging rows at the columns ***order_id, date format, city, branch, and product_category***, now importing the data with this shape is going to cause a lot of ***NULL*** values to appear, and as it is easier to fix in Excel rather than SQL, I will be using Power Query to fix this, but first I have to select all the merged cells, then click on “Unmerge Cells” in the home ribbon, now the merged cells are gone and replace with empty cells, so to fill these sells I’m going to use a function from Power Query called “Fill Down”, which will fill all the empty cells. Now the data is ready to be imported into SQL Server.

Now in SQL Server all data types are adjusted when importing, and column names with spaces are changed.
Now we move to the cleaning process step-by-step

- Rename the ***date_format*** column as it appears to have the name of a function and might cause confusion

```
EXEC sp_rename 'sales.date_format', 'order_date', 'COLUMN';
```

- Finding if any product name is associated with different categories

```
WITH products_mapping AS (
SELECT DISTINCT product_name, product_category
FROM sales
WHERE product_category IS NOT NULL

)

SELECT *
FROM products_mapping as mp
JOIN products_mapping as mp2
ON mp.product_name = mp2.product_name
WHERE mp.product_name = mp2.product_name AND
mp.product_category <> mp2.product_category
```

- Handling all ***NULL*** values in the ***product_category*** column

```
WITH products_mapping AS (
SELECT DISTINCT product_name, product_category
FROM sales
WHERE product_category IS NOT NULL

)

UPDATE sales
SET product_category = (
SELECT pm.product_category
FROM products_mapping pm
WHERE pm.product_name = sales.product_name
)
WHERE product_category IS NULL;
```

- Checking to see if all ***NULL*** values in ***product_category*** are handled

```
SELECT *
FROM sales
WHERE product_category IS NULL
```

- Checking if the ***order_date*** column has any inconsistency

```
SELECT *
From sales
Order By order_date DESC
```

- As a result of the previous query two orders came up with unrealistic dates:
  - 1. '2030-12-12 23:20:05.000'
  - This date is problematic as both the year and month are off, which suggests it's either an error or generated by a system glitch (likely not a data entry mistake)
  - So, I can either remove the order or set a default date (Average or any other date)
  - I will remove the entire order, with all its rows

```
DELETE FROM sales
WHERE order_date = '2030-12-12 23:20:05.000'
```

  - 2. '2030-05-08 01:25:53.000'
  - This date is more reasonable as it falls within the same months as other entries (**April and May**) but has an unrealistic year (**2030**).
  - Given that the month and day are consistent with other valid entries, this seems more like a data entry error, which can be corrected
  - I will adjust the year back to (**2021**)

```
UPDATE sales
SET order_date = '2021-05-08 01:25:53.000'
WHERE order_date = '2030-05-08 01:25:53.000'
```

- Checking if the ***city*** column has any inconsistency

```
SELECT DISTINCT city
From sales
```

- We have empty cells, let's check if those records have empty ***branch*** also

```
SELECT city, branch
From sales
WHERE city = ''
```

- As the ***branch*** is not empty, we can use mapping to fill in the cells

```
WITH city_branch AS (
SELECT DISTINCT city, branch
FROM sales
WHERE city <> '' and branch <> ''
)

UPDATE sales
SET city = (
SELECT city
FROM city_branch as cb
WHERE cb.branch =  sales.branch
)
WHERE city = ''
```

- Checking if any ***city*** value is still missing

```
SELECT DISTINCT city
FROM sales
```

- Now we move to the ***branch*** column, as we saw in the output there's also missing values in here

```
SELECT DISTINCT branch
From sales
```

- Let's check which ones are missing

```
SELECT order_id, order_date, city, branch
From sales
WHERE branch = ''
```

- As we can see, both the orders with the missing branches are in Jeddah, now with Jeddah having 3 different branches (Unlike Makkah and Taif) we must decide what to do with these missing values:
  1. We can find the ***branch*** with the highest number of orders and choose it as a default.
  2. We can create another value like ***N/A*** for missing values.
  3. We can delete the rows as they might skew our analysis.

- As the missing values here are only 10 rows which won't skew the analysis, I will go with the first approach, and choose a default ***branch***

- Finding the ***branch*** with the most orders

```
SELECT branch, COUNT( DISTINCT order_id) as num_of_orders
FROM sales
WHERE city = 'Jeddah'
GROUP BY branch
```

```
ORDER BY num_of_orders DESC
```

- Now we replace the missing values with the ***branch*** with the most orders

```
UPDATE sales
SET branch = (
SELECT TOP 1 branch
FROM sales
WHERE city = 'Jeddah'
GROUP BY branch
ORDER BY COUNT( DISTINCT order_id) DESC
)
WHERE branch = ''
```

- Checking if the ***product_category*** column has any inconsistency

```
SELECT DISTINCT product_category
FROM sales
```

- Checking if the ***product_name*** column has any inconsistency

```
SELECT DISTINCT product_name
FROM sales
Order By product_name
```

- Checking if the ***quantity*** column has any inconsistency

```
SELECT DISTINCT quantity
FROM sales
Order By quantity DESC
```

- The numbers (**110**) and (**100**) seem high, but not to the point of being outliers.

- Checking if the ***total*** column has any inconsistency

```
SELECT  total
FROM sales
ORDER BY total DESC
```

- Before going any further, we need to create a column to price items

```
ALTER TABLE sales
ADD product_price FLOAT;
```

- Add values to the column

```
UPDATE sales
SET product_price = (
total / quantity
)
```

- Checking if the ***order_total*** column has any inconsistency

```
SELECT  order_total
FROM sales
ORDER BY order_total
```

- There are ***NULL*** values

- Let's check which ones are ***NULL***

```
SELECT *
FROM sales
WHERE order_total IS NULL
```

- Before going further and changing the ***NULL*** values, we need to do some calculations to better understand this column, since it doesn't make any sense at first glance, we need to better understand how these results came up.
- Let's create a new calculated column to count the total for each order
- But first, we need to rename the ***order_total*** column to ***discounted_total*** for more readability

```
EXEC sp_rename 'sales.order_total', 'discounted_total', 'COLUMN'
```

- Now let's create a new calculated column and name it ***order_total***
- *Note: This might cause confusion, but the naming makes more sense this way*

```
ALTER TABLE sales
ADD order_total float;
```

- Now we add values to the ***order_total*** column

```
WITH add_real_total AS (
SELECT order_id, quantity, product_price, total, SUM(total) OVER
(PARTITION BY order_id ORDER BY order_id) as sum_order_total,
discounted_total, order_total
FROM sales
)

UPDATE sales
SET order_total = art.sum_order_total
FROM sales as s
JOIN add_real_total as art
ON art.order_id = s.order_id
```

- Now we see what type of discount was applied to the ***discounted_total***

```
SELECT *, round((100 - discounted_total/ order_total * 100),2) as discount
FROM sales
ORDER BY discount
```

- So, after some basic calculations, it seems that a discount has been applied to the ***discounted_total***.
- **66.67%** seems to be the discount amount, with only 4 orders not meeting that.
- Order Id (**479602**) has different ***discounted_total*** across different rows which shouldn't be the case.
- This could have multiple reasons, one way of fixing this is manually by choosing the last value (**76**) which is the only correct ***discounted_total*** value of this order based on the discount percentage.

```
UPDATE sales
SET discounted_total = 76
WHERE order_id = '479602'
```

- Another way to do this is to do some calculations, and then update the wrong entries based on the outcomes.
- Let's try this on the other order ids which have wrong ***discounted_total***

```
WITH discount_cte AS (
SELECT *, ROUND((100 - discounted_total/ order_total * 100),2) as discount
FROM sales
)

UPDATE sales
SET discounted_total = (s.order_total - (s.order_total * 0.6667))
FROM sales s
JOIN discount_cte as d
ON d.order_id = s.order_id
WHERE discount <> 66.67
```

- Now let's change the rows with ***NULL*** values in the ***discounted_total***

```
UPDATE sales
SET discounted_total = (order_total - (order_total * 0.6667))
FROM sales
WHERE sales.discounted_total IS NULL
```

- Let's check everything one last time

```
SELECT *
FROM sales
```

- Now I believe the data is ready for analysis,
- Let's dive in!

Analysis

In the analysis phases, I plan to divide this process into two stages, the first is a simple descriptive analysis, and the goal of it is to better understand the data. This will be done in SQL.
The second stage is the actual exploratory analysis, some insights will only be revealed once within a graph or a chart. This will be done in Power BI, alongside the dashboard.

The following are the step-by-step analysis done in SQL:

- Number of orders:

```
SELECT COUNT(DISTINCT order_id) as number_of_orders
FROM sales
```

- Number of items purchased:

```
SELECT SUM(quantity) as number_of_items
FROM sales
```

- Number of orders per month:

```
SELECT CASE WHEN MONTH(order_date) = 4 THEN 'April'
ELSE 'May'
END as month_of_year
, count(DISTINCT order_id) as number_of_orders
FROM sales
GROUP BY MONTH(order_date)
ORDER BY number_of_orders DESC
```

- Number of orders per city:

```
SELECT city, COUNT(DISTINCT order_id) as number_of_orders
FROM sales
GROUP BY city
ORDER BY number_of_orders DESC
```

- Number of orders per branch:

```
SELECT city, branch, COUNT(DISTINCT order_id) as number_of_orders
FROM sales
GROUP BY city, branch
ORDER BY number_of_orders DESC
```

- Percentage of orders per branch:

```
SELECT city, branch, COUNT(DISTINCT order_id) AS orders_per_branch,
CAST(ROUND(COUNT(DISTINCT order_id) * 100.0 / SUM(COUNT(DISTINCT
order_id)) OVER (),2) AS decimal(5,2)) AS percentage_of_total
FROM sales
GROUP BY city, branch
```

```
ORDER BY percentage_of_total DESC
```

- Most purchased items:

```
SELECT product_name, COUNT(product_name) as number_of_orders,
SUM(quantity) as quantity_of_items
FROM sales
GROUP BY product_name
ORDER BY quantity_of_items DESC
```

- Number of orders per category, and number of items per category:
    - *Number of orders = the number of orders where a specific category was purchased.*
    - *Quantity of items = the number of orders * the quantity of the items purchased*

```
SELECT product_category, COUNT(DISTINCT order_id) as number_of_orders,
SUM(quantity) as quantity_of_items
FROM sales
GROUP BY product_category
ORDER BY number_of_orders DESC
```

- Percentage of orders per category:

```
SELECT product_category, COUNT(DISTINCT order_id) as number_of_orders,
CAST(ROUND(COUNT(DISTINCT order_id) * 100.0 / SUM(COUNT(DISTINCT
order_id)) OVER (),2) AS decimal(5,2)) AS percentage_of_total
FROM sales
GROUP BY product_category
ORDER BY number_of_orders DESC
```

- The price of each item:

```
SELECT DISTINCT product_category, product_price
FROM sales
ORDER BY product_category, product_price DESC
```

- The max, min, and average prices per category:

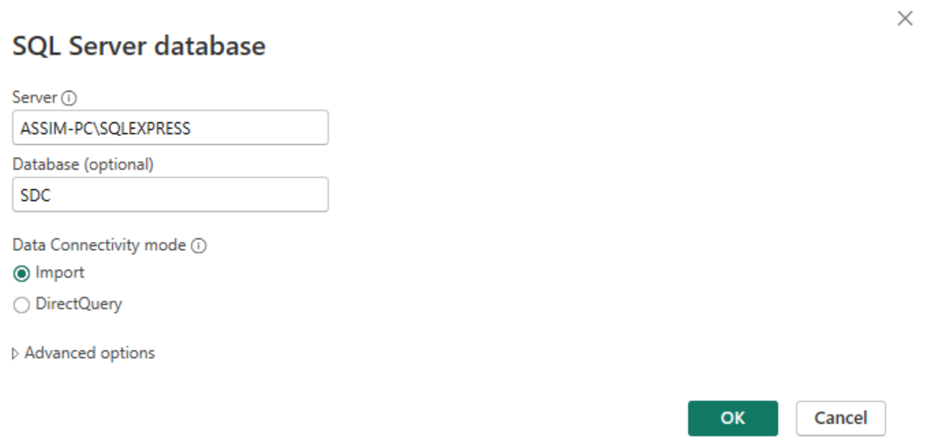
```
SELECT product_category, MAX(product_price) as max, MIN(product_price) as
min, ROUND(AVG(product_price),2) as avg
FROM sales
GROUP BY product_category;
```

- The total before and after the discount, and the amount discounted:

```
WITH total_income_CTE AS (
SELECT DISTINCT order_id, order_total, discounted_total
FROM sales
)
```

```
SELECT SUM(order_total) AS total_before_discount,
ROUND(SUM(discounted_total),2) as total_after_discount,
SUM(order_total) - ROUND(SUM(discounted_total),2) as discounted_amount
FROM total_income_CTE
```

- Now that the SQL analysis is done, let's move to Power BI to continue the analysis and create the dashboard.
- First thing, we need to export the data from SQL Server, we have two options here, **import** or **DirectQuery**.
- I will choose to **import** in this case, as the data will not be changed hence, there is no need to use **DirectQuery**.



SQL Server database

Server: ASSIM-PC\SQLEXPRESS

Database (optional): SDC

Data Connectivity mode: [✓ Import] DirectQuery

Figure 1: Import dataset to Power Bi

- Now that the data is ready to be discovered even more, I'll move to the visualizations and explain my findings.
- Note: I've created a new section for the findings, so the graphs in this section will only have a brief description.

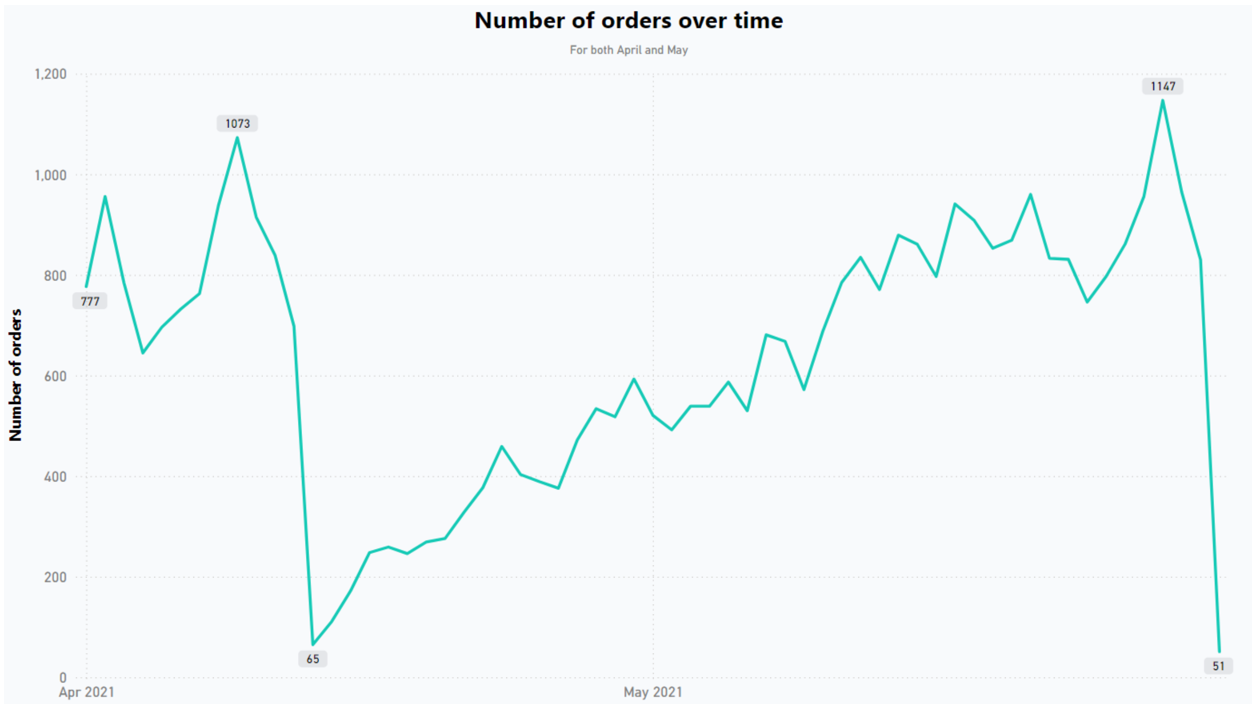


Figure 2 Number of orders over time

- A line chart that shows the number of orders over time, as the data is only for two months, this won't reveal any patterns or customer habits, but it shows the days with the highest and lowest orders.

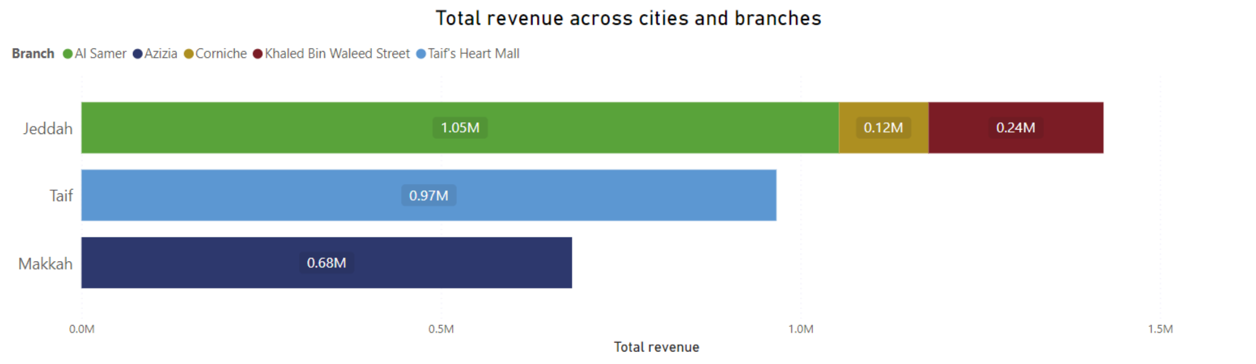


Figure 3 Total revenue across cities and branches

- This bar chart shows the contribution of all cities and branches to the total revenue, as we can see **Jeddah** has the highest revenue, with Al Samer branch leading with over **1 million**.

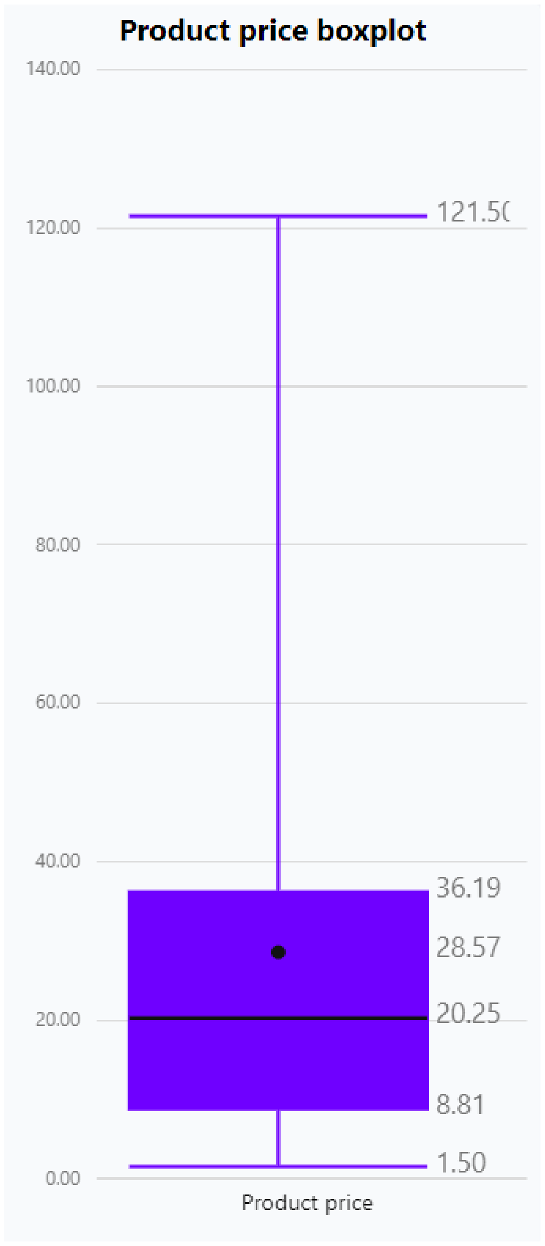


Figure 4 Product price boxplot

- A boxplot that shows the distribution of the product’s prices:
    - The average price: **28.57**
    - The highest price: **120.50**
    - The lowest price: **1.50**
    - The median price: **20.25**
    - **50%** are between **8.81** and **36.19**

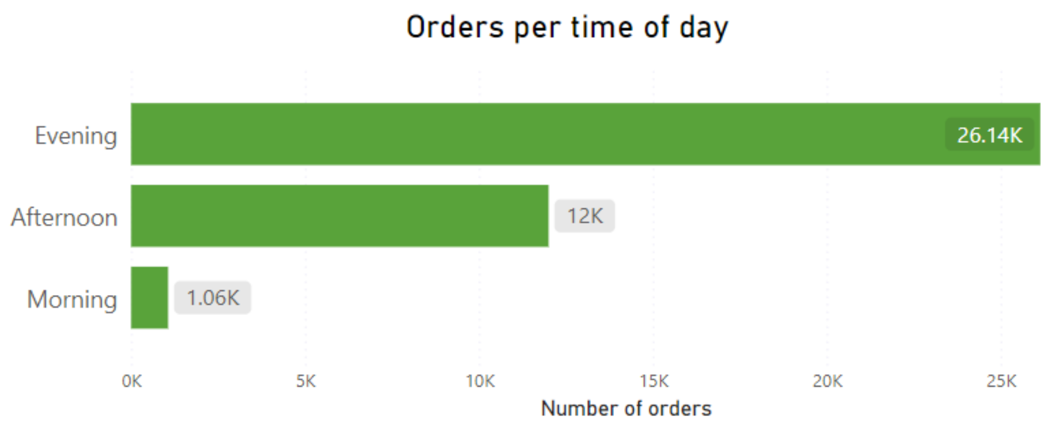


Figure 5 Orders per time of day

- I created a calculated column to see the distribution of orders during the time of day.
- This showed the domination of the evening's orders with over **26K** orders.
- The formula:

```
if Time.Hour([order_date]) >= 6 and Time.Hour([order_date]) < 12 then
"Morning"
else if Time.Hour([order_date]) >= 12 and Time.Hour([order_date]) < 18
then "Afternoon"
else "Evening"
```

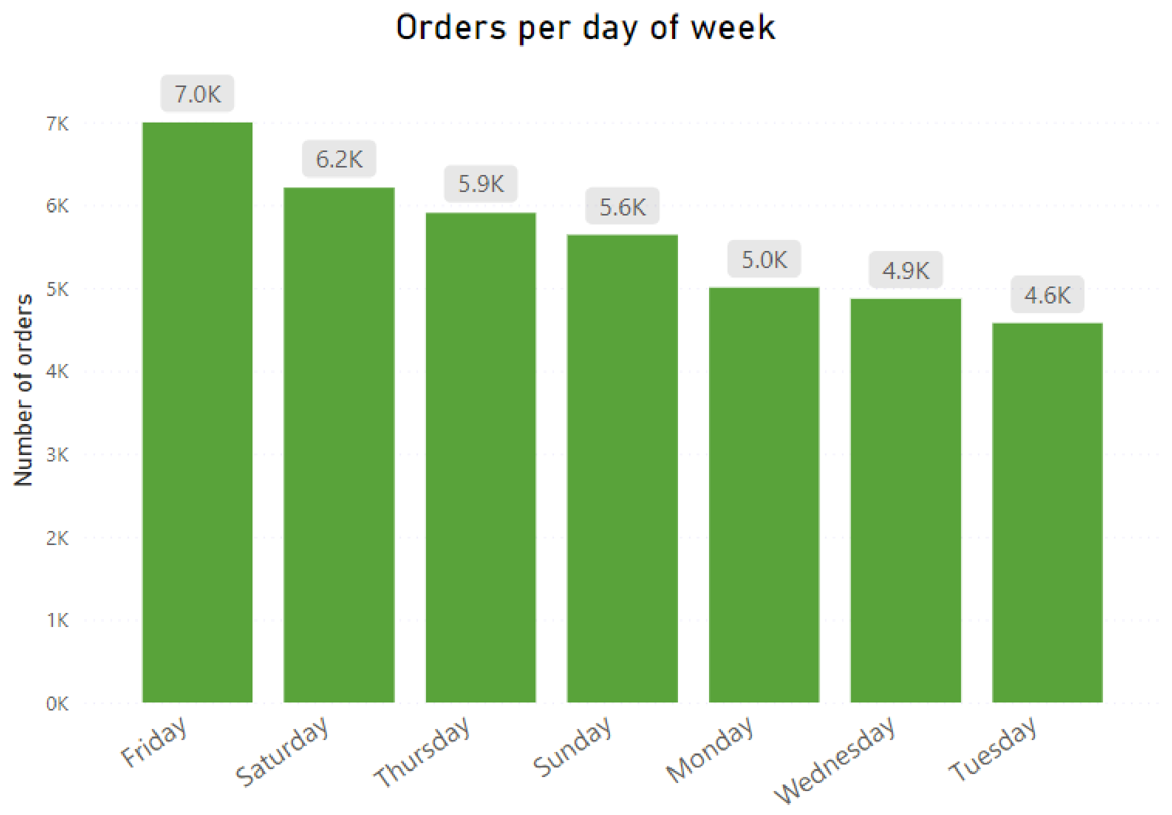
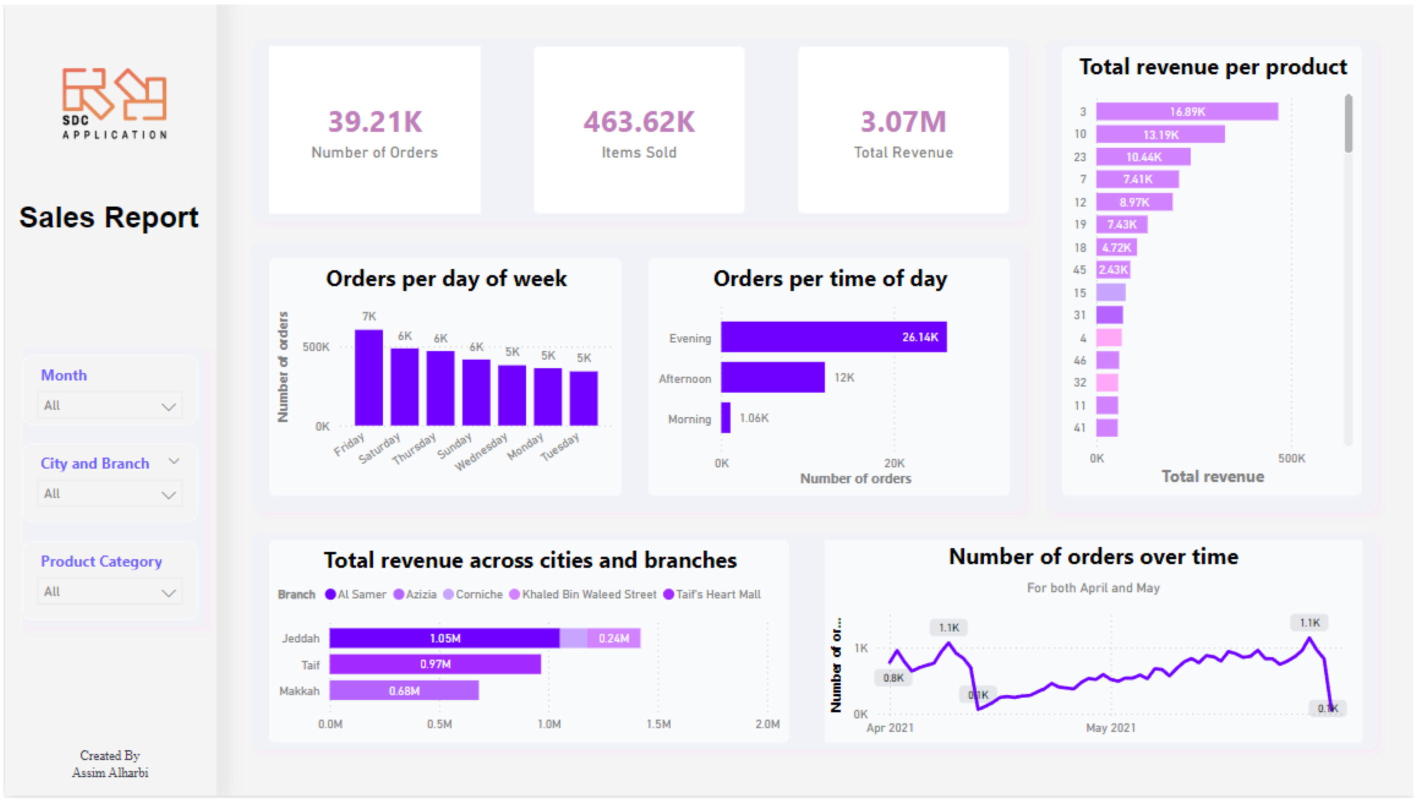


Figure 6 Orders per day of week

- I created another calculated column to see the distribution of orders during the week.
- This showed very close results, with Friday leading with **7K** orders
- The DAX formula:

```
day_of_week = FORMAT('sales'[order_date], "dddd")
```

- Finally, I've created an interactive dashboard that summarizes almost everything using Power BI, you can find it here, or through the link that I've sent in the email.



Findings

- Order Trends:
  - Number of orders: **39.21K**
  - Number of items sold: **463.62K**
  - Total Revenue: **3.07M**
  - Total before discount: **9.21M**
  - Discount amount: **66.67%**
  - Discounted amount: **6.14M**
  - May had **23294** orders, while April had **15911** orders

- Product Performance:
  - Sides are the most purchased: **189156** times
  - Combos are the least purchased: **5655** items
  - Item number 6 is the most purchased: **53737** times
  - Item number 76 is the least purchased: **22** times
  - **36.82%** of orders had meals, while only **1.27%** had deserts

- City and branches Insights:

    - Al Samer branch has the highest revenue: **1.01M**
    - The Corniche branch has the lowest revenue: **124K**
    - Makkah has the lowest revenue: **682K**

- Number of orders over time:
    - May 28 has the highest orders: **1147**
    - May 31 has the lowest orders: **51**
    - May 31 also had the biggest decline compared to the previous day, moving from **830** orders on May 30 to **51** orders on May 31.
    - **26.14K** orders were made in the evening, while only **1.06K** were made in the morning.
    - Friday has the highest number of orders with **7K**, while Tuesday has the lowest with **4.6K** orders.

Conclusion

This project utilized SQL and Power BI to perform an exploratory data analysis on sales data, uncovering key insights and trends. The analysis provided a comprehensive understanding of revenue distribution, ordering trends, and performance across various dimensions, enabling data-driven decision-making and strategic planning.