Wolt BI Developer assignment

Pauli Lahtinen
+358445140813
paulilahtinen01@gmail.com

TABLE OF CONTENTS

1. Introduction

I completed this project using SQLite3 on SQLiteStudio, Python and Excel. I imported the data to a database named Wolt_BI0.db using the Python script below:

```python
import sqlite3
import pandas as pd

conn = sqlite3.connect("Wolt_BI.db")
cursor = conn.cursor()
conn.commit()

df = pd.read_csv('item_data.csv') #reading the csv file into a dataframe
df.to_sql('item_data', conn, if_exists='append', index=False) #creating a table in the database

df = pd.read_csv('purchase_item_data_final.csv')
df.to_sql('purchase_item_data', conn, if_exists='append', index=False)

df = pd.read_csv('purchase_data_final.csv')
df.to_sql('purchase_data', conn, if_exists='append', index=False)

conn.commit()
conn.close()
```

1.1 Data Cleaning

I didn't set any primary of foreign key constraints for the data, since I didn't know the dataset well enough. I did howerer create a new table called clean_item_data, where I queried items based on distinct PRODUCT_ID and VENUE_ID and the latest TIMESTAMP. This came with the assumption that prices of products or currency exchange rates haven't dramatically changed over the given time period and therefore calculations wouldn't be dramatically changed either. It also came with the benefit of simplifying purchase profitability calculations, since the same PRODUCT_ID always had the same data. A more accurate solution would match each purchased item with the latest price information at the time of purchase.

Here's the query used to create the table clean_item_data:

```
CREATE TABLE clean_item_data AS
SELECT * FROM (
SELECT * , ROW_NUMBER() OVER( PARTITION BY PRODUCT_ID, VENUE_ID ORDER BY
TIMESTAMP DESC) as row_number
FROM item_data)
WHERE ROW_NUMBER = 1;
```

2. Task 1

We need to find the tax-free buying and selling price of each product and multiply these by the product count for each product in each purchase then we need to calculate the conversion rate (price in euros / price in local currency) to be able to convert all values to euros.

Here's query to do that:

```
CREATE TABLE purchase_item_profit_data AS
SELECT
SUM(BASEPRICE/(1+VAT_PERCENTAGE/100)*COUNT) as ITEM_REVENUE,
SUM(COST_PER_UNIT*COUNT) as ITEM_COST,
COST_PER_UNIT_EUR/COST_PER_UNIT as EXCHANGE_RATE,
cid.PRODUCT_ID, cid.VENUE_ID, PURCHASE_ID
FROM clean_item_data as cid JOIN purchase_item_data as pid
ON cid.PRODUCT_ID = pid.PRODUCT_ID AND cid.VENUE_ID = pid.VENUE_ID
GROUP BY pid.PURCHASE_ID, cid.PRODUCT_ID
ORDER BY cid.PRODUCT_ID, PURCHASE_ID, cid.VENUE_ID;
```

From this table, let's calculate the profitability for each purchase. It's good to calculate and save multiple financial metrics to make future analysis, like the calculation of country or venue specific metrics more intuitive and simpler. The following query creates a table containing the revenue, COGS, gross margin, gross profit as well as total order size for each purchase.

```
CREATE TABLE purchase_profit_data AS
SELECT
(PURCHASE_REVENUE_EUR-PURCHASE_COGS_EUR)/PURCHASE_REVENUE_EUR AS
PURCHASE_GROSS_MARGIN,
PURCHASE_REVENUE_EUR-PURCHASE_COGS_EUR AS PURCHASE_GROSS_PROFIT_EUR, *
FROM (
        SELECT
        SUM(COUNT) AS ORDER_SIZE,
        SUM(ITEM_REVENUE)*EXCHANGE_RATE AS PURCHASE_REVENUE_EUR,
        SUM(ITEM_COST)*EXCHANGE_RATE AS PURCHASE_COGS_EUR,
        p.purchase_ID, p.VENUE_ID
        FROM purchase_data AS p JOIN purchase_item_profit_data AS i ON p.PURCHASE_ID
        = i.PURCHASE_ID
        GROUP BY p.PURCHASE_ID
);
```

This new table purchase_profit_data contains 237020 rows. Here’s what the table looks like:

	PURCHASE_GROSS_MARGIN	PURCHASE_GROSS_PROFIT_EUR	ORDER_SIZE	PURCHASE_REVENUE_EUR	PURCHASE_COGS_EUR	purchase_ID	VENUE_ID
1	0.14454800223561	4.93984821731748	16	34.17444821731748	29.2346	000023dc87	5069bf9875
2	0.33564969450102	19.18579546493821	14	57.1601755617865	37.97438009684829	0000890580	b2c886cd0a
3	0.20472252848218	4.92964601769912	14	24.07964601769912	19.15	0000e5eded	986fb70def
4	0.31318416236899	7.38254901960784	9	23.57254901960784	16.19	000167f209	ffda50ac8d
5	0.29665917602996	3.27305785123967	7	11.03305785123967	7.76	000196878a	9900b487b1
6	0.19927630495866	5.09686268912361	3	25.57686268912361	20.48	0001d62c16	a86797f9f1
7	0.25952606635071	2.28166666666667	5	8.79166666666667	6.51	0001fbf606	5ee6d50d1a
8	0.2254295532646	3.74857142857143	6	16.62857142857143	12.88	0002453da9	9eb13b4899
9	0.20818985574686	3.72833333333333	11	17.90833333333333	14.18	00028b6fd9	5ee6d50d1a
10	0.28991578947368	2.73740071260734	9	9.44205459653246	6.70465388392512	0002b500b8	15911591dc

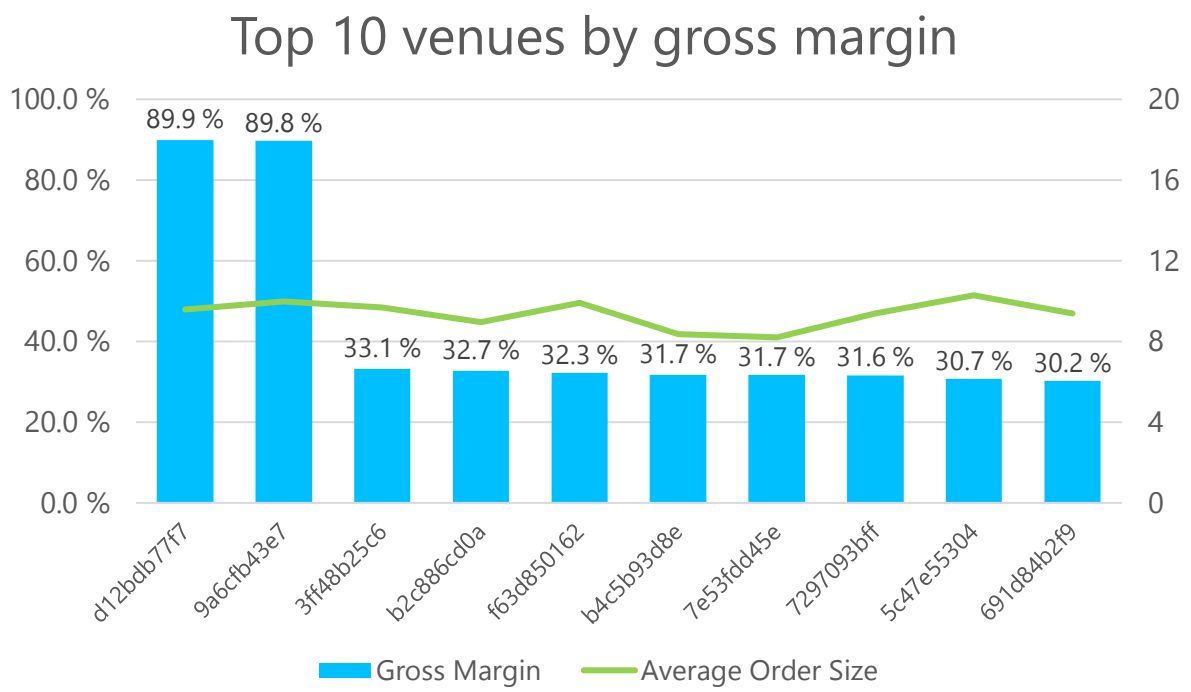
3. Task 2

In this section, I have included the SQL queries, Python scripts, visualizations and additional notes concerning each KPI. These can be found under their respective headers. I wrote a short python script to directly transfer the data through queries from the database to excel files. Before executing the queries in Python, I ran the queries in SQLiteStudio to ensure the integrity of each result.

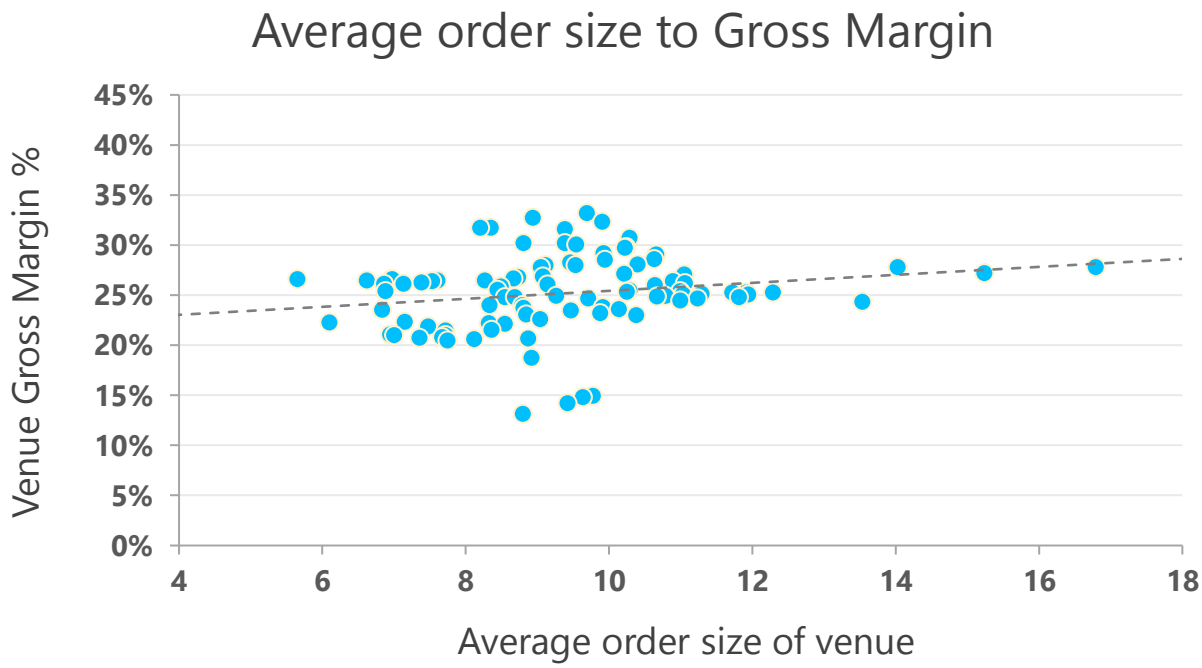
3.1 Top 10 Venues with the Highest Margins & Relationship between Average Order Size and Gross Margin

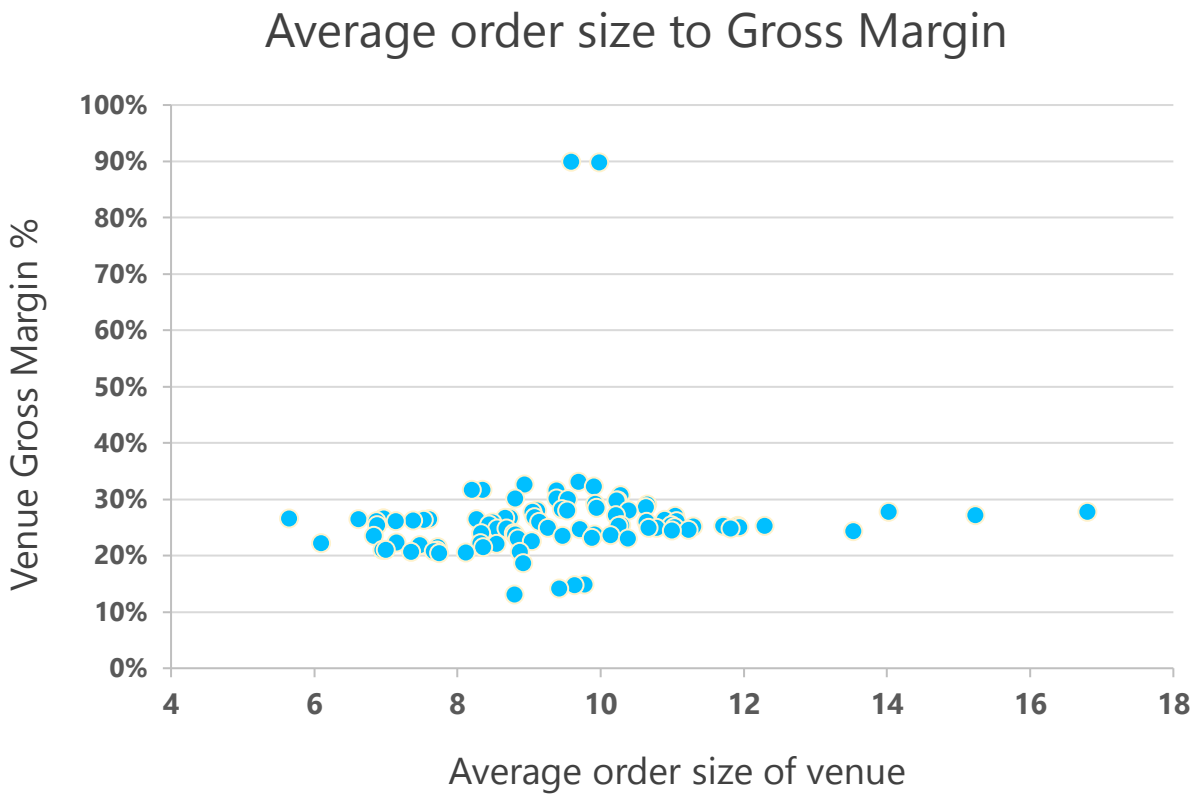
I used the purchase_profit_data table I just created to calculate the gross margin and average order sizes grouped by venue. I first defined venue revenue and venue cost of goods sold in the inner query to then calculate the venue gross margin. The results were then transferred to an excel file using the Python script down below.

```
import sqlite3
import pandas as pd
conn = sqlite3.connect("Wolt_BI0.db")

cursor = conn.cursor()

try:
    query_1 = pd.read_sql_query('''
        SELECT *, (VENUE_REVENUE-VENUE_COGS)/VENUE_REVENUE AS VENUE_GROSS_MARGIN
        FROM(
        SELECT VENUE_ID, SUM(PURCHASE_REVENUE_EUR) AS VENUE_REVENUE,
        SUM(PURCHASE_COGS_EUR) AS VENUE_COGS, AVG(ORDER_SIZE)
        FROM purchase_profit_data as ppd
        GROUP BY VENUE_ID
        )
        ORDER BY VENUE_GROSS_MARGIN DESC;
        ''', conn)
    conn.commit()
except:
    None

finally:
    query_1.to_excel('venue_profit.xlsx')
    conn.commit()
    conn.close()
```



To visualize the relationship between the average order size and gross margin of a venue I used all the datapoints. The visualization shows slight correlation between the two. The top 2 venues by gross margin seem strongly deviate from the general distribution of the datapoints. The first visualization shows the trend with the two top venues removed and the second one show the entire distribution of datapoints.

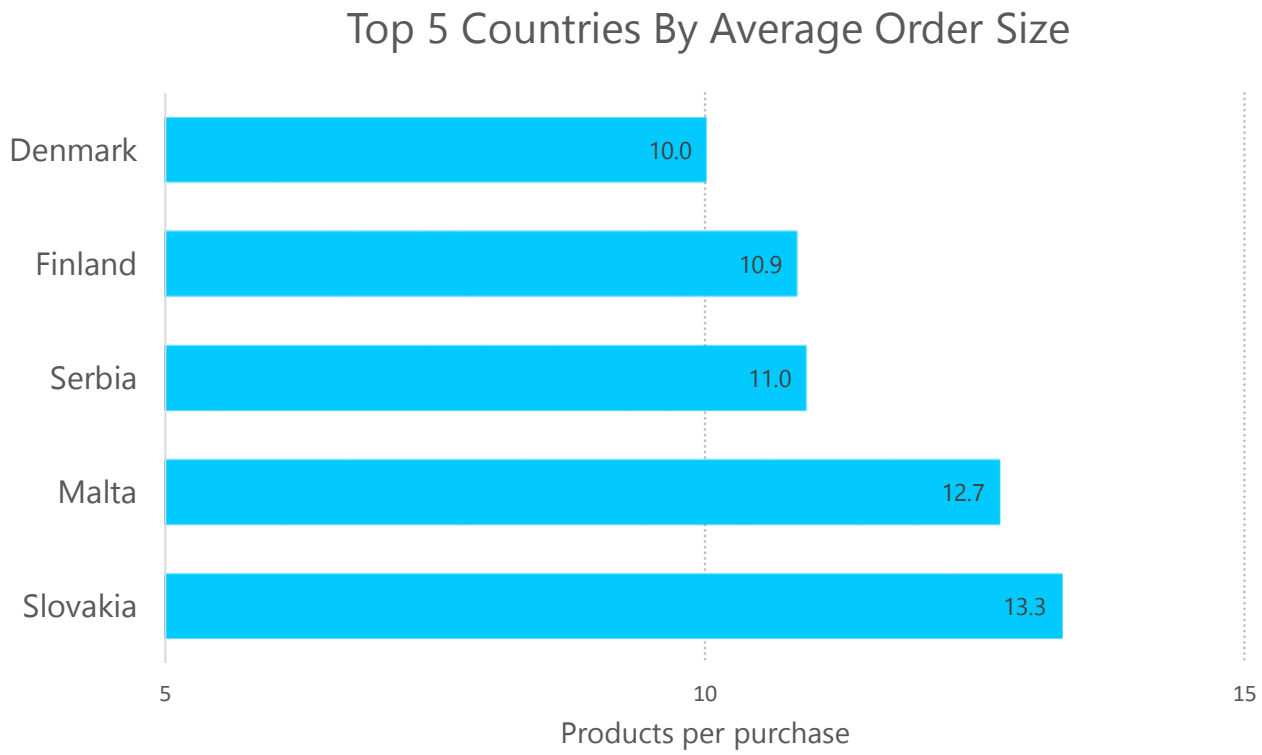



Average order size to Gross Margin
Venue Gross Margin %
100%
90%
80%
70%
60%
50%
40%
30%
20%
10%
0%
4
6
8
10
12
14
16
18
Average order size of venue

3.2 Top 5 Countries by Average Order Size

First, I queried the desired data and write it to an excel file through the same script as before. To calculate the average order size for each country I conveniently used the AVG function provided in SQL and group the results by country. Then I chose only the top 5 countries and transferred their data to excel.

```
import sqlite3
import pandas as pd
conn = sqlite3.connect("Wolt_BI0.db")
cursor = conn.cursor()

try:
    query_1 = pd.read_sql_query('''
SELECT COUNTRY, AVG(ORDER_SIZE) as AVG_ORDER_SIZE
FROM purchase_profit_data as ppd JOIN purchase_data as p ON p.PURCHASE_ID = ppd.PURCHASE_ID
GROUP BY COUNTRY
ORDER BY AVG_ORDER_SIZE DESC
LIMIT 5
;
        ''', conn)
    conn.commit()
except:
    None

finally:
    query_1.to_excel('Country_OrderSize.xlsx')
    conn.commit()
    conn.close()

    print("We're all done here, let's go home")
```

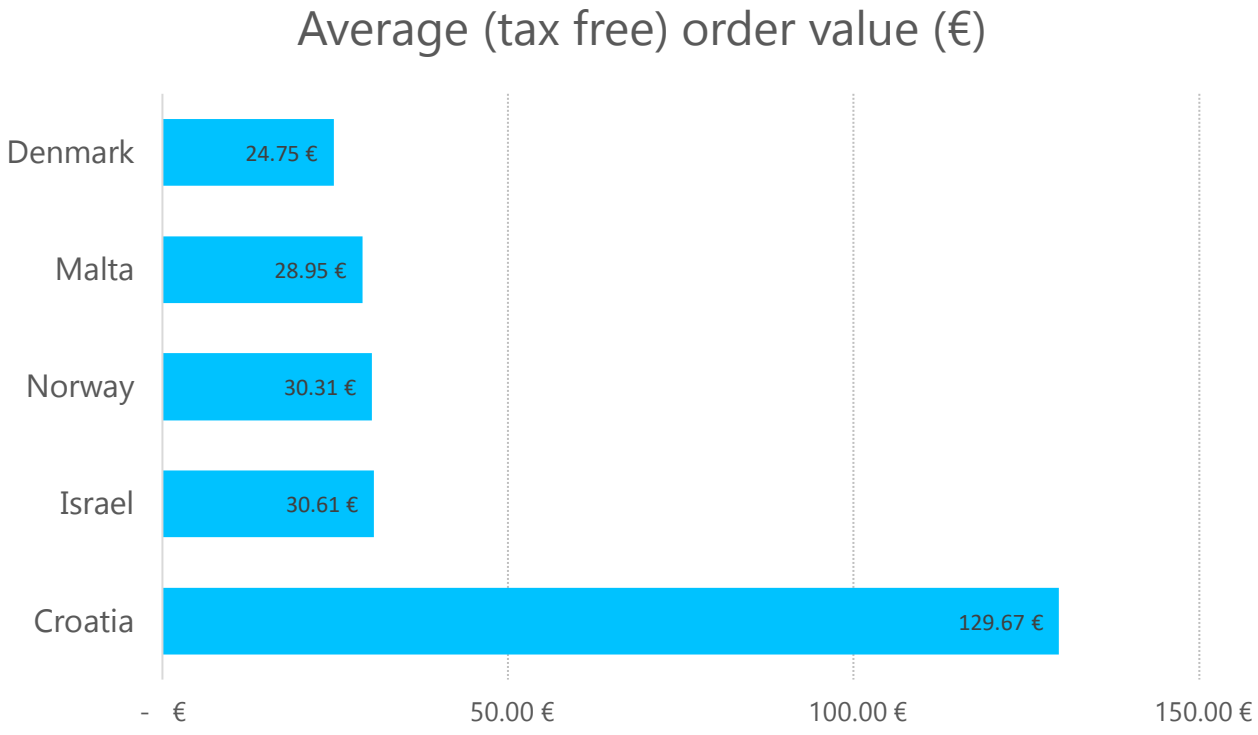


The top 2 countries have around 30% higher average order size than Denmark.

3.3 Top 5 Countries by Average Order Value

To calculate the average tax-free order value for each country I again used the AVG function on purchase revenues and grouped the results by country. Then I selected only the top 5 countries and transfer their data to an excel file.

```
import sqlite3
import pandas as pd
conn = sqlite3.connect("Wolt_BI0.db")

cursor = conn.cursor()

try:
    query_1 = pd.read_sql_query('''
        SELECT COUNTRY, AVG(PURCHASE_REVENUE_EUR) AS AVG_VALUE
        FROM purchase_profit_data as ppd JOIN purchase_data as p ON p.PURCHASE_ID = ppd.PURCHASE_ID
        GROUP BY COUNTRY
        ORDER BY AVG_VALUE DESC
        LIMIT 5
        ;
        ''', conn)
    conn.commit()
except:
    None

finally:
    query_1.to_excel('Country_OrderValue.xlsx')
    conn.commit()
    conn.close()
```

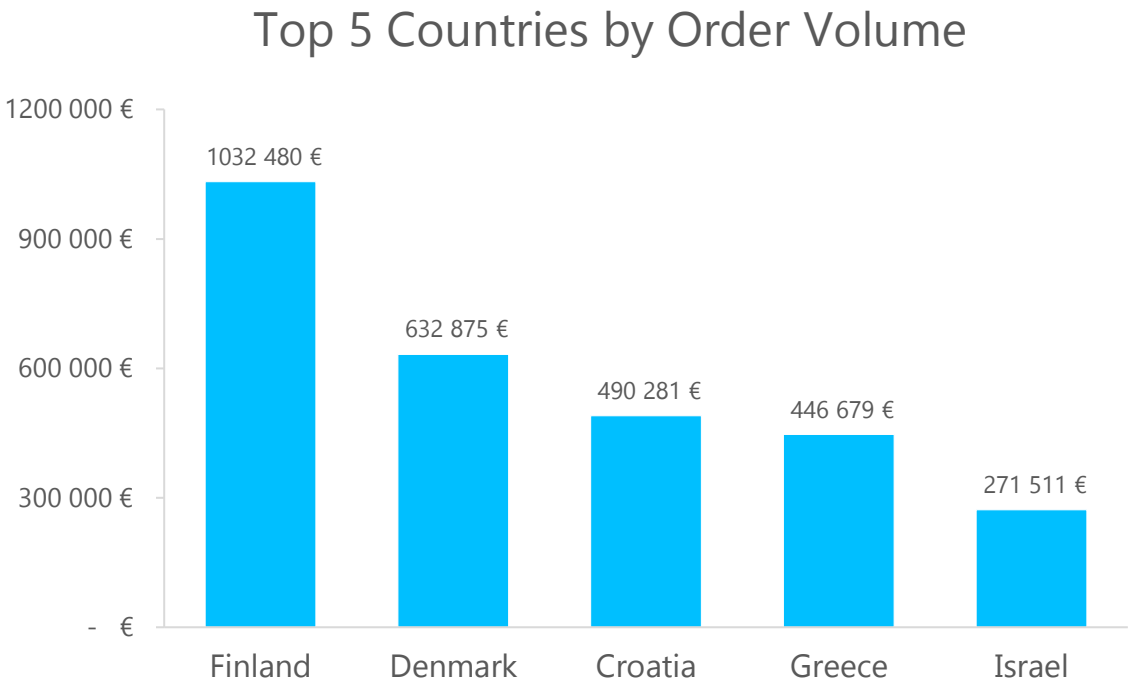


Croatia has more than 4 times the average order value than the second biggest country. On the contrary, Croatia only has 2 venues in the database so average order value might significantly be affected by the price level of these venues.

3.4 Top 5 Countries by Order Volume

To calculate order volume as total tax-free value of all orders, we simply need to join purchase_profit_data with purchase_data and sum all purchase revenues by country. Then we select only the top 5 countries and transfer their data to an excel file.

```python
import sqlite3
import pandas as pd
conn = sqlite3.connect("Wolt_BI0.db")

cursor = conn.cursor()

try:
    query_1 = pd.read_sql_query('''
        SELECT COUNTRY, SUM(PURCHASE_REVENUE_EUR) AS ORDER_VOLUME
        FROM purchase_profit_data as ppd JOIN purchase_data as p ON p.PURCHASE_ID = ppd.PURCHASE_ID
        GROUP BY COUNTRY
        ORDER BY ORDER_VOLUME DESC
        LIMIT 5
        ;
        ''', conn)
    conn.commit()
except:
    None

finally:
    query_1.to_excel('Country_OrderVolume.xlsx')
    conn.commit()
    conn.close()
```

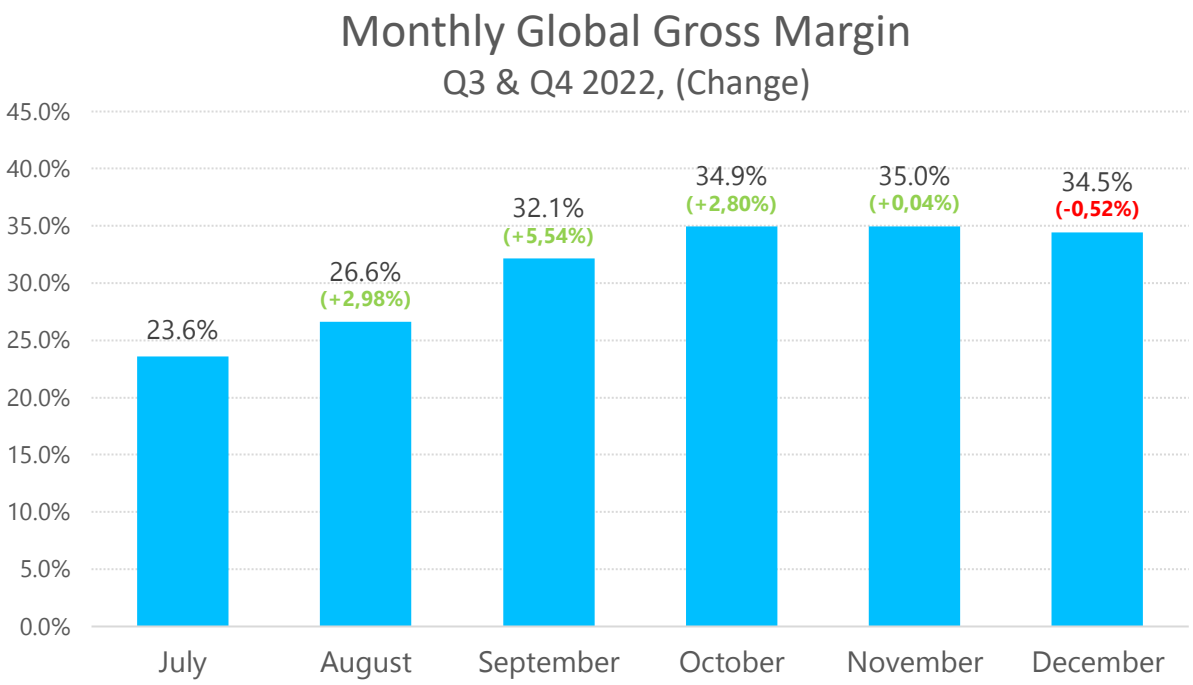


Finland is our biggest marketplace with a volume 1.6x larger than that of Denmark. 4 out of 5 biggest countries are in Europe.

3.5 Monthly Woltwide Margin

Again, I queried the desired data and wrote it to an excel file through the same script as before. To calculate gross margin for each month I needed to first calculate the profit and revenue of each month. After this I calculated the gross profit margin and fetched both the margin and its month to an excel file for visualization.

```
import sqlite3
import pandas as pd
conn = sqlite3.connect("Wolt_BI0.db")
cursor = conn.cursor()

try:
    query_1 = pd.read_sql_query('''
        SELECT MONTH_PROFIT/MONTH_REVENUE AS GROSS_MARGIN, *
        FROM (
        SELECT
        SUM(PURCHASE_GROSS_PROFIT_EUR) MONTH_PROFIT,
        SUM(PURCHASE_REVENUE_EUR) AS MONTH_REVENUE,
        strftime('%m', TIME_DELIVERED) AS MONTH
        FROM purchase_profit_data AS ppd JOIN purchase_data AS pd ON ppd.PURCHASE_ID = pd.PURCHASE_ID
        GROUP BY MONTH )
        ;
        ''', conn)
    conn.commit()
except:
    None

finally:
    query_1.to_excel('Wolt_Global_Margin_2022.xlsx')
    conn.commit()
    conn.close()
```



Global gross margin has been steadily growing through Q3 and the beginning of Q4, leveling out towards the end of the year.

4. Thoughts, assumptions and issues

4.1 Assumptions

I made some assumptions both about the validity of the given data, as well as the level of variability in the data, especially concerning the prices of products.

While creating the table clean_item_data, I assumed the latest currency conversion rates to be sufficiently accurate as to be applicable to all purchases regardless of timestamp . Closely relating to this, I also assumed the prices of items have stayed the same or changed insignificantly during the given period of time. I also assumed that Wolt purchases products in the same local currency as the sell them while.

I also made the assumption that there are no duplicates in certain tables in the data.
For example, if there were two purchase_item_data rows with the same PRODUCT_ID and PURCHASE_ID, these were both considered valid rows to be taken into account when making calculations. The correctness of this assumption is dependent on the protocol used for updating the data i.e. how modifications in the count of a product in a purchase are logged.

4.2 Issues and thoughts

I completed the queries using SQLiteStudio and SQLite3. Initially, I imported the dataset to the database using SQLStudio's import feature. This method seemed to somehow corrupt or alter the data, since my queries joining item_data and purchase_item_data on PRODUCT_ID returned zero rows. I tried dropping and re-importing the data quite a few times, but the problem persisted. After a few tries I imported the data to a database using the sqlite3 and pandas packages in Python. After the import by script, everything seemed to work fine and my queries didn't come up empty anymore.

It took quite a lot of thinking and querying to choose which assumptions to make, and which shortcuts and constraints to create when working with an unknown dataset. This was a great exercise to keep up my SQL and Python skills. It's rare to get to work on problems this interesting with such big datasets.