Customer Cohort and RFM Automation

MySQL, Power BI

Aaron Xie

Table of Contents

---

1. The Problem

Customer retention and churn rates are important for businesses to have sustainable revenues, especially for e-commerce businesses. The retention rate of e-commerce is not as obvious as those businesses that rely on subscriptions. Subscription businesses can easily find those who unsubscribe, but e-commerce businesses only have sales reports with a large amount of transactions. This difference raises the questions:

- How can the e-commerce business keep track of customer retention easily?
- How can they create the customer segmentation of the business for improving the retention rate?

- Can they use MySQL and/or Power BI to automate these processes?

The Goal

- Use SQL language to extract tables from the database to conduct cohort, trending, and RFM analysis.
- Connect Power BI to MySQL database and visualize the data analyses results on dashboards.
- Use the **stored procedure** function of **MySQ**L and **Power BI** to automate the cohort and trending analyses.
- Use the **Power Query Editor** and **DAX** language in **Power BI** to automate RFM analysis.
- This project is for exploring the probability of MySQL and Power BI, so don't use Python!

2. Data Preparation

Data ETL Pipline Design

Analysis	Extract	Transform	Load
Cohort	MySQL	MySQL	Power BI
RFM	MySQL	Power Query, DAX	Power BI

Import the data set

This project uses a Online Retail Data Set from the UCI Machine Learning Repository

Creating a table

```
DROP TABLE IF EXISTS project.sales_report;
CREATE TABLE project.sales_report
(`InvoiceNo` varchar(255),
`StockCode` varchar(255),
`Description` varchar(1000),
`Quantity` int,
`InvoiceDate` varchar(255),
`UnitPrice` double,
`CustomerID` varchar(255),
`Country` varchar(255));
```

The csv file is large, so we use LOAD DARA INFILE method to upload the data set. This method is much faster.

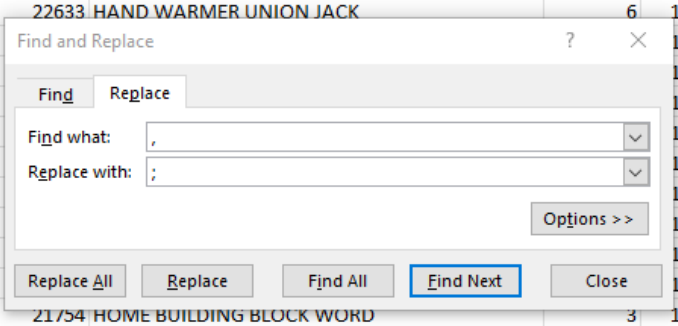
```
LOAD DATA INFILE 'C:/ProgramData/MySQL/MySQL Server 8.0/Uploads/Online Retail.csv'
INTO TABLE sales_report
FIELDS TERMINATED BY ','
LINES TERMINATED BY '\n'
IGNORE 1 LINES;
```

```
541909 row(s) affected Records: 541909  Deleted: 0  Skipped: 0  Warnings: 0                    4.141 sec
```

It took only 4 seconds to import a csv with 541K rows.

Note: to ensure MySQL can detect each column from csv file, I used Excel to replace comma with semicolon

	A	B	C	D	E	F	G	H	I
1	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	Customer	Country	
2	536365	85123A	WHITE HANGING HEART T-LIGHT HOLDER	6	12/1/2010 8:26	2.55	17850	United Kingdom	
3	536365	71053	WHITE METAL LANTERN	6	12/1/2010 8:26	3.39	17850	United Kingdom	
4	536365	84406B	CREAM CUPID HEARTS COAT HANGER	8	12/1/2010 8:26	2.75	17850	United Kingdom	
5	536365	84029G	KNITTED UNION FLAG HOT WATER BOTTLE	6	12/1/2010 8:26	3.39	17850	United Kingdom	
6	536365	84029E	RED WOOLLY HOTTIE WHITE HEART.	6	12/1/2010 8:26	3.39	17850	United Kingdom	
7	536365	22752	SET 7 BABUSHKA NESTING BOXES	2	12/1/2010 8:26	7.65	17850	United Kingdom	
8	536365	21730	GLASS STAR FROSTED T-LIGHT HOLDER	6	12/1/2010 8:26	4.25	17850	United Kingdom	
9	536366	22633	HAND WARMER UNION JACK	6	12/1/2010 8:28	1.85	17850	United Kingdom	
10	536366				12/1/2010 8:28	1.85	17850	United Kingdom	
11	536367				12/1/2010 8:34	1.69	13047	United Kingdom	
12	536367				12/1/2010 8:34	2.1	13047	United Kingdom	
13	536367				12/1/2010 8:34	2.1	13047	United Kingdom	
14	536367				12/1/2010 8:34	3.75	13047	United Kingdom	
15	536367				12/1/2010 8:34	1.65	13047	United Kingdom	
16	536367				12/1/2010 8:34	4.25	13047	United Kingdom	
17	536367				12/1/2010 8:34	4.95	13047	United Kingdom	
18	536367				12/1/2010 8:34	9.95	13047	United Kingdom	
19	536367	21754	HOME BUILDING BLOCK WORD	3	12/1/2010 8:34	5.95	13047	United Kingdom	
20	536367	21755	LOVE BUILDING BLOCK WORD	3	12/1/2010 8:34	5.95	13047	United Kingdom	
21	536367	21777	RECIPE BOX WITH METAL HEART	4	12/1/2010 8:34	7.95	13047	United Kingdom	



Go back to the top

3. Data Processing

Data set Basic Info

```
DESCRIBE sales_report
```

	Field	Type	Null	Key	Default	Extra
▶	InvoiceNo	varchar(255)	YES		NULL	
	StockCode	varchar(255)	YES		NULL	
	Description	varchar(1000)	YES		NULL	
	Quantity	int	YES		NULL	
	InvoiceDate	varchar(255)	YES		NULL	
	UnitPrice	double	YES		NULL	
	CustomerID	varchar(255)	YES		NULL	
	Country	varchar(255)	YES		NULL	

As a sales report, the invoice number should be unique. Let's check.

```
SELECT
    InvoiceNo,
    COUNT(InvoiceNo)
FROM sales_report
GROUP BY InvoiceNo
HAVING COUNT(InvoiceNo) > 1;
```

	InvoiceNo	COUNT(InvoiceNo)
▶	536365	7
	536366	2
	536367	12
	536368	4
	536370	20

Apparently the InvoiceNo is not the primary key as I thought.

Removing duplicate rows

Creating a view to store

```
CREATE VIEW report_unique
AS
SELECT DISTINCT *
FROM sales_report;
```

Let's check

```
SELECT
    'W/ Duplicates' AS 'table',
    COUNT(*)
FROM sales_report
UNION
SELECT
    'W/O Duplicates' AS 'table',
    COUNT(*)
FROM report_unique
```

	table	COUNT(*)
▶	W/ Duplicates	541909
	W/O Duplicates	536641

We removed around 5000 duplicates!

Dealing with missing values

Checking missing values

```
SELECT
    *
FROM report_unique
WHERE COALESCE(Quantity, InvoiceDate, UnitPrice, CustomerID, Country) IS NULL
```

Great! No missing value detected!

For practice. What if there are missing values?

- drop the missing rows
- use imputation

Example: creating another view with mean imputation for UnitPrice

```
CREATE VIEW report_imputed
AS
SELECT
    *,
    COALESCE(UnitPrice, AVG(UnitPirce)) AS UnitPrice
FROM report_unique
```

Checking whether the records of the last month is complete.

```
SELECT
    *
FROM sales_report
WHERE invoicedate LIKE '%12/%/%2011%'
ORDER BY invoicedate DESC;
```

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country
▶	C581490	23144	ZINC T-LIGHT HOLDER STARS SMALL	-11	12/9/2011 9:57	0.83	14397	United Kingdom
	C581490	22178	VICTORIAN GLASS HANGING T-LIGHT	-12	12/9/2011 9:57	1.95	14397	United Kingdom
	581489	22182	CAKE STAND VICTORIAN FILIGREE SMALL	24	12/9/2011 9:46	1.25	16954	United Kingdom
	581489	22061	LARGE CAKE STAND HANGING STRAWBERY	48	12/9/2011 9:46	2.95	16954	United Kingdom
	581488	23118	PARISIENNE JEWELLERY DRAWER	16	12/9/2011 9:45	6.65	17428	United Kingdom
	581488	22179	S PARISIENNE JEWELLERY DRAWER	24	12/9/2011 9:45	5.95	17428	United Kingdom
	581488	23111	PARISIENNE SEWING BOX	16	12/9/2011 9:45	10.4	17428	United Kingdom

No, the transactions after 12/09/2011 were not recorded. Need to delete the transctions of this month for trending analysis.

Go back to the top

Detecting Anormal Values/Outliers

The data set has two numeric variables that need attention: Quantity and UnitPrice.

```
SELECT
    'Quantity' AS 'Column',
    MAX(Quantity) AS MAX,
    MIN(Quantity) AS MIN,
    round(AVG(Quantity),2) AS AVG,
    round(STDDEV(Quantity),2) AS STD
FROM report_unique
UNION
SELECT
    'UnitPrice' AS 'Column',
    MAX(UnitPrice) AS MAX,
    MIN(UnitPrice) AS MIN,
    round(AVG(UnitPrice),2) AS AVG,
    round(STDDEV(UnitPrice),2) STD
FROM report_unique
```

	Column	MAX	MIN	AVG	STD
▶	Quantity	80995	-80995	9.62	219.13
	UnitPrice	38970	-11062.06	4.63	97.23

The result indicates some anormalies:

- The values include negative numbers; nagetive quantity may indicate that the order was returned, while negative price should be an error.
- The maximum values are too large compared to the average and standard deviation

Checking negative price

```
SELECT
    InvoiceNo,
    UnitPrice
FROM report_unique
WHERE UnitPrice < 0
```

	InvoiceNo	UnitPrice
▶	A563186	-11062.06
	A563187	-11062.06

Only two rows have negative price; need to delete them

```
SET SQL_SAFE_UPDATES = 0;
DELETE FROM sales_report
WHERE UnitPrice < 0;
SET SQL_SAFE_UPDATES = 1;
```

Note: when SQL_SAFE_UPDATES = 1, cannot delete rows based on non-primary key

Checking negative quantity

```
SELECT
    InvoiceNo,
    Quantity,
    UnitPrice
FROM report_unique
WHERE Quantity < 0
LIMIT 1000, 10
```

InvoiceNo	Quantity	UnitPrice
C540555	-1	7.95
540558	-29	0
C540559	-4	3.75
540560	-14	0
540564	-2600	0
C540634	-3	4.95
C540634	-12	1.69
C540634	-4	6.75
540637	-5	0

This sample shows that negative quantity may be associated with either invoice number starting with "c" or 0 unit price. According to UCI website, c indicates canceled orders. Let's check.

```
SELECT
    invoiceno,
    Quantity,
    UnitPrice
FROM report_unique
WHERE Quantity < 0
AND UnitPrice > 0
AND invoiceno NOT LIKE "C%"
```

invoiceno	Quantity	UnitPrice

The empty results shows that negative quantity must be an canceled order or have 0 unit price. We can ignore all these orders based on our purpose.

Creating another view with the only data we need.

```
CREATE VIEW report_positive
AS
SELECT
    *
FROM report_unique
WHERE Quantity > 0
AND UnitPrice > 0
```

Handling Outliers

Using Boxplot to better visualize outliers (Opitional because this is no Python project)

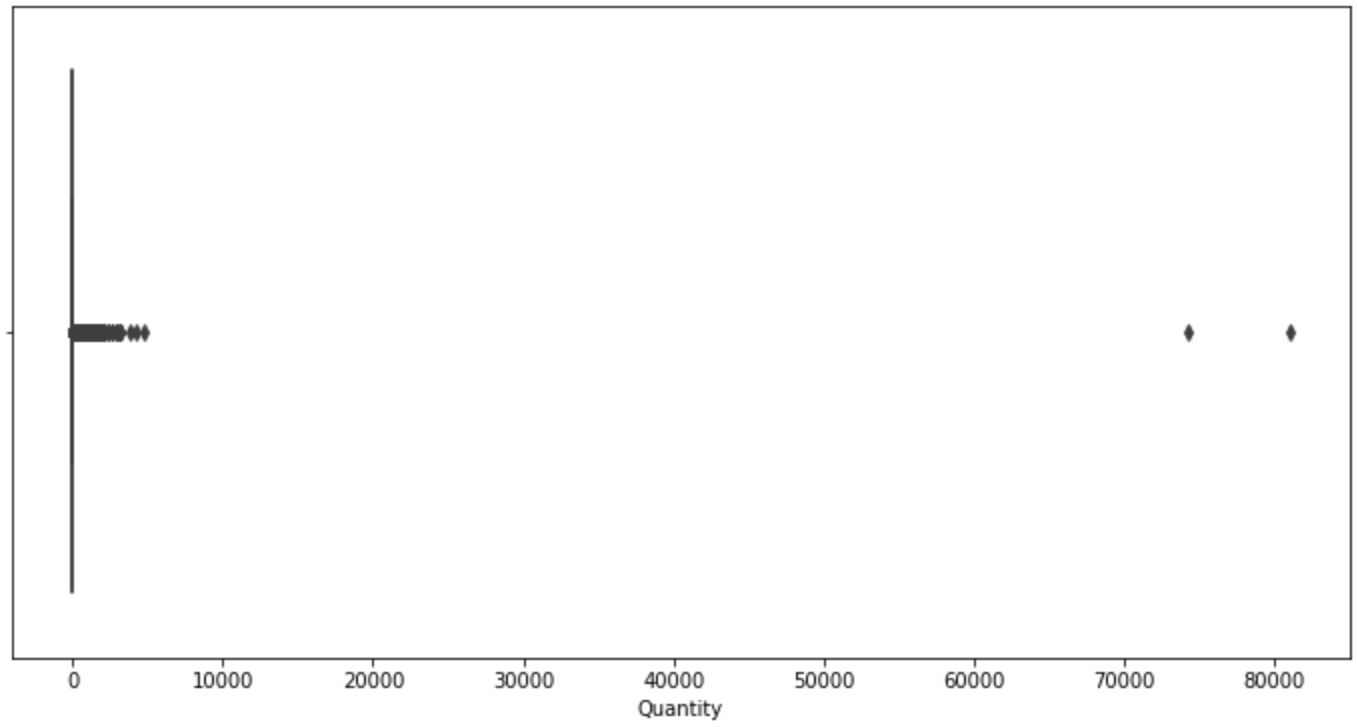
In [23]:
```
import sqlalchemy
import pandas as pd
engine = sqlalchemy.create_engine('mysql://root:ftgbvnth10@localhost/project')
engine.connect()
df = pd.read_sql('select * from report_positive', con = engine)
```

In [ ]:
```
import seaborn as sns
import matplotlib.pyplot as plt
plt.rcParams['figure.figsize'] = (12,6)
```

Quantity outliers

In [29]:
```
sns.boxplot(x = df.Quantity)
```

Out[29]:
```
<AxesSubplot: xlabel='Quantity'>
```



Checking the two extreme outliers

```
SELECT *
FROM project.report_positive
WHERE Quantity > 70000;
```

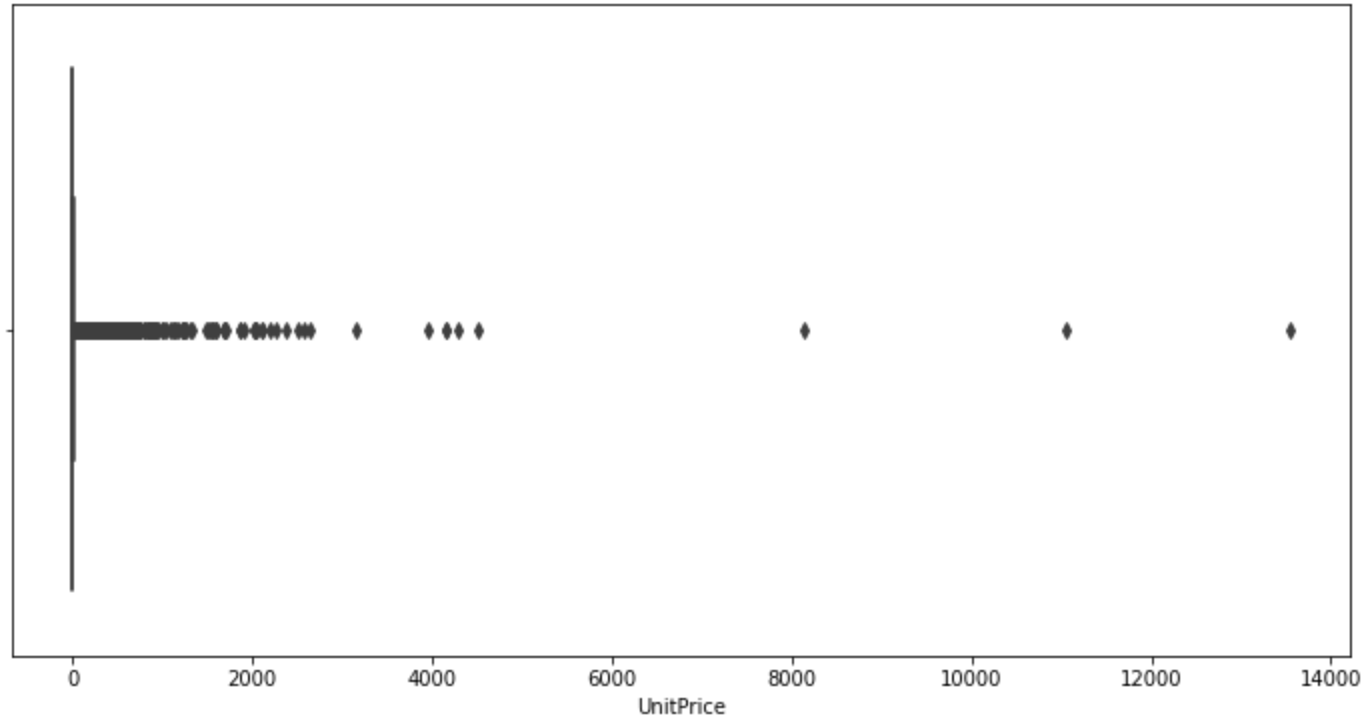
	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country
▶	541431	23166	MEDIUM CERAMIC TOP STORAGE JAR	74215	1/18/2011 10:01	1.04	12346	United Kingdom
	581483	23843	PAPER CRAFT ; LITTLE BIRDIE	80995	12/9/2011 9:15	2.08	16446	United Kingdom

Although the business has many wholesaler customers according to the UCI website, these two quantity values are still too large for "storage jar" and "paper craft." It does not make sense for wholesalers to store this amount of products in their warehouse.

UnitPrice outliers

In [30]:
```
sns.boxplot(x = df.UnitPrice)
```

Out[30]:
```
<AxesSubplot: xlabel='UnitPrice'>
```



Checking outliers

```
SELECT *
FROM project.report_positive
WHERE UnitPrice > 7000;
```

InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country
537632	AMAZONFEE	AMAZON FEE	1	12/7/2010 15:08	13541.33		United Kingdom
551697	POST	POSTAGE	1	5/3/2011 13:46	8142.75	16029	United Kingdom
560373	M	Manual	1	7/18/2011 12:30	4287.63		United Kingdom
562955	DOT	DOTCOM POSTAGE	1	8/11/2011 10:14	4505.17		United Kingdom
A563185	B	Adjust bad debt	1	8/12/2011 14:50	11062.06		United Kingdom
569382	M	Manual	1	10/3/2011 16:44	3155.95	15502	United Kingdom
571751	M	Manual	1	10/19/2011 11:18	3949.32	12744	Singapore
573077	M	Manual	1	10/27/2011 14:13	4161.06	12536	France
573080	M	Manual	1	10/27/2011 14:20	4161.06	12536	France

Based on the description, although these values are large, they can still make sense because they are not the unit price of gift products.

When we see these anomalies in a data set, the best practice is to talk to the data set creators or related people. Now since I don't have this resource, I choose **to keep the UnitPrice values and to replace the two outliers in Quantity with the average** because:

- Outliers are not related to cohort analysis.
- This data set is huge and outliers won't have too much negative impact on the result.
- Eliminating outliers may omit some information.

Creating another view for cohort analysis

```
CREATE VIEW report_cohort
AS
SELECT
    InvoiceNo,
    StockCode,
    Description,
    IF(Quantity > 70000, 10, Quantity) AS Quantity,
```

```
    DATE_FORMAT(STR_TO_DATE(InvoiceDate, '%m/%d/%Y %H:%i'), '%Y-%m-01') AS
InvoiceDate,
    UnitPrice,
    CustomerID,
    LEFT(Country, CHAR_LENGTH(Country) - 1) AS Country
FROM report_positive
WHERE CustomerID <> '' AND InvoiceDate NOT LIKE '%12/%/%2011%'
```

InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country
536365	85123A	WHITE HANGING HEART T-LIGHT HOLDER	6	12/2010	2.55	17850	United Kingdom
536365	71053	WHITE METAL LANTERN	6	12/2010	3.39	17850	United Kingdom
536365	84406B	CREAM CUPID HEARTS COAT HANGER	8	12/2010	2.75	17850	United Kingdom
536365	84029G	KNITTED UNION FLAG HOT WATER BOTTLE	6	12/2010	3.39	17850	United Kingdom
536365	84029E	RED WOOLLY HOTTIE WHITE HEART.	6	12/2010	3.39	17850	United Kingdom
536365	22752	SET 7 BABUSHKA NESTING BOXES	2	12/2010	7.65	17850	United Kingdom
536365	21730	GLASS STAR FROSTED T-LIGHT HOLDER	6	12/2010	4.25	17850	United Kingdom

Changes made:

- Replaced outliers with the average 10.
- Cast the 'InvoiceDate' from VARCHAR data type into DATE data type; only kept the month and year.
- Removed rows with empty 'CustomerID'
- Removed the non-ascii white space, which can't be removed by TRIM(), at the last character of the country column.
- Removed the transactions of the last month 12/2011.

Go back to the top

4. Cohort Analysis

Step 1: get the cohort month (the month the customers first joined)

```
SELECT
    DISTINCT CustomerID,
    InvoiceDate,
    MIN(InvoiceDate) OVER(PARTITION BY CustomerID) AS CorhortMonth
FROM report
ORDER BY CustomerID, InvoiceDate
```

CustomerID	InvoiceDate	CorhortMonth
12346	01/2011	01/2011
12347	01/2011	01/2011
12347	04/2011	01/2011
12347	06/2011	01/2011
12347	08/2011	01/2011
12347	10/2011	01/2011
12347	12/2010	01/2011
12347	12/2011	01/2011

Step 2: get the cohort index (The time difference between invoice date and cohort month)

```
WITH cte AS
(
SELECT
    DISTINCT CustomerID,
    InvoiceDate,
    MIN(InvoiceDate) OVER(PARTITION BY CustomerID) AS CohortMonth
```

```
FROM report
ORDER BY CustomerID, InvoiceDate
)

SELECT
    *,
    TIMESTAMPDIFF(MONTH, CohortMonth, InvoiceDate) AS CohortIndex
FROM cte
```

CustomerID	InvoiceDate	CohortMonth	CohortIndex
12346	2011-01-01	2011-01-01	0
12347	2011-01-01	2010-12-01	1
12347	2011-04-01	2010-12-01	4
12347	2010-12-01	2010-12-01	0
12347	2011-12-01	2010-12-01	12
12347	2011-10-01	2010-12-01	10

Step 3: create a cohort pivot table

```
CREATE VIEW cohort_pivot
AS
WITH cte AS
(
SELECT
    DISTINCT CustomerID,
    InvoiceDate,
    MIN(InvoiceDate) OVER(PARTITION BY CustomerID) AS CohortMonth
FROM report
ORDER BY CustomerID, InvoiceDate
),
cte2 AS
(
SELECT
    *,
    TIMESTAMPDIFF(MONTH, CohortMonth, InvoiceDate) AS CohortIndex
FROM cte
)
SELECT
    DISTINCT CohortMonth,
    SUM(CohortIndex = 0) AS join_mth,
    SUM(CohortIndex = 1) AS mth01,
    SUM(CohortIndex = 2) AS mth02,
    SUM(CohortIndex = 3) AS mth03,
    SUM(CohortIndex = 4) AS mth04,
    SUM(CohortIndex = 5) AS mth05,
    SUM(CohortIndex = 6) AS mth06,
    SUM(CohortIndex = 7) AS mth07,
    SUM(CohortIndex = 8) AS mth08,
    SUM(CohortIndex = 9) AS mth09,
    SUM(CohortIndex = 10) AS mth10,
    SUM(CohortIndex = 11) AS mth11
FROM cte2
GROUP BY CohortMonth
ORDER BY CohortMonth
```

CohortMonth	join_mth	mth01	mth02	mth03	mth04	mth05	mth06	mth07	mth08	mth09	mth10	mth11
2010-12-01	885	324	286	340	321	352	321	309	313	350	331	445
2011-01-01	417	92	111	96	134	120	103	101	125	136	152	0
2011-02-01	380	71	71	108	103	94	96	106	94	116	0	0
2011-03-01	452	68	114	90	101	76	121	104	126	0	0	0
2011-04-01	300	64	61	63	59	68	65	78	0	0	0	0
2011-05-01	284	54	49	49	59	66	75	0	0	0	0	0
2011-06-01	242	42	38	64	56	81	0	0	0	0	0	0
2011-07-01	188	34	39	42	51	0	0	0	0	0	0	0
2011-08-01	169	35	42	41	0	0	0	0	0	0	0	0
2011-09-01	299	70	90	0	0	0	0	0	0	0	0	0
2011-10-01	358	86	0	0	0	0	0	0	0	0	0	0
2011-11-01	323	0	0	0	0	0	0	0	0	0	0	0

The downside of using mysql to get pivot table is that it does not have a PIVOT command, so it is not very scalable when we need to calculate the retention rate for a very long period. However, calculating monthly retention rate for a year would not be a problem.

Step 4: get retention rates (in percentage).

```
SELECT
    CohortMonth,
    join_mth/join_mth*100 AS join_mth,
    mth01/join_mth*100 AS mth01,
    mth02/join_mth*100 AS mth02,
    mth03/join_mth*100 AS mth03,
    mth04/join_mth*100 AS mth04,
    mth05/join_mth*100 AS mth05,
    mth06/join_mth*100 AS mth06,
    mth07/join_mth*100 AS mth07,
    mth08/join_mth*100 AS mth08,
    mth09/join_mth*100 AS mth09,
    mth10/join_mth*100 AS mth10,
    mth11/join_mth*100 AS mth11
FROM cohort_pivot
```

CohortMonth	join_mth	mth01	mth02	mth03	mth04	mth05	mth06	mth07	mth08	mth09	mth10	mth11
2010-12-01	100.0000	36.6102	32.3164	38.4181	36.2712	39.7740	36.2712	34.9153	35.3672	39.5480	37.4011	50.2825
2011-01-01	100.0000	22.0624	26.6187	23.0216	32.1343	28.7770	24.7002	24.2206	29.9760	32.6139	36.4508	0.0000
2011-02-01	100.0000	18.6842	18.6842	28.4211	27.1053	24.7368	25.2632	27.8947	24.7368	30.5263	0.0000	0.0000
2011-03-01	100.0000	15.0442	25.2212	19.9115	22.3451	16.8142	26.7699	23.0088	27.8761	0.0000	0.0000	0.0000
2011-04-01	100.0000	21.3333	20.3333	21.0000	19.6667	22.6667	21.6667	26.0000	0.0000	0.0000	0.0000	0.0000
2011-05-01	100.0000	19.0141	17.2535	17.2535	20.7746	23.2394	26.4085	0.0000	0.0000	0.0000	0.0000	0.0000
2011-06-01	100.0000	17.3554	15.7025	26.4463	23.1405	33.4711	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
2011-07-01	100.0000	18.0851	20.7447	22.3404	27.1277	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
2011-08-01	100.0000	20.7101	24.8521	24.2604	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
2011-09-01	100.0000	23.4114	30.1003	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
2011-10-01	100.0000	24.0223	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
2011-11-01	100.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000

We get a table of total retention rate.
To automate this calculation, I created a stored procedure without parameter called all_retention().

Now, what if we want to see the retention of each country?

Cohort Analysis by Country

Creating a stored procedure with IN parameter to achieve that; also adding WHERE clause to the first cte.

```
DELIMITER $$
USE `project`$$
```

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `country_retention`(IN country_input
VARCHAR(30))
BEGIN
DROP TABLE IF EXISTS retention;
CREATE TABLE retention
WITH cte AS
(
SELECT
    DISTINCT CustomerID,
    InvoiceDate,
    MIN(InvoiceDate) OVER(PARTITION BY CustomerID) AS CohortMonth
FROM report
WHERE Country = country_input # The key to change to the country we want
),
cte2 AS
(
SELECT
    *,
    TIMESTAMPDIFF(MONTH, CohortMonth, InvoiceDate) AS CohortIndex
FROM cte
),
cte3 AS
(
SELECT
    DISTINCT CohortMonth,
    SUM(CohortIndex = 0) AS join_mth,
    SUM(CohortIndex = 1) AS mth01,
    SUM(CohortIndex = 2) AS mth02,
    SUM(CohortIndex = 3) AS mth03,
    SUM(CohortIndex = 4) AS mth04,
    SUM(CohortIndex = 5) AS mth05,
    SUM(CohortIndex = 6) AS mth06,
    SUM(CohortIndex = 7) AS mth07,
    SUM(CohortIndex = 8) AS mth08,
    SUM(CohortIndex = 9) AS mth09,
    SUM(CohortIndex = 10) AS mth10,
    SUM(CohortIndex = 11) AS mth11
FROM cte2
GROUP BY CohortMonth
ORDER BY CohortMonth
)
SELECT
    CohortMonth,
    join_mth/join_mth*100 AS join_mth,
    mth01/join_mth*100 AS mth01,
    mth02/join_mth*100 AS mth02,
    mth03/join_mth*100 AS mth03,
    mth04/join_mth*100 AS mth04,
    mth05/join_mth*100 AS mth05,
    mth06/join_mth*100 AS mth06,
    mth07/join_mth*100 AS mth07,
    mth08/join_mth*100 AS mth08,
    mth09/join_mth*100 AS mth09,
    mth10/join_mth*100 AS mth10,
    mth11/join_mth*100 AS mth11
FROM cte3
;
END$$
```

```
DELIMITER ;
;
```

With this procedure, every time I run:

```
CALL country_retention(<country name>);
```

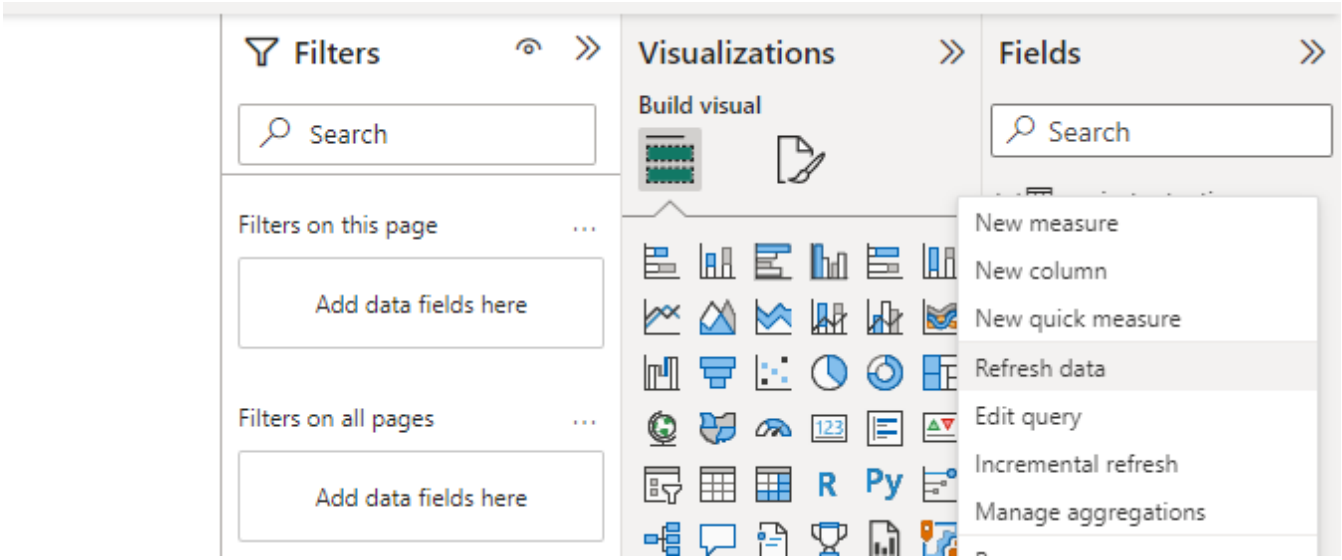
That country's retention rate will be recorded to "retention" table.

For example:

```
CALL country_retention('United Kingdom');
SELECT * FROM retention;
```

CohortMonth	join_mth	mth01	mth02	mth03	mth04	mth05	mth06	mth07	mth08	mth09	mth10	mth11	mth12
2010-12-01	100.0000	35.4601	32.2699	37.3006	35.9509	39.6319	35.7055	34.1104	35.4601	39.8773	36.6871	49.6933	26.7485
2011-01-01	100.0000	21.2291	25.9777	23.4637	33.2402	27.6536	25.1397	24.3017	30.1676	32.6816	35.4749	12.0112	0.0000
2011-02-01	100.0000	18.8235	19.4118	28.5294	28.8235	25.2941	25.5882	28.2353	26.4706	30.5882	7.3529	0.0000	0.0000
2011-03-01	100.0000	15.2745	26.0143	19.8091	22.4344	16.4678	26.4916	22.9117	28.4010	9.0692	0.0000	0.0000	0.0000
2011-04-01	100.0000	20.9386	20.2166	21.6606	20.2166	22.0217	22.0217	26.3538	7.2202	0.0000	0.0000	0.0000	0.0000
2011-05-01	100.0000	18.7500	17.1875	17.1875	20.7031	22.6563	26.5625	8.9844	0.0000	0.0000	0.0000	0.0000	0.0000
2011-06-01	100.0000	17.7570	14.4860	23.8318	23.8318	32.2430	9.8131	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
2011-07-01	100.0000	17.7515	19.5266	23.0769	27.8107	10.6509	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
2011-08-01	100.0000	22.6950	22.6950	24.1135	12.0567	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
2011-09-01	100.0000	22.8261	30.0725	11.5942	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
2011-10-01	100.0000	24.3827	11.1111	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
2011-11-01	100.0000	11.7845	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
2011-12-01	100.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000

The reason to store the output into retention table is that I could update the visualization on Power BI by simply refresh the data.



The retention heatmap

CohortMonth	mth01	mth02	mth03	mth04	mth05	mth06	mth07	mth08	mth09	mth10	mth11
2010-12-01	35.46	32.27	37.30	35.95	39.63	35.71	34.11	35.46	39.88	36.69	49.69
2011-01-01	21.23	25.98	23.46	33.24	27.65	25.14	24.30	30.17	32.68	35.47	0.00
2011-02-01	18.82	19.41	28.53	28.82	25.29	25.59	28.24	26.47	30.59	0.00	0.00
2011-03-01	15.27	26.01	19.81	22.43	16.47	26.49	22.91	28.40	0.00	0.00	0.00
2011-04-01	20.94	20.22	21.66	20.22	22.02	22.02	26.35	0.00	0.00	0.00	0.00
2011-05-01	18.75	17.19	17.19	20.70	22.66	26.56	0.00	0.00	0.00	0.00	0.00
2011-06-01	17.76	14.49	23.83	23.83	32.24	0.00	0.00	0.00	0.00	0.00	0.00
2011-07-01	17.75	19.53	23.08	27.81	0.00	0.00	0.00	0.00	0.00	0.00	0.00
2011-08-01	22.70	22.70	24.11	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
2011-09-01	22.83	30.07	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
2011-10-01	24.38	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
2011-11-01	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00

Go back to the top

5. Trending Analysis

Creating another VIEW called report_clean for this purpose

```
DROP VIEW IF EXISTS report_clean;
CREATE VIEW report_clean
AS
SELECT
    InvoiceNo,
    StockCode,
    Description,
    IF(Quantity > 70000, 10, Quantity) AS Quantity,
    DATE_FORMAT(STR_TO_DATE(InvoiceDate, '%m/%d/%Y %H:%i'), '%Y-%m-01') AS
InvoiceDate,
    UnitPrice,
    ROUND(IF(Quantity > 70000, 10, Quantity)*UnitPrice,2) AS TotalRevenue,
    CustomerID,
    LEFT(Country, CHAR_LENGTH(Country) - 1) AS Country
FROM report_positive
```

Creating a time series table for trending analysis.
Same as the retention rate, creating two stored procedures to save the result to the table "time_series".

- country_series(): getting time series table for each country.

```
DELIMITER $$
USE `project`$$
CREATE PROCEDURE `country_series` (IN country_input VARCHAR(30))
BEGIN
DROP TABLE IF EXISTS time_series;
CREATE TABLE time_series
WITH cte AS
(
```

```
SELECT
    DISTINCT CustomerID,
    MIN(InvoiceDate) AS CohortMonth
FROM report
WHERE Country = country_input
GROUP BY CustomerID
),
cte2 AS
(
SELECT
    COUNT(CustomerID) AS NewUsers,
    CohortMonth
FROM cte
GROUP BY CohortMonth
)
SELECT
    InvoiceDate,
    COUNT(*) AS TotalOrders,
    ROUND(SUM(TotalRevenue),2) AS TotalRevenue,
    NewUsers,
    Country
FROM report_clean
JOIN cte2
ON InvoiceDate = CohortMonth AND Country = country_input
GROUP BY InvoiceDate;
END$$
DELIMITER ;
```

Running:

```
CALL country_series('United Kingdom');
```

InvoiceDate	TotalOrders	TotalRevenue	NewUsers	Country
2010-12-01	38738	746082.22	815	United Kingdom
2011-01-01	30923	482802.63	358	United Kingdom
2011-02-01	24629	428986.62	340	United Kingdom
2011-03-01	32308	584810.78	419	United Kingdom
2011-04-01	27067	475677.63	277	United Kingdom
2011-05-01	32614	638104.89	256	United Kingdom
2011-06-01	32130	618345.53	214	United Kingdom
2011-07-01	34961	600919.07	169	United Kingdom
2011-08-01	30259	606784.93	141	United Kingdom
2011-09-01	44311	895098.64	276	United Kingdom
2011-10-01	52596	935404.14	324	United Kingdom
2011-11-01	76133	1319934.07	297	United Kingdom

- all_series(): getting the overall series table

```
DELIMITER $$
USE `project`$$
CREATE DEFINER=`root`@`localhost` PROCEDURE `all_series`()
BEGIN
DROP TABLE IF EXISTS time_series;
CREATE TABLE time_series
WITH cte AS
(
SELECT
    DISTINCT CustomerID,
    MIN(InvoiceDate) AS CohortMonth
```

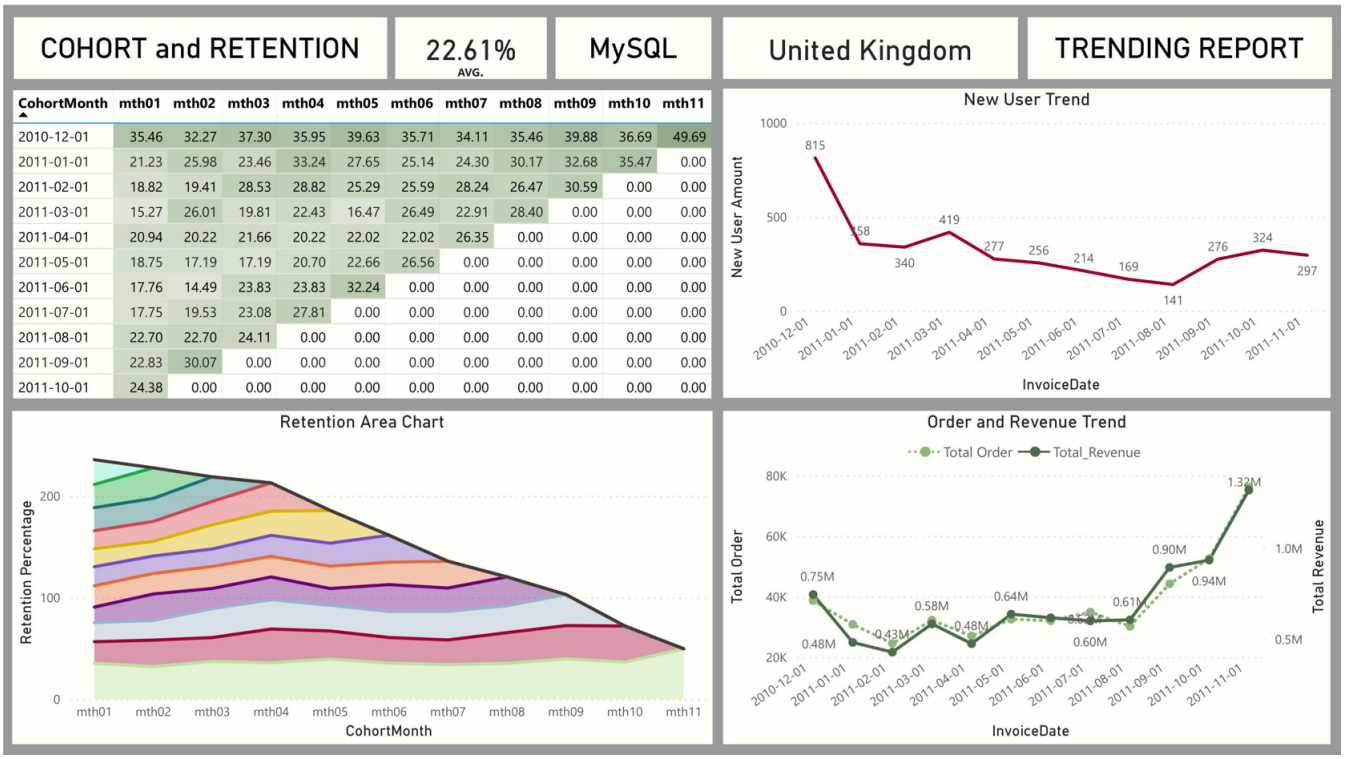
```
FROM report
GROUP BY CustomerID
),
cte2 AS
(
SELECT
    COUNT(CustomerID) AS NewUsers,
    CohortMonth
FROM cte
GROUP BY CohortMonth
)
SELECT
    InvoiceDate,
    COUNT(*) AS TotalOrders,
    ROUND(SUM(TotalRevenue),2) AS TotalRevenue,
    NewUsers
FROM report_clean
JOIN cte2
ON InvoiceDate = CohortMonth
GROUP BY InvoiceDate;
END$$
DELIMITER ;
```

Go back to the top

6. Data Sharing Part I

With the "time_series" table and "retention" table, creating the first dashboard.

Dashboard 1



CohortMonth	mth01	mth02	mth03	mth04	mth05	mth06	mth07	mth08	mth09	mth10	mth11
2010-12-01	35.46	32.27	37.30	35.95	39.63	35.71	34.11	35.46	39.88	36.69	49.69
2011-01-01	21.23	25.98	23.46	33.24	27.65	25.14	24.30	30.17	32.68	35.47	0.00
2011-02-01	18.82	19.41	28.53	28.82	25.29	25.59	28.24	26.47	30.59	0.00	0.00
2011-03-01	15.27	26.01	19.81	22.43	16.47	26.49	22.91	28.40	0.00	0.00	0.00
2011-04-01	20.94	20.22	21.66	20.22	22.02	22.02	26.35	0.00	0.00	0.00	0.00
2011-05-01	18.75	17.19	17.19	20.70	22.66	26.56	0.00	0.00	0.00	0.00	0.00
2011-06-01	17.76	14.49	23.83	23.83	32.24	0.00	0.00	0.00	0.00	0.00	0.00
2011-07-01	17.75	19.53	23.08	27.81	0.00	0.00	0.00	0.00	0.00	0.00	0.00
2011-08-01	22.70	22.70	24.11	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
2011-09-01	22.83	30.07	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
2011-10-01	24.38	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00

The visualization on dashboard 1 shows in United Kingdom:

- There are much more new customers in Dec 2010 and their retention rates are higher than those who joined later. These customers may include more wholesalers and have higher demands for the products of the company, so they learned about the business fast and have higher loyalty.
- The number of order is positively correlated to the total revenue.
- After the first month, the number of new customers slightly decreases, but the revenue and order do not follow the trend of new customers.
- The order and revenue increase sharply after Sep 2011 maybe because it was the holiday season and this e-commerce mainly sells gifts; their business is seasonal.
- The area chart shows that the retention rates increase for the first 4 months after new customers place their first order and decrease a little after 4 months. Gifts are not necessities, so those who have purchased from this business may not need to come back in a short time.

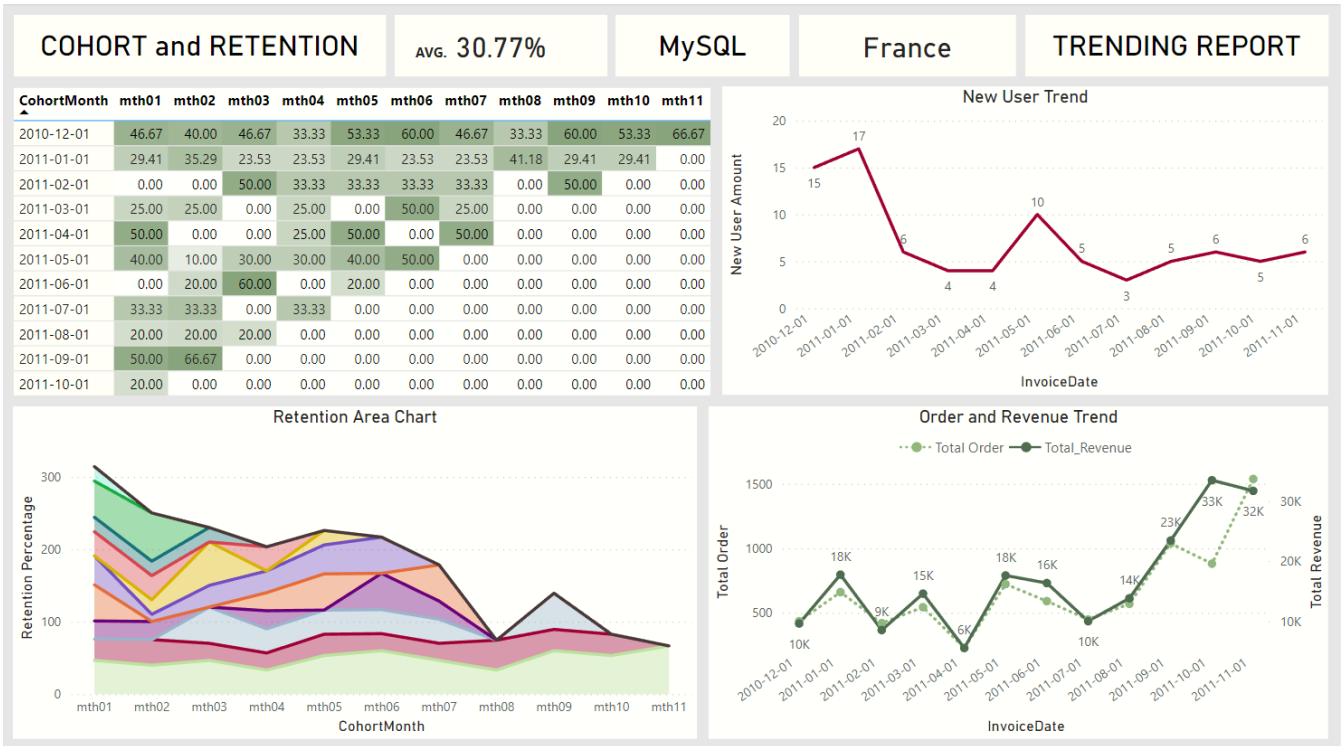
Manipulating the Dashboard

Steps to switch the dashboard to another country (e.g. France):

1. Use MySQL to run:

```
CALL country_series("France");
CALL country_retention("France");
```

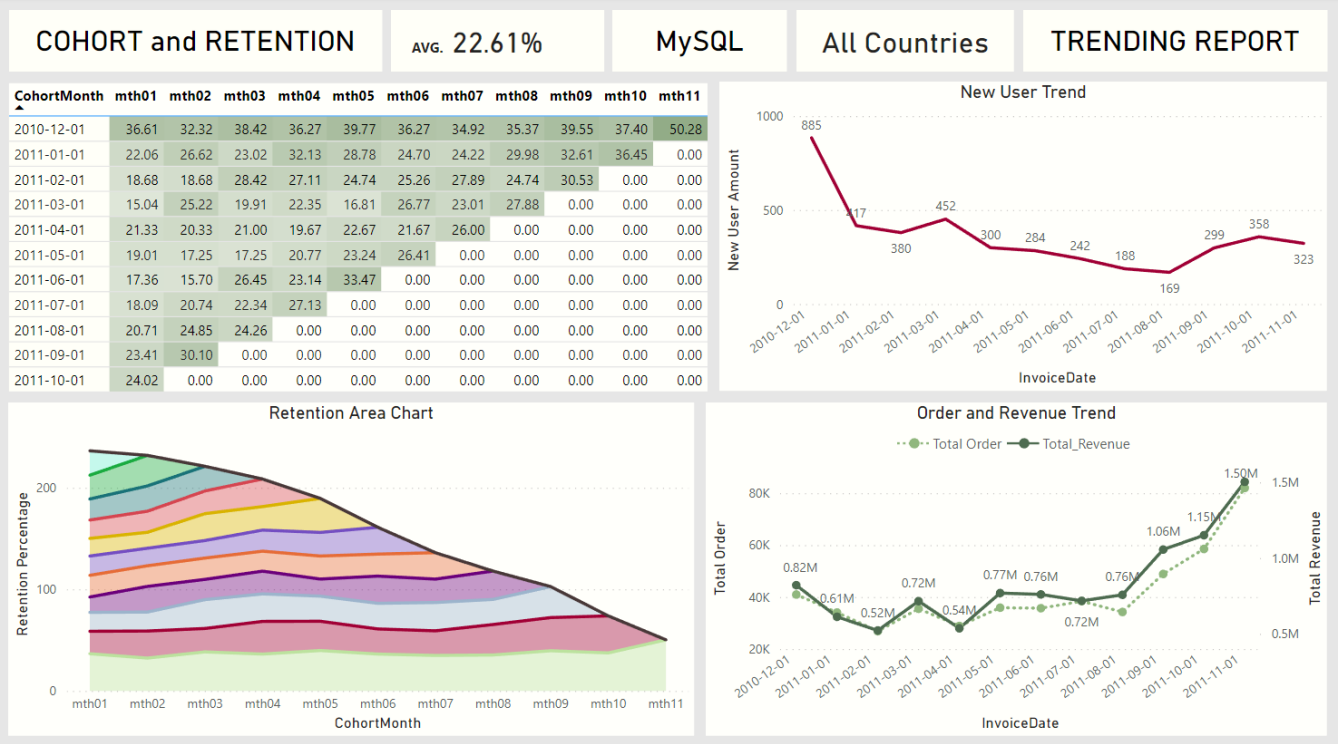
2. Refresh the data in Power BI and we get:



The data in France has a somewhat similar pattern with the data in the United Kingdom; however, the pattern is not obvious. If we look at the "New User Trend" chart, we can see that there are no more than 20 users placing orders each month. Therefore, the data in France are just too small to be random and to capture the patterns.

Switching to all countries
Running:

```
CALL all_series();
CALL all_retention();
```



CohortMonth	mth01	mth02	mth03	mth04	mth05	mth06	mth07	mth08	mth09	mth10	mth11
2010-12-01	36.61	32.32	38.42	36.27	39.77	36.27	34.92	35.37	39.55	37.40	50.28
2011-01-01	22.06	26.62	23.02	32.13	28.78	24.70	24.22	29.98	32.61	36.45	0.00
2011-02-01	18.68	18.68	28.42	27.11	24.74	25.26	27.89	24.74	30.53	0.00	0.00
2011-03-01	15.04	25.22	19.91	22.35	16.81	26.77	23.01	27.88	0.00	0.00	0.00
2011-04-01	21.33	20.33	21.00	19.67	22.67	21.67	26.00	0.00	0.00	0.00	0.00
2011-05-01	19.01	17.25	17.25	20.77	23.24	26.41	0.00	0.00	0.00	0.00	0.00
2011-06-01	17.36	15.70	26.45	23.14	33.47	0.00	0.00	0.00	0.00	0.00	0.00
2011-07-01	18.09	20.74	22.34	27.13	0.00	0.00	0.00	0.00	0.00	0.00	0.00
2011-08-01	20.71	24.85	24.26	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
2011-09-01	23.41	30.10	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
2011-10-01	24.02	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00

The patterns of the whole data set are very similar to the patterns of the United Kingdom; it is very likely that the company has more businesses in the United Kingdom.

Go back to the top

7. Summary I

- With the above 4 stored procedures I created, it is very fast and easy to update a 1-year cohort and trending analysis dashboard for a company. The data cleaning processes can also be stored in procedures as long as the data sets of the company are alway dirty in the same way, which is not hard to achieve if the company improves its data gathering methods. I chose not to do that because I used a third-party data set.
- The limitation of these MySQL procedures is that if I want to do a 2-year cohort analysis, I need to edit the procedures or create new ones because MySQL does not have PIVOT commands. It is easier to use pandas in Python to achieve that.
- Using MySQL and Power BI together to do cohort analysis is redundant because we can just use the power query editor in Power BI to get retention rate tables. In that case, all we need to do is to refresh the data source. I will demonstrate this theory in the following analysis.

8. RFM Analysis

https://clevertap.com/blog/rfm-analysis/ Creating another clean report table that is suitable for RFM and other analyses

```sql
DROP VIEW IF EXISTS report_rfm;
CREATE VIEW report_rfm
AS
SELECT
    InvoiceNo,
    StockCode,
    Description,
    IF(Quantity > 70000, 10, Quantity) AS Quantity,
    DATE_FORMAT(STR_TO_DATE(InvoiceDate, '%m/%d/%Y %H:%i'), '%Y-%m-%d') AS
InvoiceDate,
    UnitPrice,
    ROUND(IF(Quantity > 70000, 10, Quantity)*UnitPrice, 2) AS TotalRevenue,
    CustomerID,
    LEFT(Country, CHAR_LENGTH(Country) - 1) AS Country
FROM report_positive
```

The only difference between report_rfm and report_clean tables is that the 'InvoiceDate' in report_rfm table has the specific day for recency calculation in RFM analysis.

1. RFM Table

```sql
DROP VIEW IF EXISTS rfm_table;
CREATE VIEW rfm_table AS
WITH cte AS
(
SELECT
    *,
    MAX(InvoiceDate) OVER() AS max_date
FROM report_rfm
)

SELECT
    CustomerID,
    DATEDIFF(max_date, MAX(InvoiceDate)) AS Recency,
    COUNT(*) AS Frequency,
    ROUND(SUM(TotalRevenue),2) AS Monetary
FROM cte
GROUP BY 1
HAVING Frequency < 100000 # DELETING NULL CustomerID
```

CustomerID	Recency	Frequency	Monetary
12346	325	1	10.4
12347	2	182	4310
12348	75	31	1797.24
12349	18	73	1757.55
12350	310	17	334.4
12352	36	85	2506.04
12353	204	4	89
12354	232	58	1079.4
12355	214	13	459.4
12356	22	59	2811.43
12357	33	131	6207.67
12358	1	19	1168.06
12359	57	245	6310.03
12360	52	129	2662.06
12361	287	10	189.9
12362	3	266	5226.23
12363	109	23	552

2. RFM Table Ranked

Ranking RFM columns based on their values

```
SELECT
    CustomerID,
    DENSE_RANK() OVER(ORDER BY Recency) AS RecencyRank,
    DENSE_RANK() OVER(ORDER BY Frequency DESC) AS FrequencyRank,
    DENSE_RANK() OVER(ORDER BY Monetary DESC) AS MonetaryRank
FROM rfm_table
```

CustomerID	RecencyRank	FrequencyRank	MonetaryRank
14646	2	7	1
18102	1	95	2
17450	8	148	3
17450	2	2	4
12415	22	33	5
14156	9	12	6
17511	3	22	7
16029	34	223	8
16684	5	194	9

3. RFM Score Table

Assigning RFM scores from 1 to 5 to customers based on their RFM ranks

```
DROP VIEW IF EXISTS rfm_scores;
CREATE VIEW rfm_scores AS
WITH cte AS
(
SELECT
    CustomerID,
    DENSE_RANK() OVER(ORDER BY Recency) AS RecencyRank,
    DENSE_RANK() OVER(ORDER BY Frequency DESC) AS FrequencyRank,
    DENSE_RANK() OVER(ORDER BY Monetary DESC) AS MonetaryRank
FROM rfm_table
),
cte2 AS
(
SELECT
    *,
    MAX(RecencyRank) OVER() AS MaxR,
    MAX(FrequencyRank) OVER() AS MaxF,
    MAX(MonetaryRank) OVER() AS MaxM
FROM cte
),
cte3 AS
(
SELECT
    CustomerID,
    (CASE
        WHEN RecencyRank < 0.2*MaxR THEN 5
        WHEN RecencyRank >= 0.2*MaxR AND RecencyRank < 0.4*MaxR THEN 4
        WHEN RecencyRank >= 0.4*MaxR AND RecencyRank < 0.6*MaxR THEN 3
        WHEN RecencyRank >= 0.8*MaxR THEN 1
        ELSE 2 END) AS RScore,
    (CASE
```

```
        WHEN FrequencyRank < 0.2*MaxF THEN 5
        WHEN FrequencyRank >= 0.2*MaxF AND FrequencyRank < 0.4*MaxF THEN 4
        WHEN FrequencyRank >= 0.4*MaxF AND FrequencyRank < 0.6*MaxF THEN 3
        WHEN FrequencyRank >= 0.8*MaxF THEN 1
        ELSE 2 END) AS FScore,
    (CASE
        WHEN MonetaryRank < 0.2*MaxF THEN 5
        WHEN MonetaryRank >= 0.2*MaxF AND MonetaryRank < 0.4*MaxF THEN 4
        WHEN MonetaryRank >= 0.4*MaxF AND MonetaryRank < 0.6*MaxF THEN 3
        WHEN MonetaryRank >= 0.8*MaxF THEN 1
        ELSE 2 END) AS MScore
FROM cte2
)
SELECT
    *,
    CONCAT(RScore, FScore, MScore) AS RFMCell,
    Round((RScore + FScore + MScore)/3,1) AS RFMScore
FROM cte3
```

	CustomerID	RScore	FScore	MScore	RFMCell	RFMScore
▶	14646	5	5	5	555	5.0
	18102	5	4	5	545	4.7
	17450	5	4	5	545	4.7
	14911	5	5	5	555	5.0
	12415	5	5	5	555	5.0
	14156	5	5	5	555	5.0
	17511	5	5	5	555	5.0
	16029	5	3	5	535	4.3
	16684	5	3	5	535	4.3
	14096	5	5	5	555	5.0

4. Checking the RFM Cell distribution

```
SELECT
    RFMCell,
    COUNT(RFMCell)
FROM project.rfm_scores
GROUP BY RFMCell
ORDER BY 2 DESC
```

	RFMCell	COUNT(RFMCell)
▶	511	1516
	411	600
	521	475
	311	372
	211	371
	111	284
	531	143
	421	58
	541	56
	532	35
	555	33
	551	29
	533	28
	321	27
	544	26
	543	25
	554	23
	522	20

The top RFM cell is 511, those who place orders recently but have only placed a few orders and generated small revenues. They may be new customers or inactive customers who come back recently. For now, I call them recent customers.

5. Setting up segmentation based on "RFMCell"

Depending on the individual situation of each business. An example:

RFMCell	Segment	Description
555	Core Customers	Best of the best
55[1-4]	Loyal Customers	Ordering frequently
5[2-4]5	Value Customers	large amounts and more frequent
54[2-4]	Promising Customers	More active than the average
5[1-2][1-3]	Recent Customers	Ordering recently
5[3-4][1-4]	Active Customers	Ordering recently and more frequently
4[2-3]1, 52[3-4]	Less Active Customers	Less recent and Less frequent
[2-3][1-2]1, 41[1-2]	Inactive Customers	Average orders with small amounts
[1-2][1-2]1	Hybernating Customers	Neither recent nor frequent
[4-5]1[4-5]	Large-Order Customers	Only few orders but large amounts

```sql
DROP VIEW IF EXISTS customer_segment;
CREATE VIEW customer_segment AS
SELECT
    *,
    (CASE
        WHEN RFMCell LIKE '555' THEN 'Core Customer'
        WHEN RFMCell IN (551, 554, 553, 552) THEN 'Loyal Customer'
        WHEN RFMCell IN (545, 535, 525) THEN 'Value Customer'
        WHEN RFMCell IN (544, 543, 542) THEN 'Promising Customer'
        WHEN RFMCell IN (421, 431, 331, 524, 523) THEN 'Less Active Customer'
        WHEN RFMCell IN (511, 521, 522, 512, 513) THEN 'Recent Customer'
        WHEN RFMCell IN (531, 541, 532, 533, 534) THEN 'Active Customer'
        WHEN RFMCell IN (311, 321, 221, 412, 411) THEN 'Inactive Customer'
        WHEN RFMCell IN (211, 111, 121) THEN 'Hybernating Customer'
        WHEN RFMCell IN (514, 515, 414) THEN 'Large-Order Customer'
        ELSE 'Other Customers' END) AS Segment
    FROM rfm_scores
```

CustomerID	RScore	FScore	MScore	RFMCell	RFMScore	Segment
16839	5	3	5	535	4.3	Value Customer
12921	5	5	5	555	5.0	Core Customer
17677	5	4	5	545	4.7	Value Customer
15189	5	3	5	535	4.3	Value Customer
14607	5	1	5	515	3.7	Large-Order C...
15856	5	5	5	555	5.0	Core Customer
14051	5	3	5	535	4.3	Value Customer
15513	5	4	5	545	4.7	Value Customer
16133	5	4	5	545	4.7	Value Customer
14866	5	2	5	525	4.0	Value Customer
16705	5	3	5	535	4.3	Value Customer
12681	5	5	5	555	5.0	Core Customer

5. Adding customer segmentation columns back to the report

```
DROP VIEW IF EXISTS report_segment;
CREATE VIEW report_segment AS
SELECT
    r.* ,
    RFMCell,
    Segment
FROM report_clean r
LEFT JOIN customer_segment
USING(CustomerID)
```

InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	TotalRevenue	CustomerID	Country	RFMCell	Segment
536365	85123A	WHITE HANGING HEART T-LIGHT HOLDER	6	2010-12-01	2.55	15.3	17850	United Kingdom	143	Other Customers
536365	71053	WHITE METAL LANTERN	6	2010-12-01	3.39	20.34	17850	United Kingdom	143	Other Customers
536365	84406B	CREAM CUPID HEARTS COAT HANGER	8	2010-12-01	2.75	22	17850	United Kingdom	143	Other Customers
536365	84029G	KNITTED UNION FLAG HOT WATER BOTTLE	6	2010-12-01	3.39	20.34	17850	United Kingdom	143	Other Customers
536365	84029E	RED WOOLLY HOTTIE WHITE HEART.	6	2010-12-01	3.39	20.34	17850	United Kingdom	143	Other Customers
536365	22752	SET 7 BABUSHKA NESTING BOXES	2	2010-12-01	7.65	15.3	17850	United Kingdom	143	Other Customers
536365	21730	GLASS STAR FROSTED T-LIGHT HOLDER	6	2010-12-01	4.25	25.5	17850	United Kingdom	143	Other Customers
536366	22633	HAND WARMER UNION JACK	6	2010-12-01	1.85	11.1	17850	United Kingdom	143	Other Customers
536366	22632	HAND WARMER RED POLKA DOT	6	2010-12-01	1.85	11.1	17850	United Kingdom	143	Other Customers
536367	84879	ASSORTED COLOUR BIRD ORNAMENT	32	2010-12-01	1.69	54.08	13047	United Kingdom	521	Recent Customer
536367	22745	POPPY'S PLAYHOUSE BEDROOM	6	2010-12-01	2.1	12.6	13047	United Kingdom	521	Recent Customer
536367	22748	POPPY'S PLAYHOUSE KITCHEN	6	2010-12-01	2.1	12.6	13047	United Kingdom	521	Recent Customer
536367	22749	FELTCRAFT PRINCESS CHARLOTTE DOLL	8	2010-12-01	3.75	30	13047	United Kingdom	521	Recent Customer
536367	22310	IVORY KNITTED MUG COSY	6	2010-12-01	1.65	9.9	13047	United Kingdom	521	Recent Customer

Now, we can combine all queries in the RFM analysis to create another stored procedure called "update_segmentation()." Whenever we have new transactions, we can CALL this procedure to update those customers' segments. I will skip this step because this data set will never update.

Go back to the top

9. Additional Analyses

- #### In which country does the company have more businesses?

```
SELECT
  Country,
  COUNT(*) AS TotalOrders,
  SUM(TotalRevenue) AS TotalRevenues,
  COUNT(DISTINCT CustomerID) AS TotalCustomers,
  COUNT(DISTINCT StockCode) AS TotalProducts,
  COUNT(DISTINCT InvoiceDate) AS SalesPeriod,
  SUM(Segment = 'Core Customers') AS CoreCustomers
FROM report_segment
GROUP BY country
ORDER BY 3 DESC
```

Country	TotalOrders	TotalRevenues	TotalCustomers	TotalProducts	SalesPeriod	CoreCustomers
United Kingdom	479985	8756122.09	3921	3806	13	37358
Netherlands	2359	285446.34	9	782	13	2076
EIRE	7879	283140.52	4	1968	13	7065
Germany	9025	228678.4	94	1664	13	926
France	8392	209625.37	88	1542	13	1162
Australia	1181	138453.81	9	599	12	714
Spain	2479	61558.56	30	1091	13	478
Switzerland	1958	57067.6	22	978	12	0
Belgium	2031	41196.34	25	777	13	0
Sweden	450	38367.83	8	261	13	0
Japan	321	37416.37	8	215	10	0

- #### Top 10 customers

```
SELECT
  CustomerID,
  COUNT(*) AS TotalOrders,
  ROUND(SUM(TotalRevenue),2) AS TotalRevenues,
  COUNT(DISTINCT StockCode) AS TotalProducts,
  COUNT(DISTINCT InvoiceDate) AS RSDuration,
  Country,
  Segment
FROM report_segment
GROUP BY CustomerID
ORDER BY 3 DESC
LIMIT 1, 10 # Skipping the first row with NULL CustomerID
```

CustomerID	TotalOrders	TotalRevenues	TotalProducts	RSDuration	Country	Segment
14646	2076	280206.02	700	13	Netherlands	Core Customers
18102	431	259657.3	150	12	United Kingdom	Value Customers
17450	336	194390.79	124	12	United Kingdom	Value Customers
14911	5670	143711.17	1787	13	EIRE	Core Customers
12415	714	124914.53	444	11	Australia	Core Customers
14156	1395	117210.08	714	12	EIRE	Core Customers
17511	963	91062.38	453	13	United Kingdom	Core Customers
16029	241	80850.84	44	12	United Kingdom	Value Customers
16684	277	66653.56	119	9	United Kingdom	Value Customers
14096	5111	65164.79	1119	5	United Kingdom	Core Customers

- Although Netherlands is not the country that does the most business with the company, the customer who contributes the most revenue is located in Netherlands.
- Although value customers place fewer orders from the business, they still contribute many revenues.

- #### Top 10 ordered produts

```
SELECT
  StockCode,
  COUNT(StockCode) AS OrderTimes,
  Description
FROM report_segment
GROUP BY StockCode
ORDER BY 2 DESC
LIMIT 10
```

StockCode	OrderTimes	Description
85123A	2320	WHITE HANGING HEART T-LIGHT HOLDER
85099B	2109	JUMBO BAG RED RETROSPOT
22423	2007	REGENCY CAKESTAND 3 TIER
47566	1699	PARTY BUNTING
20725	1582	LUNCH BAG RED RETROSPOT
84879	1476	ASSORTED COLOUR BIRD ORNAMENT
22197	1418	SMALL POPCORN HOLDER
22720	1392	SET OF 3 CAKE TINS PANTRY DESIGN
21212	1352	PACK OF 72 RETROSPOT CAKE CASES
22383	1306	LUNCH BAG SUKI DESIGN

The champion is "White Hanging Heart T-Light Holder". Love is in the air!

- #### Which segment has the most customers and revenues?

```
SELECT
  Segment,
  COUNT(DISTINCT CustomerID) AS TotalCustomers,
  COUNT(*) AS TotalOrders,
  ROUND(SUM(TotalRevenue),2) AS TotalRevenues,
  COUNT(DISTINCT StockCode) AS TotalProducts,
  COUNT(DISTINCT InvoiceDate) AS SalesPeriod,
  COUNT(DISTINCT Country) AS TotalRegion
FROM report_segment
WHERE Segment IS NOT NULL
GROUP BY segment
ORDER BY 2 DESC
```

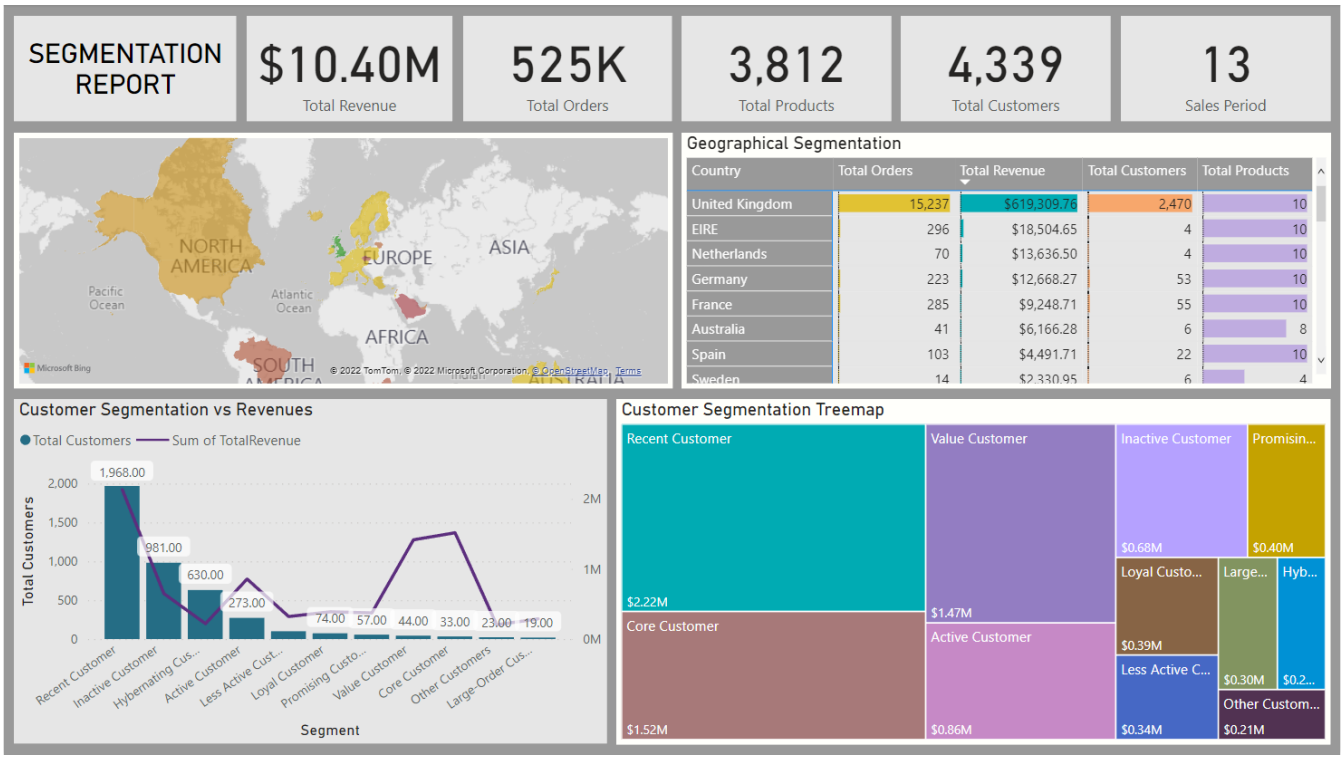
Segment	TotalCustomers	TotalOrders	TotalRevenues	TotalProducts	SalesPeriod	TotalRegion
Recent Customer	2025	123345	2218064.06	3341	13	26
Inactive Customer	1011	33751	683970.04	2870	10	26
Hybernating Customer	662	14191	246862.29	2337	6	26
Active Customer	274	69500	862667.86	3138	13	7
Less Active Customer	106	15424	336936.99	2459	13	8
Loyal Customer	74	48369	389335.93	2939	13	2
Promising Customer	61	21252	404255.19	2477	13	8
Value Customer	47	11802	1471423.21	1799	13	7
Core Customer	33	49779	1516343.41	3047	13	7
Other Customers	24	4265	208085.06	1434	10	5
Large-Order Customer	21	1014	303642.85	308	13	4

Go back to the top

10. Data Sharing Part II

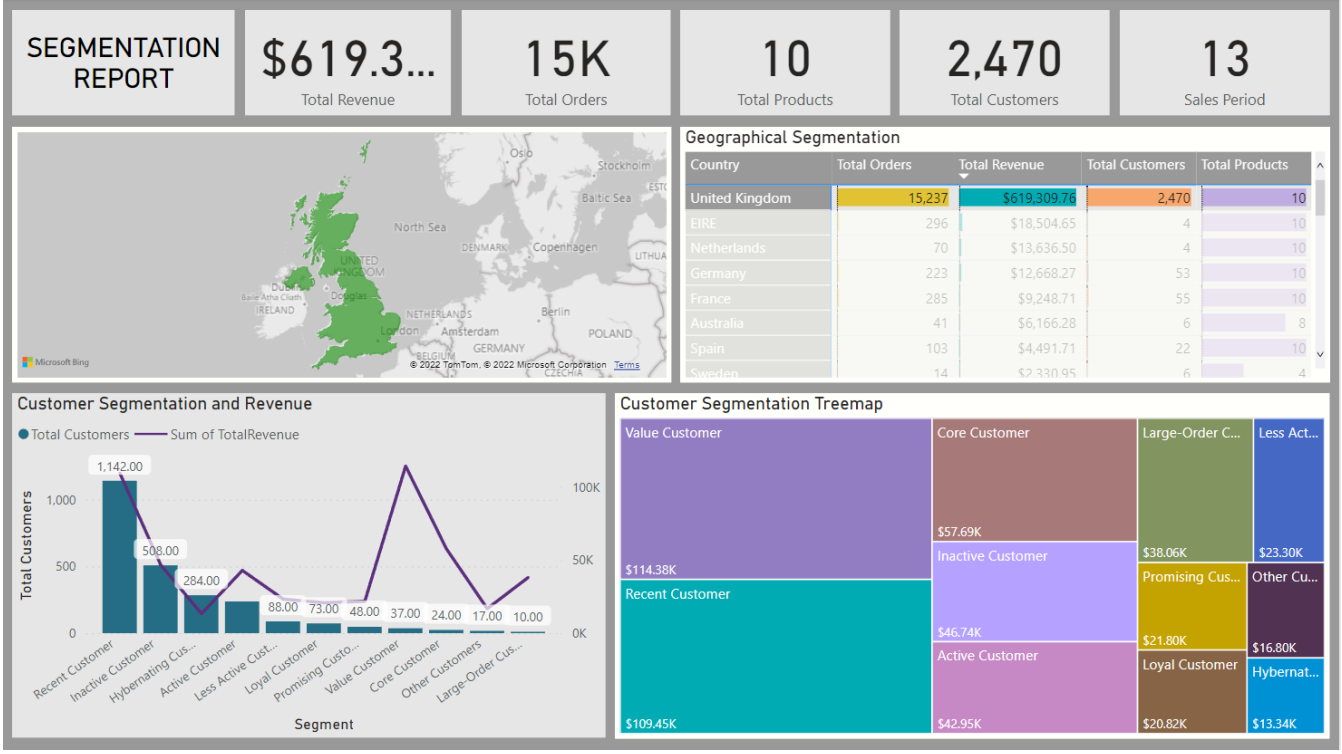
Dashboard 2

NOTE: compared to dashboard 1, I only imported the report with segmentation for dashboard 2 and used Power Query Editor and DAX language to manipulate the table for visualization.



- The second dashboard reports both geographical segmentation and customer segmentation.
- The geographical segmentation table points out that the company has much more businesses in the United Kingdom than in other countries.
- The "customer segmentation vs revenue" chart shows that although there are only a few core and value customers, they contribute many revenues to the company. Therefore, the company should try its best not to lose these customers.
- Other than that, those recent customers are also very valuable, but this segmentation is vague; we don't know whether they are new customers or those who just happen to place an order recently. We need techniques beyond RFM analysis to solve this problem.

Interaction with the dashboard



- By clicking the countries in the geographical map or the rows, we could view the customer segmentation based on countries.
- As the dashboard shows, the largest customer segment of the United Kingdom is "value customer" and they contribute the most revenues.
- The geography map in this dashboard is more aesthetic than practical. I am ready to replace it with other charts as needed by the stakeholders.

Go back to the top

11. Summary II

- With MySQL and Power BI, it is easy to refresh the data source and update the dashboard with new e-commerce transactions.
- Updating dashboard 2 is easier than updating dashboard 1 because the query and visualization were both performed in Power BI for dashboard 2; meanwhile, the query was performed in MySQL for dashboard 1.
- For the same reason, dashboard 2 has more interactions on the charts and maps.
- For automation purpose, it is easier to use Python than MySQL or Power BI. The libraries in Python have already had many "stored procedures"(functions) to speed the coding process. For example, MySQL does not even have a PIVOT command.
- RFM segmentation has its limitation. We still need techniques like K-mean Clustering to improve our segmentation. That would be another Python project.

Thank you for reading!