

巴西电商数据分析实战项目

背景：这是在 Olist Store 下达的订单的巴西电子商务公共数据集。该数据集包含 2016 年至 2018 年在巴西多个市场下达的 100k 个订单的信息。其功能允许从多个维度查看订单：从订单状态、价格、付款和货运绩效到客户位置、产品属性，最后是客户撰写的评论。我们还发布了一个地理位置数据集，将巴西邮政编码与纬度/液化天然气坐标相关联。

这是真实的商业数据，已被匿名化，评论文本中对公司和合作伙伴的引用已被《权力的游戏》各大家族的名称所取代。

数据来源：

https://www.kaggle.com/olistbr/brazilian-ecommerce?select=olist_customers_dataset.csv

包括九张表。

(1) **olist_customers_dataset**: 为客户数据集，包含有关客户及其位置的信息。使用它来识别订单数据集中的唯一客户并查找订单交付位置。每个订单都分配给一个唯一的 **customer_id**。这意味着同一买家将获得不同订单的不同 ID。

(2) **olist_geolocation_dataset**: 为地理位置数据集，包含巴西邮政编码及其纬度/液化天然气坐标信息。使用它来绘制地图并查找卖家和客户之间的距离。

(3) **olist_order_items_dataset**: 为订单项数据集，包含有关每个订单中购买的商品的数据，包括产品价格 (**price**) 和运费价格 (**freight_value**)，其中 **shipping_limit_date** 为卖家给物流合作伙伴设定的发货限制日期。

(4) **olist_order_payments_dataset**: 为支付数据集，包含有关订单付款选项的数据，其中 **payment_installmentss** 为客户选择的分期付款数。

(5) **olist_order_reviews_dataset**: 为订购评论数据集，包含有关客户所做评论的数据。客户从 Olist Store 购买产品后，卖家会收到通知以履行该订单。一旦客户收到产品或预计交货日期到期，客户将通过电子邮件收到满意度调查，他可以在其中提供购买体验的说明并写下一些评论。

(6) **olist_orders_dataset**: 为订单数据集，是核心数据集。从每个订单中可以找到所有其他信息。其中 **order_purchase_timestamp** 为购买时间戳，**order_approved_at** 为付款批准时间戳，**order_delivered_carrier_date** 为订单过账给物流的时间戳，**order_delivered_customer_date** 为客户实际订单交货时间戳，**order_estimated_delivery_date** 为购买时通知客户的预计交货时间。

(7) **olist_products_dataset**: 为产品数据集，包括有关 Olist 销售的产品数据。

(8) **olist_sellers_dataset**: 为卖家数据集，包括有关履行在 Olist 下的订单的卖家的数据。使用它来查找卖家位置并确定哪个卖家配送了每件商品。

(9) **product_category_name_translation**: 为类别名称翻译，将 **product_category_name** 翻译成英文。

一、基于 sql+powerbi

1. 数据导入到 navicat

将 9 个 csv 文件导入到 navicat

(1) 新建数据库 olist，设置字符集

编辑数据库

常规 SQL 预览

数据库名: olist

字符集: utf8mb4

排序规则: utf8mb4_unicode_ci

(2) 导入向导-csv

注意：如果直接将 csv 文件一次性全部导入，会存在部分数据 error，没法完整导入到 olist 数据库中，原因可能是：Navicat 默认导入编码是 utf8，部分表格下载后是 ANSI，由于编码格式不对等，导入乱码且 Navicat 对表格名称字符长度有要求，表名最好不要超过 20 个字符。因此在导入向导前，将原文件以记事本的形式打开，并逐一另存为 utf-8 的形式，同时修改文件名。

olist_customers_dataset	category
olist_geolocation_dataset	customer
olist_order_items_dataset	geo
olist_order_payments_dataset	item
olist_order_reviews_dataset	order
olist_orders_dataset	payment
olist_products_dataset	product
olist_sellers_dataset	review
product_category_name_translation	seller

以 priduct 表的导入示例，最好是一个个导入，因为每张表的记录分隔符不一样，一起导入的话，可能最后还是会报错。

导入向导

这个向导允许你指定如何导入数据。你要选择哪种数据导入格式？

导入类型:

- ☐ DBase 文件 (*.dbf)
- ☐ Paradox 文件 (*.db)
- ☐ 文本文件 (*.txt)
- ☒ CSV 文件 (*.csv)
- ☐ Excel 文件 (*.xls; *.xlsx)
- ☐ XML 文件 (*.xml)
- ☐ JSON 文件 (*.json)
- ☐ MS Access 数据库 (*.mdb; *.accdb)
- ☐ ODBC

下一步 > >> 取消

导入向导

你必须选择一个文件作为数据源。

导入从:

数据集\data\product.csv

编码:

65001 (UTF-8)

<<

< 上一步

下一步 >

>>

取消

你的字段要用什么分隔符来分隔？请选择合适的分隔符。

记录分隔符:

LF

☒ 分隔符 - 字符如逗号或制表符，用来界定每个字段

☐ 固定宽度 - 每个列内字段对齐，用空格分隔

字段分隔符:

逗号(,)

文本识别符号:

"

<<

< 上一步

下一步 >

>>

取消

导入向导

你可以为源定义一些附加的选项。

字段名行:

1

第一个数据行:

2

最后一个数据行:

格式

日期排序:

YMD

日期分隔符:

/

时间分隔符:

:

小数点符号:

.

日期时间排序:

日期 时间

二进制数据编码:

Base64

导入过程都按照系统默认的，不需要改动，只需要改日期排序YMD，即年月日

<<

< 上一步

下一步 >

>>

取消

选择目标表。你可选择现有的表，或输入新的表名。

源表	目标表	新建表
product	product	<input checked="" type="checkbox"/>

这个向导已对表结构进行一些猜测，现在你可以进行调整。

源表: product

目标表: product

	源字段	目标字段	类型	长度	比例	主键
<input checked="" type="checkbox"/>	product_id	product_id	varchar	255	0	
<input checked="" type="checkbox"/>	product_category_n	product_category_n	varchar	255	0	
<input checked="" type="checkbox"/>	product_name_lengl	product_name_lengl	varchar	255	0	
<input checked="" type="checkbox"/>	product_description	product_description	varchar	255	0	
<input checked="" type="checkbox"/>	product_photos_qty	product_photos_qty	varchar	255	0	
<input checked="" type="checkbox"/>	product_weight_g	product_weight_g	varchar	255	0	
<input checked="" type="checkbox"/>	product_length_cm	product_length_cm	varchar	255	0	
<input checked="" type="checkbox"/>	product_height_cm	product_height_cm	varchar	255	0	
<input checked="" type="checkbox"/>	product_width_cm	product_width_cm	varchar	255	0	

请选择一个所需的导入模式。

- 导入模式:
- ☒ 追加: 添加记录到目标表

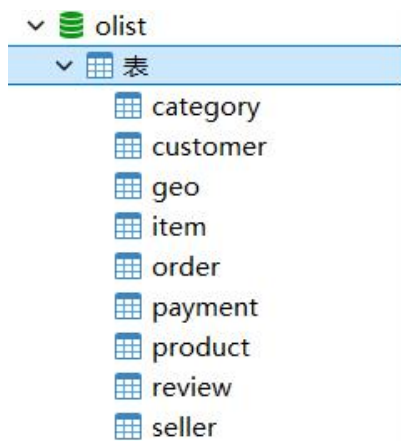
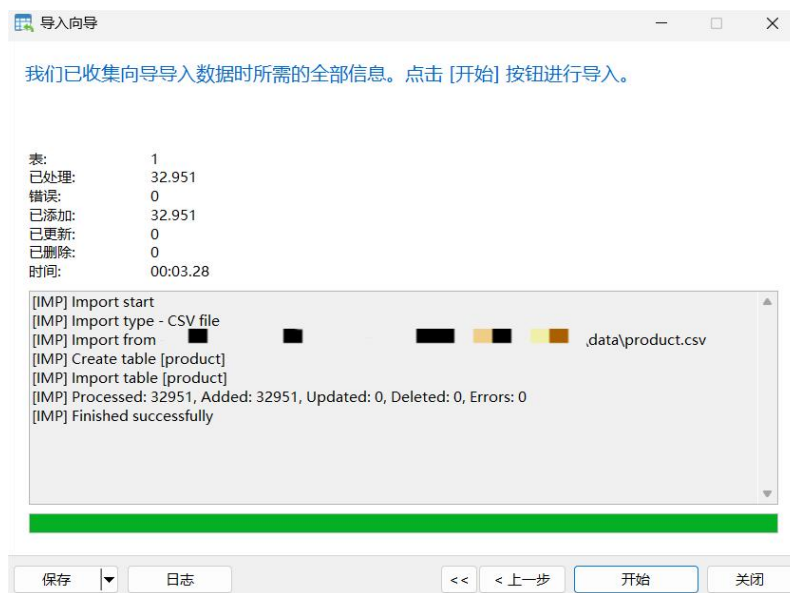
☐ 更新: 更新目标和源记录相符的记录

☐ 追加或更新: 如果目标存在相同记录, 更新它。否则, 添加它

☐ 删除: 删除目标中和源记录相符的记录

☐ 复制: 删除目标全部记录, 并从源重新导入

高级



2. 数据预处理

(1) 空值处理

从 kaggle 数据源提供的各表各列空值情况可知，review 表的空值集中在 review_comment_title 列和 review_comment_message 列，order 表的空值集中在 order_approved_at 列、order_delivered_carrier_date 列和 order_delivered_customer_date 列。用 0 替换这几列空值数据。

--源码:

#空值处理，update 修改替换

UPDATE review

SET review_comment_title=0

WHERE review_comment_title is NULL;

UPDATE review

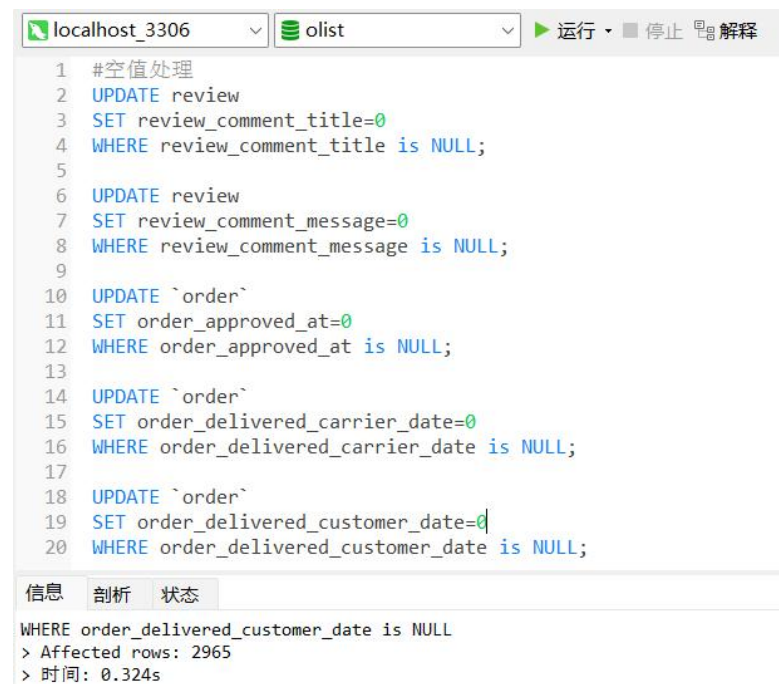
SET review_comment_message=0

WHERE review_comment_message is NULL;

```
UPDATE `order`  
SET order_approved_at=0  
WHERE order_approved_at is NULL;
```

```
UPDATE `order`  
SET order_delivered_carrier_date=0  
WHERE order_delivered_carrier_date is NULL;
```

```
UPDATE `order`  
SET order_delivered_customer_date=0  
WHERE order_delivered_customer_date is NULL;
```



The screenshot shows a SQL IDE interface with a toolbar at the top containing icons for localhost_3306, olist, and buttons for running (运行), stopping (停止), and explaining (解释) queries. The main area displays a SQL script with line numbers 1 through 20. The script includes comments and three UPDATE statements. The first two statements update the 'review' table, and the third updates the 'order' table. The execution results are shown at the bottom, indicating that the last statement affected 2965 rows in 0.324 seconds.

```
1 #空值处理  
2 UPDATE review  
3 SET review_comment_title=0  
4 WHERE review_comment_title is NULL;  
5  
6 UPDATE review  
7 SET review_comment_message=0  
8 WHERE review_comment_message is NULL;  
9  
10 UPDATE `order`  
11 SET order_approved_at=0  
12 WHERE order_approved_at is NULL;  
13  
14 UPDATE `order`  
15 SET order_delivered_carrier_date=0  
16 WHERE order_delivered_carrier_date is NULL;  
17  
18 UPDATE `order`  
19 SET order_delivered_customer_date=0  
20 WHERE order_delivered_customer_date is NULL;
```

信息	剖析	状态
WHERE order_delivered_customer_date is NULL > Affected rows: 2965 > 时间: 0.324s		

(2) 重复值处理

通过查询语句，逐一排查每张表的主键是否唯一。

--源码

#检查重复值

#category 表

```
SELECT product_category_name FROM category  
GROUP BY product_category_name  
HAVING COUNT(*)>1;
```

#customer 表

```
SELECT customer_id FROM customer  
GROUP BY customer_id  
HAVING COUNT(*)>1;
```

#geo 表

```
SELECT geolocation_zip_code_prefix FROM geo
GROUP BY geolocation_zip_code_prefix
HAVING COUNT(*)>1;
```

#item 表

```
SELECT order_id FROM item
GROUP BY order_id,order_item_id
HAVING COUNT(*)>1;
```

#`order`表

```
SELECT order_id FROM `order`
GROUP BY order_id
HAVING COUNT(*)>1;
```

#payment 表

```
SELECT order_id FROM payment
GROUP BY order_id
HAVING COUNT(*)>1;
```

#product 表

```
SELECT product_id FROM product
GROUP BY product_id
HAVING COUNT(*)>1;
```

#review 表

```
SELECT review_id,order_id FROM review
GROUP BY review_id,order_id
HAVING COUNT(*)>1;
```

#seller 表

```
SELECT seller_id FROM seller
GROUP BY seller_id
HAVING COUNT(*)>1;
```

The screenshot shows a SQL query execution window with the following content:

```

localhost_3306 olist 运行 停止 解释
4 GROUP BY product_category_name
5 HAVING COUNT(*)>1;
6
7 #customer表
8 SELECT customer_id FROM customer
9 GROUP BY customer_id
10 HAVING COUNT(*)>1;
11
12 #geo表
13 SELECT geolocation_zip_code_prefix FROM geo
14 GROUP BY geolocation_zip_code_prefix
15 HAVING COUNT(*)>1;
16
17 #item表

```

Below the code, there is a table of results with columns: 信息, 结果 1, 结果 2, 结果 3, 结果 4, 结果 5, 结果 6, 结果 7, 结果 8, 结果 9, 剖析, 状态.

信息	结果 1	结果 2	结果 3	结果 4	结果 5	结果 6	结果 7	结果 8	结果 9	剖析	状态
geolocation_zip_code_prefix	01037										
	01046										
	01041										
	01035										
	01012										
	01047										
	01013										

结果分析:

[1]category 表、customer 表、item 表、`order`表、product 表、review 表、seller 表都不存在重复值。

[2]payment 表和 geo 表存在重复值。经分析，payment 表中“order_id”重复是正常现象，因为订单设置中有分期付款的选项，同一个订单可能存在多次分期付款的数据记录。geo 表中“geolocation_zip_code_prefix”重复是正常现象，其表示的是巴西邮政编码前 5 位数字，在实际生活中存在某一片区域共用同一个邮政编码。

(3) 数据类型转换（一般对时间类型的数据进行处理）。

对时间类型的数据进行年季月日分隔处理，目的是为了分析年活跃度、月活跃度、日活跃度等。重点研究的是 order 订单表中“order_purchase_timestamp”

注意：在一开始导入数据的时候，用的都是系统默认的数据类型，并没有进行修改，在编写 sql 语言前，需要将 order_purchase_timestamp 的数据类型改为“datetime（6）”，否则会报错。

--源码

```

CREATE TABLE order_time AS
SELECT order_id,customer_id,
YEAR(order_purchase_timestamp) AS y,
QUARTER(order_purchase_timestamp) AS q,
MONTH(order_purchase_timestamp) AS m,
DATE(order_purchase_timestamp) AS d,
HOUR(order_purchase_timestamp) AS h
FROM `order`
WHERE order_purchase_timestamp!=0

```



```
localhost_3306 olist 运行 停止 解释
1 CREATE TABLE order_time AS
2 SELECT order_id,customer_id,
3 YEAR(order_purchase_timestamp) AS y,
4 QUARTER(order_purchase_timestamp) AS q,
5 MONTH(order_purchase_timestamp) AS m,
6 DATE(order_purchase_timestamp) AS d,
7 HOUR(order_purchase_timestamp) AS h
8 FROM `order`
9 WHERE order_purchase_timestamp!=0
```

信息 剖析 状态

```
SELECT order_id,customer_id,
YEAR(order_purchase_timestamp) AS y,
QUARTER(order_purchase_timestamp) AS q,
MONTH(order_purchase_timestamp) AS m,
DATE(order_purchase_timestamp) AS d,
HOUR(order_purchase_timestamp) AS h
FROM `order`
WHERE order_purchase_timestamp!=0
> OK
> 时间: 1.536s
```

3.数据分析

(1) 流量指标：活跃用户量和订单量分析

先计算出每笔订单的金额，再将每笔订单的订单号、用户号、购买时间和订单金额整合再一张表中。

a.创建 total_order_value 表，并计算订单总金额，将计算结果放在“total_price”字段中。

--源码

#计算每笔订单金额

-- order_id 有 3 件商品（同一商品）。每件物品的运费都根据其尺寸和重量计算。要获得每个订单的总运费价值，只需求和即可。

-- 例如订单号 00143d0f86d6fbd9f9b38ab440ac16f5

-- 总 order_item 值为: $21.33 * 3 = 63.99$

-- 货物总价值为: $15.10 * 3 = 45.30$

-- 订单总价值（产品 + 运费）为: $45.30 + 63.99 = 109.29$

CREATE TABLE total_order_value AS

SELECT order_id,product_id,seller_id,(price*COUNT(*))+(freight_value*COUNT(*)) AS total_price

FROM item

GROUP BY order_id

```
localhost_3306 olist 运行 停止 解释
1 #计算每笔订单金额
2 --
3 -- order_id有3件商品（同一商品）。每件物品的运费都根据其尺寸和重量计算。要获得每个订单的总运费价值，只需求和即可。
4 -- 例如订单号00143d0f86d6fbd9f9b38ab440ac16f5
5 -- 总order_item值为: 21.33 * 3 = 63.99
6 -- 货物总价值为: 15.10 * 3 = 45.30
7 -- 订单总价值（产品 + 运费）为: 45.30 + 63.99 = 109.29
8 CREATE TABLE total_order_value AS
9 SELECT order_id,product_id,seller_id,(price*COUNT(*))+(freight_value*COUNT(*)) AS total_price
10 FROM item
11 GROUP BY order_id
12
```

信息 剖析 状态

```
-- 例如订单号00143d0f86d6fbd9f9b38ab440ac16f5
-- 总order_item值为: 21.33 * 3 = 63.99
-- 货物总价值为: 15.10 * 3 = 45.30
-- 订单总价值（产品 + 运费）为: 45.30 + 63.99 = 109.29
CREATE TABLE total_order_value AS
SELECT order_id,product_id,seller_id,(price*COUNT(*))+(freight_value*COUNT(*)) AS total_price
FROM item
GROUP BY order_id
> OK
> 时间: 1.826s
```

b.创建 order_detail 表，将 order_time 表和 total_order_value 表中的金额和时间整合在一张表中。

--源码

#整合每笔订单的时间和金额，为后面计算 GMV 等指标做准备

CREATE VIEW order_detail AS

SELECT tv.order_id,product_id,seller_id,customer_id,

ROUND(total_price,2) AS total_price,y,q,m,d,h

FROM total_order_value tv JOIN order_time ot ON tv.order_id=ot.order_id;

```
1 #整合每笔订单的时间和金额，为后面计算GMV等指标做准备
2 CREATE VIEW order_detail AS
3 SELECT tv.order_id,product_id,seller_id,customer_id,
4 ROUND(total_price,2) AS total_price,y,q,m,d,h
5 FROM total_order_value tv JOIN order_time ot ON tv.order_id=ot.order_id;
6
7
```

信息 状态

```
#整合每笔订单的时间和金额，为后面计算GMV等指标做准备
CREATE VIEW order_detail AS
SELECT tv.order_id,product_id,seller_id,customer_id,
ROUND(total_price,2) AS total_price,y,q,m,d,h
FROM total_order_value tv JOIN order_time ot ON tv.order_id=ot.order_id
> 1050 - Table 'order_detail' already exists
> 时间: 0.001s
```

注意：order_detail 表、order_time 表、total_order_value 表在创建的时候，大家可以根据自己的需求，自行决定是创建表（CREATE TABLE）还是创建视图（CREATE VIEW），两者的区别在于表是实际存储在我们数据库中的，而视图是逻辑上的虚拟表，不存储在数据库中，相当于中间转换的虚拟表，不占存储内存。打算在 navicat 中先通过 sql 语句看看数据是什么情况时，建议大家创建 view，可以很快就创建好（几秒内），创建 table 平均需要 20 分钟。打算将 mysql 与 powerbi 或是 tableaiu 连接进行可视化时，建议大家创建 table，连接和加载数据的时间远超于 view。

【1】sql 中活跃用户数及订单数分析

#活跃用户时间分布

#1.日活跃用户数 DAU（运行 21min）

SELECT d 日期,COUNT(DISTINCT customer_id) DAU FROM order_detail

GROUP BY d

ORDER BY d;

#2.月活跃用户数 MAU

SELECT y 年份,m 月份,COUNT(DISTINCT customer_id) MAU FROM order_detail

GROUP BY y,m

ORDER BY y,m;

#3.日时段活跃用户数 HAU

SELECT h 时, COUNT(DISTINCT customer_id) HAU FROM order_detail

GROUP BY h

ORDER BY h;

#订单数时间分布

#1.每日订单数

```
SELECT d 日期,COUNT(order_id) 日订单数 FROM order_detail  
GROUP BY d  
ORDER BY d;
```

#2.每月订单数

```
SELECT y 年份,m 月份,COUNT(order_id) 月订单数 FROM order_detail  
GROUP BY y,m  
ORDER BY y,m;
```

#3.各时段订单数

```
SELECT h 时,COUNT(order_id) 时订单数 FROM order_detail  
GROUP BY h  
ORDER BY h;
```

localhost_3306 olist 运行

```
1 SELECT  
2   d 日期,  
3   COUNT( DISTINCT customer_id ) DAU,  
4   COUNT( DISTINCT order_id ) 日订单数  
5 FROM  
6   order_detail  
7 GROUP BY  
8   d  
9 ORDER BY  
10  d;
```

日期	DAU	日订单数
2016-09-04	1	1
2016-09-05	1	1
2016-09-15	1	1
2016-10-02	1	1
2016-10-03	8	8
2016-10-04	60	60
2016-10-05	42	42
2016-10-06	49	49
2016-10-07	45	45
2016-10-08	40	40

localhost_3306 olist 运行 停止 解释

```
1 SELECT y 年份,m 月份,COUNT(DISTINCT customer_id) MAU,COUNT(order_id) 月订单数 FROM order_detail  
2 GROUP BY y,m  
3 ORDER BY y,m;
```

年份	月份	MAU	月订单数
2017	8	4293	4293
2017	9	4243	4243
2017	10	4568	4568
2017	11	7451	7451
2017	12	5624	5624
2018	1	7220	7220
2018	2	6694	6694
2018	3	7188	7188
2018	4	6934	6934
2018	5	6853	6853
2018	6	6160	6160
2018	7	6273	6273
2018	8	6452	6452
2018	9	1	1



结果分析：上述截图的 sql 将用户量和订单量的计算放在一起进行查询，故只有三张截图。通过 sql 初步分析可以看出用户量和订单量实则是相同的，存在重复的用户，但订单是唯一。

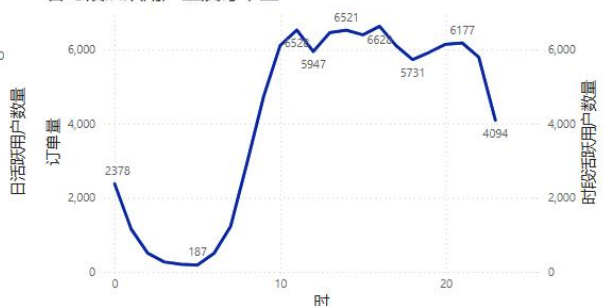
【2】powerbi 可视化分析

将 mysql 数据库 olist 中的 order_detail 数据表加载到 powerbi 中，分析日、月、季度、时段活跃用户和订单量。因为该数据集为只包括 2016 到 2018 年的数据，故不进行年活跃用户和订单量的分析。

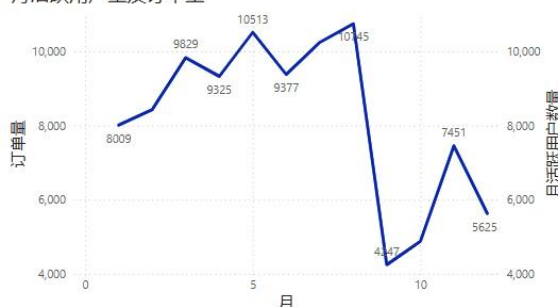
日活跃用户量及订单量



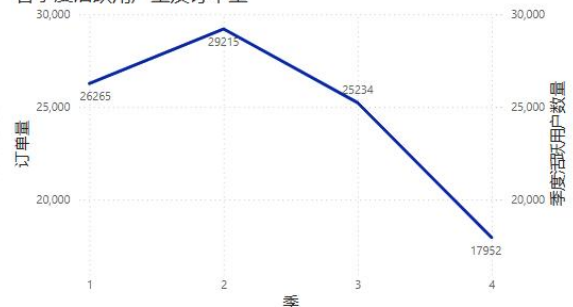
各时段活跃用户量及订单量



月活跃用户量及订单量



各季度活跃用户量及订单量



可视化结果分析：

1) 从日活跃用户量及订单量的图中可以观察到活跃用户数和订单数持续稳步上升，并在 17 年 11 月下旬出现明显峰值，18 年后 5 月和 7 月有所下降。可以重点关注出现峰值的那段时间是通过怎样的促销手段提高订单量的。

2) 从各时段活跃用户量及订单量图中可以观察到上午 10 点到晚上 22 点的用户数和订单数明显高于平均水平，是用户活跃时段。其中推测由于饭点休息时间导致 12 点和 18 点活

跃度略有下滑。22 点后用户数和订单数明显下降，到次日 5 点后开始回升，直到 9 点后逐渐恢复到正常活跃水平。

3) 从月活跃用户量及订单量图中可以观察到 2016-2018 年平均每月的订单量中 3 月、5 月、8 月是销量最好的月份，9 月和 10 月的订单销量过低，可以重点分析销量过低的原因，作为突破口。

4) 从季活跃用户量及订单量图中可以观察到 2016-2018 年平均每季度中第二季度，即 4 月、5 月、6 月的销量达到峰值，基本情况与 MAU 相同。

(2) 运营指标：GMV（商品交易总额）和 ARPU（企业从每个活跃用户处获得的平均收益）分析

【1】sql 初步分析

--源码

1.季度 GMV

```
select y 年份,q 季度,sum(total_price) 季度 GMV from order_detail
group by y,q
order by y,q;
```

2.月 GMV

```
select y 年份,m 月份,sum(total_price) 月 GMV from order_detail
group by y,m
order by y,m;
```

3.各季度 ARPU 值=GMV/季度活跃用户数

```
select y 年份,q 季度, round((sum(total_price)/count(DISTINCT customer_id)),2) 季度 ARPU
from order_detail
group by y,q
order by y,q;
```

4.各月 ARPU 值=GMV/月活跃用户数

```
select y 年份,m 月份,round((sum(total_price)/count(DISTINCT customer_id)),2) 月 ARPU from
order_detail
group by y,m
order by y,m;
```

```
1 # 季GMV
2 select y 年份,q 季度,sum(total_price) 季GMV from order_detail
3 group by y,q
4 order by y,q;
5
```

信息	结果 1	剖析	状态
年份	季度	季GMV	
2016	3	361.84	
2016	4	56675.99	
2017	1	855043.16	
2017	2	1502970.04	
2017	3	1976885.85	
2017	4	2813599.55	
2018	1	3248998.58	
2018	2	3351099.44	
2018	3	2060923.66	

```

1 # 月GMV
2 select y 年份,m 月份,sum(total_price) 月GMV from order_detail
3 group by y,m
4 order by y,m;

```

信息	结果 1	剖析	状态
年份	月份	月GMV	
2016	9	361.84	
2016	10	56656.37	
2016	12	19.62	
2017	1	137348.61	
2017	2	286132.45	
2017	3	431562.10	
2017	4	412682.35	
2017	5	588389.53	
2017	6	501898.16	
2017	7	587522.29	
2017	8	668524.17	
2017	9	720839.39	
2017	10	771086.25	
2017	11	1176986.51	
2017	12	865526.79	
2018	1	1106669.78	
2018	2	987070.83	
2018	3	1155257.97	
2018	4	1161030.94	
2018	5	1167155.07	
2018	6	1022913.43	
2018	7	1059695.21	

```

1 # 各季ARPU值=GMV/季度活跃用户数
2 select y 年份,q 季度, round((sum(total_price)/count(DISTINCT customer_id)),2) 季ARPU from order_detail
3 group by y,q
4 order by y,q;

```

信息	结果 1	剖析	状态
年份	季度	季ARPU	
2016	3	120.61	
2016	4	183.42	
2017	1	165.61	
2017	2	162.17	
2017	3	158.09	
2017	4	159.47	
2018	1	153.97	
2018	2	168.00	
2018	3	161.95	


```

1 # 各月ARPU值=GMV/月活跃用户数
2 select y 年份,m 月份,round((sum(total_price)/count(DISTINCT customer_id)),2) 月ARPU from order_detail
3 group by y,m
4 order by y,m;
5

```

信息 结果 1 剖析 状态

年份	月份	月ARPU
2016	9	120.61
2016	10	183.95
2016	12	19.62
2017	1	174.08
2017	2	165.11
2017	3	163.41
2017	4	172.60
2017	5	160.76
2017	6	156.01
2017	7	148.03
2017	8	155.72
2017	9	169.89
2017	10	168.80
2017	11	157.96
2017	12	153.90
2018	1	153.28
2018	2	147.46
2018	3	160.72
2018	4	167.44
2018	5	170.31
2018	6	166.06
2018	7	168.93

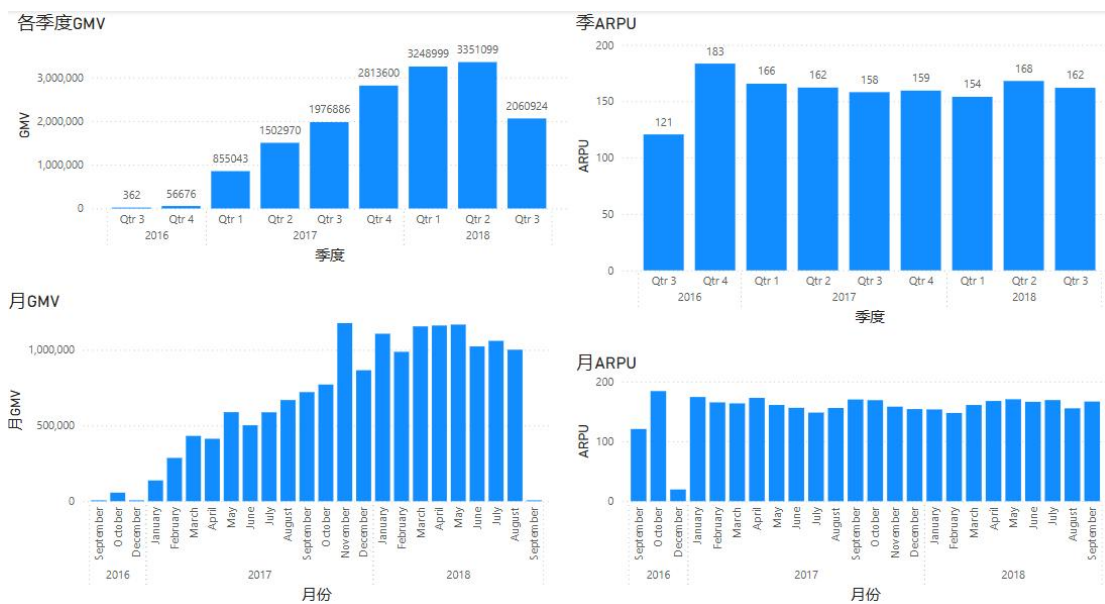
结果分析:

通过 sql 的查询语言可以初步看出各时间段的 GMV 和 APRU 基本情况,大致明白哪个时间段的商品交易额最高,哪个最低,但具体的还是需要可视化分析。

【2】powerbi 可视化分析

将 mysql 数据库 olist 中的 order_detail 数据表加载到 powerbi 中,分析月、季度的 GMV 和 APRU,其中 GMV 为 order_detail 中的字段“order_value”,APRU 则在 powerbi 中新建一个快速度量值,将字段“order_value”总和除以“customer_id”的非重复计数。

注意:字段“order_value”和“total_price”是同样的含义(销售额),只不过命名不同,在 order_detail 视图中销售额命名为“total_price”,在 order_detail 表中命名为“order_value”。



可视化结果分析:

1) 从各季度 GMV 的图中可以观察到, 2016 到 2017 年间该电商平台的 GMV 快速增长, 平均季度增幅近 50%, 但在进入 18 年后增速放缓, 甚至在 18 年 Q3 出现负增长。

2) 从月 GMV 图中细化观察各月情况, 可以观察到该平台的快速发展期其实是在 16 年的 Q4 和 17 年的 Q1, 随后增速整体趋缓, 仅靠个别月份的小峰值维持整体增速。2018 年 Q3 的 GMV 出现负增长的原因一方面是缺少 2018 年 9 月的数据, 但同时也可以看到 2018 年 7-8 月的 GMV 整体是不及 4-6 月的水平的, 一定程度上可以说明出现了负增长的苗头。

3) 从季度 APRU 的图中可以观察到用户平均收入在 2016-2017 年间有明显提升后, 一直维持在一定水平上下波动, 没有进一步提升, 并在 2018 年 Q3 开始下滑, 其下滑原因跟 GMV 类似, 跟数据缺失有一定关系, 但 ARPU 值无法突破的问题也值得平台重视。

4) 从月 APRU 的图中可以观察到 2016 年 12 月的用户平均收入极低, 需要平台重视。

(3) RFM 模型分层分析: 各层次用户品类, 其中热门指数=(销售金额+评价分数)的加权求和。

R, Recency, 即每个客户有多少天没回购了, 可以理解为最近一次购买到现在隔了多少天。

F, Frequency, 是每个客户购买了多少次。

M, Monetary, 代表每个客户平均购买金额, 这里也可以是累计购买金额。

1.RFM 用户分群

(1) 构造 F: 查看所有用户的消费次数 (运行 24min)

```
SELECT customer_id, count(*) 消费频率 from order_detail
GROUP BY customer_id
HAVING
count(*)>=1;
```

1	# RFM用户分群
2	# 构造F: 查看所有用户的消费次数
3	SELECT customer_id, count(*) 消费频率 from order_detail
4	GROUP BY customer_id
5	HAVING count(*)>=1;

信息	结果 1	剖析	状态
	customer_id	消费频率	
	6772a0a230a266	1	
	3a897024068ed	1	
	d2a7963608459	1	
	57ee2ef64f17a5	1	
	a78b75a2ad061i	1	
	61449fa1b8b899	1	
	2e068340b3037	1	
	2d8748fb35d51i	1	
	2528510465c75	1	
	3c628393675b4	1	
	076dcea2eb6c8i	1	
	1a1b5f9e903aa3	1	
	53379b14e4048	1	
	da2fbad9b2235c	1	

SELECT customer_id, count(*) 消费频率 from orde 只读 查询时间: 0.435s 第 1 条记录 (共 98666 条)

结果分析:

通过查询结果可以看出每一个用户的消费频率都为 1，因此在后续的 RFM 模型中不需要考虑 F。

(2) 构造 R 值

```
CREATE VIEW Rencency AS
SELECT customer_id,(CASE WHEN
DATEDIFF(d,(SELECT MAX(d) FROM order_detail))>(SELECT AVG(DATEDIFF(d,(SELECT MAX(d)
FROM order_detail))) FROM order_detail)
THEN 1 ELSE 0 END) AS R
FROM order_detail;
```

(3) 构造 M 值

```
CREATE VIEW Monetary AS
SELECT customer_id,order_value>(SELECT AVG(order_value) FROM order_detail) as M
FROM order_detail;
```

(4) 构造 RFM 分层

```
CREATE VIEW RFM AS
SELECT Rencency.customer_id, (CASE
WHEN R=1 AND M=1 THEN '重要发展用户'
WHEN R=0 AND M=1 THEN '重要挽留用户'
WHEN R=1 AND M=0 THEN '一般发展用户'
WHEN R=0 AND M=0 THEN '一般挽留用户'
ELSE '其他' END) AS 用户类型
FROM Rencency INNER JOIN Monetary ON Rencency.customer_id=Monetary.customer_id;
```

customer_id	用户类型
6772a0a230a2667d16c3620f000e1348	一般挽留用户
3a897024068ed42a183de61d5727d866	一般发展用户
d2a79636084590b7465af8ab374a8cf5	一般发展用户
57ee2ef64f17a5f9a4bf30489b06326c	一般发展用户
a78b75a2ad06180de06b82857ca442b3	一般发展用户
61449fa1b8b8998c9c3f3a7f0ae954ef	一般挽留用户
2e068340b3037f667ce3224bd59260af	一般挽留用户
2d8748fb35d51c2fd273ee67fff78b7a	一般挽留用户
2528510465c7572dedee4431a7d52e08	一般挽留用户
3c628393675b42c6b5ef89461f68ecef	重要挽留用户
076dcea2eb6c8dbe9354f7c0d8670a00	重要挽留用户
1a1b5f9e903aa3c203caf5cb63fca2b1	一般发展用户
53379b14e4048e025dc5a2185f461bb7	重要发展用户
da2fbad9b2235c67c9bc7c7a274ee389	一般发展用户
b6f05b99667a02c3722610a7bb6b4062	一般挽留用户
369ec69ef76c47e51980b88a5740dc1b	重要发展用户
7bb3b0d45b4e1a13cd914e4a135e6bd9	一般发展用户
ee4a4b5842e03fbaff366b9459e08b95	重要发展用户
cf4604a56d4e374de013f1ec35280d00	一般发展用户
1fc77bb785f297743a18fac02b30de62	一般挽留用户
03b3813a78be4cf5205e70977ad78d29	一般挽留用户

#2.各类型用户的热门商品类型分析

热门指数=0.7*消费金额+0.3*商品评分*10000（这里乘 10000 是为了平衡消费金额和评分之间的数量级差距）

（1）查看各类型用户数量

```
SELECT 用户类型,count(*) 数量 from rfm
```

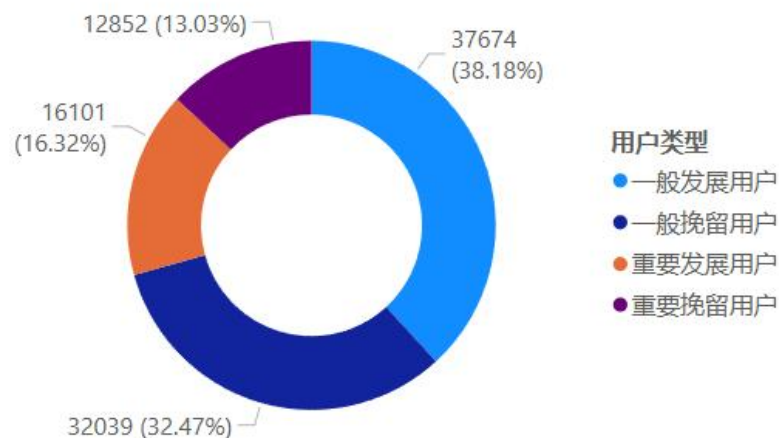
```
GROUP BY 用户类型
```

```
ORDER BY 用户类型;
```

```
1 # 查看各类型用户数量
2 SELECT 用户类型,count(*) 数量 from rfm
3 GROUP BY 用户类型
4 ORDER BY 用户类型;
```

信息	结果 1	剖析	状态
用户类型		数量	
▶一般发展用户		37674	
一般挽留用户		32039	
重要发展用户		16101	
重要挽留用户		12852	

通过创建表 rfm_1 将其导入 poewrbi 中即可形成环形图。



结果分析:

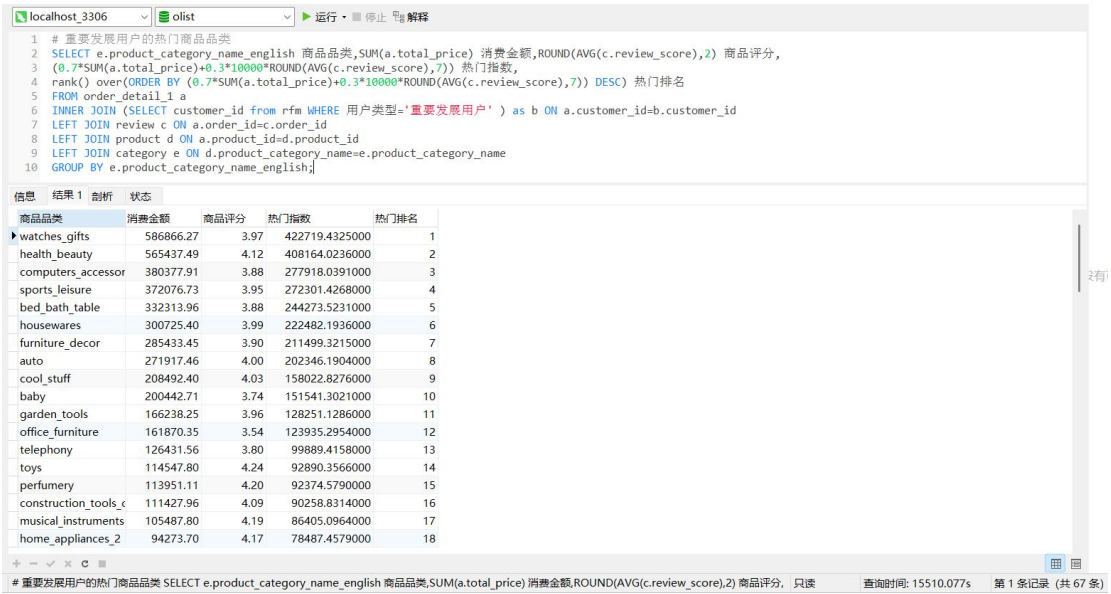
可视化的展示不同用户在占比情况，其中一般发展用户占比最多，为 38.18%，一般挽留用户次之，占比 32.47%，重要发展用户占比 16.32%，重要挽留用户占比 13.03%。对不同类型的用户通过研究其热门商品品类可以更好的帮助平台了解用户的偏好并促进商品的销售。

接下来通过 sql 语言来查询各类用户的热门商品品类，因为耗时过久（生成表，连接 poerbi），故只展示 sql 的运行结果，powerbi 可视化的图表就不展示了。

重要发展用户的热门商品品类（运行 4.3h）

```
SELECT e.product_category_name_english 商品品类 ,SUM(a.total_price) 消费金额,ROUND(AVG(c.review_score),2) 商品评分,
```

(0.7*SUM(a.total_price)+0.3*10000*ROUND(AVG(c.review_score),7)) 热门指数,
rank() over(ORDER BY (0.7*SUM(a.total_price)+0.3*10000*ROUND(AVG(c.review_score),7)) DESC)
热门排名
FROM order_detail_1 a
INNER JOIN (SELECT customer_id from rfm WHERE 用户类型='重要发展用户') as b ON
a.customer_id=b.customer_id
LEFT JOIN review c ON a.order_id=c.order_id
LEFT JOIN product d ON a.product_id=d.product_id
LEFT JOIN category e ON d.product_category_name=e.product_category_name
GROUP BY e.product_category_name_english;



The screenshot shows a SQL query execution interface. The query is as follows:

```
1 # 重要发展用户的热门商品品类
2 SELECT e.product_category_name_english 商品品类,SUM(a.total_price) 消费金额,ROUND(AVG(c.review_score),2) 商品评分,
3 (0.7*SUM(a.total_price)+0.3*10000*ROUND(AVG(c.review_score),7)) 热门指数,
4 rank() over(ORDER BY (0.7*SUM(a.total_price)+0.3*10000*ROUND(AVG(c.review_score),7)) DESC) 热门排名
5 FROM order_detail_1 a
6 INNER JOIN (SELECT customer_id from rfm WHERE 用户类型='重要发展用户' ) as b ON a.customer_id=b.customer_id
7 LEFT JOIN review c ON a.order_id=c.order_id
8 LEFT JOIN product d ON a.product_id=d.product_id
9 LEFT JOIN category e ON d.product_category_name=e.product_category_name
10 GROUP BY e.product_category_name_english;
```

The results are displayed in a table with the following columns: 商品品类, 消费金额, 商品评分, 热门指数, 热门排名. The data is sorted by 热门排名 in descending order.

商品品类	消费金额	商品评分	热门指数	热门排名
watches_gifts	586866.27	3.97	422719.4325000	1
health_beauty	565437.49	4.12	408164.0236000	2
computers_accessory	380377.91	3.88	277918.0391000	3
sports_leisure	372076.73	3.95	272301.4268000	4
bed_bath_table	332313.96	3.88	244273.5231000	5
housewares	300725.40	3.99	222482.1936000	6
furniture_decor	285433.45	3.90	211499.3215000	7
auto	271917.46	4.00	202346.1904000	8
cool_stuff	208492.40	4.03	158022.8276000	9
baby	200442.71	3.74	151541.3021000	10
garden_tools	166238.25	3.96	128251.1286000	11
office_furniture	161870.35	3.54	123935.2954000	12
telephony	126431.56	3.80	99889.4158000	13
toys	114547.80	4.24	92890.3566000	14
perfumery	113951.11	4.20	92374.5790000	15
construction_tools_c	111427.96	4.09	90258.8314000	16
musical_instruments	105487.80	4.19	86405.0964000	17
home_appliances_2	94273.70	4.17	78487.4579000	18

The interface also shows the query execution time: 15510.077s, and the number of records: 第1条记录 (共67条).

重要挽留用户的热门商品品类（运行 4.4h）
SELECT e.product_category_name_english 商品品类 ,SUM(a.total_price) 消费金
额,ROUND(AVG(c.review_score),2) 商品评分,
(0.7*SUM(a.total_price)+0.3*10000*ROUND(AVG(c.review_score),7)) 热门指数,
rank() over(ORDER BY (0.7*SUM(a.total_price)+0.3*10000*ROUND(AVG(c.review_score),7)) DESC)
热门排名
FROM order_detail_1 a
INNER JOIN (SELECT customer_id from rfm WHERE 用户类型='重要挽留用户') as b ON
a.customer_id=b.customer_id
LEFT JOIN review c ON a.order_id=c.order_id
LEFT JOIN product d ON a.product_id=d.product_id
LEFT JOIN category e ON d.product_category_name=e.product_category_name
GROUP BY e.product_category_name_english;

localhost_3306	olist	运行 · 停止 解释
<pre> 1 # 重要挽留用户的热门商品品类 2 SELECT e.product_category_name_english 商品品类,SUM(a.total_price) 消费金额,ROUND(AVG(c.review_score),2) 商品评分, 3 (0.7*SUM(a.total_price)+0.3*10000*ROUND(AVG(c.review_score),7)) 热门指数, 4 rank() over(ORDER BY (0.7*SUM(a.total_price)+0.3*10000*ROUND(AVG(c.review_score),7)) DESC) 热门排名 5 FROM order_detail_1 a 6 INNER JOIN (SELECT customer_id from rfm WHERE 用户类型='重要挽留用户') as b ON a.customer_id=b.customer_id 7 LEFT JOIN review c ON a.order_id=c.order_id 8 LEFT JOIN product d ON a.product_id=d.product_id 9 LEFT JOIN category e ON d.product_category_name=e.product_category_name 10 GROUP BY e.product_category_name_english; </pre>		
信息	结果 1	剖析 状态
商品品类	消费金额	商品评分 热门指数 热门排名
▶ watches_gifts	448398.33	4.20 326473.9398000 1
health_beauty	353362.64	4.16 259819.1945000 2
computers_accessory	337249.01	3.97 247973.3789000 3
cool_stuff	305923.59	4.20 226736.0478000 4
sports_leisure	300073.68	4.08 222303.4596000 5
bed_bath_table	298956.42	3.88 220915.2033000 6
furniture_decor	227948.18	3.77 170865.5162000 7
auto	207715.56	3.89 157082.9730000 8
toys	197149.99	4.10 150300.8329000 9
garden_tools	188490.69	3.97 143868.1692000 10
computers	164938.63	4.16 127944.8458000 11
perfumery	157531.79	4.25 123022.2530000 12
housewares	145750.01	4.01 114054.8876000 13
office_furniture	137570.02	3.58 107046.1924000 14
baby	107936.48	3.74 86761.6898000 15
small_appliances	97527.26	4.02 80330.3066000 16
telephony	87589.41	4.06 73486.6485000 17
musical_instruments	79749.65	4.02 67890.9314000 18

重要挽留用户的热门商品品类 SELECT e.product_category_name_english 商品品类,SUM(a.total_price) 消费金额,ROUND(AVG(c.review_score),2) 商品评分, 只读 查询时间: 15811.995s 第 1 条记录 (共 66 条)

一般发展用户的热门商品品类（运行 4h）

```

SELECT e.product_category_name_english 商品品类 ,SUM(a.total_price) 消费金
额,ROUND(AVG(c.review_score),2) 商品评分,
(0.7*SUM(a.total_price)+0.3*10000*ROUND(AVG(c.review_score),7)) 热门指数,
rank() over(ORDER BY (0.7*SUM(a.total_price)+0.3*10000*ROUND(AVG(c.review_score),7)) DESC)
热门排名
FROM order_detail_1 a
INNER JOIN (SELECT customer_id from rfm WHERE 用户类型='一般发展用户' ) as b ON
a.customer_id=b.customer_id
LEFT JOIN review c ON a.order_id=c.order_id
LEFT JOIN product d ON a.product_id=d.product_id
LEFT JOIN category e ON d.product_category_name=e.product_category_name
GROUP BY e.product_category_name_english;

```

localhost_3306	olist	运行 · 停止 解释
<pre> 1 # 一般发展用户的热门商品品类 2 SELECT e.product_category_name_english 商品品类,SUM(a.total_price) 消费金额,ROUND(AVG(c.review_score),2) 商品评分, 3 (0.7*SUM(a.total_price)+0.3*10000*ROUND(AVG(c.review_score),7)) 热门指数, 4 rank() over(ORDER BY (0.7*SUM(a.total_price)+0.3*10000*ROUND(AVG(c.review_score),7)) DESC) 热门排名 5 FROM order_detail_1 a 6 INNER JOIN (SELECT customer_id from rfm WHERE 用户类型='一般发展用户') as b ON a.customer_id=b.customer_id 7 LEFT JOIN review c ON a.order_id=c.order_id 8 LEFT JOIN product d ON a.product_id=d.product_id 9 LEFT JOIN category e ON d.product_category_name=e.product_category_name 10 GROUP BY e.product_category_name_english; </pre>		
信息	结果 1	剖析 状态
商品品类	消费金额	商品评分 热门指数 热门排名
bed_bath_table	324005.06	3.98 238736.5667000 1
health_beauty	320583.76	4.20 237012.8752000 2
sports_leisure	252514.72	4.17 189264.8398000 3
▶ computers_accessory	230081.44	4.03 173149.2145000 4
furniture_decor	195368.22	4.12 149123.4711000 5
housewares	194224.07	4.21 148583.1692000 6
watches_gifts	185202.35	4.05 141797.2976000 7
auto	134328.93	4.16 106504.6779000 8
stationery	101388.84	4.26 83756.3985000 9
baby	99733.30	4.11 82140.6295000 10
garden_tools	92670.47	4.22 77537.2325000 11
perfumery	91945.03	4.24 77072.4874000 12
telephony	91671.59	3.99 76139.5991000 13
toys	85983.84	4.11 72522.6555000 14
electronics	73711.40	4.10 63891.4163000 15
pet_shop	70806.71	4.22 62225.9873000 16
cool_stuff	66792.01	4.15 59189.5534000 17
fashion_bags_access	49290.21	4.32 47461.3059000 18

一般发展用户的热门商品品类 SELECT e.product_category_name_english 商品品类,SUM(a.total_price) 消费金额,ROUND(AVG(c.review_score),2) 商品评分, 只读 查询时间: 15246.388s 第 4 条记录 (共 71 条)

一般挽留用户的热门商品品类（运行 4.3h）


```

SELECT e.product_category_name_english 商品品类,SUM(a.total_price) 消费金额,ROUND(AVG(c.review_score),2) 商品评分,
(0.7*SUM(a.total_price)+0.3*10000*ROUND(AVG(c.review_score),7)) 热门指数,
rank() over(ORDER BY (0.7*SUM(a.total_price)+0.3*10000*ROUND(AVG(c.review_score),7)) DESC)
热门排名
FROM order_detail_1 a
INNER JOIN (SELECT customer_id from rfm WHERE 用户类型='一般挽留用户') as b ON
a.customer_id=b.customer_id
LEFT JOIN review c ON a.order_id=c.order_id
LEFT JOIN product d ON a.product_id=d.product_id
LEFT JOIN category e ON d.product_category_name=e.product_category_name
GROUP BY e.product_category_name_english;

```

商品品类	消费金额	商品评分	热门指数	热门排名
bed_bath_table	305900.60	3.99	226096.5518000	1
sports_leisure	238464.08	4.27	179724.9287000	2
health_beauty	206026.63	4.17	156713.7526000	3
furniture_decor	202851.00	4.03	154077.2748000	4
toys	164591.75	4.22	127863.3188000	5
housewares	142574.39	4.14	112231.5239000	6
cool_stuff	140851.42	4.22	111263.4046000	7
garden_tools	137434.91	4.18	108742.2296000	8
computers_accesso	129257.55	4.12	102841.5681000	9
perfumery	92169.15	4.14	76927.7406000	10
telephony	89701.71	4.03	74868.6942000	11
watches_gifts	84749.84	4.04	71442.5351000	12
baby	74496.25	4.23	64823.5765000	13
auto	74160.45	4.13	64290.1653000	14
fashion_bags_acces	63038.60	4.12	56487.6758000	15
stationery	61651.68	4.29	56035.4862000	16
pet_shop	50189.46	4.31	48059.4960000	17
[Null]	50681.14	3.94	47304.7051000	18

结果分析：

- 1）重要发展用户中排名 TOP10 的热门商品为 watches_gifts（手表礼物）、health_beauty（健美）、computers_accessories（电脑配件）、sports_leisure（运动休闲）、bed_bath_table（床）、housewares（厨房用品）、furniture_decor（家具装饰）、auto（汽车）、cool_stuff（潮玩）、baby（母婴）
- 2）重要挽留用户中排名 TOP10 的热门商品为 watches_gifts（手表礼物）、health_beauty（健美）、computers_accessories（电脑配件）、cool_stuff（潮玩）、sports_leisure（运动休闲）、bed_bath_table（床）、furniture_decor（家具装饰）、auto（汽车）、toys（玩具）、garden_tools（花园用具）
- 3）一般发展用户中排名 TOP10 的热门商品为 bed_bath_table（床）、health_beauty（健美）、sports_leisure（运动休闲）、computers_accessories（电脑配件）、furniture_decor（家具装饰）、housewares（厨房用具）、watches_gifts（手表礼物）、auto（汽车）、stationery（文具）、baby（母婴）
- 4）一般挽留用户中排名 TOP10 的热门商品为 bed_bath_table（床）、sports_leisure（运动休闲）、health_beauty（健美）、furniture_decor（家具装饰）、toys（玩具）、housewares（厨房用品）、cool_stuff（潮玩）、garden_tools（花园用具）、computers_accessories（电脑配件）、perfumery（香水）

5) 各类型用户的热门商品品类中, 健美产品、运动休闲等品类均出现在热门排名前 5, 说明这类商品是大众热门品类, 适合对各类用户做推广和促销, 而手表和家居用品则属于小众热门商品, 对特定类型用户来说更受欢迎, 适合选择合适的用户群体进行推荐。同时还要注意长尾效应, 一些小众的用户可能感兴趣的产品也需要推荐给用户。

二、基于 python+sql+tableau

先在 jupyter notebook 中对数据进行预处理, 再导入 mysql 中, 通过 tableau 连接 mysql 进行数据分析。

围绕人货场三个角度进行分析, 人包括客户和商家, 货为商品的销量和销售额, 场为销售额、订单量和客单价。

整体分为数据导入与合并、数据预处理、数据的导出。

一、客户角度

1、数据导入与合并

(1) 导入相关库

```
import numpy as np # 科学计算工具包
import pandas as pd # 数据分析工具包
import matplotlib.pyplot as plt # 图表绘制工具包
import seaborn as sns # 基于 matplotlib 的图形可视化 python 包
from sqlalchemy import create_engine # 导入数据库
```

(2) 处理格式乱码

```
plt.rcParams['font.sans-serif'] = ['SimHei'] # 中文字体设置一黑体
plt.rcParams['axes.unicode_minus'] = False # 解决保存图像是负号 '-' 显示为方块的问题
sns.set(font='SimHei') # 解决 seaborn 中文显示问题
```

(3) pandas 调整显示的行数列数

```
pd.set_option('display.max_columns', 1000)
pd.set_option('display.width', 1000)
pd.set_option('display.max_colwidth', 1000)
```

(4) 获取数据集

```
orders = pd.read_csv('../olist_orders_dataset.csv')
payments = pd.read_csv('../olist_order_payments_dataset.csv')
customers = pd.read_csv('../olist_customers_dataset.csv')
reviews = pd.read_csv('../olist_order_reviews_dataset.csv')
items = pd.read_csv('../olist_order_items_dataset.csv')
products = pd.read_csv('../olist_products_dataset.csv')
seller = pd.read_csv('../olist_sellers_dataset.csv')
geolocation = pd.read_csv('../olist_geolocation_dataset.csv')
pro_trans = pd.read_csv('../product_category_name_translation.csv')
```

(5) 合并有联系的数据表 (订单表、付款表、顾客表、下单产品表)

```
order_payment = pd.merge(orders, payments, on='order_id', how='left')
```

```
payorder_customer = pd.merge(order_payment, customers, on='customer_id', how='left')
merge_data = pd.merge(payorder_customer, items, on='order_id', how='left')
```

merge_data.info()#查看数据基本信息

merge_data.head()#预览前 5 行的数据

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 118434 entries, 0 to 118433
Data columns (total 22 columns):
#   Column                                Non-Null Count  Dtype
---  -
0   order_id                             118434 non-null object
1   customer_id                           118434 non-null object
2   order_status                           118434 non-null object
3   order_purchase_timestamp               118434 non-null object
4   order_approved_at                      118258 non-null object
5   order_delivered_carrier_date           116360 non-null object
6   order_delivered_customer_date          115037 non-null object
7   order_estimated_delivery_date          118434 non-null object
8   payment_sequential                     118431 non-null float64
9   payment_type                           118431 non-null object
10  payment_installments                   118431 non-null float64
11  payment_value                           118431 non-null float64
12  customer_unique_id                     118434 non-null object
13  customer_zip_code_prefix                118434 non-null int64
14  customer_city                           118434 non-null object
15  customer_state                           118434 non-null object
16  order_item_id                           117604 non-null float64
17  product_id                             117604 non-null object
18  seller_id                               117604 non-null object
19  shipping_limit_date                     117604 non-null object
20  price                                  117604 non-null float64
21  freight_value                           117604 non-null float64
dtypes: float64(6), int64(1), object(15)
memory usage: 20.8+ MB
```

Out[4]:

	order_id	customer_id	order_status	order_purchase_timestamp	order_approved_at	order_delivered_carrier_d
0	e481f51cbdc54678b7cc49136f2d6af7	9ef432eb6251297304e76186b10a928d	delivered	2017-10-02 10:56:33	2017-10-02 11:07:15	2017-10-04 19:55
1	e481f51cbdc54678b7cc49136f2d6af7	9ef432eb6251297304e76186b10a928d	delivered	2017-10-02 10:56:33	2017-10-02 11:07:15	2017-10-04 19:55
2	e481f51cbdc54678b7cc49136f2d6af7	9ef432eb6251297304e76186b10a928d	delivered	2017-10-02 10:56:33	2017-10-02 11:07:15	2017-10-04 19:55
3	53cdb2fc8bc7dce0b6741e2150273451	b0830fb4747a6c6d20dea0b8c802d7ef	delivered	2018-07-24 20:41:37	2018-07-26 03:24:27	2018-07-26 14:31
4	47770eb9100c2d0c44946d9cf07ec85d	41ce2a54c0b03bf3443c3d931a367089	delivered	2018-08-08 08:38:49	2018-08-08 08:55:23	2018-08-08 13:50

2、数据预处理

(2) 缺失值处理 (本案例采用删除缺失值的操作, 可以采用填充缺失值为 0, 或者用均值/最值来填充缺失值)

```
merge_data = merge_data.dropna()
```

(2) 异常值处理

merge_data.describe()# 观察数据基本情况

#分析:

#a.payment_installments 的最小值为 0, 属于异常值, 分期付款最小值应该为 1;

#b.payment_value 和 freight_value 为 0 数以异常值，成功下单的商品价格和运费不应该为 0，但可能存在用户使用代金券等形式。

```
merge_data[merge_data['payment_installments']==0]
```

#观察可知，payment_installments 为 0 时，其对应的 payment_value 不为 0，说明该三条数据为异常数据，需删除

```
merge_data[merge_data['payment_value']==0]
```

#观察可知，其支付方式 payment_type='voucher',即采用代金券支付的方式，故 payment_value 为 0 是正常现象

#删除异常值

```
merge_data = merge_data.drop(index=merge_data[merge_data['payment_installments']==0].index)
```

（3）重复值处理

```
merge_data.duplicated().sum()
```

#观察可知，检索的重复值为 0，故不需要删除重复值

（4）数据类型转换

#通过查看数据基本信息，可以看出时间类型的字段对应的数据类型为 object，应更改成 datetime 类型

```
merge_data.info()
```

```
def transform_datetime(data, colum_list):
```

```
    for i in colum_list:
```

```
        data[i] = pd.to_datetime(data[i])
```

```
    print('数据类型转化成功！')
```

```
colum_list = ['order_purchase_timestamp', 'order_approved_at',
```

```
              'order_delivered_carrier_date', 'order_delivered_customer_date',
```

```
              'order_estimated_delivery_date']
```

```
transform_datetime(merge_data,colum_list)
```



```

<class 'pandas.core.frame.DataFrame'>
Int64Index: 115015 entries, 0 to 118433
Data columns (total 22 columns):
#   Column                                     Non-Null Count  Dtype
---  -
0   order_id                                  115015 non-null  object
1   customer_id                              115015 non-null  object
2   order_status                             115015 non-null  object
3   order_purchase_timestamp                 115015 non-null  datetime64[ns]
4   order_approved_at                       115015 non-null  datetime64[ns]
5   order_delivered_carrier_date             115015 non-null  datetime64[ns]
6   order_delivered_customer_date           115015 non-null  datetime64[ns]
7   order_estimated_delivery_date            115015 non-null  datetime64[ns]
8   payment_sequential                      115015 non-null  float64
9   payment_type                            115015 non-null  object
10  payment_installments                    115015 non-null  float64
11  payment_value                           115015 non-null  float64
12  customer_unique_id                      115015 non-null  object
13  customer_zip_code_prefix                115015 non-null  int64
14  customer_city                           115015 non-null  object
15  customer_state                           115015 non-null  object
16  order_item_id                           115015 non-null  float64
17  product_id                              115015 non-null  object
18  seller_id                               115015 non-null  object
19  shipping_limit_date                     115015 non-null  object
20  price                                    115015 non-null  float64
21  freight_value                           115015 non-null  float64
dtypes: datetime64[ns](5), float64(6), int64(1), object(10)
memory usage: 20.2+ MB
数据类型转化成功!

```

经过数据预处理，数据由一开始的 **118434** 条数据变成 **115015** 条数据。

3、连接 MySQL 数据库，并将预处理之后的数据导入数据库中，再用 Tableau 连接数据库进行可视化分析

```

def export_mysql(data, user, password, host_port, db, table_name):
    print('将清洗后的数据导出到 mysql')
    engine = create_engine("mysql+pymysql://{}:{}_{}@{}/{}".format(user, password, host_port, db))
    con = engine.connect() # 创建连接
    data.to_sql(table_name, engine, if_exists='replace', index=False)
    print('导出成功!')

```

```

user = 'root'
password = '1234'
host_port = 'localhost:3306'
db = 'olist2'
data = merge_data
table_name = 'merge_data'
export_mysql(data, user, password, host_port, db, table_name)

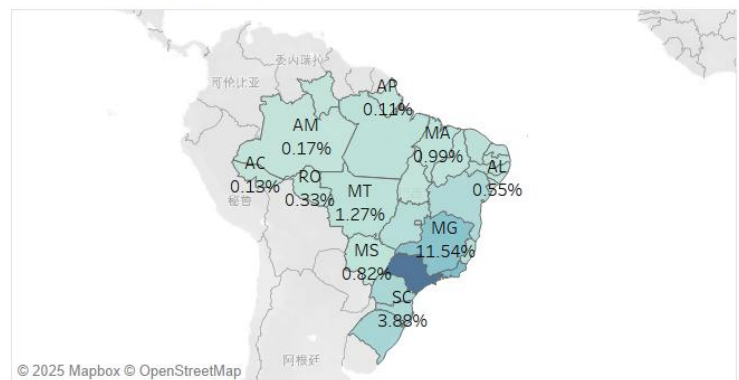
```

将清洗后的数据导出到mysql
导出成功!

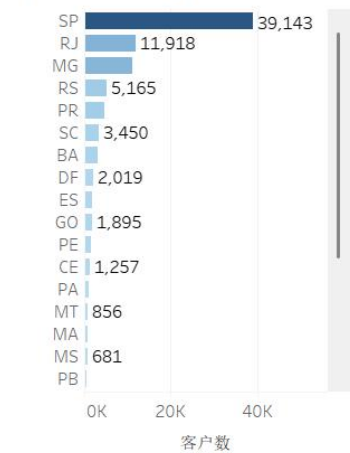
将 mysql 数据库 olist2 中的 merge_data 数据表与 tableau 进行连接，对客户信息进行基

本数据的可视化和行为偏好的可视化。

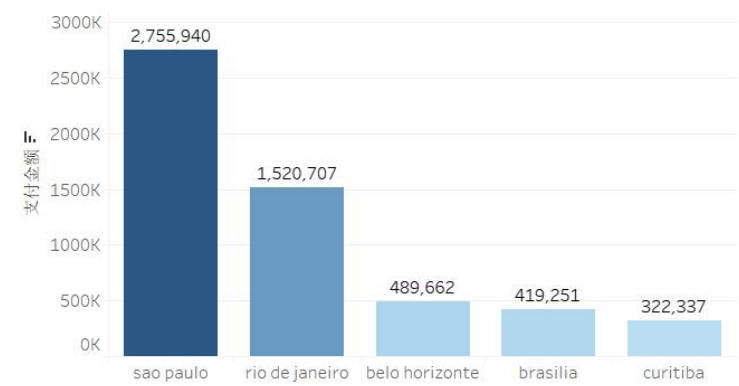
各州客户数分布情况



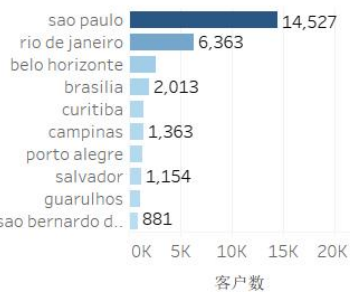
各州客户数排名



购买力排名TOP5的城市



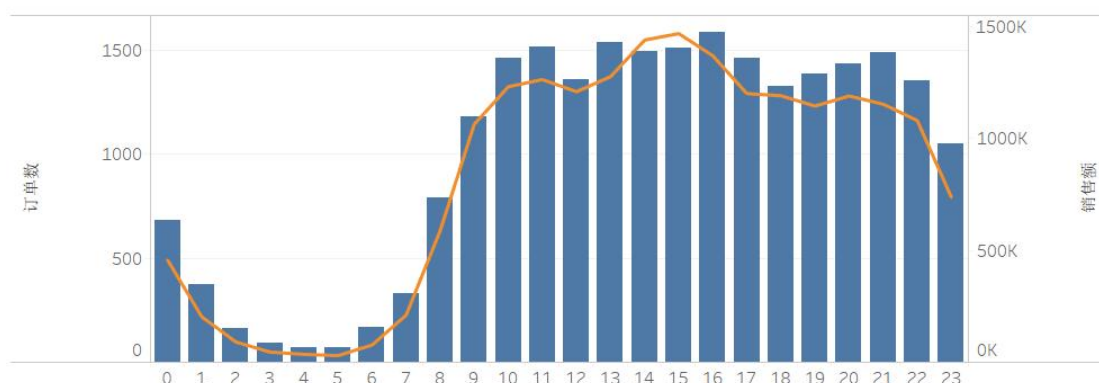
客户数排名TOP10的城市



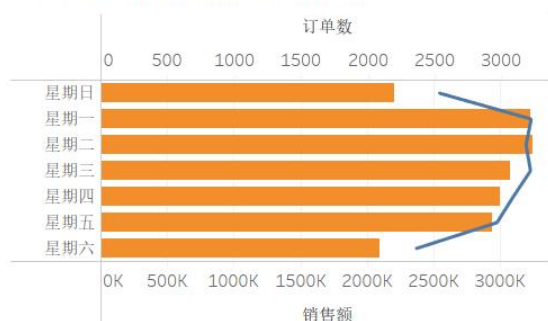
结果分析:

- 1) 客户地理位置分布情况: olist2 客户主要集中上图可视化的州和城市, 其中巴西圣保罗 (SP)、里约热内卢 (RJ)、米纳斯吉拉斯州 (MG) 这三个州的客户数位于前列, 巴西圣保罗 (SP) 是客户数最多的州, 占客户数的 37.44%, 而客户数分布最多的城市是巴西圣保罗 (SP) 的 sao paulo (圣保罗), 有 14527 为客户数量。
- 2) 城市购买力排名: 购买力排名前三强为 sao paulo (圣保罗)、rio de janeiro (约热内卢)、belo horizonte (贝洛奥里藏特)。
- 3) 购买力排名靠前的核心城市和州需重点维护, 同时对排后的城市和州也要进行重点发掘, 对不同重要程度的城市和州采取不同的营销策略。
- 4) 对卖家核心的城市和州加快物流建设, 打造电子商务的核心区域, 同时在卖家集中的区域进一步完善供应链建设, 降低产品成本, 提高电子商务的价格优势。

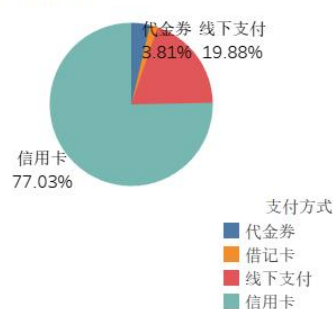
各时段订单数和销售额



周订单数和销售额分布情况



客户付款方式偏好



结果分析:

1) 从各时段订单量和销售额图中可以观察到客户下单时间主要集中在早上 9 点到晚上 22 点。从周订单量和销售额分布情况图中可以观察到工作日的订单量和销售额表现优于周末, 说明该平台的用户更倾向于在下班时间段和工作日进行购物。

2) 从客户付款方式偏好的饼状图中可以观察到四种最常用的客户付款方式, 即代金券、借记卡、信用卡、线下支付。其中信用支付, 占比高达 77.03%, 说明大部分用户都是透支消费, 其次是通过线下付款方式, 占比高达 19.88%, 代金券和借记卡分别占比 3.81% 和 1.54%。

3) 在购买旺盛的时段, 可以采取折扣券、包邮、产品组合等优惠形式吸引用户消费。

二、商家角度

1、合并数据库

```
review_data=pd.merge(merge_data,reviews,on='order_id',how='left')
review_data = review_data[['order_id','customer_unique_id','order_purchase_timestamp',
'review_id','review_score','review_creation_date','review_answer_timestamp']]
review_data.info()
```

```

<class 'pandas.core.frame.DataFrame'>
Int64Index: 115708 entries, 0 to 115707
Data columns (total 7 columns):
#   Column                                Non-Null Count  Dtype
---  -
0   order_id                             115708 non-null object
1   customer_unique_id                   115708 non-null object
2   order_purchase_timestamp             115708 non-null datetime64[ns]
3   review_id                            115708 non-null object
4   review_score                         115708 non-null int64
5   review_creation_date                 115708 non-null object
6   review_answer_timestamp              115708 non-null object
dtypes: datetime64[ns](1), int64(1), object(5)
memory usage: 7.1+ MB

```

2、数据预处理

(1) 缺失值处理

```
review_data = review_data.dropna()
```

(2) 异常值处理

```
review_data.describe()
```

(3) 重复值处理

```
review_data.duplicated().sum()
```

```
review_data=review_data.drop_duplicates()
```

```
review_data.info()
```

```

<class 'pandas.core.frame.DataFrame'>
Int64Index: 96995 entries, 0 to 115707
Data columns (total 7 columns):
#   Column                                Non-Null Count  Dtype
---  -
0   order_id                             96995 non-null object
1   customer_unique_id                   96995 non-null object
2   order_purchase_timestamp             96995 non-null datetime64[ns]
3   review_id                            96995 non-null object
4   review_score                         96995 non-null int64
5   review_creation_date                 96995 non-null object
6   review_answer_timestamp              96995 non-null object
dtypes: datetime64[ns](1), int64(1), object(5)
memory usage: 5.9+ MB

```

经过数据预处理，从原本 115708 条数据变成 96995 条数据。

(4) 数据类型转化

```
column_list=['order_purchase_timestamp','review_creation_date','review_answer_timestamp']
```

```
transform_datetime(review_data,column_list)
```

#3、导入数据库

```
user = 'root'
```

```
password = '1234'
```

```
host_port = 'localhost:3306'
```

```
db = 'olist2'
```

```
data = review_data
```

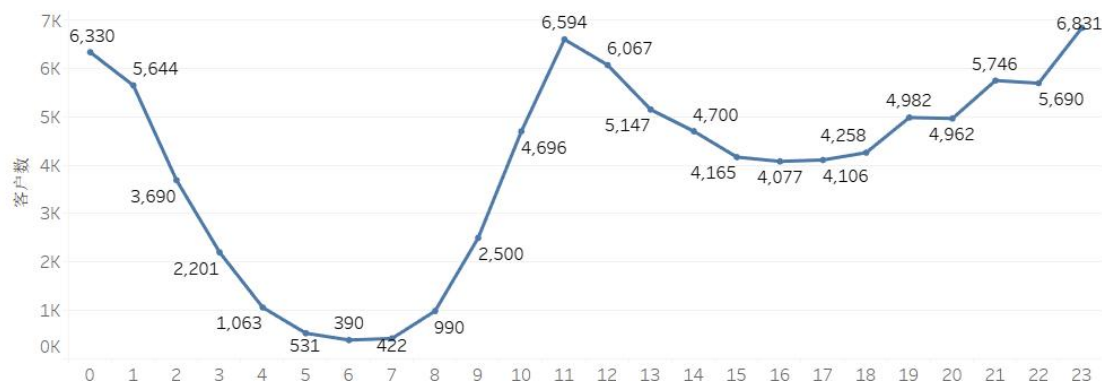
```
table_name = 'review_data'
```

```
export_mysql(data, user, password, host_port, db, table_name)
```

将清洗后的数据导出到mysql 导出成功!

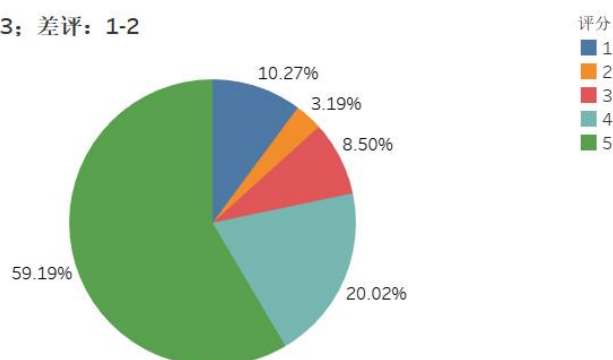
将 olist2 数据库与 tableau 连接，从商家角度分析用户的满意度。

客户满意度填写时间段分析



客户满意度评分占比

【评分标准】好评：4-5；中评：3；差评：1-2



结果分析：

1) 通过客户满意度填写时间段分析图中可以观察到，大部分用户会在中午 11 点、12 点以及晚上 23 点、24 点填写满意度调查问卷，说明用户倾向于在吃饭期间和睡觉前进行满意度的填写，因此该平台在下发满意度调查的时候可以在这两个时间段进行下发，该时间段正好是客户放松时间段，会问卷的填写不会产生厌烦且具有一定真实性。

2) 从客户满意度评分占比的饼状图中以（1-2 分：差评，3 分：中评，4-5 分：好评）为评价标准。参与满意度调查的客户数一共有 4,213 名，可以观察到客户对该电商平台满意度较高，好评率（4-5 分）近似 80%。但 1 分和 2 分合计占比 13.6% 左右，平台还有一定的提高空间。可以重点关注差评客户的评论，以此作为突破口进行平台服务的改进。

三、商品角度

1、合并相关数据集

```
data1 = pd.merge(merge_data,products, on='product_id',how='left')
```

```
data2 = pd.merge(data1,pro_trans, on='product_category_name',how='left')
```

```
product_data = data2[['order_id','order_item_id','customer_unique_id','product_id','product_category_name_english','price','freight_value','payment_value','order_status','order_purchase_timestamp']]
```



```
product_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 115015 entries, 0 to 115014
Data columns (total 10 columns):
#   Column                                Non-Null Count  Dtype
---  -
0   order_id                             115015 non-null object
1   order_item_id                         115015 non-null float64
2   customer_unique_id                   115015 non-null object
3   product_id                           115015 non-null object
4   product_category_name_english        113365 non-null object
5   price                                115015 non-null float64
6   freight_value                        115015 non-null float64
7   payment_value                        115015 non-null float64
8   order_status                         115015 non-null object
9   order_purchase_timestamp              115015 non-null datetime64[ns]
dtypes: datetime64[ns](1), float64(4), object(5)
memory usage: 9.7+ MB
```

2、数据预处理

(1) 缺失值处理

```
product_data=product_data.dropna()
```

(2) 异常值处理

```
product_data.describe()
```

(3) 重复值处理

```
product_data.duplicated().sum()#672
```

```
product_data = product_data.drop_duplicates()
```

```
product_data.info()
```

(4) 数据类型转换

```
column_list=['order_purchase_timestamp']
```

```
transform_datetime(product_data,column_list)
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 112722 entries, 0 to 115014
Data columns (total 10 columns):
#   Column                                Non-Null Count  Dtype
---  -
0   order_id                             112722 non-null object
1   order_item_id                         112722 non-null float64
2   customer_unique_id                   112722 non-null object
3   product_id                           112722 non-null object
4   product_category_name_english        112722 non-null object
5   price                                112722 non-null float64
6   freight_value                        112722 non-null float64
7   payment_value                        112722 non-null float64
8   order_status                         112722 non-null object
9   order_purchase_timestamp              112722 non-null datetime64[ns]
dtypes: datetime64[ns](1), float64(4), object(5)
memory usage: 9.5+ MB
```

经过数据预处理，从原本 115015 条数据变成 112722 条数据。

3、导入数据库

```
user = 'root'
```

```
password = '1234'
```

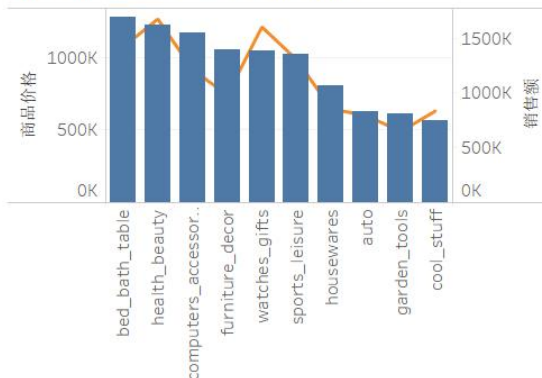
```
host_port = 'localhost:3306'
```

```
db = 'olist2'
data = product_data
table_name = 'product_data'
export_mysql(data, user, password, host_port, db, table_name)
```

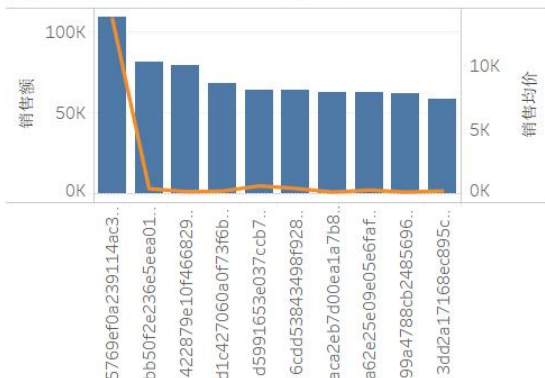
将清洗后的数据导出到mysql
导出成功！

将 olist2 与 tableau 进行连接，对商品数据进行可视化分析。

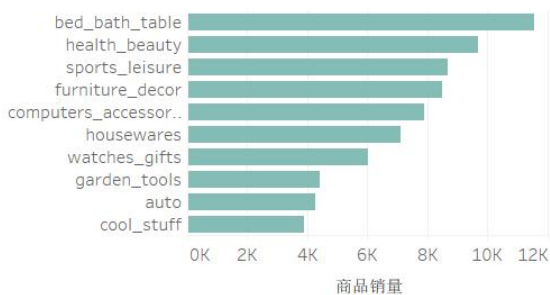
销售额排名TOP10 的商品品类



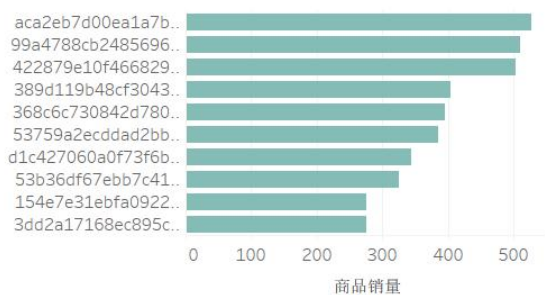
销售额排名TOP10 的商品



销量排名TOP10的商品品类



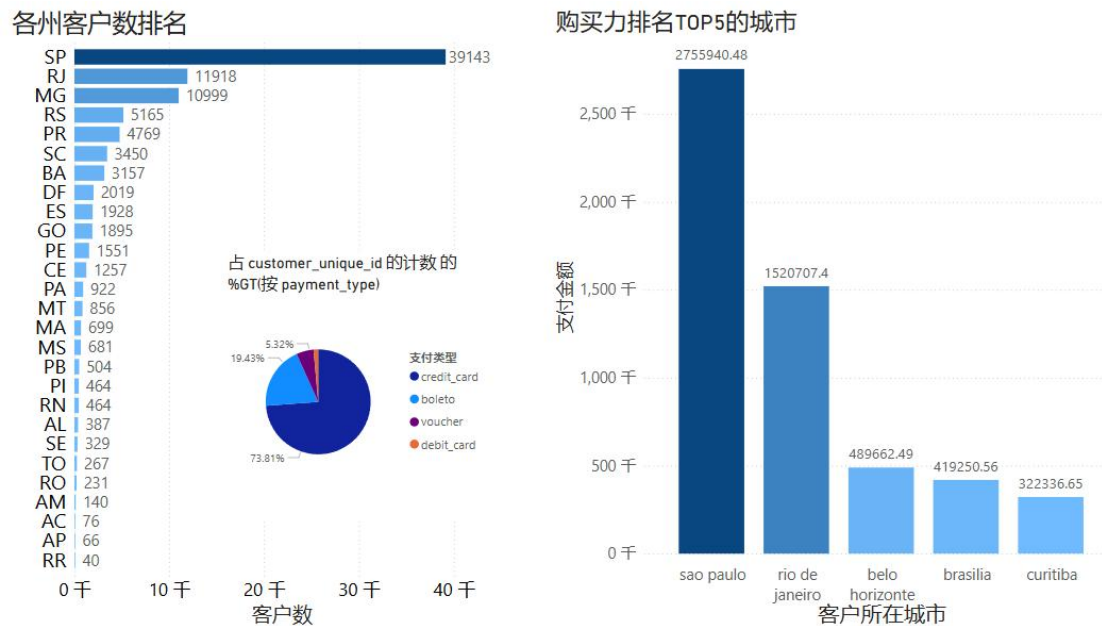
销量排名TOP10 的商品



结果分析：

通过上述四张图可以观察到无论是销量还是销售额，热销的商品主要是床上用品、运动用品、家具类和电脑等电子产品。以上商品品类需要好好维护，做好质量保障。同时也要观察哪些商品是比较小众的，通过推荐算法推荐给可能感兴趣的客户，提高各类商品的销售量。

Powerbi 中的部分可视化绘图



对比 powerbi 和 tableau 有感:

同样的数据表在绘图的时候, 感觉在 tableau 中更加方便一些。

1. 在更改横纵坐标的时候, tableau 会更加方便一些, 双击选中即可更改, powerbi 则需要在右侧设置视觉对象格式-视觉对象-x 轴-标题, 一层层打开才能进行更改。

2. 不同深浅颜色的更改上, tableau 只需要拖拽凸显颜色深浅的字段值到标记-颜色即可, powerbi 则需要在右侧设置视觉对象格式-视觉对象-数据标签-值-颜色旁边的 fx 中设置格式样式为渐变, 再选择哪个字段, 以及具体的最深的颜色和最浅的颜色。如果采用系统默认的最深、最浅的颜色效果不是很明显, 还是需要自行选择颜色的, 而 tableau 则采用系统默认的颜色即可。

3. 在图例中, tableau 不仅可以修改图例的标题, 还可以修改图标的名称 (因为有的数据集字段采用的是英文, 需要更改成中文), 而在 powerbi 中只能更改图例的标题, 暂且没找到哪个选项可以更改图标。

4. 在横纵坐标的刻度单位中, tableau 的单位是用英文 k 表示千, w 表示万, 但在 powerbi 中只能用中文的千、万。

不过在绘制仪表板的时候, powerbi 会方便一些, 可以直接将好几张图绘制在一张仪表板中, 而 tableau 需要一个新建工作表, 再将工作表拖拽到仪表板中。