DataFrame questions

Data file: customers.csv

```
+----------+--------+--------+-------+--------+-----+
|      date|    time|customer|product|quantity|price|
+----------+--------+--------+-------+--------+-----+
|01/10/2018| 2:20 PM|     100|      1|       6|   86|
|04/08/2018|11:38 AM|     200|      2|       8|   79|
```

```
df=spark.read.option(“header”,True).option(“inferSchema”
,True).csv(“/home/alumno/Descargas/customers.csv”)
```

1. How many elements?
2. 2-How many DISTINCT customers?

1-How many elements?

need DataFrame global aggregations as question refers to global view

```
c.selectExpr("count(customer)").show
c.count()
val totalNum=c.count()
```

```
+---------------+
|count(customer)|
+---------------+
|           1000|
+---------------+
```

2-How many DISTINCT customers?

Global view again, we need to focus on **customers column**

```
c.selectExpr("count(distinct(customer))").show
```

```
+------------------------+
|count(DISTINCT customer)|
+------------------------+
|                      31|
+------------------------+
```

3-How many products per customer?

Aggregation question: need to aggregate values per customer

which is the relevant key?

which is the calculation we need?

3-How many products per customer?

Aggregation question: need to aggregate values per customer

which is the relevant key?

customer

which is the calculation we need?

add all product quantities for each customer transaction

3-How many products per customer?

Aggregation question: need to aggregate values per customer

which is the relevant key?

customer

which is the calculation we need?

add all quantities

aggregation expression that we need?

```
c.groupBy("customer").AGGREGATION-TO-DO
```

3-How many products per customer?

Aggregation question:

which is the relevant key?

```
customer
```

which is aggregation expression that we need?

```
sum("quantity")
```

```
from pyspark.sql.functions import expr
c.groupBy("customer").agg(expr("sum(quantity)")).show(5)
```

3-How many products per customer?

```
c.groupBy("customer").agg(expr("sum(quantity)"))
.show(5)
```

```
+----------+-------------+
|  customer|sum(quantity)|
+----------+-------------+
|       101|          196|
|       100|          148|
|       121|          146|
+----------+-------------+
```

4-Sort customers by quantity

sort the whole table:

```
c.orderBy("quantity").show
```

Use only needed columns

```
val sorted=c.select("customers", "quantity").
        orderBy("quantity").show
```

why is the second a better answer? think about data volume

5-how many times customer id number 100 has purchased more than 5 items?

key? customer id = 100

conditions to meet for customer 100 transactions are:

quantity > 5

Then, count how many times this has happened in dataset

5-how many times customer id number 100 has purchased more than 5 items?

key: customer = 100
conditions are

a) quantity > 5
b) count how many times this has happened

Designing our **data flow**
1:filter customer=100
2:filter quantity>5
3:counting how many elements in step 2: key? function?
key: customer, function: count

5-how many times customer id number 100 has purchased more than 5 items?

```
c.where(“customer=100”).where(“quantity>5”).show
```

5-how many times customer id number 100 has purchased more than 5 items?

```
c.where(“customer=100”).where(“quantity>5”).show

c.select(“customer”,”quantity”).
   where(“customer = 100”).
   where(“quantity > 5”).show
```

Why is the last a better solution?

5-**how many times** customer id number 100 has purchased more than 5 items?

```
c.select("customer","quantity").
   where("customer = 100").
   where("quantity > 5").
   groupBy("customer").
   count().show
```

```
+----------+-----+
|  customer|count|
+----------+-----+
|       100|   33|
+----------+-----+
```

5-how many times customer id number 100 has purchased more than 5 items?

```
c.select(“customer”,”quantity”).
   where(“customer = 100”).
   where(“quantity > 5”).
   groupBy(“customer”).
   count().show
```

Data flow:

Select columns -> filter by customer/quantity -> groupBy + count

6-which were the products bought by customer with the **transaction with the largest number of products**?

Two questions to solve:

1-which is the customer with largest number of products in a transaction?

key: customer, value: max quantity

2-which are the products of selected customer?

key customer:200, value: product id list

6- max(quantity)

```
c.groupBy("customer").agg(expr("max(quantity)"))
.show
+----------+-------------+
|  customer|max(quantity)|
+----------+-------------+
|       121|            9|
|       111|            9|
|       100|           10|
+----------+-------------+
c.select("product").
  where(expr("customer = 100")).show
+-------+
|product|
+-------+
|      5|
|      8|
```

BONUS:
which is the product with highest price?

BONUS:
which is the product with highest price?

```
c.groupBy("product").agg(max("price")).
sort("max(price)").first: ERROR
```

Data flow:

1. groupBy product
2. aggregate maximum price
3. sort prices
4. get first

BONUS:
which is the product with highest price?

```
c.groupBy("product").
    agg(max("price").alias("max_price")).
    orderBy(desc("max_price")).show
```

```
+-------+---------+
| product|max_price|
+-------+---------+
|       1|      816|
|       7|       99|
|       3|       99|
+-------+---------+
```

BONUS:
which is the product with highest price?

```
Alternative solution:
c.groupBy("product").orderBy("price").show

c.groupBy("product").
    agg(max("price").alias("max_price")).
    orderBy("max_price").first
```

which is the best solution?

```
c.groupBy("product").orderBy("price")
c.groupBy("product").agg(max("price"))
```

Think in scale: 100 Million products. What is more expensive to do?

- sort: result is 100 Million values
- finding maximum: result is 1 value