Spark SQL

WinStar 陈光跃

Spark SQL是Spark用来操作结构化和半结构化数据的接口。结构化数据是指任 何有结构信息的数据。所谓结构信息，就是每条记录共用的已知的字段集合。当数据符合 这样的条件时，Spark SQL就会使得针对这些数据的读取和查询变得更加简单高效。具体 来说，Spark SQL 提供了以下三大功能(见图 9-1)。

(1) Spark SQL 可以从各种结构化数据源(例如 JSON、Hive、Parquet 等)中读取数据。

(2)Spark SQL不仅支持在Spark程序内使用SQL语句进行数据查询，也支持从类似商业 智能软件 Tableau 这样的外部工具中通过标准数据库连接器(JDBC/ODBC)连接 Spark SQL 进行查询。

(3)当在Spark程序内使用Spark SQL时，Spark SQL支持SQL与常规的Python/Java/Scala 代码高度整合，包括连接 RDD 与 SQL 表、公开的自定义 SQL 函数接口等。这样一来， 许多工作都更容易实现了。

Hive 是将MapReduce进程封装成了SQL语句，我们称它为HQL。而Spark在DataFrame出现后，出现了Spark SQL。他的出现目的与Hive比较相似，都是为了使用者能够更简单的处理数据，使用大数据框架。

```
#Spark SQL 库
import org.apache.spark.sql.catalyst.encoders.ExpressionEncoder
import org.apache.spark.sql.Encoder
#为了实现数据隐示转换
import spark.implicits._

case class RecordNeed(activity_id:Int, coupon_templet_id:String, create_time:String,
discount_amount:Int,
item_id:String,item_total_value:Int,pay_price:Double,pay_status:Int,pay_type:Int,refu
nd:Int,send_status:Int,status:Int,auth_driver_license:String)

#创建DataFrame
val recordDF = spark.sparkContext.textFile("file:///home/cgy/Downloads/Winstar/
order/oader.txt").map(_.split(",",-1)).map(attributes =>
RecordNeed(attributes(2).trim.toInt, attributes(5), attributes(6).trim,
attributes(7).trim.toInt, attributes(11), attributes(12).trim.toInt,
attributes(16).trim.toDouble, attributes(17).trim.toInt, attributes(19).trim.toInt,
attributes(21).trim.toInt, attributes(23).trim.toInt, attributes(25).toInt,
attributes(29))).toDF()

#创建视图
recordDF.createOrReplaceTempView("records")
```

pingguo — sudo ssh 192.168.56.101 — 192.168.56.101 — ssh ◂ sudo — 179×46

```
scala>

scala> recordDF.show
+-----------+----------------+------------------+-------------+--------------------+----------------+---------+----------+--------+------+-----------+------+--------------------+
|activity_id|coupon_templet_id|       create_time|discount_amount|             item_id|item_total_value|pay_price|pay_status|pay_type|refund|send_status|status|auth_driver_license|
+-----------+----------------+------------------+-------------+--------------------+----------------+---------+----------+--------+------+-----------+------+--------------------+
|        101|             101|2018/6/20 00:11:53|           50|                  11|             100|     50.0|         1|     106|     0|          3|     3|                    |
|        101|             101|2018/6/20 00:48:42|           50|                  12|             200|    150.0|         1|     106|     0|          3|     3| 612133197910273914 |
|          2|               3|2018/6/20 00:57:23|            0|                   6|             300|    288.0|         1|       2|     0|          3|     3|                    |
|          2|               2|2018/6/20 01:44:46|            0|                   5|             200|    192.0|         1|       2|     0|          3|     3|                    |
|        101|             101|2018/6/20 02:17:30|           50|                  11|             100|     50.0|         1|     106|     0|          3|     3| 610502198408248019 |
|        101|             101|2018/6/20 04:57:52|           50|                  12|             200|    150.0|         1|     106|     0|          3|     3| 610121198504133075 |
|          1|                |2018/6/20 05:57:45|            0|50db459b612b082d0...|             500|    432.0|         1|     106|     0|          3|     3| 220502197102221211 |
|          1|                |2018/6/20 06:32:56|            0|50db459b612b082d0...|             500|    432.0|         1|     106|     0|          3|     3| 610481199205103818 |
|          3|               4|2018/6/20 06:57:48|           20|                  10|             200|    180.0|         1|     106|     0|          3|     3| 612722198302042766 |
|          2|               2|2018/6/20 07:16:39|            0|                   5|             200|    192.0|         1|     106|     0|          3|     3| 612133197304010014 |
|          2|               2|2018/6/20 07:19:46|            0|                   5|             200|    192.0|         1|       2|     0|          3|     3| 61042219841028369X |
|        101|             101|2018/6/20 07:32:48|           50|                  11|             100|     50.0|         1|     106|     0|          3|     3| 610502198110130 43X |
|          2|               1|2018/6/20 07:32:57|            0|                   4|             100|     96.0|         1|       2|     0|          3|     3| 610524199204247 21X |
|          2|               2|2018/6/20 07:33:08|            0|                   5|             200|    192.0|         1|       2|     0|          3|     3| 610323199001254252 |
|          2|               2|2018/6/20 07:41:31|            0|                   5|             200|    192.0|         1|     106|     0|          3|     3| 612101197211091010 |
|        101|             101|2018/6/20 07:46:45|           50|                  11|             100|     50.0|         1|     106|     0|          3|     3| 610429199612150018 |
|          1|                |2018/6/20 08:05:27|            0|50db459b612b082d0...|             500|    432.0|         1|     106|     0|          3|     3| 612133196901098638 |
|          3|               4|2018/6/20 08:18:09|           20|                   9|             100|     80.0|         1|     106|     0|          3|     3| 610523199103253673 |
|          2|               3|2018/6/20 08:21:16|            0|                   6|             300|    288.0|         1|       2|     0|          3|     3|                    |
```

所选取的数据

WinStar 陈光跃

```
 val personDF = spark.sql("select auth_driver_license as
person,sum(item_total_value-pay_price) as count from records where refund = 0
and pay_status =1 group by auth_driver_license order by count DESC ")
#.repartition(1)规定了将文件写入到一个文件里，否则会生成很多小文件
personDF.repartition(1).select("person","count").write.format("csv").save("file:///home/
cgy/Downloads/ForEachPersonCount.csv")

personDF.show()

 val sumDF = spark.sql("select sum(pay_price) as Montant  from records where
refund = 0 and pay_status =1 order by Montant DESC ")

 sumDF.repartition(1).write.format("csv").save("file:///home/cgy/Downloads/Sum.csv")
```

统计共为每一个顾客节省了多少钱，以及上个月
万盛达油券资金流水。

WinStar 陈光跃

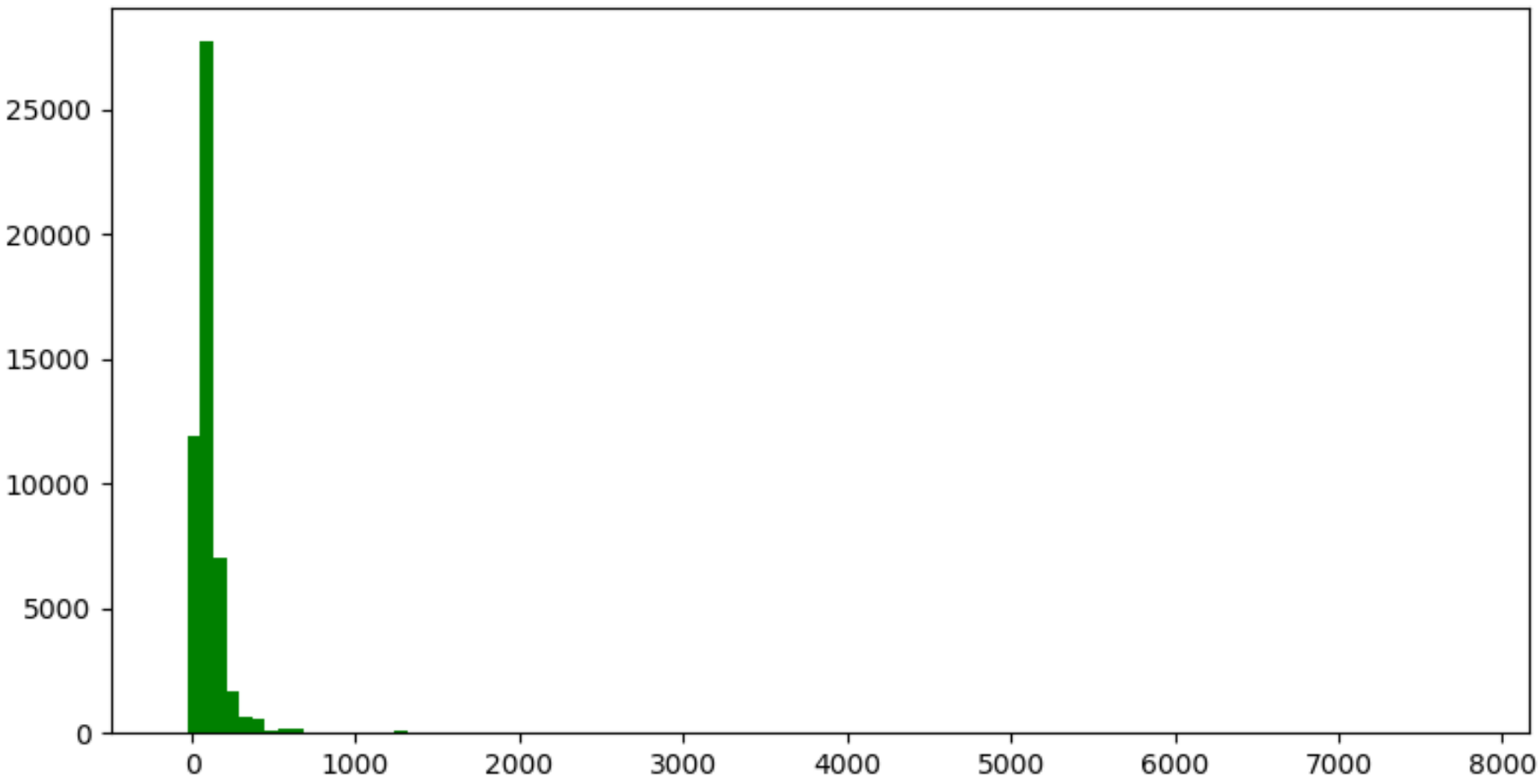
```
+------------------+--------+
|                  |755570.0|
|6105251986101213 15|  7768.0|
|6123211959031510 18|  5168.0|
|6123211984091800 44|  2568.0|
|6101231976101340 10|  2140.0|
|610404198503100 01X|  2036.0|
|6101131963052604 27|  1995.0|
|6104211972031600 55|  1988.0|
|6106321966122100 39|  1744.0|
|1529221974041500 18|  1716.0|
|3707821980021743 15|  1670.0|
|6103211993122721 13|  1666.0|
|6101211985062166 40|  1608.0|
|6103031985121942 39|  1580.0|
|6101111985041000 19|  1580.0|
|6124291980012260 76|  1580.0|
|6101151988041307 92|  1538.0|
|6105261990010300 13|  1488.0|
|6101111986052300 31|  1488.0|
|6127011989081400 19|  1480.0|
+------------------+--------+
only showing top 20 rows
```

```
scala> sumDF.show
+--------------------+
|              Montant|
+--------------------+
|4.4204290999995 73E7|
+--------------------+
```

上月油券交易流水：

4420万

通过对客户上个月所节省金额的统计，可以交与客户一个反馈，从而刺激消费



使用Python将数据制成图表形式

节省在几百元的顾客人数最多也比较正常

WinStar 陈光跃

```
val itemDF  = spark.sql("select item_id,count(*) as number from  records group by item_id order by number DESC")

itemDF.repartition(1).write.format("csv").save("file:///home/cgy/Downloads/ForEachItemCount.csv")

val activityDF  = spark.sql("select activity_id,count(*) as number from  records group by activity_id order by number DESC")

activityDF.repartition(1).write.format("csv").save("file:///home/cgy/Downloads/ForEachActivityCount.csv")
```

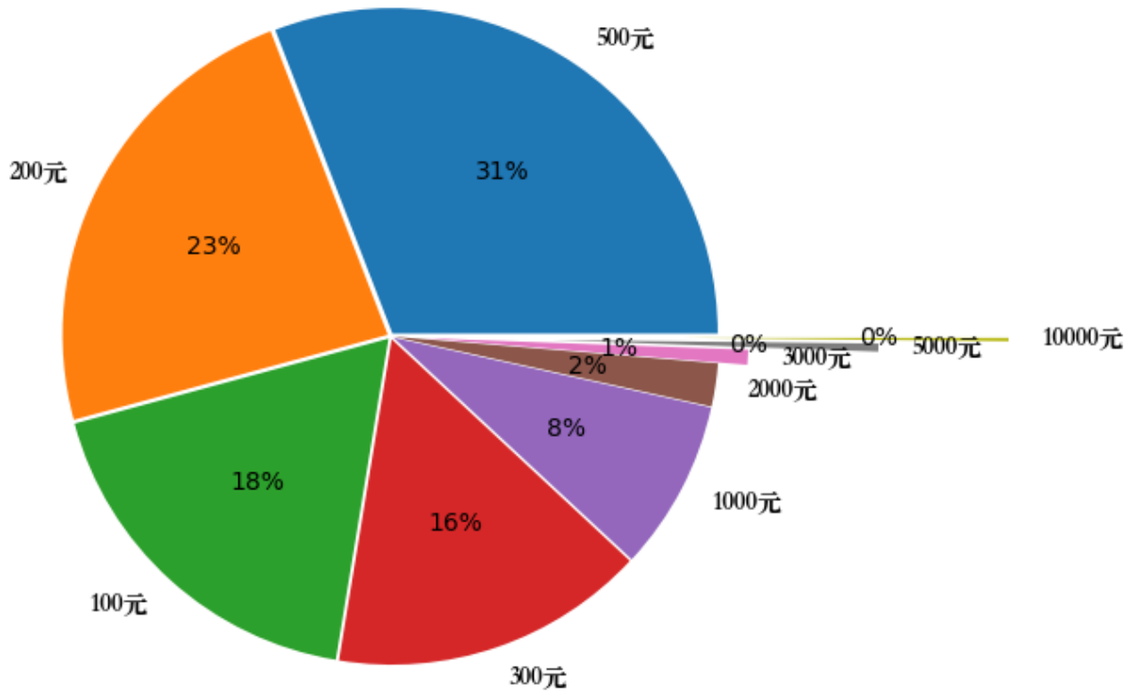
查询不同商品和不同活动的参与情况

item_id	number
50db459b612b082d0...	32423
5	12835
2	6385
4	6167
3	5755
11	5494
6	4874
50db459b612b082d0...	4756
50db459b612b082d0...	4404
50db459b612b082d0...	3801
50db459b612b082d0...	3792
2021	3322
50db459b612b082d0...	2000
1	1880
3023	1598
12	1323
2022	1146
50db459b612b082d0...	1000
50db459b612b082d0...	897
9	731

上个月万盛达不同商品销量

activity_id	number
1	52827
2	24970
202	17003
101	7328
201	4152
9	2680
3	834
105	312
104	120
103	67

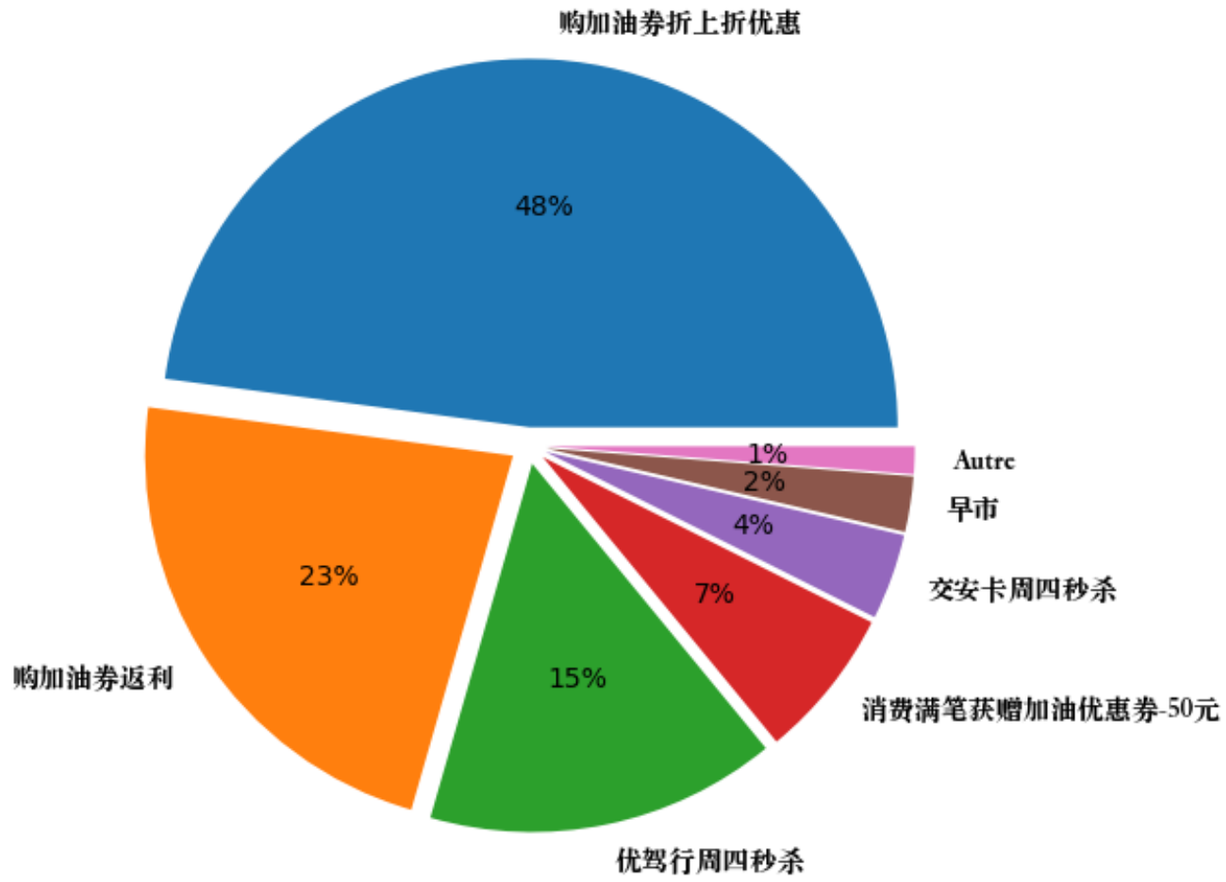
客户对不同活动的参与程度



油券是主要收入活动

Python饼图

可见小面额油券更加受欢迎，500销量最佳



WinStar 陈光跃

```
scala> spark.sql("select *  from records where pay_status = 0  ").show
+-----------+-----------------+-----------+---------------+-------+----------------+---------+----------+--------+------+-----------+------+--------------------+
|activity_id|coupon_templet_id|create_time|discount_amount|item_id|item_total_value|pay_price|pay_status|pay_type|refund|send_status|status|auth_driver_license|
+-----------+-----------------+-----------+---------------+-------+----------------+---------+----------+--------+------+-----------+------+--------------------+
+-----------+-----------------+-----------+---------------+-------+----------------+---------+----------+--------+------+-----------+------+--------------------+


scala> spark.sql("select *  from records where refund =1  ").show
+-----------+-----------------+-----------+---------------+-------+----------------+---------+----------+--------+------+-----------+------+--------------------+
|activity_id|coupon_templet_id|create_time|discount_amount|item_id|item_total_value|pay_price|pay_status|pay_type|refund|send_status|status|auth_driver_license|
+-----------+-----------------+-----------+---------------+-------+----------------+---------+----------+--------+------+-----------+------+--------------------+
+-----------+-----------------+-----------+---------------+-------+----------------+---------+----------+--------+------+-----------+------+--------------------+
```

由于数据源里未见支付失败数据，无法分析下订单未支付客户的数据。下订单未支付客户是最潜在目标用户，通过分析可能能够得到应对取消下单的措施吸引对方下单。

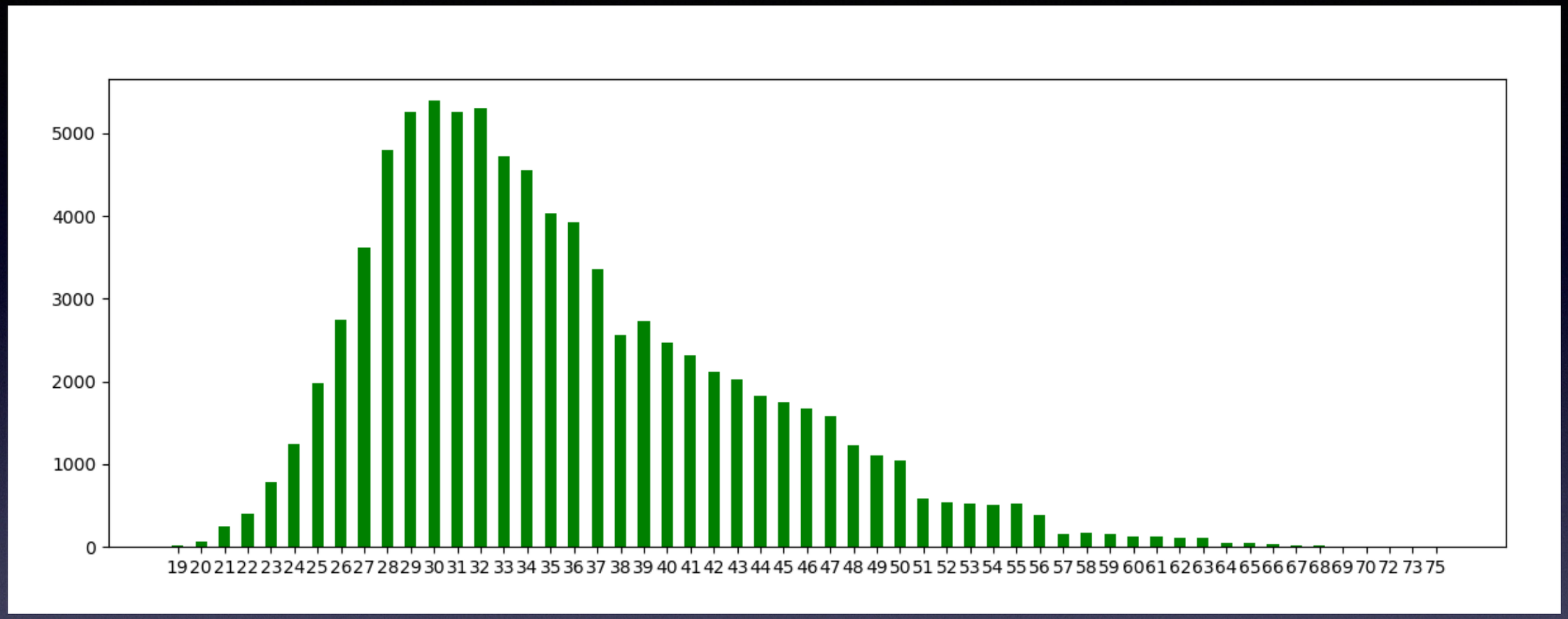
```
#.rdd 将数据转成rdd格式，以实现Spark的各种键值对操作
val people = spark.sql("select auth_driver_license  from records where auth_driver_license !
='' ").rdd
#读身份证号的出生年月算年龄
val age = people.map(x=>x.toString.substring(7,11).toInt).map(x=>2018-x)


val ageCounts = age.map(x => (x, 1)).reduceByKey{case (x, y) => x + y}.sortByKey()

ageCounts.toDF.repartition(1).write.format("csv").save("file:///home/cgy/Downloads/
ForEachAge.csv")
#读支付时间算时间
val time = spark.sql("select create_time  from records
").rdd.map(x=>x.toString.reverse.substring(7,9).reverse)

val timeCount = time.map(x => (x, 1)).reduceByKey{case (x, y) => x + y}.sortByKey()

timeCount.toDF.repartition(1).write.format("csv").save("file:///home/cgy/Downloads/
ForEachTime.csv")
```

```
scala> counts.toDF.foreach
[19,16]
[20,60]
[21,246]
[22,402]
[23,792]
[24,1240]
[25,1986]
[26,2747]
[27,3617]
[28,4798]
[29,5255]
[30,5395]
[31,5266]
[32,5302]
[33,4721]
[34,4554]
[35,4034]
[36,3928]
[37,3366]
[38,2560]
[39,2727]
[40,2468]
[41,2324]
[42,2123]
[43,2021]
[44,1823]
[45,1753]
[46,1676]
[47,1587]
[48,1225]
[49,1105]
[50,1041]
[51,591]
[52,543]
[53,523]
[54,514]
[55,521]
[56,389]
[57,157]
[58,174]
[59,160]
[60,132]
[61,120]
[62,106]
[63,104]
```

不同年龄段客户量

```
scala> timeCount.toDF.forea
[00,5451]
[01,804]
[02,390]
[03,264]
[04,277]
[05,662]
[06,2534]
[07,4614]
[08,6274]
[09,10861]
[10,7397]
[11,6840]
[12,5783]
[13,5019]
[14,5730]
[15,6261]
[16,6810]
[17,6450]
[18,6893]
[19,5699]
[20,5177]
[21,4358]
[22,3559]
[23,2186]
```

不同时间段，客户下单数量



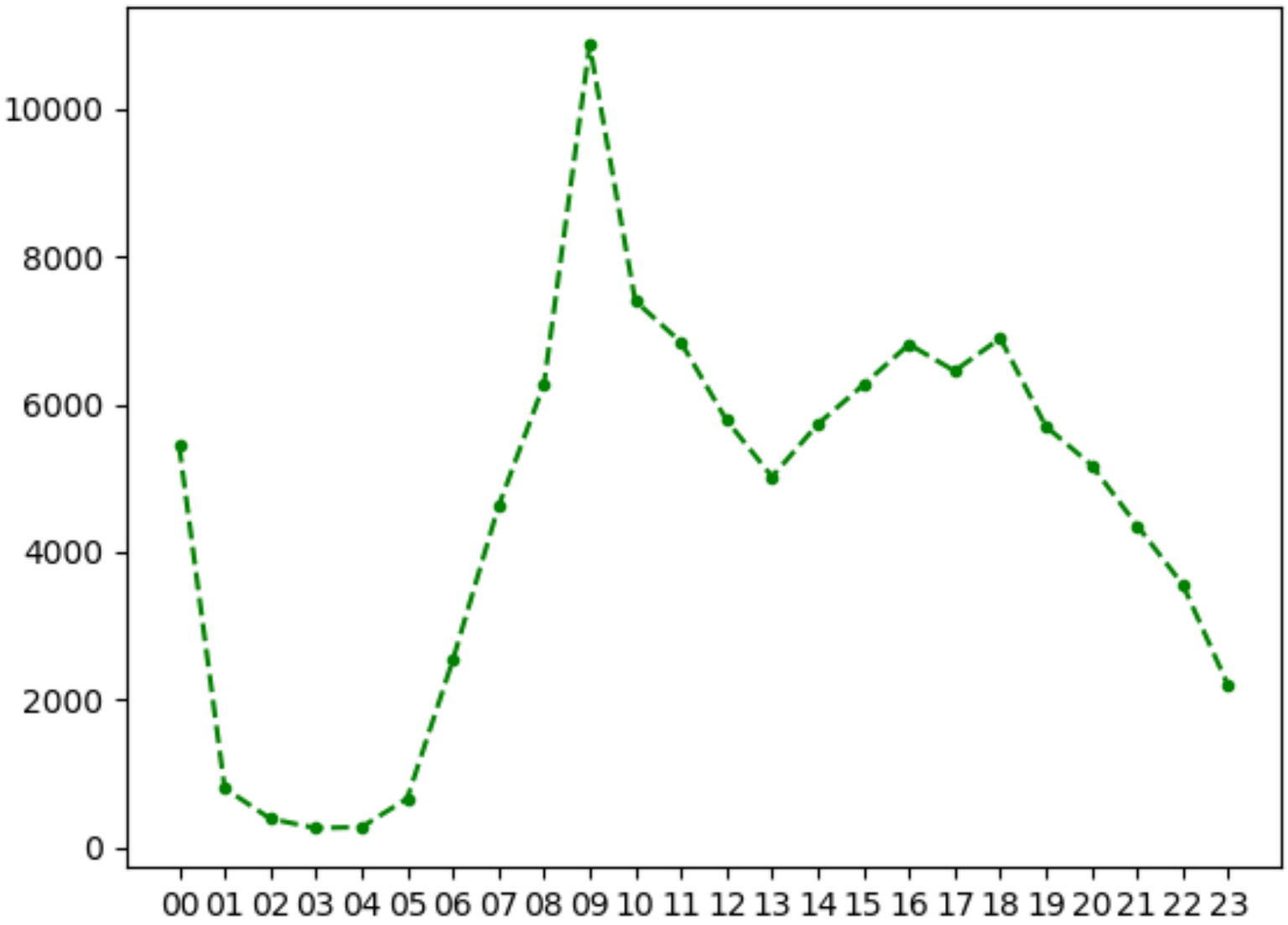
27岁到37岁是公司主流客户

（年轻的买不起车，老的驾照过期了）

可以通过年龄分布 来确定客户界面喜好云云

或是可以有目的根据喜好制定发展方向

WinStar 陈光跃



九点是加油高峰期，其次是下午和午夜12点到1点期间。所以可以有选择性的挑时间段发送微信或是APP消息推送

WinStar 陈光跃