野蛮时代氪金预测分析

一、项目背景

本案例是针对SLG手游《野蛮时代》(Brutal Age)用户行为数据的氪金预测。该案例数据来源于DC竞赛第二届智慧中国杯(ICC)。《在这个竞赛里，我们将利用玩家在游戏内前7日的行为数据，预测他们每个人在45日内的付费总金额。

用前7天玩家的行为数据预测他们在未来45天内一共会消费多少金额，这个预测一方面指向**转化**，另一方面指向**综合用户价值**。在45天内付出更多金额的用户是更高价值的用户，如果能够在运营的前期就预测出用户价值，则可以更精准地针对高价值用户进行营销

二、数据探索

在基本了解业务背景和建模目标之后，我们开始围绕数据集进行解读和探索。

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

#改变plt里的默认设置
#改变默认字体——黑体
plt.rcParams['font.sans-serif'] = ['Simhei'] #将jupyter中图画上的中文显示出来
#改成黑体后编码的模式不改变
plt.rcParams['axes.unicode_minus'] = False

data = pd.read_csv(r'D:\data\python\DataCastle2w\tap4fun\tap_fun_train.csv')
data.head()
```

	user_id	register_time	wood_add_value	wood_reduce_value	stone_add_value	stone_reduce_value	ivory_add_value	ivory_reduce_value	meat_add_value	me
0	1	2018-02-02 19:47:15	20125.0	3700.0	0.0	0.0	0.0	0.0	16375.0	
1	1593	2018-01-26 00:01:05	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
2	1594	2018-01-26 00:01:58	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
3	1595	2018-01-26 00:02:13	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
4	1596	2018-01-26 00:02:46	0.0	0.0	0.0	0.0	0.0	0.0	0.0	

5 rows × 109 columns

1、字段解释

将列名更新为中文

```
column_name = pd.read_excel(r'D:\data\python\DataCastle2w\tap4fun\tap4fun 数据字段解释.xlsx')
data.columns = column_name['字段解释']
data.head()
```

字段解释	玩家唯一ID	玩家注册时间	木头获取数量	木头消耗数量	石头获取数量	石头消耗数量	象牙获取数量	象牙消耗数量	肉获取数量	肉消耗数量	...	PVP次数	主动发起PVP次数	PVP胜利次数	PVE次数	主动发起PVE次数	PVE胜利次数	在线时长	付费金额	付费次数	45日付费金额
0	1	2018-02-02 19:47:15	20125.0	3700.0	0.0	0.0	0.0	0.0	16375.0	2000.0	...	0	0	0	0	0	0	0.333333	0.0	0	0.0
1	1593	2018-01-26 00:01:05	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0	0	0	0	0	0	0.333333	0.0	0	0.0
2	1594	2018-01-26 00:01:58	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0	0	0	0	0	0	1.166667	0.0	0	0.0
3	1595	2018-01-26 00:02:13	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0	0	0	0	0	0	3.166667	0.0	0	0.0
4	1596	2018-01-26 00:02:46	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0	0	0	0	0	0	2.333333	0.0	0	0.0

5 rows × 109 columns

```
#查看完整的特征列表
data.columns.tolist()
```

2、数据质量探索

- **数据集正确性校验**

首先是数据集正确性校验。一般来说数据集正确性校验分为两种，其一是**检验数据集字段是否和数据字典中的字段一致**，其二则是**检验数据集中ID列有无重复**。由于该数据集并未提供数据字典，因此此处主要校验数据集ID有无重复：

```
# 玩家ID是否有重复
data['玩家唯一ID'].duplicated().sum()
#0
```

- **数据缺失值检验**

```
#第一个sum()按列
data.isnull().sum().sum()
#0
```

- **重复值检验**

```
# 检查是否存在重复值
data.duplicated().sum()
#0
```

3、数据类型探索

```
#查看其他信息
#数据量太大，只显示部分信息
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2288007 entries, 0 to 2288006
Columns: 109 entries, 玩家唯一ID to 45日付费金额
dtypes: float64(13), int64(95), object(1)
memory usage: 1.9+ GB
```

有1列数据是对象类型，也就是注册时间。有95列数据类型都为整数。通常来说，当一个结构化数据中的特征数据类型是整数时，倾向于认为该特征可能是离散型特征。但从游戏业务的独特性、以及特征含义来看，资源数量/等级/发起PVP次数等信息一般不会以浮点数表示，因此显示为整数也十分正常。

从特征含义来看，该数据集中没有任何离散型特征(数据类型超过25个，当连续型数据处理)

4、查看分布和统计信息

```
data.describe().T
```

字段解释	count	mean	std	min	25%	50%	75%	max
玩家唯一ID	2288007.0	1.529543e+06	9.399393e+05	1.0	749992.5	1.419095e+06	2.299006e+06	3.190530e+06
木头获取数量	2288007.0	4.543069e+05	4.958667e+06	0.0	0.0	4.203800e+04	1.531180e+05	1.239962e+09
木头消耗数量	2288007.0	3.698433e+05	3.737720e+06	0.0	0.0	9.830000e+03	9.855700e+04	7.995875e+08
石头获取数量	2288007.0	1.897788e+05	4.670620e+06	0.0	0.0	0.000000e+00	0.000000e+00	1.214869e+09
石头消耗数量	2288007.0	1.376074e+05	3.370166e+06	0.0	0.0	0.000000e+00	0.000000e+00	7.962378e+08
...	...	...	...	...	...	...	...	...
PVE胜利次数	2288007.0	2.556749e+00	1.184737e+01	0.0	0.0	0.000000e+00	1.000000e+00	4.880000e+02
在线时长	2288007.0	1.020749e+01	3.895946e+01	0.0	0.5	1.833333e+00	4.833333e+00	2.049667e+03
付费金额	2288007.0	5.346691e-01	2.263835e+01	0.0	0.0	0.000000e+00	0.000000e+00	7.457950e+03
付费次数	2288007.0	5.770699e-02	7.090886e-01	0.0	0.0	0.000000e+00	0.000000e+00	1.050000e+02
45日付费金额	2288007.0	1.793146e+00	8.846303e+01	0.0	0.0	0.000000e+00	0.000000e+00	3.297781e+04

108 rows × 8 columns

游戏资源数据往往都是左偏的，因为大于50%的新增用户会在第二天就流失掉，因此大部分用户所掌握的资源量都很少。

然而该数据集左偏的程度过高。对于"石头"这样明显是基础资源的特征来说，位于上四分位数的值依然是0，至少有75%的玩家没有坚持到开采出木头以外的资源，玩家流失情况可能很严重。

高度偏态的信息不利于异常值检测，可以预见大量的长尾数据会被归为异常值，因此我们需要谨慎处理异常值

5、标签探索

```
y = data.iloc[:,-1]
SevenDayPayAmount = data.iloc[:,-3]#探索7日内经营状况
```

- **分布探索**

```
y.describe([0.75,0.90,0.95,0.97,0.99])
```

```
count    2.288007e+06
mean     1.793146e+00
std      8.846303e+01
min      0.000000e+00
50%      0.000000e+00
75%      0.000000e+00
90%      0.000000e+00
95%      0.000000e+00
97%      0.000000e+00
99%      3.970000e+00
max      3.297781e+04
Name: 45日付费金额, dtype: float64
```

从分布来看超过97%的用户都没有付费，游戏转化可能存在很大问题

- **前7日和45日氪金情况分析**

```
#定义函数：展示7日与45日氪金情况的关键指标
def RevenueFocus(RevenueSeries):

    #中间变量
    AllUsers = len(RevenueSeries)#全部用户
    PaidUsers = (RevenueSeries!=0).sum()#付费用户数
    TotalPayment = RevenueSeries.sum() #总付费金额

    #打印结果
    print('付费率:{:.3f}%'.format(100*PaidUsers/AllUsers))#显示3位小数
    print('付费人数:{}'.format(PaidUsers))
    print('转化总金额:{:.3f}'.format(TotalPayment))
    print('ARPU:{:.3f}'.format(TotalPayment/AllUsers))#ARPU：Average Revenue Per User用户平均收入
    print('ARPPU:{:.3f}'.format(TotalPayment/PaidUsers))#ARPPU：Average Revenue Per Paying User付费用户平均收入
    print('前500高氪用户金额占比:{:.3f}%'.format(100*RevenueSeries.sort_values(ascending=False)[:500].sum()/TotalPayment))
    print('前1000高氪用户金额占比:{:.3f}%'.format(100*RevenueSeries.sort_values(ascending=False)[:1000].sum()/TotalPayment))
    print('前5000高氪用户金额占比:{:.3f}%'.format(100*RevenueSeries.sort_values(ascending=False)[:5000].sum()/TotalPayment))
```

45日氪金情况

```
RevenueFocus(y)
```

```
付费率:2.010%
付费人数:45988
转化总金额:4102730.110
ARPU:1.793
ARPPU:89.213
前500高氪用户金额占比:51.618%
前1000高氪用户金额占比:64.878%
前5000高氪用户金额占比:89.375%
```

前7日氪金情况

```
RevenueFocus(SevenDayPayAmount)
```

```
付费率:1.811%
付费人数:41439
转化总金额:1223326.660
ARPU:0.535
ARPPU:29.521
前500高氪用户金额占比:45.211%
前1000高氪用户金额占比:57.231%
前5000高氪用户金额占比:83.002%
```

整体来看付费率是偏低的，2016年时全球F2P(free to play)手游的付费率就有2.3%，《野蛮时代》的制作公司Tap4fun具有丰富的SLG手游经验，并且在开服测试阶段就有200w用户数据，转化率应该更高。

2020年SLG手游的平均付费率大约在5%左右，顶级手游的付费率能够接近15%。

(另外，在流量昂贵的现在，一个月400w的收入，或许还不能覆盖掉200w用户的流量成本，收入状况不算特别理想)

```
#7-45日新增氪金用户
45988 - 41439
#4549

#前7天没有氪金的用户
(SevenDayPayAmount == 0).sum()
#2246568

#7-45日新增氪金玩家/前7天没有氪金的玩家(%)
100*(45988 - 41439)/((SevenDayPayAmount == 0).sum())
#0.2024866373953515
```

从数据逻辑来看，45日总付费不为0的用户一定包括前7日内付过费的用户。

从7日到45日，新增付费用户仅有4549人(占不氪金玩家的0.2%)，即是说如果一个新玩家在服的前七天没有氪金，那他/她在未来一个月内99.8%的可能性都不会氪金。

这可能时由于如下原因导致的:

1.大部分用户都留存到了45日之后，这说明新用户福利过去后，游戏中的转化手段非常贫瘠，转化刺激严重不足

2.大部分用户在45日之前已经流失，因此才没有继续转化为氪金用户的机会

同时，我们来计算7日之后再也没有付费的用户:

```
#7日之后再也没有氪金的玩家：
#即45日时付费金额与7日付费金额相同，且7日时付过费的玩家
SevenDayPayAmount[SevenDayPayAmount == y][SevenDayPayAmount!=0].count()
30130
```

```
# 占前7日氪金玩家人数的占比
100*30130/((SevenDayPayAmount!=0).sum())
#72.70928352518159
```

不难发现，7日之前付过费，并且7日之后再也没有付费的用户有3万多人，占到前7天总付费用户的72%

即一个新玩家在服前七天有氪金，但她/他在未来一个月还是有7成的概率不再继续氪金

这可能说明(按可能性排序):

1.对大部分用户而言，最初氪的是超低价/超实惠新手礼匀，后续氪金礼包中的福利/以及价格超出了大部分用户对游戏氪金的心理预期

2 .新手礼包力度过大，导致后续氪金礼包中的福利看起来不够实惠，用户再付费门槛较高

3.游戏本身粘性不足，无论用户是否氪金，大部分人都在7-45日内自然流失了(甚至在前7日就流失了大部分)

4.开局需要发育时玩家更愿意氪金解决问题（痒感充足)，发育起来之后便不再具有强烈氪金需求/发育不起来直接退服了

5.(可能性较小，仅针对少部分前7日巨但7日后却不再氪的大佬）氪金后体验一般甚至较差，例如，发现氪几百、上千元后对玩家竞争力无明显提升

```
#7日后再也没有付过费的用户前七日的氪金情况
SevenDayPayAmount[SevenDayPayAmount == y][SevenDayPayAmount!=0].describe()
```

```
count    30130.000000
mean        11.407407
std         71.134875
min          0.990000
25%          0.990000
50%          1.980000
75%          5.980000
max       4086.520000
Name: 付费金额, dtype: float64
```

```
SevenDayPayAmount[SevenDayPayAmount == y][SevenDayPayAmount!=0].sum()
#343705.17000000004
```

```
#7日后继续付费的用户前7日的氪金情况
SevenDayPayAmount[SevenDayPayAmount != y][SevenDayPayAmount!=0].describe()
```

```
count    11309.000000
mean        77.780661
std        289.610458
min          0.990000
25%          2.980000
50%         15.970000
75%         45.940000
max       7457.950000
Name: 付费金额, dtype: float64
```

```
SevenDayPayAmount[SevenDayPayAmount != y][SevenDayPayAmount!=0].sum()
#879621.4900000001
```

非常明显，7日后继续氪金的玩家在前7日会氪更多的金额，7日后不再氪金的玩家中有75%都只氪了6元以下的金额，
而7日后继续氪金的玩家中有50%以上都氪了16元左右，这极大地印证了我们提出的第一条理由:

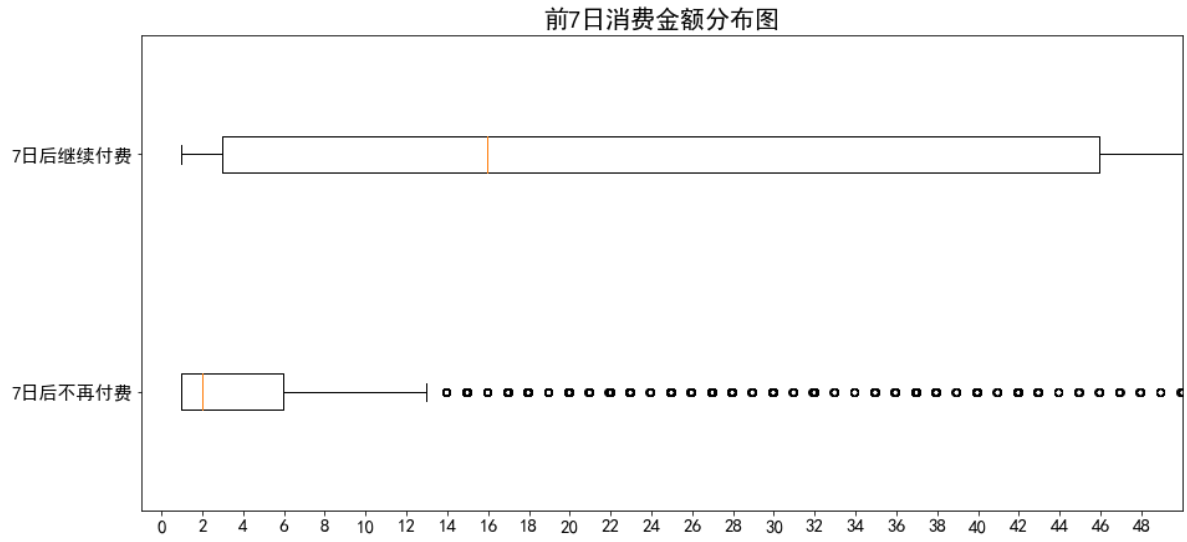
大部分人氪新手礼包，而后续礼包的价格在大部分人的心理预期之外。

同时，7日后不再氪金的玩家中也有不少氪过百过千的玩家，最终放弃游戏可能是因为自然流失/氪金后体验不足以满足部分用户的想象。

但从7日后继续付费的用户氪金情况来看，用户是倾向于越氪越多的，因此大部分用户对于氪金后的感受应该是满意的。

(后续我们会看到，结合ARPU和ARPPU的数据，可以推测氪金后的游戏体验是很不错的)。

```
# 使用箱线图对比：两类用户在前7日的氪金情况
plt.figure(figsize=(15,7))
plt.boxplot([SevenDayPayAmount[SevenDayPayAmount == y][SevenDayPayAmount!=0]
             ,SevenDayPayAmount[SevenDayPayAmount != y]
[SevenDayPayAmount!=0]]
             ,labels=['7日后不再付费','7日后继续付费']
             ,vert=False#是否垂直摆放
            )
plt.xlim(-1,50)
plt.title('前7日消费金额分布图',fontsize=20)
plt.yticks(fontsize=15.0)
plt.xticks(range(0,50,2),fontsize=15.0)
plt.show()
```



不难发现，氪金的范围虽然一样，但继续付费用户的分布要靠右许多，说明7日后付费用户在前7日消费更多。

高价值用户评判的第一个可能的标准:前7日氰金超过6元的用户是更有价值的用户

以此推测，新手礼包设置为5-7元，或许可以明确辨析出用户的付费能力。

对于7日内已氪用户而言，氪任意金额的用户都有一定的概率继续氪金，也有一定的概率不再氪金，且两个概率相加为1。举个简单的例子:

假设前7日付费金额超过n元的有60人，在这60人中，后续继续氪金的是20人，不再氪金的是40人，那我们就可以说，当用户前7日消费超过n元时，继续氪金的概率是20/60 = 33%，不再氪金的概率是40/60 = 66%。

根据历史数据，当用户7日内氪的金额越大，用户对氪金的心理预期越高，继续氪金的可能性就越大不再氪金的可能性就越小。

以上面例子中的数字，我们应该认为，前7日消费n+1元的用户继续氪金的概率大于33%，不在氪金的概率小于66%。
随着前7日内氪金的金额变大，继续氪金的概率会越来越大，不再氪金的概率会越来越小，因此，必然存在一个金额点，当用户在前7日内付费的金额超过该金额点时，该用户继续氪金的可能性大于不再氪金的可能性。该金额点就是适合作为新手礼包设置的点，现在我们来找出这个点。

考虑到后续付费用户有50%都付了16元及以上，我们推测这个金额点应该是在16元以下的，因此我们设置潜在金额的范围为1-16。

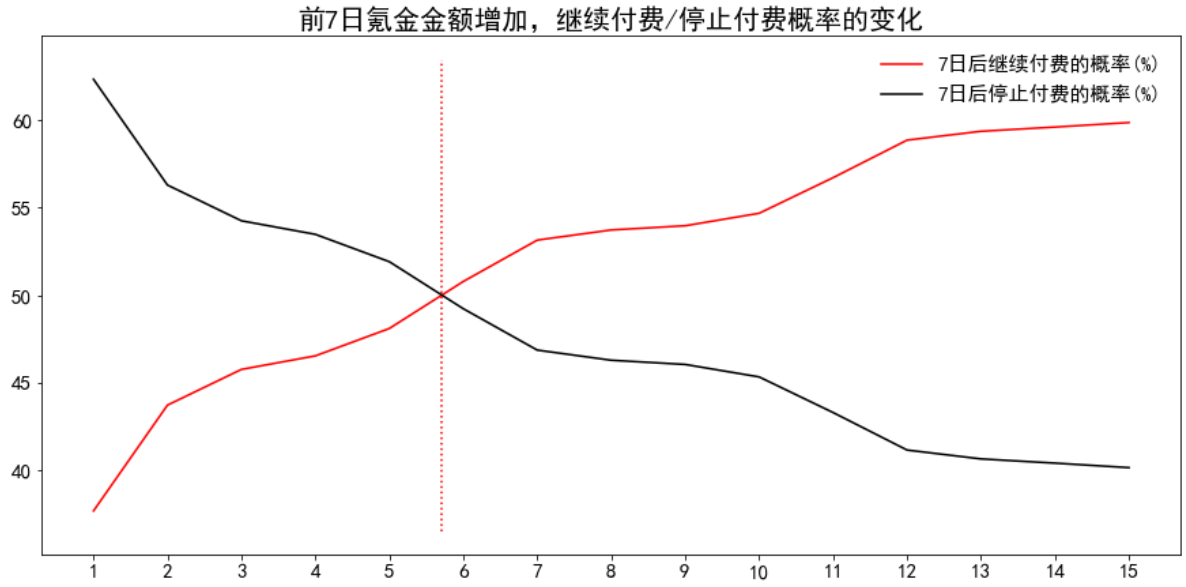
```
PotentialPoints = range(1,16)
for i in PotentialPoints:
```

```
    #取出所有7天内付费金额超过i的付费用户
    HighThanPoint = SevenDayPayAmount[SevenDayPayAmount>=i]
    #这些用户7天后继续付费的比例
    KeepPaid = len(HighThanPoint[SevenDayPayAmount!=y])/len(HighThanPoint)
    #这些用户停止付费的比例
    StopPaid = len(HighThanPoint[SevenDayPayAmount==y])/len(HighThanPoint)
    print(i)
    print('{:.3f}%'.format(100*KeepPaid))
    print('{:.3f}%'.format(100*StopPaid))
    print('{:.3f}%'.format(KeepPaid-StopPaid))
    if KeepPaid - StopPaid > 0:
        print('当7日内付费金额大于{:.2f}时，用户继续氪金的可能性比不再氪金的可能性更高！'.format(i))
        break
```

```
1
37.683%
62.317%
-0.246%
2
43.720%
56.280%
-0.126%
3
45.759%
54.241%
-0.085%
4
46.532%
53.468%
-0.069%
5
48.094%
51.906%
-0.038%
6
50.778%
49.222%
0.016%
当7日内付费金额大于6.00时，用户继续氪金的可能性比不再氪金的可能性更高！
```

```
# 用图像来呈现则更明显
KeepPaidList = []
StopPaidList = []
for i in PotentialPoints:
    #取出所有7天内付费金额超过i的付费用户
    HighThanPoint = SevenDayPayAmount[SevenDayPayAmount>=i]
    #这些用户7天后继续付费的比例
    KeepPaid = len(HighThanPoint[SevenDayPayAmount!=y])/len(HighThanPoint)
    #这些用户停止付费的比例
    StopPaid = len(HighThanPoint[SevenDayPayAmount==y])/len(HighThanPoint)
    KeepPaidList.append(100*KeepPaid)
    StopPaidList.append(100*StopPaid)

plt.figure(figsize=(15,7))
plt.plot(PotentialPoints,KeepPaidList,label='7日后继续付费的概率(%)',color='red')
plt.plot(PotentialPoints,StopPaidList,label='7日后停止付费的概率(%)',color='black')
# plt.xlim(-1,50)
plt.title('前7日氪金金额增加，继续付费/停止付费概率的变化',fontsize=20)
plt.yticks(fontsize=15)
plt.xticks(range(0,16,1),fontsize=15)
plt.legend(fontsize=15,frameon=False)
```

```
plt.vlines(5.7,ymin=plt.ylim()[0]+0.1,ymax=plt.ylim()
[1]-0.1,colors='red',linestyles='dotted')
plt.show()
```



在前7日内付费6元以下的用户，无论是否自然流失，未来氪金的可能性都是小于继续付费的可能性的。

可以通过设置游戏体验，给与付费5元左右的用户更多的"刺激"，让他们后续继续付费。也可选择直接放弃6元以下的付费用户，体验维持现状或减少资源分配，让更多的资源/服务围绕更有可能付费的用户展开。

回到我们的收入分析:

```
# 45日转化情况
RevenueFocus(y)
```

```
付费率:2.010%
付费人数:45988
转化总金额:4102730.110
ARPU:1.793
ARPPU:89.213
前500高氪用户金额占比:51.618%
前1000高氪用户金额占比:64.878%
前5000高氪用户金额占比:89.375%
```

```
y.max()
#32977.81
```

```
#7日转化情况
RevenueFocus(SevenDayPayAmount)
```

```
付费率:1.811%
付费人数:41439
转化总金额:1223326.660
ARPU:0.535
ARPPU:29.521
前500高氪用户金额占比:45.211%
前1000高氪用户金额占比:57.231%
前5000高氪用户金额占比:83.002%
```

来看ARPU与ARPPU数

ARPU是全服用户平均收入，ARPPU是全服付费用户平均收入

目前来看全服的ARPU较低，但ARPPU还不错，这说明虽然付费人数不多，但是大R(大人民币玩家）的付费能力相对较强。

从前5000高氪用户金额占比来看，45日时全服氪金前5000的人(z占比小于0.22%)提供了全服89%的收入，大R们贡献巨大。

这令我猜测，《野蛮时代》中的氪金行为应该与玩家竞争力有较为直接的联系

因为许多大R玩家非常偏爱强PK类游戏，付费可以直接让PVP变得异常简单

不过全服最氪玩家在一个月内的消费只有3w2，在SLG手游领域并非一个很高的数字

如果游戏体验优秀含应该还有很大的上升空间。

而付费人数不多的问题还可以改善，游戏整体的经营成果还可以上一个台阶

6、特征的探索

（1）在线时长

通过判断用户在线时长，我们可以大约估计出用户流失的速度。

```
PlayTime = data.loc[:,'在线时长']
PlayTime.max()
#2049.666667
```

原数据并未说明这个衡量时间的特征的单位，但7日为168小时，因此该特征的单位应该是分钟。前7日内游戏时长最长的玩家玩了2049分钟，即34个小时，每日玩5个小时以上，也是铁杆玩家可以达得到的数据，单位为分钟是合理的。

```
# 对pandas进行属性设置--避免显示问题，设置不显示科学计数法
pd.set_option('display.float_format',lambda x : '{:.3f}'.format(x))
PlayTime.describe()
```

```
count   2288007.000
mean         10.207
std          38.959
min           0.000
25%           0.500
50%           1.833
75%           4.833
max        2049.667
Name: 在线时长, dtype: float64
```

75%的人在前7日一共只玩了4.8min或以下，这说明75%的用户只是稍微看了一眼游戏就被劝退了

```
PlayTime.describe([0.75,0.9,0.95,0.99])
```

```
count    2288007.000
mean          10.207
std           38.959
min            0.000
50%            1.833
75%            4.833
90%           15.000
95%           41.333
99%          183.657
max         2049.667
Name: 在线时长, dtype: float64
```

SLG游戏天生会充满许多任务、奖励和剧情，除非用户登录即退出，否则登入一次至少应该停留5分钟以上。
新用户停留的时间应当更长（有非常多的内容需要探索、任务列表也很长）。

如果一个用户在注册游戏后第二天依然被留存下来、并参与到游戏当中，那两天在线时长至少也应该有15-20分钟。

75%的用户在前7日内只玩了4.8分钟，而如下所示，90%的用户在7日内只玩了15分钟以下。

据推断，大约只有10~15%的用户在第二天依然留存下来，同时每天大约玩10分钟，7日后依然留存下来的用户大约少于5%。

用户的确以超出常规的方式流失，在这种流失方式下，付费率也很难被拯救。

- **关于在线时长最长的玩家**

```
(data['在线时长']>2000).sum()
#1
```

```
data.loc[data['在线时长']>2000,:]
```

玩家唯一ID	玩家注册时间	木头获取数量	木头消耗数量	石头获取数量	石头消耗数量	象牙获取数量	象牙消耗数量	肉获取数量	肉消耗数量	...	PVP次数	主动发起PVP次数	PVP胜利次数	PVE次数	主动发起PVE次数	PVE胜利次数	在线时长	付费金额	付费次数	45日付费金额
645487	2018-03-04 13:43:36	697852.000	300255.000	0.000	0.000	6000.000	0.000	690379.000	196602.000	...	5	0	0	2	2	2	2049.667	0.000	0	0.000

109 columns

在线时间最长，但是获取的游戏资源并不多，游戏内容没有深入。猜测可能是测试人员或者GM（管理员账号）

- **氪金用户的在线时长分析**

一般来说，投入巨量时间的玩家不会氪太多，时间可以获取需要的资源以及克服游戏中设置的氪金点

```
PaidPlayTime = data.loc[data['付费金额']!=0,'在线时长']
PaidPlayTime.describe().T
```

```
count    41439.000
mean       140.188
std        149.973
min          0.000
25%         33.000
50%         88.833
75%        194.667
max       1674.667
Name: 在线时长, dtype: float64
```

只有25%以下的用户在线时间在半小时以下，游戏时间明显比非氪金用户多了很多。

```
data.loc[data['在线时长']<=1,'付费金额'].value_counts()
```

```
0.000     896061
0.990         51
9.990          4
36.960         3
5.980          3
4.990          3
99.990         2
56.950         2
19.990         1
26.970         1
25.970         1
6.980          1
49.990         1
1.990          1
Name: 付费金额, dtype: int64
```

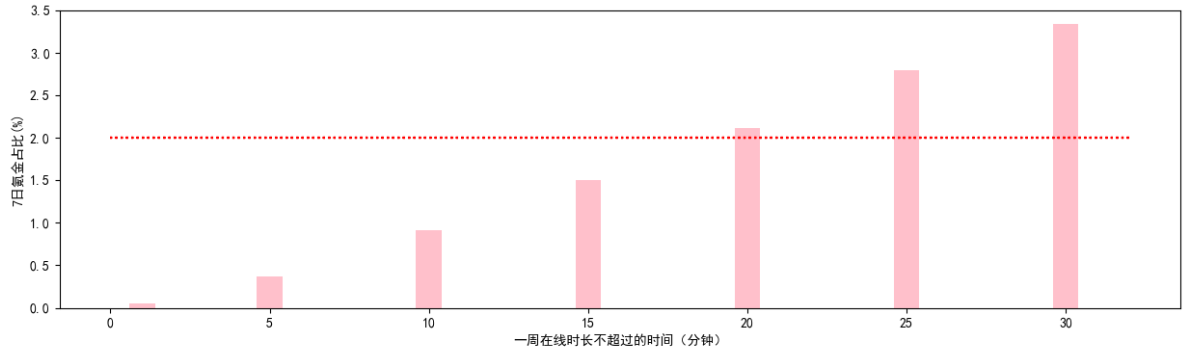
在线时长甚至不足一分钟的玩家们，有40人氪了0.99的新手礼包。

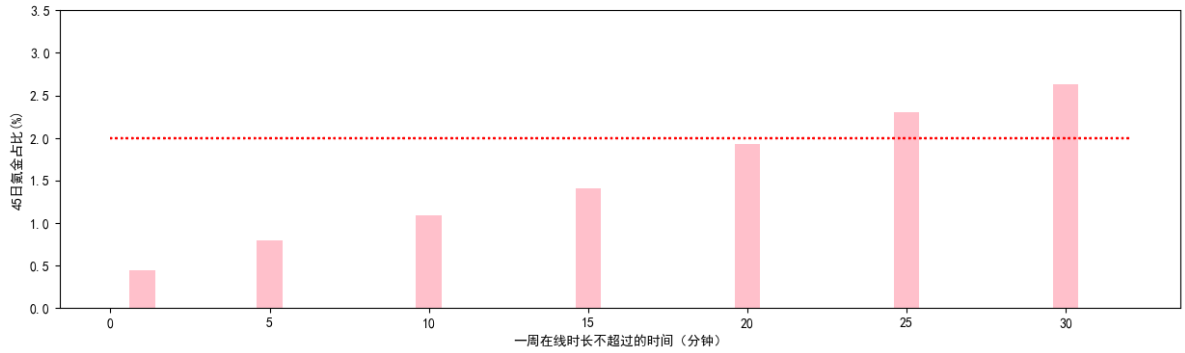
在线时间很长的用户不一定高氪，在线时间很短的用户也不一定完全不氪，下面就在线时间这个点来进行用户价值判断

- **在线时长很少的玩家们贡献的氪金数额**

```
plt.figure(figsize=(15,9),dpi=100)
for playtime in [1,5,10,15,20,25,30]:

    MeanPay45 = data.loc[data['在线时长']<playtime,'45日付费金额'].mean()
    TotalPay45 = data.loc[data['在线时长']<playtime,'45日付费金额'].sum()
    MeanPay7 = data.loc[data['在线时长']<playtime,'付费金额'].mean()
    TotalPay7 = data.loc[data['在线时长']<playtime,'付费金额'].sum()

    print('一周在线时长不足{}分钟'.format(playtime))
    print('\t45日平均消费额为{:.3f},45日总消费额为{:.3f}
元'.format(MeanPay45,TotalPay45))
    print('\t7日平均消费额为{:.3f},7日总消费额为{:.3f}
元'.format(MeanPay7,TotalPay7))
    print('\t7日氪金占比{:.3f}%,45日氪金占比
{:.3f}%'.format(100*TotalPay7/(data['付费金额'].sum())

,100*TotalPay45/(data['45日付费金额'].sum())))



    plt.subplot(211)
    plt.bar(playtime,100*TotalPay7/(data['付费金额'].sum()),color='pink')
    plt.xlabel('在线时长(分钟)')
    plt.ylabel('7日氪金占比(%)')
    plt.hlines(2,0,32,colors='red',linestyles='dotted')
    plt.ylim(0,3.5)
    plt.subplot(212)
    plt.bar(playtime,100*TotalPay45/(data['45日付费金
额'].sum()),color='pink')
    plt.xlabel('在线时长(分钟)')
    plt.ylabel('45日氪金占比(%)')
    plt.hlines(2,0,32,colors='red',linestyles='dotted')
    plt.ylim(0,3.5)
plt.show()
```

```
一周在线时长不足1分钟
        45日平均消费额为0.022,45日总消费额为18323.490元
        7日平均消费额为0.001,7日总消费额为574.180元
        7日氪金占比0.047%,45日氪金占比0.447%
一周在线时长不足5分钟
        45日平均消费额为0.019,45日总消费额为32572.700元
        7日平均消费额为0.003,7日总消费额为4519.280元
        7日氪金占比0.369%,45日氪金占比0.794%

一周在线时长不足10分钟
        45日平均消费额为0.023,45日总消费额为44527.180元
        7日平均消费额为0.006,7日总消费额为11151.310元
        7日氪金占比0.912%,45日氪金占比1.085%
一周在线时长不足15分钟
        45日平均消费额为0.028,45日总消费额为57753.190元
        7日平均消费额为0.009,7日总消费额为18428.260元
        7日氪金占比1.506%,45日氪金占比1.408%

一周在线时长不足20分钟
        45日平均消费额为0.038,45日总消费额为79124.920元
        7日平均消费额为0.012,7日总消费额为25904.310元
        7日氪金占比2.118%,45日氪金占比1.929%
一周在线时长不足25分钟
        45日平均消费额为0.044,45日总消费额为94650.110元
        7日平均消费额为0.016,7日总消费额为34217.390元
        7日氪金占比2.797%,45日氪金占比2.307%

一周在线时长不足30分钟
        45日平均消费额为0.050,45日总消费额为107763.140元
        7日平均消费额为0.019,7日总消费额为40862.200元
        7日氪金占比3.340%,45日氪金占比2.627%
```





不难发现，在线时长很少的玩家们贡献的氪金数额都很少。在对*200*多万人进行建模时，我们可以忽略一部分在线时间很短的用户。
在这里，我们可以将一周在线时长不足20分钟（氪金占比约2%）的用户判断为低价值用户。

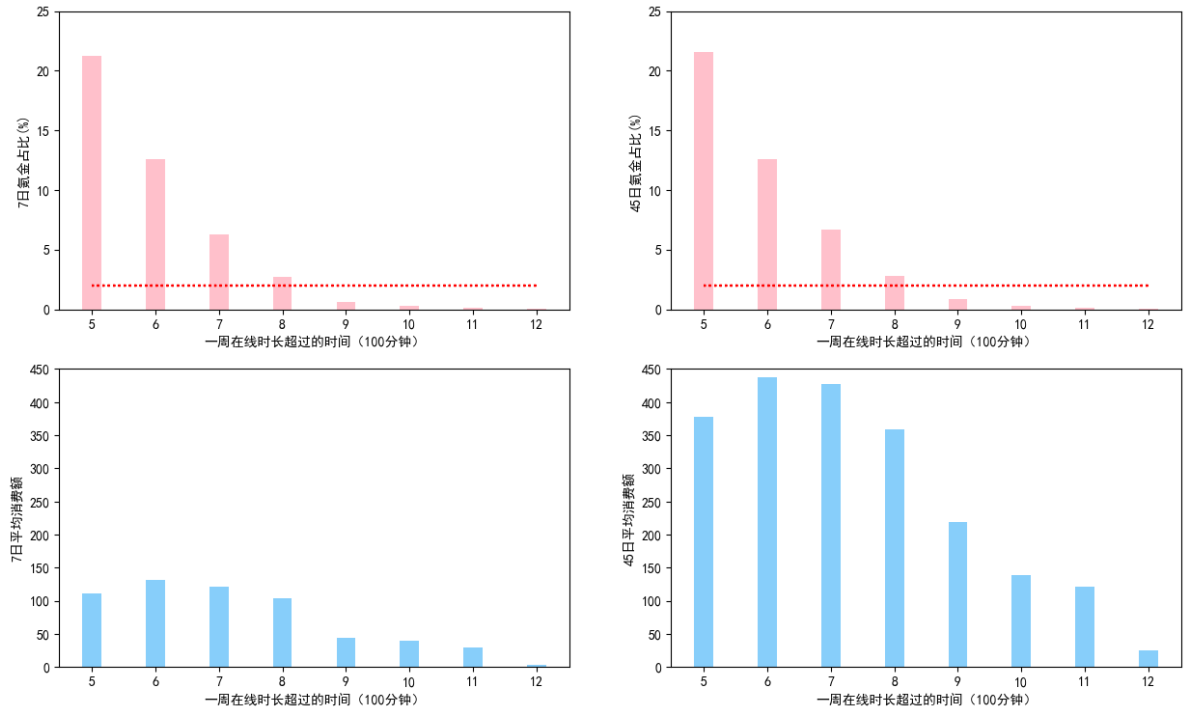
- **在线时长超长的玩家们贡献的氪金数额**

```
# 最肝的前0.1%的用户7日在线时长
data['在线时长'].sort_values(ascending=False)[:2288]
```

```
488497     2049.667
1263615    1674.667
882241     1482.000
1558221    1452.333
1335561    1452.333
             ...
527722      504.500
2065723     504.333
1815137     504.333
1257763     504.333
2145496     504.333
Name: 在线时长, Length: 2288, dtype: float64
```

取500作为超肝的玩家在线时长的下界

```
# 在线时长超长的玩家们贡献的氪金数额
plt.figure(figsize=(15,9),dpi=100)
for playtime in [500,600,700,800,900,1000,1100,1200]:
    MeanPay45 = data.loc[data['在线时长']>=playtime,'45日付费金额'].mean()
    TotalPay45 = data.loc[data['在线时长']>=playtime,'45日付费金额'].sum()
    MeanPay7 = data.loc[data['在线时长']>=playtime,'付费金额'].mean()
    TotalPay7 = data.loc[data['在线时长']>=playtime,'付费金额'].sum()

    print('一周在线时长超过{}分钟'.format(playtime))
    print('\t45日平均消费额为{:.3f},45日总消费额为{:.3f}
元'.format(MeanPay45,TotalPay45))
    print('\t7日平均消费额为{:.3f},7日总消费额为{:.3f}
元'.format(MeanPay7,TotalPay7))
    print('\t7日氪金占比{:.3f}%,45日氪金占比
{:.3f}%'.format(100*TotalPay7/(data['付费金额'].sum())

,100*TotalPay45/(data['45日付费金额'].sum())))
    plt.subplot(221)
    plt.bar(playtime/100,100*TotalPay7/(data['付费金
额'].sum()),color='pink',width=0.3)
    plt.xlabel('一周在线时长超过的时间（100分钟）')
    plt.ylabel('7日氪金占比(%)')
    plt.hlines(2,5,12,colors='red',linestyles='dotted')
    plt.ylim(0,25)
    plt.subplot(222)
    plt.bar(playtime/100,100*TotalPay45/(data['45日付费金
额'].sum()),color='pink',width=0.3)
    plt.xlabel('一周在线时长超过的时间（100分钟）')
    plt.ylabel('45日氪金占比(%)')
    plt.hlines(2,5,12,colors='red',linestyles='dotted')
    plt.ylim(0,25)

    plt.subplot(223)
    plt.bar(playtime/100,MeanPay7,color='lightskyblue',width=0.3)
    plt.xlabel('一周在线时长超过的时间（100分钟）')
    plt.ylabel('7日平均消费额')
    plt.ylim(0,450)
    plt.subplot(224)
    plt.bar(playtime/100,MeanPay45,color='lightskyblue',width=0.3)
    plt.xlabel('一周在线时长超过的时间（100分钟）')
    plt.ylabel('45日平均消费额')
    plt.ylim(0,450)

plt.show()
```



很多用户虽然深度参与游戏，但是氪金占比较低，所以过于肝的用户也不是高价值用户。我们可以规定，一周在线时长超出800分钟的用户为(氪金层面)低价值用户。同样的，氪金用户一般在资源上占绝对的主导，但是资源很多的用户不一定是氪金用户。

（2）初始资源类特征—木头获取数量

```
data.loc[data['付费金额']!=0,'木头获取数量'].describe()
```

```
count        41439.000
mean      10712783.554
std       34752584.812
min              0.000
25%        2132829.500
50%        5267091.000
75%       10249321.000
max     1239962311.000
Name: 木头获取数量, dtype: float64
```

```
data.loc[data['付费金额']==0,'木头获取数量'].describe()
```

```
count     2246568.000
mean       265084.447
std        887773.641
min             0.000
25%             0.000
50%         40088.000
75%        144657.000
max     158249380.000
Name: 木头获取数量, dtype: float64
```

资源量上氪金用户占绝对主导

（2）分布与偏度

探索游戏是否对新手不友好，以及资源获取难度设置是否合理

在进行简单数据探索时，我们使用.describe()探索过特征的分布情况，并且发现大量的特征都是严重左偏的(即大部分值集中在左侧，分布上的高峰偏左严重)。

```
data.describe().T
```

字段解释	count	mean	std	min	25%	50%	75%	max
玩家唯一ID	2288007.000	1529543.498	939939.279	1.000	749992.500	1419095.000	2299006.500	3190530.000
木头获取数量	2288007.000	454306.859	4958667.146	0.000	0.000	42038.000	153118.000	1239962311.000
木头消耗数量	2288007.000	369843.252	3737720.038	0.000	0.000	9830.000	98557.000	799587506.000
石头获取数量	2288007.000	189778.774	4670619.517	0.000	0.000	0.000	0.000	1214869437.000
石头消耗数量	2288007.000	137607.363	3370166.356	0.000	0.000	0.000	0.000	796237770.000
象牙获取数量	2288007.000	80756.230	2220540.322	0.000	0.000	0.000	0.000	574496104.000
象牙消耗数量	2288007.000	36131.699	1782498.688	0.000	0.000	0.000	0.000	448197157.000
肉获取数量	2288007.000	585515.505	5868629.397	0.000	0.000	34587.000	136001.000	1470643810.000
肉消耗数量	2288007.000	354810.206	3400632.455	0.000	0.000	6470.000	66054.000	888953714.000

数据左偏说明游戏中拥有大量资源人较少，因为相比起中线，大部分人在资源/等级上都值离原点更近。

如果所有的用户都留存到7日之后，资源数据分布应该是接近正太分布，现在我们已经知道用户在前7天是严重流失的，资源分布会左偏也合理
但为何用户会流失如此严重呢?
可能从流量端开始就有问题（比如投放到了不太合适的流量渠道)
投放的广告可能有问题（比如让用户误以为是其他类型游戏)，
也可能是游戏美工不足、吸引力较低等方面
（在75%的用户看一眼就被劝退的前提下，有理由相信游戏美工和流畅度有较大问题)。
除了上述理由，用户大量流失还可能是游戏对新手不太友好或者游戏平衡性严重受氪金影响导致的。
如果游戏中资源获取很困难，比如大部分资源都需要较长的时间或氪金来进行积累
(这可能是刺激氪金的策略:新手阶段不友好，游戏上手难度大)，
大部分玩家在10~15分钟内无法获得足够的刺激与获得感，那用户在半小时内流失的可能性就很高。
我们可以尝试着观察几个资源层次的平均游戏时间，以此来判断游戏中的资源是否难以获得:

在现有数据下，我们可以通过分析资源存量来分析游戏中资源获取难度分类的合理性。

资源的分布本身就可以帮助进行分析。由于数据量过大的缘故，我们无法对所有的特征都绘制分布图，但我们可以计算所有特征的偏度与峰度。

1.偏度的中间值为0，正偏度则说明分布左偏，负偏度则说明分布右偏

2.偏度值越大，则说明分布偏移得越厉害。例如，偏度远远大于0时，说明大部分数据都集中在原点附近。

3.正态分布的峰度为3(和巨大峰度相比约等于0)，大于3的峰度说明山峰陡峭，小于3的峰度说明山峰平缓

4.峰度值越大，则说明数据的分布越不均匀、越集中于某些值。

对游戏数据而言，偏度与峰度衡量的内容是类似的:偏度说明分布对称的情况

峰度说明分布集中在少数点的情况，巨大的正偏态和巨大的峰度都指向"大部分玩家只能收集到很少资源/或无资源"的状态。

结合单一资源的均值，我们可以进一步判断资源获取的真实情况。

在这里，我们只衡量资源/技能等级，不衡量PVP次数等涉及到对战的因素。

```
1 data.columns
```

```
Index(['玩家唯一ID', '玩家注册时间', '木头获取数量', '木头消耗数量', '石头获取数量', '石头消耗数量', '象牙获取数量',
       '象牙消耗数量', '肉获取数量', '肉消耗数量',
       ...
       'PVP次数', '主动发起PVP次数', 'PVP胜利次数', 'PVE次数', '主动发起PVE次数', 'PVE胜利次数',
       '在线时长', '付费金额', '付费次数', '45日付费金额'],
      dtype='object', name='字段解释', length=109)
```

```
resource = data.iloc[:,2:-10]
```

```
col = []
skew = []
kurt = []
mean = []
for i,ColName in enumerate(resource.columns):
    col.append(ColName)
    skew.append(data.loc[:,ColName].skew())#偏度
    kurt.append(data.loc[:,ColName].kurt())#峰度
    mean.append(data.loc[:,ColName].mean())#均值
sak = pd.concat([pd.DataFrame(col),pd.DataFrame(skew),pd.DataFrame(kurt),pd.DataFrame(mean)],axis=1)
sak.columns = ['特征','偏度','峰度','均值']
sak.head()
```

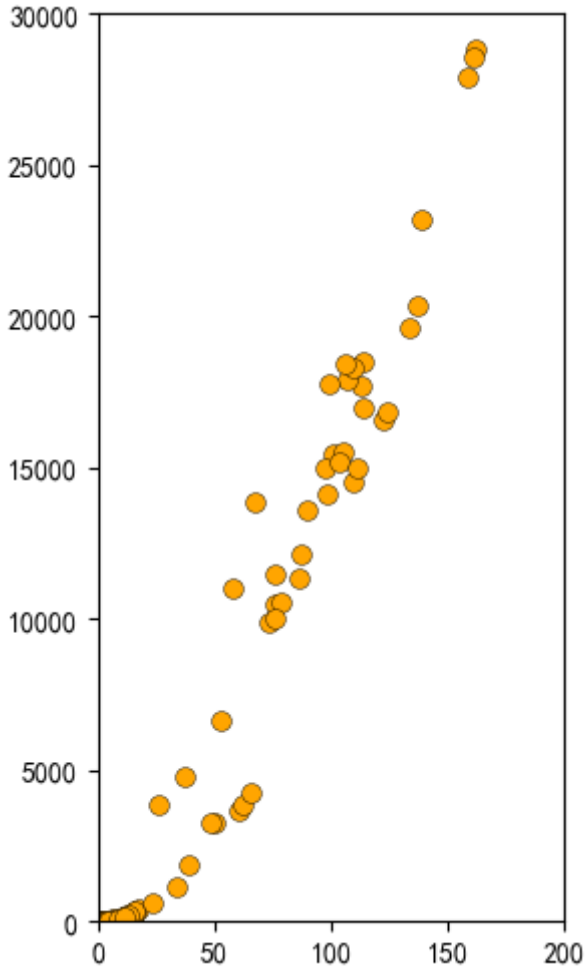
	特征	偏度	峰度	均值
0	木头获取数量	100.635	15449.582	454306.859
1	木头消耗数量	86.310	11367.448	369843.252
2	石头获取数量	113.794	18489.107	189778.774
3	石头消耗数量	105.043	15515.988	137607.363
4	象牙获取数量	112.636	17682.030	80756.230

```
#避免显示问题，设置一下最大行数
pd.set_option('display.max_rows',120)
sak.sort_values('偏度')
```

	特征	偏度	峰度	均值
34	建筑：要塞等级	1.229	0.970	2.098
35	建筑：据点传送门等级	1.383	1.477	1.764
32	建筑：士兵小屋等级	1.878	3.314	1.306
42	建筑：魔法幸运树等级	2.050	3.388	1.146
36	建筑：兵营等级	2.114	4.327	1.284
33	建筑：治疗小井等级	2.183	4.567	1.027
41	建筑：瞭望塔等级	2.409	5.619	0.913
37	建筑：治疗之泉等级	2.429	5.627	0.924
40	建筑：仓库等级	2.472	5.525	0.934
38	建筑：智慧神庙等级	2.495	5.893	0.969

峰度和偏度似乎呈现高度一致的趋势——当偏度很大时，峰度也倾向于很大。

```
plt.figure(figsize=[3,6],dpi=100)
plt.scatter(sak.loc[:,'偏度'],sak.loc[:,'峰度'],s=50,c='orange',edgecolor='k',linewidth=0.3)
plt.xlim(0,200)
plt.ylim(0,30000)
plt.show()
```



因此我们可以只考虑偏度与均值，不再考虑峰度的情况了。

如果我们能够绘制横坐标为偏度，纵坐标为均值的散点图，就可以概括出偏度与均值之间的关系，并总结出不同资源/不同等级的设置类型，以此来判断游戏中的各项资源的获得难易程度是否合理。

在绘图之前，我们大致可以判断，均值与偏度应该可以以如下方式分割特征的类型:

·A类均值高，偏度低:用户拥有该资源的分布较为均匀，并且大家都拥有很多这个资源，这类资源是最容易获取的

·B类均值高，偏度也高:用户拥有该资源的分布不均匀，少量玩家掌握巨量资源，拉高了均值，这类资源可能是氪金资源，也可能是爆肝才能获得的资源(高门槛)

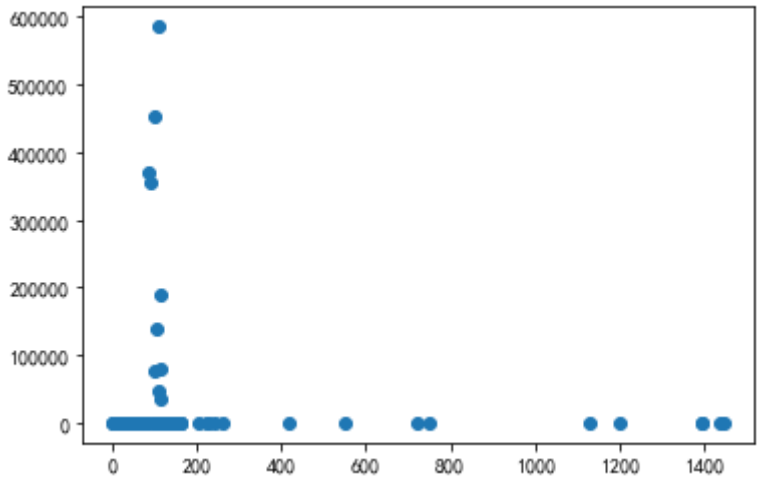
·C类均值低，偏度也低:用户拥有该资源的分布较为均匀，但大家都没有这个资源，这类资源可能是需要时间来积累才能够获得的，随着开服时间变长，这类资源应该会逐渐变成其他类型A/B/D的资源

·D类均值低，偏度高:用户拥有该资源的分布不均匀，只有少量玩家掌握该资源，但是掌握的总量也不多。这类资源天生稀缺并且无法靠氪金获得（比如必须触发特定剧情/达到特定成就才能够获得的资源)，只有少数人能够拥有

在一个健康的游戏中，A类资源应该是最多的，这类资源决定了剧本能够被推进的程度、决定了大部分玩家的游戏体验，B类资源主要用于付费，收入的主要来源，但可能严重破坏游戏平衡，D类资源主要在于增加游戏的趣味和公平性，应该是最少的。一个游戏的A类资源要足够多，才算是对新手比较友好，只要我们绘制出散点图，四类资源的分布情况就目了然了。

现在的难点在于，偏度与均值都包括部分极大异常点，因此数据极差非常高，而大部分数据点又集中在较小的值附近，这样的数据在绘图时很容易出现以下情况:

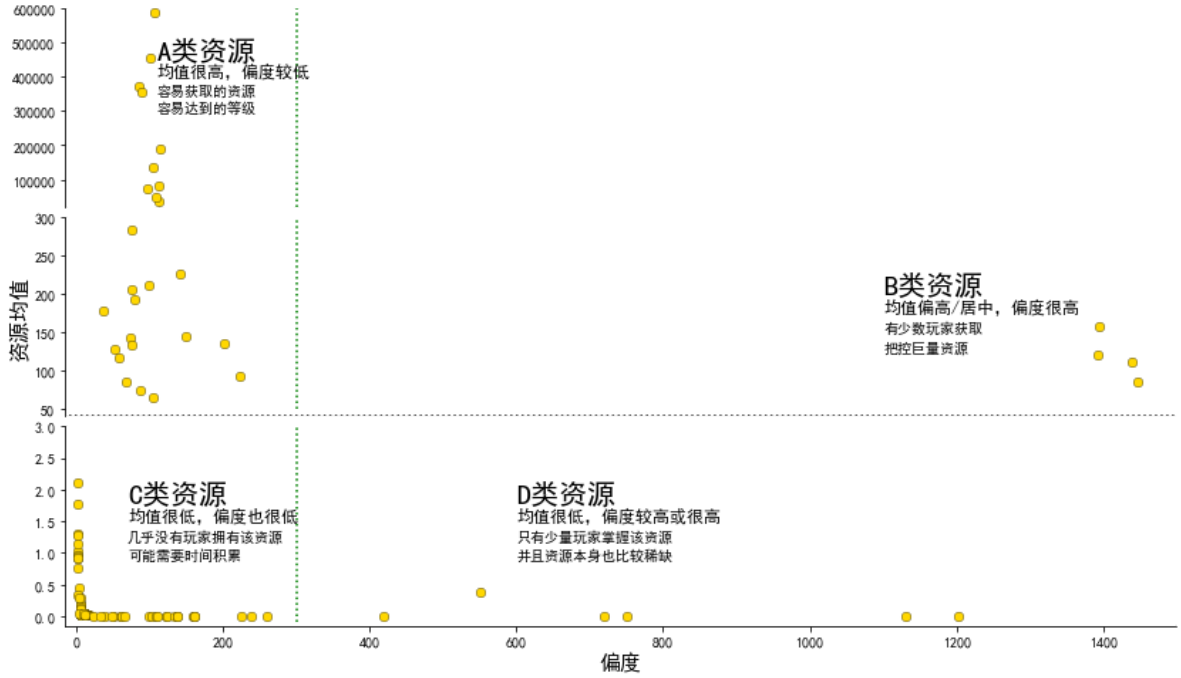
```
# 所有数据堆在一起，看不出任何规律
plt.scatter('偏度','均值',data=sak)
plt.show()
```



为了解决这个问题，需要绘制三个纵坐标数值不同的图像：

```
plt.figure(figsize=(15,9),dpi=100)
#绘制三个坐标轴不一样的图像,省略上中的横坐标

#上
plt.subplot(311)
plt.scatter(sak.loc[:,'偏度'],sak.loc[:,'均值']
            ,s=45
            ,c='gold'
            ,edgecolors='k'
            ,linewidth=0.3
            )
plt.xlim(-15,1500)
plt.ylim(20000,600000)
plt.vlines(300,20000,600000,colors='green',linestyles='dotted')
ax = plt.gca()
ax.spines['top'].set_visible(False)
ax.spines['right'].set_visible(False)
ax.spines['bottom'].set_visible(False)
plt.yticks(fontsize=10)
plt.text(110,450000,s='A类资源',fontsize=20)
plt.text(110,400000,s='均值很高，偏度较低',fontsize=12)
plt.text(110,350000,s='容易获取的资源')
plt.text(110,300000,s='容易达到的等级')
plt.xticks([])

#中
plt.subplot(312)
plt.scatter(sak.loc[:,'偏度'],sak.loc[:,'均值']
            ,s=45
            ,c='gold'
            ,edgecolors='k'
            ,linewidth=0.3
            )
plt.xlim(-15,1500)
plt.ylim(40,300)
plt.vlines(300,50,300,colors='green',linestyles='dotted')
```

```
plt.hlines(40,-10,1500,colors='black',linestyles='dotted')
ax = plt.gca()
ax.spines['top'].set_visible(False)
ax.spines['right'].set_visible(False)
ax.spines['bottom'].set_visible(False)
plt.yticks(fontsize=10)
plt.ylabel('资源均值',fontsize=15)
plt.text(1100,200,s='B类资源',fontsize=20)
plt.text(1100,175,s='均值偏高/居中，偏度很高',fontsize=12)
plt.text(1100,150,s='有少数玩家获取')
plt.text(1100,125,s='把控巨量资源')
plt.xticks([])

#下
plt.subplot(313)
plt.scatter(sak.loc[:,'偏度'],sak.loc[:,'均值']
            ,s=45
            ,c='gold'
            ,edgecolors='k'
            ,linewidth=0.3
            )
plt.xlim(-15,1500)
plt.ylim(-0.15,3)
plt.vlines(300,-0.1,5,colors='green',linestyles='dotted')
ax = plt.gca()
ax.spines['top'].set_visible(False)
ax.spines['right'].set_visible(False)
plt.xlabel('偏度',fontsize=15)
plt.yticks(fontsize=10)
plt.text(70,1.8,s='C类资源',fontsize=20)
plt.text(70,1.5,s='均值很低，偏度也很低',fontsize=12)
plt.text(70,1.2,s='几乎没有玩家拥有该资源')
plt.text(70,0.9,s='可能需要时间积累')
plt.text(600,1.8,s='D类资源',fontsize=20)
plt.text(600,1.5,s='均值很低，偏度较高或很高',fontsize=12)
plt.text(600,1.2,s='只有少量玩家掌握该资源')
plt.text(600,0.9,s='并且资源本身也比较稀缺')
plt.subplots_adjust(hspace=0.05)
plt.show()
```



规定偏度大于300认为大，小于300认为小。均值大于3认为大，小于3认为小。

查看一下四类资源分别有哪些，以及数量是否合理：

```
#A类:最容易获取的资源
A = sak.loc[sak['均值']>3,:].loc[sak['偏度']<300,:]
A
```

	特征	偏度	峰度	均值
0	木头获取数量	100.635	15449.582	454306.859
1	木头消耗数量	86.310	11367.448	369843.252
2	石头获取数量	113.794	18489.107	189778.774
3	石头消耗数量	105.043	15515.988	137607.363
4	象牙获取数量	112.636	17682.030	80756.230
5	象牙消耗数量	113.405	16974.594	36131.699
6	肉获取数量	106.645	17907.029	585515.505
7	肉消耗数量	89.403	13600.967	354810.206
8	魔法获取数量	97.762	14958.235	75389.535
9	魔法消耗数量	109.334	18322.774	47253.994
10	勇士招募数量	150.291	56627.286	143.610
11	勇士损失数量	142.013	59680.484	226.782
12	驯兽师招募数量	52.394	6653.517	128.264
13	驯兽师损失数量	37.613	4799.376	178.006
16	勇士伤兵产生数量	201.952	96877.947	135.304
17	勇士伤兵恢复数量	223.474	111369.002	91.984
18	驯兽师伤兵产生数量	58.235	10995.135	116.503
19	驯兽师伤兵恢复数量	67.356	13844.793	84.346
22	通用加速获取数量	76.190	10461.629	282.545
23	通用加速使用数量	78.756	10571.988	192.031
24	建筑加速获取数量	75.562	11468.080	205.557
25	建筑加速使用数量	73.617	9912.930	142.776
26	科研加速获取数量	75.606	10038.624	132.671
27	科研加速使用数量	86.781	12166.169	73.653
28	训练加速获取数量	99.010	17738.981	210.808
29	训练加速使用数量	105.763	18411.777	65.022
30	治疗加速获取数量	26.467	3883.361	10.416

```
A.shape
#(27, 4)
```

```
#B类:可能与氪金有关
B = sak.loc[sak['均值']>3,:].loc[sak['偏度']>300,:]
B
```

	特征	偏度	峰度	均值
14	萨满招募数量	1393.339	2048745.112	119.543
15	萨满损失数量	1394.118	2050265.411	156.853
20	萨满伤兵产生数量	1439.747	2141351.573	110.431
21	萨满伤兵恢复数量	1446.870	2155525.025	84.763

```
#C类--大量还未开发的资源&大量的等级
C = sak.loc[sak['均值']<3,:].loc[sak['偏度']<300,:]
C
```

	特征	偏度	峰度	均值
32	建筑：士兵小屋等级	1.878	3.314	1.306
33	建筑：治疗小井等级	2.183	4.567	1.027
34	建筑：要塞等级	1.229	0.970	2.098
35	建筑：据点传送门等级	1.383	1.477	1.764
36	建筑：兵营等级	2.114	4.327	1.284
37	建筑：治疗之泉等级	2.429	5.627	0.924
38	建筑：智慧神庙等级	2.495	5.893	0.969
39	建筑：联盟大厅等级	4.015	15.755	0.441
40	建筑：仓库等级	2.472	5.525	0.934
41	建筑：瞭望塔等级	2.409	5.619	0.913
42	建筑：魔法幸运树等级	2.050	3.388	1.146
43	建筑：战争大厅等级	6.427	64.375	0.102
44	建筑：联盟货车等级	2.727	7.144	0.772
45	建筑：占卜台等级	5.722	35.427	0.201
46	建筑：祭坛等级	6.024	40.077	0.180
47	建筑：冒险传送门等级	5.017	26.012	0.292
48	科研：侦查等级	2.923	10.701	0.331
49	科研：训练速度等级	3.815	14.593	0.299
50	科研：守护者	6.236	36.893	0.024
51	科研：巨兽驯兽师	6.372	38.602	0.023
52	科研：吟唱者	6.914	45.804	0.020
53	科研：勇士攻击	15.486	312.204	0.024
54	科研：驯兽师攻击	16.032	333.790	0.022
55	科研：萨满攻击	17.736	407.577	0.019
56	科研：战斗大师	60.287	3632.539	0.000
57	科研：高阶巨兽骑兵	61.883	3827.516	0.000
58	科研：图腾大师	65.373	4271.658	0.000
59	科研：部队防御	122.425	16602.243	0.001
60	科研：勇士防御	158.768	27877.942	0.000
61	科研：驯兽师防御	161.650	28807.251	0.000
62	科研：萨满防御	161.270	28593.152	0.000
63	科研：勇士生命	124.296	16848.445	0.001
64	科研：驯兽师生命	134.002	19641.283	0.001
65	科研：萨满生命	137.095	20351.197	0.001
66	科研：狂战士	239.159	57195.300	0.000
67	科研：龙骑兵	225.481	50839.711	0.000
68	科研：神谕者	259.406	67289.471	0.000
70	科研：建造速度	6.780	53.866	0.122
71	科研：资源保护	11.338	173.489	0.035

72	科研：部队消耗	12.375	204.846	0.033
73	科研：木材生产	11.439	162.607	0.035
74	科研：石头生产	38.649	1881.487	0.003
75	科研：象牙生产	98.607	14098.969	0.001
76	科研：肉类生产	11.180	163.219	0.042
77	科研：木材采集	15.330	295.240	0.022
78	科研：石头采集	49.888	3234.002	0.002
79	科研：象牙采集	103.602	15170.809	0.000
80	科研：肉类采集	15.750	330.005	0.018
81	科研：部队负重	13.202	198.280	0.029
82	科研：魔法采集	48.295	3245.438	0.002
83	科研：魔法生产	23.047	645.105	0.008
84	科研：据点耐久	5.959	43.095	0.110
85	科研：据点二	4.426	17.592	0.044
86	科研：医院容量	10.935	153.092	0.029
87	科研：领土采集奖励	9.219	101.032	0.034
88	科研：治疗速度	11.462	157.764	0.022
89	科研：据点三	33.686	1132.749	0.001
90	科研：联盟行军速度	138.627	23213.480	0.000
91	科研：战斗行军速度	109.113	14509.348	0.001
92	科研：采集行军速度	111.196	14976.855	0.001

```
C.shape[0]
#60
```

```
# D类–极度稀缺的资源
D = sak.loc[sak['均值']<3,:].loc[sak['偏度']>300,:]
D
```

	特征	偏度	峰度	均值
31	治疗加速使用数量	551.767	567898.986	0.370
69	科研：部队攻击	720.086	552951.148	0.000
93	科研：据点四	419.521	175995.923	0.000
94	科研：增援部队容量	1202.368	1569458.101	0.000
95	科研：行军大小	750.277	623089.356	0.000
96	科研：资源帮助容量	1130.278	1333821.383	0.000

从分析的结果来看，各项资源的配比数量其实还算合理。
假设一款游戏只想服务氪金玩家，那B类资源会很多、大量A类资源的获取渠道也会转向氪金方向，但《野蛮时代》明显没有这样的设计。
在游戏早期只有兵种萨满相关的集中资源和技能是需要氪金的，说明游戏依然在致力于为普通玩家提供较好的体验
所以大部分用户应该不是因为新手关太难、剧情无法推进才退游的。
然而，各类资源的可获取量其实有很大的差异，基础资源动辄就上百万甚至上亿，可能会让用户获得感严重不足。
而从A类资源的分布来看，只有木头和肉这两种最最最基础的资源是分布较为正常的，其他资源都是严重左偏，说明"留存率低"是原因，“资源数据的异常"是结果。
最有可能的情况就是产品逻辑不顺、获得感不足、美工不足、系统BUG(比如疯狂闪退等等)、或者强烈的捞金感/不高级感让大部分用户在第一天就流失掉了。

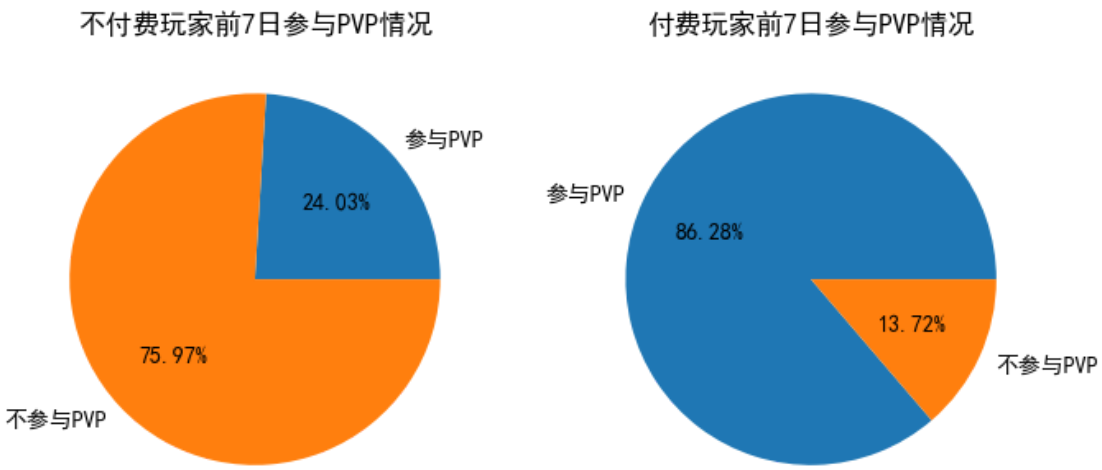
（3）平衡性影响—氪金与战斗优势

如果游戏对于新手并非很难上手，那是否是战斗力系统的设置让非氪玩家感觉不太友好呢?过于强大的角色、过于强大的技能和道具都可能影响游戏平衡性，平衡性这个复杂的指标其实可以从很多方面来进行考虑，可惜在现有数据下，我们能够考虑的唯有胜率。我们可以分析氪金用户与不氪金用户的战力差异，来判断氪金对游戏平衡性的影响。

- **PVP战斗的参与程度对比分析——付费与非付费**

  7日内玩家参与PVP情况

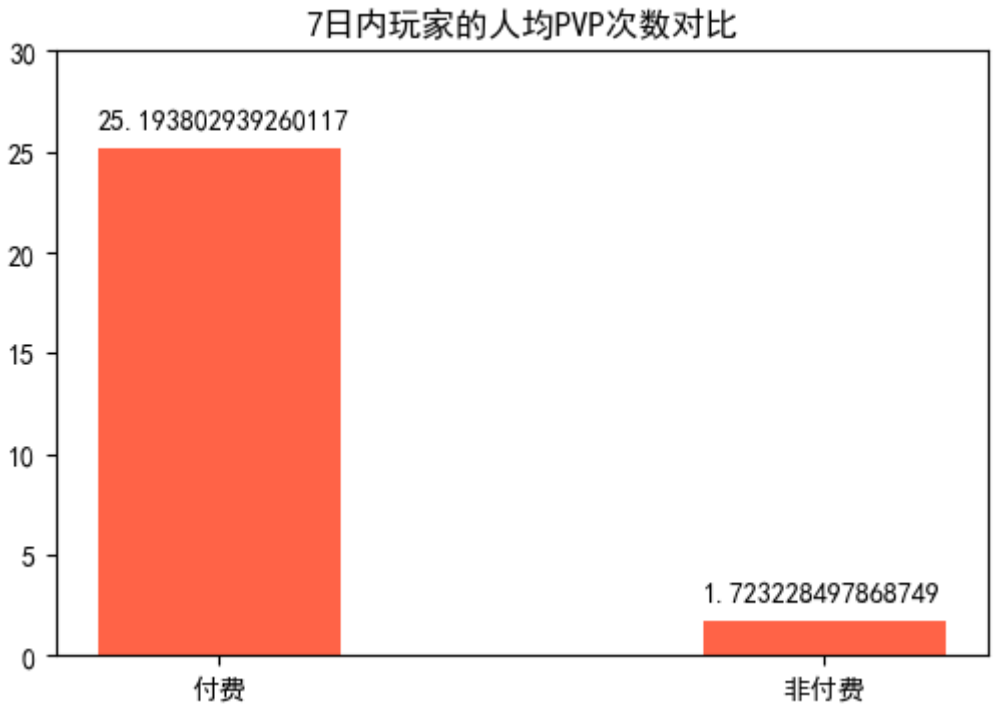
```
plt.figure(figsize=(8,8),dpi=100)
plt.subplot(121)
per1 = 539947/data[data['付费金额']==0].shape[0]
#13%的氪金玩家前7日不玩PVP
plt.pie(x=[per1,1-per1],labels=['参与PVP','不参与PVP'],autopct="%0.2f%%")
plt.title('不付费玩家前7日参与PVP情况')

plt.subplot(122)
per2 = 35755/data[data['付费金额']!=0].shape[0]
#13%的氪金玩家前7日不玩PVP
plt.pie(x=[per2,1-per2],labels=['参与PVP','不参与PVP'],autopct="%0.2f%%")
plt.title('付费玩家前7日参与PVP情况')
```



付费玩家参与PVP的比例很高，当然，不氪金用户中有许多人可能是已经流失的用户，因此当然不会参与PVP对战。

7日内玩家的人均PVP次数

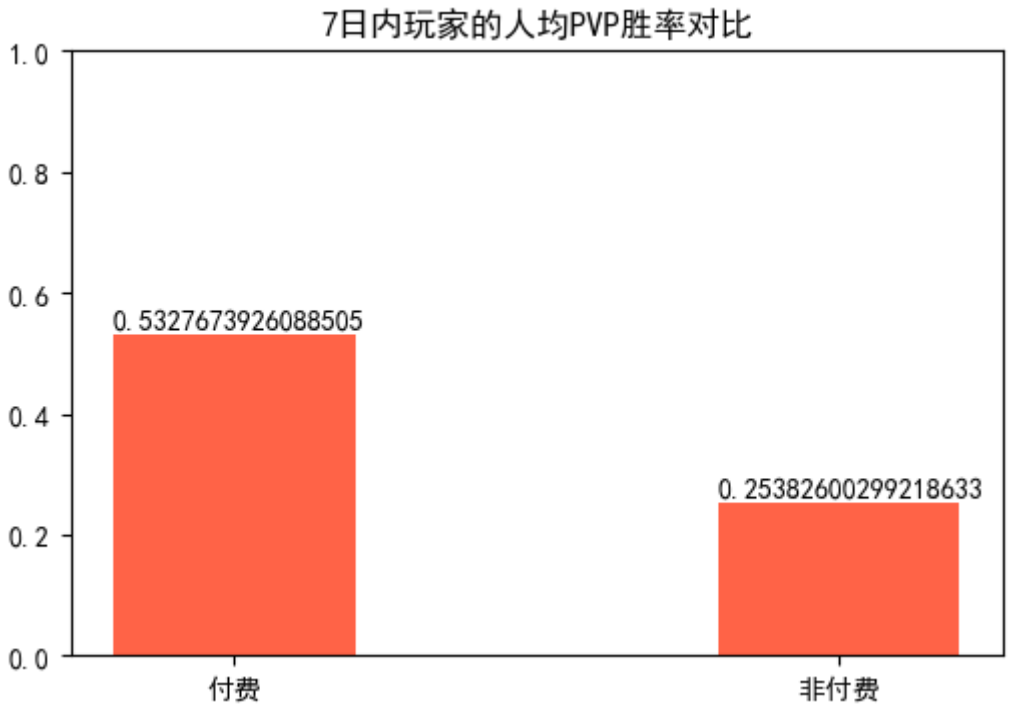
```
plt.figure(figsize=(6,4),dpi=100)
a = ['付费','非付费']
b = [data[data['付费金额']!=0]['PVP次数'].mean(),data[data['付费金额']==0]['PVP次数'].mean()]
plt.bar(a,b,width = 0.4,color='tomato')
plt.ylim(0,30)
for i in range(len(a)):
    plt.text(i-0.2,b[i]+1,s=b[i] ) # s表示注释内容
plt.title('7日内玩家的人均PVP次数对比')
plt.show()
```



氪金玩家人均参与的PVP战斗次数远高于非付费玩家

7日内玩家PVP胜率均值对比

```
PaidV = data[data['付费金额']!=0]['PVP胜利次数']/data[data['付费金额']!=0]['PVP次数']
NPaidV = data[data['付费金额']==0]['PVP胜利次数']/data[data['付费金额']==0]['PVP次数']

plt.figure(figsize=(6,4),dpi=100)
a = ['付费','非付费']
b = [PaidV.mean(),NPaidV.mean()]
plt.bar(a,b,width = 0.4,color='tomato')
plt.ylim(0,1)
for i in range(len(a)):
    plt.text(i-0.2,b[i]+0.01,s=b[i] ) # s表示注释内容
plt.title('7日内玩家的人均PVP胜率对比')
plt.show()
```



可以看到，付费玩家人均PVP胜率是53.3%，对手游付费玩家来说是个可以接受的数字，但并不是特别大的优势。对非付费玩家而言，人均PVP胜率只有25%，并且75%的人的场均胜率不到50%，这个条件对非付费玩家来说的确比较苛刻。

同样，不氪金玩家中也有部分玩家的PVP次数为0，即一次也没有参与。来看不氪金玩家中不参与PVP的人数占比:

前7日主动发起PVP的比率对比

```
(data[data['付费金额']==0]['主动发起PVP次数']/data[data['付费金额']==0]['PVP次数']).describe()
```

```
count   539947.000
mean         0.262
std          0.365
min          0.000
25%          0.000
50%          0.000
75%          0.500
max          1.000
dtype: float64
```

```
(data[data['付费金额']!=0]['主动发起PVP次数']/data[data['付费金额']!=0]['PVP次数']).describe()
```

```
count   35755.000
mean        0.452
std         0.357
min         0.000
25%         0.000
50%         0.500
75%         0.783
max         1.000
dtype: float64
```

不付费玩家平均26%的时候主动发起对战，不过大多数时候75%的人都不会发起对战。如果PVP状况下是付费玩家占绝对优势，那PVE也是相同的情况吗?

- **PVE战斗的参与程度对比分析——付费与非付费**

玩家PVE胜率对比

```
#氪金玩家PVE胜率分布
(data[data['付费金额']!=0]['PVE胜利次数']/data[data['付费金额']!=0]['PVE次数']).describe()
```

```
count   37389.000
mean        0.885
std         0.141
min         0.000
25%         0.840
50%         0.926
75%         0.988
max         1.000
dtype: float64
```

```
#不氪金玩家PVE胜率分布
(data[data['付费金额']==0]['PVE胜利次数']/data[data['付费金额']==0]['PVE次数']).describe()
```

```
count   703148.000
mean         0.921
std          0.179
min          0.000
25%          0.965
50%          1.000
75%          1.000
max          1.000
dtype: float64
```

不难发现，不氪金玩家的PVE平均胜率有92%，是高于氪金玩家的胜率88%的，看来环境怪兽都比较弱小,大部分玩家都能够轻松战胜。
当PVP上氪金玩家有绝对优势时，不氪金玩家许多会选择在PVE环境中进行战斗,因此我们推断不氪金玩家主动发起PVE战斗的情况会更多:

```
#不氪金玩家主动发起PVE的概率
(data[data['付费金额']==0]['主动发起PVE次数']/data[data['付费金额']==0]['PVE次数']).describe()
```

```
count   703148.000
mean         0.996
std          0.056
min          0.000
25%          1.000
50%          1.000
75%          1.000
max          1.000
dtype: float64
```

```
#氪金玩家主动发起PVE的概率
(data[data['付费金额']!=0]['主动发起PVE次数']/data[data['付费金额']!=0]['PVE次数']).describe()
```

```
count   37389.000
mean        0.991
std         0.067
min         0.000
25%         1.000
50%         1.000
75%         1.000
max         1.000
dtype: float64
```

氪金与不氪金玩家主动放弃PVE的可能性都非常高，几乎所有人都是主动发起PVE战斗
这可能说明PVE战斗是发展城市必须的环节(比如收集资源肉类需要猎杀野猪等生物、获取魔法道具需要猎杀龙、哥布林等生物)。

从分析的结果来看，氪金并不会对PVE战斗有影响，主要是影响PVP。

《野蛮游戏》中玩家或许可以夺取其他玩家的领地和城池，因此PVP对战的平衡性对玩家流失有重要的影响。
前期留存下来，但后期逐渐流失的非氪金用户有一定的可能是因为PVP胜率太低而离开。

（4）左偏带来长尾:谁是异常玩家?

在特征高度左偏的情况下，数据中必然含有大量的异常值，这些异常值可能代表着某些特殊的人群，但带入异常值进行建模会严重影响模型的稳定性，导致模型容易过拟合。因此在建模之前，我们需要对异常值进行简单的探索，以确定异常值的处理方法。

查看业务上的异常值

从游戏业务角度而言，所有的资源/人物等级/胜利次数/金额等不应该出现负数，因此首先排查负数

如果存在用户信息，则需要排查年龄不为负，年龄不过高或过低等状况

查看是否有上线时间较短，但是资源量异常丰富的账号，可能是GM，也可能有人作弊

```
#不对ID、注册时间和标签（45日付费金额）计算异常值
AbnormalCheck = data.iloc[:,2:-1]
#确认没有object对象
AbnormalCheck.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2288007 entries, 0 to 2288006
Columns: 106 entries, 木头获取数量 to 付费次数
dtypes: float64(12), int64(94)
memory usage: 1.8 GB
```

```
#无负值
(AbnormalCheck<0).sum().sum()
#0
```

使用箱线图法则，看看会筛选出多少异常值

```
#打印异常比例，保存异常样本索引
NumOfSamples = data.shape[0]
abnormal = pd.DataFrame()
DataNoAbnormal = data.copy()
BoxAbnormalIdx = [] #用来保存异常值的索引
for idx,column in enumerate(data.columns[2:-1]):
    feature = data.loc[:,column]
    QL = np.quantile(feature,0.25)
    QU = np.quantile(feature,0.75)
    IQU = QU - QL
    #过大或过小的都属于异常值
    ab = feature[((feature < (QL - 1.5*IQU)).astype(int) + (feature > (QU+1.5*IQU)).astype(int))!=0]
    BoxAbnormalIdx.extend(ab.index)#extend与append的区别
    abnormal.loc[idx,'特征'] = column
    abnormal.loc[idx,'异常值数量'] = ab.shape[0]
    abnormal.loc[idx,'异常值比例'] = '{:.3f}%'.format(ab.shape[0]*100/NumOfSamples)
```

```
#这里面是有重复值的
len(BoxAbnormalIdx)
```

```
# 去重
BoxAbnormalIdx=set(BoxAbnormalIdx)
len(BoxAbnormalIdx)
#1123706
```

超过100万数据被归为异常，数量太多，况且付费用户只有4万多，很有可能被这100多万数据囊括。

```
#左偏比较严重的特征 异常值更多
abnormal.sort_values('异常值比例',ascending=False)
```

在箱线图规则下，查看异常用户的氪金情况

```
# 异常用户中前7日付费用户的数量
(data.loc[BoxAbnormalIdx,'付费金额']!=0).sum()
#41439
```

```
# 前7日所有付费用户的数量
(data.loc[:,'付费金额']!=0).sum()
#41439
```

```
# 异常用户中前45日付费用户的数量
(data.loc[BoxAbnormalIdx,'45日付费金额']!=0).sum()
#45670
```

```
# 前45日所有付费用户的数量
(data.loc[:,'45日付费金额']!=0).sum()
#45988
```

前7日付费用户被100%包含在了异常用户里，这说明:

付费用户的数据在现有数据的大背景下,(大部分用户短时间快速流失掉)下，在统计上是完全异常的，因此建模时想要辨认出付费角户，模型需要很深入的学习，有很高的过拟合风险。异常用户有100w，其中只有4w多付费，即许多与氪金用户行为高度相似的用户都没有氪金，对算法来说这部分用户是很难判断的

在进行算法建模时，需要更加灵活的方法来处理这些异常数据。在进行了这么多探索之后，我们开始进入建模流程。我们将在建模流程中使用我们之前分析的众多结果来处理特征。

三、数据预处理

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import sklearn
from sklearn.linear_model import LinearRegression as LR
from sklearn.model_selection import train_test_split as TTS
from sklearn.metrics import mean_squared_error as MSE
%matplotlib inline
plt.rcParams['font.sans-serif'] = ['Simhei']
plt.rcParams['axes.unicode_minus'] = False
```

1.数据预处理：注册时间与氪金状况

从游戏世界的常识以及之前的分析结果来看，资源量的累计、在线时长等特征与用户体验的深度有较大的关联，也就与是否氪金、氪多少金额有较大的关联。

但在我们的特征矩阵中，还有一类尚未分析的特征，玩家注册时间，这是全数据集中唯一一个object类型的对象。

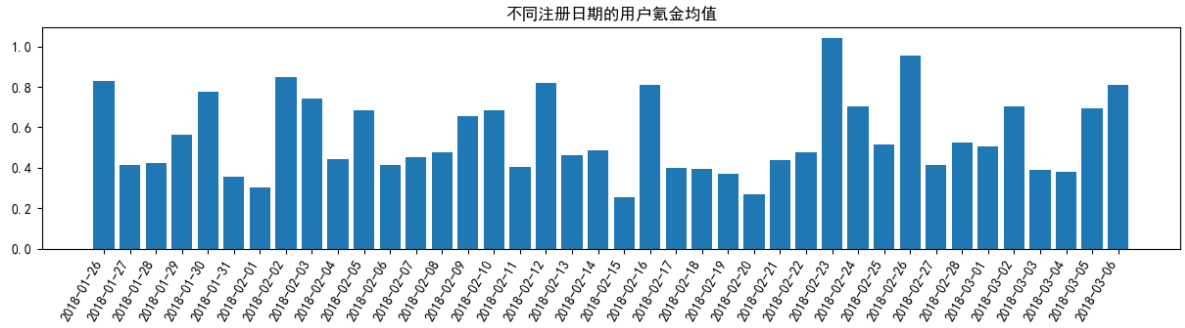
从常识来看，一般注册时间与用户的氪金行为应该关系不大。

在这里我们将时间分成日期和时刻来考虑，分别绘制横坐标为注册日期/注册时刻、纵坐标为该日期/该时刻注册用户氪金均值的关系图，以此来观察注册日期.时刻与氪金是否有关。

- **不同注册日期的用户氪金均值**

```
#将玩家注册日期提取出来作为新的一列
import time
import datetime
RegisterDate = data.loc[:,'玩家注册时间'].apply(lambda x:x[:10])
```

```
# 按照玩家注册日期对氪金玩家7日付费金额进行聚合平均计算
RegisterDateMean = data['付费金额'].groupby(RegisterDate).mean()
#绘制图像
plt.figure(figsize=(15,3),dpi=100)
plt.title('不同注册日期的用户氪金均值')
plt.bar(RegisterDateMean.index,RegisterDateMean.values)
plt.xticks(RegisterDateMean.index,rotation=60,ha='right')
plt.show()
```



从图上来看，几乎无法看出日期与氪金数额有所联系。

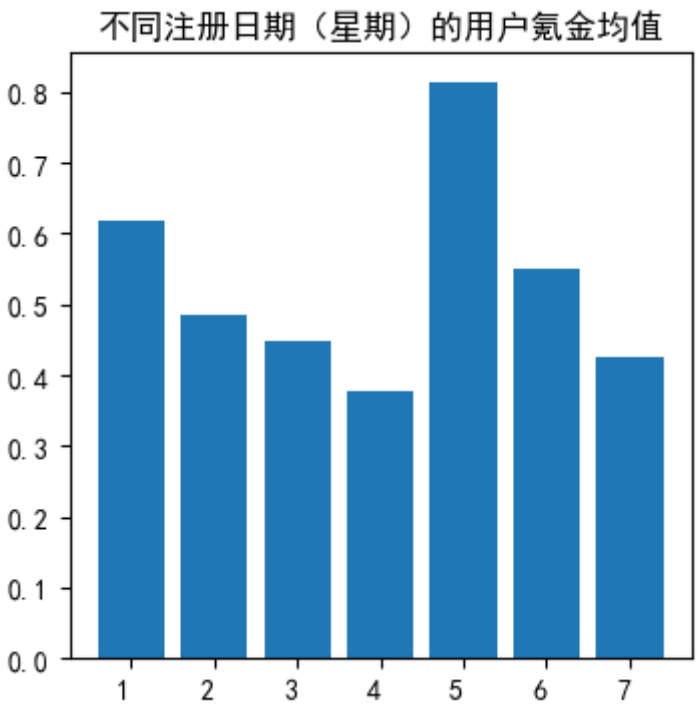
看起来整体是比较随机而且杂乱的，但是每隔几天就会有一个高峰。

我们可以试试看将日期转化为一星期中的七天，观察一下星期和氪金数额是否有明显的关系。

注：2018-01-26是腊月初十，2018-2-15除夕，2018-2-16大年初一，2018-3-2元宵

- **不同注册日期（星期）的用户氪金情况**

```
RegisterWeekDay = data.loc[:,'玩家注册时间'].apply(lambda x :
datetime.datetime.strptime(x[:10],'%Y-%m-%d').weekday()+1)
#按星期分组聚合
RegisterWeekDayMean = data['付费金额'].groupby(RegisterWeekDay).mean()
#绘制图像
plt.figure(figsize=(4,4),dpi=100)
plt.title('不同注册日期（星期）的用户氪金均值')
plt.bar(RegisterWeekDayMean.index,RegisterWeekDayMean.values)
plt.xticks(RegisterWeekDayMean.index,ha='right')
plt.show()
```



很明显，周五、周一注册的人有较高的氪金金额，并且在高峰之后会逐渐下降。

这个趋势看起来是氪金金额与星期相关，但实际上更可能是跟游戏买量、投放的节奏和渠道有关。

或许每周五、每周一都可以排到质量较高的用户所在的渠道，或者周五会吸引到大量准备过周末的玩家的注意、周一则会吸引到大量还不想投入工作的玩家的注意。

周五与周一的高峰暗示这两天进入的流量可能是有规律工作、有经济收入的人。

但无论真相是什么，从特征的角度来看，我们可以计算一下星期与45日付费金额的相关系数:

```
pd.concat([data['45日付费金额'],RegisterWeekDay],axis=1).corr()
```

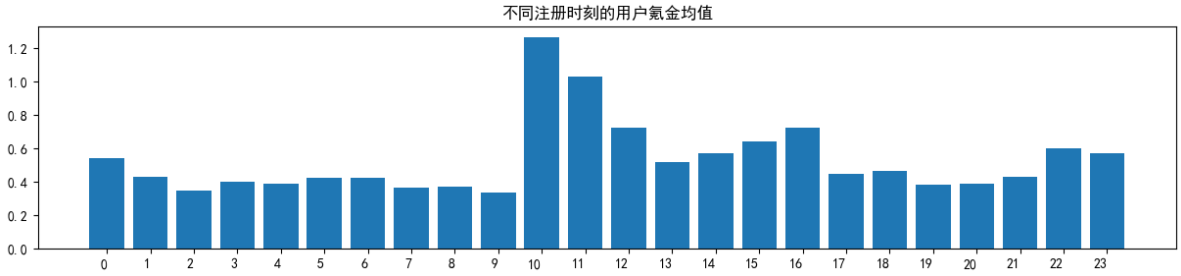
	45日付费金额	玩家注册时间
45日付费金额	1.000	0.000
玩家注册时间	0.000	1.000

相关系数非常非常小，两者几乎不相关，我们可以不将星期放入矩阵特征。

按照玩家注册时刻进行聚合，求均值

```
RegisterTimeMean = data['付费金额'].groupby(RegisterTime).mean()
```

```
# 绘制图像
plt.figure(figsize=(15,3),dpi=100)
plt.title('不同注册时刻的用户氪金均值')
plt.bar(RegisterTimeMean.index,RegisterTimeMean.values)
plt.xticks(RegisterTimeMean.index,ha='right')
plt.show()
```



早上10点-12点之间注册的用户、以及下午4点-5点之间表现出比其他时段注册的用户更高的氪金能力，这可能是与推广渠道、开服时间有关，也可能真的是注册时刻与氪金金额有关。

巧合的是，上午10点与下午4点正是上班时间中摸鱼喝茶的时间，如果能够确定这个时间在推广的是那些渠道，就能够确定相关用户的画像，来判断是否真是时间与氪金数额有关了。同时，也来看看相关系数:

```
pd.concat([data['45日付费金额'],RegisterTime],axis=1).corr()
```

	45日付费金额	玩家注册时间
45日付费金额	1.000	0.001
玩家注册时间	0.001	1.000

看起来注册时刻与付费金额有一点点关系(非常微弱)，我们可以保留注册时刻为建模用。

- **修改特征**

```
#增加'玩家注册时刻(小时)'列
data.insert(data.shape[1]-1,'玩家注册时刻',RegisterTime)
# 删除玩家ID列，不是有助于建模的特征
data.drop(columns=['玩家注册时间','玩家唯一ID'],inplace=True)
```

2.模型选择与benchmark

在2018年DC竞赛上，《野蛮时代》数据集上拿到冠军的团队实现了RMSE=40，前10名的RMSE值为50，前20名的RMSE值为57.67
我们可以以50为目标来进行调整。在建模之前，我们需要构建自己的benchmark，来对比我们的优化流程是否有效。

竞赛benchmark: https://js.dclab.run/v2/cmptDetail.html?id=226

```
# 将标签提取出来，分割标签与特征
X = data.iloc[:,:-1]
y = data.iloc[:,-1]
```

在现有数据量的要求下，我们首先要考虑计算迅速且简单的线性模型:线性回归。

- **建立benchmark**

```
from sklearn.linear_model import LinearRegression as LR
from sklearn.model_selection import train_test_split as TTS
from sklearn.metrics import mean_squared_error as MSE
```

```
#数据集分割
Xtrain,Xtest,Ytrain,Ytest = TTS(X,y,test_size=0.3,random_state=1412)
#恢复索引
for i in [Xtrain,Xtest]:
    i.index = range(i.shape[0])
```

```
reg = LR()
reg_benchmark = reg.fit(Xtrain,Ytrain)
```

```
reg_benchmark.score(Xtrain,Ytrain)
#0.5604160157526339
```

```
reg_benchmark.score(Xtest,Ytest)
#0.5581998030190463
```

模型处于欠拟合状态——在训练集和测试集的分数上都不是很高，并且两者分数相近

简单的线性模型在学习能力上略显不足

```
#测试集上的预测标签
y_pred = reg_benchmark.predict(Xtest)
```

```
# 根均方误差（RMSE）
np.sqrt(MSE(Ytest,y_pred))
#62.00116183660748
```

benchmark就是62的RMSE以及55.8%的R2。

先尽一切努力降低数据集上的RMSE。

四、特征工程

1、根据业务模式新增特征

在之前众多业务模式分析当中，我们已经对SLG游戏的业务以及游戏本身有了很多的了解。现在我们可以基于这些了解来构筑新的特征。在对算法进行建模时，我们经常使用以下方式对现有特征进行重组，来生成新特征:

1.特征之间加减乘除

2.对特征进行简单数学运算(绝对值，取对数)

3.对特征进行逻辑运算(and, or,并集，交集，最大值，最小值)

4.对特征进行聚合运算（按照离散特征聚合后取中位数、均值、众数、标准差、方差、频数)

5.对特征使用多项式拓展方法(即特征与自身、与其他特征相乘)

6.对连续特征进行分箱

7.对离散特征进行独热编码

基于我们对业务的理解，事实上现在的数据集中有非常多可以增加的特征，例如:

·个人PVP胜率=PVP胜利次数/PVP次数，无PVP的用户胜率为0

·主动发起PVP的概率=主动发起PVP次数/PVP次数，无PVP的用户主动发起为0

·玩家发育效率=平均资源获取数/在线时长(该指标高的人擅长游戏)

·玩家升级效率=技能的平均等级/在线时长

·氪金发育效率=平均资源获取数/7日付费金额（该指标高的人善用氪金资源)

·氪金升级效率=技能的平均等级/7日付费金额

还可以对7日付费金额、在线时长等明显对氪金有影响的特征分箱，并按分箱后的类别求资源、PVP胜率等与游戏体验深度相关的特征的聚合函数，这一点对于树模型会尤其有效。

在有限的时间内，我们无法对所有可能增加的特征都进行计算，因此我们选择手动增加以上6个特征，作为一种尝试:

```
# 在原有的特征中分别出哪些是技能，哪些是资源
X.columns.tolist()
```

```
#发育——关于资源的部分
GrowthFeature = []
for i in X.columns:
    if '招募' in i:
        GrowthFeature.append(i)
    elif '获取' in i:
        GrowthFeature.append(i)
```

```
#等级——关于技能的部分
LevelUpFeature = []
for i in X.columns:
    if '建筑' in i:
        LevelUpFeature.append(i)
    elif '科研' in i:
        LevelUpFeature.append(i)
```

```
X['PVP胜率'] = X['PVP胜利次数']/X['PVP次数']
X['主动发起PVP的概率'] = X['主动发起PVP次数']/X['PVP次数']
X['玩家发育效率'] = X.loc[:,GrowthFeature].mean(axis=1)/X['在线时长']
X['氪金发育效率'] = X.loc[:,GrowthFeature].mean(axis=1)/X['付费金额']
X['玩家升级效率'] = X.loc[:,LevelUpFeature].mean(axis=1)/X['在线时长']
X['氪金升级效率'] = X.loc[:,LevelUpFeature].mean(axis=1)/X['付费金额']
```

查看除0错误与极值错误

```
X['PVP胜率'].isnull().sum()
#1712305
```

```
X['氪金发育效率'].describe()
```

```
count    1665563.000
mean             inf
std              nan
min            0.000
25%              inf
50%              nan
75%              inf
max              inf
Name: 氪金发育效率, dtype: float64
```

```
X.loc[:,LevelUpFeature].mean(axis=1).describe()
```

```
count    2288007.000
mean           8.266
std           76.768
min            0.000
25%            0.000
50%            0.754
75%            6.174
max        20363.493
dtype: float64
```

查看分子情况，虽然有些分子的确比较大，但是大部分分子其实都是个位数或者很小

氪金发育效率出现问题的根本还是因为许多氪金数额过小(分母过小)，而不是因为分子过高因此对于出现inf问题的样本，使用0覆盖

```
# 处理0错误，处理极值错误
for newfeature in ['PVP胜率','主动发起PVP的概率','玩家发育效率','氪金发育效率','玩家升级效率','氪金升级效率']:
    X.loc[X[newfeature].isnull(),newfeature] = 0 #将所有的空值变为0
    X.loc[X[newfeature] == float('inf'),newfeature] = 0
```

检查:

```
X['玩家发育效率'].isnull().sum()
#0
```

```
(X['氪金发育效率'] == float('inf')).sum()
#0
```

除此之外，我们在之前分析游戏的经营状况时，已经得出了一些关于用户价值的关键结论:

·分析营收状况时，我们发现如果用户**7日内付费超过6元**，则在7日后继续付费的概率大于不再付费的概率，是高价值用户

·分析用户在线时间时，当用户的**在线时间小于20或者大于900分钟**，用户对氪金金额的贡献率较小，是低价值用户

·分析游戏竞争状况与游戏平衡性时，我们发现，**主动发起PVP概率大于50%**的用户贡献了较多氪金金额，是高价值用户

·在分析资源获取难度、新手友好程度时，我们发现**初始资源（木头）获取数量为0或很低**的玩家是低价值用户

·在分析异常值时，我们发现异常用户覆盖了全部氪金用户，因为氪金用户的资源量往往都很大。然而不氪金的用户也有许多拥有大量资源，他们是肝帝，肝到一定程度就不会氪金，因此**资源量巨大（异常)但7日内没有付费**的用户是低价值用户

基于这些发现，我们统一让低价值用户被标记为0，高价值用户被标记为1，为特征矩阵创造如下新特征:

·高价值玩家

潜力玩家:7日内付费超过5.98元

好战玩家:主动发起PVP概率大于50%

·低价值玩家

佛系玩家:主动发起PVP概率小于20%，或一次也没有参与过PVP

肝帝玩家:7日在线时间过长，或资源量巨大但7日内没有付费

菜鸡玩家:参与过PVP且个人PVP胜率小于10%

流失玩家:7日在线时间过短，或初始资源获取数量小于10000

事实上还有相当多类似的特征我们可以细致地划分，这可能涉及到用户在游戏中的行为画像。例如，PVP胜率很高的"大神玩家"，PVP次数很多但是胜率很低的"又菜又勇"玩家，从来不玩PVP但是PVE胜率很高、资源量高到飞起的"圈地自萌°玩家，这些复杂的情况可能都与最终氪金行为有关。现在让我们根据这些发现，来进行特征衍生:

```
#高价值玩家，符合条件的被标注为1
X['潜力玩家'] = (X['付费金额']>=5.98).astype(int)
X['好战玩家'] = (X['主动发起PVP的概率']>=0.5).astype(int)

#低价值玩家，符合条件的被标注为0
X['肝帝玩家'] = (X['在线时长']>800).apply(lambda x:not x).astype(int)
X['佛系玩家'] = ((X['主动发起PVP的概率'] < 0.2) | (X['PVP次数'] ==
0)).apply(lambda x:not x).astype(int)
X['菜鸡玩家'] = ((X['PVP胜率']<0.1) & (X['PVP次数'] != 0)).apply(lambda x:not
x).astype(int)
X['流失玩家'] = ((X['在线时长']<=15) | X['木头获取数量'] <= 0).apply(lambda x:not
x).astype(int)
```

```
X.isnull().sum().sum()
```

（2）达成建模所需的统计假设

为了数据能够有效在线性回归中运行，我们希望能够避免异常值、偏左分布等因子带来的影响。因此，我们希望对训练集做出以下预处理:

1.相关性分析:我们可以删除与标签相关性不足或为0的特征，并观察哪些特征与标签有关联

2.训练/测试比例调优:调整训练集测试集比例,找出最佳的test_size超参数

3.异常值处理:按照箱线图的规则，对异常值中7伯内无氟金的玩家进行中位数覆盖处理，对于7日内有氯金的玩家不处理

4.数据归一化:为排除量纲不统一问题带来的困扰，同时在大数据量情况下加速计算速度，我们需将所有特征属性归一化到[0,1]范围内

5.1相关性分析:筛选特征/特征重要性

现在模型有190多个特征，为了能够增加计算速度/降低计算成本/做类似于多项式的特征衍生方法，我们可以简单使用相关系数对特征进行筛选。
最简单的方式就是使用皮尔逊相关系数。

DataFrame可以直接用函数.corrl()来计算相关系数，但由于我们有190多个特征，每条特征220w样本，直接使用corr()函数对所有特征进行计算不仅非常占用计算内存，并且计算会非常缓慢，因此在这里我们简化一下:只考虑特征与标签之间的相关性，不考虑特征与特征之间的相关性带来的统计学方面的问题。

在传统统计学中，如果特征与特征之间的线性相关性过高，线性回归是不能够处理的，不过现在sklearn库中的线性回归是使用SVD分解方式在进行计算，可以克服传统线性回归无法处理高线性相关特征的问题，因此我们可以忽略这个问题。

```
corr_list = pd.DataFrame()
for idx,column in enumerate(X.columns):
    corr_ = pd.concat([y,X.loc[:,column]],axis=1).corr().iloc[0,1]
    corr_list.loc[idx,'特征'] = column
    corr_list.loc[idx,'相关系数'] = corr_
```

```
corr_list.sort_values('相关系数',ascending=False).head()
```

	特征	相关系数
104	付费金额	0.735
4	象牙获取数量	0.658
2	石头获取数量	0.648
5	象牙消耗数量	0.641
0	木头获取数量	0.640

```
corr_list[abs(corr_list['相关系数'])<0.01]
```

	特征	相关系数
106	玩家注册时刻	0.001
117	菜鸡玩家	0.006
118	流失玩家	0.001

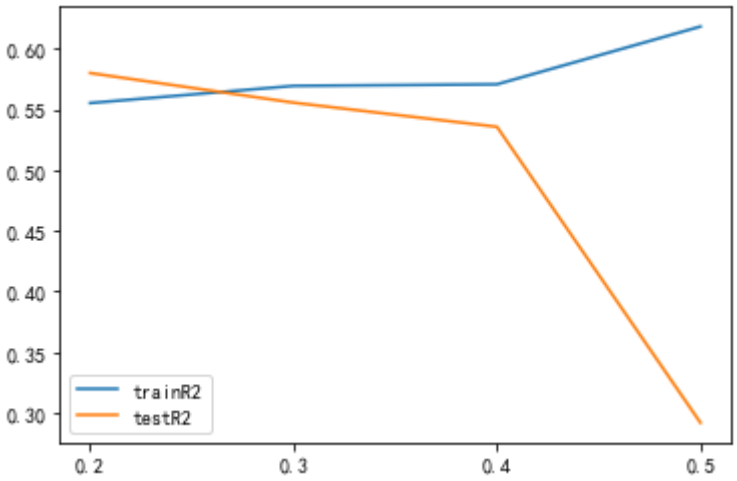
只有两个特征的相关系数低于0.01，我们可以考虑都统一保留这些特征。

5.2训练/测试集分割的调优

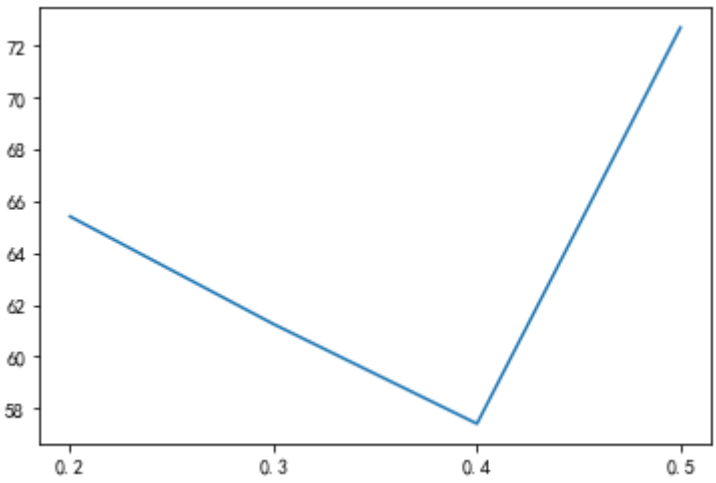
正常在进行数据分割时，我们是按训练集7，测试集3的比例进行分割。但考虑到，我们将使用线性回归进行建模，线性回归是是复杂度较低的模型，学习能力和抗过拟合能力有限，因此训练集过多或者过少都很容易导致过拟合，为此我们稍微可以尝试几组训练集测试集的分割比例，以确定最适合线性回归的训练集数量。

```
trainr2 = [] #训练集上的R2
testr2 = [] #测试集上的R2
testRMSE = [] #测试集上的RMSE
for i in [0.2,0.3,0.4,0.5]:
    Xtrain,Xtest,Ytrain,Ytest = TTS(X,y,test_size = i,random_state=0)
    model = LR().fit(Xtrain,Ytrain)
    trainr2.append(model.score(Xtrain,Ytrain))
    testr2.append(model.score(Xtest,Ytest))
    testRMSE.append(np.sqrt(MSE(Ytest,model.predict(Xtest))))
    print('done')
```

```
plt.plot(trainr2,label = 'trainR2')
plt.plot(testr2,label = 'testR2')
plt.xticks(ticks=[0,1,2,3],labels=[0.2,0.3,0.4,0.5])
plt.legend()
plt.show()
```




```
plt.plot(testRMSE)
plt.xticks(ticks=[0,1,2,3],labels=[0.2,0.3,0.4,0.5])
plt.show()
```




```
testRMSE
#[65.40513888232609, 61.25254728964682, 57.409896451713955,
72.69313409149458]
```

从结果的角度来看，在测试集上RMSE最低的训练集数量是0.4，从R2的表现来看，测试集比例为0.4时，训练集上R2表现出过拟合的倾向，但并不严重。

但考虑到现在模型的核心评估指标是RMSE，因此R2稍有问题，但RMSE明显有优势时，我们优先考虑RMSE上的表现。

```
Xtrain,Xtest,Ytrain,Ytest = TTS(X,y,test_size=0.4,random_state=0)
```

这次不恢复索引，而让索引跟随Xtrain，Xtest分割后的样子：

```
Ytrain = pd.DataFrame(Ytrain)
Ytest = pd.DataFrame(Ytest)

Ytrain.index = Xtrain.index
Ytest.index = Xtest.index
```

5.3导常值处理

在处理异常值时，我们发现所有的氪金玩家都被归类到了异常数据里，这与氪金玩家往往都是深入参与到游戏的玩家有很大的关系。

由于玩家都被包括在异常数据中，我们不能对异常数据进行简单的删除，并且由于数据左偏严重，异常值数量众多(超出100w)，删掉50%的数据也不可取。因此在这里，我们对异常值进行类似于盖帽的特殊处理。

按照箱线图的规则进行异常检测，并对异常值中7日内无氪金的玩家进行中位数覆盖(如果效果不够好，可以试着使用0覆盖)，对于7日内氪金的玩家则不处理，尽量放大氪金玩家与非氪金玩家特征上的区别，为建模提供更好的保障。

注意:异常值处理必须在训练集测试集被分割的情况下进行，因为测试集上的异常值处理是必须依赖对测试集进行异常值处理所产生的中间变量的

```
def AbnormalReplace(Xtrain,Xtest):
    #先对所有特征进行异常检测
    #对7日内付过费的用户，不处理
    #对7日内没付过费的用户，替换所有异常值为当前特征的中位数或者0

    Xtrain_ = Xtrain.copy()
    Xtest_ = Xtest.copy()
    for column in Xtrain_.columns:
        #提取当前要检测的列
        f_train = Xtrain_.loc[:,column]
        f_test = Xtest_.loc[:,column]

        #从训练集中计算QL、QU、IQR、中位数
        QL = np.quantile(f_train,0.25)
        QU = np.quantile(f_train,0.75)
        IQR = QU - QL
        medium_ = f_train.median()

        #对训练集和测试集同时进行检测，得到Ture/False列表
        errortrain = ((f_train < (QL-1.5*IQR)).astype(int) + (f_train > (QU+1.5*IQR)).astype(int)) != 0
        errortest = ((f_test < (QL-1.5*IQR)).astype(int) + (f_test > (QU+1.5*IQR)).astype(int)) != 0

        #将原矩阵中的异常值替换为中位数，主题排除7日付费用户
        #如果效果不够强烈，替换为0
        Xtrain_.loc[((Xtrain_['付费金额']==0).values & errortrain.values),column] = 0#medium_
        Xtest_.loc[((Xtest_['付费金额']==0).values & errortest.values),column] = 0#medium_

    return Xtrain_,Xtest_
```

```
Xtrain['木头获取数量'].describe()
```

```
count       1372804.000
mean         459707.645
std         5253011.374
min               0.000
25%               0.000
50%           41934.000
75%          153061.750
max      1239962311.000
Name: 木头获取数量, dtype: float64
```

```
#为了保留原始的Xtrain和Xtest，新生成的值我们写作小写
xtrain,xtest = AbnormalReplace(Xtrain,Xtest)
```

```
xtrain['木头获取数量'].describe()
```

```
count       1372804.000
mean         253442.438
std         5186103.010
min               0.000
25%               0.000
50%           10000.000
75%           96271.250
max      1239962311.000
Name: 木头获取数量, dtype: float64
```

5.4 归一化处理

加速运算，各个特征量纲统一——使用线性回归的时候就可以查看系数来决定特征的重要性

```
from sklearn.preprocessing import MinMaxScaler
mm = MinMaxScaler(feature_range=[0,1])
mm = mm.fit(xtrain)
```

```
#使用训练集上的最小值和最大值对训练集/测试集同时进行归一化
xtrain = mm.transform(xtrain)
xtest = mm.transform(xtest)
```

```
xtrain.min()
#0.0
```

```
xtrain.max()
#1.0
```

```
xtest.max()
#1.3636363636363638
```

```
xtest.min()
#0.0
```

6.模型融合:处理极度偏态的数据带来的问题

在进行异常值处理时，我们发现所有的氪金用户都被包括在了异常用户中,这说明有大量未氪金的用户表现在游戏资源积累、游戏参与深度上的行为与氪金用户极为相似。同时,氪金用户与不氪金用户的用户组成都相对复杂。

在氪金用户中，有进入游戏1分钟就闭眼氪0.99的羊毛玩家，也有在游戏中7日都未氪、7日后不知为何突然磕了一点点的玩家。在不氪金用户中,有深度参与游戏但一分钱不氪肝帝,又有玩得很糟但一直不退游的玩家。因此在预测前其实就可以想象到，模型在用户辨别、金额预测上的效果恐怕不会太好。

在之前的特征处理中，我们已经对异常值进行过“盖帽"处理，并在特征中增加我们认为可以让氪金和非氪金用户区别变大的特征，但直接让单-模型对氪金用户的氪金数额进行预测是非常困难的。 因此，我们可以使用2个模型进行融合预测，第-个模型是分类模型逻辑回归，我们让逻辑回归先对用户进行“氪与不氪”的预测，然后将逻辑回归认为会氪的用户放入线性回归，再让线性回归直接对“本来就很可能会氪金”的用户进行氪金金额预测。我
们认为这样的方式经过适当调整后,应该能够提升模型表现。

•创造逻辑回归使用的y2

氪金用户转化为1，不氪金用户转化为0

```
y2 = (y!=0).astype(int)
```

```
y2.value_counts()
```

```
0    2242019
1      45988
Name: 45日付费金额, dtype: int64
```

已经有了Xtrain,Xtest,线性回归用的Ytrain,Ytest,逻辑回归的y2，变成可以和Xtrain,Xtest相匹配的y2train,y2test

按分割后的Xtrain和Xtest的索引来分割y2，这样y2就是与Ytrain相匹配的

```
Ytrain2 = y2[Xtrain.index]
Ytest2 = y2[Xtest.index]
```

- **使用逻辑回归进行建模并使用ROC进行评估**

```
from sklearn.linear_model import LogisticRegression as LogiR
from sklearn.metrics import roc_auc_score as ROC
from sklearn.metrics import recall_score as Recall#召回
from sklearn.metrics import precision_score as Precision
```

```
clf = LogiR(random_state=0)
#使用经过一切特征工程处理的xtrain
clf.fit(xtrain,Ytrain2)
clf.score(xtrain,Ytrain2)
#0.9966470086042872
clf.score(xtest,Ytest2)
#0.9966892591042643
```

由于数据高度偏态导致的高分数

```
# 测试集的拟合情况
y2_proba = clf.predict_proba(xtest)
y2_proba[:,1]
#array([0.00552264, 0.00132681, 0.00127126, ..., 0.00137865,
0.00365359,0.00160699])
```

```
ROC(Ytest2,y2_proba[:,1])
#0.9720289456896506
Recall(Ytest2,clf.predict(xtest))
#0.8366891597736178
Precision(Ytest2,clf.predict(xtest))
#0.9981173721111399
```

逻辑回归的效果好，能够在样本量如此不均衡的情况下达到0.97的ROC，证明数据与逻辑回归很适配。

- 调节阈值提升Recall

```
for tol in np.linspace(0,0.5,20):
    pred = (prob >= tol).astype(int)
    recall = Recall(Ytest2,pred)
    print("{:.3f}Recall:{:.3f}".format(tol,recall))
```

```
0.000Recall:1.000
0.026Recall:0.905
0.053Recall:0.878
0.079Recall:0.869
0.105Recall:0.864
0.132Recall:0.858
0.158Recall:0.853
0.184Recall:0.850
0.211Recall:0.848
0.237Recall:0.847
0.263Recall:0.845
0.289Recall:0.844
0.316Recall:0.843
0.342Recall:0.842
0.368Recall:0.841
0.395Recall:0.840
0.421Recall:0.839
0.447Recall:0.838
0.474Recall:0.837
0.500Recall:0.837
```

```
for tol in np.linspace(0,0.026,20):
    pred = (prob >= tol).astype(int)
    recall = Recall(Ytest2,pred)
    print("{:.3f}Recall:{:.3f}".format(tol,recall))
```

```
0.000Recall:1.000
0.001Recall:0.986
0.003Recall:0.956
0.004Recall:0.948
0.005Recall:0.940
0.007Recall:0.935
0.008Recall:0.930
0.010Recall:0.925
0.011Recall:0.923
0.012Recall:0.920
0.014Recall:0.917
0.015Recall:0.916
0.016Recall:0.915
0.018Recall:0.914
0.019Recall:0.912
0.021Recall:0.912
0.022Recall:0.910
0.023Recall:0.908
0.025Recall:0.906
0.026Recall:0.905
```

```
# 经过之后的RMSE判断，0.02是个比较合适的值
tol_=0.02
```

```
# 对于所有不氪金的样本，预测到此结束——y=0
result = pd.DataFrame(y2_proba[:,1],index=Xtest.index)
result.columns = ['logi_proba']
result['logi_y_pred'] = (result['logi_proba']>=tol_).astype(int)
```

```
# 训练集，同样索引需要导入
logi_train_result = pd.DataFrame(clf.predict_proba(xtrain)
[:,1],index=Xtrain.index)
logi_train_result.columns = ['logi_train_proba']
logi_train_result['logi_y_pred_train'] =
(logi_train_result['logi_train_proba']>=tol_).astype(int)
```

```
# 选出应该放在线性回归中的训练集和测试集
# 训练集
xtrain_linear_reg = xtrain[logi_train_result['logi_y_pred_train'] == 1]
ytrain_linear_reg = Ytrain[logi_train_result['logi_y_pred_train'] == 1]
# 测试集
xtest_linear_reg = xtest[result['logi_y_pred'] == 1]
ytest_linear_reg = Ytest[result['logi_y_pred'] == 1]
```

```
# 开始回归预测
def reg_predict(model):
    reg = model.fit(xtrain_linear_reg,ytrain_linear_reg)
    y_linear_pred = reg.predict(xtest_linear_reg)#对测试集进行预测
    print('训练集R2:
{:.3f}'.format(reg.score(xtrain_linear_reg,ytrain_linear_reg)))
    print('测试集R2:
{:.3f}'.format(reg.score(xtest_linear_reg,ytest_linear_reg)))
    print('测试集RMSE:
{:.3f}'.format(np.sqrt(MSE(ytest_linear_reg,y_linear_pred))))
    return y_linear_pred #融合模型测试出结果中的回归部分
```

```
#线性回归
reg0 = LR()
y_linear_pred0 = reg_predict(reg0)
#训练集R2:0.582
#测试集R2:0.537
#测试集RMSE:288.430
```

```
from sklearn.ensemble import RandomForestRegressor as RFR
from sklearn.ensemble import GradientBoostingRegressor as GBR
reg1 = RFR(n_estimators=100,max_depth=2,max_features=20,random_state=1412)
reg2 = GBR(n_estimators=100,max_depth=2,max_features=20,random_state=1412)
```

```
y_linear_pred1 = reg_predict(reg1)
#训练集R2:0.515
#测试集R2:0.431
#测试集RMSE:319.896

y_linear_pred2 = reg_predict(reg2)
#训练集R2:0.737
#测试集R2:0.531
#测试集RMSE:290.533
```

不是很严格的限制，随机森林不太适合现在的数据
GBR和逻辑回归差不多，因为逻辑回归参数调节性小，因此选择GBR（树模型过拟合，通过参数调整就可解决）

模型调优