

# 目录

LOL 游戏玩家类型与胜利条件大数据分析 .....	3
1 实验要求 .....	3
2 实验背景 .....	3
3 实验目的 .....	4
3.1 实验主题 .....	4
◆ 英雄联盟资深玩家用户聚类 .....	4
◆ 英雄联盟游戏的重要获胜条件 .....	4
3.2 实验目的 .....	4
4 实验环境 .....	4
5 实验内容 .....	5
5.1 环境搭建 .....	5
5.2 数据获取 .....	5
5.3 数据描述 .....	8
5.4 探索性分析 .....	10
5.4.1 缺失值分析 .....	10
5.4.2 异常值分析 .....	11
5.4.3 重复值分析 .....	16
5.4.4 数据分布分析 .....	17
5.5 数据预处理 .....	20
5.5.1 数据去重 .....	20
5.5.2 数据空值处理 .....	21
5.6 数据特征提取 .....	23
5.6.1 数值特征 .....	23
5.6.2 类别特征 .....	23
5.6.3 规范化特征 .....	24
5.7 建立模型 .....	25
5.7.1 聚类模型 .....	25
5.7.2 逻辑回归模型 .....	27
5.8 结果分析 .....	30
6 实验总结 .....	32
6.1 数据挖掘流程分析 .....	32
6.2 实验心得 .....	33
6.2.1 实验总结 .....	33
6.2.2 模型改进 .....	34
6.3 实验问题记录 .....	34
7 实验拓展：动态数据处理 .....	35
7.1 环境搭建 .....	35
7.2 实验内容 .....	36
7.2.1 基于 java+flink 的数据批处理 .....	36
7.2.2 基于 scala+spark 的数据流处理 .....	38
8 附件说明 .....	38

# LOL 游戏玩家类型与胜利条件大数据分析

## 1 实验要求

- (1) 以掌握大数据分析技术路线为目标，课程设计可依托百分点大数据网络实验平台或文波楼实验室大数据实验平台完成，自拟具体应用场景，如社会时事热点分析、电商平台 XXX 类产品大数据分析等，尽量体现课程讲授相关技术内容。
- (2) 围绕拟定题目能够获得有效数据集，并能对数据进行预处理。
- (3) 围绕拟定目标能够给出明确数据分析模型和方法。
- (4) 能够采用相关技术或平台实现数据分析。
- (5) 分析结果准确，采用数据可视化方法进行呈现。
- (6) 报告格式设计合理，技术描述清晰，结果准确。
- (7) 代码可直接写在报告中，也可以作为附件提交。
- (8) 提交时间：17 周周末。

## 2 实验背景

《LOL》是一款类似于刀塔、魔兽世界的免费的多人在线战斗推塔游戏，由 Riot Games 公司开发，在 2009 年底正式发行，至今已有 13 年，且在国内已经推出手游版本。

在普通匹配模式的游戏竞技中，游戏分为两个队伍各五名玩家，游戏的目标是摧毁对方的水晶枢纽，当其中一队的水晶枢纽被摧毁时，比赛结束。在游戏过程中，一个团队可以执行很多操作，如：

1. 获得团队增益：摧毁沿途炮塔，击杀野区大型野怪；
2. 获得个人增益：获得第一滴血，击杀对方英雄，击杀野区小型野怪；清除对方兵线等。

作为该游戏的 7 年资深玩家，希望通过相关数据分析，深入了解顶尖玩家游戏胜利的重要因素；站在未来游戏开发企业的角度，明确资深玩家的用户分类，研究他们的偏好特点，从而针对性进行营销和道具资源服务，达到企业最大利益。

### 3 实验目的

#### 3.1 实验主题

以五个地区最新联赛比赛的前 100 名玩家数据为样本，爬取到的 2022 年 12 月 19 日最新联赛比赛数据为对象，研究以下内容：

##### ◆ 英雄联盟资深玩家用户聚类

基于爬取并清洗后的数据集，提取可量化的特征数据，对这前 100 名玩家进行分类。

##### ◆ 英雄联盟游戏的重要获胜条件

以玩家每场游戏相关的各项指标（如野怪信息、人头数、死亡次数等）为参考，研究游戏胜利的重要条件。

#### 3.2 实验目的

理解分布式文件系统、非关系型数据库、MapReduce 实现原理、Spark 大数据分析基础、流式数据分析、数据获取等内容，掌握 pysparksql、pysparkcore、rdd 的基本使用方法，完成从数据爬取、数据预处理、数据提取、数据建模、数据分析、数据可视化流程。

### 4 实验环境

✧ jupyter+MiniConda+spark-3.3.1+hadoop3本地模式（python3.9）；

✓ 可视化：seaborn；pyplot；missingno；

✓ 数据获取：Rriotwatcher；

✓ 数据处理：pyspark.sql；pyspark.ml；sklearn；pandas；numpy。

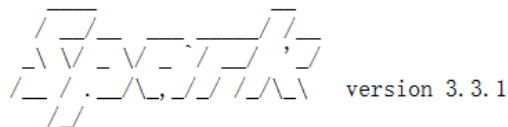
## 5 实验内容

### 5.1 环境搭建

#### Spark 环境部署情况

```
import os
import sys
spark_name = os.environ.get('SPARK_HOME', None)
if not spark_name:
    raise ValueError('spark环境没有配置好')
sys.path.insert(0, os.path.join(spark_name, 'python'))
sys.path.insert(0, os.path.join(spark_name, 'python/lib/py4j-0.10.7-src.zip'))
exec(open(os.path.join(spark_name, 'python/pyspark/shell.py')).read())
```

Welcome to



Using Python version 3.9.12 (main, Apr 4 2022 05:22:27)  
Spark context Web UI available at <http://jelly:4040>  
Spark context available as 'sc' (master = local[\*], app id = local-1671507475024).  
SparkSession available as 'spark'.

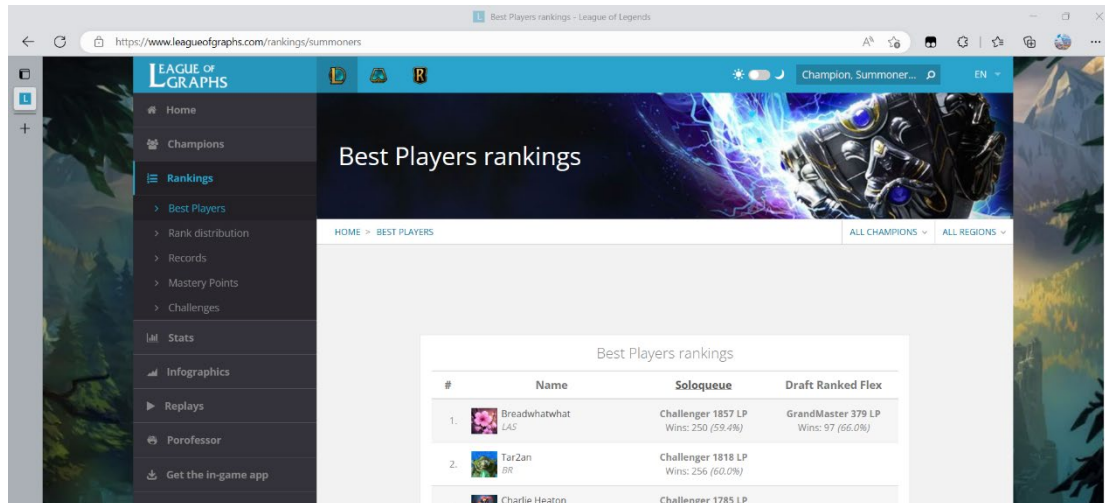
```
import findspark
findspark.init()
import pyspark
print("successful")
```

successful

### 5.2 数据获取

#### 爬取网页

URL: <https://www.leagueofgraphs.com/rankings/summoners/{},>



from requests\_html import HTMLSession:导入库.

session=HTMLSession():创建 session 对象.

response=session.get(url):GET 请求访问指定的 url.

response.html.full\_text:获取 response 的全文本.

通过调用 Rriotwatcher 包获取五个地区最新联赛比赛的前 100 名玩家的数据。

from riotwatcher import LolWatcher

lol\_watcher = LolWatcher('app-api-here')

定义获取信息函数：

```
def pull_summoner_names(url, start_tag, end_tag):
    website_string = session.get(url).html.html

    start_indices = [m.start() for m in re.finditer(start_tag, website_string)]
    end_indices = [m.start() for m in re.finditer(end_tag, website_string)]

    if (len(start_indices) != len(end_indices)):
        return('List lengths do not match.')

    result = [website_string[(start_indices[i]+len(start_tag)):(end_indices[i])] for i in range(0,100)]
    return(result)
```

定义参数，调用函数：

```
url = 'https://www.leagueofgraphs.com/rankings/summoners/{}'
regions = ['na', 'br', 'kr', 'euw', 'eune']
start_tag = '<span class="name">'
end_tag = '</span>\n' | '<br/>\n'

top_summoners = dict()

for region in regions:
    top_summoners[region] = pull_summoner_names(url.format(region), start_tag, end_tag)
```

爬取结果：

	na	br	kr	euw	eune
0	C9 Zven	xiaolongbao	DWG ShowMaker	Agurin	Litny
1	Palafox	Aryze	T1 Burdol	Legendary Manaty	UnExpecteDGanGS
2	qpalzmwoiaj	COME BACK HOME	mars8	RGE Inspired	Kits&ugrave;ne
3	Ssumdayday	v f N b	개종벌레a	Elyoyaaa	Scarlet Rose
4	C9 ZVENNN	VK Hidan	owoowoowoowoowo	SUP Armutke	EXSEMI
...	...	...	...	...	...
95	Cazamareas	Scavage	ZED99	Juhozkin	Atmo
96	Plux	RNG m1ng	xiaozhabi	RoseLMonster	I Doom Warrior I
97	Mentally strong	fvckk	군밤갓	MRS JaVa&aacute;a	Liam Nees&omicron;n
98	DrewDozer	esA	Keine1	RGE3	MkdPro123
99	FudoMyoo	Spl4sh	HLE Mireu	llamame papi	Zalech23

以召唤师名称为索引为每个玩家生成最近的比赛列表。

```
def account_id_for_col(region, column):
    temp_list = []

    for summoner in column:
        try:
            temp_list.append(lol_watcher.summoner.by_name(region, summoner)['accountId'])
        except:
            print('Error for {}'.format(summoner))
            temp_list.append(np.nan)

    return(temp_list)

for column in account_id_columns:
    current_column = summoners[column]
    current_region = column.split('_')[0]
    print('Starting {}'.format(current_region))
    for index in range(0,100):
        if type(summoners[column].iloc[index]) == float:
            print('Skipping index {}, column {} because nan value.'.format(index, column))
            continue
        try:
            temp_game_ids = [game['gameId'] for game in lol_watcher.match.matchlist_by_account(current_region,
            except:
                print('Skipping games at index {}, column {}'.format(index, column))
                continue
            for gameid in temp_game_ids[:10]:
                try:
                    temp_df = pd.DataFrame(lol_watcher.match.by_id(current_region, gameid)['teams'])
                    temp_df['region'] = [current_region] * len(temp_df)
                    match_df = pd.concat([match_df, temp_df], sort = False)
                except:
                    print('Skipping game at index {}, column {}'.format(index, column))
                    continue
```

调整后得到的列表形式：

	teamId	win	firstBlood	firstTower	firstInhibitor	firstBaron	firstDragon	firstRiftHerald	towerKills	inhibitorKills	baronKills
0	100	Fail	False	False	False	False	False	False	0	0	0
1	200	Win	True	True	False	False	True	False	1	0	0
0	100	Fail	False	False	False	False	False	False	0	0	0
1	200	Win	True	True	True	False	True	True	8	1	0

进一步将 bool 类转化为整型数据，初步删除不必要的列：  
`atch_df['win'] = match_df['win'].map({'Win': 1, 'Fail': 0})`  
`match_df['firstBlood'] = match_df['firstBlood'].astype(int)` #其他列类似  
 得到初步 matches.csv 文件，见附件 matches.csv。

### 5.3 数据描述

接下来对爬取到的数据表正式进行数据挖掘操作，首先是数据描述。  
 数据列字段含义解释：

字段名	含义
<b>win</b>	胜利 1: true/0: false（下同）
<b>firstBlood</b>	第一滴血
<b>firstTower</b>	第一座塔
<b>firstInhibitor</b>	第一个水晶
<b>firstBaron</b>	第一个纳什男爵
<b>firstDragon</b>	第一条小龙
<b>firstRiftHerald</b>	第一个峡谷先锋
<b>towerKills</b>	推塔数 n: 数量（下同）
<b>inhibitorKills</b>	水晶摧毁数
<b>baronKills</b>	纳什男爵击杀数
<b>dragonKills</b>	龙击杀数
<b>riftHeraldKills</b>	峡谷先锋击杀数
<b>region</b>	地区

创建 PysparkSession 入口对象

```
from pyspark.sql import SparkSession
# 构建SparkSession执行环境入口对象
spark = SparkSession.builder. \
    appName('bigdata preana'). \
    master('local'). \
    getOrCreate()
sc = spark.sparkContext
```

导入数据，共 9226 行

```
dfread = spark.read.format('csv'). \
    option('sep', ','). \
    option('header', True). \
    option('encoding', 'GBK'). \
    load('D:\pycharm\pysparkProject_work\bigdata_base\data\matches.csv',
        inferSchema=True)
print('Total Records = {}'.format(dfread.count()))
dfread = dfread.selectExpr("_c0 as index", "*").drop("_c0")
print(type(dfread))
dfread.show(10)
```

```
Total Records = 9226
<class 'pyspark.sql.dataframe.DataFrame'>
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
|index|win|firstBlood|firstTower|firstInhibitor|firstBaron|firstDragon|firstRiftHerald|towerKills|inhibitorKi|
lls|baronKills|dragonKills|rifHeraldKills|region|
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
|0|0|0|0|0|0|0|0|0|0|0|0|0|0|
0|0|0|0|0|0|0|0|0|0|0|0|0|0|
```

原数据表描述， `dfread.drop('index').describe().show()`

```
dfread.drop('index').describe().show()
+-----+-----+-----+-----+-----+-----+-----+-----+
|summary|win|firstBlood|firstTower|firstInhibitor|firstBaron|firstDragon|firstRiftHerald|towerKills|
firstDragon|firstRiftHerald|towerKills|inhibitorKills|baronKills|dragonKills|
rifHeraldKills|region|
+-----+-----+-----+-----+-----+-----+-----+-----+-----+
|count|9226|9226|9226|9226|9226|9226|9226|9226|
9226|9226|9226|9226|9226|9226|9226|9226|
|mean|0.5|0.49956644266204203|0.496206373292868|0.3881422068068502|0.2772599176241058|0.43962714068935616|0.4393019726858877|4.611207457186213|0.6845870366355951|0.3639713852156948|1.554194667244743|0.7000867114675916|null|
|stddev|0.5000270995366178|0.500026911554429|0.5000127069458769|0.48735364781707974|0.44767016636756063|0.49636863621756366|0.4963289739421095|3.4131008741494084|0.957760954646272|0.5835861555207791|1.3406062886946142|0.7380586804492858|null|
|min|0|0|0|0|0|0|0|0|0|0|0|0|0|
0|0|0|0|0|0|0|0|0|0|0|0|0|
0|brl|1|1|1|1|1|1|1|1|1|1|1|
1|max|1|11|7|3|5|1|
2|nal|
```

Jupyter 对 dataframe 排版比较乱，为了 dataframe 结果的直观表现，部分代码改成 pycharm 编译运行后的结果展示（同上），本文其他部分均适用。

```
+-----+-----+-----+-----+-----+-----+-----+-----+
|summary|win|firstBlood|firstTower|firstInhibitor|firstBaron|firstDragon|firstRiftHerald|towerKills| | | | | |
|count|9226|9226|9226|9226|9226|9226|9226|9226|
|mean|0.5|0.49956644266204203|0.496206373292868|0.3881422068068502|0.2772599176241058|0.43962714068935616|0.4393019726858877|4.611207457186213|0.6845870366355951|0.3639713852156948|1.554194667244743|0.7000867114675916|null|
|stddev|0.5000270995366178|0.500026911554429|0.5000127069458769|0.48735364781707974|0.44767016636756063|0.49636863621756366|0.4963289739421095|3.4131008741494084|0.957760954646272|0.5835861555207791|1.3406062886946142|0.7380586804492858|null|
|min|0|0|0|0|0|0|0|0|0|0|0|0|0|
|max|1|11|7|3|5|1|11|
```

图表 1 pycharm 中数据表描述运行结果

重点关注数值表示的后属性列和类别列：



```
dfread.drop('index', 'win', 'firstBlood', 'firstTower',
            'firstInhibitor', 'firstBaron', 'firstDragon',
            'firstRiftHerald').describe().show()
```

summary	towerKills	inhibitorKills	baronKills	dragonKills	riftHeraldKills	region
count	9226	9226	9226	9226	9226	9226
mean	4.611207457186213	0.6845870366355951	0.3639713852156948	1.554194667244743	0.7000867114675916	null
stddev	3.4131008741494084	0.957760954646272	0.5835861555207791	1.3406062886946142	0.7380586804492858	null
min	0	0	0	0	0	br1
max	11	7	3	5	2	na1

## 5.4 探索性分析

探索性研究一般是在研究专题的内容与性质不太明确时，为了了解问题的性质，确定调研的方向与范围而进行的搜集初步资料的数据探索，这样可以了解情况，发现问题，从而得到关于资深玩家的某些描述性信息，以供进一步调查研究。

常见的数据有信息不规整、数据点缺失和异常值问题，下面对本文数据表分别进行重复值、数据点缺失和异常值分析。

### 5.4.1 缺失值分析

缺失率统计

```
import pyspark.sql.functions as F
```

```
dfread.agg([
```

```
    *[(1 - (F.count(c)/F.count('*'))).alias(c + 'missing') for c in
```

```
dfread.columns]
```

```
).show()
```

```
import pyspark.sql.functions as F
dfread.agg(
    *[(1 - (F.count(c)/F.count('*'))).alias(c + 'missing') for c in dfread.columns]
).show()
```

indexmissing	winmissing	firstBloodmissing	firstTowermissing	firstInhibitormissing	firstBaronmissing	firstDragonmissing	firstRiftHeraldmissing	towerKillsmissing	inhibitorKillsmissing	baronKillsmissing	dragonKillsmissing	riftHeraldKillsmissing	regionmissing
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

可以看出从官方网站上爬取的数据没有空值出现，这里熟悉一下空值查找，进行每行空值数据的 rdd 查找：

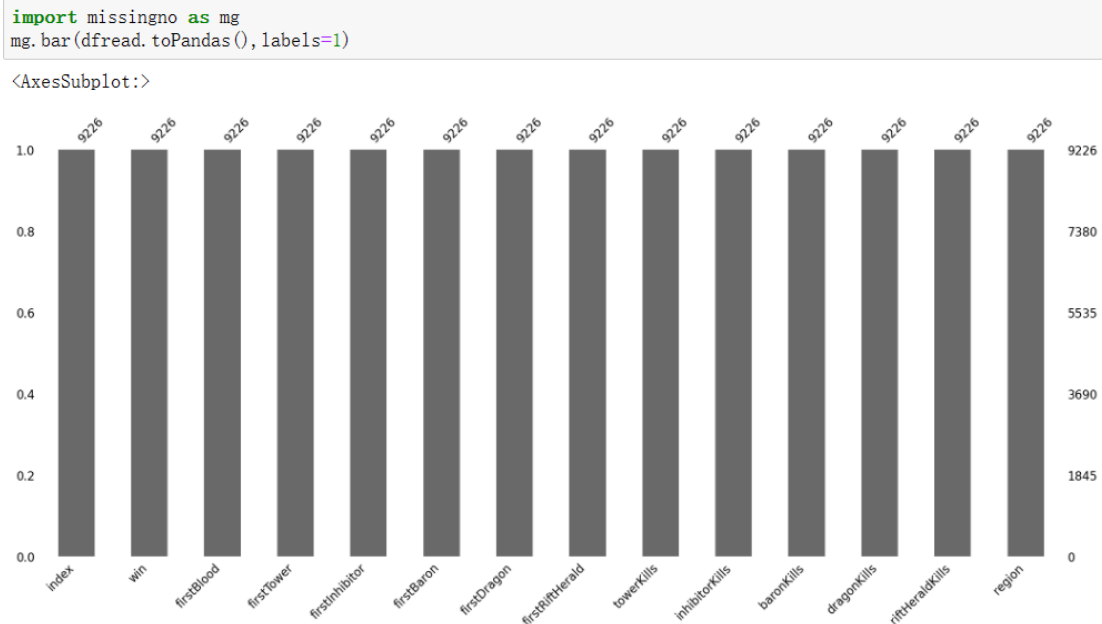
```
dfread.rdd.map(lambda row: (row[0], sum([c == None for c in
```

```
row]])).collect()
```

由可视化数据可知每列均无空值，可以看到官方网站对玩家统计的数据比较完整，空值率可视化如下：

```
import missingno as mg
```

```
mg.bar(dfread.toPandas(),labels=1)
```



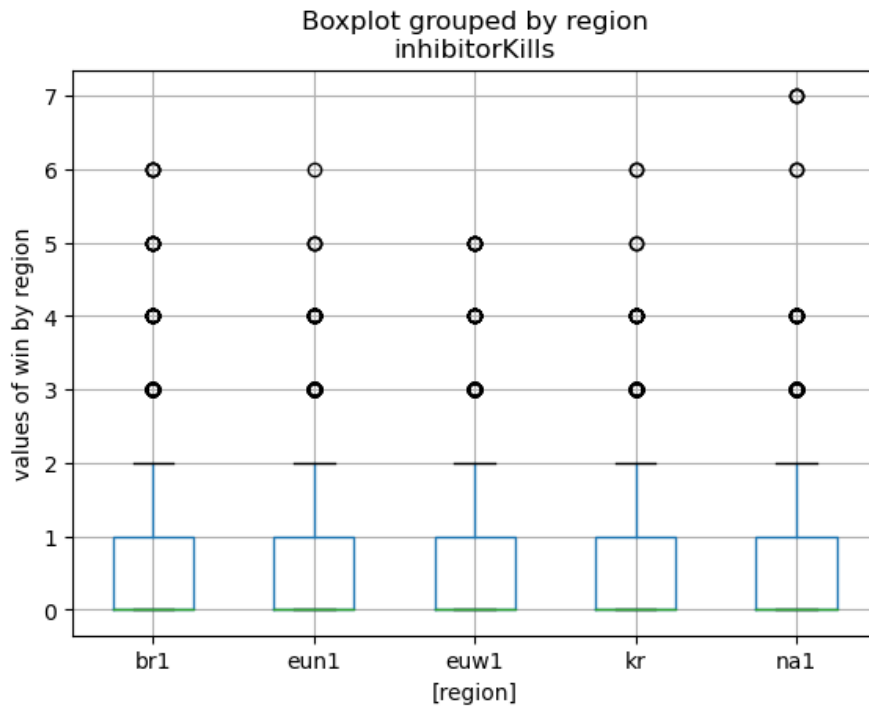
#### 5.4.2 异常值分析

根据前面爬虫和初步数据处理的内容，可以知道前七行数据为 `bool` 型转化而来，`region` 已经进行类别划分，因此这里只对后面的玩家增幅获得数量进行异常值分析：

首先更改列数据类型为整型数据，便于后续分析



```
dfread.toPandas().boxplot(column='inhibitorKills',by=['region'],ax=axes)
# column 参数表示要绘制成箱形图的数据，可以是一列或多列
# by 参数表示分组依据
axes.set_ylabel('values of win by region')
```



根据得到的箱型图，离群点有 3，4，5，6，7，数据比预期的要分散。  
将每一条游戏数据按照 'inhibitorKills' 属性降序排列得到下表：

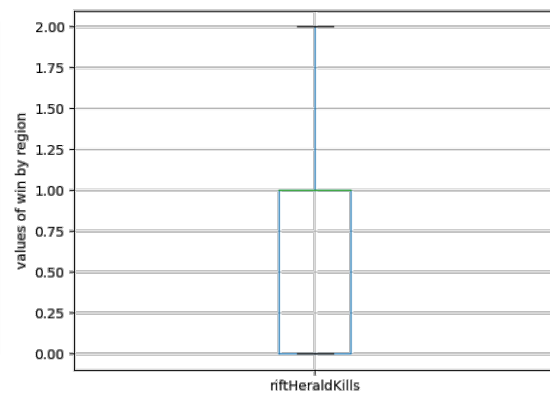
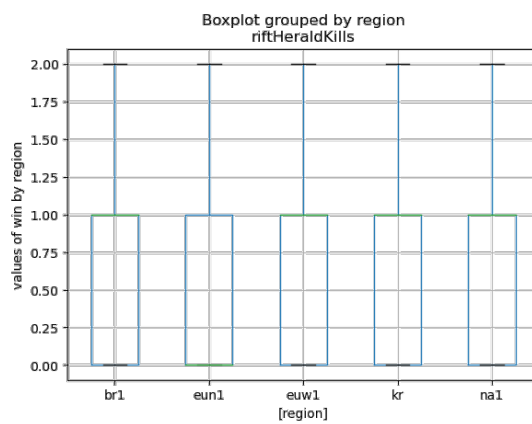
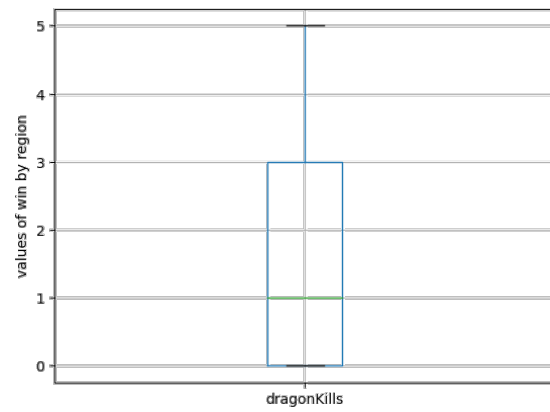
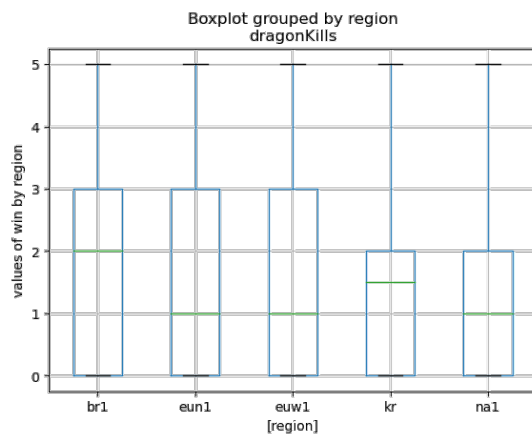
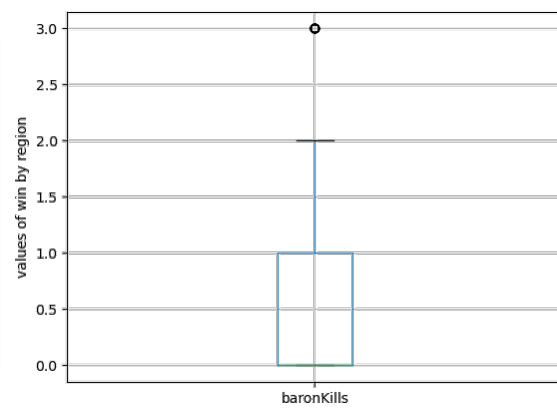
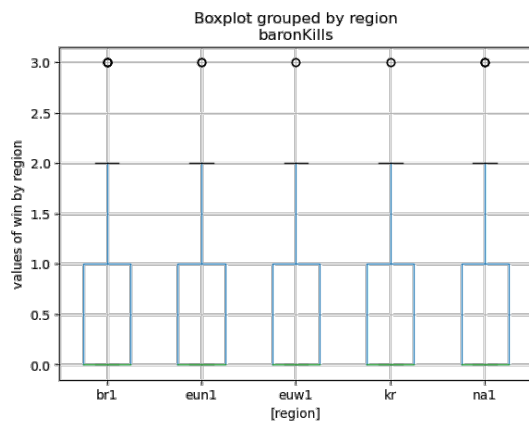
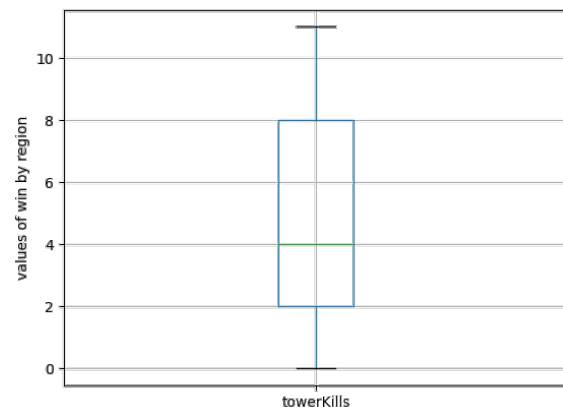
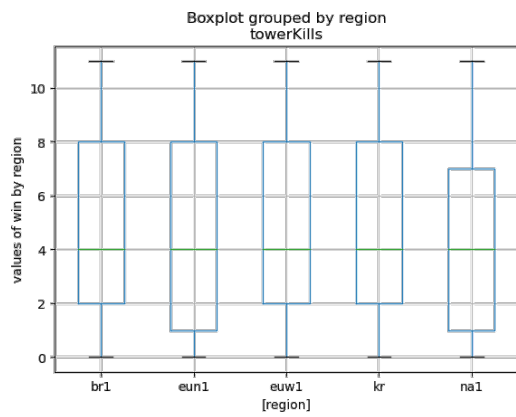
```
dfread.orderBy(
    ['inhibitorKills'],ascending=0).show()
```

```
dfread.orderBy(
    ['inhibitorKills'],ascending=0).show()
```

	index	win	firstBlood	firstTower	firstInhibitor	firstBaron	firstDragon	firstRiftHerald	towerKills	inhibitorKills	baronKills	riftHeraldKills	region
1	94	1	1	1	1	0	0	1	11	7	1		
2	nal												
1	1560	1	1	1	1	0	0	1	11	7	1		
2	nal												
1	498	1	1	1	1	0	0	1	11	7	1		
2	nal												
1	2495	1	1	1	1	1	1	1	11	6	2		
1	br1												
1	1212	1	0	1	1	0	0	0	10	6	1		
0	nal												
1	7552	1	0	1	1	0	1	0	11	6	0		

可以看到离群点 3，4，5，6，7 数量很多，且根据玩家游戏实战经验，每一局游戏时长在 20-60 分钟之间，存在水晶被攻破又恢复的情况，表中水晶攻破数量最大值 7 符合实际游戏情况，因此不是异常值，可以保留。

用同样的方法分别对剩下的四个属性 `towerKills` 、 `baronKills`、`dragonKills`、`riftHeraldKills` 进行分析：



根据箱型图 towerKills、dragonKills、riftHeraldKills 属性没有异常离群点，但 baronKills 属性存在离群点 3。baronKills 为纳什男爵击杀数，纳什男爵为游戏场景中的上路野怪，游戏开局时间 20 分钟后出现。被击杀后起 7 分钟后刷新一次，根据总游戏时长 20-60 分钟的预估，每个团队的玩家每局击杀 3 个纳什男爵是合理的，因此也不是异常点。

这里也按顺序查找一下击杀纳什男爵数最多的游戏局数量：

```
dfread.registerTempTable("dfread_tmp")#生成临时表
sql_tmp = "select count(*) from dfread_tmp WHERE dfread_tmp.baronKills=3"
baronkill_3_count = spark.sql(sql_tmp)

dfread.registerTempTable("dfread_tmp")#生成临时表
sql_tmp = "select count(*) from dfread_tmp WHERE dfread_tmp.baronKills=3"
baronkill_3_count = spark.sql(sql_tmp)
baronkill_3_count.show()
```

count(1)
19

根据 sparksql 的查找结果，击杀纳什男爵数的游戏局有 19 场，显然不是个别离群点，3 不是异常值。

### 5.4.3 重复值分析

具有相同游戏信息的数据可以看作为同一类型玩家，且对本实验中的聚类无参考价值，因此在这里进行数据行重复分析，查看重复数据数量：

```
#检查重复数据
print('Count of rows: {0}'.format(dfread.count()))
print('Count of distinct rows: {0}'.format(dfread.distinct().count()))

#检查重复数据
print('Count of rows: {0}'.format(dfread.count()))
print('Count of distinct rows: {0}'.format(dfread.distinct().count()))

Count of rows: 9226
Count of distinct rows: 9226
```

可以看到不存在完全一样的两行数据，数据表去重前后数据量不变，原因可能是数据来自官网，完整性很高。

#### 5.4.4 数据分布分析

探索性分析中的常用函数，可以很快帮我们理解变量对之间的关系，根据地区对召唤师进行划分。

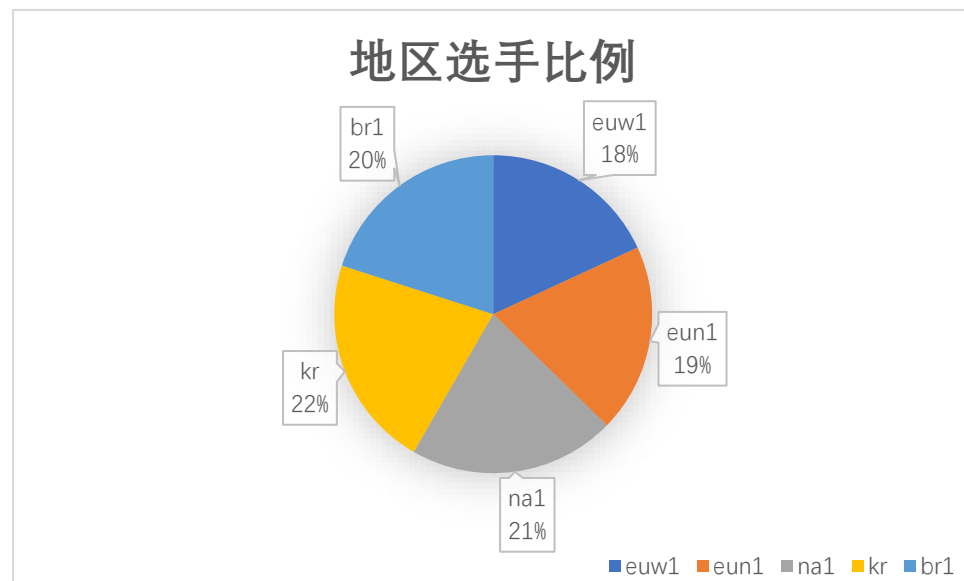
首先统计各个地区召唤师比例

```
#分组统计#地区选手比例
print(type(dfread['region']))
partition = dfread.groupby('region').count().show()
```

```
<class 'pyspark.sql.column.Column'>
```

region	count
euw1	1672
eun1	1772
na1	1938
kr	1994
br1	1850

得到饼状图：



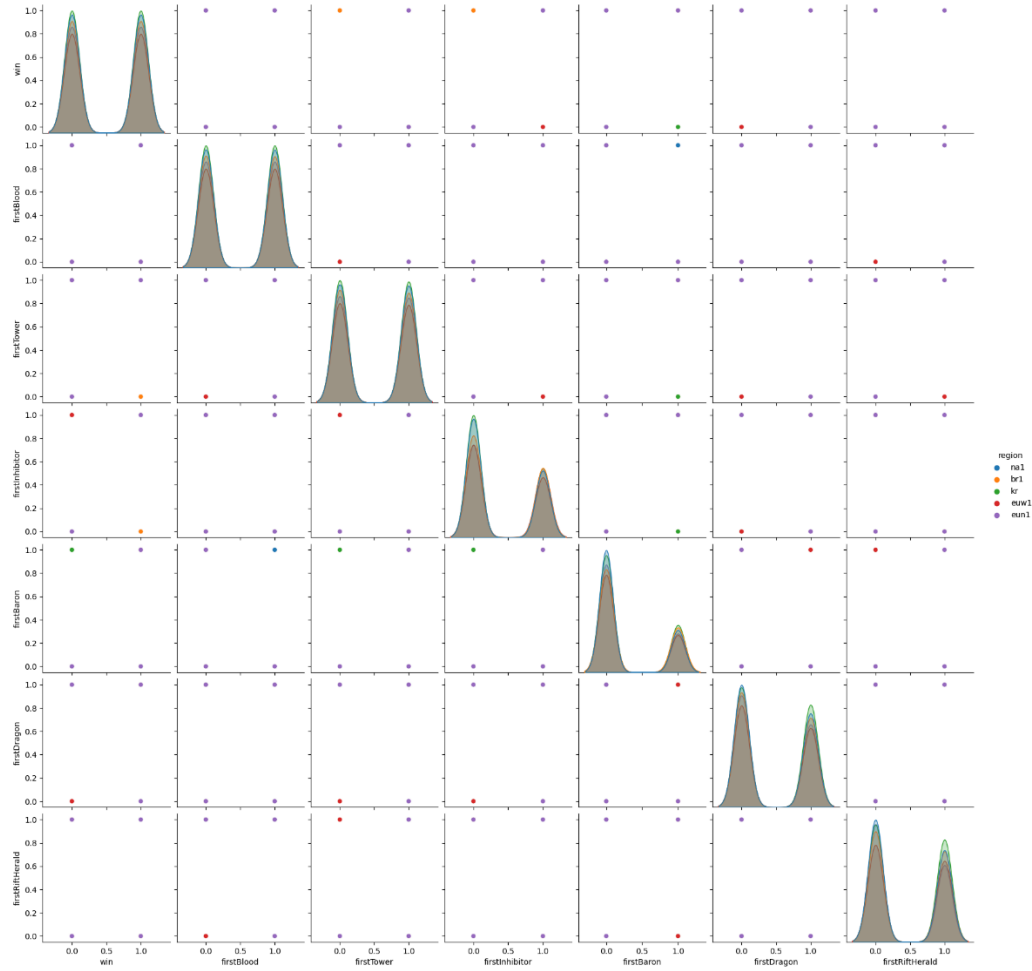
可以看到五个地区在数据集中玩家人数分布较均匀，基本都在 20%的比例，因此该数据集真实有代表性，适合分析玩家类型和游戏胜利因素。

接着分别对前 7 和后 5 列数据进行 pairlot 图分析，掌握数据整体情况

对前 7 列 pairlot 绘图：

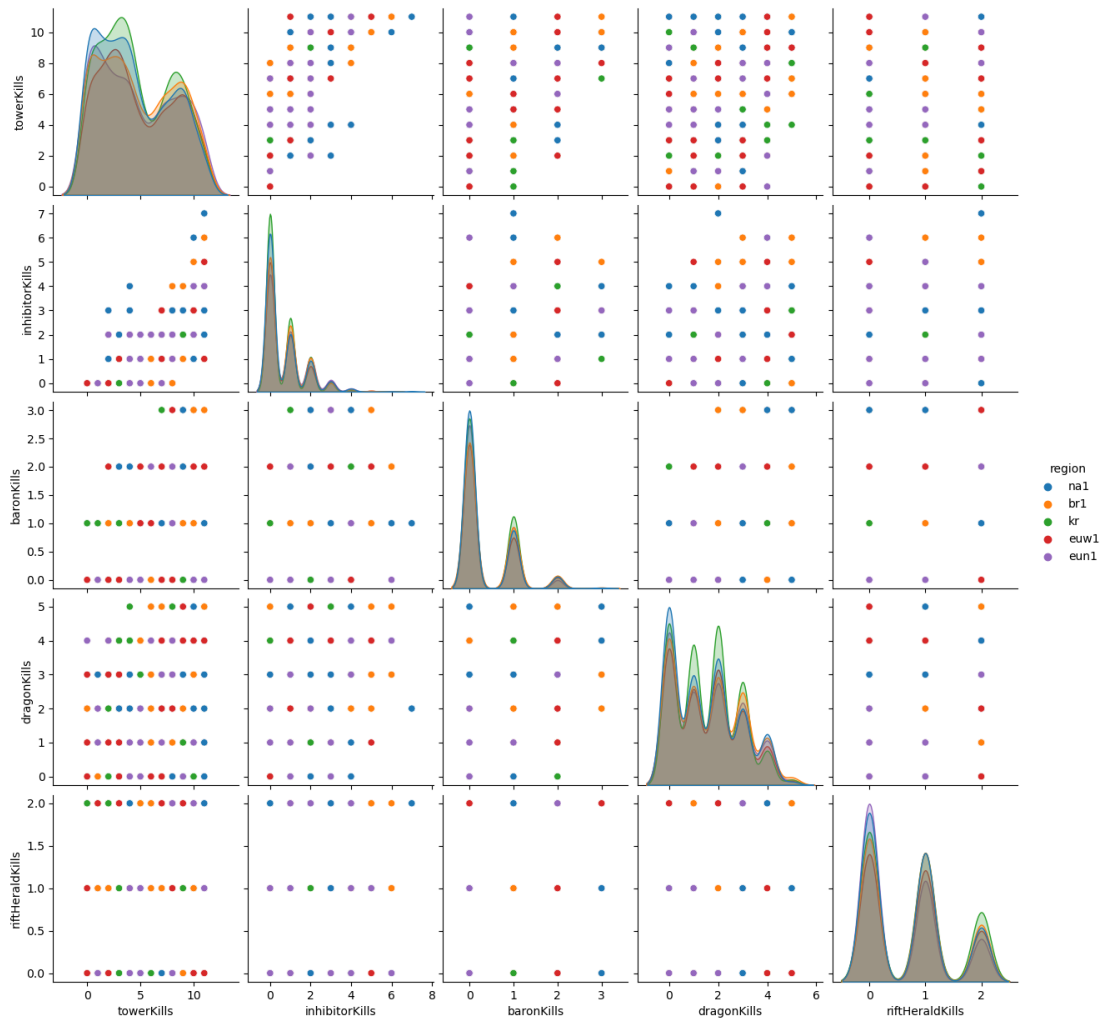
```
dfseries = dfread.select('towerKills','inhibitorKills',
                        'baronKills','dragonKills','riftHeraldKills','region')
sns.pairplot(data = dfseries.toPandas(),height = 2.5,hue = 'region')
```





可以看到前 7 列的值 0/1，其中 `firstBaron` 属性即击杀第一个纳什男爵在本数据集样本中最少，`firstInhibitors` 属性即摧毁第一个水晶在本数据集中相对较少。

对后 5 列 pairlot 绘图：



根据得到的 pairlot 图。显然水晶摧毁数、纳什男爵击杀数、龙击杀数、峡谷先锋击杀数对应属性 `inhibitorKills`、`baronKills`、`dragonKills`、`riftHeraldKills` 数量多的游戏局数整体呈下降趋势，初步可以认为和玩家实力有关，即实力越强的玩家单局游戏中能够个人击杀的目标比普通玩家更多，且数量只是很小一部分。

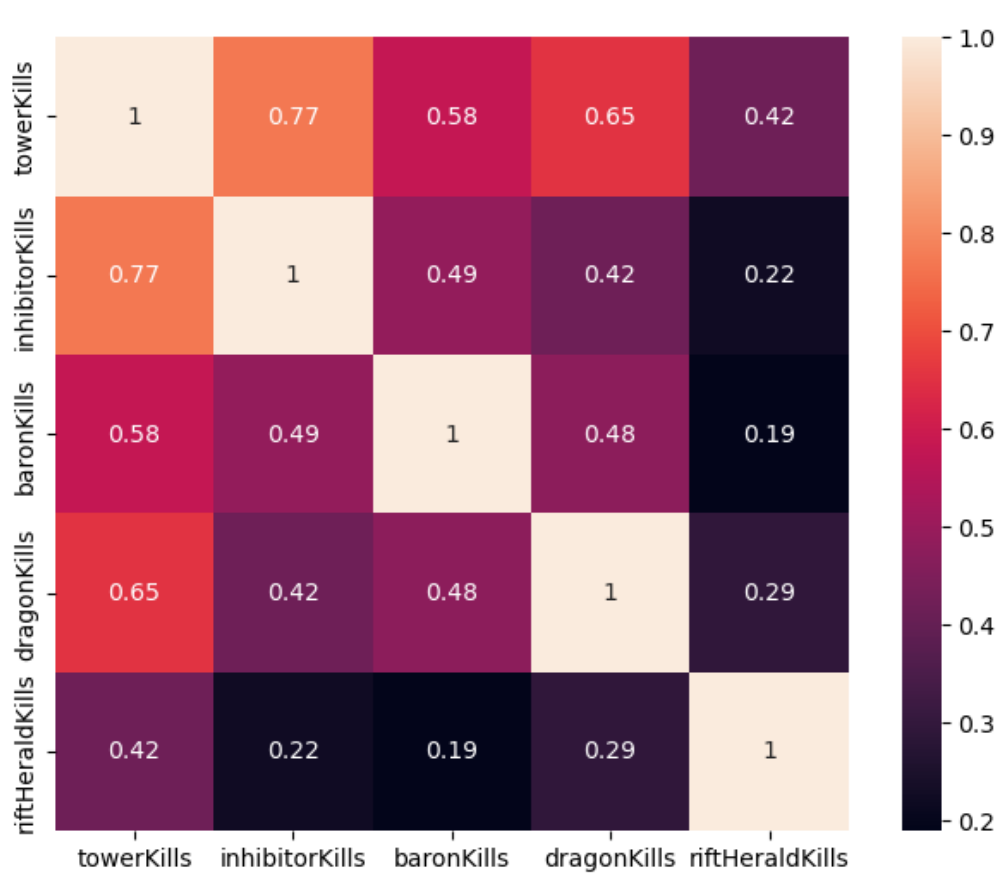
上述整体 pairlot 图数据无论是从不同地区的角度还是整体角度相似程度都很大，每个地区和总体的击杀数量、击杀分布都有很大重合度，没有极端值和异常属性。结合上面得到的箱型图和地区人数比例结果，可以认为来自不同地区的玩家没有特征上的差异，因此接下来的分析不需要单独对不同地区玩家进行区域划分。

而这五个属性在图中表现非常离散，暂时不能看出相关性。下面通过热力图探究这五个属性的相关性：

```
column = dfseries.toPandas().columns.tolist() # 列表头
corr = plt.subplots(figsize = (8,6))
```

```
corr=
sns.heatmap(dfseries.toPandas()[column].corr(),annot=True,square=True)
e)
```

图为基于皮尔逊相关系数计算的相关系数热力图结果：



可以看到这五个属性相关性程度都不大，最大的皮尔逊相关系数仅仅为 0.77 ( $<0.8$ )，可以认为没有较强的相关性，如果作为自变量进行回归分析可以认为因子共线性不高。

## 5.5 数据预处理

通过数据的探索分析发现数据中没有空值、异常值记录。且没有离群点数据。原因可能是在爬取网页数据时已经进行了正则表达式过滤，但是为了更好地实现数据挖掘的基本思想，本节还是进行一下数据去重、数据空值处理操作。

### 5.5.1 数据去重

```
dfread = dfread.dropDuplicates()
```

```
print('Total Records before = {}'.format(dfread.count()))
dfread = dfread.dropDuplicates()
dfread.show()
print('Total Records after = {}'.format(dfread.count()))
```

```
Total Records before = 9226
```

	index	win	firstBlood	firstTower	firstInhibitor	firstBaron	firstDragon	firstRiftHerald	towerKills	inhibitorKills	baronKills	dragonKills	riftHeraldKills	region
0	468	1	1	1	0	0	1	0	1	0	0	0	0	1
1	550	1	1	0	1	0	1	1	6	0	0	0	0	6
1	741	1	1	1	1	0	1	1	6	0	0	0	0	6

### 5.5.2 数据空值处理

根据数据挖掘的思想，数据空值处理的方法主要有删除控制的行、填充空值、filter 过滤数据等。缺失值处理的方法又分为删除和填充，填充主要有均值填充和随机数填充。

这里我根据数据的实际特性进行理论填充，删除空值大于两个的行：

```
dfread = dfread.dropna(how='all',thresh=2)
```

```
dfread = dfread.dropna(how='all', thresh=2)
dfread.show(5)
```

	index	win	firstBlood	firstTower	firstInhibitor	firstBaron	firstDragon	firstRiftHerald	towerKills	inhibitorKills	baronKills	dragonKills	riftHeraldKills	region
0	468	1	1	1	0	0	1	0	1	0	0	0	0	1
1	550	1	1	0	1	0	1	1	6	0	0	0	0	6
1	741	1	1	1	1	0	1	1	6	0	0	0	0	6
1	798	0	1	0	1	1	0	1	3	0	1	0	1	3
0	1138	1	1	1	1	0	0	1	9	0	0	0	1	9

对删除控制大于两个行之后的数据表进行空值填充，根据实际数据列进行填充方法判断。

summary	towerKills	inhibitorKills	baronKills	dragonKills	riftHeraldKills	region
count	9226	9226	9226	9226	9226	9226
mean	4.611207457186213	0.6845870366355951	0.3639713852156948	1.554194667244743	0.7000867114675916	null
stddev	3.4131008741494084	0.957760954646272	0.5835861555207791	1.3406062886946142	0.7380586804492858	null
min	0	0	0	0	0	br1
max	11	7	3	5	2	na1

根据原数据表描述性统计数据可知 towerKills 与 dragonKills 属性的标准差都大于 1，inhibitorKills、baronKills、riftheraldKills 属性标准差均小

于 1，这三个属性实用均值填充。且 towerKills 与 dragonKills 属性相对剩余的 inhibitorKills、baronKills、rifthermaldKills 属性极差偏大，因此不适用于均值填充。

1) 对 inhibitorKills、baronKills、rifthermaldKills 属性采用均值填充，这里展示 inhibitorKillss 属性的均值填充，其他属性类似。

```
import pyspark.sql.functions as F
mean_frame = dfread.select(F.avg(dfread['inhibitorKills']))
inhibitorKills_mean = mean_frame.columns[0]
data_fillna = dfread.fillna(inhibitorKills_mean,'inhibitorKills')
```

```
#均值填充
mean_frame = dfread.select(F.avg(dfread['inhibitorKills']))
inhibitorKills_mean = mean_frame.columns[0]
data_fillna = dfread.fillna(inhibitorKills_mean,'inhibitorKills')
data_fillna.show(5)
dfread = data_fillna
```

index	win	firstBlood	firstTower	firstInhibitor	firstBaron	firstDragon	firstRiftHerald	towerKills	inhibitorKills	baronKills	dragonKills	riftHeraldKills	region
468	1	1	1	0	0	1	0	1	0	1	0	1	1
550	1	1	0	1	0	1	1	1	6	1	0	2	1
741	1	1	1	2	1	0	1	1	6	1	0	2	1
798	0	1	0	1	0	1	0	1	3	0	1	1	0
1138	1	1	1	1	1	0	0	1	9	1	0	2	1

2) 对 towerKills 与 dragonKills 属性进行空值过滤

```
import numpy as np
dfread.filter(dfread.towerKills != np.nan).filter(dfread.dragonKills != np.nan).show(5)
```

```
import numpy as np
dfread.filter(dfread.towerKills != np.nan).filter(dfread.dragonKills != np.nan).show(5)
```

index	win	firstBlood	firstTower	firstInhibitor	firstBaron	firstDragon	firstRiftHerald	towerKills	inhibitorKills	baronKills	dragonKills	riftHeraldKills	region
468	1	1	1	0	0	1	0	1	0	1	0	1	1
550	1	1	0	1	0	1	1	1	6	1	0	2	1
741	1	1	1	2	1	0	1	1	6	1	0	2	1
798	0	1	0	1	0	1	0	1	3	0	1	1	0
1138	1	1	1	1	1	0	0	1	9	1	0	2	1

## 5.6 数据特征提取

### 5.6.1 数值特征

根据上述的数据探索和数据分布分析，可以知道这 13 列均有可能成为游戏胜利的重要因素，因此都可以引入线性回归模型。

### 5.6.2 类别特征

根据热力图分析，除了 region 的后 5 列数据为数值型连续性数据，线性相关程度不大，可以用来进行用户的聚类分析。

```
dfseries = dfread.select('index','towerKills','inhibitorKills',  
'baronKills','dragonKills','riftHeraldKills','region')  
dfseries.show(3)
```

```
dfseries = dfread.select('index','towerKills','inhibitorKills',  
                          'baronKills','dragonKills','riftHeraldKills','region')  
dfseries.show(3)
```

index	towerKills	inhibitorKills	baronKills	dragonKills	riftHeraldKills	region
0	0	0	0	0	0	na
1	1	0	0	2	0	na
2	0	0	0	0	0	na

选取特征项，将特征合并为向量：

```
from pyspark.ml.feature import VectorAssembler  
vecAss = VectorAssembler(inputCols =  
dfseries.drop('region').columns[1:], outputCol = 'features')  
df_featrues = vecAss.transform(dfseries).select('index', 'features')
```

```
#选取特征项，将特征项合并成向量  
from pyspark.ml.feature import VectorAssembler  
vecAss = VectorAssembler(inputCols = dfseries.drop('region').columns[1:], outputCol = 'features')  
df_featrues = vecAss.transform(dfseries).select('index', 'features')  
df_featrues.show(5)
```

index	features
0	(5, [], [])
1	(5, [0, 3], [1.0, 2.0])
2	(5, [], [])
3	[8.0, 1.0, 0.0, 2.0, ...]
4	[10.0, 2.0, 0.0, 1.0, ...]

only showing top 5 rows

### 5.6.3 规范化特征

对玩家聚类相关的特征进行规范化处理，常用的规范化处理是标准化，即对数据集特征每一数据减去特征均值后除以特征标准差。数据标准化可以将对应特征数据变换均值为 0 方差为 1。

经过数据标准化之后，数据集所有特征有了同样的变化范围。数据标准化一个最直接的应用场景就是：当数据集的各个特征取值范围存在较大差异时，或者是各特征取值单位差异较大时，是需要使用标准化来对数据进行预处理的。虽然本次分析使用到的数据量纲相同，且方差均较小，卫视结果更具可信度，为了体现数据挖掘的流程，继续进行标准化处理。

向量化：

```
from pyspark.ml.feature import VectorAssembler
assemble=VectorAssembler(inputCols=['towerKills','inhibitorKills',
                                     'baronKills','dragonKills','riftHeraldKills'
], outputCol='features')
assembled_data=assemble.transform(dfseries)
```

```
from pyspark.ml.feature import VectorAssembler
assemble=VectorAssembler(inputCols=['towerKills','inhibitorKills',
                                     'baronKills','dragonKills','riftHeraldKills'
], outputCol='features')

assembled_data=assemble.transform(dfseries)
assembled_data.show(2)
#向量化
```

index	towerKills	inhibitorKills	baronKills	dragonKills	riftHeraldKills	region	features
0	0	0	0	0	0	na1	(5, [], [])
1	1	0	0	2	0	na1	(5, [0, 3], [1.0, 2.0])

only showing top 2 rows

这里以后五列数据为例，将数据表标准化：

```
data_scale=scale.fit(assembled_data)
data_scale_output=data_scale.transform(assembled_data)
```

```

from pyspark.ml.feature import StandardScaler
#标准化
scale=StandardScaler(inputCol='features', outputCol='standardized')
data_scale=scale.fit(assembled_data)
data_scale_output=data_scale.transform(assembled_data)
data_scale_output.show(3)

```

index	towerKills	inhibitorKills	baronKills	dragonKills	riftHeraldKills	region	features	standardized
0	0	0	0	0	0	na1	(5, [], [])	
1	1	0	0	2	0	na1	(5, [0, 3], [1.0, 2.0])	(5, [0, 3], [0.29298..., 0.29298...])
2	0	0	0	0	0	na1	(5, [], [])	

only showing top 3 rows

## 5.7 建立模型

### 5.7.1 聚类模型

建立游戏相关属性的召唤师聚类模型。

首先需要用一个 for 循环确定聚类数：

```
for i in range(2,10):
```

```
    KMeans_algo=KMeans(featuresCol='standardized', k=i)
```

```
    KMeans_fit=KMeans_algo.fit(data_scale_output)
```

```
    output=KMeans_fit.transform(data_scale_output)
```

```
    score=evaluator.evaluate(output)
```

```
    silhouette_score.append(score)
```

```

from pyspark.ml.clustering import KMeans
from pyspark.ml.evaluation import ClusteringEvaluator
silhouette_score=[]
evaluator = ClusteringEvaluator(predictionCol='prediction', featuresCol='standardized', \
                                | metricName='silhouette', distanceMeasure='squaredEuclidean')

for i in range(2,10):
    KMeans_algo=KMeans(featuresCol='standardized', k=i)
    KMeans_fit=KMeans_algo.fit(data_scale_output)
    output=KMeans_fit.transform(data_scale_output)
    score=evaluator.evaluate(output)
    silhouette_score.append(score)
    print(" Score:", score)

```

Score: 0.5855903767494791

Score: 0.4488890651232723

Score: 0.4335229568824496

Score: 0.4008011413697012

Score: 0.449763633655343

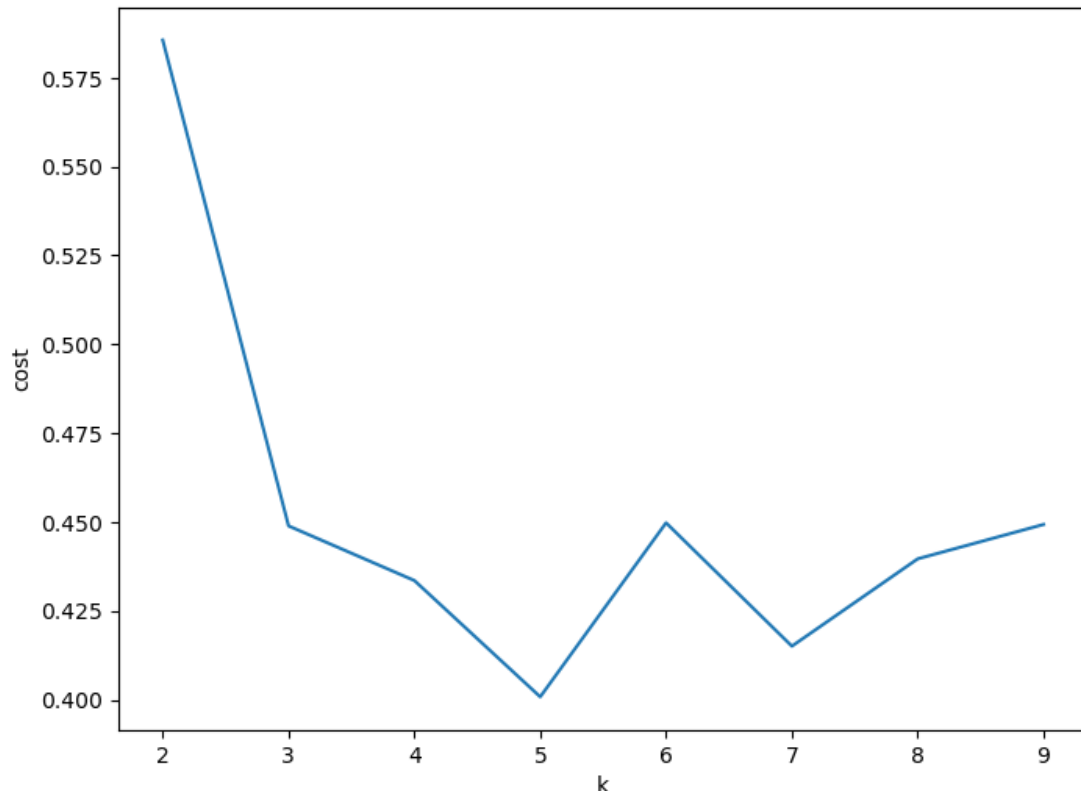
Score: 0.41505961628221794

Score: 0.4396324579585447

Score: 0.44929733790293097

可视化：





观察到  $k=6$  时轮廓分数的局部最大值,  $k=9$  时折线有向上的趋势, 但是类别过多不利于进行召唤师的特征分析, 因此选择聚为 6 类。

因此建立  $k=6$  的 K-Means 聚类模型, 这里直接使用 `pyspark.ml` 库中的 `KMeans` 函数进行 K-means 聚类:

```
kmeans = KMeans(k=6, seed=1)
```

```
model = kmeans.fit(df_feats)
```

```
centers = model.clusterCenters()
```

*#k=6创建模型*

```
from pyspark.ml.clustering import KMeans
kmeans = KMeans(k=6, seed=1)
model = kmeans.fit(df_feats)
centers = model.clusterCenters()
```

*#聚类中心*

```
kmeans_center = model.clusterCenters()
```

```
print(kmeans_center)
```

*#聚类中心*

```
kmeans_center = model.clusterCenters()
print(kmeans_center)
```

```
[array([2.97295944, 0.41111668, 0.09113671, 0.37656485, 0.57736605]), array([8.90084034, 1.70672269, 0.98907563, 3.61764706, 0.90588235]), array([3.13968548, 0.0120259, 0.1813136, 2.40703053, 0.72062905]), array([9.43768769, 1.88138138, 0.76126126, 1.92792793, 1.16591592]), array([6.07261825, 0.62958028, 0.49233844, 2.16189207, 0.9586942]), array([0.43952941, 0.02352941, 0.408, 0.21505882])]
```

```
#聚类个数
kmeans_add = model.summary.clusterSizes
print(kmeans_add)
```

```
#聚类个数
kmeans_add = model.summary.clusterSizes
print(kmeans_add)
```

```
[1997, 1190, 1081, 1332, 1501, 2125]
```

聚成三类的玩家数量分别为 1997, 1190, 1081, 1332, 1501, 2125，每个聚类的样本数分布均匀，可认为聚类效果良好。

# 获取聚类结果

```
transformed = km_model.transform(df_feats).select('index',
'prediction')
```

```
# 获取聚类结果
transformed = model.transform(df_feats).select('index', 'prediction')
# 合并表格
df_pred = dfseries.join(transformed, 'index')
df_pred.show(5)
```

index	towerKills	inhibitorKills	baronKills	dragonKills	riftHeraldKills	region	prediction
0	0	0	0	0	0	na1	5
1	1	0	0	2	0	na1	5
2	0	0	0	0	0	na1	5
3	8	1	0	2	1	na1	3
4	10	2	0	1	0	na1	3

only showing top 5 rows

将聚类结果保存为 result\_julei.csv 文件，见附件。

```
df_pred.toPandas().to_csv("C:\\Users\\yingm\\Desktop\\bigdatabase\\
result_julei.csv",
```

```
index=False,encoding="GBK",header = True,sep=',')
```

聚类结果图表：

index	towerKills	inhibitorK	baronKills	dragonKill	riftHerald	region	prediction
0	0	0	0	0	0	na1	1
1	1	0	0	2	0	na1	1
2	0	0	0	0	0	na1	1
3	8	1	0	2	1	na1	4
4	10	2	0	1	0	na1	0
5	0	0	0	0	0	na1	1
6	4	0	1	2	1	na1	3

## 5.7.2 逻辑回归模型

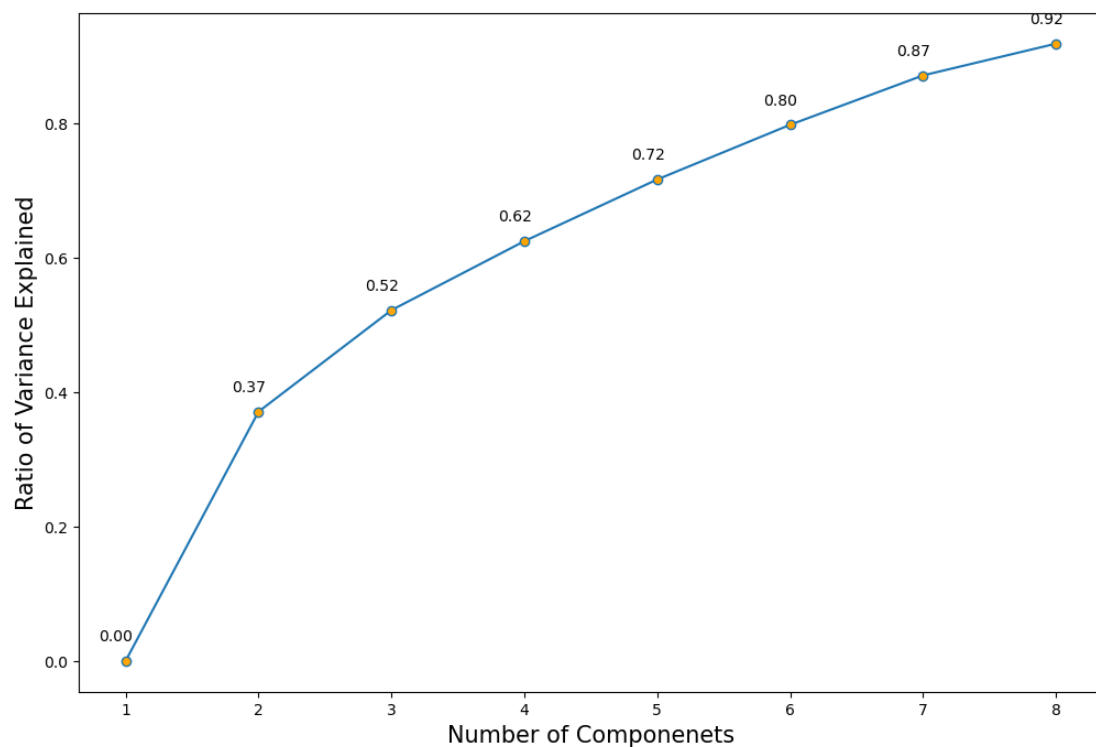
建立游戏胜利因素的 Logistic 回归模型。

将数据分割成一组特征和一组目标，特征是除‘win’和‘region’列之外的所有列，我的目标是‘win’列。

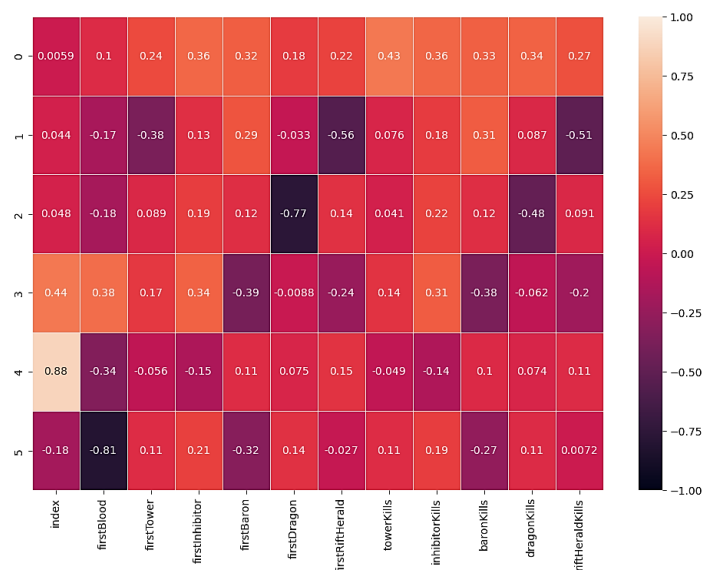
可以看到原表中有 7（是否加成）+5（击杀数）+1（地区）列，需要先进行主成分分析，简化特征。

先看一下可以用多少特征作为主成分：

```
for n in range(0,8):  
    pca = PCA(n_components = n)  
    pca.fit(scaled_data)  
    pca.transform(scaled_data)  
    exp_var_ratio.append(sum(pca.explained_variance_ratio_))
```



可以看到十个预测列中 80%的方差可以用 6 个特征量解释，



查看因子得分热力图，可知无论一个团队是否摧毁了第一个兵营，一个团队推掉了多少塔，以及一个团队摧毁了多少兵营都与获胜有最高的相关性。

我选择将特征缩小为 6 个进行逻辑回归。

接下来进行逻辑回归数据建模。

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = .3)
```

```
log = LogisticRegression()
```

```
log.fit(X_train, y_train)
```

```
y_pred = log.predict(X_test)
```

```
X = match_df.drop(['win', 'region'], axis = 1)
y = match_df.win
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = .3)
```

```
log = LogisticRegression()
```

```
log.fit(X_train, y_train)
```

```
y_pred = log.predict(X_test)
```

```
print(classification_report(y_test, y_pred))
```

```
print(confusion_matrix(y_test, y_pred))
```

```
print()
```

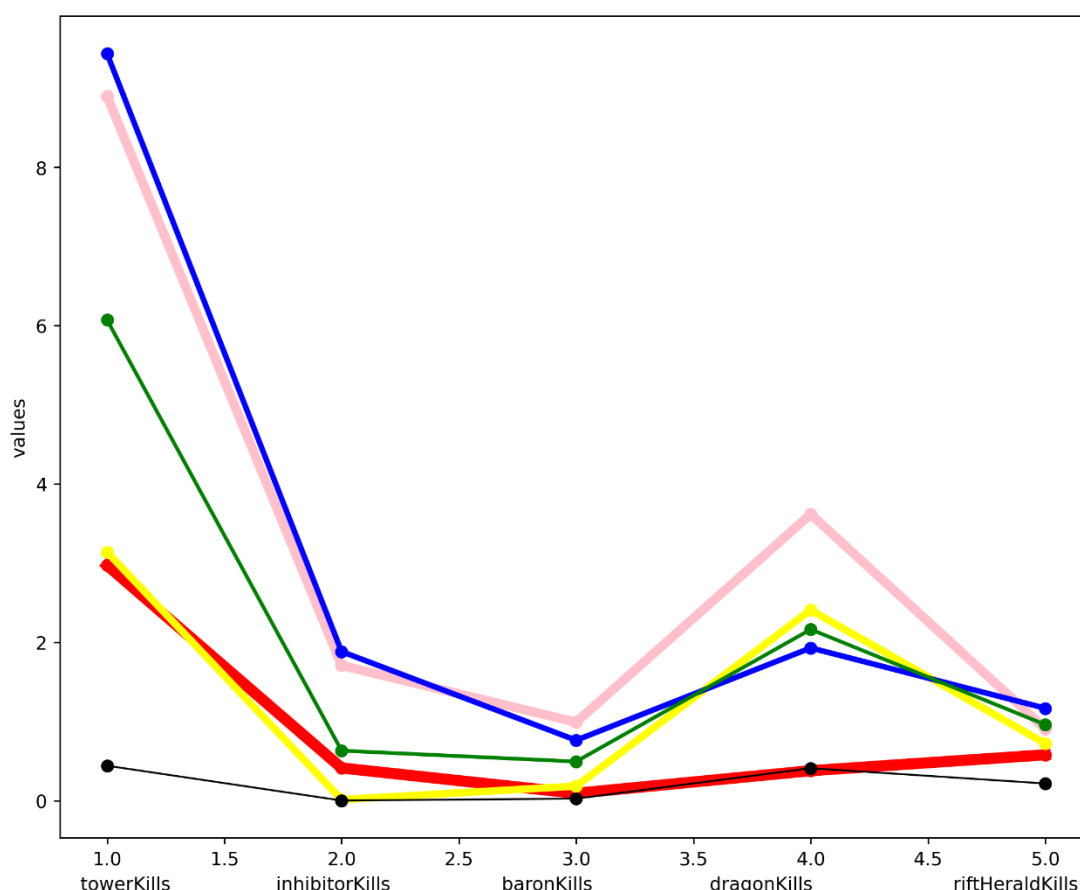
```
log_coeff_tot = pd.Series(log.coef_[0], index = match_df.drop(['win', 'region'], axis = 1).columns)
print(log_coeff_tot)
```

	precision	recall	f1-score	support
0	0.84	0.89	0.86	1360
1	0.89	0.83	0.86	1408
accuracy			0.86	2768
macro avg	0.86	0.86	0.86	2768
weighted avg	0.86	0.86	0.86	2768

```
[[1215 145]
 [ 238 1170]]
```

## 5.8 结果分析

1. 对聚类后的玩家游戏数据进行可视化分析，下图为聚类结果，横轴为五个游戏信息相关的属性，分别为防御塔破坏数、水晶破坏数、纳什男爵击杀数、龙击杀数、峡谷先锋击杀数，每一条曲线代表每一种召唤师聚类。['red','pink','yellow','blue','green','black'] 分别对应第 [0,1,2,3,4,5]类。



根据折线我们可以看到所有聚类类别的玩家在折线上整体的分布都很相似，但都有各自差异，其中每一个聚类都有和其他聚类不同的特点。

例如粉色折线在 **dragonKills** 属性上的值最大，可以认为这类玩家喜欢刷龙，获得团队加成，团队辅助型玩家；蓝色折线在 **towerKills** 的值所有簇中最大，且远大于其他游戏信息属性，可以认为这类玩家注重推塔，是典型的整体目标型玩家；绿色曲线代表的簇位于整体中间，归类于全能型玩家；黄色曲线代表的簇的水晶破坏数在整体最小，相对其他属性也是最小，且纳什男爵击杀数也相对较小，这两个目标在游戏局中都是可再生的，水晶的复活会拉长游戏战线，可以认为这类玩家是速战速决型玩家，不注重水晶破坏和纳什男爵的击杀；重点注意的是红色曲线，这类玩家在整体曲线基本位于低端，是资深玩家中实力相对最弱的玩家，但整体实力除了推塔数量以外都比较均衡，可以认为这类玩家是普通资深玩家；黑色

折线代表的玩家簇接近水平，推塔数量与其他类别玩家相差很大，可以认为这类玩家偏向“划水型”，喜欢在召唤师峡谷游荡刷野怪和水晶，对防御塔的攻击比较消极，可以认为这类玩家是目标消极型。（LOL 整体为推塔游戏，因此本段所指目标均为“游戏胜利”）

因此我们可以得到以下玩家分类类型，整合结果如下表：

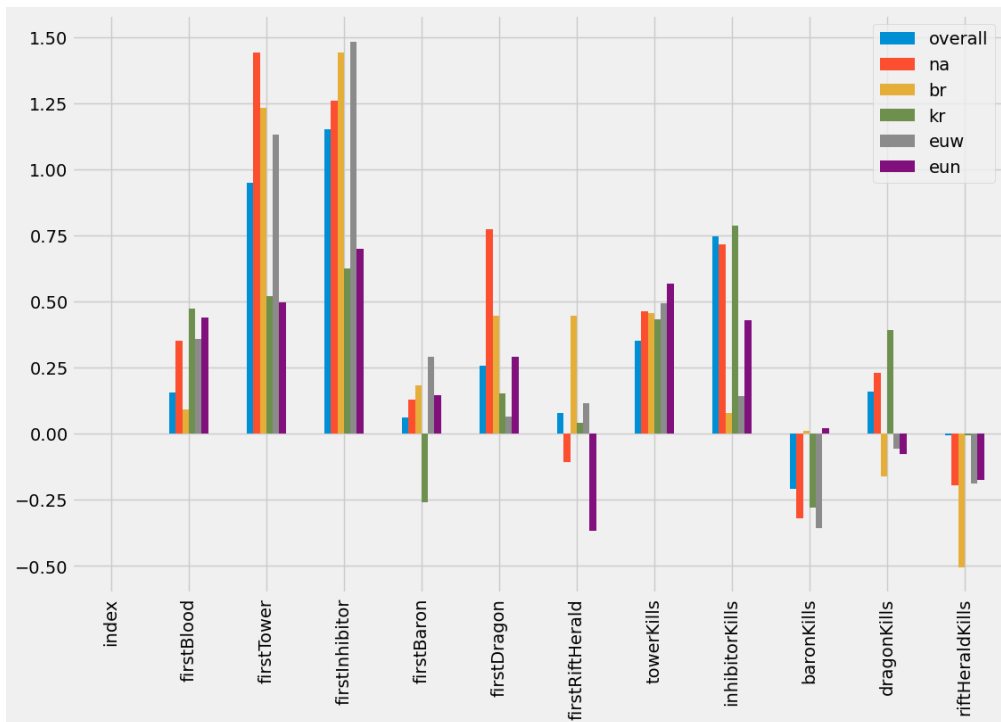
类别	玩家类别命名	簇中样本数
召唤师 0 类	普通资深玩家	1997
召唤师 1 类	团队辅助型玩家	1190
召唤师 2 类	速战速决型玩家	1081
召唤师 3 类	整体目标型玩家	1332
召唤师 4 类	全能型玩家	1501
召唤师 5 类	目标消极型玩家	2125

图表 2 资深玩家聚类表

2. 针对不同地区也进行同样的逻辑回归分析计算系数，得到下表：

	overall	na	br	kr	euw	eun
index	-0.000056	-0.000057	-0.000465	-0.000643	-0.000496	-0.000387
firstBlood	0.155285	0.352765	0.093045	0.472741	0.359047	0.439183
firstTower	0.949051	1.442339	1.233045	0.519728	1.129837	0.495924
firstInhibitor	1.149886	1.259752	1.441163	0.626625	1.482528	0.699291
firstBaron	0.062520	0.130073	0.181838	-0.257803	0.292176	0.146625
firstDragon	0.256334	0.774502	0.447680	0.153970	0.066093	0.291205
firstRiftHerald	0.079523	-0.108139	0.447263	0.039781	0.115502	-0.365775
towerKills	0.350033	0.462630	0.457028	0.431703	0.492825	0.567218
inhibitorKills	0.748038	0.716690	0.079818	0.788355	0.141359	0.428386
baronKills	-0.210058	-0.318264	0.011648	-0.279269	-0.358533	0.021375
dragonKills	0.157777	0.230313	-0.161270	0.393061	-0.057275	-0.076711
riftHeraldKills	-0.006571	-0.193891	-0.505117	-0.005139	-0.187464	-0.174046

将数据进行可视化得到如下柱形图：



根据图中柱子高度，最大到最小的顺序，攻破第一个水晶和第一个防御塔是数据集中最重要的获胜条件，其次是峡谷先锋击杀数。作为一个推塔游戏，本实验得到的数据结果和经验的结果符合。水晶决定对方兵线的再生，防御塔影响召唤师角色的游戏进度，峡谷先锋能为团队获得增益，显然结果具有可信度，对游戏胜利影响最大的因素分别为：

- 1) 第一颗水晶；
- 2) 第一个防御塔；
- 3) 峡谷先锋击杀数。

## 6 实验总结

### 6.1 数据挖掘流程分析

参考了很多网络资料，发现数据挖掘主要进行读取、理解、预处理、建模和分析这几个流程，主要内容和框架可以总结入下：

（一）数据读取：

- 读取数据，并进行展示
- 统计数据各项指标
- 明确数据规模与要完成任务

（二）特征理解分析

- 单特征分析，逐个变量分析其对结果的影响
- 多变量统计分析，综合考虑多种情况影响
- 统计绘图得出结论

### （三）数据清洗与预处理

- 对缺失值进行填充
- 特征标准化/归一化
- 筛选有价值的特征
- 分析特征之间的相关性

### （四）建立模型

- 特征数据与标签准备
- 数据集切分
- （建模算法对比）
- 集成策略等方案改进

## 6.2 实验心得

### 6.2.1 实验总结

聚类算法模型方面，K-means 算法简单易懂且聚类效果较好。对于本文分析的召唤师游戏数据近一万条，K-means 聚类在数据量庞大的数据集相比传统系统聚类具有收敛速度快，聚类效率高的优点。

但由于K-Means聚类算法是结果受初始值影响的局部最优的迭代算法，且受初始聚类中心点选择影响，若初始聚类中心为离群属性点，则会影响整体客户聚类分类效果。算法是不断迭代划分数据中心点的，噪声和离群点会干扰中心划分的距离计算。

本次课程的结课实验过程让我对大数据的处理框架和处理过程有了初步的了解，先后尝试了python、java、scala进行数据处理和分析，maven数据仓库管理jar包，flink和spark用于处理流数据，另外RDD在数据进行计算时有很大的效率优势，结果保存在内存中，大大降低单个算子计算延迟以及不同算子之间的加载延迟。

实验主题内容主要针对pandas从LOL官网上爬取的游戏数据，通过pyspark实现英雄联盟游戏数据的探索性分析、预处理、建模和处理后数据的结果可视化分析，python的很多包为算法的调用和数据计算处理提供了很多方便。总体看来是走了一遍简单的数据挖掘流程，对数据挖掘这个领域也只是一个初步了解，在做实验的过程中发现大数据开发方向很多，要掌握的技术和框架非常发散，分布式系统体系太庞大了，因为不熟悉所以在idea上对动态数据处理的实验过程稍微有点吃力。希望在以后的学习中能够找到自己的兴趣点并针对性学



习，有一个精确的努力方向。

## 6.2.2 模型改进

### 1. 聚类中心选择优化

目前已有K-Means聚类算法改进的相关研究，如K-Means++算法在选取第n+1个聚类中心时，距离当前n个聚类中心越远的点会有更高的概率被选为第n+1个聚类中心，对K-Means随机初始化质心进行优化。二分K-Means算法首先将所有数据看做一个聚类，然后进行聚类划分，保证每一步得到的总体误差最小。

### 2. 距离计算优化

如elkan K-Means算法利用两边之和大于等于第三边,以及两边之差小于第三边的三角形性质，来减少距离的计算。对于一个样本点和两个质心，第一种思路是预先计算两个质心间的距离，若样本点到其中一个质心距离的两倍小于到另一个质心的距离，则该样本点直接属于该质心所属的簇；第二种思路是利用三角形的性质，得到样本到距离最短的质心。对K-Means算法的距离计算进行优化，加快迭代速度。

## 6.3 实验问题记录

1. 数据可视化的时候会遇到部分画图方法只能用 pandas 的情况？包括 dataframe 的 column 也不能被调用为可视化对象，

```
sns.pairplot(data = df0lto,height = 2.5,hue = 'region')
```

```
TypeError: 'data' must be pandas DataFrame object, not: <class 'pyspark.sql.dataframe.DataFrame'>
```

可能是不熟悉这三个库的原因。。，还需要进一步学习加强。最后是通过 toPandas() 方法直接转化成 pandas 再进行数据可视化解决。

2. Spark 读取 pandas 生成的带索引的表的时候会出现 schema 和表头不匹配的警告，检查了一下发现分隔符没有错，表头属性也设置为 True，没报错就忽略了。

```
22/12/20 15:48:54 WARN CSVHeaderChecker: CSV header does not conform to the schema.
Header: , win, firstBlood, firstTower, firstInhibitor, firstBaron, firstDragon, firstRiftHerald,
Schema: _c0, win, firstBlood, firstTower, firstInhibitor, firstBaron, firstDragon, firstRiftHerald
Expected: _c0 but found:
```

3. Spark 在 windows 的环境配置和部署完成后报错找不到 py4j 包，无法启动 spark，但是在命令行可以启动 spark-shell，检查文件路径没有发现错误，问

题原因未解；两行命令解决；

```
import findspark
findspark.init()
```

4. 流数据处理的时候 args[0]，报空对象找不到，args[0]尝试访问 args 数组中的第一个元素，因为它是由命令行参数填充的。如果不传递任何参数，数组是空的，并且试图访问数组中不存在的元素会出现这个异常，修改 run 的启动参数为 localhost 也还是找不到对应路径。暂时未解决，

```
Exception in thread "main" java.io.FileNotFoundException: Create breakpoint : localhost (系统找不到指定的文件。)
    at java.io.FileInputStream.open0(Native Method)
    at java.io.FileInputStream.open(FileInputStream.java:195)
    at java.io.FileInputStream.<init>(FileInputStream.java:138)
    at scala.io.Source$.fromFile(Source.scala:94)
    at scala.io.Source$.fromFile(Source.scala:79)
    at scala.io.Source$.fromFile(Source.scala:57)
    at bigdata_base.testworkcount$.main(testworkcount.scala:18)
    at bigdata_base.testworkcount.main(testworkcount.scala)
```

5. 进行东头数据处理实验的时候，使用端口动态传数据的时候发生错误，原因未知，9000 端口应该是开启了，但是无法发送数据，数据的动态传送没有完成。



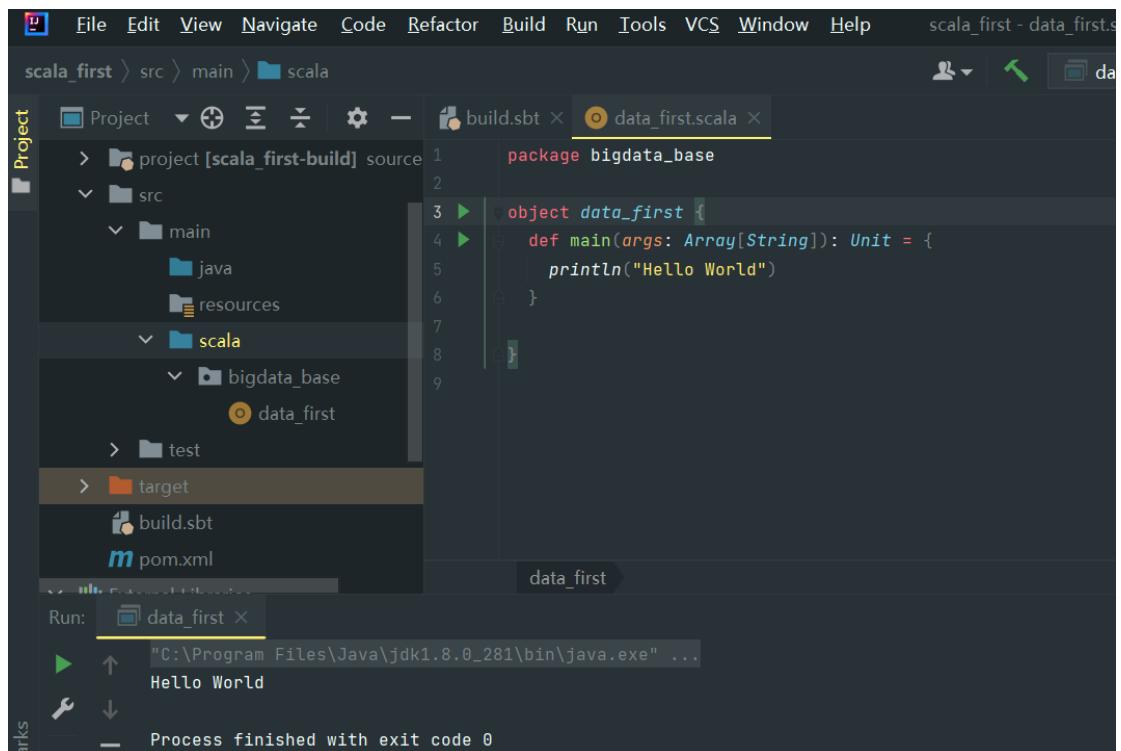
## 7 实验拓展：动态数据处理

本次实验是基于静态的网页数据进行爬取分析，属于数据的离线处理。实际业务中需要对动态数据进行处理分析，事实上如果爬虫是间隔性对网络数据进行爬取，然后对爬取存储后的数据进行调用程序分析也是能够保证数据的时效性的。

学了这门课程还是想尝试一下动态数据处理的流程。所以我搭建了 maven+flink+spark 环境，集成在 idea 上，尝试用 java/scala 进行简单动态数据处理分析作为拓展。

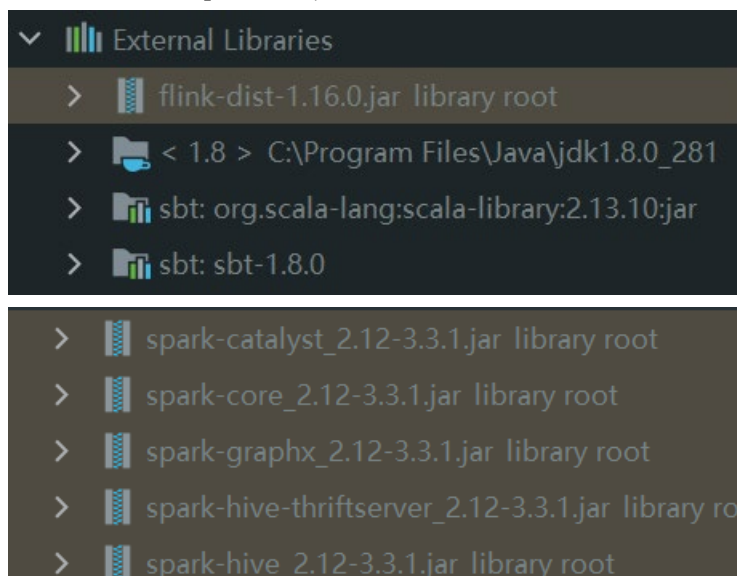
### 7.1 环境搭建

简化实验过程，在 windows 进行动态数据处理实验环境搭建



第一个 scala 程序，运行成功。

Flink 和 spark 以 jars 包的形式导入：



## 7.2 实验内容

在单机模式下实现简单的词频计算批处理和流处理。

### 7.2.1 基于 java+flink 的数据批处理

使用 java 语言和 flink 包实现简单的字符串数据集批处理，DataSet API 用于处理批量数据，数据集通过 source 进行初始化，

导入包：

```
import org.apache.flink.api.common.functions.FlatMapFunction;
import org.apache.flink.api.java.DataSet;
import org.apache.flink.api.java.ExecutionEnvironment;
import org.apache.flink.api.java.tuple.Tuple2;
import org.apache.flink.util.Collector;
```

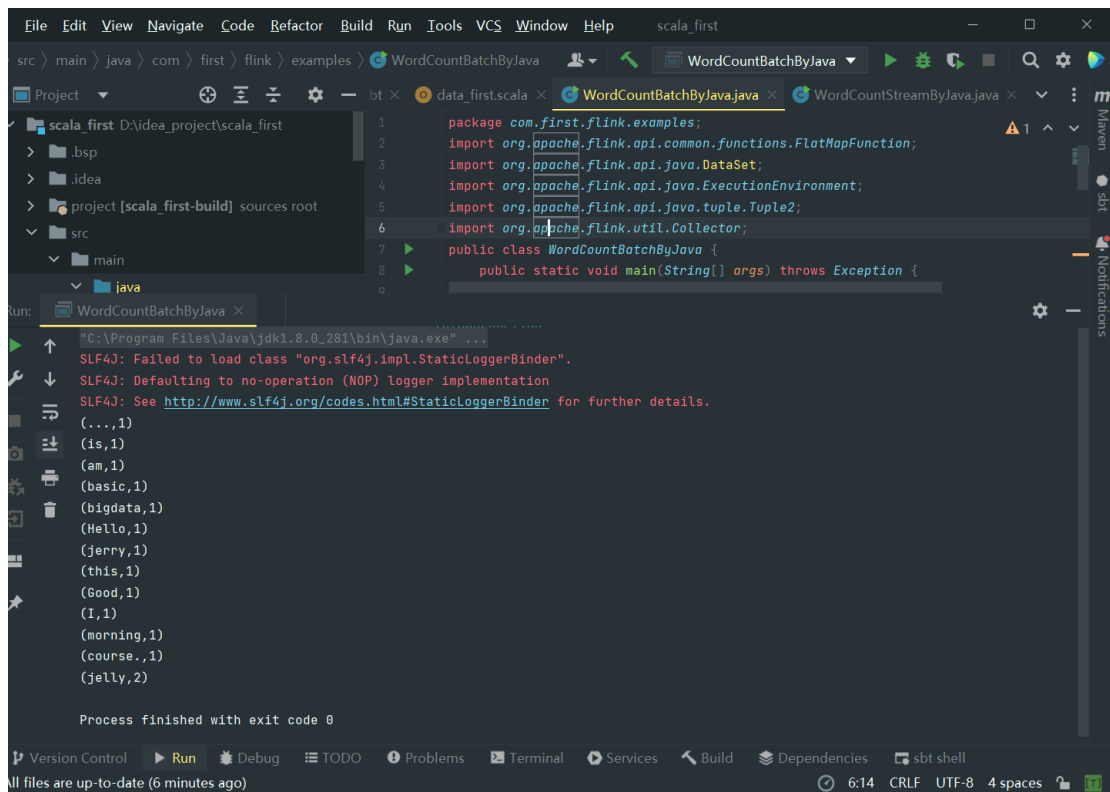
这里构建一个字符串数据集进行处理：

```
DataSet<String> text = env.fromElements(
    ...data: "I am jelly ",
    "Hello jerry",
    "Good morning ... jelly",
    "this is bigdata basic course."
);
```

分组统计：

```
DataSet<Tuple2<String, Integer>> wordCount = text
    .flatMap(new LineSplitter()) FlatMapOperator<String, Tuple2<String, Integer>>
    .groupBy( ...fields: 0) UnsortedGrouping<Tuple2<String, Integer>
    .sum( field: 1);
wordCount.print();
```

分组统计结果：



```
File Edit View Navigate Code Refactor Build Run Tools VCS Window Help scala_first
src > main > java > com > first > flink > examples > WordCountBatchByJava
Project > scala_first D:\idea_project\scala_first
  > .bsp
  > .idea
  > project [scala_first-build] sources root
  > src
    > main
      > java
        WordCountBatchByJava.java
          1 package com.first.flink.examples;
          2 import org.apache.flink.api.common.functions.FlatMapFunction;
          3 import org.apache.flink.api.java.DataSet;
          4 import org.apache.flink.api.java.ExecutionEnvironment;
          5 import org.apache.flink.api.java.tuple.Tuple2;
          6 import org.apache.flink.util.Collector;
          7 public class WordCountBatchByJava {
          8     public static void main(String[] args) throws Exception {
          9
          10
          11
          12
          13
          14
          15
          16
          17
          18
          19
          20
          21
          22
          23
          24
          25
          26
          27
          28
          29
          30
          31
          32
          33
          34
          35
          36
          37
          38
          39
          40
          41
          42
          43
          44
          45
          46
          47
          48
          49
          50
          51
          52
          53
          54
          55
          56
          57
          58
          59
          60
          61
          62
          63
          64
          65
          66
          67
          68
          69
          70
          71
          72
          73
          74
          75
          76
          77
          78
          79
          80
          81
          82
          83
          84
          85
          86
          87
          88
          89
          90
          91
          92
          93
          94
          95
          96
          97
          98
          99
          100
          101
          102
          103
          104
          105
          106
          107
          108
          109
          110
          111
          112
          113
          114
          115
          116
          117
          118
          119
          120
          121
          122
          123
          124
          125
          126
          127
          128
          129
          130
          131
          132
          133
          134
          135
          136
          137
          138
          139
          140
          141
          142
          143
          144
          145
          146
          147
          148
          149
          150
          151
          152
          153
          154
          155
          156
          157
          158
          159
          160
          161
          162
          163
          164
          165
          166
          167
          168
          169
          170
          171
          172
          173
          174
          175
          176
          177
          178
          179
          180
          181
          182
          183
          184
          185
          186
          187
          188
          189
          190
          191
          192
          193
          194
          195
          196
          197
          198
          199
          200
          201
          202
          203
          204
          205
          206
          207
          208
          209
          210
          211
          212
          213
          214
          215
          216
          217
          218
          219
          220
          221
          222
          223
          224
          225
          226
          227
          228
          229
          230
          231
          232
          233
          234
          235
          236
          237
          238
          239
          240
          241
          242
          243
          244
          245
          246
          247
          248
          249
          250
          251
          252
          253
          254
          255
          256
          257
          258
          259
          260
          261
          262
          263
          264
          265
          266
          267
          268
          269
          270
          271
          272
          273
          274
          275
          276
          277
          278
          279
          280
          281
          282
          283
          284
          285
          286
          287
          288
          289
          290
          291
          292
          293
          294
          295
          296
          297
          298
          299
          300
          301
          302
          303
          304
          305
          306
          307
          308
          309
          310
          311
          312
          313
          314
          315
          316
          317
          318
          319
          320
          321
          322
          323
          324
          325
          326
          327
          328
          329
          330
          331
          332
          333
          334
          335
          336
          337
          338
          339
          340
          341
          342
          343
          344
          345
          346
          347
          348
          349
          350
          351
          352
          353
          354
          355
          356
          357
          358
          359
          360
          361
          362
          363
          364
          365
          366
          367
          368
          369
          370
          371
          372
          373
          374
          375
          376
          377
          378
          379
          380
          381
          382
          383
          384
          385
          386
          387
          388
          389
          390
          391
          392
          393
          394
          395
          396
          397
          398
          399
          400
          401
          402
          403
          404
          405
          406
          407
          408
          409
          410
          411
          412
          413
          414
          415
          416
          417
          418
          419
          420
          421
          422
          423
          424
          425
          426
          427
          428
          429
          430
          431
          432
          433
          434
          435
          436
          437
          438
          439
          440
          441
          442
          443
          444
          445
          446
          447
          448
          449
          450
          451
          452
          453
          454
          455
          456
          457
          458
          459
          460
          461
          462
          463
          464
          465
          466
          467
          468
          469
          470
          471
          472
          473
          474
          475
          476
          477
          478
          479
          480
          481
          482
          483
          484
          485
          486
          487
          488
          489
          490
          491
          492
          493
          494
          495
          496
          497
          498
          499
          500
          501
          502
          503
          504
          505
          506
          507
          508
          509
          510
          511
          512
          513
          514
          515
          516
          517
          518
          519
          520
          521
          522
          523
          524
          525
          526
          527
          528
          529
          530
          531
          532
          533
          534
          535
          536
          537
          538
          539
          540
          541
          542
          543
          544
          545
          546
          547
          548
          549
          550
          551
          552
          553
          554
          555
          556
          557
          558
          559
          560
          561
          562
          563
          564
          565
          566
          567
          568
          569
          570
          571
          572
          573
          574
          575
          576
          577
          578
          579
          580
          581
          582
          583
          584
          585
          586
          587
          588
          589
          590
          591
          592
          593
          594
          595
          596
          597
          598
          599
          600
          601
          602
          603
          604
          605
          606
          607
          608
          609
          610
          611
          612
          613
          614
          615
          616
          617
          618
          619
          620
          621
          622
          623
          624
          625
          626
          627
          628
          629
          630
          631
          632
          633
          634
          635
          636
          637
          638
          639
          640
          641
          642
          643
          644
          645
          646
          647
          648
          649
          650
          651
          652
          653
          654
          655
          656
          657
          658
          659
          660
          661
          662
          663
          664
          665
          666
          667
          668
          669
          670
          671
          672
          673
          674
          675
          676
          677
          678
          679
          680
          681
          682
          683
          684
          685
          686
          687
          688
          689
          690
          691
          692
          693
          694
          695
          696
          697
          698
          699
          700
          701
          702
          703
          704
          705
          706
          707
          708
          709
          710
          711
          712
          713
          714
          715
          716
          717
          718
          719
          720
          721
          722
          723
          724
          725
          726
          727
          728
          729
          730
          731
          732
          733
          734
          735
          736
          737
          738
          739
          740
          741
          742
          743
          744
          745
          746
          747
          748
          749
          750
          751
          752
          753
          754
          755
          756
          757
          758
          759
          760
          761
          762
          763
          764
          765
          766
          767
          768
          769
          770
          771
          772
          773
          774
          775
          776
          777
          778
          779
          780
          781
          782
          783
          784
          785
          786
          787
          788
          789
          790
          791
          792
          793
          794
          795
          796
          797
          798
          799
          800
          801
          802
          803
          804
          805
          806
          807
          808
          809
          810
          811
          812
          813
          814
          815
          816
          817
          818
          819
          820
          821
          822
          823
          824
          825
          826
          827
          828
          829
          830
          831
          832
          833
          834
          835
          836
          837
          838
          839
          840
          841
          842
          843
          844
          845
          846
          847
          848
          849
          850
          851
          852
          853
          854
          855
          856
          857
          858
          859
          860
          861
          862
          863
          864
          865
          866
          867
          868
          869
          870
          871
          872
          873
          874
          875
          876
          877
          878
          879
          880
          881
          882
          883
          884
          885
          886
          887
          888
          889
          890
          891
          892
          893
          894
          895
          896
          897
          898
          899
          900
          901
          902
          903
          904
          905
          906
          907
          908
          909
          910
          911
          912
          913
          914
          915
          916
          917
          918
          919
          920
          921
          922
          923
          924
          925
          926
          927
          928
          929
          930
          931
          932
          933
          934
          935
          936
          937
          938
          939
          940
          941
          942
          943
          944
          945
          946
          947
          948
          949
          950
          951
          952
          953
          954
          955
          956
          957
          958
          959
          960
          961
          962
          963
          964
          965
          966
          967
          968
          969
          970
          971
          972
          973
          974
          975
          976
          977
          978
          979
          980
          981
          982
          983
          984
          985
          986
          987
          988
          989
          990
          991
          992
          993
          994
          995
          996
          997
          998
          999
          1000
          1001
          1002
          1003
          1004
          1005
          1006
          1007
          1008
          1009
          1010
          1011
          1012
          1013
          1014
          1015
          1016
          1017
          1018
          1019
          1020
          1021
          1022
          1023
          1024
          1025
          1026
          1027
          1028
          1029
          1030
          1031
          1032
          1033
          1034
          1035
          1036
          1037
          1038
          1039
          1040
          1041
          1042
          1043
          1044
          1045
          1046
          1047
          1048
          1049
          1050
          1051
          1052
          1053
          1054
          1055
          1056
          1057
          1058
          1059
          1060
          1061
          1062
          1063
          1064
          1065
          1066
          1067
          1068
          1069
          1070
          1071
          1072
          1073
          1074
          1075
          1076
          1077
          1078
          1079
          1080
          1081
          1082
          1083
          1084
          1085
          1086
          1087
          1088
          1089
          1090
          1091
          1092
          1093
          1094
          1095
          1096
          1097
          1098
          1099
          1100
          1101
          1102
          1103
          1104
          1105
          1106
          1107
          1108
          1109
          1110
          1111
          1112
          1113
          1114
          1115
          1116
          1117
          1118
          1119
          1120
          1121
          1122
          1123
          1124
          1125
          1126
          1127
          1128
          1129
          1130
          1131
          1132
          1133
          1134
          1135
          1136
          1137
          1138
          1139
          1140
          1141
          1142
          1143
          1144
          1145
          1146
          1147
          1148
          1149
          1150
          1151
          1152
          1153
          1154
          1155
          1156
          1157
          1158
          1159
          1160
          1161
          1162
          1163
          1164
          1165
          1166
          1167
          1168
          1169
          1170
          1171
          1172
          1173
          1174
          1175
          1176
          1177
          1178
          1179
          1180
          1181
          1182
          1183
          1184
          1185
          1186
          1187
          1188
          1189
          1190
          1191
          1192
          1193
          1194
          1195
          1196
          1197
          1198
          1199
          1200
          1201
          1202
          1203
          1204
          1205
          1206
          1207
          1208
          1209
          1210
          1211
          1212
          1213
          1214
          1215
          1216
          1217
          1218
          1219
          1220
          1221
          1222
          1223
          1224
          1225
          1226
          1227
          1228
          1229
          1230
          1231
          1232
          1233
          1234
          1235
          1236
          1237
          1238
          1239
          1240
          1241
          1242
          1243
          1244
          1245
          1246
          1247
          1248
          1249
          1250
          1251
          1252
          1253
          1254
          1255
          1256
          1257
          1258
          1259
          1260
          1261
          1262
          1263
          1264
          1265
          1266
          1267
          1268
          1269
          1270
          1271
          1272
          1273
          1274
          1275
          1276
          1277
          1278
          1279
          1280
          1281
          1282
          1283
          1284
          1285
          1286
          1287
          1288
          1289
          1290
          1291
          1292
          1293
          1294
          1295
          1296
          1297
          1298
          1299
          1300
          1301
          1302
          1303
          1304
          1305
          1306
          1307
          1308
          1309
          1310
          1311
          1312
          1313
          1314
          1315
          1316
          1317
          1318
          1319
          1320
          1321
          1322
          1323
          1324
          1325
          1326
          1327
          1328
          1329
          1330
          1331
          1332
          1333
          1334
          1335
          1336
          1337
          1338
          1339
          1340
          1341
          1342
          1343
          1344
          1345
          1346
          1347
          1348
          1349
          1350
          1351
          1352
          1353
          1354
          1355
          1356
          1357
          1358
          1359
          1360
          1361
          1362
          1363
          1364
          1365
          1366
          1367
          1368
          1369
          1370
          1371
          1372
          1373
          1374
          1375
          1376
          1377
          1378
          1379
          1380
          1381
          1382
          1383
          1384
          1385
          1386
          1387
          1388
          1389
          1390
          1391
          1392
          1393
          1394
          1395
          1396
          1397
          1398
          1399
          1400
          1401
          1402
          1403
          1404
          1405
          1406
          1407
          1408
          1409
          1410
          1411
          1412
          1413
          1414
          1415
          1416
          1417
          1418
          1419
          1420
          1421
          1422
          1423
          1424
          1425
          1426
          1427
          1428
          1429
          1430
          1431
          1432
          1433
          1434
          1435
          1436
          1437
          1438
          1439
          1440
          1441
          1442
          1443
          1444
          1445
          1446
          1447
          1448
          1449
          1450
          1451
          1452
          1453
          1454
          1455
          1456
          1457
          1458
          1459
          1460
          1461
          1462
          1463
          1464
          1465
          1466
          1467
          1468
          1469
          1470
          1471
          1472
          1473
          1474
          1475
          1476
          1477
          1478
          1479
          1480
          1481
          1482
          1483
          1484
          1485
          1486
          1487
          1488
          1489
          1490
          1491
          1492
          1493
          1494
          1495
          1496
          1497
          1498
          1499
          1500
          1501
          1502
          1503
          1504
          1505
          1506
          1507
          1508
          1509
          1510
          1511
          1512
          1513
          1514
          1515
          1516
          1517
          1518
          1519
          1520
          1521
          1522
          1523
          1524
          1525
          1526
          1527
          1528
          1529
          1530
          1531
          1532
          1533
          1534
          1535
          1536
          1537
          1538
          1539
          1540
          1541
          1542
          1543
          1544
          1545
          1546
          1547
          1548
          1549
          1550
          1551
          1552
          1553
          1554
          1555
          1556
          1557
          1558
          1559
          1560
          1561
          1562
          1563
          1564
          1565
          1566
          1567
          1568
          1569
          1570
          1571
          1572
          1573
          1574
          1575
          1576
          1577
          1578
          1579
          1580
          1581
          1582
          1583
          1584
          1585
          1586
          1587
          1588
          1589
          1590
          1591
          1592
          1593
          1594
          1595
          1596
          1597
          1598
          1599
          1600
          1601
          1602
          1603
          1604
          1605
          1606
          1607
          1608
          1609
          1610
          1611
          1612
          1613
          1614
          1615
          1616
          1617
          1618
          1619
          1620
          1621
          1622
          1623
          1624
          1625
          1626
          1627
          1628
          1629
          1630
          1631
          1632
          1633
          1634
          1635
          1636
          1637
          1638
          1639
          1640
          1641
          1642
          1643
          1644
          1645
          1646
          1647
          1648
          1649
          1650
          1651
          1652
          1653
          1654
          1655
          1656
          1657
          1658
          1659
          1660
          1661
          1662
          1663
          1664
          1665
          1666
          1667
          1668
          1669
          1670
          1671
          1672
          1673
          1674
          1675
          1676
          1677
          1678
          1679
          1680
          1681
          1682
          1683
          1684
          1685
          1686
          1687
          1688
          1689
          1690
          1691
          1692
          1693
          1694
          1695
          1696
          1697
          1698
          1699
          1700
          1701
          1702
          1703
          1704
          1705
          1706
          1707
          1708
          1709
          1710
          1711
          1712
          1713
          1714
          1715
          1716
          1717
          1718
          1719
          1720
          1721
          1722
          1723
          1724
          1725
          1726
          1727
          1728
          1729
          1730
          1731
          1732
          1733
          1734
          1735
          1736
          1737
          1738
          1739
          1740
          1741
          1742
          1743
          1744
          1745
          1746
          1747
          1748
          1749
          1750
          1751
          1752
          1753
          1754
          1755
          1756
          1757
          1758
          1759
          1760
          1761
          1762
          1763
          1764
          1765
          1766
          1767
          1768
          1769
          1770
          1771
          1772
          1773
          1774
          1775
          1776
          1777
          1778
          1779
          1780
          1781
          1782
          1783
          1784
          1785
          1786
          1787
          1788
          1789
          1790
          1791
          1792
          1793
          1794
          1795
          1796
          1797
          1798
          1799
          1800
          1801
          1802
          1803
          1804
          1805
          1806
          1807
          1808
          1809
          1810
          1811
          1812
          1813
          1814
          1815
          1816
          1817
          1818
          1819
          1820
          1821
          1822
          1823
          1824
          182
```

7.2.2 基于 scala+spark 的数据流处理

使用 scala 语言和 spark 包进行简单的数据流处理。  
最后模仿网络资料尝试了一下 scala 语言实现的动态数据流处理，但是由于端口号开启和文件路径的问题暂时未解决，流数据处理没有实现。。以下为实验过程。  
导包：

```
import org.apache.spark.SparkConf
import org.apache.spark.streaming._;
```

开本地线程两个处理，每隔 5 秒计算一批数据

```
def main(args: Array[String]) {
    //开本地线程两个处理，local[4]：意思本地起4个进程运行，setAppName("SparkStreaming")：设置运行处理类
    val conf = new SparkConf().setMaster("local[4]").setAppName("SparkStreaming")
    //每隔5秒计算一批数据
    val ssc = new StreamingContext(conf, Seconds(5))
```

Wordcount 计算

```
//按\t 切分输入数据
val words = lines.flatMap(_.split( regex = " "))
//计算wordcount
val pairs = words.map(word => (word, 1))
//word ++
val wordCounts = pairs.reduceByKey(_ + _)
//排序结果集打印，先转成rdd，然后排序true升序，false降序，可以指定key和value排序_._1是key，_._2是val
val sortResult = wordCounts.transform(rdd => rdd.sortBy(_._2, ascending = false))
```

8 附件说明

下表为本次实验报告的附件

文件类别	文件名	说明
数据表 文件	matches.csv	网页爬虫结果数据表
	result_julei.csv	聚类结果数据表
源代码文件	data_grasp.ipynb	网页爬虫源代码
	data_deal.ipynb	数据探索性分析、召唤师聚类及可视化源代码
	data_logistic.ipynb	LOL 游戏胜利因素分析可视化源代码
文件	scala_first	动态数据处理项目文件

图表 3 附件说明

