数据导论项目：关于排球比赛的胜率预测

一、序言

在得知我们将要完成一次个人的数据作品时，我第一时间就想到将自己的兴趣和大作业相结合，也就是对VNL（FIVB Volleyball Nations League）的比赛结果进行一次预测和胜率的分析。在整个项目过程中，由于对于机器学习的不熟悉，经历了两次数据集完全推倒重来，以及数据探索和数据建模上的困难。所幸，在室友和网络资源的帮助下，我排尽万难，完成了本次项目。

二、文件结构

（一）视频链接

（二）具体结构

三、数据收集

本次数据的来源是'https://en.volleyballworld.com/volleyball/competitions/vnl-2023/schedule/'及其子网页。由于VNL只有近三年的比赛数据，因此只爬取了2021-2023年的比赛数据。然而即使只有三年，总共也大约二十多万行，总共爬取了一个多小时；外加爬虫需要阅读纷繁复杂的html代码，soup解析出的对象格式也不尽相同，反复打磨爬虫的代码，真的很辛苦！

（一）第一次收集

在最开始，我打算采用request库和beautiful soup库对网页进行数据收集。很幸运的是，最初网站不需要动态加载，能够之间使用request静态读取。因此，我使用如下代码（spider_volley.py）就对数据完成了收集。

```
from bs4 import BeautifulSoup
import requests
import csv
import pandas as pd
# 先创建好一个表格的框架
team_scores = pd.read_csv('team_scores.csv')
country_name = team_scores.columns.tolist()
team_scores_v1_index1 = [item for item in country_name for _ in
range(len(country_name))]  # 把这个列表中的元素重复21次
team_scores_v1_columns = ['Total Score', 'Match1', 'Match2', 'Match3', 'Match4',
'Match5']
team_scores_v1_index2 = country_name * len(country_name)
team_scores1 = pd.DataFrame(data=None, columns=[team_scores_v1_columns],
                            index=[team_scores_v1_index1, team_scores_v1_index2])

# 从排球网站中爬取相应数据
second_response =
requests.get('https://en.volleyballworld.com/volleyball/competitions/vnl-
2023/schedule/')
html_content2 = second_response.text.encode(second_response.encoding).decode()
soup2 = BeautifulSoup(html_content2, 'lxml')

# 获得每场大比赛的信息
matches = soup2.find_all('a', class_='vbw-mu-finished vbw-mu__data')
```

```
for match in matches:
    # 获得本场比赛的队伍的名称
    home_team_name = match.find_all('div', class_='vbw-mu__team__name vbw-
mu__team__name--abbr')[0].text
    away_team_name = match.find_all('div', class_='vbw-mu__team__name vbw-
mu__team__name--abbr')[1].text

    # 获得本场比赛的total scores
    match_score = match.find('div', class_='vbw-mu__score').text
    team_scores1.at[(home_team_name, away_team_name), 'Total Score'] =
match_score

    # 获得本场比赛的小分
    round_scores = match.find_all('div', class_='vbw-mu__sets--result')
    round_scores_list = [item.text for item in round_scores[0:5]]  # 在item后面加
text，而非round_scores!
    team_scores1.loc[
        (home_team_name, away_team_name), ['Match1', 'Match2', 'Match3',
'Match4', 'Match5']] = round_scores_list

team_scores1 = team_scores1.fillna('0:0')  # 使用fillna将NaN值填充为0:0

team_scores1.to_csv('team_scores_23T.csv')
```

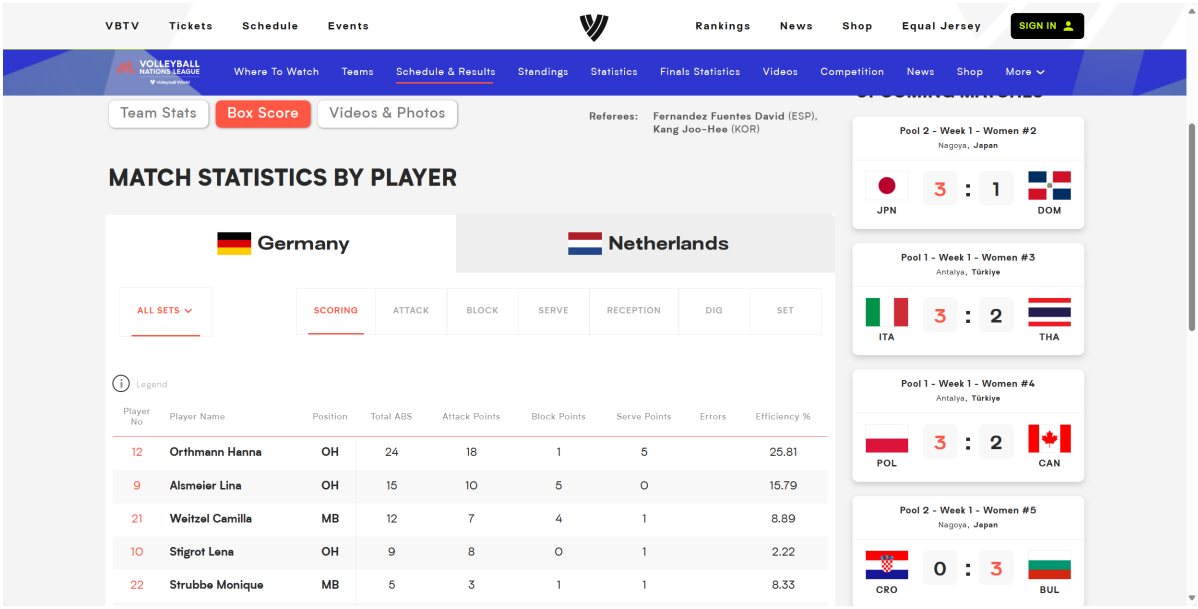
这个数据爬取出来的csv文件结构大致为：

```
,,Total Score,Match1,Match2,Match3,Match4,Match5
CRO,CRO,0:0,0:0,0:0,0:0,0:0,0:0
CRO,BRA,0:3,24-26,18-25,8-25,-,-
CRO,BUL,0:3,12-25,17-25,19-25,-,-
CRO,CAN,0:3,21-25,21-25,26-28,-,-
CRO,CHN,0:0,0:0,0:0,0:0,0:0,0:0
```

也就是先给出Home_Country和Away_Country，然后判断他们是否有进行比赛。如果有，则把相关数据填入DataFrame中。然而这样做有一个显著的弊端：赤裸裸的数据，不能衡量各个国家队伍之间的水平。因此，我们还需要对每场比赛的细节，每个国家的战术战略做出分析。

（二）第二次数据收集

在网站中，每场比赛的数据存放在这样的位置：



最初，我只打算使用浏览器的检查功能，仿照第一次收集数据的思路，爬取对应的表格即可。然而，在我尝试**使用request库解析代码时，却总是检测不到任何表格元素。**

随后我查阅了网上的相关资料，得知某些网站的数据是动态加载的，同时，部分数据是必须要接受cookie权限才能阅读的。很不幸，这个网站也是如此。因此，必须在本次爬取的过程中使用selenium库了！

查阅网上相关资料得知，我们必须先取得网站的cookie，然后把cookie保存到本地，最后使用这些cookie给我们的爬虫授权。总结下列相关代码：

```
# 获得起始网页的解析，由于这个网站的超链接是动态加载的，request库已经不能满足需求，因此必须使用更高级的selenium
start_url = 'https://en.volleyballworld.com/volleyball/competitions/vnl-2023/schedule/'
driver = webdriver.Chrome()  # 请确保你的ChromeDriver已经安装并在PATH中
driver.get(start_url)
driver.implicitly_wait(10)
driver.delete_all_cookies()  # 固定搭配暂不知道具体原因
cookies = [{'domain': '.volleyballworld.com', 'expiry': 1705370977, 'httpOnly': False, 'name': '_hjIncludedInSessionSample_1859763', 'path': '/', 'sameSite': 'None', 'secure': True, 'value': '0'},
 {'domain': '.volleyballworld.com', 'expiry': 1705371008, 'httpOnly': False, 'name': '_hjSession_1859763', 'path': '/', 'sameSite': 'None', 'secure': True, 'value': 'eyJpZCI6ImJiY2EwMDZhLTVmMDEtNGUzNi04NjFmLWNiOTQ1ZWZhNDU1NyIsImMiOjE3MDUzNjkxNzc4OTcsInMiOjAsInIiOjAsInNiIjoxLCJzciI6MCwic2UiOjAsImZzIjoxLCJzcCI6MX0='},
 {'domain': '.volleyballworld.com', 'expiry': 1736905177, 'httpOnly': False, 'name': '_hjSessionUser_1859763', 'path': '/', 'sameSite': 'None', 'secure': True, 'value': 'eyJpZCI6IjVhOTU5YjI0LTUyNTctNWIyMy05ZmJjLTM2OGRlYjU4MGFhMyIsImNyZWF0ZWQiOjE3MDUzNjkxNzc4OTYsImV4aXN0aW5nIjpmYWxzZX0='},
 {'domain': '.volleyballworld.com', 'expiry': 1739237978, 'httpOnly': False, 'name': '__qca', 'path': '/', 'sameSite': 'Lax', 'secure': False, 'value': 'P0-1930540427-1705369177238'},
```

```
 {'domain': '.volleyballworld.com', 'expiry': 1705455576, 'httpOnly': False,
'name': '_gid', 'path': '/', 'sameSite': 'Lax', 'secure': False, 'value':
'GA1.2.1968396925.1705369177'},
 {'domain': '.volleyballworld.com', 'expiry': 1739929176, 'httpOnly': False,
'name': '_ga_R215V1S3VM', 'path': '/', 'sameSite': 'Lax', 'secure': False,
'value': 'GS1.1.1705369176.1.1.1705369176.60.0.0'},
 {'domain': '.volleyballworld.com', 'expiry': 1739929176, 'httpOnly': False,
'name': '_ga_0XQMFZ8Y93', 'path': '/', 'sameSite': 'Lax', 'secure': False,
'value': 'GS1.2.1705369176.1.1.1705369176.0.0.0'},
 {'domain': '.volleyballworld.com', 'expiry': 1739929176, 'httpOnly': False,
'name': '_ga', 'path': '/', 'sameSite': 'Lax', 'secure': False, 'value':
'GA1.2.2082547222.1705369177'},
 {'domain': '.volleyballworld.com', 'expiry': 1705369236, 'httpOnly': False,
'name': '_gat_UA-185656906-1', 'path': '/', 'sameSite': 'Lax', 'secure': False,
'value': '1'},
 {'domain': '.volleyballworld.com', 'expiry': 1739929176, 'httpOnly': False,
'name': '_ga_5ZY57D5RSS', 'path': '/', 'sameSite': 'Lax', 'secure': False,
'value': 'GS1.1.1705369176.1.0.1705369176.60.0.0'},
 {'domain': '.en.volleyballworld.com', 'expiry': 1736991576, 'httpOnly': False,
'name': '_vwo_uuid_v2', 'path': '/', 'sameSite': 'Lax', 'secure': False, 'value':
'D72DEF0D3CC158B21EE65E432171BA7A3|4d10c619ffe29b899362Obe3fe219b07'},
 {'domain': '.volleyballworld.com', 'expiry': 1713145177, 'httpOnly': False,
'name': '_gcl_au', 'path': '/', 'sameSite': 'Lax', 'secure': False, 'value':
'1.1.350320605.1705369177'}]

for cookie in cookies:
    if 'expiry' in cookie:  # 有的cookie里面有这个参数，有的没有。有的话，需要做处理。
        del cookie['expiry']   # 这个expiry参数值得类型会影响到登录识别，所以需要删掉或者
更改值为整形
    driver.add_cookie(cookie)
driver.refresh()   # 刷新页面
driver.get(start_url)   # 可以直接登录后到网站
```

有了这样的模块，我们就能直接爬取动态加载的数据了。

那么，为什么这次数据收集（spider_volleyTeams.py）不成功呢？本次数据收集希望的csv结构如下：

```
Country,Total,OH,MB,OP,Block,Serve,Dig,Reception,Error,Turns
BEL,668.0,477.0,188.0,0.0,121.0,76.0,1222.0,1129.0,288.0,54.8
BRA,690.0,502.0,162.0,19.0,128.0,45.0,934.0,771.0,187.0,46.2
RUS,728.0,397.0,110.0,200.0,152.0,55.0,1066.0,1045.0,257.0,53.0
CAN,643.0,452.0,148.0,31.0,130.0,61.0,915.0,1140.0,320.0,52.4
CHN,784.0,520.0,218.0,38.0,129.0,47.0,1115.0,1128.0,215.0,54.599999999999994
DOM,621.0,429.0,115.0,68.0,98.0,48.0,895.0,912.0,260.0,44.800000000000004
```

事实上我也完成了相关数据的收集，并且获得了相应的结果。然而，可以注意到，这个表格中的Block、Serve等数目非常地大（注：一场比赛中这些数据一般不会超过10）。这是因为我在爬取过程中直接把每一局的数据叠加到了原有的表格上，尽管最后可以通过turns求得平均值，但是却损失了非常多的数据，自然，完全无法完成后续的数据建模。因此，我们只有重新构建我们的爬虫逻辑。

（三）最终的收集

这个爬虫前前后后花费了近一天半的时间才调试完成。尽管网站对爬虫几乎没有限制，然而由于官方网站是国外网站，网速本来就慢，加上爬取的过程中总是不停地中断，算是这次大作业非常痛苦的环节之一了。接下来，我将讲解这个爬虫的具体逻辑。（Final_spider.py）

1."宏定义"

虽然python语言中没有C语言的宏定义功能，但是我们借用这个概念，去强调程序的可维护性和健壮性。具体则是由于我们需要爬取多个起始页不同的网页，同时存储文件的名称与格式都不一样，所以，我们必须在最开始给一些元数据进行命名和赋值。例如：

```
start_url23 = 'https://en.volleyballworld.com/volleyball/competitions/vnl-
2023/schedule/'
start_url22 = 'https://en.volleyballworld.com/volleyball/competitions/vnl-
2022/schedule/'
start_url21 = 'https://en.volleyballworld.com/volleyball/competitions/vnl-
2021/schedule/'
years = 2023
start_url = start_url23
"""接下来，按照女排比赛顺序，依次爬取表格内容。
补充：2023年女排比赛序号：16024 - 16127
     2022年女排比赛序号：13754 - 13857
     2021年女排比赛序号：11830 - 11953"""
begin_num = 16024
end_num = 16127
```

2.动态加载

上文提及，无需多言。

3.爬取逻辑

根据我们对网站的观察和排球比赛经验的积累，可以知道比赛分为两支队伍，采取五局三胜制。网站把每一局的数据和总局数据都保存在*'tbody',class_='vbw-o-table__body'*的元素中，所以，我们可以直接用soup定位表格主体这一元素， 而且由于其长度固定，出现的形式也固定，因此能够用数字索引对每个表格实现精确定位。
例如，在只有4局的一场比赛中，一定会有（2x7x5+1）个表格。而且7个表格总是以['SCORING', 'ATTACK', 'BLOCK', 'SERVE', 'RECEPTION', 'DIG', 'SET']的顺序出现。因此，我们能直接定位每个表格是什么类型的，属于哪个国家。

根据我们pandas库的知识，一个表格的元属性只能通过df.attrs['items']来刻画，而且attrs不能会被打印出来。也不会显式地保存在csv文件中。因此，我们必须把元属性全部保存到每一行中，尽管这样比较浪费空间，但是方便后续对表格进行分类。元属性包括：
Meta_columns = ['Country', 'type', 'num_set', 'this_turn_score', 'num_race', 'win_or_lose']
其中‘win_or_lose‘是借助'this_turn_score'得出的，后续可以直接用于数据建模，非常方便。

在第二次收集中存在一个非常严重的逻辑漏洞，影响了最终收集的数据准确性。在爬取scores时，我想当然地认为文本格式总是”XX-XX“,所以简单地使用了切片读取数据。这样做的后果是忽视了”X-XX“等结果，导致部分数据未读出，使得后续的数据全部错位！
正确的处理方式是采用正则表达式读取数据。细节是应该把正则表达式读出的结果整数化，否则字符串比较是字典序地，将会导出（15<9）这样滑稽的结果，误导后续判断。

部分内容如下：

```
        if number_set == 0:
            This_turn_score = 0
            WinOrLose = None
        else:
        # 使用正则表达式提取数字
            # sco_numbers = re.findall(r'\d+', scores[number_set - 1])  # 现在每组
scores的数字都存在一个数组中
            sco_numbers = [int(num) for num in re.findall(r'\d+',
scores[number_set - 1])]
            This_turn_score = sco_numbers[1]
            WinOrLose = 1 if sco_numbers[1] > sco_numbers[0] else 0
```

4.文件保存

考虑到总共有三年，每年有七个不同的表格，因此我们保存文件的时候不能直接把文件的名称写入路径中，这样会显得臃肿，且难以维护。因此，我们借助格式化文本结构对文件进行保存。

另外，由于每次保存都是保存的一个新的表格，因此总是会把表头写入。我们只保留第一次的表头，采取追加写入的方式，并后续的表头应该用*header=False*沉默掉。

5.小结

这个爬虫让我领略了网页从静态到动态的飞跃，提高了个人实践操作的严谨和细心程度，体会到项目的艰难和可维护性的重要性。必须经历过维护困难和文件储存出错的痛苦才能写出好代码！

四、数据探索

经过上述数据挖掘后，我们可以得到21个文件。例如，其中一个文件如下：

```
    Player No   Player Name Position    Point   Errors  Attempts    Total
Efficiency %    Country type    num_set this_turn_score num_race    win_or_lose
0   13  Boz Meryem  O   16  8   17  41  19.51   TUR ATTACK  0   0   11831
1   99  Karakurt Ebrar  O   13  6   13  32  21.88   TUR ATTACK  0   0   11831
2   14  Erdem Dündar Eda (C)    MB  10  7   6   23  13.04   TUR ATTACK  0   0
11831
```

这样的文件保留了很多珍贵的原始数据，但是很明显，网页原本的排版顺序是以选手为基础的，而我们此处的逻辑应该以国家为基础。那么这个时候我们就应该搬出来第一次爬取的数据了。

```
Country,Total,OH,MB,OP,Block,Serve,Dig,Reception,Error,Turns
BEL,668.0,477.0,188.0,0.0,121.0,76.0,1222.0,1129.0,288.0,54.8
BRA,690.0,502.0,162.0,19.0,128.0,45.0,934.0,771.0,187.0,46.2
RUS,728.0,397.0,110.0,200.0,152.0,55.0,1066.0,1045.0,257.0,53.0
CAN,643.0,452.0,148.0,31.0,130.0,61.0,915.0,1140.0,320.0,52.4
CHN,784.0,520.0,218.0,38.0,129.0,47.0,1115.0,1128.0,215.0,54.599999999999994
DOM,621.0,429.0,115.0,68.0,98.0,48.0,895.0,912.0,260.0,44.800000000000004
```

把这些数据全部除以Turns，就可以得到每个队伍各项能力的平均指标。注意，这里的Turns也是二次加工后的产物——它统计了一个国家本次赛季的小局数量，第1-4局算1局，第5局算0.6局（因为第五局率先获得15颗球即胜利，0.6 = 15/25）

那么拥有了这样的数据，我们就可以利用matlablib初步作图了。

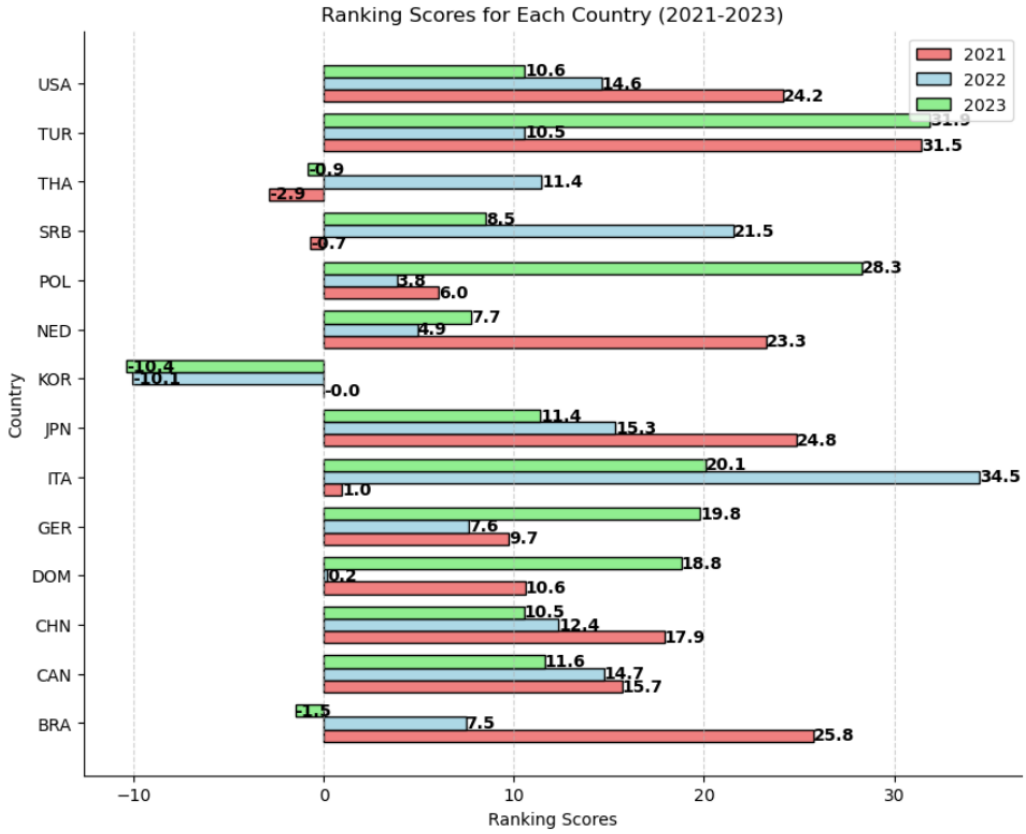
最直观的想法就是先观察每支队伍的进攻风格。
首先是概念上的科普。OH指outside hitter，即主攻，一般在四号位；MB指middle blocker，即副攻，一般在三号位；OP指opposite，即边攻，又称接应，一般在2号位。而不同球队的主力进攻人员擅长的进攻点位不同，那么球队Attack得分中，各个位置的占比也就不同。此时，我们能根据OH、MB、OP的大小关系作饼状图：

不难看出，**大多数队伍OH进攻数量较多，但仍有少部分球队如SRB等，以接应为主力或者CRO等，三个位置平分进攻机会。**

而后，我们还可以通过大比分对各个国家进行排名。依据2019年的积分数据（这个数据来源于volleybox.com），我们能得到下述字典：

```
# 首先需要一个基本的排名积分
rank_2019 = {'USA':235.75, 'DOM':187.6, 'BRA':167, 'CHN':162.5,
             'JPN':124.9, 'SRB':118.1, 'RUS':103.6, 'TUR':103,
             'CAN':96.4, 'ITA':92, 'POL':87, 'NED':80.8,
             'KOR':72.6, 'THA':60.8, 'GER':61.8, 'BEL':54.8,
             'BUL':31, 'CRO':59.2}
```

接着将积分值取10的对数，再按照赢球加分，输球减分的原则，并结合积分排名偏差值的思想，对双方的排名进行动态的分析：



不难看出，初始排名积分低的国家在赢球次数多的情况下，更具有积分增长的优势。因为偏差值决定他们在赢球时，大概率比他们的对手排名积分低，而这将导致他们赢球获得的积分更多。例如，2022年的意大利表现就十分突出，而意大利上一年的分数并不算高。

还有一些当时进行数据探索时作的图片，详情可以移步 *VNLdata_exploring-checkpoint.ipynb*。

五、特征工程

数据探索时我们使用的大多是第一次和第二次收集的结果，而没有对第三次收集的表格加以利用。那么接下来我们将要深入解析第三次收集的数据对我们接下来数据建模的作用。

1.数据预处理：合并文件

根据第三次数据收集获得的7种表格可知，我们合并文件的时候不能简单地把他们并在一起。那么如何简洁地处理不同年份，不同种类的21张表格呢?

首先，我们将不同的表格按照种类分出7个大类，因为种类相同的表格可以直接去掉表头并入后续项，而年份相同的不行。执行这个操作的文件为*ContactFile.py*，部分代码如下：

```
k = 0
for i in range(0,21,3):
    df1 = pd.read_csv(filepath[i])
    df2 = pd.read_csv(filepath[i+1])
    df3 = pd.read_csv(filepath[i+2])

    df_combined = pd.concat([df1, df2])
    df_combined = pd.concat([df_combined, df3])
    df_combined.to_csv(f'combined_{type_name[k]}.csv',index=False)
    k += 1
```

这样做，我们就获得了7个csv文件。

2.特征提取

通过对排球比赛的了解，我们应该能够初步地预测哪些因素对一场比赛的胜负有影响。在这里，我选择了13个特征向量，其中胜负是因变量，其后12个是自变量，而num_race和num_set等是用于表格合并的辅助列：

```
feature_columns = ['num_race', 'num_set','Country', 'win_or_lose',
                   'attack_scores', 'attack_errors', 'block_effective' ,
'block_errors',
                   'dig_succeeded', 'dig_errors', 'reception_succeeded',
'reception_errors',
                   'serve_scores', 'serve_errors', 'set_efficiency'
                   ]
```

而具体的提取方法则分为三个步骤：

首先，确认我们新建的DataFrame需要的列。（上文以给出）
接下来，分表格提取相应列。在这里我们需要用到经典的groupby函数和紧随其后的三板斧：transform，apply和agg。注意，这些特征列事实上都在原始表格的基础上进行了加和，或者进行了特定函数运算而得出。详细的运算过程见*Final_featuring.py*。例如：

```
features_original6['efficiency'] = features_original6['Point'] /
features_original6['Total']
features['set_efficiency'] = features_original6['efficiency'].round(3)
```

最后，我们将得到的DataFrame输出。

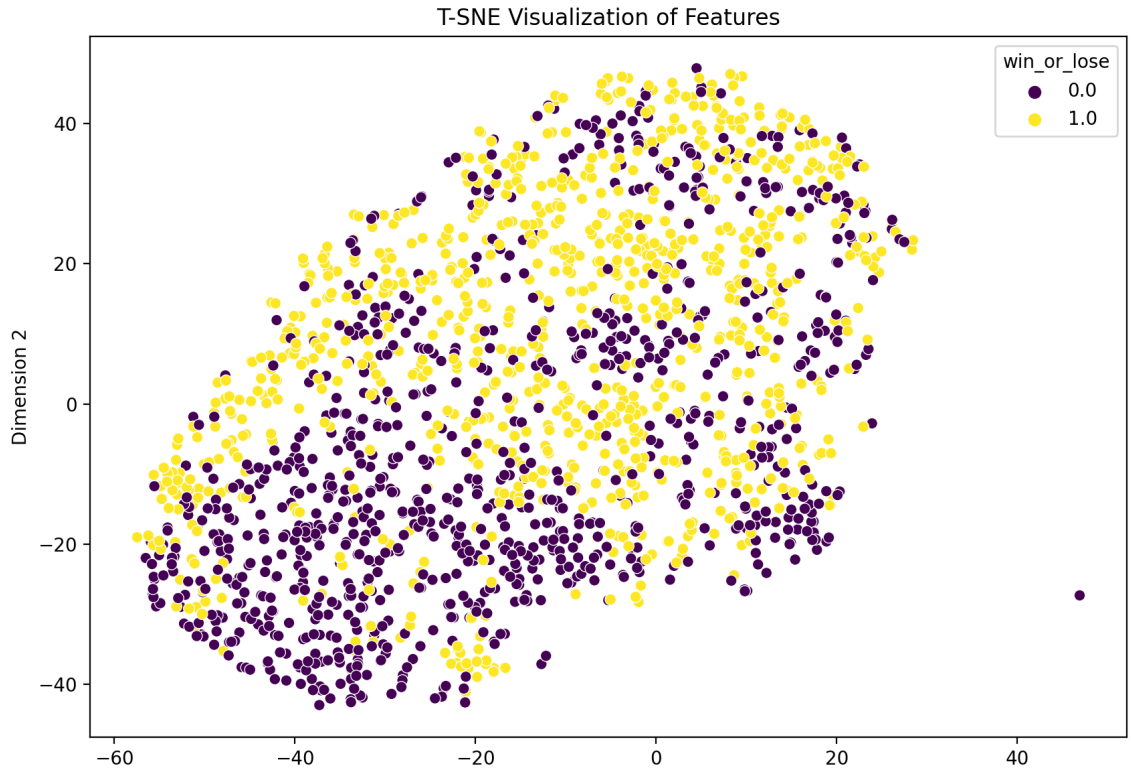
3.特征转换与选择

特征值的归一化过程我们放在了数据建模之中。接下来我们需要做的就是确定我们初步筛选出的特征值有没有不适合作为训练集的。为了完成这个目的，我们需要使用T-SNE图和热力图。

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.manifold import TSNE

# 从文件中读取数据
filepath = 'Final_features.csv'
df = pd.read_csv(filepath)

# 选择特征向量
features = df[['attack_scores', 'attack_errors', 'block_effective',
'block_errors',
               'dig_succeeded', 'dig_errors', 'reception_succeeded',
'reception_errors',
               'serve_scores', 'serve_errors']]

# 初始化 T-SNE 模型
tsne = TSNE(random_state=42)

# 使用 T-SNE 进行降维
```

```
features_tsne = tsne.fit_transform(features)

# 创建 DataFrame 以方便绘图
df_tsne = pd.DataFrame(features_tsne, columns=['Dimension 1', 'Dimension 2'])

# 添加 'win_or_lose' 列以区分不同类别
df_tsne['win_or_lose'] = df['win_or_lose']

# 使用 Seaborn 绘制散点图
plt.figure(figsize=(10, 8))
sns.scatterplot(x='Dimension 1', y='Dimension 2', hue='win_or_lose',
data=df_tsne, palette='viridis')
plt.title('T-SNE Visualization of Features')
plt.show()
```

这是绘制T-SNE图的源代码，而后我们可以得到如下图片：

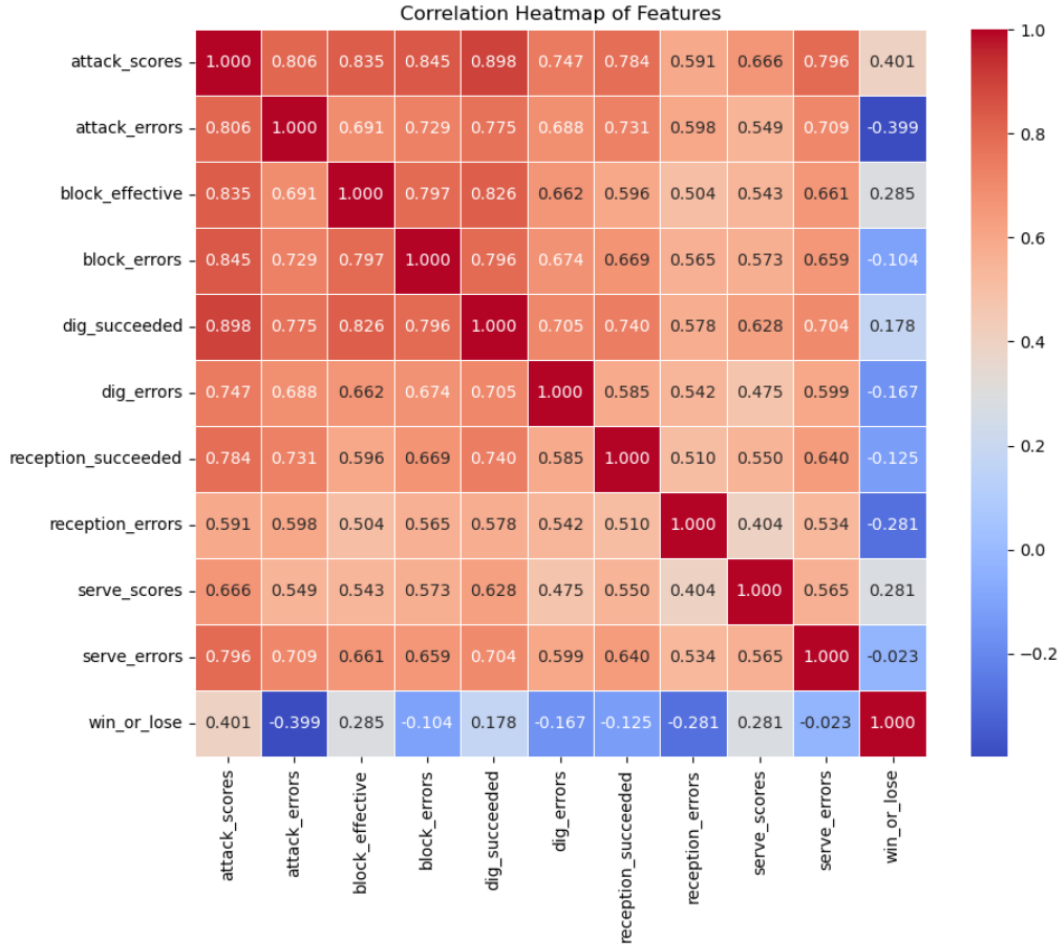


根据这张图我们将高维的数据降为低维，并且可视化了win_or_lose两个集合之间的差异。不难看出，0和1，即胜与负答案大体上可以被区分出来，但是仍然有约25%的数据难以被区分，这是事实也指向了比赛也存在爆冷结局的可能性。因此，我们能判断选出的这些特征向量可以被确定。

再使用热力图来验证我们上述猜想。

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

# 从文件中读取数据
filepath = 'Final_features.csv'
df = pd.read_csv(filepath)

# 选择特征向量
```

```
features = df[['attack_scores', 'attack_errors', 'block_effective',
'block_errors',
               'dig_succeeded', 'dig_errors', 'reception_succeeded',
'reception_errors',
               'serve_scores', 'serve_errors', 'win_or_lose']]

# 计算特征之间的相关性矩阵
correlation_matrix = features.corr()

# 使用Seaborn绘制热力图
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt='.3f',
linewidths=.5)
plt.title('Correlation Heatmap of Features')
plt.show()
```



这张图片中，自变量之间的相关系数大多超过0.5，尤其是进攻分数与许多指标都超过0.8，**这说明一个队伍的进攻实力很大程度上与队伍的综合水平正相关。**表格中最大的值除去1则是0.898，是dig_succeeded和attack_scores的相关系数，这说明进攻强力的队伍往往在防守方面也十分强悍，这对应于排球比赛中，强力主攻往往自身的救球能力和一传能力也是一流。

再看胜负和自变量之间的关系。其中多数都是负值，最小的是attack_errors，这说明一场比赛的胜负不会被队伍的长板所决定，而是由短板所影响。当然，这个相关系数的绝对值也不算高，这也说明比赛作为一个混沌的系统，不会被某一个因素轻易的影响。

六、数据建模与结果预测

得到了特征工程后的DataFrame后，我们就需要进行数据建模了。在本次数据建模中，我采取了四种建模方式。

（一）数据建模

1.SVM模型（支持向量机）

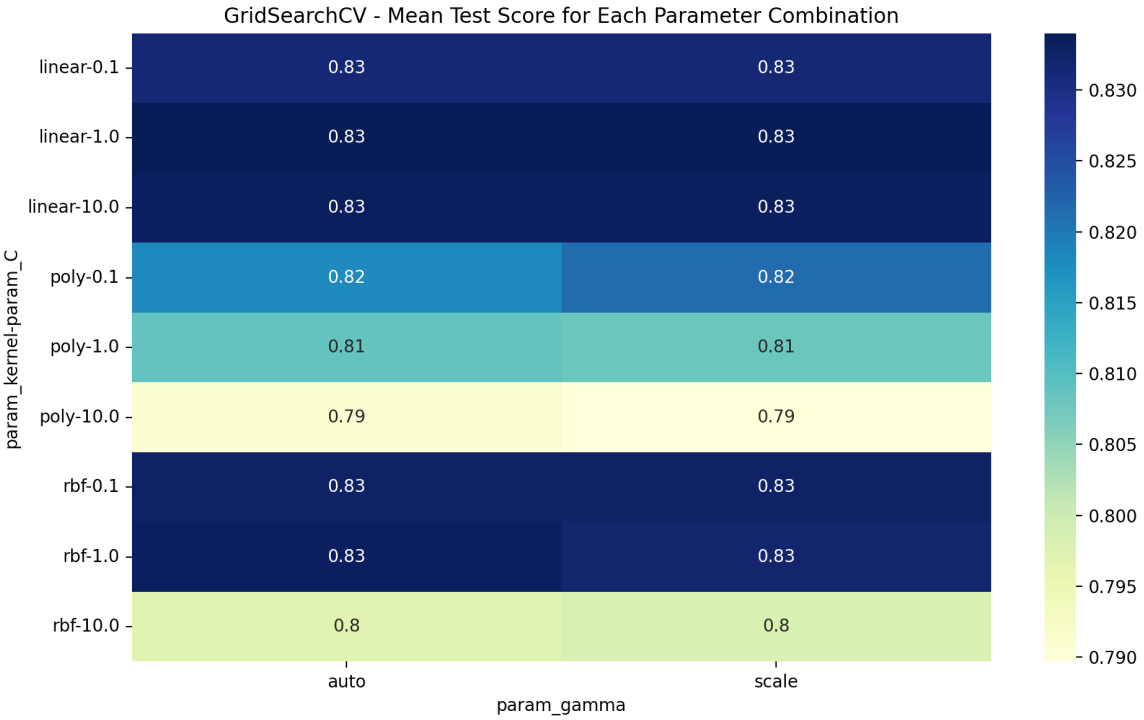
参照实验手册的代码，我们可以得到支持向量机的结果。

结果如下：

```
SVM:
Best Parameters: {'C': 1, 'gamma': 'scale', 'kernel': 'linear'}
Accuracy: 0.8386243386243386
Confusion Matrix:
 [[151  38]
 [ 23 166]]
Classification Report:
              precision    recall  f1-score   support

         0.0       0.87      0.80      0.83       189
         1.0       0.81      0.88      0.84       189

    accuracy                           0.84       378
   macro avg       0.84      0.84      0.84       378
weighted avg       0.84      0.84      0.84       378
```

我们看到，预测的正确率高达0.84，这说明我们的特征选取得非常不错。同时，不论是精确度，召回率还是F1分数，都超过了0.8，这说明我们的模型基本拟合正确。

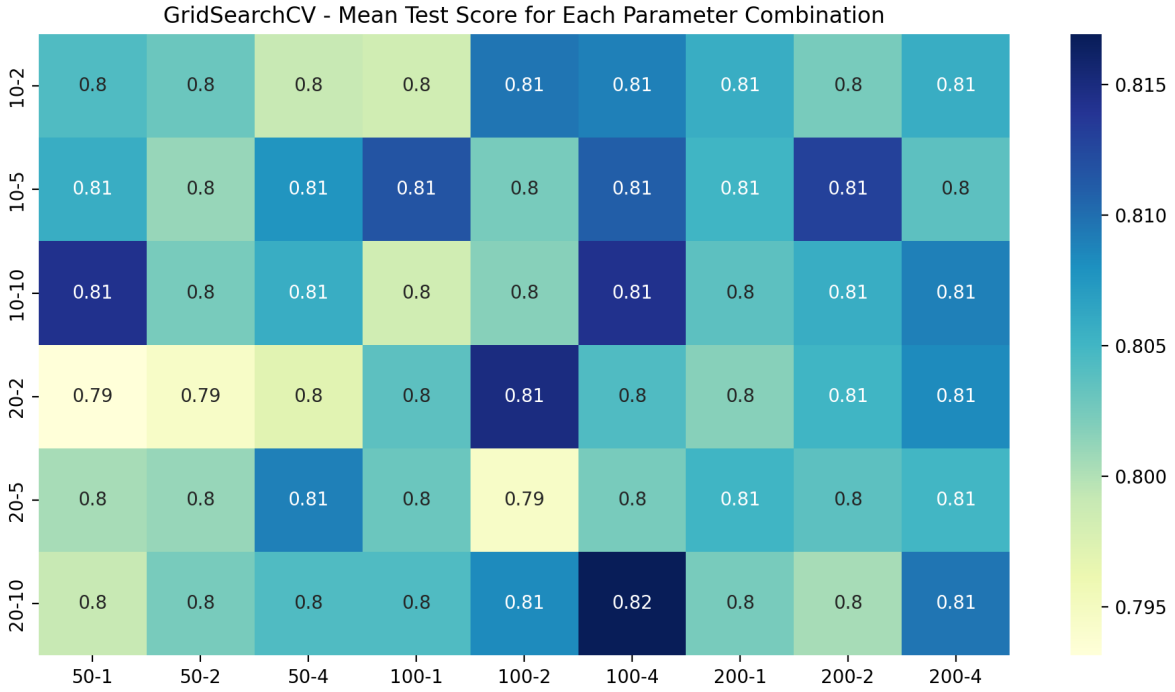
以下是采用不同参数和核函数后获得的参数。当然，我们在上述模型中已经选择了最优的模型。



2.随机森林

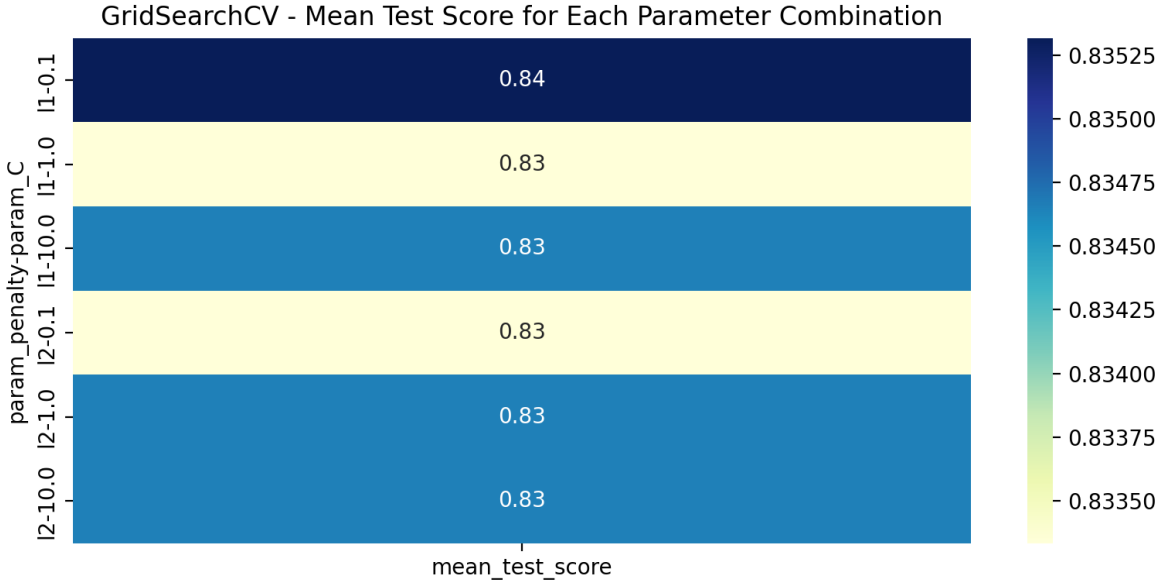
我们实现随机森林的方式也是仿照了实验手册的步骤。较为不同的一点在于，set_efficiency是一个值非常不稳定的小数，因此我们在这里把他舍弃，以免影响我们的决策。

```
Random Forest Classifier:
Best Parameters: {'max_depth': 20, 'min_samples_leaf': 4, 'min_samples_split': 10, 'n_estimators': 100}
Accuracy: 0.8021108179419525
Confusion Matrix:
 [[149  43]
 [ 32 155]]
Classification Report:
              precision    recall  f1-score   support

         0.0       0.82      0.78      0.80       192
         1.0       0.78      0.83      0.81       187

    accuracy                           0.80       379
   macro avg       0.80      0.80      0.80       379
weighted avg       0.80      0.80      0.80       379
```



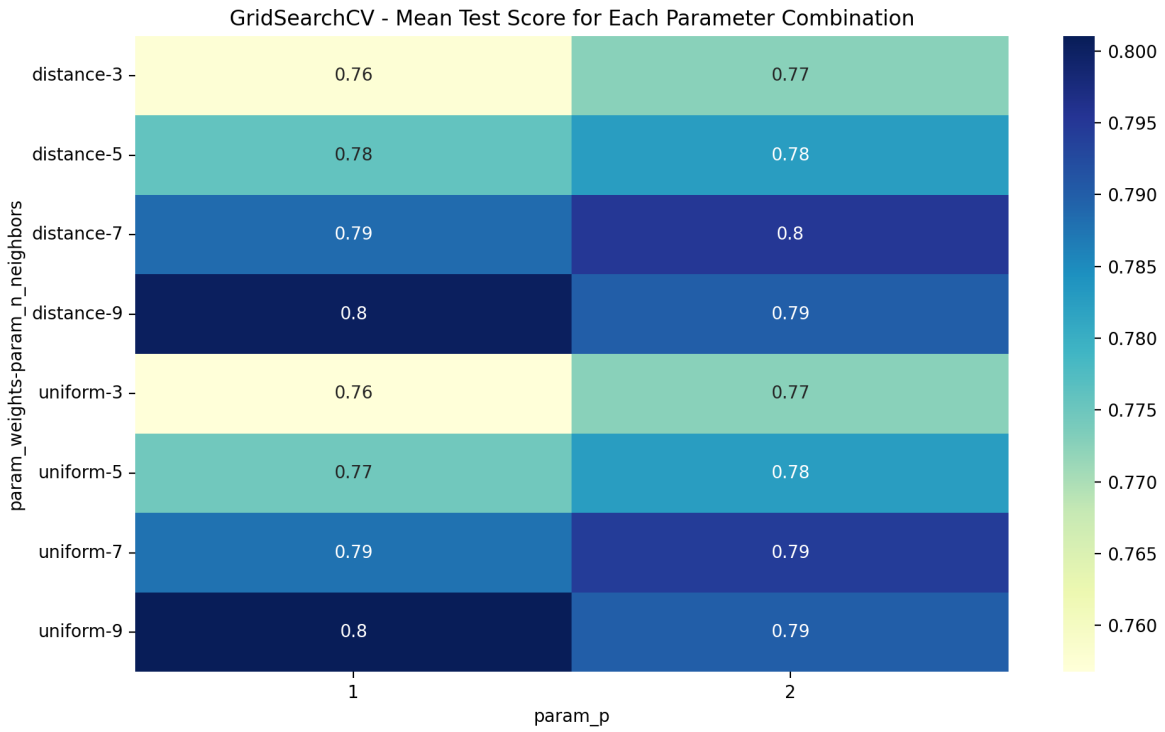
3.逻辑回归

```
Logistic Regression:
Best Parameters: {'C': 0.1, 'penalty': 'l1', 'solver': 'liblinear'}
Accuracy: 0.828042328042328
Confusion Matrix:
 [[146  43]
 [ 22 167]]
Classification Report:
              precision    recall  f1-score   support

         0.0       0.87      0.77      0.82       189
         1.0       0.80      0.88      0.84       189

    accuracy                           0.83       378
   macro avg       0.83      0.83      0.83       378
weighted avg       0.83      0.83      0.83       378
```



4.K-Nearest Neighbors

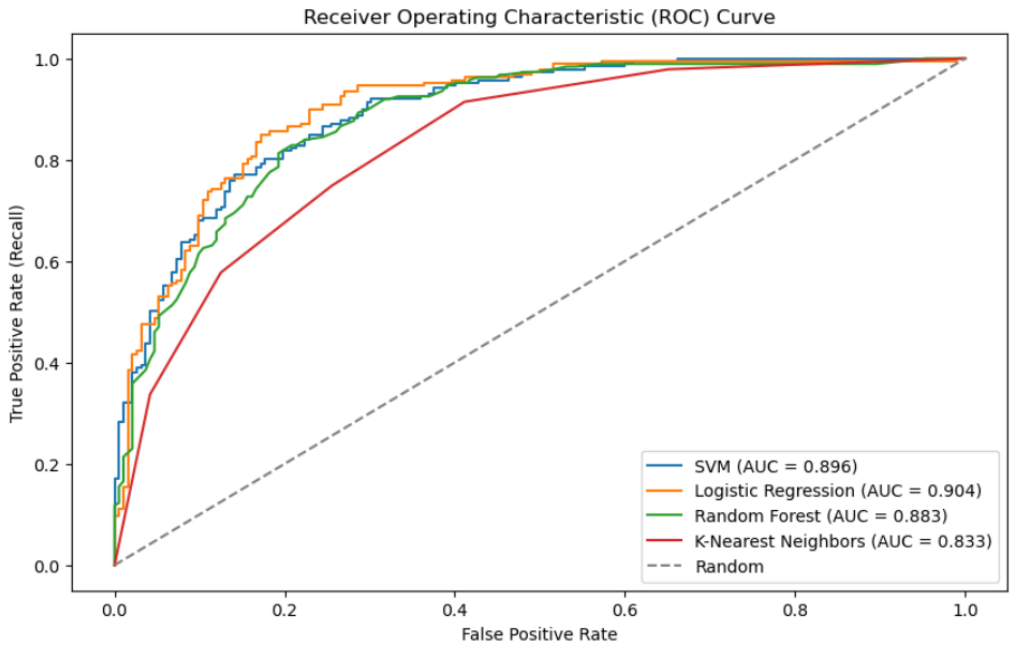
```
K-Nearest Neighbors Classifier:
Best Parameters: {'n_neighbors': 9, 'p': 1, 'weights': 'uniform'}
Accuracy: 0.7968337730870713
Confusion Matrix:
 [[145  47]
 [ 30 157]]
Classification Report:
              precision    recall  f1-score   support

         0.0       0.83      0.76      0.79       192
         1.0       0.77      0.84      0.80       187

    accuracy                           0.80       379
   macro avg       0.80      0.80      0.80       379
weighted avg       0.80      0.80      0.80       379
```



5.各模型数据横向对比

这是代码的部分，我采用了ROC曲线对各个模型的准确率进行了比较。

```
# 绘制 ROC 曲线
plt.figure(figsize=(10, 6))
plt.plot(fpr_svm, tpr_svm, label=f'SVM (AUC = {roc_auc_svm:.3f})')
plt.plot(fpr_logreg, tpr_logreg, label=f'Logistic Regression (AUC =
{roc_auc_logreg:.3f})')
plt.plot(fpr_rf, tpr_rf, label=f'Random Forest (AUC = {roc_auc_rf:.3f})')
plt.plot(fpr_knn, tpr_knn, label=f'K-Nearest Neighbors (AUC =
{roc_auc_knn:.3f})')

plt.plot([0, 1], [0, 1], linestyle='--', color='gray', label='Random')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate (Recall)')
plt.legend()
plt.show()
```



不难看出，各项模型的AUC指标都非常好，都接近于1。这说明我们拟合出的所有模型都能在测试集上很好地区分正负样本。而在这其中，我们又知道逻辑回归的区分度略优于其他模型（其AUC的值最高）。又由于在前四项模型的模拟中，SVM的准确率最高，**因此，我们在最后将会采取SVM的方式进行胜率预测。**

（二）模型预测

现在，我们使用刚刚的测试集和训练集对SVM进行训练，获得一个判断胜负率的模型。接下来，我们导入14个主要参赛国在近年来的平均能力数据，最后写一个输入两个国家名称的函数即可。具体实现如下：

```
import pandas as pd
from sklearn.svm import SVC
from sklearn.preprocessing import StandardScaler

def load_avg_matrix(filepath):
    # 读取平均值 csv 文件
    avg_matrix = pd.read_csv(filepath)
    # 将 'Country' 列设置为索引
    avg_matrix.set_index('Country', inplace=True)
```

```
    return avg_matrix

def predict_match_result(country1, country2, avg_matrix, model):
    # 获取两个国家的平均值特征
    country1_features = avg_matrix.loc[country1].values.reshape(1, -1)
    country2_features = avg_matrix.loc[country2].values.reshape(1, -1)

    # 使用模型进行预测
    country1_win_prob = model.predict_proba(country1_features)[:, 1][0]
    country2_win_prob = model.predict_proba(country2_features)[:, 1][0]

    return country1_win_prob, country2_win_prob

# 读取原始数据
filepath_original = 'Final_features.csv'
df_original = pd.read_csv(filepath_original)

# 处理缺失值
df_original = df_original.dropna(subset=['win_or_lose'])

# 选择特征和目标变量
features_original = df_original[['attack_scores', 'attack_errors',
'block_effective', 'block_errors',
                                 'dig_succeeded', 'dig_errors',
'reception_succeeded', 'reception_errors',
                                 'serve_scores', 'serve_errors']]
target_original = df_original['win_or_lose']

# 训练SVM模型
scaler_original = StandardScaler()
features_scaled_original = scaler_original.fit_transform(features_original)

svm_model_original = SVC(probability=True)
svm_model_original.fit(features_scaled_original, target_original)

# 读取平均值 csv 文件
filepath_avg_matrix = 'avg_values.csv'
avg_matrix = load_avg_matrix(filepath_avg_matrix)

# 输入两个国家的名字
country1_name = input("请输入第一个国家的名字：")
country2_name = input("请输入第二个国家的名字：")

# 预测比赛结果
country1_win_prob, country2_win_prob = predict_match_result(country1_name,
country2_name, avg_matrix, svm_model_original)

# 输出预测结果
print(f"{country1_name} 获胜概率: {country1_win_prob:.4f}")
print(f"{country2_name} 获胜概率: {country2_win_prob:.4f}")
```

需要注意的是，我在导入的过程中采用了基于特征矩阵给出的平均值矩阵，这就导致模型的预测总是让两个国家胜率相同。因此，如果我们想要获得正确的数据，就必须导入基于其他比赛获得的平均能力值数据。很可惜，历年奥运会的比赛数据较难爬取，因此我们只在这里留下一个框架。

这是结果的一个示例：

七、信息可视化