# -

基于Spark的1996-2022多赛季NBA球员数据分析

数据集描述代码

import pandas as pd

import numpy as np

country\_codes = pd.read\_csv(r"D:\BaiduNetdiskDownload\基于Spark的1996-2022多赛季NBA球员数据分析\数据集\all\_seasons\_processed.csv", index\_col=0)

country\_codes = pd.read\_csv("D:\BaiduNetdiskDownload\基于Spark的1996-2022多赛季NBA球员数据分析\数据集\all\_seasons\_processed.csv", index\_col=0)

df.head()

df.info()

df.describe()

categoricals = df.select\_dtypes(exclude=[np.number])

[朱俞恺1221004045.docx](https://github.com/user-attachments/files/21488263/1221004045.docx)

categor[朱俞恺1221004045.docx](https://github.com/user-attachments/files/21488258/1221004045.docx)

ical[朱俞恺1221004045.docx](https://github.com/user-attachments/files/21488180/1221004045.docx)

s.describe()

import pandas as pd

import numpy as np

# 假设 df 已加载

df = pd.read\_csv(r"D:\BaiduNetdiskDownload\基于Spark的1996-2022多赛季NBA球员数据分析\数据集\all\_seasons\_processed.csv")# 修复 draft\_year（处理 'Undrafted' 并转 datetime）

df['draft\_year'] = df['draft\_year'].replace(r'Undrafted', np.nan, regex=True)

df['draft\_year'] = pd.to\_datetime(df['draft\_year'])

# ✅ 修复 season 列（确保是字符串后再提取年份）

df['season'] = pd.to\_datetime(df['season'].astype(str).str[:4]) # 方法

# 处理国家/地区名称（统一命名）

df['country'] = df['country'].replace({

'Great Britain': 'United Kingdom',

'England': 'United Kingdom',

'Scotland': 'United Kingdom',

'Wales': 'United Kingdom',

'Republic of Ireland': 'Ireland',

'Northern Ireland': 'Ireland',

'Cabo Verde': 'Cape Verde',

'St. Vincent & Grenadines': 'Saint Vincent and Grenadines'

})

# 保存处理后的数据

df.to\_csv(r"D:\BaiduNetdiskDownload\基于Spark的1996-2022多赛季NBA球员数据分析\数据集\all\_seasons\_processed.csv")

1.启用Hadoop中的HDFS框架

/usr/local/hadoop/sbin/start-dfs.sh

2.登录用户创建目录及data子目录

/usr/local/hadoop/bin/hdfs dfs -mkdir -p /user/hadoop

/usr/local/hadoop/bin/hdfs dfs -mkdir -p data

language-shell

3.把本地文件系统中的数据集all\_season\_processed.csv上传到分布式文件系统中

cd nbaPlayerDataAnalysis

/usr/local/hadoop/bin/hdfs dfs -put ./D:\BaiduNetdiskDownload\基于Spark的1996-2022多赛季NBA球员数据分析\数据集\all\_seasons\_processed.csv

main.py文件

from pyspark.sql import SparkSession

from pyspark.sql.functions import avg, col, corr, count, sum, when

from functools import wraps

def save\_results(output\_path):

"""

Decorator to save DataFrame results to a specified JSON path.

"""

def decorator(func):

@wraps(func)

def wrapper(\*args, \*\*kwargs):

# Call the data processing function

df = func(\*args, \*\*kwargs)

# Save the DataFrame results as a single JSON file by coalescing.

df.write.json(output\_path, mode='overwrite')

return df

return wrapper

return decorator

def create\_spark\_session():

"""

Create a Spark session configured for using PySpark.

"""

return SparkSession.builder \

.appName("NBA Player Analysis") \

.getOrCreate()

def load\_data(spark, file\_path):

"""

Load data from HDFS into a Spark DataFrame.

"""

return spark.read.csv(file\_path, header=True, inferSchema=True)

@save\_results("json\_output/player\_performance\_by\_age")

def analyze\_player\_performance\_by\_age(df):

"""

Analyze how players' ages affect their average points, assists, and rebounds per game.

"""

# Select relevant columns and convert to proper types if necessary

result\_df = df.select(

col('age').cast('integer'),

col('pts').cast('float'),

col('ast').cast('float'),

col('reb').cast('float')

)

# Group by age and calculate average points, assists, and rebounds

age\_analysis = result\_df.groupBy('age').agg(

avg('pts').alias('avg\_points'),

avg('ast').alias('avg\_assists'),

avg('reb').alias('avg\_rebounds')

).orderBy('age')

return age\_analysis

@save\_results("json\_output/correlations\_over\_time")

def analyze\_correlations\_over\_time(df):

"""

Analyze the correlations between height, weight, average assists,

and average rebounds for each season.

"""

# Select necessary columns and cast to appropriate data types

correlations\_df = df.select(

col('season'),

col('player\_height').cast('float'),

col('player\_weight').cast('float'),

col('ast').cast('float').alias('average\_assists'),

col('reb').cast('float').alias('average\_rebounds')

)

# Calculate correlations for each season

correlation\_results = correlations\_df.groupBy('season').agg(

corr('player\_height', 'average\_rebounds').alias('height\_rebounds\_corr'),

corr('player\_height', 'average\_assists').alias('height\_assists\_corr'),

corr('player\_weight', 'average\_rebounds').alias('weight\_rebounds\_corr'),

corr('player\_weight', 'average\_assists').alias('weight\_assists\_corr')

).orderBy('season')

return correlation\_results

@save\_results("json\_output/performance\_by\_university")

def analyze\_performance\_by\_university(df):

"""

Analyze average scoring, assist, and rebound performance of players trained by different universities.

"""

# Select necessary columns and ensure proper data types

university\_performance\_df = df.select(

col('college').alias('university'),

col('pts').cast('float').alias('points'),

col('ast').cast('float').alias('assists'),

col('reb').cast('float').alias('rebounds')

)

# Group by university and calculate average points, assists, and rebounds

result\_df = university\_performance\_df.groupBy('university').agg(

avg('points').alias('avg\_points'),

avg('assists').alias('avg\_assists'),

avg('rebounds').alias('avg\_rebounds')

).orderBy('university')

return result\_df

@save\_results("json\_output/net\_rating\_by\_draft\_number")

def analyze\_net\_rating\_by\_draft\_number(df):

"""

Analyze the 'net rating' differences for different 'draft numbers'.Analyze the 'net rating' differences for different 'draft numbers'.

legend\_title='Legend',

template='plotly\_dark', # Using a dark theme for better visual contrast

)

# Save the figure as an HTML file

fig.write\_html('templates/non\_usa\_players\_proportion\_by\_season.html')

fig = px.scatter(df\_uni, x='university', y='avg\_points', color='avg\_rebounds',

size='avg\_assists', hover\_data=['university'],

title='Performance by University',

labels={'university': 'University', 'avg\_points': 'Average Points'})

# Save the figure as an HTML file

fig.write\_html('templates/performance\_by\_university.html')

fig = go.Figure()

# Add traces for average height and average age

fig.add\_trace(

go.Scatter(x=df\_season\_stats['season'], y=df\_season\_stats['avg\_height'], name='Average Height (cm)', yaxis='y1')

)

fig.add\_trace(

go.Scatter(x=df\_season\_stats['season'], y=df\_season\_stats['avg\_age'], name='Average Age (years)', yaxis='y2')

)

# Create axis objects

fig.update\_layout(

xaxis=dict(title='Season'),

yaxis=dict(title='Average Height (cm)', side='left',

range=[min(df\_season\_stats['avg\_height']) - 1, max(df\_season\_stats['avg\_height']) + 1]),

yaxis2=dict(title='Average Age (years)', overlaying='y', side='right',

range=[min(df\_season\_stats['avg\_age']) - 1, max(df\_season\_stats['avg\_age']) + 1]),

title="Average Height and Age of NBA Players by Season",

template='plotly\_dark'

)

# Save the figure as an HTML file

fig.write\_html('templates/average\_height\_age\_by\_season.html')

# 六、Web本地服务器部署

## 6.1 框架搭建

基于flask框架，可以将程序快速部署。

新建一个index.html作为默认界面，并存放在templates目录下，方便flask接口检测。

```html

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<meta http-equiv="X-UA-Compatible" content="ie=edge">

<title>NBA Players Data</title>

</head>

<body>

<h2>NBA players 1996-2022 多赛季数据分析</h2>

<ul style="line-height: 2em">

<li><a href="non\_usa\_players\_proportion\_by\_season.html">国际球员（非美国）占比变化统计图</a></li>

<li><a href="average\_height\_age\_by\_season.html">球员平均身高和年龄变化统计图</a></li>

<li><a href="correlations\_over\_time.html">球员身高和体重与平均篮板、平均助攻相关性统计图</a></li>

<li><a href="net\_rating\_by\_draft\_number.html">不同选秀顺位球员上场百回合得分统计图</a></li>

<li><a href="performance\_by\_university.html">不同大学出身球员平均得分、助攻和篮板表现</a></li>

<li><a href="player\_performance\_by\_age.html">球员场均得分、助攻和篮板随年龄变化统计图</a></li>

<li><a href="top\_colleges\_by\_stats.html">NBA总得分、总篮板和总助攻排名前50的大学</a></li>

</ul>

</body>

</html>

flask\_start.py编写如下:

from flask import Flask

from flask import render\_template

app = Flask(\_\_name\_\_)

@app.route('/')

def index():

# 使用 render\_template() 方法来渲染模板

return render\_template('index.html')

@app.route('/<filename>')

def req\_file(filename):

if \_\_name\_\_ == '\_\_main\_\_':

app.debug = False

app.run()