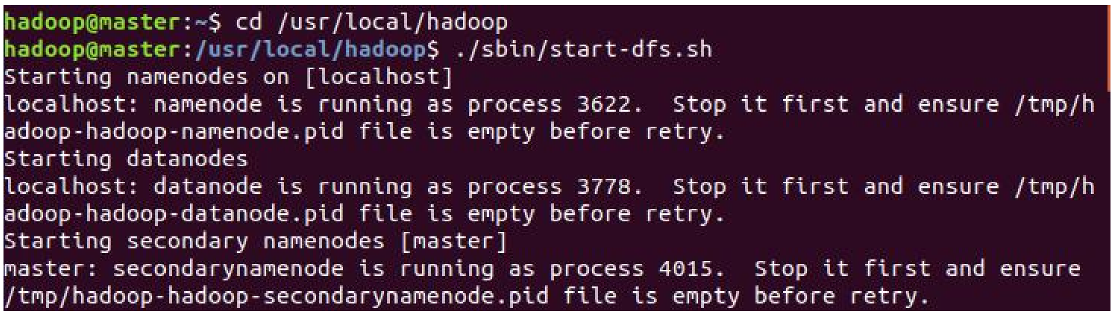
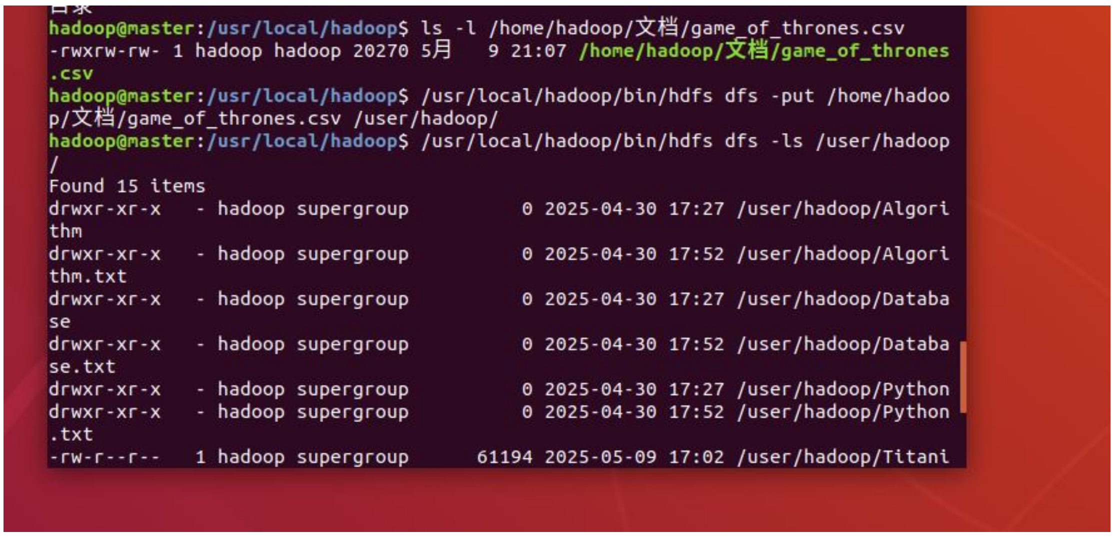
基于美剧《权力的游戏》剧集数据的 Spark 数据处理分析

1. 启动虚拟机并上传文件：

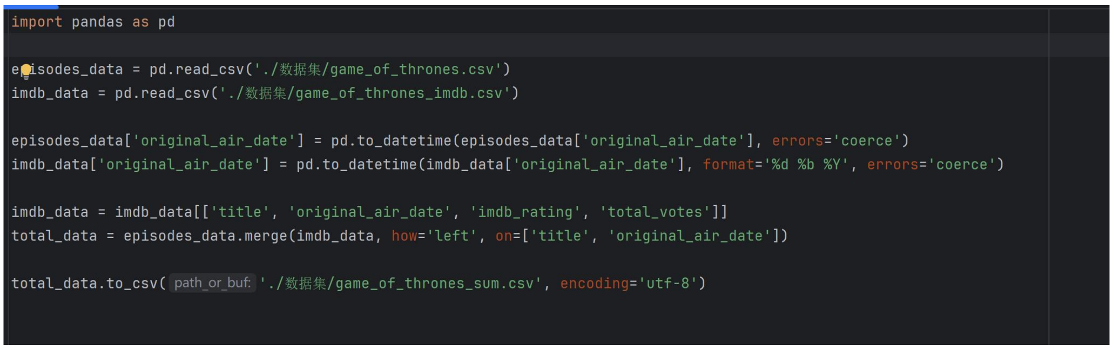


```
hadoop@master:~$ cd /usr/local/hadoop
hadoop@master:/usr/local/hadoop$ ./sbin/start-dfs.sh
Starting namenodes on [localhost]
localhost: namenode is running as process 3622.  Stop it first and ensure /tmp/h
adoop-hadoop-namenode.pid file is empty before retry.
Starting datanodes
localhost: datanode is running as process 3778.  Stop it first and ensure /tmp/h
adoop-hadoop-datanode.pid file is empty before retry.
Starting secondary namenodes [master]
master: secondarynamenode is running as process 4015.  Stop it first and ensure
/tmp/hadoop-hadoop-secondarynamenode.pid file is empty before retry.
```



```
hadoop@master:/usr/local/hadoop$ ls -l /home/hadoop/文档/game_of_thrones.csv
-rwxrw-rw- 1 hadoop hadoop 20270 5月   9 21:07 /home/hadoop/文档/game_of_thrones
.csv
hadoop@master:/usr/local/hadoop$ /usr/local/hadoop/bin/hdfs dfs -put /home/hadoo
p/文档/game_of_thrones.csv /user/hadoop/
hadoop@master:/usr/local/hadoop$ /usr/local/hadoop/bin/hdfs dfs -ls /user/hadoop
/
Found 15 items
drwxr-xr-x   - hadoop supergroup          0 2025-04-30 17:27 /user/hadoop/Algori
thm
drwxr-xr-x   - hadoop supergroup          0 2025-04-30 17:52 /user/hadoop/Algori
thm.txt
drwxr-xr-x   - hadoop supergroup          0 2025-04-30 17:27 /user/hadoop/Databa
se
drwxr-xr-x   - hadoop supergroup          0 2025-04-30 17:52 /user/hadoop/Databa
se.txt
drwxr-xr-x   - hadoop supergroup          0 2025-04-30 17:27 /user/hadoop/Python
drwxr-xr-x   - hadoop supergroup          0 2025-04-30 17:52 /user/hadoop/Python
.txt
-rw-r--r--   1 hadoop supergroup      61194 2025-05-09 17:02 /user/hadoop/Titani
```

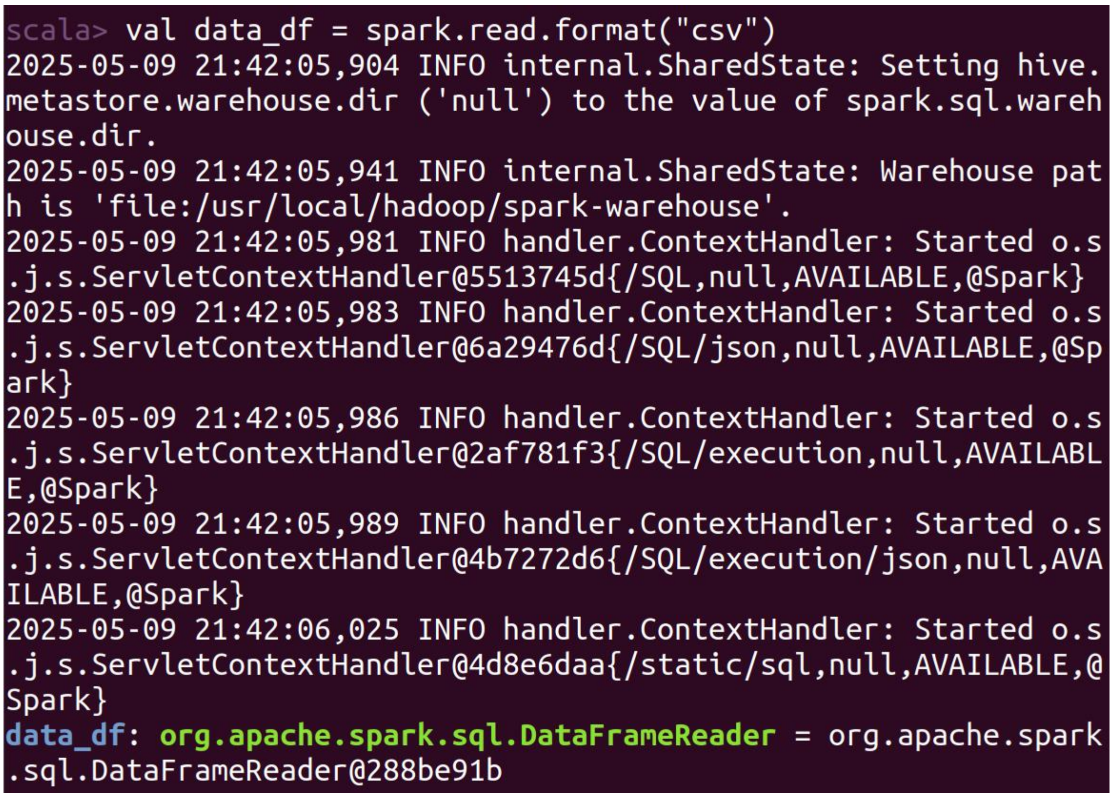
2. 数据预处理



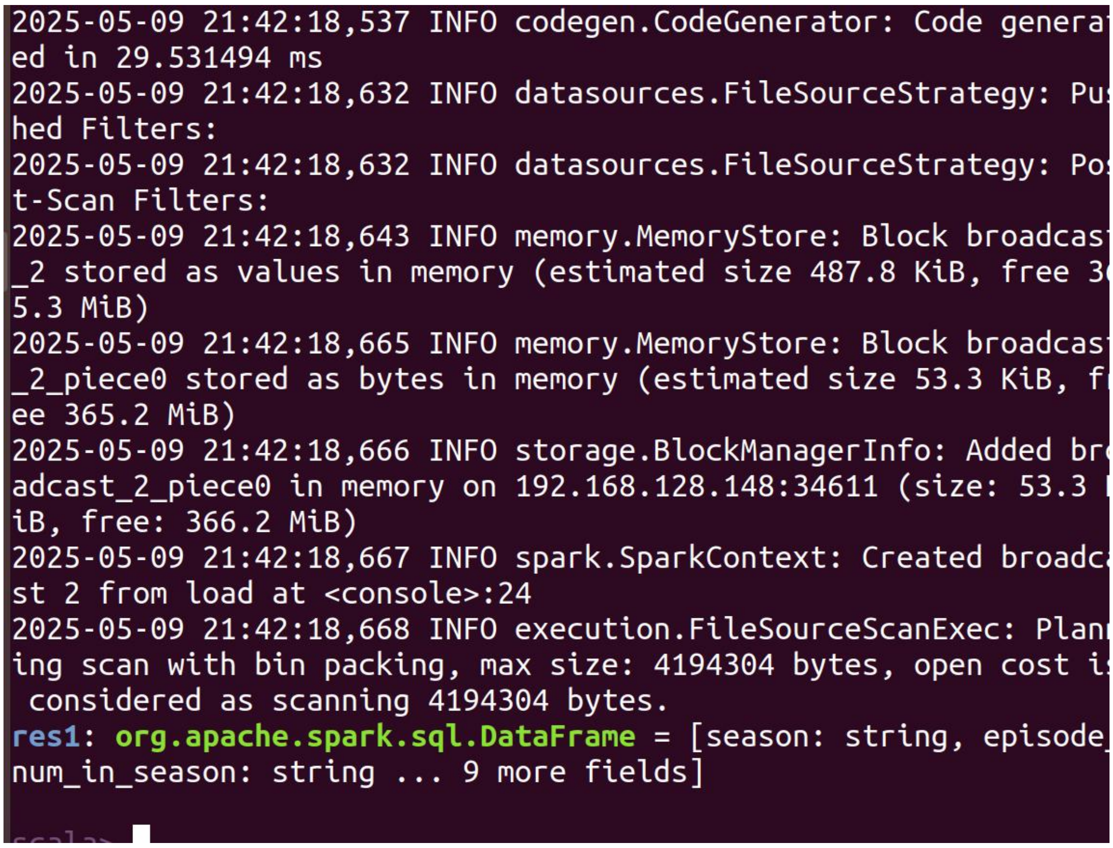
```python
import pandas as pd

episodes_data = pd.read_csv('./数据集/game_of_thrones.csv')
imdb_data = pd.read_csv('./数据集/game_of_thrones_imdb.csv')

episodes_data['original_air_date'] = pd.to_datetime(episodes_data['original_air_date'], errors='coerce')
imdb_data['original_air_date'] = pd.to_datetime(imdb_data['original_air_date'], format='%d %b %Y', errors='coerce')

imdb_data = imdb_data[['title', 'original_air_date', 'imdb_rating', 'total_votes']]
total_data = episodes_data.merge(imdb_data, how='left', on=['title', 'original_air_date'])

total_data.to_csv( path_or_buf: './数据集/game_of_thrones_sum.csv', encoding='utf-8')
```

3. Spark 数据分析

（1）读取 CSV 文件创建 DataFrame

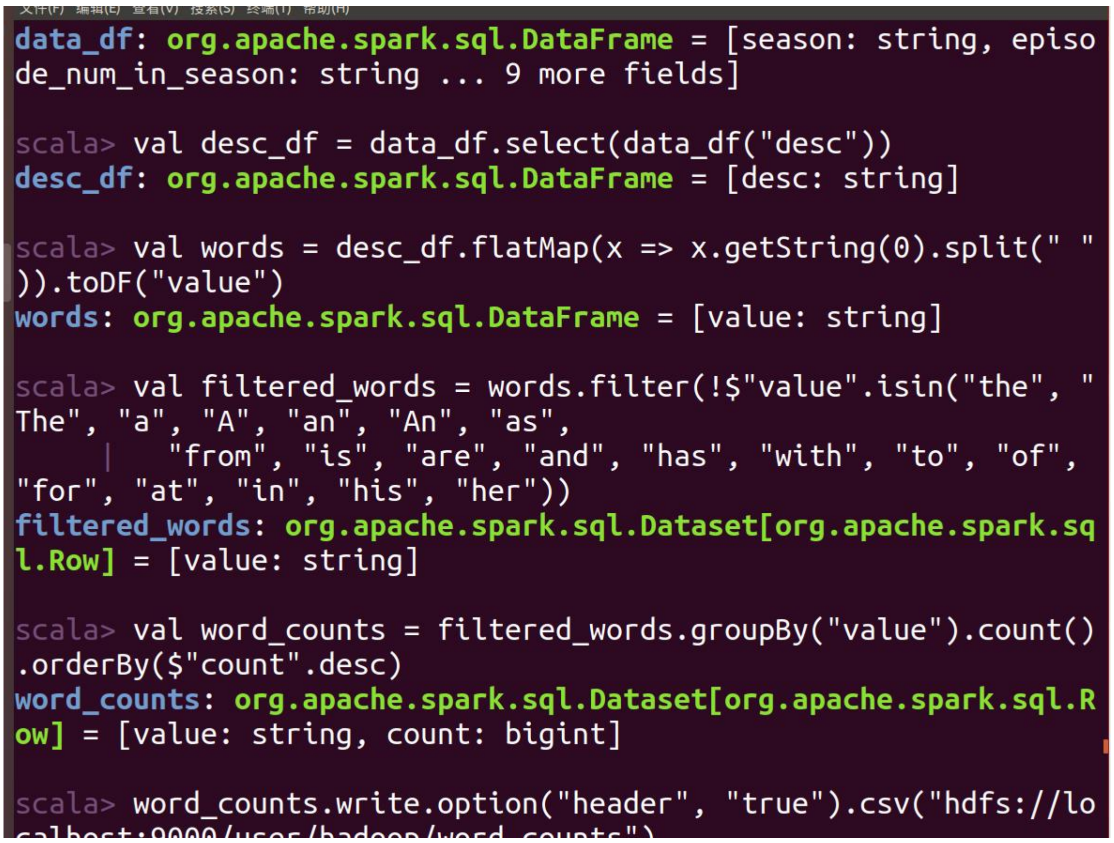


```
scala> val data_df = spark.read.format("csv")
2025-05-09 21:42:05,904 INFO internal.SharedState: Setting hive.
metastore.warehouse.dir ('null') to the value of spark.sql.wareh
ouse.dir.
2025-05-09 21:42:05,941 INFO internal.SharedState: Warehouse pat
h is 'file:/usr/local/hadoop/spark-warehouse'.
2025-05-09 21:42:05,981 INFO handler.ContextHandler: Started o.s
.j.s.ServletContextHandler@5513745d{/SQL,null,AVAILABLE,@Spark}
2025-05-09 21:42:05,983 INFO handler.ContextHandler: Started o.s
.j.s.ServletContextHandler@6a29476d{/SQL/json,null,AVAILABLE,@Sp
ark}
2025-05-09 21:42:05,986 INFO handler.ContextHandler: Started o.s
.j.s.ServletContextHandler@2af781f3{/SQL/execution,null,AVAILABL
E,@Spark}
2025-05-09 21:42:05,989 INFO handler.ContextHandler: Started o.s
.j.s.ServletContextHandler@4b7272d6{/SQL/execution/json,null,AVA
ILABLE,@Spark}
2025-05-09 21:42:06,025 INFO handler.ContextHandler: Started o.s
.j.s.ServletContextHandler@4d8e6daa{/static/sql,null,AVAILABLE,@
Spark}
data_df: org.apache.spark.sql.DataFrameReader = org.apache.spark
.sql.DataFrameReader@288be91b
```



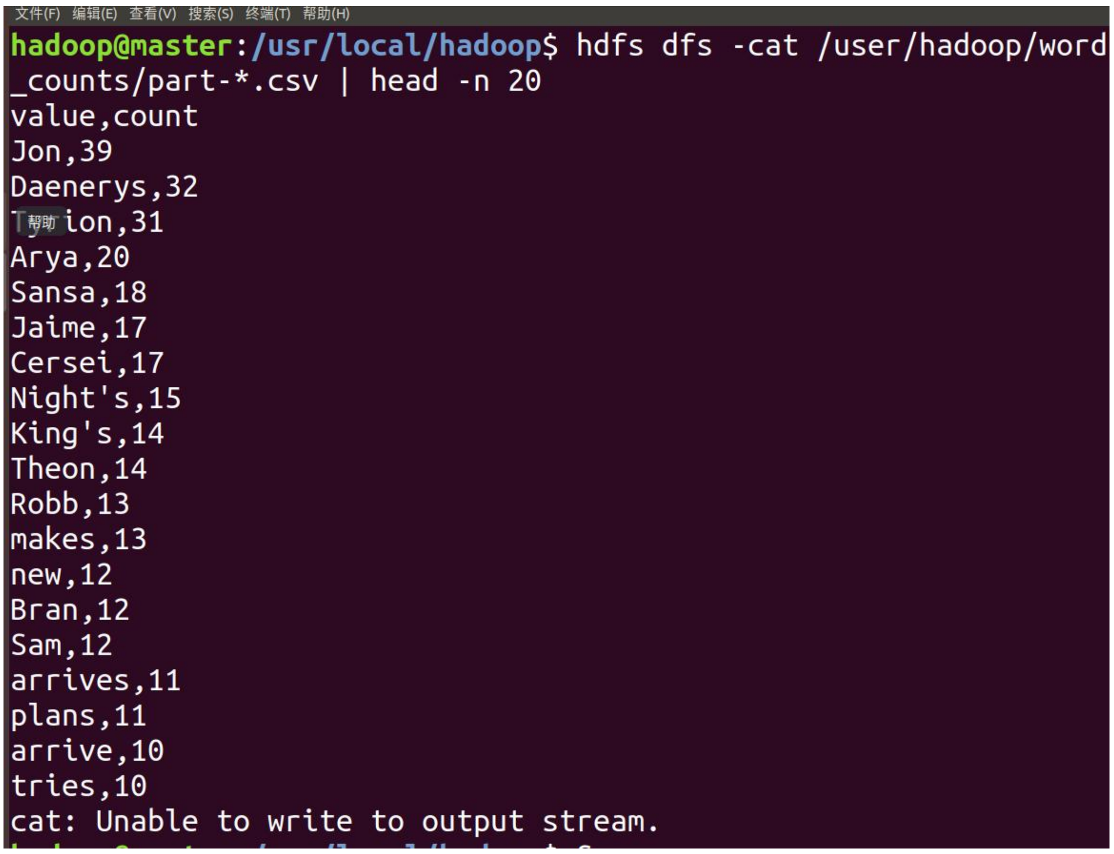
```
2025-05-09 21:42:18,537 INFO codegen.CodeGenerator: Code genera
ed in 29.531494 ms
2025-05-09 21:42:18,632 INFO datasources.FileSourceStrategy: Pu
hed Filters:
2025-05-09 21:42:18,632 INFO datasources.FileSourceStrategy: Po
t-Scan Filters:
2025-05-09 21:42:18,643 INFO memory.MemoryStore: Block broadcas
_2 stored as values in memory (estimated size 487.8 KiB, free 3
5.3 MiB)
2025-05-09 21:42:18,665 INFO memory.MemoryStore: Block broadcas
_2_piece0 stored as bytes in memory (estimated size 53.3 KiB, f
ee 365.2 MiB)
2025-05-09 21:42:18,666 INFO storage.BlockManagerInfo: Added br
adcast_2_piece0 in memory on 192.168.128.148:34611 (size: 53.3 
iB, free: 366.2 MiB)
2025-05-09 21:42:18,667 INFO spark.SparkContext: Created broadc
st 2 from load at <console>:24
2025-05-09 21:42:18,668 INFO execution.FileSourceScanExec: Plan
ing scan with bin packing, max size: 4194304 bytes, open cost i
 considered as scanning 4194304 bytes.
res1: org.apache.spark.sql.DataFrame = [season: string, episode
num_in_season: string ... 9 more fields]

scala> 
```

（2）对 desc 字段进行词频统计以及可视化



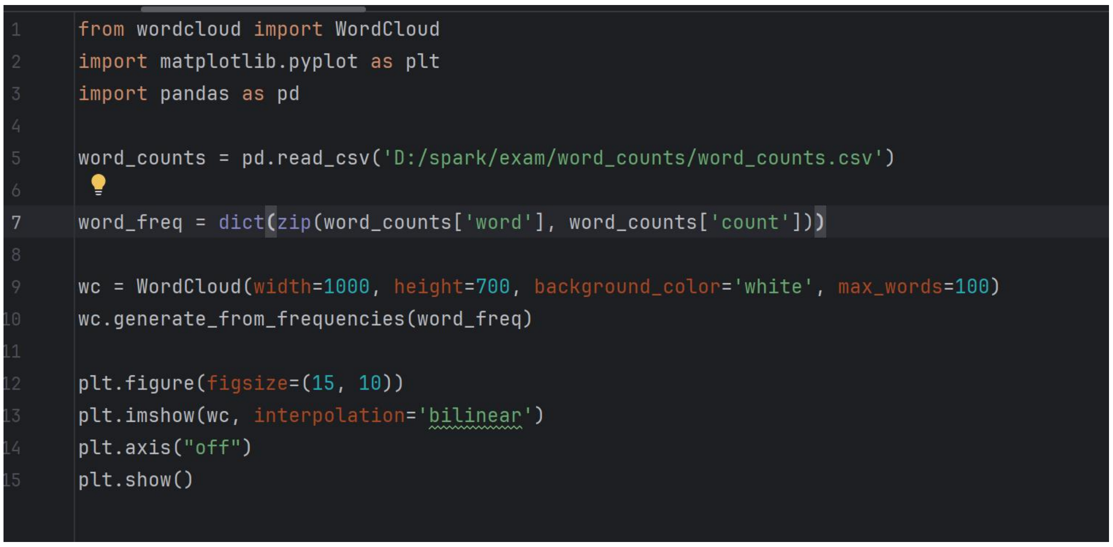
```
data_df: org.apache.spark.sql.DataFrame = [season: string, episo
de_num_in_season: string ... 9 more fields]

scala> val desc_df = data_df.select(data_df("desc"))
desc_df: org.apache.spark.sql.DataFrame = [desc: string]

scala> val words = desc_df.flatMap(x => x.getString(0).split(" "
)).toDF("value")
words: org.apache.spark.sql.DataFrame = [value: string]

scala> val filtered_words = words.filter(!$"value".isin("the", "
The", "a", "A", "an", "An", "as",
     |   "from", "is", "are", "and", "has", "with", "to", "of",
"for", "at", "in", "his", "her"))
filtered_words: org.apache.spark.sql.Dataset[org.apache.spark.sq
l.Row] = [value: string]

scala> val word_counts = filtered_words.groupBy("value").count()
.orderBy($"count".desc)
word_counts: org.apache.spark.sql.Dataset[org.apache.spark.sql.R
ow] = [value: string, count: bigint]

scala> word_counts.write.option("header", "true").csv("hdfs://lo
calhost:9000/user/hadoop/word_counts")
```

部分数据：

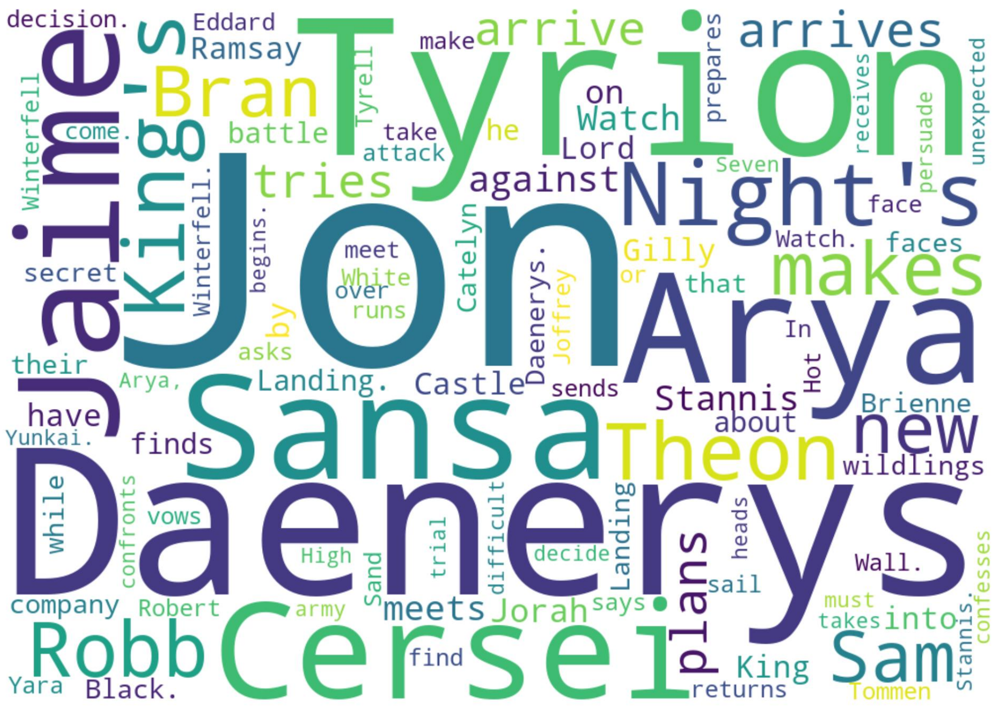


```
hadoop@master:/usr/local/hadoop$ hdfs dfs -cat /user/hadoop/word
_counts/part-*.csv | head -n 20
value,count
Jon,39
Daenerys,32
T帮助ion,31
Arya,20
Sansa,18
Jaime,17
Cersei,17
Night's,15
King's,14
Theon,14
Robb,13
makes,13
new,12
Bran,12
Sam,12
arrives,11
plans,11
arrive,10
tries,10
cat: Unable to write to output stream.
```

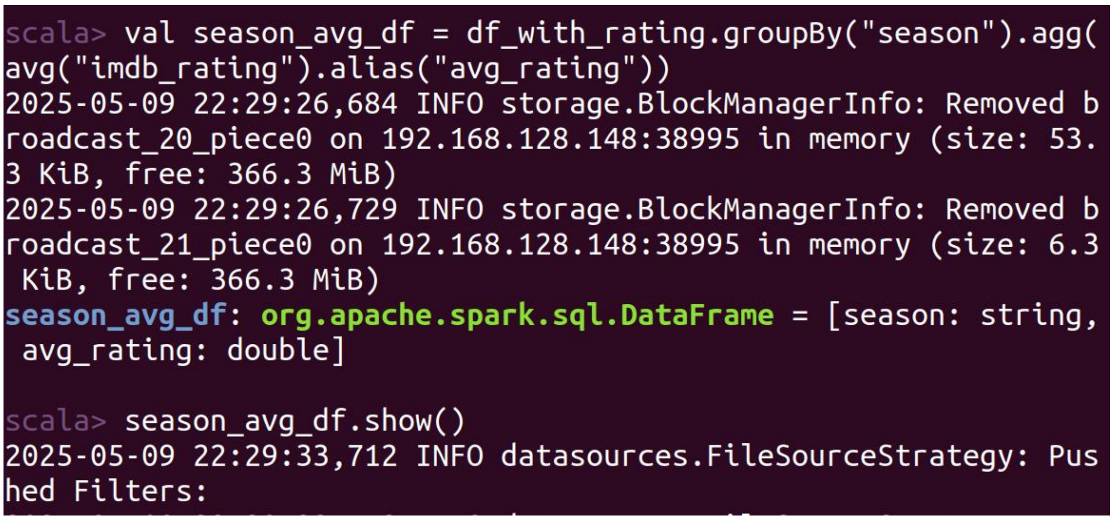
可视化：



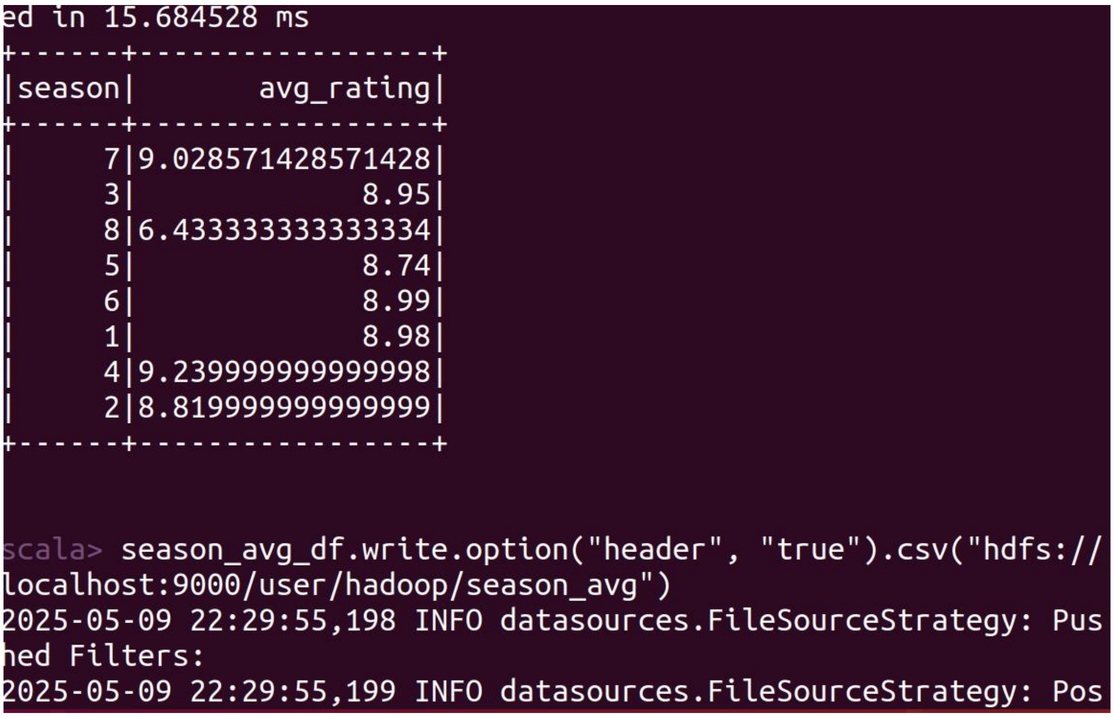
```python
from wordcloud import WordCloud
import matplotlib.pyplot as plt
import pandas as pd

word_counts = pd.read_csv('D:/spark/exam/word_counts/word_counts.csv')

word_freq = dict(zip(word_counts['word'], word_counts['count']))

wc = WordCloud(width=1000, height=700, background_color='white', max_words=100)
wc.generate_from_frequencies(word_freq)

plt.figure(figsize=(15, 10))
plt.imshow(wc, interpolation='bilinear')
plt.axis("off")
plt.show()
```



3）每季观看人数、IMDb 评分还有参与评分人数的分析

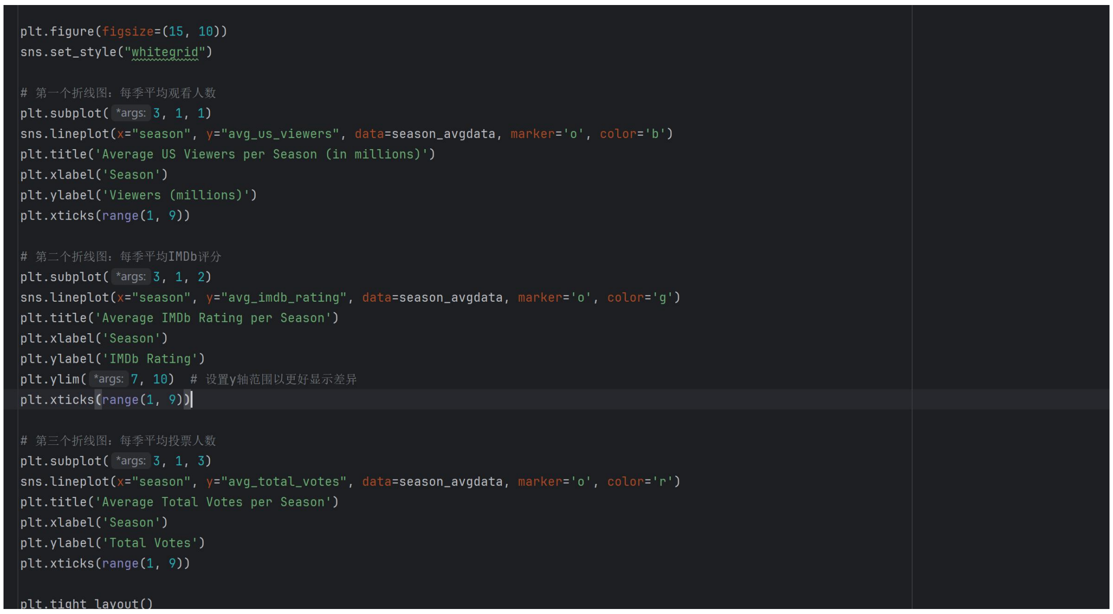


```
scala> val season_avg_df = df_with_rating.groupBy("season").agg(
avg("imdb_rating").alias("avg_rating"))
2025-05-09 22:29:26,684 INFO storage.BlockManagerInfo: Removed b
roadcast_20_piece0 on 192.168.128.148:38995 in memory (size: 53.
3 KiB, free: 366.3 MiB)
2025-05-09 22:29:26,729 INFO storage.BlockManagerInfo: Removed b
roadcast_21_piece0 on 192.168.128.148:38995 in memory (size: 6.3
 KiB, free: 366.3 MiB)
season_avg_df: org.apache.spark.sql.DataFrame = [season: string,
 avg_rating: double]

scala> season_avg_df.show()
2025-05-09 22:29:33,712 INFO datasources.FileSourceStrategy: Pus
hed Filters:
```

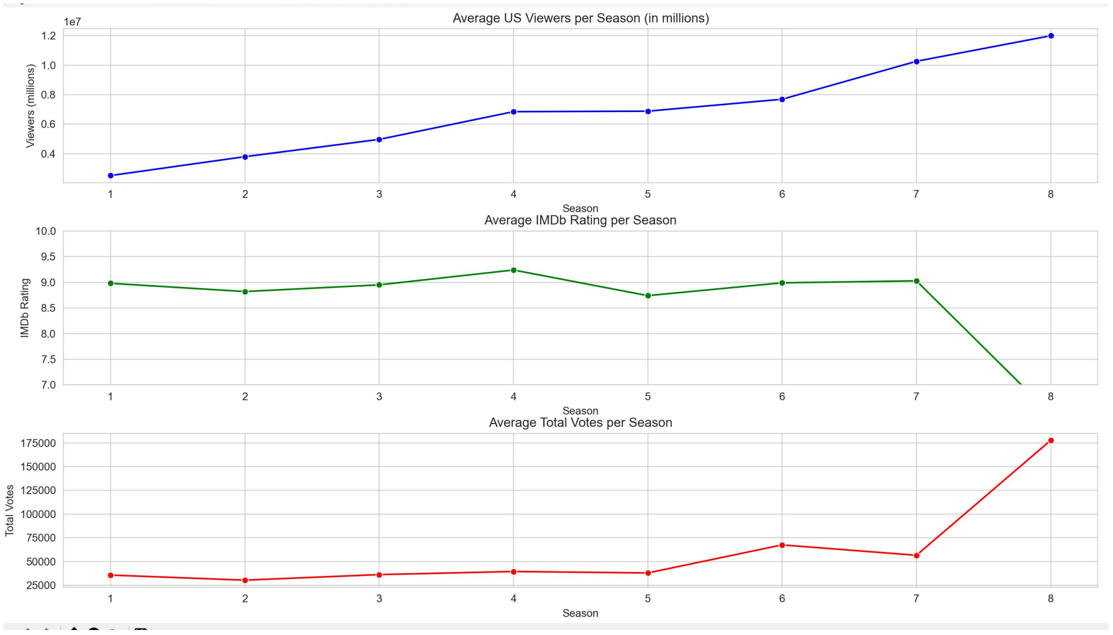


```
ed in 15.684528 ms
+------+-----------------+
|season|       avg_rating|
+------+-----------------+
|     7|9.028571428571428|
|     3|             8.95|
|     8|6.433333333333334|
|     5|             8.74|
|     6|             8.99|
|     1|             8.98|
|     4|9.239999999999998|
|     2|8.819999999999999|
+------+-----------------+


scala> season_avg_df.write.option("header", "true").csv("hdfs://
localhost:9000/user/hadoop/season_avg")
2025-05-09 22:29:55,198 INFO datasources.FileSourceStrategy: Pus
hed Filters:
2025-05-09 22:29:55,199 INFO datasources.FileSourceStrategy: Pos
```

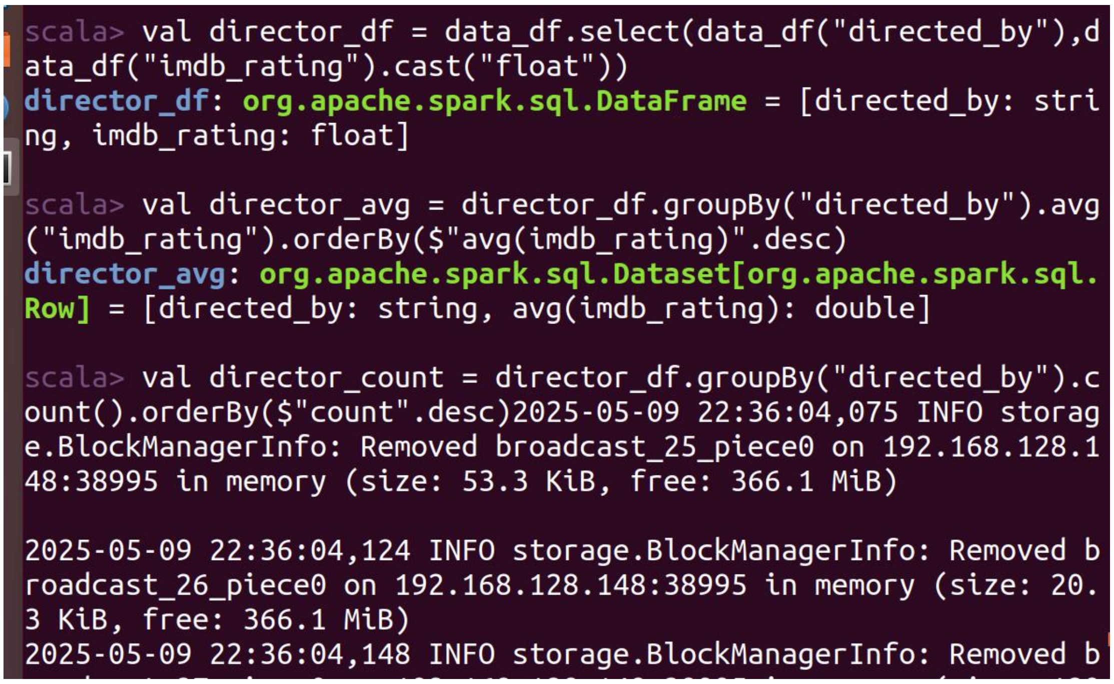
可视化：



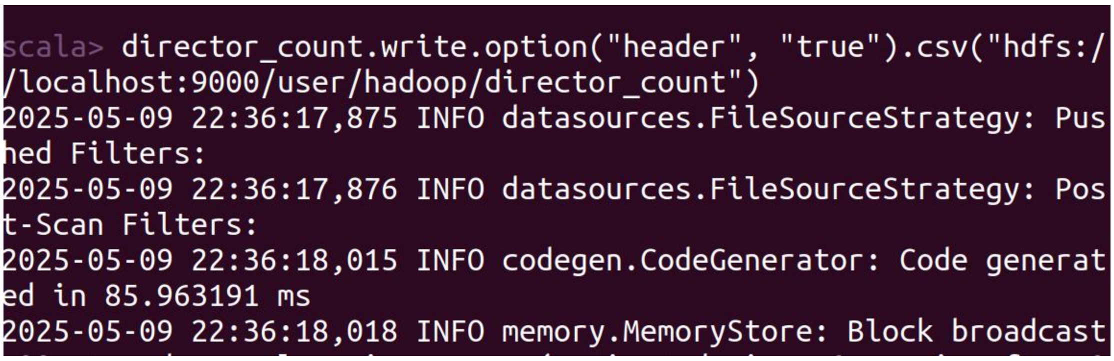
```
plt.figure(figsize=(15, 10))
sns.set_style("whitegrid")

# 第一个折线图：每季平均观看人数
plt.subplot( *args: 3, 1, 1)
sns.lineplot(x="season", y="avg_us_viewers", data=season_avgdata, marker='o', color='b')
plt.title('Average US Viewers per Season (in millions)')
plt.xlabel('Season')
plt.ylabel('Viewers (millions)')
plt.xticks(range(1, 9))

# 第二个折线图：每季平均IMDb评分
plt.subplot( *args: 3, 1, 2)
sns.lineplot(x="season", y="avg_imdb_rating", data=season_avgdata, marker='o', color='g')
plt.title('Average IMDb Rating per Season')
plt.xlabel('Season')
plt.ylabel('IMDb Rating')
plt.ylim( *args: 7, 10)  # 设置y轴范围以更好显示差异
plt.xticks(range(1, 9))

# 第三个折线图：每季平均投票人数
plt.subplot( *args: 3, 1, 3)
sns.lineplot(x="season", y="avg_total_votes", data=season_avgdata, marker='o', color='r')
plt.title('Average Total Votes per Season')
plt.xlabel('Season')
plt.ylabel('Total Votes')
plt.xticks(range(1, 9))

plt.tight_layout()
```



4）不同导演与评分关系分析

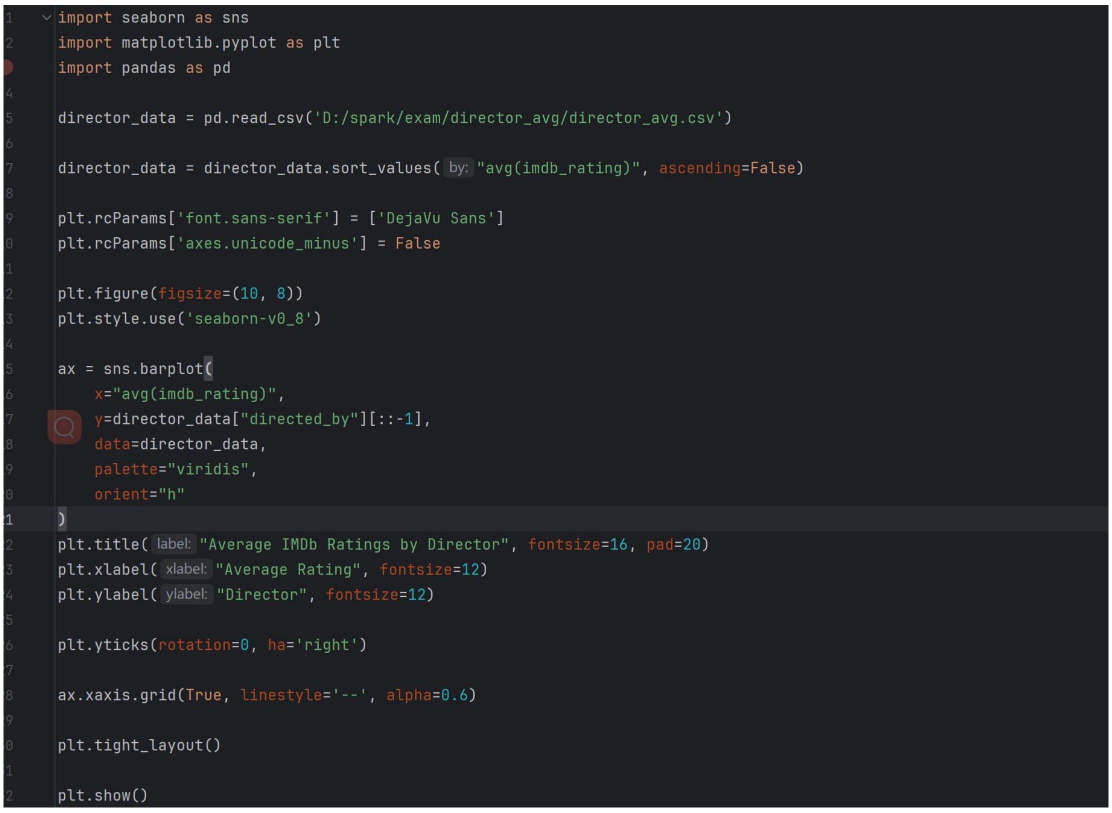


```
scala> val director_df = data_df.select(data_df("directed_by"),d
ata_df("imdb_rating").cast("float"))
director_df: org.apache.spark.sql.DataFrame = [directed_by: stri
ng, imdb_rating: float]

scala> val director_avg = director_df.groupBy("directed_by").avg
("imdb_rating").orderBy($"avg(imdb_rating)".desc)
director_avg: org.apache.spark.sql.Dataset[org.apache.spark.sql.
Row] = [directed_by: string, avg(imdb_rating): double]

scala> val director_count = director_df.groupBy("directed_by").c
ount().orderBy($"count".desc)2025-05-09 22:36:04,075 INFO storag
e.BlockManagerInfo: Removed broadcast_25_piece0 on 192.168.128.1
48:38995 in memory (size: 53.3 KiB, free: 366.1 MiB)

2025-05-09 22:36:04,124 INFO storage.BlockManagerInfo: Removed b
roadcast_26_piece0 on 192.168.128.148:38995 in memory (size: 20.
3 KiB, free: 366.1 MiB)
2025-05-09 22:36:04,148 INFO storage.BlockManagerInfo: Removed b
```

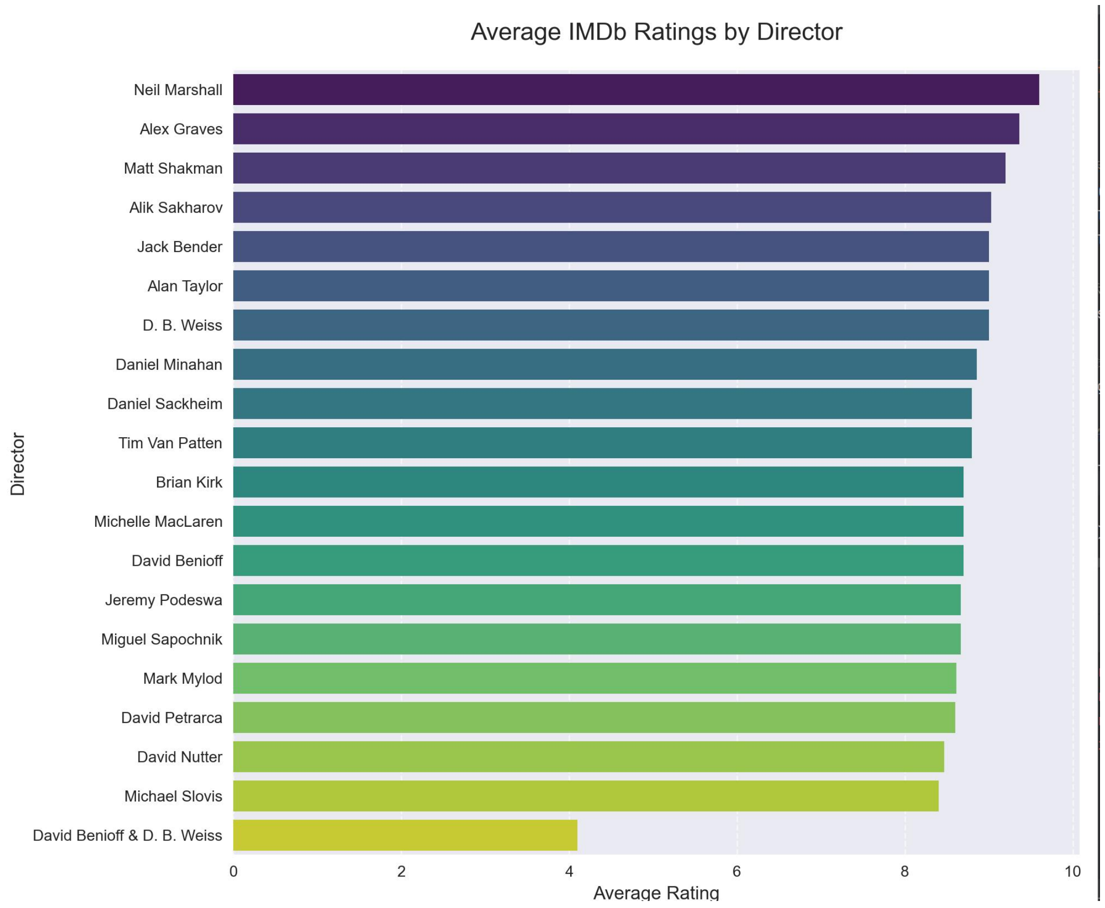


```
scala> director_count.write.option("header", "true").csv("hdfs:/
/localhost:9000/user/hadoop/director_count")
2025-05-09 22:36:17,875 INFO datasources.FileSourceStrategy: Pus
hed Filters:
2025-05-09 22:36:17,876 INFO datasources.FileSourceStrategy: Pos
t-Scan Filters:
2025-05-09 22:36:18,015 INFO codegen.CodeGenerator: Code generat
ed in 85.963191 ms
2025-05-09 22:36:18,018 INFO memory.MemoryStore: Block broadcast
```

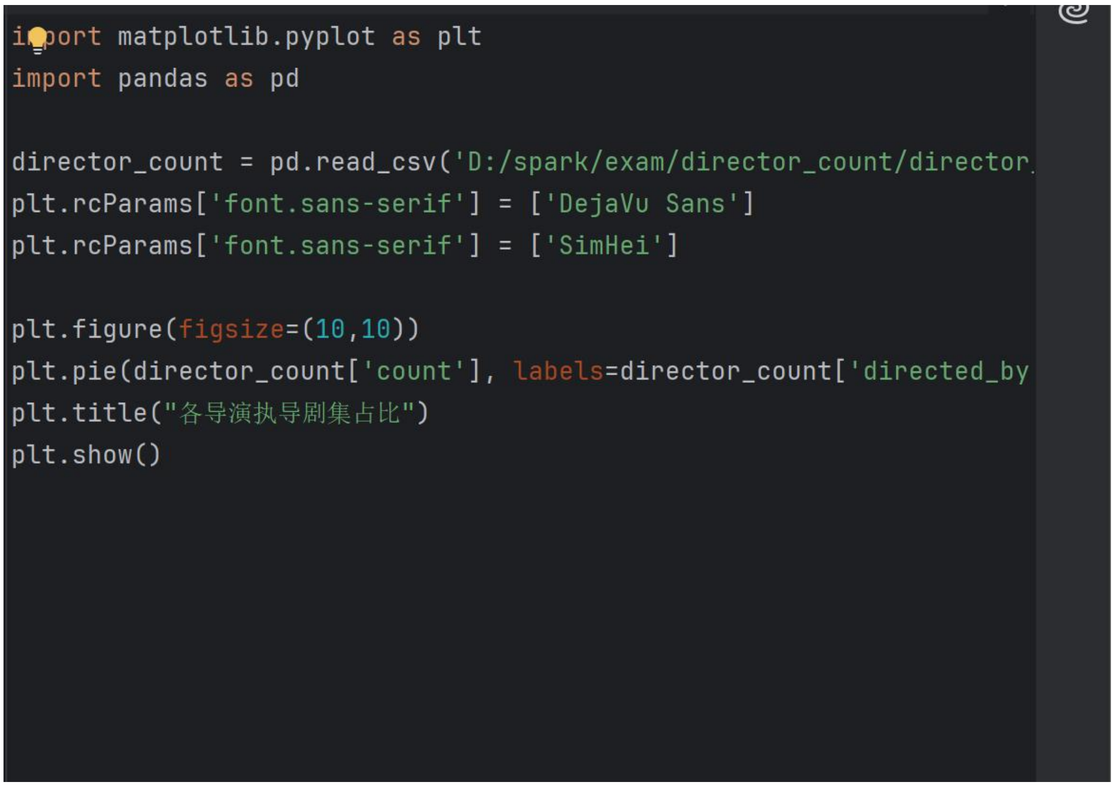
可视化



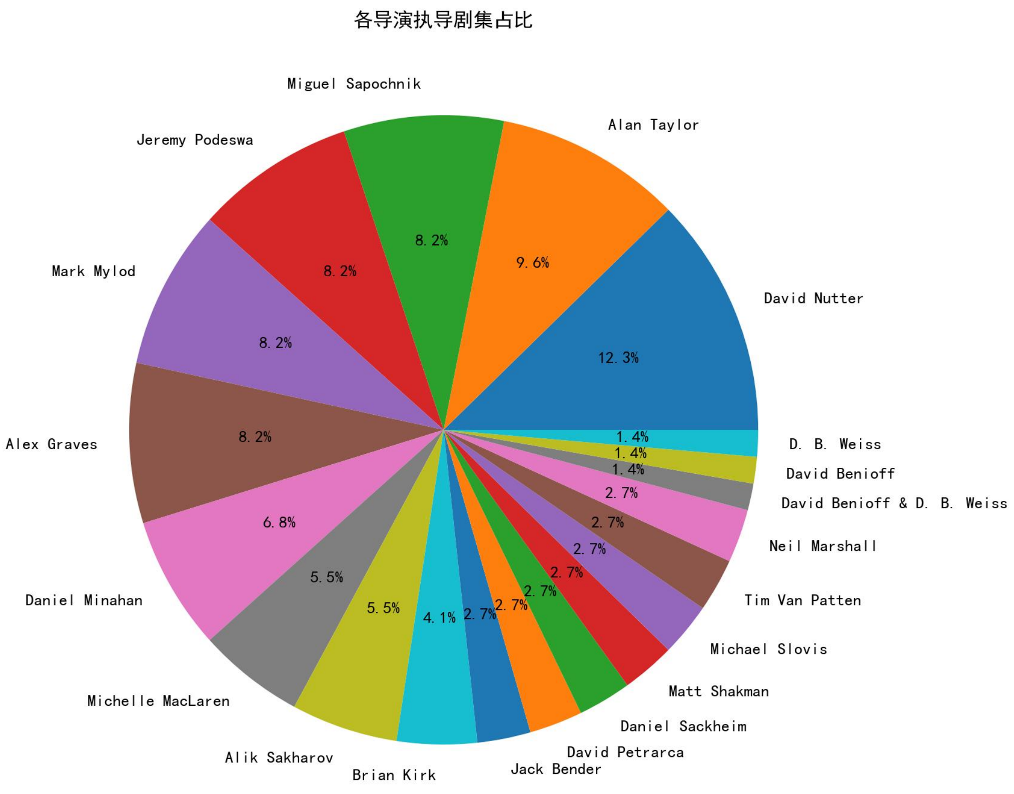
```python
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd

director_data = pd.read_csv('D:/spark/exam/director_avg/director_avg.csv')

director_data = director_data.sort_values( by: "avg(imdb_rating)", ascending=False)

plt.rcParams['font.sans-serif'] = ['DejaVu Sans']
plt.rcParams['axes.unicode_minus'] = False

plt.figure(figsize=(10, 8))
plt.style.use('seaborn-v0_8')

ax = sns.barplot(
    x="avg(imdb_rating)",
    y=director_data["directed_by"][::-1],
    data=director_data,
    palette="viridis",
    orient="h"
)
plt.title( label: "Average IMDb Ratings by Director", fontsize=16, pad=20)
plt.xlabel( xlabel: "Average Rating", fontsize=12)
plt.ylabel( ylabel: "Director", fontsize=12)

plt.yticks(rotation=0, ha='right')

ax.xaxis.grid(True, linestyle='--', alpha=0.6)

plt.tight_layout()

plt.show()
```



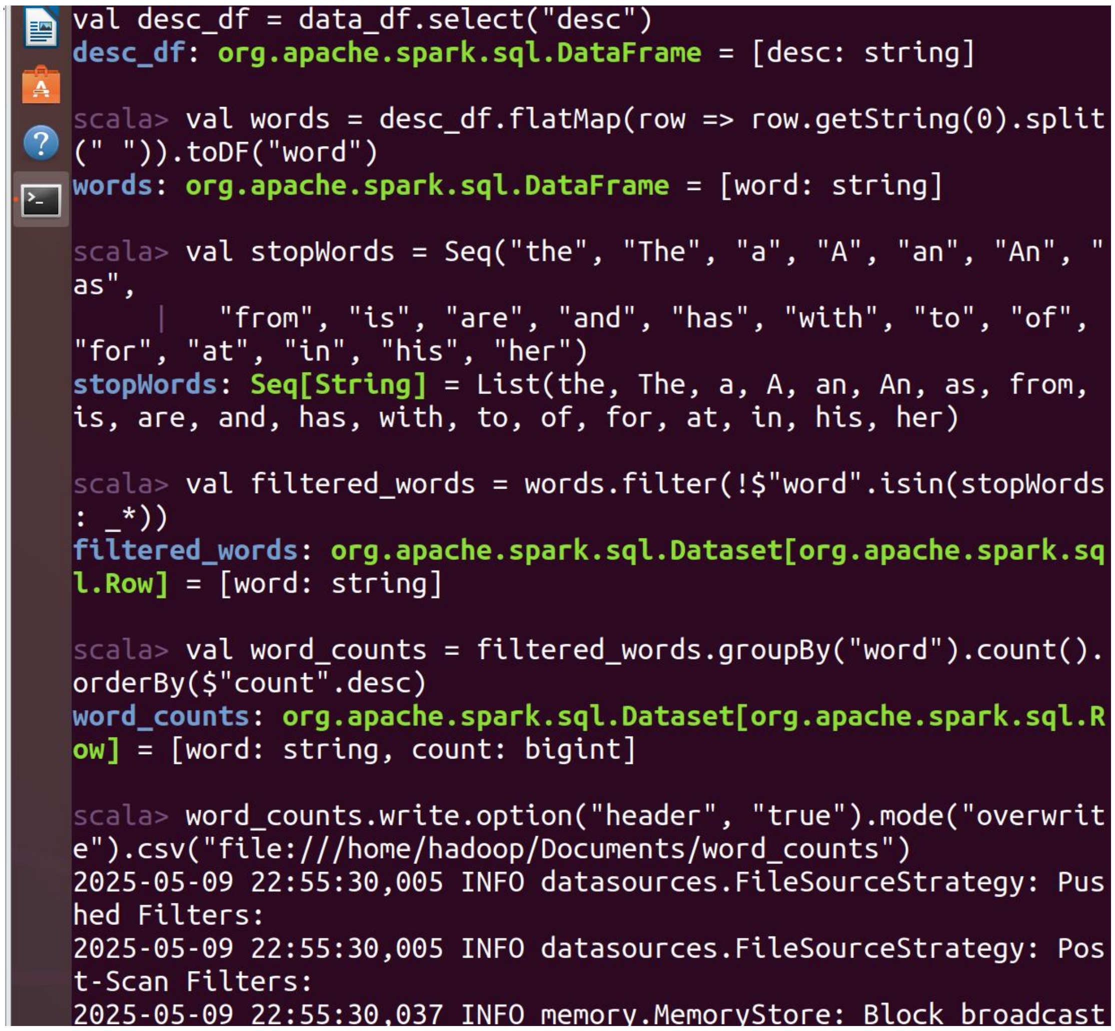
导演所导演的剧集占比



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

director_count = pd.read_csv('D:/spark/exam/director_count/director
plt.rcParams['font.sans-serif'] = ['DejaVu Sans']
plt.rcParams['font.sans-serif'] = ['SimHei']

plt.figure(figsize=(10,10))
plt.pie(director_count['count'], labels=director_count['directed_by
plt.title("各导演执导剧集占比")
plt.show()
```

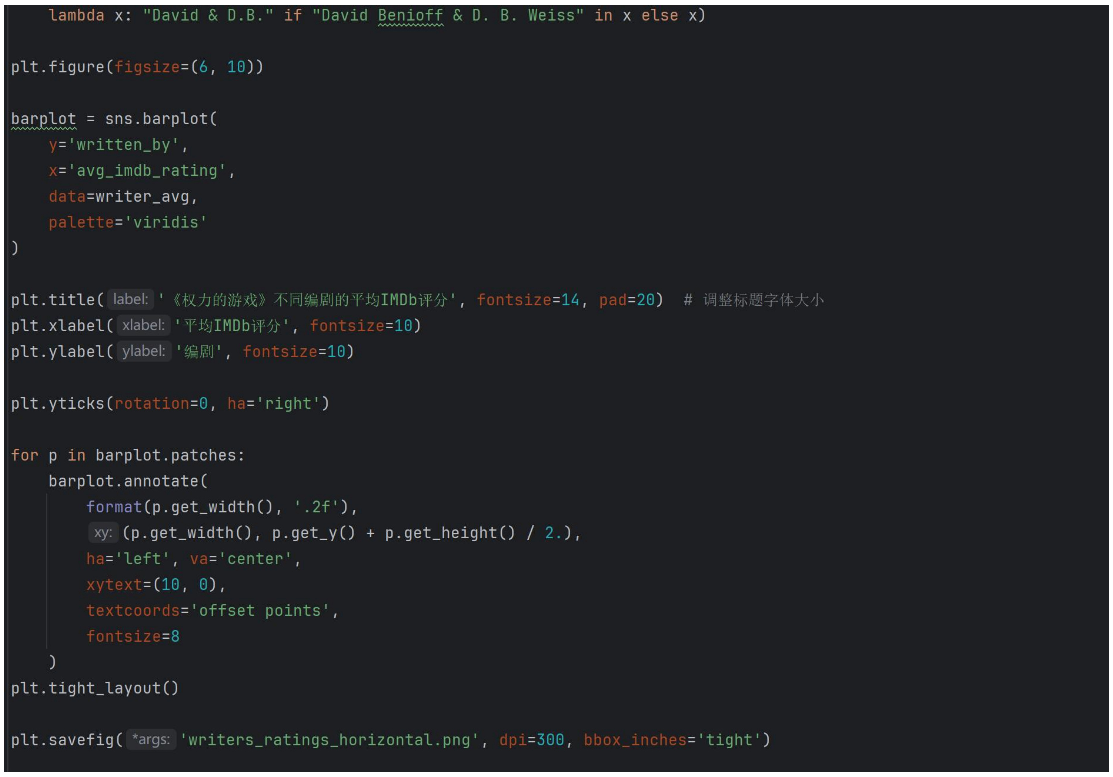


5）不同作者与评分关系

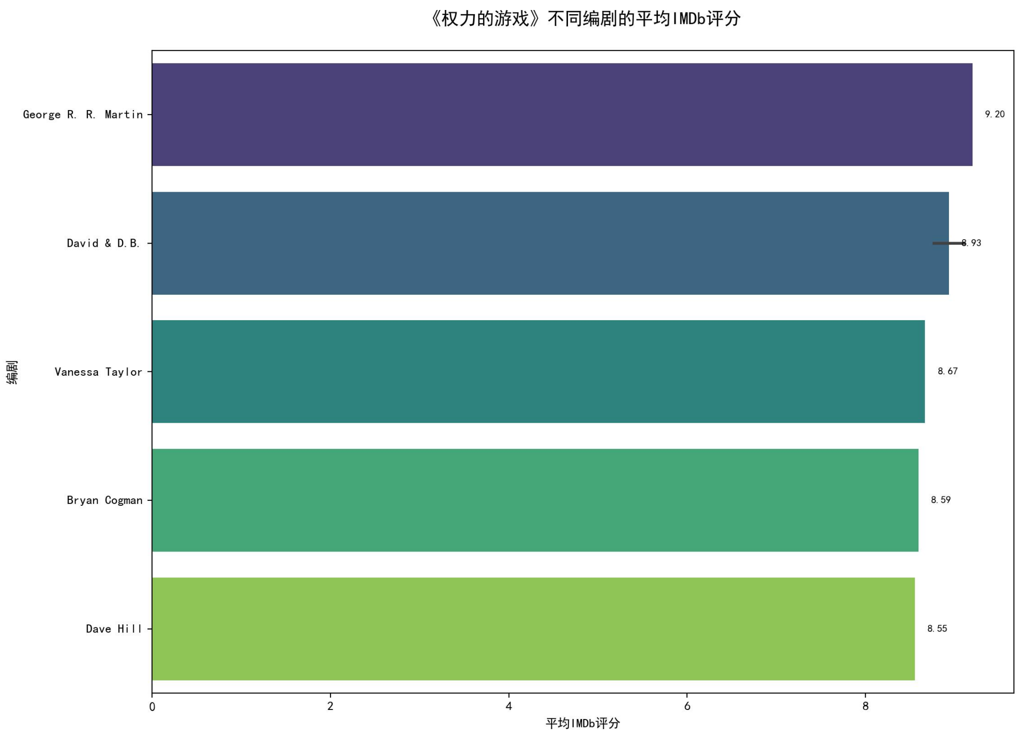


```
val desc_df = data_df.select("desc")
desc_df: org.apache.spark.sql.DataFrame = [desc: string]

scala> val words = desc_df.flatMap(row => row.getString(0).split
(" ")).toDF("word")
words: org.apache.spark.sql.DataFrame = [word: string]

scala> val stopWords = Seq("the", "The", "a", "A", "an", "An", "
as",
     |   "from", "is", "are", "and", "has", "with", "to", "of",
"for", "at", "in", "his", "her")
stopWords: Seq[String] = List(the, The, a, A, an, An, as, from,
is, are, and, has, with, to, of, for, at, in, his, her)

scala> val filtered_words = words.filter(!$"word".isin(stopWords
: _*))
filtered_words: org.apache.spark.sql.Dataset[org.apache.spark.sq
l.Row] = [word: string]

scala> val word_counts = filtered_words.groupBy("word").count().
orderBy($"count".desc)
word_counts: org.apache.spark.sql.Dataset[org.apache.spark.sql.R
ow] = [word: string, count: bigint]

scala> word_counts.write.option("header", "true").mode("overwrit
e").csv("file:///home/hadoop/Documents/word_counts")
2025-05-09 22:55:30,005 INFO datasources.FileSourceStrategy: Pus
hed Filters:
2025-05-09 22:55:30,005 INFO datasources.FileSourceStrategy: Pos
t-Scan Filters:
2025-05-09 22:55:30,037 INFO memory.MemoryStore: Block broadcast
```

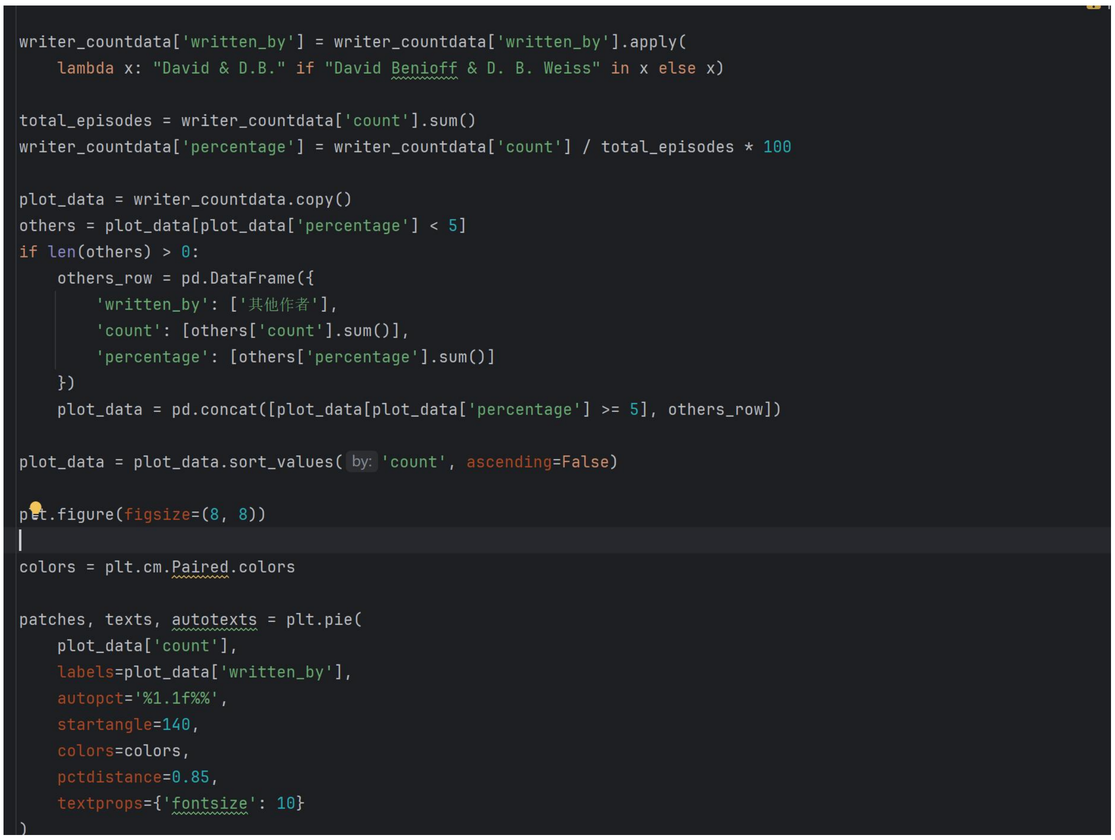
可视化



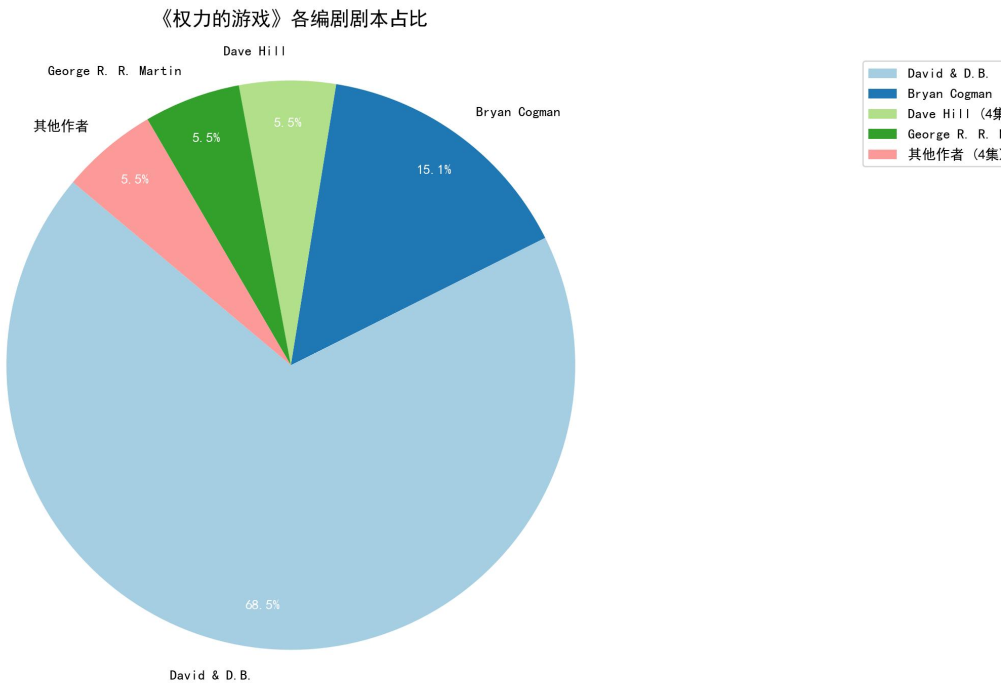
```
    lambda x: "David & D.B." if "David Benioff & D. B. Weiss" in x else x)

plt.figure(figsize=(6, 10))

barplot = sns.barplot(
    y='written_by',
    x='avg_imdb_rating',
    data=writer_avg,
    palette='viridis'
)

plt.title( label: '《权力的游戏》不同编剧的平均IMDb评分', fontsize=14, pad=20)  # 调整标题字体大小
plt.xlabel( xlabel: '平均IMDb评分', fontsize=10)
plt.ylabel( ylabel: '编剧', fontsize=10)

plt.yticks(rotation=0, ha='right')

for p in barplot.patches:
    barplot.annotate(
        format(p.get_width(), '.2f'),
        xy: (p.get_width(), p.get_y() + p.get_height() / 2.),
        ha='left', va='center',
        xytext=(10, 0),
        textcoords='offset points',
        fontsize=8
    )
plt.tight_layout()

plt.savefig( *args: 'writers_ratings_horizontal.png', dpi=300, bbox_inches='tight')
```



每位作者所写作的剧本在整个剧集中占了多大比重

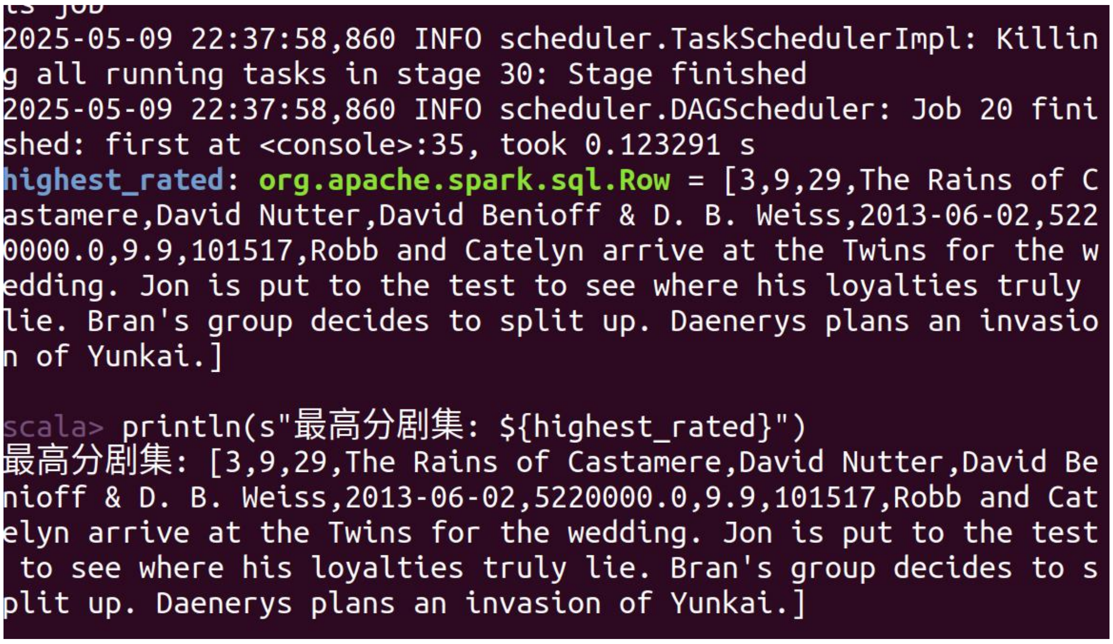


```
writer_countdata['written_by'] = writer_countdata['written_by'].apply(
    lambda x: "David & D.B." if "David Benioff & D. B. Weiss" in x else x)

total_episodes = writer_countdata['count'].sum()
writer_countdata['percentage'] = writer_countdata['count'] / total_episodes * 100

plot_data = writer_countdata.copy()
others = plot_data[plot_data['percentage'] < 5]
if len(others) > 0:
    others_row = pd.DataFrame({
        'written_by': ['其他作者'],
        'count': [others['count'].sum()],
        'percentage': [others['percentage'].sum()]
    })
    plot_data = pd.concat([plot_data[plot_data['percentage'] >= 5], others_row])

plot_data = plot_data.sort_values( by: 'count', ascending=False)

plt.figure(figsize=(8, 8))

colors = plt.cm.Paired.colors

patches, texts, autotexts = plt.pie(
    plot_data['count'],
    labels=plot_data['written_by'],
    autopct='%1.1f%%',
    startangle=140,
    colors=colors,
    pctdistance=0.85,
    textprops={'fontsize': 10}
)
```



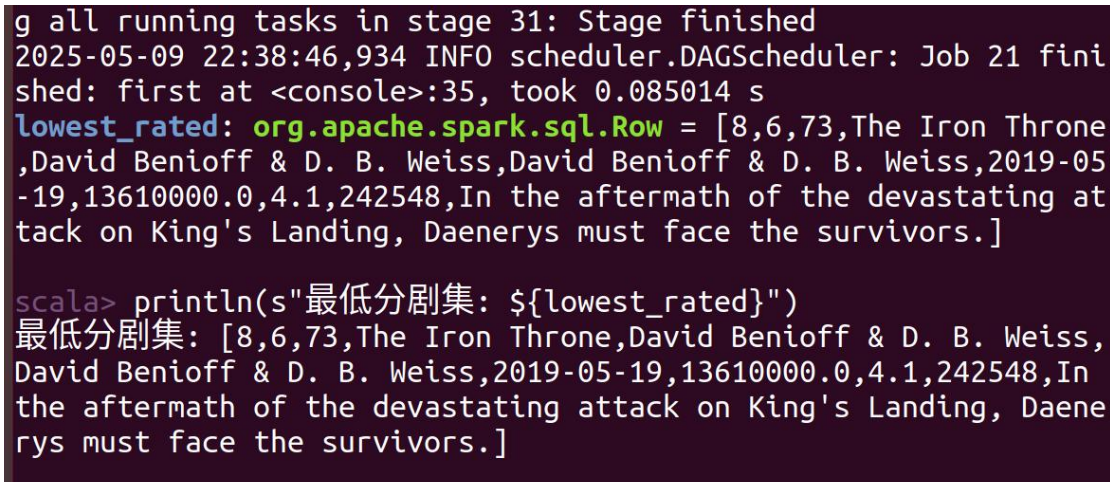
6）整部剧评分最高的剧集和评分最低的剧集

评分最高



```
ts job
2025-05-09 22:37:58,860 INFO scheduler.TaskSchedulerImpl: Killin
g all running tasks in stage 30: Stage finished
2025-05-09 22:37:58,860 INFO scheduler.DAGScheduler: Job 20 fini
shed: first at <console>:35, took 0.123291 s
highest_rated: org.apache.spark.sql.Row = [3,9,29,The Rains of C
astamere,David Nutter,David Benioff & D. B. Weiss,2013-06-02,522
0000.0,9.9,101517,Robb and Catelyn arrive at the Twins for the w
edding. Jon is put to the test to see where his loyalties truly
lie. Bran's group decides to split up. Daenerys plans an invasio
n of Yunkai.]

scala> println(s"最高分剧集: ${highest_rated}")
最高分剧集: [3,9,29,The Rains of Castamere,David Nutter,David Be
nioff & D. B. Weiss,2013-06-02,5220000.0,9.9,101517,Robb and Cat
elyn arrive at the Twins for the wedding. Jon is put to the test
 to see where his loyalties truly lie. Bran's group decides to s
plit up. Daenerys plans an invasion of Yunkai.]
```

评分最低：



```
g all running tasks in stage 31: Stage finished
2025-05-09 22:38:46,934 INFO scheduler.DAGScheduler: Job 21 fini
shed: first at <console>:35, took 0.085014 s
lowest_rated: org.apache.spark.sql.Row = [8,6,73,The Iron Throne
,David Benioff & D. B. Weiss,David Benioff & D. B. Weiss,2019-05
-19,13610000.0,4.1,242548,In the aftermath of the devastating at
tack on King's Landing, Daenerys must face the survivors.]

scala> println(s"最低分剧集: ${lowest_rated}")
最低分剧集: [8,6,73,The Iron Throne,David Benioff & D. B. Weiss,
David Benioff & D. B. Weiss,2019-05-19,13610000.0,4.1,242548,In
the aftermath of the devastating attack on King's Landing, Daene
rys must face the survivors.]
```