**Spark课程设计**

**系统设计说明书**

项目名称： 基于《权力的游戏》的spark数据分析

专业班级： 大数据2101班

成员分工： 石子阳（50%） 康宏扬（50%）

指导教师： 张健

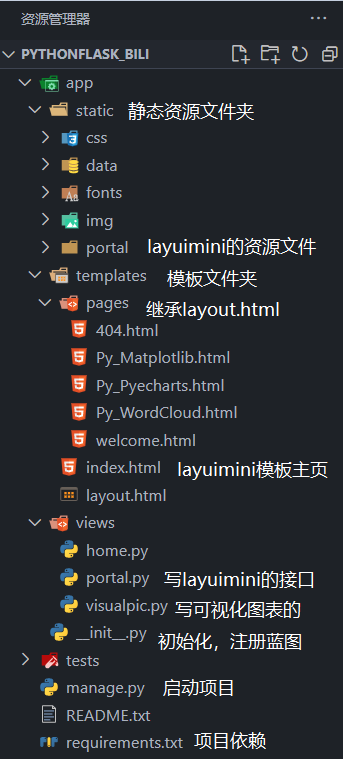
提交时间： 2024.5.29

基于《权力的游戏》的spark数据分析

本项目基于Flask框架，结合LayuiMini前端框架、Flask蓝图和Jinja模板继承，实现了一个功能强大且易于维护的数据分析展示系统。该系统主要用于数据展示和图表可视化，是一个典型的Dashboard应用。

# 系统概述

本系统的总体布局如下图所示。



## 1.1 系统架构

（1）前端：使用LayuiMini模板，实现响应式布局和用户交互。

响应式设计：通过LayuiMini实现自适应布局，确保在不同设备上（如PC、平板和手机）都能有良好的用户体验。

丰富的组件：LayuiMini提供了丰富的UI组件，如表格、表单、按钮、导航栏等，便于快速构建页面。

高度可定制：支持自定义主题和样式，能够根据项目需求灵活调整界面设计。

（2）后端：基于Flask框架，使用蓝图分离视图，Jinja模板继承实现页面的模板化。

模块化设计：通过Flask蓝图将不同功能模块分离，使代码结构更加清晰，便于管理和维护。

模板继承：使用Jinja模板引擎，定义基础模板和可复用的组件，减少重复代码，提高开发效率。

## 1.2 技术选型

（1）Flask：轻量级Web框架，适合快速开发。Flask提供了简洁而强大的核心功能，便于快速构建Web应用。扩展性强，通过丰富的扩展（如Flask-SQLAlchemy、Flask-Migrate等），可以方便地添加所需的功能模块。

（2）LayuiMini：基于layui的前端框架，提供丰富的UI组件。LayuiMini的预构建组件和模板，使得前端开发更加高效，减少手动编写样式和布局的时间。

用户体验佳，过精美的设计和交互效果，提高用户体验，增强系统的易用性。

（3）Jinja强大的模板引擎，支持模板继承和扩展。通过定义基础模板和块，子模板可以继承并扩展这些块，简化页面开发和维护。灵活性，Jinja模板引擎支持复杂的逻辑和表达式，能够灵活生成动态内容。

# 2 详细设计

1. manage.py

功能：项目的入口文件，负责启动Flask应用。

实现：调用应用工厂函数create\_app创建Flask应用实例。

使用app.run()运行项目。

1. app/\_\_init\_\_.py

功能：包含应用工厂函数，初始化Flask应用。

实现：定义create\_app函数，创建并配置Flask应用实例。告诉Python app文件夹应当视作为一个包。

3、 app/static

1) css/: 导入字体和自定义样式文件。

功能：提供项目所需的样式和字体文件，确保前端页面的一致性和美观性。

示例：自定义的CSS文件，如style.css

2) data/：皴法可视化图表所需的文件。

功能：存放生成词云所需的文本资源，如停用词列表和用户词典文件以及其他用于绘图的csv文件等。

示例：desc.txt、director\_countdata.csv。

3） fonts：得意黑字体

4） img/：生成词云需要的遮罩图。

5） portal/：LayuiMini相关资源文件。

功能：存放LayuiMini框架的静态资源文件，如CSS和JS。

示例：layuimini.css、layuimini.js。

1. app/templates/layout.html

功能：定义基础“骨架”模板，包含可被子模板继承的块。

1. app/views
2. home.py

功能：定义根路径、404页面和其他静态页面的路由。

1. portal.py

功能：LayuiMini初始化接口。

1. visualpic.py

功能：提供多个展示页面，实时生成图

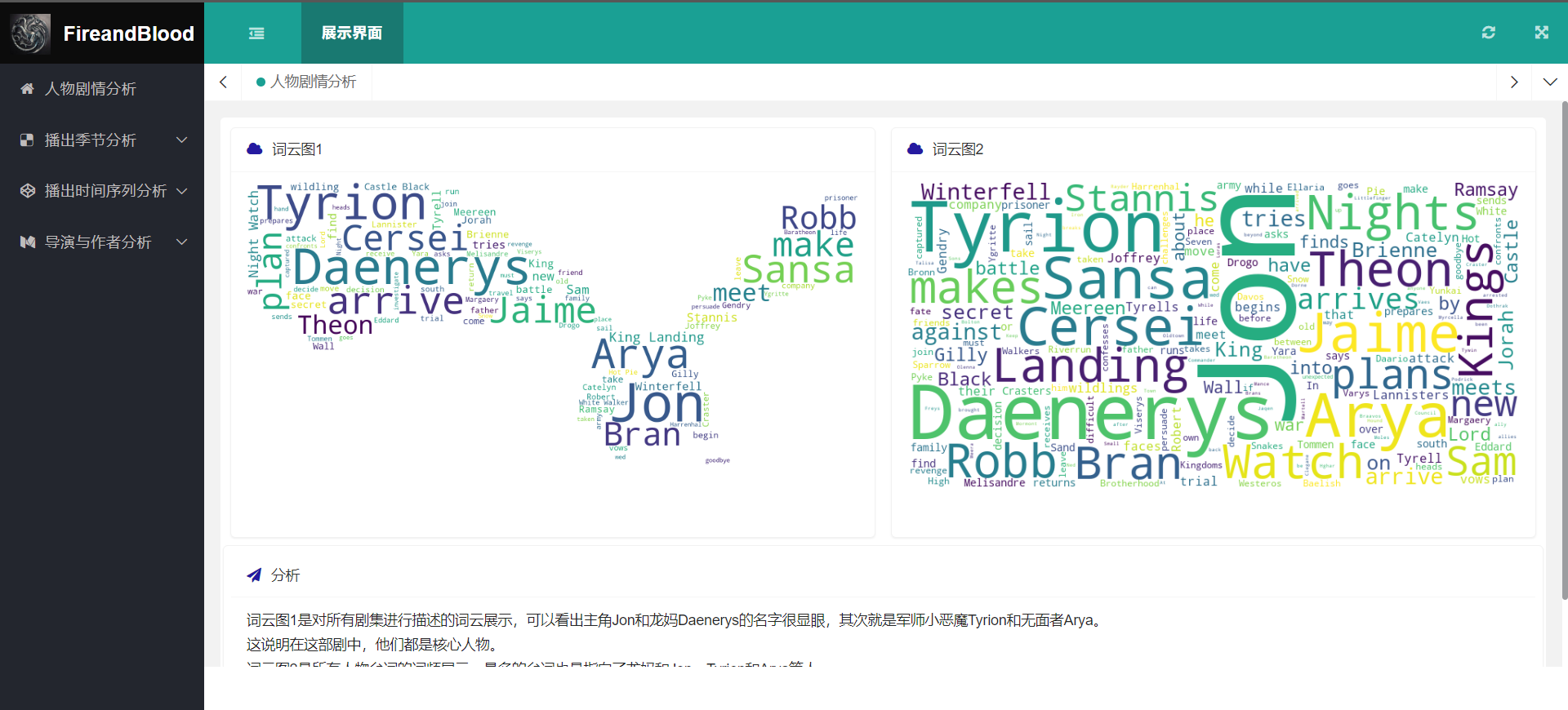
1. test

测试文件夹

# 3 系统展示

## 3.1 人物剧情分析

通过绘制词云图寻找《权力的游戏》中的核心人物。

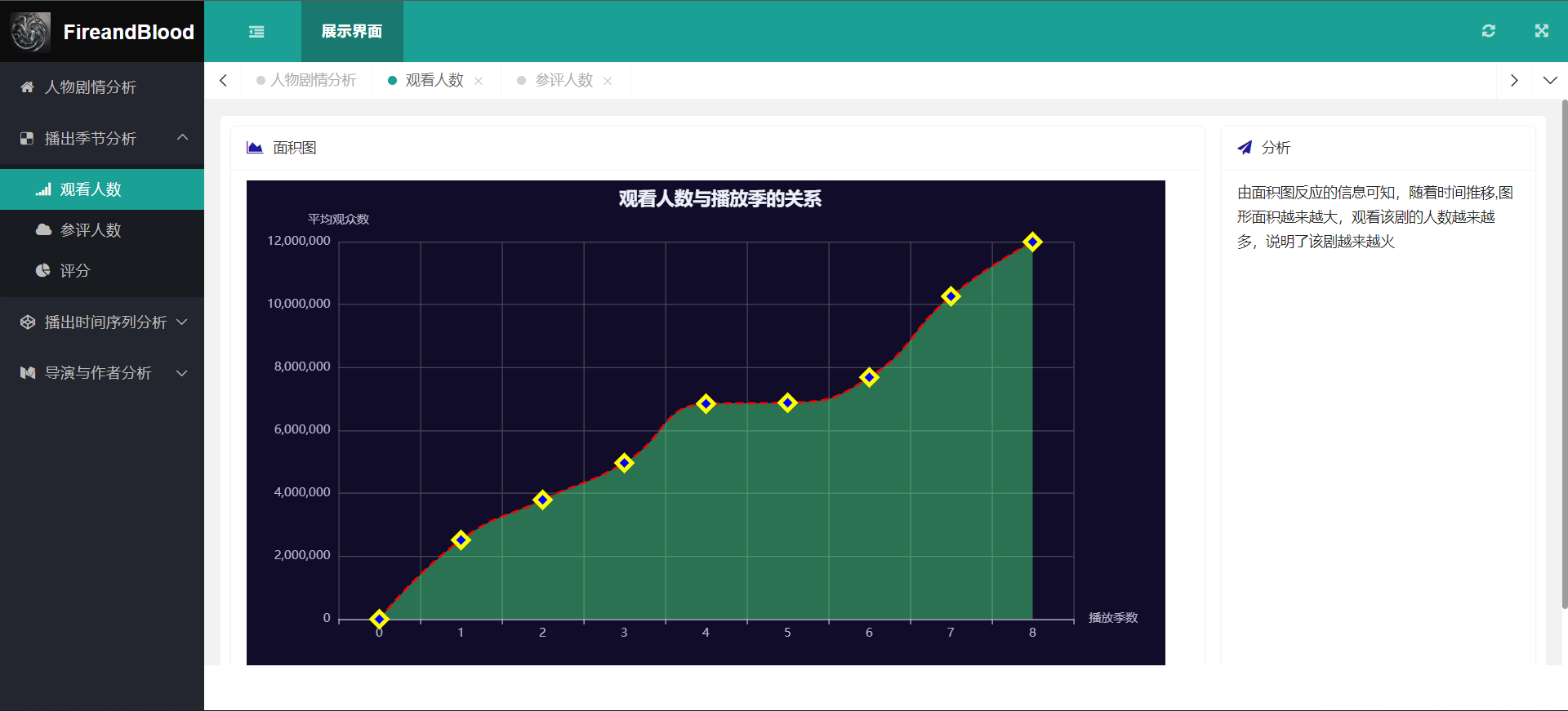


词云图1是对所有剧集进行描述的词云展示，可以看出主角Jon和龙妈Daenerys的名字很显眼，其次就是军师小恶魔Tyrion和无面者Arya。这说明在这部剧中，他们都是核心人物。

词云图2是所有人物台词的词频展示，最多的台词也是指向了龙妈和Jon，Tyrion和Arya等人。足以看出这些人物在剧中有着举足轻重的作用。

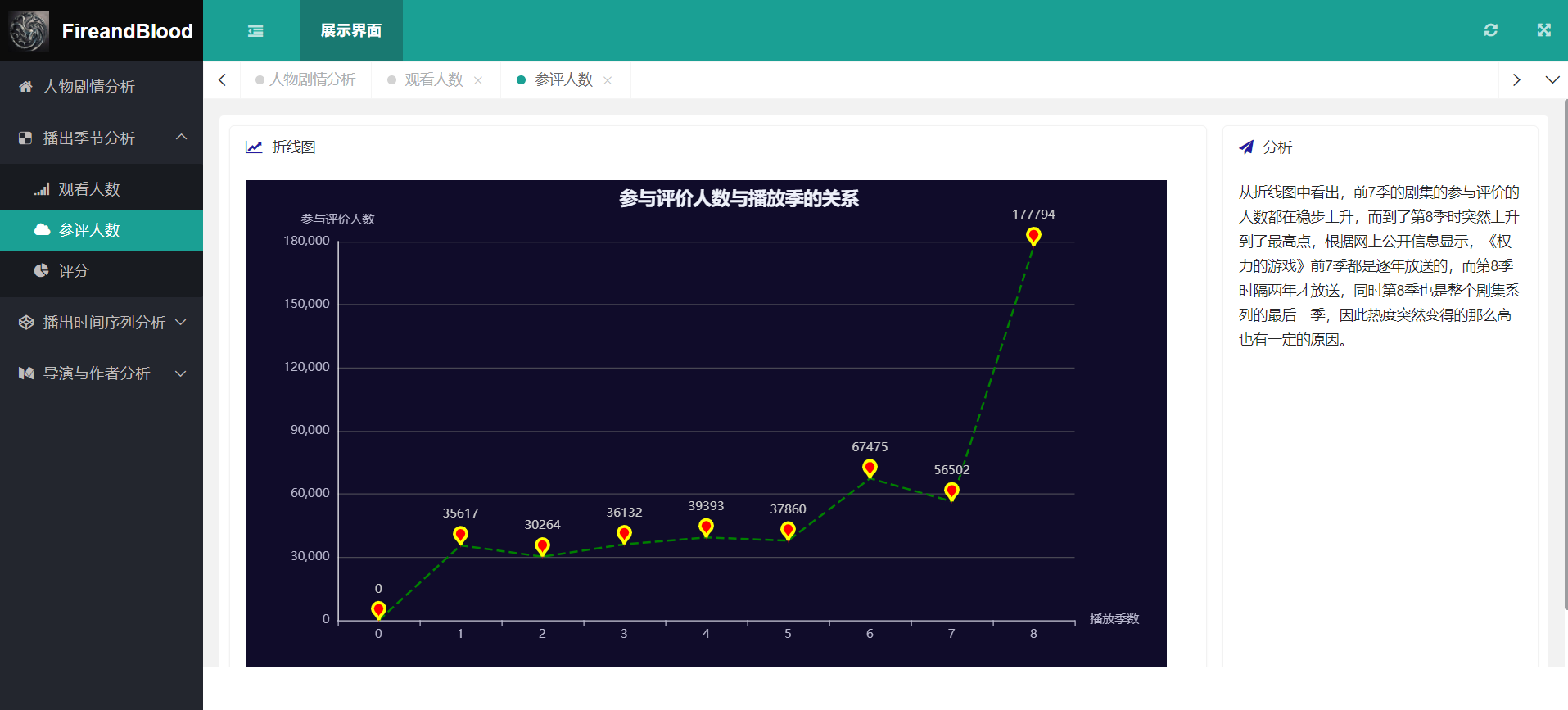
## 3.2 播出季节分析

1. 观看人数与播出季节分析



由面积图反应的信息可知，随着时间推移,图形面积越来越大，观看该剧的人数越来越多，说明了该剧越来越火。

1. 参与评价人数与播出季节分析



从折线图中看出，前7季的剧集的参与评价的人数都在稳步上升，而到了第8季时突然上升到了最高点，根据网上公开信息显示，《权力的游戏》前7季都是逐年放送的，而第8季时隔两年才放送，同时第8季也是整个剧集系列的最后一季，因此热度突然变得那么高也有一定的原因。

1. 评分与播出季节分析



图 《权力的游戏》第四季评分



图 《权力的游戏》第七季评分



图 《权力的游戏》第八季评分

由仪表盘可以看出，这部剧在第4季时候评价达到了顶峰，到了第7季也还不错，但到了第8季的评价就断崖式下跌了，可以很直观感受出

这部剧烂尾了。

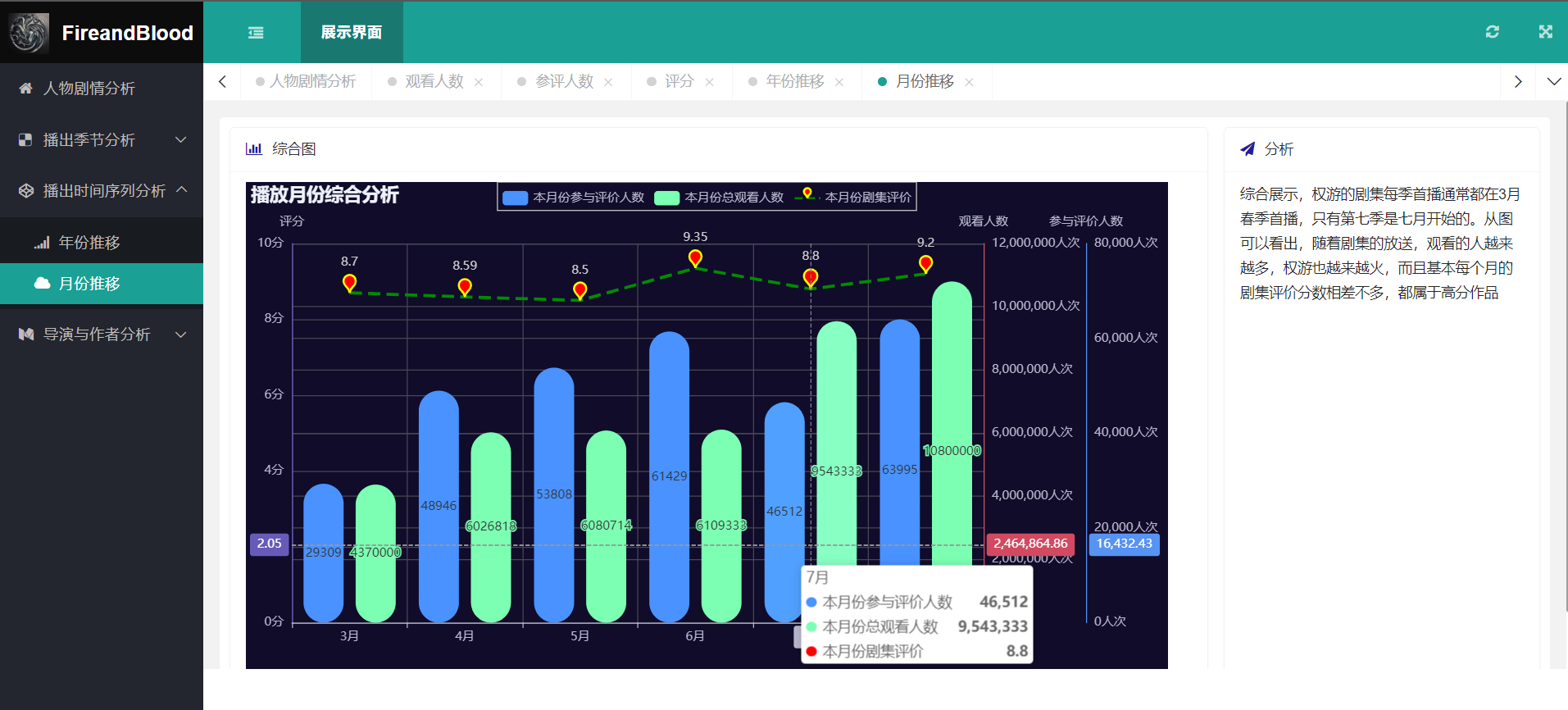
## 3.3 播出时间序列分析

（1）年份推移



通过3D的柱状图，更加立体感受出，随着时间推移,《权力的游戏》越发火爆。

（2）月份推移



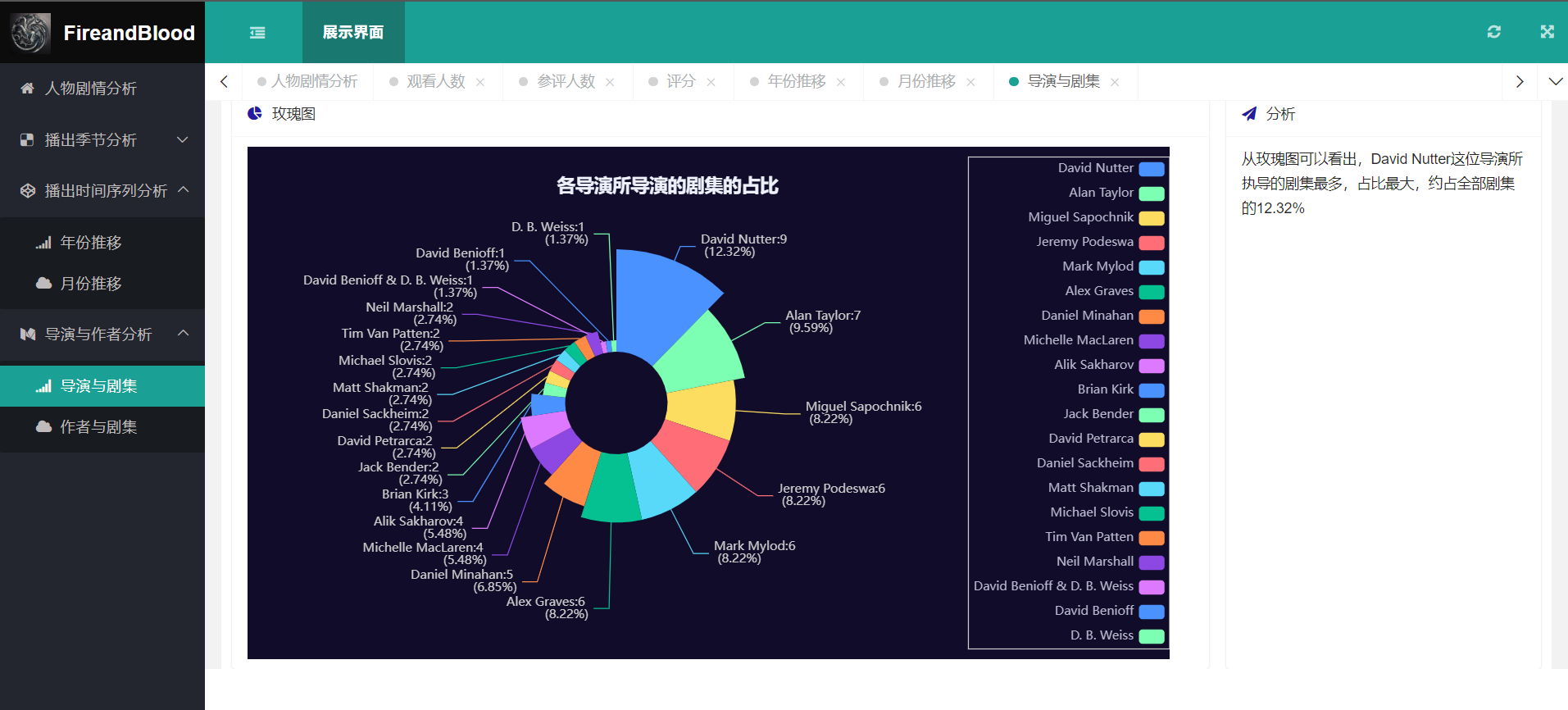
综合展示，权游的剧集每季首播通常都在3月春季首播，只有第七季是七月开始的。从图可以看出，随着剧集的放送，观看的人越来越多，权游也越来越火，而且基本每个月的剧集评价分数相差不多，都属于高分作品。

## 导演与作者分析

1. 导演与剧集



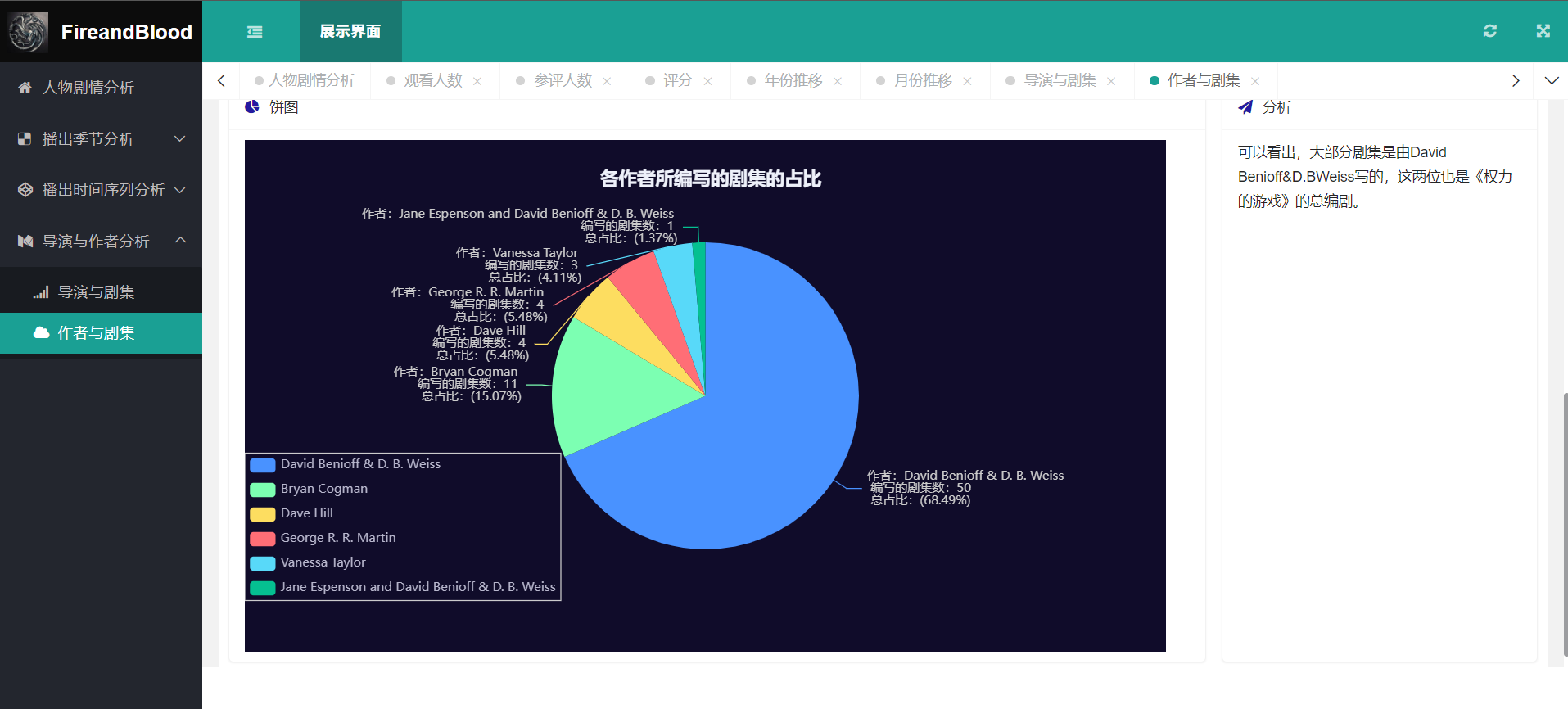
从柱状图可以看出，Neil Marshall导演的剧集平均得分最高，除了David Benioff & D.B.Weiss这个导演的平均得分是最低的，其他的导演导演的剧集得分都相差不多。



从玫瑰图可以看出，David Nutter这位导演所执导的剧集最多，占比最大，约占全部剧集的12.32%。

1. 作者与剧集



由柱状图可知，剧本本身是没有问题的，各个作者的剧集的得分都差不多（可以看到最高分就是原著作者George R.R Martin，高达9.2分）。

由饼图可以看出，大部分剧集是由David Benioff&D.BWeiss写的，这两位也是《权力的游戏》的总编剧。

# 4 总结

本系统基于Flask框架，结合LayuiMini前端框架、Flask蓝图和Jinja模板，实现了一个功能强大且易于维护的数据分析展示系统。利用Scala强大的数据处理能力，对《权力的游戏》相关数据进行高效处理并将数据导出到本地，进而使用Python进行可视化展示。通过模块化设计和模板继承机制，项目具备了良好的可维护性和扩展性，同时提供了卓越的用户体验和系统性能。

通过本系统的可视化展示可直观看出，《权力的游戏》是一部人物众多且个性刻画鲜明的优秀美剧，其中龙妈、小恶魔等人物给我们留下了深刻的印象。纵观《权力的游戏》所有系列剧集，其火爆和受欢迎程度不言而喻。《权力的游戏》前七季豆瓣评分均在9分以上！虽然第八季口碑有所下跌，《权力的游戏》最终烂尾，但《权力的游戏》仍然是一部史诗级的美剧。《权力的游戏》系列剧集共获得过40多次艾美奖，多次创下收视纪录，美国前总统特朗普甚至模仿剧中的经典台词“Winter is coming.”（凛冬将至）发推“Sanctions are coming.”（制裁将至）。

# 程序核心代码

|  |
| --- |
| import json  from flask import Blueprint, render\_template, url\_for  portal = Blueprint('portal', \_\_name\_\_)  @portal.route("/portal/initMenu")  def init\_menu():  rdata = dict()  head = dict()  head['title'] = "人物剧情分析"  head['href'] = "/welcome"  rdata['homeInfo'] = head  logos = dict()  logos['title'] = "FireandBlood"  logos['image'] = url\_for("static", filename="portal/images/logo.jpg")  logos['href'] = ""  rdata['logoInfo'] = logos  rdata['menuInfo'] = [{  "title":  "展示界面",  "icon":  "fa fa-address-book",  "href":  '',  "target":  "\_self",  "child": [{  "title": "人物剧情分析",  "href": "/welcome",  "icon": "fa fa-home",  "target": "\_self"  },{  "title": "播出季节分析",  "href": "",  "icon": "fa fa-delicious",  "target": "\_self",  "child": [{  "title": "观看人数",  "href": "/season1",  "icon": "fa fa-signal",  "target": "\_self"  }, {  "title": "参评人数",  "href": "/season2",  "icon": "fa fa-cloud",  "target": "\_self"  }, {  "title": "评分",  "href": "/season3",  "icon": "fa fa-pie-chart",  "target": "\_self"  },]  },{  "title": "播出时间序列分析",  "href": "",  "icon": "fa fa-codepen",  "target": "\_self",  "child": [{  "title": "年份推移",  "href": "/time1",  "icon": "fa fa-signal",  "target": "\_self"  }, {  "title": "月份推移",  "href": "/time2",  "icon": "fa fa-cloud",  "target": "\_self"  },]  },{  "title": "导演与作者分析",  "href": "",  "icon": "fa fa-medium",  "target": "\_self",  "child": [{  "title": "导演与剧集",  "href": "/direct1",  "icon": "fa fa-signal",  "target": "\_self"  }, {  "title": "作者与剧集",  "href": "/direct2",  "icon": "fa fa-cloud",  "target": "\_self"  },] }  ]  }]  return json.dumps(rdata, ensure\_ascii=False)  from flask import Blueprint, render\_template  import pymongo  import pandas as pd  from io import BytesIO  import base64  import imageio  import matplotlib.pyplot as plt  from wordcloud import WordCloud  from PIL import Image  import numpy as np  import re  from pyecharts.charts import Scatter, Pie, Bar  from pyecharts.charts import Line,Bar3D,Gauge, Timeline,Grid  from pyecharts.globals import ThemeType  from pyecharts import options as opts  from pyecharts.commons.utils import JsCode  import seaborn as sns  plt.rcParams['font.family'] = ['SimHei']  visualpic = Blueprint('visualpic', \_\_name\_\_)  season\_avgdata = pd.read\_csv('app\static\data\season\_avgdata.csv')  top\_words = pd.read\_csv('app\static\data\\top\_words.csv')  years\_data = pd.read\_csv('app\static\data\years\_data.csv')  months\_data = pd.read\_csv('app\static\data\months\_data.csv')  director\_avgdata = pd.read\_csv('app\static\data\director\_avgdata.csv')  director\_countdata = pd.read\_csv('app\static\data\director\_countdata.csv')  writer\_avgdata = pd.read\_csv('app\static\data\writer\_avgdata.csv')  writer\_countdata = pd.read\_csv('app\static\data\writer\_countdata.csv')  @visualpic.route("/welcome")  def cloud():  with open("app\static\data\desc.txt", encoding="utf-8") as f:  s = f.read()  # 读取自定义图片  custom\_mask = np.array(Image.open('app\static\img\dargon1.jpeg'))  # 创建词云对象，设置自定义图片作为底图  wc = WordCloud(  width=1920,  height=1080,  background\_color='white',  max\_words=100,  mask=custom\_mask # 设置自定义图片作为底图  )  # 加载词云文本  wc.generate(s)  wordcloud\_img1 = BytesIO()  wc.to\_image().save(wordcloud\_img1, format='PNG')  wordcloud1 = base64.b64encode(  wordcloud\_img1.getvalue()).decode('utf-8')    # 读取数据    # 清洗数据，去除value列中的各种杂乱符号  top\_words['value'] = top\_words['value'].astype(str) # 确保所有值都是字符串  top\_words['value'] = top\_words['value'].apply(lambda x: re.sub(r'[^A-Za-z\s]', '', x))  # 分组并计算总频次  cleaned\_top\_words = top\_words.groupby('value')['count'].sum().reset\_index()  # 将清洗后的数据转换为词云需要的字符串  word\_frequencies = {row['value']: row['count'] for \_, row in cleaned\_top\_words.iterrows()}  # 读取自定义图片  # custom\_mask = np.array(Image.open(r'./3.jpg'))  # 生成词云  wc2 = WordCloud(width=800, height=400, background\_color='white').generate\_from\_frequencies(word\_frequencies)  wordcloud\_img = BytesIO()  wc2.to\_image().save(wordcloud\_img, format='PNG')  wordcloud2 = base64.b64encode(  wordcloud\_img.getvalue()).decode('utf-8')    return render\_template('pages/welcome.html', plt1=wordcloud1, plt2=wordcloud2)    # return render\_template('pages/welcome.html', plt\_base64=wordcloud, ghvh=hjj)  @visualpic.route("/direct2")  def direct2():    x\_data = writer\_avgdata['written\_by'].tolist()[::-1]  y\_data = writer\_avgdata['avg(imdb\_rating)'].round(1).tolist()[::-1]  bar = (  Bar(init\_opts=opts.InitOpts(theme=ThemeType.DARK))  .add\_xaxis(x\_data)  .add\_yaxis(  '',  y\_data,  category\_gap='50%',  itemstyle\_opts={  "normal": {  "color": JsCode(  """new echarts.graphic.LinearGradient(0, 0, 0, 1, [{  offset: 0,  color: 'rgba(0, 244, 255, 1)'  }, {  offset: 1,  color: 'rgba(100, 160, 167, 1)'  }], false)"""  ),  "barBorderRadius": [30, 30, 30, 30],  "shadowColor": "rgb(0, 160, 221)",  }  }, # 设置柱形图颜色为渐变色  )  .reversal\_axis()  .set\_series\_opts(label\_opts=opts.LabelOpts(position='right'))  .set\_global\_opts(  xaxis\_opts=opts.AxisOpts(name='评分'),  title\_opts=opts.TitleOpts(title='不同作者编写剧集的平均得分',pos\_left='33%',pos\_top="3%"),  legend\_opts=opts.LegendOpts(is\_show=False) # 隐藏图例  )  )  # 设置整体布局  grid = (  Grid(init\_opts=opts.InitOpts(theme=ThemeType.DARK))  .add(  bar,  grid\_opts=opts.GridOpts(  pos\_right='10%', # 将柱形图的中心位置偏向右侧  pos\_top='10%', # 调整图表位置  width='70%', # 调整整体图表宽度  height='80%', # 调整整体图表高度  )  )  )  line\_html1 = grid.render\_embed()  x\_data = writer\_countdata['written\_by'].tolist()  y\_data = writer\_countdata['count'].tolist()  pie = (  Pie(init\_opts=opts.InitOpts(    theme=ThemeType.DARK)  )  .add(  '',  [list(z) for z in zip(x\_data, y\_data)],  radius=['0%', '60%'],  center=['50%', '50%'],  label\_opts=opts.LabelOpts(is\_show=True),  )  .set\_global\_opts(  title\_opts=opts.TitleOpts(title='各作者所编写的剧集的占比',pos\_left='38%',pos\_top="5%"),  legend\_opts=opts.LegendOpts(  type\_="scroll",  pos\_left="left", # 调整图例位置  pos\_bottom="10%",  orient="vertical"  )  )  .set\_series\_opts(  label\_opts=opts.LabelOpts(formatter='作者：{b} \n 编写的剧集数：{c} \n 总占比：({d}%)'),  position="outside"  )  )  line\_html2 = pie.render\_embed()  # 显示图表  return render\_template('pages/direct2.html',  script=grid.js\_dependencies,  direct21=line\_html1,  direct22=line\_html2,  )  @visualpic.route("/direct1")  def direct1():  x\_data = director\_avgdata['directed\_by'].tolist()[::-1]  y\_data = director\_avgdata['avg(imdb\_rating)'].round(1).tolist()[::-1]  bar = (  Bar(init\_opts=opts.InitOpts(theme=ThemeType.DARK))  .add\_xaxis(x\_data)  .add\_yaxis(  'y轴',  y\_data,  category\_gap='50%',  itemstyle\_opts={  "normal": {  "color": JsCode(  """new echarts.graphic.LinearGradient(0, 0, 0, 1, [{  offset: 0,  color: 'rgba(0, 244, 255, 1)'  }, {  offset: 1,  color: 'rgba(100, 160, 167, 1)'  }], false)"""  ),  "barBorderRadius": [30, 30, 30, 30],  "shadowColor": "rgb(0, 160, 221)",  }  }, # 设置柱形图颜色为渐变色  )  .reversal\_axis()  .set\_series\_opts(label\_opts=opts.LabelOpts(position='right'))  .set\_global\_opts(    xaxis\_opts=opts.AxisOpts(name='评分'),  title\_opts=opts.TitleOpts(title='不同导演拍摄的剧集的平均得分',pos\_left='33%',pos\_top="3%"),  legend\_opts = opts.LegendOpts(is\_show = False),  )  )    # 设置整体布局  grid = (  Grid(init\_opts=opts.InitOpts(theme=ThemeType.DARK))  .add(  bar,  grid\_opts=opts.GridOpts(  pos\_right='10%', # 将柱形图的中心位置偏向右侧  pos\_top='10%', # 调整图表位置  width='70%', # 调整整体图表宽度  height='80%', # 调整整体图表高度  )  )  )    line\_html1 = grid.render\_embed()  director\_countdata = pd.read\_csv('app\static\data\director\_countdata.csv')    # 提取 x 和 y 数据  x\_data = director\_countdata['directed\_by'].tolist()  y\_data = director\_countdata['count'].tolist()  pie = (  Pie(init\_opts=opts.InitOpts(  theme=ThemeType.DARK)  )  .add(  '',  [list(z) for z in zip(x\_data, y\_data)],  radius=['20%', '60%'],  center=['40%', '50%'],  rosetype="radius",  label\_opts=opts.LabelOpts(is\_show=True),  )  .set\_global\_opts(  title\_opts=opts.TitleOpts(title='各导演所导演的剧集的占比',pos\_left='33%',pos\_top="5%"),  legend\_opts=opts.LegendOpts(  type\_="scroll",  pos\_left="right", # 调整图例位置  pos\_top="2%",  orient="vertical"  )  )  .set\_series\_opts(  label\_opts=opts.LabelOpts(formatter='{b}:{c} \n ({d}%)'),  position="outside"  )  )  line\_html2 = pie.render\_embed()  # 显示图表  return render\_template('pages/direct1.html',  script=grid.js\_dependencies,  direct1=line\_html1,  direct2=line\_html2,  )  @visualpic.route("/time2")  def time2():  def transMonthStr(x):  return str(x)+'月'  x\_data = months\_data['month'].apply(transMonthStr).tolist()  views = months\_data['us\_viewers'].round(0).tolist()  imdb = months\_data['imdb\_rating'].round(2).tolist()  total\_votes = months\_data['total\_votes'].round(0).tolist()  bar = (  Bar()  .add\_xaxis(x\_data)  .add\_yaxis(  "本月份参与评价人数",  total\_votes,  yaxis\_index=0,  color="#d14a61",  itemstyle\_opts={"barBorderRadius": [30, 30, 20, 20]}  )  .add\_yaxis(  "本月份总观看人数",  views,  yaxis\_index=1,  color="#5793f3",  itemstyle\_opts={"barBorderRadius": [30, 30, 20, 20]}  )  .extend\_axis(  yaxis=opts.AxisOpts(  name="观看人数",  type\_="value",  min\_=0,  max\_=12000000,  position="right",  axisline\_opts=opts.AxisLineOpts(  linestyle\_opts=opts.LineStyleOpts(color="#d14a61")  ),  axislabel\_opts=opts.LabelOpts(formatter="{value}人次"),  )  )  .extend\_axis(  yaxis=opts.AxisOpts(  type\_="value",  name="评分",  min\_=0,  max\_=10,  position="left",  axisline\_opts=opts.AxisLineOpts(  linestyle\_opts=opts.LineStyleOpts(color="#675bba")  ),  axislabel\_opts=opts.LabelOpts(formatter="{value}分"),  splitline\_opts=opts.SplitLineOpts(  is\_show=True, linestyle\_opts=opts.LineStyleOpts(opacity=1)  ),  )  )  .set\_global\_opts(  yaxis\_opts=opts.AxisOpts(  name="参与评价人数",  min\_=0,  max\_=80000,  position="right",  offset=100,  axisline\_opts=opts.AxisLineOpts(  linestyle\_opts=opts.LineStyleOpts(color="#5793f3")  ),  axislabel\_opts=opts.LabelOpts(formatter="{value}人次"),  ),  title\_opts=opts.TitleOpts(title="播放月份综合分析"),  tooltip\_opts=opts.TooltipOpts(trigger="axis", axis\_pointer\_type="cross"),    )  )  line = (  Line()  .add\_xaxis(x\_data)  .add\_yaxis(  "本月份剧集评价",  imdb,  linestyle\_opts=opts.LineStyleOpts(color='green', width=3, type\_='dashed'), # 配置折线样式  itemstyle\_opts=opts.ItemStyleOpts(color='red', border\_color='yellow', border\_width=2), # 配置图元样式  symbol='pin', # 设置图元形状  symbol\_size=20, # 设置图元大小  yaxis\_index=2,  color="#675bba",  label\_opts=opts.LabelOpts(is\_show=True),  )  )  bar.overlap(line)  grid = Grid(init\_opts=opts.InitOpts(theme=ThemeType.DARK))  grid.add(bar, opts.GridOpts(pos\_left="5%", pos\_right="20%"), is\_control\_axis\_index=True)  line\_html = grid.render\_embed()  # 显示图表  return render\_template('pages/time2.html',  script=grid.js\_dependencies,  time2=line\_html,)  @visualpic.route("/time1")  def time1():  # 统一列名  season\_columns\_name = season\_avgdata.columns.tolist()  new\_colums\_name = ['year']  for colum in season\_columns\_name[1:]:  new\_colums\_name.append((colum))  years\_data.columns = new\_colums\_name  years\_season\_data = pd.merge(season\_avgdata[['season','avg(total\_votes)']],years\_data,on='avg(total\_votes)',how='left')  def transYearStr(x):  return str(x)+'年'  def tranSeasonStr(x):  return '第'+str(x)+'季'  x\_axis = years\_season\_data['year'].apply(transYearStr).tolist()  y\_axis = years\_season\_data['season'].apply(tranSeasonStr).tolist()  views\_data = years\_season\_data['avg(us\_viewers)'].round(0).tolist()  x\_len = len(x\_axis)  y\_len = len(y\_axis)  data = []  for i in range(x\_len):  for j in range(y\_len):  if(i == j):  tmp = [i,j,views\_data[i]]  else:  tmp = [i,j,0]  data.append(tmp)  range\_color = [  '#313695', '#4575b4', '#74add1', '#abd9e9', '#e0f3f8', '#ffffbf',  '#fee090', '#fdae61', '#f46d43', '#d73027', '#a50026'  ]  # 创建 Bar3D 图表  bar3d = Bar3D(init\_opts=opts.InitOpts(  theme=ThemeType.DARK))  # 添加数据到 Bar3D 图表  bar3d.add(  "",  [[d[1], d[0], d[2]] for d in data],  xaxis3d\_opts=opts.Axis3DOpts(  type\_='category',  data=x\_axis,  axislabel\_opts=opts.LabelOpts(color="white"), # 设置轴标签颜色为白色  ),  yaxis3d\_opts=opts.Axis3DOpts(  type\_='category',  data=y\_axis,  axislabel\_opts=opts.LabelOpts(color="white"), # 设置轴标签颜色为白色  ),  zaxis3d\_opts=opts.Axis3DOpts(  type\_='value',  axislabel\_opts=opts.LabelOpts(color="white"), # 设置轴标签颜色为白色  ),  grid3d\_opts=opts.Grid3DOpts(is\_rotate=True,rotate\_speed=15,width=200,depth=80),  shading="lambert",  )  # 设置全局配置  bar3d.set\_global\_opts(  visualmap\_opts=opts.VisualMapOpts(  max\_=12000000,  range\_color=range\_color,    ),  legend\_opts = opts.LegendOpts(is\_show = False),  )  line\_html = bar3d.render\_embed()  # 显示图表  return render\_template('pages/time1.html',  script=bar3d.js\_dependencies,  time1=line\_html,)  @visualpic.route("/season3")  def season3():  x\_data = season\_avgdata['season'].tolist()  y\_data = season\_avgdata['avg(imdb\_rating)'].round(2).tolist()  # 创建 Timeline 对象  timeline = Timeline(init\_opts=opts.InitOpts(theme=ThemeType.DARK))  # 为每个季数创建一个 Gauge 图表并添加到 Timeline  for season, rating in zip(x\_data, y\_data):  # 将评分转换为0-100的百分比  # 现在是0.0作为底线，10.0作为满分，可修改让指针变化更明显  percentage = round((rating-0 )/ 10 \* 100, 1)  gauge = (  Gauge()  .add(  "",  [("本季评分", percentage)],  title\_label\_opts=opts.LabelOpts(  font\_size=28, color="white", font\_family="Microsoft YaHei"  ),  min\_=0,  max\_=100,  split\_number=5,  axisline\_opts=opts.AxisLineOpts(  linestyle\_opts=opts.LineStyleOpts(  color=[(0.2, "red"), (0.8, "yellow"), (1, "green")], width=30  )  ),  )  .set\_global\_opts(  title\_opts=opts.TitleOpts(title=f"第{season}季 - 本季平均评分",pos\_left='center'),  legend\_opts = opts.LegendOpts(is\_show = False),  )  )  timeline.add(gauge, f"第{season}季")  # 设置 Timeline 配置  timeline.add\_schema(  play\_interval=1000,  is\_auto\_play=True,  is\_loop\_play=True,  pos\_bottom="10px",  pos\_left="195px",  )  line\_html = timeline.render\_embed()  # 显示图表  return render\_template('pages/season3.html',  script=timeline.js\_dependencies,  chart3\_html=line\_html,)  @visualpic.route("/season2")  def season2():  x\_data = x\_data = season\_avgdata['season'].tolist()  x\_data.insert(0,0)  y\_data = y\_data = season\_avgdata['avg(total\_votes)'].round(0).tolist()  y\_data.insert(0,0)  # 创建 Line 图表对象  line = Line(init\_opts={"theme": ThemeType.DARK})  line.add\_xaxis(x\_data)  line.add\_yaxis("平均观众数", y\_data, is\_smooth=False,  linestyle\_opts=opts.LineStyleOpts(color='green', width=2, type\_='dashed'), # 配置折线样式  itemstyle\_opts=opts.ItemStyleOpts(color='red', border\_color='yellow', border\_width=3), # 配置图元样式  symbol='pin', # 设置图元形状  symbol\_size=20, # 设置图元大小  )  # 设置全局图表选项  line.set\_global\_opts(  title\_opts=opts.TitleOpts(title="参与评价人数与播放季的关系",pos\_top='1%',pos\_left='40%'),  xaxis\_opts=opts.AxisOpts(name="播放季数",  splitline\_opts=opts.SplitLineOpts(is\_show=False) # 隐藏 x 轴网格线  ),  yaxis\_opts=opts.AxisOpts(name= "参与评价人数",  splitline\_opts=opts.SplitLineOpts(is\_show=True), # 显示 y 轴网格线  axisline\_opts=opts.AxisLineOpts( # 配置 y 轴轴线  linestyle\_opts=opts.LineStyleOpts(color='white')  )  ),  legend\_opts=opts.LegendOpts(is\_show=False) # 隐藏图例  )  line\_html = line.render\_embed()  # 显示图表  return render\_template('pages/season2.html',  script=line.js\_dependencies,  chart2\_html=line\_html,)  @visualpic.route("/season1")  def season1():  x\_data = x\_data = season\_avgdata['season'].tolist()  x\_data.insert(0,0)  y\_data = y\_data = season\_avgdata['avg(us\_viewers)'].round(0).tolist()  y\_data.insert(0,0)  # 创建 Line 图表对象  line = Line(init\_opts={"theme": ThemeType.DARK})  line.add\_xaxis(x\_data)  line.add\_yaxis("平均观众数", y\_data, is\_smooth=True,  linestyle\_opts=opts.LineStyleOpts(color='red', width=2, type\_='dashed'), # 配置折线样式  itemstyle\_opts=opts.ItemStyleOpts(color='blue', border\_color='yellow', border\_width=4), # 配置图元样式  symbol='diamond', # 设置图元形状  symbol\_size=15, # 设置图元大小  ) # 设置 is\_smooth=True 来显示平滑的曲线  # 设置线条样式  line.set\_series\_opts(  areastyle\_opts=opts.AreaStyleOpts(color='rgba(80, 255, 140, 0.7)',opacity= 0.6), # 设置面积图透明度  label\_opts={"is\_show": False} # 设置不显示标签  )  # 设置全局图表选项  line.set\_global\_opts(  title\_opts=opts.TitleOpts(title="观看人数与播放季的关系",pos\_top='1%',pos\_left='40%'),  xaxis\_opts=opts.AxisOpts(name="播放季数"),  yaxis\_opts=opts.AxisOpts(name= "平均观众数"),  legend\_opts=opts.LegendOpts(is\_show=False) # 隐藏图例  )  line\_html = line.render\_embed()  # 显示图表  return render\_template('pages/season1.html',  script=line.js\_dependencies,  chart1\_html=line\_html,) |