第5章

健康医疗大数据分析

5.1 移动健康大数据来源与获取

在收集数据时，数据来源包含两种方式。第一种方式就是所谓的直接来源，通过直接来源获取的数据就是第一手数据，这类数据主要来源于直接的调查或实验的结果。第二种方式就是间接数据，也称为第二手数据，第二手数据一般来源于他人的调查或实验，是对结果进行加工整理后的数据。在移动健康大数据中，包含通过可穿戴式设备（如健康手环、智能体脂秤等）直接记录的个人健康数据，也包含通过移动医疗 APP 记录的与健康相关的信息。

5.1.1 可穿戴式设备

可穿戴式设备即直接穿在身上，或是整合到用户的衣服或配件中的一种便携式设备。可穿戴式设备不仅仅是一种硬件设备，还是可通过软件支持以及数据交互、云端交互来实现强大的功能的设备。可穿戴式设备将会对我们的生活、感知带来很大的转变。随着传感技术的发展，可穿戴式设备的种类越来越多样化，智能可穿戴式设备作为新型的便携式电子产品类别，主要产品有智能手表、智能手环、无线耳机等，通过蓝牙连接或 APP 的支持，与手机建立实时传输或数据同步。智能可穿戴式设备有着便携式的特点和多功能化的交互式体验。目前，智能手表/智能手环的市场份额主要由手机厂商和早布局的厂商占据，国内智能手表/手环主要有华为、荣耀、小米、红米、oppo、vivo、一加、努比亚等手机品牌，也有华米、出门问问、小天才等专注于智能穿戴的品牌，国外智能手表/手环品牌主要有苹果、三星、Fitbit、佳明、boAt 等。未来智能可穿戴式设备的功能将不只局限于监测心率、计步、感应体温、睡眠等，还能检测到生理指标的异常，在医疗健康方面起到切实的作用。

智能手表/手环在医疗健康方面提供的检测功能主要是记录用户的运动习惯、健身效果、睡眠质量、饮食安排等一系列相关的数据，并且可以将这些数据同步到用户的移动终端设备中，终端设备可能会根据自己的“分析功能”给出相关建议，并将用户分为不同的状态（健康状态、亚健康状态、生病状态），起到通过数据指导健康生活的作用。然而，对于数据监测状态，大部分人所处的一种状态是无数据或极少数据监测的状态（此时，将其定义为无监测状态），一旦感觉身体不适，才来到医院进行检查，并获得此次检查的身体数据（此时，可将其定义为准确监测状态），以实现健康状态评估和疾病精准诊断。

5.1.2 移动医疗 APP

移动医疗为一个新兴行业，从 2009 年起，我国为满足群众医疗健康需求，缓解医疗卫生发展不均衡矛盾而出台多项推动移动医疗发展的利好政策。互联网的特征表现在：信息量大、使用频率高、用户平均收入低、维护成本低、标准化易复制、爆点之后赢者通吃、边际成本为零；而医疗行业的特征表现在：用户平均收入高、成本高、属地化难以赢者通吃、支付方与决定者不对等、政策及监管的壁垒等。移动医疗通过移动通信技术可以快速提供给患者对应的医疗服务和信息。与传统医疗相比，移动医疗对医疗资源配置进行了优化利用，提高了患者、医生、医院之间的问诊效率，省时更省心。

移动医疗 APP 发展至今已经具备比较成熟的商业模式，根据移动医疗 APP 功能，可以分为以下几类：

(1) 在线寻医问诊类：如平安好医生、春雨掌上医生、丁香医生、阿里健康等。

在线问诊类 APP 是用户通过付费方式向医生问诊，医生提供相应医疗方案。可以利用图文、视频、电话等方式沟通。在线问诊可以说是现在移动医疗 APP 市场竞争最激烈的领域，平台需要提供大量优质的线上医生资源，来为用户提供专业的医疗服务。服务形式有平台分发、医生抢单、用户付费指定医生、付费悬赏等。在线问诊尤其在疫情期间发挥了巨大的作用。

(2) 预约挂号/导诊：如好大夫在线、健康 160、趣医院、挂号网等。

医疗导诊 APP 因渠道丰富，解决了用户挂号、候诊、缴费排队时间长等问题，同时帮助医院进一步优化了医疗资源匹配方式和服务流程，提升了用户就医体验而备受好评。主要通过连接医院应用程序编程接口(application programming interface，API)、数据接口或号源池的方式来实现挂号、导诊等功能。但受医院开发、运维能力以及数据安全稳定等因素制约，并非所有医院都可以提供 API 接口，目前市面上只有少部分挂号 APP 具备这种模式。

(3) 医药服务类：如叮当快药、1 药网、用药助手。

受国家发展医药电商政策影响，医药电商资质审核进一步扩大，不再被互联网药品资质牵制，医药 O2O 平台开始快速发展，市场上开始出现自营医药 B2C、平台医药 B2C、医药 O2O 等服务企业。

(4) 健康管理类(饮食、数据监测)：如 keep、微脉、悦动圈等。

这类 APP 通过鼓励用户参与平台各种活动，上传自己身体、健康、饮食。睡眠等数据，形成用户健康档案，为用户提供定制化健康管理方案，一部分平台经过前期用户沉淀已经建立自己的生态圈，开始向线上商城等更多元的方向发展。

(5) 健康知识库：如健康汇、掌上糖医等。

这类 APP 主要整合有价值的健康医疗信息，形成相应的健康知识库。表现形式包含有文字、图片、动画、视频，并且还会有专题知识讲座，不管是小孩还是老人看了图文结合的介绍后都能理解。

5.1.3 开源健康数据获取

穿戴式设备、移动医疗 APP 多元化发展催生了移动健康大数据，而这些设备或 APP 都会考虑保护用户数据隐私，因此很难直接获取到数据。如何获取这些直接或间接产生的数

据进行下一步分析成为目前的难题。

Kaggle 成立于 2010 年，是一个进行数据发掘和预测竞赛的在线平台。企业和研究者可在其上发布数据，统计学者和数据挖掘专家可在其上进行竞赛，通过“众包”的形式以产生最好的模型。Kaggle 上的内容可以分为竞赛（competitions）、数据集（datasets）、模型（models）、代码（code）等模块，如图 5-1 所示。

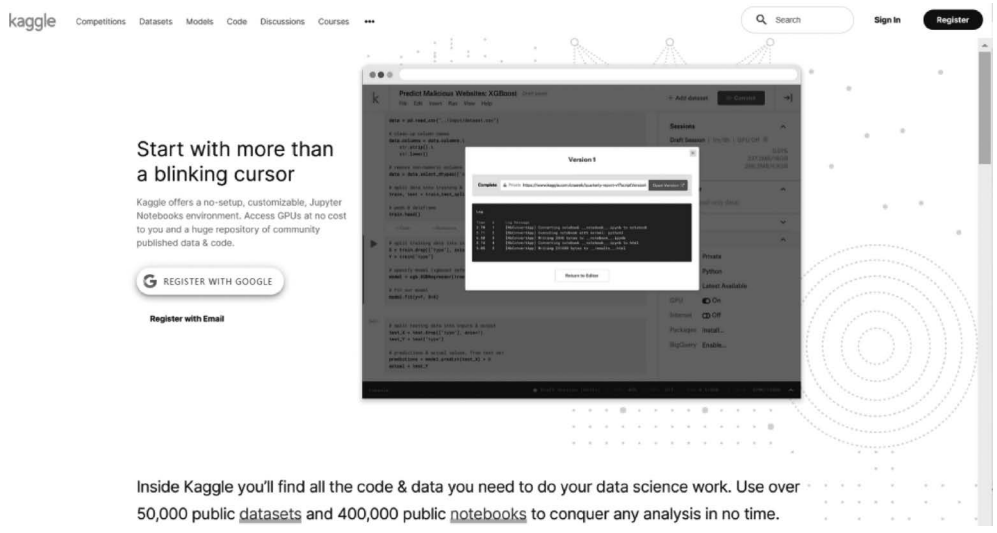

图 5-1　Kaggle 官网

1. 竞赛

Kaggle 上的竞赛有各种分类，例如奖金极高、竞争激烈的“Featured”，相对平民化的“Research”，等等。但整体的项目模式是一样的，就是通过出题方给予的训练集建立模型，再利用测试集算出结果用来评比。参加比赛的流程相当简单，简单来说就是分为 Download、Build、Submit 三个步骤，如图 5-2 所示。

（1）Download

每个比赛都会有较详细的子页面来介绍背景、数据集、评价指标和提交方式等相关要求。Kaggle 负责对这一过程以及竞赛模型的构建、数据的匿名化以及集成最终获胜的模型提供咨询服务。

（2）Build

参赛者在本地构建模型后，输入比赛提供的数据集进行训练，得到的预测结果上传回 Kaggle。

（3）Submit

参赛者相互竞赛以获得最优的模型，参赛者提交答案后会根据预测精度被立即评分，并实时显示。

由于这类问题并没有标准答案，只有无限逼近最优解，所以这样的模式可以激励参与者提出更好的方案，甚至推动整个行业的发展。

2. 数据集

Kaggle 上的数据集服务收集了许多公共的数据集（包含计算机视觉、NLP、教育、农业、互联网等数据），并且提供数据下载、介绍、相关脚本（scripts）以及独立的论坛等服务。利用

Kaggle 上的数据集，通过简单下载，就可以自己在本地或在线进行数据挖掘、分析。Kaggle 上的数据集竞赛如图 5-3 所示。

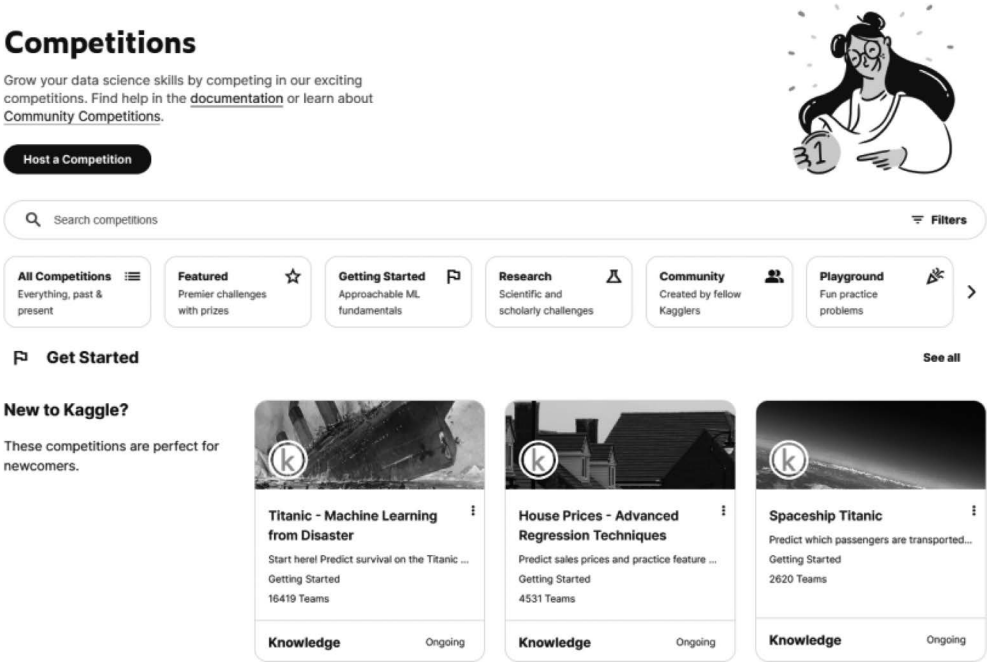

图 5-2　Kaggle 官网竞赛

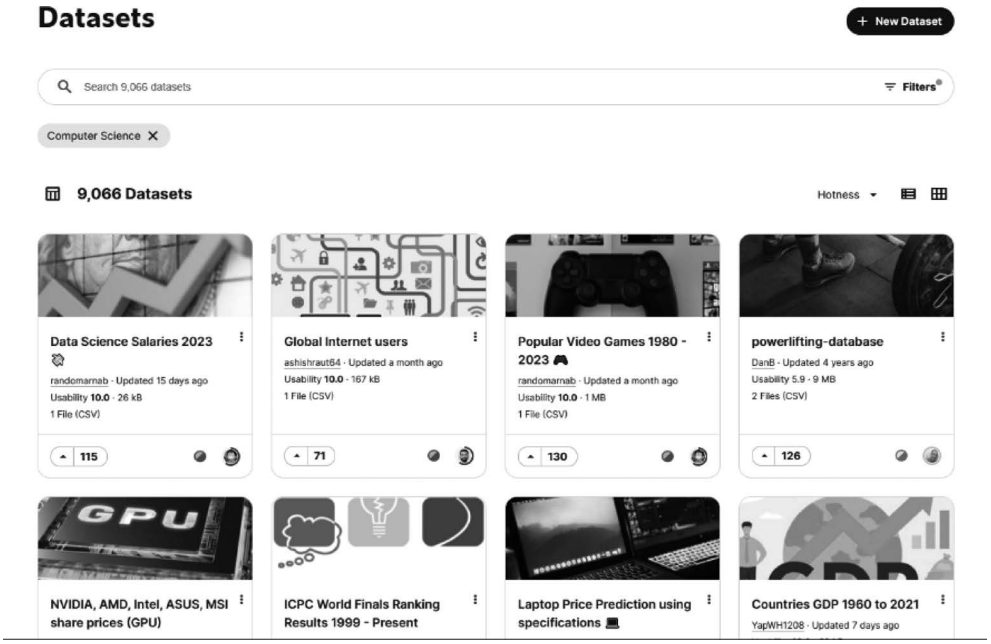

图 5-3　Kaggle 官网数据集竞赛

关于 Kaggle 上的数据集下载的问题，主要有两种途径：①直接下载；②通过提供的 API 下载。

（1）直接下载

对于案例中使用的健康相关数据，采用其他用户分享的小米手环记录的步数和睡眠记录。它的网址如下：https://www.kaggle.com/datasets/damirgadylyaev/more-than-4-years-of-steps-and-sleep-data-mi-band。进入页面，可以看到数据的分享者及数据介绍，如图 5-4 所示。

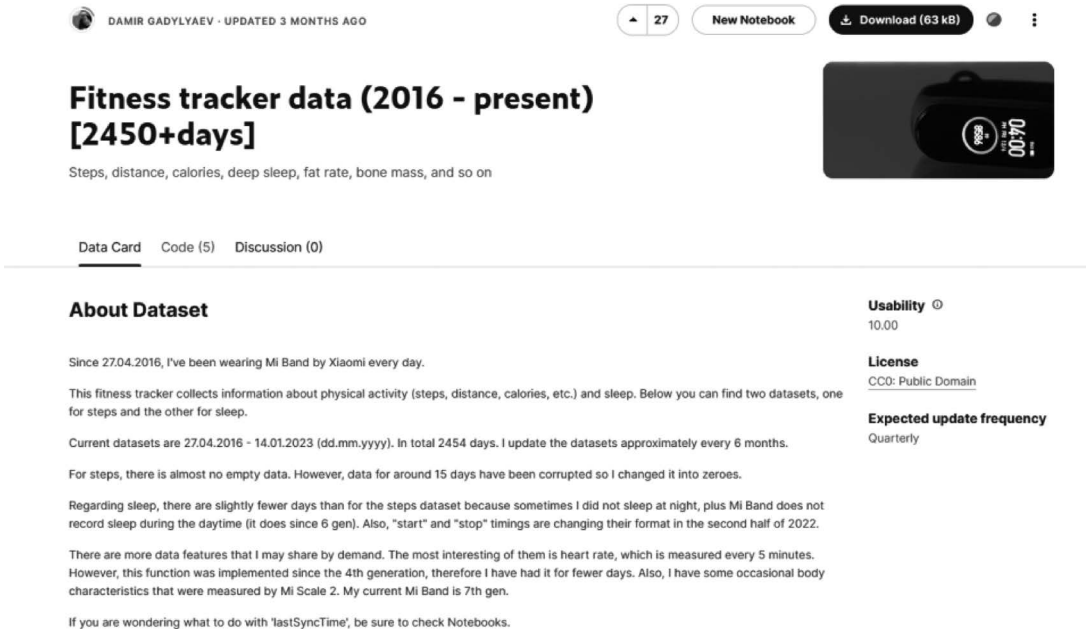

图 5-4　Fitness tracker data 页面介绍

该页面给出了直接的下载链接，这种数据直接单击下载即可。

开源的手环数据相对较少，在 Kaggle 平台上搜索到还有另一个 google data analytics 项目，创建者分享了包括 Fitbit、小米手环等数据，具体包含活动、心率、步数、睡眠等详细信息，如图 5-5 所示。

（2）通过 API 下载

有些数据集没有提供直接下载的按钮，只提供下载的 API 接口。而且一般都是在 Python 下对 Kaggle 上的数据集进行处理，下面默认已安装过 Python 并且可以使用 pip 进行 Python 库的安装。

首先，使用 pip installl kaggle 语句安装包；完成后输入命令 kaggle compitions list 检查是否成功安装 kaggle 库。

其次，单击“My account”，进去之后，单击“Create New API Token”就可以下载 kaggle.json 文件了；把下载的 kaggle.json 文件放到用户目录下，根据提示 Ensure kaggle.json is in the location ～/.kaggle/kaggle.json to use the API.，下载的 kaggle.json 文件要放到用户目录下的隐藏文件.kaggle 文件夹下。如果安装完 kaggle 之后是没有看到这个文件夹，就手动创建一下即可。

然后直接在命令行下输入下面的命令即可下载数据集。

kaggle datasets download -d dataname

Description of data sources used

S. No.	Dataset Name	Contributor	Source URL	Year of tracking	Device used	Number of participants	Number of days	Activity data	Sleep Data	Heart rate data	E
1	FitBit Fitness Tracker Data	Mobius	https://www.kaggle.com/arashnic/fitbit	2016	FitBit Tracker	33	31	Present	Present	Present	F
2	fitbit dataset	Akash Kumar	https://www.kaggle.com/singhakash/fitbit-dataset	2016	Fitbit Charge HT fitness tracker	35	32	Present	Absent	Absent	A
3	One year of Fitbit ChargeHR data	Alket Cecaj	https://www.kaggle.com/alketcecaj/one-year-of-fitbit-chargehr-data	2015	Fitbit Charge HT fitness tracker	1	366	Present	Absent	Absent	A
4	Fitness Trends Dataset	Arooj Anwar Khan	https://www.kaggle.com/aroojanwarkhan/fitness-data-trends	2017-2018	Samsung Health Application	1	96	Present	Present	Absent	F
5	Fitbit Dataset	Josh Smith	https://www.kaggle.com/joshsmith21/fitbit-dataset	2106	Fitbit	1	51	Present	Absent	Present	A
6	More than 4 years of steps and sleep data. Mi band	Damir Gadylyaev	https://www.kaggle.com/damirgadylyaev/more-than-4-years-of-steps-and-sleep-data-mi-band	2016-2020	Xiaomi Mi Band	1	1570	Present	Present	Absent	F
7	MI FitBit Dataset	Parul Garg	https://www.kaggle.com/parulgarg123/mi-fitbit-dataset	2018-2019	Mi Fitbit	1	269	Present	Present	Present	A
8	Exported data from Xiaomi Mi Band fitness tracker	Bekbolat Kuralbayev	https://www.kaggle.com/bekbolsky/exported-data-from-xiaomi-mi-band-fitness-tracker	2017-2019	Xiaomi Mi Band	1	385	Present	Present	Present	A

图 5-5　其他健康数据

5.1.4　案例：健康数据获取与可视化

利用 Kaggle 上的数据集服务，通过网站链接的方式下载了小米手环记录的步数和睡眠记录，保存文件命名为 Steps.csv。该数据集包含了 2454 条 2016 年 4 月 27 日—2023 年 1 月 14 日的健康数据集，每一条数据均包括日期(date)、步数(steps)、距离(distance)、跑步距离(runDistance)和卡路里(calories)，数据的分布情况如图 5-6 所示。本节的可视化方法数据集均使用“./5.1.4/Steps.csv”。

date	steps	distance	runDistance	calories
2016-04-27	4948	3242	46	281
2016-04-28	16573	12060	79	751
...	...	...	...	...
2023-01-13	4508	3169	2578	193
2023-01-14	2104	1408	1176	111

图 5-6　数据集中的数据分布情况

数据可视化有众多展现方式，不同的数据类型要选择适合的展现方法。常见的可视化图表类型包括：直方图、柱状图、条形图、折线图、饼图、散点图和箱形图。

1. 直方图

直方图，又称作质量分布图，它是由一系列高度不等的纵向条纹或线段表示数据分布的情况，一般用横轴表示数据的类型，纵轴表示分布情况。直方图可以利用方块的高度来反映数据的差异，只适用于中小规模的数据集，不适用于大规模的数据集。

直方图主要应用在定量数据的可视化场景中，或者用来进行连续型数据的可视化展示。比如，公共英语考试分数的区间分布、抽样调查中的人均寿命的分布特征以及居民可支配收入的分布特征。

为了构建直方图，第一步是将值的范围分段，即将整个值的范围分成一系列间隔，然后计算每个间隔中有多少值。这些值通常被指定为连续的、不重叠的变量间隔。间隔必须相邻，并且通常是相等的大小。代码如下所示。

针对 Steps.csv 数据集构建的直方图如图 5-7 所示。

```
import pandas as pd
import matplotlib.pyplot as plt
df = pd.read_csv('Steps.csv', header = None)
data = df.iloc[:, :2]
df = pd.DataFrame(data)
#取 steps 列的数据
# 将列转换为数值型
df[1] = pd.to_numeric(df[1], errors = 'coerce')
# 绘制直方图
plt.hist(df[1], bins = 10, edgecolor = 'black')
# 设置图形标题和坐标轴标签
plt.title('Steps Distribution')
plt.xlabel('Steps')
plt.ylabel('Frequency')
# 给每个柱形加上数量标签
for rect in plt.gca().patches:
    x = rect.get_x() + rect.get_width() / 2
    y = rect.get_height()
    plt.text(x, y, f'{int(y)}', ha = 'center', va = 'bottom')
# 设置横轴的显示范围增大
plt.xlim(df[1].min() - 10, df[1].max() + 10)
# 显示图形
plt.show()
```

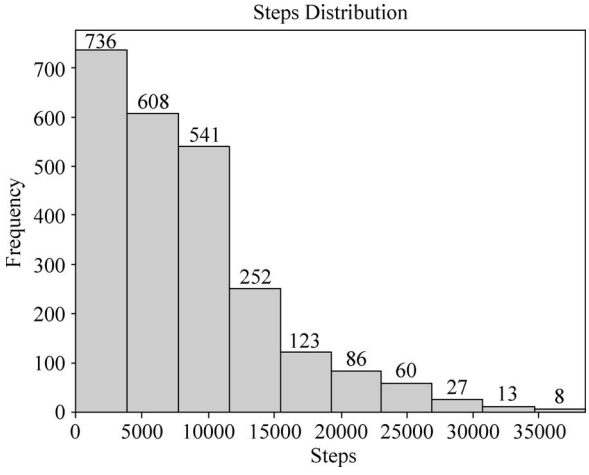

图 5-7　直方图统计步数分布结果

2. 柱状图

柱状图，是实际工作中最常使用的图表类型之一。它通过垂直的条形展示维度字段的分布情况，直观地反映一段时间内各项数据的变化，在数据统计和销售报表中被广泛应用。

柱状图主要应用在定性数据的可视化场景中，或者离散型数据的分布展示。例如，一个本科班级的学生的籍贯分布，出国旅游人士的职业分布以及下载一款 APP 产品的操作系统的分布。

直方图与柱状图的区别：一方面，直方图和柱状图在展示效果上是非常类似的，只是直方图描述的是连续型数据的分布，柱状图描述的是离散型数据的分布，也可以讲：一个是描述定量数据；另一个是描述定性数据。另一方面，从图形展示效果来看，柱状图的柱体之间有空隙，直方图的柱体之间没有空隙。代码如下所示，柱状图的结果如图 5-8 所示。

```
import pandas as pd
import matplotlib.pyplot as plt
df = pd.read_csv('Steps.csv')
data = df.iloc[:20, :2]
df = pd.DataFrame(data)
# 绘制柱状图
plt.bar(df['date'], df['steps'], width = 0.8, edgecolor = 'black')
# 设置图形标题和坐标轴标签
plt.title('Steps Distribution')
plt.xlabel('Date')
plt.ylabel('Steps')
# 给每个柱体加上数量标签
for i, v in enumerate(df['steps']):
    plt.text(i, v + 100, str(v), ha = 'center', va = 'bottom',fontsize = 8)
# 设置横轴标签倾斜显示以避免重叠
plt.xticks(rotation = 80)
# 调整图形尺寸以适应横轴标签
plt.tight_layout()
# 显示图形
plt.show()
```

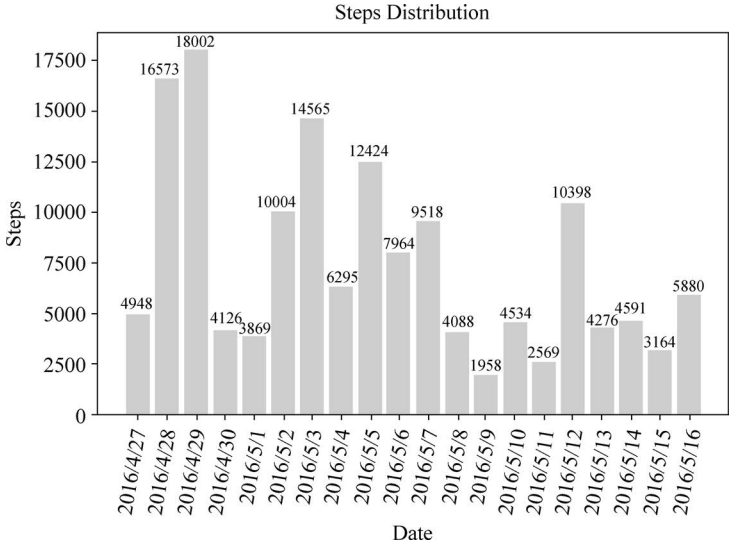

图 5-8　柱状图统计步数分布结果

3. 条形图

条形图，与柱形图类似，几乎可以表达相同多的数据信息。在条形图中，用宽度相同的条形的高度或者长短来表示数据的多少。通过条形的长短，可以比较不同组别数据分布的

差异情况。

与柱状图不同的是条形图主要突出数值的差异，而淡化时间的差异。尤其在项目名称较长以及数量较多时，采用条形图可视化数据会更加美观、清晰。条形图的展示类型与样式与柱状图类似。代码如下所示，条形图结果如图 5-9 所示。

```
import pandas as pd
import matplotlib.pyplot as plt
df = pd.read_csv('Steps.csv')
data = df.iloc[:20, :2]
df = pd.DataFrame(data)
# 绘制水平条形图
plt.barh(df['date'], df['steps'], height = 0.8, edgecolor = 'black')
# 设置图形标题和坐标轴标签
plt.title('Steps Distribution')
plt.xlabel('Steps')
plt.ylabel('Date')
# 给每个柱体加上数量标签
for i, v in enumerate(df['steps']):
    plt.text(v + 100, i, str(v), ha = 'left', va = 'center', fontsize = 8)
# 调整图形尺寸以适应纵轴标签
plt.tight_layout()
# 显示图形
plt.show()
```

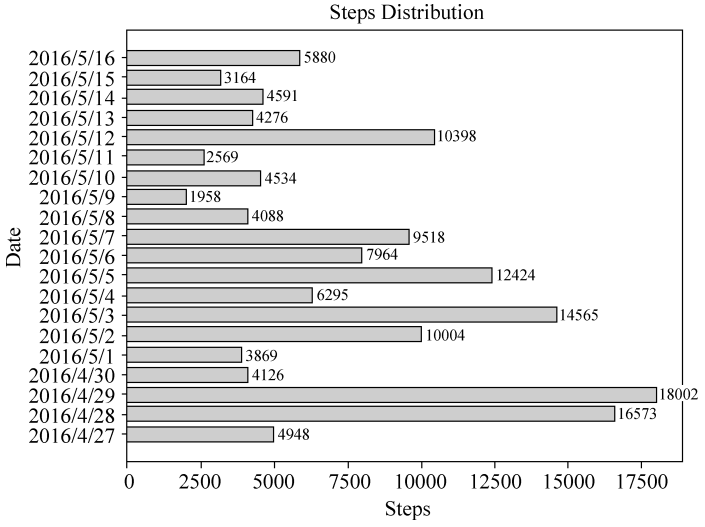

图 5-9　医疗卫生机构水平条形图

4. 折线图

折线图，是用直线段将各数据点连接起来而组成的图形，以折线的方式显示数据的变化趋势。折线图主要用来表示数据的连续性和变化趋势，也可以显示相同的时间间隔内数据的预测趋势。强调的是数据的时间性和变动率，而不是变量。

要绘制折线图，先在笛卡儿坐标系上定出数据点，然后用直线把这些点连接起来。折线图主要应用于时间序列数据的可视化，适合二维的大数据集，尤其那些趋势比单个数据点更重要的数据。代码如下所示，折线图如图 5-10 所示。

```
import pandas as pd
import matplotlib.pyplot as plt
df = pd.read_csv('Steps.csv')
data = df.iloc[:20, :2]
df = pd.DataFrame(data)
# 绘制折线图
plt.plot(df['date'], df['steps'], marker = 'o', linestyle = '-', color = 'b')
# 设置图形标题和坐标轴标签
plt.title('Steps Distribution')
plt.xlabel('Date')
plt.ylabel('Steps')
# 给每个点添加数量标签
for i, v in enumerate(df['steps']):
    plt.text(df['date'][i], v + 100, str(v), ha = 'center', va = 'bottom', fontsize = 7)
# 设置横轴标签倾斜显示以避免重叠
plt.xticks(rotation = 80)
# 调整图形尺寸以适应横轴标签
plt.tight_layout()
# 显示图形
plt.show()
```

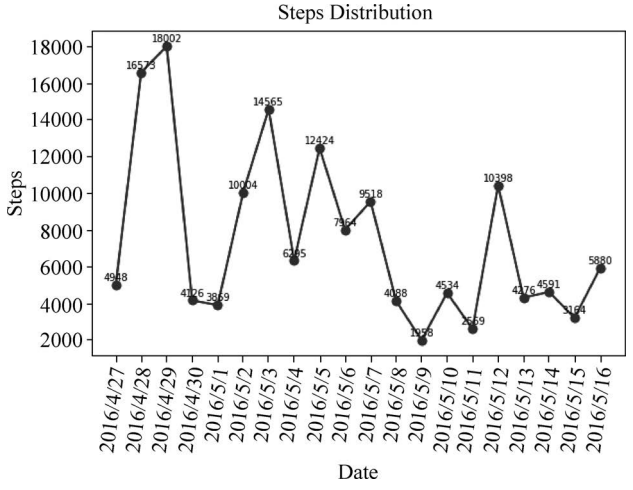

图 5-10 折线图统计步数分布

5. 饼图

饼图，可以显示一个数据序列中各项的大小与各项总和的比例，每个数据序列具有唯一的颜色或图形，并且与图例中的颜色是相对应的。饼图可以很清晰地反映出各数据系列的百分比情况。

饼图主要应用在定性数据的可视化场景中，或者是用来进行离散型数据的比例展示。如果需要展示参加硕士研究生考试的性别比例、某市一年中四季使用天然气用量的比重以及家庭生活开支用途的比例分布，这些场景都可以使用饼图进行数据的可视化，通过绘制饼图，就可以直观地反映研究对象定性数据的比例分布情况。

在绘制饼图前一定要注意把多个类别按一定的规则排序，推荐将饼图的最大占比部分放置在钟表的 12 点位置的右边，以强调其重要性。再将第二大占比部分设置在 12 点位置的左边，剩余的类别则按逆时针方向放置。代码如下所示，饼图如图 5-11 所示。

```
import pandas as pd
import matplotlib.pyplot as plt
import calendar
df = pd.read_csv('Steps.csv')
data = df.iloc[:, :2]
df = pd.DataFrame(data)
# 提取月份信息
df['month'] = pd.to_datetime(df['date']).dt.month
# 将月份数字转换为英文简写
df['month'] = df['month'].apply(lambda x: calendar.month_abbr[x])
# 按月份分组,并计算每个月的步数总和
monthly_steps = df.groupby('month')['steps'].sum()
# 绘制饼图
plt.pie(monthly_steps, labels = monthly_steps.index, autopct = '%.1f%%')
# 设置图形标题
plt.title('Steps Distribution by Month')
# 显示图形
plt.show()
```

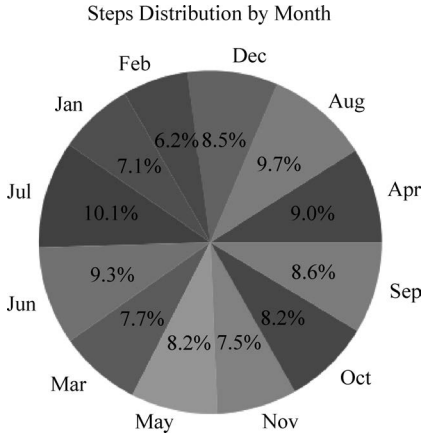

图 5-11　饼图统计步数分布

6. 散点图

散点图,是指数据点在直角坐标系平面上的分布图,通常用于比较跨类别的数据。散点图包含的数据点越多,比较的效果就会越好。散点图中每个坐标点的位置是由变量的值决定的,用于表示因变量随自变量而变化的大致趋势,以判断两种变量的关系与相关性。散点图可以提供 3 类关键信息:①变量之间是否存在数量关联趋势;②如果存在关联趋势,那么是线性还是非线性的;③观察是否存在离群值,从而分析这些离群值对建模分析的影响。

散点图类似于折线图,它可以显示单个或者多个数据系列的数据在某时间间隔下的变化趋势。通常用于科学数据的表达、实验数据的拟合和趋势的预测等。代码如下所示,散点图如图 5-12 所示。

```
import pandas as pd
import matplotlib.pyplot as plt
df = pd.read_csv('Steps.csv')
data = df.iloc[:20, :2]
df = pd.DataFrame(data)
# 绘制散点图
plt.scatter(df['date'], df['steps'])
# 设置横轴和纵轴标签
```

```
plt.xlabel('Date')
plt.ylabel('Steps')
# 设置标题
plt.title('Steps by Date')
# 旋转横轴刻度标签,以便更好地显示日期
plt.xticks(rotation = 45)
# 添加数字标签
for i, row in df.iterrows():
    plt.annotate(row['steps'], (row['date'], row['steps']) ,fontsize = 6)
# 显示图形
plt.show()
```

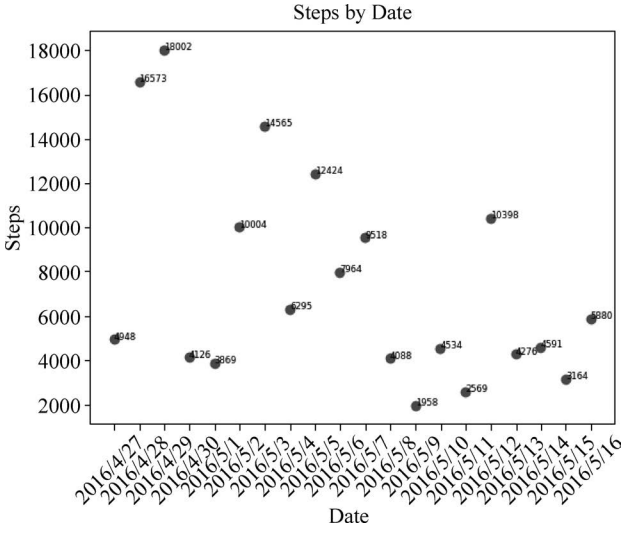

图 5-12 散点图统计步数分布

7. 箱形图

箱形图,又称为盒须图、盒式图或箱线图,能显示出一组数据的最大值、最小值、中位数、及上下四分位数,可以用来反映一组或多组连续型定量数据分布的中心位置和散布范围。

箱形图由一个箱体和一对箱须所组成,箱体由第一四分位数、中位数(第二四分位数)和第三四分位数所组成。在箱须的末端之外的数值可以理解成离群值,因此,箱须是对一组数据范围的大致直观描述。

箱形图主要应用在一系列测量或观测数据的比较场景中,例如,学校间或班级间测试成绩的比较,球队中的队员体能对比,产品优化前后的测试数据比较以及同类产品的各项性能的比较等,都可以借助箱形图来完成相关分析或研究任务。因此,箱形图的应用范围非常广泛,而且实现起来也非常简单。代码如下所示,箱形图如图 5-13 所示。

```
import pandas as pd
import matplotlib.pyplot as plt
df = pd.read_csv('Steps.csv')
# 提取需要绘制箱形图的列
columns_to_plot = ['steps', 'distance', 'runDistance', 'calories']
# 绘制箱形图
plt.boxplot(df[columns_to_plot].values)
# 设置横轴标签
plt.xticks(range(1, len(columns_to_plot) + 1), columns_to_plot)
```

```
# 设置纵轴标签
plt.ylabel('Value')
# 设置标题
plt.title('Boxplot of Activity Metrics')
# 显示图形
plt.show()
```

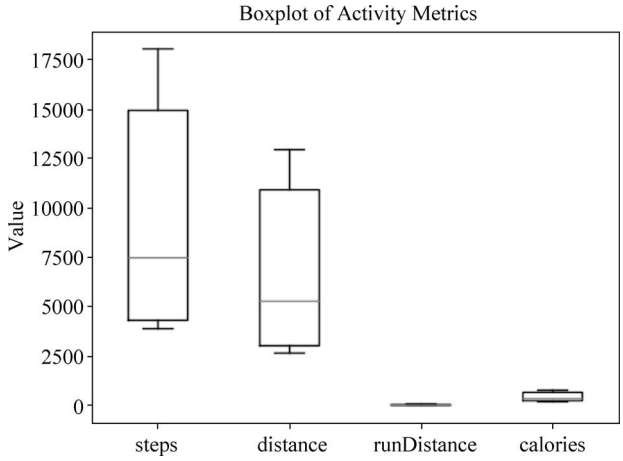

图 5-13　箱形图展示运动健康数据分布情况

5.2　移动健康数据可视化大屏设计

数据可视化是指将数据以图表的形式表示，并利用数据分析和开发工具发现其中未知信息的处理过程。使用可视化工具对数据进行可视化有许多优势：

（1）更直观的理解：可视化将抽象的数据转化为图形、图表和图像，使数据变得更加直观和易于理解。人们通常更容易通过可视化捕捉到数据之间的关系、趋势和模式，而不是从原始数据表格中。

（2）发现隐藏的信息：可视化可以揭示数据中的隐藏信息和趋势，帮助用户发现在原始数据中可能被忽略的关键见解。这有助于作出更准确的决策和预测。

（3）交互性：可视化工具通常提供交互式功能，允许用户探索数据，根据需求放大、缩小、筛选和排序。这种交互性能够使用户更深入地探索数据，并且让用户可以根据兴趣进行个性化的分析。

（4）有效的沟通工具：可视化可以将复杂的数据和分析结果以简洁的方式传达给非专业人士。这对于向非技术人员、利益相关者或团队成员传递信息和见解非常有帮助。

（5）快速的洞察力：可视化工具能够快速生成图表和图像，使数据分析变得高效。用户可以快速地创建多种类型的图表，以便更快地获取洞察力。

（6）跨数据源比较：如果您拥有来自不同来源的数据，可视化可以将它们融合到一个图表或图形中，以便进行比较和分析。这有助于发现不同数据源之间的关联性和差异。

（7）支持决策制定：可视化工具可以帮助决策者更好地理解问题和机会，从而作出更明智的决策。它们可以将复杂的情况可视化，使决策制定变得更加有根据。

(8) 提高报告和演示质量：在报告、演示和展示中使用可视化可以提高内容的吸引力和影响力。图表和图像能够更生动地展示信息，使受众更易于记住和理解。

总之，可视化工具能够将数据转化为易于理解和分析的形式，帮助用户更好地理解数据、发现见解并作出更明智的决策。无论是在数据分析、报告制作、教育还是决策制定方面，可视化都是一个强大的工具。对移动健康数据的可视化大屏设计流程如图 5-14 所示。

图 5-14　可视化大屏设计流程

5.2.1　可视化工具库

目前有许多流行的可视化工具库可供选择，用于创建各种类型的数据可视化。其中包含 Matplotlib，它是 Python 中最常用的数据可视化库之一，用于创建静态、交互式和动画图表；Seaborn 也是基于 Python 的库，建立在 Matplotlib 之上，专注于更美观的统计图表；Pyecharts 是 Python 与 ECharts 结合的产物，封装了 ECharts 各类图表的基本操作，然后通过渲染机制，输出一个包含 JS 代码的 HTML 文件。

1. Matplotlib

Matplotlib 是一个非常强大的 Python 画图工具，使用该工具可以将很多数据通过图表的形式更直观地呈现出来。Matplotlib 可以绘制线图、散点图、等高线图、条形图、柱状图、3D 图形，甚至图形动画等各种静态、动态、交互式的图表。要想使用 Matplotlib 绘制图表，需要先导入绘制图表的模块 pyplot，该模块提供了一种类似 MATLAB 的绘图方式，主要用于绘制简单或复杂的图形。Matplotlib 通常的调入形式是 import matplotlib.pyplot as plt。

使用 pyplot 绘图的一般过程：首先生成或读入数据，然后根据实际需要绘制二维折线图、散点图、柱状图、饼状图、雷达图或三维曲线、曲面、柱状图等，接下来设置坐标轴标签(xlabel、ylabel 函数)、坐标轴刻度线(xticks、yticks 函数)、图例(legend 函数)、标题(title 函数)等图形属性，最后显示或保存绘图结果。下面利用 Matplotlib 工具绘制玫瑰图，所利用的数据集为“./5.2.1/Steps.csv”，代码如下所示。绘制好的玫瑰图如图 5-15 所示。

```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler
import numpy as np
df = pd.read_csv('Steps.csv')
# 提取需要绘制箱形图的列
columns_to_plot = ['steps', 'distance', 'runDistance', 'calories']
# 创建 MinMaxScaler 对象进行归一化
scaler = MinMaxScaler()
# 归一化数据
normalized_values = scaler.fit_transform(df[columns_to_plot])
# 计算每个度数的平均值
mean_values = np.mean(normalized_values, axis = 0)
# 将度数转换为弧度
theta = [i * 2 * np.pi / len(columns_to_plot) for i in range(len(columns_to_plot))]
# 定义颜色映射
```

```
cmap = plt.get_cmap('rainbow')
# 绘制玫瑰图
ax = plt.subplot(111, polar = True)
bars = ax.bar(theta, mean_values, width = 0.4)
# 设置每个柱子的颜色和标签
for bar, column, angle in zip(bars, columns_to_plot, theta):
  color = cmap(angle / (2 * np.pi))
  bar.set_facecolor(color)
  bar.set_label(column)
# 设置角度标签
ax.set_xticks(theta)
ax.set_xticklabels(columns_to_plot)
# 添加图例
ax.legend()
# 设置标题
plt.title('Rose Plot of Normalized Activity Metrics')
# 显示图形
plt.show()
```

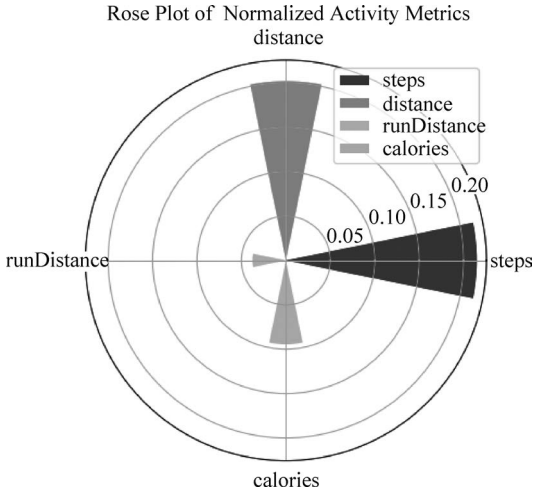

图 5-15　玫瑰图展示运动健康数据分布情况

从图 5-15 可看出，小米手环运动数据集数据的维度较少，玫瑰图不能展示出很好的效果。因此利用 5.2.2 节中处理好的医疗卫生数据进行维度的增加和图表显示效果的优化，所用到的数据集为“./5.2.1/all_data.json”，代码如下所示，优化后的玫瑰图如图 5-16 所示。

```
# 玫瑰图
import json
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import matplotlib as mpl
import json
mpl.rcParams["font.sans-serif"] = ["SimHei"]          # 用来正常显示中文标签
mpl.rcParams["axes.unicode_minus"] = False            # 用来正常显示负号
def rosetype_pie(country,confirmed,size,colors):
  num = len(size)                                     # 柱子的数量
  width = 2 * np.pi / num                             # 每个柱子的宽度
  rad = np.cumsum([width] * num)                      # 每个柱子的角度
```

```
    plt.figure(figsize = (6,6),dpi = 500)                    # 创建画布
    ax = plt.subplot(projection = 'polar')
    ax.set_ylim( - 1, np.ceil(max(size) + 1))                # 中间空白, - 1 为空白半径大小,可自行调整
    ax.set_theta_zero_location('N', - 5.0)                   # 设置极坐标的起点方向 W,N,E,S, - 5.0 为偏离数值,可自行调整
    ax.set_theta_direction(1)                                # 1 为逆时针, - 1 为顺时针
    ax.grid(False)                                           # 不显示极轴
    ax.spines['polar'].set_visible(False)                    # 不显示极坐标最外的圆形
    ax.set_yticks([])                                        # 不显示坐标间隔
    ax.set_thetagrids([])                                    # 不显示极轴坐标
    ax.bar(rad, size, width = width, color = colors, alpha = 1)      # 画图
    ax.bar(rad, 1, width = width, color = 'white', alpha = 0.15)     # 中间添加白色色彩使图案变浅
    ax.bar(rad, 3, width = width, color = 'white', alpha = 0.1)      # 中间添加白色色彩使图案变浅
    ax.bar(rad, 5, width = width, color = 'white', alpha = 0.05)     # 中间添加白色色彩使图案变浅
    ax.bar(rad, 7, width = width, color = 'white', alpha = 0.03)     # 中间添加白色色彩使图案变浅
    # 设置 text
    for i in np.arange(num):
      if i < 8:
        ax.text(rad[i],                                              # 角度
            size[i] - 0.2,                                           # 长度
            country[i] + '\n' + str(confirmed[i]) + '个',            # 文本
            rotation = rad[i] * 180 / np.pi - 5,                     # 文字角度
            rotation_mode = 'anchor',
            # alpha = 0.8,                                           # 透明度
            fontstyle = 'normal',       # 设置字体类型,可选参数[ 'normal' | 'italic' | 'oblique' ],italic 斜体,oblique 倾斜
            fontweight = 'black',       # 设置字体粗细,可选参数 ['light', 'normal', 'medium', 'semibold', 'bold', 'heavy', 'black']
            color = 'white',                                         # 设置字体颜色
            size = size[i]/4.4,                                      # 设置字体大小
            ha = "center", # 'left','right','center'
            va = "top", # 'top', 'bottom', 'center', 'baseline', 'center_baseline'
            )
      elif i < 15:
        ax.text(rad[i] + 0.02,
            size[i] - 0.7,
            country[i] + '\n' + str(confirmed[i]) + '个',
            fontstyle = 'normal',
            fontweight = 'black',
            color = 'white',
            size = size[i] / 3.2,
            ha = "center",
            )
      else:
        ax.text(rad[i],
            size[i] + 0.1,
            str(confirmed[i]) + '例 ' + country[i],
            rotation = rad[i] * 180 / np.pi + 85,
            rotation_mode = 'anchor',
            fontstyle = 'normal',
            fontweight = 'black',
            color = 'black',
            size = 2,
```

```
            ha = "left",
            va = "bottom",
            )
    plt.title("全国医疗卫生机构数统计情况", size = 5)
      # verticalalignment 设置水平对齐方式 ,可选参数 : 'center' , 'top' , 'bottom' , 'baseline
    plt.show()
if __name__ == '__main__':
    filename = 'D:\\all_data.json'  # JSON 文件路径
    with open(filename, 'r', encoding = 'utf - 8') as f:
      data = json.load(f)
    words = []
    for province, province_data in data.items():
      for indicator, years_data in province_data.items():
        if indicator == "医疗卫生机构数(个)":
          max_value = max(years_data.values())
          max_year = [year for year, value in years_data.items() if value == max_value]
          words.append((province, max_value))
    sorted_words = sorted(words, key = lambda x: x[1], reverse = False)  # 根据数据值对柱子进行排序(升序)
    df = pd.DataFrame(sorted_words, columns = ['省份', '数据'])
    country = list(df['省份'])
    confirmed = list(df['数据'])
    # df = pd.read_csv('WHO - COVID - 19 - global - table - data.csv')
    # df = df.reset_index()
    # df = df.drop(0)
    # df['name'] = df['index']
    # df['nowconfirm'] = df['Cases - newly reported in last 7 days per 100000 population']
    # df = df.sort_values('nowconfirm',ascending = False).head(30)
    # world_name = pd.read_excel("中英文对照表.xlsx",engine = 'openpyxl')
    # data = pd.merge(df, world_name, left_on = "index", right_on = "英文", how = "inner")
    # country = list(data['中文'])
    # confirmed = list(data['nowconfirm'])
    colors = [(0.68359375, 0.02734375, 0.3203125),(0.78125, 0.05078125, 0.2578125),
(0.875, 0.0390625, 0.1796875),
              (0.81640625, 0.06640625, 0.0625),(0.8515625, 0.1484375, 0.08203125),
(0.90625, 0.203125, 0.13671875),
              (0.89453125, 0.2890625, 0.0703125),(0.84375, 0.2421875, 0.03125),
(0.9140625, 0.26953125, 0.05078125),
              (0.85546875, 0.31640625, 0.125),(0.85546875, 0.3671875, 0.1171875),
(0.94921875, 0.48046875, 0.28125),
              (0.9375, 0.51171875, 0.1484375),(0.93359375, 0.59765625, 0.0625),
(0.93359375, 0.62890625, 0.14453125),
             (0.86328125, 0.5859375, 0.15234375),(0.86328125, 0.71875, 0.16015625),
(0.86328125, 0.8203125, 0.16015625),
              (0.76171875, 0.8671875, 0.16015625),(0.53125, 0.85546875, 0.15625),
(0.4765625, 0.94140625, 0.0703125),
             (0.21484375, 0.91015625, 0.0625),(0.15234375, 0.88671875, 0.08203125),
(0.11328125, 0.87890625, 0.19921875),
              (0.11328125, 0.8125, 0.1796875),(0.1875, 0.76953125, 0.2109375),
(0.2109375, 0.78125, 0.38671875),
                (0.1484375, 0.76953125, 0.30859375),(0.22265625, 0.73046875,
 0.35546875),(0.2890625, 0.6875, 0.4765625)]        # 转化为小数的 rgb 色列表
size = [22, 19, 17, 12, 11, 10, 9, 8, 7.2, 7.0, 6.8, 6.6, 6.4, 6.2, 6.0,
       5.8, 5.6, 5.4, 5.2, 5.0, 4.8, 4.6, 4.4, 4.2, 4.0, 3.8, 3.6, 3.4, 3.2, 3.0]
                                                    # 自定义一个柱长度列
rosetype_pie(country, confirmed, size, colors)
```

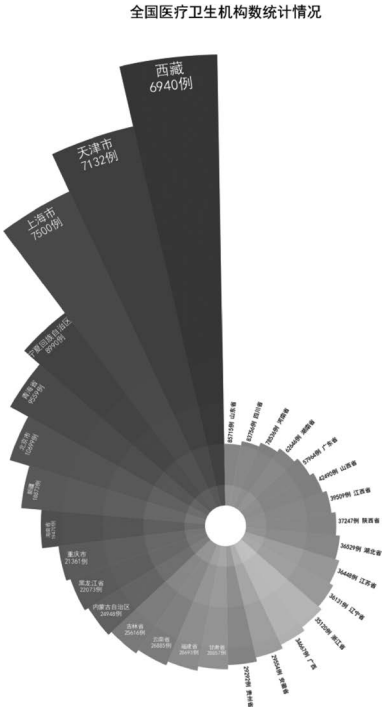

图 5-16　优化后的玫瑰图效果展示

2. Seaborn

Seaborn 同 Matplotlib 一样，也是 Python 进行数据可视化分析的重要的第三方包。但是在 Matplotlib 的基础上进行了更高级的 API 封装，从而使得作图更加容易，在大多数情况下，使用 Seaborn 能绘制具有吸引力的图表，而使用 Matplotlib 却能绘制出具有更多特色的图表。应该把 Seaborn 视为 Matplotlib 的补充，而不是替代物。Seaborn 在 Matplotlib 的基础上，侧重数据统计分析图表的绘制，包括带误差线的柱形图、散点图、箱形图、小提琴图、一维和二维的统计直方图和核密度估计图等。Seaborn 的默认导入语句为 import seaborn as sns。接下来利用小米手环运动数据集，即“./5.2.1/Steps.csv”进行小提琴图的绘制，代码如下所示，小提琴图的绘制结果如图 5-17 所示。

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import MinMaxScaler
df = pd.read_csv('Steps.csv')
# 提取需要绘制小提琴图的列
columns_to_plot = ['steps', 'distance', 'runDistance', 'calories']
# 创建 MinMaxScaler 对象进行归一化
scaler = MinMaxScaler()
normalized_values = scaler.fit_transform(df[columns_to_plot])
# 将归一化后的数据转换回 DataFrame
normalized_df = pd.DataFrame(normalized_values, columns = columns_to_plot)
# 绘制小提琴图
sns.violinplot(data = normalized_df,width = 1)
# 设置标题和坐标轴标签
plt.title('Normalized Violin Plot of Activity Metrics')
```

```
plt.xlabel('Activity')
plt.ylabel('Normalized Value')
# 显示图形
plt.show()
```

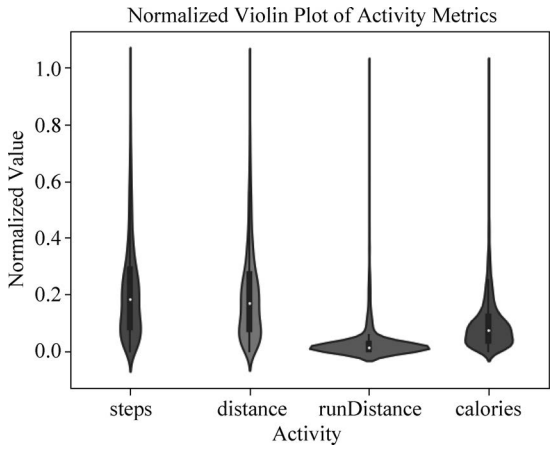

图 5-17　小提琴图显示运动健康数据分布情况

3. Pyecharts

Pyecharts 是一个用于创建交互式图表的 Python 库，它基于 ECharts(百度开发的一个强大的 JavaScript 图表库)开发而来。ECharts 可以生成各种类型的图表，包括折线图、柱状图、饼图、散点图等，并支持动态数据更新、数据联动、数据可视化等特性。Pyecharts 通过将 Python 数据转换为 ECharts 所需的 JSON 格式来创建图表。它提供了一组简单的 API，使用户能够轻松地创建各种图表，而无需深入了解 JavaScript。Pyecharts 提供了一种在 Python 中创建交互式数据可视化图表的方便方式，适用于数据分析、报告制作、演示等多种应用场景。接下来利用小米手环运动数据集，即“./5.2.1/Steps.csv”进行图表绘制，代码如下所示，其中用到了 grid 函数对图标进行组合显示，组合图的绘制结果如图 5-18 所示。

```
import pandas as pd
from pyecharts.charts import Line, Bar
from pyecharts import options as opts
from pyecharts.globals import ThemeType
from pyecharts.charts import Grid
# 读取 CSV 文件
df = pd.read_csv('Steps.csv')
# 提取数据，截断时间至 2016 年
df['date'] = pd.to_datetime(df['date'])
df = df[df['date'].dt.year <= 2016]
x_data = df['date']
y_data = df['steps']
# 折线图
line = (
  Line(init_opts = opts.InitOpts(theme = ThemeType.LIGHT))
  .add_xaxis(x_data.dt.strftime('%Y-%m-%d').tolist())  # 格式化日期为年月日
  .add_yaxis("折线图步数", y_data.tolist(), itemstyle_opts = opts.ItemStyleOpts(color = "blue"))
  .set_series_opts(label_opts = opts.LabelOpts(is_show = False))
  .set_global_opts(legend_opts = opts.LegendOpts(pos_left = "10%"))  # 设置折线图图例位置
```

```
)
# 柱状图
bar = (
  Bar(init_opts = opts.InitOpts(theme = ThemeType.LIGHT))
  .add_xaxis(x_data.dt.strftime('%Y-%m-%d').tolist())    # 格式化日期为年月日
  .add_yaxis("柱状图步数", y_data.tolist(), itemstyle_opts = opts.ItemStyleOpts(color = "green"))
  .set_series_opts(label_opts = opts.LabelOpts(is_show = False))
  .set_global_opts(legend_opts = opts.LegendOpts(pos_left = "35%"))  # 设置柱状图图例位置
)
# 使用 Grid 组合折线图和柱状图
grid = (
  Grid()
  .add(bar, grid_opts = opts.GridOpts(pos_bottom = "65%", pos_left = "70%", pos_right = "65%"))
  .add(line, grid_opts = opts.GridOpts(pos_left = "15%"))
)
# 渲染图表
grid.render_notebook()
```

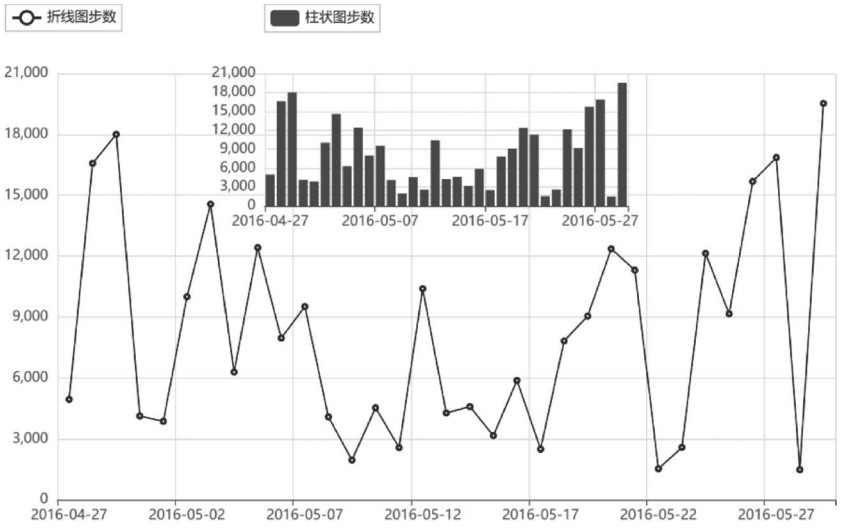

图 5-18　利用 Pyecharts 实现组合图显示

5.2.2　案例：搭建医疗卫生数据可视化大屏

本节中将利用 Pyecharts 框架实现医疗卫生数据的大屏可视化，数据来源为 https://data.stats.gov.cn/easyquery.htm?cn=E0103，单击“卫生”→“医疗卫生机构”，下载最近 20 年中国各省市的医疗卫生机构数据，存入名为 country_data 的文件夹中。经过数据预处理后，保存处理好的 json 文件为 all_data.json。上述数据集的位置为“./5.2.2/all_data.json”，接下来利用 Pyecharts 框架实现可视化大屏，详细过程如下所述。

1. 数据预处理

将下载好的数据经过数据预处理存为 JSON 文件的格式，命名为 all_data.json，便于后期画图的使用，代码如下所示，JSON 文件结构示例如图 5-19 所示。

```
import pandas as pd
import numpy as np
```

```
import json
import os
# 存储所有数据的字典
all_data = {}
# 数据文件夹路径
data_folder = r"country_data"
# 遍历所有的 XLS 格式文件
for filename in os.listdir(data_folder):
    if filename.endswith('.xls'):
        # 读取 XLS 格式文件
        file_path = os.path.join(data_folder, filename)
        df = pd.read_excel(file_path, header = None)
        # print(df)
        # 获取地区信息(第 2 行)
        region_info = str(df.iloc[1, 0])
        # print(region_info)
        # 提取地区名称
        region = region_info.split(":")[1].strip()
        # 提取需要的数据(从第 5 行开始,指标名称在第一列,年份在后续列)
        data = {}
        for row in range(4, 20):  # 只处理前 20 行数据
            indicator_name = df.iloc[row, 0]
            years_data = df.iloc[row, 1:].fillna(0).tolist()  # 将 NaN 值替换为 0
            data[indicator_name] = dict(zip(df.iloc[3, 1:].tolist(), years_data))
        # 更新总数据字典
        all_data[region] = data
# 保存为 JSON 文件
output_file_path = "all_data.json"  # 修改为您想要保存的 JSON 文件路径
with open(output_file_path, "w", encoding = "utf - 8") as f:
    json.dump(all_data, f, ensure_ascii = False, indent = 4)
# 读取 JSON 文件
filename = 'all_data.json'  # JSON 文件路径
with open(filename, 'r', encoding = 'utf - 8') as f:
    data = json.load(f)
# 修改数据
data["广西"] = data.pop("广西壮族自治区")
data["西藏"] = data.pop("西藏自治区")
data["新疆"] = data.pop("新疆维吾尔自治区")
# 保存回文件
with open(filename, 'w', encoding = 'utf - 8') as f:
    json.dump(data, f, ensure_ascii = False)
```

地区：天津市
指标：医疗卫生机构数(个)
年份：2022年，值：0
年份：2021年，值：6076
…
年份：2004年，值：2560
年份：2003年，值：7132

图 5-19　JSON 文件结构示例

2. 柱状图

创建柱状图如图 5-20 所示，绘制方法是先通过对每个省市的所有指标值求和，选出值最大的省市，即河北省的各级医疗卫生机构不同年份的统计情况，在本节中为了显示更加美观，只选取了最近六年的数据进行展示。

该柱状图中工具箱包含“保存为图片”“还原”“数据视图”“区域缩放”“区域缩放还原”“切换为折线图”“切换为柱状图”“切换为堆叠”等，可以单击柱状图下方的 label 图标，图标显示色彩和显示空白分别可以实现“选择展示”和“不选择展示”的效果，柱状图还可以通过鼠标拖拉进行收缩。代码如下所示。

```
def bar():
    # 读取 JSON 文件
    filename = 'all_data.json'              # JSON 文件路径
    with open(filename, 'r', encoding = 'utf - 8') as f:
      data = json.loads(f.read())
    # 获取所有地区
    regions = list(data.keys())
    #指标映射
    indicator_mapping = {
    '医疗卫生机构数': '医疗卫生机构数(个)',
    '医院数': '医院数(个)',
    '综合医院数': '综合医院数(个)',
    '中医医院数': '中医医院数(个)',
    '专科医院数': '专科医院数(个)',
    '基层医疗卫生机构': '基层医疗卫生机构(个)',
    '社区卫生服务中心(站)数': '社区卫生服务中心(站)数(个)',
    '街道卫生院数': '街道卫生院数(个)',
    '乡镇卫生院数': '乡镇卫生院数(个)',
    '村卫生室数': '村卫生室数(个)',
    '门诊部(所)数': '门诊部(所)数(个)',
    '专业公共卫生机构数': '专业公共卫生机构数(个)',
    '疾病预防控制中心数': '疾病预防控制中心数(个)',
    '专科疾病防治院(所/站)数': '专科疾病防治院(所/站)数(个)',
    '妇幼保健院(所/站)数': '妇幼保健院(所/站)数(个)',
    '卫生监督所(中心)数': '卫生监督所(中心)数(个)'
    }
    # 按照每个省份所有指标值之和的准则选择地区
    max_sum = 0                             # 最大和初始化为 0
    selected_region = ''                    # 初始化选中的地区为空字符串
    for region in regions:
      region_sum = sum(sum(data[region][key].values()) for key in indicator_mapping.
values())
      if region_sum > max_sum:
        max_sum = region_sum
        selected_region = region
    # 输出选中的地区
    # print("选中的地区:", selected_region)
    years = ["2003 年", "2004 年", "2005 年", "2006 年", "2007 年", "2008 年",
"2009 年","2010 年", "2011 年",
        "2012 年", "2013 年", "2014 年", "2015 年", "2016 年", "2017 年", "2018 年",
"2019 年","2020 年", "2021 年"]
    years = years[ - 6:]                    # 只保留最后 6 个年份
    x_data = years
    y_data_bar = []
    y_data_line = []
    for indicator, key in indicator_mapping.items():
      values_bar = []
      values_line = []
      for year in years:
        try:
          value = data[selected_region][key][year]
        except KeyError:
```

```
            value = 0
          values_bar.append(value)
          values_line.append(round(value / 100, 2))  # 将值除以 100,并保留 2 位小数
        y_data_bar.append(values_bar)
        y_data_line.append(values_line)
    # 创建图表
    bar_chart = (
      Bar(init_opts = opts.InitOpts(theme = ThemeType.LIGHT))
      .add_xaxis(x_data)
      .set_global_opts(
        title_opts = opts.TitleOpts(title = f"{selected_region}的医疗卫生机构统计",
pos_top = "1 %"),  # 设置柱状图标题为选中地区的名称
        legend_opts = opts.LegendOpts(pos_top = "85 %",pos_left = "center", orient =
"horizontal",item_width = 8, item_height = 8, border_width = 0),
        toolbox_opts = opts.ToolboxOpts(
          orient = "horizontal",
          pos_top = "5 %",
          pos_left = "40 %",
          # pos_right = "10 %"
          item_size = 15  # 设置工具箱图标尺寸为 14
        ),
        datazoom_opts = [opts.DataZoomOpts(type_ = "inside", range_start = 0, range_end =
100)],
      )
      .set_series_opts(label_opts = opts.LabelOpts(is_show = False))
    )
    # 添加每个指标的数据到图中
    for i, indicator in enumerate(indicator_mapping.keys()):
      bar_chart.add_yaxis(indicator, y_data_bar[i], stack = f"stack_{i + 1}")
    return bar_chart
```

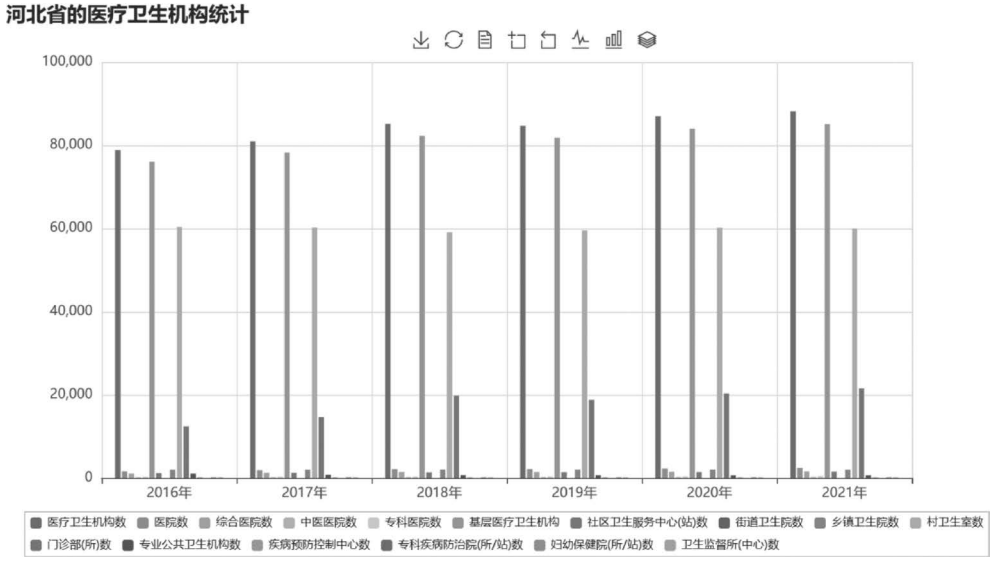

图 5-20　柱状图结果显示医疗卫生机构的统计情况

3. 饼图

创建饼图如图 5-21 所示,绘制方法是先计算“医疗卫生机构数(个)”单个指标历年的全国各省市数据的总和,按照 10 年为周期划分为一个双层饼图,在饼图中单击图标后分别呈

现彩色和白色可以实现“选择展示”和“选择不展示”的效果。代码如下所示。

```
def calculate_national_total(data):
    indicator_name = '医疗卫生机构数(个)'
    national_total = {}
    for region_data in data.values():
        for year, value in region_data[indicator_name].items():
            if year not in national_total:
                national_total[year] = 0
            national_total[year] += value
    return {'全国': {indicator_name: national_total}}
def create_double_layer_pie(data):
    indicator_name = '医疗卫生机构数(个)'
    region_data = data['全国'][indicator_name]
    years = list(region_data.keys())
    legend_opts = opts.LegendOpts(pos_bottom = "bottom", border_width = 0, item_width = 8, item_height = 8, pos_top = "85 % ")
    pie = (
        Pie()
        .set_global_opts(
            title_opts = opts.TitleOpts(
                title = f"{indicator_name}的双层饼图",
                pos_top = "top"
            ),
            legend_opts = legend_opts
        )
        .set_series_opts(label_opts = opts.LabelOpts(formatter = "{b}: {d} % "))
    )
    outer_radius = "75 % "  # 设置饼图显示半径
    inner_radius = "60 % "
    for i in range(0, len(years), 10):
        start_year = years[i]
        end_year = years[i + 9] if i + 9 < len(years) else years[ - 1]
        label = f"{start_year} - {end_year}"
        values = list(region_data.values())[i:i + 10]
        values.reverse()
        pie.add(
            series_name = label,
            data_pair = [(year, value) for year, value in zip(years[i:i + 10], values)],
            radius = [inner_radius, outer_radius]
        )
        inner_radius = str(int(inner_radius[: - 1]) - 10) + " % "
        outer_radius = str(int(outer_radius[: - 1]) - 10) + " % "
    return pie
def pie():
    # 读取 JSON 文件
    filename = 'all_data.json'  # JSON 文件路径
    with open(filename, 'r', encoding = 'utf - 8') as f:
        data = json.loads(f.read())
    # 计算全国数据总和
    data1 = calculate_national_total(data)
    # 生成双层饼图
    chart = create_double_layer_pie(data1)
    return chart
```

4. 热图

创建热图的方法是先计算全国各省市历年数据的总和，然后显示针对每个省市的数量分布，该热图可以利用鼠标滚轴实现放大或者缩小区域的功能，代码如下所示，热图显示

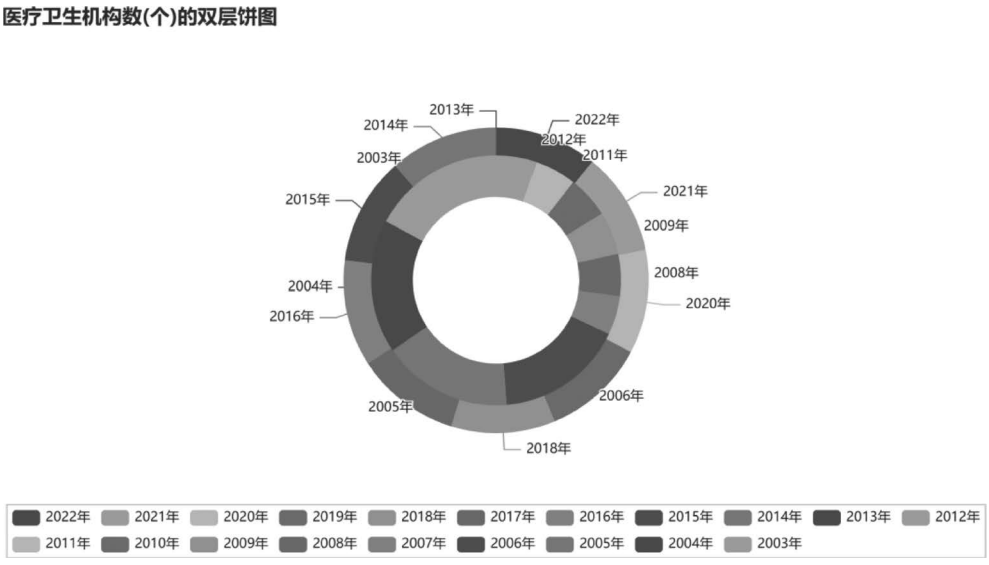

图 5-21　饼图结果显示全国每年的医疗卫生机构分布情况

结果。

```
def heat():
    # 读取 JSON 文件
    filename = 'all_data.json'  # JSON 文件路径
    with open(filename, 'r', encoding = 'utf - 8') as f:
        data = json.load(f)
    # 构造地区和数值的列表
    region_list = []
    value_list = []
    for region, region_data in data.items():
      region_list.append(region)
      value = sum([sum(years_data.values()) for years_data in region_data.values()])
      value_list.append(value)
    # 创建 Geo 实例并添加数据
    heat = (
      Geo()
      .add_schema(maptype = "china")
      .add(
        "地区数据",
        [list(z) for z in zip(region_list, value_list)],
        type_ = "heatmap",
        label_opts = opts.LabelOpts(formatter = "{b}: {c}"),
      )
      .set_series_opts(label_opts = opts.LabelOpts(is_show = True))
      .set_global_opts(visualmap_opts = opts.VisualMapOpts(), title_opts = opts.
TitleOpts(title = "中国地区数据"))
    )
    return heat
```

5. 词图

创建词图如图 5-22 所示，该图是针对“医疗卫生机构数(个)”单个指标在各省市的历年数据最大值的展示效果。代码如下所示。

```
def word():
    def create_word_cloud(data):
```

```
    words = []
    for province, province_data in data.items():
      for indicator, years_data in province_data.items():
        if indicator == "医疗卫生机构数(个)":
          max_value = max(years_data.values())
          # print(max_value)
          max_year = [year for year, value in years_data.items() if value == max_value]
          # 使用 f-string 格式化字符串
          # words.append((province, f"{max_year[0]} :{max_value}"))
          words.append((province, max_value))
    # print(words)
    wordcloud = (
      WordCloud()
      .add("", words, word_size_range=[10, 50])  #设置词的显示大小,进而控制画布上显示词语的数量
      .set_global_opts(title_opts=opts.TitleOpts(title="省份医疗卫生机构数词云图", pos_top="top", pos_left="left"))
    )
    return wordcloud
  # 读取 JSON 文件
  filename = 'all_data.json'  # JSON 文件路径
  with open(filename, 'r', encoding='utf-8') as f:
    data = json.load(f)
  chart = create_word_cloud(data)
  return chart
```

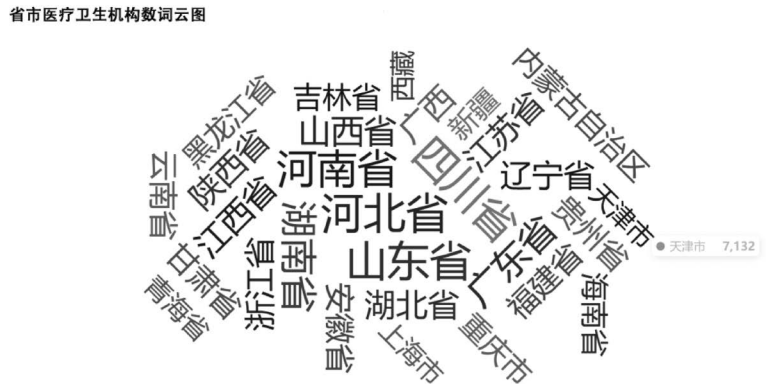

图 5-22 词图显示省市医疗卫生机构数量分布

6. 大屏格式分布设置

设置大屏的标题显示。代码如下。

```
def title(name,color):      #标题
  c = (Pie().
    set_global_opts(
    title_opts=opts.TitleOpts(title=name,pos_left='center',pos_top='center',
                    title_textstyle_opts=opts.TextStyleOpts(color=color,font_size=25))))
  return c
```

利用 page.add 方法将各个子图集成在一张画布上显示,但不设置显示格式。代码如下。

```
from pyecharts.charts import Geo, WordCloud
from pyecharts.globals import ThemeType
```

```
from pyecharts import options as opts
from pyecharts.faker import Faker
from pyecharts.render import make_snapshot
from pyecharts.globals import ThemeType
from snapshot_selenium import snapshot as driver
from pyecharts.charts import Page
# 创建一个空白的 Page 页面
page = Page()
page.add(
    title('医疗卫生机构数据可视化大屏', color='red'),
    heat(),
    pie(),
    bar(),
    word()
)
# 渲染为 HTML 字符串
html_content = page.render_embed()
# 添加 CSS 样式,为整个页面加边框
styled_html_content = f"""
<!DOCTYPE html>
<html>
<head>
<style>
  /* 设置边框样式 */
  body {{
    border: 5px solid #ccc;
    padding: 0;
    margin: 0;
  }}
  /* 设置页面内容样式 */
  .content {{
    padding: 300px;
  }}
</style>
</head>
<body>
  <div class="content">
    {html_content}
  </div>
</body>
</html>
"""
# 将 HTML 内容保存到文件中
with open("test.html", "w", encoding="utf-8") as f:
  f.write(html_content)
```

调整显示格式：通过调整 div 元素的样式调整子图的布局，使大屏的各个子图和子元素在画布上合理分布。代码如下。

```
from bs4 import BeautifulSoup
# 读取 HTML 文件并解析
with open("test.html", "r+", encoding='utf-8') as html:
  html_bf = BeautifulSoup(html, 'lxml')
  # 添加样式
  style_tag = html_bf.new_tag('style')
  style_tag.string = ".chart-container { background-color: black; }"
  html_bf.head.append(style_tag)
  # 获取所有 class 为 chart-container 的 div 元素
```

```
    divs = html_bf.select('.chart-container')
    # 修改第一个 div 的样式(标题)
    divs[0]['style'] = "width:100%;height:10%;position:absolute;top:0;left:0;border-style:solid;border-color:#444444;border-width:0px;"
    # 修改第二个 div 的样式(左上角)
    divs[1]["style"] = "width:50%;height:50%;position:absolute;top:10%;left:0;border-style:solid;border-color:#444444;border-width:0px;"
    # 修改第三个 div 的样式(右上角)
    divs[2]["style"] = "width:50%;height:40%;position:absolute;top:10%;right:0;border-style:solid;border-color:#444444;border-width:0px;"
    # 修改第四个 div 的样式(左下角)
    divs[3]["style"] = "width:50%;height:40%;position:absolute;bottom:0;left:0;border-style:solid;border-color:#444444;border-width:0px;"
    # 修改第五个 div 的样式(右下角)
    divs[4]["style"] = "width:50%;height:50%;position:absolute;bottom:0;right:0;border-style:solid;border-color:#444444;border-width:0px;"
# 将修改后的 HTML 写回文件
with open("test.html", "w", encoding='utf-8') as html:
    html.write(str(html_bf))
```

大屏制作过程中所用到的依赖的总结补充代码如下。

```
import pandas as pd
import numpy as np
import json
import os
from pyecharts.charts import WordCloud,Bar, Line, Page,Pie,Geo
from pyecharts import options as opts
from pyecharts.commons.utils import JsCode
from pyecharts.globals import ThemeType
from pyecharts.render import make_snapshot
```

调整完的可视化大屏的显示效果如图 5-23 所示。

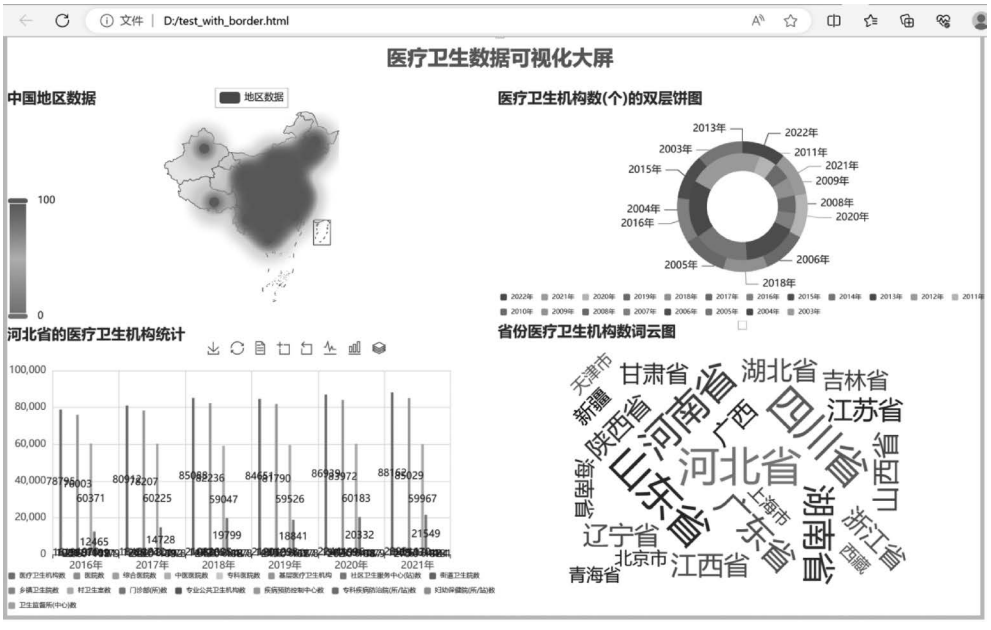

图 5-23 综合效果图

5.3　小米手环步态数据分析

基于下载好的小米手环步态数据，利用时间序列方法进行分析，主要通过自回归积分滑动平均模型(autoregressive integrated moving average model，ARIMA)和长短期记忆网络(long short-term memory，LSTM)对时序数据进行可视化、平稳性检验以及模型构建，最终实现预测。

5.3.1　案例：ARIMA 模型分析数据

1. ARIMA 模型原理

ARIMA 是一个时间序列分析方法，被广泛应用于时间序列数据的预测和建模。ARIMA 模型根据原序列是否平稳以及回归中所含部分的不同，包括移动平均过程(MA)、自回归过程(AR)、自回归移动平均过程(ARMA)以及 ARIMA 过程。ARIMA 模型的基本原理如式(5-1)所示。

$$\mathrm{ARIMA}(p,d,q)=\mathrm{AR}(p)+\mathrm{I}(d)+\mathrm{MA}(q) \tag{5-1}$$

式中，AR(p)表示自回归模型，p 为自回归阶数；I(d)表示差分模型，d 为时间序列成为平稳时所做的差分次数；MA(q)表示移动平均模型，q 为移动平均阶数。

自回归：ARIMA 模型基于自回归，即当前时间点的值与前面若干时间点的值有关。自回归阶数 p 表示当前时间点与前面 p 个时间点的值有关。

差分：为了消除时间序列数据的非平稳性，ARIMA 模型通常需要进行差分操作，即将原始数据转换为差分数据，消除趋势和季节性等影响。差分阶数 d 表示需要对时间序列数据进行差分的次数。

移动平均：ARIMA 模型基于移动平均，即当前时间点的值与前面若干时间点的误差有关。移动平均阶数 q 表示当前时间点与前面 q 个时间点的误差有关。

ARIMA 模型可以通过对时间序列数据进行分析和拟合，估计出合适的模型参数，从而进行数据预测和建模。

2. 实验过程

ARIMA 模型预测的基本程序包括以下步骤：

(1) 根据时间序列的散点图、自相关函数(autocorrelation function，ACF)图和偏自相关函数(partial autocorrelation function，PACF)图以单位根检验(augmented dickey-fuller，ADF)其方差、趋势及其季节性变化规律，对序列的平稳性进行识别。

(2) 对非平稳序列进行平稳化处理。如果数据序列是非平稳的，并存在一定的增长或下降趋势，则需要对数据进行差分处理；如果数据存在异方差，则需对数据进行技术处理，直到处理后的数据的自相关函数值和偏相关函数值无显著地异于零。

(3) 根据时间序列模型的识别规则，建立相应的模型。若平稳序列的 PACF 是截尾的，而 ACF 是拖尾的，可断定序列适合 AR 模型；若平稳序列的 PACF 是拖尾的，而 ACF 是截尾的，则可断定序列适合 MA 模型；若平稳序列的 PACF 和 ACF 均是拖尾的，则序列适合 ARMA 模型(截尾是指时间序列的 ACF 或 PACF 在某阶后均为 0 的性质；拖尾是 ACF 或 PACF 并不在某阶后均为 0 的性质)。

(4) 进行参数估计,检验是否具有统计意义。

(5) 进行假设检验,诊断残差序列是否为白噪声。

(6) 利用已通过检验的模型进行预测分析。

接下来是利用 ARIMA 模型分析小米手环步态数据的详细过程。

导入分析所需要的包,代码如下:

```
import pandas as pd
import datetime
import math
import numpy as np
import matplotlib.pylab as plt
from matplotlib.pylab import style
from scipy.special import logsumexp
import statsmodels.tsa.api as smtsa
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
from statsmodels.tsa.stattools import adfuller as ADF
from sklearn.metrics import mean_squared_error as mse, mean_absolute_error as mae
```

数据读取和预览,代码如下,结果如图 5-24 所示。

```
steps_df = pd.read_csv("Steps.csv",index_col = 0,parse_dates = [0])
#steps_df
plt.plot(steps_df['runDistance'])
plt.title('Daily runDistance')
plt.show()
```

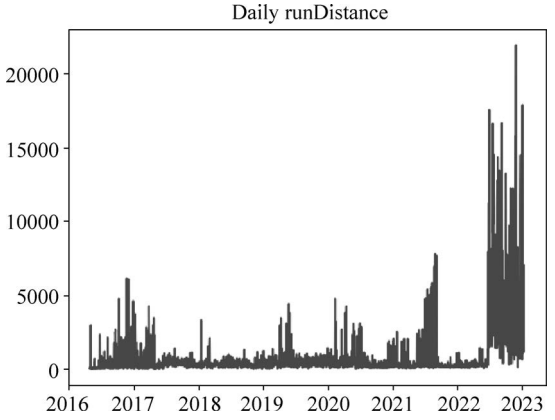

图 5-24 跑步距离的统计显示

数据预处理,缺失数据填充,代码如下,结果如图 5-25 所示。

```
#输出数据缺失值填充之前的数据
print(steps_df.runDistance['20160427':'20230114'])
#数据缺失值填充及可视化
#这里以天为单位,把缺失数据填充上 这里填充的是线性取值
steps_df = steps_df.resample('D').interpolate('linear')
print(steps_df.runDistance['20160427': '20230114'])
```

时间序列 ACF 是时间序列分析中的一种重要工具,用于研究时间序列数据中的自相关性。自相关性是指在同一时间序列中,不同时刻的数据之间的相关性。ACF 是一种描述时间序列数据在不同时间延迟下的相关性的方法。ACF 可以通过计算时间序列数据与其自身在不同时间延迟下的相关性系数来计算。具体来说,ACF 是时间序列数据与其自身在不

```
date
2016-04-27      46
2016-04-28      79
2016-04-29      29
2016-04-30      11
2016-05-01       0
               ...
2023-01-10    2495
2023-01-11    7066
2023-01-12    6506
2023-01-13    2578
2023-01-14    1176
Name: runDistance, Length: 2454, dtype: int64
date
2016-04-27      46
2016-04-28      79
2016-04-29      29
2016-04-30      11
2016-05-01       0
               ...
2023-01-10    2495
2023-01-11    7066
2023-01-12    6506
2023-01-13    2578
2023-01-14    1176
Freq: D, Name: runDistance, Length: 2454, dtype: int64
```

图 5-25　缺失数据填充结果

同时间间隔内的相关系数的函数。在计算 ACF 时，一个时间序列数据集被拆分成不同的时间段，并计算在不同时间延迟下的相关性系数。

PACF 是一种描述时间序列的自相关性的统计工具。它用来衡量在给定两个时间点之间的距离上，一个时间序列在去除其前面的时间点对它的影响后，对后面的时间点的影响。故 PACF 可以帮助确定一个时间序列需要多少个滞后值才能很好地对其进行预测。PACF 通常用于进行时间序列建模、诊断和选择自回归模型阶数。

画出 ACF、PACF 图像，代码如下，结果如图 5-26 所示。

```
#画出 ACF、PACF 图像
def plotds(xt, nlag = 30, fig_size = (12, 10)):
    if not isinstance(xt, pd.Series):
        xt = pd.Series(xt)
    plt.figure(figsize = fig_size)
    layout = (2, 2)
     plt.rcParams.update({'font.size': 20})
    ax_xt = plt.subplot2grid(layout, (0, 0), colspan = 2)
    ax_acf = plt.subplot2grid(layout, (1, 0))
    ax_pacf = plt.subplot2grid(layout, (1, 1))
    xt.plot(ax = ax_xt)
    ax_xt.set_title('Time Series')
    plot_acf(xt, lags = nlag, ax = ax_acf)
    plot_pacf(xt, lags = nlag, ax = ax_pacf)
    plt.tight_layout()
    return None
#画出 ACF、PACF 图像
plotds(steps_df['runDistance'].dropna(), nlag = 50)
```

进行时间序列分析前，必须判断其是否平稳，需要使用 ADF 方法进行平稳性检验，代码如下：

```
#平稳性检验
diff = 0
```

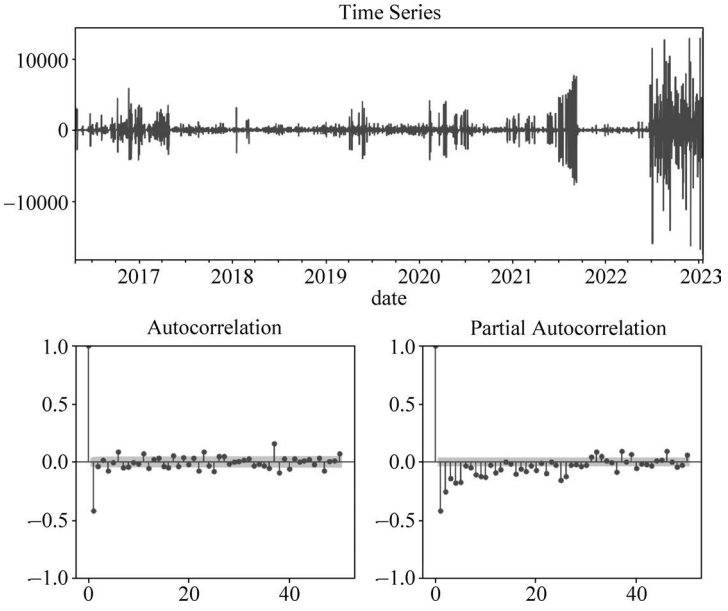

图 5-26 ACF、PACF 图像

```
adf = ADF(steps_df['runDistance'])
if adf[1] > 0.05:
    print(u'原始序列经检验不平稳,p 值为:%s'%(adf[1]))
else:
    print(u'原始序列经检验平稳,p 值为:%s'%(adf[1]))
```

实验结果发现,原始序列经检验不平稳,p 值为 0.8338039027785843。

如果序列值彼此之间没有任何相关性,那就意味着该序列是一个没有记忆的序列,过去的行为对将来的发展没有丝毫影响,这种序列称之为纯随机序列。从统计分析的角度而言,纯随机序列是没有任何分析价值的序列,也称白噪声检验。白噪声序列检验代码如下:

```
from statsmodels.stats.diagnostic import acorr_ljungbox
lb = acorr_ljungbox(steps_df['runDistance'],lags = 1)    #返回的是 DataFrame 不能用嵌套列表
p = lb.values[0,1]                                       #获取第 0 行第 1 列数据
if p < 0.05:
    print (u'原始序列为非白噪声序列,p 值为:%s'%p)
else:
    print (u'原始序列为白噪声序列,p 值为:%s'%p)
```

实验结果发现,原始序列为非白噪声序列,p 值为 3.084362836456756e-193。

接下来对不平稳的数据进行平稳化处理,代码如下,平稳化后的数据如图 5-27 所示。

```
# 3. 处理数据,平稳化数据
# 这里只是简单地作了一阶差分,还有其他平稳化时间序列的方法
# 可以查询资料后改进这里的平稳化效果
steps_diff = steps_df.diff(1)
steps_diff = steps_diff.dropna()
print(steps_diff.head())
print(steps_diff.dtypes)
plotds(steps_diff['runDistance'].dropna(),nlag = 50)
```

```
              steps  distance  runDistance  calories
date
2016-04-28  11625.0    8818.0         33.0     470.0
2016-04-29   1429.0     856.0        -50.0     -14.0
2016-04-30 -13876.0   -9935.0        -18.0    -530.0
2016-05-01   -257.0    -330.0        -11.0      37.0
2016-05-02   6135.0    4679.0          0.0     182.0
steps          float64
distance       float64
runDistance    float64
calories       float64
dtype: object
```

图 5-27　平稳化后数据显示

对平稳化后的数据进行平稳性检验。代码如下：

```
#平稳性检验
diff = 0
adf = ADF(steps_diff['runDistance'])
if adf[1] > 0.05:
  print(u'一阶差分序列经检验不平稳,p 值为:%s'%(adf[1]))
else:
  print(u'一阶差分序列经检验平稳,p 值为:%s'%(adf[1]))
```

实验结果发现，一阶差分序列经检验平稳，p 值为 0.0。

白噪声检验，代码如下：

```
from statsmodels.stats.diagnostic import acorr_ljungbox
lb = acorr_ljungbox(steps_diff['runDistance'],lags = 1)   #返回的是 DataFrame 不能用嵌套列表
p = lb.values[0,1] #获取第 0 行第 1 列数据
if p < 0.05:
  print (u'一阶差分序列为非白噪声序列,p 值为:%s'%p)
else:
  print (u'一阶差分序列为白噪声序列,p 值为:%s'%p)
```

实验结果发现，一阶差分序列为非白噪声序列，p 值为 7.93144056649956e-95。

根据 ACF 和 PACF 定阶并建立模型，代码如下，结果如图 5-28 所示。

```
# 4. 根据 ACF 和 PACF 定阶并建立模型
model = ARIMA(steps_df['runDistance'], order = (2, 1, 2))
model_fit = model.fit()
print(model_fit.summary())
```

模型拟合代码如下。结果如图 5-29 所示。

```
#模型拟合
steps_df['ARIMA'] = model_fit.predict(typ = 'levels')
f, axarr = plt.subplots(1, sharex = True)
f.set_size_inches(12, 8)
steps_df['runDistance'].plot(color = 'b', linestyle = '-', ax = axarr)
steps_df['ARIMA'].plot(color = 'r', linestyle = '--', ax = axarr)
axarr.set_title('ARIMA(2, 1, 2)')
plt.rcParams.update({'font.size': 20})
plt.xlabel('Index')
plt.ylabel('runDistance')
plt.show()
```

SARIMAX Results

Dep. Variable:	runDistance	No. Observations:	2454
Model:	ARIMA(2, 1, 2)	Log Likelihood	-21128.937
Date:	Sun, 23 Jul 2023	AIC	42267.873
Time:	15:15:00	BIC	42296.899
Sample:	04-27-2016 - 01-14-2023	HQIC	42278.421
Covariance Type:	opg		

	coef	std err	z	P>|z|	[0.025	0.975]
ar.L1	0.4373	0.106	4.138	0.000	0.230	0.644
ar.L2	0.0324	0.031	1.039	0.299	-0.029	0.093
ma.L1	-1.1464	0.105	-10.877	0.000	-1.353	-0.940
ma.L2	0.1837	0.100	1.842	0.066	-0.012	0.379
sigma2	1.917e+06	1.56e+04	122.559	0.000	1.89e+06	1.95e+06

Ljung-Box (L1) (Q):	0.00	Jarque-Bera (JB):	103919.47
Prob(Q):	0.97	Prob(JB):	0.00
Heteroskedasticity (H):	11.41	Skew:	3.54
Prob(H) (two-sided):	0.00	Kurtosis:	34.09

图 5-28　模型拟合结果显示

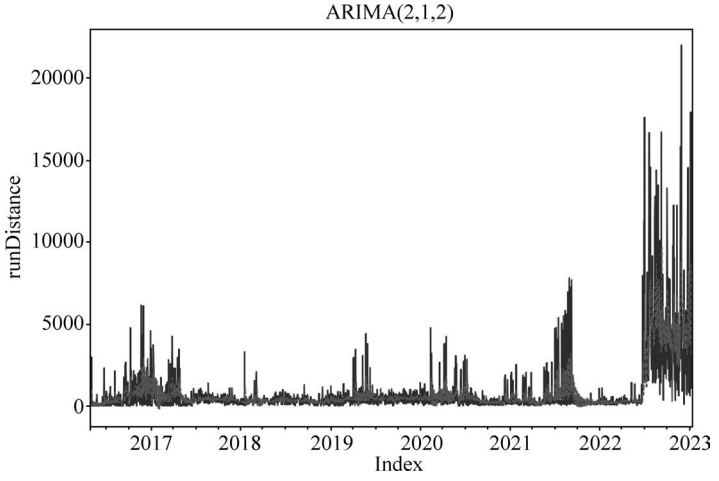

图 5-29　模型拟合结果可视化

模型预测并可视化预测结果，代码如下，预测结果可视化如图 5-30 所示。

```
#预测
output = model_fit.predict('20180201','20180207',dynamic = True,typ = 'levels')
#使用模型预测数据
#print(output)
# 6.可视化预测结果
steps_forcast = pd.concat([steps_df.runDistance['20180115':'20180207'],output],axis = 1,
keys = ['original','predicted'])
plt.figure()
plt.plot(steps_forcast)
plt.title('Original vs predicted')
plt.savefig('./steps_pred.png',format = 'png')
plt.show()
```

模型评估代码如下：

```
#模型评估,平均绝对误差 MAE、均方误差 MSE、均方根误差 RMSE
#对短期预测结果进行评估
short_label = steps_df.runDistance['20200201': '20200202']
```

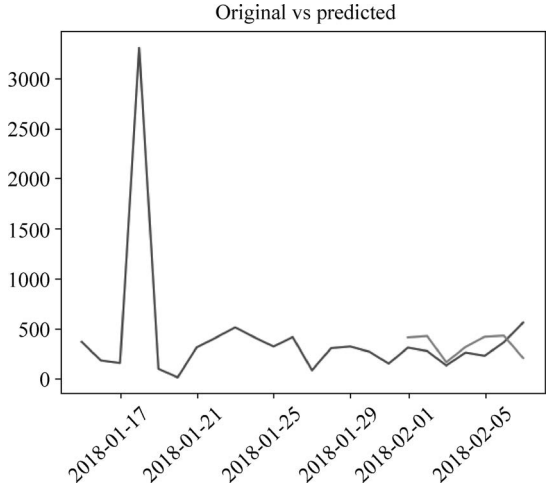

图 5-30　预测可视化结果

```
short_prediction = output[:2]
short_mse_score = mse(short_label,short_prediction)
short_rmse_score = math.sqrt(mse(short_label,short_prediction))
short_mae_score = mae(short_label,short_prediction)
print('short MSE: %.4f,short_RMSE: %.4f,short_MAE: %.4f' % (short_mse_score,short_
    rmse_score,short_mae_score))
```

短期预测评估结果为：short MSE：133442.5341，short RMSE：365.2979，short MAE：353.5008。

```
#对长期预测结果进行评估
all_label = steps_df.runDistance['20200201': '20200207']
all_prediction = output
long_mse_score = mse(all_label,all_prediction)
long_rmse_score = math.sqrt(mse(all_label, all_prediction))
long_mae_score = mae(all_label, all_prediction)
print('long_MSE: %.4f,long_RMSE: %.4f,long_MAE: %.4f' % (long_mse_score, long_rmse_
score, long_mae_score))
```

长期预测评估结果为：long_MSE：94948.9715，long_RMSE：308.1379，long_MAE：250.0324。

由短期预测结果和长期预测结果的对比显示，长期预测结果更好。

对后续数据进行预测，代码如下，预测结果如图 5-31 所示。

```
#对后续数据进行预测
from statsmodels.graphics.tsaplots import plot_predict
from statsmodels.tsa.arima.model import ARIMA
dta = steps_df.runDistance['2020/12/15': '2020/12/30'] #选择可视化的原始数据区间
fig, ax = plt.subplots()
ax = dta.loc['2020/12/15':].plot(ax=ax) #设置坐标轴起点
plot_predict(model.fit(),'2020/12/25','2021/01/30',ax=ax) #可视化预测时间起点与终点
plt.show()
```

5.3.2　案例：LSTM 模型分析数据

1. LSTM 模型原理

在深度学习领域中，尤其是循环神经网络(recurrent neural network，RNN)，“长期依

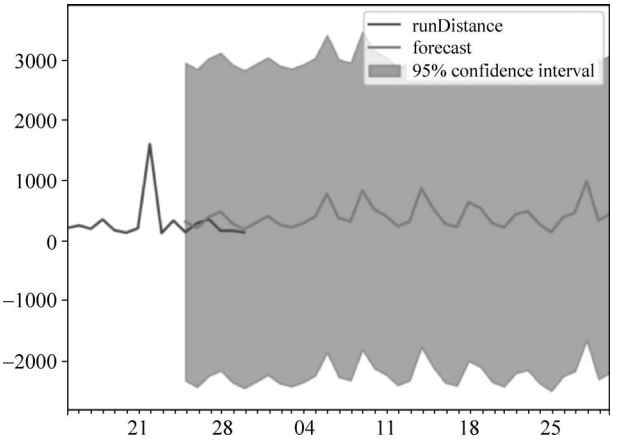

图 5-31　后续数据预测结果

赖”问题是普遍存在的。长期依赖产生的原因是当神经网络的节点经过许多阶段的计算后，之前比较长的时间片的特征已经被覆盖。长短期记忆(long short-term memory，LSTM)网络是 RNN 的一种变体，可以很有效地解决简单 RNN 的梯度爆炸或消失问题。而 LSTM 之所以能够解决 RNN 的长期依赖问题，是因为 LSTM 引入了门(gate)机制用于控制特征的流通和损失。LSTM 是由一系列 LSTM 单元(LSTM unit)组成，其链式结构如图 5-32 所示。

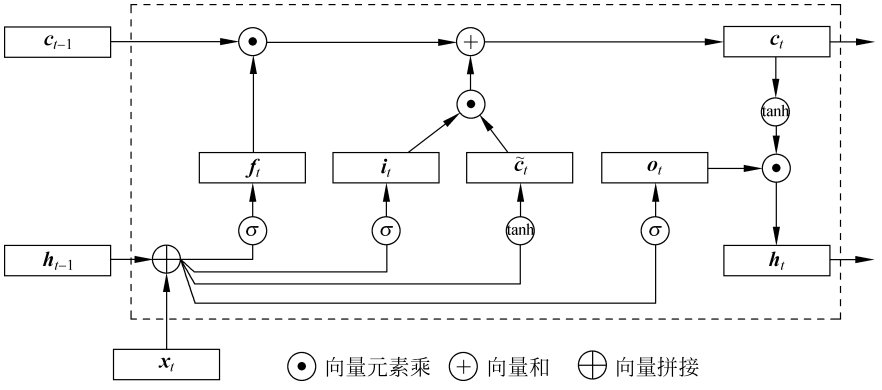

图 5-32　LSTM 链式结构

LSTM 区别于 RNN 的地方，主要就在于它在算法中加入了一个判断信息有用与否的“处理器”，这个处理器作用的结构被称为 cell。

一个 cell 当中放置了三扇门，分别叫作输入门、遗忘门和输出门。一个信息进入 LSTM 的网络当中，可以根据规则来判断是否有用。只有符合算法认证的信息才会留下，不符合的信息则通过遗忘门被遗忘。具体来讲，三个门的基本功能如下所述：

(1) 遗忘门的功能是决定应丢弃或保留哪些信息。来自前一个隐藏状态的信息和当前输入的信息同时传递到 sigmoid 函数中，输出值介于 0～1，越接近 0 意味着越应该丢弃，越接近 1 意味着越应该保留；

(2) 输入门用于更新 cell 状态。首先将前一层隐藏状态的信息和当前输入的信息传递到 sigmoid 函数中。将值调整到 0～1 来决定要更新哪些信息。0 表示不重要，1 表示重要。

其次还要将前一层隐藏状态的信息和当前输入的信息传递到 tanh 函数中，创造一个新的候选值向量。最后将 sigmoid 函数的输出值与 tanh 函数的输出值相乘，sigmoid 函数的输出值将决定 tanh 函数的输出值中哪些信息是重要且需要保留下来的；

（3）输出门用来确定下一个隐藏状态的值，隐藏状态包含了先前输入的信息。首先，将前一个隐藏状态和当前输入传递到 sigmoid 函数中，然后将新得到的 cell 状态传递给 tanh 函数。最后将 tanh 函数的输出与 sigmoid 函数的输出相乘，以确定隐藏状态应携带的信息。再将隐藏状态作为当前 cell 的输出，把新的 cell 状态和新的隐藏状态传递到下一个时间步长中。

2. 分析实验过程

LSTM 模型预测的基本程序包括构建网络模型、对模型进行训练、利用训练好的模型进行预测、对预测结果进行评估。利用 LSTM 处理数据的流程如下。

导入相应的包，代码如下：

```
import numpy as np
import pandas as pd
import torch.nn as nn
import torch
import matplotlib.pyplot as plt
import matplotlib.ticker as mticker
import statsmodels.api as sm
import statsmodels.tsa.api as tsa
import math
import torch.utils.data as Data
```

绘制序列图像设置，代码如下：

```
#绘制序列
def plot_series(time,series,format = " - ",start = 0,end = None,label = None):
    #根据时间轴和对应数据列表绘制序列图像
    plt.plot(time[start:end], series[start:end],format,label = label)
    #设置横纵轴意义
    plt.xlabel("Time")
    plt.ylabel("Value")
    #设置图例说明字体大小
    if label:
      plt.legend(fontsize = 14)
plt.rcParams.update({'font.size': 20})
    #显示网格
    plt.grid(True)
```

读取数据，代码如下：

```
data = pd.read_csv('Steps.csv',index_col = 0,na_values = ' + 9999,9') #读取文件，将日期设为索引
data = data['runDistance'] #选取 runDistance 一列

data.index = pd.to_datetime(data.index) #将 data.index 设置为时间格式
start_time = pd.to_datetime('2019 - 01 - 01 00:00:00')
end_time = pd.to_datetime('2019 - 06 - 30 23:00:00')
data = data[start_time:end_time]
data = data.dropna() #剔除 NaN 的数据
#  data = data.str.split(",",expand = True)[0]
#  data = data.astype("int")/10 #将 TMP 数据从 str 转换为 int，并获取正确数值

```

```
index = pd.date_range(start = start_time,end = end_time,freq = "H")
data = data.reindex(index)#将时间补全(前面有丢弃操作),并将间隔时间设置为 1h,重新设置 data 的索引
#进行插值(部分补全的时间没有对应的数据)
data = data.interpolate()
#将数据转换为 array 类型,
series = np.array(data)
```

数据可视化,代码如下,可视化结果如图 5-33 所示。

```
#数据可视化
fig,ax = plt.subplots(figsize = (20,6))
#设置纵轴单位
ax.yaxis.set_major_formatter(mticker.FormatStrFormatter('%d℃'))
data.plot()
plt.xlabel('Date')
plt.ylabel('runDistance')
plt.grid(True)
plt.show()
```

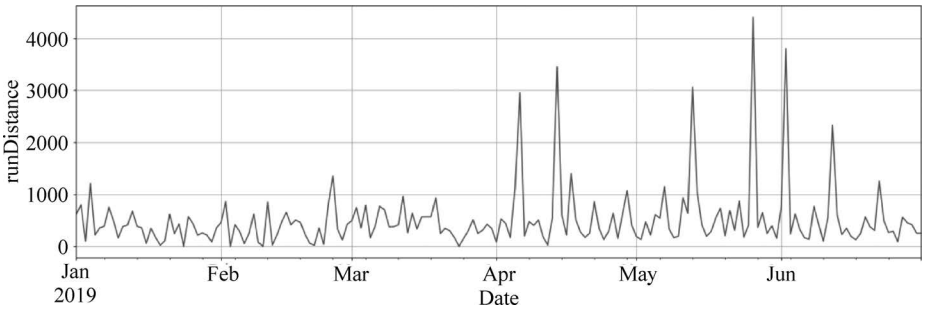

图 5-33 数据可视化结果显示

设置超参数,代码如下:

```
#设置超参数
input_size = 1
hidden_size = 128
output_size = 1
epochs = 100
lr = 0.05
batch_size = 20
time_step = 12
```

划分训练集和测试集,代码如下:

```
#前 140 天的数据作为训练集
train_data = data[0:140 * 24]
#剩下的时间数据作为测试集
test_data = data[140 * 24:]
#数据归一化
train_data_normalized = (train_data - train_data.min())/(train_data.max() - train_data.min())
test_data_normalized = (test_data - train_data.min())/(train_data.max() - train_data.min())
```

滑动窗口采样,代码如下:

```
train_x = []
train_y = []
test_x = []
test_y = []
```

```
#对训练数据采样
i = 0
while(i + time_step + output_size < len(train_data_normalized)):
    #输入的序列
    train_x.append(train_data_normalized[i:i + time_step])
    #输出的序列
    train_y.append(train_data_normalized[i + time_step:i + time_step + output_size])
    i += output_size
#对测试数据采样
j = 0
while(j + time_step + output_size < len(test_data_normalized)):
    test_x.append(test_data_normalized[j:j + time_step])
    test_y.append(test_data_normalized[j + time_step:j + time_step + output_size])
    j += output_size
```

装入数据，代码如下：

```
#将数据转换为 tensor 模式
train_x = torch.tensor(train_x,dtype = torch.float32)
train_y = torch.tensor(train_y,dtype = torch.float32)
test_x = torch.tensor(test_x,dtype = torch.float32)
test_y = torch.tensor(test_y,dtype = torch.float32)
#将训练数据装入 dataloader
train_dataset = Data.TensorDataset(train_x,train_y)
train_loader = Data.DataLoader(dataset = train_dataset,
                    batch_size = batch_size,
                    shuffle = True,num_workers = 0)
```

构建 LSTM 网络，代码如下：

```
class MYLSTM(nn.Module):
    def __init__(self,input_size,hidden_size,output_size,time_step):
        super(MYLSTM,self).__init__()
        self.input_size = input_size
        self.hidden_size = hidden_size
        self.output_size = output_size
        self.time_step = time_step
        #创建 LSTM 层和 linear 层，LSTM 层提取特征，linear 层用作最后的预测
        self.rnn = nn.LSTM(
            input_size = self.input_size,
            hidden_size = self.hidden_size,
            num_layers = 1,
            batch_first = True,
            bidirectional = True
        )
        self.out = nn.Linear(self.hidden_size * 2,self.output_size)
    def forward(self,x):
        #获得 LSTM 的计算结果，舍去 h_n
        r_out,_ = self.rnn(x)
        #按照 LSTM 模型结构修改 input_seq 的形状，作为 linear 层的输入
        r_out = r_out.reshape(-1,self.hidden_size * 2)
        out = self.out(r_out)
        #将 out 恢复成(batch,seq_len,output_size)
        out = out.reshape(-1,self.time_step,self.output_size)
        #return 所有 batch 的 seq_len 的最后一项
        return out[:,-1,:]
```

模型参数初始化，代码如下：

```
#实例化神经网络
```

```
net = MYLSTM(input_size,hidden_size,output_size,time_step)
#初始化网络参数
for param in net.parameters():
    nn.init.normal_(param,mean = 0,std = 0.01)
#设置损失函数
loss = nn.MSELoss()
#设置优化器
optimizer = torch.optim.SGD(net.parameters(),lr = lr)
#如果 GPU 可用,就用 GPU 计算,否则使用 CPU 运算
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
#将 net 复制到 device(GPU 或 CPU)
net.to(device)
```

开始训练,代码如下,训练结果如图 5-34 所示。

```
#开始训练
train_loss = []
test_loss = []
for epoch in range(epochs):
    train_1 = []
    test_1 = 0
    for x,y in train_loader:
        #RNN 输入应为 input(seq_len,batch,input_size),将 x 转换为三维数据
        x = torch.unsqueeze(x,dim = 2)
        #将 x 和 y 放入 device 中
        x = x.to(device)
        y = y.to(device)
        #计算得到预测值
        y_predict = net(x)
        #计算预测值与真实 y 的 loss
        l = loss(y_predict,y)
        #清空所有被优化过的 variable 的梯度
        optimizer.zero_grad()
        #反向传播,计算当前梯度
        l.backward()
        #根据梯度更新网络参数
        optimizer.step()
        train_1.append(l.item())
    #修改测试集的维度以便放入网络中
    test_x_temp = torch.unsqueeze(test_x,dim = 2)
    #测试集放入 device 中
    test_x_temp = test_x_temp.to(device)
    test_y_temp = test_y.to(device)
    #得到测试集的预测结果
    test_predict = net(test_x_temp)
    #计算测试集 loss
    test_1 = loss(test_predict,test_y_temp)
    print('Epoch %d:train loss = %.5f,test loss = %.5f' %(epoch + 1,np.array(train_
1).mean(),test_1.item()))
    train_loss.append(np.array(train_1).mean())
    test_loss.append(test_1.item())
```

```
Epoch 1:train loss= 0.01491,test loss= 0.04446
Epoch 2:train loss= 0.01426,test loss= 0.04423
Epoch 3:train loss= 0.01418,test loss= 0.04428
...
Epoch 98:train loss= 0.00030,test loss= 0.00112
Epoch 99:train loss= 0.00030,test loss= 0.00111
Epoch 100:train loss= 0.00030,test loss= 0.00111
```

图 5-34 模型训练结果

绘制 loss 曲线，代码如下，loss 曲线如图 5-35 所示。

```
#绘制 loss 曲线
plt.plot(range(epochs),train_loss,label = 'train_loss',linewidth = 2)
plt.plot(range(epochs),test_loss,label = 'test_loss',linewidth = 2)
plt.xlabel('epoch')
plt.ylabel('loss')
plt.legend()
plt.show
```

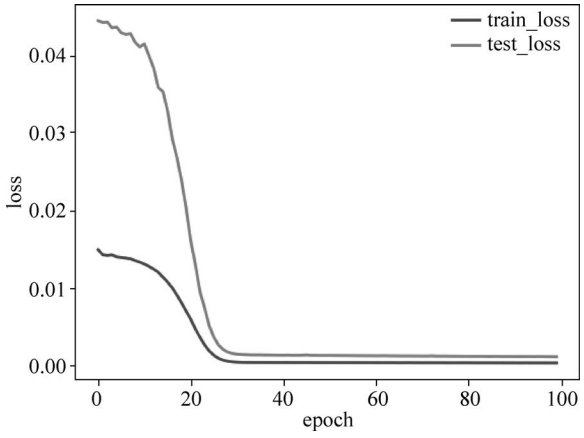

图 5-35　loss 曲线展示

预测并对比结果，代码如下，预测结果局部对比图如图 5-36 所示，整体对比图如图 5-37 所示。

```
#将测试集放入模型计算预测结果
test_x_temp = torch.unsqueeze(test_x,dim = 2)
test_x_temp = test_x_temp.to(device)
predict = net(test_x_temp)
#逆归一化
predict = predict.cpu().detach().numpy() * (train_data.max() - train_data.min()) +
train_data.min()
test_y = np.array(test_y) * (train_data.max() - train_data.min()) + train_data.min()
predict_result = []
test_y_result = []
for item in predict:
  predict_result += list(item)
for item in test_y:
  test_y_result += list(item)
#指定 figure 的宽和高
fig_size = plt.rcParams['figure.figsize']
fig_size[0] = 10
fig_size[1] = 6
plt.rcParams['figure.figsize'] = fig_size
#画出实际和预测的对比图
plt.plot(range(len(test_y_result)),test_y_result,label = 'True')
plt.plot(range(len(predict_result)),predict_result,label = 'Prediction')
plt.xlabel("TIME")
plt.ylabel("Value")
plt.grid(True)
plt.legend()
plt.show()
#与整体数据进行比较
```

```
plt.plot(range(len(series)),series,label = 'True')
plt.plot(range(len(series) - len(predict_result),len(series)),predict_result,label = '
Prediction')
plt.xlabel("TIME")
plt.ylabel("Value")
plt.grid(True)
plt.legend()
plt.show()
```

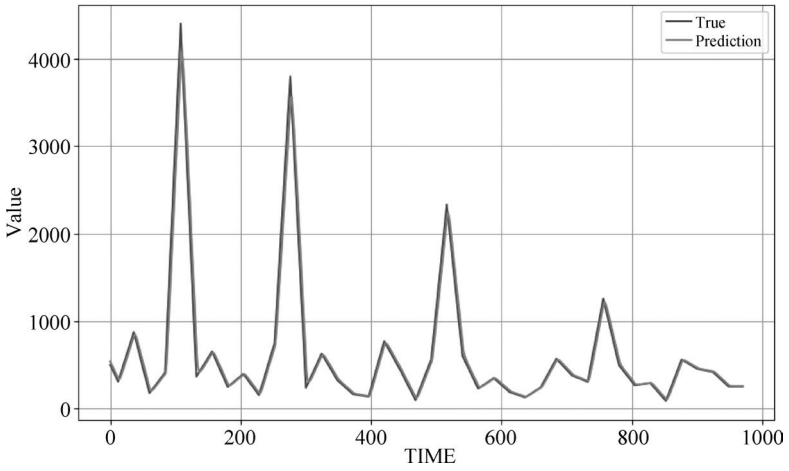

图 5-36　预测结果局部对比图

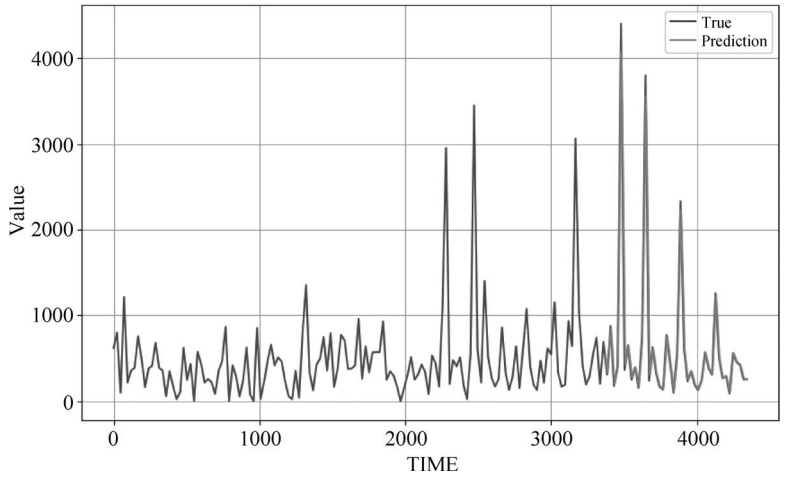

图 5-37　预测结果整体对比图

预测结果评估，代码如下：

```
from sklearn.metrics import mean_squared_error as mse, mean_absolute_error as mae
mae_nn = mae(test_y_result,predict_result)
mse_nn = mse(test_y_result,predict_result)
print(MAE:',MAE_nn)
print(MSE:',MSE_nn)
```

评估结果为 MAE：64.240135；MSE：13164.344。

LSTM 模型和 ARIMA 模型的结果对比显示，LSTM 模型的 MSE 和 MAE 值更小，说明模型的预测效果更好。

参考文献

[1] Goodfellow I, Bengio Y, Courville A. Deep learning[M]. MIT Press, 2016.
[2] Zhang A. Lipto Z C. Li M, et al. Dive into deep learning[M]. Cambridge University Press, 2023.
[3] 邱锡鹏. 神经网络与深度学习[M]. 北京：机械工业出版社, 2020.
[4] Palma W. Time series analysis[M]. John Wiley & Sons, 2016.
[5] Tsay R S, Chen R. Nonlinear time series analysis[M]. John Wiley & Sons, 2018.
[6] Chatfield C, Xing H. The analysis of time series: An introduction with R[M]. CRC Press, 2019.
[7] Gulli A, Pal S. Deep learning with Keras[M]. Packt Publishing Ltd., 2017.
[8] Patterson J, Gibson A. Deep learning: A practitioner's approach[M]. O'Reilly Media, Inc., 2017.
[9] Montgomery D C, Jennings C L, Kulahci M. Introduction to time series analysis and forecasting[M]. John Wiley & Sons, 2015.