**数 据 分 析**

**考 查 作 品**

**学 号 220901108**

**姓 名 侯栩涛**

**系 别 计算机与信息工程学院**

**年 级 2022级计算机科学与技术3班**

**专 业 计算机科学与技术**

项目基本信息

|  |  |
| --- | --- |
| 项目名称 |  |
| **项目简介** | **1 项目描述：**  本项目利用Python的网络爬虫、数据分析及可视化能力，聚焦安阳市2024年天气数据。旨在精准获取数据后深入剖析当地天气状况，通过可视化呈现每月天气变化与全年天气占比，为多领域提供参考助力决策规划。执行时先以Python爬虫库从专业气象平台获取每日天气详情，再用Pandas库清洗、整理数据，最后借助Tkinter库把数据转化为柱状图、扇形图等，直观展现全年天气全貌，期望以此发挥数据价值，服务各方需求。  **2实现的功能及结果：**  **2.1 主要功能：**  （1）数据采集功能：编写 Python 爬虫程序，通过设定特定请求头模拟浏览器访问，精准锁定安阳市 2024 年天气数据页面，抓取每日最高温、最低温、天气现象等信息，最后存储为 CSV格式，便于后续操作。  （2）数据清洗功能：利用 Pandas 库全方位梳理采集来的数据，精准揪出重复记录予以剔除。  （3）运用Tkinter 库，绘制安阳市 2024 年每月天气情况柱形图，直观展现天气月度统计；绘制全年天气类型占比饼图，精准量化各类天气出现频率，生动呈现全年天气格局。  **2.2 结果：**  （1）每月天气变化可视化：产出系列柱形图清晰呈现安阳市 2024 年每月天气情况变化；  （2）全年天气占比可视化：生成的扇形图精准量化晴、多云、小雨、中雨、大雪等各类天气在 2024 年所占比重，直观显示晴天、多云天气为当地主导天气类型，降水及降雪天气占比较小但分布不均；借助可视化图表，用户能迅速掌握安阳市全年天气构成。  **3 优缺点：**   * 1. **优点：**   （1）数据精准、具时效性，直采专业气象源，如实反映 2024 年安阳天气。  （2）高度定制化，按需灵活调整爬虫、可视化内容；  （3）自动化高效，编写后自动流程作业；  （4）可视化直观，各类人群都能轻松洞悉天气规律。   * 1. **缺点：**   （1）依赖数据源稳定性，页面改版等易使爬虫代码失效，需维护更新；  （2）数据质量受局限，存在异常及维度不足问题；（3）技术门槛高，对开发者专业及耐心要求高，入门调试较难。 |

**一、系统描述**

本系统依托 Python 技术构建，专为深度剖析安阳市 2024 年天气状况而设。核心模块分工明确，爬虫模块运用 requests、BeautifulSoup 等库，向权威气象源精准发请求，抓取气温、降水、天气现象等详尽信息；数据处理模块借助 Pandas，高效清洗、结构化原始数据，修复异常、填补缺失值；可视化模块采用 Tkinter，将处理好的数据绘制成直观柱状图、扇形图，清晰呈现天气变化趋势、占比，为气象、民生等领域输出精准参考。

**二、系统设计**

1、系统架构

数据采集层处于最前端，利用 Python 的 requests 库模拟 HTTP 请求，搭配 BeautifulSoup 精准解析专业气象网站页面结构，定位安阳市 2024 年天气数据元素，确保稳定、高效抓取每日天气详情，为后续流程输送原始素材。

2、存储设计

原始数据与清洗后的数据分别存储，原始数据用 CSV 格式存档，保留最原始信息，便于回溯排查；清洗后的数据存入结构化更强的 SQLite 数据库，方便快速查询、聚合分析，满足系统不同阶段数据使用需求。

3、数据可视化层

选用 Tkinter 库绘制图表，为适配多场景需求，打造交互式可视化界面。针对每月天气变化，绘制动态柱状图，鼠标悬停实时显示具体数值；设计扇形图展示全年天气占比，用不同颜色鲜明区分天气类型，用户点击扇区可弹窗展示详情，实现直观、便捷的数据呈现，助力精准决策。

**三、系统实现**

一、数据采集层实现

1. 环境搭建与库导入：在 Python 开发环境中，确保已安装 requests 和 BeautifulSoup 库。创建独立的采集脚本文件，于文件头部导入所需库。

2. 请求发送与页面获取：精心定制请求头，模拟常见浏览器访问，规避反爬限制。使用 requests.get() 方法向目标专业气象网站发送获取安阳市 2024 年天气页面的请求，若响应状态码为 200，代表请求成功，获取网页源码，存入变量供后续解析。

3. 数据解析与提取：调用 BeautifulSoup(html, 'html.parser')`将网页源码转化为可解析对象，依据网站 HTML 结构特点，运用 XPath 精准定位气温、降水、天气状况等元素。

二、存储设计实现

1. 原始数据存储：在数据采集脚本成功抓取数据后，借助 Python 的内置 csv 模块，设定字段名（如日期、最高温、最低温等），利用 csv.writer()实例化写入对象，逐行将原始数据写入 CSV 文件。

2. 清洗后数据存储：数据处理层整理完数据后，连接 SQLite 数据库，利用 Python 的 sqlite3 库，执行 SQL 创建表语句定义合适的数据表结构，随后通过 INSERT INTO语句将清洗后的数据逐行插入表中，方便后续查询分析。

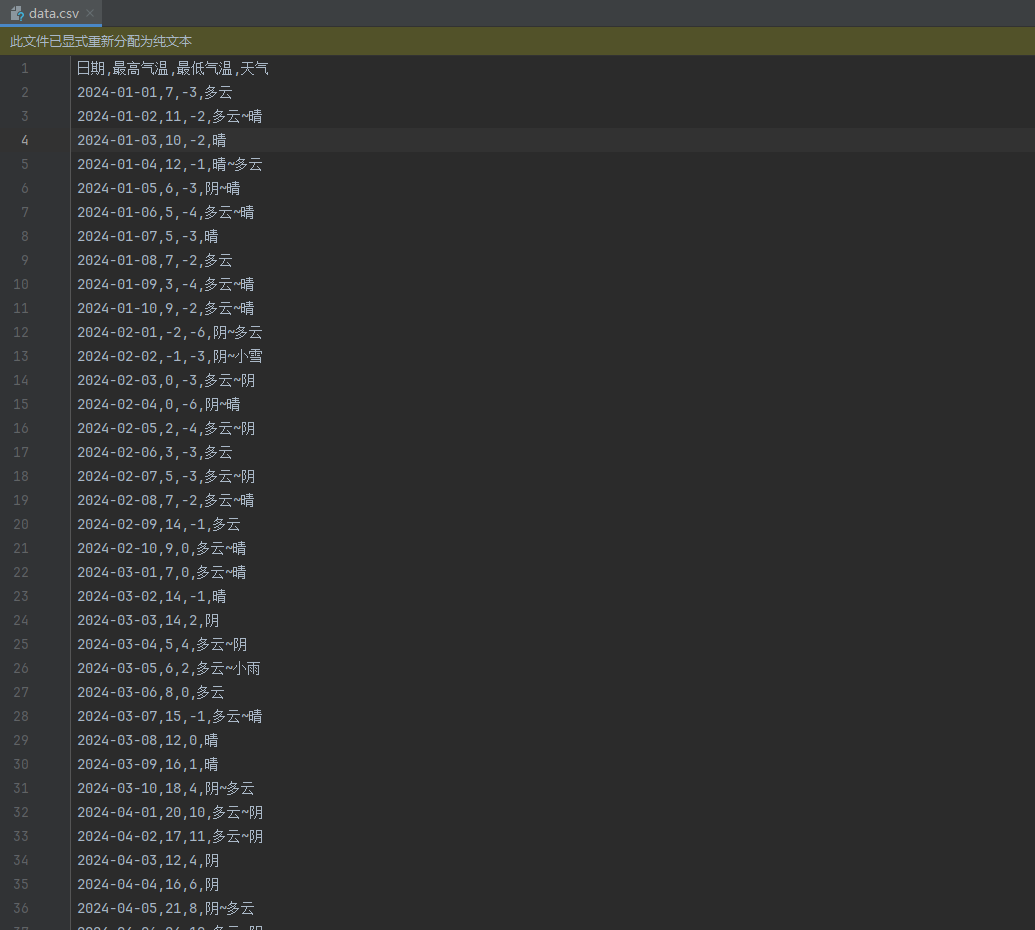
三、数据可视化层实现

1. Tkinter 环境准备与窗口搭建：在可视化脚本文件开头导入 import tkinter as tk，创建 Tkinter 主窗口对象 root = tk.Tk()，并合理设置窗口标题、大小等属性，搭建可视化基础框架。

2. 图表绘制与交互功能添加：针对每月天气变化绘制柱状图，调用 matplotlib（常配合 Tkinter 使用）绘图函数，结合数据绘制柱子，利用 canvas.tag\_bind() 绑定鼠标悬停事件，在事件处理函数里显示具体数值；绘制扇形图展示全年天气占比时，为不同扇区设置独特颜色与点击事件，鼠标经过触发弹窗，借助 tk.messagebox展示详情，完整代码涵盖绘图逻辑、事件绑定、界面布局调整，最终呈现直观交互界面，达成精准数据呈现。

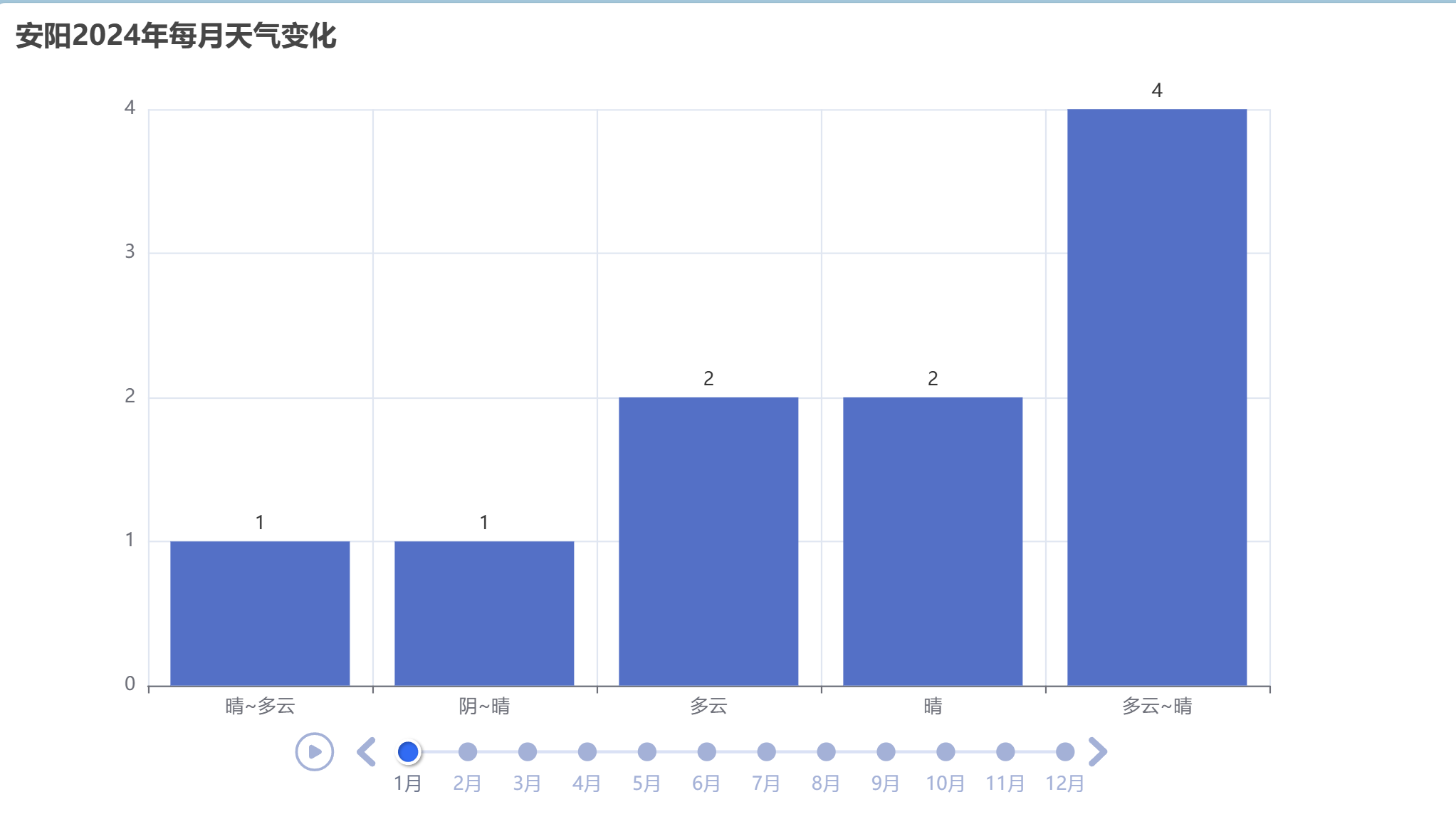
**四、系统演示**

（1）运行可执行文件weather.py,自动生成data.csv文件



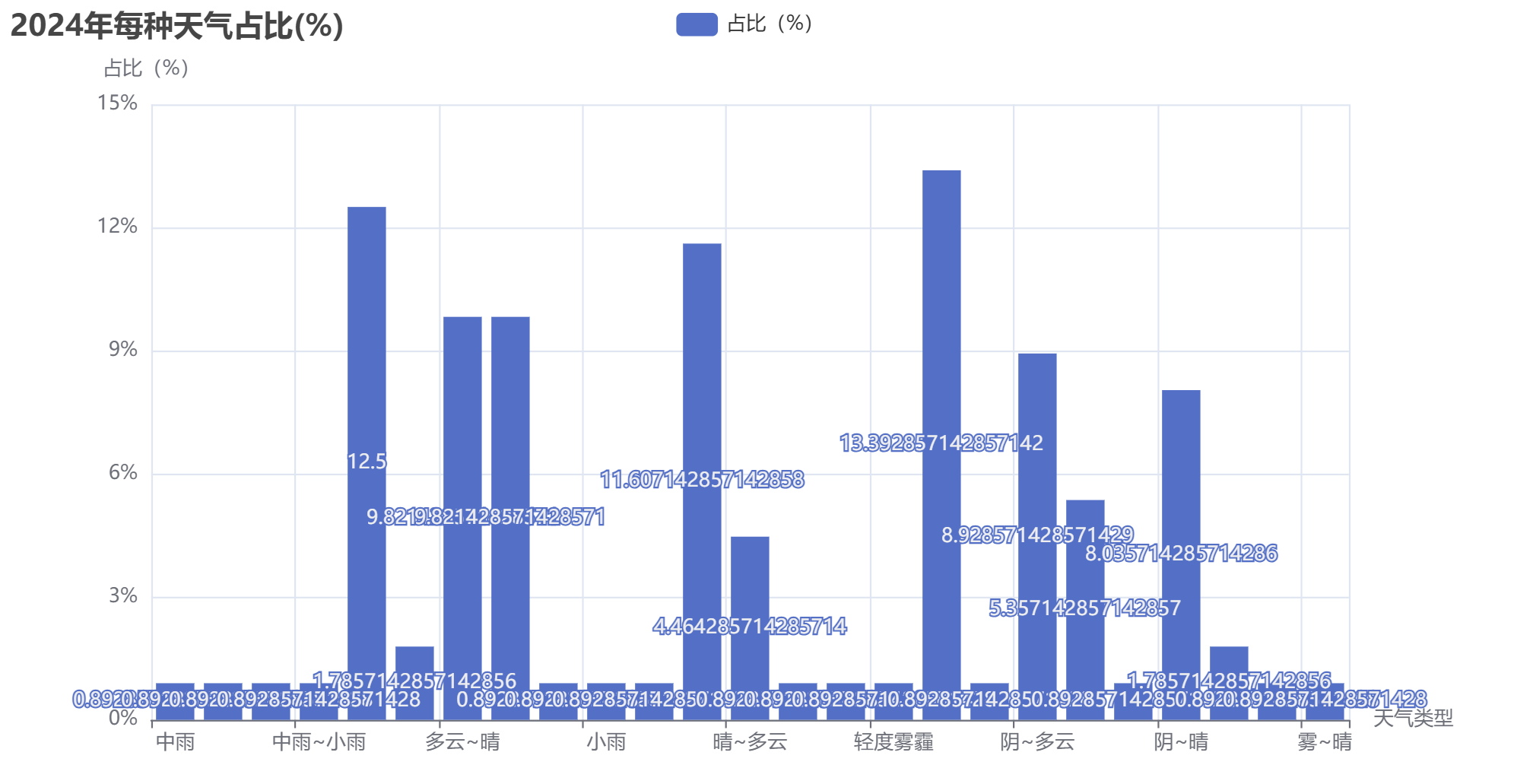
（2）运行可执行文件lunbo.py,生成可视化图表

安阳市2024年天气变化：

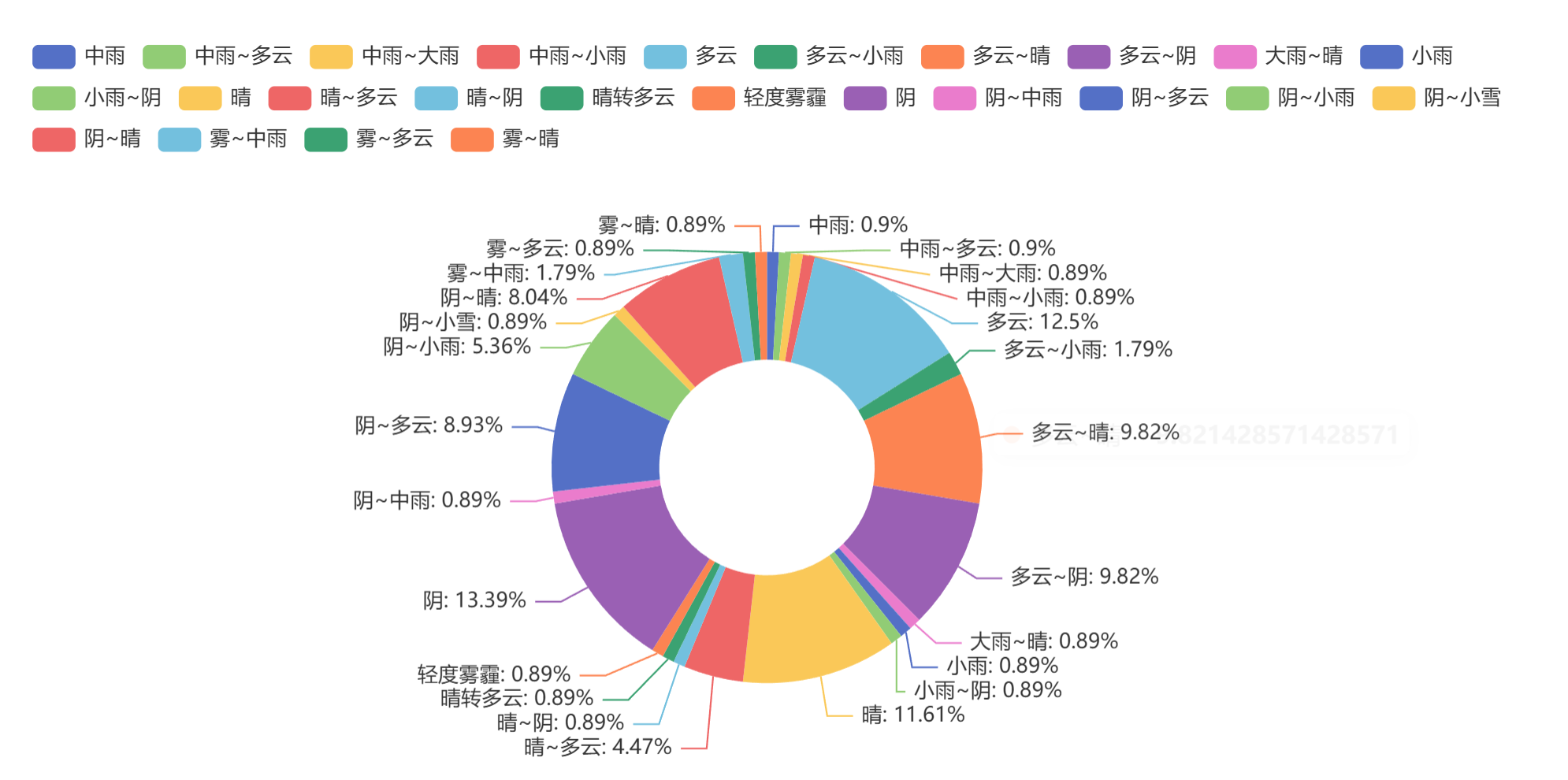


安阳市2024年各种天气占比：

柱形图：



扇形图：



**五、系统使用说明**

（一）运行系统前准备

确认网络连接正常，系统依靠网络爬取网站天气信息，网络中断将导致数据获取失败。

（二）启动系统

找到系统可执行文件，双击执行脚本，或在命令行输入相应的启动指令，

（三）爬取天气信息

系统启动后，等待片刻，系统依据内置的网页解析规则，定位网页中的天气关键信息，涵盖日期、气温、天气状况（晴、雨、多云等）等板块，精准提取数据，此过程视网站页面复杂程度及网络速度，耗时数秒至数分钟不等。

（四）生成 CSV 文件

成功爬取信息后，系统自动开启数据清洗与整理流程，剔除无效、重复数据，纠正格式错误。

将规整后的数据按行列形式，精准写入新建的 CSV 文件，默认文件名为 data.csv，存储于系统运行目录下；文件编码采用常见的 UTF-8，便于多平台兼容查阅，每行代表一天的完整天气数据，各列对应不同天气指标。

（五）生成可视化图表

CSV 文件生成完毕，系统随即读取文件内容，利用 pandas 库高效加载数据至内存数据结构，结合 matplotlib 或其他绘图库，按预设图表类型绘制图表。

**六、问题及解决方法**

（一）问题：系统向气象网站发送请求时，频繁遭遇 403 Forbidden 或 404 Not Found 错误，无法获取页面内容，这通常是由于网站的反爬机制识别出非自然浏览行为。

解决方法：仔细研究常见浏览器的请求头信息，完善系统中的请求头设置，添加诸如 User-Agent（模拟真实浏览器标识）、Referer（表明请求来源页面）等字段，骗过网站反爬检测；还可设置合理的请求间隔，避免短时间内高频请求触发封禁，

（二）问题：仅能抓取到部分天气信息，像缺失风力风向、湿度数据等，原因在于页面结构复杂，爬虫定位元素不够精准。

解决方法：精准定位元素，借助浏览器开发者工具，深入分析气象网站页面结构，定位存放完整天气数据的 HTML 标签、类名或 ID；对于动态加载数据，考虑使用 Selenium 库结合 ChromeDriver 等浏览器驱动，让程序模拟真实浏览器操作。

**七、源代码**

**Weather.py**

import requests # 进行网络请求  
from lxml import etree # 数据预处理  
import csv # 写入csv文件  
  
def getWeather(url):  
 weather\_info = []  
 # 请求头信息:浏览器版本型号,接收数据的编码格式  
 headers = {  
 'User-Agent': 'Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/535.1 (KHTML, like Gecko) Chrome/14.0.835.163 Safari/535.1'  
 }  
 # 发送请求  
 resp = requests.get(url, headers=headers)  
  
 # 解析html的数据  
 resp\_html = etree.HTML(resp.text)  
 # xpath提取所有数据  
 resp\_list = resp\_html.xpath("//ul[@class='thrui']/li")  
 # for循环迭代遍历  
 for li in resp\_list:  
 weather = {}  
 # 日期  
 weather['date\_time'] = li.xpath("./div[1]/text()")[0].split(' ')[0]  
 # 最高气温 (包含摄氏度符号)  
 high = li.xpath("./div[2]/text()")[0]  
 weather['high'] = high[:high.find('℃')]  
 # 最低气温  
 low = li.xpath("./div[3]/text()")[0]  
 weather['low'] = low[:low.find('℃')]  
 # 天气  
 weather['weather'] = li.xpath("./div[4]/text()")[0]  
 weather\_info.append(weather)  
 return weather\_info  
  
weathers = []  
  
# for循环生成有顺序的1-12  
for month in range(1, 13):  
 # 获取某一月的天气信息  
 # 三元表达式  
 weather\_time = '2024' + ('0' + str(month) if month < 10 else str(month))  
 print(weather\_time)  
 url = f'https://lishi.tianqi.com/anyang/{weather\_time}.html'  
 # 爬虫获取这个月的天气信息  
 weather = getWeather(url)  
 # 存到列表中  
 weathers.append(weather)  
print(weathers)  
  
# 数据写入(一次性写入)  
with open("data.csv", "w", newline='') as csvfile:  
 writer = csv.writer(csvfile)  
 # 先写入列名:columns\_name 日期 最高气温 最低气温 天气  
 writer.writerow(["日期", "最高气温", "最低气温", '天气'])  
 # 一次写入多行用writerows(写入的数据类型是列表,一个列表对应一行)  
 writer.writerows([list(day\_weather\_dict.values()) for month\_weather in weathers for day\_weather\_dict in month\_weather])

**lunbo.py**

import pandas as pd  
from pyecharts import options as opts  
from pyecharts.charts import Pie ,Bar,Timeline,Page  
  
# 读csv的数据  
f = pd.read\_csv('data.csv', encoding='gb18030')  
#print("111")  
print(f['日期'])  
  
# 转换日期类型  
f['日期'] = f['日期'].apply(lambda x: pd.to\_datetime(x))  
#print("222")  
print(f['日期'])  
  
# 提取日期中的月份数据  
f['month'] = f['日期'].dt.month  
print(f['month'])  
print(f)  
# 按照月份进行分组  
f\_agg = f.groupby(['month','天气']).size().reset\_index()  
print(f\_agg)  
  
# 设置列名  
f\_agg.columns = ['month','tianqi','count']  
print(f\_agg)  
  
# 提取1月的数据，并转换为列表形式  
print(f\_agg[f\_agg['month']==1][['tianqi','count']].sort\_values(by='count',ascending=False).values.tolist())  
  
timeline = Timeline()  
timeline.add\_schema(play\_interval=1000) # 单位是:ms(毫秒)  
  
#第一个网页  
#遍历月份  
for month in f\_agg['month'].unique():  
 data = (  
 f\_agg[f\_agg['month']==month][['tianqi','count']]  
 .sort\_values(by='count',ascending=True)  
 .values.tolist()  
 )  
 #画图  
 bar = Bar()  
 # x  
 bar.add\_xaxis([x[0] for x in data])  
 # y  
 bar.add\_yaxis('',[x[1] for x in data])  
 bar.set\_series\_opts(label\_opts=opts.LabelOpts(position='top'))  
 # 图表的名称  
 bar.set\_global\_opts(title\_opts=opts.TitleOpts(title='安阳2024年每月天气变化 '))  
 # 将表放置到时间轮播图中  
 timeline.add(bar, f'{month}月')  
# 将第一个表存到weathers1文件  
timeline.render('show1.html')  
#第二个网页  
#按天气统计分组，统计天气的次数  
w\_counts = f.groupby('天气').size().reset\_index(name='count')  
#计算  
total\_count = w\_counts['count'].sum()  
w\_counts['percentage'] = (w\_counts['count'] / total\_count) \* 100  
#绘图数据  
w\_names = w\_counts['天气'].tolist() # 天气类型  
percent = w\_counts['percentage'].tolist() # 百分比  
#柱状图  
bar1 = Bar()  
bar1.add\_xaxis(w\_names)  
bar1.add\_yaxis("占比（%）", percent)  
bar1.set\_global\_opts(  
 title\_opts=opts.TitleOpts(title="2024年每种天气占比(%)"),  
 yaxis\_opts=opts.AxisOpts(name="占比（%）", axislabel\_opts=opts.LabelOpts(formatter="{value}%")),  
 xaxis\_opts=opts.AxisOpts(name="天气类型"),  
)  
#扇形图  
pie = Pie()  
pie.add(  
 "",  
 [list(z) for z in zip(w\_names, percent)],  
 radius=["25%", "50%"], # 设置内外半径  
)  
pie.set\_series\_opts(  
 label\_opts=opts.LabelOpts(formatter="{b}: {d}%") # 显示百分比  
)  
# 将两个图表放入一个容器  
page = Page()  
page.add(bar1,pie)  
# 渲染为 HTML 文件  
page.render("show2.html")