武漢理工大學

Python 数据分析与可视化

大　作　业

题　　目	武汉天气数据分析及可视化
学　　院	自动化学院
专　　业	自动化
班　　级	自动化 2005
姓　　名	陈富昌

2022 年 5 月 16 日

一、项目概述

本项目对近十年来武汉天气进行了分析以及可视化，实现了对武汉天气数据的爬取、将爬取到的数据整合分析，并将部分数据进行了可视化分析，最后将爬取到的数据作为数据集，进行训练以及测试，使用了两种算法进行预测分析，分别是线性预测以及支持向量机预测，对得出的结果进行分析比较，得出了该项目的相对较合适进行预测的算法。

项目使用到的全部的数据来源于 http://www.tianqihoubao.com/网站。采用了 requests 库进行网页请求，并且在网页请求的过程中，为了合法爬取信息，使用了 time 库进行必要的时间延时，以及 BeautifulSoup 库进行网页分析，爬取到有利与本实验调查的信息。利用 pandas 库整理信息，并将爬取到的数据利用 xlswriter 库、os 库保存至 excel 表格之中，得到原始的 excel 表格信息。

提取数据之后，利用 matplotlib.pyplot 库进行可视化，结合 pandas 和 numpy 进行数据处理，作出武汉天气近十年来的饼状图。

对于武汉空气污染的数据同理利用了 matplotlib.pyplot 和 pandas 进行数据处理和可视化处理。

在本项目中，利用了 sklearn.model_selection. train_test_split 分离训练集和测试集，对数据整合过程中运用到了 sklearn.metrics，还利用了 sklearn.linear_model 进行线性预测，还利用了 sklearn.svm 进行支持向量机的预测并用 GridSearchCV 得出最有超参数，对比两种预测方法，得出结论。

二、功能实现

具体实现流程：程序首先执行数据获取以及整理，所以先运行 ReadWHWeather.py 和 ReadWHaqi.py。得到数据以后，再运行 MergeTwoExcel.py 进行数据整合。WeatherPie.py 对 2011-2022 年的每一年，统计这一年中白天为晴、雨、多云、阴、雪、雾霾、扬沙的天数，并绘制成饼图。ContinuePollution.py 对 2014-2021 年的每一年，统计这一年中持续 1 天污染的次数、持续 2 天污染的次数、持续 3 天污染的次数、持续 4 天污染的次数和持续 5 天及以上有污染的次数，把所有年份的统计结果绘制成一幅柱状图。Prediction.py 在武汉历史天气和空气质量数据的基础上，根据当天的天气情况以及前两天的天气及空气质量情况，预测当天的空气质量等级，比较线性预测以及 SVM 两种算法，从中选出较优的算法并确定最优超参数。

1. 数据爬取

数据爬取的实现文件主要文件有两个，分别为 ReadWHWeather.py（爬取武汉天气情况）以及 ReadWHaqi.py（爬取武汉空气质量）。

```
if __name__ == '__main__':
    ExtractWHWeather()  # 爬取天气数据

    # 去除重复行的数据
    WHWeatherExcel = pd.ExcelFile('武汉天气爬虫.xlsx')
    for sheetnames in WHWeatherExcel.sheet_names:
        realWHWeatherpath = "武汉天气数据.xlsx"
        WHWeatherdf = pd.DataFrame(pd.read_excel('武汉天气爬虫.xlsx', sheet_name=sheetnames))
        deleteRe = WHWeatherdf.drop_duplicates()
        if not os.path.exists(realWHWeatherpath):
            deleteRe.to_excel(realWHWeatherpath, encoding='GBK', sheet_name=sheetnames)
        else:
            with pd.ExcelWriter(realWHWeatherpath, engine='openpyxl', mode='a', if_sheet_exists='replace') as writer:
                deleteRe.to_excel(writer, sheet_name=sheetnames)
    WHWeatherExcel.close()
```

主函数实现可以分为数据爬取和数据整理两部分。数据整理为将其中重复的数据给去除，先将原始得到的数据进行 drop_duplicates 进行去重，并保存到新的文件里面。对于数据爬取函数，经过对网页的分析，必须先爬取首页的各个链接，再到各个链接里面去一个个寻

找关键字，并按年份逐年保存到同一个 excel 中的不同 sheet 当中。

```
def ExtractWHWeather():
    WHWeatherExcel = xlsxwriter.Workbook('武汉天气爬虫.xlsx')  # 创建Excel表格武汉天气爬虫.xlsx,之后每年添加一个sheet
    url = 'http://www.tianqihoubao.com/lishi/wuhan.html'
    head = {
        'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/83.0.4103.106
Safari/537.36',
        'Connection': 'close'}  # head,防止网站知道我们是爬虫，并且请求完要关闭
```

经过对网页的分析，所有的链接数据存在于 http://www.tianqihoubao.com/lishi/wuhan.html 当中，对此网页进行链接的爬取。并进行 head 伪装。

```
html = requests.get(url, headers=head)  # 网页数据
bsObj = BeautifulSoup(html.content, 'lxml')  # 网页请求
AllYearWeather = bsObj.find_all('div', class_="wdetail")  # 所有的天气数据在此
OneYearWeather = AllYearWeather[0].find_all('div', attrs={'class': 'box pcity'})
```

一年的数据按四个季度来存放，并且全部的数据存放于 div 的 wdetail 标签里，每一年的数据在 box pcity 标签里。

```
Year = 2010
for Quarter in OneYearWeather:  # 一个季度的数据
    Year += 1
    # if Year <= 2022:  # 首先底部还有许多冗余信息会被爬进去，
    WorkSheet = WHWeatherExcel.add_worksheet(str(Year))
```

对爬取到的数据进行遍历，并且每一年对应创建一个 sheet，对应创建表头。

```
WorkSheet.write(row, col, '日期')
WorkSheet.write(row, col + 1, '天气状况')
WorkSheet.write(row, col + 2, '气温')
WorkSheet.write(row, col + 3, '风力风向')
```

一个季度的数据全部在 ul 标签里面，所以遍历取出一个月的数据。

```
for AMonth in AQuarterData:  # 一个季度内遍历每个月的数据
    ThreeMonthLink = AMonth.find_all('a')  # 每个月的链接都在一个a标签内
    for Link in ThreeMonthLink:  # 遍历一个季度（三个月）的链接，分别爬取数据
        AMonthLink = Link['href']
        if '/lishi' in str(AMonthLink):  # 2016年及之后某些年份的链接存在缺省
            AMonthurl = 'http://www.tianqihoubao.com' + AMonthLink
        else:  # 缺省处理
            AMonthurl = 'http://www.tianqihoubao.com/lishi/' + AMonthLink
```

有些链接存在缺省，比如说 2016 年之后的 12 月份都存在缺省，加个 if 判断，生成链接的数据。

```
try:  # 请求过度频繁的处理
    AMonth_HTML = requests.get(AMonthurl, headers=head)  #
    AMonthObj = BeautifulSoup(AMonth_HTML.content, 'lxml')
    AMonthData = AMonthObj.find_all('tr')  # 表格中的一行,即为
    i = 1
    for ADay in AMonthData:  # 遍历一个月数据的所有行，ADay即为
        if i == 1:  # 表格的第一行是表头，不是天气数据，排除
            i = 2
            continue
        else:
            ALine = ADay.find_all('td')  # 找出表格一行内容
            col = 0  # 从0列开始写入
            row += 1  # 比起上一次写入时，行+1
            for info in ALine:  # 遍历行的四项信息
                AnInfo = str(info.get_text())  # 获取行数据下
                AnInfo = CleanData(AnInfo)  # 除去数据中的\n
                WorkSheet.write(row, col, AnInfo)  # 数据写
                col += 1  # 列增加
except:  # 请求过快时，按5s休息处理
    print("requests speed so high,need sleep!")
    time.sleep(5)
    print("continue...")
    continue
```

用 try 语句来抛出请求过快的异常，对每一个链接，提取有效数据，并且存放于表格中，对提取到的数据进行处理，转换格式方便后续操作，并且去除多余的空格和回车符号。

上述即为数据爬取以及整理的流程，对于另外一个爬取文件，操作类似。

2. 数据的整合

对爬取到的天气数据以及空气质量数据进行整合，主要实现的文件名为 MergeTwoExcel.py。

```
if __name__ == '__main__':
    AllWHAirQualityData = pd.read_excel(r'武汉质量数据
.xlsx')  # 读取空气质量数据
    WHWeatherData = pd.ExcelFile(r'武汉天气数据.xlsx')  # 读
取天气数据
    # 因为爬取的时候为了显示好看，将数据按年份分了sheet存储
    # 这里为了合并方便需要先把天气数据合并到一起
    SheetNames = WHWeatherData.sheet_names  # 获取武汉天气爬
虫工作区的工作表名称
    AllWHWeatherData = pd.DataFrame()
    for i in SheetNames:
        OneYearData = pd.read_excel('武汉天气数据.xlsx',
sheet_name=i)  # 武汉天气爬虫的sheetname
        AllWHWeatherData = pd.concat([AllWHWeatherData,
OneYearData])  # 不同sheet数据级联
    Merge(AllWHAirQualityData, AllWHWeatherData)
```

对天气数据和空气质量进行级联，便利两个表格的日期信息，若相同则加入存放日期的 list 当中。

```
    for i in AllWHAirQualityData['日期']:
        for j in AllWHWeatherData['日期']:
            if i == j and i not in MergedDate and j not in
MergedDate:  # 相同的日期，存储起来
                MergedDate.append(i)
                break
```

接下来就是数据存放和整理。

```
if Date in MergedDate:
    AQIData = str(OneLineData['AQI指数']).replace('\nName: AQI指数, dtype: int64', '')[-4:].replace(' ', '')
    MergedAQI.append(AQIData)
    QuaLevelData = str(OneLineData['质量等级']).replace('\nName: 质量等级, dtype: object', '')[-5:].replace(' ', '')
    MergedQuaLevel.append(QuaLevelData)

    PM25Data = str(OneLineData['PM2.5']).replace('\nName: PM2.5, dtype: int64', '')[-4:].replace(' ', '')
    MergedPM25.append(PM25Data)

    PM10Data = str(OneLineData['PM10']).replace('\nName: PM10, dtype: int64', '')[-4:].replace(' ', '')
    MergedPM10.append(PM10Data)

    SO2Data = str(OneLineData['So2']).replace('\nName: So2, dtype: int64', '')[-4:].replace(' ', '')
    MergedSO2.append(SO2Data)

    COData = str(OneLineData['Co']).replace('\nName: Co, dtype: float64', '')[-6:].replace(' ', '')
    MergedCO.append(COData)

    NO2Data = str(OneLineData['No2']).replace('\nName: No2, dtype: int64', '')[-4:].replace(' ', '')
    MergedNO2.append(NO2Data)

    O3 = str(OneLineData['O3']).replace('\nName: O3, dtype: int64', '')[-4:].replace(' ', '')
    MergedO3.append(O3)
```

```
remove_digits = str.maketrans('', '', digits)
for i in range(0, AllWHWeatherDataRows):
    OneLineData = AllWHWeatherData[i:i + 1]
    # Date = str(OneLineData['日期']).replace('\nName: 日期, dtype: datetime64[ns]', '')[-11: ].replace(' ', '')
    Date = str(OneLineData['日期']).replace('Name: 日期, dtype: object', '')[-11:-1]
    if Date in MergedDate:
        WeatherConditionData = str(OneLineData['天气状况']).translate(remove_digits)
        WeatherConditionData = WeatherConditionData.replace('\nName: 天气状况, dtype: object', '')[-10:].replace(' ',
                                                                                                          '')

        MergedWeatherCondition.append(WeatherConditionData)

        TemperatureData = str(OneLineData['气温']).replace('\nName: 气温, dtype: object', '')[-10:].replace(' ', '')
        MergedTemperature.append(TemperatureData)

        WindDirectionData = str(OneLineData['风力风向']).replace('\nName: 风力风向, dtype: object', '')[-20:].replace(' ',
                                                                                                            '')
        while WindDirectionData[0] in ['0', '1', '2', '3', '4', '5', '6', '7', '8', '9']:  # 修正格式
            WindDirectionData = WindDirectionData[1:]
```

```
df1 = pd.DataFrame({'日期': MergedDate})
df2 = pd.DataFrame({'AQI': MergedAQI})
df3 = pd.DataFrame({'质量等级': MergedQuaLevel})
df4 = pd.DataFrame({'PM2.5': MergedPM25})
df5 = pd.DataFrame({'PM10': MergedPM10})
df6 = pd.DataFrame({'So2': MergedSO2})
df7 = pd.DataFrame({'Co': MergedCO})
df8 = pd.DataFrame({'No2': MergedNO2})
df9 = pd.DataFrame({'O3': MergedO3})
df10 = pd.DataFrame({'天气状况': MergedWeatherCondition})
df11 = pd.DataFrame({'气温': MergedTemperature})
df12 = pd.DataFrame({'风力风向': MergedWindDirection})

result = pd.concat([df1, df2, df3, df4, df5, df6, df7, df8, df9, df10, df11, df12], axis=1)
result.to_excel('武汉合并数据.xlsx', sheet_name='合并数据', startcol=0, index=False)
return
```

3. 绘制饼图

```
if __name__ == '__main__':
    WHWeatherData = pd.ExcelFile(r'武汉天气数据.xlsx')
    # 数据按年份分了sheet存储,每次提取一个sheet即可
    SheetNames = WHWeatherData.sheet_names  # 获取武汉
```

读取武汉的天气数据并开始绘图。

```
plt.figure(1)

label = [u'晴', u'雨', u'多云', u'阴']
color = ['green', 'purple', 'lightskyblue', 'red']

k = 1
plt.rcParams['font.sans-serif'] = ['SimHei']
plt.rcParams['axes.unicode_minus'] = False
```

设置饼图的标签以及颜色还有字体。

```
for i in SheetNames:  # i为年份
    OneYearData = pd.read_excel('武汉天气数据.xlsx', sheet_name=i)
    OneYearData = list(OneYearData['天气状况'])
    PieData = StatisticOneYear(OneYearData)  # 获得天数统计的list
    ax = plt.subplot(3, 4, k)  # 选中子区域
    plt.pie(PieData, radius=1, labels=label, labeldistance=1.2, colors=color, startangle=90, shadow=True,
            autopct=absolute_value)
    # 标签存在重叠问题，加入labeldistance参数
    ax.set_title(str(i) + '年天气状况统计',fontsize=10)
    k += 1
    if k == 13:
        break
plt.savefig('./WeatherPie.jpg')
plt.show()
```

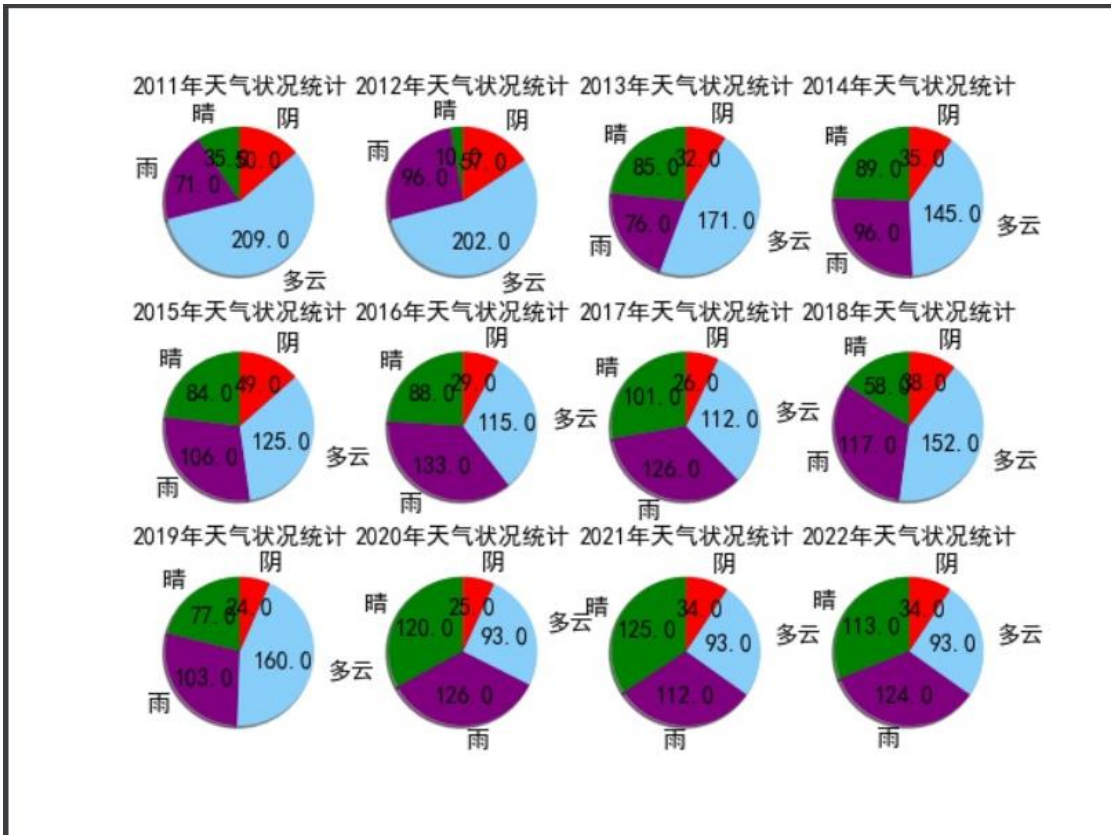
遍历每一年的数据并且保存图片。

```
def StatisticOneYear(OneYearData):
    label = ['晴', '雨', '多云', '阴']
    WeatherConditionDict = {}
    AllWeatherConditionDict = {}

    for WeatherConditionName in label:
        WeatherConditionDict[WeatherConditionName] = 0

    for OneDayData in OneYearData:
        DayData = OneDayData.split('/')[0]  # 拆分天气，只保留左边，即白天
        if DayData in AllWeatherConditionDict:
            AllWeatherConditionDict[DayData] += 1
        else:
            AllWeatherConditionDict[DayData] = 1
```

再进行数据整合的时候，将数据的白天黑夜部分做处理，只保留白天的部分。

```
for WeatherConditionName in AllWeatherConditionDict:
    if WeatherConditionName == '晴':
        WeatherConditionDict['晴'] += AllWeatherConditionDict[WeatherConditionName]
    # elif '雾' in WeatherConditionName or '霾' in WeatherConditionName:
    #     WeatherConditionDict['雾霾'] += AllWeatherConditionDict[WeatherConditionName]#雾霾数据太少，不算
    elif '雨' in WeatherConditionName:
        WeatherConditionDict['雨'] += AllWeatherConditionDict[WeatherConditionName]
    elif WeatherConditionName == '多云':
        WeatherConditionDict['多云'] += AllWeatherConditionDict[WeatherConditionName]
    # elif '雪' in WeatherConditionName and WeatherConditionName != '雨夹雪':
    #     WeatherConditionDict['雪'] += AllWeatherConditionDict[WeatherConditionName]#雪的数据太少
    elif WeatherConditionName == '阴':
        WeatherConditionDict['阴'] += AllWeatherConditionDict[WeatherConditionName]
```

雾雪的数据太少，不用于画图。



做出的图片效果如图所示。可以看出，武汉的天气以雨天和多云天气为主，这造成了武汉的雨水过多，多雨时节天气过于潮湿。并且，雪天和雾霾的数据已经省略，这表明，武汉的天气状况，从表面的天气状况来观察，还是较为适宜居住的。雨水较多，用来发展农业还是可以的。

4. 连续污染天数的柱状图绘制

收集连续污染天数的数据，可以直观的看到污染的具体情况。连续污染的 5 天数量越多，则那一年的污染就越严重。

```
if __name__ == '__main__':
    WHAirData = pd.read_excel(r'武汉合并数据.xlsx')  # 读
    YearAirData = SplitToOneYear(WHAirData)  # 按年份把污
    AllYearStatistic = PolluDaysStatistic(YearAirData)
    DrawConPollution(AllYearStatistic)  # 绘制柱状图
```

首先读取数据，其次将数据按年份分割。

```
def SplitToOneYear(WHAirData):
    WHAirData_Dict = WHAirData.to_dict('records')  # dataframe转化为d
    YearAirData = {}
    for i in range(2014, 2022):  # 14-21年数据Dict，年份为key
        YearAirData[str(i)] = []  # 不同index的存放不同year的数据

    # 按年份提取每一天的质量等级
    for OneDay in WHAirData_Dict:
        Year = OneDay['日期'].split('-')[0].replace('\ufeff', '', 1)
  是数据中的一个乱码
        if Year != '2013' and Year != '2022':  # 因为2013和2022数据不完
            YearAirData[Year].append(OneDay['质量等级'])
    return YearAirData
```

将数据转换为字典形式，并且将提取拥有完全年份数据的质量等级数据，由此，我们只能获得 2014 至 2021 的数据。

```
def PolluDaysStatistic(YearAirData):
    AllYearStatistic = []
    for OneYearData in YearAirData:
        DaysNum = len(YearAirData[OneYearData])
        OneYearStatistic = {'连续一天': 0, '连续两天': 0, '连续三天': 0,
                            '连续四天': 0, '连续五天及以上': 0}
```

统计污染情况，并将数据存入字典当中。按 if 判断来分割数据。

连续一天污染的分法：

1. 该年的第一天：当前天有污染，之后一天无污染
2. 该年最后一天：当前天有污染，前面一天无污染
3. 该年中间：当前天有污染，前面一天无污染，之后一天无污染

```
for i in range(0, DaysNum):
    if i == 0 and ('污染' in YearAirData[OneYearData][i]) and \
            ('污染' not in YearAirData[OneYearData][i + 1]):
        OneYearStatistic['连续一天'] += 1
    elif i == DaysNum - 1 and ('污染' in YearAirData[OneYearData][i]) and
            ('污染' not in YearAirData[OneYearData][i - 1]):
        OneYearStatistic['连续一天'] += 1
    elif ('污染' in YearAirData[OneYearData][i]) and \
            ('污染' not in YearAirData[OneYearData][i - 1]) and \
            ('污染' not in YearAirData[OneYearData][i + 1]):
        OneYearStatistic['连续一天'] += 1
```

连续两天的分法：

1. 非该年的最后一天：当前天有污染，前一天有污染，前二天无污染，后一天无污染
2. 该年的最后一天：当前天有污染，前一天有污染，前二天无污染

```
elif i == DaysNum - 1 and ('污染' in YearAirData[OneYearData][i]) and \
        ('污染' in YearAirData[OneYearData][i - 1]) and \
        ('污染' not in YearAirData[OneYearData][i - 2]):
    OneYearStatistic['连续两天'] += 1
elif i >= 1 and ('污染' in YearAirData[OneYearData][i]) and \
        ('污染' in YearAirData[OneYearData][i - 1]) and \
        ('污染' not in YearAirData[OneYearData][i - 2]) and \
        ('污染' not in YearAirData[OneYearData][i + 1]):
    OneYearStatistic['连续两天'] += 1
```

连续三天的分法：

1.非该年的最后一天：当前天有污染，前一天有污染，前二天有污染，前三天无污染，后一天无污染

2. 该年的最后一天：当前天有污染，前一天有污染，前二天有污染，前三天无污染

```
elif i == DaysNum - 1 and ('污染' in YearAirData[OneYearData][
        ('污染' in YearAirData[OneYearData][i - 1]) and \
        ('污染' in YearAirData[OneYearData][i - 2]) and \
        ('污染' not in YearAirData[OneYearData][i - 3]):
    OneYearStatistic['连续三天'] += 1
elif i >= 2 and ('污染' in YearAirData[OneYearData][i]) and \
        ('污染' in YearAirData[OneYearData][i - 1]) and \
        ('污染' in YearAirData[OneYearData][i - 2]) and \
        ('污染' not in YearAirData[OneYearData][i - 3]) and \
        ('污染' not in YearAirData[OneYearData][i + 1]):
    OneYearStatistic['连续三天'] += 1
```

连续四天的分法：

1. 非该年的最后一天：当前天有污染，前一天有污染，前二天有污染，前三天有污染，前四天无污染，后一天无污染

2. 该年的最后一天：当前天有污染，前一天有污染，前二天有污染，前三天有污染，前四天无污染

```
elif i == DaysNum - 1 and ('污染' in YearAirData[OneYearData][i]) and
        ('污染' in YearAirData[OneYearData][i - 1]) and \
        ('污染' in YearAirData[OneYearData][i - 2]) and \
        ('污染' in YearAirData[OneYearData][i - 3]) and \
        ('污染' not in YearAirData[OneYearData][i - 4]):
    OneYearStatistic['连续四天'] += 1
elif i >= 3 and ('污染' in YearAirData[OneYearData][i]) and \
        ('污染' in YearAirData[OneYearData][i - 1]) and \
        ('污染' in YearAirData[OneYearData][i - 2]) and \
        ('污染' in YearAirData[OneYearData][i - 3]) and \
        ('污染' not in YearAirData[OneYearData][i - 4]) and \
        ('污染' not in YearAirData[OneYearData][i + 1]):
    OneYearStatistic['连续四天'] += 1
```

连续五天以上的分法：

1. 非该年的最后一天：当前天有污染，前一天有污染，前二天有污染，前三天有污染，前四天有污染，后一天无污染

2. 该年的最后一天：当前天有污染，前一天有污染，前二天有污染，前三天有污

染，前四天有污染

```
elif i == DaysNum - 1 and ('污染' in YearAirData[OneYearData][i]) and
        ('污染' in YearAirData[OneYearData][i - 1]) and \
        ('污染' in YearAirData[OneYearData][i - 2]) and \
        ('污染' in YearAirData[OneYearData][i - 3]) and \
        ('污染' in YearAirData[OneYearData][i - 4]):
    OneYearStatistic['连续五天及以上'] += 1
elif i >= 4 and ('污染' in YearAirData[OneYearData][i]) and \
        ('污染' in YearAirData[OneYearData][i - 1]) and \
        ('污染' in YearAirData[OneYearData][i - 2]) and \
        ('污染' in YearAirData[OneYearData][i - 3]) and \
        ('污染' in YearAirData[OneYearData][i - 4]) and \
        ('污染' not in YearAirData[OneYearData][i + 1]):
    OneYearStatistic['连续五天及以上'] += 1
```

分好情况之后将所有数据保存在 list 中。

```
    AllYearStatistic.append(OneYearStatistic)
return AllYearStatistic
```

得到数据，开始画图。首先分割数据。

```
def DrawConPollution(AllYearStatistic):
    y1 = []
    y2 = []
    y3 = []
    y4 = []
    y5 = []
    for OneYearData in AllYearStatistic:
        y1.append(OneYearData['连续一天'])
        y2.append(OneYearData['连续两天'])
        y3.append(OneYearData['连续三天'])
        y4.append(OneYearData['连续四天'])
        y5.append(OneYearData['连续五天及以上'])
    width = 0.16  # 设置柱与柱之间的宽度
```

设置宽度和横坐标还有 Bar 的属性。

```
width = 0.16  # 设置柱与柱之间的宽度

x1 = range(len(y1))  # 横坐标
x2 = [i + width for i in x1]
x3 = [i + width for i in x2]
x4 = [i + width for i in x3]
x5 = [i + width for i in x4]
# Bar的属性
Bar1 = plt.bar(x1, y1, width=0.16, alpha=0.9, color='purple')
Bar2 = plt.bar(x2, y2, width=0.16, alpha=0.9, color='red')
Bar3 = plt.bar(x3, y3, width=0.16, alpha=0.9, color='blue')
Bar4 = plt.bar(x4, y4, width=0.16, alpha=0.9, color='green')
Bar5 = plt.bar(x5, y5, width=0.16, alpha=0.9, color='yellow')
```

设置刻度和字体。

```
Year = ['2014', '2015', '2016', '2017', '2018', '2019', '2020', '2021'
plt.rcParams['font.sans-serif'] = ['SimHei']
plt.rcParams['axes.unicode_minus'] = False
```

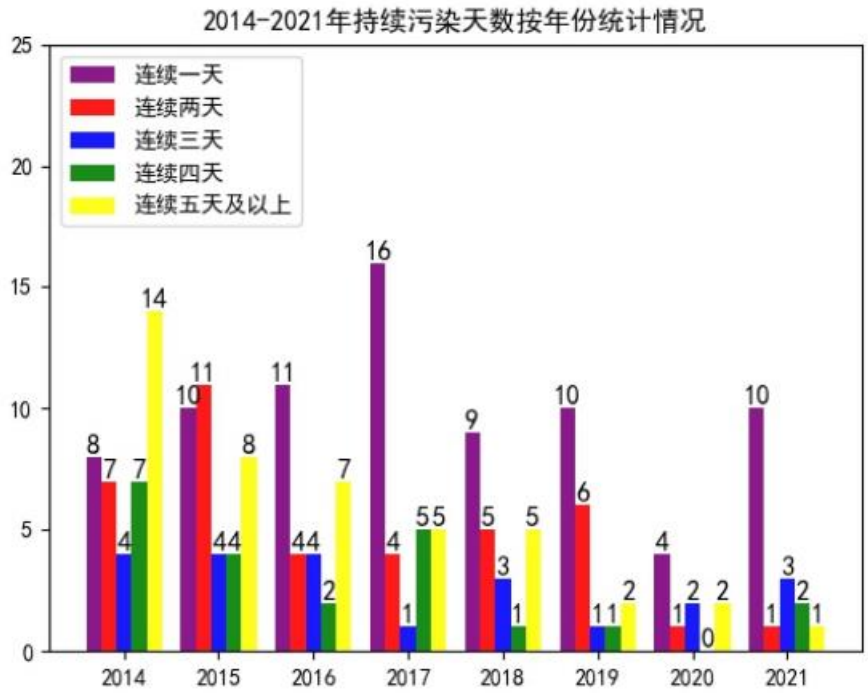
设置图例和图题。

```
plt.xticks([i + 2 * width for i in x1], Year)
plt.yticks([0, 5, 10, 15, 20, 25])
plt.legend(['连续一天', '连续两天', '连续三天', '连续四天', '连续五天及以上'],
           loc="upper left")
plt.title('2014-2021年持续污染天数按年份统计情况')
```

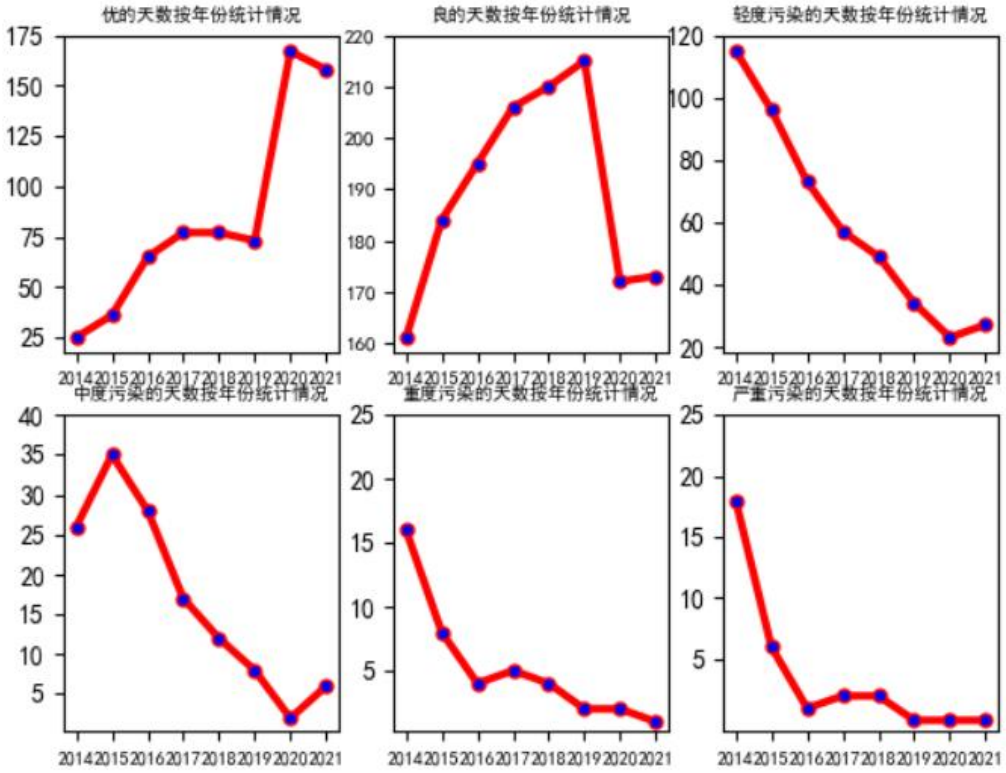
对所有的 Bar 进行数字标注。

```
for rect in Bar1:
    height = rect.get_height()
    plt.text(rect.get_x() + rect.get_width() / 2, height, str(height), size=12, ha='center', va='bottom')
for rect in Bar2:
    height = rect.get_height()
    plt.text(rect.get_x() + rect.get_width() / 2, height, str(height), size=12, ha='center', va='bottom')
for rect in Bar3:
    height = rect.get_height()
    plt.text(rect.get_x() + rect.get_width() / 2, height, str(height), size=12, ha='center', va='bottom')
for rect in Bar4:
    height = rect.get_height()
    plt.text(rect.get_x() + rect.get_width() / 2, height, str(height), size=12, ha='center', va='bottom')
for rect in Bar5:
    height = rect.get_height()
    plt.text(rect.get_x() + rect.get_width() / 2, height, str(height), size=12, ha='center', va='bottom')
plt.show()
```

最终得出图形。



由图可以看出，连续污染五天以上的天数逐年下降，说明空气质量表面上看，还是逐年上升的。可以看出 2020 年的污染最小，可能由 2020 年爆发疫情引起的，居民少出户，私家车的排放减少，对空气质量的改善有着很大的帮助。从下图的各年份空气质量为优的直线图也可以看得出来。



重度污染等天气情况逐年下降。

5. 根据天气数据预测天气质量

实现的程序是 Prediction.py。主要进行了两种预测，一种是多元线性回归预测，还有一种是支持向量机预测。

```
if __name__ == '__main__':
    WHMergedData = pd.read_excel(r'武汉合并数据.xlsx')
取武汉合并数据,得到DataFrame
    EncodedList = EncodeData(WHMergedData)  # 当天+前
编码放入一个list
    LinearRegPre(EncodedList)  # 多元线性回归预测
    SVMPre(EncodedList)  # 支持向量机分类预测
```

整体思路为获得各种天气的空气质量数据（AQI，PM2.5 等）和天气状况数据，来预测质量等级这个数据。首先建立训练集和测试集。然后将一些中文字符进行编码，比如说空气质量情况等。做好数据集之后，用 sklenrn.model_selection 的 train_test_split 模块根据 3：7 的比例做数据分割。分割完就设置超参数，训练，预测，得出结果。

两种预测结果的效果如下。

```
Linear训练结束!
多元线性回归结果:
准确率:  0.9581497797356828
精确率:  0.9582685037934636
召回率:  0.9581497797356828
Linear预测结束!
SVM训练结束!
支持向量机结果:
最优超参数:  {'C': 0.56, 'gamma': 0.01, 'kernel': 'linear'}
准确率:  0.9922907488986784
精确率:  0.9923530878563711
召回率:  0.9922907488986784
SVM预测结束!
```

明显可知，支持向量机的预测结果优于多元线性回归的结果。