南京师范大学地理科学学院

2021-2022学年 第二学期

《Python与空间信息处理》期末考试

姓　　名:朱沁韦

学　　号:10200409

任课教师:乐松山

南京师范大学 地理科学学院《Python 与地理信息处理》2022 年期末测试

姓　　名	**朱沁韦**
学　　号	**10200409**
专　　业	**地理信息科学**
任课老师	**乐松山**
完成时间	**2022.6.28**

作业要求：

（1） 答题内容包括两个部分：PDF 格式的报告，每个题目的代码文件。PDF 文档按照学号+姓名命名（示例：10200436 罗小.pdf）；代码文件按照题目编号命名（示例：T1.py）；如有结果文件上传， 自定义不重复、不冲突、合理的文件名即可。

（2） 找到对应上课班级☮“期末作业”文件夹☮自己学号+姓名文件夹，提交 PDF 报告和 Python 代码文件 。注意： **不要提交压缩包** ， **单独上传每个文件** 。 网址 http://1.13.163.229/visualCoursePlatform/classList

（3） PDF 格式的报告，封面就是本页面。正文是题目必要的思路说明、运行结果和代码截图。

（4） PDF 报告的封面（即本页面），右上角有“照片+学号+姓名”，需要利用 Pillow 或者其它库读取自己的照片，并在照片上用代码写上学号和姓名，示例如文档中的图。注意如何处理中文，请查找网络相关资料解决。

（5） 每位同学需独立完成，鼓励使用各种文档、网络资源。完成时间截止到 2022 年 6 月 28 日 24: 00。

题目 1（10 分）

读取自己的真实照片，在照片上用代码写上学号和姓名，文字位置、大小、颜色自己指定，合理好看即可。生成的照片请贴到封面的右上角。

答题思路：先进行照片读取，然后运用 axe.annotate 函数用箭头和文字注明自己的身份。解决中文显示问题。

代码：

#读取图像

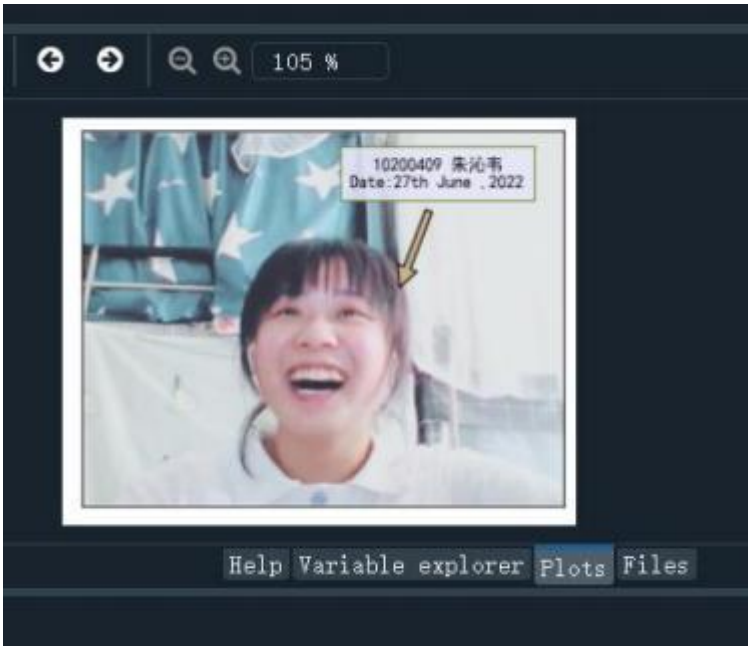
```
import matplotlib.pyplot as plt
import numpy as np
#解决中文显示问题
plt.rcParams['font.sans-serif']= ['SimHei']
plt.rcParams['axes.unicode_minus'] = False

img=plt.imread("E:\派森\期末\T1\my_picture.png")
axe=plt.subplot(111)
axe.imshow(img)
#axe.set_title("我的哈哈照")
axe.annotate('10200409 朱沁韦\nDate:27th June ,2022',
                    bbox={'facecolor':'Lavender',
                      'edgecolor':'OliveDrab',
                      'alpha':0.9,#类似于像素透明度，0 是透明
                      'pad':6#填充形状的大小
                           },
                    xy=(750,400),#此时的文字会默认出现在右边
                    xytext=(850, 100),#划定文本开头的位置
                    arrowprops=dict(facecolor='Tan' ,shrink=0.1),#箭头收缩
                    horizontalalignment='center',#平面线形
                    verticalalignment='center')#垂直线性
axe.set_xticks([])
axe.set_yticks([])
```

plt.savefig("E:\派森\期末\T1\my_picture1.png")

plt.show()

结果展示：

```
# -*- coding: utf-8 -*-
"""
Created on Fri Jun 24 21:09:13 2022

@author: Hello
"""

#读取图像

import matplotlib.pyplot as plt
import numpy as np
#解决中文显示问题
plt.rcParams['font.sans-serif']=['SimHei']
plt.rcParams['axes.unicode_minus'] = False

img=plt.imread("E:\派森\期末\T1\my_picture.png")
axe=plt.subplot(111)
axe.imshow(img)
#axe.set_title("我的哈哈照")
axe.annotate('10200409 朱沁韦\nDate:27th June ,2022',
             bbox={'facecolor':'Lavender',
               'edgecolor':'OliveDrab',
               'alpha':0.9,#类似于像素透明度，0是透明
               'pad':6#填充形状的大小
                   },
             xy=(750,400),#此时的文字会默认出现在右边
             xytext=(850,100),#划定文本开头的位置
             arrowprops=dict(facecolor='Tan' ,shrink=0.1),#箭头收缩
             horizontalalignment='center',#平面线形
             verticalalignment='center')#垂直线性
axe.set_xticks([])
axe.set_yticks([])

plt.savefig("E:\派森\期末\T1\my_picture1.png")
plt.show()
```



题目 2（15 分）

平面上有矩形 A 和矩形 B，它们的边分别平行于 X 轴和 Y 轴。编写一个程序，提示用户输入矩形 A 和矩形 B 在对角线上的顶点坐标，计算两个矩形的公共部分的面积（计算结果保存小数点后两位）。

解题思路：思路 1：先输入 2 个矩形的对角线顶点坐标，将它们赋给 a 和 b 两个列表中，找出 Xmin,Xmax,Ymin,Ymax, 在 Xmin<Xmax 并且 Ymin<Ymax 的情况下用函数求出相交部分的面积

思路 2：先通过 2 个矩形的对角线顶点得到两个矩形，然后通过 shapely 库运用几何对象的通用属性和方法。运用相交函数 bject.intersects(other)求得两个矩形的公共部分的面积。

代码：

思路 1：

```
def intersection_solution(a,b):
    if a [0]>a [2] :
        a [0],a [2] = a[2],a [0]
    if a [ 1]>a [3] :
        a [ 1],a [3] = a[3],a [ 1]
    if b[0]>b[2] :
        b[0],b[2] = b[2],b[0]
    if b[1]>b[3] :
        b[1],b[3] = b[3],b[1]
     x1 = max(a [0],b[0])
     y1 = max(a [ 1],b[1])
     x2 = min(a [2],b[2])
     y2 = min(a [3],b[3])
    if y2<y1 or x2<x1:
        c = 0
    else:
c = (y2-y1)*(x2-x1)
```

```
        print('\n')
        print('结果如下')
        print("{ :.2f}" .format(c))
```

思路 2：

```
def solution_merge(a,b):
    if a [0]>a [2] :
        a [0],a [2] = a[2],a [0]
    if a [ 1]>a [3] :
        a [ 1],a [3] = a[3],a [ 1]
    if b[0]>b[2] :
        b[0],b[2] = b[2],b[0]
    if b[1]>b[3] :
        b[1],b[3] = b[3],b[1]
    polygon1=Polygon([(a [0],a [ 1]),(a [0],a [3]),(a [2],a [3]),(a [2],a [ 1])])
    polygon2=Polygon([(b[0],b[1]),(b[0],b[3]),(b[2],b[3]),(b[2],b[1])])
    c=polygon1.intersection(polygon2)
    q=c.area
    print('\n')
    print('结果如下')
    print("%.2f"%q)
```

代码：

```
# -*- coding: utf-8 -*-
"""
Created on Sat Jun 25 20:19:08 2022

@author: Hello
"""

#期末 T2
```

#平面上有矩形 A 和矩形 B，它们的边分别平行于 X 轴和 Y 轴。

#编写一个程序，提示用户输入矩形 A 和矩形 B 在对角线上的顶点坐标，计算两个矩形的公共部分的面积（计算结果保存小数点后两位）。

```
from shapely.geometry import Polygon
def solution_merge(a,b):
    if a[0]>a[2]:
        a[0],a[2] = a[2],a[0]
    if a[1]>a[3]:
        a[1],a[3] = a[3],a[1]
    if b[0]>b[2]:
        b[0],b[2] = b[2],b[0]
    if b[1]>b[3]:
        b[1],b[3] = b[3],b[1]

    polygon1=Polygon([(a[0],a[1]),(a[0],a[3]),(a[2],a[3]),(a[2],a[1])])
polygon2=Polygon([(b[0],b[1]),(b[0],b[3]),(b[2],b[3]),(b[2],b[1])])

    c=polygon1.intersection(polygon2)
    q=c.area
    print('\n')
    print('结果如下')
print("%.2f"%q)

def intersection_solution(a,b):
    if a[0]>a[2]:
        a[0],a[2] = a[2],a[0]
    if a[1]>a[3]:
        a[1],a[3] = a[3],a[1]
    if b[0]>b[2]:
        b[0],b[2] = b[2],b[0]
```

```
    if b[1]>b[3] :
        b[1],b[3] = b[3],b[1]
    x1 = max(a [0],b[0])
    y1 = max(a [ 1],b[1])
    x2 = min(a [2],b[2])
    y2 = min(a [3],b[3])
    if y2<y1 or x2<x1:
        c = 0
    else:
        c = (y2-y1)*(x2-x1)
    print('\n')
    print('结果如下')
    print("{ :.2f}".format(c))
if __name__ == '__main__' :
    print('请输入矩形 1 和矩形 2 的对线顶点坐标，一个矩形一行，共两行\n')
    print('矩形 1')
    a = list(map(float, input().split()))
    print('\n')
    print('矩形 2')
    b = list(map(float, input().split()))

    intersection_solution(a,b)

    solution_merge(a,b)
```

```
# -*- coding: utf-8 -*-
"""
Created on Sat Jun 25 20:10:08 2022

@author: Hello
"""

#期末T2
#平面上有矩形A和矩形B，它们的边分别平行于x轴和y轴。
#编写一个程序，提示用户输入矩形A和矩形B在对角线上的顶点坐标，计算两个矩形的公共部分的面
from shapely.geometry import Polygon
def solution_merge(a,b):
    if a[0]>a[2]:
        a[0],a[2] = a[2],a[0]
    if a[1]>a[3]:
        a[1],a[3] = a[3],a[1]
    if b[0]>b[2]:
        b[0],b[2] = b[2],b[0]
    if b[1]>b[3]:
        b[1],b[3] = b[3],b[1]


    polygon1=Polygon([(a[0],a[1]),(a[0],a[3]),(a[2],a[3]),(a[2],a[1])])
    polygon2=Polygon([(b[0],b[1]),(b[0],b[3]),(b[2],b[3]),(b[2],b[1])])

    c=polygon1.intersection(polygon2)
    q=c.area
    print('\n')
    print('结果如下')
    print("%.2f"%q)

def intersection_solution(a,b):


    if a[0]>a[2]:
        a[0],a[2] = a[2],a[0]
    if a[1]>a[3]:
        a[1],a[3] = a[3],a[1]
```

```
def intersection_solution(a,b):


    if a[0]>a[2]:
        a[0],a[2] = a[2],a[0]
    if a[1]>a[3]:
        a[1],a[3] = a[3],a[1]
    if b[0]>b[2]:
        b[0],b[2] = b[2],b[0]
    if b[1]>b[3]:
        b[1],b[3] = b[3],b[1]

    x1 = max(a[0],b[0])
    y1 = max(a[1],b[1])
    x2 = min(a[2],b[2])
    y2 = min(a[3],b[3])


    if y2<y1 or x2<x1:
        c = 0
    else:
        c = (y2-y1)*(x2-x1)
    print('\n')
    print('结果如下')
    print("{:.2f}".format(c))


if __name__ == '__main__':
    print('请输入矩形1和矩形2的对线顶点坐标，一个矩形一行，共两行\n')
    print('矩形1')
    a = list(map(float, input().split()))
    print('\n')
    print('矩形2')
    b = list(map(float, input().split()))

    intersection_solution(a,b)

    solution_merge(a,b)
```

```
In [26]: runfile('E:/资源/期末/T2/T2.py', wdir='E:/资源/期末/T2')
请输入矩形1和矩形2的对线顶点坐标，一个矩形一行，共两行

矩形1

2 2 5 5


矩形2

3 3 7 7


结果如下
4.00


结果如下
4.00

In [27]:
```

题目 3（15 分）

根据期末测试文件夹中的“POI.csv”，“county_pop.shp”等数据，生成上海市范围内的 17 个区/县的 POI 矢量数据（Shapefile 格式）。命名以区/县+POI 命名（示例：“徐汇区 POI.shp”）。

【相关要点：读取文件、判断点是否在多边形内、创建 Shapefile；也有其他方法】

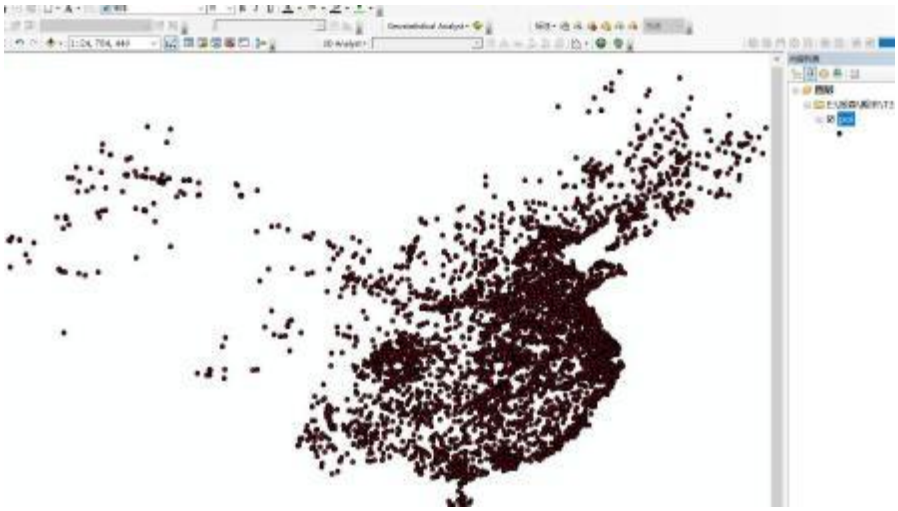
【注意 county_pop.shp 是 Web Mercator 投影，POI 里面的坐标是经纬度，注意处理投影不一致的问题】

【生成的 Shapefile 结果，使用 ArcGIS、QGIS、SuperMap 等软件截图】

解题思路： ①通过 POI.csv 创建 poi.shp，给 poi.shp 添加和 county_pop.shp 一样的投影。②根据 country_popu.shp 分别生成上海 17 个区县的 17 个 shp 图层。③通过 poi.shp 和各区县.shp 的相交操作获得各个区县的 shp 文件。

代码 1：生成 poi 的 shp 文件

```
# -*- coding: utf-8 -*-

"""
Created on Tue Jun 28 17:06:30 2022

@author: Hello
"""
import pandas as pd
import geopandas
import pyproj
#添加墨卡托投影的 poi 矢量文件
df= geopandas.read_file(r"E:\派森\期末\T3\POI.csv",encoding='utf-8')
df['Lon'] = df['Lon'].apply(pd.to_numeric)
df['Lat'] = df['Lat'].apply(pd.to_numeric)
gdf = geopandas.GeoDataFrame(df, geometry=geopandas.points_from_xy(df.Lon, df.Lat))
gdf.crs = pyproj.CRS.from_user_input('epsg:4326') #给输出的 shp 增加投影
# gdf.rename(columns={'addrees':'area_ave_price'},inplace=True)
gdf.to_file(r"E:\派森\期末\T3\poi.shp", driver='ESRI Shapefile',encoding='utf-8')
```



代码 2：生成各区县的面 shp 图层

先找出所在曲线的 PAC 代码，存到新文件中，再给新的区县 shp 图层添加投影。然后输出带投影的区县 shp 。代码以黄浦区为例。

```
from osgeo import gdal
from osgeo import ogr
from osgeo import osr


'''
通过 Filter 读取 Shapefile 中属性过滤后的数据
'''
shape_path = "E:\派森\期末\T3\county_popu.shp"
ds = ogr.Open(shape_path)
layer = ds.GetLayer(0)
srs = layer.GetSpatialRef()
layer_defn = layer.GetLayerDefn()
field_count = layer_defn.GetFieldCount()
filed_name_list = [] #暂存原始 Shapefile 的属性字段的名字
for iField in range(0, field_count):
    temp_field = layer_defn.GetFieldDefn(iField)
    temp_name = temp_field.GetName()
    filed_name_list.append(temp_name)
```

```
        print(temp_name)
#上海是 310 开头
layer.SetAttributeFilter('("PAC"/ 1)=310101')
feature_count = layer.GetFeatureCount()

layer.ResetReading()
feature_list = [] #暂存过滤后的 Feature 数据
temp_feature = layer.GetNextFeature()
while temp_feature:
        temp_geom = temp_feature.GetGeometryRef()
        temp_field_area = temp_feature.GetField('Shape_Area')
        temp_field_name = temp_feature.GetField('NAME')
        temp_calculatet_area = temp_geom.Area()
        print('Name:', temp_field_name,
                'Area:', temp_field_area,
                'Cal_Area:', temp_calculatet_area)
        feature_list.append(temp_feature)
        temp_feature = layer.GetNextFeature()



'''
将过滤后的 Feature 保存到新的文件中
'''
gdal.SetConfigOption("SHAPE_ENCODING", "UTF-8") # IMPORTANT
new_shape_path = 'E:/派森/期末/T3/shanghai_黄浦区.shp'

driverName = "ESRI Shapefile"
driver = ogr.GetDriverByName(driverName)
ds = driver.CreateDataSource(new_shape_path) #创建 DataSource
```

```
geomType = ogr.wkbPolygon #指定几何类型
#创建图层，注意 srs 借用的是原来的
poLayer = ds.CreateLayer('shanghai_黄浦区', srs, geomType)

#对原始数据的属性字段进行遍历
for iField in range(0, field_count):
    temp_field = layer_defn.GetFieldDefn(iField)
    poLayer.CreateField(temp_field, 1)

#创建完所有的属性字段之后，获取整个图层的定义
new_layerdef = poLayer.GetLayerDefn()

for iF in range(0, feature_count):
    temp_old_feature = feature_list[iF]
    temp_geom = temp_old_feature.GetGeometryRef()

    tempNewFeature = ogr.Feature(new_layerdef)
    tempNewFeature.SetGeometry(temp_geom)
    for temp_name in filed_name_list :
        tempNewFeature.SetField(temp_name,temp_old_feature.GetField(temp_name))

    poLayer.CreateFeature(tempNewFeature)
    del tempNewFeature

ds.FlushCache()
if ds != None: ds.Destroy()

'''
将过滤后的 Feature 保存到新的文件中，并且做投影转换
```

```
'''
new_shape_path = "E:\派森\期末\T3\shanghai_wgs84_黄浦区.shp"

driverName = "ESRI Shapefile"
driver = ogr.GetDriverByName(driverName)
ds = driver.CreateDataSource(new_shape_path)

geomType = ogr.wkbPolygon
srs_4326 = osr.SpatialReference(osr.SRS_WKT_WGS84_ LAT_LONG) # 指定新的空间参考
srs_4326.SetAxisMappingStrategy(osr.OAMS_TRADITIONAL_GIS_ORDER) # 按照传统先 X 后 Y 的顺序进行坐标转换
#注意此处，指定了 options=["ENCODING=UTF-8"]，会生成了以 *.cpg 的文件，里面给出了文字编码
layerjs_4326 = ds.CreateLayer('sh_黄浦区', srs_4326, geomType, options=["ENCODING=UTF-8"])

#选用了某几个属性字段放到新的 Shapefile 中
field1 = ogr.FieldDefn("PAC", ogr.OFTInteger64)
layerjs_4326.CreateField(field1, 1)
field1.Destroy()

field2 = ogr.FieldDefn("NAME", ogr.OFTString)
layerjs_4326.CreateField(field2, 1)
field2.Destroy()

field3 = ogr.FieldDefn("AREA", ogr.OFTReal)
layerjs_4326.CreateField(field3, 1)
field3.Destroy()

#获取新数据的图层属性
layerjs_def = layerjs_4326.GetLayerDefn()
```

```
#根据原始坐标参考和新的坐标参考，构建一个投影转换器
trans = osr.CoordinateTransformation(srs, srs_4326)

for iF in range(0, feature_count):
    temp_feature = feature_list[iF]

    temp_geom = temp_feature.GetGeometryRef() # 拿到 Feature 的 Geometry 对象
    temp_geom_type = temp_geom.GetGeometryType() # 拿到 Geometry 的几何类型
    temp_geom_count = temp_geom.GetGeometryCount() # 有可能是 Multi- Polygon

    temp_new_polygon = None
    if temp_geom_type == ogr.wkbMultiPolygon and temp_geom_count>1:
        print("geometry name:", temp_geom.GetGeometryName())
        #print(temp_geom.GetGeometryName())
        temp_new_polygon = ogr.Geometry(ogr.wkbMultiPolygon)
        for iGeo in range(0, temp_geom_count):
            temp_part = temp_geom.GetGeometryRef(iGeo) # 获取 Multi 中的每一个 Part
            temp_part_ring_count = temp_part.GetGeometryCount() # 每个 Polygon 可能会有 ExRing 和 InRing
            #print("polygon's ring count :", temp_part_ring_count)

            temp_new_part = ogr.Geometry(ogr.wkbPolygon)
            for iRing in range(0, temp_part_ring_count):
                temp_ring = temp_part.GetGeometryRef(iRing)

                temp_new_ring = ogr.Geometry(ogr.wkbLinearRing)

                # 第一种写法
                temp_ring_point_count = temp_ring.GetPointCount()
```

```
                for iP in range(0, temp_ring_point_count):
                    temp_point = temp_ring.GetPoint(iP)
                    new_p = trans.TransformPoint(temp_point[0], temp_point[1])
                    #print(new_p)
                    temp_new_ring.AddPoint(new_p [0], new_p [ 1])
                temp_new_part.AddGeometry(temp_new_ring)

            temp_new_polygon.AddGeometry(temp_new_part)
        else:
            print("geometry name:", temp_geom.GetGeometryName())
            temp_new_polygon = ogr.Geometry(ogr.wkbPolygon)

            temp_ring_count = temp_geom.GetGeometryCount() # Polygon 有可能带岛
            #print("polygon's ring count :", temp_ring_count)
            for iRing in range(0, temp_ring_count):
                temp_ring = temp_geom.GetGeometryRef(iRing) # 遍历每一个 Ring，默认第一个是外环
                import pyproj
                source_proj = pyproj.Proj(srs.ExportToProj4())
                target_proj = pyproj.Proj(4326)
                prjtrans = pyproj.Transformer.from_proj(source_proj, target_proj)
                # 注意以上四行应该写到循环外面
                temp_ring_point_count = temp_ring.GetPointCount()
                new_points = []
                for iP in range(0, temp_ring_point_count):
                    temp_point = temp_ring.GetPoint(iP)
                    new_p = prjtrans.transform(temp_point[0], temp_point[1])
                    #print(new_p) #注意，转出来的经纬度是反的
                    new_points.append((new_p [ 1], new_p [0]))
```

```
                temp_new_ring = ogr.Geometry(ogr.wkbLinearRing)
                for new_p in new_points:
                    temp_new_ring.AddPoint(new_p [0], new_p [ 1])
                temp_new_polygon. AddGeometry( temp_new_ring)
        # end if-else geometry type

        if temp_new_polygon != None:
            tempNewFeature = ogr.Feature(layerjs_def)
            tempNewFeature. SetGeometry( temp_new_polygon)
            tempNewFeature.SetField("PAC",temp_feature.GetField("PAC"))
            tempNewFeature.SetField("NAME",temp_feature.GetField("NAME"))
            tempNewFeature.SetField("AREA",temp_feature.GetField("AREA"))
            layerjs_4326.CreateFeature(tempNewFeature)
            tempNewFeature. Destroy()

ds.FlushCache()
if ds != None: ds.Destroy()
```

结果如下

代码 3：点图层与面图层相交。

```
# -*- coding: utf-8 -*-
"""
Created on Tue Jun 28 16:39:15 2022
@author: Hello
"""
import time
import fiona
import rtree
from shapely.geometry import shape, mapping
def intersect_shp_shp(manager_shp_path,input_shp_path,output_shp_path,min_area=None) :
    """
    manager_shp_path : 第一 shp 文件路径  该项目中是 林地小班
    input_shp_path   : 第二 shp 文件路径  该项目中是 算法产生的变化图斑
    output_shp_path  : 输出 shp 文件路径
    min_area         : 用于图斑面积筛选(面积小于该值的图斑，将会 pass ，也提高算法执行时间)
    return None
    """
    start_=time.time()
    with fiona.open(manager_shp_path, 'r',encoding='utf-8') as layer1:
        with fiona.open(input_shp_path, 'r',encoding='utf-8') as layer2:
            if layer1.crs['init']!=layer2.crs['init']:
                raise Exception('shapefile 的坐标系不同，无法进行计算！请先转换坐标系！')
            # 合并两 shp 的 schema
            schema3 = layer2.schema.copy()
            schema3['properties'].update(layer1.schema['properties'])
            # 新增面积属性
            schema3['properties']['area'] = 'float'
            # schema3['geometry']='Polygon'
```

```
                with fiona.open(output_shp_path, mode='w', schema=schema3,driver='ESRI Shapefile',crs_wkt=layer2.meta ['crs_wkt'],encoding='utf-8') as layer3:
                    # 建立 rtree 索引
                    print('开始建立索引.....')
                    index = rtree.index.Index()
                    manager_num=0
                    for feat1 in layer1:
                        fid = int(feat1 ['id'])
                        geom1 = shape(feat1 ['geometry'])
                        index.insert(fid, geom1.bounds)
                        manager_num+=1
                    print('plygon 数量共有： %d 个， 建立索引共耗时 %.2f s'%(manager_num,time.time()-start_))
                    # 执行相交运算
                    intersect_execute(layer1,layer2,layer3,index,min_area)
                    print('完成本次运算共耗时%.2f s'%(time.time()-start_,))


def intersect_execute(layer1,layer2,layer3,index,min_area):
    print('开始进行交集运算.....')
    result_num,polygon_num,small_num=0,0,0
    for feat2 in layer2:
        polygon_num+=1
        geom2 = shape(feat2 ['geometry'])
        if min_area and  geom2.area<min_area:
            small_num+=1
            continue
        # 检测合法性，并进行合法化(主要针对 有洞的 polygon)
        if not geom2.is_valid:
            geom2=geom2.buffer(0)
```

```
            for fid in list(index.intersection(geom2.bounds)):
                feat1 = layer1[fid]
                geom1 = shape(feat1 ['geometry'])
                if not geom1.is_valid:
                    geom1=geom1.buffer(0)
                if geom1.intersects(geom2):
                    # 合并属性
                    props = feat2 ['properties'].copy()
                    props.update(feat1 ['properties'])
                    intersect=geom1.intersection(geom2)
                    if min_area and    intersect.area<min_area:
                        continue
                    # 给 area 属性赋值
                    props['area']=intersect.area
                    try:
                        layer3.write({
                            'properties': props,
                            'geometry': mapping(intersect)
                        })
                        result_num+=1
                    except Exception as e:
                        print(e)
        print('输入POI数量 %d 个，得到符合要求的POI共： %d 个'%(polygon_num,result_num,))


if __name__ == "__main__":
    print("宝山区 poi")
    intersect_shp_shp("E:\派森\期末\T3\shanghai_wgs84_宝山区.shp","E:\派森\期末\T3\poi.shp","E:\派森\期末\T3\结果\宝山区 POI.shp")
```

```
print("崇明县 poi")
intersect_shp_shp("E:\派 森 \期 末 \T3\shanghai_wgs84_崇 明 县 .shp","E:\派 森 \期 末 \T3\poi.shp","E:\派森\期末\T3\结果\崇明县 POI.shp")
print("奉贤县 poi")
intersect_shp_shp("E:\派 森 \期 末 \T3\shanghai_wgs84_奉 贤 区 .shp","E:\派 森 \期 末 \T3\poi.shp","E:\派森\期末\T3\结果\奉贤区 POI.shp")
print("虹口区 poi")
intersect_shp_shp("E:\派 森 \期 末 \T3\shanghai_wgs84_虹 口 区 .shp","E:\派 森 \期 末 \T3\poi.shp","E:\派森\期末\T3\结果\虹口区 POI.shp")
print("黄浦区 poi")
intersect_shp_shp("E:\派 森 \期 末 \T3\shanghai_wgs84_黄 浦 区 .shp","E:\派 森 \期 末 \T3\poi.shp","E:\派森\期末\T3\结果\黄浦区 POI.shp")
print("嘉定区 poi")
intersect_shp_shp("E:\派 森 \期 末 \T3\shanghai_wgs84_嘉 定 区 .shp","E:\派 森 \期 末 \T3\poi.shp","E:\派森\期末\T3\结果\嘉定区 POI.shp")
print("金山区 poi")
intersect_shp_shp("E:\派 森 \期 末 \T3\shanghai_wgs84_金 山 区 .shp","E:\派 森 \期 末 \T3\poi.shp","E:\派森\期末\T3\结果\金山区 POI.shp")
print("静安区 poi")
intersect_shp_shp("E:\派 森 \期 末 \T3\shanghai_wgs84_静 安 区 .shp","E:\派 森 \期 末 \T3\poi.shp","E:\派森\期末\T3\结果\静安区 POI.shp")
print("闵行区 poi")
intersect_shp_shp("E:\派 森 \期 末 \T3\shanghai_wgs84_闵 行 区 .shp","E:\派 森 \期 末 \T3\poi.shp","E:\派森\期末\T3\结果\闵行区 POI.shp")
print("浦东新区 poi")
intersect_shp_shp("E:\派 森\期 末\T3\shanghai_wgs84_浦 东 新 区 .shp","E:\派 森\期 末 \T3\poi.shp","E:\派森\期末\T3\结果\浦东新区 POI.shp")
print("普陀区 poi")
intersect_shp_shp("E:\派 森 \期 末 \T3\shanghai_wgs84_普 陀 区 .shp","E:\派 森 \期 末 \T3\poi.shp","E:\派森\期末\T3\结果\普陀区 POI.shp")
```

print("青浦区 poi")

intersect_shp_shp("E:\派 森 \期 末 \T3\shanghai_wgs84_青 浦 区 .shp","E:\派 森 \期 末 \T3\poi.shp","E:\派森\期末\T3\结果\青浦区 POI.shp")

print("松江区 poi")

intersect_shp_shp("E:\派 森 \期 末 \T3\shanghai_wgs84_松 江 区 .shp","E:\派 森 \期 末 \T3\poi.shp","E:\派森\期末\T3\结果\松江区 POI.shp")

print("徐汇区 poi")

intersect_shp_shp("E:\派 森 \期 末 \T3\shanghai_wgs84_徐 汇 区 .shp","E:\派 森 \期 末 \T3\poi.shp","E:\派森\期末\T3\结果\徐汇区 POI.shp")

print("闸北区 poi")

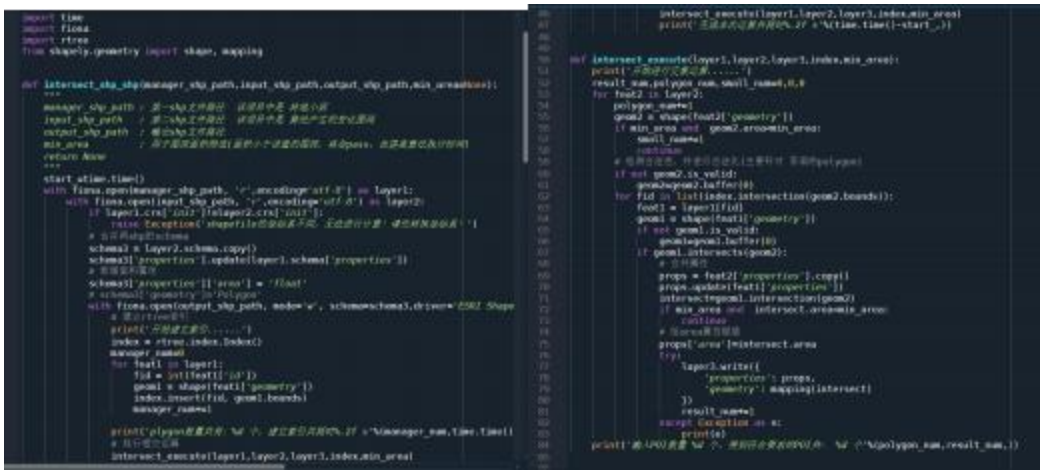
intersect_shp_shp("E:\派 森 \期 末 \T3\shanghai_wgs84_闸 北 区 .shp","E:\派 森 \期 末 \T3\poi.shp","E:\派森\期末\T3\结果\闸北区 POI.shp")

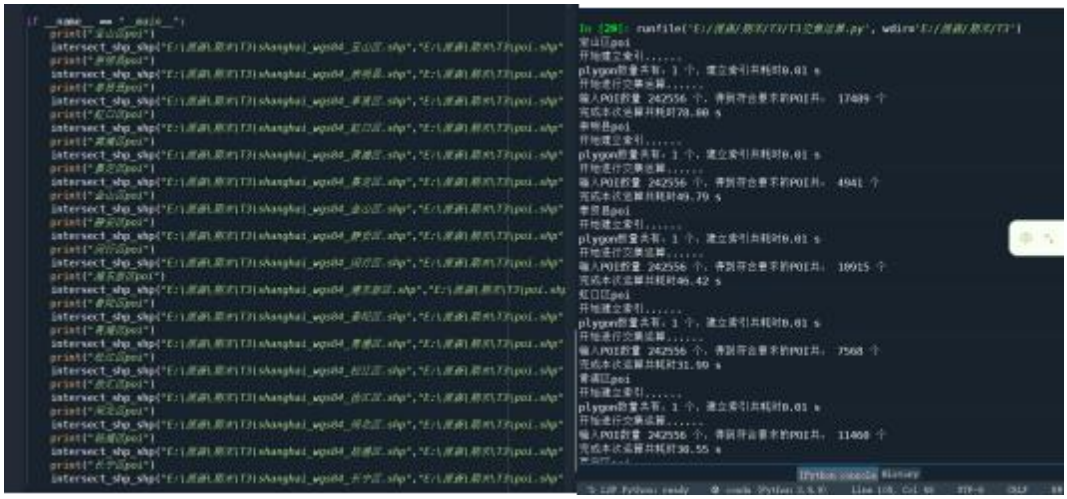
print("杨浦区 poi")

intersect_shp_shp("E:\派 森 \期 末 \T3\shanghai_wgs84_杨 浦 区 .shp","E:\派 森 \期 末 \T3\poi.shp","E:\派森\期末\T3\结果\杨浦区 POI.shp")

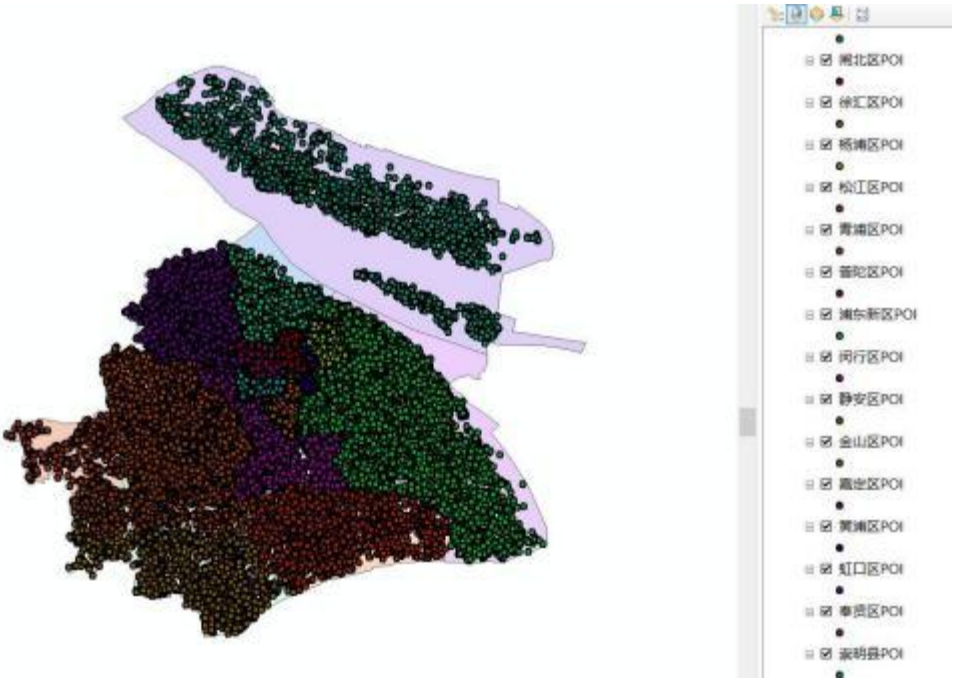
print("长宁区 poi")

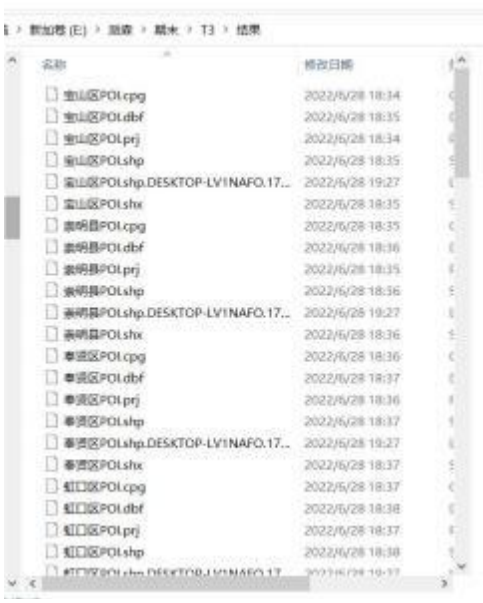
intersect_shp_shp("E:\派 森 \期 末 \T3\shanghai_wgs84_长 宁 区 .shp","E:\派 森 \期 末 \T3\poi.shp","E:\派森\期末\T3\结果\长宁区 POI.shp")





结果


闵北区POI
徐汇区POI
杨浦区POI
松江区POI
青浦区POI
普陀区POI
浦东新区POI
闵行区POI
静安区POI
金山区POI
嘉定区POI
黄浦区POI
虹口区POI
奉贤区POI
崇明县POI

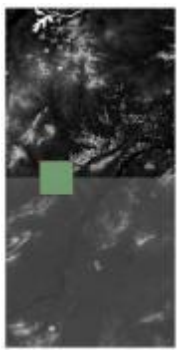


题目 4（30 分）

根据给定的两幅 DEM 数据，通过 Python 代码从中获取一幅新的 DEM 数据，新 DEM 数据的范围是：左下角(118.2, 31.9) 右上角(118.4, 32. 1) 。针对新 DEM 数据，编写算法提取高程数据中所有的局部最高点，所谓局部最高点就是高程大于周围所有栅格的栅格所在的位置。使用 matplotlib 显示 DEM 数据，并在局部最高点放置旗子图片（可用 flag-x.png)。

【相关要点：读取两幅 DEM 数据，通过新 DEM 数据的范围计算需要裁切的行列号区间，生成新的 DEM 数据注意 SetGeoTransform，SetProjection】

【局部最高点的绘制：不用考虑地理坐标，直接按照栅格坐标处理；可使用课上讲的坡度、坡向图等作为底图。】



思路：①先将两幅 DEM 图进行拼接，②然后将所需区域写成 shp 文件，添加投影信息③拼接好的 mosic.tif 文件按 shp 文件进行裁剪④给新的 dem 文件添加投影⑤求所得的 new_dem 的 slope 和 aspect⑥在山顶点加小红旗

代码 1：拼接代码

```
#mosica 两张图像
import os, sys, gdal
from gdalconst import *
os.chdir("E:/派森/期末/2022 期末试题-含 DEM 数据")#改变文件夹路径
# 注册 gdal(required)
gdal.AllRegister()
# 读入第一幅图像
ds1 = gdal.Open("E:/派森/期末/2022 期末试题-含 DEM 数据/ASTGTM2_ N31E118_dem.tif")
band 1 = ds1.GetRasterBand(1)
rows1 = ds1.RasterYSize
cols1 = ds1.RasterXSize

# 获取图像角点坐标
transform1 = ds1.GetGeoTransform()
minX1 = transform1[0]
maxY1 = transform1[3]
pixelWidth1 = transform1[ 1]
pixelHeight1 = transform1[5]#是负值 （important）
maxX1 = minX1 + (cols1 * pixelWidth1)
minY1 = maxY1 + (rows1 * pixelHeight1)

# 读入第二幅图像
ds2 = gdal.Open("E:/派森/期末/2022 期末试题-含 DEM 数据/ASTGTM2_ N32E118_dem.tif")
band2 = ds2.GetRasterBand(1)
rows2 = ds2.RasterYSize
cols2 = ds2.RasterXSize

# 获取图像角点坐标
transform2 = ds2.GetGeoTransform()
minX2 = transform2[0]
```

```
maxY2 = transform2[3]
pixelWidth2 = transform2[ 1]
pixelHeight2 = transform2[5]
maxX2 = minX2 + (cols2 * pixelWidth2)
minY2 = maxY2 + (rows2 * pixelHeight2)

# 获取输出图像坐标
minX = min(minX1, minX2)
maxX = max(maxX1, maxX2)
minY = min(minY1, minY2)
maxY = max(maxY1, maxY2)

#获取输出图像的行与列
cols = int((maxX - minX) / pixelWidth1)
rows = int((maxY - minY) / abs(pixelHeight1))

# 计算图 1 左上角的偏移值（在输出图像中）
xOffset1 = int((minX1 - minX) / pixelWidth1)
yOffset1 = int((maxY1 - maxY) / pixelHeight1)

# 计算图 2 左上角的偏移值（在输出图像中）
xOffset2 = int((minX2 - minX) / pixelWidth1)
yOffset2 = int((maxY2 - maxY) / pixelHeight1)

# 创建一个输出图像
driver = ds1.GetDriver()
dsOut = driver.Create('E:\ 派森\期末\T4\mosiac.tiff', cols, rows, 1, band 1.DataType)#1 是
bands，默认
bandOut = dsOut.GetRasterBand(1)
```

```
# 读图 1 的数据并将其写到输出图像中
data1 = band 1.ReadAsArray(0, 0, cols1, rows1)
bandOut.WriteArray(data1, xOffset1, yOffset1)

#读图 2 的数据并将其写到输出图像中
data2 = band2.ReadAsArray(0, 0, cols2, rows2)
bandOut.WriteArray(data2, xOffset2, yOffset2)
''' 写图像步骤'''
# 统计数据
bandOut.FlushCache()#刷新磁盘
stats = bandOut.GetStatistics(0, 1)#第一个参数是 1 的话，是基于金字塔统计，第二个
#第二个参数是 1 的话：整幅图像重度，不需要统计
# 设置输出图像的几何信息和投影信息
geotransform = [minX, pixelWidth1, 0, maxY, 0, pixelHeight1]
dsOut. SetGeoTransform( geotransform)
dsOut.Set Projection(ds1.GetProjection())

# 建立输出图像的金字塔
gdal.SetConfigOption('HFA_USE_RRD', 'YES')
dsOut.BuildOverviews(overviewlist= [2,4,8, 16])#4 层
```

代码 2：创建所需区域的 shp 文件

```
#生成有关裁剪区域的点图层 shp 文件，然后运用掩膜裁剪获得所需区域
import shapefile
#from osgeo import osr
outshp = r'E:\派森\期末\T4\所需区域.shp'
```

w = shapefile.Writer(outshp) # 注意，这里的参数不可以是 shapeType=5 ，必须是文件路径，否则会报错

#设置字段，最大长度为 254 ，C 为字符串

w.field('FIRST_FLD')

w.field('SECOND_FLD','C','40')

#添加几何和添加字段信息，添加两个示例，字段顺序区分先后

with open(r'E:\派森\期末\T4\新建文本文档.txt')as f:

arr = []

for line in f:

line = line.strip()

line = line.split(',')

第一列，第二列作为经纬度（x ，y）创建点

arr.append([float(line[0]), float(line[1])])

w.poly([arr])

w.record('First','Point')

w.poly([[[123,37], [118,36], [116,32],[119,20], [124,24],[123,37]]])

w.record('Second','Point')

#保存

w.close()

设置投影，通过.prj 文件设置，需要写入一个 wkt 字符串

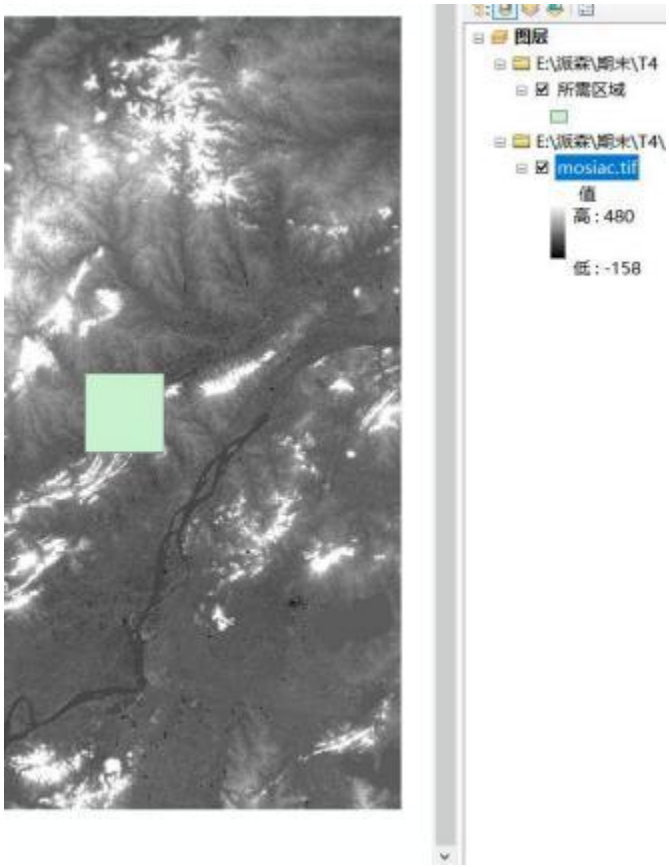
##gdal 的 GetProjection()返回的是 wkt 字符串，需要 ImportFromWkt

#projstr="""PROJCS["WGS_ 1984_UTM_zone_50N",GEOGCS["WGS 84",DATUM["WGS_ 1984",SPHEROID["WGS 84",6378137,298.257223563,AUTHORITY["EPSG","7030"]],AUTHORITY["EPSG","6326"]],PRIMEM["Greenwich",0],UNIT["degree",0.0174532925199433],AUTHORITY["EPSG","4326"]],PROJECTION["Transverse_Mercator"],PARAMETER["latitude_of_origin",0],PARAMETER["central_meridian", 117],PARAMETER["scale_factor",0.9996],PARAMETER["false_easting",500000],PARAMETER["false_northing",

0],UNIT["metre", 1,AUTHORITY["EPSG","9001"]],AUTHORITY["EPSG","32650"]]"""

```
proj = osr.SpatialReference()
proj.ImportFromEPSG(4326)
#或  proj.ImportFromProj4(proj4str)等其他的来源
wkt = proj.ExportToWkt()
#写出  prj 文件
f = open(outshp.replace(".shp",".prj"), 'w')
f.write(wkt)
f.close()
```

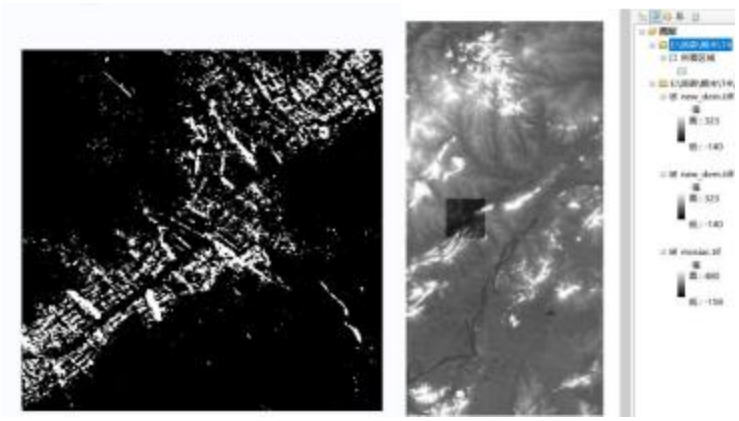
结果;



代码 3：按所给的 shp 裁剪

```
from osgeo import gdal
import os
import shapefile
#要裁剪的原图
```

```
input_raster = r"E:\派森\期末\T4\mosiac.tif"
input_raster=gdal.Open(input_raster)

#shp 文件所在的文件夹
path=r'E:/派森/期末/T4/'

#裁剪结果保存的文件夹
savepath=r"E:/派森/期末/T4/new_dem"

#读取  shp 文件所在的文件夹
files= os.listdir(path)

for f in files: #  循环读取路径下的文件并筛选输出
    if os.path.splitext(f)[1] == '.shp':
        name=os.path.splitext(f)[0]
        input_shape=path+f
        r = shapefile.Reader(input_shape)
        output_raster=savepath+'.tif'
        ds=gdal.Warp(output_raster,
                 input_raster,
                 format = 'GTiff',
                outputBounds=r.bbox,
                cutlineDSName = input_shape,
                cutlineWhere="FIELD = 'whatever'",
                dstNodata = - 1000)
ds=None
```



代码 4：给图像添加投影

```
#给图像设置投影
# 1.获取原数据信息
#  该数据只有地理坐标  WGS84
ds1 = gdal.Open("E:\派森\期末\T4\new_dem.tif")
im_geotrans1 = ds1.GetGeoTransform()      #仿射矩阵信息
im_proj 1 = ds1.GetProjection()              #地图投影信息
# print(im_geotrans)
# print(im_proj)
im_width1 = ds1.RasterXSize    #  栅格矩阵的列数
im_height1 = ds1.RasterYSize    #  栅格矩阵的行数
im_bands1 = ds1.RasterCount
ds_array1 = ds1.ReadAsArray(0, 0, im_width1, im_height1)    #  获取原数据信息，包括数据类型  int16 ，维度，数组等信息


# #  设置数据类型(原图像有负值)
datatype1 = gdal.GDT_Float32
# 2.原图像的仿射变换矩阵参数，即  im_geotransfor,m()
img_transf1 = (117.99986111111112,
 0.0002777777777777778,
 0.0,
 33.00013888888889,
 0.0,
```

```
 -0.0002777777777777778)
img_proj 1 = '''PROJCS["WGS 84 / UTM zone 50N",
    GEOGCS["WGS 84",
        DATUM["WGS_1984",
            SPHEROID["WGS 84",6378137,298.257223563,
                AUTHORITY["EPSG","7030"]],
            AUTHORITY["EPSG","6326"]],
        PRIMEM["Greenwich",0,
            AUTHORITY["EPSG","8901"]],
        UNIT["degree",0.01745329251994328,
            AUTHORITY["EPSG","9122"]],
        AUTHORITY["EPSG","4326"]],
    UNIT["metre", 1,
        AUTHORITY["EPSG","9001"]],
    PROJECTION["Transverse_Mercator"],
    PARAMETER["latitude_of_origin",0],
    PARAMETER["central_meridian", 117],
    PARAMETER["scale_factor",0.9996],
    PARAMETER["false_easting",500000],
    PARAMETER["false_northing",0],
    AUTHORITY["EPSG","32650"],
    AXIS["Easting", EAST],
    AXIS["Northing", NORTH]]'''

# 3.设置新文件及各项参数
filename = "E:\派森\期末\T4\new_dem1.tif"
driver = gdal.GetDriverByName("GTiff")  # 创建文件驱动
dataset = driver.Create(filename, im_width1, im_height1, im_bands1, datatype1)
dataset.SetGeoTransform(img_transf)  # 写入仿射变换参数
dataset.SetProjection(img_proj)  # 写入投影
```

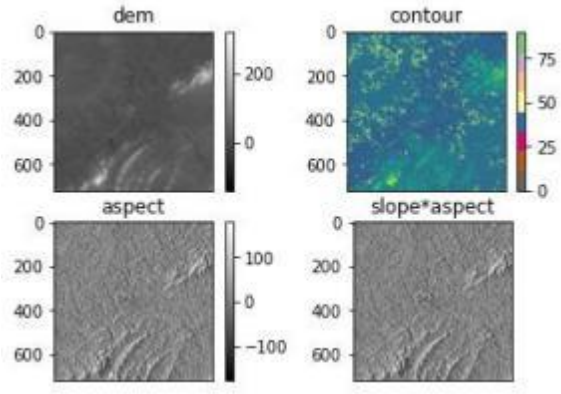
```
# 写入影像数据
dataset.GetRasterBand(1).WriteArray(ds_array)
del dataset
```

代码 5：求坡度和坡向

```
import gdal
import matplotlib.pyplot as plt
import numpy as np
def slope_x_y(grid,xcellsize,ycellsize):
    ycount=grid.shape[0]
    xcount=grid.shape[ 1]
    dx=1.0/(xcellsize*2)
    dy=1.0/(ycellsize*2)

    slopex=np.zeros(grid.shape,dtype=np.float64)
    slopey=np.zeros(grid.shape,dtype=np.float64)
    for iy in range(1,ycount- 1):
        for ix in range(1,xcount- 1):
            slopex[iy,ix]=(grid[iy,ix+1]-grid[iy,ix-1])*dx
            slopey[iy,ix]=(grid[iy+ 1,ix]-grid[iy- 1,ix])*dy
    slopex[ :,0]=slopex[ :, 1]
    slopex[ :,- 1]=slopex[ :,-2]
    slopey[0, :] = slopey[ 1, :]
    slopey[- 1, :]=slopey[-2, :]
    return slopex, slopey


demDS=gdal.Open("E:\派森\期末\T4\clip1.tifa.tif")
gt=( 118.2, 0.00027777777777778173, 0.0, 32.1, 0.0, -0.00027777777777778173)
xcellsize=gt[1]
ycellsize=gt[5]
```

```
grid=demDS.ReadAsArray(0,0).astype(np.float64)
slopex, slopey=slope_x_y(grid,xcellsize,ycellsize)
slope=np.sqrt(slopex*slopex+slopey*slopey)
slope=np.arctan(slope)*180/np.pi
aspect=np.arctan2(slopex,slopey)*180/np.pi


plt.subplot(221)
plt.imshow(grid,cmap="gray")
plt.title('dem')
plt.xticks([])


plt.colorbar()

plt.subplot(222)
cs=plt.contour(grid,20)
plt.clabel(cs)
plt.imshow(slope,cmap='Accent_r')
plt.title('contour')
plt.xticks([])
plt.colorbar()

plt.subplot(223)
plt.imshow(aspect,cmap='gray')
plt.title('aspect')
plt.xticks([])
plt.colorbar()
plt.subplot(224)
plt.imshow(slope*aspect,cmap='gray')
```

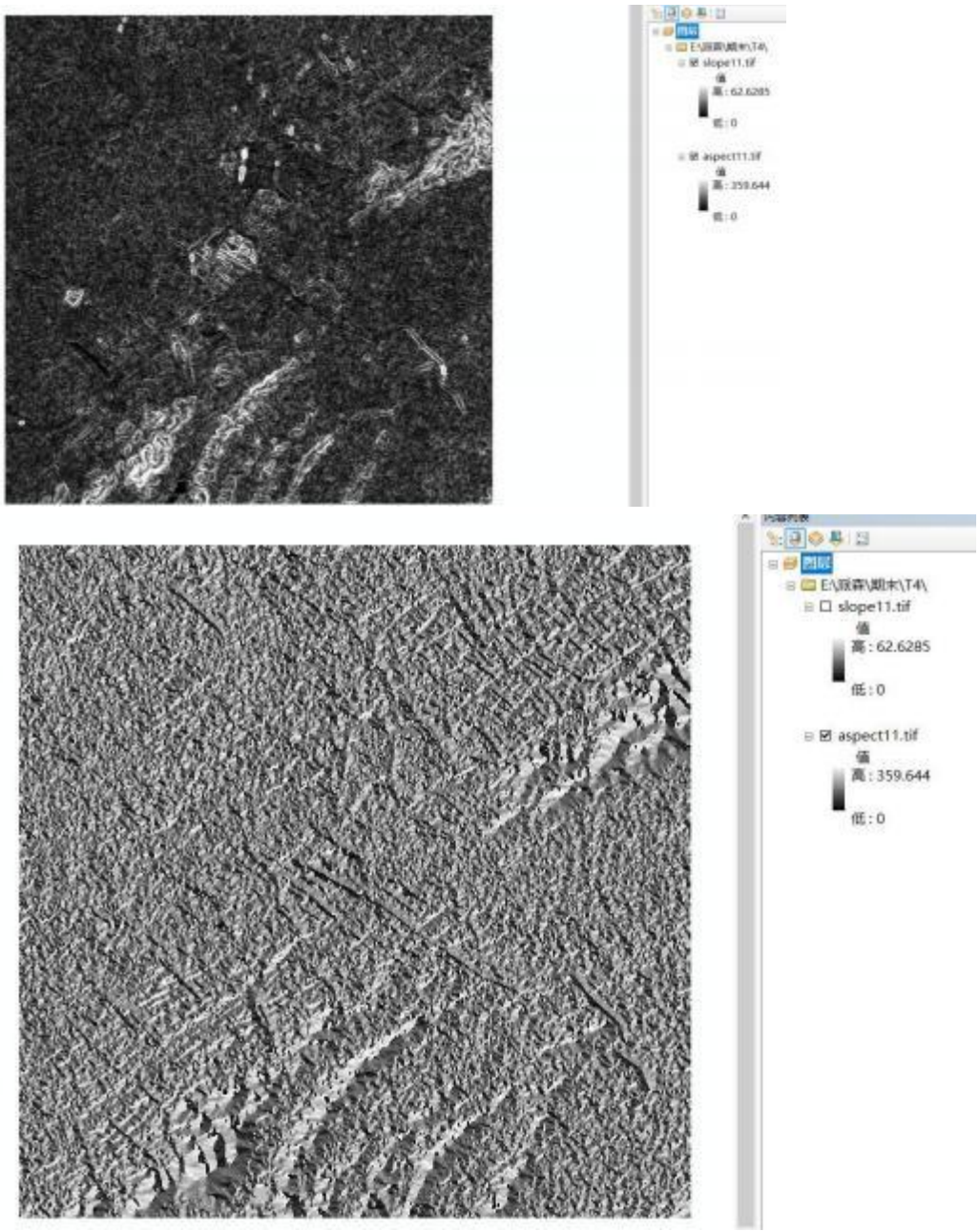
```
plt.title('slope*aspect')
plt.xticks([])
```




```
import DEMslopeAspect as dem
from DEMslopeAspect import Drawgrid
import datetime
# 程序入口
if __name__ == "__main__":
    startime = datetime.datetime.now() # 程序开始时间
    # 读取 ASTER GDEM 遥感影像
    demgrid, proj, geotrans, row, column, band, type =dem.read_img(r"E:\派 森 \期 末 \T4\new_dem.tif")
    # geotrans = (114.79763889, 0.00027777777778, 0.0, 38.21347222, 0.0, -0.00027777777778)
    # row = 13777
    # col = 28449

    demgridata = demgrid
    # 为计算梯度给影像添加周围一圈数据
    demgrid = dem.AddRound(demgrid)
    # 梯度计算
    dx1,dy1,dx2,dy2 = dem.Cacdxdy(demgrid,30,30)
    # 坡度、坡向计算
```

```
slope,aspect =dem.CacSlopAsp(dx1,dy1,dx2,dy2)
# 设置要投影的投影信息，此处是 WGS84- UTM-50N
tar_proj = '''PROJCS["WGS 84 / UTM zone 50N",
    GEOGCS["WGS 84",
        DATUM["WGS_1984",
            SPHEROID["WGS 84",6378137,298.257223563,
                AUTHORITY["EPSG","7030"]],
            AUTHORITY["EPSG","6326"]],
        PRIMEM["Greenwich",0,
            AUTHORITY["EPSG","8901"]],
        UNIT["degree",0.01745329251994328,
            AUTHORITY["EPSG","9122"]],
        AUTHORITY["EPSG","4326"]],
    UNIT["metre", 1,
        AUTHORITY["EPSG","9001"]],
    PROJECTION["Transverse_Mercator"],
    PARAMETER["latitude_of_origin",0],
    PARAMETER["central_meridian", 117],
    PARAMETER["scale_factor",0.9996],
    PARAMETER["false_easting",500000],
    PARAMETER["false_northing",0],
    AUTHORITY["EPSG","32650"],
    AXIS["Easting", EAST],
    AXIS["Northing", NORTH]]'''
# 输出TIFF 格式遥感影像,并设置投影坐标
slopeT = dem.write_img(r"E:\派森\期末\T4\slope1.tif", tar_proj, geotrans, slope, type)
aspectT = dem.write_img(r"E:\派森\期末\T4\aspect1.tif", tar_proj, geotrans, aspect,
type)
endtime = datetime.datetime.now()  # 程序结束时间
runtime = endtime - startime  # 程序运行时间
```

```
    print('运行时间为：    %d  秒' % (runtime.seconds))
```



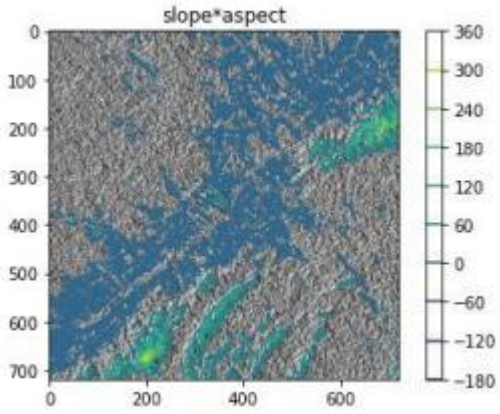
代码 6：在山顶插旗子

先生成底图

```
import gdal
import matplotlib.pyplot as plt
import numpy as np
def slope_x_y(grid,xcellsize,ycellsize):
    ycount=grid.shape[0]
```

```
    xcount=grid.shape[ 1]
    dx=1.0/(xcellsize*2)
    dy=1.0/(ycellsize*2)

    slopex=np.zeros(grid.shape,dtype=np.float64)
    slopey=np.zeros(grid.shape,dtype=np.float64)
    for iy in range(1,ycount- 1):
        for ix in range(1,xcount- 1):
            slopex[iy,ix]=(grid[iy,ix+1]-grid[iy,ix-1])*dx
            slopey[iy,ix]=(grid[iy+ 1,ix]-grid[iy- 1,ix])*dy
    slopex[ :,0]=slopex[ :, 1]
    slopex[ :,- 1]=slopex[ :,-2]
    slopey[0, :] = slopey[ 1, :]
    slopey[- 1, :]=slopey[-2, :]
    return  slopex, slopey


demDS=gdal.Open("E:\派森\期末\T4\clip1.tifa.tif")
gt=( 118.2, 0.00027777777777778173, 0.0, 32.1, 0.0, -0.00027777777777778173)
xcellsize=gt[1]
ycellsize=gt[5]
grid=demDS.ReadAsArray(0,0).astype(np.float64)
slopex, slopey=slope_x_y(grid,xcellsize,ycellsize)
slope=np.sqrt(slopex*slopex+slopey*slopey)
slope=np.arctan(slope)*180/np.pi
aspect=np.arctan2(slopex,slopey)*180/np.pi

ax,fig=plt.subplots()

plt.title('slope*aspect')
```

```
#plt.xticks([])
plt.contour(grid)
plt.colorbar()


plt.imshow(slope*aspect,cmap='gray')
plt.savefig("E:\派森\期末\T4/aspect.tif")
```

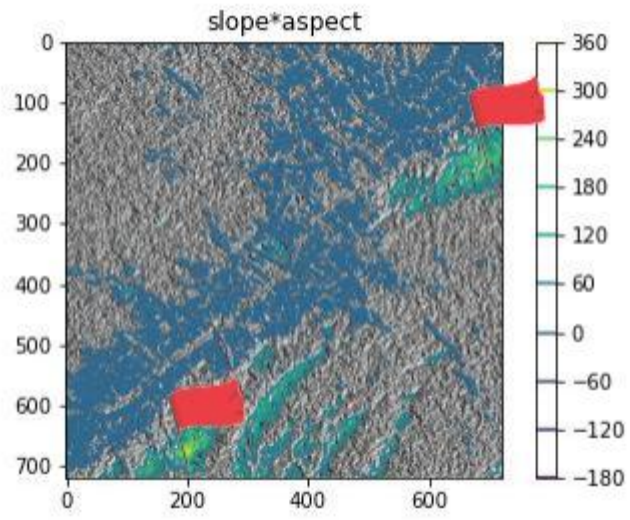


然后以添加水印的方式添加两遍红旗

```
import os
from PIL import Image, ImageFile
ImageFile.LOAD_TRUNCATED_IMAGES = True # tolerate large image file

FIT_WIDTH = 700
LOG_WIDTH = 200
LOGO_FILENAME = "E:/派森/期末/T4/flag.png" # in current working directory
GIVEN_DIR ="E:\派森\期末\T4\withLogo1/"
logoIm = Image.open( LOGO_FILENAME)
logoWidth, logoHeight = (50,50) # (808,768)
logoHeight, logoWidth = (50,50)
logoIm = logoIm.resize((logoWidth, logoHeight))

os.makedirs('withLogo', exist_ok = True) # create a directory to save changed images
```

```
for filename in os.listdir(GIVEN_DIR):
    if not (filename.endswith('.png') or filename.endswith('.jpg')):
        continue # skip non-image files
    im = Image.open(os.path.join(GIVEN_ DIR, filename))
    width, height = im.size
    if width > FIT_WIDTH:
        height = int((FIT_WIDTH / width) * height)
        width = FIT_WIDTH
    print('Resizing {}...'.format(filename))
    im = im.resize((width, height)) # resize the image
    print('Adding logo to {}...'.format(filename))
    im.paste(logoIm, (300, 50), logoIm) # add the logo
    im.save(os.path.join('withLogo', filename)) # save changes
```

结果



题目 5（30 分）

根据提供的 simple_typhoon 脚本，获取 2021 年西太平洋的台风数据。任选一条或多条台风记录，了解台风记录中提供的信息(每个轨迹点上的时间、位置、速度、强度、压强等信息)，利用 Cartopy 、Matplotlib 进行制图。

【相关要点：不仅仅是在地图上画出轨迹，可以查找其他数据作为背景，可以在轨迹上标注不同的颜色，可以在台风级别变化的点上标注台风级别（热带风暴、强热带风暴等）】

【按照台风轨迹的时间顺序，对其属性（速度、强度、压强等）进行统计制图，有能力的同学可以制作动图】

【可以选择多条台风路径，制作 2021 年登陆中国大陆的台风专题图，充分设计柱状图、饼状图、折线图等，实现好看的效果】

【本题为开放性试题，**制图的美观程度是主要评分标准**】

思路：①爬取有关台风的信息②将有关字段放进 csv 表格中③加载底图④为不同的强度设置不同的颜色⑤单台风路径的图⑥gif 图

代码 1：爬取有关台风信息，代码由老师提供

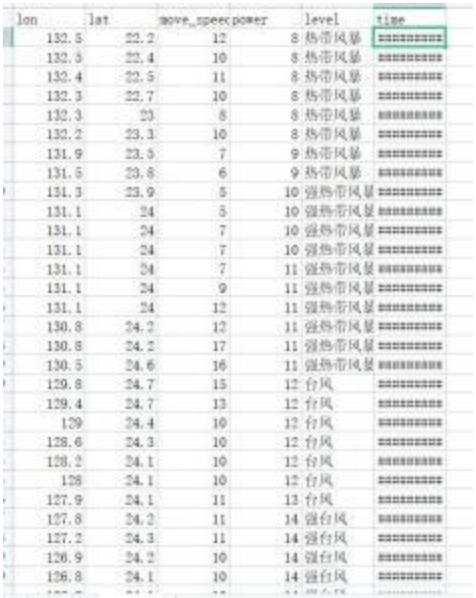
代码 2：将有关字段放进 csv 中

```
url='https://typhoon.slt.zj.gov.cn/Api/TyphoonList/2021?callback=jQuery183076694718233
66587_ 1654483413702&_=1654483442956'
list_info = request.urlopen(url).read().decode('utf-8')

# 处理网络请求得到的字符串
list_info = list_info[list_info.find(' ['):]
list_info = list_info[ :-2]

# 按照 JSON 进行解析
obj_list = json.loads(list_info)

# 指定一个台风对象，再次请求网络数据
idx = 5
instance_id = obj_list[idx]['tfid']
url_inst='https://typhoon.slt.zj.gov.cn/Api/TyphoonInfo/'+instance_id+'?callback=jQuery183
07669471823366587_ 1654483413702&_=1654483464147'
instance_info = request.urlopen(url_inst). read().decode('utf-8')
```

```
        # 对单个台风的请求结果进行字符串处理
        instance_info = instance_info [instance_info.find(' [']:]
        instance_info = instance_info [:-2]

        # 获取单个台风的 JSON 对象
        obj_inst = json.loads(instance_info)

        # 获取台风的路径
        point_count = len(o bj_inst[0]['points'])
        x = []
        y = []
        move_speed= []
        power=[]
        level= []
        data=[]
        time=[]
        data.append(['lon','lat','move_speed','power','level','time'])
        for iP in range(0, point_count):
            #这边只取了位置 lng,lat 信息，json 中还有其他信息，如 pressure ， power ， strong，
speed 等
            temp_lng = obj_inst[0]['points'][iP]['lng']
            temp_lat = obj_inst[0]['points'][iP]['lat']
            temp_speed=obj_inst[0]['points'][iP]['movespeed']
            temp_power=obj_inst[0]['points'][iP]['power']
            temp_strong=obj_inst[0]['points'][iP]['strong']
            temp_time=obj_inst[0]['points'][iP]['time']
            x.append(float(temp_lng))
            y.append(float(temp_lat))
            move_speed.append(temp_speed)
```

```
            power.append(temp_power)
            level.append(temp_strong)
            time.append(temp_time)
        data.append([float(temp_lng),float(temp_lat),temp_speed,temp_power,temp_strong,temp_
time])
            print(float(temp_lng),float(temp_lat))
        #temp_speed=obj_inst[0]['points'][iP]['movespeed']
        #temp_power=obj_inst[0]['points'][iP]['power']
        #temp_strong=obj_inst[0]['points'][iP]['strong']

        with open('E:\派森\期末\data1.csv', 'w',newline='') as csvfile:
            writer    = csv.writer(csvfile)
            for row in data:
                writer.writerow(row)
            csvfile.close()
```

结果



lon	lat	move_speed	power	level	time
132.5	22.2	12	8	热带风暴	=========
132.5	22.4	10	8	热带风暴	=========
132.4	22.5	11	8	热带风暴	=========
132.3	22.7	10	8	热带风暴	=========
132.3	23	8	8	热带风暴	=========
132.2	23.3	10	8	热带风暴	=========
131.9	23.5	7	9	热带风暴	=========
131.5	23.8	6	9	热带风暴	=========
131.3	23.9	5	10	强热带风暴	=========
131.1	24	5	10	强热带风暴	=========
131.1	24	7	10	强热带风暴	=========
131.1	24	7	10	强热带风暴	=========
131.1	24	7	11	强热带风暴	=========
131.1	24	9	11	强热带风暴	=========
131.1	24	12	11	强热带风暴	=========
130.8	24.2	12	11	强热带风暴	=========
130.8	24.2	17	11	强热带风暴	=========
130.5	24.6	16	11	强热带风暴	=========
129.8	24.7	15	12	台风	=========
129.4	24.7	13	12	台风	=========
129	24.4	10	12	台风	=========
128.6	24.3	10	12	台风	=========
128.2	24.1	10	12	台风	=========
128	24.1	10	12	台风	=========
127.9	24.1	11	13	台风	=========
127.8	24.2	11	14	强台风	=========
127.2	24.3	11	14	强台风	=========
126.9	24.2	10	14	强台风	=========
126.8	24.1	10	14	强台风	=========

代码 3：设置底图

```
import csv
import urllib.request as request
import json
```

```
# cartopy：用来获取地图
import cartopy.crs as ccrs
import cartopy.feature as cfeature
# matplotlib：用来绘制图表
import matplotlib.pyplot as plt
# shapely ：用来处理点线数据
import shapely.geometry as sgeom
import  warnings
#import re
import numpy as np
import pandas as pd
warnings.filterwarnings('ignore')
plt.rcParams['font.sans-serif'] = [u'SimHei']
plt.rcParams['axes.unicode_minus'] = False
# 通过 cartopy 获取底图
fig = plt.figure(figsize=( 10, 10))
ax = fig.add_subplot(1, 1, 1, projection=ccrs.PlateCarree())

# 用经纬度对地图区域进行截取，这里只展示我国沿海区域
ax.set_extent([85, 170,-20,60], crs=ccrs.PlateCarree())

# 设置名称
ax.set_title('2021 年台风路径图',fontsize=16)

# 设置地图属性，比如加载河流、海洋
ax.add_feature(cfeature.LAND)
ax.add_feature(cfeature.OCEAN)
ax.add_feature(cfeature.COASTLINE)
ax.add_feature(cfeature.RIVERS)
ax.add_feature(cfeature.COASTLINE)
```

```
ax.add_feature(cfeature.LAKES, alpha=0.5)


# 展示地图
plt.show()
```

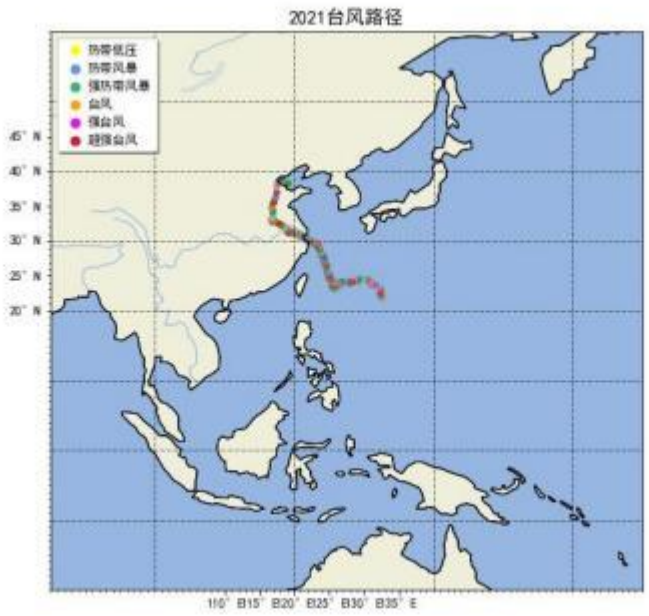
结果

代码 4：为不同的强度设置不同的颜色

```
def get_color(lever) :
    global color
    if level == '热带低压' or level == '热带扰动':
        color='#FFFF00'
    elif level == '热带风暴':
        color='#6495ED'
    elif level == '强热带风暴':
        color='#3CB371'
    elif level == '台风':
        color='#FFA500'
    elif level == '强台风':
        color='#FF00FF'
    elif level == '超强台风':
        color='#DC143C'
        return color
```

代码 5：单台风路径图

```
def draw_single(df):
    ax = create_map('2021 台风路径',extent= [ 110, 135, 20, 45])
    for i in range(len(df)):
        ax.scatter(list(df['lon'])[i], list(df['lat'])[i], marker='o', s=20,
color=get_color(list(df['level'])[i]))

    for i in range(len(df)- 1) :
        pointA = list(df['lon'])[i],list(df['lat'])[i]
        pointB = list(df['lon'])[i+1],list(df['lat'])[i+1]
        ax.add_geometries([sgeom.LineString([pointA, pointB])],
color=get_color(list(df['level'])[i+1]),crs=ccrs.PlateCarree())
    plt.savefig('./typhoon_one.png')
```



代码 6：gif 图，根据 202 个时间点数据生成 202 张轨迹图，然后拼接成 gif

```
def draw_single_gif(df):
    for state in range(len(df.index))[:] :
        ax = create_map(f'2021 台风路径 {df["time"].iloc[state]}', [ 110, 135, 20, 45])
        for i in range(len(df[:state])):
```

```
                ax.plot(df['lon'].iloc[i], df['lat'].iloc[i], linestyle='-',
lw=1,marker='o',color=get_color(df['level'].iloc[i]))
            for i in range(len(df[:state])- 1) :
                pointA = df['lon'].iloc[i],df['lat'].iloc[i]
                pointB = df['lon'].iloc[i+1],df['lat'].iloc[i+1]
                ax.add_geometries([sgeom.LineString([pointA, pointB])],
color=get_color(df['level'].iloc[i+1]),crs=ccrs.PlateCarree())
            print(f'正在绘制第{state}张轨迹图')
            plt.savefig(f'E:\派森\期末\T5/{str(state).zfill(3)}.png', bbox_inches='tig ht')
        #import matplotlib. pyplot as plt
        import imageio,os
        images = []
        filenames=sorted((fn for fn in os.listdir('.') if fn.endswith('.png')))
        for filename in filenames:
            images.append(imageio.imread(filename))
            imageio.mimsave('E:\派森\期末\T5\gif.gif', images,duration=1)
        #将图片拼接成动画
        imgFiles = list(glo b.glob(f'台风.png'))
        images = [ Image.open(fn) for fn in imgFiles]
        im = images[0]
        filename = f'台风.gif'
        im.save(fp=filename, format='gif', save_all=True, append_images=images[ 1:],
duration=500)
draw_single_gif(df)
```

gif 图见文件夹

路径

```
import pandas as pd
import matplotlib.pyplot as plt
import cartopy.crs as ccrs
import cartopy.feature as cfeature
import numpy as np
#warnings.filterwarnings('ignore')
########################################  路  径
###################################
extent= [ 110, 135,20,45]

df = pd.read_csv("E:\派森\期末\T5\data1.csv")  # 读取训练数据
#print(data.shape)
x=df['lon'][:72]
y=df['lat'][:72]
x1=df['lon'][72:144]
y1=df['lat'][72:144]
x2=df['lon'][144:203]
y2=df['lat'][144:203]
fig = plt.figure(figsize=( 10, 10))
fig = plt.figure(figsize=( 12, 8))
ax =fig.subplot(111, projection=ccrs.PlateCarree())
   #    url = 'http://map1c.vis.earthdata.nasa.gov/wmts-geo/wmts.cgi'
   #    layer = 'BlueMarble_Shaded Relief'
    # ax.add_wmts(url, layer)
    # ax.set_extent(extent,crs=ccrs.PlateCarree())
    # 用经纬度对地图区域进行截取，这里只展示我国沿海区域
ax.set_extent([85, 170,-20,60], crs=ccrs.PlateCarree())
```

```
ax.plot(x,y,color='#900302',linestyle='-',label='2021/7/23 14:00:00 前')
ax.plot(x1,y1,label='2021/7/26 14:00:00 前')
ax.plot(x2,y2,label='2021/7/30 17:00:00 前')
legend=('2021/7/23 14:00:00 前','2021/7/26 14:00:00 前','2021/7/30 17:00:00 前')
ax.legend()
ax.set_title('2021 年某台风不同时刻路径图',fontsize=16)
gl = ax.gridlines(draw_labels=False, linewidth=1, color='k', alpha=0.5, linestyle='--')
gl.xlabels_top = gl.ylabels_right = False
# 设置地图属性，比如加载河流、海洋
ax.add_feature(cfeature.LAND)
ax.add_feature(cfeature.OCEAN)
ax.add_feature(cfeature.COASTLINE)
ax.add_feature(cfeature.RIVERS)
ax.add_feature(cfeature.COASTLINE)
ax.add_feature(cfeature.LAKES, alpha=0.5)
# 设置名称
      #ax.set_title('2021 年台风路径图',fontsize=16)


      #ax.add_geometries([buffer],ccrs.PlateCarree(),facecolor='# F9009F', edgecolor='none')
    #ax.add_geometries([line],ccrs.PlateCarree(),facecolor='none', edgecolor='#9F0000')
gl = ax.gridlines(draw_labels=False, linewidth=1, color='k', alpha=0.5, linestyle='-')
gl.xlabels_top = gl.ylabels_right = False
ax.set_xticks(np.arange(extent[0], extent[1]+5, 5))
ax.set_yticks(np.arange(extent[2], extent[3]+5, 5))
#ax.xaxis.set_major_formatter( LongitudeFormatter())
ax.xaxis.set_minor_locator(plt.MultipleLocator(1))
#ax.yaxis.set_major_formatter(LatitudeFormatter())
ax.yaxis.set_minor_locator(plt.MultipleLocator(1))
ax.tick_params(axis='both', labelsize=10, direction='out')
plt.rcParams['font.sans-serif'] = [u'SimHei']
```

```
plt.rcParams['axes.unicode_minus'] = False


ax.set_extent([85, 170,-20,60], crs=ccrs.PlateCarree())


plt.savefig('../台风不同时刻路径图')
```

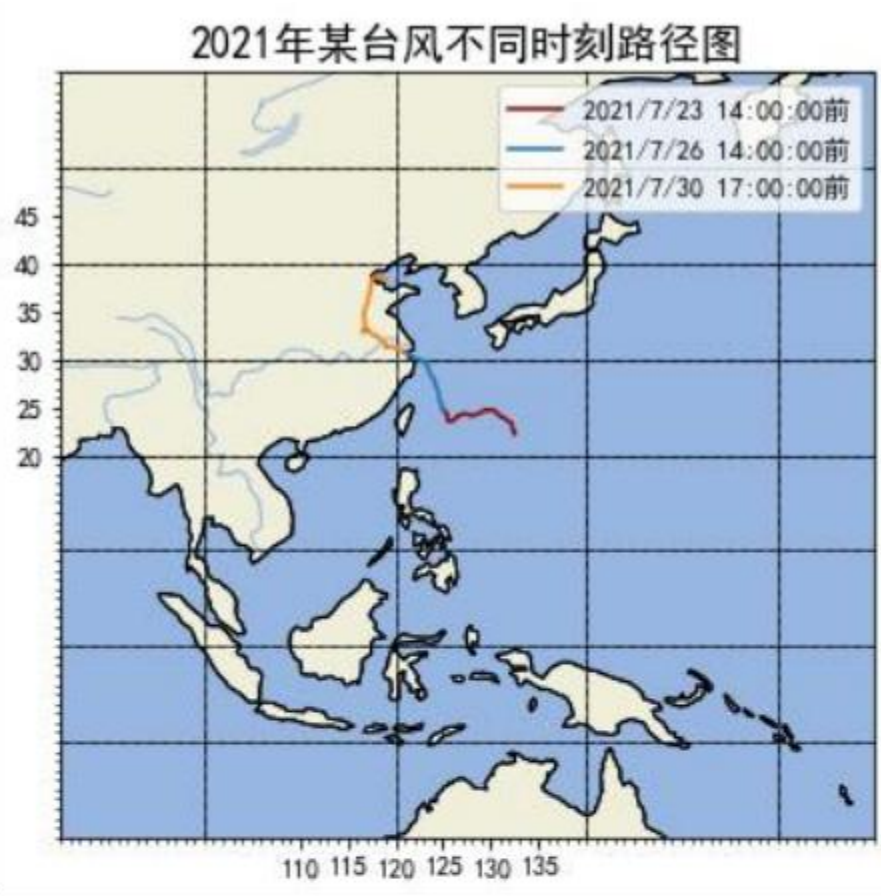
速度

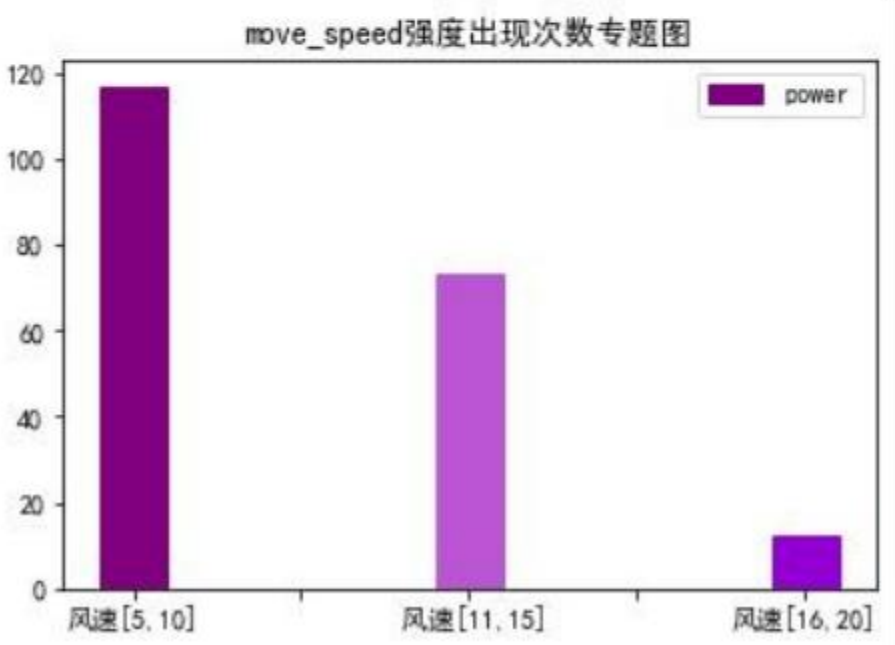
```
import pandas as pd

import matplotlib.pyplot as plt

import cartopy.crs as ccrs

import cartopy.feature as cfeature

import numpy as np


fig=plt.figure(figsize=(8,4),dpi=300)

df = pd.read_csv("E:\派森\期末\T5\data1.csv")   # 读取训练数据

speed 1=0

speed2=0

speed3=0

for item in range(0,len(df)- 1) :

            speed=df['move_speed'].iloc[item]

            if(4<speed<11) :

               speed 1+=1

            if(10<speed<16):

               speed2+=1

            if(15<speed<21):

               speed3+=1


ax = fig.add_subplot(224)

speed_map = {

      '风速[5, 10]': (speed 1, '#CC3366'),

      '风速[11, 15]': (speed2, '#CC0099'),
```

```
        '风速[16,20]': (speed3, '#CC0000')


    }
    bar_width = 0.2
    ax.set_title('move_speed 强度出现次数专题图')   #子图标题
    xticks = np.arange(3)
    power = [x [0] for x in speed_map.values()]
    #设置 x、y 轴的范围
    #ax.set_xlim([bar_width/2- 1, 3-bar_width/2])
    #ax.set_ylim([0, 125])
    bars = ax.bar(xticks,power, width=bar_width, edgecolor='none',label='power')    #设置柱的
边缘为透明


    colors = [x [ 1] for x in speed_map.values()]   #对应颜色
    for bar, color in zip(bars, colors):   #给每个 bar 分配指定的颜色
        bar.set_color(color)
    labels=( ' ','风速[5, 10]',' ',' 风速[11, 15]',' ','风速[16,20]')
    ax.set_xticklabels(labels)
    ax.legend()
    plt.show()
    plt.savefig("E:\派森\期末\T5\move_speed 强度出现次数专题图.png")
    强度
    import pandas as pd
    import matplotlib.pyplot as plt
    import cartopy.crs as ccrs
    import cartopy.feature as cfeature
    import numpy as np
    fig=plt.figure(figsize=(8,4))
```

```
df = pd.read_csv("E:\派森\期末\T5\data1.csv")   # 读取训练数据
peight=0
pnine=0
pten=0
peleven=0
ptwel=0
pthir=0
pfort=0
pseven=0
for item in range(0,len(df)- 1) :
        z=df['power'].iloc[item]
        if(z==8):
            peight+=1
        if(z==9):
            pnine=pnine+ 1
        if(z==10) :
            pten+=1
        if(z==11) :
            peleven+=1
        if(z==12) :
            ptwel+=1
        if(z==13) :
            pthir+=1
        if(z==14) :
            pfort+=1
        if(z==7):
            pseven+=1
ax = fig.add_subplot(111)
```

```
power_map = {
        '七级': (pseven, '#800080'),
        '八级': (peight, '#BA55D3'),
        '九级': (pnine, '#9400D3'),
        '十级':(pten,'#9932CC'),
        '十一级':(peleven,'#4B0082'),
        '十二级':(ptwel,'#8A2BE2'),
        '十三级':(pthir,'#9370D8'),
        '十四级':(pfort,'#7B68EE')

}
bar_width = 0.2
ax.set_title('power 时间段内出现次数专题图')   #子图标题
xticks = np.arange(8)
power = [x [0] for x in power_map.values()]
#设置 x、y 轴的范围
#ax.set_xlim([bar_width/2- 1, 3-bar_width/2])
#ax.set_ylim([0, 125])
bars = ax.bar(xticks,power, width=bar_width, edgecolor='none',label='power')    #设置柱的
边缘为透明


colors = [x [ 1] for x in power_map.values()]   #对应颜色
for bar, color in zip(bars, colors):   #给每个 bar 分配指定的颜色
        bar.set_color(color)
labels=( '','七级','八级','九级','十级','十一级','十二级','十三级','十四级')
ax.set_xticklabels(labels)
ax.legend()
plt.show()
```


2021年某台风不同时刻路径图
2021/7/23 14:00:00前
2021/7/26 14:00:00前
2021/7/30 17:00:00前
45
40
35
30
25
20
110 115 120 125 130 135


move_speed强度出现次数专题图
power
120
100
80
60
40
20
0
风速[5, 10]
风速[11, 15]
风速[16, 20]


power时间段内出现次数专题图
power
50
40
30
20
10
0
八级
九级
十级
十一级
十二级
十三级
十四级

gif 见文件夹