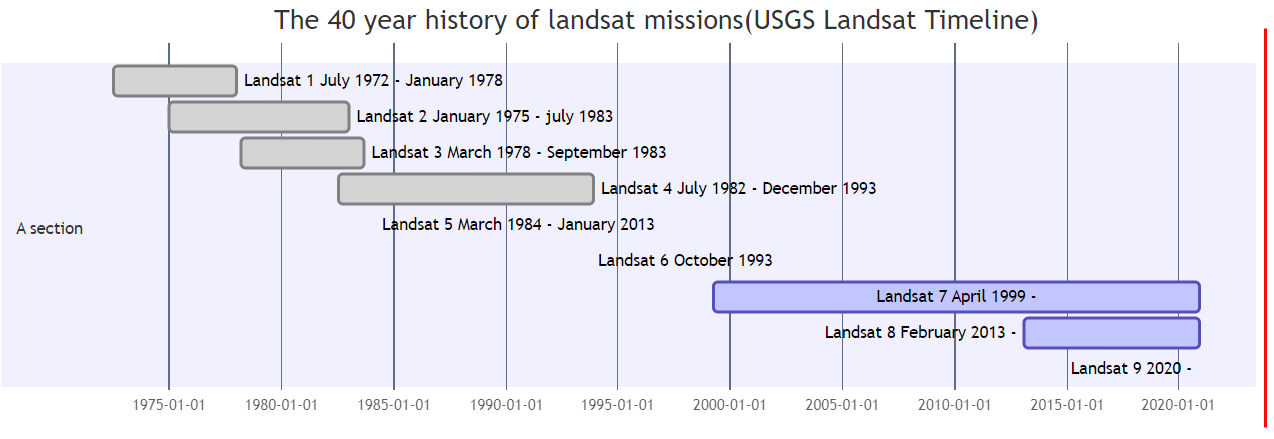
## 2.2.2 主成成分分析与Landsat遥感影像

### 2.2.2.1 Landsat地理信息数据读取、裁切、融合和打印显示

从1970s年代，地球资源卫星（The Landsat series of satellites）几乎不间断的提供全球范围地球表面中分辨率多光谱遥感影像数据。不间断的数据记录，增强了应用数据分析的潜力，可以帮助各领域的研究者在人口不断增长，城市不断发展下，关注粮食、水、森林等自然资源，应用其无以伦比的影像质量、覆盖率、细节价值分析城市的发展与自然平衡的问题（美国地质调查局 [United States Geological Survey，USGS-earthexplorer](https://earthexplorer.usgs.gov/)①）。

注，下述[Mermaid](https://mermaid-js.github.io/mermaid/#/)②图表代码，在JupyterLab下无法显示，但是在Markdown下通常显示。

Landsat陆地资源卫星的数据为多光谱，通过安装在环绕地球轨道的太空卫星上的平台收集。各个卫星传感器所获遥感影像数据的波段情况有所差异，下述为Landsat 8各波段的情况：

| 波段(band) | 波长（wavelength range）/nanometers | 空间分辨率（spatial resolution）/m | 光谱宽度（spetral width）/nm | 用途 |
| --- | --- | --- | --- | --- |
| Band 1- Coastal aerosol | 430 - 450 | 30 | 20 | 沿海和气溶胶研究 |
| Band 2- Blue 蓝色波段 | 450 - 510 | 30 | 60 | 水深绘图，从植被中区分土壤，针叶林植被中区分落叶 |
| Band 3- Green 绿色波段 | 530 - 590 | 30 | 60 | 强调植被生长高峰，用于评估植被活力 |
| Band 4- Red 红色波段 | 640 - 670 | 30 | 30 | 处于叶绿素吸收区，用于观测道路，裸露土壤，植被种类等 |
| Band 5- Near Infrared(NIR) 近红外波段 | 850 - 880 | 30 | 30 | 强调生物量和海岸线 |
| Band 6- Shortwave Infrared 1(SWIR1) 短波红外波段1 | 1570 - 1650 | 30 | 80 | 区分土壤和植被含水量，并能穿透薄云 |
| Band 7- Shortwave Infrared 2(SWIR2) 短波红外波段2 | 2110 - 2290 | 30 | 180 | 增加的波段，区分土壤和植被含水量，并能穿透薄云 |
| Band 8- Panchromatic 全色波段 | 500 - 680 | 15 | 180 | 15米分辨率，更清晰的图像清晰度 |
| Band 9- Cirrus clouds 卷云波段 | 1360 - 1380 | 30 | 20 | 改进了对卷云污染的检测 |
| Band 10- Thermal Infrared 1(TIRS\_1) 热红外波段1 | 10600 - 11190 | 100 | 590 | 100米分辨率，热红外图，可估计的土壤湿度 |
| Band 11- Thermal Infrared 2(TIRS\_2) 热红外波段2 | 11500 - 12510 | 100 | 1010 | 100米分辨率，增加的热红外图，可估计的土壤湿度 |

Landsat系列遥感影像在城市研究中作用举足轻重，在不同的研究领域中均需要使用该数据分析，例如从上个世纪70年代至今城市及自然环境的演化，用地类型的变化，反演地表温度研究城市热环境，绿地变化与生态影响，水资源情况等，不计其数。除了Landsate中分辨率卫星，还有高空分辨率低于1m的高分辨率遥感影像，探究城市的细微变化，尤其包含有近红外波段的高分辨率影像，可以在微尺度上深入植被研究，及拓展相关研究。

* Landsat文件名所包含的信息

Landsat文件名格式：LXSS\_LLLL\_PPPRRR\_YYYYMMDD\_yyyymmdd\_CC\_TX，例如：LC08\_L1TP\_023031\_20180310\_20180320\_01\_T1

| 序号 | 占位符 | 代表含义 | 备注 | 案例中的值 |
| --- | --- | --- | --- | --- |
| 1 | L | 卫星名 | Landsat | L |
| 2 | X | 传感器 | “C”代表OLI/TIRS组合，“O”代表OLI，“T”代表TIRS，“E”代表ETM+，“T”代表TM，“M”代表MSS | C |
| 3 | SS | 卫星编号 | 截至2021年，编号为1~9 | 08 |
| 4 | [LLLL](https://www.usgs.gov/land-resources/nli/landsat/landsat-levels-processing) ③ | 处理矫正水平（Processing correction level） ，由地面控制点、数字高程和传感器收集的数据确定 | “L1TP”（Precision Terrain）为1级精度地形矫正产品，如果参考数据不充足，则处理为系统的、地形矫正的L1GT（Systematic Terrain）和L1GS（Systematic）产品 | L1TP |
| 5 | PPP | WRS path 世界参考系统（the Worldwide Reference），是Landsat卫星数据的全球定位系统 | 可以指定path，row查询世界上任何部分的卫星图像 | 023 |
| 6 | RRR | WRS row | \ | 031 |
| 7 | YYYYMMDD | 获取年份 | \ | 20180310 |
| 8 | yyyymmdd | 处理年份 | \ | 20180320 |
| 9 | CC | 采集编号 | （01，02） | 01 |
| 10 | [TX](https://www.usgs.gov/land-resources/nli/landsat/landsat-collection-1?qt-science_support_page_related_con=1#qt-science_support_page_related_con) ④ | 采集类别 | “RT”为实时，“T1”为层1（Tier1）包含最高数据质量；“T2”为层2，因为缺失数据精度较T1低 | T1 |

对于遥感影像的处理，即为对raster（栅格数据）的处理，可以使用[rasterio](https://rasterio.readthedocs.io/en/latest/)⑤，[EarthPy](https://earthpy.readthedocs.io/en/latest/)⑥，[GDAL](https://gdal.org/)⑦等库读写，定义或重定义投影坐标系统，裁切与融合，数据处理及可视化等。

import earthpy.spatial as es  
import earthpy.plot as ep  
import earthpy.mask as em  
import matplotlib.pyplot as plt  
  
from glob import glob  
import os,re  
plt.rcParams.update({'font.size': 20})  
  
Landsat\_fp={  
 "w\_180310":r"F:\data\Landsat\LC08\_L1TP\_023031\_20180310\_20180320\_01\_T1", # 冬季  
 "s\_190820":r"F:\data\Landsat\LC08\_L1TP\_023031\_20190804\_20190820\_01\_T1", # 夏季  
 "a\_191018":r"F:\data\Landsat\LC08\_L1TP\_023031\_20191007\_20191018\_01\_T1" # 秋季   
 }  
w\_landsat=glob(os.path.join(Landsat\_fp["w\_180310"],"\*\_B[0-9]\*.tif"))  
  
band\_name={'B1':'coastal\_aerosol','B2':'blue','B3':'green','B4':'red','B5':'near\_infrared','B6':'SWIR\_1','B7':'SWIR\_2','B8':'panchromatic','B9':'cirrus','B10':'TIRS\_1','B11':'TIRS\_2'}  
  
def fp\_sort(fp\_list,str\_pattern,prefix=""):  
 '''  
 function - 按照文件名中的数字排序文件列表  
   
 Params:  
 fp\_list - 文件列表；list(string)  
 str\_pattern - 字符串匹配模式，用于提取文件名中的数字；re.compile()  
 prefix - 字典数据格式的键名前缀；string  
   
 Returns:  
 fn\_sort - 返回排序后的列表；list(string)  
 fn\_dict - 返回字典；dict  
 '''  
 fn\_num\_extraction=[(int(re.findall(str\_pattern, fn)[0]),fn) for fn in fp\_list]  
 fn\_sort=sorted(fn\_num\_extraction)  
 fn\_dict=dict([("%s"%prefix+str(i[0]),i[1]) for i in fn\_sort])  
 return fn\_sort,fn\_dict  
   
str\_pattern=re.compile(r'B(.\*?)[.]', re.S)   
fn\_sort,fn\_dict=fp\_sort(w\_landsat,str\_pattern,prefix="B")   
  
# 打印遥感影像波段  
array\_stack, meta\_data=es.stack([fn[1] for fn in fn\_sort][:7], nodata=-9999)  
ep.plot\_bands(array\_stack, title=list(band\_name.values())[:7],cols=7,cbar=True,figsize=(10\*7,10))  
plt.show()



上述通过给定Landsat文件夹，用glob库获取指定字符串匹配模式的所有文件路径，即Landsat各个波段的路径名，然后使用EarthPy库.stack叠合层为一个变量。注意只有空间分辨率相同的层可以叠合，因此叠合了前7个波段，并使用该库打印显示。下述代码则定义了LandsatMTL\_info(fp)函数，给定Landsat所在根目录，读取元文件，即\_MTL.txt文件，通过读取元文件获取相关影像信息，并获取各个波段的路径。

def LandsatMTL\_info(fp):  
 '''  
 function - 读取landsat \*\_MTL.txt文件，提取需要的信息  
   
 Paras:  
 fp - Landsat 文件根目录；string  
   
 return:  
 band\_fp\_dic - 返回各个波段的路径字典；dict  
 Landsat\_para - 返回Landsat 参数 ；dict  
 '''  
 import os,re  
   
 fps=[os.path.join(root,file) for root, dirs, files in os.walk(fp) for file in files] # 提取文件夹下所有文件的路径  
 MTLPattern=re.compile(r'\_MTL.txt',re.S) # 匹配对象模式，提取\_MTL.txt遥感影像的元数据文件  
 MTLFn=[fn for fn in fps if re.findall(MTLPattern,fn)][0]  
 with open(MTLFn,'r') as f: # 读取所有元数据文件信息  
 MTLText=f.read()  
 bandFn\_Pattern=re.compile(r'FILE\_NAME\_BAND\_[0-9]\d\* = "(.\*?)"\n',re.S) # Landsat 波段文件  
 band\_fn=re.findall(bandFn\_Pattern,MTLText)  
 band\_fp=[[(re.findall(r'B[0-9]\d\*',fn)[0], re.findall(r'.\*?%s$'%fn,f)[0]) for f in fps if re.findall(r'.\*?%s$'%fn,f)] for fn in band\_fn] # (文件名，文件路径)  
 band\_fp\_dic={i[0][0]:i[0][1] for i in band\_fp}  
 # 需要数据的提取标签/根据需要读取元数据信息  
 values\_fields=["RADIANCE\_ADD\_BAND\_10",  
 "RADIANCE\_ADD\_BAND\_11",  
 "RADIANCE\_MULT\_BAND\_10",  
 "RADIANCE\_MULT\_BAND\_11",  
 "K1\_CONSTANT\_BAND\_10",  
 "K2\_CONSTANT\_BAND\_10",  
 "K1\_CONSTANT\_BAND\_11",  
 "K2\_CONSTANT\_BAND\_11",  
 "DATE\_ACQUIRED",  
 "SCENE\_CENTER\_TIME",  
 "MAP\_PROJECTION",  
 "DATUM",  
 "UTM\_ZONE"]  
  
 Landsat\_para={field:re.findall(re.compile(r'%s = "\*(.\*?)"\*\n'%field),MTLText)[0] for field in values\_fields} #（参数名，参数值）  
 return band\_fp\_dic,Landsat\_para # 返回所有波段路径和需要的参数值  
  
band\_fp\_dic,Landsat\_para=LandsatMTL\_info(Landsat\_fp["w\_180310"])  
print(band\_fp\_dic)  
print(Landsat\_para)

{'B1': 'F:\\data\\Landsat\\LC08\_L1TP\_023031\_20180310\_20180320\_01\_T1\\LC08\_L1TP\_023031\_20180310\_20180320\_01\_T1\_B1.TIF', 'B2': 'F:\\data\\Landsat\\LC08\_L1TP\_023031\_20180310\_20180320\_01\_T1\\LC08\_L1TP\_023031\_20180310\_20180320\_01\_T1\_B2.TIF', 'B3': 'F:\\data\\Landsat\\LC08\_L1TP\_023031\_20180310\_20180320\_01\_T1\\LC08\_L1TP\_023031\_20180310\_20180320\_01\_T1\_B3.TIF', 'B4': 'F:\\data\\Landsat\\LC08\_L1TP\_023031\_20180310\_20180320\_01\_T1\\LC08\_L1TP\_023031\_20180310\_20180320\_01\_T1\_B4.TIF', 'B5': 'F:\\data\\Landsat\\LC08\_L1TP\_023031\_20180310\_20180320\_01\_T1\\LC08\_L1TP\_023031\_20180310\_20180320\_01\_T1\_B5.TIF', 'B6': 'F:\\data\\Landsat\\LC08\_L1TP\_023031\_20180310\_20180320\_01\_T1\\LC08\_L1TP\_023031\_20180310\_20180320\_01\_T1\_B6.TIF', 'B7': 'F:\\data\\Landsat\\LC08\_L1TP\_023031\_20180310\_20180320\_01\_T1\\LC08\_L1TP\_023031\_20180310\_20180320\_01\_T1\_B7.TIF', 'B8': 'F:\\data\\Landsat\\LC08\_L1TP\_023031\_20180310\_20180320\_01\_T1\\LC08\_L1TP\_023031\_20180310\_20180320\_01\_T1\_B8.TIF', 'B9': 'F:\\data\\Landsat\\LC08\_L1TP\_023031\_20180310\_20180320\_01\_T1\\LC08\_L1TP\_023031\_20180310\_20180320\_01\_T1\_B9.TIF', 'B10': 'F:\\data\\Landsat\\LC08\_L1TP\_023031\_20180310\_20180320\_01\_T1\\LC08\_L1TP\_023031\_20180310\_20180320\_01\_T1\_B10.TIF', 'B11': 'F:\\data\\Landsat\\LC08\_L1TP\_023031\_20180310\_20180320\_01\_T1\\LC08\_L1TP\_023031\_20180310\_20180320\_01\_T1\_B11.TIF'}  
{'RADIANCE\_ADD\_BAND\_10': '0.10000', 'RADIANCE\_ADD\_BAND\_11': '0.10000', 'RADIANCE\_MULT\_BAND\_10': '3.3420E-04', 'RADIANCE\_MULT\_BAND\_11': '3.3420E-04', 'K1\_CONSTANT\_BAND\_10': '774.8853', 'K2\_CONSTANT\_BAND\_10': '1321.0789', 'K1\_CONSTANT\_BAND\_11': '480.8883', 'K2\_CONSTANT\_BAND\_11': '1201.1442', 'DATE\_ACQUIRED': '2018-03-10', 'SCENE\_CENTER\_TIME': '16:34:42.6511940Z', 'MAP\_PROJECTION': 'UTM', 'DATUM': 'WGS84', 'UTM\_ZONE': '16'}

从元文件获取的信息中，能够得知最基本的信息，包括投影坐标系统，为WGS84，UTM\_ZONE 16。如果要对影像裁切，需要保持用于裁切的边界.shp文件与Landsat影像保持相同的投影。可以用rasterio库读取一个Landsat波段，直接获取该波段的crs格式的投影坐标系统来转化.shp边界的坐标系统。EarthPy库提供的裁切工具.crop\_all可以一次性裁切多个波段（即栅格文件）。

import geopandas as gpd  
from pyproj import CRS  
import rasterio as rio  
  
shape\_polygon\_fp=r'.\data\LandsatChicago\_boundary\LandsatChicago\_boundary.shp'  
crop\_bound=gpd.read\_file(shape\_polygon\_fp)  
  
# 获取Landsat遥感影像的投影坐标系统，用于.shp格式的裁切边界，保持投影一致，裁切正确  
with rio.open(band\_fp\_dic['B1']) as raster\_crs:  
 crop\_raster\_profile=raster\_crs.profile  
 crop\_bound\_utm16N=crop\_bound.to\_crs(crop\_raster\_profile["crs"])  
print(crop\_bound\_utm16N.crs)  
  
workspace=r"F:\data\Landsat\data\_processed"  
output\_dir=os.path.join(workspace,r"w\_180310")  
w\_180310\_band\_paths\_list=es.crop\_all(band\_fp\_dic.values(), output\_dir, crop\_bound\_utm16N, overwrite=True) # 对所有波段band执行裁切  
print("cropping finished ...")

PROJCS["WGS 84 / UTM zone 16N",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",0.0174532925199433,AUTHORITY["EPSG","9122"]],AUTHORITY["EPSG","4326"]],PROJECTION["Transverse\_Mercator"],PARAMETER["latitude\_of\_origin",0],PARAMETER["central\_meridian",-87],PARAMETER["scale\_factor",0.9996],PARAMETER["false\_easting",500000],PARAMETER["false\_northing",0],UNIT["metre",1,AUTHORITY["EPSG","9001"]],AXIS["Easting",EAST],AXIS["Northing",NORTH],AUTHORITY["EPSG","32616"]]  
cropping finished ...

将裁切后的文件堆叠在一个变量下，可以直接使用索引的方式读取任何一层，方便对多光谱遥感影像管理。

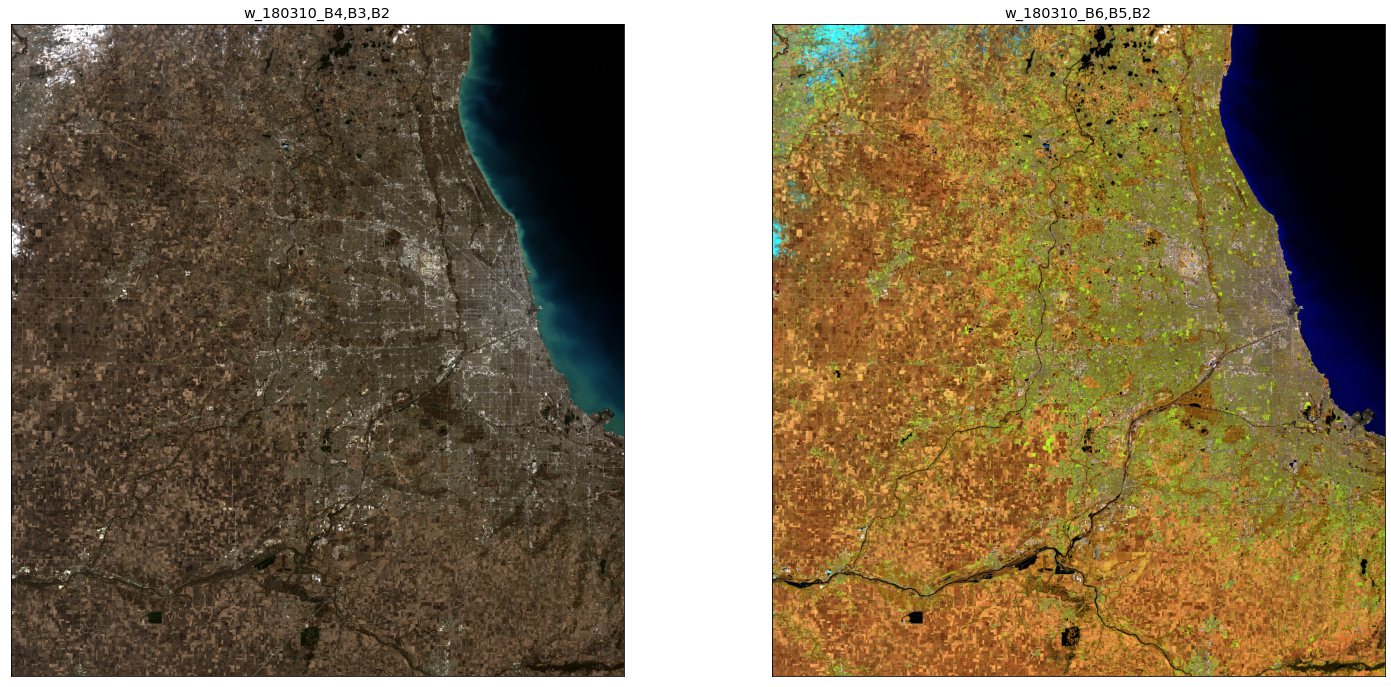
w\_180310\_stack\_fp=os.path.join(workspace,r"w\_180310\_stack.tif")  
w\_180310\_array, w\_180310\_raster\_prof=es.stack(w\_180310\_band\_paths\_list[:7], out\_path=w\_180310\_stack\_fp)  
print("stacking finished...")

stacking finished...

将不同的波段合成显示，可以有不同的显示结果，方便不同目的性的观察，通常如下：

| RGB通道编号 | 波段名 | 主要用途 |
| --- | --- | --- |
| 4、 3 、 2 | Red、 Green 、 Blue | 自然真彩色 |
| 7、 6 、 4 | SWIR2、 SWIR1 、 Red | 城市 |
| 5、4、3 | NIR、 Red 、 Green | 标准假彩色图像，植被 |
| 6、5、2 | SWIR1、 NIR 、 Blue | 农业 |
| 7、 6 、 5 | SWIR2、 SWIR1 、 NIR | 穿透大气层 |
| 5、 6 、 2 | NIR、 SWIR1 、 Blue | 健康植被 |
| 5、 6 、 4 | NIR、 SWIR1 、 Red | 陆地/水 |
| 7、 5 、 3 | SWIR2、 NIR 、 Green | 移除大气影响的自然表面 |
| 7、 5 、 4 | SWIR2、 NIR 、 Red | 短波红外 |
| 6、 5 、 4 | SWIR1、 NIR 、 Red | 植被分析 |
|  |  |  |

from rasterio.plot import plotting\_extent  
extent=plotting\_extent(w\_180310\_array[0], w\_180310\_raster\_prof["transform"])  
  
plt.rcParams.update({'font.size': 12})  
fig, axs = plt.subplots(1,2,figsize=(25, 12))  
ep.plot\_rgb(  
 w\_180310\_array,  
 rgb=[3,2,1], # 对应波段文件为：B4,B3,B2  
 stretch=True,  
 extent=extent,  
 str\_clip=0.5,  
 title="w\_180310\_B4,B3,B2",  
 ax=axs[0]  
)  
  
ep.plot\_rgb(  
 w\_180310\_array,  
 rgb=[5,4,1], # 对应波段文件为：B6,B5,B2  
 stretch=True,  
 extent=extent,  
 str\_clip=0.5,  
 title="w\_180310\_B6,B5,B2",  
 ax=axs[1]  
)  
  
plt.show()



### 2.2.2.2 主成分分析（Principal components analysis，PCA）

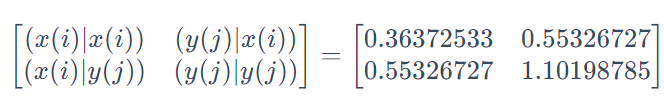
#### 1）主成分分析解析

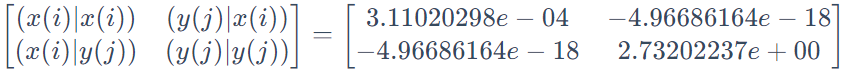
除了全色之外的所有波段，是否有一种方式可以将10个波段的信息降维为1个，并保留最大的信息量，然后再结合相关波段共同打印RGB图像？主成分分析（principal component analysis ，PCA）即为其中一种降低数据维度的技术，可以用于压缩数据，减少内存使用和减负CPU/GPU的处理能力，探索数据集并将高维数据在二三维度上显示观察。

理解该部分的知识，建议先阅读“编程线性代数”部分内容。

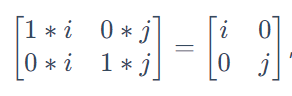
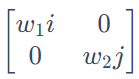
主成分分析是一种统计分析，简化数据集的方法。利用**正交变换**对一系列可能相关的变量观测值进行线性变换，从而投影为一系列线性无关变量的值，这些不相关变量称为主成分（Principal Componnets）。主成分分析经常用于减少数据集的维数，并保留数据集中对方差贡献最大的特征。主成分分析依赖所给数据，数据的准确性对分析结果影响很大。对主成分分析的解读，amoeba给出的[PCA解读](https://stats.stackexchange.com/questions/2691/making-sense-of-principal-component-analysis-eigenvectors-eigenvalues/140579#140579)⑧配合GIF动画是最容易让人理解的一种解读，因此以该解读为主要结构，使用Python实现。这里为了方便阐述PCA，自行生成了一个随机数据集X，包含两个特征，即两组解释变量，为了使得数据两个特征具有一定的线性关系，对该数据集乘以一个变换矩阵transformation= [[-0.6, -0.6], [-0.3, -0.9]]（参看线性映射部分）。

这些特征点的分布（分别由一个特征作为x坐标，另一个作为y坐标），将其绘制于二维的笛卡尔坐标系（Cartesian coordinate system，也称直角坐标系）。目的是试图找到一条直线（新的坐标轴），再将所有点投影到该直线上，从而构建一个新属性，这个属性将由线性组合定义，每条线对应$w\_{1} w\_{2} $的特定值。那么在下述旋转的直线中，哪条直线的投影点能够保留原特征数据集的最大信息量？首先这条线上投影点的差异应该最大化，即反应分布的方差最大。从下图的变化中，可以观察到新特征即红色投影点（重建于原本的两个特征）方差的变化，当直线旋转到蓝色轴v\_n\_j时，方差达到最大，从特征点的分布上也能够容易观察到这个变化趋势。这个投影点当其方差大于到第2个轴的投影时为第1主成分，否则为第2主成分。整体的重建误差通过特征点到相应投影点的距离均方根来衡量，而两个旋转的轴永远是垂直的。投影点到特征点的距离与投影点到原点的距离正好与特征点到原点的距离构成直角三角形，因此对于重建误差的测量可以转换为投影点方差的测量，即投影点方差越高，误差越低。

使得投影点方差为最大的新坐标轴是由特征点的协方差求得，

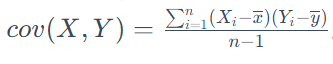
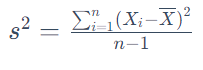
x(i)|x(i)y(j)|y(j)y(j)|x(i)x(i)|y(j) 则是特征之间的协方差，其值相同。所要寻找新的坐标系是使得特征点到新坐标系其中一个轴的投影的方差为最大，相对到其它轴投影方差则次第减小，则说明轴之间的方差即y(j)∣x(i)和x(i)∣y(j)趋近于0，其它轴的投影点方差也相对较小。的方差最大，其它的均趋于0。轴之间的协方差基本为0，基本不存在相关性；即当协方差为0时，表示两个特征基本完全独立。而为了让协方差为0，在确定第2个轴（基）时只能在与第一个轴（基）正交方向上选择，因此新坐标系的两个轴是正交垂直的。

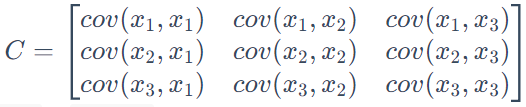
由上述表述可以进一步推广到一般维度上的降维问题，将一组N维特征向量降到K维，其中，选择K个单位（模为1）的正交基（新的坐标系或向量空间），使得原始数据变换到这组基上后，各个特征间协方差为0，而特征的方差则尽可能大，在正交的约束下取最大的K各方差对应的基（轴）。

由原始特征值的协方差方阵计算特征向量和特征值，所计算的单位特征向量就是新坐标轴的方向，而特征值是坐标轴的长度（相对于原始对应坐标轴单位的倍数）。对于为什么协方差矩阵的特征向量和特征值的求得就为新的坐标系，可以理解为，假设原始特征的坐标系的轴表示为：，单位长度为1，所有特征向量可以表示为，如果将方差的值认为是能反映该类特征分布的一个代表，则所有的特征向量可以简化表示为，那么协方差矩阵可以表示为：新的坐标系可以表示为：

$。可以观察到协方差矩阵与新坐标系形式相似，对协方差矩阵对角化（即除了对角线外的其它元素化为0）后，就可以获得新坐标系的轴（基），对角化就是求特征向量和特征值。

* 方差和协方差

在方差和协方差部分已经给出了方差和协方差的解释，方差（Variance）用于衡量一组值的分布，是每个值和均值平方差的平均，公式为：  。如果协方差为0，则变量不相关；如果非0，正负号表示正负相关。正相关时，一个变量随另一个变量的增加而增加；为负相关时，一个变量随其均值增加时，另一个变量相对于其均值减小。协方差矩阵（离差矩阵，或方差-协方差矩阵）描述了一个数据集中每一对维度数变量的协方差。如果元素表示数据

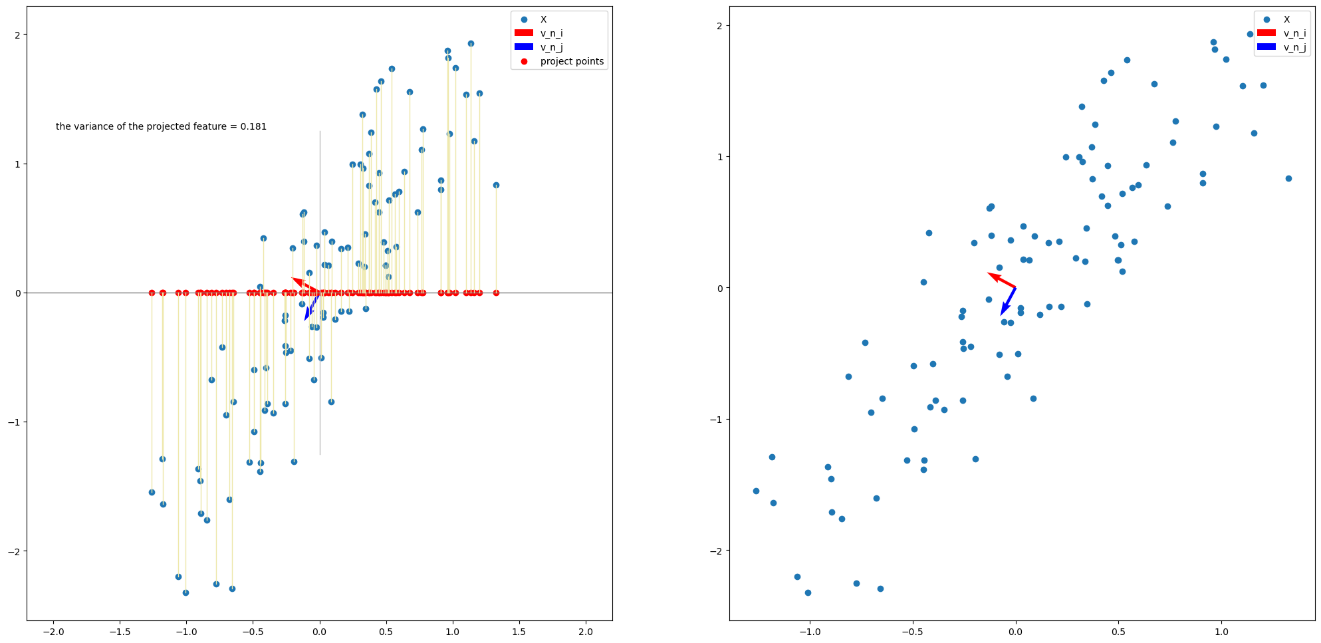
使用经典的iris数据集，拓展协方差二维到4维数据，每一维度为一个特征。协方差的计算直接使用NumPy、pandas等库。

* 再议特征值和特征向量

对于公式A 或 仅仅在尺度上变为原来的倍。称向量*ξ*或 是的一个特征向量，是对应的特征值，即一个矩阵乘以它的特征向量，等于对这个特征向量作缩放。特征向量和特征值只能由方阵衍生，同时并非所有的方阵都有特征向量和特征值。如果一个矩阵有特征向量和特征值，它的每一个维度上都有一个特征向量和特征值。一个矩阵的主成分是它的协方差矩阵的特征向量，对应最大特征值的特征向量是第1个主成分，对应第2大特征值的特征向量是第2个主成分，以此类推。

%matplotlib inline  
import numpy as np  
from numpy import linalg as LA  
from sympy.vector.coordsysrect import CoordSys3D  
from sympy.vector.vector import Vector, BaseVector  
from sympy import pprint  
import matplotlib.pyplot as plt  
import matplotlib.animation  
from IPython.display import HTML  
  
fig, axs=plt.subplots(1,2,figsize=(25,12))  
np.random.seed(42)  
  
# 定义向量空间 C  
C=CoordSys3D('C')  
i, j, k = C.base\_vectors()  
v1\_origin=Vector.zero  
  
# 建立随机数据集  
n\_samples=100  
X=np.random.randn(100,2)  
transformation= [[-0.6, -0.6], [-0.3, -0.9]]  
X=np.dot(X, transformation)  
axs[0].scatter(X[:,0],X[:,1],label='X')  
axs[1].scatter(X[:,0],X[:,1],label='X')  
  
# 计算特征值和特征向量  
w,v=LA.eig(np.cov(X.T))  
print("covariance:\n",np.cov(X.T))  
print("eigenvalues:\n",w)  
print("eigenvectors:\n",v)  
  
# 特征值乘以特征向量和特征值，获得样本新的特征  
new\_X=np.matmul(X,w\*v)  
print("new build fearure covariance:\n",np.cov(new\_X.T))  
  
# 根据特征向量建立新坐标轴  
v\_n\_i=v[0][0]\*i+v[1][0]\*j  
v\_n\_j=v[0][1]\*i+v[1][1]\*j  
  
def vector\_plot\_2d(ax\_2d,C,origin\_vector,vector,color='r',label='vector',width=0.022):  
 '''  
 funciton - 转换SymPy的vector及Matrix数据格式为matplotlib可以打印的数据格式  
  
 Params:  
 ax\_2d - matplotlib的2d格式子图；ax(matplotlib)  
 C - /coordinate\_system - SymPy下定义的坐标系；CoordSys3D()  
 origin\_vector - 如果是固定向量，给定向量的起点（使用向量，即表示从坐标原点所指向的位置），如果是自由向量，起点设置为坐标原点；vector(SymPy)  
 vector - 所要打印的向量；vector(SymPy)  
 color - 向量色彩，The default is 'r'；string  
 label - 向量标签，The default is 'vector'；string  
 arrow\_length\_ratio - 向量箭头大小，The default is 0.022；float   
   
 Returns:  
 None  
 '''  
 origin\_vector\_matrix=origin\_vector.to\_matrix(C)  
 x=origin\_vector\_matrix.row(0)[0]  
 y=origin\_vector\_matrix.row(1)[0]  
  
 vector\_matrix=vector.to\_matrix(C)  
 u=vector\_matrix.row(0)[0]  
 v=vector\_matrix.row(1)[0]  
  
 ax\_2d.quiver(float(x),float(y),float(u),float(v),color=color,label=label,width=width)  
  
vector\_plot\_2d(axs[0],C,v1\_origin,v\_n\_i,color='r',label='v\_n\_i',width=0.005)  
vector\_plot\_2d(axs[0],C,v1\_origin,v\_n\_j,color='b',label='v\_n\_j',width=0.005)  
  
vector\_plot\_2d(axs[1],C,v1\_origin,v\_n\_i,color='r',label='v\_n\_i',width=0.005)  
vector\_plot\_2d(axs[1],C,v1\_origin,v\_n\_j,color='b',label='v\_n\_j',width=0.005)  
  
# 绘制旋转的坐标轴  
def circle\_lines(center,radius,division):   
 '''  
 function - 给定圆心，半径，划分份数，计算所有直径的首尾坐标  
  
 Params:  
 center - 圆心，例如(0,0)；tuple  
 radius - 半径；float  
 division - 划分份数；int  
   
 Returns:  
 xy - 首坐标数组；array  
 xy\_ -尾坐标数组；array  
 xy\_head\_tail - 收尾坐标数组；array  
 '''   
 import math  
 import numpy as np  
   
 angles=np.linspace(0,2\*np.pi,division)  
 x=np.cos(angles)\*radius  
 y=np.sin(angles)\*radius  
 xy=np.array(list(zip(x,y)))   
 xy=xy+center  
  
 x\_=-x  
 y\_=-y  
 xy\_=np.array(list(zip(x\_,y\_)))  
 xy\_=xy\_+center  
   
 xy\_head\_tail=np.concatenate((xy,xy\_),axis=1)  
 return xy,xy\_,xy\_head\_tail  
  
center=(0,0)  
radius=5  
division=360  
\_,\_,xy\_head\_tail=circle\_lines(center,radius,division)  
  
# 点到直线的投影  
def point\_Proj2Line(line\_endpts,point):   
 '''  
 function - 计算二维点到直线上的投影  
   
 Params:  
 line\_endpts - 直线首尾点坐标，例如((2,0),(-2,0))；tuple/list  
 point - 待要投影的点，例如[-0.11453985, 1.23781631]；tuple/list  
   
 Returns:  
 P - 投影点；tuple/list  
 '''  
 import numpy as np  
   
 pts=np.array(point)  
 Line\_s=np.array(line\_endpts[0])  
 Line\_e=np.array(line\_endpts[1])  
   
 n=Line\_s - Line\_e  
 n\_=n/np.linalg.norm(n, 2)   
 P=Line\_e + n\_\*np.dot(pts - Line\_e, n\_)  
   
 return P  
  
line,=axs[0].plot([2, -2],[0,0], color='gray', linewidth = 1) # 旋转的轴1初始化  
line\_v,=axs[0].plot([0, 0],[2,-2], color='silver', linewidth = 1)# 旋转的轴2（垂直轴1）初始化  
  
pts\_proj\_initial=np.array([point\_Proj2Line(((2,0),(-2,0)),p) for p in X ]) # 计算投影点  
pts\_proj=axs[0].scatter(pts\_proj\_initial[:,0],pts\_proj\_initial[:,1],c='r',label='project points') # 投影点初始化  
  
var\_text=axs[0].text(0.05, 0.8, '', transform=axs[0].transAxes) # 打印文字初始化  
var\_template='the variance of the projected feature = %.3f'  
  
v\_lines=axs[0].plot([pts\_proj\_initial[:,0],X[:,0]],[pts\_proj\_initial[:,1],X[:,1]], color='palegoldenrod',linewidth = 1) # 特征点和投影点垂直线初始化  
  
# 子图更新  
def update(xy\_head\_tail):  
 '''  
 function - matplolib animation update method  
 '''  
 x,y,x\_,y\_=xy\_head\_tail[0], xy\_head\_tail[1],xy\_head\_tail[2],xy\_head\_tail[3]  
 pts\_p=np.array([point\_Proj2Line(((x,y),(x\_,y\_)),p) for p in X ])  
   
 line.set\_data([x,x\_],[y,y\_])   
 line\_v.set\_data([-y/4,-y\_/4],[x/4,x\_/4])   
   
 pts\_proj.set\_offsets(pts\_p)  
 pts\_proj\_var=np.var(pts\_p)  
   
 var\_text.set\_text(var\_template % (pts\_proj\_var))   
 for i in range(pts\_p.shape[0]):  
 v\_lines[i].set\_data([pts\_p[i][0],X[i][0]],[pts\_p[i][1],X[i][1]])  
   
 return line, # 需要保留“,”  
  
axs[0].legend()  
axs[1].legend()  
anima=matplotlib.animation.FuncAnimation(fig, update, frames=xy\_head\_tail, blit=True, interval=100)  
HTML(anima.to\_html5\_video()) # conda install -c conda-forge ffmpeg

covariance:  
 [[0.36372533 0.55326727]  
 [0.55326727 1.10198785]]  
eigenvalues:  
 [0.06775316 1.39796002]  
eigenvectors:  
 [[-0.88175912 -0.47169996]  
 [ 0.47169996 -0.88175912]]  
new build fearure covariance:  
 [[ 3.11020298e-04 -4.96686164e-18]  
 [-4.96686164e-18 2.73202237e+00]]



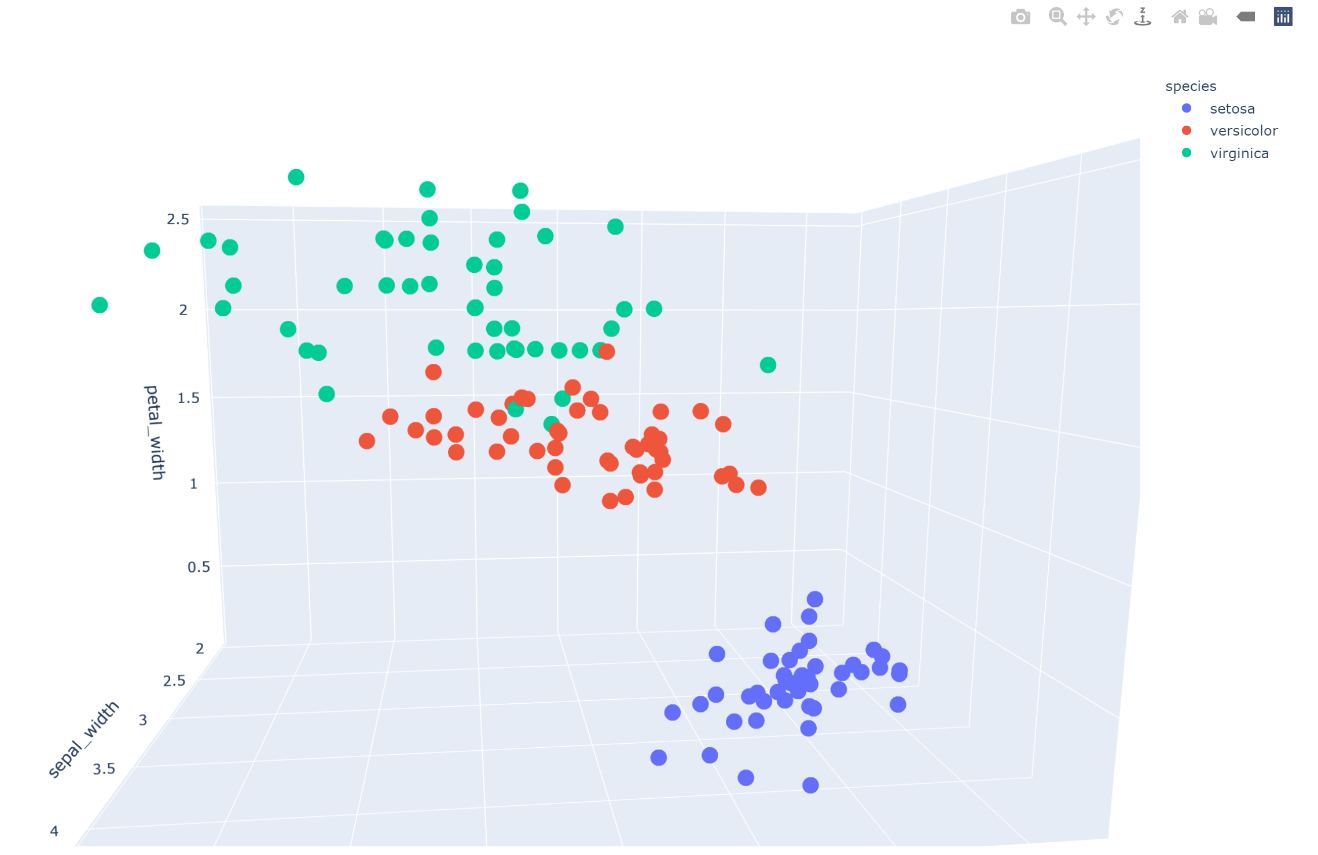
#保存动画为.gif文件  
from matplotlib.animation import FuncAnimation, PillowWriter   
writer = PillowWriter(fps=25)   
anima.save("./graph/pca.gif", writer=writer)

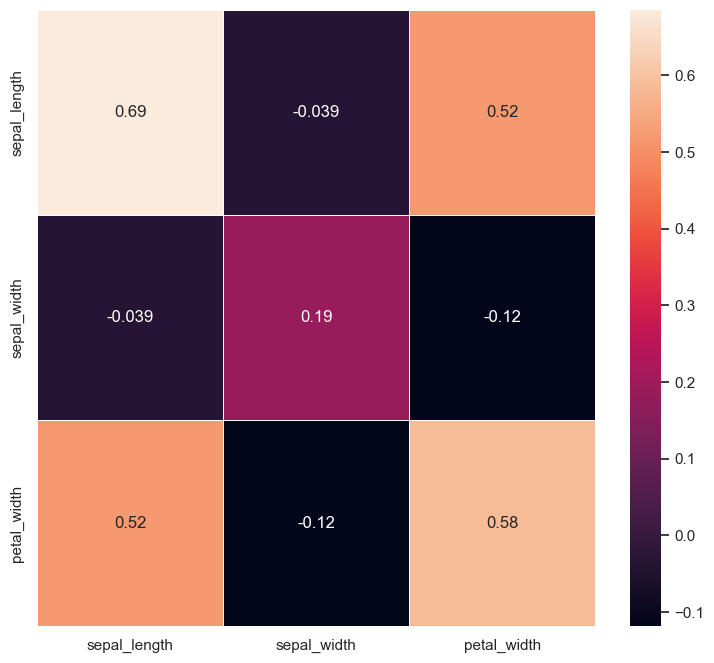
#### 2）三维度PCA

上述阐述PCA时，从二维度入手，这样比较方便观察。下述则以机器学习经典鸢尾花（iris）数据集为例，提取其中的三个特征用于说明三个维度下特征值的PCA降维。基本过程同两个维度，只是在图表分析上结合使用三维图形，方便观察。对于更高的维度，则可以借助对2，3维的理解，想象构建方便理解的高纬度空间。目前对于高纬度空间的图表研究也有很多研究成果，可以尝试借助其中的方法来可视化高维空间。

import numpy as np  
import pandas as pd  
import plotly.express as px  
import matplotlib.pyplot as plt  
import seaborn as sns  
sns.set()  
  
iris\_df=px.data.iris()  
feature\_selection=["sepal\_length","sepal\_width","petal\_width","species"]  
iris\_selection=iris\_df[feature\_selection]  
print(iris\_selection)  
print("-"\*50)  
COV\_iris=iris\_selection[["sepal\_length","sepal\_width","petal\_width"]].cov() # 先从3维度讲起，方便打印观察，["sepal\_length","sepal\_width","petal\_length","petal\_width"]  
print("iris covariance")  
print(COV\_iris)  
f, ax = plt.subplots(figsize=(9,8))  
sns.heatmap(COV\_iris, annot=True, linewidths=.5, ax=ax)  
  
fig=px.scatter\_3d(iris\_df, x='sepal\_length', y='sepal\_width', z='petal\_width',color='species',width=1200, height=800) #symbol='species'  
fig.show()

sepal\_length sepal\_width petal\_width species  
0 5.1 3.5 0.2 setosa  
1 4.9 3.0 0.2 setosa  
2 4.7 3.2 0.2 setosa  
3 4.6 3.1 0.2 setosa  
4 5.0 3.6 0.2 setosa  
.. ... ... ... ...  
145 6.7 3.0 2.3 virginica  
146 6.3 2.5 1.9 virginica  
147 6.5 3.0 2.0 virginica  
148 6.2 3.4 2.3 virginica  
149 5.9 3.0 1.8 virginica  
  
[150 rows x 4 columns]  
--------------------------------------------------  
iris covariance  
 sepal\_length sepal\_width petal\_width  
sepal\_length 0.685694 -0.039268 0.516904  
sepal\_width -0.039268 0.188004 -0.117981  
petal\_width 0.516904 -0.117981 0.582414

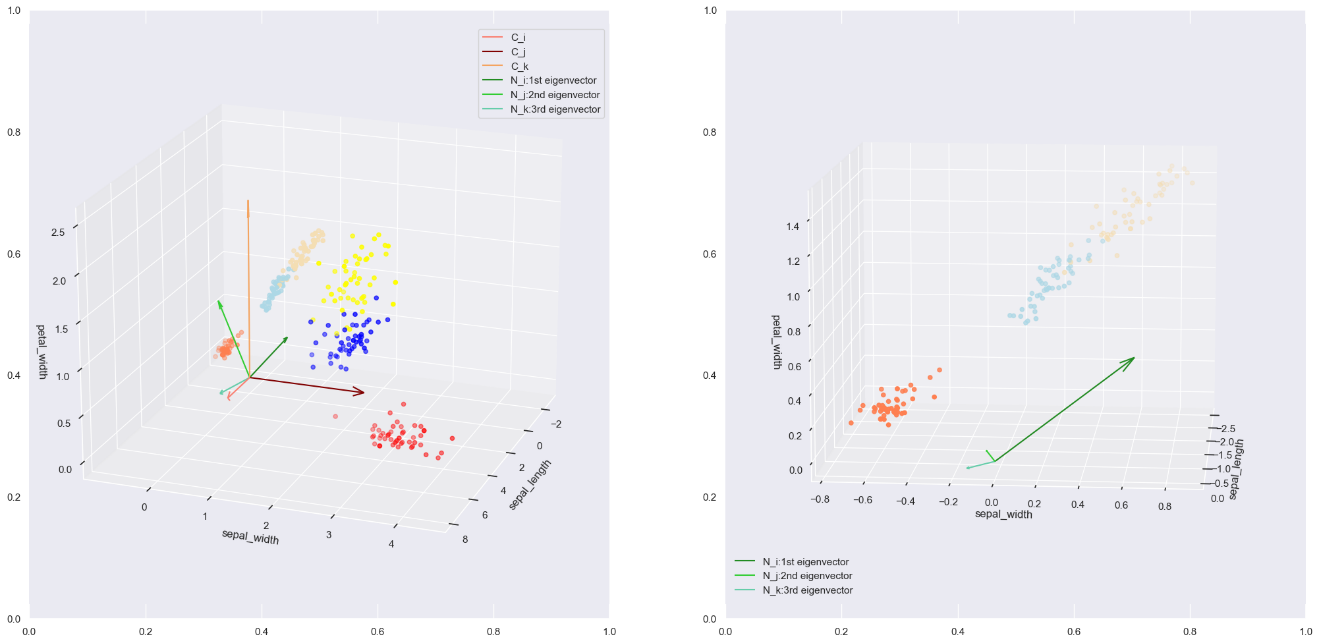




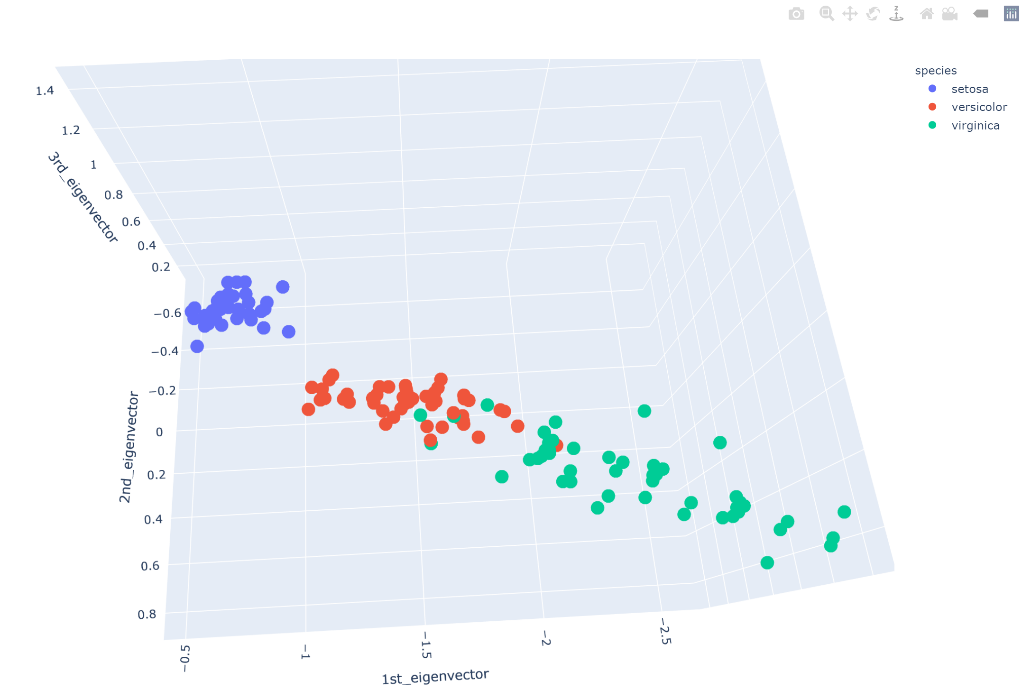
在三维空间打印特征点，和对应的投影点，及各自的坐标系。第1主成分延着N\_i轴分布，具有较大的方差，而另外两个轴的特征值相对第一个轴值很小，即在2，3轴的特征点对应的投影点的方差很小。因此第一主成分很好的保留了iris特征值的属性，达到了降维的目的。

from numpy import linalg as LA  
import util\_A  
from sympy.vector.coordsysrect import CoordSys3D  
from sympy.vector.vector import Vector, BaseVector  
from sympy import pprint  
  
COV\_iris\_array=COV\_iris.to\_numpy()  
w,v=LA.eig(COV\_iris\_array)  
print("eigenvalues:\n",w)  
print("eigenvectors\_normalized:\n",v)  
  
iris\_3Features=iris\_df[feature\_selection]  
import pandas as pd  
pd.options.mode.chained\_assignment = None  
species\_colors={'setosa':'red', 'versicolor':'blue', 'virginica':'yellow'}  
iris\_3Features['color']=iris\_3Features.species.apply(lambda row:species\_colors[row])  
  
fig, axs=plt.subplots(1,2,figsize=(25,12))  
axs[0]=fig.add\_subplot(1,2,1, projection='3d')  
axs[1]=fig.add\_subplot(1,2,2,projection='3d')  
axs[0].scatter(iris\_3Features.sepal\_length,iris\_3Features.sepal\_width,iris\_3Features.petal\_width,c=iris\_3Features.color,)  
  
C=CoordSys3D('C')  
i, j, k = C.base\_vectors()  
v1\_origin=Vector.zero  
util\_A.vector\_plot\_3d(axs[0],C,v1\_origin,2\*i,color='salmon',label='C\_i',arrow\_length\_ratio=0.1)  
util\_A.vector\_plot\_3d(axs[0],C,v1\_origin,2\*j,color='maroon',label='C\_j',arrow\_length\_ratio=0.1)  
util\_A.vector\_plot\_3d(axs[0],C,v1\_origin,2\*k,color='sandybrown',label='C\_k',arrow\_length\_ratio=0.1)  
  
# 单位特征向量\*特征值（倍数）  
N\_i\_=w[0]\*(v[0][0]\*i+v[0][1]\*j+v[0][2]\*k)  
N\_j\_=w[1]\*(v[1][0]\*i+v[1][1]\*j+v[1][2]\*k)  
N\_k\_=w[2]\*(v[2][0]\*i+v[2][1]\*j+v[2][2]\*k)  
  
# 单位特征向量  
N\_i=v[0][0]\*i+v[0][1]\*j+v[0][2]\*k  
N\_j=v[1][0]\*i+v[1][1]\*j+v[1][2]\*k  
N\_k=v[2][0]\*i+v[2][1]\*j+v[2][2]\*k  
util\_A.vector\_plot\_3d(axs[0],C,v1\_origin,N\_i,color='forestgreen',label='N\_i:1st eigenvector',arrow\_length\_ratio=0.1)  
util\_A.vector\_plot\_3d(axs[0],C,v1\_origin,N\_j,color='limegreen',label='N\_j:2nd eigenvector',arrow\_length\_ratio=0.1)  
util\_A.vector\_plot\_3d(axs[0],C,v1\_origin,N\_k,color='mediumaquamarine',label='N\_k:3rd eigenvector',arrow\_length\_ratio=0.1)  
  
util\_A.vector\_plot\_3d(axs[1],C,v1\_origin,N\_i\_,color='forestgreen',label='N\_i:1st eigenvector',arrow\_length\_ratio=0.1)  
util\_A.vector\_plot\_3d(axs[1],C,v1\_origin,N\_j\_,color='limegreen',label='N\_j:2nd eigenvector',arrow\_length\_ratio=0.1)  
util\_A.vector\_plot\_3d(axs[1],C,v1\_origin,N\_k\_,color='mediumaquamarine',label='N\_k:3rd eigenvector',arrow\_length\_ratio=0.1)  
  
# 特征值映射到N\_i,N\_j,N\_k的向量空间N  
# 构建变换矩阵  
N\_f\_matrix=N\_i\_.to\_matrix(C)  
for v in [N\_j\_,N\_k\_]:  
 v\_temp=v.to\_matrix(C)  
 N\_f\_matrix=N\_f\_matrix.col\_insert(-1,v\_temp)  
print("变换矩阵：")  
pprint(N\_f\_matrix)  
  
iris\_mapping\_N=np.matmul(iris\_df[feature\_selection[:3]].to\_numpy(),N\_f\_matrix.T).astype(float) # 线性映射  
species\_colors\_={'setosa':'coral', 'versicolor':'lightblue', 'virginica':'wheat'}  
iris\_3Features['color\_']=iris\_3Features.species.apply(lambda row:species\_colors\_[row])  
axs[0].scatter(iris\_mapping\_N[:,0],iris\_mapping\_N[:,1],iris\_mapping\_N[:,2],c=iris\_3Features.color\_,)  
axs[1].scatter(iris\_mapping\_N[:,0],iris\_mapping\_N[:,1],iris\_mapping\_N[:,2],c=iris\_3Features.color\_,)  
  
axs[0].legend()  
axs[1].legend()  
  
axs[0].set(xlabel='sepal\_length',ylabel='sepal\_width',zlabel='petal\_width')  
axs[1].set(xlabel='sepal\_length',ylabel='sepal\_width',zlabel='petal\_width')  
axs[1].legend(loc='lower left', frameon=False)  
axs[0].view\_init(20,20)  
axs[1].view\_init(10,5)  
plt.show()

eigenvalues:  
 [1.1655692 0.07718794 0.21335471]  
eigenvectors\_normalized:  
 [[-0.73255285 0.53278011 0.42368818]  
 [ 0.11049104 -0.52110099 0.84631288]  
 [-0.671683 -0.66678266 -0.32286659]]  
变换矩阵：  
⎡0.00852857583562477 -0.143306730789405 -0.85384104677628⎤  
⎢ ⎥  
⎢-0.0402227130979333 -0.14226122114419 0.620992083779818⎥  
⎢ ⎥  
⎣0.0653251494721517 -0.0688851077741475 0.493837898598557⎦

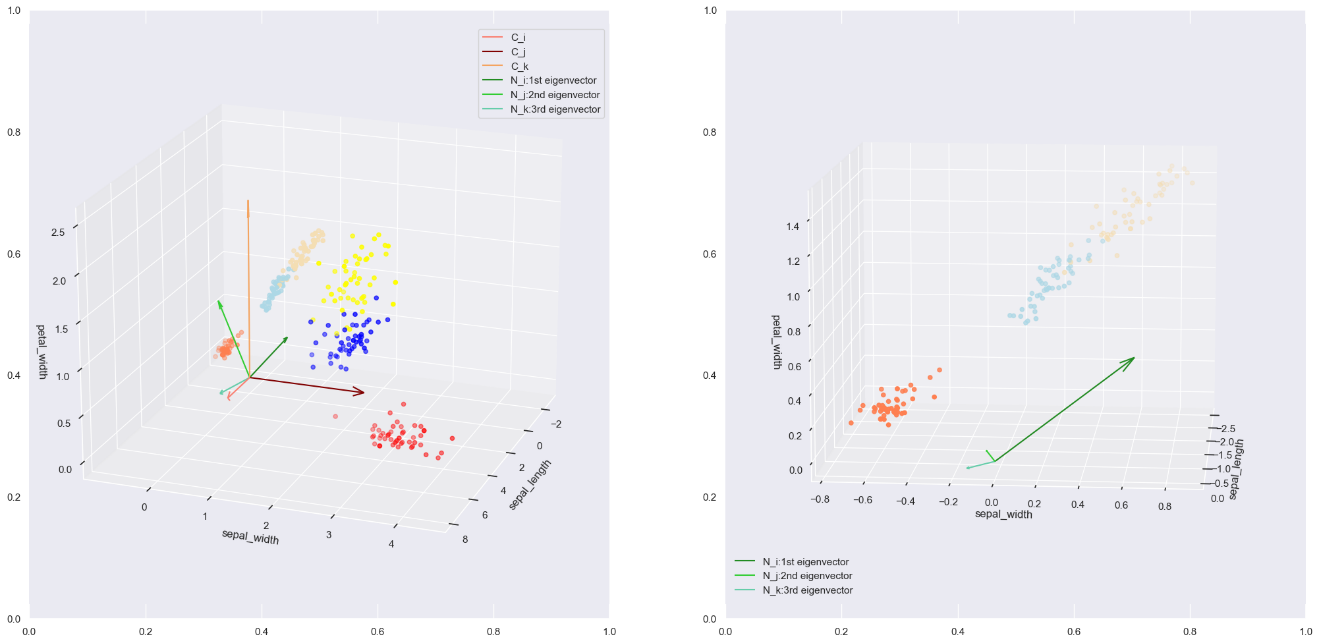


iris\_df\_N=pd.DataFrame(np.hstack((iris\_mapping\_N,iris\_df.species.to\_numpy().reshape(-1,1))),columns=["1st\_eigenvector","2nd\_eigenvector","3rd\_eigenvector","species"])  
fig=px.scatter\_3d(iris\_df\_N, x='1st\_eigenvector', y='2nd\_eigenvector', z='3rd\_eigenvector',color='species',width=1200, height=800) # symbol='species'  
fig.show()



仅保留第1，2个主成分，通过二维图表查看特征点（值）的分布情况。因为具有较大的方差，各个特征值对应的响应变量（iris的种类）能够很好的分离。同样，仅打印第1个主成分，线性分布的结果也可以很好的分离响应变量。

#A - 降维-保留第1，2主成分  
fig, axs=plt.subplots(1,2,figsize=(25,12))  
iris\_PCA\_1st\_2nd=np.matmul(iris\_df[feature\_selection[:3]].to\_numpy(),np.array(N\_f\_matrix).astype(float)[0:2].T).astype(float) # 降维第1，2主成分  
axs[0].scatter(iris\_PCA\_1st\_2nd[:,0],iris\_PCA\_1st\_2nd[:,1],c=iris\_3Features.color\_)  
axs[0].set(xlabel='1st\_eigenvector',ylabel='2nd\_eigenvector',)  
  
iris\_PCA\_1st=np.matmul(iris\_df[feature\_selection[:3]].to\_numpy(),np.array(N\_f\_matrix).astype(float)[0:1].T).astype(float) # 降维第1主成分  
axs[1].scatter(iris\_PCA\_1st,np.zeros(iris\_PCA\_1st.shape),c=iris\_3Features.color\_)  
axs[1].set(xlabel='1st\_eigenvector',ylabel='y',)  
plt.show()



#### 3）Landsat遥感影像波段的PCA降维与RGB显示

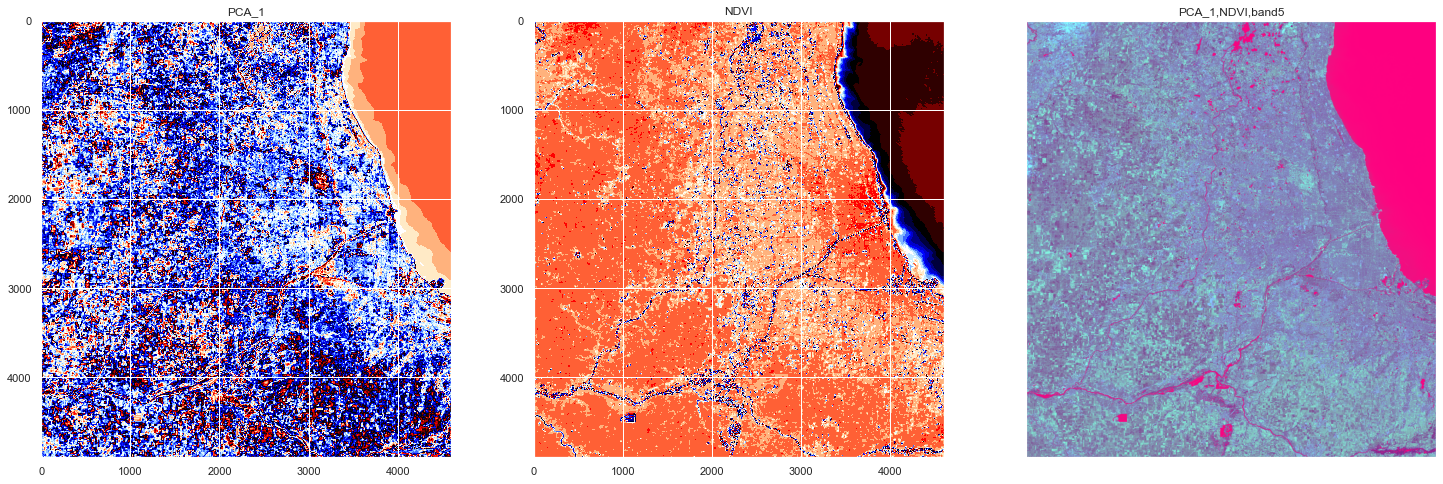
降维Landate的前7个波段为1个，打印比较PCA\_1（主成分分析第1主成分），归一化植被指数（Normalized Difference Vegetation Index，NDVI），及[PCA\_1,NDVI,band\_5]组合。NDVI能够很好的区分出植被区域，当与PCA，和band5组合时，水体区域能够比较好的区分开来。对于何种波段组合或波段间的计算能够较好的反映哪种地物，除了在上述开始引入了各个波段的主要功能，也有很多计算指数辅助解译。例如NDVI，NDWI（Normalized Difference Water Index，归一化水指数），NDBI（Normalized Difference Built-up Index，归一化建筑指数）等，其公式分别为：。

from sklearn.decomposition import PCA  
n\_components=1  
w\_180310\_band\_reduced=PCA(n\_components=n\_components).fit\_transform(w\_180310\_array.T.reshape(-1,w\_180310\_array.shape[0]))  
print("PCA finished.")  
w\_180310\_band\_reduced\_reshape=w\_180310\_band\_reduced.reshape(w\_180310\_array.shape[2],w\_180310\_array.shape[1],n\_components).T

PCA finished.

band\_merge=np.concatenate((w\_180310\_array,w\_180310\_band\_reduced\_reshape),axis=0)  
  
from rasterio.plot import plotting\_extent  
extent=plotting\_extent(w\_180310\_array[0], w\_180310\_raster\_prof["transform"])  
  
plt.rcParams.update({'font.size': 12})  
fig, axs = plt.subplots(1,3,figsize=(25, 12))  
  
axs[0].imshow(w\_180310\_band\_reduced\_reshape[0],cmap='flag') # 'prism'  
axs[0].set\_title("PCA\_1")  
  
# 计算NDVI  
def NDVI(RED\_band,NIR\_band):  
 '''  
 function - 计算NDVI指数  
   
 Params:  
 RED\_band - 红色波段；array  
 NIR\_band - 近红外波段；array  
   
 Returns:  
 NDVI - NDVI指数值；array  
 '''  
 RED\_band=np.ma.masked\_where(NIR\_band+RED\_band==0,RED\_band)  
 NDVI=(NIR\_band-RED\_band)/(NIR\_band+RED\_band)  
 NDVI=NDVI.filled(-9999)  
 print("NDVI"+"\_min:%f,max:%f"%(NDVI.min(),NDVI.max()))  
 return NDVI  
  
RED\_band=w\_180310\_array[3]  
NIR\_band=w\_180310\_array[4]  
w\_180310\_NDVI=NDVI(RED\_band,NIR\_band)  
  
band\_merge=np.concatenate((band\_merge,np.expand\_dims(w\_180310\_NDVI,axis=0)),axis=0)  
  
axs[1].imshow(w\_180310\_NDVI,cmap='flag')   
axs[1].set\_title("NDVI")  
  
ep.plot\_rgb(  
 band\_merge,  
 rgb=[-1,-2,4],   
 stretch=True,  
 extent=extent,  
 str\_clip=0.5,  
 title="PCA\_1,NDVI,band5",  
 ax=axs[2]  
)  
  
plt.show()

!\_min:0.000000,max:15.654902



### 2.2.2.3 数字高程（Digital Elevation）（附）

数字高程模型（digital elevation model，DEM）数据也可以从[美国地质调查局（United States Geological Survey，USGS-earthexplorer）](https://earthexplorer.usgs.gov/)下载。DEM的数据格式为栅格（raster， .tif）格式，处理方法同Landsat遥感影像的处理方式，所有栅格数据的处理模式是基本相同的。

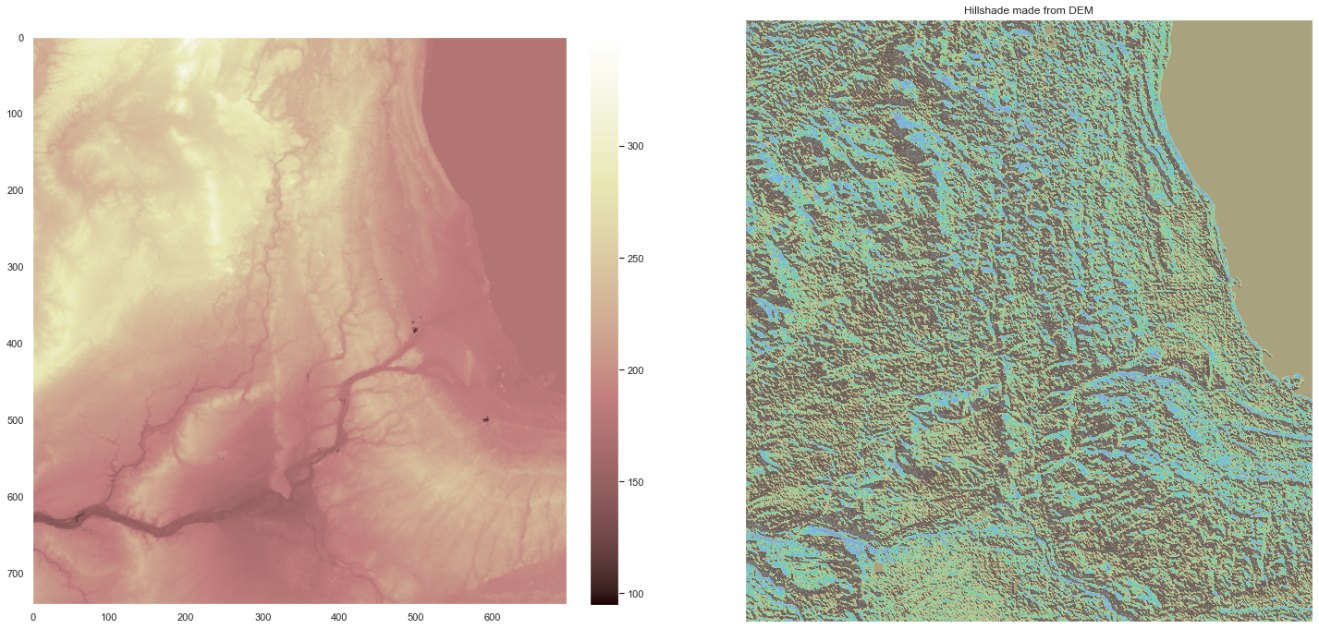
import os  
import numpy as np  
import matplotlib.pyplot as plt  
import earthpy as et  
import earthpy.spatial as es  
import earthpy.plot as ep  
import rasterio as rio  
from rasterio.warp import calculate\_default\_transform, reproject, Resampling  
  
DE\_Chicago\_fp=r"F:\data\Landsat\GMTED2010N30W090\_075\30n090w\_20101117\_gmted\_mea075.tif"  
DEChicago\_reprojecting\_savePath=os.path.join(workspace,r"DE\_Chicago.tif")  
# 投影转换DEM，.tif栅格数据  
dst\_crs=crop\_raster\_profile["crs"]  
with rio.open(DE\_Chicago\_fp) as src:  
 transform, width, height = calculate\_default\_transform(src.crs, dst\_crs, src.width, src.height, \*src.bounds)  
 kwargs = src.meta.copy()  
 kwargs.update({  
 'crs': dst\_crs,  
 'transform': transform,  
 'width': width,  
 'height': height  
 })  
 with rio.open(DEChicago\_reprojecting\_savePath, 'w', \*\*kwargs) as dst:  
 for i in range(1, src.count + 1):  
 reproject(  
 source=rio.band(src, i),  
 destination=rio.band(dst, i),  
 src\_transform=src.transform,  
 src\_crs=src.crs,  
 dst\_transform=transform,  
 dst\_crs=dst\_crs,  
 resampling=Resampling.nearest)   
print("reprojecting finished ...")

reprojecting finished ...

# 裁切DEM .tif数据  
DE\_Chicago=es.crop\_all([DEChicago\_reprojecting\_savePath],workspace, crop\_bound\_utm16N, overwrite=True) #对所有波段band执行裁切  
print("cropping finished ...")

cropping finished ...

DE\_Chicago\_tif=rio.open(DE\_Chicago[0])  
  
# DEM打印  
fig, axs=plt.subplots(1,2,figsize=(25,12))  
dem=axs[0].imshow(DE\_Chicago\_tif.read(1),cmap='pink')  
plt.colorbar(dem,fraction=0.0485, pad=0.04,ax=axs[0])  
  
# 增加山体阴影  
hillshade=es.hillshade(DE\_Chicago\_tif.read(1),azimuth=210,altitude=10)  
ep.plot\_bands(hillshade, cbar=False, title="Hillshade made from DEM", figsize=(10, 6),ax=axs[1],cmap='terrain')  
axs[1].imshow(hillshade, cmap="Greys", alpha=0.5)  
axs[0].grid(False)  
plt.show()



注释（Notes）：

① 美国地质调查局 United States Geological Survey，USGS-earthexplorer，EarthExplor（EE）为美国地质调查局（USGS）档案中的地球科学数据提供在线检索，浏览显示，元数据（metadata）输出和数据下（<https://earthexplorer.usgs.gov/>）。

② Mermaid，是一个基于JavaScript的图表工具，使用Markdown-inspired文本定义和渲染器来创建和修改复杂的图表（<https://mermaid-js.github.io/mermaid>）。

③ LLLL，Landsat Levels of Processing（<https://www.usgs.gov/landsat-missions/landsat-levels-processing>）。

④ TX，Landsat Collection（<https://www.usgs.gov/landsat-missions/landsat-collection-1?qt-science_support_page_related_con=1#qt-science_support_page_related_con>）。

⑤ rasterio，地理信息系统（Geographic information systems，GIS）使用GeoTIFF和其它格式组织和存储栅格数据（raster datasets），如卫星影像和地形模型等。rasterio读写这些文件格式，并提供基于NumPy N-维数组和GeoJSON的Python API（<https://rasterio.readthedocs.io/en/latest>）。

⑥ EarthPy，是一个Python包，使得使用开源工具绘制和处理空间栅格和矢量数据变得更加容易。EarthPy依赖于专注矢量数据的GeoPandas库和方便输入输出栅格数据的rasterio库，并需要matplotlib库绘图，其目标是让科学家更容易地处理空间数据（<https://earthpy.readthedocs.io/en/latest>）。

⑦ GDAL，一些用于编程和操作GDAL地理空间数据抽象库的工具，实际上包括两个库GDAL库和OGR库，分布用于处理地理空间的栅格数据和矢量数据（<https://gdal.org/>）。

⑧ PCA解读，Making sense of principal component analysis, eigenvectors & eigenvalues（<https://stats.stackexchange.com/questions/2691/making-sense-of-principal-component-analysis-eigenvectors-eigenvalues/140579#140579>）。

参考文献（References）:

[1] Cavin Hackeling.张浩然译.scikit-learning 机器学习[M].人民邮电出版社.2019.2.（ Cavin Hackeling. Mastering Machine Learning with scikit-learn[M].Packt Publishing Ltd.July 2017.Second published.）

[2] [日]高桥 信著作,Inoue Iroha,株式会社 TREND-PRO漫画制作,腾永红译.漫画线性代数[M].科学出版社.北京.2009.08.