

# Data Visualization - Homework 5 - Report

邓淇升 大数据学院 16307110232

## 1 编程实现灰度向彩色转换的伪彩色变换算法。

### ● Python script code:

1\_pseudocolor\_transformation.py

### ● Input & Output:

**Input A:** NGC2237.jpg

- NGC2237\_by\_intensity\_slicing\_transformation.png
- NGC2237\_by\_intensity\_to\_color\_transformation.png

**Input B:** lena\_std.tif

- lena\_std\_by\_intensity\_slicing\_transformation.png
- lena\_std\_by\_intensity\_to\_color\_transformation.png

### ● Description:

Python 源代码实现了灰度分层算法及灰度至彩色算法两种伪彩色变换算法。灰度分层算法对灰度值进行阈值分层，其变换函数为阶梯函数；灰度至彩色算法以灰度值作为函数输入，使用三个通道函数对 RGB 值进行映射，其变换函数为分段线性函数。实际操作中，灰度至彩色算法的映射函数可以设计为非线性函数以适应更复杂的需求。

本次作业代码中，灰度分层算法实现了 12 个分层，灰度至彩色算法实现了 5 个分段，具体设计请查阅代码。

### ● Source Code:

```
import os # 文件
import time # 时间

import numpy as np # 数组
from PIL import Image, ImageColor # 图像

HOME_PATH = os.getcwd()
DATA_PATH = os.path.join(HOME_PATH, 'data')
RESULT_PATH = os.path.join(HOME_PATH, 'result')

class ImageData:
    # 初始化
    def __init__(self, image_name):
        self.image_name = image_name # 图像的名称

    # 图像输入
    def input_image(self):
        image = Image.open(os.path.join(DATA_PATH, self.image_name)) # 输入图像
        self.full_image_matrix = np.array(image.convert('RGB')) # 图像的全彩色矩阵
        self.gray_image_matrix = np.array(image.convert('L')) # 图像的灰度矩阵
        self.pseudo_image_matrix = np.zeros(self.full_image_matrix.shape, dtype=np.uint8) # 图像的伪彩色矩阵
        self.image_height, self.image_width = image.size # 图像的高与宽
```

```

# 图像输出
def output_image(self):
    canvas = Image.new(mode='RGB', size=(self.image_width * 3 + 64, self.image_height + 32), color="#D7D7D7") # 生成空白画布
    canvas.paste(im=Image.fromarray(self.full_image_matrix), box=(16, 16)) # 生成全彩色图像(原图)
    canvas.paste(im=Image.fromarray(self.gray_image_matrix), box=(self.image_width + 32, 16)) # 生成灰度图像
    canvas.paste(im=Image.fromarray(self.pseudo_image_matrix), box=(self.image_width * 2 + 48, 16)) # 生成伪彩色图像
    canvas.save(os.path.join(RESULT_PATH, '%s_by_%s_transformation.png' % (self.image_name.split('.')[0], self.algorithm))) # 保存图像
    # canvas.show() # 显示图像

# 灰度分层
def intensity_slicing(self):
    for h in range(self.image_height):
        for w in range(self.image_width):
            intensity = self.gray_image_matrix[h, w]
            if 0 <= intensity <= 18:
                self.pseudo_image_matrix[h, w, :] = np.array([127, 0, 255], dtype=np.uint8)
            elif 19 <= intensity <= 48:
                self.pseudo_image_matrix[h, w, :] = np.array([0, 0, 255], dtype=np.uint8)
            elif 49 <= intensity <= 62:
                self.pseudo_image_matrix[h, w, :] = np.array([0, 63, 255], dtype=np.uint8)
            elif 63 <= intensity <= 84:
                self.pseudo_image_matrix[h, w, :] = np.array([0, 127, 255], dtype=np.uint8)
            elif 85 <= intensity <= 106:
                self.pseudo_image_matrix[h, w, :] = np.array([0, 210, 210], dtype=np.uint8)
            elif 107 <= intensity <= 128:
                self.pseudo_image_matrix[h, w, :] = np.array([0, 255, 0], dtype=np.uint8)
            elif 129 <= intensity <= 150:
                self.pseudo_image_matrix[h, w, :] = np.array([127, 255, 0], dtype=np.uint8)
            elif 151 <= intensity <= 172:
                self.pseudo_image_matrix[h, w, :] = np.array([255, 255, 0], dtype=np.uint8)
            elif 173 <= intensity <= 194:
                self.pseudo_image_matrix[h, w, :] = np.array([255, 210, 0], dtype=np.uint8)
            elif 195 <= intensity <= 216:
                self.pseudo_image_matrix[h, w, :] = np.array([255, 165, 0], dtype=np.uint8)
            elif 217 <= intensity <= 238:
                self.pseudo_image_matrix[h, w, :] = np.array([255, 82, 0], dtype=np.uint8)
            elif 239 <= intensity < 256:
                self.pseudo_image_matrix[h, w, :] = np.array([255, 0, 0], dtype=np.uint8)

# 灰度彩色变换
def intensity_to_color(self):
    for h in range(self.image_height):
        for w in range(self.image_width):
            intensity = self.gray_image_matrix[h, w]
            if 0 <= intensity < 42:
                self.pseudo_image_matrix[h, w, :] = np.array([-3 * intensity + 128, 0, 255], dtype=np.uint8)
            elif 42 <= intensity < 85:
                self.pseudo_image_matrix[h, w, :] = np.array([0, 6 * intensity - 252, 255], dtype=np.uint8)
            elif 85 <= intensity < 128:
                self.pseudo_image_matrix[h, w, :] = np.array([0, 255, -6 * intensity + 765], dtype=np.uint8)
            elif 128 <= intensity < 170:
                self.pseudo_image_matrix[h, w, :] = np.array([6 * intensity - 768, 255, 0], dtype=np.uint8)
            elif 170 <= intensity < 256:
                self.pseudo_image_matrix[h, w, :] = np.array([255, -3 * intensity + 765, 0], dtype=np.uint8)

# 伪彩色变换
def pseudo_color_transformation(self, alg):
    self.input_image() # 输入图像
    self.algorithm = alg # 伪彩色变换算法
    getattr(self, alg)() # 伪彩色变换
    self.output_image() # 输出图像

# 测试接口
def test(self, alg):
    print('\nImage: %s\n' % self.image_name) # 图像名称
    start = time.time() # 开始测试时刻
    self.pseudo_color_transformation(alg) # 使用伪彩色变换算法处理图像
    end = time.time() # 结束测试时刻
    print('Algorithm: %s transformation %s join(alg.split("."))) # 变换算法
    print('Program execute time: %.2f s\n' % (end - start)) # 测试时长

if __name__ == '__main__':
    # 图像: NGC2237.jpg, 算法: 灰度分层
    ImageData('NGC2237.jpg').test(alg='intensity_slicing')
    # 图像: NGC2237.jpg, 算法: 灰度彩色变换
    ImageData('NGC2237.jpg').test(alg='intensity_to_color')

    # 图像: lena_std.tif, 算法: 灰度分层
    ImageData('lena_std.tif').test(alg='intensity_slicing')
    # 图像: lena_std.tif, 算法: 灰度彩色变换
    ImageData('lena_std.tif').test(alg='intensity_to_color')

```

## ● Results:

为了测试伪彩色算法效果，实验采用了两幅图像进行测试。实验时先将原图转换为灰度图像，再使用伪彩色算法对灰度图进行变换。



对 NGC2237.jpg 使用灰度分层算法



对 NGC2237.jpg 使用灰度至彩色算法

## 结果分析：

灰度分层算法中，红端颜色代表高灰度，蓝端颜色代表低灰度。灰度图的主体为星云，星云中较亮的部分往往是因恒星聚集而形成，观察伪彩色图像可得，中上部的红色区域代表密集的恒星集团，而中部的橙色和绿色区域代表较为稀疏的气体云，背景的蓝色区域代表背景宇宙。相较于灰度图，伪彩色图更容易被识别。灰度至彩色算法的结果图和灰度分层算法类似，不同的是下部的外围气体云细节更明显，这是因为映射函数将不同灰度值映射至不同 RGB 值而非简单的分层。灰度至彩色算法更有助于辨别星云的外围气体结构和边界区域。



对 lena\_std.tif 使用灰度分层算法



对 lena\_std.tif 使用灰度至彩色算法

结果分析：

将伪彩色算法应用至人像时，红端颜色代表人像中的高光部分，蓝端颜色代表人像中的阴影部分。灰度分层算法中，由于灰度等量分层，伪彩色图中不同层的像素数量较为均衡，可见结果图中面部的轮廓较为明确，结构更清晰。而灰度至彩色算法采用连续的映射函数，且伪彩色图中的大部分像素集中在绿色调附近，可以观察到图像整体偏绿，面部轮廓和背景结构容易混淆。但与灰度分层算法相比，灰度至彩色算法结果图背景的噪点更少，图像整体较为柔和，而灰度分层算法结果图的锐度较高，噪点也比较多。

2 使用世界各国 GDP 总量数据：(1) 使用折线图及散点图制作可视化图，显示世界各国 20 年的 GDP 数值；(2) 使用地图制作可视化图，显示世界各国 GDP 在 20 年来的动态变化。

● **Python script code:**

2\_1\_GDP\_stat.py

2\_2\_GDP\_map.py

● **Input & Output:**

**Input:** GDP.csv

**Output:**

Part 1:

- GDP\_stat\_line.png
- GDP\_stat\_line\_log.png
- GDP\_stat\_line\_except\_USA\_and\_China.png
- GDP\_stat\_scatter.png
- GDP\_stat\_scatter\_log.png
- GDP\_stat\_scatter\_except\_USA\_and\_China.png

Part 2:

- GDP\_map.html
- GDP\_map.mp4 (后期制作)
- GDP\_map.gif (后期制作)

● **Description:**

第一部分使用折线图及散点图制作 GDP 总量的可视化图，采用的技术为 matplotlib，选取的数据是 2016 年 GDP 总量前 20 位国家的 GDP 总量数据。作图后观察到中美两国的 GDP 总量较其他国家高出很多，故再尝试可视化除中美两国外的 GDP 总量走势图，另一方面考虑到数值的相似性，最后使用了对数坐标轴增大数值的纵向差异以优化可视化效果。

第二部分实现了基于地图的时间轴轮播图，采用的技术为 pyecharts，选取的数据是 2016 年 GDP 总量前 50 位国家的 GDP 总量数据。图中使用 GDP 数值的对数进行衡量，按照数值大小按顺序共分为 8 层，浅色代表 GDP 总量低，深色代表 GDP 总量高。因仅采用了 50 个国家的数据进行可视化，地图中没有数据的国家呈灰色。制作轮播图时，每个画面停留的时间为 1 秒，可视化交互界面可查看 html 文件，轮播图动态结果可查看 gif 动图或 mp4 视频。

## ● Source Code:

### Part 1:

```
import csv # 数据
import os # 文件

import matplotlib.pyplot as plt # 绘图
import numpy as np # 数组

# 路径
HOME_PATH = os.getcwd() # 主目录
DATA_PATH = os.path.join(HOME_PATH, 'data') # 数据目录
RESULT_PATH = os.path.join(HOME_PATH, 'result') # 结果目录

# 颜色
COLOR = [
    '#FF0000', # 红色 (255, 0, 0)
    '#FF5200', # 深橙色 (255, 82, 0)
    '#FFA500', # 橙色 (255, 165, 0)
    '#00FF00', # 绿色 (0, 255, 0)
    '#00D2D2', # 青色 (0, 210, 210)
    '#006400', # 深绿色 (0, 100, 0)
    '#64D2FF', # 淡蓝色 (100, 210, 255)
    '#007FFF', # 天蓝色 (0, 127, 255)
    '#0000FF', # 蓝色 (0, 0, 255)
    '#808080', # 灰色 (128, 128, 128)
]

# 标记
MARKER = [
    'o', # 圆形
    'v', # 倒三角形
    's', # 正方形
    '^', # 正三角形
    'd', # 菱形
    '<', # 左三角形
    'd', # 窄菱形
    '>', # 右三角形
    'x', # 十字形
    'p', # 正五边形
]

# 数据数量
NUM_COUNTRIES = 10 # 1 <= NUM_COUNTRIES <= 10

# 数据导入
with open(os.path.join(DATA_PATH, 'GDP.csv'), 'r') as csv_file: # 打开csv文件
    csv_reader = csv.reader(csv_file, delimiter=',') # 读取csv文件
    csv_reader.__next__() # 跳过首行标题行
    data = np.array([next(csv_reader) for _ in range(NUM_COUNTRIES)]) # 提取前NUM_COUNTRIES个国家的数据

# 年份
YEAR = [year for year in range(1996, 2017)] # 1996-2016

# 国家
COUNTRY = data[:, 0]

# GDP
GDP = data[:, 2:].astype(np.float) / 10e12 # 单位为trillion

# 折线图
def plot_line_chart():
    plt.figure(num=1, figsize=(10, 8)) # 创建新图片
    for rank in range(NUM_COUNTRIES):
        plt.plot(
            YEAR, # 年份
            GDP[rank], # GDP总量
```

```

        label=COUNTRY[rank], # 国家
        color=COLOR[rank], # 颜色
        marker=MARKER[rank], # 标记
        markersize=6, # 标记大小
        linestyle='-', # 折线形式
        linewidth=1 # 折线粗细
    )
plt.title('Country GDP 1996-2016') # 标题
plt.xlabel('Year') # X轴标签
plt.ylabel('GDP (Trillions of US$)') # Y轴标签
plt.grid(ls='--') # 背景网格
plt.xticks(ticks=YEAR[::5]) # X轴刻度
plt.yticks(
    ticks=[0, 0.25, 0.5, 0.75, 1, 1.25, 1.5, 1.75], # Y轴刻度位置
    labels=[0, 0.25, 0.5, 0.75, 1, 1.25, 1.5, 1.75] # Y轴刻度标签
) # Y轴刻度
plt.legend(loc='best') # 图例
plt.savefig(os.path.join(RESULT_PATH, 'GDP_stat_line.png')) # 保存折线图
# plt.show() # 显示折线图

# 折线图（对数坐标）
def plot_line_log_chart():
    _, axis = plt.subplots(num=3, figsize=(10, 8)) # 创建新图片
    axis.set_yscale('log') # 对数坐标
    for rank in range(NUM_COUNTRIES):
        plt.plot(
            YEAR, # 年份
            GDP[rank], # GDP总量
            label=COUNTRY[rank], # 国家
            color=COLOR[rank], # 颜色
            marker=MARKER[rank], # 标记
            markersize=6, # 标记大小
            linestyle='-', # 折线形式
            linewidth=1 # 折线粗细
        )
    plt.title('Country GDP 1996-2016') # 标题
    plt.xlabel('Year') # X轴标签
    plt.ylabel('GDP (Trillions of US$)') # Y轴标签
    plt.grid(ls='--', which='both') # 背景网格
    plt.xticks(ticks=YEAR[::5]) # X轴刻度
    plt.yticks(
        ticks=[0.05, 0.1, 0.5, 1, 2], # Y轴刻度位置
        labels=[0.05, 0.1, 0.5, 1, 2] # Y轴刻度标签
    ) # Y轴刻度
    plt.legend(loc='best') # 图例
    plt.savefig(os.path.join(RESULT_PATH, 'GDP_stat_line_log.png')) # 保存折线图
# plt.show() # 显示折线图

# 折线图（除中美）
def plot_line_chart_except_USA_and_China():
    plt.figure(num=5, figsize=(10, 8)) # 创建新图片
    for rank in range(2, NUM_COUNTRIES):
        plt.plot(
            YEAR, # 年份
            GDP[rank], # GDP总量
            label=COUNTRY[rank], # 国家
            color=COLOR[rank], # 颜色
            marker=MARKER[rank], # 标记
            markersize=6, # 标记大小
            linestyle='-', # 折线形式
            linewidth=1 # 折线粗细
        )
    plt.title('Country GDP 1996-2016') # 标题
    plt.xlabel('Year') # X轴标签
    plt.ylabel('GDP (Trillions of US$)') # Y轴标签
    plt.grid(ls='--') # 背景网格
    plt.xticks(ticks=YEAR[::5]) # X轴刻度
    plt.legend(loc='best', ncol=2) # 图例
    plt.savefig(os.path.join(RESULT_PATH, 'GDP_stat_line_except_USA_and_China.png')) # 保存折线图
# plt.show() # 显示折线图

```

```

# 散点图
def plot_scatter_chart():
    plt.figure(num=2, figsize=(10, 8)) # 创建新图片
    for rank in range(NUM_COUNTRIES):
        plt.plot(
            YEAR, # 年份
            GDP[rank], # GDP总量
            label=COUNTRY[rank], # 国家
            color=COLOR[rank], # 颜色
            marker=MARKER[rank], # 标记
            markersize=6, # 标记大小
            linestyle='.' # 散点形式
        )
    plt.title('Country GDP 1996-2016') # 标题
    plt.xlabel('Year') # X轴标签
    plt.ylabel('GDP (Trillions of US$)') # Y轴标签
    plt.grid(ls='--') # 背景网格
    plt.xticks(ticks=YEAR[::5]) # X轴刻度
    plt.yticks(
        ticks=[0, 0.25, 0.5, 0.75, 1, 1.25, 1.5, 1.75], # Y轴刻度位置
        labels=[0, 0.25, 0.5, 0.75, 1, 1.25, 1.5, 1.75] # Y轴刻度标签
    ) # Y轴刻度
    plt.legend(loc='best') # 图例
    plt.savefig(os.path.join(RESULT_PATH, 'GDP_stat_scatter.png')) # 保存折线图
    # plt.show() # 显示折线图

# 散点图（对数坐标）
def plot_scatter_log_chart():
    _, axis = plt.subplots(num=4, figsize=(10, 8)) # 创建新图片
    axis.set_yscale('log') # 对数坐标
    for rank in range(NUM_COUNTRIES):
        plt.plot(
            YEAR, # 年份
            GDP[rank], # GDP总量
            label=COUNTRY[rank], # 国家
            color=COLOR[rank], # 颜色
            marker=MARKER[rank], # 标记
            markersize=6, # 标记大小
            linestyle='.' # 散点形式
        )
    plt.title('Country GDP 1996-2016') # 标题
    plt.xlabel('Year') # X轴标签
    plt.ylabel('GDP (Trillions of US$)') # Y轴标签
    plt.grid(ls='--', which='both') # 背景网格
    plt.xticks(ticks=YEAR[::5]) # X轴刻度
    plt.yticks(
        ticks=[0.05, 0.1, 0.5, 1, 2], # Y轴刻度位置
        labels=[0.05, 0.1, 0.5, 1, 2] # Y轴刻度标签
    ) # Y轴刻度
    plt.legend(loc='best') # 图例
    plt.savefig(os.path.join(RESULT_PATH, 'GDP_stat_scatter_log.png')) # 保存折线图
    # plt.show() # 显示折线图

# 散点图（除中美）
def plot_scatter_chart_except_USA_and_China():
    plt.figure(num=6, figsize=(10, 8)) # 创建新图片
    for rank in range(2, NUM_COUNTRIES):
        plt.plot(
            YEAR, # 年份
            GDP[rank], # GDP总量
            label=COUNTRY[rank], # 国家
            color=COLOR[rank], # 颜色
            marker=MARKER[rank], # 标记
            markersize=6, # 标记大小
            linestyle='.' # 散点形式
        )
    plt.title('Country GDP 1996-2016') # 标题
    plt.xlabel('Year') # X轴标签
    plt.ylabel('GDP (Trillions of US$)') # Y轴标签

```

```

plt.grid(ls='--') # 背景网格
plt.xticks(ticks=YEAR[::5]) # X轴刻度
plt.legend(loc='best', ncol=2) # 图例
plt.savefig(os.path.join(RESULT_PATH, 'GDP_stat_scatter_except_USA_and_China.png')) # 保存折线图
# plt.show() # 显示折线图

if __name__ == "__main__":
    plot_line_chart() # 折线图
    plot_line_log_chart() # 折线图（对数坐标）
    plot_line_chart_except_USA_and_China() # 折线图（除中美）
    plot_scatter_chart() # 散点图
    plot_scatter_log_chart() # 散点图（对数坐标）
    plot_scatter_chart_except_USA_and_China() # 散点图（除中美）

```

## Part 2:

```

import os # 文件

import numpy as np # 数组
import pandas as pd # 数组
import pyecharts.options as opts # 选项
from pyecharts.charts import Map, Timeline # 图表
from pyecharts.globals import ThemeType # 主题

# 路径设置
HOME_PATH = os.getcwd() # 主目录
DATA_PATH = os.path.join(HOME_PATH, 'data') # 数据目录
RESULT_PATH = os.path.join(HOME_PATH, 'result') # 结果目录

# 数据导入
GDP = pd.read_csv(os.path.join(DATA_PATH, 'GDP.csv'), index_col='Country') # GDP数据
YEAR = GDP.columns.tolist()[1:] # 年份
COUNTRY = GDP.index.tolist() # 国家
MIN_GDP = np.log(GDP.iloc[:, 1:].min().min()) # GDP最大值的对数
MAX_GDP = np.log(GDP.iloc[:, 1:].max().max()) # GDP最小值的对数

# 按年份添加图层
def add_chart_by_year(year):

    map_chart = (
        Map() # 地图图层
        .add(
            series_name='', # 系列名称
            data_pair=list(zip(COUNTRY, np.log(GDP[year]))), # 数据项
            maptype='world', # 地图类型
            is_roam=False, # 禁止缩放
            is_map_symbol_show=False, # 不显示标记
            label_opts=opts.LabelOpts(is_show=False), # 不显示标签
        ) # 图层设置
        .set_global_opts(
            title_opts=opts.TitleOpts(
                title='Variation of World GDP (1996-2016)', # 标题
                pos_left='center', # 标题水平位置
                pos_top='60px', # 标题竖直位置
                title_textstyle_opts=opts.TextStyleOpts(
                    color='#000000', # 字体颜色
                    font_style='normal', # 字体风格
                    font_size=25, # 字体大小
                ), # 标题字体样式
        ),
    ),

```

```

        visualmap_opts=opts.VisualMapOpts(
            min_=MIN_GDP, # 图例最小值
            max_=MAX_GDP, # 图例最大值
            range_text=['<log(US$)'], # 图例标识
            range_color=[
                '#FACDAA', # 浅 <-> 小
                '#F4A49E',
                '#EE7B91',
                '#E85285',
                '#BE408C',
                '#942D93',
                '#6A1B9A',
                '#56167D', # 深 <-> 大
            ], # 图例过渡颜色
            pos_left='190px', # 图例水平位置
            pos_top='380px', # 图例竖直位置
            is_piecewise=True, # 图例分段
            split_number=8, # 图例分段数量
            pieces=[
                {'min': 29.5, 'label': '> 29.5'},
                {'min': 28.7, 'max': 29.5, 'label': '28.7 - 29.5'},
                {'min': 27.9, 'max': 28.7, 'label': '27.9 - 28.7'},
                {'min': 27.1, 'max': 27.9, 'label': '27.1 - 27.9'},
                {'min': 26.3, 'max': 27.1, 'label': '26.3 - 27.1'},
                {'min': 25.5, 'max': 26.3, 'label': '25.5 - 26.3'},
                {'min': 24.7, 'max': 25.5, 'label': '24.7 - 25.5'},
                {'max': 24.7, 'label': '< 24.8'},
            ], # 图例分段标签
            border_color='#000000', # 边框颜色
            border_width=1, # 边框宽度
            textstyle_opts=opts.TextStyleOpts(
                color='#000000', # 字体颜色
                font_style='normal', # 字体风格
                font_weight='bold', # 字体粗细
                font_size=12, # 字体大小
            ), # 图例字体样式
        ),
    ), # 全局设置
)

return map_chart # 返回图层
}

if __name__ == '__main__':
    timeline = Timeline(init_opts=opts.InitOpts(
        height='750px', # 画布高度
        width='1125px', # 画布宽度
        theme=ThemeType.LIGHT, # 画布主题
    )) # 实例化时间轴轮播图

    for year in YEAR:
        timeline.add(chart=add_chart_by_year(year), time_point=year) # 按年份添加图层

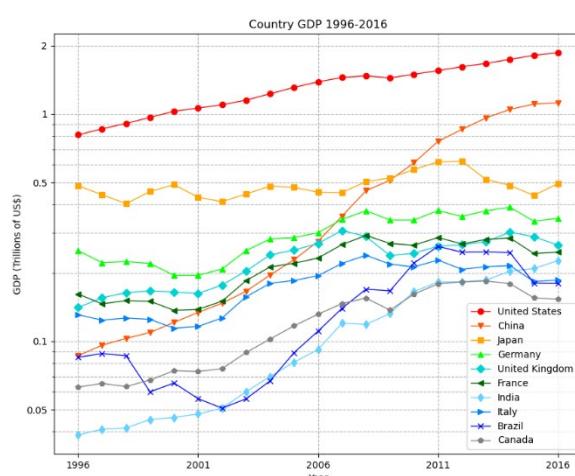
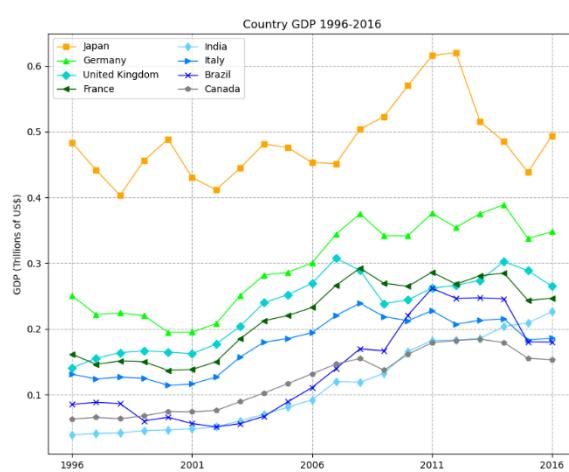
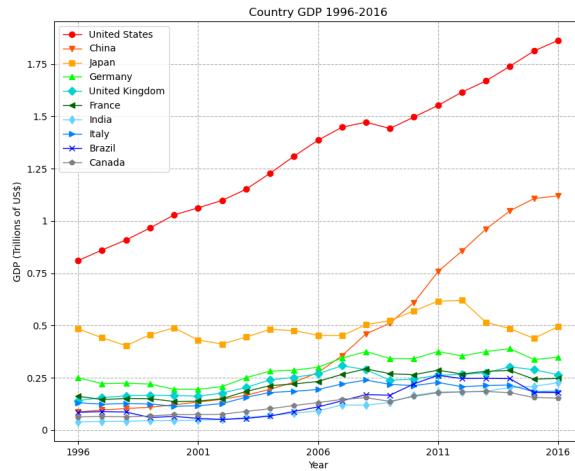
    timeline.add_schema(
        is_auto_play=True, # 自动播放
        play_interval=1000, # 播放速度
        pos_left='center', # 时间轴水平位置
        pos_bottom='60px', # 时间轴竖直位置
        width='800px', # 时间轴长度
        label_opts=opts.LabelOpts(is_show=True), # 显示时间轴标签
    ) # 时间轴设置

    timeline.render(path=os.path.join(RESULT_PATH, 'GDP_map.html')) # 渲染动图

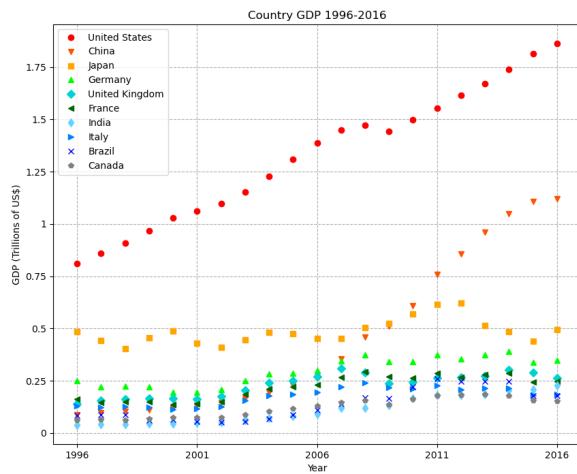
```

## ● Results:

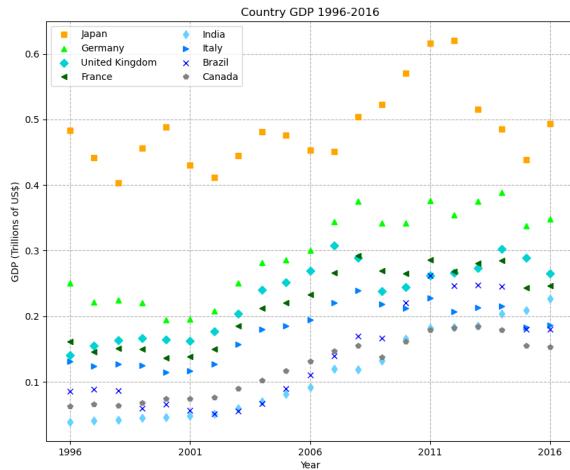
### Part 1: 折线图



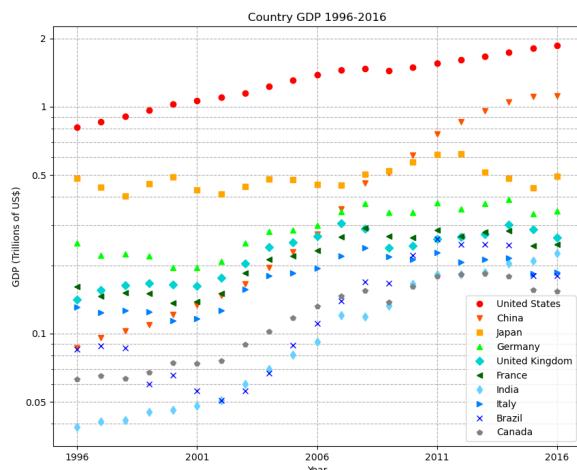
## Part 1: 散点图（折线图去除折线的效果）



散点图

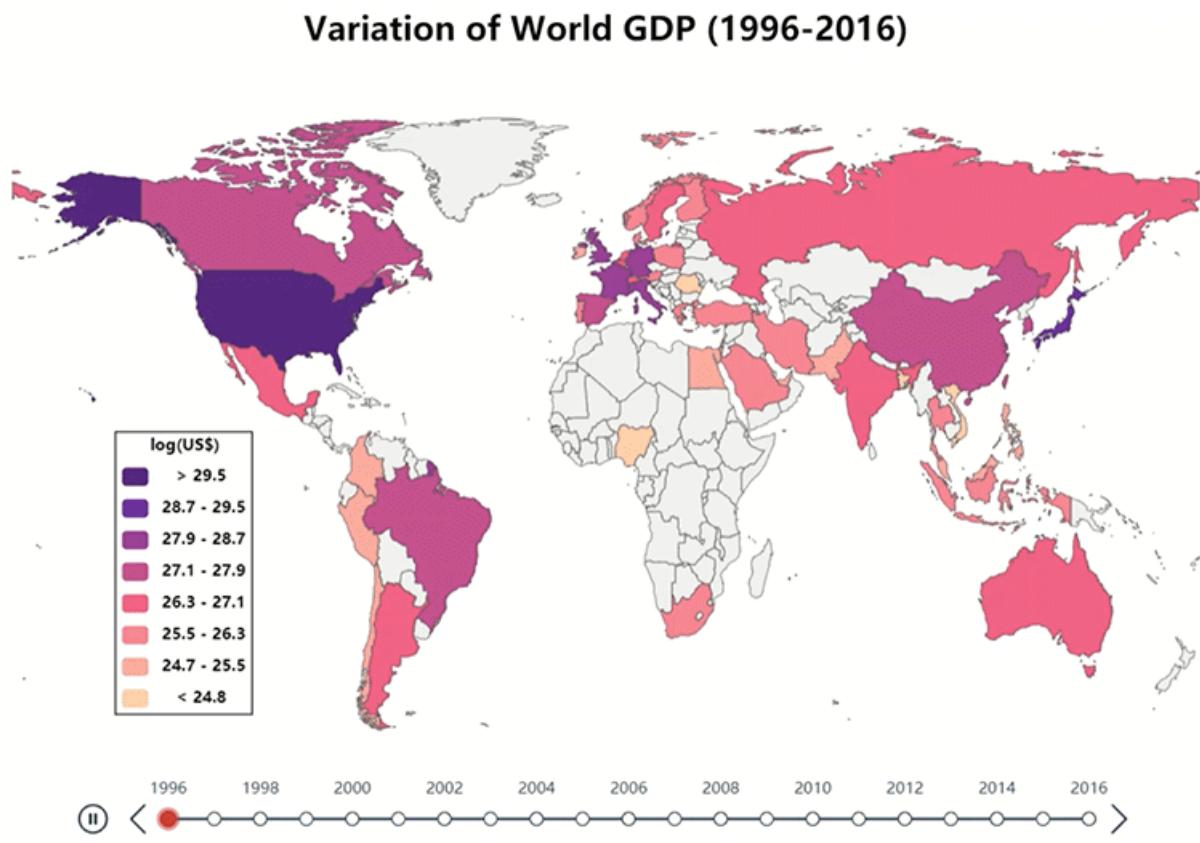


散点图（除中美）



散点图（对数坐标）

## Part 2: 时间轴轮播图 (仅展示首帧)



结果分析：

第一部分中，对比散点图和折线图，可以发现折线图的时间连续性更强，其可视化效果更好。由于折线图更容易被看出变化趋势，在可视化时间序列的数据时一般采用折线图。对比结果还可以发现，对数坐标扩大了数值的差异，可视化效果更明显。

第二部分中，由于各国 GDP 总量差异较大，若使用原始值作为指标进行颜色可视化会导致大部分数据聚集在浅颜色处，可视化效果较差。因此，需要使用取对数的方式优化可视化效果。当轮播速度提升时，可以更容易地看到各个国家在 20 年间 GDP 总量的大致趋势变化。

3 使用地震数据，采用地图可视化方法对数据进行可视化，展现地震的地点。

#### ● Python script code:

3\_quakes.py

- **Input & Output:**

**Input:** quakes.csv

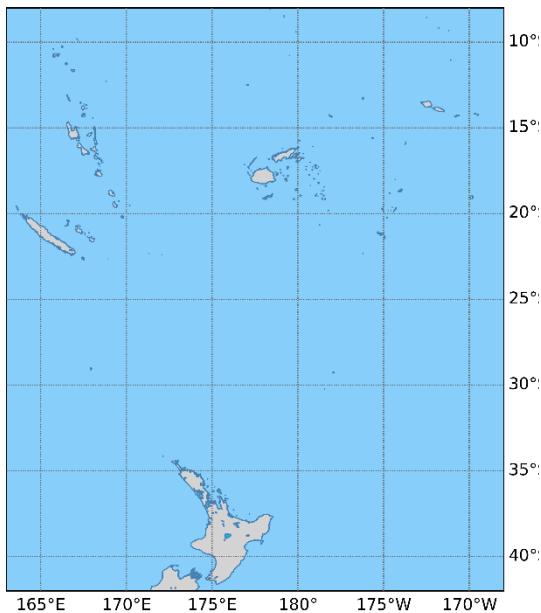
**Output:** quakes.png

- **Description:**

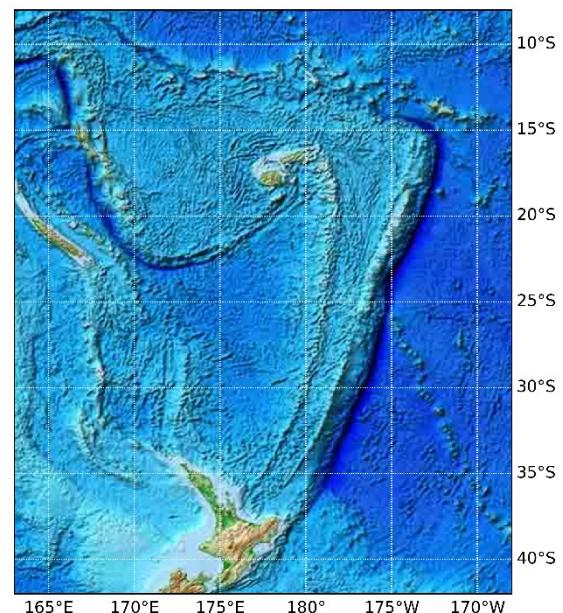
该部分实现了基于地图的地理信息可视化，采用的数据集为南太平洋斐济附近海域自 1964 年以来震级超过 4.0 级的地震信息，包括了经纬度、震源深度、地震等级以及观测站等信息，使用的技术为 basemap。数据集一共包含 1000 条数据，经纬度位于南太平洋局部海域，地震等级区间为 4.0-6.4，震源深度从 100 千米至 600 千米不等。实现震源位置可视化时，采用标记大小记录地震等级，采用颜色深度记录震源深度。对于地震等级，标记越大表示地震等级越高；对于震源深度，颜色越深表示震源深度越深。

为了更好地进行地震信息可视化，开始可视化前先观察经纬度的地理信息，如下图所示。可以发现该数据集记录的震源位置属于南太平洋附近海域，观察地形图可以发现，该区域属于地震带，结合地震数据，可以分析出更有用的地理信息。

**Map near Fiji**



**Map near Fiji**



## ● Source Code:

```
import os # 文件

import matplotlib.colors as colors # 颜色
import matplotlib.pyplot as plt # 绘图
import numpy as np # 数组
import pandas as pd # 数组
from mpl_toolkits.basemap import Basemap # 地图

# 路径
HOME_PATH = os.getcwd() # 主目录
DATA_PATH = os.path.join(HOME_PATH, 'data') # 数据目录
RESULT_PATH = os.path.join(HOME_PATH, 'result') # 结果目录

# 颜色
SunsetOrange = colors.LinearSegmentedColormap.from_list(
    name='SunsetOrange',
    colors=[
        '#FDEDBE', # 浅 <-> 小
        '#FFDF80',
        '#FFCB33',
        '#FFB200',
        '#FF8C00',
        '#FF6500',
        '#E6450F',
        '#B22C00',
        '#661900', # 深 <-> 大
    ],
) # 橙色系
LeafYellow = colors.LinearSegmentedColormap.from_list(
    name='LeafYellow',
    colors=[
        '#FFEBC0', # 浅 <-> 小
        '#FFDF80',
        '#FACA3E',
        '#E6B80B',
        '#B5AC23',
        '#6A9A48',
        '#20876B',
        '#06746B',
        '#044E48', # 深 <-> 大
    ],
) # 绿色系
GeekBlue = colors.LinearSegmentedColormap.from_list(
    name='GeekBlue',
    colors=[
        '#D2EDC8', # 浅 <-> 小
        '#A9DACC',
        '#75C6D1',
        '#42B3D5',
        '#3993C2',
        '#3073AE',
        '#27539B',
    ],
)
```

```

        '#1E3388',
        '#171E6D', # 深 <-> 大
    ],
) # 蓝色系
GoldenPurple = colors.LinearSegmentedColormap.from_list(
    name='GoldenPurple',
    colors=[
        '#FACDAE', # 浅 <-> 小
        '#F4A49E',
        '#EE7B91',
        '#E85285',
        '#BE408C',
        '#942D93',
        '#6A1B9A',
        '#56167D',
        '#42105F', # 深 <-> 大
    ],
) # 紫色系
COLOR = GeekBlue # 选择颜色

# 数据导入
NUM_QUAKES = 1000 # 数据数量
QUAKE = pd.read_csv(
    os.path.join(DATA_PATH, 'quakes.csv'), # 读取文件
    index_col=0, # 首列设为序号
    nrows=NUM_QUAKES, # 读取数量
) # 地震数据
LATITUDE = QUAKE['lat'] # 纬度
LONGITUDE = QUAKE['long'] # 经度
DEPTH = QUAKE['depth'] # 震源深度
MAGNITUDE = QUAKE['mag'] # 地震等级

if __name__ == "__main__":
    # 初始化画布
    plt.figure(figsize=(18, 18)) # 画布大小

    # 初始化地图
    fiji_map = Basemap(
        llcrnrlat=-42, # 地图下边界纬度: 42°S
        llcrnrlon=163, # 地图左边界经度: 163°E
        urcrnrlat=-8, # 地图上边界纬度: 8°S
        urcrnrlon=192, # 地图右边界经度: 172°W
        projection='cyl', # 投影方式: 等距圆柱投影
        resolution='f', # 分辨率: 最高
    )

    # 绘制线
    fiji_map.drawcoastlines(
        color='steelblue', # 海岸线颜色
        linewidth=1.8, # 海岸线宽度
    ) # 海岸线

```

```

fiji_map.drawcountries(
    color='darkgrey', # 国境线颜色
    linewidth=1.5, # 国境线宽度
) # 国境线
fiji_map.drawparallels(
    circles=np.arange(-40.0, 5.0, 5.0), # 纬线间隔: 5°
    color='dimgrey', # 纬线颜色
    linewidth=2, # 纬线宽度
    labels=[False, True, False, False], # 纬线刻度: 右边界
    fontsize=28, # 刻度大小
    zorder=20, # 图层位置: 最顶层
) # 纬线
fiji_map.drawmeridians(
    meridians=np.arange(165.0, 195.0, 5.0), # 经线间隔: 5°
    color='dimgrey', # 经线颜色
    linewidth=2, # 经线宽度
    labels=[False, False, False, True], # 经线刻度: 下边界
    fontsize=28, # 刻度大小
    zorder=20, # 图层位置: 最顶层
) # 经线

# 绘制面
fiji_map.fillcontinents(
    color='lightgrey', # 陆地
    lake_color='deepskyblue', # 湖泊
) # 前景
fiji_map.drawmapboundary(
    color='black', # 边框颜色
    linewidth=2, # 边框宽度
    fill_color='lightskyblue', # 海洋
) # 背景

# 绘制地震数据
quakes_position = fiji_map.scatter(
    x=LONGITUDE, # 经度
    y=LATITUDE, # 纬度
    s=np.power(10, MAGNITUDE * 0.7), # 地震等级
    c=DEPTH, # 震源深度
    marker='.', # 震源标记
    cmap=COLOR, # 震源深度图例颜色
    alpha=0.75, # 震源标记透明度
    linewidths=1.5, # 震源标记边缘线宽度
    edgecolors='white', # 震源标记边缘线颜色
    zorder=10, # 图层位置: 次顶层
) # 地震数据
color_bar = fiji_map.colorbar(
    mappable=quakes_position, # 映射对象: 地震数据
    location='left', # 图例位置: 左边界
    size='6%', # 图例宽度
    pad='2%', # 图例距离
) # 震源深度图例
color_bar.set_alpha(1) # 图例透明度

```

```

color_bar.draw_all() # 渲染图例
color_bar.outline.set_linewidth(2) # 图例边框宽度
color_bar.set_label(
    label='Focal Depth (km)', # 标签文本
    labelpad=-180, # 标签位置
    size=32, # 标签大小
) # 图例标签
color_bar.ax.tick_params(
    axis='y', # 刻度方向
    direction='inout', # 刻度线形式
    length=10, # 刻度线长度
    width=2, # 刻度线宽度
    pad=10, # 刻度位置
    labelsize=28, # 刻度大小
    left=True, # 左刻度线
    right=False, # 不显示右刻度线
    labelleft=True, # 左刻度
    labelright=False, # 不显示右刻度
) # 图例刻度

# 绘制文本
plt.title(
    label='Locations of Earthquakes near Fiji since 1964', # 标题文本
    fontsize=40, # 字体大小
    fontweight='bold', # 字体宽度
    pad=50, # 标题位置
) # 标题

# 保存地图
plt.savefig(
    fname=os.path.join(RESULT_PATH, 'quakes_%d.png' % NUM_QUAKES), # 文件名
    dpi=240, # 分辨率
) # 保存图像
# plt.show() # 显示图像

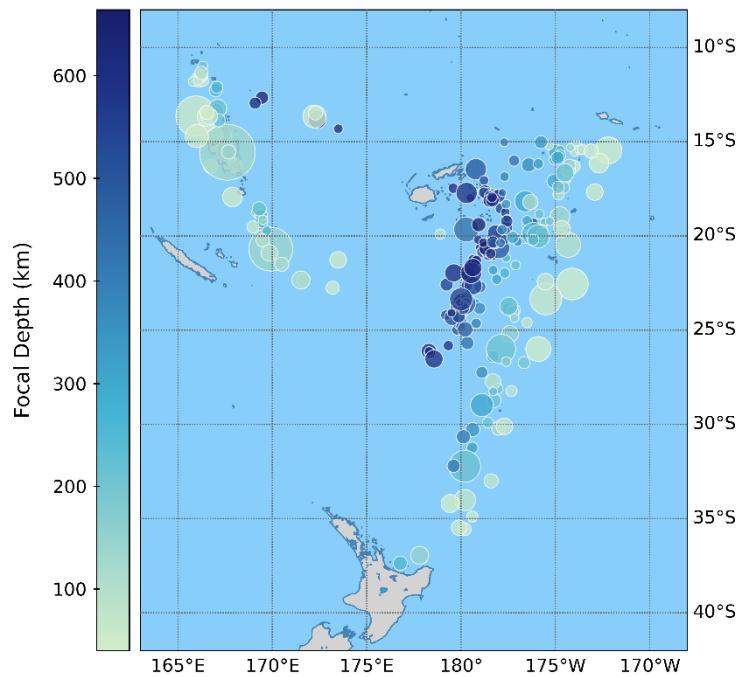
```

带浮雕地形图的实现代码与本部分代码类似，具体请查看所附代码。

### ● Results:

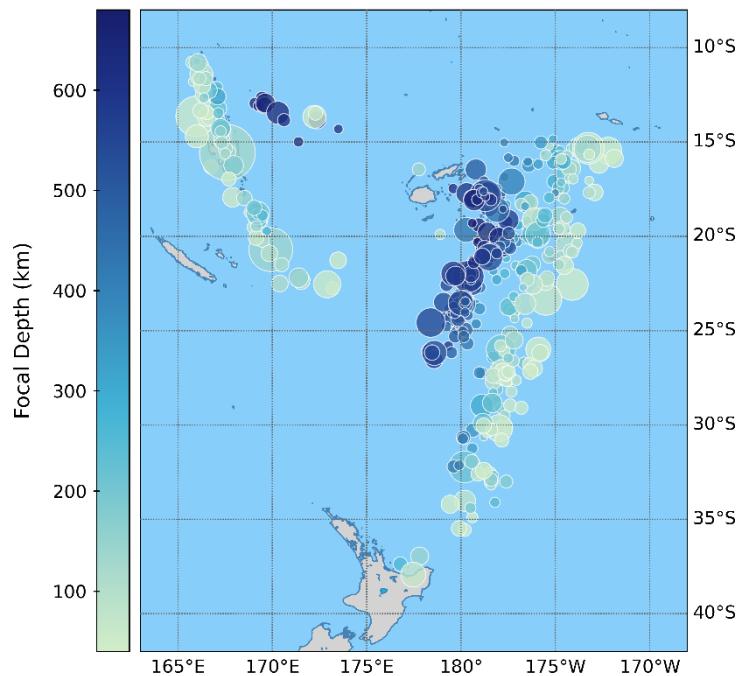
斐济附近海域的地震信息可视化结果如下所示。按照数据量不断增加的顺序观察，可以发现地震的位置始终位于特定的区域。加入地形图后，发现地震的位置大部分处于两条较深的海沟中，结合地震带的知识，可以认为图中两条较深的海沟位为两个活动大陆板块的交界，地震是因为大陆板块之间的碰撞而产生的，结合地形图可以证实这个推论。

### Locations of Earthquakes near Fiji since 1964



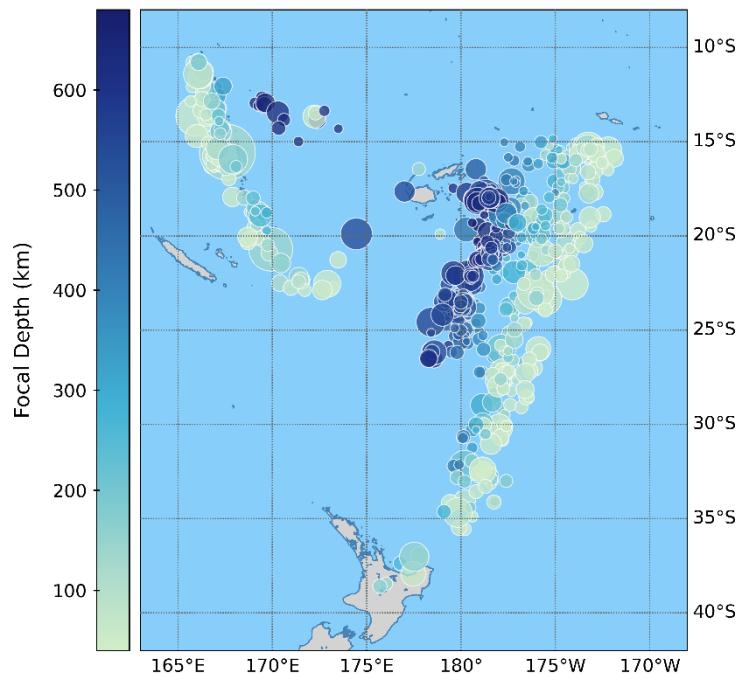
数据量： 250

### Locations of Earthquakes near Fiji since 1964



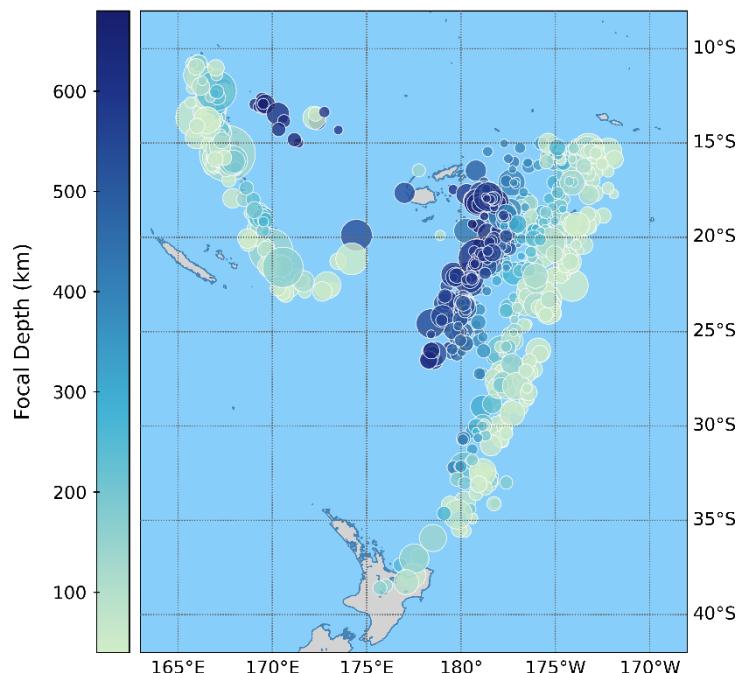
数据量： 500

### Locations of Earthquakes near Fiji since 1964



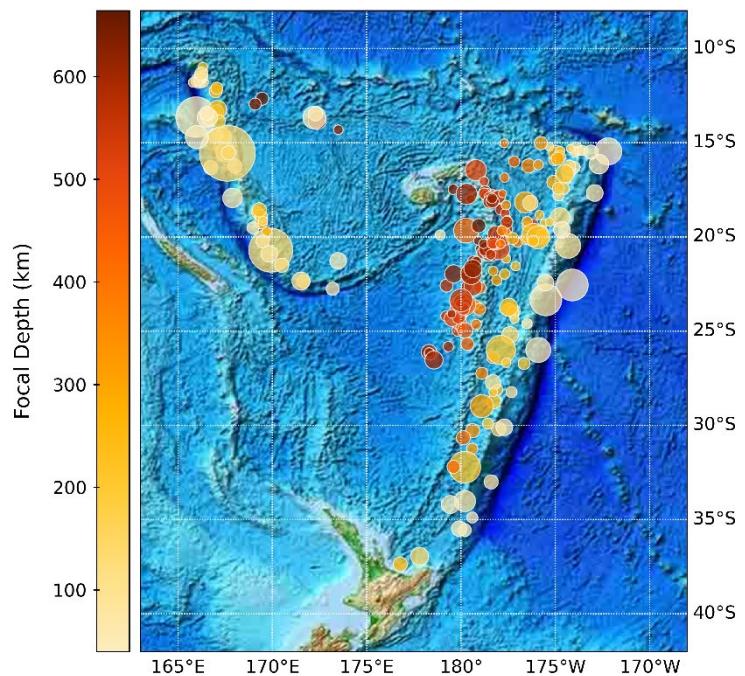
数据量：750

### Locations of Earthquakes near Fiji since 1964



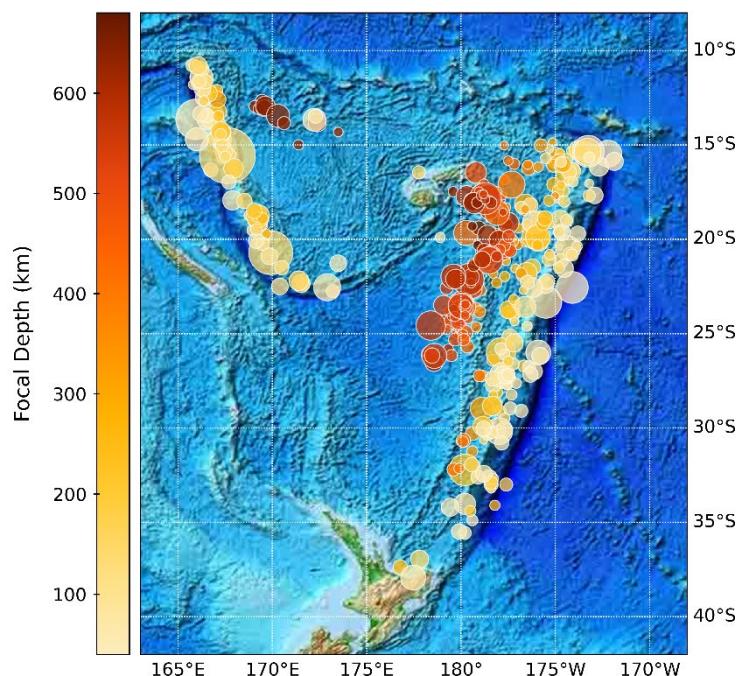
数据量：1000

### Locations of Earthquakes near Fiji since 1964



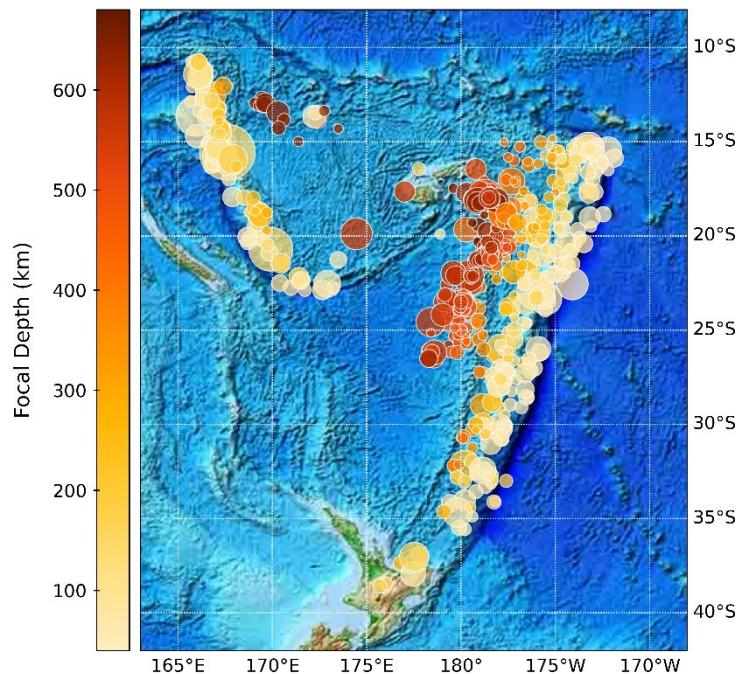
数据量：250

### Locations of Earthquakes near Fiji since 1964



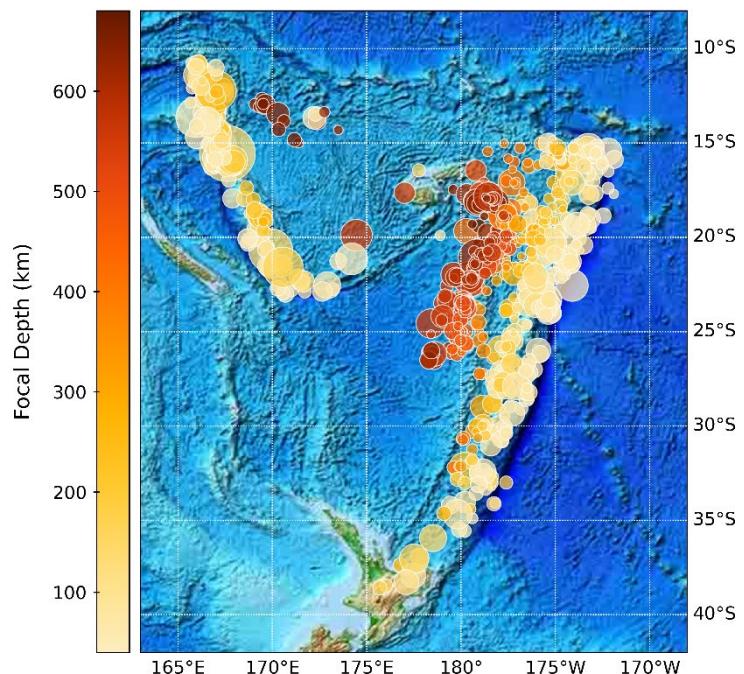
数据量：500

### Locations of Earthquakes near Fiji since 1964



数据量：750

### Locations of Earthquakes near Fiji since 1964



数据量：1000