第五次作业报告

邓文 17307130171

一、 设计灰度向彩色（伪彩）变换的算法、实现代码并用图片进行测试。

1. 算法的流程以及原理：

将灰度图像转换为伪彩色图像有助于图像的可视化。
伪颜色变换的**主要思想**是对灰度或强度图像进行红、绿、蓝三种独立的变换，将图像中相应的强度值映射到对应的结果。

主要流程：

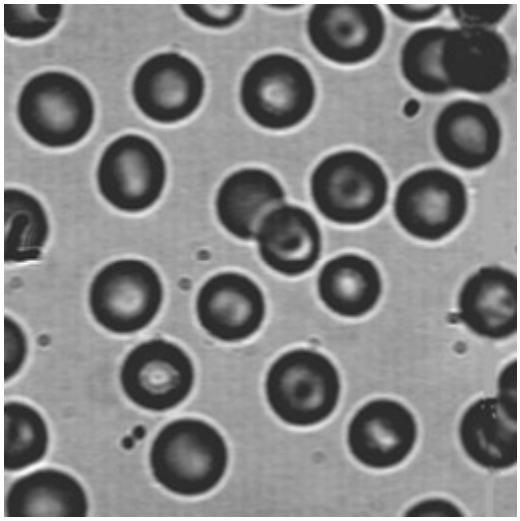
- 首先输入一个 W * H 大小的图像 A
- 然后要得到一个 N * 3 大小的 colormap， N 取决于图像 A 中可能的最大的灰度值(一般为 256)，而 colormap 的每一列对应了 Red， Greed， Blue 三个颜色通道。
- 然后再提前定义好一个 W * H * 3 的图像 B 用于储存输出的结果。
- 然后再根据每个点的像素值，将他的值映射到三个颜色通道中。
  例如， A 图像中 (2, 3) 坐标中的灰度值为 10。则在 colormap 中找到坐标（10, ）对应的三个颜色通道的值，(r, g, b)，然后将他们分别填入输出图像 B 中的 坐标 (2, 3, 0), (2, 3, 1), (2, 3, 2) 三个位置中。
- 上述过程完成后，将 B 转换为 unit8 的标准形式，然后输出并保存。

2. 具体的代码如下所示：

```python
def gray2color(img, colormap="rainbow", save=None):
    """
    该函数用于将 灰度图 向 彩色图(伪彩) 变换。
    算法
    :param save: 是否保存图片
    :param colormap: 用于灰度向彩色映射的分段颜色表，可以调用matplotlib中自带的各种colormap
    :param img: 输入的灰度图图像
    :return:  伪彩图像
    """
    colors = plt.get_cmap(colormap)(range(256))  # 从matplotlib库中获取对应的分段颜色表
    # 由于得到的分段颜色表是 RGBA 的形式，所以要将 RGBA 的形式转换为 RGB 形式
    red = colors[:, 0]
    green = colors[:, 1]
    blue = colors[:, 2]
    # 创建一个空的变量来存储 伪彩 后的图片
    pseudo_colored_img = np.zeros((img.shape[0], img.shape[1], 3))
    # 进行映射
    pseudo_colored_img[:, :, 0] = red[img]
    pseudo_colored_img[:, :, 1] = green[img]
    pseudo_colored_img[:, :, 2] = blue[img]

    # 转换为 uint8 的格式
    pseudo_colored_img = (pseudo_colored_img * 256).astype(np.uint8)

    if save is not None:  # 是否保存
        Image.fromarray(pseudo_colored_img).save("./result/"+save)

    return pseudo_colored_img
```

3. 最终测试的结果如下：

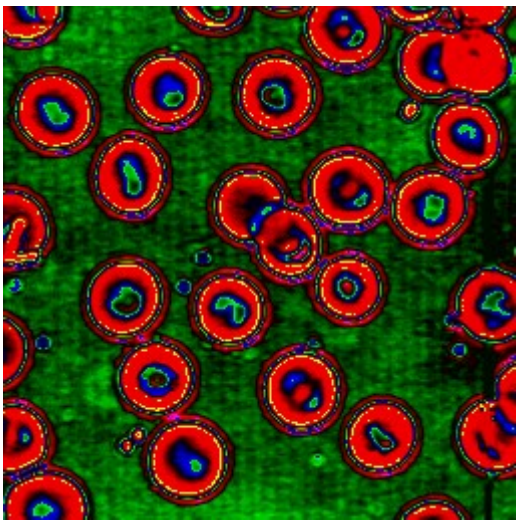
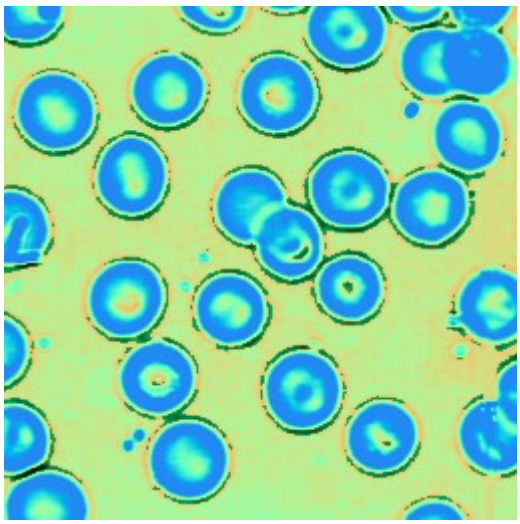
输入图片：

输出结果：

左边的结果为使用“hsv” colormap 的结果；右边为使用“rainbow” colormap 的结果



4. 结果分析:
   我们可以看到，伪彩过后的图片能够使里面的元素对比更加明显，并且不同的 colormap 进行映射过后的效果也不一样。

二、 使用世界各国 GDP 总量数据（1）用折线、散点做一个完整可视化图，显示世界各国 20 年的 GDP 数值；(2) 使用地图做图，显示世界各国 GDP 在 20 年来的动态变化。建议选择显示至少超过 5 个国家的数据

1. 用折线、散点做一个完整可视化图，显示世界各国 20 年的 GDP 数值;

   1） 算法流程:

   - 在这道题中，我仅选择了 2016 年 gdp 排名前十的国家，在过去二十年来的 gdp 变化情况，这十个国家分别为：美国，中国，日本，德国，英国，法国，印度，意大利，巴西，加拿大。
   - 统计出这 10 个国家在 1997~2016 年 20 年内的 gdp 变化情况，然后利用 pyecharts 的 Line()函数进行可视化。

   2） 具体的代码:

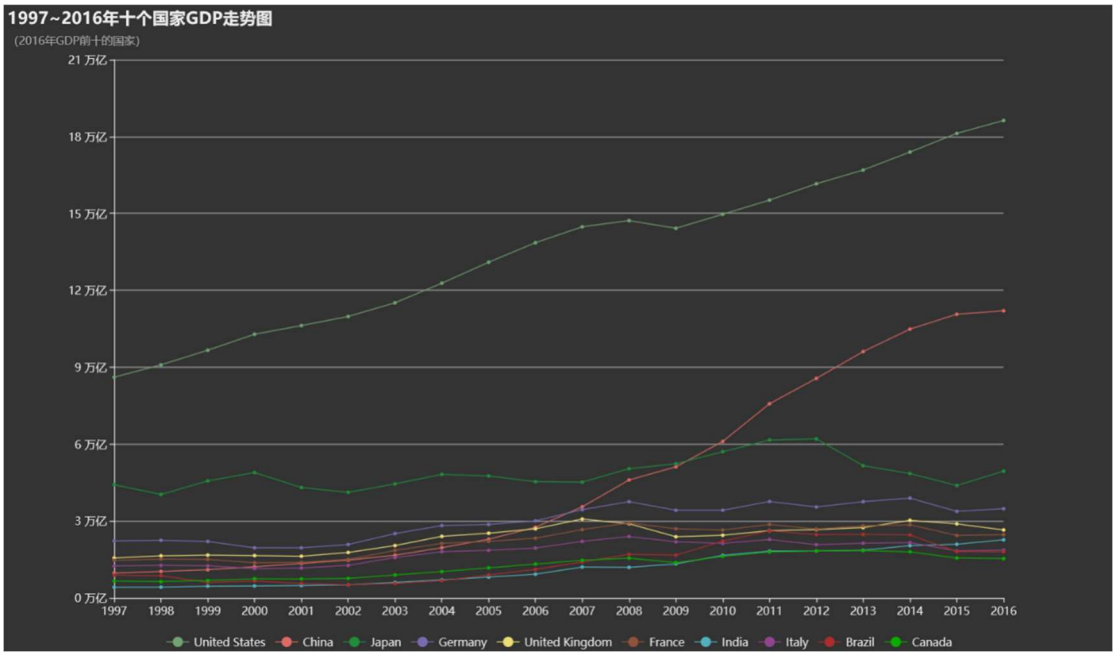
   - 数据的读取以及处理

```
# 读取数据
gdp_file = "./data/GDP-fromworldbank.xls"
gdp = xlrd.open_workbook(gdp_file)
data = gdp.sheet_by_name("Data")
# 获取2016年度的世界GDP数据，并为他们排序，选取其中的前100个经济体
gdp2016 = data.col_values(60)[4:]
gdp2016 = np.array([x if x != "" else 0 for x in gdp2016])
top100 = gdp2016.argsort()[::-1][:100]
# 获取年份信息
label = data.row_values(3)


# ------------------------------(1.)-------------------------------------------------------------
# 以下国家缩写，为从上面的2016年的 gdp 数据中手动筛选出来的前十个国家。
countries_list = ["USA", "CHN", "JPN", "DEU", "GBR", "FRA", "IND", "ITA", "BRA", "CAN"]

# 用于储存前十个国家的名字以及他们的gdp数据
countries_top10 = []
country_gdp_top10 = []

# 获取前十个国家的名字及GDP数据
for i in top100:
    tmp = data.row_values(i+4)
    if tmp[1] in countries_list:
        countries_top10.append(tmp[:2])
        country_gdp_top10.append([x/1e12 for x in tmp[41:-1]])  # 转化为万亿作为单位，并只取最近的20年，即1997~2016
# recent 20 years
year = label[41:-1]
```

- 折线图的绘制以及保存

```
# 自定义颜色
colors = ["#73a373", "#dd6b66", "#228d3c", "#7b6bad", "#eedd78", "#90533c", "#52b1bd", "#904790", "#aa2f2f", "#0dab00"]
# 初始化一个折线图，并定义大小和主题风格
line = Line(init_opts=opts.InitOpts(width="1200px", height="700px", theme=ThemeType.DARK))
# 添加 x 轴坐标
line.add_xaxis(xaxis_data=year)
# 逐个添加 y 轴的数据，并设置他们对应的线条颜色
for i in range(len(country_gdp_top10)):
    line.add_yaxis(
        series_name=countries_top10[i][0],
        y_axis=country_gdp_top10[i],
        label_opts=opts.LabelOpts(is_show=False, color=colors[i]),
        linestyle_opts=opts.LineStyleOpts(color=colors[i]),
        itemstyle_opts=opts.ItemStyleOpts(color=colors[i])
    )

# 折线图的全局设置，设计标题、交互式的形式、坐标轴的属性、图例的样式。
line.set_global_opts(
        title_opts=opts.TitleOpts(title="1997~2016年十个国家GDP走势图", subtitle="（2016年GDP前十的国家）"),
        tooltip_opts=opts.TooltipOpts(trigger="axis"),
        yaxis_opts=opts.AxisOpts(
            type_="value",
            axistick_opts=opts.AxisTickOpts(is_show=True),
            splitline_opts=opts.SplitLineOpts(is_show=True),
            axislabel_opts=opts.LabelOpts(formatter="{value} 万亿")
        ),
        xaxis_opts=opts.AxisOpts(type_="category", boundary_gap=False),
        legend_opts=opts.LegendOpts(type_="scroll", pos_bottom=0, orient="horizontal")
    )
# 输出并保存
line.render("./result/gdp_line.html")
```

3） 结果展示



由于结果为交互式的图表，这里仅能展示一个截图，可以打开提交文件中的 gdp_line.html 文件进行体验。

具体的功能还有：a）鼠标放在下面的图例会高亮显示该国家的 gdp 曲线。b）鼠标点击图例的国家可以取消折线图中该国家的 gdp 曲线，因此可以选择显示自己想要的国家。c）鼠标放在折线上会出现该点竖直方向上 10 个国家在该年的 gdp 数据。

2. 使用地图做图，显示世界各国 GDP 在 20 年来的动态变化。

1） 算法流程

- 在这道题中，我仅选择了 2016 年 gdp 排名前十的国家，在过去二十年来的 gdp 变化情况，这十个国家分别为：美国，中国，日本，德国，英国，法国，印度，意大利，巴西，加拿大，韩国，俄罗斯，西班牙，澳大利亚，墨西哥，印度尼西亚，土耳其，荷兰，瑞士，沙特阿拉伯，阿根廷，瑞典，波兰，比利时，伊朗，泰国，尼日利亚，奥地利，挪威，阿联酋。
- 统计出这三十个国家在 1997~2016 年的 gdp 数据，并将他们用 pyecharts 可视化出来。
- 设计了一个 timeline 用来记录每一年的数据，每一个时间节点，设计了一个组合图表，包含一个条形图，用来可视化 30 个国家的 gdp 数值；一个地图，用颜色来可视化 30 个国家的 gdp 数值；一个饼图，用来可视化 30 个国家的 gdp 占世界总量的比例；一个折线图，用来可视化 20 年来世界 gdp 总量的变化情况。

2） 具体代码：

- 30 个国家 gdp 数据的处理

```
# --------------------------------(2.)-----------------------------------------------------------
# 在这个图中，选择了2016年 gdp 数值排名前30的国家
countries_list = [
    "USA", "CHN", "JPN", "DEU", "GBR", "FRA", "IND", "ITA", "BRA", "CAN", "KOR", "RUS", "ESP", "AUS", "MEX", "IDN",
    "TUR", "NLD", "CHE", "SAU", "ARG", "SWE", "POL", "BEL", "IRN", "THA", "NGA", "AUT", "NOR", "ARE"
]

# 用于保存适合pyecharts的数据格式，30个国家的名称，以及他们的gdp数据
gdp_data = []
countries_top30 = []
country_gdp_top30 = []
# 获取世界总量的gdp数据，30个国家的名称，以及他们的gdp数据
for i in top100:
    tmp = data.row_values(i+4)
    if tmp[1] == "WLD":
        total_num = [x/1e12 for x in tmp[41:-1]]  # 世界总量的gdp数据 单位为万亿
    if tmp[1] in countries_list:
        # 这一步是为了能够在地图上正确显示国家的区域，因此需要将他们的名字修改为地图上对应的名字
        if tmp[1] == "RUS":
            tmp[0] = "Russia"
        if tmp[1] == "IRN":
            tmp[0] = "Iran"
        if tmp[1] == "KOR":
            tmp[0] = "Korea"
        countries_top30.append(tmp[:2])
        country_gdp_top30.append([x/1e8 for x in tmp[41:-1]])  # 单位为亿
```

```
# 将30个国家的gdp数据转为 符合 pyecharts 输入的格式
for i in range(len(year)):
    tmp1 = {"time": year[i] + "年", "data": []}
    for j in range(len(countries_list)):
        tmp2 = {"name": countries_top30[j][0], "value": [country_gdp_top30[j][i], countries_top30[j][0]]}
        tmp1["data"].append(tmp2)
    gdp_data.append(tmp1)

# 修改年份的数据格式
years = [x + "年" for x in year]

# gdp数据的最大值和最小值 用于制作图例
minNum, maxNum = 320, 186245
```

- 利用 pyecharts 可视化 30 个国家 1997~2016 年的 gdp 数据。

定义了一个函数 get_year_chart(year) 制作每一年的组合图表：包含了地图可视化，条形图可视化，饼图可视化，以及折线图可视化。

```
# 定义函数制作每一年的可视化图表
def get_year_chart(year):
    # 将数据转为符合 地图输入 的形式
    map_data = [
        [[x["name"], x["value"]] for x in d["data"]] for d in gdp_data if d["time"] == year
    ][0]
    min_data, max_data = (minNum, maxNum)
    # 用于保存折线图中每年需要mark的数据点
    data_mark: List = []
    i = 0
    for x in years:
        if x == year:
            data_mark.append(total_num[i])
            total_gdp = total_num[i]
        else:
            data_mark.append("")
        i = i + 1
```

地图组件：

```
    # 定义一个地图 展示30个国家每年的 gdp 数值
    map_chart = (
        Map()
        .add(
            series_name="",
            data_pair=map_data,
            maptype="world",
            zoom=0.75,  # 放大倍数
            center=[119.5, 34.5],  # 地图的中心位置
            is_map_symbol_show=False,  # 不显示地图地区名
            itemstyle_opts={  # 地图区域颜色
                "normal": {"areaColor": "#323c48", "borderColor": "#404a59"},
                "emphasis": {
                    "label": {"show": Timeline},
                    "areaColor": "rgba(255,255,255, 0.5)",
                },
            },
            label_opts=opts.LabelOpts(is_show=False),  # 不显示标签
            tooltip_opts=opts.TooltipOpts(  # 交互式触发的形式
                formatter="{b}: {c}"
            )
        )
        .set_global_opts(
            title_opts=opts.TitleOpts(  # 标题的设置
                title="" + str(year) + "世界30个国家年GDP情况（单位：亿）",
                subtitle="30个国家为2016年世界GDP前三十",
                pos_left="center",
                pos_top="top",
                title_textstyle_opts=opts.TextStyleOpts(
                    font_size=25, color="rgba(255,255,255, 0.9)"
                ),
            ),
```

```
            tooltip_opts=opts.TooltipOpts(  # 交互时触发的形式
                is_show=True,
                formatter=JsCode(
                    """function(params) {
                    if ('value' in params.data) {
                        return params.data.value[1] + ': ' + params.data.value[0];
                    }
                }"""
                ),
            ),
            visualmap_opts=opts.VisualMapOpts(  # 视觉映射的定义，用于制作图例
                is_calculable=True,
                dimension=0,
                pos_left="30",
                pos_top="center",
                range_text=["High", "Low"],
                range_color=["lightskyblue", "yellow", "orangered"],
                textstyle_opts=opts.TextStyleOpts(color="#ddd"),
                min_=min_data,
                max_=max_data,
            ),
        )
    )
```

折线图组件:

```
# 定义一个折线图，  展示世界gdp总量的变换
line_chart = (
    Line()
    .add_xaxis(years)
    .add_yaxis("", total_num)
    .add_yaxis(
        "",
        data_mark,
        markpoint_opts=opts.MarkPointOpts(data=[opts.MarkPointItem(type_="max")]),
    )
    .set_series_opts(label_opts=opts.LabelOpts(is_show=False))
    .set_global_opts(
        title_opts=opts.TitleOpts(
            title="世界GDP总量1997-2016年（单位：万亿）", pos_left="72%", pos_top="5%"
        )
    )
)
```

条形图组件:

```
    # 定义一个条形图，可视化30个国家的gdp数值
    bar_x_data = [x[0] for x in map_data]
    bar_y_data = [{"name": x[0], "value": int(x[1][0]+1)} for x in map_data]
    bar = (
        Bar()
        .add_xaxis(xaxis_data=bar_x_data)
        .add_yaxis(
            series_name="",
            yaxis_data=bar_y_data,
            label_opts=opts.LabelOpts(
                is_show=True, position="right", formatter="{b} : {c}"
            ),
        )
        .reversal_axis()  # 翻转坐标轴
        .set_global_opts(
            xaxis_opts=opts.AxisOpts(
                max_=maxNum, axislabel_opts=opts.LabelOpts(is_show=False)
            ),
            yaxis_opts=opts.AxisOpts(axislabel_opts=opts.LabelOpts(is_show=False)),
            tooltip_opts=opts.TooltipOpts(is_show=False),
            visualmap_opts=opts.VisualMapOpts(  # 视觉映射的设置
                is_calculable=True,
                dimension=0,
                pos_left="10",
                pos_top="top",
                range_text=["High", "Low"],
                range_color=["lightskyblue", "yellow", "orangered"],
                textstyle_opts=opts.TextStyleOpts(color="#ddd"),
                min_=min_data,
                max_=max_data,
            ),
        )
```

饼图组件：

```
# 制作一个饼图，可视化30个国家和其他国家占gdp总量的数值
pie_data = [[x[0], x[1][0]] for x in map_data]
others_gdp = total_gdp*1e4 - sum([x["value"] for x in bar_y_data])
pie_data.append(["others", others_gdp])
pie = (
    Pie()
    .add(
        series_name="",
        data_pair=pie_data,
        radius=["15%", "35%"],
        center=["80%", "82%"],
        itemstyle_opts=opts.ItemStyleOpts(
            border_width=1, border_color="rgba(0,0,0,0.3)"
        ),
    )
    .set_global_opts(
        tooltip_opts=opts.TooltipOpts(is_show=True, formatter="{b} {d}%"),
        legend_opts=opts.LegendOpts(is_show=False),
    )
)
```

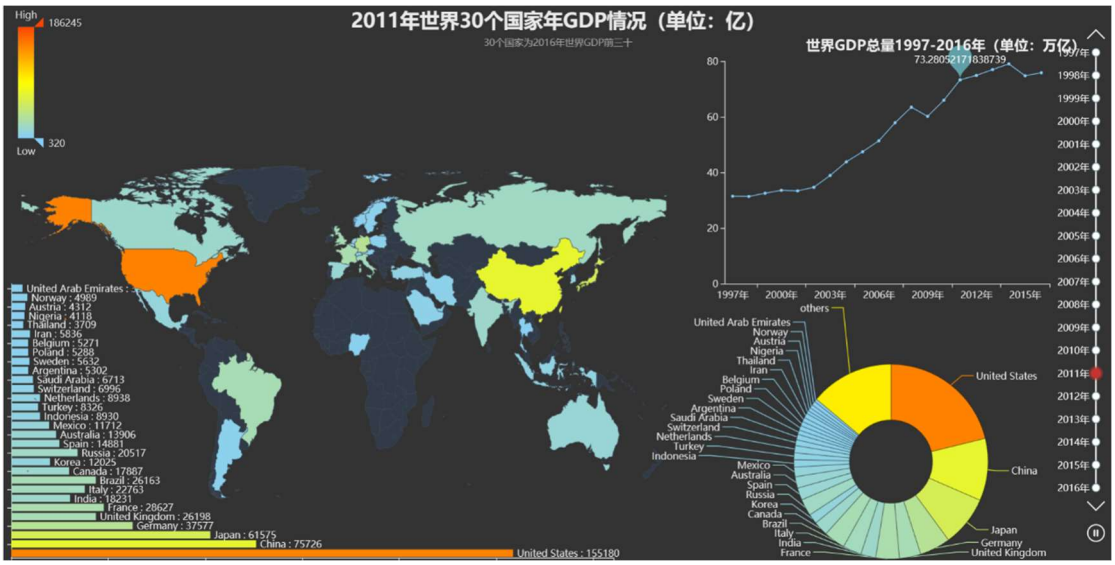
组合图表：

```
    # 制作一个组合图表，  将上面几个可视化图放在一个图表中
    grid_chart = (
        Grid()
        .add(
            bar,
            grid_opts=opts.GridOpts(
                pos_left="10", pos_right="45%", pos_top="50%", pos_bottom="5"
            ),
        )
        .add(
            line_chart,
            grid_opts=opts.GridOpts(
                pos_left="65%", pos_right="80", pos_top="10%", pos_bottom="50%"
            ),
        )
        .add(pie, grid_opts=opts.GridOpts(pos_left="45%", pos_top="60%"))
        .add(map_chart, grid_opts=opts.GridOpts())
    )

    return grid_chart
```

以上为 get_year_chat() 的函数，以下为 timeline 的设置：

```
# 制作一个 timeline 用于可视化每一年的数据
timeline = Timeline(
    init_opts=opts.InitOpts(width="1400px", height="700px", theme=ThemeType.DARK)
)
for y in years:
    g = get_year_chart(year=y)
    timeline.add(g, time_point=str(y))

timeline.add_schema(
    orient="vertical",
    is_auto_play=True,
    is_inverse=True,
    play_interval=5000,  # 自动播放的时间间隔
    pos_left="null",
    pos_right="5",
    pos_top="20",
    pos_bottom="20",
    width="60",
    label_opts=opts.LabelOpts(is_show=True, color="#fff"),
)
# 输出并保存结果
timeline.render("./result/World_gdp_1997_2016.html")
```

3） 可视化结果展示：



由于结果为交互式的图表，这里仅能展示一个截图，可以打开提交文件中的 World_gdp_1997_2016.html 文件进行体验。

具体的功能还有：a)鼠标放在任意图表的区域，都会在左上角的图例用映射对应的值；b）右边的 timeline 可以随着时间自动播放，也可以暂定，也可以自己点击上面的时间轴查看具体某年的情况；c）图中的每一个图表都会随着 timeline 中的时间变化而发生变化。d）右下角的饼图，包含了 30 个国家占世界 gdp 的总量水平，其他国家占 gdp 的总量水平归为 others 一类中。e）鼠标滑动或点击任意图表中的任意元素都会有高亮和数据提示的交互式表现。

三、请使用地震数据，使用地图可视化的方法对数据进行可视化，展现地震的地点。

1. 算法流程：
   - 首先读取数据，将数据转化为适应 pyecharts 或者 basemap 的输入格式
   - 然后调用 pyecharts 或者 basemap 的函数进行可视化。

2. 具体实现：

   1） Python + Basemap 可视化：
   - 读取数据并处理

```
# 读取数据并存储到字典中方便后面使用。

file = open("./data/quakes.csv")
content = file.readlines()
quakes = {
    "id": [],
    "lat": [],
    "long": [],
    "depth": [],
    "mag": []
}
```

```
for i in range(1, len(content)):
    row = content[i].strip().replace('"', '').split(",")
    quakes["id"].append("id"+row[0])
    quakes["lat"].append(float(row[1]))
    quakes["long"].append(float(row[2]))
    quakes["depth"].append(float(row[3]))
    quakes["mag"].append(float(row[4]))
file.close()
```

- 利用 basemap 进行可视化

```
# ---------------------------------- visualizations using basemap----------------------------------------
"""
以下代码为使用 Basemap 进行的可视化结果
"""
fig = plt.figure(figsize=(15, 15))
bmap = Basemap(projection='lcc', resolution=None, width=8E6, height=8E6, lat_0=-20, lon_0=180)

# map.etopo(scale=0.5, alpha=0.5)
# 绘制阴影的浮雕图像。
bmap.shadedrelief()

# 绘制经纬线
lat_lines = np.arange(-80, 80, 20.)
bmap.drawparallels(lat_lines, labels=[True, False, True, False], fontsize=20)
lon_lines = np.arange(-180., 181., 30.)
bmap.drawmeridians(lon_lines, labels=[False, False, False, True], fontsize=20)

# 转为np.array 形式 方便后面调整点的大小
quakes["mag"] = np.array(quakes["mag"])
# 在地图上绘制散点图
scatter = bmap.scatter(
    np.array(quakes["long"]), np.array(quakes["lat"]), latlon=True, s=100 * quakes["mag"]/quakes["mag"].max(),
    c=quakes["depth"], cmap=plt.get_cmap("YlOrRd"), alpha=1
)
# 可视化设置，并保存结果
plt.title("Earthquakes")
plt.colorbar(scatter, shrink=0.5)
plt.savefig("./result/earthquakes1.png", quality=100)
plt.show()
```

2） Pyecharts

- 初始化两个地理坐标系，并将数据处理为适应的形式，以及加入地震点的坐标。

```
# ---------------------------------- visualizations using pyecharts----------------------------------------
"""
以下代码为使用 Pyecharts 进行的可视化结果
"""
# 定义并初始化地理坐标系
geo1 = Geo(init_opts=opts.InitOpts(width="1200px", height="600px", theme=ThemeType.VINTAGE))
geo1.add_schema(
    maptype="world", zoom=3, center=[180, -20], label_opts=opts.LabelOpts(is_show=False),
)
geo2 = Geo(init_opts=opts.InitOpts(width="1200px", height="600px", theme=ThemeType.VINTAGE))
geo2.add_schema(
    maptype="world", zoom=3, center=[180, -20], label_opts=opts.LabelOpts(is_show=False),
)

# 处理经纬度以及深度和地震级数的数据，使其符合 pyecharts 的输入格式
mag_data = []
dep_data = []
for i in range(len(quakes["id"])):
    geo1.add_coordinate(quakes["id"][i], quakes["long"][i], quakes["lat"][i])
    geo2.add_coordinate(quakes["id"][i], quakes["long"][i], quakes["lat"][i])
    mag_data.append([quakes["id"][i], quakes["mag"][i]])
    dep_data.append([quakes["id"][i], quakes["depth"][i]])
```

- 地震级数的可视化图

```
# 在地理坐标系上绘制地震级数的可视化图
geo1.add(
    "", mag_data, type_=GeoType.SCATTER, symbol_size=5,
    tooltip_opts=opts.TooltipOpts(
        formatter=JsCode(
            """function(params) { return 'lon:' + params.value[0] + ', lat:' + params.value[1] + ',
            mag:' + params.value[2]; } """
        ),
    ),
)
geo1.set_series_opts(label_opts=opts.LabelOpts(is_show=False))
geo1.set_global_opts(
    title_opts=opts.TitleOpts(
        title="Earthquakes-Magnitude",
        pos_left="center",
        pos_top="top",
        title_textstyle_opts=opts.TextStyleOpts(
            font_size=25, color="black"
        ),
    ),
    visualmap_opts=opts.VisualMapOpts(
        is_calculable=True,
        dimension=2,
        pos_left="30",
        pos_top="center",
        range_text=["Magnitude"],
        range_color=["lightskyblue", "yellow", "orangered"],
        textstyle_opts=opts.TextStyleOpts(color="black"),
        min_=min(quakes["mag"]),
        max_=max(quakes["mag"]),
    ),
)
```

- 地震深度的可视化图

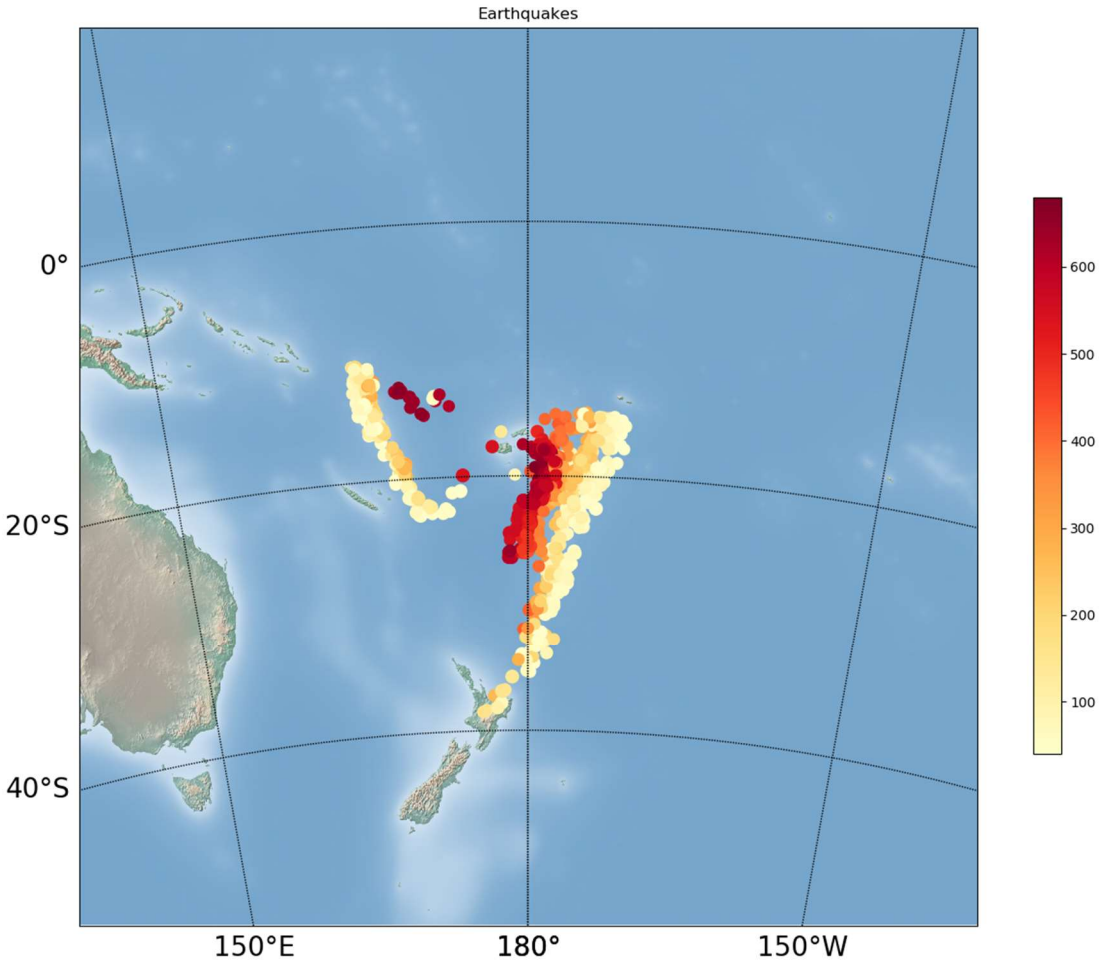
```
# 在地理坐标系上绘制地震深度的可视化图
geo2.add(
    "", dep_data, type_=GeoType.SCATTER, symbol_size=5,
    tooltip_opts=opts.TooltipOpts(
        formatter=JsCode(
            """function(params) { return 'lon:' + params.value[0] + ', lat:' + params.value[1] + ',
            depth:' + params.value[2]; } """
        ),
    ),
)
geo2.set_series_opts(label_opts=opts.LabelOpts(is_show=False))
geo2.set_global_opts(
    title_opts=opts.TitleOpts(
        title="Earthquakes-Depth",
        pos_left="center",
        pos_top="top",
        title_textstyle_opts=opts.TextStyleOpts(
            font_size=25, color="black"
        ),
    ),
    visualmap_opts=opts.VisualMapOpts(
        is_calculable=True,
        dimension=2,
        pos_left="30",
        pos_top="center",
        range_text=["Depth"],
        range_color=["lightskyblue", "yellow", "orangered"],
        textstyle_opts=opts.TextStyleOpts(color="black"),
        min_=min(quakes["depth"]),
        max_=max(quakes["depth"]),
    ),
)
```

- 制作分页展示

```
# 将上述两个图组合起来
page = Page(layout=Page.DraggablePageLayout)

page.add(
    geo1,
    geo2
)
# 输出结果
page.render("./result/earthquake2.html")
# ----------------------------------------------------------------------------------------------------------------
```

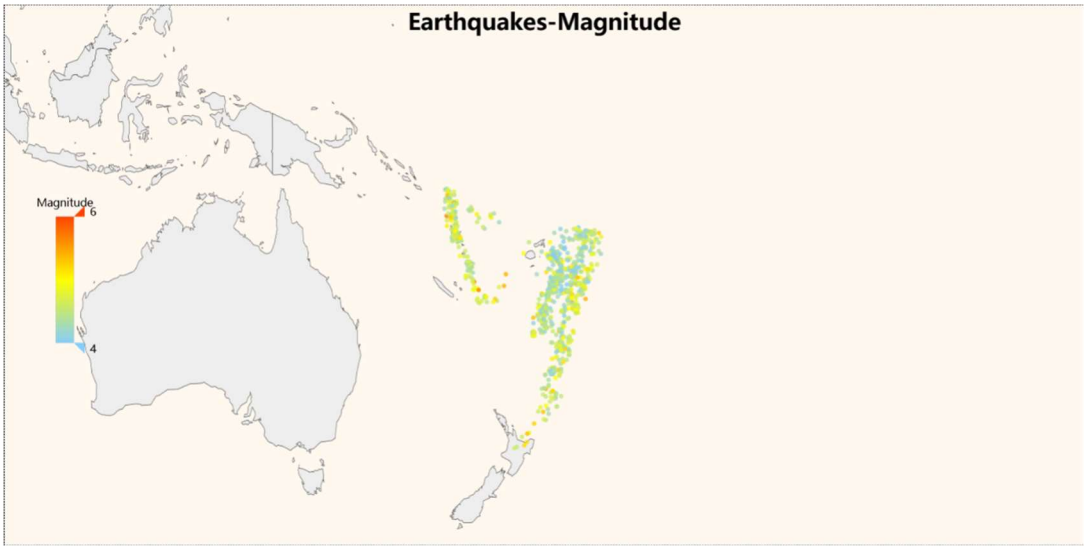
3. 可视化结果展示：

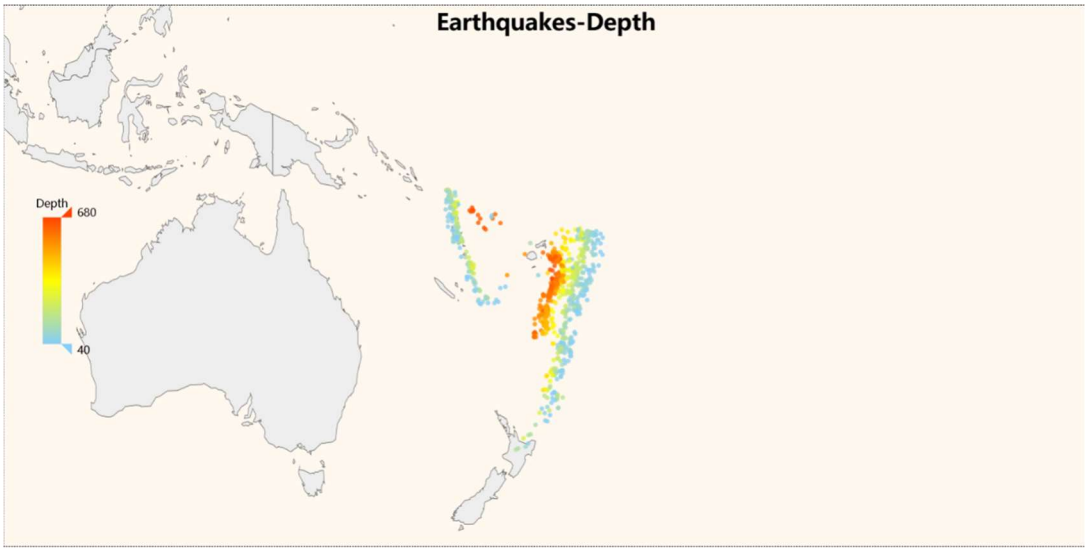
1） Python + basemap 的可视化结果：



在这个图中，用颜色深浅代表地震的深度，点的大小代表了地震的级数。(由于级数的数值过于接近，都在 4~6 之前，直接可视化效果不明显，因此将级数归一化后再乘了 100 才作为点的 size 输入，才能比较明显) 但是，basemap 并不知道如何设置一个关于圆点 size 的视觉映射的图例，因此尝试用 pyecharts 可视化。

2） Pyecharts





由于结果为交互式的图表，这里仅能展示一个截图，可以打开提交文件中的 earthquakes2.html 文件进行体验。

第一张图为，关于地震级数的可视化图， 第二张图为关于地震深度的可视化图。

（pyecharts 也不会在一个图里面同时加入两个 视觉映射，只能分开画两个地理坐标系，然后用 Page()组合起来）

3） 结果分析:

从两个方法的可视化结果来看，我们可以看出，

a）地震的深度在 lon=180，lat=-20 左右的位置最深，从周围散开逐渐变浅；

b）地震的级数大多都在 4~4.5 之间，级数大于 5 的地震点十分少，并且并不是深度越深的地点级数越大，分布偏随机一点。

四、 总结：

至此，第五次作业完成。第一题较为简单，而第二题和第三题，花费很多时间，阅读了大量的 pyecharts 和 basemap 的官方文档，对不同的函数和可视化设置进行了尝试，才能做出相应的可视化图，收获还是很多的。