# 二氧化碳数据分析

🌸 19377404 王临寒

## STEP 1 实现数据分析类

本次作业的数据分析是为可视化服务的，目的是通过 Excel 里面的数据得到可视化数据的输入。

数据中主要体现了四个维度的信息：时间（年份不同）、地域（省份）、燃料种类、行业。

所以主要分析三个层次的问题：

- 二氧化碳排放量的地区分布
- 燃料种类和行业之间的能量流动
- 不同时间不同城市（或者同一城市不同行业）的排放量变化

### STEP 1.1 二氧化碳排放量的地区分布

```
class Analytics:
    def __init__(self):
        # 由于pyecharts的地图可视化需要中文，所以先存储中英文对照字典
        self.dictionary = {'Beijing': "北京",
                           'Tianjin': "天津",
                           'Hebei': "河北",
                           'Shanxi': "山西",
                           'InnerMongolia': "内蒙古",
                           'Liaoning': "辽宁",
                           'Jilin': "吉林",
                           'Heilongjiang': "黑龙江",
                           'Shanghai': "上海",
                           'Jiangsu': "江苏",
                           'Zhejiang': "浙江",
                           'Anhui': "安徽",
                           'Fujian': "福建",
                           'Jiangxi': "江西",
                           'Shandong': "山东",
                           'Henan': "河南",
                           'Hubei': "湖北",
                           'Hunan': "湖南",
                           'Guangdong': "广东",
                           'Guangxi': "广西",
                           'Hainan': "海南",
                           'Chongqing': "重庆",
                           'Sichuan': "四川",
                           'Guizhou': "贵州",
                           'Yunnan': "云南",
                           'Shaanxi': "陕西",
                           'Gansu': "甘肃",
                           'Qinghai': "青海",
                           'Ningxia': "宁夏",
                           'Xinjiang': "新疆"}
```

```python
    def analyze_area(self, year, category):
        '''
        :param year: 年份
        :param category: 燃料种类
        :return:
        '''
        analyze_area_df = pd.read_excel(
            "./co2_demo/Province sectoral CO2 emissions " + str(year) + ".xlsx",
            sheet_name="Sum", index_col=[0])
        analyze_area_eng_name = list(analyze_area_df.index)[0:30]
        analyze_area_chi_name = []

        for i in analyze_area_eng_name:
            analyze_area_chi_name.append(self.dictionary[i])
        analyze_area_data = list(analyze_area_df[category])[0:30]
        return dict(zip(analyze_area_chi_name, analyze_area_data))
```

Python

测试 analyze_area()：

```python
if __name__ == "__main__":
    a = Analytics()
    print(a.analyze_area(2015, "Raw Coal"))
```

Python

输出为一个字典，键为省份，值为数值：

```
F:\anaconda3\envs\homework\python.exe D:/大三上/10现代程序设计技术/作业/第7次/main.py
{'北京': 95.2, '天津': 151.9, '河北': 734.1, '山西': 440.2, '内蒙古': 584.7, '辽宁': 472.1, '吉林': 207.6, '黑龙江': 265.5,

Process finished with exit code 0
```

## STEP 1.2 燃料种类和行业之间的能量流动

如果要分析燃料和行业的关系，就需要描述从燃料到行业的能量流动过程。

```python
    def analyze_fuel_distribution(self, year, area):
        analyze_fuel_distribution_df = pd.read_excel(
            "./co2_demo/Province sectoral CO2 emissions " + str(year) + ".xlsx",
            sheet_name=area, index_col=[0])
        industry = list(analyze_fuel_distribution_df.index)[3:]
        fuel_type = list(analyze_fuel_distribution_df.columns)[:-4]
        node_list = []
        links = []
        for i in industry:
            for j in fuel_type:
                if analyze_fuel_distribution_df.loc[i, j] != 0:
                    node_list.append(i)
                    node_list.append(j)
                    links.append({"source": j, "target": i, "value": analyze_fuel_distribution
        nodes = []
        for i in list(set(node_list)):
            nodes.append({"name": i})
        return nodes, links
```

Python

输出的 nodes 代表节点，也就是所有的行业和燃料；

links 包含源点（燃料）、终点（行业）、值（i 燃料到 j 行业的具体量）

测试的主函数：

```python
if __name__ == "__main__":
    a = Analytics()
    nodes, links = a.analyze_fuel_distribution(2015, "Hebei")
    print("Nodes:", nodes)
    print("Links:", links)
```

输出结果：

```
F:\anaconda3\envs\homework\python.exe D:/大三上/10现代程序设计技术/作业/第7次/main.py
Nodes: [{'name': 'Electric Equipment and Machinery                    '}, {'name': 'Furniture Manufacturing
Links: [{'source': 'Raw Coal', 'target': 'Farming, Forestry, Animal Husbandry, Fishery and Water Conservancy      ', 'value': 2.2},
```

与预期数据结构相符。

## STEP 1.3 排放量随时间的变化

主要指定了两种变化：

- 不同城市总排放量随着时间的变化
- 同一城市不同行业排放量随着时间的变化

### STEP 1.3.1 不同城市总排放量随着时间的变化

```python
def analyze_city_time_distribution(self):
    all_time_dict = {}
    for i in range(1997, 2016):
        data_now = pd.read_excel("./co2_demo/Province sectoral CO2 emissions " + str(i) +
                                 index_col=[0])
        all_time_dict[str(i)] = dict(data_now["Total"][:-2])
    x_data = list(all_time_dict['1997'].keys())
    y_data = []
    # 时间
    for i in all_time_dict.keys():
        # 地点
        for j in all_time_dict[i].keys():
            y_data.append([i, all_time_dict[i][j], j])
    return x_data, y_data
```

### STEP 1.3.2 同一城市不同行业排放量随着时间的变化

```python
def analyze_city_industry_distribution(self, city):
    all_time_dict = {}
    data = pd.read_excel("./co2_demo/Province sectoral CO2 emissions 2015.xlsx", sheet_nam
    industry = list(data.index)[3:]
    for i in range(1997, 2016):
        data_now = pd.read_excel("./co2_demo/Province sectoral CO2 emissions " + str(i) +
                                 sheet_name=city, index_col=[0])
        a_year_dict = {}
        for j in industry:
            a_year_dict[j] = data_now.loc[j, "Total"]
        all_time_dict[str(i)] = a_year_dict
    x_data_industry = list(all_time_dict['1997'].keys())
    y_data_industry = []
    # 时间
    for i in all_time_dict.keys():
        # 地点
        for j in all_time_dict[i].keys():
            y_data_industry.append([i, all_time_dict[i][j], j])
    return x_data_industry, y_data_industry
```

这两个函数大部分相似，输出的数据结构也是相同的，不同之处在于 Excel 结构的不同导致前半部分数据处理的细节略有差异。

测试函数：

```python
if __name__ == "__main__":
    a = Analytics()
    x_data, y_data = a.analyze_city_time_distribution()
    print("x_data:",x_data)
    print("y_data:",y_data)
```

输出结果：

```
F:\anaconda3\envs\homework\python.exe D:/大三上/10现代程序设计技术/作业/第7次/main.py
x_data: ['Beijing', 'Tianjin', 'Hebei', 'Shanxi', 'InnerMongolia', 'Liaoning', 'Jilin', 'Heilongjiang', 'Shanghai',
y_data: [['1997', 61.9, 'Beijing'], ['1997', 51.4, 'Tianjin'], ['1997', 212.1, 'Hebei'], ['1997', 148.7, 'Shanxi'],

Process finished with exit code 0
```

Ref.



# STEP 2 实现数据可视化类

## STEP 2.1 地区分布热力图

如果要体现二氧化碳排放量的地区分布，那么最好的选择就是热力图。

这里选择 2015 年所有的燃料的排放量之和，效果会更明显。

```python
class Visualization:
    def __init__(self, data):
        self.data = data

    def plot_area(self):
        province_distribution = self.data
        province = list(province_distribution.keys())
        values = list(province_distribution.values())
        chinese_map = Map("中国地图", '中国地图', width=1200, height=600)
        chinese_map.add("", province, values, visual_range=[0, 600], maptype='china', is_visua
                        visual_text_color='#000')
        chinese_map.render(path="指定时间指定种类的CO2区域分布.html")
```

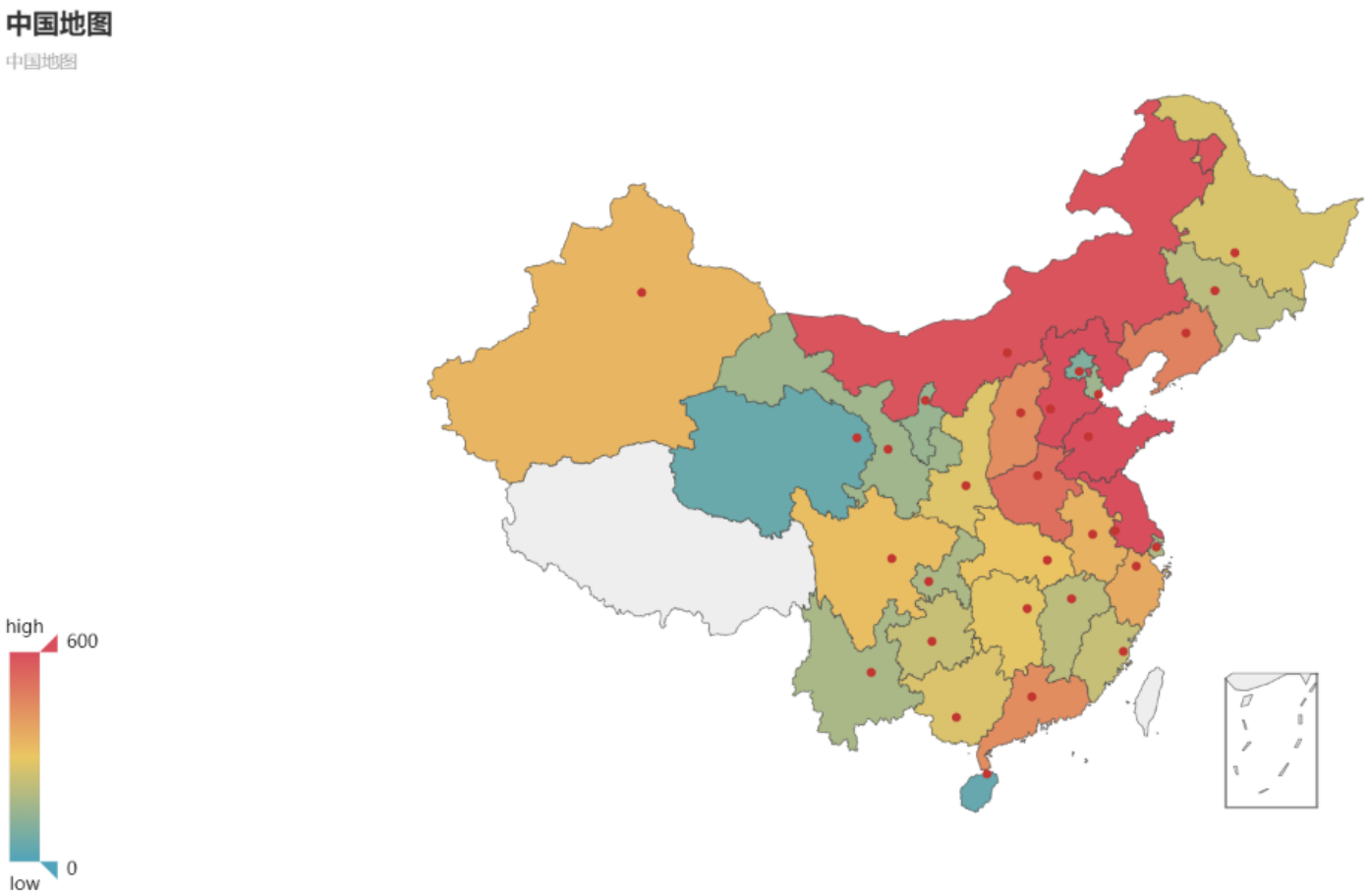
进行测试的 main 函数为：

```python
if __name__ == "__main__":
    a = Analytics()
    data = a.analyze_area(2015, "Total")
```

```
    b = Visualization(data)
    b.plot_area()
```

Python

输出 html 文件，打开后得到：




Ref.

**python 利用echarts画地图(热力图)(世界地图，省市地图，区县地图...**

首先安装对应的python模块$ pip install pyecharts==0.5.10 $ pip install echarts-countries-pypkg $ pip install echarts-china-provinces-pypkg $ pip install echarts-china-cities-pypkg $ pip install echarts-ch...

blog.csdn.net

## STEP 2.2 能量流动桑基图

```
def plot_sankey(self, nodes, links):
    c = (
        Sankey()
            .add(
            "sankey",
            nodes,
            links,
            linestyle_opt=opts.LineStyleOpts(opacity=0.2, curve=0.5, color="source"),
            label_opts=opts.LabelOpts(position="right"),
        )
            .set_global_opts(title_opts=opts.TitleOpts(title="燃料种类和行业分布的关系"))

            .render("燃料种类和行业分布的关系.html")
    )
```

Python

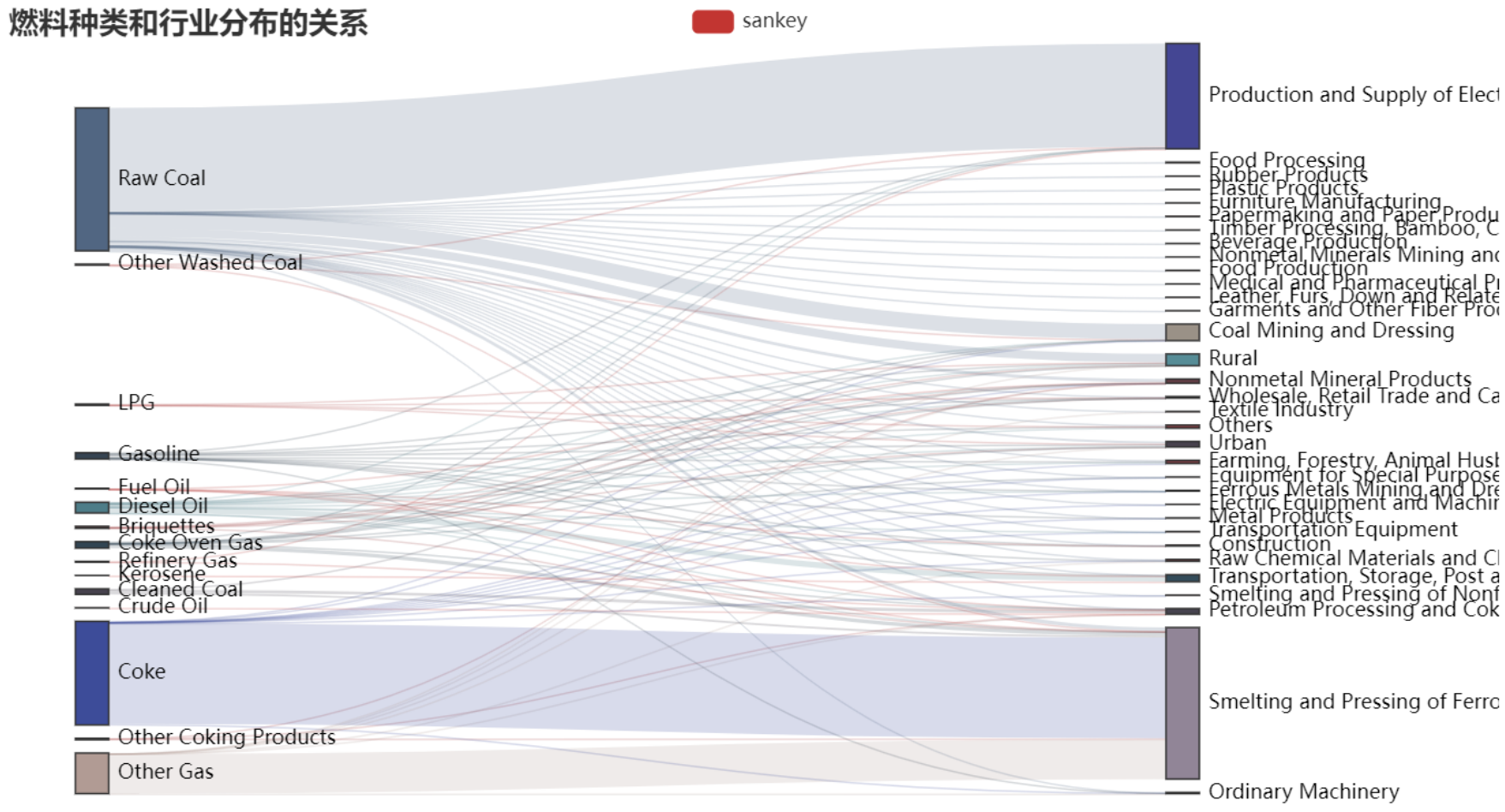
同样利用 pyecharts 中的函数生成 html 文件。

测试 main 函数：

```
if __name__ == "__main__":
    a = Analytics()
    nodes, links = a.analyze_fuel_distribution(2015, "Hebei")
    b = Visualization(nodes)
    b.plot_sankey(nodes, links)
```

Python

输出图片：




可以发现，在 2015 年的河北，大量的生煤被用作发电，大量的焦炭被用作炼铁。桑基图能够直观反映出不同主体之间的结构关系，易于分辨某行业的主要燃料及燃料的消费结构。

Ref.

**Document**

Description

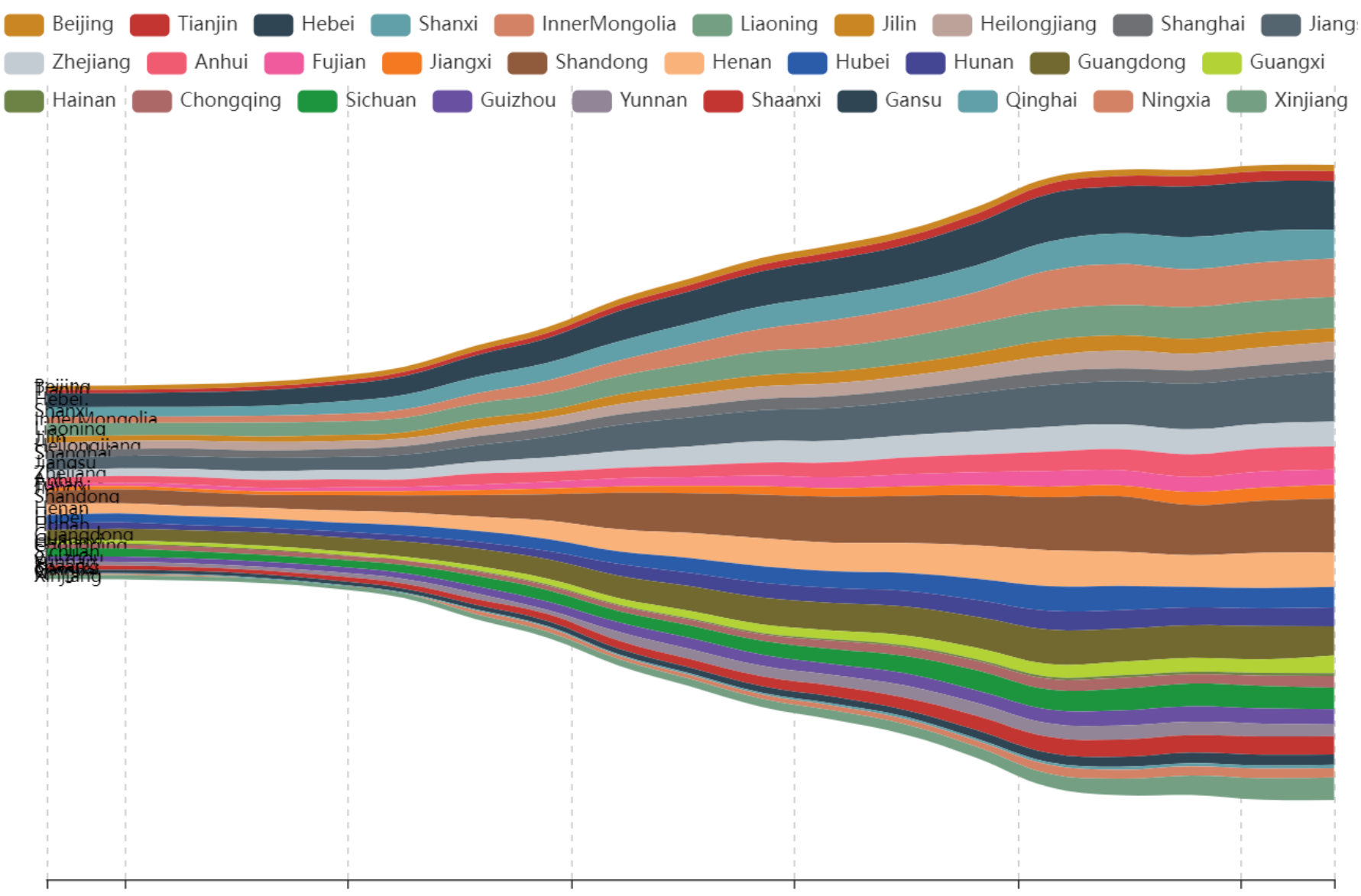
gallery.pyecharts.org

## STEP 2.3 数量变化河流图

```python
    def plot_themeriver(self, x_data, y_data):
        c = (
            ThemeRiver(init_opts=opts.InitOpts(width="900px", height="600px"))
                .add(
                series_name=x_data,
                data=y_data,
                singleaxis_opts=opts.SingleAxisOpts(
                    pos_top="50", pos_bottom="50", type_="time"
                ),
            )
                .set_global_opts(
                tooltip_opts=opts.TooltipOpts(trigger="axis", axis_pointer_type="line")
            )
                .render("指定城市不同行业二氧化碳排放量随时间变化.html")
        )
```

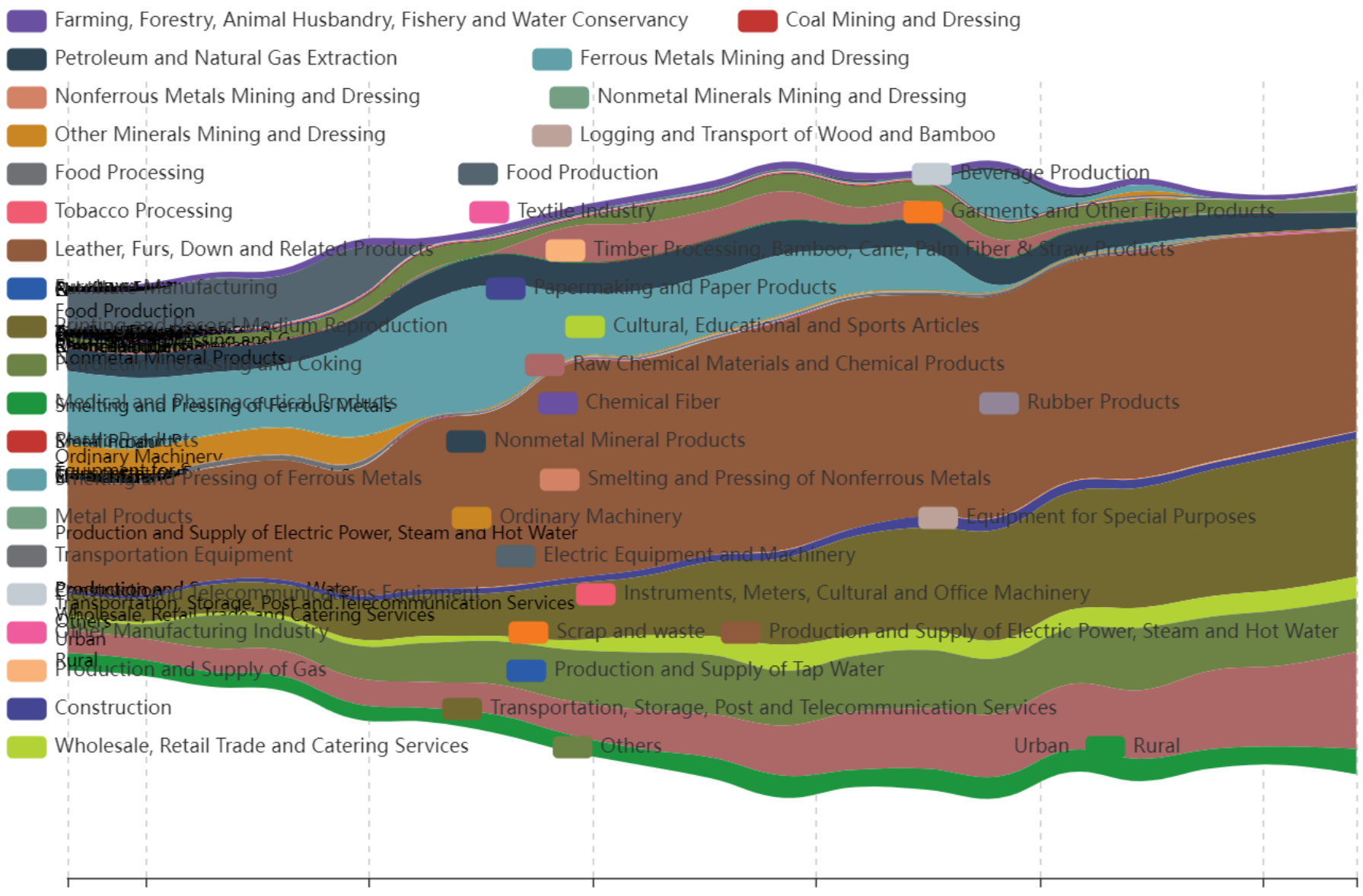
Python

首先是各个城市排放总量随时间变化图：




可以发现，从 1997 年到 2015 年，排放量在大多数省都是上升的趋势。

之后是同一城市不同行业的变化：




此处因为行业过多而图片稍显混乱，不过还是可以看出 Production and Supply of Electric Power, Steam and Hot Water 的比例占有绝对优势。

Ref.

## STEP 3 实现 NotNumError 类

继承了 ValueError 的 NotNumError 类：

```python
class NotNumError(ValueError):
    def __init__(self, year, province, industry, fuel_type):
        self.year = year
        self.province = province
        self.industry = industry
        self.fuel_type = fuel_type
        self.message = "nan error in {}, {}, {}, {}".format(year, province, industry, fuel_typ
```

同时也新建了一个 Check 类，把判断的过程封装在函数中：

```python
class Check:
    def __init__(self, year, province, industry, fuel_type):
        self.year = year
        self.province = province
        self.industry = industry
        self.fuel_type = fuel_type
    def check_nan(self):
        data = pd.read_excel("./co2_demo/Province sectoral CO2 emissions "+str(self.year)+".xl
                             sheet_name=self.province, index_col=[0])
        val = data.loc[self.industry, self.fuel_type]
        if pd.isna(val):
            raise NotNumError(self.year, self.province, self.industry, self.fuel_type)
```

测试的 main 函数：

```python
if __name__ == "__main__":
    check = Check(2015, "Beijing", "Food Processing                                    "
    try:
        check.check_nan()
    except NotNumError as nne:
        print(nne.message)
    else:
        print("The value is not nan!")
```

正常情况显示：

```
F:\anaconda3\envs\homework\python.exe D:/大三上/10现代程序设计技术/作业/第7次/main.py
The value is not nan!

Process finished with exit code 0
```

刻意修改为 nan 之后显示：

```
F:\anaconda3\envs\homework\python.exe D:/大三上/10现代程序设计技术/作业/第7次/main.py
nan error in 2015, Beijing, Food Processing                                    , Coke Oven Gas

Process finished with exit code 0
```

## STEP 4 ZeroDivisionError 的实现

在 Check 类中新增一个检验函数：

```python
    def check_zero_sum(self):
        data = pd.read_excel("./co2_demo/Province sectoral CO2 emissions "+str(self.year)+".xl
```

```
                                sheet_name=self.province, index_col=[0])
        total = data.loc[self.industry, "Total"]
        if total == 0:
            raise ZeroDivisionError
```

Python

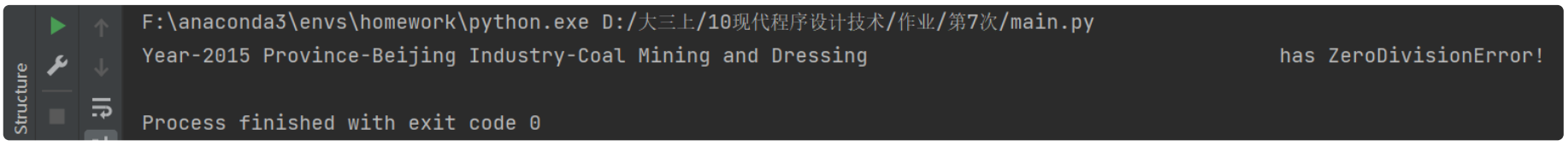
如果 Total 列没有问题则未出现该错误。

测试的 main 函数：

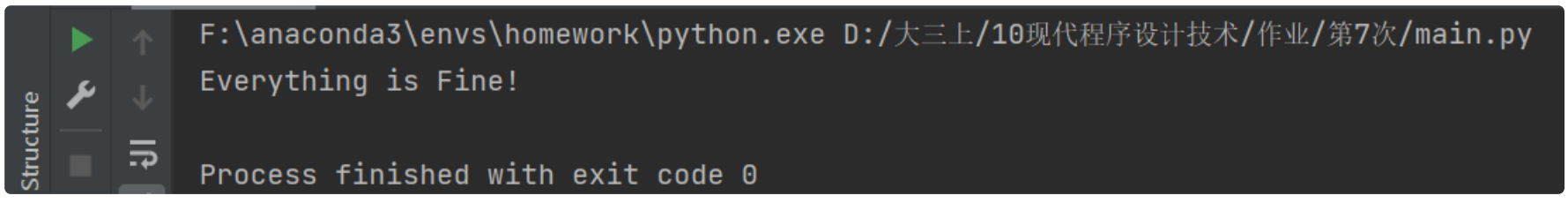
```
if __name__ == "__main__":
    check = Check(2015, "Beijing", "Coal Mining and Dressing                                    "
    try:
        check.check_zero_sum()
    except ZeroDivisionError:
        print("Year-{} Province-{} Industry-{} has ZeroDivisionError!".format(check.year, chec
    else:
        print("Everything is Fine!")
```

Python

当总量为 0 时输出：




当总量不为 0 时输出：




Ref.

**浅谈pandas中对nan空值的判断和陷阱–pandas判断nan空值–IT技...**

pandas基于numpy，所以其中的空值nan和numpy.nan是等价的。numpy中的nan并不是空对象，其实际上是numpy.float64对象，所以

www.pcppw.com