本页不打印

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 开发/优化者 | 时间 | 审核人 | 开发类型（新开发/优化） | 更新说明 |
| 陈傲 | 2020.04.17 |  | 新开发 |  |
| 陈傲 | 2020.07.01 |  | 优化 |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| 课程编码 | 适用产品 | 产品版本 | 课程版本ISSUE |
|  | 鲲鹏应用数据分析与管理创新实践课 |  | V1.2 |

修订记录

关卡4

基于litemall的经营决策分析可视化体现



华为技术有限公司

目录

[1 Pyecharts可视化大屏搭建指导 2](#_Toc51602992)

[1.1 实验介绍 2](#_Toc51602993)

[1.1.1 关于本实验 2](#_Toc51602994)

[1.1.2 实验目的 2](#_Toc51602995)

[1.2 实验代码 2](#_Toc51602996)

[1.2.1 Pyecharts基本操作简介 2](#_Toc51602997)

[1.2.2 数据准备 9](#_Toc51602998)

[1.2.3 分析2020年3月各省GMV数据，绘制玫瑰图 11](#_Toc51602999)

[1.2.4 分析各省用户数量，在中国地图上绘制热力图 14](#_Toc51603000)

[1.2.5 分析各商品销量及销售金额，同一副图里显示折线图与柱状图 16](#_Toc51603001)

[1.2.6 下钻与上卷 19](#_Toc51603002)

[1.2.7 利用Page函数，完成可视化大屏的搭建。 31](#_Toc51603003)

[1.3 创新指导 38](#_Toc51603004)

[1.4 本章小结 38](#_Toc51603005)

# Pyecharts可视化大屏搭建指导

## 实验介绍

### 关于本实验

本实验主要介绍Pyecharts可视化的基本操作，包括图表绘制基础练习代码，分析2020年3月各省GMV数据并绘制玫瑰图，分析全国各省用户数量并绘制地图，分析各商品销量及销售金额并在同一副图里绘制折线图与柱状图，利用JavaScript代码完成下钻与上卷的操作，最后将所有这些图表汇集在同一个大屏内，通过简单的拖拽来调整页面布局。

### 实验目的

帮助学员理解Pyecharts可视化的基本思想，让学员掌握如何进行数据挖掘分析结果可视化的基本流程，以便帮助学员更好地完成商业智能BI分析。

## 实验代码

### Pyecharts基本操作简介

安装Pyecharts库

在Jupyter Notebook 代码块中输入：

|  |
| --- |
| !pip install -i https://pypi.tuna.tsinghua.edu.cn/simple pyecharts==1.7.1 |

显示 ”Succesfully installed pyecharts” 就表示已经安装成功了。

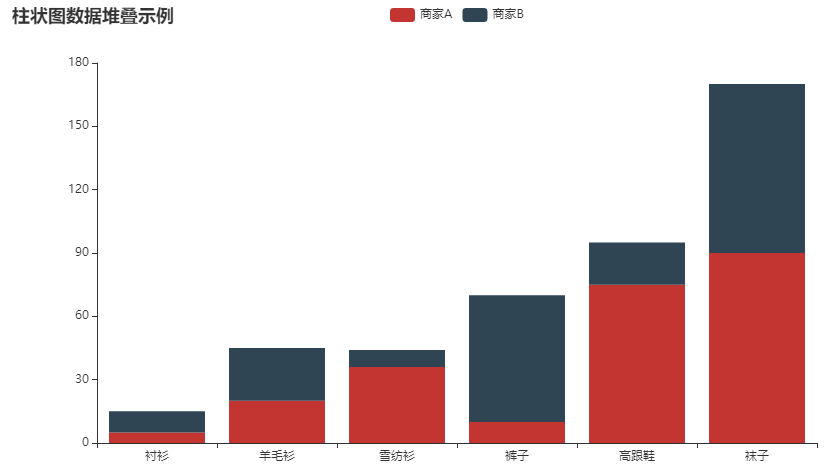
绘制柱状图

代码如下：

|  |
| --- |
| import pyecharts.options as opts  from pyecharts.charts import Bar  import os  attr = ["衬衫", "羊毛衫", "雪纺衫", "裤子", "高跟鞋", "袜子"]  v1 = [5, 20, 36, 10, 75, 90]  v2 = [10, 25, 8, 60, 20, 80]    bar = (  Bar()  .add\_xaxis(attr)  .add\_yaxis("商家A", v1, stack="stack1")  .add\_yaxis("商家B", v2, stack="stack1") # 指定stack="stack1"时，两列数据会重叠，如下图所示  .set\_series\_opts(label\_opts=opts.LabelOpts(is\_show=False)) #是否显示数据标签，False为不显示  .set\_global\_opts(title\_opts=opts.TitleOpts(title="柱状图数据堆叠示例")) #设置标题  )  bar.render(os.path.join(os.path.expanduser('~'),’Desktop’,"bar1.html")) #保存为html文件，如果想在Jupyter里直接显示，可以用bar.render\_notebook() |

运行代码

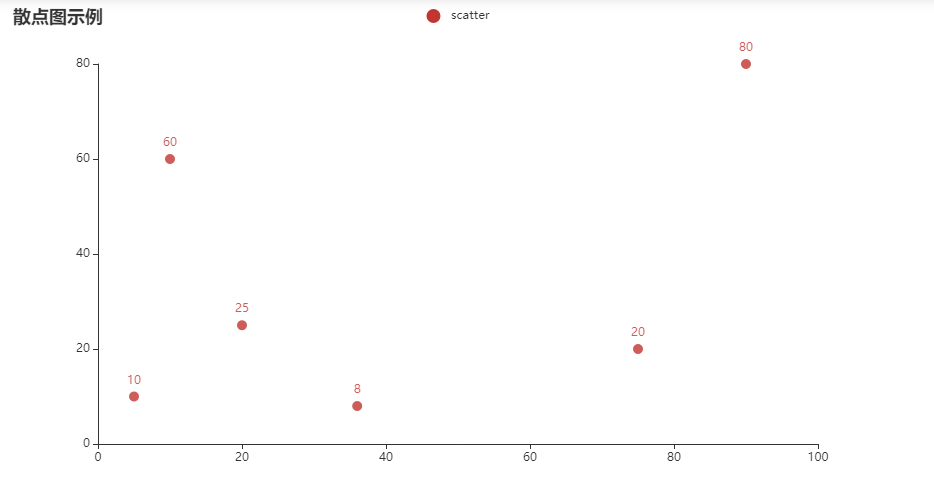
运行完成后，会在项目路径下中生成一个bar1.html文件，在桌面找到该文件，用浏览器打开，可以看到可视化效果：



绘制散点图和动态散点图

|  |
| --- |
| import pyecharts.options as opts  from pyecharts.charts import Scatter,EffectScatter,Grid  v1 = [5, 20, 36, 10, 75, 90]  v2 = [10, 25, 8, 60, 20, 80]  scatter = (  Scatter()  .add\_xaxis(xaxis\_data=v1)  .add\_yaxis(  series\_name="scatter", #系列名称  y\_axis=v2, #传入数据  label\_opts=opts.LabelOpts(is\_show=True) #显示数据标签  )  .set\_global\_opts(  xaxis\_opts=opts.AxisOpts( #定制横坐标相关参数  type\_="value",  #splitline\_opts=opts.SplitLineOpts(is\_show=True) #是否添加网格线，True表示显示网格线  ),  yaxis\_opts=opts.AxisOpts(#定制纵坐标相关参数  type\_="value",  #splitline\_opts=opts.SplitLineOpts(is\_show=True), #是否添加网格线，True表示显示网格线  ),  title\_opts=opts.TitleOpts(title="散点图示例"))  )  scatter.render\_notebook() # 直接在Jupyter Notebook里显示 |

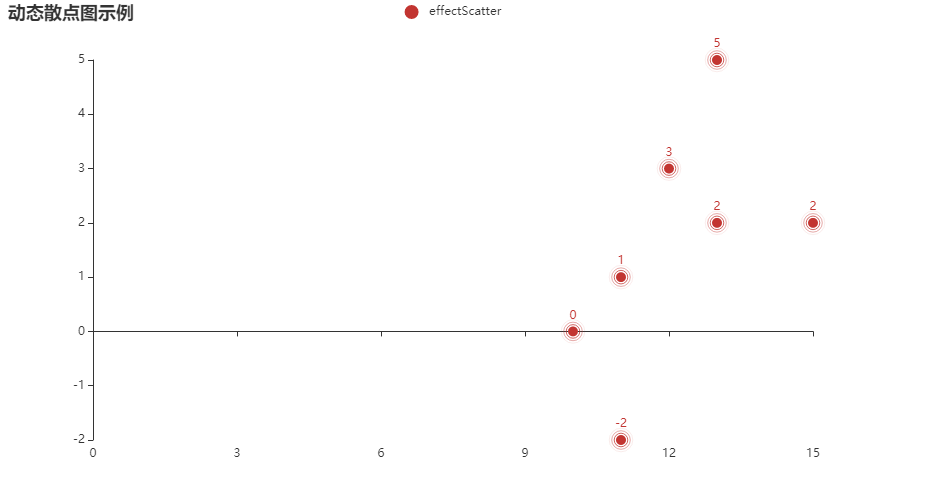
运行后效果如下：



Pyecharts还支持“涟漪散点图“：

|  |
| --- |
| from pyecharts.globals import SymbolType #这是图标的类型，在下文会有介绍  es = (  EffectScatter() # 涟漪（动态）散点图  .add\_xaxis(xaxis\_data=[11, 11, 15, 13, 12, 13, 10])  .add\_yaxis(  series\_name="effectScatter",  y\_axis=[1, -2, 2, 5, 3, 2, 0],  label\_opts=opts.LabelOpts(is\_show=True),  #symbol=SymbolType.DIAMOND #之前导入了图标类型模块，这里就是设置图标为菱形  )  .set\_global\_opts(  xaxis\_opts=opts.AxisOpts(  type\_="value",  #splitline\_opts=opts.SplitLineOpts(is\_show=True)  ),  yaxis\_opts=opts.AxisOpts(  type\_="value",  #splitline\_opts=opts.SplitLineOpts(is\_show=True),  ),  title\_opts=opts.TitleOpts(title="动态散点图示例"))  )  es.render\_notebook() |

运行后效果如下：

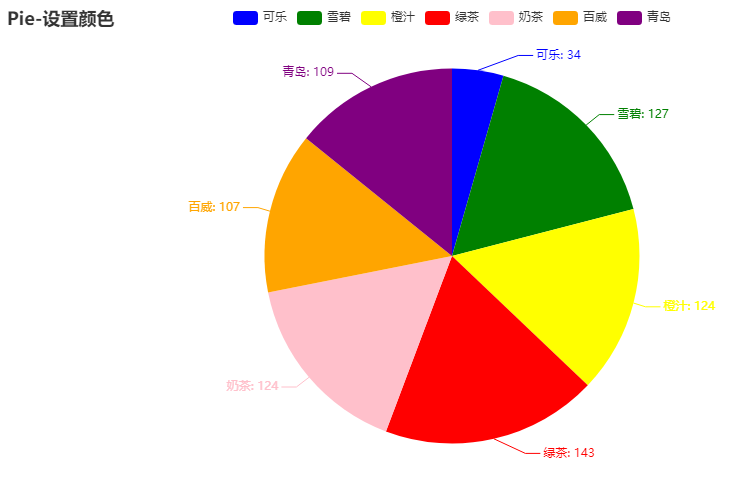


绘制饼图

代码如下：

|  |
| --- |
| from pyecharts import options as opts  from pyecharts.charts import Pie  from pyecharts.faker import Faker  c = (  Pie()  .add("", [list(z) for z in zip(Faker.choose(), Faker.values())]) #zip()是打包（拉链）函数，可以把两个序列的元素按索引顺序一一对应进行咬合，形成新的序列  .set\_colors(["blue", "green", "yellow", "red", "pink", "orange", "purple"])  .set\_global\_opts(title\_opts=opts.TitleOpts(title="Pie-设置颜色"))  .set\_series\_opts(label\_opts=opts.LabelOpts(formatter="{b}: {c}"))#formmatter可以指定标签类型，比如饼状图里的{b}: {c}，表示 类别:数据，详情请见下图  )  c.render\_notebook() |

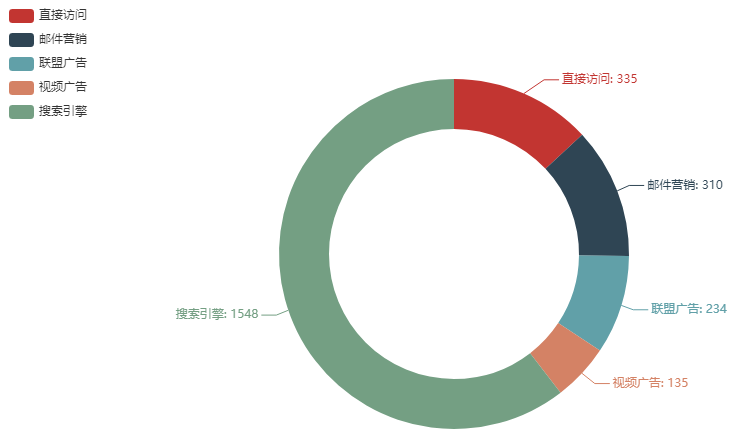
效果如下：



环形饼状图代码如下：

|  |
| --- |
| x\_data = ["直接访问", "邮件营销", "联盟广告", "视频广告", "搜索引擎"]  y\_data = [335, 310, 234, 135, 1548]  pie = (  Pie()  .add(  series\_name="访问来源",  data\_pair=[list(z) for z in zip(x\_data, y\_data)],  radius=["50%", "70%"], # 设置环形的内外半径，推荐使用百分比  label\_opts=opts.LabelOpts(is\_show=False, position="center"),  )  .set\_global\_opts(legend\_opts=opts.LegendOpts(pos\_left="left", orient="vertical")) # pos\_left="left"表示图例靠左放置；orient="vertical"表示图例垂直排列  .set\_series\_opts(label\_opts=opts.LabelOpts(formatter="{b}: {c}"))#formatter可以指定标签类型，比如这里的{b}: {c}，表示 ”类别:数据“  )  pie.render\_notebook() |

运行效果如下：

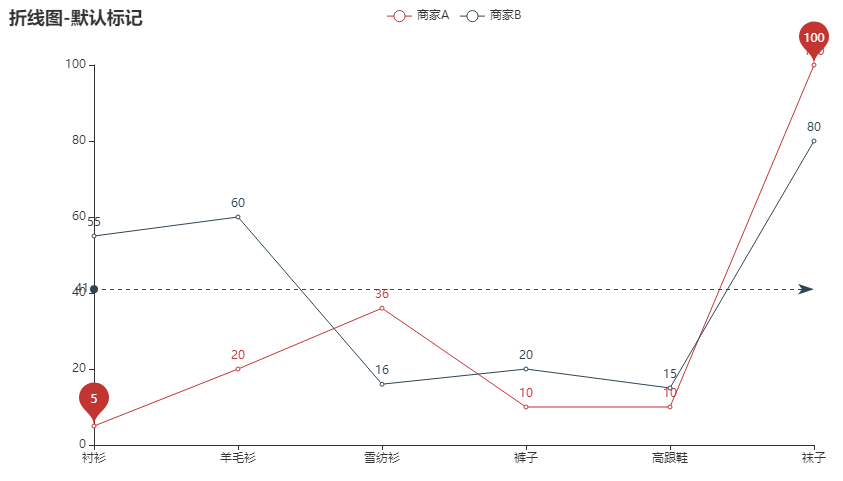


绘制折线图

代码如下：

|  |
| --- |
| from pyecharts.charts import Line  attr = ["衬衫", "羊毛衫", "雪纺衫", "裤子", "高跟鞋", "袜子"]  v1 = [5, 20, 36, 10, 10, 100]  v2 = [55, 60, 16, 20, 15, 80]  chart=(Line()  .add\_xaxis(xaxis\_data=attr)  .add\_yaxis(  series\_name = "商家A",  y\_axis=v1,  label\_opts=opts.LabelOpts(is\_show=True),  markpoint\_opts=opts.MarkPointOpts(  data=[  opts.MarkPointItem(type\_="max", name="最大值"),# 标出每条折线的最大值  opts.MarkPointItem(type\_="min", name="最小值"),# 标出每条折线的最小值  ]  )  )  .add\_yaxis(  series\_name = "商家B",  y\_axis=v2,  label\_opts=opts.LabelOpts(is\_show=True),  markline\_opts=opts.MarkLineOpts(  data=[opts.MarkLineItem(type\_="average", name="平均值")]# 标出每条折线的平均值  ),  #is\_smooth=True  )  .set\_global\_opts(  title\_opts=opts.TitleOpts(title="折线图-默认标记"),  xaxis\_opts=opts.AxisOpts(type\_="category", boundary\_gap=False),  )  )  chart.render\_notebook() |

运行后效果如下：

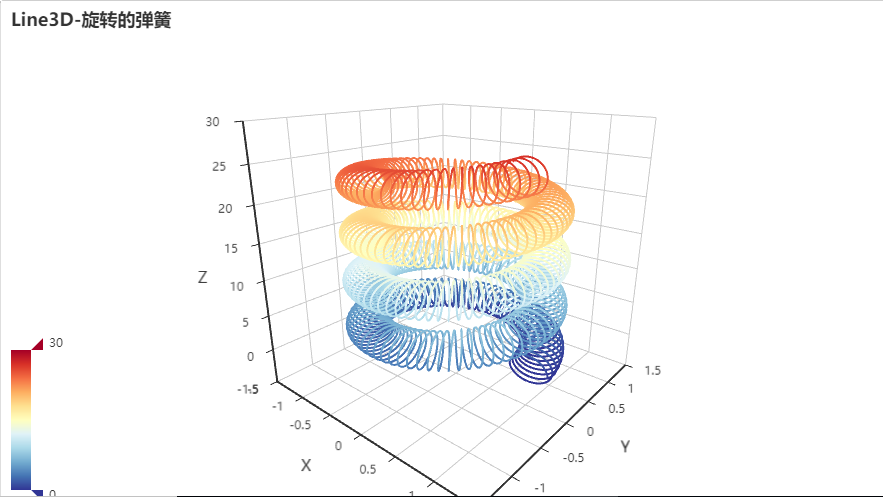


绘制3D折线图

代码如下：

|  |
| --- |
| import math  from pyecharts import options as opts  from pyecharts.charts import Line3D  from pyecharts.faker import Faker  data = []  for t in range(0, 25000):  \_t = t / 1000  x = (1 + 0.25 \* math.cos(75 \* \_t)) \* math.cos(\_t)  y = (1 + 0.25 \* math.cos(75 \* \_t)) \* math.sin(\_t)  z = \_t + 2.0 \* math.sin(75 \* \_t)  data.append([x, y, z])  c = (  Line3D()  .add(  "",  data,  xaxis3d\_opts=opts.Axis3DOpts(Faker.clock, type\_="value"),  yaxis3d\_opts=opts.Axis3DOpts(Faker.week\_en, type\_="value"),  grid3d\_opts=opts.Grid3DOpts(width=100, depth=100, rotate\_speed=50, is\_rotate=True),  )  .set\_global\_opts(  visualmap\_opts=opts.VisualMapOpts(max\_=30, min\_=0, range\_color=Faker.visual\_color),  title\_opts=opts.TitleOpts(title="Line3D-旋转的弹簧"),  )  )  c.render\_notebook() |

运行后效果如下：



### 数据准备

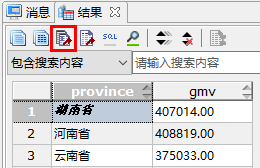
各省GMV数据

该数据主要用于完成下方玫瑰图的绘制。

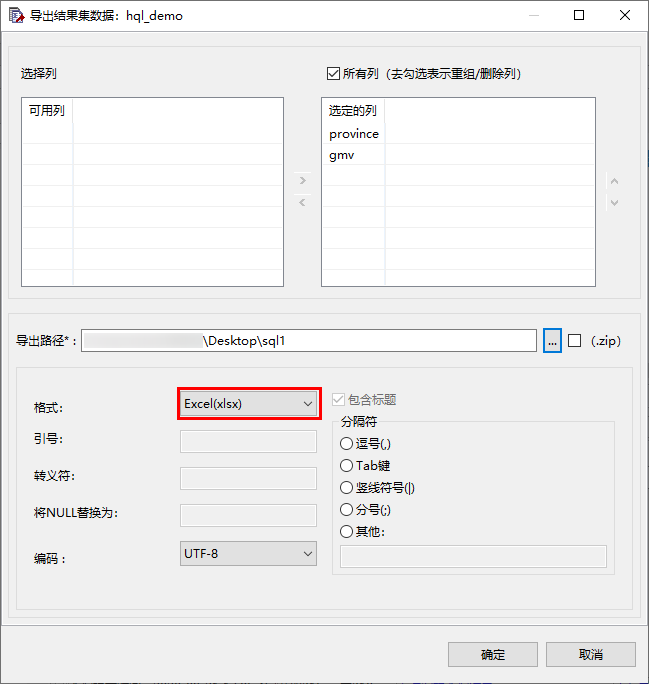
在DataStudio中输入如下命令：

|  |
| --- |
| SELECT ad.province AS province, SUM(o.actual\_price) AS GMV  FROM target.orders o, target.address\_dimension ad, target.date\_dimension dd  WHERE o.address\_key = ad.address\_key  AND o.add\_date = dd.date\_key  AND dd.year = 2020  AND dd.month = 3  GROUP BY ad.province; |

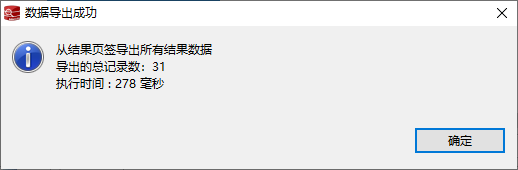
在DataStudio结果展示区选择“导出所有数据”。



在跳转到的导出界面，导出路径选择桌面，命名为sql1，格式选择xlxs，其他保持默认。



成功导出31条数据（含标题）。



各省用户数量数据

该数据主要用于在中国地图上完成热力图的绘制。

在DataStudio中输入如下命令：

|  |
| --- |
| SELECT ad.province, COUNT(DISTINCT o.user\_key) AS totaluser  FROM target.orders o, target.address\_dimension ad, target.date\_dimension dd  WHERE o.add\_date = dd.date\_key  AND o.address\_key = ad.address\_key  AND dd.year = 2020  GROUP BY ad.province  ORDER BY COUNT(DISTINCT o.user\_key) DESC; |

参考步骤1完成数据导出，数据文件命名为sql2.xlxs。

商品销量及销售金额数据

该数据主要用于在同一副图里完成折线图与柱状图的绘制。

在DataStudio中输入如下命令：

|  |
| --- |
| SELECT gd.goods\_sn, gd.goods\_name, SUM(og.number) AS totalnum, SUM(og.price\*og.number) AS totalsales  FROM target.order\_goods og  LEFT JOIN target.goods\_dimension gd ON og.goods\_key = gd.goods\_key  GROUP BY gd.goods\_sn, gd.goods\_name, gd.category\_name  ORDER BY SUM(og.number) DESC; |

参考步骤1完成数据导出，数据文件命名为sql3.xlxs。

各省市的GMV数据

该数据主要用于完成下钻及上卷操作。

在DataStudio中输入如下命令：

|  |
| --- |
| SELECT ad.province AS province, ad.city AS city, SUM(o.actual\_price) AS GMV  FROM target.orders o, target.address\_dimension ad, target.date\_dimension dd  WHERE o.address\_key = ad.address\_key  AND o.add\_date = dd.date\_key  AND to\_date(dd.full\_date) BETWEEN to\_date('2020/1/1') AND current\_date  AND province not in ('北京市', '上海市', '天津市', '重庆市')  GROUP BY ad.province, ad.city  UNION ALL  SELECT ad.province AS province, ad.county AS city, SUM(o.actual\_price) AS GMV  FROM target.orders o, target.address\_dimension ad, target.date\_dimension dd  WHERE o.address\_key = ad.address\_key  AND o.add\_date = dd.date\_key  AND to\_date(dd.full\_date) BETWEEN to\_date('2020/1/1') AND current\_date  AND province in ('北京市', '上海市', '天津市', '重庆市')  GROUP BY ad.province, ad.county; |

参考步骤1完成数据导出，数据文件命名为sql4.xlxs。

### 分析2020年3月各省GMV数据，绘制玫瑰图

将DWS分析后的结果导入到Python中

首先，导入通用的依赖包。

|  |
| --- |
| import pandas as pd  import pyecharts.options as opts  import numpy as np  import os |

这一步非常重要，否则后续的操作都会报错。

下面我们从本地读取之前从DWS里导出的结果，用于查询2020年3月各省GMV：

|  |
| --- |
| df1 = pd.read\_excel(os.path.join(os.path.expanduser('~'),'Desktop','sql1.xlsx'))#从本地读取excel文件，推荐使用pandas.read\_excel()方法。括号内添加的是文件的路径，推荐使用os.path.join方法，以免造成跨平台兼容性问题。  #'''1.指定表的列索引为 province和 GMV '''  df1.columns = ['province','GMV'] |

这样我们就成功把结果保存在df1里面了。

如果想查看df1的前5条数据，可以用：

|  |
| --- |
| df1.head(5) #查看前5条数据，重点关注每一列的行索引是否从1开始编号，以及列索引是否正确修改为province和 GMV。 |

显示结果为：

Out:

province GMV

1 湖南省 407014.00

2 河南省 408819.00

3 云南省 375033.00

4 江苏省 6217626.00

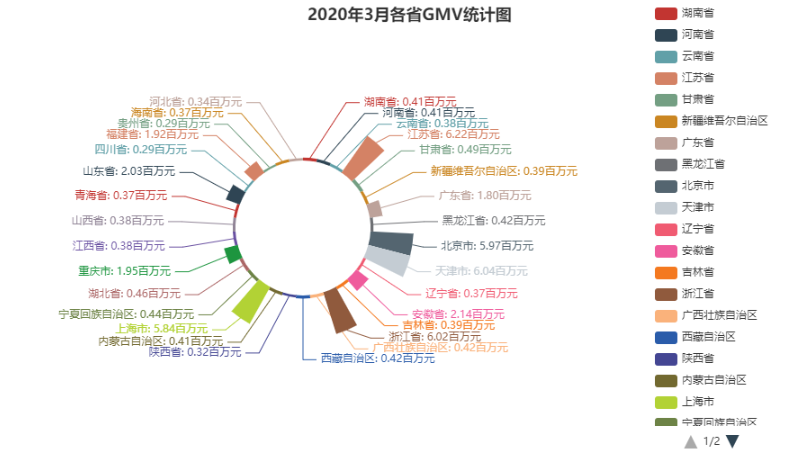
5 甘肃省 491934.00

使用Pyecharts完成可视化

这里我们选用玫瑰图来完成可视化操作。玫瑰图本质上还是一个饼图，所以我们需要调用Pie函数来完成相应的操作。

|  |
| --- |
| from pyecharts.charts import Pie #调用Pie函数，注意P需要大写  #把 province这一列的所有信息，也就是所有省份名称以列表的形式存入x中，df1['province'].values表示取出province这一列的数据  x = list(df1['province'].values)  #'''2.下一步需要把 GMV这一列的所有数据存入y中。'''  #由于数据太大，我们将其除以1,000,000，转化为以（百万元）为单位的数据，并保留两位小数。推荐使用列表生成式：  y = ['%.2f' % (x/1000000) for x in df1['GMV'].values]  #开始画图  pie\_2020\_MAR = (  Pie()  .add(  "",    #'''3.向data\_pair里传入数据：'''  data\_pair = [list(z) for z in zip(x, y)],    #'''4.指定内外直径：'''  radius = ["30%", "50%"],  center = ["40%", "50%"],#指定圆心所在的位置  rosetype="area", #area表示以环的大小来反映数据大小，也就是玫瑰图的特性  )  # 设定全局的配置项  .set\_global\_opts(    #'''5.调用opts.TitleOpts设定标题，标题为“2020年3月各省GMV统计图”：'''  title\_opts= opts.TitleOpts(title="2020年3月各省GMV统计图", pos\_left="40%"),    legend\_opts=opts.LegendOpts(  #设定图例，scroll表示可滚动查看，pos\_right表示距离图片右端的距离，orient="vertical"表示图例垂直排列  type\_="scroll", pos\_right="2%", orient="vertical"  )  )    #'''6.设定系列配置项，在括号内指定标签形式为 省份：数据+百万元 的形式'''  .set\_series\_opts( label\_opts=opts.LabelOpts(formatter="{b}: {c}百万元"))  )  pie\_2020\_MAR.render\_notebook() #在Jupyter Notebook里编译图像 |

至此，我们已完成了第一项数据分析+可视化的操作。看到图片效果如下，说明已正确完成以上9道题目。



创新指导小Tips:

学有余力的同学可以试一试在此加入一些定制化的内容。比如我们仅仅分析了2020年3月各省GMV的数据，但在我们的数据库中不仅仅只有3月的数据，那么其他月份的数据能否一起进行分析呢？

更进一步，我们在理论部分介绍了“**时间线轮播多图**”的基本操作，那么我们是否可以把1月到4月的四张玫瑰图，通过时间轴轮播的方式进行呈现呢？

### 分析各省用户数量，在中国地图上绘制热力图

完成分析，导入数据

有了上一小节的基础，这一部分就相对比较简单了。我们只需要修改相应导入数据的语句即可：

|  |
| --- |
| df2 = pd.read\_excel(os.path.join( os.path.expanduser('~'),'Desktop','sql2.xlsx'))  #'''7.指定表的列索引为 province和 totaluser '''  df2.columns = ['province','totaluser'] |

数据分析的结果就被保存在了df2里。

同样的，我们可以用 head(5) 命令查看前5条数据：

|  |
| --- |
| df2.head(5) #查看前5条数据，重点关注每一列的行索引是否从1开始编号，以及列索引是否正确修改为province和totaluser。 |

显示结果为：

Out:

province totaluser

1 天津市 1947

2 江苏省 1925

3 浙江省 1881

4 上海市 1878

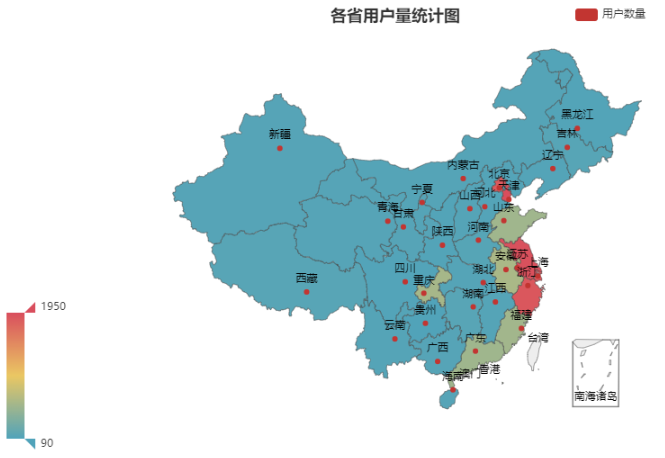
5 北京市 1859

使用Pyecharts完成可视化

这里我们选用地图来完成可视化操作。需要调用Map函数来完成相应的操作。

|  |
| --- |
| from pyecharts.charts import Map # 调用Map函数，注意M大写  # 把省份名称以列表方式写入x中。  # 由于Pyecharts默认的省份名称都是简写，因此我们需要把“省”、“市”、“xx族自治区”等信息都去掉，只保留简写。  # 数据中未包含港澳台地区，除了黑龙江和内蒙古的简写是三个字，其余都是两个字，因此可直接按需截取：  x = [p[:2] if p not in ['黑龙江省','内蒙古自治区'] else p[:3] for p in df2['province'].values]  # 把用户数量这一列的所有信息存入y中。注意要强制转化为int类型，否则numpy.int64会无法正常显示。  y = [int(x) for x in df2['totaluser'].values]  map\_user\_num = (  Map()  .add(  "用户数量", #指定系列名称    #'''8.请学员完成以下内容，向data\_pair里传入数据：'''  data\_pair = [list(z) for z in zip(x, y)],    maptype='china', #指定地图类型是中国地图    #'''9.设定显示标签：'''  label\_opts= opts.LabelOpts(is\_show=True),  )  .set\_global\_opts(  #'''10.调用opts.TitleOpts设定标题及其位置，标题为“各省用户量统计图”：'''  title\_opts= opts.TitleOpts(title="各省用户量统计图", pos\_left="40%"),    #'''11.使用opts.VisualMapOpts，指定 color bar的显示范围为90到1950：'''  visualmap\_opts= opts.VisualMapOpts(min\_=90,max\_=1950),    #'''12.使用opts.LegendOpts，指定图例的右端离画布的右端大约20%的距离：'''  legend\_opts= opts.LegendOpts(pos\_right="20%")  )  )  map\_user\_num.render\_notebook() |

可以看到如下结果，说明这7道题都已经全部做对了。



创新指导小Tips:

我们这里只给出了各省的用户数据信息，同学们可以尝试一下深入到各市，绘制一下你家乡的用户信息；更进一步，我们可以做一个下钻操作，比如点击某个省，就能显示对应各市的用户数量信息。

### 分析各商品销量及销售金额，同一副图里显示折线图与柱状图

完成分析，导入数据

再次修改数据导入语句：

|  |
| --- |
| df3 = pd.read\_excel(os.path.join( os.path.expanduser('~'),'Desktop','sql3.xlsx'))  #'''13.指定表的列索引为 goods\_sn,goods\_name,totalnum,totalsales '''  df3.columns = ['goods\_sn','goods\_name','totalnum','totalsales'] |

数据分析的结果就被保存在了df3里。

用 head(5) 命令查看前5条数据：

|  |
| --- |
| df3.head(5) #查询df3的前5条数据。修改括号内的数字可修改查询的数据量 |

显示结果为：

Out:

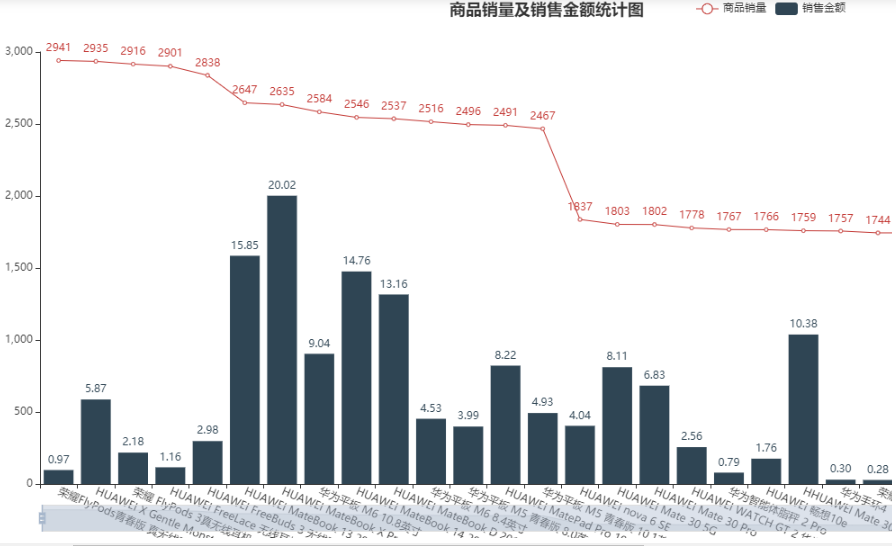


使用Pyecharts完成可视化

这里我们选用折线图来展示销售量，用柱状图来展示销售金额。对应地，需要调用Line函数和Bar函数来完成相应的操作。

|  |
| --- |
| from pyecharts.charts import Line,Bar  #'''14.将df3中的goods\_name这一列以列表的形式赋值给x：'''  x = list(df3['goods\_name'].values)  #'''15.将df3中的totalnum这一列以列表的形式赋值给y1，并将每一个元素转化为int格式：'''  y1 = [int(ele) for ele in df3['totalnum'].values]  #'''将df3中的totalsales这一列以列表的形式赋值给y1，并将每一个元素除以1,000,000，转化为百万元的格式，并保留两位小数：'''  y2 = ['%.2f' % (ele/1000000) for ele in df3['totalsales'].values]  #先绘制折线图：  line1=(  #设计风格  #'''17.调用opts.InitOpts方法，将画布的宽设为1500px，高度设为600px：'''  Line(init\_opts = opts.InitOpts(width='1500px',height='600px'))    .add\_xaxis(x) #添加x轴数据  .add\_yaxis('商品销量',y1) #将y1数据添加到y轴，注意'商品销量'指定了系列名称，如果不想要系列名称，需要用空字符串''来填充  #下面的命令其实是添加了一个新的y轴，因为销售金额和销量的数量级相差太多，因此需要单独的纵轴。  #注意，添加新的纵坐标，需要在第一次初始化图的时候就完成，而不是等到绘制新数据的时候再进行  #比如在这里，虽然我们还没开始绘制销售金额，但对应的纵坐标需要先添加完成：  .extend\_axis(yaxis=opts.AxisOpts(axislabel\_opts=opts.LabelOpts(formatter="{value} (百万元)")))  .set\_global\_opts( #全局配置项  #设置全局参数  datazoom\_opts=opts.DataZoomOpts(range\_end=500,is\_zoom\_lock=False), #允许缩放  title\_opts=opts.TitleOpts(title="商品销量及销售金额统计图", pos\_left="40%"),#设置title    #'''18.调用opts.LabelOpts方法，将x轴上的标签顺时针旋转23度'''  xaxis\_opts=opts.AxisOpts(axislabel\_opts= opts.LabelOpts(rotate=-23)),    legend\_opts=opts.LegendOpts(pos\_right="30%")  )  )  bar2=(  #'''19.请完成柱状图的绘制，包括  #1）使用Bar方法创建柱状图对象，并调用opts.InitOpts方法，将画布的宽设为1500px，高度设为600px；  #2）添加x轴的数据；  #3）将y2数据添加到y轴，注意需要指定yaxis\_index=1：'''  Bar(init\_opts = opts.InitOpts(width='1500px',height='500px'))  .add\_xaxis(x)  .add\_yaxis('销售金额',y2,yaxis\_index=1)  )  line1.overlap(bar2) #调用overlap函数就可以完成同一张图里显示折线图和柱状图了  line1.render\_notebook() |

同样的，如果可以看到如下结果，说明这7道题都已经全部做对了。



创新指导小Tips:

我们这里是把折线图和柱状图进行了整合，那么还有没有其他的整合方式呢？

或者，我们在PPT中展示了“选项卡多图”的操作，在这里感兴趣的同学还可以把柱状图和折线图作为选项卡多图进行呈现。

### 下钻与上卷

下钻与上卷是BI数据可视化中一项非常强大的操作，它把不同的图表。本指导书内仅完成一层下钻操作。然而，由于Pyecharts里没有封装对应Echarts下钻操作的函数或方法，因此这一部分内容需要我们自己来完成HTML源文件的修改。具体而言，我们将鼠标点击事件对应的JavaScript代码添加到所生成的HTML源文件中，以此来完成下钻操作的编写。

这一部分需要学员有一定的JavaScript代码基础，但完全不懂前端开发的同学也完全不用担心，我们所用到的都是非常简单的Python文件读写操作。而对Python文件读写有任何疑问的同学，可以尝试完成《Python基础操作练习手册》中的相关代码，相信会有比较大的帮助。

完成分析，导入数据

本小节内，我们用各省及对应的各市的总GMV值来演示下钻与上卷操作。首先，修改相应的SQL语句，读取对应的数据：

|  |
| --- |
| df4 = pd.read\_excel(os.path.join( os.path.expanduser('~'),'Desktop','sql4.xlsx'))  #'''20.指定表的列索引为 province, city, GMV '''  df4.columns = ['province','city','GMV'] |

用 head(5) 命令查看前5条数据：

|  |
| --- |
| df4.head(5) #查询df4的前5条数据。修改括号内的数字可修改查询的数据量 |

显示结果为：

Out[322]:



分析HTML文件构成，理解下钻操作的例子。

首先，我们画出各省对应的总GMV柱状图，作为图表下钻操作的第一级。在步骤一中我们可以看到，表中的GMV数据其实对应了每个市，所以在绘制各省对应的总GMV时，需要对每个省的所有地级市或市辖区的数据进行求和。在Pandas中，可以用groupby方法将每个省对应的数据进行汇总，而sum方法则可以对GMV数据进行求和：

|  |
| --- |
| #从原始数据中获取各省对应的数据：  #groupby方法表示把关键词相同的数据视为一个整体，比如这里我们把'province'相同的数据放在了一起，然后取出其中名为'GMV'的那一列  #sum方法就是对所有元素进行求和，也就是对GMV数据进行求和。  df\_GMV\_province = df4.groupby(['province'])['GMV'].sum()  #'''21.将省份名称取出，作为列表传入province\_name里'''  province\_name = list(df\_GMV\_province.index)  print(province\_name) #输出看一下是否正确显示为列表。注意每个省份只出现一次。  #'''22.将GMV数据取出，转化为百万元，保留两位小数，作为列表传入GMV\_province里'''  GMV\_province = ['%.2f' % (x/1000000) for x in df\_GMV\_province.values]  print(GMV\_province) #输出看一下是否正确显示为列表，并且是否已经保留两位小数。 |

输出结果如下所示：

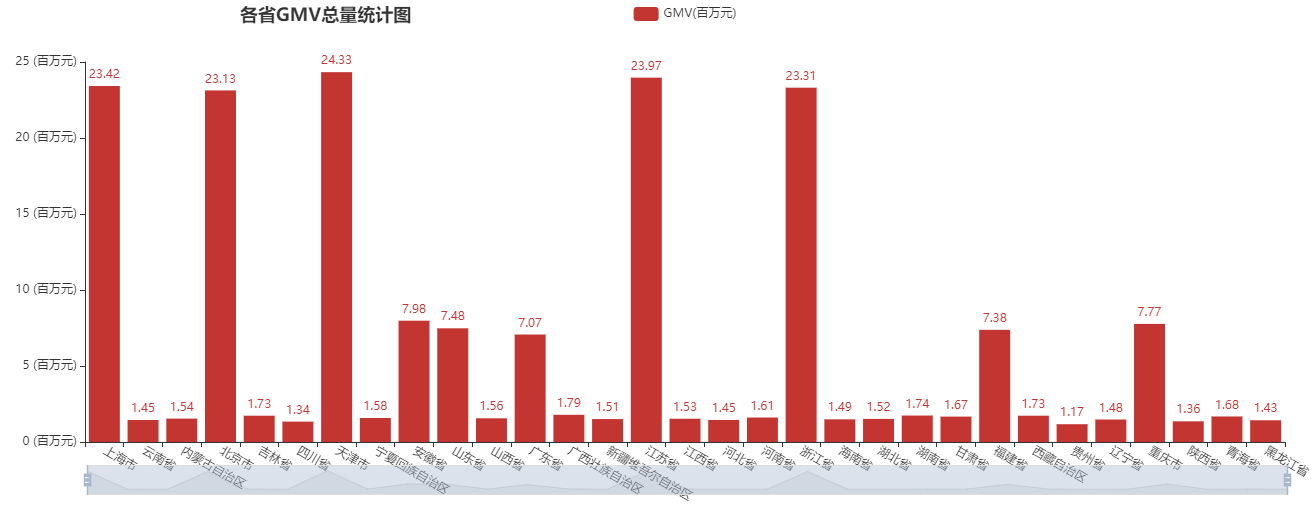
['上海市', '云南省', '内蒙古自治区', '北京市', '吉林省', '四川省', '天津市', '宁夏回族自治区', '安徽省', '山东省', '山西省', '广东省', '广西壮族自治区', '新疆维吾尔自治区', '江苏省', '江西省', '河北省', '河南省', '浙江省', '海南省', '湖北省', '湖南省', '甘肃省', '福建省', '西藏自治区', '贵州省', '辽宁省', '重庆市', '陕西省', '青海省', '黑龙江省']

['23.42', '1.45', '1.54', '23.13', '1.73', '1.34', '24.33', '1.58', '7.98', '7.48', '1.56', '7.07', '1.79', '1.51', '23.97', '1.53', '1.45', '1.61', '23.31', '1.49', '1.52', '1.74', '1.67', '7.38', '1.73', '1.17', '1.48', '7.77', '1.36', '1.68', '1.43']

有了数据，我们就可以绘制柱状图了：

|  |
| --- |
| #开始画图  bar\_province = (  #'''23.绘制各省GMV总量柱状图，包括：  # 1) 初始化画布，宽1500px，高500px；'''  Bar(init\_opts = opts.InitOpts(width='1500px',height='500px'))  # 2) 添加省份名称为x轴数据；  .add\_xaxis(province\_name)  # 3) 添加GMV\_province数据到y轴，指定系列名称为：GMV(百万元)'''  .add\_yaxis('GMV(百万元)',GMV\_province)  .set\_global\_opts(  title\_opts=opts.TitleOpts(title="各省GMV总量统计图",pos\_left='20%'), #设置标题  datazoom\_opts=opts.DataZoomOpts(range\_end=500,is\_zoom\_lock=False), #允许缩放  yaxis\_opts=opts.AxisOpts(axislabel\_opts=opts.LabelOpts(formatter = '{value} (百万元)')), #设置标签类型为： 数值（百万元）  #'''4) 设定标签的配置项，将x轴上的标签顺时针旋转30度'''  xaxis\_opts=opts.AxisOpts(axislabel\_opts= opts.LabelOpts(rotate=-30)),  ))  bar\_province.render(os.path.join(os.path.expanduser('~'),'Desktop','bar\_province.html')) #编译为HTML文件，注意此时必须完成编译 |

找到bar\_province.html这个文件，用浏览器打开就可以看到结果了。



但是，此时我们并不能完成下钻，所以我们还需要添加各省所辖地级市或市辖区的信息。在本指导手册里，由于Pyecharts没有集成下钻函数，因此需要我们手动添加JavaScript (JS) 操作到HTML文件中。

右键点击bar\_province.html，在打开方式里选择用记事本打开，查看对应的源代码。经过分析，我们发现，其实整个文件的主体只包含四个部分：1. JS修饰符；2. 画布生成； 3. 添加数据与配置信息；4. 将数据和配置信息传递给画布，从而在画布上显示对应的图表。

|  |
| --- |
| <!DOCTYPE html>  <html>  这些都是JS修饰符或者声明文件主体的限制符，对我们的下钻操作没有影响。  <head>  <meta charset="UTF-8">  <title>Awesome-pyecharts</title>  <script type="text/javascript" src="https://assets.pyecharts.org/assets/echarts.min.js"></script>  </head>  <body>  <div id="f4f0d1aa5baf4c56ade8afa7d1c82452" class="chart-container" style="width:1500px; height:500px;"></div>  <script>  var chart\_f4f0d1aa5baf4c56ade8afa7d1c82452 = echarts.init(  document.getElementById('f4f0d1aa5baf4c56ade8afa7d1c82452'), 'white', {renderer: 'canvas'});  这些是生成画布的操作，由于我们需要保留画布，因此这些信息非常重要。其中f4f0d1aa5baf4c56ade8afa7d1c82452是该画布的id，每一次生成图表，id都不同。  var option\_f4f0d1aa5baf4c56ade8afa7d1c82452 = {  "animation": true,  "animationThreshold": 2000,  "animationDuration": 1000,  "animationEasing": "cubicOut",  "animationDelay": 0,  "animationDurationUpdate": 300,  "animationEasingUpdate": "cubicOut",  "animationDelayUpdate": 0,  以下是添加数据与相应的配置项。  可以看到，所有的数据和配置信息都被赋值给了option\_f4f0d1aa5baf4c56ade8afa7d1c82452这个变量。同样的，f4f0d1aa5baf4c56ade8afa7d1c82452是对应的id。  "color": [  "#c23531",  "#2f4554",  "#61a0a8",  "#d48265",  "#749f83",  "#ca8622",  "#bda29a",  "#6e7074",  "#546570",  "#c4ccd3",  "#f05b72",  "#ef5b9c",  "#f47920",  "#905a3d",  "#fab27b",  "#2a5caa",  "#444693",  "#726930",  "#b2d235",  "#6d8346",  "#ac6767",  "#1d953f",  "#6950a1",  "#918597"  ],  "series": [  {  "type": "bar",  "name": "GMV(\u767e\u4e07\u5143)",  "data": [  "32.83",  "2.30",  "1.57",  "32.47",  "2.42",  "1.76",  "32.33",  "2.16",  "9.64",  "10.66",  "1.63",  "9.33",  "2.18",  "2.45",  "32.85",  "1.87",  "1.77",  "2.47",  "33.23",  "2.50",  "2.05",  "2.05",  "1.98",  "9.53",  "2.46",  "2.37",  "2.35",  "10.00",  "2.13",  "1.95",  "1.94"  ],  "barCategoryGap": "20%",  "label": {  "show": true,  "position": "top",  "margin": 8  }  }  ],  "legend": [  {  "data": [  "GMV(\u767e\u4e07\u5143)"  ],  "selected": {  "GMV(\u767e\u4e07\u5143)": true  },  "show": true,  "padding": 5,  "itemGap": 10,  "itemWidth": 25,  "itemHeight": 14  }  ],  "tooltip": {  "show": true,  "trigger": "item",  "triggerOn": "mousemove|click",  "axisPointer": {  "type": "line"  },  "textStyle": {  "fontSize": 14  },  "borderWidth": 0  },  "xAxis": [  {  "show": true,  "scale": false,  "nameLocation": "end",  "nameGap": 15,  "gridIndex": 0,  "axisLabel": {  "show": true,  "position": "top",  "rotate": -30,  "margin": 8  },  "inverse": false,  "offset": 0,  "splitNumber": 5,  "minInterval": 0,  "splitLine": {  "show": false,  "lineStyle": {  "show": true,  "width": 1,  "opacity": 1,  "curveness": 0,  "type": "solid"  }  },  "data": [  "\u4e0a\u6d77\u5e02",  "\u4e91\u5357\u7701",  "\u5185\u8499\u53e4\u81ea\u6cbb\u533a",  "\u5317\u4eac\u5e02",  "\u5409\u6797\u7701",  "\u56db\u5ddd\u7701",  "\u5929\u6d25\u5e02",  "\u5b81\u590f\u56de\u65cf\u81ea\u6cbb\u533a",  "\u5b89\u5fbd\u7701",  "\u5c71\u4e1c\u7701",  "\u5c71\u897f\u7701",  "\u5e7f\u4e1c\u7701",  "\u5e7f\u897f\u58ee\u65cf\u81ea\u6cbb\u533a",  "\u65b0\u7586\u7ef4\u543e\u5c14\u81ea\u6cbb\u533a",  "\u6c5f\u82cf\u7701",  "\u6c5f\u897f\u7701",  "\u6cb3\u5317\u7701",  "\u6cb3\u5357\u7701",  "\u6d59\u6c5f\u7701",  "\u6d77\u5357\u7701",  "\u6e56\u5317\u7701",  "\u6e56\u5357\u7701",  "\u7518\u8083\u7701",  "\u798f\u5efa\u7701",  "\u897f\u85cf\u81ea\u6cbb\u533a",  "\u8d35\u5dde\u7701",  "\u8fbd\u5b81\u7701",  "\u91cd\u5e86\u5e02",  "\u9655\u897f\u7701",  "\u9752\u6d77\u7701",  "\u9ed1\u9f99\u6c5f\u7701"  ]  }  ],  "yAxis": [  {  "show": true,  "scale": false,  "nameLocation": "end",  "nameGap": 15,  "gridIndex": 0,  "axisLabel": {  "show": true,  "position": "top",  "margin": 8,  "formatter": "{value} (\u767e\u4e07\u5143)"  },  "inverse": false,  "offset": 0,  "splitNumber": 5,  "minInterval": 0,  "splitLine": {  "show": false,  "lineStyle": {  "show": true,  "width": 1,  "opacity": 1,  "curveness": 0,  "type": "solid"  }  }  }  ],  "title": [  {  "text": "\u5404\u7701GMV\u603b\u91cf\u7edf\u8ba1\u56fe",  "left": "20%",  "padding": 5,  "itemGap": 10  }  ],  "dataZoom": {  "show": true,  "type": "slider",  "realtime": true,  这条语句就是把数据和配置项提交给画布，在画布上显示出我们的图表。这说明，使用Echarts绘图的时候，就只需要这四大块的语句就可以了。  "start": 20,  "end": 500,  "orient": "horizontal",  "zoomLock": false  }  };  chart\_f4f0d1aa5baf4c56ade8afa7d1c82452.setOption(option\_f4f0d1aa5baf4c56ade8afa7d1c82452);  </script>  </body>  </html> |

而下钻以及上卷操作的本质，其实就是在同一张画布上生成了一个新的图表。通过分析以上代码，我们发现，如果把不同的option传递给同一个画布，是不是就可以在同一张画布上显示不同的图表了呢？这就是我们完成下钻上卷操作的核心思想，就是清空原始画布，然后添加新的数据及配置信息。

有了以上的分析，想要完成下钻操作就有了思路：我们只需要写入一个鼠标点击事件，通过判断所点击的数据属于哪一个省，相应的把该省下划所有区市的信息展示出来，就可以完成下钻了。

如何写入上述操作？我们可以直接把bar\_province.html这个文件以只读的方式读取到Python中，然后我们就可以用Python修改其中的JS语句了！

比如说，拿浙江省为例，我们新代码的框架就应该是以下的内容。（**从这里开始直到步骤3，这几部分代码不需执行，仅为方便理解。直到步骤3，学员们均无需完成以下代码的实现。**）

|  |
| --- |
| # 绘制浙江省所管辖的各市区GMV统计图，编译到本地 bar\_city.html中：  bar\_city = (Bar(init\_opts = opts.InitOpts(width='1500px',height='500px'))  .add\_xaxis(city\_name)  .add\_yaxis('GMV(百万元)',GMV\_city)  .set\_global\_opts(  title\_opts=opts.TitleOpts(title="%s各市（区）GMV总量统计图" %p, pos\_left='20%'),  datazoom\_opts=[opts.DataZoomOpts(range\_end=500,is\_zoom\_lock=False)],  xaxis\_opts=opts.AxisOpts(axislabel\_opts=opts.LabelOpts(rotate=-30)),  yaxis\_opts=opts.AxisOpts(axislabel\_opts=opts.LabelOpts(formatter = '{value} (百万元)')),  ))  bar\_city.render( os.path.join(os.path.expanduser('~'),'Desktop','bar\_city.html')) #编译 |

先绘制好浙江省所管辖的各市区GMV统计图，这样我们就可以直接使用Pyecharts帮我们编译好的option\_[id]的内容，而不需要自己编写了。

下面就是完成下钻和上卷。通过调用.on(“click”)方法，我们可以编写一个鼠标单击事件，而事件的结果是一个函数，所以我们要用以下的代码：

|  |
| --- |
| id\_province = bar\_province.chart\_id #获取全国各省GMV统计图的id，以便完成后续的上卷下钻操作  id\_city = bar\_city.chart\_id #获取浙江省所管辖的各市区GMV统计图的id，以便完成后续的上卷下钻操作  f2 = open(os.path.join(os.path.expanduser('~'),'Desktop','bar\_city.html'),'r') #读取浙江省所管辖的各市区GMV统计图的代码，方便将数据和配置信息插入到新的文件中  content2 = f2.readlines()#使用readlines方法，将读取后的文件内容保存为列表形式，每一行是列表中的一个元素。  f2.close() #使用open函数后一定要记得close，否则会一直占用该文件，造成‘permission denied’错误，影响下一次操作。  #下面是新添加的内容，也就是完整的鼠标点击事件。  add\_in = [' chart\_%s.on("click", function (params) {\n'% id\_province]+\ #.on表示一个触发事件，"click"表示鼠标单击，function (params) 表示这是一个函数，所需要的是params这些参数，其中就有所点击数据对应的name  [' if (params.name=="浙江省"){\n']+\ #如果数据的name等于“浙江省”，说明我们鼠标单击了“浙江省”对应的那条柱子，于是开始下列操作。  [' chart\_%s.clear();\n'% id\_province]+\ #调用clear方法清空画布  content2[13:-4]+\ #加入bar\_city.html的option变量的内容，也就是浙江省所管辖的各市区GMV统计图所有的数据以及配置信息  [' chart\_%s.setOption(option\_%s);\n'% (id\_province,id\_city)]+\ #将新的option传递给画布，完成数据的可视化  [' }\n']+\ #if语句的结尾  [' else {\n']+\ #从这里开始是上卷。只要鼠标点击的不是省份名称，就返回到初始界面。  [' chart\_%s.clear();\n'% id\_province]+\ #再次清空画布  [' chart\_%s.setOption(option\_%s);\n'% (id\_province,id\_province)]+\ #恢复bar\_province.html的option变量的内容，也就是回到全国各省的统计图  [' };\n']+\ #else语句的结尾  [' } );\n'] #function的结尾以及.on触发事件的结尾 |

这样我们就编写好了这一个鼠标点击事件了。下面需要做的，就是把以上的代码插入到bar\_province.html的倒数第4行，也就是“chart\_id.setOption(option\_id)”这条语句的后面：

|  |
| --- |
| f1 = open(os.path.join(os.path.expanduser('~'),'Desktop','bar\_province.html'),'r') #打开省份柱状图的HTML源文件  content = f1.readlines()  f1.close()  new\_content = [ele for ele in content[:-3]] +add\_in+ [ele for ele in content[-3:]] #新HTML文件就是在 bar\_province.html文件的倒数第4行插入我们的鼠标点击事件，以及所有的新图表的数据与配置项。  #创建一个新的HTML文件，将我们的内容写进该文件中  #注意，'w'表示只写，如果目录下存在同名文件是会被新文件覆盖的。  f3 = open( os.path.join(os.path.expanduser('~'),'Desktop','drill\_down.html'),'w',encoding='utf-8-sig') #encoding='utf-8-sig'为了保证中文显示正确  f3.writelines(new\_content) #使用writelines方法，把一整个列表一次性写入 drill\_down.html 文件中  f3.close() #别忘了解除占用 |

以上就是针对浙江省的上钻下卷代码演示。

对所有省份进行类似的操作

在步骤2中，我们仅仅针对浙江省完成了上钻下卷的代码演示。而对每个省来说，基本操作都是类似的，因此我们可以写一个循环，遍历每个省份；每一次循环的时候，只需要把对应的省份名称修改成对应的字符串p，然后将每个省下划区市的信息都写入HTML中，就可以完成我们的操作了。

具体而言，如下所示（**注意！从这里开始需要学员继续完成操作**）：

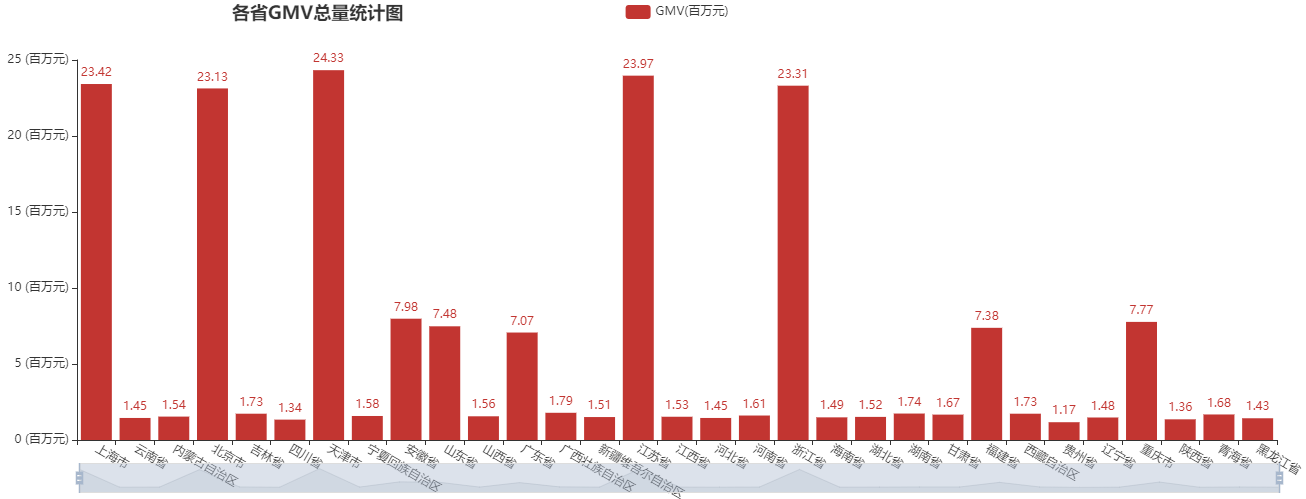
|  |
| --- |
| df\_GMV\_cities = [] #初始化一个空列表，用于存入每个省份下划区市对应的数据  for p in province\_name: #遍历每一个省份  df\_GMV\_cities.append(df4[df4['province']==p].groupby(['city'])['GMV'].sum())    id\_province = bar\_province.chart\_id #获取当前柱状图的id，以便完成后续的上卷下钻操作  #编写触发条件：鼠标点击事件，由于鼠标点击事件仅有一个，所以放在循环外：  add\_in = [' chart\_%s.on("click", function (params) {\n'% id\_province] |

下面开始开始编写上卷下钻，欢迎学有余力的同学深入进行探索。

注意，该部分代码需要在完成步骤1和步骤2中的bar\_province.html绘制之后才能进行。

|  |
| --- |
| import numpy as np  for i,p in enumerate(province\_name):  #对每一个省份都写入下钻操作的相关代码。由于下钻操作的原理相同，因此可用循环来完成。  city\_name = list(df\_GMV\_cities[i].index) #地级市（市辖区）名称  GMV\_city = ['%.2f' % (x/1000000) for x in df\_GMV\_cities[i].values] #对应的GMV数据  sorted\_index = [i for i,v in sorted(enumerate(GMV\_city), key=lambda x:x[1])] #对数据进行排序，保存排序后的索引  city\_name = list(np.array(city\_name)[sorted\_index]) #按照索引调整地级市（市辖区）名称  GMV\_city = list(np.array(GMV\_city)[sorted\_index]) #按照索引调整GMV数据    #画图：  bar\_city = (  Bar(init\_opts = opts.InitOpts(width='1500px',height='500px'))  .add\_xaxis(city\_name)  .add\_yaxis('GMV(百万元)',GMV\_city)  .set\_global\_opts(  title\_opts=opts.TitleOpts(title="%s各市（区）GMV总量统计图" %p, pos\_left='20%'),  datazoom\_opts=[opts.DataZoomOpts(range\_end=500,is\_zoom\_lock=False)],  xaxis\_opts=opts.AxisOpts(axislabel\_opts=opts.LabelOpts(rotate=-30)),  yaxis\_opts=opts.AxisOpts(axislabel\_opts=opts.LabelOpts(formatter = '{value} (百万元)')),  )  )  bar\_city.render(os.path.join(os.path.expanduser('~'),'Desktop','bar\_city.html')) #编译    f2 = open(os.path.join(os.path.expanduser('~'),'Desktop','bar\_city.html'),'r') #打开p这个省下属各地级市（市辖区）对应柱状图的HTML源文件。  content2 = f2.readlines()  f2.close() #使用open函数后一定要记得close。  id\_city = bar\_city.chart\_id #获取当前柱状图的id，以便完成后续的上卷下钻操作  if i == 0 : #对于第一次写入，我们要写入 if 条件；而对于剩下的，我们要写入 else if 条件。  add\_in = add\_in+\  [' if (params.name=="%s"){\n' % p]+\  [' chart\_%s.clear();\n'% id\_province]+\  content2[13:-4]+\  [' chart\_%s.setOption(option\_%s);\n'% (id\_province,id\_city)]+\  [' }\n']  else:  add\_in = add\_in+\  [' else if (params.name=="%s"){\n' % p]+\  [' chart\_%s.clear();\n'% id\_province]+\  content2[13:-4]+\  [' chart\_%s.setOption(option\_%s);\n'% (id\_province,id\_city)]+\  [' }\n']  #以上是下钻的全部内容  #下面完成上卷。只要鼠标点击的不是省份名称，就返回到初始界面。  add\_in = add\_in+\  [' else {\n']+\  [' chart\_%s.clear();\n'% id\_province]+\  [' chart\_%s.setOption(option\_%s);\n'% (id\_province,id\_province)]+\  [' };\n']+\  [' });\n']    f1 = open( os.path.join(os.path.expanduser('~'),'Desktop','bar\_province.html'),'r') #打开省份柱状图的HTML源文件  content = f1.readlines()  f1.close()  new\_content = [ele for ele in content[:-3]] +add\_in+\  [ele for ele in content[-3:]] #新HTML文件就是在 bar\_province.html文件的倒数第四行插入我们的鼠标点击事件以及所有的新图表的数据与配置项。  #创建一个新的HTML文件，将我们的内容写进该文件中  #注意，'w'表示只写，如果目录下存在同名文件是会被新文件覆盖的。  f3 = open( os.path.join(os.path.expanduser('~'),'Desktop','drill\_down.html'),'w',encoding='utf-8-sig') #encoding='utf-8-sig'为了保证中文显示正确  f3.writelines(new\_content) #使用writelines方法，把一整个列表一次性写入 drill\_down.html 文件中  f3.close() #别忘了解除占用 |

通过以上的代码，我们终于实现了下钻上卷的操作。在桌面找到drill\_down.html这个文件，用浏览器打开就可以看到结果了。点击每一条柱子，看一看是否能正常执行。



创新指导小Tips:

我们这里仅仅是完成了柱状图的上卷下钻。感兴趣的同学们还可以尝试一下，如果用地图进行上卷下钻，会有哪些不同呢？

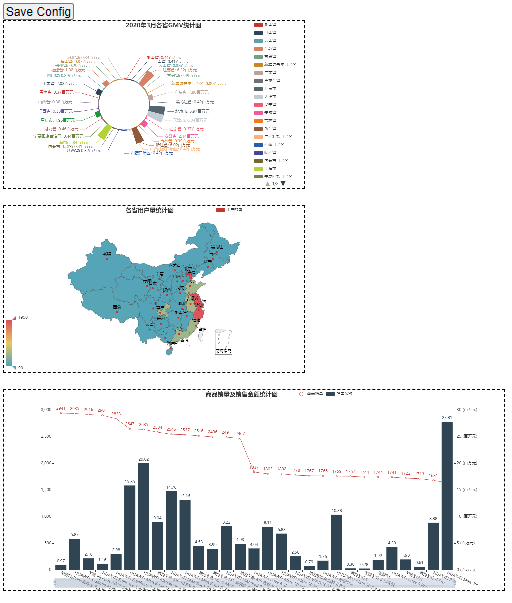
### 利用Page函数，完成可视化大屏的搭建。

将图表添加至大屏中

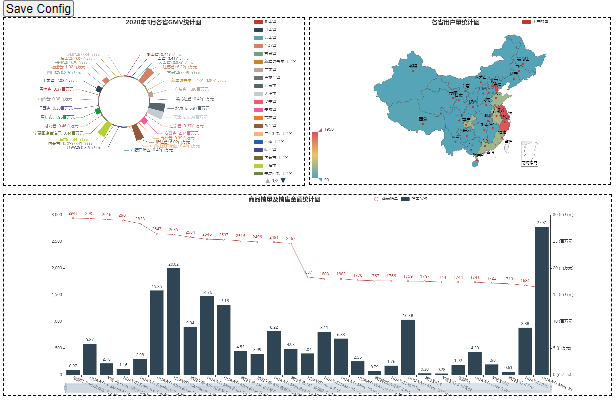
直接调用Page函数，就可以把图表放入大屏中了：

|  |
| --- |
| from pyecharts.charts import Page  page = Page(layout=Page.DraggablePageLayout) # DraggablePageLayout表示可拖拽布局  page.add(pie\_2020\_MAR,map\_user\_num,line1) # 注意，这里没有drill\_down这张图  page.render( os.path.join(os.path.expanduser('~'),'Desktop','page1.html')) |

这里我们用到了DraggablePageLayout 属性，其实打开page1.html我们就能发现，所有的图表周围都有一圈虚线：



这里的虚线就表示可以拖拽和放大缩小，比如我们可以调整前两张图的位置，并且把第三张图缩小：



不过这里需要注意，在放大缩小的时候，Echarts仅仅允许我们拖拽图片右侧和下端的虚线，左侧和上端的虚线是不可修改的；另外我们的设置不能自动保存，需要点击左上角的Save Config按键，下载一个chart\_config.json 文件，然后把该文件复制粘贴到桌面，通过以下命令，生成新的HTML文件才行：

|  |
| --- |
| Page.save\_resize\_html(os.path.join(os.path.expanduser('~'),'Desktop','page1.html'), cfg\_file= os.path.join(os.path.expanduser('~'),'Desktop','chart\_config.json'), dest= os.path.join(os.path.expanduser('~'),'Desktop',‘my\_visualization.html’)) |

page1.html就是我们原始的大屏文件。

**不过，这一步骤中我们不推荐大家进行页面布局，因为我们的drill\_down.html还没有放进来呢。等到完成下一步骤后，同学们就可以进行页面布局，最终输出我们的可视化大屏了。**

往page1.html里添加drill\_down.html配置信息。

由于Page函数只能添加未编译的图表，而drill\_down.html已经是编译后的文件了，因此我们又一次需要修改HTML文件的原始代码了。

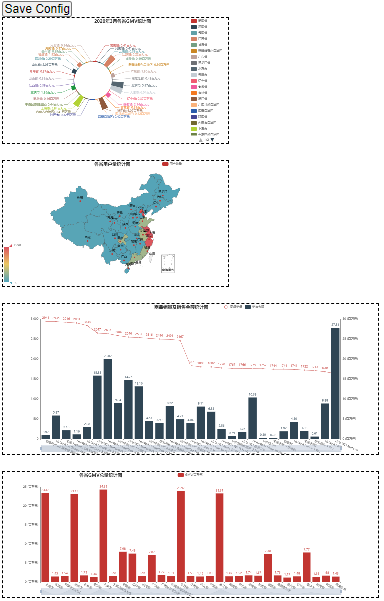
用记事本打开page1.html文件，查看代码如下：

|  |
| --- |
| <!DOCTYPE html>  这一块代码是HTML声明信息，以及一些脚本的配置信息，我们无需关注太多。  <html>  <head>  <meta charset="UTF-8">  <title>Awesome-pyecharts</title>  <script type="text/javascript" src="https://assets.pyecharts.org/assets/echarts.min.js"></script>  <script type="text/javascript" src="https://assets.pyecharts.org/assets/maps/china.js"></script>  <script type="text/javascript" src="https://assets.pyecharts.org/assets/jquery.min.js"></script>  <script type="text/javascript" src="https://assets.pyecharts.org/assets/jquery-ui.min.js"></script>  <script type="text/javascript" src="https://assets.pyecharts.org/assets/ResizeSensor.js"></script>  <link rel="stylesheet" href="https://assets.pyecharts.org/assets/jquery-ui.css">  </head>  <body>  <style>.box { }; </style>  <button onclick="downloadCfg()">Save Config</button>  <div class="box">  <div id="42c7719c0f6d40929f41a9c0f5dffd74" class="chart-container" style="width:900px; height:500px;"></div>  <script>  var chart\_42c7719c0f6d40929f41a9c0f5dffd74 = echarts.init(  document.getElementById('42c7719c0f6d40929f41a9c0f5dffd74'), 'white', {renderer: 'canvas'});  这是第一个图表的声明过程，我们需要插入的的就是这样的代码。  var option\_42c7719c0f6d40929f41a9c0f5dffd74 = {  #这里省略具体数据和配置信息  };  chart\_42c7719c0f6d40929f41a9c0f5dffd74.setOption(option\_42c7719c0f6d40929f41a9c0f5dffd74);  </script>  <br/> <div id="c0ec27a87bc34ac6a3af4169849d4127" class="chart-container" style="width:900px; height:500px;"></div>  <script>  var chart\_c0ec27a87bc34ac6a3af4169849d4127 = echarts.init(  document.getElementById('c0ec27a87bc34ac6a3af4169849d4127'), 'white', {renderer: 'canvas'});  这是第二个图表的声明过程。注意代码的最前面多了一个<br/>，表示两个图表的连接。我们最后参考的就是这样的格式。  var option\_c0ec27a87bc34ac6a3af4169849d4127 = {  #这里省略具体数据和配置信息  };  chart\_c0ec27a87bc34ac6a3af4169849d4127.setOption(option\_c0ec27a87bc34ac6a3af4169849d4127);  </script>  <br/> <div id="2a31f641166844b6a9bf5fce6254c13f" class="chart-container" style="width:1500px; height:600px;"></div>  <script>  var chart\_2a31f641166844b6a9bf5fce6254c13f = echarts.init(  document.getElementById('2a31f641166844b6a9bf5fce6254c13f'), 'white', {renderer: 'canvas'});  这是第三个图表的声明过程。  在此之后，我们需要添加drill\_down.html里的类似的配置信息  var option\_2a31f641166844b6a9bf5fce6254c13f = {  #这里省略具体数据和配置信息  };  chart\_2a31f641166844b6a9bf5fce6254c13f.setOption(option\_2a31f641166844b6a9bf5fce6254c13f);  </script>  <br/> </div>  这里是定义图表拖拽以及缩放的代码。每两行对应一个图表。注意id的对应关系。  下一步要做的就是在此之后再添加两行，内容不变，只改变id就可以了。  <script>  $('#42c7719c0f6d40929f41a9c0f5dffd74').resizable().draggable().css('border-style', 'dashed').css('border-width', '1px');$("#42c7719c0f6d40929f41a9c0f5dffd74>div:nth-child(1)").width("100%").height("100%");  new ResizeSensor(jQuery('#42c7719c0f6d40929f41a9c0f5dffd74'), function() { chart\_42c7719c0f6d40929f41a9c0f5dffd74.resize()});  $('#c0ec27a87bc34ac6a3af4169849d4127').resizable().draggable().css('border-style', 'dashed').css('border-width', '1px');$("#c0ec27a87bc34ac6a3af4169849d4127>div:nth-child(1)").width("100%").height("100%");  new ResizeSensor(jQuery('#c0ec27a87bc34ac6a3af4169849d4127'), function() { chart\_c0ec27a87bc34ac6a3af4169849d4127.resize()});  $('#2a31f641166844b6a9bf5fce6254c13f').resizable().draggable().css('border-style', 'dashed').css('border-width', '1px');$("#2a31f641166844b6a9bf5fce6254c13f>div:nth-child(1)").width("100%").height("100%");  new ResizeSensor(jQuery('#2a31f641166844b6a9bf5fce6254c13f'), function() { chart\_2a31f641166844b6a9bf5fce6254c13f.resize()});  var charts\_id = ['42c7719c0f6d40929f41a9c0f5dffd74','c0ec27a87bc34ac6a3af4169849d4127','2a31f641166844b6a9bf5fce6254c13f'];  最后，需要把所有的图表id放进charts\_id中来。  function downloadCfg () {  const fileName = 'chart\_config.json'  let downLink = document.createElement('a')  downLink.download = fileName  let result = []  for(let i=0; i<charts\_id.length; i++) {  chart = $('#'+charts\_id[i])  最后这一部分保持原样即可。  result.push({  cid: charts\_id[i],  width: chart.css("width"),  height: chart.css("height"),  top: chart.offset().top + "px",  left: chart.offset().left + "px"  })  }  let blob = new Blob([JSON.stringify(result)])  downLink.href = URL.createObjectURL(blob)  document.body.appendChild(downLink)  downLink.click()  document.body.removeChild(downLink)  }  </script>  </body>  </html> |

分析以上的JS代码，我们发现，所要做的工作还是比较简单的，只需要**把drill\_down.html 的相关信息插入到第三个图表的声明过程之后，作为第四个图表显示**；然后在**定义图表拖拽以及缩放的代码中加入两行，其中所有的内容都保持不变，只需要把chart\_id修改成drill\_down.html画布对应的id即可**；最后再**往图表id里添加新的id**，就完成了所有的操作。同样，我们可以用基本的读写操作来完成这一项任务：

|  |
| --- |
| import re #用于正则表达式  with open( os.path.join(os.path.expanduser('~'),'Desktop',"page1.html"),'r',encoding="utf-8-sig") as html1:  with open( os.path.join(os.path.expanduser('~'),'Desktop',"drill\_down.html"),'r',encoding="utf-8-sig") as html2:  content1 = html1.readlines() #读取page1.html的信息  content2 = html2.readlines() #读取drill\_down.html的信息  #下面用了一个正则表达式，搜索drill\_down.html里对应的画布id信息：  drill\_down\_id = re.search('\_[\w\d]\* =',content2[11]).group().replace('\_','').replace(' =','')  #把画布id信息添加到chart\_id的最后一个位置：  content1[-27] = content1[-27][:-3]+",'"+drill\_down\_id+"'"+content1[-27][-3:]  #下面用了正则表达式，把定义图表拖拽以及缩放代码中有关图表id的部分替换成了drill\_down.html对应的画布id：  s1 = re.sub("#[\w\d]\*","#"+drill\_down\_id,content1[-29],count=2)  s2 = re.sub("#[\w\d]\*","#"+drill\_down\_id,content1[-28])  s2 = re.sub("\_[\w\d]\*","\_"+drill\_down\_id,s2)  #把新语句插入到前三条代码的后面：  content1\_new = content1[:-27]+[s1]+[s2]+content1[-27:]  #以下是drill\_down.html这张图表的声明过程，也就是我们需要插入作为第四张图表的声明信息  add\_info = content2[9:-2]  #别忘了在声明信息的最前端添加<br/>：  add\_info[0] = '<br/>'+add\_info[0]  #注意，由于我们之前只放了三张图表，所以第四张图表被插在倒数第37行的位置。如果图表数量变多，这里的37是需要按实际情况调整的。具体可以打开HTML源文件，看一下新的图片应该放在当前最后一个图表声明过程的后面，定义图表拖拽以及缩放的代码的前面。  content1\_new = content1\_new[:-37]+add\_info+content1\_new[-37:]    with open(os.path.join(os.path.expanduser('~'),'Desktop',"page\_new.html"),'w',encoding="utf-8-sig") as html\_new:  #写入文件即可  html\_new.writelines(content1\_new) |

执行完以上的代码，我们就完成了可视化大屏的基本绘制。在桌面打开page\_new.html，可以看到，我们的可视化大屏就如下图所示：

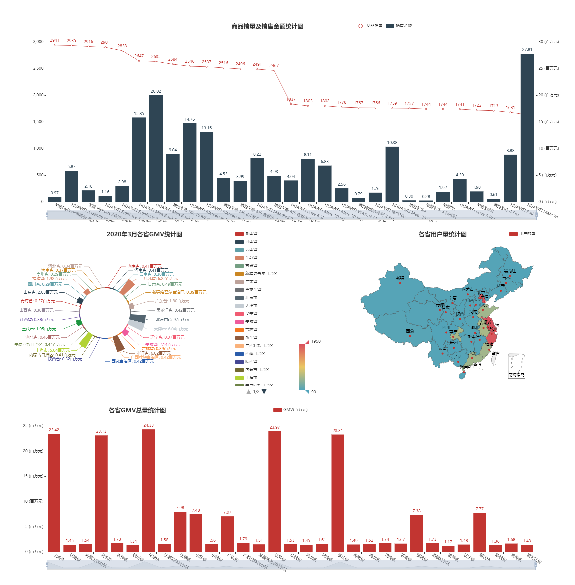


同样的，这时我们可以自由拖拽以及放大缩小，来完成我们的页面布局。

调整好之后，记得点击**Save Config按键**（这里需要注意，用IE打开HTML文件时可能会出现无法下载的问题，建议用谷歌浏览器完成），下载chart\_config.json 文件，然后把该文件复制粘贴到桌面，通过以下命令，生成新的my\_visualization.html文件：

|  |
| --- |
| Page.save\_resize\_html(os.path.join( os.path.expanduser('~'),'Desktop','page\_new.html'), cfg\_file= os.path.join( os.path.expanduser('~'),'Desktop','chart\_config.json'), dest= os.path.join( os.path.expanduser('~'),'Desktop','my\_visualization.html')) |

执行以上代码后，我们打开my\_visualization.html文件，看到的就是已经布置好的可视化大屏了：



当然，咱们每一名学员最后的HTML文件需要改个名，统一改成my\_visualization\_姓名.html，比如my\_visualization\_张三.html，方便老师为大家打分。

至此，我们所有的实验内容都已完结。大家上传作业的时候，把可视化部分的代码（最好能在自主创新的部分加上批注，批注里写上自己创新时的想法。在Python中，批注用#键表示）复制粘贴到作业模板中，再将最后生成的HTML文件作为文件对象插入到该word文档中，然后将word文档上传到指定位置即可。

## 创新指导

本手册只是简单介绍了如何利用Pyecharts搭建可视化大屏的基本操作，其中最有挑战性的部分就是下钻上卷的内容。但是各位同学在自主创新的时候，可以有非常非常多的突破点。比如页面的布局并不一定要和手册中所呈现的完全相同，而每位学员都可以多多加入一些定制化的内容。以下是几个显而易见的突破口：

1. 每张图表不一定非的是白底的，可以在一开始初始化每张图表的时候就修改对应的主题，比如指定bar = Bar( init\_opts = opts.InitOpts(theme='dark') )，就可以把背景修改成黑色了。

2. 两张柱状图的颜色都比较单一，是否可以尝试修改一下柱子的颜色？

3. 最后的下钻上卷过程中，省份排列是乱序的，大家可以尝试一下，能否按照一定的顺序将省份进行排列呢？

3. 我们只呈现了四种图表类型，但是Pyecharts里有非常多不同种类的图表，比如BI里非常常用的漏斗图，大家都可以加入自己的定制化大屏搭建之中。

4. 我们进行的数据分析其实比较简单，有兴趣的同学可以深入挖掘一下其他相关的经营数据，方便指导后续的经营决策，这才是BI的作用所在。

最后，本小节以及前面几个小节内所有的“创新指导小Tips”都只是抛砖引玉。期待能在最后看见同学们提出更多更有意思的创新点！

## 本章小结

本实验在前几个关卡的基础上，利用开源的Pyecharts工具包，实现了从数据分析到数据可视化的全部过程。其中最富有挑战性的内容在于如何实现图表的下钻上卷。但这一部分内容对于有JS编程基础的学员而言也不算太难，因此推荐学有余力的同学进一步深入探索，毕竟对于一名数据分析师而言，对数据可视化的钻研永远没有尽头。