

Pyecharts可视化分析

鲲鹏创新实践课：鲲鹏应用数据分析与管理实战



前言

- 本课程主要介绍数据可视化的相关概念、基本思想、以及Pyecharts的相关知识点，帮助学员更好的完成BI数据可视化。

浙大-华为鲲鹏创新实践课



目标

- 学完本课程后，您将能够：
 - 描述什么是可视化，可视化技术有哪些；
 - 掌握数据可视化概念及基本流程；
 - 能够基于Pyecharts实现交互式数据可视化。

浙大-华为鲲鹏创新实践课



目录

1. Python基础知识
2. 可视化概述
3. 数据可视化综述
4. 基于Pyecharts搭建可视化大屏流程
5. BI可视化实验演示

浙大-华为鲲鹏创新实践课



Python语言的诞生和发展历史

- Python语言是一种解释型、面向对象、动态数据类型的高级程序设计语言。
- Python语言能用于多种领域的程序开发：



与数据库交互

编写系统工具

写基于网络的软件

开发图形界面的应用

科学计算、游戏开发、数据分析



变量

- Python 中的变量不需要声明；变量的赋值操作就完成了声明和定义的过程，在其他语言中需要制定类型；每个变量在使用前都必须赋值，变量赋值以后该变量才会被创建。
- 在 Python 中，变量就是变量，它没有类型，我们所说的数据“类型”是变量所指的内存中对象的类型。
- 等号（=）用来给变量赋值：
 - 等号运算符左边是一个变量名，右边是存储在变量中的值。
 - 例如：x=3，x是变量名，3是存储在变量中的值。



变量命名规则

- 变量名又称标识符。
- 变量名必须是大小写英文字母、数字或下划线 “_” 的组合，不能用数字开头，并且对大小写敏感。
- 关键字不能用于命名变量（31个），如and、as、assert、break、class、continue、def、del等：

```
import keyword  
print(keyword.kwlist)
```

```
['False', 'None', 'True', 'and', 'as', 'assert', 'async', 'await',  
'break', 'class', 'continue', 'def', 'del', 'elif', 'else', 'except',  
'finally', 'for', 'from', 'global', 'if', 'import', 'in', 'is',  
'lambda', 'nonlocal', 'not', 'or', 'pass', 'raise', 'return', 'try',  
'while', 'with', 'yield']
```



列表 (list)

- 一个列表就是一列变量：
 - 列表是一个**有序的**序列结构，序列中的元素可以是不同的数据类型。
 - 列表可以进行一系列序列操作，如索引、切片、加、乘和检查成员等。

```
l = [1, -3.4, 'python', False]  
print(type(l))
```

```
<class 'list'>
```




列表推导式 (1)

- list是使用python过程中是一个非常常用的数据结构，无论是作为最终数据的保存结果，还是中间数据结果的临时存储，都能提供很方便的功能。使用列表推导式可以让循环在列表内完成。

```
list = [i**2 for i in range(5)]  
print (list)
```

```
[0, 1, 4, 9, 16]
```



列表推导式 (2)

- 以下为例，对列表中每个数值逐个减去均值。

```
total = [1,2,3,4,5,6,7,8,9,100]
mean_total = sum(total) / len(total)
delta_total = [i-mean_total for i in total]
print (delta_total)
```

```
[-13.5, -12.5, -11.5, -10.5, -9.5, -8.5, -7.5, -6.5, -5.5, 85.5]
```



索引

- Python语言中所有的索引都是从 0 开始计数的，如果列表中有 n 个元素，那么最后一个元素的索引是 n-1 。注意Python里的索引要用中括号。

```
l = [1, -3.4, 'python', False]
```

表1 索引位置及相应的值				
index (索引位置)	0	1	2	3
value (值)	1	-3.4	"python"	False

- 如果我们想要获取 l 中第 3 个元素及倒数第二个元素，索引分别为2和-2：

```
l = [1, -3.4, 'python', False]
print(l[2], l[-2])
```

```
python python
```



切片 (1)

- 切片操作需要提供起始索引位置和最后索引位置，然后用冒号：将两者分开。切片同样使用中括号。

列表名称[起始索引位置：最后索引位置：步长]

- 如果未输入步长，则默认步长为 1。
- 切片操作返回一系列从起始索引位置开始到最后索引位置结束的数据元素。
 - 需要注意的是，起始索引位置的值包含在返回结果中，而最后索引位置的值不包含在返回结果中。换句话说，从集合的角度出发，这是个“前闭后开”的区间。

```
list_2 = [1, -3.4, 'python', False, 1, 'x', 'y', 'z']  
print(list_2[1:4])
```

```
[-3.4, 'python', False]
```



切片 (2)

- 一般来说，元素位置是从左往右数的，当然也可以从右往左数，这时就可以使用逆向索引。

```
list_2[-7:-4]
```

```
[-3.4, 'python', False]
```

- 逆向切片只需将索引值前加上负号即可，-1表示最后（最右边）的元素。由于步长还是1，所以依然是从左往右切片。
- 我们可以省略起始索引位置，表示从最开始进行切片，当我们将两个索引都省略之后，我们将按原样复制一个列表，如果想要将列表的顺序颠倒，则可以使用[::-1]。

```
print(list_2[:-2])  
print(list_2[:])  
print(list_2[::-1])
```

```
[1, -3.4, 'python', False, 1, 'x']  
[1, -3.4, 'python', False, 1, 'x', 'y', 'z']  
['z', 'y', 'x', 1, False, 'python', -3.4, 1]
```



元组 (tuple)

```
tuple_1 = (1, -3.4, 'python', False, 1, 'x', 'y', 'z')
```

- 元组 (tuple) 数据结构与列表类似，其中元素可以有不同的类型。
- 但是元组中的元素是不可变的，即一旦初始化之后，就不能够再做修改（报错：元组对象不支持赋值）。

```
list_1 = [1, -3.4, 'python', False, 1, 'x', 'y', 'z']  
list_1[0] = 3  
print(list_1)
```

```
[3, -3.4, 'python', False, 1, 'x', 'y', 'z']
```

```
tuple_1 = (1, -3.4, 'python', False, 1, 'x', 'y', 'z')  
tuple_1[0] = 3  
print(tuple_1)
```

TypeError

Traceback (most recent call 1

ast)

<ipython-input-84-bfc150b53acc> in <module>

```
1 tuple_1 = (1, -3.4, 'python', False, 1, 'x', 'y', 'z')  
——> 2 tuple_1[0] = 3  
3 print(tuple_1)
```

TypeError: 'tuple' object does not support item assignment



字典 (dict)

- 字典 (dict) 在其他语言中被称作哈希映射 (hash map) 或者相关数组 (associative arrays)
- 字典是一种大小可变的键值对集，其中的键 (key) 和值 (value) 都是Python对象。
- 字典的创建使用大括号 {} 包含键值对，并用冒号 : 分隔键和值，形成 “键 : 值” 对。
- 不同键所对应的值可以相同，但是键必须是唯一的。

```
dict_language = {1:'python',
                  4:'java',
                  3:'c++',
                  2:'c#',
                  5:'Ruby',
                  6:'python',
                  7:'java'}
print(dict_language)
```

```
{1: 'python', 4: 'java', 3: 'c++', 2: 'c#', 5: 'Ruby', 6: 'python', 7:
'java'}
```



字典索引

- 字典也可以进行索引，且索引的语法相同；不同的是，列表和元组的索引号是按照顺序自动生成，而字典的索引号只能是键。

```
dict_language['f']  
  
'python'
```

- 字典中某值的索引还可以通过 get 方法来完成，如果字典不包含某个键，可以返回 None，或者自己指定的值。

```
print(dict_language.get('a'))  
print(dict_language.get('x'))  
print(dict_language.get('x', -1000))  
  
python  
None  
-1000
```

- 如果不太确定字典中有哪些键或者值，我们可以使用 keys() 方法或者 values() 方法。

```
print('keys:', dict_language.keys())  
print('values:', dict_language.values())  
  
keys: dict_keys(['a', 'b', 'c', 'd', 'e', 'f', 'g'])  
values: dict_values(['python', 'java', 'c++', 'c#', 'Ruby', 'python', 'java'])
```




集合 (set)

```
{ 'python', 'java', 'c++', 'c#', 'Ruby', 'python', 'java' }
```

- 集合 (set) 也是一种无序集，它和数学上的离散集合定义非常类似，就是指放在一起的一些对象。
- 在集合中，重复的对象是不被允许的。集合可以用于去除重复值。
- 集合也可以进行数学集合运算，如并、交、差以及对称差等。
- 应用：
 - 去重。把一个列表变成集合，就自动去重了。
 - 关系测试。测试两组数据之前的交集、差集、并集等关系。



自定义函数的语法规则

关键字 函数名 参数

函数主体

```
def take_sum(list1):  
    s = 0  
    for ele in list1:  
        s += ele  
    return s
```

返回变量

```
list1 = [9, 11, 17, 10, 16, 4, 3,  
         19, 5, 3, 10, 10, 13, 3]  
#调用函数, 保存至s  
s = take_sum(list1)  
#输出s  
print(s)
```

133



数据的读取 (1)

- 对文件操作之前需要用 `open()` 函数打开文件。

```
f = open('helloworld.txt', mode='r')
```

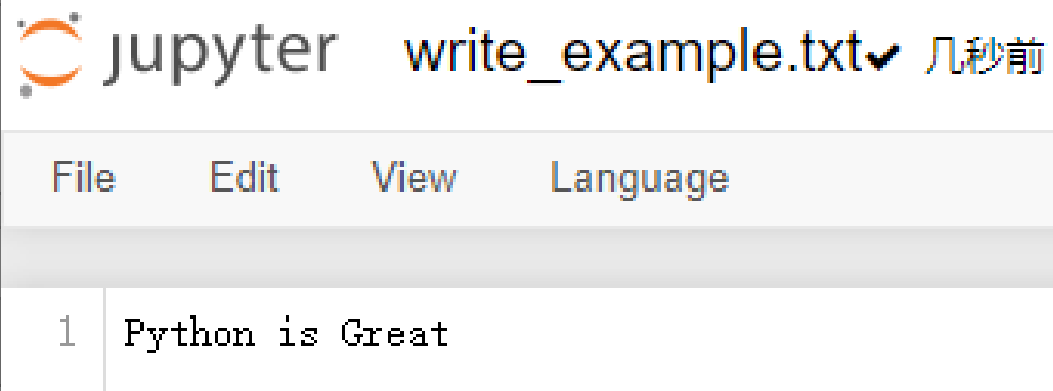
- `mode` 参数中的 'r' 指读出（只读），'w' 指写入。
- 打开之后将返回一个文件对象（file object），换句话说f就是一个文件对象；后续对文件内数据的操作都是基于这个文件对象的方法（method）来实现的。



数据的写入

- 写入的操作和读取是类似的，不过用的是 `write()` 函数，同时需要将打开文件的 `mode` 参数设置为 `w` 。

```
f = open('write_example.txt', 'w')  
f.write('Python is Great')  
f.close()
```





多行文本文件的读取 (1)

- 通过 `open()` 函数打开文件，返回文件对象。
- 对文件对象进行读取操作，除了前面介绍的 `read()` 之外还有两种读取数据的方法：
`readline()` 是每次读入一条数据的方式， `readlines()` 是一次性读入文件所有数据。

```
path = 'economic_freedom_index2019_data.txt'
f = open(path, 'r') #打开文件并存入f对象
content = f.readlines() #读取所有数据
f.close()
print(content[1:3])
```

```
['1\tAfghanistan\tAfghanistan\tAsia-Pacific\t152\t39\t51.5\t19.6\t29.6\t25.2\t91.7\t80.3\t99.3\t49.2\t60.4\t76.7\t66\t10\t10\t7\t20\t20\t5\t25.6\tAfghanistan\t35.5\t$69.60\t2.5\t2.9\t$1,958\t8.8\t5\t53.9\t7.3\n',
'2\tAlbania\tAlbania\tEurope\t52\t27\t66.5\t54.8\t30.6\t40.4\t86.3\t73.9\t80.6\t69.3\t52.7\t81.5\t87.8\t70\t70\t1.1\t23\t15\t24.9\t29.5\tAlbania\t2.9\t$36.00\t3.9\t2.5\t$12,507\t13.9\t2\t1,119.10\t71.2\n']
```



多行文本文件的读取 (2)

- `readlines()` 读取后得到的是每行数据组成的列表。但是一行样本数据全部存储为一个字符串，并且数据读入后并没有将换行符和制表符去掉（制表符号是 `\t`，换行符号是 `\n`）。
- 如果需要去掉换行符和制表符，就需要一些字符串处理的函数，比如 `split()` 函数和 `strip()` 函数。

```
sub1 = content[1]
tmp = sub1.split('\t') #split()方法里的参数为分隔的依据, 比如这里指的是按'\t'进行分割
print('sub1 =', sub1)
print('tmp =', tmp) #用split()分割开后的结果也是一个列表
```

```
sub1 = 1      Afghanistan      Afghanistan      Asia-Pacific      152      39      51.5      19.6      29.6      25.2      91.7      80.3      99.3      49.2
60.4      76.7      66      10      10      7      20      20      5      25.6      Afghanistan      35.5      $69.60      2.5      2.9      "$1,958 "
8.8      5      53.9      7.3
```

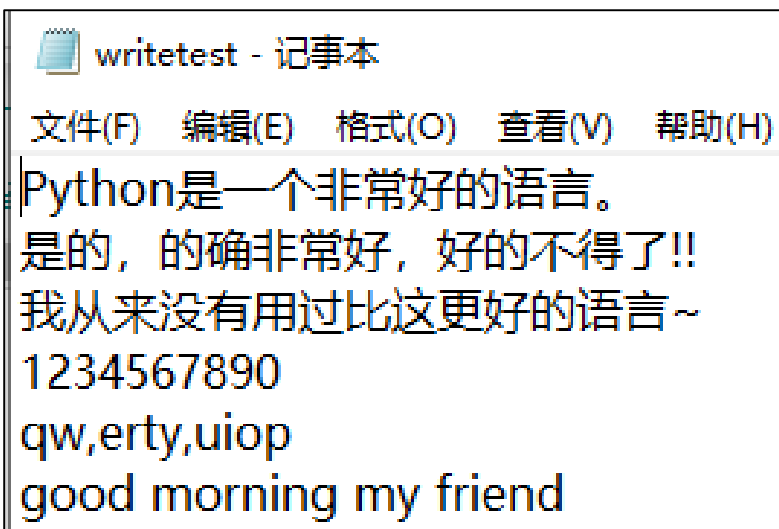
```
tmp = ['1', 'Afghanistan', 'Afghanistan', 'Asia-Pacific', '152', '39', '51.5', '19.6', '29.6', '25.2', '91.7', '80.3', '99.3', '49.2', '60.4', '76.7', '66', '10', '10', '7', '20', '20', '5', '25.6', 'Afghanistan', '35.5', '$69.60 ', '2.5', '2.9', '"$1,958 "', '8.8', '5', '53.9', '7.3\n']
```



多行文本文件的写入

- write() 、 writelines() 是两个对文件对象的写入数据的方法。write() 是逐次写入，writelines() 可对一个列表里的所有数据一次性写入文件中。
- 设置参数 为"w" 。

```
content = ["Python是一个非常好的语言。\\n",  
           "是的，的确非常好，好的不得了!!\\n",  
           "我从来没有用过比这更好的语言~\\n",  
           "1234567890\\n",  
           "qw,erty,uiop\\n",  
           "good morning my friend"]  
f = open("writetest.txt", "w")  
f.writelines(content)  
f.close()
```



writetest - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

Python是一个非常好的语言。
是的，的确非常好，好的不得了!!
我从来没有用过比这更好的语言~
1234567890
qw,erty,uiop
good morning my friend



NumPy (1)

- NumPy几乎是一个无法回避的**科学计算**工具包，最常用的也许是它的n维数组对象，其他还包括一些成熟的函数库，用于整合C/C++和Fortran代码的工具包，线性代数、傅里叶变换和随机数生成函数等。NumPy提供了两种基本的对象：ndarray和ufunc。ndarray是存储单一数据类型的高维数组，而ufunc则是能够对数组进行处理的函数。
- 特点：
 - 高效矩阵与数组处理。
 - 丰富的数值计算函数。
 - 实用的线性代数、傅里叶变换和随机数生成函数。





NumPy (2)

- `np.zeros(shape, dtype=float, order='C')`: 创建一个形状为`shape`的全零数组。`dtype`为数据类型。`order=C`代表与c语言类似，行优先;`order=F`代表列优先。
- `numpy.full(shape, fill_value, dtype=None, order='C')`: 生成一个`fill_value`填充的数组。
- `numpy.linspace(start,stop,num=50,endpoint=True,retstep=False, dtype=None, axis=0)`: 生成一个等间隔的数组，`start`起始值，`stop`终止值，`num`数量，`endpoint=True`表示`stop`为最后一个值。
- `numpy.array(object, dtype=None, copy=True, order='K', subok=False, ndmin=0)`: 生成一个数组。



Pandas (1)

- Pandas全名Python Data Analysis Library，是基于NumPy的一种工具，还可以结合matplotlib来使用。该工具是为了解决**数据分析**任务而创建的。
 - Pandas 纳入了大量库和一些标准的数据模型，提供了高效地操作大型数据集所需的工具。
 - Pandas提供了大量能使我们快速便捷地处理数据的函数和方法，比如数据读取，索引切分，分组，时间序列，重塑，合并。
 - 对各类数据读取，包括csv, json, excel, txt, api, html, 数据库等等；可以把数据高效简单存储成多种格式，包括hdf5, csv, excel等。



Pandas (2)

- `pandas.Series(data=None, index=None, dtype=None, name=None, copy=False, fastpath=False)`: 生成一个Series数据。data为数据可以是数组和字典等；index为索引值，要求与数据长度相同，dtype为数据类型。
- `pandas.DataFrame(data=None, index=None, columns=None, dtype=None, copy=False)`: 生成一个DataFrame数据。data是数据，index是索引，columns是列名。
- `DataFrame.head(n=5)`: 显示前n条数据，n表示显示的数据量。
- `DataFrame.tail(n=5)`: 显示底部数据，n表示显示的数据量。
- `DataFrame.loc`: 按标签或布尔数组访问一组行和列。
- `DataFrame.iloc`: 按索引或者切片访问一组行和列。



目录

1. Python基础知识
- 2. 可视化概述**
3. 数据可视化综述
4. 基于Pyecharts搭建可视化大屏流程
5. BI可视化实验演示

浙大-华为鲲鹏创新实践课



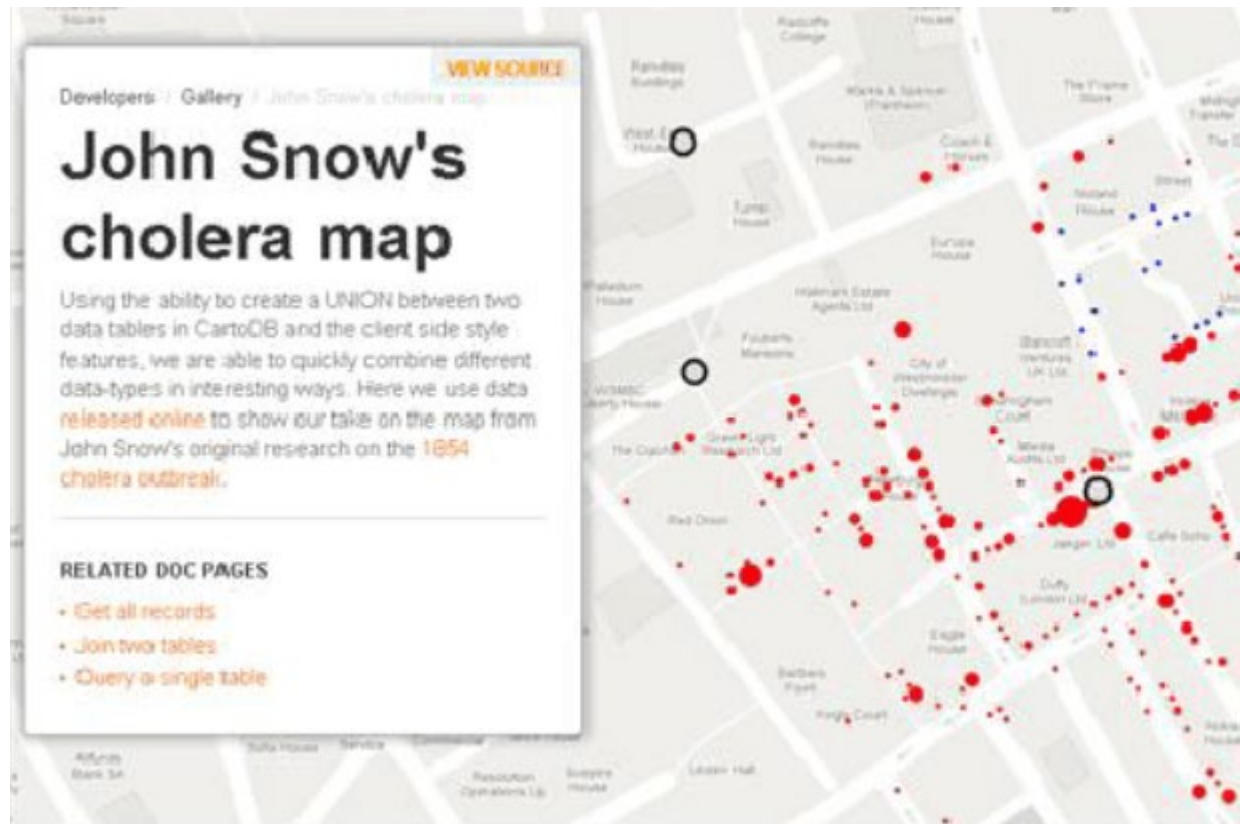
数据可视化解决霍乱疾病

- 1854年英国伦敦爆发严重霍乱。





现代数据可视化技术解决霍乱传播





什么是可视化？

- 可视化又称视觉化，它的基本含义是将生活生产中产生的大量非直观的、抽象的或者不可见的数字借助于计算机图形学和图像处理等技术，用几何图形和色彩、纹理、透明度、对比度及动画技术等手段，以图形图像信息的形式，直观、形象地表达出来，并进行交互处理。





数据可视化的基本特性

数据可视化可以根据数据纬度进行综合分析。



数据可视化是不断优化的过程，需要不断引入新数据并创建新视图。

数据以线条、图表、图像等进行可视化分析。



为什么需要数据可视化?

直观

将抽象的数据转化为直观的图表。



降低学习理解数据的门槛。

易懂

高效

方便管理者能够更加快速、便捷、高效的理解数据。



对数据有效的展示能够极大提高我们的洞察力。

挖掘





目录

1. Python基础知识
2. 可视化概述
- 3. 数据可视化综述**
4. 基于Pyecharts搭建可视化大屏流程
5. BI可视化实验演示

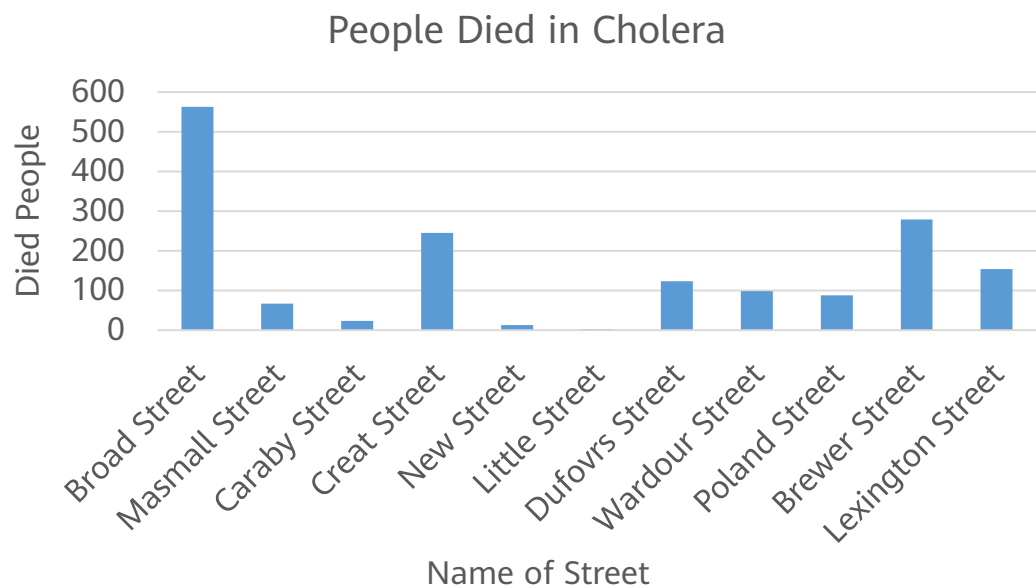
浙大-华为鲲鹏创新实践课



关于John Snow打败霍乱传播的思考

- 运用数据可视化可以解决当时医疗水平无法解决的疾病聚集与传播途径的问题；
- 选用恰当的可视化展现方式，使用地图而非其他图表。

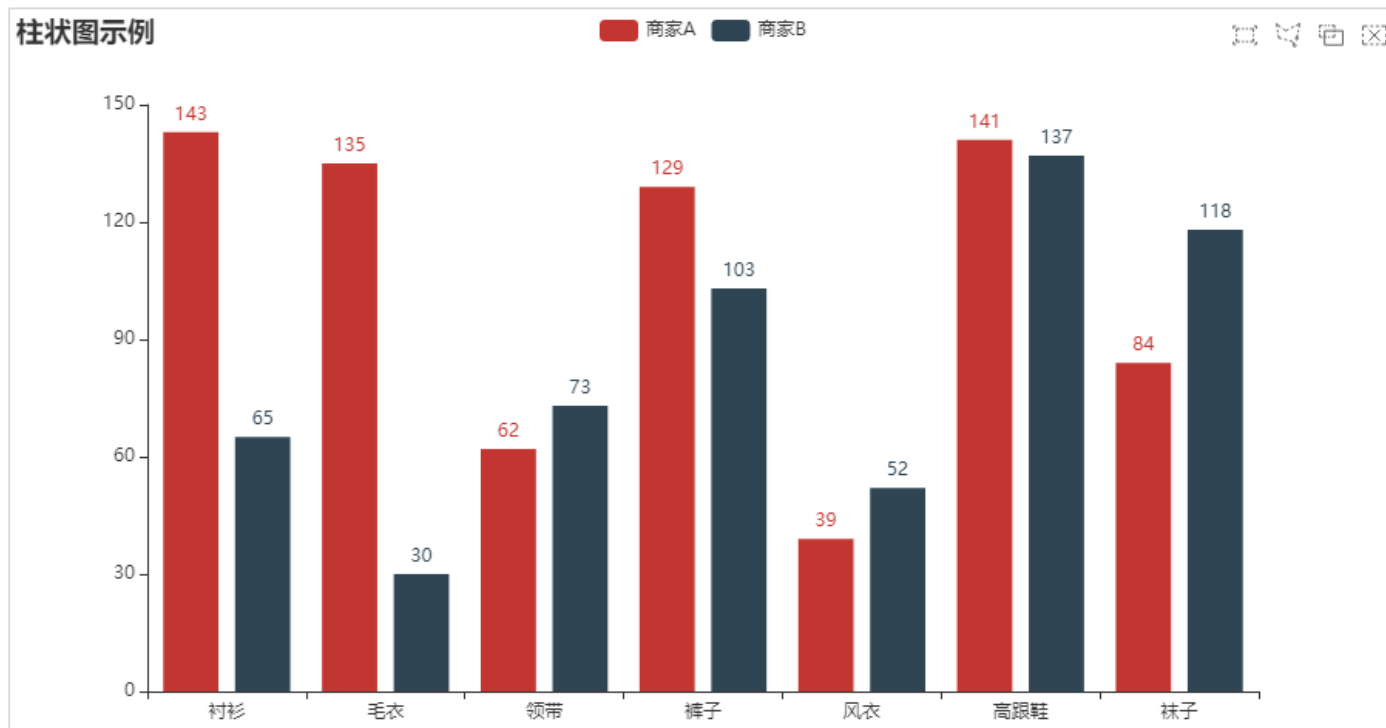
num	street	died_people
1	Broad Street	563
2	Masmall Street	67
3	Caraby Street	23
4	Creat Street	245
5	New Street	13
6	Little Street	2
7	Dufovrs Street	123
8	Wardour Street	98





常见的可视化图表 - 柱状图

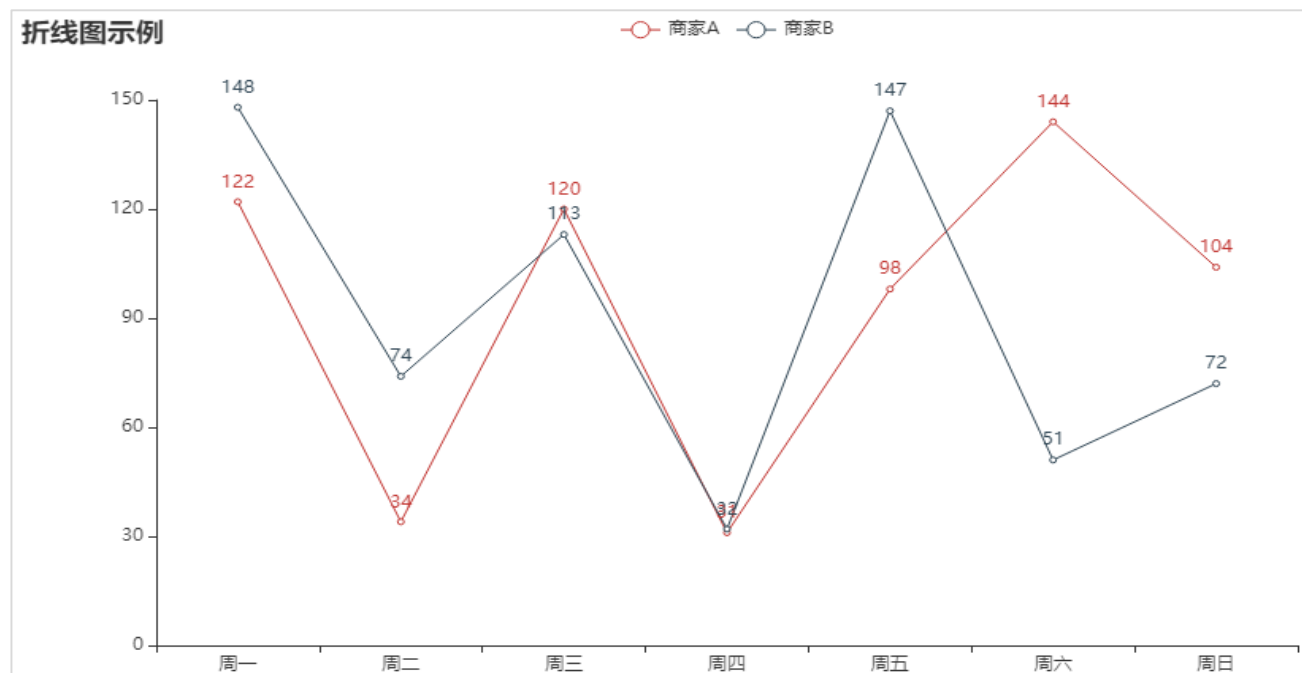
- 柱状图：展示多个分类的数据变化和同类别各变量之间的比较情况
- 适用：对比分类数据
- 局限：分类过多则无法展示数据特点





常见的可视化图表 - 折线图

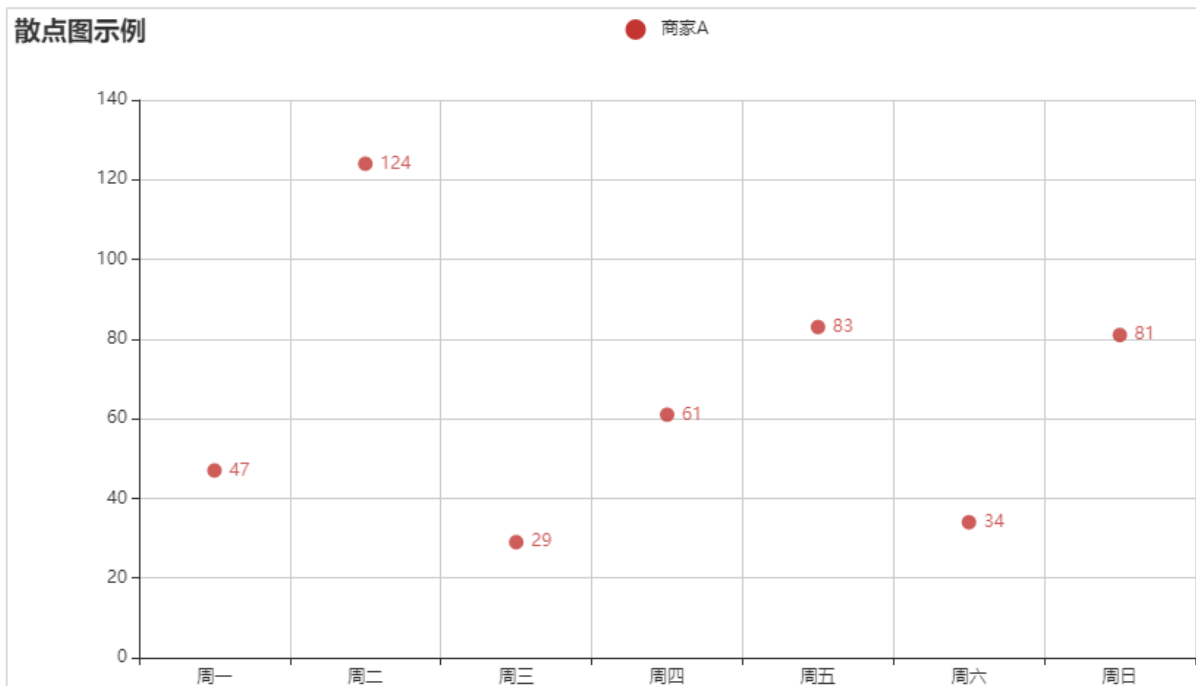
- 折线图：展示数据随时间或有序类别的波动情况的趋势变化
- 适用：有序类别，比如时间
- 局限：无序的类别无法展示数据特点





常见的可视化图表 - 散点图

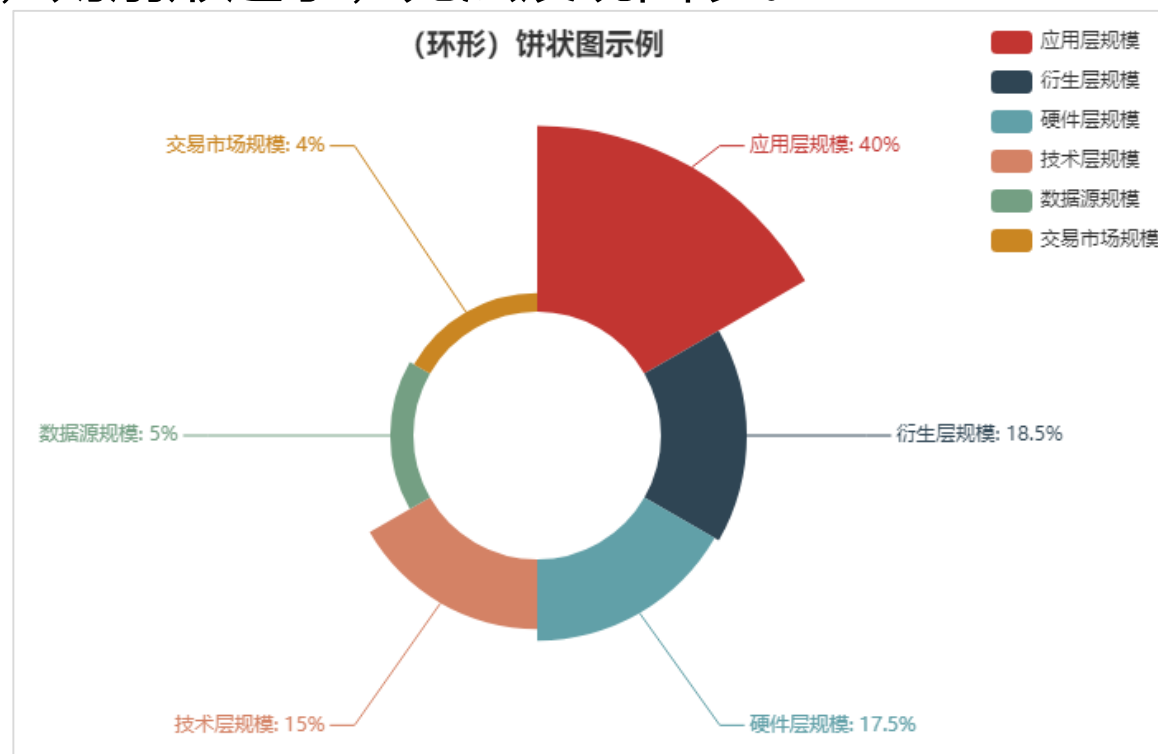
- 散点图：用于发现各变量之间的关系
- 适用：存在大量数据点，结果更精准，比如回归分析
- 局限：数据量小的时候会比较混乱





常见的可视化图表 - 饼图

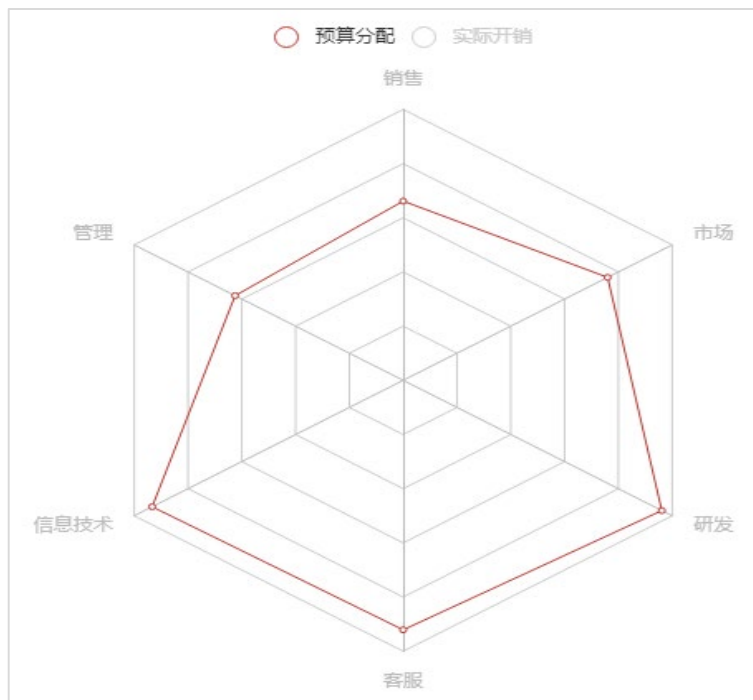
- 饼图：用来展示各类别占比，比如各类别所占比例。
- 适用：了解数据的分布情况。
- 缺陷：分类过多，则扇形越小，无法展现图表。





常见的可视化图表 - 雷达图

- 雷达图：用来展示同一管理下不同类别的量纲分布
- 适合：展示不同类别的相互差别
- 局限：类别不宜过多，指标值展示不清晰





常见的可视化图表 - 其他



南丁格尔玫瑰图



蜡烛图



多组条形图



不等宽柱状图



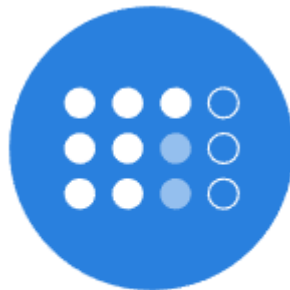
气泡地图



旭日图



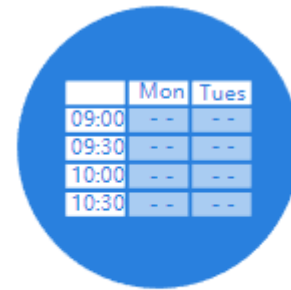
堆叠式面积图



点阵图表



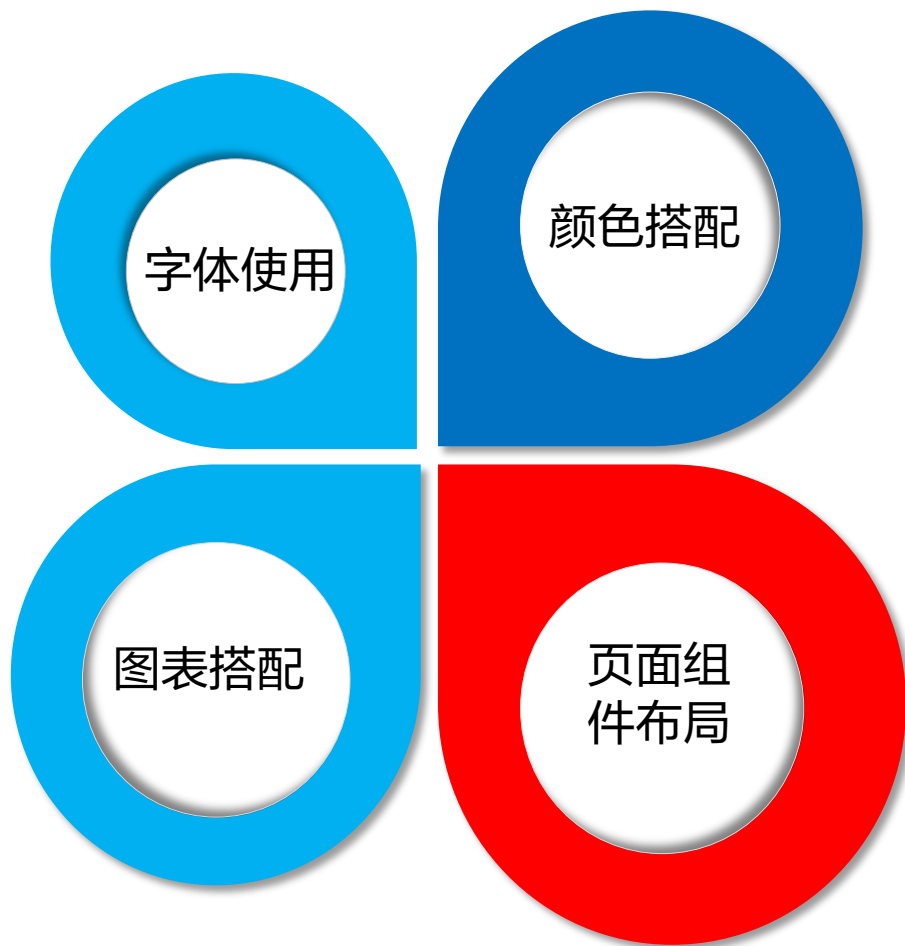
人口金字塔



时间表



可视化大屏设计原则



字体使用

选择合适的字体，字体类型忌多



颜色搭配

配色需要合理化布局，同一图表颜色一般不超过四种



图表搭配

图表选择需要根据数据特点，搭配统一美观



页面组件布局

页面组件布局尺寸、颜色搭配合理、美观



常用的可视化工具

- 专业的可视化工具有很多，大致可分为三类：企业级专业可视化工具、轻量级可视化工具、编程式图表工具。





目录

1. Python基础知识
2. 可视化概述
3. 数据可视化综述
- 4. 基于Pyecharts搭建可视化大屏流程**
5. BI可视化实验演示



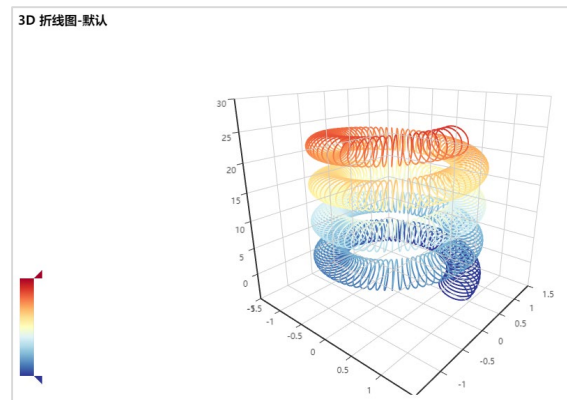
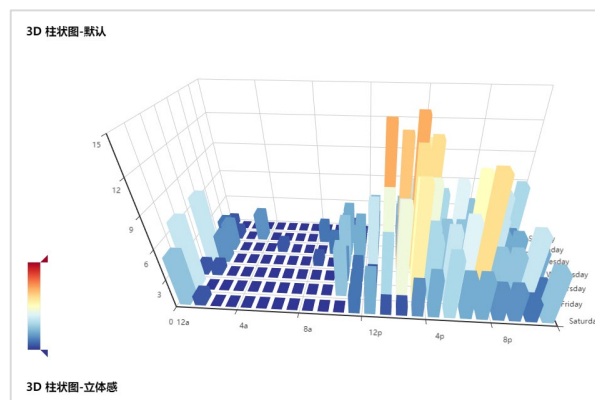
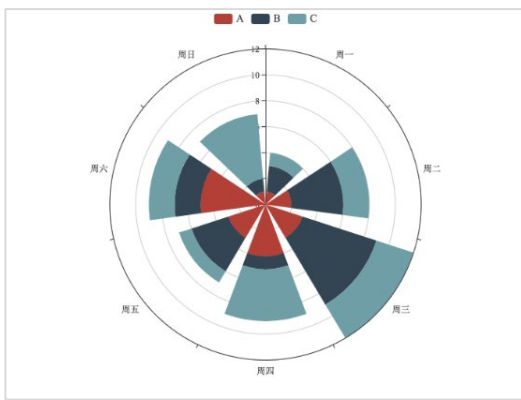
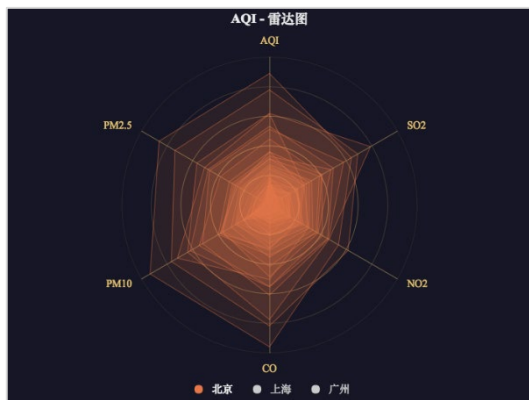
Echarts简介

- ECharts，一个使用 JavaScript 实现的开源可视化库，提供了常规的折线图、柱状图、散点图、饼图、K线图，用于统计的盒形图，用于地理数据可视化的地图、热力图、线图，用于关系数据可视化的关系图、treemap、旭日图，多维数据可视化的平行坐标，还有用于 BI 的漏斗图，仪表盘，并且支持图与图之间的混搭。
- ECharts 能够展现千万级的数据量，并且在这个数据量级依然能够进行流畅的缩放平移等交互。
- ECharts 针对移动端交互做了细致的优化，例如移动端小屏上适于用手指在坐标系中进行缩放、平移。



Pyecharts简介

- Pyecharts 是一个用于生成 Echarts 图表的类库。
- Pyecharts是为了Echarts与 Python 进行对接，方便在 Python 中直接使用数据生成图。
- Pyecharts可以制作直角坐标系图、3D图、地图等可视化图表。





Pyecharts可视化

- Pyecharts可视化的过程可以分为以下几步：
 - 数据准备
 - 实例化图形对象
 - 填充数据
 - 设置图形参数
 - 保存和展示



Pyecharts版本

- Pyecharts可以分为0.5x和1.x两个大的版本，这两个版本是互不兼容的，并且在方法调用上也存在不同。
 - 0.5x支持Python2.7和Python3.4/3.5；1.x的版本只支持Python3.6+。
 - 0.5x的版本支持插件（如地图、主题等）；1.x版本弃用了插件，采用了本地静态文件和官方在线的文件（local和online两种模式）。
 - 1.x的版本减少了对于其他库的依赖，相较于0.5x更加轻量级。
 - API和方法调用不同。
 - 1.x开始支持jupyterlab。



Pyecharts v0.5x和v1.x对比

0.5x

```
from pyecharts import Bar

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("柱状图数据堆叠示例")
bar.add("商家A", attr, v1, is_stack=True)
bar.add("商家B", attr, v2, is_stack=True)
bar.render()
```

1.x

```
import pyecharts.options as opts
from pyecharts.charts import Bar
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")
    .set_series_opts(label_opts=opts.LabelOpts(is_show=False))
    .set_global_opts(title_opts=opts.TitleOpts(title="柱状图数据堆叠示例"))
)
bar.render()
```



Pyecharts v1.x可视化实现

```
import pyecharts.options as opts #导入选项模块
from pyecharts.charts import Bar #导入工具库

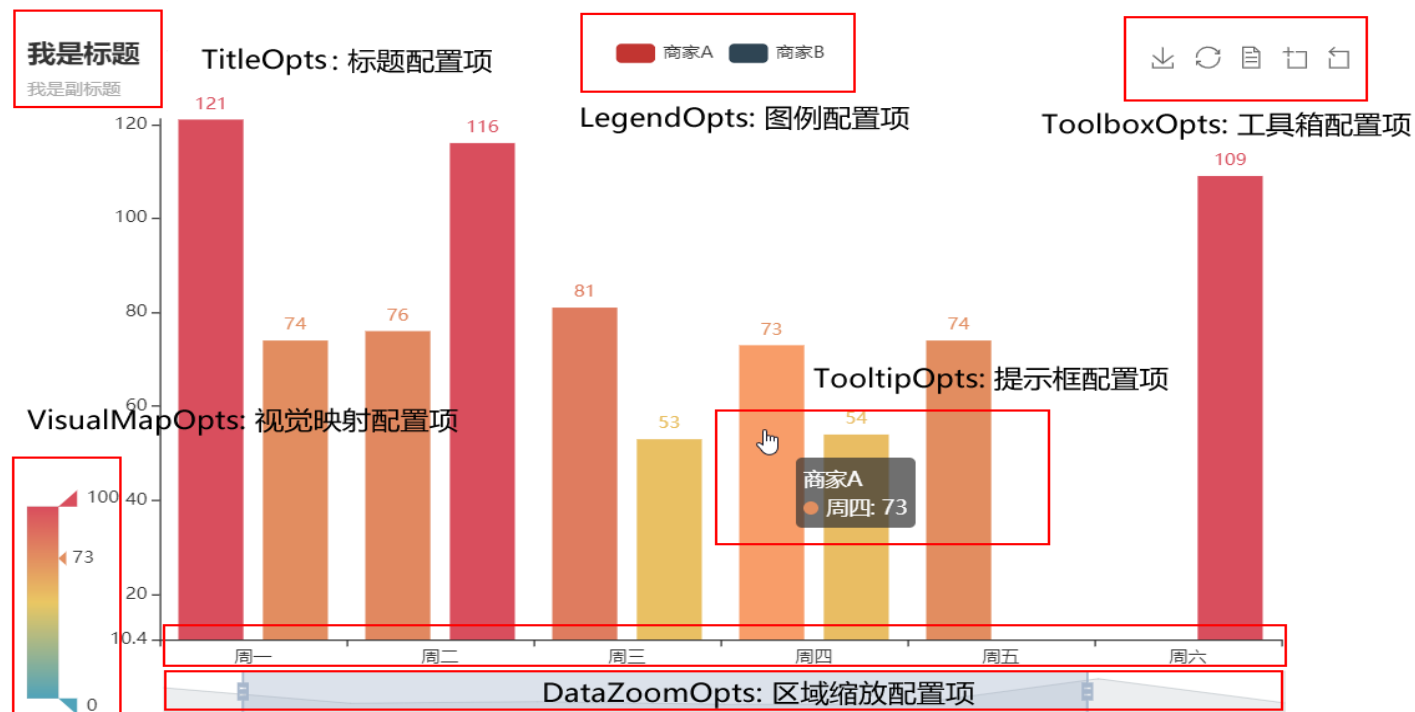
attr = ["衬衫", "羊毛衫", "雪纺衫", "裤子", "高跟鞋", "袜子"] #数据准备: 横坐标数据
v1 = [5, 20, 36, 10, 75, 90] #数据准备: 第一组数据
v2 = [10, 25, 8, 60, 20, 80] #数据准备: 第二组数据

bar = (
    Bar() #实例化图形对象
    .add_xaxis(attr) #添加横坐标
    .add_yaxis("商家A", v1, stack="stack1") #添加第一组数据
    .add_yaxis("商家B", v2, stack="stack1") #添加第二组数据
    .set_series_opts(label_opts=opts.LabelOpts(is_show=False)) #系列化参数设置
    .set_global_opts(title_opts=opts.TitleOpts(title="柱状图数据堆叠示例")) #全局参数设置
    bar.render("可视化示例.html") #编译成html文件
```



全局配置项综述

- 全局配置项：用于修改全局参数。
- 常见的全局配置项包括动画配置项、图形初始化、标题配置项、图例配置项、工具箱配置项、视觉映射配置项、提示框配置项以及区域缩放配置项等。





全局配置项 - 图形初始化

- 初始化配置项 (InitOpts): 主要用于初始化 Echarts 图表的画布 (canvas) 信息。
 - 使用规则:

```
import pyecharts.options as opts #导入选项模块
from pyecharts.charts import Bar #导入工具库
bar = Bar( init_opts = opts.InitOpts(width='900px',height='500px',theme='dark') )
```

- 常见的可选参数包括:

```
class InitOpts(
    width: str = "900px", # 图表画布宽度, css 长度单位。
    height: str = "500px", # 图表画布高度, css 长度单位。
    chart_id: Optional[str] = None, # 图表 ID, 图表唯一标识, 用于在多图表时区分。
    renderer: str = RenderType.CANVAS, # 渲染风格, 可选 "canvas", "svg"
    page_title: str = "Awesome-pyecharts", # 网页标题, 注意和图表标题不同
    theme: str = "white", # 图表主题
    bg_color: Optional[str] = None, # 图表背景颜色
)
```



全局配置项 - 标题配置项

- 标题配置项 (TitleOpts): 主要用于配置所绘制图表的主标题和副标题的相关参数。

▫ 使用规则:

```
import pyecharts.options as opts #导入选项模块
from pyecharts.charts import Bar #导入工具库
bar = Bar().set_global_opts( title_opts = opts.TitleOpts( title="标题示例", pos_left='20%' ) )
```

▫ 常见的可选参数包括:

```
class TitleOpts(
    title: Optional[str] = None, # 主标题文本, 支持使用 \n 换行。
    subtitle: Optional[str] = None, # 副标题文本, 支持使用 \n 换行。
    pos_left: Optional[str] = None, # title 组件离容器左侧的距离。
    pos_right: Optional[str] = None, # title 组件离容器右侧的距离。
    pos_top: Optional[str] = None, # title 组件离容器上侧的距离。
    pos_bottom: Optional[str] = None, # title 组件离容器下侧的距离。
    item_gap: Numeric = 10, # 主副标题之间的间距。
)
```



全局配置项 - 图例配置项

- 图例配置项 (LegendOpts): 主要用于配置所绘制图表中图例的相关参数。

```
import pycharts.options as opts #导入选项模块
from pycharts.charts import Bar #导入工具库
bar = Bar().set_global_opts( legend_opts = opts.LegendOpts( is_show = True, pos_left='20%' ) )
```

- 常见的可选参数包括:

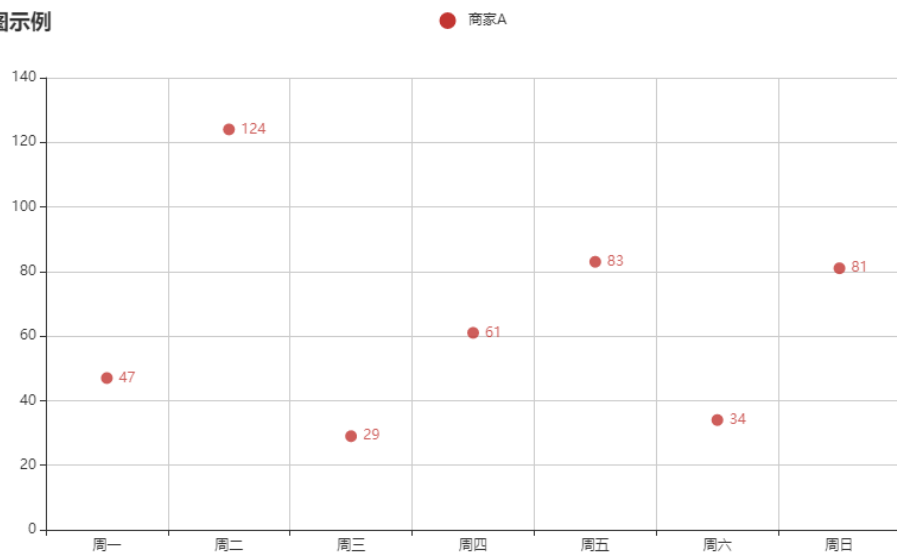
```
class LegendOpts(
    type_: Optional[str] = None, # 图例的类型。可选值: plain': 普通图例; 'scroll': 可滚动翻页的图例。
    is_show: bool = True, # 是否显示图例组件
    pos_left: Union[str, Numeric, None] = None, # 图例组件离容器左侧的距离。
    pos_right: Union[str, Numeric, None] = None, # 图例组件离容器右侧的距离。
    pos_top: Union[str, Numeric, None] = None, # 图例组件离容器上侧的距离。
    pos_bottom: Union[str, Numeric, None] = None, # 图例组件离容器下侧的距离。
    orient: Optional[str] = None, # 图例列表的布局朝向。可选: 'horizontal', 'vertical'
    item_gap: int = 10, # 图例每项之间的间隔。横向布局时为水平间隔, 纵向布局时为纵向间隔。
    item_width: int = 25, # 图例标记的图形宽度。默认宽度为 25
    item_height: int = 14, # 图例标记的图形高度。默认高度为 14
    legend_icon: Optional[str] = None, # 图例项的 icon。
)
```



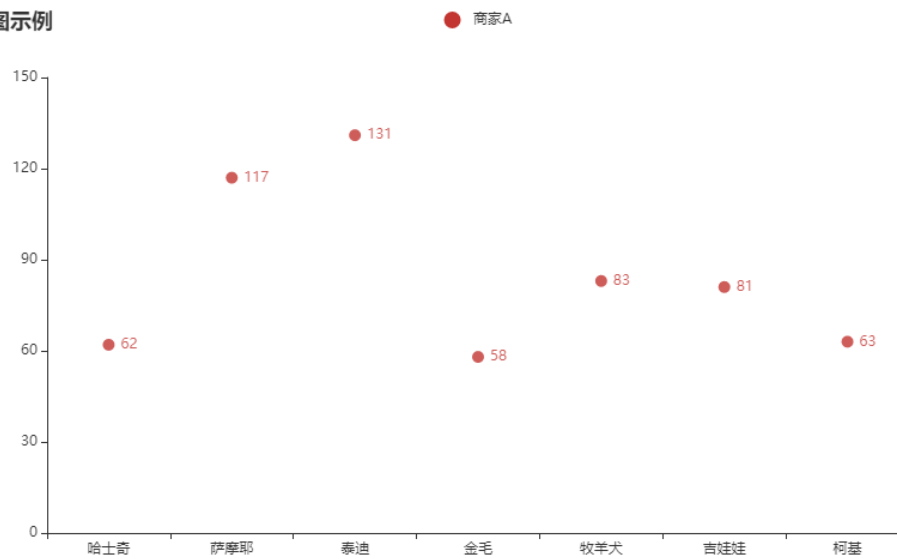
系列配置项综述

- 系列配置项：用于修改局部参数，常见的全局配置项包括标签配置项、文字样式配置项、线样式配置项、标记点配置项、标记线配置项、标记区域配置项、分隔区域配置项等。
 - 从以上这些“常用”配置项我们也能看出，其实系列配置项是为了**定制化配置**我们的Pyecharts图表而存在的。

散点图示例



散点图示例





系列配置项 - 标签配置项

- 标签配置项 (LabelOpts): 主要用于配置图表中各个标签的相关参数。

```
import pyecharts.options as opts
from pyecharts.charts import Bar
bar = Bar().set_series_opts(label_opts=opts.LabelOpts(is_show=False))
```

```
class LabelOpts(
    is_show: bool = True, # 是否显示标签。
    position: Union[str, Sequence] = "top", # 标签的位置。
    color: Optional[str] = None, # 文字的颜色。
    font_size: Numeric = 12, # 文字的字体大小。
    font_style: Optional[str] = None, # 文字字体的风格。可选: 'normal', 'italic', 'oblique'
    font_weight: Optional[str] = None, # 文字字体的粗细。可选: 'normal', 'bold', 'bolder', 'lighter'
    font_family: Optional[str] = None, # 文字的字体系列, 包括'Arial'和'Courier New'等
    rotate: Optional[Numeric] = None, # 标签旋转。从 -90 度到 90 度。正值是逆时针。
    margin: Optional[Numeric] = 8, # 刻度标签与轴线之间的距离。
    interval: Union[Numeric, str, None] = None, # 坐标轴刻度标签的显示间隔。
    horizontal_align: Optional[str] = None, # 文字水平对齐方式。
    vertical_align: Optional[str] = None, # 文字垂直对齐方式。
    formatter: Optional[str] = None, # 标签内容格式器, 模板变量有 {a}, {b}, {c}, {d}, {e}, 表示不同含义
)
```




系列配置项 - 标记点配置项

- 标记点配置项(MarkPointOpts): 在图中按需设置的标记点, 比如最大、最小值点。

- 使用规则:

```
# 标import pyecharts.options as opts #导入选项模块
from pyecharts.charts import Bar #导入工具库
bar = Bar() .set_series_opts(markpoint_opts=opts.MarkPointOpts(data=[opts.MarkPointItem(type_="max",name="
最大值")]))出每条折线的最大值
```

- 常用参数:

```
class MarkPointOpts(
    data: Sequence[Union[MarkPointItem, dict]] = None, # 标记点数据, 此处MarkPointItem指的是“标记点数据项”
    symbol: Optional[str] = None, # 标记的图形。提供的类型包括 'circle', 'triangle', 'diamond', 'arrow'等
    symbol_size: Union[None, Numeric] = None, # 标记的大小
    label_opts: LabelOpts = LabelOpts(position="inside", color="#fff"), # 标签配置项
)
```



系列配置项 - 标记点数据项

- 标记点数据项(MarkPointItem): 搭配MarkPointOpts使用

```
class MarkPointItem(  
    name: Optional[str] = None, # 标注名称。  
    type_: Optional[str] = None, # 特殊的标注类型，用于标注最大值最小值等。可选: 'min' 最大值; 'max' 最大值;  
    'average' 平均值。  
    value_index: Optional[Numeric] = None, # 在使用 type 时有效，用于指定在哪个维度上指定最大值最小值，可以是  
    0或1。  
    value_dim: Optional[str] = None, # 在使用 type 时有效，用于指定在哪个维度上指定最大值最小值。  
    coord: Optional[Sequence] = None, # 标注的坐标，比如直角坐标系上的 x, y  
    symbol: Optional[str] = None, # 标记的图形。  
    symbol_size: Union[Numeric, Sequence] = None, # 标记的大小  
    itemstyle_opts: Union[ItemStyleOpts, dict, None] = None, # 标记点样式配置项  
)
```



系列配置项 - 标记线配置项

- 标记线配置项(MarkLineOpts): 在图中按需设置的标记线, 比如平均值等。

- 使用规则:

```
import pyecharts.options as opts #导入选项模块
from pyecharts.charts import Bar #导入工具库
bar = Bar().set_series_opts(markline_opts=opts.MarkLineOpts(data=[opts.MarkLineItem(type_='max',name="最大值")])) # 标出每条折线的最大值
```

- 常用参数:

```
class MarkLineOpts(
    is_silent: bool = False, # 图形是否不响应和触发鼠标事件, 默认为 false, 即响应和触发鼠标事件。
    data: Sequence[Union[MarkLineItem, dict]] = None, # 标记线数据
    symbol: Optional[str] = None, # 标线两端的标记类型
    symbol_size: Union[None, Numeric] = None, # 标线两端的标记大小
    precision: int = 2, # 标线数值的精度, 在显示平均值线的时候有用。
    label_opts: LabelOpts = LabelOpts(), # 标签配置项
    linestyle_opts: Union[LineStyleOpts, dict, None] = None, # 标记线样式配置项
)
```



系列配置项 - 标记线数据项

- 标记线数据项(MarkLineItem): 搭配MarkLineOpts使用

```
class MarkLineItem(  
    name: Optional[str] = None, # 标注名称。  
    type_: Optional[str] = None, # 特殊的标注类型，用于标注最大值最小值等。可选: 'min' 最大值; 'max' 最大值;  
    'average' 平均值。  
    value_index: Optional[Numeric] = None, # 在使用 type 时有效，用于指定在哪个维度上指定最大值最小值，可以是  
    0或1。  
    value_dim: Optional[str] = None, # 在使用 type 时有效，用于指定在哪个维度上指定最大值最小值。  
    coord: Optional[Sequence] = None, # 标注的坐标，比如直角坐标系上的 x, y  
    symbol: Optional[str] = None, # 标记的图形。  
    symbol_size: Union[Numeric, Sequence] = None, # 标记的大小  
    itemstyle_opts: Union[ItemStyleOpts, dict, None] = None, # 标记点样式配置项  
)
```



基本图表

- Calendar: 日历图
- Funnel: 漏斗图
- Gauge: 仪表盘
- WordCloud: 词云图
- Liquid: 水球图
- Parallel: 平行坐标系
- Pie: 饼图
- Polar: 极坐标系
- Radar: 雷达图

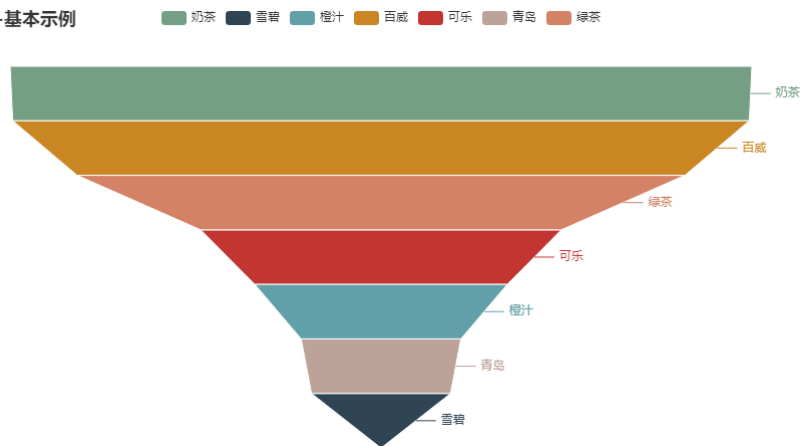
Liquid-基本示例



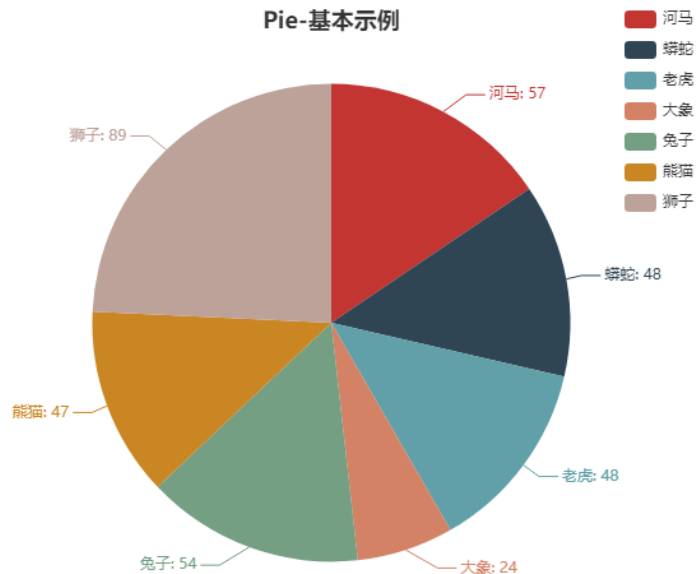
Gauge-基本示例



Funnel-基本示例



Pie-基本示例





基本图表 - 漏斗图

```
import pyecharts.options as opts
from pyecharts.charts import Funnel
from pyecharts.faker import Faker

c = (
    Funnel()
    .add("商品", [list(z) for z in zip(Faker.choose(), Faker.values())])
    .set_global_opts(title_opts=opts.TitleOpts(title="Funnel-基本示例"))
    .render("funnel_base.html")
)
```



基本图表 - 仪表盘图

```
import pyecharts.options as opts
from pyecharts.charts import Gauge

c = (
    Gauge()
    .add("", [{"完成率", 66.6}])
    .set_global_opts(title_opts=opts.TitleOpts(title="Gauge-基本示例", pos_left='center'))
    .render("gauge_base.html")
)
```



基本图表 - 水球图

```
import pyecharts.options as opts
from pyecharts.charts import Liquid
from pyecharts.faker import Faker

c = (
    Liquid()
    .add("lq", [0.6, 0.7])
    .set_global_opts(title_opts=opts.TitleOpts(title="Liquid-基本示例", pos_left='center', pos_top='10%'))
    .render("liquid_base.html")
)
```




基本图表 - 饼图

```
import pyecharts.options as opts
from pyecharts.charts import Pie
from pyecharts.faker import Faker

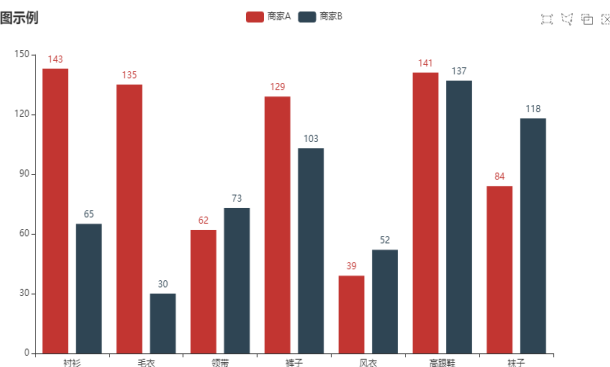
c = (
    Pie()
    .add("", [list(z) for z in zip(Faker.choose(), Faker.values())])
    .set_global_opts(
        title_opts=opts.TitleOpts(title="Pie-基本示例", pos_left='center'),
        legend_opts=opts.LegendOpts(
            pos_left="75%", orient="vertical"
        )
    )
    .set_series_opts(label_opts=opts.LabelOpts(formatter="{b}: {c}"))
    .render("pie_base.html")
)
```



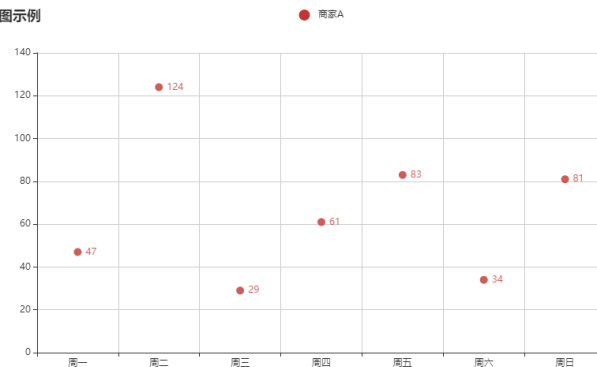
直角坐标系图

- Bar: 柱状图/条形图
- Line: 折线/面积图
- Scatter: 散点图
- Boxplot: 箱形图
- EffectScatter: 涟漪特效散点图
- HeatMap: 热力图
- Kline/Candlestick: K线图
- PictorialBar: 象形柱状图

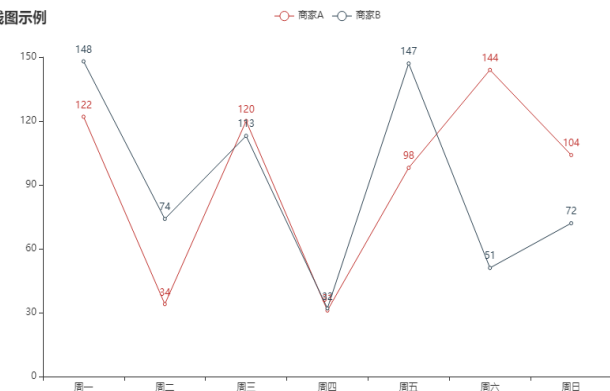
柱状图示例



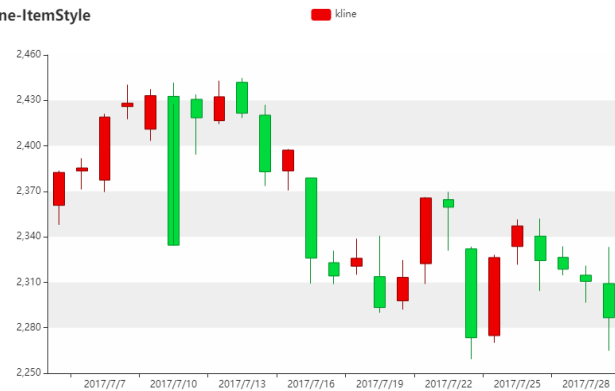
散点图示例



折线图示例



Kline-ItemStyle





直角坐标系图 - 柱状图示例

```
import pyecharts.options as opts #导入选项模块
from pyecharts.charts import Bar #导入工具库

attr = ["衬衫", "羊毛衫", "雪纺衫", "裤子", "高跟鞋", "袜子"] #数据准备: 横坐标数据
v1 = [5, 20, 36, 10, 75, 90] #数据准备: 第一组数据
v2 = [10, 25, 8, 60, 20, 80] #数据准备: 第二组数据

bar = (
    Bar() #实例化图形对象
    .add_xaxis(attr) #添加横坐标
    .add_yaxis("商家A", v1, stack="stack1") #添加第一组数据
    .add_yaxis("商家B", v2, stack="stack1") #添加第二组数据
    .set_series_opts(label_opts=opts.LabelOpts(is_show=False)) #系列化参数设置
    .set_global_opts(title_opts=opts.TitleOpts(title="柱状图数据堆叠示例"))) #全局参数设置
bar.render("可视化示例.html") #编译成html文件
```



直角坐标系图 - 散点图示例

```
import pyecharts.options as opts
from pyecharts.charts import Scatter
from pyecharts.faker import Faker

c = (
    Scatter()
    .add_xaxis(Faker.choose())
    .add_yaxis("商家A", Faker.values())
    .set_global_opts(
        title_opts=opts.TitleOpts(title="散点图示例"),
    )
    .render("scatter.html")
)
```



直角坐标系图 - 折线图示例

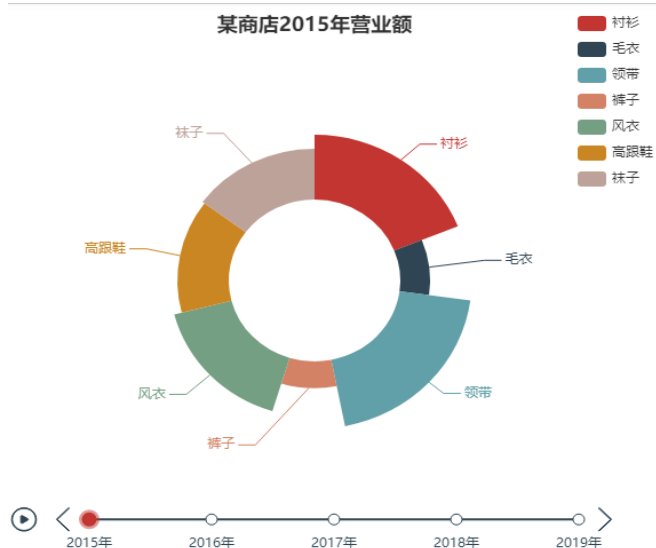
```
import pyecharts.options as opts
from pyecharts.charts import Line
from pyecharts.faker import Faker

c = (
    Line()
    .add_xaxis(Faker.choose())
    .add_yaxis("商家A", Faker.values())
    .add_yaxis("商家B", Faker.values())
    .set_global_opts(title_opts=opts.TitleOpts(title="折线图示例")) # 设置标题
    .set_series_opts(
        markpoint_opts=opts.MarkPointOpts(data=[opts.MarkPointItem(type_='max', name="最大值")]), # 标出每条折线的最大值
        markline_opts=opts.MarkLineOpts(data=[opts.MarkLineItem(type_='average', name='平均值')]) # 标出每条折线的平均值
    )
    .render("line_base.html")
)
```

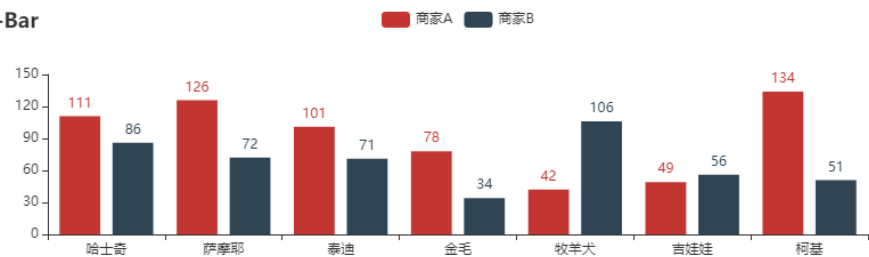


组合型图表

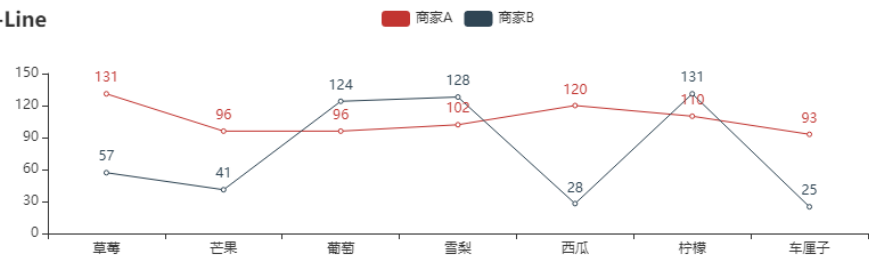
- Timeline: 时间线轮播多图
- Grid: 并行多图
- Tab: 选项卡多图
- Page: 顺序多图



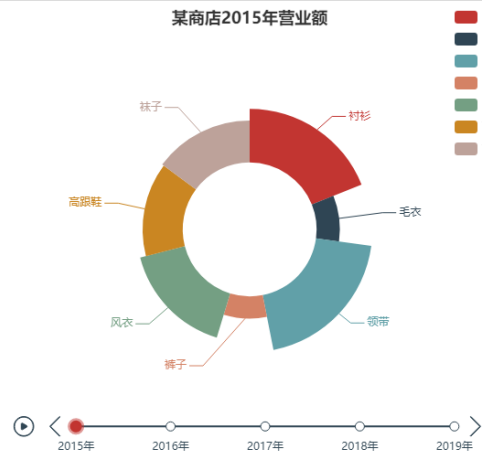
Grid-Bar



Grid-Line



timeline-pie grid_vertical





组合型图表 - 时间线轮播多图示例

```
import pyecharts.options as opts #导入选项模块
from pyecharts.charts import Pie, Timeline #导入工具库

attr = Faker.choose()
tl = Timeline()
for i in range(2015, 2020):
    pie = (
        Pie()
        .add("商家A", [list(z) for z in zip(attr, Faker.values())], rosetype="radius", radius=["30%", "55%"])
        .set_global_opts(title_opts=opts.TitleOpts("某商店{}年营业额".format(i), pos_left='center'),
            legend_opts=opts.LegendOpts(pos_left="75%", orient="vertical"))
    )
    tl.add(pie, "{}年".format(i))

tl.render("timeline_pie.html")
```



组合型图表 - 并行多图示例

```
from pyecharts import options as opts
from pyecharts.charts import Bar, Grid, Line
from pyecharts.faker import Faker

bar = (
    Bar()
    .add_xaxis(Faker.choose())
    .add_yaxis("商家A", Faker.values()))

line = (
    Line()
    .add_xaxis(Faker.choose())
    .add_yaxis("商家A", Faker.values())
    .set_global_opts(legend_opts=opts.LegendOpts(pos_top="48%")))

grid = (
    Grid()
    .add(bar, grid_opts=opts.GridOpts(pos_bottom="60%"))
    .add(line, grid_opts=opts.GridOpts(pos_top="60%"))
)

grid.render("grid_vertical.html")
```




组合型图表 - 选项卡多图示例

```
from pyecharts import options as opts
from pyecharts.charts import Tab
from pyecharts.faker import Faker

tab = Tab()
tab.add(tl, "timeline-pie")
tab.add(grid, "grid_vertical") # tl和grid分别在时间轴和并排多图中出现过
tab.render("tab_base.html")
```



组合型图表 - 顺序多图示例

```
from pyecharts import options as opts
from pyecharts.charts import Page
from pyecharts.faker import Faker

# SimplePageLayout 和 DraggablePageLayout 表示不同的布局设置
page = Page(layout=Page.DraggablePageLayout) # DraggablePageLayout表示可拖拽布局
page.add(tl,grid) # tl和grid分别在时间轴和并行多图中出现过
page.render("page_draggable_layout.html")
```



本章总结

- 本章主要介绍数据可视化的相关概念、基本思想，并且介绍Pyecharts可视化中常用的配置项和图表类型，并且演示基本代码，帮助学员更好的完成BI数据可视化。

浙大-华为鲲鹏创新头



学习推荐

- Pyecharts 中文官方学习资料库：
 - <http://pyecharts.org/#/zh-cn/intro>
- Pyecharts 源代码：
 - <https://github.com/pyecharts/pyecharts/>
- Echarts 官网：
 - <https://www.echartsjs.com/zh/index.html>



谢谢

www.huawei.com