Python

数据处理与分析

—— 基于 pandas

李庆辉

作者介绍

李庆辉，《深入浅出Pandas》作者，数据产品专家，某电商公司数据产品团队负责人，擅长通过数据治理、数据分析、数据化运营提升公司的数据应用水平。

精通 Python 数据科学及 Python Web 开发，曾独立开发公司的自动化数据分析平台，参与教育部“1+X”数据分析（Python）职业技能等级标准评审。

中国人工智能学会会员，企业数字化、数据产品和数据分析讲师，在个人网站“盖若”上编写的技术和产品教程广受欢迎。

作者网站：https://www.gairuo.com
公众号：盖若 （ID: gairuo）
作者微信：gairuo123

1 . Python 数据科学体系

2 .《深入浅出Pandas》介绍

3 . Python 基础

4 . Numpy

5 . Pandas

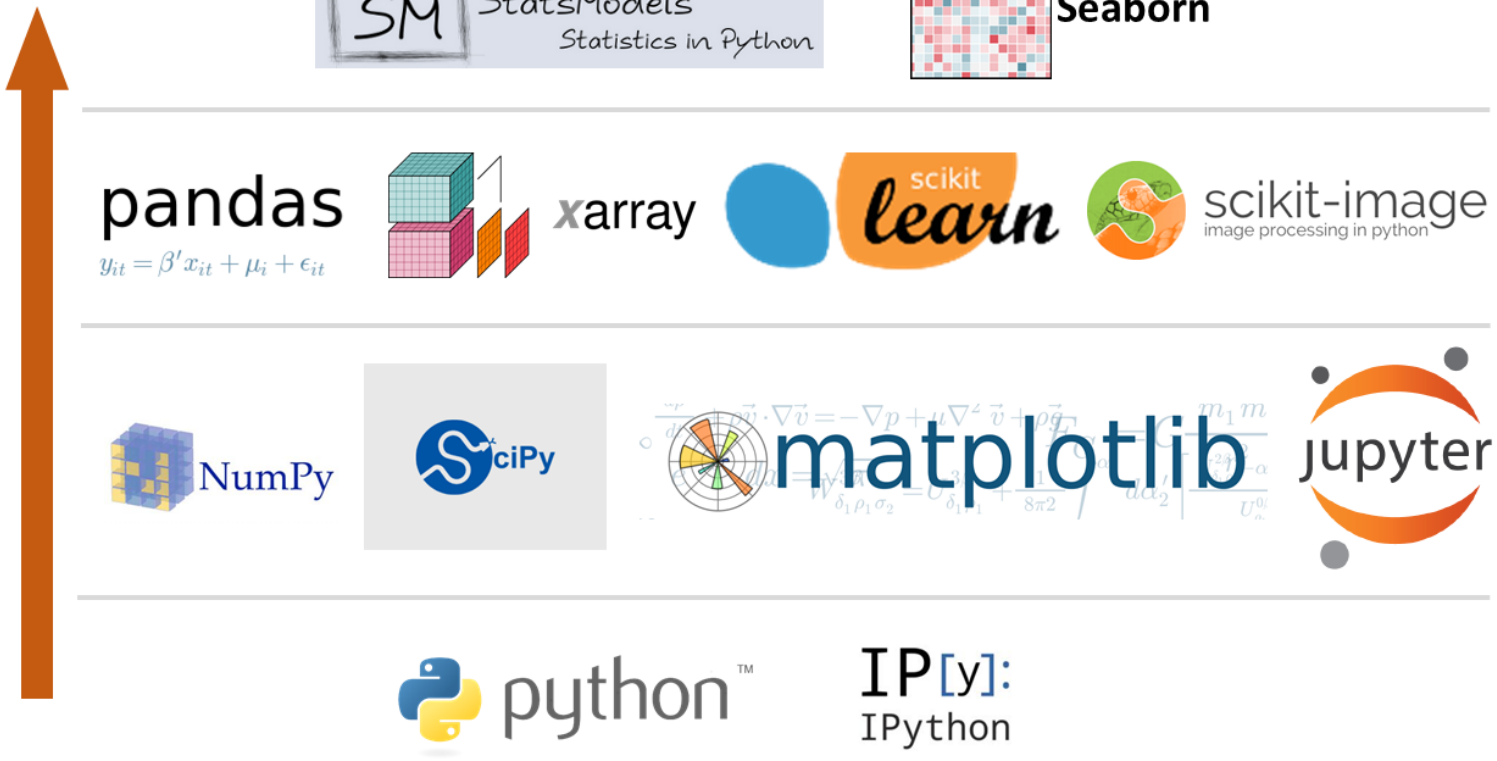
6. 可视化

7. 实践案例

PART 1

Python 数据科学体系

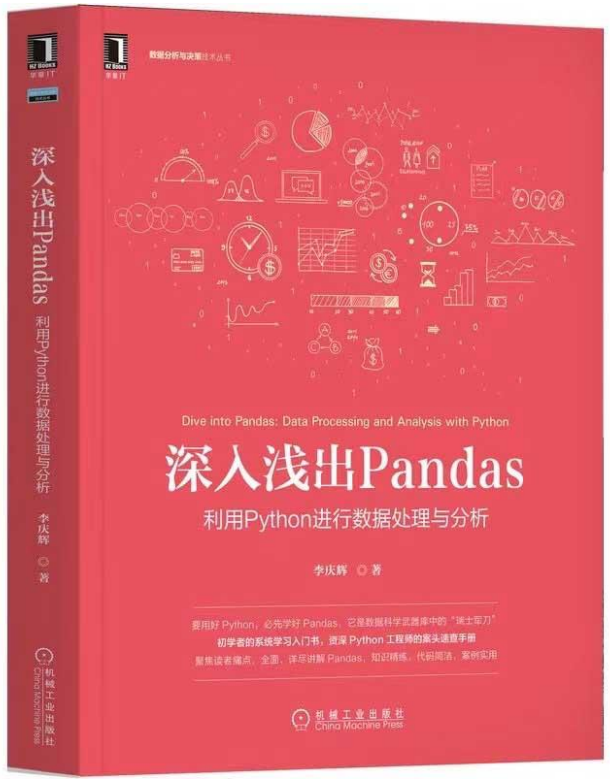
Python 数据分析生态

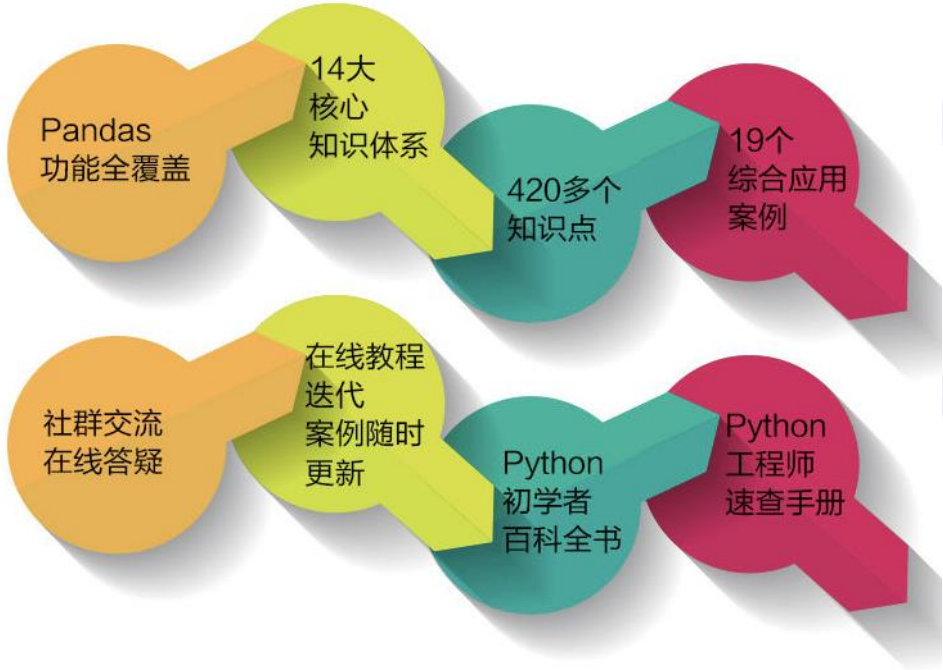


PART 2

《深入浅出Pandas》介绍

《深入浅出Pandas：利用Python进行数据处理与分析》



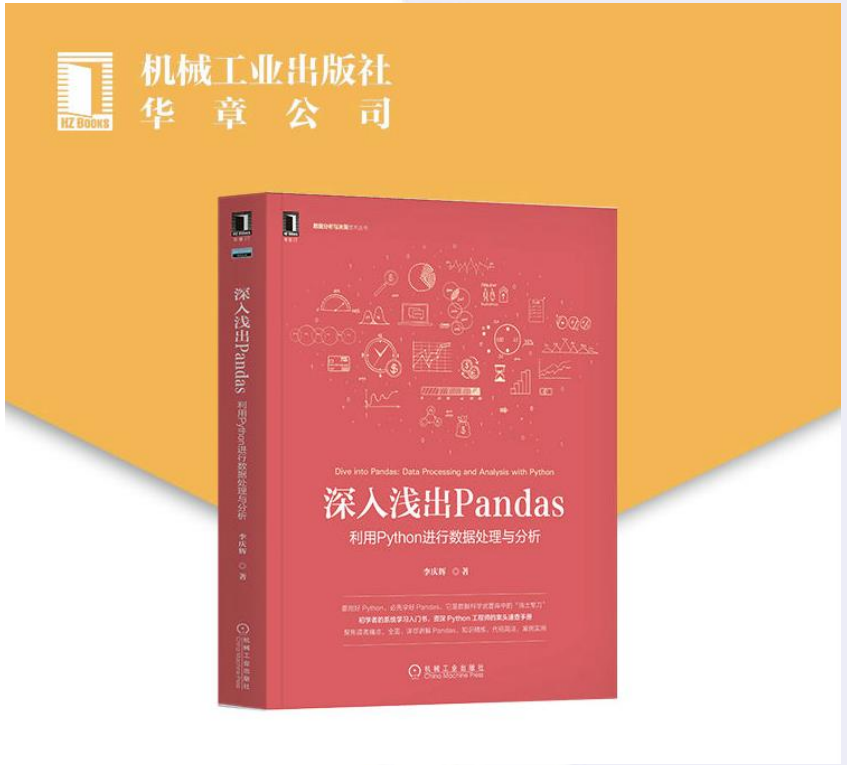


京东 https://item.jd.com/13344278.html

当当 http://product.dangdang.com/29271487.html

书籍特色

- ✧ 聚焦Pandas常用场景和痛点；
- ✧ 非技术思维，语言通俗易懂，面向应用；
- ✧ 不需要相关背景知识，不引入 Python 高级用法；
- ✧ 减少变量的传递，代码短小精练；
- ✧ 知识全面，几乎囊括 Pandas 所有函数和方法；
- ✧ 较少使用数学和统计学知识；
- ✧ 案例使用极简数据集，方便理解；
- ✧ 使用流行的链式方法，代码简洁，逻辑清晰，可读性强；
- ✧ 有大量实用案例。



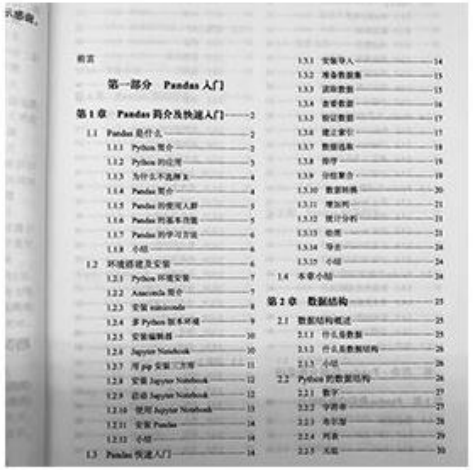
书籍网站：

https://www.gairuo.com/p/pandas

内容简介

本书共17章，分为七部分：

- ✧ 第一部分（第1～2章）Pandas入门
- ✧ 第二部分（第3～5章）Pandas数据分析基础
- ✧ 第三部分（第6～9章）数据形式变化
- ✧ 第四部分（第10～12章）数据清洗
- ✧ 第五部分（第13～14章）时序数据分析
- ✧ 第六部分（第15～16章）可视化
- ✧ 第七部分（第17章）实战案例



读者对象

- Excel中度、重度使用者，如文秘、公关人员、教师，从事行政、人力资源、市场和销售等工作的人员；
- 数据分析师、商业分析师、数据科学家；
- 互联网运营人员、数据运营人员；
- 互联网产品经理、项目经理；
- 开发人员、测试人员、算法人员；
- 财务、会计、金融从业者；
- 企业决策者、管理人员。

PART 2

Python 基础

环境搭建

1. 下载 miniconda：https://mirrors.tuna.tsinghua.edu.cn/anaconda/miniconda/

2. 进入终端，创建虚拟环境：

```
# 查看所有虚拟环境及当前环境
conda info -e
# 创建新环境，指定环境名称和Python版本
conda create -n py39data python=3.9
# 删除环境
conda remove -n py39data --all
# 进入、激活环境
conda activate py39data
# 退出环境
conda deactivate
```

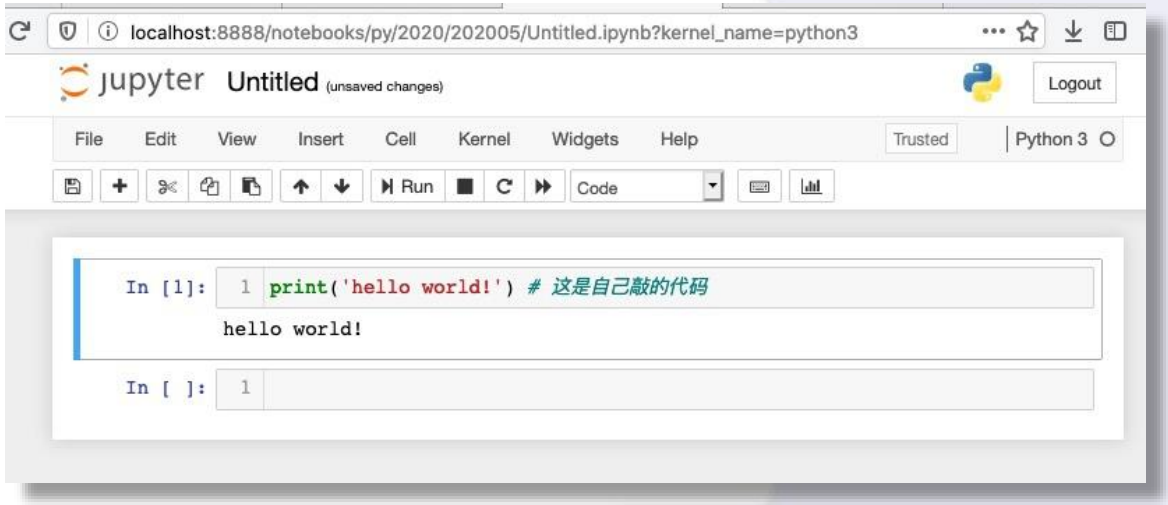
3.

```
# 安装 Jupyter Notebook，使用清华大学下载源加快下载速度
pip install jupyter -i https://pypi.tuna.tsinghua.edu.cn/simple
# 安装 Jupyter Lab 的命令如下
pip install jupyterlab
```

安装编辑器

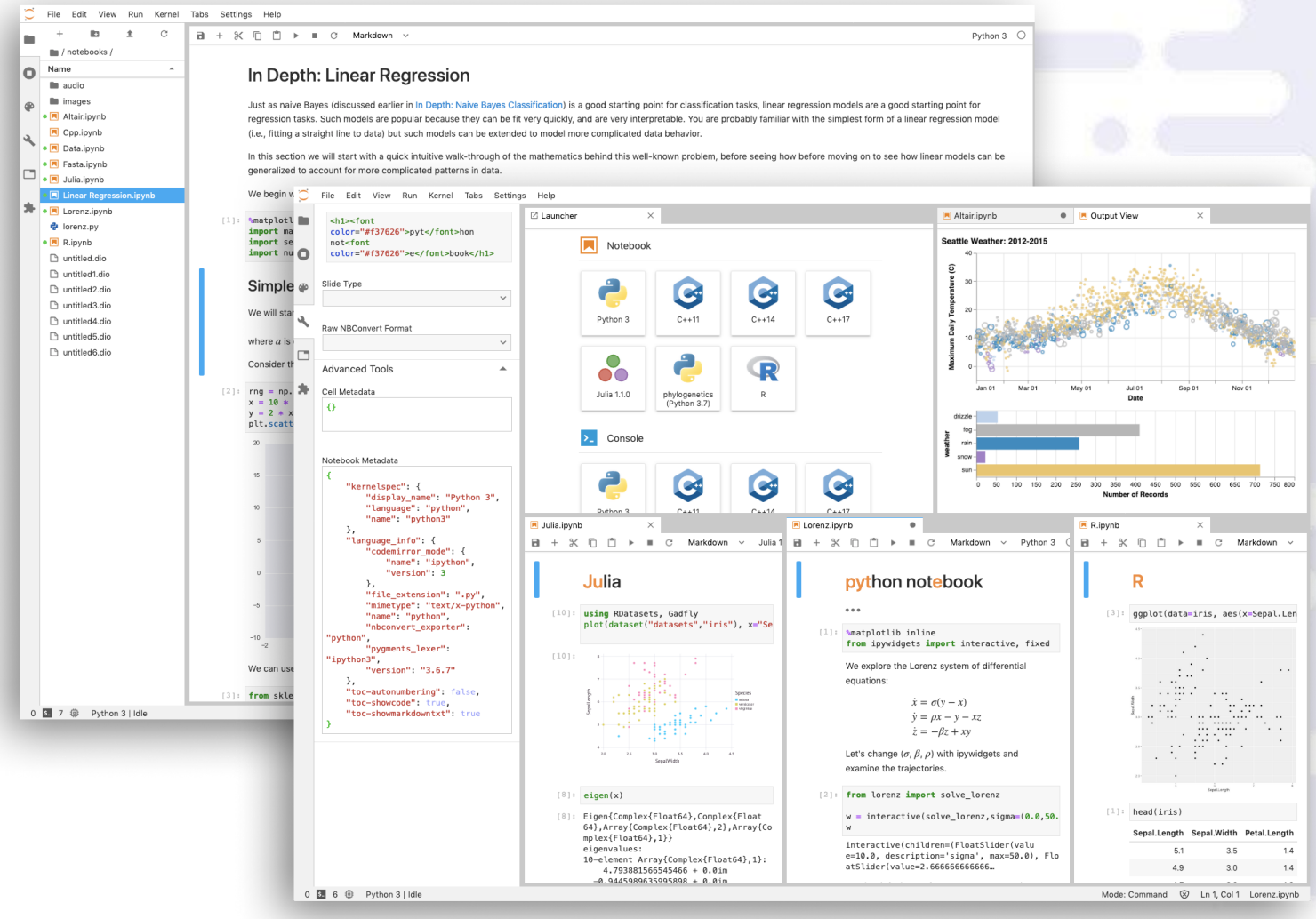
4. 启动编辑器：

```
jupyter notebook
```



JupyterLab

```
# 启动
jupyter lab
```



Python 数据类型

https://www.gairuo.com/p/python-sheet

类型	描述	示例
字符串 str	文本字符，空格也是一个字符	'Hello World!', '123', ''
数字 number	数字数据类型用于存储数值	1313，-323，0
布尔类型 boolean	逻辑真假	True, False
列表 list	一个列表的每个元素被分配一个数字来表示它的位置或索引	[1, 2, 3, 4, 5], ['apple', 'orange', 'lemon', 'cherry']
元组 tupple	可以简单地理解为“只读列表”	(1, 2, 3)， ('physics', 'math', 1997, 2000)
字典 dict	每个键与其值使用一个冒号(:)分开，键-值对是使用逗号分隔	`{'Name': 'Maxsu', 'Age': 7, 'Class': 'First'}`
集合 set	一组不重复key的集合，不存储value	set([1, 1, 2, 2, 3, 3]), `{'5元', '10元', '20元'}`

Python 基础功能

https://www.gairuo.com/p/python-sheet

功能	内容
流程控制	if else，while，for loop，break/continue，match case
函数	def，lambda，装饰器，生成器，async def，内置函数
类	class，继承，重写
模块	import 语句，内置模块
其他	解包，错误处理，类型注解，多任务（进程、线程、协程）

pandas 的功能

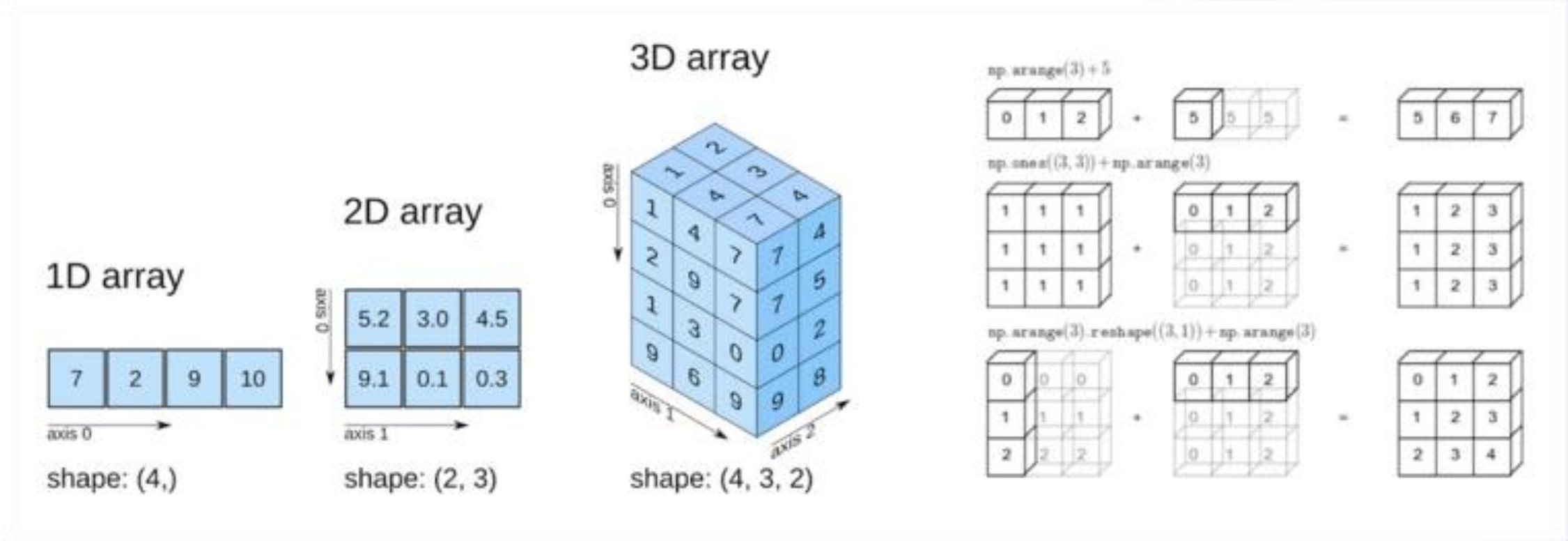
- 从Excel、CSV、网页、SQL、剪贴板等文件或工具中读取数据；
- 合并多个文件或者电子表格中的数据，将数据拆分为独立文件；
- 数据清洗，如去重、处理缺失值、填充默认值、补全格式、处理极端值；
- 建立高效的索引；
- 支持大体量数据；
- 按一定业务逻辑插入计算后的列、删除列；
- 灵活方便的数据查询、筛选；
- 分组聚合数据，可独立指定分组后的各字段计算方式；
- 数据的转置，如行转列、列转行变更处理；
- 连接数据库，直接用SQL 查询数据并进行处理；
- 对时序数据进行分组采样，如按季、按月、按工作小时，也可以自定义周期，如工作日；
- 窗口计算，移动窗口统计、日期移动等；
- 灵活的可视化图表输出，支持所有的统计图形；
- 为数据表格增加展示样式，提高数据识别效率。

PART 3

NumPy

NumPy 数据结构

https://www.gairuo.com/p/numpy-glance



NumPy 构造数据

https://www.gairuo.com/p/numpy-glance

```
# 导入 numpy 库，约定俗成别名为 np
import numpy as np

np.array([1, 2, 3])
np.array((1, 2, 3)) # 同上
# array([1, 2, 3])

np.array(((1, 2),(1, 2)))
np.array(([1, 2],[1, 2])) # 同上
# out:
array([[1, 2],
       [1, 2]])
```

NumPy 统计计算

https://www.gairuo.com/p/numpy-glance

```
np.array( [10,20,30,40] )[:3] # 支持类似列表的切片
a = np.array( [10,20,30,40] )
b = np.array( [1, 2, 3, 4] )
a+b # array([11, 22, 33, 44]) 矩阵相加
a-1 # array([ 9, 19, 29, 39])
4*np.sin(a)

# 以下举例数学函数，还支持非常多的数据函数
a.max() # 40
a.min() # 10
a.sum() # 100
a.std() # 11.180339887498949
a.all() # True
a.cumsum() # array([ 10,  30,  60, 100])
b.sum(axis=1) # 多维可以指定方向
np.negative(-1) # 1 相反数
```

PART 4

pandas

pandas 是什么

pandas 是一个 Python 包，提供了快速、灵活和富有表现力的数据设计，用于处理“关系”或“标记”数据的结构，简单直观。它的目标是用 Python 进行实际的数据分析。此外，它还更广泛的目标是成为最强大、最灵活的开放源代码数据任何语言的分析/操作工具。简单说，可以代替 Excel 做分析统计，实现批量、自动、复杂的数据处理和分析。

https://pandas.pydata.org/

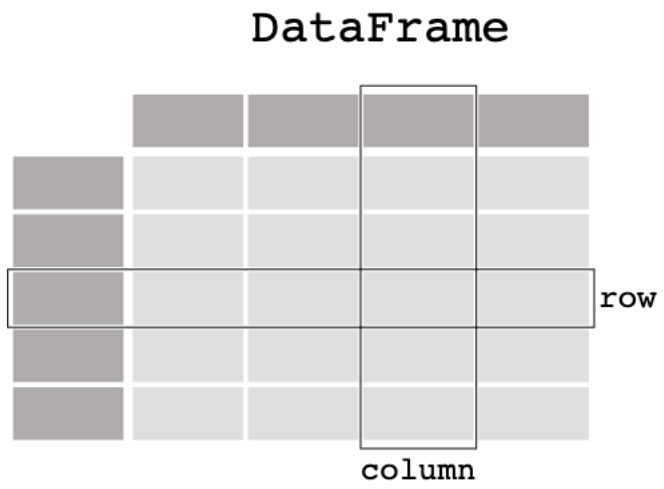
安装 pandas

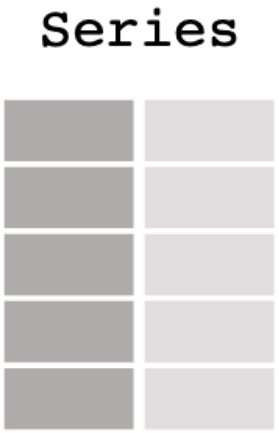
```
pip install pandas
# 如网络慢，可指定国内源快速下载安装
pip install pandas -i https://pypi.tuna.tsinghua.edu.cn/simple
```

```
# 与常用依赖包同时安装
pip install pandas xlrd openpyxl xlsxwriter requests lxml html5lib BeautifulSoup4
    matplotlib seaborn plotly bokeh scipy statsmodels -i https://pypi.tuna.
    tsinghua.edu.cn/simple
```

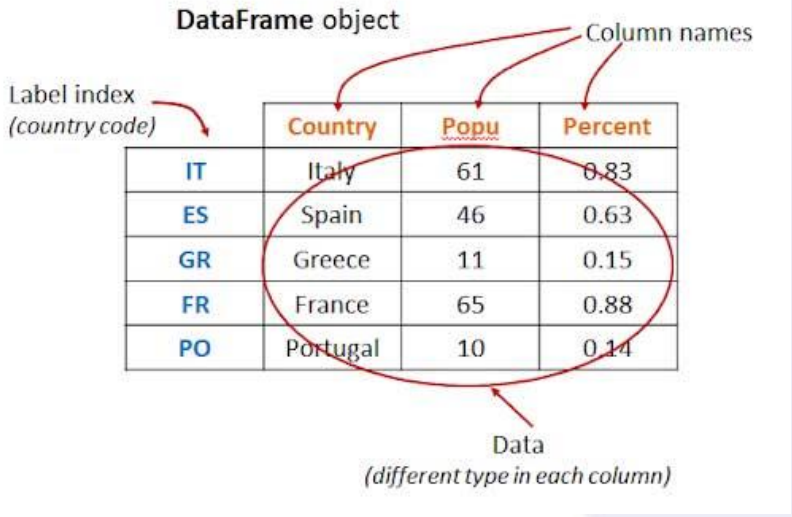
```
# 导入 pandas
import pandas as pd
pd.__version__
# '1.3.1'
```

数据结构





	A	B	C
1	国家	人口	GDP
2	中国	13.97	14.34
3	美国	3.28	21.43
4	日本	1.26	5.08
5			



- ✧ 索引 index
- ✧ 轴 axis
- ✧ 行 row、列 column
- ✧ 标签 label

构造数据

```python
s = pd.Series(['a', 'b', 'c', 'd', 'e'])
s
```

```
0    a
1    b
2    c
3    d
4    e
dtype: object
```

Series：

- 列表和元组
- 字典
- ndarray
- 标量

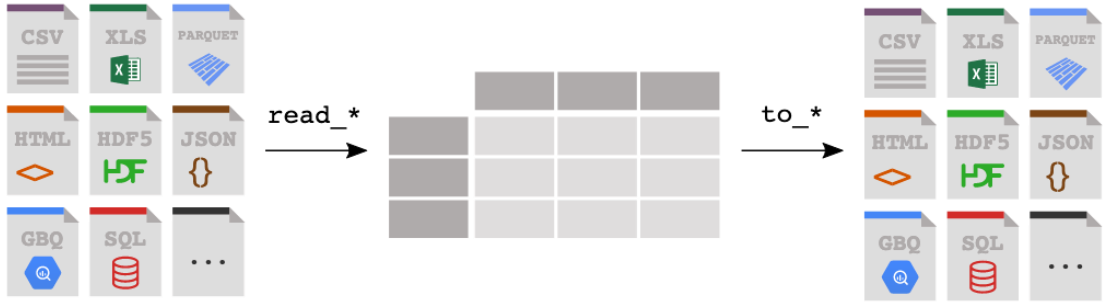
DataFrame：

- 字典（值为Series有可构造类型）
- Series
- pd.DataFrame.from_dict()
- pd.DataFrame.from_records()
- pd.json_normalize()

```python
df = pd.DataFrame({'国家': ['中国', '美国', '日本'],
                   '地区': ['亚洲', '北美', '亚洲'],
                   '人口': [13.97, 3.28, 1.26],
                   'GDP': [14.34, 21.43, 5.08],
                  })
df
```

	国家	地区	人口	GDP
0	中国	亚洲	13.97	14.34
1	美国	北美	3.28	21.43
2	日本	亚洲	1.26	5.08

读取/导出数据




```
# 读取 CSV
df = pd.read_csv('data.csv')
# 使用网址 url
pd.read_csv('https://www.gairuo.com/file/data/dataset/GDP-China.c

pd.read_excel('team.xlsx') # 默认读取第一个标签页 Sheet

pd.read_json('data.json')
pd.read_html(url, attrs={'id': 'table'})

pd.read_clipboard()
pd.read_sql_table('data', conn)
pd.read_xml(file_path, xpath="//book[year=2005]")
```

Format Type	Data Description	Reader	Writer
text	CSV	read_csv	to_csv
text	Fixed-Width Text File	read_fwf	
text	JSON	read_json	to_json
text	HTML	read_html	to_html
text	LaTeX		Styler.to_latex
text	XML	read_xml	to_xml
text	Local clipboard	read_clipboard	to_clipboard
binary	MS Excel	read_excel	to_excel
binary	OpenDocument	read_excel	
binary	HDF5 Format	read_hdf	to_hdf
binary	Feather Format	read_feather	to_feather
binary	Parquet Format	read_parquet	to_parquet
binary	ORC Format	read_orc	
binary	Stata	read_stata	to_stata
binary	SAS	read_sas	
binary	SPSS	read_spss	
binary	Python Pickle Format	read_pickle	to_pickle
SQL	SQL	read_sql	to_sql
SQL	Google BigQuery	read_gbq	to_gbq

数据类型

类型	Python 原生类型	NumPy类型	用途
object	str or mixed	string_, unicode_, mixed types	文本或者混合数字
int64	int	int_, int8, int16, int32, int64, uint8, uint16, uint32, uint64	整型数字
float64	float	float_, float16, float32, float64	浮点数字
bool	bool	bool_	True/False 布尔型
datetime64[ns]	nan	datetime64[ns]	日期时间
timedelta[ns]	nan	nan	两个时间之间的距离，时间差
category	nan	nan	有限文本值，枚举

```
df.dtypes # 各字段的数据类型
df.team.dtype # 某个字段的类型
s.dtype # S 的类型
df.dtypes.value_counts() # 各类型有多少个字段
```

```
df.astype('int32') # 所有数据转换为 int32
df.astype({'col1': 'int32'}) # 指定字段转指定类型

# 自动转换合适的数据类型
df.convert_dtypes() # 推荐！新的方法，支持 string 类型
df.infer_objects()
```

索引处理

- 行索引是数据的索引，列索引指向的是一个 Series
- DataFrame 的索引也是系列形成的 Series 的索引
- 建立索引让数据更加直观明确，如每行数据是针对一个国家的
- 建立索引方便数据处理
- 索引允许重复，但业务上一般不会让它重复

	国家	地区	人口	GDP
0	中国	亚洲	13.97	14.34
1	美国	北美	3.28	21.43
2	日本	亚洲	1.26	5.08

```
df.index # 查看索引
df.columns # 查看列索引
```

```
data = 'https://www.gairuo.com/file/data/dataset/team.xlsx'
df = pd.read_excel(data, index_col='name') # 设置索引为 name
```

```
s = pd.Series([1, 2, 3, 4])
df.set_index(s) # 指定一个索引
```

```
df.reset_index() # 重置索引
df.rename(columns={"A": "a", "B": "c"}) # 一一对应修改列索引
df.set_axis(['a', 'b', 'c'], axis='index') # 修改索引
df.rename_axis('info', axis="columns") # 修改行索引名
```

数据信息

```python
df.head(10) # 查看前10条数据
df.tail() # 查看后五条数据
df.tail(10) # 查看后10条数据
df.sample() # 随机查看一条数据
df.sample(3) # 随机查看3条数据
df.sample(10, ignore_index=True) # 随机后重置索引
df.shape # (100, 6) 数据形状
df.info() # 数据信息

df.index  # RangeIndex(start=0, stop=100, step=1)
df.columns # 列索引，Series 不支持
df.dtypes # 数据类型
df.axes  # 行列索引内容

df.size # 600 行x列的总数，就是总共有多少数据
df.empty # False

df.to_numpy() # numpy array(<所有值的列表矩阵>)
df.attrs={'info': '学生成绩表'} # 设置元信息，可用于数据基础信息描述
```

```python
1 df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100 entries, 0 to 99
Data columns (total 6 columns):
 #   Column  Non-Null Count  Dtype 
---  ------  --------------  ----- 
 0   name    100 non-null    object
 1   team    100 non-null    object
 2   Q1      100 non-null    int64 
 3   Q2      100 non-null    int64 
 4   Q3      100 non-null    int64 
 5   Q4      100 non-null    int64 
dtypes: int64(4), object(2)
memory usage: 4.8+ KB
```

数学统计

```
df.describe() # 描述性统计
df.mean() # 返回所有列的均值
df.mean(1) # 返回所有行的均值，下同
df.corr() # 返回列与列之间的相关系数
df.count() # 返回每一列中的非空值的个数
df.max() # 返回每一列的最大值
df.min() # 返回每一列的最小值
df.abs() # 绝对值
df.median() # 返回每一列的中位数
df.std() # 返回每一列的标准差，贝塞尔校正的样本标准偏差
df.var() # 无偏方差
df.sem() # 平均值的标准误差
```

```
df.mode() # 众数
df.prod() # 连乘
df.mad() # 平均绝对偏差
df.cumprod() # 累积连乘，累乘
df.cumsum(axis=0) # 累积连加，累加
df.nunique() # 去重数量，不同值的量
df.idxmax() # 每列最大的值的索引名
df.idxmin() # 最小
df.cummax() # 累积最大值
df.cummin() # 累积最小值
df.skew() # 样本偏度（第三阶）
df.kurt() # 样本峰度（第四阶）
df.quantile() # 样本分位数（不同 % 的值）
```

```
1 df.describe()
```

	Q1	Q2	Q3	Q4
count	100.000000	100.000000	100.000000	100.000000
mean	49.200000	52.550000	52.670000	52.780000
std	29.962603	29.845181	26.543677	27.818524
min	1.000000	1.000000	1.000000	2.000000
25%	19.500000	26.750000	29.500000	29.500000
50%	51.500000	49.500000	55.000000	53.000000
75%	74.250000	77.750000	76.250000	75.250000
max	98.000000	99.000000	99.000000	99.000000

数据计算

```
# 位差
df.diff()
df.diff(axis=1) # 向右一列
df.diff(2)
df.diff(-1) # 新的本行为本行减去后一行

# 整体下移一行，最顶的一行为 NaN
df.shift()
df.shift(3) # 移三行

# 排名，将值变了序号数
df.rank()
df.rank(axis=1) # 横向排
```

```
df + 1 # 等运算
df.add() # 加
df.sub() # 减
df.mul() # 乘
df.div() # 除
df.divmod() # 返回 (a // b, a % b)
df.truediv() # Divide DataFrames (float division).
df.floordiv() # Divide DataFrames (integer division).
df.mod() # 模，除后的余数
df.pow() # 指数幂
df.dot(df2) # 矩阵运算
```

```
df.all() # 返回所有列all()值的Series
df.any()

# 用表达式计算生成列。仅支持列，不是太安全
df.eval('Q2Q3 = Q2 + Q3')
df.eval('Q2Q3 = Q2 + Q3', inplace=True) # 替换生效

# 四舍五入
df.round(2) # 指定字段指定保留小数位，如有
df.round({'Q1': 2, 'Q2': 0})
df.round(-1) # 保留10位

# 每个列的去重值的数量
df.nunique()
s.nunique() # 本列的去重值

# 真假检测
df.isna() # 值的真假值替换
df.notna() # 与上相反
```

查询筛选

```
df['name'] # 会返回本列的 Series
df.name
df.Q1
# df.1Q 即使列名叫 1Q 也无法使用
# df.my name 有空格也无法调用，可以处理加上下划线
```

```
# 数据截取，去掉索引之前和之后的数据
df.truncate(before=2, after=4) # 只要索引 2-4
s.truncate(before="60", after="66")
df.truncate(before="A", after="B", axis="columns") #
选取列
```


```
# 数据截取，去掉索引之前和之后的数据
df.truncate(before=2, after=4) # 只要索引 2-4
s.truncate(before="60", after="66")
df.truncate(before="A", after="B", axis="columns") #
选取列
```


```
# 注：索引是字符需要加引号
df.at[4, 'Q1'] # 65
df.at['lily', 'Q1'] # 65 假定索引是 name
df.at[0, 'name'] # 'Liver'
df.loc[0].at['name'] # 'Liver'
```

```
s.where(s > 90, 0) # 不符合条件的为 0
np.where(s>80, True, False)
s.mask(s > 90, 0) # 符合条件的为 0

df.query('Q1 > Q2 > 90') # 直接写类似 sql where 语句
df.query('Q1 + Q2 > 180')
df.query('Q1 == Q2')
df.query('(Q1<50) & (Q2>40) and (Q3>90)')

df.filter(items=['Q1', 'Q2']) # 选择两列
df.filter(regex='Q', axis=1) # 列名包含Q的
df.filter(regex='e$', axis=1) # 以 e 结尾的

df.select_dtypes(include=['float64']) # 选择 float64 型数
据
df.select_dtypes(include='bool')
df.select_dtypes(include=['number']) # 只取数字型
df.select_dtypes(exclude=['int']) # 排除 int 类型
df.select_dtypes(exclude=['datetime64'])

# Q1 Q2 成绩全为 80 分的
df[(df.loc[:,['Q1','Q2']] > 80).all(1)]
# Q1 Q2 成绩至少有一个 80 分的
df[(df.loc[:,['Q1','Q2']] > 80).any(1)]
```

行列复杂查询

操作	语法	返回结果
选择列	`df[col]`	Series
按索引选择行	`df.loc[label]`	Series
按数字索引选择行	`df.iloc[loc]`	Series
使用切片选择行	`df[5:10]`	DataFrame
用表达式筛选行	`df[bool_vec]`	DataFrame

```
dft.loc[:, ['Q1', 'Q2']] # 所有行，Q1 和 Q2两列
dft.loc[:, ['Q1', 'Q2']] # 所有行，Q1 和 Q2两列
dft.loc[:10, 'Q1':]

df.iloc[:3]
df.iloc[:]
df.iloc[:, [1, 2]]
df.iloc[2:20:3]
s.iloc[:3]

df[df['Q1'] == 8] # Q1 等于8
df[~(df['Q1'] == 8)] # 不等于8
df[df.name == 'Ben'] # 姓名为Ben
df.loc[df['Q1'] > 90, 'Q1':]  # Q1 大于90，显示Q1后边的所有列
df.loc[(df.Q1 > 80) & (df.Q2 < 15)] # and 关系
df.loc[(df.Q1 > 90) | (df.Q2 < 90)] # or 关系
df[df.Q1 > df.Q2]
```

```
# 通过列位置筛选列
df.loc[:, lambda df: df.columns.str.len()==4] # 真假组成的序列
df.loc[:, lambda df: [i for i in df.columns if 'Q' in i]] # 列名列表
df.iloc[:3, lambda df: df.columns.str.len()==2] # 真假组成的序列
```

```
df.eq() # 等于相等 ==
df.ne() # 不等于 !=
df.le() # 小于等于 >=
df.lt() # 小于 <
df.ge() # 大于等于 >=
df.gt() # 大于 >
df[df.Q1.ne(89)] # Q1 不等于8
df.loc[df.Q1.gt(90) & df.Q2.lt(90)] # and 关系 Q1>90 Q2<90
# isin
df[df.team.isin(['A','B'])] # 包含 AB 两组的
df[df.isin({'team': ['C', 'D'], 'Q1':[36,93]})] # 复杂查询，其他值为 NaN
```

排序

```python
# 索引排序
s.sort_index() # 升序排列
df.sort_index() # df 也是按索引进行排序
df.team.sort_index()
s.sort_index(ascending=False) # 降序排列

# 数值排序
s.sort_values() # 升序
s.sort_values(ascending=False) # 降序
# 全降序
df.sort_values(by=['team', 'Q1'], ascending=False)
# 对应指定team升Q1降
df.sort_values(by=['team', 'Q1'], ascending=[True, False])
```

```python
1 df.sort_values(by=['team', 'Q1'], ascending=[True, False])
```

	name	team	Q1	Q2	Q3	Q4
88	Aaron	A	96	75	55	8
17	Henry	A	91	15	75	17
70	Nathan	A	87	77	62	13
42	Dylan	A	86	87	65	20
71	Blake	A	78	23	93	9
...	...	...	...	...	...	...
98	Eli	E	11	74	58	91
43	Jude	E	8	45	13	65
45	Rory9	E	8	12	58	27
61	Jackson5	E	6	10	15	33
82	Finn	E	4	1	55	32

100 rows × 6 columns

```python
s.nsmallest(3) # 最小的三个
s.nlargest(3) # 最大的三个
# 指定列
df.nlargest(3, 'Q1')
df.nlargest(5, ['Q1', 'Q2'])
df.nsmallest(5, ['Q1', 'Q2'])
```

添加/修改/删除数据

```python
df.Q1 = [1, 3, 5, 7, 9] * 20 # 就会把值进行修改
df.loc[1:3, 'Q1':'Q2'] = 99 # 这个范围的数据会全变成99

# 对索引值进行修改
df.rename(columns={"Q1": "a", "Q2": "b"}) # 对表头进行修改
df.rename(index={0: "x", 1: "y", 2: "z"}) # 对索引进行修改

df.replace(0, 5) # 将数据中所有 0 换为 5
df.fillna(0) # 空全修改为 0

df['foo'] = 100 # 增加一列 foo，所有值都是 100
df['foo'] = df.Q1 + df.Q2 # 新列为两列相加
df['foo'] = df['Q1'] + df['Q2'] # 同上

# 增加一列并赋值，不满足条件的为 NaN
df.loc[df.num >= 60, '成绩'] = '合格'
df.loc[df.num < 60, '成绩'] = '不合格'
```

```python
1 df.assign(total=df.select_dtypes('number').sum(1))
```

	name	team	Q1	Q2	Q3	Q4	total
0	Liver	E	89	21	24	64	198
1	Arry	C	36	37	37	57	167
2	Ack	A	57	60	18	84	219
3	Eorge	C	93	96	71	78	338
4	Oah	D	65	49	61	86	261
...	...	...	...	...	...	...	...

```python
df.assign(Q5=[100]*100) # 新增加一列 Q5
df = df.assign(Q5=[100]*100) # 赋值生效
df.assign(Q6=df.Q2/df.Q1) # 计算并增加 Q6

df['C1'] = df.eval('Q2 + Q3')
df.eval('C2 = Q2 + Q3') # 计算

df.pop('Q1') # 删除一列
s.pop(3) # 删除一个索引位
# 也可以把想要的列筛选出来赋值给 df 达到删除的目的
df.drop(['B', 'C'], axis=1) # 删除两列
df.drop(columns=['B', 'C']) # 同上
```

数据迭代遍历

```python
# 按行
for index, row in df.iterrows():
    print(index, row['name'], row.team, row.Q1)

# 按行，命名元组
for row in df.itertuples():
    print(row)
# Pandas(Index=0, name='Liver', team='E', Q1=89, Q2=21, Q3=24,
Q4=64)

# 按列
for label, ser in df.loc[1].items():
    print(label, ser)
```

[101]:

```python
for index, row in df.iterrows():
    print(index, row['name'], row.team, row.Q1)
    print('*'*10)
```

```
0 Liver E 89
**********
1 Arry C 36
**********
2 Ack A 57
**********
3 Eorge C 93
**********
4 Oah D 65
**********
5 Harlie C 24
**********
6 Acob B 61
**********
7 Lfie A 9
**********
8 Reddie D 64
**********
```

函数应用

```python
# 其中 xy 是变量，df_ 是前边的数据内容
df.pipe(lambda df_, x, y: df_[(df_.a >= x) & (df_.b <= y)],
2, 8)
df.apply(fun) # 自定义函数
df.applymap(lambda x: x*2) # 所有元素的最大值
df.team.map({'A':'一班', 'B':'二班','C':'三班', 'D':'四班',})
# 枚举替换
# 将所有列聚合产生 sum 和 min 两行
df.agg(['sum', 'min'])
df.transform({'A': np.abs, 'B': lambda x: x + 1})
```

[105]:
```python
df.pipe(len)
```

[105]:
```
100
```

[104]:
```python
def mymax(x):
    return x.max()

df.apply(mymax) # 应用函数
```

[104]:
```
name    Zachary
team          E
Q1           98
Q2           99
Q3           99
Q4           99
dtype: object
```

[107]:
```python
df.agg(['max', 'min'])
```

[107]:

	name	team	Q1	Q2	Q3	Q4
max	Zachary	E	98	99	99	99
min	Aaron	A	1	1	1	2

分组聚合

```python
df.groupby('team').describe() # 描述性统计
df.groupby('team').sum() # 求和

# 所有列使用一个计算计算方法
df.groupby('team').aggregate(sum
df.groupby('team').agg(sum)
grouped.agg(np.size)
grouped['Q1'].agg(np.mean)

## 多个计算方法
# 所有列指定多个计算方法
grouped.agg([np.sum, np.mean, np
# 指定列使用多个计算方法
grouped['Q1'].agg([sum, np.mean, np.std])
# 一列使用多个计算方法
df.groupby('team').agg({'Q1': ['min', 'max'], 'Q2':
'sum'})
```

```
[117]: 1 df.groupby('team').sum()
```

[117]:

team	Q1	Q2	Q3	Q4
A	1066	639	875	783
B	975	1218	1202	1136
C	1056	1194	1068	1127
D	860	1191	1241	1199
E	963	1013	881	1033

```python
df.groupby('team').transform(np.std) # 标准差

# 所有成员平均成绩大于 60 的组
df.groupby(['team']).filter(lambda x: (x.mean() >=
60).all())

# 一个分组导出一个 Excel 文件
(
    df.groupby('team')
    .apply(lambda x: x.to_excel(f'{x.name}.xlsx'))
)

# 最大值和最小会之间的差值
df.groupby('team').pipe(lambda x: x.max() +
x.min())

# 按月重采样
df.groupby('a').resample('M').sum()
```

数据分箱（分桶）

```
# 不用区间，使用数字做为标签（0，1，2，n）
pd.cut(df.Q1, bins=[0, 60, 100],labels=False)
# 指定标签名
pd.cut(df.Q1, bins=[0, 60, 100],labels=['不及格','及
格',])
# 包含最低部分
pd.cut(df.Q1, bins=[0, 60, 100],
include_lowest=True)
# 是否包含右边，闭区间，下例 [89, 100)
pd.cut(df.Q1, bins=[0, 89, 100], right=False)

pd.qcut(range(5), 4)
pd.qcut(range(5), 4, labels=False)
# 指定标签名
pd.qcut(range(5), 3, labels=["good", "medium",
"bad"])
```

```
[119]: 1 pd.cut(df.Q1, bins=[0, 60, 100],labels=['不及格','及格',])

[119]: 0      及格
       1     不及格
       2     不及格
       3      及格
       4      及格
             ...
       95    不及格
       96    不及格
       97     及格
       98    不及格
       99    不及格
       Name: Q1, Length: 100, dtype: category
       Categories (2, object): ['不及格' < '及格']
```

数据连接

```
frames = [df1, df2, df3]
df = pd.concat(frames)
df = pd.concat(frames, keys=['x', 'y', 'z'])

result = df1.append(df2)

# 以左为表基表
pd.merge(left, right, how='left', on=['key1',
'key2'])

pd.merge_asof(left, right, on='a')
df1.combine_first(df2)
df.compare(df2)
```

```
# 以右为表基表
pd.merge(left, right, how='right', on=['key1', 'key2'])
```

left

	key1	key2	A	B
0	K0	K0	A0	B0
1	K0	K1	A1	B1
2	K1	K0	A2	B2
3	K2	K1	A3	B3

right

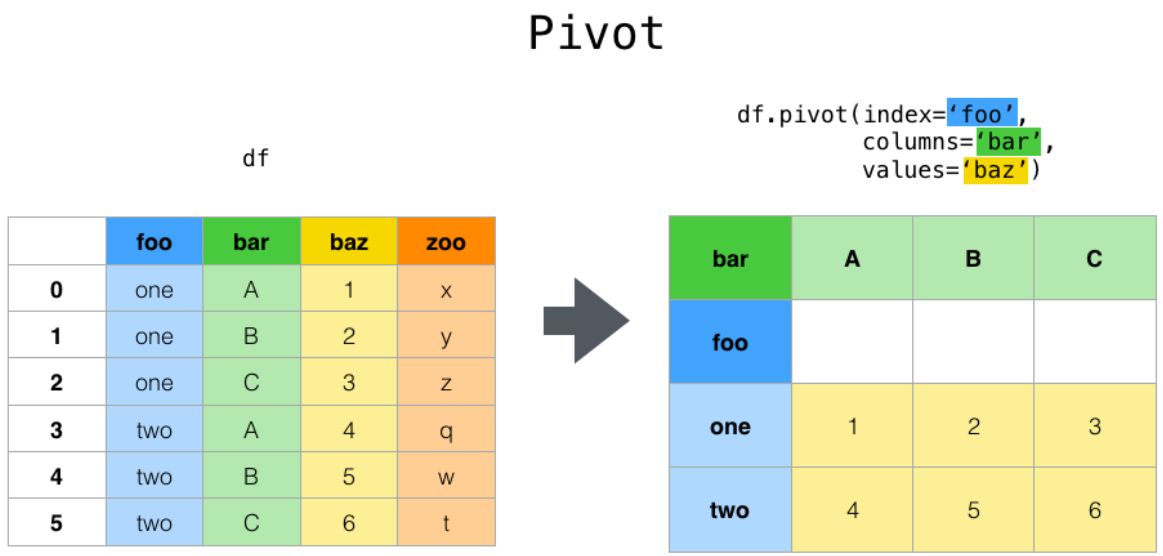
	key1	key2	C	D
0	K0	K0	C0	D0
1	K1	K0	C1	D1
2	K1	K0	C2	D2
3	K2	K0	C3	D3

Result

	key1	key2	A	B	C	D
0	K0	K0	A0	B0	C0	D0
1	K1	K0	A2	B2	C1	D1
2	K1	K0	A2	B2	C2	D2
3	K2	K0	NaN	NaN	C3	D3

数据透视

```
# 透视
df.pivot(index='foo', columns='bar', values='baz')
```



	foo	bar	baz	zoo
0	one	A	1	x
1	one	B	2	y
2	one	C	3	z
3	two	A	4	q
4	two	B	5	w
5	two	C	6	t

bar	A	B	C
foo			
one	1	2	3
two	4	5	6

数据堆叠

Stack

df2

		A	B
first	second		
bar	one	1	2
	two	3	4
baz	one	5	6
	two	7	8

MultiIndex

stacked = df2.stack()

first	second		
bar	one	A	1
		B	2
	two	A	3
		B	4
baz	one	A	5
		B	6
	two	A	7
		B	8

MultiIndex

Unstack

stacked

first	second		
bar	one	A	1
		B	2
	two	A	3
		B	4
baz	one	A	5
		B	6
	two	A	7
		B	8

MultiIndex

stacked.unstack()

		A	B
first	second		
bar	one	1	2
	two	3	4
baz	one	5	6
	two	7	8

MultiIndex

数据融合

Melt

df3

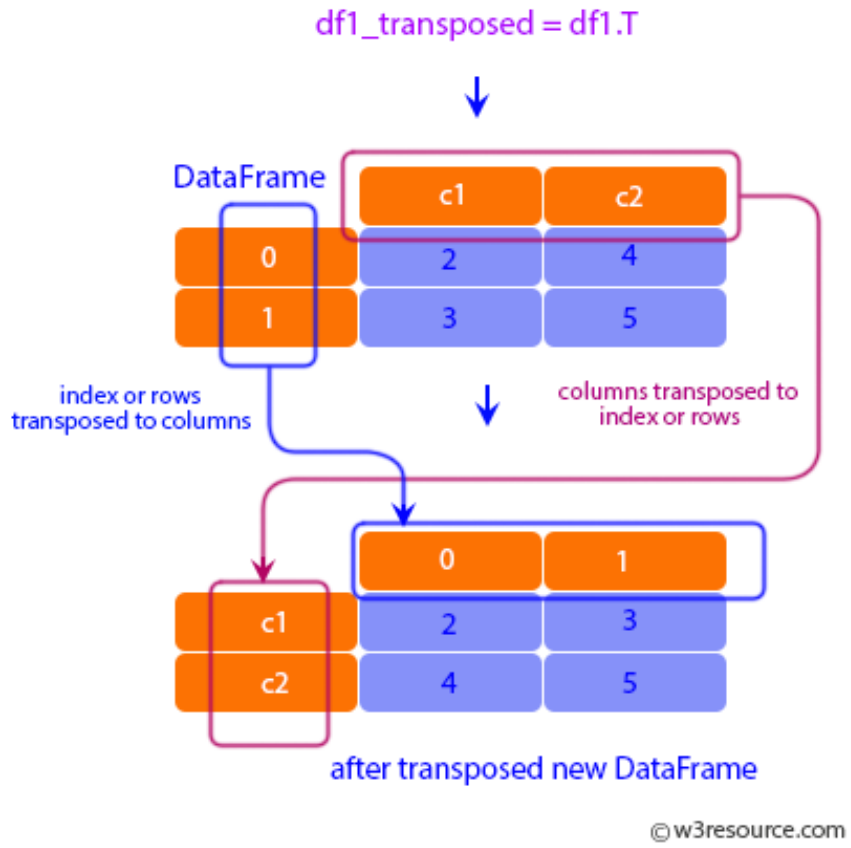
	first	last	height	weight
0	John	Doe	5.5	130
1	Mary	Bo	6.0	150

```
df3.melt(id_vars=['first', 'last'])
```

	first	last	variable	value
0	John	Doe	height	5.5
1	Mary	Bo	height	6.0
2	John	Doe	weight	130
3	Mary	Bo	weight	150

数据转置

```
df.T # or dftranspose()
```



时序数据

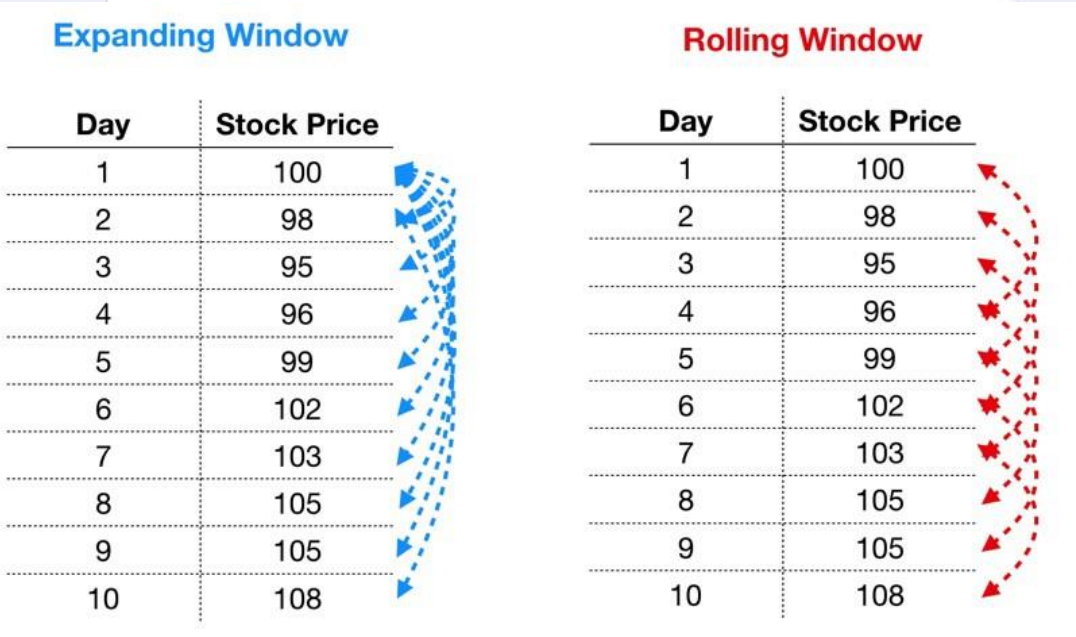
```
# 指定开始时间和频率，周期数
pd.Series(range(3), index=pd.date_range('2000', freq='D',
periods=3)
pd.Timestamp('2012-05-01')
pd.Timestamp(1513393355.5, unit='s') # 单位为秒
# 指定为北京时间
pd.Timestamp(1513393355, unit='s', tz='Asia/Shanghai')

time.week # 24 当年第几周
time.weekofyear # 24 同上
time.day # 9 日

s.astype('datetime64[ns]')
pd.to_datetime(['2005/11/23', '2010.12.31'])

pd.date_range(start='1/1/2018', periods=8)
pd.bdate_range(start='1/1/2021', end='1/08/2021')

two_business_days = 2 * pd.offsets.BDay()
ts.resample('5Min').mean() # 平均
```

文本处理

```
pd.Series(['a', 'b', 'c'], dtype="string")
pd.Series(['a', 'b', 'c'], dtype=pd.StringDtype())
# storage 默认为 python，指定 pyarrow
pd.StringDtype(storage="pyarrow")
pd.Series(['abc', None, 'def'],
dtype="string[pyarrow]")
s2.str.split('_')
s2.str.split('_', expand=True)
s.str.replace('$', '')

s.str.lower() # 转为小写
s.str.upper() # 转为大写
s.str.title() # 标题格式，每个单词大写
s.str.capitalize() # 首字母大写
s.str.swapcase() # 大小写互换
```

窗口计算

```
# 移动窗口
ser.rolling(window=5, win_type='boxcar').mean()
ser.rolling(window=5).mean()

ser.rolling(window=5, win_type='triang').mean()
ser.rolling(window=5,
win_type='gaussian').mean(std=0.1)

# 扩展窗口
df.expanding(min_periods=1).mean()
```



样式

```
df.style.highlight_null(null_color='blue')
# 最大值高亮，默认黄色
df.head().style.highlight_max()
# 最小值高亮
df.head().style.highlight_min()
# Q4 列的 60 到 100 高亮
df.style.highlight_between(left=60, right=100,
subset=['Q4'])
df.style.highlight_quantile(axis=1, q_left=0.8,
color="#fffd75")
df.style.text_gradient(axis=0)
df.style.background_gradient(subset=['Q1'],
cmap='BuGn')
df.style.bar(subset=['Q1'])

# 百分号，类似 29.57%
df.style.format("{:.2%}")
# 常用的格式
{'a': '¥{0:,.0f}', # 货币符号
 'b': '{:%Y-%m}', # 年月
 'c': '{:.2%}',  # 百分号
 'd': '{:,f}',  # 千分位
 'e': str.upper} # 大写
```

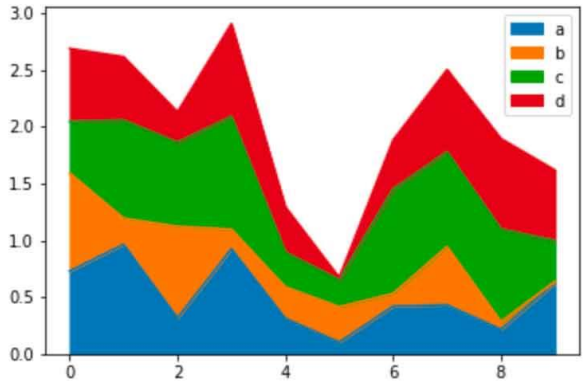
```
(df.head(10)
 .assign(avg=df.mean(axis=1, numeric_only=True)) # 增加平均值
 .assign(diff=lambda x: x.avg.diff()) # 和前位同学差值
 .style
 .bar(color='yellow', subset=['Q1'])
 .bar(subset=['avg'],
      width=90,
      align='mid',
      vmin=60, vmax=100,
      color='#5CADAD')
 .bar(subset=['diff'],
      color=['#ffe4e4','#bbf9ce'], # 上涨下降的颜色
      vmin=0, vmax=30, # 范围定以0为基准的上下30
      align='zero') # 0 值居中
)
```

	name	team	Q1	Q2	Q3	Q4	avg	diff
0	Liver	E	89	21	24	64	49.500000	nan
1	Arry	C	36	37	37	57	41.750000	-7.750000
2	Ack	A	57	60	18	84	54.750000	13.000000
3	Eorge	C	93	96	71	78	84.500000	29.750000
4	Oah	D	65	49	61	86	65.250000	-19.250000
5	Harlie	C	24	13	87	43	41.750000	-23.500000
6	Acob	B	61	95	94	8	64.500000	22.750000
7	Lfie	A	9	10	99	37	38.750000	-25.750000
8	Reddie	D	64	93	57	72	71.500000	32.750000
9	Oscar	A	77	9	26	67	44.750000	-26.750000

可视化

```
df.plot.line() # 折线的全写方式
df.plot.bar() # 柱状图
df.plot.barh() # 横向柱状图（条形图）
df.plot.hist() # 直方图
df.plot.box() # 箱形图
df.plot.kde() # 核密度估计图
df.plot.density() # 同 df.plot.kde()
df.plot.area() # 面积图
df.plot.pie() # 饼图
df.plot.scatter() # 散点图
df.plot.hexbin() # 六边形箱体图，或简称六边形图

import pandas as pd
import pandas_bokeh
pandas_bokeh.output_notebook() # notebook 展示
pd.set_option('plotting.backend', 'pandas_bokeh')
```

```
In [2]:
df = pd.DataFrame(np.random.rand(10, 4),
                  columns=['a', 'b', 'c', 'd'])

In [3]:
df.plot.area()

Out[3]: <matplotlib.axes._subplots.AxesSubplot at 0x7fd0c26545e0>
```



PART 5

可视化

Python 数据可视化

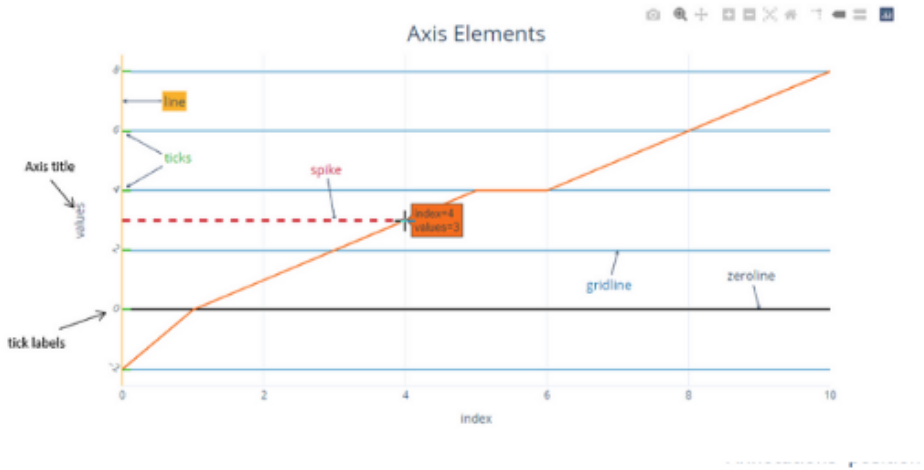
- matplotlib
- pyecharts (*)
- seaborn (*)
- plotly (*)
- bokeh (*)
- bqplot
- voila
- folium

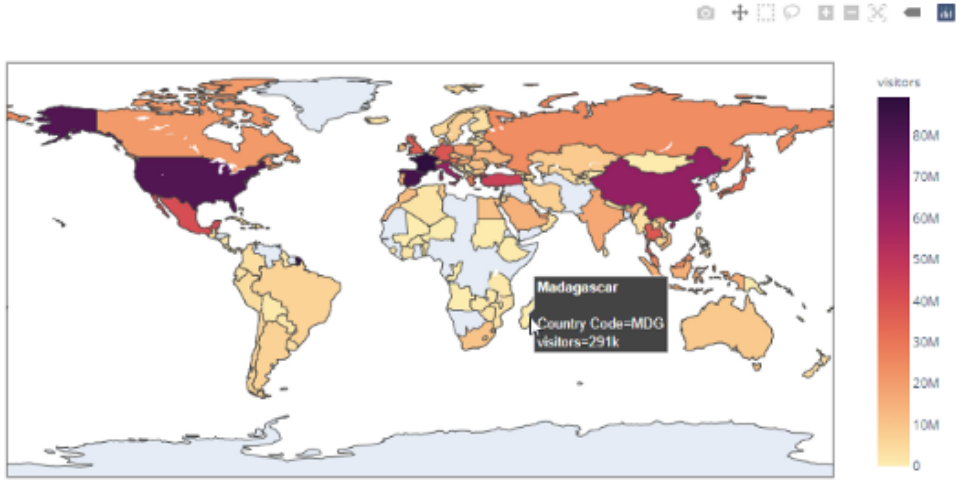
* 推荐学习

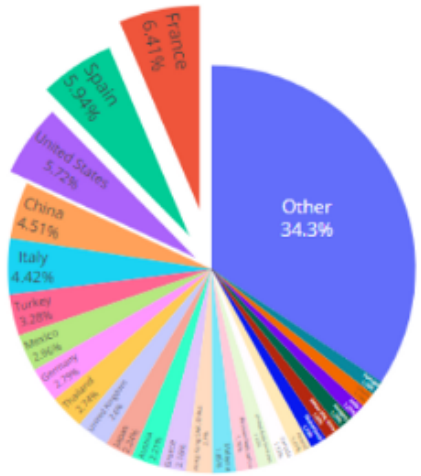
Plotly

https://plotly.com/python/plotly-express/

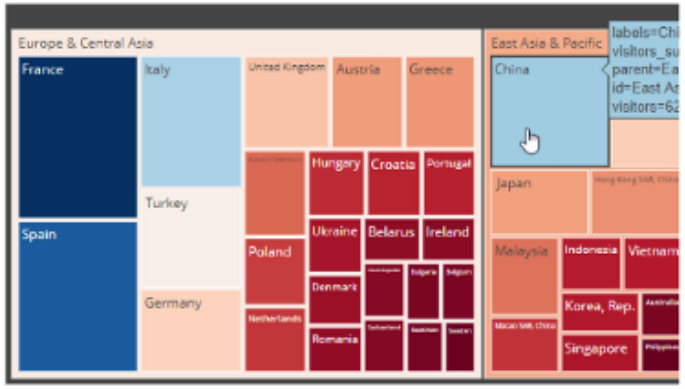
px.chart_type(df, parameters)

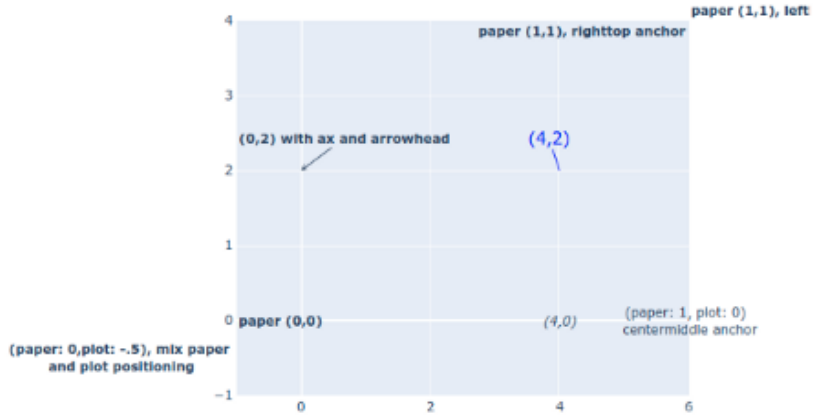


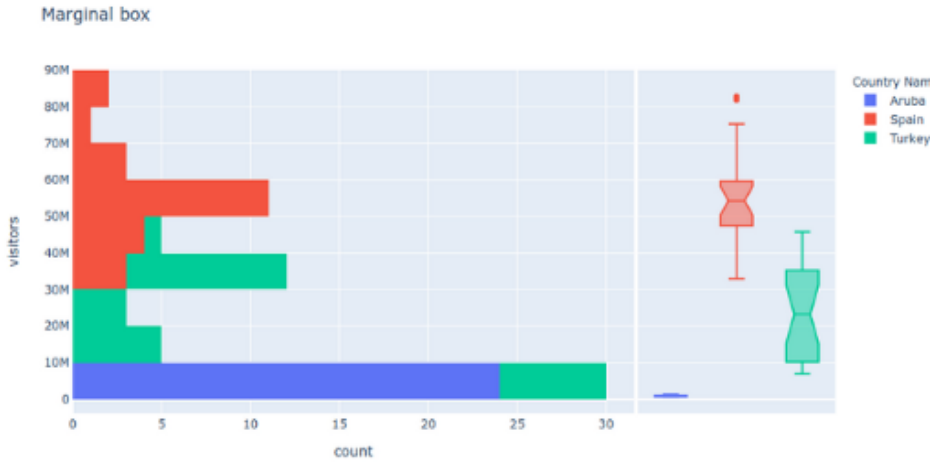




2018 visitors (top 50 countries)







PART 6

实践

案例：圣诞节的星期分布

利用Pandas 分析近100年圣诞节的星期分布，目的是知道圣诞节都在星期几，哪天多些。

由于本需求没有给定源数据，需要我们自己生成，整体思路如下：

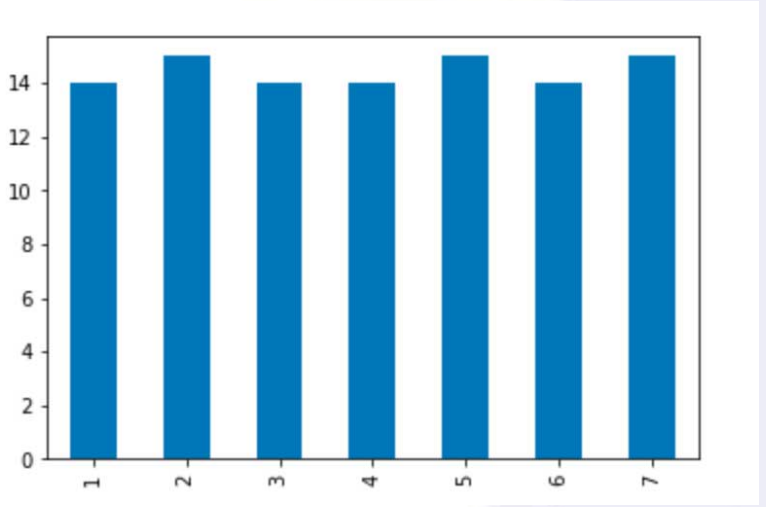
- 用 pd.date_range 生成 100 年日期数据
- 筛选出12月25日的所有日期
- 将日期转换为星期几
- 统计重复值的数量
- 绘图

代码实现

```
import pandas as pd


(
    # 生成100年时间序列
    pd.Series(pd.date_range('1920', '2021'))
    # 筛选 12月25日 的所有日期
    .loc[lambda s: (s.dt.month==12) & (s.dt.day==25)]
    .dt.day_of_week # 转为星期数
    .add(1) # 由于0代表周一，对序列加1，符合日常认知
    .value_counts() # 重复值计数
    .sort_values() # 排序，星期从1-7
    .plot
    .bar() # 绘制柱状图
)
```

[121]:
```
(
    # 生成100年时间序列
    pd.Series(pd.date_range('1920', '2021'))
    # 筛选 12月25日 的所有日期
    .loc[lambda s: (s.dt.month==12) & (s.dt.day==25)]
    .dt.day_of_week # 转为星期数
    .add(1) # 由于0代表周一，对序列加1，符合日常认知
    .value_counts() # 重复值计数
    .sort_values() # 排序，星期从1-7
#     .plot
#     .bar() # 绘制柱状图
)
```

[121]:
```
6    14
1    14
4    14
3    14
7    15
2    15
5    15
dtype: int64
```



结论：通过图形我们可以观察到，圣诞节在星期上的分布基本上比较均匀，与我们日常感知的在周五有很大出入，可能对于周五将休假的气氛与圣诞节的气氛对我们印象深刻。

链式方法

使用链式方法有如下优点：

- 代码简洁，逻辑清晰；
- 不破坏原始数据；
- 方便注释、调试；
- 可读性强；
- 可维护性强；
- 方便封装。

案例

统计分组后指定值的个数

自动填充 Excel 文本时间值

筛选查询每月1日的数据

标记一行是否包含一些内容

更多案例：

https://www.gairuo.com/p/pandas-case

相关资源

- Pandas 教程
- Pandas 速查手册
- Pandas 实战案例集
- Pandas 函数详细介绍

感谢您的观看