南京大学
自然语言处理研究组
Natural Language Processing Group

南京大學
NANJING UNIVERSITY

数据分析和可视化

黄书剑

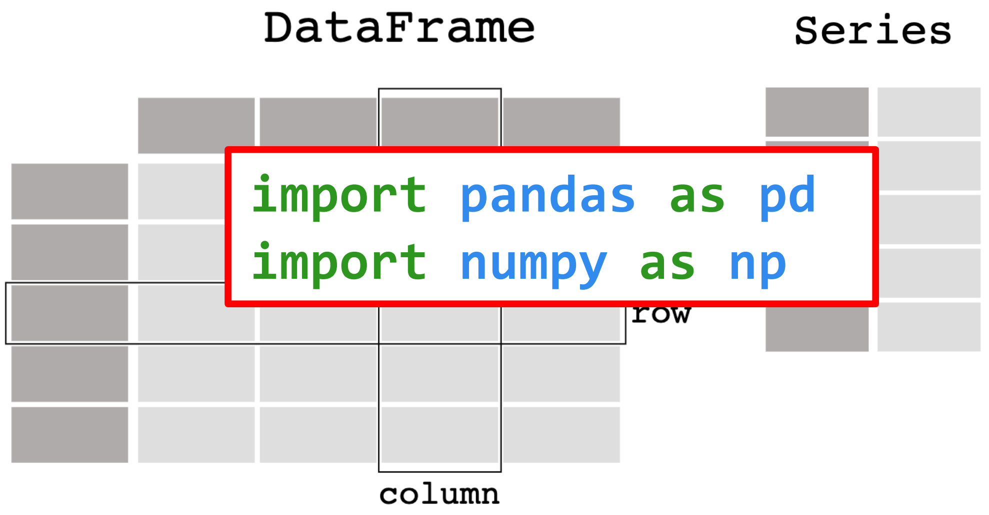
SciPy生态系统

- **SciPy (pronounced “Sigh Pie”) is a Python-based ecosystem of open-source software for mathematics, science, and engineering.**

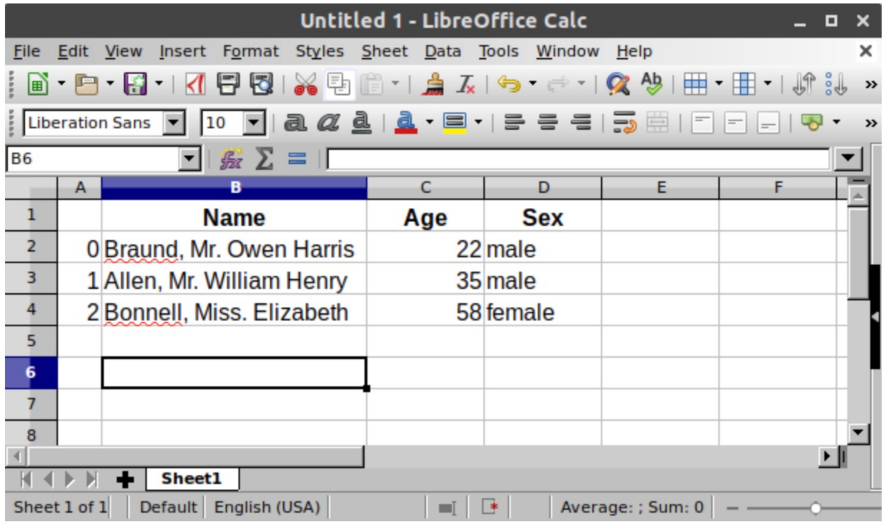
	NumPy Base N-dimensional array package		SciPy library Fundamental library for scientific computing		Matplotlib Comprehensive 2-D plotting
IP[y]: IPython	IPython Enhanced interactive console		SymPy Symbolic mathematics		pandas Data structures & analysis

Pandas

- pandas is a Python package providing fast, flexible, and expressive **data structures** designed to make working with “relational” or “labeled” data both easy and intuitive.

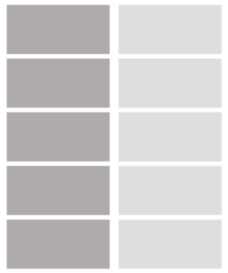


```
import pandas as pd
import numpy as np
```


Untitled 1 - LibreOffice Calc
File Edit View Insert Format Styles Sheet Data Tools Window Help
Liberation Sans
10
B6
Sheet1
Sheet 1 of 1
Default
English (USA)
Average: ; Sum: 0

	A	B	C	D	E	F
1		Name	Age	Sex		
2	0	Braund, Mr. Owen Harris	22	male		
3	1	Allen, Mr. William Henry	35	male		
4	2	Bonnell, Miss. Elizabeth	58	female		
5						
6						
7						
8						

Series



PANDAS-SERIES

Series

- **Series is a one-dimensional labeled array capable of holding any data type (integers, strings, floating point numbers, Python objects, etc.).**

3000
4500
8000

```
In [3]: salaries = [3000, 4500, 8000]
   ...: names = ['Mayue', 'Lilin', 'Wuyun']
   ...: dict(zip(names, salaries))
Out[3]: {'Mayue': 3000, 'Lilin': 4500, 'Wuyun': 8000}
```

label	Value
'Mayue'	3000
'Lilin'	4500
'Wuyun'	8000

```
In [5]: pd.Series(sd1)
Out[5]:
Mayue    3000
Lilin    4500
Wuyun    8000
dtype: int64
```

https://pandas.pydata.org/pandas-docs/stable/user_guide/dsintro.html#series

创建Series对象

- labeled array：记录 array values以及对应的label/index
- From ndarry

```
In [6]: s1 = pd.Series([1.0, 2.0, 3.0], index=["a", "b", "c"])
In [7]: s3 = pd.Series(np.random.rand(5), index = list("abcde"))
```

- From dict

```
In [8]: pd.Series({"b": 1, "a": 0, "c": 2})
In [9]: pd.Series(dict(zip(list("abcde"), np.arange(5))))
```

- From scalar value

```
In [10]: pd.Series(10, index = list("abc"))
```

```
Out[10]:
a     10
b     10
c     10
dtype: int64
```

named Series

- **可以给series命名，以表示特定的数值关系**
  - name, rename() 等

```
In [12]: pd.Series([1.0, 2.0, 3.0],
index=["x", "y", "z"], name =
"values")
Out[12]:
x    1.0
y    2.0
z    3.0
Name: values, dtype: float64
```

```
In [13]: s1.rename("r-values")

Out[13]:
a    1.0
b    2.0
c    3.0
Name: r-values, dtype: float64
```

Series的操作

- **ndarray操作：series对象可以像np array一样参与运算和使用**
  - 索引、切片
  - 向量化运算
  - to_numpy()

```
In [14]: s =
pd.Series(dict(zip(list("abcdef"),
np.random.randn(6))))

In [15]: s[0]
Out[15]: 0.1720155849962188
In [16]: s[1:3]
Out[16]:
b   -1.248262
c    0.063130
dtype: float64
```

```
In [17]: s[s > s.median()]
Out[17]:
a    0.172016
e    0.679836
f    0.360209
dtype: float64
```

```
In [19]: s[1:3].to_numpy()
Out[19]: array([-1.2482615 ,  0.06313031])
```

运算中保持label和数据的对应关系

Series的操作

- **dict操作：series对象可以像dict一样参与运算和使用**
  - 按label/index访问元素

```
In [20]: s
Out[20]:
a    0.172016
b   -1.248262
c    0.063130
d    0.075025
e    0.679836
f    0.360209
dtype: float64
```

```
In [21]: s["a"] = 0

In [22]: s["d"]
Out[22]: 0.07502540004838684

In [23]: s.get("g", np.nan)
Out[23]: nan
```

Series的index alignment

- **index/label将作为多个series合并运算的依据**

```
In [24]: s1 = pd.Series([1.0, 2.0, 3.0], index=["a", "b", "c"])
    ...: s2 = pd.Series([4.0, 5.0, 6.0], index=["a", "b", "c"])
    ...: s1 + s2
Out[24]:
a     5.0
b     7.0
c     9.0
dtype: float64
```

```
s1:
a     1.0
b     2.0
c     3.0
dtype: float64
```

```
s2:
a     4.0
b     5.0
c     6.0
dtype: float64
```

Series的index alignment

- **index/label将作为多个series合并运算的依据**
  - alignment与label的值有关，与顺序无关

```
In [25]: s3 = pd.Series([4.0, 5.0, 6.0], index=["b", "c", "a"])
    ...: s1 + s3
Out[25]:
a     7.0
b     6.0
c     8.0
dtype: float64
```

```
s1:
a     1.0
b     2.0
c     3.0
dtype: float64
```

```
s3:
b     4.0
c     5.0
a     6.0
dtype: float64
```

Series的index alignment

- **index/label将作为多个series合并运算的依据**
  - 运算中默认将不能对应的index值视为NaN

```
In [26]: s4 = pd.Series([4.0, 5.0, 6.0], index=["b", "c", "d"])
    ...: s1 + s4
Out[26]:
a     NaN
b     6.0
c     8.0
d     NaN
dtype: float64
```

```
s1:
a     1.0
b     2.0
c     3.0
dtype: float64
```

```
s4:
b     4.0
c     5.0
d     6.0
dtype: float64
```

Series示例

- **为若干员工发放工资，给指定员工发放奖金，计算工资总额**

```
In [31]: salaries = [3000, 4500, 8000]
    ...: names = ['Mayue', 'Lilin', 'Wuyun']
    ...: salaries_list = pd.Series(salaries, index = names)

In [32]: bonus = pd.Series({'Mayue': 500, 'Wuyun': 1000})
```

```
salaries_list + bonus
```

```
salaries
Mayue     3000
Lilin     4500
Wuyun     8000
dtype: int64
```

```
bonus:
Mayue      500
Wuyun     1000
dtype: int64
```

```
Lilin       NaN
Mayue    3500.0
Wuyun    9000.0
dtype: float64
```

Series示例

- 为若干员工发放工资，给指定员工发放奖金，计算工资总额

```
salaries              bonus:
Mayue     3000        Mayue      500
Lilin     4500        Wuyun     1000
Wuyun     8000        dtype: int64
dtype: int64
```

```
salaries_list.add(bonus, fill_value = 0)
```

在函数中指定缺失值的处理策略

Series示例

- 为若干员工发放工资，给指定员工发放奖金，计算工资总额

```
salaries            bonus:
Mayue     3000      Mayue      500
Lilin     4500      Wuyun     1000
Wuyun     8000      dtype: int64
dtype: int64
```

```
In [36]: bonus = bonus.reindex(names)

In [37]: bonus = bonus.fillna(0)
```

预先处理缺失值

```
salaries_list + bonus
```

```
Mayue       500.0     Mayue       500.0
Lilin         NaN     Lilin         0.0
Wuyun      1000.0     Wuyun      1000.0
dtype: float64        dtype: float64
```

Series示例

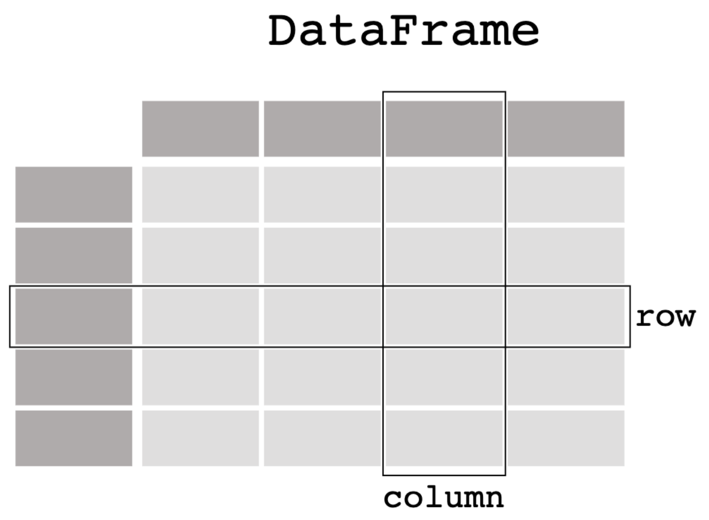
- 为若干员工发放工资，给指定员工发放奖金，计算工资总额

```
salaries          bonus:
Mayue     3000    Mayue      500
Lilin     4500    Wuyun     1000
Wuyun     8000    dtype: int64
dtype: int64
```

```
In [49]: salaries_list[bonus.index] += bonus
```

仅选择部分元素进行操作

PANDAS-DATAFRAME



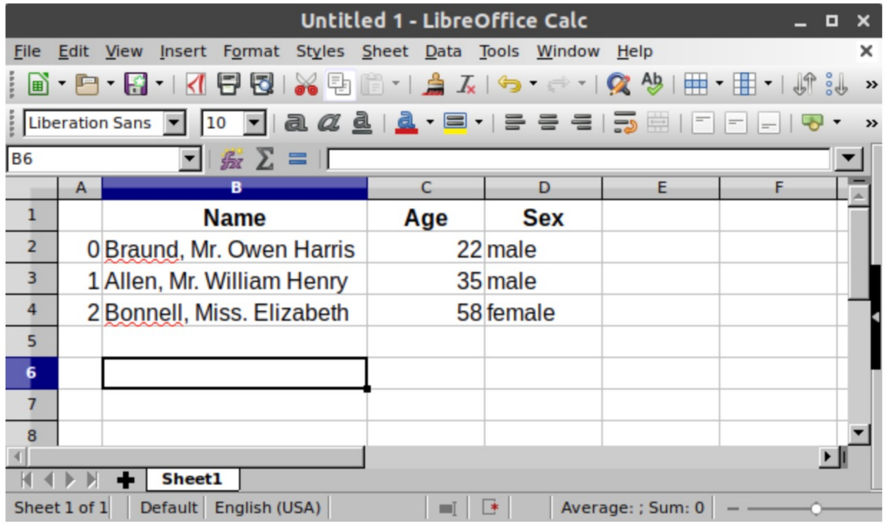
DataFrame

- **DataFrame is a 2-dimensional labeled data structure with columns of potentially different types. You can think of it like a spreadsheet or SQL table, or a dict of Series objects.**

columns / attribute labels

name	salary	bonus
'Mayue'	3000	500
'Lilin'	4500	
'Wuyun'	8000	1000

index / item label


Untitled 1 - LibreOffice Calc
File Edit View Insert Format Styles Sheet Data Tools Window Help
Liberation Sans
10
B6
Sheet1
Sheet 1 of 1
Default
English (USA)
Average: ; Sum: 0

	A	B	C	D	E	F
1		Name	Age	Sex		
2	0	Braund, Mr. Owen Harris	22	male		
3	1	Allen, Mr. William Henry	35	male		
4	2	Bonnell, Miss. Elizabeth	58	female		
5						
6						
7						
8						

创建DataFrame对象

- **from series**
  - 构建一个单列的DataFrame
  - Series的index会成为表格的index
  - 默认的列名（column name）为数值索引

```
In [53]: s = pd.Series([4, 5, 6],
index = list("bca"))
In [54]: s
Out[54]:
b     4
c     5
a     6
dtype: int64
```

```
In [55]: pd.DataFrame(s)
Out[55]:
   0
b  4
c  5
a  6
```

创建DataFrame对象

- **from series**
  - 构建一个单列的DataFrame
  - Series的index会成为表格的index
  - Series的name会自动转换为表格的列名（column name)

```
In [58]: sr = s.rename("values")

In [59]: pd.DataFrame(sr)
Out[59]:
   values
b       4
c       5
a       6
```

创建DataFrame对象

- **from dict of series**
  - series的index将被自动对齐一致，作为index
  - dict的index将作为column name

```
In [53]: s = pd.Series([4, 5, 6], index = list("bca"))
In [63]: s2 = pd.Series([1, 2, 3], index = list("abc"))

In [65]: pd.DataFrame({"one":s, "two":s2})
```

```
                                        Out[65]:
                                           one  two
b    4           a    1                 a    6    1
c    5           b    2                 b    4    2
a    6           c    3                 c    5    3
dtype: int64     dtype: int64
```

创建DataFrame对象

- **from np.array**

```
In [66]: data = np.array([(1, 2.0, "Hello"), (2, 3.0, "World")])
array([['1', '2.0', 'Hello'],
       ['2', '3.0', 'World']], dtype='<U32')
In [67]: pd.DataFrame(data)
Out[67]:
   0   1      2
0  1  2.0  Hello
1  2  3.0  World
```

默认的index和column name都为数值索引

```
In [68]: pd.DataFrame(data, index = ["first","second"], columns =
["id","value", "string"])
Out[68]:
       id value string
first   1   2.0  Hello
second  2   3.0  World
```

指定index和columns

创建DataFrame对象

- **from a list of dicts**

```
In [69]: data2 = [{"a": 1, "b": 2}, {"a": 5, "b": 10, "c": 20}]
In [70]: pd.DataFrame(data2)
Out[70]:
   a   b     c
0  1   2   NaN
1  5  10  20.0

In [71]: pd.DataFrame(data2, columns = ["a", "b", "z"])
Out[71]:
   a   b   z
0  1   2 NaN
1  5  10 NaN
```

用指定的column创建表格

创建DataFrame对象

- **从文件中读取数据**

```
In [77]: anscombe = pd.read_excel("seaborn-data/anscombe.xlsx")

In [78]: anscombe = pd.read_csv("seaborn-data/anscombe.csv")
```

- **部分包中带有一些示例数据**

```
import seaborn as sns
anscombe = sns.load_dataset("anscombe")
```

创建DataFrame对象

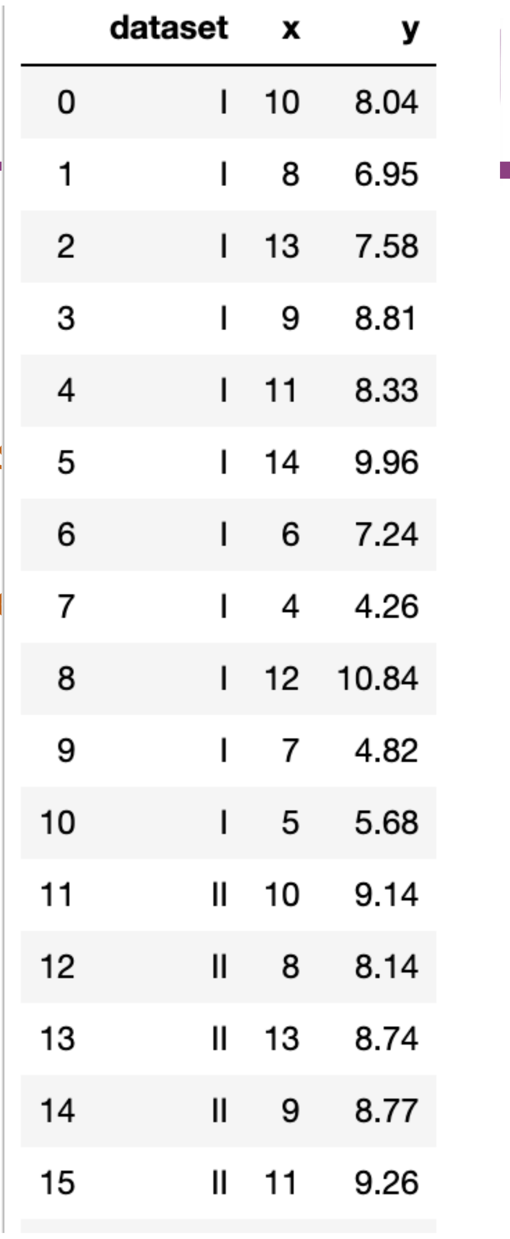
- **从文件中读取数据**

```
In [77]: anscombe = pd.read_excel("seaborn-data/ans
```

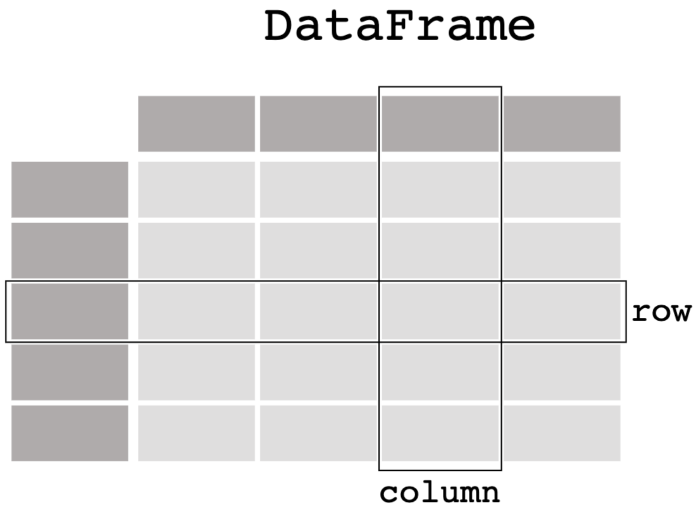
```
In [78]: anscombe = pd.read_csv("seaborn-data/ansc
```

- **部分包中带有一些示例数据**

```
import seaborn as sns
anscombe = sns.load_dataset("anscombe")
```



	dataset	x	y
0	I	10	8.04
1	I	8	6.95
2	I	13	7.58
3	I	9	8.81
4	I	11	8.33
5	I	14	9.96
6	I	6	7.24
7	I	4	4.26
8	I	12	10.84
9	I	7	4.82
10	I	5	5.68
11	II	10	9.14
12	II	8	8.14
13	II	13	8.74
14	II	9	8.77
15	II	11	9.26



PANDAS-DATAFRAME的使用

DataFrame的属性

- **df.info() 查看表格的形状、数值、类型、内存等信息**

```
In [80]:
anscombe.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 44 entries, 0 to 43
Data columns (total 3 columns):
dataset    44 non-null object
x          44 non-null float64
y          44 non-null float64
dtypes: float64(2), object(1)
memory usage: 1.2+ KB
```

DataFrame的属性

- **对给定表格计数、统计等**
  - count, mean, std, min, max
  - describe()

```
In [81]: anscombe.describe()
Out[81]:
               x          y
count  44.000000  44.000000
mean    9.000000   7.500682
std     3.198837   1.958925
min     4.000000   3.100000
25%     7.000000   6.117500
50%     8.000000   7.520000
75%    11.000000   8.747500
max    19.000000  12.740000
```

Selection

- 通过切片、指定index和column等方式选择表格的部分内容

```
In [83]: df = pd.DataFrame(np.arange(12).reshape(3, 4),
columns  = ["a", "b", "c", "d"])

In [84]: df["b"]

In [85]: df["e"] = df["a"] + df["b"]

In [86]: df["flag"] = df["e"] > 2
```

	a	b	c	d
0	0	1	2	3
1	4	5	6	7
2	8	9	10	11

```
0      1
1      5
2      9
Name: b, dtype: int64
```

	a	b	c	d	e
0	0	1	2	3	1
1	4	5	6	7	9
2	8	9	10	11	17

	a	b	c	d	e	flag
0	0	1	2	3	1	False
1	4	5	6	7	9	True
2	8	9	10	11	17	True

align on both columns and index

```
In [88]: df = pd.DataFrame(np.random.randint(1, 10, (8, 4)), columns=["A", "B", 
"C", "D"])
In [89]: df2 = pd.DataFrame(np.random.randn(7, 3), columns=["A", "B", "C"])
In [90]: df + df2
```

	A	B	C	D
0	1	2	2	7
1	9	1	7	4
2	8	9	4	1
3	1	3	1	8
4	8	3	2	7
5	2	6	9	2
6	6	5	4	1
7	8	9	3	1

	A	B	C
0	1.469132	2.476580	0.020756
1	0.171547	-0.046935	0.701937
2	0.074832	1.362343	0.641403
3	0.313266	0.298053	1.117556
4	-0.200598	1.329258	0.253670
5	1.002973	-0.699963	1.355623
6	-1.668525	0.507279	-0.950214

	A	B	C	D
0	2.469132	4.476580	2.020756	NaN
1	9.171547	0.953065	7.701937	NaN
2	8.074832	10.362343	4.641403	NaN
3	1.313266	3.298053	2.117556	NaN
4	7.799402	4.329258	2.253670	NaN
5	3.002973	5.300037	10.355623	NaN
6	4.331475	5.507279	3.049786	NaN
7	NaN	NaN	NaN	NaN

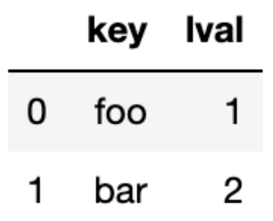
Merging

- 按照给定column将两个表合并为一个大表

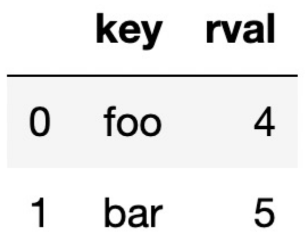
```
In [91]: left = pd.DataFrame({"key": ["foo", "bar"], "lval":
[1, 2]})

In [92]: right = pd.DataFrame({"key": ["foo", "bar"],
"rval": [4, 5]})

In [93]: pd.merge(left, right, on="key")
Out[93]:
   key  lval  rval
0  foo     1     4
1  bar     2     5
```



	key	lval
0	foo	1
1	bar	2



	key	rval
0	foo	4
1	bar	5

Grouping

```
In [94]: df = pd.DataFrame(
   ...:     {
   ...:     "A": ["foo", "bar", "foo", "bar", "foo", "bar", "foo", "foo"],
   ...:     "B": ["one", "one", "two", "three", "two", "two", "one", "three"],
   ...:     "C": np.random.randn(8),
   ...:     "D": np.random.randn(8),
   ...:     }
   ...:     )
```

```
In [95]: df.groupby("A").sum()
```

	A	B	C	D
0	foo	one	1.390122	-0.183875
1	bar	one	0.001490	0.822334
2	foo	two	-1.503483	0.076195
3	bar	three	0.194430	1.254074
4	foo	two	0.727498	-0.365359
5	bar	two	2.756550	0.850067
6	foo	one	-0.836635	-2.160404
7	foo	three	1.239052	0.609955

Grouping

- 按照给定类别分组并进行指定操作
  - split-apply-combine

```
In [95]: df.groupby("A").sum()
```

A	C	D
bar	2.952470	2.926475
foo	1.016554	-2.023488

	A	B	C	D
0	foo	one	1.390122	-0.183875
1	bar	one	0.001490	0.822334
2	foo	two	-1.503483	0.076195
3	bar	three	0.194430	1.254074
4	foo	two	0.727498	-0.365359
5	bar	two	2.756550	0.850067
6	foo	one	-0.836635	-2.160404
7	foo	three	1.239052	0.609955

Grouping

- 按照给定类别分组并进行指定操作
  - split-apply-combine

```
In [96]: df.groupby(["A", "B"]).sum()
```

A	B	C	D
bar	one	0.001490	0.822334
	three	0.194430	1.254074
	two	2.756550	0.850067
foo	one	0.553487	-2.344279
	three	1.239052	0.609955
	two	-0.775985	-0.289164

	A	B	C	D
0	foo	one	1.390122	-0.183875
1	bar	one	0.001490	0.822334
2	foo	two	-1.503483	0.076195
3	bar	three	0.194430	1.254074
4	foo	two	0.727498	-0.365359
5	bar	two	2.756550	0.850067
6	foo	one	-0.836635	-2.160404
7	foo	three	1.239052	0.609955

Pivot table

- 数据透视表（选择给定的行、列构成新的视图）

```
In [97]: pd.pivot_table(df, values =
"D", index="A",
columns=["B"])
```

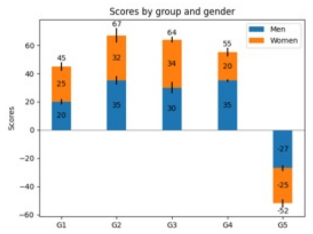
B	one	three	two
A			
bar	0.822334	1.254074	0.850067
foo	-1.172139	0.609955	-0.144582

	A	B	C	D
0	foo	one	1.390122	-0.183875
1	bar	one	0.001490	0.822334
2	foo	two	-1.503483	0.076195
3	bar	three	0.194430	1.254074
4	foo	two	0.727498	-0.365359
5	bar	two	2.756550	0.850067
6	foo	one	-0.836635	-2.160404
7	foo	three	1.239052	0.609955

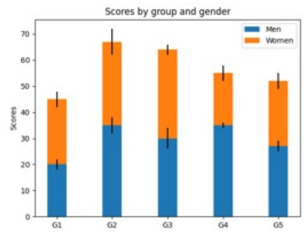
MATPLOTLIB和数据可视化

https://matplotlib.org/stable/gallery/index.html

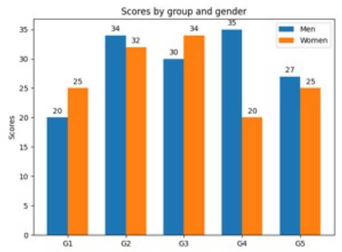
Lines, bars and markers



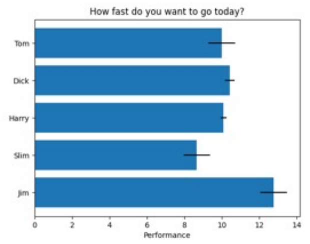
Bar Label Demo



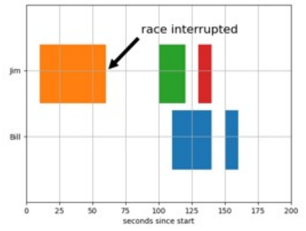
Stacked bar chart



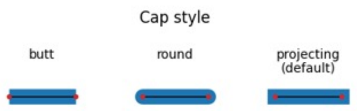
Grouped bar chart with labels



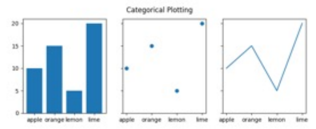
Horizontal bar chart



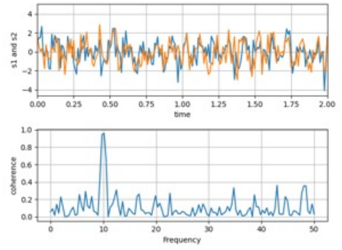
Broken Barh



CapStyle

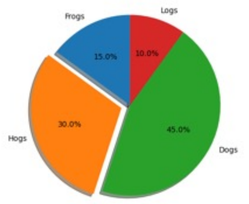


Plotting categorical variables

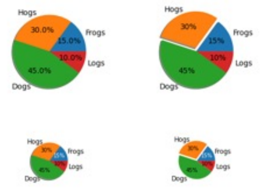


Plotting the coherence of two signals

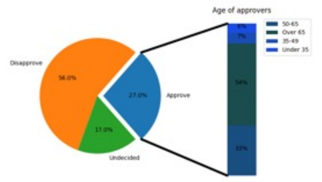
Pie and polar charts



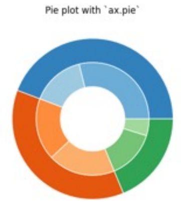
Basic pie chart



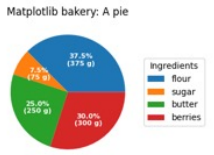
Pie Demo2



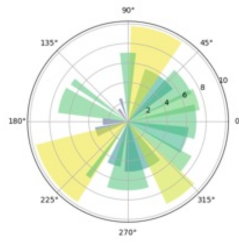
Bar of pie



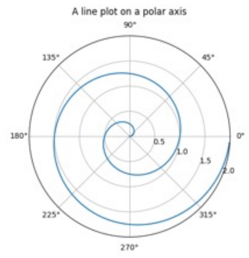
Nested pie charts



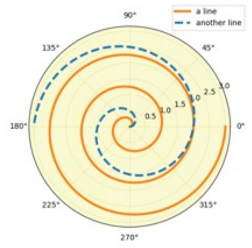
Labeling a pie and a donut



Bar chart on polar axis

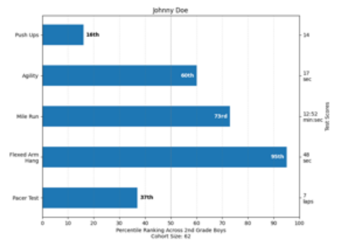


Polar plot

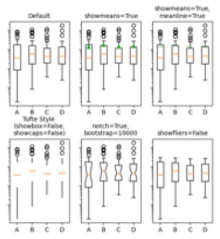


Polar Legend

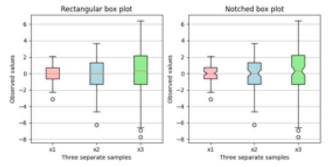
Statistics



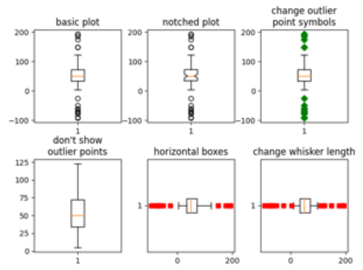
Percentiles as horizontal bar chart



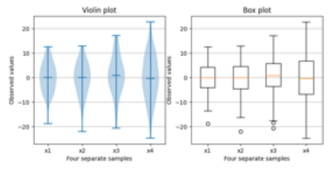
Artist customization in box plots



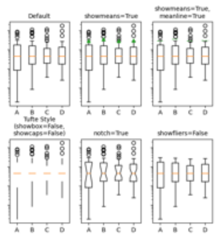
Box plots with custom fill colors



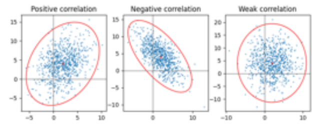
Boxplots



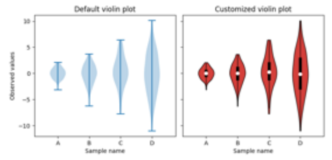
Box plot vs. violin plot comparison



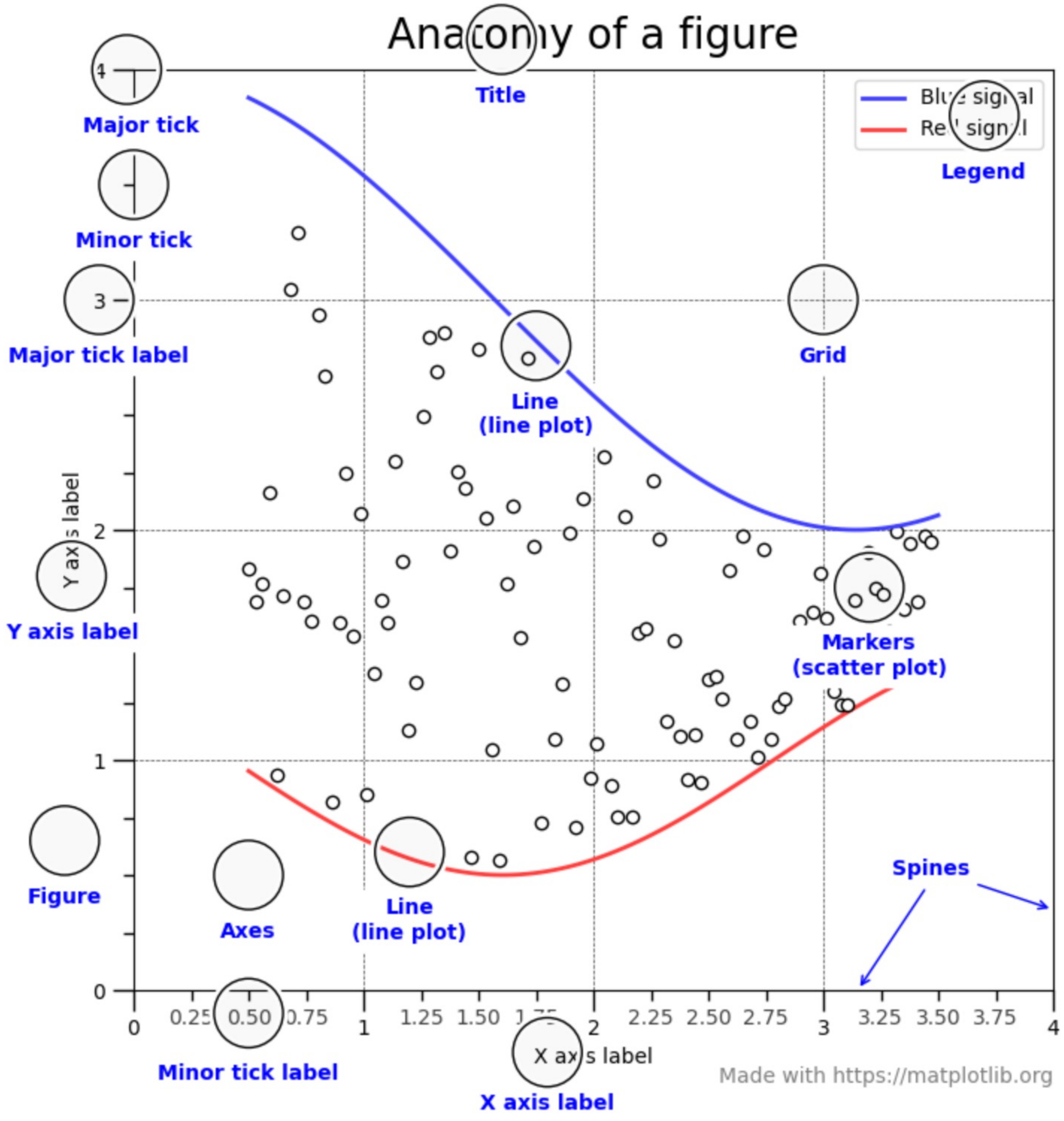
Boxplot drawer function



Plot a confidence ellipse of a two-dimensional dataset

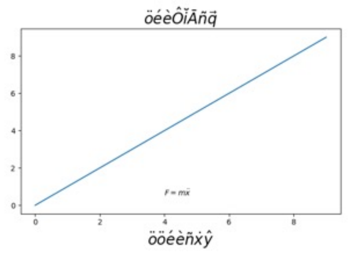


Violin plot customization

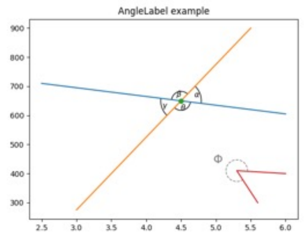


Anatomy of a figure
Title
Major tick
Minor tick
Major tick label
Blue signal
Red signal
Legend
Grid
Line
(line plot)
Y axis label
Markers
(scatter plot)
Figure
Axes
Line
(line plot)
Spines
Minor tick label
X axis label
Made with https://matplotlib.org
Y axis label
X axis label
0
1
2
3
4
0.25
0.50
0.75
1.25
1.50
1.75
2.25
2.50
2.75
3.25
3.50
3.75

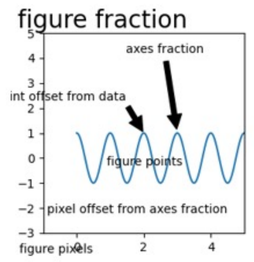
Text, labels and annotations



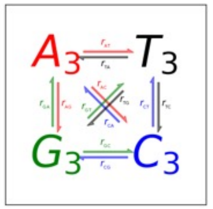
Using accented text in matplotlib



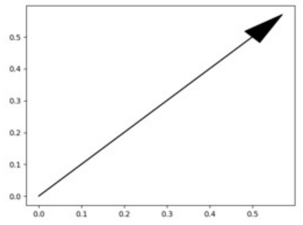
Scale invariant angle label



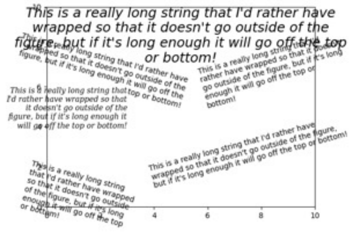
Annotating Plots



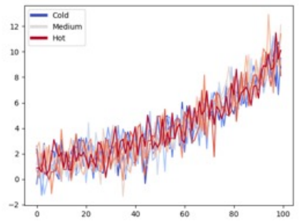
Arrow Demo



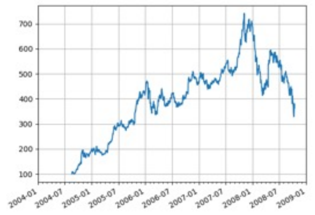
Arrow Simple Demo



Auto-wrapping text

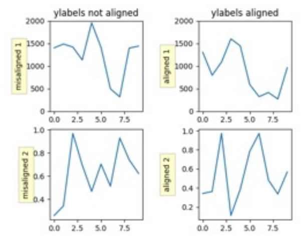


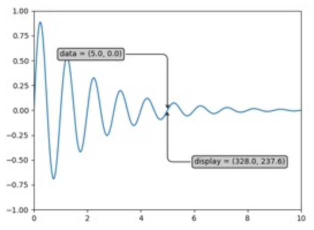
Composing Custom Legends

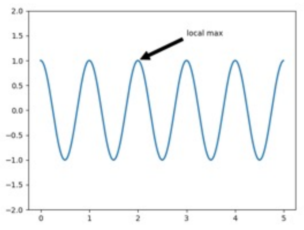


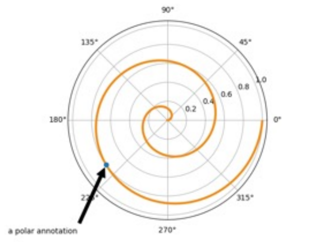
Date tick labels

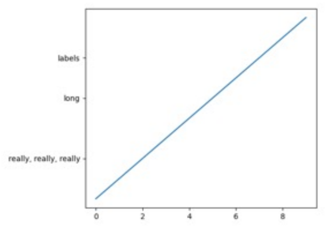
Pyplot

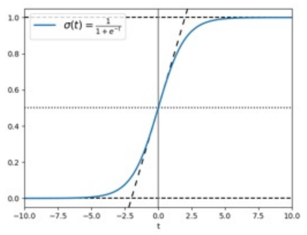

Align y-labels

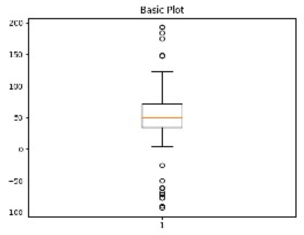

Annotate Transform

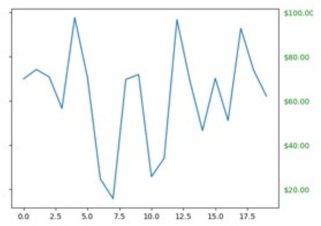

Annotating a plot


Annotation Polar

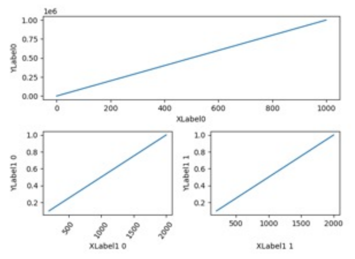

Auto Subplots Adjust


Infinite lines

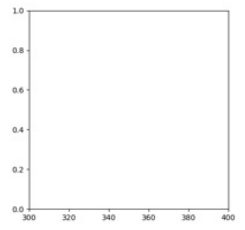

Boxplot Demo


Dollar Ticks

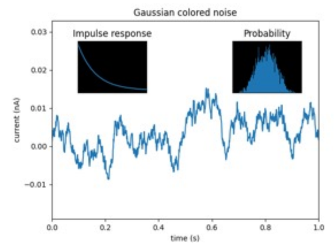
Subplots, axes and figures



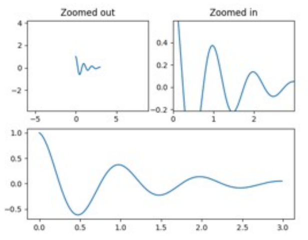
Aligning Labels



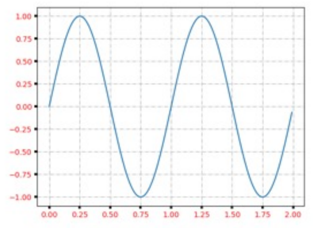
Axes box aspect



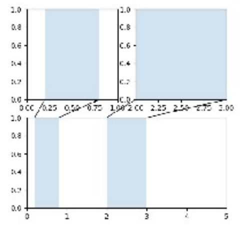
Axes Demo



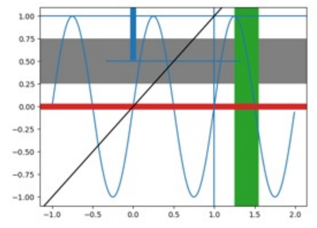
Controlling view limits using margins and sticky_edges



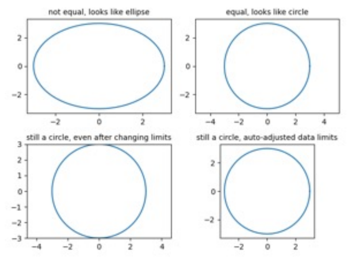
Axes Props



Axes Zoom Effect

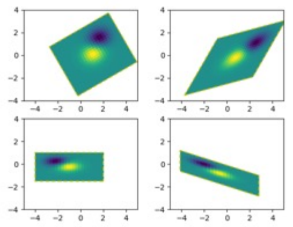


axhspan Demo

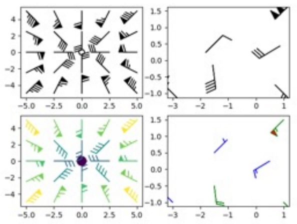


Equal axis aspect ratio

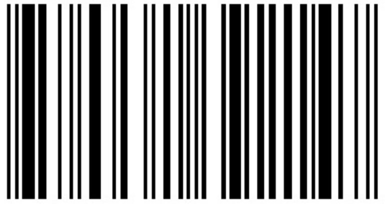
Images, contours and fields



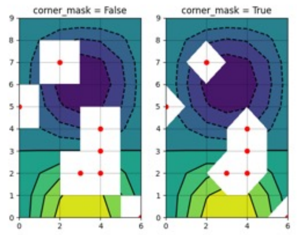
Affine transform of an image



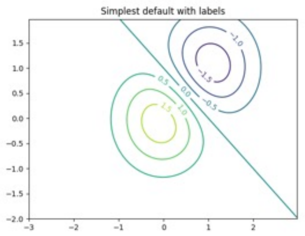
Wind Barbs



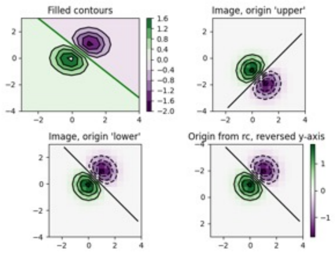
Barcode



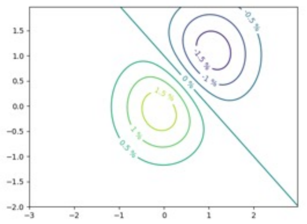
Contour Corner Mask



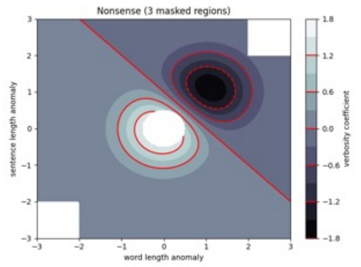
Contour Demo



Contour Image

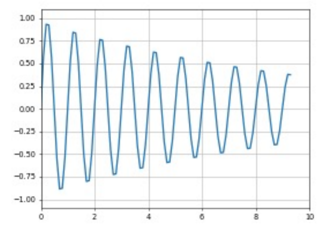


Contour Label Demo

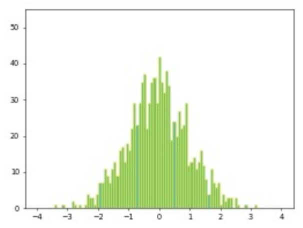


Contourf Demo

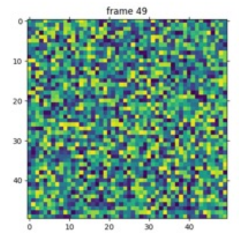
Animation



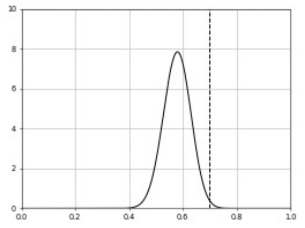
Decay



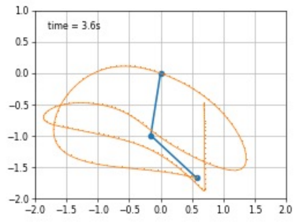
Animated histogram



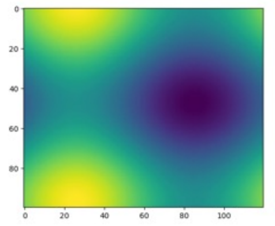
pyplot animation



The Bayes update

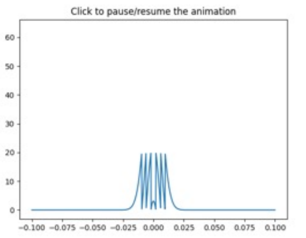


The double pendulum problem



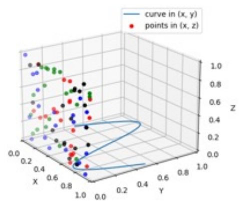
Animated image using a precomputed list of images

Frame grabbing

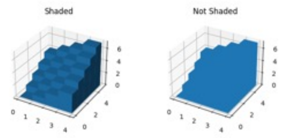


Pausing and Resuming an Animation

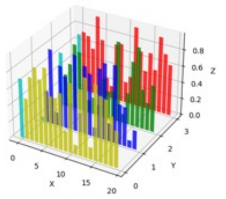
3D plotting



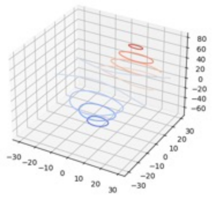
Plot 2D data on 3D plot



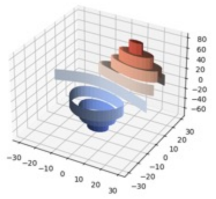
Demo of 3D bar charts



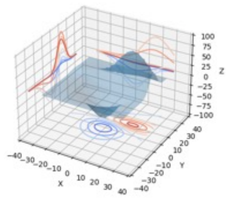
Create 2D bar graphs in different planes



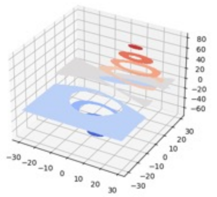
Demonstrates plotting contour (level) curves in 3D



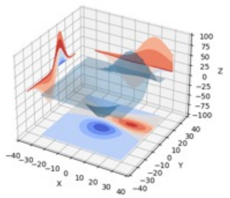
Demonstrates plotting contour (level) curves in 3D using the extend3d



Projecting contour profiles onto a graph



Filled contours

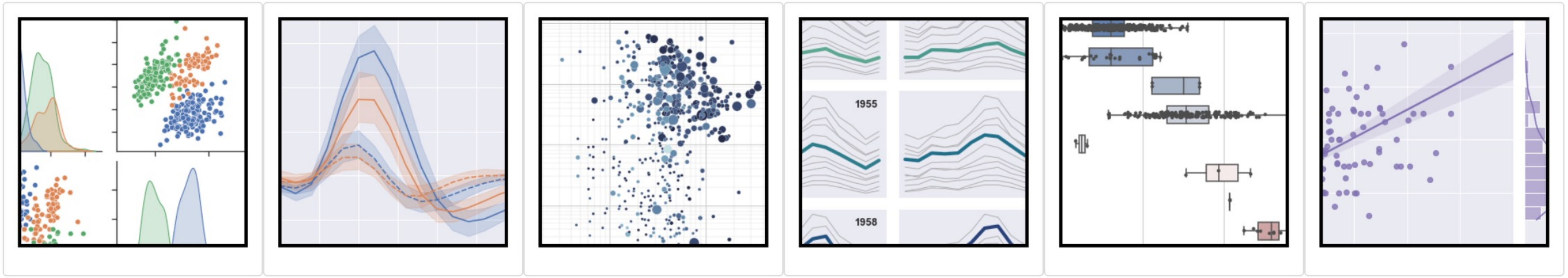


Projecting filled contour onto a graph

Seaborn

- https://seaborn.pydata.org/

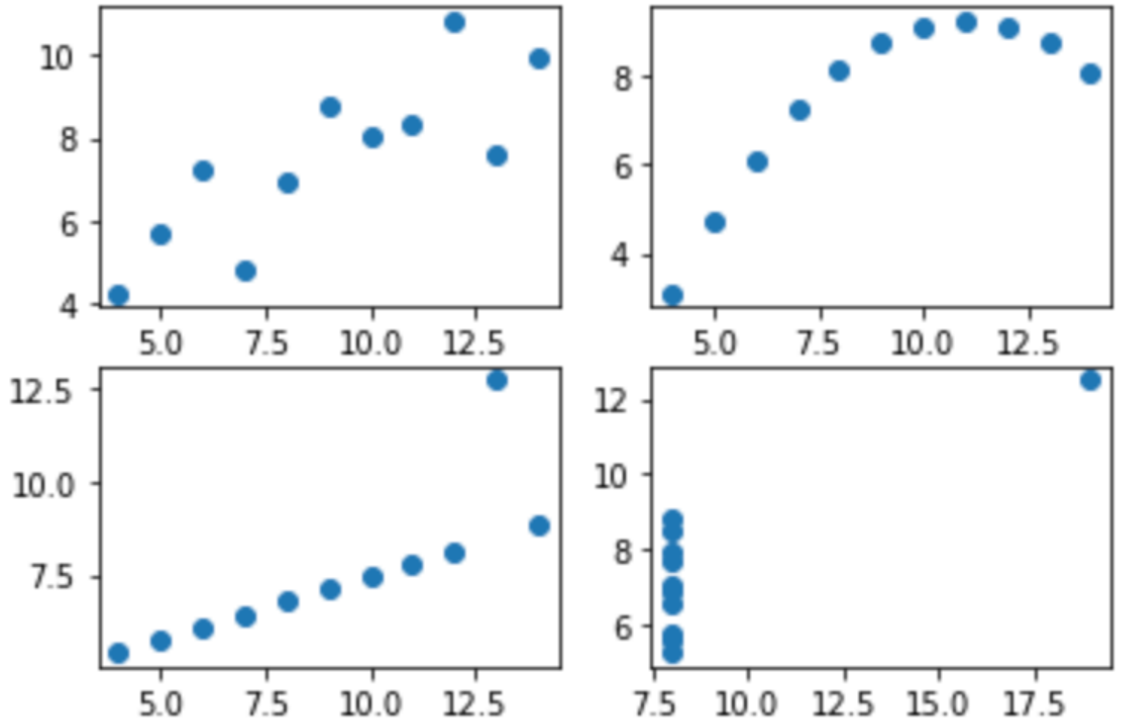
seaborn: statistical data visualization



数据分析实例

example 1: anscombe

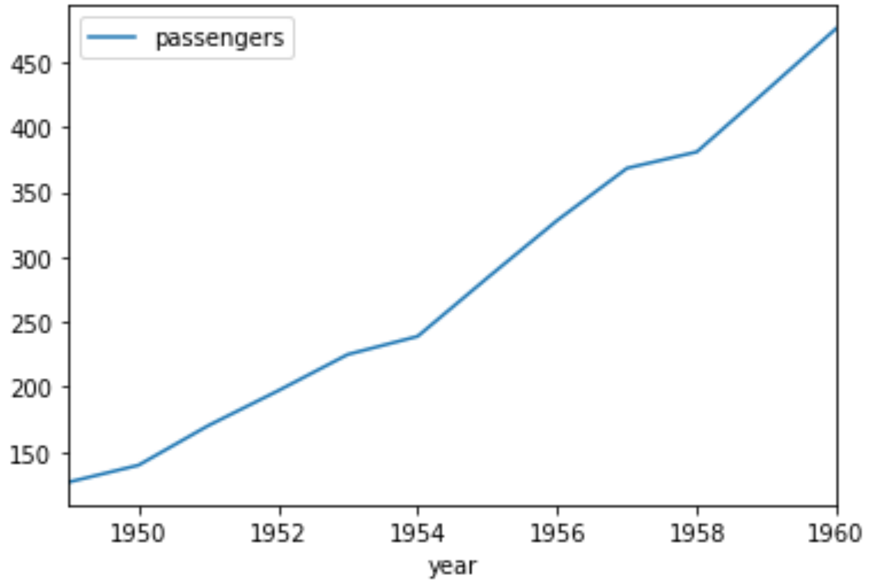
- 绘制散点图
- 多个子图

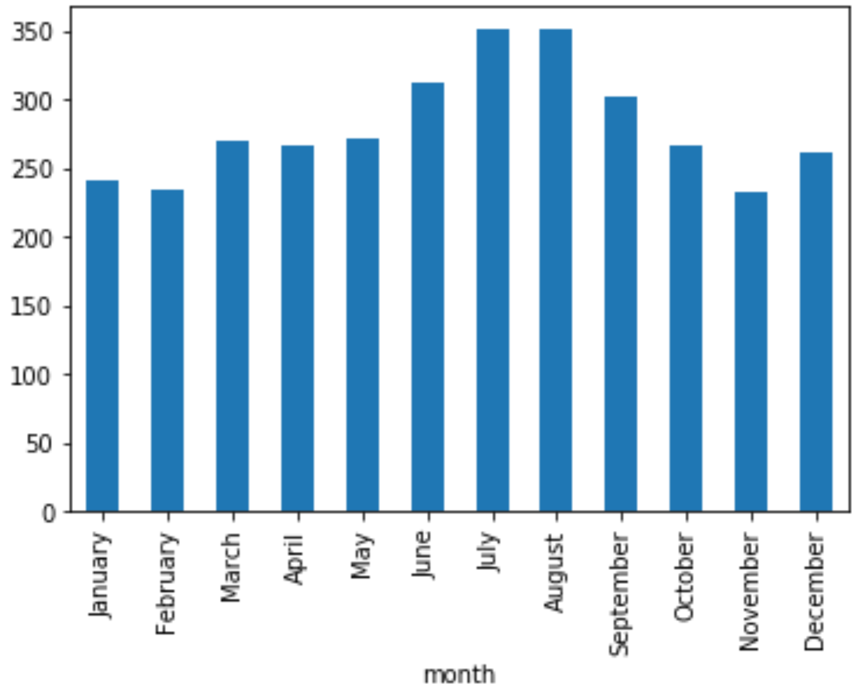


https://en.wikipedia.org/wiki/Anscombe%27s_quartet

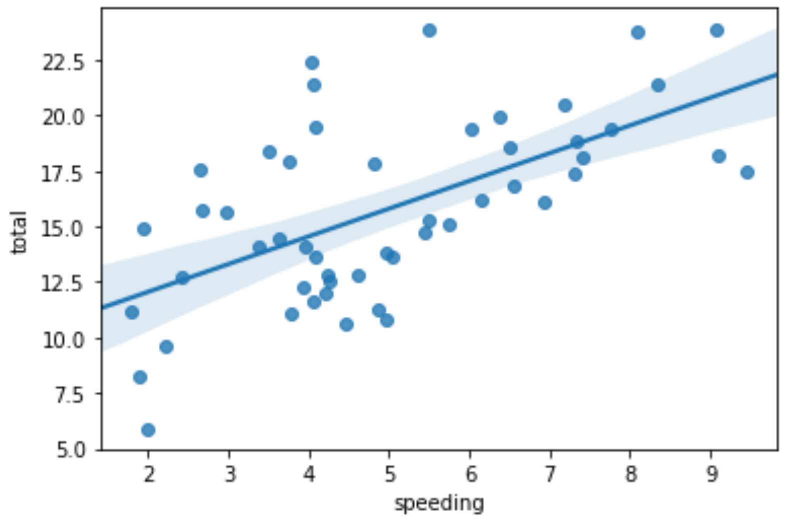
example2： flights

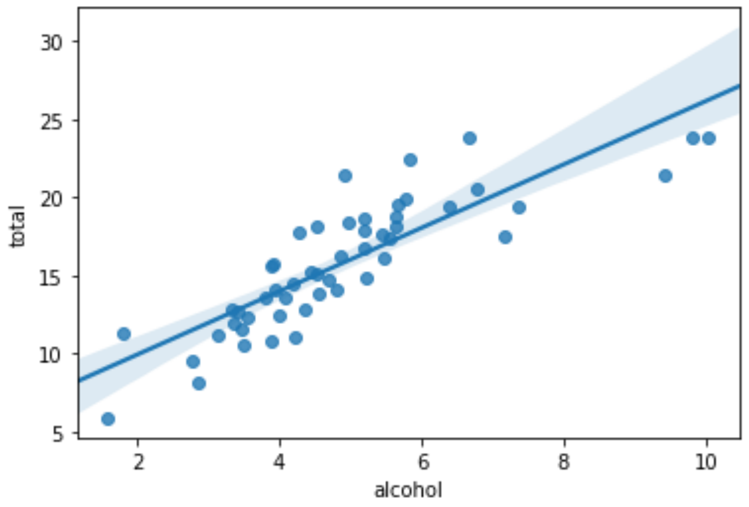
- **数据分组**



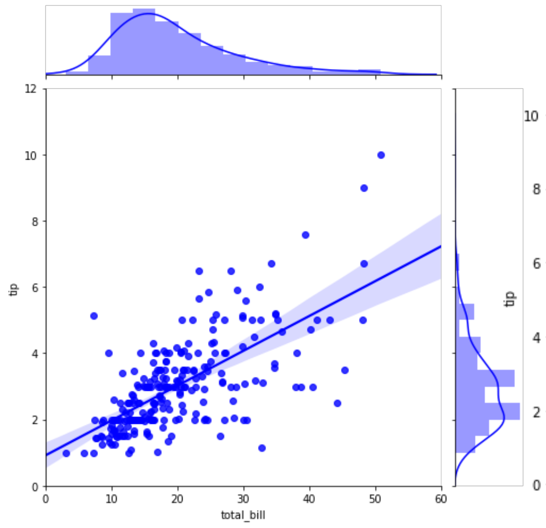


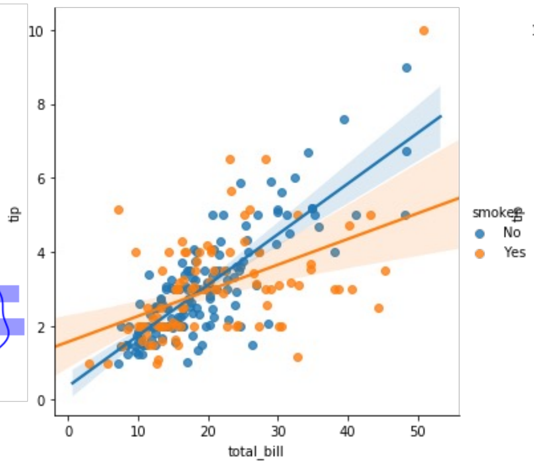
example3: car_crashes

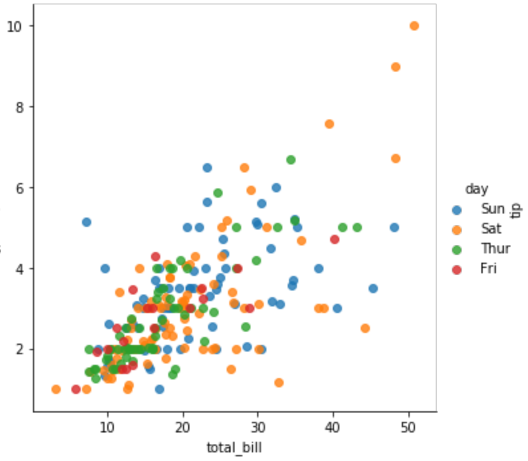


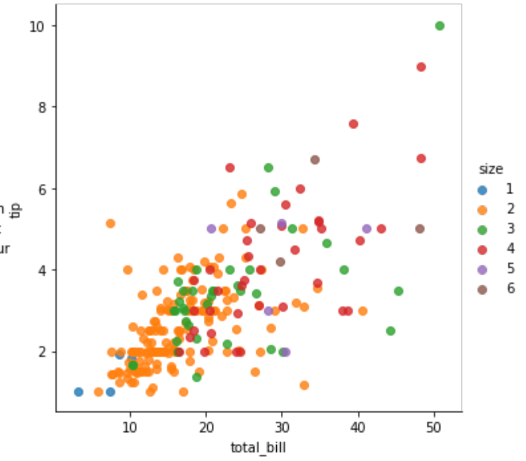


example 4: tips

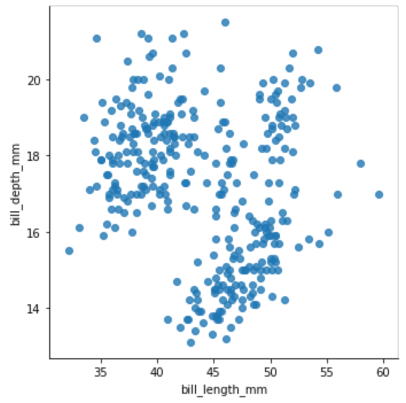


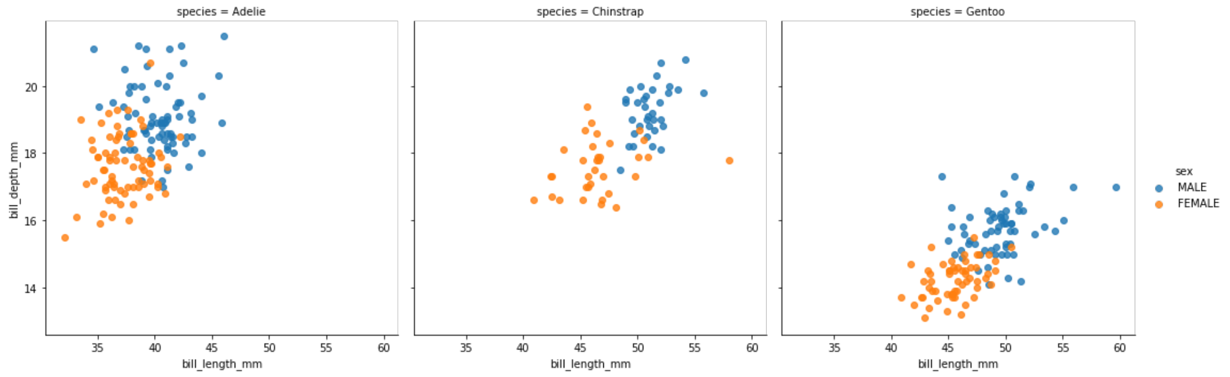


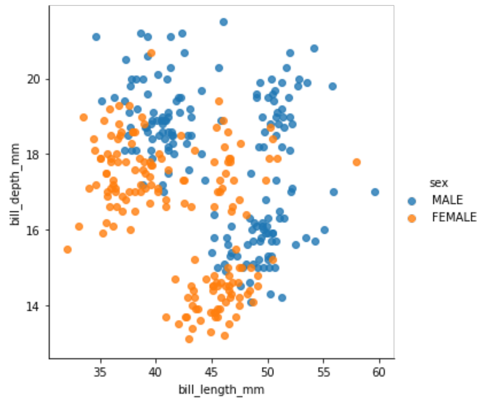


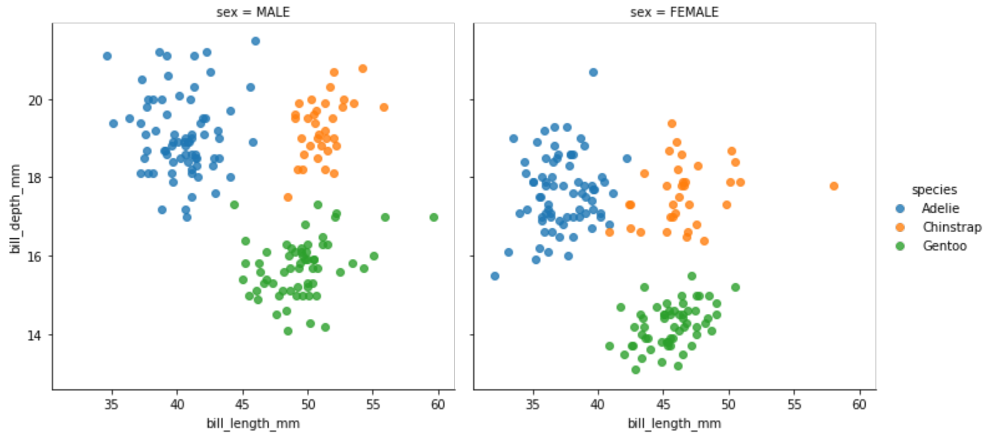


example 5: penguins









- **Quick intro to padas:**
  - https://pandas.pydata.org/pandas-docs/stable/user_guide/10min.html
- **Series and Dataframe**
  - https://pandas.pydata.org/pandas-docs/stable/user_guide/dsintro.html