第2讲：Python科学计算库

- NumPy
- Scipy
- Matplotlib
- Pandas
- Scikit-Learn

WANGBIANQIN

Python科学计算库

- Numpy：科学计算的基础库之一，数值计算的基础库，创建多维数组，以及多维数据的各种运算。
- Scipy： 建立在 Numpy 库之上，一个数值算法集合和特定领域的工具箱，包括信号处理、优化、统计等。
- Matplotlib：科学绘图库，其功能为数据的可视化，它依赖Numpy和Scipy两个库
- Pandas：建立在Numpy之上，用于处理和分析数据的库。
- Scikit-Learn：支持回归、分类、聚类等机器学习算法和工具库，依赖Numpy、Scipy，常与Matplotlib、Pandas协同工作。
- Tensorflow：Google的深度学习库，支持DNN，CNN，RNN等

Python科学计算库官网

- NumPy http://www.numpy.org/
- SciPy https://www.scipy.org/
- Matplotlib https://matplotlib.org/
- Pandas http://pandas.pydata.org/
- Scikit-learn https://scikit-learn.org
- Tensorflow http://www.tensorfly.cn/

Anaconda自带常用库

- Anaconda自带Numpy、Scipy、Matplotlib、Pandas、Scikit-Learn等
- 一些第三方库，例如，Tensorflow等需要单独安装

  安装Tensorflow，在终端命令窗口中，执行

  > python -m pip install --upgrade pip

  > pip install tensorflow

  > conda uninstall werkzeug

  > pip install tensorflow

- 安装PIL库，执行pip install pillow进行安装，安装库的名字是pillow
- 注意：对于Windows10，需用管理员身份运行命令提示符，否则安装过程中可能会提示拒绝访问

Python库引用

- Python自带标准库不用安装，如math，而第三方库需要安装后才可以引用
- 库的引入有两种方式：
- 第一种：import math

  WANGBIANQIN

  例如，对math库中函数采用math.<b>()形式使用

  除了使用“import 库名”之外，还可以为库起一个别名
- 第二种：from math import <函数名>

  例如，对math库中函数可直接用<函数名>()形式使用，例如：

Python库引用

```
In [1]: # 第1种导入库的方法
        import math
        # 计算正弦
        print(math.sin(1))
        # 计算指数
        print(math.exp(1))
        # 内置的圆周率常数
        math.pi
```

```
0.8414709848078965
2.718281828459045
```

```
Out[1]: 3.141592653589793
```

```
In [2]: # 除了使用“import 库名”之外，还可以为库起一个别名：
        import math as m
        m.sin(1)
```

```
Out[2]: 0.8414709848078965
```

```
In [3]: # 第二种：from math import <函数名>,只导入math库中的exp函数，并起别名e
        from math import exp as e
        # 计算指数
        print(e(1))

        # 此时sin(1)和math.sin(1)都会出错，因为没被导入
        print(sin(1))
```

```
2.718281828459045
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-3-3823d6dc766c> in <module>()
      5
      6 # 此时sin(1)和math.sin(1)都会出错，因为没被导入
----> 7 print(sin(1))

NameError: name 'sin' is not defined
```

```
In [4]: # 直接导入库中的所有函数：
        from math import *
        print(exp(1))
        print(sin(1))
```

```
2.718281828459045
0.8414709848078965
```

NumPy

- Numpy处理的最基础数据类型是由同种元素构成的多维数组（ndarray），简称“数组”
- 数组中所有元素的类型必须相同，数组中元素可以用整数索引，序号从0开始。
- ndarray类型的维度(dimensions)叫做轴(axes)，轴的个数叫做秩(rank)。一维数组的秩为1，二维数组的秩为2，二维数组相当于由两个一维数组构成。
- Numpy常用导入方式：import numpy as np

  # np为numpy别名

NumPy

- **Numpy常用创建数组（ndarry类型）函数：**

函数	描述
np.array([x,y,z], dtype=int)	从 Python 列表和元组创造数组
np.arange(x,y,i)	创建一个由 x 到 y，以 i 为步长的数组
np.linspace(x,y,n)	创建一个由 x 到 y，等分成 n 个元素的数组
np.indices((m,n))	创建一个 m 行 n 列的矩阵
np.random.rand(m,n)	创建一个 m 行 n 列的随机数组
np.ones((m,n),dtype)	创建一个 m 行 n 列全 1 的数组，dtype 是数据类型
np.empty((m,n),dtype)	创建一个 m 行 n 列全 0 的数组，dtype 是数据类型


```
In [5]: # 导入numpy库
        import numpy as np

In [6]: # 创建一维数组
        my_list = [1, 2, 3]
        x = np.array(my_list)
        print('List: ', my_list)
        print('Array: ', x)

        List:  [1, 2, 3]
        Array:  [1 2 3]

In [7]: # 创建二维数组
        m = np.array([[1, 2, 3], [4, 5, 6]])
        print(m)

        [[1 2 3]
         [4 5 6]]

In [8]: # 创建由0到30以2为步长的数组
        n = np.arange(0, 30, 2)
        print(n)

        [ 0  2  4  6  8 10 12 14 16 18 20 22 24 26 28]
```

```
In [9]: #创建特殊数组
        print('ones:\n', np.ones((3, 2)))
        print('zeros:\n', np.zeros((3, 2)))
        print('eye:\n', np.eye(3))
        print('diag:\n', np.diag(my_list))

        ones:
         [[ 1.  1.]
         [ 1.  1.]
         [ 1.  1.]]
        zeros:
         [[ 0.  0.]
         [ 0.  0.]
         [ 0.  0.]]
        eye:
         [[ 1.  0.  0.]
         [ 0.  1.  0.]
         [ 0.  0.  1.]]
        diag:
         [[1 0 0]
         [0 2 0]
         [0 0 3]]
```

NumPy

- **ndarray类的常用属性：**

属性	描述
ndarray.ndim	数组轴的个数，也被称作秩
ndarray.shape	数组在每个维度上大小的整数元组
ndarray.size	数组元素的总个数
ndarray.dtype	数组元素的数据类型，dtype 类型可以用于创建数组中
ndarray.itemsize	数组中每个元素的字节大小
ndarray.data	包含实际数组元素的缓冲区地址
ndarray.flat	数组元素的迭代器


```
In [10]: # 创建二维数组
m = np.array([[1, 2, 3], [4, 5, 6]])
print(m)
# 查看m的属性
print('ndim: ', m.ndim)
print('shape: ', m.shape)
print('size: ', m.size)
print('dtype: ', m.dtype)
print('itemsize: ', m.itemsize)
print('data: ', m.data)
print('flat: ', m.flat)
```

```
[[1 2 3]
 [4 5 6]]
ndim:  2
shape:  (2, 3)
size:  6
dtype:  int32
itemsize:  4
data:  <memory at 0x000000000503CC18>
flat:  <numpy.flatiter object at 0x0000000003EE4700>
```

创建一个数组后，可查看ndarray类型的基本属性

NumPy

- **ndarray类的形态操作：**

方法	描述
ndarray.reshape(n,m)	不改变数组 ndarray，返回一个维度为(n,m)的数组
ndarray.resize(new_shape)	与 reshape()作用相同，直接修改数组 ndarray
ndarray.swapaxes(ax1, ax2)	将数组 n 个维度中任意两个维度进行调换
ndarray. flatten()	对数组进行降维，返回一个折叠后的一维数组
ndarray.ravel()	作用同 np.flatten()，但是返回数组的一个视图


```
# 创建二维数组
m = np.array([[1, 2, 3], [4, 5, 6]])
print('m =', m)
# 二维数组变形
n = m.reshape(3, 2)
print('n =', n)
q = m.resize(3, 2)
print('q =', q)
print('m =', m)
p = m.flatten()
print('p =', p)
r = m.ravel()
print('r =', r)
```

```
m = [[1 2 3]
 [4 5 6]]
n = [[1 2]
 [3 4]
 [5 6]]
q = None
m = [[1 2]
 [3 4]
 [5 6]]
p = [1 2 3 4 5 6]
r = [1 2 3 4 5 6]
```

数组在numpy中被当作对象，可以采用<a>.<b>()方式调用一些方法

NumPy

- ndarray类的索引和切片方法：

方法	描述
x[i]	索引第 i 个元素
x[-i]	从后向前索引第 i 个元素
x[n:m]	默认步长为 1，从前往后索引，不包含 m
x[-m:-n]	默认步长为 1，从后往前索引，结束位置为 n
x[n,m,i]	指定 i 步长的由 n 到 m 的索引


```
In [22]: # 创建二维数组
m = np.array([[1, 2, 4], [2, 4, 6], [3, 7, 9]])
print('m[0]:',m[0])
print('m[1]:',m[1])
print('m[-1]:',m[-1])
print('m[0:3]:',m[0:3])
# 访问区间为左开右闭

m[0]: [1 2 4]
m[1]: [2 4 6]
m[-1]: [3 7 9]
m[0:3]: [[1 2 4]
 [2 4 6]
 [3 7 9]]
```

数组切片得到的是原始数组的视图，所有修改都会直接反映到源数组

NumPy

- **Numpy算术运算函数:**

函数	描述
np.add(x1, x2 [, y])	y = x1 + x2
np.subtract(x1, x2 [, y])	y = x1 - x2
np.multiply(x1, x2 [, y])	y = x1 * x2
np.divide(x1, x2 [, y])	y = x1 / x2
np floor_divide(x1, x2 [, y])	y = x1 // x2，返回值取整
np.negative(x [,y])	y = -x
np.power(x1, x2 [, y])	y = x1**x2
np.remainder(x1, x2 [, y])	y = x1 % x2


```
In [23]: p1 = np.ones((3, 3))
         p2 = np.arange(9).reshape(3, 3)

In [24]: print('p1: \n', p1)
         print('p2: \n', p2)
         print('p1 + p2 = \n', p1 + p2)
         print('p1 * p2 = \n', p1 * p2)
         print('p2^2 = \n', p2 ** 2)
         print('p1.p2 = \n', p1.dot(p2))

p1:
 [[ 1.  1.  1.]
 [ 1.  1.  1.]
 [ 1.  1.  1.]]
p2:
 [[0 1 2]
 [3 4 5]
 [6 7 8]]
p1 + p2 =
 [[ 1.  2.  3.]
 [ 4.  5.  6.]
 [ 7.  8.  9.]]
p1 * p2 =
 [[ 0.  1.  2.]
 [ 3.  4.  5.]
 [ 6.  7.  8.]]
p2^2 =
```

NumPy

- **Numpy比较运算函数:**

函数	符号描述
np. equal(x1, x2 [, y])	y = x1 == x2
np. not_equal(x1, x2 [, y])	y = x1 != x2
np. less(x1, x2, [, y])	y = x1 < x2
np. less_equal(x1, x2, [, y])	y = x1 <= x2
np. greater(x1, x2, [, y])	y = x1 > x2
np. greater_equal(x1, x2, [, y])	y = x1 >= x2
np.where(condition[x,y])	根据给出的条件判断输出 x 还是 y


```
a = np.array([-4, -2, 1, 3, 5])
b = np.array([-4, 0, 1, 3, 5])
print('a =',a)
print('b =',b)
print('a==b',a==b)
y = np.mean(a == b)
print('y:',y)
```

```
a = [-4 -2  1  3  5]
b = [-4  0  1  3  5]
a==b [ True False  True  True  True]
y: 0.8
```

NumPy

Numpy统计函数:

1. np.mean(x [, axis]) :
   所有元素的平均值，参数是 number 或 ndarray
2. np.sum(x [, axis]) :
   所有元素的和，参数是 number 或 ndarray
3. np.max(x [, axis]) :
   所有元素的最大值，参数是 number 或 ndarray
4. np.min(x [, axis]) :
   所有元素的最小值，参数是 number 或 ndarray
5. np.std(x [, axis]) :
   所有元素的标准差，参数是 number 或 ndarray
6. np.var(x [, axis]) :
   所有元素的方差，参数是 number 或 ndarray
7. np.argmax(x [, axis]) :
   最大值的下标索引值，参数是 number 或 ndarray
8. np.argmin(x [, axis]) :
   最小值的下标索引值，参数是 number 或 ndarray
9. np.cumsum(x [, axis]) :
   返回一个同纬度数组，每个元素都是之前所有元素的 累加和，参数是 number 或 ndarray
10. np.cumprod(x [, axis]) :
    返回一个同纬度数组，每个元素都是之前所有元素的 累乘积，参数是 number 或 ndarray

```
In [2]: a = np.array([-4, -2, 1, 3, 5])
        print('sum: ', a.sum())
        print('min: ', a.min())
        print('max: ', a.max())
        print('mean: ', a.mean())
        print('std: ', a.std())
        print('argmax: ', a.argmax())
        print('argmin: ', a.argmin())
```

```
sum:  3
min:  -4
max:  5
mean:  0.6
std:  3.26190128606
argmax:  4
argmin:  0
```

NumPy

- **Numpy其它运算函数:**

函数	描述
np.abs(x)	计算基于元素的整形，浮点或复数的绝对值。
np.sqrt(x)	计算每个元素的平方根
np.squre(x)	计算每个元素的平方
np.sign(x)	计算每个元素的符号：1(+)，0， -1(-)
np.ceil(x)	计算大于或等于每个元素的最小值
np.floor(x)	计算小于或等于每个元素的最大值
np.rint (x[, out])	圆整,取每个元素为最近的整数,保留数据类型
np.exp(x[, out])	计算每个元素指数值
np.log(x), np.log10(x), np.log2(x)	计算自然对数(e),基于 10,2 的对数,log(1 + x)

Scipy

- **Scipy：建立在 Numpy 库之上，一个数值算法集合和特定领域的工具箱，包括信号处理、优化、统计等**
- **Scipy有很多子模块可以应对不同的应用，例如插值运算，优化算法、图像处理、数学统计等** WANGBIANQIN
- **导入方式：import scipy as sp**

  from scipy. <…> import …

  例如，from scipy.linalg import svd

Scipy

- 主要包括以下子模块：
- 特殊函数 (scipy.special)
- 积分 (scipy.integrate)
- 最优化 (scipy.optimize)
- 插值 (scipy.interpolate)
- 傅立叶变换 (scipy.fftpack)
- 信号处理 (scipy.signal)
- 线性代数 (scipy.linalg)
- 稀疏特征值 (scipy.sparse)
- 统计 (scipy.stats)
- 多维图像处理 (scipy.ndimage)
- 文件 IO (scipy.io)
- 空间数据结果和算法(scipy.spatial)

模块	功能
scipy.cluster	矢量量化 / K-均值
scipy.constants	物理和数学常数
scipy.fftpack	傅里叶变换
scipy.integrate	积分程序
scipy.interpolate	插值
scipy.io	数据输入输出
scipy.linalg	线性代数程序
scipy.ndimage	n维图像包
scipy.odr	正交距离回归
scipy.optimize	优化
scipy.signal	信号处理
scipy.sparse	稀疏矩阵
scipy.spatial	空间数据结构和算法
scipy.special	任何特殊数学函数
scipy.stats	统计

Scipy

- 稀疏特征值 (scipy.sparse)

```
import numpy as np
# sparse函数的用法
from scipy import sparse

# 生成一个6x6的对角线元素为1，其余元素为0的对角矩阵
m = np.eye(6)
# 将数组转化为CSR格式的Scipy稀疏矩阵
sparse_m = sparse.csr_matrix(m)

# 查看结果
print('m:\n',m)
print('sparse_m:\n',sparse_m)
```

```
m:
 [[ 1.  0.  0.  0.  0.  0.]
 [ 0.  1.  0.  0.  0.  0.]
 [ 0.  0.  1.  0.  0.  0.]
 [ 0.  0.  0.  1.  0.  0.]
 [ 0.  0.  0.  0.  1.  0.]
 [ 0.  0.  0.  0.  0.  1.]]
sparse_m:
   (0, 0)        1.0
  (1, 1)        1.0
  (2, 2)        1.0
  (3, 3)        1.0
  (4, 4)        1.0
  (5, 5)        1.0
```

WANGBIANQIN

Scipy

- 线性代数 (scipy.linalg)

SVD分解公式：对于任意矩阵A，可分解成三个矩阵：

$$A_{m\times n} = U_{m\times m}\sum_{m\times n} V^T_{n\times n}$$

WANGBIANQIN

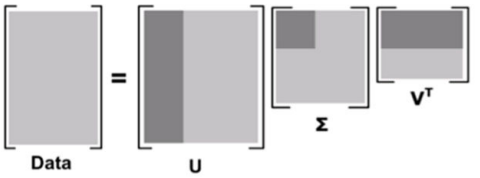
- 计算Σ前*r*个奇异值的平方和占所有奇异值的平方和的比例，如果大于90%，选*r*个奇异值重构矩阵（剩余的数据代表的可能是噪声，无用数据）。重构的A'为：

$$A_{m\times n} \approx U_{m\times r}\sum_{r\times r} V^T_{r\times n}$$

$$r = \frac{\sum_{i=1}^{r}\sigma_i^2}{\sum_{i=1}^{n}\sigma_i^2} > 90\%$$

```
# Linalg库中的svd函数对矩阵进行奇异值分解
# 其调用形式为：U,s,V = scd(M)
# 奇异值分解--举例
import numpy as np
from scipy import linalg
A = np.array([[1,-0.3],[-0.1,0.9]])
U,s,V = linalg.svd(A)

print('A:\n',A)
print('U:\n',U)
print('s:\n',s)
print('V:\n',V)
```

```
A:
 [[ 1.  -0.3]
 [-0.1  0.9]]
U:
 [[-0.81937847  0.57325293]
 [ 0.57325293  0.81937847]]
s:
 [ 1.16140394  0.74909338]
V:
 [[-0.7548655   0.65587962]
 [ 0.65587962  0.7548655 ]]
```



Scipy

http://liao.cpython.org/scipy06/

a10	2019/3/6 9:07	PNG 图像	380 KB
a50	2019/3/6 9:07	PNG 图像	548 KB
a100	2019/3/6 9:07	PNG 图像	642 KB
org	2019/3/6 9:07	PNG 图像	781 KB


```
# 利用svd实现对图像的压缩
import scipy.misc
from scipy.linalg import svd
import matplotlib.pyplot as plt
%matplotlib inline
import numpy as np

img = scipy.misc.face()[:,:,0]

print(img.shape,type(img))
img = np.matrix(img)
U,s,Vh = svd(img)
plt.gray()
plt.subplot(221,aspect='equal')
plt.title("orignal")
plt.imshow(img)
plt.imsave('org.png', img)

A = np.dot(U[:,0:10],np.dot(np.diag(s[0:10]),Vh[0:10,:]))
plt.subplot(222,aspect='equal')
plt.title(":10")
plt.imshow(A)
plt.imsave('a10.png', A)

A = np.dot(U[:,0:50],np.dot(np.diag(s[0:50]),Vh[0:50,:]))
plt.subplot(223,aspect='equal')
plt.title(":50")
plt.imshow(A)
plt.imsave('a50.png', A)

A = np.dot(U[:,0:100],np.dot(np.diag(s[0:100]),Vh[0:100,:]))
plt.subplot(224,aspect='equal')
plt.title(":100")
plt.imshow(A)
plt.imsave('a100.png', A)
```

数据可视化

- **Anscombe's quartet (安斯库姆四重奏)**

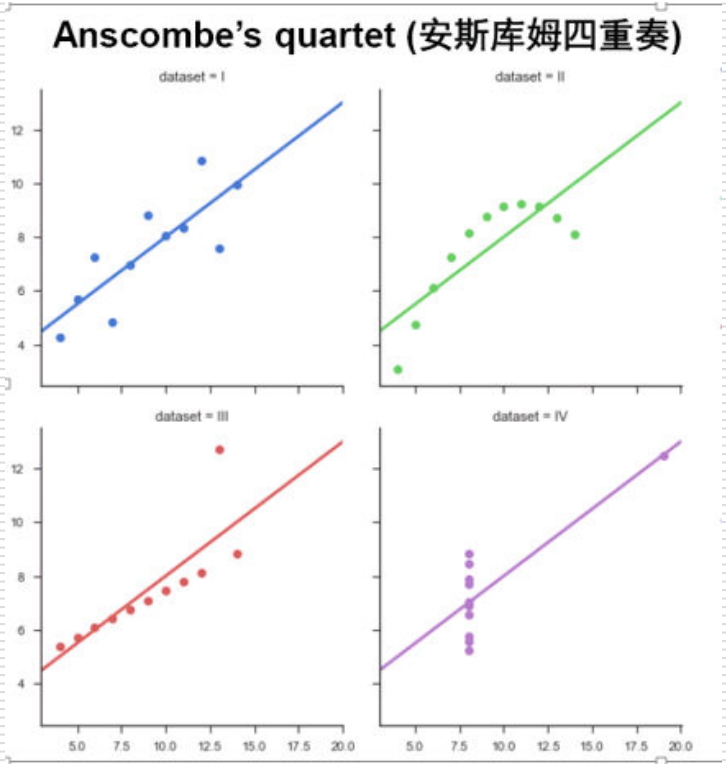
	dataset	x	y
0	I	10.0	8.04
1	I	8.0	6.95
2	I	13.0	7.58
3	I	9.0	8.81
4	I	11.0	8.33
5	I	14.0	9.96
6	I	6.0	7.24
7	I	4.0	4.26
8	I	12.0	10.84
9	I	7.0	4.82
10	I	5.0	5.68
11	II	10.0	9.14
12	II	8.0	8.14
13	II	13.0	8.74
14	II	9.0	8.77
15	II	11.0	9.26
16	II	14.0	8.10
17	II	6.0	6.13
18	II	4.0	3.10
19	II	12.0	9.13
20	II	7.0	7.26
21	II	5.0	4.74
22	III	10.0	7.46
23	III	8.0	6.77
24	III	13.0	12.74
25	III	9.0	7.11
26	III	11.0	7.81
27	III	14.0	8.84
28	III	6.0	6.08
29	III	4.0	5.39
30	III	12.0	8.15
31	III	7.0	6.42
32	III	5.0	5.73
33	IV	8.0	6.58
34	IV	8.0	5.76
35	IV	8.0	7.71
36	IV	8.0	8.84
37	IV	8.0	8.47
38	IV	8.0	7.04
39	IV	8.0	5.25
40	IV	19.0	12.50
41	IV	8.0	5.56
42	IV	8.0	7.91
43	IV	8.0	6.89

数据可视化

- 这四组数据的共同统计特性如下：

性质	数值
x的平均数	9
x的方差	11
y的平均数	7.50（精确到小数点后两位）
y的方差	4.122或4.127（精确到小数点后三位）
x与y之间的相关系数	0.816（精确到小数点后三位）
线性回归线	$y = 3.00 + 0.500x$（分别精确到小数点后两位和三位）

	x		y	
	mean	var	mean	var
dataset				
I	9.0	11.0	7.500909	4.127269
II	9.0	11.0	7.500909	4.127629
III	9.0	11.0	7.500000	4.122620
IV	9.0	11.0	7.500909	4.123249

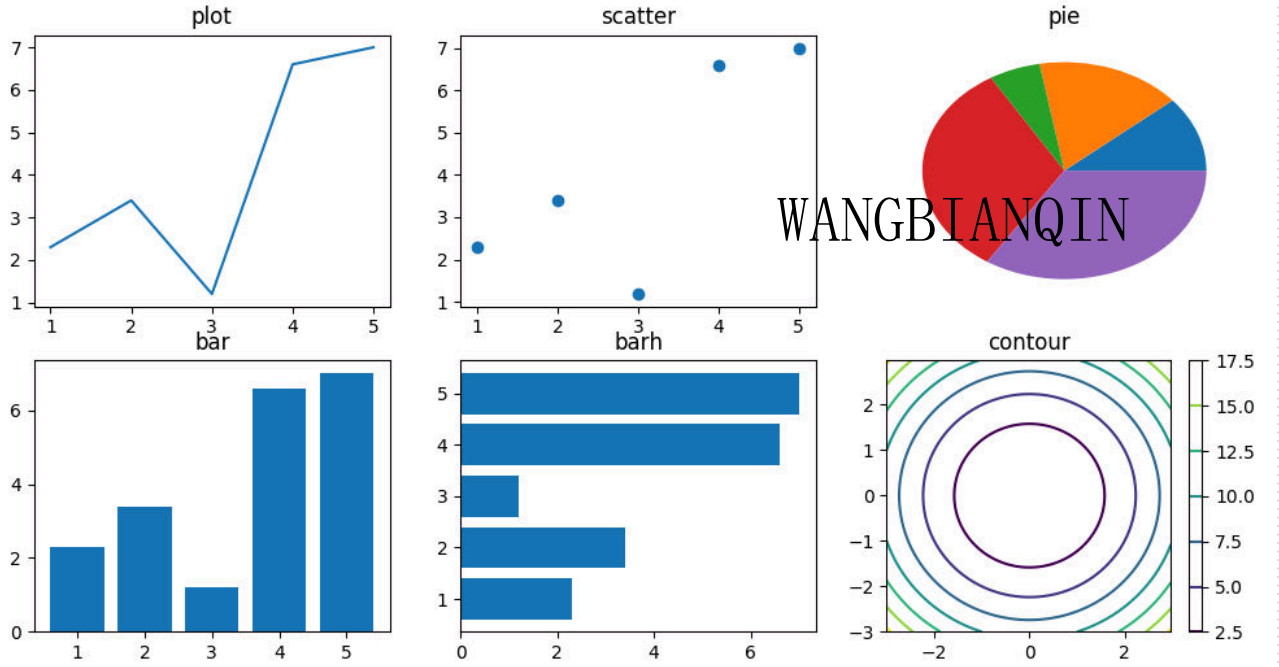


- 请利用pandas或numpy的统计函数验证Anscombe数据的统计值

Matplotlib

- 可视化库： Matplotlib，Pandas，Seaborn
- Matplotlib：提供数据绘图功能的第三方库，其子库pyplot主要用于实现各种数据展示图的绘制
- 导入方式：import matplotlib.pyplot as plt
- plt子库提供了一批操作和绘图函数，每个函数代表对图像进行的一个操作，比如创建绘图区域、添加标注或者修改坐标轴等
- 函数采用plt.<b>()形式调用，其中<b>是具体函数名称

Matplotlib




```
In [30]: #导入matplotlib.pyplot，并用plt作biem别名
import matplotlib.pyplot as plt
#绘制一维数组图像
x = [1,2,3,4,5]
y = [2.3,3.4,1.2,6.6,7.0]
#定义图的大小
plt.figure(figsize=(12,6))

#定义图片展示方式为二行三列
plt.subplot(231)
plt.plot(x,y)
plt.title("plot")

plt.subplot(232)
plt.scatter(x, y)
plt.title("scatter")

plt.subplot(233)
plt.pie(y)
plt.title("pie")

plt.subplot(234)
plt.bar(x, y)
plt.title("bar")

plt.subplot(235)
plt.barh(x, y)
plt.title("barh")

import numpy as np
delta = 0.025
x = y = np.arange(-3.0, 3.0, delta)
X, Y = np.meshgrid(x, y)
Z = Y**2 + X**2

plt.subplot(236)
plt.contour(X,Y,Z)
plt.colorbar()
plt.title("contour")

plt.show()
```

plt 绘图区域函数

函数	描述
plt.figure(figsize=None， facecolor=None)	创建一个全局绘图区域
plt.axes(rect, axisbg='w')	创建一个坐标系风格的子绘图区域
plt.subplot(nrows, ncols, plot_number)	在全局绘图区域中创建一个子绘图区域
plt.subplots_adjust()	调整子图区域的布局

plt 读取和显示函数

函数	描述
plt.legend()	在绘图区域中方式绘图标签（也称图注）
plt.show()	显示创建的绘图对象
plt.matshow()	在窗口显示数组矩阵
plt.imshow()	在 axes 上显示图像
plt.imsave()	保存数组为图像文件
plt.imread()	从图像文件中读取数组

用于在绘图区域中增加显示内容及读入数据，这些函数需要与其他函数搭配使用

plt 基础图表函数

操作	描述
plt.polt(x, y, label, color, width)	根据 x, y 数组绘制直/曲线
plt.boxplot(data, notch, position)	绘制一个箱型图（Box-plot）
plt.bar(left, height, width, bottom)	绘制一个条形图
plt.barh(bottom, width, height, left)	绘制一个横向条形图

plt.contour(X, Y, Z, N)	绘制等值线
plt.vlines()	绘制垂直线
plt.stem(x, y, linefmt, markerfmt, basefmt)	绘制曲线每个点到水平轴线的垂线
plt.plot_date()	绘制数据日期
plt.plotfile()	绘制数据后写入文件

plt.polar(theta, r)	绘制极坐标图
plt.pie(data,explode)	绘制饼图
plt.psd(x, NFFT=256, pad_to, Fs)	绘制功率谱密度图
plt.specgram(x, NFFT=256, pad_to, F)	绘制谱图
plt.cohere (x, y, NFFT=256, Fs)	绘制 X-Y 的相关性函数
plt.scatter()	绘制散点图（x, y 是长度相同的序列）
plt.step(x, y, where)	绘制步阶图
plt.hist(x, bins, normed)	绘制直方图

plt 坐标轴设置函数

函数	描述
plt.axis('v','off','equal','scaled','tight','image')	获取设置轴属性的快捷方法
plt.xlim(xmin, xmax)	设置当前 x 轴取值范围
plt.ylim(ymin,ymax)	设置当前 y 轴取值范围
plt.xscale()	设置 x 轴缩放
plt.yscale()	设置 y 轴缩放
plt.autoscale()	自动缩放轴视图的数据
plt.thetagrids(angles, labels, fmt, frac)	设置极坐标网格 theta 的位置
plt.grid(on/off)	打开或者关闭坐标网格

plt 标签设置函数

函数	描述
plt.figlegend(handles, label, loc)	为全局绘图区域放置图注
plt.legend()	为当前坐标图放置图注
plt.xlabel(s)	设置当前 x 轴的标签
plt.ylabel(s)	设置当前 y 轴的标签
plt.xticks(array, 'a', 'b', 'c')	设置当前 x 轴刻度位置的标签和值
plt.yticks(array, 'a', 'b', 'c')	设置当前 y 轴刻度位置的标签和值
plt.clabel(cs,v)	为等值线图设置标签
plt.get_figlabels()	返回当前绘图区域的标签列表
plt.figtext(x, y, s, fontdic)	为全局绘图区域添加文字
plt.title()	设置标题
plt.suptitle()	为当前绘图区域添加中心标题
plt.text(x, y, s, fontdic, withdash)	为坐标图轴添加注释
plt.annotate(note, xy, xytext, xycoords, textcoords, arrowprops)	用箭头在指定数据点创建一个注释或一段文本

WANGBIANQIN

plt.pcolormesh可以画彩图，例如绘制分类边界

Pandas

- Pandas：高性能易用数据类型和分析工具
- 在终端命令行安装：pip install pandas
- 常用导入方式：import pandas as pd
- Pandas基于NumPy，常与NumPy，Matplotlib一同使用
- 两个数据类型：Series，DataFrame
- Series，DataFrame数据类型的各类操作：基本操作、运算操作、特征类操作、关联类操作

Pandas

In [31]:
```
# 通常用pd作为Pandas的别名
import pandas as pd
# 创建一个序列s
s = pd.Series([1, 2, 3], index = ['a', 'b', 'c'])
# 创建一个表
d = pd.DataFrame([[1, 2, 3], [4, 5, 6]], columns = ['a', 'b', 'c'])
# 也可以用已有的序列来创建表格
d2 = pd.DataFrame(s)
# 预览前5行数据
d.head()
# 数据基本统计量
d.describe()
```

WANGBIANQIN

Out[31]:

	a	b	c
count	2.00000	2.00000	2.00000
mean	2.50000	3.50000	4.50000
std	2.12132	2.12132	2.12132
min	1.00000	2.00000	3.00000
25%	1.75000	2.75000	3.75000
50%	2.50000	3.50000	4.50000
75%	3.25000	4.25000	5.25000
max	4.00000	5.00000	6.00000

Scikit-learn

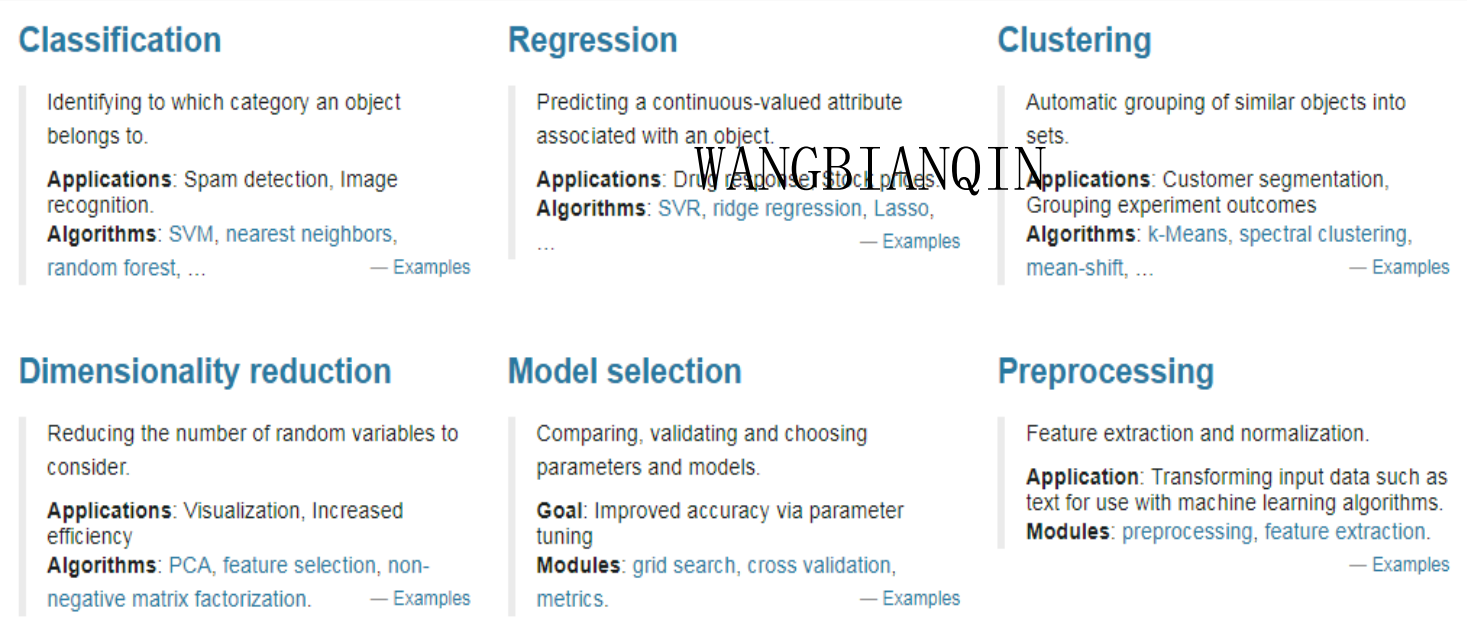
- **Scikit-learn：封装了常用机器学习算法，依赖NumPy、SciPy，常与 pandas、Matplotlib 协同工作**

- **安装：pip install scikit-learn** WANGBIANQIN
- **导入：import sklearn**
  from sklearn. <…> import …

例如，**from sklearn import model_selection**
from sklearn.linear_model import LogisticRegression

Scikit-learn

- 官网：http://scikit-learn.org/stable/testimonials/testimonials.html



Classification

Identifying to which category an object belongs to.

Applications: Spam detection, Image recognition.
Algorithms: SVM, nearest neighbors, random forest, ... — Examples

Regression

Predicting a continuous-valued attribute associated with an object.

Applications: Drug response, Stock prices.
Algorithms: SVR, ridge regression, Lasso, ... — Examples

Clustering

Automatic grouping of similar objects into sets.

Applications: Customer segmentation, Grouping experiment outcomes
Algorithms: k-Means, spectral clustering, mean-shift, ... — Examples

Dimensionality reduction

Reducing the number of random variables to consider.

Applications: Visualization, Increased efficiency
Algorithms: PCA, feature selection, non-negative matrix factorization. — Examples

Model selection

Comparing, validating and choosing parameters and models.

Goal: Improved accuracy via parameter tuning
Modules: grid search, cross validation, metrics. — Examples

Preprocessing

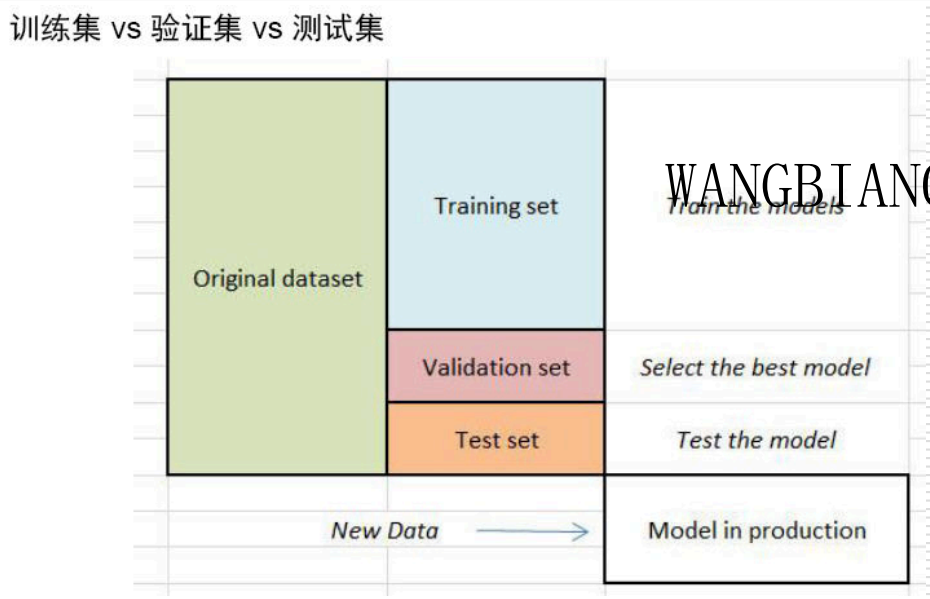
Feature extraction and normalization.

Application: Transforming input data such as text for use with machine learning algorithms.
Modules: preprocessing, feature extraction. — Examples

Scikit-Learn

Scikit-Learn

- **数据集划分**

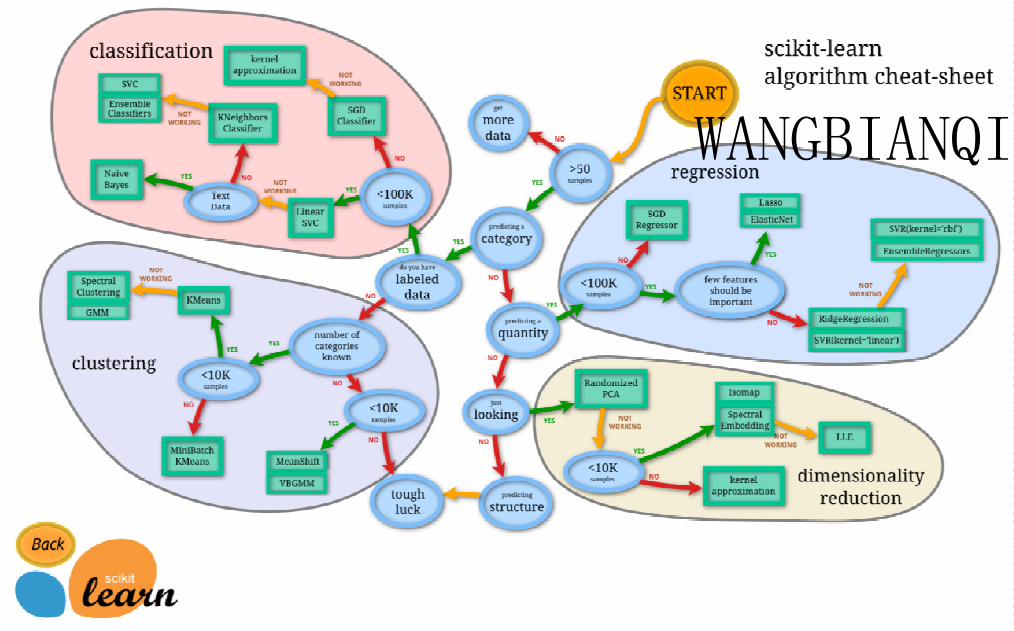


- **内置数据集：https://scikit-learn.org/stable/modules/classes.html#module-sklearn.datasets**

Scikit-Learn

- **模型选择：选择路线图**

http://scikit-learn.org/stable/tutorial/machine_learning_map/index.html



Scikit-Learn

训练模型

- Estimator对象
- 从训练数据学习得到的
- 可以是分类算法、回归算法或者是特征提取算法
- fit方法用于训练Estimator
- Estimator的参数可以训练前初始化，或者之后更新


```
In [50]: from sklearn.neighbors import KNeighborsClassifier
         knn = KNeighborsClassifier(n_neighbors=1)

In [51]: knn.fit(X_train, y_train)

Out[51]: KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                    metric_params=None, n_jobs=1, n_neighbors=1, p=2,
                    weights='uniform')
```

Scikit-Learn

测试模型

- model.predict(X_test)
  - 返回测试样本的预测标签
- model.score(X_test, y_test)
  - 根据预测值和真实值计算评分

```
In [55]: y_pred = knn.predict(X_test)
         print("Test set predictions:\n {}".format(y_pred))

Test set predictions:
 [2 1 0 2 0 2 0 1 1 1 2 1 1 1 1 0 1 1 0 0 2 1 0 0 2 0 0 1 1 0 2 1 0 2 2 1 0
 2]
```



```
In [56]: print("Test set score: {:.2f}".format(np.mean(y_pred == y_test)))

Test set score: 0.97

In [59]: print("Test set score: {:.2f}".format(knn.score(X_test, y_test)))
```

- score()对Estimator进行评分
  - 回归模型：使用“决定系数”评分（Coefficient of Determination）
  - 分类模型：使用“准确率”评分（accuracy）

案例--鸢尾花(Iris)分类

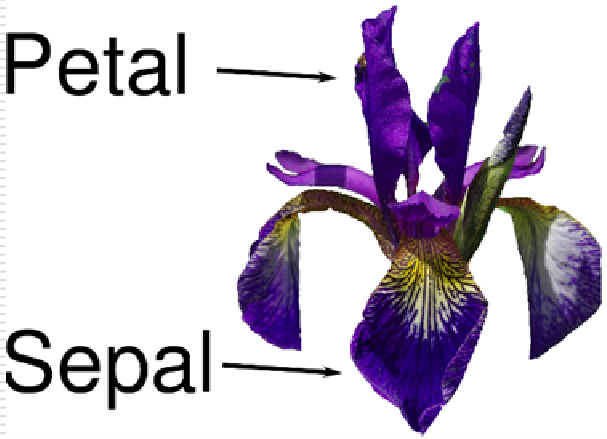
三种鸢尾花类型

山鸢尾花 Setosa、变色鸢尾花 Versicolor、韦尔吉尼娅鸢尾花 Virginica

初识数据

```
In [34]: from sklearn.datasets import load_iris
         iris_dataset = load_iris()

In [35]: print("Keys of iris_dataset: {}".format(iris_dataset.keys()))
         Keys of iris_dataset: dict_keys(['target', 'DESCR', 'data', 'target_names', 'feature_names'])

In [37]: print(iris_dataset['DESCR'][:193] + "\n...")
         Iris Plants Database
         ====================

         Notes
         -----
         Data Set Characteristics:
             :Number of Instances: 150 (50 in each of three classes)
             :Number of Attributes: 4 numeric, predictive att
         ...

In [38]: print("Target names: {}".format(iris_dataset['target_names']))
         Target names: ['setosa' 'versicolor' 'virginica']

In [39]: print("Feature names: {}".format(iris_dataset['feature_names']))
         Feature names: ['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width (cm)']

In [40]: print("Type of data: {}".format(type(iris_dataset['data'])))
         Type of data: <class 'numpy.ndarray'>

In [41]: print("Shape of data: {}".format(iris_dataset['data'].shape))
         Shape of data: (150, 4)
```

WANGBIANQIN



初识数据

```
In [42]: print("First five rows of data:\n{}".format(iris_dataset['data'][:5]))

First five rows of data:
[[ 5.1  3.5  1.4  0.2]
 [ 4.9  3.   1.4  0.2]
 [ 4.7  3.2  1.3  0.2]
 [ 4.6  3.1  1.5  0.2]
 [ 5.   3.6  1.4  0.2]]

In [43]: print("Type of target: {}".format(type(iris_dataset['target'])))

Type of target: <class 'numpy.ndarray'>

In [44]: print("Shape of target: {}".format(iris_dataset['target'].shape))

Shape of target: (150,)

In [45]: print("Target:\n{}".format(iris_dataset['target']))

Target:
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2]
```

数据集分割

In [46]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(
    iris_dataset['data'], iris_dataset['target'], random_state=0)
```

In [47]:
```
print("X_train shape: {}".format(X_train.shape))
print("y_train shape: {}".format(y_train.shape))
```

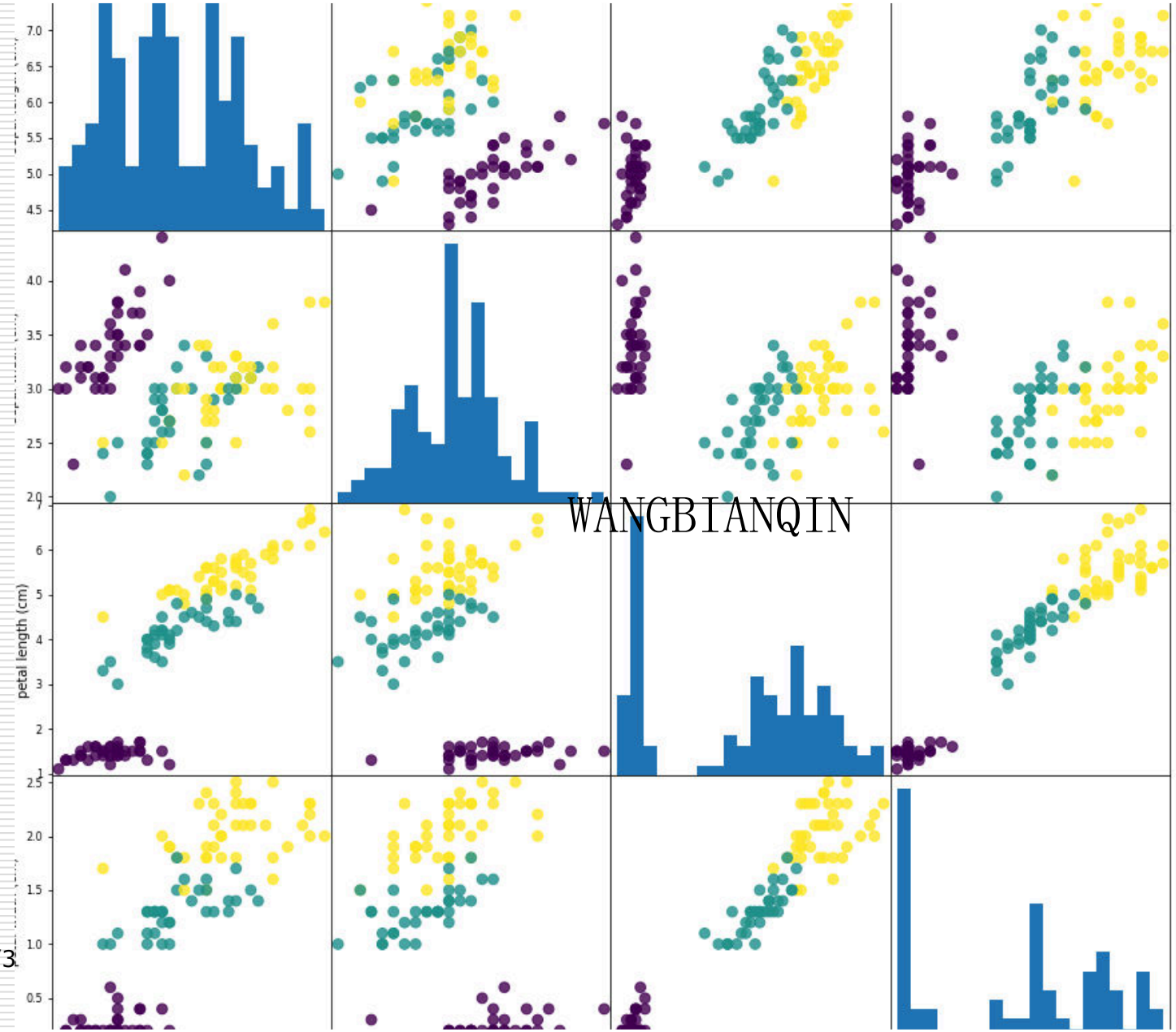
```
X_train shape: (112, 4)
y_train shape: (112,)
```


In [48]:
```
print("X_test shape: {}".format(X_test.shape))
print("y_test shape: {}".format(y_test.shape))
```

```
X_test shape: (38, 4)
y_test shape: (38,)
```

探索数据

In [61]:
```
# create dataframe from data in X_train
# label the columns using the strings in iris_dataset.feature_names
iris_dataframe = pd.DataFrame(X_train, columns=iris_dataset.feature_names)
# create a scatter matrix from the dataframe, color by y_train
grr=pd.scatter_matrix(iris_dataframe, c=y_train, figsize=(15, 15), marker='o',
                      hist_kwds={'bins': 20}, s=60, alpha=.8)
```


petal length (cm)
WANGBIANQIN

构建模型：k近邻算法

```
In  [50]: from sklearn.neighbors import KNeighborsClassifier
          knn = KNeighborsClassifier(n_neighbors=1)

In  [51]: knn.fit(X_train, y_train)WANGBIANQIN

Out[51]: KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                              metric_params=None, n_jobs=1, n_neighbors=1, p=2,
                              weights='uniform')
```

评估模型

```
In [55]: y_pred = knn.predict(X_test)
         print("Test set predictions:\n {}".format(y_pred))
```

```
Test set predictions:
 [2 1 0 2 0 2 0 1 1 1 2 1 1 1 1 0 1 1 0 0 2 1 0 0 2 0 0 1 1 0 2 1 0 2 2 1 0
 2]
```

WANGBIANQIN

```
In [56]: print("Test set score: {:.2f}".format(np.mean(y_pred == y_test)))
```

```
Test set score: 0.97
```

```
In [59]: print("Test set score: {:.2f}".format(knn.score(X_test, y_test)))
```

```
Test set score: 0.97
```

模型预测

```
In [53]: X_new = np.array([[5, 2.9, 1, 0.2]])
         print("X_new.shape: {}".format(X_new.shape))

X_new.shape: (1, 4)
```

WANGBIANQIN

```
In [54]: prediction = knn.predict(X_new)
         print("Prediction: {}".format(prediction))
         print("Predicted target name: {}".format(
               iris_dataset['target_names'][prediction]))

Prediction: [0]
Predicted target name: ['setosa']
```

案例总结

```
In [60]: X_train, X_test, y_train, y_test = train_test_split(
             iris_dataset['data'], iris_dataset['target'], random_state=0)

         knn = KNeighborsClassifier(n_neighbors=1)
         knn.fit(X_train, y_train)

         print("Test set score: {:.2f}".format(knn.score(X_test, y_test)))

         Test set score: 0.97
```


- fit, predict, score是scikit-learn监督学习模型中最常用的接口

小结

- NumPy：多维数组及其运算
- Scipy：数值算法与特定领域算法集合
- Matplotlib：绘制2D，3D图
- Pandas：高性能的数据结构及其分析
- Scikit-Learn：机器学习算法和工具
- Tensorflow：第5讲介绍