第二章 统计学习

2.1 统计学的一般概念

可约误差和不可约误差：可约误差是预测函数中可以消去的误差，通过优化模型可以使之减小；不可约误差是始终存在的误差，永远不可能为零。不可约误差记为ϵ

> 可以认为可约误差对应偏差，不可约误差对应方差

参数化方法和非参数化方法：参数化方法需要找到一系列参数，求出参数后得到估计函数；非参数化方法则是直接给出估计函数，使之完全拟合已有的数据

> 参数数量少的估计函数，估计误差比较大；而复杂的模型则可能出现过拟合的问题

监督学习和无监督学习：监督学习包含训练集和测试样本，典型的有**回归分析**；而无监督学习没有测试样本，需要在数据集中寻找规律，典型的是**聚类**

回归与分类问题：变量可以采用数值的是定量问题，否则是定性问题。前者采用回归分析，后者采用分类。

2.2 统计方法的可解释性和灵活性分布

按照灵活性增大，可解释性下降进行排布：

1. 子集选择；Lasso
2. 最小二乘
3. 线性可加模型；树
4. 装袋；Boosting
5. 支持向量机
6. 深度学习

2.3 统计模型准确性的评估

最近邻方法(Nearest neighbor)： 考虑一个比较小的数据集，若在某点上没有对应数据，则可用附近点的值作平均来估计该点值大小

> 此方法在高维度时效果不好，这被称为**高维度诅咒(curse of dimensionality)**

均方误差：均方误差(MSE)可以用于衡量预测值在多大程度上接近于观测的真实值，它通过计算预测值与实际观测值之间差异的平方的平均值来衡量预测的准确度。MSE越小，说明模型的拟合效果越好。

$$MSE = \frac{\sum_{i=1}^{n}(y_i - \hat{f}(x_i))^2}{n}$$

训练均方误差：用于训练的数据集与测试模型之间误差平方的平均值，记为训练均方误差(tranning MSE)

测试均方误差：训练模型完成后，用一组用于检测模型训练效果的数据集计算test MSE，根据

$$Ave(y_0 - \hat{f}(x_0))^2$$

比较哪个模型估计得最精确

> 训练均方误差和测试均方误差使用的数据集不同。前者使用的是训练数据集，后者使用的是测试数据集

自由度：自由度可以用于描述统计模型的灵活性。随着模型灵活性的增加，训练MSE会降低，但测试MSE可能不会降低。当一个模型的训练均方误差很小但测试均方误差比较大时，我们认为这个模型出现了**过拟合**

> 过拟合特指灵活性较差的模型会产生较小的测试均方误差的情况。模型通过不断地拟合训练数据集，可以对训练数据集产生很小的均方误差

期望测试均方误差：用大量的训练集重复估计f，并在x_0处分别检验，可以得到期望测试均方误差(expected test MSE)。对测试集中x_0的所有可能值取平均，可以计算期望测试均方误差。

$$E(y_0 - \hat{f}(x_0))^2 = Var(\hat{f}(x_0)) + [Bias(\hat{f}(x_0))]^2 + Var(\epsilon)$$

> $Var(\hat{f}(x_0))^2$表示不同数据集在x_0处给出的$\hat{f}(x_0)$的变异程度，$[Bias(\hat{f}(x_0))]^2$表示模型与实际值偏差的程度

方差：方差(variance)可以用来表示模型对检验数据集的拟合程度。

> 方差是相对于检验数据集而言的。高方差意味着模型对测试数据集的变动非常敏感，泛化能力差

偏差：偏差(bias)是指用一个简单得多的模型去近似一个可能非常复杂的实际问题所引入的误差。普遍来讲，更高自由度的模型具有更小的偏差

> 偏差指的是模型预测值与真实值之间的差距，高偏差意味着模型对测试数据集和训练数据集的拟合程度都不好

方差和偏差的相对变化率，决定了test MSE是增大还是减小。因此，test MSE 随着自由度的变化曲线是一个向下的抛物线

方差-偏差权衡：方差-偏差权衡(bias-variance trade-off)表明，在实际的模型选择中，我们需要对模型的方差和偏差找到一个平衡点。过高的偏差可能导致模型无法识别数据特征，过高的方差则可能导致模型对噪声敏感

2.4 统计模型分类体系

误差率：通过计算

$$\frac{1}{n}\sum_{i=1}^{n} I(y_i \neq \hat{y}_i)$$

可以得到模型的误差率(error rate)，从而判断模型的准确程度。这个数据表示了分类的错误比率。

训练误差：分类器对训练数据的误差率被称为训练误差

测试误差：分类器对测试数据的误差率被称为测试误差

$$Ave(I(y_0 \neq \hat{y_0}))$$

2.4.1 贝叶斯分类器

贝叶斯分类器(Bayes Classifier) 是一种基于贝叶斯定理的统计分类方法。

记Y为类别，X为特征，则根据**预测向量**x_0的值将其分配给类中概率最大的第j类，概率可表示为：

$$Pr(Y = j|X = x_0)$$

在一个二元分类中，当$Pr(Y = j|X = x_0) > 0.5$时，我们认为它是属于分类j的；在$Pr(Y = j|X = x_0) = 0.5$处，形成了**贝叶斯决策边界(Bayes decision boundary)**

贝叶斯分类器产生的最低可能的错误率，被称为**贝叶斯错误率(Bayes error rate)**：

$$1 - max_j Pr(Y = j|X = x_0)$$

> 因为贝叶斯分类器总是选择贝叶斯概率最大的分类，因此实际上非此分类的变量都认为是发生了错误。$max_j(Pr(Y = j|X = x_0)$表示的是$X = x_0$时，X最有可能得分类

根据上式，可以计算总体贝叶斯错误率：

$$1 - E(max_j Pr(Y = j|X = x_0))$$

> 贝叶斯误差率类似于不可约误差
> 贝叶斯分类是测试其他分类性能的金标准

2.4.2 K近邻

K近邻(K-Nearest Neighbors KNN) 是一种通过测量不同特征值之间的距离来进行预测的分类方法

选取一个合适的K值，对于一个观测变量，考虑训练数据与其最近的K个点，记为N_0，然后将第j类的条件概率估计为N_0中响应值为j的点的分数：

$$Pr(Y = j|X = x_0) = \frac{1}{K}\sum_{i \in N_0} I(y_i = j)$$

最后选取x_0在不同分类中的一个最大的概率，作为x_0的分类

当K值越大时，K近邻的灵活性下降；当K值越小时，K近邻的灵活性上升。过高的灵活性可能出现过拟合，因此当K值上升时，实验误差呈现U型特征

> 例如，x_0临近的K个值中，黄色点最多，则考虑x_0也是黄色点

2.5 Python环境配置

2.5.1 安装Anaconda

Anaconda是一个Python工具集。常用到的功能包括：

7. 使用conda配置虚拟环境
8. 在anaconda中安装各种包

本书中使用Anaconda的流程如下

2.5.1.1 下载Anaconda

在官网中下载Anaconda，选择好位置，**勾选配置环境变量**

2.5.1.2 安装Anaconda Prompt

在Anaconda Navigator中安装Anaconda Prompt，之后的环境创建和包配置都在此处

2.5.1.3 配置虚拟环境

为了安装ISLP包，先配置虚拟环境。打开Anaconda Prompt，输入：

```
conda create --name islp python
```

创建一个名为*islp*的虚拟环境

之后，继续输入：

```
conda activate islp
```

使用名为*islp*的虚拟环境。以后每次进入Anaconda Prompt，都需要使用此命令

为虚拟环境安装ISLP包：

```
pip install ISLP
```

> 使用清华源的命令 `pip install ISLP -i https://pypi.mirrors.ustc.edu.cn/simple`

2.5.2 使用Jupyter Notebook

在Anaconda 中安装Jupyter Notebook，在浏览器中打开

右上角新建一个Python文件，开始创作

2.6 关于Python和包的简单介绍

2.6.1 Python简介

Python中的三个序列是：

- 列表 List
- 元组 Tuple
- 字符串 String

对列表直接进行相加，返回的是连接后的列表

> 相当于字符串的连接

2.6.2 numpy简介

数组、元组等可以用numpy转换为array型，进行数学运算：

```
x = np.array([1, 2, 3]) #一维数组表示向量

y = np.array([[1, 2], [3, 4]]) #多维数组表示矩阵
```

> 注意array只能接受一个序列，因此y表示为嵌套列表

array型包含许多**属性(attribute)** ，可以用各种函数访问，也可以指定属性：

```
x.ndim #获取x的维度，也就是每一列的元素个数，也即行数
x.dtype #获取x的数据类型
np.array([[1, 2], [3, 4]], float) #指定数据类型为浮点型
x.shape #获取x的行列信息，返回一个元组
```

方法(method) 是一种与对象相关联的函数，下面的代码展示了其与函数的区别：

```
x.sum() #用sum方法对向量内的元素求和
np.sum(x) #用sum函数求x内的元素和
```

reshape方法可以更方便的创建矩阵：

```
z = np.array([1, 2, 3, 4, 5, 6])
print('beginning x:\n', z)
z_reshape = z.reshape((2, 3))
print('reshaped x:\n', z_reshape)
```

```
z_reshape.T #对z进行转置
```

与python一样，numpy采用0为起始索引：

```
z_reshape[0, 0] #访问第一行第一列的元素
z_reshape[0] #访问第一行
```

修改reshape后的array，会导致初始array也被修改，因为它们使用了相同的内存；tuple不能被修改

> .reshap()方法不会修改原array，只返回一个reshape后的array

一些数学运算可以直接运用到array上：

```
np.sqrt(x) #对x中的每个元素开根号
x**2 #每个元素乘方
x**0.5 #每个元素开方
```

关键字参数(keyword) 在函数中可以以名称的方式指代，且没有顺序：

```
x = np.random.normal(size=50) #创建一个长度为50的随机数array
```

np.random.normal是用于生成正态分布数据的函数，包含三个关键字参数loc(指定正态分布的均值)、scale(指定正态分布的标准差)和size(表示数据规模，也可以是元组)：

```
np.random.normal(loc = 50, scale = 1, size = 50)
```

np.corrcoef()是用于计算相关矩阵的函数：

```
np.corrcoef(x, y)
```

> 相关矩阵显示了多个变量之间的相关系数r的大小，对角线上的值为1，是一个对称矩阵。
> 相关矩阵可以用协方差矩阵类比

通过指定**随机数种子(random seed)** 可以实现随机数固定化生成：

```
rng = np.random.default_rng(1303) #1303是随机数种子
```

np.mean()，np.var()，np.std()分别计算array的平均值，方差和标准差，也可作为方法使用：

```
rng = np.random.default_rng(3)
y = rng.standard_normal(10)#生成标准正态分布的array
np.mean(y) #作为函数使用
```

```
y.var() #作为方法使用
y.std(axis=0) #指定第一个维度
```

2.6.3 matplotlib介绍

在matplotlib中，一个图由一个图形和一个或多个坐标轴组成；一般使用plt作为包的别名：

```
import matplotlib.pyplot as plt
```

使用plt.subplots()创建一副图表。这个函数将生成一个包含两个参数的元组，这两个参数分别表示图像和坐标轴对象：

```
fig, ax = plt.subplots(figsize=(8, 8)) #figsize指定图表的大小

rng = np.random.default_rng()
 x = rng.standard_normal(100)
 y = rng.standard_normal(100) #准备数据

 ax.plot(x, y) #确定坐标轴的数据
 plt.show() #展示图片
```

通过指定参数或使用其他函数，可以指定图表格式：

```
ax.scatter(x, y, marker='o') #marker指定绘图颜色,scatter()改为散点图
```

用set_xlabel()，set_ylabel()，set_title()指定坐标轴标题：

```
fig , ax = subplots(figsize =(8, 8))
ax.scatter(x, y, marker='o') ax.set_xlabel("this is the x-axis")
ax.set_ylabel("this is the y-axis")
ax.set_title("Plot of X vs Y")
```

> fig可以看做整幅图像，ax是其中的一个子图；显然可以创建多幅子图

可以在同一个图像对象中创建多幅图像：

```
fig , axes = subplots(nrows=2, ncols=3, figsize=(15, 5)) #通过nrows和ncol参数
指定小图表的数量

axes[0 ,1].plot(x, y, 'o') #指定第一行第二列图表
axes[1 ,2].scatter(x, y, marker='+') #指定第二行第三列的图表
```

使用fig.savefig()保存图像：

```
fig.savefig("Figure1.png", dpi=400) #dpi指定分辨率
```

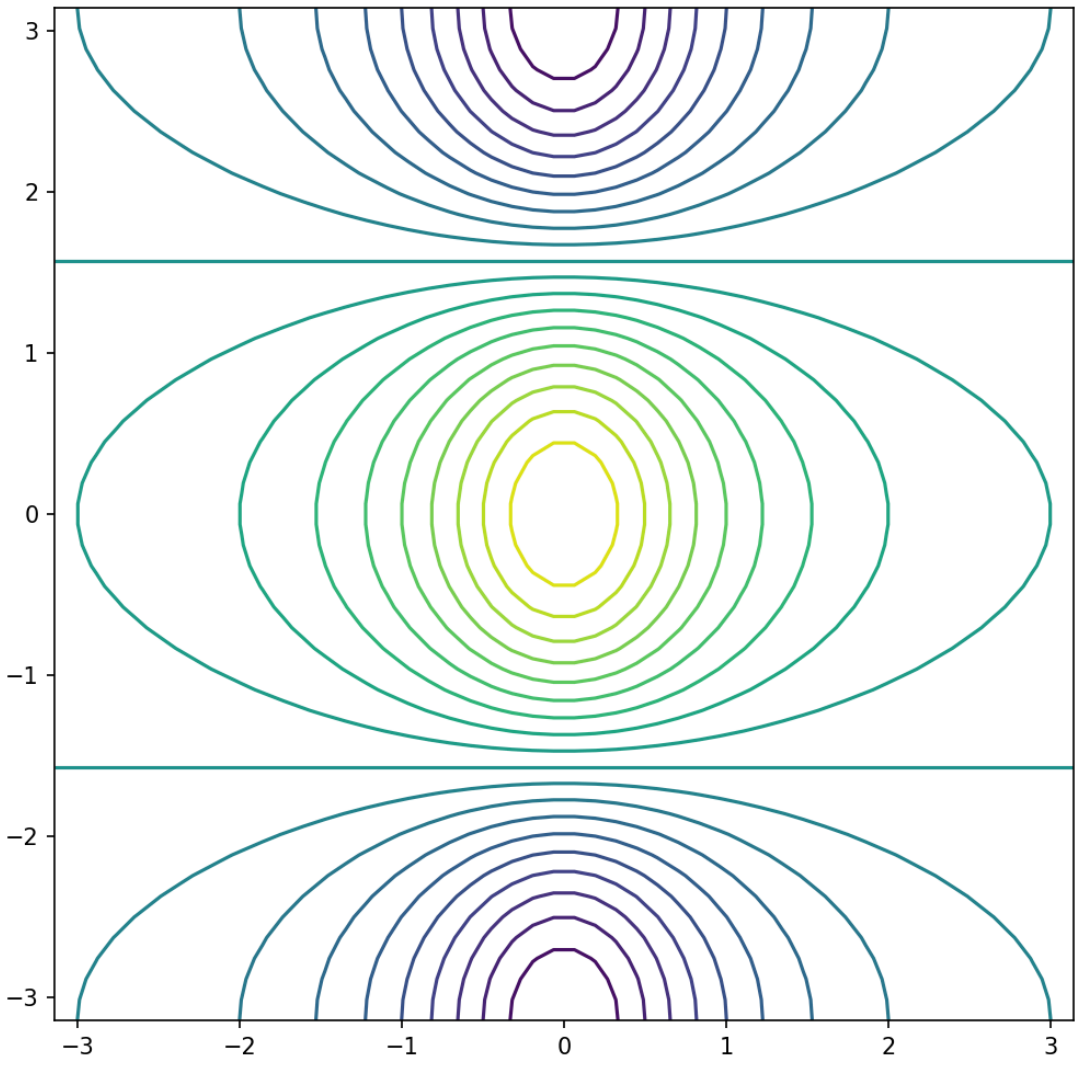
我们可以继续采用逐步更新的方式对图进行修改；例如，我们可以修改x轴的范围，重新保存图形，甚至重新显示图形：

```
axes [0 ,1]. set_xlim ([-1 ,1])
fig.savefig("Figure_updated.jpg")
```

等值线图(contour plot) 是一种显示二元函数$z = f(x, y)$在不同z值上的等值区域的图片，用等高线连接函数值相等的点。用countour方法可以创建这样的图片。ax.counter接受三个参数，分别是x向量，y向量和z矩阵(表示每个(x, y)对的乘积)：

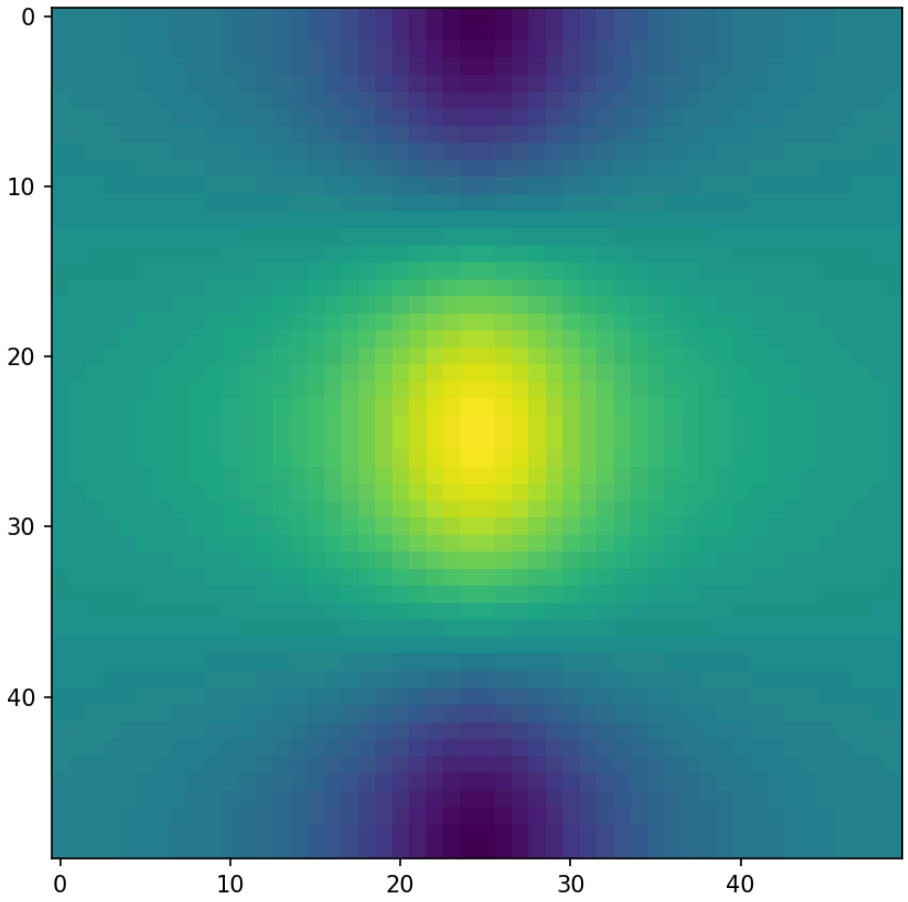
```
import matplotlib.pyplot as plt

fig, ax = plt.subplots(figsize=(8, 8))

x = np.linspace(-np.pi, np.pi, 50) #linspace(a, b, n)生成n个从a到b的等间隔值

y = x
f = np.multiply.outer(np.cos(y), 1 / (1 + x**2)) #np.multiply.outer() 计算两个数组的逐元素乘积，.outer()指定外积

ax.contour(x, y, f, levels=20)
plt.show()
```

> 外积（Outer Product）也称为张量积（Tensor Product）是一种将两个向量组合成一个新的向量或矩阵的操作。外积的结果是一个二维数组（矩阵），其元素是输入向量的元素相乘的结果



热值图(heatmap) 是一类用来表示二元函数值分布的图像。值的密度大时，颜色会更深；否则更浅：

```
fig , ax = subplots(figsize =(8, 8))
ax.imshow(f) #imshow()是用来创建热值图的命令
```



在Python中，切片的索引是一个左闭右开区间：

```
seq2 = np.arrange(0, 10)

"Hello  world"[3:6] #输出"lo  "
```

2.6.4 标引数据

2.6.4.1 数据切片

在array生成的矩阵中，索引数据的方式类似与切片。例如，在一个二维矩阵中：

```
A = np.array(np.arange(16)).reshape((4, 4))

A[1, 2] #获取第二行，第三列的数据
```

> 切片索引的第一个参数表示行，第二个表示列

传入一个列表，可以获取行元素：

```
A[[1, 3]] #获取第二行和第四行的数据

A[:, [0, 2]] #获取所有行，第一和三列的数据
```

> ':'可以表示所有的行或列
> 注意列表表示"和"而非"到"

获取某行某列的子矩阵时，直接传入列表会产生问题：

```
print(A[[1, 3], [0, 2]])
print([A[1, 0], A[3, 2]])
#以上两种方式等价，只能得到一个行向量
```

将索引更改后可以得到子矩阵：

```
A[[1 ,3]][: ,[0 ,2]] #这个方法比较复杂

idx = np.ix_([1, 3], [0, 2, 3])
A[idx] #用np.ix_()函数构建一个网格

A[1:4:2 ,0:3:2] #使用切片；第三项指定步长，连续获取元素
```

2.6.4.2 bool型索引

bool型在Numpy中等于True或False(也表示为0和1的类型)：

```
keep_rows = np.zeros(A.shape[0], bool) #创建一个与A的行数一样大的初始元素都为0的
array，表示为bool型
```

> .shape返回是一个array的属性，与其维度有关

使用"=="来判断是否等价：

```
np.all(keep_rows == np.array ([0 ,1 ,0 ,1]))
```

> 虽然二者等价，但在索引中效果不一样；前者表示第一行，第二行，第一行和第二行，后者表示第二和四行

.all()，.any()检查是否全为1和是否存在1：

```
np.all(keep_rows)
keep_rows.any()
```

通过构建bool型网格，可以快速获取某些行和列的元素：

```
keep_cols = np.zeros(A.shape[1], bool)
keep_cols[[0, 2, 3]] = True
idx_bool = np.ix_(keep_rows, keep_cols)
print(A[idx_bool])
```

> 也可以使用花式索引

2.6.5 Pandas简介

pandas使用data frame作为数据的表格，包含表头，数据和数据类型

2.6.5.1 阅读数据集

pandas使用read_csv()读取csv文件：

```
import pandas as pd

A = pd.read_csv('Auto.csv')
```

pandas读取这些文件时，默认采用','作为分隔符，如果用其他的符号例如空格作为分隔符，则需要指定分隔符：

```
Auto = pd.read_csv('Auto.data ', delim_whitespace=True) #这个方式将会逐渐被正则表达式取代

Auto = pd.read_csv('Auto.data ', sep='\s+') #改用seq参数
```

查找表头：

```
Auto['horsepower']
```

> numpy中的unique()函数可以返回array中的唯一元素，相当于求集合

na_values用于指定缺失值：

```
Auto = pd.read_csv('Auto.data ', na_values =['?'], delim_whitespace=True)

Auto['horsepower'].sum()
```

dropna()方法能直接删去缺失数据的行：

```
Auto_new = Auto.dropna()
Auto_new.shape #shape缩小
```

> 指定缺失值后，才能求数学运算

2.6.5.2 选择行与列

可以直接获取列的表头：

```
Auto = Auto_new
Auto.columns #获取columns属性
```

未指定行序号时，将会用数字表示；可以用列表头作为行序号：

```
Auto_re = Auto.set_index('name ') #set_index()方法指定行序号
```

> 用上面的方法指定行序号后，原来的列将会从columns属性中消失

.loc()方法可以用于索引：

```
rows = ['amc rebel sst','ford torino ' ]

Auto_re.loc[rows] #获取特定的行
```

iloc()方法与.loc()方法不同，直接采用整数来进行索引：

```
Auto_re.iloc [[3 ,4]]
print(Auto_re.iloc[[3, 4], [0, 2, 3]]) #传入两个列表
```

> 索引项不必唯一

2.6.5.3 更多的选择行列的方法

使用花式索引：

```
idx_80 = Auto_re['year'] > 80
print(Auto_re.loc[idx_80, ['weight', 'origin']])
```

使用lambda表达式：

```
print(Auto_re.loc[lambda df:(df['year'] > 80) & (df['mpg'] > 30), ['weight',
'origin']])
```

使用.index.contains()方法检测字符串中的特定字符串：

```
Auto_re.loc[lambda df: (df['displacement'] < 300) &
(df.index.str.contains('ford') | df.index.str.contains('datsun')), ['weight'
```

```
,'origin' ]]
```

2.6.6 循环

for循环是循环语句中最常用的循环。使用zip解包可以减少循环的嵌套层数：

```
total = 0
for value, weight in zip([2,3,19], [0.2,0.3,0.5]):
    total += weight * value
print('Weighted average is: {0}'.format(total))
```

考虑用概率函数.choice()生成随机值。choice函数的第一个列表指定可能的值，参数p指定对应值的概率，size指定数据规模：

```
import pandas as pd
import numpy as np

rng = np.random.default_rng(1)
A = rng.standard_normal((127, 5))
M = rng.choice([0, np.nan], p=[0.8, 0.2], size=A.shape)
A += M
D = pd.DataFrame(A, columns=["food", 'bar', 'pickle', 'snack', 'popcorn'])

print(D)
print(D[:3])
```

> DataFrame用columns关键字指定列表头

使用for循环，搭配字符串格式化输出可以达到很好的统计效果：

```
for col in D.columns:
    template = 'Column "{0}" has {1: .2%} missing values' #:.2%表示按小数点后两位的百分数表示
    print(template.format(col, np.isnan(D[col]).mean())) #isnan()求NaN值数量
```

2.6.7 更多图形和数据总结

matplotlib不能直接识别pandas中的表头，需要手动输入：

```
fig, ax = plt.subplots(figsize=(8, 8))
ax.plot(Auto['horsepower'], Auto['mpg'], 'o')
```

对pandas文件使用plot方法也能达到效果：

```
ax = Auto.plot.scatter('horsepower', 'mpg')
```

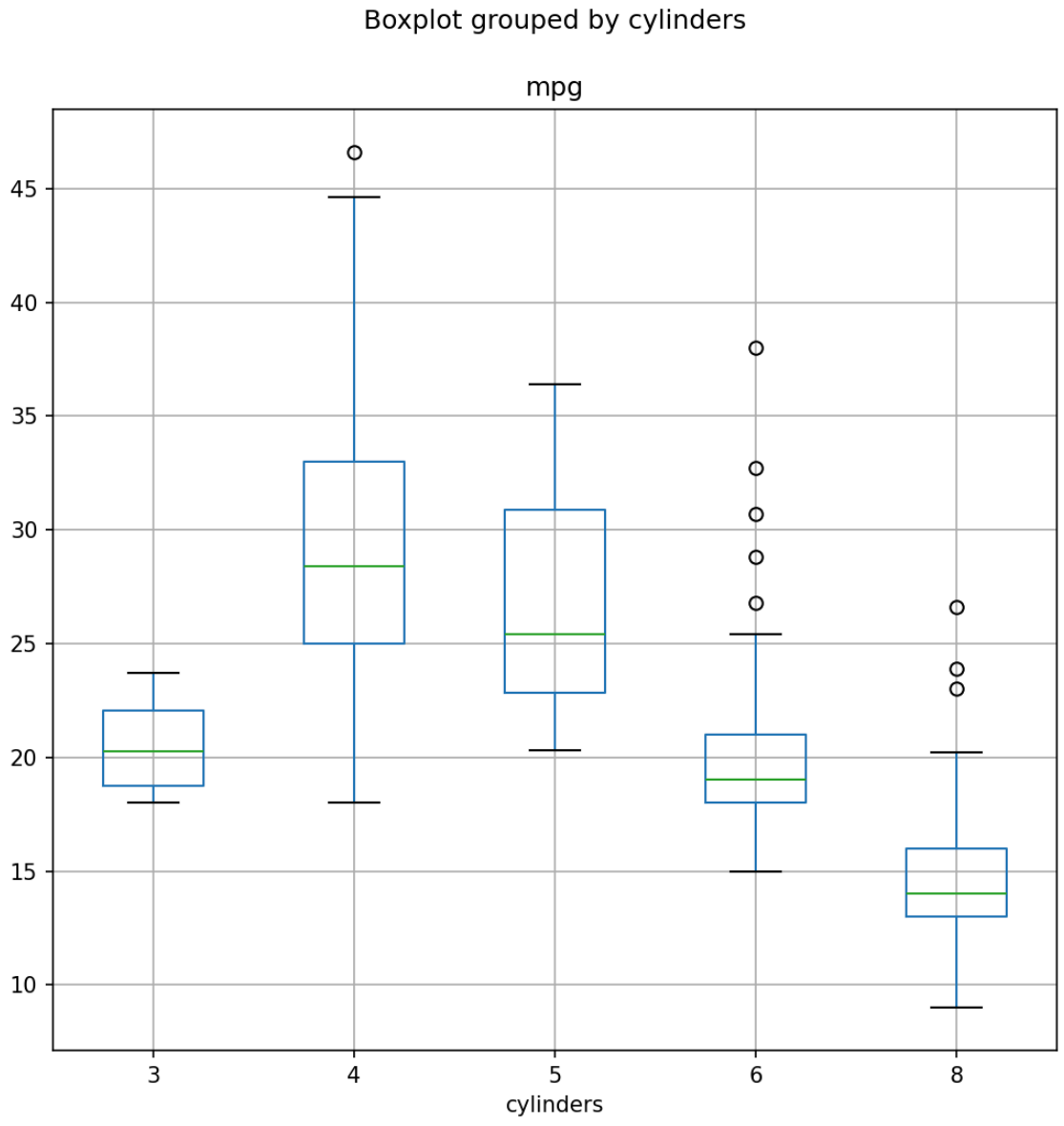
保存给定坐标轴的图形，可以通过访问图形属性实现：

```
fig = ax.figure
fig.savefig('horsepower_mpg.png')
```

用pandas数据创建多幅图像：

```
fig, axes = plt.subplots(ncols=3, figsize=(15, 5))
Auto.plot.scatter('horsepower', 'mpg', ax=axes[1])
```

箱型图(boxplot) 是一种用于展示一组数据分布特征的统计图表。它能够提供数据的最小值、第一四分位数（Q1）、中位数（Q2）、第三四分位数（Q3）和最大值的摘要信息，并且可以显示数据的异常值：



```
fig, ax = plt.subplots(figsize=(8, 8))
Auto.boxplot('mpg', by='cylinders', ax=ax) #by指定分裂组的依据(即x轴),此处表示根据不同的cylinder来进行分组，生成不同组的mpg情况；ax=ax表示使用之前生成plot的ax值
```

生成直方图：

```
fig , ax = subplots(figsize =(8, 8))
Auto.hist('mpg', color='red', bins=12, ax=ax) #bins关键字指定柱的高度
```

对pandas中的所有数据进行分析或部分数据分析：

```
pd.plotting.scatter_matrix(Auto)

pd.plotting.scatter_matrix(Auto[['mpg', 'displacement', 'weight']])

plt.show()
```

> 对角线为直方图，其他为散点图

获取一列或多列数据的分析：

```
print(Auto[['mpg', 'weight']].describe())
print(Auto['cylinders'].describe())
```

#CS
#ML