目录

[p21 2](#_Toc63242526)

[p22 3](#_Toc63242527)

[p23 4](#_Toc63242528)

[p31 5](#_Toc63242529)

[p70 6](#_Toc63242530)

[p75 7](#_Toc63242531)

[p145 8](#_Toc63242532)

[p167 9](#_Toc63242533)

[P171 10](#_Toc63242534)

[p179-180 11](#_Toc63242535)

[p194 12](#_Toc63242536)

[p246 13](#_Toc63242537)

[p248 15](#_Toc63242538)

# p21

**原文：**

【例 2‑3】

|  |  |  |
| --- | --- | --- |
|  | 1  2  3  4  5  6  7  8 | str = 'www.gduf.edu.cn'  print('str = ', str) # 输出整个字符串  print('str[0] = ',str[0]) # 输出第1个字符，索引号从0开始计算  print('str[4:7] = ',str[4:7]) # 输出第5-7个字符, 索引号从0开始计算  print('str[4:] = ',str[4:]) # 输出从第5个字符至最后一个字符  print('str[-1] = ',str[-1]) # 输出最后一个字符  print('str \* 2 = ',str \* 2) # 输出整个字符串两次  print('str + "TEST" = ',str + "TEST") # 输出两个字符串连接后的字符 |
| Out: | str = www.gduf.edu.cn  str[0] = w  str[4:7] = gdu  str[4:] = gduf.edu.cn  str[-1] = n  str \* 2 = www.gduf.edu.cnwww.gduf.edu.cn  str + "TEST" = www.gduf.edu.cnTEST | |

**勘误：**

【例 2‑3】

|  |  |  |
| --- | --- | --- |
|  | 1  2  3  4  5  6  7  8 | str = 'www.gduf.edu.cn'  print('str = ', str) # 输出整个字符串  print('str[0] = ',str[0]) # 输出第0个字符，索引号从0开始计算  print('str[4:7] = ',str[4:7]) # 输出第4、5、6字符, 索引号从0开始计算  print('str[4:] = ',str[4:]) # 输出从第4个字符至最后一个字符  print('str[-1] = ',str[-1]) # 输出最后一个字符  print('str \* 2 = ',str \* 2) # 输出整个字符串两次  print('str + "TEST" = ',str + "TEST") # 输出两个字符串连接后的字符 |
| Out: | str = www.gduf.edu.cn  str[0] = w  str[4:7] = gdu  str[4:] = gduf.edu.cn  str[-1] = n  str \* 2 = www.gduf.edu.cnwww.gduf.edu.cn  str + "TEST" = www.gduf.edu.cnTEST | |

# p22

**原文：**

【例 2‑4】

|  |  |  |
| --- | --- | --- |
|  | 1  2  3  4  5  6  7  8  9  10 | list = [ 'yes', 'no', 786 , 2.23, 'hoffman', 70.2 ]  tinylist = [100, 'hoffman']  print('list = ', list) # 输出列表的所有元素  print('list[0] = ',list[0]) # 输出列表的第1个元素，索引号从0开始  print('list[1:3] = ',list[1:3]) # 输出第2-3个元素  print('list[2:] = ',list[2:]) # 输出从第3个元素至最后一个元素  print('list[-3:-1] = ',list[-3:-1]) # 输出倒数第3个和倒数第2号元素  print('tinylist \* 2 = ',tinylist \* 2) # 输出整个列表的元素两次  print('list + tinylist = ', list + tinylist) # 输出两个列表连接后的元素 |
| Out: | list = ['yes', 'no', 786, 2.23, 'hoffman', 70.2]  list[0] = yes  list[1:3] = ['no', 786]  list[2:] = [786, 2.23, 'hoffman', 70.2]  list[-3:-1] = [2.23, 'hoffman']  tinylist \* 2 = [100, 'hoffman', 100, 'hoffman']  list + tinylist = ['yes', 'no', 786, 2.23, 'hoffman', 70.2, 100, 'hoffman'] | |

**勘误：**

【例 2‑4】

|  |  |  |
| --- | --- | --- |
|  | 1  2  3  4  5  6  7  8  9  10 | list = [ 'yes', 'no', 786 , 2.23, 'hoffman', 70.2 ]  tinylist = [100, 'hoffman']  print('list = ', list) # 输出列表的所有元素  print('list[0] = ',list[0]) # 输出列表的第0个元素，索引号从0开始  print('list[1:3] = ',list[1:3]) # 输出第1、2元素  print('list[2:] = ',list[2:]) # 输出从第2个元素至最后一个元素  print('list[-3:-1] = ',list[-3:-1]) # 输出倒数第3个和倒数第2号元素  print('tinylist \* 2 = ',tinylist \* 2) # 输出整个列表的元素两次  print('list + tinylist = ', list + tinylist) # 输出两个列表连接后的元素 |
| Out: | list = ['yes', 'no', 786, 2.23, 'hoffman', 70.2]  list[0] = yes  list[1:3] = ['no', 786]  list[2:] = [786, 2.23, 'hoffman', 70.2]  list[-3:-1] = [2.23, 'hoffman']  tinylist \* 2 = [100, 'hoffman', 100, 'hoffman']  list + tinylist = ['yes', 'no', 786, 2.23, 'hoffman', 70.2, 100, 'hoffman'] | |

# p23

**原文：**

【例 2‑5】

|  |  |  |
| --- | --- | --- |
|  | 1  2  3  4  5  6  7  8  9  10 | tuple = ( 'hoffman', 786 , 2.23, 'gduf', 70.2) #建立元组  tinytuple = (999.0, 'hoffman')  print('tuple = ', tuple) # 输出元组的所有元素  print('tuple[0] = ', tuple[0]) # 输出元组的第1个元素，索引号从0开始  print('tuple[1:3] = ', tuple[1:3]) # 输出第2个和第3元素  print('tuple[-3:-1] = ', tuple[-3:-1]) # 输出倒数第3个和倒数第2号元素  print('tuple[2:] = ', tuple[2:]) # 输出从第3个元素至最后一个元素  print('tinytuple \* 2 = ',tinytuple \* 2) # 输出元组的元素两次  print('tuple + tinytuple = ', tuple + tinytuple) # 输出两个元组连接后的元素 |
| Out: | tuple = ('hoffman', 786, 2.23, 'gduf', 70.2)  tuple[0] = hoffman  tuple[1:3] = (786, 2.23)  tuple[-3:-1] = (2.23, 'gduf')  tuple[2:] = (2.23, 'gduf', 70.2)  tinytuple \* 2 = (999.0, 'hoffman', 999.0, 'hoffman')  tuple + tinytuple = ('hoffman', 786, 2.23, 'gduf', 70.2, 999.0, 'hoffman') | |

**勘误：**

【例 2‑5】

|  |  |  |
| --- | --- | --- |
|  | 1  2  3  4  5  6  7  8  9  10 | tuple = ( 'hoffman', 786 , 2.23, 'gduf', 70.2) #建立元组  tinytuple = (999.0, 'hoffman')  print('tuple = ', tuple) # 输出元组的所有元素  print('tuple[0] = ', tuple[0]) # 输出元组的第0个元素，索引号从0开始  print('tuple[1:3] = ', tuple[1:3]) # 输出第1、2元素  print('tuple[-3:-1] = ', tuple[-3:-1]) # 输出倒数第3个和倒数第2号元素  print('tuple[2:] = ', tuple[2:]) # 输出从第3个元素至最后一个元素  print('tinytuple \* 2 = ',tinytuple \* 2) # 输出元组的元素两次  print('tuple + tinytuple = ', tuple + tinytuple) # 输出两个元组连接后的元素 |
| Out: | tuple = ('hoffman', 786, 2.23, 'gduf', 70.2)  tuple[0] = hoffman  tuple[1:3] = (786, 2.23)  tuple[-3:-1] = (2.23, 'gduf')  tuple[2:] = (2.23, 'gduf', 70.2)  tinytuple \* 2 = (999.0, 'hoffman', 999.0, 'hoffman')  tuple + tinytuple = ('hoffman', 786, 2.23, 'gduf', 70.2, 999.0, 'hoffman') | |

# p31

**原文：**

【例 2‑14】

|  |  |  |
| --- | --- | --- |
|  | 1  2  3  4  5  6  7 | i=0  while (i<=10):  i=i+1  if i%2!=0: #如果i是奇数，则continue,跳至while语句，  #不执行后面的print  continue  print(i) #注意此语句的缩进与上面的if对齐，表示其仍在while内 |
| Out: | 2  4  6  8  10 | |

**勘误：**

【例 2‑14】

|  |  |  |
| --- | --- | --- |
|  | 1  2  3  4  5  6  7 | i=0  while (i<=10):  i=i+1  if i%2!=0: #如果i是奇数，则continue,跳至while语句，  #不执行后面的print  continue  print(i) #注意此语句的缩进与上面的if对齐，表示其仍在while内 |
| Out: | 2  4  6  8  10 | |

# p70

**原文：**

|  |  |  |
| --- | --- | --- |
|  | 1  2  3  4 | print(" df4与df5的合并，按照右表的数据:")  print(' right右连接，表示按df5存在的数据做连接，df4中不存在的自动用NaN填充:')  print(pd.merge(df4,df5,how='outer')) |
| Out: | df4与df5的合并，按照右表的数据:  right右连接，表示按df5存在的数据做连接，df4中不存在的自动用NaN填充:   |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | |  | id | date | city | category | age | price | gender | m-point | pay | | 0 | 1001 | 2013-01-02 | Beijing | 100-A | 23.0 | 1200.0 | male | 10 | Y | | 1 | 1002 | 2013-01-03 | SH | 100-B | 44.0 | NaN | female | 12 | N | | 2 | 1003 | 2013-01-04 | guangzhou | 110-A | 54.0 | 2133.0 | male | 20 | Y | | 3 | 1004 | 2013-01-05 | Shenzhen | 110-C | 32.0 | 5433.0 | female | 40 | Y | | 4 | 1005 | 2013-01-06 | shanghai | 210-A | 34.0 | NaN | male | 40 | N | | 5 | 1006 | 2013-01-07 | BEIJING | 130-F | 32.0 | 4432.0 | female | 40 | Y | | 6 | 1007 | NaT | NaN | NaN | NaN | NaN | male | 30 | N | | 7 | 1008 | NaT | NaN | NaN | NaN | NaN | female | 20 | Y | | |

**勘误：**

|  |  |  |
| --- | --- | --- |
|  | 1  2  3  4 | print(" df4与df5的合并，按照右表的数据:")  print(' right右连接，表示按df5存在的数据做连接，df4中不存在的自动用NaN填充:')  print(pd.merge(df4,df5,how='right')) |
| Out: | df4与df5的合并，按照右表的数据:  right右连接，表示按df5存在的数据做连接，df4中不存在的自动用NaN填充:   |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | |  | id | date | city | category | age | price | gender | m-point | pay | | 0 | 1001 | 2013-01-02 | Beijing | 100-A | 23.0 | 1200.0 | male | 10 | Y | | 1 | 1002 | 2013-01-03 | SH | 100-B | 44.0 | NaN | female | 12 | N | | 2 | 1003 | 2013-01-04 | guangzhou | 110-A | 54.0 | 2133.0 | male | 20 | Y | | 3 | 1004 | 2013-01-05 | Shenzhen | 110-C | 32.0 | 5433.0 | female | 40 | Y | | 4 | 1005 | 2013-01-06 | shanghai | 210-A | 34.0 | NaN | male | 40 | N | | 5 | 1006 | 2013-01-07 | BEIJING | 130-F | 32.0 | 4432.0 | female | 40 | Y | | 6 | 1007 | NaT | NaN | NaN | NaN | NaN | male | 30 | N | | 7 | 1008 | NaT | NaN | NaN | NaN | NaN | female | 20 | Y | | |

# p75

**原文：**

In[22]第27行：#奇数列表项是卡号的前n位，偶数列表项是发卡行信息

**勘误：**

In[22]第27行：#偶数列表项是卡号的前n位，奇数列表项是发卡行名称

# p145

**原文：**

图8-2

**勘误：**

图8-2



# p167

**原文：**

**遗漏In[4],splitDataSet函数实现**

**勘误：**

**在原In[4]位置添加以下cell，后续序号增1**

|  |  |  |
| --- | --- | --- |
|  | 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18 | #按照feat指定的标签分割数据集  #参数：dataset-List列表，待分割样本集合  # axie-数值，属性列表项序号，按照此属性的取值来分割样本集合  # value-数值，属性取值  #返回值：retDataSet-List列表，分割后的样本集合，去除了分割属性  def splitDataSet(dataSet, axis, value):  retDataSet = []  for featVec in dataSet:  if featVec[axis] == value:  reduceFeatVec = featVec[:axis]  reduceFeatVec.extend(featVec[axis+1:])  retDataSet.append(reduceFeatVec)  return retDataSet  print('splitDataSet函数示例')  print('将原始样本分割出Income（在labels中序号为1）值为0的集合：')  retDS1 = splitDataSet(dS1,1, 0)  print(retDS1) |
| Out: | splitDataSet函数示例  将原始样本分割出Income（在labels中序号为1）值为0的集合：  [[2, 1, 0, 'Y'], [2, 1, 1, 'N'], [1, 1, 1, 'Y'], [0, 1, 0, 'Y']] | |

# P171

**原文：**

**In[7]输出内容错误**

**勘误：**

|  |  |  |
| --- | --- | --- |
|  | 1  2  3  4  5  6  7  8 | #结果显示  #createTree会改变lab,因此传入它的副本  lab\_temp = lab[:]  mTr=createTree(dS1,lab\_temp)  print('构造出的决策树：')  print(mTr)  #调用Cell1中实现的createPlot函数绘制决策树。  createPlot(mTr) |
| Out: | 构造出的决策树：  {'Age': {0: {'Job': {0: 'N', 1: 'Y'}}, 1: 'Y', 2: {'Credit': {0: 'Y', 1: 'N'}}}}    图 9‑5 决策树 | |

# p179-180

**原文：**

考虑上一章中的用户购买保险产品的后验概率：用A来表示做出购买决策的事件，P(A)表示购买的可能性，P()表示不购买的可能性；年龄、收入范围、工作性质和信用评分是用来对样本*Ω*进行划分的属性，记为Y，S，J和C（朴素贝叶斯算法假设这四个划分彼此独立，互不影响），这个例子中的每个划分都对应了两种或者三种取值。当购买的行为已经发生时，所有这四个属性的某种取值组合的后验概率是：

|  |  |
| --- | --- |
|  | (10‑1) |

以现有样本集根据公式(10‑1)计算的后验概率为基础，

**勘误：**

考虑上一章中的用户购买保险产品的先验概率：用A来表示做出购买决策的事件，P(A)表示购买的可能性，P()表示不购买的可能性；年龄、收入范围、工作性质和信用评分是用来对样本*Ω*进行划分的属性，记为Y，S，J和C（朴素贝叶斯算法假设这四个划分彼此独立，互不影响），这个例子中的每个划分都对应了两种或者三种取值。当购买的行为已经发生时，所有这四个属性的某种取值组合的先验概率是：

|  |  |
| --- | --- |
|  | (10‑1) |

以现有样本集根据公式(10‑1)计算的先验概率为基础，

# p194

**原文：**

图11-6

**勘误：**



图 11‑6 K均值聚类算法流程图

# p246

**原文：**

**In[7]22行之后代码错误**

**勘误：**

|  |  |  |
| --- | --- | --- |
|  | 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31 | #分析MEDV的数据，根据其1/4分位点的房价数据，  #将整个样本分为贵价房屋和平价房屋  #1/4分位点用于统计描述，是指所有样本中3/4中的数据小于该值  #实际数据集的1/4分位点值为17.02，以小于该值为平价房屋标准  #NormalPrice列的值为0，表示平价；值为1，表示贵价  print(bostonDF[‘MEDV’].describe())  bostonDF[‘NormalPrice’]=bostonDF[‘MEDV’].apply(lambda s:np.float(s>=17.025))  print(bostonDF[‘NormalPrice’].head())  X = bostonDF.drop(labels=[‘MEDV’,’NormalPrice’], axis=1)  y = bostonDF[‘NormalPrice’]  #随机抽取90%的样本用于训练  #10%的样本用于测试  from sklearn.model\_selection import train\_test\_split  X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.1)  #均一化数据，将每个训练数据减去其所在列的均值  #处以其所在列的方差，这样改善训练效果  from sklearn.preprocessing import StandardScaler  scaler = StandardScaler().fit(X\_train)  #引入逻辑回归模型  from sklearn import linear\_model, datasets  X\_train = pd.DataFrame(scaler.transform(X\_train), columns=X\_train.columns)  X\_test = pd.DataFrame(scaler.transform(X\_test), columns=X\_test.columns)  ExpensiveHouseLR=linear\_model.LogisticRegression()  ExpensiveHouseLR.fit(X\_train, y\_train)  print("系数:",ExpensiveHouseLR.coef\_)  print("截距:",ExpensiveHouseLR.intercept\_) |
| Out: | count 506.000000  mean 22.532806  std 9.197104  min 5.000000  25% 17.025000  50% 21.200000  75% 25.000000  max 50.000000  Name: MEDV, dtype: float64  0 1.0  1 1.0  2 1.0  3 1.0  4 1.0  Name: NormalPrice, dtype: float64  系数: [[-0.92164569 0.65126997 0.51493293 0.13653517 -1.21525748 -0.09805405  -0.92736533 -0.88089149 0.91338015 -0.72460478 -0.7381216 0.47520731  -1.76043042]]  截距: [2.73826373] | |

# p248

**原文：**

**In[8]输出内容错误**

**勘误：**

|  |  |
| --- | --- |
| Out: | R值(准确率): 0.9340659340659341    图 14‑4 波士顿房价逻辑回归分类结果 |