

一、泰坦尼克号生存预测分析

1. 项目步骤
2. 描述性统计分析概念
3. 探索性统计分析概念
4. 泰坦尼克号数据描述性分析
 - (1)、DataFrame的plot绘图方法
代码练习：DataFrame_plot.ipynb
 - (2)、数据描述性统计分析
代码位置：Titanic_analysis.ipynb
 - (3)、数据探索性分析

二、pandas数据预处理

1. 合并数据
 - (1)、横向堆叠合并数据
 - (2)、纵向堆叠合并数据
代码练习 concat_append.ipynb
2. 清洗数据
 - (1)、检测与处理缺失值
 - 1、删除法
 - 2、替换法
代码练习：fillna_demo.ipynb
3. 标准化数据
 - (1)、离差标准化
 - (2)、标准差标准化
代码练习：normalization_demo.ipynb
4. 转换数据
 - (1)、哑变量处理类别数据
代码练习：dummies.ipynb
 - (2)、离散化连续型数据
5. 数据规约
 - (1)、属性规约
 - (2)、数值规约
6. 泰坦尼克号数据预处理

一、泰坦尼克号生存预测分析

大家都熟悉的Jack and Rose的故事，豪华游艇倒了，大家都惊恐逃生，可是救生艇的数量有限，副船长说lady and kid first！，所以是否获救其实并非随机，而是基于一些背景有先后顺序的。

今日目标：对数据进行描述性分析，探索性分析，构建特征，数据预处理

后续：训练和测试数据是一些乘客的个人信息以及存活状况，要尝试根据它生成合适的模型并预测其他人的存活状况。

1. 项目步骤

- 简单分析方法：（现状分析或问题定位）掌握数理统计等知识完成基础的业务数据分析，提取有价值的信息，形成有效的结论。比如：描述性统计分析，探索性数据分析
- 深层业务逻辑建模分析：（未来数据预测）使用数据挖掘算法中的分类、聚类、回归等方法搭建模型，分析完成复杂的数据分析工作，重点挖掘数据价值，寻找模式与规律。

本项目步骤

1. 读取数据
2. 数据的简单描述性分析
3. 通过可视化的方式深入了解数据中的特征
4. 查看每一个属性与获救情况的可视化，找到与结果有关的特征
5. 数据预处理：包含缺失值处理，one-hot处理，标准化处理

2. 描述性统计分析概念

描述性统计分析：描述性统计分析要对调查总体所有变量的有关数据做统计性描述，主要包括数据的集中趋势分析、数据离散程度分析、数据的频数分析

1. 集中趋势的描述性统计量

- 均值：是指一组数据的算术平均数，描述一组数据的平均水平，是集中趋势中波动最小、最可靠的指标，但是均值容易受到极端值（极小值或极大值）的影响。
- 中位数：是指当一组数据按照顺序排列后，位于中间位置的数，不受极端值的影响，对于定序型变量，中位数是最适合的表征集中趋势的指标。
- 众数：是指一组数据中出现次数最多的观测值，不受极端值的影响，常用于描述定性数据的集中趋势

2. 离散程度的描述性统计量

- 最大值和最小值：是一组数据中的最大观测值和最小观测值
- 极差：又称全距，是一组数据中的最大观测值和最小观测值之差，记作R，一般情况下，极差越大，离散程度越大，其值容易受到极端值的影响。
- 方差和标准差：是描述一组数据离散程度的最常用、最适用的指标，值越大，表明数据的离散程度越大。

3. 频数分析

- 对列进行频数分布分析，需要区分数值型数据和类别型数据的分析方式 频数分布分析（又称频率分析）主要通过频数分布表、条形图和直方图、百分位数等来描述数据的分布特征。

3. 探索性统计分析概念

1. 探索性分析由约翰·图基(john Tukey)在20世纪70年代开发，经常被描述为一种哲学，对于如何进行分析没有硬性规定，在统计学上，EDA是指通过分析数据，来总结数据主要特征的方法，可视化方法是使用较多的方式
2. 这个过程并不一定需要统计模型，EDA的主要目的是在尽量少的先验假设下，心无杂念的让数据告诉我们一切，使用可视化和定量方法来了解数据所讲述的故事，在其中寻找线索、逻辑、问题或研究领域等线索。
3. 主要目的：
 - 1、寻找数据特征之间的关系、鉴别特征之间有趣或者意想不到的关系
 - 2、分析并希望找出与结果有关的特征，进而完成特征工程的构建
 - 3、是否需要更多数据做数据分析的支撑

4. 泰坦尼克号数据描述性分析

本次目的：对泰坦尼克号训练集数据进行描述性分析

主要包含：读取数据并对列进行分析

1. 获救情况人数可视化
2. 乘客等级分布可视化

3. 各登船口岸上船人数可视化
4. 性别分布

(1)、DataFrame的plot绘图方法

```
 DataFrame.plot(x=None, y=None, kind='line', ax=None, subplots=False,
    sharex=None, sharey=False, layout=None, figsize=None,
    use_index=True, title=None, grid=None, legend=True,
    style=None, logx=False, logy=False, loglog=False,
    xticks=None, yticks=None, xlim=None, ylim=None, rot=None,
    xerr=None, secondary_y=False, sort_columns=False, **kwds)
```

使用DataFrame的plot方法绘制图像会按照数据的每一列绘制一条曲线，参数中的columns就是列的名称而index本来是DataFrame的行名称。图形绘制成功之后还会按照列的名称绘制图例

参考链接：<https://blog.csdn.net/brucewong0516/article/details/80524442>这个功能确实是比较赞的。如果使用matplotlib的基本绘制功能，图例的添加还需要自己额外处理。看来，数据的规整化不仅仅是为了向量化以及计算加速做准备，而且为数据的可视化提供了不少便捷的方法。

kind参数：

- 'line' : line plot (default) #折线图
- 'bar' : vertical bar plot #柱状图
- 'barh' : horizontal bar plot #横向条形图
- 'hist' : histogram #直方图
- 'box' : boxplot #箱线图
- 'kde' : Kernel Density Estimation 密度估计图
- 'pie' : pie plot #饼图
- 'scatter' : scatter plot #散点图

代码练习：DataFrame_plot.ipynb

```
1 import numpy as np
2 import pandas as pd
3 import matplotlib.pyplot as plt
4 np.random.seed(123)
5 df=pd.DataFrame(np.random.randn(3,4),index=[1,2,3],columns=list('MNJK'))
6 df
7 df.plot()
8 # 同时画多个子图，可以设置 subplots = True
9 df.plot(subplots=True,figsize=(6,6))
10 fig=plt.figure(figsize=(10,6))
11 ax1=fig.add_subplot(2,1,1)
12 df.plot('M','J',ax=ax1,grid=True,style='-' ,title='random',xticks=range(4),yticks=range(4),rot=20,kind='line')
13 ax2=fig.add_subplot(2,1,2)
14 df.plot('N','K',ax=ax2,grid=True,xticks=range(5),yticks=range(5),rot=20,kind='scatter')
```

(2)、数据描述性统计分析

代码目标：对数据进行读取并做描述性分析

代码位置 : Titanic_analysis.ipynb

```
1 #一: 读取数据
2 # 导入环境库
3 import numpy as np
4 import pandas as pd
5 import matplotlib.pyplot as plt
6 # 正常显示中文
7 plt.rcParams['font.sans-serif']=['SimHei']
8 # 正常显示符号
9 plt.rcParams['axes.unicode_minus']=False
10 data_train=pd.read_csv('data/train.csv')
11 data_train
12 #二: 对数据做描述性统计分析
13 # 完整性分析
14 data_train.info()
15 # 数值型数据描述分析
16 data_train.describe()
17 # 类别型数据描述分析
18 # data_train.select_dtypes('object').describe().T
19 #1、获救情况人数可视化
20 data_Survived=data_train.Survived.value_counts()
21 data_Survived
22 data_train.Survived.value_counts().plot(kind='bar',
23                                         rot=0,
24                                         figsize=(6,6),
25                                         fontsize=18,
26                                         color=['b','r'])
27 plt.title('获救情况人数可视化',fontsize=20)
28 plt.xlabel('获救情况',fontsize=15)
29 plt.ylabel('人数',fontsize=15)
30 for a,b in zip(data_Survived.index,data_Survived.values):
31     plt.text(a,b,'%.0f'%b,ha='center',va='bottom',fontsize=15)
32 # 2、各登船口岸人数可视化
33 data_train.Embarked.value_counts()
34 data_train.Embarked.value_counts().plot(kind='bar',
35                                         rot=0,
36                                         figsize=(6,6),
37                                         fontsize=18)
38 plt.title('登船口岸人数可视化',fontsize=20)
39 # 3、乘客等级人数可视化
40 data_train.Pclass.value_counts()
41 data_train.Pclass.value_counts().plot(kind='bar',
42                                         rot=0,
43                                         figsize=(6,6),
44                                         fontsize=18)
45 plt.title('乘客等级人数可视化',fontsize=20)
46 # 4、性别分布
47 data_train.Sex.value_counts()
48 data_train.Sex.value_counts().plot(kind='bar',
49                                         rot=0,
50                                         figsize=(6,6),
51                                         fontsize=18)
52 plt.title('性别分布可视化',fontsize=20)
```

(3)、数据探索性分析

1. 各乘客等级的获救情况
2. 看看各性别的获救情况
3. 查看各登船港口的获救情况
4. 堂兄弟和父母字段对于获救情况分析
5. 分析cabin这个值的有无，对于survival的分布状况
6. 分析票价

```

1 #1、分析各等级乘客年龄分布情况
2 plt.figure(figsize=(8,6))
3 data_train.Age[data_train.Pclass==1].plot(kind='kde', fontsize=15)
4 data_train.Age[data_train.Pclass==2].plot(kind='kde', fontsize=15)
5 data_train.Age[data_train.Pclass==3].plot(kind='kde', fontsize=15)
6 plt.xlabel('年龄', fontsize=15)
7 plt.ylabel('密度', fontsize=15)
8 plt.title('各等级乘客年龄分布情况', fontsize=18)
9 plt.legend(['一等舱', '二等舱', '三等舱'], fontsize=14)
10 # 查看各乘客等级的获救情况
11 # 未获救情况统计
12 survived_0=data_train.Pclass[data_train.
    Survived==0].value_counts()
13 # 获救情况统计
14 survived_1=data_train.Pclass[data_train.
    Survived==1].value_counts()
15 # survived_1
16 df_pcclas=pd.DataFrame({'获救':survived_1,
    df_pcclas.plot(kind='bar', fontsize=15, rot=0, figsize=(8,6))
18 plt.xlabel('乘客等级', fontsize=15)
19 plt.ylabel('人数', fontsize=15)
20 plt.title('各乘客等级的获救情况分析', fontsize=18)
21         #'未获救':survived_0})
22 df_pcclas
23 df_pcclas.plot(kind='bar', fontsize=15, rot=0, figsize=(8,6))
24 plt.xlabel('乘客等级', fontsize=15)
25 plt.ylabel('人数', fontsize=15)
26 plt.title('各乘客等级的获救情况分析', fontsize=18)
27 # 查看各性别的获救情况
28 survived_0=data_train.Sex[data_train.Survived==0].value_counts()
29 survived_0
30 survived_1=data_train.Sex[data_train.survived==1].value_counts()
31 survived_1
32 df_Sex=pd.DataFrame({'获救':survived_1, '未获救':survived_0})
33 df_Sex
34 df_Sex.plot(kind='bar', rot=0, fontsize=15, figsize=(8,6))
35 rate_sex=df_Sex['获救']/(df_Sex['获救']+df_Sex['未获救'])
36 rate_sex=np.round(rate_sex, 2)
37 df_Sex['rate_sex']=rate_sex
38 df_Sex
39 # 查看各登船港口的获救情况
40 survived_0=data_train.Embarked[data_train.
    Survived==0].value_counts()
41 survived_0
42 survived_1=data_train.Embarked[data_train.
    Survived==1].value_counts()
43 survived_1
44 df_Embarked=pd.DataFrame({'获救':survived_1, '未获救':survived_0})
45 df_Embarked
46 rate_Embarked=df_Embarked['获救']/(df_Embarked['获救']+df_Embarked['未获救'])
47
48

```

```
49 rate_Embarked=np.round(rate_Embarked,2)
50 df_Embarked['rate_Embarked']=rate_Embarked
51 df_Embarked
52 # 堂兄弟和父母字段对于获救情况分析
53 # (1) 有无兄弟姐妹/父母子女和存活与否的关系
54 # 根据题目要求, 分四种情况(有无兄弟, 有无父母)计算存活率。
55 # 先进行数据筛选。
56 sib_data = data_train[data_train['SibSp'] != 0]
57 # sib_data
58 nosib_data = data_train[data_train['SibSp'] == 0]
59 par_data = data_train[data_train['Parch'] != 0]
60 nopal_data = data_train[data_train['Parch'] == 0]
61 sib_data['Survived'].value_counts()
62 fig=plt.figure(figsize = (16,4))
63 # #有无兄弟姐妹的存活率对比
64 ax1=fig.add_subplot(141)
65 plt.axis('equal')
66 sib_data['Survived'].value_counts().plot(kind='pie',labels = ['No
Survived','Survived'],autopct = '%.2f%%',colormap = 'Blues')
67 plt.xlabel('SibSp')
68
69 ax2=fig.add_subplot(142)
70 plt.axis('equal')
71 nosib_data['Survived'].value_counts().plot.pie(labels = ['No
Survived','Survived'],autopct = '%.2f%%',colormap = 'Blues')
72 plt.xlabel('no_SibSp')
73 #有无父母子女的存活率对比
74 ax3=fig.add_subplot(143)
75 plt.axis('equal')
76 par_data['Survived'].value_counts().plot.pie(labels = ['No
Survived','Survived'],autopct = '%.2f%%',colormap = 'Reds')
77 plt.xlabel('Parch')
78
79 ax4=fig.add_subplot(144)
80 plt.axis('equal')
81 nopal_data['Survived'].value_counts().plot.pie(labels = ['No
Survived','Survived'],autopct = '%.2f%%',colormap = 'Reds')
82 plt.xlabel('no_Parch')
83 # cabin有没有值对于获救情况的影响
84 # 有cabin记录的数据
85 survived_cabin=data_train.Survived[pd.notnull(data_train.Cabin)].value_coun
ts()
86 # 没有cabin记录的数据
87 survived_nocabin=data_train.Survived[pd.isnull(data_train.Cabin)].value_cou
nts()
88 survived_nocabin
89 df_cabin=pd.DataFrame({'有':survived_cabin,'无':survived_nocabin}).T
90 df_cabin
91 rate_cabin=df_cabin[1]/(df_cabin[0]+df_cabin[1])
92 rate_cabin=np.round(rate_cabin,2)
93 df_cabin['rate_cabin']=rate_cabin
94 df_cabin
95 # 分析票价
96 fare_die = data_train['Fare'][data_train['Survived'] == 0]
97 fare_sur = data_train['Fare'][data_train['Survived'] == 1]
98 fare_die.describe()
99 fare_sur.describe()
```

二、pandas数据预处理

1. 数据缺失值处理：age , Cabin , Embarked -----数据清洗

2. 数据one-hot处理：逻辑回归算法需要传入的数据是数值型-----转换数据

3. 数据标准化处理：对数据进行标准化处理-----标准化数据

1. 合并数据

(1)、横向堆叠合并数据

当axis=1的时候，concat做横向合并

当两张表完全一样时，不论join参数取值是inner或者outer，结果都是将两个表完全按照X轴拼接起来。

	df1				df4			Result							
	A	B	C	D	B	D	F	A	B	C	D	B	D	F	
0	A0	B0	C0	D0	2	B2	D2	F2	0	A0	B0	C0	D0	NaN	NaN
1	A1	B1	C1	D1	3	B3	D3	F3	1	A1	B1	C1	D1	NaN	NaN
2	A2	B2	C2	D2	6	B6	D6	F6	2	A2	B2	C2	D2	B2	F2
3	A3	B3	C3	D3	7	B7	D7	F7	3	A3	B3	C3	D3	B3	F3
								6	NaN	NaN	NaN	NaN	B6	F6	
								7	NaN	NaN	NaN	NaN	B7	F7	

	df1				df4			Result							
	A	B	C	D	B	D	F	A	B	C	D	B	D	F	
0	A0	B0	C0	D0	2	B2	D2	F2	2	A2	B2	C2	D2	B2	F2
1	A1	B1	C1	D1	3	B3	D3	F3	3	A3	B3	C3	D3	B3	F3
2	A2	B2	C2	D2	6	B6	D6	F6							
3	A3	B3	C3	D3	7	B7	D7	F7							

1. 横向表堆叠

横向堆叠，即将两个表在X轴向拼接在一起，可以使用concat函数完成，concat函数的基本语法如下：

参数名称	说明
objs	接收多个Series , DataFrame的组合。表示参与连接的pandas对象的列表

参数名称	说明
axis	接收0或1。表示连接的轴向，默认为0。
join	接收inner或outer。表示其他轴向上的索引是按交集（inner）还是并集（outer）进行合并。默认为outer。
join_axes	接收Index对象。表示用于其他n-1条轴的索引，不执行并集 / 交集运算。
ignore_index	接收boolean。表示是否不保留连接轴上的索引，产生一组新索引range(total_length)。默认为False。
keys	接收sequence。使用传递的键作为最外层来构造层次索引。加上一个层次的key来识别数据源自于哪张表，区分不同的表数据来源，默认为None。
names	接收list。生成的层次结构索引中的级别的名称。默认为None。
verify_integrity	接收boolearn。表示是否检查结果对象新轴上的重复情况，如果发现则引发异常。默认为False。

(2)、纵向堆叠合并数据

当axis=0的时候，concat做纵向合并

	a	b	c	d
1	0.0	0.0	0.0	0.0
2	0.0	0.0	0.0	0.0
3	0.0	0.0	0.0	0.0
	b	c	d	e
2	1.0	1.0	1.0	1.0
3	1.0	1.0	1.0	1.0
4	1.0	1.0	1.0	1.0

	a	b	c	d	e
1	0.0	0.0	0.0	0.0	NaN
2	0.0	0.0	0.0	0.0	NaN
3	0.0	0.0	0.0	0.0	NaN
2	NaN	1.0	1.0	1.0	1.0
3	NaN	1.0	1.0	1.0	1.0
4	NaN	1.0	1.0	1.0	1.0

	b	c	d
0	0.0	0.0	0.0
1	0.0	0.0	0.0
2	0.0	0.0	0.0
3	1.0	1.0	1.0
4	1.0	1.0	1.0
5	1.0	1.0	1.0

纵向堆叠——append方法

append方法也可以用于纵向合并两张表。但是append方法实现纵向表堆叠有一个前提条件，那就是两张表的列名需要完全一致。append方法的基本语法如下

参数名称	说明
other	接收DataFrame或Series。表示要添加的新数据。无默认。
ignore_index	接收boolean。如果输入True，会对新生成的DataFrame使用新的索引（自动产生）而忽略原来数据的索引。默认为False。
verify_integrity	接收boolean。如果输入True，那么当ignore_index为False时，会检查添加的数据索引是否冲突，如果冲突，则会添加失败。默认为False。

代码练习 concat_append.ipynb

```

1 import numpy as np
2 import pandas as pd
3 #定义资料集
4 df1 = pd.DataFrame(np.ones((3,4))0, columns=['a', 'b', 'c', 'd'])
5 df2 = pd.DataFrame(np.ones((3,4))1, columns=['a', 'b', 'c', 'd'])
6 df3 = pd.DataFrame(np.ones((3,4))2, columns=['a', 'b', 'c', 'd'])
7 print(df1)

```

```

8 print(df2)
9 print(df3)
10 # concat基本参数练习
11 #res接收 concat纵向合并 注意参数axis合并方向
12 # res=pd.concat(objs=[df1,df2,df3],axis=0)
13 # res
14 #承上一个例子，并将并将ignore_index设定为True: 重置index
15 res=pd.concat(objs=[df1,df2,df3],axis=0,ignore_index=True)
16 res
17 #concat_join参数的练习
18 # join (合并方式) 如果为'inner'得到的是两表的交集,如果是outer, 得到的是两表的并集
19 #定义资料集
20 df1 = pd.DataFrame(np.ones((3,4))0,columns=['a','b','c','d'], index=[1,2,3])
21 df2 = pd.DataFrame(np.ones((3,4))1,columns=['b','c','d','e'], index=[2,3,4])
22 # join (合并方式) 如果为'inner'得到的是两表的交集,如果是outer, 得到的是两表的并集
23 res=pd.concat(objs=[df1,df2],axis=0,join='outer',sort=True,keys=['df1','df2'],names=['df_name','row_id'])
24 res
25 # 内合并
26 res=pd.concat(objs=[df1,df2],axis=0,join='inner',ignore_index=True)
27 res
28 # append合并案例练习
29 # append合并
30 #定义资料集
31 df1 = pd.DataFrame(np.ones((3,4))0, columns=['a','b','c','d'])
32 df2 = pd.DataFrame(np.ones((3,4))1, columns=['a','b','c','d'])
33 #要求将df2合并到df1的下面, 以及重置index, 并打印出结果
34 df1.append(df2,ignore_index=True)

```

2. 清洗数据

(1)、检测与处理缺失值

- 利用isnull或notnull找到缺失值
- 数据中的某个或某些特征的值是不完整的，这些值称为缺失值。
- pandas提供了识别缺失值的方法isnull以及识别非缺失值的方法notnull，这两种方法在使用时返回的都是布尔值True和False。
- 结合sum函数和isnull、notnull函数，可以检测数据中缺失值的分布以及数据中一共含有多少缺失值。isnull和notnull之间结果正好相反，因此使用其中任意一个都可以判断出数据中缺失值的位置。

1、删除法

删除法分为删除观测记录和删除特征两种，它属于利用减少样本量来换取信息完整度的一种方法，是一种最简单的缺失值处理方法。

pandas中提供了简便的删除缺失值的方法dropna，该方法既可以删除观测记录，亦可以删除特征。
pandas.DataFrame.dropna(self, axis=0, how='any', thresh=None, subset=None, inplace=False)

参数名称	说明
axis	接收0或1。表示轴向，0为删除观测记录（行），1为删除特征（列）。默认为0。
how	接收特定string。表示删除的形式。any表示只要有缺失值存在就执行删除操作。all表示当且仅当全部为缺失值时执行删除操作。默认为any。
subset	接收类array数据。表示进行去重的列行。默认为None，表示所有列/行。
inplace	接收boolean。表示是否在原表上进行操作。默认为False。

2、替换法

- 替换法是指用一个特定的值替换缺失值。
- 特征可分为数值型和类别型，两者出现缺失值时的处理方法也是不同的。
- 缺失值所在特征为数值型时，通常利用其均值、中位数和众数等描述其集中趋势的统计量来代替缺失值。
- 缺失值所在特征为类别型时，则选择使用众数来替换缺失值

插补方法	方法描述
均值 / 中位数 / 众数插补	根据属性值的类型，用该属性取值的平均数 / 中位数 / 众数进行插补
使用固定值	将缺失的属性值用一个常量替换。如广州一个工厂普通外来务工人员的“基本工资”属性的空缺值可以用2015年广州市普通外来务工人员工资标准1895元/月，该方法就是使用固定值
最近邻插补	在记录中找到与缺失样本最接近的样本的该属性值插补
回归方法	对带有缺失值的变量，根据已有数据和与其有关的其他变量（因变量）的数据建立拟合模型来预测缺失的属性值
插值法	插值法是利用已知点建立合适的插值函数 $f(x)$ ，未知值由对应点 x_i 求出的函数值 $f(x_i)$ 近似代替

pandas库中提供了缺失值替换的方法名为fillna，其基本语法如下。

pandas.DataFrame.fillna(value=None, method=None, axis=None, inplace=False, limit=None)

参数名称	说明
value	接收scalar, dict, Series或者DataFrame。表示用来替换缺失值的值。无默认。
method	接收特定string。backfill或bfill表示使用下一个非缺失值填补缺失值。pad或ffill表示使用上一个非缺失值填补缺失值。默认为None。
axis	接收0或1。表示轴向。默认为1。
inplace	接收boolean。表示是否在原表上进行操作。默认为False。
limit	接收int。表示填补缺失值个数上限，超过则不进行填补。默认为None。

代码练习：fillna_demo.ipynb

```

1 import pandas as pd
2 import numpy as np
3 # 处理丢失数据
4 dates = pd.date_range('20130101', periods=6)

```

```

5 df = pd.DataFrame(np.arange(24).reshape((6,4)),index=dates, columns=['A','B','C','D'])
6 df.iloc[0,1] = np.nan
7 df.iloc[1,2] = np.nan
8 # df1接收，去掉有 NaN 的行，可以使用 dropna 一旦出现NaN就去除整行
9 df1=df.dropna(axis=0,how='any')
10 df1
11 # df2接收，如果是将 NaN 的值用其他值代替，比如代替成 0：
12 df2=df.fillna(value=0)
13 df2
14 # b列数据，填补平均值
15 df['B']=df['B'].fillna(df['B'].mean())
16 df
17 # c列数据，填补平均值
18 df['C']=df['C'].fillna(df['C'].mean())
19 df
20 # df3接收，判断是否有缺失数据 NaN，为 True 表示缺失数据：
21 df3=df.isnull()
22 df3
23 #每一列的缺失数据
24 # df4=df.isnull().sum(axis=0)
25 # df4
26 #每一行的缺失数据
27 df5=df.isnull().sum(axis=1)
28 df5
29 #整个表的缺失数据
30 df5.sum()

```

3. 标准化数据

(1)、离差标准化

离差标准化是对原始数据的一种线性变换，结果是将原始数据的数值映射到[0,1]区间之间，转换公式为

$$X^* = \frac{X - \min}{\max - \min}$$

数据的整

其中 \max 为样本数据的最大值， \min 为样本数据的最小值， $\max - \min$ 为极差。离差标准化保留了原始数据值之间的联系，是消除量纲和数据取值范围影响最简单的方法。

体分布情况并不会随离差标准化而发生改变，原先取值较大的数据，在做完离差标准化后的值依旧较大。

当数据和最小值相等的时候，通过离差标准化可以发现数据变为0。

(2)、标准差标准化

标准差标准化也叫零均值标准化或分数标准化，是当前使用最广泛的数据标准化方法。经过该方法处理的数据均值为0，标准差为1，转化公式如下。

$$X^* = \frac{X - \bar{X}}{\delta}$$

其中 \bar{X} 为原始数据的均值， δ 为原始数据的标准差。标准差标准化后的值区间不局限于[0,1]，并且存在负值。同时也不难发现，标准差标准化和离差标准化一样不会改变数据的分布情况。

代码练习：normalization_demo.ipynb

```
1 import numpy as np
2 import pandas as pd
3 np.random.seed(1)
4 df = pd.DataFrame(np.random.randn(4, 4) * 4 + 3)
5 df
6 # 离差标准化
7 df_norm=(df-df.min())/(df.max()-df.min())
8 df_norm
9 # 标准差标准化
10 df_norm1=(df-df.mean())/(df.std())
11 df_norm1
```

4. 转换数据

(1)、哑变量处理类别数据

数据分析模型中有相当一部分的算法模型都要求输入的特征为数值型，但实际数据中特征的类型不一定只有数值型，还会存在相当一部分的类别型，这部分的特征需要经过哑变量处理才可以放入模型之中。哑变量处理的原理示例如图

哑变量处理前		哑变量处理后				
	城市	城市_广州	城市_上海	城市_杭州	城市_北京	城市_深圳
1	广州	1	0	0	0	0
2	上海	0	1	0	0	0
3	杭州	0	0	1	0	0
4	北京	0	0	0	1	0
5	深圳	0	0	0	0	1
6	北京	0	0	0	1	0
7	上海	0	1	0	0	0
8	杭州	0	0	1	0	0
9	广州	1	0	0	0	0
10	深圳	0	0	0	0	1

Python中可以利用pandas库中的get_dummies函数对类别型特征进行哑变量处理。

```
pandas.get_dummies(data, prefix=None, prefix_sep='_', dummy_na=False, columns=None)
```

参数名称	说明
data	接收array、DataFrame或者Series。表示需要哑变量处理的数据。无默认。
prefix_sep	接收string。表示前缀的连接符。默认为'_'。
dummy_na	接收boolean。表示是否为Nan值添加一列。默认为False。
columns	接收类似list的数据。表示DataFrame中需要编码的列名。默认为None，表示对所有object和category类型进行编码。
prefix	接收string、string的列表或者string的dict。表示哑变量化后列名的前缀。默认为None。

哑变量处理特点

虚拟变量，也叫哑变量和离散特征编码，可用来表示分类变量、非数量因素可能产生的影响。

离散特征的编码分为两种情况：

- 离散特征的取值之间没有大小的意义，比如color : [red,green]，那么就使用one-hot编码
- 离散特征的取值有大小的意义，比如size:[X,XL,XXL]，那么就使用数值的映射{X:1,XL:2,XXL:3}
- 对于一个类别型特征，若其取值有m个，则经过哑变量处理后就变成了m个二元特征，并且这些特征互斥，每次只有一个激活，这使得数据变得稀疏。
- 对类别型特征进行哑变量处理主要解决了部分算法模型无法处理类别型数据的问题，这在一定程度上起到了扩充特征的作用。由于数据变成了稀疏矩阵的形式，因此也加速了算法模型的运算速度。

代码练习：dummies.ipynb

```
1 import pandas as pd
2 df = pd.DataFrame([
3     ['green', 'M', '10.2', 'class1'],
4     ['red', 'L', '13.5', 'class2'],
5     ['blue', 'XL', '15.3', 'class1'],
6 ])
7 df.columns = ['color', 'size', 'prize', 'class_label']
8 df
9 df.info()
10 size_mapping={'XL':3,'L':2,'M':1}
11 df['size']=df['size'].map(size_mapping)
12 df
13 class_mapping={label:idx for idx,label in enumerate(set(df['class_label']))}
14 class_mapping
15 df['class_label']=df['class_label'].map(class_mapping)
16 df
17 pd.get_dummies(df)
```

知识点补充：enumerate() 函数用于将一个可遍历的数据对象(如列表、元组或字符串)组合为一个索引序列，同时列出数据和数据下标，一般用在 for 循环当中。

代码练习：enumerate_test.ipynb

```
1 seasons=['Spring','Summer','Fall','Winter']
2 list(enumerate(seasons))
3 list(enumerate(seasons,start=2))
4 # 普通for循环
5 i=0
6 seq=['one','two','three']
7 for element in seq:
8     print (i,seq[i])
9     i+=1
10 seq=['one','two','three']
11 for i,element in enumerate(seq):
12     print(i,element)
```

(2)、离散化连续型数据

- 某些模型算法，特别是某些分类算法如ID3决策树算法和Apriori算法等，要求数据是离散的，此时就需要将连续型特征（数值型）转换成离散型特征（类别型）。
- 连续特征的离散化就是在数据的取值范围内设定若干个离散的划分点，将取值范围划分为一些离散化的区间，最后用不同的符号或整数值代表落在每个子区间中的数据值。

- 因此离散化涉及两个子任务，即确定分类数以及如何将连续型数据映射到这些类别型数据上。其原理如图。

离散化处理前		离散化处理后	
	年龄		年龄
1	18	1	(17.955, 27]
2	23	2	(17.955, 27]
3	35	3	(27, 36]
4	54	4	(45, 54]
5	42	5	(36, 45]
6	21	6	(17.955, 27]
7	60	7	(54, 63]
8	63	8	(54, 63]
9	41	9	(36, 45]
10	38	10	(36, 45]

将数据的值域分成具有相同宽度的区间，区间的个数由数据本身的特点决定或者用户指定，与制作频率分布表类似。pandas提供了cut函数，可以进行连续型数据的等宽离散化，其基础语法格式如下。
`pandas.cut(x, bins, right=True, labels=None, retbins=False)`

参数名称	说明
x	接收数组或Series。代表需要进行离散化处理的数据。无默认。
bins	接收int , list , array , tuple。若为int , 代表离散化后的类别数目 ; 若为序列类型的数据 , 则表示进行切分的区间 , 每两个数间隔为一个区间。无默认。
right	接收boolean。代表右侧是否为闭区间。默认为True。
labels	代表离散化后的各个类别名称
retbins	接收boolean。代表是否返回区间标签。默认为False。

```

1 import pandas as pd
2 ages=[20,22,34,56,34,12,9,43,23,18,45]
3 bins=[5,10,15,20,25,30,40,45,50,60]
4 cut_data=pd.cut(x=ages,bins=bins)
5 cut_data
6 cut_data.value_counts().plot(kind='barh')

```

5. 数据规约

分为属性规约和数值规约

在大数据集上进行复杂的数据分析和挖掘需要很长的时间 , 数据规约产生更小但保持数据完整性的新数据集 , 在规约后的数据集上进行分析和挖掘将更有效率

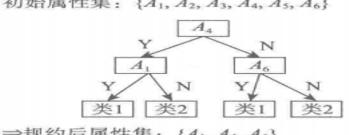
意义在于 :

- 降低无效 , 错误数据对建模的影响 , 提高建模的准确性
- 少量且具有代表性的数据将大幅缩减数据挖掘所用的时间
- 减低储存数据成本

(1)、属性规约

通过属性合并来创建新属性维数 , 或者直接删除不相关的属性来减少数据维数 , 从而提高数据分析与挖掘的速度 , 目的是寻找最小的属性子集并确保新数据子集的概率分布尽可能的接近原来的数据集概率分布 , 常用方法见下表

表4-6 属性规约常用方法

属性规约方法	方法描述	方法解析
合并属性	将一些旧属性合为新属性	初始属性集: $\{A_1, A_2, A_3, A_4, B_1, B_2, B_3, C\}$ $\{A_1, A_2, A_3, A_4\} \rightarrow A$ $\{B_1, B_2, B_3\} \rightarrow B$ \Rightarrow 规约后属性集: $\{A, B, C\}$
逐步向前选择	从一个空属性集开始 , 每次从原来属性集中选择一个当前最优的属性添加到当前属性子集中。直到无法选择出最优属性或满足一定阈值约束为止	初始属性集: $\{A_1, A_2, A_3, A_4, A_5, A_6\}$ $\{\} \Rightarrow \{A_1\} \Rightarrow \{A_1, A_4\}$ \Rightarrow 规约后属性集: $\{A_1, A_4, A_6\}$
逐步向后删除	从一个全属性集开始 , 每次从当前属性子集中选择一个当前最差的属性并将其从当前属性子集中消去。直到无法选择出最差属性为止或满足一定阈值约束为止	初始属性集: $\{A_1, A_2, A_3, A_4, A_5, A_6\}$ $\Rightarrow \{A_1, A_3, A_4, A_5, A_6\} \Rightarrow \{A_1, A_4, A_5, A_6\}$ \Rightarrow 规约后属性集: $\{A_1, A_4, A_6\}$
决策树归纳	利用决策树的归纳方法对初始数据进行分类归纳学习 , 获得一个初始决策树 , 所有没有出现在这个决策树上的属性均可认为是无关属性 , 因此将这些属性从初始集合中删除 , 就可以获得一个较优的属性子集	初始属性集: $\{A_1, A_2, A_3, A_4, A_5, A_6\}$  \Rightarrow 规约后属性集: $\{A_1, A_4, A_6\}$
主成分分析	用较少的变量去解释原始数据中的大部分变量 , 即将许多相关性很高的变量转化成彼此相互独立或不相关的变量	详见下面计算步骤

(2)、数值规约

- 指的是通过选择替代的，较小的数据来减少数据量，包括有参数，无参数方法
- 有参数：使用模型来评估数据，只存放参数，不存放实际数据 eg：线性回归/多元回归
- 无参数：存放实际数据，直方图/聚类/抽样
- (1) 直方图。如一连串的数据，通过绘制直方图（R中用hist()函数绘制直方图），分为“3~15”、“16~28”、“29~41”三个范围。
- (2) 聚类。将对象划分为簇，使一个簇中的对象相互“相似”，而与其他簇中的对象“相异”，用数据的簇替换实际数据。
- (3) 抽样。有放回/无放回

6. 泰坦尼克号数据预处理

1. 数据缺失值处理：age，Cabin，Embarked -----数据清洗
2. 数据one-hot处理：逻辑回归算法需要传入的数据是数值型-----转换数据
3. 数据标准化处理：对数据进行标准化处理-----标准化数据

```

1 # 缺失值处理 Age
2 data_train['Age']=data_train['Age'].fillna(data_train['Age'].mean())
3 # 缺失值处理 Embarked
4 data_train['Embarked']=data_train['Embarked'].fillna('S')
5 # 缺失值处理 Cabin
6 def set_cabin_type(df):
7     df.loc[(df.Cabin.notnull()), 'Cabin']='Yes'
8     df.loc[(df.Cabin.isnull()), 'Cabin']='No'
9     return df
10 data_train1=set_cabin_type(data_train)
11 data_train1.info()
12 # 数据类型转换 类别型数据转换为数值型
13 dummies_Cabin=pd.get_dummies(data_train1['Cabin'],prefix='Cabin')
14 dummies_Sex=pd.get_dummies(data_train1['Sex'],prefix='Sex')
15 dummies_Embarked=pd.get_dummies(data_train1['Embarked'],prefix='Embarked')
16 dummies_Pclass=pd.get_dummies(data_train1['Pclass'],prefix='Pclass')
17 df=pd.concat([data_train1,dummies_Cabin,
18                 dummies_Embarked,dummies_Pclass,dummies_Sex],axis=1)
19 df.head()
20 df.columns
21 df.drop(['Pclass','Name','Ticket','Cabin','Embarked','Sex']
22         ,axis=1,inplace=True)
23 # 数据标准化处理 Age,Fare
24 a=df.Age
25 df['Age_scaled']=(a-a.mean())/(a.std())
26 b=df.Fare
27 df['Fare_scaled']=(b-b.mean())/(b.std())
28 df.head()
29 df.drop('Age',axis=1,inplace=True)

```