知乎 首发于 Data Analyst

特征工程（一）数据预处理标准四个方法

Alan
数据分析、挖掘、机器学习

关注他

4 人赞同了该文章

一、特征工程数据处理的标准四个方法

目录

```
import warnings
warnings.filterwarnings('ignore')
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from scipy import stats
%matplotlib inline
```

1、缺失值处理

```
# 判断是否有缺失值数据 - isnull，notnull
# isnull：缺失值为True，非缺失值为False
# notnull：缺失值为False，非缺失值为True

s = pd.Series([12,33,45,23,np.nan,np.nan,66,54,np.nan,99])
df = pd.DataFrame({'value1':[12,33,45,23,np.nan,np.nan,66,54,np.nan,99,190],
                   'value2':['a','b','c','d','e',np.nan,np.nan,'f','g',np.nan,'g']})
# 创建数据

print(s.isnull())  # Series直接判断是否是缺失值，返回一个Series
print(df.notnull())  # Dataframe直接判断是否是缺失值，返回一个Series
print(df['value1'].notnull())  # 通过索引判断
print('------')

s2 = s[s.isnull() == False]
df2 = df[df['value2'].notnull()]    # 注意和 df2 = df[df['value2'].notnull()] ['value1'
print(s2)
print(df2)
# 筛选非缺失值
0    False
1    False
2    False
3    False
4     True
5     True
6    False
7    False
8     True
```

```
1     True    True
2     True    True
3     True    True
4    False    True
5    False   False
6     True   False
7     True    True
8    False    True
9     True   False
10    True    True
0      True
1      True
2      True
3      True
4     False
5     False
6      True
7      True
8     False
9      True
10     True
Name: value1, dtype: bool
------
0    12.0
1    33.0
2    45.0
3    23.0
6    66.0
7    54.0
9    99.0
dtype: float64
    value1 value2
0     12.0      a
1     33.0      b
2     45.0      c
3     23.0      d
4      NaN      e
7     54.0      f
8      NaN      g
10   190.0      g
# 删除缺失值 - dropna
```

```
# 创建数据

s.dropna(inplace = True)
df2 = df['value1'].dropna()
print(s)
print(df2)
# drop方法：可直接用于Series，Dataframe
# 注意inplace参数，默认False → 生成新的值
0    12.0
1    33.0
2    45.0
3    23.0
6    66.0
7    54.0
9    99.0
dtype: float64
0      12.0
1      33.0
2      45.0
3      23.0
6      66.0
7      54.0
9      99.0
10    190.0
Name: value1, dtype: float64
# 填充/替换缺失数据 - fillna、replace

s = pd.Series([12,33,45,23,np.nan,np.nan,66,54,np.nan,99])
df = pd.DataFrame({'value1':[12,33,45,23,np.nan,np.nan,66,54,np.nan,99,190],
                   'value2':['a','b','c','d','e',np.nan,np.nan,'f','g',np.nan,'g']})
# 创建数据

s.fillna(0,inplace = True)
print(s)
print('------')
# s.fillna(value=None, method=None, axis=None, inplace=False, limit=None, downcast=Non
# value：填充值
# 注意inplace参数

df['value1'].fillna(method = 'pad',inplace = True)
print(df)
```

```
s = pd.Series([1,1,1,1,2,2,2,3,4,5,np.nan,np.nan,66,54,np.nan,99])
s.replace(np.nan,'缺失数据',inplace = True)
print(s)
print('------')
# df.replace(to_replace=None, value=None, inplace=False, limit=None, regex=False, meth
# to_replace → 被替换的值
# value → 替换值

s.replace([1,2,3],np.nan,inplace = True)
print(s)
# 多值用np.nan代替
0    12.0
1    33.0
2    45.0
3    23.0
4     0.0
5     0.0
6    66.0
7    54.0
8     0.0
9    99.0
dtype: float64
------
    value1 value2
0     12.0      a
1     33.0      b
2     45.0      c
3     23.0      d
4     23.0      e
5     23.0    NaN
6     66.0    NaN
7     54.0      f
8     54.0      g
9     99.0    NaN
10   190.0      g
------
0       1
1       1
2       1
3       1
```

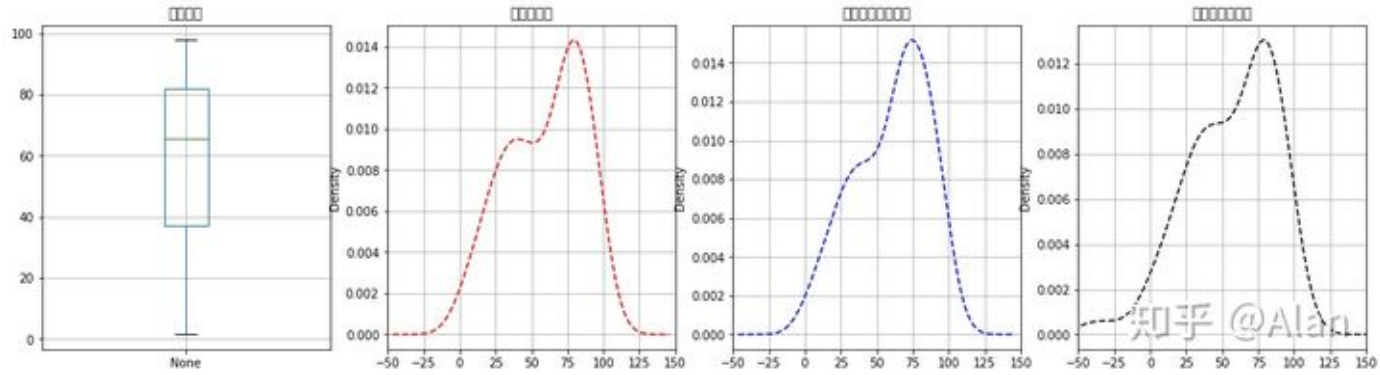
```
8        4
9        5
10    缺失数据
11    缺失数据
12       66
13       54
14    缺失数据
15       99
dtype: object
------
0      NaN
1      NaN
2      NaN
3      NaN
4      NaN
5      NaN
6      NaN
7      NaN
8        4
9        5
10    缺失数据
11    缺失数据
12       66
13       54
14    缺失数据
15       99
dtype: object
# 缺失值插补
# 几种思路：均值/中位数/众数插补、临近值插补、插值法
# （1）均值/中位数/众数插补

s = pd.Series([1,2,3,np.nan,3,4,5,5,5,5,np.nan,np.nan,6,6,7,12,2,np.nan,3,4])
#print(s)
print('------')
# 创建数据

u = s.mean()     # 均值
me = s.median()  # 中位数
mod = s.mode()   # 众数
print('均值为：%.2f，中位数为：%.2f' % (u,me))
print('众数为：', mod.tolist())
```

```
print(s)
# 用均值填补

s.fillna(s.mode()[0], inplace=True)
print(s)
# 用众数填补
------
均值为：4.56, 中位数为：4.50
众数为：  [5.0]
------
0      1.0000
1      2.0000
2      3.0000
3      4.5625
4      3.0000
5      4.0000
6      5.0000
7      5.0000
8      5.0000
9      5.0000
10     4.5625
11     4.5625
12     6.0000
13     6.0000
14     7.0000
15    12.0000
16     2.0000
17     4.5625
18     3.0000
19     4.0000
dtype: float64
# 缺失值插补
# 几种思路：均值/中位数/众数插补、临近值插补、插值法
# （2）临近值插补

s = pd.Series([1,2,3,np.nan,3,4,5,5,5,5,np.nan,np.nan,6,6,7,12,2,np.nan,3,4])
#print(s)
print('------')
# 创建数据

s.fillna(method = 'ffill',inplace = True)# 用前值插补
```

```
2      3.0
3      3.0
4      3.0
5      4.0
6      5.0
7      5.0
8      5.0
9      5.0
10     5.0
11     5.0
12     6.0
13     6.0
14     7.0
15    12.0
16     2.0
17     2.0
18     3.0
19     4.0
dtype: float64
# 缺失值插补
# 几种思路：均值/中位数/众数插补、临近值插补、插值法
# （3）插值法 —— 拉格朗日插值法

from scipy.interpolate import lagrange
x = [3, 6, 9]
y = [10, 8, 4]
print(lagrange(x,y))
print(type(lagrange(x,y)))
# 的输出值为的是多项式的n个系数
# 这里输出3个值，分别为a0,a1,a2
# y = a0 * x**2 + a1 * x + a2 → y = -0.11111111 * x**2 + 0.33333333 * x + 10

print('插值10为：%.2f' % lagrange(x,y)(10))
print('------')
# -0.11111111*100 + 0.33333333*10 + 10 = -11.11111111 + 3.33333333 +10 = 2.22222222
2
-0.1111 x + 0.3333 x + 10
<class 'numpy.poly1d'>
插值10为：2.22
------
# 缺失值插补
```

```
data[3,6,33,56,45,66,67,80,90] = np.nan
print(data.head())
print('总数据量:%i' % len(data))
print('------')
# 创建数据

data_na = data[data.isnull()]
print('缺失值数据量:%i' % len(data_na))
print('缺失数据占比:%.2f%%' % (len(data_na) / len(data) * 100))
# 缺失值的数量

data_c = data.fillna(data.median())  #  中位数填充缺失值
fig,axes = plt.subplots(1,4,figsize = (20,5))
data.plot.box(ax = axes[0],grid = True,title = '数据分布')
data.plot(kind = 'kde',style = '--r',ax = axes[1],grid = True,title = '删除缺失值',xlim
data_c.plot(kind = 'kde',style = '--b',ax = axes[2],grid = True,title = '缺失值填充中位
# 密度图查看缺失值情况

def na_c(s,n,k=5):
    y = s[list(range(n-k,n+1+k))] # 取数
    y = y[y.notnull()]  # 剔除空值
    return(lagrange(y.index,list(y))(n))
# 创建函数，做插值，由于数据量原因，以空值前后5个数据（共10个数据）为例做插值

na_re = []
for i in range(len(data)):
    if data.isnull()[i]:
        data[i] = na_c(data,i)
        print(na_c(data,i))
        na_re.append(data[i])
data.dropna(inplace=True)  # 清除插值后仍存在的缺失值
data.plot(kind = 'kde',style = '--k',ax = axes[3],grid = True,title = '拉格朗日插值后',
print('finished!')
# 缺失值插值
0    89.806020
1    28.068053
2    74.170920
3          NaN
4    81.800916
dtype: float64
总数据量:100
```

```
36.717798103077826
78.69398498535156
66.837890625
-62.7578125
-28.5439453125
-42.125
-192.0
-7136.0
finished!
```

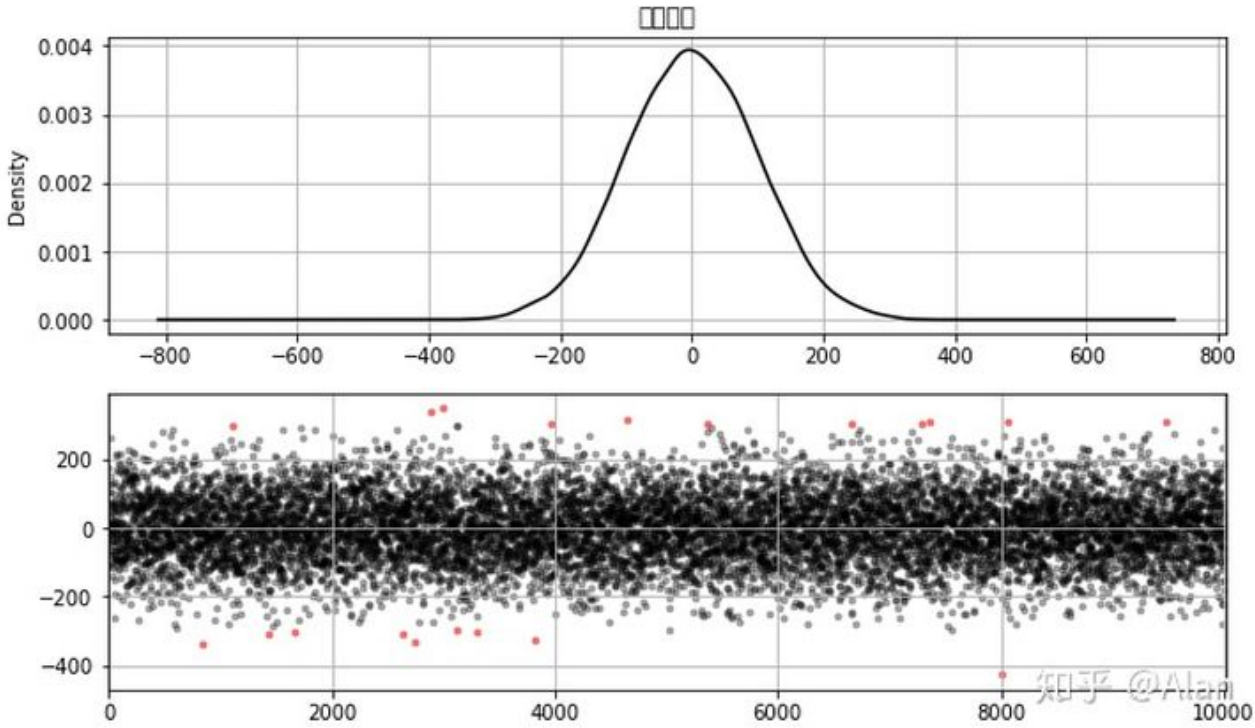


2、异常值处理

异常值是指样本中的个别值，其数值明显偏离其余的观测值。 异常值也称离群点，异常值的分析也称为离群点的分析 异常值分析 → 3σ原则 / 箱型图分析 异常值处理方法 → 删除 / 修正填补

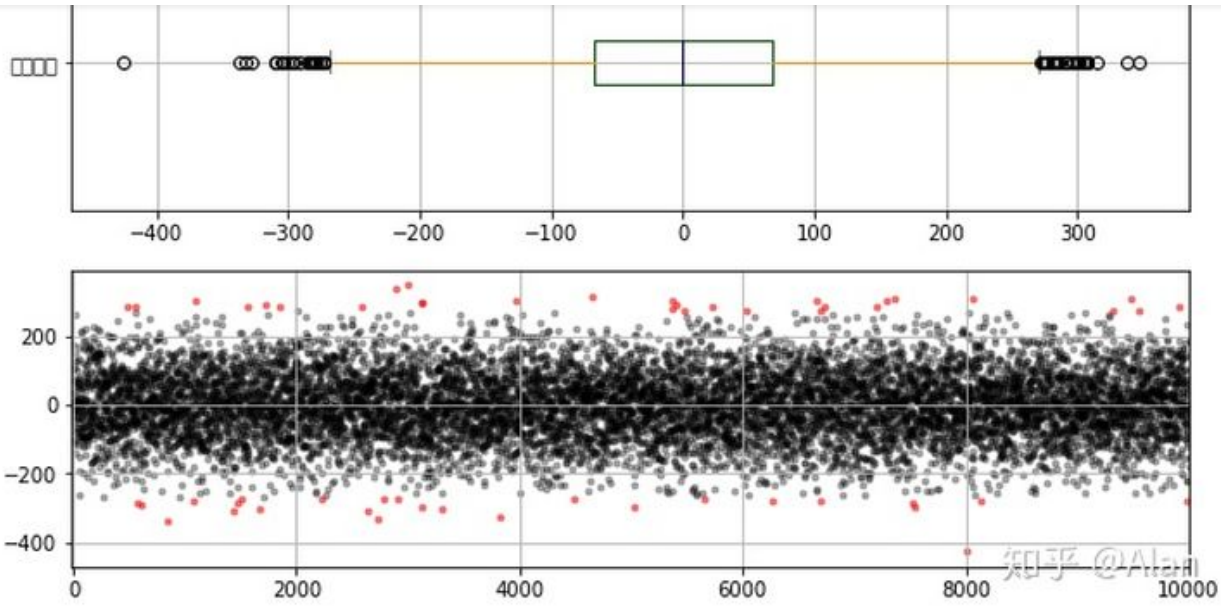
```
# 异常值分析
# （1）3σ原则：如果数据服从正态分布，异常值被定义为一组测定值中与平均值的偏差超过3倍的值 → p(|x

data = pd.Series(np.random.randn(10000)*100)
# 创建数据

u = data.mean()  # 计算均值
std = data.std()  # 计算标准差
stats.kstest(data, 'norm', (u, std))
print('均值为：%.3f，标准差为：%.3f' % (u,std))
print('------')
# 正态性检验
```

```
# 绘制数据密度曲线

ax2 = fig.add_subplot(2,1,2)
error = data[np.abs(data - u) > 3*std]
data_c = data[np.abs(data - u) <= 3*std]
print('异常值共%i条' % len(error))
# 筛选出异常值error、剔除异常值之后的数据data_c

plt.scatter(data_c.index,data_c,color = 'k',marker='.',alpha = 0.3)
plt.scatter(error.index,error,color = 'r',marker='.',alpha = 0.5)
plt.xlim([-10,10010])
plt.grid()
# 图表表达
均值为：0.494，标准差为：99.586
------
异常值共20条
```



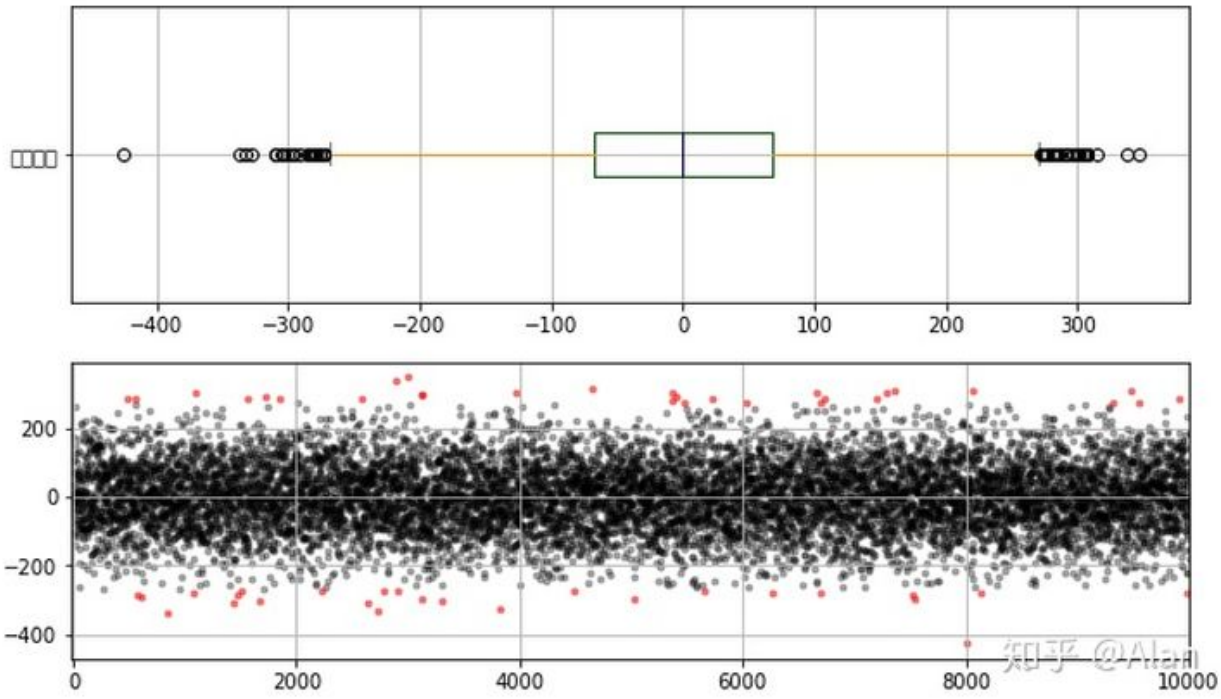

```
# 异常值分析
# （2）箱型图分析

fig = plt.figure(figsize = (10,6))
ax1 = fig.add_subplot(2,1,1)
```

```
s = data.describe()
print(s)
print('------')
# 基本统计量

q1 = s['25%']
q3 = s['75%']
iqr = q3 - q1
mi = q1 - 1.5*iqr
ma = q3 + 1.5*iqr
print('分位差为：%.3f，下限为：%.3f，上限为：%.3f' % (iqr,mi,ma))
print('------')
# 计算分位差

ax2 = fig.add_subplot(2,1,2)
error = data[(data < mi) | (data > ma)]
data_c = data[(data >= mi) & (data <= ma)]
print('异常值共%i条' % len(error))
# 筛选出异常值error、剔除异常值之后的数据data_c

plt.scatter(data_c.index,data_c,color = 'k',marker='.',alpha = 0.3)
plt.scatter(error.index,error,color = 'r',marker='.',alpha = 0.5)
plt.xlim([-10,10010])
plt.grid()
# 图表表达
count    10000.000000
mean         0.494461
std         99.586426
min       -425.512973
25%        -67.155857
50%          0.081517
75%         68.075496
max        347.035886
dtype: float64
------
分位差为：135.231，下限为：-270.003，上限为：270.923
------
异常值共56条
```




```
# 异常值分析
# （2）箱型图分析

fig = plt.figure(figsize = (10,6))
ax1 = fig.add_subplot(2,1,1)
color = dict(boxes='DarkGreen', whiskers='DarkOrange', medians='DarkBlue', caps='Gray'
data.plot.box(vert=False, grid = True,color = color,ax = ax1,label = '样本数据')
# 箱型图看数据分布情况
# 以内限为界

s = data.describe()
print(s)
print('------')
# 基本统计量

q1 = s['25%']
q3 = s['75%']
iqr = q3 - q1
mi = q1 - 1.5*iqr
ma = q3 + 1.5*iqr
print('分位差为：%.3f，下限为：%.3f，上限为：%.3f' % (iqr,mi,ma))
print('------')
# 计算分位差

ax2 = fig.add_subplot(2,1,2)
error = data[(data < mi) | (data > ma)]
```

```
plt.scatter(data_c.index,data_c,color = 'k',marker='.',alpha = 0.3)
plt.scatter(error.index,error,color = 'r',marker='.',alpha = 0.5)
plt.xlim([-10,10010])
plt.grid()
# 图表表达
count    10000.000000
mean         0.494461
std         99.586426
min       -425.512973
25%        -67.155857
50%          0.081517
75%         68.075496
max        347.035886
dtype: float64
------
分位差为：135.231，下限为：-270.003，上限为：270.923
------
异常值共56条
```



3、数据归一化/标准化

位或量级的指标能够进行比较和加权.

最典型的就是数据的归一化处理，即将数据统一映射到[0,1]区间上

- 0-1标准化 / Z-score标准化（更好）

```
# 数据标准化
# （1）0-1标准化（传统小型数据采用）
# 将数据的最大最小值记录下来，并通过Max-Min作为基数（即Min=0，Max=1）进行数据的归一化处理
# x = (x - Min) / (Max - Min)

df = pd.DataFrame({"value1":np.random.rand(10)*20,
                   'value2':np.random.rand(10)*100})
print(df.head())
print('------')
# 创建数据

def data_norm(df,*cols):
    df_n = df.copy()
    for col in cols:
        ma = df_n[col].max()
        mi = df_n[col].min()
        df_n[col + '_n'] = (df_n[col] - mi) / (ma - mi)
    return(df_n)
# 创建函数，标准化数据

df_n = data_norm(df,'value1','value2')
print(df_n.head())
# 标准化数据
```

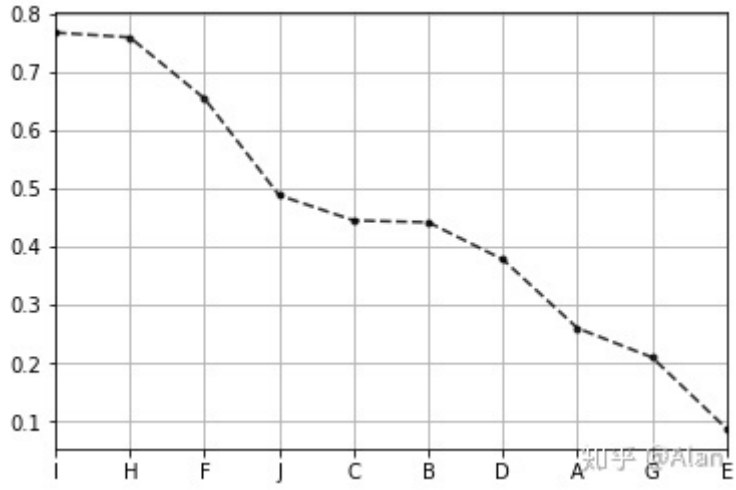
```
   value1     value2
0  15.674391  17.843683
1   4.075408  26.448068
2  15.923498  90.539146
3   9.493163  52.470951
4   6.275863  27.203625
------
      value1     value2  value1_n  value2_n
0  15.674391  17.843683  0.782627  0.130739
1   4.075408  26.448068  0.100390  0.223487
2  15.923498  90.539146  0.797279  0.914338
```

```
# Z分数（z-score），是一个分数与平均数的差再除以标准差的过程 → z=(x-μ)/σ，其中x为某一具体分数，
# Z值的量代表着原始分数和母体平均值之间的距离，是以标准差为单位计算。在原始分数低于平均值时Z则为
# 数学意义：一个给定分数距离平均数多少个标准差?

df = pd.DataFrame({"value1":np.random.rand(10) * 100,
                   'value2':np.random.rand(10) * 100})
print(df.head())
print('------')
# 创建数据

def data_Znorm(df, *cols):
    df_n = df.copy()
    for col in cols:
        u = df_n[col].mean()
        std = df_n[col].std()
        df_n[col + '_Zn'] = (df_n[col] - u) / std
    return(df_n)
# 创建函数，标准化数据

df_z = data_Znorm(df,'value1','value2')
u_z = df_z['value1_Zn'].mean()
std_z = df_z['value1_Zn'].std()
print(df_z)
print('标准化后value1的均值为:%.2f, 标准差为: %.2f' % (u_z, std_z))
# 标准化数据
# 经过处理的数据符合标准正态分布，即均值为0，标准差为1

# 什么情况用Z-score标准化：
# 在分类、聚类算法中，需要使用距离来度量相似性的时候，Z-score表现更好
```

```
      value1     value2
0  83.986301  58.261401
1  29.204116  10.039485
2  95.787421  87.983382
3   8.701032  74.906583
4  59.934405  10.303871
------
      value1     value2  value1_Zn  value2_Zn
0  83.986301  58.261401   0.740361   0.233820
1  29.204116  10.039485  -1.058607  -1.347609
2  95.787421  87.983382   1.127893   1.208547
3   8.701032  74.906583  -1.731899   0.779696
```

```
8  28.313196  28.792964  -1.087864  -0.732592
9  63.154840  78.364726   0.056286   0.893105
标准化后value1的均值为:-0.00, 标准差为：1.00
# 八类产品的两个指标value1，value2，其中value1权重为0.6，value2权重为0.4
# 通过0-1标准化，判断哪个产品综合指标状况最好

df = pd.DataFrame({"value1":np.random.rand(10) * 30,
                   'value2':np.random.rand(10) * 100},
                  index = list('ABCDEFGHIJ'))
#print(df.head())
#print('------')
# 创建数据"

df_n1 = data_norm(df,'value1','value2')
# 进行标准化处理

df_n1['f'] = df_n1['value1_n'] * 0.6 + df_n1['value2_n'] * 0.4
df_n1.sort_values(by = 'f',inplace=True,ascending=False)
df_n1['f'].plot(kind = 'line', style = '--.k', alpha = 0.8, grid = True)
df_n1
# 查看综合指标状况
```

value1 value2
0 83.986301 58.261401
1 29.204116 10.039485
2 95.787421 87.983382
3 8.701032 74.906583
4 59.934405 10.303871
---
value1 value2 value1_Zn value2_Zn
0 83.986301 58.261401 0.740361 0.233820
1 29.204116 10.039485 -1.058607 -1.347609
2 95.787421 87.983382 1.127893 1.208547
3 8.701032 74.906583 -1.731899 0.779696
4 59.934405 10.303871 -0.049469 -1.338939
5 99.112824 31.040091 1.237094 -0.658898
6 77.755465 43.865911 0.535749 -0.238278
7 68.458697 87.757755 0.230457 1.201148
8 28.313196 28.792964 -1.087864 -0.732592



4、数据连续属性离散化

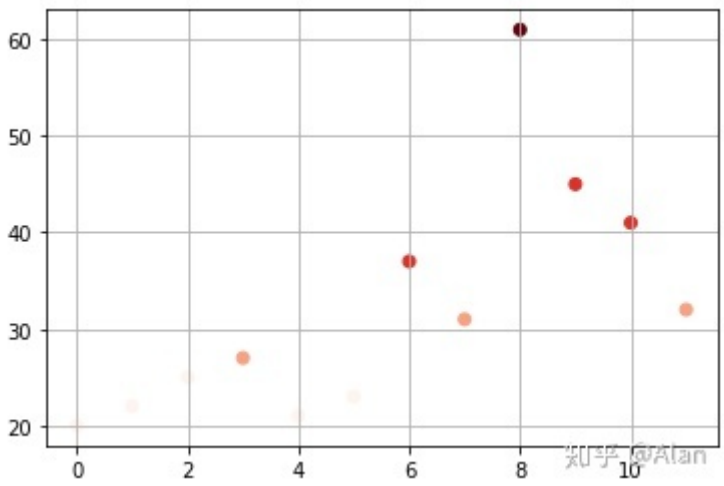
连续属性变换成分类属性，即连续属性离散化 在数值的取值范围内设定若干个离散划分点，将取值范围划分为一些离散化的区间， 最后用不同的符号或整数值代表每个子区间中的数据值

- 等宽法 / 等频法

```
# 等宽法 → 将数据均匀划分成n等份，每份的间距相等，一般用在“年龄、年份”等字段
# cut方法

ages=[20,22,25,27,21,23,37,31,61,45,41,32]
# 有一组人员年龄数据，希望将这些数据划分为“18到25”,“26到35”,“36到60”,“60以上”几个面元

bins = [18,25,35,60,100]
cats = pd.cut(ages,bins)
print(cats)
print(type(cats))
print('-------')
# 返回的是一个特殊的Categorical对象 → 一组表示面元名称的字符串

print(cats.codes, type(cats.codes))  # 0-3对应分组后的四个区间，用代号来注释数据对应区间，结
print(cats.categories, type(cats.categories))  # 四个区间，结果为index
print(pd.value_counts(cats))  # 按照区间计数
print('-------')
# cut结果含有一个表示不同分类名称的层级数组以及一个年龄数据进行标号的代号属性
```

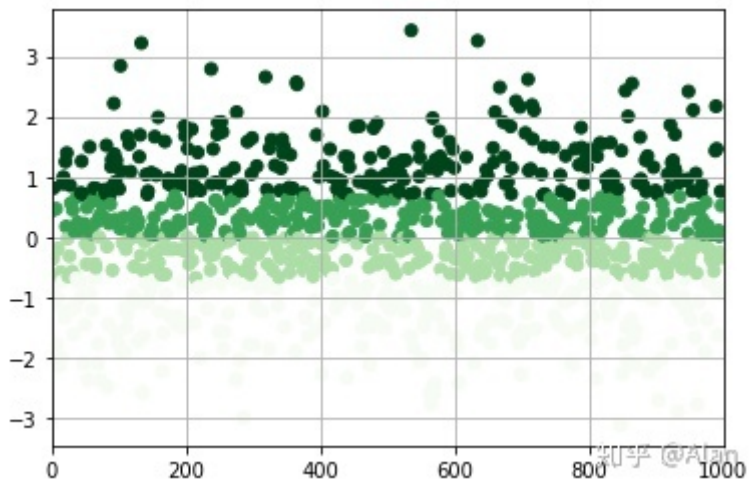
```
group_names=['Youth','YoungAdult','MiddleAged','Senior']
print(pd.cut(ages,bins,labels=group_names))
print('-------')
# 可以设置自己的区间名称，用labels参数

df = pd.DataFrame({'ages':ages})
group_names=['Youth','YoungAdult','MiddleAged','Senior']
s = pd.cut(df['ages'],bins)  # 也可以 pd.cut(df['ages'],5),将数据等分为5份
df['label'] = s
cut_counts = s.value_counts(sort=False)
print(df)
print(cut_counts)
# 对一个Dataframe数据进行离散化，并计算各个区间的数据计数

plt.scatter(df.index,df['ages'],cmap = 'Reds',c = cats.codes)
plt.grid()
# 用散点图表示，其中颜色按照codes分类
# 注意codes是来自于Categorical对象
[(18, 25], (18, 25], (18, 25], (25, 35], (18, 25], ..., (25, 35], (60, 100], (35, 60],
Length: 12
Categories (4, interval[int64]): [(18, 25] < (25, 35] < (35, 60] < (60, 100]]
<class 'pandas.core.arrays.categorical.Categorical'>
-------
[0 0 0 1 0 0 2 1 3 2 2 1] <class 'numpy.ndarray'>
IntervalIndex([(18, 25], (25, 35], (35, 60], (60, 100]],
              closed='right',
              dtype='interval[int64]') <class 'pandas.core.indexes.interval.IntervalIn
(18, 25]     5
(35, 60]     3
(25, 35]     3
(60, 100]    1
dtype: int64
-------
[[18, 26), [18, 26), [18, 26), [26, 36), [18, 26), ..., [26, 36), [61, 100), [36, 61),
Length: 12
Categories (4, interval[int64]): [[18, 26) < [26, 36) < [36, 61) < [61, 100)]
-------
[Youth, Youth, Youth, YoungAdult, Youth, ..., YoungAdult, Senior, MiddleAged, MiddleAg
Length: 12
Categories (4, object): [Youth < YoungAdult < MiddleAged < Senior]
```

```
2     25   (18, 25]
3     27   (25, 35]
4     21   (18, 25]
5     23   (18, 25]
6     37   (35, 60]
7     31   (25, 35]
8     61  (60, 100]
9     45   (35, 60]
10    41   (35, 60]
11    32   (25, 35]
(18, 25]     5
(25, 35]     3
(35, 60]     3
(60, 100]    1
Name: ages, dtype: int64
```




```
# 等频法 → 以相同数量的记录放进每个区间
# qcut方法

data = np.random.randn(1000)
s = pd.Series(data)
cats = pd.qcut(s,4)  # 按四分位数进行切割，可以试试 pd.qcut(data,10)
print(cats.head())
print(pd.value_counts(cats))
print('------')
# qcut → 根据样本分位数对数据进行面元划分，得到大小基本相等的面元，但并不能保证每个面元含有相同
# 也可以设置自定义的分位数（0到1之间的数值，包含端点） → pd.qcut(data1,[0,0.1,0.5,0.9,1])
```

```
# 用散点图表示，其中颜色按照codes分类
# 注意codes是来自于Categorical对象
0    (0.00553, 0.688]
1    (-0.68, 0.00553]
2      (0.688, 3.432]
3     (-3.126, -0.68]
4     (-3.126, -0.68]
dtype: category
Categories (4, interval[float64]): [(-3.126, -0.68] < (-0.68, 0.00553] < (0.00553, 0.6
(0.688, 3.432]      250
(0.00553, 0.688]    250
(-0.68, 0.00553]    250
(-3.126, -0.68]     250
dtype: int64
------
```



5、目标与建议

以上适合机器学习或传统的数据分析的（足够），为了让模型进一步获得更好的学习效果，还需掌握**高级方法**此建议你必须掌握以下几点特征工程的方法：

one-hot（类别特征）、 TF-IDF（词频统计）、PCA（聚类）、LDA、余弦相似度计算（推荐系统）、Word2Vec （文本）等（方法）；

集成模型XGBoost、高斯贝叶斯、LSTM、Seq2Seq、FastText、TextCNN（一个算法的输出=另一个算法的输入，因此也是特征工程的部分）；

1、CV 方向（不做详细描述，请自行百度）

图像的采集爬取、特征抽取、特征优化等

2、NLP方向（本人方向）

- **语料清洗**：保留有用的数据，删除噪音数据，常见的清洗方式有：人工去重、对齐、删除、标注等。
- **分词**：将文本分成词语，比如通过基于规则的、基于统计的分词方法进行分词。
- **词性标注**：给词语标上词类标签，比如名词、动词、形容词等，常用的词性标注方法有基于规则的、基于统计的算法，比如：最大熵词性标注、HMM 词性标注等。
- **去停用词**：去掉对文本特征没有任何贡献作用的字词，比如：标点符号、语气、“的”等

编辑于 2020-10-30

数据预处理 特征工程 数据挖掘