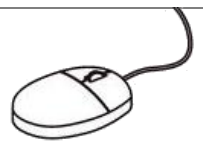
南通师范高等专科学校
Nantong Normal College

pandas数据处理之

数据处理

执教：朱亚林

数据清洗



为什么要进行数据清洗

- 数据清洗的目的是：提高数据质量。
- 数据清洗的主要任务是：处理缺失数据及清除无意义的信息。

重复值的处理

- 数据录入过程、数据整合过程都可能会产生重复数据，直接删除是重复数据处理的主要方法。
- pandas提供查看、处理重复数据的方法：

duplicated 和 *drop_duplicates*

使用duplicated（）查重

```
>>> sample =
pd.DataFrame({'id':[1,1,1,3,4,5],'name':['Bob','Bob','Mark','Miki','Sully','Rose'],
'score':[99,99,87,77,77,np.nan],'group':[1,1,1,2,1,2],})
>>> sample.duplicated()  # 以布尔值的形式反馈重复的行
>>> sample[sample.duplicated()] # 出示重复数据
```

使用drop_duplicates（）去重

- 对于上题数据，可以使用drop_duplicates()来进行去重

```
>>> sample = pd.DataFrame({'id':[1,1,1,3,4,5],'name':['Bob','Bob','Mark','Miki','Sully','Rose'],'score':[99,99,87,77,77,np.nan],'group':[1,1,1,2,1,2],})
>>> sample.drop_duplicates()  # 括号中可设置列名，从而对指定列进行去重。查重亦可
```

缺失值的处理

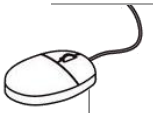
数据处理面向的对象更多是我们所说的大数据，大数据中的价值密度低，很多时候会出现值的缺失等等这样一些问题。

缺失值的处理

对于缺失数据一般的处理方式有：

- 删除缺失值
- 增补缺失值
- 不处理

缺失值的处理



简单粗暴型——删除缺失值

使用dropna()函数去除数据结构中值为空的数据行。一般步骤为：

- 使用isnull()函数来检测是否存在缺失值的情况

  可以使用df.isnull().any()查看哪些列有缺失值

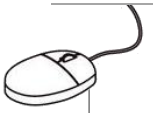
  可以使用df.apply(lambda col:sum(col.isnull())/col.size)来查看缺失比例

  可以使用df[df.isnull().values==True]查看具体缺失值的列

  可以使用np.sum(df.isnull(),axis=0)来统计每一列缺失值的数量

- 使用dropna()函数来删除缺失值的行

用指定值来填补缺失值



- 指定一个字符或数值来填充空白处，一句话：df.fillna('#')

你可能发现jupyter中数据表太大，你的数据没能显示出来。那就来设置一下pandas中显示的行和列的上限范围吧。比如：设置数据表显示300行就可以用以下语句。

```
pd.set_option('max_row',300)
```

用指定值来填补缺失值

- 使用前一个数据值来替代缺失值：

```
df.fillna(method='pad')
```

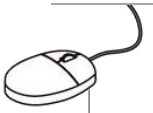
- 使用后一个数据值来替代缺失值：

```
df.fillna(method='bfill')
```

- 使用平均数或其他描述性统计量来替代缺失值：

```
df.fillna(df.mean())--平均值
df.fillna(df.median())--分位数
```

用指定值来填补缺失值



- pandas使用strip()函数来清除字符型数据首尾指定的字符，默认为空格，中间的不清除。如：

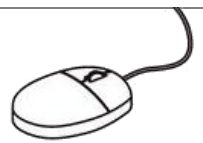
```
newDF=df['loan_staus'].strip()
```

异常值的处理

- 在一组数据当中，有时候会出现一些异常数据（不符合实际，特别大或特别小），这些数据会影响到后续的处理。因此，对于这些个别值（离群点）需要去进行处理。
- 使用方法对数据中的异常数据进行发现。
- 例如：describe()就可以对每一列数据进行统计，包括计数，均值，std，各个分位数等。通过观看这些数值之间的关系从而进行判断与调整。

```
df.describe().astype(np.int64).T
newDF=df.replace(50000,df['loan_amnt'].mean())
```

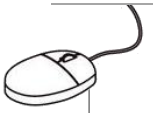
数据抽取



什么是数据抽取

- 数据抽取就是从数据集中将需要用到的数据信息提取出来的操作。

数据抽取的几种方式



- 字段抽取
- 字段拆分
- 记录抽取
- 随机抽样
- 切片抽取
- 字典数据

字段抽取

- pandas用slice()函数抽出某列上指定位置的数据，做成新的列。

```
slice(start,stop)
```

```
>>> df.str.slice(0,3)   #设置起始终止范围
>>> df.str.slice(6)     #设置起始点
```

字段拆分

- pandas用split()函数按指定的字符拆分已有的字符串。

```
split(sep,n,expand=False)
sep->用于分割的字符串
n -> 分割后新增的列数，要注意，n+1应该与实际切割后的字符串列数相等
expand -> True时返回DateFrame
       -> False时返回Series
```

```
>>> df.str.split('-',2,True)
```

记录抽取

pandas可以根据一定的条件对数据进行抽取。

```
DataFrame[condition]
```

condition常见的类型有：

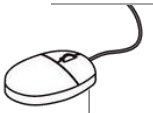
1. 比较运算符：<、>、<=、>=、!=
2. 范围运算：between(left,right)
3. 空置运算：isnull(column)
4. 字符匹配：str.contains(pattern,na=False)
5. 逻辑运算：&、|、not
6. 多个数值匹配：isin()

记录抽取

例：

```
>>> df[df.grade=='A']   # df[df['grade']=='A']
>>> df[df.loan_amnt > 25000] #df[df['loan_amnt'] > 25000]
>>> df[df.loan_amnt.between(25000,35000)]
>>> df[df.annual_inc.isnull]
>>> df[df.emp_length.str.contains('10+',na=False)]
>>> df[df.sth.isin([a,b,c,d])]
>>> df[(df.loan_amnt > 25000) & (df.loan_amnt<35000)]
```

随机抽样



- 使用randint生成随机数，再将其作为索引值代入DataFrame，从而实现随机抽样。

```
>>> rand = np.random.randint(0,10,3)
>>> df.loc[rand]
```

切片抽取

切片抽取之一：使用loc方法

DataFrame.loc[行索引名称或条件，列索引名称]

```
>>> df.loc[3:8]
>>> df.loc[3:8,'loan_status']
>>> df.loc[3:8,'loan_status':'emp_length']
```

切片抽取

切片抽取之二：使用iloc方法

DataFrame.iloc[行索引位置条件，列索引位置条件]

```
>>> df.iloc[3:8,2:5]
>>> df.iloc[[2,3,5],[1,2,6]]
>>> df.loc[3:8,'loan_status':'emp_length']
```

切片抽取

切片抽取之三：使用ix方法

DataFrame.ix[行索引名称、位置、条件，列索引名称、位置]

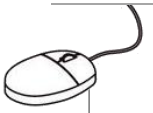
```
>>> df.ix[2:6,2]
```

ix使用注意事项

1. 使用ix参数时，尽量保持行索引名称和行索引位置重叠，使用时就无须考虑取值敬意的问题。
2. 使用列索引名称，而非列索引位置，主要用来保证代码的可读性。
3. 使用列索引位置时需要注解，同样用来保证代码的可读性。

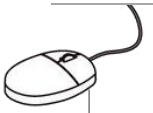
ix的缺点是在面对数据量巨大的任务时，其效率会低于loc和iloc方法。

字典数据



- DataFrame可以从字典进行创建，所以将字典数据抽取为DataFrame时也就非常方便。主要有以下三种方法。
- 将字典中的key和value各作为一列；
- 将字典中的每一个元素作为一列（同长）；
- 将字典中的每一个元素作为一列（不同长）；

字典数据



- 将字典中的key和value各作为一列

```
>>> d1 = {'a':'[1,2,3]','b':'[4,5,6]'}
>>> a1 = pd.DataFrame.from_dict(d1,orient='index')
>>> a1.index.name='key'
>>> b1 = a1.reset_index()
>>> b1.column=['key','value']
```

字典数据

- 将字典中的每一个元素作为一列（同长）

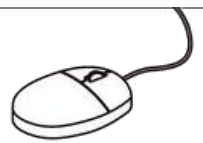
就是DataFrame的生成方法

字典数据

- 将字典中的每一个元素作为一列（不同长）

```
>>> d1 = {'a':pd.Series([1,2,3]),'b':pd.Series([4,5,6,7])}
>>> a1 = pd.DataFrame(d1)
```

排名索引



排序

排序是数据处理时的一种常见操作，在pandas当中对数值进行排序主要使用sort_index()函数。

对于Series，sort_index()函数中包含ascending参数。当值为True时进行升序排序，当值为False时，进行降序排序。

对于DataFrame，sort_index()函数中还包含axis和by参数。axis表示排序轴，1为横向，0为纵向。by表示可选择字段进行排序。

排序

```
import pandas as pd
import numpy as np
```

```
dic1={'Australia':[0,6,3],'China':[7,4,1],'Korea':[2,8,5]}
df = pd.DataFrame(dic1,index=list('ACK'))
```

```
df
```

	Australia	China	Korea
A	0	7	2
C	6	4	8
K	3	1	5

1

```
df.sort_index(ascending=True)
```

	Australia	China	Korea
A	0	7	2
C	6	4	8
K	3	1	5

```
df.sort_index(ascending=False)
```

	Australia	China	Korea
K	3	1	5
C	6	4	8
A	0	7	2

2

```
df.sort_index(ascending=False,axis=0)
```

	Australia	China	Korea
K	3	1	5
C	6	4	8
A	0	7	2

3

```
df.sort_index(ascending=False,axis=1)
```

	Korea	China	Australia
A	2	7	0
C	8	4	6
K	5	1	3

```
df.sort_values(by='China')
```

	Australia	China	Korea
K	3	1	5
C	6	4	8
A	0	7	2

4

排名

- 排名函数rank()用来对一组数据进行排名，如存在一组数据4、3、2、3，如果要从小到大对该组数据进行排名，应该排成如下图所示：

数值	4	3	2	3
排名	4	2	1	2

- **rank()函数中的参数之一：ascending：**指定排名方式（正向或逆向）

排名

以上是常规排名方式，在pandas中对于相同两值的排名，默认采取的不是从上一值顺延下来，而取相同值顺次排名下来的平均值。

数值	4	3	2	3
排名	4.0	2.5	1.0	2.5

rank()函数参数之二method，其值有4个可选项

- ✓ average-- 默认，取平均值
- ✓ min -- 如有重复值，均取第1个值出现的排名
- ✓ max -- 如有重复值，均取最后一个值出现的排名
- ✓ first -- 如有重复值，按照出现顺序依次排名

排名

- rank()函数对于DataFrame而言，比Series多了一个参数，即axis。

即：排名的方向。

不讲了，自己悟！

对照sort_index()中的数据进行练习。

重新索引

- 使用reindex()函数对Series对象进行索引重建。
- reindex()函数有两个参数，即fill_value和method。

fill_value对重建后的内容为空的值进行指定值的填充。

```
ser.reindex(A,fill_value=0)
```

method指定填充方式，ffill/pad 向前或进位填充，bfill/backfill 向后或进位填充。

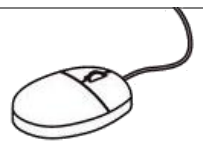
```
ser.reindex(A,method='bfill')
```

重新索引

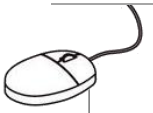
- 使用reindex()函数也可对DataFrame对象进行索引重建。
- 此时的参数中多了一个columns参数，用以设置重建的列名。

```
df.reindex(index=['a','b','d'],columns=['one','two','five'])
```

数据合并



堆叠合并数据



- 所谓堆叠合并数据，就是将两个表按照一定的要求拼在一起。根据拼接的方向，可以分为横向堆叠和纵向堆叠。

- 对于pandas中的数据堆叠，用得最多的函数就是：**concat()**

堆叠合并数据

concat()函数的基本语法

```
concat(objs,axis=1,join='outer',join_axes=None,ignore_index=False,keys=None,levels=No
ne,names=None,verify_integrity=False)
```

当axis = 1的时候，concat就是行对齐，然后将不同列名称的两张表合并

```
result = pd.concat([df1, df4], axis=1)
```

df1

	A	B	C	D
0	A0	B0	C0	D0
1	A1	B1	C1	D1
2	A2	B2	C2	D2
3	A3	B3	C3	D3

df4

	B	D	F
2	B2	D2	F2
3	B3	D3	F3
6	B6	D6	F6
7	B7	D7	F7

Result

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

堆叠合并数据

concat()函数的基本语法

```
concat(objs,axis=1,join='outer',join_axes=None,ignore_index=False,keys=None,levels=No
ne,names=None,verify_integrity=False)
```

加上join参数的属性，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。

```
result = pd.concat([df1, df4], axis=1, join='inner')
```

df1

	A	B	C	D
0	A0	B0	C0	D0
1	A1	B1	C1	D1
2	A2	B2	C2	D2
3	A3	B3	C3	D3

df4

	B	D	F
2	B2	D2	F2
3	B3	D3	F3
6	B6	D6	F6
7	B7	D7	F7

Result

	A	B	C	D	B	D	F
2	A2	B2	C2	D2	B2	D2	F2
3	A3	B3	C3	D3	B3	D3	F3

堆叠合并数据

concat()函数的基本语法

```
concat(objs,axis=1,join='outer',join_axes=None,ignore_index=False,keys=None,levels=No
ne,names=None,verify_integrity=False)
```

join_axes的参数传入，可以指定根据那个轴来对齐数据。例如根据df1表对齐数据，就会保留指定的df1表的轴，然后将df4的表与之拼接

```
result = pd.concat([df1, df4], axis=1, join_axes=[df1.index])
```

df1

	A	B	C	D
0	A0	B0	C0	D0
1	A1	B1	C1	D1
2	A2	B2	C2	D2
3	A3	B3	C3	D3

df4

	B	D	F
2	B2	D2	F2
3	B3	D3	F3
6	B6	D6	F6
7	B7	D7	F7

Result

	A	B	C	D	B	D	F
0	A0	B0	C0	D0	NaN	NaN	NaN
1	A1	B1	C1	D1	NaN	NaN	NaN
2	A2	B2	C2	D2	B2	D2	F2
3	A3	B3	C3	D3	B3	D3	F3

堆叠合并数据

concat()函数的基本语法

```
concat(objs,axis=1,join='outer',join_axes=None,ignore_index=False,keys=None,levels=None,names=None,verify_integrity=False)
```

如果两个表的index都没有实际含义，使用ignore_index参数，置true，合并的两个表就会根据列字段对齐，然后合并。最后再重新整理一个新的index。

```
result = pd.concat([df1, df4],
axis=0,ignore_index=True )
```

df1

	A	B	C	D
0	A0	B0	C0	D0
1	A1	B1	C1	D1
2	A2	B2	C2	D2
3	A3	B3	C3	D3

df4

	B	D	F
2	B2	D2	F2
3	B3	D3	F3
6	B6	D6	F6
7	B7	D7	F7

Result

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

堆叠合并数据

使用append来对数据进行纵向合并。

```
result = df1.append(df2,ignore_index=False,verify_integrity=False)
```

append要求合并的两张表的列名要完全一致。

ignore_index:True为重新索引

verify_integrity:检查数据索引是否冲突

df1

	A	B	C	D
0	A0	B0	C0	D0
1	A1	B1	C1	D1
2	A2	B2	C2	D2
3	A3	B3	C3	D3

df2

	A	B	C	D
4	A4	B4	C4	D4
5	A5	B5	C5	D5
6	A6	B6	C6	D6
7	A7	B7	C7	D7

Result

	A	B	C	D
0	A0	B0	C0	D0
1	A1	B1	C1	D1
2	A2	B2	C2	D2
3	A3	B3	C3	D3
4	A4	B4	C4	D4
5	A5	B5	C5	D5
6	A6	B6	C6	D6
7	A7	B7	C7	D7

主键合并数据

- 主键合并即通过一个或多个键将两个数据集的行连接起来。针对两张包含不同字段的表，将其根据某几个字段一一对应拼接起来，结果集的列数为两个原数据的和减去连接键的数量。
- 主键合并使用到的两个函数是merge()和join()。

主键合并数据

- merge的操作非常类似sql里面的join。其作用是将两个Dataframe根据一些共有的列连接起来，当然，在实际场景中，这些共有列一般是Id，
- merge的连接方式也丰富多样，可以选择inner(默认)，left,right,outer 这几种模式，分别对应的是内连接，左连接，右连接

主键合并数据

merge的默认操作

```
import numpy as np
import pandas as pd
```

```
df1 = pd.DataFrame({'key':['A','B','A','C','B','C'],'value_of_df1':np.arange(6)})
```

```
df1
```

	key	value_of_df1
0	A	0
1	B	1
2	A	2
3	C	3
4	B	4
5	C	5


```
df2 = pd.DataFrame({'key':['A','D','C'],'value_of_df2':[1,3,5]})
```

```
df2
```

	key	value_of_df2
0	A	1
1	D	3
2	C	5

```
pd.merge(df1,df2)
```

	key	value_of_df1	value_of_df2
0	A	0	1
1	A	2	1
2	C	3	5
3	C	5	5

merge的默认操作使用的是内连接，

即how='inner'，on='key'，

合并的数据为两个DF中都存在的'key'。

主键合并数据

merge的LeftMerge

df1

	key	value_of_df1
0	A	0
1	B	1
2	A	2
3	C	3
4	B	4
5	C	5

df2

	key	value_of_df2
0	A	1
1	D	3
2	C	5

```
pd.merge(df1,df2,how='left')
```

	key	value_of_df1	value_of_df2
0	A	0	1.0
1	B	1	NaN
2	A	2	1.0
3	C	3	5.0
4	B	4	NaN
5	C	5	5.0

可以看到返回的是df1的所有key值对应的结果，如果在df2中不存在，显示为Nan空值

主键合并数据

merge的RightMerge

df1

	key	value_of_df1
0	A	0
1	B	1
2	A	2
3	C	3
4	B	4
5	C	5

df2

	key	value_of_df2
0	A	1
1	D	3
2	C	5

```
pd.merge(df1,df2,how='right')
```

	key	value_of_df1	value_of_df2
0	A	0.0	1
1	A	2.0	1
2	C	3.0	5
3	C	5.0	5
4	D	NaN	3

可以看到返回的是df2的所有key值对应的结果，如果在df1中不存在，显示为Nan空值

主键合并数据

merge的OuterMerge

df1

	key	value_of_df1
0	A	0
1	B	1
2	A	2
3	C	3
4	B	4
5	C	5

df2

	key	value_of_df2
0	A	1
1	D	3
2	C	5

```
pd.merge(df1,df2,how='outer')
```

	key	value_of_df1	value_of_df2
0	A	0.0	1.0
1	A	2.0	1.0
2	B	1.0	NaN
3	B	4.0	NaN
4	C	3.0	5.0
5	C	5.0	5.0
6	D	NaN	3.0

这里就是一个并集的形式啦，其实就是一个union的结果，会把key这一列在两个Dataframe出现的所有值全部显示出来，如果有空值显示为Nan

主键合并数据

MultipleKey Merge (基于多个key上的merge)

```
import pandas as pd
import numpy as np
```

```
df_left = pd.DataFrame({'key1': ['SF', 'SF', 'LA'],
                        'key2': ['one', 'two', 'one'],
                        'left_data': [10,20,30]})
```

```
df_right = pd.DataFrame({'key1': ['SF', 'SF', 'LA', 'LA'],
                         'key2': ['one', 'one', 'one', 'two'],
                         'right_data': [40,50,60,70]})
```

1

df_left

	key1	key2	left_data
0	SF	one	10
1	SF	two	20
2	LA	one	30

df_right

2

	key1	key2	right_data
0	SF	one	40
1	SF	one	50
2	LA	one	60
3	LA	two	70

内连接（交集）的结果

```
pd.merge(df_left, df_right, on=['key1', 'key2'])
```

	key1	key2	left_data	right_data
0	SF	one	10	40
1	SF	one	10	50
2	LA	one	30	60

3

外连接（并集）的结果

```
pd.merge(df_left, df_right, on=['key1', 'key2'],how='outer')
```

	key1	key2	left_data	right_data
0	SF	one	10.0	40.0
1	SF	one	10.0	50.0
2	SF	two	20.0	NaN
3	LA	one	30.0	60.0
4	LA	two	NaN	70.0

4

主键合并数据

使用join()函数也可以实现部分主键的合并功能。但是在使用join()函数时，两个主键的名字必须相同。

```
left
```

	A	B
K0	A0	B0
K1	A1	B1
K2	A2	B2
K3	A3	B3

```
right
```

	C	D
K0	C0	D0
K1	C1	D1
K2	C2	D2

```
left.join(right)
```

	A	B	C	D
K0	A0	B0	C0	D0
K1	A1	B1	C1	D1
K2	A2	B2	C2	D2
K3	A3	B3	NaN	NaN

```
left.join(right,how='right')
```

	A	B	C	D
K0	A0	B0	C0	D0
K1	A1	B1	C1	D1
K2	A2	B2	C2	D2

```
right.join(left)
```

	C	D	A	B
K0	C0	D0	A0	B0
K1	C1	D1	A1	B1
K2	C2	D2	A2	B2

重叠合并数据

- 对两份数据进行缺失值补充时，可以使用combine_first()函数。即：A表中的数据有缺失，但B表中有部分数据可以用来进行填充。

```
a
```

	0	1	2
0	NaN	9	12.0
1	0.0	1	21.0
2	NaN	18	NaN

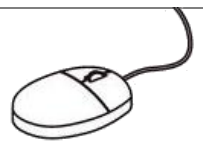
```
b
```

	0	1	2
0	9.0	8	7
1	NaN	1	3
2	NaN	2	26

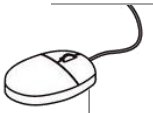
```
a.combine_first(b)
```

	0	1	2
0	9.0	9	12.0
1	0.0	1	21.0
2	NaN	18	26.0

数据计算



简单计算



- 字段间的四则运算

```
df.a * df.b
```

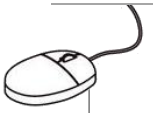
- 不同DF之间的四则运算

```
df1 * df2
```

- DF与其他值之间的四则运算

```
df * 3
```

数据标准化



归一化

就是将训练集中某一列数值特征（假设是第i列）的值缩放到0和1之间。方法如下所示：

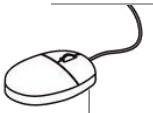
$$\frac{x_i - \min(x_i)}{\max(x_i) - \min(x_i)}$$

标准化

就是将训练集中某一列数值特征（假设是第i列）的值缩放成均值为0，方差为1的状态。如下所示：

$$\frac{x_i - \bar{x}}{sd(x)}$$

数据标准化



- 归一化和标准化的相同点都是对某个特征（column）进行缩放（scaling）而不是对某个样本的特征向量（row）进行缩放。对特征向量进行缩放是毫无意义的。比如三列特征：身高、体重、血压。每一条样本（row）就是三个这样的值，对这个row无论是进行标准化还是归一化都是好笑的，因为你不能将身高、体重和血压混到一起去！

数据标准化

一个数据归一化的例子

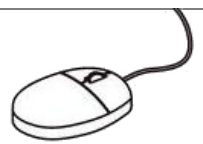
```
df_c
```

	red	blue	green
0	0	1	2
1	3	4	5
2	6	7	8
3	9	10	11

```
(df_c - df_c.min())/(df_c.max()-df_c.min())
```

	red	blue	green
0	0.000000	0.000000	0.000000
1	0.333333	0.333333	0.333333
2	0.666667	0.666667	0.666667
3	1.000000	1.000000	1.000000

数据分组



数据分组

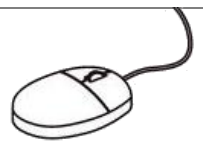
数据分组是指根据数据分析对象的特征，按照一定的数据指标，把数据划分成不同的敬意来进行研究，以提示其内在的联系和规律性。简单地说，就是新增一列，将原来的数据按照其性质归入新的类别中。

cut(series,bins,right,labels)

```
df2=pd.read_csv('loan_data.csv',sep=',',encoding='gb18030')
```

```
bins =[0,50000,100000,200000,1000000]
group_names = ['D','C','B','A']
df2['categories']=pd.cut(df2['年收入'],bins,labels=group_names,right=True)
```

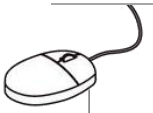
日期处理



日期处理

数据分析对象中常常会使用到日期与时间，对于这一类的数据一般会以文本的形式存在于数据集中。因此，在读入这些数据后，要将其转换成日期类型就要使用到日期处理的相关功能。pandas中可以快速实现时间字符串的转换、信息提取和时间运算。

日期转换



日期转换是指将字符型的日期格式转换为日期格式数据的过程。一般使用如下函数。

```
to_datetime(datestring,format)  #此处format是指数据中文本表示时间的格式
```

参数名称	说明
%Y	代表年份
%m	代表月份
%d	代表日期
%H	代表小时
%M	代表分钟
%S	代表秒

日期格式化

日期格式化是指将日期型的数据按照指定样式进行表达。

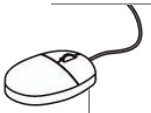
```
apply(lambda x:pd.datetime.strftime(x,format))
```

日期抽取

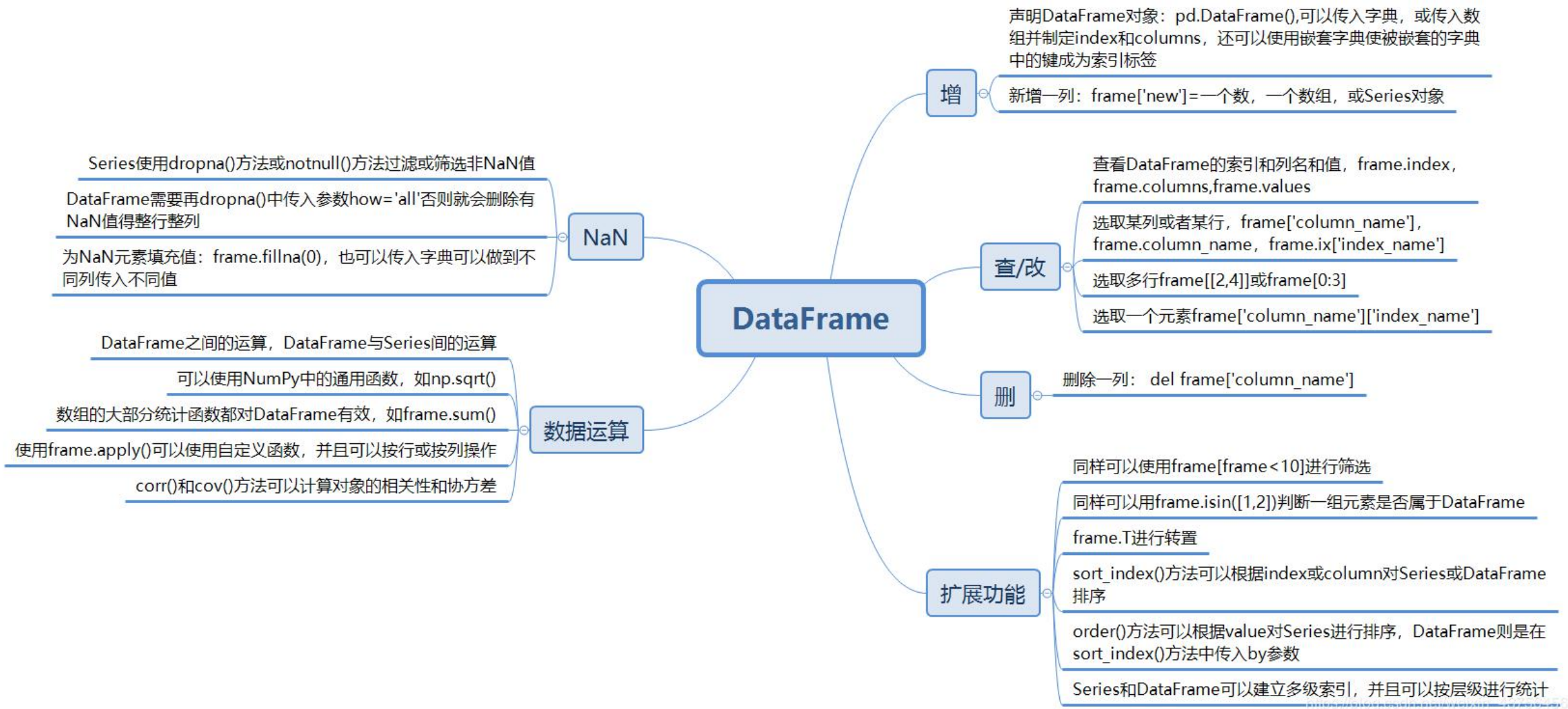
使用代表特殊含义的参数，将日期类型数值中的指定对象提取出来。

```
date.dt.XXX
```

参数名称	说明
second	秒，取值为1~60
minute	分，取值为1~60
hour	时，取值为1~24
day	日，取值为1~31
month	月，取值为1~12
year	年
weekday	取值为1~7



- Series
  - 基础
    - 声明Series对象：s=pd.Series([1,2,3,4], index=['a','b','c','d'])，不指定index时默认为从0递增的数字
    - a = pd.Series()可以传入np数组，但是要注意这里也是传入引用而不是副本，注意只能传入一维数组
    - a = pd.Series()还可以传入字典，键会变成index，值会变成value，这里是副本
    - 查看Series的索引数组和值数组：s.index或s.values
    - 获取/修改值元素：s[2] / s['a'] / s[0:2] / s[['b','c']]
    - 筛选元素：s[s>8]
    - 判断一组元素是否属于Series对象，series_name.isin([1,2])返回逻辑数组，因此也可以用于筛选
    - 根据索引删除值series.drop()，传入值或者数组
  - 空值NaN
    - NaN（Not a Number,非数值）
    - 在创建Series时，可以把NaN加入数据结构中的，s=pd.Series([1,np.NaN,2])
    - 识别Series中有/无NaN的方法：s.isnull() / s.notnull() 返回逻辑Series（由布尔值组成），然后可以用作筛选条件
  - 运算
    - 基础运算+ - * /与np相同（标量运算）
    - 可以使用np中的数学函数，如np.log(series)
    - Series对象间的加法，只有两者都有的index才会加，其余的值都为NaN


DataFrame
增
声明DataFrame对象：pd.DataFrame(),可以传入字典，或传入数组并制定index和columns，还可以使用嵌套字典使被嵌套的字典中的键成为索引标签
新增一列：frame['new']=一个数，一个数组，或Series对象
查/改
查看DataFrame的索引和列名和值，frame.index，frame.columns,frame.values
选取某列或者某行，frame['column_name']，frame.column_name，frame.ix['index_name']
选取多行frame[[2,4]]或frame[0:3]
选取一个元素frame['column_name']['index_name']
删
删除一列： del frame['column_name']
扩展功能
同样可以使用frame[frame<10]进行筛选
同样可以用frame.isin([1,2])判断一组元素是否属于DataFrame
frame.T进行转置
sort_index()方法可以根据index或column对Series或DataFrame排序
order()方法可以根据value对Series进行排序，DataFrame则是在sort_index()方法中传入by参数
Series和DataFrame可以建立多级索引，并且可以按层级进行统计
NaN
Series使用dropna()方法或notnull()方法过滤或筛选非NaN值
DataFrame需要再dropna()中传入参数how='all'否则就会删除有NaN值得整行整列
为NaN元素填充值：frame.fillna(0)，也可以传入字典可以做到不同列传入不同值
数据运算
DataFrame之间的运算，DataFrame与Series间的运算
可以使用NumPy中的通用函数，如np.sqrt()
数组的大部分统计函数都对DataFrame有效，如frame.sum()
使用frame.apply()可以使用自定义函数，并且可以按行或按列操作
corr()和cov()方法可以计算对象的相关性和协方差