





Python

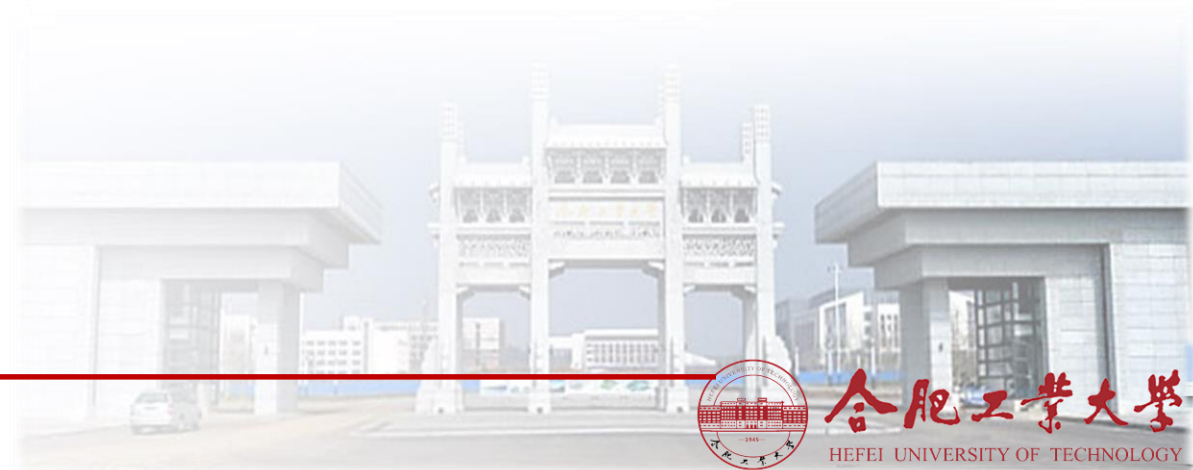
数据分析及可视化-9

数据处理分析



本章学习目标

-  掌握数据清洗的常见操作
-  掌握数据合并的常用方法





Pandas数据清洗

空值和缺失值的处理

空 值：表示数据未知，不适用或将在以后添加数据，一般用None表示。

缺失值：是指数据集中某个或某些属性的值是不完整的，使用NaN表示。

isnull() 函数

格式：pd.isnull(series_obj)

notnull() 函数

格式：pd.notnull(series_obj)

```
>>> ds0=pd.Series(range(6))
```

```
>>> ds1=ds0.copy()
```

```
>>> ds1[0]=np.nan
```

```
>>> ds1[5]=None
```

```
>>> ds1[3]=' '
```

```
>>> ds1[1]=""
```

```
>>> ds0
```

```
0    0
```

```
1    1
```

```
2    2
```

```
3    3
```

```
4    4
```

```
5    5
```

```
dtype: int64
```

```
>>> ds1
```

```
0    NaN
```

```
1
```

```
2    2
```

```
3
```

```
4    4
```

```
5    NaN
```

```
>>> ds1.isnull()
```

```
0    True
```

```
1    False
```

```
2    False
```

```
3    False
```

```
4    False
```

```
5    True
```

```
dtype: bool
```



数据预处理

空值和缺失值的处理

dropna() 方法

dropna() 方法是删除含有空值或缺失值的行或列。

格式: `pd_obj.dropna(axis=0, how='any', inplace=False)`

axis: 确定过滤行或列, 0或index, 删除包含缺失值的行, 1或columns, 删除包含缺失值的列。

how: 确定过滤标准, any为默认值, 表示如果存在NaN值, 则删除该行或该列。all为如果所有值都是NaN值, 则删除该行或该列。

inplace: 默认值为False, 创建新对象。

```
>>> df=pd.DataFrame({'a':[1,2,3,np.NaN],  
                    'b':[np.NaN,4,np.NaN,6],  
                    'c':['a',7,8,9],  
                    'd':[np.NaN,2,3,np.NaN]})
```

```
>>> df  
   a    b    c    d  
0  1.0 NaN  a  NaN  
1  2.0  4.0  7  2.0  
2  3.0 NaN  8  3.0  
3  NaN  6.0  9  NaN
```

```
>>> df0=df.copy()  
>>> df0.dropna(axis=0)  
   a    b    c    d  
1  2.0  4.0  7  2.0
```

```
>>> df0.dropna(axis=1)
```

```
   c  
0  a  
1  7  
2  8  
3  9
```





数据预处理

空值和缺失值的处理

fillna() 函数

fillna() 方法是填充空值或缺失值。

格式: `pd_obj.fillna(value=None, method=None, axis=None)`

value: 用于填充的值。

method: 表示填充方式，默认为None，另外还支持以下取值：pad/ffill(用缺失值前面的一个值代替缺失值)，backfill/bfill(用缺失值后面的一个值代替缺失值)。

注意: value参数不能与method参数同时使用。

```
>>> df
      a    b    c    d
0  1.0  NaN  a   NaN
1  2.0  4.0  7   2.0
2  3.0  NaN  8   3.0
3  NaN  6.0  9   NaN
>>> df0=df.copy()
```

```
>>> df0.fillna(10)
```

	a	b	c	d
0	1.0	10.0	a	10.0
1	2.0	4.0	7	2.0
2	3.0	10.0	8	3.0
3	10.0	6.0	9	10.0

```
>>> df0.fillna(method='ffill', axis=0)
```

	a	b	c	d
0	1.0	NaN	a	NaN
1	2.0	4.0	7	2.0
2	3.0	4.0	8	3.0
3	3.0	6.0	9	3.0

#指定列填充指定数据

```
>>> df0.fillna({'b':5, 'd':'b'})
```

	a	b	c	d
0	1.0	5.0	a	b
1	2.0	4.0	7	2
2	3.0	5.0	8	3
3	NaN	6.0	9	b





数据预处理

重复值的处理

duplicated() 方法

duplicated() 方法用于标记Pandas对象的数据是否重复，重复标记为True，否则标记为False，所以该方法返回一个由布尔值组成的Series对象，它的行索引保持不变，数据则改为布尔值。

格式：pd_obj.duplicated(subset=None, keep='first')

subset：用于识别重复的列标签或列标签序列，默认识别所有的列标签。

keep：删除重复项并保留第一次出现的项，取值可以为first、last或False。

```
>>> df
```

	id	name	age	sex
0	1001	susan	16	F
1	1002	kali	17	F
2	1003	davi	18	M
3	1003	davi	18	M
4	1004	jack	16	M

```
>>> df.duplicated()
```

```
0    False
```

```
1    False
```

```
2    False
```

```
3     True
```

```
4    False
```

```
dtype: bool
```

```
>>> df.duplicated(subset='age', keep='last')
```

```
0     True
```

```
1    False
```

```
2     True
```

```
3    False
```

```
4    False
```

```
dtype: bool
```





数据预处理

重复值的处理

drop_duplicates() 方法

drop_duplicates() 方法用于Pandas对象中重复的数据删除。

格式: `pd_obj.drop_duplicates(subset=None, keep='first', inplace=False)`

inplace: 默认为False, 表示删除结果是否替换原来的数据

```
>>> df0=df.copy()
```

```
>>> df
```

	id	name	age	sex
0	1001	susan	16	F
1	1002	kali	17	F
2	1003	davi	18	M
3	1003	davi	18	M
4	1004	jack	16	M

```
>>> df0.drop_duplicates('age', inplace=True)
```

```
>>> df0
```

	id	name	age	sex
0	1001	susan	16	F
1	1002	kali	17	F
2	1003	davi	18	M

```
>>> df.drop_duplicates()
```

	id	name	age	sex
0	1001	susan	16	F
1	1002	kali	17	F
2	1003	davi	18	M
4	1004	jack	16	M





数据预处理

异常值处理

基于 3σ 原则检测异常值

3σ 原则是指假设一组检测数据只含有随机误差，对其进行计算处理得到标准偏差，按一定概率确定一个区间，凡是超过这个区间的误差都是粗大误差，在这误差范围外的数据应予以剔除。

正态分布公式：
$$f(x) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)$$

σ 表示方差， μ 表示平均数。

根据正态分布函数图可知， 3σ 原则大各个区间所占的概率：

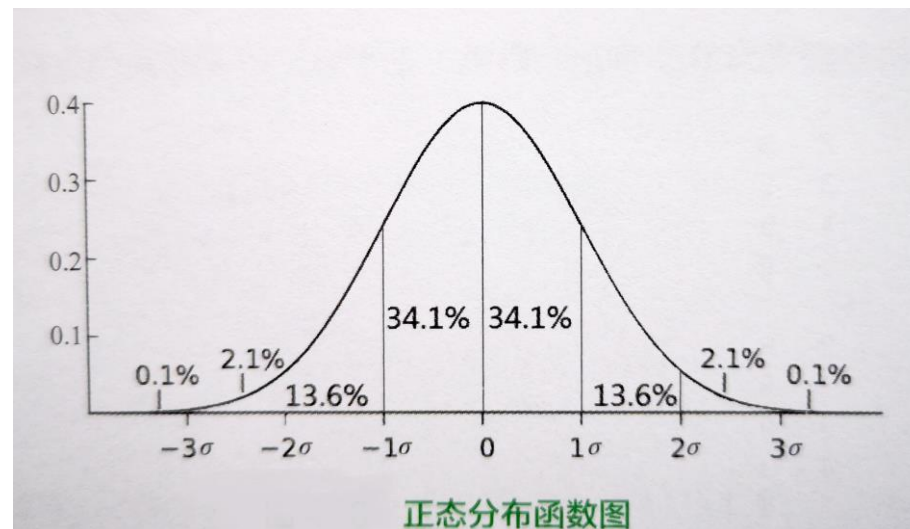
- 数值分布在 $(\mu-\sigma, \mu+\sigma)$ 中的概率为0.682
- 数值分布在 $(\mu-2\sigma, \mu+2\sigma)$ 中的概率为0.954
- 数值分布在 $(\mu-3\sigma, \mu+3\sigma)$ 中的概率为0.997

可见，数值几乎全部集中在 $(\mu-3\sigma, \mu+3\sigma)$ 区间内，超出这个范围的可能性仅占不到0.3%，所以，凡误差之个区间的就属异常值，应予以剔除。

```
>>> df=pd.read_csv('d:/p_train/sjfx.csv')
```

```
>>> three_sigma(df['A'])
```

```
5      450
```



```
>>> def three_sigma(serq):      #自定义检测异常值函数
    m_v=serq.mean()             #求平均值
    s_v=serq.std()              #求标准差
    rule=(m_v-3*s_v>serq) | (m_v+3*s_v<serq)
    index=np.arange(serq.shape[0])[rule]
    #返回异常值的位置索引
    outrange=serq.iloc[index]   #获取异常数据
    return outrange
```





数据预处理

异常值处理

基于箱形图检测异常值

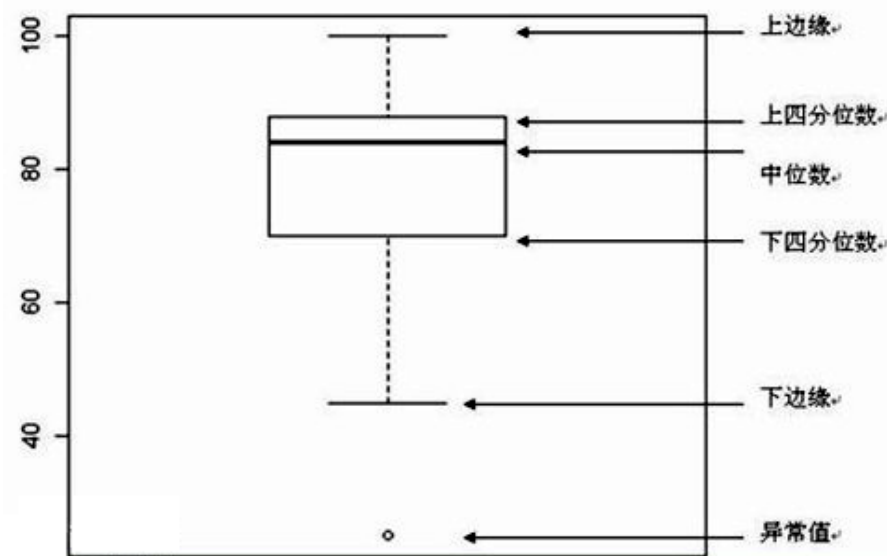
箱型图是一种用作显示一组数据分散情况的统计图。在箱型图中，异常值通常被定义为小于 $Q_L - 1.5Q_R$ 或大于 $Q_U + 1.5Q_R$ 的值

- Q_L 称为下四分位数，表示全部观察中四分之一的数据取值比它小。
- Q_U 称为上四分位数，表示全部观察中四分之一的数据取值比它大。
- Q_R 称为四分位间距，是上四分位 Q_U 与下四分位数 Q_L 之差。其间包含全部观察值的一半。

离散点表示的是异常值，上界表示除异常值以外数据中的最大值，下界表示除异常值以外数据中最小值。为了能从箱型图中查看异常值，Pandas中提供了一个`boxplot()`方法，专门绘制箱型图。

检测出异常值后，首先要进一步确认它们是否为真正的异常值，再从如下四种方式中选择一种处理。

- ✓ 直接将含有异常值的记录删除。
- ✓ 用具体的值来进行替换，可用前后两个观测值的平均值修正该异常值。
- ✓ 不处理，直接在具有异常值的数据集上进行统计分析。
- ✓ 视为缺失值，利用缺失值的处理方法修正该异常值。



箱型图结构示意图



数据预处理

更改数据类型

明确指定数据的类型

创建Pandas对象时，在构造方法中使用dtype参数直接指定数据的类型。

```
>>> ar=np.random.randint(1, 50, 9).reshape(3, 3)
>>> df=pd.DataFrame(ar, columns=list('abc'), dtype=float)
>>> df
```

	a	b	c
0	35.0	40.0	2.0
1	39.0	28.0	26.0
2	24.0	48.0	9.0

```
>>> type(df.iloc[0][0])
<class 'numpy.float64'>
```

astype() 方法强制转换数据类型

```
>>> df['b']=df['b'].astype(int)
>>> df
```

	a	b	c
0	35.0	40	2.0
1	39.0	28	26.0
2	24.0	48	9.0

```
>>> df=df.astype(np.int)
>>> df
```

	a	b	c
0	35	40	2
1	39	28	26
2	24	48	9

```
>>> type(df.iloc[0][1])
<class 'numpy.int32'>
```





数据预处理

Pandas数据分组

groupby() 方法

通过groupby()方法将数据集按照某些标准分成若干个组，返回一个GroupBy对象，注意，该对象并没有进行任何计算，只是包含一些关于分组键的中间数据。。

格式：Pd_obj.groupby(by=None, axis=0, level=None, as_index=True, sort=True)

参数：by：用于确定进行分组的依据。

axis：表示分组的方向，可以为0(表示按行)或1(表示按列)，默认为0。

level：如果某个轴是一个MultiIndex对象(索引层次结构)，则会按特定级别或多个级别分组。

as_index：表示聚合后的数据是否以组标签作为索引的DataFrame对象输入，默认为True。

sort：表示是否对分组标签进行排序，默认为True。

```
>>> k=pd.Series(np.random.choice(list('abc'),10))
>>> df=pd.DataFrame(np.random.randint(1,20,30).reshape(10,3),
                    columns=['da1','da2','da3'])
>>> df['key']=k
```





数据预处理

Pandas数据分组

groupby() 方法

```
>>> df
   da1  da2  da3 key
0     1   11   17   c
1    18    7   19   b
2    18   18   17   a
3    10    4   15   a
4    10    4    4   c
5     4    7   10   c
6     9   18   17   a
7     6    7   13   b
8    14    2   10   b
9     6   19   13   a
```

```
>>> df_gby=df.groupby('key')
>>> for da in df_gby:
    print(da)
```

```
('a',      da1  da2  da3 key
2     18   18   17   a
3     10    4   15   a
6      9   18   17   a
9      6   19   13   a)
('b',      da1  da2  da3 key
1     18    7   19   b
7      6    7   13   b
8     14    2   10   b)
('c',      da1  da2  da3 key
0      1   11   17   c
4     10    4    4   c
5      4    7   10   c)
```





数据预处理

Pandas数据分组

常用的分组计算函数方法

`df_gby_obj.first()`: 非NaN的第一个值
`df_gby_obj.last()`: 非NaN的最后一个值
`df_gby_obj.sum()`: 非NaN的累加和
`df_gby_obj.mean()`: 非NaN的平均值
`df_gby_obj.median()`: 非NaN的算术中位数
`df_gby_obj.count()`: 非NaN的统计个数
`df_gby_obj.min()`: , 非NaN的最小值
`df_gby_obj.max()`: , 非NaN的最大值
`df_gby_obj.std()`: 非NaN的标准差
`df_gby_obj.var()`: 非NaN的方差
`df_gby_obj.prod()`: 非NaN的累乘积

```
>>> df_gby.sum()
      da1  da2  da3
```

```
key
a      43   59   62
b      38   16   42
c      15   22   31
```

```
>>> df_gby.mean()
```

```
      da1      da2      da3
key
a  10.750000  14.750000  15.500000
b  12.666667   5.333333  14.000000
c   5.000000   7.333333  10.333333
```

```
>>> df_gby.std()
```

```
      da1      da2      da3
key
a   5.123475   7.182154   1.914854
b   6.110101   2.886751   4.582576
c   4.582576   3.511885   6.506407
```





Pandas数据合并

主键合并merge()函数

主键合并类似于关系型数据库的连接方式，它是指根据一个或多个键将不同的DataFrame对象连接起来，通常是将两个DataFrame对象中**重叠的列**作为**合并的键**。

格式：`pd.merge(left, right, how= 'inner' , on=None, left_on=None, right_on=None, left_index=False, right_index=False, sort=False)`

参数：

- `left`: 参与合并的左侧DataFrmae对象。
- `right`: 参与合并的右侧DataFrmae对象。
- `how`: 表示连接方式，默认为inner(内连接), 还可选left、right、outer。
- `on`: 用于连接的列名，必须存在于左右两个DataFrame对象中。
- `left_on`: 以左侧DataFrame做为连接键。
- `right_on`: 以右侧DataFrame做为连接键。
- `left_index`: 左侧的行索引用作连接键。
- `right_index`: 右侧的行索引用作连接键。
- `sort`: 是否排序，默认为False。

使用merge()函数进行合并时，默认使用重叠的列索引做为合并键，并采用内连接方式合并数据。





数据预处理

案例



读入分别读入两个数据文件stcj1.xlsx和stcj2.xlsx，如右图，将它们以学号作为主键连接一起，并去除多余的数据。

```
>>> import pandas as pd
>>> df1=pd.read_excel('stcj1.xlsx')
>>> df1.head() #默认前5行
```

	学号	姓名	性别	写作	英语	逻辑	计算机
0	1001	马红丽	女	89	89	92	95
1	1002	刘绪	女	79	81	95	80
2	1003	付艳丽	女	86	93	94	91
3	1004	张建立	男	82	88	92	77
4	1005	魏翠香	女	83	97	94	93

```
>>> df2=pd.read_excel('stcj2.xlsx')
>>> df2.head()
```

	学号	姓名	性别	体育	法律	思想理论	操行分
0	1001	马红丽	女	72	93	94	20.0
1	1002	刘绪	女	73	88	92	19.5
2	1003	付艳丽	女	73	97	94	19.0
3	1004	张建立	男	80	84	84	18.5
4	1005	魏翠香	女	80	90	92	18.0

stcj1.xlsx

	A	B	C	D	E	F	G
1	学号	姓名	性别	写作	英语	逻辑	计算机
2	1001	马红丽	女	89	89	92	95
3	1002	刘绪	女	79	81	95	80
4	1003	付艳丽	女	86	93	94	91
5	1004	张建立	男	82	88	92	77
6	1005	魏翠香	女	83	97	94	93
7	1006	赵晓娜	女	75	84	84	85
8	1007	刘宝英	女	82	90	92	84
9	1008	郑会锋	女	85	77	96	91
10	1009	申永琴	女	72	96	88	91
11	1010	许宏伟	女	90	90	89	86

stcj2.xlsx

	A	B	C	D	E	F	G
1	学号	姓名	性别	体育	法律	思想理论	操行分
2	1001	马红丽	女	72	93	94	20
3	1002	刘绪	女	73	88	92	19.5
4	1003	付艳丽	女	73	97	94	19
5	1004	张建立	男	80	84	84	18.5
6	1005	魏翠香	女	80	90	92	18
7	1006	赵晓娜	女	80	77	96	18.5
8	1007	刘宝英	女	81	96	88	19
9	1008	郑会锋	女	74	90	89	19.5
10	1009	申永琴	女	75	90	89	20
11	1010	许宏伟	女	82	70	70	20.5





案例



读入分别读入两个数据文件stcj1.xlsx和stcj1.xlsx，将它们以学号作为主键连接一起，并去除多余的数据。

```
>>> df=pd.merge(df1,df2,on='学号')
```

```
>>> df.head()
```

	学号	姓名_x	性别_x	写作	英语	逻辑	计算机	姓名_y	性别_y	体育	法律	思想理论	操行分
0	1001	马红丽	女	89	89	92	95	马红丽	女	72	93	94	20.0
1	1002	刘绪	女	79	81	95	80	刘绪	女	73	88	92	19.5
2	1003	付艳丽	女	86	93	94	91	付艳丽	女	73	97	94	19.0
3	1004	张建立	男	82	88	92	77	张建立	男	80	84	84	18.5
4	1005	魏翠香	女	83	97	94	93	魏翠香	女	80	90	92	18.0

```
>>> df.drop(axis=1,columns=['姓名_y','性别_y'],inplace=True)
```

```
>>> df.rename(columns={'姓名_x':'姓名','性别_x':'性别'},inplace=True)
```

```
>>> df.head()
```

	学号	姓名	性别	写作	英语	逻辑	计算机	体育	法律	思想理论	操行分
0	1001	马红丽	女	89	89	92	95	72	93	94	20.0
1	1002	刘绪	女	79	81	95	80	73	88	92	19.5
2	1003	付艳丽	女	86	93	94	91	73	97	94	19.0
3	1004	张建立	男	82	88	92	77	80	84	84	18.5
4	1005	魏翠香	女	83	97	94	93	80	90	92	18.0

```
>>> df.to_excel('stcj.xlsx',index=False)
```





案例



按”性别”分组，分别统计出男、女生的人数，每门课的最高分、最低分和平均分。

```
>>> df_gpb=df.groupby('性别')
>>> df_gpb.count().loc[:,'姓名']
```

性别

女 13

男 41

Name: 姓名, dtype: int64

```
>>> df_gpb.max().iloc[:,2:]
```

写作 英语 逻辑 计算机 体育 法律 思想理论 操行分

性别

女 90 97 97 95 82 97 96 20.5

男 91 97 93 96 87 97 93 23.0

```
>>> df_gpb.mean().iloc[:,2:]
```

英语 逻辑 计算机 体育 法律 思想理论 操行分

性别

女 87.307692 92.000000 88.692308 76.692308 84.769231 88.307692 19.192308

男 79.585366 76.731707 81.902439 74.902439 75.902439 78.756098 20.024390





用蒙特卡洛方法研究社会财富的随机分配



问题的提出：有人提出这样一个有趣的问题：房间里有10个人，每人都有10元钱，他们反复玩一个游戏，每轮游戏中，每个人都要拿出一元钱随机给另一个人，10次后，10个人的财富分布是怎样的？50次后呢？

模拟结果

	0	1	2	3	4	5	6	7	8	9	10
person1	10	10	10	11	11	11	11	11	10	9	9
person2	10	10	11	12	13	13	13	13	12	12	12
person3	10	10	10	9	8	8	8	9	9	9	9
person4	10	11	10	10	10	11	11	12	11	10	10
person5	10	10	10	9	8	9	8	7	9	8	8
person6	10	9	8	10	11	10	12	13	15	15	14
person7	10	9	9	8	7	6	6	5	6	7	7
person8	10	11	12	12	11	12	11	11	10	12	11
person9	10	11	11	10	10	10	10	10	10	11	12
person10	10	9	9	9	11	10	10	9	8	7	8

每次游戏的变化

	上次结存	本次给出	给出对象	给出收到	本次增减	本次结存
person1	10	1	P2	1	0	10
person2	10	1	P8	1	0	10
person3	10	1	P4	1	0	10
person4	10	1	P9	2	1	11
person5	10	1	P1	1	0	10
person6	10	1	P3	0	-1	9
person7	10	1	P9	0	-1	9
person8	10	1	P4	2	1	11
person9	10	1	P5	2	1	11
person10	10	1	P8	0	-1	9





数据处理分析案例

用蒙特卡洛方法研究社会财富的随机分配

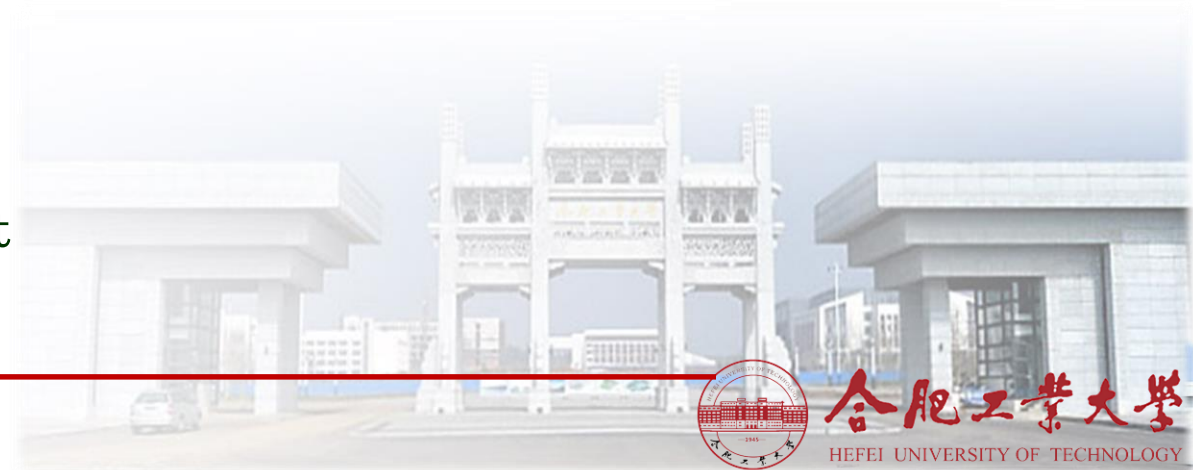


模型假设：

1. 有100个人，每个人初始基金100元；
2. 从18岁到65岁，简化运算约17000天；
3. 每天每人捐出一元钱，随机分配给任何一人；
4. 当某人的财富值降至0元时，继续负债捐出1元钱，
5. 到65岁时，各人拥有的财富情况是怎样的？
6. 分别统计出30岁、40岁、50岁、60岁时各人财富的变化情况。
7. 如果当某人的财富值降至0元时，将不再捐出1元钱，但仍能继续收到别人捐出的钱，那么，5、6的变化结果会是怎样的？

#导入功能模块库

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```



合肥工业大学
HEFEI UNIVERSITY OF TECHNOLOGY

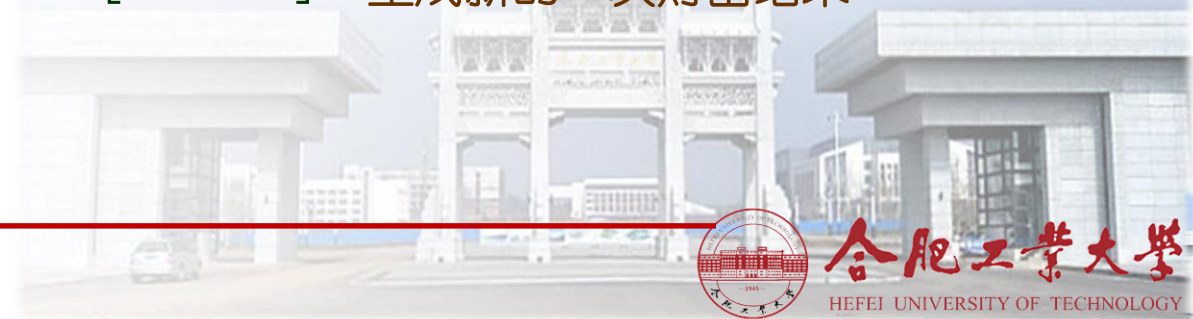


数据处理分析案例

用蒙特卡洛方法研究社会财富的随机分配

```
def get_change(fort, n):  
    roud=pd.DataFrame({'pre_fort':fort[n], 'lost':1})  
    #roud['lost'][roud['pre_fort']<1]=0  
    gain=pd.Series(np.random.choice(fort.index, 100))  
    obtain=pd.DataFrame({'souru':gain.value_counts()})  
    obtain.index.name='id'  
    roud=pd.merge(roud, obtain, on='id', how="left")  
    roud.fillna(0, inplace=True)  
    roud['souru']=roud['souru'].astype(int)  
    fort[n+1]=roud['pre_fort']-roud['lost']+roud['souru']  
    return fort
```

#每人上次的财富值及默认支出1元
#上次财富为0时，支出更改为0元
#产生100个可重复随机获得1元钱的人
#对获得1元钱的人进行次数统计
#将索引命名为id, 以便和roud连接
#将索引为主键，以roud为主进行连接
#将缺失值NaN更改为0
#将数据类型更改为整数
#生成新的一次财富结果





数据处理分析案例

用蒙特卡洛方法研究社会财富的随机分配

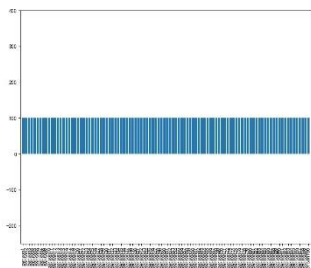
```
def get_f(n):
    person_n=['person'+str(i) for i in range(1,101)] #生成100人编号
    #以人编号为索引，生成每人100初始财富值
    fort=pd.DataFrame({n:[100 for i in range(100)]},index=person_n)
    fort.index.name='id' #指定索引的名称
    return fort

def main():
    n=0
    fortune=get_f(n)
    for i in range(17000):
        fortune=get_change(fortune,n)
        n+=1
        print("正在运行第%d次模拟...."%n)
    #fortune.to_excel("fortune.xlsx")
    forT=fortune.T #转置矩阵
    forT.to_excel("forT.xlsx")
main()
```

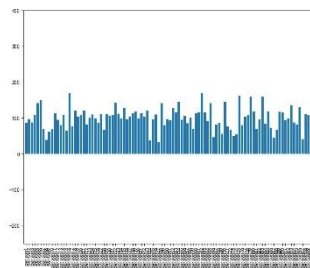




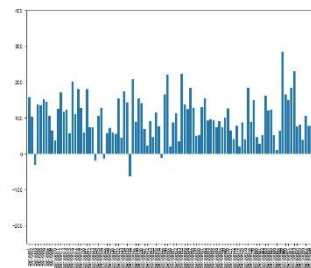
用蒙特卡洛方法研究社会财富的随机分配



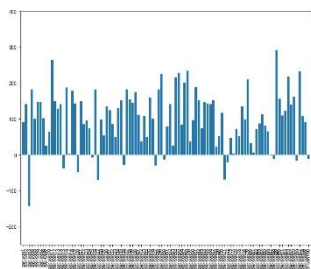
18岁时财富值



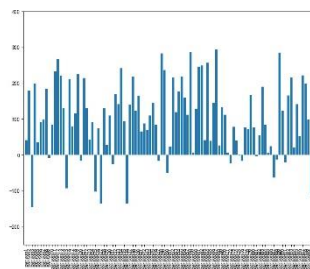
20岁时财富值



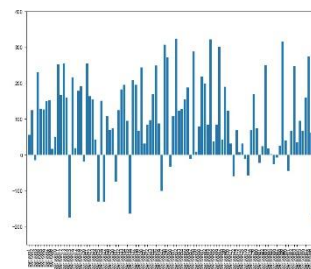
30岁时财富值



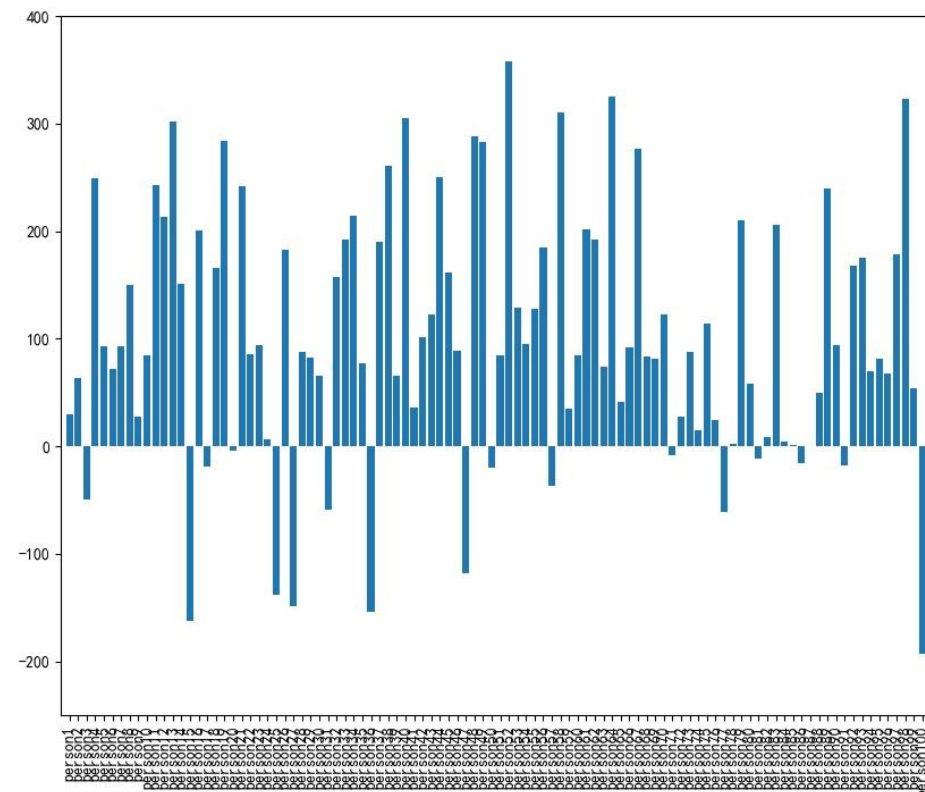
40岁时财富值



50岁时财富值



60岁时财富值

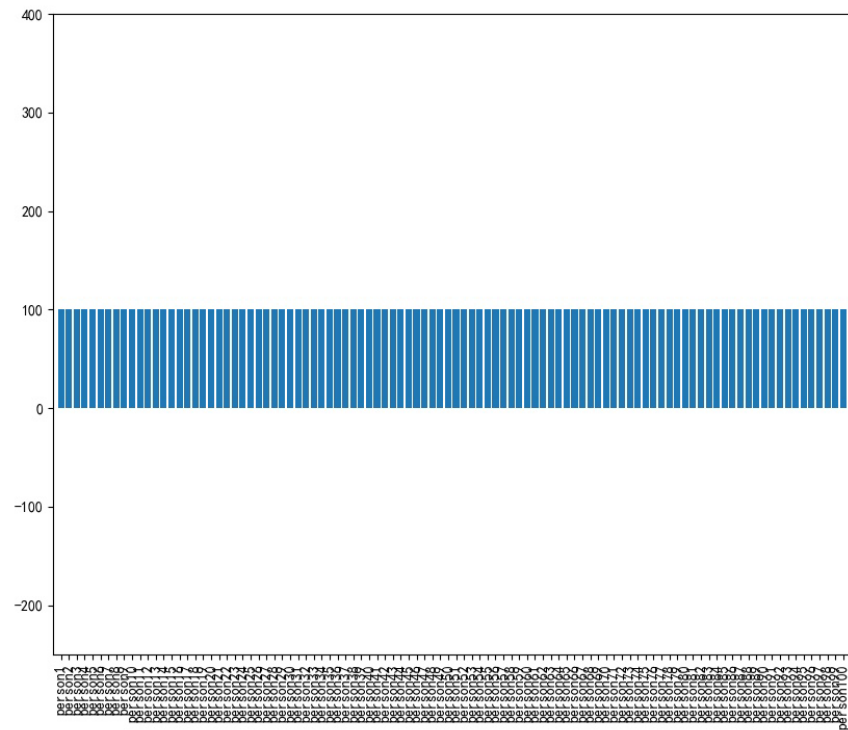
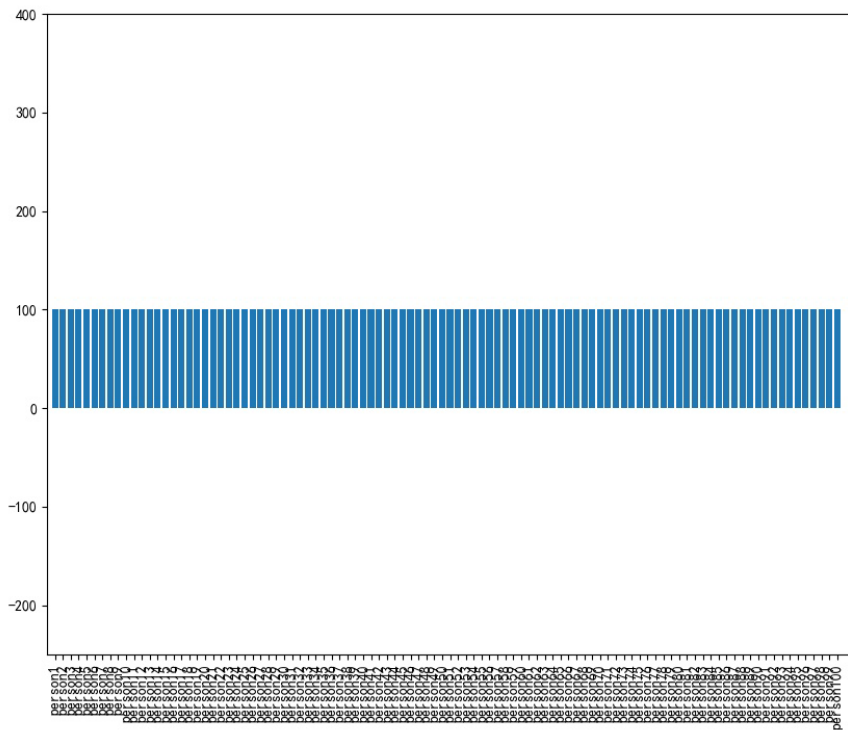




数据处理分析案例



用蒙特卡洛方法研究社会财富的随机分配





合肥工业大学
HEFEI UNIVERSITY OF TECHNOLOGY

祝学习进步!



计算机基础教育研究所

2021.3