**金融贷款数据的清洗**

# 1简介

本案例使用一份真实的Lending Club贷款数据集，展示如何使用Pandas等工具对数据进行清洗。包括如何进行缺失值的检测和处理，如何进行异常数据的检测和处理，如何检测数据中的重复值等。

本案例主要背景为贷款情况审查。银行可以通过个人贷款状况对个人信用进行分类，从而更好地避免金融诈骗的发生。

# 2准备

## 2.1实验环境搭建

本实验使用Anaconda + Jupyter Notebook环境。

导入类库，并对名字较长的类库取简短的别名，便于使用。

*# 导入类库*import pandas as pd  
import numpy as np  
import matplotlib.pyplot as plt  
import matplotlib

from sklearn import preprocessing

import seaborn as sns  
import re

## 2.2数据准备

在电脑的D盘中创建input和output文件夹，其中input文件夹用来存放要读入的数据集，output存放输出数据集，随后将实验数据集拷贝到input文件夹内。

本案例所选的数据集是来自LendingClub中统计的2019年第三季度的借贷数据。

本数据集共143035行、150列，因为数据列数较多，此处选取具有代表性的属性介绍其含义，具体的属性描述可以查看源数据网站的数据字典。

| **列名** | **含义说明** |
| --- | --- |
| id | 为贷款列表分配的唯一信用ID。 |
| member\_id | 为借款人成员分配的唯一信用证ID。 |
| loan\_amnt | 借款人申请贷款的金额。如果在某个时间点，信贷部门减少贷款金额，那么它将反映在这个值中。 |
| funded\_amnt | 在某时间点承诺的贷款总额。 |
| funded\_amnt\_inv | 当时投资者为该笔贷款承诺的总金额。 |
| term | 贷款月份。值以月为单位，可以是36或60。 |
| int\_rate | 贷款利率 |
| installment | 如果贷款发放，借款人每月所需要还款的数额。 |
| grade | 信用证指定贷款等级 |
| sub\_grade | 信用证指定贷款基础 |
| emp\_title | 职业 |
| emp\_length | 就业年限（年）。可能的值介于0和10之间，其中0表示一年以下，10表示十年或十年以上。 |
| home\_ownership | 借款人在登记期间提供的或从信贷报告中获得的房屋所有权状况。其值为：租金、自有、抵押、其他 |
| hardship\_payoff\_balance\_amount | 困难计划开始日期的收支差额 |
| hardship\_last\_payment\_amount | 截至困难计划开始日期的最后一笔付款金额 |
| disbursement\_method | 借款人获得贷款的方式。可能的值是：现金，直接支付 |
| debt\_settlement\_flag | 标记已注销的借款人是否与债务结算公司合作。 |
| debt\_settlement\_flag\_date | 设置债务结算标志的最新日期 |
| settlement\_status | 借款人结算计划的状态。可能的值有：完成、进行、中断、取消、拒绝 |
| settlement\_date | 借款人同意结算计划的日期 |
| settlement\_amount | 借款人同意偿还的贷款金额 |
| settlement\_percentage | 结算金额占贷款支付余额的百分比 |
| settlement\_term | 借款人在结算计划中的月数 |

## 2.3数据读入

实验数据集为较常见的csv格式，其格式全称为comma-separated value（逗号分隔值），一般每条记录占一行，记录之间用换行符分开；记录内的不同属性以逗号为分隔符。

时常用到的数据格式除了csv还有txt文件和xlsx（Excel）文件，txt文件默认同样一条记录占一行，记录之间用换行符分开，但记录不同属性之间默认使用制表符【TAB】为分隔符；xlsx文件则很简单，一条记录占一行，每个属性占一列。

大家可以分别尝试使用pandas的read\_table、read\_excel和read\_csv方法读取txt文件、xlsx文件和csv文件，尝试正确的对应关系后，使用其他两个方法打开文件，看看会发生什么。

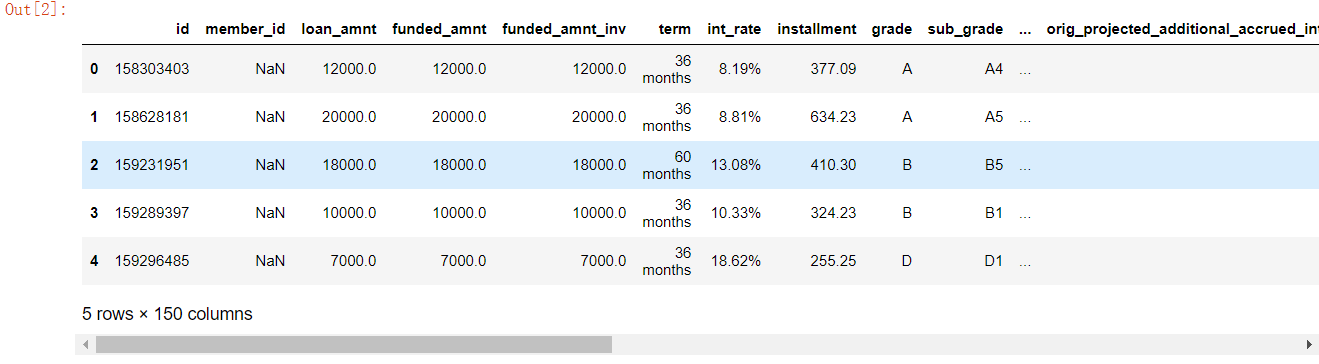
使用Pandas的read\_csv()函数读取数据，并查看数据。

对read\_csv()函数，可以设置多个属性，对常用的几个属性的说明如下：

|  |  |
| --- | --- |
| read\_csv | |
| filepath | 文件存储路径，可以用r""进行非转义限定，路径最好是纯英文（文件名也是） |
| encoding | pandas默认编码是utf-8，如果同样读取默认uft-8的txt或者json格式，则可以忽略这个参数，如果是csv，且数据中有中文时，则要指定encoding=‘gbk’ |
| sep | 指定分割符形式，CSV默认逗号分割，可以忽略这个参数，如果是其它分割方式，则要填写 |
| skiprows | 跳过多少行再读取数据，通常是数据不太干净，需要去除掉表头才会用到 |
| nrows | 仅读取多少行，后面的处理也都仅限于读取的这些行 |

*# 使用Pandas的read\_csv()函数读取数据*dataset = pd.read\_csv(**"D:/input/LoanStats\_securev1\_2019Q3.csv"**, encoding=**"gbk"**, low\_memory=False)

*# 查看前五行数据*  
dataset.head(5)



查看数据集中行与列数量。

dataset.shape



调用read\_csv方法，返回的是一个DataFrame对象。

DataFrame 可以理解是一个二维数组,是一个表格型的数据结构,它包含了有序的列。其列的值 ,它可以是字符串、数值或者是布尔值等，从行和列看，DataFrame是一个带有标签的二维数组

# 3缺失值处理

## 3.1缺失值查看

首先介绍在数据缺失值处理中所经常用到的函数isnull、dropna、fillna和notnull。

其中isnull函数是元素级别的判断，把对应的所有元素的位置都列出来，元素为空或者NaN就显示True，否则就是False。

dropna用来删除有缺失值的函数，默认为删除任何含有缺失值的行。

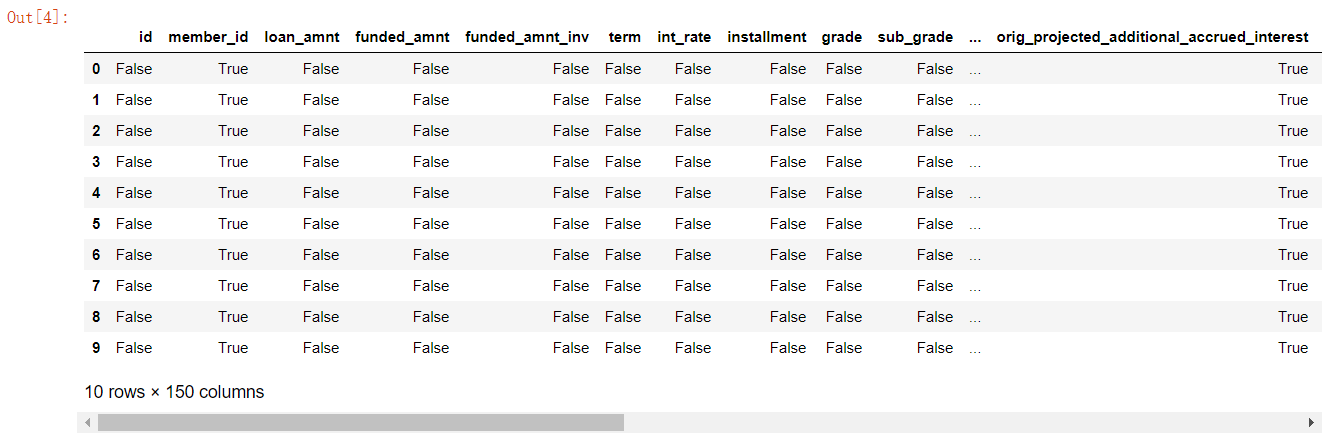
fillna函数用来进行缺失值填补，具体的使用方法在第3.3和3.4节有详细的说明。

notnull可以说是isnull的反面，同样是元素级别的判断，元素为空或者NaN就显示False，否则就是True。

一般来说，在进行数据清洗的时候会先使用isnull函数来查看对应的缺失值所对应的地方，如果直接使用isnull函数来对数据进行缺失值的直接查看，那么返回一个布尔类型数据集，该数据集与原始数据格式相同，例如一个数据集使用了Numpy的数组存储，那么返回的就是含有布尔值的数组，如果使用的是Pandas的DataFrame存储，那么返回的就是含有布尔值的DataFrame。

下面使用isnull查看缺失数据

*# 查缺失值的前10行数据*dataset.isnull().head(10)



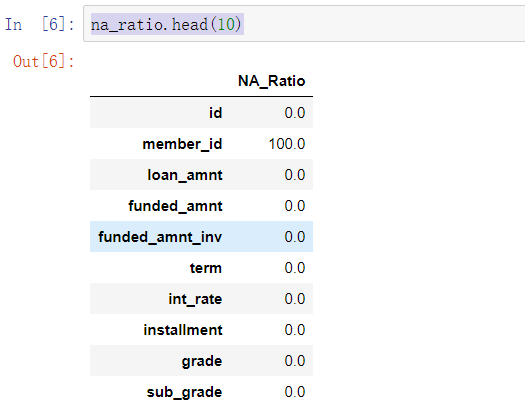
可见直接调用isnull()函数处理大数据集只能得到一个存有True或者False的数据集，结果并不直观，需要对结果进一步处理。查看数据中缺失值数量所占总数据量的百分比，从而使结果更加直观，以便进一步处理缺失值。

*# 创建一个新的DataFrame数据表来存储每列数据中缺失值所占的百分比*

*# 此数据表只有两列，一列为原来的属性名，另一列为缺失值所占百分比，命名为NA\_Ratio*na\_ratio = pd.DataFrame(dataset.isnull().sum() / len(dataset) \* 100, columns=[**'NA\_Ratio'**])

可以使用head()函数查看na\_ratio格式

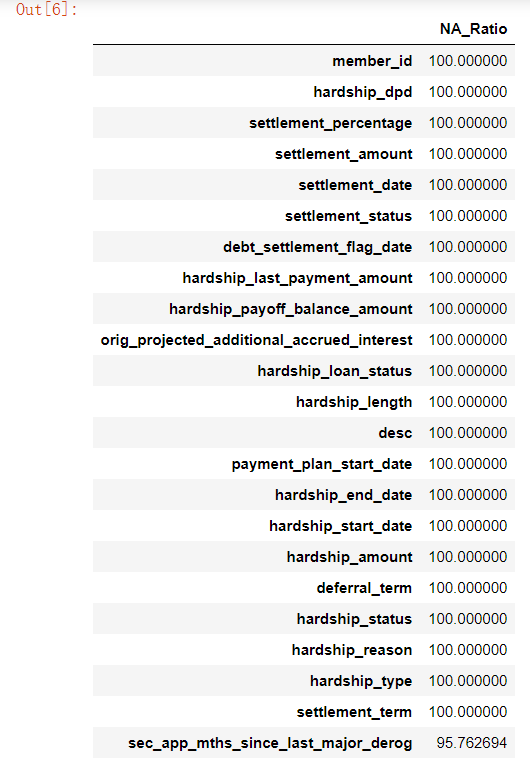
na\_ratio.head(10)



由于数据列过多，选取查看缺失值占总数据百分比大于90%的列名及其数值。代码中sort\_values函数即按行索引进行排序，by='NA\_Ratio'指定根据NA\_Ratio列进行排序，ascending=False表示按照降序排列。大家可以尝试变更参数，查看结果的变化。

df\_na = na\_ratio[na\_ratio[**'NA\_Ratio'**] >= 90].sort\_values(by=**'NA\_Ratio'**, ascending=False)

*# 查看结果，如果使用pycharm等其他运行环境，需要使用print（df\_na）代替*  
df\_na



大家也可以将中间数据df\_na导出，打开文件查看统计结果。常用的数据保存方法有两种，分别为Python原生的方法和pandas包的方法，此处使用较为简单的pandas方法，对两种方法的具体说明则放到文章的最后一节。

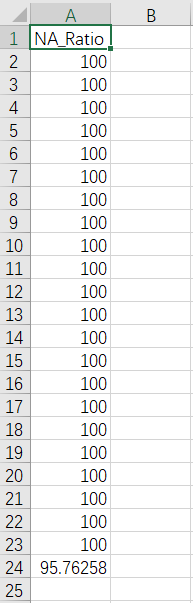
*# 将中间结果保存为csv文件，存到output文件夹*

*# 文件不需要提前创建，由pandas自动生成，若已存在该文件，则已存在文件会被覆盖*df\_na.to\_csv(**"D:/output/df\_na.csv"**,index = False)

运行后，即可在之前创建的output文件夹中找到它。



双击打开文件，查看内容。



可见还是有较多的列几乎全部为空，对于这些列一般都可以简单地认为它在数据分析中提供极有限信息，所以可以直接删除。

由于数据列过多，选取缺失值占总数据百分比大于0.01%，小于80%的列名及其数值显示到图上。

Python提供的绘图工具包有很多，其中数据科学领域较为常用的主要有Matplotlib、Seaborn等。

其中Matplotlib是python最著名的绘图库，它提供了一整套和MATLAB相似的命令API，十分适合交互式地进行制图。而且也可以方便地将它作为绘图控件，嵌入GUI应用程序中，通过简单的绘图语句，就可以绘制出高质量的图。

Seaborn的绘图是建立在 Matplotlib 之上的，实际上使用Seaborn绘图时，会在底层调用Matplotlib代码，因此，这两个工具包在美化方面是相似的，自定义图时用的语法也都非常相似。

我们在这里使用Matplotlib进行绘图。首先生成绘图所用的数据集df\_na

df\_na = na\_ratio[(na\_ratio[**'NA\_Ratio'**] < 80) & (na\_ratio[**'NA\_Ratio'**] >= 0.01)].sort\_values(by=**'NA\_Ratio'**, ascending=False)  
*# 绘图，使用subplots函数创建图像，*

*# 其中figsize用来设置图形的大小，20为图形的宽， 10为图形的高，单位为英寸。*fig, ax = plt.subplots(figsize=(20, 10))

*# barplot指定图形为条形图，绑定x轴和y轴的数据*

*# orient='v'表示竖直展示即数值条方向；palette：调色板，控制不同的颜色风格*  
nar = sns.barplot(x=df\_na.index, y=df\_na[**'NA\_Ratio'**], orient=**'v'**, palette=**"Blues\_d"**)

*# set\_xticklabels设置x轴标签的旋转*  
ax.set\_xticklabels(df\_na.index, rotation=90, fontsize=20)

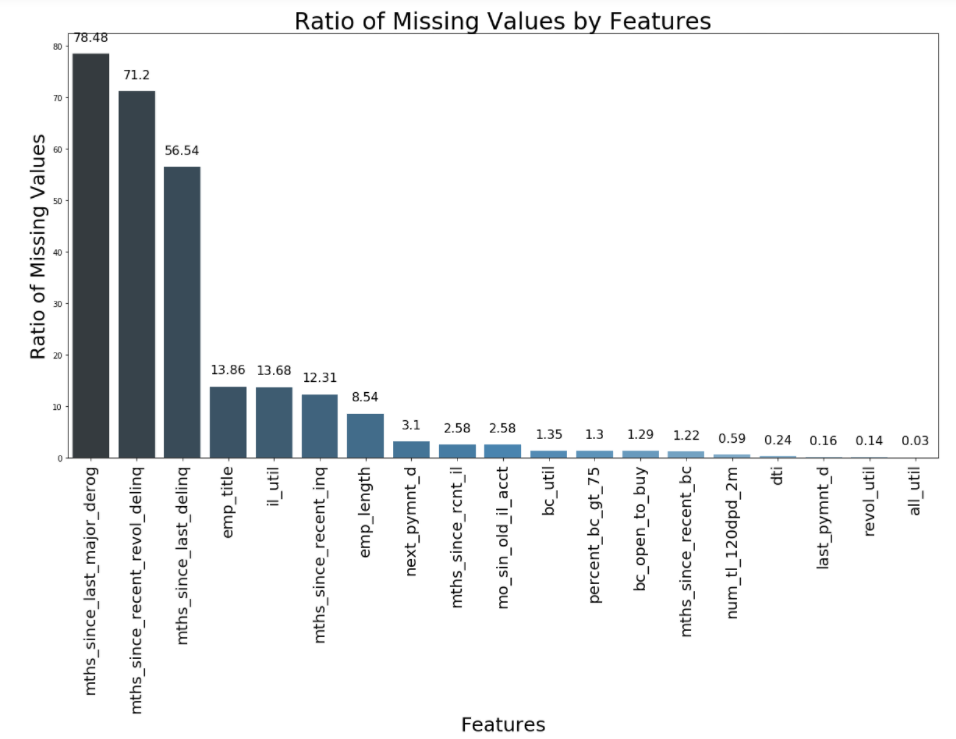
*# 设置y轴的坐标轴名称*  
ax.set\_ylabel(**'Ratio of Missing Values'**, fontsize=25)

*# 设置x轴的坐标轴名称*  
ax.set\_xlabel(**'Features'**, fontsize=25)

*# 设置整个图的名称*  
ax.set\_title(**'Ratio of Missing Values by Features'**, fontsize=30)

*# 设置每个数据条上的小数表示*  
for loc, value in zip(ax.patches, df\_na.NA\_Ratio):  
 ax.text(loc.get\_x() + loc.get\_width() / 2, loc.get\_height() + 2, round(value, 2), ha=**'center'**, va=**'bottom'**, fontsize=16)

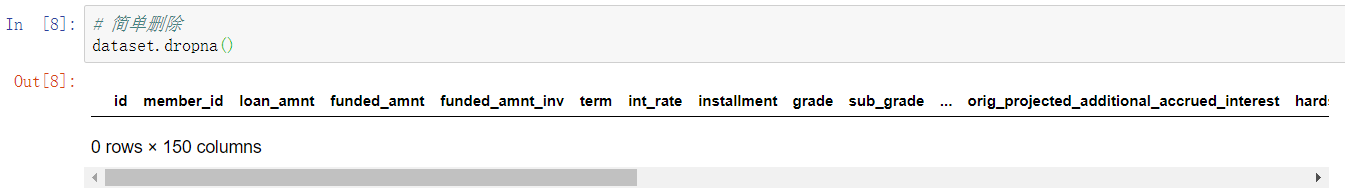
*# 显示图像*  
plt.show()



由此可见，部分数据中缺失值较多，故可以尝试使用多种方法删除或填补。

## 3.2缺失值删除

一般来说，删除缺失值所用的函数是dropna()，其原理是删除带有任何存有缺失值的行，对于真实数据集中不同列有不同的缺失值存在的地方，甚至可能有某一列全是缺失值，简单使用dropna()函数就会直接得到如下的结果。

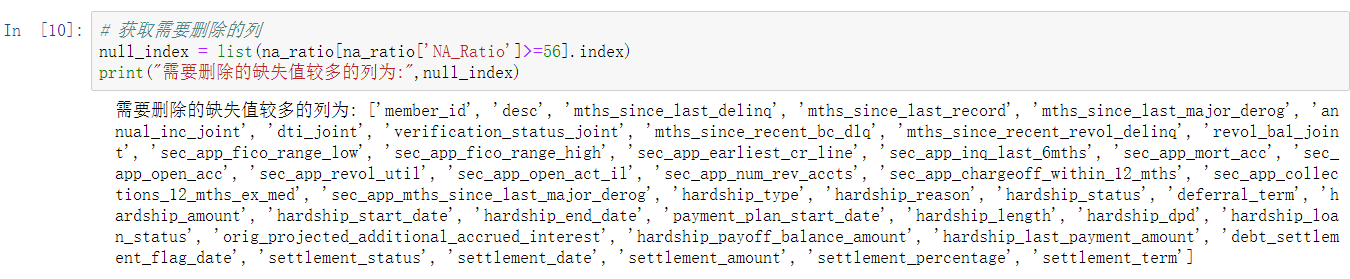


此时将文件输出后打开，可发现文件的数据部分为空。

dataset.to\_csv(**"D:/output/test1.csv"**,index = False)

依据上面的表可以发现，简单使用dropna()函数只会让数据结果无效化，不仅删除了缺失值的信息，也删除了所有的暂时不需要删除的信息，故而依据之前所查看缺失值画的柱状图，可以发现缺失值比例在（0.01%，80%）的列中，除3列数据缺失值在56%以上，其余列数据的缺失值均小于17%，故可以简单认为在此数据集中缺失值在56%以上的数据列提供信息有限，故将缺失百分比56%以上的列数据全部删除。

null\_index = list(na\_ratio[na\_ratio[**'NA\_Ratio'**] >= 56].index)  
print(**"需要删除的缺失值较多的列为:"**, null\_index)



使用copy()函数拷贝一份原数据的副本进行处理，以免接下来的步骤出错导致需要重新读取数据。处理完毕后查看新数据集行与列的情况以确认删除成功。

*# 使用copy()函数拷贝一份原数据的副本*dataset\_copy = dataset.copy()  
*# 使用drop()函数直接删除先前得到的要删除的列*

*# axis=0时为删除行；axis=1为删除列。inplaces是否替换原来的dataframe*dataset\_copy.drop(null\_index, axis=1, inplace=True)

*# 打印此时数据的大小*  
dataset\_copy.shape



在处理完毕后，可以发现行的数量没有改变，列数量减少到了102列，故而删除成功。

此时可以保存一个中间副本到output文件夹。

*# 将中间结果保存为csv文件，存到output文件夹*dataset\_copy.to\_csv(**"D:/output/copy.csv"**,index = False)



可以在output文件夹找到保存的文件。

## 3.3简单填补

删除掉含有较多缺失值的列后，需要填补剩余的包含缺失值的列。填补方法有很多，常用的方法有固定值填补、均值填补、中位数填补、众数填补、后值向前填补、前值向后填补、插值填补、基于模型的填补等。在接下来的实验中，我们将依次尝试几种填补方法。

我们一般将固定值填补、均值填补、中位数填补、众数填补合称为简单填补，这也是我们要首先掌握的填补方法。

固定值填补，顾名思义选取某个固定值/默认值填充缺失值，一般适用于不能用其他取值替换的情况，例如：“工作”属性为空，很可能代表没有工作，当然不能用其他值代替。

均值填补，适用于连续数值型数据，将这一列的平均值填补到空缺处；

众数填补，适用于离散型数据，将这一列除空缺外的众数填补到空缺处；

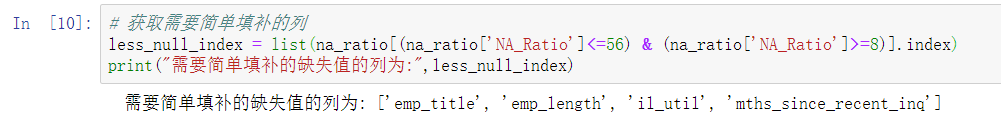
中位数填补，将此列的中位数填补到空缺处。

这三种方法胜在实现简单，意义明确，但会使数据过分地集中于平均值或众数上，导致特征的方差被低估，且由于没有考虑特征之间的关系大大弱化特征间的相关性。

使用fillna()函数填补缺失值，该函数能自动定位到所有缺失值所在的位置，并将其补齐。Fillna函数有多个输入参数，各个参数的说明如下表。

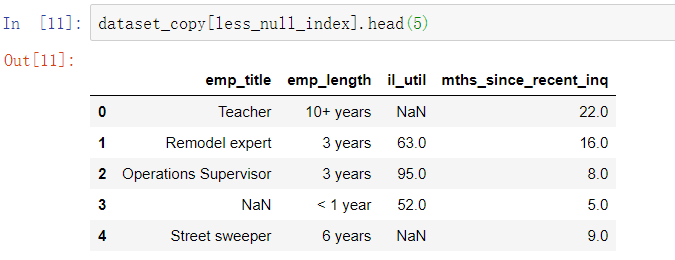
| **参数** | **说明** |
| --- | --- |
| value | 用于填充缺失值的标量值或字典对象 |
| method | 插值方式，如果函数调用时未指定其他参数的话，默认为ffill |
| axis | 带填充的轴。默认为0,代表行，1代表列 |
| inplace | 是否修改调用者对象而不产生副本，True代表修改，False代表不修改 |
| limit | （对于前向和后向填充）可以连续填充的最大数量 |

*# 选出需要简单填补的列*less\_null\_index = list(na\_ratio[(na\_ratio[**'NA\_Ratio'**] <= 56) & (na\_ratio[**'NA\_Ratio'**] >= 8)].index)  
print(**"需要简单填补的缺失值的列为:"**, less\_null\_index)

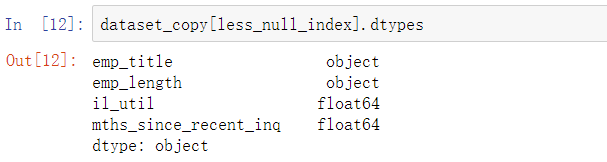


首先分别查看这四列的信息，以方便之后进行填补。

dataset\_copy[less\_null\_index].head(5)



dataset\_copy[less\_null\_index].dtypes



通过查看可以得知，这四列数据分别为字符型以及浮点型数据，对这四列分别进行不同的办法的填补。

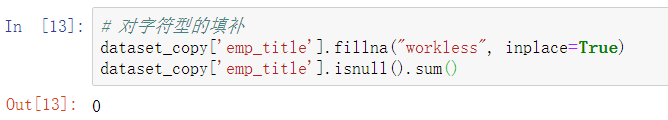
DataFrame求其一列，即Series对象中的均值的方法为mean，众数的方法为mode，中位数的方法为median。

### 3.3.1固定值填补

emp\_title列的意思是借款人在申请贷款时提供的职务，此处为空的情况下不能简单的进行填补,因为有可能空值代表该人无职业或者职业不明确，也没有好的办法对这些缺失值的职业进行判断，此处使用哑变量的思路进行填补，对缺失值进行新属性的添加，设置其为workless即可。

*# 对字符型的填补*dataset\_copy[**'emp\_title'**].fillna(**"workless"**, inplace=True)

*# 查看emp\_title列的空缺值数量*  
dataset\_copy[**'emp\_title'**].isnull().sum()



可以看到在进行哑变量的填补之后，此列的缺失值被填补完毕。

### 3.3.2中位数填补

emp\_length列是所有贷款帐户用户的工作年限，此处尝试使用中位数的方法进行填补。

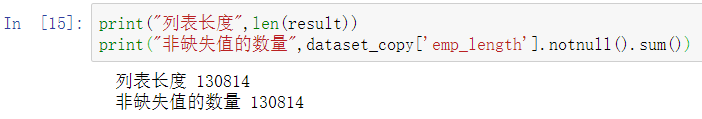
首先查看该列的中位数是多少，由于该列存储的是字符串数据，且都为年份,但是表达的含义是数值型，故而先使用正则表达式将其匹配转换为数值型，然后求其中位数。对应代码如下：

compile 函数用于编译正则表达式，生成一个 Pattern 对象。

*# \d是匹配数字字符[0-9]，+匹配一个或多个，放在一起是匹配一个或多个数字字符*pattern = re.compile(**r'\d+'**)  
*# 初始化一个列表*result = []  
*# 因为有缺失值的存在，会使得正则表达式失败,使程序报错退出，故而引入异常检测来规避缺失值*for i in list(range(0, len(dataset\_copy[**'emp\_length'**]))):  
 try:  
 replace\_number = int(pattern.findall(dataset\_copy[**'emp\_length'**][i])[0])  
 result.append(replace\_number)  
 *# 此处可以直接对原DataFrame的数据进行替换，若此处进行替换操作，会因为DataFrame拷贝操作而导致耗时极长* except:  
 pass

查看一下是否将所有的非缺失值存储到列表中了，此处使用notnull()函数来查看所有的非缺失值数量。

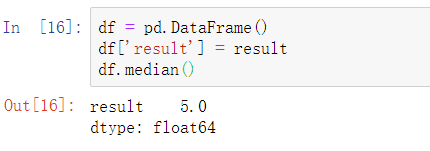
print(**"列表长度"**, len(result))  
print(**"非缺失值的数量"**, dataset\_copy[**'emp\_length'**].notnull().sum())



新建一个DataFrame来存储新的数据，由此新的DataFrame来计算得到所需的中位数的值，再填补回原数据中。在此使用median函数来求得该列的中位数。

*# 创建一个dataframe，用来存放之前正则表达式得到的数字*df = pd.DataFrame()  
df[**'result'**] = result

*# 使用median函数来求得该列的中位数*  
df.median()

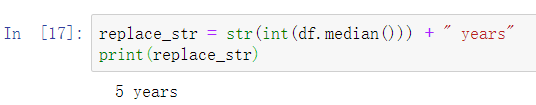


可见其中位数的值是5，也代表若将客户的工作年限从小到大排列，其中间的工作年限是5年。将信息补完，再填补回原数据中。此处使用简单的字符串粘贴即可。

*# 创建要填补的字符串“5 years”*

replace\_str = str(int(df.median())) + **" years"**

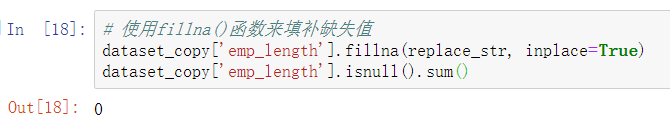
*# 打印字符串*print(replace\_str)



可见生成的字符串正确，使用fillna()函数来填补缺失值。

*# 使用fillna()函数来填补缺失值*dataset\_copy[**'emp\_length'**].fillna(replace\_str, inplace=True)

*# 查看本列的缺失值数量*  
dataset\_copy[**'emp\_length'**].isnull().sum()

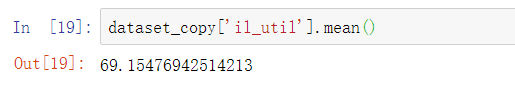


可见该列缺失值的数量为0，填补成功。

### 3.3.3均值填补

il\_util列是所有贷款帐户的当前总余额占信用限额的比例，采取简单的均值填补的方式。

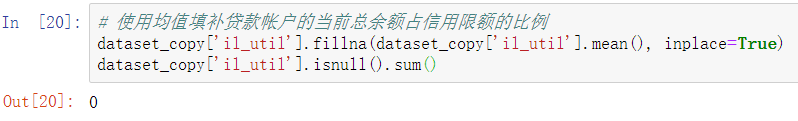
首先查看该列的均值为多少。



接下来就使用fillna()函数来对其进行填补。

*# 使用均值填补贷款帐户的当前总余额占信用限额的比例*dataset\_copy[**'il\_util'**].fillna(dataset\_copy[**'il\_util'**].mean(), inplace=True)

*# 查看本列的缺失值数量*  
dataset\_copy[**'il\_util'**].isnull().sum()



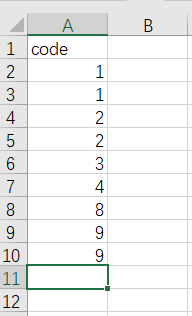
可以看到此时该列的缺失值的数量为0，填补成功。

### 3.3.4众数填补

mths\_since\_recent\_inq列是距离上一次调查的时间（单位是月），对其使用众数填补。

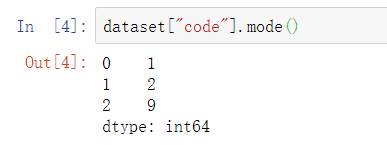
众数顾名思义，是数据集在该属性取值中，出现次数最多的值。众所周知的是，数据集中可能存在多个众数，此时我们便会面临一个问题：使用哪个众数进行填补呢？

下面便请同学们自行创建1个众数个数不为1的csv文件，看一下pandas取众数返回的结果是什么。



*# 打开数据文件*dataset = pd.read\_csv(**"D:/input/moreMode.csv"**, encoding=**"gbk"**, low\_memory=False)  
*# 查看数据前10行*dataset.head(10)  
*# 获取众数*dataset[**"code"**].mode()

查看返回的结果，如下图

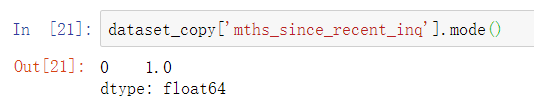


可见mode()函数返回的并不是一个数值，而是一个Series数组对象，不能直接使用它进行填充，而应指定使用哪个数值。

只需在进行填充时，在mode()后加上[0],指定使用第一个众数填补即可。

首先查看该列的众数为多少。

dataset\_copy[**'mths\_since\_recent\_inq'**].mode()

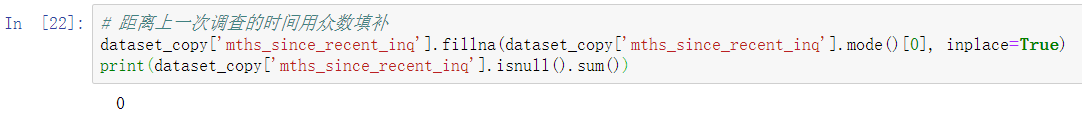


可见多数人距上一次贷款调查的时间为一个月左右，此处就用这个值进行填补。注意，此处虽然众数只有一个，但它仍然是一个Series对象，不能直接使用它进行填补。

使用fillna()函数进行填补，代码如下：

*# 距离上一次调查的时间用众数填补*dataset\_copy[**'mths\_since\_recent\_inq'**].fillna(dataset\_copy[**'mths\_since\_recent\_inq'**].mode()[0], inplace=True)

*# 查看本列的缺失值数量*  
print(dataset\_copy[**'mths\_since\_recent\_inq'**].isnull().sum())



缺失值的数量为0，该列填补成功。此时对这四列的简单填补成功，下面介绍一种更加简单的方式来对这四列进行填补。

### 3.3.5使用字典进行快速填补

首先拷贝一份原数据副本，避免再次读取原数据，造成不必要的内存占用。

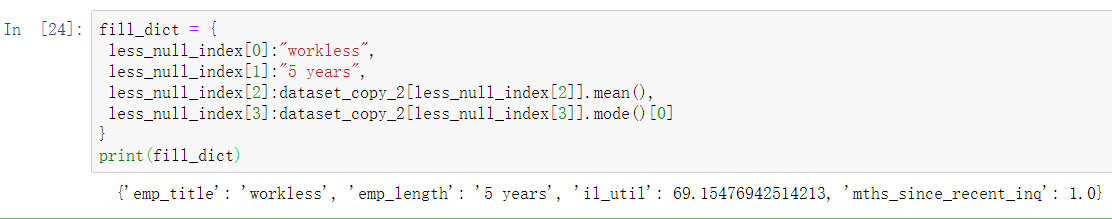
dataset\_copy\_2 = dataset.copy()

在前面介绍fillna()函数时，其value参数可以用一个字典进行传入，这样对其四列需要填补的属性来进行一个字典的创建，就可以只需要一个fillna()函数来进行填补了。

首先创建一个字典用于存储填补缺失值所需要传入的字典。

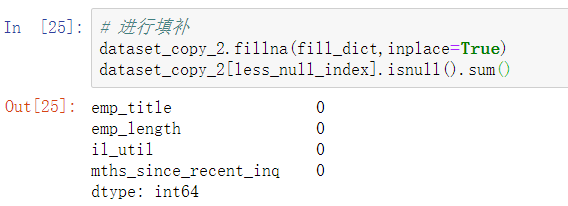
因之前已经计算完毕了填补各列所需的值，此处就直接使用计算得到的值即可。

fill\_dict = {  
 less\_null\_index[0]:**"workless"**,  
 less\_null\_index[1]:**"5 years"**,  
 less\_null\_index[2]:dataset\_copy\_2[less\_null\_index[2]].mean(),  
 less\_null\_index[3]:dataset\_copy\_2[less\_null\_index[3]].mode()[0]  
}  
print(fill\_dict)



*# 进行填补*dataset\_copy\_2.fillna(fill\_dict,inplace=True)

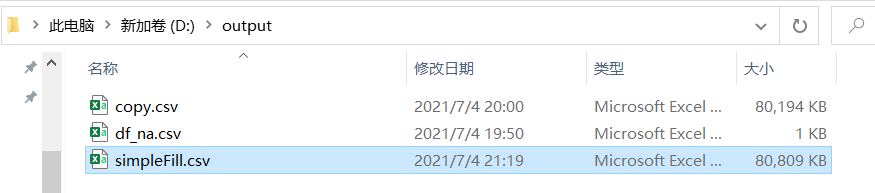
*# 查看四列的缺失值数量*  
dataset\_copy\_2[less\_null\_index].isnull().sum()



可以发现各列的缺失值数量都为0，填补成功。

到了这里，简单填补的操作就完成了，大家可以将此时的中间数据保存为一个副本，查看其变化。

*# 将中间结果保存为csv文件，存到output文件夹*dataset\_copy.to\_csv(**"D:/output/simpleFill.csv"**,index = False)



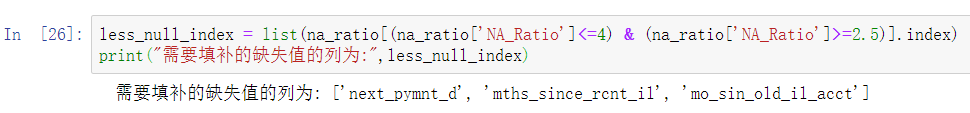
## 3.4向前向后与插值法进行缺失值的填补

进行前向与后向填补时，也是使用上文介绍的fillna()函数，对该函数中的method参数进行设置，设置为bfill即为后值向前填补，设置为pad即为前值向后填补。

接下来对剩下缺失百分比在3%左右的列进行前后方式以及插值法填补，首先查看缺失百分比在2.5%到4%的列有哪些。

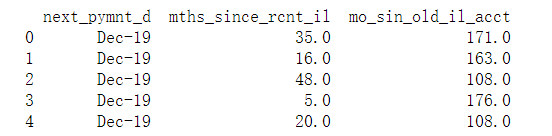
### 3.4.1属性查看

*# 首先查看缺失百分比在2.5%到4%的列有哪些*less\_null\_index = list(na\_ratio[(na\_ratio[**'NA\_Ratio'**] <= 4) & (na\_ratio[**'NA\_Ratio'**] >= 2.5)].index)  
print(**"需要填补的缺失值的列为:"**, less\_null\_index)

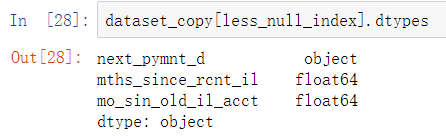


查看此三列的信息。

*# 查看此三列前五行的信息*print(dataset\_copy[less\_null\_index].head(5))



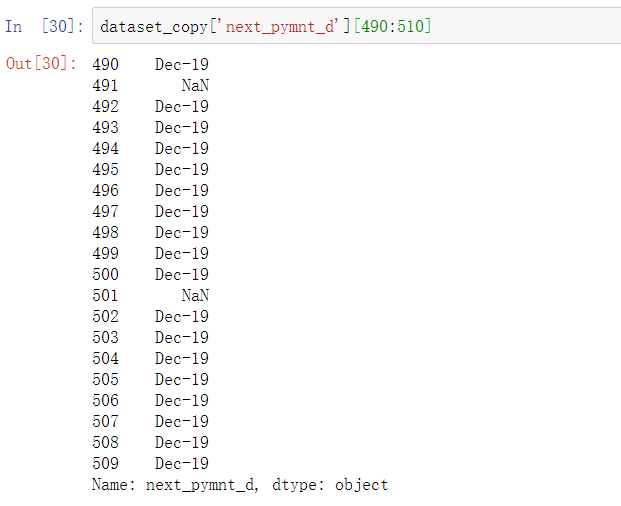
print(dataset\_copy[less\_null\_index].dtypes)



### 3.4.2后值向前填补

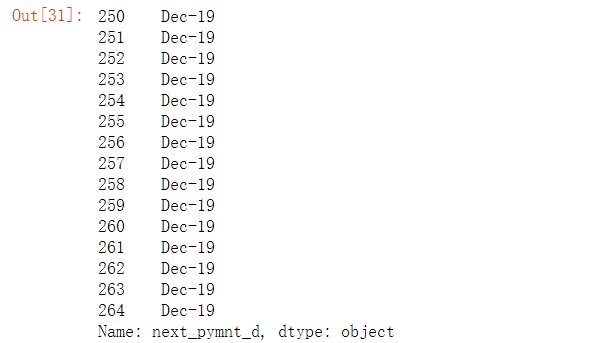
next\_pymnt\_d列是指下一个计划还款日期，对所有缺失值均使用缺失值所在的后一个非缺失值的值来进行填补。

首先简单查看缺失值在该列中的位置，方便之后进行对比，此处选取索引为490到510的几行来作为对比



可以看到此处有两个缺失值，接下来进行填补，填补完毕之后再查看其效果。

*# 进行填补，查看效果*dataset\_copy[**'next\_pymnt\_d'**] = dataset\_copy[**'next\_pymnt\_d'**].fillna(method=**"bfill"**)  
print(dataset\_copy[**'next\_pymnt\_d'**][250:265])



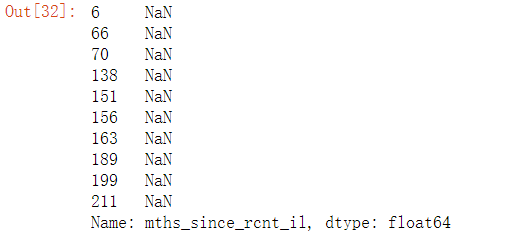
可见填补完毕后，缺失值消失，且均填补为了后值。

### 3.4.3前值向后填补

此处使用前值向后填补来填补mths\_since\_rcnt\_il列。此列的含义是最近分期付款账户开立后的月份时间。

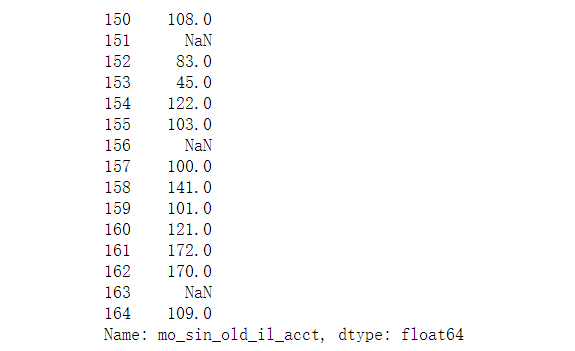
查看前十个缺失值的位置

dataset\_copy[**'mths\_since\_rcnt\_il'**][dataset\_copy[**'mths\_since\_rcnt\_il'**].isnull()].head(10)

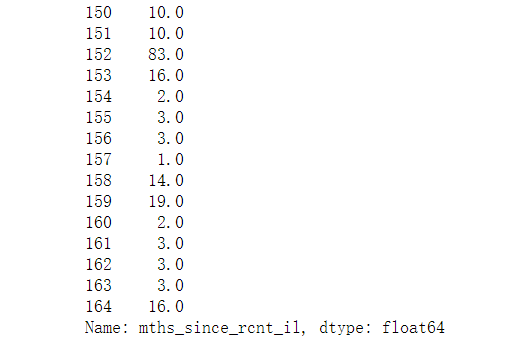


选取中间部分150-165来方便之后进行对比查看。

*# 原数据*print(dataset\_copy[**'mo\_sin\_old\_il\_acct'**][150:165])



*# 进行填补，对比*dataset\_copy[**'mths\_since\_rcnt\_il'**] = dataset\_copy[**'mths\_since\_rcnt\_il'**].fillna(method=**"pad"**)  
print(dataset\_copy[**'mths\_since\_rcnt\_il'**][150:165])



填补完毕后，该列不存在缺失值了。

### 3.4.4插值填补

简单的删除、填充、替换缺失数据会导致整体数据方差的变化，从而导致数据信息量的变换。使用插值法可解决此问题，即使用interpolate()函数填补缺失值。

该函数的主要参数是method，常见的插入方法包括：linear, time, index, values，spline等，参数不赋值时默认为线性插入法linear，即用该列数据缺失值前一个数据和后一个数据建立插值直线，然后使用缺失点在线性插值函数的函数值填充该缺失值。对于method参数设置为polynomial和spline等，需要设置对应的order参数，代表所使用方法的逼近次数。

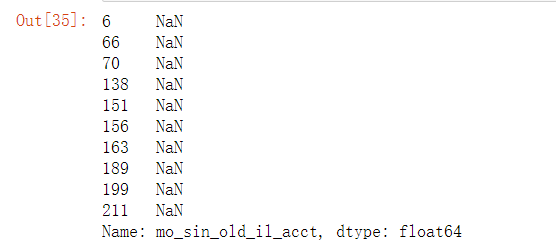
该函数的默认值填补是使用了在一个或多个缺失值的前后非空值部分，将其等分填入，即简单的拉格朗日插值法。

#### 3.4.4.1线性插值

mo\_sin\_old\_il\_acct列代表客户首次开立分期付款账户至今的累计月份数，对其使用线性插值填补，即简单的拉格朗日插值法。

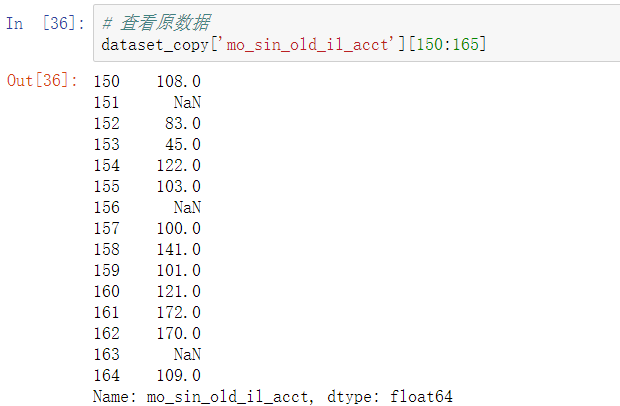
查看前十个缺失值的分布。

dataset\_copy[**'mo\_sin\_old\_il\_acct'**][dataset\_copy[**'mo\_sin\_old\_il\_acct'**].isnull()].head(10)



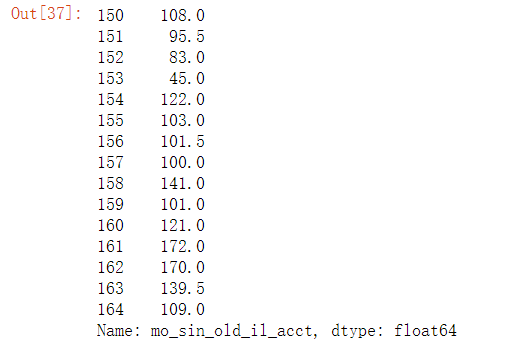
此处选取索引为150到165的行来作为对比查看。

*# 原数据*dataset\_copy[**'mo\_sin\_old\_il\_acct'**][150:165]



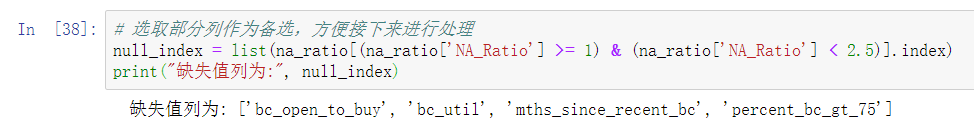
*# 进行线性插值，对比数据*dataset\_copy[**'mo\_sin\_old\_il\_acct'**] = dataset\_copy[**'mo\_sin\_old\_il\_acct'**].interpolate()

*# 打印150-165行填充后此列的值*  
print(dataset\_copy[**'mo\_sin\_old\_il\_acct'**][150:165])



对此列的插值已经完成，接下来选取缺失值占1%~2.5%的属性使用其他其他插值方法进行填补。

*# 选取部分列作为备选，方便接下来进行处理*null\_index = list(na\_ratio[(na\_ratio[**'NA\_Ratio'**] >= 1) & (na\_ratio[**'NA\_Ratio'**] < 2.5)].index)  
print(**"缺失值列为:"**, null\_index)



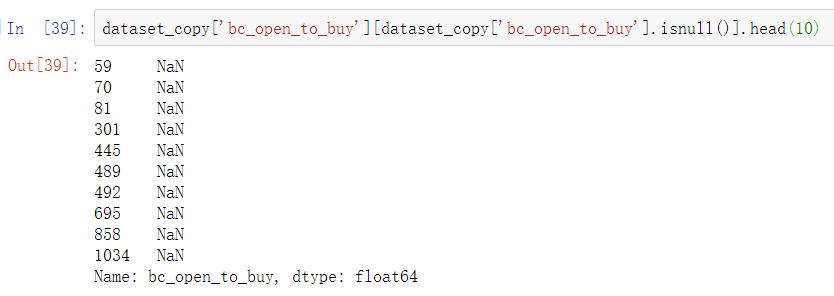
可见缺失值列为: ['bc\_open\_to\_buy', 'bc\_util', 'mths\_since\_recent\_bc', 'percent\_bc\_gt\_75']

#### 3.4.4.2多项式插值

在interpolate函数中，将method参数设置为polynomial代表多项式插值，进一步将order参数设置为2，代表使用二次多项式插值。

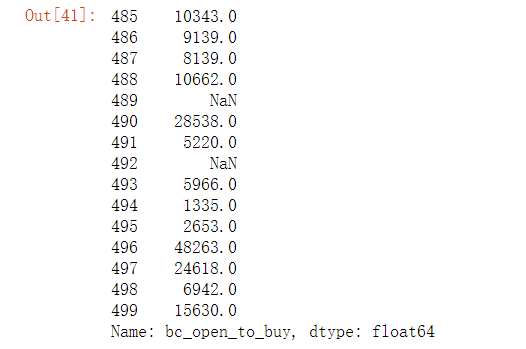
bc\_open\_to\_buy列代表着在循环银行卡上的购买金额，对此列采用二次多项式插值。

查看此列前十个缺失值的分布情况



选取485-500行作为对象，观察插值效果。

*# 查看原数据*dataset\_copy[**'bc\_open\_to\_buy'**][485:500]



*# 进行插值，查看效果*dataset\_copy[**'bc\_open\_to\_buy'**].interpolate(method = **"polynomial"**, order = 2, inplace=True)  
dataset\_copy[**'bc\_open\_to\_buy'**][485:500]



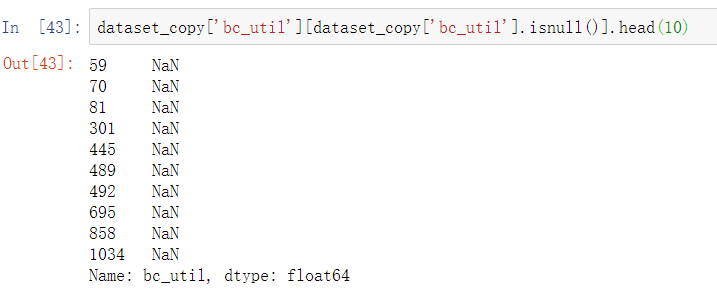
可见插值完成。

#### 3.4.4.3样条插值

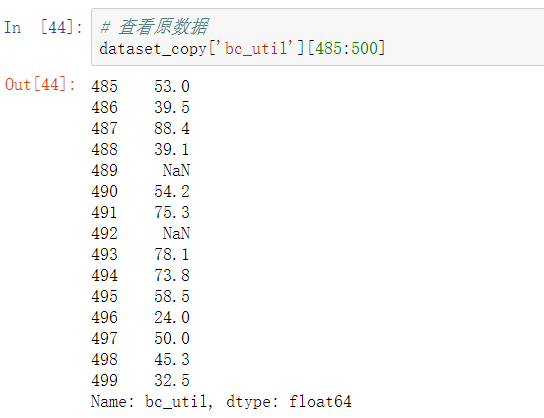
bc\_util列代表所有银行卡帐户的当前总余额与信用限额的比例，对其使用三次样条插值填补。

使用interpolate来进行三次样条插值的方式，将method参数设置为spline，将order参数设置为3。

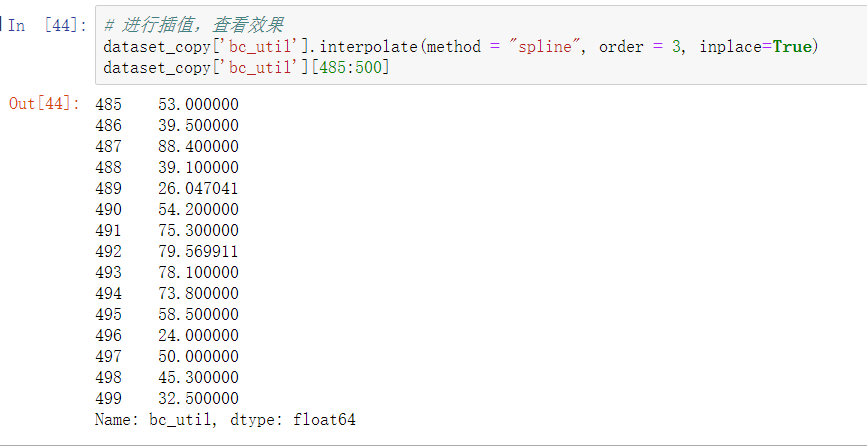
查看前十个空缺值的分布。



选择485-500行数据进行对比，观察插值效果。



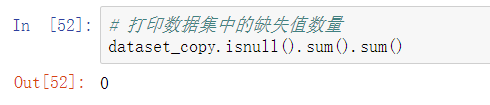
*# 进行插值，查看效果*dataset\_copy[**'bc\_util'**].interpolate(method = **"spline"**, order = 3, inplace=True)  
dataset\_copy[**'bc\_util'**][485:500]



对剩下的所有列的缺失值进行简单的向前填补处理。

*# 剩下的所有列的缺失值进行简单的向前填补处理*dataset\_copy.fillna(method=**"bfill"**, inplace=True)

查看此时的缺失值个数



可见，经过一系列的处理后数据中已经没有缺失值。

在缺失值处理中还有使用模型填补的处理方式，其代表是使用KNN算法对缺失值进行填补。该方法一般适用于在数据集中只有一列或者两列有缺失值的时候，因为较多缺失值进行了填补之后会导致数据本身就存在一定误差，与事实有较大偏离，那么在运算模型的时候，这些填补过的数据列就会产生较大影响，从而使得最后的填补建模以及预测或者回归建模都会存在较大的误差，而只有较少数据列有缺失的时候，此时使用建模方法进行填充就等于使用别的所有的无缺失值的列来预测该存在缺失值的列，从而就转化为了一个建模与预测的问题。除了KNN算法，还有随机森林、GBDT等集成方法可以使用。使用模型进行填补的方法需要用到扩展包Fancyimpute或者Sklearn，其具体做法也与之后的数据建模类似。

# 4异常值处理

首先，我们要知道什么是异常值，异常值是指样本中的个别值，其数值明显偏离它（或他们）所属样本的其余观测值。一般情况下，我们将偏离第一个或第三个四分位数严重的点，称为孤立点即异常点。

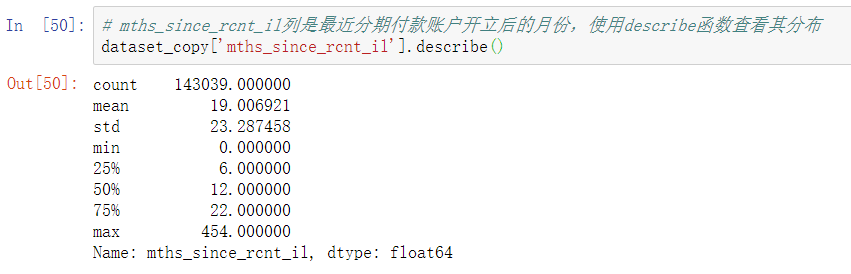
四分位数（Quartile）也称四分位点，是指在统计学中把所有数值由小到大排列并分成四等份，处于三个分割点位置的数值。四分位数是通过3个点将全部数据等分为4部分，其中每部分包含25%的数据。很显然，中间的四分位数就是中位数，因此通常所说的四分位数是指处在25%位置上的数值（称为下四分位数）和处在75%位置上的数值（称为上四分位数）。第三四分位数与第一四分位数的差距成为四分位极差，又称四分位距（InterQuartile Range,IQR）。

通常我们认为落在至少高于第三个四分位数或低于第一个四分位数 1.5倍四分位极差处的值，为异常值。

在数据预处理时很容易忽略异常值的处理，因为其发现难度较大。但是异常值的存在是会对模型产生非常大的影响的，所以找到异常值并处理异常值很重要。

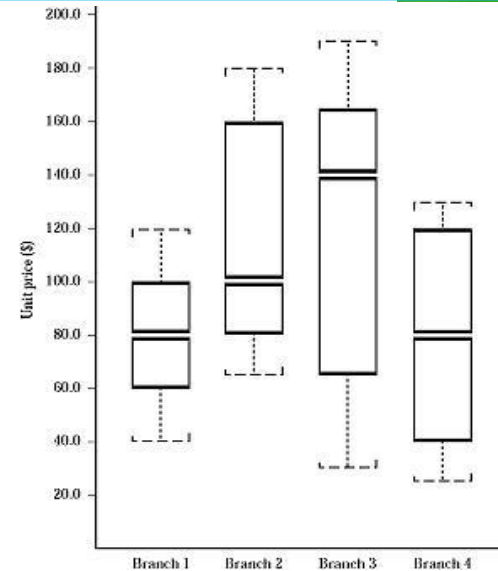
处理异常值的过程中，较难的是如何找到，一般来说会绘制箱线图或者该列的折线图来进行异常值的查看，找到异常值后可以有各种方法来对其进行处理，例如直接删除该数据，或者进行各类填补，此处填补方式与缺失值类似就不多介绍，主要介绍如何找到缺失值。

mths\_since\_rcnt\_il列是最近分期付款账户开立后的月份，使用describe函数查看其分布。



可以发现最大值异常的大，有异常值的趋势，但是是否真的符合以及是否只有最大值是异常值还有待进一步挖掘。故而此处绘制盒图来查看数据的分布。

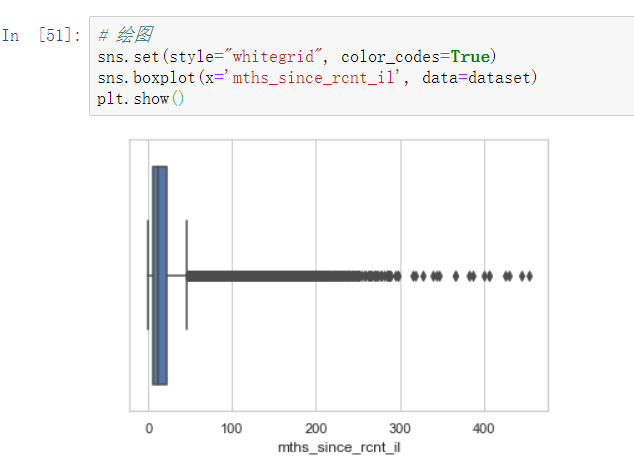
盒图是在1977年由美国的统计学家约翰·图基(John Tukey)发明的。它由五个数值点组成：最小值(min)，下四分位数(Q1)，中位数(median)，上四分位数(Q3)，最大值(max)。下四分位数、中位数、上四分位数组成一个“带有隔间的盒子”。上四分位数到最大值之间建立一条延伸线，这个延伸线成为“胡须(whisker)”。盒图示例如下图：



*# 设置背景*sns.set(style=**"whitegrid"**, color\_codes=True)

*# 设置横轴为数值*  
sns.boxplot(x=**'mths\_since\_rcnt\_il'**, data=dataset)

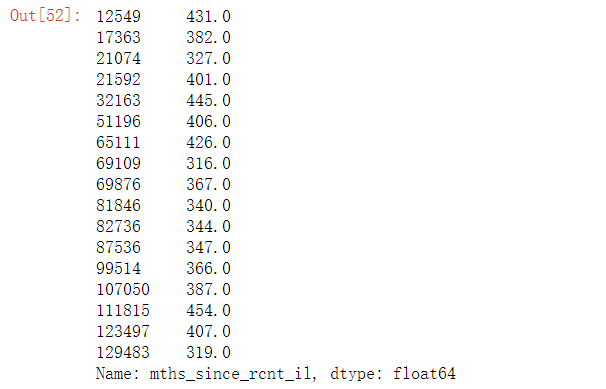
*# 显示图像*  
plt.show()



因为此列的含义是最近分期付款账户开立后的月份，有其特殊性，不适合采用一般的异常值判定规则。由盒图可以发现，值在300以上较为分散，且相对别的数据这几个数据较为离群，故而简单的认为对于该列，大于300的均为离群值，即异常值。

查看这些异常值。

dataset\_copy[**'mths\_since\_rcnt\_il'**][dataset\_copy[**'mths\_since\_rcnt\_il'**] > 300]



使用drop()函数来删除这些异常值所存在的行。

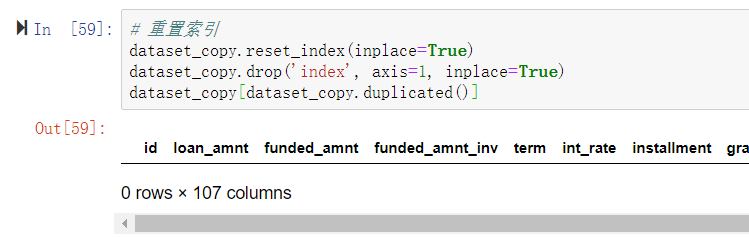
*# 使用drop()函数来删除这些异常值所存在的行*print(**"删除异常值之前的行数"**, len(dataset\_copy[**'mths\_since\_rcnt\_il'**]))  
dataset\_copy.drop(list(dataset\_copy[**'mths\_since\_rcnt\_il'**][dataset\_copy[**'mths\_since\_rcnt\_il'**] > 300].index), axis=0,inplace=True)  
print(**"删除异常值之后的行数"**, len(dataset\_copy[**'mths\_since\_rcnt\_il'**]))



异常值处理前行数为143039，处理后为143022。

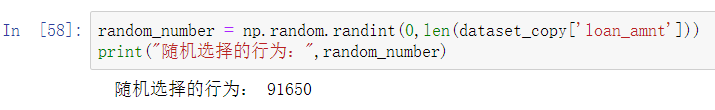
# 5重复值处理

一般检测重复数据使用duplicated函数。

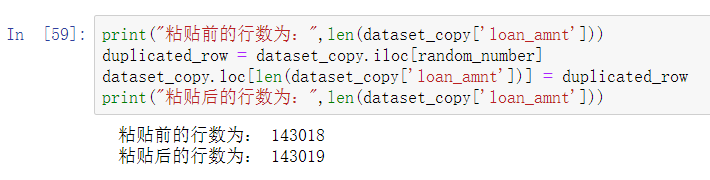


可见数据中不存在重复值。为了演示重复值检测的方法，此处从数据中随机选取一个行并将其添加到数据中。

*# 随机选择一行*random\_number = np.random.randint(0,len(dataset\_copy[**'loan\_amnt'**]))  
print(**"随机选择的行为："**,random\_number)

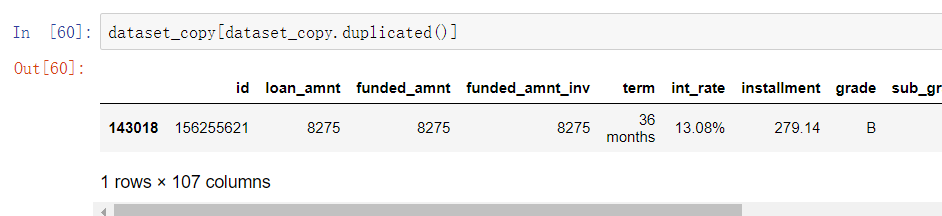


print(**"粘贴前的行数为："**,len(dataset\_copy[**'loan\_amnt'**]))  
duplicated\_row = dataset\_copy.iloc[random\_number]  
dataset\_copy.loc[len(dataset\_copy[**'loan\_amnt'**])] = duplicated\_row  
print(**"粘贴后的行数为："**,len(dataset\_copy[**'loan\_amnt'**]))



此时再使用duplicated()函数来查看重复值。

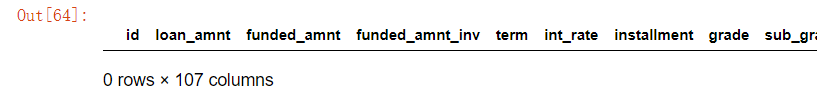
dataset\_copy[dataset\_copy.duplicated()]



此时可以看到复制的一行重复值已经被找到。

接下来就是删除重复值，一般使用drop\_duplicated()来删除，其参数keep设置为first时，代表删除重复值时保留第一次出现的数据，设置为last时代表删除重复值时保留最后出现的数据，设置为False时代表去除所有重复的数据,inplace代表是否替换原DataFrame。

dataset\_copy.drop\_duplicates(keep=**'first'**,inplace=True)  
dataset\_copy[dataset\_copy.duplicated()]



删除完毕后再次调用duplicated()函数来查看重复值，可以发现重复值已经被删除。

此处请同学们使用之前的数据保存方法，保存数据副本到磁盘。

# 6数据标准化

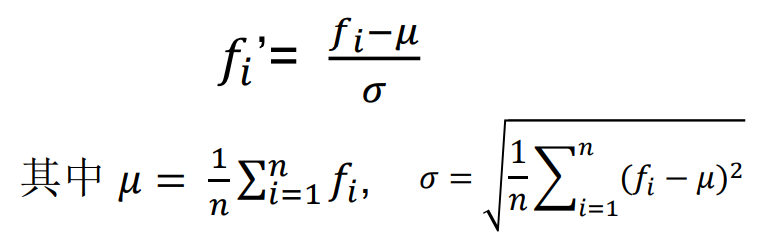
数据标准化（归一化）处理是数据挖掘的一项基础工作。不同评价指标往往具有不同的量纲，数值见的差别可能很大，不进行处理可能会影响到数据分析的结果；同时有很多机器学习算法需要其输入特征为标准化的形式。为了消除指标之间的量纲和取值范围差异的影响，需要进行标准化处理，将数据按照比例进行缩放，使之落入一个特定的区域，便于进行综合分析，这就是数据标准化。

常用的数据标准化方法有Z-score标准化、Min-Max标准化、小数定标标准化和Logistic标准化。本节将使用sklearn中的包，就最常用的Z-score标准化和Min-Max标准化进行演示。

## 6.1 Z-score标准化

Z-score标准化也称为标准差标准化，是最常用的标准化方法，该方法通过对数据集的每一个样本进行处理，使得处理后的数据具有固定的均值和标准差。

设原数据集中的元素为f，标准化后的取值为f’则

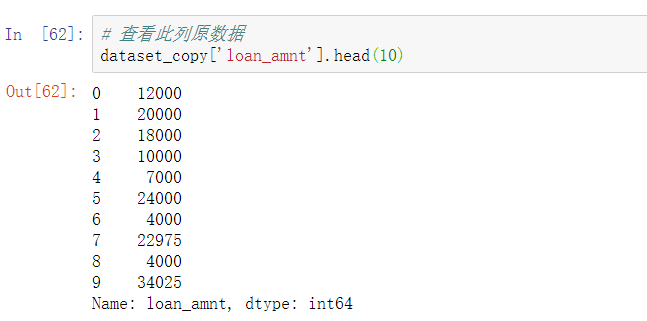


经过Z-Score标准化后的数据，能够直观反应每个数据点距离平均值点的标准差距离，从而理解整体数据的分布情况。均值将落在0附近，而每一个数据点离零点的距离可解释为其远离均值的标准差距离。

Z-Score的标准化方法适用于数据系列中最大值或最小值未知、数据分布非常离散的情况。

loan\_amnt属性是借款人申请贷款的金额，我们对其进行Z-score标准化。

*# 查看此列原数据*dataset\_copy[**'loan\_amnt'**].head(10)



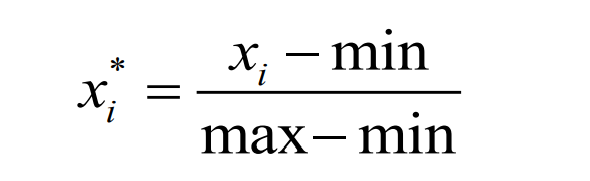
*# z-score标准化，适配数据集获取参数*std\_scale = preprocessing.StandardScaler().fit(dataset\_copy[[**'loan\_amnt'**]])  
*# 进行标准化，得到标准化的数据*np\_std = std\_scale.transform(dataset\_copy[[**'loan\_amnt'**]])  
*# 将标准化后的数据加到dataset\_copy中，替换初始数据*dataset\_copy[[**'loan\_amnt'**]] = np\_std  
*# 查看此列标准化后的数据*dataset\_copy[**'loan\_amnt'**].head(10)



可见数据已完成标准化。

## 6.2 Min-Max标准化（0-1标准化）

Min-Max标准化也称为离散标准化，该方法的核心即是对数据系列作线性变换，使得处理过后数据均落在［0，1］区间内。其公式为

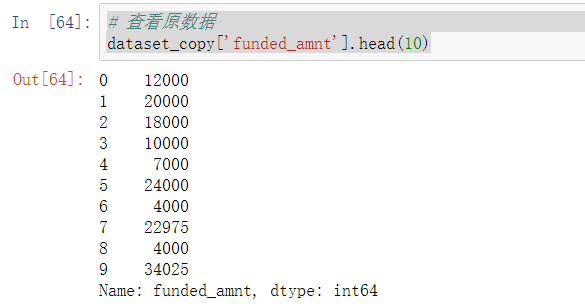


0-1标准化适用于需要将数据简单地变换映射到某一区间中，保留了原来数据中存在的关系，是消除量纲和数据取值范围影响的最简单方法；但其不足之处在于当有新数据加入时，可能会导致数据系列中的最大值或最小值发生变化，此时便需要重新定义最大值、最小值。

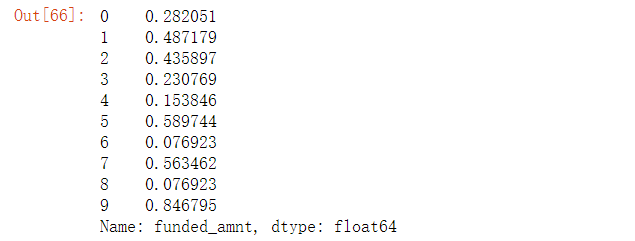
funded\_amnt是在某时间点承诺的贷款总额，我们对此属性采用Min-Max标准化，观察其标准化前后的变化。

首先查看此列原数据。

*# 查看原数据*dataset\_copy[**'funded\_amnt'**].head(10)



*# 0-1标准化，适配数据集，确定参数*minmax\_scale = preprocessing.MinMaxScaler().fit(dataset\_copy[[**'funded\_amnt'**]])  
*# 进行标准化，得到标准化的数据*np\_minmax = minmax\_scale.transform(dataset\_copy[[**'funded\_amnt'**]])  
*# 将标准化后的数据加到dataset\_copy中，替换初始数据*dataset\_copy[[**'funded\_amnt'**]] = np\_minmax  
*# 查看此列标准化后的数据*dataset\_copy[**'funded\_amnt'**].head(10)



可以看到，原数据被映射到[0,1]区间，此列数据的标准化完成。

此处各种标准化的实验步骤并无太大差异，请同学们自行尝试使用小数定标标准化和Logistic标准化对数值数据列进行处理，查看数据的变化。

# 7数据编码

很多机器学习算法不能处理或不善于处理字符串等复杂数据，为了保证算法的可用性和性能、准确度，同时使数据便于记忆，我们时常需要对数据进行编码，将复杂的记录转化为简单的符号或数字。

较为常用的数据编码方式主要有数字编码、one-hot编码和哑变量编码，由于哑变量编码只是在one-hot编码基础上稍作调整，接下来我们只对数字编码和one-hot编码两种编码方式进行尝试，哑变量编码的实操，请同学们自行完成。

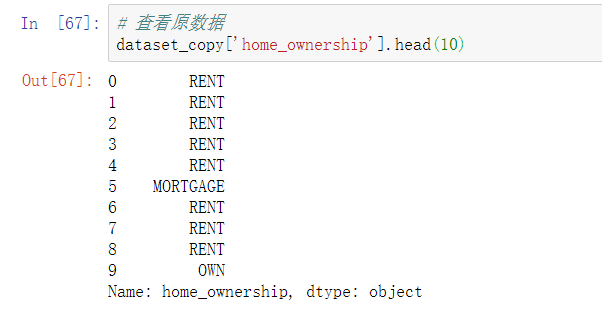
## 7.1数字编码

数字编码是最简单的编码方式，使用单个数字特征来表示非数字特征。用国籍信息举例，用1来表示国籍为美国，2表示英国，3表示法国，用对应的数字替换原数据即完成了数字编码。

这种编码方式胜在实现简单，意义明确，缺点是会引入错误的序和距离，可能会对结果产生不利影响。

home\_ownership是借款人在登记期间提供的或从信贷报告中获得的房屋所有权状况，我们对其进行数字编码。这个属性有四个取值分别为租金、自有、抵押、其他，我们依次用0，1，2，3表示这四种情况。

*# 查看原数据*dataset\_copy[**'home\_ownership'**].head(10)



*# 数字编码*col\_dicts = {**'home\_ownership'**: {**'MORTGAGE'**: 0,  
 **'OWN'**: 1,  
 **'RENT'**: 2,  
 **'others'** : 3}}  
*# 进行转换*dataset\_copy[**'home\_ownership'**] = dataset\_copy[**'home\_ownership'**].map(col\_dicts[**'home\_ownership'**])  
*# 查看转化后的数据*dataset\_copy[**'home\_ownership'**].head(10)



编码完成，已成功将字符串数据转换为数字数据。此处请同学们保存数据副本到磁盘。

## 7.2 one-hot编码

独热编码，又称一位有效编码，其方法是使用N位状态寄存器来对N个状态进行编码，每个状态都有他独立的寄存器位，并且在任意时候，其中只有一位有效（为1），其余全为0。

可以这样理解，对于每一个特征，如果它有m个可能值，那么经过独热编码后，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，因此，数据会变成稀疏的。

同样以国籍信息举例，使用独热编码后，就会把三个空间的位置予国籍特征。三个国籍的编码状态就为：美国（1,0,0）、英国（0,1,0）、法国（0,0,1）。

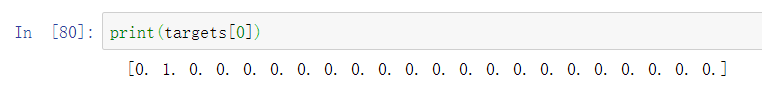
相比数字编码，独热编码有很多优势，包括不会给名义型特征的取值人为地引入次序关系；经过One-Hot编码之后，不同的原始特征取值之间拥有相同的距离；对包含离散型特征的分类模型的效果有很好的提升等。但同时，它也有它的缺点：特征维度会显著增多；增加特征之间的相关性。

delinq\_2yrs是申贷日期前2年逾期次数，我们对其进行one-hot编码。

首先查看原数据



查看第一条数据的编码结果



可见数据被编码为22维的向量。

## 7.3哑变量编码

对于一个包含K个取值的离散型特征，将其转换成K-1个二元特征，这种编码方法称为哑变量编码 (dummy encoding)。

哑变量编码可以看作对one-hot编码的改进，例如：特征“汽车品牌” = {路虎，吉利，奥迪，大众，奔驰}，一共包含5个不同的取值. 我们可以将其编码为4个二元特征. 当原始特征取不同取值时，转换后的特征取值如下：



其与one-hot编码实现基本相同，此处请同学们自行完成相关操作，查看操作效果。

# 8数据保存

在做完数据清洗后会对新的“好数据”进行存储，以方便之后的建模等工作时可以直接调用。

## 8.1使用python原生方法

Python自带的函数写入文件较为简单，首先需要将文件作为对象读取，也就是使用open()函数将文件载入到内存中并创建一个对应的对象，其中第一个字符串代表着文件的路径，第二个w代表是对该文件进行写处理，若为r则代表对该文件进行读处理。

fw = open(**"D:/output/finish.csv"**,**"w"**)

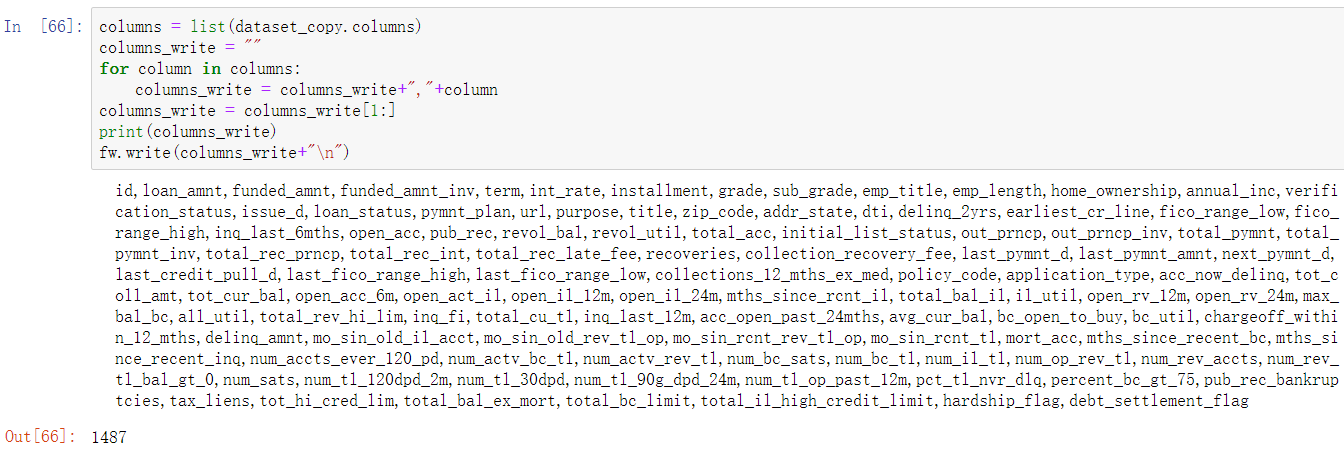
在创建完毕写入文件对象后，可以使用write()或者writelines()函数将内容传入，二者的区别在于write()函数只能传入字符串对象，而writelines()函数可以传入列表作为写入的内容，需要在列表中添加换行符进行换行。

此处使用write()作为写入的函数使用，此处输出查看传入的字符串参数来检查写入到文件的内容。

首先读取DataFrame的列名，并将其写入到文件的第一行，因为写入文件函数write()的参数需要是一个字符串，所以首先对读取到的列名进行简单的字符串粘贴，且在最后加入转义字符\n进行换行，方便接下来的内容的写入。

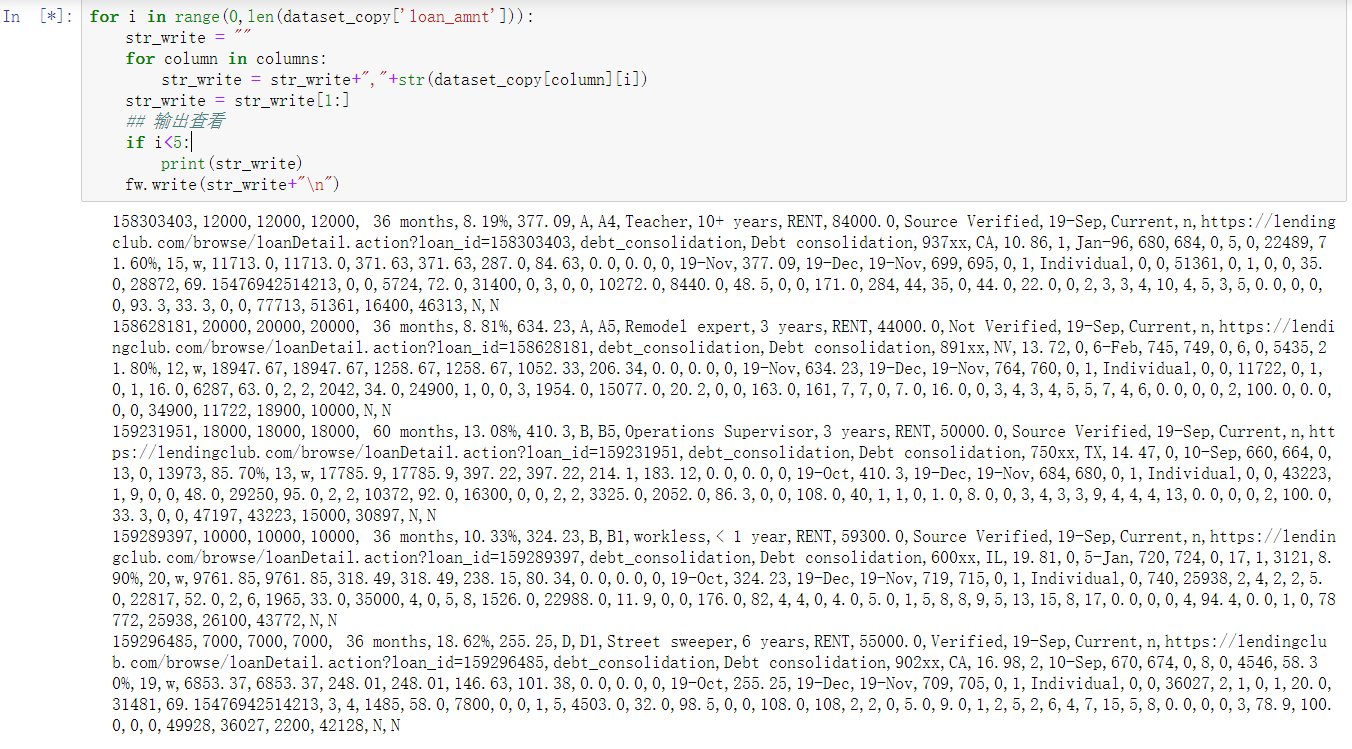
因为此处输出的文件是csv格式文件，该文件格式是逗号分隔符，故而需要在每一个不同的列或者内容后面添加逗号用以区分，不同格式文件有不同存储规则，例如txt一般是空格分隔符文件，则需要在不同内容后面加入空格予以区分。

columns = list(dataset\_copy.columns)  
columns\_write = **""**for column in columns:  
 columns\_write = columns\_write+**","**+column  
columns\_write = columns\_write[1:]  
print(columns\_write)  
fw.write(columns\_write+**"**\n**"**)



写入文件内容，将每一行的内容全部粘贴为一个字符串，再使用write()函数进行写入。

for i in range(0,len(dataset\_copy[**'loan\_amnt'**])):  
 str\_write = **""** for column in columns:  
 str\_write = str\_write+**","**+str(dataset\_copy[column][i])  
 str\_write = str\_write[1:]  
 *## 输出查看* if i<5:  
 print(str\_write)  
 fw.write(str\_write+**"**\n**"**)



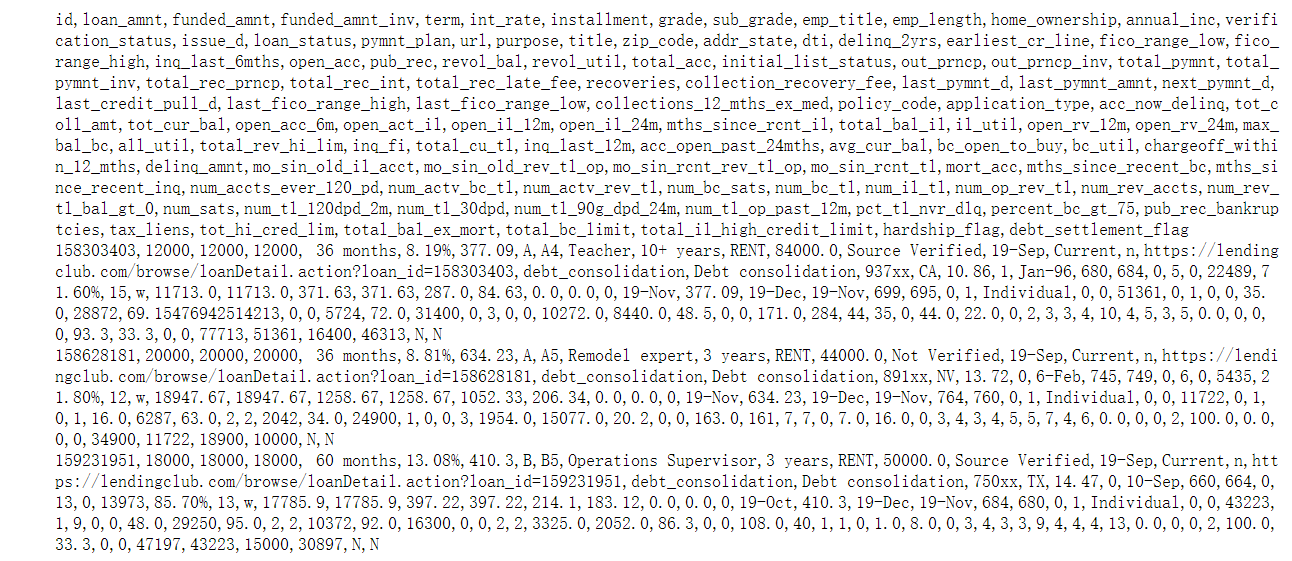
无论是读取文件还是写入文件，一旦创建了文件对象则需要对其进行关闭，释放内存。在Python中直接使用close()函数即可。

fw.close()

在Python中提供with方法来简易创建并关闭文件对象的操作。该方法可以不对文件进行一次手动的读取与关闭，该方法会自动关闭文件对象，使得代码更加简单直观。

with open(**"D:/output/finish.csv"**,**"w"**) as fw:

columns\_write = **""** *## 写入列名* for column in columns:  
 columns\_write = columns\_write+**","**+column  
 columns\_write = columns\_write[1:]  
 print(columns\_write)  
 fw.write(columns\_write+**"**\n**"**)  
 *## 写入内容* for i in range(0,len(dataset\_copy[**'loan\_amnt'**])):  
 str\_write = **""** for column in columns:  
 str\_write = str\_write+**","**+str(dataset\_copy[column][i])  
 str\_write = str\_write[1:]  
 *## 输出查看* if i<3:  
 print(str\_write)  
 fw.write(str\_write+**"**\n**"**)



## 8.2使用pandas中的函数

在Pandas中，可以直接对格式为DataFrame的数据进行文件的存储。使用Pandas中的to\_csv()函数可以进行csv文件的输出,因为不需要写入索引信息，所以此处对index参数设置为False。

dataset\_copy.to\_csv(**"D:/output/finish2.csv"**,index = False)

Pandas同样支持很多其他格式文件的输出，例如输出txt文件可以将to\_csv()函数的sep参数设置为"\s"分隔符。

若想输出excel文件则可以使用to\_excel()函数，json文件可使用to\_json()函数等。