```
import pandas as pd
import numpy as np
from datetime import datetime
```

在一些数据分析业务中，数据缺失是我们经常遇见的问题，缺失值会导致数据质量的下降，从而影响模型预测的准确性，
这对于机器学习和数据挖掘影响尤为严重。因此妥善的处理缺失值能够使模型预测更为准确和有效。

## 为什么会存在缺失值？

在前面的学习过程中,我们遇到过很多 NaN 值，关于缺失值您可能会有很多疑问，数据为什么会丢失数据呢，又是从什么时候丢失的呢？通过下面场景，您会得到答案。

其实在很多时候，人们往往不愿意过多透露自己的信息。假如您正在对用户的产品体验做调查，在这个过程中您会发现，一些用户很乐意分享自己使用产品的体验，但他是不愿意透露自己的姓名和联系方式；

还有一些用户愿意分享他们使用产品的全部经过，包括自己的姓名和联系方式。

因此，总有一些数据会因为某些不可抗力的因素丢失，这种情况在现实生活中会经常遇到。

## 什么是稀疏数据？

稀疏数据，指的是在数据库或者数据集中存在大量缺失数据或者空值，我们把这样的数据集称为稀疏数据集。稀疏数据不是无效数据，只不过是信息不全而已，只要通过适当的方法就可以“变废为宝”。

稀疏数据的来源与产生原因有很多种，大致归为以下几种：

```
由于调查不当产生的稀疏数据;
由于天然限制产生的稀疏数据;
文本挖掘中产生的稀疏数据。
```

# 一、缺失值类型

在pandas中，缺失数据显示为NaN。缺失值有3种表示方法，`np.nan`，`None`，`pd.NA`

## 1、np.nan

缺失值有个特点，它不等于任何值，连自己都不相等。如果用nan和任何其它值比较都会返回nan

```
np.nan == np.nan
```

```
也正由于这个特点，在数据集读入以后，不论列是什么类型的数据，默认的缺失值全为np.nan。

因为nan在Numpy中的类型是浮点，因此整型列会转为浮点；而字符型由于无法转化为浮点型，只能归并为object类型（‘O’），原来是浮点型的则类型不变。
```

```
type(np.nan)
```

```
float
```

```
pd.Series([1,2,3]).dtype
```

```
dtype('int64')
```

```
pd.Series([1,np.nan,3]).dtype
```

```
dtype('float64')
```

初学者做数据处理遇见object类型会发懵，不知道这是个啥，明明是字符型，导入后就变了，其实是因为缺失值导致的。

除此之外，还要介绍一种针对时间序列的缺失值，它是单独存在的，用`NaT`表示，是pandas的内置类型，可以视为时间序列版的np.nan，也是与自己不相等

```
s_time = pd.Series([pd.Timestamp('20220101')]*3)
s_time
```

```
s_time[2] = np.nan
s_time
```

## 2、None

还有一种就是None，它要比nan好那么一点，因为它至少自己与自己相等

```
None == None
```

在传入数值类型后，会自动变为np.nan

```
pd.Series([1,None])
```

```
0    1.0
1    NaN
dtype: float64
```

None 不会自动出现在pandas中，所以None大家基本也看不到。

## 3、NA标量

pandas1.0以后的版本中引入了一个专门表示缺失值的标量pd.NA，它代表空整数、空布尔值、空字符

对于不同数据类型采取不同的缺失值表示会很乱。pd.NA就是为了统一而存在的。pd.NA的目标是提供一个缺失值指示器，可以在各种数据类型中一致使用(而不是np.nan、None或者NaT分情况使用)。

```
s_new = pd.Series([1, 2], dtype="Int64")
s_new
```

```
0    1
1    2
dtype: Int64
```

```
s_new[1] = pd.NA
s_new
```

```
0       1
1    <NA>
dtype: Int64
```

下面是pd.NA的一些常用算术运算和比较运算的示例：

```
##### 算术运算
# 加法
print("pd.NA + 1 :\t", pd.NA + 1)

# 乘法

print('"a" * pd.NA:\t', "a" * pd.NA)

# 以下两种其中结果为1

print("pd.NA ** 0 :\t", pd.NA ** 0)


print("1 ** pd.NA:\t", 1 ** pd.NA)
##### 比较运算

print("pd.NA == pd.NA:\t", pd.NA == pd.NA)

print("pd.NA < 2.5:\t", pd.NA < 2.5)



print("np.add(pd.NA, 1):\t", np.add(pd.NA, 1))

```

```
pd.NA + 1 :  <NA>
"a" * pd.NA:     <NA>
pd.NA ** 0 :     1
1 ** pd.NA:  1
pd.NA == pd.NA:  <NA>
pd.NA < 2.5:     <NA>
np.add(pd.NA, 1):    <NA>
```

# 二、缺失值判断

对于缺失值一般有2种处理方式，要么删除，要么填充(用某个值代替缺失值)。
缺失值一般分2种，

- 一种是某一列的数据缺失。
- 另一种是整行数据都缺失，即一个空行

本文所用到的Excel表格内容如下:

| | A | B | C | D | E |
|---|---|---|---|---|---|
| 1 | 区域 | 省份 | 城市 | 时间 | 指标 |
| 2 | 东北 | 辽宁 | 大连 | 2022年1月5日 | 12.00 |
| 3 | 西北 | | 西安 | 2021年12月11日 | 87.00 |
| 4 | 华南 | 广东 | 深圳 | | |
| 5 | 华北 | 北京 | 北京 | 2022年3月4日 | 45.00 |
| 6 | 华中 | 湖北 | 武汉 | 2020年6月1日 | 21.00 |
| 7 | 东北 | 黑龙江 | 哈尔滨 | | 35.00 |
| 8 | | | | | |
| 9 | 华南 | | 广州 | 2019年9月8日 | 34.00 |
| 10 | 华北 | 内蒙 | 呼和浩特 | 2022年4月24日 | 56.00 |
| 11 | 华中 | | 益阳 | | 14.00 |
| 12 | 东北 | 辽宁 | 鞍山 | 2021年11月2日 | |
| 13 | 西北 | 陕西 | 西安 | 2022年3月15日 | 32.00 |

```
df = pd.read_excel(r"data\data_test.xlsx")
df
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

| | 区域 | 省份 | 城市 | 时间 | 指标 |
|---|---|---|---|---|---|
| **0** | 东北 | 辽宁 | 大连 | 2022-01-05 | 12.0 |
| **1** | 西北 | NaN | 西安 | 2021-12-11 | 87.0 |
| **2** | 华南 | 广东 | 深圳 | NaT | NaN |
| **3** | 华北 | 北京 | 北京 | 2022-03-04 | 45.0 |
| **4** | 华中 | 湖北 | 武汉 | 2020-06-01 | 21.0 |
| **5** | 东北 | 黑龙江 | 哈尔滨 | NaT | 35.0 |
| **6** | NaN | NaN | NaN | NaT | NaN |
| **7** | 华南 | NaN | 广州 | 2019-09-08 | 34.0 |
| **8** | 华北 | 内蒙 | 呼和浩特 | 2022-04-24 | 56.0 |
| **9** | 华中 | NaN | 益阳 | NaT | 14.0 |
| **10** | 东北 | 辽宁 | 鞍山 | 2021-11-02 | NaN |
| **11** | 西北 | 陕西 | 西安 | 2022-03-15 | 32.0 |

从结果来看，每一列均有缺失值。这里特别注意，时间日期类型的数据缺失值用NaT表示，其他类型的都用NaN来表示。

千万不要笼统的认为缺失值都是用NaN来表示

## 1、查看缺失值的情形

```
1 df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 12 entries, 0 to 11
Data columns (total 5 columns):
 #   Column  Non-Null Count  Dtype         
---  ------  --------------  -----         
 0   区域      11 non-null     object        
 1   省份      8 non-null      object        
 2   城市      11 non-null     object        
 3   时间      8 non-null      datetime64[ns]
 4   指标      9 non-null      float64       
dtypes: datetime64[ns](1), float64(1), object(3)
memory usage: 608.0+ bytes
```

从结果来看，省份这一列是8 non-null。

说明省份这一列有4个null值。同理，时间这一列有4个缺失值，指标这一列有3个缺失值，城市这一列有1个缺失值，区域这一列有1个缺失值

## 2.缺失值的判断

isnull()：判断具体的某个值是否是缺失值，如果是则返回True,反之则为False

```
df.isnull()
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

| | 区域 | 省份 | 城市 | 时间 | 指标 |
|---|---|---|---|---|---|
| **0** | False | False | False | False | False |
| **1** | False | True | False | False | False |
| **2** | False | False | False | True | True |
| **3** | False | False | False | False | False |
| **4** | False | False | False | False | False |
| **5** | False | False | False | True | False |
| **6** | True | True | True | True | True |
| **7** | False | True | False | False | False |
| **8** | False | False | False | False | False |
| **9** | False | True | False | True | False |
| **10** | False | False | False | False | True |
| **11** | False | False | False | False | False |

## 3.删除缺失值

`df.dropna(axis=0, how='any', thresh=None, subset=None, inplace=False)`

- axis:{0或'index'，1或'columns'}，默认为0 确定是否删除了包含缺少值的行或列

```
*0或“索引”：删除包含缺少值的行。
*1或“列”：删除包含缺少值的列。
```

- how:{'any'，'all'}，默认为'any' 确定是否从DataFrame中删除行或列,至少一个NA或所有NA。

```
*“any”：如果存在任何NA值，请删除该行或列。

*“all”：如果所有值都是NA，则删除该行或列。
```

- thresh: int 需要至少非NA值数据个数。

- subset: 定义在哪些列中查找缺少的值
- inplace:是否更改源数据

```
df = pd.DataFrame({"name": ['Alfred', 'Batman', 'Catwoman'],
                   "toy": [np.nan, 'Batmobile', 'Bullwhip'],
                   "born": [pd.NaT, pd.Timestamp("1940-04-25"),
pd.NaT]})
df
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

| | name | toy | born |
|---|---|---|---|
| **0** | Alfred | NaN | NaT |
| **1** | Batman | Batmobile | 1940-04-25 |
| **2** | Catwoman | Bullwhip | NaT |

```
# 删除至少缺少一个元素的行。
df.dropna()
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

| | name | toy | born |
|---|---|---|---|
| **1** | Batman | Batmobile | 1940-04-25 |

```
# 删除至少缺少一个元素的列。
df.dropna(axis='columns')
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

| | name |
|---|---|
| **0** | Alfred |
| **1** | Batman |
| **2** | Catwoman |

```
# 删除缺少所有元素的行
df.dropna(how='all')
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

| | name | toy | born |
|---|---|---|---|
| **0** | Alfred | NaN | NaT |
| **1** | Batman | Batmobile | 1940-04-25 |
| **2** | Catwoman | Bullwhip | NaT |

```
# 仅保留至少有2个非NA值的行
df.dropna(thresh=2)
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

| | name | toy | born |
|---|---|---|---|
| **1** | Batman | Batmobile | 1940-04-25 |
| **2** | Catwoman | Bullwhip | NaT |

```
# 定义在哪些列中查找缺少的值
df.dropna(subset=['toy'])
```

| | name | toy | born |
|---|---|---|---|
| **1** | Batman | Batmobile | 1940-04-25 |
| **2** | Catwoman | Bullwhip | NaT |

```
# 在同一个变量中保留操作数据
df.dropna(inplace=True)
df
```

| | name | toy | born |
|---|---|---|---|
| 1 | Batman | Batmobile | 1940-04-25 |

## 4.缺失值补充

一般有用0填充，

用平均值填充，

用众数填充（大多数时候用这个），众数是指一组数据中出现次数最多的那个数据，一组数据可以有多个众数，也可以没有众数

向前填充(用缺失值的上一行对应字段的值填充，比如D3单元格缺失，那么就用D2单元格的值填充)、

向后填充(与向前填充对应)等方式。

```
df.fillna(
    value=None,
    method=None,
    axis=None,
    inplace=False,
    limit=None,
    downcast=None,
)
```

- value: 用于填充的值（例如0），或者是一个dict/Series/DataFrame值，指定每个索引（对于一个系列）或列（对于一个数据帧）使用哪个值。不在dict/Series/DataFrame中的值将不会被填充。此值不能是列表。
- method:ffill-->将上一个有效观察值向前传播  bfill-->将下一个有效观察值向后传播
- axis:用于填充缺失值的轴。
- inplace:是否操作源数据
- limit:要向前/向后填充的最大连续NaN值数

```
df = pd.DataFrame([
                    [np.nan, 2, np.nan, 0],

                   [3, 4, np.nan, 1],

                   [np.nan, np.nan, np.nan, np.nan],

                   [np.nan, 3, np.nan, 4]
                  ],
                  columns=list("ABCD")
                 )
df
```

| | A | B | C | D |
|---|---|---|---|---|
| **0** | NaN | 2.0 | NaN | 0.0 |
| **1** | 3.0 | 4.0 | NaN | 1.0 |
| **2** | NaN | NaN | NaN | NaN |
| **3** | NaN | 3.0 | NaN | 4.0 |

```
# 将所有NaN元素替换为0
df.fillna(0)
```

```
# 我们还可以向前或向后传播非空值
df.fillna(method="ffill")
```

| | A | B | C | D |
|---|---|---|---|---|
| **0** | NaN | 2.0 | NaN | 0.0 |
| **1** | 3.0 | 4.0 | NaN | 1.0 |
| **2** | 3.0 | 4.0 | NaN | 1.0 |
| **3** | 3.0 | 3.0 | NaN | 4.0 |

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

| | A | B | C | D |
|---|---|---|---|---|
| **0** | 3.0 | 2.0 | NaN | 0.0 |
| **1** | 3.0 | 4.0 | NaN | 1.0 |
| **2** | NaN | 3.0 | NaN | 4.0 |
| **3** | NaN | 3.0 | NaN | 4.0 |

```
# 将列“A”、“B”、“C”和“D”中的所有NaN元素分别替换为0、1、2和3。
values = {"A": 0, "B": 1, "C": 2, "D": 3}

df.fillna(value=values)
```

```
# 只替换第一个NaN元素
df.fillna(0, limit=1)
```

```
# 当使用数据填充时，替换会沿着相同的列名和索引进行
df2 = pd.DataFrame(np.random.rand(4,4), columns=list("ABCE"))
df2
```

| | A | B | C | E |
|---|---|---|---|---|
| **0** | 0.719214 | 0.038547 | 0.886814 | 0.705284 |
| **1** | 0.784260 | 0.114823 | 0.853345 | 0.468336 |
| **2** | 0.886700 | 0.373519 | 0.165613 | 0.215390 |
| **3** | 0.848140 | 0.926927 | 0.816328 | 0.435943 |

```
print(df)
df.fillna(value=df2)
```

```
     A    B   C    D
0  NaN  2.0 NaN  0.0
1  3.0  4.0 NaN  1.0
2  NaN  NaN NaN  NaN
3  NaN  3.0 NaN  4.0
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

| | A | B | C | D |
|---|---|---|---|---|
| **0** | 0.719214 | 2.000000 | 0.886814 | 0.0 |
| **1** | 3.000000 | 4.000000 | 0.853345 | 1.0 |
| **2** | 0.886700 | 0.373519 | 0.165613 | NaN |
| **3** | 0.848140 | 3.000000 | 0.816328 | 4.0 |

> 请注意，D列不受影响，因为它不在df2中。

```
作业:
    1. 针对`data_test.xlsx`文件,练习删除缺失值每个参数的使用
    2. 针对`data_test.xlsx`文件,练习缺失值补充每个参数的使用

以word文档的形式提交
```