# 缺失值填充

## 1. 产生原因

- 无意的：信息被遗漏，比如由于工作人员的疏忽，忘记而缺失；或者由于数据采集器等故障等原因造成的缺失，比如系统实时性要求较高的时候，机器来不及判断和决策而造成缺失；
- 有意的：有些数据集在特征描述中会规定将缺失值也作为一种特征值，这时候缺失值就可以看作是一种特殊的特征值；
- 不存在：有些特征属性根本就是不存在的，比如一个未婚者的配偶名字就没法填写，再如一个孩子的收入状况也无法填写

## 2. 缺失值类型

- 完全随机缺失（Missing Completely at Random） 指的是数据的缺失是完全随机的，不依赖于任何不完全变量或完全变量，不影响样本的无偏性，如家庭地址缺失；
- 随机缺失（Missing at Random） 指的是数据的缺失不是完全随机的，即该类数据的缺失依赖于其他完全变量（但和自己取值无关），如: 财务数据缺失情况与企业的大小有关；
- 非随机缺失（Missing not at Random） 指的是数据的缺失与不完全变量自身的取值有关，如高收入人群不原意提供家庭收入；
- ps:数据集中不含缺失值的变量称为完全变量，数据集中含有缺失值的变量称为不完全变量。

  ### 如何处理?
  对于随机缺失和非随机缺失，直接删除记录是不合适的，原因上面已经给出。随机缺失可以通过已知变量对缺失值进行估计，而非随机缺失的非随机性还没有很好的解决办法。

## 3. 常见方案：

- 删除记录 样本够多，且待删除的样本分布随机时使用。
- 数据填补
- 不处理

In [1]:

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

executed in 672ms, finished 01:01:33 2020-12-14

In [2]:

```
df = pd.DataFrame({'A':['a','b','c','d','e','f','g','h'],'B':[1,2,np.nan,4,np.nan,6,7,8]})
```

executed in 13ms, finished 01:01:41 2020-12-14

In [3]:

```
df.info()
```

executed in 27ms, finished 01:01:43 2020-12-14

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8 entries, 0 to 7
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   A       8 non-null      object
 1   B       6 non-null      float64
dtypes: float64(1), object(1)
memory usage: 256.0+ bytes
```

## 3.1 直接删除缺失记录

In [4]:

```
df_drop = df.dropna()
df_drop.info()
```

executed in 39ms, finished 01:04:18 2020-12-14

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 6 entries, 0 to 7
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   A       6 non-null      object
 1   B       6 non-null      float64
dtypes: float64(1), object(1)
memory usage: 144.0+ bytes
```

## 3.2 数据填补

### a. 插值填充

In [6]:

```
df_interp = df.interpolate()
print(df_interp.info())
df_interp
```

executed in 39ms, finished 01:06:13 2020-12-14

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8 entries, 0 to 7
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   A       8 non-null      object 
 1   B       8 non-null      float64
dtypes: float64(1), object(1)
memory usage: 256.0+ bytes
None
```

Out[6]:

| | A | B |
|---|---|---|
| **0** | a | 1.0 |
| **1** | b | 2.0 |
| **2** | c | 3.0 |
| **3** | d | 4.0 |
| **4** | e | 5.0 |
| **5** | f | 6.0 |
| **6** | g | 7.0 |
| **7** | h | 8.0 |

### b. 前向填充，后向填充

In [7]:

```
df_front = df.ffill()
df_behind = df.bfill()
print(df_front,'\n',df_behind)
```

executed in 22ms, finished 01:06:23 2020-12-14

```
   A    B
0  a  1.0
1  b  2.0
2  c  2.0
3  d  4.0
4  e  4.0
5  f  6.0
6  g  7.0
7  h  8.0
    A    B
0  a  1.0
1  b  2.0
2  c  4.0
3  d  4.0
4  e  6.0
5  f  6.0
6  g  7.0
7  h  8.0
```

### c.二次插值

如果数据是非线性的， 可以尝试

In [8]:

```
df_quadratic = df.interpolate(method="quadratic")
df_quadratic
```

executed in 305ms, finished 01:08:01 2020-12-14

Out[8]:

| | A | B |
|---|---|---|
| **0** | a | 1.0 |
| **1** | b | 2.0 |
| **2** | c | 3.0 |
| **3** | d | 4.0 |
| **4** | e | 5.0 |
| **5** | f | 6.0 |
| **6** | g | 7.0 |
| **7** | h | 8.0 |

### d. 使用sklearn的SimpleImputer方法进行单变量插补

主要参数说明:

missing_values：缺失值，可以为整数或NaN(缺失值numpy.nan用字符串'NaN'表示)，默认为NaN
strategy：替换策略，字符串，默认用均值'mean'替换
①若为mean时，用特征列的均值替换
②若为median时，用特征列的中位数替换
③若为most_frequent时，用特征列的众数替换

In [9]:

```
from sklearn.impute import SimpleImputer
# 均值填充
imp_mean = SimpleImputer(strategy='mean')
# 中位数填充
imp_median = SimpleImputer(strategy='median')
# 众数填充
imp_most = SimpleImputer(strategy='most_frequent')
```

executed in 1.05s, finished 01:09:35 2020-12-14

In [10]:

```
# 待填充特征
df[['B']]
```

executed in 10ms, finished 01:09:43 2020-12-14

Out[10]:

| | B |
|---|---|
| **0** | 1.0 |
| **1** | 2.0 |
| **2** | NaN |
| **3** | 4.0 |
| **4** | NaN |
| **5** | 6.0 |
| **6** | 7.0 |
| **7** | 8.0 |

In [11]:

```
df_mean= imp_mean.fit_transform(df[['B']])
print('均值：\n',df_mean)
df_median = imp_median.fit_transform(df[['B']])
print('中位数：\n',df_median)
df_most = imp_most.fit_transform(df[['B']])
print('众数：\n',df_most)  # 这里每个数只出现了一次，填充了第一个
```

executed in 24ms, finished 01:09:51 2020-12-14

```
均值：
 [[1.        ]
 [2.        ]
 [4.66666667]
 [4.        ]
 [4.66666667]
 [6.        ]
 [7.        ]
 [8.        ]]
中位数：
 [[1.]
 [2.]
 [5.]
 [4.]
 [5.]
 [6.]
 [7.]
 [8.]]
众数：
 [[1.]
 [2.]
 [1.]
 [4.]
 [1.]
 [6.]
 [7.]
 [8.]]
```

#### e. Kmeans聚类

- 按照完全变量将样本分为k类，各类填充各类的不完全变量的均值

#### f. 拟合插值

- 随机森林
- 线性回归
- K近邻
- 等机器学习回归算法

#### 以随机森林和KNN为例

#### (1) 随机森林插值填充

In [12]:

```
df = pd.DataFrame({'A':[1,2,3,4,5,6,7,8],'B':list(map(lambda x : x**2 ,[1,2,3,4,5,6,7,8])),'C':[1
df
```

executed in 13ms, finished 01:11:50 2020-12-14

Out[12]:

| | A | B | C |
|---|---|---|---|
| **0** | 1 | 1 | 1.0 |
| **1** | 2 | 4 | 2.0 |
| **2** | 3 | 9 | NaN |
| **3** | 4 | 16 | 4.0 |
| **4** | 5 | 25 | NaN |
| **5** | 6 | 36 | 6.0 |
| **6** | 7 | 49 | 7.0 |
| **7** | 8 | 64 | 8.0 |

In [13]:

```
from sklearn.ensemble import RandomForestRegressor
def set_missing_C(df_):

    # 把已有的数值型特征取出来丢进Random Forest Regressor中
    # 假装本来有很多列吧
    df = df_[['A','B','C']]

    known = df[df.C.notnull()].values
    unknown= df[df.C.isnull()].values

    # y即目标特征
    y = known[:,2]

    # X即特征属性值
    X = known[:,:2]

    # fit到RandomForestRegressor之中
    rfr = RandomForestRegressor(random_state=0, n_estimators=2000, n_jobs=-1)
    rfr.fit(X, y)

    # 用得到的模型进行未知结果预测
    predicted = rfr.predict(unknown[:,:2])

    # 用得到的预测结果填补原缺失数据
    df.loc[(df.C.isnull()), 'C' ] = predicted
    return df, rfr
```

executed in 65ms, finished 01:12:09 2020-12-14

In [14]:

```
set_missing_C(df)
```

executed in 2.91s, finished 01:12:18 2020-12-14

Out[14]:

```
(   A   B       C
 0  1   1  1.0000
 1  2   4  2.0000
 2  3   9  2.4445
 3  4  16  4.0000
 4  5  25  4.4925
 5  6  36  6.0000
 6  7  49  7.0000
 7  8  64  8.0000,
 RandomForestRegressor(n_estimators=2000, n_jobs=-1, random_state=0))
```

#### (2) KNN插值填充

通过欧几里德距离矩阵寻找最近邻，帮助估算观测中出现的缺失值。

In [15]:

```
from sklearn.impute import KNNImputer
df
```

executed in 15ms, finished 01:12:51 2020-12-14

Out[15]:

| | A | B | C |
|---|---|---|---|
| **0** | 1 | 1 | 1.0 |
| **1** | 2 | 4 | 2.0 |
| **2** | 3 | 9 | NaN |
| **3** | 4 | 16 | 4.0 |
| **4** | 5 | 25 | NaN |
| **5** | 6 | 36 | 6.0 |
| **6** | 7 | 49 | 7.0 |
| **7** | 8 | 64 | 8.0 |

In [16]:

```
imp_KNN = KNNImputer(n_neighbors=1)
imp_KNN.fit_transform(df)
```

executed in 28ms, finished 01:13:04 2020-12-14

Out[16]:

```
array([[ 1.,   1.,   1.],
       [ 2.,   4.,   2.],
       [ 3.,   9.,   2.],
       [ 4.,  16.,   4.],
       [ 5.,  25.,   4.],
       [ 6.,  36.,   6.],
       [ 7.,  49.,   7.],
       [ 8.,  64.,   8.]])
```

In [17]:

```
imp_KNN = KNNImputer(n_neighbors=2)# 这里正好是上下两个邻居的均值，恰好是3和5，注意连起来仅仅是巧合
imp_KNN.fit_transform(df)
```

executed in 14ms, finished 01:13:12 2020-12-14

Out[17]:

```
array([[ 1.,   1.,   1.],
       [ 2.,   4.,   2.],
       [ 3.,   9.,   3.],
       [ 4.,  16.,   4.],
       [ 5.,  25.,   5.],
       [ 6.,  36.,   6.],
       [ 7.,  49.,   7.],
       [ 8.,  64.,   8.]])
```

In [18]:

```
imp_KNN = KNNImputer(n_neighbors=3)
# (1+2+4)/3和(2+4+6)/3
imp_KNN.fit_transform(df)
```

executed in 18ms, finished 01:13:22 2020-12-14

Out[18]:

```
array([[ 1.        ,  1.        ,  1.        ],
       [ 2.        ,  4.        ,  2.        ],
       [ 3.        ,  9.        ,  2.33333333],
       [ 4.        , 16.        ,  4.        ],
       [ 5.        , 25.        ,  4.        ],
       [ 6.        , 36.        ,  6.        ],
       [ 7.        , 49.        ,  7.        ],
       [ 8.        , 64.        ,  8.        ]])
```

### 3.3 不处理

一些模型本身就可以应对具有缺失值的数据，此时无需对数据进行处理，比如Xgboost，rfr等模型。

## 4. reference

- 缺失值的四种处理方法 :https://bbs.pinggu.org/thread-3027700-1-1.html (https://bbs.pinggu.org/thread-3027700-1-1.html)
- python缺失值填充的几种方法 :https://blog.csdn.net/vivian_ll/article/details/91900323 (https://blog.csdn.net/vivian_ll/article/details/91900323)
- Python数据分析基础-数据缺失值处理：https://mp.weixin.qq.com/s?__biz=MzUzODYwMDAzNA==&mid=2247484441&idx=1&sn=0bfd745b551636c7d8776c2f98b64769&chksm (https://mp.weixin.qq.com/s?__biz=MzUzODYwMDAzNA==&mid=2247484441&idx=1&sn=0bfd745b551636c7d8776c2f98b64769&chksm
- sklearn 文档：http://scikitlearn.com.cn/0.21.3/41/#542 (http://scikitlearn.com.cn/0.21.3/41/#542)
- KNNImputer,一种可靠的缺失值插补方法 : https://www.cnblogs.com/panchuangai/p/13390354.html (https://www.cnblogs.com/panchuangai/p/13390354.html)
- 机器学习中处理缺失值的7种方法 : https://blog.csdn.net/lrzh0123/article/details/110530837?utm_medium=distribute.pc_relevant.none-task-blog-baidulandingword-3&spm=1001.2101.3001.4242 (https://blog.csdn.net/lrzh0123/article/details/110530837?utm_medium=distribute.pc_relevant.none-task-blog-baidulandingword-3&spm=1001.2101.3001.4242)

In [ ]:

```

```