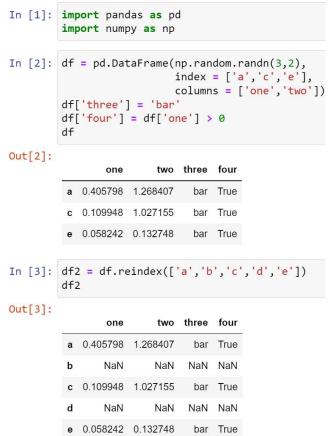
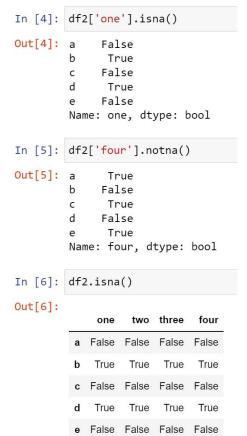


Pandas – missing data - NA

NaN is the default missing value but there is a need to be able to easily detect this value with data of different types: floating point, integer, boolean, and general object. In many cases the Python **None** arise and we need to also consider that "missing" or "not available" or "NA". To make detecting missing values easier, pandas provides the **isna**() and **notna**() functions, which are also methods on Series and DataFrame objects:







Pandas – missing data - NA

In Python (and NumPy), the nan's don't compare equal, but None's do. Pandas/NumPy uses the fact that np.nan!= np.nan, and treats None like np.nan.

```
In [7]: None == None
Out[7]: True
In [8]: np.nan == np.nan
Out[8]: False
```

Because NaN is a float, a column of integers with even one missing values is cast to floating-point dtype. Pandas provides a nullable integer array, which can be used by explicitly requesting the dtype.

```
In [9]: pd.Series([1,2,np.nan,4])
Out[9]: 0    1.0
    1    2.0
    2    NaN
    3    4.0
    dtype: float64
```



Pandas – missing data - Datetimes

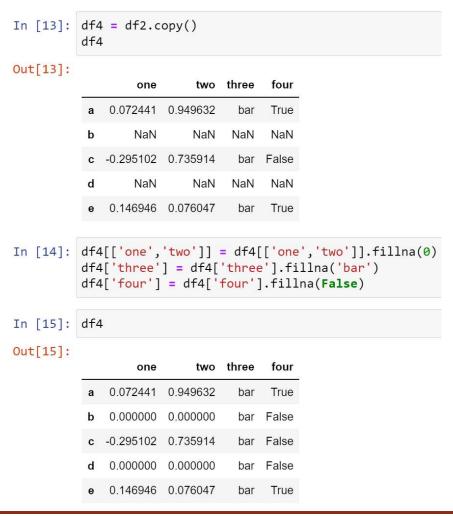
For datetime64[ns] types, NaT represents missing values. This is a pseudo-native sentinel value that can be represented by NumPy in a singular dtype. Pandas objects provide compatibility between NaT and NaN

```
In [11]: df3 = df.copy()
          df3['timestamp'] = pd.Timestamp('2021/12/25')
          df3.loc['c','timestamp'] = np.nan
          df3
Out[11]:
                                      four timestamp
                  one
                           two three
             1.602123 0.445034
                                      True 2021-12-25
                                 bar
             -0.336193 0.093943
                                 bar False
                                                 NaT
             0.380750 -0.598101
                                      True 2021-12-25
                                 bar
In [12]: df3.dtypes.value counts()
Out[12]: float64
                             2
          object
                             1
          bool
          datetime64[ns]
                             1
          dtype: int64
```



Pandas – filling missing data – fillna()

fillna() can "fill in" NA values with non-NA data in a couple of ways.





Pandas – filling missing data – fillna()

We can propagate non-NA values forward or backward:

Method	Action
pad / ffill	Fill values forward
bfill / backfill	Fill values backward

```
In [16]: df5 = df2.copy()
    df5.loc['c', 'two'] = np.nan
    df5
```

Out[16]:

	one	two	three	four
а	1.066309	0.070883	bar	True
b	NaN	NaN	NaN	NaN
С	-0.175732	NaN	bar	False
d	NaN	NaN	NaN	NaN
е	1.044383	-1.030072	bar	True

```
In [17]: df5['one'] = df5['one'].fillna(method = "pad")
    df5['two'] = df5['two'].fillna(method = "bfill", limit = 2)
    df5
```

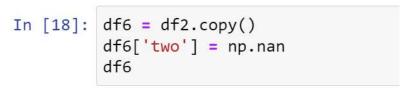
Out[17]:

	one	two	three	four
а	1.066309	0.070883	bar	True
b	1.066309	NaN	NaN	NaN
С	-0.175732	-1.030072	bar	False
d	-0.175732	-1.030072	NaN	NaN
е	1.044383	-1.030072	bar	True



Pandas – dropping missing data – dropna()

Exclude labels from a data set which refer to missing data



Out[18]:

	one	two	three	four
а	-0.440424	NaN	bar	False
b	NaN	NaN	NaN	NaN
С	0.442517	NaN	bar	True
d	NaN	NaN	NaN	NaN
е	-0.384090	NaN	bar	False

