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:

```
In [1]: import pandas as pd
        import numpy as np
```

```
In [2]: df = pd.DataFrame(np.random.randn(3,2),
                          index = ['a','c','e'],
                          columns = ['one','two'])
        df['three'] = 'bar'
        df['four'] = df['one'] > 0
        df
```

Out[2]:

	one	two	three	four
a	0.405798	1.268407	bar	True
c	0.109948	1.027155	bar	True
e	0.058242	0.132748	bar	True

```
In [3]: df2 = df.reindex(['a','b','c','d','e'])
        df2
```

Out[3]:

	one	two	three	four
a	0.405798	1.268407	bar	True
b	NaN	NaN	NaN	NaN
c	0.109948	1.027155	bar	True
d	NaN	NaN	NaN	NaN
e	0.058242	0.132748	bar	True

```
In [4]: df2['one'].isna()
```

```
Out[4]: a    False
        b     True
        c    False
        d     True
        e    False
        Name: one, dtype: bool
```

```
In [5]: df2['four'].notna()
```

```
Out[5]: a     True
        b    False
        c     True
        d    False
        e     True
        Name: four, dtype: bool
```

```
In [6]: df2.isna()
```

Out[6]:

	one	two	three	four
a	False	False	False	False
b	True	True	True	True
c	False	False	False	False
d	True	True	True	True
e	False	False	False	False

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
```

```
In [10]: pd.Series([1,2,np.nan,4], dtype = 'Int64')
Out[10]: 0       1
         1       2
         2    <NA>
         3       4
         dtype: Int64
```

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]:

	one	two	three	four	timestamp
a	1.602123	0.445034	bar	True	2021-12-25
c	-0.336193	0.093943	bar	False	NaT
e	0.380750	-0.598101	bar	True	2021-12-25

```
In [12]: df3.dtypes.value_counts()
```

```
Out[12]: float64           2
         object            1
         bool              1
         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.

```
In [13]: df4 = df2.copy()
         df4
```

Out[13]:

	one	two	three	four
a	0.072441	0.949632	bar	True
b	NaN	NaN	NaN	NaN
c	-0.295102	0.735914	bar	False
d	NaN	NaN	NaN	NaN
e	0.146946	0.076047	bar	True

```
In [14]: df4[['one','two']] = df4[['one','two']].fillna(0)
         df4['three'] = df4['three'].fillna('bar')
         df4['four'] = df4['four'].fillna(False)
```

```
In [15]: df4
```

Out[15]:

	one	two	three	four
a	0.072441	0.949632	bar	True
b	0.000000	0.000000	bar	False
c	-0.295102	0.735914	bar	False
d	0.000000	0.000000	bar	False
e	0.146946	0.076047	bar	True

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
a	1.066309	0.070883	bar	True
b	NaN	NaN	NaN	NaN
c	-0.175732	NaN	bar	False
d	NaN	NaN	NaN	NaN
e	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
a	1.066309	0.070883	bar	True
b	1.066309	NaN	NaN	NaN
c	-0.175732	-1.030072	bar	False
d	-0.175732	-1.030072	NaN	NaN
e	1.044383	-1.030072	bar	True

Pandas – dropping missing data – dropna()

Exclude labels from a data set which refer to missing data

```
In [18]: df6 = df2.copy()
         df6['two'] = np.nan
         df6
```

Out[18]:

	one	two	three	four
a	-0.440424	NaN	bar	False
b	NaN	NaN	NaN	NaN
c	0.442517	NaN	bar	True
d	NaN	NaN	NaN	NaN
e	-0.384090	NaN	bar	False

```
In [19]: df6.dropna(how = 'all', inplace = True)
         df6
```

Out[19]:

	one	two	three	four
a	-0.440424	NaN	bar	False
c	0.442517	NaN	bar	True
e	-0.384090	NaN	bar	False

```
In [20]: df6.dropna(axis=1, inplace = True)
         df6
```

Out[20]:

	one	three	four
a	-0.440424	bar	False
c	0.442517	bar	True
e	-0.384090	bar	False