Handling missing data

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

Pandas Utility functions:

```
Similarly to numpy, pandas also has a few utility functions to
identify and detect null values:
```

In [ ]:
```
print(pd.isnull(np.nan), pd.isnull(None), pd.isna(np.nan), pd.isna(None))
```

```
True True True True
```

the opposite also exists:

In [ ]:
```
print(pd.notnull(None), pd.notnull(np.nan), pd.notna(None), pd.notna(np.nan), pd.n
```

```
False False False False True
```

These also work with series and Dataframes

In [ ]:
```
print(pd.isnull(pd.Series([1, np.nan, 7])),'\n', pd.notnull(pd.Series([1, np.nan,
```

```
0    False
1     True
2    False
dtype: bool
 0     True
1    False
2     True
dtype: bool
```

In [ ]:
```
pd.isnull(pd.DataFrame({
    'Column A': [1, np.nan, 7],
    'Column B': [np.nan, 2, 3],
    'Column C': [np.nan, 2, np.nan]
}))
```

Out[ ]:

	Column A	Column B	Column C
0	False	True	True
1	True	False	False
2	False	False	True

Filtering missing data

In [ ]:
```
s = pd.Series([1, 2, 3, np.nan, np.nan, 4])
s
```

Out[ ]:
```
0    1.0
1    2.0
2    3.0
3    NaN
4    NaN
5    4.0
dtype: float64
```

In [ ]:
```
print("Table with boolean for not null: ",'\n',pd.notnull(s),'\n','Table with bool
```

```
Table with boolean for not null:
 0     True
1     True
2     True
3    False
4    False
5     True
dtype: bool
 Table with boolean for null:
 0    False
1    False
2    False
3     True
4     True
5    False
dtype: bool
```

In [ ]:
```
print("number of null entries: ",pd.isnull(s).sum(),'\n','number of allowed entrie
```

```
number of null entries:  2
 number of allowed entries:  4
```

In [ ]:
```
s[pd.notnull(s)]
```

Out[ ]:
```
0    1.0
1    2.0
2    3.0
5    4.0
dtype: float64
```

Both isnull and notnull are also functions of dataframes and series, hence we can use them directly!

In [ ]:
```
s[s.notnull()]
```

Out[ ]:
```
0    1.0
1    2.0
2    3.0
5    4.0
dtype: float64
```

Dropping null values from dataframes:

In [ ]:
```
df = pd.DataFrame({
    'Column A': [1, np.nan, 30, np.nan],
    'Column B': [2, 8, 31, np.nan],
    'Column C': [np.nan, 9, 32, 100],
    'Column D': [5, 8, 34, 110],
})
df
```

Out[ ]:

	Column A	Column B	Column C	Column D
0	1.0	2.0	NaN	5
1	NaN	8.0	9.0	8
2	30.0	31.0	32.0	34
3	NaN	NaN	100.0	110

In [ ]:
```
df.isnull()
```

Out[ ]:

	Column A	Column B	Column C	Column D
0	False	False	True	False
1	True	False	False	False
2	False	False	False	False
3	True	True	False	False

In [ ]:
```
df=df.rename(columns={'Column A':'a','Column B':'b','Column C':'c','Column D':'d'}
```

In [ ]:
```
df
```

Out[ ]:

	a	b	c	d
0	1.0	2.0	NaN	5
1	NaN	8.0	9.0	8
2	30.0	31.0	32.0	34
3	NaN	NaN	100.0	110

In [ ]:
```
df.dropna() #this ends up dropping all rows with any *null* value
```

Out[ ]:

	a	b	c	d
2	30.0	31.0	32.0	34

In [ ]:
```
df.dropna(axis='columns') #removes all columns with any null value
```

Out[ ]:

	d
0	5
1	8
2	34
3	110

In [ ]:
```
df2 = pd.DataFrame({
    'Column A': [1, np.nan, 30],
    'Column B': [2, np.nan, 31],
    'Column C': [np.nan, np.nan, 100]
})
df2
```

Out[ ]:

	Column A	Column B	Column C
0	1.0	2.0	NaN
1	NaN	NaN	NaN
2	30.0	31.0	100.0

now we'll see "how"-> 'any' means that it'll drop all rows/columns with a null value 'all' means that it'll drop all rows/columns with ALL null values

In [ ]:
```
df2.dropna(how='any')
```

Out[ ]:

	Column A	Column B	Column C
2	30.0	31.0	100.0

In [ ]:
```
df2.dropna(how='all')
```

Out[ ]:

	Column A	Column B	Column C
0	1.0	2.0	NaN
2	30.0	31.0	100.0

In [ ]:
```
df2.dropna(thresh=3) #how many non-na values to keep by default looks at indexes (
```

Out[ ]:

	Column A	Column B	Column C
2	30.0	31.0	100.0

In [ ]:
```
df.dropna(thresh=3, axis='columns')
```

Out[ ]:

	b	c	d
0	2.0	NaN	5
1	8.0	9.0	8
2	31.0	32.0	34
3	NaN	100.0	110

Filling out null values:

```
Sometimes instead than dropping the null values, we might need to
replace them with some other value. This highly depends on your
context and the dataset you're currently working. Sometimes a nan
can be replaced with a 0, sometimes it can be replaced with the
mean of the sample, and some other times you can take the closest
value. Again, it depends on the context. We'll show you the
different methods and mechanisms and you can then apply them to
your own problem.
```

In [ ]:
```
s
```

Out[ ]:
```
0    1.0
1    2.0
2    3.0
3    NaN
4    NaN
5    4.0
dtype: float64
```

In [ ]:
```
s.fillna(0) #replaces all Nan values with 0
```

Out[ ]:
```
0    1.0
1    2.0
2    3.0
3    0.0
4    0.0
5    4.0
dtype: float64
```

In [ ]:
```
s.fillna(s.mean())
```

Out[ ]:
```
0    1.0
1    2.0
2    3.0
3    2.5
4    2.5
5    4.0
dtype: float64
```

The "method argument is used to fill null values with other values close to that null one:

In [ ]:
```
s.fillna(method='ffill') #f forward
```

Out[ ]:
```
0    1.0
1    2.0
2    3.0
3    3.0
4    3.0
5    4.0
dtype: float64
```

In [ ]:
```
s.fillna(method='bfill') # b backwards
```

Out[ ]:
```
0    1.0
1    2.0
2    3.0
3    4.0
4    4.0
5    4.0
dtype: float64
```

the following method always leaves null values at an extreme end (maybe even both!)

In [ ]:
```
pd.Series([np.nan, 3, np.nan, 9]).fillna(method='ffill')
```

Out[ ]:
```
0    NaN
1    3.0
2    3.0
3    9.0
dtype: float64
```

Filna also works with DataFrames:

The fillna method also works on DataFrames, and it works similarly. The main differences are that you can specify the axis (as usual, rows or columns) to use to fill the values (specially for methods) and that you have more control on the values passed:

In [ ]:
```
df
```

Out[ ]:

	a	b	c	d
0	1.0	2.0	NaN	5
1	NaN	8.0	9.0	8
2	30.0	31.0	32.0	34
3	NaN	NaN	100.0	110

In [ ]:
```
df.fillna({'a': 0, 'b': 99, 'c': df['c'].mean()})
```

Out[ ]:

	a	b	c	d
0	1.0	2.0	47.0	5
1	0.0	8.0	9.0	8
2	30.0	31.0	32.0	34
3	0.0	99.0	100.0	110

In [ ]:
```
df.fillna(method='ffill', axis=0)  #you can see here how it forgot an extreme!
#AXIS=0 IS COLUMNS AND AXIS=1 IS ROW BASED.
```

Out[ ]:

	a	b	c	d
0	1.0	2.0	NaN	5
1	1.0	8.0	9.0	8
2	30.0	31.0	32.0	34
3	30.0	31.0	100.0	110

Checking if there are any NAs:

The question is: Does this Series or DataFrame contain any missing value? The answer should be yes or no: True or False. How can you verify it?

In [ ]:
```
s.dropna().count()
```

Out[ ]:
```
4
```

In [ ]:
```
missing_values = len(s.dropna()) != len(s)
missing_values
```

Out[ ]:
```
True
```

Pythonic solutions:

The methods any and all check if either there's any true value in a Series or all the values are True.

```
In [ ]: print(pd.Series([True, False, False]).any(), pd.Series([True, True, True]).all())
```

```
True True
```

```
In [ ]: pd.Series([1, 2]).isnull().any()
```

```
Out[ ]: False
```

```
In [ ]: s.isnull().values
```

```
Out[ ]: array([False, False, False,  True,  True, False])
```

```
In [ ]: s.isnull().values.any()
```

```
Out[ ]: True
```