Missing Data

Hands on!

In [1]:

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

What does "missing data" mean? What is a missing value? It depends on the origin of the data and the context it was generated. For example, for a survey, a `Salary` field with an empty value, or a number 0, or an invalid value (a string for example) can be considered "missing data". These concepts are related to the values that Python will consider "Falsy":

In [ ]:

```
falsy_values = (0, False, None, '', [], {})
```

For Python, all the values above are considered "falsy":

In [ ]:

```
any(falsy_values)
```

Numpy has a special "nullable" value for numbers which is `np.nan` . It's *NaN*: "Not a number"

In [ ]:

```
np.nan
```

The `np.nan` value is kind of a virus. Everything that it touches becomes `np.nan`:

In [ ]:

```
3 + np.nan
```

In [ ]:

```
a = np.array([1, 2, 3, np.nan, np.nan, 4])
```

In [ ]:

```
a.sum()
```

In [ ]:

```
a.mean()
```

This is better than regular `None` values, which in the previous examples would have raised an exception:

In [ ]:

```
3 + None
```

For a numeric array, the `None` value is replaced by `np.nan`:

In [ ]:

```
a = np.array([1, 2, 3, np.nan, None, 4], dtype='float')
```

In [ ]:

```
a
```

As we said, `np.nan` is like a virus. If you have any `nan` value in an array and you try to perform an operation on it, you'll get unexpected results:

In [ ]:

```
a = np.array([1, 2, 3, np.nan, np.nan, 4])
```

In [ ]:

```
a.mean()
```

In [ ]:

```
a.sum()
```

Numpy also supports an "Infinite" type:

In [ ]:

```
np.inf
```

Which also behaves as a virus:

In [ ]:

```
3 + np.inf
```

In [ ]:

```
np.inf / 3
```

In [ ]:

```
np.inf / np.inf
```

In [ ]:

```
b = np.array([1, 2, 3, np.inf, np.nan, 4], dtype=np.float)
```

In [ ]:

```
b.sum()
```

Checking for `nan` or `inf`

There are two functions: `np.isnan` and `np.isinf` that will perform the desired checks:

In [ ]:

```
np.isnan(np.nan)
```

In [ ]:

```
np.isinf(np.inf)
```

And the joint operation can be performed with `np.isfinite` .

In [ ]:

```
np.isfinite(np.nan), np.isfinite(np.inf)
```

`np.isnan` and `np.isinf` also take arrays as inputs, and return boolean arrays as results:

In [ ]:

```
np.isnan(np.array([1, 2, 3, np.nan, np.inf, 4]))
```

In [ ]:

```
np.isinf(np.array([1, 2, 3, np.nan, np.inf, 4]))
```

In [ ]:

```
np.isfinite(np.array([1, 2, 3, np.nan, np.inf, 4]))
```

Note: It's not so common to find infinite values. From now on, we'll keep working with only `np.nan`

Filtering them out

Whenever you're trying to perform an operation with a Numpy array and you know there might be missing values, you'll need to filter them out before proceeding, to avoid `nan` propagation. We'll use a combination of the previous `np.isnan` + boolean arrays for this purpose:

In [ ]:

```
a = np.array([1, 2, 3, np.nan, np.nan, 4])
```

In [ ]:

```
a[~np.isnan(a)]
```

Which is equivalent to:

In [ ]:

```
a[np.isfinite(a)]
```

And with that result, all the operation can be now performed:

In [ ]:

```
a[np.isfinite(a)].sum()
```

In [ ]:

```
a[np.isfinite(a)].mean()
```