(http://www.pieriandata.com)

Missing Data

Make sure to review the video for a full discussion on the strategies of dealing with missing data.

What Null/NA/nan objects look like:

Source: https://github.com/pandas-dev/pandas/issues/28095 (https://github.com/pandas-dev/pandas/issues/28095)

A new pd.NA value (singleton) is introduced to represent scalar missing values. Up to now, pandas used several values to represent missing data: np.nan is used for this for float data, np.nan or None for object-dtype data and pd.NaT for datetime-like data. The goal of pd.NA is to provide a “missing” indicator that can be used consistently across data types. pd.NA is currently used by the nullable integer and boolean data types and the new string data type

In [127]:
```
import numpy as np
import pandas as pd
```

In [128]:
```
np.nan
```

Out[128]:
```
nan
```

In [129]:
```
pd.NA
```

Out[129]:
```
<NA>
```

In [130]:
```
pd.NaT
```

Out[130]:
```
NaT
```

Note! Typical comparisons should be avoided with Missing Values

- https://towardsdatascience.com/navigating-the-hell-of-nans-in-python-71b12558895b (https://towardsdatascience.com/navigating-the-hell-of-nans-in-python-71b12558895b)
- https://stackoverflow.com/questions/20320022/why-in-numpy-nan-nan-is-false-while-nan-in-nan-is-true (https://stackoverflow.com/questions/20320022/why-in-numpy-nan-nan-is-false-while-nan-in-nan-is-true)

This is generally because the logic here is, since we don't know these values, we can't know

In [131]:
```
np.nan == np.nan
```

Out[131]:
```
False
```

In [132]:
```
np.nan in [np.nan]
```

Out[132]:
```
True
```

In [133]:
```
np.nan is np.nan
```

Out[133]:
```
True
```

In [134]:
```
pd.NA == pd.NA
```

Out[134]:
```
<NA>
```

Data

People were asked to score their opinions of actors from a 1-10 scale before and after watching one of their movies. However, some data is missing.

In [135]:
```
df = pd.read_csv('movie_scores.csv')
```

In [136]:
```
df
```

Out[136]:

	first_name	last_name	age	sex	pre_movie_score	post_movie_score
0	Tom	Hanks	63.0	m	8.0	10.0
1	NaN	NaN	NaN	NaN	NaN	NaN
2	Hugh	Jackman	51.0	m	NaN	NaN
3	Oprah	Winfrey	66.0	f	6.0	8.0
4	Emma	Stone	31.0	f	7.0	9.0

Checking and Selecting for Null Values

In [137]:
```
df
```

Out[137]:

	first_name	last_name	age	sex	pre_movie_score	post_movie_score
0	Tom	Hanks	63.0	m	8.0	10.0
1	NaN	NaN	NaN	NaN	NaN	NaN
2	Hugh	Jackman	51.0	m	NaN	NaN
3	Oprah	Winfrey	66.0	f	6.0	8.0
4	Emma	Stone	31.0	f	7.0	9.0

In [138]:
```
df.isnull()
```

Out[138]:

	first_name	last_name	age	sex	pre_movie_score	post_movie_score
0	False	False	False	False	False	False
1	True	True	True	True	True	True
2	False	False	False	False	True	True
3	False	False	False	False	False	False
4	False	False	False	False	False	False

In [139]:
```
df.notnull()
```

Out[139]:

	first_name	last_name	age	sex	pre_movie_score	post_movie_score
0	True	True	True	True	True	True
1	False	False	False	False	False	False
2	True	True	True	True	False	False
3	True	True	True	True	True	True
4	True	True	True	True	True	True

In [140]:
```
df['first_name']
```

Out[140]:
```
0      Tom
1      NaN
2     Hugh
3    Oprah
4     Emma
Name: first_name, dtype: object
```

In [141]:
```
df[df['first_name'].notnull()]
```

Out[141]:

	first_name	last_name	age	sex	pre_movie_score	post_movie_score
0	Tom	Hanks	63.0	m	8.0	10.0
2	Hugh	Jackman	51.0	m	NaN	NaN
3	Oprah	Winfrey	66.0	f	6.0	8.0
4	Emma	Stone	31.0	f	7.0	9.0

In [142]:
```
df[(df['pre_movie_score'].isnull()) & df['sex'].notnull()]
```

Out[142]:

	first_name	last_name	age	sex	pre_movie_score	post_movie_score
2	Hugh	Jackman	51.0	m	NaN	NaN

Drop Data

In [143]:
```
df
```

Out[143]:

	first_name	last_name	age	sex	pre_movie_score	post_movie_score
0	Tom	Hanks	63.0	m	8.0	10.0
1	NaN	NaN	NaN	NaN	NaN	NaN
2	Hugh	Jackman	51.0	m	NaN	NaN
3	Oprah	Winfrey	66.0	f	6.0	8.0
4	Emma	Stone	31.0	f	7.0	9.0

In [144]:
```
help(df.dropna)
```

```
Help on method dropna in module pandas.core.frame:

dropna(axis=0, how='any', thresh=None, subset=None, inplace=False) method
of pandas.core.frame.DataFrame instance
    Remove missing values.

    See the :ref:`User Guide <missing_data>` for more on which values are
    considered missing, and how to work with missing data.

    Parameters
    ----------
    axis : {0 or 'index', 1 or 'columns'}, default 0
        Determine if rows or columns which contain missing values are
        removed.

        * 0, or 'index' : Drop rows which contain missing values.
        * 1, or 'columns' : Drop columns which contain missing value.

        .. versionchanged:: 1.0.0

           Pass tuple or list to drop on multiple axes.
           Only a single axis is allowed.

    how : {'any', 'all'}, default 'any'
        Determine if row or column is removed from DataFrame, when we have
        at least one NA or all NA.

        * 'any' : If any NA values are present, drop that row or column.
        * 'all' : If all values are NA, drop that row or column.

    thresh : int, optional
        Require that many non-NA values.
    subset : array-like, optional
        Labels along other axis to consider, e.g. if you are dropping rows
        these would be a list of columns to include.
    inplace : bool, default False
        If True, do operation inplace and return None.

    Returns
    -------
    DataFrame
        DataFrame with NA entries dropped from it.

    See Also
    --------
    DataFrame.isna: Indicate missing values.
    DataFrame.notna : Indicate existing (non-missing) values.
    DataFrame.fillna : Replace missing values.
    Series.dropna : Drop missing values.
    Index.dropna : Drop missing indices.

    Examples
    --------
    >>> df = pd.DataFrame({"name": ['Alfred', 'Batman', 'Catwoman'],
    ...                    "toy": [np.nan, 'Batmobile', 'Bullwhip'],
    ...                    "born": [pd.NaT, pd.Timestamp("1940-04-25"),
    ...                             pd.NaT]})
    >>> df
           name        toy       born
    0    Alfred        NaN        NaT
    1    Batman  Batmobile 1940-04-25
```

```
2  Catwoman   Bullwhip        NaT

Drop the rows where at least one element is missing.

>>> df.dropna()
     name        toy       born
1  Batman  Batmobile 1940-04-25

Drop the columns where at least one element is missing.

>>> df.dropna(axis='columns')
       name
0    Alfred
1    Batman
2  Catwoman

Drop the rows where all elements are missing.

>>> df.dropna(how='all')
       name        toy       born
0    Alfred        NaN        NaT
1    Batman  Batmobile 1940-04-25
2  Catwoman   Bullwhip        NaT

Keep only the rows with at least 2 non-NA values.

>>> df.dropna(thresh=2)
       name        toy       born
1    Batman  Batmobile 1940-04-25
2  Catwoman   Bullwhip        NaT

Define in which columns to look for missing values.

>>> df.dropna(subset=['name', 'born'])
       name        toy       born
1    Batman  Batmobile 1940-04-25

Keep the DataFrame with valid entries in the same variable.

>>> df.dropna(inplace=True)
>>> df
     name        toy       born
1  Batman  Batmobile 1940-04-25
```

In [145]:
```
df.dropna()
```

Out[145]:

	first_name	last_name	age	sex	pre_movie_score	post_movie_score
0	Tom	Hanks	63.0	m	8.0	10.0
3	Oprah	Winfrey	66.0	f	6.0	8.0
4	Emma	Stone	31.0	f	7.0	9.0

In [146]:
```
df.dropna(thresh=1)
```

Out[146]:

	first_name	last_name	age	sex	pre_movie_score	post_movie_score
0	Tom	Hanks	63.0	m	8.0	10.0
2	Hugh	Jackman	51.0	m	NaN	NaN
3	Oprah	Winfrey	66.0	f	6.0	8.0
4	Emma	Stone	31.0	f	7.0	9.0

In [147]:
```
df.dropna(axis=1)
```

Out[147]:

0
1
2
3
4

In [148]:
```
df.dropna(thresh=4,axis=1)
```

Out[148]:

	first_name	last_name	age	sex
0	Tom	Hanks	63.0	m
1	NaN	NaN	NaN	NaN
2	Hugh	Jackman	51.0	m
3	Oprah	Winfrey	66.0	f
4	Emma	Stone	31.0	f

Fill Data

In [149]:
```
df
```

Out[149]:

	first_name	last_name	age	sex	pre_movie_score	post_movie_score
0	Tom	Hanks	63.0	m	8.0	10.0
1	NaN	NaN	NaN	NaN	NaN	NaN
2	Hugh	Jackman	51.0	m	NaN	NaN
3	Oprah	Winfrey	66.0	f	6.0	8.0
4	Emma	Stone	31.0	f	7.0	9.0

In [150]:
```
df.fillna("NEW VALUE!")
```

Out[150]:

	first_name	last_name	age	sex	pre_movie_score	post_movie_score
0	Tom	Hanks	63	m	8	10
1	NEW VALUE!	NEW VALUE!	NEW VALUE!	NEW VALUE!	NEW VALUE!	NEW VALUE!
2	Hugh	Jackman	51	m	NEW VALUE!	NEW VALUE!
3	Oprah	Winfrey	66	f	6	8
4	Emma	Stone	31	f	7	9

In [151]:
```
df['first_name'].fillna("Empty")
```

Out[151]:
```
0      Tom
1    Empty
2     Hugh
3    Oprah
4     Emma
Name: first_name, dtype: object
```

In [152]:
```
df['first_name'] = df['first_name'].fillna("Empty")
```

In [153]:
```
df
```

Out[153]:

	first_name	last_name	age	sex	pre_movie_score	post_movie_score
0	Tom	Hanks	63.0	m	8.0	10.0
1	Empty	NaN	NaN	NaN	NaN	NaN
2	Hugh	Jackman	51.0	m	NaN	NaN
3	Oprah	Winfrey	66.0	f	6.0	8.0
4	Emma	Stone	31.0	f	7.0	9.0

In [154]:
```
df['pre_movie_score'].mean()
```

Out[154]:
```
7.0
```

In [155]:
```
df['pre_movie_score'].fillna(df['pre_movie_score'].mean())
```

Out[155]:
```
0    8.0
1    7.0
2    7.0
3    6.0
4    7.0
Name: pre_movie_score, dtype: float64
```

In [156]:
```
df.fillna(df.mean())
```

Out[156]:

	first_name	last_name	age	sex	pre_movie_score	post_movie_score
0	Tom	Hanks	63.00	m	8.0	10.0
1	Empty	NaN	52.75	NaN	7.0	9.0
2	Hugh	Jackman	51.00	m	7.0	9.0
3	Oprah	Winfrey	66.00	f	6.0	8.0
4	Emma	Stone	31.00	f	7.0	9.0

Filling with Interpolation

Be careful with this technique, you should try to really understand whether or not this is a valid choice for your data. You should also note there are several methods available, the default is a linear method.

Full Docs on this Method:
https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.interpolate.html
(https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.interpolate.html)

In [164]:
```
airline_tix = {'first':100,'business':np.nan,'economy-plus':50,'economy':30
```

In [165]:
```
ser = pd.Series(airline_tix)
```

In [166]:
```
ser
```

Out[166]:
```
first           100.0
business          NaN
economy-plus     50.0
economy          30.0
dtype: float64
```

In [167]:
```
ser.interpolate()
```

Out[167]:
```
first           100.0
business         75.0
economy-plus     50.0
economy          30.0
dtype: float64
```

In [163]:
```
ser.interpolate(method='spline')
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-163-106f2287918c> in <module>
----> 1 ser.interpolate(method='spline')

c:\users\marcial\anaconda3\envs\ml_master\lib\site-packages\pandas\core\generic.py in interpolate(self, method, axis, limit, inplace, limit_direction, limit_area, downcast, **kwargs)
   6992             if method not in methods and not is_numeric_or_datetime:
   6993                 raise ValueError(
-> 6994                     "Index column must be numeric or datetime type when "
   6995                     f"using {method} method other than linear. "
   6996                     "Try setting a numeric or datetime index column before "

ValueError: Index column must be numeric or datetime type when using spline method other than linear. Try setting a numeric or datetime index column before interpolating.
```

In [169]:
```
df = pd.DataFrame(ser,columns=['Price'])
```

In [170]:
```
df
```

Out[170]:

	Price
first	100.0
business	NaN
economy-plus	50.0
economy	30.0

In [171]:
```
df.interpolate()
```

Out[171]:

	Price
first	100.0
business	75.0
economy-plus	50.0
economy	30.0

In [174]:
```
df = df.reset_index()
```

In [175]:
```
df
```

Out[175]:

	index	Price
0	first	100.0
1	business	NaN
2	economy-plus	50.0
3	economy	30.0

In [178]:
```
df.interpolate(method='spline',order=2)
```

Out[178]:

	index	Price
0	first	100.000000
1	business	73.333333
2	economy-plus	50.000000
3	economy	30.000000