# Part_3_Cheatsheet_for_Handling_Missing_Values

October 31, 2022

[16]:
```
import pandas as pd
import numpy as np

df = pd.read_csv('/content/sample_data/movie_scores.csv')
df.head()
```

[16]:

| | 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 |

### 0.1 Checking missing data in the dataframe:

[8]:
```
# It will tell which values are null in the dataframe

df.isnull()
```

[8]:

| | 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 |

[10]:
```
# It will tell which values are not null in the dataframe

df.notnull()
```

[10]:

| | 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 |

[11]:
```
# To check missing values of a column

df[df['pre_movie_score'].isnull()]
```

[11]:

| | first_name | last_name | age | sex | pre_movie_score | post_movie_score |
|---|---|---|---|---|---|---|
| 1 | NaN | NaN | NaN | NaN | NaN | NaN |
| 2 | Hugh | Jackman | 51.0 | m | NaN | NaN |

[12]:
```
# To get rows that has missing values in two columns

df[(df['pre_movie_score'].isnull()) & (df['first_name'].isnull())]
```

[12]:

| | first_name | last_name | age | sex | pre_movie_score | post_movie_score |
|---|---|---|---|---|---|---|
| 1 | NaN | NaN | NaN | NaN | NaN | NaN |

[13]:
```
# To get rows that has missing values in either columns

df[(df['pre_movie_score'].isnull()) | (df['first_name'].isnull())]
```

[13]:

| | first_name | last_name | age | sex | pre_movie_score | post_movie_score |
|---|---|---|---|---|---|---|
| 1 | NaN | NaN | NaN | NaN | NaN | NaN |
| 2 | Hugh | Jackman | 51.0 | m | NaN | NaN |

### 0.2 Methods of handling missing data:

Many Machine Learning models cannot work with missing data. When reading in missing values, Pandas display them as **NaN** for normal values and **pd.NaT** for the missing timestamp values.

Following methods are available to handle missing data:

1. Keep the missing data
2. Dropping all missing data

- `Dropping a row`
- `Dropping a column`

3. Filling in missing data

- `Fill with same value`
- `Fill with interpolated/estimated values`

**Nota Bene:** To check if value of a variable is null or not, we cannot write:

```
myvar = np.nan
```

because in Python it is wrong. So to verify, always use:

```
myvar is np.nan
```

It tells correctly if the variable is null or not.

### 0.3 1. Keep the Missing Data:

**Advantages:**

- Easiest to do
- Does not manipulate or change the true data

**Disdvantages:**

- Many ML models doesn't support NaN values
- Often there are reasonable guesses that we can make to handle missing data. So its better to use them

### 0.4 2. Dropping all missing data:

**Advantages:**

- Easy to do

**Disadvantages:**

- Potential to lose a lot of data or useful information
- Limits the use of such trained models to use for future data

[17]:
```
# to drop all missing data from the dataframe

# to make changes permanent use: df.dropna(inplace = True)

df.dropna()
```

[17]:

| | 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 |

[18]:
```
# to drop only those rows which has at least 2 NaN values

df.dropna(thresh=2)
```

[18]:

| | 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 |

**By default, rows get dropped as axis = 0 is default. To delete columns, use axis=1.**

[19]:
```
df.dropna(axis=1)
```

[19]:
```
Empty DataFrame
Columns: []
Index: [0, 1, 2, 3, 4]
```

*It dropped all columns which means all columns were having NaN values.*

To delete rows which has NaN values in a specific column, use **subset = [ columnName ]**

[20]:
```
df.dropna(subset = ['last_name'])
```

[20]:

| | 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 |

## 0.5 3. Filling in missing data:

**Advantages:**

- Potential to save a lot of data for use in training model

**Disadvantages:**

- Hardest to do and somewhat arbitrary
- Potential to lead to false conclusions

**1. Filling with same value:**

It is a good choice if NaN was used as a placeholder for some value e.g. 0. Means the data collector used NaN for the values which has 0 records. So we can fill all the NaN values with 0 which is value able to be processed in ML Models.

**2. Filling with interpolated or estimated values:**

It is much harder and requires reasonable assumptions & domain experience e.g. calculating Percentage. We can take values of some other columns of the dataframe and fill the missing values of percentage column.

Or, it fills the NaN values with interpolating the upper and lower value of NaN values.

[21]:
```
df.interpolate(method = 'spline', order=2)
```

[21]:

| | first_name | last_name | age | sex | pre_movie_score | post_movie_score |
|---|---|---|---|---|---|---|
| 0 | Tom | Hanks | 63.000000 | m | 8.000000 | 10.000000 |
| 1 | NaN | NaN | 48.588688 | NaN | 6.500000 | 8.500000 |
| 2 | Hugh | Jackman | 51.000000 | m | 5.833333 | 7.833333 |
| 3 | Oprah | Winfrey | 66.000000 | f | 6.000000 | 8.000000 |
| 4 | Emma | Stone | 31.000000 | f | 7.000000 | 9.000000 |

***We have used spline method. By default, linear method is used that is only method which is available to use for multi-indexes.***