Handling Missing values in Data set

Missing data handling is coming under topic of data **pre-processing.**

When we are working on data in the job of Data science or Data analyst, we should expect the data should be cleaned. And it helps in stage of data 'Prediction/forecast' to get more accuracy in outcome(results).

Types of missing values:

Before knowing types of missing values, we should understand the meaning, use, relation of each variable in the dataset.

Name	**Gender**	**Age**	**Education**	**Salary**	**Hobbies**	**City**
NameA	Male	19	Intermediate	2 LPA	Music	CityA
NameB	Male	26	B.Tech	4 LPA	Cricket	
NameC	Female	29	M.Tech	6.5 LPA		CityC
NameD	Male	35			TV	CityD
NameE	Female		B.Tech	7 LPA	Cooking	CityE

- **MCAR**: It stands for Missing completely at random. The reason behind the missing value is not dependent on any other features(variables). In other words, there is no particular reason for the missing values.

  Ex: From the above table If you see 'Hobbies' and 'City' variables are having null values and are seems independent.

- **MAR**: It stands for Missing at random. The reason behind the missing value may be associated with some other features.

  Ex: From the above table If you see 'Education' and 'Salary' variables are having null values and are seems dependent.

- **MNAR**: It stands for Missing not at random. There is a specific reason behind the missing value.

  Ex: From the above table If you see 'Age' variable having null value and is seems might she don't want to mention her age.

How to check missing data in dataset:

Here we are using python for practical examples.

#to check count of missing values for each column

df.isnull().sum()

#to check count of missing values for single column

df[‘column-name1’].isnull().sum()

#to check count of missing values for multiple columns

df[[‘column-name1’,’column-name2’]].isnull().sum()

#to check percentage of missing values for multiple columns

(df[[‘column-name1’,’column-name2’]].isnull().sum())*100/len(df)

#to check percentage of missing values for each category in a column

df[‘column-name1’].value_counts(normalize=True)

Missing data handling methods:

There multiple methods to handle the missing data for 2 types of data problem.

1. Non-time series data
2. Time series data

Non-Time series data:

Data which considering without time(date) based for analysis.

Step1: **Set values as missing values**

Identify the missing data unique values in each column, sometimes missing data replace with ‘NA’,’NaN’,’nan’,’xxxx’,’999’, ‘blank’ etc. after identifying and confirming the names of missing values we should replace those with unique(common) name, because it will help us in the first step of analysis.

Step2**: Adding is good, exaggerating is bad**

You should try to get information from reliable external sources as much as possible, but if you can’t, then it is better to retain missing values rather than exaggerating the existing rows/columns.

Step3: Here, we have 2 methods to do deal with missing data are **Dilation** and/or **Imputation**.

Dilation:

1. When entire row or column are missing values.

#dropping rows data based on a column missing value.

df = df[~df[‘column-name1’].isnull()]

2. If our objective is to ignore missing values in the calculations then simply replace all the missing values with NaN. Then all the summary statistics like mean, median, mode etc. will ignore the missing values.

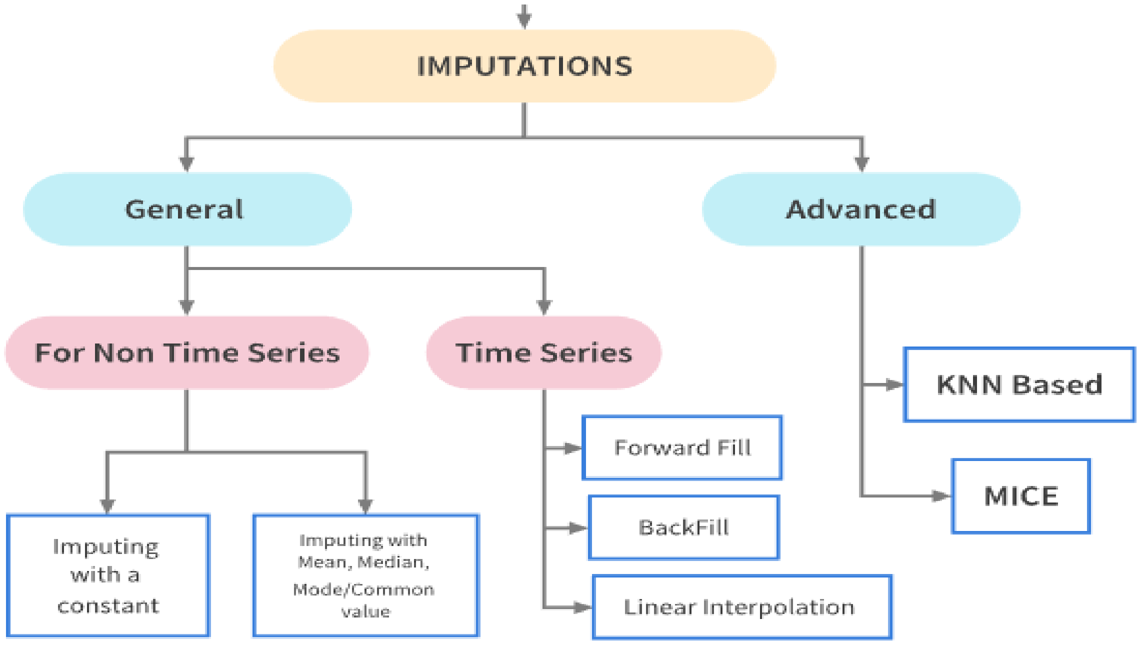
#describe the data before replacing NaN values

df['column-name1'].describe()

#replace NaN values
#If missing value is '-1', because missing values always doesn't have to present as null.

```
import numpy as np
df.loc[df['column-name1'] < 0,"column-name1"] = np.NaN
df['column-name'].describe()
```

Fill missing values (Imputation):



1. Filling with mean:
   #If column is numerical
   df['column-name1'].fillna(np.mean(df['column-name1'],inplace=True)
2. Filing with median:
   df['column-name1'].fillna(np.median(df['column-name1'],inplace=True)
3. Filling with mode:
   df['column-name1'].fillna(df['column-name1'].mode[0],inplace=True)
4. Imputing with a constant:

```
from sklearn.impute import SimpleImputer
#setting strategy to 'constant'
mean_imputer = SimpleImputer(strategy='constant')
df.iloc[:,:] = mean_imputer.fit_transform(df)
df.isnull().sum()
```

Time series data:

Now let's look at ways to impute data in a typical time series problem. Tackling missing values in time Series problem is a bit different. The fillna() method is used for imputing missing values in such problems.

Basic Imputation Techniques.

- 'ffill' or 'pad' - Replace NaN s with last observed value
- 'bfill' or 'backfill' - Replace NaN s with next observed value
- Linear interpolation method

Date	Product	Sales	Total income
1-10-2020	Dairy milk(100gm)	19	
2-10-2020	Dairy milk(100gm)	26	3120
3-10-2020	Dairy milk(100gm)	29	3480
4-10-2020	Dairy milk(100gm)	35	4200
5-10-2020	Dairy milk(100gm)	24	
6-10-2020	Dairy milk(100gm)		

From above table, Total income = sales*price of product → sales*120

If you know relation between each column and logic, then you can apply your own logic for filling missing value.

1. **Imputation using ffill**

```
df.fillna(method='ffill',inplace=True)
df

#result
```

Date	Product	Sales	Total income
1-10-2020	Dairy milk(100gm)	19	
2-10-2020	Dairy milk(100gm)	26	3120
3-10-2020	Dairy milk(100gm)	29	3480
4-10-2020	Dairy milk(100gm)	35	4200
5-10-2020	Dairy milk(100gm)	24	4200
6-10-2020	Dairy milk(100gm)	24	4200

2. **Imputation using bfill**

```
df.fillna(method='bfill',inplace=True)
df

#result
```

Date	Product	Sales	Total income
1-10-2020	Dairy milk(100gm)	19	3120
2-10-2020	Dairy milk(100gm)	26	3120
3-10-2020	Dairy milk(100gm)	29	3480
4-10-2020	Dairy milk(100gm)	35	4200
5-10-2020	Dairy milk(100gm)	24	
6-10-2020	Dairy milk(100gm)		

3. **Imputation using Linear Interpolation method**

Time series data has a lot of variations against time. Hence, imputing using backfill and forward fill isn't the ebst possible solution to address the missing value problem. A more apt alternative would be to use interpolation methods, where the values are filled with incrementing or decrementing values.

Linear interpolation is an imputation technique that assumes a linear relationship between data points and utilises non-missing values from adjacent data points to compute a value for a missing data point.

```
# Interpolate using the linear method
df.interpolate(limit_direction="both",inplace=True)
```

4. **Advanced Imputation Techniques**

Advanced imputation techniques uses machine learning algorithms to impute the missing values in a dataset unlike the previous techniques where we used other column values to predict the missing values. We shall look at the following two techniques in this notebook:

- Nearest neighbors imputation
- Multivariate feature imputation

K-Nearest Neighbor Imputation

The KNNImputer class provides imputation for filling in missing values using the k-Nearest Neighbors approach.Each missing feature is imputed using values from n_neighbors nearest neighbors that have a value for the feature. The feature of the neighbors is averaged uniformly or weighted by distance to each neighbor.

```
from sklearn.impute import KNNImputer
df_knn = df.copy(deep=True)

knn_imputer = KNNImputer(n_neighbors=2, weights="uniform")
df_knn[‘Total income’] = knn_imputer.fit_transform(df_knn[[‘Total income’]])
df_knn[‘Total income’].isnull().sum()
```

Multivariate feature imputation - Multivariate imputation by chained equations (MICE)

A strategy for imputing missing values by modeling each feature with missing values as a function of other features in a round-robin fashion. It performns multiple regressions over random sample ofthe data, then takes the average ofthe multiple regression values and uses that value to impute the missing value. In sklearn, it is implemented as follows:

```
from sklearn.experimental import enable_iterative_imputer
from sklearn.impute import IterativeImputer
train_mice = train.copy(deep=True)

mice_imputer = IterativeImputer()
df_mice[‘Total income’] = mice_imputer.fit_transform(train_mice[[‘Total income’]])
df_mice[‘Total income’].isnull().sum()
```

Note: In Machine learning XGBoost and LightGBM algorithms are can handle missing values automatically.