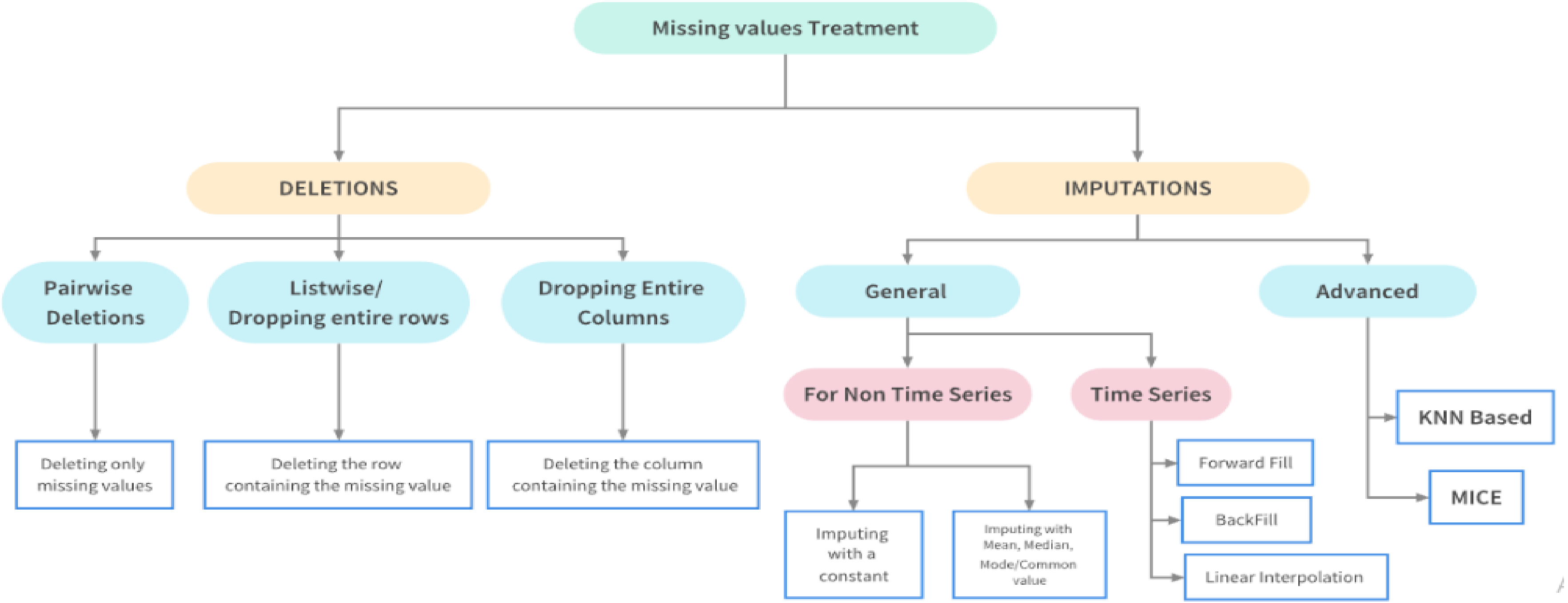
Handling Missing Values in Python



Real world data is messy and often contains a lot of missing values. There could be multiple reasons for the missing values but primarily the reason for missing-ness can be attributed to

Either way we need to address this issue before we proceed with the modelling stuff. It is also important to note that some algorithms like XGBoost and LightGBM can treat missing data without any pre-processing.

Syed Afroz Ali (Data Scientist)
https://www.kaggle.com/pythonafroz
https://www.linkedin.com/in/syed-afroz-70939914/

Reasons for Missing Values

Before we start treating the missing values, it is important to understand the various reasons for the missing-ness in data. Broadly speaking, there can be three possible reasons:

1. Missing Completely at Random (MCAR)

The missing values on a given variable (Y) are not associated with other variables in a given data set or with the variable (Y) itself. In other words, there is no particular reason for the missing values.

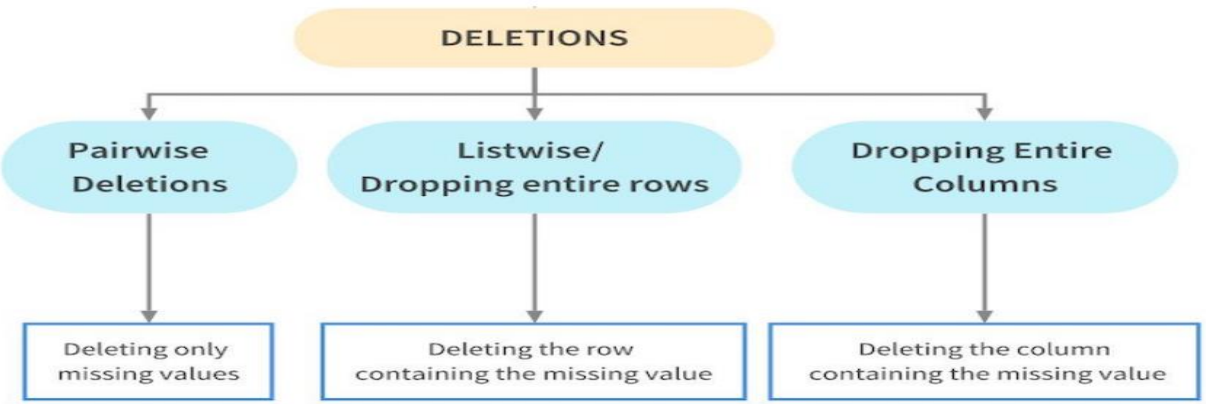
2. Missing at Random (MAR)

MAR occurs when the missing-ness is not random, but where missing-ness can be fully accounted for by variables where there is complete information.

3. Missing Not at Random (MNAR)

Missing-ness depends on unobserved data or the value of the missing data itself.

Deletions



Deletion means to delete the missing values from a dataset. This is however not recommended as it might result in loss of information from the dataset. We should only delete the missing values from a dataset if their proportion is very small. Deletions are further of three types:

Syed Afroz Ali (Data Scientist)
https://www.kaggle.com/pythonafroz
https://www.linkedin.com/in/syed-afroz-70939914/

Pairwise Deletion

Pairwise Deletion is used when values are missing completely at random i.e. MCAR. During Pairwise deletion, only the missing values are deleted. All operations in pandas like mean, sum etc. intrinsically skips missing values.

List wise Deletion/ Dropping rows

During List wise deletion, complete rows (which contain the missing values) are deleted. As a result, it is also called Complete Case deletion. Like Pairwise deletion, list wise deletions are also only used for MCAR values.

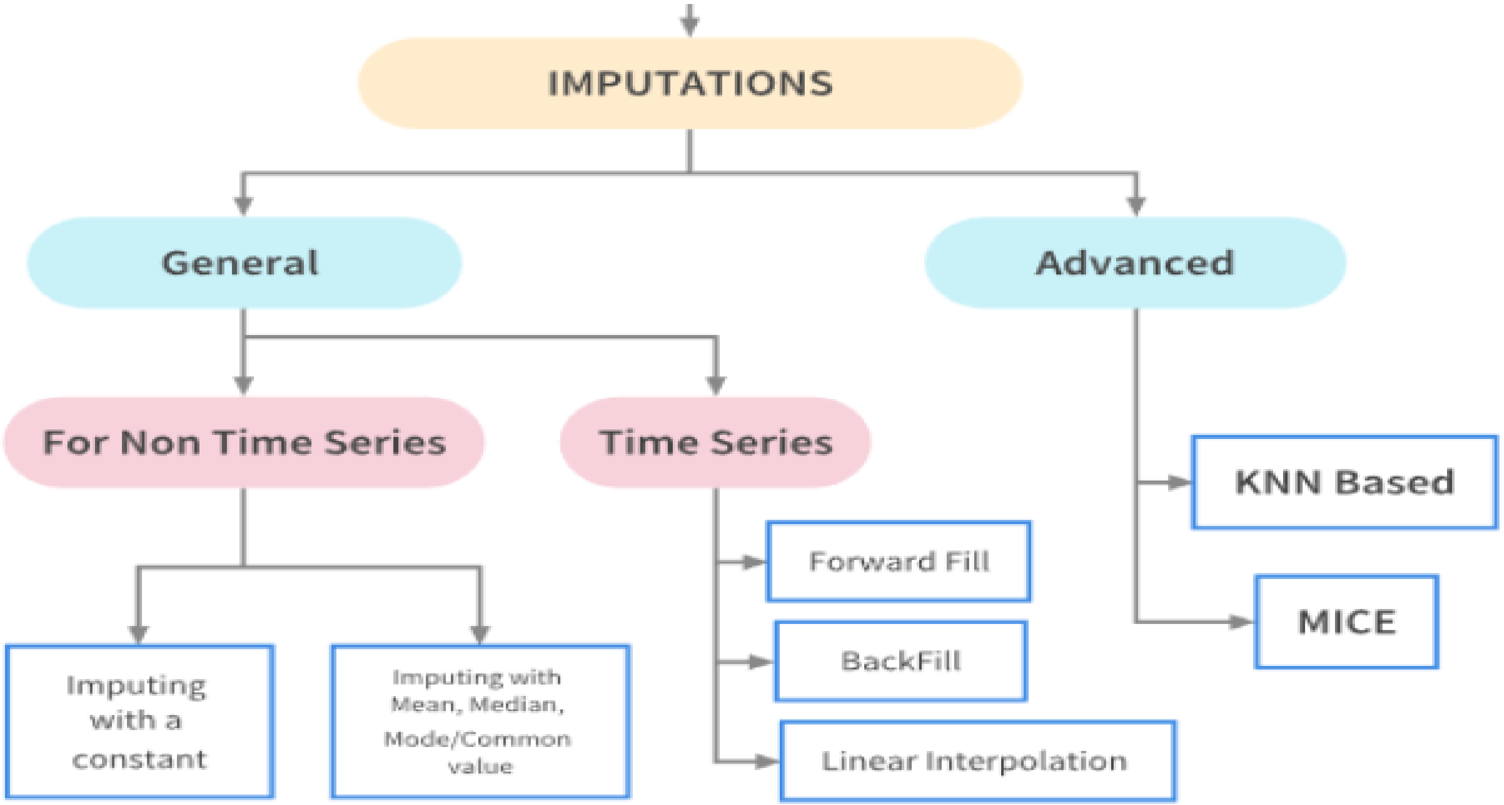
```
#Drop rows which contains any NaN or missing value for Age column
train_1.dropna(subset=['Age'],how='any',inplace=True)
train_1['Age'].isnull().sum()
```

The Age column doesn't have any missing values. A major disadvantage of List wise deletion is that a major chunk of data and hence a lot of information is lost. Hence, it is advisable to use it only when the number of missing values is very small.

Dropping complete columns

If a column contains a lot of missing values, say more than 80%, and the feature is not significant, you might want to delete that feature. However, again, it is not a good methodology to delete data.

Imputations Techniques for non-Time Series Problems



Imputation refers to replacing missing data with substituted values. There are a lot of ways in which the missing values can be imputed depending upon the nature of the problem and data. Depending upon the nature of the problem, imputation techniques can be broadly they can be classified as follows:

Syed Afroz Ali (Data Scientist)
https://www.kaggle.com/pythonafroz
https://www.linkedin.com/in/syed-afroz-70939914/

Basic Imputation Techniques

- Imputating with a constant value
- Imputation using the statistics (mean, median or most frequent) of each column in which the missing values are located

For this we shall use the `The SimpleImputer` class from sklearn.

```
# imputing with a constant
from sklearn.impute import SimpleImputer
train_constant = train.copy()
mean_imputer = SimpleImputer(strategy='constant')
train_constant.iloc[:,:] = mean_imputer.fit_transform(train_constant)
train_constant.isnull().sum()

from sklearn.impute import SimpleImputer
train_most_frequent = train.copy()
#setting strategy to 'mean' to impute by the mean
mean_imputer = SimpleImputer(strategy='most_frequent')#strategy can also be mean or median
train_most_frequent.iloc[:,:] = mean_imputer.fit_transform(train_most_frequent)
train_most_frequent.isnull().sum()
```

Imputations Techniques for Time Series Problems

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

Syed Afroz Ali (Data Scientist)
https://www.kaggle.com/pythonafroz
https://www.linkedin.com/in/syed-afroz-70939914/

Time Series dataset

The dataset is called [Air Quality Data in India (2015 - 2020)] Tand it contains air quality data and AQI (Air Quality Index) at hourly and daily level of various stations across multiple cities in India. The dataset has a lot of missing values and is a classic Time series problem.

```
city_day['Xylene'][50:64]
```

```
Date
2015-02-20     7.48
2015-02-21    15.44
2015-02-22     8.47
2015-02-23    28.46
2015-02-24     6.05
2015-02-25     0.81
2015-02-26      NaN
2015-02-27      NaN
2015-02-28      NaN
2015-03-01     1.32
2015-03-02     0.22
```

```
city_day.fillna(method='ffill',inplace=True)
city_day['Xylene'][50:65]
```

```
Date
2015-02-20     7.48
2015-02-21    15.44
2015-02-22     8.47
2015-02-23    28.46
2015-02-24     6.05
2015-02-25     0.81
2015-02-26     0.81
2015-02-27     0.81
2015-02-28     0.81
2015-03-01     1.32
```

Syed Afroz Ali (Data Scientist)
https://www.kaggle.com/pythonafroz
https://www.linkedin.com/in/syed-afroz-70939914/

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 best 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.

```
city_day1['Xylene'][50:65]
```

```
Date
2015-02-20     7.48
2015-02-21    15.44
2015-02-22     8.47
2015-02-23    28.46
2015-02-24     6.05
2015-02-25     0.81
2015-02-26      NaN
2015-02-27      NaN
2015-02-28      NaN
2015-03-01     1.32
```

```
# Interpolate using the linear method
city_day1.interpolate(limit_direction="both",inplace=True)
city_day1['Xylene'][50:65]
```

```
Date
2015-02-20     7.4800
2015-02-21    15.4400
2015-02-22     8.4700
2015-02-23    28.4600
2015-02-24     6.0500
2015-02-25     0.8100
2015-02-26     0.9375
2015-02-27     1.0650
2015-02-28     1.1925
2015-03-01     1.3200
```


Syed Afroz Ali (Data Scientist)
https://www.kaggle.com/pythonafroz
https://www.linkedin.com/in/syed-afroz-70939914/

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.

K-Nearest Neighbour Imputation

The KNN-Imputer class provides imputation for filling in missing values using the k-Nearest Neighbours approach. Each missing feature is imputed using values from n_neighbors nearest neighbour's that have a value for the feature. The feature of the neighbours are averaged uniformly or weighted by distance to each neighbour.

```
train_knn = train.copy(deep=True)
from sklearn.impute import KNNImputer
train_knn = train.copy(deep=True)
knn_imputer = KNNImputer(n_neighbors=2, weights="uniform")
train_knn['Age'] = knn_imputer.fit_transform(train_knn[['Age']])
train_knn['Age'].isnull().sum()
```

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

A strategy for imputing missing, values by modelling each feature with missing values, as a function of other features in a round-robin fashion. It performs multiple regressions over random sample of the data, then takes the average of the 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()
train_mice['Age'] = mice_imputer.fit_transform(train_mice[['Age']])
train_mice['Age'].isnull().sum()
```

Syed Afroz Ali (Data Scientist)
https://www.kaggle.com/pythonafroz
https://www.linkedin.com/in/syed-afroz-70939914/

Algorithms which handle missing values

Some algorithms like XGBoost and LightGBM can handle missing values without any pre-processing, by supplying relevant parameters.

https://xgboost.readthedocs.io/en/latest/python/python_api.html#module-xgboost.sklearn

Core XGBoost Library.

***class* `xgboost.DMatrix` (*data, label=None, weight=None, base_margin=None, missing=None, silent=False, feature_names=None, feature_types=None, nthread=None*)**

Bases: `object`

Data Matrix used in XGBoost.

DMatrix is a internal data structure that used by XGBoost which is optimized for both memory efficiency and training speed. You can construct DMatrix from numpy.arrays

Parameters

- **data** (*os.PathLike/string/numpy.array/scipy.sparse/pd.DataFrame/*) – dt.Frame/cudf.DataFrame/cupy.array/dlpack Data source of DMatrix. When data is string or os.PathLike type, it represents the path libsvm format txt file, csv file (by specifying uri parameter 'path_to_csv?format=csv'), or binary file that xgboost can read from.
- **label** (*list, numpy 1-D array or cudf.DataFrame, optional*) – Label of the training data.
- **missing** (*float, optional*) – Value in the input data which needs to be present as a missing value. If None, defaults to np.nan.
- **weight** (*list, numpy 1-D array or cudf.DataFrame , optional*) –

  Weight for each instance.

Missing Value Handle

- LightGBM enables the missing value handle by default. Disable it by setting `use_missing=false`.
- LightGBM uses NA (NaN) to represent missing values by default. Change it to use zero by setting `zero_as_missing=true`.
- When `zero_as_missing=false` (default), the unshown values in sparse matrices (and LightSVM) are treated as zeros.
- When `zero_as_missing=true`, NA and zeros (including unshown values in sparse matrices (and LightSVM)) are treated as missing.

Syed Afroz Ali (Data Scientist)
https://www.kaggle.com/pythonafroz
https://www.linkedin.com/in/syed-afroz-70939914/