1. The [**SimpleImputer**](https://scikit-learn.org/stable/modules/generated/sklearn.impute.SimpleImputer.html#sklearn.impute.SimpleImputer) class provides basic strategies for imputing missing values. Missing values can be imputed with a provided constant value, or using the statistics (mean, median or most frequent) of each column in which the missing values are located.
2. **KNNImputer (Note: For Categorical features - Encoding needs to be done)**

The [**KNNImputer**](https://scikit-learn.org/stable/modules/generated/sklearn.impute.KNNImputer.html#sklearn.impute.KNNImputer) uses the k-Nearest Neighbors approach. By default, a euclidean distance metric that supports missing values, ***nan\_euclidean\_distances***, is used to find the nearest neighbors. Each missing feature is imputed using values from n\_neighbors nearest neighbors that have a value for the feature. The features of the neighbors are ***averaged*** uniformly or weighted by distance to each neighbor. If a sample has more than one feature missing, then the

neighbors for that sample can be different depending on the particular feature being imputed.

(Scale the values before imputation)

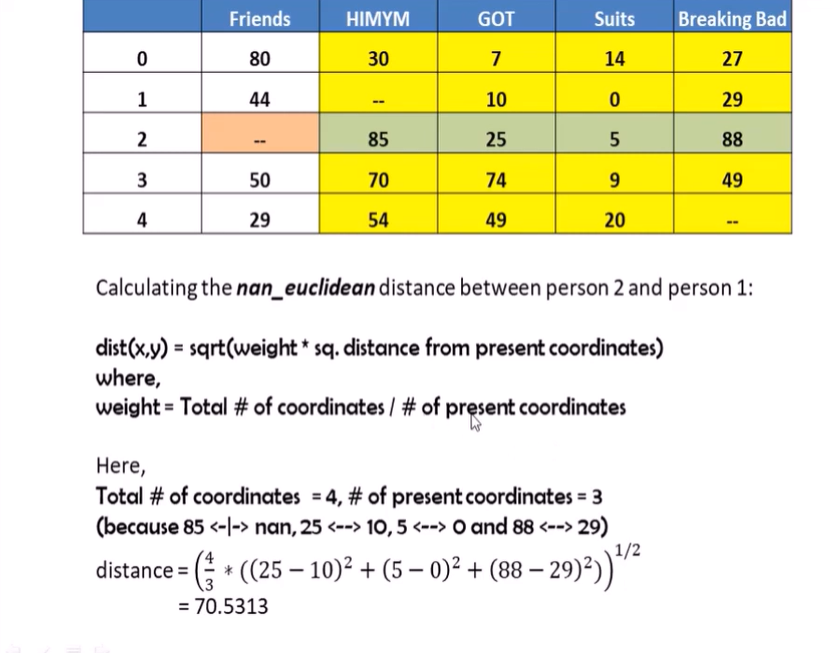
**Workflow:**

Step 1: Choose a missing value to fill

Step 2: Select the other values in that row

Step 3: Choose the no.of neighbors (n=?)

Step 4: Calculate nan\_euclidian distance from all other corresponding row elements.

**Example:**

In the shown example,

If n=2, Four calculations will be performed (i.e., person 2 with persons 0,1,3,4). Among the results, the **mean** of the smallest two values (as n=2) is taken.

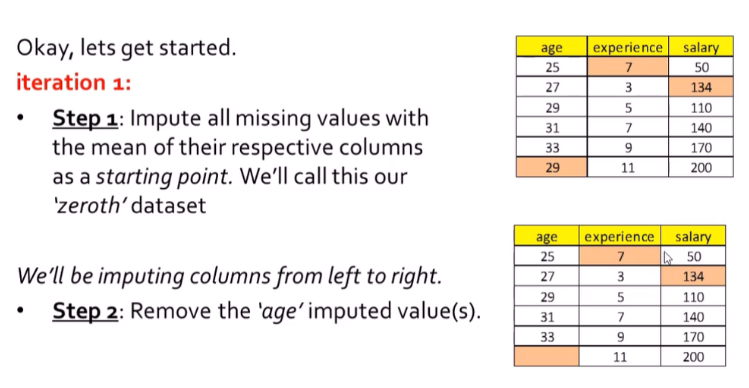
[**https://scikit-learn.org/stable/modules/impute.html**](https://scikit-learn.org/stable/modules/impute.html)

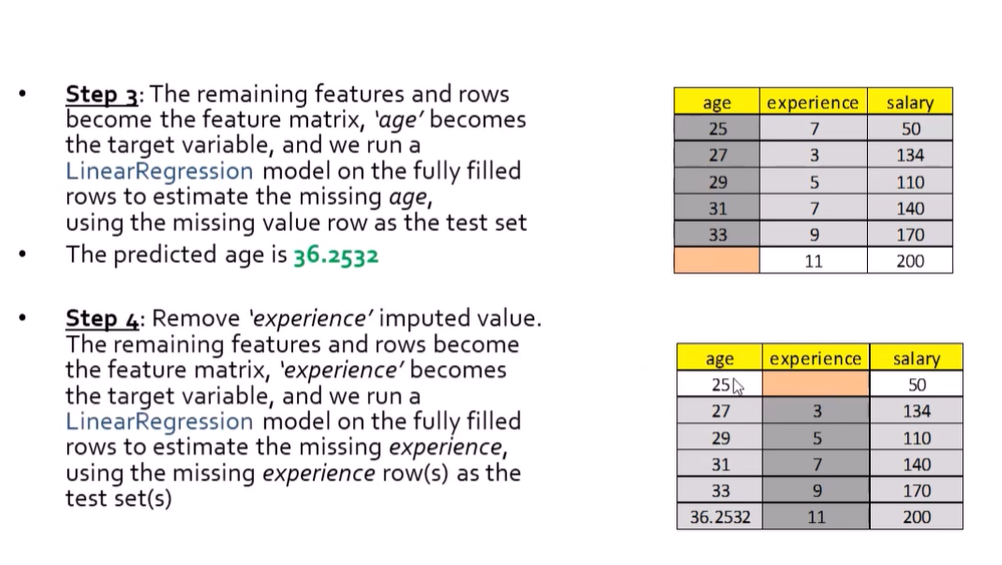
1. **Iterative Imputer / MICE Imputer (default iterations=10)**

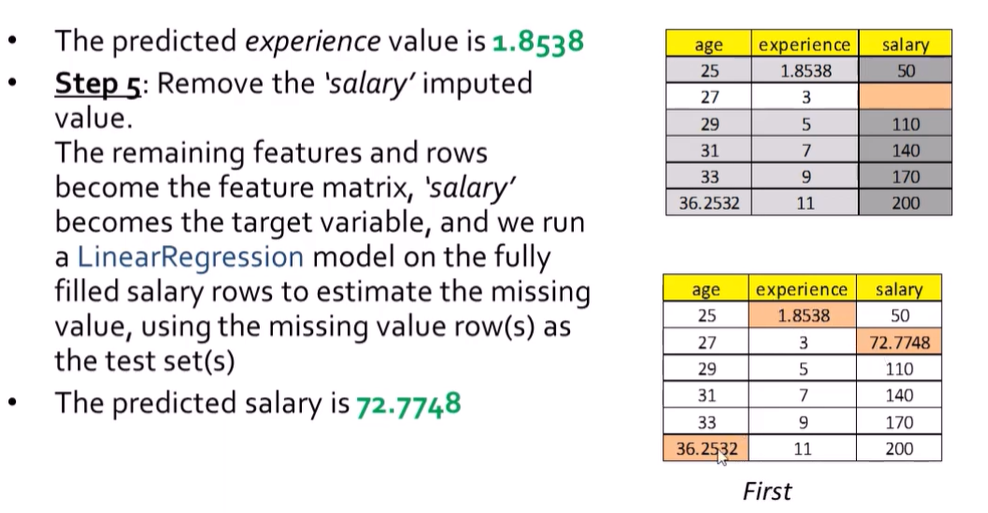
[**IterativeImputer**](https://scikit-learn.org/stable/modules/generated/sklearn.impute.IterativeImputer.html#sklearn.impute.IterativeImputer) class, which models each feature with missing values as a function of other features, and uses that estimate for imputation. It does so in an iterated round-robin fashion:

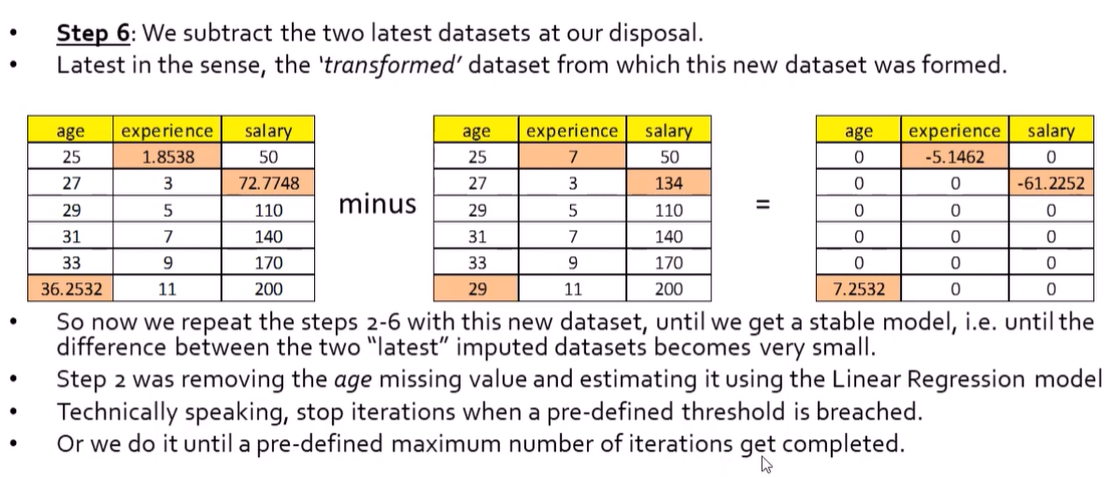
* At each step, a feature column is designated as output y and the other feature columns are treated as inputs X.
* A regressor is fit on (X, y) for known y.
* The regressor is used to predict the missing values of y.
* This is done for each feature in an iterative fashion, and then is repeated for max\_iter imputation rounds.
* The results of the final imputation round are returned.

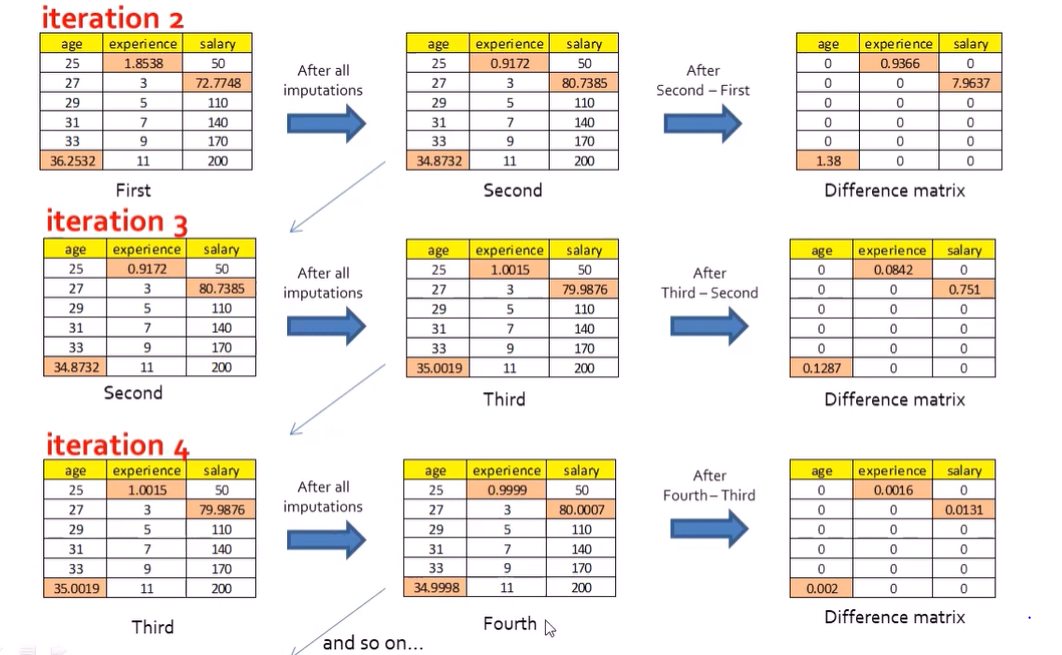
**Example:**











1. **Autoimpute**

A python package for analysis and implementation of Imputation Methods. Autoimpute is designed to be user friendly and flexible. When performing imputation, Autoimpute fits directly into scikit-learn machine learning projects. Imputers inherit from sklearn's BaseEstimator and TransformerMixin and implement fit and transform methods, making them valid Transformers in a sklearn pipeline.

from autoimpute.imputations import SingleImputer, MultipleImputer, MiceImputer

Which to use, and When?

* There are tradeoffs between the three imputers
* We won't get into the specifics regarding why, but here are a couple points to keep in mind:
* Execution time (best to worst): SingleImputer, MultipleImputer, MiceImputer
* Imputation quality (best to worst): MiceImputer, MultipleImputer, SingleImputer
* This shouldn't come as a surprise. The MiceImputer does the most work, while the SingleImputer does the least

**Reference:**

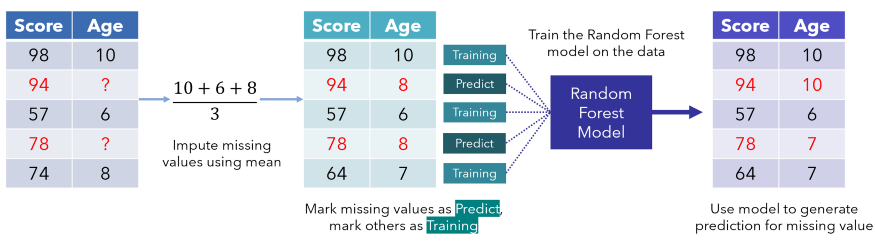
<https://pypi.org/project/autoimpute/>

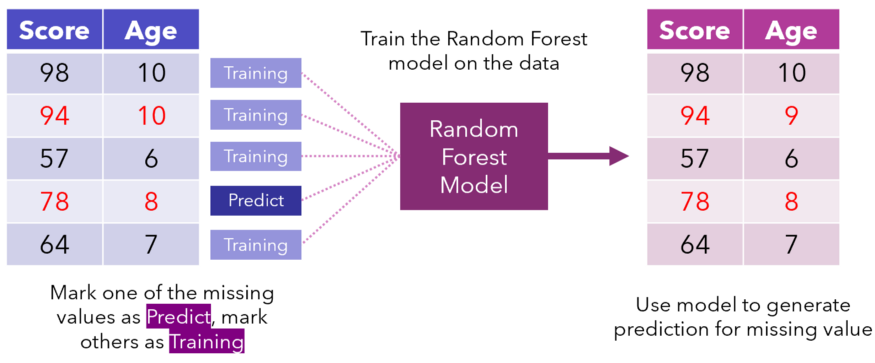
<https://towardsdatascience.com/getting-started-with-data-imputation-using-autoimpute-c3d53484a4bf>

1. **Missforest**

MissForest is a machine learning-based imputation technique. It uses a **Random Forest** algorithm to do the task. It is based on an **iterative approach**, and at each iteration the generated predictions are better.

First, the missing values are filled in using median/mode imputation. Then, we mark the missing values as ‘Predict’ and the others as training rows, which are fed into a Random Forest model trained to predict, in this case, Age based on Score. The generated prediction for that row is then filled in to produce a transformed dataset.





Iterations continue until some stopping criteria is met or after a certain number of iterations has elapsed. As a general rule, datasets become well imputed after four to five iterations, but it depends on the size and amount of missing data.

**Reference:**

<https://towardsdatascience.com/missforest-the-best-missing-data-imputation-algorithm-4d01182aed3>

<https://towardsdatascience.com/how-to-use-python-and-missforest-algorithm-to-impute-missing-data-ed45eb47cb9a>

<https://github.com/epsilon-machine/missingpy>

<https://pypi.org/project/missingpy/>

<https://github.com/HindyDS/MissForest/blob/main/missforest/miss_forest.py>

1. **Missforest extra**

[**https://pypi.org/project/MissForestExtra/**](https://pypi.org/project/MissForestExtra/)

***Notes:***

1.Missing Completely at Random (MCAR)

2. Missing at Random (MAR)

3- Missing NOT at Random (MNAR)

**MCAR** implies the reason for the missingness of a field is completely random, and that we probably can't predict that value from any other value in the data. Maybe the questionnaires failed to reach the respondent?

**MAR** implies that the missingness of a field can be explained by the values in other columns, but not from that column.

**MNAR** implies there WAS a reason why the respondent didn't fill up that field, and hence that data is NOT missing at random. For example, if someone is obese, they'd be less likely to disclose their weights.

If we have MNAR, we need to inspect why the data was missing, rather than straight away imputing them.