Handling missing values in EDA

Methods of handling missing values in exploratory data analysis

When I started learning data analysis everyone told me that handling missing values is the first step we do in exploratory data analysis and every time when I start a new dataset there used to be a new way of handling missing values, some by replacing it with mode simply, some using very complex methods

Here is a compilation of all the methods that I have used so far for handling missing values for column of any data type.

1. Replacement with Mean/Median/Mode
2. Random Sample Imputation
3. Capturing NAN with new feature
4. End of Distribution Imputation
5. Arbitrary Imputation
6. Frequent Category Imputation (only used with Categorical feature)
7. Make ‘NAN’ a new Category(only used with Categorical feature)

Before going through the description you should know what are the types of missing data.

There are three types of missing data

- **MCAR**: means Missing completely at random. The probability of a missing data value is independent of any observation in the data set means it has no relation with the data, it just miss at random places.
- **MNAR**: means Missing Not at Random. For example, when most of the missing people from work are sickest people, people with the lowest education are missing on education, this kind of missing is referred as Missing Not at Random (MNAR).
- **MAR**: means Missing at Random

Consider the following example.

- men are more likely to tell you their age than women
- men are more likely to tell you their weight than women.
- men are more likely to dropout in clinical trials than women.
- men are less likely to fill in a depression survey but this has nothing to do with their level of depression, after accounting for maleness.

familiar right? this is kind of missing is referred as missing at random. missing data at random(MAR) is more common than missing completely at random(MCAR) in all disciplines.

In this case, clearly the missing and observed observations are no longer coming from the same distribution and this is a crucial distinction between the two methods

- **Replacement with Mean/Median/Mode:**

When should we apply Mean/Median replacement-

a. When data is **MCAR**(Missing Completely at Random)

Process:

Find out the median of that column:

```
median=df.Age.median()
median
```

write a function for imputation which will take data, column_name, and value of mean/median/mode to replace it with

```
def impute_nan(df,variable,median):
    df[variable+"_median"]=df[variable].fillna(median)
```

output-

	Survived	Age	Fare	Age_median
0	0	22.0	7.2500	22.0
1	1	38.0	71.2833	38.0
2	1	26.0	7.9250	26.0
3	1	35.0	53.1000	35.0
4	0	35.0	8.0500	35.0

Advantages:

1. It is easy to implement
2. Median replacement is robust to outliers
3. faster way to get complete dataset

Disadvantages:

1. Change in variance (distortion in the original variance and standard deviation of data).After applying median the distribution becomes high in the middle(due to median replacement more valued are concentrated in middle) than the rest
2. It impacts correlation

- **Random Sample Imputation:** In case of Random Sampling we use Random observations from the dataset and it is then used for imputing the NAN values

When should we apply Random Sample Imputation:

When the data is MCAR (missing completely at Random)

Process:

Here I am replacing 'Age' column null values using random sampling . The index 423 has nan value and it got replaces with 28.00.

```
df['Age'].dropna().sample(df['Age'].isnull().sum(),random_state=0)
```

```
423    28.00
177    50.00
305     0.92
292    36.00
889    26.00
       ...
539    22.00
267    25.00
352    15.00
99     34.00
689    15.00
Name: Age, Length: 177, dtype: float64
```

```
def impute_nan(df,variable,median):
    df[variable+"_Random_sampling"]=df[variable].fillna(median)
    df[variable+ "_random"]=df[variable]
    random_sample=df[variable].dropna().sample(df[variable].isnull().sum(),random_state=0)
    ##pandas needs to have same index to merge

    random_sample_index=df[df[variable].isnull()].index

    df.loc[df[variable].isnull(),variable+ '_random']=random_sample
```

```
impute_nan(df,'Age',median)
```

```
df.head()
```

	Survived	Age	Fare	Age_Random_sampling	Age_random
0	0	22.0	7.2500	22.0	22.0
1	1	38.0	71.2833	38.0	38.0
2	1	26.0	7.9250	26.0	26.0
3	1	35.0	53.1000	35.0	35.0
4	0	35.0	8.0500	35.0	35.0

Command Prompt - jupyter notebook

Advantages:

a. less Variance distortion

Disadvantages:

a. Every situation randomness wont work

- **Capturing NAN with a new feature:**

Where should we apply this?

It works well with MNAR data(missing completely not at random)

Process:

It will make a new column ‘Age_nan’ having values as 0 and 1. 0 means that is does not contain null value and 1 means it contains null values.

```
df['Age_nan'] = np.where(df['Age'].isnull(),1,0)
```

	Survived	Age	Fare	Age_nan
0	0	22.0	7.2500	0
1	1	38.0	71.2833	0
2	1	26.0	7.9250	0
3	1	35.0	53.1000	0
4	0	35.0	8.0500	0

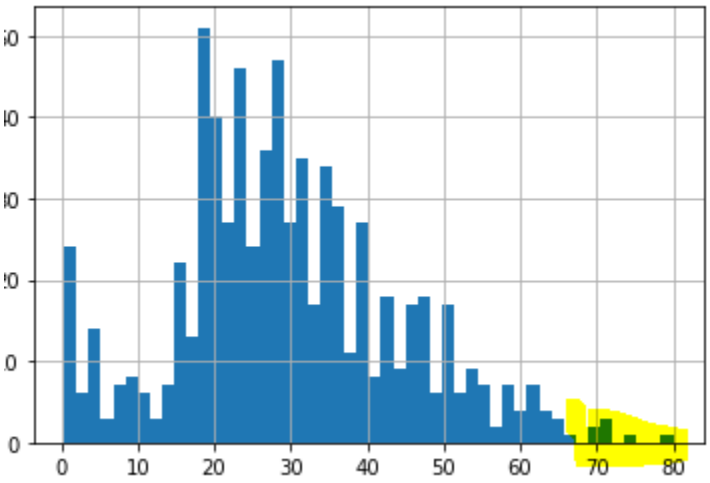
Advantages:

It captures the importance of missing values which will help model understand the data better

Disadvantages:

Curse of Dimensionality

- **End of Distribution Imputation**: In case of end of distribution imputation we basically takes a value from end of the distribution(After third standard deviation)and replace nan with that value . For example in this below distribution we take any one of the highlighted value and replace and replace nan with it



It is also used in case of missing completely at random(MCAR)

process:

```
extreme=df.Age.mean()+ 3*df.Age.std()
```

```
def impute_nan(df,variable,median,extreme):
    df[variable+"_end_distribution"]=df[variable].fillna(extreme)
    df[variable].fillna(median,inplace=True)
```

```
impute_nan(df,'Age',median,extreme)
```

```
df.head()
```

	Survived	Age	Fare	Age_end_distribution
0	0	22.0	7.2500	22.0
1	1	38.0	71.2833	38.0
2	1	26.0	7.9250	26.0
3	1	35.0	53.1000	35.0
4	0	35.0	8.0500	35.0

Advantages:

captures the importance of missing ness if there is any

disadvantages:

- Distorts the original distribution of data
- If NA is big, it will mask outliers
- If NA is small, the replaced NA will be considered as outlier
- Arbitrary value imputation: It means imputing missing values with an arbitrary value

What is an Arbitrary value?

1. It should not be more frequently present

It is not heavily used

process:

```
def impute_nan(df,variable):
    df[variable+"_zero"]=df[variable].fillna('0')
```

Advantages: it is easy to implement

Disadvantages:

1. it distorts the original variance of dataset
2. hard to decide which value to use as an arbitrary value

- **Frequency Category Imputation:** It is used for categorical feature imputation . It means replacing NAN value with most frequently occurring value

Process:

```
#Replacing Function
def impute_nan(df,variable):
    most_frequent_category= df[variable].value_counts().index[0]
    df[variable]=df[variable].fillna(most_frequent_category,inplace=True)
```

```
for feature in ['BsmtQual','FireplaceQu','GarageType']:
    impute_nan(df,feature)
```

Advantages:

It is fast and easy

Disadvantages:

a. if there are many nans the value of most frequent labels may increase more

b.it distorts the relation of the most frequent label

- Make ‘NAN’ a new Category

This is used in categorical columns when we have more frequent missing values we create a separate label for those NAN values. Example in below we have replaced NAN with word “Missing”

```
def impute_nan(df,variable):
    df[variable+"new_var"]=np.where(df[variable].isnull(),"Missing",df[variable])
```