Handling Missing Values in Machine Learning: Part 1

Georgios Drakos Follow

Jul 27, 2018 · 5 min read ★

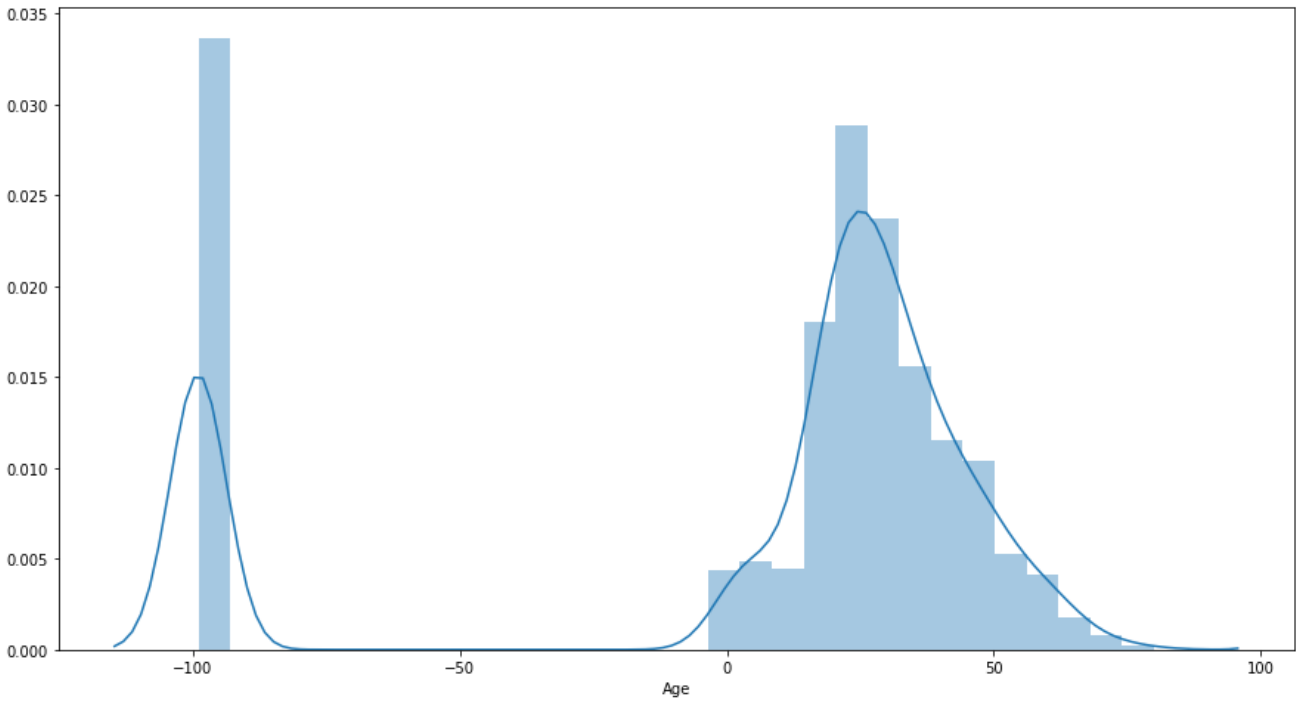
Data in real world are rarely clean and homogeneous. Typically, they tend to be incomplete, noisy, and inconsistent and it is an important task of a Data scientist to prepossess the data by filling missing values. It is important to be handled as they could lead to wrong prediction or classification for any given model being used. The goal of this article is to cover the basic techniques for handling missing values in a dataset.

Find missing values in the dataset

Missing values could be: NaN, empty string, ?,-1,-99,-999 and so on. In order to understand if -1 is a missing value or not we could draw a histogram. If this variable has a uniform distribution between 0 and 1 and it has a small peak at -1 then -1 is actually a missing value.

Missing values can be hidden from us and by hidden mean replaced by some other value beside NaN. Therefore, it is always beneficial to plot a **histogram** in order to identify those values.

```
# Plot histogram using seaborn
import seaborn as sns
plt.figure(figsize=(15,8))
sns.distplot(dataframe.column_name, bins =30)
```



Identify missing value in a dataset

Typically, I run the following commands to 'get' a felling of the missing values of my dataset. The commands applied for the titanic dataset.

```
# Load Data
train = pd.read_csv('data/train.csv')
test = pd.read_csv('data/test.csv')
# Concatenate train & test
train_objs_num = len(train)
y = train['Survived']
dataset = pd.concat(objs=[train.drop(columns=['Survived']), test],
axis=0)
```

In case we have train and test dataset it is always better to concatenate both dataset to figure out which features has missing values.

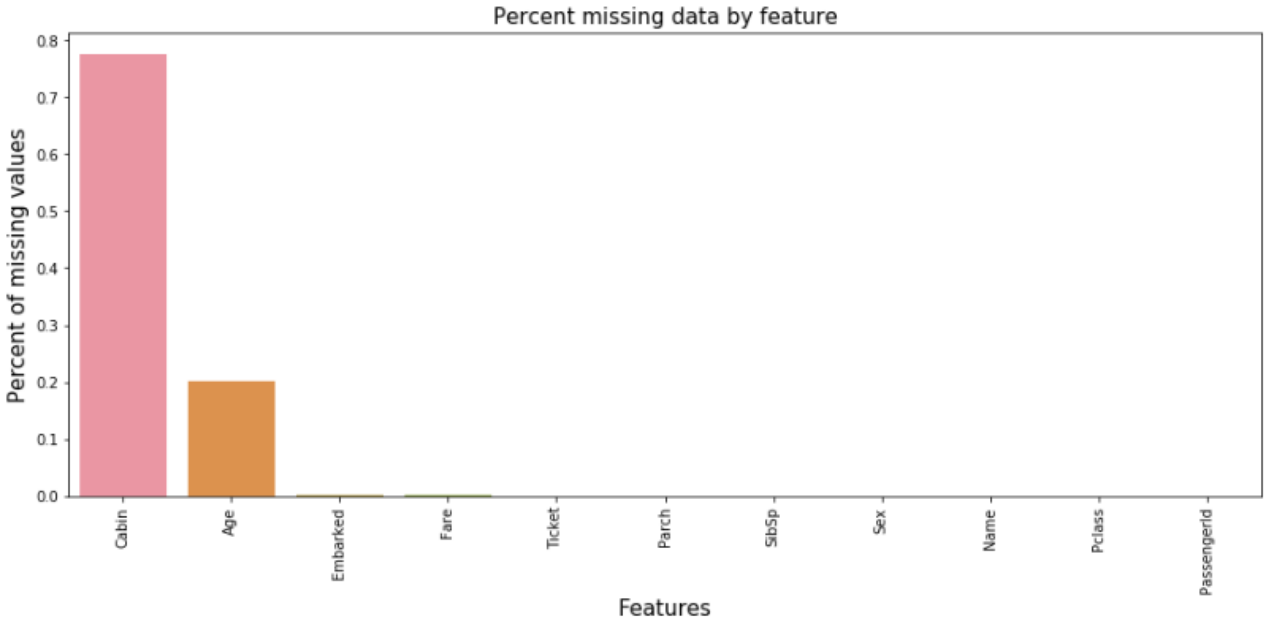
```
dataset.info()

<class 'pandas.core.frame.DataFrame'>
Int64Index: 1309 entries, 0 to 417
Data columns (total 11 columns):
PassengerId    1309 non-null int64
Pclass         1309 non-null int64
Name           1309 non-null object
Sex            1309 non-null object
Age            1046 non-null float64
SibSp          1309 non-null int64
Parch          1309 non-null int64
Ticket         1309 non-null object
Fare           1308 non-null float64
Cabin          295 non-null object
Embarked       1307 non-null object
dtypes: float64(2), int64(4), object(5)
memory usage: 122.7+ KB
```

In case the number of features is small we can easily understand that features ‘Age’, ‘Fare’, ‘Cabin’ and ‘Embarked’ contain missing values.

```
total = dataset.isnull().sum().sort_values(ascending=False)
percent =
(dataset.isnull().sum()/dataset.isnull().count()).sort_values(ascendi
ng=False)
missing_data = pd.concat([total, percent], axis=1, keys=['Total',
'Percent'])
f, ax = plt.subplots(figsize=(15, 6))
plt.xticks(rotation='90')
sns.barplot(x=missing_data.index, y=missing_data['Percent'])
plt.xlabel('Features', fontsize=15)
```

```
plt.ylabel('Percent of missing values', fontsize=15)
plt.title('Percent missing data by feature', fontsize=15)
missing_data.head()
```

	Total	Percent
Cabin	1014	0.774637
Age	263	0.200917
Embarked	2	0.001528
Fare	1	0.000764
Ticket	0	0.000000



Basic missing values importation

1. *Ignore the data row*

This is a quick solution and typically is preferred in cases where the percentage of missing values is relatively low (<5%). It is a dirty approach as you loose data. Imagine you drop one whole observation just because one of the features had a missing value, even if the rest of the features are perfectly filled and informative!

```
# Will drop all rows that have any missing values.
dataframe.dropna(inplace=True)
```

You can also select to drop the rows only if all of the values in the row are missing.

```
dataframe.dropna(how='all',inplace=True)
```

Sometimes, you may just want to drop a column (variable) that has some missing values.

```
dataframe.dropna(axis=1,inplace=True)
```

Finally, you may want to keep only the rows with at least 4 non-na values:

```
dataframe.dropna(thresh=4,inplace=True)
```

Even if it's a small percentage of the dataset doesn't mean you should drop it. This method is the least preferable out of all methods illustrated in this article.

In my opinion, it is always better to keep data than to delete them. The only case that it may worth deleting a variable is when its missing values are more than 60% of the observations but only if that variable is insignificant. Taking this into consideration, imputation is always a preferred choice over deleting variables.

2. Back-fill or forward-fill to propagate next or previous values respectively:

```
#for back fill
dataframe.fillna(method='bfill',inplace=True)

#for forward-fill
dataframe.fillna(method='ffill',inplace=True)
```

Note that the NaN value will remain even after forward filling or back filling if a next or previous value isn't available or it is also a NaN value.

3. Replace with some constant value outside fixed value range-999,-1 etc

This method is useful as it gives the possibility to group missing values as a separate category represented by a constant value. It is a preferred option when it doesn't make

sense to try and predict a missing value. The downside is that performance of linear models can suffer.Use a global constant to fill in for missing values.

For example, in the titanic dataset filling in the missing value of the Embarked feature with the most common Port of Embarkation might not really makes sense as opposed to using something like “N/A”.

```
dataframe.Column_Name.fillna(-99,inplace=True)
```

4. Replace with mean, median value

This simple imputation method is based on treating every variable individually, ignoring any interrelationships with other variables.This method is beneficial for simple linear models and NN. But for tree based methods it can be harder for the algorithm to understand that there was a missing value.

MEAN: Suitable for continuous data without outliers

```
dataframe.Column_Name.fillna(dataframe.Column_Name.mean(),inplace=Tru
e)
```

MEDIAN :Suitable for continuous data with outliers

```
dataframe.Column_Name.fillna(dataframe.Column_Name.median(),inplace=T
rue)
```

Another option would be to randomly fill them with values close to the mean value but within one standard deviation.

```
Column_Name_avg = dataframe['Column_Name'].mean()
Column_Name_std = dataframe['Column_Name'].std()
Column_Name_null_count = dataframe['Column_Name'].isnull().sum()
Column_Name_null_random_list = np.random.randint(Column_Name_avg -
Column_Name_std, Column_Name_avg + Column_Name_std,
```

```
size=Column_Name_null_count)
dataframe['Column_Name'][np.isnan(dataframe['Column_Name'])] =
Column_Name_null_random_list
dataframe['Column_Name'] = dataframe['Column_Name'].astype(int)
```

For categorical feature you can select to fill in the missing values with the most common value as illustrated below.

```
dataframe.Column_Name.fillna(dataframe.Column_Name.mode()[0],
inplace=True)
```

Please note that Mean, Median and Mode imputation diminishes any correlations involving the variable(s) that are imputed. This is because we assume that there is no relationship between the imputed variable and any other measured variables. Thus, those imputations have some attractive properties for univariate analysis but become problematic for multivariate analysis.

5. Isnull feature

Adding new feature *isnull* indicating which rows have missing values for this feature. By doing this the tree based methods it can now understand that there was a missing value. The downside is that we double the number of features.

```
dataframe['is_null_Column_Name'] =
dataframe.Column_Name.isnull().astype(int)
```

Conclusion

The take-home message is:

1. the approach to deal with missing values is heavily dependent on the nature of the dataset
2. It is always useful to ask, why are there missing values?

3. Different ways of handling missing data have difference effects on the performance of our models.

Thank you for reading and I am looking forward to hear your questions.

P.S: Watch out for my next article that discusses more advance methods for handling missing values.

Happy coding :)

Data Science Machine Learning Python Missing Values Programming