HAFEEZ JIMOH Follow
Tech. Enthusiast , Aspiring Machine Learning Engineer
Jan 6 · 7 min read

# The tale of missing values in Python

Imagine buying a chocolate box with 60 chocolate samples where there are 15 different unique shapes of chocolates. Unfortunately, on opening the chocolate box, you find two empty segments of chocolate. Can you accurately find a way out off handling the missing chocolate segments. Should one just pretend as if the missing chocolate isn't missing.? Should one return the chocolate box to the seller? Should one go and buy two other chocolates to fill the missing portion. Or can one just predict the shape of the missing chocolate based on previous experience of arrangement and shapes of chocolate in the box and then buy a chocolate of such predicted shape.

http://www.alphabetastats.com

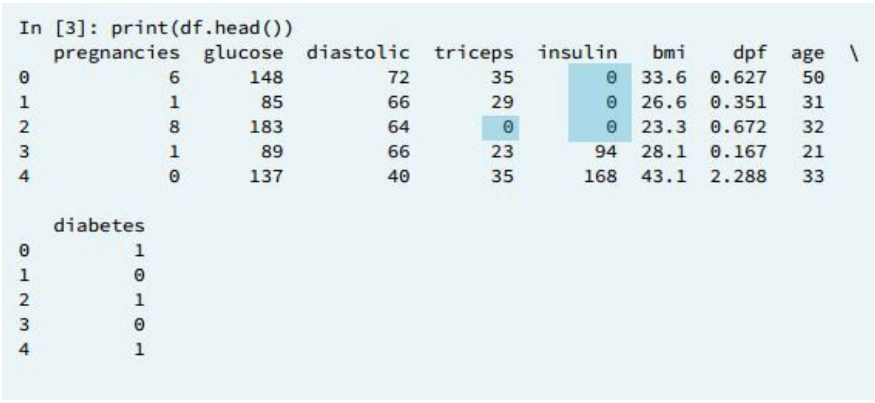
The above and some others are mind throbbing questions a data scientist need to answer in order to handle missing data correctly. Hence, this write-up aims to elucidate on several approaches available for handling missing values in our data exploration journey.

Data in real world are rarely clean and homogeneous. Data can either be missing during data extraction or collection. Missing values need to be handled because they reduce the quality for any of our performance metric. It can also lead to wrong prediction or classification and can also cause a high bias for any given model being used.

Depending on data sources, missing data are identified differently. Pandas always identify missing values as NaN. However, unless the data has been pre-processed to a degree that an analyst will encounter missing values as NaN. Missing values can appear as a question mark (?) or a zero (0) or minus one (-1) or a blank. As a result, it is always important that a data scientist always perform exploratory data analysis(EDA) first before writing any machine learning algorithm. EDA is simply a litmus for understanding and knowing the behaviour of our data.

> *Exploratory data analysis can never be the whole story, but nothing else can serve as the foundation stone.—John Tukey*

For example, if we have a data set that should be used to to predict average salary based on years of experience and in our years of experience column, a value of -1 is indiscriminately found, then we can flag such value as a missing value. Else, it could be that we have a continuous variable feature (observed or independent variable) of height/weight/age and our EDA shows us a value of 0 or -1 or values less than 1 for some observation; then one can conclude that such value is a missing value. Missing values could also be blank. This will usually occur when there is no observed measurement for such feature either by respondents or instruments used for capturing such data.



```
In [3]: print(df.head())
   pregnancies  glucose  diastolic  triceps  insulin   bmi    dpf  age  \
0            6      148         72       35        0  33.6  0.627   50
1            1       85         66       29        0  26.6  0.351   31
2            8      183         64        0        0  23.3  0.672   32
3            1       89         66       23       94  28.1  0.167   21
4            0      137         40       35      168  43.1  2.288   33

   diabetes
0         1
1         0
2         1
3         0
4         1
```

Source: DataCamp Supervised Learning Course

## WHAT DO WE DO TO MISSING VALUES

There are several options for handling missing values each with its own PROS and CONS. However, the choice of what should be done is largely dependent on the nature of our data and the missing values. Below is a summary highlight of several options we have for handling missing values.

1. DROP MISSING VALUES
2. FILL MISSING VALUES WITH TEST STATISTIC
3. PREDICT MISSING VALUE WITH A MACHINE LEARNING ALGORITHM

Below is a few list of commands to detect missing values with EDA.

```
data_name.info()
```

This will tell us the total number of non null observations present including the total number of entries. Once number of entries isn't equal to number of non null observations, we can begin to suspect missing values.

```
data_name.describe()
```

This will display a summary statistics of all observed features and labels. The most important to note here is the min value. Once we see

-1/0 in an observation like age/height/weight, then we have been able to detect missing value.

```
data_name.head(x)
```

This will output the first x rows of our data. Viewing this will give one a quick view on the presence of NaN/-1/0/blank/? among others.

```
data_name.isnull().sum()
```

This will tell us the total number of NaN in or data.

If the missing value isn't identified as NaN , then we have to first convert or replace such non NaN entry with a NaN.

```
data_name['column_name'].replace(0, np.nan, inplace= True)
```

This will replace values of zero with NaN in the column named column_name of our data_name .

## 1) DROPPING NULL OR MISSING VALUES

This is the fastest and easiest step to handle missing values. However, it is not generally advised. This method reduces the quality of our model as it reduces sample size because it works by deleting all other observations where any of the variable is missing. The process can be done by:

```
data_name.dropna()
```

The code snippet shows the danger of dropping missing values.

```
In [2]: train.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 12 columns):
PassengerId    891 non-null int64
Survived       891 non-null int64
Pclass         891 non-null int64
Name           891 non-null object
Sex            891 non-null object
Age            714 non-null float64
SibSp          891 non-null int64
Parch          891 non-null int64
Ticket         891 non-null object
Fare           891 non-null float64
Cabin          204 non-null object
Embarked       889 non-null object
dtypes: float64(2), int64(5), object(5)
memory usage: 83.6+ KB
```

```
In [3]: df=train.dropna()

In [4]: df.info()
<class 'pandas.core.frame.DataFrame'>
Int64Index: 183 entries, 1 to 889
Data columns (total 12 columns):
PassengerId    183 non-null int64
Survived       183 non-null int64
Pclass         183 non-null int64
Name           183 non-null object
Sex            183 non-null object
Age            183 non-null float64
SibSp          183 non-null int64
Parch          183 non-null int64
Ticket         183 non-null object
Fare           183 non-null float64
Cabin          183 non-null object
Embarked       183 non-null object
dtypes: float64(2), int64(5), object(5)
memory usage: 18.6+ KB
```

It will be observed that of 891 entries will be reduced to 183 just by dropping NaN values.!!! Dropping is only advised to be used if missing values are few (say 0.01–0.5% of our data). Percent is just a rule of thumb.

## 2) FILLING MISSING VALUES

This is the most common method of handling missing values. This is a process whereby missing values are replaced with a test statistic like mean, median or mode of the particular feature the missing value belongs to. One can also specify a forward-fill or back-fill to propagate the next values backward or previous value forward.

*Filling missing values with a test statistic*

```
#Age is a column name for our train data

mean_value=train['Age'].mean()
train['Age']=train['Age'].fillna(mean_value)
```

```
#this will replace all NaN values with the mean of the non
null values


#For Median


meadian_value=train['Age'].median()
train['Age']=train['Age'].fillna(median_value)
```

*Alternative way of filling missing value with test statistic is by using our Imputer method found in sklearn.preprocessing.*

```
In [1]: from sklearn.preprocessing import Imputer
In [2]: imp = Imputer(missing_values='NaN', strategy='mean',
axis=0)
In [3]: imp.fit(train)
In [4]: train= imp.transform(train)
#This will look for all columns where we have NaN value and
replace the NaN value with specified test statistic.


#for mode we specify strategy='most_frequent'
```

*For Back-fill or forward-fill to propagate next or previous values respectively:*

```
#for back fill


train.fillna(method='bfill')
#for forward-fill


train.fillna(method=''ffill)


#one can also specify an axis to propagate (1 is for rows
and 0 is for columns)


train.fillna(method='bfill', axis=1)
```

Please note that if a previous or next value isn't available or rather if it is also a NaN value, then, the NaN remains even after back-filling or forward-filling.

Also, the disadvantage of using mean is that the mean is greatly affected by outliers in our data. As a result, if outliers are present in our

data, then median will be the best out of the box tool to use.

### Imputing test statistic within a Pipeline

Data pipelines allow one to transform data from one representation to another through a series of steps. Pipelines allow one to apply and chain intermediate steps of transform to our data. For example, one can fill missing values, pass the output to cross validation and grid search and then fit the model in series of steps chained together where the output of one is the input to another.

You can learn more about pipelines here.

This is an example of a pipeline that imputes data with most frequent value of each column, and then fit to a logistic regression.

```
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import Imputer
imp = Imputer(missing_values='NaN', strategy='mean', axis=0)
logreg = LogisticRegression()
steps = [('imputation', imp),('logistic_regression',
logreg)]
pipeline = Pipeline(steps)
X_train, X_test, y_train, y_test = train_test_split(X, y,
                                    test_size=0.3,
random_state=42)
pipeline.fit(X_train, y_train)
y_pred = pipeline.predict(X_test)
pipeline.score(X_test, y_test)
```

## 3) PREDICTIVE MODEL FOR HANDLING MISSING DATA

This is by far one of the best and most efficient method for handling missing data. Depending on the class of data that is missing, one can either use a regression model or classification to predict missing data. This works by turning missing features to labels themselves and now using columns without missing values to predict columns with missing values

The process goes thus:

*Call the variable where you have missing values as y.*

*Split data into sets with missing values and without missing values, name the missing set X_text and the one without missing values X_train and take y (variable or feature where there is missing values) off the second set, naming it y_train.*

*Use one of classification methods to predict y_pred.*

*Add it to X_test as your y_test column. Then combine sets together.*

For a beginner or newbie in machine learning, this approach might seem more difficult. The only drawback to this approach is that if there is no correlation between attributes with missing data and other attributes in the data set, then the model will be bias for predicting missing values.

## WRAP UP WITH ASSERT

Apart from using isnull() to check for missing values as done above, one can use assert to programmatically check that no missing or unexpected '0' value is present. This gives confidence that code is running properly.

The following and some other boolean operations can be carried out on assert. Assert will return nothing is the assert statement is true and will return an AssertionError if statement is false.

```
#fill null values with 0

df=train.fillna(value=0)

#assert that there are no missing values

assert pd.notnull(df).all().all()

#or for a particular column in df

assert df.column_name.notall().all()

#assert all values are greater than 0

assert (df >=0).all().all()

#assert no entry in a column is equal to 0

assert (df['column_name']!=0).all().all()
```

In conclusion, the approach to deal with missing values is heavily dependent on the nature of such data. Therefore, the more different

types of data you work with the better the experience you have with different solutions for different data classes.

Thank you for reading and please do drop your comments and do not forget to share.

P.S: Watch out for my next article that discusses various alternatives for handling categorical variable

- *all images are from web**