How to Handle Missing Data with Python

by **Jason Brownlee** on March 20, 2017 in **Data Preparation**

I think with our data ... we already have a structure that deals w/ missing values such as masked array

Last Updated on August 28, 2020

Real-world data often has **missing values**.

Data can have missing values for a number of reasons such as observations that were not recorded and data corruption.

Handling missing data is important as many machine learning algorithms do not support data with missing values.

In this tutorial, you will discover how to handle missing data for machine learning with Python.

Specifically, after completing this tutorial you will know:

- How to marking invalid or corrupt values as missing in your dataset.
- How to remove rows with missing data from your dataset.
- How to impute missing values with mean values in your dataset.

Kick-start your project with my new book Data Preparation for Machine Learning, including *step-by-step tutorials* and the *Python source code* files for all examples.

Let's get started.

Note: The examples in this post assume that you have Python 3 with Pandas, NumPy and Scikit-Learn installed, specifically scikit-learn version 0.22 or higher. If you need help setting up your environment see this tutorial.

- **Update Mar/2018**: Changed link to dataset files.
- **Update Dec/2019**: Updated link to dataset to GitHub version.
- **Update May/2020**: Updated code examples for API changes. Added references.

How to Handle Missing Values with Python
Photo by CoCreatr, some rights reserved.

Overview

This tutorial is divided into 6 parts:

1. **Diabetes Dataset**: where we look at a dataset that has known missing values.
2. **Mark Missing Values**: where we learn how to mark missing values in a dataset.
3. **Missing Values Causes Problems**: where we see how a machine learning algorithm can fail when it contains missing values.
4. **Remove Rows With Missing Values**: where we see how to remove rows that contain missing values.
5. **Impute Missing Values**: where we replace missing values with sensible values.
6. **Algorithms that Support Missing Values**: where we learn about algorithms that support missing

values.

First, let's take a look at our sample dataset with missing values.

1. Diabetes Dataset

The Diabetes Dataset involves predicting the onset of diabetes within 5 years in given medical details.

- Dataset File.
- Dataset Details

It is a binary (2-class) classification problem. The number of observations for each class is not balanced. There are 768 observations with 8 input variables and 1 output variable. The variable names are as follows:

- 0. Number of times pregnant.
- 1. Plasma glucose concentration a 2 hours in an oral glucose tolerance test.
- 2. Diastolic blood pressure (mm Hg).
- 3. Triceps skinfold thickness (mm).
- 4. 2-Hour serum insulin (mu U/ml).
- 5. Body mass index (weight in kg/(height in m)^2).
- 6. Diabetes pedigree function.
- 7. Age (years).
- 8. Class variable (0 or 1).

The baseline performance of predicting the most prevalent class is a classification accuracy of approximately 65%. Top results achieve a classification accuracy of approximately 77%.

A sample of the first 5 rows is listed below.

```
6,148,72,35,0,33.6,0.627,50,1
1,85,66,29,0,26.6,0.351,31,0
8,183,64,0,0,23.3,0.672,32,1
1,89,66,23,94,28.1,0.167,21,0
0,137,40,35,168,43.1,2.288,33,1
...
```

This dataset is known to have missing values.

Specifically, there are missing observations for some columns that are marked as a zero value.

We can corroborate this by the definition of those columns and the domain knowledge that a zero value is invalid for those measures, e.g. a zero for body mass index or blood pressure is invalid.

Download the dataset from here and save it to your current working directory with the file name *pima-indians-diabetes.csv* .

- pima-indians-diabetes.csv

2. Mark Missing Values

Most data has missing values, and the likelihood of having missing values increases with the size of the dataset.

> *Missing data are not rare in real data sets. In fact, the chance that at least one data point is missing increases as the data set size increases.*

— Page 187, Feature Engineering and Selection, 2019.

In this section, we will look at how we can identify and mark values as missing.

We can use plots and summary statistics to help identify missing or corrupt data.

We can load the dataset as a Pandas DataFrame and print summary statistics on each attribute.

```
# load and summarize the dataset
from pandas import read_csv
# load the dataset
dataset = read_csv('pima-indians-diabetes.csv', header=None)
# summarize the dataset
print(dataset.describe())
```

Running this example produces the following output:

```
                0           1           2  ...           6           7           8
count  768.000000  768.000000  768.000000  ...  768.000000  768.000000  768.000000
mean     3.845052  120.894531   69.105469  ...    0.471876   33.240885    0.348958
std      3.369578   31.972618   19.355807  ...    0.331329   11.760232    0.476951
min      0.000000    0.000000    0.000000  ...    0.078000   21.000000    0.000000
25%      1.000000   99.000000   62.000000  ...    0.243750   24.000000    0.000000
50%      3.000000  117.000000   72.000000  ...    0.372500   29.000000    0.000000
75%      6.000000  140.250000   80.000000  ...    0.626250   41.000000    1.000000
max     17.000000  199.000000  122.000000  ...    2.420000   81.000000    1.000000

[8 rows x 9 columns]
```

This is useful.

We can see that there are columns that have a minimum value of zero (0). On some columns, a value of zero does not make sense and indicates an invalid or missing value.

> *Missing values are frequently indicated by out-of-range entries; perhaps a negative number (e.g., -1) in a numeric field that is normally only positive, or a 0 in a numeric field that can never normally be 0.*

— Page 62, Data Mining: Practical Machine Learning Tools and Techniques, 2016.

Specifically, the following columns have an invalid zero minimum value:

- 1: Plasma glucose concentration
- 2: Diastolic blood pressure
- 3: Triceps skinfold thickness
- 4: 2-Hour serum insulin
- 5: Body mass index

Let's confirm this my looking at the raw data, the example prints the first 20 rows of data.

```
# load the dataset and review rows
from pandas import read_csv
# load the dataset
dataset = read_csv('pima-indians-diabetes.csv', header=None)
# print the first 20 rows of data
print(dataset.head(20))
```

Running the example, we can clearly see 0 values in the columns 2, 3, 4, and 5.

```
        0    1   2   3    4     5      6   7  8
0       6  148  72  35    0  33.6  0.627  50  1
1       1   85  66  29    0  26.6  0.351  31  0
2       8  183  64   0    0  23.3  0.672  32  1
3       1   89  66  23   94  28.1  0.167  21  0
4       0  137  40  35  168  43.1  2.288  33  1
5       5  116  74   0    0  25.6  0.201  30  0
6       3   78  50  32   88  31.0  0.248  26  1
7      10  115   0   0    0  35.3  0.134  29  0
8       2  197  70  45  543  30.5  0.158  53  1
9       8  125  96   0    0   0.0  0.232  54  1
10      4  110  92   0    0  37.6  0.191  30  0
11     10  168  74   0    0  38.0  0.537  34  1
12     10  139  80   0    0  27.1  1.441  57  0
13      1  189  60  23  846  30.1  0.398  59  1
14      5  166  72  19  175  25.8  0.587  51  1
15      7  100   0   0    0  30.0  0.484  32  1
16      0  118  84  47  230  45.8  0.551  31  1
17      7  107  74   0    0  29.6  0.254  31  1
18      1  103  30  38   83  43.3  0.183  33  0
19      1  115  70  30   96  34.6  0.529  32  1
```

We can get a count of the number of missing values on each of these columns. We can do this my marking all of the values in the subset of the DataFrame we are interested in that have zero values as True. We can then count the number of true values in each column.

We can do this my marking all of the values in the subset of the DataFrame we are interested in that have zero values as True. We can then count the number of true values in each column.

zero values as True. We can then count the number of true values in each column.

```
# example of summarizing the number of missing values for each variable
from pandas import read_csv
# load the dataset
dataset = read_csv('pima-indians-diabetes.csv', header=None)
# count the number of missing values for each column
num_missing = (dataset[[1,2,3,4,5]] == 0).sum()
# report the results
print(num_missing)
```

Running the example prints the following output:

```
1      5
2     35
3    227
4    374
5     11
```

We can see that columns 1,2 and 5 have just a few zero values, whereas columns 3 and 4 show a lot more, nearly half of the rows.

This highlights that different “missing value” strategies may be needed for different columns, e.g. to ensure that there are still a sufficient number of records left to train a predictive model.

> *When a predictor is discrete in nature, missingness can be directly encoded into the predictor as if it were a naturally occurring category.*

— Page 197, Feature Engineering and Selection, 2019.

In Python, specifically Pandas, NumPy and Scikit-Learn, we mark missing values as NaN.

Values with a NaN value are ignored from operations like sum, count, etc.

We can mark values as NaN easily with the Pandas DataFrame by using the replace() function on a subset of the columns we are interested in.

After we have marked the missing values, we can use the isnull() function to mark all of the NaN values in the dataset as True and get a count of the missing values for each column.

```
# example of marking missing values with nan values
from numpy import nan
from pandas import read_csv
# load the dataset
dataset = read_csv('pima-indians-diabetes.csv', header=None)
# replace '0' values with 'nan'
dataset[[1,2,3,4,5]] = dataset[[1,2,3,4,5]].replace(0, nan)
# count the number of nan values in each column
print(dataset.isnull().sum())
```

Running the example prints the number of missing values in each column. We can see that the columns 1:5 have the same number of missing values as zero values identified above. This is a sign that we have marked the identified missing values correctly.

We can see that the columns 1 to 5 have the same number of missing values as zero values identified above. This is a sign that we have marked the identified missing values correctly.

```
0      0
1      5
2     35
3    227
4    374
5     11
6      0
7      0
8      0

dtype: int64
```

This is a useful summary. I always like to look at the actual data though, to confirm that I have not fooled myself.

Below is the same example, except we print the first 20 rows of data.

```
# example of review rows from the dataset with missing values marked
from numpy import nan
from pandas import read_csv
# load the dataset
dataset = read_csv('pima-indians-diabetes.csv', header=None)
# replace '0' values with 'nan'
dataset[[1,2,3,4,5]] = dataset[[1,2,3,4,5]].replace(0, nan)
# print the first 20 rows of data
print(dataset.head(20))
```

Running the example, we can clearly see NaN values in the columns 2, 3, 4 and 5. There are only 5 missing values in column 1, so it is not surprising we did not see an example in the first 20 rows.

It is clear from the raw data that marking the missing values had the intended effect.

```
     0      1     2     3      4     5      6   7  8
0    6  148.0  72.0  35.0    NaN  33.6  0.627  50  1
1    1   85.0  66.0  29.0    NaN  26.6  0.351  31  0
2    8  183.0  64.0   NaN    NaN  23.3  0.672  32  1
3    1   89.0  66.0  23.0   94.0  28.1  0.167  21  0
4    0  137.0  40.0  35.0  168.0  43.1  2.288  33  1
5    5  116.0  74.0   NaN    NaN  25.6  0.201  30  0
6    3   78.0  50.0  32.0   88.0  31.0  0.248  26  1
7   10  115.0   NaN   NaN    NaN  35.3  0.134  29  0
8    2  197.0  70.0  45.0  543.0  30.5  0.158  53  1
9    8  125.0  96.0   NaN    NaN   NaN  0.232  54  1
10   4  110.0  92.0   NaN    NaN  37.6  0.191  30  0
11  10  168.0  74.0   NaN    NaN  38.0  0.537  34  1
12  10  139.0  80.0   NaN    NaN  27.1  1.441  57  0
13   1  189.0  60.0  23.0  846.0  30.1  0.398  59  1
14   5  166.0  72.0  19.0  175.0  25.8  0.587  51  1
15   7  100.0   NaN   NaN    NaN  30.0  0.484  32  1
16   0  118.0  84.0  47.0  230.0  45.8  0.551  31  1
17   7  107.0  74.0   NaN    NaN  29.6  0.254  31  1
18   1  103.0  30.0  38.0   83.0  43.3  0.183  33  0
19   1  115.0  70.0  30.0   96.0  34.6  0.529  32  1
```

Before we look at handling missing values, let's first demonstrate that having missing values in a dataset can cause problems.

3. Missing Values Causes Problems

3. Missing Values Causes Problems

Having missing values in a dataset can cause errors with some machine learning algorithms.

> *Missing values are common occurrences in data. Unfortunately, most predictive modeling techniques cannot handle any missing values. Therefore, this problem must be addressed prior to modeling.*

— Page 203, Feature Engineering and Selection, 2019.

In this section, we will try to evaluate a the Linear Discriminant Analysis (LDA) algorithm on the dataset with missing values.

This is an algorithm that does not work when there are missing values in the dataset.

The below example marks the missing values in the dataset, as we did in the previous section, then attempts to evaluate LDA using 3-fold cross validation and print the mean accuracy.

```
# example where missing values cause errors
from numpy import nan
from pandas import read_csv
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
# load the dataset
dataset = read_csv('pima-indians-diabetes.csv', header=None)
# replace '0' values with 'nan'
dataset[[1,2,3,4,5]] = dataset[[1,2,3,4,5]].replace(0, nan)
# split dataset into inputs and outputs
values = dataset.values
X = values[:,0:8]
y = values[:,8]
# define the model
model = LinearDiscriminantAnalysis()
# define the model evaluation procedure
cv = KFold(n_splits=3, shuffle=True, random_state=1)
# evaluate the model
result = cross_val_score(model, X, y, cv=cv, scoring='accuracy')
# report the mean performance
print('Accuracy: %.3f' % result.mean())
```

Running the example results in an error, as follows:

```
ValueError: Input contains NaN, infinity or a value too large for dtype('float64').
```

This is as we expect.

We are prevented from evaluating an LDA algorithm (and other algorithms) on the dataset with missing values.

> *Many popular predictive models such as support vector machines, the glmnet, and neural networks, cannot tolerate any amount of missing values.*

— Page 195, Feature Engineering and Selection, 2019.

Now, we can look at methods to handle the missing values.

4. Remove Rows With Missing Values

The simplest strategy for handling missing data is to remove records that contain a missing value.

> *The simplest approach for dealing with missing values is to remove entire predictor(s) and/or sample(s) that contain missing values.*

— Page 196, Feature Engineering and Selection, 2019.

We can do this by creating a new Pandas DataFrame with the rows containing missing values removed.

Pandas provides the dropna() function that can be used to drop either columns or rows with missing data. We can use dropna() to remove all rows with missing data, as follows:

```
# example of removing rows that contain missing values
from numpy import nan
from pandas import read_csv
# load the dataset
dataset = read_csv('pima-indians-diabetes.csv', header=None)
# summarize the shape of the raw data
print(dataset.shape)
# replace '0' values with 'nan'
dataset[[1,2,3,4,5]] = dataset[[1,2,3,4,5]].replace(0, nan)
# drop rows with missing values
dataset.dropna(inplace=True)
# summarize the shape of the data with missing rows removed
print(dataset.shape)
```

Running this example, we can see that the number of rows has been aggressively cut from 768 in the original dataset to 392 with all rows containing a NaN removed.

```
(768, 9)
(392, 9)
```

We now have a dataset that we could use to evaluate an algorithm sensitive to missing values like LDA.

```
# evaluate model on data after rows with missing data are removed
from numpy import nan
from pandas import read_csv
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
# load the dataset
dataset = read_csv('pima-indians-diabetes.csv', header=None)
# replace '0' values with 'nan'
dataset[[1,2,3,4,5]] = dataset[[1,2,3,4,5]].replace(0, nan)
# drop rows with missing values
dataset.dropna(inplace=True)
# split dataset into inputs and outputs
values = dataset.values
X = values[:,0:8]
```

```
X = values[:,0:8]
y = values[:,8]
# define the model
model = LinearDiscriminantAnalysis()
# define the model evaluation procedure
cv = KFold(n_splits=3, shuffle=True, random_state=1)
# evaluate the model
result = cross_val_score(model, X, y, cv=cv, scoring='accuracy')
# report the mean performance
print('Accuracy: %.3f' % result.mean())
```

Note: Your results may vary given the stochastic nature of the algorithm or evaluation procedure, or differences in numerical precision. Consider running the example a few times and compare the average outcome.

The example runs successfully and prints the accuracy of the model.

```
Accuracy: 0.781
```

Removing rows with missing values can be too limiting on some predictive modeling problems, an alternative is to impute missing values.

5. Impute Missing Values

Imputing refers to using a model to replace missing values.

> *… missing data can be imputed. In this case, we can use information in the training set predictors to, in essence, estimate the values of other predictors.*

— Page 42, Applied Predictive Modeling, 2013.

There are many options we could consider when replacing a missing value, for example:

- A constant value that has meaning within the domain, such as 0, distinct from all other values.
- A value from another randomly selected record.
- A mean, median or mode value for the column.
- A value estimated by another predictive model.

Any imputing performed on the training dataset will have to be performed on new data in the future when predictions are needed from the finalized model. This needs to be taken into consideration when choosing how to impute the missing values.

For example, if you choose to impute with mean column values, these mean column values will need to be stored to file for later use on new data that has missing values.

Pandas provides the fillna() function for replacing missing values with a specific value.

For example, we can use fillna() to replace missing values with the mean value for each column, as follows:

```
# manually impute missing values with numpy
from pandas import read_csv
from numpy import nan
# load the dataset
dataset = read_csv('pima-indians-diabetes.csv', header=None)
# mark zero values as missing or NaN
dataset[[1,2,3,4,5]] = dataset[[1,2,3,4,5]].replace(0, nan)
# fill missing values with mean column values
dataset.fillna(dataset.mean(), inplace=True)
# count the number of NaN values in each column
print(dataset.isnull().sum())
```

Running the example provides a count of the number of missing values in each column, showing zero missing values.

```
0    0
1    0
2    0
3    0
4    0
5    0
6    0
7    0
8    0
dtype: int64
```

The scikit-learn library provides the SimpleImputer pre-processing class that can be used to replace missing values.

It is a flexible class that allows you to specify the value to replace (it can be something other than NaN) and the technique used to replace it (such as mean, median, or mode). The SimpleImputer class operates directly on the NumPy array instead of the DataFrame.

The example below uses the SimpleImputer class to replace missing values with the mean of each column then prints the number of NaN values in the transformed matrix.

```
# example of imputing missing values using scikit-learn
from numpy import nan
from numpy import isnan
from pandas import read_csv
from sklearn.impute import SimpleImputer
# load the dataset
dataset = read_csv('pima-indians-diabetes.csv', header=None)
# mark zero values as missing or NaN
dataset[[1,2,3,4,5]] = dataset[[1,2,3,4,5]].replace(0, nan)
# retrieve the numpy array
values = dataset.values
# define the imputer
imputer = SimpleImputer(missing_values=nan, strategy='mean')
# transform the dataset
transformed_values = imputer.fit_transform(values)
# count the number of NaN values in each column
print('Missing: %d' % isnan(transformed_values).sum())
```

Running the example shows that all NaN values were imputed successfully.

```
Missing: 0
```

In either case, we can train algorithms sensitive to NaN values in the transformed dataset, such as LDA.

The example below shows the LDA algorithm trained in the *SimpleImputer* transformed dataset.

We use a Pipeline to define the modeling pipeline, where data is first passed through the imputer transform, then provided to the model. This ensures that the imputer and model are both fit only on the training dataset and evaluated on the test dataset within each cross-validation fold. This is important to avoid data leakage.

```
# example of evaluating a model after an imputer transform
from numpy import nan
from pandas import read_csv
from sklearn.pipeline import Pipeline
from sklearn.impute import SimpleImputer
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
dataset = read_csv('pima-indians-diabetes.csv', header=None)
# mark zero values as missing or NaN
dataset[[1,2,3,4,5]] = dataset[[1,2,3,4,5]].replace(0, nan)
# split dataset into inputs and outputs
values = dataset.values
X = values[:,0:8]
y = values[:,8]
# define the imputer
imputer = SimpleImputer(missing_values=nan, strategy='mean')
# define the model
lda = LinearDiscriminantAnalysis()
# define the modeling pipeline
pipeline = Pipeline(steps=[('imputer', imputer),('model', lda)])
# define the cross validation procedure
kfold = KFold(n_splits=3, shuffle=True, random_state=1)
# evaluate the model
result = cross_val_score(pipeline, X, y, cv=kfold, scoring='accuracy')
# report the mean performance
print('Accuracy: %.3f' % result.mean())
```

Note: Your results may vary given the stochastic nature of the algorithm or evaluation procedure, or differences in numerical precision. Consider running the example a few times and compare the average outcome.

Running the example prints the accuracy of LDA on the transformed dataset.

```
Accuracy: 0.762
```

Try replacing the missing values with other values and see if you can lift the performance of the model.

Maybe missing values have meaning in the data.

For a more detailed example of imputing missing values with statistics see the tutorial:

- Statistical Imputation for Missing Values in Machine Learning

Next we will look at using algorithms that treat missing values as just another value when modeling.

6. Algorithms that Support Missing Values

Not all algorithms fail when there is missing data.

There are algorithms that can be made robust to missing data, such as k-Nearest Neighbors that can ignore a column from a distance measure when a value is missing. Naive Bayes can also support missing values when making a prediction.

> *One of the really nice things about Naive Bayes is that missing values are no problem at all.*

— Page 100, Data Mining: Practical Machine Learning Tools and Techniques, 2016.

There are also algorithms that can use the missing value as a unique and different value when building the predictive model, such as classification and regression trees.

> *… a few predictive models, especially tree-based techniques, can specifically account for missing data.*

— Page 42, Applied Predictive Modeling, 2013.

Sadly, the scikit-learn implementations of naive bayes, decision trees and k-Nearest Neighbors are not robust to missing values. Although it is being considered.

Nevertheless, this remains as an option if you consider using another algorithm implementation (such as xgboost) or developing your own implementation.

Further Reading

This section provides more resources on the topic if you are looking to go deeper.

Related Tutorials

- Statistical Imputation for Missing Values in Machine Learning

Books

- Feature Engineering and Selection, 2019.
- Data Mining: Practical Machine Learning Tools and Techniques, 2016.
- Feature Engineering and Selection, 2019.
- Applied Predictive Modeling, 2013.

APIs

- Working with missing data, in Pandas
- Imputation of missing values, in scikit-learn

Summary

In this tutorial, you discovered how to handle machine learning data that contains missing values.

Specifically, you learned:

- How to mark missing values in a dataset as numpy.nan.
- How to remove rows from the dataset that contain missing values.
- How to replace missing values with sensible values.

Do you have any questions about handling missing values?
Ask your questions in the comments and I will do my best to answer.

About Jason Brownlee

Jason Brownlee, PhD is a machine learning specialist who teaches developers how to get results with modern machine learning methods via hands-on tutorials.
View all posts by Jason Brownlee →

< How to Train a Final Machine Learning Model

Time Series Forecasting with Python 7-Day Mini-Course >