How to Handle Missing Data with Python

by **Jason Brownlee** on March 20, 2017 in **Python Machine Learning**

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.

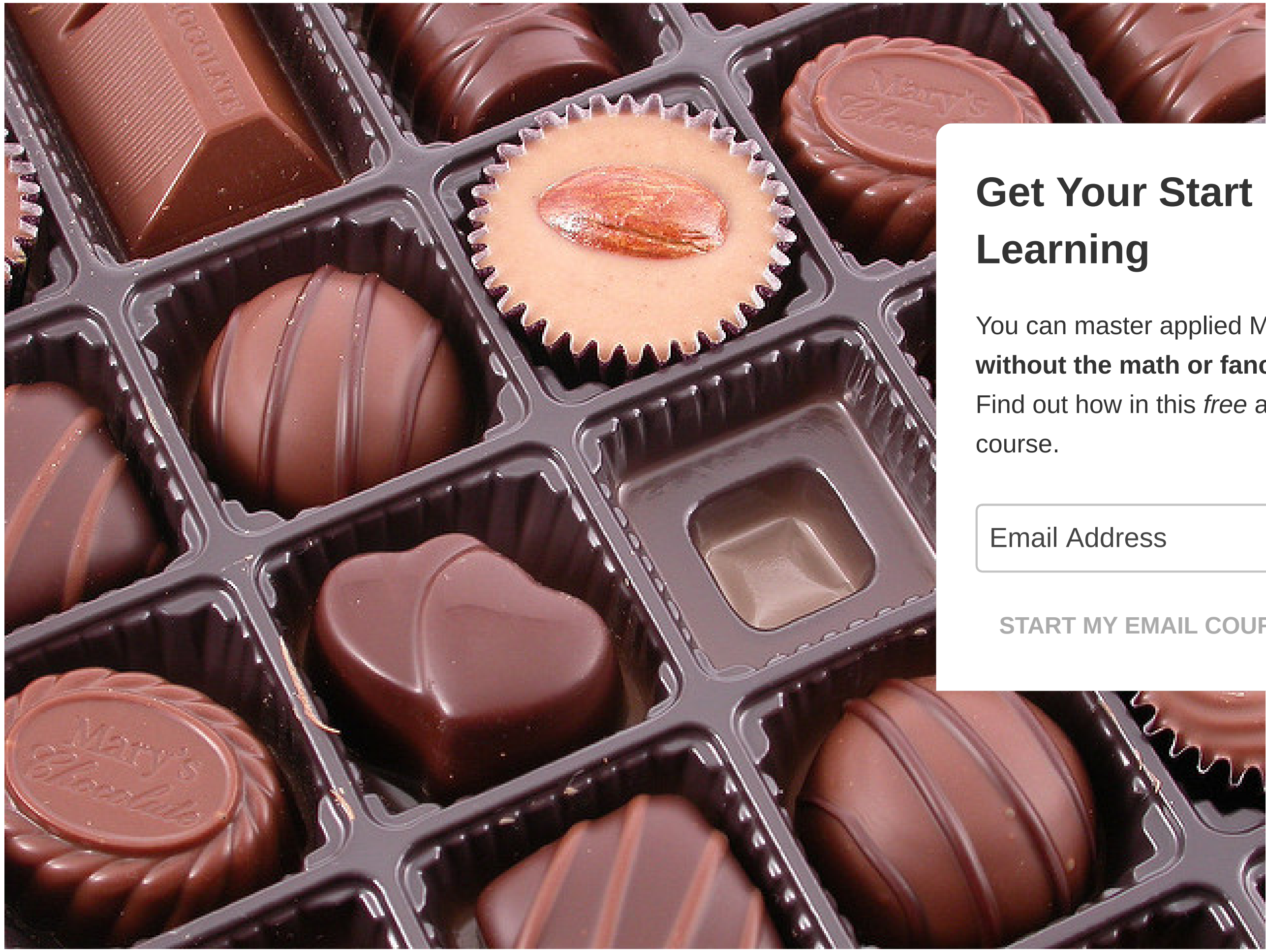
In this tutorial, you will discover how to handle missing data for machine learning with Python.

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.

Let’s get started.

Note: The examples in this post assume that you have Python 2 or 3 with Pandas, NumPy and Scikit-Learn installed, specifically scikit-learn version 0.18 or higher.



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

Overview

This tutorial is divided into 6 parts:

1. **Pima Indians 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. Pima Indians Diabetes Dataset

The Pima Indians Diabetes Dataset involves predicting the onset of diabetes within 5 years in Pima

It is a binary (2-class) classification problem. The number of observations for each class is not balan 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*

2. Mark Missing Values

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.

```python
from pandas import read_csv
dataset = read_csv('pima-indians-diabetes.csv', header=None)
print(dataset.describe())
```

Running this example produces the following output:

```
                0           1           2           3           4           5  \
count  768.000000  768.000000  768.000000  768.000000  768.000000  768.000000
mean     3.845052  120.894531   69.105469   20.536458   79.799479   31.992578
std      3.369578   31.972618   19.355807   15.952218  115.244002    7.884160
min      0.000000    0.000000    0.000000    0.000000    0.000000    0.000000
25%      1.000000   99.000000   62.000000    0.000000    0.000000   27.300000
50%      3.000000  117.000000   72.000000   23.000000   30.500000   32.000000
75%      6.000000  140.250000   80.000000   32.000000  127.250000   36.600000
max     17.000000  199.000000  122.000000   99.000000  846.000000   67.100000
```

```

                6           7           8
count  768.000000  768.000000  768.000000
mean     0.471876   33.240885    0.348958
std      0.331329   11.760232    0.476951
min      0.078000   21.000000    0.000000
25%      0.243750   24.000000    0.000000
50%      0.372500   29.000000    0.000000
75%      0.626250   41.000000    1.000000
max      2.420000   81.000000    1.000000
```

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.

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

```
from pandas import read_csv
import numpy
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 tru

We can do this my marking all of the values in the subset of the DataFrame we are interested in that

number of true values in each column.

```
from pandas import read_csv
dataset = read_csv('pima-indians-diabetes.csv', header=None)
print((dataset[[1,2,3,4,5]] == 0).sum())
```

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.

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 [redacted]

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.

```
from pandas import read_csv
import numpy
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, numpy.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 correct

We can see that the columns 1 to 5 have the same number of missing values as zero values identifi identified missing values correctly.

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

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

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

```
from pandas import read_csv
import numpy
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, numpy.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 [redacted] 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 dat

3. Missing Values Causes Problems

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

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.

```
from pandas import read_csv
import numpy
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, numpy.NaN)
# split dataset into inputs and outputs
values = dataset.values
X = values[:,0:8]
y = values[:,8]
# evaluate an LDA model on the dataset using k-fold cross validation
model = LinearDiscriminantAnalysis()
kfold = KFold(n_splits=3, random_state=7)
result = cross_val_score(model, X, y, cv=kfold, scoring='accuracy')
print(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 miss

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.

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:

```
from pandas import read_csv
import numpy
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, numpy.NaN)
# drop rows with missing values
dataset.dropna(inplace=True)
# summarize the number of rows and columns in the dataset
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.

```
(392, 9)
```

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

```
from pandas import read_csv
import numpy
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, numpy.NaN)
# drop rows with missing values
dataset.dropna(inplace=True)
# split dataset into inputs and outputs
values = dataset.values
X = values[:,0:8]
y = values[:,8]
# evaluate an LDA model on the dataset using k-fold cross validation
model = LinearDiscriminantAnalysis()
kfold = KFold(n_splits=3, random_state=7)
result = cross_val_score(model, X, y, cv=kfold, scoring='accuracy')
print(result.mean())
```


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

```
0.78582892934
```

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.

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

```
from pandas import read_csv
import numpy
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, numpy.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 zer

```
0    0
1    0
2    0
3    0
4    0
5    0
6    0
7    0
8    0
```

The scikit-learn library provides the Imputer() pre-processing class that can be used to replace missi

It is a flexible class that allows you to specify the value to replace (it can be something other than Na[redacted] mean, median, or mode). The Imputer class operates directly on the NumPy array instead of the DataFrame.

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

```
from pandas import read_csv
from sklearn.preprocessing import Imputer
import numpy
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, numpy.NaN)
# fill missing values with mean column values
values = dataset.values
imputer = Imputer()
transformed_values = imputer.fit_transform(values)
# count the number of NaN values in each column
print(numpy.isnan(transformed_values).sum())
```

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

```

```

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

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

```
from pandas import read_csv
import numpy
from sklearn.preprocessing import Imputer
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, numpy.NaN)
# split dataset into inputs and outputs
values = dataset.values
X = values[:,0:8]
y = values[:,8]
# fill missing values with mean column values
imputer = Imputer()
transformed_X = imputer.fit_transform(X)
```

```
# evaluate an LDA model on the dataset using k-fold cross validation
model = LinearDiscriminantAnalysis()
kfold = KFold(n_splits=3, random_state=7)
result = cross_val_score(model, transformed_X, y, cv=kfold, scoring='accuracy')
print(result.mean())
```

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

```
0.766927083333
```

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.

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 ca a value is missing.

There are also algorithms that can use the missing value as a unique and different value when build d regression trees.

Sadly, the scikit-learn implementations of decision trees and k-Nearest Neighbors are not robust to r

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

Further Reading

- 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

Dr. Jason Brownlee is a husband, proud father, academic researcher, author, professional developer and a machine learning practitioner. He is dedicated to helping developers get started and get good at applied machine learning. Learn more.

View all posts by Jason Brownlee →

‹ How to Train a Final Machine Learning Model

rse ›

17 Responses to *How to Handle Missing Data with Python*

Mike March 20, 2017 at 3:16 pm #

Fancy impute is a library i've turned too for imputation:

https://github.com/hammerlab/fancyimpute

Also missingno is great for visualizations!

https://github.com/ResidentMario/missingno

Jason Brownlee March 21, 2017 at 8:37 am #

REPLY

Thanks for the tip Mike.

bakyalakshmi September 27, 2017 at 2:56 pm # REPLY

please tell me about how to impute median using one dataset

Jozo Kovac April 1, 2017 at 8:06 am # REPLY

Thanks for pointing on interesting problem. I would love another one about how to deal with cate

And dear reader, please never ever remove rows with missing values. It changes the distribution of your
Learn from mistakes of others and don't repeat them 🙂

Jason Brownlee April 2, 2017 at 6:22 am #

Thanks Jozo.

This post will help with categorical input data:
http://machinelearningmastery.com/data-preparation-gradient-boosting-xgboost-python/

Tommy Carstensen April 4, 2017 at 3:56 am # REPLY

Super duper! Thanks for writing! Would it have been worth mentioning interpolate of Pandas? http://pandas.pydata.org/pandas-docs/stable/generated/pandas.Series.interpolate.html

Jason Brownlee April 4, 2017 at 9:18 am #

Thanks Tommy.

Aswathy April 14, 2017 at 12:10 pm # REPLY

Hi Jason,

I was just wondering if there is a way to use a different imputation strategy for each column. Say, for a categorical feature you want to impute using the mode but for a continuous attribute, you want to impute using mean.

Jason Brownlee April 15, 2017 at 9:30 am #

Yes, try lots of techniques, go with whatever results in the most accurate models.

Salu Khadka June 11, 2017 at 12:29 am #

thanks for your tutorial sir.
I would also seek help from you for multi label classification of a textual data , if possible.

For example, categorizing a twitter post as related to sports, business , tech , or others.

Jason Brownlee June 11, 2017 at 8:26 am # REPLY

Sure, see this post:
http://machinelearningmastery.com/sequence-classification-lstm-recurrent-neural-networks-python-keras/

Ali Gabriel Lara June 13, 2017 at 4:51 am #

Hello Mr. Brownlee. Thank you so much for your post. [redacted]

Do you know any approach to recognize the pattern of missing data? I mean, I am interested in discovering the pattern of missing data on a time series data.

The database is historical data of a chemical process. I think I should apply some pattern recognition approach columnwise because each column represents a process variable and the value coming from a transmisor.

My goal is to predict if the missing data is for a mechanical fault or a desviation in registration process or for any other causes. Then I should apply a kind of filling methods if it is required.

Have you any advice? Thanks in advance

Jason Brownlee June 13, 2017 at 8:25 am #

I would invert the problem and model the series of missing data and mark all data you do h
as “1”.

Great problem!

Let me know how you go.

Patricia Villa October 5, 2017 at 3:45 pm #

You helped me keep my sanity. THANK YOU!!

Jason Brownlee October 5, 2017 at 5:23 pm #

REPLY

I’m really glad to hear that Patricia!

Sachin Raj October 6, 2017 at 7:58 pm # REPLY

How to know whether to apply mean or to replace it with mode?

Jason Brownlee October 7, 2017 at 5:54 am # REPLY

Try both and see what results in the most skillful models.

Leave a Reply

Name (required)

Email (will not be published) (required)

Website

SUBMIT COMMENT

Welcome to Machine Learning Mastery

Hi, I'm Dr. Jason Brownlee.
My goal is to make practitioners like YOU awesome at applied machine learning.

Read More

Develop Predictive Models With Python

Want to develop your own models in scikit-learn?
Want step-by-step tutorials?
Looking for sample code and templates?

Get Your Start in Machine Learning

POPULAR


Time Series Prediction with LSTM Recurrent Neural Networks in Python with Keras
JULY 21, 2016


Your First Machine Learning Project in Python Step-By-Step
JUNE 10, 2016


Develop Your First Neural Network in Python With Keras Step-By-Step
MAY 24, 2016

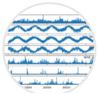

Sequence Classification with LSTM Recurrent Neural Networks in Python with Keras
JULY 26, 2016

How to Setup a Python Environment for Machine Learning and Deep Learning with Anaconda
MARCH 13, 2017

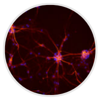
Time Series Forecasting with the Long Short-Term Memory Network in Python
APRIL 7, 2017

Multi-Class Classification Tutorial with the Keras Deep Learning Library
JUNE 2, 2016

Regression Tutorial with the Keras Deep Learning Library in Python
JUNE 9, 2016



Multivariate Time Series Forecasting with LSTMs in Keras
AUGUST 14, 2017



How to Implement the Backpropagation Algorithm From Scratch In Python
NOVEMBER 7, 2016

Privacy | Contact | About

Get Your Start in Machine Learning