# How to Convert a Time Series to a Supervised Learning Problem in Python

by **Jason Brownlee** on May 8, 2017 in **Time Series**

Machine learning methods like deep learning can be used for time series forecasting.

Before machine learning can be used, time series forecasting problems must be re-framed as super s of input and output sequences.

In this tutorial, you will discover how to transform univariate and multivariate time series forecasting p e with machine learning algorithms.

After completing this tutorial, you will know:

- How to develop a function to transform a time series dataset into a supervised learning dataset.
- How to transform univariate time series data for machine learning.

- How to transform multivariate time series data for machine learning.

Let's get started.

How to Convert a Time Series to a Supervised Learning Problem i
Photo by Quim Gil, some rights reserved.

## Time Series vs Supervised Learning

Before we get started, let's take a moment to better understand the form of time series and supervis

A time series is a sequence of numbers that are ordered by a time index. This can be thought of as a

For example:

```
0
1
```

```
3  2
4  3
5  4
6  5
7  6
8  7
9  8
10 9
```



A supervised learning problem is comprised of input patterns (*X*) and output patterns (*y*), such that an algorithm can learn how to predict the output patterns from the input patterns.

For example:

```
1 X,  y
2 1   2
3 2,  3
4 3,  4
5 4,  5
6 5,  6
7 6,  7
8 7,  8
9 8,  9
```

For more on this topic, see the post:

- Time Series Forecasting as Supervised Learning

## Pandas shift() Function

A key function to help transform time series data into a supervised learning problem is the Pandas s

Given a DataFrame, the *shift()* function can be used to create copies of columns that are pushed for pulled back (rows of NaN values added to the end).

This is the behavior required to create columns of lag observations as well as columns of forecast ob ed learning format.

Let's look at some examples of the shift function in action.

We can define a mock time series dataset as a sequence of 10 numbers, in this case a single column in a DataFrame as follows:

```
from pandas import DataFrame
df = DataFrame()
df['t'] = [x for x in range(10)]
print(df)
```



Running the example prints the time series data with the row indices for each observation.

```
   t
0  0
1  1
2  2
3  3
4  4
5  5
6  6
7  7
8  8
9  9
```

We can shift all the observations down by one time step by inserting one new row at the top. Because the new row has no data, we can use NaN to represent “no data”.

The shift function can do this for us and we can insert this shifted column next to our original series.

```
from pandas import DataFrame
df = DataFrame()
df['t'] = [x for x in range(10)]
df['t-1'] = df['t'].shift(1)
print(df)
```

Running the example gives us two columns in the dataset. The first with the original observations an

We can see that shifting the series forward one time step gives us a primitive supervised learning pr
Ignore the column of row labels. The first row would have to be discarded because of the NaN value
second column (input or *X*) and the value of 1 in the first column (output or *y*).

```
   t  t-1
0  0  NaN
1  1  0.0
2  2  1.0
```

```
5  3  3  2.0
6  4  4  3.0
7  5  5  4.0
8  6  6  5.0
9  7  7  6.0
10 8  8  7.0
11 9  9  8.0
```



We can see that if we can repeat this process with shifts of 2, 3, and more, how we could create long input sequences (*X*) that can be used to forecast an output value (*y*).

The shift operator can also accept a negative integer value. This has the effect of pulling the observations up by inserting new rows at the end. Below is an example:

```
from pandas import DataFrame
df = DataFrame()
df['t'] = [x for x in range(10)]
df['t+1'] = df['t'].shift(-1)
print(df)
```

Running the example shows a new column with a NaN value as the last value.

We can see that the forecast column can be taken as an input (*X*) and the second as an output value (*y*). That is the input value of 0 can be used to forecast the output value of 1.

```
1     t  t+1
2  0  0  1.0
3  1  1  2.0
4  2  2  3.0
5  3  3  4.0
6  4  4  5.0
7  5  5  6.0
8  6  6  7.0
9  7  7  8.0
10 8  8  9.0
11 9  9  NaN
```



Technically, in time series forecasting terminology the current time (*t*) and future times (*t+1*, *t+n*) are used to make forecasts.

We can see how positive and negative shifts can be used to create a new DataFrame from a time series with sequences of input and output patterns for a supervised learning problem.



This permits not only classical *X -> y* prediction, but also *X -> Y* where both input and output can be

Further, the shift function also works on so-called multivariate time series problems. That is where instead of having one set of observations for a time series, we have multiple (e.g. temperature and pressure). All variates in the time series can be shifted forward or backward to create multivariate input and output sequences. We will explore this more later in the tutorial.

## The series_to_supervised() Function

We can use the *shift()* function in Pandas to automatically create new framings of time series problems given the desired length of input and output sequences.

This would be a useful tool as it would allow us to explore different framings of a time series problem with machine learning algorithms to see which might result in better performing models.

In this section, we will define a new Python function named *series_to_supervised()* that takes a univariate or multivariate time series and frames it as a supervised learning dataset.

The function takes four arguments:

- **data**: Sequence of observations as a list or 2D NumPy array. Required.
- **n_in**: Number of lag observations as input (*X*). Values may be between [1..len(data)] Optional. D
- **n_out**: Number of observations as output (*y*). Values may be between [0..len(data)-1]. Optional.
- **dropnan**: Boolean whether or not to drop rows with NaN values. Optional. Defaults to True.

The function returns a single value:

- **return**: Pandas DataFrame of series framed for supervised learning.

The new dataset is constructed as a DataFrame, with each column suitably named both by variable a variety of different time step sequence type forecasting problems from a given univariate or multivari

Once the DataFrame is returned, you can decide how to split the rows of the returned DataFrame into *X* and *y* components for supervised learning any way you wish.

The function is defined with default parameters so that if you call it with just your data, it will construc

The function is confirmed to be compatible with Python 2 and Python 3.

The complete function is listed below, including function comments.

```
from pandas import DataFrame
from pandas import concat

def series_to_supervised(data, n_in=1, n_out=1, dropnan=True):
    """
    Frame a time series as a supervised learning dataset.
    Arguments:
        data: Sequence of observations as a list or NumPy array.
        n_in: Number of lag observations as input (X).
        n_out: Number of observations as output (y).
        dropnan: Boolean whether or not to drop rows with NaN values.
    Returns:
        Pandas DataFrame of series framed for supervised learning.
    """
    n_vars = 1 if type(data) is list else data.shape[1]
    df = DataFrame(data)
    cols, names = list(), list()
    # input sequence (t-n, ... t-1)
    for i in range(n_in, 0, -1):
        cols.append(df.shift(i))
        names += [('var%d(t-%d)' % (j+1, i)) for j in range(n_vars)]
    # forecast sequence (t, t+1, ... t+n)
    for i in range(0, n_out):
        cols.append(df.shift(-i))
        if i == 0:
            names += [('var%d(t)' % (j+1)) for j in range(n_vars)]
        else:
            names += [('var%d(t+%d)' % (j+1, i)) for j in range(n_vars)]
    # put it all together
    agg = concat(cols, axis=1)
    agg.columns = names
    # drop rows with NaN values
    if dropnan:
        agg.dropna(inplace=True)
    return agg
```

Can you see obvious ways to make the function more robust or more readable?
Please let me know in the comments below.

Now that we have the whole function, we can explore how it may be used.



## One-Step Univariate Forecasting

It is standard practice in time series forecasting to use lagged observations (e.g. t-1) as input variables to forecast the current time step (t).

This is called one-step forecasting.

The example below demonstrates a one lag time step (t-1) to predict the current time step (t).

```
from pandas import DataFrame
from pandas import concat

def series_to_supervised(data, n_in=1, n_out=1, dropnan=True):
    """
    Frame a time series as a supervised learning dataset.
    Arguments:
        data: Sequence of observations as a list or NumPy array.
        n_in: Number of lag observations as input (X).
        n_out: Number of observations as output (y).
        dropnan: Boolean whether or not to drop rows with NaN values.
    Returns:
        Pandas DataFrame of series framed for supervised learning.
    """
    n_vars = 1 if type(data) is list else data.shape[1]
    df = DataFrame(data)
    cols, names = list(), list()
    # input sequence (t-n, ... t-1)
    for i in range(n_in, 0, -1):
        cols.append(df.shift(i))
        names += [('var%d(t-%d)' % (j+1, i)) for j in range(n_vars)]
    # forecast sequence (t, t+1, ... t+n)
    for i in range(0, n_out):
        cols.append(df.shift(-i))
        if i == 0:
            names += [('var%d(t)' % (j+1)) for j in range(n_vars)]
        else:
            names += [('var%d(t+%d)' % (j+1, i)) for j in range(n_vars)]
    # put it all together
```

```
    agg = concat(cols, axis=1)
    agg.columns = names
    # drop rows with NaN values
    if dropnan:
        agg.dropna(inplace=True)
    return agg


values = [x for x in range(10)]
data = series_to_supervised(values)
print(data)
```



Running the example prints the output of the reframed time series.

```
   var1(t-1)  var1(t)
1        0.0        1
2        1.0        2
3        2.0        3
4        3.0        4
5        4.0        5
6        5.0        6
7        6.0        7
8        7.0        8
9        8.0        9
```

We can see that the observations are named "*var1*" and that the input observation is suitably named (t-1) and the output time step is named (t).

We can also see that rows with NaN values have been automatically removed from the DataFrame.

We can repeat this example with an arbitrary number length input sequence, such as 3. This can be ce as an argument; for example:

```
data = series_to_supervised(values, 3)
```

The complete example is listed below.

```
from pandas import DataFrame
from pandas import concat

def series_to_supervised(data, n_in=1, n_out=1, dropnan=True):
    """
    Frame a time series as a supervised learning dataset.
    Arguments:
        data: Sequence of observations as a list or NumPy array.
```

```
        n_in: Number of lag observations as input (X).
        n_out: Number of observations as output (y).
        dropnan: Boolean whether or not to drop rows with NaN values.
    Returns:
        Pandas DataFrame of series framed for supervised learning.
    """
    n_vars = 1 if type(data) is list else data.shape[1]
    df = DataFrame(data)
    cols, names = list(), list()
    # input sequence (t-n, ... t-1)
    for i in range(n_in, 0, -1):
        cols.append(df.shift(i))
        names += [('var%d(t-%d)' % (j+1, i)) for j in range(n_vars)]
    # forecast sequence (t, t+1, ... t+n)
    for i in range(0, n_out):
        cols.append(df.shift(-i))
        if i == 0:
            names += [('var%d(t)' % (j+1)) for j in range(n_vars)]
        else:
            names += [('var%d(t+%d)' % (j+1, i)) for j in range(n_vars)]
    # put it all together
    agg = concat(cols, axis=1)
    agg.columns = names
    # drop rows with NaN values
    if dropnan:
        agg.dropna(inplace=True)
    return agg

values = [x for x in range(10)]
data = series_to_supervised(values, 3)
print(data)
```

Again, running the example prints the reframed series. We can see that the input sequence is in the ... o be predicted on the far right.

```
   var1(t-3)  var1(t-2)  var1(t-1)  var1(t)
3        0.0        1.0        2.0        3
4        1.0        2.0        3.0        4
5        2.0        3.0        4.0        5
6        3.0        4.0        5.0        6
7        4.0        5.0        6.0        7
8        5.0        6.0        7.0        8
9        6.0        7.0        8.0        9
```

# Multi-Step or Sequence Forecasting

A different type of forecasting problem is using past observations to forecast a sequence of future ob

This may be called sequence forecasting or multi-step forecasting.

We can frame a time series for sequence forecasting by specifying another argument. For example, we could frame a forecast problem with an input sequence of 2 past observations to forecast 2 future observations as follows:

```
data = series_to_supervised(values, 2, 2)
```

The complete example is listed below:

```
from pandas import DataFrame
from pandas import concat

def series_to_supervised(data, n_in=1, n_out=1, dropnan=True):
    """
    Frame a time series as a supervised learning dataset.
    Arguments:
        data: Sequence of observations as a list or NumPy array.
        n_in: Number of lag observations as input (X).
        n_out: Number of observations as output (y).
        dropnan: Boolean whether or not to drop rows with NaN values.
    Returns:
        Pandas DataFrame of series framed for supervised learning.
    """
    n_vars = 1 if type(data) is list else data.shape[1]
    df = DataFrame(data)
    cols, names = list(), list()
    # input sequence (t-n, ... t-1)
    for i in range(n_in, 0, -1):
        cols.append(df.shift(i))
        names += [('var%d(t-%d)' % (j+1, i)) for j in range(n_vars)]
    # forecast sequence (t, t+1, ... t+n)
    for i in range(0, n_out):
        cols.append(df.shift(-i))
        if i == 0:
            names += [('var%d(t)' % (j+1)) for j in range(n_vars)]
        else:
            names += [('var%d(t+%d)' % (j+1, i)) for j in range(n_vars)]
    # put it all together
    agg = concat(cols, axis=1)
```

```
    agg.columns = names
    # drop rows with NaN values
    if dropnan:
        agg.dropna(inplace=True)
    return agg


values = [x for x in range(10)]
data = series_to_supervised(values, 2, 2)
print(data)
```



Running the example shows the differentiation of input (t-n) and output (t+n) variables with the current observation (t) considered an output.

```
   var1(t-2)  var1(t-1)  var1(t)  var1(t+1)
2        0.0        1.0        2        3.0
3        1.0        2.0        3        4.0
4        2.0        3.0        4        5.0
5        3.0        4.0        5        6.0
6        4.0        5.0        6        7.0
7        5.0        6.0        7        8.0
8        6.0        7.0        8        9.0
```

## Multivariate Forecasting

Another important type of time series is called multivariate time series.

This is where we may have observations of multiple different measures and an interest in forecastin

For example, we may have two sets of time series observations obs1 and obs2 and we wish to forec

We can call *series_to_supervised()* in exactly the same way.

For example:

```
from pandas import DataFrame
from pandas import concat

def series_to_supervised(data, n_in=1, n_out=1, dropnan=True):
    """
    Frame a time series as a supervised learning dataset.
    Arguments:
        data: Sequence of observations as a list or NumPy array.
```

```
        n_in: Number of lag observations as input (X).
        n_out: Number of observations as output (y).
        dropnan: Boolean whether or not to drop rows with NaN values.
    Returns:
        Pandas DataFrame of series framed for supervised learning.
    """
    n_vars = 1 if type(data) is list else data.shape[1]
    df = DataFrame(data)
    cols, names = list(), list()
    # input sequence (t-n, ... t-1)
    for i in range(n_in, 0, -1):
        cols.append(df.shift(i))
        names += [('var%d(t-%d)' % (j+1, i)) for j in range(n_vars)]
    # forecast sequence (t, t+1, ... t+n)
    for i in range(0, n_out):
        cols.append(df.shift(-i))
        if i == 0:
            names += [('var%d(t)' % (j+1)) for j in range(n_vars)]
        else:
            names += [('var%d(t+%d)' % (j+1, i)) for j in range(n_vars)]
    # put it all together
    agg = concat(cols, axis=1)
    agg.columns = names
    # drop rows with NaN values
    if dropnan:
        agg.dropna(inplace=True)
    return agg

raw = DataFrame()
raw['ob1'] = [x for x in range(10)]
raw['ob2'] = [x for x in range(50, 60)]
values = raw.values
data = series_to_supervised(values)
print(data)
```

Running the example prints the new framing of the data, showing an input pattern with one time step ... me step for both variables.

Again, depending on the specifics of the problem, the division of columns into *X* and *Y* components ... observation of *var1* was also provided as input and only *var2* was to be predicted.

```
   var1(t-1)  var2(t-1)  var1(t)  var2(t)
1        0.0       50.0        1       51
2        1.0       51.0        2       52
```

```
4  3      2.0      52.0      3      53
5  4      3.0      53.0      4      54
6  5      4.0      54.0      5      55
7  6      5.0      55.0      6      56
8  7      6.0      56.0      7      57
9  8      7.0      57.0      8      58
10 9      8.0      58.0      9      59
```



You can see how this may be easily used for sequence forecasting with multivariate time series by specifying the length of the input and output sequences as above.

For example, below is an example of a reframing with 1 time step as input and 2 time steps as forecast sequence.

```
from pandas import DataFrame
from pandas import concat

def series_to_supervised(data, n_in=1, n_out=1, dropnan=True):
    """
    Frame a time series as a supervised learning dataset.
    Arguments:
        data: Sequence of observations as a list or NumPy array.
        n_in: Number of lag observations as input (X).
        n_out: Number of observations as output (y).
        dropnan: Boolean whether or not to drop rows with NaN values.
    Returns:
        Pandas DataFrame of series framed for supervised learning.
    """
    n_vars = 1 if type(data) is list else data.shape[1]
    df = DataFrame(data)
    cols, names = list(), list()
    # input sequence (t-n, ... t-1)
    for i in range(n_in, 0, -1):
        cols.append(df.shift(i))
        names += [('var%d(t-%d)' % (j+1, i)) for j in range(n_vars)]
    # forecast sequence (t, t+1, ... t+n)
    for i in range(0, n_out):
        cols.append(df.shift(-i))
        if i == 0:
            names += [('var%d(t)' % (j+1)) for j in range(n_vars)]
        else:
            names += [('var%d(t+%d)' % (j+1, i)) for j in range(n_vars)]
    # put it all together
    agg = concat(cols, axis=1)
    agg.columns = names
    # drop rows with NaN values
    if dropnan:
```

```
        agg.dropna(inplace=True)
    return agg


raw = DataFrame()
raw['ob1'] = [x for x in range(10)]
raw['ob2'] = [x for x in range(50, 60)]
values = raw.values
data = series_to_supervised(values, 1, 2)
print(data)
```



Running the example shows the large reframed DataFrame.

```
   var1(t-1)  var2(t-1)  var1(t)  var2(t)  var1(t+1)  var2(t+1)
1        0.0       50.0        1       51        2.0       52.0
2        1.0       51.0        2       52        3.0       53.0
3        2.0       52.0        3       53        4.0       54.0
4        3.0       53.0        4       54        5.0       55.0
5        4.0       54.0        5       55        6.0       56.0
6        5.0       55.0        6       56        7.0       57.0
7        6.0       56.0        7       57        8.0       58.0
8        7.0       57.0        8       58        9.0       59.0
```

Experiment with your own dataset and try multiple different framings to see what works best.

## Summary

In this tutorial, you discovered how to reframe time series datasets as supervised learning problems

Specifically, you learned:

- About the Pandas *shift()* function and how it can be used to automatically define supervised lea
- How to reframe a univariate time series into one-step and multi-step supervised learning proble
- How to reframe multivariate time series into one-step and multi-step supervised learning proble

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

### About Jason Brownlee

Dr. Jason Brownlee is a husband, proud father, academic researcher, author, professional develo
to helping developers get started and get good at applied machine learning. Learn more.
View all posts by Jason Brownlee →

‹ How to Use Weight Regularization with LSTM Networks for Time Series Forecasting

Multi-step Time Series I hon ›

## 40 Responses to *How to Convert a Time Series to a Supervised Learning P*



**Mikkel** May 8, 2017 at 7:07 pm #

REPLY

Hi Jason, thanks for your highly relevant article 🙂

I am having a hard time following the structure of the dataset. I understand the basics of t-n, t-1, t, t+1, t+n and so forth. Although, what exactly are we describing here in the t and t-1 column? Is it the change over time for a specific explanatory variable? In that case, wouldn't it make more sense to transpose the data, so that the time were described in the rows rather than columns?

Also, how would you then characterise following data:

Customer_ID Month Balance
1 01 1,500
1 02 1,600
1 03 1,700
1 04 1,900
2 01 1,000
2 02 900
2 03 700
2 04 500
3 01 3,500
3 02 1,500
3 03 2,500
3 04 4,500

Let's say, that we wanna forcast their balance using supervised learning, or classify the customers as "s



**Jason Brownlee** May 9, 2017 at 7:40 am #

Yes, it is transposing each variable, but allowing control over the length of each row back in

**Daniel** May 9, 2017 at 4:56 pm #

Hey Jason,

this is an awesome article! I was looking for that the whole time.

The only thing is I am general programming in R, so I only found something similar like your code, but I am not sure if it is the same. I have got this from https://www.r-bloggers.com/generating-a-laglead-variables/ and it deals with lagged and leaded values. Also the output includes NA values.

```
shift1)
return(sapply(shift_by,shift, x=x))

out 0 )
out<-c(tail(x,-abs_shift_by),rep(NA,abs_shift_by))
else if (shift_by < 0 )
out<-c(rep(NA,abs_shift_by), head(x,-abs_shift_by))
else
out<-x
out
}
```

Output:

```
x df_lead2 df_lag2
1 1 3 NA
2 2 4 NA
3 3 5 1
4 4 6 2
5 5 7 3
6 6 8 4
7 7 9 5
8 8 10 6
9 9 NA 7
10 10 NA 8
```

I also tried to recompile your code in R, but it failed.

**Jason Brownlee** May 10, 2017 at 8:44 am #

I would recommend contacting the authors of the R code you reference.

**chris** May 12, 2017 at 3:28 am #

REPLY

Can you answer this in Go, Java, C# and COBOL as well????? Thanks, I really don't want to do anything

**Jason Brownlee** May 12, 2017 at 7:45 am #

REPLY

I do my best to help, some need more help than others.

**Lee** May 9, 2017 at 11:40 pm #

REPLY

Hi Jason, good article, but could be much better if you illustrated everything with some actual ti
code 5 times 😉 Gripes aside, this was very timely as I'm just about to get into some time series forecas

**Jason Brownlee** May 10, 2017 at 8:48 am #

Thanks for the suggestion Lee.

**Christopher** May 12, 2017 at 9:16 pm #

Hi Jason,
thank you for the good article! I really like the shifting approach for reframing the training data!
But my question about this topic is: What do you think is the next step for a one-step univariate
most suitable for that?
Obviously a regressor is the best choice but how can I determine the size of the sliding window for the training?

Thanks a lot for your help and work
~ Christopher



**Jason Brownlee** May 13, 2017 at 6:14 am #

REPLY

My advice is to trial a suite of different window sizes and see what works best.

You can use an ACF/PACF plots as a heuristic:
http://machinelearningmastery.com/gentle-introduction-autocorrelation-partial-autocorrelation/

**tom** June 8, 2017 at 4:06 pm #

REPLY

hi Jason：
In this post, you create new framings of time series ,such as t-1, t, t+1.But, what's the use of these time
good effect on model? Maybe
my question is too simple ,because I am a newer ,please understand! thank you !

**Jason Brownlee** June 9, 2017 at 6:19 am #

I am providing a technique to help you convert a series into a supervised learning problem.

This is valuable because you can then transform your time series problems into supervised learning
and regression techniques in order to make forecasts.

**tom** June 9, 2017 at 11:40 am #

REPLY

Wow, your answer always makes me learn a lot。Thank you Jason!



**Jason Brownlee** June 10, 2017 at 8:12 am #

REPLY

You're welcome.

**Brad Suzon** June 23, 2017 at 11:32 pm #

REPLY

If there are multiple variables varXi to train and only one variable varY to predict will the same technique be used in the below way:
varX1(t-1) varX2(t-1) varX1(t) varX2(t) … varY(t-1) varY(t)
.. .. .. .. .. ..
and then use linear regression and as Response= varY(t) ?

Thanks in advance

**Jason Brownlee** June 24, 2017 at 8:03 am #

Not sure I follow your question Brad, perhaps you can restate it?

**Brad** June 25, 2017 at 4:47 pm #

In case there are multiple measures and then make the transformation in order to forecast

var1(t-1) var2(t-1) var1(t) var2(t) … varN(t-1) varN(t)

linear regression should use as the response variable the varN(t) ?

**Jason Brownlee** June 26, 2017 at 6:06 am #

Sure.

**Geoff** June 24, 2017 at 8:10 am #

REPLY

Hi Jason,
I've found your articles very useful during my capstone at a bootcamp I'm attending. I have two questions that I hope you could advise where to find better info about.
First, I've run into an issue with running PCA on the newly supervised version only the data. Does PCA recognize that the lagged series are actually the same data? If one was to do PCA do they need to perform it before supervising the data?
Secondly, what do you propose as the best learning algorithms and proper ways to perform train test splits on the data?
Thanks again,

**Jason Brownlee** June 25, 2017 at 5:58 am #

Sorry, I have not used PCA on time series data. I expect careful handling of the framing of t

I have tips for backtesting on time series data here:
http://machinelearningmastery.com/backtest-machine-learning-models-time-series-forecasting/

**Kushal** July 1, 2017 at 1:31 pm #

Hi Jason

Great post.

Just one question. If the some of the input variables are continuous and some are categorical with one binary, predicting two output variables.

How does the shift work then?

Thanks
Kushal



**Jason Brownlee** July 2, 2017 at 6:26 am #

REPLY

The same, but consider encoding your categorical variables first (e.g. number encoding or one hot encoding).

**Kushal** July 15, 2017 at 5:22 pm #

REPLY

Thanks

Should I then use the lagged versions of the predictors?

Kushal

**Jason Brownlee** July 16, 2017 at 7:57 am #

Perhaps, I do not follow your question, perhaps you can restate it with more informa

**Viorel Emilian Teodorescu** July 8, 2017 at 9:45 am #

great article, Jason!

**Jason Brownlee** July 9, 2017 at 10:50 am #

REPLY

Thanks, I hope it helps.

**Chinesh** August 10, 2017 at 5:15 pm #

REPLY

very helpful article !!

I am working on developing an algorithm which will predict the future traffic for the restaurant. The features I am using are: Day,whether there was festival,temperature,climatic condition , current rating,whether there was holiday,service rating,number of reviews etc.Can I solve this problem using time series analysis along with these features,If yes how.
Please guide me

**Jason Brownlee** August 11, 2017 at 6:36 am #

REPLY

This process will guide you:
http://machinelearningmastery.com/start-here/#process

**Hossein** August 23, 2017 at 1:16 am #

Great article Jason. Just a naive question: How does this method different from moving averag
Thanks

**Jason Brownlee** August 23, 2017 at 6:56 am #

This post is just about the framing of the problem.

Moving average is something to do to the data once it is framed.



**pkl520** August 26, 2017 at 10:29 pm #

Hi , Jason! Good article as always~

I have a question.

“Running the example shows the differentiation of input (t-n) and output (t+n) variables with the current observation (t) considered an output.”

```
values = [x for x in range(10)]
data = series_to_supervised(values, 2, 2)
print(data)
```

```
var1(t-2) var1(t-1) var1(t) var1(t+1)
2 0.0 1.0 2 3.0
3 1.0 2.0 3 4.0
4 2.0 3.0 4 5.0
5 3.0 4.0 5 6.0
6 4.0 5.0 6 7.0
7 5.0 6.0 7 8.0
8 6.0 7.0 8 9.0
```

So above example, var1(t-2) var1(t-1) are input , var1(t) var1(t+1) are output, am I right?

Then,below example.

```
raw = DataFrame()
raw[‘ob1’] = [x for x in range(10)]
raw[‘ob2’] = [x for x in range(50, 60)]
values = raw.values
data = series_to_supervised(values, 1, 2)
print(data)
```

Running the example shows the large reframed DataFrame.

var1(t-1) var2(t-1) var1(t) var2(t) var1(t+1) var2(t+1)
1 0.0 50.0 1 51 2.0 52.0
2 1.0 51.0 2 52 3.0 53.0
3 2.0 52.0 3 53 4.0 54.0
4 3.0 53.0 4 54 5.0 55.0
5 4.0 54.0 5 55 6.0 56.0
6 5.0 55.0 6 56 7.0 57.0
7 6.0 56.0 7 57 8.0 58.0
8 7.0 57.0 8 58 9.0 59.0

var1(t-1) var2(t-1) are input, var1(t) var2(t) var1(t+1) var2(t+1) are output.

can u answer my question? I will be very appreciate!



**Jason Brownlee** August 27, 2017 at 5:48 am #

REPLY

Yes, or you can interpret and use the columns any way you wish.

**Thabet** August 30, 2017 at 7:36 am #

Thank you Jason!!
You are the best teacher ever

**Jason Brownlee** August 30, 2017 at 4:17 pm #

Thanks Thabet.

**Charles** September 29, 2017 at 12:24 am #

Jason,

I love your articles! Keep it up! I have a generalization question. In this data set:

var1(t-1) var2(t-1) var1(t) var2(t)
1 0.0 50.0 1 51
2 1.0 51.0 2 52
3 2.0 52.0 3 53
4 3.0 53.0 4 54
5 4.0 54.0 5 55
6 5.0 55.0 6 56
7 6.0 56.0 7 57
8 7.0 57.0 8 58
9 8.0 58.0 9 59

If I was trying to predict var2(t) from the other 3 data, would the input data X shape would be (9,1,3) and the target data Y would be (9,1)? To generalize, what if this was just one instance of multiple time series that I wanted to use. Say I have 1000 instances of time series. Would my data X have the shape (1000,9,3)? And the input target set Y would have shape (1000,9)?

Is my reasoning off? Am I framing my problem the wrong way?

Thanks!
Charles



**Jason Brownlee** September 29, 2017 at 5:06 am #

This post provides help regarding how to reshape data for LSTMs:
https://machinelearningmastery.com/reshape-input-data-long-short-term-memory-networks-keras/

**Sean Maloney** October 1, 2017 at 5:24 pm #

Hi Jason!

I'm really struggling to make a new prediction once the model has been build. Could you give an example? I've been trying to write a method that takes the past time data and returns the yhat for the next time.

Thanks you.



**Jason Brownlee** October 2, 2017 at 9:37 am #

REPLY

Yes, see this post:
https://machinelearningmastery.com/make-sample-forecasts-arima-python/

**Sean Maloney** October 1, 2017 at 5:28 pm #

REPLY

P.S. I'm the most stuck at how to scale the new input values.

**Jason Brownlee** October 2, 2017 at 9:38 am #

REPLY

Any data transforms performed on training data must be performed on new data for which y

## Leave a Reply

Get Your Start in Machine Learning

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

**Get Good at Time Series Forecasting**

Need visualizations and forecast models?
Looking for step-by-step tutorials?

## Get Your Start in Machine Learning

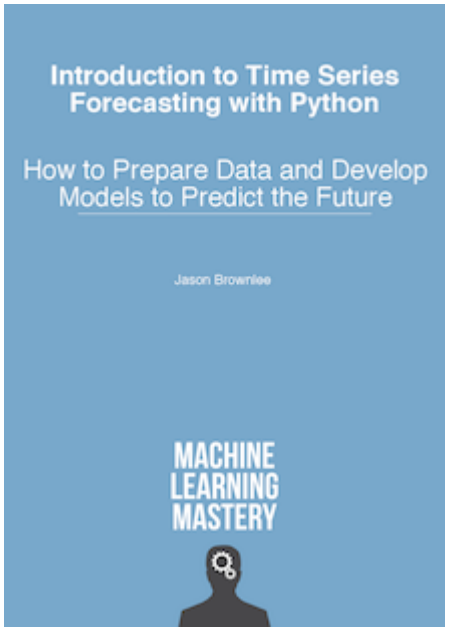
You can master applied Machine Learning **without the math or fancy degree**.
Find out how in this *free* and *practical* email course.

Email Address

START MY EMAIL COURSE

Want end-to-end projects?

Get Started with Time Series Forecasting in Python!




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

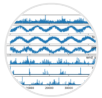

**Sequence Classification with LSTM Recurrent Neural Networks in Python with Keras**
JULY 26, 2016

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

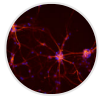
**Time Series Forecasting with the Long Short-Term Memory Network in Python**
APRIL 7, 2017

**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

Get Your Start in Machine Learning



Privacy | Contact | About

## Get Your Start in Machine Learning

You can master applied Machine Learning **without the math or fancy degree**.
Find out how in this *free* and *practical* email course.

Email Address

START MY EMAIL COURSE