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.

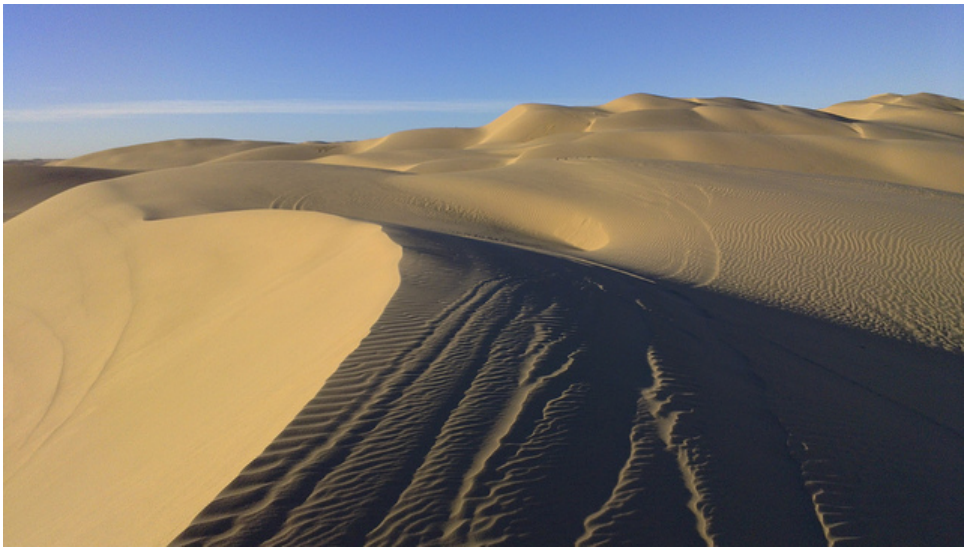
Before machine learning can be used, time series forecasting problems must be re-framed as supervised learning problems. From a sequence to pairs of input and output sequences.

In this tutorial, you will discover how to transform univariate and multivariate time series forecasting problems into supervised learning problems for use 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 in Python
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

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

For example:

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

A supervised learning problem is comprised of input patterns (*X*) and output patterns (*y*), su
output patterns from the input patterns.

For example:

```
X,  y
1   2
2,  3
3,  4
4,  5
5,  6
6,  7
7,  8
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 shift() function.

Given a DataFrame, the *shift()* function can be used to create copies of columns that are pushed forward (rows of NaN values added to the front) or 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 observations for a time series dataset in a supervised 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 and a new shifted column.

We can see that shifting the series forward one time step gives us a primitive supervised learning problem, although with *X* and *y* in the wrong order. Ignore the column of row labels. The first row would have to be discarded because of the NaN value. The second row shows the input value of 0.0 in the 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
3   3  2.0
4   4  3.0
5   5  4.0
6   6  5.0
7   7  6.0
8   8  7.0
9   9  8.0
```

We can see that if we can repeat this process with shifts of 2, 3, and more, how we could c… forecast an output value (*y*).

The shift operator can also accept a negative integer value. This has the effect of pulling th… 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.

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

Technically, in time series forecasting terminology the current time (*t*) and future times (*t+1*, *t+n*) are forecast times and past observations (*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 sequences.

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 tak… …es it as a supervised learning dataset.

Your Start in Machine Learning

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. Defaults to 1.
- **n_out**: Number of observations as output (*y*). Values may be between [0..len(data)-1]. Optional. Defaults to 1.
- **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 number and time step. This allows you to design a variety of different time step sequence type forecasting problems from a given univ

Once the DataFrame is returned, you can decide how to split the rows of the returned Data learning any way you wish.

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

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 done by specifying the length of the input sequence 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
variable to 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 observations.

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
```

```
36
37
38 values = [x for x in range(10)]
39 data = series_to_supervised(values, 2, 2)
40 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.

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

For example, we may have two sets of time series observations obs1 and obs2 and we wis

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

For example:


```
1  from pandas import DataFrame
2  from pandas import concat
3
4  def series_to_supervised(data, n_in=1, n_out=1, dropnan=True):
5      """
6      Frame a time series as a supervised learning dataset.
7      Arguments:
8          data: Sequence of observations as a list or NumPy array.
9          n_in: Number of lag observations as input (X).
10         n_out: Number of observations as output (y).
11         dropnan: Boolean whether or not to drop rows with NaN values.
12     Returns:
13         Pandas DataFrame of series framed for supervised learning.
14     """
15     n_vars = 1 if type(data) is list else data.shape[1]
16     df = DataFrame(data)
17     cols, names = list(), list()
18     # input sequence (t-n, ... t-1)
19     for i in range(n_in, 0, -1):
20         cols.append(df.shift(i))
21         names += [('var%d(t-%d)' % (j+1, i)) for j in range(n_vars)]
22     # forecast sequence (t, t+1, ... t+n)
23     for i in range(0, n_out):
24         cols.append(df.shift(-i))
25         if i == 0:
26             names += [('var%d(t)' % (j+1)) for j in range(n_vars)]
27         else:
28             names += [('var%d(t+%d)' % (j+1, i)) for j in range(n_vars)]
29     # put it all together
30     agg = concat(cols, axis=1)
31     agg.columns = names
32     # drop rows with NaN values
33     if dropnan:
34         agg.dropna(inplace=True)
35     return agg
36
37
38 raw = DataFrame()
39 raw['ob1'] = [x for x in range(10)]
40 raw['ob2'] = [x for x in range(50, 60)]
41 values = raw.values
42 data = series_to_supervised(values)
43 print(data)
```

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

Again, depending on the specifics of the problem, the division of columns into *X* and *Y* components can be chosen arbitrarily, such as if the current 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
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
```

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 with Python.

Specifically, you learned:

- About the Pandas *shift()* function and how it can be used to automatically define super

- How to reframe a univariate time series into one-step and multi-step supervised learning problems.
- How to reframe multivariate time series into one-step and multi-step supervised learning problems.

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




About Jason Brownlee

Jason Brownlee, Ph.D. 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 Use Weight Regularization with LSTM Networks for Time Series Forecasting

Multi-step Time Series Forecasting with Long Short-Term Memory Networks in Python ›

119 Responses to *How to Convert a Time Series to a Supervised Learning Problem in Python*

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 "savers" or "spenders"

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

Yes, it is transposing each variable, but allowing control over the length of each ro

Mostafa March 2, 2018 at 2:43 am #

Hi Jason, thanks for very helpful tutorials, I have the same question as Mikkel

how would you then characterise following data?

let's suppose we have a dataset same as the following.
and we want to predict the Balance of each Customer at the fourth month, how should

Thanks a bunch in advance

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

Jason Brownlee March 2, 2018 at 5:35 am #

REPLY

Test different framing of the problem.

Try modeling all customers together as a first step.

Yavuz June 21, 2018 at 11:08 pm #

REPLY

Hi Mostafa,

I am dealing with a similar kind of problem right now. Have you found any simple and coherent answer to your question? Any article, code example or video lecture?

I appreciate if you found something and let me know.

Thanks, regards.

WangGang June 25, 2018 at 10:16 pm #

I would like to ask if I have the data for the first 5 hours, how to get the data for the sixth hour, Thanks

Daniel May 9, 2017 at 4:56 pm #

REPLY

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 #

REPLY

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 time series data. Also, no need to repeat the function code 5 times 😉 Gripes aside, this was very timely as I'm just about to get into some time series forecasting, so thanks for this article!!!

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

REPLY

Thanks for the suggestion Lee.

Christopher May 12, 2017 at 9:16 pm #

REPLY

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 forecasting? Which machine learning method is the most suitable for that?
Obviously a regressor is the best choice but how can I determine the size of the sliding windo

Thanks a lot for your help and work
~ Christopher

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

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-autocorrelati

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 series .Do you mean these time series can make a 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 #

REPLY

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 problems and apply a suite of standard classification 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 #

REPLY

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

Brad June 25, 2017 at 4:47 pm #

REPLY

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

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 #

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 #

REPLY

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

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 #

REPLY

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 #

Thanks

Should I then use the lagged versions of the predictors?

Kushal

REPLY

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

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

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

great article, Jason!

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

Thanks, I hope it helps.

Chinesh August 10, 2017 at 5:15 pm #

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

REPLY

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

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

REPLY

Hossein August 23, 2017 at 1:16 am #

Great article Jason. Just a naive question: How does this method different from moving average smoothing? I'm a bit confused!
Thanks

REPLY

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.

REPLY

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 #

REPLY

Thank you Jason!!
You are the best teacher ever

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

REPLY

Thanks Thabet.

Charles September 29, 2017 at 12:24 am #

REPLY

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

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 you want to make a prediction.

Nish October 23, 2017 at 11:42 am #

REPLY

Hi Jason,
This is great, but what if I have around ten features (say 4 categorical and 6 continuous), a couple of thousand data points per day, around 200 days worth of data in my training set? The shift function could work in theory but you'd be adding hundreds of thousands of columns, which would be computationally horrendous.
In such situations, what is the recommended approach?

Jason Brownlee October 23, 2017 at 4:11 pm #

Yes, you will get a lot of columns.

REPLY

Shud November 1, 2017 at 5:37 pm #

Hey Jason,

I converted my time series problem into regression problem and i used GradientBoostingRegressor to model the data. I see my adjusted R-squared keep changing everytime i run the model. I believe this is because of the correlation that exists between the independent variable (lag variables). How to handle this scenario? Though the range of fluctuation is small, i am concerned that this might be a bad model

REPLY

Jason Brownlee November 2, 2017 at 5:08 am #

It may be because of the stochastic nature of the algorithm:
https://machinelearningmastery.com/randomness-in-machine-learning/

Nitin Gupta November 13, 2017 at 10:10 pm #

Hey Jason,

I applied the concept that you have explained to my data and used linear regression. Can I exp
squaring the t-1 terms?

Jason Brownlee November 14, 2017 at 10:12 am #

Sure, let me know how you go.

REPLY

Samuel November 15, 2017 at 9:43 pm #

Hey Jason,

thanks a lot for your article! I already read a lot of your articles. These articles are great, they really helped me a lot.

But I still have a rather general question, that I can't seem to wrap my head around.

The question is basically:
In which case do I treat a supervised learning problem as a time series problem, or vice versa?

For further insight, this is my problem I am currently struggling with:
I have data out of a factory (hundreds of features), which I can use as my input.
Additionally I have the energy demand of the factory as my output.
So I already have a lot of input-output-pairs.
The energy demand of the factory is also the quantity I want to predict.
Each data point has its own timestamp.
I can transform the timestamp into several features to take trends and seasonality into account.
Subsequently I can use different regression models to predict the energy demand of the factory.
This would then be a classical supervised regression problem.

But as I unterstood it from your time series articles, I could as well treat the same problem as a time series problem.
I could use the timestamp to extract time values which I can use in multivariate time series forecasting.

In most examples you gave in your time series articles, you had the output over time.
https://machinelearningmastery.com/reframe-time-series-forecasting-problem/
And in this article you shifted the time series to get an input, in order to treat the problem as a supervised learning problem.

So let's suppose you have the same number of features in both cases.
Is it a promising solution to change the supervised learning problem to a time series problem?

REPLY

What would be the benefits and drawbacks of doing this?

As most regression outputs are over time.
Is there a general rule, when to use which framing(supervised or time series) of the problem?

I hope, that I could phrase my confusion in an ordered fashion.

Thanks a lot for your time and help, I really appreciate it!

Cheers Samuel

Jason Brownlee November 16, 2017 at 10:29 am #

REPLY

To use supervised learning algorithms you must represent your time series as a s

Not sure I follow what you mean by tuning a supervised learning problem into a series?

Samuel November 29, 2017 at 10:19 pm #

Dear Jason,

thank you for your fast answer.
I'm sorry that I couldn't frame my question comprehensibly, I'm still new to ML.
I'll try to explain what I mean with an example.

Let's suppose you have the following data, I adapted it from your article:
https://machinelearningmastery.com/time-series-forecasting-supervised-learning/

input1(time), input2, output
1, 0.2, 88
2, 0.5, 89
3, 0.7, 87
4, 0.4, 88
5, 1.0, 90

This data is, what you would consider a time series. But as you already have 2 inputs and 1 output you could already use the data for supervised machine learning.
In order to predict future outputs of the data you would have to know input 1 and 2 at timestep 6. Let's assume you know from your production plan in a factory that the input2 will have a value of 0.8 at timestep 6 (input1). With this data you could gain y_pred from your model. You would have treated the data purely as a supervised machine learning problem.

input1(time), input2, output
1, 0.2, 88
2, 0.5, 89
3, 0.7, 87
4, 0.4, 88
5, 1.0, 90
6, 0.8, y_pred

But you could do time-series forecasting with the same data as well, if I understood your articles correctly.

input1(time), input2, output
nan, nan, 88
1, 0.2, 89
2, 0.5, 87
3, 0.7, 88
4, 0.4, 90
5, 1.0, y_pred

This leads to my questions:

In which case do I treat the data as a supervised learning problem and in which case as a time series problem?
Is it a promising solution to change the supervised learning problem to a time series problem?
What would be the benefits and drawbacks of doing this?
As my regression outputs are over time.
Is there a general rule, when to use which framing (supervised or time series) of the pr

I hope, that I stated my questions more clearly.

Thanks a lot in advance for your help!

Best regards Samuel

Jason Brownlee November 30, 2017 at 8:16 am #

REPLY

I follow your first case mostly, but time would not be an input, it would be removed and assumed. I do not follow your second case.

I believe it would be:

```
output(t) = f(input2(t), output(t-1))
```

What is best for your specific data, I have no idea. Try a suite of different framings models give the best skill on your problem. That is the only trade-off to consider.

MJ November 18, 2017 at 12:46 am #

Ver helpful, thanks!

Jason Brownlee November 18, 2017 at 10:19 am #

Glad to hear it.

Michael November 30, 2017 at 6:47 am #

REPLY

Jason:
Thank you for all the time and effort you have expended to share your knowledge of Deep Learning, Neural Networks, etc. Nice work.

I have altered your series_to_supervised function in several ways which might be helpfut to other novices:
(1) the returned column names are based on the original data
(2) the current period data is always included so that leading and lagging period counts can be 0.
(3) the selLag and selFut arguments can limit the subset of columns that are shifted.

There is a simple set of test code at the bottom of this listing:

```
from pandas import DataFrame
from pandas import concat
import random

def time_series_to_supervised(data, n_lag=1, n_fut=1, selLag=None, selFut=None, dropnan=True):
    """
    Converts a time series to a supervised learning data set by adding time-shifted prior and future period
    data as input or output (i.e., target result) columns for each period
    :param data:  a series of periodic attributes as a list or NumPy array
    :param n_lag: number of PRIOR periods to lag as input (X); generates: Xa(t-1), Xa(t-2); min= 0 --> nothing lagged
    :param n_fut: number of FUTURE periods to add as target output (y); generates Yout(t+1); min= 0 --> no future periods
    :param selLag:  only copy these specific PRIOR period attributes; default= None; EX: ['Xa', 'Xb' ]
    :param selFut:  only copy these specific FUTURE period attributes; default= None; EX: ['rslt', 'xx']
    :param dropnan: True= drop rows with NaN values; default= True
    :return: a Pandas DataFrame of time series data organized for supervised learning
    NOTES:
    (1) The current period's data is always included in the output.
    (2) A suffix is added to the original column names to indicate a relative time reference: e.g., (t) is the current
        period; (t-2) is from two periods in the past; (t+1) is from the next period
    (3) This is an extension of Jason Brownlee's series_to_supervised() function, customized for MFI use
    """
    n_vars = 1 if type(data) is list else data.shape[1]
    df = DataFrame(data)
    origNames = df.columns
    cols, names = list(), list()
    # include all current period attributes
    cols.append(df.shift(0))
    names += [('%s' % origNames[j]) for j in range(n_vars)]

```

```
    # lag any past period attributes (t-n_lag,...,t-1)
    n_lag = max(0, n_lag)  # force valid number of lag periods
    for i in range(n_lag, 0, -1):
        suffix= '(t-%d)' % i
        if (None == selLag):   # copy all attributes from PRIOR periods?
            cols.append(df.shift(i))
            names += [('%s%s' % (origNames[j], suffix)) for j in range(n_vars)]
        else:
            for var in (selLag):
                cols.append(df[var].shift(i))
                names+= [('%s%s' % (var, suffix))]

    # include future period attributes (t+1,...,t+n_fut)
    n_fut = max(n_fut, 0)  # force valid number of future periods to shift back
    for i in range(1, n_fut + 1):
        suffix= '(t+%d)' % i
        if (None == selFut):  # copy all attributes from future periods?
            cols.append(df.shift(-i))
            names += [('%s%s' % (origNames[j], suffix)) for j in range(n_vars
        else:  # copy only selected future attributes
            for var in (selFut):
                cols.append(df[var].shift(-i))
                names += [('%s%s' % (var, suffix))]
    # combine everything
    agg = concat(cols, axis=1)
    agg.columns = names
    # drop rows with NaN values introduced by lagging
    if dropnan:
        agg.dropna(inplace=True)
    return agg


"""
def main():    # simple test example
    df= DataFrame()
    df['xa']= [x for x in range(10)]
    df['xb']= [2*x for x in range(10)]
    df['rslt']= random.sample(range(100),10)
    print(df)
    dfx= time_series_to_supervised(df, n_lag= 1, n_fut= 2, dropnan= False, selLag= ['xa'], selFut=['rslt'])
    print (dfx)


if __name__ ==  '__main__':
    main()

"""
```


Jason Brownlee November 30, 2017 at 8:30 am #

REPLY

Very cool Michael, thanks for sharing!

Maciej December 1, 2017 at 7:11 am #

REPLY

When I do forecasting, let's say only one step ahead, as the first input value I should use any value that belongs i.e. to validation data (in order to set up initial state of forecasting). In second, third and so on prediction step I should use previous output of forecasting as input of NN. Do I understand correctly ?

Jason Brownlee December 1, 2017 at 7:46 am #

REPLY

I think so.

Maciej December 2, 2017 at 4:15 am #

REPLY

Ok, so another question. In the blog post here: https://machinelearningmastery.com/time-series-forecasting-long-short-term-memory-network-python/, as an input for NN you use test values. The predictions are only saved to a list and they are not used to predict further values of timeseries.

My question is. Is it possible to predict a series of values knowing only the first value ?
For example. I train a network to predict values of sine wave. Is it possible to predict next N values of sine wave starting from value zero and feeding NN with result of prediction to predict t + 1, t + 2 etc ?

Maciej December 2, 2017 at 4:18 am #

REPLY

If my above understanding is incorrect then it means that if your test values are completely different than those which were used to train network, we will get even worse predictions.

Jason Brownlee December 2, 2017 at 9:05 am #

Yes. Bad predictions in a recursive model will give even worse subse

Ideally, you want to get ground truth values as inputs.

Jason Brownlee December 2, 2017 at 9:04 am #

Yes, this is called multi-step forecasting. Here is an example:
https://machinelearningmastery.com/multi-step-time-series-forecasting-long-short-

Maciej December 3, 2017 at 5:34 am #

Does it mean that using multi-step forecast (let's say I will predict 4 values) I can predict a timeseries which contains 100 samples providing only initial step (for example providing only first two values of the timeseries) ?

Jason Brownlee December 4, 2017 at 7:40 am #

Yes, but I would expect the skill to be poor – it's a very hard problem to predict so many time steps from so little information.

Liz January 12, 2018 at 7:02 am #

REPLY

Hello Mr. Brownlee,

thank you for all of your nice tutorials. They really help!
I have two questions about the input data for an LSTM for multi-step predictions.
1. If I have multiple features that I use as input for the prediction and at a point (t) I have no new values for any of them. Do I have to predict all my input features in order to make make a multi-step forecast?
2. If some of my input data is binary data and not continuous can I still predict it with the same LSTM? Or do I need a separate Classification?

Sorry if its very basic, I am quite new to LSTM.
Best regards Liz

Jason Brownlee January 12, 2018 at 11:49 am #

REPLY

No, you can use whatever inputs you choose.

Sure you can have binary inputs.

Liz January 13, 2018 at 1:18 am #

Thank you for your quick answer.
Unfortunately I still have some trouble with the implementation.
If I use feature_1 and feature_2 as input for my my LSTM but only predict feature_1 at time (t+1) how do I make the next step to know feature_1 at time (t+2).
Somehow I seem to miss feature_2 at time (t+1) for this approach.
Could you tell me where I am off?
Best regards Liz

Jason Brownlee January 13, 2018 at 5:34 am #

REPLY

Perhaps double check your input data is complete?

strawberry lv January 31, 2018 at 6:41 pm #

Hello,thank you for the article and i have learned a lot from it.
Now i have a question about it.
The method can be understood as using the value before to forecast the next value. If i need t
use the model to first forecat the value at t + 1, and then using the value to forecast t+ 2, then,
or do you have any aother methed

Arslan Ahmed March 17, 2018 at 9:01 am #

Hi,
I am working on energy consumption data and I have the same question. Did you get to know any efficient method to forecast the value at t+1, t+2, t+3 + …… t+N?

Sameer January 31, 2018 at 11:05 pm #

REPLY

Hello Dr.Brownlee,

I'm planning to purchase your Introduction to Time series forecasting book. I just want to know that if you've covered Multivariate cum multistep LSTM

Jason Brownlee February 1, 2018 at 7:22 am #

REPLY

No, only univariate.

Victor February 21, 2018 at 5:10 am #

REPLY

Hi Jason,

Thanks for the article. I have a question about going back n periods in terms of choosing the features. If I have a feature and make for example 5 new features based off of some lag time, my new features are all very highly correlated (between 0.7 and 0.95). My model is resulting in training score of 1 and test score of 0.99. I'm concerned that there is an issue with multicollinearity between all the lag features that is causing my model to overfit. Is this a legitimate concern and how could I go about fixing it if so? Thanks!

Jason Brownlee February 21, 2018 at 6:42 am #

REPLY

Try removing correlated features, train a new model and compare model skill.

Ram Seshadri February 21, 2018 at 12:06 pm #

REPLY

Dear Jason:

My sincere thanks for all you do. Your blogs were very helpful when I started on the ML journey.

I read this blog post for a Time Series problem I was working on. While I liked the “series_to_supervised” function, I typically use data frames to store and retrieve data while working in ML. Hence, I thought I would modify the code to send in a dataframe and get back a dataframe with just the new columns added. Please take a look at my revised code.

```
def ts_dataframe_to_supervised(df, target, n_in=1, n_out=1, dropT=True):
    """
    Transform a time series dataframe into a supervised learning dataset.
    Arguments:
        df: a dataframe.
        target: this is the target variable you intend to use in supervised l
        n_in: Number of lag observations as input (X).
        n_out: Number of observations as output (y).
        dropT: Boolean - whether or not to drop columns at time "t".
    Returns:
        Pandas DataFrame of series framed for supervised learning.
    """
    namevars = df.columns.tolist()
    # input sequence (t-n, ... t-1)
    drops = []
    for i in range(n_in, -1, -1):
        if i == 0:
            for var in namevars:
                addname = var+'(t)'
                df.rename(columns={var:addname},inplace=True)
                drops.append(addname)
        else:
            for var in namevars:
                addname = var+'(t-'+str(i)+')'
                df[addname] = df[var].shift(i)
    # forecast sequence (t, t+1, ... t+n)
    if n_out == 0:
        n_out = False
    for i in range(1, n_out):
        for var in namevars:
            addname = var+'(t+'+str(i)+')'
            df[addname] = df[var].shift(-i)
    # drop rows with NaN values
    df.dropna(inplace=True,axis=0)
    # put it all together
    target = target+'(t)'
    if dropT:
        drops.remove(target)
        df.drop(drops, axis=1, inplace=True)
    preds = [x for x in list(df) if x not in [target]]
    return df, target, preds
```


Usage:

```
df, target,preds = ts_dataframe_to_supervised(df, target, 1, 0, False)
```

Please take a look and let me know. Hope this helps others,
Ram

Jason Brownlee February 22, 2018 at 11:14 am #

REPLY

Very cool Ram, thanks for sharing!

Marius Terblanche February 26, 2018 at 11:33 pm #

REPLY

Dear Jason,
great article, as always!
May I ask a question, please?
Once the time series data (say for multi-step, univariate forecasting) have been prepared using code described above, is it then ready (and in the 3D structure) required for feeding into the first hidden layer of a LSTM RNN?
May be dumb question!

Many thanks in advance.
Marius.

REPLY

Jason Brownlee February 27, 2018 at 6:30 am #

It really depends on your data and your framing of the problem.

The output will be 2D, you may still need to make it 3D. See this post:
https://machinelearningmastery.com/prepare-univariate-time-series-data-long-short-term-memory-networks/

And this post:
https://machinelearningmastery.com/reshape-input-data-long-short-term-memory-networks

MikeF March 7, 2018 at 12:49 pm #

Hi Jason, thanks for this post. Its simple enough to understand. However, after conve
are from the future and won't be available when trying to make predictions. How do you sugge

Jason Brownlee March 7, 2018 at 3:08 pm #

Perhaps remove those features?

Adarsh March 27, 2018 at 3:11 pm #

REPLY

i have a dataset liike this

accno dateofvisit
12345 12/05/15 9:00:00
123345 13/06/15 13:00:00
12345 12/05/15 13:00:00

how will i forecast when that customer will visit again

Jason Brownlee March 27, 2018 at 4:20 pm #

REPLY

You can get started with time series forecasting here:
https://machinelearningmastery.com/start-here/#timeseries

Fatima April 10, 2018 at 6:32 pm #

REPLY

Hi,

I need to develop input vector which uses every 30 minutes prior to time t for example:

input vector is like (t-30,t-60,t-90,…,t-240) to predict t.

If I wanna use your function for my task, Is it correct to change the shift function to df.shift(3*i) ?

Thanks

Jason Brownlee April 11, 2018 at 6:33 am #

REPLY

One approach might be to selectively retrieve/remove columns after the transform

fatima April 12, 2018 at 7:14 pm #

REPLY

Hi,

So I should take these steps:

1- transform for different lags
2-select column related to first lag (for example 30min(
3- transform for other lags
4- concatenate along axis=1

When I perform such steps seems the result is equivalent to when I shift by 3?
I have some questions
which one is better to use?(Shift by 3 or do above steps)
should I remove time t after each transform and just keep time t for last lag?

Thanks

Jason Brownlee April 13, 2018 at 6:37 am #

Use an approach that you feel makes the most sense for your problem.

vishwas April 16, 2018 at 3:15 pm #

Hi Jason,

Amazing article for creating supervised series. But I have a doubt,
Suppose If I wanted the predict sales for next 14 days using Daily sales historical data. Would that require me too take 14 lags to predict the next 14 days??
Ex: (t-14, t-13 …..t-1) to predict (t,t+1,t+2,t+14)

Jason Brownlee April 17, 2018 at 5:53 am #

REPLY

No, the in/out obs are separate. You could have 1 input and 14 outputs if you really wanted.

Vishwas April 17, 2018 at 3:31 pm #

REPLY

Thanks for the quick response Jason!!

Sanketh Nagarajan April 17, 2018 at 8:37 am #

REPLY

Hi Jason,

I want to predict if the next value will be higher or lower than the previous value. Can I use the same method to frame it as a classification problem? For example:

V(t) class

0.2 0
0.3 1
0.1 0
0.5 0
2.0 1
1.5 0

where class zero represents a decrease and class 1 represents an increase?

Thanks,
Sanketh

Jason Brownlee April 17, 2018 at 2:48 pm #

REPLY

Sure.

brandon May 7, 2018 at 11:17 pm #

REPLY

Hi Jason, really nice explanations in your blog. When I have the shape e.g. (180,20) c
original data back with shape (200,1) back ?

Jason Brownlee May 8, 2018 at 6:14 am #

You will have to write custom code to reverse the transform.

Farooq Arshad May 8, 2018 at 8:23 pm #

Hi Jason,

Amazing article.
I have a question, Suppose I want to move the window by 24 steps instead of just one step, what modifications do I have to do in this case? Like i have Energy data with one hour interval and I want to predict next 24 hours (1 day) looking at last 21 days (504 hours) then for the next prediction i want to move window by 24 hours (1 day).

Jason Brownlee May 9, 2018 at 6:23 am #

REPLY

Perhaps re-read the description of the function to understand what arguments to provide to the function.

Alex May 19, 2018 at 5:43 am #

REPLY

Models blindly fit on data specified like this are guaranteed to overfit.

Suppose you estimate model performance with a cross-validation procedure and you have folds:

Fold1 (January)
Fold2 (February)
Fold3 (March)
Fold4 (April)

Consider a model fit on folds 1, 2 and 4. Now you predicting some feature for March based on the value of that feature in April!

If you choose to use a lagged regressor matrix like this, please please please look into appropriate model validation.

One good reason is Hyndman's textbook, available freely online: https://otexts.org/fpp2/accuracy.html

Jason Brownlee May 19, 2018 at 7:46 am #

REPLY

Indeed, one should never use cross validation for time series.

Instead, one would use walk-forward validation:
https://machinelearningmastery.com/backtest-machine-learning-models-time-series-forecasting/

marc May 23, 2018 at 10:28 am #

REPLY

Hi Jason, really nice blog and learned much from you. I implement one LSTM encoder decoder with sliding windows. The prediction was nearly the same as the input, is it usual that this happens on sliding windows ? I am a bit surprised because the model saw only in the training a little part of data and the model later predicted almost the same input. That makes me thinking I might be wrong. I do not want to post the coding it ist just standard lstm encoder decoder code, but the fact that the model saw only a little part of the data in training is confusing me.

Jason Brownlee May 23, 2018 at 2:39 pm #

REPLY

It sounds like the model learned a persistence forecast, it might suggest further tuning is required. I outline more on this here:
https://machinelearningmastery.com/faq/single-faq/why-is-my-forecasted-time-series-right-

james May 23, 2018 at 1:15 pm #

HI json，it's so good your code,but i have a question that i change the window size(re
reframed = series_to_supervised(scaled, 2, 1)),then i get bad prediction,how can i solve or wh
Please take a look at my revised code.

```
from math import sqrt
from numpy import concatenate
from matplotlib import pyplot
from pandas import read_csv
from pandas import DataFrame
from pandas import concat
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import mean_squared_error
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
# convert series to supervised learning
def series_to_supervised(data, n_in=1, n_out=1, dropnan=True):
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
# load dataset
dataset = read_csv('pollution.csv', header=0, index_col=0)
values = dataset.values

# integer encode direction
```

```
encoder = LabelEncoder()
values[:,4] = encoder.fit_transform(values[:,4])

# ensure all data is float
values = values.astype('float32')

# normalize features
scaler = MinMaxScaler(feature_range=(0, 1))
scaled = scaler.fit_transform(values)

# frame as supervised learning
reframed = series_to_supervised(scaled, 2, 1)
# drop columns we don't want to predict
reframed.drop(reframed.columns[[9,10,11,12,13,14,15]], axis=1, inplace=True)
print(reframed.head())

# split into train and test sets
values = reframed.values
n_train_hours = 365*24
train = values[:n_train_hours, :]
test = values[n_train_hours:, :]

# split into input and outputs
train_X, train_y = train[:, :-1], train[:, -1]
test_X, test_y = test[:, :-1], test[:, -1]

# reshape input to be 3D [samples, timesteps, features]
train_X = train_X.reshape((train_X.shape[0], 1, train_X.shape[1]))
test_X = test_X.reshape((test_X.shape[0], 1, test_X.shape[1]))
print(train_X.shape, train_y.shape, test_X.shape, test_y.shape)

# design network
model = Sequential()
model.add(LSTM(50, input_shape=(train_X.shape[1], train_X.shape[2])))
model.add(Dense(1))
model.compile(loss='mae', optimizer='adam')
# fit network
history = model.fit(train_X, train_y, epochs=50, batch_size=72, validation_data=(test_X, test_y), verbose=2, shuffle=False)
# plot history
pyplot.plot(history.history['loss'], label='train')
pyplot.plot(history.history['val_loss'], label='test')
pyplot.legend()
pyplot.show()
# make a prediction
yhat = model.predict(test_X)
test_X = test_X.reshape((test_X.shape[0], test_X.shape[2]))
# invert scaling for forecast
inv_yhat = concatenate((yhat, test_X[:, 1:8]), axis=1)
inv_yhat = scaler.inverse_transform(inv_yhat)
inv_yhat = inv_yhat[:,0]
# invert scaling for actual
inv_y = scaler.inverse_transform(test_X[:,:8])
inv_y = inv_y[:,0]
# calculate RMSE
rmse = sqrt(mean_squared_error(inv_y, inv_yhat))
print('Test RMSE: %.3f' % rmse)
# plot prediction and actual
pyplot.plot(inv_yhat[:100], label='prediction')
pyplot.plot(inv_y[:100], label='actual')
pyplot.legend()
pyplot.show()
```


Jason Brownlee May 23, 2018 at 2:40 pm #

The model may require further tuning for the change in problem.

REPLY

james May 23, 2018 at 4:51 pm #

I noticed that your code takes into account the effect of the last point in time on the current point in time.But this is not applicable in many cases. What are the optimization ideas?

REPLY

Jason Brownlee May 24, 2018 at 8:08 am #

Most approaches assume that the observation at t is a function of prior tir
case?

james May 24, 2018 at 11:45 am #

oh,maybe i don't describe my question clearly,my question is why just
you gave has poor performance

Jason Brownlee May 24, 2018 at 1:51 pm #

No good reason, just demonstration. You may change the model to include any set of features you wish.

REPLY

Ishrat Sarwar May 24, 2018 at 2:21 pm #

Dear Sir:
I have 70 input time series. I Only need to predict 1, 2 or 3 time series out of input(70 features) time series. Here are my questions.

-> Should I use LSTM for such problem?
-> Should I predict all 70 of time series?
-> If not LSTM then what approach should I use?

(Forex trading prediction problem)

REPLY

Jason Brownlee May 25, 2018 at 9:17 am #

Great questions!

– Try a suite of methods to see what works.
– Try different amounts of history and different numbers of forward time steps, find a sweet spot suitable for your project goals.
– Try classical ts methods, ml methods and dl methods.

REPLY

marc June 1, 2018 at 6:50 pm #

HI Jason, I have a huge data with small steps between data time series, they nearly change not in total till the last cycles. I thought maybe not only shifting by 1, how can I shiift more e.g. t-1 and t by 20 steps. Does also this make sense ?

REPLY

Jason Brownlee June 2, 2018 at 6:27 am #

Not sure I follow, sorry. Perhaps give a small example?

marc June 2, 2018 at 9:55 pm #

REPLY

lets say I have this data:

5
6
7
8
9
10
11
12

and usually if you make sliding windows, shifting them by 1 from t-2 to t

5 6 7
6 7 8
7 8 9
8 9 10
9 10 11
10 11 12
11 12
12

how can I do shifting not by 1 but maybe 3 looking at the first row (in this case) or more

5 8 11
6 9 12
7 10
8 11
9 12
10
11
12

I ask that because my data range is so small that shifting by 1 is not having much effect and thought maybe something like this could help. How do I have to adjust your codes for supervised learning to do that. And do you think this is a good idea ?

Jason Brownlee June 3, 2018 at 6:24 am #

REPLY

Specify the lag (n_in) as 3 to the function.

Bootstrap June 17, 2018 at 10:44 am #

REPLY

Hi Jason!

Once I apply this function to my data, what's the best way to split the data between train and test set?

Normally I would use sklearn train_test_split, which can shuffle the data and apply a split based on a user set proportion. However, intuitively, something tells me this incorrect, rather I would need to split the data based on the degree of shift(). Could you please clarify?

Jason Brownlee June 18, 2018 at 6:37 am #

REPLY

The best way depends on your specific prediction problem and your project goals.

This post has some suggestions for how to evaluate models on time series problems:
https://machinelearningmastery.com/backtest-machine-learning-models-time-series-forecasting/

brad June 27, 2018 at 5:15 am #

REPLY

When I give the function a sliding window of 20 series_to_supervised(values, 20) my new data shape is (None,21) none is variable here. Why do I get 21 ? Do i need to remove the last column ? or how do I move on ? thanks a lot for your posts.

Jason Brownlee June 27, 2018 at 8:22 am #

REPLY

I would guess 20 for the input 1 for the output.

Confirm this by printing the head() of the returned data frame.

lara June 28, 2018 at 7:54 am #

why we must convert it into a supervised learning for lstm problem ?

Jason Brownlee June 28, 2018 at 2:04 pm #

Because the LSTM is a supervised learning algorithm.

Leave a Reply

Name (required)

Email (will not be published) (required)

Website

SUBMIT COMMENT

Welcome to Machine Learning Mastery

Hi, I'm Jason Brownlee, Ph.D.
My goal is to make developers 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?
Want end-to-end projects?

Your Start in Machine Learning

POPULAR

You might also like…

- Your First Machine Learning Project in Python
- Your First Neural Network in Python
- How to Setup a Python for Machine Learning
- Your First Classifier in Weka
- Your First Model for Time Series Forecasting

Your Start in Machine Learning