**02. Exponential Smoothing Model**

## **What is Exponential Smoothing?**

Exponential smoothing of time series data assigns exponentially decreasing weights for newest to oldest observation. In other words, the older the data, the less priority (“weight”) the data is given; newer data is seen as more relevant and is assigned more weight. Smoothing parameters it is known as smoothing constants are usually denoted by α (Alpha) to determine the weights for observations.

Exponential smoothing is usually used to make short term forecasts, as longer-term forecasts using this technique can be quite unreliable.

* Simple (single) exponential smoothing uses a weighted moving average with exponentially decreasing weights.
* Holt’s trend-corrected double exponential smoothing is usually more reliable for handling data that shows trends compared to the single procedure.
* Triple exponential smoothing (also called the Multiplicative Holt-Winters) is usually more reliable for parabolic trends or data that shows trends and seasonality.

**Types of Exponential Smoothing :**

## **Simple Exponential Smoothing:** smoothing, which is suitable for a series with no trend or no seasonality.

## **Holt’s Exponential Smoothing:** smoothing, which is suitable for a series that has a trend but no seasonality.

## **Winter’s Exponential Smoothing:** smoothing which is suitable for series that have both trend and seasonality.

## **1. Simple Exponential Smoothing simply(SES):** makes the assumption that our series only contains level, no trend or seasonality, and it does include error.So if we only have level the assumption of the Exponential Smoother is that this level will stay put and not move. Therefore, the k-step-ahead SES forecast is simply the most recent estimate of our level at time, t.

|  |
| --- |
| **Forecast=estimated level at most recent time point**  **Ft+k = Lt** |

Series has only Level (Lt ) The level will “stay put” and noise - unpredictable.

**Estimating and updating the Level (Lt ) :**

To do this ,we are going to have to estimate the level. To do that we are going to use something called a Level updating Equation.

**The Level Updating Equation :**

|  |
| --- |
| **Lt = αyt + (1 – α) Lt-1** |

Where :

α: smoothing constant

0<=α<=1

In this equation we are talking the level at time, t and updating the previous level at time,t minus 1 by integrating information from our most recent data point ,Yt.You can see that it’s a weighted average where we have alpha and 1 minus alpha as our weights. Alpha is called the smoothing constant and it’s a number

Somewhere between 1 and 0. What this tells us is that the algorithm is learning the new level from the newest data that it’s seeing .

How do you start this whole system?

Well,you have to initialize it with L1 at some point .There are different ways of doing it. One option is simply setting L1 equal to the first record in your series ,Y1

**Possible Initialisation :**

|  |
| --- |
| F1=L1=Y1 |

Why is the algorithm called Exponential Smoothing?

To see this let’s rewrite the updating equation a little bit differently. We have the level updating equation as we wrote it before, but now let’s substitute L sub t minus 1 with its own formula.

Level Updating Equation :

|  |
| --- |
| **Lt = αyt + (1 – α) Lt-1** |

Substitute **Lt with its own formula :**

|  |
| --- |
| **Lt = αyt + (1 – α)[ αyt-1 + (1 – α)Lt-2=....**  **= αyt + α(1 – α)yt-1 + (1 – α)2Lt-2 =...**  **= αyt + α(1 – α)yt-1 + α (1 – α)2yt-2+....** |

So, we start with normal formula and then substitute L sub t minus 1 with its own formula that is based on L sub t minus 2.We can then do the same thing again and substitute Lt minus 2 with its predecessor..And if you write this out all the way down, you will end up with something that looks like this, alpha times Yt1 plus alpha times 1 minus alpha Yt minus 1 and so on and so forth

## The **basic formula** is:

|  |
| --- |
| St = αyt-1 + (1 – α) St-1 |

Where:

* α is the smoothing constant, a value from 0 to 1. If the value of α is close to zero, smoothing happens more slowly. Following this, the best value for α is the one that results in the smallest mean squared error (MSE). Various ways exist to do this, but a popular method is the Levenberg-Marquardt algorithm.
* t is time period.

**Many alternative formulas exist**. For example, Roberts (1959) replaced yt-1 with the current observation, yt. Another formula uses the forecast for the previous period and current period:

|  |
| --- |
| exponential smoothing formula |

Where:

* Ft – 1 is the forecast for the previous period,
* At – 1 is the Actual demand for the period,
* a is the weight (between 0 and 1). The closer to zero, the smaller the weight.

Which formula to use is usually a moot point, as most exponential smoothing is performed using the software. Whichever formula you use though, you’ll have to set an initial observation. This is a judgment call. You could use an averageof the first few observations, or you could set the second smoothed value equal to the original observation value to get the ball rolling.

**The Idea** : The idea behind simple Exponential Smoothing is to forecasting future values using a weighted average of all previous values in the series.

Uses : We can use this for forecasting a series that doesn’t have trend and doesn’t have seasonality.

**Advantages :**

The simple exponential smoothing is very popular because

1. it’s simple,
2. it’s adaptive and
3. it’s cheap to compute.

**Key Concept :**

smoothing constants

## **2. Double Exponential Smoothing**

This method is deemed more reliable for analyzing data that shows a trend. In addition, this is a more complicated method which adds a second equation to the procedure:

|  |
| --- |
| bt = γ(St – St-1) + (1 – γ)bt-1 |

Where:

* γ is a constant that is chosen with reference to α. Like α it can be chosen through the [Levenberg–Marquardt algorithm](http://people.duke.edu/~hpgavin/ce281/lm.pdf).

## **3. Triple Exponential Smoothing**

If your data shows a trend and seasonality, use triple exponential smoothing. In addition to the equations for single and double smoothing, a third equation is used to handle the seasonality aspect:

|  |
| --- |
| It = Β yt/St + (1-Β)It-L+m |

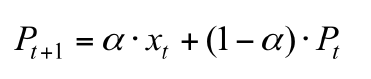
Where:

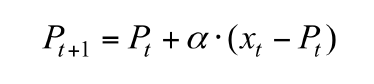
* y = observation,
* S = smoothed observation,
* b = trend factor,
* I = seasonal index,
* F = forecast m periods ahead,
* t = time period.

Like α and γ, the optimal Β minimizes the MSE.

## 

## **Exponential Smoothing algorithm theory**

This algorithm helps us to forecast new observations based on a time series. This algorithm uses smoothing methods. The exponential smoothing algorithm is used only on time series that DON'T have a trend. Exponential smoothing is based on the use of window functions to smooth time series data. The mathematical model for this algorithm is:

From this mathematical model it results:

where alpha is the smoothing constant and it can take values between 0 and 1, ,,P\_t'' is the forecast at time,,t'', X\_t is the time series observation at time ,,t''. In this case we have to solve two problems:

1. Choosing the P\_1 value;
2. Choosing the alpha value.

For the first problem we have two common options:

1. Choosing the P\_1 value to be equal to the first observation;
2. Choosing the P\_1 value to be equal to the arithmetic mean of the first four or five observations.

To solve the second problem we can use the ,,incremental method'': We set the value of alpha (0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9), then we calculate the corresponding MSE for each value that alpha takes. After that, we choose the alpha with the minimum MSE.

## **Exponential Smoothing implementation**

The following section will propose an algorithm for finding the best alpha. The algorithm will start at alpha = 0.1 and will go up to alpha = 0.9. For each alpha, the algorithm will forecast the already known observations along with the corresponding MSE followed by choosing the alpha with the minimum MSE value.

|  |
| --- |
| #Dataset description  '''Our time-series observations that will be used in this example are the  number of acres burned in forest fires in Canada over a period of 70 years.  We can see below how our time series looks like:''' |

|  |
| --- |
| import numpy as np  import pandas as pd  import matplotlib.pyplot as plt  csv\_dataset = pd.read\_csv("number\_of\_acres\_burned\_in\_forest.csv")  csv\_dataset.plot()  plt.show() |

Output:

|  |
| --- |
|  |

|  |
| --- |
| optimal\_alpha = None  best\_mse = None  db = csv\_dataset.iloc[:, :].values.astype('float32')  mean\_results\_for\_all\_possible\_alpha\_values = np.zeros(9)  for alpha in range(0, 9):  pt = np.mean(db[:, 0][0:5])  mean\_for\_alpha = np.zeros(len(db))  mean\_for\_alpha[0] = np.power(db[0][0] - pt, 2)  for i in range(1, len(db)):  pt = pt + ((alpha + 1) \* 0.1) \* (db[i - 1][0] - pt)  mean\_for\_alpha[i] = np.power(db[i][0] - pt, 2)  mean\_results\_for\_all\_possible\_alpha\_values[alpha] = np.mean(mean\_for\_alpha)  optimal\_alpha = (np.argmin(mean\_results\_for\_all\_possible\_alpha\_values) + 1) \* 0.1  best\_mse = np.min(mean\_results\_for\_all\_possible\_alpha\_values)  print("Best MSE = %s" % best\_mse)  print("Optimal alpha = %s" % optimal\_alpha) |

Output :

|  |
| --- |
| Best MSE = 4417861521188.411  Optimal alpha = 0.1 |

After the optimal ,,alpha'' has been found, we can forecast the t+1 observation as following:

|  |
| --- |
| pt = np.mean(db[:, 0][0:5])  for i in range(1, len(db) + 1):  pt = pt + optimal\_alpha \* (db[i - 1][0] - pt)  print("Next observation = %s" % pt) |

Output :

|  |
| --- |
| Next observation = 2610828.585561429 |

## 

## **Forecast evaluation :**

In this section we will compare the forecast data with the real data for the optimal ,,alpha''.

|  |
| --- |
| forecast = np.zeros(len(db) + 1)  pt = np.mean(db[:, 0][0:5])  forecast[0] = pt  for i in range(1, len(db) + 1):  pt = pt + optimal\_alpha \* (db[i - 1][0] - pt)  forecast[i] = pt  plt.plot(db[:, 0],label = 'real data')  plt.plot(forecast, label = 'forecast')  plt.legend()  plt.show() |

Output :

|  |
| --- |
|  |

Exponential smoothing is a *rule of thumb* technique for smoothing time series data using the exponential window function.

The English phrase ***rule of thumb*** refers to a principle with broad application that is not intended to be strictly accurate or reliable for every situation. It refers to an easily learned and easily applied procedure or standard, based on practical experience rather than theory. This usage of the phrase can be traced back to the seventeenth century and has been associated with various [trades](https://en.wikipedia.org/wiki/Trade_(occupation)) where quantities were measured by comparison to the width or length of a thumb.

## 

## **Conclusion :**

It was shown how the,, Exponential Smoothing" algorithm forecasts based on the smoothing constant,, alpha''. Moreover, it was presented an implementation of how you can find the optimal,, alpha''. When dealing with time series, multiple algorithms should be tested to find out which of them gives the minimum MSE. The algorithm with the minimum MSE should be used for further forecasts on that time series.

**Time Series - Double Exponential Smoothing**

## **1. Introduction**

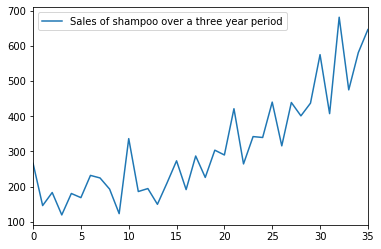
A smoothing method reduces the effects of random variations that the deterministic components of a time series can have. With the help of smoothing methods, we can also forecast new observations for a given time series. The algorithms that use smoothing methods can forecast data for time series that have got or haven't got a trend. If an algorithm using smoothing methods is designed to forecast an observation on a time series that has a trend, we should NOT use that algorithm to forecast a time series that does not have a trend and vice versa.

## **2. Dataset description**

Our time series observations that will be used in this example are the numbers of shampoo sales over a period of 3 years. We can see below how our time series looks like:

|  |
| --- |
| import numpy as np  import pandas as pd  import matplotlib.pyplot as plt  csv\_dataset = pd.read\_csv("sales\_of\_shampoo\_over\_a\_three\_ye.csv")  csv\_dataset.plot()  plt.show() |

Output:

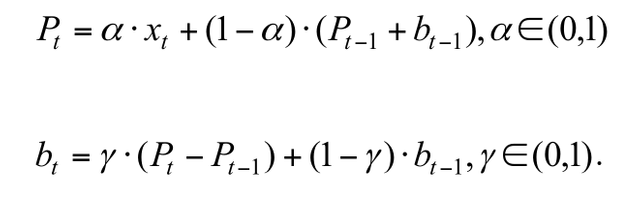


As we can see, the data HAS got an ascending trend and it's composed of 36 observations. In this example, we will forecast the next observation.

## 

## **3. Double Exponential Smoothing algorithm theory**

This algorithm helps us to forecast new observations based on a time series. This algorithm uses smoothing methods. The, Double Exponential Smoothing" algorithm is used only on time series that HAVE a trend. On-time series that have a trend the ,,Exponential Smoothing'' algorithm does not perform very well. This problem was solved by adding a second smoothing constant:, gamma". The mathematical model for this algorithm is:



where ,,alpha" and ,,gamma" are the smoothing constants and they can take values between 0 and 1, ,,P\_t'' is the forecast at time ,,t'', X\_t is the time series observation at time ,,t'', b\_t is the trend value at time ,,t''. In this case, we have to solve two problems:

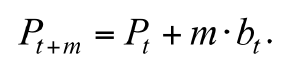
1. Choosing the values of P\_1 and b\_1;
2. Choosing the values of,, alpha" and,, gamma".

For the first problem we can choose P\_1 to be equal to the first observation and for b\_1 we have two common options:

1. we can choose it to be the second observation minus the first (b\_1 = x\_2 - x\_1);
2. we can choose it to be the last observation minus the first, all divided by the total number of observations minus 1 ( b\_1 = (x\_n - x\_1)/(n - 1)).

To solve the second problem we can use the ,,incremental method'': We set the value of alpha (0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9) and gamma (0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9), then we calculate the coresponding MSE for each value that alpha and gamma take. After that, we choose the alpha and gamma with the minimum MSE.

After we got the optimal alpha and gamma and we have calibrated the trend with the formula presented above we can forecast the next,,m'' periods of time using the following formula:



## **4. Double Exponential Smoothing implementation**

The following section will propose an algorithm for finding the best alpha and gamma. The algorithm will start at alpha = 0.1 and gamma = 01 and will go up to gamma = 0.9 then incrementing the alpha to alpha = 0.9. For each alpha and beta, the algorithm will forecast the already known observations along with the corresponding MSE followed by choosing the alpha and beta with the minimum MSE value.

|  |
| --- |
| optimal\_gamma = None  best\_mse = None  db = csv\_dataset.iloc[:, :].values.astype('float32')  mean\_results\_for\_all\_possible\_alpha\_gamma\_values = np.zeros((9, 9))  for gamma in range(0, 9):  for alpha in range(0, 9):  pt = db[0][0]  bt = db[1][0] - db[0][0]  mean\_for\_alpha\_gamma = np.zeros(len(db))  mean\_for\_alpha\_gamma[0] = np.power(db[0][0] - pt, 2)  for i in range(1, len(db)):  temp\_pt = ((alpha + 1) \* 0.1) \* db[i][0] + (1 - ((alpha + 1) \* 0.1)) \* (pt + bt)  bt = ((gamma + 1) \* 0.1) \* (temp\_pt - pt) + (1 - ((gamma + 1) \* 0.1)) \* bt  pt = temp\_pt  mean\_for\_alpha\_gamma[i] = np.power(db[i][0] - pt, 2)  mean\_results\_for\_all\_possible\_alpha\_gamma\_values[gamma][alpha] = np.mean(mean\_for\_alpha\_gamma)  optimal\_gamma, optimal\_alpha = np.unravel\_index(  np.argmin(mean\_results\_for\_all\_possible\_alpha\_gamma\_values),  np.shape(mean\_results\_for\_all\_possible\_alpha\_gamma\_values))  optimal\_alpha = (optimal\_alpha + 1) \* 0.1  optimal\_gamma = (optimal\_gamma + 1) \* 0.1  best\_mse = np.min(mean\_results\_for\_all\_possible\_alpha\_gamma\_values)  print("Best MSE = %s" % best\_mse)  print("Optimal alpha = %s" % optimal\_alpha)  print("Optimal gamma = %s" % optimal\_gamma) |

Output :

|  |
| --- |
| Best MSE = 125.81620704283071  Optimal alpha = 0.9  Optimal gamma = 0.2 |

After the optimal ,,alpha'' and ,,gamma" have been found, we can calibrate the trend as following:

|  |
| --- |
| pt = db[0][0]  bt = db[1][0] - db[0][0]  for i in range(1, len(db)):  temp\_pt = optimal\_alpha \* db[i][0] + (1 - optimal\_alpha) \* (pt + bt)  bt = optimal\_gamma \* (temp\_pt - pt) + (1 - optimal\_gamma) \* bt  pt = temp\_pt  print("P\_t = %s" % pt)  print("b\_t = %s" % bt ) |

Output :

|  |
| --- |
| P\_t = 641.6334222987338  b\_t = 30.033728841040205 |

Now we can forecast the next ,,m'' periods of time using the formula from section 3 like this:

|  |
| --- |
| print("Next observation = %s" % (pt + (1 \* bt))) |

Output:

|  |
| --- |
| Next observation = 671.667151139774 |

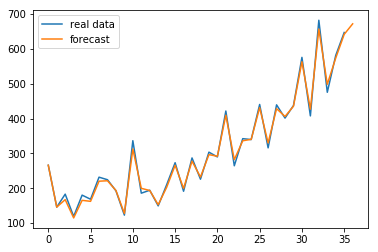
## 

## **5. Forecast evaluation**

In this section we will compare the forecast data with the real data for the optimal ,,alpha'' and ,,gamma''.

|  |
| --- |
| forecast = np.zeros(len(db) + 1)  pt = db[0][0]  bt = db[1][0] - db[0][0]  forecast[0] = pt  for i in range(1, len(db)):  temp\_pt = optimal\_alpha \* db[i][0] + (1 - optimal\_alpha) \* (pt + bt)  bt = optimal\_gamma \* (temp\_pt - pt) + (1 - optimal\_gamma) \* bt  pt = temp\_pt  forecast[i] = pt  forecast[-1] = pt + (1 \* bt)  plt.plot(db[:, 0],label = 'real data')  plt.plot(forecast, label = 'forecast')  plt.legend()  plt.show() |

Output :



As we can see above, the algorithm gives good results on time series that HAVE a trend.

## 

## **6. Conclusion**

It was shown how the ,,Double Exponential Smoothing" algorithm forecasts based on the smoothing constant ,,alpha'' and ,,gamma". Moreover, it was presented an implementation of how you can find the optimal ,,alpha'' and ,,gamma". It was proven that the algorithm gives very good results on time series that have a trend. When dealing with time series, multiple algorithms should be tested to find out which of them gives the minimum MSE. The algorithm with the minimum MSE should be used for further forecasts on that time series.

## **2.1 Implement\_Exponential\_smooting**

## **Import libraries and loading the data**

|  |
| --- |
| import os  import numpy as np  import pandas as pd  import matplotlib.pyplot as plt  from statsmodels.tsa.api import ExponentialSmoothing, SimpleExpSmoothing, Holt  %matplotlib inline |

## **Loading The Data**

|  |
| --- |
| data = [446.6565, 454.4733, 455.663 , 423.6322, 456.2713, 440.5881, 425.3325, 485.1494, 506.0482, 526.792 , 514.2689, 494.211 ]  index= pd.date\_range(start='1996', end='2008', freq='A')  oildata = pd.Series(data, index)  data = [17.5534, 21.86 , 23.8866, 26.9293, 26.8885, 28.8314, 30.0751, 30.9535, 30.1857, 31.5797, 32.5776, 33.4774, 39.0216, 41.3864, 41.5966]  index= pd.date\_range(start='1990', end='2005', freq='A')  air = pd.Series(data, index)  data = [263.9177, 268.3072, 260.6626, 266.6394, 277.5158, 283.834 , 290.309 , 292.4742, 300.8307, 309.2867, 318.3311, 329.3724, 338.884 , 339.2441, 328.6006, 314.2554, 314.4597, 321.4138, 329.7893, 346.3852, 352.2979, 348.3705, 417.5629, 417.1236, 417.7495, 412.2339, 411.9468, 394.6971, 401.4993, 408.2705, 414.2428]  index= pd.date\_range(start='1970', end='2001', freq='A')  livestock2 = pd.Series(data, index)  data = [407.9979 , 403.4608, 413.8249, 428.105 , 445.3387, 452.9942, 455.7402]  index= pd.date\_range(start='2001', end='2008', freq='A')  livestock3 = pd.Series(data, index)  data = [41.7275, 24.0418, 32.3281, 37.3287, 46.2132, 29.3463, 36.4829, 42.9777, 48.9015, 31.1802, 37.7179, 40.4202, 51.2069, 31.8872, 40.9783, 43.7725, 55.5586, 33.8509, 42.0764, 45.6423, 59.7668, 35.1919, 44.3197, 47.9137]  index= pd.date\_range(start='2005', end='2010-Q4', freq='QS-OCT')  aust = pd.Series(data, index) |

## **Simple Exponential Smoothing**[**¶**](https://render.githubusercontent.com/view/ipynb?commit=b706a69f845bc62e3463690901f7d8844500a0e4&enc_url=68747470733a2f2f7261772e67697468756275736572636f6e74656e742e636f6d2f73746174736d6f64656c732f73746174736d6f64656c732f623730366136396638343562633632653334363336393039303166376438383434353030613065342f6578616d706c65732f6e6f7465626f6f6b732f6578706f6e656e7469616c5f736d6f6f7468696e672e6970796e62&nwo=statsmodels%2Fstatsmodels&path=examples%2Fnotebooks%2Fexponential_smoothing.ipynb&repository_id=1885237&repository_type=Repository#Simple-Exponential-Smoothing)

Lets use Simple Exponential Smoothing to forecast the below oil data.

|  |
| --- |
| ax=oildata.plot()  ax.set\_xlabel("Year")  ax.set\_ylabel("Oil (millions of tonnes)")  print("Figure Oil production in Saudi Arabia from 1996 to 2007.") |

Output :

Figure : Oil production in Saudi Arabia from 1996 to 2007.

|  |
| --- |
|  |

Here we run three variants of simple exponential smoothing:

In fit1 we do not use the auto optimization but instead choose to explicitly provide the model with the parameter

In fit2 as above we choose an

In fit3 we allow statsmodels to automatically find an optimized value for us. This is the recommended approach.

|  |
| --- |
| fit1 = SimpleExpSmoothing(oildata).fit(smoothing\_level=0.2,optimized=False)  fcast1 = fit1.forecast(3).rename(r'$\alpha=0.2$')  fit2 = SimpleExpSmoothing(oildata).fit(smoothing\_level=0.6,optimized=False)  fcast2 = fit2.forecast(3).rename(r'$\alpha=0.6$')  fit3 = SimpleExpSmoothing(oildata).fit()  fcast3 = fit3.forecast(3).rename(r'$\alpha=%s$'%fit3.model.params['smoothing\_level'])  plt.figure(figsize=(12, 8))  plt.plot(oildata, marker='o', color='black')  plt.plot(fit1.fittedvalues, marker='o', color='blue')  line1, = plt.plot(fcast1, marker='o', color='blue')  plt.plot(fit2.fittedvalues, marker='o', color='red')  line2, = plt.plot(fcast2, marker='o', color='red')  plt.plot(fit3.fittedvalues, marker='o', color='green')  line3, = plt.plot(fcast3, marker='o', color='green')  plt.legend([line1, line2, line3], [fcast1.name, fcast2.name, fcast3.name]) |

Output :

|  |
| --- |
|  |

## **Holt's Method**

Lets take a look at another example. This time we use air pollution data and the Holt's Method. We will fit three examples again.

1. In fit1 we again choose not to use the optimizer and provide explicit values for and and
2. In fit2 we do the same as in fit1 but choose to use an exponential model rather than a Holt's additive model.
3. In fit3 we used a damped versions of the Holt's additive model but allow the dampening parameter to be optimized while fixing the values for and

|  |
| --- |
| fit1 = Holt(air).fit(smoothing\_level=0.8)  fcast1 = fit1.forecast(5).rename("Holt's linear trend")  fit2 = Holt(air, exponential=True).fit(smoothing\_level=0.8)  fcast2 = fit2.forecast(5).rename("Exponential trend")  fit3 = Holt(air).fit(smoothing\_level=0.8)  fcast3 = fit3.forecast(5).rename("Additive damped trend")  plt.figure(figsize=(12, 8))  plt.plot(air, marker='o', color='black')  plt.plot(fit1.fittedvalues, color='blue')  line1, = plt.plot(fcast1, marker='o', color='blue')  plt.plot(fit2.fittedvalues, color='red')  line2, = plt.plot(fcast2, marker='o', color='red')  plt.plot(fit3.fittedvalues, color='green')  line3, = plt.plot(fcast3, marker='o', color='green')  plt.legend([line1, line2, line3], [fcast1.name, fcast2.name, fcast3.name]) |

Output :

|  |
| --- |
|  |

### **Seasonally adjusted data**

Lets look at some seasonally adjusted livestock data. We fit five Holt's models. The below table allows us to compare results when we use exponential versus additive and damped versus non-damped.

Note: fit4 does not allow the parameter to be optimized by providing a fixed value of

|  |
| --- |
| fit1 = SimpleExpSmoothing(livestock2).fit()  fit2 = Holt(livestock2).fit()  fit3 = Holt(livestock2).fit()  fit4 = Holt(livestock2).fit()  fit5 = Holt(livestock2).fit()  #params = ['smoothing\_level']  results=pd.DataFrame(index=[r"$\alpha$",r"$\beta$",r"$\phi$",r"$l\_0$","$b\_0$","SSE"] ,columns=['SES', "Holt's","Exponential", "Additive", "Multiplicative"])  results["SES"] =fit1.sse  results["Holt's"] = fit2.sse  results["Exponential"] =fit3.sse  results["Additive"] = fit4.sse  results["Multiplicative"] = fit5.sse  results |

Output :

|  |
| --- |
| SES Holt's Exponential Additive Multiplicative  𝛼 6761.350218 6004.1382 6004.1382 6004.1382 6004.1382  𝛽 6761.350218 6004.1382 6004.1382 6004.1382 6004.1382  𝜙 6761.350218 6004.1382 6004.1382 6004.1382 6004.1382  𝑙0 6761.350218 6004.1382 6004.1382 6004.1382 6004.1382  𝑏0 6761.350218 6004.1382 6004.1382 6004.1382 6004.1382  SSE 6761.350218 6004.1382 6004.1382 6004.1382 6004.1382 |

### **Plots of Seasonally Adjusted Data**

The following plots allow us to evaluate the level and slope/trend components of the above table's fits.

|  |
| --- |
| for fit in [fit2,fit4]:  pd.DataFrame(np.c\_[fit.level]).rename(columns={0:'level',1:'slope'}).plot  (subplots=True)  plt.show()  print('Figure : Level and slope components for Holt’s linear trend method and the additive damped trend method.') |

Output :

|  |
| --- |
|  |

|  |
| --- |
|  |

Figure : Level and slope components for Holt’s linear trend method and the additive damped trend method.

## **Comparison**

Here we plot a comparison Simple Exponential Smoothing and Holt's Methods for various additive, exponential and damped combinations. All of the models parameters will be optimized by statsmodels.

|  |
| --- |
| fit1 = SimpleExpSmoothing(livestock2).fit()  fcast1 = fit1.forecast(9).rename("SES")  fit2 = Holt(livestock2).fit()  fcast2 = fit2.forecast(9).rename("Holt's")  fit3 = Holt(livestock2).fit()  fcast3 = fit3.forecast(9).rename("Exponential")  fit4 = Holt(livestock2).fit()  fcast4 = fit4.forecast(9).rename("Additive Damped")  fit5 = Holt(livestock2).fit()  fcast5 = fit5.forecast(9).rename("Multiplicative Damped")  ax = livestock2.plot(color="black", marker="o", figsize=(12,8))  livestock3.plot(ax=ax, color="black", marker="o", legend=False)  fcast1.plot(ax=ax, color='red', legend=True)  fcast2.plot(ax=ax, color='green', legend=True)  fcast3.plot(ax=ax, color='blue', legend=True)  fcast4.plot(ax=ax, color='cyan', legend=True)  fcast5.plot(ax=ax, color='magenta', legend=True)  ax.set\_ylabel('Livestock, sheep in Asia (millions)')  plt.show()  print('Figure: Forecasting livestock, sheep in Asia: comparing forecasting performance of non-seasonal methods.') |

Output :

|  |
| --- |
|  |

Figure: Forecasting livestock, sheep in Asia: comparing forecasting performance of non-seasonal methods.

## **Holt's Winters Seasonal**

Finally we are able to run full Holt's Winters Seasonal Exponential Smoothing including a trend component and a seasonal component. statsmodels allows for all the combinations including as shown in the examples below:

1. fit1 additive trend, additive seasonal of period season\_length=4 and the use of a Box-Cox transformation.
2. fit2 additive trend, multiplicative seasonal of period season\_length=4 and the use of a Box-Cox transformation.
3. fit3 additive damped trend, additive seasonal of period season\_length=4 and the use of a Box-Cox transformation.
4. fit4 additive damped trend, multiplicative seasonal of period season\_length=4 and the use of a Box-Cox transformation.

The plot shows the results and forecast for fit1 and fit2. The table allows us to compare the results and parameterizations.

|  |
| --- |
| fit1 = ExponentialSmoothing(aust, seasonal\_periods=4, trend='add', seasonal='add').fit()  fit2 = ExponentialSmoothing(aust, seasonal\_periods=4, trend='add', seasonal='mul').fit()  fit3 = ExponentialSmoothing(aust, seasonal\_periods=4, trend='add', seasonal='add').fit()  fit4 = ExponentialSmoothing(aust, seasonal\_periods=4, trend='add', seasonal='mul').fit()  results=pd.DataFrame(index=[r"$\alpha$",r"$\beta$",r"$\phi$",r"$\gamma$",r"$l\_0$","$b\_0$","SSE"])  #params = ['smoothing\_level', 'smoothing\_trend', 'damping\_trend', 'smoothing\_seasonal', 'initial\_level', 'initial\_trend']  results["Additive"] = fit1.sse  results["Multiplicative"] = fit2.sse  results["Additive Dam"] = fit3.sse  results["Multiplica Dam"] = fit4.sse  ax = aust.plot(figsize=(10,6), marker='o', color='black', title="Forecasts from Holt-Winters' multiplicative method" )  ax.set\_ylabel("International visitor night in Australia (millions)")  ax.set\_xlabel("Year")  fit1.fittedvalues.plot(ax=ax, style='--', color='red')  fit2.fittedvalues.plot(ax=ax, style='--', color='green')  fit1.forecast(8).rename('Holt-Winters (add-add-seasonal)').plot(ax=ax, style='--', marker='o', color='red', legend=True)  fit2.forecast(8).rename('Holt-Winters (add-mul-seasonal)').plot(ax=ax, style='--', marker='o  ', color='green', legend=True)  plt.show()  print("Figure 7.6: Forecasting international visitor nights in Australia using Holt-Winters method with both additive and multiplicative seasonality.")  results |

Output :

|  |
| --- |
|  |