Chapter 5

Exponential Smoothing

5.1 Overview

Time series distinguish themselves from other types of data because of their dependency on time. Most forecasting methods utilise this relationship between current data and past data to predict future data. Within this context, exponential smoothing is a method that assumes a future observation is the weighted sum of past observations. However, what really sets exponential smoothing apart is that these weights decrease exponentially as you go further into the past.

There are three common variants of exponential smoothing that differ in the degree of complexity they can model.

5.2 Simple Exponential Smoothing

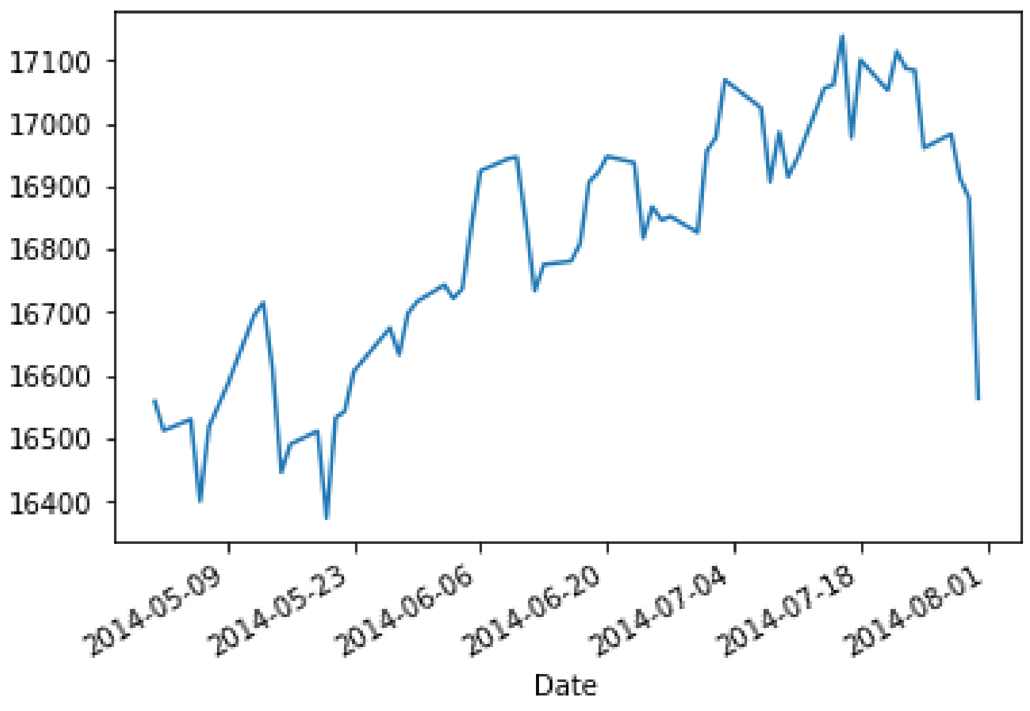
Simple exponential smoothing (SES) models a time series as the weighted sum of past observations and specifically assumes no trend or seasonality. The formula for SES is as follows:

$$level_t = \alpha y_t + (1 - \alpha) level_{t-1}$$
$$level_0 = y_0$$

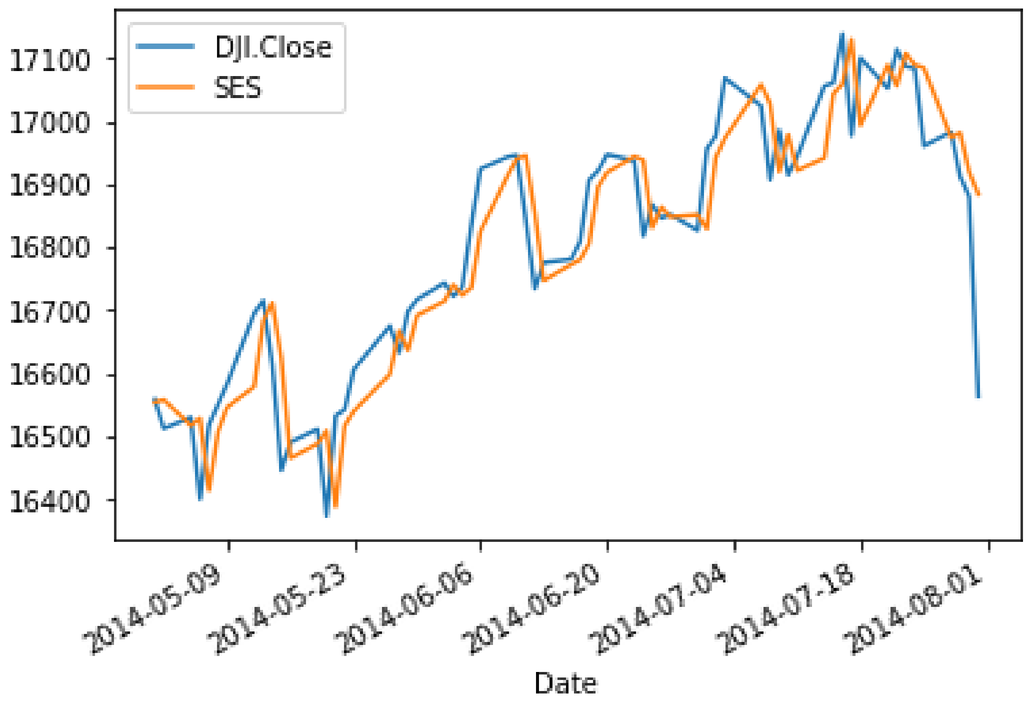
The parameter α controls the strength of the exponential decay. A large α puts more weight on recent observations and other weights decrease rapidly. A small α on the other hand allows past observations to still have some influence on the forecasted values.

We fit a SES model using `SimpleExpSmoothing` from the **statsmodels** package which estimates the α parameter by minimizing the sum of squared errors.

```
>>> from matplotlib.dates import AutoDateLocator
>>> import pandas as pd
>>> dji = pd.read_csv("dji.csv", parse_dates=['Date'])
>>> dji = dji.sort_values(by="Date",
ascending=True).set_index("Date")
>>> dji_summer = dji["2014-05":"2014-07"].copy()
>>> ax = dji_summer["DJI.Close"].plot()
>>>
ax.xaxis.set_major_locator(AutoDateLocator(interval_multiples=False)
)
```



```
>>> from statsmodels.tsa.holtwinters import SimpleExpSmoothing
>>> ses = SimpleExpSmoothing(dji_summer["DJI.Close"].values)
>>> ses_model = ses.fit()
>>> dji_summer["SES"] = ses_model.fittedvalues
>>> ax = dji_summer[["DJI.Close", "SES"]].plot()
>>>
ax.xaxis.set_major_locator(AutoDateLocator(interval_multiples=False)
)
```

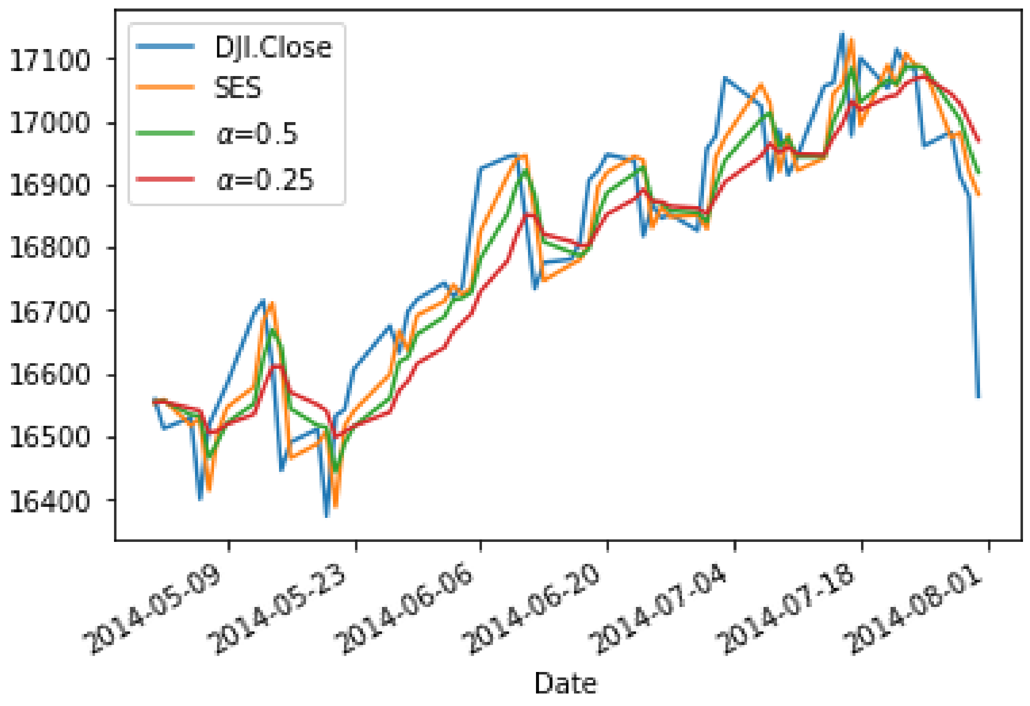


```
>>> ses_model.params['smoothing_level']

0.8922003343920327
```

The fitted model has a high α, meaning recent observations are deemed more important. This makes sense as the fitted values follow the actual values closely. However, it's interesting to see how different values for α result in a much smoother line.

```
>>> initial_level = ses_model.params['initial_level']
>>> dji_summer[r'$\alpha$=0.5'] = ses.predict(params={
      'smoothing_level':0.5, 'initial_level': initial_level},
start=0)
>>> dji_summer[r'$\alpha$=0.25'] = ses.predict(params={
      'smoothing_level':0.25, 'initial_level': initial_level},
start=0)
>>> ax = dji_summer[["DJI.Close", "SES", r'$\alpha$=0.5',
      r'$\alpha$=0.25']].plot()
>>>
ax.xaxis.set_major_locator(AutoDateLocator(interval_multiples=False)
)
```



The goal of fitting a model is of course to generate a forecast. Since SES does not make any assumptions about the trend or seasonality, what is left is a fixed forecast. Meaning that whether we want a forecast for 1 period ahead or 10 periods ahead makes no difference, the value is the same:

```
>>> num_observations = dji_summer.shape[0]
>>> ses_model.predict(start=num_observations, end=num_observations +
10)

array([16597.93567066, 16597.93567066, 16597.93567066,
16597.93567066,
       16597.93567066, 16597.93567066, 16597.93567066,
16597.93567066,
       16597.93567066, 16597.93567066, 16597.93567066])
```

5.3 Double Exponential Smoothing

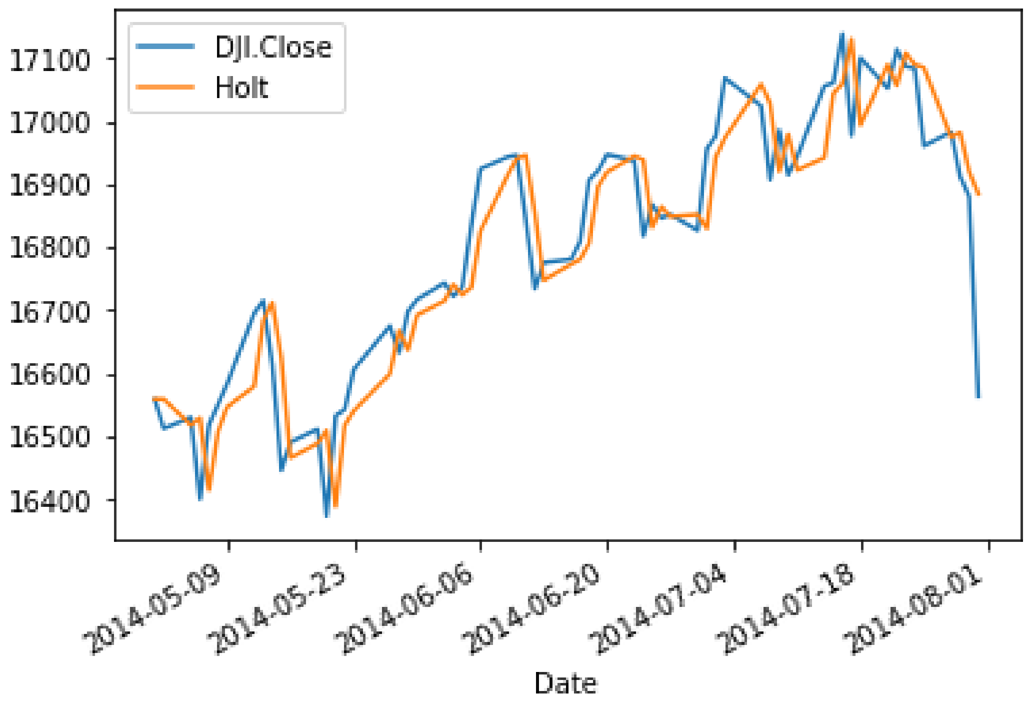
Holt’s linear method or double exponential smoothing (DES) extends the above model by adding a trend component. Moreover, the weight that is assigned to this trend component is also exponentially decaying. This allows the impact of the trend component on the time series to vary over time. The formulation for Holt’s linear method is as follows:

$$forecast_{t+h} = level_t + h * trend_t$$
$$level_t = \alpha y_t + (1-\alpha)(level_{t-1} + trend_{t-1})$$
$$trend_t = \beta(level_t - level_{t-1}) + (1-\beta)trend_{t-1}$$

On top of α we now also have a β parameter which is also referred to as the *trend smoothing factor*. Now that we have included a trend component our forecast is no longer fixed but instead a function of the number of steps ahead (*h*). We will come back to how this looks after an example.

To fit Holt’s linear method we can use `Holt` from the **statsmodels** package.

```
>>> from statsmodels.tsa.holtwinters import Holt
>>> holt = Holt(dji_summer["DJI.Close"].values, exponential=True)
>>> holt_model = holt.fit()
>>> dji_summer["Holt"] = holt_model.fittedvalues
>>> ax = dji_summer[["DJI.Close", "Holt"]].plot()
>>>
ax.xaxis.set_major_locator(AutoDateLocator(interval_multiples=False)
)
```



```
>>> (holt_model.params['smoothing_level'],
holt_model.params['smoothing_slope'])

(0.8922561812124674, 0.0)
```

The zero value for β means the initial trend does not change over time, which makes sense looking at the above plot that seems to have a constant trend over the whole series. To generate predictions, we again use the `predict` function. As mentioned before, this forecast is a function of the last estimated level, last estimated trend and the number of periods we're looking ahead. Since the level and trend never change the forecast is always increasing by the same amount.

```
>>> predictions = holt_model.predict(start=num_observations,
      end=num_observations + 10)
>>> predictions

array([16598.40698937, 16598.84297481, 16599.27897169,
16599.71498003,
        16600.15099983, 16600.58703107, 16601.02307377,
16601.45912792,
        16601.89519353, 16602.33127059, 16602.7673591 ])

>>> pd.Series(predictions).diff().values
```

```
array([        nan, 0.43598544, 0.43599689, 0.43600834, 0.43601979,
       0.43603124, 0.4360427 , 0.43605415, 0.43606561, 0.43607706,
       0.43608851])
```

An always increasing forecast may not make sense, so in the next section we will look at how to model this differently.

5.4 Holt-Winters

Beyond a level and trend a time series may also have a seasonal component. After adding this to Holt's linear method we end up with what is called Holt-Winters' exponential smoothing. This includes a seasonal component that also has an exponentially decaying weight.

To infer a seasonal component we must also specify the periodicity of the seasonality (e.g. 4 months or 7 days). The weights then decay after each full cycle of seasonality (e.g after 4 months).

The formulation for Holt-Winters is as follows:

$$forecast_{t+h} = level_t + h * trend_t + seasonal_{t+h-m(k+1)}$$
$$level_t = \alpha(y_t - seasonal_{t-m}) + (1-\alpha)(level_{t-1} + trend_{t-1})$$
$$trend_t = \beta(level_t - level_{t-1}) + (1-\beta)trend_{t-1}$$
$$seasonal_t = \gamma(y_t - level_{t-1} - trend_{t-1}) + (1-\gamma)seasonal_{t-m}$$

The addition of a seasonal component has made the model more complicated so let's unpack it.

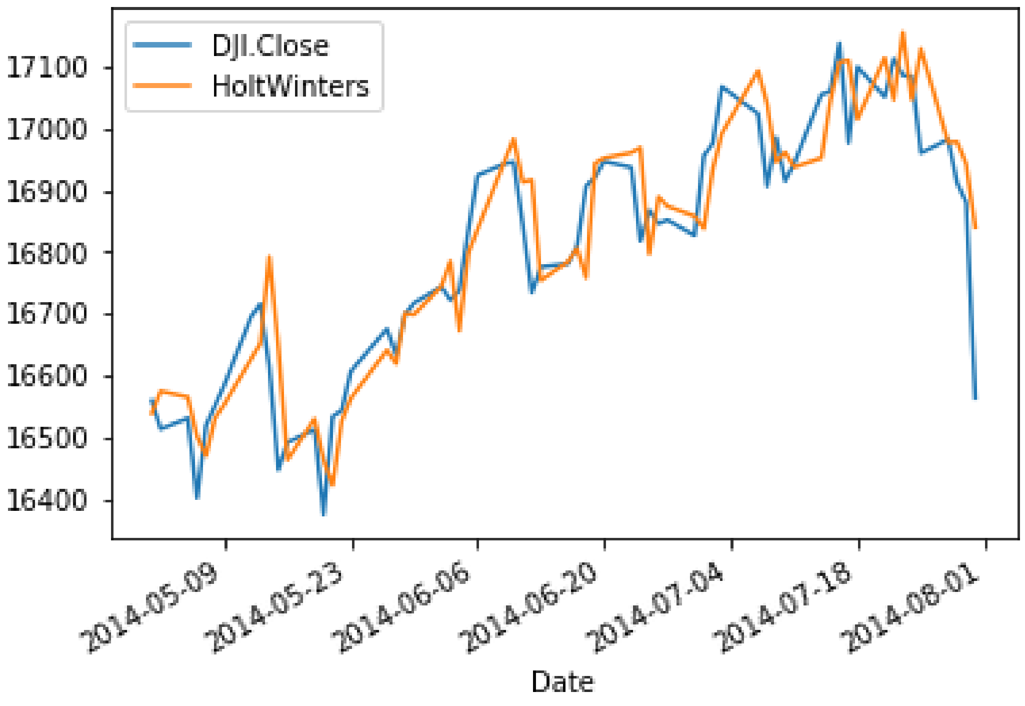
First, the periodicity of the seasonal component is captured by *m* and its inclusion (together with *k*) into the indices ensures that only seasonal values of the same period are used together. For example, if the periodicity is weekly, then the seasonal value for next Monday is calculated by looking at the seasonal values for previous Mondays.

Second, the *level* is now calculated by first subtracting the seasonal value (from a full cycle ago) from the observed value.

Finally, the seasonal value is updated by multiplying γ with the current seasonal value (observed minus level and trend) and $(1-\gamma)$ with the previous seasonal value.

To fit Holt-Winters' exponential smoothing we can use `ExponentialSmoothing` from the **statsmodels** package.

```
>>> from statsmodels.tsa.holtwinters import ExponentialSmoothing
>>> # our seasonal periods will be 5 (number of working days in a
week)
>>> hw = ExponentialSmoothing(dji_summer["DJI.Close"].values,
trend="add", seasonal="add", seasonal_periods=5)
>>> hw_model = hw.fit()
>>> dji_summer["HoltWinters"] = hw_model.fittedvalues
>>> ax = dji_summer[["DJI.Close", "HoltWinters"]].plot()
>>>
ax.xaxis.set_major_locator(AutoDateLocator(interval_multiples=False)
)
```



```
>>> (hw_model.params['smoothing_level'],
      hw_model.params['smoothing_slope'],
      hw_model.params['smoothing_seasonal'])

(0.894736839708109, 0.04968950918756429, 0.10526315549758278)
```

In the above model specification we have explicitly added a periodicity of 5 days which seems reasonable considering stocks are mostly traded during the week. Nevertheless it is an assumption which may need tweaking to achieve a better model. Additionally, we now see the trend component is no longer zero indicating it changes slightly after correcting for seasonality.

Using the `predict` function now generates the following forecasts:

```
>>> hw_model.predict(start=num_observations, end=num_observations +
10)

array([16595.19353256, 16595.13030782, 16566.04345607,
16575.04157568,
       16522.56614866, 16525.31220633, 16525.24898159,
16496.16212984,
       16505.16024945, 16452.68482242, 16455.4308801 ])
```

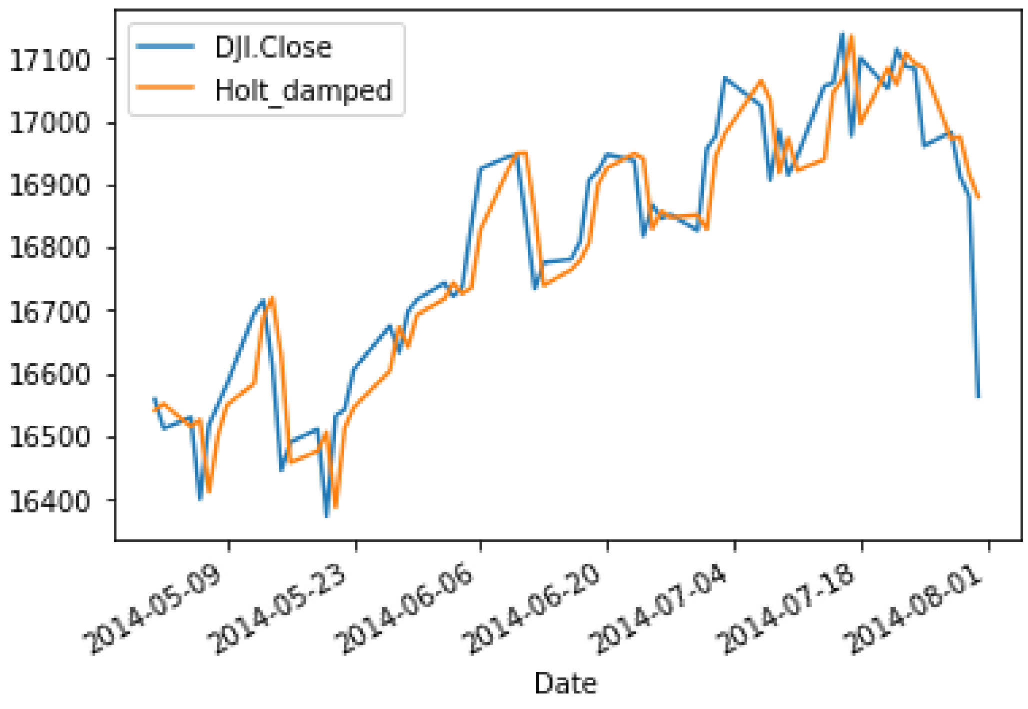
1. Apply all three algorithms to the *Close* column from the `barclays.csv` data with suitable parameter values.
2. Create a plot with the fitted values from the three models and the original data
3. Add forecasts from the three models to the same plot

5.4.1 Dampening

The Holt-Winters forecasts are no longer strictly increasing but nevertheless the components (level, trend and seasonal) on which they are based do not change as the horizon changes. In some cases, this may be sufficient but in most cases it won't be realistic. For example, it is unlikely the Dow Jones Index will keep on increasing into infinity.

To overcome this disadvantage, we introduce *dampening*. Dampening decreases the effect of the trend on the time series thus reducing it back to the level. Both the implementation of Holt's linear method as well as Holt-Winters in **statsmodels** allow dampening, which is achieved by multiplying the trend with an extra parameter ϕ.

```
>>> holt_damped = Holt(dji_summer["DJI.Close"].values,
exponential=True, damped=True)
>>> holt_damped_model = holt_damped.fit()
>>> dji_summer["Holt_damped"] = holt_damped_model.fittedvalues
>>> ax = dji_summer[["DJI.Close", "Holt_damped"]].plot()
>>>
ax.xaxis.set_major_locator(AutoDateLocator(interval_multiples=False)
)
```



```
>>> (holt_damped_model.params['smoothing_level'],
holt_damped_model.params['smoothing_slope'],
>>> holt_damped_model.params['damping_slope'])

(0.7372791581264476, 0.7372791565751314, 0.31759732826678067)

>>> holt_damped_model.predict(start=num_observations,
end=num_observations + 10)

array([16589.21802981, 16571.09167992, 16565.3389447 ,
16563.51230925,
       16562.93221686, 16562.74798532, 16562.6894743 , 16562.6708914
,
       16562.66498953, 16562.66311511, 16562.6625198 ])
```

The introduction of dampening has resulted in a non-zero value for β (the smoothing slope). The value for ϕ (damping slope) is quite small which means the dampening effect is strong. We can see this in the forecast values as they quickly become almost constant after only four steps ahead.

5.4.2 *Additive vs Multiplicative Seasonality*

In the above version of the Holt-Winters model the seasonal component is assumed to be additive. This means the seasonality is assumed to be approximately constant through the time series. That is not to be confused with the γ parameter which can cause the *influence* of past seasonal values to change.

For example, consider a weekly seasonal component. Then an additive model will add or subtract a constant amount to each day and this amount will only depend on what day of the week it is.

The other option for the Holt-Winters model is to assume the seasonal component is multiplicative. Consider again the above example. This time the amount you add to the time series depends not only on the day of the week but also on the value of the time series. So now the seasonal values are proportional to the value of the time series.

The formulation of Holt-Winters multiplicative model is different from the additive model and we add it here for completeness.

$$forecast_{t+h} = (level_t + h * trend_t) * seasonal_{t+h-m(k+1)}$$
$$level_t = \alpha \frac{y_t}{seasonal_{t-m}} + (1-\alpha)(level_{t-1} + trend_{t-1})$$
$$trend_t = \beta^*(level_t - level_{t-1}) + (1-\beta^*)trend_{t-1}$$
$$seasonal_t = \gamma \frac{y_t}{(level_{t-1} + trend_{t-1})} + (1-\gamma)seasonal_{t-m}$$

We can apply the multiplicative model by using the same `ExponentialSmoothing` class from **statsmodels**.

```
>>> # we now change the value of seasonal from 'add' to 'mul'
>>> hw = ExponentialSmoothing(dji_summer["DJI.Close"].values,
trend="add", seasonal="mul", seasonal_periods=7)
>>> hw_model = hw.fit()
>>> (hw_model.params['smoothing_level'],
hw_model.params['smoothing_slope'],
>>>  hw_model.params['smoothing_seasonal'])

(0.8421052596746049, 0.05263157889700815, 0.1578947338007189)
```

The implementation of Holt-Winters in **statsmodels** also allows you to specify if the trend is additive or multiplicative by setting the `trend` argument to either “add” or “mul”.

●

1. Apply Double Exponential Smoothing and HoltWinters to the *Close* column from the `barclays.csv` dataset with:
    a. dampening on/off
    b. multiplicative/additive trend
    c. multiplicative/additive seasonality
2. Add the above models to a plot with the original data. Based on the visualisations which model is the best fit?

5.5 Comparing models

So far we have fitted models and judged their fit through visualisations. There are, however, more objective ways to asses model fit. One natural choice is the squared sum of errors, as that is what the method minimizes.

```
>>> (ses_model.sse, holt_model.sse, holt_damped_model.sse,
hw_model.sse)

(457655.7907193868, 457664.41899104137, 454214.9179883455,
500488.72056735225)
```

The **statsmodels** implementation also includes the Akaike Information Criterion (`<model>.aic`) and the Bayesian Information Criterion (`<model>.bic`).

```
>>> index = ['SES', 'Holt', 'Holt Damped', 'Holt-Winters']
>>> pd.DataFrame({'SSE':(ses_model.sse, holt_model.sse,
holt_damped_model.sse, hw_model.sse),
>>>                'AIC': (ses_model.aic, holt_model.aic,
holt_damped_model.aic, hw_model.aic),
>>>                'BIC':(ses_model.bic, holt_model.bic,
holt_damped_model.bic, hw_model.bic)},
>>>               index=index)

                  SSE         AIC         BIC
SES         457655...   571.99...   576.31...
Holt        457664...   576.00...   584.63...
Holt Da...  454214...   577.51...   588.31...
Holt-Wi...  500488...   595.72...   619.47...
```

●

Using a metric of your choice determine which model produces the best fit on the `barclays.csv` dataset.