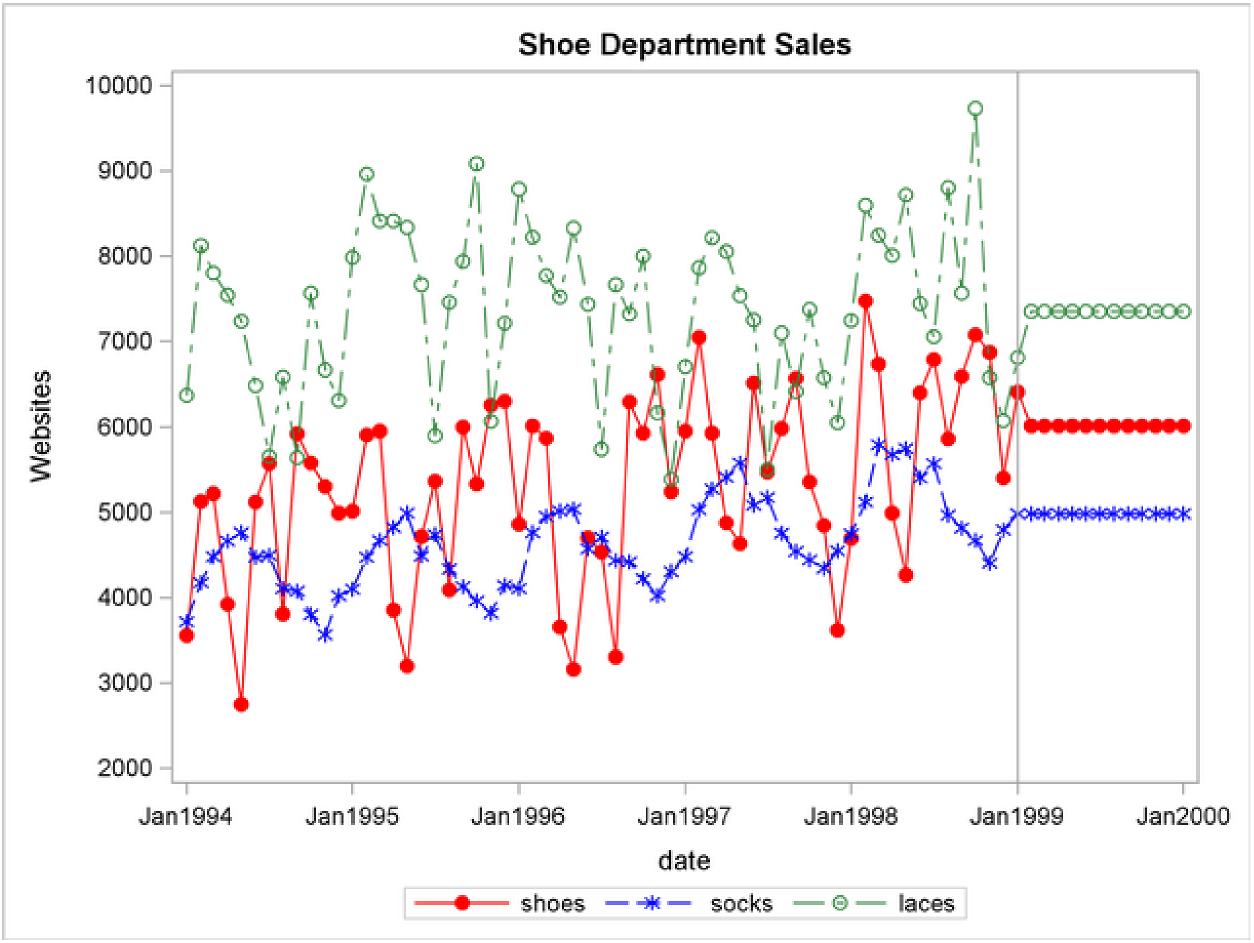
Time Series Analysis

So far predictive modelling techniques that we have seen have involved predicting a target which depends on some other predictor variables. In this module however we'll see how to predict future values of variables which depend on themselves [Their own past values].

We are going to focus on Univariate Time Series models only.

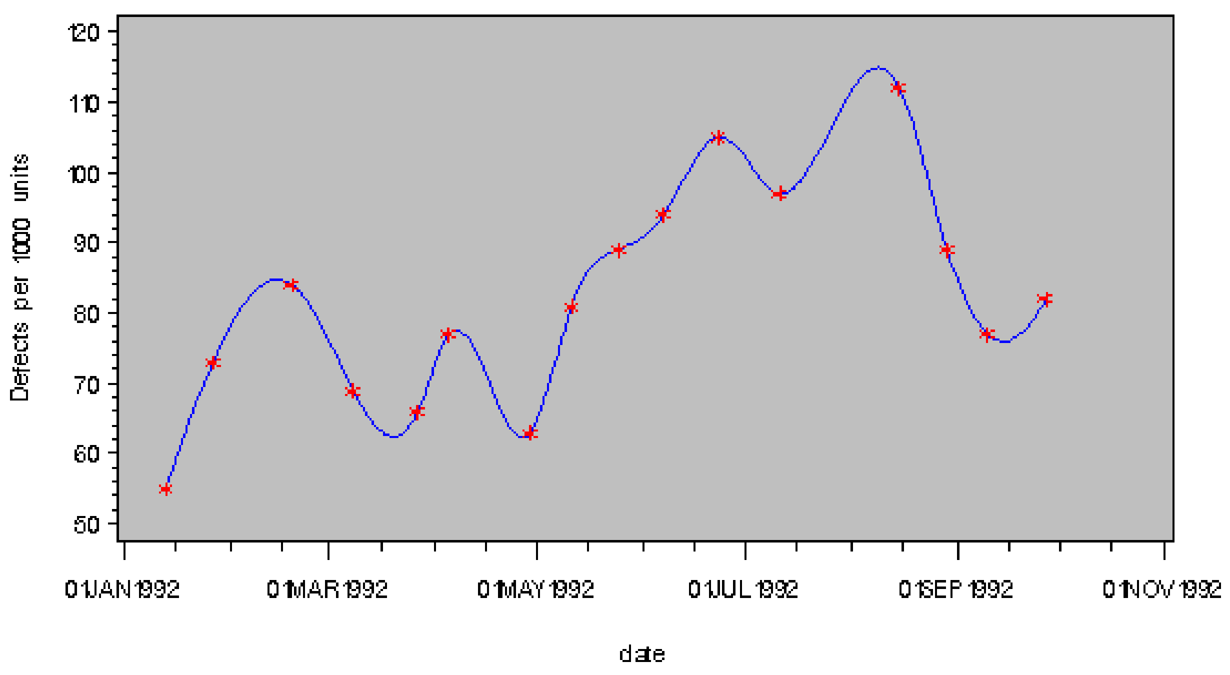
Essentially every modelling task is about finding a pattern in the data. Before we go ahead and start finding these patterns and their equations , lets first see what kind of patterns can their be in a variable which depends on its own past values.

First kind of pattern is where there is seemingly no pattern. [Although later we'll see that this the most general challenge to solve in time series analysis].

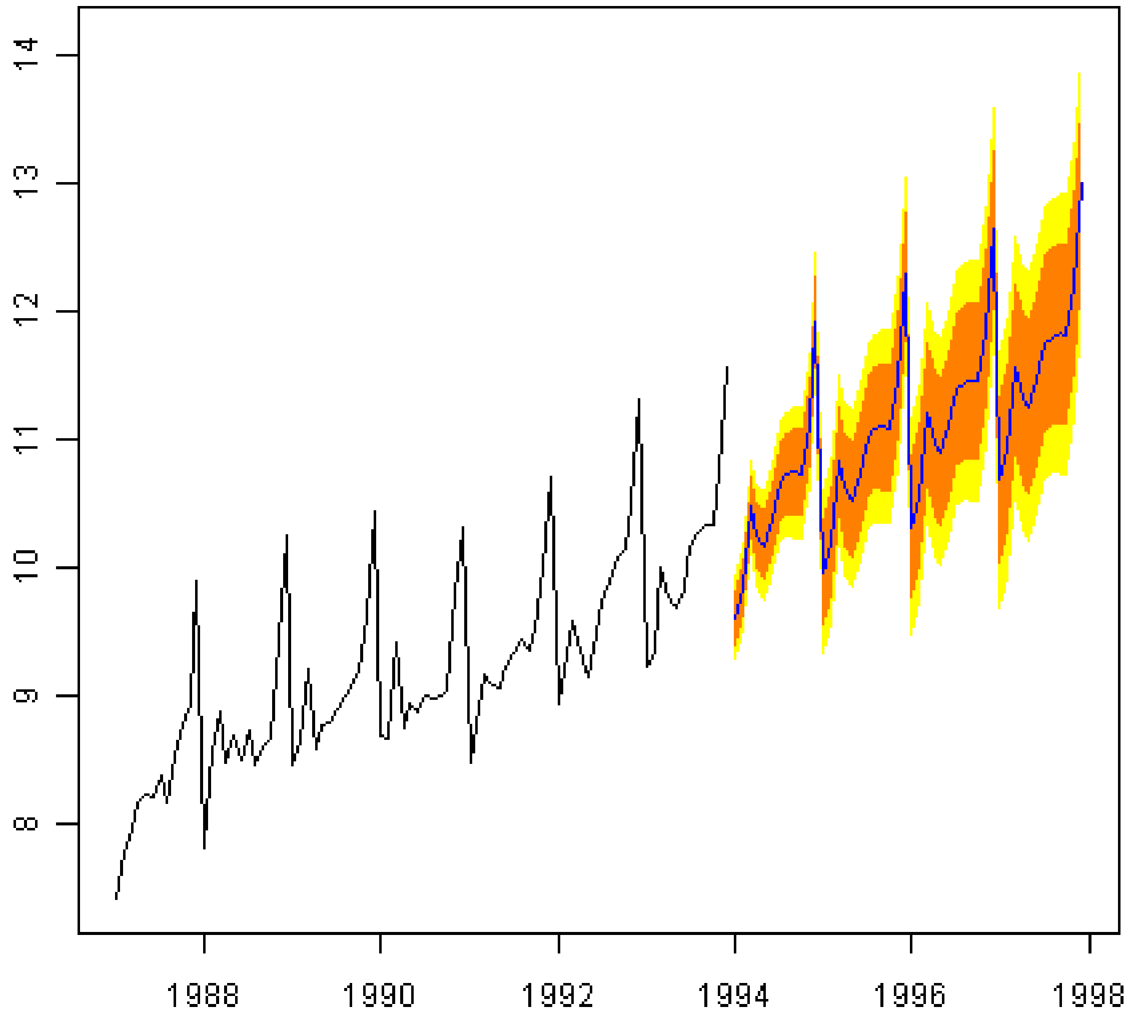


Here you can see that level of each time series seems to change but there is no apparent trend in the data. Next kind is where there is some persistent trend through out the data, although it might change directions at times.

Plot of Interpolated Defect Rate Curve



Building on the same, next type of data can be where there is a persistent but constant cyclic pattern. This can be with or without linear trend in the data.



We'll first look at a simpler method modelling these kind of data, also known as exponential smoothing.

First kind of pattern that we saw , essentially has no pattern. Our forecast will be simply a constant value. But that level has been updating itself depending on its past value. To know current value of level we need to evaluate updation model as well.

Forecast Model : $\hat{y}_{t+1|t} \;=\; l_t \;+\; \epsilon_t$

Updation Model : $l_t \;=\; \alpha * y_t \;+\; (1-\alpha) * l_{t-1}$

Here α in the updation model equation is the weight-age given to most recent value of level and rest $[(1-\alpha)]$ is given to previous level values. If you expand this recurrence equation it looks like this:

$$l_t \;=\; \alpha * y_t \;+\; (1-\alpha) * \alpha * y_{t-1} \;+\; (1-\alpha)^2 * \alpha * y_{t-2}.....$$

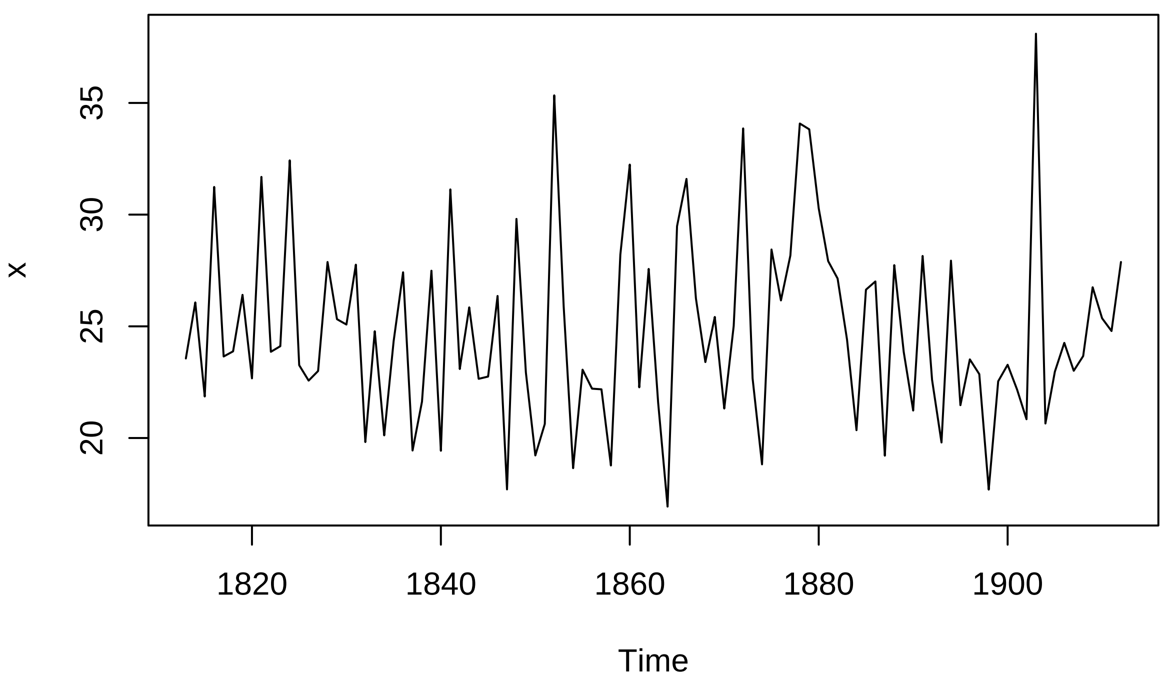
You can see here that weight-age for past values keeps on decreasing exponentially, that is where the name exponential smoothing comes from. You can fit in this equation in the forecasting model and get estimate of α my minimizing errors sum of squares. $\sum \epsilon_t^2$

Let us see an example of this model in R. We'll look at 100 year rain fall data starting from the year 1813.

```
rain=read.csv("rain.csv")
```

To be able to model this data with time series functions , we need to first convert this to a time series data object.

```
raints=ts(rain,start=c(1813))
plot(raints)
```



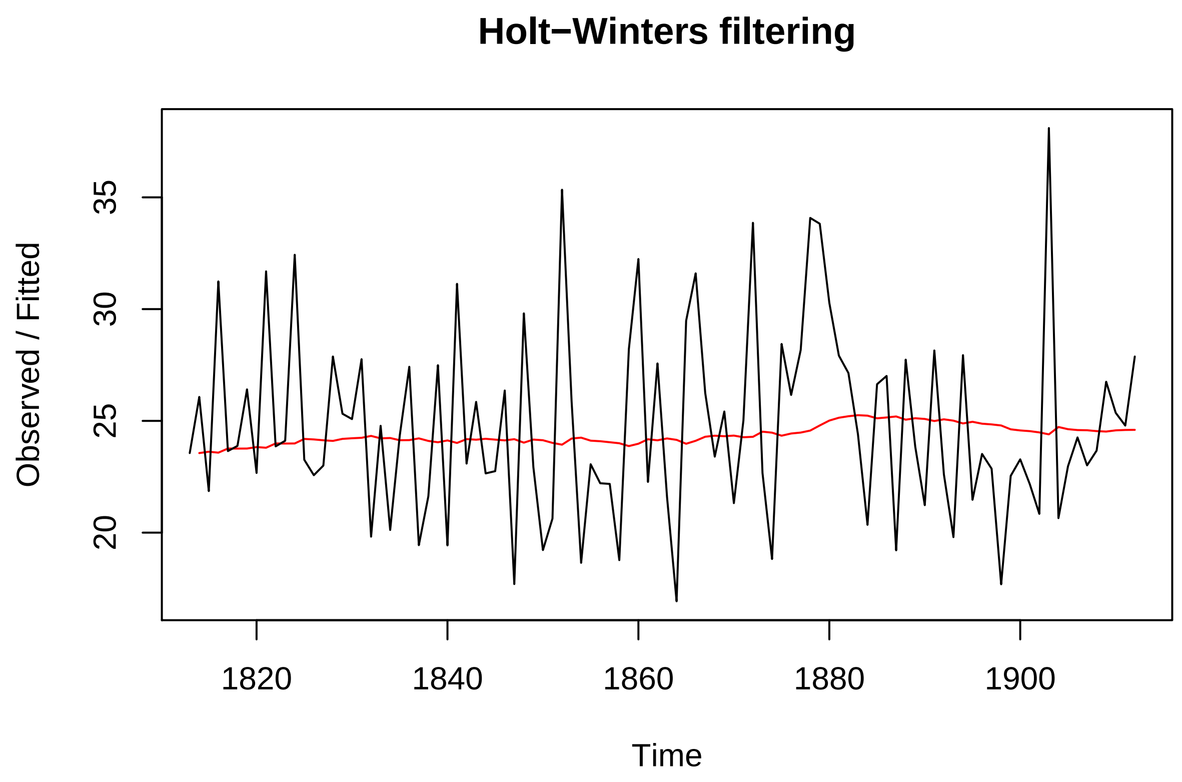
You can see , that this data doesn't have any of the linear trend or seasonality for that matter.We are going to use function `HoltWinters` to build this exponential smoothing model. `Holtwinters` has three parameters : alpha, beta and gamma which correspond to level, linear trend and seasonality in the data. For this data we'll be setting `beta` and `gamma` to `FALSE`.

```
rainforecast=HoltWinters(raints,beta=F,gamma=F)
rainforecast
```

```
## Holt-Winters exponential smoothing without trend and without seasonal component.
##
## Call:
## HoltWinters(x = raints, beta = F, gamma = F)
##
## Smoothing parameters:
##  alpha: 0.02412151
##  beta : FALSE
##  gamma: FALSE
##
## Coefficients:
##       [,1]
## a 24.67819
```

Value of `alpha` here tells how much weight-age is given to most recent values. value of `a` at the end is the constant forecast. You can plot object `rainforecast` to see how level is getting updated over time.

```r
plot(rainforecast)
```

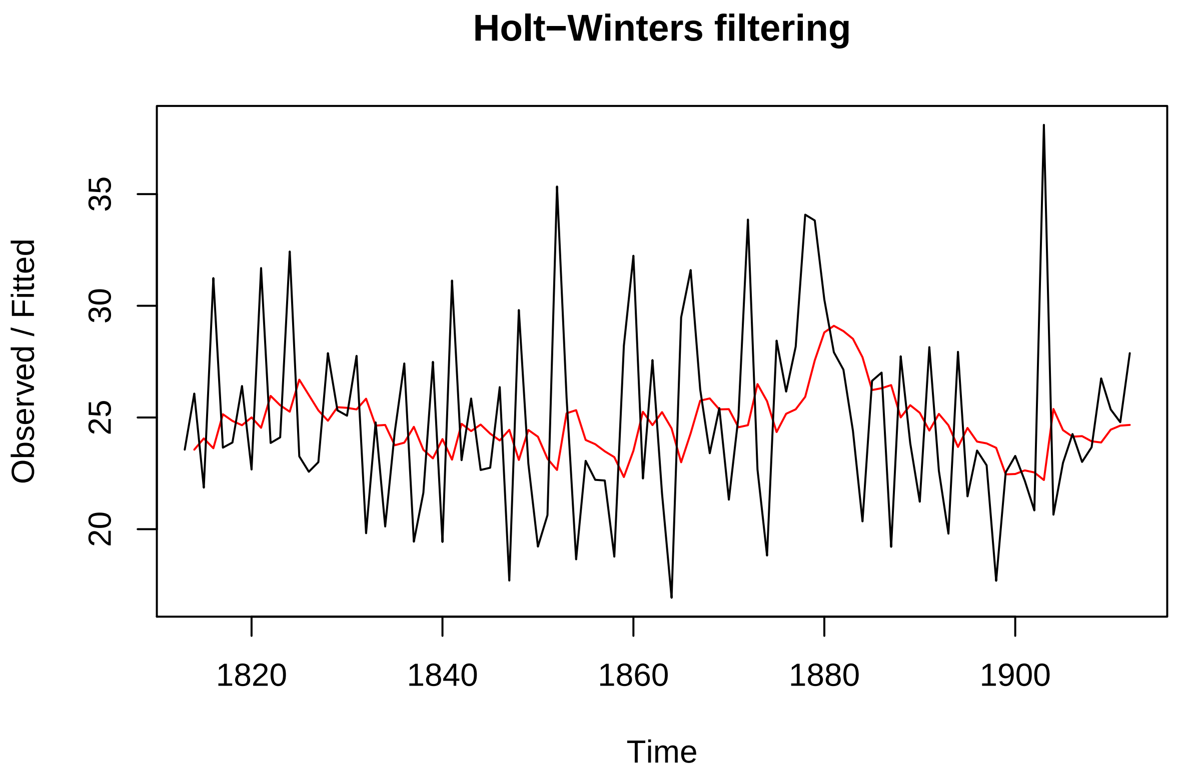


Value of `alpha` that you see in the model summary is the estimated value according to entire data. You can use your own preconceived value of `alpha` as well. This will change your forecast equation. Notice that value of `a` at then end of the model summary has been changed.

```r
r2=HoltWinters(raints,alpha=0.2,beta=F,gamma=F)
r2
```

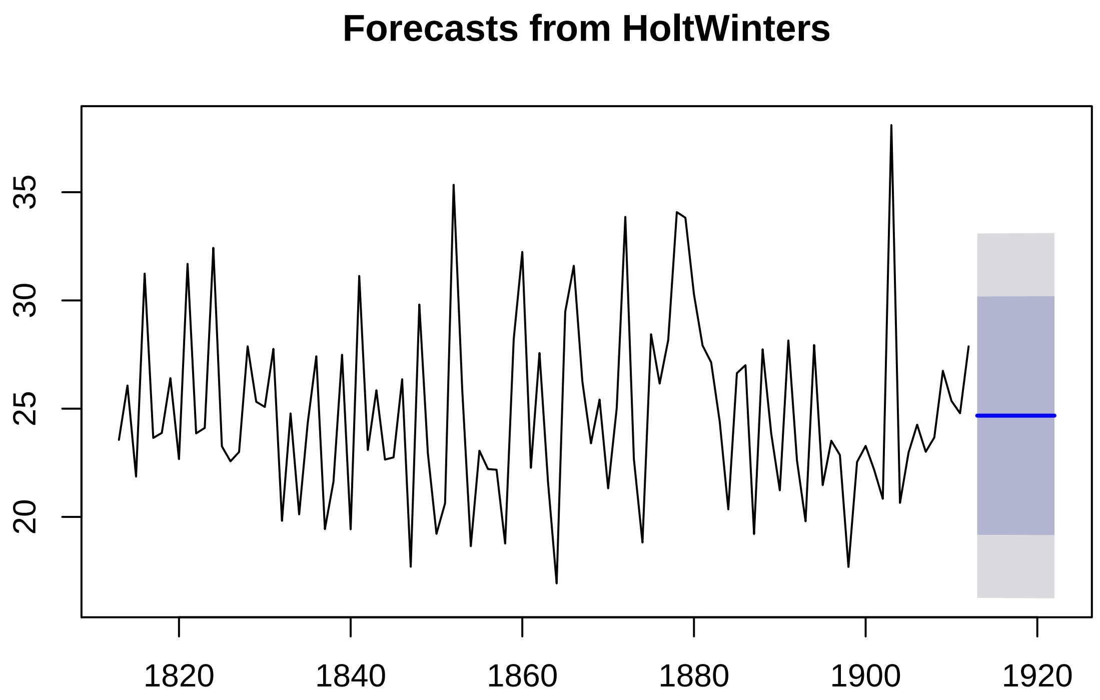
```
## Holt-Winters exponential smoothing without trend and without seasonal component.
##
## Call:
## HoltWinters(x = raints, alpha = 0.2, beta = F, gamma = F)
##
## Smoothing parameters:
##  alpha: 0.2
##  beta : FALSE
##  gamma: FALSE
##
## Coefficients:
##       [,1]
## a 25.30941
```

```r
plot(r2)
```



Now to get forecast and confidence intervals, we'll be using `forecast.HoltWinters` function from the library `forecast`.

```
library(forecast)
r2forecast=forecast.HoltWinters(rainforecast,h=10)
plot(r2forecast)
```



Parameter `h` in the forecast function call corresponds to horizon units for which we want to forecast values.

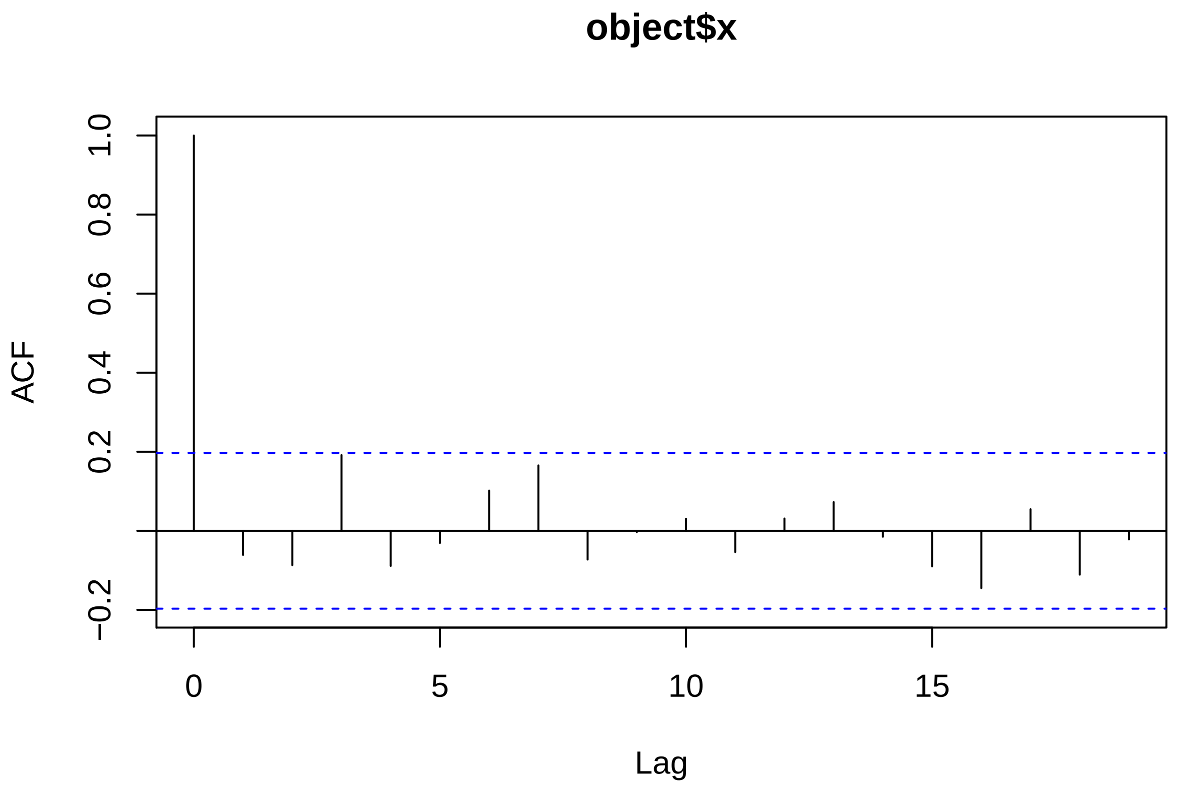
Forecast values and confidence band values can be accessed here.

```
r2forecast
```

```
##      Point Forecast    Lo 80    Hi 80    Lo 95    Hi 95
## 1913       24.67819 19.17493 30.18145 16.26169 33.09470
## 1914       24.67819 19.17333 30.18305 16.25924 33.09715
## 1915       24.67819 19.17173 30.18465 16.25679 33.09960
## 1916       24.67819 19.17013 30.18625 16.25434 33.10204
## 1917       24.67819 19.16853 30.18785 16.25190 33.10449
## 1918       24.67819 19.16694 30.18945 16.24945 33.10694
## 1919       24.67819 19.16534 30.19105 16.24701 33.10938
## 1920       24.67819 19.16374 30.19265 16.24456 33.11182
## 1921       24.67819 19.16214 30.19425 16.24212 33.11427
## 1922       24.67819 19.16054 30.19584 16.23968 33.11671
```

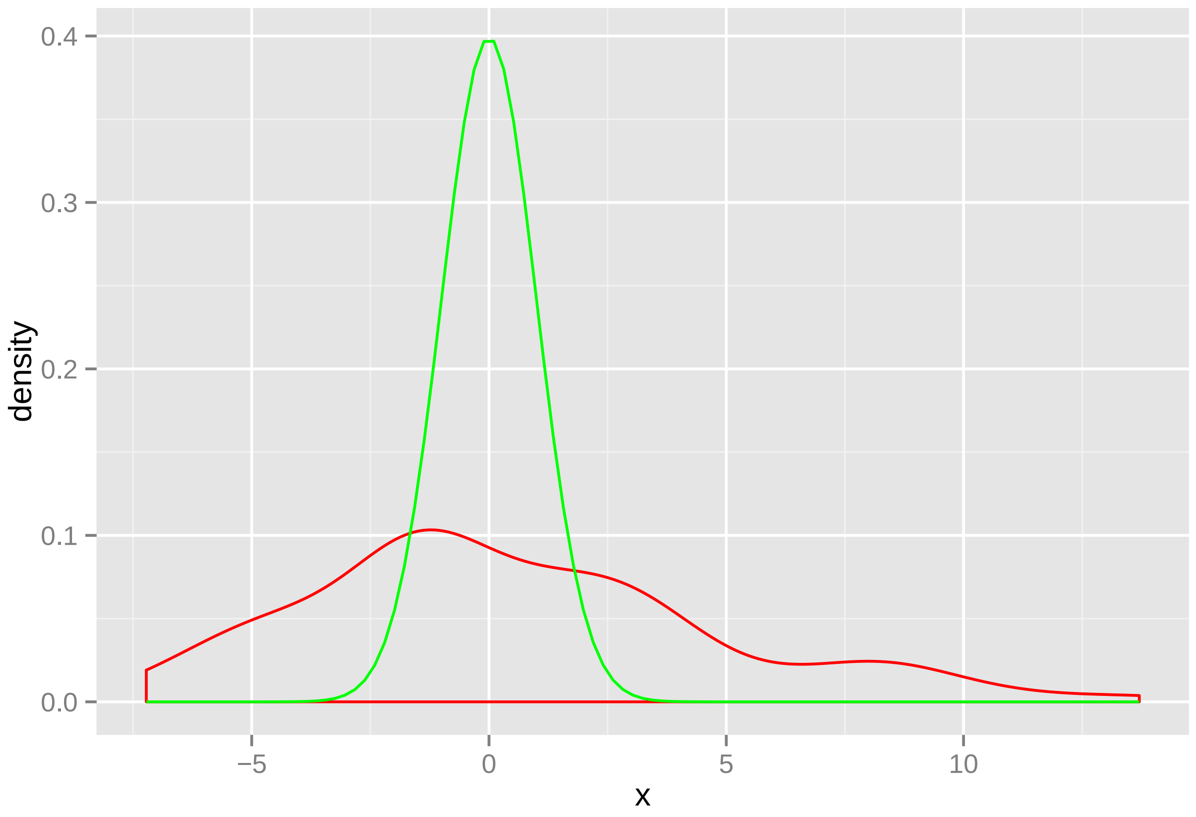
Now for model to be adequate, our errors should be independent and for our confidence bands to be correct , errors should follow normal distribution. We can check that by looking at auto-correlation plot and normal density curves.

```
acf(r2forecast$residuals)
```



None of the auto-correlation bars are going above the blue significance line, which tells that errors are independent. Lets look at normality curves to see whether we can rely on the confidence bands.

```
library(ggplot2)
d=data.frame(x=as.vector(r2forecast$residuals))
ggplot(d,aes(x))+
  geom_density(color="red")+
  stat_function(fun=dnorm,aes(x=x),color="green")
```



Clearly, errors are not following normal distribution, you should be wary of confidence bands provided with that assumption.

Next we look at the model when data has linear trend. Equations for level models will updated to accommodate linear trend as well. Here an addition parameter β will come in picture which is nothing but weight-age given to most recent value of linear trend.

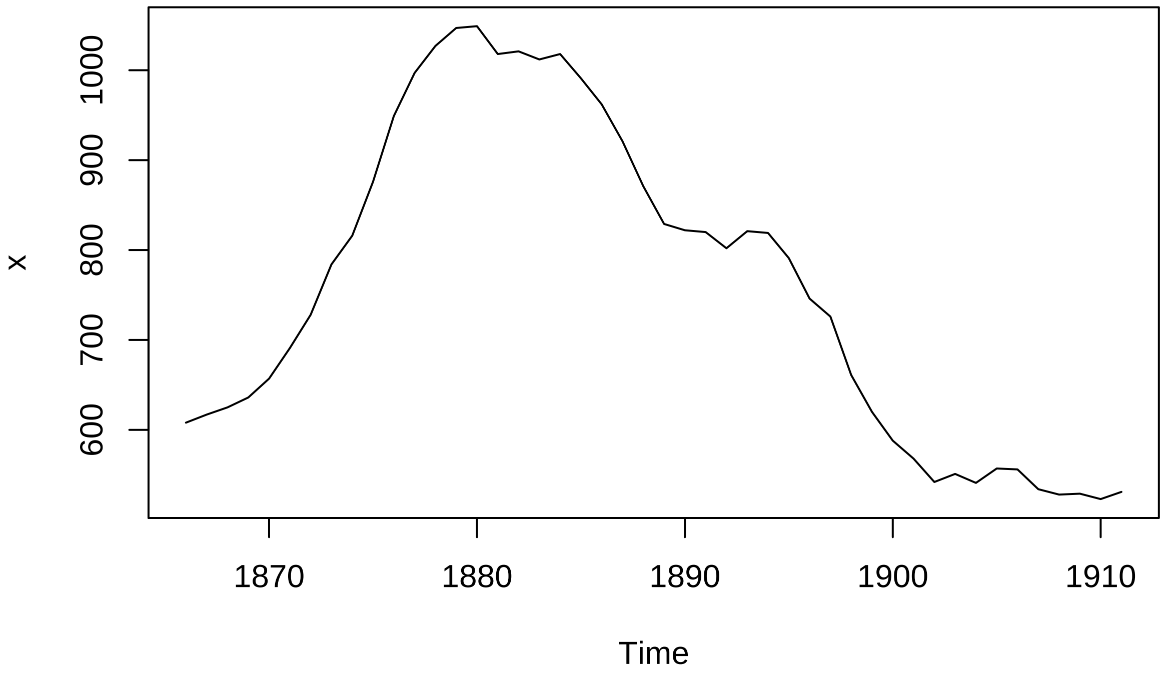
Forecast Model : $\hat{y}_{t+h|t} \;=\; l_t \;+\; h * b_t \;+\; \epsilon_t$

Updation Model : $l_t \;=\; \alpha * y_t \;+\; (1-\alpha) * (l_{t-1} \;+\; b_{t-1}$

$b_t \;=\; \beta * (l_t \;-\; l_{t-1}) \;+\; (1-\beta) * b_{t-1}$

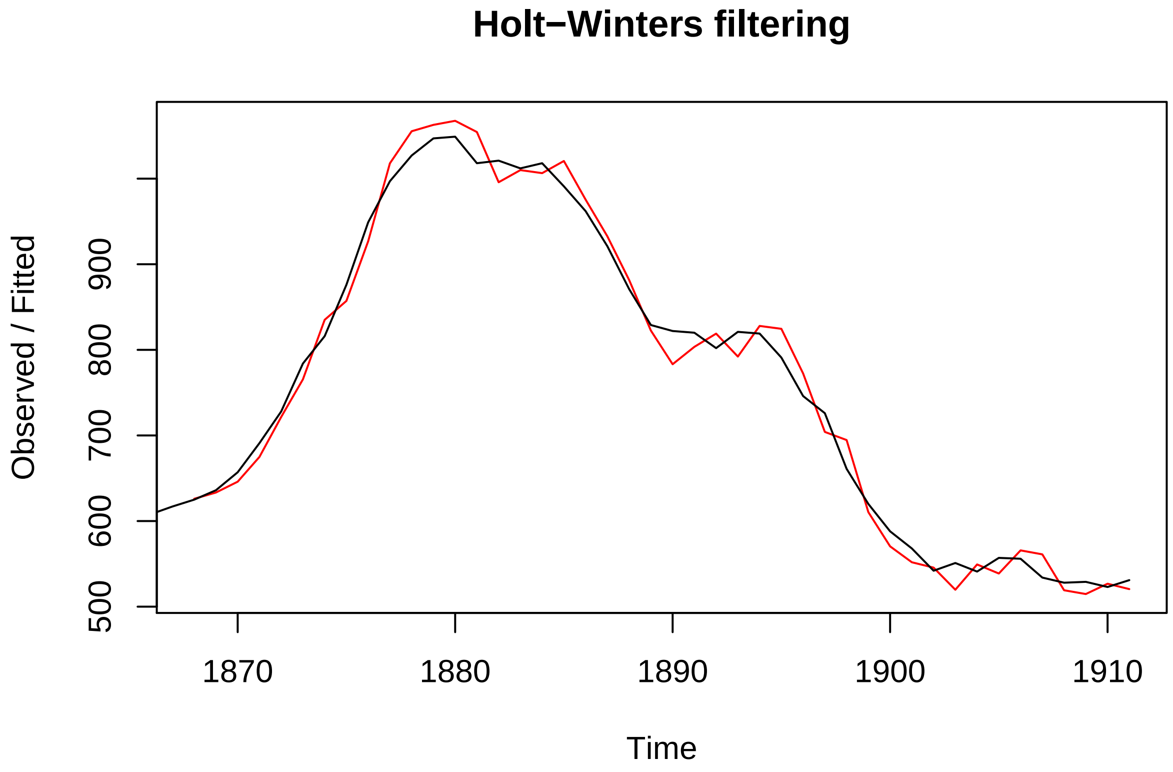
Lets look at an example for the same in R. We'll again be using function `HoltWinters`. This time there is both level and linear trend component, hence we'll set only seasonality parameter `gamma` to FALSE.

```
skirts=read.csv("skirts.csv")
skirts=ts(skirts,start=c(1866))
plot(skirts)
```



As you can see this time series has trend and a simple "level" forecast will not be enough.

```
skirtforecast=HoltWinters(skirts,gamma=F)
plot(skirtforecast)
```



```
skirtforecast
```

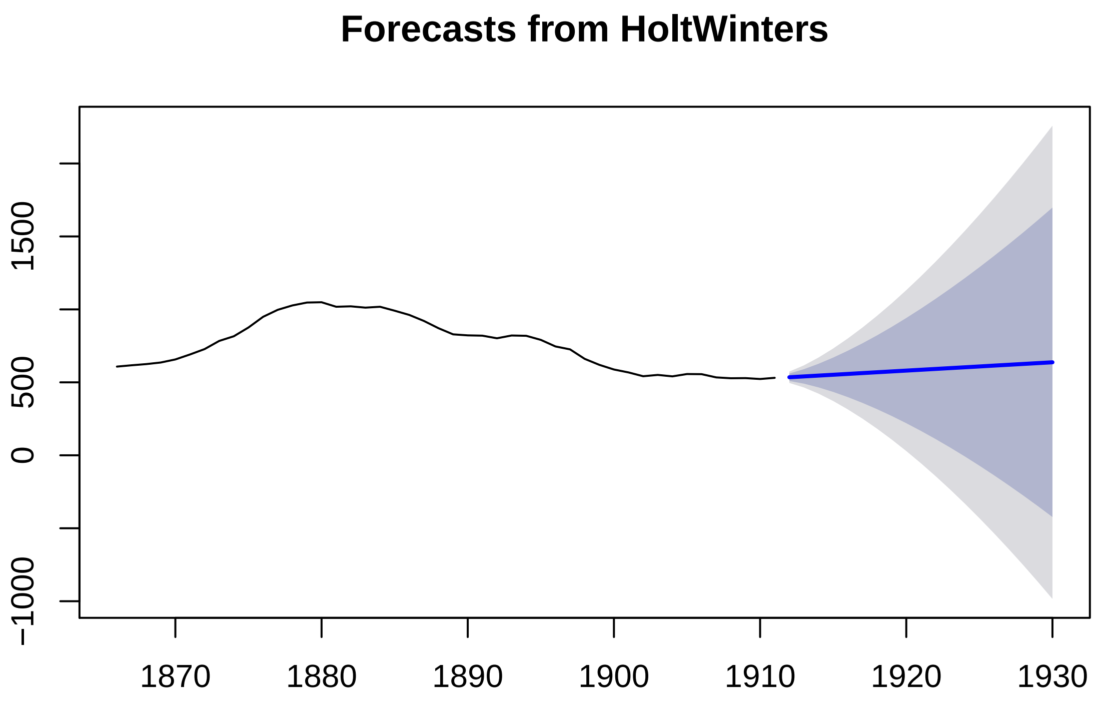
```
## Holt-Winters exponential smoothing with trend and without seasonal component.
```

```
##
## Call:
## HoltWinters(x = skirts, gamma = F)
##
## Smoothing parameters:
##  alpha: 0.8383481
##  beta : 1
##  gamma: FALSE
##
## Coefficients:
##         [,1]
## a 529.308585
## b   5.690464
```

Your forecast equation here is nothing but `a+b*h`, which is : `529.308585+h*5.690464`. Keep in mind that trend has been changing a lot for this data, it will not be wise to use this forecast equation to make forecast too long in future.

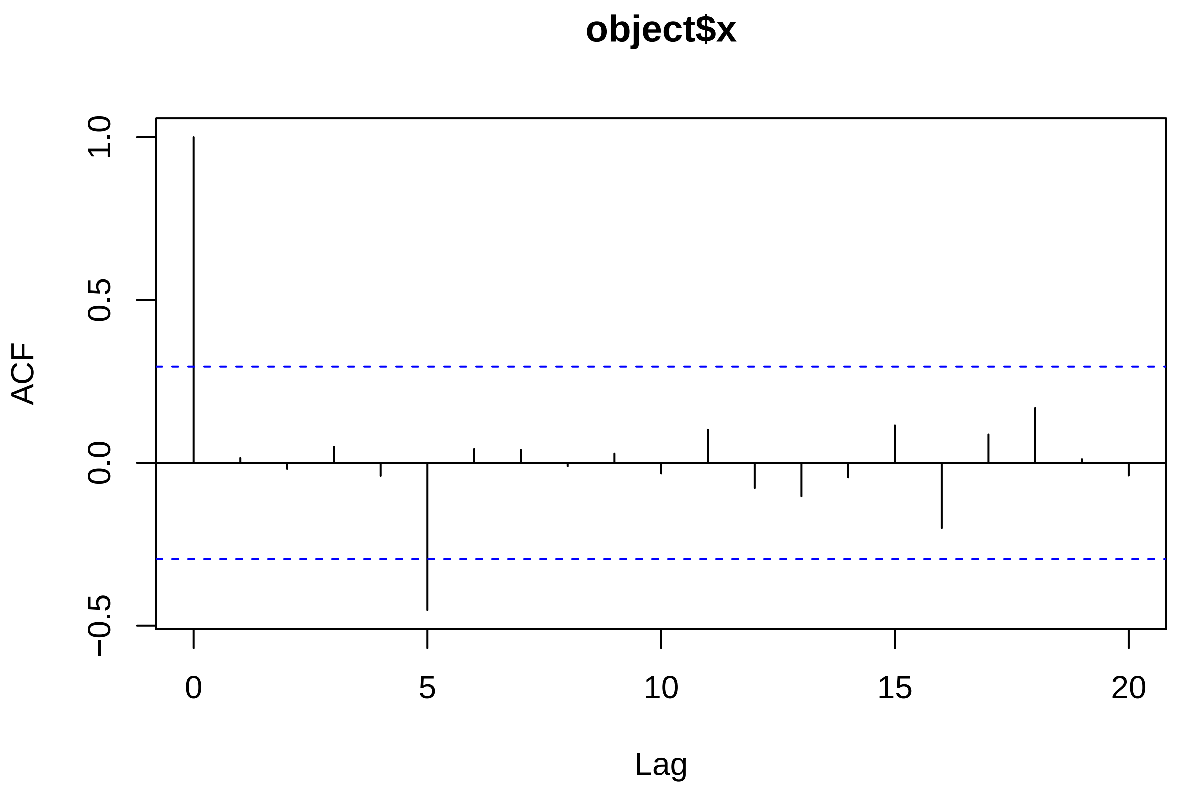
```
skirtfuture=forecast.HoltWinters(skirtforecast,h=19)

plot(skirtfuture)
```



Lets do model adequacy check as before.

```r
acf(skirtfuture$residuals,lag.max=20)
```



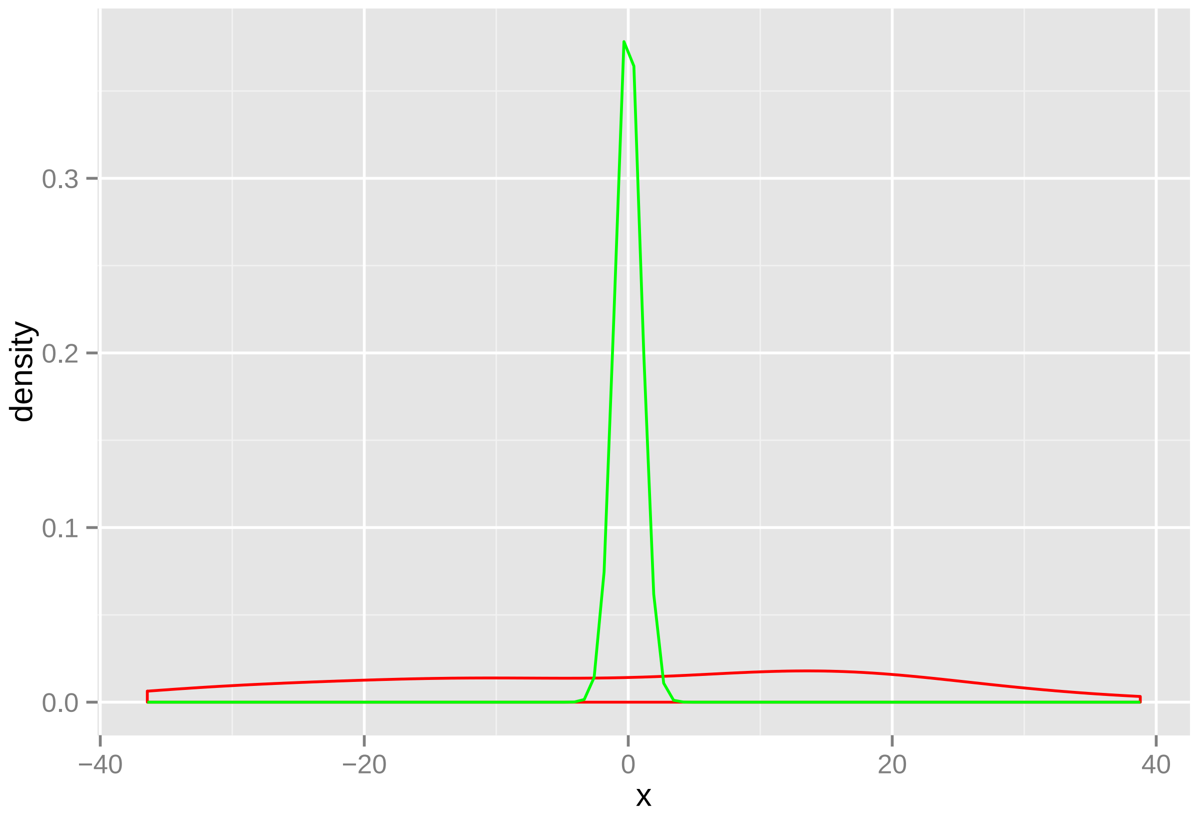
This indicates that there might be some auto-correlation present. To see whether this correlation is significant or not, we'll do box-ljung test.

```r
Box.test(skirtfuture$residuals,lag=20,type=c("Ljung-Box"))
```

```
##
##  Box-Ljung test
##
## data:  skirtfuture$residuals
## X-squared = 19.731, df = 20, p-value = 0.4749
```

Which tells you that this auto-correlation is not significant. Lets look at density plots to see if normality assumption holds.

```r
d=data.frame(x=as.vector(skirtfuture$residuals))
ggplot(d,aes(x))+
  geom_density(color="red")+
  stat_function(fun=dnorm,aes(x=x),color="green")
```



This again indicates that assumption of normality doesn't hold and we should be wary of confidence intervals provided.

Next we look at models where all three kind of patterns are present.

Forecast Model : $\hat{y}_{t+h|t} \;=\; l_t \;+\; h * b_t \;+\; s_{t-m} \;+\; \epsilon_t$

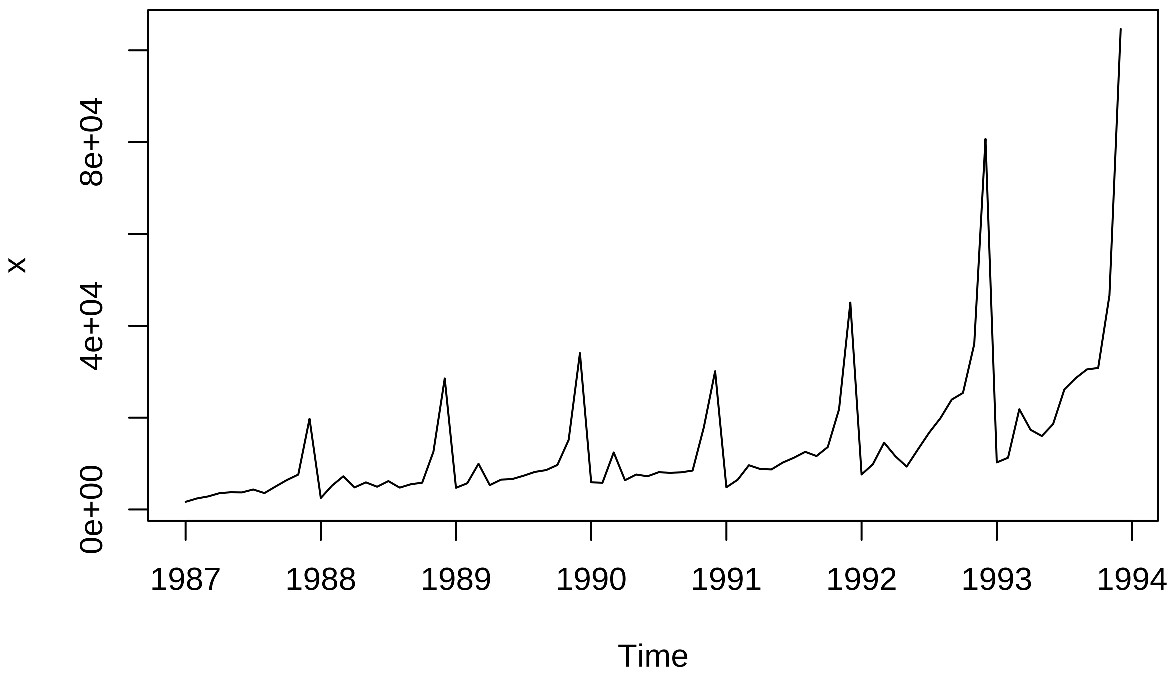
Updation Model : $l_t \;=\; \alpha * (y_t \;-\; s_{t-m}) + \;(1-\alpha) * (l_{t-1} \;+\; b_{t-1}$

$b_t \;=\; \beta * (l_t \;-\; l_{t-1}) \;+\; (1-\beta) * b_{t-1}$

$s_t \;=\; \gamma * (y_t \;-\; l_{t-1} \;-\; b_{t-1}) \;+\; (1-\gamma) * s_{t-m}$

When you have this kind of data, you have to mention seasonality with option `frequency` while creating the time series object from your data. Since seasonality is 12 in this case, you also need to mention which month the first observation in the data belong to in the option `start`.

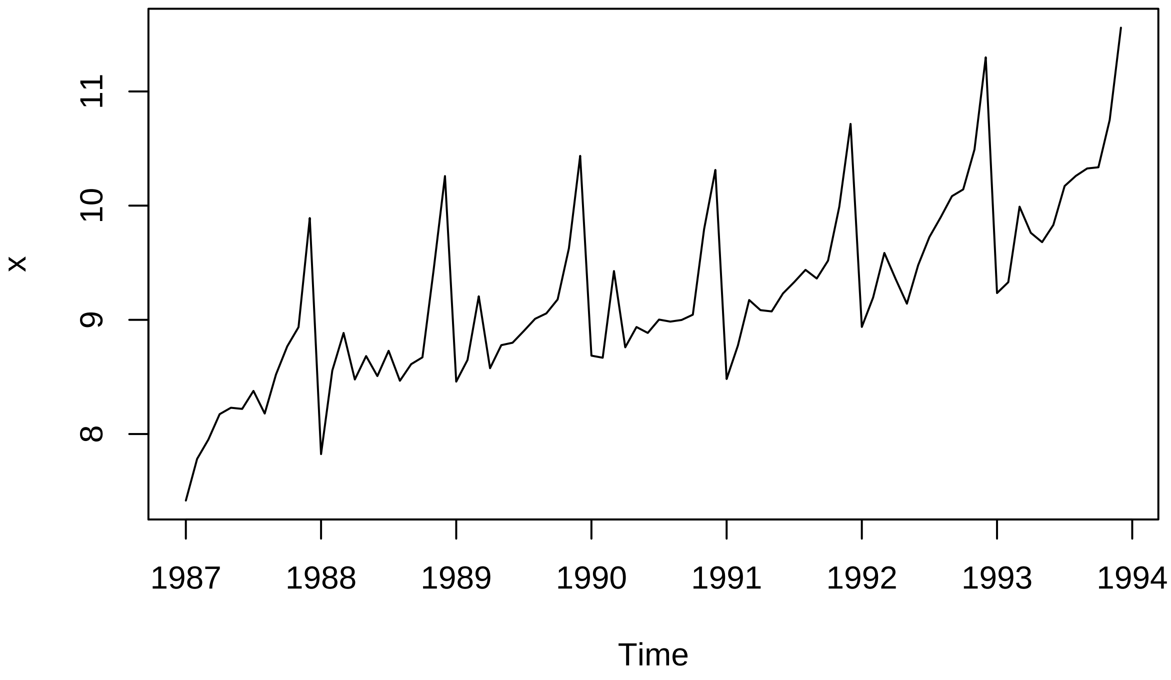
```
souvenir=read.csv("souvenir.csv")
souvenirts = ts(souvenir, frequency=12, start=c(1987,1))
plot(souvenirts)
```



Here the trend is not linear , also the effect of seasonality is not additive, we can take care of that by log transforming our data.

```
souvenirts=log(souvenirts)

plot(souvenirts)
```



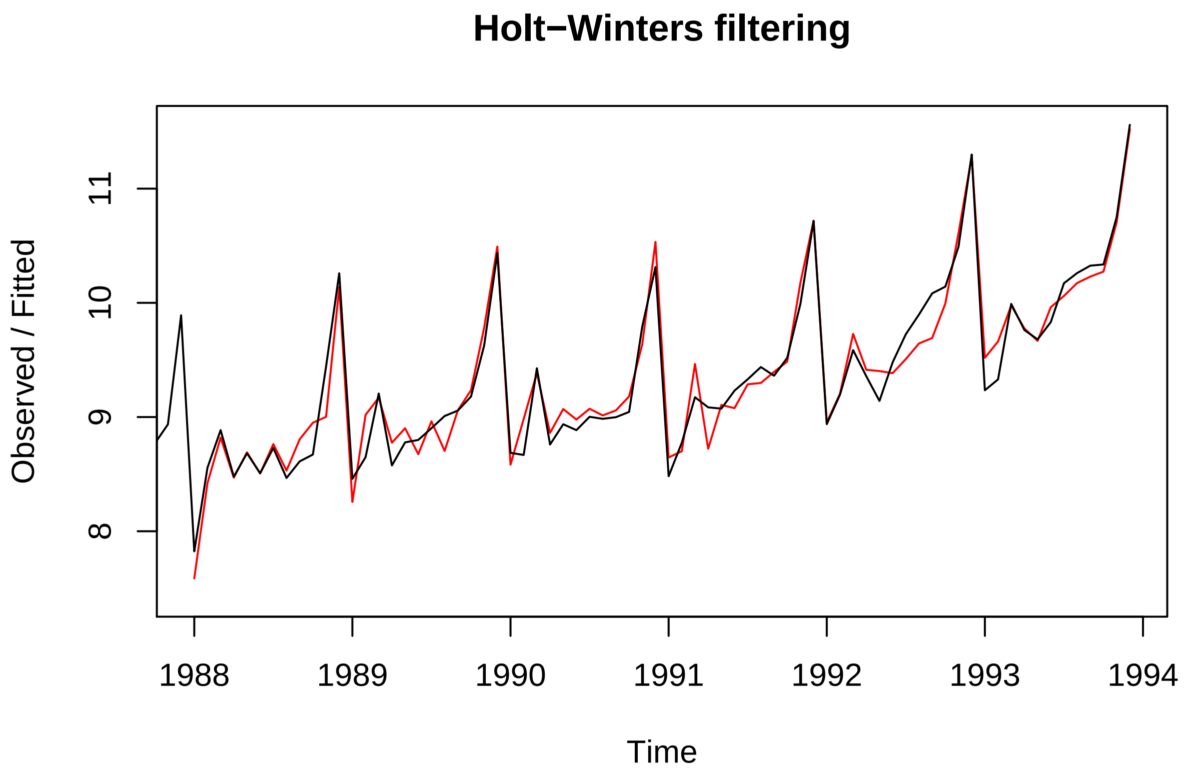
This looks alright. Lets model this data with function `HoltWinters` . Now all three components are present, we don't have to set any parameter to FALSE.

```r
souvenirforecast=HoltWinters(souvenirts)

souvenirforecast
```

```
## Holt-Winters exponential smoothing with trend and additive seasonal component.
##
## Call:
## HoltWinters(x = souvenirts)
##
## Smoothing parameters:
##  alpha: 0.413418
##  beta : 0
##  gamma: 0.9561275
##
## Coefficients:
##             [,1]
## a   10.37661961
## b    0.02996319
## s1  -0.80952063
## s2  -0.60576477
## s3   0.01103238
## s4  -0.24160551
## s5  -0.35933517
## s6  -0.18076683
## s7   0.07788605
## s8   0.10147055
## s9   0.09649353
## s10  0.05197826
## s11  0.41793637
## s12  1.18088423
```

```r
plot(souvenirforecast)
```

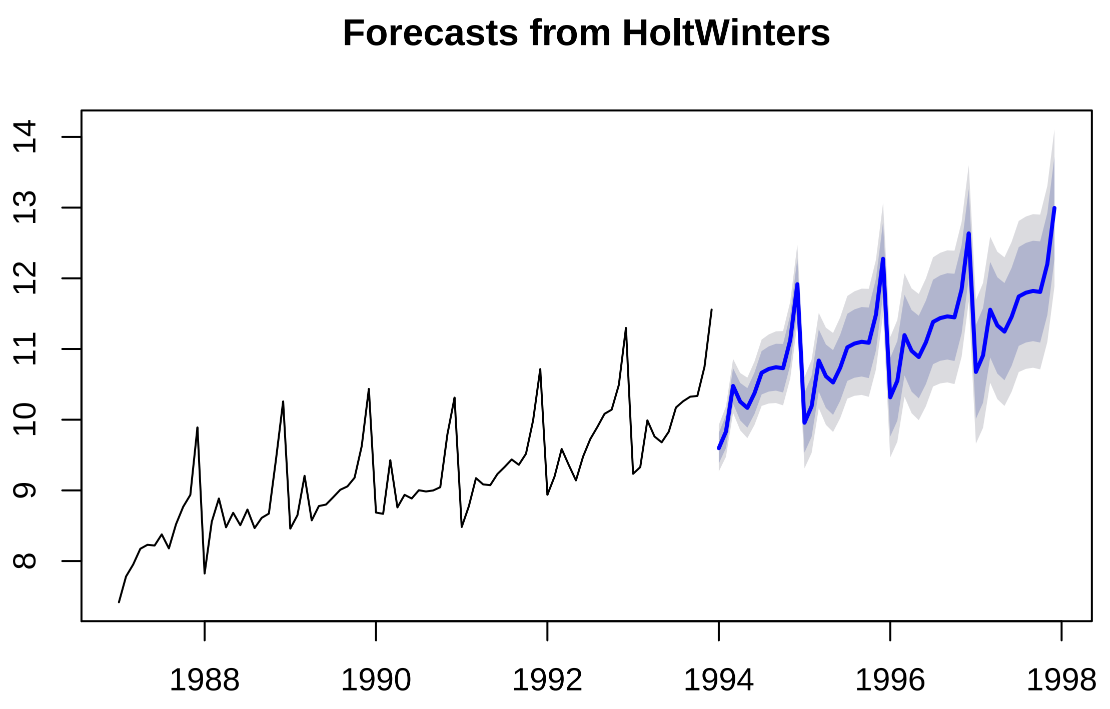


The value of beta is 0.00, indicating that the estimate of the slope b of the trend component is not updated over the time series, and instead is set equal to its initial value.

This makes good intuitive sense, as the level changes quite a bit over the time series, but the slope b of the trend component remains roughly the same. Lets make forecasts now.

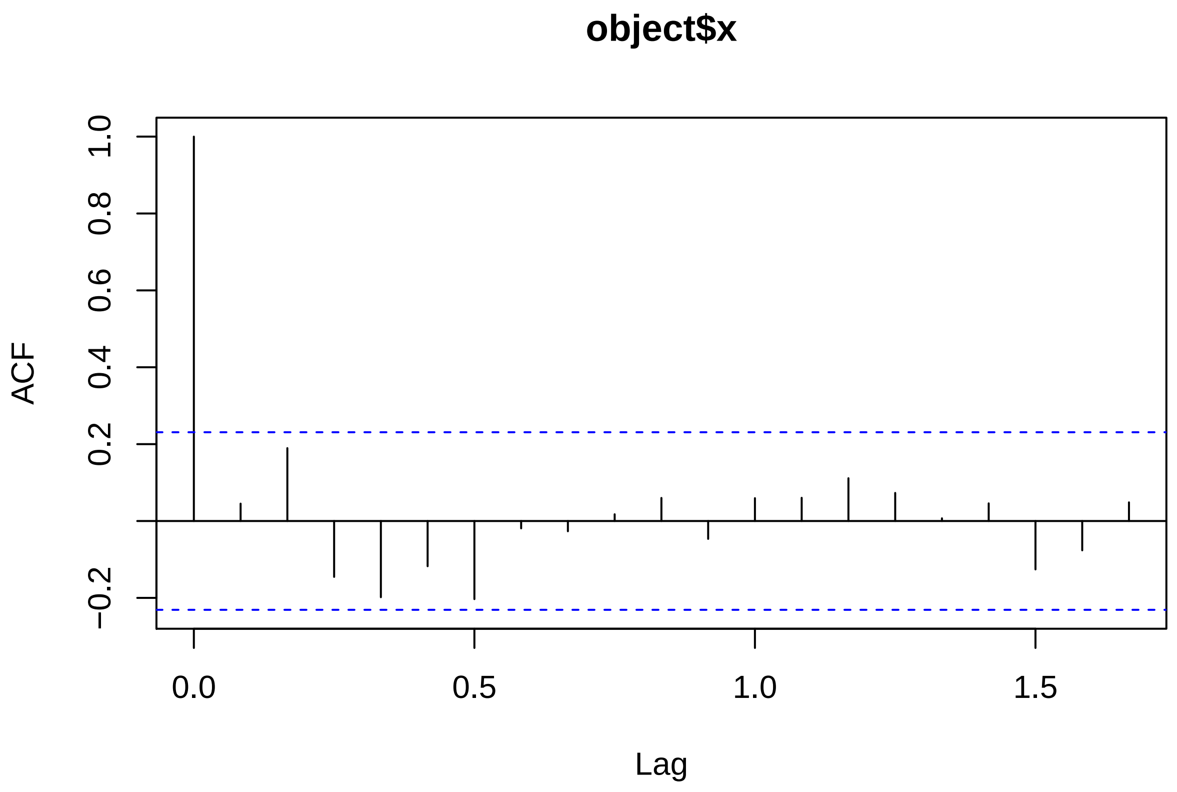
```
souvenirfuture=forecast.HoltWinters(souvenirforecast,h=48)

plot(souvenirfuture)
```



Lets examine adequacy of the model.

```
acf(souvenirfuture$residuals,lag.max=20)
```



No auto-correlation, good, lets look at normality bit.

```r
shapiro.test(souvenirfuture$residuals)
```

```
##
##  Shapiro-Wilk normality test
##
## data:  souvenirfuture$residuals
## W = 0.98618, p-value = 0.6187
```

Everything looks alright here too. Next we discuss more general time series models which do not assume that your data depends on all previous values.

AR, MA and ARIMA models

AR or Autoregressive models of order p assume that data values depend on its last p-value only. This is how you right an AR(p) model.

$$y_t \ = \ \psi_1 y_{t-1} \ + \ \psi_2 y_{t-2} \ + ...\psi_p y_{t-p} \ + \ \epsilon_t$$

You can write the same equation with the help of backshift operator as well. backshift operator shifts back the value by some time period. It is denoted by `B`.

$$B^k * y_t \ = \ y_{t-k}$$

writing AR(p) equation in terms of backshift operator:

$$(1 \ - \ B\psi_1 \ - \ B^2\psi_2 \ - \ .....B^p\psi_p) * y_t \ = \ \epsilon_t$$

MA or moving average model of order q assume that current data value is dependent on q previous error terms [or random shocks]. This is how we write this with backshift operator.

$$y_t \ = \ (1 \ + \ B\theta_1 \ + \ B^2\theta_2 \ + \ ......B^q\theta_q) * \epsilon_t$$

AR(p) models are considered to be deterministic where as MA(q) models are considered to be stochastic models. Any general time series can be expressed as an addition of an MA(q) and AR(p) series , given that it is stationary. [Wold's Theorem]

Any series can be made stationary by differencing. Combining AR(p) and MA(q) models gives rise to ARIMA(p,d,q) models where d is the order of differencing needed to achieve stationarity.

A general ARIMA(p,d,q) model written with backshift operator:

$$(1 \ - \ B\psi_1 \ - \ B^2\psi_2 \ - \ .....B^p\psi_p) * (1 \ - \ B^d) * y_t \ = \ (1 \ + \ B\theta_1 \ + \ B^2\theta_2 \ + \ ......B^q\theta_q) * \epsilon_t$$

An even general ARIMA model will be the one with seasonality component as well. This is called a SARIMA(p,d,q)(P,D,Q)[m] model. where (p,d,q) is the order of non seasonal ARIMA component. (P,D,Q) is order of seasonal ARIMA component. m is period of seasonality. A general SARIMA model is written as:

$$(1 \ - \ B\psi_1 \ - \ B^2\psi_2 \ - \ .....B^p\psi_p) * (1 \ - \ B^d) *$$
$$(1 \ - \ B^m\Psi_1 \ - \ B^{2m}\Psi_2 \ - \ .....B^{Pm}\Psi_P) * (1 \ - \ B^{Dm}) * y_t$$

$$=$$

$$(1\ +\ B\theta_1\ +\ B^2\theta_2\ +\ ......B^q\theta_q)*$$

$$(1\ +\ B^m\Theta_1\ +\ B^{2m}\Theta_2\ +\ ......B^{Qm}\Theta_Q)*\epsilon_t$$

You don't need to worry about finding order of ARIMA or SARIMA model. Function `auto.arima` in the library `forecast` gives you this automatically. Don't bother with long and vague articles on guessing orders by looking at acf and pacf plots.

```
auto.arima(souvenirts)
```

```
## Series: souvenirts
## ARIMA(2,0,0)(0,1,1)[12]
##
## Coefficients:
##          ar1     ar2     sma1
##       0.4860  0.4894  -0.4922
## s.e.  0.1013  0.1028   0.1622
##
## sigma^2 estimated as 0.03099:  log likelihood=20.3
## AIC=-32.6   AICc=-32   BIC=-23.49
```

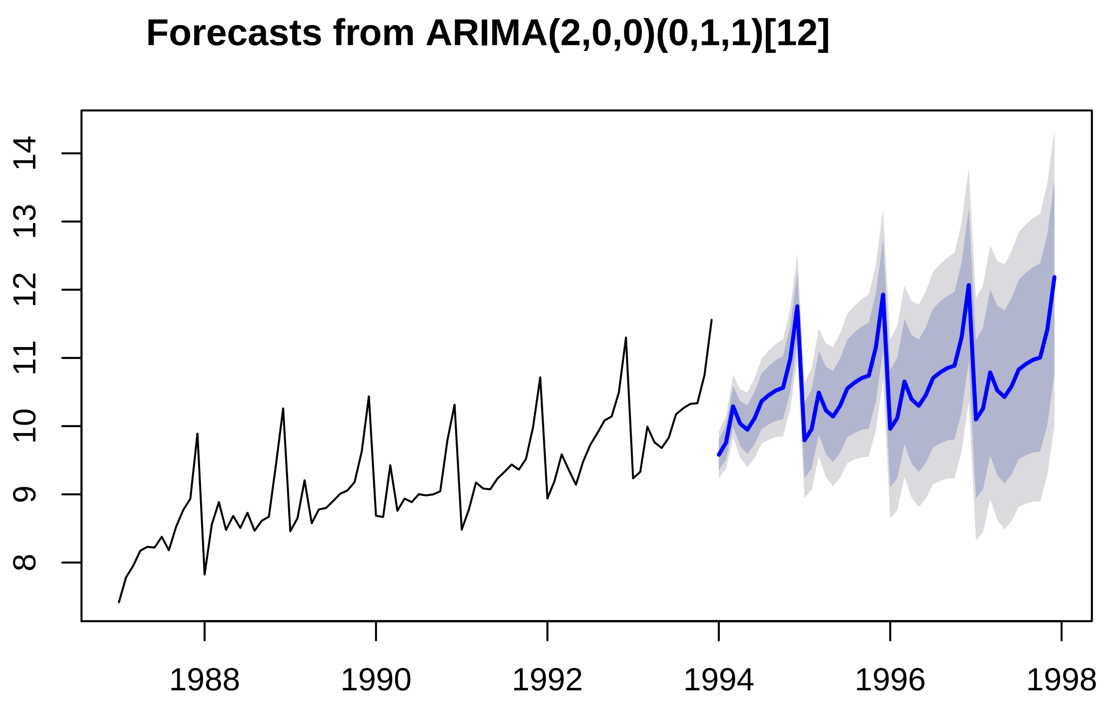
This turns out to be a SARIMA model with , p=2, d=0, q=0 , P=0 ,D=1 ,Q=1 and m=12. Simplified equation for this will be:

$$(1\ -\ B\psi_1\ -\ B^2\psi_2)*(1\ -\ B^{12})*y_t\ =(1\ +\ B^{12}\Theta_1)*\epsilon_t$$

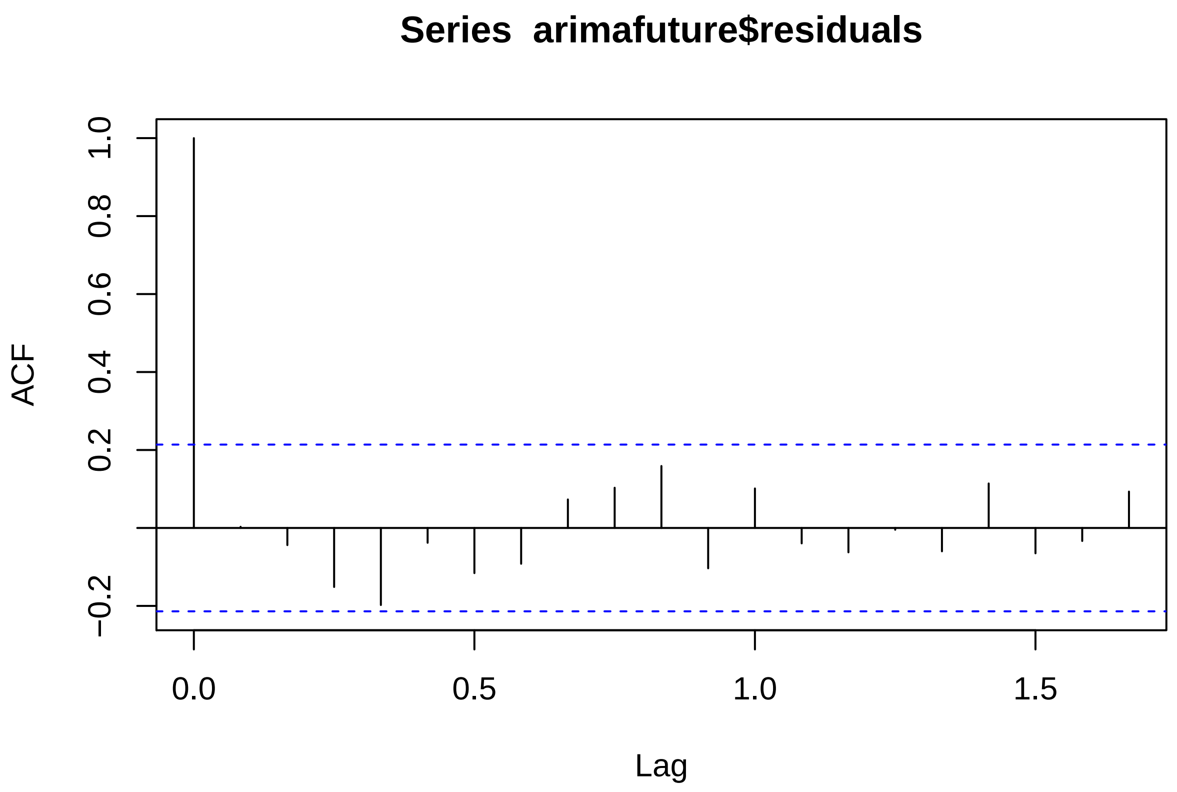
where values of ψ_1,ψ_2 and Θ_1 are 0.486 , 0.4894 and -0.4922 respectively. Having order of the SARIMA model, you can use function `arima` in the base package also.

```
arimafit=arima(souvenirts,order=c(2,0,0),seasonal=c(0,1,1))

arimafuture=forecast.Arima(arimafit,h=48)
plot(arimafuture)
```



```r
acf(arimafuture$residuals,lag.max=20)
```



```r
shapiro.test(arimafuture$residuals)
```

```
##
##  Shapiro-Wilk normality test
```

```
##
## data:  arimafuture$residuals
## W = 0.98509, p-value = 0.4446
```

We'll conclude here. In case of doubts , please take to QA forum.

Prepared By : Lalit Sachan

Contact : lalit.sachan@edvancer.in