Time_Series_Analysis

```
library(astsa, quietly=TRUE, warn.conflicts=FALSE)
require(knitr)
```

```
## Loading required package: knitr
```

```
library(ggplot2)
```

```
## Registered S3 methods overwritten by 'ggplot2':
##   method         from
##   [.quosures     rlang
##   c.quosures     rlang
##   print.quosures rlang
```

Chapter 1 : Time Series.

Step 1 : Reading

```
#Read in the data and skip the first three lines.
kings<-scan('http://robjhyndman.com/tsdldata/misc/kings.dat', skip=3)
kings
```

```
##  [1] 60 43 67 50 56 42 50 65 68 43 65 34 47 34 49 41 13 35 53 56 16 43 69
## [24] 59 48 59 86 55 68 51 33 49 67 77 81 67 71 81 68 70 77 56
```

```
#This dataset records the age of death for 42 successive kings of England.
```

STEP 2 : Convert the data into a time series

It is best to convert the data into an R time series object after you have successfully loaded the data.

```
kings <- ts(kings)
kings
```

```
## Time Series:
## Start = 1
## End = 42
## Frequency = 1
##  [1] 60 43 67 50 56 42 50 65 68 43 65 34 47 34 49 41 13 35 53 56 16 43 69
## [24] 59 48 59 86 55 68 51 33 49 67 77 81 67 71 81 68 70 77 56
```

However, it is common to come across time series that have been collected at regular intervals that are less than the one year of the kings dataset, for example, monthly, weekly or quarterly. In these cases we can specify the number of times that data was collected per year by using the frequency parameter in the ts( ) function. For monthly data, we set frequency = 12. We can also specify the first year that the data were collected and the first interval in that year by using the 'stat' parameter. For example, the third quarter of 1909 would be 'start = c(1909, 3).

STEP 3 : Next we load in a dataset of number of births per month in New York city, from January 1946 to December 1958.

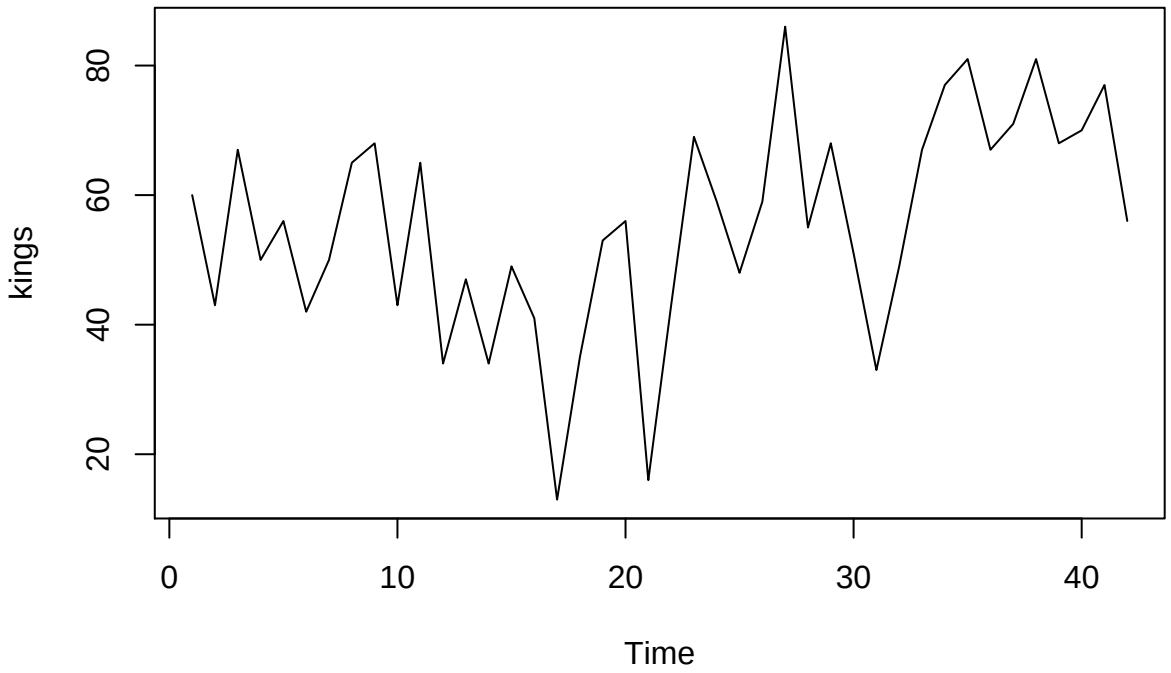
```
births <- scan("http://robjhyndman.com/tsdldata/data/nybirths.dat")

births <- ts(births, frequency = 12, start = c(1946, 1))
births
```

```
##          Jan    Feb    Mar    Apr    May    Jun    Jul    Aug    Sep    Oct
## 1946 26.663 23.598 26.931 24.740 25.806 24.364 24.477 23.901 23.175 23.227
## 1947 21.439 21.089 23.709 21.669 21.752 20.761 23.479 23.824 23.105 23.110
## 1948 21.937 20.035 23.590 21.672 22.222 22.123 23.950 23.504 22.238 23.142
## 1949 21.548 20.000 22.424 20.615 21.761 22.874 24.104 23.748 23.262 22.907
## 1950 22.604 20.894 24.677 23.673 25.320 23.583 24.671 24.454 24.122 24.252
## 1951 23.287 23.049 25.076 24.037 24.430 24.667 26.451 25.618 25.014 25.110
## 1952 23.798 22.270 24.775 22.646 23.988 24.737 26.276 25.816 25.210 25.199
## 1953 24.364 22.644 25.565 24.062 25.431 24.635 27.009 26.606 26.268 26.462
## 1954 24.657 23.304 26.982 26.199 27.210 26.122 26.706 26.878 26.152 26.379
## 1955 24.990 24.239 26.721 23.475 24.767 26.219 28.361 28.599 27.914 27.784
## 1956 26.217 24.218 27.914 26.975 28.527 27.139 28.982 28.169 28.056 29.136
## 1957 26.589 24.848 27.543 26.896 28.878 27.390 28.065 28.141 29.048 28.484
## 1958 27.132 24.924 28.963 26.589 27.931 28.009 29.229 28.759 28.405 27.945
## 1959 26.076 25.286 27.660 25.951 26.398 25.565 28.865 30.000 29.261 29.012
##          Nov    Dec
## 1946 21.672 21.870
## 1947 21.759 22.073
## 1948 21.059 21.573
## 1949 21.519 22.025
## 1950 22.084 22.991
## 1951 22.964 23.981
## 1952 23.162 24.707
## 1953 25.246 25.180
## 1954 24.712 25.688
## 1955 25.693 26.881
## 1956 26.291 26.987
## 1957 26.634 27.735
## 1958 25.912 26.619
## 1959 26.992 27.897
```

STEP 4 : Plotting Time Series

The next step in any time series analysis is to plot the data

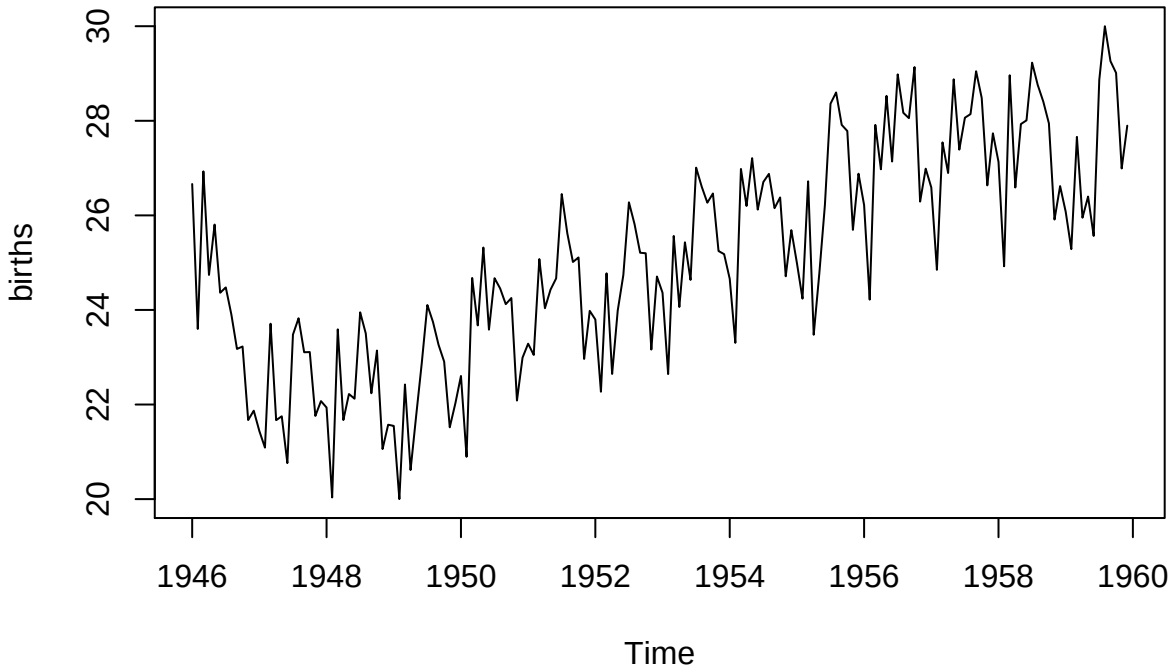
```
plot.ts(kings)
```



At this point we could guess that this time series could be described using an additive model, since the random fluctuations in the data are roughly constant in size over time.

Let's look at the births data.

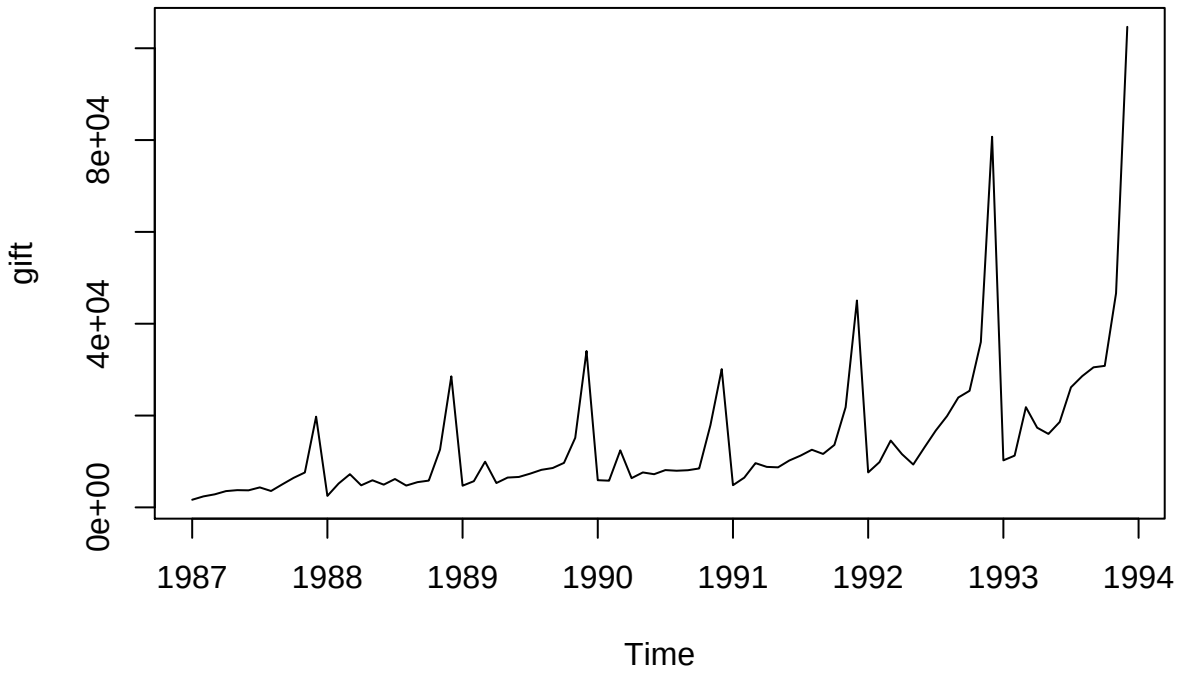
```
plot.ts(births)
```



We can see from this time series that there is certainly some seasonal variation in the number of births per month; there is a peak every summer, and a trough every winter. Again the it seems like this could be described using an additive model, as the seasonal fluctuations are roughly constant in size over time and do not seem to depend on the level of the time series, and the random fluctuations seem constant over time.

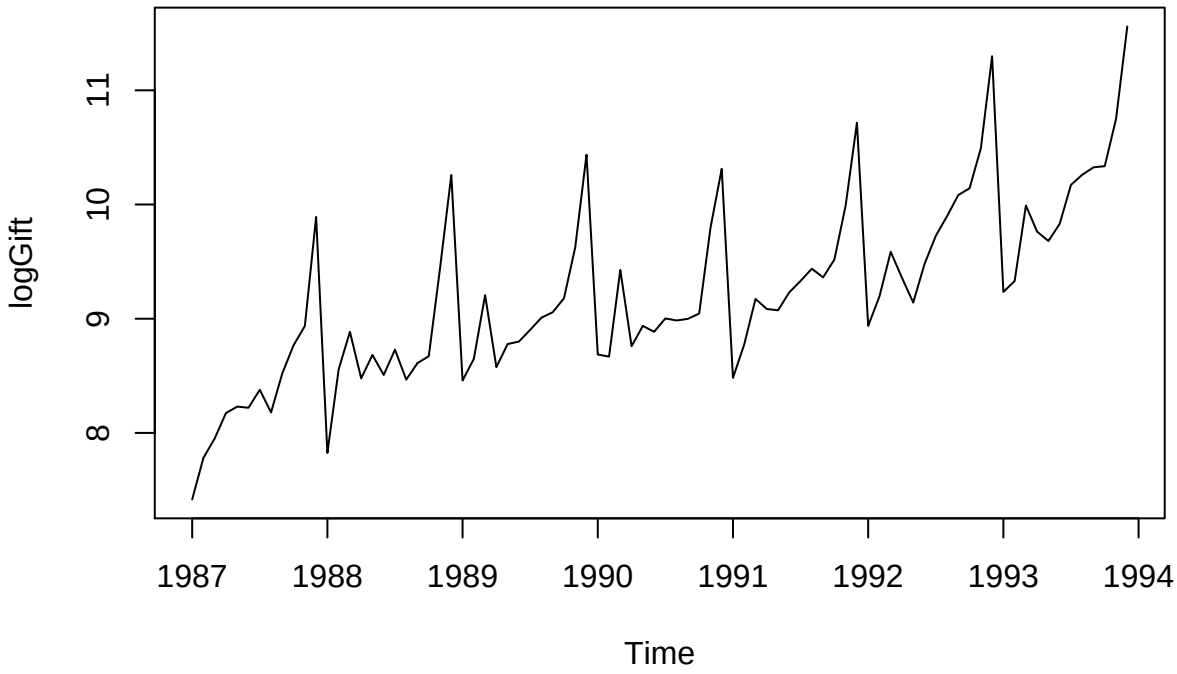
How about a time series of a beach town souvenir shop

```
gift <- scan("http://robjhyndman.com/tsdldata/data/fancy.dat")
gift<- ts(gift, frequency=12, start=c(1987,1))

plot.ts(gift)
```



In this case, an additive model is not appropriate since the size of the seasonal and random fluctuations change over time and the level of the time series. It is then appropriate to transform the time series so that we can model the data with a classic additive model

```
logGift <- log(gift)
plot.ts(logGift)
```



STEP 6 : Decomposing Time Series Decomposing a time series means separating it into it's constituent

components, which are often a trend component and a random component, and if the data is seasonal, a seasonal component. ## Decomposing non-Seasonal Data Recall that non-seasonal time series consist of a trend component and a random component. Decomposing the time series involves tying to separate the time series into these individual components.

One way to do this is using some smoothing method, such as a simple moving average.

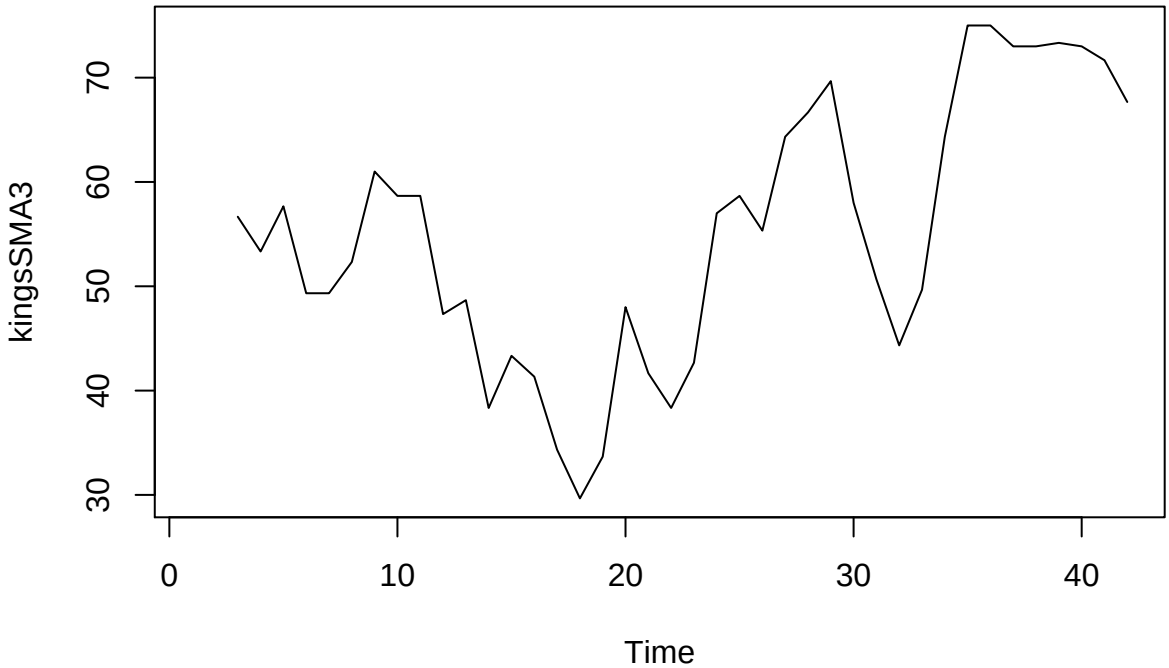
The SMA() function in the TTR R package can be used to smooth time series data using a moving average. The SMA function takes a span argument as n order. To calculate the moving average of order 5, we set n = 5.

Let's try to see a clearer picture of the Kings dataset trend component by applying an order 3 moving average.

```
library(TTR)
```

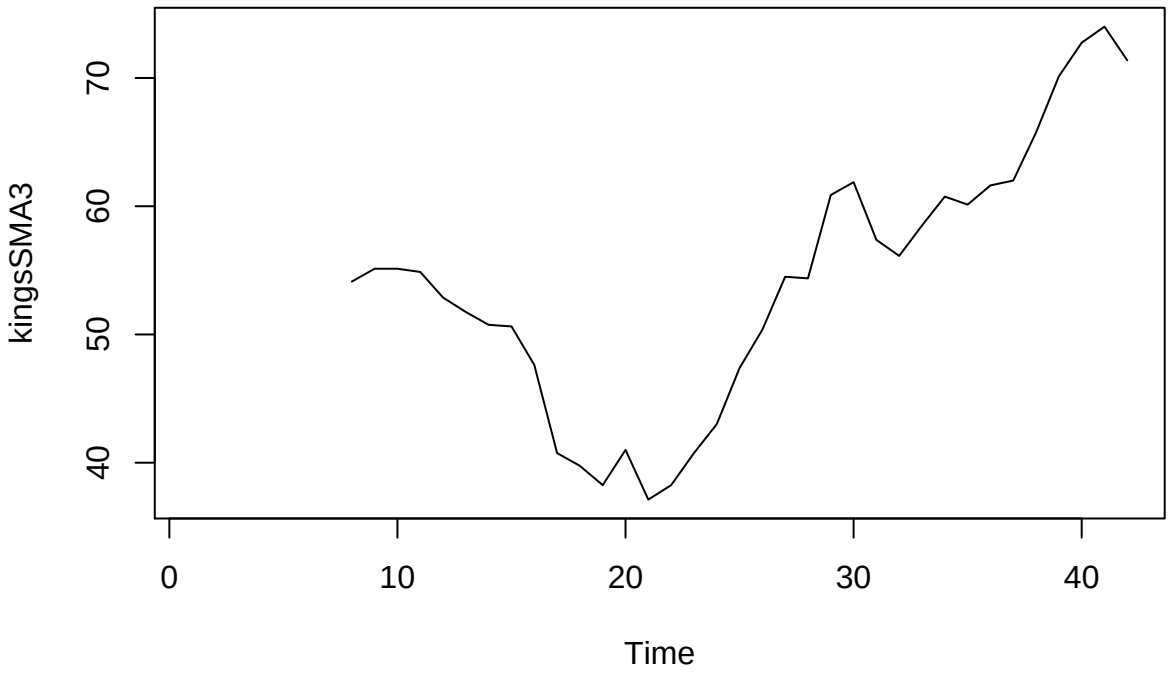
```
## Registered S3 method overwritten by 'xts':
##   method     from
##   as.zoo.xts zoo
```

```
kingsSMA3 <- SMA(kings, n=3)
plot.ts(kingsSMA3)
```



It seems like there is still some random fluctuations in the data, we might want to try a big larger of a smoother.

```
library(TTR)

kingsSMA3 <- SMA(kings, n=8)
plot.ts(kingsSMA3)
```



This is better, we can see that the death of English kings has declined from ~55 years to ~40 years for a brief period, followed by a rapid increase in the next 20 years to ages in the 70's.

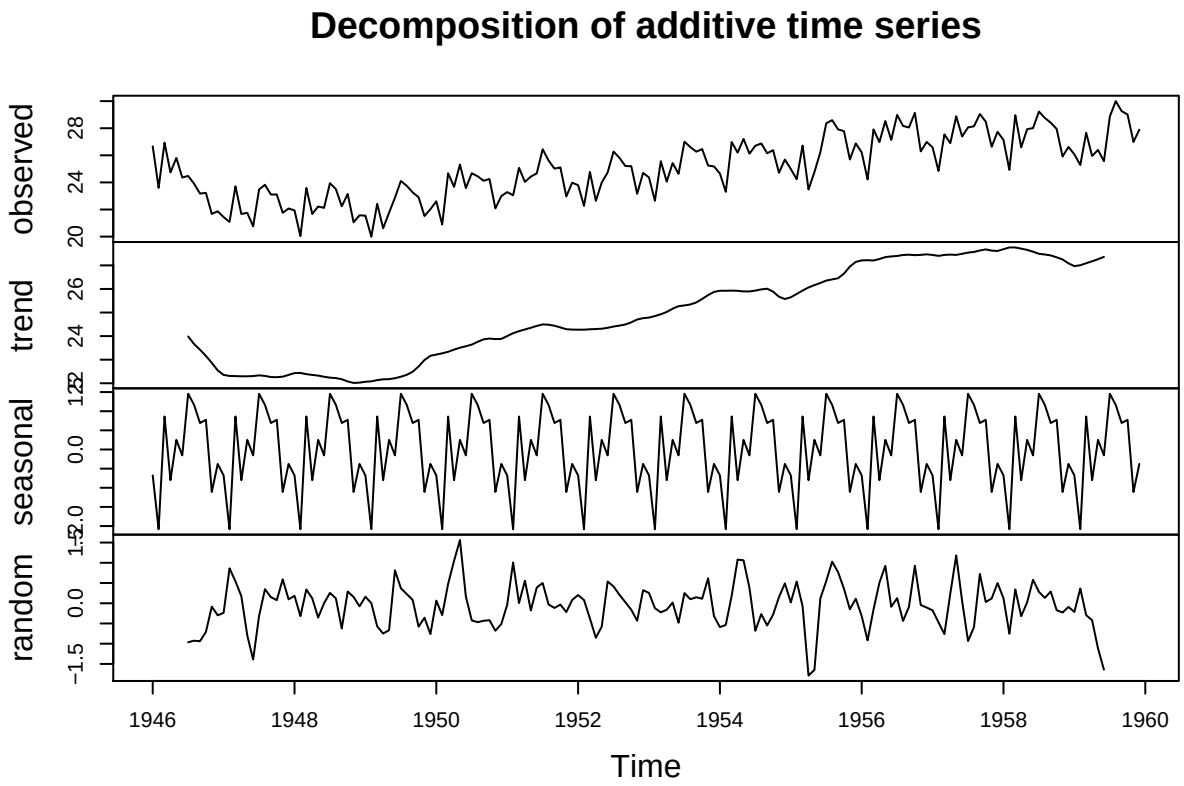
STEP 7 : Decomposing Seasonal Data

A seasonal time series, in addition to the trend and random components, also has a seasonal component. Decomposing a seasonal time series means separating the time series into these three components.

In R we can use the decompose() function to estimate the three components of the time series.

To estimate the trend, seasonal, and random components of the New York births dataset we can ...

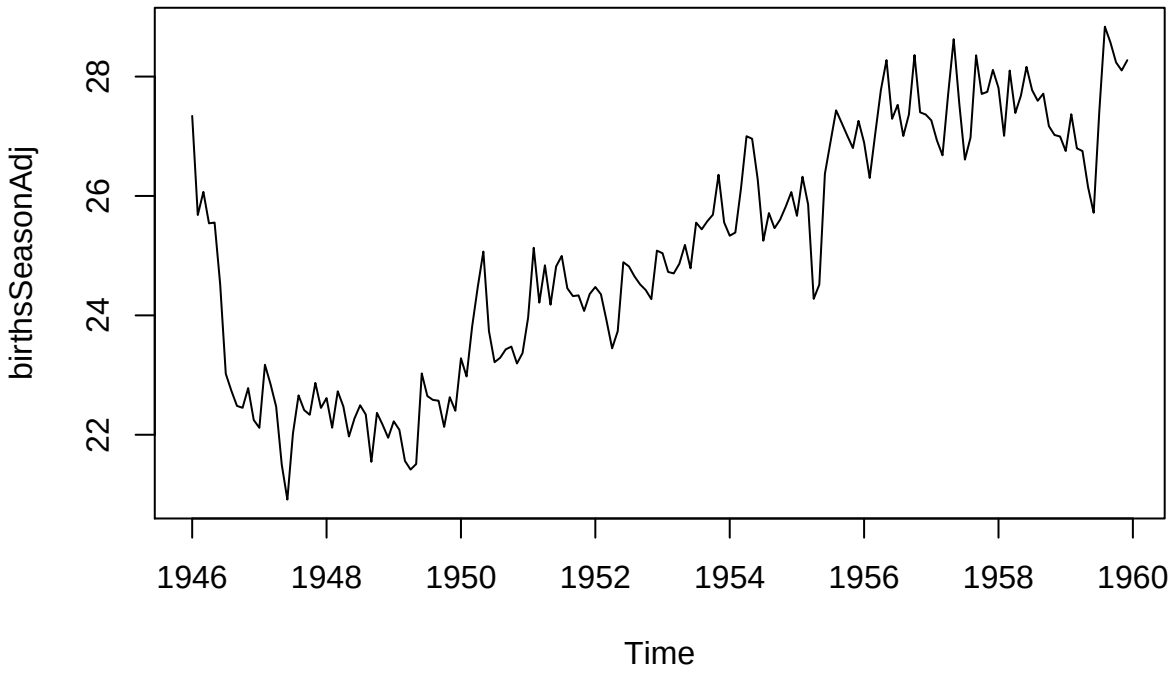
```
birthsComp <- decompose(births)
plot(birthsComp)
```



STEP 8 : Seasonally Adjusting

If you have a seasonal time series, you can seasonally adjust the series by estimating the seasonal component, and subtracting it from the original time series.

```
birthsSeasonAdj <- births - birthsComp$seasonal
plot(birthsSeasonAdj)
```



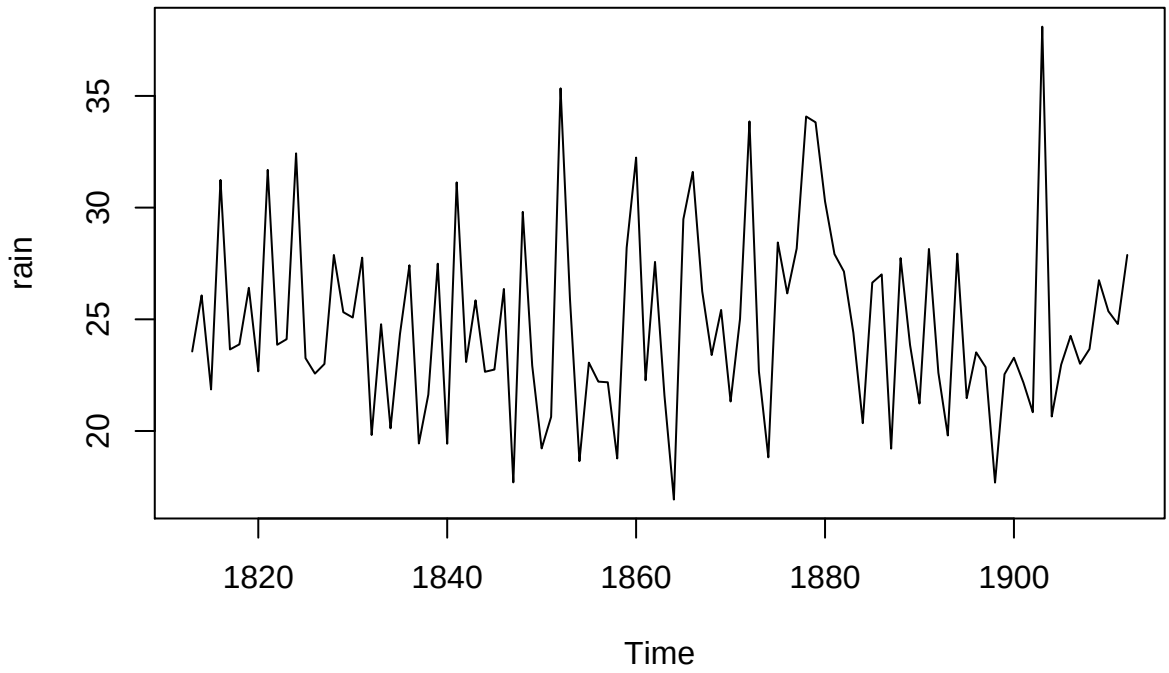
We can see now that time time series simply consists of the trend and random components.

STEP 9: Forecasts using Exponential Smoothing

Exponential smoothing is a common method for making short-term forecasts in time series data. ## Simple Exponential Smoothing If you have a time series with constant level and no seasonality, you can use simple exponential smoothing for short term forecasts.

This method is a way of estimating the level at the current time point. Smoothing is controlled by the parameter alpha for the estimate of the level at the current time point. The value of alpha lies between 0 and 1. Values of alpha close to 0 mean that little weight is places on the most recent observations when making forecasts of future values. ### Example 1 Total annual rainfall in inches for London, from 1813 - 1912.

```
rain <- ts(scan("http://robjhyndman.com/tsdldata/hurst/precip1.dat",skip=1), start=c(1813))
plot.ts(rain)
```



You can see from the plat that there is roughly constant variance over time, thus we can describe this as an additive model, thus we can make forecasts using simple exponential smoothing.

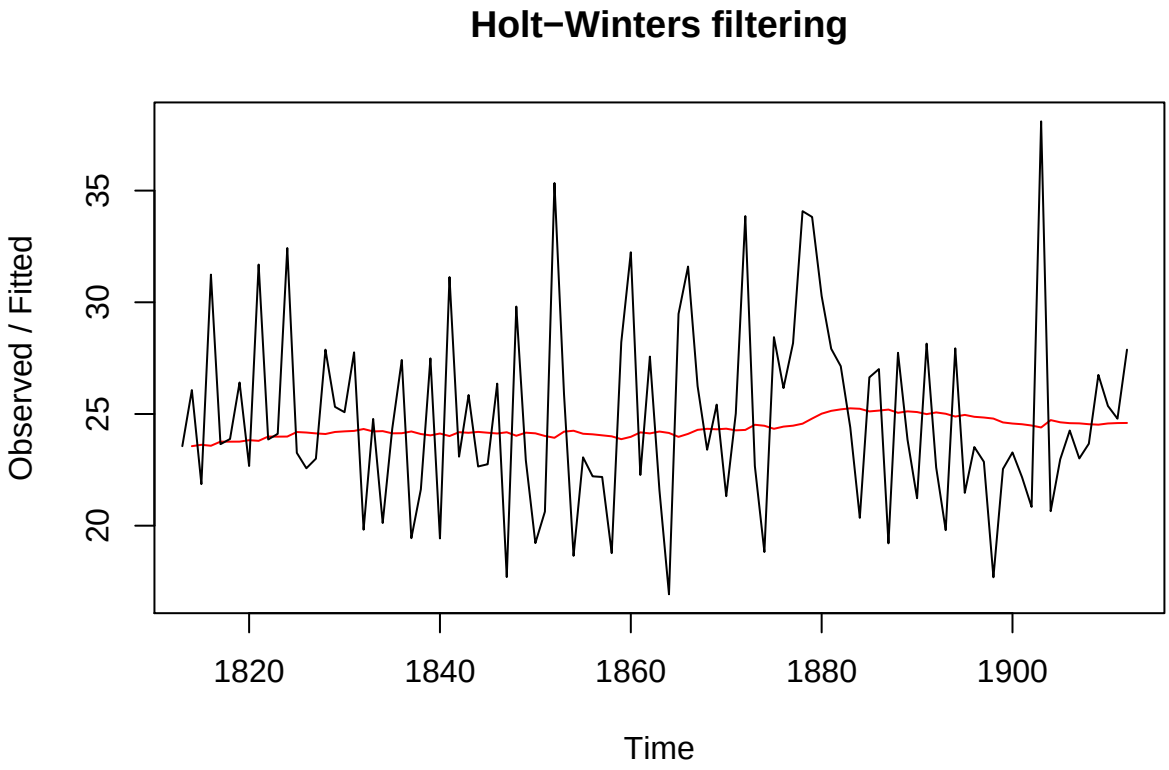
In R we can use the HoltWinters() function. For simple exponential smoothing, we need to set the parameters beta = FALSE and gamma = FALSE.

```
rainF <- HoltWinters(rain, beta=FALSE, gamma = FALSE)
rainF
```

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

The output tells us that the estimated value of the alpha parameter is about 0.024, which is very close to zero. This means that the forecasts are based on both recent and less recent observations, though there is more weight placed on recent observations. By default, HoltWinters makes forecasts for the total time period covered in the original series. We can get the fitted values of the forecasts and plot them simply by calling plot() on our variable storing the model we fit above.

```
plot(rainF)
```



To test the accuracy, we can calculate the SSE for the in-sample forecast errors.

```
rainF$SSE
```

```
## [1] 1828.855
```

If we want to make predictions into the future (past 1912) we can utilize the forecast.HoltWinters() function from the R package forecast. We specify how many predictions into the future we will make with the h parameter. For example: to make predictions into the next 8 years we would type:

```
library(forecast)
```

```
## Registered S3 method overwritten by 'quantmod':
##   method            from
##   as.zoo.data.frame zoo
```

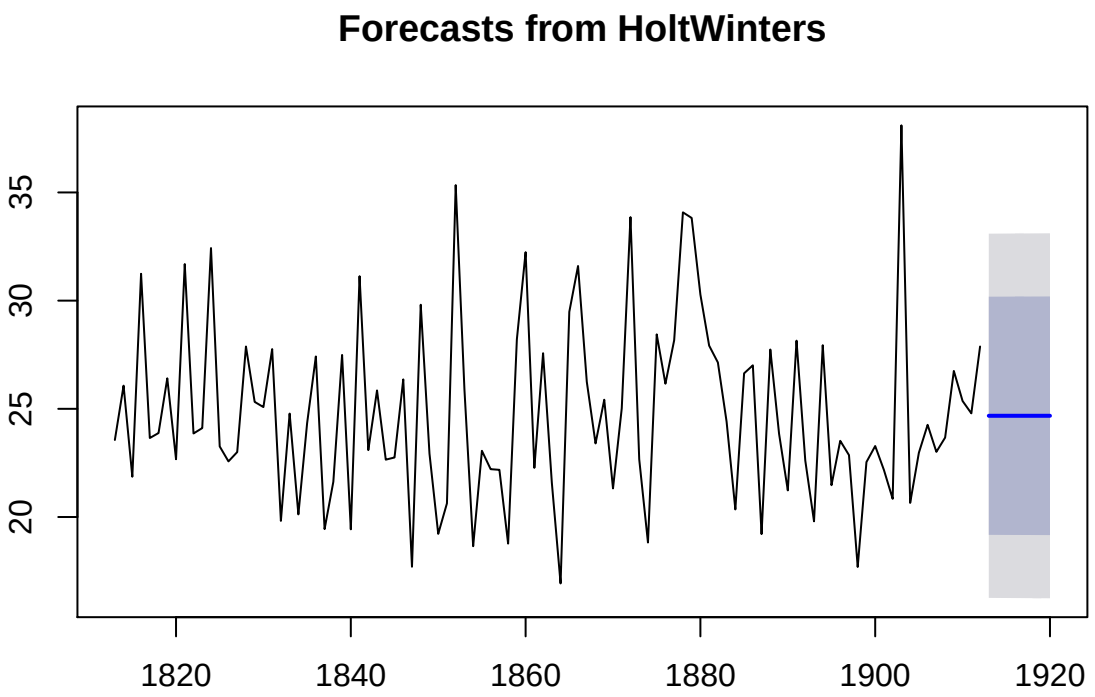
```
## Registered S3 methods overwritten by 'forecast':
##   method             from
##   fitted.fracdiff    fracdiff
##   residuals.fracdiff fracdiff
```

```
##
## Attaching package: 'forecast'
```

```
## The following object is masked from 'package:astsa':
##
##     gas
```

```
library(stats)
library(ggplot2)
library(reshape)
```

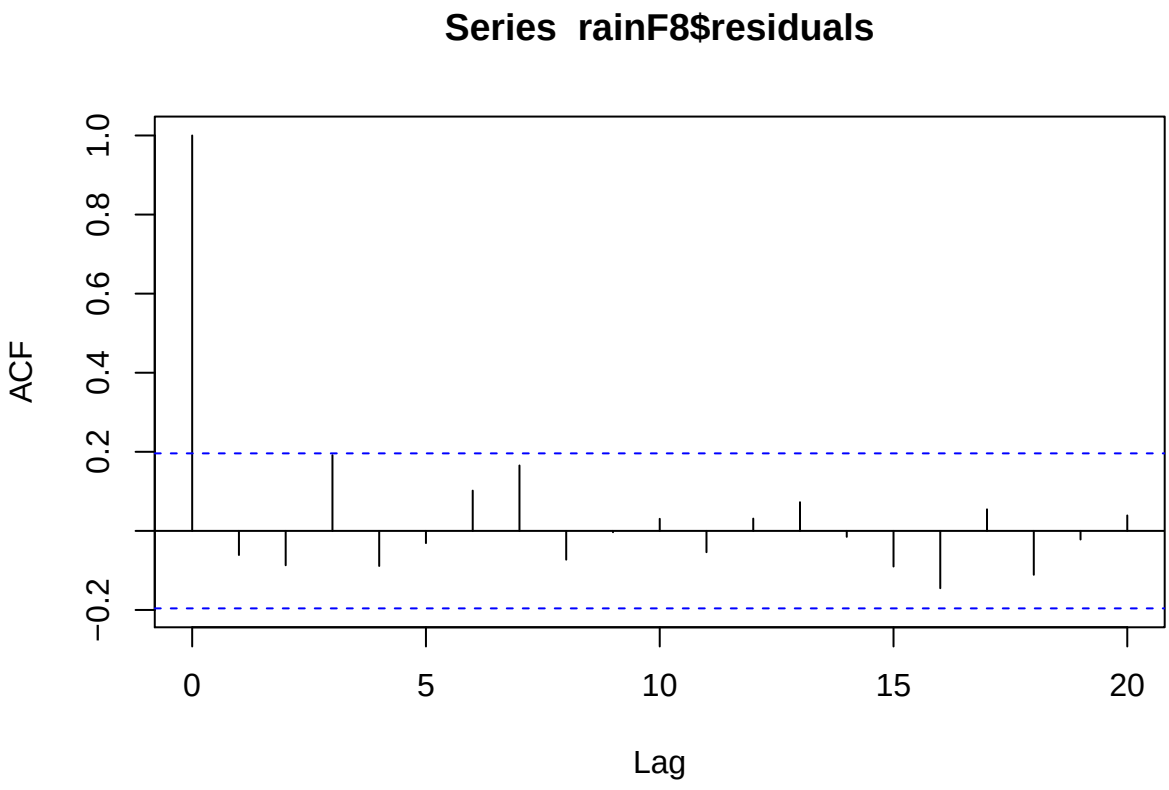
```
rainF8 <- forecast(rainF,8)
plot(rainF8)
```



where the dark gray is the 80% confidence interval, the light gray is the 95% confidence interval, and the blue line are the actual predictions.

To test the validity of this model we can again compute the SSE, however, we should also examine the correlations between the forecast errors. If correlation exists in the error terms, it is likely that the simple exponential smoothing forecasts could be improved upon by another technique.

```
acf(rainF8$residuals,na.action = na.pass)
```



We can

see that lag 3 is just about touching the significance interval. To test whether there is significant evidence for non-zero correlations we can carry out a Ljung-Box test. In R we can use the Box-test() function.

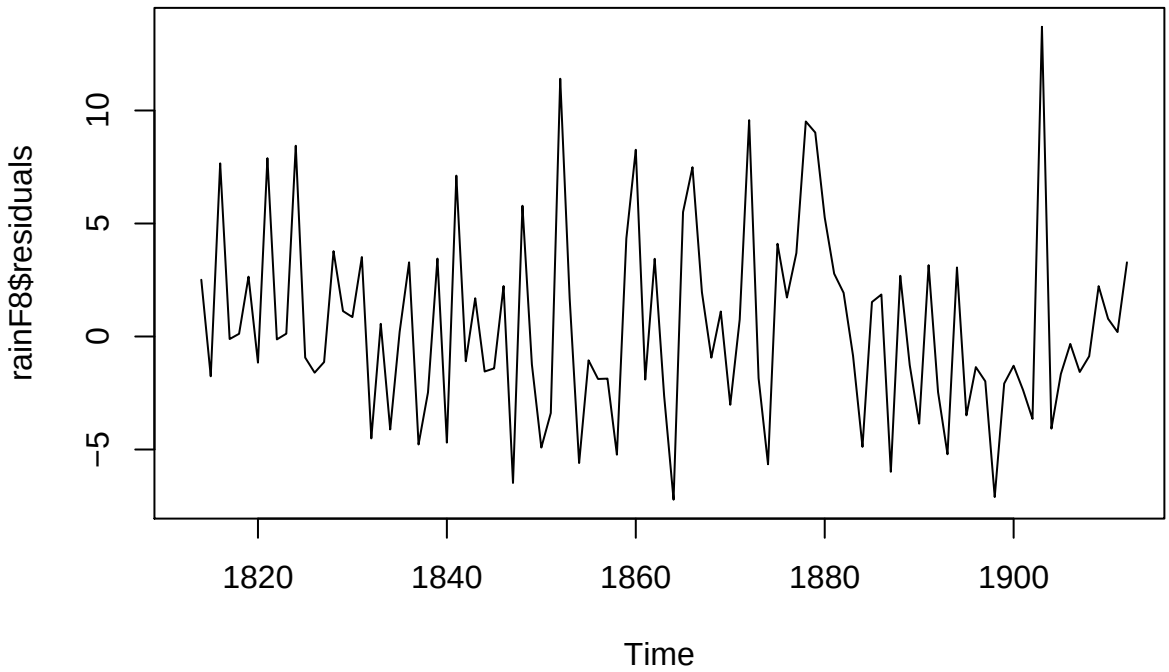
```
Box.test(rainF8$residuals, lag=20, type='Ljung-Box')
```

```
##
##  Box-Ljung test
##
## data:  rainF8$residuals
## X-squared = 17.401, df = 20, p-value = 0.6268
```

Here we see that the p-value is ~0.6, so there is little evidence of non-zero autocorrelation in the forecast errors.

To be sure the predictive model cannot be improved upon we can check whether the forecast errors are normally distributed with mean = 0 and constant variance.

```
plot.ts(rainF8$residuals)
```



While the mean is roughly constant, the variance does not seem to be obviously constant, but it is close enough.

Exponential Smoothing

If you have a time series that can be described using an additive model with a trend and no seasonality, you could use Holt's exponential smoothing to describe the series. The smoothing is controlled by parameters alpha: the estimate of the level at the current time point, and beta: the estimate of the slope b of the trend component at the current time point. These two parameters range form 0 - 1, and similar to simple smoothing, values close to 0 mean that little weight is placed on the most recent observations when making forecasts.

Here we utilize this model on a time series of annual diameter of woman's skirts at the hem, from 1866 to 1911.

```
skirts <- ts(scan("http://robjhyndman.com/tsdldata/roberts/skirts.dat",skip=5), start=c(1866))
plot.ts(skirts)
```

To make forecasts we will again use the HotWinters() function. For exponential smoothing, we only set the gamma = FALSE.

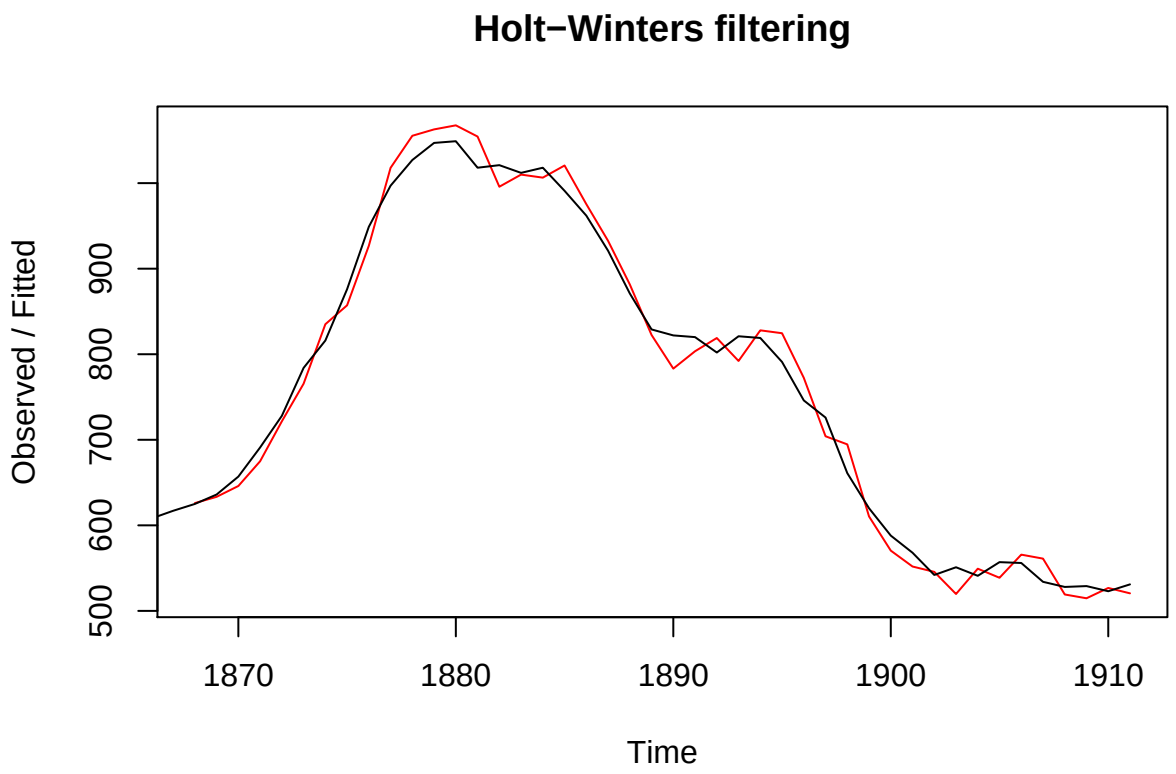
```
skirtsF <- HoltWinters(skirts, gamma=F)
skirtsF; skirtsF$SSE
```

```
## 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
```

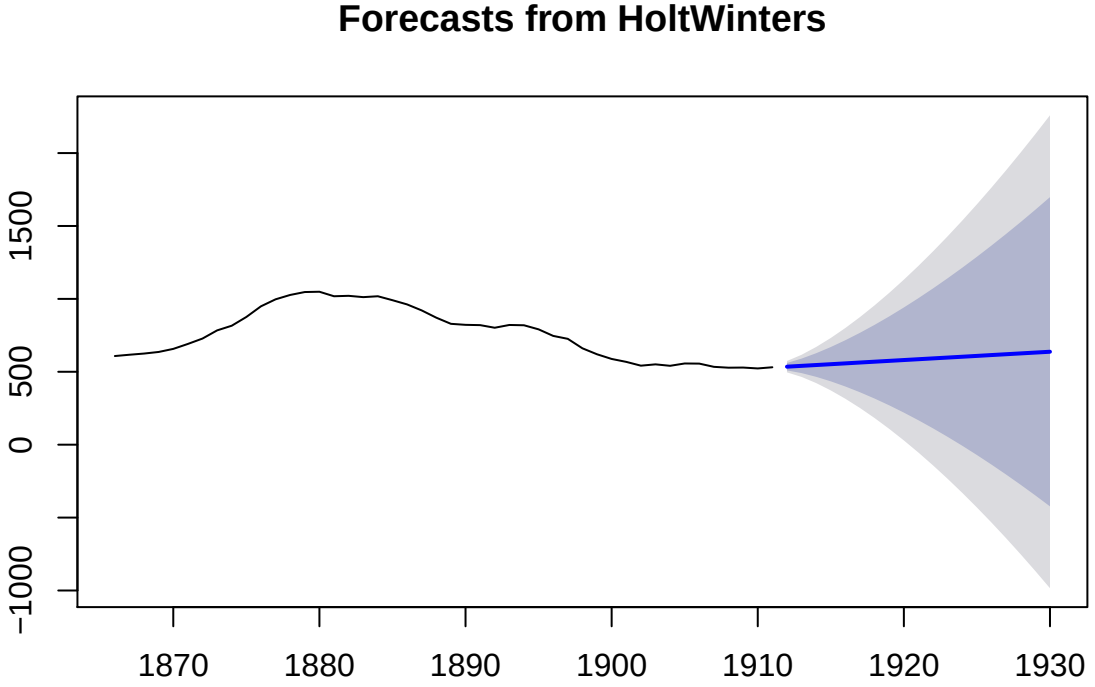
```
## [1] 16954.18
```

The high estimates of alpha and beta tell us that the estimate of the current level are basted mostly on recent observations. This makes sense since the data change rapidly. To examine the fit visually, we can plot the forecasts vs the original data.

```
plot(skirtsF)
```

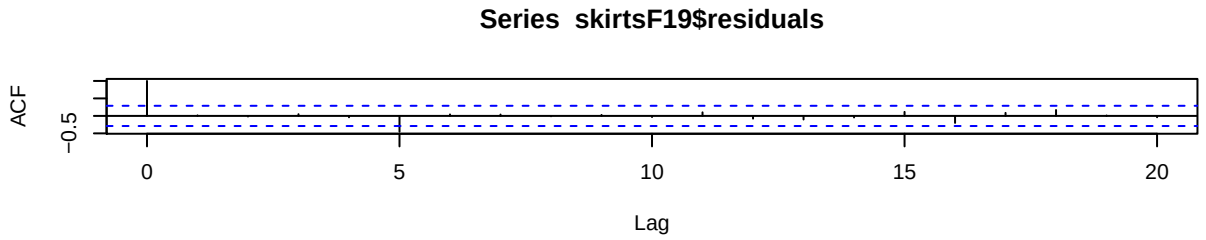



```
# Forecast into the future
skirtsF19 <- forecast(skirtsF,19)
plot(skirtsF19)
```



To validate the predictive model, we can check to verify the residuals are not correlated.

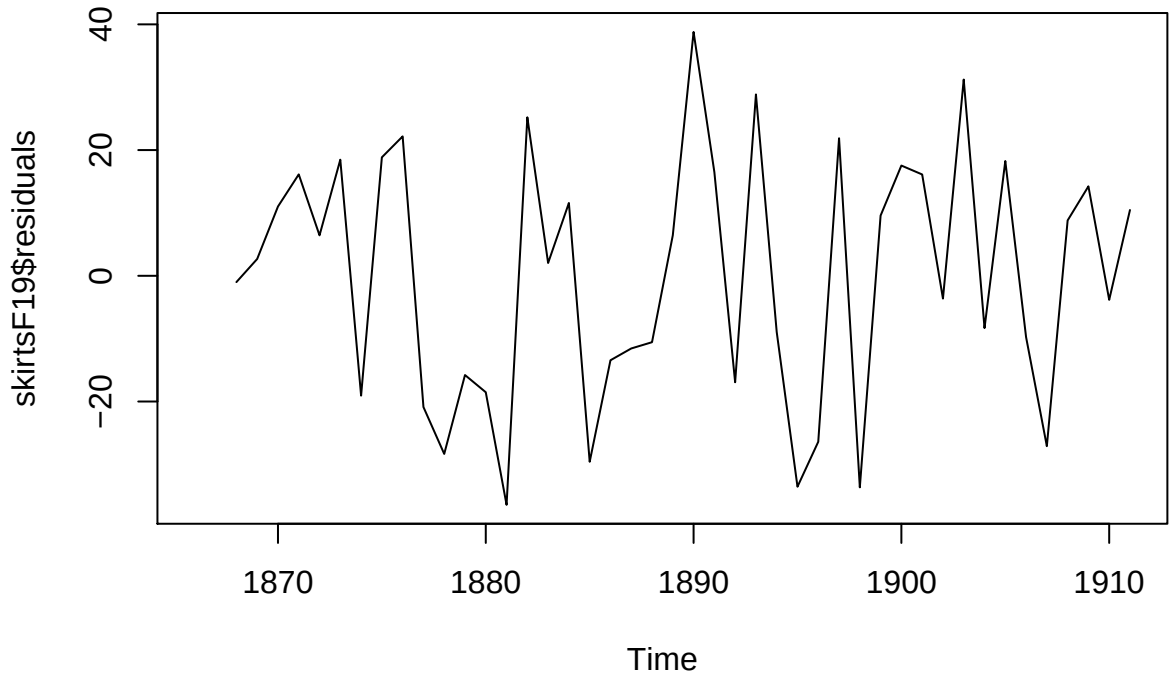
```
par(mfrow=c(3, 1))
acf(skirtsF19$residuals, lag.max=20,na.action = na.pass)
```




```
Box.test(skirtsF19$residuals, lag=20, type='Ljung-Box')
```

```
##
##  Box-Ljung test
##
## data:  skirtsF19$residuals
## X-squared = 19.731, df = 20, p-value = 0.4749
```

```
plot.ts(skirtsF19$residuals)
```

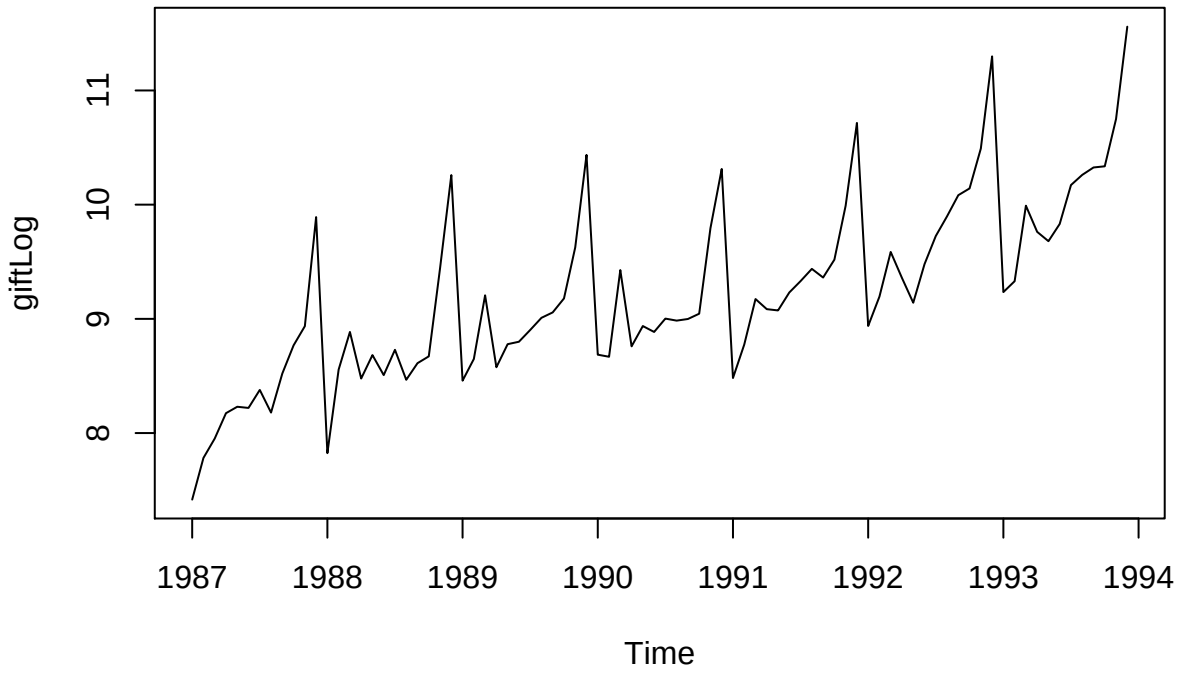


Here, we see that the ACF shows significant correlation at lag 5. However, the Ljung-Box test indicates little evidence of non-zero autocorrelations for 20 lags, which means the autocorrelation at lag 5 could be expected by chance. Further, the residuals are normally distributed, thus we can probably conclude this model as an adequate representation of the data.

Example 2

The beach-side gift shop dataset is an example of a time series with a trend component and seasonality.

```
giftLog <- log(gift) # take natural log
plot.ts(giftLog)
```




```
giftLogF <- forecast(giftLog )
giftLogF
```

```
##          Point Forecast     Lo 80     Hi 80     Lo 95     Hi 95
## Jan 1994       9.674641  9.464046  9.885235  9.352565  9.996717
## Feb 1994       9.958925  9.733371 10.184479  9.613970 10.303880
## Mar 1994      10.439733 10.200143 10.679323 10.073312 10.806155
## Apr 1994      10.121042  9.868185 10.373898  9.734331 10.507752
## May 1994      10.157291  9.891822 10.422759  9.751292 10.563289
## Jun 1994      10.227196  9.949682 10.504711  9.802774 10.651618
## Jul 1994      10.403289 10.114223 10.692356  9.961200 10.845378
## Aug 1994      10.382661 10.082480 10.682842  9.923574 10.841748
## Sep 1994      10.518849 10.207944 10.829754 10.043361 10.994337
## Oct 1994      10.612257 10.290980 10.933534 10.120906 11.103608
## Nov 1994      11.108401 10.777070 11.439732 10.601674 11.615129
## Dec 1994      11.898748 11.557653 12.239843 11.377088 12.420408
## Jan 1995       9.985303  9.634703 10.335903  9.449107 10.521499
## Feb 1995      10.269587  9.909735 10.629440  9.719240 10.819934
## Mar 1995      10.750396 10.381517 11.119274 10.186244 11.314547
## Apr 1995      10.431704 10.054009 10.809399  9.854070 11.009338
## May 1995      10.467953 10.081638 10.854268  9.877135 11.058770
## Jun 1995      10.537858 10.143107 10.932610  9.934137 11.141579
## Jul 1995      10.713952 10.310934 11.116969 10.097590 11.330314
## Aug 1995      10.693323 10.282201 11.104445 10.064567 11.322080
## Sep 1995      10.829511 10.410437 11.248586 10.188592 11.470430
## Oct 1995      10.922919 10.496035 11.349803 10.270057 11.575781
## Nov 1995      11.419063 10.984506 11.853621 10.754465 12.083661
## Dec 1995      12.209410 11.767308 12.651512 11.533273 12.885547
```

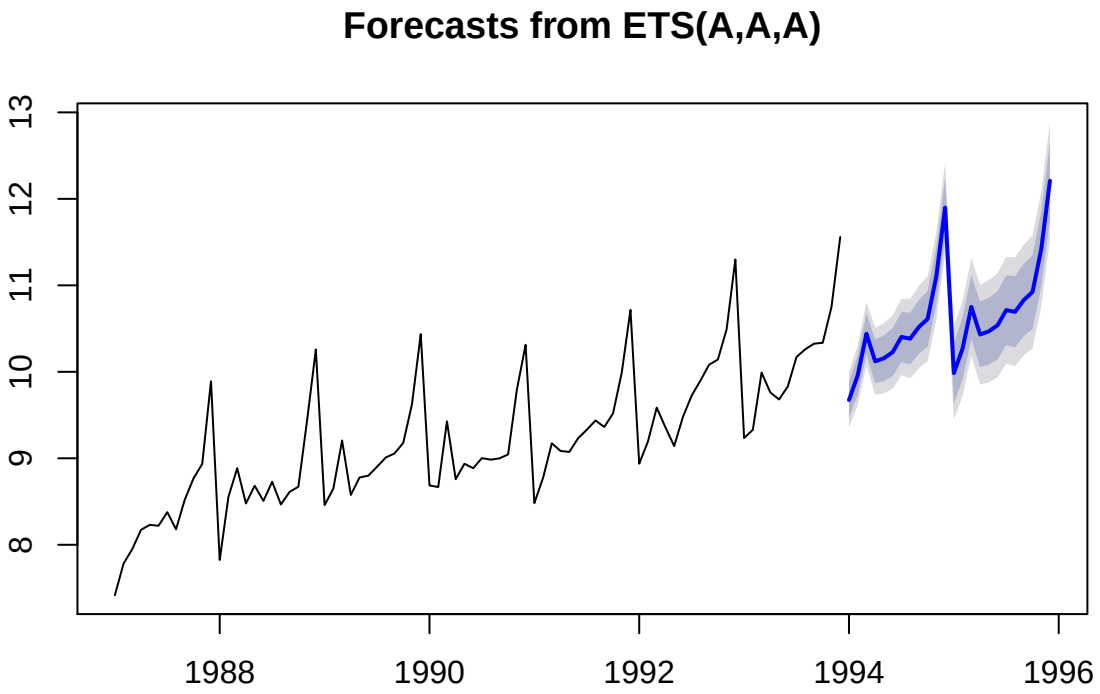
```
giftLogF$SSE
```

```
## NULL
```

From this output we see that the alpha = 0.41 which is closer to 0 than 1 and indicates that the estimate of

the level at the current time point is based upon both recent and past observations. The beta = 0.0 which means that the estimate of the slope of the trend component is not updated over the time series, and is instead equal to the initial value. This makes sense since we can see the trend component maintains near constant slope over the time series. The gamma = 0.95 is very high, which indicates that the estimate of the seasonal component at the current time point is based on very recent observations.

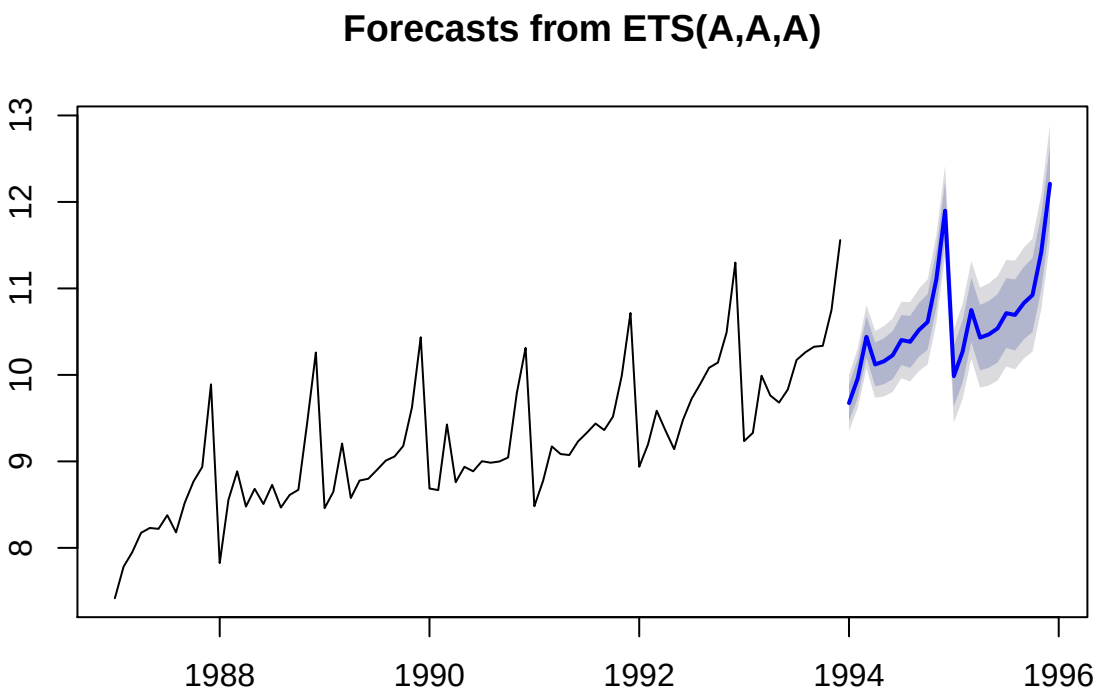
```
plot(giftLogF)
```



It is impressive how well we can predict the peaks in November each year.

Make forecasts into the future..

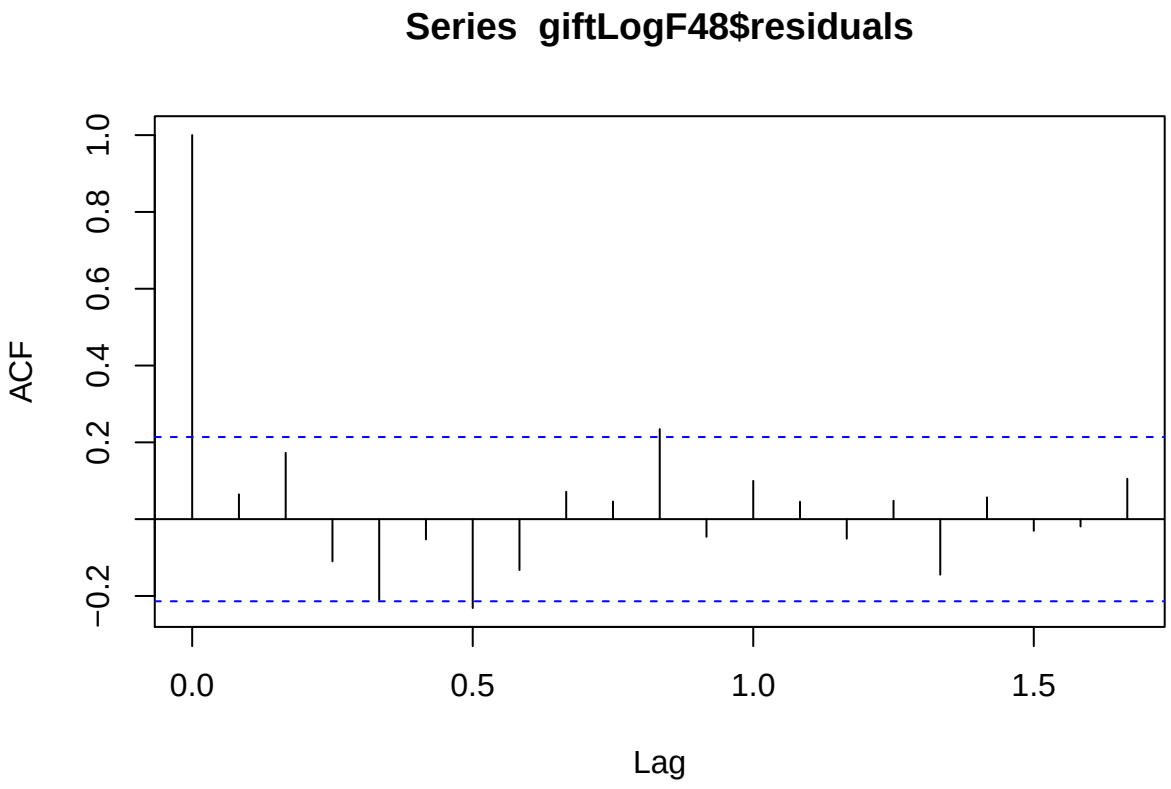
```
giftLogF48 <- forecast(giftLogF, 48) # predict 48 months ahead
plot(giftLogF48)
```



Test the model...

Again, we utilize various types of residual analysis to make sure our model cannot be approved upon.

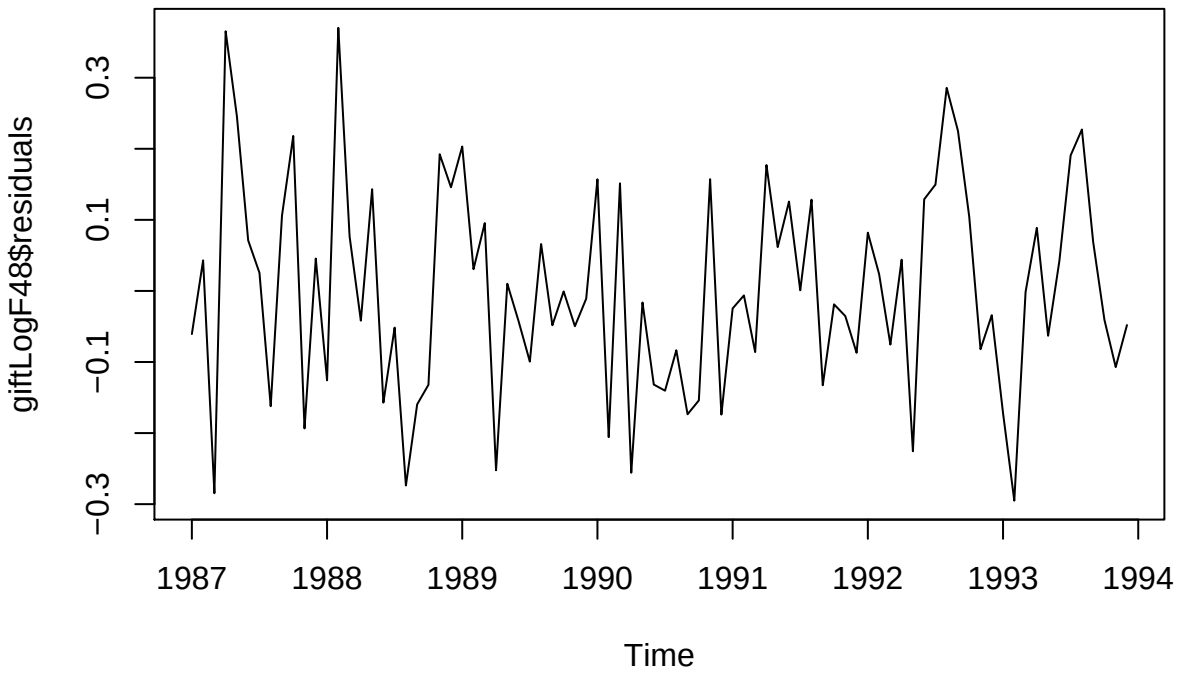
```
acf(giftLogF48$residuals, lag.max=20)
```



```
Box.test(giftLogF48$residuals, lag=20, type='Ljung-Box')
```

```
##
##  Box-Ljung test
##
## data:  giftLogF48$residuals
## X-squared = 26.832, df = 20, p-value = 0.14
```

```
plot.ts(giftLogF48$residuals)
```



Chapter 3 : ARIMA Models

While exponential smoothing methods are useful for forecasting, they make no assumptions about the correlations between successive values of the time series. We can sometimes make better models by utilizing these correlations in the data, using Autoregressive Integrated Moving Average (ARIMA) models.

##Differencing a Time Series ARIMA models are defined only for stationary time series. Therefore, if the raw data are non-stationary, you will need to difference the series until you obtain stationarity. Once you know the order of differencing you can use it as the d parameter in an ARIMA(p, d, q) model.

In R you can difference a time series using the diff() function. For example: the time series of annual skirt diameter is not stationary as the mean level changes a lot over time.

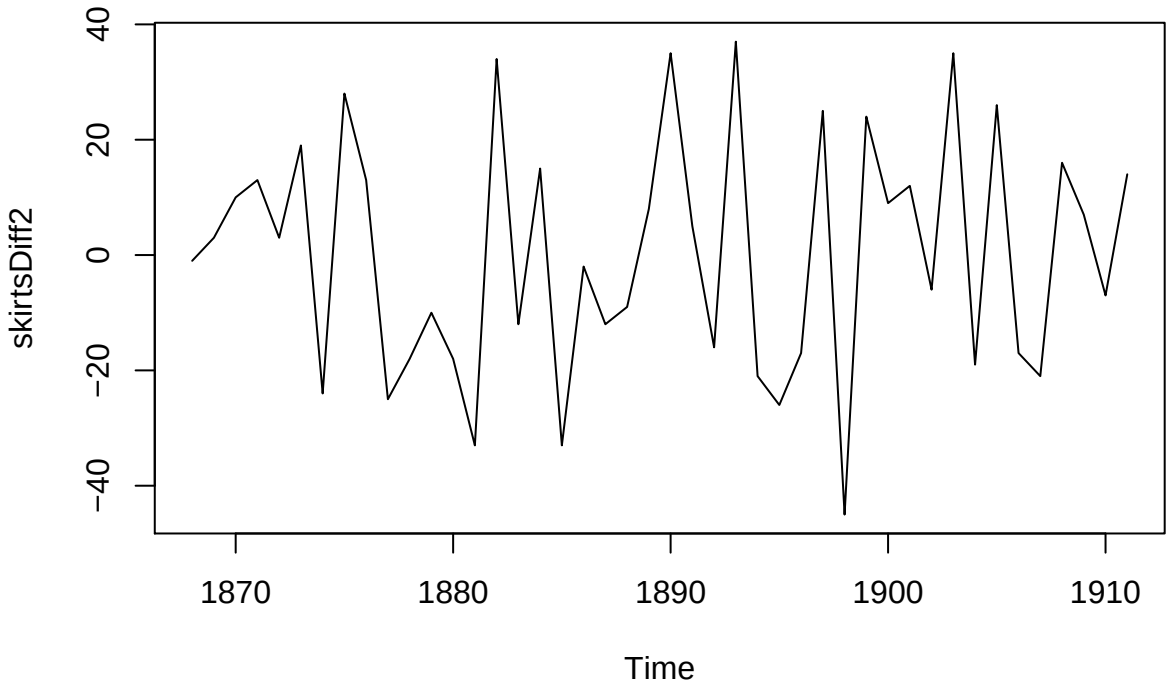
```
plot.ts(skirts)
```

We can take the first difference and plot the differenced series...

```
skirtsDiff <- diff(skirts, differences = 1)
plot.ts(skirtsDiff)
```

Again, the mean is not quite stationary. Therefore we can take the first difference of the first difference we computed above and see how that looks.

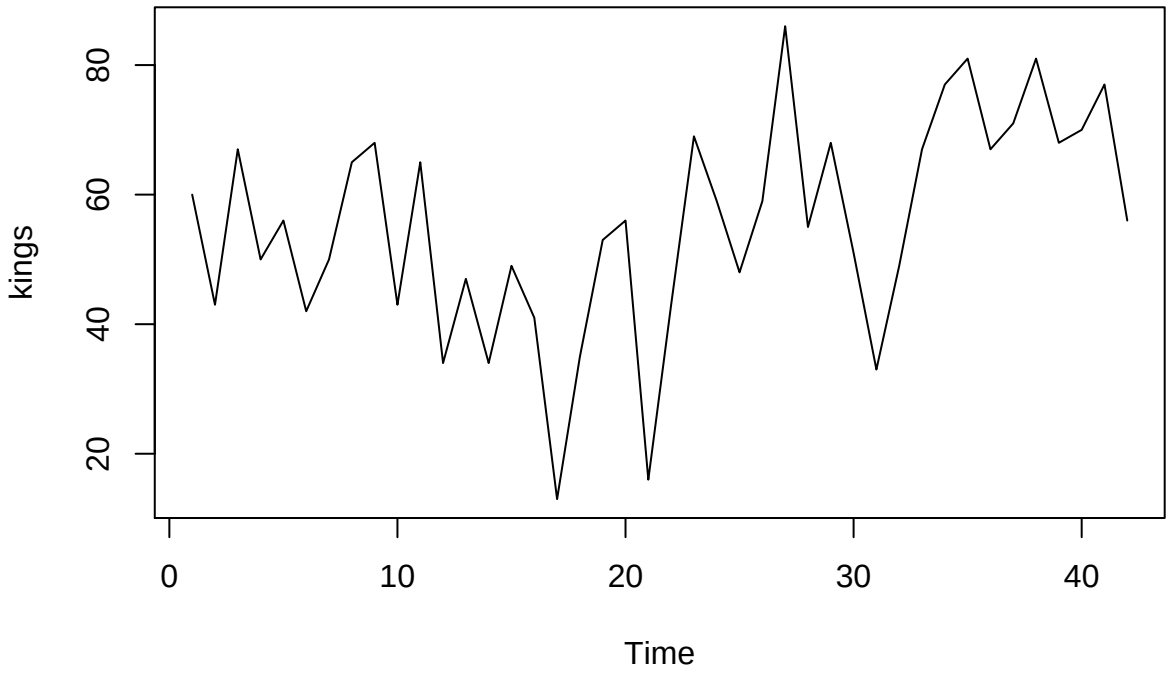
```
skirtsDiff2 <- diff(skirts, differences = 2)
plot.ts(skirtsDiff2)
```



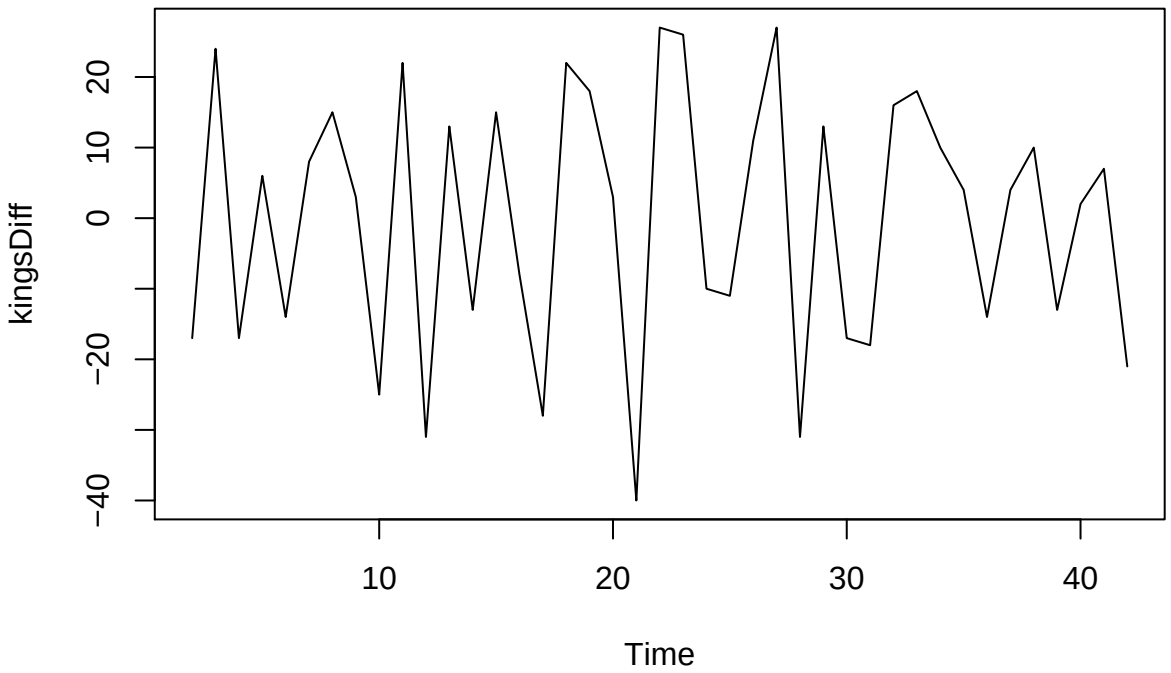
There we go, it appears that the mean and variance remain constant and stable over time. So far, our model definition is ARIMA(p, 2, q). The next step is to figure out the p and q values of this model. ### Another Example of Differencing

Kings death dataset...

```r
plot.ts(kings)
```




```r
# First differencing
kingsDiff <- diff(kings, differences = 1)
plot.ts(kingsDiff)
```



In this case, we can see that an ARIMA(p, 1, q) model would be appropriate.

Next we can examine the correlation between successive terms of this irregular differenced component.

Chapter 4 : Selecting a Cadidate ARIMA Model

Once we have a stationary time series, the next still is to select the best ARIMA model. To do this we will inspect the autocorrelation function (ACF) and partial autocorrelation function (PACF). You can find a good collection of some of the 'rules' for selecting ARIMA models from Duke University here.

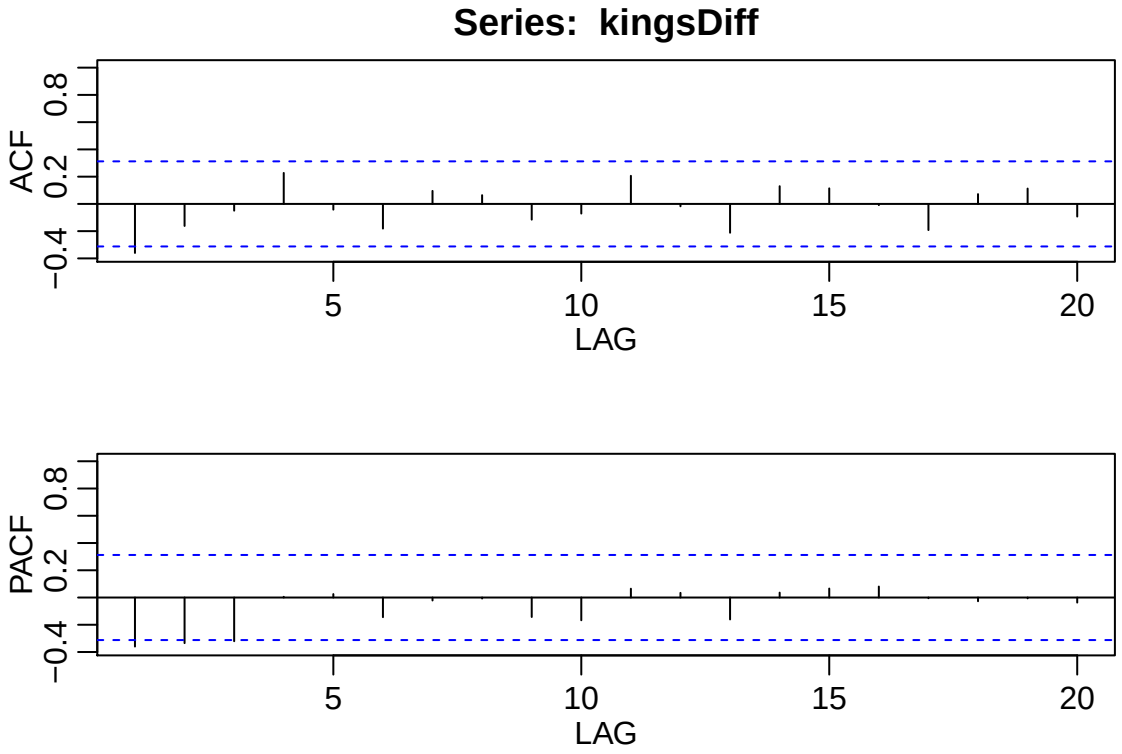
Example of Ages of Kings Death

I find it easiest to use some of the functions provided.

```
# Install the tools
source(url("http://lib.stat.cmu.edu/general/tsa2/Rcode/itall.R"))
```

```
##   itall has been installed
```

```
# Plot ACF and PACF
acf2(kingsDiff, max.lag = 20)
```




```
##          ACF  PACF
##  [1,] -0.36 -0.36
##  [2,] -0.16 -0.33
##  [3,] -0.05 -0.32
##  [4,]  0.23  0.01
##  [5,] -0.04  0.03
##  [6,] -0.18 -0.14
##  [7,]  0.10 -0.02
##  [8,]  0.06 -0.01
##  [9,] -0.12 -0.14
## [10,] -0.07 -0.17
## [11,]  0.21  0.06
## [12,] -0.02  0.03
```

```
## [13,] -0.21 -0.16
## [14,]  0.13  0.04
## [15,]  0.11  0.07
## [16,] -0.01  0.08
## [17,] -0.19 -0.01
## [18,]  0.07 -0.03
## [19,]  0.11 -0.01
## [20,] -0.09 -0.04
```

We can see that the ACF is significant at lag 1 and has an alternating pattern. The PACF depicts significant autocorrelation at the first three lags.

The two most basic rules are:

If the ACF is a sharp cutoff the q component is equal to the last significant lag. This type of model also often has a PACF with a tapering pattern. If the PACF has a sharp cutoff, the p component is equal to the last number of significant lags. Again, the ACF may exhibit a tapering pattern. Understanding these observations, we would probably test these 2 ARIMA models first.

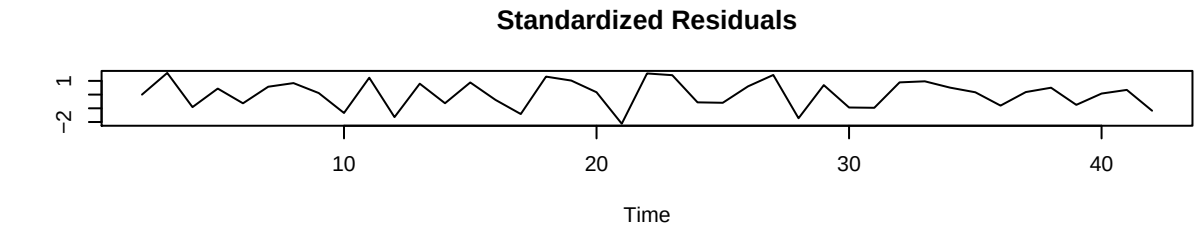
ARIMA (3, 1, 0), since the PACF cuts off sharply after lag 3, and the ACF exhibits a tapering pattern. AR(3), MA(0)

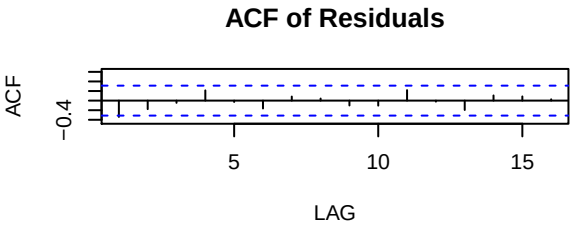
ARIMA(0, 1, 1), since the ACF is zero after lag 1 and the PACF does also taper off to some degree. AR(0), MA(1)

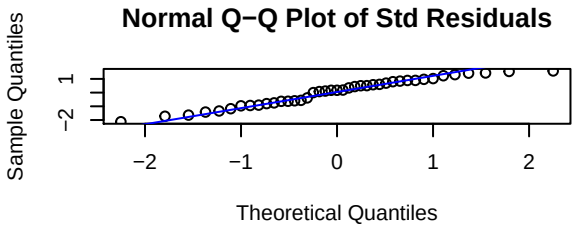
Under the case that we have a few potential models, it is convention to pick the simplest model. For now, we will test the diagnostics of the both models, and examine the residual plots, and AIC to determine the best model.

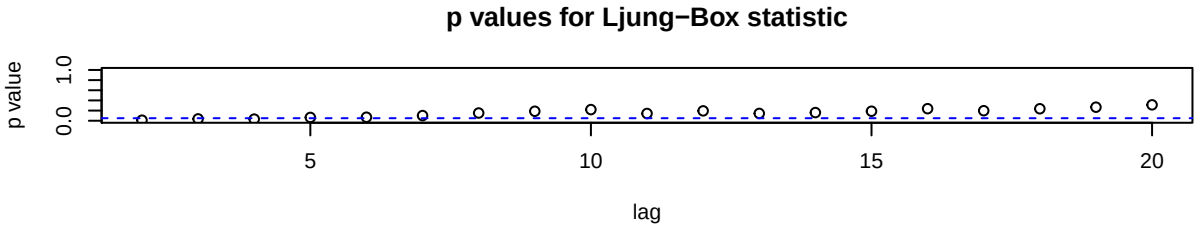
```
sarima(kingsDiff, 0, 1, 1)
```

```
## initial  value 3.400079
## iter   2 value 3.130716
## iter   3 value 3.082518
## iter   4 value 3.031836
## iter   5 value 3.014638
## iter   6 value 2.996626
## iter   7 value 2.996427
## iter   8 value 2.993689
## iter   9 value 2.992911
## iter  10 value 2.992715
## iter  11 value 2.992675
## iter  12 value 2.992675
## iter  13 value 2.992672
## iter  14 value 2.992672
## iter  14 value 2.992672
## iter  14 value 2.992672
## final  value 2.992672
## converged
## initial  value 2.981319
## iter   2 value 2.967670
## iter   3 value 2.959423
## iter   4 value 2.956145
## iter   5 value 2.956067
## iter   6 value 2.956066
## iter   6 value 2.956066
## final  value 2.956066
## converged
```










```
## $fit
##
## Call:
## arima(x = xdata, order = c(p, d, q), seasonal = list(order = c(P, D, Q), period = S),
##     xreg = constant, optim.control = list(trace = trc, REPORT = 1, reltol = tol))
##
## Coefficients:
##           ma1  constant
##       -0.9998    0.0201
## s.e.   0.0637    0.2423
##
## sigma^2 estimated as 336.8:  log likelihood = -175,  aic = 356
##
## $AIC
## [1] 6.917013
##
## $AICc
## [1] 6.981614
##
## $BIC
## [1] 6.000602
```

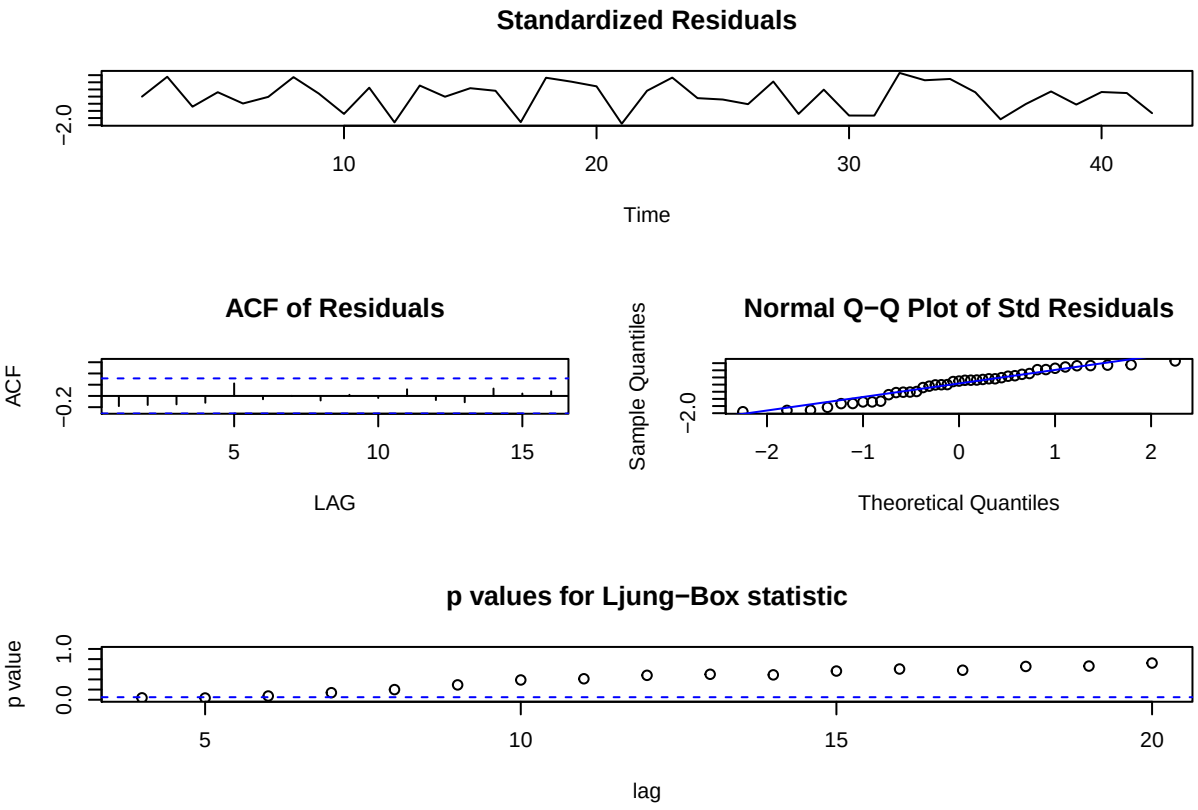
```
Box.test(kingsDiff, lag=20, type='Ljung-Box')
```

```
##
##  Box-Ljung test
##
## data:  kingsDiff
## X-squared = 25.103, df = 20, p-value = 0.1975
```

While there is an auto correlated residual at lag 1, this model is valid.

```
sarima(kingsDiff, 3, 1, 0)
```

```
## initial  value 3.381661
## iter   2 value 3.206938
## iter   3 value 3.160710
## iter   4 value 3.067232
## iter   5 value 3.049600
## iter   6 value 3.007382
## iter   7 value 2.951981
## iter   8 value 2.933148
## iter   9 value 2.931406
## iter  10 value 2.931149
## iter  11 value 2.931132
## iter  12 value 2.931075
## iter  13 value 2.931003
## iter  14 value 2.930991
## iter  15 value 2.930974
## iter  15 value 2.930974
## iter  15 value 2.930974
## final  value 2.930974
## converged
## initial  value 2.950994
## iter   2 value 2.950920
## iter   3 value 2.950057
## iter   4 value 2.948720
## iter   5 value 2.948683
## iter   6 value 2.948682
## iter   6 value 2.948682
## iter   6 value 2.948682
## final  value 2.948682
## converged
```




```
## $fit
##
## Call:
## arima(x = xdata, order = c(p, d, q), seasonal = list(order = c(P, D, Q), period = S),
##     xreg = constant, optim.control = list(trace = trc, REPORT = 1, reltol = tol))
##
## Coefficients:
##           ar1      ar2      ar3  constant
##       -1.0447  -0.8536  -0.5253   -0.1210
## s.e.   0.1332   0.1591   0.1312    0.8908
##
## sigma^2 estimated as 348.3:  log likelihood = -174.7,  aic = 359.41
##
## $AIC
## [1] 7.04828
##
## $AICc
## [1] 7.138873
##
## $BIC
## [1] 6.215458
```

```
Box.test(kingsDiff, lag=20, type='Ljung-Box')
```

```
##
##  Box-Ljung test
##
## data:  kingsDiff
## X-squared = 25.103, df = 20, p-value = 0.1975
```

The Diagnostics are quite similar here. To choose the best model we can utilize the AIC and see that the first model (also the simplest) performs marginally better.

We can also use the auto.arima() function to test the best model with an automated approach and see how it compares to our manual model selection.

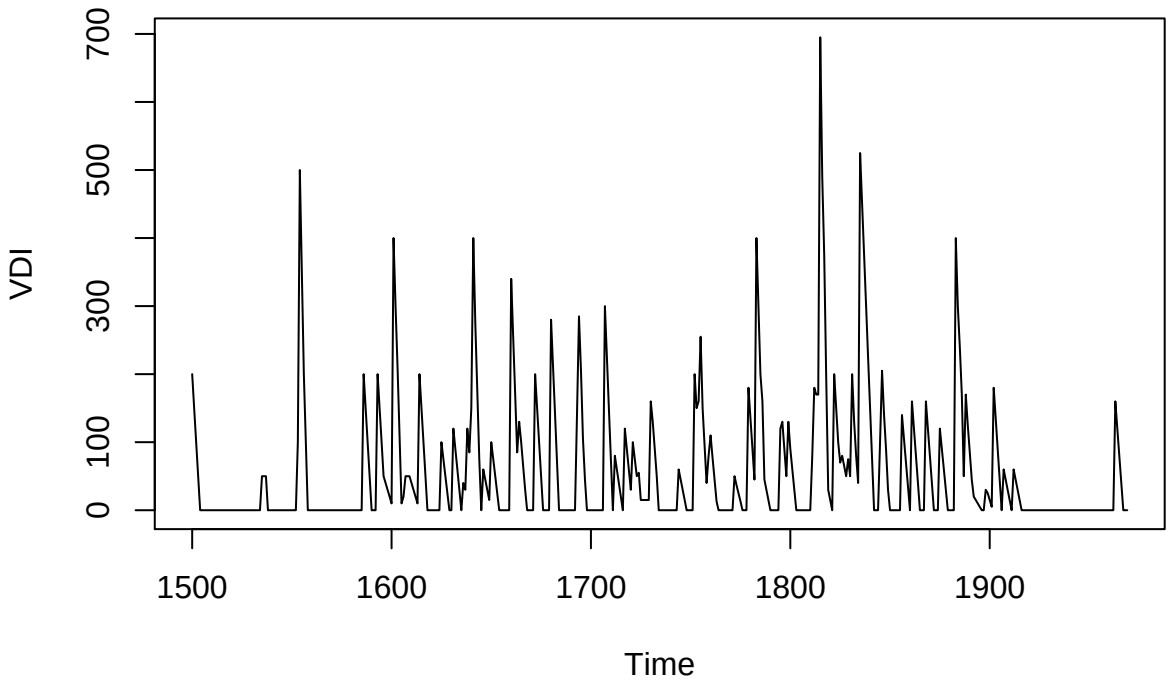
```
auto.arima(kings)
```

```
## Series: kings
## ARIMA(0,1,1)
##
## Coefficients:
##           ma1
##       -0.7218
## s.e.   0.1208
##
## sigma^2 estimated as 236.2:  log likelihood=-170.06
## AIC=344.13   AICc=344.44   BIC=347.56
```

Example of Volcanic Dust in N Hemisphere

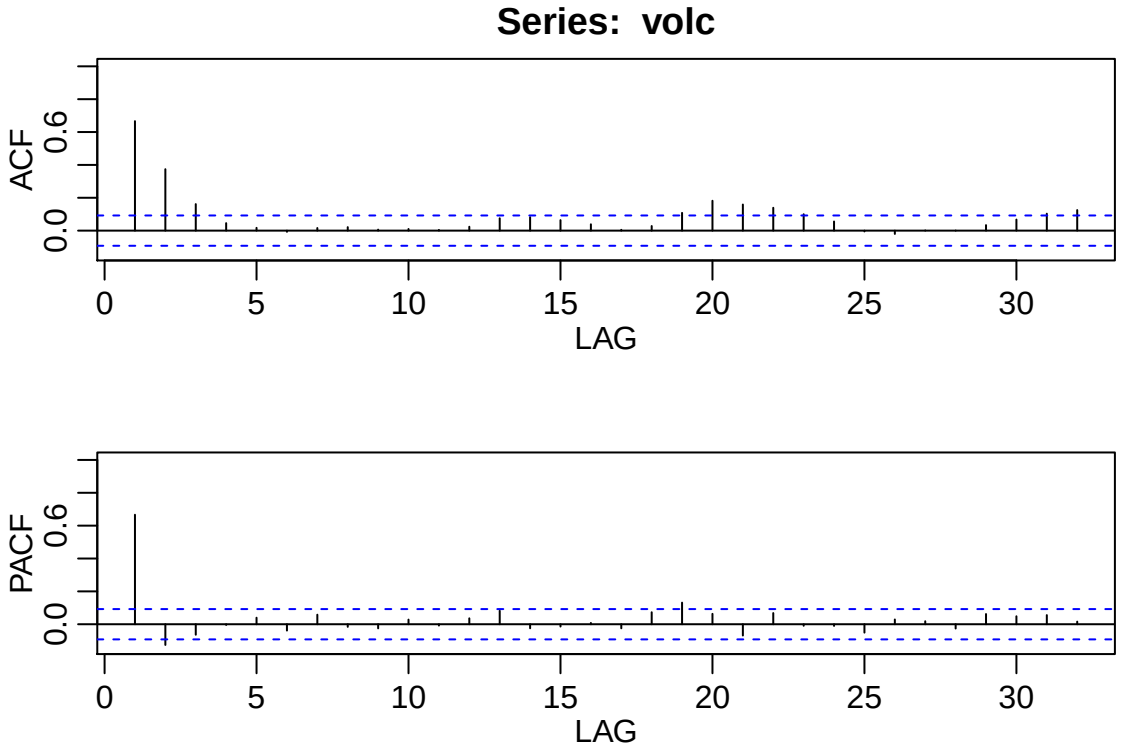
This dataset contains the _volcanic dust veil index in the northern hemisphere from 1500 - 1969. It is a measure of the impact of volcanic eruptions release of dist an aerosols into the environment.

```
volc <- ts(scan("http://robjhyndman.com/tsdldata/annual/dvi.dat", skip=1), start=c(1500))
plot.ts(volc, ylab = 'VDI')
```



We can see that the this time series may be stationary, since the mean is constant and the variance appears relatively constant over time. Therefore, we do not need to difference this series. Let's investigate the ACF and PACF.

```
acf2(volc)
```




```
##          ACF  PACF
##  [1,]  0.67  0.67
##  [2,]  0.37 -0.13
##  [3,]  0.16 -0.06
##  [4,]  0.05  0.00
##  [5,]  0.02  0.04
##  [6,] -0.01 -0.04
##  [7,]  0.02  0.06
##  [8,]  0.02 -0.02
##  [9,]  0.01 -0.03
## [10,]  0.01  0.03
## [11,]  0.00 -0.01
## [12,]  0.02  0.04
## [13,]  0.08  0.08
## [14,]  0.08 -0.02
## [15,]  0.06 -0.01
## [16,]  0.04  0.01
## [17,]  0.00 -0.02
## [18,]  0.03  0.07
## [19,]  0.11  0.13
## [20,]  0.18  0.06
## [21,]  0.16 -0.07
## [22,]  0.14  0.07
## [23,]  0.10 -0.01
## [24,]  0.06 -0.01
## [25,] -0.01 -0.05
## [26,] -0.02  0.03
## [27,]  0.00  0.02
## [28,]  0.00 -0.03
## [29,]  0.03  0.06
```

```
## [30,]  0.07  0.05
## [31,]  0.10  0.06
## [32,]  0.13  0.02
```

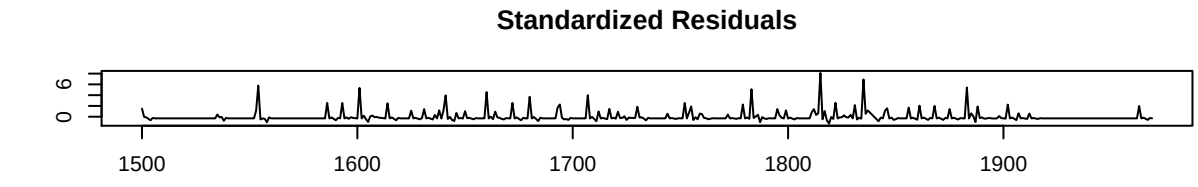
The ACF depicts a tapering pattern where the first 3 lags are auto correlated. The lags of 19 - 23 are also significant, but we expect that these are by chance, since the autocorrelations for lags 4 - 18 are not significant.

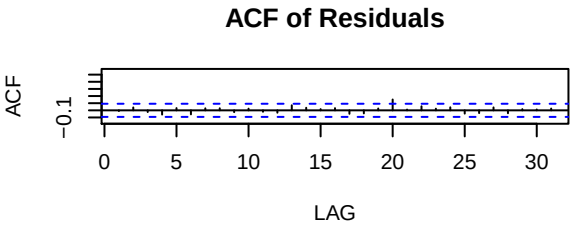
The PACF depicts a sharp cutoff where the first two lags are significant.

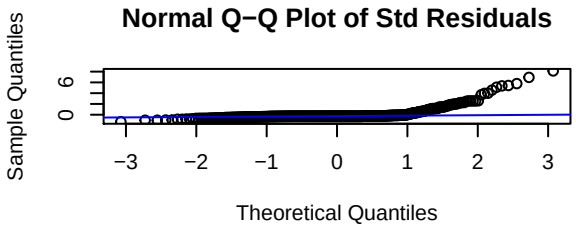
Given this information, we have two possible models...

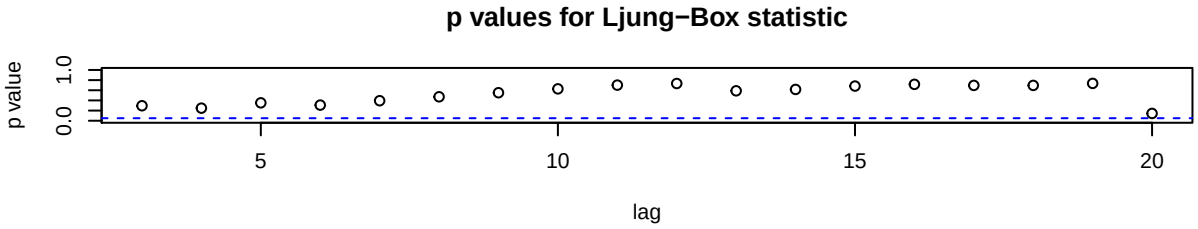
```
sarima(volc, 2,0,0)
```

```
## initial  value 4.547990
## iter   2 value 4.404559
## iter   3 value 4.272216
## iter   4 value 4.247505
## iter   5 value 4.245129
## iter   6 value 4.245081
## iter   7 value 4.245079
## iter   8 value 4.245078
## iter   9 value 4.245078
## iter  10 value 4.245076
## iter  11 value 4.245074
## iter  12 value 4.245074
## iter  12 value 4.245074
## iter  12 value 4.245074
## final  value 4.245074
## converged
## initial  value 4.246070
## iter   2 value 4.246066
## iter   3 value 4.246062
## iter   4 value 4.246058
## iter   5 value 4.246051
## iter   6 value 4.246049
## iter   7 value 4.246048
## iter   7 value 4.246048
## iter   7 value 4.246048
## final  value 4.246048
## converged
```







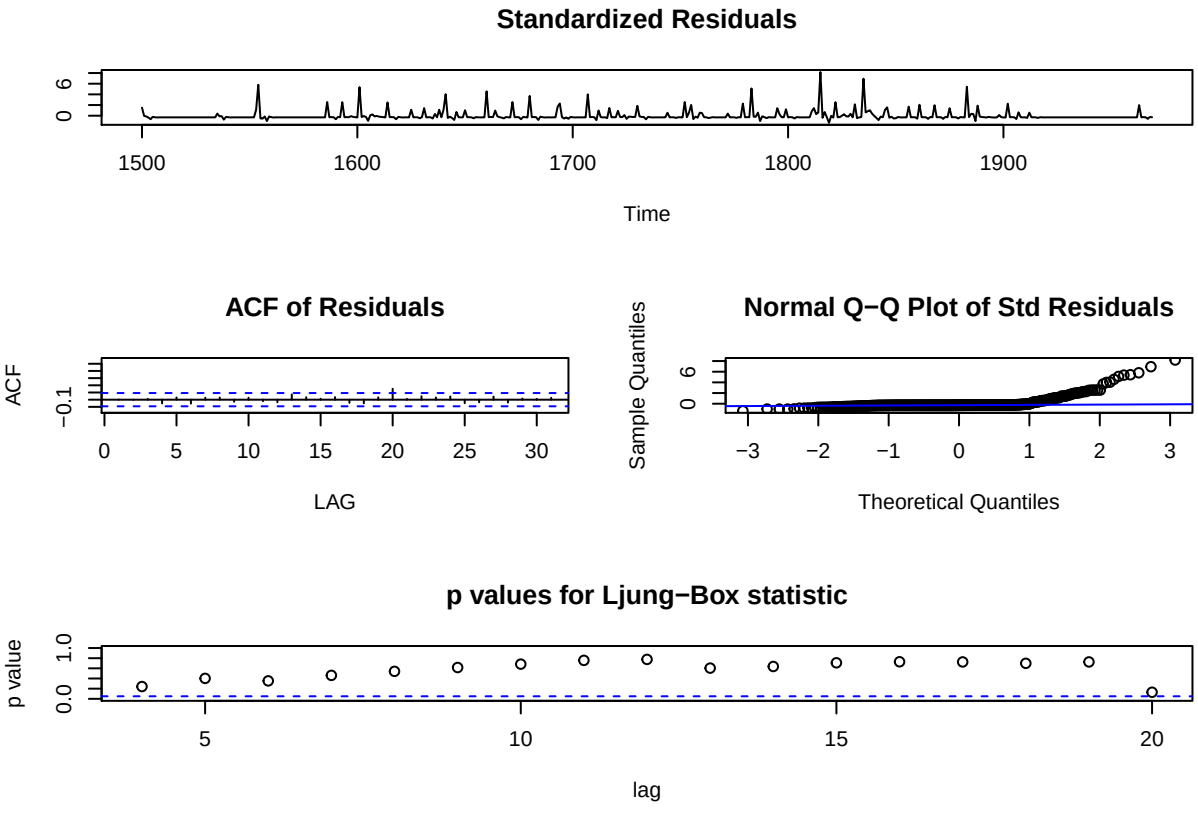



```
## $fit
##
## Call:
## arima(x = xdata, order = c(p, d, q), seasonal = list(order = c(P, D, Q), period = S),
##     xreg = xmean, include.mean = FALSE, optim.control = list(trace = trc, REPORT = 1,
##         reltol = tol))
##
## Coefficients:
##          ar1      ar2    xmean
##       0.7533  -0.1268  57.5274
## s.e.  0.0457   0.0458   8.5958
##
## sigma^2 estimated as 4870:  log likelihood = -2662.54,  aic = 5333.09
##
## $AIC
## [1] 9.503533
##
## $AICc
## [1] 9.507972
##
## $BIC
## [1] 8.53004
```

```
sarima(volc, 1,0,2)
```

```
## initial  value 4.547952
## iter   2 value 4.410381
## iter   3 value 4.251298
## iter   4 value 4.246934
```

```
## iter   5 value 4.244047
## iter   6 value 4.243760
## iter   7 value 4.243019
## iter   8 value 4.242747
## iter   9 value 4.242674
## iter  10 value 4.242670
## iter  11 value 4.242670
## iter  12 value 4.242670
## iter  13 value 4.242670
## iter  14 value 4.242669
## iter  14 value 4.242669
## iter  14 value 4.242669
## final  value 4.242669
## converged
## initial  value 4.244570
## iter   2 value 4.244568
## iter   3 value 4.244565
## iter   4 value 4.244554
## iter   5 value 4.244553
## iter   6 value 4.244553
## iter   7 value 4.244553
## iter   8 value 4.244553
## iter   8 value 4.244553
## iter   8 value 4.244553
## final  value 4.244553
## converged
```




```
## $fit
##
```

```
## Call:
## arima(x = xdata, order = c(p, d, q), seasonal = list(order = c(P, D, Q), period = S),
##     xreg = xmean, include.mean = FALSE, optim.control = list(trace = trc, REPORT = 1,
##         reltol = tol))
##
## Coefficients:
##          ar1     ma1     ma2    xmean
##       0.4723  0.2694  0.1279  57.5178
## s.e.  0.0936  0.0969  0.0752   8.4883
##
## sigma^2 estimated as 4855:  log likelihood = -2661.84,  aic = 5333.68
##
## $AIC
## [1] 9.504779
##
## $AICc
## [1] 9.509309
##
## $BIC
## [1] 8.540121
```

Even though the optimal model is ARIMA(1,0,2), we prefer a simpler model and choose ARIMA(2,0,0). If we change our selection criterion to BIC, which penalizes for extra parameters, instead of AIC, we find that the simpler model indeed is the best description of the time series.

Forecasting Using an ARIMA Model

The Kings Death Dataset

Recall that this dataset can be described using a ARIMA(0,1,1) model.

```
kingsARIMA <- arima(kings, order=c(0,1,1))
kingsARIMA
```

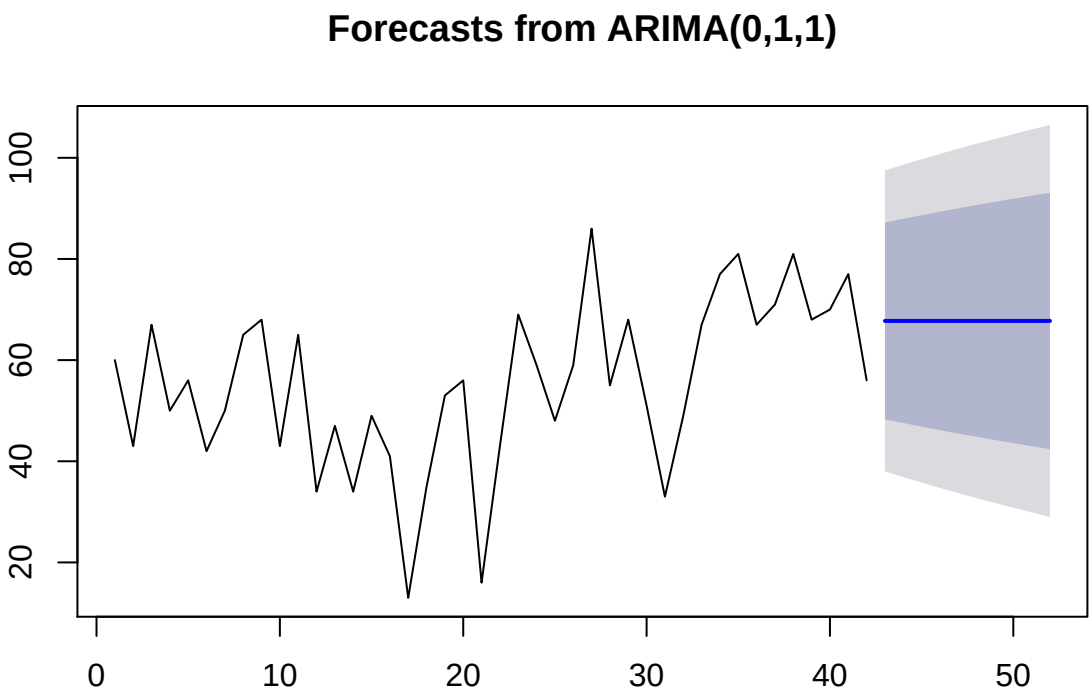
```
##
## Call:
## arima(x = kings, order = c(0, 1, 1))
##
## Coefficients:
##           ma1
##       -0.7218
## s.e.   0.1208
##
## sigma^2 estimated as 230.4:  log likelihood = -170.06,  aic = 344.13
```

```
library('ggplot2')
library('forecast')
library('tseries')
kingsF <- forecast::forecast(kingsARIMA) # modified kingsF <- forecast.Arima(kingsARIMA)
kingsF
```

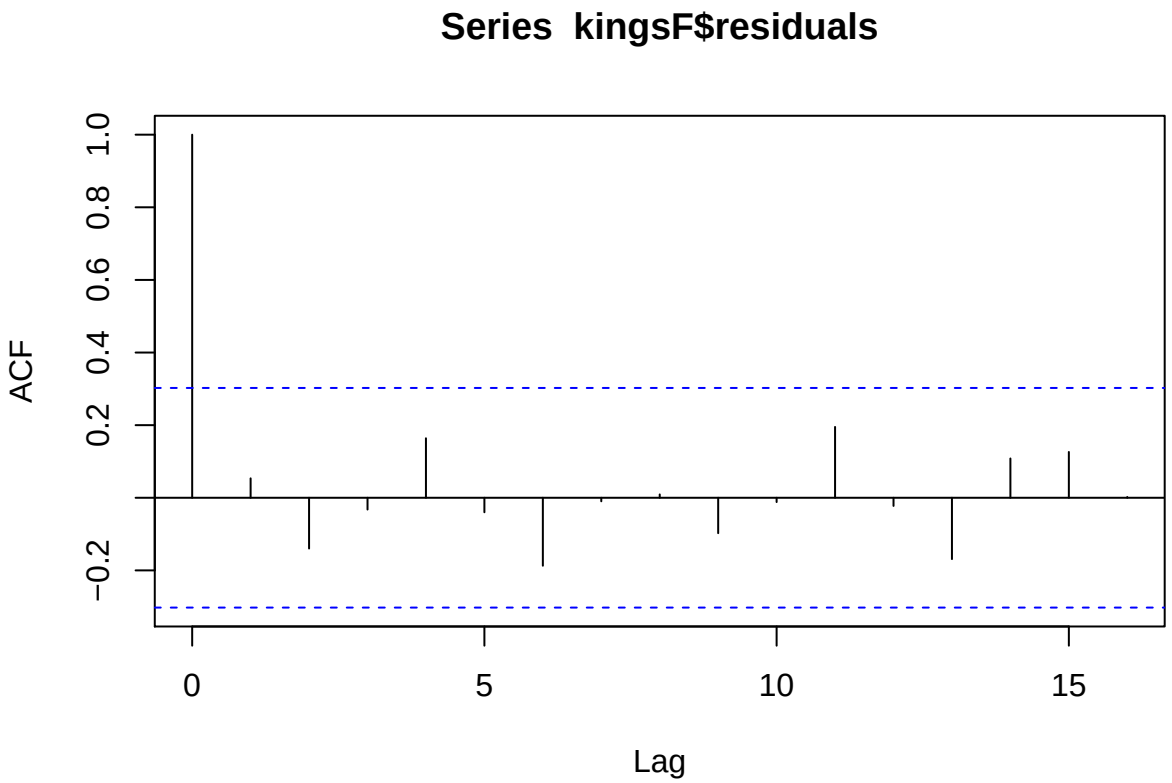
```
##    Point Forecast    Lo 80    Hi 80    Lo 95     Hi 95
## 43       67.75063 48.29647 87.20479 37.99806  97.50319
## 44       67.75063 47.55748 87.94377 36.86788  98.63338
```

```
## 45       67.75063 46.84460 88.65665 35.77762  99.72363
## 46       67.75063 46.15524 89.34601 34.72333 100.77792
## 47       67.75063 45.48722 90.01404 33.70168 101.79958
## 48       67.75063 44.83866 90.66260 32.70979 102.79146
## 49       67.75063 44.20796 91.29330 31.74523 103.75603
## 50       67.75063 43.59372 91.90753 30.80583 104.69543
## 51       67.75063 42.99472 92.50653 29.88974 105.61152
## 52       67.75063 42.40988 93.09138 28.99529 106.50596
```

```
plot(kingsF)
```




```
# Examine the residuals
acf(kingsF$residuals)
```




```
Box.test(kingsF$residuals, lag = 20, type='Ljung-Box')
```

```
##
##  Box-Ljung test
##
## data:  kingsF$residuals
## X-squared = 13.584, df = 20, p-value = 0.8509
```