Time Series

Packages

Uses my package `mkac` which is on Github. Install with:

```
library(devtools)
install_github("nxskok/mkac")
```

Plus these. You might need to install some of them first:

```
library(ggfortify)
library(forecast)
library(tidyverse)
library(mkac)
```

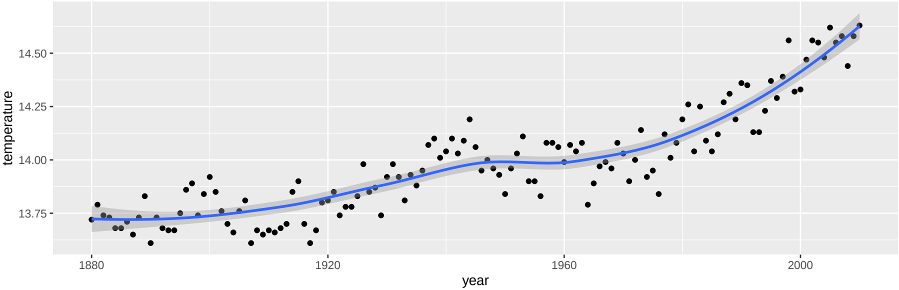
Time trends

- Assess existence or nature of time trends with:
  - correlation
  - regression ideas.
  - (later) time series analysis

World mean temperatures

Global mean temperature every year since 1880:

```
temp=read_csv("temperature.csv")
ggplot(temp, aes(x=year, y=temperature)) +
  geom_point() + geom_smooth()
```



Examining trend

- Temperatures increasing on average over time, but pattern very irregular.
- Find (Pearson) correlation with time, and test for significance:

```
with(temp, cor.test(temperature,year))
```

```
##
##  Pearson's product-moment correlation
##
## data:  temperature and year
## t = 19.996, df = 129, p-value < 2.2e-16
## alternative hypothesis: true correlation is not equal to 0
## 95 percent confidence interval:
##  0.8203548 0.9059362
## sample estimates:
##       cor
## 0.8695276
```

Comments

- Correlation, 0.8695, significantly different from zero.
- CI shows how far from zero it is.

Tests for *linear* trend with *normal* data.

Kendall correlation

Alternative, Kendall (rank) correlation, which just tests for monotone trend (anything upward, anything downward) and is resistant to outliers:

```
with(temp, cor.test(temperature,year,method="kendall"))
```

```
##
##  Kendall's rank correlation tau
##
## data:  temperature and year
## z = 11.776, p-value < 2.2e-16
## alternative hypothesis: true tau is not equal to 0
## sample estimates:
##       tau
## 0.6992574
```

Kendall correlation usually closer to 0 for same data, but here P-values comparable. Trend again strongly significant.

Mann-Kendall

- Another way is via **Mann-Kendall**: Kendall correlation with time.
- Use my package `mkac`:

```
kendall_Z_adjusted(temp$temperature)
```

```
## $z
## [1] 11.77267
##
## $z_star
## [1] 4.475666
##
## $ratio
## [1] 6.918858
##
## $P_value
## [1] 0
##
## $P_value_adj
## [1] 7.617357e-06
```

Comments

- Standard Mann-Kendall assumes observations *independent*.
- Observations close together in time often *correlated* with each other.
- Correlation of time series "with itself" called **autocorrelation**.
- Adjusted P-value above is correction for autocorrelation.

Examining rate of change

- Having seen that there *is* a change, question is “how fast is it?”
- Examine slopes:
  - regular regression slope, if you believe straight-line regression
  - Theil-Sen slope: resistant to outliers, based on medians

Ordinary regression against time

```
lm(temperature~year, data=temp) %>% tidy() -> temp.tidy
temp.tidy
```

term	estimate	std.error	statistic	p.value
(Intercept)	2.5794197	0.5703984	4.522137	1.37e-05
year	0.0058631	0.0002932	19.996448	0.00e+00

- Slope about 0.006 degrees per year
- about this many degrees over course of data):

```
temp.tidy %>% pluck("estimate", 2)*130
```

```
## [1] 0.7622068
```

Theil-Sen slope

also from `mkac`:

```
theil_sen_slope(temp$temperature)
```

```
## [1] 0.005675676
```

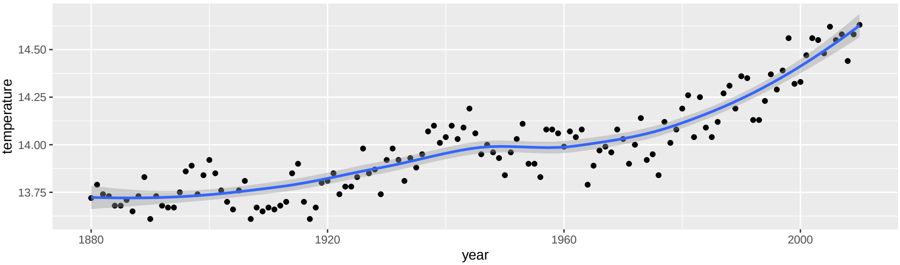
Conclusions

- Slopes:
  - Linear regression: 0.005863
  - Theil-Sen slope: 0.005676
  - Very close.
- Correlations:
  - Pearson 0.8675
  - Kendall 0.6993
  - Kendall correlation smaller, but P-value equally significant (often the case)

Constant rate of change?

Slope assumes that the rate of change is same over all years, but trend seemed to be accelerating:

```
ggplot(temp, aes(x=year, y=temperature)) +
  geom_point() + geom_smooth()
```



Pre-1970 and post-1970:

```
temp %>%
  mutate(time_period=
           ifelse(year<=1970, "pre-1970", "post-1970")) %>%
  nest(-time_period) %>%
  mutate(theil_sen=map_dbl(
    data, ~theil_sen_slope(.$temperature)))
```

```
## Warning: All elements of `...` must be named.
## Did you want `data = c(X1, Year, temperature, year)`?
```

time_period	data	the
pre-1970	1.00, 2.00, 3.00, 4.00, 5.00, 6.00, 7.00, 8.00, 9.00, 10.00, 11.00, 12.00, 13.00, 14.00, 15.00, 16.00, 17.00, 18.00, 19.00, 20.00, 21.00, 22.00, 23.00, 24.00, 25.00, 26.00, 27.00, 28.00, 29.00, 30.00, 31.00, 32.00, 33.00, 34.00, 35.00, 36.00, 37.00, 38.00, 39.00, 40.00, 41.00,	0.0

Actual time series: the Kings of England

- Age at death of Kings and Queens of England since William the Conqueror (1066):

```
kings=read_table("kings.txt", col_names=F)
```

```
##
## -- Column specification ----------------------------
## cols(
##   X1 = col_double()
## )
```

Data in one long column `X1`, so `kings` is data frame with one column.

Turn into ts time series object

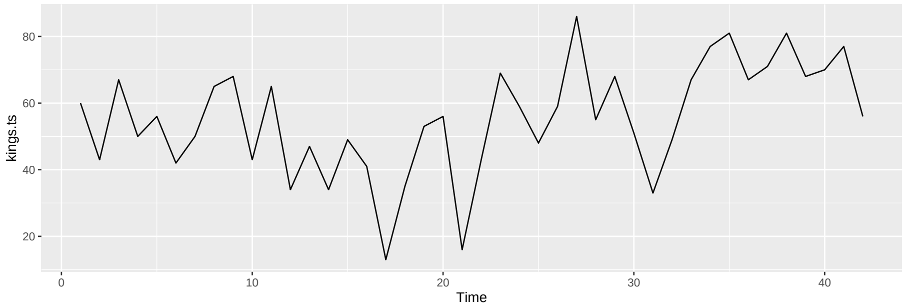
```
kings.ts=ts(kings)
kings.ts
```

```
## Time Series:
## Start = 1
## End = 42
## Frequency = 1
##       X1
##  [1,] 60
##  [2,] 43
##  [3,] 67
##  [4,] 50
##  [5,] 56
##  [6,] 42
##  [7,] 50
##  [8,] 65
```

Plotting a time series

`autoplot` from `ggfortify` gives time plot:

```
autoplot(kings.ts)
```



Comments

- "Time" here is order of monarch from William the Conqueror (1st) to George VI (last).
- Looks to be slightly increasing trend of age-at-death
- but lots of irregularity.

Stationarity

A time series is **stationary** if:

- mean is constant over time
- variability constant over time and not changing with mean.

Kings time series seems to have:

- non-constant mean
- but constant variability
- not stationary.

Getting it stationary

- Usual fix for non-stationarity is *differencing*: get new series from original one's values: 2nd - 1st, 3rd - 2nd etc.

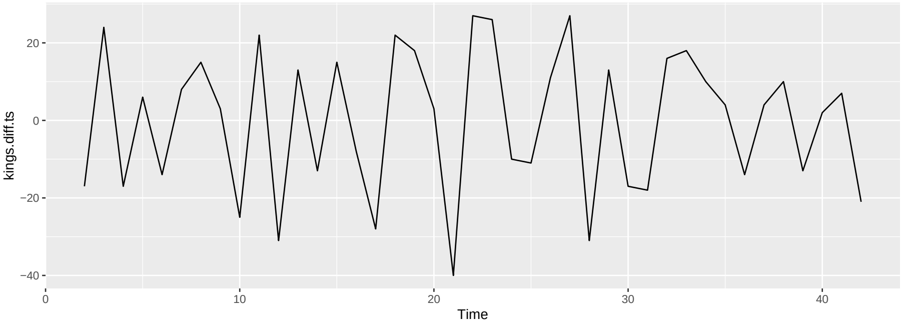
In R, `diff`:

```
kings.diff.ts=diff(kings.ts)
```

Did differencing fix stationarity?

Looks stationary now:

```r
autoplot(kings.diff.ts)
```



Births per month in New York City

from January 1946 to December 1959:

```
ny=read_table("nybirths.txt",col_names=F)
ny
```

X1
26.663
23.598
26.931
24.740
25.806
24.364
24.477
23.901
23.175
23.227
21.672
21.870

As a time series

```
ny.ts=ts(ny,freq=12,start=c(1946,1))
ny.ts
```

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

Comments

Note extras on `ts`:

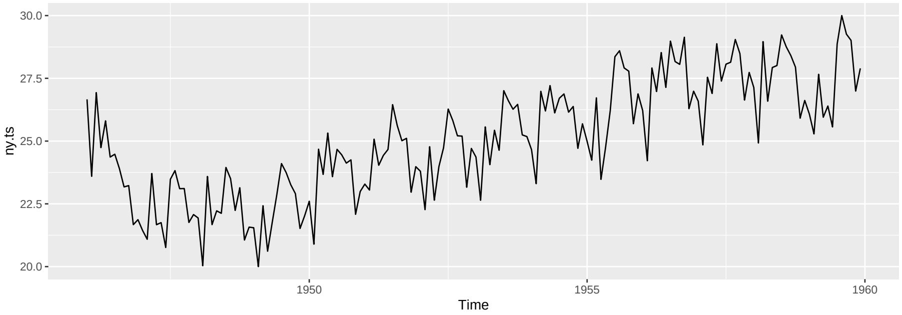
- Time period is 1 year
- 12 observations per year (monthly) in `freq`
- First observation is 1st month of 1946 in `start`

Printing formats nicely.

Time plot

- Time plot shows extra pattern:

```
autoplot(ny.ts)
```



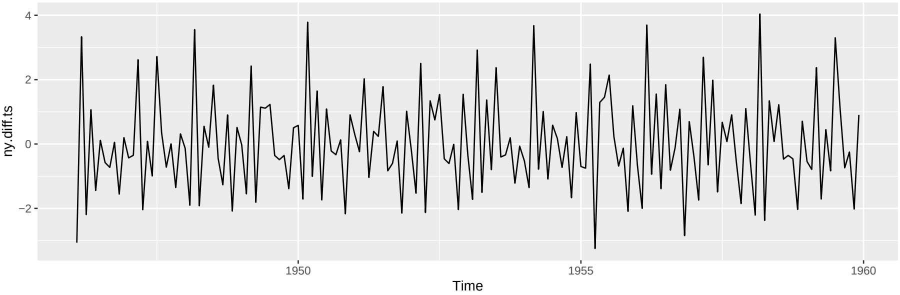
Comments on time plot

- steady increase (after initial drop)
- repeating pattern each year (seasonal component).
- Not stationary.

Differencing the New York births

Does differencing help here? Looks stationary, but some regular spikes:

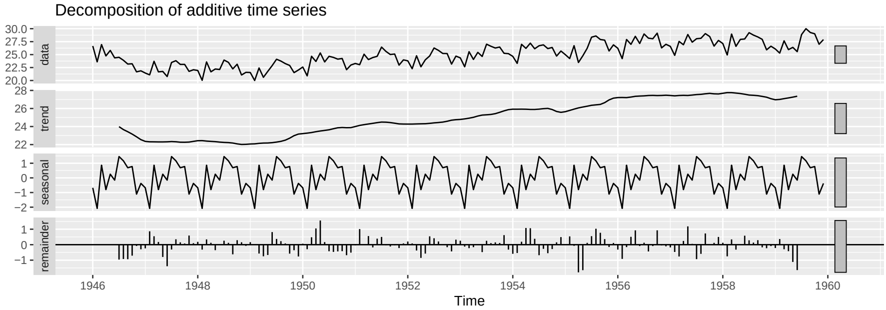
```
ny.diff.ts=diff(ny.ts)
autoplot(ny.diff.ts)
```



Decomposing a seasonal time series

A visual (using original data):

```
ny.d <- decompose(ny.ts)
ny.d %>% autoplot()
```



Decomposition bits

Shows:

- original series
- a "seasonal" part: something that repeats every year
- just the trend, going steadily up (except at the start)
- random: what is left over ("remainder")

The seasonal part

Fitted seasonal part is same every year, births lowest in February and highest in July:

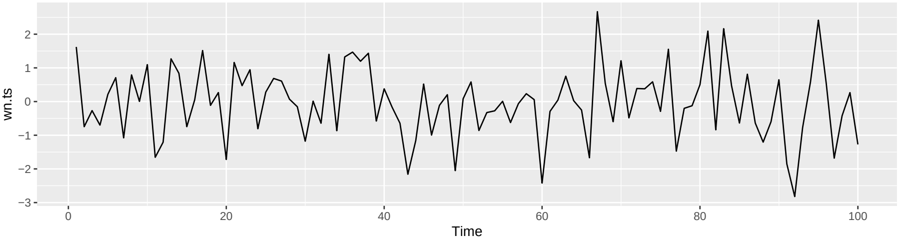
```
ny.d$seasonal
```

```
##            Jan        Feb        Mar        Apr        May        Jun        Jul        Aug
## 1946 -0.6771947 -2.0829607  0.8625232 -0.8016787  0.2516514 -0.1532556  1.4560457  1.1645938
## 1947 -0.6771947 -2.0829607  0.8625232 -0.8016787  0.2516514 -0.1532556  1.4560457  1.1645938
## 1948 -0.6771947 -2.0829607  0.8625232 -0.8016787  0.2516514 -0.1532556  1.4560457  1.1645938
## 1949 -0.6771947 -2.0829607  0.8625232 -0.8016787  0.2516514 -0.1532556  1.4560457  1.1645938
## 1950 -0.6771947 -2.0829607  0.8625232 -0.8016787  0.2516514 -0.1532556  1.4560457  1.1645938
## 1951 -0.6771947 -2.0829607  0.8625232 -0.8016787  0.2516514 -0.1532556  1.4560457  1.1645938
## 1952 -0.6771947 -2.0829607  0.8625232 -0.8016787  0.2516514 -0.1532556  1.4560457  1.1645938
## 1953 -0.6771947 -2.0829607  0.8625232 -0.8016787  0.2516514 -0.1532556  1.4560457  1.1645938
## 1954 -0.6771947 -2.0829607  0.8625232 -0.8016787  0.2516514 -0.1532556  1.4560457  1.1645938
## 1955 -0.6771947 -2.0829607  0.8625232 -0.8016787  0.2516514 -0.1532556  1.4560457  1.1645938
## 1956 -0.6771947 -2.0829607  0.8625232 -0.8016787  0.2516514 -0.1532556  1.4560457  1.1645938
## 1957 -0.6771947 -2.0829607  0.8625232 -0.8016787  0.2516514 -0.1532556  1.4560457  1.1645938
## 1958 -0.6771947 -2.0829607  0.8625232 -0.8016787  0.2516514 -0.1532556  1.4560457  1.1645938
## 1959 -0.6771947 -2.0829607  0.8625232 -0.8016787  0.2516514 -0.1532556  1.4560457  1.1645938
##            Sep        Oct        Nov        Dec
## 1946  0.6916162  0.7752444 -1.1097652 -0.3768197
## 1947  0.6916162  0.7752444 -1.1097652 -0.3768197
## 1948  0.6916162  0.7752444 -1.1097652 -0.3768197
## 1949  0.6916162  0.7752444 -1.1097652 -0.3768197
## 1950  0.6916162  0.7752444 -1.1097652 -0.3768197
## 1951  0.6916162  0.7752444 -1.1097652 -0.3768197
## 1952  0.6916162  0.7752444 -1.1097652 -0.3768197
## 1953  0.6916162  0.7752444 -1.1097652 -0.3768197
## 1954  0.6916162  0.7752444 -1.1097652 -0.3768197
```

Time series basics: white noise

Each value independent random normal. Knowing one value tells you nothing about the next. “Random” process.

```
wn=rnorm(100)
wn.ts=ts(wn)
autoplot(wn.ts)
```



Lagging a time series

This means moving a time series one (or more) steps back in time:

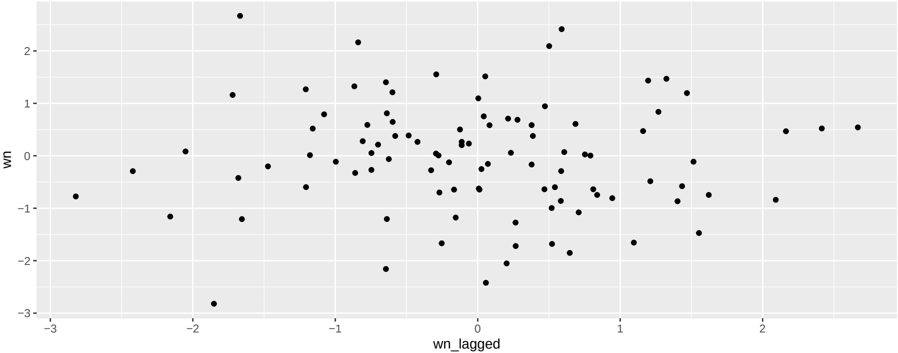
```
x=rnorm(5)
tibble(x) %>% mutate(x_lagged=lag(x)) -> with_lagged
with_lagged
```

x	x_lagged
-2.0360948	NA
-0.5786158	-2.0360948
0.6083646	-0.5786158
0.1180334	0.6083646
0.0563443	0.1180334

Gain a missing because there is nothing before the first observation.

Lagging white noise

```
tibble(wn) %>% mutate(wn_lagged=lag(wn)) -> wn_with_lagged
ggplot(wn_with_lagged, aes(y=wn, x=wn_lagged))+geom_point()
```




```
with(wn_with_lagged, cor.test(wn, wn_lagged, use="c")) # ignor
```

```
##
```

Correlation with lagged series

If you know about white noise at one time point, you know *nothing* about it at the next. This is shown by the scatterplot and the correlation.

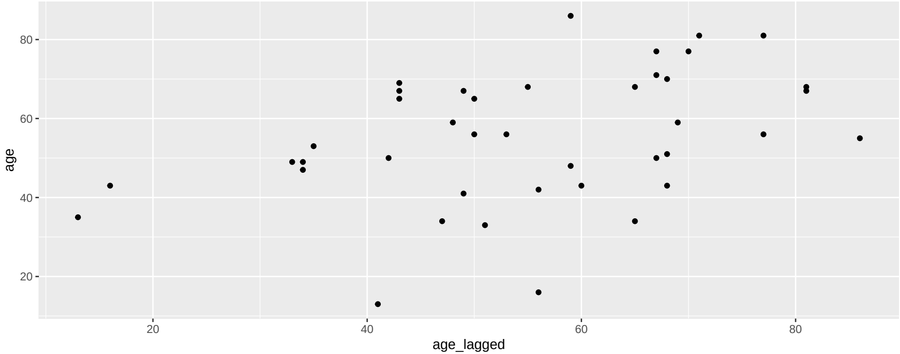
On the other hand, this:

```
tibble(age=kings$X1) %>%
  mutate(age_lagged=lag(age)) -> kings_with_lagged
with(kings_with_lagged, cor.test(age, age_lagged))
```

```
##
##  Pearson's product-moment correlation
##
## data:  age and age_lagged
## t = 2.7336, df = 39, p-value = 0.00937
## alternative hypothesis: true correlation is not equal to 0
## 95 percent confidence interval:
##  0.1064770 0.6308209
## sample estimates:
##       cor
## 0.4009919
```

Correlation with next value?

```
ggplot(kings_with_lagged, aes(x=age_lagged, y=age)) +
  geom_point()
```



Two steps back:

```
kings_with_lagged %>%
  mutate(age_lag_2=lag(age_lagged)) %>%
  with(., cor.test(age, age_lag_2))
```

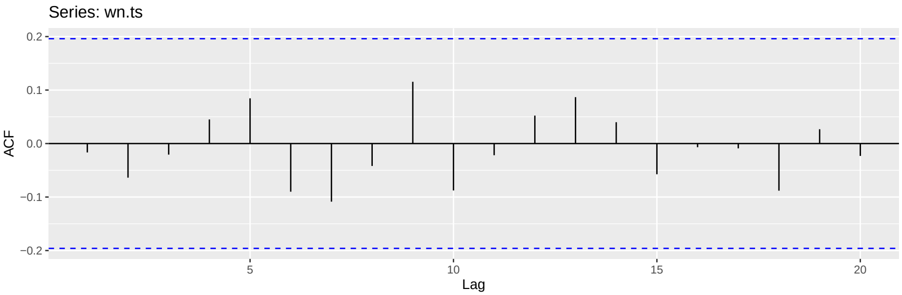
```
##
##  Pearson's product-moment correlation
##
## data:  age and age_lag_2
## t = 1.5623, df = 38, p-value = 0.1265
## alternative hypothesis: true correlation is not equal to 0
## 95 percent confidence interval:
##  -0.07128917  0.51757510
## sample estimates:
##      cor
## 0.245676
```

Still a correlation two steps back, but smaller (and no longer significant).

Autocorrelation

Correlation of time series with *itself* one, two,... time steps back is useful idea, called **autocorrelation**. Make a plot of it with `acf` and `autoplot`. Here, white noise:

```
acf(wn.ts, plot=F) %>% autoplot()
```

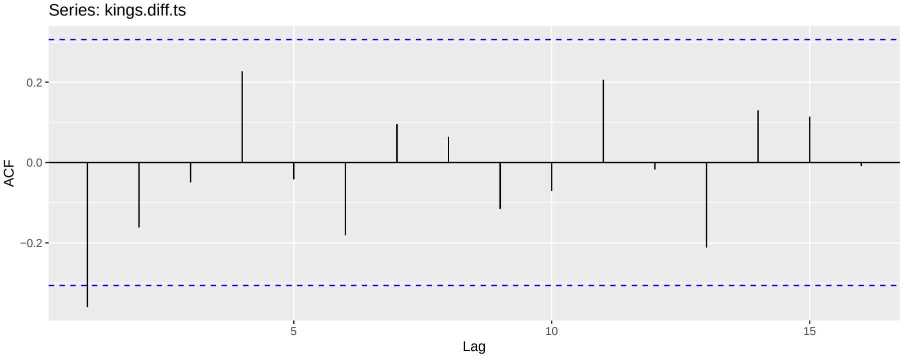


No autocorrelations beyond chance, anywhere (except *possibly* at lag 13).

Autocorrelations work best on *stationary* series.

Kings, differenced

```
acf(kings.diff.ts, plot=F) %>% autoplot()
```



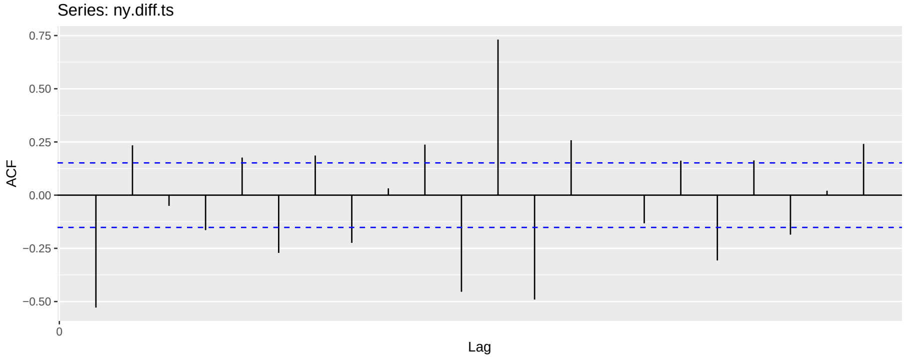
Comments on autocorrelations of kings series

Negative autocorrelation at lag 1, nothing beyond that.

- If one value of differenced series positive, next one most likely negative.
- If one monarch lives longer than predecessor, next one likely lives shorter.

NY births, differenced

```r
acf(ny.diff.ts, plot=F) %>% autoplot()
```



Lots of stuff:

- large positive autocorrelation at 1.0 years (July one year like July last year)
- large negative autocorrelation at 1 month.
- smallish but significant negative autocorrelation at 0.5 year $= 6$ months.
- Other stuff – complicated.

Souvenir sales

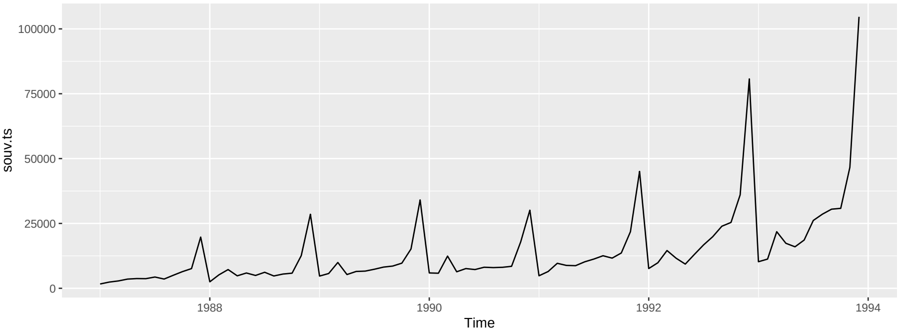
Monthly sales for a beach souvenir shop in Queensland, Australia:

```
souv=read_table("souvenir.txt", col_names=F)
souv.ts=ts(souv,frequency=12,start=1987)
souv.ts
```

```
##              Jan       Feb       Mar       Apr       May       Jun       Jul
## 1987   1664.81   2397.53   2840.71   3547.29   3752.96   3714.74   4349.61
## 1988   2499.81   5198.24   7225.14   4806.03   5900.88   4951.34   6179.12
## 1989   4717.02   5702.63   9957.58   5304.78   6492.43   6630.80   7349.62
## 1990   5921.10   5814.58  12421.25   6369.77   7609.12   7224.75   8121.22
## 1991   4826.64   6470.23   9638.77   8821.17   8722.37  10209.48  11276.55
## 1992   7615.03   9849.69  14558.40  11587.33   9332.56  13082.09  16732.78
## 1993  10243.24  11266.88  21826.84  17357.33  15997.79  18601.53  26155.15
##              Aug       Sep       Oct       Nov       Dec
## 1987   3566.34   5021.82   6423.48   7600.60  19756.21
## 1988   4752.15   5496.43   5835.10  12600.08  28541.72
## 1989   8176.62   8573.17   9690.50  15151.84  34061.01
## 1990   7979.25   8093.06   8476.70  17914.66  30114.41
## 1991  12552.22  11637.39  13606.89  21822.11  45060.69
## 1992  19888.61  23933.38  25391.35  36024.80  80721.71
## 1993  28586.52  30505.41  30821.33  46634.38 104660.67
```

Plot of souvenir sales

```r
autoplot(souv.ts)
```



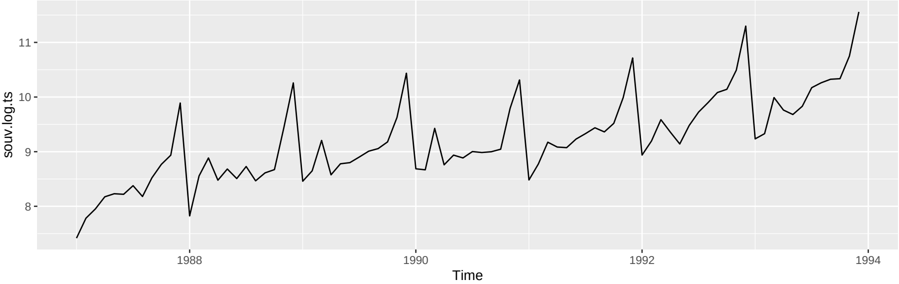
Several problems:

- Mean goes up over time
- Variability gets larger as mean gets larger
- Not stationary

Problem-fixing:

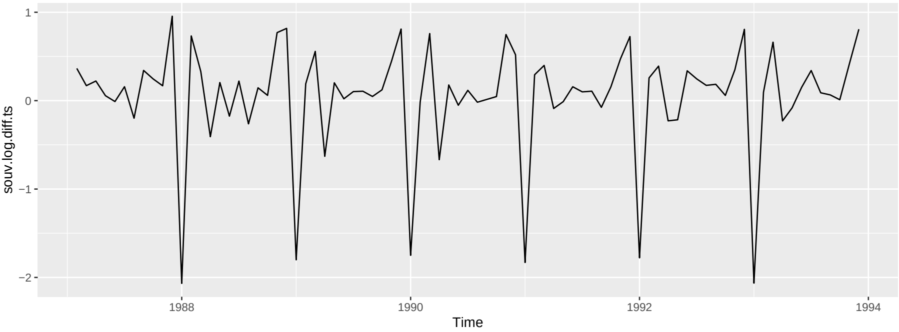
Fix non-constant variability first by taking logs:

```
souv.log.ts=log(souv.ts)
autoplot(souv.log.ts)
```



Mean still not constant, so try taking differences

```
souv.log.diff.ts=diff(souv.log.ts)
autoplot(souv.log.diff.ts)
```

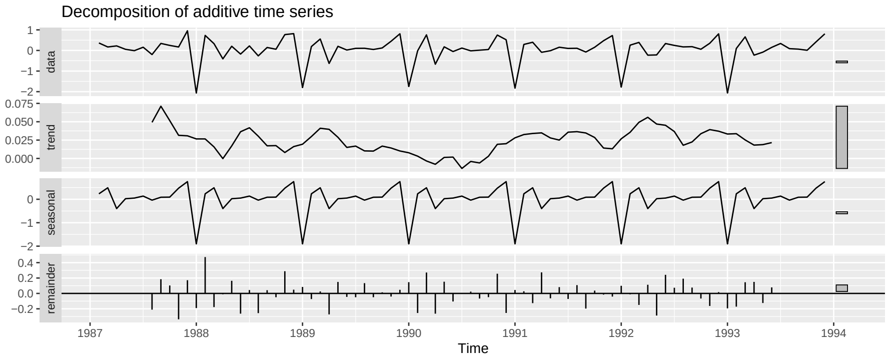


Comments

- Now stationary
- but clear seasonal effect.

Decomposing to see the seasonal effect

```
souv.d=decompose(souv.log.diff.ts)
autoplot(souv.d)
```



Comments

Big drop in one month's differences. Look at seasonal component to see which:

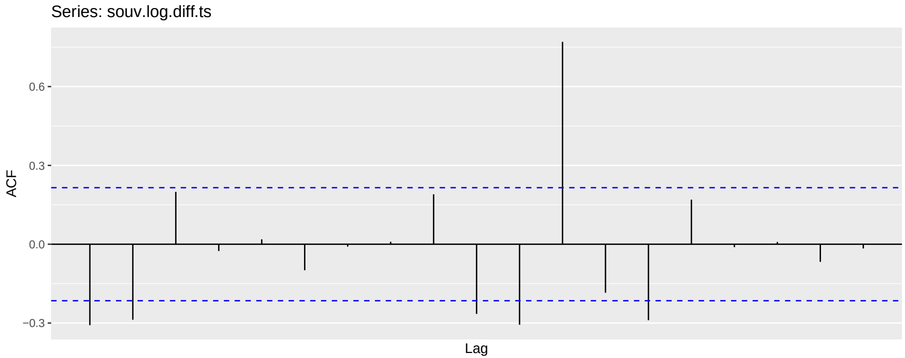
```
souv.d$seasonal
```

```
##             Jan         Feb         Mar         Apr         May         Jun         Jul
## 1987              0.23293343  0.49068755 -0.39700942  0.02410429  0.05074206  0.13552988
## 1988 -1.90372141  0.23293343  0.49068755 -0.39700942  0.02410429  0.05074206  0.13552988
## 1989 -1.90372141  0.23293343  0.49068755 -0.39700942  0.02410429  0.05074206  0.13552988
## 1990 -1.90372141  0.23293343  0.49068755 -0.39700942  0.02410429  0.05074206  0.13552988
## 1991 -1.90372141  0.23293343  0.49068755 -0.39700942  0.02410429  0.05074206  0.13552988
## 1992 -1.90372141  0.23293343  0.49068755 -0.39700942  0.02410429  0.05074206  0.13552988
## 1993 -1.90372141  0.23293343  0.49068755 -0.39700942  0.02410429  0.05074206  0.13552988
##             Aug         Sep         Oct         Nov         Dec
## 1987 -0.03710275  0.08650584  0.09148236  0.47311204  0.75273614
## 1988 -0.03710275  0.08650584  0.09148236  0.47311204  0.75273614
## 1989 -0.03710275  0.08650584  0.09148236  0.47311204  0.75273614
## 1990 -0.03710275  0.08650584  0.09148236  0.47311204  0.75273614
## 1991 -0.03710275  0.08650584  0.09148236  0.47311204  0.75273614
## 1992 -0.03710275  0.08650584  0.09148236  0.47311204  0.75273614
## 1993 -0.03710275  0.08650584  0.09148236  0.47311204  0.75273614
```

January.

Autocorrelations

```r
acf(souv.log.diff.ts, plot=F) %>% autoplot()
```



- Big positive autocorrelation at 1 year (strong seasonal effect)
- Small negative autocorrelation at 1 and 2 months.

Moving average

- A particular type of time series called a **moving average** or MA process captures idea of autocorrelations at a few lags but not at others.
- Here's generation of MA(1) process, with autocorrelation at lag 1 but not otherwise:

```
beta=1
tibble(e=rnorm(100)) %>%
  mutate(e_lag=lag(e)) %>%
  mutate(y=e+beta*e_lag) %>%
  mutate(y=ifelse(is.na(y), 0, y)) -> ma
```

The series

```
ma
```

e	e_lag	y
0.0779806	NA	0.0000000
1.6644479	0.0779806	1.7424284
-1.4539253	1.6644479	0.2105226
-0.4015166	-1.4539253	-1.8554419
0.6809716	-0.4015166	0.2794549
0.4051565	0.6809716	1.0861281
0.2755077	0.4051565	0.6806642
-0.1823334	0.2755077	0.0931743
0.2264065	-0.1823334	0.0440731
0.3606240	0.2264065	0.5870305
2.2764053	0.3606240	2.6370293
-1.7780947	2.2764053	0.4983106
0.9387412	-1.7780947	-0.8393535
0.8939353	0.9387412	1.8326765
-0.1715134	0.8939353	0.7224219

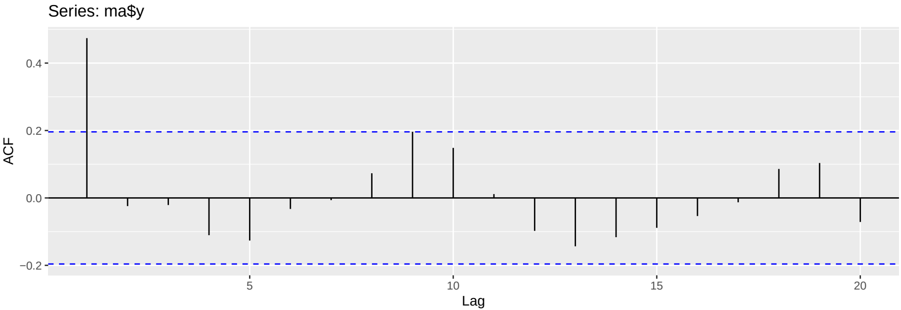
Comments

- e contains independent "random shocks".
- Start process at 0.
- Then, each value of the time series has that time's random shock, plus a multiple of the last time's random shock.
- `y[i]` has shock in common with `y[i-1]`; should be a lag 1 autocorrelation.
- But `y[i]` has no shock in common with `y[i-2]`, so no lag 2 autocorrelation (or beyond).

ACF for MA(1) process

Significant at lag 1, but beyond, just chance:

```
acf(ma$y, plot=F, na.rm=T) %>% autoplot()
```



AR process

Another kind of time series is AR process, where each value depends on previous one, like this (loop):

```
e=rnorm(100)
x=numeric(0)
x[1]=0
alpha=0.7
for (i in 2:100)
{
  x[i]=alpha*x[i-1]+e[i]
}
```

The series

```
x
```

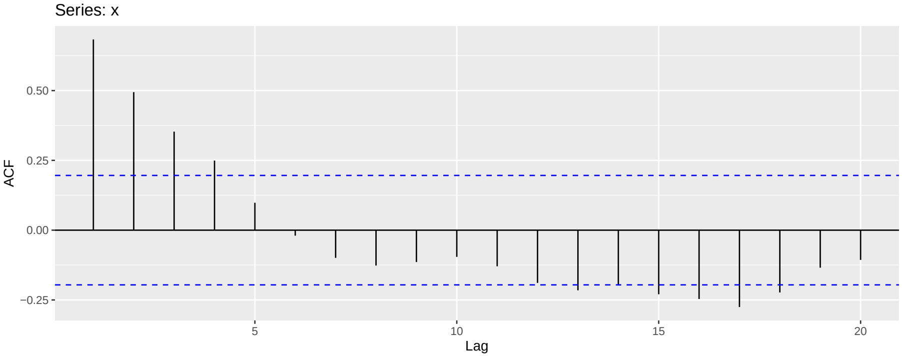
```
##   [1]  0.00000000  0.48114799  0.71103525
##   [4]  0.65900463  0.90960457  1.12185864
##   [7]  1.02390502 -0.69991592 -0.64803277
##  [10]  0.23657047  0.16418958 -0.15043538
##  [13] -0.09667020  0.88635091  1.65971423
##  [16]  2.62912167  1.43019873  1.45511765
##  [19]  3.11845464  2.95303071  0.36676230
##  [22]  0.22714525  0.48741737 -1.12005103
##  [25]  0.83144302 -0.07876862  0.52370605
##  [28] -0.32393795 -0.31129337  2.06203136
##  [31]  1.74299095  1.93791340  1.04509322
##  [34]  0.68711330  1.83912870  1.06043342
##  [37]  2.56960344  1.72169161  1.23413651
##  [40]  1.17561493  3.12403601  1.58765927
##  [43]  0.13744074 -0.05372973 -0.44291441
```

Comments

- Each random shock now only used for its own value of `x`
- but `x[i]` also depends on previous value `x[i-1]`
- so correlated with previous value
- *but* `x[i]` also contains multiple of `x[i-2]` and previous x's
- so all x's correlated, but autocorrelation dying away.

ACF for AR(1) series

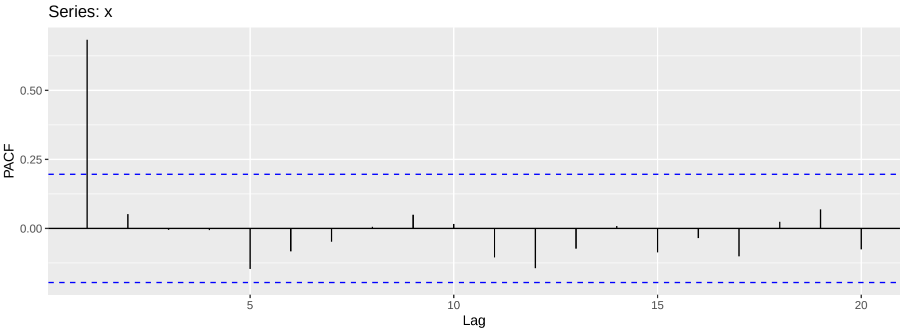
```
acf(x, plot=F) %>% autoplot()
```



Partial autocorrelation function

This cuts off for an AR series:

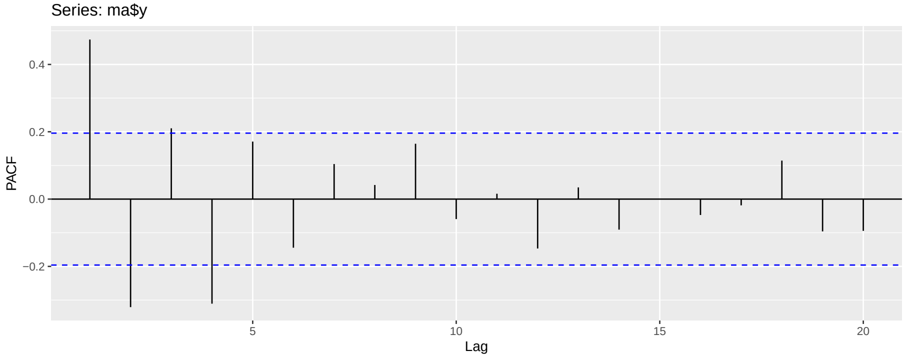
```
pacf(x, plot=F) %>% autoplot()
```



The lag-2 autocorrelation should not be significant, and isn't.

PACF for an MA series decays slowly

```r
pacf(ma$y, plot=F) %>% autoplot()
```



The old way of doing time series analysis

Starting from a series with constant variability (eg. transform first to get it, as for souvenirs):

- Assess stationarity.
- If not stationary, take differences as many times as needed until it is.
- Look at ACF, see if it dies off. If it does, you have MA series.
- Look at PACF, see if that dies off. If it does, have AR series.
- If neither dies off, probably have a mixed "ARMA" series.
- Fit coefficients (like regression slopes).
- Do forecasts.

The new way of doing time series analysis (in R)

- Transform series if needed to get constant variability
- Use package `forecast`.
- Use function `auto.arima` to estimate what kind of series best fits data.
- Use `forecast` to see what will happen in future.

Anatomy of auto.arima output

```r
auto.arima(ma$y)
```

```
## Series: ma$y
## ARIMA(5,0,0) with zero mean
##
## Coefficients:
##          ar1      ar2     ar3      ar4     ar5
##       0.8477  -0.7214  0.5633  -0.4953  0.1973
## s.e.  0.0988   0.1221  0.1289   0.1239  0.1037
##
## sigma^2 estimated as 1.273:  log likelihood=-152.06
## AIC=316.12   AICc=317.03   BIC=331.76
```

Comments over.

Comments

- ARIMA part tells you what kind of series you are estimated to have:
  - first number (first 0) is AR (autoregressive) part
  - second number (second 0) is amount of differencing here
  - third number (1) is MA (moving average) part
- Below that, coefficients (with SEs)
- AICc is measure of fit (lower better)

What other models were possible?

Run `auto.arima` with `trace=T`:

```
auto.arima(ma$y,trace=T)
```

```
##
##  ARIMA(2,0,2) with non-zero mean : Inf
##  ARIMA(0,0,0) with non-zero mean : 365.3271
##  ARIMA(1,0,0) with non-zero mean : 342.1337
##  ARIMA(0,0,1) with non-zero mean : Inf
##  ARIMA(0,0,0) with zero mean     : 363.2452
##  ARIMA(2,0,0) with non-zero mean : 332.1106
##  ARIMA(3,0,0) with non-zero mean : 329.8212
##  ARIMA(4,0,0) with non-zero mean : 320.5205
##  ARIMA(5,0,0) with non-zero mean : 319.3257
##  ARIMA(5,0,1) with non-zero mean : Inf
##  ARIMA(4,0,1) with non-zero mean : Inf
##  ARIMA(5,0,0) with zero mean     : 317.0272
##  ARIMA(4,0,0) with zero mean     : 318.2995
##  ARIMA(5,0,1) with zero mean     : Inf
```

Doing it all the new way: white noise

```
wn.aa=auto.arima(wn.ts)
wn.aa
```

```
## Series: wn.ts
## ARIMA(0,0,0) with zero mean
##
## sigma^2 estimated as 1.111:  log likelihood=-147.16
## AIC=296.32   AICc=296.36   BIC=298.93
```

Best fit *is* white noise (no AR, no MA, no differencing).

Forecasts:

```
forecast(wn.aa)
```

```
##      Point Forecast     Lo 80    Hi 80     Lo 95    Hi 95
## 101              0 -1.350869 1.350869 -2.065975 2.065975
## 102              0 -1.350869 1.350869 -2.065975 2.065975
## 103              0 -1.350869 1.350869 -2.065975 2.065975
## 104              0 -1.350869 1.350869 -2.065975 2.065975
## 105              0 -1.350869 1.350869 -2.065975 2.065975
## 106              0 -1.350869 1.350869 -2.065975 2.065975
## 107              0 -1.350869 1.350869 -2.065975 2.065975
## 108              0 -1.350869 1.350869 -2.065975 2.065975
## 109              0 -1.350869 1.350869 -2.065975 2.065975
## 110              0 -1.350869 1.350869 -2.065975 2.065975
```

Forecasts all 0, since the past doesn't help to predict future.

MA(1)

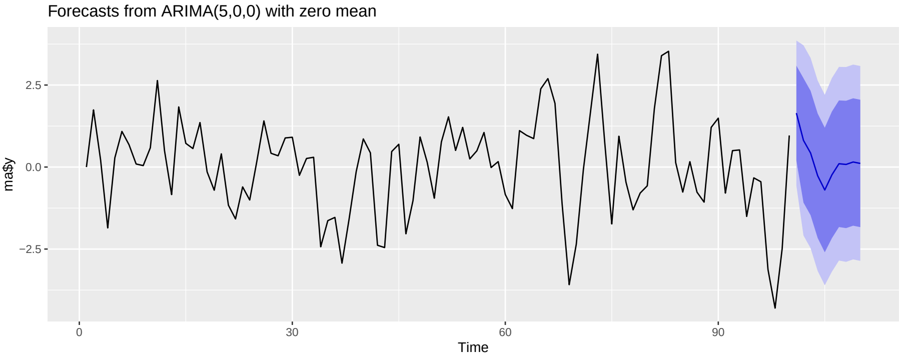
```
y.aa=auto.arima(ma$y)
y.aa
```

```
## Series: ma$y
## ARIMA(5,0,0) with zero mean
##
## Coefficients:
##          ar1      ar2     ar3      ar4     ar5
##       0.8477  -0.7214  0.5633  -0.4953  0.1973
## s.e.  0.0988   0.1221  0.1289   0.1239  0.1037
##
## sigma^2 estimated as 1.273:  log likelihood=-152.06
## AIC=316.12   AICc=317.03   BIC=331.76
```

```
y.f=forecast(y.aa)
```

Plotting the forecasts for MA(1)

```
autoplot(y.f)
```



AR(1)

```
x.aa=auto.arima(x)
x.aa
```

```
## Series: x
## ARIMA(1,0,0) with non-zero mean
##
## Coefficients:
##          ar1    mean
##       0.6843  0.4811
## s.e.  0.0720  0.2866
##
## sigma^2 estimated as 0.8717:  log likelihood=-134.33
## AIC=274.66   AICc=274.91   BIC=282.48
```

Oops! Thought it was MA(1), not AR(1)!

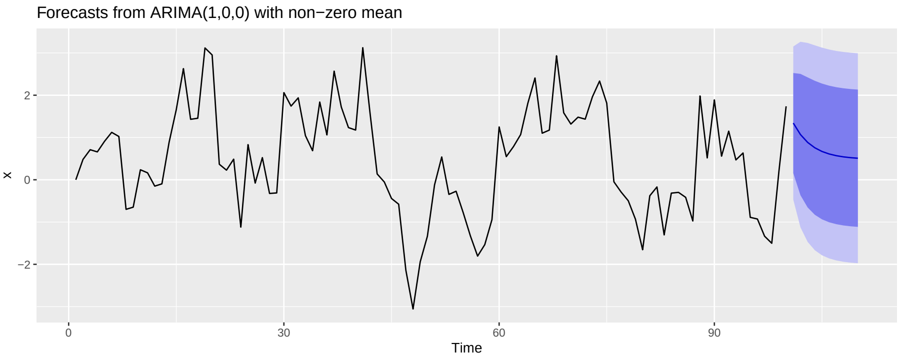
Fit right AR(1) model:

```
x.arima=arima(x,order=c(1,0,0))
x.arima
```

```
##
## Call:
## arima(x = x, order = c(1, 0, 0))
##
## Coefficients:
##          ar1  intercept
##       0.6843     0.4811
## s.e.  0.0720     0.2866
##
## sigma^2 estimated as 0.8542:  log likelihood = -134.33,  ai
```

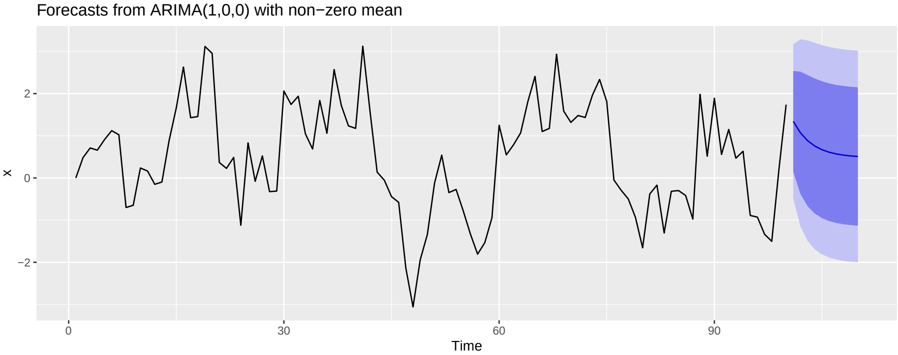
Forecasts for x

```r
forecast(x.arima) %>% autoplot()
```



Comparing wrong model:

```
forecast(x.aa) %>% autoplot()
```



Kings

```
kings.aa=auto.arima(kings.ts)
kings.aa
```

```
## Series: kings.ts
## 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
```

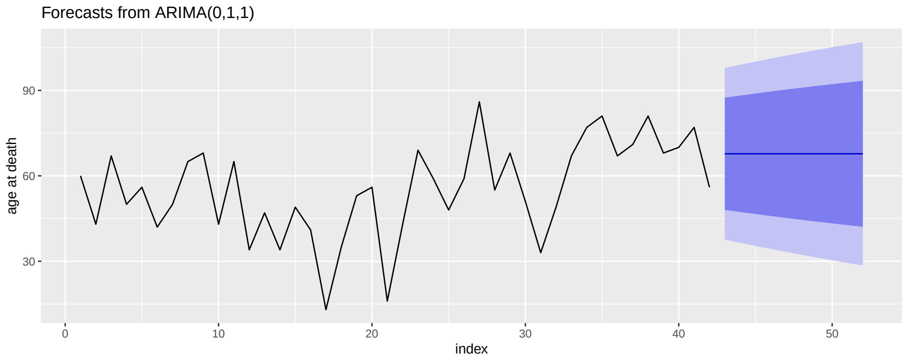
Kings forecasts:

```
kings.f=forecast(kings.aa)
kings.f
```

```
##    Point Forecast    Lo 80    Hi 80    Lo 95     Hi 95
## 43       67.75063 48.05479 87.44646 37.62845  97.87281
## 44       67.75063 47.30662 88.19463 36.48422  99.01703
## 45       67.75063 46.58489 88.91637 35.38042 100.12084
## 46       67.75063 45.88696 89.61429 34.31304 101.18822
## 47       67.75063 45.21064 90.29062 33.27869 102.22257
## 48       67.75063 44.55402 90.94723 32.27448 103.22678
## 49       67.75063 43.91549 91.58577 31.29793 104.20333
## 50       67.75063 43.29362 92.20763 30.34687 105.15439
## 51       67.75063 42.68718 92.81408 29.41939 106.08187
## 52       67.75063 42.09507 93.40619 28.51383 106.98742
```

Kings forecasts, plotted

```
autoplot(kings.f) + labs(x="index", y= "age at death")
```



NY births

Very complicated:

```
ny.aa=auto.arima(ny.ts)
ny.aa
```

```
## Series: ny.ts
## ARIMA(2,1,2)(1,1,1)[12]
##
## Coefficients:
##          ar1      ar2      ma1     ma2     sar1     sma1
##       0.6539  -0.4540  -0.7255  0.2532  -0.2427  -0.8451
## s.e.  0.3003   0.2429   0.3227  0.2878   0.0985   0.0995
##
## sigma^2 estimated as 0.4076:  log likelihood=-157.45
## AIC=328.91   AICc=329.67   BIC=350.21
```

NY births forecasts

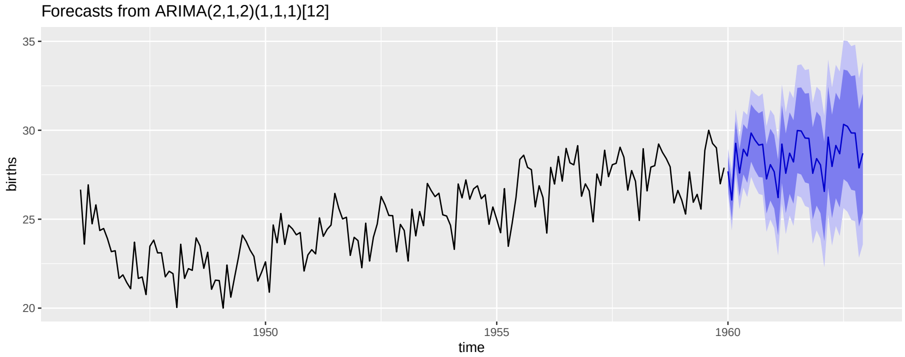
Not *quite* same every year:

```r
ny.f=forecast(ny.aa,h=36)
ny.f
```

```
##          Point Forecast    Lo 80    Hi 80    Lo 95    Hi 95
## Jan 1960       27.69056 26.87069 28.51043 26.43668 28.94444
## Feb 1960       26.07680 24.95838 27.19522 24.36632 27.78728
## Mar 1960       29.26544 28.01566 30.51523 27.35406 31.17683
## Apr 1960       27.59444 26.26555 28.92333 25.56208 29.62680
## May 1960       28.93193 27.52089 30.34298 26.77392 31.08995
## Jun 1960       28.55379 27.04381 30.06376 26.24448 30.86309
## Jul 1960       29.84713 28.23370 31.46056 27.37960 32.31466
## Aug 1960       29.45347 27.74562 31.16132 26.84155 32.06539
## Sep 1960       29.16388 27.37259 30.95517 26.42433 31.90342
## Oct 1960       29.21343 27.34498 31.08188 26.35588 32.07098
## Nov 1960       27.26221 25.31879 29.20563 24.29000 30.23441
## Dec 1960       28.06863 26.05137 30.08589 24.98349 31.15377
## Jan 1961       27.66908 25.59684 29.74132 24.49986 30.83830
## Feb 1961       26.21255 24.08615 28.33895 22.96051 29.46460
## Mar 1961       29.22612 27.04347 31.40878 25.88804 32.56420
## Apr 1961       27.58011 25.34076 29.81945 24.15533 31.00488
## May 1961       28.71354 26.41925 31.00783 25.20473 32.22235
## Jun 1961       28.21736 25.87042 30.56429 24.62803 31.80668
## Jul 1961       29.98728 27.58935 32.38521 26.31996 33.65460
## Aug 1961       29.96127 27.51330 32.40925 26.21743 33.70512
## Sep 1961       29.56515 27.06786 32.06243 25.74588 33.38441
## Oct 1961       29.54543 26.99965 32.09121 25.65200 33.43886
## Nov 1961       27.57845 24.98510 30.17181 23.61226 31.54465
## Dec 1961       28.40796 25.76792 31.04801 24.37036 32.44556
## Jan 1962       28.05431 25.33756 30.77106 23.89939 32.20922
```

Plotting the forecasts

```
autoplot(ny.f)+labs(x="time", y="births")
```



Log-souvenir sales

```
souv.aa=auto.arima(souv.log.ts)
souv.aa
```

```
## Series: souv.log.ts
## ARIMA(2,0,0)(0,1,1)[12] with drift
##
## Coefficients:
##          ar1     ar2     sma1   drift
##       0.3470  0.3516  -0.5205  0.0238
## s.e.  0.1092  0.1115   0.1700  0.0031
##
## sigma^2 estimated as 0.02953:  log likelihood=24.54
## AIC=-39.09   AICc=-38.18   BIC=-27.71
```

```
souv.f=forecast(souv.aa,h=27)
```

The forecasts

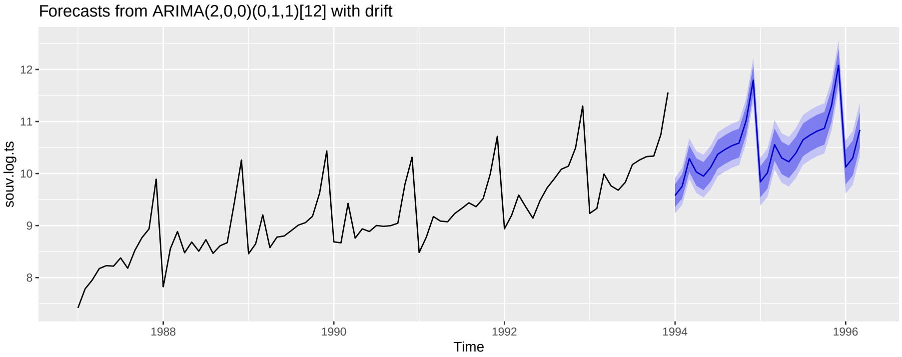
Differenced series showed low value for January (large drop). December highest, Jan and Feb lowest:

```
souv.f
```

```
##          Point Forecast     Lo 80     Hi 80     Lo 95     Hi 95
## Jan 1994       9.578291  9.358036  9.798545  9.241440  9.915141
## Feb 1994       9.754836  9.521700  9.987972  9.398285 10.111386
## Mar 1994      10.286195 10.030937 10.541453  9.895811 10.676578
## Apr 1994      10.028630  9.765727 10.291532  9.626555 10.430704
## May 1994       9.950862  9.681555 10.220168  9.538993 10.362731
## Jun 1994      10.116930  9.844308 10.389551  9.699991 10.533868
## Jul 1994      10.369140 10.094251 10.644028  9.948734 10.789545
## Aug 1994      10.460050 10.183827 10.736274 10.037603 10.882498
## Sep 1994      10.535595 10.258513 10.812677 10.111835 10.959356
## Oct 1994      10.585995 10.308386 10.863604 10.161429 11.010561
## Nov 1994      11.017734 10.739793 11.295674 10.592660 11.442807
## Dec 1994      11.795964 11.517817 12.074111 11.370575 12.221353
## Jan 1995       9.840884  9.540241 10.141527  9.381090 10.300678
## Feb 1995      10.015540  9.711785 10.319295  9.550987 10.480093
## Mar 1995      10.555070 10.246346 10.863794 10.082918 11.027222
## Apr 1995      10.299676  9.989043 10.610309  9.824604 10.774749
## May 1995      10.225535  9.913326 10.537743  9.748053 10.703017
```

Plotting the forecasts

```r
autoplot(souv.f)
```



Global mean temperatures, revisited

```
temp.ts=ts(temp$temperature,start=1880)
temp.aa=auto.arima(temp.ts)
temp.aa
```

```
## Series: temp.ts
## ARIMA(1,1,3) with drift
##
## Coefficients:
##           ar1     ma1      ma2      ma3   drift
##       -0.9374  0.5038  -0.6320  -0.2988  0.0067
## s.e.   0.0835  0.1088   0.0876   0.0844  0.0025
##
## sigma^2 estimated as 0.008939:  log likelihood=124.34
## AIC=-236.67   AICc=-235.99   BIC=-219.47
```

Forecasts

```
temp.f=forecast(temp.aa)
autoplot(temp.f)+labs(x="year", y="temperature")
```