# STAD29: Statistics for the Life and Social Sciences

Lecture notes

# Section 1

# 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: xxx

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

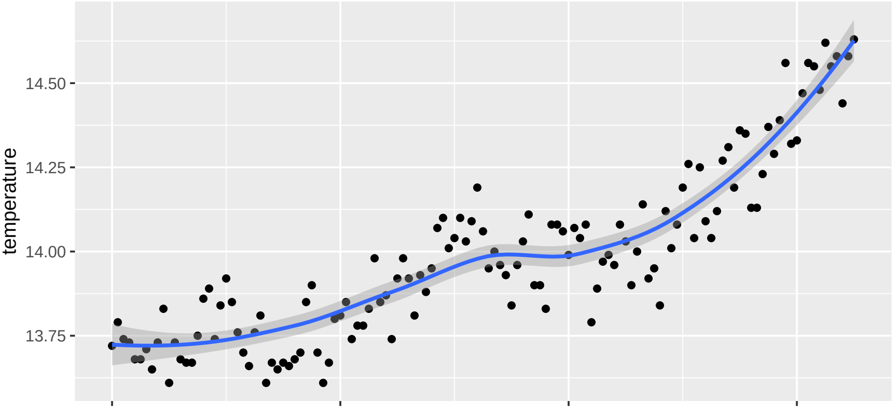
# Time trends

- 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: xxx

```
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 xxx

- 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 xxx

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

```
## # A tibble: 2 x 5
##   term        estimate std.error statistic  p.value
##   <chr>          <dbl>     <dbl>     <dbl>    <dbl>
## 1 (Intercept)  2.58     0.570         4.52 1.37e- 5
## 2 year         0.00586  0.000293     20.0  2.42e-41
```

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

```
coef(temp.lm)[2]*130
```

```
##      year
## 0.7622068
```

# Theil-Sen slope

also from `mkac`:

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

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

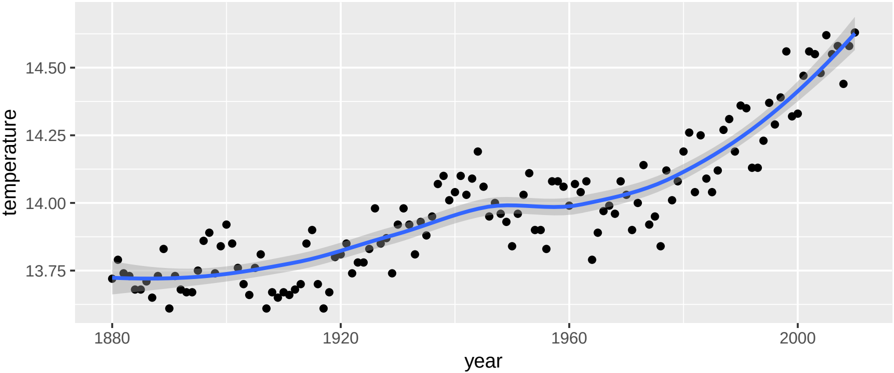
# Conclusions

- 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? xxx

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

```
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)))
```

```
## # A tibble: 2 x 3
##   time_period data                theil_sen
##   <chr>       <list>                  <dbl>
## 1 pre-1970    <tibble [91 × 4]>     0.00429
## 2 post-1970   <tibble [40 × 4]>     0.0168
```

Theil-Sen slope is very nearly *four times* as big since 1970 vs. before.

# 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)
```

```
## Parsed with 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

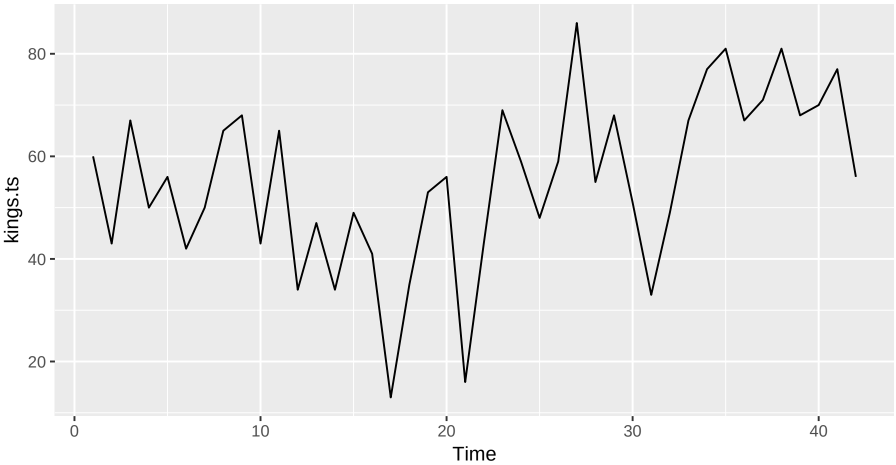
```
kings.ts=ts(kings)
kings.ts
```

```
## 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 xxx

`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.

# xxx Getting it stationary

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

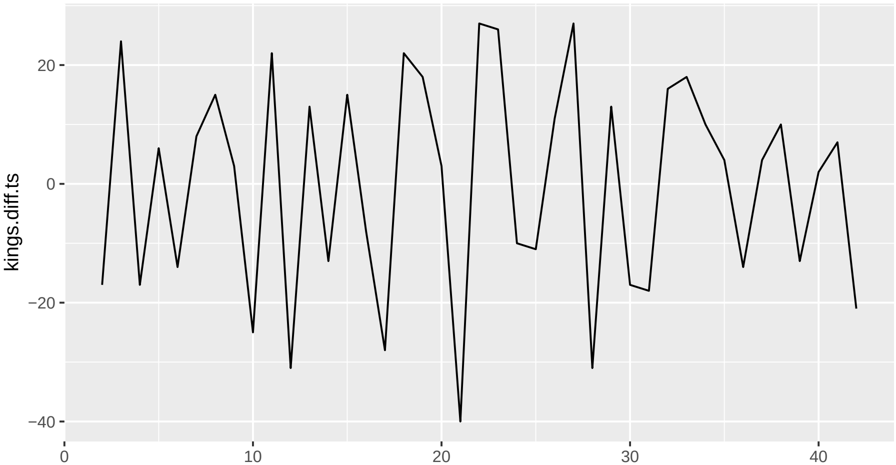
In R, `diff`:

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

# xxx Did differencing fix stationarity?

Looks stationary now: xxx

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



# xxx Births per month in New York City

from January 1946 to December 1959:

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

```
## # A tibble: 168 x 1
##       X1
##    <dbl>
##  1  26.7
##  2  23.6
##  3  26.9
##  4  24.7
##  5  25.8
##  6  24.4
##  7  24.5
##  8  23.9
##  9  23.2
## 10  23.2
## # … with 158 more rows
```

# As a time series xxx

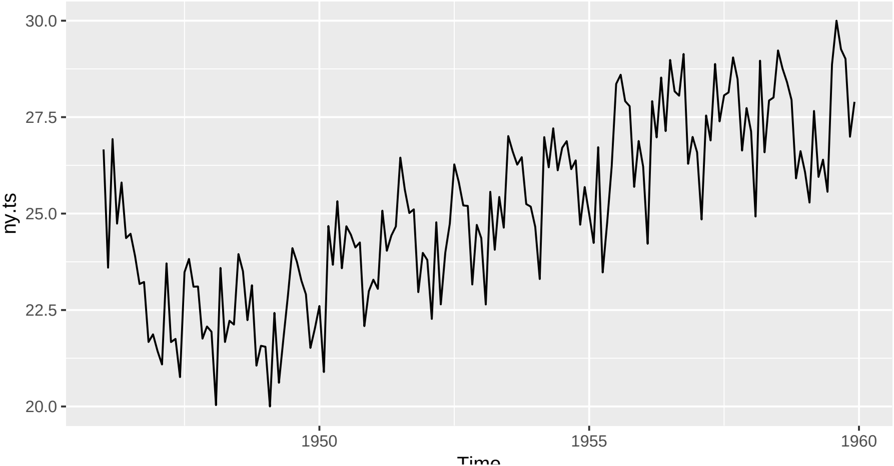
```
ny.ts=ts(ny,freq=12,start=c(1946,1))
ny.ts
```

```
##         Jan    Feb    Mar    Apr    May    Jun
## 1946 26.663 23.598 26.931 24.740 25.806 24.364
## 1947 21.439 21.089 23.709 21.669 21.752 20.761
## 1948 21.937 20.035 23.590 21.672 22.222 22.123
## 1949 21.548 20.000 22.424 20.615 21.761 22.874
## 1950 22.604 20.894 24.677 23.673 25.320 23.583
## 1951 23.287 23.049 25.076 24.037 24.430 24.667
## 1952 23.798 22.270 24.775 22.646 23.988 24.737
## 1953 24.364 22.644 25.565 24.062 25.431 24.635
## 1954 24.657 23.304 26.982 26.199 27.210 26.122
## 1955 24.990 24.239 26.721 23.475 24.767 26.219
## 1956 26.217 24.218 27.914 26.975 28.527 27.139
## 1957 26.589 24.848 27.543 26.896 28.878 27.390
```

# xxx Time plot

- Time plot shows extra pattern: xxx

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



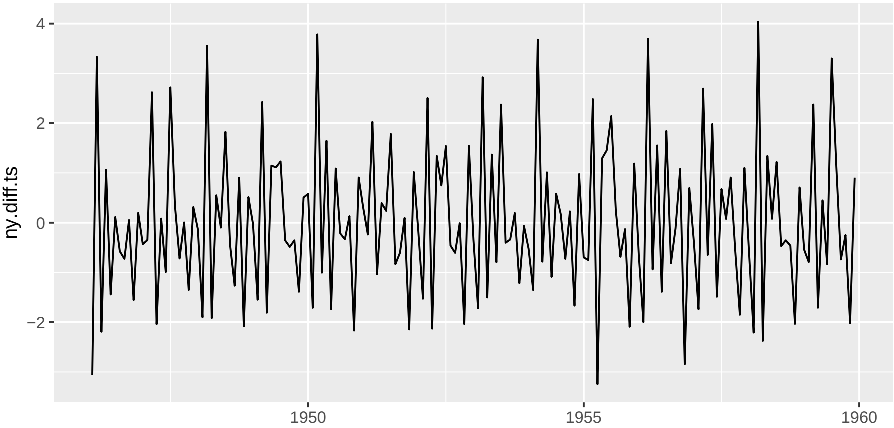
# xxx Comments on time plot

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

# xxx Differencing the New York births

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

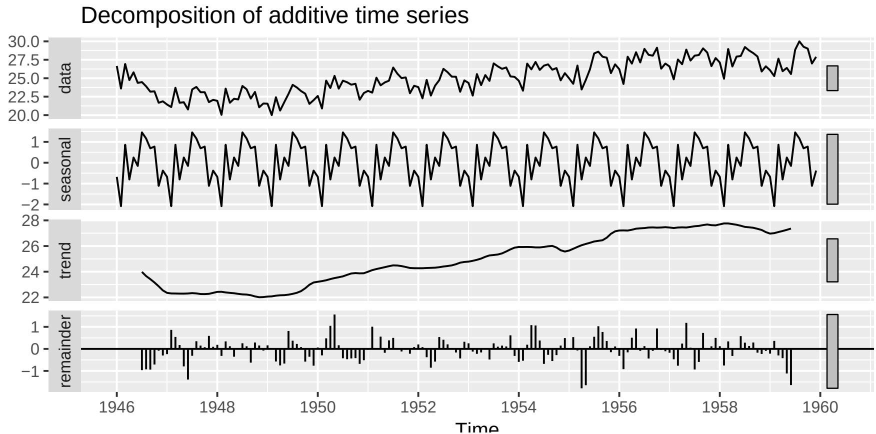
```
ny.diff.ts=diff(ny.ts)
autoplot(ny.diff.ts)
```



# xxx Decomposing a seasonal time series

A visual (using original data): xxx

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



# xxx 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")

# xxx The seasonal part

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

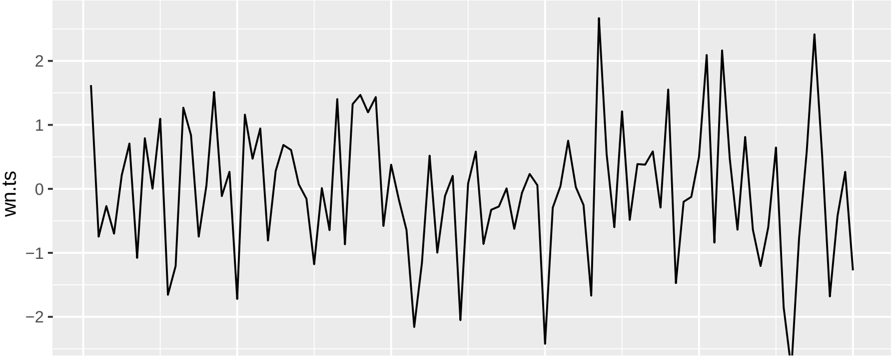
```
ny.d$seasonal
```

```
##             Jan        Feb        Mar        Apr
## 1946 -0.6771947 -2.0829607  0.8625232 -0.8016787
## 1947 -0.6771947 -2.0829607  0.8625232 -0.8016787
## 1948 -0.6771947 -2.0829607  0.8625232 -0.8016787
## 1949 -0.6771947 -2.0829607  0.8625232 -0.8016787
## 1950 -0.6771947 -2.0829607  0.8625232 -0.8016787
## 1951 -0.6771947 -2.0829607  0.8625232 -0.8016787
## 1952 -0.6771947 -2.0829607  0.8625232 -0.8016787
## 1953 -0.6771947 -2.0829607  0.8625232 -0.8016787
## 1954 -0.6771947 -2.0829607  0.8625232 -0.8016787
## 1955 -0.6771947 -2.0829607  0.8625232 -0.8016787
## 1956 -0.6771947 -2.0829607  0.8625232 -0.8016787
```

# xxx Time series basics: white noise

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

```
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:

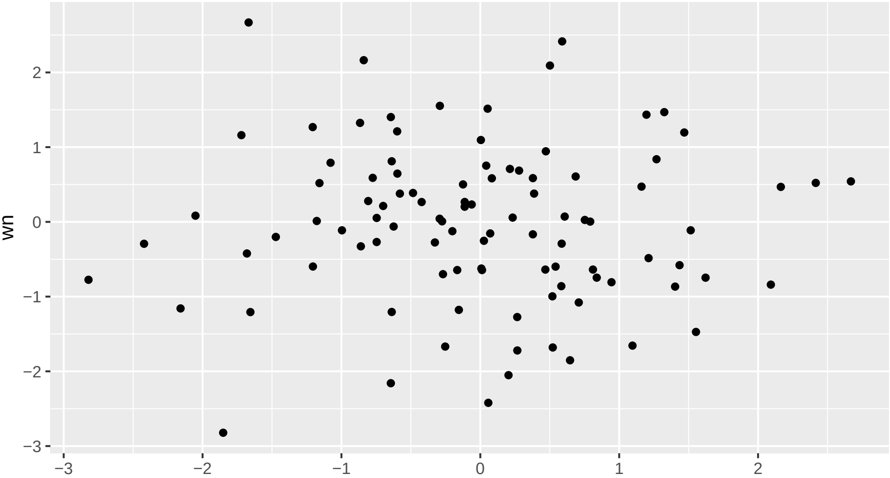
```
x=rnorm(5)
tibble(x) %>% mutate(x_lagged=lag(x)) -> with_lagged
with_lagged
```

```
## # A tibble: 5 x 2
##         x x_lagged
##     <dbl>    <dbl>
## 1 -2.04     NA
## 2 -0.579    -2.04
## 3  0.608    -0.579
## 4  0.118     0.608
## 5  0.0563    0.118
```

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

# xxx 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()
```



# xxx 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.

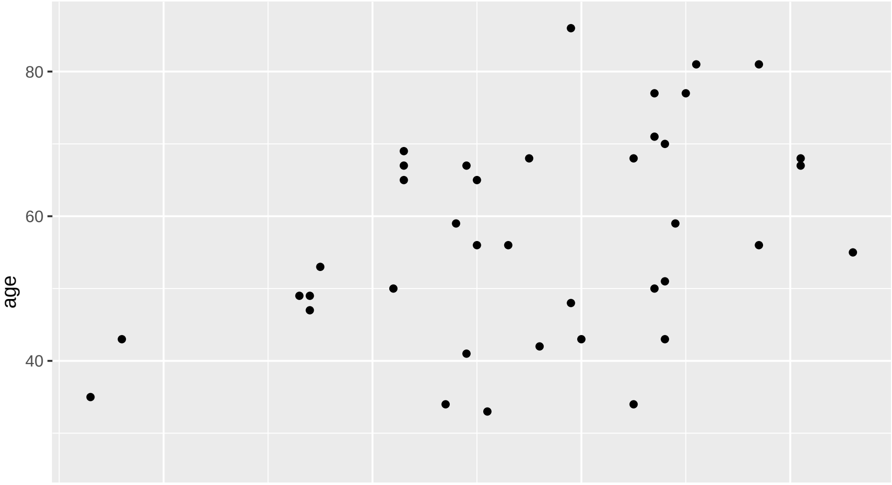
On the other hand, this:

```
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
```

# xxx Correlation with next value?

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



# xxx Two steps back:

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

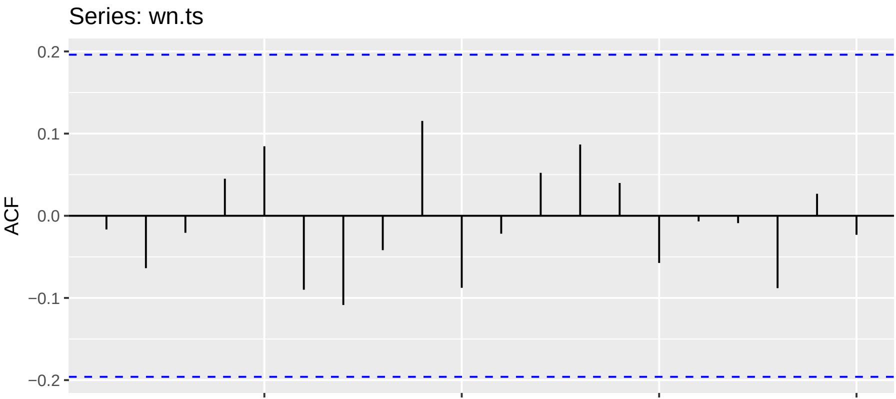
```
##
##  Pearson's product-moment correlation
##
## data:  age and age_lag_2
## t = 1.5623, df = 38, p-value = 0.1265
## alternative hypothesis: true correlation is not equal to 0
## 95 percent confidence interval:
##  -0.07128917  0.51757510
## sample estimates:
##      cor
## 0.245676
```

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

# xxx Autocorrelation

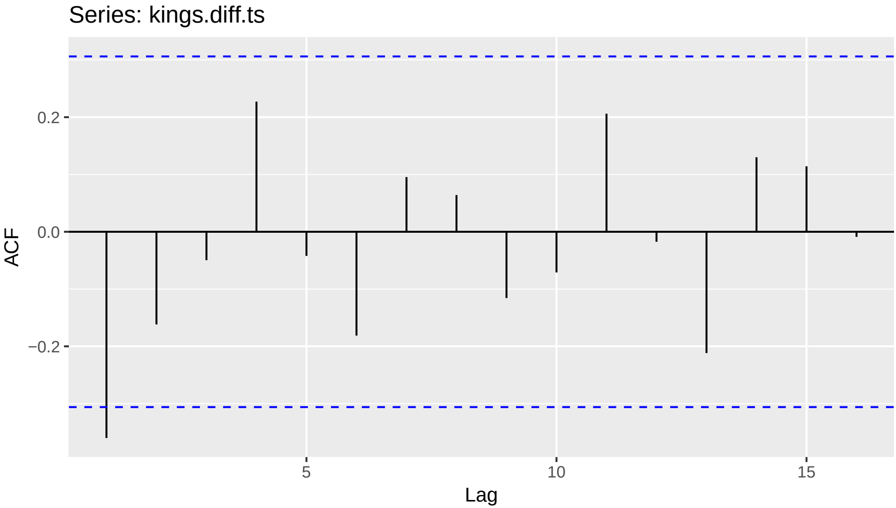
Correlation of time series with *itself* one, two,... time steps back is useful idea, called **autocorrelation**. Make a plot of it with `acf` and `autoplot`. Here, white noise: xxx

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



# Kings, differenced

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



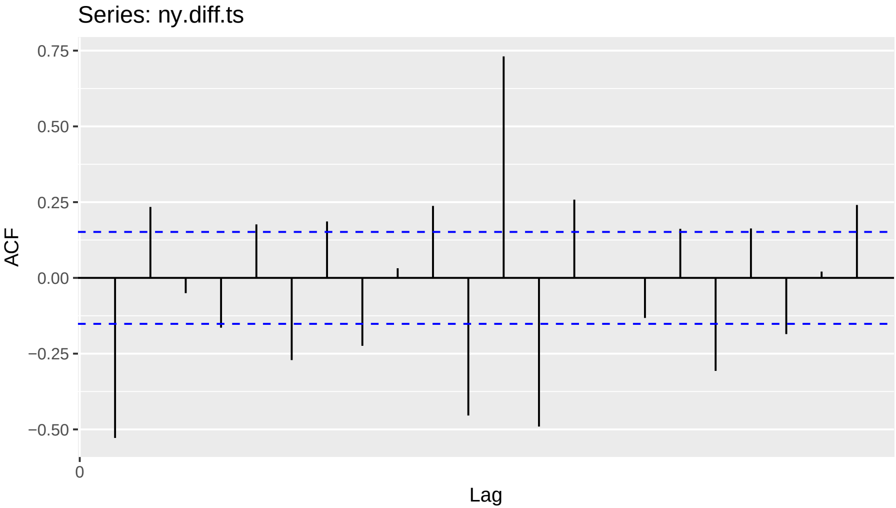
# xxx Comments on autocorrelations of kings series

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

```
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.

# xxx Souvenir sales

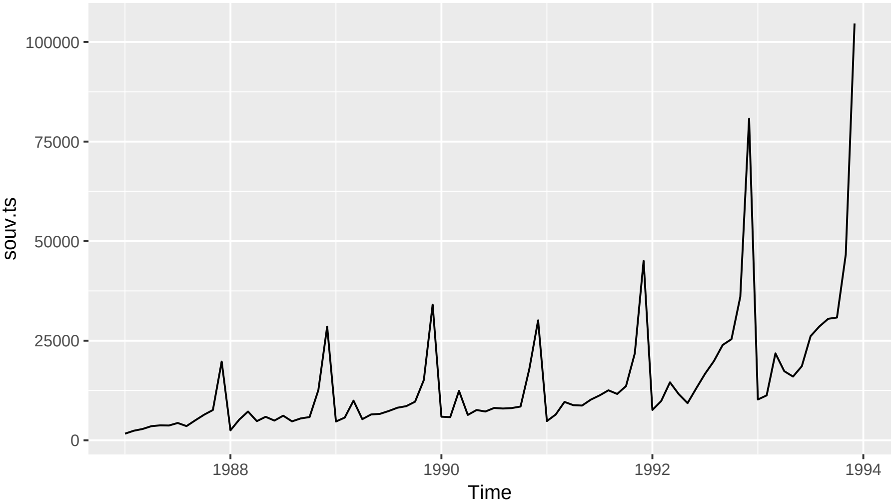
Monthly sales for a beach souvenir shop in Queensland, Australia:

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

```
##            Jan      Feb      Mar      Apr
## 1987   1664.81  2397.53  2840.71  3547.29
## 1988   2499.81  5198.24  7225.14  4806.03
## 1989   4717.02  5702.63  9957.58  5304.78
## 1990   5921.10  5814.58 12421.25  6369.77
## 1991   4826.64  6470.23  9638.77  8821.17
## 1992   7615.03  9849.69 14558.40 11587.33
## 1993  10243.24 11266.88 21826.84 17357.33
##            May      Jun      Jul      Aug
## 1987   3752.96  3714.74  4349.61  3566.34
## 1988   5900.88  4951.34  6179.12  4752.15
## 1989   6492.43  6630.80  7349.62  8176.62
## 1990   7609.12  7224.75  8121.22  7979.25
## 1991   8722.37 10209.48 11276.55 12552.22
## 1992   9332.56 13082.09 16732.78 19888.61
## 1993  15997.79 18601.53 26155.15 28586.52
##            Sep      Oct      Nov      Dec
## 1987   5021.82  6423.48  7600.60 19756.21
```

# Plot of souvenir sales

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



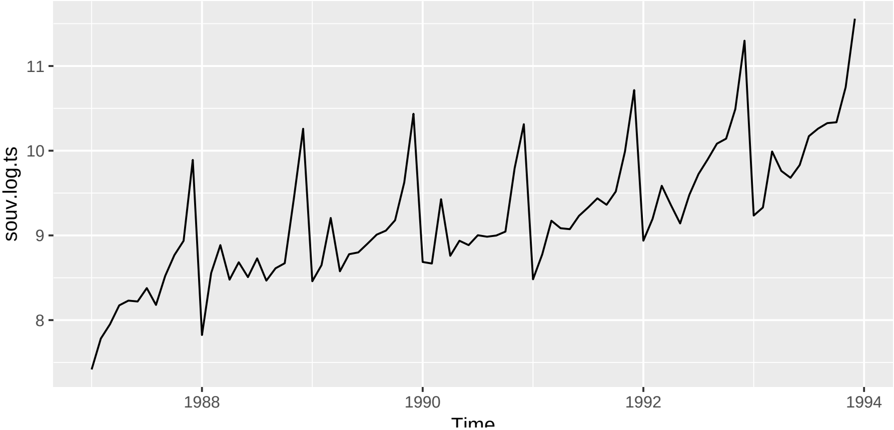
# xxx Several problems:

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

# xxx Problem-fixing:

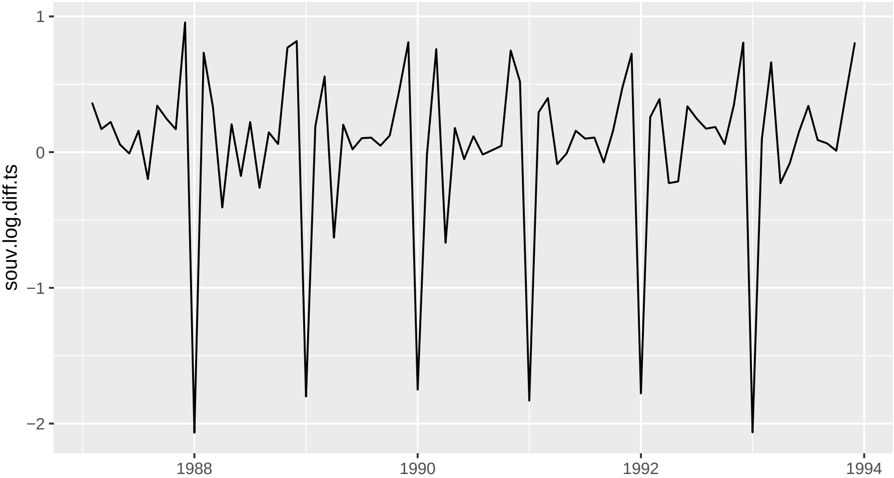
Fix non-constant variability first by taking logs: xxx

```
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)
```

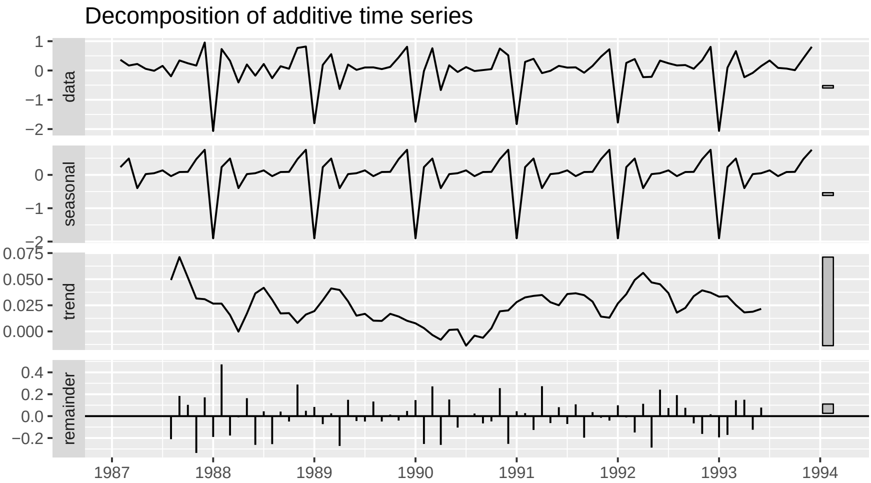


# Comments

- Now stationary
- but clear seasonal effect.

# Decomposing to see the seasonal effect

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



# xxx Comments

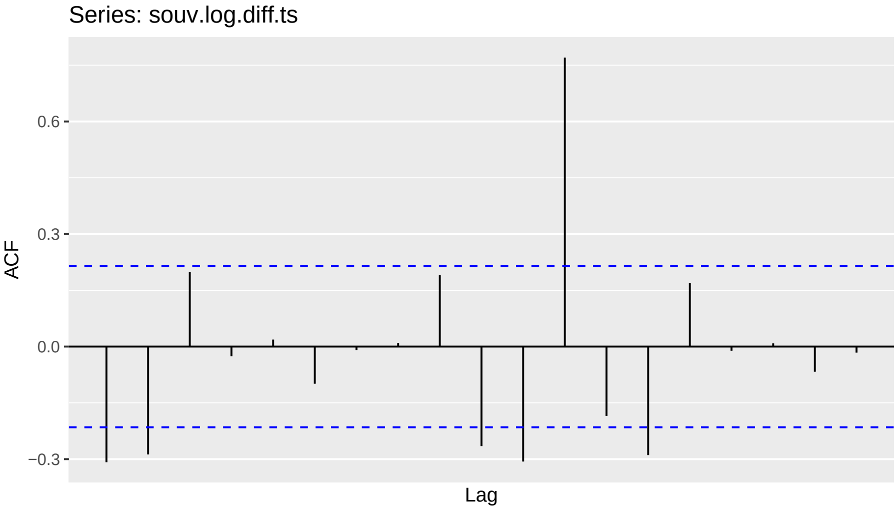
**Big** drop in one month's differences. Look at seasonal component to see which:

```
souv.d$seasonal
```

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

# Autocorrelations

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



# xxx 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 xxx

```
ma
```

```
## # A tibble: 100 x 3
##         e   e_lag      y
##     <dbl>   <dbl>  <dbl>
##  1  0.991  NA      0
##  2  0.469   0.991  1.46
##  3  0.535   0.469  1.00
##  4 -0.244   0.535  0.291
##  5  1.17   -0.244  0.928
##  6 -0.473   1.17   0.699
##  7  1.56   -0.473  1.08
##  8 -0.355   1.56   1.20
##  9 -0.400  -0.355 -0.755
## 10 -2.10   -0.400 -2.50
## # … with 90 more rows
```

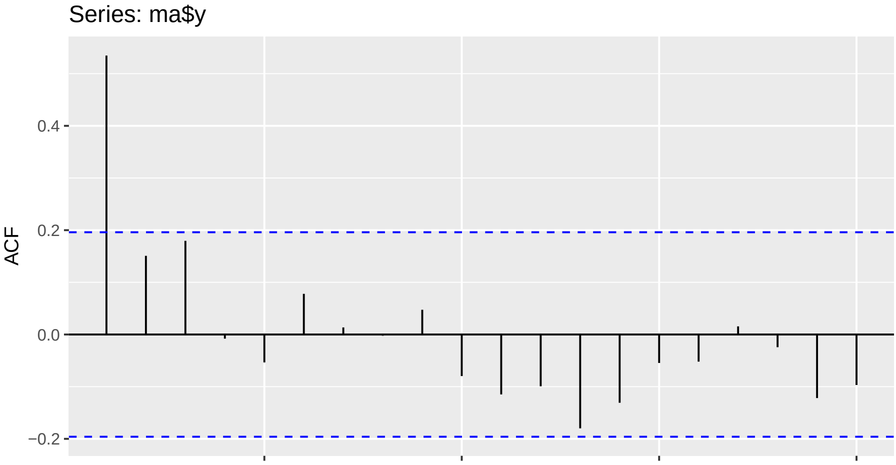
# Comments

- 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 xxx

Everything beyond lag 1 appears to be just chance:

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



# xxx 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 xxx

```
x
```

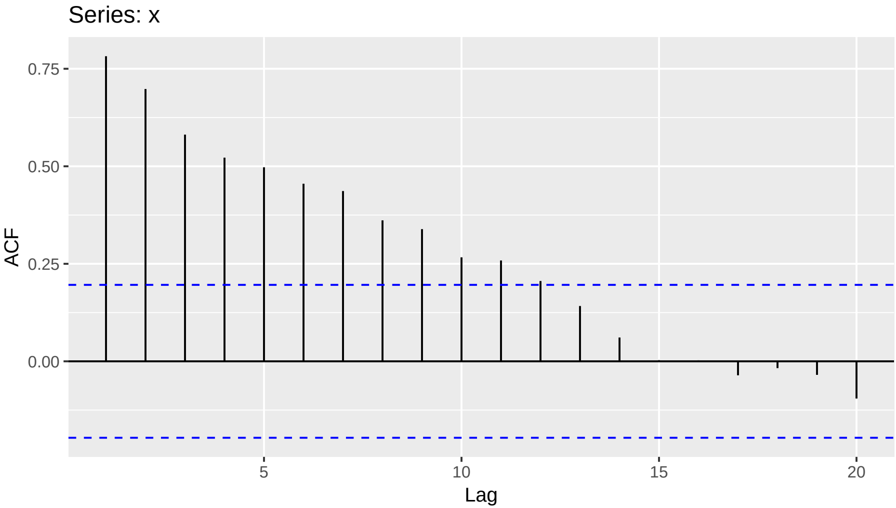
```
##   [1]  0.00000000  0.69150384 -0.27156693
##   [4] -1.69374385 -0.04624706 -0.61289729
##   [7]  0.26464756 -0.21493841 -1.31429232
##  [10]  0.44277420  0.09918044  0.19080999
##  [13] -1.02379326  0.16693770  0.98374525
##  [16]  0.04866219  1.22331904 -0.04784703
##  [19] -0.21367820 -0.68228901  0.25079396
##  [22] -0.86025292  1.75818244  1.19266409
##  [25]  0.30513461  2.41224530  1.28151011
##  [28]  1.68979182  2.01815565  3.53754507
##  [31]  1.85840920  2.32513921  1.77111656
##  [34]  2.12223993  0.91095776  1.58477201
##  [37]  2.08225425  1.09623045 -0.76369221
##  [40] -0.70809836 -1.84439667 -0.38985352
##  [43] -1.04265756 -0.86988314 -1.14485961
```

# 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

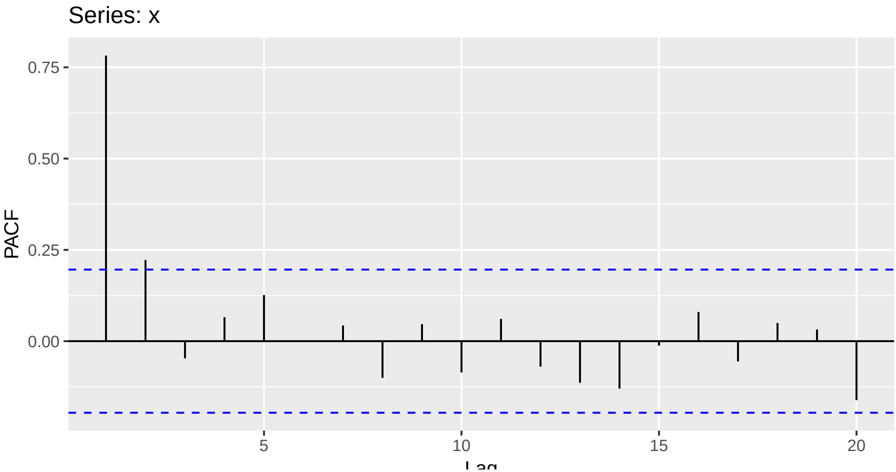
```
acf(x, plot=F) %>% autoplot()
```



# xxx Partial autocorrelation function

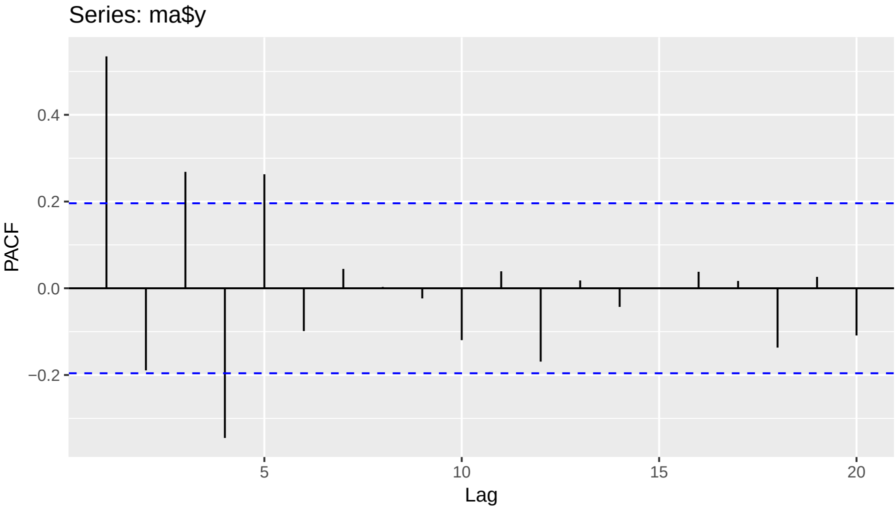
This cuts off for an AR series: xxx

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



# PACF for an MA series decays slowly

```
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.

# xxx Anatomy of auto.arima output

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

```
## Series: ma$y
## ARIMA(0,0,1) with zero mean
##
## Coefficients:
##          ma1
##       0.9070
## s.e.  0.0617
##
## sigma^2 estimated as 0.9878:  log likelihood=-141.64
## AIC=287.29   AICc=287.41   BIC=292.5
```

Comments over.

# Comments xxx

- 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)

# xxx 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 : 345.2328
##  ARIMA(1,0,0) with non-zero mean : 313.9535
##  ARIMA(0,0,1) with non-zero mean : 287.9463
##  ARIMA(0,0,0) with zero mean     : 346.0889
##  ARIMA(1,0,1) with non-zero mean : 290.112
##  ARIMA(0,0,2) with non-zero mean : 290.1128
##  ARIMA(1,0,2) with non-zero mean : 291.7865
##  ARIMA(0,0,1) with zero mean     : 287.4124
##  ARIMA(1,0,1) with zero mean     : 289.4909
##  ARIMA(0,0,2) with zero mean     : 289.4993
##  ARIMA(1,0,0) with zero mean     : 312.7625
##  ARIMA(1,0,2) with zero mean     : 290.6071
##
```

# 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).

# xxx Forecasts:

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

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

# MA(1)

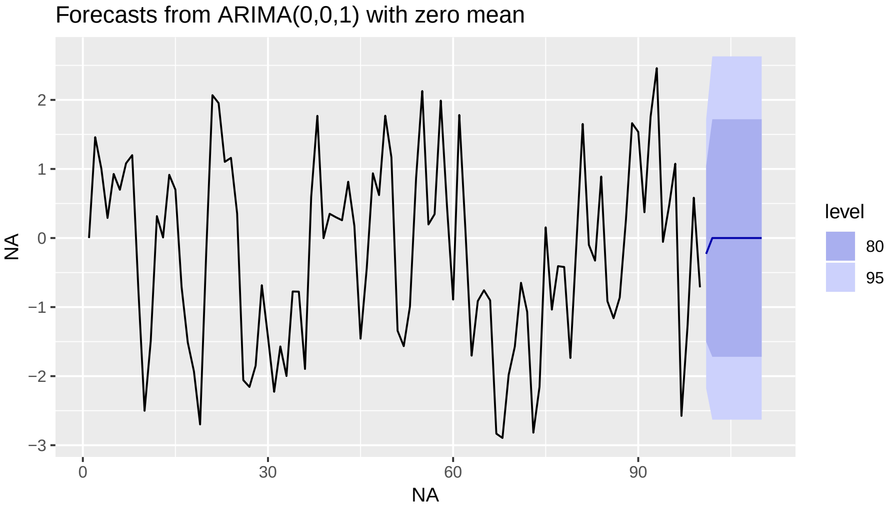
```r
y.aa=auto.arima(ma$y)
y.aa
```

```
## Series: ma$y
## ARIMA(0,0,1) with zero mean
##
## Coefficients:
##          ma1
##       0.9070
## s.e.  0.0617
##
## sigma^2 estimated as 0.9878:  log likelihood=-141.64
## AIC=287.29   AICc=287.41   BIC=292.5
```

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

# Plotting the forecasts for MA(1)

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



# xxx AR(1)

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

```
## Series: x
## ARIMA(0,1,1)
##
## Coefficients:
##           ma1
##       -0.3544
## s.e.   0.1062
##
## sigma^2 estimated as 0.979:  log likelihood=-138.99
## AIC=281.97   AICc=282.1   BIC=287.16
```

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

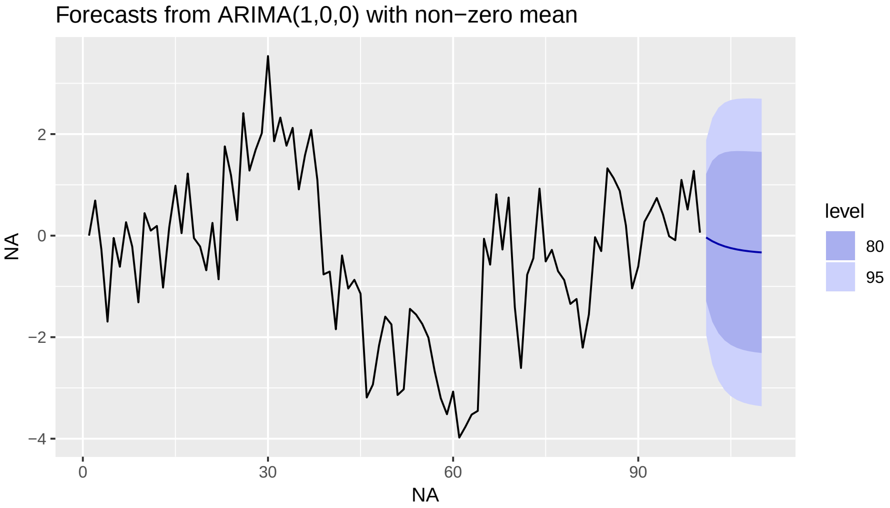
# xxx Fit right AR(1) model:

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

```
##
## Call:
## arima(x = x, order = c(1, 0, 0))
##
## Coefficients:
##          ar1  intercept
##       0.7758    -0.3646
## s.e.  0.0611     0.4220
##
## sigma^2 estimated as 0.957:  log likelihood = -140.16,  aic
```

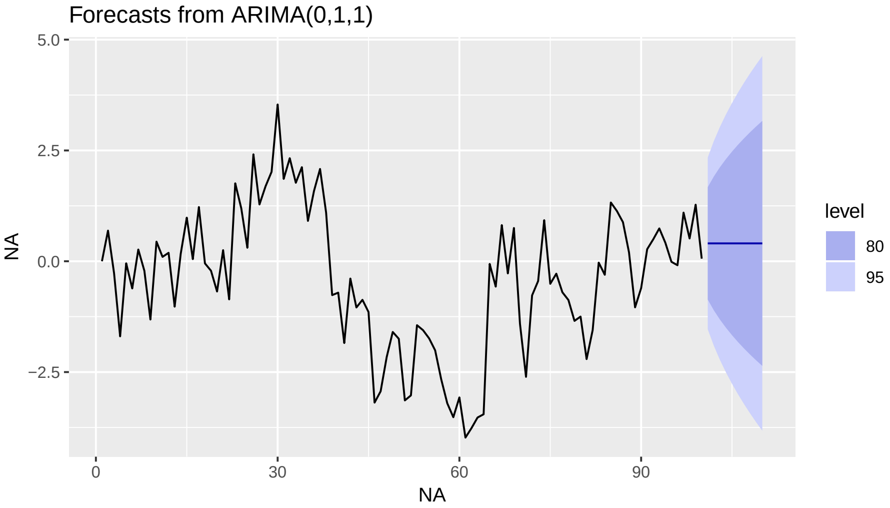
# Forecasts for x

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



# Comparing wrong model: xxx

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



# xxx 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
```

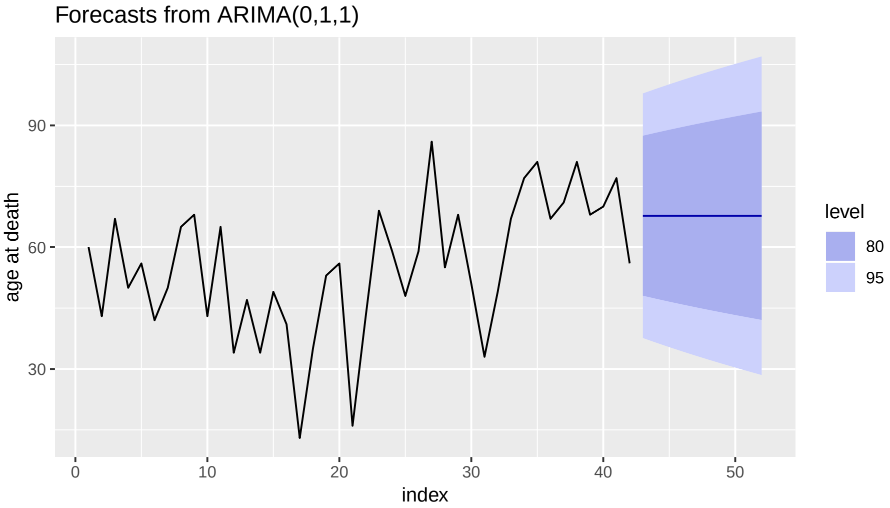
# xxx Kings forecasts:

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

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

# 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
##       0.6539  -0.4540  -0.7255  0.2532  -0.2427
## s.e.  0.3003   0.2429   0.3227  0.2878   0.0985
##          sma1
##       -0.8451
## s.e.   0.0995
##
## sigma^2 estimated as 0.4076:  log likelihood=-157.45
## AIC=328.91   AICc=329.67   BIC=350.21
```

# xxx NY births forecasts

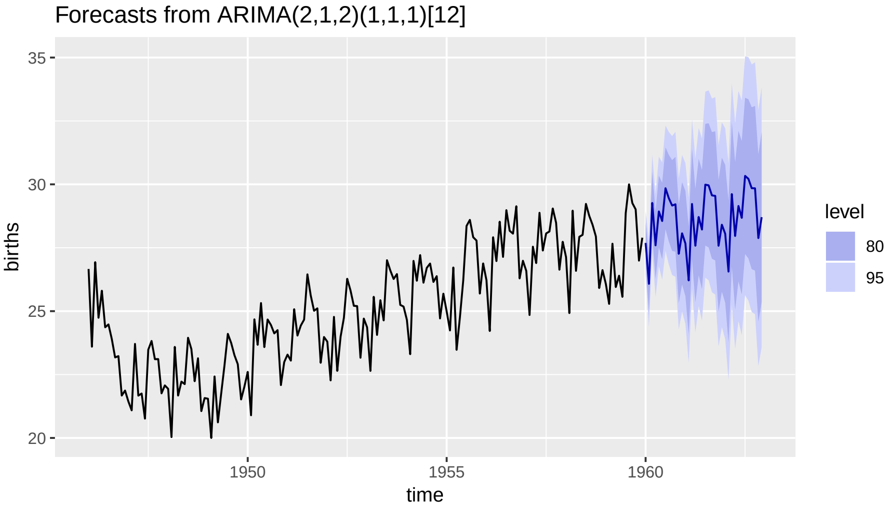
Not *quite* same every year:

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

```
##          Point Forecast    Lo 80    Hi 80    Lo 95
## Jan 1960       27.69056 26.87069 28.51043 26.43668
## Feb 1960       26.07680 24.95838 27.19522 24.36632
## Mar 1960       29.26544 28.01566 30.51523 27.35406
## Apr 1960       27.59444 26.26555 28.92333 25.56208
## May 1960       28.93193 27.52089 30.34298 26.77392
## Jun 1960       28.55379 27.04381 30.06376 26.24448
## Jul 1960       29.84713 28.23370 31.46056 27.37960
## Aug 1960       29.45347 27.74562 31.16132 26.84155
## Sep 1960       29.16388 27.37259 30.95517 26.42433
## Oct 1960       29.21343 27.34498 31.08188 26.35588
## Nov 1960       27.26221 25.31879 29.20563 24.29000
## Dec 1960       28.06863 26.05137 30.08589 24.98349
## Jan 1961       27.66908 25.59684 29.74132 24.49986
```

# 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)
```

# xxx The forecasts

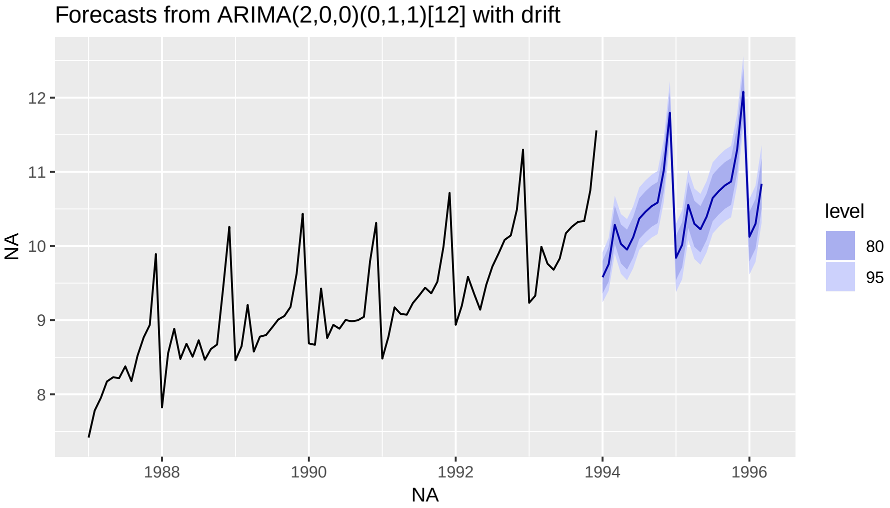
Differenced series showed low value for January (large drop). December highest, Jan and Feb lowest:

```
souv.f
```

```
##          Point Forecast      Lo 80     Hi 80
## Jan 1994       9.578291  9.358036  9.798545
## Feb 1994       9.754836  9.521700  9.987972
## Mar 1994      10.286195 10.030937 10.541453
## Apr 1994      10.028630  9.765727 10.291532
## May 1994       9.950862  9.681555 10.220168
## Jun 1994      10.116930  9.844308 10.389551
## Jul 1994      10.369140 10.094251 10.644028
## Aug 1994      10.460050 10.183827 10.736274
## Sep 1994      10.535595 10.258513 10.812677
## Oct 1994      10.585995 10.308386 10.863604
## Nov 1994      11.017734 10.739793 11.295674
## Dec 1994      11.795964 11.517817 12.074111
## Jan 1995       9.840884  9.540241 10.141527
## Feb 1995      10.015540  9.711785 10.319295
## Mar 1995      10.555070 10.246346 10.863794
## Apr 1995      10.299676  9.989043 10.610309
## May 1995      10.225535  9.913326 10.537743
```

# Plotting the forecasts

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



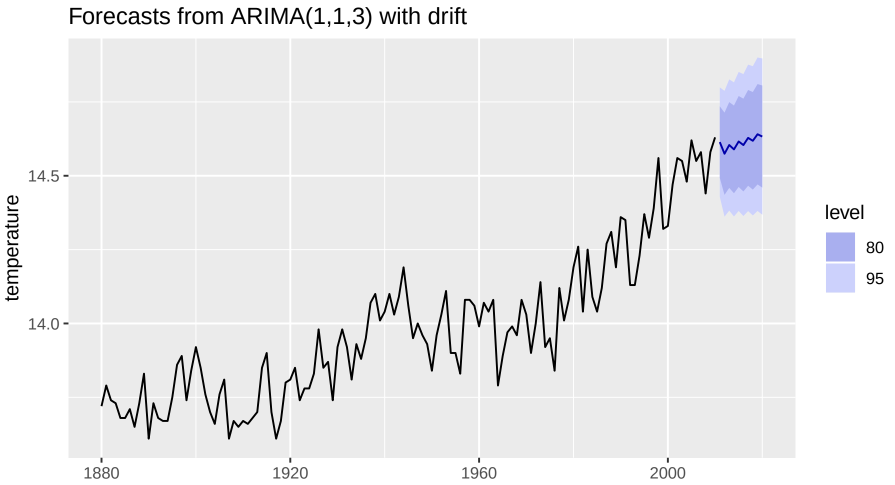
# Global mean temperatures, revisited

```
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")
```



# Section 2

# Multiway frequency tables

# Packages

```r
library(tidyverse)
```

# Multi-way frequency analysis

- A study of gender and eyewear-wearing finds the following frequencies:

| Gender | Contacts | Glasses | None |
|---|---|---|---|
| Female | 121 | 32 | 129 |
| Male | 42 | 37 | 85 |

- Is there association between eyewear and gender?
- Normally answer this with chisquare test (based on observed and expected frequencies from null hypothesis of no association).
- Two categorical variables and a frequency.
- We assess in way that generalizes to more categorical variables.

# The data file

```
gender contacts glasses none
female 121      32      129
male   42       37      85
```

- This is *not tidy!*
- Two variables are gender and *eyewear*, and those numbers all frequencies.

```
my_url <- "http://www.utsc.utoronto.ca/~butler/d29/eyewear.txt
eyewear <- read_delim(my_url, " ")
eyewear
```

```
## # A tibble: 2 x 4
##   gender contacts glasses  none
##   <chr>     <dbl>   <dbl> <dbl>
## 1 female      121      32   129
```

# Tidying the data

```
eyes <- eyewear %>%
  gather(eyewear, frequency, contacts:none)
eyes
```

```
## # A tibble: 6 x 3
##   gender eyewear  frequency
##   <chr>  <chr>        <dbl>
## 1 female contacts       121
## 2 male   contacts        42
## 3 female glasses         32
## 4 male   glasses         37
## 5 female none           129
## 6 male   none            85
```

```
xt <- xtabs(frequency ~ gender + eyewear, data = eyes)
xt
```

# Modelling

- Last table on previous page is "reconstituted" contingency table, for checking.
- Predict frequency from other factors and combos. `glm` with `poisson` family.

```
eyes.1 <- glm(frequency ~ gender * eyewear,
  data = eyes,
  family = "poisson"
)
```

def

- Called **log-linear model**.

# What can we get rid of?

{

```
drop1(eyes.1, test = "Chisq")
```

```
## Single term deletions
##
## Model:
## frequency ~ gender * eyewear
##                Df Deviance    AIC    LRT  Pr(>Chi)
## <none>                0.000 47.958
## gender:eyewear  2   17.829 61.787 17.829 0.0001345
##
## <none>
## gender:eyewear ***
## ---
## Signif. codes:
## 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

def }

# Conclusions

- `drop1` says what we can remove at this step. Significant = must stay.
- Cannot remove anything.
- Frequency depends on `gender-wear` *combination*, cannot be simplified further.
- Gender and eyewear are *associated*.
- Stop here.

# prop.table

Original table:

{

```
xt
```

```
##          eyewear
## gender   contacts glasses none
##   female      121      32  129
##   male         42      37   85
```

} Calculate eg. row proportions like this:

{

```
prop.table(xt, margin = 1)
```

```
##          eyewear
## gender     contacts   glasses      none
##   female 0.4290780 0.1134752 0.4574468
##   male   0.2560976 0.2256098 0.5182927
```

# No association

- Suppose table had been as shown below:

```
my_url <- "http://www.utsc.utoronto.ca/~butler/d29/eyewear2.tx
eyewear2 <- read_table(my_url)
eyes2 <- eyewear2 %>% gather(eyewear, frequency, contacts:none
xt2 <- xtabs(frequency ~ gender + eyewear, data = eyes2)
xt2
```

```
##         eyewear
## gender   contacts glasses none
##   female      150      30  120
##   male         75      16   62
```

```
prop.table(xt2, margin = 1)
```

```
##         eyewear
## gender    contacts   glasses      none
##   female 0.5000000 0.1000000 0.4000000
```

# Analysis for revised data

```
eyes.2 <- glm(frequency ~ gender * eyewear,
  data = eyes2,
  family = "poisson"
)
drop1(eyes.2, test = "Chisq")
```

```
## Single term deletions
##
## Model:
## frequency ~ gender * eyewear
##                Df Deviance    AIC      LRT Pr(>Chi)
## <none>            0.000000 47.467
## gender:eyewear  2 0.047323 43.515 0.047323   0.9766
```

No longer any association. Take out interaction.

# No interaction

{

```
eyes.3 <- update(eyes.2, . ~ . - gender:eyewear)
drop1(eyes.3, test = "Chisq")
```

```
## Single term deletions
##
## Model:
## frequency ~ gender + eyewear
##         Df Deviance     AIC     LRT  Pr(>Chi)
## <none>        0.047  43.515
## gender   1   48.624  90.091  48.577 3.176e-12 ***
## eyewear  2  138.130 177.598 138.083 < 2.2e-16 ***
## ---
## Signif. codes:
## 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

}

- More females (gender effect)

# Chest pain, being overweight and being a smoker

- In a hospital emergency department, 176 subjects who attended for acute chest pain took part in a study.
- Each subject had a normal or abnormal electrocardiogram reading (ECG), were overweight (as judged by BMI) or not, and were a smoker or not.
- How are these three variables related, or not?

# The data

In modelling-friendly format:

```
ecg bmi smoke count
abnormal overweight yes 47
abnormal overweight no 10
abnormal normalweight yes 8
abnormal normalweight no 6
normal overweight yes 25
normal overweight no 15
normal normalweight yes 35
normal normalweight no 30
```

# First step

```
my_url <- "http://www.utsc.utoronto.ca/~butler/d29/ecg.txt"
chest <- read_delim(my_url, " ")
chest.1 <- glm(count ~ ecg * bmi * smoke,
  data = chest,
  family = "poisson"
)
drop1(chest.1, test = "Chisq")
```

```
## Single term deletions
##
## Model:
## count ~ ecg * bmi * smoke
##               Df Deviance    AIC    LRT Pr(>Chi)
## <none>             0.0000 53.707
## ecg:bmi:smoke  1   1.3885 53.096 1.3885   0.2387
```

That 3-way interaction comes out.

# Removing the 3-way interaction

```
chest.2 <- update(chest.1, . ~ . - ecg:bmi:smoke)
drop1(chest.2, test = "Chisq")
```

```
## Single term deletions
##
## Model:
## count ~ ecg + bmi + smoke + ecg:bmi + ecg:smoke + bmi:smoke
##           Df Deviance    AIC     LRT  Pr(>Chi)
## <none>         1.3885 53.096
## ecg:bmi    1  29.0195 78.727 27.6310 1.468e-07 ***
## ecg:smoke  1   4.8935 54.601  3.5050   0.06119 .
## bmi:smoke  1   4.4689 54.176  3.0803   0.07924 .
## ---
## Signif. codes:
## 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

At $\alpha = 0.05$, `bmi:smoke` comes out.

# Removing bmi:smoke

```
chest.3 <- update(chest.2, . ~ . - bmi:smoke)
drop1(chest.3, test = "Chisq")
```

```
## Single term deletions
##
## Model:
## count ~ ecg + bmi + smoke + ecg:bmi + ecg:smoke
##           Df Deviance    AIC    LRT  Pr(>Chi)
## <none>          4.469 54.176
## ecg:bmi    1   36.562 84.270 32.094 1.469e-08 ***
## ecg:smoke  1   12.436 60.144  7.968  0.004762 **
## ---
## Signif. codes:
## 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

`ecg:smoke` has become significant. So we have to stop.

# Understanding the final model

- Thinking of `ecg` as "response" that might depend on anything else.
- What is associated with `ecg`? Both `bmi` on its own and `smoke` on its own, but *not* the combination of both.
- `ecg:bmi` table:

```
xtabs(count ~ ecg + bmi, data = chest)
```

```
##              bmi
## ecg           normalweight overweight
##   abnormal              14         57
##   normal                65         40
```

- Most normal weight people have a normal ECG, but a majority of overweight people have an *abnormal* ECG. That is, knowing about BMI says something about likely ECG.

# ecg:smoke

- ecg:smoke table:

```
xtabs(count ~ ecg + smoke, data = chest)
```

```
##           smoke
## ecg        no yes
##   abnormal 16  55
##   normal   45  60
```

- Most nonsmokers have a normal ECG, but smokers are about 50–50 normal and abnormal ECG.
- Don't look at smoke:bmi table since not significant.

# Simpson's paradox: the airlines example

| Airport | Alaska Airlines On time | Alaska Airlines Delayed | America West On time | America West Delayed |
|---|---|---|---|---|
| Los Angeles | 497 | 62 | 694 | 117 |
| Phoenix | 221 | 12 | 4840 | 415 |
| San Diego | 212 | 20 | 383 | 65 |
| San Francisco | 503 | 102 | 320 | 129 |
| Seattle | 1841 | 305 | 201 | 61 |
| Total | 3274 | 501 | 6438 | 787 |

Use `status` as variable name for "on time/delayed".

- Alaska: 13.3% flights delayed $(501/(3274+501))$.
- America West: 10.9% $(787/(6438+787))$.
- America West more punctual, right?

# Arranging the data

- Can only have single thing in columns, so we have to construct column names like this: \begin{small}

```
airport     aa_ontime aa_delayed aw_ontime aw_delayed
LosAngeles    497          62        694        117
Phoenix       221          12       4840        415
SanDiego      212          20        383         65
SanFrancisco 503          102        320        129
Seattle      1841         305        201         61
```

\end{small}

- Some tidying gets us the right layout, with frequencies all in one column and the airline and delayed/on time status separated out:

```
my_url <- "http://www.utsc.utoronto.ca/~butler/d29/airlines.tx
airlines <- read_table2(my_url)
```

# The data frame punctual

```
## # A tibble: 20 x 4
##    airport       airline status   freq
##    <chr>         <chr>   <chr>   <dbl>
##  1 LosAngeles    aa      ontime    497
##  2 Phoenix       aa      ontime    221
##  3 SanDiego      aa      ontime    212
##  4 SanFrancisco  aa      ontime    503
##  5 Seattle       aa      ontime   1841
##  6 LosAngeles    aa      delayed    62
##  7 Phoenix       aa      delayed    12
##  8 SanDiego      aa      delayed    20
##  9 SanFrancisco  aa      delayed   102
## 10 Seattle       aa      delayed   305
## 11 LosAngeles    aw      ontime    694
## 12 Phoenix       aw      ontime   4840
## 13 SanDiego      aw      ontime    383
```

# Proportions delayed by airline

- Two-step process: get appropriate subtable:

```
xt <- xtabs(freq ~ airline + status, data = punctual)
xt
```

```
##          status
## airline delayed ontime
##      aa     501   3274
##      aw     787   6438
```

- and then calculate appropriate proportions:

```
prop.table(xt, margin = 1)
```

```
##          status
## airline   delayed    ontime
##      aa 0.1327152 0.8672848
##      aw 0.1089273 0.8910727
```

# Proportion delayed by airport, for each airline

```
xt <- xtabs(freq ~ airline + status + airport, data = punctual
xp <- prop.table(xt, margin = c(1, 3))
ftable(xp,
  row.vars = c("airport", "airline"),
  col.vars = "status"
)
```

```
##                         status    delayed     ontime
## airport       airline
## LosAngeles    aa                0.11091234 0.88908766
##               aw                0.14426634 0.85573366
## Phoenix       aa                0.05150215 0.94849785
##               aw                0.07897241 0.92102759
## SanDiego      aa                0.08620690 0.91379310
##               aw                0.14508929 0.85491071
## SanFrancisco  aa                0.16859504 0.83140496
##               aw                0.28730512 0.71269488
```

# Simpson's Paradox

| Airport | Alaska | America West |
|---|---|---|
| Los Angeles | 11.4 | 14.4 |
| Phoenix | 5.2 | 7.9 |
| San Diego | 8.6 | 14.5 |
| San Francisco | 16.9 | 28.7 |
| Seattle | 14.2 | 23.2 |
| Total | 13.3 | 10.9 |

- America West more punctual overall,
- but worse at *every single* airport!
- How is that possible?
- Log-linear analysis sheds some light.

# Model 1 and output

```
punctual.1 <- glm(freq ~ airport * airline * status,
  data = punctual, family = "poisson"
)
drop1(punctual.1, test = "Chisq")
```

```
## Single term deletions
##
## Model:
## freq ~ airport * airline * status
##                         Df Deviance    AIC    LRT
## <none>                       0.0000 183.44
## airport:airline:status   4   3.2166 178.65 3.2166
##                         Pr(>Chi)
## <none>
## airport:airline:status    0.5223
```

def

# Remove 3-way interaction

```
punctual.2 <- update(punctual.1, ~ . - airport:airline:status)
drop1(punctual.2, test = "Chisq")
```

```
## Single term deletions
##
## Model:
## freq ~ airport + airline + status + airport:airline + airpo
##     airline:status
##                 Df Deviance    AIC    LRT  Pr(>Chi)
## <none>                  3.2  178.7
## airport:airline  4   6432.5 6599.9 6429.2 < 2.2e-16
## airport:status   4    240.1  407.5  236.9 < 2.2e-16
## airline:status   1     45.5  218.9   42.2 8.038e-11
##
## <none>
## airport:airline ***
```

# Understanding the significance

- `airline:status`:

```
xt <- xtabs(freq ~ airline + status, data = punctual)
prop.table(xt, margin = 1)
```

```
##          status
## airline    delayed    ontime
##       aa 0.1327152 0.8672848
##       aw 0.1089273 0.8910727
```

- More of Alaska Airlines' flights delayed overall.
- Saw this before.

# Understanding the significance (2)

- `airport:status`:

```
xt <- xtabs(freq ~ airport + status, data = punctual)
prop.table(xt, margin = 1)
```

```
##                 status
## airport            delayed     ontime
##   LosAngeles    0.13065693 0.86934307
##   Phoenix       0.07780612 0.92219388
##   SanDiego      0.12500000 0.87500000
##   SanFrancisco 0.21916509 0.78083491
##   Seattle       0.15199336 0.84800664
```

- Flights into San Francisco (and maybe Seattle) are often late, and flights into Phoenix are usually on time.
- Considerable variation among airports.

# Understanding the significance (3)

- `airport:airline`:

```
xt <- xtabs(freq ~ airport + airline, data = punctual)
prop.table(xt, margin = 2)
```

```
##                airline
## airport                 aa         aw
##   LosAngeles    0.14807947 0.11224913
##   Phoenix       0.06172185 0.72733564
##   SanDiego      0.06145695 0.06200692
##   SanFrancisco  0.16026490 0.06214533
##   Seattle       0.56847682 0.03626298
```

- What fraction of each airline's flights are to each airport.
- Most of Alaska Airlines' flights to Seattle and San Francisco.
- Most of America West's flights to Phoenix.

# The resolution

- Most of America West's flights to Phoenix, where it is easy to be on time.
- Most of Alaska Airlines' flights to San Francisco and Seattle, where it is difficult to be on time.
- Overall comparison looks bad for Alaska because of this.
- But, *comparing like with like*, if you compare each airline's performance *to the same airport*, Alaska does better.
- Aggregating over the very different airports was a (big) mistake: that was the cause of the Simpson's paradox.
- Alaska Airlines is *more* punctual when you do the proper comparison.

# Ovarian cancer: a four-way table

- Retrospective study of ovarian cancer done in 1973.
- Information about 299 women operated on for ovarian cancer 10 years previously.
- Recorded:
- stage of cancer (early or advanced)
- type of operation (radical or limited)
- X-ray treatment received (yes or no)
- 10-year survival (yes or no)
- Survival looks like response (suggests logistic regression).
- Log-linear model finds any associations at all.

# The data

after tidying:

{

```
stage operation xray survival freq
early radical no no 10
early radical no yes 41
early radical yes no 17
early radical yes yes 64
early limited no no 1
early limited no yes 13
early limited yes no 3
early limited yes yes 9
advanced radical no no 38
advanced radical no yes 6
advanced radical yes no 64
advanced radical yes yes 11
advanced limited no no 3
advanced limited no yes 1
advanced limited yes no 13
advanced limited yes yes 5
```

# Stage 1

hopefully looking familiar by now:

```
my_url <- "http://www.utsc.utoronto.ca/~butler/d29/cancer.txt"
cancer <- read_delim(my_url, " ")
cancer %>% print(n = 6)
```

```
## # A tibble: 16 x 5
##   stage operation xray  survival  freq
##   <chr> <chr>     <chr> <chr>    <dbl>
## 1 early radical   no    no          10
## 2 early radical   no    yes         41
## 3 early radical   yes   no          17
## 4 early radical   yes   yes         64
## 5 early limited   no    no           1
## 6 early limited   no    yes         13
## # … with 10 more rows
```

```
cancer.1 <- glm(freq ~ stage * operation * xray * survival
```

# Output 1

See what we can remove:

```
drop1(cancer.1, test = "Chisq")
```

```
## Single term deletions
##
## Model:
## freq ~ stage * operation * xray * survival
##                               Df Deviance    AIC
## <none>                             0.00000 98.130
## stage:operation:xray:survival  1   0.60266 96.732
##                                    LRT Pr(>Chi)
## <none>
## stage:operation:xray:survival 0.60266   0.4376
```

def

Non-significant interaction can come out.

# Stage 2

```r
cancer.2 <- update(cancer.1, ~ .
- stage:operation:xray:survival)
drop1(cancer.2, test = "Chisq")
```

```
## Single term deletions
##
## Model:
## freq ~ stage + operation + xray + survival + stage:operatio
##     stage:xray + operation:xray + stage:survival + operatio
##     xray:survival + stage:operation:xray + stage:operation:
##     stage:xray:survival + operation:xray:survival
##                           Df Deviance    AIC     LRT
## <none>                      0.60266 96.732
## stage:operation:xray       1  2.35759 96.487 1.75493
## stage:operation:survival   1  1.17730 95.307 0.57465
## stage:xray:survival        1  0.95577 95.085 0.35311
```

# Take out stage:xray:survival

```
cancer.3 <- update(cancer.2, . ~ . - stage:xray:survival)
drop1(cancer.3, test = "Chisq")
```

```
## Single term deletions
##
## Model:
## freq ~ stage + operation + xray + survival + stage:operatio
##     stage:xray + operation:xray + stage:survival + operatio
##     xray:survival + stage:operation:xray + stage:operation:
##     operation:xray:survival
##                          Df Deviance    AIC     LRT
## <none>                       0.95577 95.085
## stage:operation:xray      1  3.08666 95.216 2.13089
## stage:operation:survival  1  1.56605 93.696 0.61029
## operation:xray:survival   1  1.55124 93.681 0.59547
##                          Pr(>Chi)
```

# Remove operation:xray:survival

```
cancer.4 <- update(cancer.3, . ~ . - operation:xray:survival)
drop1(cancer.4, test = "Chisq")
```

```
## Single term deletions
##
## Model:
## freq ~ stage + operation + xray + survival + stage:operatio
##     stage:xray + operation:xray + stage:survival + operatio
##     xray:survival + stage:operation:xray + stage:operation:
##                          Df Deviance    AIC    LRT
## <none>                        1.5512 93.681
## xray:survival             1   1.6977 91.827 0.1464
## stage:operation:xray      1   6.8420 96.972 5.2907
## stage:operation:survival  1   1.9311 92.061 0.3799
##                          Pr(>Chi)
## <none>
```

# Comments

- `stage:operation:xray` has now become significant, so won't remove that.
- Shows value of removing terms one at a time.
- There are no higher-order interactions containing both `xray` and `survival`, so now we get to test (and remove) `xray:survival`.

# Remove xray:survival

```
cancer.5 <- update(cancer.4, . ~ . - xray:survival)
drop1(cancer.5, test = "Chisq")
```

```
## Single term deletions
##
## Model:
## freq ~ stage + operation + xray + survival + stage:operatio
##     stage:xray + operation:xray + stage:survival + operatio
##     stage:operation:xray + stage:operation:survival
##                          Df Deviance    AIC    LRT
## <none>                         1.6977 91.827
## stage:operation:xray      1   6.9277 95.057 5.2300
## stage:operation:survival  1   2.0242 90.154 0.3265
##                          Pr(>Chi)
## <none>
## stage:operation:xray       0.0222 *
## stage:operation:survival   0.5677
```

# Remove stage:operation:survival

```
cancer.6 <- update(cancer.5, . ~ . - stage:operation:survival)
drop1(cancer.6, test = "Chisq")
```

```
## Single term deletions
##
## Model:
## freq ~ stage + operation + xray + survival + stage:operatio
##     stage:xray + operation:xray + stage:survival + operatio
##     stage:operation:xray
##                      Df Deviance     AIC     LRT
## <none>                     2.024  90.154
## stage:survival        1  135.198 221.327 133.173
## operation:survival    1    4.116  90.245   2.092
## stage:operation:xray  1    7.254  93.384   5.230
##                      Pr(>Chi)
## <none>
```

# Last step?

Remove `operation:survival`.

```
cancer.7 <- update(cancer.6, . ~ . - operation:survival)
drop1(cancer.7, test = "Chisq")
```

```
## Single term deletions
##
## Model:
## freq ~ stage + operation + xray + survival + stage:operatic
##     stage:xray + operation:xray + stage:survival + stage:op
##                       Df Deviance     AIC    LRT
## <none>                     4.116  90.245
## stage:survival         1  136.729 220.859 132.61
## stage:operation:xray   1    9.346  93.475   5.23
##                       Pr(>Chi)
## <none>
## stage:survival          <2e-16 ***
```

# Conclusions

- What matters is things associated with survival (survival is “response”).
- Only significant such term is `stage:survival`:

```
xt <- xtabs(freq ~ stage + survival, data = cancer)
prop.table(xt, margin = 1)
```

```
##           survival
## stage             no       yes
##   advanced 0.8368794 0.1631206
##   early    0.1962025 0.8037975
```

- Most people in early stage of cancer survived, and most people in advanced stage did not survive.
- This true *regardless* of type of operation or whether or not X-ray treatment was received. These things have no impact on survival.

# What about that other interaction?

```r
xt <- xtabs(freq ~ operation + xray + stage, data = cancer)
ftable(prop.table(xt, margin = 3))
```

```
##                    stage   advanced      early
## operation xray
## limited   no             0.02836879 0.08860759
##           yes            0.12765957 0.07594937
## radical   no             0.31205674 0.32278481
##           yes            0.53191489 0.51265823
```

- Out of the people at each stage of cancer (since `margin=3` and `stage` was listed 3rd).
- The association is between `stage` and `xray` *only for those who had the limited operation*.
- For those who had the radical operation, there was no association between `stage` and `xray`.

# General procedure

- Start with "complete model" including all possible interactions.
- `drop1` gives highest-order interaction(s) remaining, remove least non-significant.
- Repeat as necessary until everything significant.
- Look at subtables of significant interactions.
- Main effects not usually very interesting.
- Interactions with "response" usually of most interest: show association with response.

```
## Error in FUN(X[[i]], ...): invalid 'name' argument
```