TimeSeries

Contents

```
library(forecast)
library(chron)
library(lubridate)
library(lattice)
library(ggplot2)
library(readr)
library(zoo)
library(tidyr)
library(tseries)
library(lmtest)
library("TTR")
```

1 Time Series Introduction

1.1 Data Preparation

ITstore_bidaily has been uploaded; data contains time stamp in column 1 (every unit represents 5 hours which are either in the morning or in the afternoon; a week has 6 days)

```
library(readr)
ITstore_bidaily <- read_delim("~/Downloads/ITstore-bidaily.csv",
    ";", escape_double = FALSE, col_names = FALSE,
    trim_ws = TRUE)
```

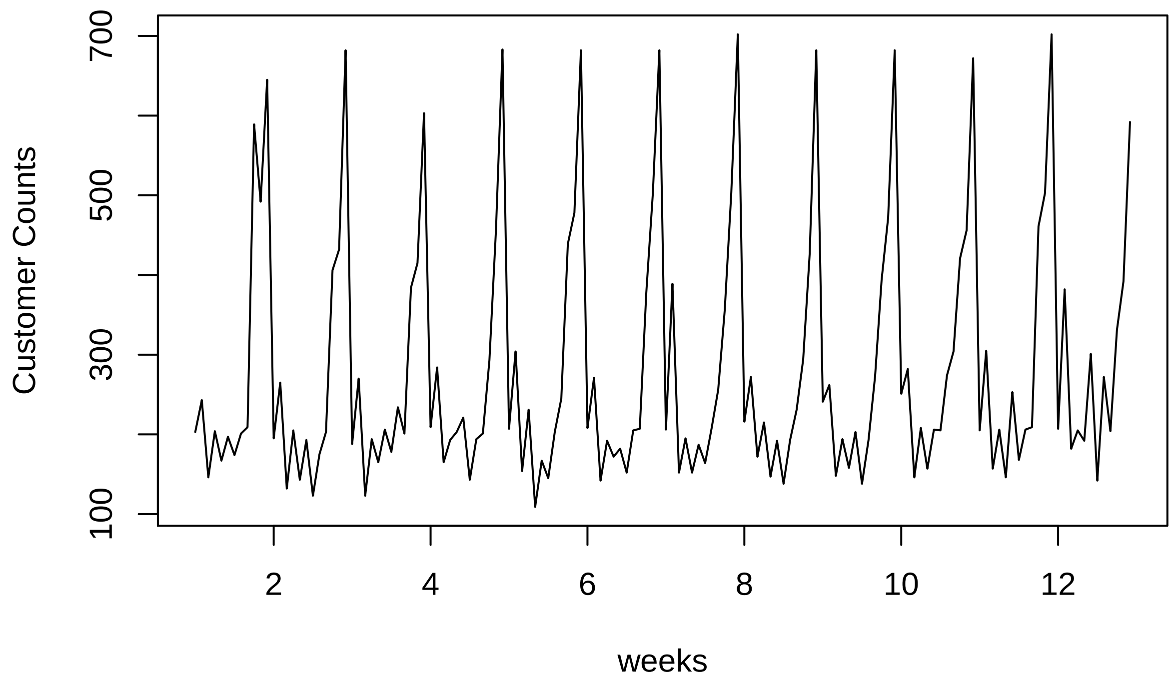
```
myts <- ITstore_bidaily
```

Data will be transfered to time series object thanks to ts() function. Start represents the starting point for the time stamp. Frequency is number of observations per cycle => 1 week contains 12 observation.

```
mycounts <- ts(myts$X2, start=1, frequency = 12)
```

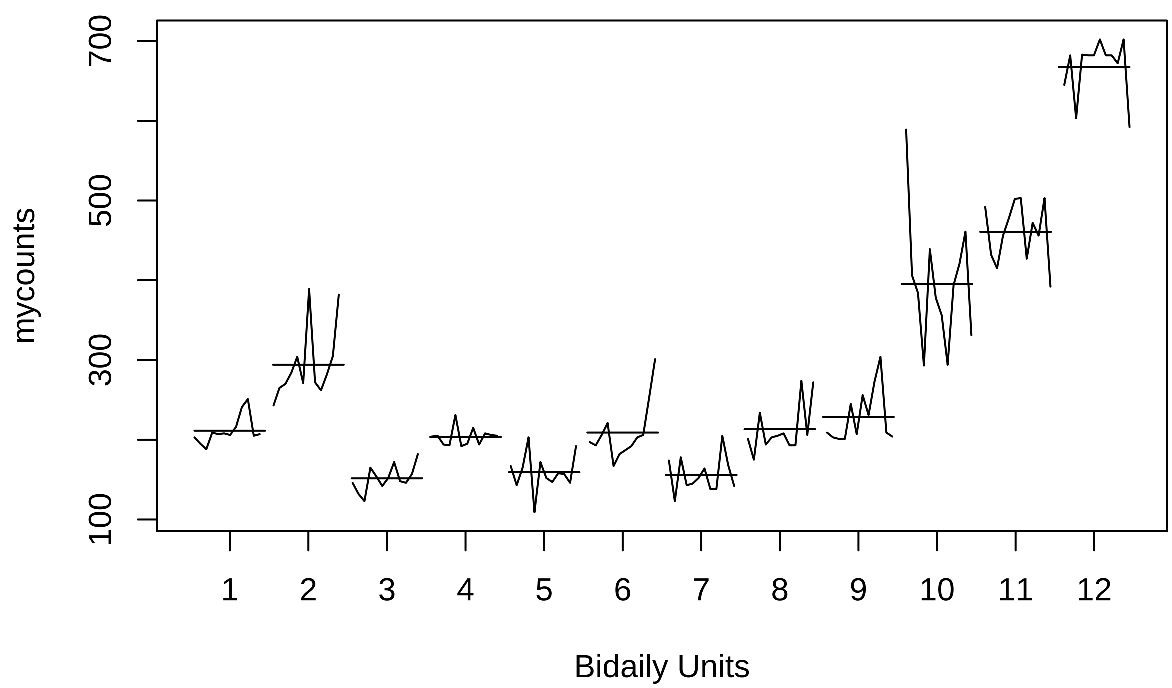
1.2 Visualisation of dataset

```
# using Rbase
plot(mycounts, ylab="Customer Counts",
     xlab="weeks")
```



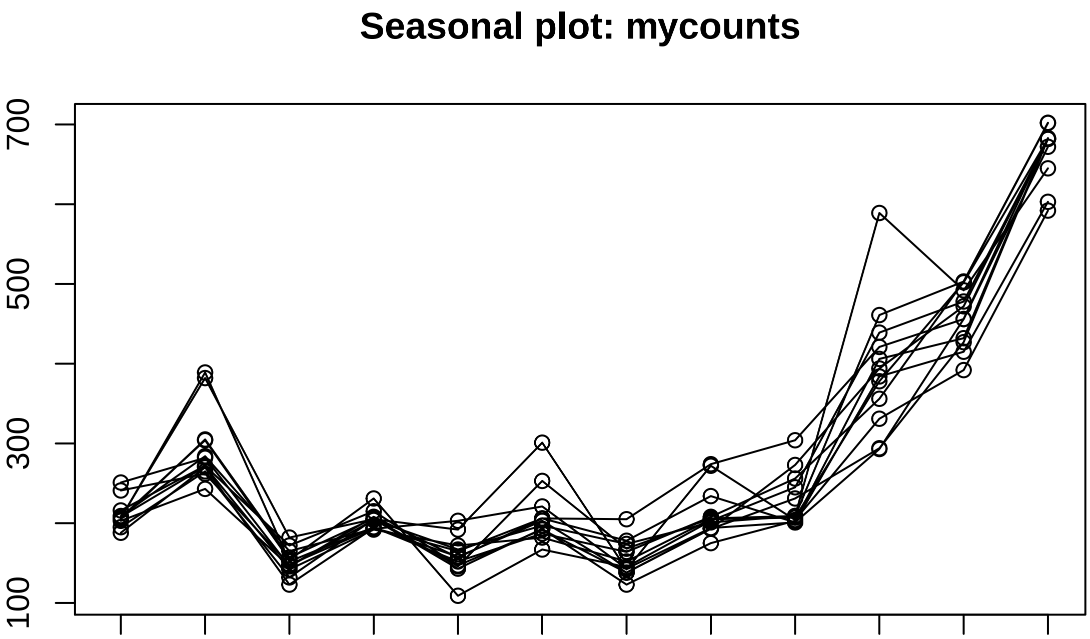
There is no trend (it doesn't go upwards), it is clean dataset, without missing data, outliers or other error.

```
# library(forecast)
# Calculates the average for each time unit. The unit of time doesn't necessarily have to be *month!!!!

monthplot(mycounts, labels=1:12, xlab="Bidaily Units")
```



Graph above represents 12 observations starting with Monday morning, followed by Monday afternoon and ending by Saturday afternoon. We can see that half days at the end of the week has much higher customer counts than the mornings of other week days.

```r
seasonplot(mycounts, season.labels=F, xlab="")
```



This graph shows that end of the week is the week when customers are coming to the store.

Month Plot - compares the single time units within the seasonal unit

Season Plot - Compares the seasonal units (cycles) to one another

Result: There is no trend in dataset but there is clear seasonal pattern. Simple models woudn't be able to catch seasonality (such as last observation carried forward or mean method).

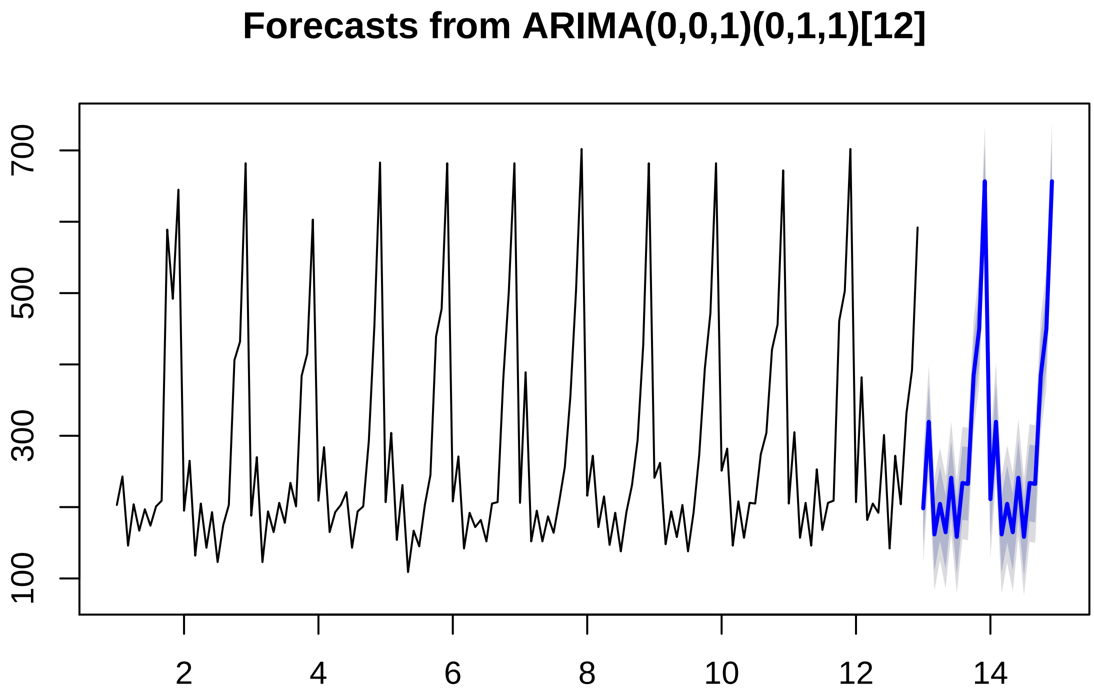
1.3 Model forecast

Choosing a suitable model: - A standard model could fit the data - ARIMA or exponential smoothing. They also perfom a forecast.

- The model needs to implement seasonality:
- Seasonal ARIMA
- Exponenential smoothing

Seasonal ARIMA - linear assumption - the forecast will be constant for the forecasted weeks - the data doesn't show any exponential character - no trend present

```r
plot(forecast(auto.arima(mycounts)))
```



Blue line represents 80% of accuracy, grey line 95%; Number of customers on the weekends is drastically higher than on the weekdays. Morning are less busy than afternoons.

2 Packages, Data Introduction, Date Classes

2.1 New packages introduction

Package lubridate - hadles time and date better than R Base - simplifies the POSIX classes of R Base

Package forecast - for time series modelling - functions for forecasting and modelling (e.g. auto.arima())

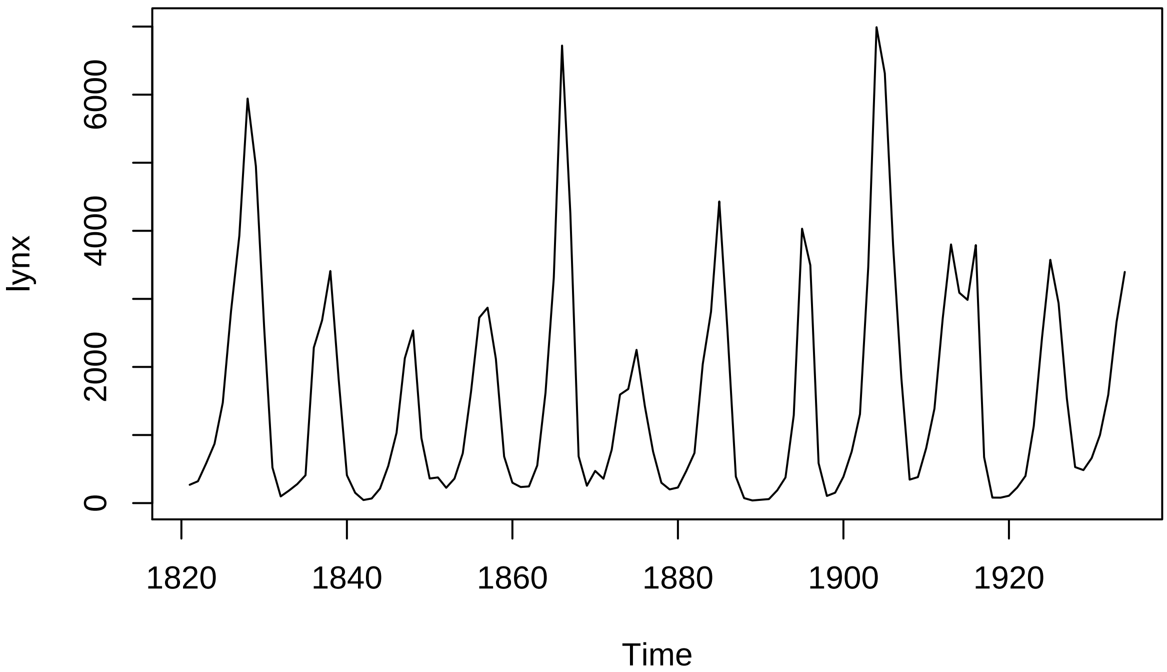
Package tseries - collects useful tools for working with time series data - modelling features, plotting tools, data formatting capabilities

2.2 Datasets Introduction

2.2.1 Lynx

- Annual numbers of lynx trappings for 1821-1934 in Canada
- The plot looks stationary with equal mean and variance
- Some autocorrelation might be present => repetitive problem

```
plot(lynx); length(lynx)
```

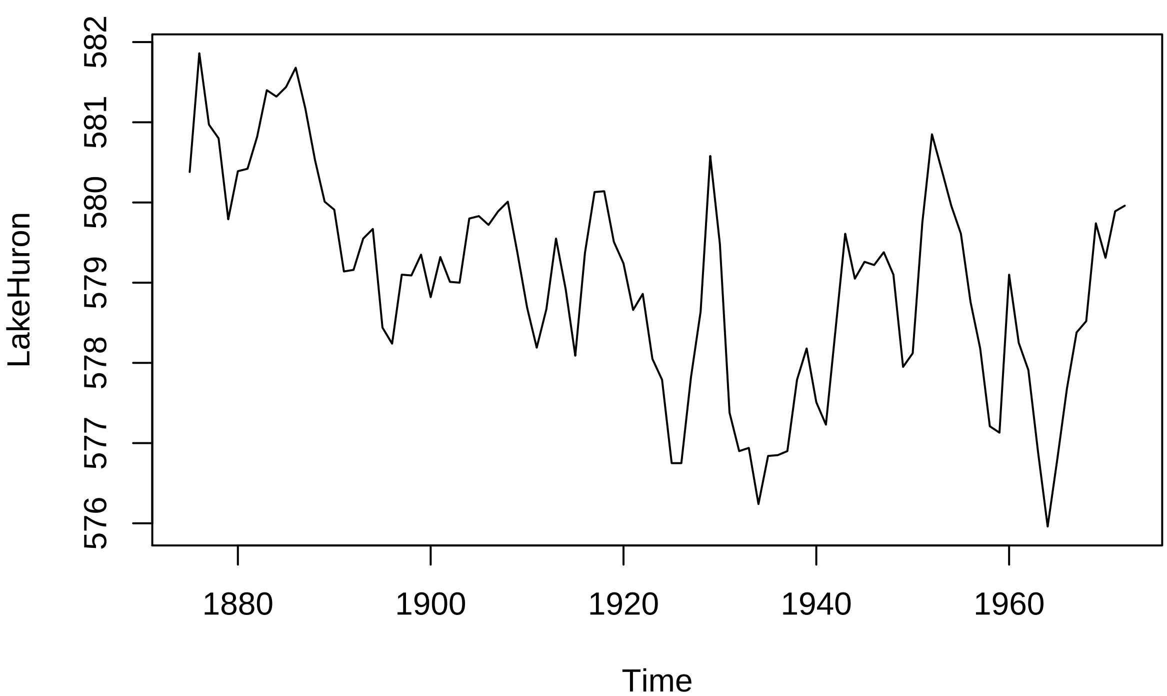



```
## [1] 114
```

2.2.2 LakeHuron

- Annual measurements (1875-1973) of Lake Huron water levels in feet
- Annual data - non-seasonal
- The plot looks like a random walk (very little pattern)

```r
plot(LakeHuron); length(LakeHuron)
```

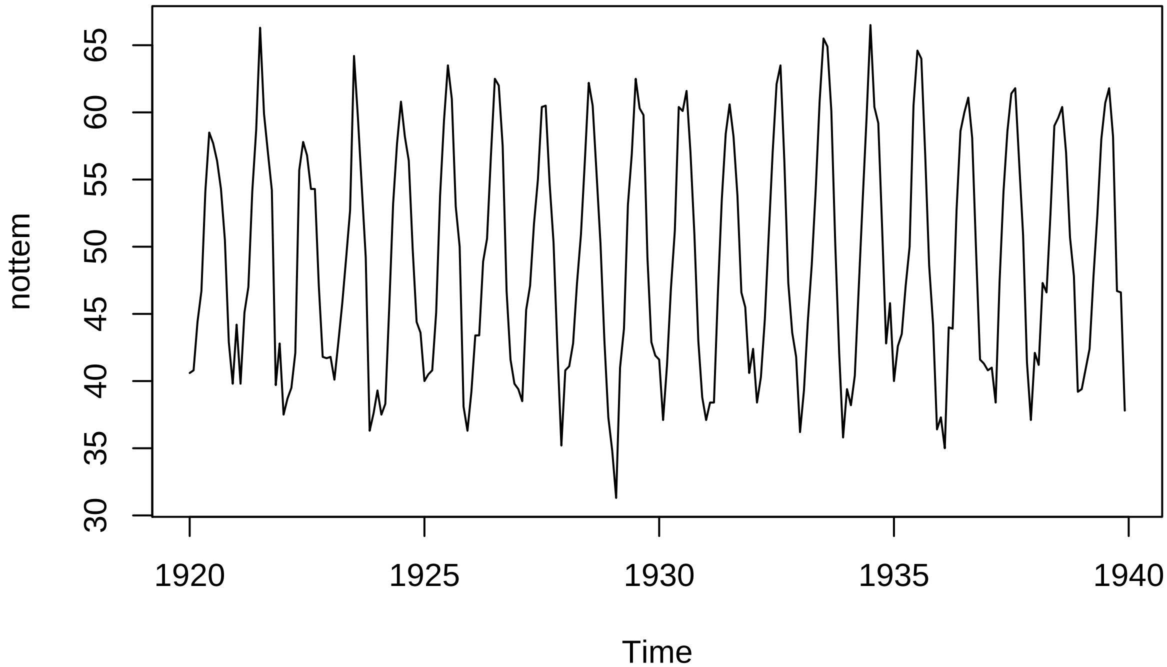



```
## [1] 98
```

2.2.3 nottem

- Temperature measurements for 1920-1940 in Nottingham
- Monthly dataset with 240 observations
- There is no trend or change in variance
- A really good example for seasonal data

```
plot(nottem); length(nottem)
```

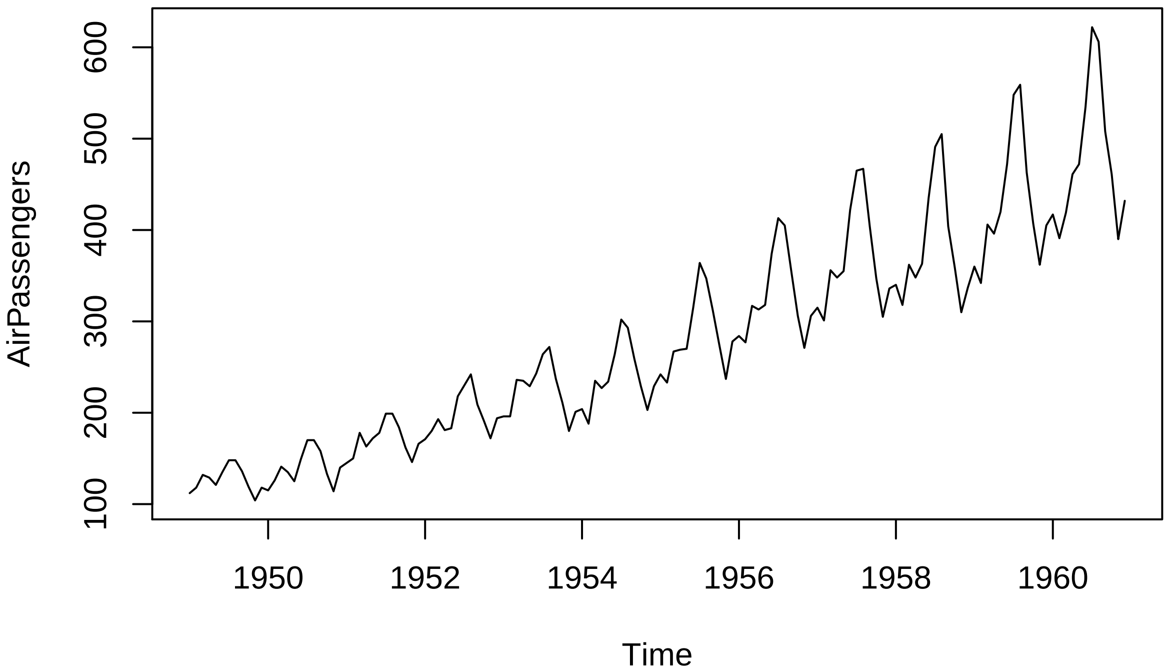



```
## [1] 240
```

2.2.4 AirPassengers

- The monthly volume of passengers for 1949 - 1961 in thousands
- Several statistical traits influence the pattern
- trend, seasonality, trend within the seasons

```
plot(AirPassengers); length(AirPassengers)
```

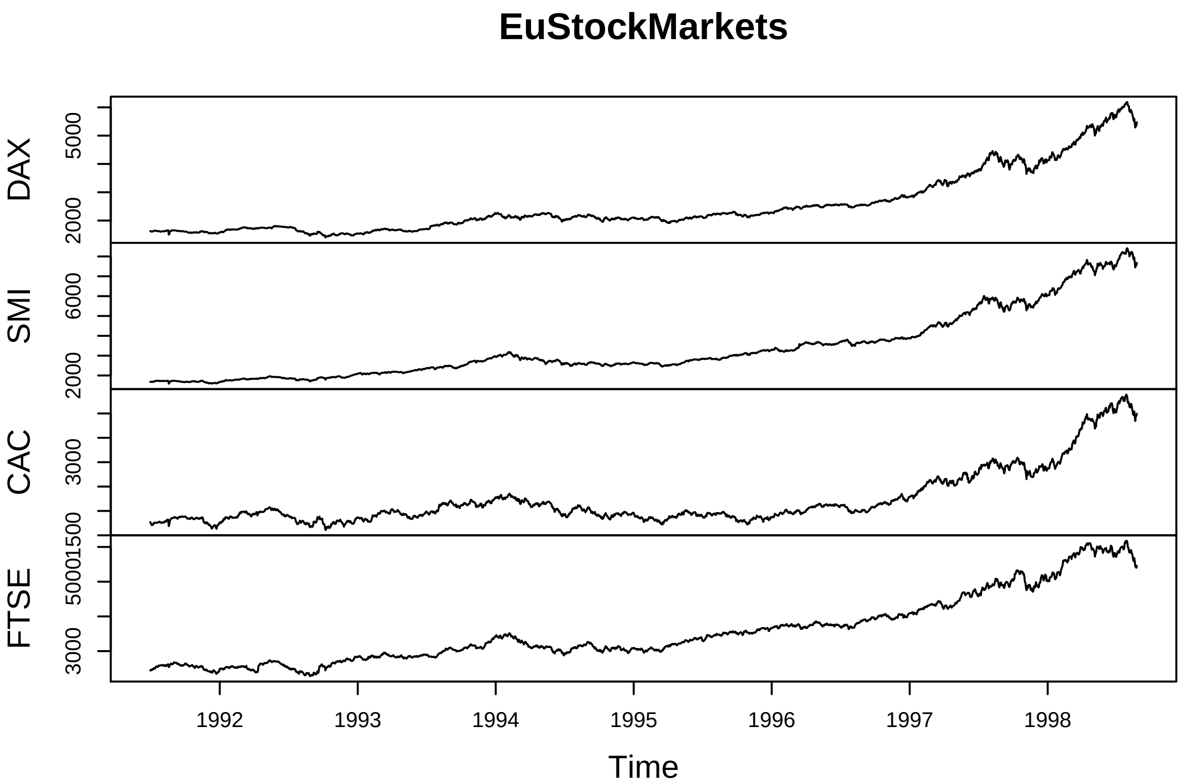



```
## [1] 144
```

2.2.5 EuStockMarkets

- The major stock indices in Europe for 1991 - 1999
- Multivariate time series data
- class = mts -> matrix structure
- Trend is present for all four indices

```r
plot(EuStockMarkets); length(EuStockMarkets)
```

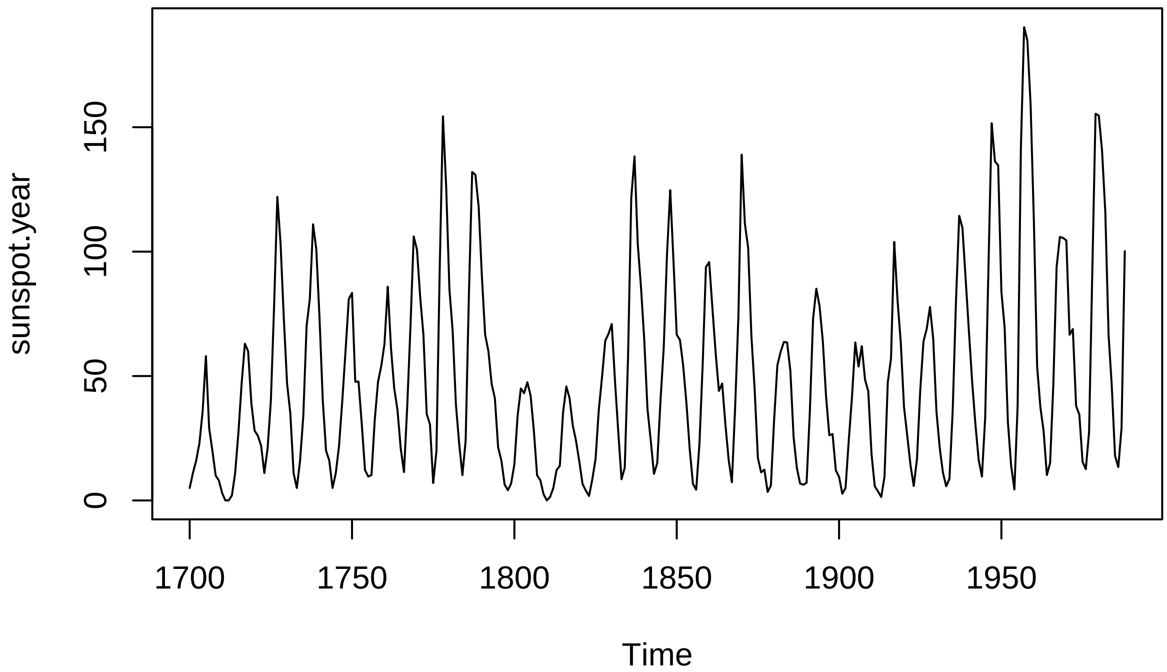



```
## [1] 7440
```

2.2.6 sunsport.year

- Yearly data for 1700-1989
- Autocorrelation might be present
- The dataset might require a more sophisticated model

```
plot(sunspot.year); length(sunspot.year)
```




```
## [1] 289
```

2.2.7 rnorm

```
?rnorm
```

2.3 POSIXt, Date and Chron Classes

POSIXt is probably the best known (R Base) package to handle a metaclass, POSIXct and POSIXit. Both (POSIX) are able to handle time zones, dates and time.

2.3.1 POSIXt classes in R

```
x = as.POSIXct("2015-12-25 11:45:34") # nr of seconds

y = as.POSIXlt("2015-12-25 11:45:34")

x; y # it gives the same output, but what is behind it?
```

```
## [1] "2015-12-25 11:45:34 GMT"
```

```
## [1] "2015-12-25 11:45:34 GMT"
```

```
unclass(x)
```

```
## [1] 1451043934
## attr(,"tzone")
## [1] ""
```

```
unclass(y)
```

```
## $sec
## [1] 34
##
## $min
## [1] 45
##
## $hour
## [1] 11
##
## $mday
## [1] 25
##
## $mon
## [1] 11
##
## $year
## [1] 115
##
## $wday
## [1] 5
##
## $yday
## [1] 358
##
## $isdst
## [1] 0
##
## $zone
## [1] "GMT"
##
## $gmtoff
## [1] NA
```

```r
# what does the number mean? (end of 2015)

46 * 365 * 24 * 60 * 60
```

```
## [1] 1450656000
```

```r
y$zone # extracting the elements from POSIXlt
```

```
## [1] "GMT"
```

```r
#x$zone # not possible since it is simply a number of seconds
```

```r
# another class based on days

x = as.Date("2015-12-25"); x
```

```
## [1] "2015-12-25"
```

```r
class(x)
```

```
## [1] "Date"
```

```r
unclass(x)
```

```
## [1] 16794
```

```r
46 * 365 # nr of days since 1970
```

```
## [1] 16790
```

```r
#library(chron)

x = chron("12/25/2015", "23:34:09"); x
```

```
## [1] (12/25/15 23:34:09)
```

```r
class(x)
```

```
## [1] "chron" "dates" "times"
```

```r
unclass(x)
```

```
## [1] 16794.98
## attr(,"format")
##   dates   times
## "m/d/y" "h:m:s"
## attr(,"origin")
## month   day  year
##     1     1  1970
```

2.4 Package lubridate

```r
#library(lubridate)
```

2.4.1 Different ways in how to input dates

```r
ymd(19931123)
```

```
## [1] "1993-11-23"
```

```r
dmy(23111993)
```

```
## [1] "1993-11-23"
```

```r
mdy(11231993)
```

```
## [1] "1993-11-23"
```

```r
# lets use time and date together

mytimepoint <- ymd_hm("1993-11-23 11:23", tz = "Europe/Prague")

mytimepoint
```

```
## [1] "1993-11-23 11:23:00 CET"
```

2.4.2 Extracting the components

```r
minute(mytimepoint)
```

```
## [1] 23
```

```r
day(mytimepoint)
```

```
## [1] 23
```

```r
hour(mytimepoint)
```

```
## [1] 11
```

```r
# we can even change time values within our object

hour(mytimepoint) <- 14

mytimepoint
```

```
## [1] "1993-11-23 14:23:00 CET"
```

2.4.3 Available Time Zones

```r
olson_time_zones()
```

```
##   [1] "Africa/Abidjan"                   "Africa/Accra"
##   [3] "Africa/Addis_Ababa"               "Africa/Algiers"
##   [5] "Africa/Asmara"                    "Africa/Bamako"
##   [7] "Africa/Bangui"                    "Africa/Banjul"
##   [9] "Africa/Bissau"                    "Africa/Blantyre"
##  [11] "Africa/Brazzaville"               "Africa/Bujumbura"
##  [13] "Africa/Cairo"                     "Africa/Casablanca"
##  [15] "Africa/Ceuta"                     "Africa/Conakry"
##  [17] "Africa/Dakar"                     "Africa/Dar_es_Salaam"
##  [19] "Africa/Djibouti"                  "Africa/Douala"
##  [21] "Africa/El_Aaiun"                  "Africa/Freetown"
##  [23] "Africa/Gaborone"                  "Africa/Harare"
##  [25] "Africa/Johannesburg"              "Africa/Juba"
##  [27] "Africa/Kampala"                   "Africa/Khartoum"
##  [29] "Africa/Kigali"                    "Africa/Kinshasa"
##  [31] "Africa/Lagos"                     "Africa/Libreville"
##  [33] "Africa/Lome"                      "Africa/Luanda"
##  [35] "Africa/Lubumbashi"                "Africa/Lusaka"
##  [37] "Africa/Malabo"                    "Africa/Maputo"
##  [39] "Africa/Maseru"                    "Africa/Mbabane"
##  [41] "Africa/Mogadishu"                 "Africa/Monrovia"
##  [43] "Africa/Nairobi"                   "Africa/Ndjamena"
##  [45] "Africa/Niamey"                    "Africa/Nouakchott"
##  [47] "Africa/Ouagadougou"               "Africa/Porto-Novo"
##  [49] "Africa/Sao_Tome"                  "Africa/Tripoli"
##  [51] "Africa/Tunis"                     "Africa/Windhoek"
##  [53] "America/Adak"                     "America/Anchorage"
##  [55] "America/Anguilla"                 "America/Antigua"
##  [57] "America/Araguaina"                "America/Argentina/Buenos_Aires"
##  [59] "America/Argentina/Catamarca"      "America/Argentina/Cordoba"
##  [61] "America/Argentina/Jujuy"          "America/Argentina/La_Rioja"
##  [63] "America/Argentina/Mendoza"        "America/Argentina/Rio_Gallegos"
##  [65] "America/Argentina/Salta"          "America/Argentina/San_Juan"
##  [67] "America/Argentina/San_Luis"       "America/Argentina/Tucuman"
```

```
##  [69] "America/Argentina/Ushuaia"      "America/Aruba"
##  [71] "America/Asuncion"               "America/Atikokan"
##  [73] "America/Bahia"                  "America/Bahia_Banderas"
##  [75] "America/Barbados"               "America/Belem"
##  [77] "America/Belize"                 "America/Blanc-Sablon"
##  [79] "America/Boa_Vista"              "America/Bogota"
##  [81] "America/Boise"                  "America/Cambridge_Bay"
##  [83] "America/Campo_Grande"           "America/Cancun"
##  [85] "America/Caracas"                "America/Cayenne"
##  [87] "America/Cayman"                 "America/Chicago"
##  [89] "America/Chihuahua"              "America/Costa_Rica"
##  [91] "America/Creston"                "America/Cuiaba"
##  [93] "America/Curacao"                "America/Danmarkshavn"
##  [95] "America/Dawson"                 "America/Dawson_Creek"
##  [97] "America/Denver"                 "America/Detroit"
##  [99] "America/Dominica"               "America/Edmonton"
## [101] "America/Eirunepe"               "America/El_Salvador"
## [103] "America/Fort_Nelson"            "America/Fortaleza"
## [105] "America/Glace_Bay"              "America/Godthab"
## [107] "America/Goose_Bay"              "America/Grand_Turk"
## [109] "America/Grenada"                "America/Guadeloupe"
## [111] "America/Guatemala"              "America/Guayaquil"
## [113] "America/Guyana"                 "America/Halifax"
## [115] "America/Havana"                 "America/Hermosillo"
## [117] "America/Indiana/Indianapolis"   "America/Indiana/Knox"
## [119] "America/Indiana/Marengo"        "America/Indiana/Petersburg"
## [121] "America/Indiana/Tell_City"      "America/Indiana/Vevay"
## [123] "America/Indiana/Vincennes"      "America/Indiana/Winamac"
## [125] "America/Inuvik"                 "America/Iqaluit"
## [127] "America/Jamaica"                "America/Juneau"
## [129] "America/Kentucky/Louisville"    "America/Kentucky/Monticello"
## [131] "America/Kralendijk"             "America/La_Paz"
## [133] "America/Lima"                   "America/Los_Angeles"
## [135] "America/Lower_Princes"          "America/Maceio"
## [137] "America/Managua"                "America/Manaus"
## [139] "America/Marigot"                "America/Martinique"
## [141] "America/Matamoros"              "America/Mazatlan"
## [143] "America/Menominee"              "America/Merida"
## [145] "America/Metlakatla"             "America/Mexico_City"
## [147] "America/Miquelon"               "America/Moncton"
## [149] "America/Monterrey"              "America/Montevideo"
## [151] "America/Montserrat"             "America/Nassau"
## [153] "America/New_York"               "America/Nipigon"
## [155] "America/Nome"                   "America/Noronha"
## [157] "America/North_Dakota/Beulah"    "America/North_Dakota/Center"
## [159] "America/North_Dakota/New_Salem" "America/Ojinaga"
## [161] "America/Panama"                 "America/Pangnirtung"
## [163] "America/Paramaribo"             "America/Phoenix"
## [165] "America/Port_of_Spain"          "America/Port-au-Prince"
## [167] "America/Porto_Velho"            "America/Puerto_Rico"
## [169] "America/Rainy_River"            "America/Rankin_Inlet"
## [171] "America/Recife"                 "America/Regina"
## [173] "America/Resolute"               "America/Rio_Branco"
## [175] "America/Santarem"               "America/Santiago"
```

```
## [177] "America/Santo_Domingo"          "America/Sao_Paulo"
## [179] "America/Scoresbysund"           "America/Sitka"
## [181] "America/St_Barthelemy"          "America/St_Johns"
## [183] "America/St_Kitts"               "America/St_Lucia"
## [185] "America/St_Thomas"              "America/St_Vincent"
## [187] "America/Swift_Current"          "America/Tegucigalpa"
## [189] "America/Thule"                  "America/Thunder_Bay"
## [191] "America/Tijuana"                "America/Toronto"
## [193] "America/Tortola"                "America/Vancouver"
## [195] "America/Whitehorse"             "America/Winnipeg"
## [197] "America/Yakutat"                "America/Yellowknife"
## [199] "Antarctica/Casey"               "Antarctica/Davis"
## [201] "Antarctica/DumontDUrville"      "Antarctica/Macquarie"
## [203] "Antarctica/Mawson"              "Antarctica/McMurdo"
## [205] "Antarctica/Palmer"              "Antarctica/Rothera"
## [207] "Antarctica/Syowa"               "Antarctica/Troll"
## [209] "Antarctica/Vostok"              "Arctic/Longyearbyen"
## [211] "Asia/Aden"                      "Asia/Almaty"
## [213] "Asia/Amman"                     "Asia/Anadyr"
## [215] "Asia/Aqtau"                     "Asia/Aqtobe"
## [217] "Asia/Ashgabat"                  "Asia/Atyrau"
## [219] "Asia/Baghdad"                   "Asia/Bahrain"
## [221] "Asia/Baku"                      "Asia/Bangkok"
## [223] "Asia/Barnaul"                   "Asia/Beirut"
## [225] "Asia/Bishkek"                   "Asia/Brunei"
## [227] "Asia/Chita"                     "Asia/Choibalsan"
## [229] "Asia/Colombo"                   "Asia/Damascus"
## [231] "Asia/Dhaka"                     "Asia/Dili"
## [233] "Asia/Dubai"                     "Asia/Dushanbe"
## [235] "Asia/Famagusta"                 "Asia/Gaza"
## [237] "Asia/Hebron"                    "Asia/Ho_Chi_Minh"
## [239] "Asia/Hong_Kong"                 "Asia/Hovd"
## [241] "Asia/Irkutsk"                   "Asia/Jakarta"
## [243] "Asia/Jayapura"                  "Asia/Jerusalem"
## [245] "Asia/Kabul"                     "Asia/Kamchatka"
## [247] "Asia/Karachi"                   "Asia/Kathmandu"
## [249] "Asia/Khandyga"                  "Asia/Kolkata"
## [251] "Asia/Krasnoyarsk"               "Asia/Kuala_Lumpur"
## [253] "Asia/Kuching"                   "Asia/Kuwait"
## [255] "Asia/Macau"                     "Asia/Magadan"
## [257] "Asia/Makassar"                  "Asia/Manila"
## [259] "Asia/Muscat"                    "Asia/Nicosia"
## [261] "Asia/Novokuznetsk"              "Asia/Novosibirsk"
## [263] "Asia/Omsk"                      "Asia/Oral"
## [265] "Asia/Phnom_Penh"                "Asia/Pontianak"
## [267] "Asia/Pyongyang"                 "Asia/Qatar"
## [269] "Asia/Qyzylorda"                 "Asia/Riyadh"
## [271] "Asia/Sakhalin"                  "Asia/Samarkand"
## [273] "Asia/Seoul"                     "Asia/Shanghai"
## [275] "Asia/Singapore"                 "Asia/Srednekolymsk"
## [277] "Asia/Taipei"                    "Asia/Tashkent"
## [279] "Asia/Tbilisi"                   "Asia/Tehran"
## [281] "Asia/Thimphu"                   "Asia/Tokyo"
## [283] "Asia/Tomsk"                     "Asia/Ulaanbaatar"
```

```
## [285] "Asia/Urumqi"                      "Asia/Ust-Nera"
## [287] "Asia/Vientiane"                   "Asia/Vladivostok"
## [289] "Asia/Yakutsk"                     "Asia/Yangon"
## [291] "Asia/Yekaterinburg"               "Asia/Yerevan"
## [293] "Atlantic/Azores"                  "Atlantic/Bermuda"
## [295] "Atlantic/Canary"                  "Atlantic/Cape_Verde"
## [297] "Atlantic/Faroe"                   "Atlantic/Madeira"
## [299] "Atlantic/Reykjavik"               "Atlantic/South_Georgia"
## [301] "Atlantic/St_Helena"               "Atlantic/Stanley"
## [303] "Australia/Adelaide"               "Australia/Brisbane"
## [305] "Australia/Broken_Hill"            "Australia/Currie"
## [307] "Australia/Darwin"                 "Australia/Eucla"
## [309] "Australia/Hobart"                 "Australia/Lindeman"
## [311] "Australia/Lord_Howe"              "Australia/Melbourne"
## [313] "Australia/Perth"                  "Australia/Sydney"
## [315] "Europe/Amsterdam"                 "Europe/Andorra"
## [317] "Europe/Astrakhan"                 "Europe/Athens"
## [319] "Europe/Belgrade"                  "Europe/Berlin"
## [321] "Europe/Bratislava"                "Europe/Brussels"
## [323] "Europe/Bucharest"                 "Europe/Budapest"
## [325] "Europe/Busingen"                  "Europe/Chisinau"
## [327] "Europe/Copenhagen"                "Europe/Dublin"
## [329] "Europe/Gibraltar"                 "Europe/Guernsey"
## [331] "Europe/Helsinki"                  "Europe/Isle_of_Man"
## [333] "Europe/Istanbul"                  "Europe/Jersey"
## [335] "Europe/Kaliningrad"               "Europe/Kiev"
## [337] "Europe/Kirov"                     "Europe/Lisbon"
## [339] "Europe/Ljubljana"                 "Europe/London"
## [341] "Europe/Luxembourg"                "Europe/Madrid"
## [343] "Europe/Malta"                     "Europe/Mariehamn"
## [345] "Europe/Minsk"                     "Europe/Monaco"
## [347] "Europe/Moscow"                    "Europe/Oslo"
## [349] "Europe/Paris"                     "Europe/Podgorica"
## [351] "Europe/Prague"                    "Europe/Riga"
## [353] "Europe/Rome"                      "Europe/Samara"
## [355] "Europe/San_Marino"                "Europe/Sarajevo"
## [357] "Europe/Saratov"                   "Europe/Simferopol"
## [359] "Europe/Skopje"                    "Europe/Sofia"
## [361] "Europe/Stockholm"                 "Europe/Tallinn"
## [363] "Europe/Tirane"                    "Europe/Ulyanovsk"
## [365] "Europe/Uzhgorod"                  "Europe/Vaduz"
## [367] "Europe/Vatican"                   "Europe/Vienna"
## [369] "Europe/Vilnius"                   "Europe/Volgograd"
## [371] "Europe/Warsaw"                    "Europe/Zagreb"
## [373] "Europe/Zaporozhye"                "Europe/Zurich"
## [375] "Indian/Antananarivo"              "Indian/Chagos"
## [377] "Indian/Christmas"                 "Indian/Cocos"
## [379] "Indian/Comoro"                    "Indian/Kerguelen"
## [381] "Indian/Mahe"                      "Indian/Maldives"
## [383] "Indian/Mauritius"                 "Indian/Mayotte"
## [385] "Indian/Reunion"                   "Pacific/Apia"
## [387] "Pacific/Auckland"                 "Pacific/Bougainville"
## [389] "Pacific/Chatham"                  "Pacific/Chuuk"
## [391] "Pacific/Easter"                   "Pacific/Efate"
```

```
## [393] "Pacific/Enderbury"                "Pacific/Fakaofo"
## [395] "Pacific/Fiji"                     "Pacific/Funafuti"
## [397] "Pacific/Galapagos"                "Pacific/Gambier"
## [399] "Pacific/Guadalcanal"              "Pacific/Guam"
## [401] "Pacific/Honolulu"                 "Pacific/Johnston"
## [403] "Pacific/Kiritimati"               "Pacific/Kosrae"
## [405] "Pacific/Kwajalein"                "Pacific/Majuro"
## [407] "Pacific/Marquesas"                "Pacific/Midway"
## [409] "Pacific/Nauru"                    "Pacific/Niue"
## [411] "Pacific/Norfolk"                  "Pacific/Noumea"
## [413] "Pacific/Pago_Pago"                "Pacific/Palau"
## [415] "Pacific/Pitcairn"                 "Pacific/Pohnpei"
## [417] "Pacific/Port_Moresby"             "Pacific/Rarotonga"
## [419] "Pacific/Saipan"                   "Pacific/Tahiti"
## [421] "Pacific/Tarawa"                   "Pacific/Tongatapu"
## [423] "Pacific/Wake"                     "Pacific/Wallis"
```

```
# we can take a look at the most common time zones

# but be aware that the time zone recognition also depends on your location and machine

## lets check which day our time point is

wday(mytimepoint)
```

```
## [1] 3
```

```
wday(mytimepoint, label=T, abbr=F) # label to display the name of the day, no abbreviation
```

```
## [1] Tuesday
## 7 Levels: Sunday < Monday < Tuesday < Wednesday < Thursday < ... < Saturday
```

```
# we can calculate which time our timepoint would be in another time zone

with_tz(mytimepoint, tz = "Europe/London")
```

```
## [1] "1993-11-23 13:23:00 GMT"
```

```
mytimepoint
```

```
## [1] "1993-11-23 14:23:00 CET"
```

2.4.4 Time intervals

```
time1 = ymd_hm("1993-09-23 11:23", tz = "Europe/Prague")

time2 = ymd_hm("1995-11-02 15:23", tz = "Europe/Prague")

# getting the interval

myinterval = interval(time1, time2); myinterval
```

```
## [1] 1993-09-23 11:23:00 CEST--1995-11-02 15:23:00 CET
```

```
class(myinterval) # interval is an object class from lubridate
```

```
## [1] "Interval"
```

```
## attr(,"package")
## [1] "lubridate"
```

2.4.5 Exercise: Creating a Data Frame with lubridate

```r
# lets now build a dataframe with lubridate that contains date and time data

a = ymd(c(19981111, 19830123, 19820904, 19450509, 19821224, 19741203, 19871210), tz = "CET")

# now I am creating a time vector - using different notations of input

b = hms(c("22 4 5", "4-9-45", "11:9:56", "23 15 12", "14 26 34", "8 8 23", "21 16 14"))

f = rnorm(7,10); f = round(f, digits = 2)

date_time_measurement = cbind.data.frame(date = a, time = b, measurement = f)

date_time_measurement
```

```
##         date        time measurement
## 1 1998-11-11   22H 4M 5S        9.17
## 2 1983-01-23 4H -9M -45S       10.18
## 3 1982-09-04  11H 9M 56S       10.38
## 4 1945-05-09 23H 15M 12S       10.73
## 5 1982-12-24 14H 26M 34S       11.61
## 6 1974-12-03   8H 8M 23S       11.63
## 7 1987-12-10 21H 16M 14S       10.00
```

2.4.6 Calculations with time

```r
minutes(7)
```

```
## [1] "7M 0S"
```

```r
# note that class "Period" needs integers - full numbers

#minutes(2.5)

# getting the duration

dminutes(3)
```

```
## [1] "180s (~3 minutes)"
```

```r
# how to add minutes and seconds

minutes(2) + seconds(5)
```

```
## [1] "2M 5S"
```

```r
# more calculations

minutes(2) + seconds(76)
```

```
## [1] "2M 76S"
```

```r
# class "duration" to perform addition

as.duration(minutes(2) + seconds(75))
```

```
## [1] "195s (~3.25 minutes)"
```

```r
# lubridate has an array of time classes, period or duration differ!

# which year was a leap year?
  ##a year, occurring once every four years, which has 366 days including 29 February as an intercalary

leap_year(2009:2014)
```

```
## [1] FALSE FALSE FALSE  TRUE FALSE FALSE
```

```r
ymd(20140101) + years(1)
```

```
## [1] "2015-01-01"
```

```r
ymd(20140101) + dyears(1)
```

```
## [1] "2015-01-01"
```

```r
# lets do the whole thing with a leap year

leap_year(2016)
```

```
## [1] TRUE
```

```r
ymd(20160101) + years(1)
```

```
## [1] "2017-01-01"
```

```r
ymd(20160101) + dyears(1)
```

```
## [1] "2016-12-31"
```

```r
# as you see the duration is the one which is always 365 days

# the standard one (the period) makes the year a whole new unit (+1)
```

2.4.7 Exercise Lubridate

```r
# create x, with time zone CET and a given time point in 2014 of your choosing

# the time point consists of year, months, day and hour

x = ymd_hm(tz = "CET", "2014-04-12 23:12")


# change now the minute of x to 7 and check x in the same line of code
minute(x) = 7 ; x
```

```
## [1] "2014-04-12 23:07:00 CEST"
```

```r
# see which time it would be in London

with_tz(x, tz="Europe/London")
```

```
## [1] "2014-04-12 22:07:00 BST"
```

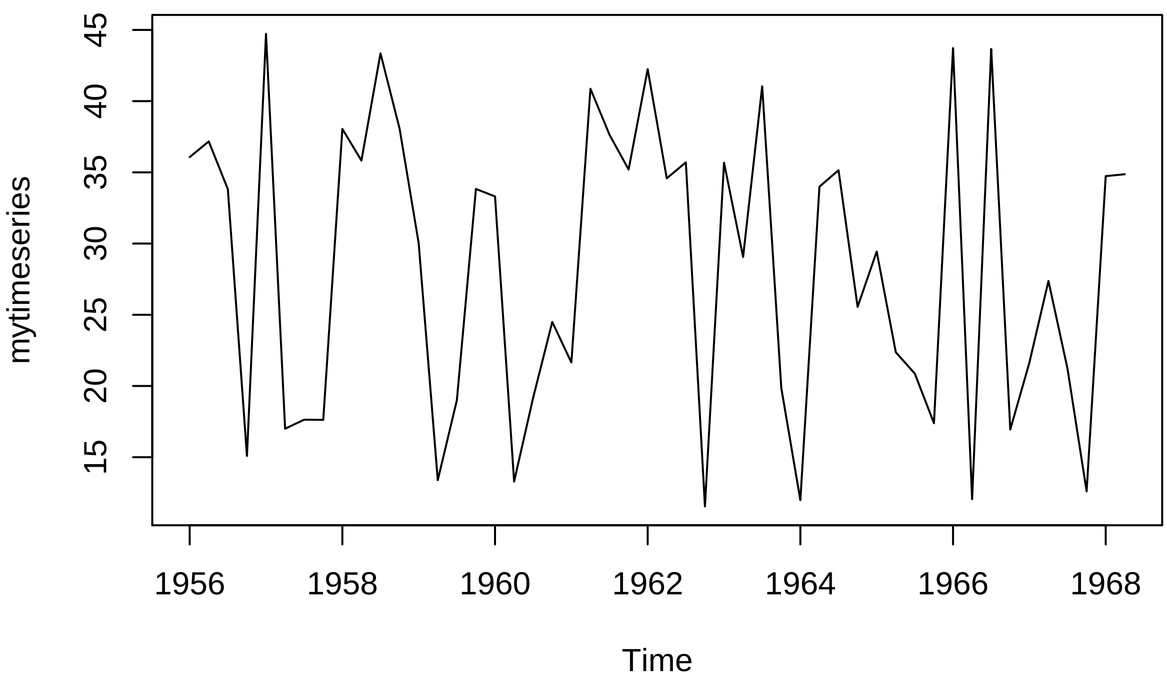
```r
# create another time point y in 2015 and get the difference between those 2 points

y = ymd_hm(tz = "CET", "2015-12-12 09:45")

y-x
```

```
## Time difference of 608.4847 days
```

3 Time Series

3.1 ts function

```r
# Getting data
mydata = runif(n = 50, min = 10, max = 45)

# ts for class time series
# Data starts in 1956 - 4 observations/year (quarterly)
mytimeseries = ts(data = mydata,
                  start = 1956, frequency = 4)

# Lets see how the data looks
plot(mytimeseries)
```




```r
# Checking the class
class(mytimeseries)
```

```
## [1] "ts"
```

```r
# Checking the timestamp
time(mytimeseries)
```

```
##         Qtr1    Qtr2    Qtr3    Qtr4
## 1956 1956.00 1956.25 1956.50 1956.75
## 1957 1957.00 1957.25 1957.50 1957.75
## 1958 1958.00 1958.25 1958.50 1958.75
## 1959 1959.00 1959.25 1959.50 1959.75
## 1960 1960.00 1960.25 1960.50 1960.75
## 1961 1961.00 1961.25 1961.50 1961.75
## 1962 1962.00 1962.25 1962.50 1962.75
## 1963 1963.00 1963.25 1963.50 1963.75
## 1964 1964.00 1964.25 1964.50 1964.75
## 1965 1965.00 1965.25 1965.50 1965.75
## 1966 1966.00 1966.25 1966.50 1966.75
## 1967 1967.00 1967.25 1967.50 1967.75
## 1968 1968.00 1968.25
```

```
# Refining the start argument
mytimeseries = ts(data = mydata,
                  start = c(1956,3), frequency = 4)

#start - When does the time series start? First available value of the first cycle
#frequency - How frequent are the observations? Number of observations per cycle?

#start and c() => use the 'start=' argument with the concatenate function - c()
#start = c(1956,3) - makes the time stamp start at the third quarter of 1956 (it the frequency is 4)
```

Time Stamp Combinations

Hourly measurements with daily patterns, starts at 8am on the first day:

start = c(1,8), frequency = 24

Measurements taken twice a day on workdays with weekly patterns, starts at the first week:

start = 1, frequency = 10 NA for holidays - regular spacing!

Monthly measurements with yearly cycle:

frequency = 12

Weekly measurements with yearly cycle:

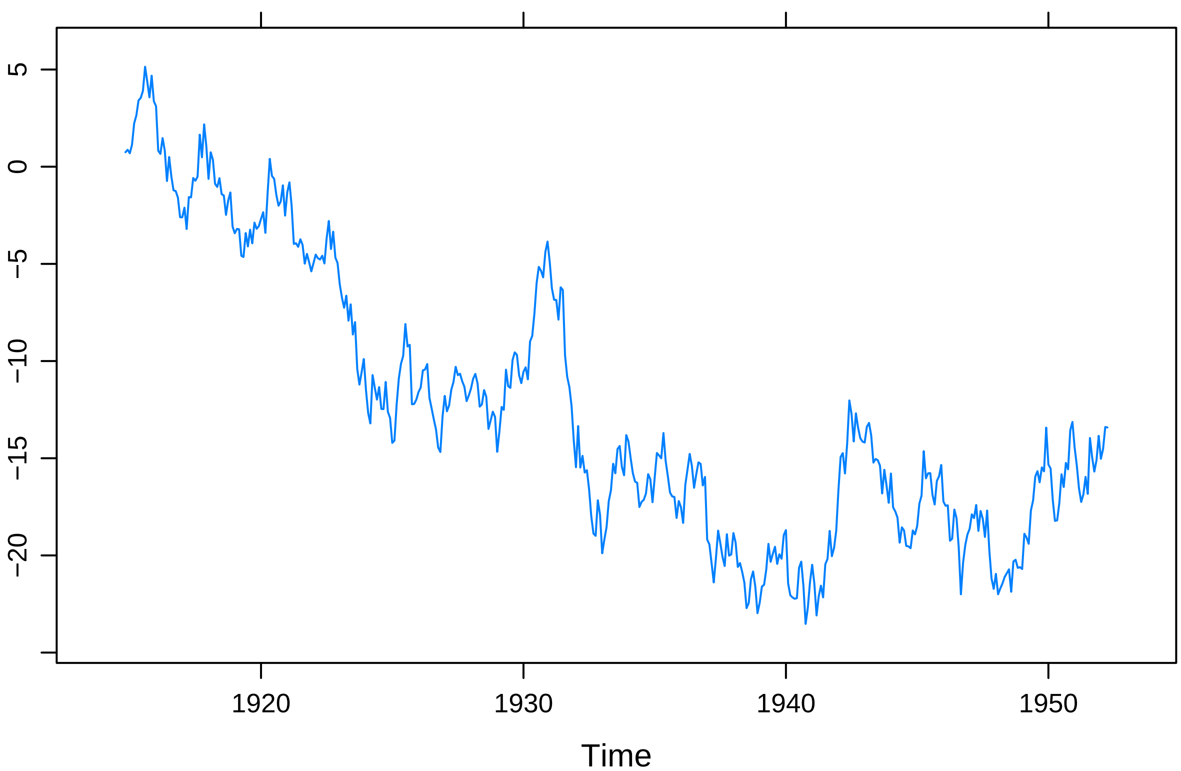
frequency = 52

3.1.1 Creating a ts object - Exercise

```
# Get a random walk of 450 numbers, eg rnorm, runif, etc

# In the solution I am going to use a cumulative sum on the normal distribution x = cumsum(rnorm(n = 45

# If you want it to be reproducible, you can set a seed

# Add the time component: it is a monthly dataset, which starts in November 1914

# Get a simple plot for this time series # Advanced: how would you get the same type with the "lattice"

x = cumsum(rnorm(n = 450))

y = ts(x, start = c(1914,11), frequency = 12)
```

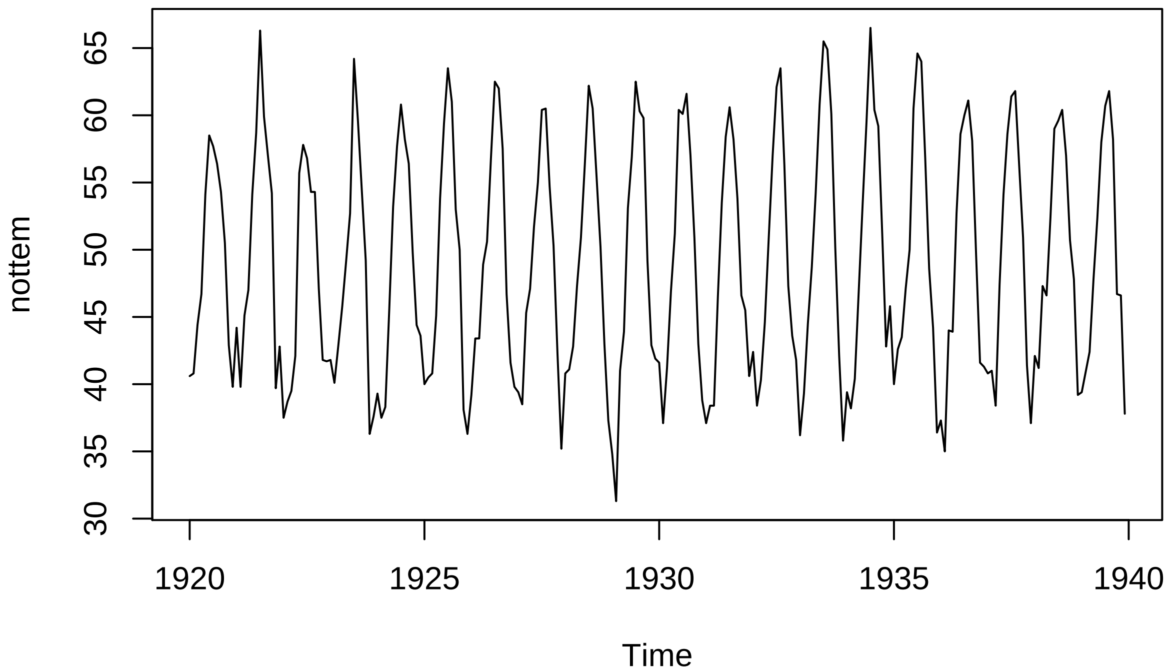
```r
plot(y)
```


```r
#same plot with lattice
#library(lattice)
xyplot.ts(y)
```

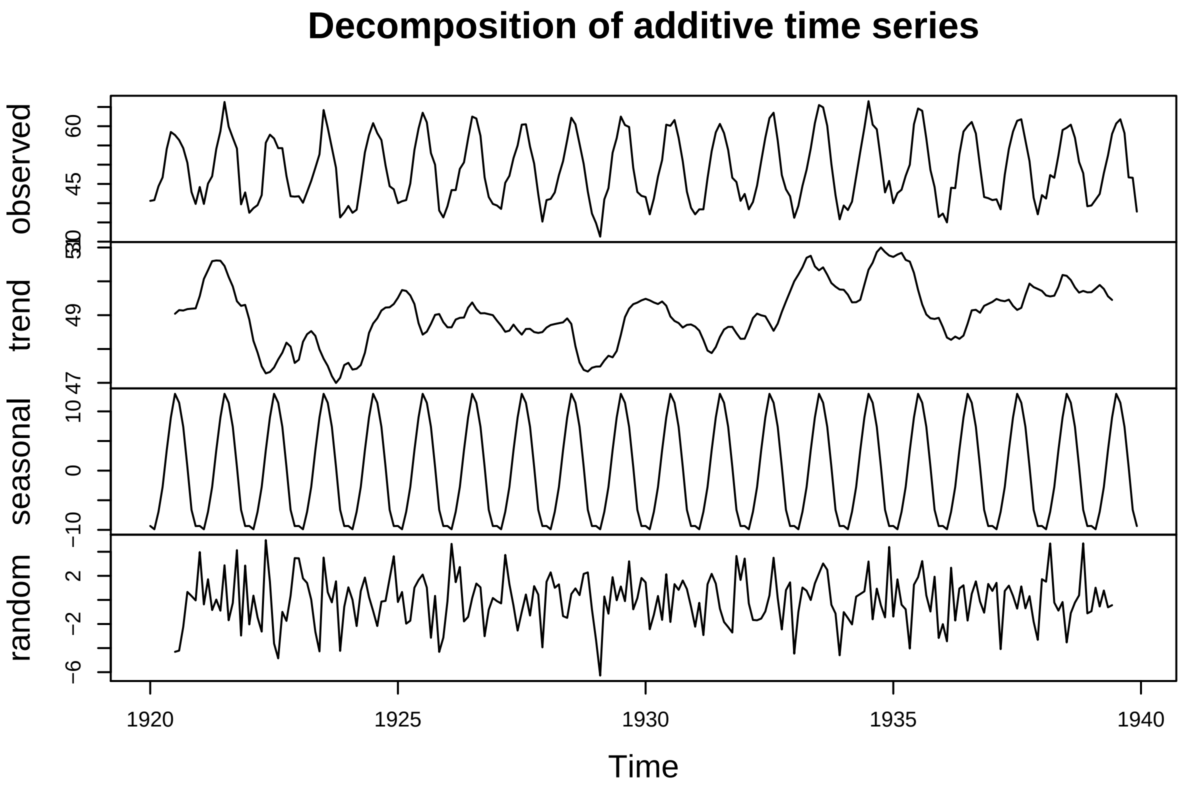


3.2 U Plots for time series data

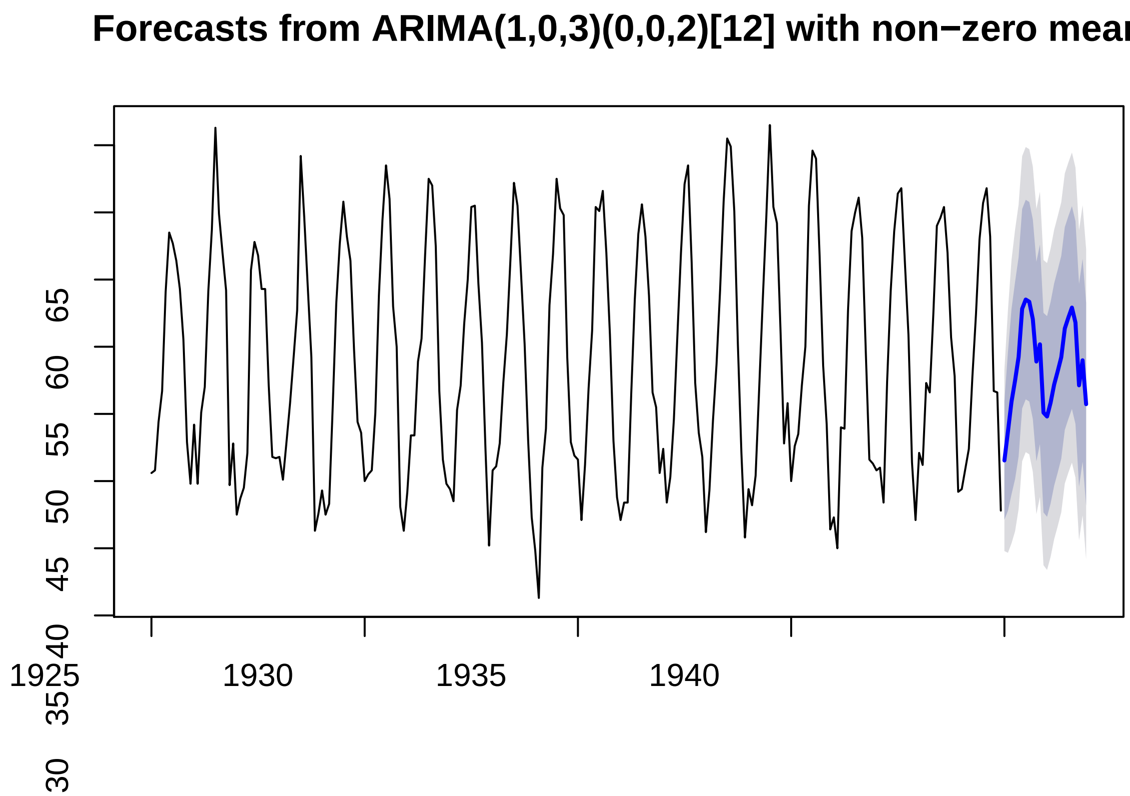
```r
# Standard R Base plots
plot(nottem)
```



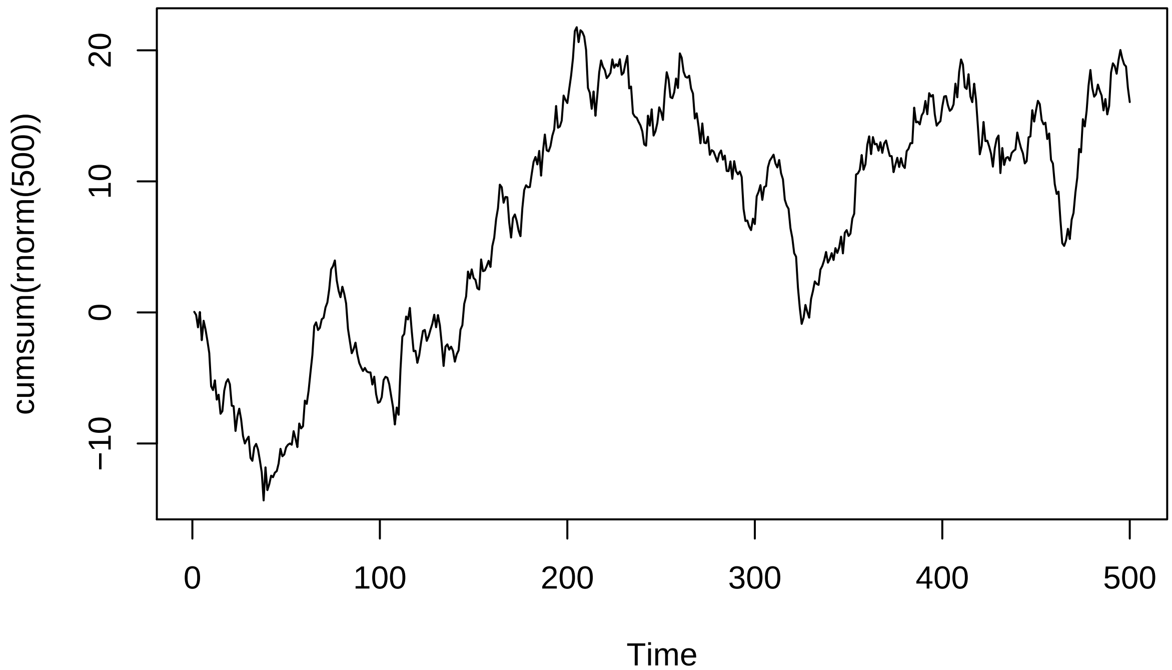

```r
# Plot of components
plot(decompose(nottem))
```



```r
# Directly plotting a forecast of a model
plot(forecast(auto.arima(nottem)), h = 5)
```



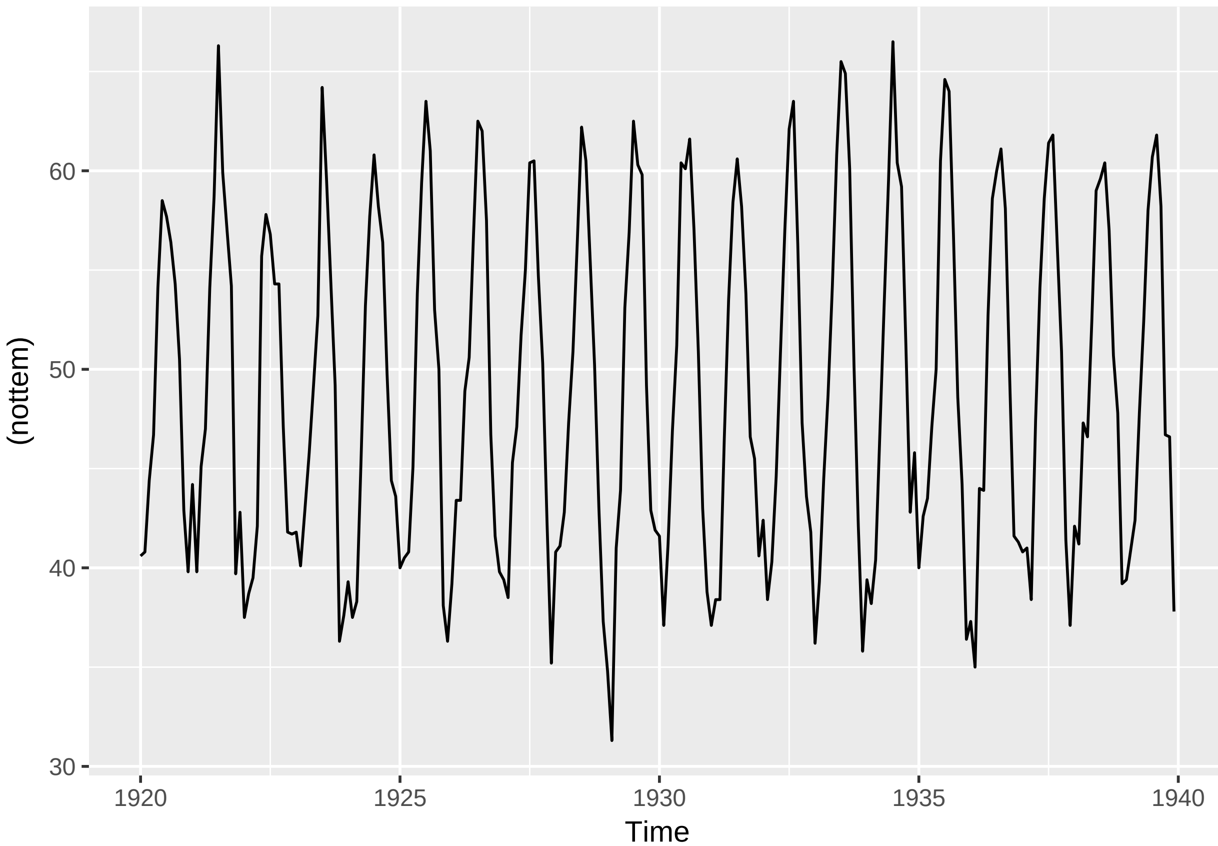

```r
#h = 5 => 5 years
# forcast function automatically recognizes which model is used and adjusts automatically. This holds t

# Random walk
plot.ts(cumsum(rnorm(500)))
```



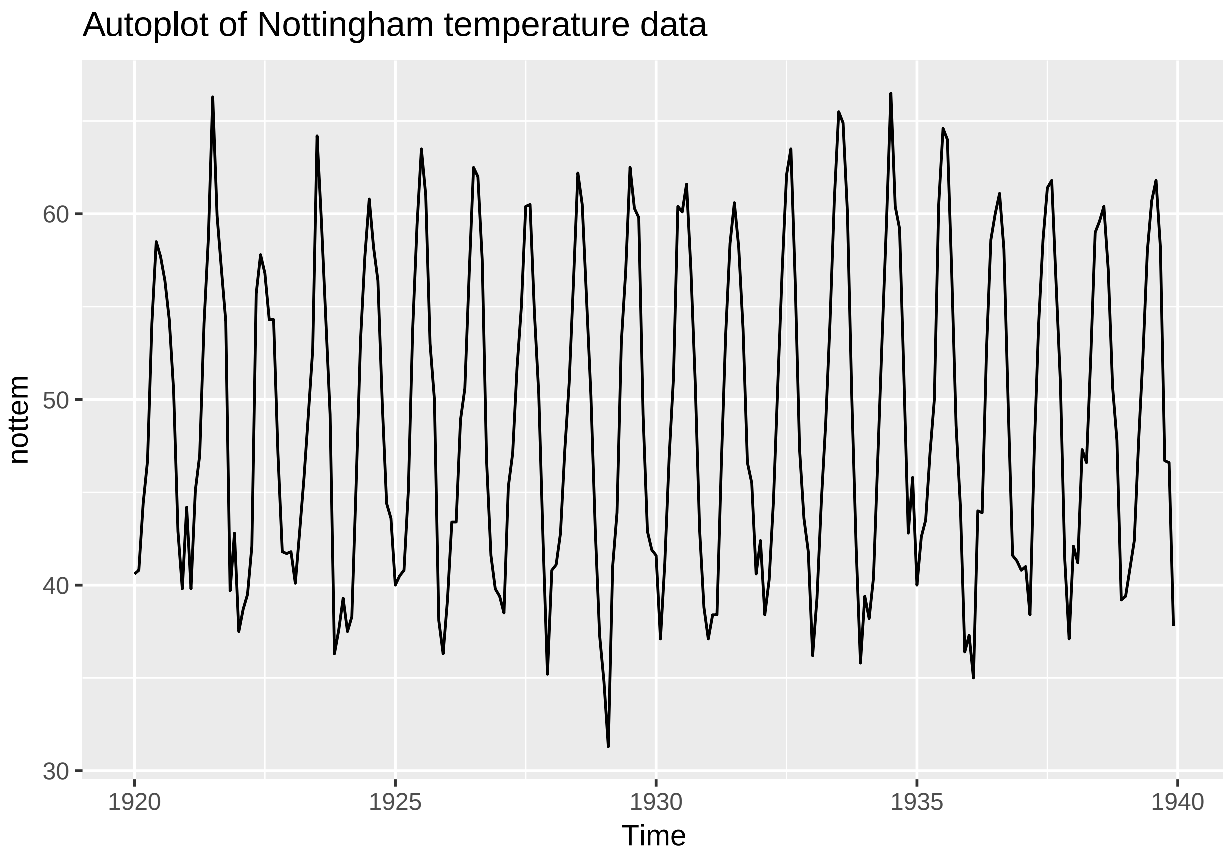

```
# This is used if my data hasn't been classified as time series yet; no conversion is required
```

3.3 Advanced plots

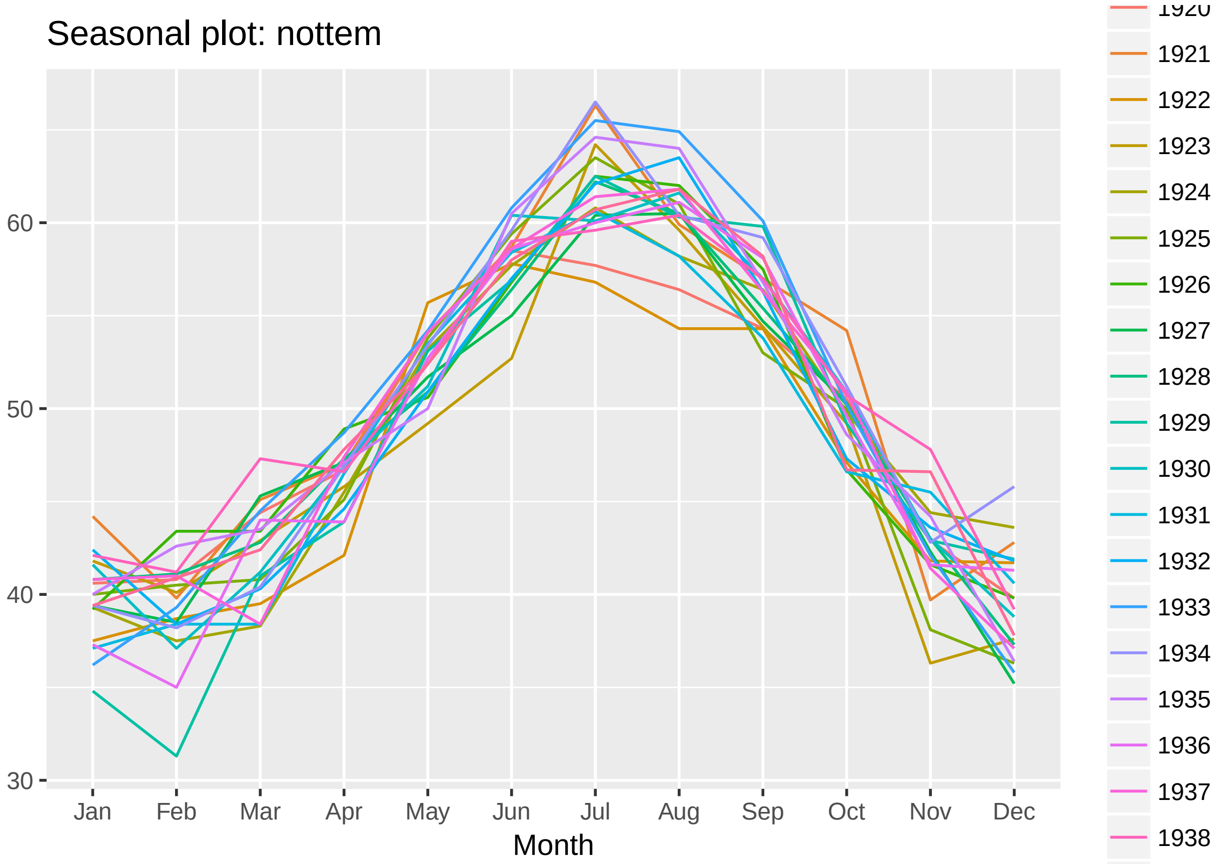
```
# Add on packages for advanced plots

  #library(forecast)
  #library(ggplot2)

# The ggplot equivalent to plot
autoplot((nottem))
```



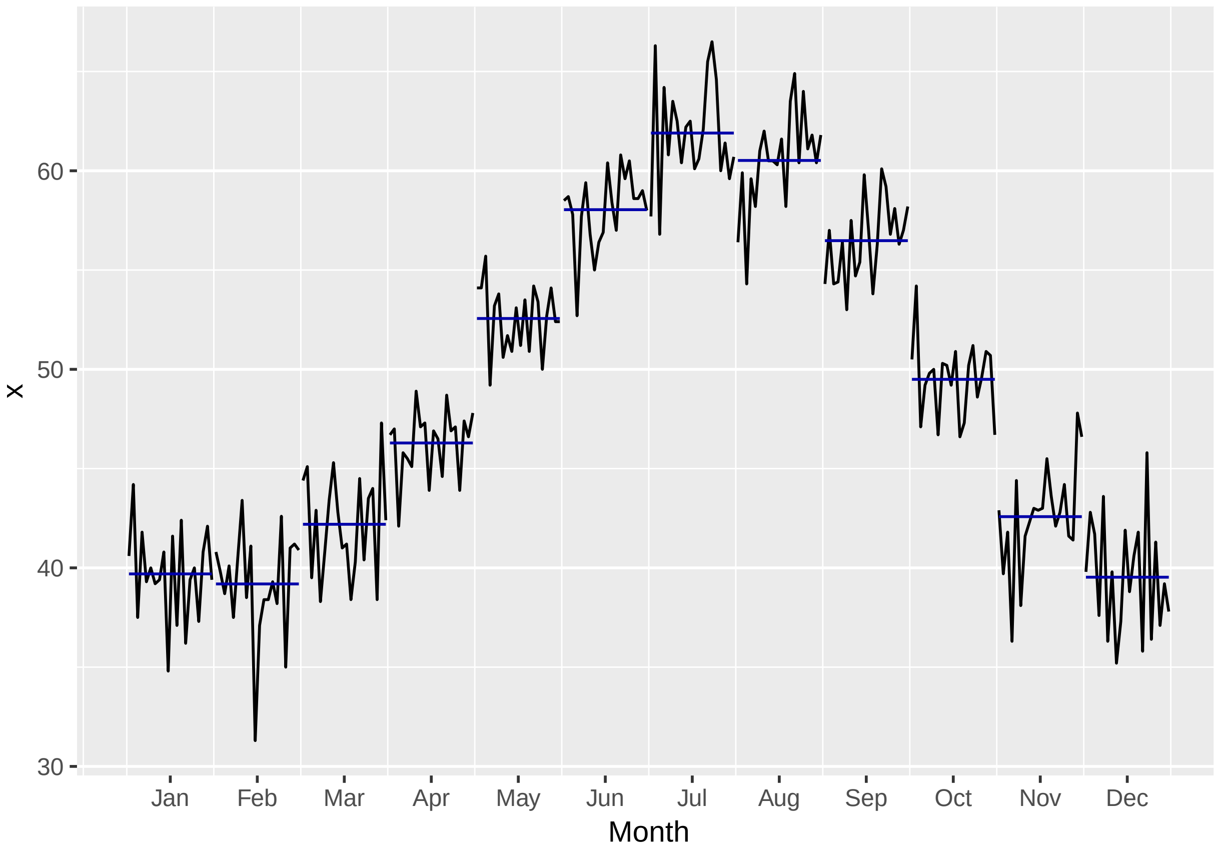

```r
# autoplot is coming from forecast package


# Ggplots work with different layers
autoplot(nottem) + ggtitle("Autoplot of Nottingham temperature data")
```




```r
# Time series specific plots
ggseasonplot(nottem)
```



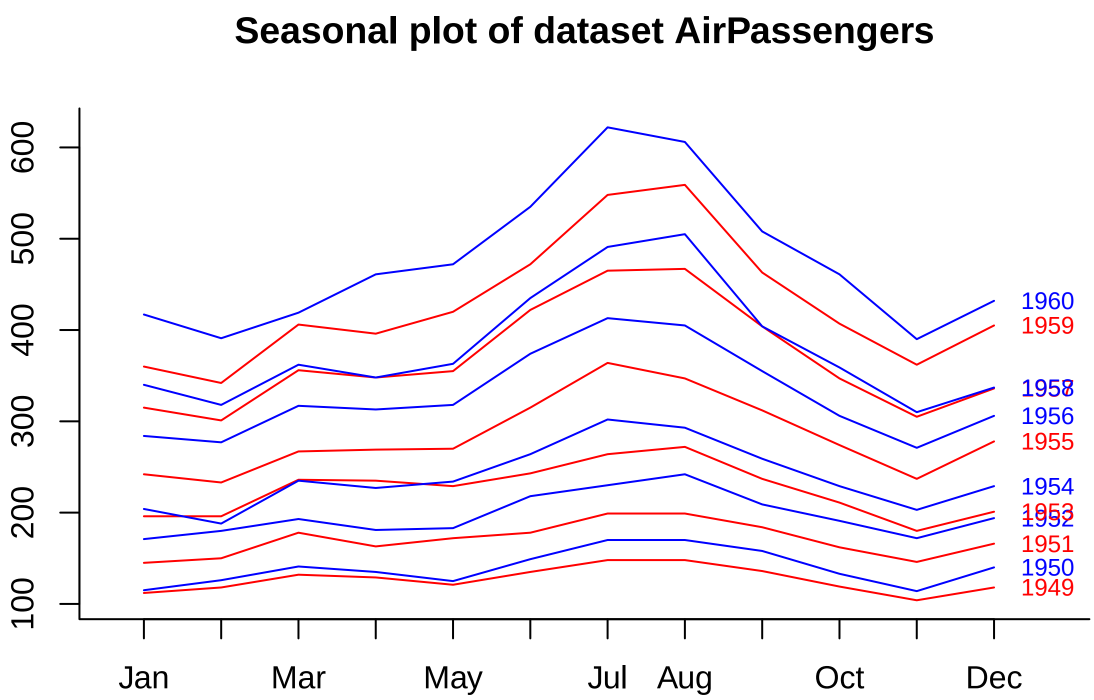

```
ggmonthplot(nottem)
```



3.4 Exercise seasonplot()

Data used = AirPassengers

```
#library(forecast)

seasonplot(AirPassengers, xlab="",
           col=c("red", "blue"),
           year.labels = T,
           labelgap = 0.35,
           type = "l",
           bty = "l",
           cex = 0.75,
           main = "Seasonal plot of dataset AirPassengers")
```



4 Importing time Series Data from Excel or Other Sources

Thing to keep in mind:

- No headers or row IDs => Make sure you have only raw data imported!
- Data needs to be sorted => It needs to be continuous series of data, earlier data comes first and latest last. For Example: It needs to start with 2008 and end with 2017.

4.1 Example

data: https://www.statbureau.org/en/germany/inflation-tables

```
#mydata = scan()
```

```
#plot.ts(mydata)
```

```
#germaninfl = ts(mydata, start=2008, frequency =12)
```

```
#plot(germaninfl)
```

5 Irregular Time Series

The interval between observations is not fixed.

Reasons behing irregular time series:

- Result of inappropriate data collection
- Hardware or software errors
- The nature of the data is irregular (e.g. logs)

Most modeling techniques require a regular time series.

Most of the tools are not able to handle differing gaps between the observations.

Possible solution:

- Min. 1 observation per unit
- Some info will be lost

5.1 Import a new dataset

```
#library(readr)
irregular_sensor <- read_csv("~/Downloads/irregular-sensor.csv",
    col_names = FALSE)
```

```
## Parsed with column specification:
## cols(
##   X1 = col_character(),
##   X2 = col_double()
## )
```

```
class(irregular_sensor$X1)
```

```
## [1] "character"
```

First column is classified as character but represents data. The hour when measurement was taken fluctuates a lot - no fixed interval. Some observations include only one record per day, some include more than 1.

1) We have to convert character to a date time format

2) Regularizing the dataset with an aggregate function - aggregate data into daily observations

3) Convertig the object into a time series(ts)

Summary:

- Specifying a time window
- Moderate amount of N/As or missing data:
  - Apply a favored missing data imputation method
- High amount of N/As or missing data:
  - Readjust the time window
- Class 'zoo' for irregular time series - library 'zoo'

```
#library(zoo)
#library(tidyr)
```

5.2 Method 1 - removing the time component

```
irreg.split = separate(irregular_sensor, col=X1, into = c('date', 'time'),
sep = 8, remove=T)
```

```
# Using only the date
sensor.date = strptime(irreg.split$date, '%m/%d/%y')
```

```
# Creating a data frame for orientation
irregts.df = data.frame(date=as.Date(sensor.date),
                        measurement = irregular_sensor)
```

```r
# Using zoo package
irreg.dates = zoo(irregts.df$measurement.X2,
order.by = irregts.df$date)
```

```r
ag.irregtime = aggregate(irreg.dates, as.Date, mean)

ag.irregtime
```

```
## 2017-05-16 2017-05-17 2017-05-18 2017-05-19 2017-05-20 2017-05-21
##   334.5000   439.2000   349.2000   345.2000   419.5000   352.9000
## 2017-05-22 2017-05-23 2017-05-24 2017-05-25 2017-05-26 2017-05-27
##   372.2000   402.4500   309.5000   382.0000   432.6000   392.6000
## 2017-05-28 2017-05-29 2017-05-30 2017-05-31
##   391.0000   405.0000   369.9500   338.2333
```

```r
length(ag.irregtime)
```

```
## [1] 16
```

Dataset is now regular and can be converted to ts().

5.3 Method 2 - date and time component kept

```r
sensor.date1 = strptime(irregular_sensor$X1, '%m/%d/%y %I:%M %p')

sensor.date1
```

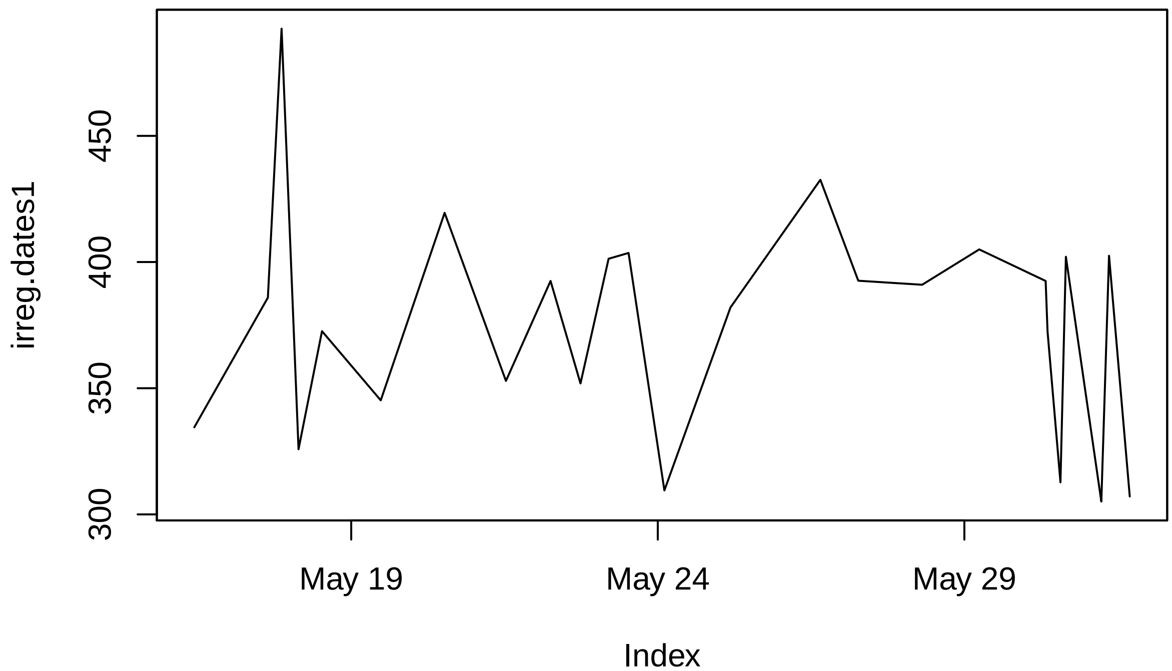
```
##  [1] "2017-05-16 10:34:00 GMT" "2017-05-17 15:23:00 GMT"
##  [3] "2017-05-17 20:45:00 GMT" "2017-05-18 03:23:00 GMT"
##  [5] "2017-05-18 12:34:00 GMT" "2017-05-19 11:34:00 GMT"
##  [7] "2017-05-20 12:34:00 GMT" "2017-05-21 12:34:00 GMT"
##  [9] "2017-05-22 17:45:00 GMT" "2017-05-22 06:02:00 GMT"
## [11] "2017-05-23 04:45:00 GMT" "2017-05-23 12:34:00 GMT"
## [13] "2017-05-24 02:35:00 GMT" "2017-05-25 04:27:00 GMT"
## [15] "2017-05-26 15:39:00 GMT" "2017-05-27 06:29:00 GMT"
## [17] "2017-05-28 07:29:00 GMT" "2017-05-29 05:49:00 GMT"
## [19] "2017-05-30 07:49:00 GMT" "2017-05-30 08:34:00 GMT"
## [21] "2017-05-30 13:37:00 GMT" "2017-05-30 15:45:00 GMT"
## [23] "2017-05-31 05:37:00 GMT" "2017-05-31 08:38:00 GMT"
## [25] "2017-05-31 16:45:00 GMT"
```

```r
irreg.dates1 = zoo(irregular_sensor$X2,
order.by = sensor.date1)

irreg.dates1
```

```
## 2017-05-16 10:34:00 2017-05-17 15:23:00 2017-05-17 20:45:00
##               334.5               385.9               492.5
## 2017-05-18 03:23:00 2017-05-18 12:34:00 2017-05-19 11:34:00
##               325.8               372.6               345.2
## 2017-05-20 12:34:00 2017-05-21 12:34:00 2017-05-22 06:02:00
##               419.5               352.9               392.5
## 2017-05-22 17:45:00 2017-05-23 04:45:00 2017-05-23 12:34:00
##               351.9               401.3               403.6
## 2017-05-24 02:35:00 2017-05-25 04:27:00 2017-05-26 15:39:00
```

```
##                309.5               382.0               432.6
## 2017-05-27 06:29:00 2017-05-28 07:29:00 2017-05-29 05:49:00
##                392.6               391.0               405.0
## 2017-05-30 07:49:00 2017-05-30 08:34:00 2017-05-30 13:37:00
##                392.5               372.5               312.7
## 2017-05-30 15:45:00 2017-05-31 05:37:00 2017-05-31 08:38:00
##                402.1               305.1               402.5
## 2017-05-31 16:45:00
##                307.1
```

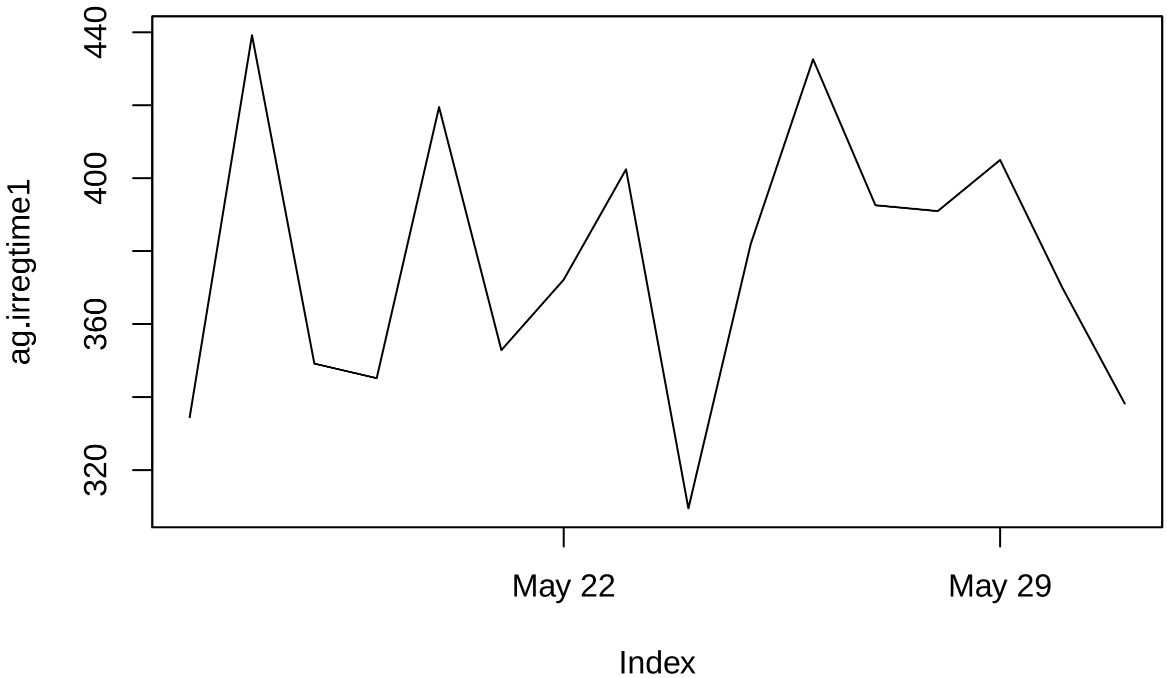
```
plot(irreg.dates1)
```



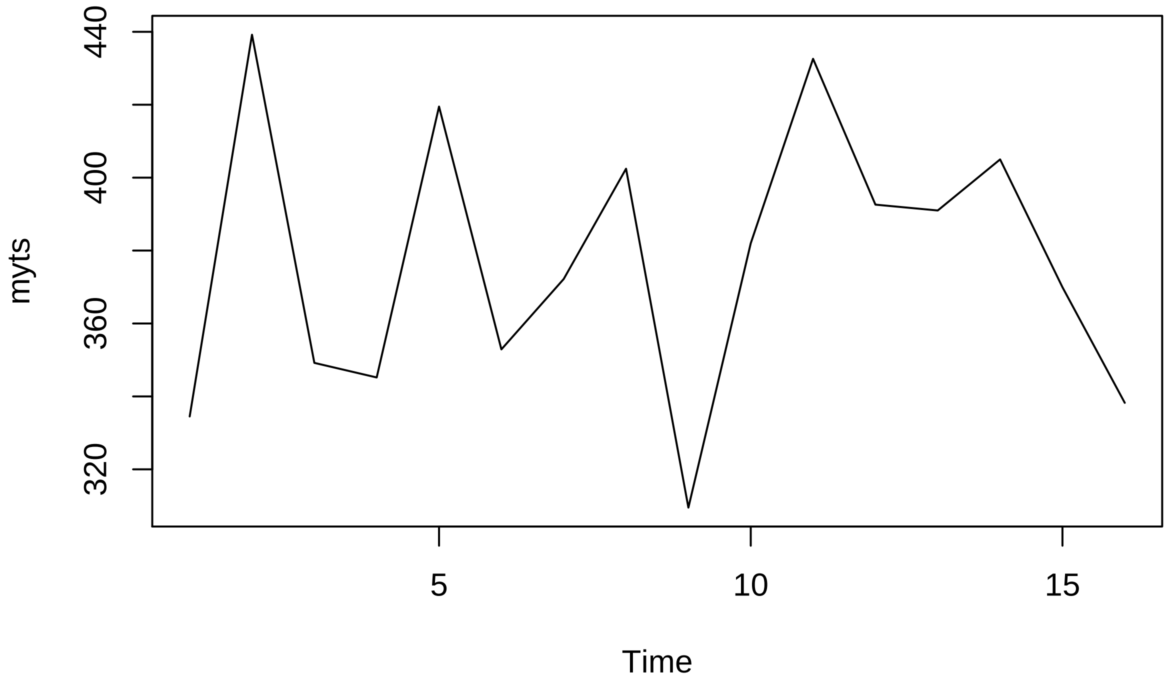

```
ag.irregtime1 = aggregate(irreg.dates1, as.Date, mean)

ag.irregtime1
```

```
## 2017-05-16 2017-05-17 2017-05-18 2017-05-19 2017-05-20 2017-05-21
##   334.5000   439.2000   349.2000   345.2000   419.5000   352.9000
## 2017-05-22 2017-05-23 2017-05-24 2017-05-25 2017-05-26 2017-05-27
##   372.2000   402.4500   309.5000   382.0000   432.6000   392.6000
## 2017-05-28 2017-05-29 2017-05-30 2017-05-31
##   391.0000   405.0000   369.9500   338.2333
```

```
plot(ag.irregtime1)
```




```r
myts = ts(ag.irregtime1)
plot(myts)
```



6 Working with Missing Data and Outliers

6.1 Import ts.NAandOutliers.csv

```
#library(readr)

mydata <- read_csv("~/Downloads/ts-NAandOutliers.csv",
col_names = TRUE,
cols(
  X1 = col_integer(),
  mydata = col_double()
))
```

6.2 Convert the 2nd column to a simple ts without frequency

```
myts = ts(mydata$mydata)
myts
```

```
## Time Series:
## Start = 1
## End = 250
## Frequency = 1
##   [1]  32.801464  42.465485         NA  32.204058  55.557647  33.050864
##   [7]  43.401620  37.768318  22.844180  36.428877  28.496485  59.037881
##  [13]  36.544163  26.668135  41.325626  28.913199  38.595417  31.341447
##  [19]  34.547023         NA  30.499324  49.391323  43.976004  22.162741
##  [25]  19.439525  41.892407  30.321857  32.899878  17.686235  10.332791
##  [31]  31.612958  40.011275  35.378517  46.167222  26.903207  36.304821
##  [37]  23.408770  42.785841  31.919674  37.571226  33.907485  17.698917
##  [43]  19.931775  23.971169 999.000000  32.853670  33.012320  47.893249
##  [49]  33.961104  40.826518  34.389579  27.210322  41.815827         NA
##  [55]  49.711080  37.246486  34.472507  27.554913  37.976930  24.503481
##  [61]  33.941547  28.582326  17.945402  40.335543  32.103075  15.609346
##  [67]  38.637130  58.877558  42.178769  34.075469  29.208206  20.409934
##  [73]  23.682860  49.014566  59.160903  24.994359  37.321672  11.830421
##  [79]  49.907975  33.288427  25.900307  34.661099  38.170951  30.246685
##  [85]  45.001326  36.082827  38.969588  24.260726   8.619401  33.933167
##  [91]  30.158056  32.211135  46.688584  36.399098  27.266510  39.706101
##  [97]  48.560701 999.000000  31.011612  33.565184  41.850476  45.780926
## [103]  21.679404  32.340497  55.904896  17.349895  32.994516  36.155426
## [109]  47.089342  33.955275  36.563838  18.773382  28.077605  40.483324
## [115]  41.341771  32.907839  59.604911  20.989279  46.886734  53.931163
## [121]  44.662468  43.125045  25.800244  22.833920  51.397357  34.775922
## [127]  50.922532  36.430258  32.975690  37.659017  48.006323  49.901919
## [133]  20.619643  24.895206   4.682543  17.049461  45.618543  28.288209
## [139]  50.446258  34.983971  38.847283  32.301493  46.044574  21.739473
## [145]  16.457915  36.157602  35.773314  23.368300  34.220736  39.443674
## [151]  26.074044  28.599269  45.410516         NA  30.585004  23.405284
## [157]  36.949438  17.647508  22.991044  40.388899  30.654440  49.261182
## [163]  31.215505  30.462442  41.294423  28.046393  24.925970  23.934094
## [169]  41.690112  46.226476  40.741721 999.000000  26.455048  12.766929
## [175]  38.133315  61.653241  11.474054  41.835335  34.572898  59.921615
```

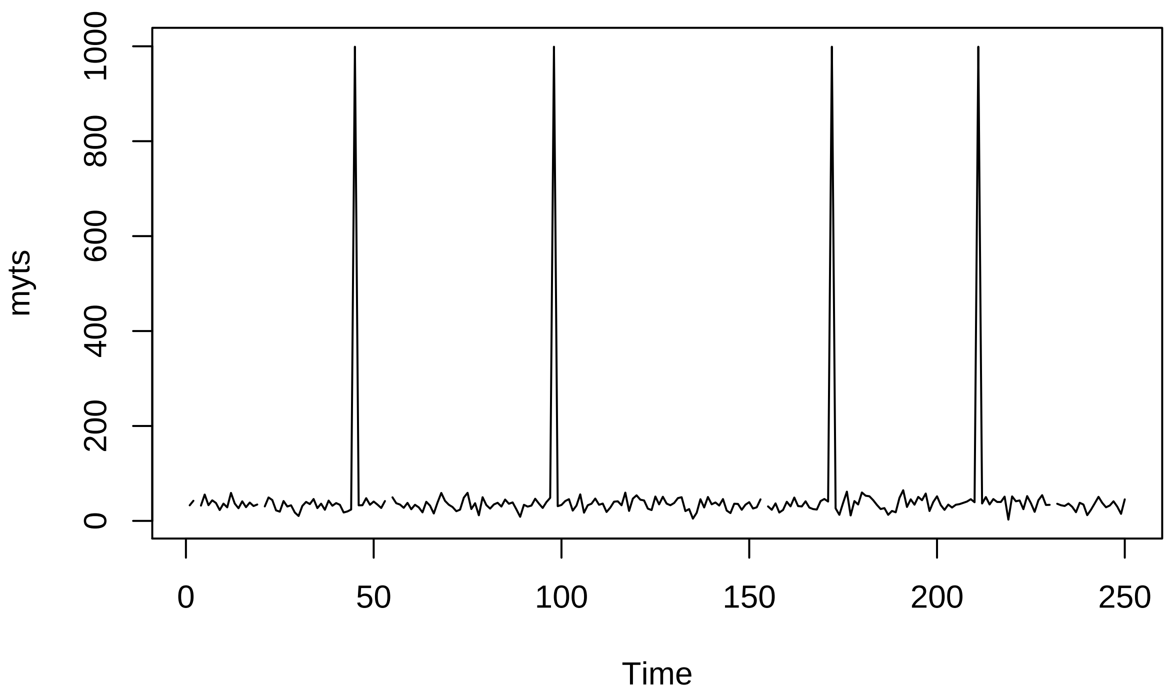
```
## [181]  53.072688  51.889866  43.700888  33.639492  24.988901  27.019376
## [187]  12.774882  21.001551  18.133113  48.563807  64.633075  29.362668
## [193]  45.478245  34.152629  50.573869  43.564419  57.644084  20.785408
## [199]  39.811707  51.854335  33.351763  23.242107  34.351879  28.113792
## [205]  33.952911  35.372668  38.059841  40.818440  45.768335  39.272128
## [211] 999.000000  36.665537  50.282893  34.561040  46.040830  39.936308
## [217]  39.873144  51.126396   2.683472  51.667975  41.336229  43.090450
## [223]  24.686842  52.300908  37.379943  19.043254  43.512121  54.236360
## [229]  33.557160  33.851597         NA  36.149460  32.985037  31.422766
## [235]  36.574851  29.648483  18.290954  38.154075  34.446452  12.037743
## [241]  23.581530  36.968395  50.747174  37.981389  28.693203  32.396009
## [247]  41.325484  30.017571  14.818111  45.403854
```

Regular time series with 250 observations. The sensor showed some malfunctions: Missing measurements, measurement is out of range.

```
summary(myts)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's 
##   2.683  28.080  34.570  50.710  42.470 999.000       5
```

```
plot(myts)
```



Plot confirmed that some values (4) are out of range (outliers).

This counts for 3.6% corupted observations (9 of 250).

Outlier detection:

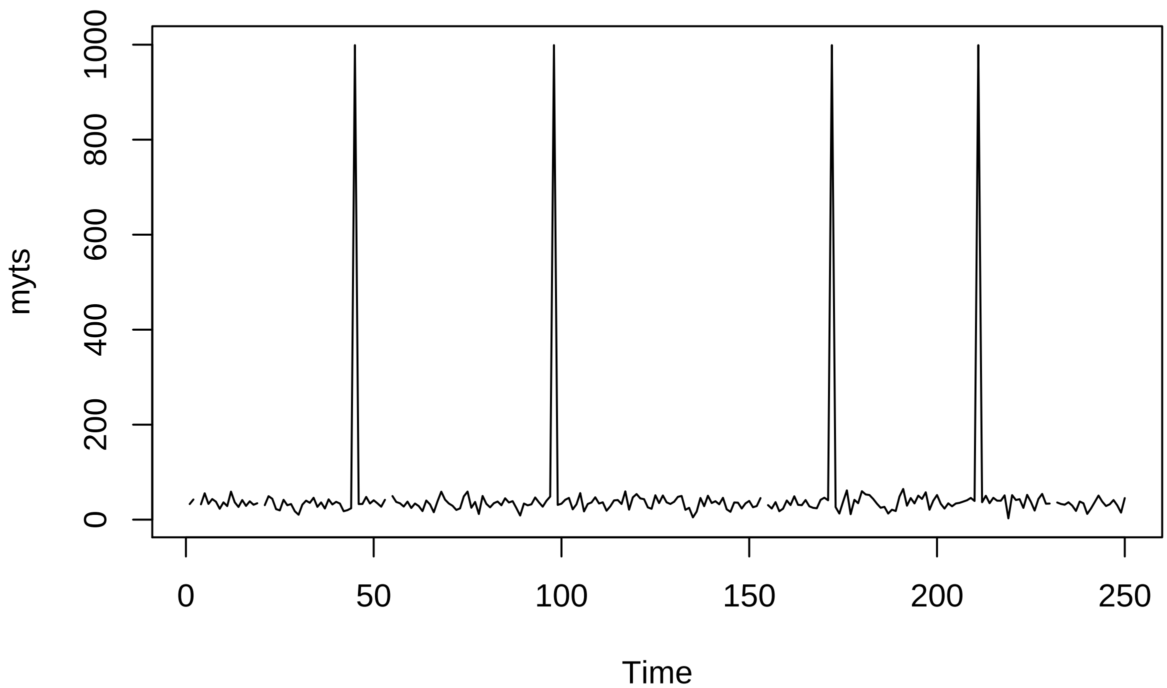
- library 'tsoutliers': tso()
- library 'forecast': tsoutliers()

6.3 Automatic detection of outliers

```r
#library(forecast)

myts1 = tsoutliers(myts)

myts1
```

```
## $index
## [1]  45  98 172 211
##
## $replacements
## [1] 28.41242 39.78616 33.59838 37.96883
```

```r
plot(myts)
```



Missing Data Imputation

- Adding a replacement value instead of themissing one
- Several available methods
- Libraries 'zoo' (na.locf()) and 'forecast'

na.locf() => last observation carried forward, last observation before themissing value will be copied and replaced missing value.

NAfill() => missing values are filled with a specific value (manually selected)

6.4 Missing data handling with zoo

```r
#library(zoo)
myts.NAlocf = na.locf(myts)
```

```r
myts.NAfill = na.fill(myts, 33)
# Tip: na.trim to get rid of NAs at the beginning or end of dataset
```

Another method is na.interp() which fits a local linear interpolation for a given missing value.

If the given dataset is seasonal, the interpollation is based on exponential smoothing.

6.5 Standard NA method in package forecast

```r
myts.NAinterp = na.interp(myts)
```

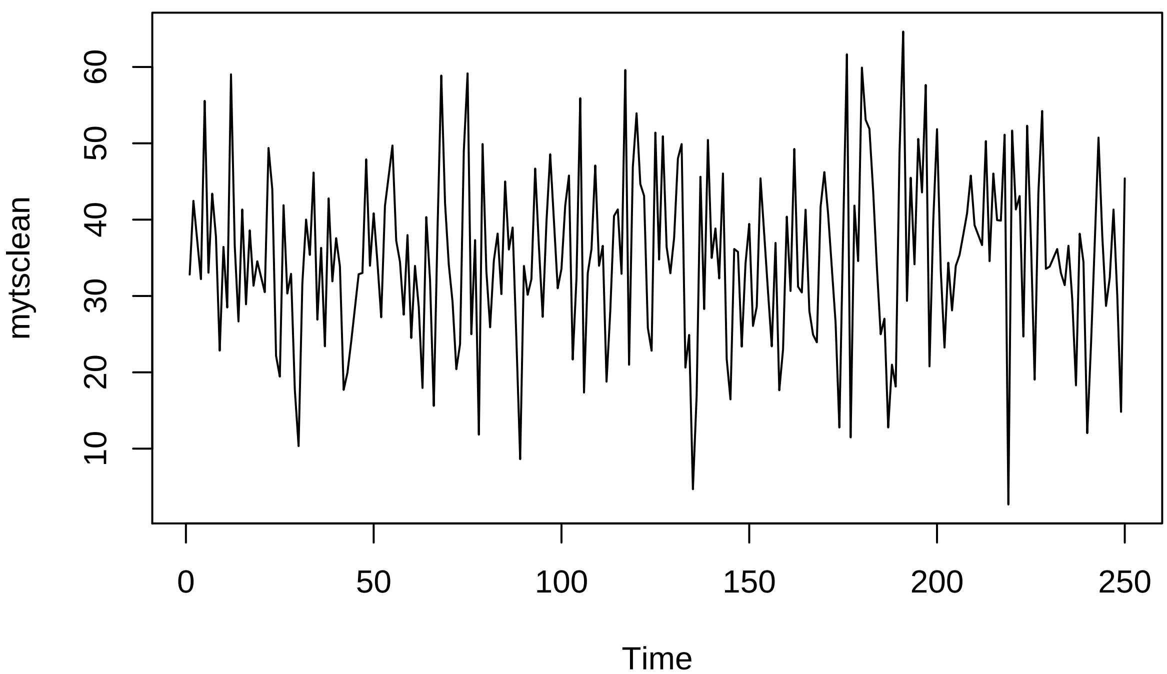
na.interp() function and tsoutliers() function is combined into one convenient function which automatically updates the data set (tsclean()).

The missing values of field and the outliers are replaced with locally smoothed values.

These are the same values as you would get with tsoutliers function.

6.6 Cleaning NA and outliers with forecast package

```r
mytsclean = tsclean(myts)

plot(mytsclean)
```




```r
summary(mytsclean)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
##   2.683  28.160  34.570  35.030  41.830  64.630
```

7 Time Series Vectors and Lags

7.1 Difference between time series and vector

Time series data contains values as other datasets do, but its values have a specific order

The order is specified by the tie stamp - normally, data is collected on yearly, montyhly etc. basis.

The order might also be specified by a vector which has a unique ID attached to the variable.

To perform time series analysis the order needs to be meaningful without randomness

- Non-meaningful: a vector of body measurements taken randomly from a given population
  - The order of people being measured doesn't provide meaningful information
  - Changing the order of people doesn't affect the basis assuptions
- Meaningful: monthly temperature measurements
  - Seasonal pattern
  - Changing the order of values corrupts the information
- Alternative to time stamp: vector of unique IDs attached to a vector
  - Can be coded in R, however, proper time series is easiar to read

7.2 Time lag

lagn = yt - yt-n

lynx has a length of 114

t=shows position of a value in the time series last observation = y114 y114 = 3396

A calculation of lag1:

lag1 = y114 - y114-1 = 3396 - 2657

A calculation of lag2:

lag2= y114 - y114-2 = 3396 - 1590

```
lynx
```

```
## Time Series:
## Start = 1821
## End = 1934
## Frequency = 1
##   [1]  269  321  585  871 1475 2821 3928 5943 4950 2577  523   98  184  279
##  [15]  409 2285 2685 3409 1824  409  151   45   68  213  546 1033 2129 2536
##  [29]  957  361  377  225  360  731 1638 2725 2871 2119  684  299  236  245
##  [43]  552 1623 3311 6721 4254  687  255  473  358  784 1594 1676 2251 1426
##  [57]  756  299  201  229  469  736 2042 2811 4431 2511  389   73   39   49
##  [71]   59  188  377 1292 4031 3495  587  105  153  387  758 1307 3465 6991
##  [85] 6313 3794 1836  345  382  808 1388 2713 3800 3091 2985 3790  674   81
##  [99]   80  108  229  399 1132 2432 3574 2935 1537  529  485  662 1000 1590
## [113] 2657 3396
```

```
time(lynx)
```

```
## Time Series:
## Start = 1821
## End = 1934
## Frequency = 1
```

```
##   [1] 1821 1822 1823 1824 1825 1826 1827 1828 1829 1830 1831 1832 1833 1834
##  [15] 1835 1836 1837 1838 1839 1840 1841 1842 1843 1844 1845 1846 1847 1848
##  [29] 1849 1850 1851 1852 1853 1854 1855 1856 1857 1858 1859 1860 1861 1862
##  [43] 1863 1864 1865 1866 1867 1868 1869 1870 1871 1872 1873 1874 1875 1876
##  [57] 1877 1878 1879 1880 1881 1882 1883 1884 1885 1886 1887 1888 1889 1890
##  [71] 1891 1892 1893 1894 1895 1896 1897 1898 1899 1900 1901 1902 1903 1904
##  [85] 1905 1906 1907 1908 1909 1910 1911 1912 1913 1914 1915 1916 1917 1918
##  [99] 1919 1920 1921 1922 1923 1924 1925 1926 1927 1928 1929 1930 1931 1932
## [113] 1933 1934
```

```r
length(lynx)
```

```
## [1] 114
```

```r
tail(lynx) # last 6 observations
```

```
## Time Series:
## Start = 1929
## End = 1934
## Frequency = 1
## [1]  485  662 1000 1590 2657 3396
```

7.3 Univariate and multivariate time series, mean and median

Univariate time series - There is one variable attached to a time stamp

Multivariate time series - several variables are connected to the time stamp (matrix)

For univariate time series (e.g. lynx) there are very common statistics (e.g. mean, median) Mean and median differ, when we look at plot, we can see that there are several peaks in the data - cycles of approx. 9 years. The peaks are short => most of the observation are well below the peaks

The high peaks affect the average value of the vector.

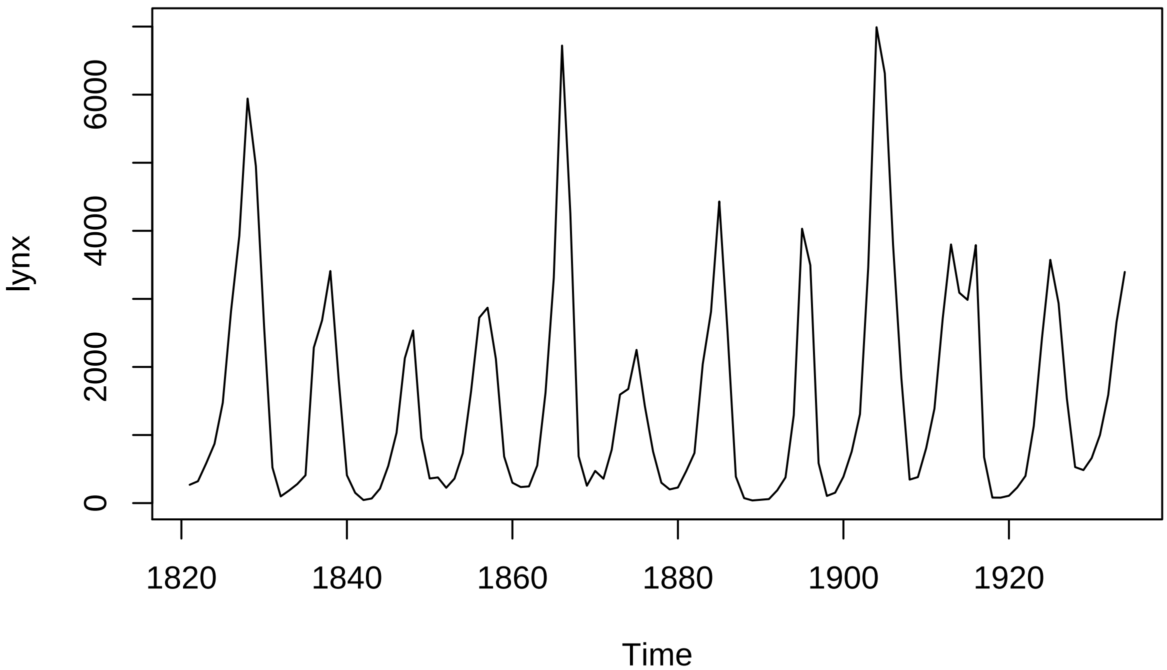
The peaks have no effect on the median.

```r
mean(lynx); median(lynx)
```

```
## [1] 1538.018
```

```
## [1] 771
```

```r
plot(lynx)
```



7.4 How is median calculated in even dataset??

```r
sort(lynx)
```

```
##   [1]   39   45   49   59   68   73   80   81   98  105  108  151  153  184
##  [15]  188  201  213  225  229  229  236  245  255  269  279  299  299  321
##  [29]  345  358  360  361  377  377  382  387  389  399  409  409  469  473
##  [43]  485  523  529  546  552  585  587  662  674  684  687  731  736  756
##  [57]  758  784  808  871  957 1000 1033 1132 1292 1307 1388 1426 1475 1537
##  [71] 1590 1594 1623 1638 1676 1824 1836 2042 2119 2129 2251 2285 2432 2511
##  [85] 2536 2577 2657 2685 2713 2725 2811 2821 2871 2935 2985 3091 3311 3396
##  [99] 3409 3465 3495 3574 3790 3794 3800 3928 4031 4254 4431 4950 5943 6313
## [113] 6721 6991
```

```r
sort(lynx)[c(57,58)]
```

```
## [1] 758 784
```

```r
quantile(lynx)
```

```
##      0%     25%     50%     75%    100%
##   39.00  348.25  771.00 2566.75 6991.00
```

```r
#50% quantile shows median
quantile(lynx, prob = seq(0, 1, length = 11), type = 5)
```

```
##     0%    10%    20%    30%    40%    50%    60%    70%    80%    90%
##   39.0  146.7  259.2  380.5  546.6  771.0 1470.1 2165.6 2818.0 3790.4
##   100%
## 6991.0
```

8 Simple forecast methods

- Time series analysis is a statistical effort that implies that datasets have different statistical traits
- These statistical traits are very distinct from standard data without a time component
- The time component specifies successive order
- To select the right modelling tool, you need to know the characterstics of the dataset

8.1 Type of datasets

- *Random normally distributed dataset* (Stationary dataset; mean stays constant throughout the time series, variance stays constant throughout the time series)
- *Heteroscedastic dataset* - the data points at the end have a much larger span than at the beginning, variance changes; non-stationary, mean stays the same
- *Dataset with trend* - looks like a linear regression, trend is clearly present, mean is not constant, variance is constant, non-stationary
- *Seasonal dataset* - Typical time intervals resulting in spikes, they have same distance to each other, variance and mean stay the same
- *Exponential Trend* - clear trend with exponential curve, mean is non constant, non-stationary dataset
- *All Traits Present* - Seasonality, changing variance trend might be also present (more complicated model)

8.2 Why to choose a simple method over a complex model?

- Simple forecasting methods can outperform more advance models in certain circumstances
- General rules:
- For mostly or completely random data simple methods work best,
- Advanced models exploit patterns (e.g. seasonality, trend) better
- With stock data or financial data primitive models do well

8.3 Three simple methods

1) Naive method

- Naive, last observation carried forward method
- Projects the last observation into the future
- Use the naive() function in the 'forecast' package
- The function can be tweaked to fit even a seasonal dataset
- example: to forecast February 2018, R takes the last observed value of February 2017

2) Average method

- Calculates the mean of the data and projects that into the future
- Use the meanf() function from the 'forecast' package

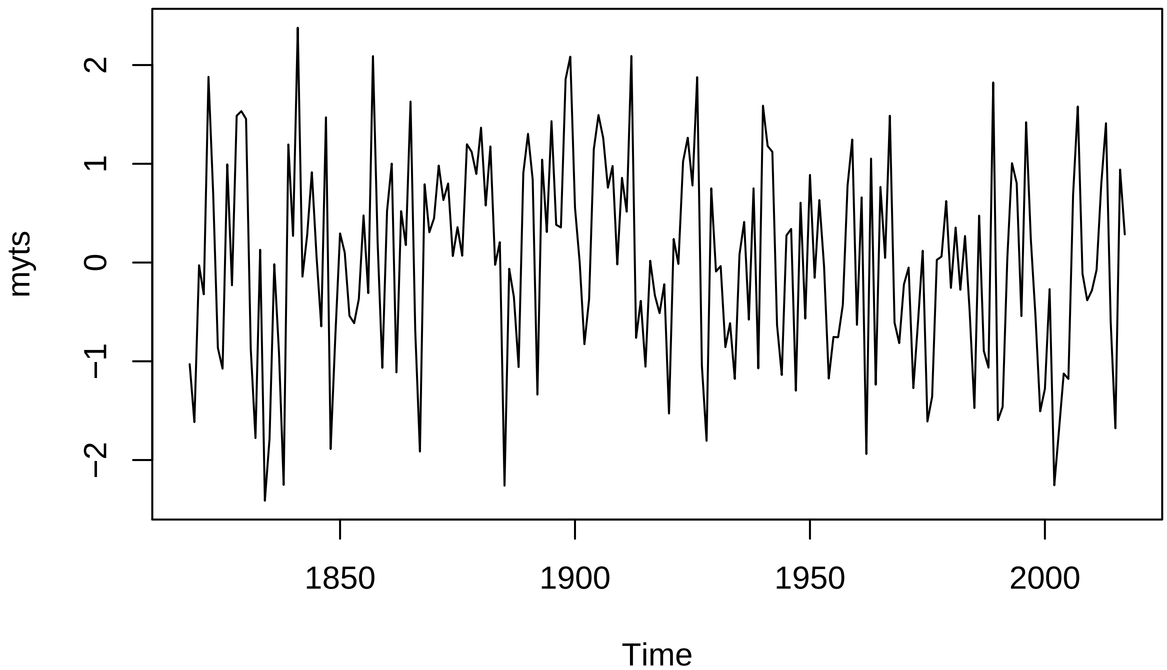
3) Drift method

- Calculated the difference between first and last observation and carries that increase into the future

- Use the rwf() function from ‘forecast’

8.4 Example of simple methods

```
set.seed(95)
myts <- ts(rnorm(200), start = (1818))
plot(myts)
```




```
#no pattern in dataset - I can use one of simple methods
```

```
# library(forecast)
meanm <- meanf(myts, h=20) #20 means that I forecast 20 years
naivem <- naive(myts, h=20)
driftm <- rwf(myts, h=20, drift = T) #drift is set to true for drift method
```

meanm: I can open a created object and see available elements such as fitted values (predicted values applied to a dataset), residuals or mean values (the forecasted values of the same mean)

Plot() the meanm object (20 forecast values)

- main “”' deletesthe header
- plot.conf = F allows to get more lines on the plot

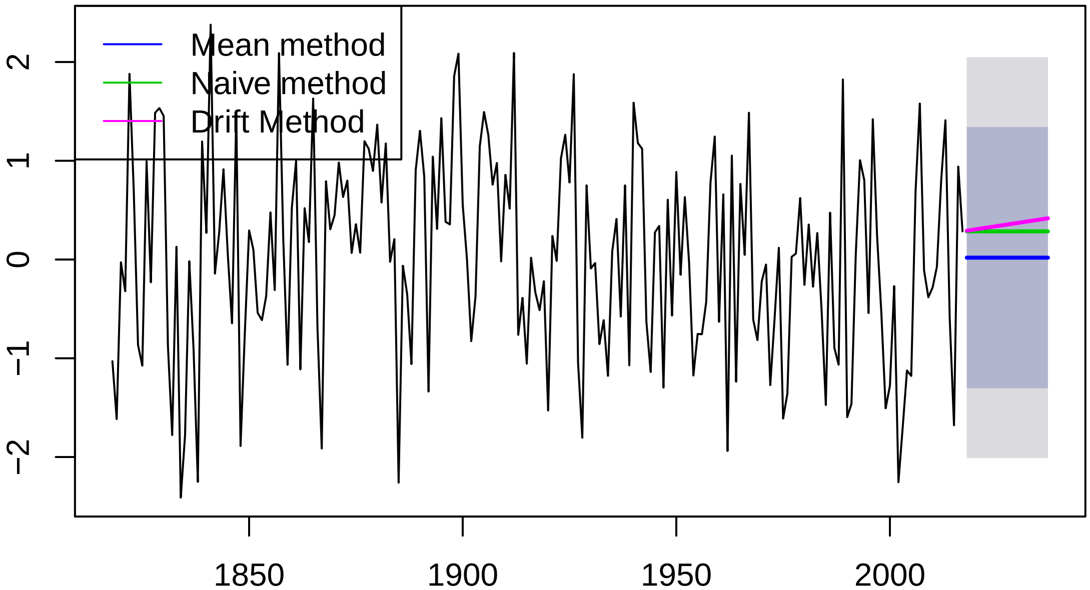
Lines() adds more lines to the plot

- naivem$meananddifrtm$mean prove the 20 forecast values of each object
- col sets the colour for the lines
- lwd sets the line width

Legend with the same colour coding

- topleft - position of the legend
- lty sets the line type
- legend specifies the titles

```r
plot(meanm, plot.conf = F, main = "")
lines(naivem$mean, col=123, lwd = 2)
lines(driftm$mean, col=22, lwd = 2)
legend("topleft",lty=1,col=c(4,123,22),
       legend=c("Mean method","Naive method","Drift Method"))
```




```r
#Blue line is mean as forecasted value

#Green line is last observation carried forward  (Random value most likely between -1 and 1)

#Purple line is first and last observations extrapolated into the future
```

8.5 Accuracy and model comparison

- Knowing which models perform best with the given data is key => forecast accuracy
- Determine how much difference thereis between the actual value and the forecast for that value
- The simplest way is via a scale dependent error - all the models you want to compare need to be on the same scale (e.g. MaE(Mean absolute error), RMSE(Root mean squared error))

A) MAE - mean absolute error

- The mean of all differences between actual and forecaste absolute values

MAE = (∑ |yi-yhati|)/n

B) RMSA - root mean squared error

RMSA = square root of (∑ yi-yhati)^2/n

C) MASE - mean absolute scaled error

- Measure the forecast error compared to the error of a naive forecast

01 the model needs a lot of improvement

D) MAPE - mean absolute percentage error

- Measures the difference of forecast errors and divides it by the actual observation value

- Does not allow for 0 values
- Puts much more weight on extreme values and positive errors
- Scale independent - you can use it to compare a model on different datasets

E) AIC - Akaike information criterion

- common measure in forecasting, statistical modelling and machine learning
- It is great to compare the complexity of different models
- penelizes more complex models
- the lower the AIC score the better

8.5.1 Accuracy and model comparison - Examples

Package forecast - accuracy() function gives all relevant accuracy statistics except the AIC

Random time series of 200 years starting 1818

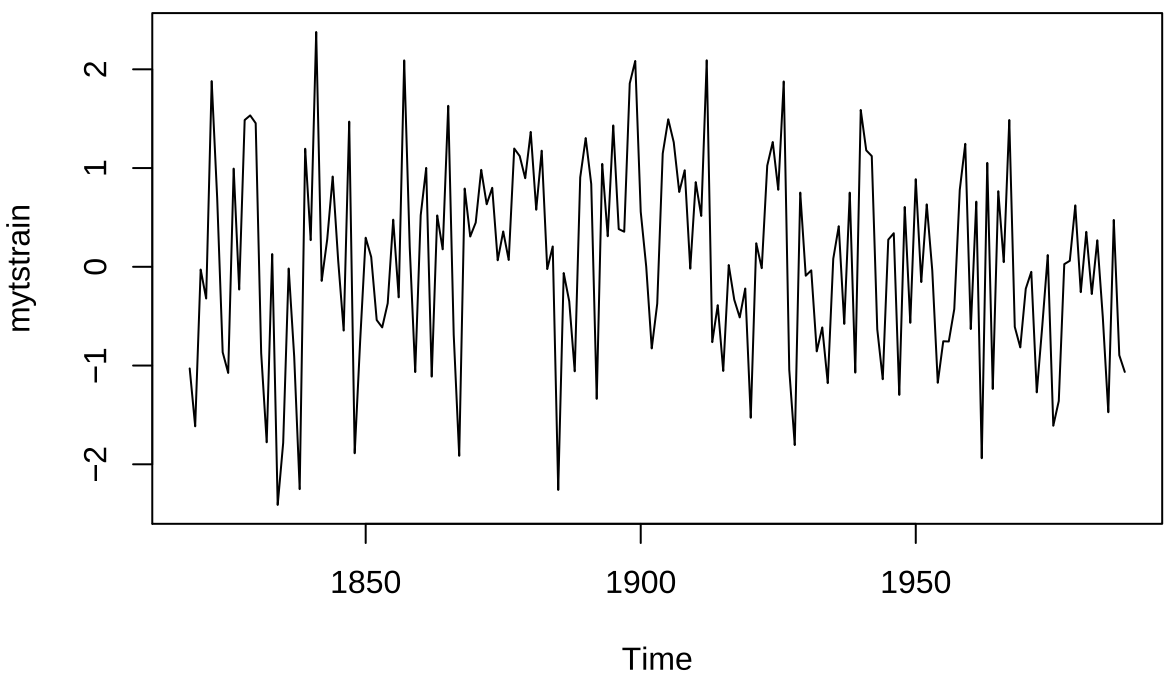
Dividing the series into a training and a test set using the windown() function (the training ata is used to fit the model mytstrain and the test set is used to seehow well the model performs)

The process of dividing int training and test datasets is widely used in machine learning and stastical modelling.

Always test the model on genuinely new data, because the performance of themodel on historic data is laregely irrelevant (split the data at about 80-20; 80% to fit the model and 20% to test it)

The model should be as simple as possible - complex models might cause overfitting.

```
set.seed(95)

myts <- ts(rnorm(200), start = (1818))

mytstrain <- window(myts, start = 1818, end = 1988)
# With the window() function we extract a time frame of 1818-1988

plot(mytstrain)
```



```r
#library(forecast)
meanm <- meanf(mytstrain, h=30)
naivem <- naive(mytstrain, h=30)
driftm <- rwf(mytstrain, h=30, drift = T)
```

```r
mytstest <- window(myts, start = 1988)

accuracy(meanm, mytstest)
```

```
##                         ME     RMSE       MAE       MPE     MAPE      MASE
## Training set  1.407307e-17 1.003956 0.8164571  77.65393 133.4892 0.7702074
## Test set     -2.459828e-01 1.138760 0.9627571 100.70356 102.7884 0.9082199
##                   ACF1 Theil's U
## Training set 0.1293488        NA
## Test set     0.2415939  0.981051
```

```r
accuracy(naivem, mytstest)
```

```
##                         ME     RMSE      MAE        MPE     MAPE     MASE
## Training set -0.0002083116 1.323311 1.060048 -152.73569 730.9655 1.000000
## Test set      0.8731935861 1.413766 1.162537   86.29346 307.9891 1.096683
##                    ACF1 Theil's U
## Training set -0.4953144        NA
## Test set      0.2415939  2.031079
```

```r
accuracy(driftm, mytstest)
```

```
##                         ME     RMSE      MAE        MPE     MAPE      MASE
## Training set -1.957854e-17 1.323311 1.060041 -152.64988 730.8626 0.9999931
## Test set      8.763183e-01 1.415768 1.163981   85.96496 308.7329 1.0980447
##                    ACF1 Theil's U
## Training set -0.4953144        NA
## Test set      0.2418493   2.03317
```

```r
#We have both arrows for both the training and the tests that are calculated the difference between the

#First model (meanm) shows the best results with all key indicators having the lowest values (RMSE, MAE
```

8.6 Forecast accuracy check

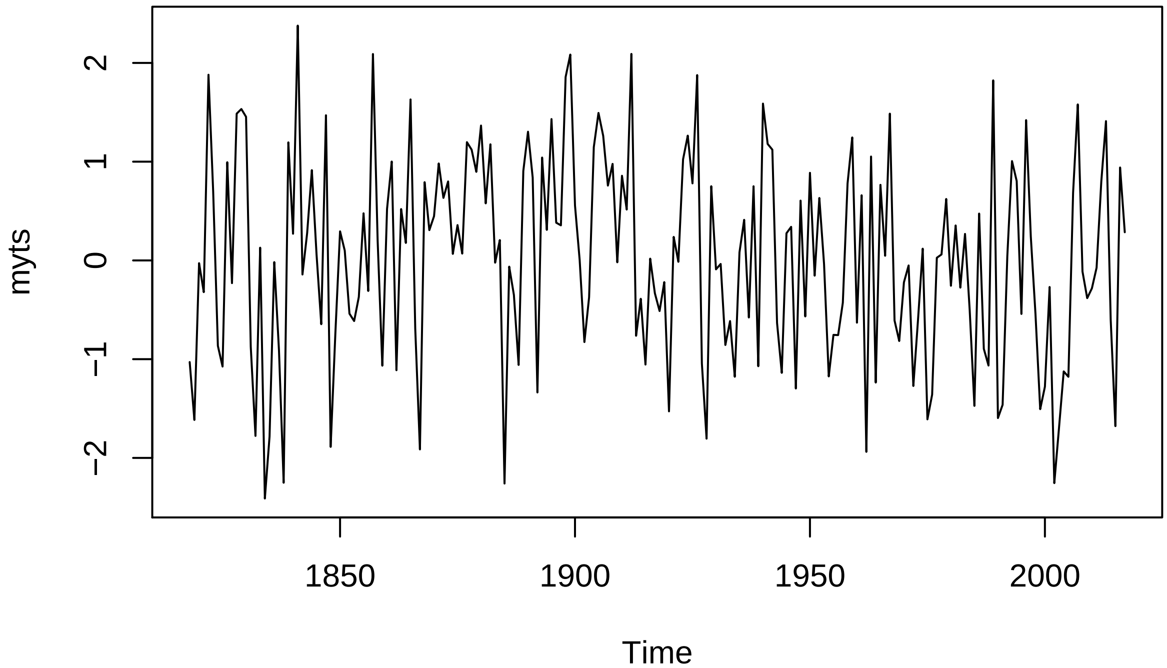
1) We generated 200 random numbers => myts
2) We took the first 170 observations (ca 80%) of myts using the window() function => mytstrain
3) We set up three forecasting models (average, naive and drift) on mytstrain to get 30 observations into the future => meanm, naivem, drftm
4) We extracted the last 30 observations (ca20%) of myts using the window() function => mytstest
5) We used the accuracy() function to see the error statistics of the hree models (meanm, naivem, drftm) compared to the error stastics of mystest

9 Residuals

- When modeling a series of data we get the residuals alongside with the forecasted and the fitted data
- Residuals is remaining data (leftovers) after modeling - they tell a lot about the quality of the model
- Rule of thumb: you want all the patterns in the model, only randomness should stay in the residuals

- Residuals shoud be the container of randomness (data you cannot explain in mathematical terms) => ideally they have a mean of zero and constant variance
- The residuals should be uncorrelated (correlated residuals still have information left in them) => ideally they are normally distributed
- A non-zero mean can be easily fixed with addition or subtraction, while correlations can be extracted via modeling tools (e.g differencing) - ensuring normal distribution (constant variance) might be impossible in some cases, however, transformations (e.g. logarithms) might help

```
set.seed(95)
myts <- ts(rnorm(200), start = (1818))
plot(myts)
```




```
#Use a random time series - ts(rnorm(200), start(1818)) - to work along
#Each of the three simple models have the residuals available $residuals
# Models drift and naive need one observation to start with => the first position variance cannot be co
# For these models residuals are what is left after the fitted value got subtracted from the original d
  # - The first value needs to be omitted = it is an N/A
  # - Residuals are depending on the dataset and the applied forecasting model - in this case they stay
```

```
#library(forecast)
meanm <- meanf(myts, h=20)
naivem <- naive(myts, h=20)
driftm <- rwf(myts, h=20, drift = T)


# Mean method
var(meanm$residuals) # Close to 1
```

```
## [1] 1.053807
```

```
mean(meanm$residuals) # Close to 0
```

```
## [1] -5.95498e-18
```

```r
mean(naivem$residuals) #Result NA
```

```
## [1] NA
```

```r
# Naive method (Getting rid off first observation)

naivwithoutNA <- naivem$residuals
naivwithoutNA <- naivwithoutNA[2:200]

var(naivwithoutNA)
```

```
## [1] 1.798592
```

```r
mean(naivwithoutNA)
```

```
## [1] 0.006605028
```

```r
# Drift method
driftwithoutNA <- driftm$residuals
driftwithoutNA <- driftwithoutNA[2:200]
var(driftwithoutNA)
```

```
## [1] 1.798592
```

```r
mean(driftwithoutNA)
```

```
## [1] -4.502054e-17
```

Use the functions var() and mean()

1.Mean method:

- variance is close to 1 (the random dataset has some unpredictability), mean is really close to 0
- this method works best with the dataset

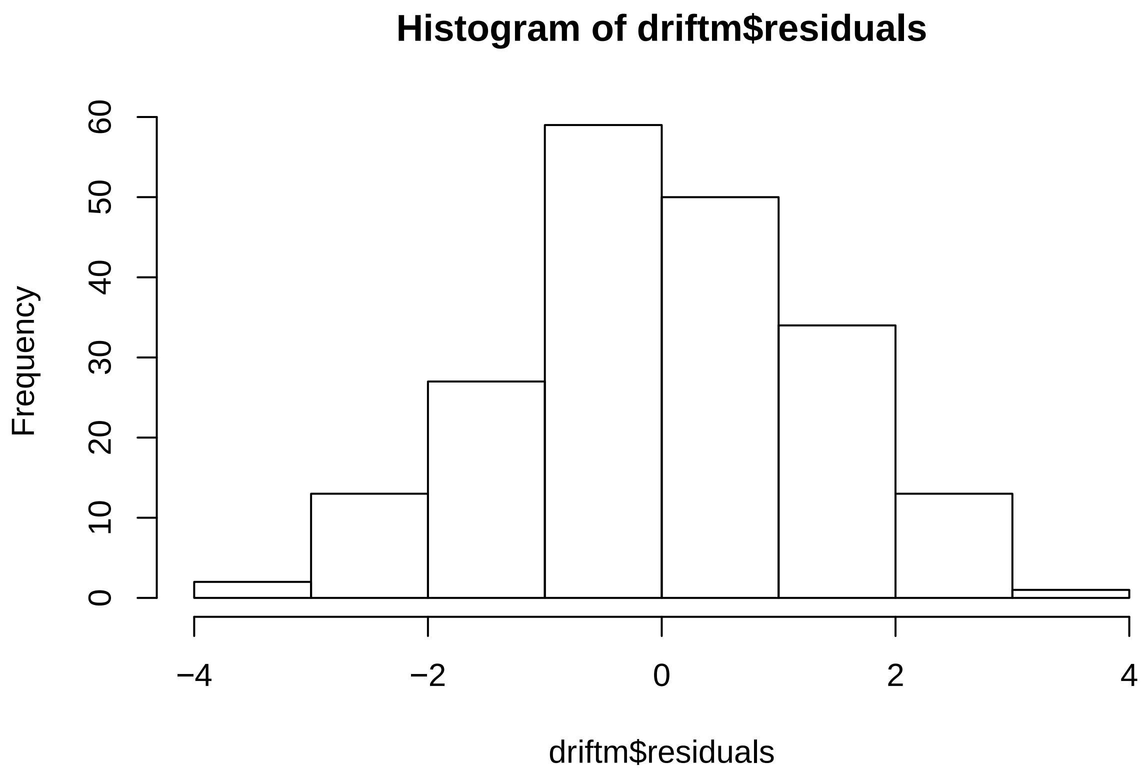
2. Naive method: first get rid of the very first value (N/A)

- Since the result depends on the last value observed, the variance (1.8) is quite off, while the mean(0.007) is moderately off ofthe original data

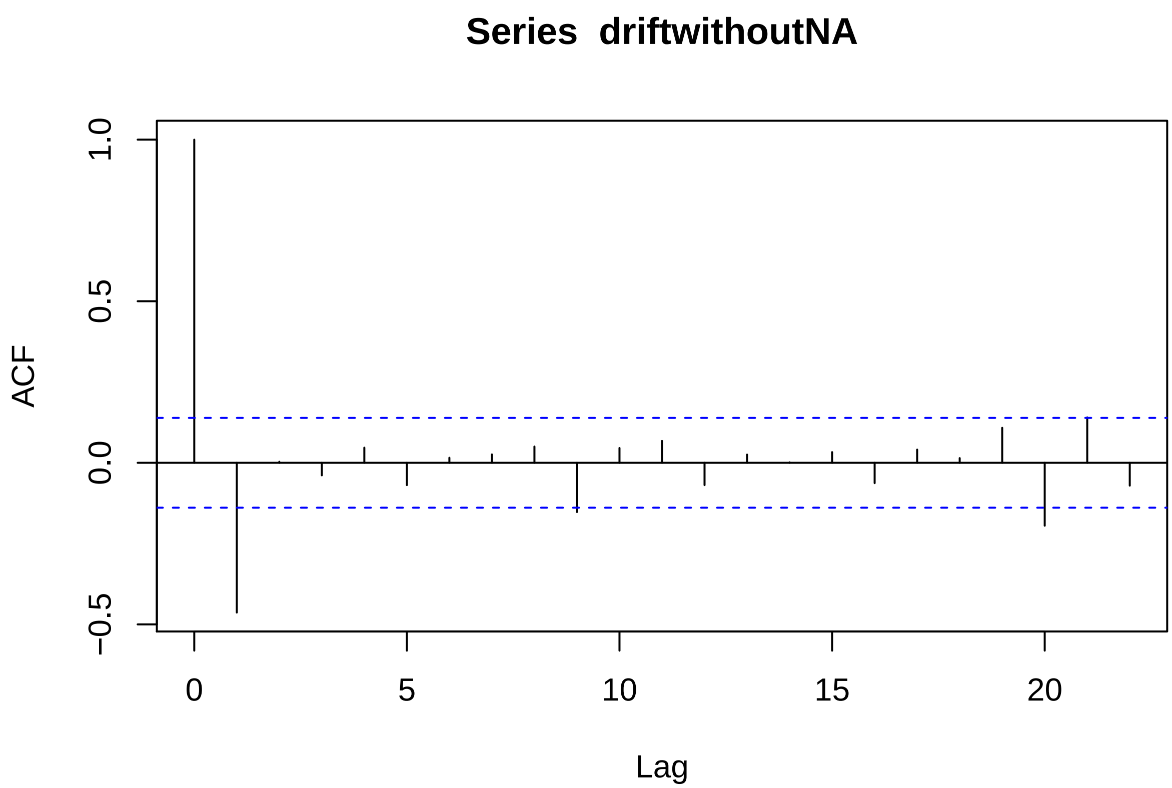
3. Drift method: first get rid of the very first value (N/A)

- Variance is significantly higher than 1, but the mean stays at 0, because the first and last observed values canceled each other out (random result)

```r
hist(driftm$residuals) # normal distribution of residuals
```




```r
acf(driftwithoutNA)
```




```r
# autocorrelation test, if we get several bars above or below the threshold levels, we get significance
# 3/20 bars are over/below the thresholds => the residuals still have information left in them
# Improving the model (e.g. applying a transformation) might reduce the bars
```

10 Stationarity

- Has the data the same statistical properties throughout the time series?
- Statistical propertis: variance, mean, autocorrelation
- Most analytical procedures in time series require stationary data
- If the data lacks stationarity there are transformations to be applied to make the data stationary, or it can be changed via differencing
- Differencing adjusts the data according to the time spans that differ in e.g. variance or mean (extensively used in ARIMA models)

10.1 De-trending

Loads of time series have a trend in it => the mean chanbges as a result of the trend => causes underestimated predictions

Solution:

1) Test if you get stationarity if you de-trend the dataset: take the trend compionent out of the dataset => trend stationarity
2) If this procedure is not enough then you can use differencing => difference stationarity
3) Unit-root tests tell whether there is a trend stationarity or a difference stationarity
   - The first difference goes from one period to the very next one (two succewssive steps)
   - The first difference is stationary and random => random walk (each value is a random step away from the previous value)
   - The first difference is stationary but not completely random (e.g. values are auto correlated) => requires a more sophisticated model (e.g. eponential smoothing, ARIMA)

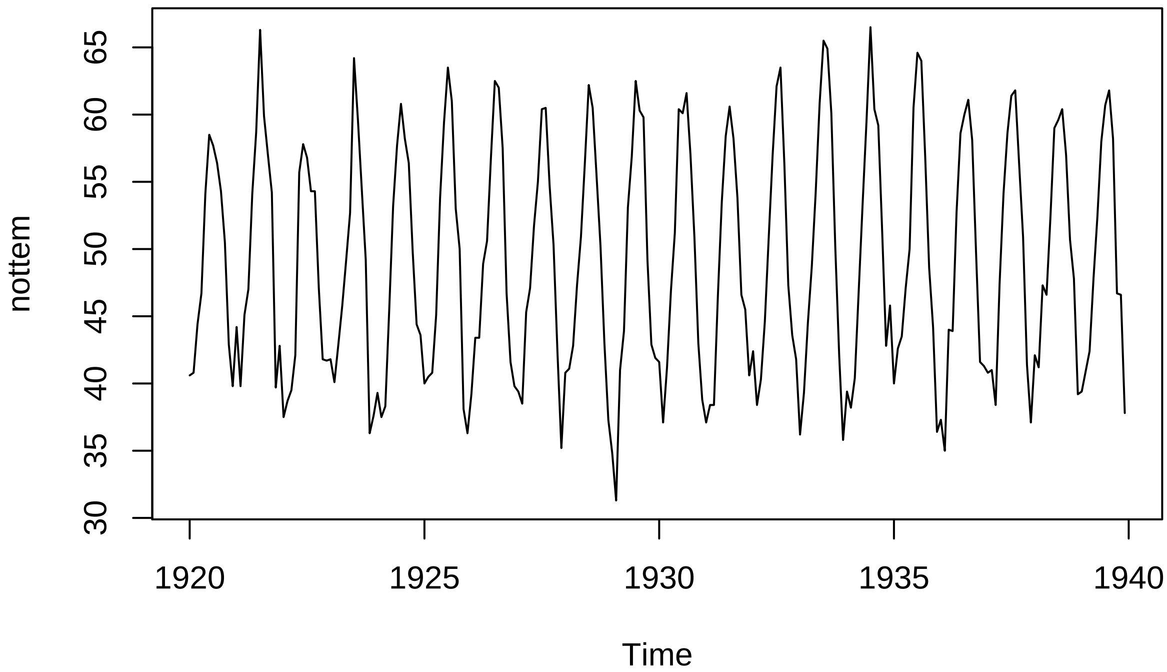
Tests for non-stationarity - (unit-root test), e.g. library urca - library tseries => adf.test(x); the augmented Dickey-Fuller test removes the autocorrelation and tests for non-stationarity

```
x <- rnorm(1000) # no unit-root, stationary

# library(tseries)

adf.test(x) # augmented Dickey Fuller Test
```

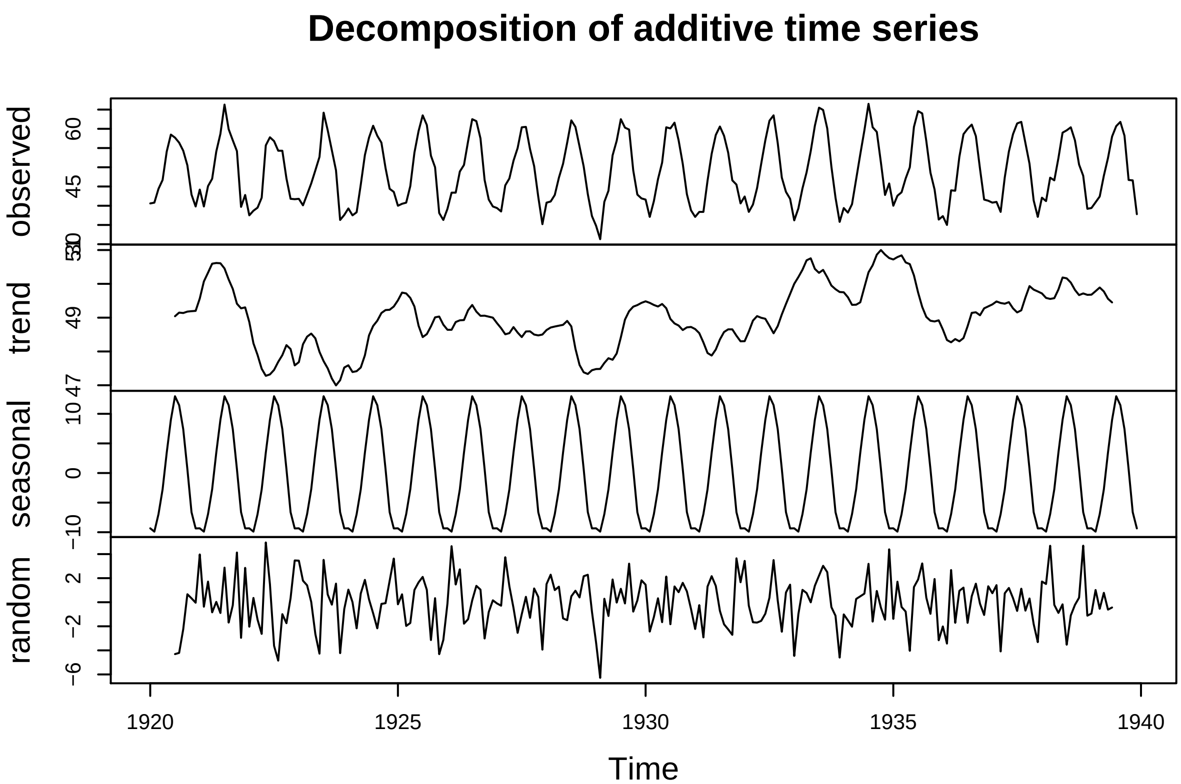
```
##
##  Augmented Dickey-Fuller Test
##
## data:  x
## Dickey-Fuller = -9.5087, Lag order = 9, p-value = 0.01
## alternative hypothesis: stationary
```

```
# very small p-value below 0.05 allows us to reject the null hypothesis of non stationary
```

```
plot(nottem) # Let s see the nottem dataset
```




```r
# clear seasonality

plot(decompose(nottem))
```



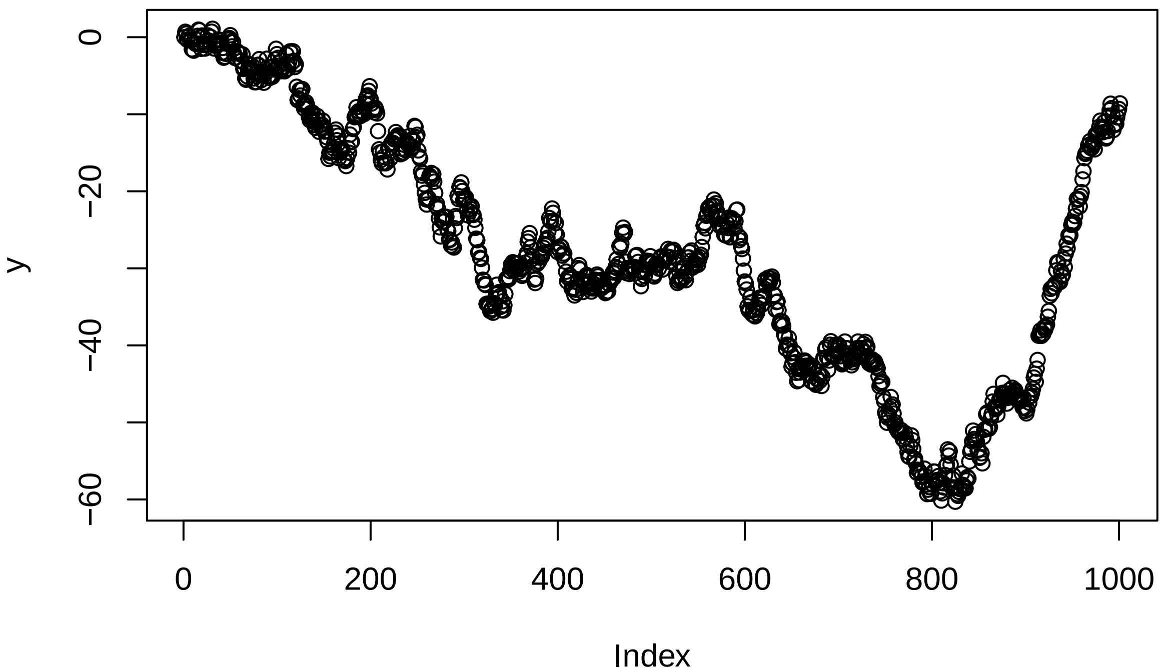

```r
# we don't see clear trend

adf.test(nottem)
```

```
## Warning in adf.test(nottem): p-value smaller than printed p-value
```

```
##
##  Augmented Dickey-Fuller Test
##
## data:  nottem
## Dickey-Fuller = -12.998, Lag order = 6, p-value = 0.01
## alternative hypothesis: stationary
```

```
# value less than 5%, leads to alternative hypothesis of stationarity
```

```
y <- diffinv(x) # non-stationary

plot(y)
```




```
# mean and variance could be changing

adf.test(y)
```

```
##
##  Augmented Dickey-Fuller Test
##
## data:  y
## Dickey-Fuller = 0.77514, Lag order = 9, p-value = 0.99
## alternative hypothesis: stationary
```

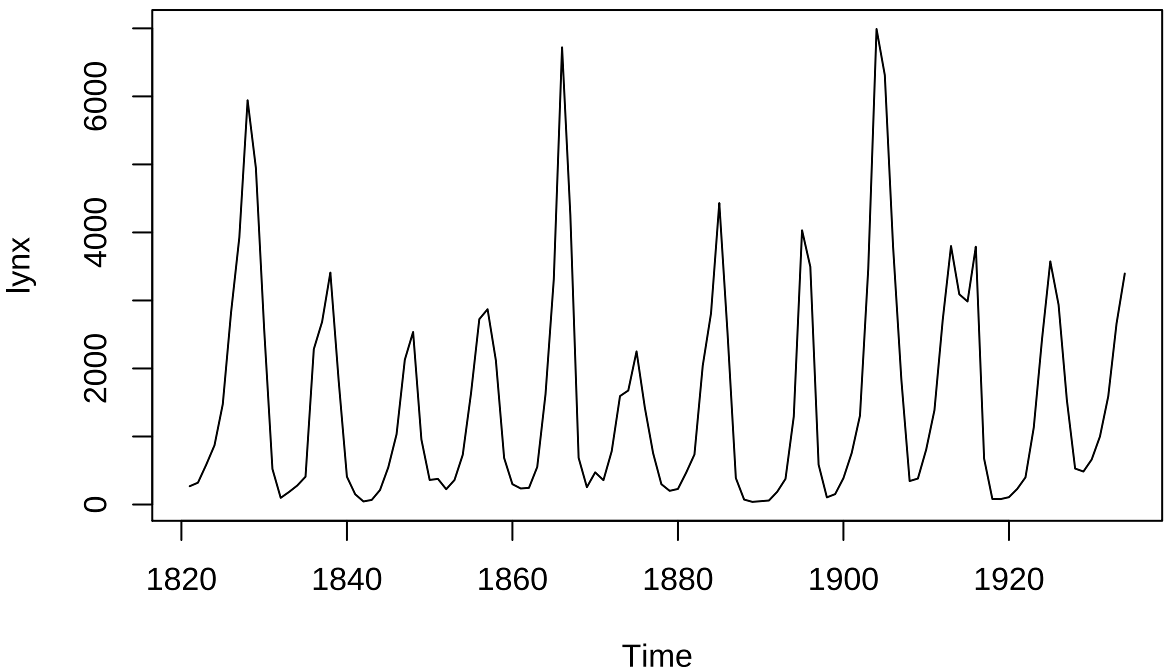
```
#p-value is higher than 0.05, we have a non-stationarity in dataset
```

11 Autocorrelation

- It is statistical term which describes the correlation (or the lack of such) in a time series dataset
- It is a key statistic, because it tells you whether previous observations influence the recent one => correlation on a time scale

- Lags: steps on a time scale
- For best statistical results, you always need to find out whether autocorrelation is present
- There are many tools available in R to test for autocorrelation but in most of the cases it is clear to see whether it present E.g. there won't be any autocorrelation in a random walk, whie the lynx dataset has it for sure

```
plot(lynx)
```



Lynx is a perfect example for an auto-correlated dataset. If you trap many lynx in one year, there will be less to catch in the following year.

11.1 Methods to get the autocorrelation calculated

acf() - Autocorrelation fuction

- Shows the autocorrelation between time lags in a time series
- It returns a measure

pacf() - Partial autocorrelation fuction

Durbin-Watson test - library (lmtest)

- Gets the autocorrelation only of the first order - between one time point and the immediate successor
- It is not robust to trends and seasonality
- Treat it with caution

11.2 Durbin-Watson test for autocorrelation

Assumption: there is autocorrelation in the lynx dataset

Preparation:

- To perform the DW test, the first (y) and last (x) observation of lynx needs to be chopped off
- This step provides 1 lag difference

- Test the formula argument (x~y) with the head() function

11.2.1 Example 1:

```r
# Durbin Watson test for autocorrelation

length(lynx); head(lynx); head(lynx[-1]); head(lynx[-114]) # check the required traits for the test
```

```
## [1] 114

## Time Series:
## Start = 1821
## End = 1826
## Frequency = 1
## [1]  269  321  585  871 1475 2821

## [1]  321  585  871 1475 2821 3928

## [1]  269  321  585  871 1475 2821
```

```r
# library(lmtest)

dwtest(lynx[-114] ~ lynx[-1]) # 1 lag time difference
```

```
##
##  Durbin-Watson test
##
## data:  lynx[-114] ~ lynx[-1]
## DW = 1.1296, p-value = 1.148e-06
## alternative hypothesis: true autocorrelation is greater than 0
```

```r
# highly significant p-value, confirmation of autocorrelation
```

11.2.2 Example 2:

```r
x = rnorm(700) # Lets take a look at random numbers

dwtest(x[-700] ~ x[-1])
```

```
##
##  Durbin-Watson test
##
## data:  x[-700] ~ x[-1]
## DW = 1.9982, p-value = 0.4904
## alternative hypothesis: true autocorrelation is greater than 0
```

```r
# can't reject null hypothesis, there is no autocorrelation
```

11.2.3 Example 3:

```r
length(nottem) # and the nottem dataset
```

```
## [1] 240
```

```r
dwtest(nottem[-240] ~ nottem[-1])
```

```
##
##  Durbin-Watson test
##
## data:  nottem[-240] ~ nottem[-1]
## DW = 1.0093, p-value = 5.097e-15
## alternative hypothesis: true autocorrelation is greater than 0
```

```r
# highly signifacnt p-value, there is true autocorrelation
```

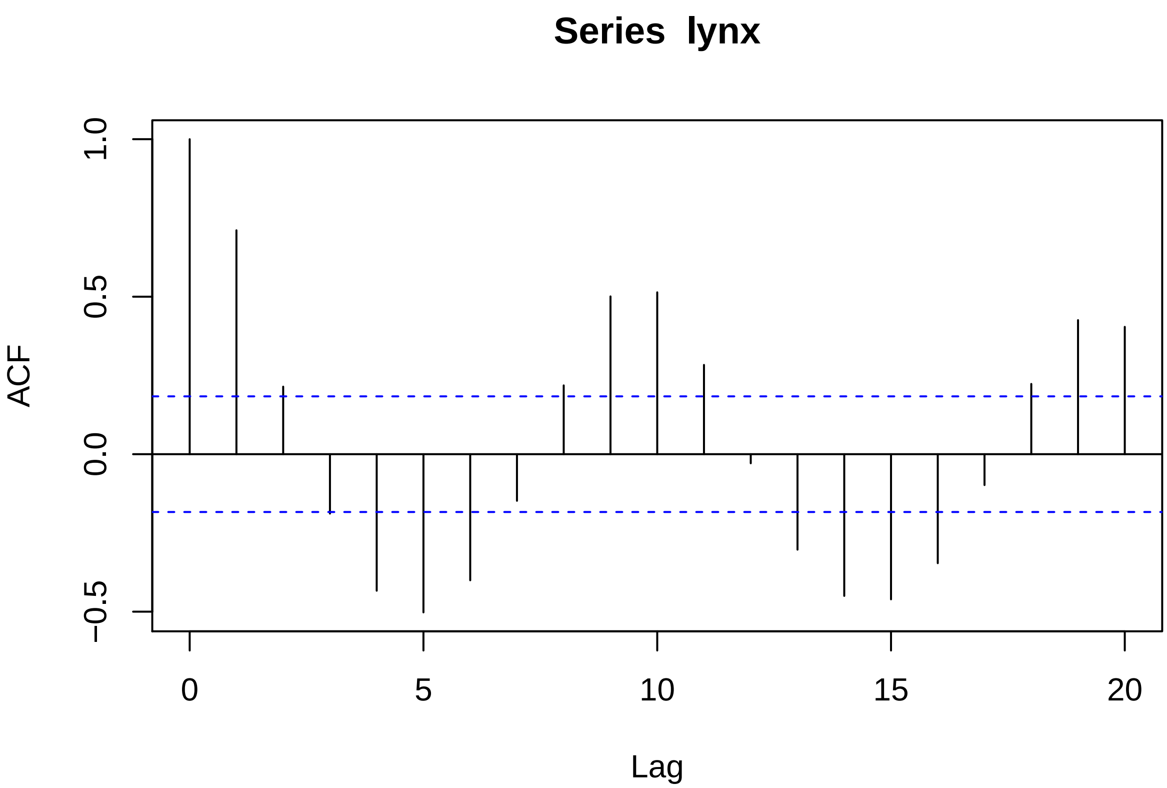
12 Functions acf() and pacf()

Function acf() and pacf() make sense on time series data - Alternatively, you could try all possible models one by one => time consuming - Using these functions provides a systematic way to identify the parameters

Autocorrelation: the correlation coefficient between different time points (lags) in a time series Partial autocorrelation: the correlation coefficient adjusted for al shorter lags in a time series

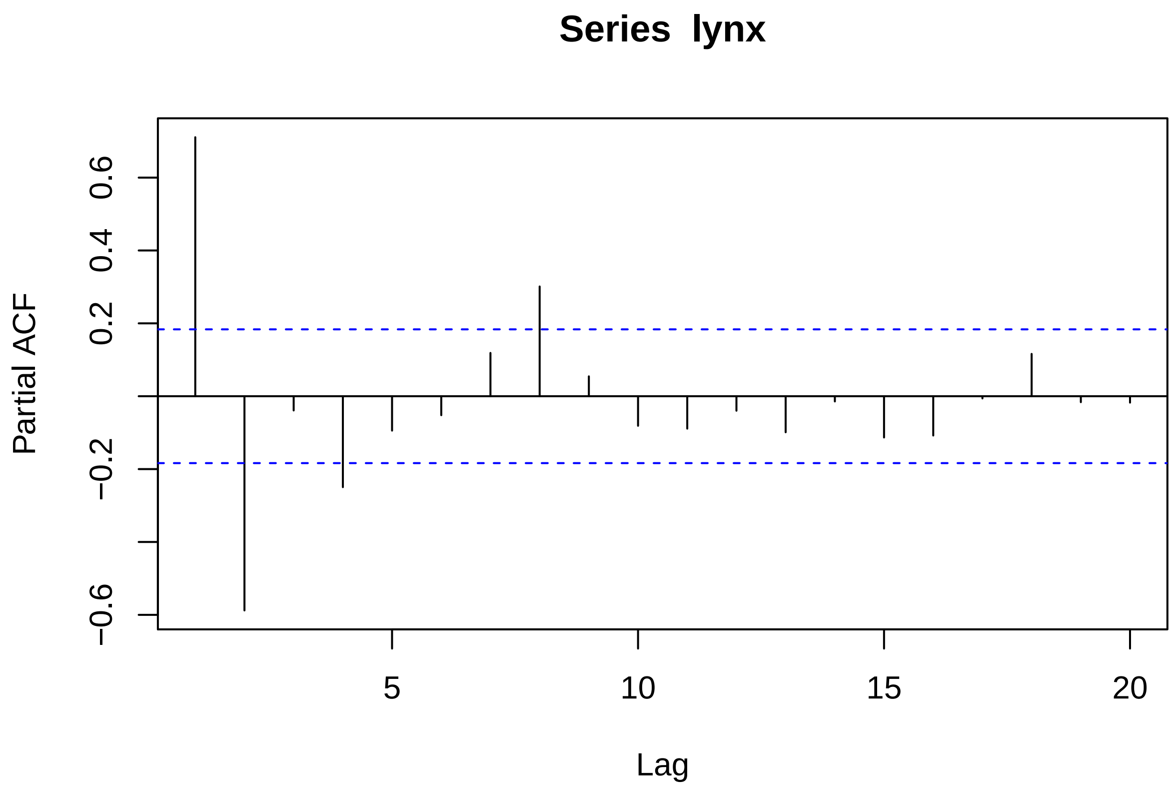
The acf() is used to identify the moving average (MA) part of the ARIMA model, while pacf() identifies the values for the autoregressive part (AR) - Both functions are part of R Base

```r
# lag.max for numbers of lags to be calculated

 acf(lynx, lag.max = 20)
```

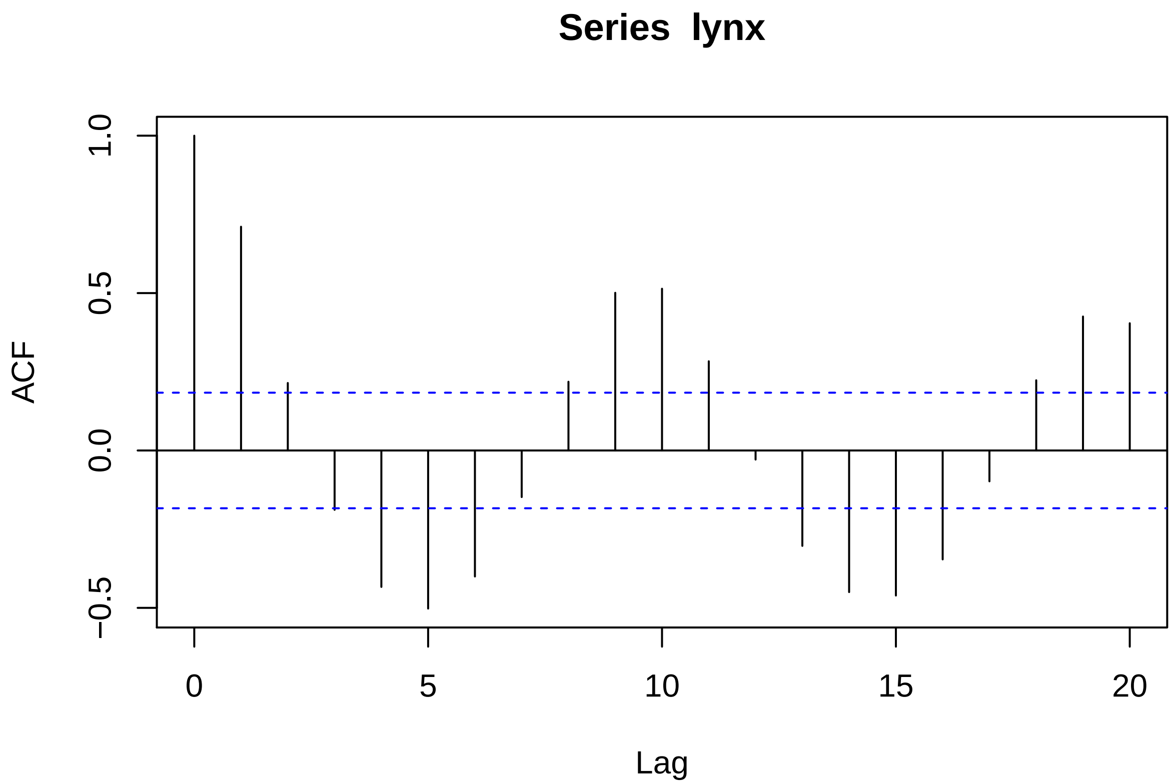



```r
# Several bars ranging out of the 95% confidence intervals
# Omit the first bar - it is the autocorrelation against itself at lag0
# The first two lags are significant
```

```r
pacf(lynx, lag.max =20)
```



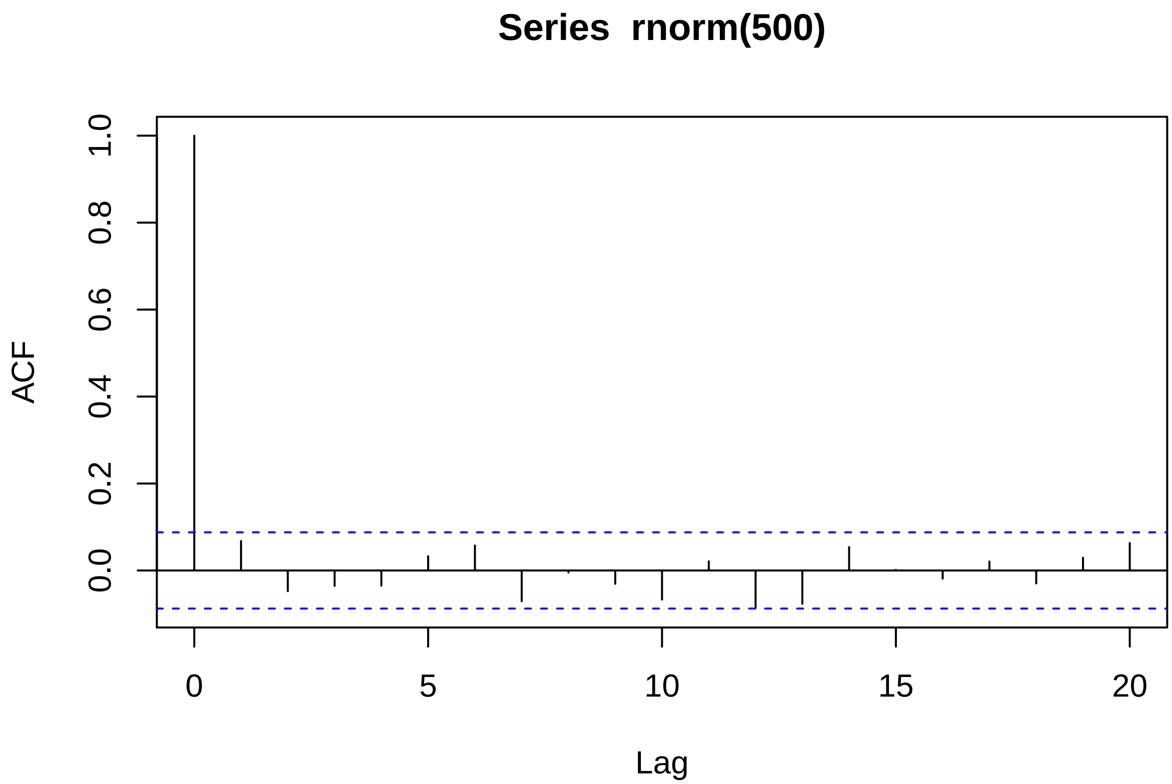

```r
# PACF starts at lag1
#The first lag is a significant lag, the second lag is significant to the negative side

#only data
acf(lynx, lag.max = 20); pacf(lynx, lag.max=20, plot=F)
```




```
##
## Partial autocorrelations of series 'lynx', by lag
##
##      1      2      3      4      5      6      7      8      9     10
##  0.711 -0.588 -0.039 -0.250 -0.094 -0.052  0.119  0.301  0.055 -0.081
##     11     12     13     14     15     16     17     18     19     20
## -0.089 -0.040 -0.099 -0.014 -0.113 -0.108 -0.006  0.116 -0.016 -0.018
```

```r
# blue line is 95% confidence interval
```

```r
acf(rnorm(500), lag.max = 20)
```



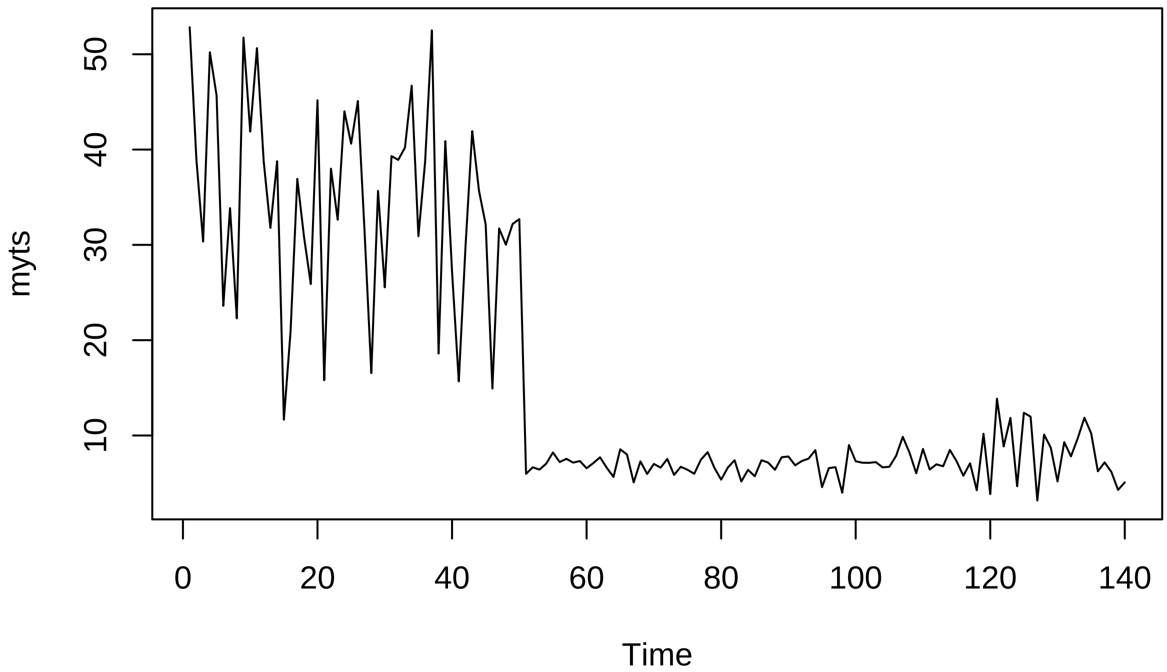

```
# I have to ignore first lag, only one lag is significant above the 95% confidence interval
# 20 observations with 95% confidence => one bar is expected to be outside the significance level
# A rnorm simulation is random, therefore the bars should be around 0
  # 1 = absolute positive correlation
  # -1 = absolute negative correlation
```

13 Exercise

13.1 TASK 1: Get the ad hoc dataset and plot it

Examine the data and the plot, explain the statistical traits of the time series and identify the problems and make a plan to fix them

```
set.seed(54)

myts <- ts(c(rnorm(50, 34, 10),
             rnorm(67, 7, 1),
             runif(23, 3, 14)))

plot(myts) #we can see that there is drop in mean and variance is not constant
```




```
myts <- log(myts)
```

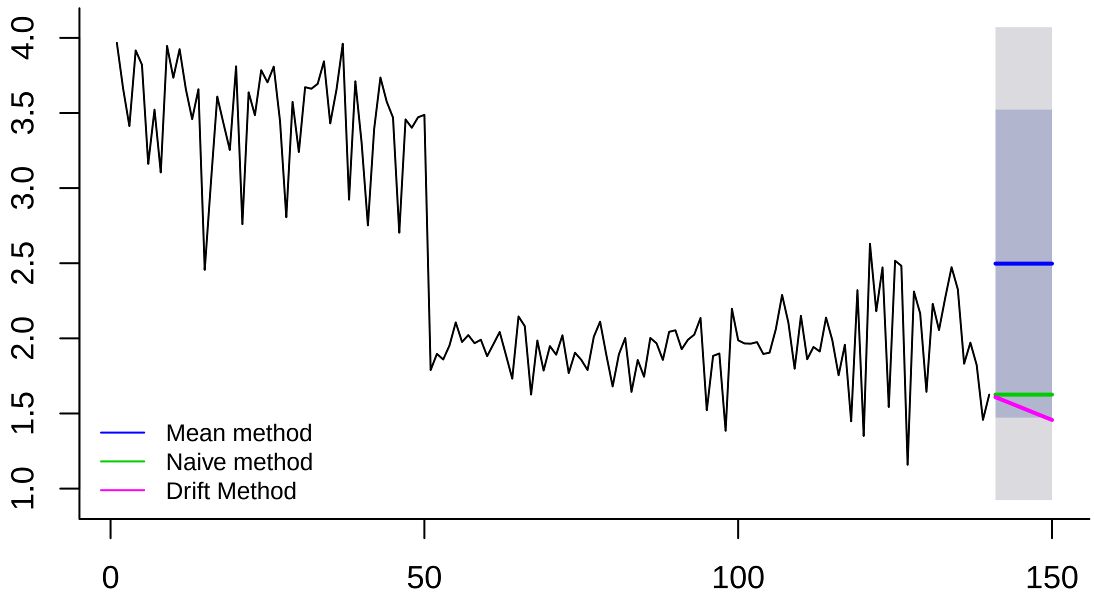
13.2 TASK 2: Set up three forecasting models with 10 steps into the future

```
#library(forecast)
meanm <- meanf(myts, h=10)
naivem <- naive(myts, h=10)
driftm <- rwf(myts, h=10, drift = T)
```

13.3 TASK 3: Get a plot with the three forecasts of the model

13.4 TASK 4: Which method looks most promising?

```
plot(meanm, main = "", bty = "l")
lines(naivem$mean, col=123, lwd = 2)
lines(driftm$mean, col=22, lwd = 2)
legend("bottomleft",lty=1,col=c(4,123,22), bty = "n", cex = 0.75,
       legend=c("Mean method","Naive method","Drift Method"))
```




```
# Drift(purple) method has a line going down due to one event; probably not a best solution

# Mean method puts equal weight on each observation which in this case gives us a blue line where not m

# Naive method puts all the weight on the most recent observations which seems to be a good approach in
```

13.5 TASK 5: Get the error measures and compare them

```
# Splitting data 80:20

length(myts)
```

```
## [1] 140
```

```
mytstrain <- window(myts, start = 1, end = 112 )

mytstest <- window(myts, start = 113)

#length 28 is taken from mytstest

meanma <- meanf(mytstrain, h=28)
naivema <- naive(mytstrain, h=28)
driftma <- rwf(mytstrain, h=28, drift = T)

# see which model is the best
accuracy(meanma, mytstest)
```

```
##                         ME      RMSE       MAE        MPE     MAPE
## Training set  1.846038e-16 0.8163346 0.7714004  -9.839234 31.23484
## Test set     -6.198835e-01 0.7307518 0.6204553 -36.737960 36.75971
##                  MASE       ACF1 Theil's U
## Training set 2.684607  0.8615304        NA
## Test set     2.159291 -0.2306541 0.9716032
```

```r
accuracy(naivema, mytstest)
```

```
##                       ME      RMSE       MAE       MPE     MAPE     MASE
## Training set -0.01824047 0.4071185 0.2873421 -1.870232 11.33626 1.000000
## Test set      0.05893104 0.3914276 0.3316489 -1.328849 17.76316 1.154196
##                    ACF1 Theil's U
## Training set -0.4450652        NA
## Test set     -0.2306541  0.577168
```

```r
accuracy(driftma, mytstest)
```

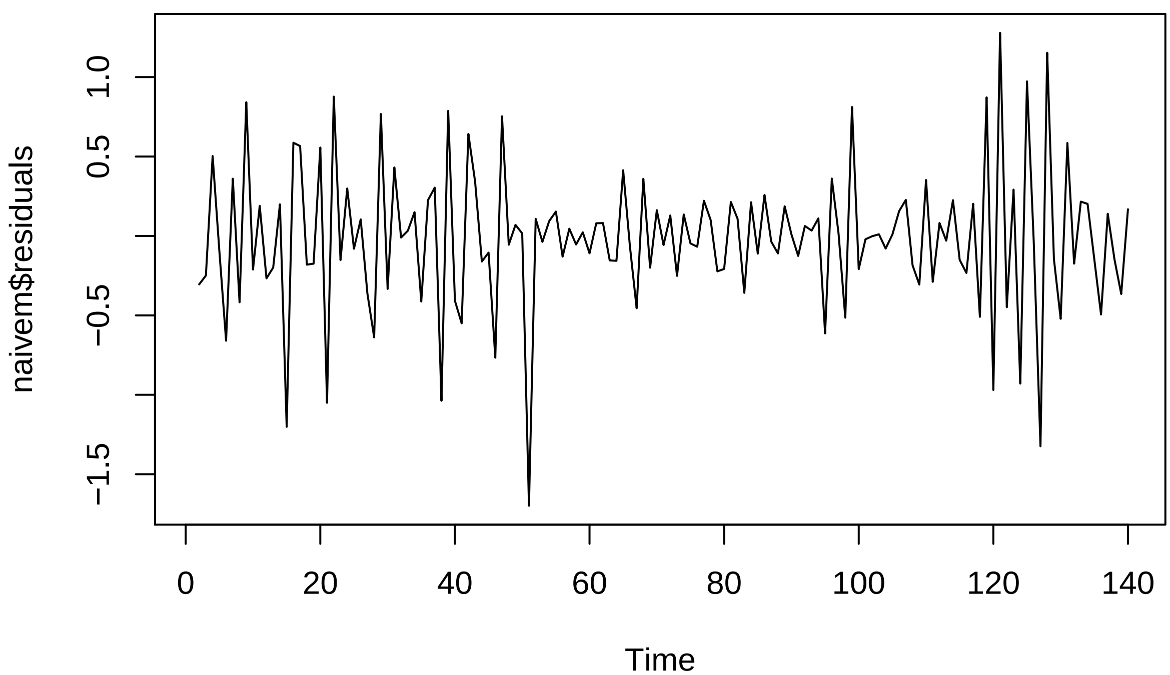
```
##                         ME      RMSE       MAE       MPE     MAPE     MASE
## Training set -9.802537e-17 0.4067096 0.2876207 -1.103181 11.31901 1.000970
## Test set      3.234179e-01 0.5206086 0.4411263 12.524858 21.27240 1.535196
##                    ACF1 Theil's U
## Training set -0.4450652        NA
## Test set     -0.1049056 0.7824451
```

```r
# lowest values for RMSE, MAE, MAPE, MASE => Naive method

# We can focus on naive model
```

13.6 TASK 6: Check all relevant statistical traits

- mean or zero
- no autocorrelaton in the residuals
- equal variance
- standard distribution of the residuals

```r
plot(naivem$residuals)
```

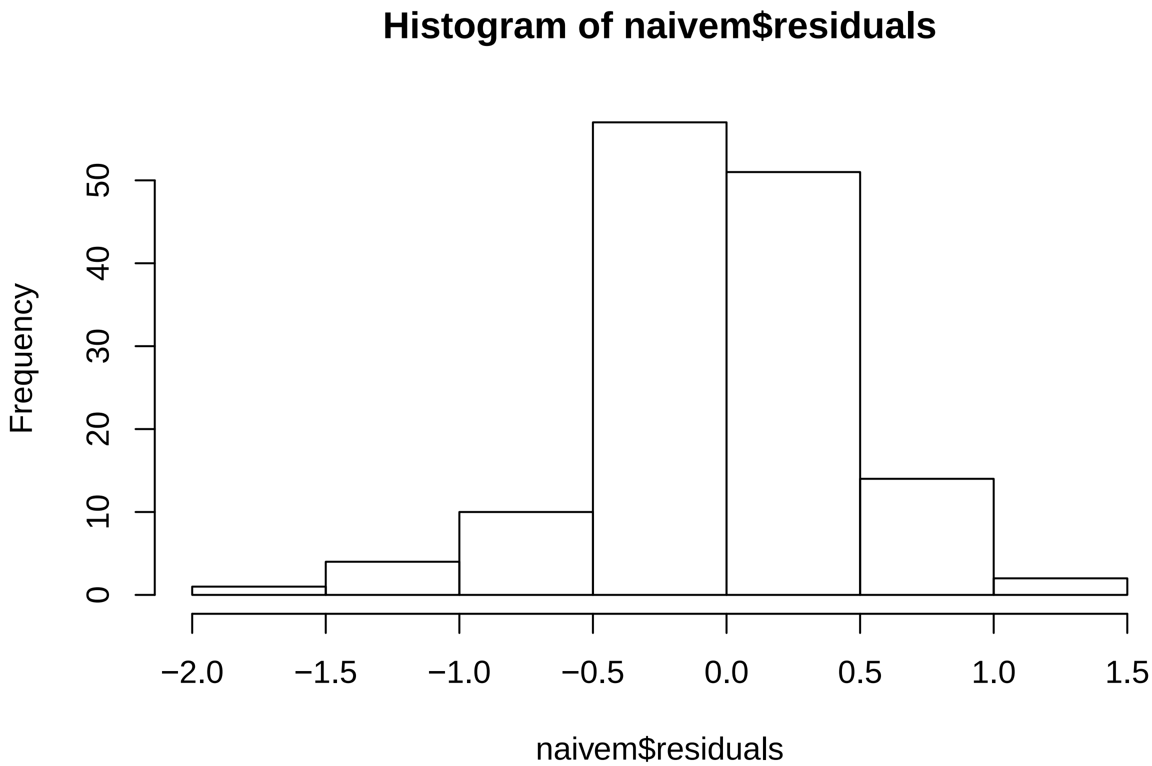



```r
# graphs is not homoscedastic and mean is not around 0
```

```r
mean(naivem$residuals[2:140])
```

```
## [1] -0.01684688
```

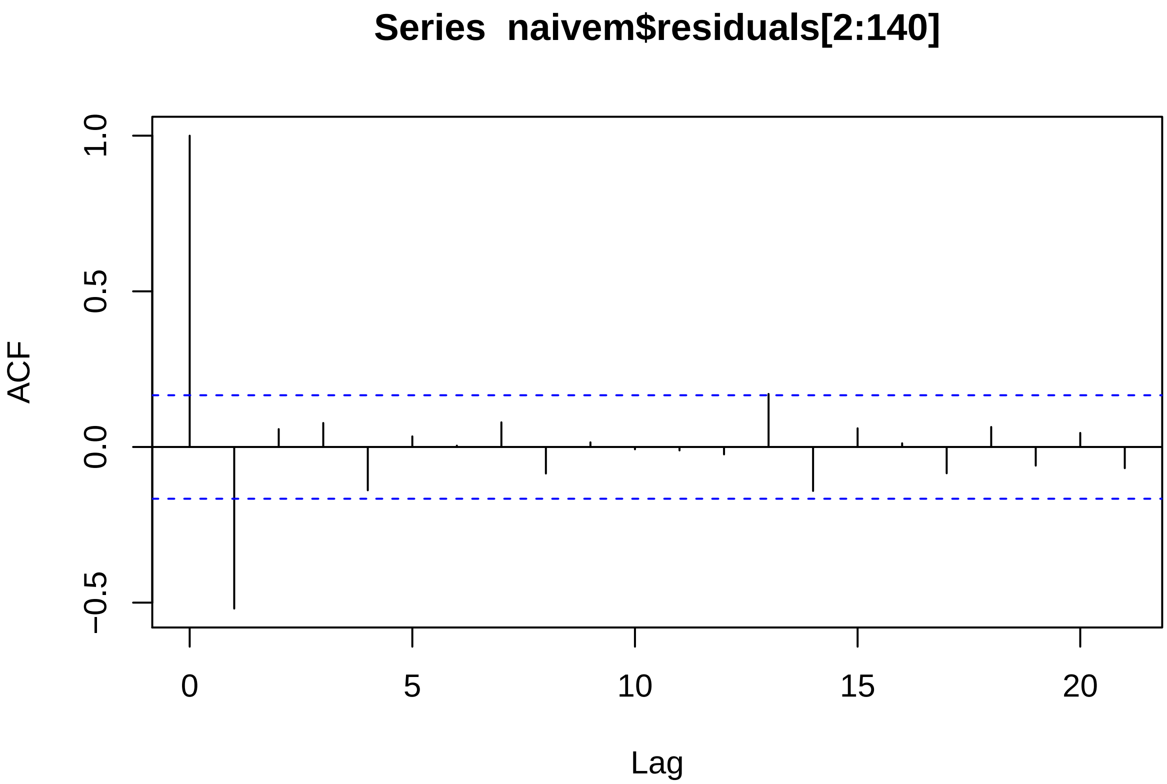
```r
hist(naivem$residuals) # doesn't seem to be a normal distribution, too much weight on the center, let's
```




```r
shapiro.test(naivem$residuals) # test for normal distribution, normal distr can be rejected (H1)
```

```
##
##  Shapiro-Wilk normality test
##
## data:  naivem$residuals
## W = 0.961, p-value = 0.0005413
```

```r
acf(naivem$residuals[2:140]) # autocorrelation test, autocorrelation present (1 bar crossing would be f
```



13.7 TASK 7: Examine the test result: are there any fixes needed? What is the easiest tool to improve the model?

13.8 TASK 8: Perform the whole Analysis with the log transformation on the data

- We can use the logarithm, rescale the data and get rid of most of these problems e.g. heteroscedasticity
- We can use the code above again, just run the myts <- log(myts) as well
- Variance is better than before and drop is not so deep.
- Naive model still looks the best. And with accuracy measures, we can see that it is still true.
- Looking at residuals variance, we can see that situation is now much better and the mean is close to zero.
- Data is still not distributed evenly but it is ok if mean and autocorrelation look ok.
- The test statistics is still below 0.05 with Shapiro-Wilk test.
- After checking acf function, we can see that there is only 1 bar getting of the confidence interval.

14 Time Series Analysis And Forecasting

14.1 Selecting a Suitable Model - Quantitative Forecasting Models

Quantitative Forecasting:

A) Linear Models

- Simple Models (not good if trend or seasonality)
- Exponential Smoothing
- ARIMA
- Seasonal Decomposition

The most widely used models are Exponential Smoothing and ARIMA model.

Exponential Smoothing - Trend and seasonality are key determinants. Can put more weight on recent observations.

ARIMA Model - Explains patterns in the data based on autoregression.

Seasonal Decomposition - The dataset needs to be seasonal or at least have a frequency. Minimum number of seasonal cycles (2).

Further Linear Models - linear regressions, dynamic regressions, vector autoregressive models (when more variables involved; library vars)

B) Non-Linear Models

- Neural Nets
- Support Vector Machines
- Clustering

Neural Nets - Tries to model the brain's neuron system: - An input vector is compressed to several layers - Each layer consists of multiple neurons - Weight of importance may be ascribed to each neuron

The amount of requred layers is specified by the dataset. Library 'forecast' - nnetar() or library 'nnfor'

Clustering - library 'kml', implements k-means clustering

15 Seasonal Decomposition

15.1 Univariate Seasonal Time Series

Modelling options:

- Seasonal ARIMA
- Holt-Winters Exponential Smoothing
- Seasonal Decomposition

To perform seasonal decomposition, the dataset must have a seasonal component

- Frequency parameter for generated data
- Frequently measured data: inflation rates, weather measurements etc.

Seasonal decomposition decomposes seasonal time series data to its components

- Trend
- Seasonality
- Remainder - random data

Methods:

Additive - adds components up, use this one if the seasonal component stays constant over several cycles Multiplicative- Multiplies components

Drawbacks of Seasonal Decomposition:

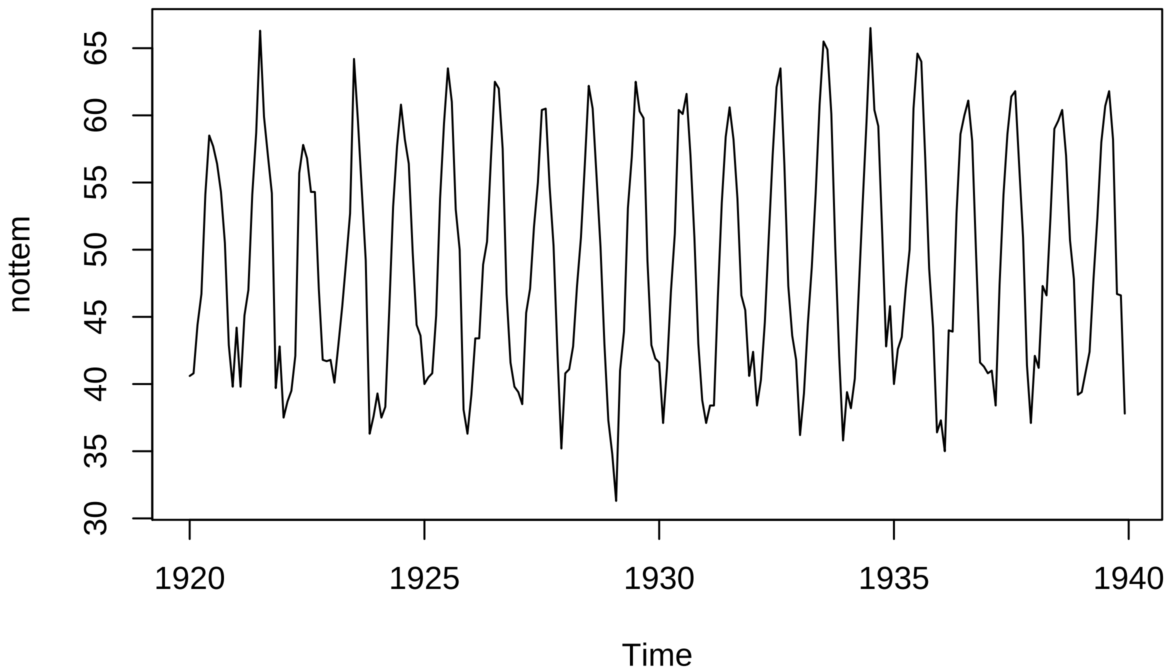
- N/A Values

- Slow to catch sudden changes
- Constant seasonality

Alternative methods:

- SEATS, x11, stl decomposition
- Values for all observations - no N/A
- Seasonal part can be adjusted over time
- Tools:
- R Base: decompose(), stl()
- Library ‘forecast’: Integration stl generated objects, stlf()
- Library ‘seasonal’: seas()

```r
plot(nottem) #stable seasonality and no trend => we can use additive model
```




```r
#If the amplitude of the seasons stay roughly the same that means the distance between highs and lows o

#Peaks of the seasons are not moving either up or down -> no trend
```

```r
length(nottem) #20 years multiply by 12 months
```

```
## [1] 240
```

```r
decompose(nottem, type = "additive")
```

```
## $x
##       Jan  Feb  Mar  Apr  May  Jun  Jul  Aug  Sep  Oct  Nov  Dec
## 1920 40.6 40.8 44.4 46.7 54.1 58.5 57.7 56.4 54.3 50.5 42.9 39.8
## 1921 44.2 39.8 45.1 47.0 54.1 58.7 66.3 59.9 57.0 54.2 39.7 42.8
## 1922 37.5 38.7 39.5 42.1 55.7 57.8 56.8 54.3 54.3 47.1 41.8 41.7
## 1923 41.8 40.1 42.9 45.8 49.2 52.7 64.2 59.6 54.4 49.2 36.3 37.6
## 1924 39.3 37.5 38.3 45.5 53.2 57.7 60.8 58.2 56.4 49.8 44.4 43.6
## 1925 40.0 40.5 40.8 45.1 53.8 59.4 63.5 61.0 53.0 50.0 38.1 36.3
```

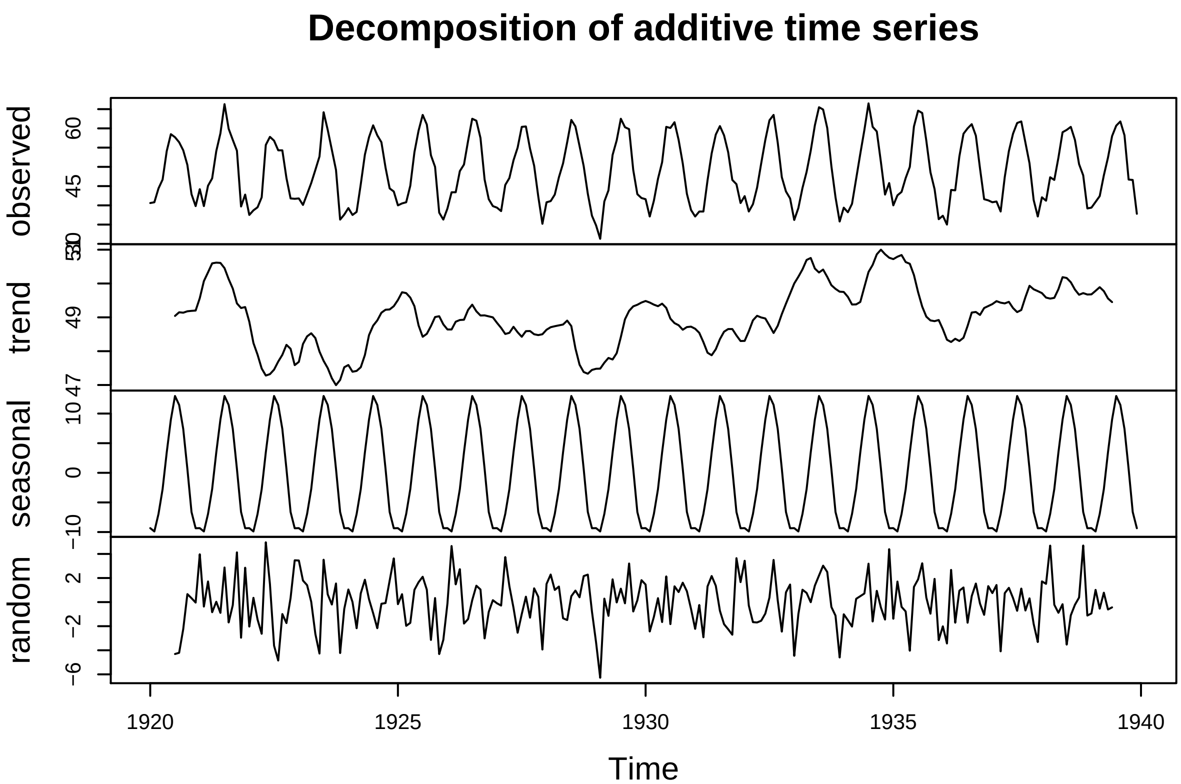
```
## 1926 39.2 43.4 43.4 48.9 50.6 56.8 62.5 62.0 57.5 46.7 41.6 39.8
## 1927 39.4 38.5 45.3 47.1 51.7 55.0 60.4 60.5 54.7 50.3 42.3 35.2
## 1928 40.8 41.1 42.8 47.3 50.9 56.4 62.2 60.5 55.4 50.2 43.0 37.3
## 1929 34.8 31.3 41.0 43.9 53.1 56.9 62.5 60.3 59.8 49.2 42.9 41.9
## 1930 41.6 37.1 41.2 46.9 51.2 60.4 60.1 61.6 57.0 50.9 43.0 38.8
## 1931 37.1 38.4 38.4 46.5 53.5 58.4 60.6 58.2 53.8 46.6 45.5 40.6
## 1932 42.4 38.4 40.3 44.6 50.9 57.0 62.1 63.5 56.3 47.3 43.6 41.8
## 1933 36.2 39.3 44.5 48.7 54.2 60.8 65.5 64.9 60.1 50.2 42.1 35.8
## 1934 39.4 38.2 40.4 46.9 53.4 59.6 66.5 60.4 59.2 51.2 42.8 45.8
## 1935 40.0 42.6 43.5 47.1 50.0 60.5 64.6 64.0 56.8 48.6 44.2 36.4
## 1936 37.3 35.0 44.0 43.9 52.7 58.6 60.0 61.1 58.1 49.6 41.6 41.3
## 1937 40.8 41.0 38.4 47.4 54.1 58.6 61.4 61.8 56.3 50.9 41.4 37.1
## 1938 42.1 41.2 47.3 46.6 52.4 59.0 59.6 60.4 57.0 50.7 47.8 39.2
## 1939 39.4 40.9 42.4 47.8 52.4 58.0 60.7 61.8 58.2 46.7 46.6 37.8
##
## $seasonal
##             Jan        Feb        Mar        Apr        May        Jun
## 1920 -9.3393640 -9.8998904 -6.9466009 -2.7573465  3.4533991  8.9865132
## 1921 -9.3393640 -9.8998904 -6.9466009 -2.7573465  3.4533991  8.9865132
## 1922 -9.3393640 -9.8998904 -6.9466009 -2.7573465  3.4533991  8.9865132
## 1923 -9.3393640 -9.8998904 -6.9466009 -2.7573465  3.4533991  8.9865132
## 1924 -9.3393640 -9.8998904 -6.9466009 -2.7573465  3.4533991  8.9865132
## 1925 -9.3393640 -9.8998904 -6.9466009 -2.7573465  3.4533991  8.9865132
## 1926 -9.3393640 -9.8998904 -6.9466009 -2.7573465  3.4533991  8.9865132
## 1927 -9.3393640 -9.8998904 -6.9466009 -2.7573465  3.4533991  8.9865132
## 1928 -9.3393640 -9.8998904 -6.9466009 -2.7573465  3.4533991  8.9865132
## 1929 -9.3393640 -9.8998904 -6.9466009 -2.7573465  3.4533991  8.9865132
## 1930 -9.3393640 -9.8998904 -6.9466009 -2.7573465  3.4533991  8.9865132
## 1931 -9.3393640 -9.8998904 -6.9466009 -2.7573465  3.4533991  8.9865132
## 1932 -9.3393640 -9.8998904 -6.9466009 -2.7573465  3.4533991  8.9865132
## 1933 -9.3393640 -9.8998904 -6.9466009 -2.7573465  3.4533991  8.9865132
## 1934 -9.3393640 -9.8998904 -6.9466009 -2.7573465  3.4533991  8.9865132
## 1935 -9.3393640 -9.8998904 -6.9466009 -2.7573465  3.4533991  8.9865132
## 1936 -9.3393640 -9.8998904 -6.9466009 -2.7573465  3.4533991  8.9865132
## 1937 -9.3393640 -9.8998904 -6.9466009 -2.7573465  3.4533991  8.9865132
## 1938 -9.3393640 -9.8998904 -6.9466009 -2.7573465  3.4533991  8.9865132
## 1939 -9.3393640 -9.8998904 -6.9466009 -2.7573465  3.4533991  8.9865132
##             Jul        Aug        Sep        Oct        Nov        Dec
## 1920 12.9672149 11.4591009  7.4001096  0.6547149 -6.6176535 -9.3601974
## 1921 12.9672149 11.4591009  7.4001096  0.6547149 -6.6176535 -9.3601974
## 1922 12.9672149 11.4591009  7.4001096  0.6547149 -6.6176535 -9.3601974
## 1923 12.9672149 11.4591009  7.4001096  0.6547149 -6.6176535 -9.3601974
## 1924 12.9672149 11.4591009  7.4001096  0.6547149 -6.6176535 -9.3601974
## 1925 12.9672149 11.4591009  7.4001096  0.6547149 -6.6176535 -9.3601974
## 1926 12.9672149 11.4591009  7.4001096  0.6547149 -6.6176535 -9.3601974
## 1927 12.9672149 11.4591009  7.4001096  0.6547149 -6.6176535 -9.3601974
## 1928 12.9672149 11.4591009  7.4001096  0.6547149 -6.6176535 -9.3601974
## 1929 12.9672149 11.4591009  7.4001096  0.6547149 -6.6176535 -9.3601974
## 1930 12.9672149 11.4591009  7.4001096  0.6547149 -6.6176535 -9.3601974
## 1931 12.9672149 11.4591009  7.4001096  0.6547149 -6.6176535 -9.3601974
## 1932 12.9672149 11.4591009  7.4001096  0.6547149 -6.6176535 -9.3601974
## 1933 12.9672149 11.4591009  7.4001096  0.6547149 -6.6176535 -9.3601974
## 1934 12.9672149 11.4591009  7.4001096  0.6547149 -6.6176535 -9.3601974
## 1935 12.9672149 11.4591009  7.4001096  0.6547149 -6.6176535 -9.3601974
```

```
## 1936 12.9672149 11.4591009  7.4001096  0.6547149 -6.6176535 -9.3601974
## 1937 12.9672149 11.4591009  7.4001096  0.6547149 -6.6176535 -9.3601974
## 1938 12.9672149 11.4591009  7.4001096  0.6547149 -6.6176535 -9.3601974
## 1939 12.9672149 11.4591009  7.4001096  0.6547149 -6.6176535 -9.3601974
##
## $trend
##           Jan      Feb      Mar      Apr      May      Jun      Jul
## 1920       NA       NA       NA       NA       NA       NA 49.04167
## 1921 49.56667 50.07083 50.32917 50.59583 50.61667 50.60833 50.45417
## 1922 48.87083 48.24167 47.89583 47.48750 47.27917 47.32083 47.45417
## 1923 47.68333 48.21250 48.43750 48.52917 48.38750 47.98750 47.71250
## 1924 47.59167 47.39167 47.41667 47.52500 47.88750 48.47500 48.75417
## 1925 49.51250 49.74167 49.71667 49.58333 49.32917 48.76250 48.42500
## 1926 48.64167 48.64167 48.87083 48.92083 48.92917 49.22083 49.37500
## 1927 48.83750 48.68750 48.50833 48.54167 48.72083 48.55833 48.42500
## 1928 48.63333 48.70833 48.73750 48.76250 48.78750 48.90417 48.74167
## 1929 47.47917 47.48333 47.65833 47.80000 47.75417 47.94167 48.41667
## 1930 49.48333 49.43750 49.37500 49.32917 49.40417 49.27917 48.96250
## 1931 48.66250 48.54167 48.26667 47.95417 47.87917 48.05833 48.35417
## 1932 48.30417 48.58750 48.91250 49.04583 48.99583 48.96667 48.75833
## 1933 50.00000 50.20000 50.41667 50.69583 50.75417 50.44167 50.32500
## 1934 49.75000 49.60417 49.37917 49.38333 49.45417 49.90000 50.34167
## 1935 50.72083 50.79167 50.84167 50.63333 50.58333 50.25000 49.74583
## 1936 48.65000 48.33750 48.27083 48.36667 48.30000 48.39583 48.74583
## 1937 49.39167 49.47917 49.43333 49.41250 49.45833 49.27500 49.15417
## 1938 49.71667 49.58333 49.55417 49.57500 49.83333 50.18750 50.16250
## 1939 49.67917 49.78333 49.89167 49.77500 49.55833 49.45000       NA
##           Aug      Sep      Oct      Nov      Dec
## 1920 49.15000 49.13750 49.17917 49.19167 49.20000
## 1921 50.12917 49.85000 49.41250 49.27500 49.30417
## 1922 47.69167 47.89167 48.18750 48.07083 47.58750
## 1923 47.50000 47.20000 46.99583 47.15000 47.52500
## 1924 48.90833 49.13750 49.22500 49.23333 49.32917
## 1925 48.51250 48.74167 49.00833 49.03333 48.79167
## 1926 49.17917 49.05417 49.05833 49.02917 49.00000
## 1927 48.59167 48.59583 48.50000 48.47500 48.50000
## 1928 48.08333 47.60000 47.38333 47.33333 47.44583
## 1929 48.94167 49.19167 49.32500 49.37083 49.43750
## 1930 48.82917 48.76667 48.63333 48.71250 48.72500
## 1931 48.57500 48.65417 48.65417 48.46667 48.30000
## 1932 48.53750 48.75000 49.09583 49.40417 49.70000
## 1933 50.41250 50.19583 49.95000 49.84167 49.75833
## 1934 50.55000 50.86250 51.00000 50.86667 50.76250
## 1935 49.31667 49.02083 48.90833 48.88750 48.92083
## 1936 49.14167 49.15833 49.07083 49.27500 49.33333
## 1937 49.21667 49.59583 49.93333 49.82917 49.77500
## 1938 50.03750 49.82083 49.66667 49.71667 49.67500
## 1939       NA       NA       NA       NA       NA
##
## $random
##              Jan          Feb          Mar          Apr          May
## 1920          NA           NA           NA           NA           NA
## 1921  3.972697368 -0.370942982  1.717434211 -0.838486842  0.029934211
## 1922 -2.031469298  0.358223684 -1.449232456 -2.630153509  4.967434211
```

```
## 1923  3.456030702  1.787390351  1.409100877  0.028179825 -2.640899123
## 1924  1.047697368  0.008223684 -2.170065789  0.732346491  1.859100877
## 1925 -0.173135965  0.658223684 -1.970065789 -1.725986842  1.017434211
## 1926 -0.102302632  4.658223684  1.475767544  2.736513158 -1.782565789
## 1927 -0.098135965 -0.287609649  3.738267544  1.315679825 -0.474232456
## 1928  1.506030702  2.291557018  1.009100877  1.294846491 -1.340899123
## 1929 -3.339802632 -6.283442982  0.288267544 -1.142653509  1.892434211
## 1930  1.456030702 -2.437609649 -1.228399123  0.328179825 -1.657565789
## 1931 -2.223135965 -0.241776316 -2.920065789  1.303179825  2.167434211
## 1932  3.435197368 -0.287609649 -1.665899123 -1.688486842 -1.549232456
## 1933 -4.460635965 -1.000109649  1.029934211  0.761513158 -0.007565789
## 1934 -1.010635965 -1.504276316 -2.032565789  0.274013158  0.492434211
## 1935 -1.381469298  1.708223684 -0.395065789 -0.775986842 -4.036732456
## 1936 -2.010635965 -3.437609649  2.675767544 -1.709320175  0.946600877
## 1937  0.747697368  1.420723684 -4.086732456  0.744846491  1.188267544
## 1938  1.722697368  1.516557018  4.692434211 -0.217653509 -0.886732456
## 1939 -0.939802632  1.016557018 -0.545065789  0.782346491 -0.611732456
##               Jun          Jul          Aug          Sep          Oct
## 1920           NA -4.308881579 -4.209100877 -2.237609649  0.666118421
## 1921 -0.894846491  2.878618421 -1.688267544 -0.250109649  4.132785088
## 1922  1.492653509 -3.621381579 -4.850767544 -0.991776316 -1.742214912
## 1923 -4.274013158  3.520285088  0.640899123 -0.200109649  1.549451754
## 1924  0.238486842 -0.921381579 -2.167434211 -0.137609649 -0.079714912
## 1925  1.650986842  2.107785088  1.028399123 -3.141776316  0.336951754
## 1926 -1.407346491  0.157785088  1.361732456  1.045723684 -3.013048246
## 1927 -2.544846491 -0.992214912  0.449232456 -1.295942982  1.145285088
## 1928 -1.490679825  0.491118421  0.957565789  0.399890351  2.161951754
## 1929 -0.028179825  1.116118421 -0.100767544  3.208223684 -0.779714912
## 1930  2.134320175 -1.829714912  1.311732456  0.833223684  1.611951754
## 1931  1.355153509 -0.721381579 -1.834100877 -2.254276316 -2.708881579
## 1932 -0.953179825  0.374451754  3.503399123  0.149890351 -2.450548246
## 1933  1.371820175  2.207785088  3.028399123  2.504057018 -0.404714912
## 1934  0.713486842  3.191118421 -1.609100877  0.937390351 -0.454714912
## 1935  1.263486842  1.886951754  3.224232456  0.379057018 -0.963048246
## 1936  1.217653509 -1.713048246  0.499232456  1.541557018 -0.125548246
## 1937  0.338486842 -0.721381579  1.124232456 -0.695942982  0.311951754
## 1938 -0.174013158 -3.529714912 -1.096600877 -0.220942982  0.378618421
## 1939 -0.436513158           NA           NA           NA           NA
##               Nov          Dec
## 1920  0.325986842 -0.039802632
## 1921 -2.957346491  2.856030702
## 1922  0.346820175  3.472697368
## 1923 -4.232346491 -0.564802632
## 1924  1.784320175  3.631030702
## 1925 -4.315679825 -3.131469298
## 1926 -0.811513158  0.160197368
## 1927  0.442653509 -3.939802632
## 1928  2.284320175 -0.785635965
## 1929  0.146820175  1.822697368
## 1930  0.905153509 -0.564802632
## 1931  3.650986842  1.660197368
## 1932  0.813486842  1.460197368
## 1933 -1.124013158 -4.598135965
## 1934 -1.449013158  4.397697368
```

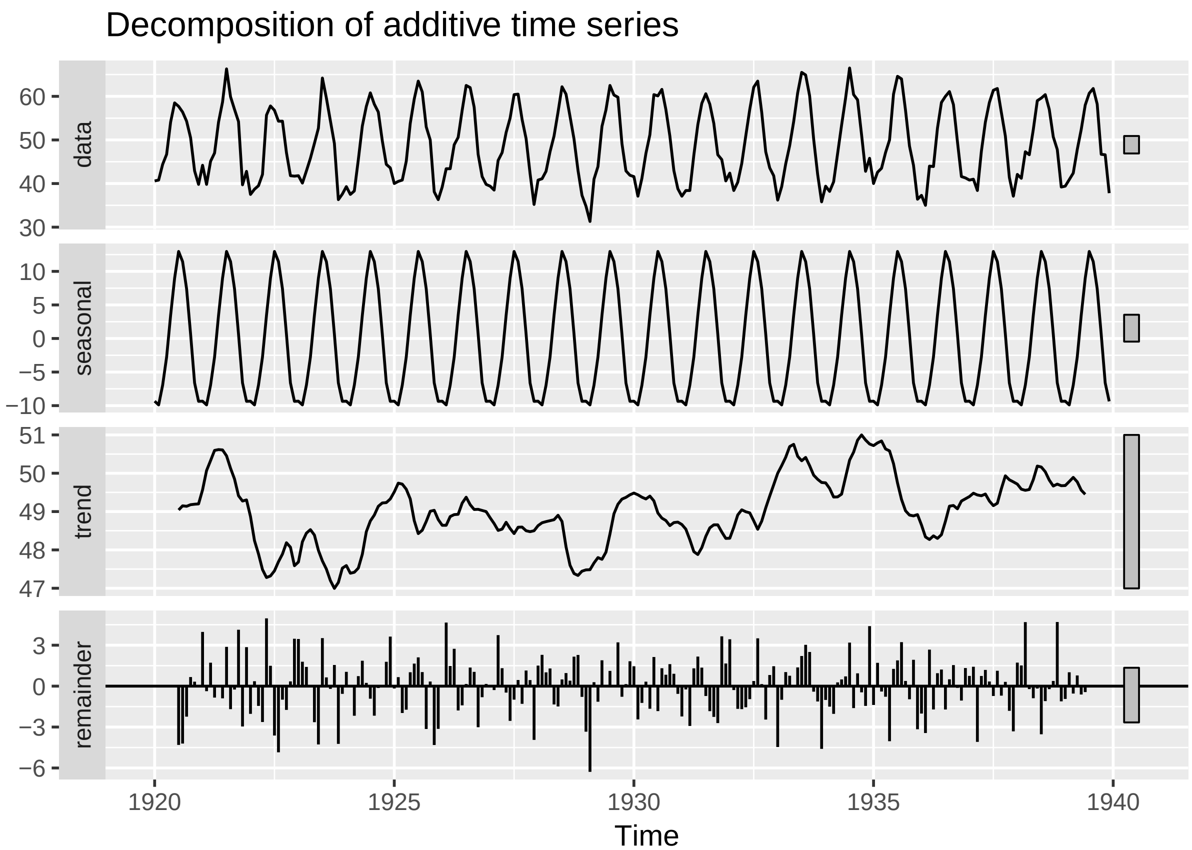
```
## 1935  1.930153509 -3.160635965
## 1936 -1.057346491  1.326864035
## 1937 -1.811513158 -3.314802632
## 1938  4.700986842 -1.114802632
## 1939           NA           NA
##
## $figure
##  [1] -9.3393640 -9.8998904 -6.9466009 -2.7573465  3.4533991  8.9865132
##  [7] 12.9672149 11.4591009  7.4001096  0.6547149 -6.6176535 -9.3601974
##
## $type
## [1] "additive"
##
## attr(,"class")
## [1] "decomposed.ts"
```

```r
plot(decompose(nottem, type="additive"))
```

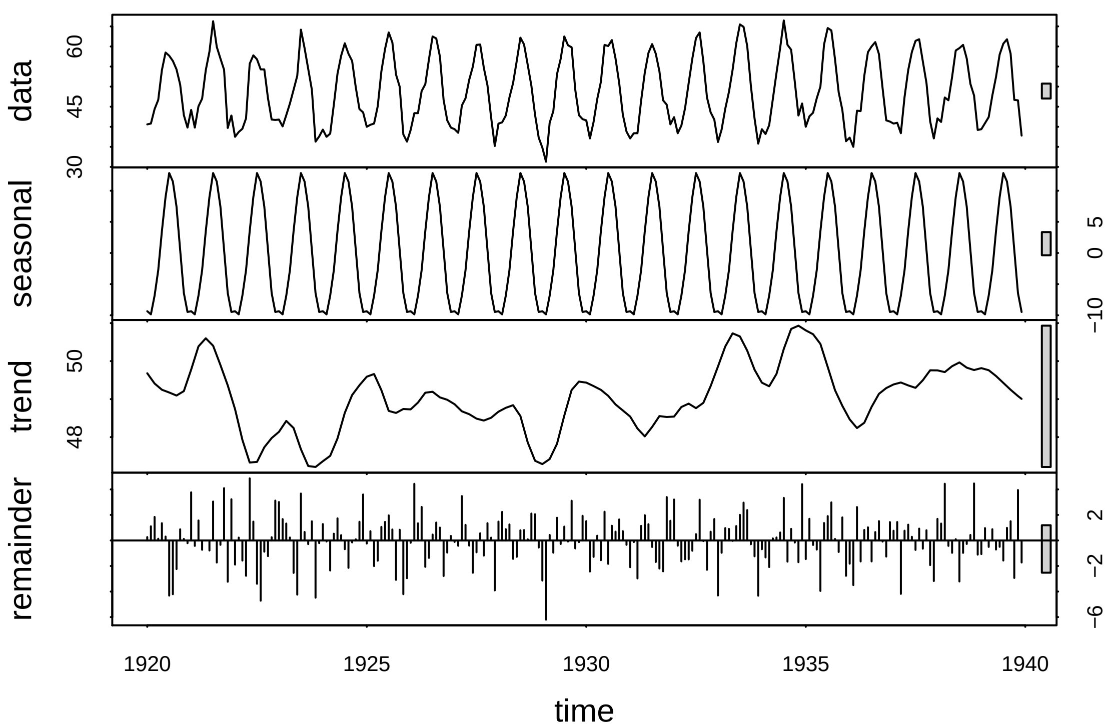



```r
# we can see there is no trend because the data stays around the mean (except two peaks over the time)
# seasonal part is quite clear to recognize and it stays constant over the whole time
```

```r
# alternative using ggplot
# in order to use autoplot, both libraries need to be activated - ggplot2 and forecast
autoplot(decompose(nottem, type="additive"))
```




```
# alternatively the function stl could be used
plot(stl(nottem, s.window="periodic"))
```



```r
stl(nottem, s.window="periodic")
```

```
##  Call:
##  stl(x = nottem, s.window = "periodic")
##
## Components
##            seasonal    trend    remainder
## Jan 1920 -9.3471980 49.68067  0.266525379
## Feb 1920 -9.8552496 49.54552  1.109728805
## Mar 1920 -6.8533008 49.41037  1.842931803
## Apr 1920 -2.7634710 49.32862  0.134848770
## May 1920  3.5013569 49.24688  1.351767558
## Jun 1920  8.9833032 49.21027  0.306425938
## Jul 1920 12.8452501 49.17367 -4.318916345
## Aug 1920 11.4763813 49.13389 -4.210271506
## Sep 1920  7.4475114 49.09411 -2.241625601
## Oct 1920  0.4736899 49.15198  0.874331161
## Nov 1920 -6.4301309 49.20984  0.120287167
## Dec 1920 -9.4781423 49.49282 -0.214674402
## Jan 1921 -9.3471980 49.77579  3.771408356
## Feb 1921 -9.8552496 50.08132 -0.426073148
## Mar 1921 -6.8533008 50.38686  1.566444922
## Apr 1921 -2.7634710 50.49363 -0.730156884
## May 1921  3.5013569 50.60040 -0.001756869
## Jun 1921  8.9833032 50.50241 -0.785710552
## Jul 1921 12.8452501 50.40441  3.050335100
## Aug 1921 11.4763813 50.15065 -1.727027471
## Sep 1921  7.4475114 49.89688 -0.344388977
## Oct 1921  0.4736899 49.62951  4.096796830
## Nov 1921 -6.4301309 49.36215 -3.232018119
## Dec 1921 -9.4781423 49.04873  3.229417050
## Jan 1922 -9.3471980 48.73530 -1.888103454
## Feb 1922 -9.8552496 48.33172  0.223530715
## Mar 1922 -6.8533008 47.92814 -1.574835542
## Apr 1922 -2.7634710 47.62971 -2.766243150
## May 1922  3.5013569 47.33129  4.867351062
## Jun 1922  8.9833032 47.33932  1.477377256
## Jul 1922 12.8452501 47.34735 -3.392597215
## Aug 1922 11.4763813 47.54030 -4.716683254
## Sep 1922  7.4475114 47.73326 -0.880768228
## Oct 1922  0.4736899 47.85392 -1.227614475
## Nov 1922 -6.4301309 47.97459  0.255538521
## Dec 1922 -9.4781423 48.05637  3.121775401
## Jan 1923 -9.3471980 48.13814  3.009056606
## Feb 1923 -9.8552496 48.28185  1.673400681
## Mar 1923 -6.8533008 48.42556  1.327744328
## Apr 1923 -2.7634710 48.33263  0.230840836
## May 1923  3.5013569 48.23970 -2.541060836
## Jun 1923  8.9833032 47.96007 -4.243372853
## Jul 1923 12.8452501 47.68044  3.674314465
## Aug 1923 11.4763813 47.45963  0.663988147
## Sep 1923  7.4475114 47.23883 -0.286337106
## Oct 1923  0.4736899 47.22609  1.500223556
## Nov 1923 -6.4301309 47.21335 -4.483216539
```

```
## Dec 1923 -9.4781423 47.28901 -0.210865451
## Jan 1924 -9.3471980 47.36467  1.282529963
## Feb 1924 -9.8552496 47.43546 -0.080209799
## Mar 1924 -6.8533008 47.50625 -2.352949988
## Apr 1924 -2.7634710 47.73513  0.528338204
## May 1924  3.5013569 47.96401  1.734628216
## Jun 1924  8.9833032 48.30128  0.415415073
## Jul 1924 12.8452501 48.63855 -0.683798735
## Aug 1924 11.4763813 48.87239 -2.148774448
## Sep 1924  7.4475114 49.10624 -0.153749096
## Oct 1924  0.4736899 49.23554  0.090766554
## Nov 1924 -6.4301309 49.36485  1.465281448
## Dec 1924 -9.4781423 49.47714  3.601004136
## Jan 1925 -9.3471980 49.58943 -0.242228849
## Feb 1925 -9.8552496 49.62480  0.730453659
## Mar 1925 -6.8533008 49.66017 -2.006864260
## Apr 1925 -2.7634710 49.44799 -1.584518944
## May 1925  3.5013569 49.23581  1.062828193
## Jun 1925  8.9833032 48.96275  1.453949252
## Jul 1925 12.8452501 48.68968  1.965069647
## Aug 1925 11.4763813 48.66249  0.861133228
## Sep 1925  7.4475114 48.63529 -3.082802125
## Oct 1925  0.4736899 48.68778  0.838525570
## Nov 1925 -6.4301309 48.74028 -4.210147490
## Dec 1925 -9.4781423 48.73479 -2.956649644
## Jan 1926 -9.3471980 48.72931 -0.182107471
## Feb 1926 -9.8552496 48.81914  4.436114170
## Mar 1926 -6.8533008 48.90897  1.344335385
## Apr 1926 -2.7634710 49.03851  2.624958422
## May 1926  3.5013569 49.16806 -2.069416721
## Jun 1926  8.9833032 49.18067 -1.363972554
## Jul 1926 12.8452501 49.19328  0.461470949
## Aug 1926 11.4763813 49.11919  1.404424201
## Sep 1926  7.4475114 49.04511  1.007378518
## Oct 1926  0.4736899 49.01380 -2.787487357
## Nov 1926 -6.4301309 48.98248 -0.952353990
## Dec 1926 -9.4781423 48.92428  0.353859697
## Jan 1927 -9.3471980 48.86608 -0.118882290
## Feb 1927 -9.8552496 48.77246 -0.417208438
## Mar 1927 -6.8533008 48.67884  3.474464987
## Apr 1927 -2.7634710 48.64104  1.222435260
## May 1927  3.5013569 48.60324 -0.404592645
## Jun 1927  8.9833032 48.54473 -2.528029817
## Jul 1927 12.8452501 48.48622 -0.931467654
## Aug 1927 11.4763813 48.46091  0.562707318
## Sep 1927  7.4475114 48.43561 -1.183116646
## Oct 1927  0.4736899 48.47250  1.353805163
## Nov 1927 -6.4301309 48.50940  0.220726214
## Dec 1927 -9.4781423 48.58914 -3.911002529
## Jan 1928 -9.3471980 48.66888  1.478313054
## Feb 1928 -9.8552496 48.71949  2.235758285
## Mar 1928 -6.8533008 48.77010  0.883203089
## Apr 1928 -2.7634710 48.80480  1.258669662
## May 1928  3.5013569 48.83951 -1.440861943
```

```
## Jun 1928  8.9833032 48.69762 -1.280921267
## Jul 1928 12.8452501 48.55573  0.799018744
## Aug 1928 11.4763813 48.20763  0.815985704
## Sep 1928  7.4475114 47.85953  0.092953729
## Oct 1928  0.4736899 47.61843  2.107877490
## Nov 1928 -6.4301309 47.37733  2.052800495
## Dec 1928 -9.4781423 47.33335 -0.555212465
## Jan 1929 -9.3471980 47.28938 -3.142181098
## Feb 1929 -9.8552496 47.35578 -6.200531177
## Mar 1929 -6.8533008 47.42218  0.431118317
## Apr 1929 -2.7634710 47.62278 -0.959311653
## May 1929  3.5013569 47.82338  1.775260197
## Jun 1929  8.9833032 48.19582 -0.279118205
## Jul 1929 12.8452501 48.56825  1.086502728
## Aug 1929 11.4763813 48.90395 -0.080335161
## Sep 1929  7.4475114 49.23966  3.112828016
## Oct 1929  0.4736899 49.35027 -0.623963046
## Nov 1929 -6.4301309 49.46089 -0.130754863
## Dec 1929 -9.4781423 49.44736  1.930783783
## Jan 1930 -9.3471980 49.43383  1.513366755
## Feb 1930 -9.8552496 49.38822 -2.432972451
## Mar 1930 -6.8533008 49.34261 -1.289312084
## Apr 1930 -2.7634710 49.29319  0.370279597
## May 1930  3.5013569 49.24377 -1.545126902
## Jun 1930  8.9833032 49.16499  2.251710214
## Jul 1930 12.8452501 49.08620 -1.831453334
## Aug 1930 11.4763813 48.97296  1.150658078
## Sep 1930  7.4475114 48.85972  0.692770554
## Oct 1930  0.4736899 48.78043  1.645879449
## Nov 1930 -6.4301309 48.70114  0.728987588
## Dec 1930 -9.4781423 48.62085 -0.342708316
## Jan 1931 -9.3471980 48.54056 -2.093359893
## Feb 1931 -9.8552496 48.38319 -0.127942861
## Mar 1931 -6.8533008 48.22583 -2.972526256
## Apr 1931 -2.7634710 48.12257  1.140904226
## May 1931  3.5013569 48.01931  1.979336529
## Jun 1931  8.9833032 48.14283  1.273868208
## Jul 1931 12.8452501 48.26635 -0.511600778
## Aug 1931 11.4763813 48.40974 -1.686116919
## Sep 1931  7.4475114 48.55312 -2.200631996
## Oct 1931  0.4736899 48.54088 -2.414570882
## Nov 1931 -6.4301309 48.52864  3.401489476
## Dec 1931 -9.4781423 48.53430  1.543840385
## Jan 1932 -9.3471980 48.53996  3.207235620
## Feb 1932 -9.8552496 48.66617 -0.410916656
## Mar 1932 -6.8533008 48.79237 -1.639069359
## Apr 1932 -2.7634710 48.83691 -1.473441358
## May 1932  3.5013569 48.88145 -1.482811536
## Jun 1932  8.9833032 48.82173 -0.805030457
## Jul 1932 12.8452501 48.76200  0.492749958
## Aug 1932 11.4763813 48.83146  3.192154743
## Sep 1932  7.4475114 48.90093 -0.048439406
## Oct 1932  0.4736899 49.12176 -2.295447972
## Nov 1932 -6.4301309 49.34259  0.687542706
```

```
## Dec 1932 -9.4781423 49.59996  1.678183472
## Jan 1933 -9.3471980 49.85733 -4.310131435
## Feb 1933 -9.8552496 50.12104 -0.965788581
## Mar 1933 -6.8533008 50.38475  0.968553847
## Apr 1933 -2.7634710 50.55788  0.905594559
## May 1933  3.5013569 50.73101 -0.032362909
## Jun 1933  8.9833032 50.69026  1.126437333
## Jul 1933 12.8452501 50.64951  2.005236911
## Aug 1933 11.4763813 50.46236  2.961261164
## Sep 1933  7.4475114 50.27520  2.377286481
## Oct 1933  0.4736899 50.02482 -0.298512893
## Nov 1933 -6.4301309 49.77444 -1.244313023
## Dec 1933 -9.4781423 49.60356 -4.325422241
## Jan 1934 -9.3471980 49.43269 -0.685487132
## Feb 1934 -9.8552496 49.38494 -1.329690994
## Mar 1934 -6.8533008 49.33720 -2.083895283
## Apr 1934 -2.7634710 49.49963  0.163845195
## May 1934  3.5013569 49.66206  0.236587495
## Jun 1934  8.9833032 49.99133  0.625370019
## Jul 1934 12.8452501 50.32060  3.334151879
## Aug 1934 11.4763813 50.58387 -1.660249972
## Sep 1934  7.4475114 50.84714  0.905349242
## Oct 1934  0.4736899 50.88993 -0.163620448
## Nov 1934 -6.4301309 50.93272 -1.702590896
## Dec 1934 -9.4781423 50.87092  4.407220183
## Jan 1935 -9.3471980 50.80912 -1.461924412
## Feb 1935 -9.8552496 50.75825  1.696998728
## Mar 1935 -6.8533008 50.70738 -0.354078560
## Apr 1935 -2.7634710 50.57994 -0.716471023
## May 1935  3.5013569 50.45250 -3.953861666
## Jun 1935  8.9833032 50.14917  1.367526515
## Jul 1935 12.8452501 49.84584  1.908914032
## Aug 1935 11.4763813 49.54007  2.983549209
## Sep 1935  7.4475114 49.23430  0.118185451
## Oct 1935  0.4736899 49.02909 -0.902784869
## Nov 1935 -6.4301309 48.82389  1.806244053
## Dec 1935 -9.4781423 48.64539 -2.767248190
## Jan 1936 -9.3471980 48.46689 -1.819696106
## Feb 1936 -9.8552496 48.35225 -3.496996331
## Mar 1936 -6.8533008 48.23760  2.615703017
## Apr 1936 -2.7634710 48.30761 -1.644138274
## May 1936  3.5013569 48.37762  0.821022256
## Jun 1936  8.9833032 48.58645  1.030250876
## Jul 1936 12.8452501 48.79527 -1.640521169
## Aug 1936 11.4763813 48.96677  0.656852958
## Sep 1936  7.4475114 49.13826  1.514228150
## Oct 1936  0.4736899 49.21434 -0.088032903
## Nov 1936 -6.4301309 49.29043 -1.260294714
## Dec 1936 -9.4781423 49.33815  1.439989280
## Jan 1937 -9.3471980 49.38588  0.761317600
## Feb 1937 -9.8552496 49.41158  1.443666982
## Mar 1937 -6.8533008 49.43728 -4.183984063
## Apr 1937 -2.7634710 49.39905  0.764425401
## May 1937  3.5013569 49.36081  1.237836686
```

```
## Jun 1937  8.9833032 49.32833  0.288370135
## Jul 1937 12.8452501 49.29585 -0.741097080
## Aug 1937 11.4763813 49.39591  0.927705554
## Sep 1937  7.4475114 49.49598 -0.643490748
## Oct 1937  0.4736899 49.62722  0.799089443
## Nov 1937 -6.4301309 49.75846 -1.928331124
## Dec 1937 -9.4781423 49.75732 -3.179182522
## Jan 1938 -9.3471980 49.75619  1.691010406
## Feb 1938 -9.8552496 49.73327  1.321982868
## Mar 1938 -6.8533008 49.71035  4.442954903
## Apr 1938 -2.7634710 49.78815 -0.424682121
## May 1938  3.5013569 49.86596 -0.967317324
## Jun 1938  8.9833032 49.91577  0.100921835
## Jul 1938 12.8452501 49.96559 -3.210839669
## Aug 1938 11.4763813 49.89688 -0.973257797
## Sep 1938  7.4475114 49.82816 -0.275674861
## Oct 1938  0.4736899 49.79656  0.429748533
## Nov 1938 -6.4301309 49.76496  4.465171170
## Dec 1938 -9.4781423 49.79001 -1.111866934
## Jan 1939 -9.3471980 49.81506 -1.067860710
## Feb 1939 -9.8552496 49.78783  0.967421826
## Mar 1939 -6.8533008 49.76060 -0.507296065
## Apr 1939 -2.7634710 49.68448  0.878994770
## May 1939  3.5013569 49.60836 -0.709712574
## Jun 1939  8.9833032 49.51780 -0.501104910
## Jul 1939 12.8452501 49.42725 -1.572497911
## Aug 1939 11.4763813 49.33655  0.987068518
## Sep 1939  7.4475114 49.24585  1.506636011
## Oct 1939  0.4736899 49.16331 -2.936997128
## Nov 1939 -6.4301309 49.08076  3.949368976
## Dec 1939 -9.4781423 49.00510 -1.726954804
```

15.2 Decomposition Demo

Extracting Components

"object$component""

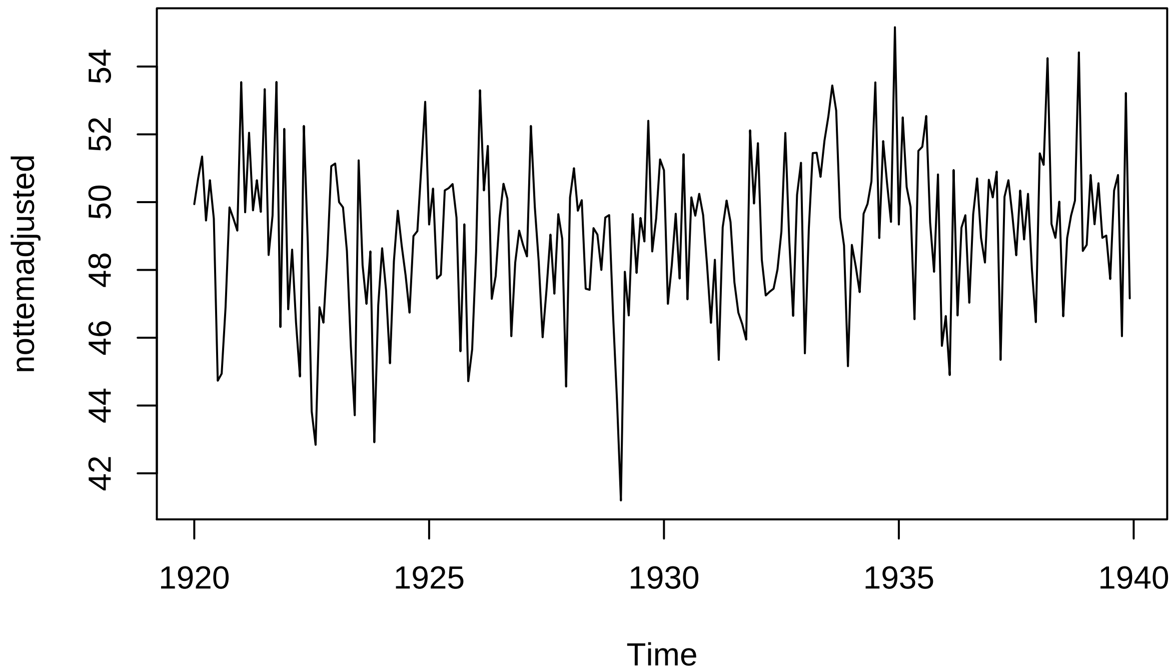
"myobject$seasonal""

Each component can be extracted and then used for creating an adjusted time series

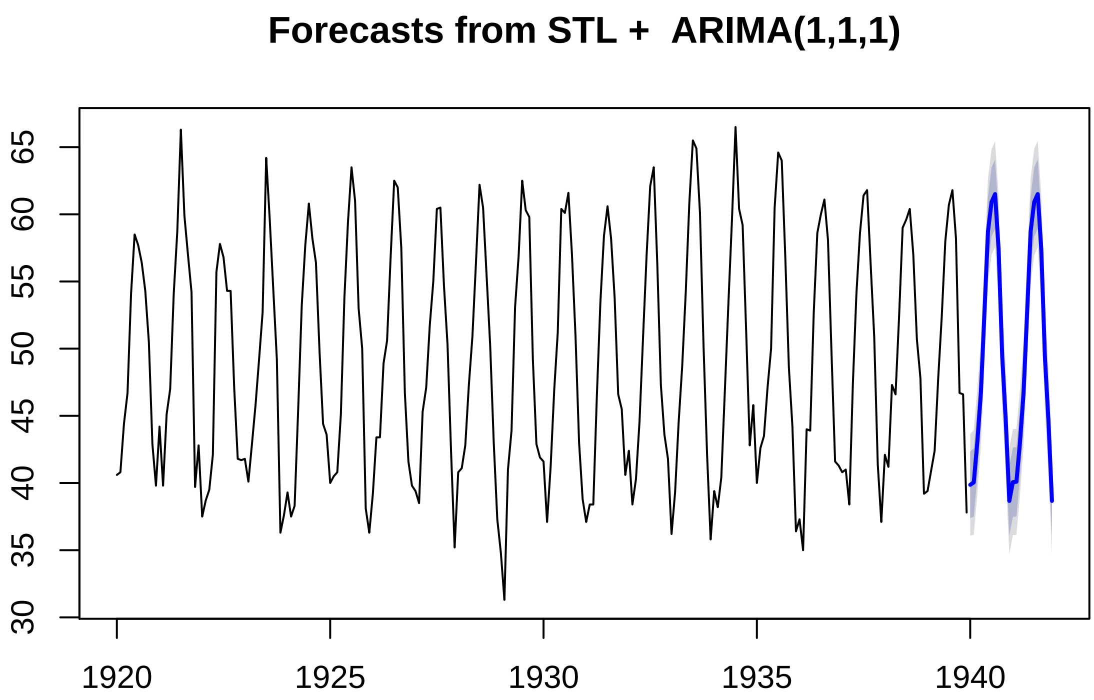
```r
#subtract the seasonality component from the dataset
mynottem=decompose(nottem, "additive")

class(mynottem)
```

```
## [1] "decomposed.ts"
```

```r
# we are subtracting the seasonal element

nottemadjusted = nottem - mynottem$seasonal

# getting a plot
plot(nottemadjusted)
```




```r
# a decomposed time series to forecast
# library(forecast)

plot(stlf(nottem, method = "arima"))
```




```r
# I can  use other method like naive or drift, I can also add h for length of forecast etc.
```

15.3 Exercise: Decomposition

1. Get and plot the dataset ‘AirPassengers’ of R Base (monthly data with lots of patterns)

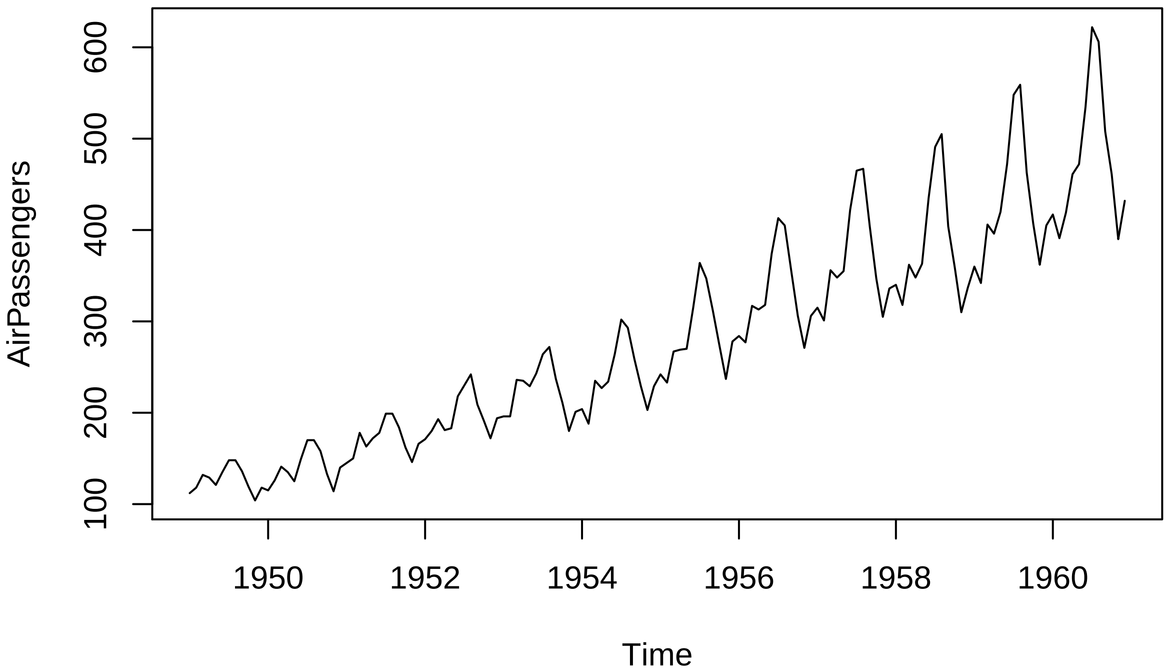
2. Set up two decomposition models with decompose() - alternative: stl()

- Additive model - mymodel1
- Multiplicative model - mymodel2

3. Plot and compare the two models
4. Produce and plot a time series of the seasonally adjusted mymodel1 (that means that there should only be the trend and the remained left in the data set)

- compare to the original dataset

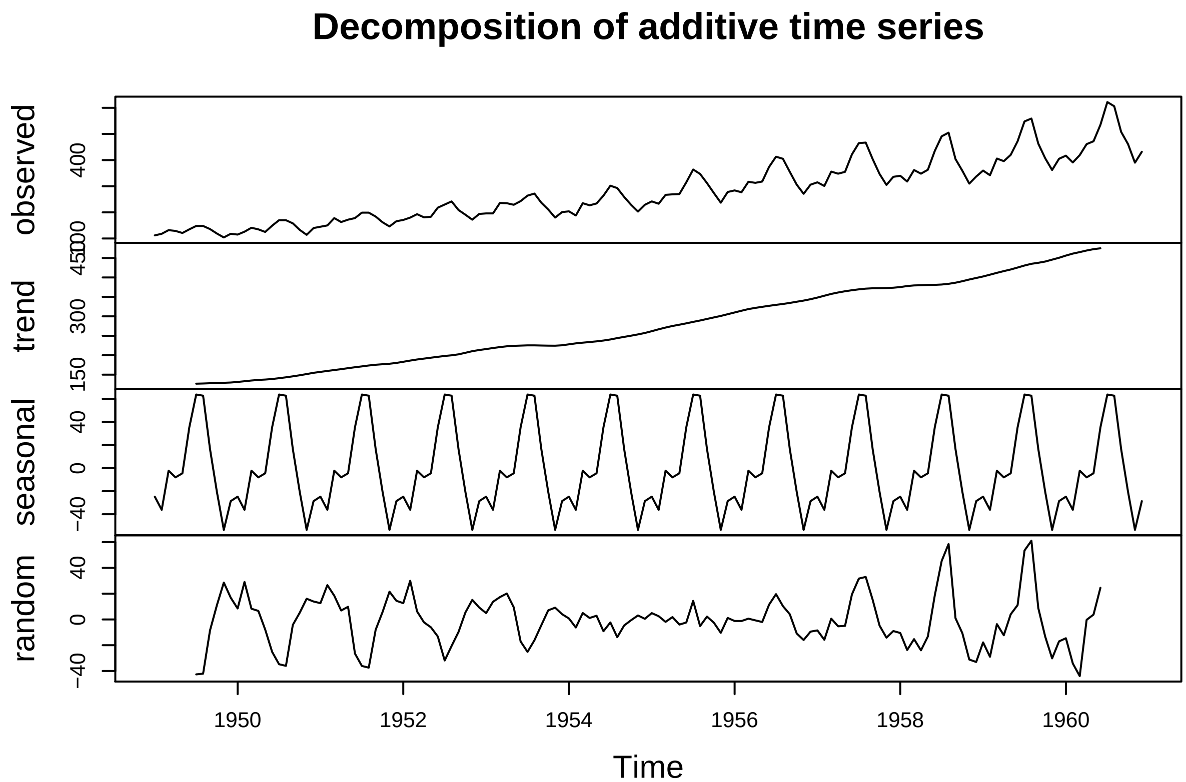
```
plot(AirPassengers)
```




```
#trend present as well as seasonal pattern, seasonal amplitude (difference between max and min) increas
frequency(AirPassengers)
```

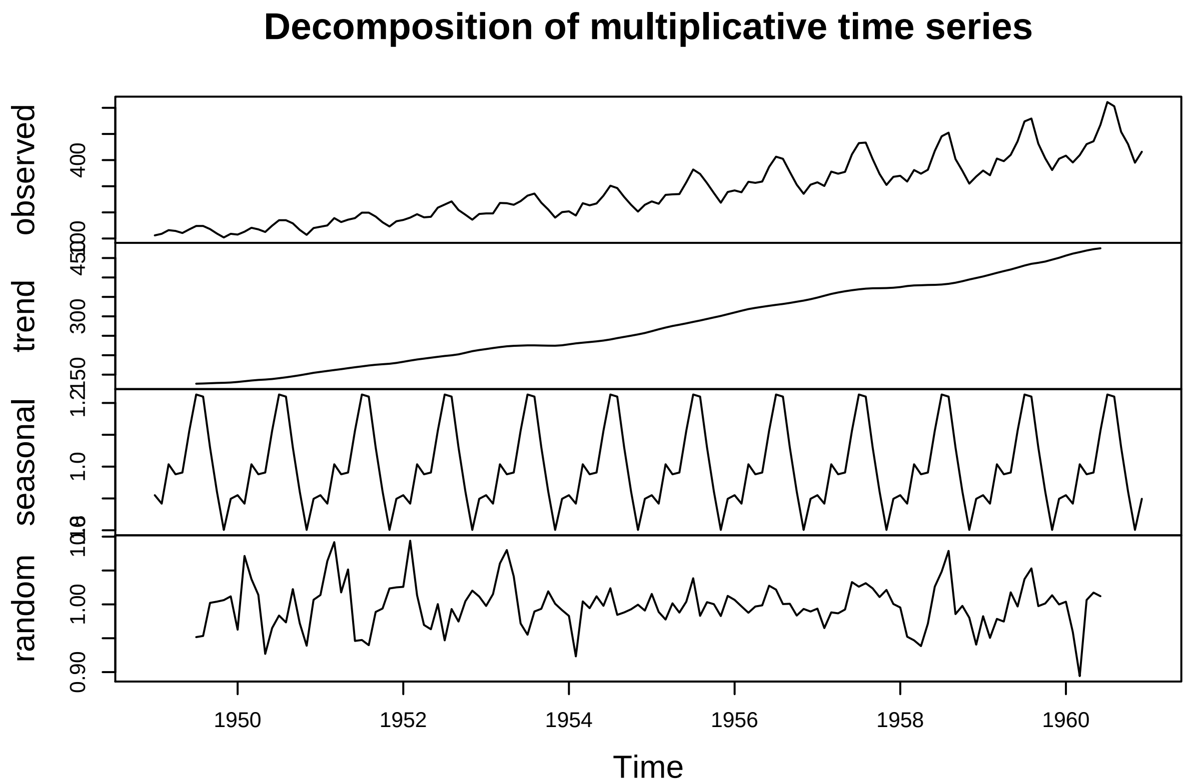
```
## [1] 12
```

```
mymodel1 = decompose(AirPassengers, type = "additive")

plot(mymodel1)
```



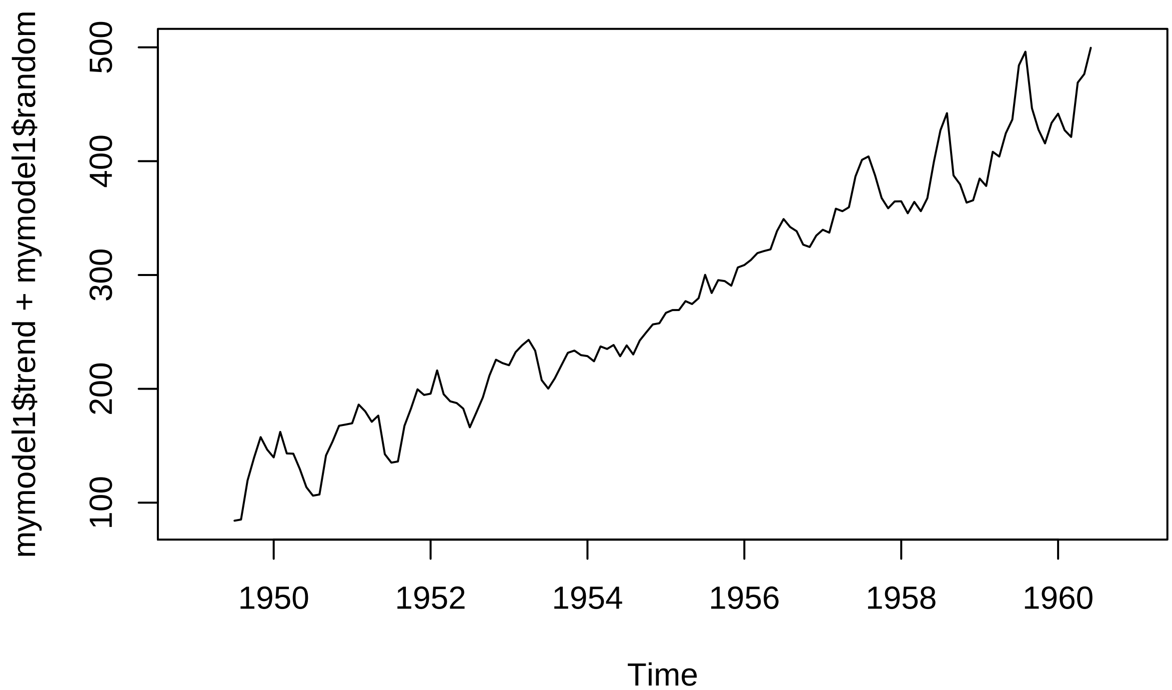

```
# trend is increasing, seasonal pattern in third line, some pattern left in reminder (last line) and th
```

```
mymodel2= decompose(AirPassengers, type= "multiplicative")

plot(mymodel2)
```




```
# last line is different to previous model, it looks more random (although there is still some pattern
```

```r
plot(mymodel1$trend + mymodel1$random)
```



Overall models are not ideal and data requires more sophisticated model

16 Simple Moving Average

Smoothing: getting the dataset closer to the center by evening out the highs and the lows => decreasing the impact of extreme values

Classic smoother: simple moving average

- Widely used in science and finance (trading)

How does a SMA work?

- Define the number of observations to use and take their average
- Period = successive values of a time series

```
#library("TTR")
# in order to identify trends, we can use smoothers
# like a simple moving avg
# n identifies the order of the SMA - you can experiment with this parameter
```

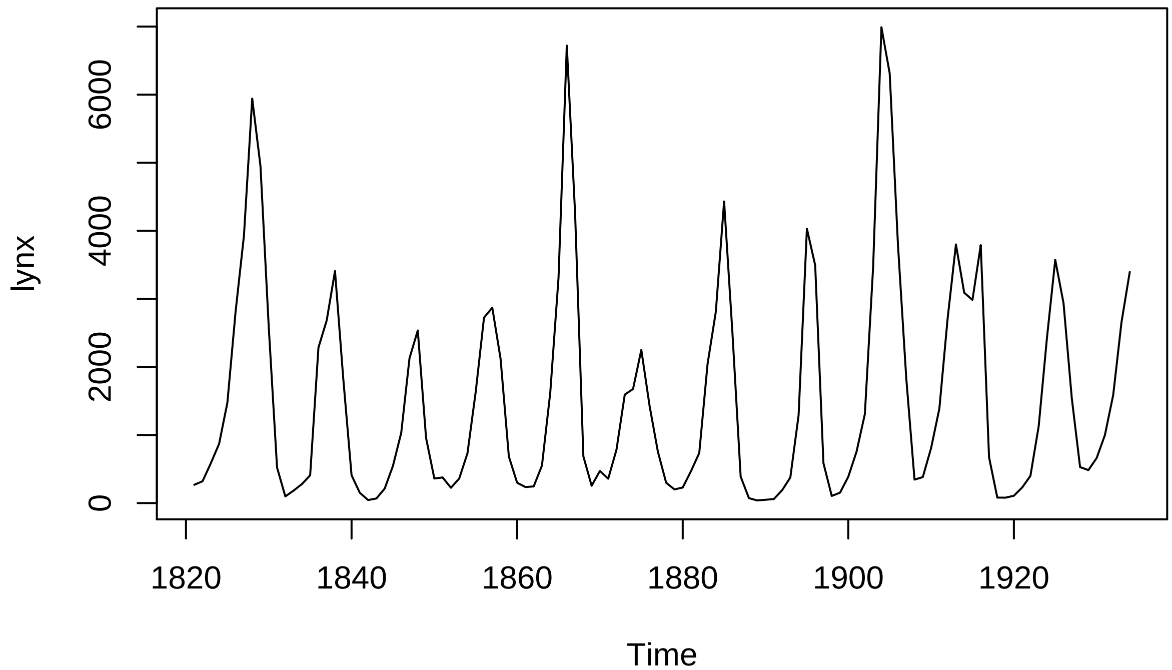
```
x=c(1,2,3,4,5,6,7)

SMA(x, n=3)
```

```
## [1] NA NA  2  3  4  5  6
```

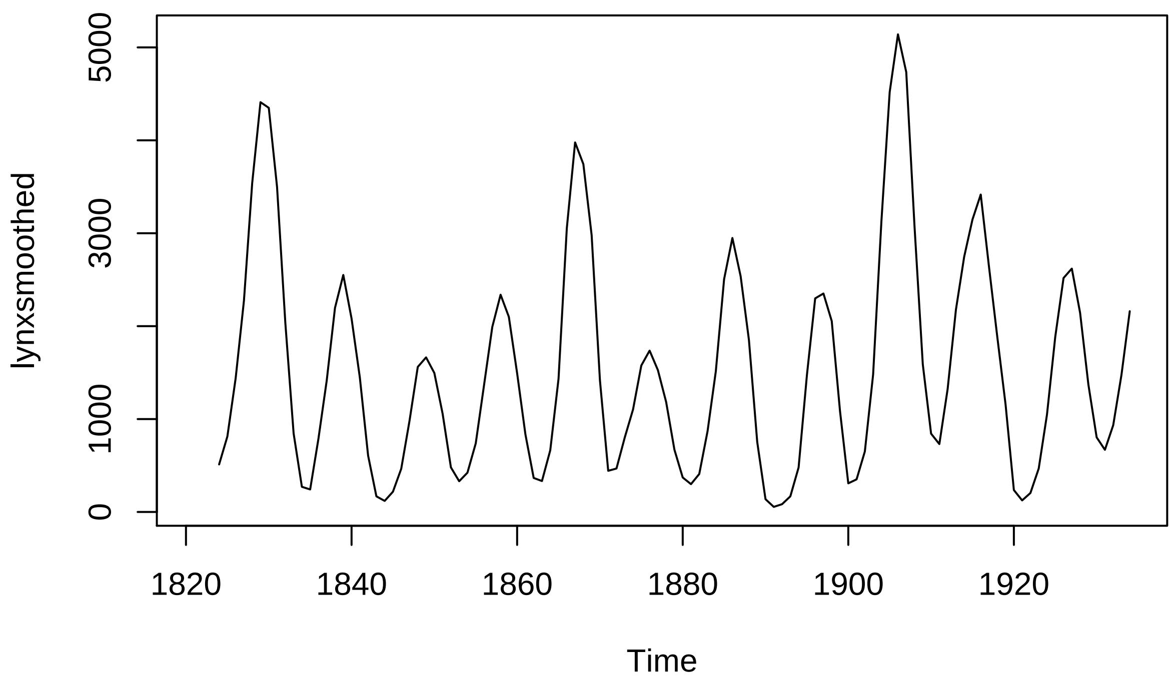
```
lynxsmoothed = SMA(lynx, n=4); lynxsmoothed
```

```
## Time Series:
## Start = 1821
## End = 1934
## Frequency = 1
##   [1]      NA      NA      NA  511.50  813.00 1438.00 2273.75 3541.75
##   [9] 4410.50 4349.50 3498.25 2037.00  845.50  271.00  242.50  789.25
##  [17] 1414.50 2197.00 2550.75 2081.75 1448.25  607.25  168.25  119.25
##  [25]  218.00  465.00  980.25 1561.00 1663.75 1495.75 1057.75  480.00
##  [33]  330.75  423.25  738.50 1363.50 1991.25 2338.25 2099.75 1493.25
##  [41]  834.50  366.00  333.00  664.00 1432.75 3051.75 3977.25 3743.25
##  [49] 2979.25 1417.25  443.25  467.50  802.25 1103.00 1576.25 1736.75
##  [57] 1527.25 1183.00  670.50  371.25  299.50  408.75  869.00 1514.50
##  [65] 2505.00 2948.75 2535.50 1851.00  753.00  137.50   55.00   83.75
##  [73]  168.25  479.00 1472.00 2298.75 2351.25 2054.50 1085.00  308.00
##  [81]  350.75  651.25 1479.25 3130.25 4519.00 5140.75 4733.50 3072.00
##  [89] 1589.25  842.75  730.75 1322.75 2177.25 2748.00 3147.25 3416.50
##  [97] 2635.00 1882.50 1156.25  235.75  124.50  204.00  467.00 1048.00
## [105] 1884.25 2518.25 2619.50 2143.75 1371.50  803.25  669.00  934.25
## [113] 1477.25 2160.75
```

```
plot(lynx)
```




```
plot(lynxsmoothed)
```




```
#higher n woudl give me a smoother result

# This method work best with non-seasonal dat and is ideal for getting the general trend and removing w
#Basically, the higher n is the less white noice you will encounter in your data
```

17 Exponential Smoothing with ETS

Describe the time series with three parameters

- Error - additive, multiplicative (x>0)
- Trend - non-present, additive, multiplicative
- Seasonality - non-present, additive, multiplicative

Values are either summed up in additive model, multiplied in multiplicative model or omitted

R functions:

- Simple exponential smoothing - ses(): for datasets withou trend and seasonality
- Holt linear exponential smoothing model - holt(): for datasets with a trend and without seasonality
- Argument 'damped' to damp down the trend over time
- Holt-Winters seasonal exponential smoothing - hw(): for data with trend and seasonal component + a damping parameter

=> Above models are set manually

Automated model selection via ets() (also library 'forecast') Model selection based on information criteria

Smoothing coefficients to manage the weighting based on the timestamp

- Reactive model relies heavily on recent data - high coefficient ~1
- Smooth model - low coefficient ~0 (which means a more round curves with even older data being quite important for the forecasted values)

Coefficients:

Alpha: Initial level Beta: trend Gama: seasonality Damped parameter

Required argument for ets():data Argument 'model' for pre-selecting a model

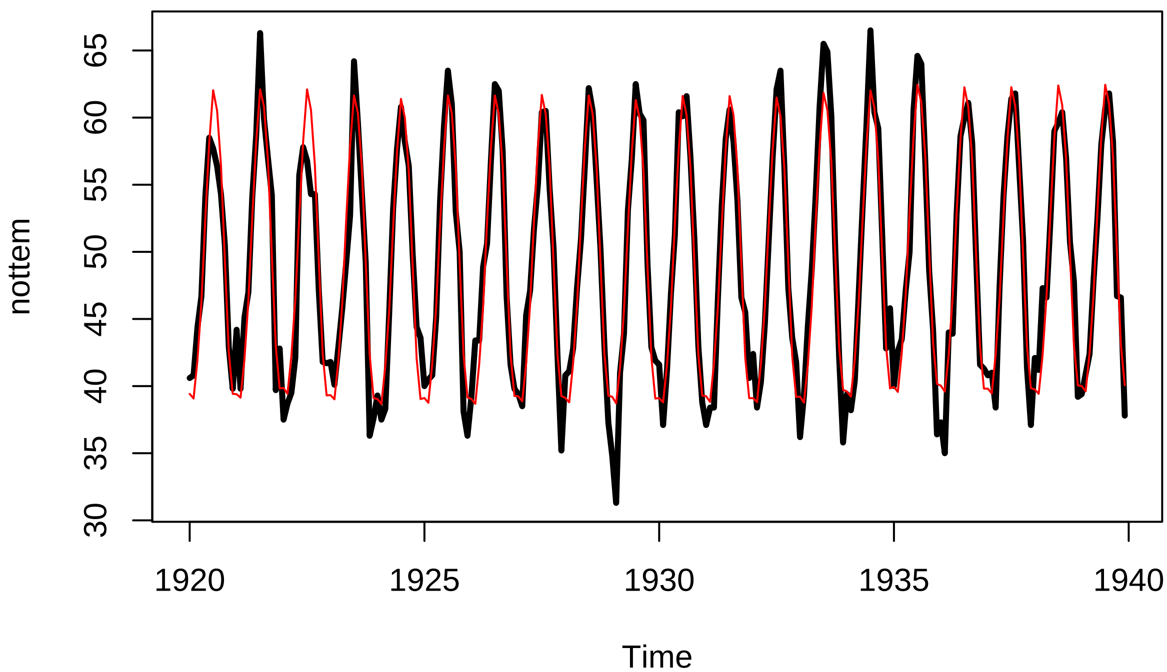
- Default 'ZZZ': auto-selection of the three components; Additive 'A', multiplicative 'M', non-present 'N'
- Coefficients and boundaries can also be pre-set

```
#library(forecast)
etsmodel = ets(nottem); etsmodel
```

```
## ETS(A,N,A)
##
## Call:
##  ets(y = nottem)
##
##   Smoothing parameters:
##     alpha = 0.0246
##     gamma = 1e-04
##
##   Initial states:
##     l = 48.92
##     s=-9.4652 -6.3677 0.4782 7.5653 11.5359 12.9304
##            9.042 3.4328 -2.7475 -7.0273 -9.8768 -9.5001
##
##   sigma:  2.2504
##
##      AIC     AICc      BIC
## 1734.682 1736.825 1786.891
```

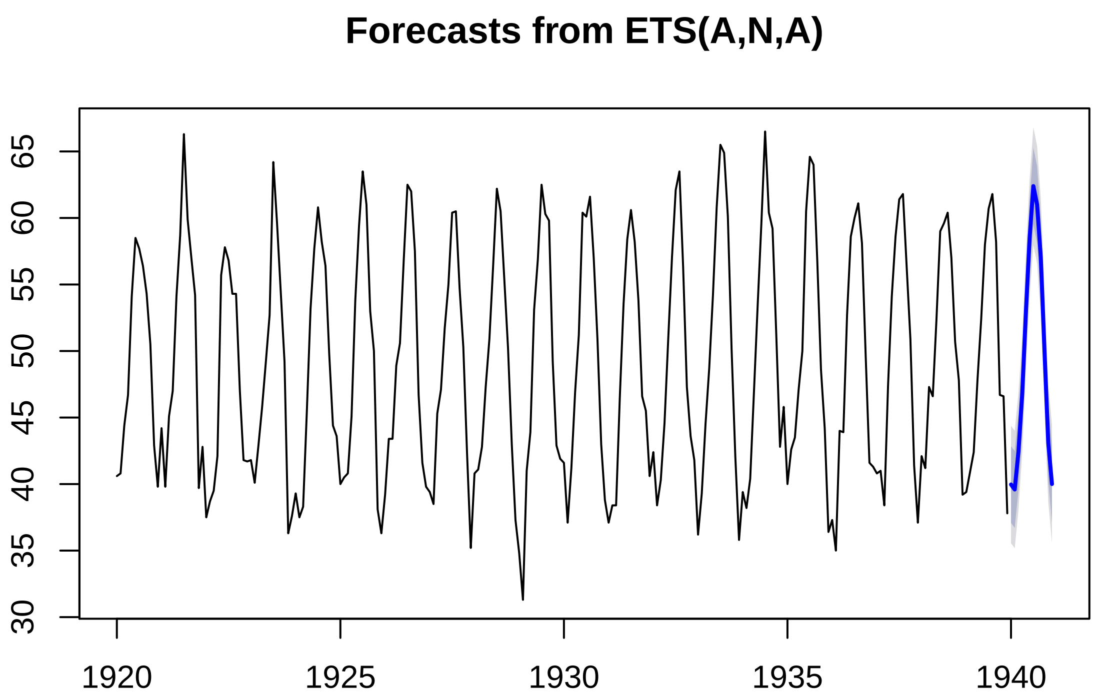
```
# ets(A,N,A) => 3 components (error, trend and seasonality)

#Smoothing coefficiets => closer they are to 1, the more the model relies on recent data, closer to 0,

# Alpha = 0.0246; gamma is almost 0
```

```r
plot(nottem, lwd=3)
  lines(etsmodel$fitted, col="red")
```



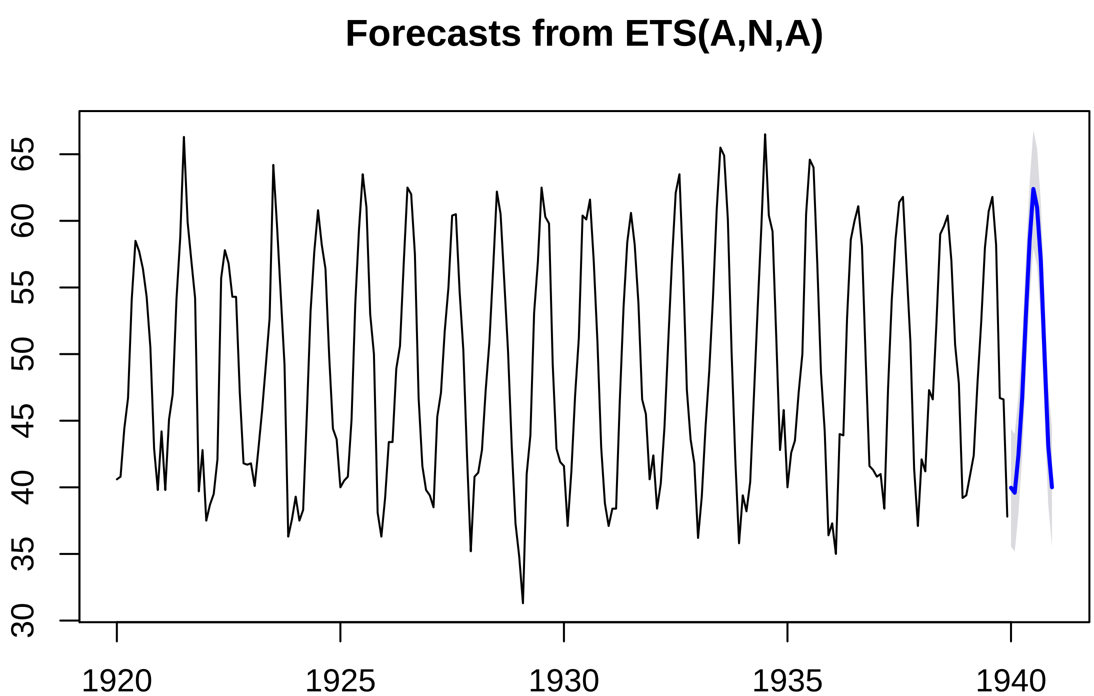

```r
# The fitted model is ploted on the top of the original dataset; the fitted values are quite close to d

# Looking at Initial states:
# l = we need the initial level to calculate the error rate
# s = 12 months' we need seasonal values to calculate each initial seasonal state
# sigma = accuracy indicator of the model
# 3 values (AIC, AICc, BIC) = basically the same as with an ARIMA model, model quality indiator and qua
```

```r
plot(forecast(etsmodel, h=12))
```




```r
# extra year in blue with prediction interval
```

```r
plot(forecast(etsmodel, h=12, level=95))
```



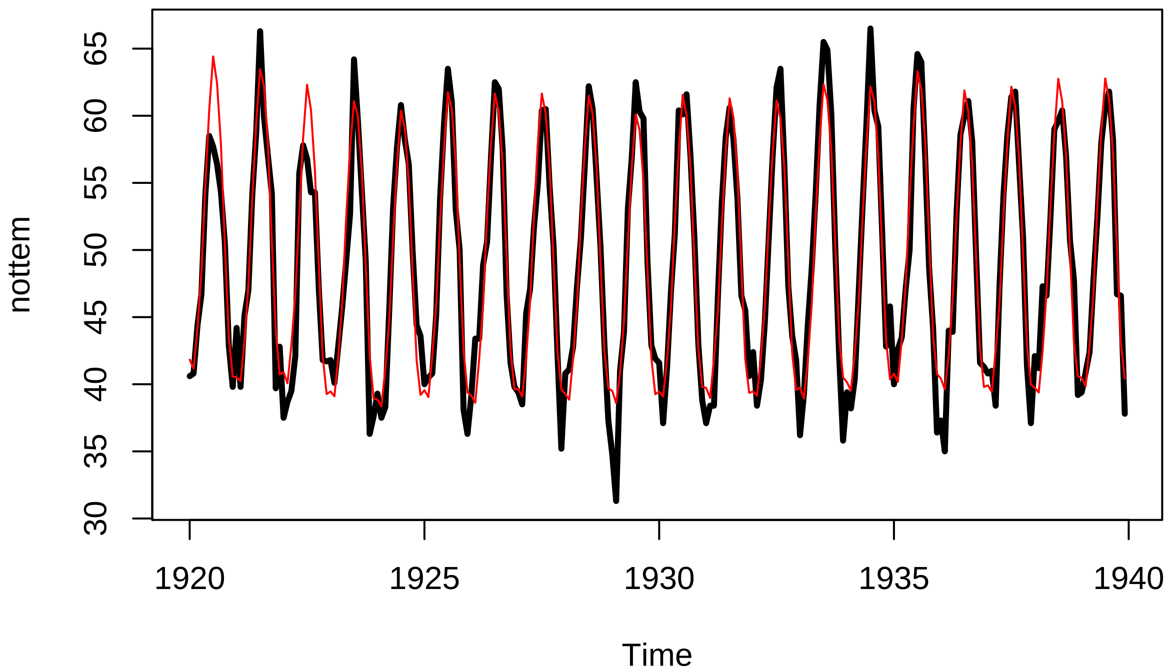

```r
etsmodmult = ets(nottem, model = "MZM"); etsmodmult
```

```
## ETS(M,N,M)
##
## Call:
##  ets(y = nottem, model = "MZM")
##
```

```
##   Smoothing parameters:
##     alpha = 0.0792
##     gamma = 1e-04
##
##   Initial states:
##     l = 51.5207
##     s=0.8119 0.8697 1.0073 1.1507 1.2325 1.2618
##            1.1812 1.0702 0.9413 0.8604 0.8008 0.812
##
##   sigma:  0.0497
##
##      AIC     AICc      BIC
## 1766.776 1768.919 1818.986
```

```
# multiplicative model; parameters have been adjusted, looking at last 3 indicators, we can see that th
```

```
plot(nottem, lwd =3)
lines(etsmodmult$fitted, col="red")
```



18 ARIMA Model

Autoregressive Integrated Moving Average Modeling univariate time series Crucial tool for an analyst

18.1 Theory

- Following code and text is focus on univariate, non-seasonal ARIMA Models

ARIMA (p,d,q)

ARIMA contains 3 elements =>

- AR - Autoregressive part: p

- I - Integration, degree of differencing: d
- MA - Moving average part: q

Parameters: Integers denoting the grade or order of the three parts

Calculating the AR and MA parts requires a stationary time series = Differencing done by the model function or manual differencing with diff(), part of element d

How to calculate the three parameters?

arima()

=> estimating the parameters manually by using ACF and PACF plots (R Base)

auto.arima()

=> R calculates the parameters automatically and chooses a suitable model (library forecast)

How to read an ARIMA model?

Summation of lags = autoregressive part Summation of forecasting errors = moving average part Coefficient: Determines the importance of a specific lag

AR(1) or ARIMA (1,0,0): first order(lag) of AR AR(2) or ARIMA (2,0,0): second order of AR MA(1) or ARIMA (0,0,1): first order of MA

AR(1) or ARIMA (1,0,0)

Yt=c+ θ Yt-1+et

The observed value(Yt) at time point t consists of

- the constant (c) plus
- the value of the previous time point (Yt-1) multiplied by a coefficient */theta* plus
- the error term of time point t (et)

ARMA(1,1) or ARIMA (1,0,1) => model was extended with a forecast error term

Yt = c + θ Yt-1 + θ et-1 + et

Extra step: Forecast error term for the first lag (et-1) multiplied by the coefficient θ Forecast error at t: The difference between the actual and the forecast value

ARIMA (0,1,0)

Random walk: the mean is not constant, which is required for a forecast Stationary dataset: dataset with constant mean

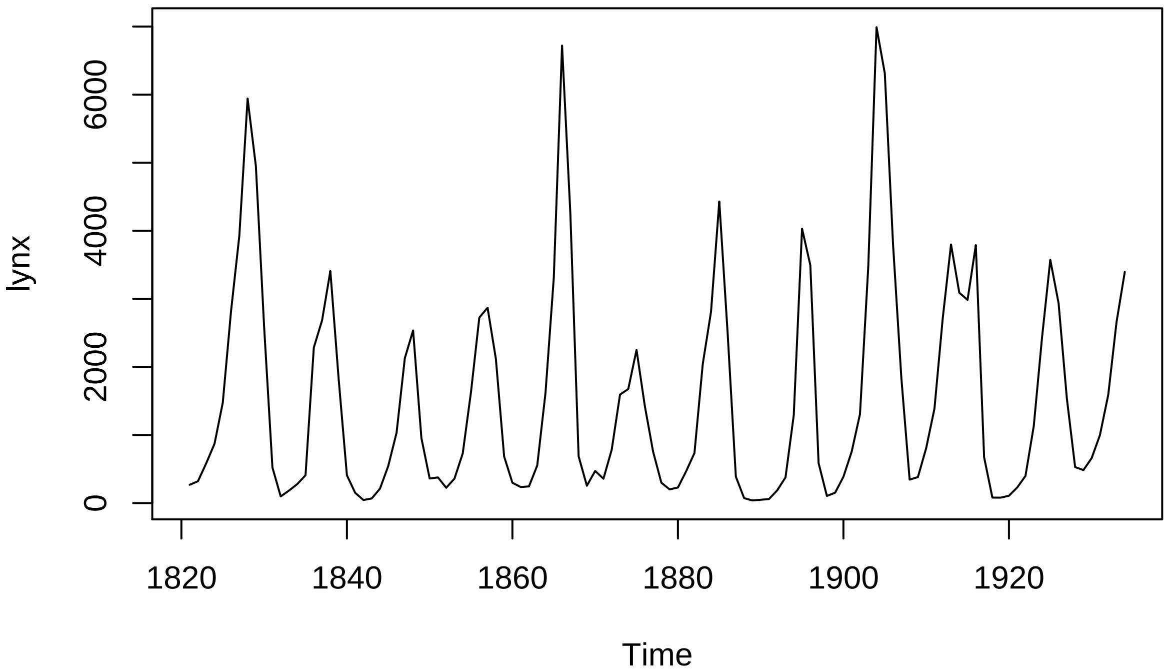
Differencing: Yt - Yt-1 = c + et

The expected value (Yt) minus the previous one (Yt-1) equals the constant (c) multiplied by the error term (et) at time point t

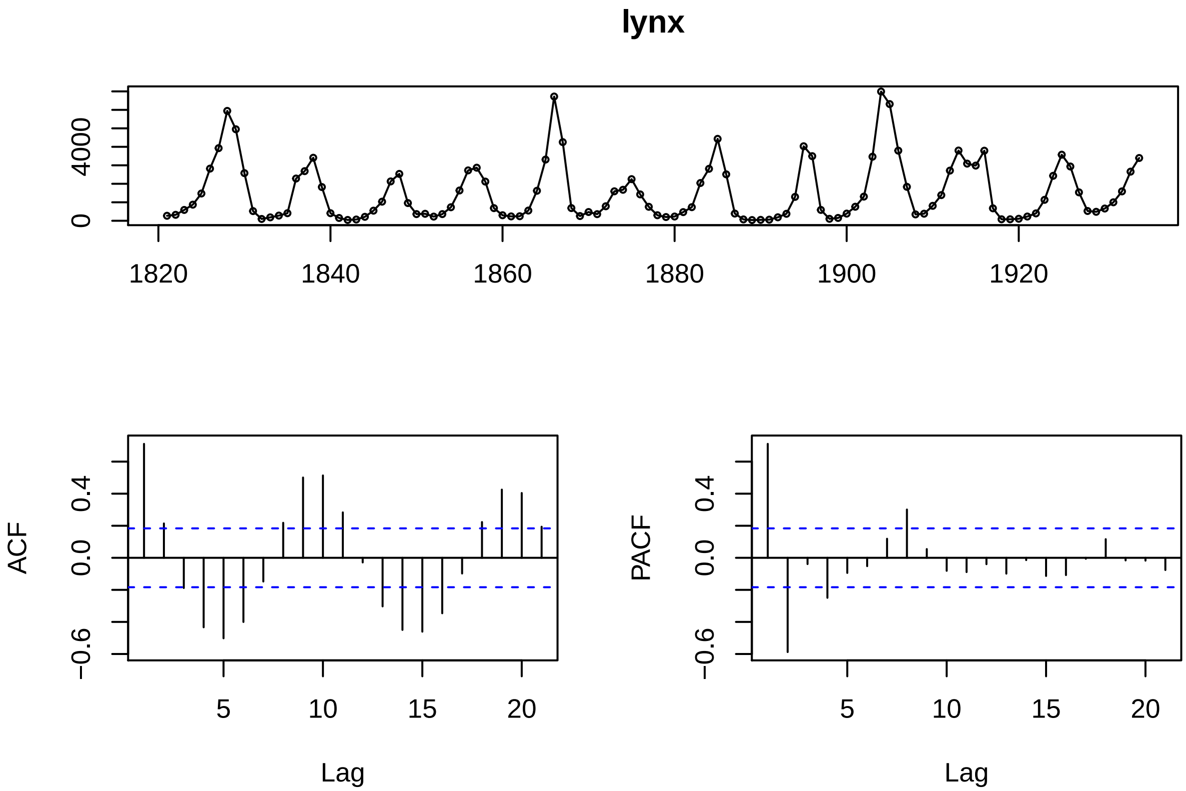
18.2 Auto.arima

It is well known and popular, part of forecast library It can have low quality and performance, danger of producing uninformed, low quality models However, it is good starting point to time series analysis

```
plot(lynx)
```




```r
# we can see cyclical pulse, no seasonality, an autoregressive dataset (Lynx trappings of prior years i

#library(forecast)

# How to confirm autoregression?
tsdisplay(lynx) # see ACF plot
```



```r
# looking at first plot, do we need differencing with parameter d? We need differencing when time series

# It will be totally plausible if the model does not contain a D parameter. I can always use ADF test f
```

```r
auto.arima(lynx) # the basic version
```

```
## Series: lynx
## ARIMA(2,0,2) with non-zero mean
##
## Coefficients:
##          ar1      ar2      ma1      ma2       mean
##       1.3421  -0.6738  -0.2027  -0.2564  1544.4039
## s.e.  0.0984   0.0801   0.1261   0.1097   131.9242
##
## sigma^2 estimated as 761965:  log likelihood=-932.08
## AIC=1876.17   AICc=1876.95   BIC=1892.58
```

```r
auto.arima(lynx, trace=T)
```

```
##
##  ARIMA(2,0,2) with non-zero mean : 1876.952
##  ARIMA(0,0,0) with non-zero mean : 2006.724
##  ARIMA(1,0,0) with non-zero mean : 1927.209
##  ARIMA(0,0,1) with non-zero mean : 1918.165
##  ARIMA(0,0,0) with zero mean     : 2080.721
##  ARIMA(1,0,2) with non-zero mean : 1888.757
##  ARIMA(3,0,2) with non-zero mean : 1878.603
##  ARIMA(2,0,1) with non-zero mean : 1880.014
##  ARIMA(2,0,3) with non-zero mean : Inf
##  ARIMA(1,0,1) with non-zero mean : 1891.442
##  ARIMA(3,0,3) with non-zero mean : 1881.515
##  ARIMA(2,0,2) with zero mean     : 1905.595
##
##  Best model: ARIMA(2,0,2) with non-zero mean
```

```
## Series: lynx
## ARIMA(2,0,2) with non-zero mean
##
## Coefficients:
##          ar1      ar2      ma1      ma2       mean
##       1.3421  -0.6738  -0.2027  -0.2564  1544.4039
## s.e.  0.0984   0.0801   0.1261   0.1097   131.9242
##
## sigma^2 estimated as 761965:  log likelihood=-932.08
## AIC=1876.17   AICc=1876.95   BIC=1892.58
```

```r
# gives me a list of alternative models and I can choose the one with lowest value
```

ARIMA(p,d,q) (P,D,Q)

=> first part is standard model parameters, second part is parameters for the seasonal components

```r
auto.arima(lynx, trace=T,
           stepwise = F,
           approximation = F)
```

```
##
```

```
##  ARIMA(0,0,0) with zero mean     : 2080.721
##  ARIMA(0,0,0) with non-zero mean : 2006.724
##  ARIMA(0,0,1) with zero mean     : 1972.791
##  ARIMA(0,0,1) with non-zero mean : 1918.165
##  ARIMA(0,0,2) with zero mean     : 1925.15
##  ARIMA(0,0,2) with non-zero mean : 1890.428
##  ARIMA(0,0,3) with zero mean     : 1913.118
##  ARIMA(0,0,3) with non-zero mean : 1888.326
##  ARIMA(0,0,4) with zero mean     : 1906.524
##  ARIMA(0,0,4) with non-zero mean : 1889.064
##  ARIMA(0,0,5) with zero mean     : 1908.619
##  ARIMA(0,0,5) with non-zero mean : 1886.754
##  ARIMA(1,0,0) with zero mean     : 1934.647
##  ARIMA(1,0,0) with non-zero mean : 1927.209
##  ARIMA(1,0,1) with zero mean     : 1903.345
##  ARIMA(1,0,1) with non-zero mean : 1891.442
##  ARIMA(1,0,2) with zero mean     : 1903.567
##  ARIMA(1,0,2) with non-zero mean : 1888.757
##  ARIMA(1,0,3) with zero mean     : 1905.59
##  ARIMA(1,0,3) with non-zero mean : 1890.03
##  ARIMA(1,0,4) with zero mean     : 1907.578
##  ARIMA(1,0,4) with non-zero mean : Inf
##  ARIMA(2,0,0) with zero mean     : 1906.685
##  ARIMA(2,0,0) with non-zero mean : 1878.399
##  ARIMA(2,0,1) with zero mean     : 1903.412
##  ARIMA(2,0,1) with non-zero mean : 1880.014
##  ARIMA(2,0,2) with zero mean     : 1905.595
##  ARIMA(2,0,2) with non-zero mean : 1876.952
##  ARIMA(2,0,3) with zero mean     : 1907.963
##  ARIMA(2,0,3) with non-zero mean : Inf
##  ARIMA(3,0,0) with zero mean     : 1903.728
##  ARIMA(3,0,0) with non-zero mean : 1880.512
##  ARIMA(3,0,1) with zero mean     : 1905.587
##  ARIMA(3,0,1) with non-zero mean : 1881.962
##  ARIMA(3,0,2) with zero mean     : Inf
##  ARIMA(3,0,2) with non-zero mean : 1878.603
##  ARIMA(4,0,0) with zero mean     : 1905.899
##  ARIMA(4,0,0) with non-zero mean : 1875.007
##  ARIMA(4,0,1) with zero mean     : Inf
##  ARIMA(4,0,1) with non-zero mean : 1876.407
##  ARIMA(5,0,0) with zero mean     : 1904.543
##  ARIMA(5,0,0) with non-zero mean : 1876.332
```

```
## Series: lynx
## ARIMA(4,0,0) with non-zero mean
##
## Coefficients:
##          ar1      ar2     ar3      ar4       mean
##       1.1246  -0.7174  0.2634  -0.2543  1547.3859
## s.e.  0.0903   0.1367  0.1361   0.0897   136.8501
##
## sigma^2 estimated as 748457:  log likelihood=-931.11
## AIC=1874.22   AICc=1875.01   BIC=1890.64
```

18.3 ARIMA Model Calculations

- reproducing an ARIMA model manually
- model ARIMA (2,0,0) (same as ARMA(2,0) or AR(2))

Autoregressive model: explains the future by regressing on the past

- goes back in time
- checks its own results
- does a forecast

```
myarima = arima(lynx, order = c(2,0,0)); myarima
```

```
##
## Call:
## arima(x = lynx, order = c(2, 0, 0))
##
## Coefficients:
##          ar1      ar2  intercept
##       1.1474  -0.5997  1545.4458
## s.e.  0.0742   0.0740   181.6736
##
## sigma^2 estimated as 768159:  log likelihood = -935.02,  aic = 1878.03
```

formula: *Yt = c + θ Yt-1 + θ Yt-2 + et*

OR

Present year's catches:

Constant (Calculated by R) + Coefficient1 x Last year's catches (Calculated by R) + Coefficient2 x Prior year's catches (Calculated by R) + Current error term

t = time in years Y = Amount of lynx trapped per year

```
# all the y values are available in the lynx dataset, we want to explain the last observed values
tail(lynx) #last value is 3396 lynx caught in 1934; year before that yt-1
```

```
## Time Series:
## Start = 1929
## End = 1934
## Frequency = 1
## [1]  485  662 1000 1590 2657 3396
```

```
# looking back at myarima model, I can see coefficients for both years
# Yt = c + $\theta$ Yt-1 + $\theta$ Yt-2 + et
# 3396 = c + 1.147 * 2657 - 0.5997 * 1590 + et
# Intercept is not equal to constant but to mean; mean of the model is 1545.4458
# We have to work with mean differently than we would work with standard constant
  # Yt - $\mu$ = $\theta$ 1(Yt-1 - $\mu$) + $\theta$ 2(Yt-2 - $\mu$) + et
# 3396 - 1545.45 = 1.147 * (2657 - 1545.45) - 0.599*(1590-1545.45) + et
```

```
residuals(myarima) # we are looking at 601.84
```

```
## Time Series:
## Start = 1821
## End = 1934
## Frequency = 1
##   [1]  -711.715800  -247.179068  -321.014839  -306.751202   127.414827
##   [6]   951.890591   876.687792  2428.733153  -212.432514  -237.541926
```

```
## [11]  -164.223204   344.415030  -313.801319  -572.372533  -499.800869
## [16]  1284.008241  -390.614888   999.532714 -1176.312892  -338.411239
## [21]    76.614594  -581.986383  -592.092428  -537.056449  -356.640535
## [26]  -164.773680   572.140125    13.626146 -1375.059569    84.838236
## [31]  -162.287575  -690.094698  -371.088497  -246.153634   316.113199
## [36]   584.894187    27.600121  -240.002495  -724.567794    85.994521
## [41]  -395.876984  -545.490420  -286.601293   437.533551  1080.751334
## [46]  3196.206424 -2171.180547  -862.324669  1319.008240  -106.590562
## [51]  -730.821550   -42.121950   210.099599  -381.832131   584.871735
## [56]  -850.724879  -229.236651  -412.244306  -387.695328  -521.330158
## [61]  -372.233398  -363.825181   779.748247   210.328753  1731.217687
## [66] -1586.424626  -533.760190   433.588275  -510.481277  -650.987938
## [71]  -672.853636  -549.330544  -502.352341   273.149380  2075.597251
## [76] -1054.463188 -1704.735336   828.545228  -314.449678  -424.603800
## [81]  -293.316062   -29.674453  1720.888636  3099.981788  -329.627515
## [86]    44.035737   569.799862  -185.278565   388.247443  -122.427802
## [91]    -9.044981   905.933577   820.433050  -341.167517  1018.287679
## [96]  1519.696544 -2583.562459   881.643131  -307.733379  -634.234854
## [101]  -545.962808  -498.009703   112.495341   673.381411   763.327803
## [106]  -406.375377  -386.254717  -173.376326   100.795394  -276.260594
## [111]  -167.745563   140.575959   733.302579   601.838001
```

```
# 3396 - 1545.45 = 1.147 * (2657 - 1545.45) - 0.599*(1590-1545.45) + 601.84
# 1850.55 = 1850.55 => both sides are same, it shows that the equation is correct
```

```
#looking at moving average model

myarima = arima(lynx, order = c(0,0,2)); myarima
```

```
##
## Call:
## arima(x = lynx, order = c(0, 0, 2))
##
## Coefficients:
##          ma1     ma2  intercept
##       1.1407  0.4697  1545.3670
## s.e.  0.0776  0.0721   224.5215
##
## sigma^2 estimated as 855092:  log likelihood = -941.03,  aic = 1890.06
```

```
residuals(myarima)
```

```
## Time Series:
## Start = 1821
## End = 1934
## Frequency = 1
##   [1]  -803.732851  -316.819775  -339.796973  -153.575542   256.164758
##   [6]  1051.017490  1062.665677  2690.592373  -162.936784   -44.605977
##  [11]  -894.921151  -405.552321  -478.418368  -530.135762  -306.914693
##  [16]  1338.739662  -243.365541  1512.454318 -1332.377780  -326.856600
##  [21]  -395.701695  -895.452231  -270.030612  -603.745610  -183.818830
##  [26]   -19.103090   691.762826   210.483679 -1153.389638    32.488716
##  [31]  -663.690549  -578.528068  -213.686644  -298.876138   533.939929
##  [36]   710.925757   263.863961   -61.283359  -915.393384  -173.356483
##  [41]  -681.659497  -441.346353  -169.735507   478.553892  1299.450551
##  [46]  3468.524287 -1858.394332  -367.558957     1.794961  -901.775190
```

```
##  [51]  -159.518680  -155.841195   301.331932  -139.911234   723.702215
##  [56]  -879.208277  -126.335608  -689.293961  -498.722732  -423.698105
##  [61]  -358.791482  -201.071547   894.524832   339.653953  2078.024947
##  [66] -1564.386859  -347.838999  -340.793301  -954.233203  -247.766795
##  [71]  -755.533899  -379.124919  -381.015812   359.344984  2254.673608
##  [76]  -791.145850 -1114.876734   203.012693 -1100.303344     1.440094
##  [81]  -272.206328    71.473327  1965.953529  3169.419791   228.755266
##  [86]   499.031993  -386.077507  -994.344061   152.258905  -444.019657
##  [91]   277.629380  1059.482295   915.638387     3.497027  1005.575888
##  [96]  1095.889333 -2593.803120   979.758850 -1364.729473  -340.749646
## [101]  -286.657856  -659.317506   473.383962   656.300763  1057.619544
## [106]  -125.095552  -362.420744  -544.182680  -269.369783  -320.487877
## [111]   -53.252771   255.911083   844.717221   766.830502
```

```
#Yt = c + $\theta$ et-1 + $\theta$ et-2 + et
#3396 - 1545.36 = 1.1407*844.7 + 0.469 * 255.9 + 766.83


#Check the equation for MA(2)
#1850.933 = 1850.63
# tiny differences in decimals don't matter
# calculation.png shows difference in formula between two models
```

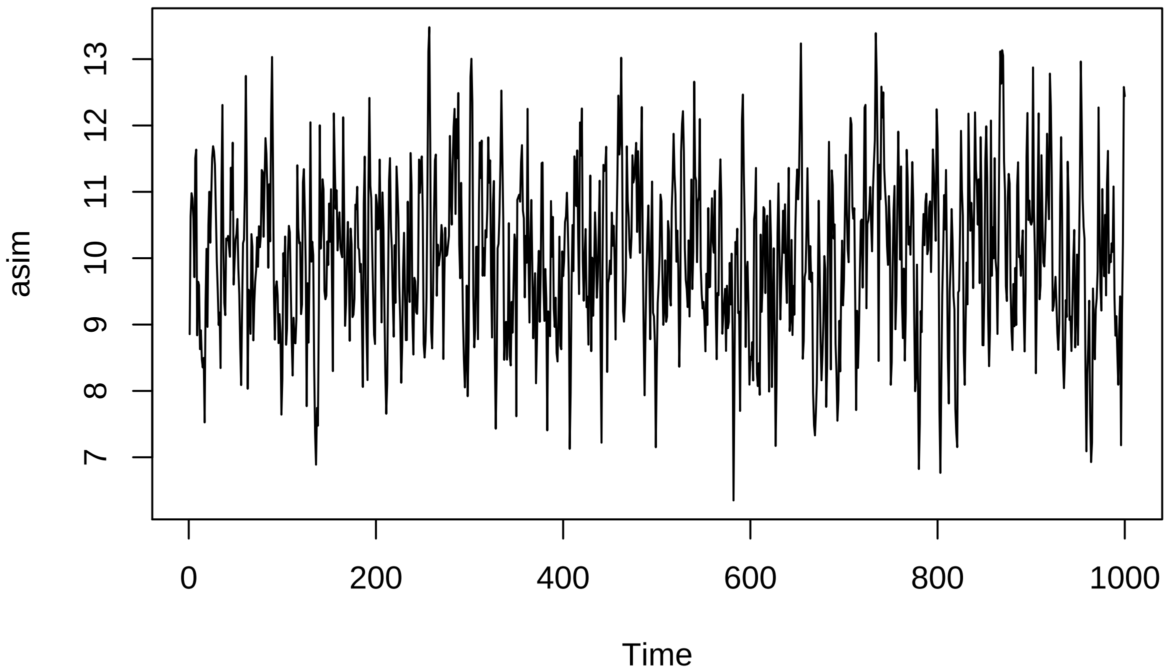
18.4 ARIMA Based Simulations

R base: arima.sim() => Generates time series based on a provided ARIMA model

A key step before you begin with any sort of simulations; => Make sure that the results are reproducible

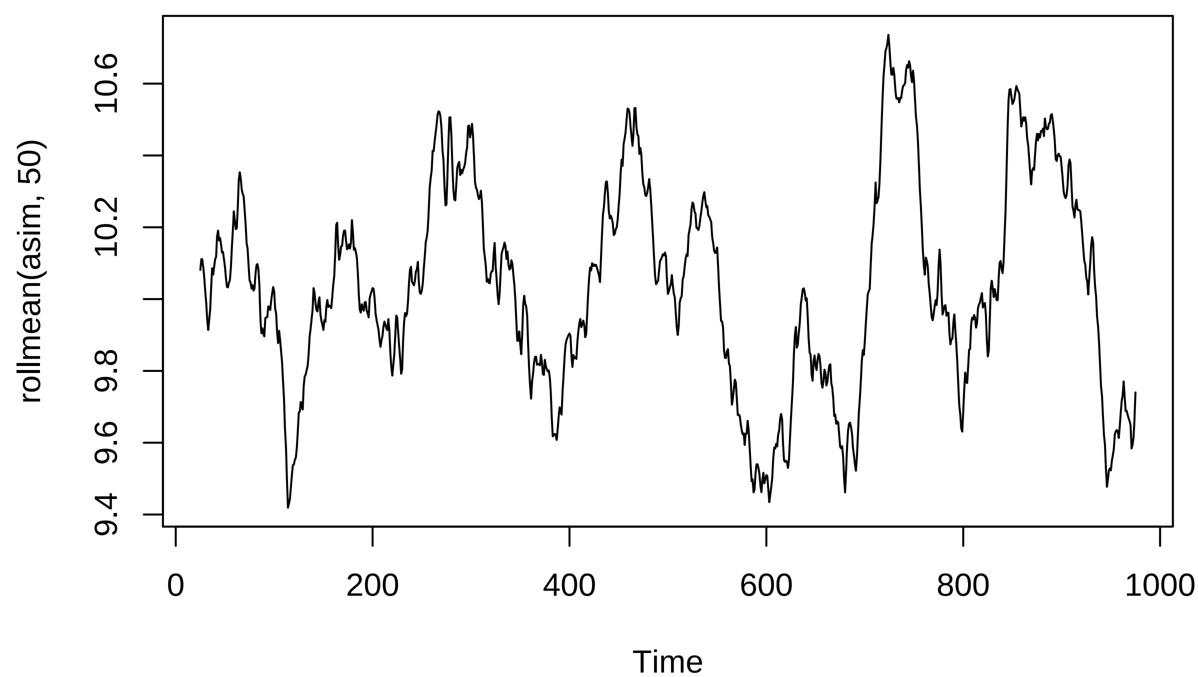
```
set.seed(123) #for reproduction
```

```
asim <- arima.sim(model =list(order=c(1,0,1),
                              ar = c(0.4),
                              ma = c(0.3)), n = 1000) + 10

plot (asim)
```

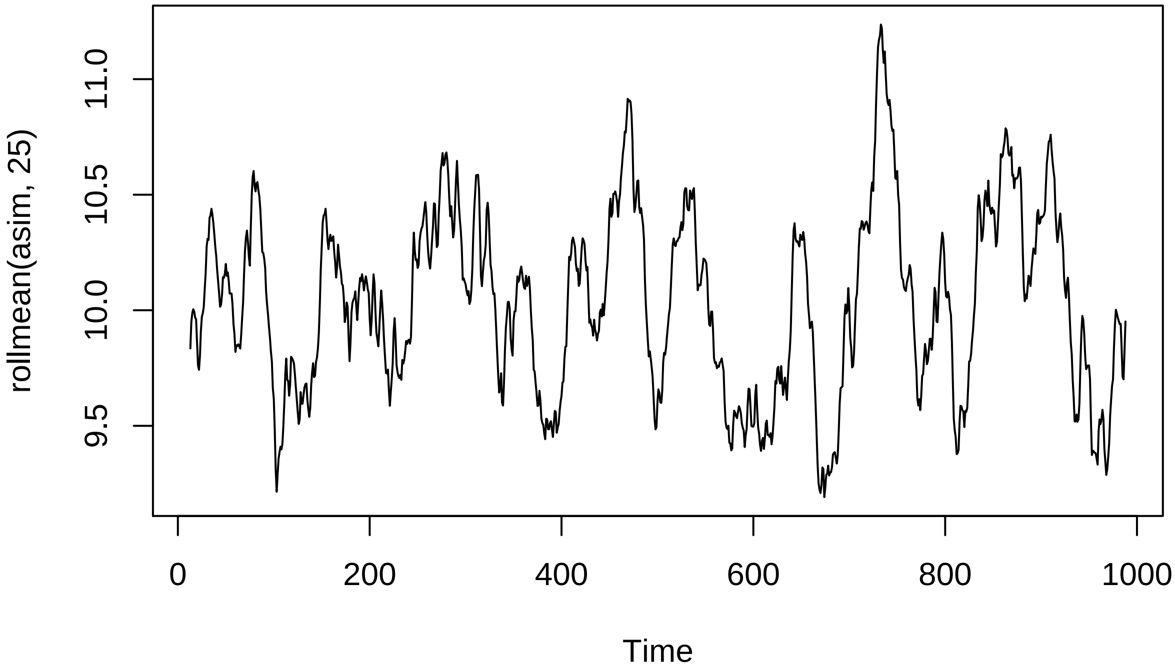



```r
# number of observation of the produced time series
# model needs to be a list with the model components:
  # Order of the model
  # AR parameter coefficient
  # MA parameter coefficient
# Specifying a mean => it can be any other number value
```

```r
#library(zoo)
plot(rollmean(asim, 50)) # 50 days moving average
```



```r
plot(rollmean(asim, 25))
```

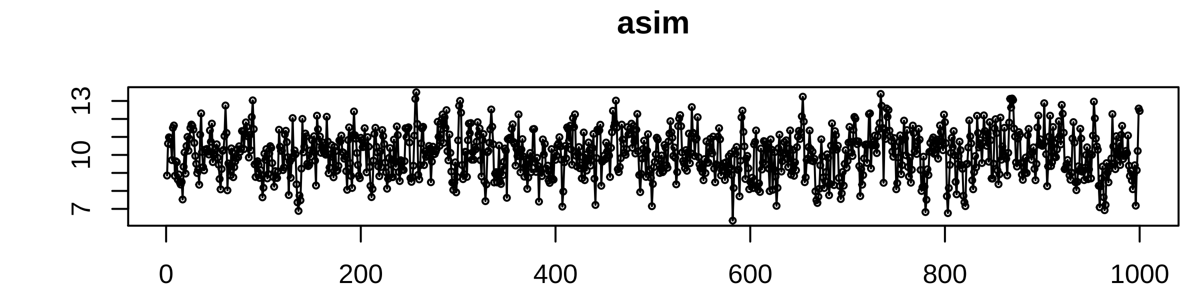


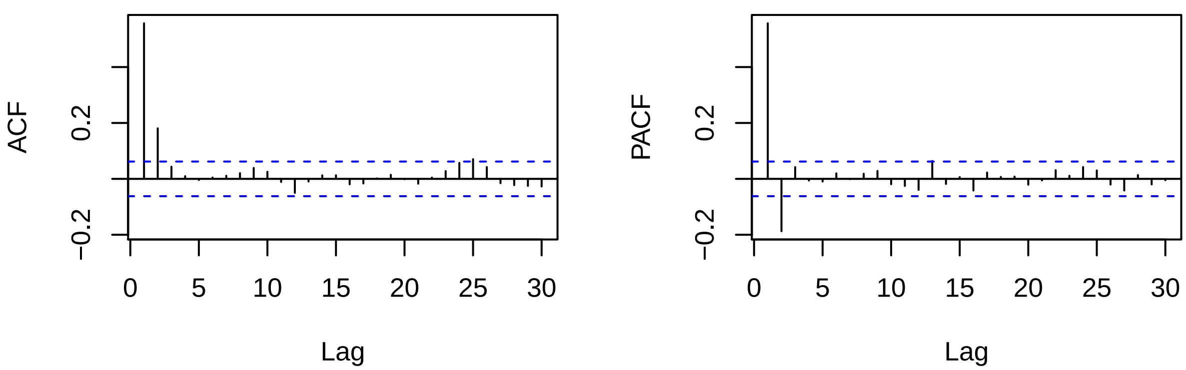
18.4.1 Stationarity and Autocorrelation

```r
library(tseries)
library(forecast)

adf.test(asim)
```

```
##
##  Augmented Dickey-Fuller Test
##
## data:  asim
## Dickey-Fuller = -9.0113, Lag order = 9, p-value = 0.01
## alternative hypothesis: stationary
```

```r
# test is significant, stationarity
tsdisplay(asim)
```






```r
# significance at first two lags
```

```r
# using auto.arima to see if it confirms parameters
auto.arima(asim, trace = T,
           stepwise = F,
           approximation = F)
```

```
##
##  ARIMA(0,0,0) with zero mean     : 7465.459
##  ARIMA(0,0,0) with non-zero mean : 3241.528
##  ARIMA(0,0,1) with zero mean     : 6218.948
##  ARIMA(0,0,1) with non-zero mean : 2878.74
##  ARIMA(0,0,2) with zero mean     : 5341.968
##  ARIMA(0,0,2) with non-zero mean : 2836.895
##  ARIMA(0,0,3) with zero mean     : 4809.724
##  ARIMA(0,0,3) with non-zero mean : 2837.534
##  ARIMA(0,0,4) with zero mean     : 4450.32
##  ARIMA(0,0,4) with non-zero mean : 2838.689
##  ARIMA(0,0,5) with zero mean     : 4219.275
##  ARIMA(0,0,5) with non-zero mean : 2840.557
##  ARIMA(1,0,0) with zero mean     : Inf
##  ARIMA(1,0,0) with non-zero mean : 2870.637
##  ARIMA(1,0,1) with zero mean     : Inf
##  ARIMA(1,0,1) with non-zero mean : 2836.047
##  ARIMA(1,0,2) with zero mean     : Inf
##  ARIMA(1,0,2) with non-zero mean : 2837.165
##  ARIMA(1,0,3) with zero mean     : Inf
##  ARIMA(1,0,3) with non-zero mean : 2839.088
##  ARIMA(1,0,4) with zero mean     : Inf
```

```
##  ARIMA(1,0,4) with non-zero mean : 2840.615
##  ARIMA(2,0,0) with zero mean     : Inf
##  ARIMA(2,0,0) with non-zero mean : 2836.945
##  ARIMA(2,0,1) with zero mean     : Inf
##  ARIMA(2,0,1) with non-zero mean : 2837.319
##  ARIMA(2,0,2) with zero mean     : Inf
##  ARIMA(2,0,2) with non-zero mean : 2838.849
##  ARIMA(2,0,3) with zero mean     : Inf
##  ARIMA(2,0,3) with non-zero mean : 2840.867
##  ARIMA(3,0,0) with zero mean     : Inf
##  ARIMA(3,0,0) with non-zero mean : 2837.297
##  ARIMA(3,0,1) with zero mean     : Inf
##  ARIMA(3,0,1) with non-zero mean : 2839.296
##  ARIMA(3,0,2) with zero mean     : Inf
##  ARIMA(3,0,2) with non-zero mean : 2840.86
##  ARIMA(4,0,0) with zero mean     : Inf
##  ARIMA(4,0,0) with non-zero mean : 2839.279
##  ARIMA(4,0,1) with zero mean     : Inf
##  ARIMA(4,0,1) with non-zero mean : 2841.309
##  ARIMA(5,0,0) with zero mean     : Inf
##  ARIMA(5,0,0) with non-zero mean : 2841.162
```

```
## Series: asim
## ARIMA(1,0,1) with non-zero mean
##
## Coefficients:
##          ar1     ma1     mean
##       0.3494  0.3183  10.0288
## s.e.  0.0478  0.0473   0.0637
##
## sigma^2 estimated as 0.9927:  log likelihood=-1414
## AIC=2836.01   AICc=2836.05   BIC=2855.64
```

```
# confirmed model is ARIMA (1,0,1), ar is 0.34 and ma is 0.31, mean 10, overall we get the result which
```

18.5 Manual ARIMA Parameter Selection

ARIMA =>

- Manual Parameter Selection
- arima() R Base
- Arima() forecast
- Automated Parameter Selection
- auto.arima() forecast

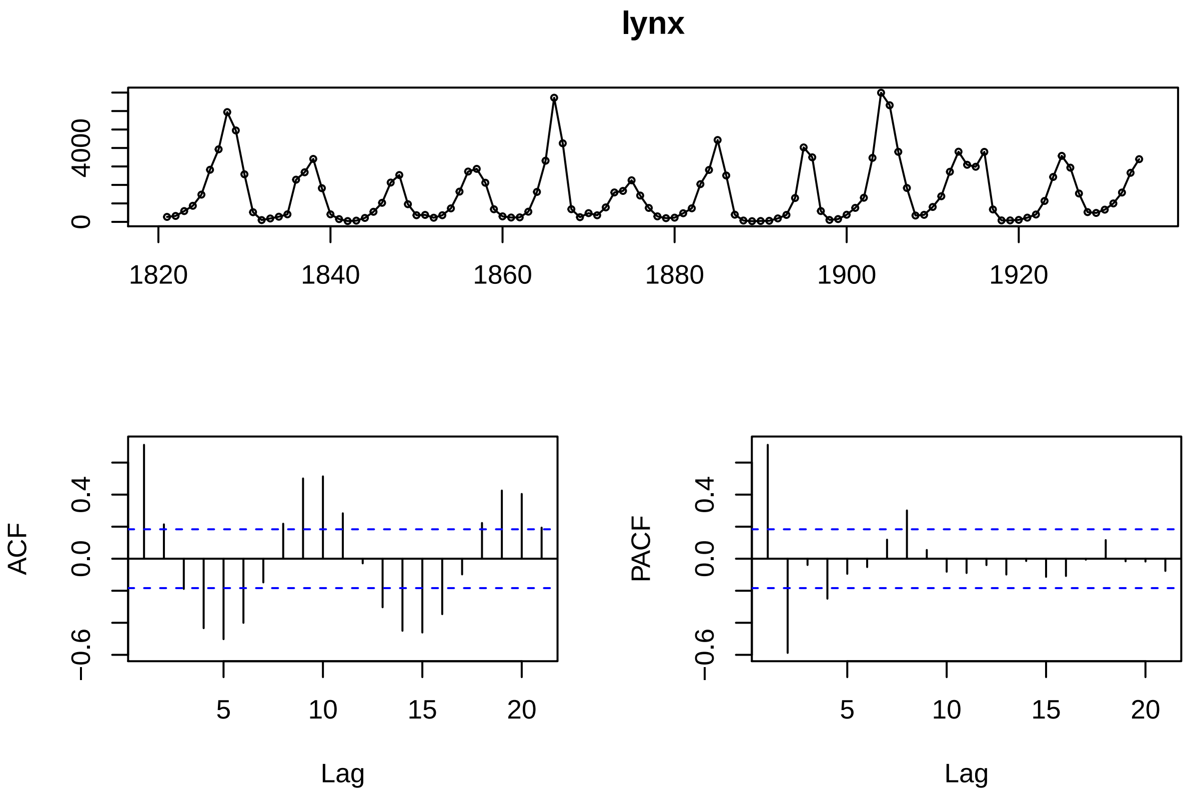
If there is a middle parameter D in your model, the output of the function will not provide a constant or a non zero mean for that matter. Therefore, it is better to use Arima() for forecast.

```
#Testing for stationarity - adf.test() from library tseries
#esting for autoregression - acf() and pacf() plots from R Base, OR tsdisplay() from library forecast

#library(tseries)

adf.test(lynx)
```

```
## Warning in adf.test(lynx): p-value smaller than printed p-value
```

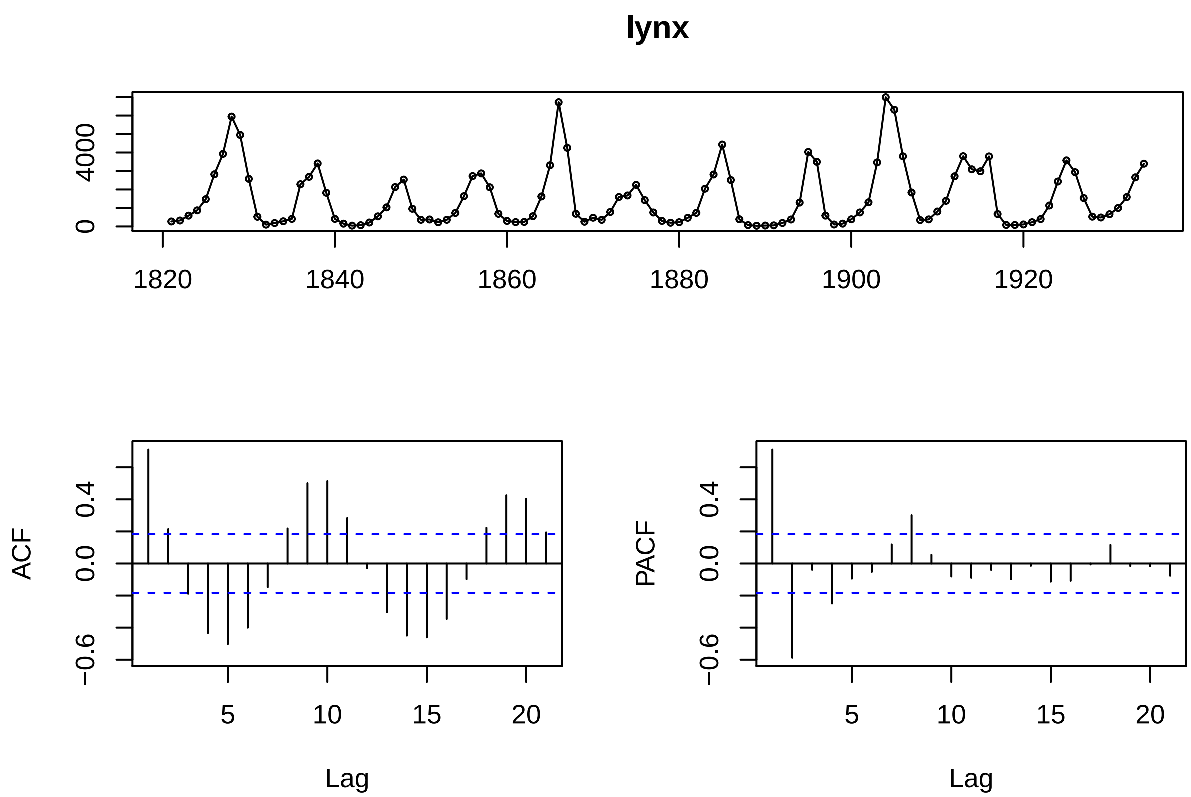
```
##
##  Augmented Dickey-Fuller Test
##
## data:  lynx
## Dickey-Fuller = -6.3068, Lag order = 4, p-value = 0.01
## alternative hypothesis: stationary
```

```
#stationarity and it means that we can set the middle parameter D to 0. ARIMA (p,0,q)

#library(forecast)

tsdisplay(lynx)
```

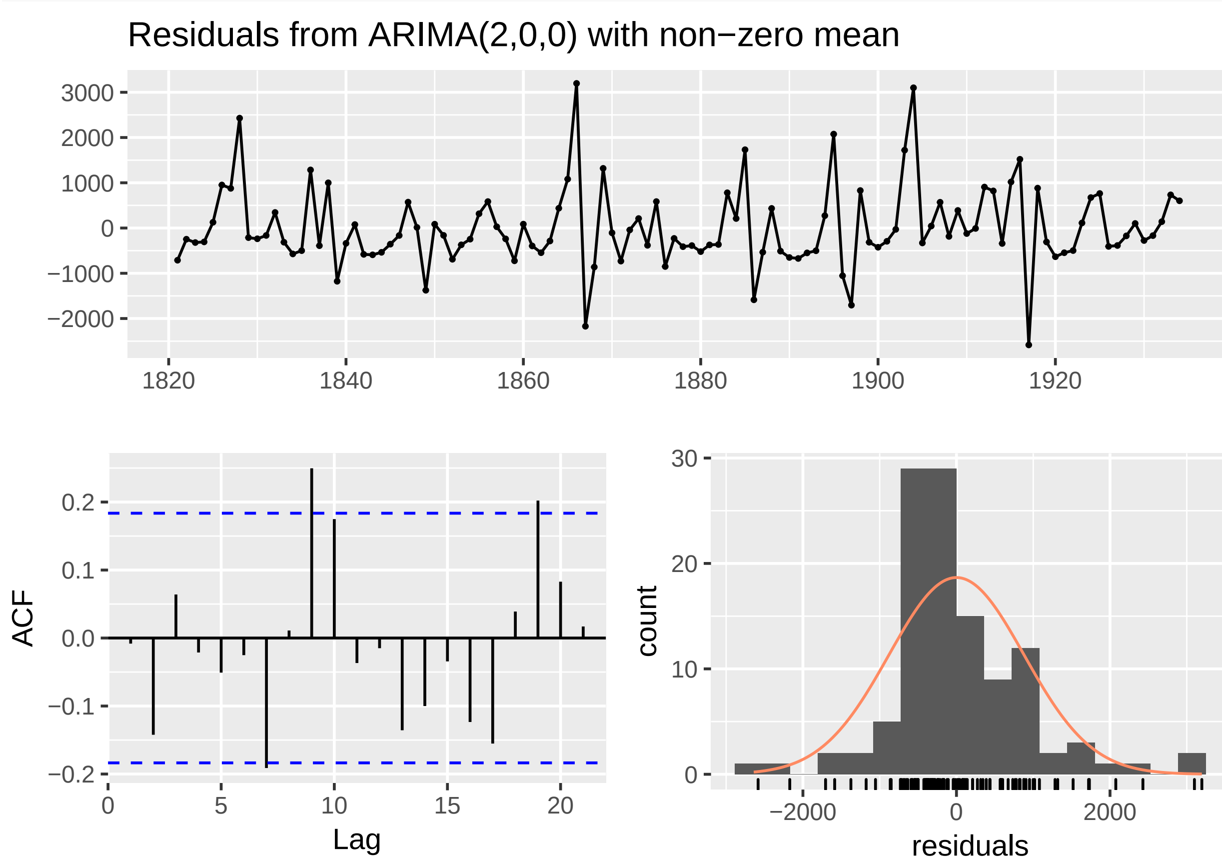



```
# Plot ACF tells me about Lags for MA - Parameter 'q' and plot PACF tells me about lags for AR - Parame
```

```
myarima <- Arima(lynx, order = c(2,0,0))
# checking results - aicc num 1878

tsdisplay(lynx)
```



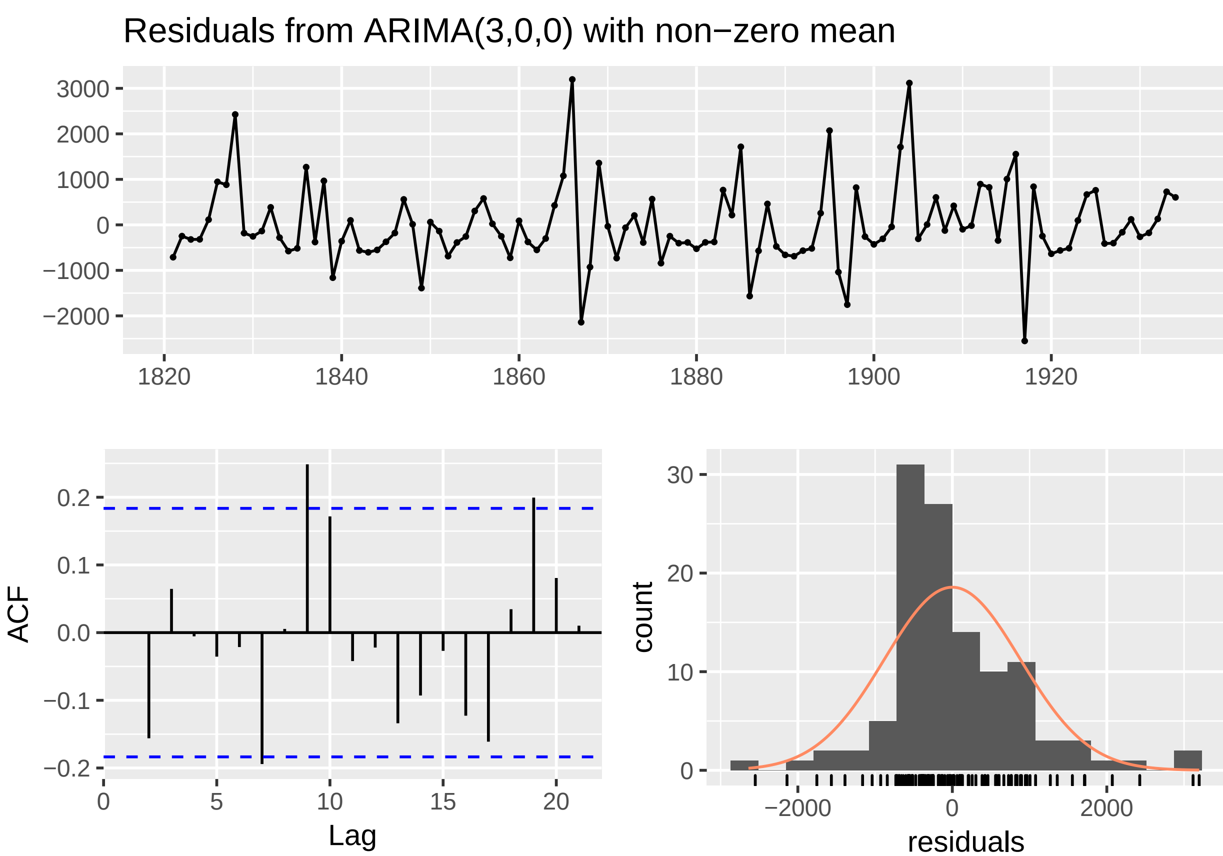

```r
checkresiduals(myarima)
```



```
##
##  Ljung-Box test
##
## data:  Residuals from ARIMA(2,0,0) with non-zero mean
## Q* = 19.603, df = 7, p-value = 0.006494
##
## Model df: 3.   Total lags used: 10
```

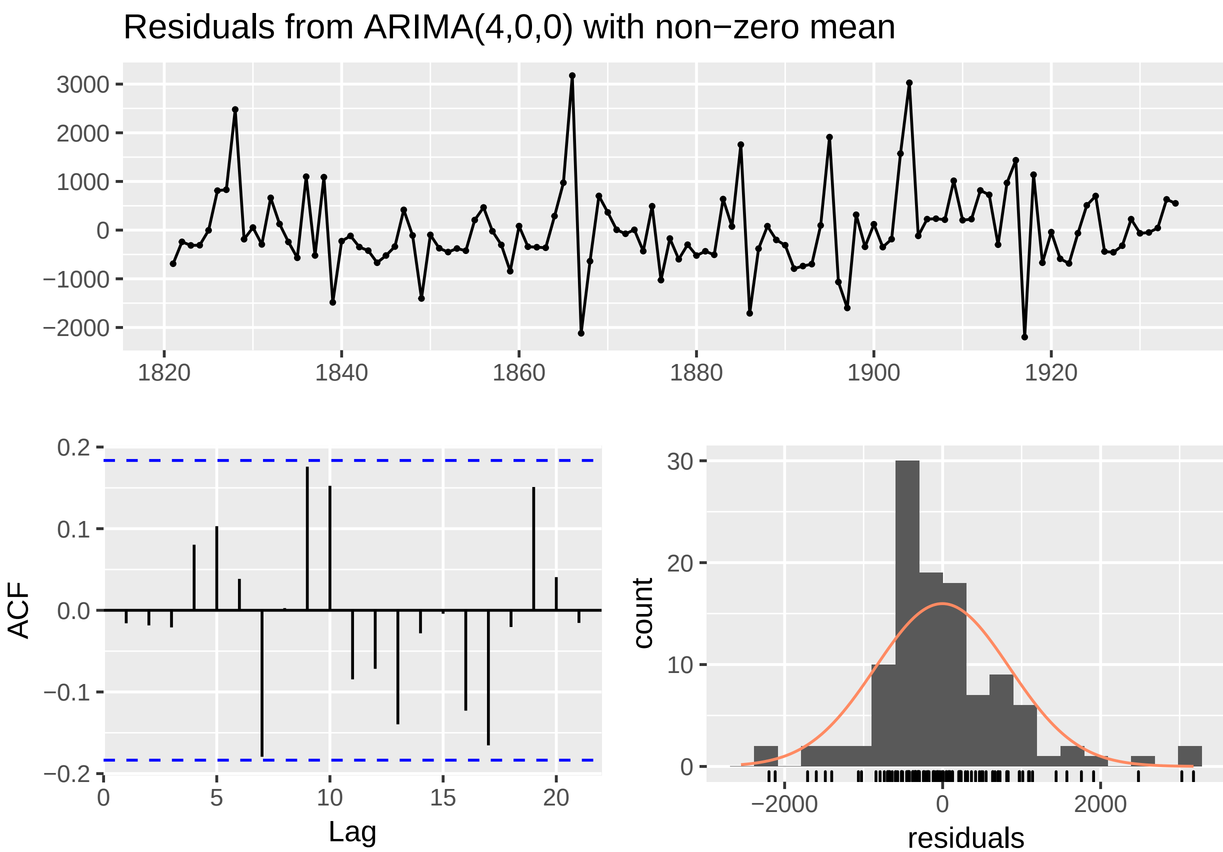
```r
# residuals should be random and normally distributed; ACF shows significance at lags 7,9,19. This show
```

```r
myarima <- Arima(lynx, order = c(3,0,0))
# checking results - aicc num 1875

checkresiduals(myarima)
```




```
##
##  Ljung-Box test
##
## data:  Residuals from ARIMA(3,0,0) with non-zero mean
## Q* = 19.786, df = 6, p-value = 0.003023
##
## Model df: 4.   Total lags used: 10
```

```r
# still no improvement

myarima <- Arima(lynx, order = c(4,0,0))
# checking results - aicc num 1881

checkresiduals(myarima)
```




```
##
##  Ljung-Box test
##
## data:  Residuals from ARIMA(4,0,0) with non-zero mean
## Q* = 13.201, df = 5, p-value = 0.02157
##
## Model df: 5.   Total lags used: 10
```

```
# still no improvement
```

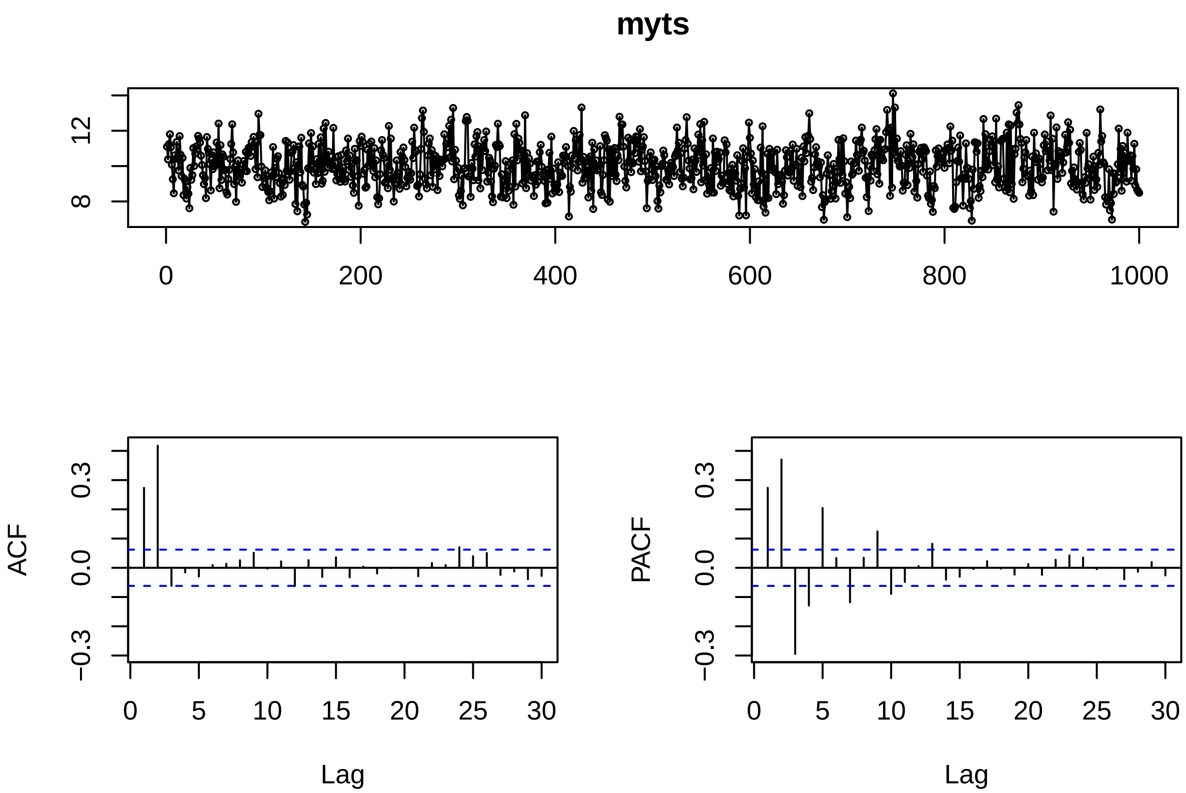
18.6 Example MA time series

```
set.seed(123) # for reproduction

#Simulation
myts <- arima.sim(model =list(order=c(0,0,2),
                              ma = c(0.3, 0.7)), n = 1000) + 10

adf.test(myts) #stationarity, significant
```

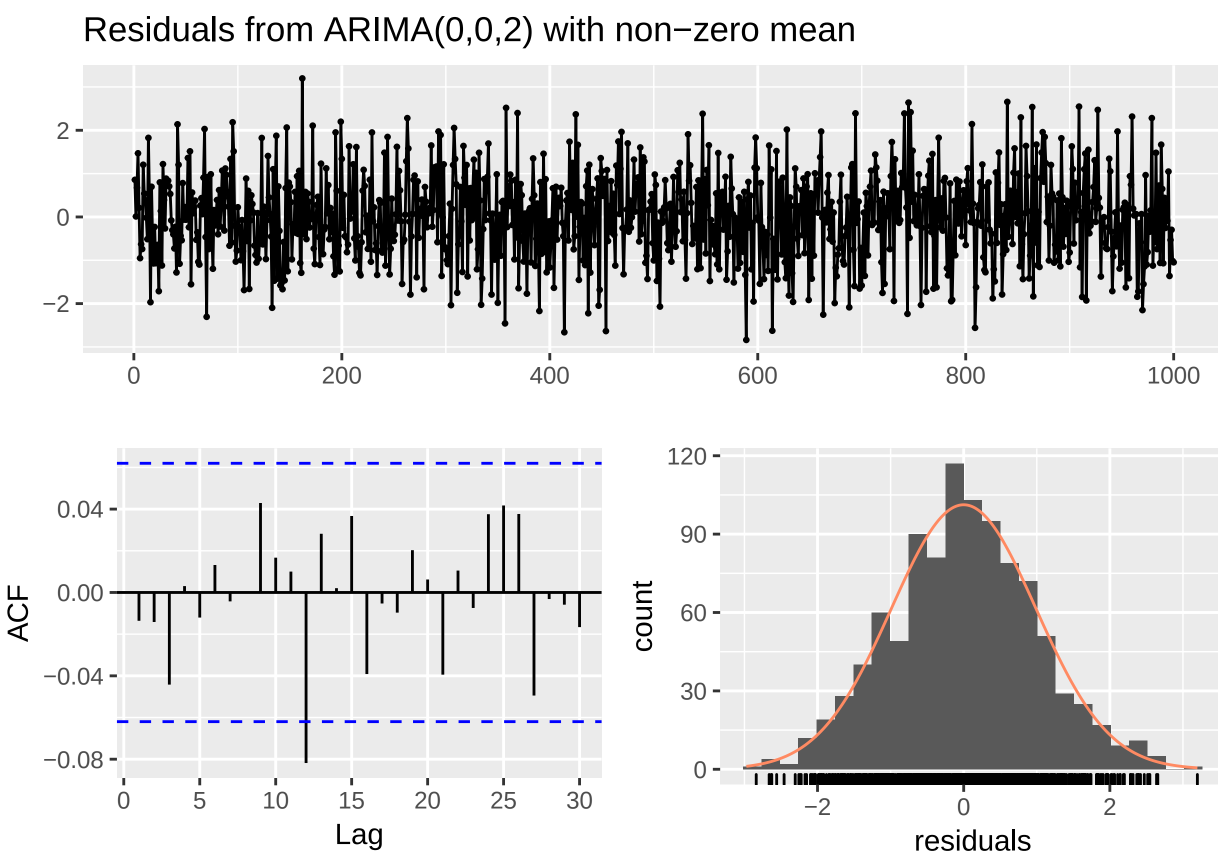
```
##
##  Augmented Dickey-Fuller Test
##
## data:  myts
## Dickey-Fuller = -9.0469, Lag order = 9, p-value = 0.01
```

```
## alternative hypothesis: stationary
```

```r
tsdisplay(myts) # Autocorrelation
```




```r
# Arima
myarima <- Arima(myts, order = c(0,0,2)) #ACF plot looked better and we are focusing on first two lags
# aicc is 2828
checkresiduals(myarima)
```




```
##
##  Ljung-Box test
##
## data:  Residuals from ARIMA(0,0,2) with non-zero mean
## Q* = 4.8457, df = 7, p-value = 0.6788
##
## Model df: 3.   Total lags used: 10
```

```
# normally distributed residuals, autocorrelation is ok, only 1 lag out of 20 is out

# Can this be improved?
auto.arima(myts, trace = T,
           stepwise = F,
           approximation = F)
```

```
##
##  ARIMA(0,0,0) with zero mean     : 7465.902
##  ARIMA(0,0,0) with non-zero mean : 3239.597
##  ARIMA(0,0,1) with zero mean     : 6414.662
##  ARIMA(0,0,1) with non-zero mean : 3199.385
##  ARIMA(0,0,2) with zero mean     : 5571.943
##  ARIMA(0,0,2) with non-zero mean : 2828.282
##  ARIMA(0,0,3) with zero mean     : 4982.239
##  ARIMA(0,0,3) with non-zero mean : 2829.867
##  ARIMA(0,0,4) with zero mean     : 4556.587
##  ARIMA(0,0,4) with non-zero mean : 2831.522
##  ARIMA(0,0,5) with zero mean     : 4300.593
##  ARIMA(0,0,5) with non-zero mean : 2831.318
```

```
##  ARIMA(1,0,0) with zero mean     : 3610.918
##  ARIMA(1,0,0) with non-zero mean : 3163.665
##  ARIMA(1,0,1) with zero mean     : Inf
##  ARIMA(1,0,1) with non-zero mean : 3120.607
##  ARIMA(1,0,2) with zero mean     : Inf
##  ARIMA(1,0,2) with non-zero mean : 2829.89
##  ARIMA(1,0,3) with zero mean     : Inf
##  ARIMA(1,0,3) with non-zero mean : 2831.04
##  ARIMA(1,0,4) with zero mean     : Inf
##  ARIMA(1,0,4) with non-zero mean : 2832.859
##  ARIMA(2,0,0) with zero mean     : Inf
##  ARIMA(2,0,0) with non-zero mean : 3017.436
##  ARIMA(2,0,1) with zero mean     : Inf
##  ARIMA(2,0,1) with non-zero mean : 2977.38
##  ARIMA(2,0,2) with zero mean     : Inf
##  ARIMA(2,0,2) with non-zero mean : 2831.603
##  ARIMA(2,0,3) with zero mean     : Inf
##  ARIMA(2,0,3) with non-zero mean : 2832.823
##  ARIMA(3,0,0) with zero mean     : Inf
##  ARIMA(3,0,0) with non-zero mean : 2929.264
##  ARIMA(3,0,1) with zero mean     : Inf
##  ARIMA(3,0,1) with non-zero mean : 2924.325
##  ARIMA(3,0,2) with zero mean     : Inf
##  ARIMA(3,0,2) with non-zero mean : 2831.357
##  ARIMA(4,0,0) with zero mean     : Inf
##  ARIMA(4,0,0) with non-zero mean : 2914.331
##  ARIMA(4,0,1) with zero mean     : Inf
##  ARIMA(4,0,1) with non-zero mean : 2899.065
##  ARIMA(5,0,0) with zero mean     : Inf
##  ARIMA(5,0,0) with non-zero mean : 2873.303

## Series: myts
## ARIMA(0,0,2) with non-zero mean
##
## Coefficients:
##          ma1     ma2     mean
##       0.2878  0.6838  10.0297
## s.e.  0.0230  0.0231   0.0617
##
## sigma^2 estimated as 0.9842:  log likelihood=-1410.12
## AIC=2828.24   AICc=2828.28   BIC=2847.87
```

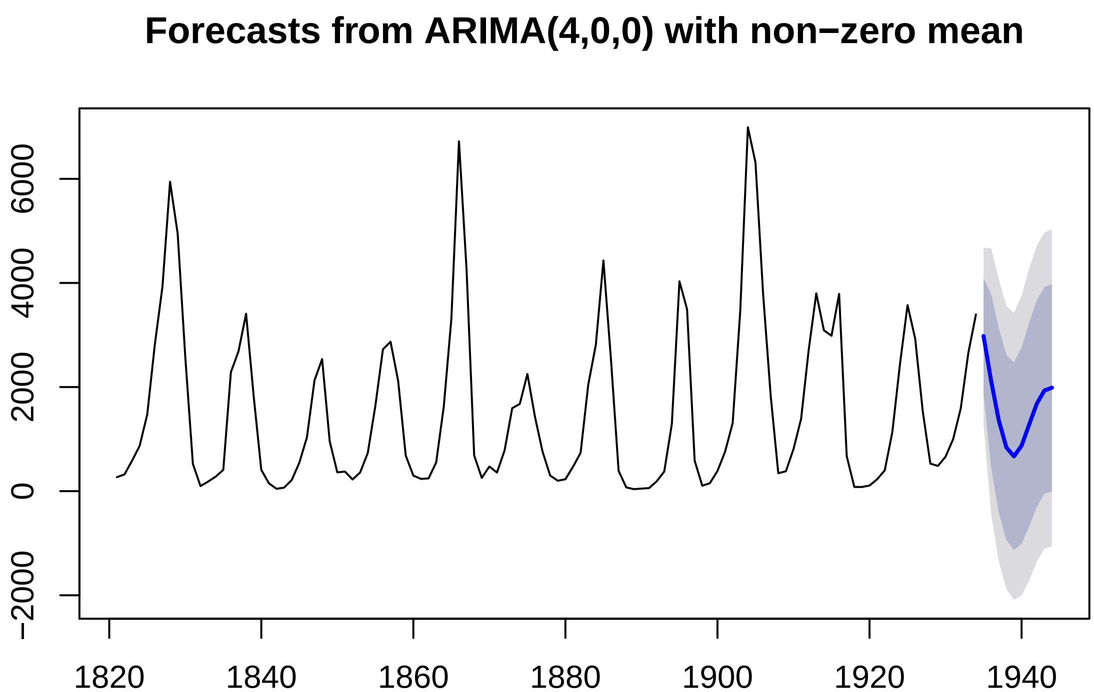
```
# aicc is 2830; not improved
```

19 Forecasting an ARIMA model

Once I decided on my model, I can move to forecasting

```
myarima <- auto.arima(lynx, stepwise = F, approximation = F)
```

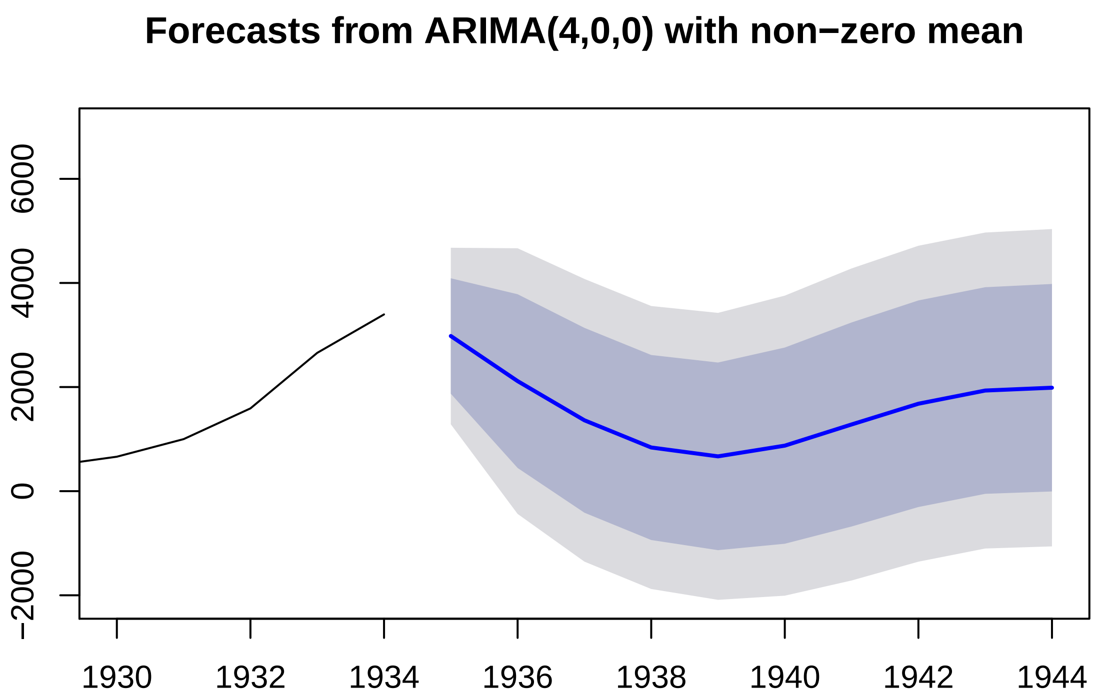
```
arimafore <- forecast(myarima, h=10) #forecast of 10 years

plot(arimafore) # with 90 and 95% intervals
```




```r
# see the forecaste values
arimafore$mean
```

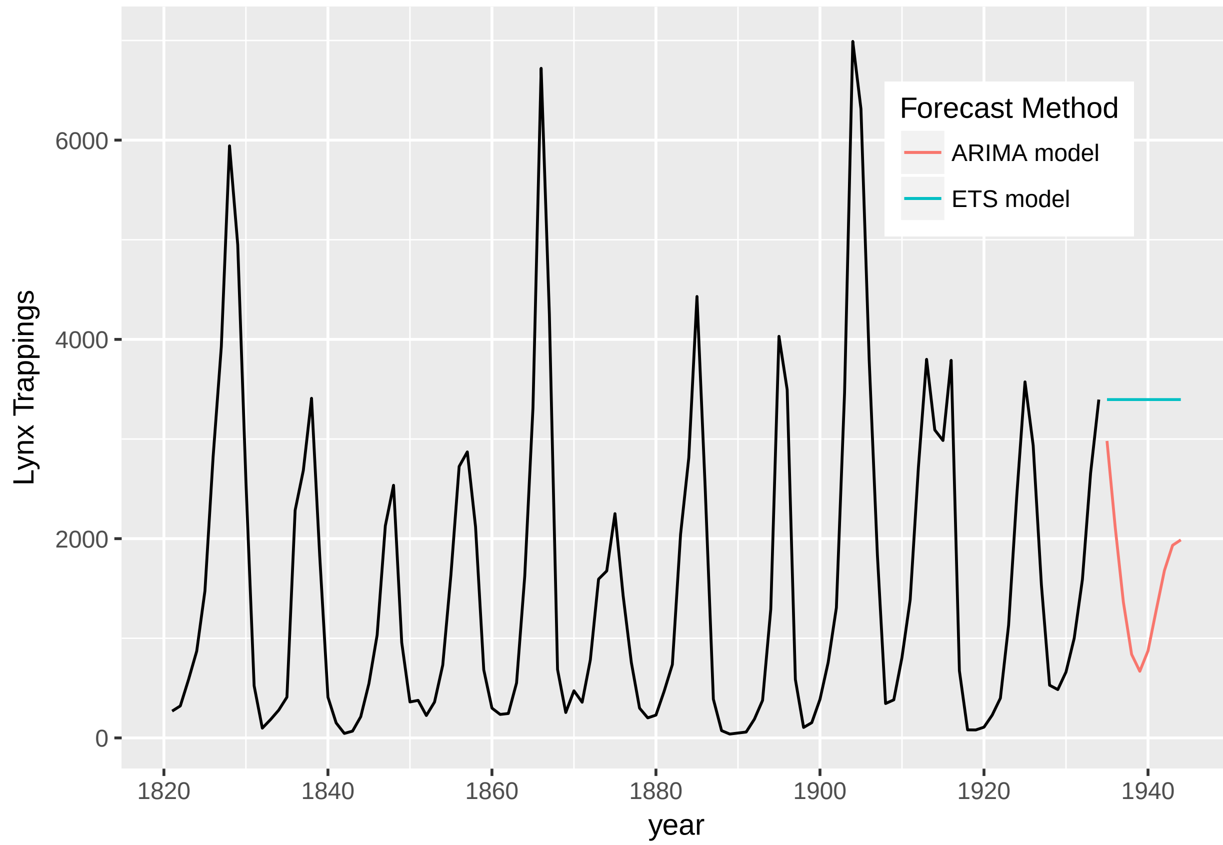
```
## Time Series:
## Start = 1935
## End = 1944
## Frequency = 1
##  [1] 2980.7782 2114.6447 1361.7211  839.0137  668.7873  874.3079 1281.3753
##  [8] 1679.8363 1933.3503 1987.5494
```

```r
# zoom-in, plot last observations and the forecast
plot(arimafore, xlim=c(1930,1944))
```




```
#ets for comparison

myets <- ets(lynx)

etsfore <- forecast(myets, h=10)
```

```
library(ggplot2)
# plotting two models
autoplot(lynx)+
  forecast::autolayer(
    etsfore$mean,
    series = 'ETS model') +
  forecast::autolayer(
    arimafore$mean,
    series = 'ARIMA model')+
  xlab('year') +
  ylab ('Lynx Trappings') +
  guides(
    colour = guide_legend(
      title = 'Forecast Method')) +
  theme(
    legend.position = c(0.8, 0.8))
```



20 ARIMA with Explanatory Variables

What is an explanatory variable? It is a second variable basides the variable to be modeled. Also called independent variable or predictor. Predictor can be of various classes: numberic, integer, character, boolean.

Previous forecasting was based on simple autoregressive model withou taking other factors into account.

The forecaster needs to know about upcoming events (such as speech of central bank to predict prices on stock market) or make forecasts on the explanatory variable.

```
#library(readr)
cyprinidae <- read_csv("~/Desktop/cyprinidae.csv")
```

```
## Parsed with column specification:
## cols(
##   X1 = col_integer(),
##   concentration = col_double(),
##   predator_presence = col_logical()
## )
```

```
cols(
  X1 = col_integer(),
  concentration = col_double(),
  predator_presence = col_logical()
)
```

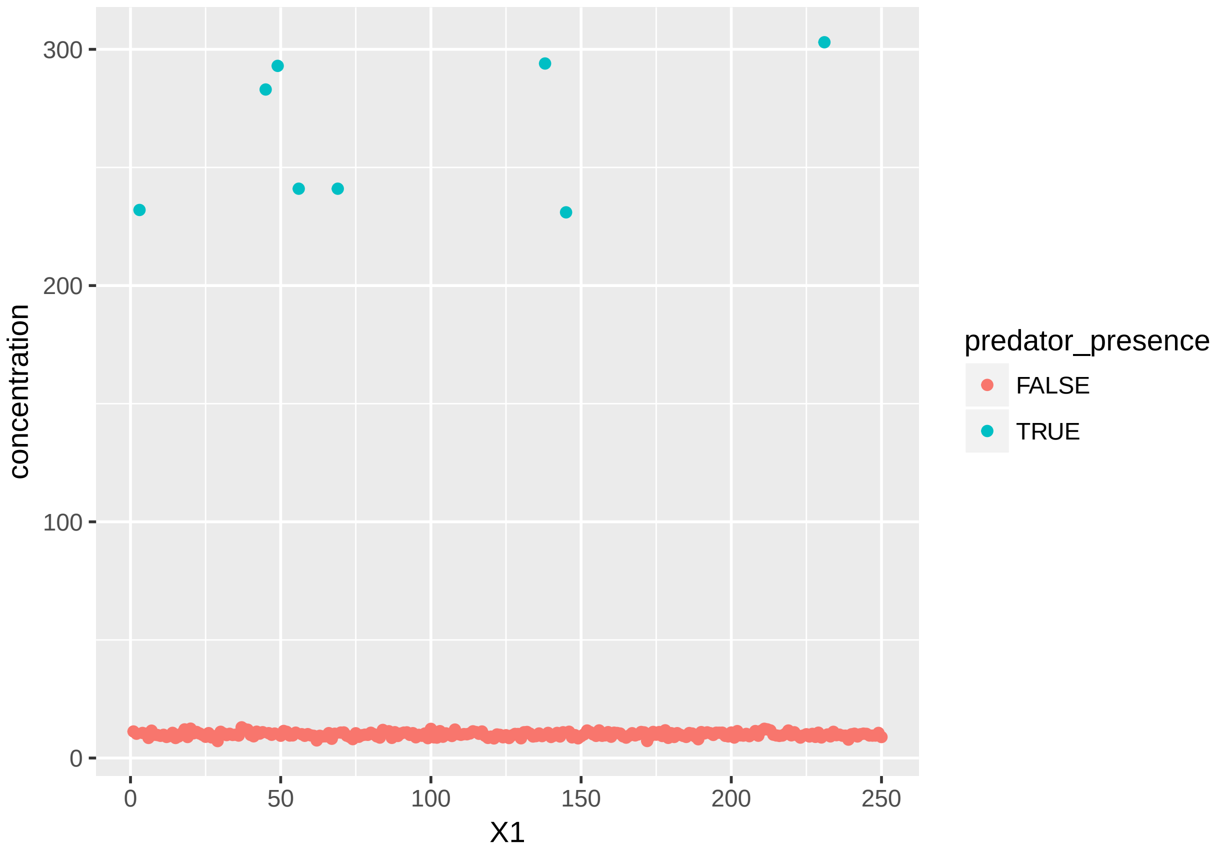
```
## cols(
```

```
##   X1 = col_integer(),
##   concentration = col_double(),
##   predator_presence = col_logical()
## )
```

Research: Do cyprinid fish answer with a hormonal reaction to the presence of a predatory catfish?

20.1 Plot the data

```
#library(ggplot2)
ggplot(cyprinidae, aes(y=concentration, x=X1))+ #X1 is time identifier
  geom_point(aes(color=predator_presence))
```




```
  # blue dots shows time where predictor the  catfish were present
```

Some functions of the 'forecast' package have the argument 'xreg' available which allows us to include an explanatory variable in the model e.g. auto.arima(), nnetar()

20.2 Convert the variables into time series

```
x= ts(cyprinidae$concentration)
y= cyprinidae$predator_presence
```

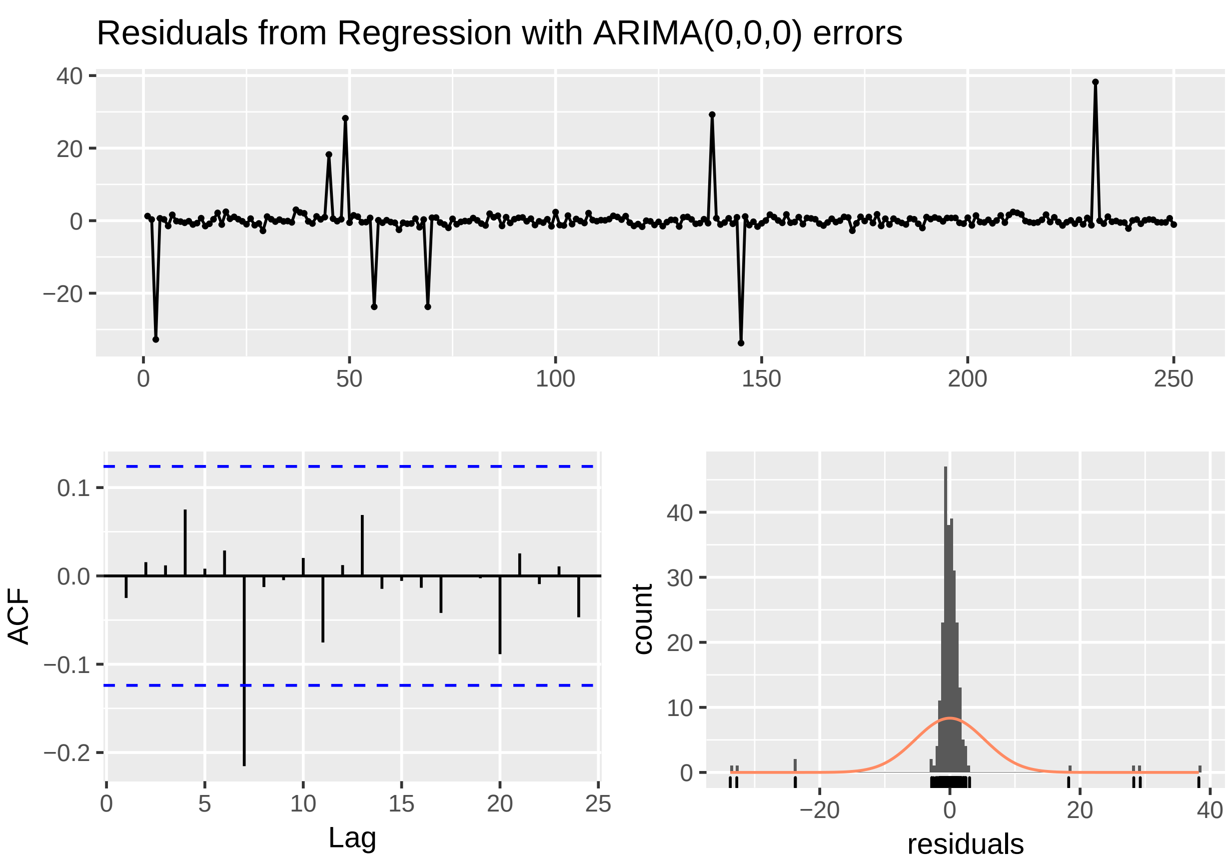
20.3 Arima model creation

```
library(forecast)
mymodel = auto.arima (x, xreg = y, stepwise = F, approximation = F)
mymodel
```

```
## Series: x
## Regression with ARIMA(0,0,0) errors
##
## Coefficients:
##       intercept      xreg
##          9.9765  254.7735
## s.e.     0.3409    1.9059
##
## sigma^2 estimated as 28.36:  log likelihood=-771.84
## AIC=1549.68   AICc=1549.77   BIC=1560.24
```

20.4 Quick check of model quality

```
checkresiduals(mymodel)
```



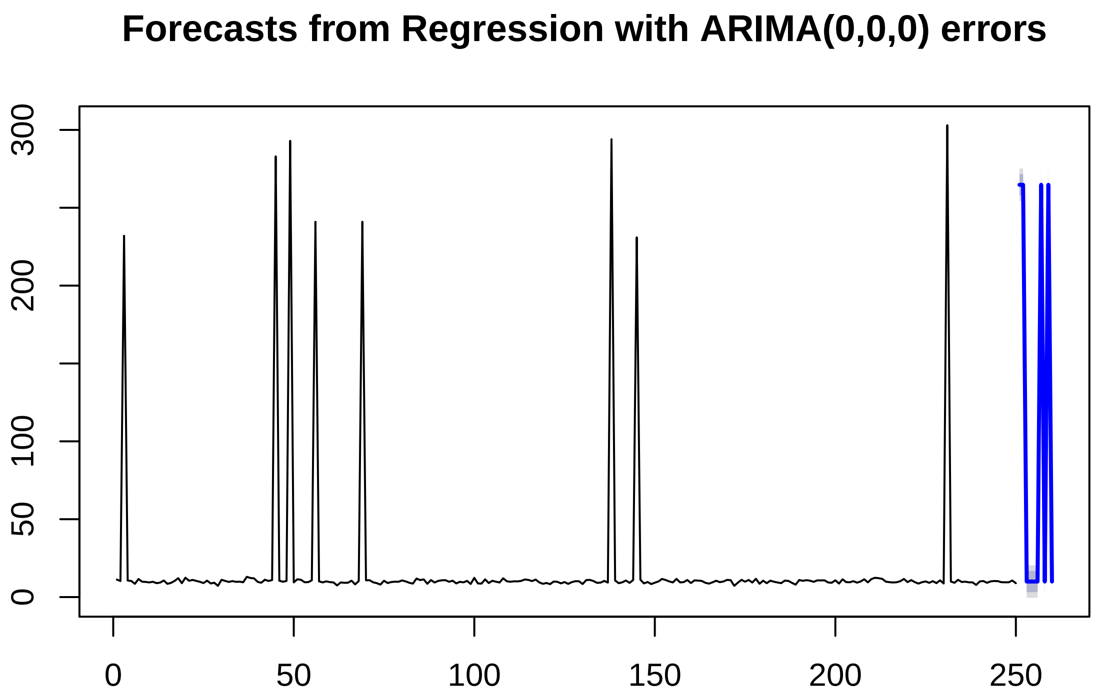

```
##
##  Ljung-Box test
##
## data:  Residuals from Regression with ARIMA(0,0,0) errors
## Q* = 14.122, df = 8, p-value = 0.07865
```

```
##
## Model df: 2.   Total lags used: 10
```

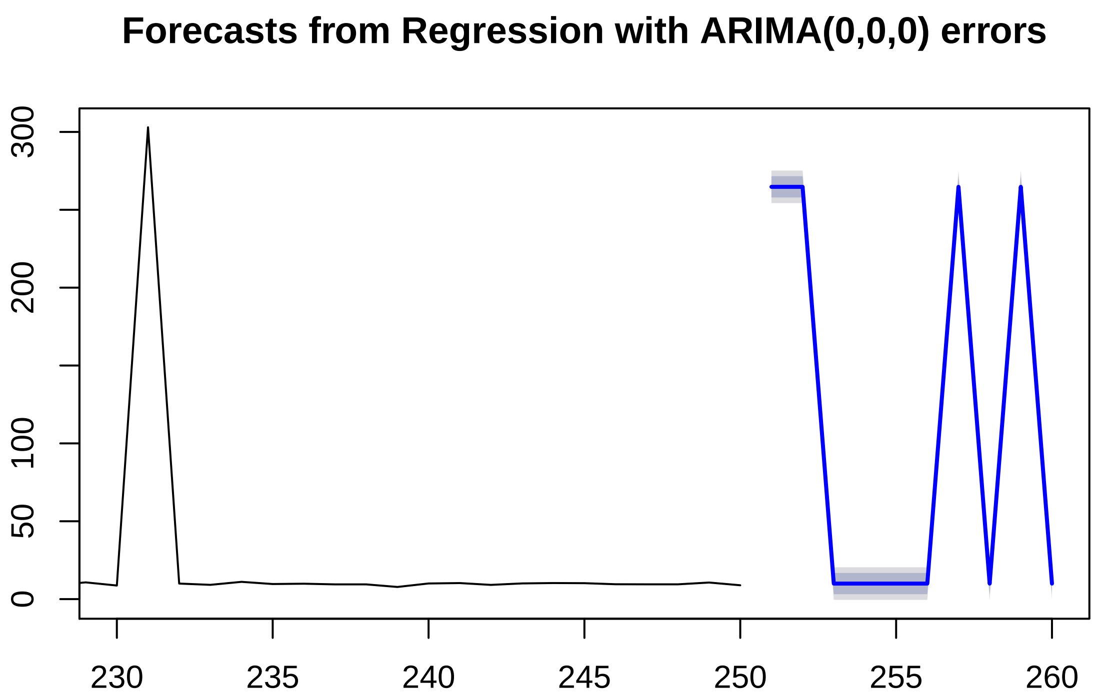
All the patterns shoud be captured by the model, only randomness should stay in the residuals.

20.5 Expected predator presence at future 10 times and getting a forecast based on future predator

```r
y1 = c(T,T,F,F,F,F,T,F,T,F)

plot(forecast(mymodel, xreg = y1))
```




```r
# to zoom in on forecast
plot(forecast(mymodel, xreg = y1), xlim=c(230,260))
```




```
# we gat a baseline of about 10 nanograms for the time points when no predator was present, and we get

# the quality of the model improves tremendously with the incorporation of trhe explanatory variable
```