Time Series Analysis

shang-chieh0830

2023-03-01

Contents

Chapter 1

About

This book is a concise lecture note about *Time Series Analysis*.

The content of this book is from the course Time Series Analysis taught by Chris Bilder. You can check his YouTube channel to get full(and correct) information about this course.

Again, I do **NOT** own the content of this book. I write this book only for studying. All credits belong to Chris Bilder.

If there is any copyright concerns, I will make this book private ASAP.

Chapter 2

Introduction to R

We will go over some of the basic R operations in this chapter.

If you have questions, you should check Chris Bilder's website for full information.

2.1 Basic Operation

```r
2+2
#> [1] 4
```

```r
2^3
#> [1] 8
```

```r
# calculate the cdf of std. normal
pnorm(1.96) # 1.96 is the quantile
#> [1] 0.9750021
```

```r
log(1)
#> [1] 0
```

```r
sin(pi/2)
#> [1] 1
```

```r
3/4
#> [1] 0.75
```

```
save <- 2+2
save
#> [1] 4
```

```
objects()
#> [1] "save"
```

```
ls()
#> [1] "save"
```

```
# quit operaiton
# q()
```

2.2 Vectors

```
x <- c(1,2,3,4,5)
x
#> [1] 1 2 3 4 5
```

```
sd(x)
#> [1] 1.581139
```

```
mysd <- function(x){
  cat(" My data \n", x, "\n has std deviation",sqrt(var(x)))
}


mysd(x)
#>  My data
#>  1 2 3 4 5
#>  has std deviation 1.581139
```

```
pnorm(q=1.96, mean=1.96, sd=1)
#> [1] 0.5
```

The full syntax for `pnorm()` is `pnorm(q, mean = 0, sd = 1, lower.tail = TRUE, log.p = FALSE)`

```
pnorm(q=c(-1.96,1.96))
#> [1] 0.0249979 0.9750021
```

```
x <- c(3.68, -3.63, 0.80, 3.03, -9.86, -8.66,
    -2.38, 8.94, 0.52, 1.25)

y <- c(0.55, 1.65, 0.98, -0.07, -0.01, -0.31,
    -0.34, -1.38, -1.32, 0.53)

x+y
#>  [1]  4.23 -1.98  1.78  2.96 -9.87 -8.97 -2.72  7.56 -0.80
#> [10]  1.78

x*y
#>  [1]   2.0240  -5.9895   0.7840  -0.2121   0.0986   2.6846
#>  [7]   0.8092 -12.3372  -0.6864   0.6625
```

```
mean(x)
#> [1] -0.631
x-mean(x)
#>  [1]  4.311 -2.999  1.431  3.661 -9.229 -8.029 -1.749  9.571
#>  [9]  1.151  1.881

x*2
#>  [1]   7.36  -7.26   1.60   6.06 -19.72 -17.32  -4.76  17.88
#>  [9]   1.04   2.50
```

The element(elt)-wise operation makes our life easier.

2.3 Files

Click gpa.csv to download the GPA csv file.

Click gpa.txt to download the GPA txt file.

```
getwd()
#> [1] "/Users/weishangjie/Documents/GitHub/Time-Series-Analysis"
```

```
gpatxt <- read.table("gpa.txt", header=TRUE, sep="")
gpacsv <- read.csv("gpa.csv")
```

```
gpacsv$HSGPA
#>  [1] 3.04 2.35 2.70 2.55 2.83 4.32 3.39 2.32 2.69 2.83 2.39
#> [12] 3.65 2.85 3.83 2.22 1.98 2.88 4.00 2.28 2.88
```

```
gpacsv$CollegeGPA
#>  [1] 3.10 2.30 3.00 2.45 2.50 3.70 3.40 2.60 2.80 3.60 2.00
#> [12] 2.90 3.30 3.20 2.80 2.40 2.60 3.80 2.20 2.60
```

```
gpacsv[1,1] # [row, col]
#> [1] 3.04
```

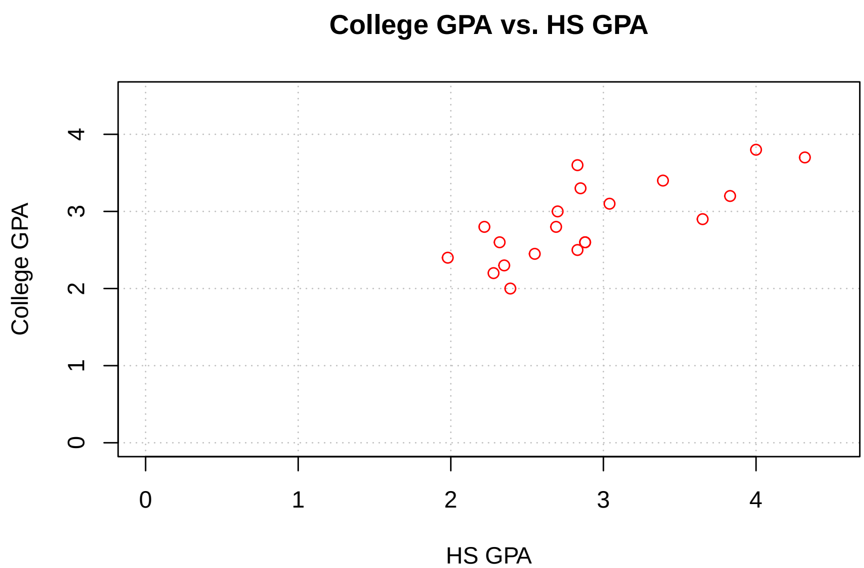
```
gpacsv[,1]
#>  [1] 3.04 2.35 2.70 2.55 2.83 4.32 3.39 2.32 2.69 2.83 2.39
#> [12] 3.65 2.85 3.83 2.22 1.98 2.88 4.00 2.28 2.88
```

```
gpacsv[c(1,3,5),2]
#> [1] 3.1 3.0 2.5
```

```
gpacsv[,"HSGPA"]
#>  [1] 3.04 2.35 2.70 2.55 2.83 4.32 3.39 2.32 2.69 2.83 2.39
#> [12] 3.65 2.85 3.83 2.22 1.98 2.88 4.00 2.28 2.88
```

```
summary(gpacsv)
#>      HSGPA          CollegeGPA   
#>  Min.   :1.980   Min.   :2.000  
#>  1st Qu.:2.380   1st Qu.:2.487  
#>  Median :2.830   Median :2.800  
#>  Mean   :2.899   Mean   :2.862  
#>  3rd Qu.:3.127   3rd Qu.:3.225  
#>  Max.   :4.320   Max.   :3.800
```

```
plot(x = gpacsv$HSGPA, y = gpacsv$CollegeGPA,
     xlab = "HS GPA", ylab = "College GPA",
     main = "College GPA vs. HS GPA",
     xlim = c(0,4.5), ylim = c(0,4.5), col = "red",
    pch = 1, cex = 1.0, panel.first = grid(col = "gray", lty
    = "dotted"))
```



The `plot()` function creates a two dimensional plot of data.

Here are descriptions of its arguments:

- x specifies what is plotted for the x-axis.
- y specifies what is plotted for the y-axis.
- xlab and ylab specify the x-axis and y-axis labels, respectively.
- main specifies the main title of the plot.
- xlim and ylim specify the x-axis and y-axis limits, respectively.
  - Notice the use of the c() function.
- col specifies the color of the plotting points.
  - Run the `colors()` function to see what possible colors can be used.
  - Also, you can see Here for the colors from colors().
- `pch` specifies the plotting characters.
- `cex`specifies the height of the plotting characters. The value 1.0 is the default.
- `panel.first = grid()` specifies grid lines will be plotted.

- The line types can be specified as follows: `1=solid`, `2=dashed`, `3=dotted`, `4=dotdash`, `5=longdash`, `6=twodash` or as one of the character strings `"blank"`, `"solid"`, `"dashed"`, `"dotted"`, `"dotdash"`, `"longdash"`, or `"twodash"`.
  These line type specifications can be used in other functions.

- The `par()`(parameter) function's Help contains more information about the different plotting options!

2.4 Regression

Our model is:

$$CollegeGPA = \beta_0 + \beta_1 HSGPA + \epsilon$$

```
mod.fit <- lm(formula= CollegeGPA~ HSGPA, data=gpacsv)
mod.fit
#>
#> Call:
#> lm(formula = CollegeGPA ~ HSGPA, data = gpacsv)
#>
#> Coefficients:
#> (Intercept)        HSGPA
#>      1.0869       0.6125
```

```
names(mod.fit)
#>  [1] "coefficients"  "residuals"     "effects"
#>  [4] "rank"          "fitted.values" "assign"
#>  [7] "qr"            "df.residual"   "xlevels"
#> [10] "call"          "terms"         "model"
```

```
mod.fit$coefficients
#> (Intercept)       HSGPA
#>   1.0868795   0.6124941
```

```
round(mod.fit$residuals[1:5],2)
#>     1     2     3     4     5
#>  0.15 -0.23  0.26 -0.20 -0.32
```

```
library(tidyverse)
#> -- Attaching packages ------------------- tidyverse 1.3.2 --
#> v ggplot2 3.4.1     v purrr   1.0.1
#> v tibble  3.1.8     v dplyr   1.1.0
```

```
#> v tidyr   1.3.0     v stringr 1.5.0
#> v readr   2.1.4     v forcats 0.5.2
#> -- Conflicts ---------------------- tidyverse_conflicts() --
#> x dplyr::filter() masks stats::filter()
#> x dplyr::lag()    masks stats::lag()
save.fit <- data.frame(gpacsv, C.GPA.hat =
    round(mod.fit$fitted.values,2), residuals =
    round(mod.fit$residuals,2))

save.fit %>% head()
#>   HSGPA CollegeGPA C.GPA.hat residuals
#> 1  3.04       3.10      2.95      0.15
#> 2  2.35       2.30      2.53     -0.23
#> 3  2.70       3.00      2.74      0.26
#> 4  2.55       2.45      2.65     -0.20
#> 5  2.83       2.50      2.82     -0.32
#> 6  4.32       3.70      3.73     -0.03
```

```
summary(mod.fit)
#>
#> Call:
#> lm(formula = CollegeGPA ~ HSGPA, data = gpacsv)
#>
#> Residuals:
#>      Min       1Q   Median       3Q      Max
#> -0.55074 -0.25086  0.01633  0.24242  0.77976
#>
#> Coefficients:
#>             Estimate Std. Error t value Pr(>|t|)
#> (Intercept)   1.0869     0.3666   2.965 0.008299 **
#> HSGPA         0.6125     0.1237   4.953 0.000103 ***
#> ---
#> Signif. codes:
#> 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#>
#> Residual standard error: 0.3437 on 18 degrees of freedom
#> Multiple R-squared:  0.5768, Adjusted R-squared:  0.5533
#> F-statistic: 24.54 on 1 and 18 DF,  p-value: 0.0001027
```

Hence, our estimated regression model is

$$coll\hat{ge}.GPA = \hat{\beta}_0 + \hat{\beta}_1 HS.GPA = 1.0869 + 0.6125HS.GPA$$

```r
# Open a new graphics window
# device new
dev.new(width = 8, height = 6, pointsize = 10)


# 1 row and 2 columns of plots
par(mfrow = c(1,2))
# par= graphic parameter
# mfrow= make a frame by row

# Same scatter plot as before
plot(x = gpacsv$HSGPA, y = gpacsv$CollegeGPA, xlab = "HS
    GPA", ylab = "College GPA", main = "College GPA vs.
    HS GPA", xlim = c(0,4.5), ylim = c(0,4.5), col =
    "red", pch = 1, cex = 1.0, panel.first = grid(col =
    "gray", lty = "dotted"))

# Puts the line y = a + bx on the plot
abline(a = mod.fit$coefficients[1], b =
    mod.fit$coefficients[2], lty = "solid", col =
    "blue", lwd = 2)


# Same scatter plot as before
plot(x = gpacsv$HSGPA, y = gpacsv$CollegeGPA, xlab = "HS
    GPA", ylab = "College GPA", main = "College GPA vs.
    HS GPA", xlim = c(0,4.5), ylim = c(0,4.5), col =
    "red", pch = 1, cex = 1.0, panel.first = grid(col =
    "gray", lty = "dotted"))


# Add line
# expr= math expression
curve(expr = mod.fit$coefficients[1] +
    mod.fit$coefficients[2]*x,
    xlim = c(min(gpacsv$HSGPA),max(gpacsv$HSGPA)),
    col= "blue", add = TRUE, lwd = 2)
```

- The `dev.new()` function can be used to open a new plotting window.
- The `abline()` function can be used to draw straight lines on a plot. In the format used here, the line y = a + bx was drawn where a was the (intercept) and b was the (slope).
- In the second plot, the curve() function was used to draw the line on the plot. This was done to have the line within the range of the high school

GPA values.

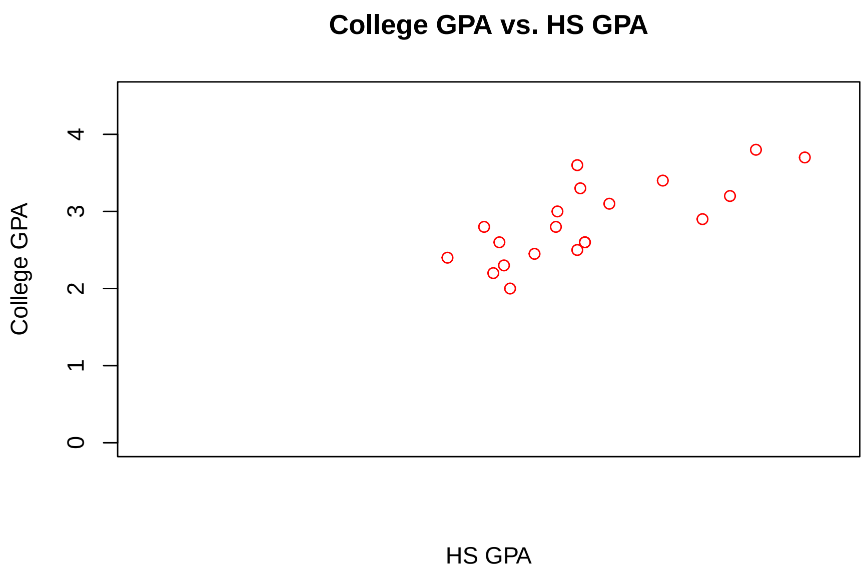
Let's use function to automate what we have done.

```
my.reg.func <- function(x, y, data) {

    # Fit the simple linear regression model and save the results in mod.fit
    mod.fit <- lm(formula = y ~ x, data = data)

    #Open a new graphics window - do not need to
    dev.new(width = 6, height = 6, pointsize = 10)

    # Same scatter plot as before
    plot(x = x, y = y, xlab = "x", ylab = "y", main = "y vs. x", panel.first=grid(col = "gray", l
      "dotted"))

    # Plot model
    curve(expr = mod.fit$coefficients[1] +
      mod.fit$coefficients[2]*x, xlim = c(min(x),max(x)),
      col = "blue", add = TRUE)

    # This is the object returned
    mod.fit
  }
```

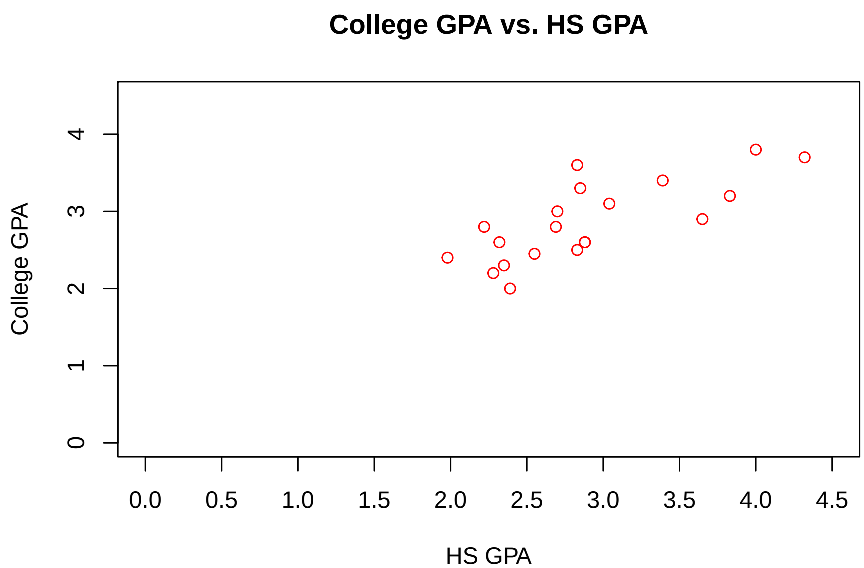
```
save.it <- my.reg.func(x = gpacsv$HSGPA, y =
    gpacsv$CollegeGPA, data = gpacsv)
```

To get specific x-axis or y-axis tick marks on a plot, use the `axis()` function. For example,

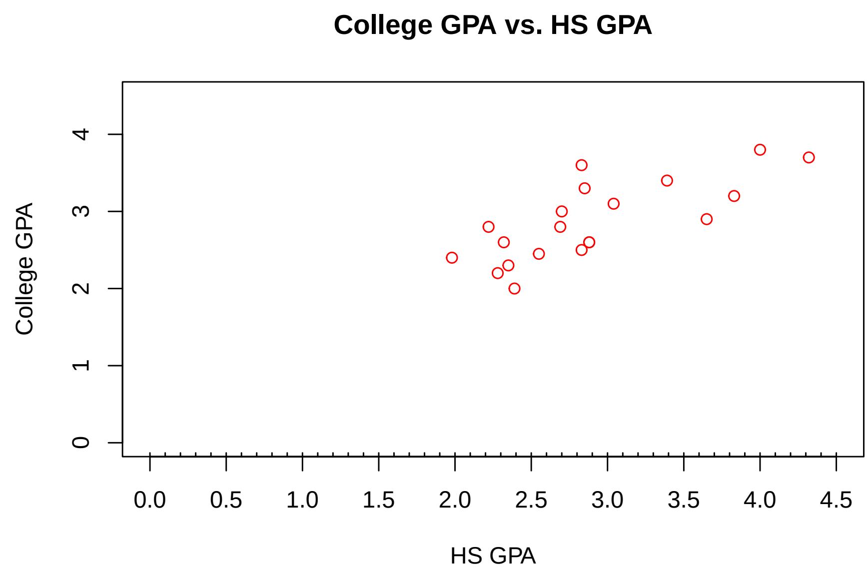
```
#Note that xaxt = "n" tells R to not give any labels on the
#  x-axis (yaxt = "n" works for y-axis)
plot(x = gpacsv$HSGPA, y = gpacsv$CollegeGPA, xlab = "HS GPA",
     ylab = "College GPA", main = "College GPA vs. HS GPA",
     xaxt = "n", xlim = c(0, 4.5), ylim = c(0, 4.5), col =
     "red", pch = 1)
```




```
plot(x = gpacsv$HSGPA, y = gpacsv$CollegeGPA, xlab = "HS GPA",
     ylab = "College GPA", main = "College GPA vs. HS GPA",
     xaxt = "n", xlim = c(0, 4.5), ylim = c(0, 4.5), col =
     "red", pch = 1)

#Major tick marks
axis(side = 1, at = seq(from = 0, to = 4.5, by = 0.5))
```




```
plot(x = gpacsv$HSGPA, y = gpacsv$CollegeGPA, xlab = "HS GPA",
     ylab = "College GPA", main = "College GPA vs. HS GPA",
     xaxt = "n", xlim = c(0, 4.5), ylim = c(0, 4.5), col =
     "red", pch = 1)

#Major tick marks
axis(side = 1, at = seq(from = 0, to = 4.5, by = 0.5))

#Minor tick marks
axis(side = 1, at = seq(from = 0, to = 4.5, by = 0.1), tck
      = 0.01, labels = FALSE)
```



2.5 Object-Oriented Language

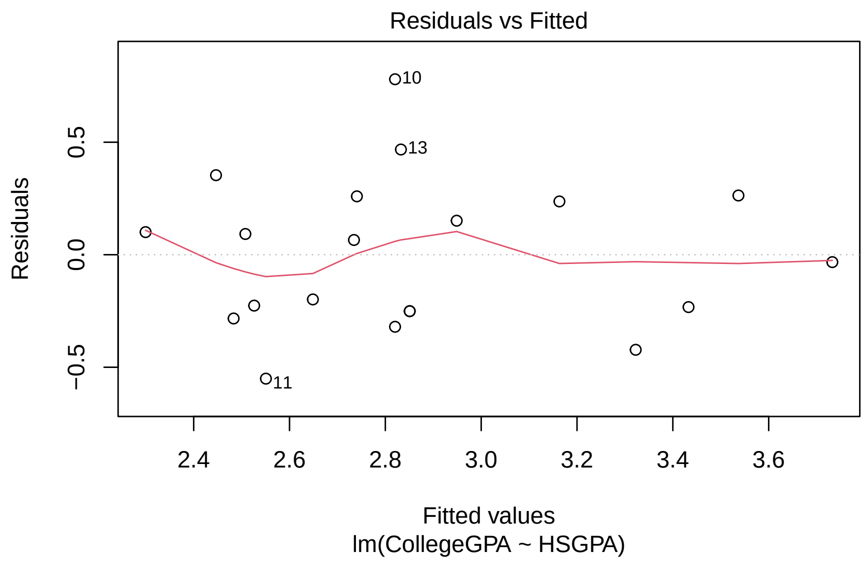
Functions are typically designed to operate on only one or very few classes of objects. However, some functions, like `summary()`, are **generic**, in the sense that essentially different versions of them have been constructed to work with different classes of objects.

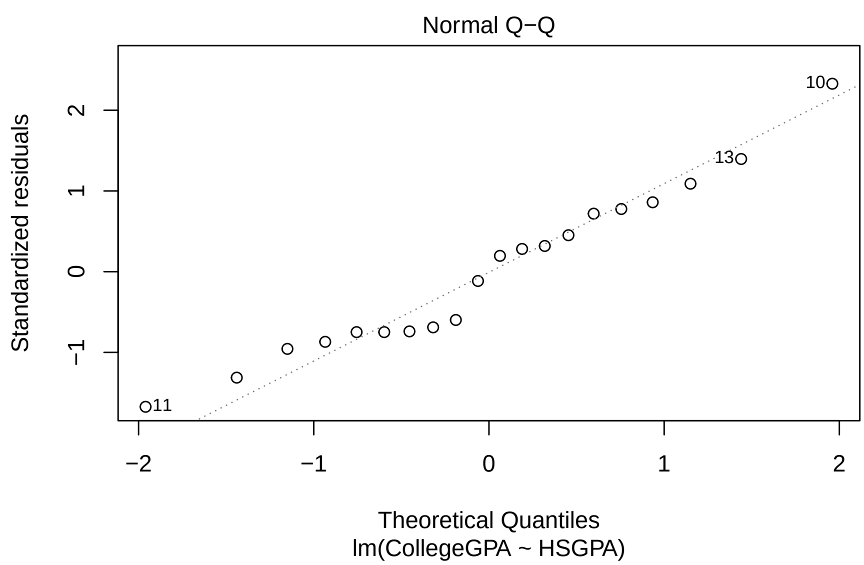
When a generic function is run with an object, R first checks the object's class type and then looks to find a method function with the name format `<generic function>.<class name>`. Below are examples for `summary()`:

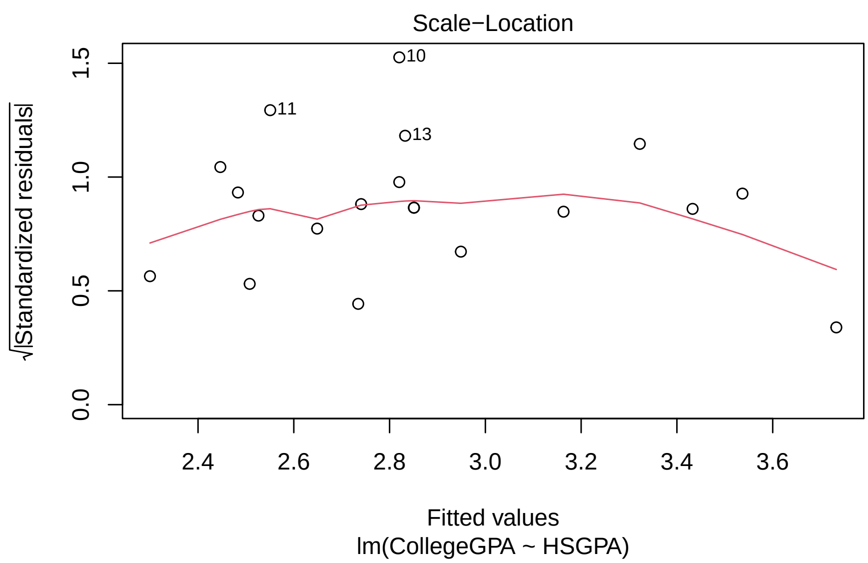
- summary(mod.fit) – The function `summary.lm()` summarizes the regression model
- summary(gpacsv) – The function `summary.data.frame()` summarizes the data frame's contents
- summary.default() – R attempts to run this function if there is no method function for a class

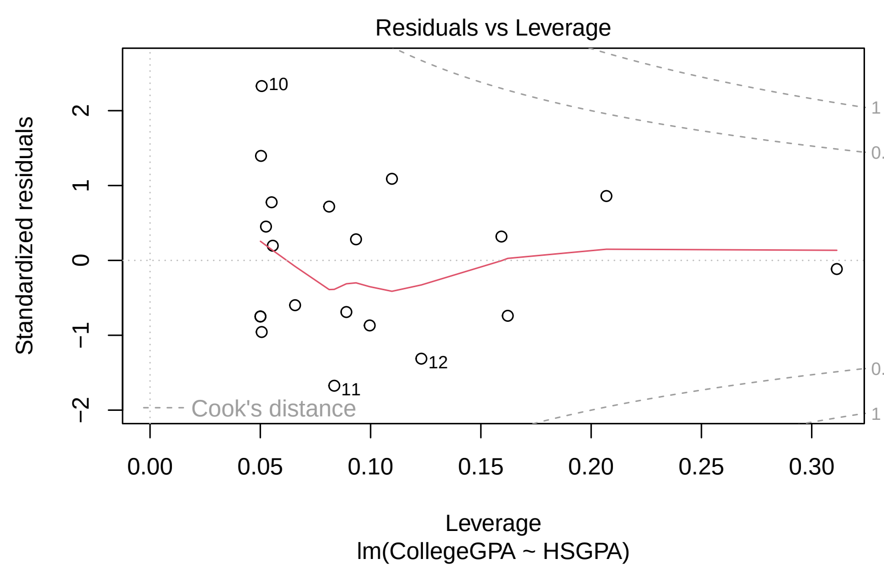
There are many generic functions! For example, `plot()` is a generic function (try`plot(mod.fit)` to see what happens!). We will also see other generic functions like `predict()` later in the notes.

```
plot(mod.fit)
```


Residuals vs Fitted
Residuals
Fitted values
lm(CollegeGPA ~ HSGPA)
10
13
11
0.5
0.0
−0.5
2.4
2.6
2.8
3.0
3.2
3.4
3.6


Normal Q−Q
Standardized residuals
Theoretical Quantiles
lm(CollegeGPA ~ HSGPA)
10
13
11
2
1
0
−1
−2
−1
0
1
2





The purpose of generic functions is to use a familiar language set with any object. So it is convenient to use the same language set no matter the application. This is why R is referred to as an object-oriented language.

To see a list of all method functions associated with a class, use `methods(class = <class name>)`. For the regression example, the method functions associated with the `lm` class are:

```
methods(class="lm") %>% head()
#> [1] "add1.lm"                      "alias.lm"
#> [3] "anova.lm"                     "case.names.lm"
#> [5] "coerce,oldClass,S3-method" "confint.lm"
```

To see a list of all method functions for a generic function, use `methods(generic.function = <generic function name>)`

```
methods(generic.function = "summary") %>% head()
#> [1] "summary,ANY-method"
#> [2] "summary,DBIObject-method"
#> [3] "summary.aov"
#> [4] "summary.aovlist"
#> [5] "summary.aspell"
#> [6] "summary.check_packages_in_dir"
```

Knowing what a name of a particular method function can be helpful to find help on it. For example, the help for `summary()` alone is not very helpful! However, the help for `summary.lm()`provides a lot of useful information about what is summarized for a regression model.

Chapter 3

Time Series Basics-Plotting

In this chapter, we will go over some *Time Series* examples. The aim of this chapter is to help you grasp some of the ideas about plotting.

3.1 Example Data

Click OSU_enroll.csv to download data.

```
osu.enroll <- read.csv(file = "OSU_enroll.csv",
    stringsAsFactors = TRUE)
```

```
head(osu.enroll)
#>   t Semester Year Enrollment      date
#> 1 1     Fall 1989      20110 8/31/1989
#> 2 2   Spring 1990      19128  2/1/1990
#> 3 3   Summer 1990       7553  6/1/1990
#> 4 4     Fall 1990      19591 8/31/1990
#> 5 5   Spring 1991      18361  2/1/1991
#> 6 6   Summer 1991       6702  6/1/1991
```

```
tail(osu.enroll)
#>     t Semester Year Enrollment      date
#> 35 35   Spring 2001      20004  2/1/2001
#> 36 36   Summer 2001       7558  6/1/2001
#> 37 37     Fall 2001      21872 8/31/2001
#> 38 38   Spring 2002      20922  2/1/2002
#> 39 39   Summer 2002       7868  6/1/2002
#> 40 40     Fall 2002      22992 8/31/2002
```

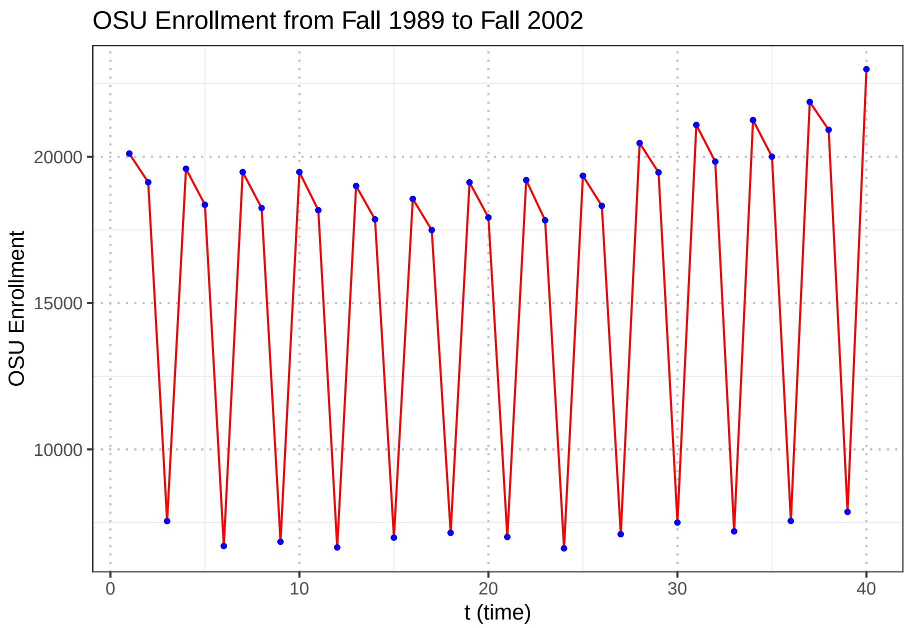
```r
x <- osu.enroll$Enrollment
```

```r
#One way to do plot
dev.new(width = 8, height = 6, pointsize = 10)

# we did not specify y-axis and R put our x in y-axis, time in  x-axis

plot(x = x, ylab = "OSU Enrollment",
       xlab = "t (time)", type="l", col = "red",
       main = "OSU Enrollment from Fall 1989 to Fall 2002",
       panel.first = grid(col = "gray", lty = "dotted"))
```

```r
dev.new(width = 8, height = 6, pointsize = 10)

# we did not specify y-axis and R put our x in y-axis, time in  x-axis

plot(x = x, ylab = "OSU Enrollment",
       xlab = "t (time)", type="l", col = "red",
       main = "OSU Enrollment from Fall 1989 to Fall 2002",
       panel.first = grid(col = "gray", lty = "dotted"))

points(x = osu.enroll$Enrollment, pch = 20, col = "blue")
```

Altenatively, you can do the same thing using ggplot.

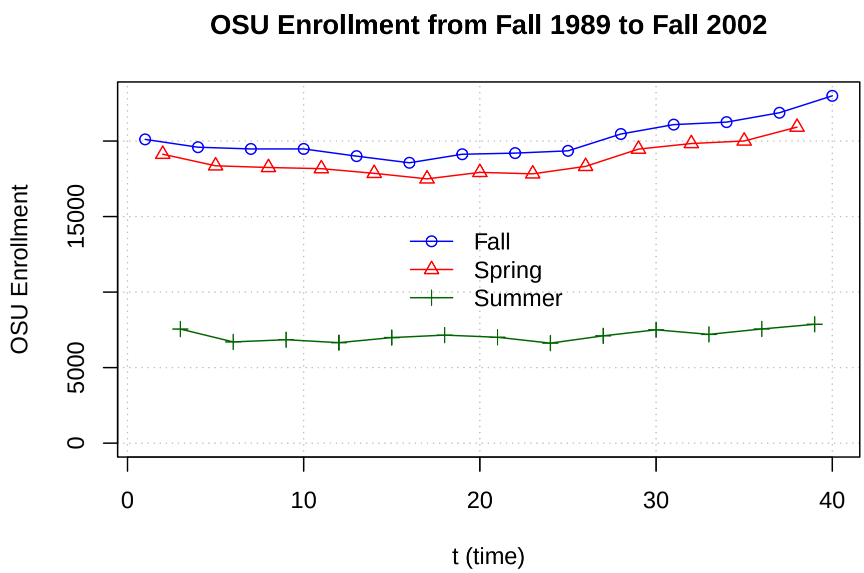
```r
library(ggplot2)

# Create a data frame
df <- data.frame(osu.enroll)

# Create the plot
ggplot(df, aes(x = t, y = Enrollment)) +
  geom_line(colour = "red") +  # Line plot
  geom_point(shape = 20, colour = "blue") +  # Add points
  labs(x = "t (time)", y = "OSU Enrollment",
       title = "OSU Enrollment from Fall 1989 to Fall 2002") +  # Set axis labels and
  theme_bw() +  # Set the theme to a white background with black lines
  theme(panel.grid.major = element_line(colour = "gray", linetype = "dotted"))  # Add
```



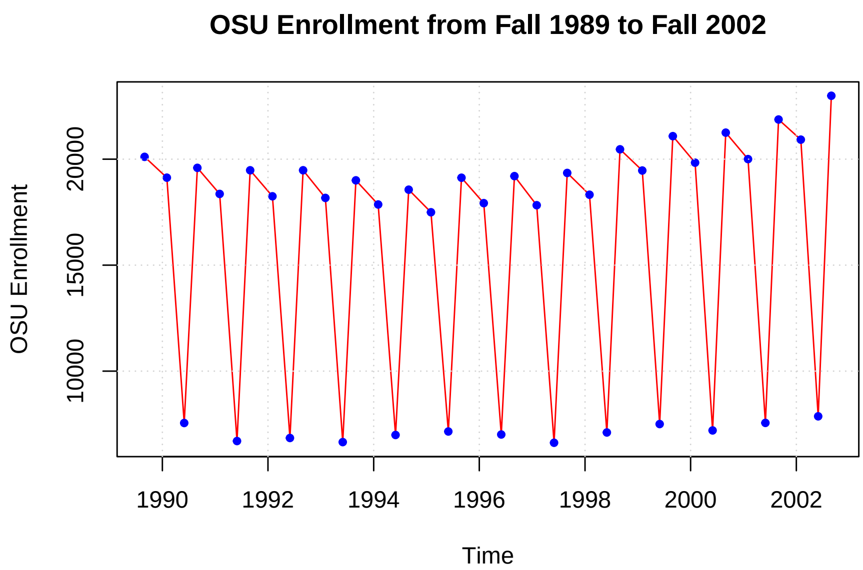
When only x is specified in the `plot()` function, R puts this on the y-axis and uses the observation number on the x-axis.

Compare this to the next plot below where both x and y arguments are specified.

```
#More complicated plot
fall <- osu.enroll[osu.enroll$Semester == "Fall",]
spring <- osu.enroll[osu.enroll$Semester == "Spring",]
summer <- osu.enroll[osu.enroll$Semester == "Summer",]

plot(y = fall$Enrollment, x = fall$t,
    ylab = "OSU Enrollment", xlab = "t (time)",
    col = "blue",
    main = "OSU Enrollment from Fall 1989 to Fall 2002",
    panel.first = grid(col = "gray", lty = "dotted"),
    pch = 1, type = "o", ylim = c(0,max(osu.enroll$Enrollment)))

lines(y = spring$Enrollment, x = spring$t, col = "red",
    type = "o", pch = 2)

lines(y = summer$Enrollment, x = summer$t, col =
    "darkgreen", type = "o", pch = 3)

legend(x="center", legend= c("Fall","Spring","Summer"), pch=c(1,2,3), lty=c(1,1,1), col=c("blue",
```




```
#Another way to do plot with actual dates
plot(y = osu.enroll$Enrollment,
    x = as.Date(osu.enroll$date, format = "%m/%d/%Y"),
    xlab = "Time", type = "l", col = "red",
    main = "OSU Enrollment from Fall 1989 to Fall 2002",
    ylab = "OSU Enrollment")

points(y = osu.enroll$Enrollment,
    x = as.Date(osu.enroll$date, format = "%m/%d/%Y"), pch
    = 20, col = "blue")

#Create own gridlines
# v specifies vertical line; h specifies horizontal line
 abline(v = as.Date(c("1990/1/1", "1992/1/1", "1994/1/1",
    "1996/1/1", "1998/1/1", "2000/1/1", "2002/1/1")),
    lty = "dotted", col = "lightgray")
 abline(h = c(10000, 15000, 20000), lty = "dotted", col =
    "lightgray")
```



3.2 S&P500 Index

Click SP500weekly.csv to download data.

```
SP500 <- read.csv(file="SP500weekly.csv",stringsAsFactors = TRUE)
```

```
head(SP500)
#>   WeekStart   Open   High    Low  Close AdjClose     Volume
#> 1  1/1/1995 459.21 462.49 457.20 460.68   460.68 1199080000
#> 2  1/8/1995 460.67 466.43 458.65 465.97   465.97 1627330000
#> 3 1/15/1995 465.97 470.43 463.99 464.78   464.78 1667400000
#> 4 1/22/1995 464.78 471.36 461.14 470.39   470.39 1628110000
#> 5 1/29/1995 470.39 479.91 467.49 478.65   478.65 1888560000
#> 6  2/5/1995 478.64 482.60 478.36 481.46   481.46 1579920000
```

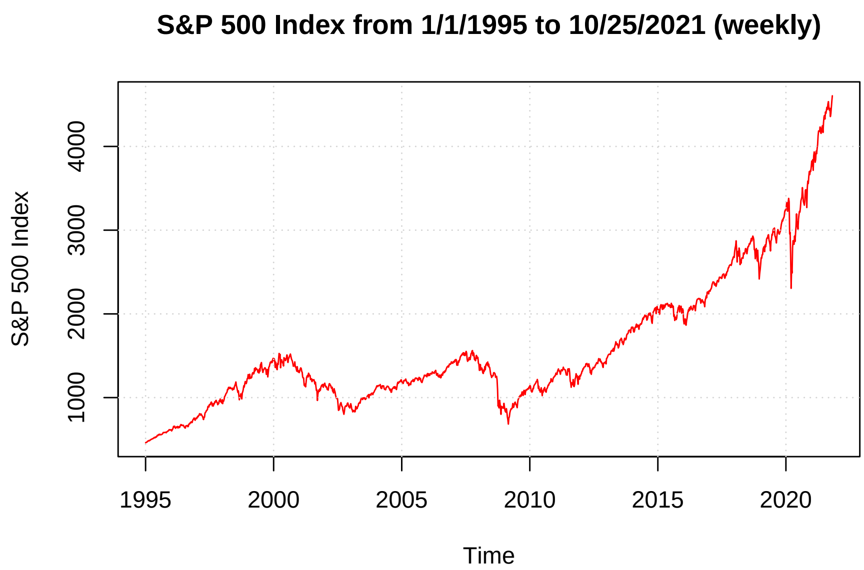
```
tail(SP500)
#>        WeekStart    Open    High     Low   Close AdjClose
#> 1395  9/19/2021 4402.95 4465.40 4305.91 4455.48  4455.48
#> 1396  9/26/2021 4442.12 4457.30 4288.52 4357.04  4357.04
#> 1397  10/3/2021 4348.84 4429.97 4278.94 4391.34  4391.34
#> 1398 10/10/2021 4385.44 4475.82 4329.92 4471.37  4471.37
#> 1399 10/17/2021 4463.72 4559.67 4447.47 4544.90  4544.90
```

```
#> 1400 10/24/2021 4553.69 4608.08 4537.36 4605.38  4605.38
#>            Volume
#> 1395 15697030000
#> 1396 15555390000
#> 1397 14795520000
#> 1398 13758090000
#> 1399 13966070000
#> 1400 16206040000
```

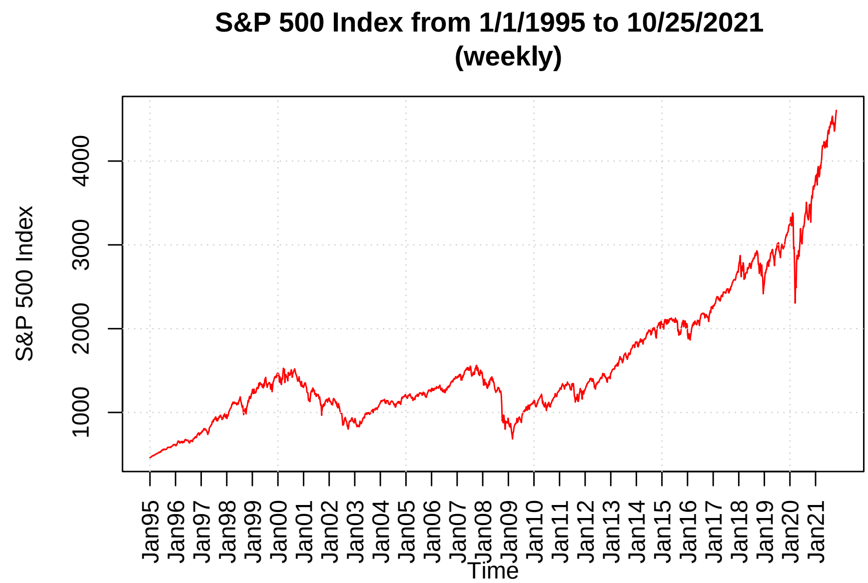
```
x <- SP500$Close
```

```
#One way to do plot
dev.new(width = 8, height = 6, pointsize = 10)
#again, we do not specify y-axis here
plot(x = x, ylab = "S&P 500 Index", xlab = "t (time)",
    type = "l", col = "red", main = "S&P 500 Index from
    1/1/1995 to 10/25/2021 (weekly)",
    panel.first = grid(col = "gray", lty = "dotted"))
```

```
#Another way to do plot with actual dates
plot(y = x, x = as.Date(SP500$WeekStart, format =
    "%m/%d/%Y"), xlab = "Time", type = "l", col = "red", main
    = "S&P 500 Index from 1/1/1995 to 10/25/2021 (weekly)",
    ylab = "S&P 500 Index")

#Create own gridlines
abline(v = as.Date(c("1995/1/1", "2000/1/1", "2005/1/1",
    "2010/1/1", "2015/1/1", "2020/1/1")), lty = "dotted",
    col = "lightgray")

abline(h = seq(from = 0, to = 5000, by = 1000), lty =
    "dotted", col = "lightgray")
```




```
# One more way with fine control of the dates
plot(y = x, x = as.Date(SP500$WeekStart, format =
    "%m/%d/%Y"), xlab = "Time", type = "l", col = "red",
    main = "S&P 500 Index from 1/1/1995 to 10/25/2021
    (weekly)", ylab = "S&P 500 Index", xaxt = "n")

axis.Date(side = 1, at = seq(from = as.Date("1995/1/1"),
    to = as.Date("2021/12/31"), by = "years"), labels =
    format(x = seq(from = as.Date("1995/1/1"), to =
    as.Date("2021/12/31"), by = "years"), format = "%b%y"),
    las = 2)  #las changes orientation of labels

#Create own gridlines
abline(v = as.Date(c("1995/1/1", "2000/1/1", "2005/1/1",
    "2010/1/1", "2015/1/1", "2020/1/1")), lty = "dotted",
    col = "lightgray")
abline(h = seq(from = 0, to = 5000, by = 1000), lty =
    "dotted", col = "lightgray")
```



3.3 Sunspots

Click SN_y_tot_V2.0.csv to download data.

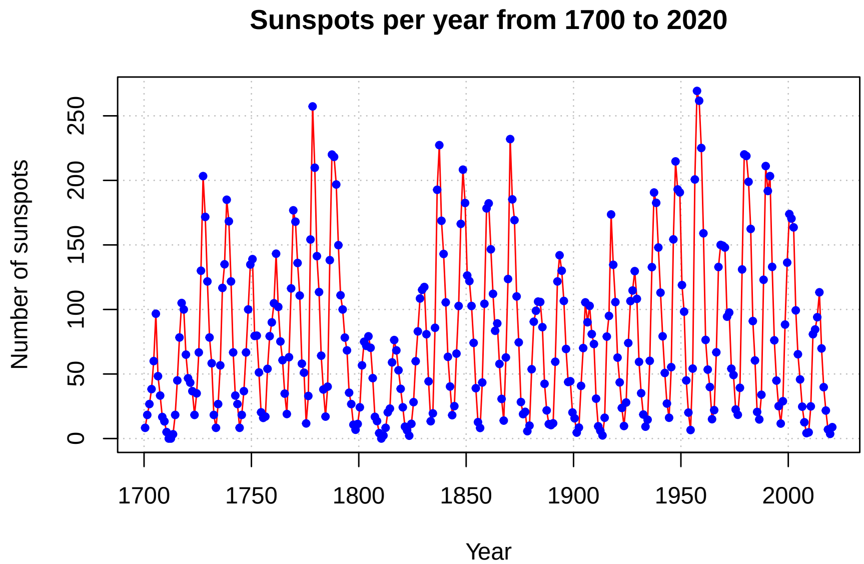
```
sunspots <- read.table(file = "SN_y_tot_V2.0.csv", sep =
    ";", col.names = c("Mid.year", "Mean.total",
   "Mean.SD.total", "Numb.obs.used", "Definitive"))
```

```
head(sunspots)
#>   Mid.year Mean.total Mean.SD.total Numb.obs.used
#> 1   1700.5        8.3            -1            -1
#> 2   1701.5       18.3            -1            -1
#> 3   1702.5       26.7            -1            -1
#> 4   1703.5       38.3            -1            -1
#> 5   1704.5       60.0            -1            -1
#> 6   1705.5       96.7            -1            -1
#>   Definitive
#> 1          1
#> 2          1
#> 3          1
#> 4          1
#> 5          1
#> 6          1
```

```
tail(sunspots)
#>     Mid.year Mean.total Mean.SD.total Numb.obs.used
#> 316   2015.5       69.8           6.4          8903
#> 317   2016.5       39.8           3.9          9940
#> 318   2017.5       21.7           2.5         11444
#> 319   2018.5        7.0           1.1         12611
#> 320   2019.5        3.6           0.5         12884
#> 321   2020.5        8.8           4.1         14440
#>     Definitive
#> 316          1
#> 317          1
#> 318          1
#> 319          1
#> 320          1
#> 321          1
```

```
dev.new(width = 8, height = 6, pointsize = 10)

#again, we did not specify y-axis here
plot(x = sunspots$Mean.total, ylab = "Number of
    sunspots", xlab = "t (time)", type = "l", col = "red",
    main = "Sunspots per year from 1700 to 2020",
    panel.first = grid(col = "gray", lty = "dotted"))

points(x = sunspots$Mean.total, pch = 20, col = "blue")
```

```
# Include dates
plot(y = sunspots$Mean.total, x = sunspots$Mid.year, ylab
    = "Number of sunspots", xlab = "Year", type = "l", col
    = "red", main = "Sunspots per year from 1700 to 2020",
    panel.first = grid(col = "gray", lty = "dotted"))

points(y = sunspots$Mean.total, x = sunspots$Mid.year,
    pch = 20, col = "blue")
```




```
#Convert to an object of class "ts"

x <- ts(data = sunspots$Mean.total, start = 1700, frequency
    = 1)

x
#> Time Series:
#> Start = 1700
#> End = 2020
#> Frequency = 1
#>   [1]   8.3  18.3  26.7  38.3  60.0  96.7  48.3  33.3  16.7
#>  [10]  13.3   5.0   0.0   0.0   3.3  18.3  45.0  78.3 105.0
#>  [19] 100.0  65.0  46.7  43.3  36.7  18.3  35.0  66.7 130.0
#>  [28] 203.3 171.7 121.7  78.3  58.3  18.3   8.3  26.7  56.7
#>  [37] 116.7 135.0 185.0 168.3 121.7  66.7  33.3  26.7   8.3
#>  [46]  18.3  36.7  66.7 100.0 134.8 139.0  79.5  79.7  51.2
#>  [55]  20.3  16.0  17.0  54.0  79.3  90.0 104.8 143.2 102.0
#>  [64]  75.2  60.7  34.8  19.0  63.0 116.3 176.8 168.0 136.0
#>  [73] 110.8  58.0  51.0  11.7  33.0 154.2 257.3 209.8 141.3
#>  [82] 113.5  64.2  38.0  17.0  40.2 138.2 220.0 218.2 196.8
#>  [91] 149.8 111.0 100.0  78.2  68.3  35.5  26.7  10.7   6.8
#> [100]  11.3  24.2  56.7  75.0  71.8  79.2  70.3  46.8  16.8
#> [109]  13.5   4.2   0.0   2.3   8.3  20.3  23.2  59.0  76.3
#> [118]  68.3  52.9  38.5  24.2   9.2   6.3   2.2  11.4  28.2
#> [127]  59.9  83.0 108.5 115.2 117.4  80.8  44.3  13.4  19.5
```

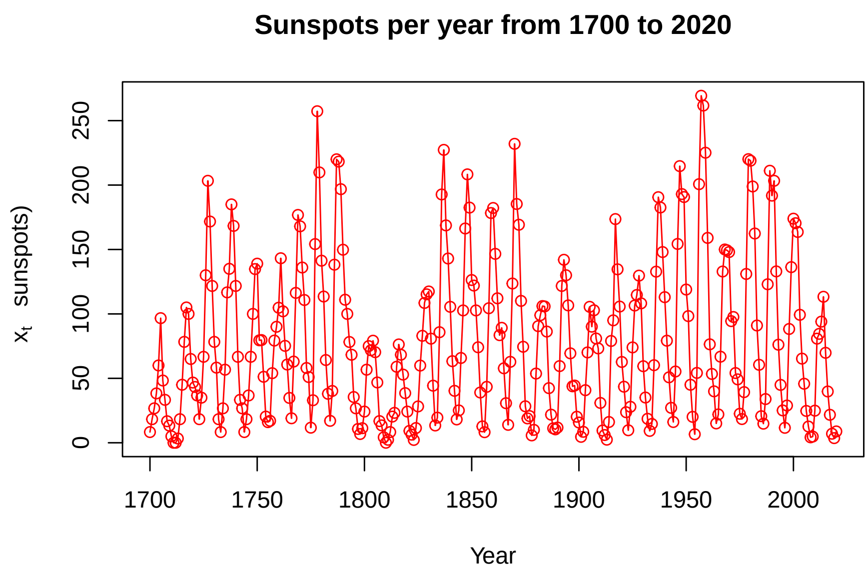
```
#> [136]  85.8 192.7 227.3 168.7 143.0 105.5  63.3  40.3  18.1
#> [145]  25.1  65.8 102.7 166.3 208.3 182.5 126.3 122.0 102.7
#> [154]  74.1  39.0  12.7   8.2  43.4 104.4 178.3 182.2 146.6
#> [163] 112.1  83.5  89.2  57.8  30.7  13.9  62.8 123.6 232.0
#> [172] 185.3 169.2 110.1  74.5  28.3  18.9  20.7   5.7  10.0
#> [181]  53.7  90.5  99.0 106.1 105.8  86.3  42.4  21.8  11.2
#> [190]  10.4  11.8  59.5 121.7 142.0 130.0 106.6  69.4  43.8
#> [199]  44.4  20.2  15.7   4.6   8.5  40.8  70.1 105.5  90.1
#> [208] 102.8  80.9  73.2  30.9   9.5   6.0   2.4  16.1  79.0
#> [217]  95.0 173.6 134.6 105.7  62.7  43.5  23.7   9.7  27.9
#> [226]  74.0 106.5 114.7 129.7 108.2  59.4  35.1  18.6   9.2
#> [235]  14.6  60.2 132.8 190.6 182.6 148.0 113.0  79.2  50.8
#> [244]  27.1  16.1  55.3 154.3 214.7 193.0 190.7 118.9  98.3
#> [253]  45.0  20.1   6.6  54.2 200.7 269.3 261.7 225.1 159.0
#> [262]  76.4  53.4  39.9  15.0  22.0  66.8 132.9 150.0 149.4
#> [271] 148.0  94.4  97.6  54.1  49.2  22.5  18.4  39.3 131.0
#> [280] 220.1 218.9 198.9 162.4  91.0  60.5  20.6  14.8  33.9
#> [289] 123.0 211.1 191.8 203.3 133.0  76.1  44.9  25.1  11.6
#> [298]  28.9  88.3 136.3 173.9 170.4 163.6  99.3  65.3  45.8
#> [307]  24.7  12.6   4.2   4.8  24.9  80.8  84.5  94.0 113.3
#> [316]  69.8  39.8  21.7   7.0   3.6   8.8
```

```
class(x)
#> [1] "ts"

class(sunspots$Mean.total)
#> [1] "numeric"
```

3.3.1 plot.ts()

plot() is a generic function - uses the plot.ts() method function

```
# we did not specify y-axis here, but x is now ts
plot(x = x, ylab = expression(paste(x[t], " (Number of
   sunspots)")), xlab = "Year", type = "o", col = "red", main
   = "Sunspots per year from 1700 to 2020")
#> Warning in title(main = main, xlab = xlab, ylab = ylab,
#> ...): font metrics unknown for character 0xa

#> Warning in title(main = main, xlab = xlab, ylab = ylab,
#> ...): font metrics unknown for character 0xa
```


Sunspots per year from 1700 to 2020
x_t sunspots)
0
50
100
150
200
250
1700
1750
1800
1850
1900
1950
2000
Year