Visualization–Base R

Max Turgeon

STAT 3150–Statistical Computing

Base R

- Whenever I talk about "base R", what I mean is the functions that are part of the basic installation:
    - E.g. `base`, `stats`, and `graphics`.
- Base R graphics are very powerful, but they typically require more manual tinkering to create "publication-ready" graphs.

Main functions for data transformation i

- `transform`: Create a new variable as a function of the other variables

```
# Switch to litres per 100km
transform(mtcars, litres_per_100km = mpg/235.215)
```

Main functions for data transformation ii

- `subset`: Select rows or columns

```r
# Only keep rows where cyl is equal to 6 or 8
subset(mtcars, subset = cyl %in% c(6, 8))
# Only keep cyl and mpg columns
subset(mtcars, select = c(cyl, mpg))
```

Main functions for data transformation iii

- `tapply`: Apply function to an array, grouping rows according to another variable.
  - **Note**: We use the operator $ to extract the columns as vectors.

```
# Average mpg for each value of cyl
tapply(mtcars$mpg, mtcars$cyl, mean)
```

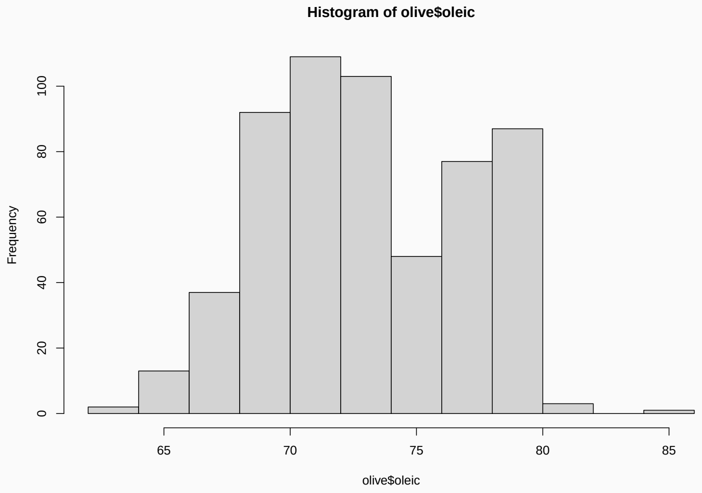
```
##        4        6        8
## 26.66364 19.74286 15.10000
```

Data Visualization

Histogram i

```r
library(dslabs)

# Create histogram for oleic acid
hist(olive$oleic)
```

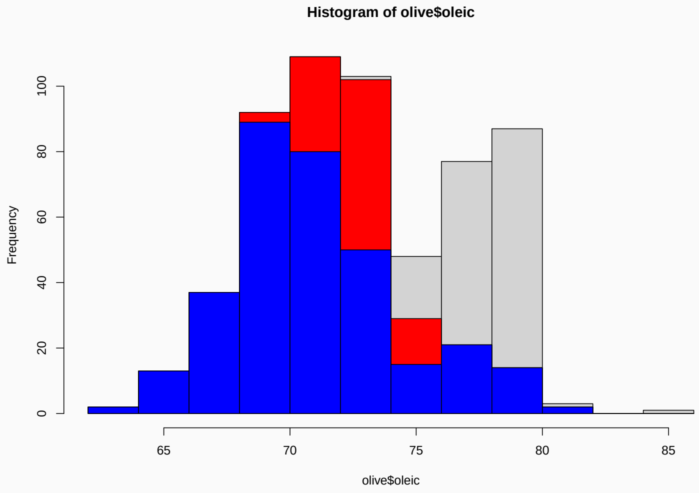
Histogram ii



Histogram iii

```r
# Look at distribution by region
# using stacked histograms
top_hist <- hist(olive$oleic)
# Gray: North; Red: Sardinia; Blue: South
hist(olive$oleic[olive$region != "Northern Italy"],
     breaks = top_hist$breaks,
     col = "red", add = TRUE)
hist(olive$oleic[olive$region == "Southern Italy"],
     breaks = top_hist$breaks,
     col = "blue", add = TRUE)
```

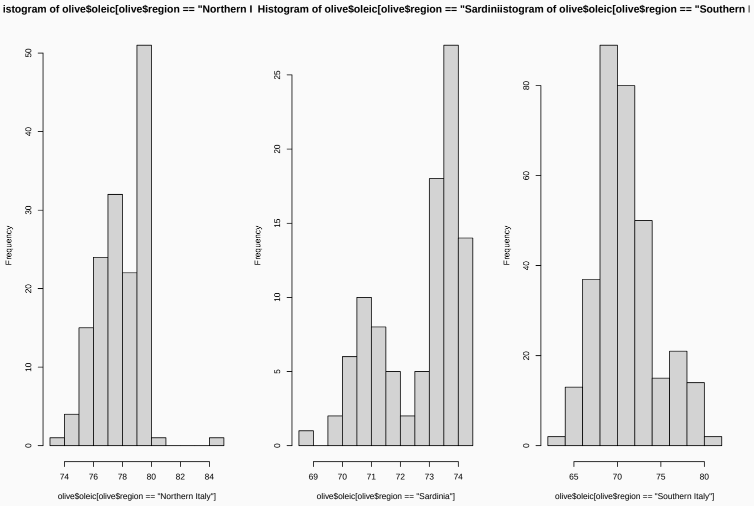
Histogram iv



Histogram v

```r
# Split plotting device in three
par(mfrow = c(1,3))
hist(olive$oleic[olive$region == ”Northern Italy”])
hist(olive$oleic[olive$region == ”Sardinia”])
hist(olive$oleic[olive$region == ”Southern Italy”])
```

Histogram vi



`with` function i

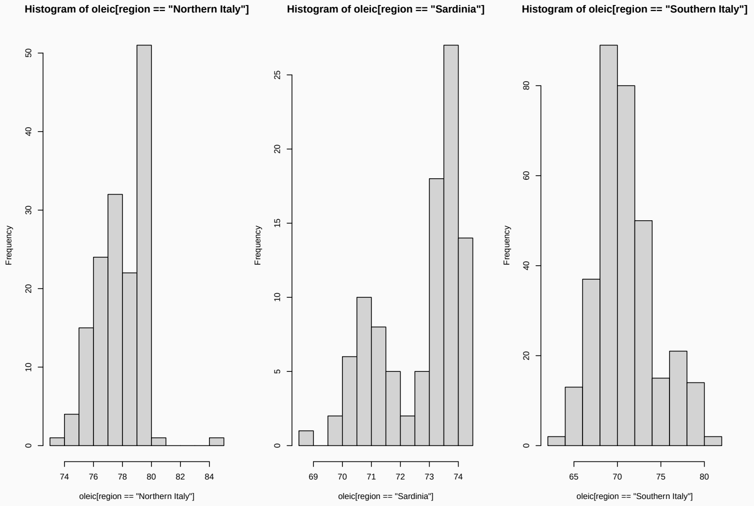
- It can quickly become cumbersome to always use the `$` operator to extract variables.
- The function `with` can help:
  - Inside `with`, the columns of a `data.frame` are like variables.

```r
# Instead of this:
hist(olive$oleic)
# You can write this:
with(olive, hist(oleic))
```

with function ii

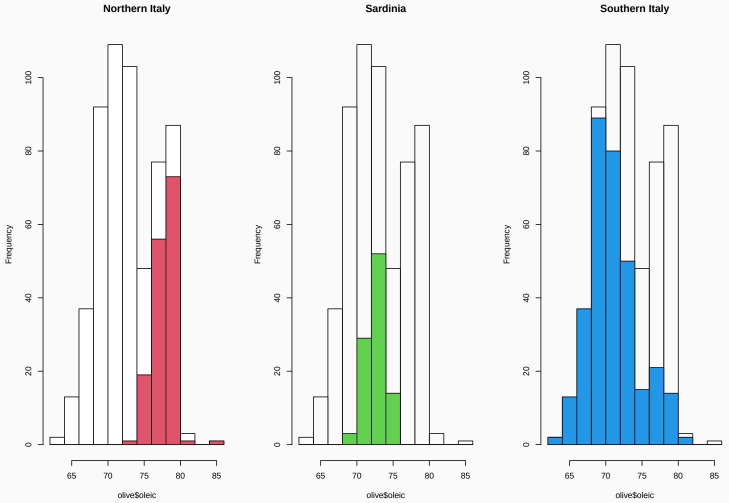
```r
# Same as above, but using with
par(mfrow = c(1,3))
# 2nd argument of with is an expression
# i.e. everything between {}
with(olive, {
  hist(oleic[region == ”Northern Italy”])
  hist(oleic[region == ”Sardinia”])
  hist(oleic[region == ”Southern Italy”])
})
```

`with` function iii



with function iv

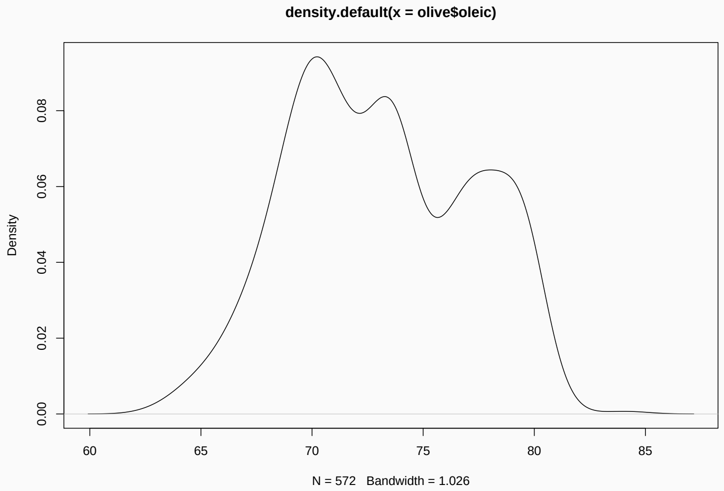
```r
# A more complex example
par(mfrow = c(1,3))
hist_bg <- hist(olive$oleic, col = 'white',
                main = "Northern Italy")
with(olive, {
  hist(oleic[region == "Northern Italy"],
       col = 2, add = TRUE, breaks = hist_bg$breaks)
  plot(hist_bg, main = "Sardinia")
  hist(oleic[region == "Sardinia"],
       col = 3, add = TRUE, breaks = hist_bg$breaks)
  plot(hist_bg, main = "Southern Italy")
  hist(oleic[region == "Southern Italy"],
       col = 4, add = TRUE, breaks = hist_bg$breaks)
})
```


Northern Italy
Sardinia
Southern Italy
Frequency
olive$oleic
0
20
40
60
80
100
65
70
75
80
85

Density plot i

```
plot(density(olive$oleic))
```

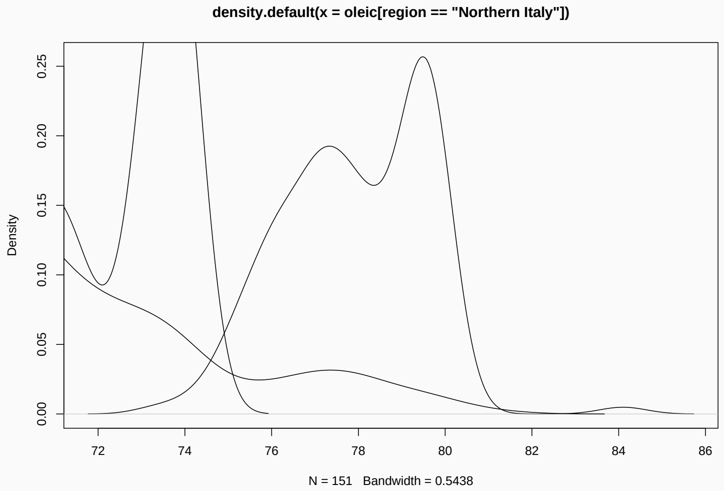
Density plot ii



Density plot iii

```
# By region--first try
with(olive, {
  plot(density(oleic[region == "Northern Italy"]))
  lines(density(oleic[region == "Sardinia"]))
  lines(density(oleic[region == "Southern Italy"]))
})
```

Density plot iv



Density plot v

```
# By region--second try
dens_ni <- with(olive,
                density(oleic[region == "Northern Italy"]))
dens_sa <- with(olive,
                density(oleic[region == "Sardinia"]))
dens_si <- with(olive,
                density(oleic[region == "Southern Italy"]))

str(dens_ni)
```

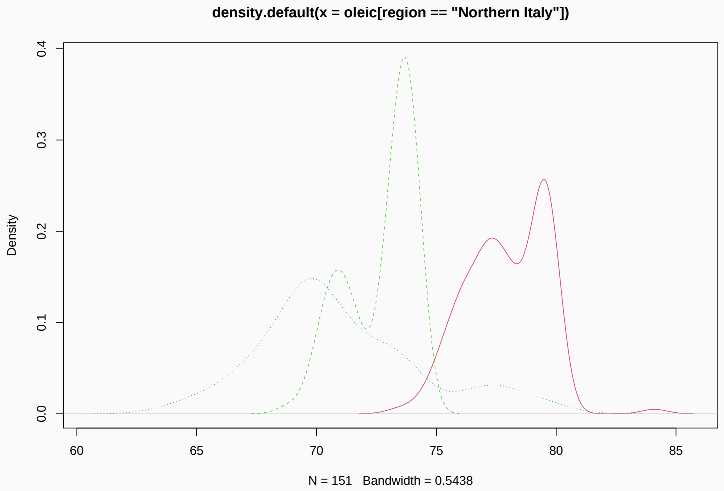
Density plot vi

```
## List of 7
##  $ x        : num [1:512] 71.8 71.8 71.8 71.9 71.9 .
##  $ y        : num [1:512] 0.000055 0.000064 0.000074
##  $ bw       : num 0.544
##  $ n        : int 151
##  $ call     : language density.default(x = oleic[reg
##  $ data.name: chr ”oleic[region == \”Northern Italy\”
##  $ has.na   : logi FALSE
##  - attr(*, ”class”)= chr ”density”
```

Density plot vii

```r
xlim <- range(c(dens_ni$x, dens_sa$x, dens_si$x))
ylim <- range(c(dens_ni$y, dens_sa$y, dens_si$y))

plot(dens_ni, xlim = xlim, ylim = ylim, col = 2)
lines(dens_sa, lty = 2, col = 3)
lines(dens_si, lty = 3, col = 4)
```

Density plot viii



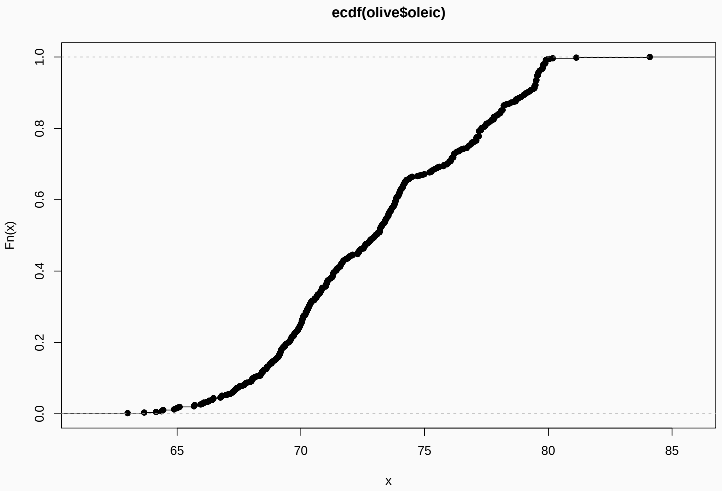
Density plot–Comments

- We used `plot` to get the first density plot, then we use `lines` to `add` the other density plots.
    - If we used `plot` again, it would create a new graph.
    - Also, the first call to plot determines the limits of the axes.
- By looking at the structure of `dens_ni`, we could see that it stored the `x` and `y` values of the density estimate.
    - Therefore, we were able combine all three estimates and make sure the first plot was large enough.
- In base R, we control colour with `col` (using numbers or character strings). We control the line type with `lty`.

ECDF plot i

```r
plot(ecdf(olive$oleic))
```

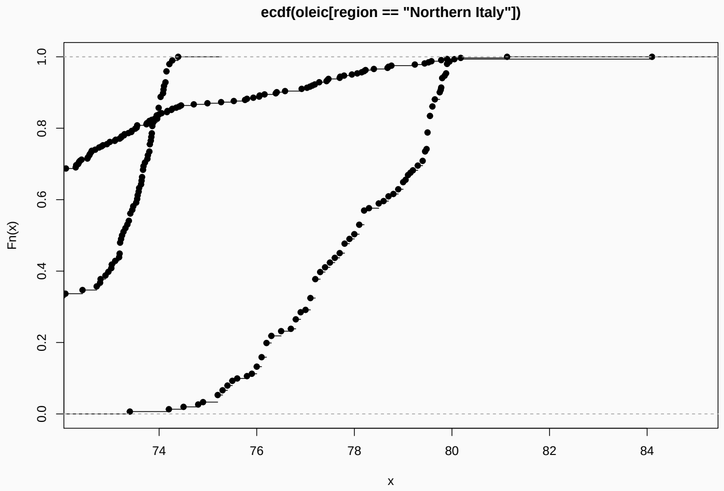
ECDF plot ii



ECDF plot iii

```r
# By region--first try
with(olive, {
  plot(ecdf(oleic[region == "Northern Italy"]))
  lines(ecdf(oleic[region == "Sardinia"]))
  lines(ecdf(oleic[region == "Southern Italy"]))
})
```

ECDF plot iv


ecdf(oleic[region == "Northern Italy"])
Fn(x)
0.0
0.2
0.4
0.6
0.8
1.0
74
76
78
80
82
84
x

ECDF plot v

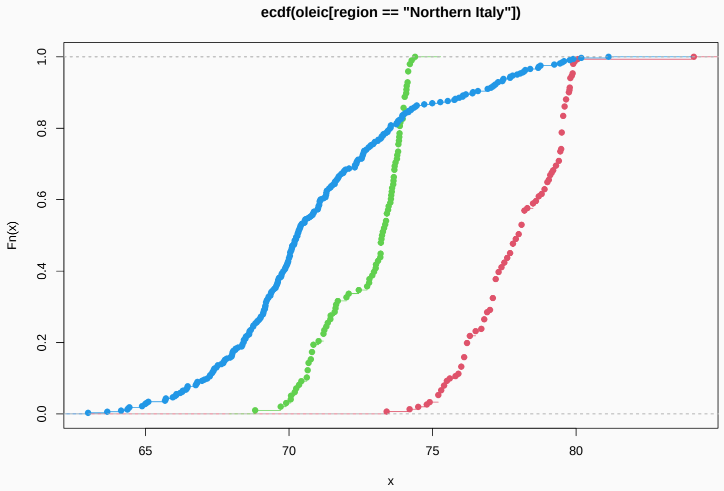
```r
# By region--second try
ecdf_ni <- with(olive,
                ecdf(oleic[region == "Northern Italy"]))
ecdf_sa <- with(olive,
                ecdf(oleic[region == "Sardinia"]))
ecdf_si <- with(olive,
                ecdf(oleic[region == "Southern Italy"]))

str(ecdf_ni) # This is a function!
```

ECDF plot vi

```
## function (v)
##  - attr(*, "class")= chr [1:3] "ecdf" "stepfun" "func
##  - attr(*, "call")= language ecdf(oleic[region == "No
```

```r
xlim <- range(olive$oleic)

plot(ecdf_ni, xlim = xlim, col = 2)
lines(ecdf_sa, col = 3)
lines(ecdf_si, col = 4)
```

ECDF plot vii



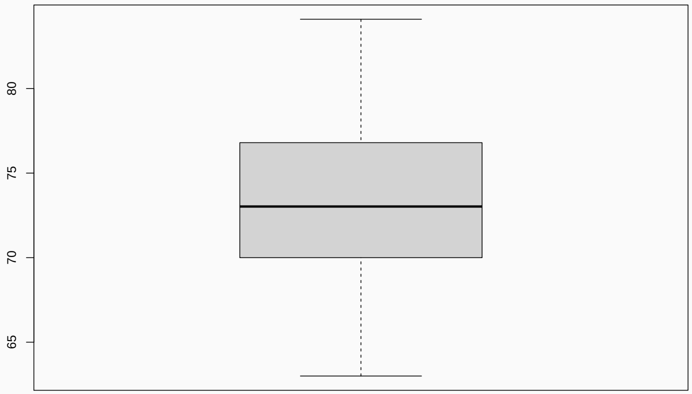
ECDF plot–Comments

- The output of `ecdf` is not a `matrix` or a `data.frame`, but a function!
    - In particular, it doesn't explicitly contain the `x` and `y` values, like density.
- But we can get the bounds from the original data:
    - The `y` limits should be `c(0, 1)`, because they are probabilities.
    - The `x`-axis should cover all values in the full dataset.

Boxplot i

```
boxplot(olive$oleic)
```

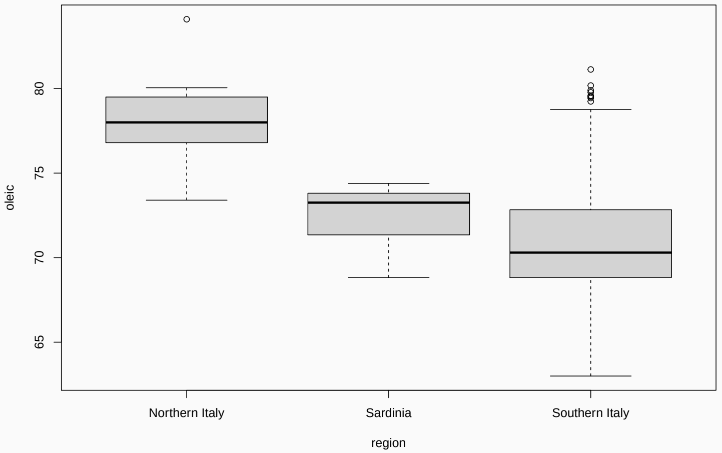
Boxplot ii



Boxplot iii

```r
# Split by region using a formula
boxplot(oleic ~ region, data = olive)
```

Boxplot iv

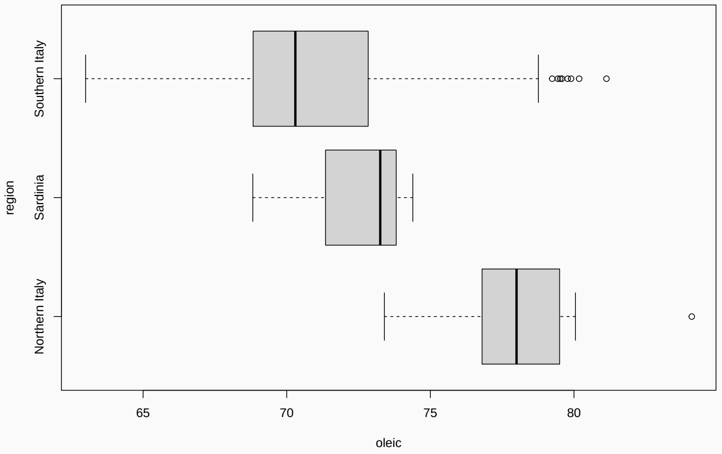


80
75
70
65
oleic
Northern Italy
Sardinia
Southern Italy
region

Boxplot v

```r
# Flip boxplots
boxplot(oleic ~ region, data = olive,
        horizontal = TRUE)
```

Boxplot vi

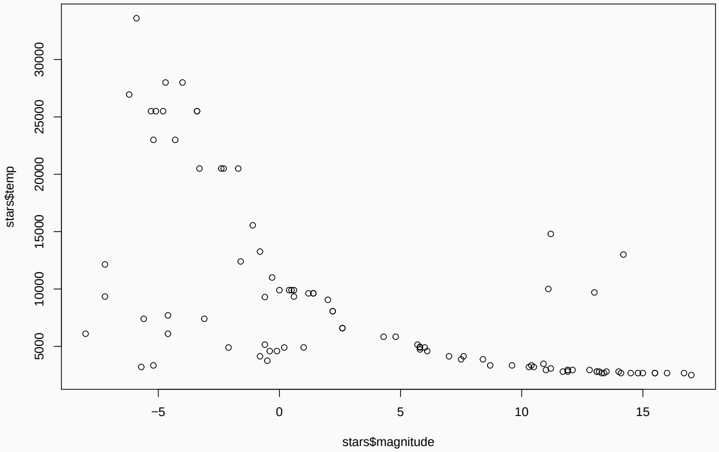

Southern Italy
Sardinia
Northern Italy
region
65
70
75
80
oleic

Bivariate plots

Scatter plot i

```r
plot(stars$magnitude,
     stars$temp)
```

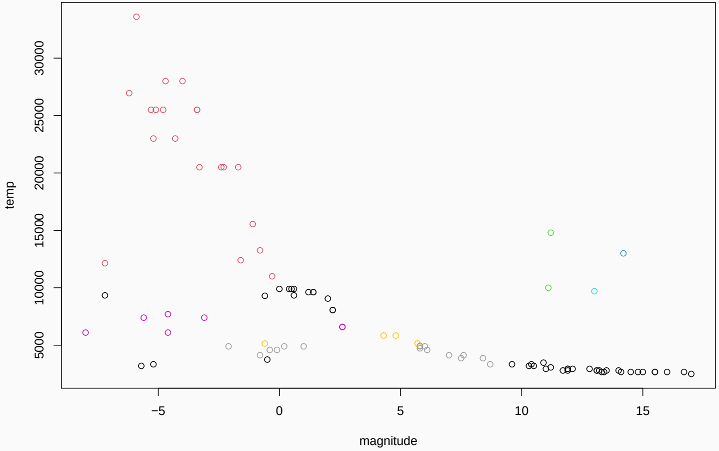
Scatter plot ii



Scatter plot iii

```r
# Add colour for type of stars
with(stars, plot(magnitude, temp,
                 col = factor(type)))
```

Scatter plot iv



Scatter plot v

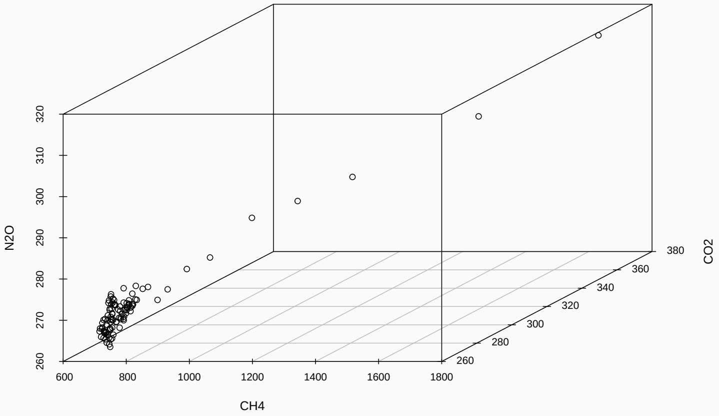
```r
library(scatterplot3d)
library(tidyr)
wide_data <- spread(greenhouse_gases,
                    gas, concentration)

head(wide_data, n = 3)
```

```
##   year   CH4   CO2   N2O
## 1   20 638.1 277.7 263.2
## 2   40 631.1 277.8 263.3
## 3   60 628.2 277.3 264.4
```

Scatter plot vi

```
with(wide_data,
     scatterplot3d(CH4,   # x axis
                   CO2,   # y axis
                   N2O    # z axis
))
```

Scatter plot vii



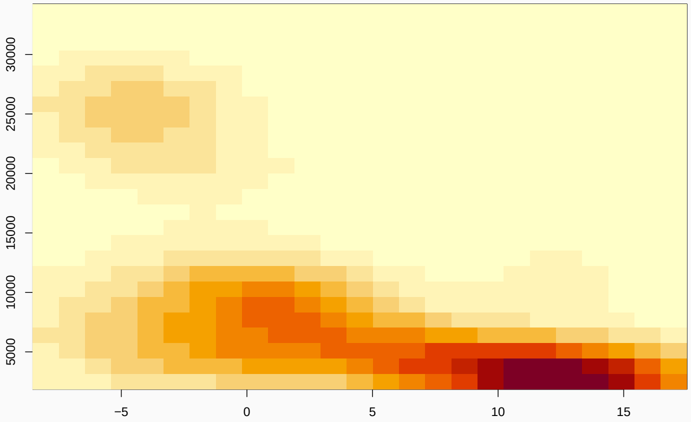
Scatter plot–Comments

- Remember: for colours, `col` is either a number or a character string describing the colour (e.g. `col = ”white”`)
    - The variable `type` is a character string, but not describing colours...
    - **Solution**: Transform into `factor`, which is treated as a number (i.e. the order of category).
- We can do 3D scatterplots, but the depth of a point is hard to read.

Bivariate density plot i

```r
library(MASS)

image(kde2d(stars$magnitude,
            stars$temp))
```

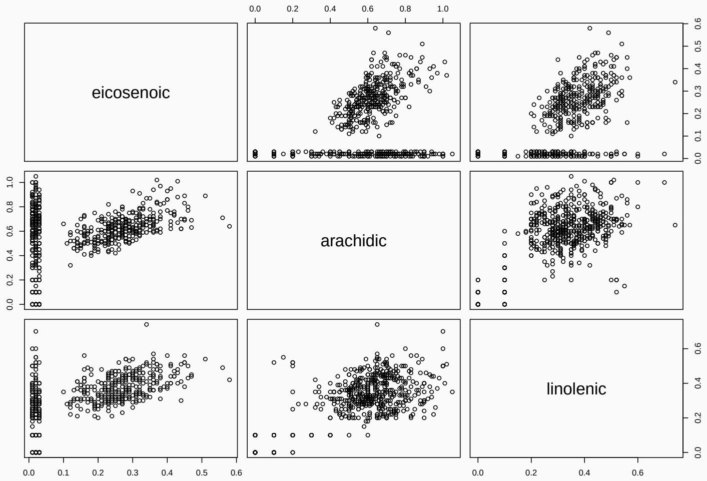
Bivariate density plot ii



Pairs plot i

```r
# Select three variables
olive_sub <- subset(olive,
                    select = c(eicosenoic, arachidic,
                               linolenic))

plot(olive_sub)
```

Pairs plot ii



Pairs plot iii

```r
# Or alternatively:
pairs(olive_sub)
```

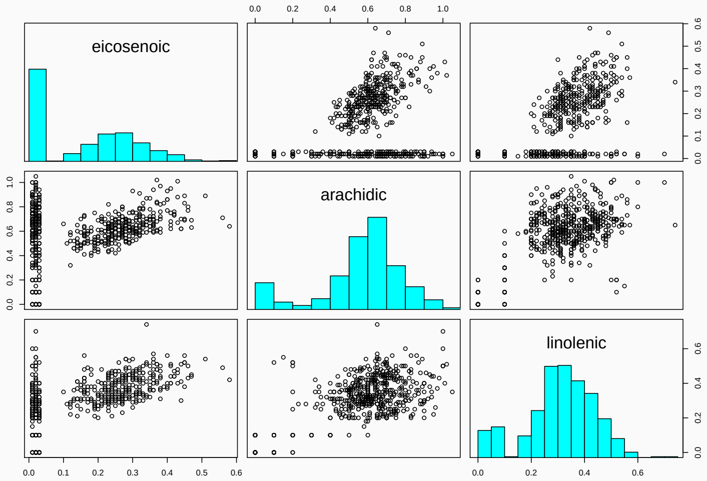
Pairs plot iv

```
# How to put histograms on the diagonal?
## From the help page for graphics::pairs
panel.hist <- function(x, ...) {
  usr <- par("usr"); on.exit(par(usr))
  par(usr = c(usr[1:2], 0, 1.5) )
  h <- hist(x, plot = FALSE)
  breaks <- h$breaks; nB <- length(breaks)
  y <- h$counts; y <- y/max(y)
  rect(breaks[-nB], 0, breaks[-1], y, col = "cyan",...)
}
```

Pairs plot v

```
pairs(olive_sub,
      diag.panel = panel.hist)
```

Pairs plot vi



Conclusion

- Base `R` is just as powerful as `ggplot2`.
    - But there is no consistent interface.
    - Adding components (e.g. data points on top of boxplots, histogram on pairs plots) is complex.
- For the record, I have seen high-quality, publication-ready graphs from both approaches.
    - My personal preference is `ggplot2`, because I find it easier to iterate quickly between graphs until I find the right one.this