Introduction to R

Basic Graphics

Jan-Philipp Kolb

03 März, 2020

FIRST THINGS TO DO

> Don’t try to kiss your data on the first date; rather, you just want to get to know the data:

1. Import the data
2. Review the codebook
3. Learn about the data
4. **Quick visual understanding of the data**

A plot says more than 1000 words

Statements on graphs in R

- Graphical data analysis is great
- Good plots can contribute to a better understanding
- Generating a plot is easy
- Making a good plot can take very long
- Generating plots with R is fun
- Plots created with R have high quality
- Almost every plot type is supported by R
- A large number of export formats are available in R

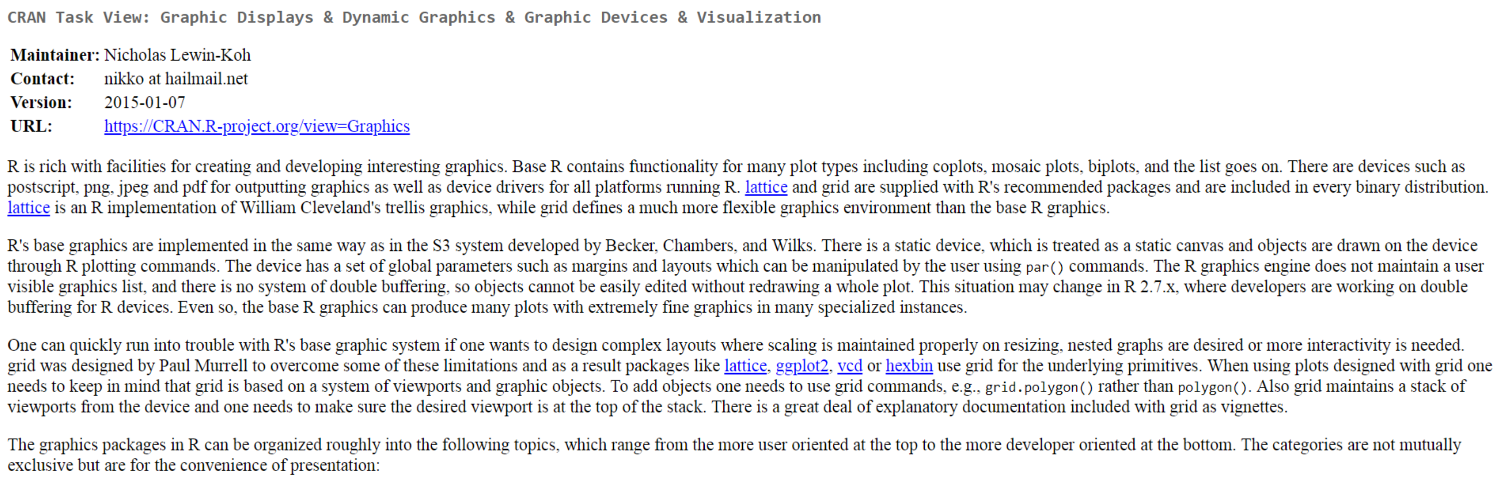
Not all plots are the same

- The base package already includes a large number of plot functions
- Other packages like `lattice`, `ggplot2`, etc extend this functionality

Manuals that go far beyond this introduction:

- Murrell, P (2006): R Graphics.
- R Development Core Group **Graphics with R**
- Wiki on **R Programming/Graphics**
- Martin Meermeyer **Creating Reproducible Publication Quality Graphics with R: A Tutorial**
- Institute For Quantitative Social Science at Harvard - **R graphics tutorial**

Task View for graphics


CRAN Task View: Graphic Displays & Dynamic Graphics & Graphic Devices & Visualization

Maintainer:	Nicholas Lewin-Koh
Contact:	nikko at hailmail.net
Version:	2015-01-07
URL:	https://CRAN.R-project.org/view=Graphics

R is rich with facilities for creating and developing interesting graphics. Base R contains functionality for many plot types including coplots, mosaic plots, biplots, and the list goes on. There are devices such as postscript, png, jpeg and pdf for outputting graphics as well as device drivers for all platforms running R. lattice and grid are supplied with R's recommended packages and are included in every binary distribution. lattice is an R implementation of William Cleveland's trellis graphics, while grid defines a much more flexible graphics environment than the base R graphics.

R's base graphics are implemented in the same way as in the S3 system developed by Becker, Chambers, and Wilks. There is a static device, which is treated as a static canvas and objects are drawn on the device through R plotting commands. The device has a set of global parameters such as margins and layouts which can be manipulated by the user using `par()` commands. The R graphics engine does not maintain a user visible graphics list, and there is no system of double buffering, so objects cannot be easily edited without redrawing a whole plot. This situation may change in R 2.7.x, where developers are working on double buffering for R devices. Even so, the base R graphics can produce many plots with extremely fine graphics in many specialized instances.

One can quickly run into trouble with R's base graphic system if one wants to design complex layouts where scaling is maintained properly on resizing, nested graphs are desired or more interactivity is needed. grid was designed by Paul Murrell to overcome some of these limitations and as a result packages like lattice, ggplot2, vcd or hexbin use grid for the underlying primitives. When using plots designed with grid one needs to keep in mind that grid is based on a system of viewports and graphic objects. To add objects one needs to use grid commands, e.g., `grid.polygon()` rather than `polygon()`. Also grid maintains a stack of viewports from the device and one needs to make sure the desired viewport is at the top of the stack. There is a great deal of explanatory documentation included with grid as vignettes.

The graphics packages in R can be organized roughly into the following topics, which range from the more user oriented at the top to the more developer oriented at the bottom. The categories are not mutually exclusive but are for the convenience of presentation:

https://cran.r-project.org/web/views/Graphics.html

The example dataset

```
install.packages("AmesHousing")
```

```
ames_df <- AmesHousing::make_ames()
```

Variables used in this section

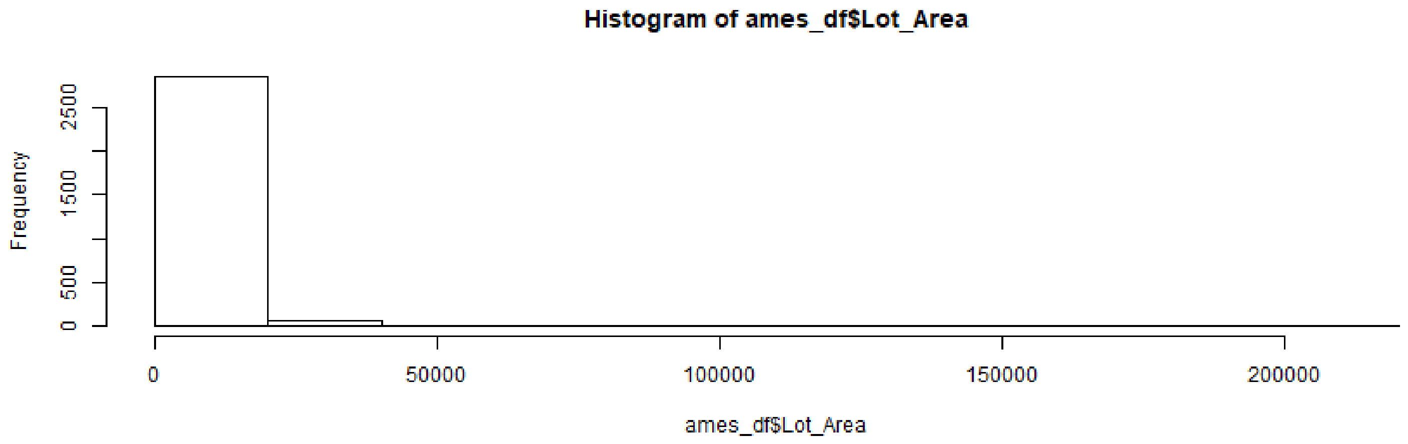
- `Lot_Area`: Lot size in square feet
- `Alley`: Type of alley access to property
- `Street`: Type of road access to property

Histogram - The `hist()` function

We create a histogram of the variable `duration`:

```
?hist
```

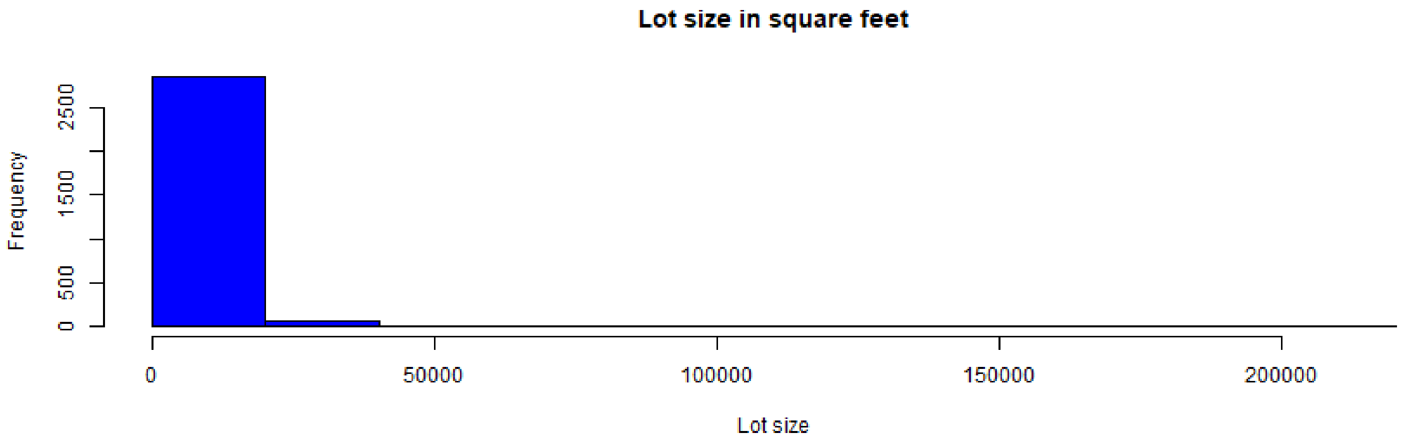
```
hist(ames_df$Lot_Area)
```



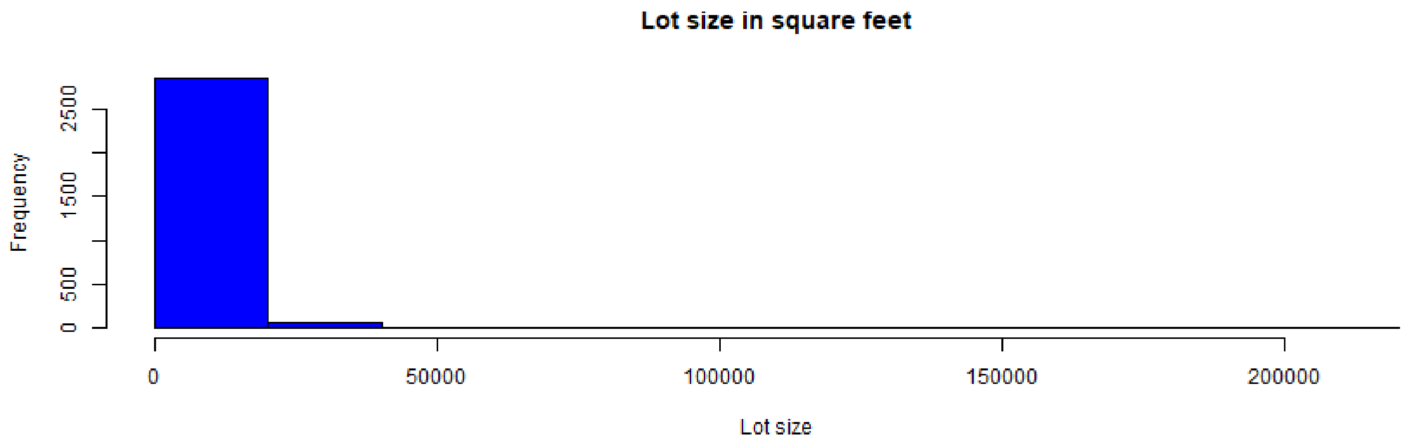
Histogram

- Command `hist()` plots a histogram
- At least one observation vector must be passed to the function
- `hist()` has many more arguments, which all have (meaningful) default values

```
hist(ames_df$Lot_Area,col="blue",
     main="Lot size in square feet",ylab="Frequency",
     xlab="Lot size")
```

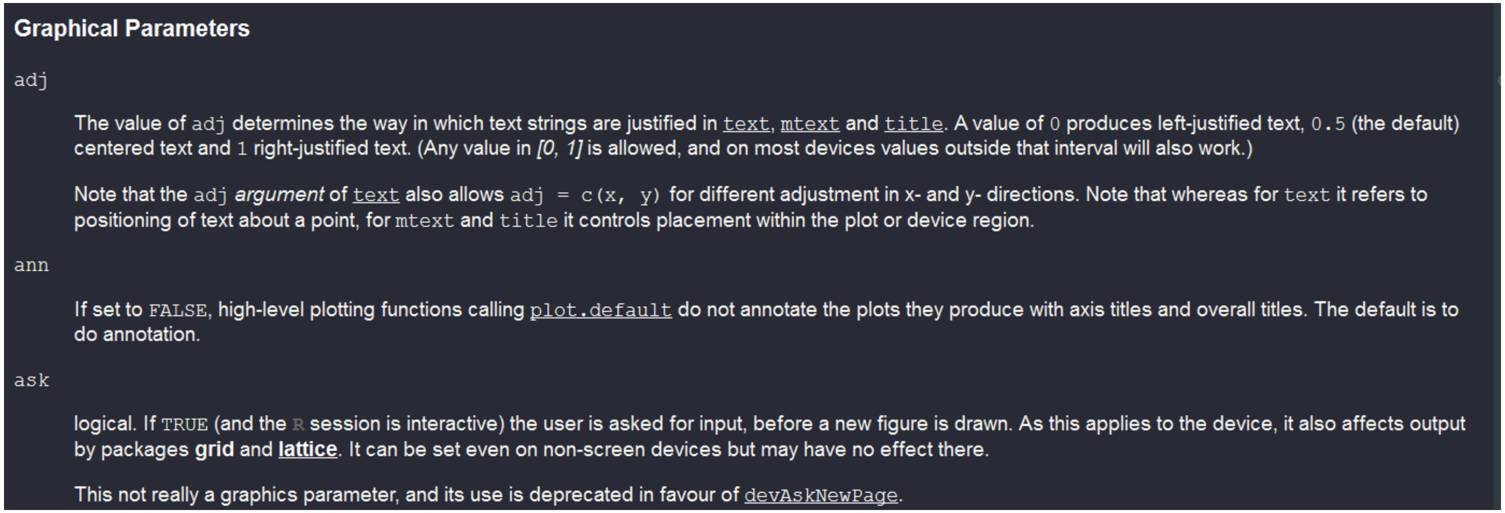


Histogram


Lot size in square feet
Frequency
0
500
1500
2500
0
50000
100000
150000
200000
Lot size

Further arguments:

```
?plot
# or
?par
```



Graphical Parameters

`adj`

The value of `adj` determines the way in which text strings are justified in `text`, `mtext` and `title`. A value of `0` produces left-justified text, `0.5` (the default) centered text and `1` right-justified text. (Any value in *[0, 1]* is allowed, and on most devices values outside that interval will also work.)

Note that the `adj` *argument* of `text` also allows `adj = c(x, y)` for different adjustment in x- and y- directions. Note that whereas for `text` it refers to positioning of text about a point, for `mtext` and `title` it controls placement within the plot or device region.

`ann`

If set to `FALSE`, high-level plotting functions calling `plot.default` do not annotate the plots they produce with axis titles and overall titles. The default is to do annotation.

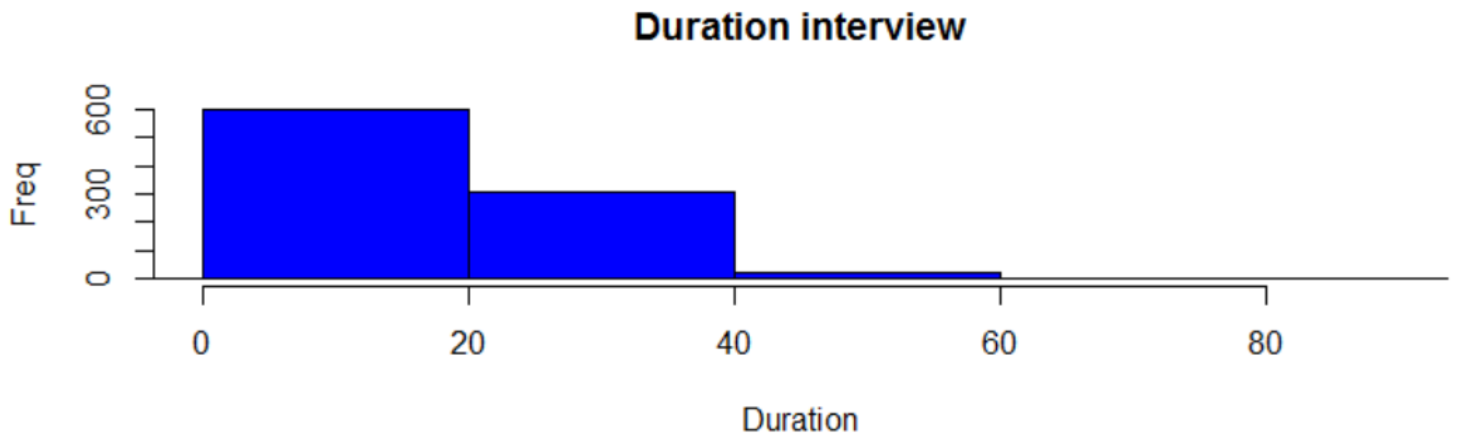
`ask`

logical. If `TRUE` (and the R session is interactive) the user is asked for input, before a new figure is drawn. As this applies to the device, it also affects output by packages **grid** and **lattice**. It can be set even on non-screen devices but may have no effect there.

This not really a graphics parameter, and its use is deprecated in favour of `devAskNewPage`.

The xlim argument

```
hist(ames_df$Lot_Area,col="blue",
     main="Lot size",ylab="Freq", xlab="Lot size",
     xlim=c(0,90))
```

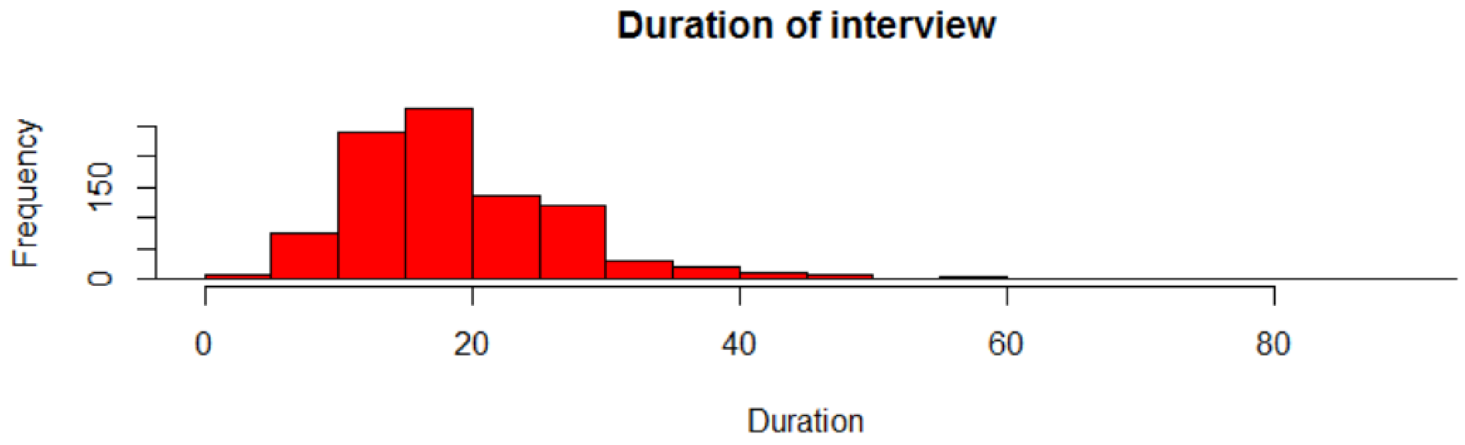


The `breaks` argument

- While the previous arguments are valid for many graphics functions, the following apply mainly to histogrames:

```
hist(ames_df$Lot_Area,col="red",
     main="Duration of interview", xlab="Duration",
     xlim=c(0,90),breaks=60)
```

- with `breaks` you can control the number of bars...

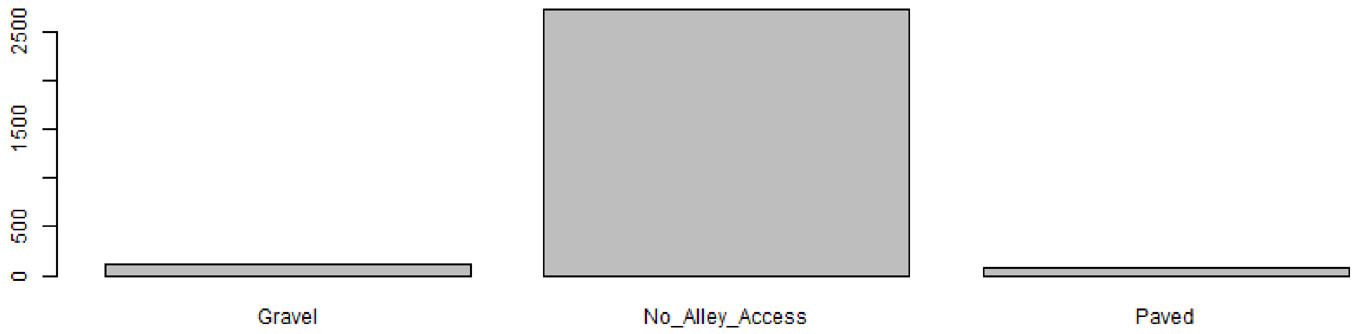


Tabulate and `barplot`

- The command `barplot()` generates a barplot from a frequency table
- We get the frequency table with the following command:

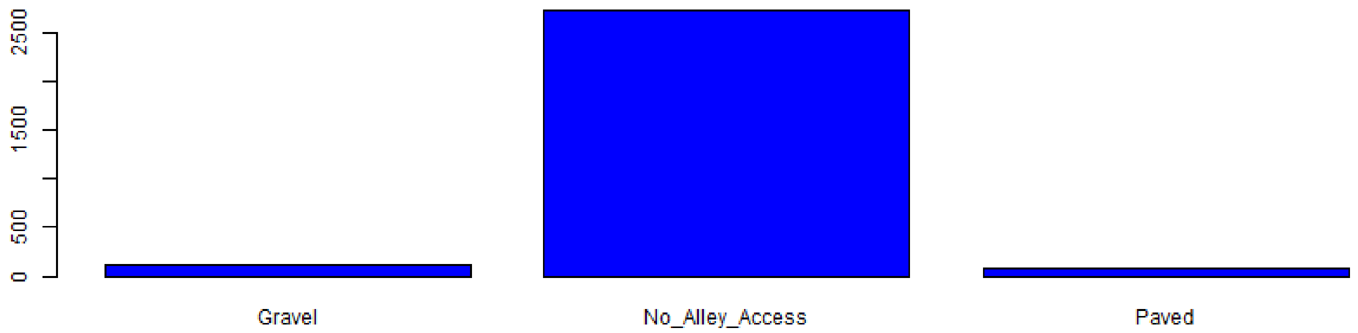
```
tab_alley <- table(ames_df$Alley)
```

```
barplot(tab_alley)
```



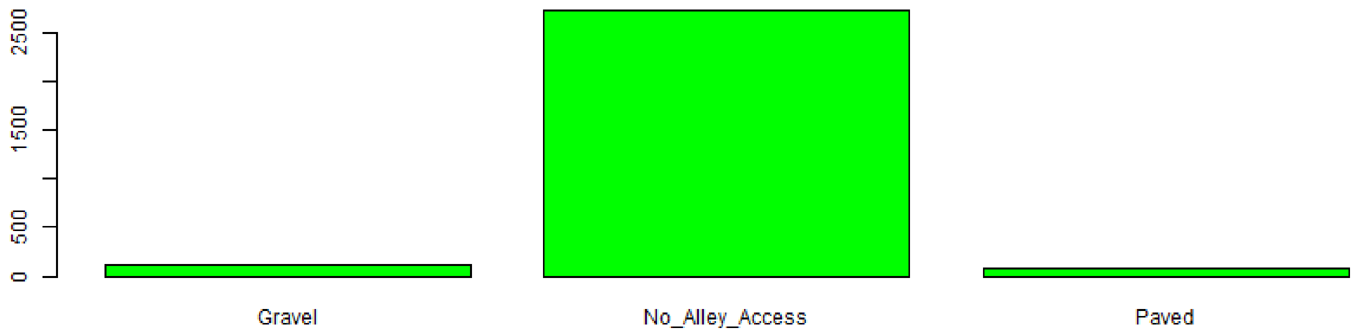
More colour:

```
barplot(tab_alley,col=rgb(0,0,1))
```



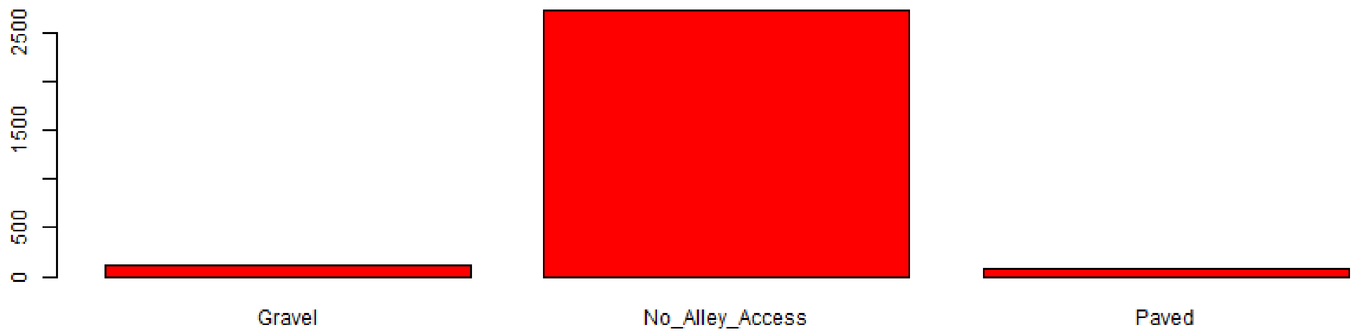
Green colour

```
barplot(tab_alley,col=rgb(0,1,0))
```



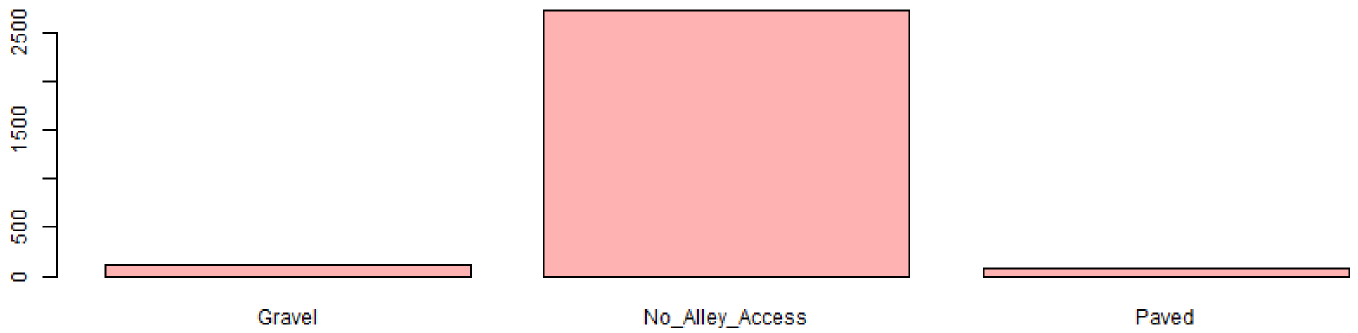
Red colour

```
barplot(tab_alley,col=rgb(1,0,0))
```



Transparent

```
barplot(tab_alley,col=rgb(1,0,0,.3))
```

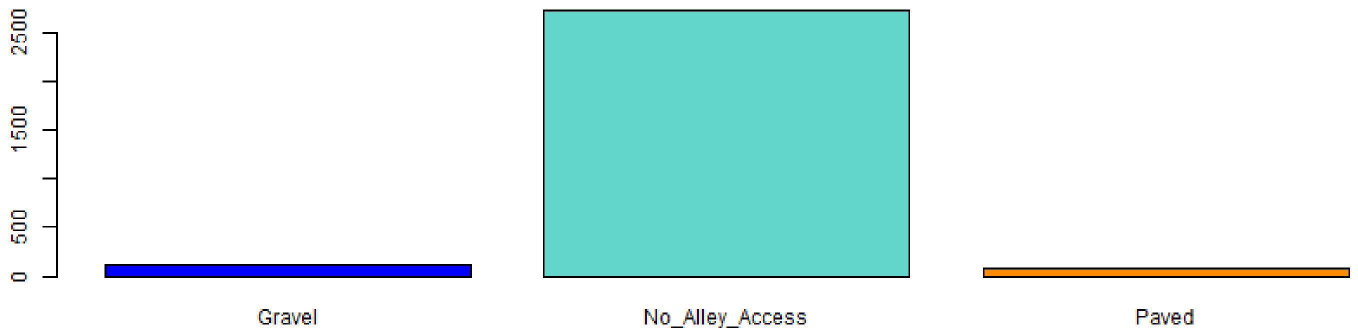


Rstudio addin `colourpicker`

```
install.packages("colourpicker")
```

Set various colors

```r
barplot(tab_alley,col=c(20,"#62D6C8", "darkorange"))
```



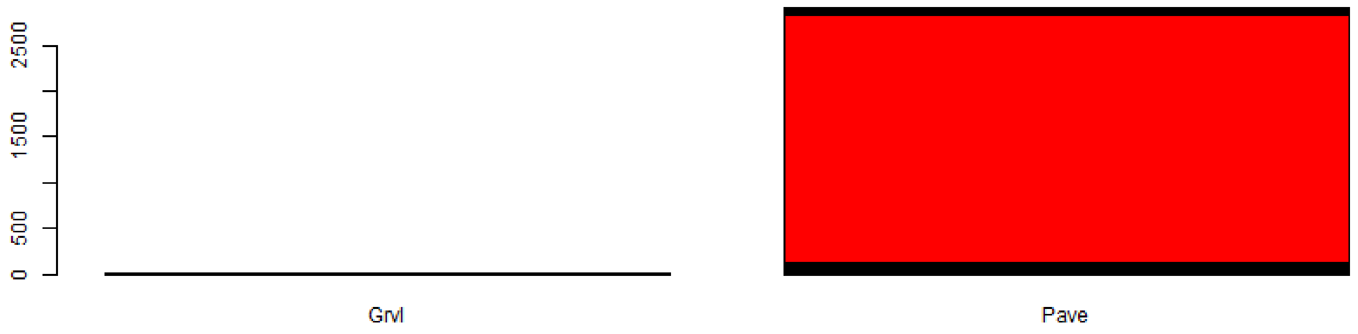
A two dimensional table

```
tab2dim <- table(ames_df$Alley,ames_df$Street)
```

- If the passed table object is two-dimensional, a conditional barplot is created

Conditional `barplot`

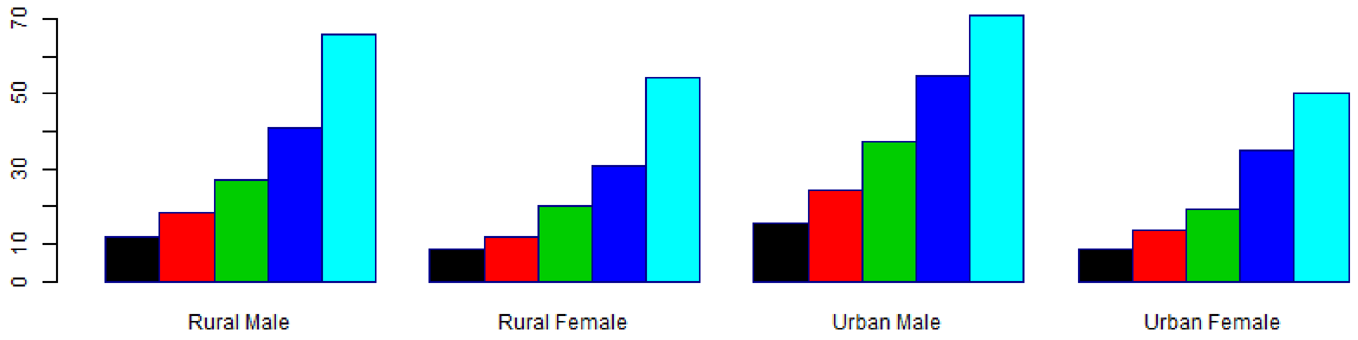
```
barplot(tab2dim,col=1:2)
```



```
barplot(tab2dim,col=3:4,beside=T)
```

Exercise: simple graphics

- Load the dataset `VADeaths` and create the following plot:

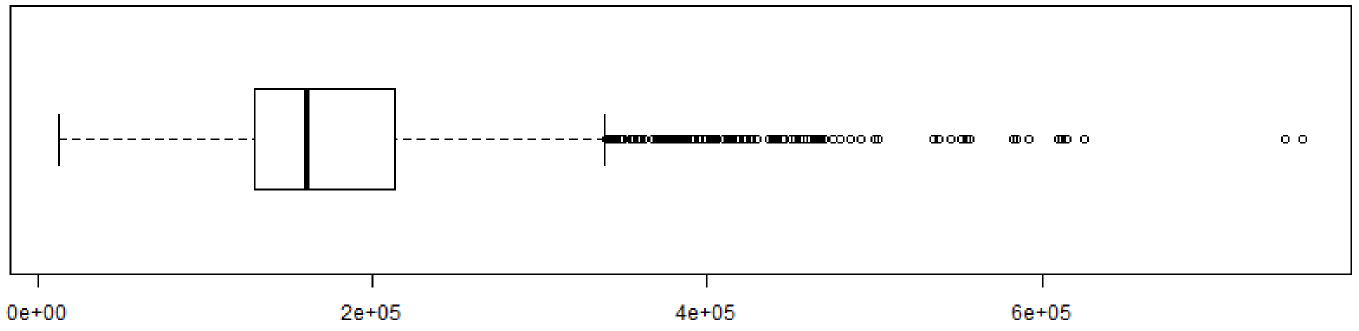


Horizontal `boxplot`

- A simple **boxplot** can be created with `boxplot()`
- For the command `boxplot()` at least one observation vector must be passed

```
?boxplot
```

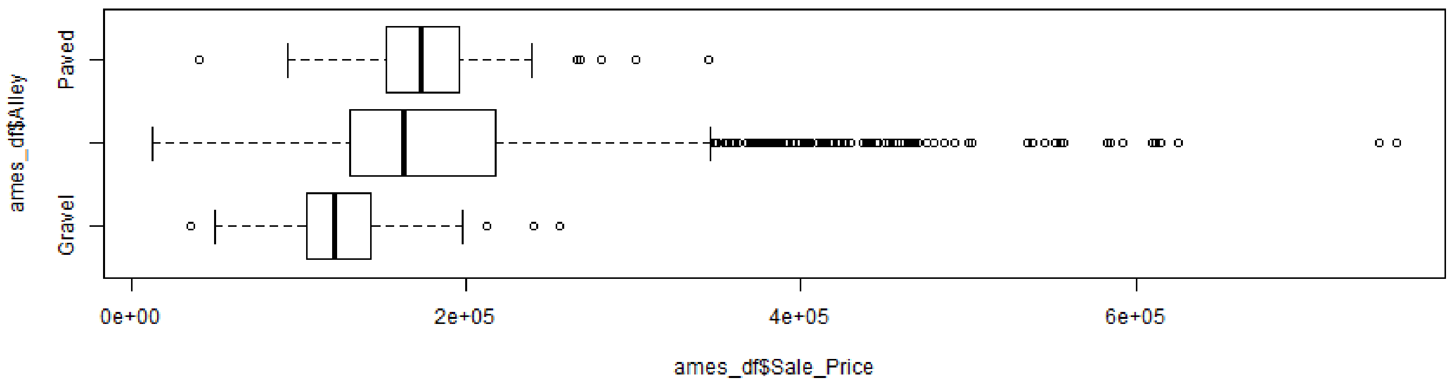
```
boxplot(ames_df$Sale_Price,horizontal=TRUE)
```



Grouped boxplots

- A very simple way to get a first impression of conditional distributions is via so-called grouped notched boxplots
- To do this, a so-called formula object must be passed to the `boxplot()` function.
- The conditional variable is located on the right side of a tilde

```
boxplot(ames_df$Sale_Price~ames_df$Alley,horizontal=TRUE)
```



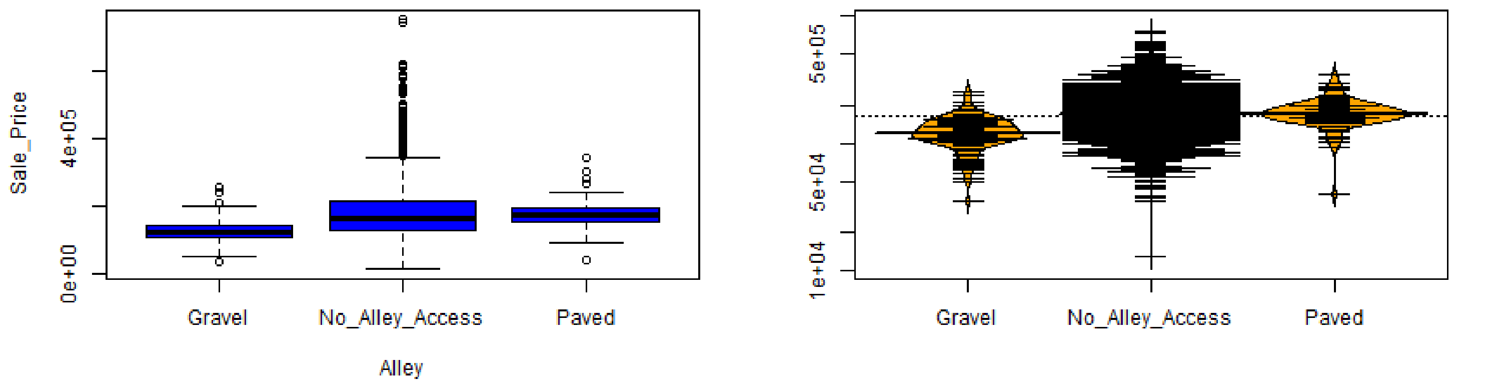
Boxplot alternatives - `vioplot`

- Builds on Boxplot - additional information about data density
- Density is calculated using the kernel method.
- The further the expansion, the higher the density at this point.
- White dot - median

```
library(vioplot)
vioplot(na.omit(ames_df$Sale_Price),col="royalblue")
```

Alternatives `boxplot()`

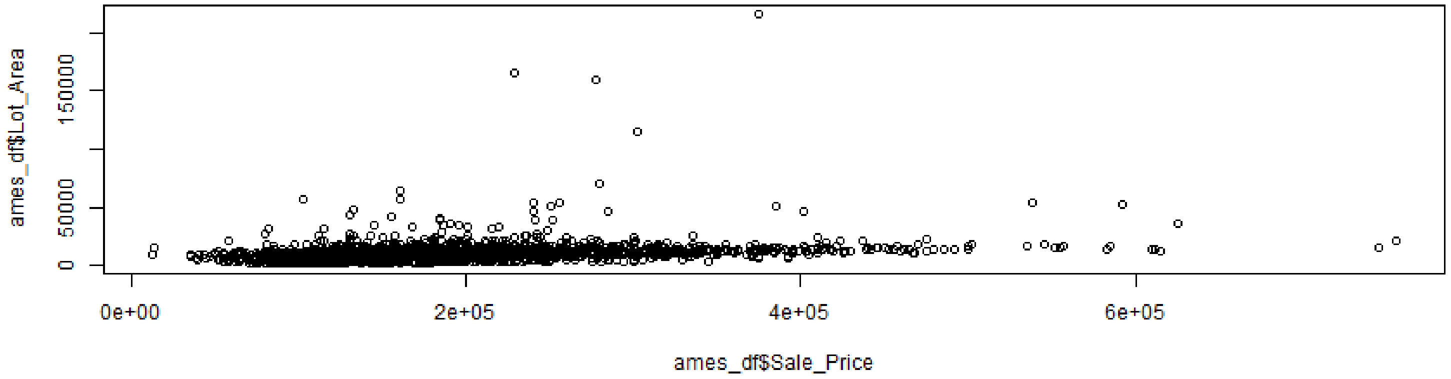
```
library(beanplot)
par(mfrow = c(1,2))
boxplot(Sale_Price~Alley,data=ames_df,col="blue")
beanplot(Sale_Price~Alley,data=ames_df,col="orange")
```



Conditional, bi- and multivariate distribution graphics - scatterplots

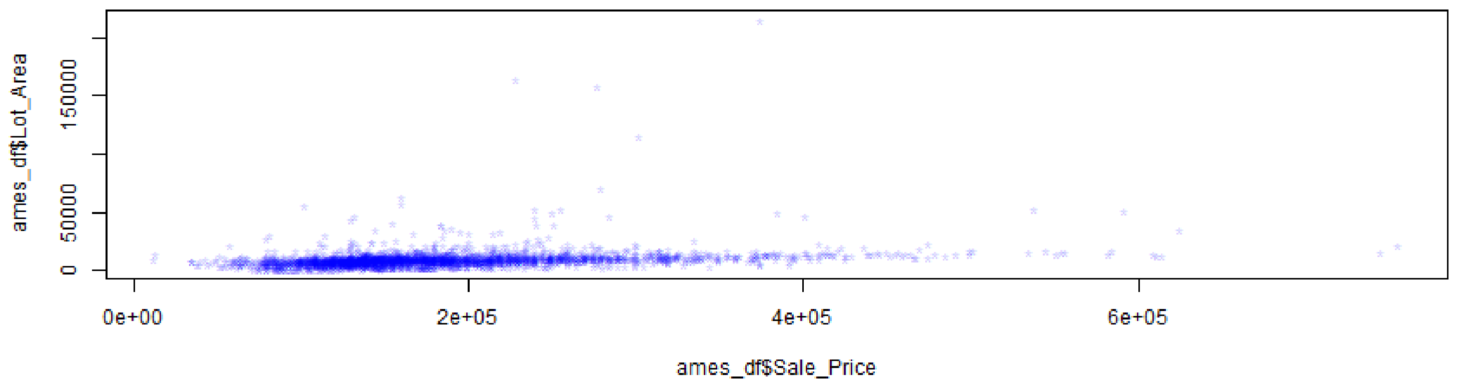
- A simple two-way scatterplot can be created with the `plot()` function
- To create a scatterplot x and y observation vector must be passed
- Argument `col` to specify the color (color as character or numeric)
- Argument `pch` to specify plot symbols (plotting character) (character or numeric)
- The labels are defined with `xlab` and `ylab`.

```
plot(ames_df$Sale_Price,ames_df$Lot_Area)
```



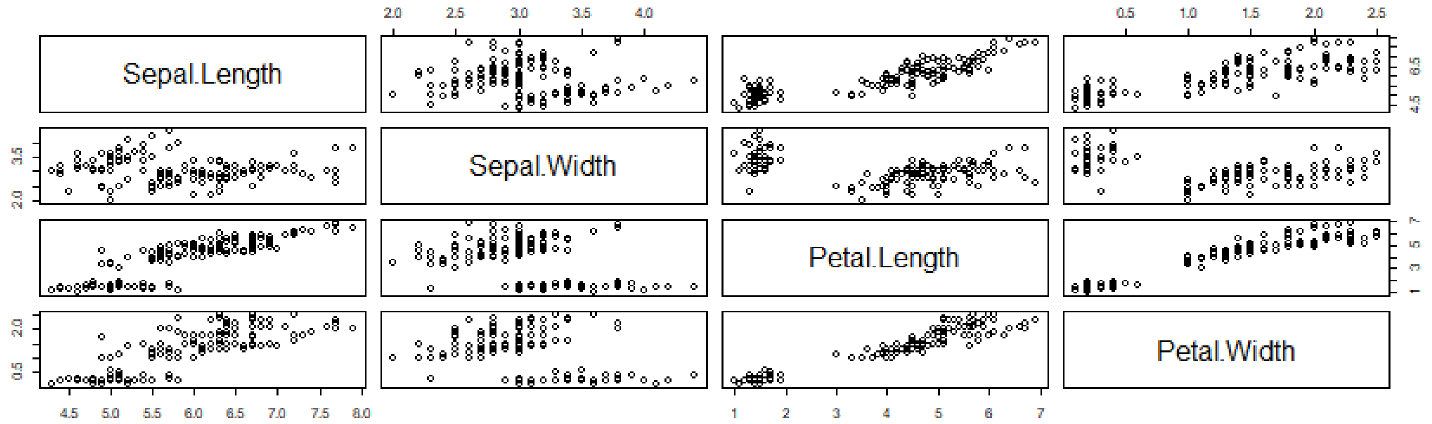
Changing the ploting character

```
plot(ames_df$Sale_Price,ames_df$Lot_Area,pch="*",col=rgb(0,0,1,.2))
```



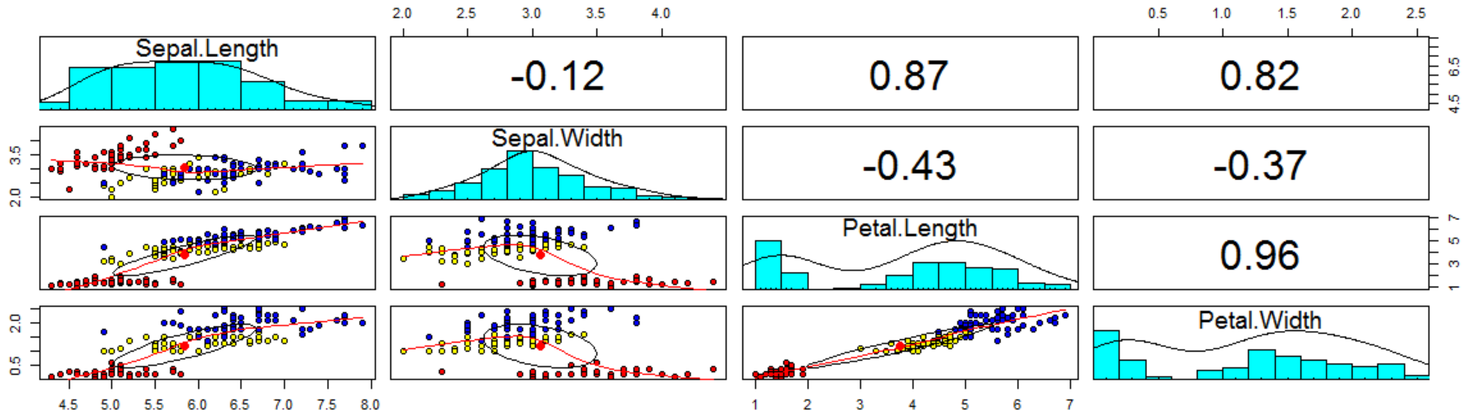
Relationship between variables - `pairs` plot

```
pairs(iris[,1:4])
```



Enhanced multivariate plots

```
library("psych")
pairs.panels(iris[,1:4],
             bg=c("red","yellow","blue")[iris$Species],
             pch=21,main="")
```



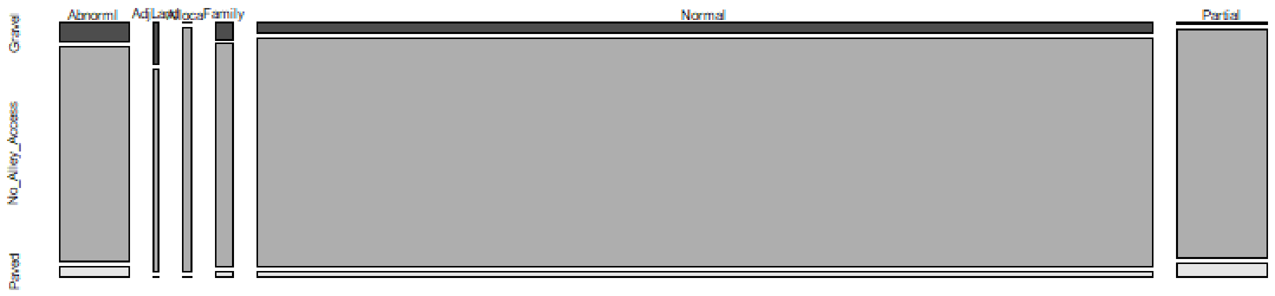
Tabulating

```
(tab2 <- table(ames_df$Sale_Condition,ames_df$Alley))
```

```
## 
##           Gravel No_Alley_Access Paved
##   Abnorml     14             167     9
##   AdjLand      2              10     0
##   Alloca       0              24     0
##   Family       3              42     1
##   Normal     100            2259    54
##   Partial      1             230    14
```

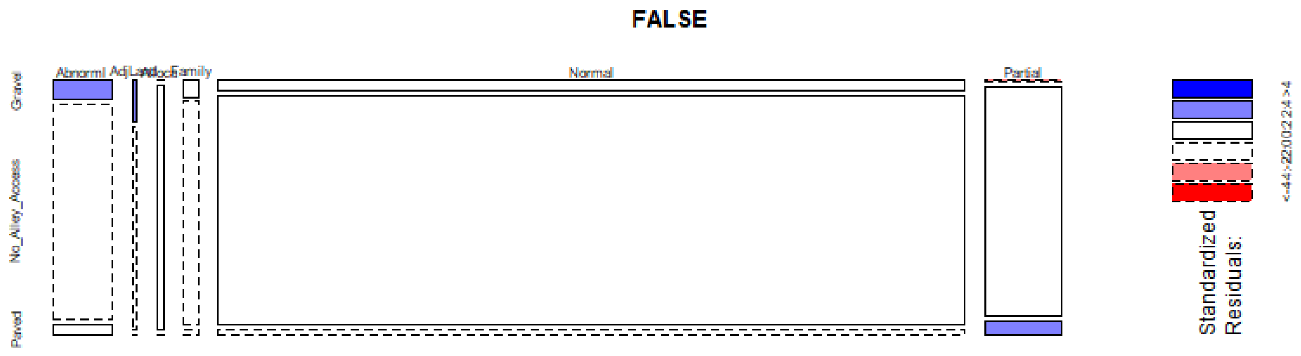
Relationship - categorial variables

```r
mosaicplot(tab2, color = TRUE,main="")
```



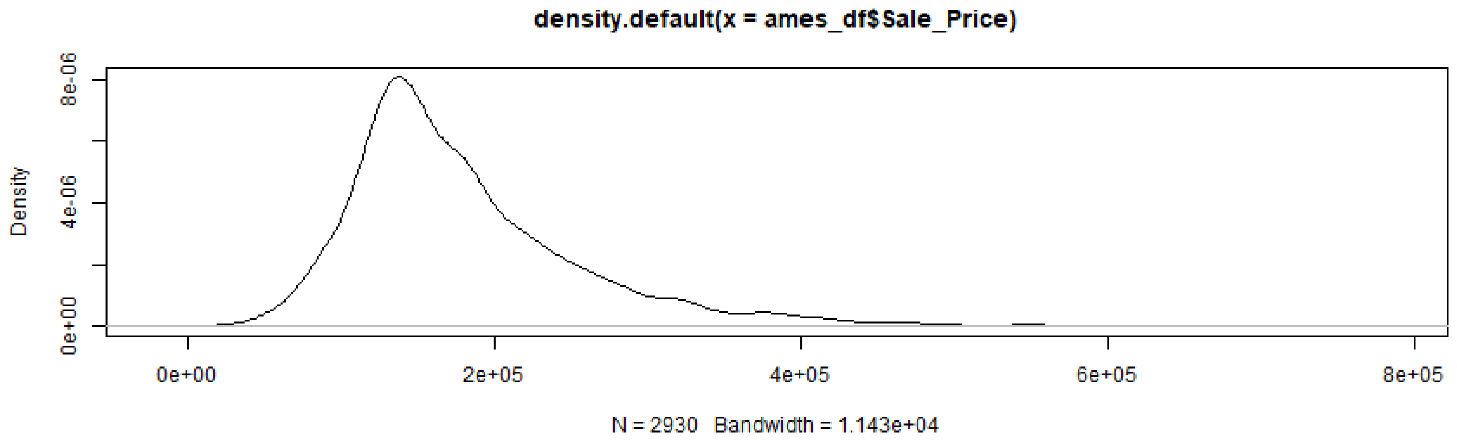
Surfaces are shaded according to the residuals:

```r
mosaicplot(tab2, main=F,shade = TRUE)
```



Density plot

```
plot(density(ames_df$Sale_Price))
```

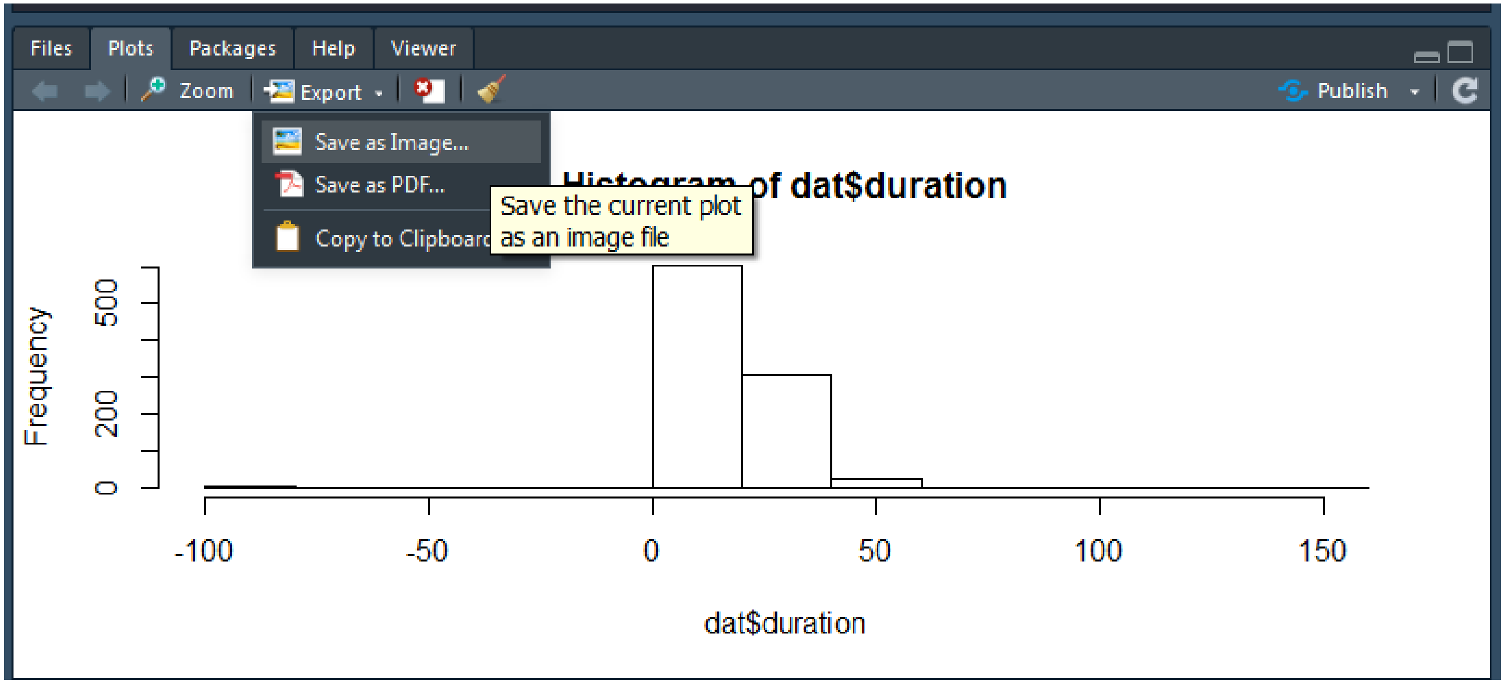


Stem and leaf plot

```
stem(airquality$Ozone)
```

```
##
##   The decimal point is 1 digit(s) to the right of the |
##
##    0 | 1467778999
##    1 | 011122333344446666688889
##    2 | 000011112333333344478889
##    3 | 001222455667799
##    4 | 01444556789
##    5 | 0299
##    6 | 134456
##    7 | 13367889
##    8 | 024559
##    9 | 1677
##   10 | 8
##   11 | 058
##   12 | 2
##   13 | 5
##   14 |
##   15 |
##   16 | 8
```

Export with Rstudio



Command to save graphic

- Alternatively also with the commands `png`, `pdf` or `jpeg` for example

```
png("Histogramm.png")
  hist(dat$duration)
dev.off()
```

```
pdf("Histogramm.pdf")
  hist(dat$duration)
dev.off()
```

```
jpeg("Histogramm.jpeg")
  hist(dat$duration)
dev.off()
```

Exercise: Advanced Base Graphics

Create Scatterplot with cars dataset

a) Load the cars dataset and create a scatterplot of the data.

b) Using the argument lab of the function plot create a new scatterplot where the thickmarks of the x and y axis specify every integer.

Adjust Scatterplot

The previous plot didn’t showed all the numbers associated to the new thickmarks, so we are going to fix them. Recreate the same plot from the previous question and using the argument `cex.axis` control the size of the numbers associated to the axes thickmarks so they can be small enough to be visible.

Change orientation

On the previous plot the numbers associated to the y-axis thickmarks aren’t easy to read. Recreate the plot from the last exercise and use the argument

Exercise: Adding things to scatterplot

The `points` function

Suppose you want to add two new observations to the previous plot, but you want to identify them on the graph. Using the points function add the new observations to the last plot using red to identify them. The values of the new observation are speed = 23, 26 and dist = 60, 61.

Adjust x-axis range

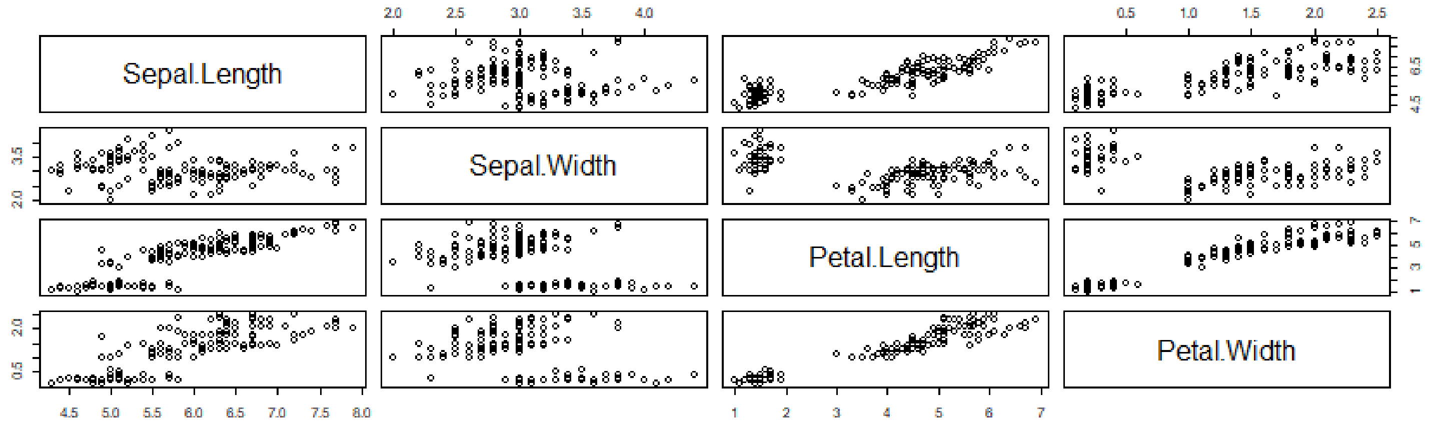
As you could see the previous plot doesn't show one of the new observations because is out the x-axis range.

a) Create again the plot for the old observations with an x-axis range that includes all the values from 4 to 26.

b) Add the two new observations using the points function.

Relationship between variables - `pairs` plot

```
pairs(iris[,1:4])
```



Enhanced multivariate plots

```
install.packages("psych")
```

```
library("psych")
bgcol <- c("red","yellow","blue")[iris$Species]
pairs.panels(iris[,1:4],bg=bgcol,pch=21,main="")
```

levelplot

- `education` in years

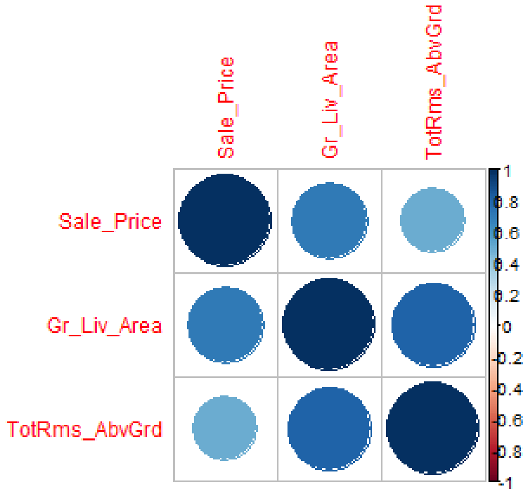
```r
library("lattice")
levelplot(table(ames_df$MS_Zoning,ames_df$Alley),
          xlab="education",ylab="job")
```

Color palettes

```
library(RColorBrewer)
display.brewer.all()
```

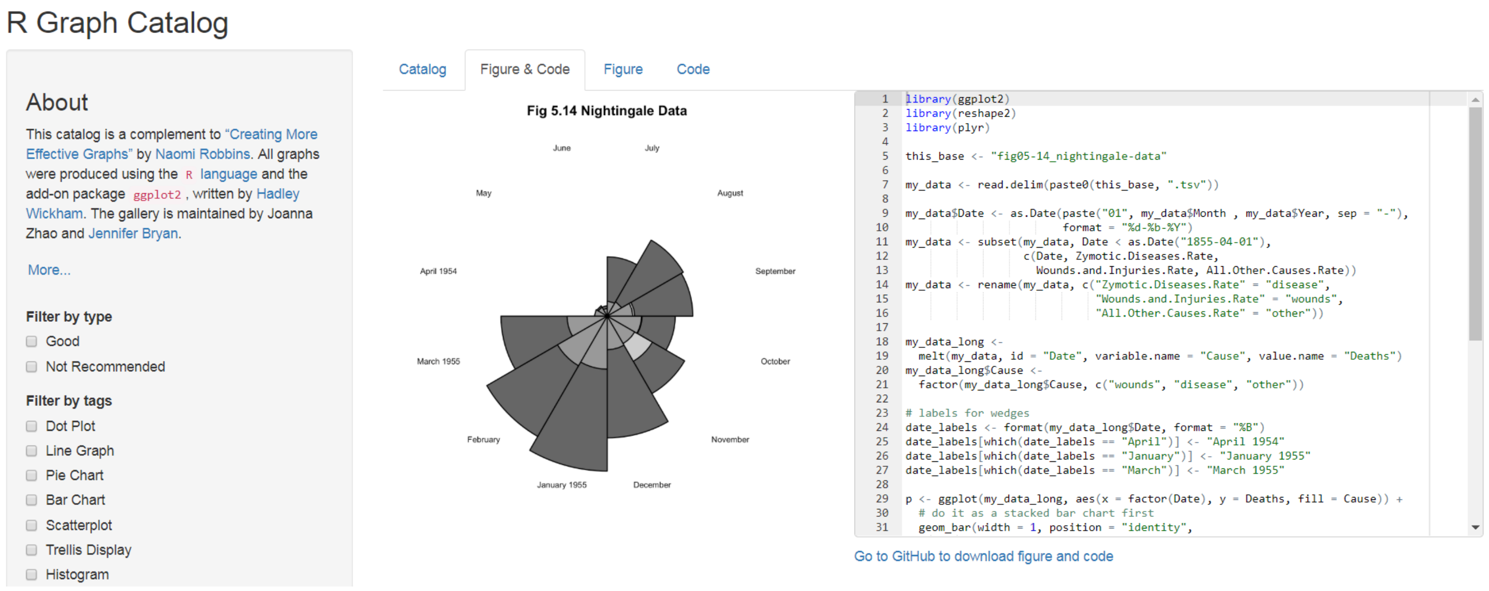
A correlation plot

```
library(corrplot)
corrplot(cor(ames_df[,c("Sale_Price",
              "Gr_Liv_Area","TotRms_AbvGrd")]))
```



Shiny App - R graphs catalogue

R graphs catalogue



About

This catalog is a complement to "Creating More Effective Graphs" by Naomi Robbins. All graphs were produced using the `R` language and the add-on package `ggplot2`, written by Hadley Wickham. The gallery is maintained by Joanna Zhao and Jennifer Bryan.

More...

Filter by type

- Good
- Not Recommended

Filter by tags

- Dot Plot
- Line Graph
- Pie Chart
- Bar Chart
- Scatterplot
- Trellis Display
- Histogram

```r
library(ggplot2)
library(reshape2)
library(plyr)

this_base <- "fig05-14_nightingale-data"

my_data <- read.delim(paste0(this_base, ".tsv"))

my_data$Date <- as.Date(paste("01", my_data$Month , my_data$Year, sep = "-"),
                        format = "%d-%b-%Y")
my_data <- subset(my_data, Date < as.Date("1855-04-01"),
                  c(Date, Zymotic.Diseases.Rate,
                    Wounds.and.Injuries.Rate, All.Other.Causes.Rate))
my_data <- rename(my_data, c("Zymotic.Diseases.Rate" = "disease",
                             "Wounds.and.Injuries.Rate" = "wounds",
                             "All.Other.Causes.Rate" = "other"))

my_data_long <-
  melt(my_data, id = "Date", variable.name = "Cause", value.name = "Deaths")
my_data_long$Cause <-
  factor(my_data_long$Cause, c("wounds", "disease", "other"))

# labels for wedges
date_labels <- format(my_data_long$Date, format = "%B")
date_labels[which(date_labels == "April")] <- "April 1954"
date_labels[which(date_labels == "January")] <- "January 1955"
date_labels[which(date_labels == "March")] <- "March 1955"

p <- ggplot(my_data_long, aes(x = factor(Date), y = Deaths, fill = Cause)) +
  # do it as a stacked bar chart first
  geom_bar(width = 1, position = "identity",
```

Go to GitHub to download figure and code

http://shinyapps.stat.ubc.ca/r-graph-catalog/

Links

- **Top 50 ggplot2 Visualizations**
- **Bioconductor R manual** with an extensive part on graphics
- **Shiny app** for **interactive plot editing**