R alapok
Vizualizáció

Lukovszki Csaba

Tartalomjegyzék

1. Adatok megjelenítése

Az adatok megjelenítése két célt szolgál. Egyrészt lehetőséget teremt az adatok vizuális áttekintésére, melyen keresztül az adatok jellege, az adatok közötti kapcsolatok jobban megérthetők, áttekinthetők, másrészt nagyon fontos szerepe van az adatelemzési feladatok megoldásainak prezentálásában.

Mind a két szempont erősen hangsúlyozza, milyen fontos az adatok megjelenítési lehetőségeinek alapos megismerése és készségszintű elsajátítása.

A következőkben áttekintjük a `base` csomag legfontosabb megjelenítési lehetőségeit és a gyakorlati szempontból legfontosabb módszereket, megoldásokat.

1.1. Ponthalmaz

Először ismerjük meg a megjelenítés bemutatása során használt adathalmazunkat!

Az `mpg` adathalmaz a `ggplot2` csomagban található, mely 38 autó modell adatait tartalmazz az 1999 és 2008 évekből. Az adatok áttekintését legegyszerűbben a következők szerint tehetjük meg.

```
library(ggplot2)
# másoljuk át az adatokat, hogy
data <- mpg
# tekintsük át az adatkeret struktúráját
str(data)
```

```
## tibble [234 x 11] (S3: tbl_df/tbl/data.frame)
##  $ manufacturer: chr [1:234] "audi" "audi" "audi" "audi" ...
##  $ model       : chr [1:234] "a4" "a4" "a4" "a4" ...
##  $ displ       : num [1:234] 1.8 1.8 2 2 2.8 2.8 3.1 1.8 1.8 2 ...
##  $ year        : int [1:234] 1999 1999 2008 2008 1999 1999 2008 1999 1999 2008 ...
```

```
##  $ cyl          : int [1:234] 4 4 4 4 6 6 6 4 4 4 ...
##  $ trans        : chr [1:234] "auto(l5)" "manual(m5)" "manual(m6)" "auto(av)" ...
##  $ drv          : chr [1:234] "f" "f" "f" "f" ...
##  $ cty          : int [1:234] 18 21 20 21 16 18 18 18 16 20 ...
##  $ hwy          : int [1:234] 29 29 31 30 26 26 27 26 25 28 ...
##  $ fl           : chr [1:234] "p" "p" "p" "p" ...
##  $ class        : chr [1:234] "compact" "compact" "compact" "compact" ...
```

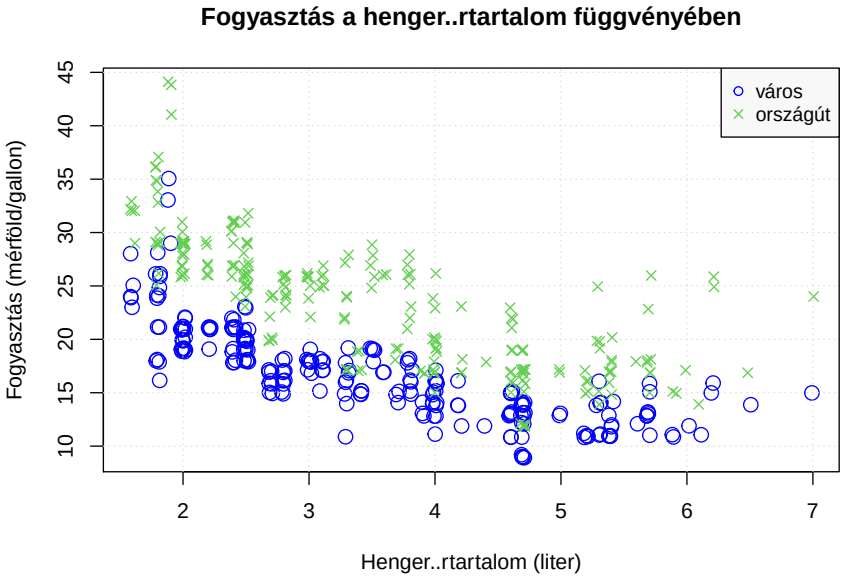
```
# elérhető részletes segítség is
?mpg
```

Általános esetben az adatok megismerését követően az egyes adatokat konvertáljuk az elvárt típusra, a hiányzó és hibás adatokat kezeljük. Ettől most eltekintünk, a megjelenítés során alkalomszerűen végezzük el.

A ponthalmazok legalapvetőbb megjelenítése két dimenzióban `x-y` értékpárok alapján történik, mely során Descartes-féle derékszögű koordináta rendszerben ábrázoljuk az egyes pontok az `x` és `y` koordináta párok alapján. Erre a legegyszerűbb lehetőséget a `plot()` függvény nyújtja.

Először ábrázoljuk az egyes modellek fogyasztását városban `cty` és országúton `hwy` a hengerűr-tartalom függvényében `displ`.

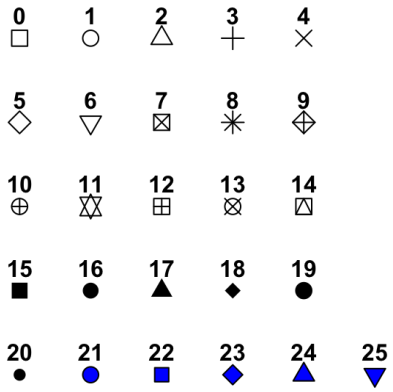
```
# Ponthalmaz kirajzolása (x,y) koordináta rendszerben
plot(x = jitter(data$displ), y = jitter(data$cty),
     col ="blue", pch = 1, cex = 1.5,
     main = "Fogyasztás a hengerűrtartalom függvényében",
     xlab = "Hengerűrtartalom (liter)",
     ylab = "Fogyasztás (mérföld/gallon)",
     ylim = range(c(data$cty, data$hwy))
     )
# Az ábra kiegészítése pontok újabb halmazával
points(jitter(data$displ), jitter(data$hwy), col = 3, pch = 4)
# Méretező rács kirajzolása
grid()
# Jelmagyarázat
legend("topright", legend = c("város", "országút"),
       pch = c(1, 4), cex = .9, col = c("blue", 3), bg = "#F8F8F8")
```



A `plot()` függvény alavetően `x` és `y` paraméterekben adott vektorokban tárolt értékpárokat ábrázol a koordináta rendszerben. Minden érték, mint pont kerül ábrázolásra.

A pontok színét a `col` (color) paraméterrel adhatjuk meg. Mint a legtöbb paraméter a szín paraméter is lehet vektor, ahol érvényesül az általánosan elvárt reprilkálási szabály. A szín megadható numerikusan az előre definiált színek indexeként, karakter vektorban szövegesen (`"blue"`, `"red"`, `"green"`, `"magenta"`, etc. . .), vagy közvetlen `RGB` (red-green-blue) színkóddal hexadecimálisan a `#` kezdő karakterrel (`"#F0E812"`), vagy függvények használatával, például az `rgb(r,g,b,o)` (red, green, blue, opacity 0-1 értéktarományon). Ezen felül számos egyéb függvény ad lehetőséget egymástól jól elkülöníthető színek generálására is, mint a `rainbow()`, vagy a `terrain.colors()`.

A pontok helyén ábrázolandó szimbólumokat a `pch` (plotting characters) paraméterrel adhatjuk meg. Itt is, mint minden más esetben numerikus vektorokat is megadhatunk, mely újra felhasználásra kerül. Az egyes használt szimbólumokat a következő táblázat mutatja be. A numerikus szimbólumok mellett használhatunk karaktereket is, például `"*"`, vagy `"o"`.



1. ábra. Szimbólumok

Az ábrázolt szimból nagyságát állíthatjuk a `cex` (character expansion ratio) paraméter segítségével, mely alapértéke 1.

Az ábrát, valamint az egyes tengelyeket feliratozhatjuk a `main` és `xlab`, `ylab` (x, y label) paraméterek használatával.

Pontok ábrázolásakor gyakorta előfordulhat, hogy több szimbólum kerül ábrázolásra egy adott pontban, ennek eredményeképpen elveszik az információ. Ekkor érdemes egy kis zajt adni a koordináta értékekhez, melyet a `jitter()` függvény alkalmazásával tehetünk meg. A hozzáadott véletlenszerű érték az értéktartomány alapján kerül megállapításra úgy, hogy a pontok lehetőleg ne lapolódjanak át, de ne vesszen el az információtartalom. Amennyiben a hozzáadott értéket magunk szeretnénk koordinálni használjuk a függvény `amount` paraméterét.

Amennyiben egy ábrázolt ponthalmaztt egyéb pontokkal szeretnénk kiegészíteni, használjuk a `points()` függvényt, melyet a `plot()` függvényhez hasonlóan használhatunk. Mivel a két koordináta tengely ábrázolási értékhatárát a `plot()` függvény határozza meg, ezért bármilyen újabb adatok felvitele során az ábrázolási értékhatárokat a `plot()` függvény `xlim` és `ylim` paramétereinek megadásával tehetjük meg.

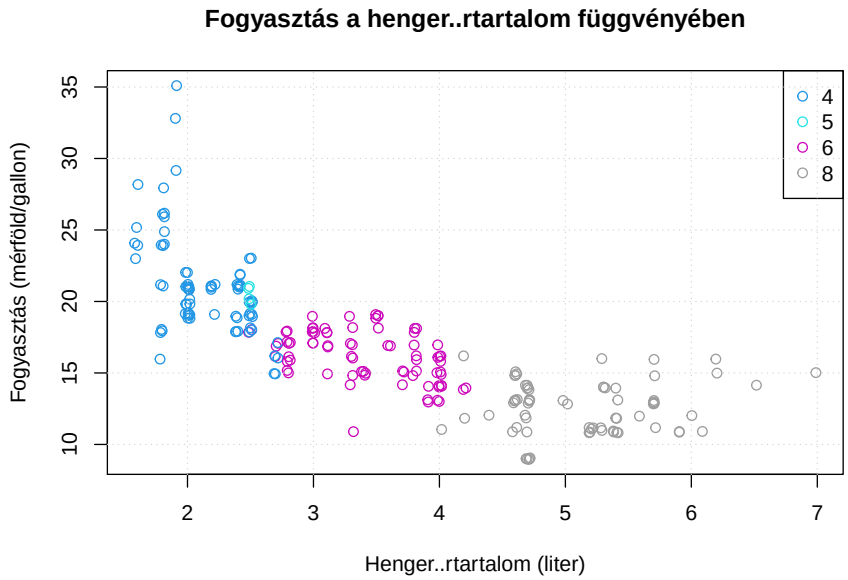
Az ábra kiegészítéseként méretvonalakat rajzolhatunk a `grid()` függvény segítségével. A függvény használata esetében is használhatjuk a grafikai paraméterek módosítását.

Egy jól reprezentált eredmény tartalmaz jelmagyarázatot. Jelmagyarázatot a `legend()` függvény segítségével rajzolhatunk. A jelmagyarázat sokféle lehetőséget teremt a testreszabásban, a

példában egy egyszerű megközelítést láthatunk. Az első paraméter segítségével megadhatjuk a jelmagyarázat pozícióját, ami a `"top"`, `"bottom"` és `"left"`, `"right"` szavak összetételével tetszőlegesen megadhatjuk. A többi grafikai paramétert a már tapasztalt módon állíthatjuk be.

A következőkben ábrázoljuk a városi fogyasztást a hengerűrtartalom függvényében a korábbi módon, viszont az egyes pontokat a hengerek száma (`cyl`) szerint színezzük ki.

```
#data$cyl <- factor(data$cyl)
plot(jitter(data$displ), jitter(data$cty), col = data$cyl,
     pch = 1,
     main = "Fogyasztás a hengerűrtartalom függvényében",
     xlab = "Hengerűrtartalom (liter)",
     ylab = "Fogyasztás (mérföld/gallon)")
# A) megoldás
#legend("topright", legend = unique(data$cyl),
#       col = unique(data$cyl), pch = 1, )
# B) sorrendhelyesen
legend("topright", legend = sort(unique(data$cyl)),
      col = sort(unique(data$cyl)), pch = 1)
# C) egy másik lehetőség
#legend("topright", legend = levels(data$cyl),
#       col = 1:length(levels(data$cyl)), pch = 1)
grid()
```




```
levels(data$cyl)
```

```
## NULL
```

```
unique(data$cyl)
```

```
## [1] 4 6 8 5
```

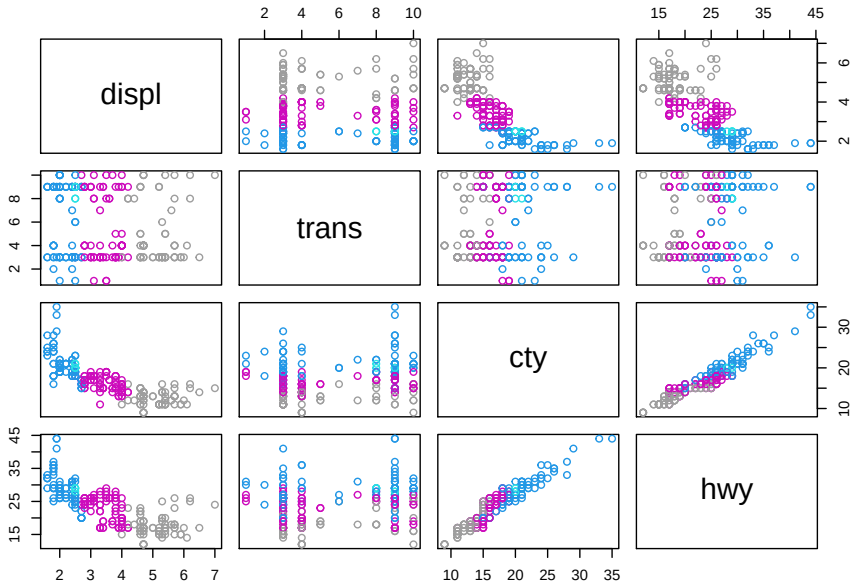
A fenti példában a színek a `cyl` vektorban tartalmazott 4, 5, 6 és 8 értékek szerint lettek meghatározva. Ugyanezeket a színeket használjuk a `legend()` függvény értékei és színezés meghatározásánál.

Amennyiben faktorizáljuk a `cyl` adatokat (töröljük a kommentet az első sorból) a színek hozzárendelése a tárolt faktor indexek alapján történik, esetünkben 1-től 4-ig (rendre a 4, 5,

6, 8 hengerszámok esetében), tehát a jelmagyarázatot is ennek alapján kell meghatározni. A példában erre három lehetőséget is találunk. A egyes megoldások működésének pontosabb megértéséhez a faktoron alkalmazott `levels()` és `unique()` függvények vezetnek. A C) változat tűnik logikusnak faktorok esetében, de ez általános esetben nem használható, ezért részesítsük előnyben a B) változatot.

A `plot()` függvény sokféle változót is kezelni képes. A lenti példában egy olyan adathalmazzal hívjuk meg a függvényt, mely három adatot is tartalmaz és nincs egyértelműen meghatározva, hogy melyiket értelmezze `x` és `y` paraméterként. Ekkor a `plot()` függvény az összes adat x-y kombinációját ábrázolja.

```
plot(mpg[,c("displ", "trans", "cty", "hwy")], col = data$cyl)
```



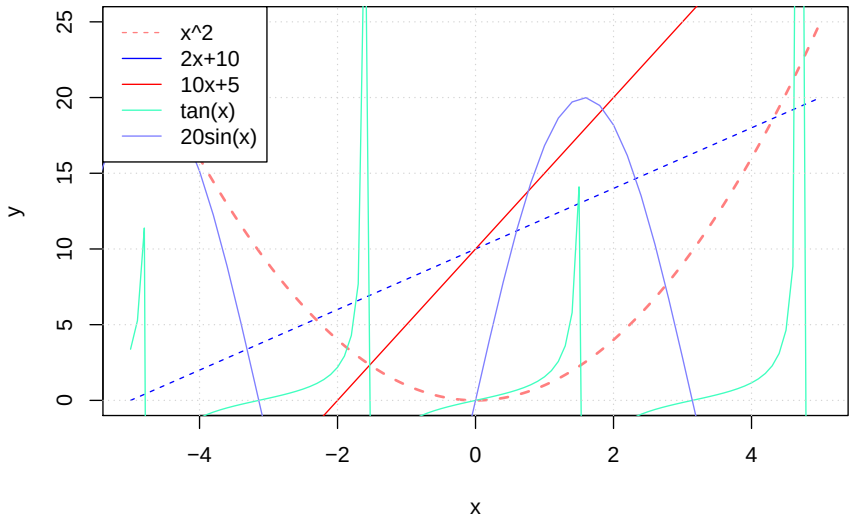
1.2. Vonaldiagram

A vonaldiagramok rajzolására ugyancsak használhatjuk a `plot()` függvényt. Ebben az esetben a úgy tekintjük, hogy a megadott `x` és `y` koordináták mint egy folytonos értékkészletű és értelmezési tartományú függvény adott pontban vett mintái, ezért a megadott pontokat összekötjük egy egyenessel ábrázolandó, hogy a két pont között is vannak értékek.

A `plot()` függvény `type` paraméterének segítségével adhatjuk meg a függvényünknek, hogy az adatokat milyen módon jelenítse meg. A "`p`" (points), mint ponthalmaz, "`l`" (line), mint vonal, "`b`" (both), mindkettő, "`h`" (histogram) hisztogram szerű, "`s`" (steps) lépcső függvény.

```
# y = x^2 függvény ábrázolása
x <- seq(-5, 5, length.out = 201) # x: értelmezési tartomány
y <- x^2                          # y: értékkészlet
plot(x, y, type = "l", lty = "dashed", lwd = 2, col = rgb(1,0,0,0.5))
# újabb vonal hozzáadása x-y értékpárok alapján
lines(x, 2*x+10, col = "blue", lty = "dashed")
# vonal hozzáadása y = b + a*x függvény alapján
abline(a = 10, b = 5, col = "red")
# beépített függvény segítségével
curve(tan, from = -5, to = 5, n = 101, col = "#40ffc0", add = T)
# felhasználó által definiált függvény segítsévégel
fn <- function(x) 20*sin(x)
curve(fn, -10, 10, col = "#8080FF", add = T)
```

```
# grid
grid()
# jelmagyarázat
legend("topleft", c("x^2", "2x+10", "10x+5", "tan(x)", "20sin(x)"),
       lwd = 1, lty = c("dashed", "solid", "solid", "solid", "solid"),
       col = c(rgb(1,0,0,0.5), "blue", "red", "#40ffc0", "#8080ff"),
       bg = "white")
```



Vonalak ábrázolását is több paraméter szerint testre szabhatjuk. A már ismert `col` paraméter segítségével adhatjuk meg a vonal színét, az `lty` (line type) segítégével a vonal típusát adhatjuk meg, mely lehet 0-`"blank"` (nem látható), 1-`"solid"` (folyamatos), 2-`"dashed"` (szaggatott), 3-`"dotted"` (pontozott), 4-`"dotdash"` (pontvonal), 5-`"longdash"` (hosszú szaggatott), 6-`"twodash"` (dupla szaggatott). Az `lwd` paraméter segítségével a vonal vastagságát adhatjuk meg.

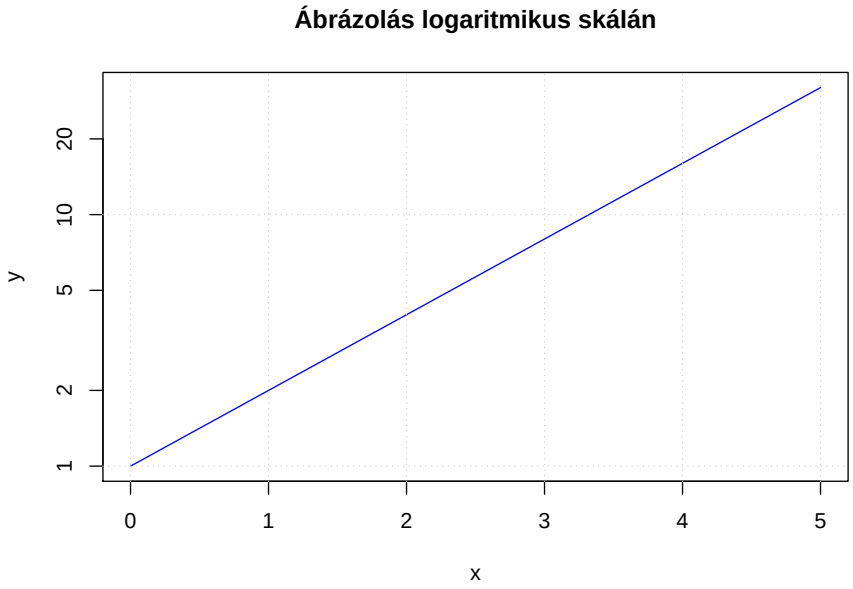
A meglévő ábrához (legyen akár ponthalmaz, vagy vonaldiagram) újabb vonaldiagram adható a `lines()` függvény használatával, ami az eddig használt módon paraméterezhető.

Az `abline()` függvényünk lehetőséget ad rá, hogy az $y = a \cdot x + b$ alakú lineáris függvényt kirajzolhassuk csupán az `a` és `b` paraméterek megadásával.

Lehetőségünk van magának a függvénynek a megadásával is vonaldiagramot rajzolnunk. Alapvetően a `curve()` függvény egy kifejezés megadásával kirajzolja az adott kifejezés eredményét a `from` és `to` értékek között. Valójában ez a függvény is `x` és `y` minták alapján dolgozik, melyeket a megadott paraméterek között mintavételez `n`-szer. A `curve()` függvény egy új ábrát hoz létre, viszont ha az `add` paraméter értékét `TRUE`-ra állítjuk, a már meglévő ábrára rajzol.

Amennyiben a megjelenítés során logaritmikus léptékű tengelyeken szeretnénk ábrázolni az értékeket a `log` paramétert kell használjuk “x”, “y”, “xy” értékek megadása esetében rendre az x, az y, illetve mindkét tengelyen logaritmikusan kerülnek ábrázolásra az értékek. Erre nézzük a következő példát.

```
x <- seq(0.0001, 5, length.out = 201)
y <- 2^x
plot(x, y, type = "l", col = "blue", log = "y",
     main = "Ábrázolás logaritmikus skálán")
grid()
```

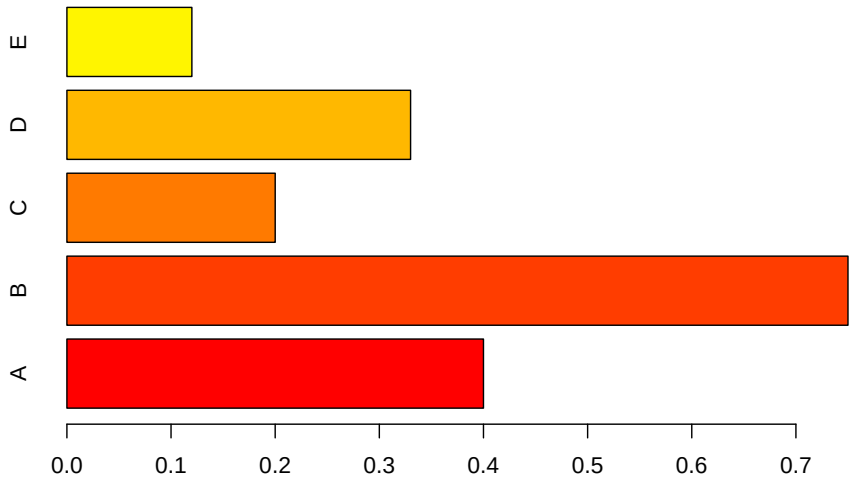


1.3. Barplot

A `barplot()` lehetőséget teremt értékek sorozatának megjelenítésére bár formátumban, legyen az értékek gyakorisága, vagy bármilyen kategóriához rendelt adat.

A következőkben ábrázoljunk egy numerikus vektort, mely névvel ellátott csoportokhoz rendelt értékeket tartalmaz. Az egyes csoportokat az ABC betűivel nevezzük el.

```
vals <- c(0.4, 0.75, 0.2, 0.33, 0.12)
names(vals) <- LETTERS[1:5]
barplot(vals, col = rainbow(25), horiz = T)
```




```
#names.arg = group.names
```

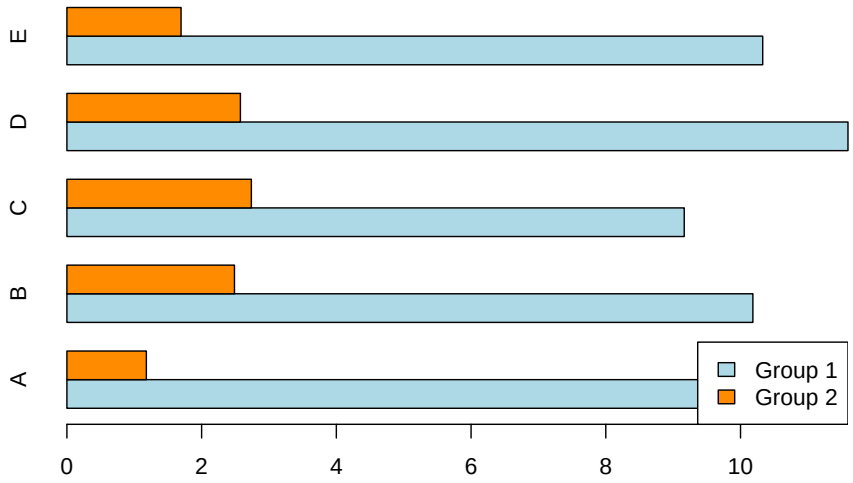
Látható, hogy a `barplot()` a vektor egyes értékei alapján rajzol egy-egy sávot. Amennyiben a vektor egyes elemei karakter értékek alapján el lett nevezve az ábrázolás során automatikusan ez került névként hozzárendelve az egyes sávokhoz. Amennyiben ezen változtatni szeretnénk, használhatjuk a `barplot()` `names.arg` paraméterét.

A `col` paraméter megadásával változtathatjuk meg a sávok színét, esetünkben a `rainbow()` függvénnyel, mely a szivárvány színei közül választ egyenközűen a paraméterként megadott számú színt. Itt a `rainbow()` 25 értéket ad vissza, viszont csak az első 5 színt használjuk.

A `horiz=T` paraméter megadásával a horizontálisan jeleníthetjük meg a sávokat.

Amennyiben nem vektorral, hanem mátrixszal (két dimenziós adattal) hívjuk meg a `barplot()` függvényt, az egyes sorok, mint az értékekhez rendelt csoportok jelennek meg. Az egyes csoportok megjelenítését szabhatjuk a megjelenítés során testre.

```
set.seed(1)
# mátrix létrehozása
data2 <- rbind(rnorm(5, 10, 1), rnorm(5, 2, 1))
# sorok, oszlopok elnevezése
colnames(data2) <- LETTERS[1:5]
rownames(data2) <- c("Group 1", "Group 2")
# megjelenítés
barplot(data2, col = c("lightblue", "darkorange"), horiz = T, beside = T)
# jelmagyarázat
legend("bottomright", legend = rownames(data2),
       fill = c("lightblue", "darkorange"), bg = "white")
```



A megjelenítés során a `beside` paraméter segítségével állíthatjuk be, hogy az egyes csoportokhoz tartozó sávok egymás mellett kerüljenek ábrázolásra.

A jelmagyarázat megadása esetében a `fill` paraméter esetében területek színezését állíthatjuk be.

Készítsünk egy összegzést, mely megmutatja az egyes autómárkák esetében, hogy hengerenként hányféle autó szerepel az adatok között!

```
# autómárkák
data$manufacturer <- factor(data$manufacturer)
levels(data$manufacturer)
```

```
##  [1] "audi"       "chevrolet"  "dodge"      "ford"       "honda"
##  [6] "hyundai"    "jeep"       "land rover" "lincoln"    "mercury"
## [11] "nissan"     "pontiac"    "subaru"     "toyota"     "volkswagen"
```

```
# a lehetséges hengerszámok
unique(data$cyl)
```

```
## [1] 4 6 8 5
```

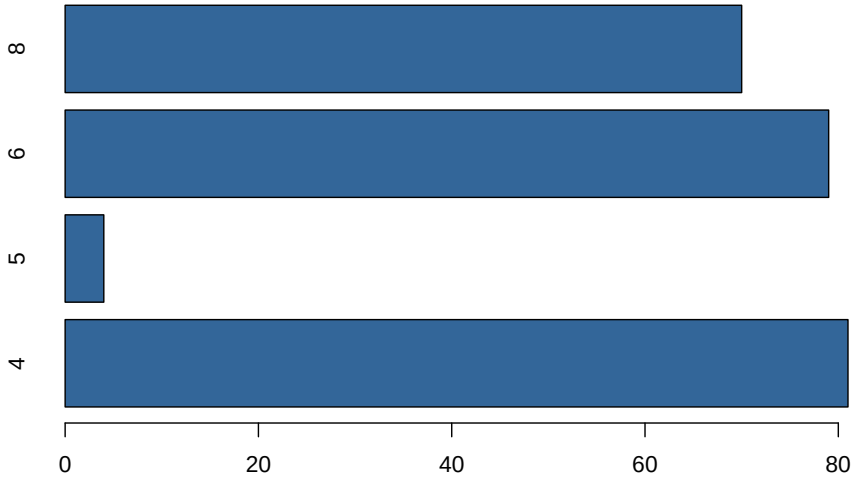
```
# egy adott hengerszámra hány autó van
length(which(data$cyl == 4))
```

```
## [1] 81
```

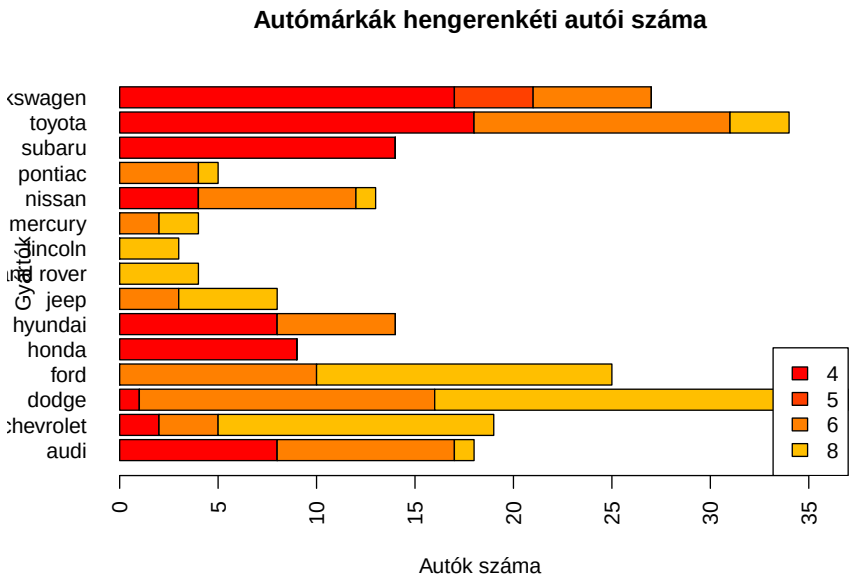
```r
#, vagy egyszerűbben
sum(data$cyl == 4)
```

```
## [1] 81
```

```r
# minden egyes hengerszámra a kocsik száma
a <- tapply(data$cyl, data$cyl, length)
barplot(a, horiz = T, col = "#336699")
```




```r
# minden egyes autómárkára hengerenkénti kocsik száma
b <- table(data[,c("cyl", "manufacturer")])
barplot(b, col = rainbow(24), horiz = T, las = 2,
        main = "Autómárkák hengerenkéti autói száma",
        xlab = "Autók száma", ylab = "Gyártók")
legend("bottomright", rownames(b), fill = rainbow(24), bg = "white")
```



A fenti példában újdonságnak számít a `las` grafikai paraméter használata. Ennek segítségével megváltoztathatjuk az egyes tengelyere írt adatok, mérőszámok megjelenítésének irányát (0 - alapértelmezett, párhuzamos a tengellyel, 1 - horizontális, 2 - merőleges a tengellyel, 3 - vertikális).

1.4. Hisztogram

Hisztogramot értékek gyakoriságának ábrázolására, eloszlására használhatjuk.

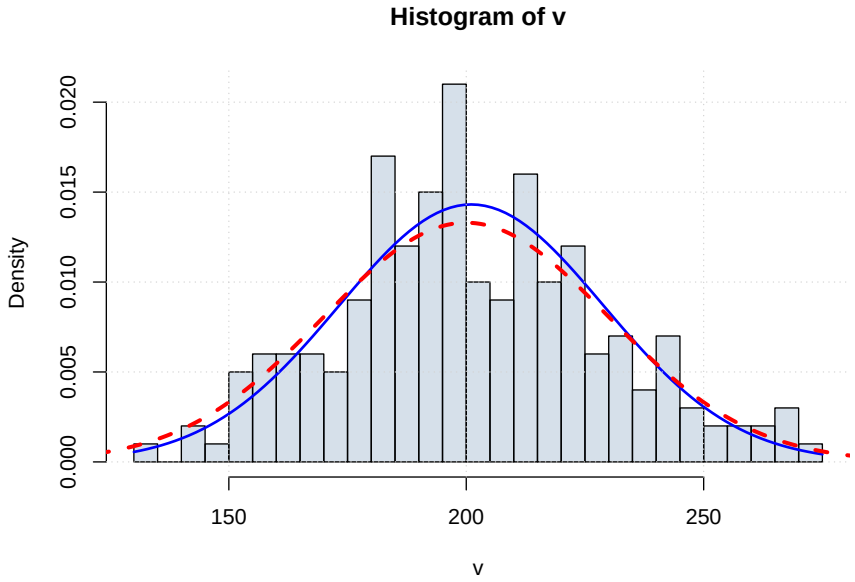
A következőkben generáljunk néhány száz értéket normális eloszlás szerint 200 várható értékkel és 30 szórással, majd ábrázoljuk a hisztogramját 20 csoport kirajzolásával.

Ezután ábrázoljunk egy-egy normális eloszlás sűrűségfüggvényét a tapasztalt, majd az eredeti várható értékek és szórások alapján.

```
# normális eloszlás szerinti változók generálása
set.seed(1)
v <- rnorm(n = 200, mean = 200, sd = 30)
# a hisztogram megjelenítése
h <- hist(v, probability = T, breaks = 20, col = rgb(0.2, 0.4, 0.6, 0.2))
attributes(h)
```

```
## $names
## [1] "breaks"   "counts"   "density"  "mids"     "xname"    "equidist"
##
## $class
## [1] "histogram"
```

```
# sűrűségfüggvéy ábrázolása
v.m <- mean(v)   # tapasztalati várható érték
v.sd <- sd(v)    # tapasztalati szórás
# parametrikusan
curve(dnorm(x, v.m, v.sd), col = "blue", lty = 1, lwd = 2, add = T)
# explicit x-y értékekkel az eredeti szórás és várható érték alapján
lines(100:300, dnorm(100:300, 200, 30), col = "red",
      lty = "dashed", lwd = 3)
grid()
```



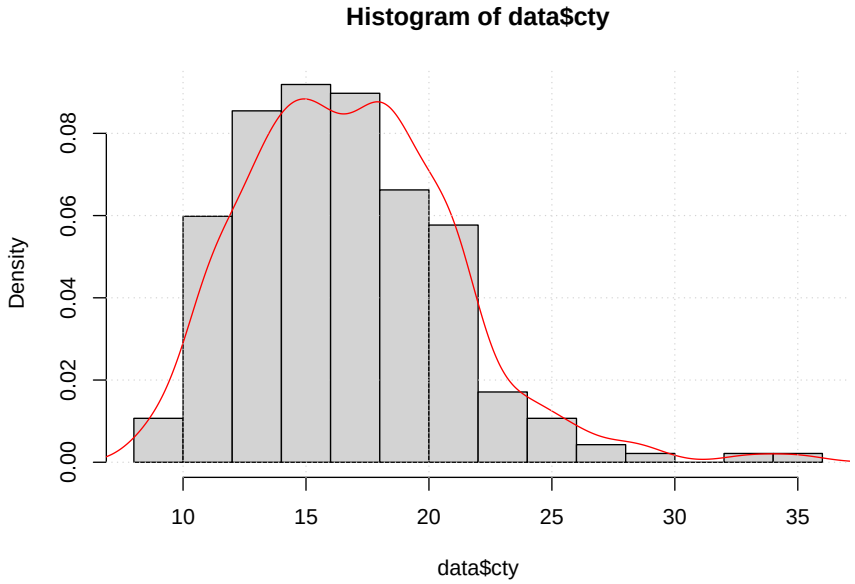
A `hist()` függvény gyakorlatilag az értéktartományt egy meghatározott számú egyenlő nagyságú intervallumba osztja és megszámolja, hogy a megadott értékek közül mennyi esik az egyes tartományokba, majd ezeket az értékeket ábrázolja. Az intervallumok számosságát, vagy vektor esetében pontos értékét a `breaks` paraméterrel adhatjuk meg, az ábrázolt értékek jellegét pedig a `freq`, illetve `probability` paraméterekkel. Alapesetben (`freq=T`) a függvény az interval-

lumban tartózkodó értékek számát (`count`) jeleníti meg, ami az egyes értékek előfordulásának frekvenciája, amennyiben szeretnénk az adott intervallum előfordulási valószínűségét megjeleníteni (`probability=T`) az intervallumok előrodulásának sűrűsége jelenik meg, mely során, mint valószínűségi változók esetében a "görbe" alatti terület összege 1 lesz. Ennek segítségével hasonlíthatjuk az előfordulási gyakoriságot valószínűségi sűrűségfüggvényekkel.

A `hist()` függvény nem csak ábrázol, de releváns értékekkel is visszatér, például a `breaks` attribútuma tartalmazza az intervallumok határpontjait, valamint a `counts` attribútuma tartalmazza, hogy egy adott intervallumba hány érték esett.

A következőkben ábrázoljuk az adathalmazunk autói városi fogyasztásának hisztogramját, valamint a közelítő sűrűség függvényét, melyet a `density()` függvény segítségével számíthatunk ki.

```
hist(data$cty, breaks = 10, probability = T)
lines(density(data$cty), col = "red")
grid()
```



1.5. Boxplot

A `boxplot()` egy nagyon hasznos függvény, mely értékek eloszlását szemlélteti. A következő értékeket jeleníti meg:

- *medián (Q2)*: Az érték, melytől a vizsgált értékek fele nagyobb, a másik fele kisebb.
- *1. és 3. kvartilis érték (Q1, Q3)*: Mely értéktől a vizsgált értékek negyede kisebb, illetve nagyobb.
- *minimum, maximum*: A nem kiugró értékek minimuma, illetve maximuma.
- *kiugró értékek*: Az adathalmaztól túl messze lévő értékek, a medián értéktől az interkvartilis (IQR, inter quartile range, IQR = Q3 - Q1) érték másfélszeresénél (1.5*IQR) messzebb lévő pontok.

A vizsgált adathalmazon ábrázoljuk az egyes hengerszámokhoz tartozó városi fogyasztás eloszlásainak boxplot ábráit.

```
# hengerszámonkénti boxplot a városi fogyasztás értékeire
bx <- boxplot(data$cty[data$cyl == 4], data$cty[data$cyl == 5],
        data$cty[data$cyl == 6], data$cty[data$cyl == 8],
        names = c("4 henger", "5 henger", "6 henger", "8 henger"),
```

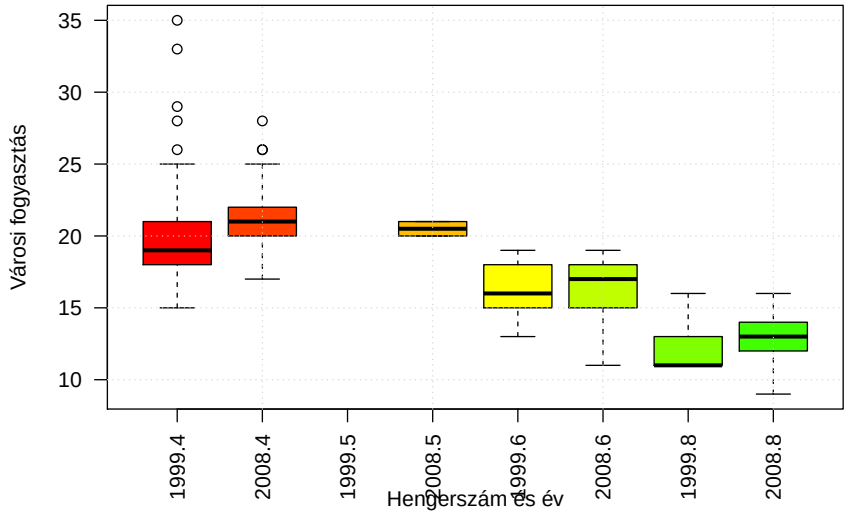
```
        xlab = "Hengerszám", ylab = "Városi fogysztás")
grid()
```

Mint látható, a `hist()` függvény annyi adat alapján végzi el az ábrázolást, amennyi adatot átadunk neki. Az első függvényhívás esetében kézzel végeztük el a szűrést a négy különféle hengerenkénti számra, így az egyes értékek neveit is kézzel kell megadni a `names` paraméter segítségével.

A hisztogram visszatérési értékéből (`bx`) kiolvashatjuk a jellemző 5 értéket mind a négy adathalmazra (`stats`), a vizsgált elemek számát (`n`), a konfidencia intervallumokat (`conf`), az kiugró értékeket (`out`) és hogy ezek az értékek melyik csoporthoz tartoznak (`group`) és az egyes csoportoknak mi a nevük (`names`).

Látható, hogy a teljes értékhalmaz csoportokra bontása rendkívül körülményes. Ugyancsak bajba kerülünk, ha több paraméter szerint szeretnénk csoportot képezni, például nem csak a hengerszámok, hanem az évjárat szerint is. Ezt mutatja a következő feladat.

```
# hengerszámonként és évenként a városi fogyasztás
boxplot(data$cty ~ data$year + data$cyl, las = 2, col = rainbow(24),
        xlab = "Hengerszám és év",
        ylab = "Városi fogyasztás")
grid()
```



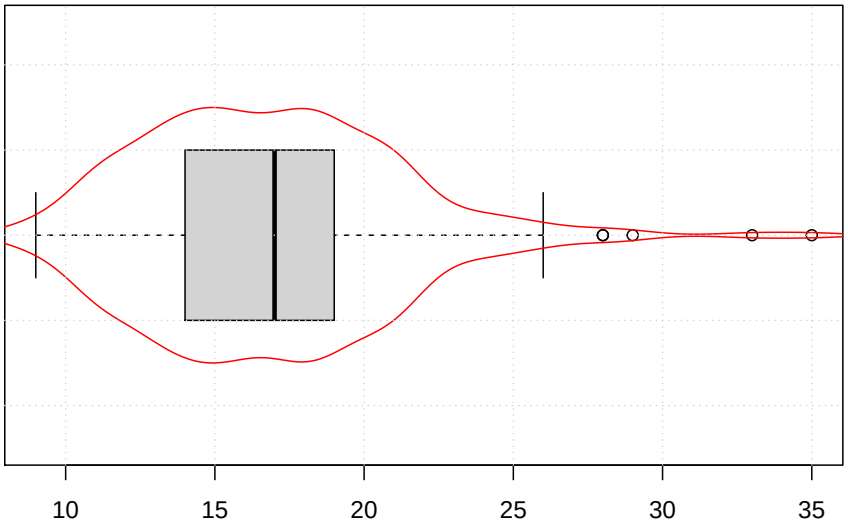
Az adatok megadása esetében a fenti példában a formula objektumot használtuk. A formulák

adatok közötti összefüggéseket határoznak meg. Az alapvetően az `y ~ x` azt mutatja, hogy az `y` érték az `x` éréktől függ. Az `y` érték nem csak egy függvénytől függhet, akár többtől is. Amennyiben ezek az értékek egymástól függetlenek, `+` jellel, míg egymással összefüggenek `*` jellel szeparáljuk. Tehát az `y ~ x1 + x2` azt jelenti, hogy `y` függ `x1`-től és `x2`-től is, de egymástól függetlenül.

Tehát a példában adott `data$cty ~ data$year + data$cyl` kifejezést úgy értelmezi az ábrázoló függvény (nem csak a `boxplot()`) hogy mivel a `cty` függ a `year`-tól és `cyl`-től, akkor nosza rajta ábrázoljuk ezen paraméterek függvényében.

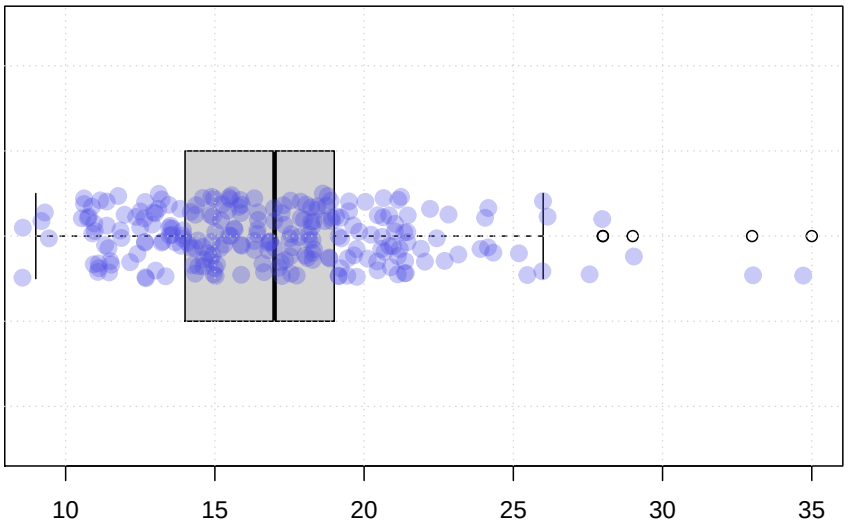
A következő példák szemléletes példái az ábrázolások rugalmasságának. A következőkben a boxplot megjelenítést egészítjük ki az értékek sűrűség függvényével.

```
boxplot(data$cty, horizontal = TRUE)
d <- density(data$cty)
# az eloszlást 0.3 értékig maximalizáljuk, tudván
# hogy a boxplot szélessége 0.5 nagyságrendű
lines(d$x, 1 + 0.3*d$y/max(d$y), col = "red")
lines(d$x, 1 - 0.3*d$y/max(d$y), col = "red")
grid()
```



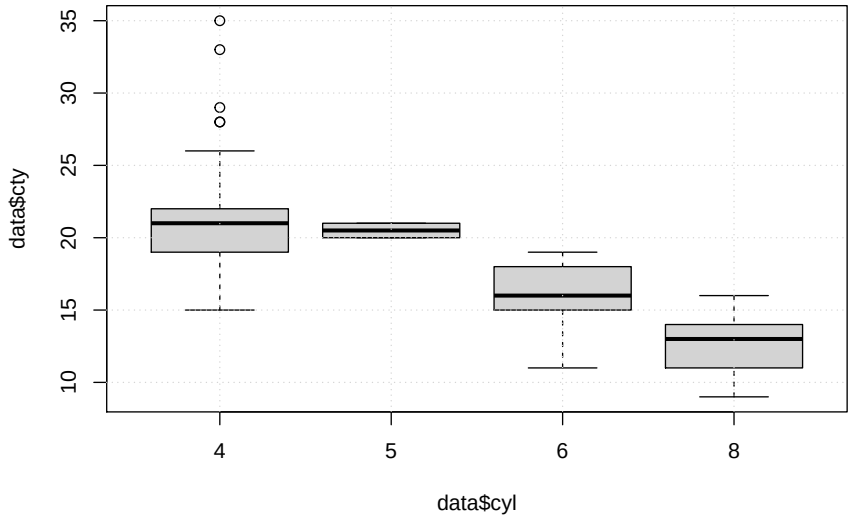
A következő példában a boxplot mellett jelenítjük meg az egyes realizációkat.

```
boxplot(data$cty, horizontal = TRUE)
points(jitter(data$cty, amount = 0.5),
       jitter(rep(1, length(data$cty)), amount = 0.1),
       pch = 19, cex = 1.5, col = rgb(0.3,0.3,0.9,0.3))
grid()
```



A realizációk megjelenítését elvégezhetjük csoportonként is a következő példában bemutatott módon. Itt megjegyezzük, hogy a `for` használata itt nem adatcentrikus, hanem procedurális, azaz nem az adatok ciklikus kinyerése, hanem műveletek parametrikus végrehajtása végett került bevezetésre.

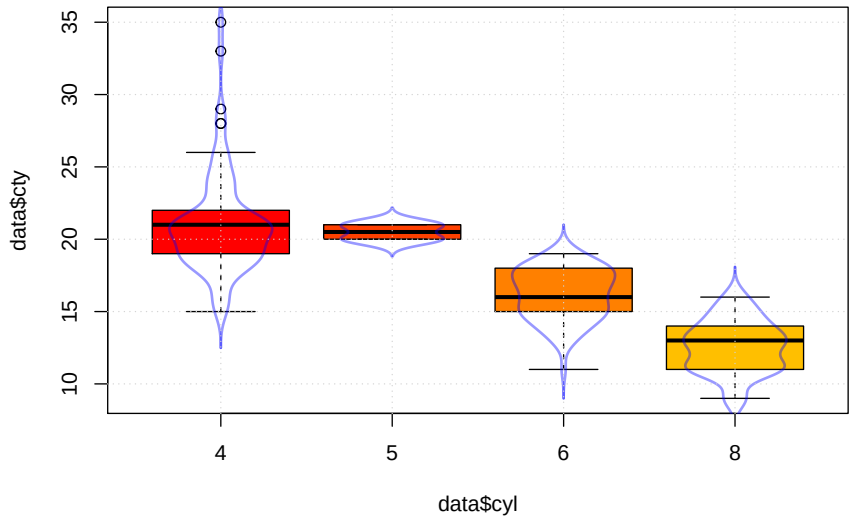
```
boxplot(data$cty ~ data$cyl)

for(cyl in levels(data$cyl)) {
  y <- data$cty[data$cyl == cyl]
  x <- which(levels(data$cyl) == cyl)
  points(jitter(rep(x, length(y)), amount = 0.1), jitter(y),
         pch = 19, col = rgb(0.3,0.3,0.9,0.3))
}
grid()
```



Ugyancsak csoportonként elvégezhetjük a sűrűségfüggvények kirajzolását, mely segítségével jobban áttekinthetők az adatok.

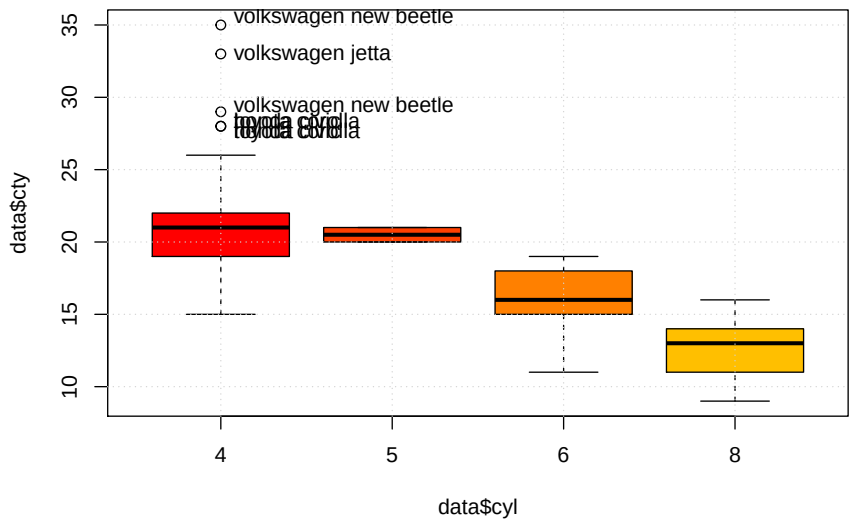
```
data$cyl <- factor(data$cyl)
boxplot(data$cty ~ data$cyl, col = rainbow(24))
for (cyl in levels(data$cyl)) {
  d <- density(data$cty[data$cyl == cyl])
  lines(rep(which(levels(data$cyl) == cyl), length(d$y)) +
```

```
          0.3*d$y/max(d$y), d$x, col = rgb(0,0,1,0.4), lwd = 2)
  lines(rep(which(levels(data$cyl) == cyl), length(d$y)) -
          0.3*d$y/max(d$y), d$x, col = rgb(0,0,1,0.4), lwd = 2)
}
grid()
```



Lehetőségünk van egyes értékek feliratozására is a `text()` függvénnyel a következő példában ismertetett módon.

```
if(!is.factor(data$cyl)) data$cyl <- factor(data$cyl)
# boxplot ábrázolása csoportonként
bp <- boxplot(data$cty ~ data$cyl, col = rainbow(24))
# minden kiugró adatra
for(i in 1:length(bp$out)) {
  # meghatározzuk mely azon adatok indexét, melyeket a kiugró adatok
  # csoportja (mint hengerszám) és fogyasztás értéke határoz meg
  caridx <- data$cyl == levels(data$cyl)[bp$group[i]] &
                        data$cty == bp$out[i]
  # meghatározzuk a kiválaszott adatok megjelenési nevét a
  # gyártók és modelnév alapján
  name <- paste(data$manufacturer[caridx], data$model[caridx])
  # az adatok helyére kiírjuk a neveket
  text(bp$group[i], jitter(bp$out[i], amount = 0.5),
       name, adj = c(0,0), offset = 0.5, pos = 4)
}
grid()
```

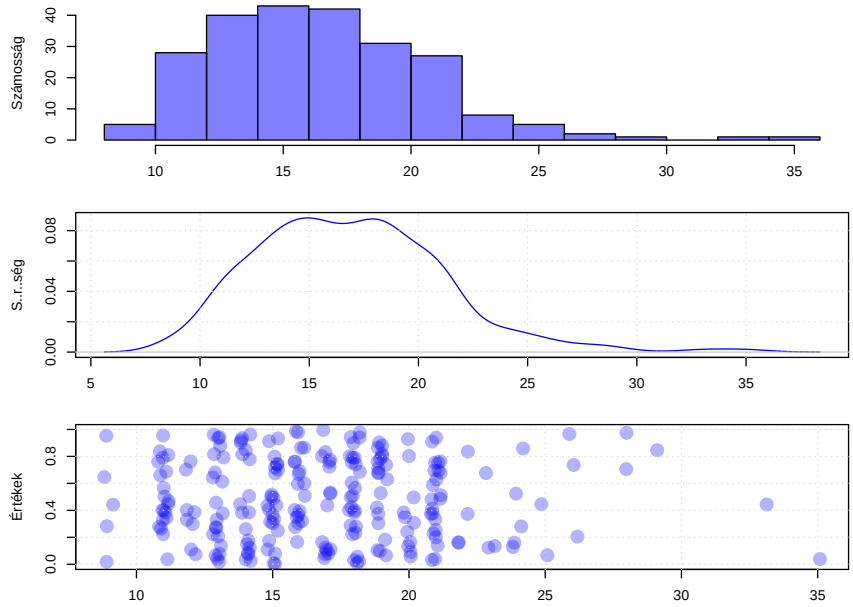


1.6. Több diagram ábrázolása

Általános esetben több egymástól független diagram is rajzolható egymás mellé, illetve alá. Ezt a `par()` függvénnyel tehetjük meg. Az `mfrow` (mutli figure by row), illetve az `mfcol` (multi figure by column) paraméterek segítségével tehetjük meg. Mind a két esetben egy kételemű vektort adunk meg, mely jelzi a megjelenítés sorainak és oszlopainak számát. A két paraméter közötti különbség, hogy míg az egyis soronként, addig a másik oszloponként rajzolja ki a diagramokat.

Lássuk ezt a következő példán.

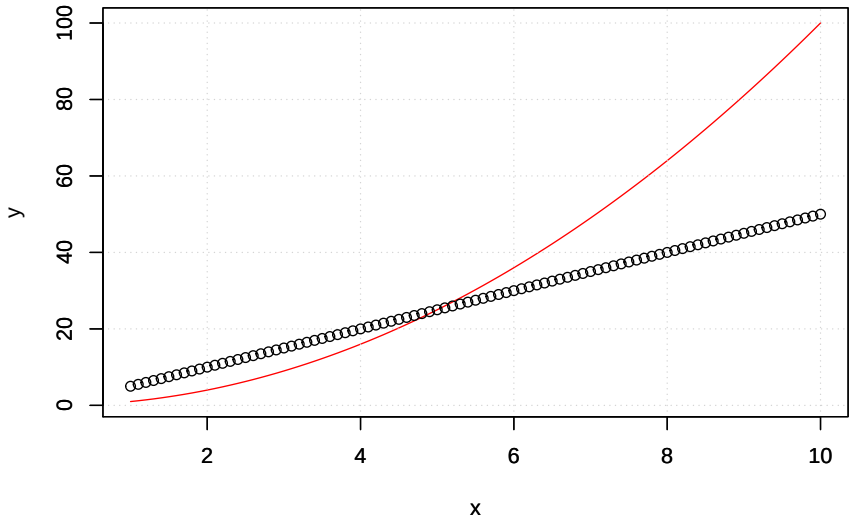
```
# Diagramok három sorban, egy oszlopban
par(mfrow = c(3,1))
# Az ábra margóinak megadása
# lent, balra, fent, jobbra
par(mar = c(2,5,2,2))
# Első ábra
hist(data$cty, breaks = 10, col = rgb(0, 0, 1, 0.5),
     main = "", ylab = "Számosság")
# Második ábra
plot(density(data$cty), col = "blue",
     main = "", ylab = "Sűrűség")
grid()
# Harmadik ábra
plot(x = jitter(data$cty), y = runif(length(data$cty)),
     type = "p", pch = 19, cex = 2, col = rgb(0, 0, 1, 0.3),
     main = "", ylab = "Értékek")
grid()
```



1.7. Ábrák egymásra rajzolása

Diagrammok a `plot()` segítségével is egymásra rajzolhatók a `par(new = TRUE)` parancs kiadásának segítségével. Ennek hatására az új rajzolás során nem keletkezik új diagram. Ebben az esetben a tengelyek is újrarajzolásra kerülnek, így az ábrázolási tartományt és a tengelyek nevét is körültekintően kell beállítani.

```
x <- seq(1, 10, by = 0.1)
y <- x**2
plot(x, y, type = "l", col = "red")
grid()
par(new = TRUE)
# rajzoljunk ugyanabban az y tartományban
plot(x, 5*x, ylim = range(y), ylab = "")
```



Vegyük észre, hogy fenti példában mind a tengelyek és a tengelyeken ábrázolt értékek kétszer kerültek kirajzolásra. Amennyiben ezt nem szeretnék, a második rajzolás esetében az `xaxt="n"`, valamint a `yaxt="n"` paraméterek megadásával mellőzhetjük a tengelyek kirajzolását.