Przetwarzanie i wizualizacja danych

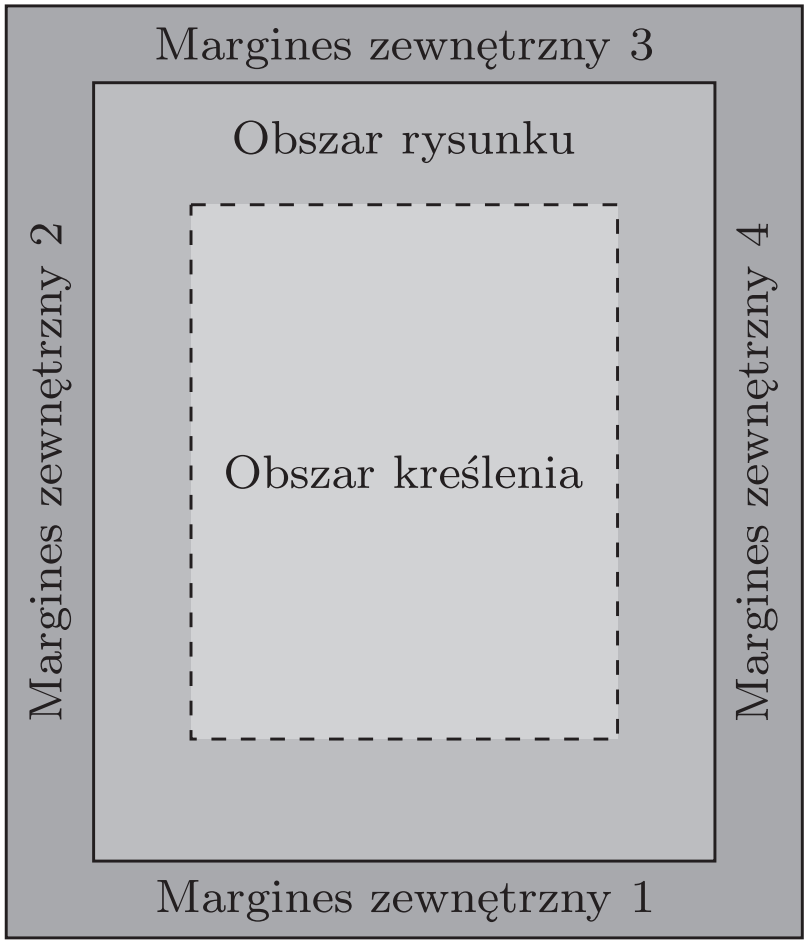
prof. UAM dr hab. Tomasz Górecki

tomasz.gorecki@amu.edu.pl

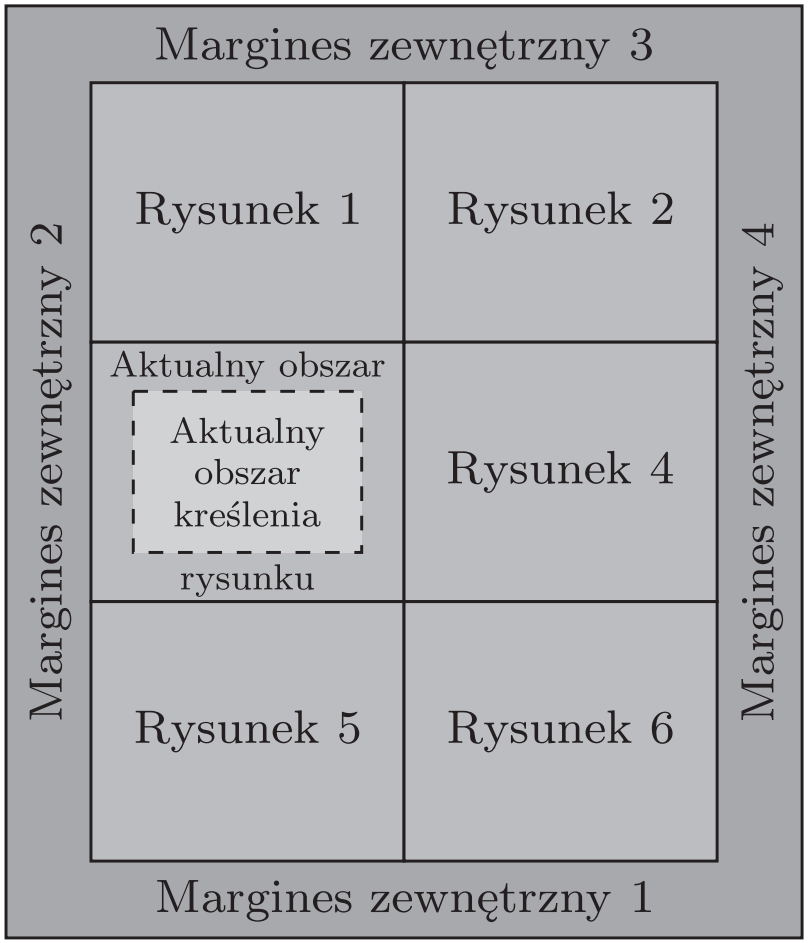
Zakład Statystyki Matematycznej i Analizy Danych
Wydział Matematyki i Informatyki
Uniwersytet im. Adama Mickiewicza w Poznaniu

R dysponuje bardzo dużymi możliwościami w zakresie graficznej prezentacji uzyskanych wyników. Aby jednak w pełni korzystać z jego możliwości zapoznamy się wpierw z systemem prezentacji grafiki. Bazowy system graficzny dzieli każdą stronę na trzy główne obszary: marginesy zewnętrzne (*ang. outer margins*), obszar rysunku (*ang. figure region*) oraz obszar kreślenia (*ang. plot region*). Obszar jaki uzyskujemy po odrzuceniu marginesów zewnętrznych nazywany jest obszarem wewnętrznym (*ang. inner region*).

Podział strony graficznej



a)



b)

Obszar na zewnątrz obszaru kreślenia, ale wewnątrz obszaru rysunku nazywa się marginesami rysunku (*ang. figure margins*). Większość funkcji rysujących wykorzystuje obszar kreślenia do rysowania symboli graficznych oraz linii, natomiast osie oraz etykiety nanoszone są na marginesach rysunku lub marginesach zewnętrznych. Rozmiar oraz położenie marginesów kontrolowane są za pomocą funkcji par. Polecenia graficzne w R można podzielić na trzy typy: funkcje wysokiego poziomu, funkcje niskiego poziomu oraz funkcje parametrów graficznych.

Funkcje wysokiego poziomu

Tworzą nowy wykres:

- plot – wykres punktowy, najpopularniejsza funkcja graficzna, przeciążona praktycznie dla każdego obiektu,
- curve – wykres funkcji.

Funkcje wysokiego poziomu

Każda funkcja wysokiego poziomu może zostać wywołana z parametrami.

$add = FALSE$ – jeżeli TRUE, nakłada wykres na już istniejący,

$axes = TRUE$ – jeżeli FALSE, nie rysuje osi i pudełka naokoło,

$type = 'p'$ – określa rodzaj wykresu. Główne typy to: 'p' – punkty, 'l' – linie bez punktów, 'b' – punkty połączone za pomocą linii, 'o' – linie oraz punkty na nich leżące, 'h' – linie pionowe, 's' oraz 'S' – linie łączą punkty (najpierw w poziomie potem w pionie lub na odwrót), 'n' – pusty wykres,

$xlim =, ylim =$ – określa początek i koniec osi,

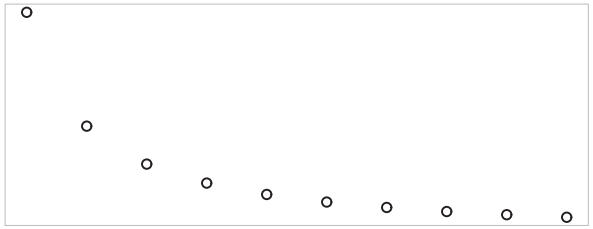
$xlab =, ylab =$ – tytuły osi,

$main =$ – tytuł wykresu,

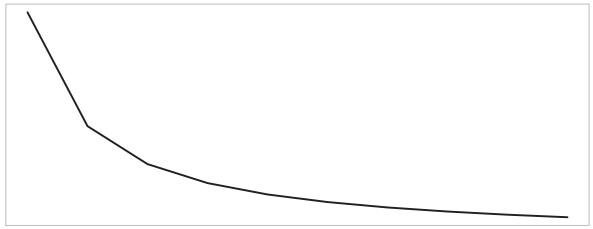
$sub =$ – podtytuł (mniejsza czcionka).

Użycie parametru type w procedurach graficznych

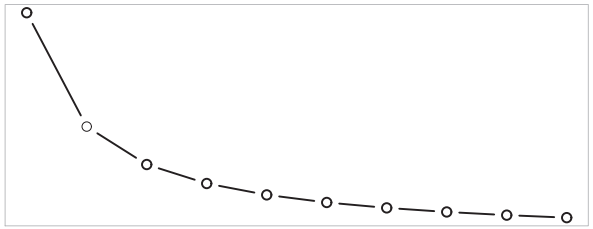
type='p'



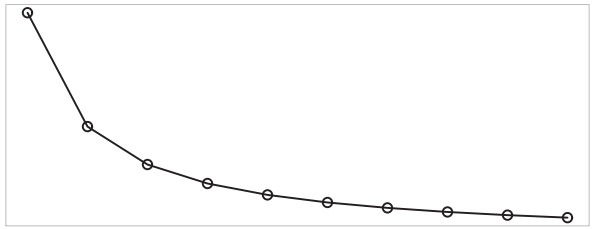
type='l'



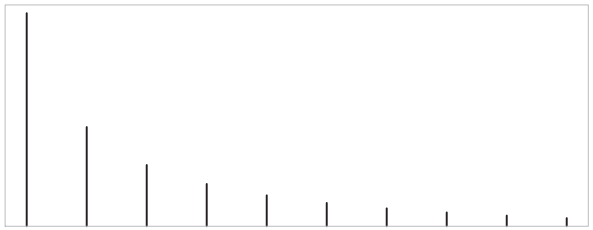
type='b'



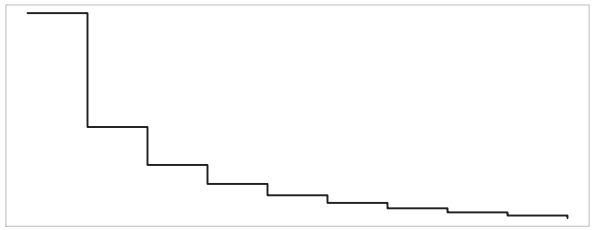
type='o'



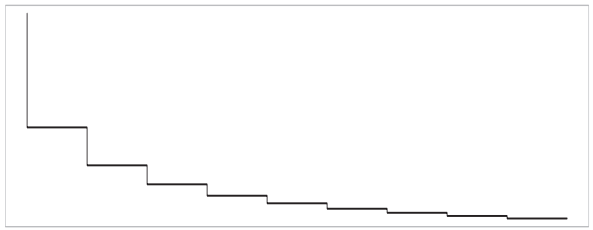
type='h'



type='s'



type='S'



Funkcje niskiego poziomu

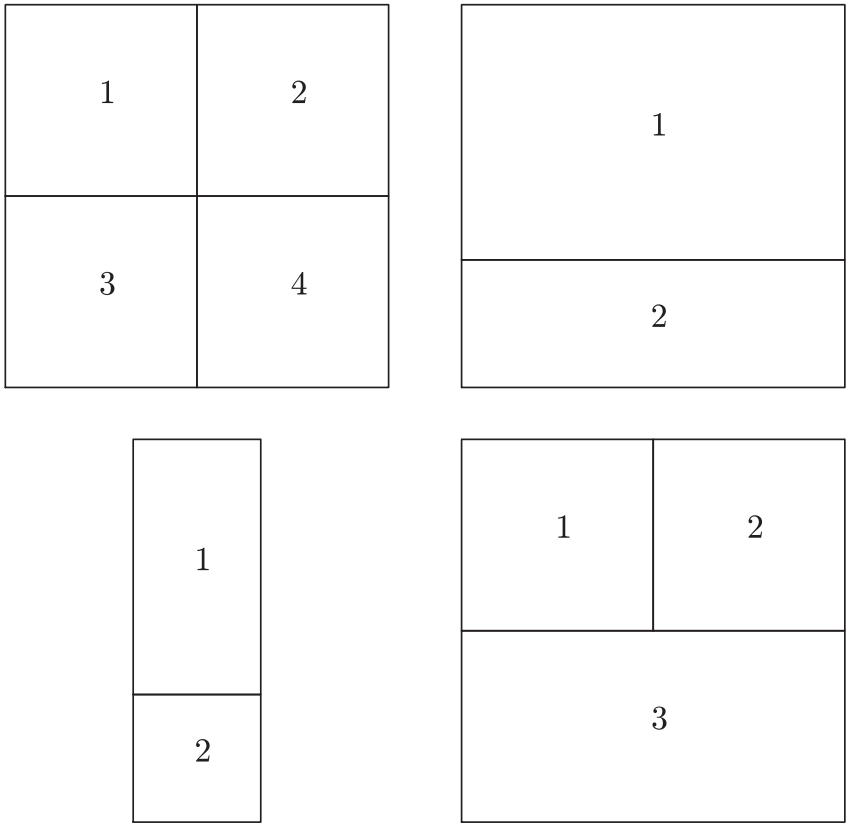
Funkcje takie dodają pewne elementy do już istniejącego wykresu:

- abline(a, b) – rysuje prostą $y = ax + b$,
- arrows(x0, y0, x1, y1) – dodaje strzałkę,
- legend(x, y, legend = 'legenda') – dodaje legendę,
- lines(x, y) – rysuje linię,
- points(x, y) – rysuje punkty,
- poly(x, y) – rysuje wielokąt,
- rect(x0, y0, x1, y1) – rysuje prostokąt,
- text(x, y, label = 'tekst') – dodaje 'tekst' w punkcie (x, y). Możemy dodawać opisy matematyczne (i nie tylko), których składnia opiera się na wewnętrznym systemie R zbliżonym do TEXa,
- title('tytuł') – dodaje tytuł.

Funkcje parametrów graficznych

Zmieniają oraz poprawiają wygląd okna graficznego. Większość ustawień jest kontrolowana poprzez funkcję par, która wywołana bez parametrów wyświetla bieżące ustawienia okna. Jedną z najważniejszych opcji oferowanych przez par jest podział okna graficznego. Uzyskujemy to za pomocą opcji mfrow oraz mfcol. W tym pierwszym przypadku obrazki rysowane są po wierszach, w drugim po kolumnach. Bardziej skomplikowane układy mogą być uzyskane za pomocą poleceń split.screen oraz layout.

Przykładowe układy rysunków stworzone za pomocą funkcji layout



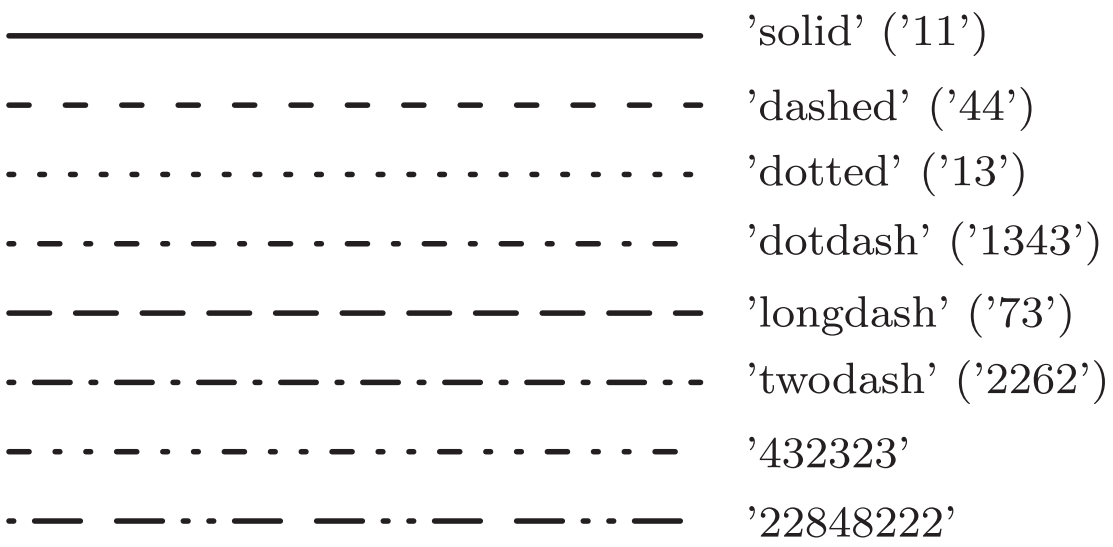
Podstawowe parametry graficzne (1)

ask	Ustawienie na *TRUE* powoduje, że przed rozpoczęciem rysowania należy nacisnąć dowolny klawisz.
adj	Wyrównanie tekstu, 0 – do lewej, 0.5 – do środka, 1 – do prawej.
bg	Kolor tła
bty	Kontroluje rodzaj obramowania rysunku (w połączeniu z funkcją box). Dopuszczalne wartości to: 'n', 'o', 'l', '7', 'c', 'u', ']', gdzie 'n' oznacza brak obramowania, natomiast pozostałe parametry określają typy obramowania zgodne z wyglądem znaków, np. : 'u' – wszystkie linie poza górną.
cex	Wielkość powiększenia symboli i tekstu. Można również używać: cex.axis, cex.lab, cex.main, cex.sub.
col	Kolor punktów, linii, tekstu oraz wypełnionych obrazków. Za pomocą dodatkowych poleceń (col.axis, col.lab, col.main, col.sub) uzyskujemy kolor osi, etykiet oraz tytułu i podtytułu.
font	Typ czcionki dla tekstu: 1 – normalna, 2 – pogrubiona, 3 – pochylona, 4 – pogrubiona i pochylona. Dla osi, etykiet oraz tytułu i podtytułu zmieniamy czcionkę za pomocą font.axis, font.lab oraz font.main i font.sub odpowiednio.

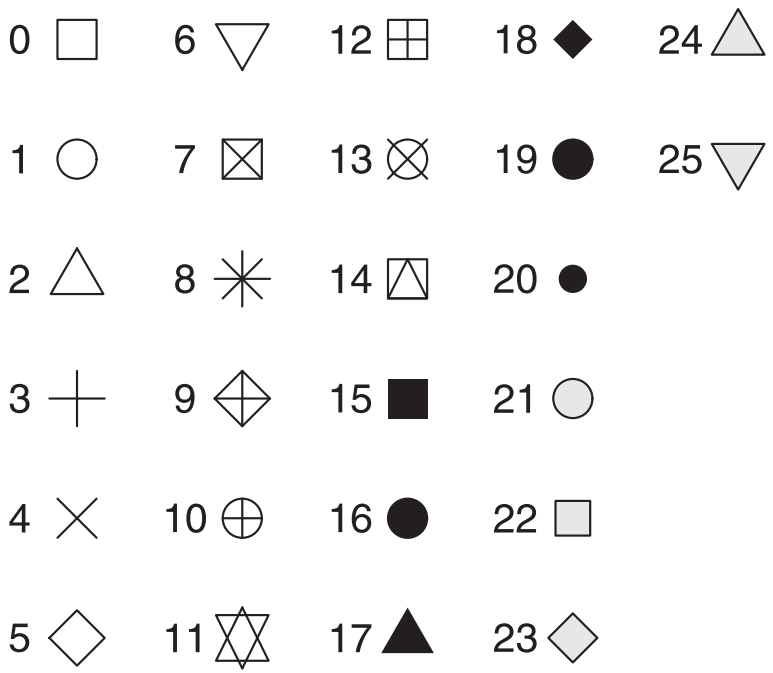
Podstawowe parametry graficzne (2)

lty	Określa typ linii, 0 – brak linii ('blank'), 1 – ciągła ('solid'), 2 – przerywana ('dashed'), 3 – kropkowana ('dotted'), 4 – kropka-kreska ('dotdash'), 5 – długa kreska ('longdash'), 6 – podwójna kreska ('twodash'). Poza tym istnieje możliwość określenia dowolnej linii, w taki sposób, że po kolei w liczbach szesnastkowych podajemy ile jednostek ma być wypełnionych linią, a ile pustych (maksymalnie osiem znaków, tylko parzyste długości), np:. lty = '13' oznacza linię kropkowaną.
lwd	Grubość linii
new	Ustawienie na *TRUE* powoduje, że nowy wykres rysowany jest na istniejącym wykresie bez jego czyszczenia.
pch	Symbol używany na obrazkach, można używać domyślnych symboli określonych liczbami od 0 do 25. Symbole 21–25 różnią się od wcześniejszych jedynie typem wypełnienia. Dodatkowo istnieje możliwość użycia znaków o kodach ASCII od 32 do 255 z bieżącej czcionki (wszystkie symbole dostępne do użycia można wyświetlić poleceniem symbolTable z pakietu `fBasics`).
ps	Wielkość czcionki (w punktach)
srt	Kąt obrotu tekstu (w stopniach)

Predefiniowane typy linii (6 pierwszych) oraz przykładowe linie użytkownika (7. i 8.)



Symbole używane na rysunkach



par()

`par()` Graphical Parameters

Visual cheat sheet for some plot parameters in R. See `?par` for more information.

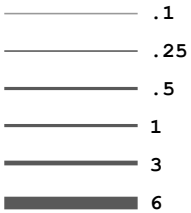
Symbol Styles

`pch` | Point Types

you can also use any character

`lty` | Line Types

`lwd` | Line Width



Figures Arrangement

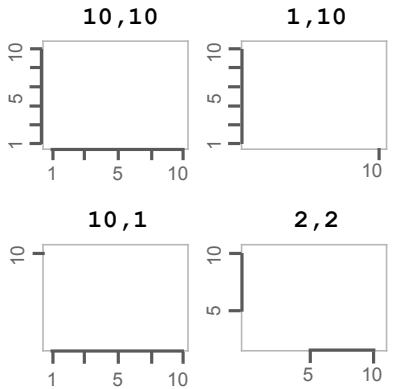
`mfrow` | Multiple Figures by Row

`2,3`

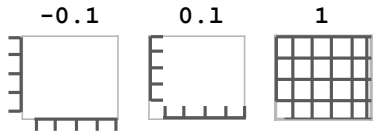
Also available `mfcol` for multiple figures by column

Axes

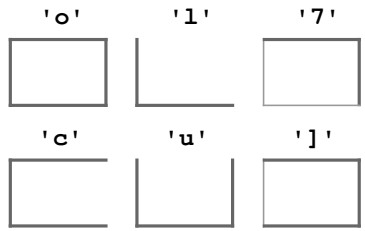
`lab` | Tick Placement



`tck` | Tick Length



`bty` | Box Type



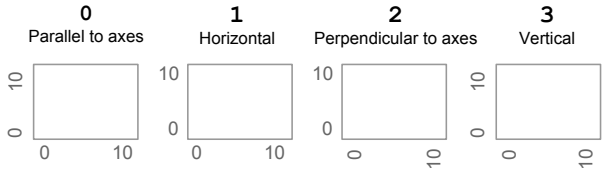
Text and Labels

`family, font` | Typeface and Font Style

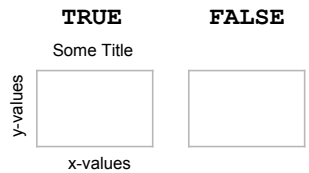
`family: mono`	family: serif	family: sans
`font: 1`	font: 1	font: 1
`family: mono`	**family: serif**	**family: sans**
`font: 2`	**font: 2**	**font: 2**
`family: mono`	*family: serif*	*family: sans*
`font: 3`	*font: 3*	*font: 3*
`family: mono`	***family: serif***	***family: sans***
`font: 4`	***font: 4***	***font: 4***

Also available: `font.main` (main title), `font.lab` (axis labels), `font.sub` (subtitle)

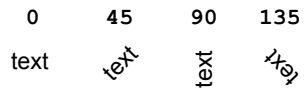
`las` | Label Orientation



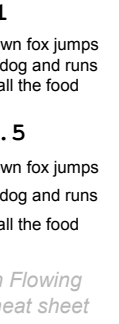
`ann` | Plot Annotation



`srt` | String Rotation



`lheight` | Line Height



Based on Flowing Data's cheat sheet

par()

How Big is Your Graph?

An R Cheat Sheet

Introduction

All functions that open a device for graphics will have **height** and **width** arguments to control the size of the graph and a **pointsize** argument to control the relative font size. In **knitr**, you control the size of the graph with the chunk options, **fig.width** and **fig.height**. This sheet will help you with calculating the size of the graph and various parts of the graph within R.

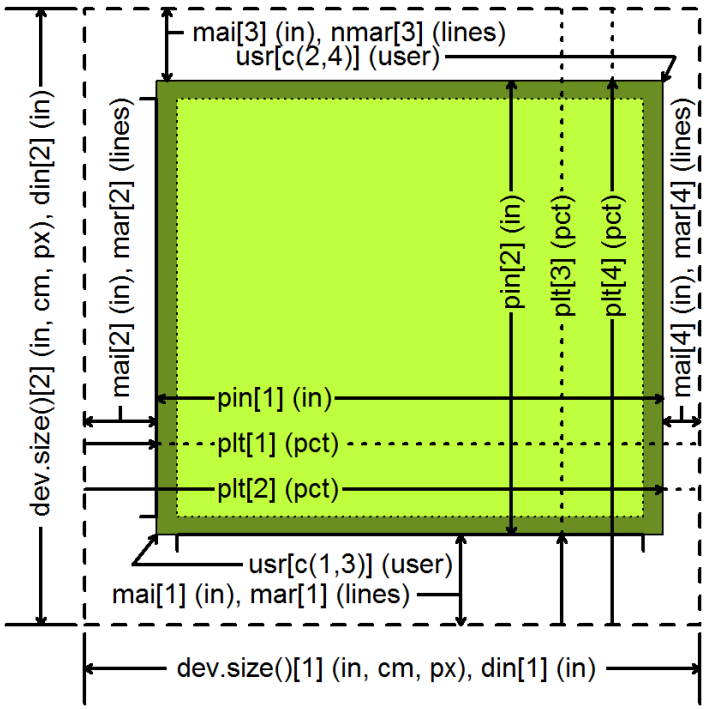
Your graphics device

dev.size() (width, height)
par("din") *(r.o.)* (width, height) in inches

Both the **dev.size** function and the **din** argument of **par** will tell you the size of the graphics device. The **dev.size** function will report the size in

1. inches (**units="in"**), the default
2. centimeters (**units="cm"**)
3. pixels (**units="px"**)

Like several other **par** arguments, **din** is read only *(r.o.)* meaning that you can ask its current value (**par("din")**) but you cannot change it (**par(din=c(5,7))** will fail).



Your plot margins

par("mai") (bottom, left, top, right) in inches
par("mar") (bottom, left, top, right) in lines

Margins provide you space for your axes, axis, labels, and titles.

A "line" is the amount of vertical space needed for a line of text.

If your graph has no axes or titles, you can remove the margins (and maximize the plotting region) with

```
par(mar=rep(0,4))
```

Your plotting region

par("pin") (width, height) in inches
par("plt") (left, right, bottom, top) in pct

The **pin** argument **par** gives you the size of the plotting region (the size of the device minus the size of the margins) in inches.

The **plt** argument gives you the percentage of the device from the left/bottom edge up to the left edge of the plotting region, the right edge, the bottom edge, and the top edge. The first and third values are equivalent to the percentage of space devoted to the left and bottom margins. Subtract the second and fourth values from 1 to get the percentage of space devoted to the right and top margins.

Your x-y coordinates

par("usr") (xmin, ymin, xmax, ymax)

Your x-y coordinates are the values you use when plotting your data. This normally is not the same as the values you specified with the **xlim** and **ylim** arguments in **plot**. By default, R adds an extra 4% to the plotting range (see the dark green region on the figure) so that points right up on the edges of your plot do not get partially clipped. You can override this by setting **xaxs="i"** and/or the **yaxs="i"** in **par**.

Run **par("usr")** to find the minimum X value, the maximum X value, the minimum Y value, and the maximum Y value. If you assign new values to **usr**, you will update the x-y coordinates to the new values.

Getting a square graph

```
par("pty")
```

You can produce a square graph manually by setting the width and height to the same value and setting the margins so that the sum of the top and bottom margins equal the sum of the left and right margins. But a much easier way is to specify **pty="s"**, which adjusts the margins so that the size of the plotting region is always square, even if you resize the graphics window.

Converting units

For many applications, you need to be able to translate user coordinates to pixels or inches. There are some cryptic shortcuts, but the simplest way is to get the range in user coordinates and measure the proportion of the graphics device devoted to the plotting region.

```
user.range <- par("usr")[c(2,4)] -
  par("usr")[c(1,3)]

region.pct <- par("plt")[c(2,4)] -
  par("plt")[c(1,3)]

region.px <-
  dev.size(units="px") * region.pct

px.per.xy <- region.px / user.range
```

To convert a horizontal or distance from the x-coordinate value to pixels, multiply by **px.per.xy[1]**. To convert a vertical distance, multiply by **region.px.per.xy[2]**. To convert a diagonal distance, you need to invoke Pyhthagoras.

```
a.px <- x.dist*px.per.xy[1]
b.px <- y.dist*px.per.xy[2]
c.px <- sqrt(a.px^2+b.px^2)
```

To rotate a string to match the slope of a line segment, you need to convert the distances to pixels, calculate the arctangent, and convert from radians to degrees.

```
segments(x0, y0, x1, y1)
delta.x <- (x1 – x0) * px.per.xy[1]
delta.y <- (y1 – y0) * px.per.xy[y]
angle.radians <- atan2(delta.y, delta.x)
angle.degrees <- angle.radians * 180 / pi
text(x1, y1, "TEXT", srt=angle.degrees)
```


Learn more at blog.pmean.com/cheatsheets • Updated: August 2017

par()

Panels

par("fig") (width, height) in pct
par("fin") (width, height) in inches

If you display multiple plots within a single graphics window (e.g., with the **mfrow** or **mfcol** arguments of **par** or with the **layout** function), then the **fig** and **fin** arguments will tell you the size of the current subplot window in percent or inches, respectively.

par("oma") (bottom, left, top, right) in lines
par("omd") (bottom, left, top, right) in pct
par("omi") (bottom, left, top, right) in inches

Each subplot will have margins specified by **mai** or **mar**, but no outer margin around the entire set of plots, unless you specify them using **oma**, **omd**, or **omi**. You can place text in the outer margins using the **mtext** function with the argument **outer=TRUE**.

par("mfg") (r, c) or (r, c, maxr, maxc)

The **mfg** argument of **par** will allow you to jump to a subplot in a particular row and column. If you query with **par("mfg")**, you will get the current row and column followed by the maximum row and column.

Character and string sizes

strheight()

The **strheight** functions will tell you the height of a specified string in inches (**units="inches"**), x-y user coordinates (**units="user"**) or as a percentage of the graphics device (**units="figure"**).

For a single line of text, **strheight** will give you the height of the letter "M". If you have a string with one of more linebreaks ("\n"), the **strheight** function will measure the height of the letter "M" plus the height of one or more additional lines. The height of a line is dependent on the line spacing, set by the **lheight** argument of **par**. The default line height (**lheight=1**), corresponding to single spaced lines, produces a line height roughly 1.5 times the height of "M".

strwidth()

The **strwidth** function will produce different widths to individual characters, representing the proportional spacing used by most fonts (a "W" using much more space than an "i"). For the width of a string, the **strwidth** function will sum up the lengths of the individual characters in the string.

par("cin") *(r.o.)* (width, height) in inches
par("csi") *(r.o.)* height in inches
par("cra") *(r.o.)* (width, height) in pixels
par("cxy") *(r.o.)* (width, height) in xy coordinates

The single value returned by the **csi** argument of **par** gives you the height of a line of text in inches. The second of the two values returned by **cin**, **cra**, and **cxy** gives you the height of a line, in inches, pixels, or xy (user) coordinates.

The first of the two values returned by the **cin**, **cra**, and **cxy** arguments to **par** gives you the approximate width of a single character, in inches, pixels, or xy (user) coordinates. The width, very slightly smaller than the actual width of the letter "W", is a rough estimate at best and ignores the variable with of individual letters.

These values are useful, however, in providing fast ratios of the relative sizes of the differing units of measure

px.per.in <- par("cra") / par("cin")
px.per.xy <- par("cra") / par("cxy")
xy.per.in <- par("cxy") / par("cin")

If your fonts are too big or too small

Fixing this takes a bit of trial and error.

1. Specify a larger/smaller value for the **pointsize** argument when you open your graphics device.

2. Trying opening your graphics device with different values for **height** and **width**. Fonts that look too big might be better proportioned in a larger graphics window.

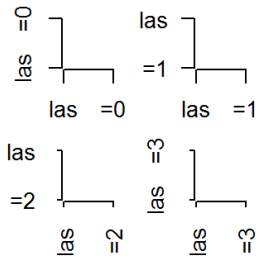
3. Use the **cex** argument to increase or decrease the relative size of your fonts.

If your axes don't fit

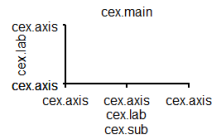
There are several possible solutions.

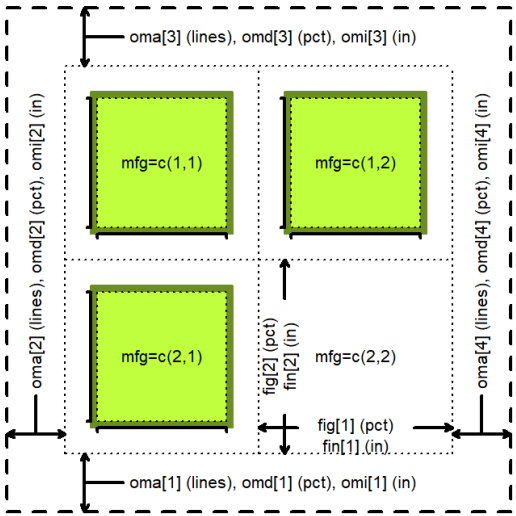
1. You can assign wider margins using the **mar** or **mai** argument in **par**.

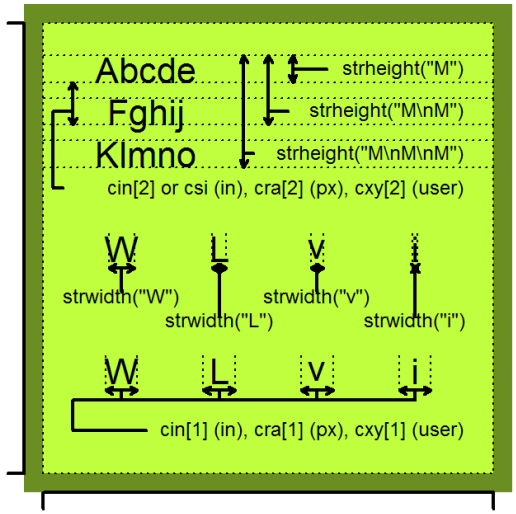
2. You can change the orientation of the axis labels with **las**. Choose among
   a. **las=0** both axis labels parallel
   b. **las=1** both axis labels horizontal
   c. **las=2** both axis labels perpendicular
   d. **las=3** both axis labels vertical.



3. change the relative size of the font
   a. **cex.axis** for the tick mark labels.
   b. **cex.lab** for **xlab** and **ylab**.
   c. **cex.main** for the main title
   d. **cex.sub** for the subtitle.



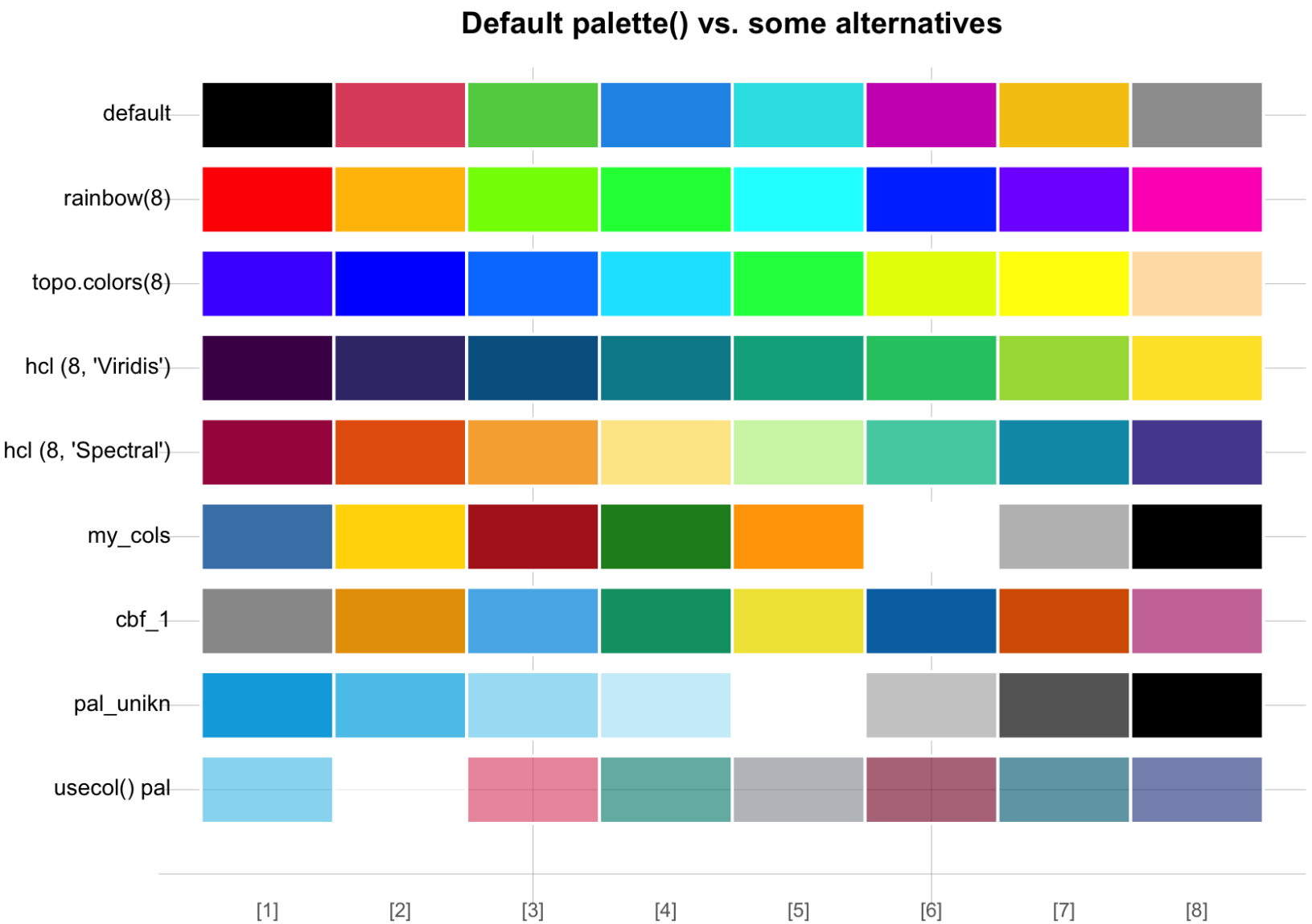




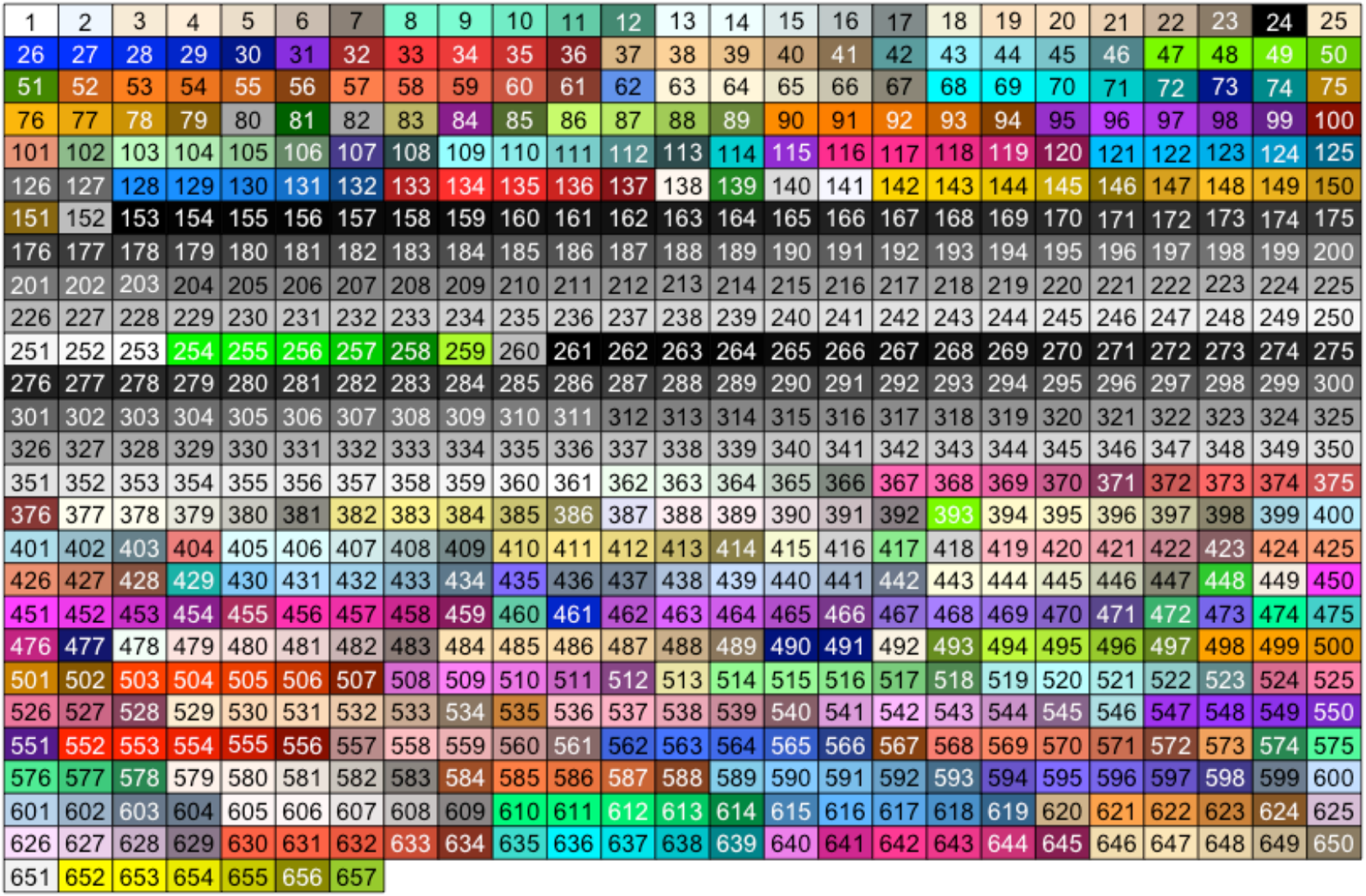
Kolory

Do kolorów można się odwoływać za pomocą nazwy lub korzystając z modelu RGB. Wszystkie 657 nazw kolorów możemy wyświetlić za pomocą funkcji colors(). Natomiast podając kolor w modelu RGB podajemy go w następującej postaci: '#RRGGBB', gdzie RR (czerwony), GG (zielony), BB (niebieski) to natężenie każdej ze składowych barwnych z przedziału (0, 255) zapisane w postaci liczby szesnastkowej. Istnieje również możliwość skorzystania z jednej z palet wbudowanych: rainbow, heat.colors, terrain.colors, topo.colors, cm.colors.

Kolory


Default palette() vs. some alternatives
default
rainbow(8)
topo.colors(8)
hcl (8, 'Viridis')
hcl (8, 'Spectral')
my_cols
cbf_1
pal_unikn
usecol() pal
[1]
[2]
[3]
[4]
[5]
[6]
[7]
[8]

Kolory


1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50
51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75
76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100
101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 119 120 121 122 123 124 125
126 127 128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 143 144 145 146 147 148 149 150
151 152 153 154 155 156 157 158 159 160 161 162 163 164 165 166 167 168 169 170 171 172 173 174 175
176 177 178 179 180 181 182 183 184 185 186 187 188 189 190 191 192 193 194 195 196 197 198 199 200
201 202 203 204 205 206 207 208 209 210 211 212 213 214 215 216 217 218 219 220 221 222 223 224 225
226 227 228 229 230 231 232 233 234 235 236 237 238 239 240 241 242 243 244 245 246 247 248 249 250
251 252 253 254 255 256 257 258 259 260 261 262 263 264 265 266 267 268 269 270 271 272 273 274 275
276 277 278 279 280 281 282 283 284 285 286 287 288 289 290 291 292 293 294 295 296 297 298 299 300
301 302 303 304 305 306 307 308 309 310 311 312 313 314 315 316 317 318 319 320 321 322 323 324 325
326 327 328 329 330 331 332 333 334 335 336 337 338 339 340 341 342 343 344 345 346 347 348 349 350
351 352 353 354 355 356 357 358 359 360 361 362 363 364 365 366 367 368 369 370 371 372 373 374 375
376 377 378 379 380 381 382 383 384 385 386 387 388 389 390 391 392 393 394 395 396 397 398 399 400
401 402 403 404 405 406 407 408 409 410 411 412 413 414 415 416 417 418 419 420 421 422 423 424 425
426 427 428 429 430 431 432 433 434 435 436 437 438 439 440 441 442 443 444 445 446 447 448 449 450
451 452 453 454 455 456 457 458 459 460 461 462 463 464 465 466 467 468 469 470 471 472 473 474 475
476 477 478 479 480 481 482 483 484 485 486 487 488 489 490 491 492 493 494 495 496 497 498 499 500
501 502 503 504 505 506 507 508 509 510 511 512 513 514 515 516 517 518 519 520 521 522 523 524 525
526 527 528 529 530 531 532 533 534 535 536 537 538 539 540 541 542 543 544 545 546 547 548 549 550
551 552 553 554 555 556 557 558 559 560 561 562 563 564 565 566 567 568 569 570 571 572 573 574 575
576 577 578 579 580 581 582 583 584 585 586 587 588 589 590 591 592 593 594 595 596 597 598 599 600
601 602 603 604 605 606 607 608 609 610 611 612 613 614 615 616 617 618 619 620 621 622 623 624 625
626 627 628 629 630 631 632 633 634 635 636 637 638 639 640 641 642 643 644 645 646 647 648 649 650
651 652 653 654 655 656 657

Interakcja z obrazkiem

Po stworzeniu wykresu możemy w niego również, w ograniczony sposób, ingerować za pomocą myszki. Najprostszą tego typu funkcją jest:

$$\text{locator(n = 512, type = 'n')}$$

gdzie n oznacza ilość punktów, a type ma analogiczne znaczenie do parametrów funkcji wysokiego poziomu.
Drugim użytecznym poleceniem jest

$$\text{identify(x, y, labels)}$$

które pozwala wskazywać punkty na wykresie. Na wykresie dodawana jest etykieta ze zbioru labels (lub numer punktu w zbiorze danych jeśli nie podano wektora etykiet), dla punktu najbliższego wskazania.

Histogram

Histogram – zbór przylegających prostokątów, których podstawy, równe rozpiętości przedziałów klasowych znajdują się na osi odciętych, a wysokości są liczebnościami przedziałów. Tworzymy go za pomocą polecenia:

hist(dane, breaks, probability = FALSE),

gdzie breaks może być liczbą określającą liczbę słupków (jest to tylko sugestia) lub wektorem zawierającym punkty, gdzie mają być słupki; probability – zamiast liczebności rysowane są częstości.

Histogram

Często do tak utworzonego histogramu dodaje się na dole wartości obserwacji za pomocą polecenia

rug(jitter(dane)),

co pozwala zaobserwować gdzie dokładnie są obserwacje w klasach. Polecenie jitter dodaje losowy szum do danych, co powoduje, że jeśli mamy kilka takich samych wartości, to nie są one przedstawione jako pojedynczy punkt, ale kilka punktów blisko siebie.

Histogram

Niestety histogram nie nadaje się do analizy zmiennych dyskretnych. W takiej sytuacji można wykorzystać wykres typu „zawieszony korzeń"(*ang. hanging rootogram*). Jest to diagram, na którym na osi pionowej znajdują się pierwiastki z zaobserwowanych częstości (aby nieco spłaszczyć wykres i uwypuklić również małe liczebności). Dodatkowo rysowana jest teoretyczna funkcja gęstości dopasowywanego rozkładu. Jednakże słupki są tak przesunięte aby dotykały dopasowywanej krzywej. Wykres taki został zaimplementowany w pakiecie `vcd` w funkcji rootogram.

Wykres pudełkowy

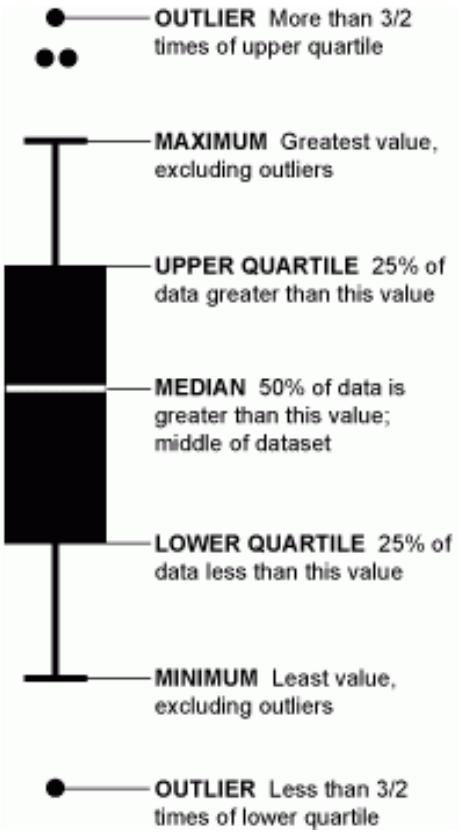
Wykres pudełkowy inaczej ramkowy (*ang. boxplot, box-and-whisker plot*). Tworzymy go odkładając na pionowej osi wartości niektórych parametrów rozkładu. Nad osią umieszczony jest prostokąt (pudełko), którego dolny bok jest wyznaczony przez pierwszy kwartyl, górny bok zaś przez trzeci kwartyl. Wysokość pudełka odpowiada wartości rozstępu ćwiartkowego. Wewnątrz prostokąta znajduje się pozioma linia, określająca wartość mediany.

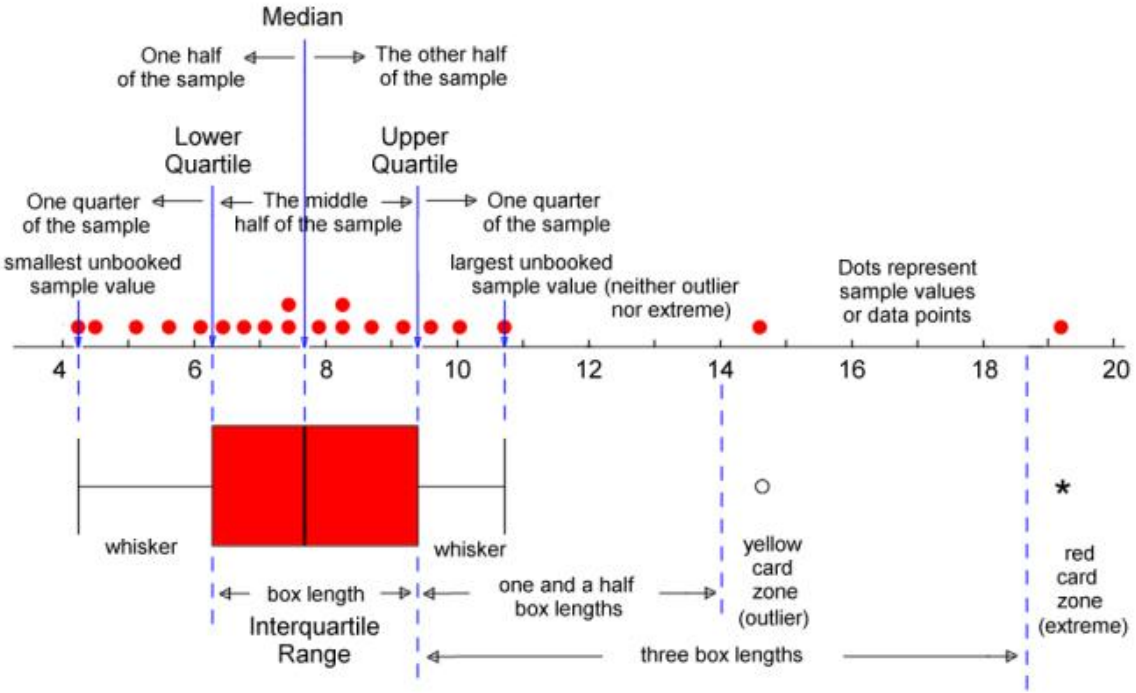
Wykres pudełkowy

Rysunek pudełka uzupełniamy od góry i od dołu odcinkami (wąsy). Dolny koniec dolnego odcinka wyznacza najmniejszą wartość w zbiorze, natomiast górny koniec górnego odcinka, to wartość największa. Końcowe wartości wąsów muszą spełniać dodatkowy warunek, a mianowicie dolny koniec nie może być mniejszy niż $Q_1 - 1{,}5 \cdot (Q_3 - Q_1)$, a górny większy niż $Q_3 + 1{,}5 \cdot (Q_3 - Q_1)$. Jeśli występują obserwacje spoza tego przedziału, to nanoszone są na wykres indywidualnie (są to tzw. obserwacje odstające (*ang. outlier*)).

boxplot(dane, formuła).

Wykres pudełkowy





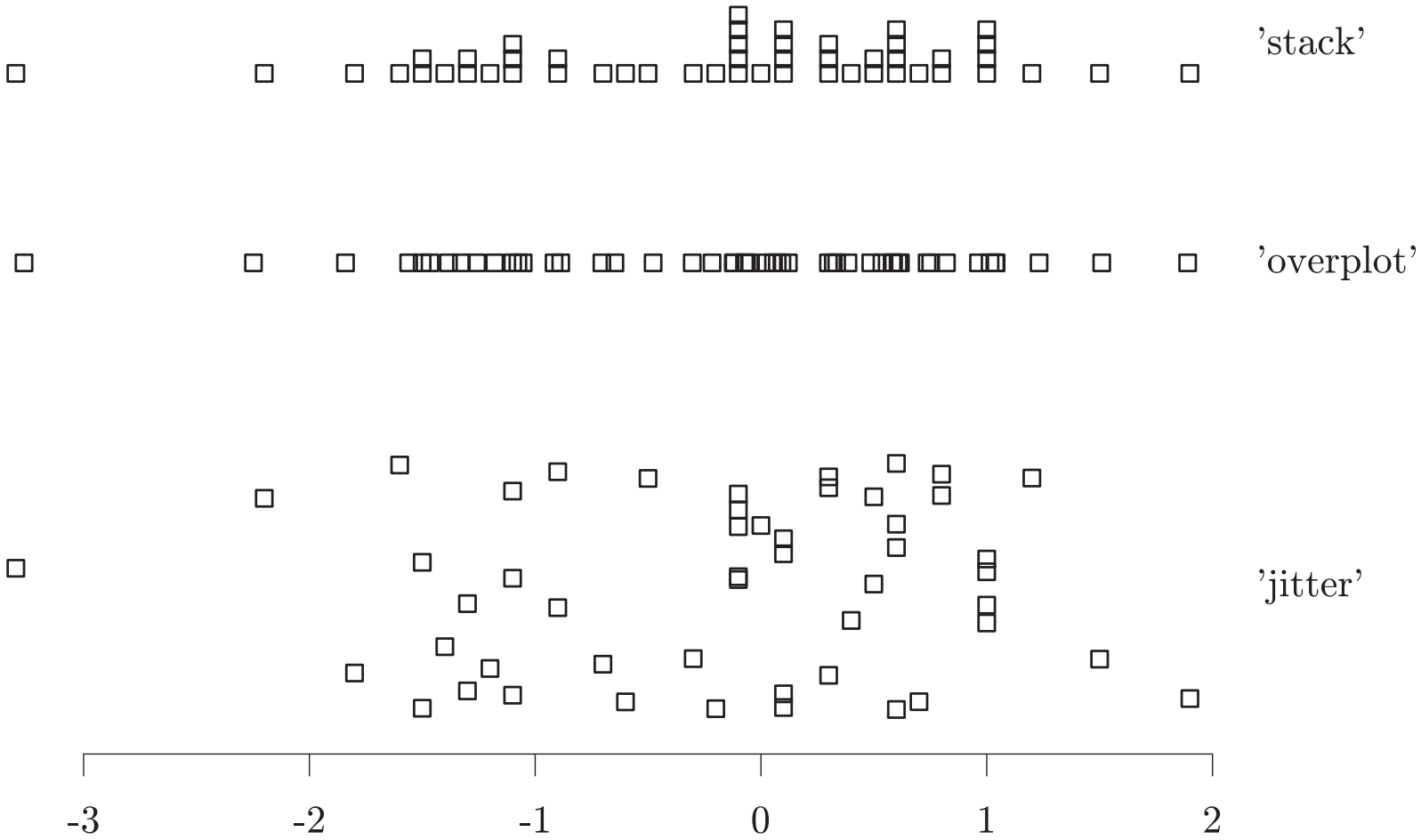
Wykres paskowy

Dla prób, w których istnieją powtarzające się wartości, używa się również wykresu paskowego (*ang. stripchart*). Na wykresie takim rysowane są wszystkie obserwacje w jednej poziomej linii, przy czym mamy możliwość wyboru metody postępowania w przypadku wartości powtarzających się.

stripchart(dane, method).

Parametr method określa sposób postępowania w przypadku obserwacji powtarzających się i może przyjmować wartość: 'stack', 'overplot' oraz 'jitter'.

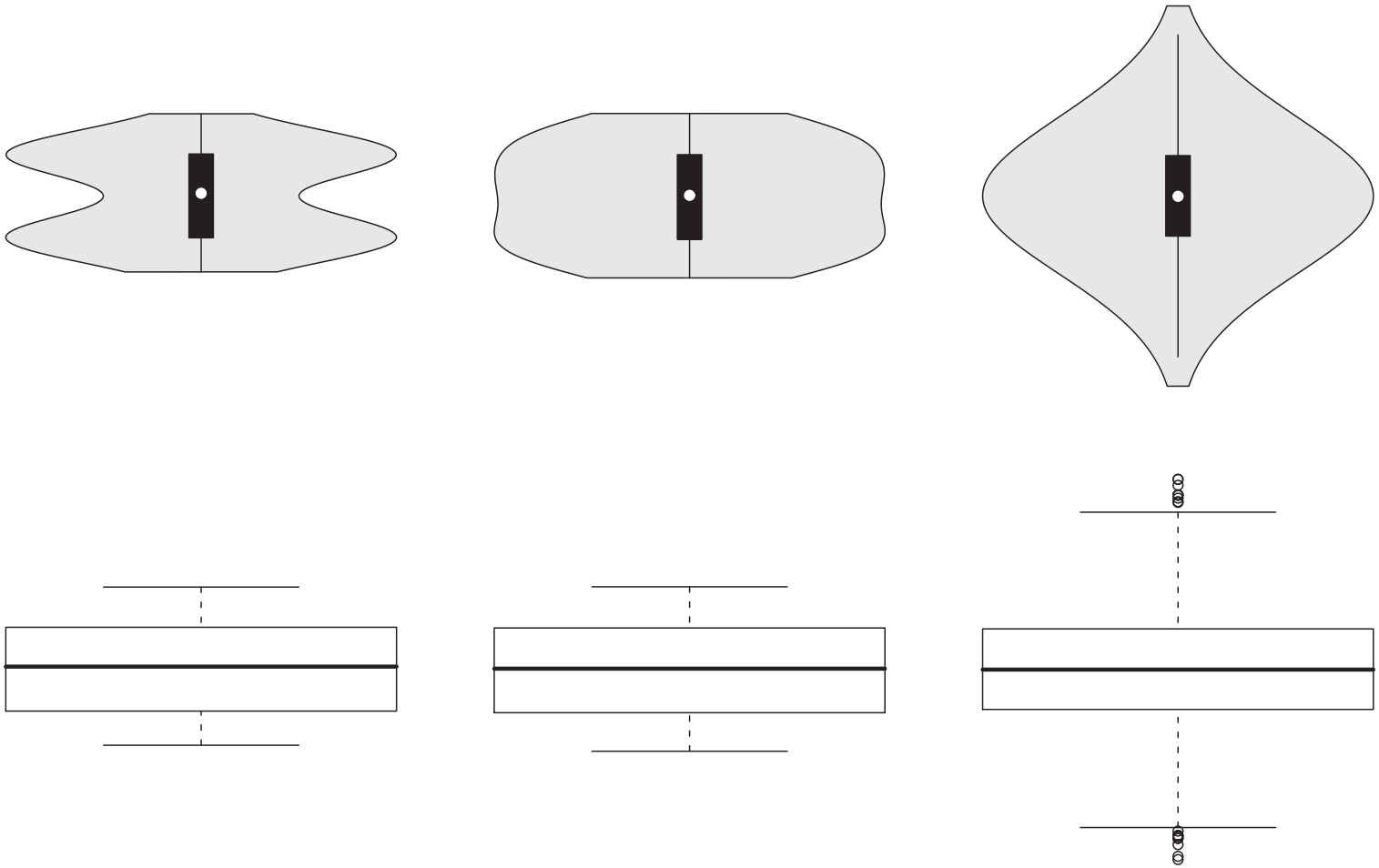
Wykres paskowy



Wykres skrzypcowy

Wykres skrzypcowy (*ang. violin plot*). Można go traktować jako wygładzoną wersję wykresu pudełkowego. Przydatny jest zwłaszcza w przypadku danych wielomodalnych. Jest to w zasadzie wykres pudełkowy, gdzie szerokość skrzypiec w punkcie x odpowiada natężeniu obserwacji o wartości cechy zbliżonej do x (estymator jądrowy gęstości). Funkcja vioplot znajduje się w pakiecie o tej samej nazwie.

Porównanie wykresu skrzypcowego i pudełkowego



Wykres typu „łodyga – liście”

Wykres typu „łodyga – liście” (*ang. stem and leaf plot, stemplot*). Jego celem jest połączenie dokładności polegającej na zachowaniu oryginalnych wartości poszczególnych obserwacji z zaletami histogramu. Jeśli np. będziemy rozpatrywać dane dotyczące wieku badanych, to wszystkie liczby można zapisać jako dziesiątki i jednostki (poza szczególnymi przypadkami osób ponad 100-letnich). Po prawej stronie od linii obrazującej dziesiątki będziemy zapisywali jednostki odpowiadające poszczególnym obserwacjom. W taki sposób otrzymujemy obrócony o 90° histogram. Dodatkowo jednak możemy z niego odczytać wartości poszczególnych obserwacji.

```
stem(dane)
```

Wykres typu „łodyga – liście”

3	0	457
21	1	00123344555566677789
18	2	00122333455667788
33	3	000111223344444555555566667899999
15	4	124555666777889
11	5	00344578999
7	6	1112235

Liczby po lewej stronie łodygi informują o liczebności klas. Cyfry ujęte w | | oznaczają liczbę dziesiątek, natomiast cyfry po prawej stronie łodygi to cyfry jedności. Zapisane w ten sposób liczby odczytujemy tak, że pierwsza cyfra uwidoczniona jest na łodydze, druga cyfra zaś tworzy liść (np. |4|5 odczytujemy jako cztery dziesiątki i 5 jedności, czyli 45). Z diagramu tego możemy odczytać np., że były 3 osoby w wieku poniżej 10 lat i miały one odpowiednio: 4, 5 oraz 7 lat; osób w wieku 10-19 lat było 21 i liczyły sobie one: 10, 10, 11, 12, 13, 13, 14, 14, 15, 15, 15, 15, 16, 16, 16, 16, 17, 17, 17, 18, 19 lat itd.

Wykres słupkowy

Wykres słupkowy (kolumnowy) (*ang. bar plot*) – używany raczej w przypadku danych jakościowych, przedstawia kolejne kategorie danych za pomocą słupków odpowiedniej wysokości. Może być poziomy lub pionowy (czasami występuje rozróżnienie, pionowy nazywany jest wtedy kolumnowym, a poziomy słupkowym). Czasami używa się również tzw. wykresu PARETO, gdzie słupki rysowane są od najwyższego do najniższego. Czasami dobrym rozwiązaniem może być zastąpienie wykresu słupkowego przez wykres kropkowy (*ang. dotchart*).

```
barplot(dane) #Wykres słupkowy
barplot(sort(dane, decreasing = TRUE)) #Wykres PARETO
dotchart(dane) #Wykres kropkowy
```

Wykres kołowy

Wykres kołowy (*ang. pie chart*). Wartości liczbowe są przedstawiane za pomocą wycinków koła. Wykres kołowy występuje w rozmaitych wariantach graficznych. Typowy jest wykres płaski, ale można też utworzyć trójwymiarowy „tort" lub „ser".

pie(dane)

Funkcja rysująca trójwymiarowy wykres kołowy (serowy) znajduje się w pakiecie `plotrix`.

pie3D(dane, radius = promień, explode = odstępy, labels = etykiety)

Wykres kwadratów

Wykres kwadratów (*ang. squareplot*). W pakiecie `UsingR` znajduje się bardzo ciekawy wykres do prezentacji liczebności kategorii danych. Powstał on na bazie podobnych wykresów prezentowanych w magazynie *New York Times*. W przeciwieństwie do wykresów słupkowych oraz kołowych widzimy dokładnie liczebność każdej kategorii.

squareplot(dane, kolory)