Midterm 3 cheat sheet

23 March 2025 20:41

ASIDE

`strptime(x, format, tz = "")` `Sys.time()`
Returns current data and time
"2025-03-23 21:38:09 PDT"

- format:
  - %y/%Y - year(auto/exact)
    - "20", "%y" = 2020
    - "20", "%Y" = 0020
  - %m - month
  - %d - day
  - %H - hour
  - %M - minute
  - %S - second
- `strptime("", "")`
  Returns current date
  2025-03-23 PDT
- `strptime("20-6-12", "%y-%m-%d")`
  2020-06-12 PDT

To query a string:
grep()
grepl()
gregexpr()

```
attach(modelcars) # to access
                  # variables directly
```

with(data, expr, ...)

Arguments:

1. data: A data frame or list that contains the variables to be used in the expression.
2. expr: The expression to evaluate using the variables from the data object.
3. ...: Additional arguments passed to the underlying functions within the expression (optional).

NOTES

If we are displaying numbers where zero is not relevant, then a dot chart is a better choice: in a dot chart it is mainly the position of the dot that is perceived.

ts()

- **Package**: Base R.
- **Explanation**: The ts() function creates time-series objects, which are particularly useful in analyzing time-dependent data, such as stock prices, climate data, or economic indicators.
- **Usage**:
  ts(data, start = NULL, end = NULL, frequency = 1)
  - data: A numeric vector or matrix representing the time-series data.
  - start: The time of the first observation (e.g., year and period).
  - end: The time of the last observation.
  - frequency: Number of observations per unit of time (e.g., 12 for monthly, 4 for quarterly).
- **Examples**:

```
# Example 1: Simple time series
sales <- ts(c(100, 120, 140, 160, 180), start = c(2025,
1), frequency = 12)
print(sales)

# Example 2: Matrix data
data_matrix <- ts(matrix(1:20, ncol = 2), start =
c(2025, 1), frequency = 4)
print(data_matrix)

# Example 3: Plotting
plot(sales, main = "Monthly Sales", ylab = "Sales", xlab
= "Time")
```

scan()

- **Package**: Base R.
- **Explanation**: The scan() function is used to read data (e.g., numbers, characters, or lines) from the console or a file into a vector. It's a flexible way to input raw data.
- **Usage**:
  scan(file = "", what = double(), ...)
  - file: The source of data (default is reading from the console).
  - what: The type of data (e.g., numeric(), character(), etc.).
  - ...: Additional arguments, like delimiters or skipping lines.
- **Examples**:

```
# Example 1: Reading numbers from the
console
numbers <- scan(what = numeric())
print(numbers)

# Example 2: Reading from a file
writeLines(c("1", "2", "3", "4"),
"example.txt")
data <- scan("example.txt", what =
numeric())
print(data)

# Example 3: Character input
names <- scan(what = character(), sep =
",")
print(names)
```

matrix()

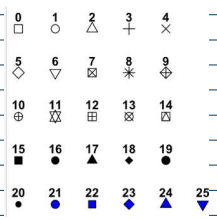
- **Package**: Base R (no external package needed).
- **Explanation**: The matrix() function is used to create matrices, which are two-dimensional arrays containing elements of the same type (numeric, character, etc.). Matrices have rows and columns and are ideal for performing mathematical operations or organizing structured data.
- **Usage**:
  matrix(data, nrow, ncol, byrow = FALSE, dimnames = NULL)
  - data: The elements to be included in the matrix.
  - nrow: Number of rows.
  - ncol: Number of columns.
  - byrow: Logical; if TRUE, fills the matrix by rows, otherwise by columns.
  - dimnames: Optional; names for the rows and columns.
- **Examples**:

```
# Example 1: Creating a numeric matrix
mat <- matrix(1:9, nrow = 3, ncol = 3)
print(mat)

# Example 2: Creating a matrix and filling it by row
mat_by_row <- matrix(1:9, nrow = 3, byrow = TRUE)
print(mat_by_row)

# Example 3: Adding row and column names
mat_named <- matrix(1:9, nrow = 3, dimnames = list(c("Row1",
"Row2", "Row3"),
                                    c("Col1", "Col2", "Col3")))
print(mat_named)

# Example 4: Transposing a matrix
transpose_mat <- t(mat)
print(transpose_mat)
```



plot()

Value	Description
"p"	**Points only**: Plots the data points as symbols without connecting them.
"l"	**Lines only**: Connects the data points with lines but doesn't plot symbols.
"b"	**Both points and lines**: Plots points and connects them with lines.
"c"	**Lines only, without the points at the ends**: A variation of "l".
"o"	**Overplotted points and lines**: Points and lines are plotted together, but points

Barplot()
-cex.names : x-axis names size
-cex.labels : tick and tick label size
-main : main title for selected plot only
-legend = T: check how to include legends
-beside = T : put row values of each column side by side. With beside off it just stacks
-ylim = c(0,10) : adjust vertical scale

dotchart()
-xlab : x-axis title
-pch = 16 : plotting character
-xlim : identical to ylim

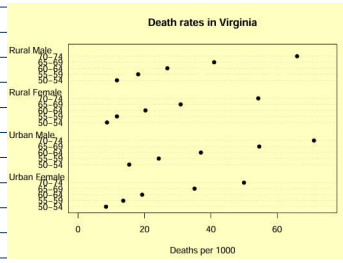
-main : main title for selected plot only
-legend = T: check how to include legends
-beside = T : put row values of each column side by side. With beside off it just stacks
-ylim = c(0,10) : adjust vertical scale

-xlim : identical to ylim

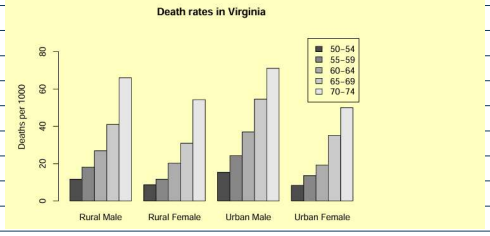
`"b"`	**Both points and lines:** Plots points and connects them with lines.
`"c"`	**Lines only, without the points at the ends:** A variation of `"l"`.
`"o"`	**Overplotted points and lines:** Points and lines are plotted together, but points overlap the lines.
`"h"`	**High-density vertical lines:** Draws vertical lines from each data point to the x-axis.
`"s"`	**Stair steps:** Connects data points with step-like lines that create a staircase pattern.
`"S"`	**Alternative stair steps:** Similar to `"s"`, but with a different alignment of steps.
`"n"`	**No plotting:** Suppresses plotting of points or lines. Useful when you want to set up a plot window without drawing anything.

```
VADeaths
##       Rural Male Rural Female Urban Male Urban Female
## 50-54       11.7          8.7       15.4          8.4
## 55-59       18.1         11.7       24.3         13.6
## 60-64       26.9         20.3       37.0         19.3
## 65-69       41.0         30.9       54.6         35.1
## 70-74       66.0         54.3       71.1         50.0
```

```
dotchart(VADeaths, xlim=c(0, 75),
         xlab="Deaths per 1000",
         main="Death rates in Virginia", pch=16)
```



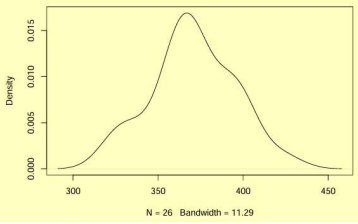

```
barplot(VADeaths, beside = TRUE, legend = TRUE, ylim = c(0, 90),
        ylab = "Deaths per 1000",
        main = "Death rates in Virginia")
```



Data: Escape times (in seconds) from a floating oil rig:

```
389 356 359 363 375 424 325 394
402 373 373 370 364 366 364 325
339 393 392 369 374 359 356 403
334 397
```

```
plot(density(escape), main=' ')
```



lines()
lty =

- 1 : Solid line (default).
- 2 : Dashed line.
- 3 : Dotted line.
- 4 : Dash-dotted line.
- 5 : Long-dashed line.
- 6 : Two-dash line.

When bars have equal width, the height of each bar is proportional to the number of observations in the corresponding interval.

If bars have different widths, then the *area* of the bar should be proportional to the count; in this way the height represents the density (i.e. the frequency per unit of x).

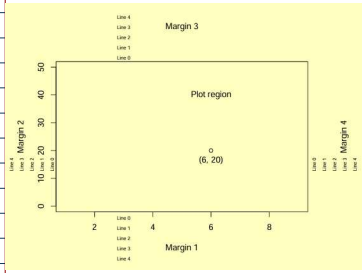
If you have n values of `x`, R, by default, divides the range into approximately $\log_2(n) + 1$ intervals, giving rise to that number of bars.

- Rounds up

The above rule (known as the "Sturges" rule) is not always satisfactory for very large values of n, giving too few bars.

Current research suggests that the number of bars should increase proportionally to $n^{1/3}$ rather than proportional to $\log_2(n)$.

Use `breaks = "scott"` or `breaks = "freedman-diaconis"`.



Outside the plot region are the margins, numbered clockwise from 1 to 4, starting at the bottom.

- `title(main, sub, xlab, ylab, ...)` **adds a main title, a subtitle, an x-axis label and/or a y-axis label**
- `mtext(text, side, line, ...)` **draws text in the margins**
- `axis(side, at, labels, ...)` **adds an axis to the plot**
- `box(...)` **adds a box around the plot region**

Several functions exist to add components to existing graphs:

```
points(x, y, ...)              # adds points
lines(x, y, ...)               # adds line segments
text(x, y, labels, ...)        # adds text into the graph
abline(a, b, ...)              # adds the line $y = a + bx$
abline(h = y, ...)             # adds a horizontal line
abline(v = x, ...)             # adds a vertical line
polygon(x, y, ...)             # adds a closed and possibly filled polygon
segments(x0, y0, x1, y1, ...) # draws line segments
arrows(x0, y0, x1, y1, ...)   # draws arrows
symbols(x, y, ...)             # draws circles, squares, thermometers, etc.
legend(x, y, legend, ...)      # draws a legend
```

Within the par(...) function:

- `mfrow=c(m, n)` **tells R to draw `m` rows and `n` columns of plots, rather than going to a new page for each plot.**
- `mfg=c(i, j)` **says to draw the figure in row `i` and column `j` next.**
- `ask=TRUE` **tells R to ask the user before erasing a plot to draw a new one.**

- `cex=1.5` **tells R to expand characters by this amount in the plot region. There are separate `cex.axis`, etc. parameters to control text in the margins.**
- `mar=c(side1, side2, side3, side4)` **sets the margins of the plot to the given numbers of lines of text on each side.**
- `oma=c(side1, side2, side3, side4)` **sets the outer margins (the region outside the array of plots).**
- `usr=c(x1, x2, y1, y2)` **sets the coordinate system within the plot with x and y coordinates on the given ranges.**

- lty : integer (0=blank, 1=solid (default), 2=dashed, 3=dotted, 4 =dotdash, 5=longdash, 6=twodash) or character strings.

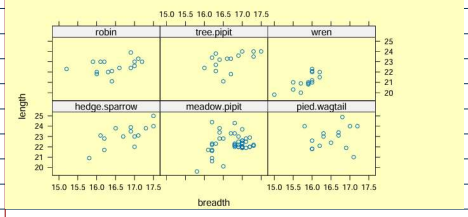
- Different version of dotplot:

```
library(lattice) # loads lattice package
dotplot(myfactor ~ mymeasurement|optionalfactor,
   data = mydata)
```

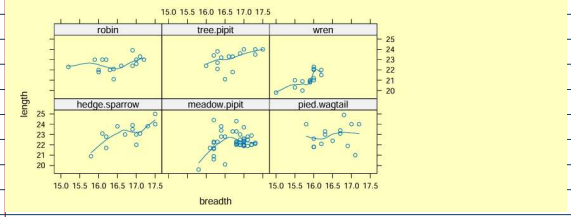
1. plot()
The plot() function is a versatile base R function for creating scatterplots, line plots, and other types of graphs.
Usage:

```
> head(cuckoos, 10)
   length breadth     species id
1    21.7    16.1 meadow.pipit 21
2    22.6    17.0 meadow.pipit 22
3    20.9    16.2 meadow.pipit 23
4    21.6    16.2 meadow.pipit 24
5    22.2    16.9 meadow.pipit 25
6    22.5    16.9 meadow.pipit 26
7    22.2    17.3 meadow.pipit 27
8    24.3    16.8 meadow.pipit 28
9    22.3    16.8 meadow.pipit 29
10   22.6    17.0 meadow.pipit 30
```

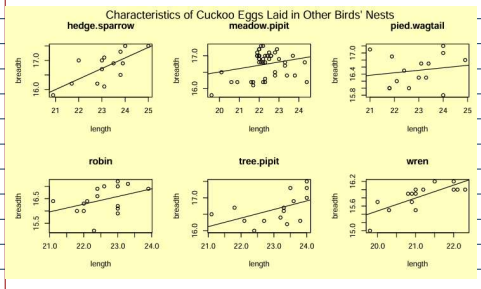
```
xyplot(length ~ breadth|species, data = cuckoos)
```




```
xyplot(length ~ breadth|species, data = cuckoos, type=c("p", "smooth"))
```




```
library(DAAG) # contains the cuckoos data frame
par(mfrow=c(2, 3))
for (i in levels(cuckoos$species)) {
      plot(breadth ~ length,
           data = subset(cuckoos, species == i))
      breadth.lm <- lm(breadth ~ length,
           data = subset(cuckoos, species==i))
      abline(breadth.lm)
      title(i)
}
mtext(side=3, line=-1.5,
    "Characteristics of Cuckoo Eggs Laid in Other Birds' Nests",
     outer=TRUE)  # outer=TRUE puts this text in the outer margin
```




```
 Nmissing <- sum(is.na(cfseal[, i]))
if(Nmissing > 0) title(sub=paste(Nmissing,  "NA's"))
```

R
plot(x, y = NULL, type = "p", main = NULL, sub = NULL, xlab = NULL, ylab = NULL,
xlim = NULL, ylim = NULL, log = "", asp = NA, col = NULL, pch = NULL, ...)
Key Parameters:
- x, y: Numeric vectors for plotting.
- type: Plot type ("p": points, "l": lines, "b": both, "h": vertical lines, etc.).
- main, sub: Main and subtitle text.
- xlab, ylab: Labels for x- and y-axes.
- xlim, ylim: Axis limits as numeric vectors (e.g., xlim = c(0, 10)).
- log: Logarithmic axis ("x", "y", or "xy").
- asp: Aspect ratio for y/x scaling.
- col, pch: Color and plotting symbol.
- ...: Additional graphical parameters like cex (size), lwd (line width), etc.

Example:
R
x <- 1:10
y <- x^2
plot(x, y, type = "b", main = "Scatterplot with Lines", xlab = "X", ylab = "Y", col = "blue", pch = 19)

2. xyplot()
The xyplot() function from the **lattice** package creates scatterplots with support for multi-panel layouts.
Usage:
R
xyplot(formula, data, panel = panel.xyplot, groups = NULL, type = "p", pch = 1, cex = 1,
col = NULL, auto.key = FALSE, layout = NULL, aspect = "fill", ...)
Key Parameters:
- formula: Formula like y ~ x | factor for conditioning.
- data: Data frame containing variables.
- panel: Custom panel function (e.g., panel.xyplot).
- groups: Factor defining groups for grouping data points.
- type: Plot type ("p" for points, "l" for lines, etc.).
- auto.key: Logical for adding a legend automatically.
- layout: Vector defining rows, columns of panels.
- aspect: Aspect ratio of panels ("fill", "iso" for equal scaling).

Example:
R
library(lattice)
xyplot(mpg ~ wt | cyl, data = mtcars, type = "p", layout = c(3, 1), auto.key = TRUE)

3. dotchart()
The dotchart() function creates dot plots, ideal for small datasets or comparing categories.
Usage:
R
dotchart(x, labels = NULL, groups = NULL, main = NULL, sub = NULL,
xlab = NULL, ylab = NULL, cex = par("cex"), pch = 21, col = par("fg"), ...)
Key Parameters:
- x: A numeric vector of values to plot.
- labels: Labels for each point.
- groups: Optional grouping variable.
- main, sub, xlab, ylab: Title, subtitle, and axis labels.
- cex: Character size for points and text.
- pch: Plotting symbol for points.
- col: Color for points.

Example:
R
data <- c(5, 7, 9)
dotchart(data, labels = c("A", "B", "C"), main = "Dotchart Example", xlab = "Values")

4. ggplot()
The ggplot() function from the **ggplot2** package creates layered, customizable visualizations.
Usage:
R
ggplot(data = NULL, mapping = aes(), ..., environment = parent.frame())
Key Parameters:
- data: A data frame containing the data.
- mapping: Aesthetic mappings (aes()), e.g., x, y, color, size, etc.
- ...: Additional layers (e.g., geom_point(), geom_line()).

Example:
R
library(ggplot2)
ggplot(mtcars, aes(x = wt, y = mpg, color = factor(cyl))) +
geom_point(size = 3) +
labs(title = "Weight vs MPG", x = "Weight", y = "MPG")

5. data.frame()
The data.frame() function creates a data frame.
Usage:
R
data.frame(..., row.names = NULL, check.rows = FALSE, check.names = TRUE,
stringsAsFactors = default.stringsAsFactors())
Key Parameters:
- ...: Named arguments for columns.
- row.names: Optional row names.
- check.rows, check.names: Logical for validation.
- stringsAsFactors: Logical; converts strings to factors.

Example:
R
df <- data.frame(Name = c("Alice", "Bob"), Age = c(25, 30), stringsAsFactors = FALSE)
print(df)

6. grep()
The grep() function searches for matches and returns the indices of matches.
Usage:
R
grep(pattern, x, ignore.case = FALSE, value = FALSE, fixed = FALSE, useBytes = FALSE, invert = FALSE)
Key Parameters:
- pattern: Regex or substring to search.
- x: Character vector to search in.
- value: Logical; return values instead of indices.

Example:
R
words <- c("apple", "banana", "cherry")
grep("a", words, value = TRUE) # Matches elements with "a"

7. grepl()
The grepl() function checks for pattern matches and returns a logical vector.
Usage:
R
grepl(pattern, x, ignore.case = FALSE, fixed = FALSE, useBytes = FALSE)
Example:
R
vec <- c("apple", "banana", "cherry")
grepl("a", vec) # TRUE for elements containing "a"

8. gregexpr()
The gregexpr() function locates all pattern matches in strings.
Usage:
R
gregexpr(pattern, text, ignore.case = FALSE, fixed = FALSE, useBytes = FALSE, perl = FALSE)
Example:
R
text <- "banana"
gregexpr("a", text) # Find all occurrences of "a"

9. barplot()
The barplot() function creates bar plots.
Usage:
R
barplot(height, width = 1, space = NULL, names.arg = NULL, beside = FALSE,
horiz = FALSE, col = NULL, main = NULL, xlab = NULL, ylab = NULL, ...)
Key Parameters:
- height: Numeric vector or matrix of bar heights.
- beside: Logical; grouped barplot.
- col, main, xlab, ylab: Customization options.

Example:
R
values <- c(4, 6, 8)
barplot(values, col = "blue", main = "Barplot Example", names.arg = c("A", "B", "C"))

10. hist()

The hist() function generates histograms.

Usage:

R

```
hist(x, breaks = "Sturges", freq = NULL, include.lowest = TRUE, right = TRUE,
    density = NULL, angle = 45, col = NULL, border = NULL, main = NULL, xlab = NULL, ...)
```

Key Parameters:

- x: Numeric vector for data.
- breaks: Number of bins or method for binning.
- freq: Logical; frequencies or densities.
- col, main, xlab: Customization options.

Example:

R

```
data <- rnorm(100)
hist(data, breaks = 10, col = "green", main = "Histogram Example")
```