Statistical Software Camp: Introduction to R

Day 2

August 25, 2009

1 Matrices

1.1 Creating, Indexing and Combining

- In addition to vectors, R can also handle **matrices**, two-dimensional array of numbers.
- The `matrix(x, N, K)` function reshapes a vector `x` into an `N` $\times$ `K` matrix.
- By default, a matrix is (counterintuitively) filled by columns. Setting the `byrow` argument to `TRUE` (or just `T`) change this behavior.

```
> a <- c(1,2,3,4)
> X <- matrix(a, 2, 2)
> X

     [,1] [,2]
[1,]    1    3
[2,]    2    4

> Y <- matrix(a, 2, 2, byrow=TRUE)
> Y # Notice how X and Y differ

     [,1] [,2]
[1,]    1    2
[2,]    3    4
```

- A matrix can also be built up from other vectors/matrices.
- The `cbind(x,y)` and `rbind(x,y)` functions combine `x` and `y` by columns and rows, respectively.

```
> b <- c(5,6,7,8)
> V <- cbind(a,b)
> V

     a b
[1,] 1 5
[2,] 2 6
[3,] 3 7
[4,] 4 8
```

```
> W <- rbind(a,b)
> W

  [,1] [,2] [,3] [,4]
a    1    2    3    4
b    5    6    7    8
```

- The `nrow(X)` function returns the number of rows in `X`; `ncol(X)` returns the number of columns.
- The `dim(X)` function reports both at the same time.
- Use square brackets and a comma (`[,]`) to specify rows and columns.

```
> nrow(V)

[1] 4

> ncol(V)

[1] 2

> dim(V)

[1] 4 2

> V[1,] # returns the first row

a b
1 5

> V[3,2] # returns the (3,2) element

b
7
```

1.2 Manipulating

- The addition operator (`+`) works on matrices in the same way as usual

```
> X + Y

     [,1] [,2]
[1,]    2    5
[2,]    5    8
```

- However, the multiplication (`*`) works element by element
- Instead, the `%*%` operator is used for usual matrix multiplication

```
> X * Y
```

```
     [,1] [,2]
[1,]    1    6
[2,]    6   16

> X %*% Y

     [,1] [,2]
[1,]   10   14
[2,]   14   20
```

- Other matrix operations include:
  - `t(X)` (transpose, X')
  - `det(X)` (determinant, $|X|$)
  - `solve(X)` (inversion, X^{-1})
  - `crossprod(X,Y)` (cross product, $X'Y$)
  - `kronecker(X,Y)` or `X %x% Y` (Kronecker product, $X \otimes Y$)

2 More on Data

2.1 Reading Data Created in Other Softwares

- Data come in various formats.
- The `foreign` package must be loaded to read files created in other softwares.
- `read.dta()` reads a Stata data file (`xxx.dta`)
- `read.spss()` reads a SPSS data file (`xxx.sav`)

```
> library(foreign)
> Humph <- read.dta("Humph.dta") # Data on African Countries by Humphreys
> BPCD <- read.spss("BPCD.sav") # British Parliament Constituency Data by Norris
```

- To read an Excel spreadsheet file (`xxx.xls`), the simplest way is to open it in Excel, save it as a .csv file and read it with `read.csv()`.
  (Note: The `gdata` package contains the `read.xls()` function, which can directly read an Excel file. However, you need to have the Perl language installed to get it to work.)

2.2 Looking at Data

- `names()` gives variable names in the data frame
- `nrow()`, `ncol()` and `dim()` work for data frames in the same as they do for matrices
- Also like matrices, data frames can be indexed by `[,]`
- Use `$` to access an individual variable
- `summary()` works for data frames too

```
> load("Africa.RData")  # Looks for the file in our working directory
> names(Africa)

[1] "Country" "GDP"     "GDP.pc"  "HDI"     "Region"

> dim(Africa)

[1] 52  5

> Africa[1:4,]  # Display the first four rows, and all columns

   Country    GDP GDP.pc   HDI          Region
1  Algeria 298448   8649 0.733 Northern Africa
2   Angola  91825   5463 0.446  Central Africa
3    Benin  12217   1507 0.437  Western Africa
4 Botswana  28454  18402 0.654 Southern Africa

> Africa[3:8, c(2,4)]  # Display rows 3-8, columns 2 and 4

    GDP   HDI
3 12217 0.437
4 28454 0.654
5 22132 0.370
6  5913 0.413
7 45777 0.532
8  4271 0.736

> Africa$HDI  # Displays only the HDI values

 [1] 0.733 0.446 0.437 0.654 0.370 0.413 0.532 0.736 0.384 0.388 0.561 0.432
[13] 0.411 0.516 0.708 0.642 0.483 0.406 0.677 0.502 0.553 0.456 0.374 0.521
[25] 0.549 0.331 0.818 0.533 0.437 0.380 0.550 0.804 0.646 0.384 0.650 0.374
[37] 0.470 0.548 0.452 0.654 0.499 0.843 0.336 0.674 0.526 0.547 0.467 0.512
[49] 0.766 0.505 0.434 0.513

> summary(Africa)

         Country        GDP              GDP.pc           HDI
 Algeria     : 1   Min.   :   616   Min.   :  500   Min.   :0.3310
 Angola      : 1   1st Qu.:  6178   1st Qu.: 1366   1st Qu.:0.4335
 Benin       : 1   Median : 19654   Median : 2162   Median :0.5125
 Botswana    : 1   Mean   : 61118   Mean   : 4616   Mean   :0.5296
 Burkina Faso: 1   3rd Qu.: 55461   3rd Qu.: 5569   3rd Qu.:0.6430
 Burundi     : 1   Max.   :703709   Max.   :23294   Max.   :0.8430
 (Other)     :46
            Region
 Central Africa : 9
 Eastern Africa :16
 Northern Africa: 6
 Southern Africa: 5
 Western Africa :16
```

2.3 Logical, Conditional Statements and Subsetting

- Logical operators: `<`, `<=`, `>`, `>=`, `==` and `!=`
- an object class of `logical` (i.e., `TRUE` or `FALSE`) is returned
- "==" versus "="
- Works with vectors too

```
> 4 > 3

[1] TRUE

> "Hello" == "hello"

[1] FALSE

> x <- 4 > 3
> x

[1] TRUE

> class(x)

[1] "logical"

> x <- c(3, 2, 1, -2, -1)
> x >= 2

[1]  TRUE  TRUE FALSE FALSE FALSE
```

- Combine logical statements with `&` (and) and `|` (or)

```
> x > 0 & x <= 2

[1] FALSE  TRUE  TRUE FALSE FALSE

> x > 0 | x <= 2

[1] TRUE TRUE TRUE TRUE TRUE
```

- Use logical statements to subset the data. Two important arguments of `subset()` are `subset` (for observations) and `select` (for variables)

```
> Africa[Africa$Region == "Northern Africa",]

   Country    GDP GDP.pc   HDI          Region
1  Algeria 298448   8649 0.733 Northern Africa
15   Egypt 423464   5643 0.708 Northern Africa
27   Libya  93402  15041 0.818 Northern Africa
33 Morocco 198785   6406 0.646 Northern Africa
45   Sudan 129447   3395 0.526 Northern Africa
49 Tunisia 107185  10269 0.766 Northern Africa
```

```
> subset(Africa, subset = (Region == "Northern Africa"))

   Country    GDP GDP.pc   HDI          Region
1  Algeria 298448   8649 0.733 Northern Africa
15   Egypt 423464   5643 0.708 Northern Africa
27   Libya  93402  15041 0.818 Northern Africa
33 Morocco 198785   6406 0.646 Northern Africa
45   Sudan 129447   3395 0.526 Northern Africa
49 Tunisia 107185  10269 0.766 Northern Africa

> Africa[(Africa$GDP.pc >= 8000) & (Africa$Region != "Northern Africa"), ]

             Country    GDP GDP.pc   HDI          Region
4           Botswana  28454  18402 0.654 Southern Africa
8         Cape Verde   4271   8481 0.736  Western Africa
16 Equatorial Guinea  26428  21316 0.642  Central Africa
32         Mauritius  19015  14954 0.804  Eastern Africa
35           Namibia  20100   9653 0.650 Southern Africa
42        Seychelles   1921  23294 0.843  Eastern Africa
44      South Africa 703709  14529 0.674 Southern Africa

> subset(Africa, subset = ((GDP.pc >= 8000) & (Region != "Northern Africa")))

             Country    GDP GDP.pc   HDI          Region
4           Botswana  28454  18402 0.654 Southern Africa
8         Cape Verde   4271   8481 0.736  Western Africa
16 Equatorial Guinea  26428  21316 0.642  Central Africa
32         Mauritius  19015  14954 0.804  Eastern Africa
35           Namibia  20100   9653 0.650 Southern Africa
42        Seychelles   1921  23294 0.843  Eastern Africa
44      South Africa 703709  14529 0.674 Southern Africa

> Africa.sub <- subset(Africa, select = c("Country", "GDP"))
> names(Africa.sub)

[1] "Country" "GDP"

> Africa.sub <- Africa[, c(1,2)]
> names(Africa.sub)

[1] "Country" "GDP"
```

- We can add up the number of `TRUE` statements using the `sum()` command

```
> x

[1]  3  2  1 -2 -1

> x >= 2
```

```
[1]  TRUE  TRUE FALSE FALSE FALSE

> sum(x>=2)  # Adds up the number of TRUE statements

[1] 2
```

- Conditional Statements evaluate a logical statement, then perform an action
- `ifelse(X, Y, Z)` performs `Y` if the statement `X` is true; performs `Z` if `X` is false

```
> regions <- c("Africa", "Africa", "Asia", "Asia", "Africa", "Middle East")
> regions

[1] "Africa"      "Africa"      "Asia"        "Asia"        "Africa"
[6] "Middle East"

> ifelse(regions == "Africa", "Yes", "No")

[1] "Yes" "Yes" "No"  "No"  "Yes" "No"
```

2.4 Working with Missing Values

- Missing values in your dataset can cause you trouble, and sometimes mess up your analysis completely.
- R uses `NA` ("not available") to indicate missing values.
- When a calculation involves `NA`s, it often behaves in a rather counterintuitive manner. Examples:

```
> x <- c(0, 1, NA)
> x > 0 # Returns "NA" for NA, rather than "FALSE"

[1] FALSE  TRUE    NA

> x == NA # Returns "NA" for everything

[1] NA NA NA
```

- The `is.na()` function tells whether a value is `NA`.
- Many R functions have an argument `na.rm=`. Setting it to `TRUE` allows the function to automatically drop `NA`s.

```
> is.na(x) # Shows which element is missing

[1] FALSE FALSE  TRUE

> mean(x) # Returns NA

[1] NA
```

```
> mean(x, na.rm=T) # Omits missing values

[1] 0.5
```

- Other special values similar to `NA` include `NaN` ("not a number") and `Inf`.

```
> 1/0

[1] Inf

> 0/0

[1] NaN

> is.na(c(1/0, 0/0, NA)) # NaN is treated as NA; Inf is not

[1] FALSE  TRUE  TRUE
```

2.5 Exporting Data Created in R

- Once finished with manipulating a data frame, you might want to save it into a non-R file.
- The `write.table(x, file)` function saves the object `x` as an ASCII file, with its name specified by the `file` argument.
- Alternatively, you can save it into a csv file by `write.csv(x, file)`

```
> write.table(Africa.sub, file="africa_sub.txt")
> write.csv(Africa.sub, file="africa_sub.csv")
```

3 More on Summary Statistics

- For a `numeric` object, we have `mean()` (mean), `median()` (median), `min()` (minimum), `max()` (maximum), `var` (variance), `sd()` (standard deviation)
- The function `summary()` will provide the mean, median, minimum, maximum, and quartiles of a `numeric` object and a table for a `factor` object (you can also use `table()` for this)
- Weighted mean, $\sum_{i=1}^{n} w_i x_i / \sum_{i=1}^{n} w_i$, can be computed using `weighted.mean()`

```
> mean(Africa$GDP.pc)  # Simple mean

[1] 4616.115

> Africa$pop <- Africa$GDP / Africa$GDP.pc  # Population of each country
> weighted.mean(Africa$GDP.pc, Africa$pop) # Weighted mean

[1] 3329.112

> median(Africa$GDP.pc)
```

```
[1] 2162.5

> summary(Africa$GDP.pc)

   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
    500    1366    2162    4616    5569   23290

> var(Africa$GDP.pc)  # Variance of GDP per capita

[1] 30692211

> sd(Africa$GDP.pc)  # the standard deviation

[1] 5540.055
```

- The function `quantile(X, P)` provides the sample quantiles of a numeric object `X` for each element of `P`
- The function `IQR()` returns the interquartile range.

```
> quantile(Africa$HDI)

    0%    25%    50%    75%   100%
0.3310 0.4335 0.5125 0.6430 0.8430

> quantile(Africa$HDI, c(0.1,0.25,0.50,0.75,0.9))  # Reports specified quantiles

   10%    25%    50%    75%    90%
0.3804 0.4335 0.5125 0.6430 0.7305

> IQR(Africa$HDI)  #Inter-Quartile Range

[1] 0.2095
```

- `tapply(X, INDEX, FUN)` applies the function `FUN` to `X` for each of the groups defined by `INDEX`
- Replace `FUN` with `mean, median, sd`, etc. to generate desired quantity.

```
> tapply(Africa$HDI, Africa$Region, mean)  # Calculates mean HDI for each region

 Central Africa  Eastern Africa Northern Africa Southern Africa  Western Africa
      0.5202222       0.5170000       0.6995000       0.6148000       0.4570000
```

`lines()`	Add a plot-line to a currently open figure.
	e.g. `lines(x,y)` where `x` and `y` are vectors of x- and y-coordinates.
`abline()`	Add a straight line.
	e.g. `abline(h=`τ`)` to place a horizontal line at height τ.
	e.g. `abline(v=`τ`)` to place a vertical line at point τ.
	e.g. `abline(a=`τ`, b=`λ`)` to place a line with intercept τ and slope λ.
`points()`	Add points.
	e.g. `points(x,y)` to place dots with `x` and `y` as vectors of x- and y-coordinates.
	e.g. `points(x,y, line=TRUE)` to connect the dots as a line.
`text()`	Add additional text to the plot.
	e.g. `text(x,y,"my text")` to write "my text" centered at coordinates x,y.

Table 1: Additional commands to append to an open graphic figure.

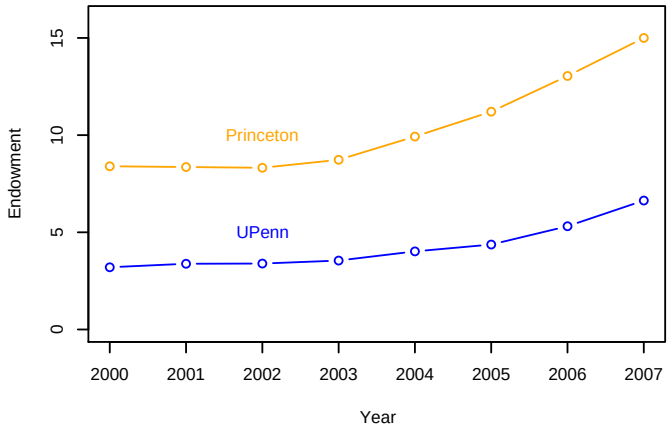
4 More on Graphs

4.1 Trend Plots

- The function `plot(X, Y)` will produce the **trend plot** where `X` is a vector for time and `Y` is a corresponding vector of values. You can set `type = "l"` or `type = "b"`.

- You can add lines, points, and texts to the graph too:

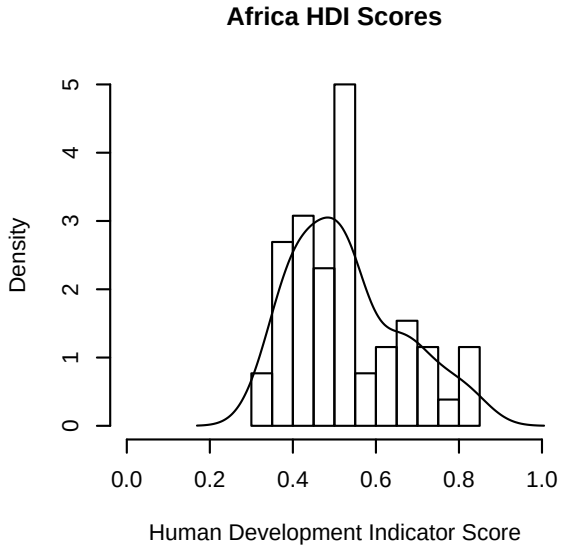
```
> Year <- 2000:2007
> # Princeton endowment 2000-2007
> Endowment <- c(8.398, 8.359, 8.320, 8.730, 9.928, 11.207, 13.045, 15.000)
> # UPenn endowment
> Endowment.penn <- c(3.201,3.382,3.393,3.547,4.019,4.370,5.313,6.635)
> par(cex = 0.6) # Use smaller font to look nicer in this handout
> plot(Year, Endowment, type = "b", ylim = c(0,16), col = "orange")
> lines(Year, Endowment.penn, type = "b", col = "blue") # Add the UPenn trend line
> text(2002, 10, "Princeton", col = "orange") # Added to the plot
> text(2002, 5, "UPenn", col = "blue")
```

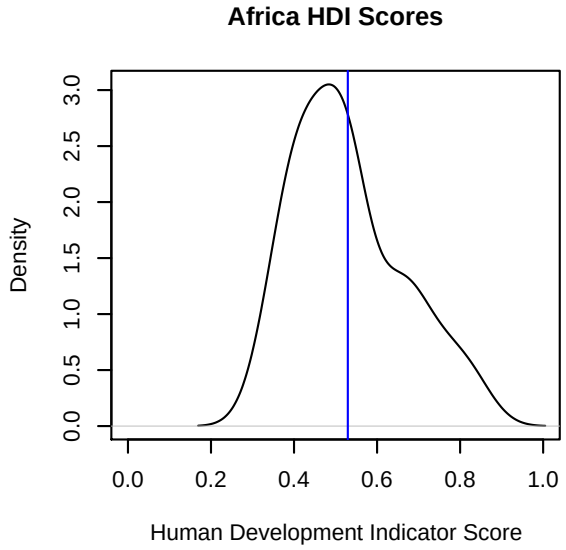


4.2 Histograms

- The function `hist(X, freq = FALSE)` will produce a **histogram**; the argument `breaks` will set the number of bins
- Setting `freq = TRUE` in `hist()` will produce a **frequency plot** rather than a histogram
- The function `density()` will calculate the density of a `numeric` object and can be used to draw smoothed histogram via `plot(density(x))` or `lines(density(x))`

```
> par(mfrow=c(1,2), cex = 0.65) # placing multiple plots in one graph
> hist(Africa$HDI, xlim = c(0, 1), freq = FALSE, main = "Africa HDI Scores",
+      breaks = 10, xlab = "Human Development Indicator Score")
> lines(density(Africa$HDI))  # Added to the histogram
> plot(density(Africa$HDI), xlim = c(0, 1),  #looks roughly normal
+     xlab = "Human Development Indicator Score", main = "Africa HDI Scores")
> avg <- mean(Africa$HDI)
> abline(v = avg, col = "blue")  # Adds a vertical line at avg
```

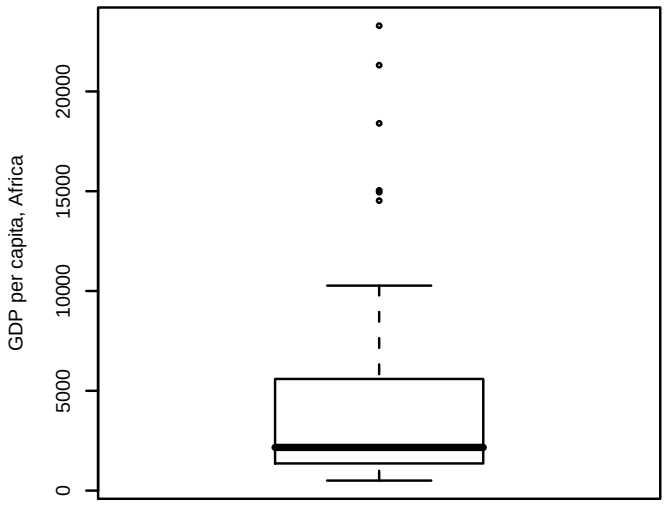




4.3 Boxplots

- The function `boxplot()` will produce a **boxplot** figure

```
> par(cex = 0.5)
> boxplot(Africa$GDP.pc, ylab = "GDP per capita, Africa")
```



4.4 Printing and Saving Graphs

There are a few ways to print and save the graphs you create in **R**.

- In the window of your graph (if you are a Mac user, make sure your graphic window rather than the R console is selected), you can click File: Save as: PDF... or File: Print....
- You can also right-click on a figure in **R** and copy the image (if you are a Mac user, you need to highlight the graph and type Apple+C to copy it). Then paste that image into Microsoft Word or any other document.
- You can also do it via a command by using `pdf()` before your plotting commands.

  ```
  > pdf(file="histogram.pdf", height=3, width=5) # height and width are in inches
  ```

  After your plotting commands, you need to type

  ```
  > dev.off()
  ```

- A variety of options available through `par()`; e.g., `par(cex = X)` where `X` is a magnification factor for text. Numbers bigger than 1 increase the font size. see `?par` for more.
- Setting `mfrow=c(X, Y)` or `mfcol=c(X, Y)` in `par()` will allow you to place multiple (`X` $\times$ `Y`) plots in one graph