

# Data frames, Lists, Matrices

AND the Apply Family of Functions

2012 Summer Olympics

# 2012 Olympic Athlete's

Craig Bloodworth at the Information Lab made this data explorer for the Guardian. It includes data on all athletes competing in the Olympics

<http://www.guardian.co.uk/sport/datablog/2012/jul/27/london-olympic-athletes-full-list>



We download the data as a csv file and read it into R with read.csv

# Reading data into R

- Many data sets are stored in text files.
- The easiest way to read these into R is using either the `read.table` or `read.csv` function, both of which return a data frame.
- There are quite a few options that can be changed. Some of the important ones are
  - file - name or URL
  - header - are column names at the top of the file?
  - sep - what divides elements of the table
  - na.strings - symbol for missing values, like 9999
  - Skip - number of lines at the top of the file to ignore

What's the relationship between GDP, population, and number of Olympic medals?

# Country-level data



**Many Eyes**

An experiment brought to you by IBM Research and the IBM Cognos software group

[Visualizations](#)  
[Data sets](#)  
[Comments](#)  
[Topic centers](#)

**Explore**

[Visualizations](#)  
[Data sets](#)  
[Comments](#)  
[Topic centers](#)

**Participate**

[Create a visualization](#)  
[Upload a data set](#)  
[Create a topic center](#)  
[Register](#)

**Learn more**

[Quick start](#)  
[Visualization types](#)  
[Data format and style](#)  
[About Many Eyes](#)  
[FAQ](#)  
[Blog](#)

**Contact us**

[Contact](#)

[Log in](#)



Visualizations

Search

## Data sets : Olympic2012withGdp

Uploaded by: [Mimosha](#) Created at: Sep 3 2012

Data source: Unknown

Description: gold 3pt silver 2pt bronze 1pt

View as text

	ISO	Gold/medals	Silver/medals	Bronze/medals	total/medals	total weight/points	GDP
1	ABW	0	0	0	0	0	2,456,000,
2	AFG	0	0	1	1	1	20,343,461
3	AGO	0	0	0	0	0	100,990,00
4	ALB	0	0	0	0	0	12,959,563
5	AND	0	0	0	0	0	3,491,000,
6	ARE	0	0	0	0	0	360,245,00
7	ARG	1	1	2	4	7	445,989,00

<http://www-958.ibm.com/software/data/cognos/manyeyes/datasets/olympic2012withgdp/versions/1.txt>

```
> ctry = read.csv("http://www-958.ibm.com/software/
data/cognos/manyeyes/datasets/olympic2012withgdp/
versions/1.txt",
skip = 1, sep = "\t", header = FALSE,
colClasses = c("character", rep("numeric", 5),
               rep("character", 3)))
```

```
> head(ctry)
```

	V1	V2	V3	V4	V5	V6	V7	V8	V9
1	ABW	0	0	0	0	0	2,456,000,000.00	108,000	22740.7407
2	AFG	0	0	1	1	1	20,343,461,030.00	34,385,000	591.6377
3	AGO	0	0	0	0	0	100,990,000,000.00	19,082,000	5292.4222
4	ALB	0	0	0	0	0	12,959,563,902.00	3,205,000	4043.5457
5	AND	0	0	0	0	0	3,491,000,000.00	84,864	41136.4065
6	ARE	0	0	0	0	0	360,245,000,000.00	7,512,000	47955.9372

Need to:

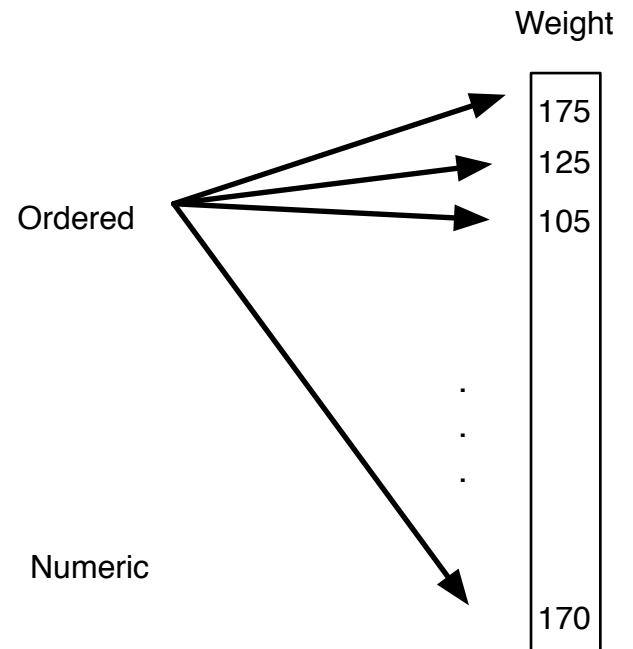
Clean up the GDP and POP by removing “,” and converting to numeric;

Adding latitude and longitude for each country

# Review Data Structures

# Review: Vector

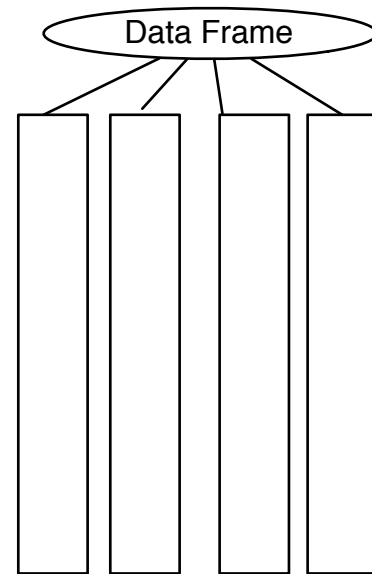
- *Ordered* container of literals
- Elements must be *same type*





# Review: Data Frame

- *Ordered* container of vectors
- Vectors must all be the *same length*
- Vectors can be *different types*



# Wireless Data

- There are 5 wireless access points in a building
- A laptop emits a signal and the strength of the signal is recorded as each access point.
- Also recorded is the location of the laptop in the building.
- Measurements are taken at 254 locations

```
w = read.table("http://  
www.stanford.edu/~vcs/StatData/  
wireless.txt", header=TRUE)
```

# Helpful Functions for finding information out about the data frame

```
> class(w)
```

```
[1] "data.frame"
```

```
> names(w)
```

```
[1] "x" "y" "S1" "S2" "S3" "S4" "S5"
```

```
> dim(w)
```

```
[1] 259 7
```

> **head(w)**

	x	y	S1	S2	S3	S4	S5
1	225.0	144	-92	-78	-49	-92	-92
2	0.0	144	-75	-92	-87	-47	-92
3	111.0	132	-92	-92	-80	-70	-65
4	110.7	132	-87	-92	-67	-67	-62
5	105.0	132	-92	-92	-79	-66	-61
6	99.0	132	-92	-92	-79	-70	-61

```
> summary(w$S1)
```

Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
-92.00	-90.00	-80.00	-77.41	-71.00	-33.00

```
> w[w$x > 200, c("S2")]
```

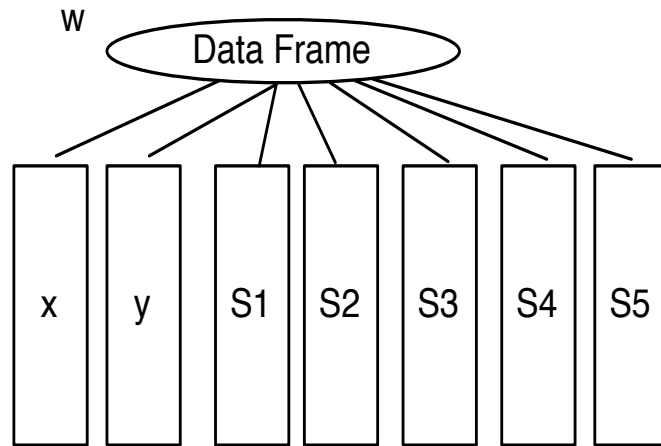
```
[1] -78 -74 -35 -43 -46 -43 -41 -48 -68 -71 -67 -70 -58 -61  
[15] -64 -73 -71 -67 -68 -61 -60 -57 -58 -56 -48 -51 -68 -54  
[29] -51 -48 -46 -35 -67
```

We subset rows and columns of data frames

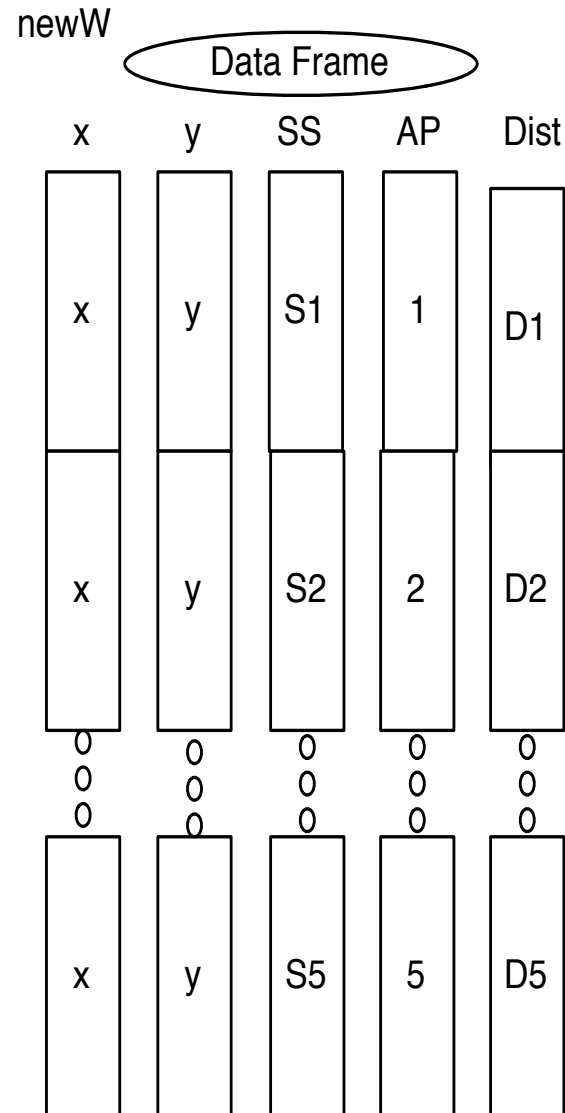
We subset by **position**, **exclusion**, **logical**, **name**, and **all**

Reformatting the data frame

# Revise the Structure



Data Frame **ap** has five rows and 2 columns (x, y) with the locations of the five access points



```
X = rep(w$x, 5)
```

```
Y = rep(w$y, 5)
```

```
AP = rep(1:5, each = nrow(w))
```

```
SS = c(w$S1, w$S2, w$S3, w$S4, w$S5)
```

```
D1 = sqrt((w$x - ap[1, "x"] )^2 + (w$y - ap[1, "y"] )^2 )
```

```
D2 = sqrt((w$x - ap[2, "x"] )^2 + (w$y - ap[2, "y"] )^2 )
```

```
D3 = sqrt((w$x - ap[3, "x"] )^2 + (w$y - ap[3, "y"] )^2 )
```

```
D4 = sqrt((w$x - ap[4, "x"] )^2 + (w$y - ap[4, "y"] )^2 )
```

```
D5 = sqrt((w$x - ap[5, "x"] )^2 + (w$y - ap[5, "y"] )^2 )
```



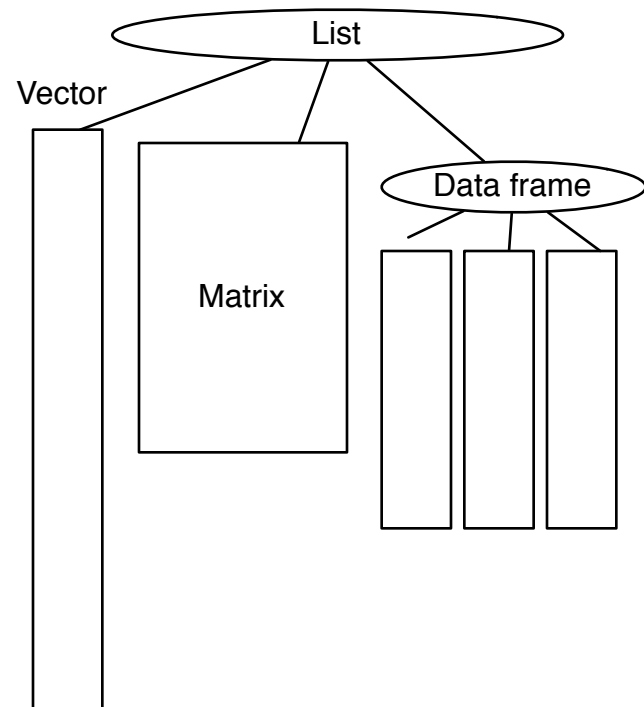
## Continued..

```
Dist = c(D1, D2, D3, D4, D5)
newW = data.frame(x = X, y = Y,
                  AP, SS, Dist)
```

# Lists

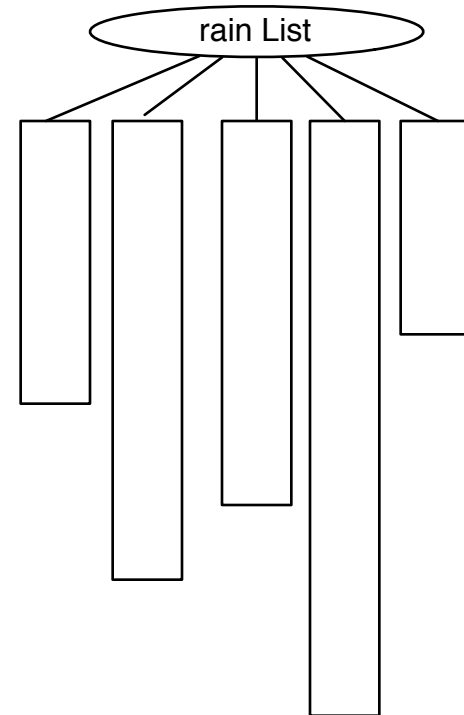
# Review

- Data frames are actually a special kind of *list*.
- Unlike a data frame each element in a list can have a different length.
- Actually, each element can be either a list, data frame, vector, matrix, ...



# Rainfall

- Daily rainfall collected at 5 weather stations
- **rain** is a list of length 5
  - One element for each station
  - Each element is a numeric vector of rain measurements
  - Stations not in operation for the same length of time



```
load(url(  
"http://www.stanford.edu/~vcs/StatData/  
rainfallCO.rda"))
```

```
> class(rain)
```

```
[1] "list"
```

```
> length(rain)
```

```
[1] 5
```

```
> names(rain)
```

```
[1] "st050183" "st050263" "st050712"  
"st050843" "st050945"
```

# Indexing lists

- Lists can be indexed by name, using \$.
- Or by [[ ]] with position or name

```
> class(rain$st050183)
```

```
[1] "numeric"
```

```
> length(rain$st050183)
```

```
[1] 9878
```

```
> head(rain$st050183)
```

```
[1] 0 10 11 1 0 0
```

```
> class(rain[["st050945"]])
```

```
[1] "numeric"
```

```
> length(rain[["st050945"]])
```

```
[1] 3692
```

```
> head(rain[[5]])
```

```
[1] 0 0 1 0 26 0
```

# Indexing lists

- Lists can also be indexed like vectors, using []. The result will be another list.

```
> class(rain["st050183"])
```

```
[1] "list"
```

```
> length(rain["st050183"])
```

```
[1] 1
```

# Indexing lists

- To extract individual elements of a list, enclose the index in `[[ ]]`. The result will be an object of the same type as the element of the list. You can only use one value in `[[ ]]`.

```
> class(rain[[1]])
```

```
[1] "numeric"
```

```
> head(rain[[1]])
```

```
[1] 0 10 11 1 0 0
```



# Matrices and Arrays

# Matrices and Arrays

- Rectangular collection of elements
- Dimensions are two, three, or more
- Homogeneous primitive elements (e.g., all numeric or all character)

# Arrays – matrices in higher dimensions

```
> x = array(1:30, c(4, 3, 2))
```

```
> x
```

```
, , 1
```

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

```
[1,]    1     5     9
```

```
[2,]    2     6    10
```

```
[3,]    3     7    11
```

```
[4,]    4     8    12
```

```
, , 2
```

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

```
[1,]   13   17   21
```

```
[2,]   14   18   22
```

```
[3,]   15   19   23
```

```
[4,]   16   20   24
```

- The integers 1, 2, ... , 30 are arranged in a 3-dimensional array
- The array has:
  - 4 rows
  - 3 columns
  - 2 panels

```
> x[1:2, 3, 2]
```

```
[1] 21 22
```

```
> x[, 2, 1]
```

```
[1] 5 6 7 8
```

```
> x[3:4, c(3, 1), 1]
```

```
    [,1] [,2]
```

```
[1,]   11    3
```

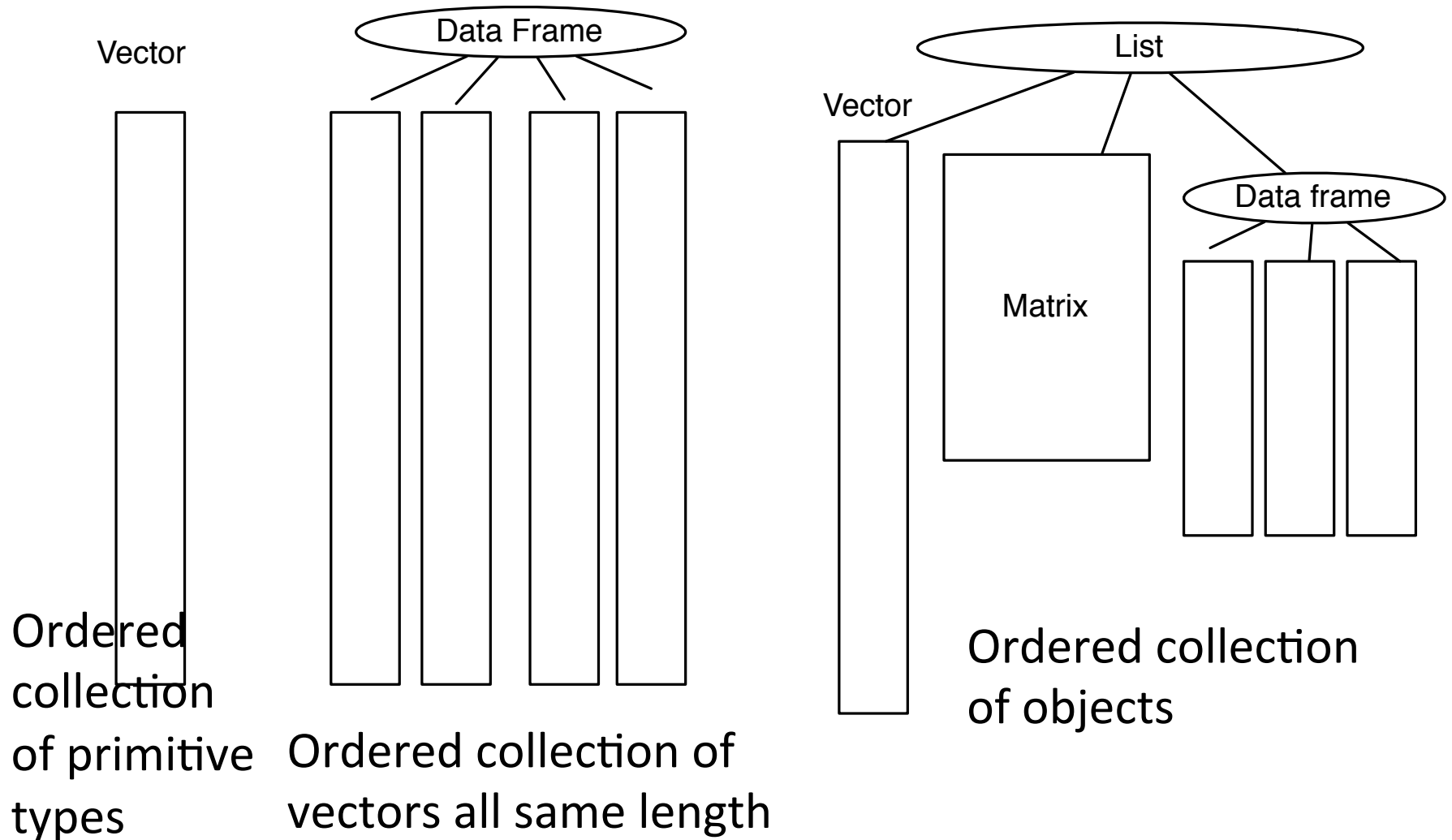
```
[2,]   12    4
```

# Summary of Data Structures

# Types of structures

- To summarize, the data structures we have encountered so far are:
  - vector
  - data frame
  - list
  - matrix

# Vector – Data frame - List



# Indexing data structures

- Vectors: [index]

> x[1:10]

> x[-3]

> x[x>3]

- Data frames: [rowindex, colindex] and \$name

> family\$weight

> family[, 3:4]

> family[famiy\$height > 70, 2] Returns a vector  
when possible,  
unless use  
drop = FALSE

- Note: both \$ can index only *one* element.

# Indexing data structures

- Lists: `$name`, `[index]`, `[[index]]`

```
> rain$stationname
```

```
> rain[1:2]
```

```
> rain[[1]]
```

- Matrices: `[rowindex, colindex]`

```
> m[1, 2]
```

```
> m[1:2, ]
```

```
> m[, "a", drop = TRUE]
```

- Note: `[[ ]]` can index only *one* element. Also we can index a matrix as if it were a vector



# Apply Functions

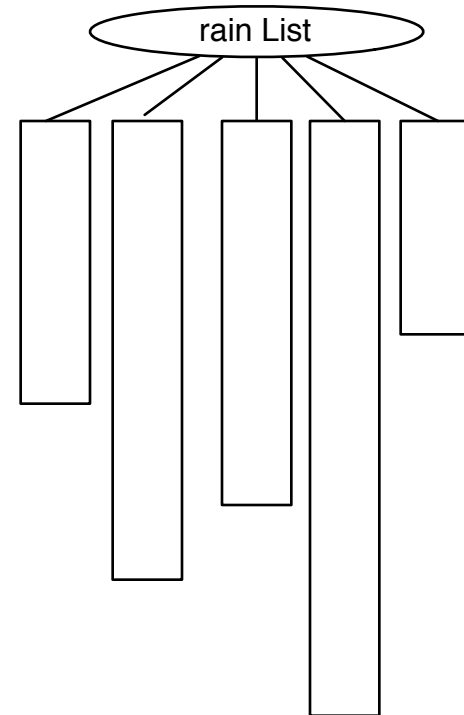
# The “Apply” Function

- Sometimes we want an operation to be applied to each element of a list, to each vector in a data frame, or to individual dimensions of a matrix
- R provides the *apply* mechanism to do this.
- There are several apply functions:
  - `sapply()` and `lapply()` for lists and data frames
  - `apply()` for matrices
  - `tapply()` for “tables”, i.e. ragged arrays as vectors

- With these functions we can avoid looping, and instead write code that is meaningful in a statistical setting.
- For example with our list of rainfall data, each element represents the measurements taken at a particular weather station and when we think about studying the average rainfall at each station - we don't think in terms of loops.

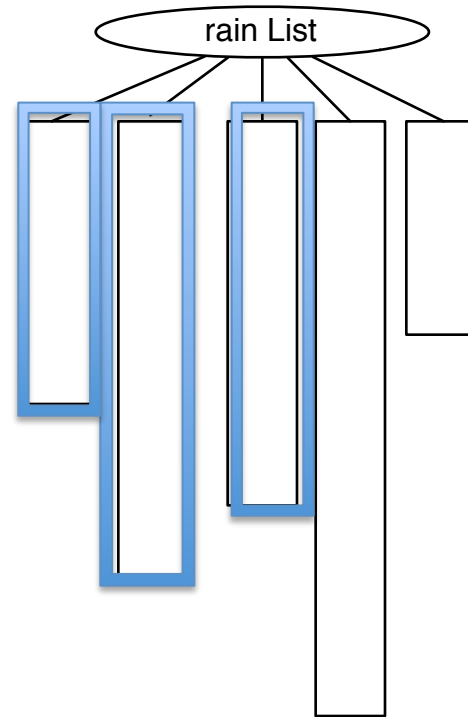
# Rainfall

- Daily rainfall collected at 5 weather stations
- **rain** is a list of length 5
  - One element for each station
  - Each element is a numeric vector of rain measurements
  - Stations not in operation for the same length of time



# Rainfall

- Apply the mean function to each element of **rain**
- Finds: average precipitation of first station,
- Second station,
- Third station,
- Etc.



# lapply() and sapply()

- The **lapply** and **sapply** both apply a specified function to each element of a list.
- The former returns a list object and the latter a vector when possible.

# Mean rainfall at each station

```
> lapply(rain, mean)
```

```
$st050183
```

```
[1] 6.631707
```

```
$st050263
```

```
[1] 3.798993
```

```
$st050712
```

```
[1] 5.102299
```

```
$st050843
```

```
[1] 6.084607
```

```
$st050945
```

```
[1] 4.549296
```

```
> sapply(rain, mean)
```

```
st050183 st050263 st050712
```

```
6.631707 3.798993 5.102299
```

```
st050843 st050945
```

```
6.084607 4.549296
```

# Additional arguments

```
> lapply(rain, mean, na.rm = TRUE,  
         trim = 0.1)
```

```
$st050183  
[1] 2.393978
```

```
$st050263  
[1] 0.9875949
```

```
$st050712  
[1] 0.7895235
```

```
$st050843  
[1] 1.238481
```

```
$st050945  
[1] 0.7366283
```

```
> args(lapply)  
function (X, FUN, ...)
```

X takes the list object  
FUN is the function to  
apply to each element  
in X

... allows any number  
of arguments to be  
passed to FUN



# tapply()

- This function is useful to apply a function to subsets of a vector.

```
> x
```

```
[1] 1 2 3 4 5 6 7 8 9 10
```

```
> v
```

```
[1] 1 1 1 0 0 0 1 1 1 0
```

```
> tapply(x, v, mean)
```

```
0      1
```

```
6.25  5.00
```

```
> tapply(x, v, median)
```

```
0      1
```

```
5.5  5.0
```

Now it's your turn to try it out

# Rainfall data

- Maximum rainfall at each station

```
sapply(rain, max)
```

- 99<sup>th</sup> percentile of rainfall at each station

```
sapply(rain, quantile, probs = 0.99)
```

# Rainfall data

- **day** is a list with the same structure as **rain**
- Check that the number of recordings at each station matches the number of days recorded at the corresponding station

```
all(sapply(rain, length)  
    == sapply(day, length))
```

# Rainfall data

- Number of years each station is in operation

```
Year = lapply(day, floor)
```

```
Uyear = lapply(Year, unique)
```

```
OpYear = sapply(Uyear, length)
```

For any one station:

```
length(unique(floor(day[[1]])))
```

```
sapply(day, length(unique(floor(?))) )
```

```
sapply(day, function(x) length(unique(floor(x))))
```

- Proportion of days it rained at each station

```
sapply(rain, function(x)  
      sum(x > 0)/length(x) )
```

# Matrices and Arrays

# apply()

- **apply(x, 1, sum)** for the matrix x, the sum function is applied across the columns so that the row dimension (i.e. dim 1) is preserved.

```
> x
```

	[ , 1 ]	[ , 2 ]	[ , 3 ]
[ 1 , ]	1	3	5
[ 2 , ]	2	4	6

```
> apply(x, 1, sum)
```

```
[1] 9 12
```



# apply()

- **apply(x, 1, min)** for the matrix x, the min function is applied down the rows so that the column dimension (i.e. dim 2) is preserved.

```
> x
```

	[ , 1 ]	[ , 2 ]	[ , 3 ]
[ 1 , ]	1	3	5
[ 2 , ]	2	4	6

```
> apply(x, 2, min)
```

```
[1] 1 3 5
```