R

- Interactive language for expressing statistically-oriented computations
  - Mathematics
  - Data cleaning and processing
  - Exploratory data analysis
  - Statistical methodology
  - Simulation
  - Graphics
- Language for creating new functionality

You can launch the R application by clicking on the R icon in you “dock”

This will give you a prompt. R has a simple user interface:

- Invoke a computation with an expression
- Pass the expression off to the computer to evaluate
- Return a value or output of the expression

Operations

You can use R as a calculator:

```
> 2+3^2
[1] 11
```

You can also perform function style computations:

```
>   rnorm(4)
[1]  0.2351375  1.0895481 -1.4124248 -1.8859612
```

The format for calling a function to perform a computation:

functionName(argument, . . . , argument)

Arguments can also be identified by name

rnorm(4, sd = 5)

generates 10 random values from a normal distribution with mean 0 and standard deviation 5.

Examples

- Single Expression:

```
rnorm(10, sd = 5)
```

- Compound Expression:

```
mean( rnorm(10, sd = 5) )
```

- Ill-formed Expression:

```
rnorm(10; sd = 5)
Error: syntax error
```

Output and Assignment

- When we evaluate a command, R prints the results to the screen as output.
- How do we get to use it again, e.g. as input to other commands?

```
> x = sample(10, size = 4)}
> x
[1] 6 7 3 9
```

- The above assignment statement puts the result from the command **sample(10, size = 4)** into a variable named **x**.
- We can see the value of x via the simple expression
  x
- This is the sames as
  "Evaluate x and print the result"
  i.e. provide the current contents of the variable x.
- In R we can also use the left arrow to assign a value to a variable:
  x $<-$ **rnorm(10)**

Variables

Variables have a name and a value.

To access the value we use the name.

```
 n = 10
x = rnorm(n)
sum(x) / n
```

Variable Names must follow some rules:

- May not start with a digit or underscore
- May contain numbers, characters (upper and lower case), and some punctuation, period . and underscore _ are okay,
  but most other other are not, e.g. commas, quotation marks, and # are not.
- Case-sensitive, so **x** and **X** are different.

Managing Sessions and Variables

We can manage our variables with R functions

- List all variables
  objects()
- Remove one or more variables
  rm(x, y)
- Save variables for future use
  save(x, y, z, file = “myfile.rda”)
- Restore variables
  load(“myfile.rda”)
- Alternatively, an entire workspace may also be saved, and it will be automatically loaded when you start R up again.
  $>$q()
  Save workspace image? [y/n/c]:
  But it keeps EVERYTHING!
- Other times we load an R package that may have data and functions
  load(”Day1”)

Managing Sessions and Variables

- Keeping track of the code you write:
  - To see the code that you have executed in the R session
    history(max.show = Inf)
    savehistory(”myCode.R”)
  - To evaluate code that you have written and saved in a file
    source(”myRevisedCode.R”)
- To get help with functions, begin your session by starting the help browser:
  help.start()
  then when you need specific help on a function you can ask for it as follows
  help(plot)
  or
  help.search(”plot”)
  or
  ?plot
  or
  apropos(”plot”)

R Data Types

R supports a few basic data types: integer, numeric, logical, character/string, factor, and complex

Logical – binary, two possible values
represented by TRUE and FALSE

```
> x = c(3, 7, 1, 2)
> x > 2
[1]  TRUE  TRUE FALSE FALSE

> x == 2
[1] FALSE FALSE FALSE  TRUE

> !(x < 3)
[1]  TRUE  TRUE FALSE FALSE

> which(x > 2)
[1] 1 2
```

Character vectors

Character/string – each element in the vector is a string of one or more characters.

Built in character vectors are **letters** and **LETTERS** which provide the 26 lower (and upper) case letters, respecitively.

```
> y = c("a", "bc", "def")

> length(y)
[1] 3

> nchar(y)
[1] 1 2 3

> y == "a"
[1]  TRUE FALSE FALSE

> y == "b"
[1] FALSE FALSE FALSE
```

Factor

A **factor**- type vector contains a set of numeric codes with character-valued levels.

Example - a family of two girls (1) and four boys (0),

```
> kids = factor(c(1,0,1,0,0,0), levels = c(0, 1),
                labels = c("boy", "girl"))
> kids
[1] girl boy  girl boy  boy  boy
Levels: boy girl

> class(kids)
[1] "factor"

> mode(kids)
[1] "numeric"
```

Regardless of the levels/labels of the factor, the numeric storage is an integer with 1 corresponding to the first level (in alph-order).

```
> kids + 1
[1] NA NA NA NA NA NA

> as.numeric(kids)
[1] 2 1 2 1 1 1

> 1 + as.numeric(kids)
[1] 3 2 3 2 2 2

> kids2 = factor(c("boy","girl","boy","girl","boy","boy"))
> kids2
[1] boy  girl boy  girl boy  boy
Levels: boy girl

> as.numeric(kids2)
[1] 1 2 1 2 1 1
```

Functions to Provide Information about Vectors

- **length(x)** - number of elements in a vector or list
- Aggregator functions - **sum**, **mean**, **range**, **min**, **max**, **summary**, **table**, **cut**, ...
- **class(x)** – returns the type of an object.
- **is.logical(x)** – tells us whether the object is a logical type. There is also **is.numeric**, **is.character**, **is.integer**
- **is.null** – determines whether an object is empty, i.e. has no content. 'NULL' is used mainly to represent the lists with zero length, and is often returned by expressions and functions whose value is undefined.
- **is.na** – NA represents a value that is not available.

```
> x
[1]  3  1 NA

> is.na(x)
[1] FALSE FALSE  TRUE
```

- **as.numeric(x)** – we use the as-type functions to coerce objects from one type (e.g. logical) to another, in this case numeric. There are several of these functions, including **as.integer**, **as.character**, **as.logical**, **as.POSIXct**.

Missing Values

- NA is different from 99999 or -8, which are numeric values that have special meaning in a particular context
- NA is a recognized element in R
  x = c(3, 1, NA)
- Functions have special actions when they encounter values of NA, and may have arguments to control the handling of NAs.

```
> mean(x)
[1] NA

> mean(x,na.rm = TRUE)
[1] 2
```

- Note that NA is not a character value. In facti, it has meaning for character vectors too.
  y = c("A", "d", NA, "ab", "NA")
  Notice that the two uses, NA and ""NA" mean very different things. The first is an NA value and the second is a character string.
- **na.omit()**, **na.exclude()**, and **na.fail()** are for dealling manually with NAs in a dataset.

Vectors, Lists, and Data Frames

Vector – a collection of **ordered homogeneous** elements.

Vectors can be created within R using some of the following functions:

- **c()** function to catenate individual values together
- **:** the infix function to create a sequence of numbers **1:10**
- **seq()** to create more complex sequences
- **rep()** to create replicates of values
- **sort()** and **order()** are useful for ordering elements in a vector,
  sort(x, decreasing = TRUE)

Examples of c()

- **c(3, 2, 1)** – a vector of three numeric elements $3, 2, 1$ **in that order**.
- **c(2, 3, 1)** – a **different** vector of the **same** three numeric elements,
  but with a **different** ordering.
- **x = c(bob = 3, alice = 2, John = 1)** – elements can have names.
  names(x)
- Vectors can also consist of characters, logicals, factors, integers provided they are all of the same type.

Examples of rep()

- **rep(3, 2)** – a vector of two threes
- The arguments of **rep()** can be vectors

```
> x = c(7, 1, 3)

> rep(x, 2)
[1] 7 1 3 7 1 3

> rep(x, c(3, 2, 1))
[1] 7 7 7 1 1 3
```

Examples of **seq()**

There are several ways to call the **seq** function. Here are three popular ones:
seq(from, to)
seq(from, to, by =)
seq(from, to, length =)

Consider arguments of **from = 1**, **to = 19**, **by = 2**, and **length = 10**. Evaluate the following function calls to **seq()** with the various combinations and ordering of arguments (**named** and **unnamed**).

```
> seq(1, 19, by = 2)
 [1]  1  3  5  7  9 11 13 15 17 19

> seq(1, 19, length = 2)
 [1]  1  19
```

Data Frames

A collection of possibly *heterogeneous* vector elements of the *same length*. The elements of a data frame can be numeric vectors, factor vectors, and logical vectors, but they must all be of the same length.

- **names(sfo.origin)** – returns the element names of the vectors:

```
> names(sfo.origin)
 [1] "ActualElapsedTime" "AirTime"
 [3] "ArrDelay"          "ArrHour"
 [5] "ArrMin"            "ArrTime"          ...
```

- **class(sfo.origin)** – a ”data.frame”
- **dim(sfo.origin)** – as a rectangular list, the data frame supports some matrix features: 140587 40

Vectors in a Data Frame

```
length(sfo.origin$AirTime)
[1] 140587

> class(sfo.origin$Cancelled)
[1] "logical"

> summary(sfo.origin$ArrDelay)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
 -71.00  -10.00   -1.00   10.62   15.00 1036.00 3132.00

> table(sfo.origin$UniqueCarrier)
   9E     AA     AQ     AS     B6     CO     DL     EV     F9     FL
    0  12008      0   5062   1923   4801   4575      0   1686    951 ...
```

Operators

- **Vectorized** – Most functions work on vectors in a **vectorized** fashion, i.e. they work on all the elements without the need for an explicit loop over the elements.

```
> mean(sfo.origin$ArrDelay)
[1] NA

> mean(sfo.origin$ArrDelay, na.rm = TRUE)
[1] 10.62057
```

- **Element-wise** – Most operators work **element-wise**, i.e. they operate on each element.

```
sfo.origin$ArrDelay
 [1]  69  46 120  36   5 266  75  -1 -16  -7 ...

> sfo.origin$ArrDelay/60)
 [1]  1.15000000  0.76666667  2.00000000  0.60000000
 [5]  0.08333333  4.43333333  1.25000000 -0.01666667
 [9] -0.26666667 -0.11666667  ...

sfo.origin$ArrDelay <200
```

```
   [1]  TRUE  TRUE  TRUE  TRUE  TRUE
   [6]  FALSE  TRUE  TRUE  TRUE  TRUE    ....
```

- **Recycling** – When two vectors have different lengths, the elements of the shorter vector may be **recycled**.
  - Typically a **Warning** is issued when this happens.
  - For some functions, an error results.

```
> x + c(1, 2)
[1] 2.2 3.0 4.0
Warning message:
longer object length
is not a multiple of shorter object length in:
x + c(1, 2)
```

Subsetting

- One of the most important things we do in statistics is to divide our data into subgroups for comparison,
  - United Airlines flights
  - Flights on Wednesday
  - Southwest flights with delays of more than 20 minutes

  What subgroup interests you?
- Vectors are ordered collections so we can extract subsets of elements by index or position.
- The **[ ]** is the subset operator for vectors (and matrices and lists).

Subsetting

There are five basic ways to refer to a subset.

Let's start by making a copy of the ArrDelay vector: `delay = sfo.origin$ArrDelay`

1. Position – **delay[2]** gives the second element of the vector.
2. Exclusion – **delay[-(1:200]** excludes the first 200 elements
3. Name – **sfo.origin["ArrDelay"]** returns the element named ArrDelay.

   ```
   > x = sfo.origin["ArrDelay"]
   > length(x)
   [1] 1
   ```

   Why is the length 1?
4. Logical -

   ```
   x = delay[ sfo.origin$UniqueCarrier == "UA"]
   ```

   What is being returned here?
5. All –

   ```
   delay[]
   ```

returns all of the elements
This can be helpful when we wish to reset all the values in a vector,
delay[] = 0
How do you think this differs from the command
delay = 0?

Data Frames support matrix-style subsetting:

```
sfo.origin[ sfo.origin$ArrDelay < 60, c("DayOfWeek", "holiday")]
```

What is being returned here?

Lists

A list is an ordered collection of objects that are possibly **heterogeneous**. The elements can be numeric vectors, character vectors, matrices, arrays, and lists.

myList = list(a = 1:10, b = ”def”, c(TRUE, FALSE, TRUE))

```
$a
 [1]  1  2  3  4  5  6  7  8  9 10
$b
 [1] "def"
[[3]]
 [1]  TRUE FALSE  TRUE
```

- **length(myList)** – there are 3 elements in the list
- **class(myList)** – the class is a “list”
- **names(myList)** – are “a”, “b” and the empty character “”
- **myList[1:2]** – returns a list with two elements
- **myList[1]** – returns a **list** with one element. What is length(myList[1]) ?
- **myList[[1]]** – returns a **vector** with ten elements, the numbers 1, 2, ..., 10 What is length(myList[[1]]) ?

Matrices

- A matrix in R is a collections of homogeneous elements arranged in 2 dimensions
- A matrix is a vector with a **dim** attribute, i.e. an integer vector giving the number or rows and columns
- To create matrices use **matrix()**
- The functions **dim()**, **nrow()** and **ncol()** provide the attributes of the matrix.
- Rows and columns can have names, **dimnames()**, **rownames()**, **colnames()**

```
> x = matrix(1:15, nrow =3, byrow=TRUE)
> dim(x)
[1] 3 5
> nrow(x)
[1] 3
> ncol(x)
[1] 5
> x
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    2    3    4    5
[2,]    6    7    8    9   10
[3,]   11   12   13   14   15
> rownames(x) =  letters[1:3]
> colnames(x) = letters[4:8]
> x
   d  e  f  g  h
a  1  2  3  4  5
b  6  7  8  9 10
c 11 12 13 14 15
```

Matrix Subsetting

```
> x
   d  e  f  g  h
a  1  2  3  4  5
b  6  7  8  9 10
c 11 12 13 14 15
```

- We can subset the rows and columns of **x** using the **[]** operator.
- **x[1:2,]** – gives all columns from the first two rows
- **x[, 3:4]** – gives all rows from the third and fourth columns
- **x[c(2, 3), c(4, 3)]** – returns a 2 by 2 matrix (notice the order of the rows and columns):

```
   g  d
b  9  6
c 14 11
```

Arrays

- Arrays are matrices in higher dimensions

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

- Subsetting carries over to arrays in the same way. What is the output from, **x[c(4, 3), 1:2, 2]**

sapply

```
>  is.factor(sfo.origin)
[1] FALSE

> sapply(sfo.origin, is.factor)[1:4]
ActualElapsedTime           AirTime          ArrDelay
            FALSE             FALSE             FALSE
          ArrHour
            FALSE
```

tapply

```
tapply(sfo.origin$ArrDelay, sfo.origin$UniqueCarrier, mean, na.rm=TRUE)
       9E        AA        AQ        AS        B6        CO
       NA 15.032760        NA 14.871052 11.963983  8.901331 ...
```