Class: 12th April 2018

In today's class we have discussed the following topics

1. Sorting a vector
2. Reversing a vector
3. Getting the elements number
4. Getting unique elements in a vector
5. Using some math functions:
    a. Log
    b. Exponential
    c. Sum
    d. Mean
    e. Max
    f. Min
    g. Rank
    h. Round
    i. Correlations and Covariance
    j. Variance
6. Matrix Multiplication
7. Lists
8. Plotting operations
    a. Add titles to Axes
    b. Changing color of the graph
    c. Overlaying graphs

Sorting or Ordering a vector

Sorting a vector in R is easy, all you need is to use sort() function.
Syntax: `sort(x, decreasing = FALSE)` or just `sort(x)` for ascending order sorting. Where x is our vector.

Example:

```
> x <- c(10,20,50,100,40,60,70,25,67,98,12,34,90,-34)
> sort(x)
 [1] -34  10  12  20  25  34  40  50  60  67  70  90  98 100
> sort(x, decreasing  = TRUE)
 [1] 100  98  90  70  67  60  50  40  34  25  20  12  10 -34
> sort(x, decreasing  = FALSE)
 [1] -34  10  12  20  25  34  40  50  60  67  70  90  98 100
```

Assignment:

1. `Can you sort characters and other data types? Justify your answer with example scripts.`

Reversing a vector

For reversing a vector, we use `rev()` function

Example:

```
> x <- c(10,50,20,30,56,66,22,45,67,1,0,2,-4,-6,22)
> rev(x)
[1] 22 -6 -4  2  0  1 67 45 22 66 56 30 20 50 10
```

Assignment:

1. Can you reverse a string using the `rev()` function?.
2. Can you sort and reverse a vector in a single script? (hints: you may think of using brackets) if yes, add an example to your answer.

Getting the elements number and Index number.

To get the number of elements, we use `length()` command.
See the example.

Example:

```
> x <- c(10,50,20,30,56,66,22,45,67,1,0,2,22)
> length(x)
[1] 13
```

Here the vector `x` has 13 elements. So, the `length(x)` command returned 13.

To get the index number of an element, we use `which()` command.
Syntax: `which(x == vector_element)`
Here, `x` is a vector.

Example:

The vector `x` has `5` elements. `22, 55, 21, 51` and `87`.

```
> x <- c (22, 55, 21, 51, 87)
> which(x==51)
[1] 4
```

Here we wanted to find the index number of 51. And our command returned 4. And hence the 4th index of the vector is equal to 51.

Assignment:

1. Find desired index numbers for a vector of integer, character and string.

Getting unique elements in a vector

`unique(vector_name)` function will return the unique elements of the vector.
In the example, `11` and `56` are input in the vector x twice each of them. So, the return of the unique() function came only the unique elements.

Example:

```
> x <- c(11,22,43,56,77,11,45,56)
> unique(x)
[1] 11 22 43 56 77 45
```

Assignment:

1. Check the function with other data types.

Using some math functions

Using Log

Using log in R requires calling the log() function.

Example:

```
> x <-60
> log(x, base = 2)
[1] 5.906891
> log(x, base = 10)
```

```
[1] 1.778151
> log(x, base = exp(10))
[1] 0.4094345
> y <- 4
> log(y, base = exp(2))
[1] 0.6931472
> log(y, base = 2)
[1] 2
> log2(y)
[1] 2
> log10(y)
[1] 0.60206
```

Using Exponential

For exponential, we use `exp()` function.

Example:

```
> x <- 100
> exp(x)
[1] 2.688117e+43
> expm1(x)
[1] 2.688117e+43
```

Using Sum

For summation in R, we use sum() function.

Example:

```
> sum(1:100)
[1] 5050
> sum(1,2,3,4,5,6,7,8,9,10)
[1] 55
> sum(1:5, 6:10)
[1] 55
Assignment:
[printing sum]
```

Using Mean

To get the mean in R, we use mean() function.
We can also get a good mean using the trim parameter.

Example:

```
> mean(1:10)
[1] 5.5
> x <- 1:10
> mean(x, trim = 0, na.rm = FALSE)
[1] 5.5
```

Using Max and Min

For finding maximum and minimum value we use max() and min() function respectively.

Example:

```
> max(1:10)
[1] 10
> max(3,6,1,66,70)
```

```
[1] 70
> max(9,34,0,122,56)
[1] 122
> min(1,6,33,98)
[1] 1
> min(1:10)
[1] 1
> min(1:5, 5:10)
[1] 1
> max(1:5,6:10)
[1] 10
```

Using Rank and Round

Returns the sample ranks of the values in a vector. Ties (i.e., equal values) and missing values can be handled in several ways.

Example:

```
> x = c(1:10)
> x
 [1]  1  2  3  4  5  6  7  8  9 10
> rank(x, na.last = TRUE,
+      ties.method = c("average", "first", "last", "random",
"max", "min"))
 [1]  1  2  3  4  5  6  7  8  9 10
```

Rounding a number

`ceiling` takes a single numeric argument `x` and returns a numeric vector containing the smallest integers not less than the corresponding elements of `x`.

`floor` takes a single numeric argument `x` and returns a numeric vector containing the largest integers not greater than the corresponding elements of `x`.

`trunc` takes a single numeric argument `x` and returns a numeric vector containing the integers formed by truncating the values in `x` toward `0`.

`round` rounds the values in its first argument to the specified number of decimal places (default 0).

`signif` rounds the values in its first argument to the specified number of significant digits.

Example:

```
> x <- 67.456
> ceiling(x)
[1] 68
> floor(x)
[1] 67
> trunc(x)
[1] 67
> round(x)
```

```
[1] 67
> signif(x)
[1] 67.456
```

Correlations and Covariance in R

cor() is used to find the correlation in R. The basic syntax for the cor() function is as follow:

Syntax: `cor(x, use=”options”, method=”options” )`
Where,
`x` is the matrix or data form.
`use` Specifies the handling of missing data. Options are **all.obs**(assumes no missing data - missing data will produce an error), **complete.obs** (listwise deletion), and **pairwise.complete.obs**(pairwise deletion)
`method` Specifies the type of correlation. Options are **pearson**, **spearman** or **kendall**
Example:

```
> cov(mtcars, use="complete.obs")
            mpg         cyl        disp          hp
drat          wt
mpg    36.324103  -9.1723790  -633.09721 -320.732056
2.19506351  -5.1166847
cyl    -9.172379   3.1895161   199.66028  101.931452  -
0.66836694   1.3673710
disp -633.097208 199.6602823 15360.79983 6721.158669 -
47.06401915 107.6842040
hp   -320.732056 101.9314516  6721.15867 4700.866935 -
16.45110887  44.1926613
drat    2.195064  -0.6683669   -47.06402  -16.451109
0.28588135  -0.3727207
wt     -5.116685   1.3673710   107.68420   44.192661  -
0.37272073   0.9573790
qsec    4.509149  -1.8868548   -96.05168  -86.770081
0.08714073  -0.3054816
vs      2.017137  -0.7298387   -44.37762  -24.987903
0.11864919  -0.2736613
am      1.803931  -0.4657258   -36.56401   -8.320565
0.19015121  -0.3381048
gear    2.135685  -0.6491935   -50.80262   -6.358871
0.27598790  -0.4210806
carb   -5.363105   1.5201613    79.06875   83.036290  -
0.07840726   0.6757903
            qsec           vs           am        gear
carb
mpg    4.50914919   2.01713710   1.80393145   2.1356855 -
5.36310484
cyl   -1.88685484  -0.72983871  -0.46572581  -0.6491935
1.52016129
```

```
disp -96.05168145 -44.37762097 -36.56401210 -50.8026210
79.06875000
hp   -86.77008065 -24.98790323  -8.32056452  -6.3588710
83.03629032
drat   0.08714073   0.11864919   0.19015121   0.2759879 -
0.07840726
wt    -0.30548161  -0.27366129  -0.33810484  -0.4210806
0.67579032
qsec   3.19316613   0.67056452  -0.20495968  -0.2804032 -
1.89411290
vs     0.67056452   0.25403226   0.04233871   0.0766129 -
0.46370968
am    -0.20495968   0.04233871   0.24899194   0.2923387
0.04637097
gear  -0.28040323   0.07661290   0.29233871   0.5443548
0.32661290
carb  -1.89411290  -0.46370968   0.04637097   0.3266129
2.60887097

R version 3.4.4 (2018-03-15) -- "Someone to Lean On"
Copyright (C) 2018 The R Foundation for Statistical Computing
Platform: x86_64-apple-darwin15.6.0 (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

  Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

> regresmod <- lm(Life.Exp ~ Population + Income + Illiteracy +
Murder + HS.Grad + Area)
Error in eval(predvars, data, env) : object 'Life.Exp' not found
> library("corrplot")
corrplot 0.84 loaded
> data <- data.frame(state.x77)
> mat <- cor(data)
> corrplot(mat,method = "circle")
> x <- c(2,3,3,4,52,3423,2332,23,25,345,345,34,534,534,5,34)
> x
```

```
 [1]    2    3    3    4   52 3423 2332   23   25  345  345   34
534  534    5   34
> sort(x)
 [1]    2    3    3    4    5   23   25   34   34   52  345  345
534  534 2332 3423
> sort(x, descending = TRUE)
Error in sort.int(x, na.last = na.last, decreasing = decreasing,
...) :
  unused argument (descending = TRUE)
> sort(x, decreasing  = TRUE)
 [1] 3423 2332  534  534  345  345   52   34   34   25   23    5
4    3    3    2
> sort(x, decreasing  = FALSE)
 [1]    2    3    3    4    5   23   25   34   34   52  345  345
534  534 2332 3423
> x <- c(10,20,50,100,40,60,70,25,67,98,12,34,90,-34)
> sort(x)
 [1] -34  10  12  20  25  34  40  50  60  67  70  90  98 100
> sort(x, decreasing  = TRUE)
 [1] 100  98  90  70  67  60  50  40  34  25  20  12  10 -34
> sort(x, decreasing  = FALSE)
 [1] -34  10  12  20  25  34  40  50  60  67  70  90  98 100
> x <- c('a','f','h','c','q','x')
> sort(x)
[1] "a" "c" "f" "h" "q" "x"
> sort(x, decreasing = TRUE)
[1] "x" "q" "h" "f" "c" "a"
> hsb2 <- read.table("https://stats.idre.ucla.edu/wp-
content/uploads/2016/02/hsb2-1.csv", header=T, sep=",")
>
> attach(hsb2)
> hsb2[1:10, ]
    id female race ses schtyp prog read write math science socst
1   70      0    4   1      1    1   57    52   41      47    57
2  121      1    4   2      1    3   68    59   53      63    61
3   86      0    4   3      1    1   44    33   54      58    31
4  141      0    4   3      1    3   63    44   47      53    56
5  172      0    4   2      1    2   47    52   57      53    61
6  113      0    4   2      1    2   44    52   51      63    61
7   50      0    3   2      1    1   50    59   42      53    61
8   11      0    1   2      1    2   34    46   45      39    36
9   84      0    4   2      1    1   63    57   54      58    51
10  48      0    3   2      1    2   57    55   52      50    51
> sort(read)
  [1] 28 31 34 34 34 34 34 34 35 36 36 36 37 37 39 39 39 39 39
39 39 39 41 41 42 42 42 42 42 42 42 42
```

```
 [33] 42 42 42 42 42 43 43 44 44 44 44 44 44 44 44 44 44 44 44
44 45 45 46 47 47 47 47 47 47 47 47 47
 [65] 47 47 47 47 47 47 47 47 47 47 47 47 47 47 47 47 47 47 48
50 50 50 50 50 50 50 50 50 50 50 50 50
 [97] 50 50 50 50 50 52 52 52 52 52 52 52 52 52 52 52 52 52 52
53 54 55 55 55 55 55 55 55 55 55 55 55
[129] 55 55 57 57 57 57 57 57 57 57 57 57 57 57 57 57 60 60 60
60 60 60 60 60 60 61 63 63 63 63 63 63
[161] 63 63 63 63 63 63 63 63 63 63 65 65 65 65 65 65 65 65 65
66 68 68 68 68 68 68 68 68 68 68 68 71
[193] 71 73 73 73 73 73 76 76
> sort(math)
  [1] 33 35 37 38 38 39 39 39 39 39 39 40 40 40 40 40 40 40 40
40 40 41 41 41 41 41 41 41 42 42 42 42
 [33] 42 42 42 43 43 43 43 43 43 43 44 44 44 44 45 45 45 45 45
45 45 45 46 46 46 46 46 46 46 46 47 47
 [65] 47 48 48 48 48 48 49 49 49 49 49 49 49 49 49 49 50 50 50
50 50 50 50 51 51 51 51 51 51 51 51 52
 [97] 52 52 52 52 52 53 53 53 53 53 53 53 54 54 54 54 54 54 54
54 54 54 55 55 55 55 55 56 56 56 56 56
[129] 56 56 57 57 57 57 57 57 57 57 57 57 57 57 57 58 58 58 58
58 58 59 59 60 60 60 60 60 61 61 61 61
[161] 61 61 61 62 62 62 62 63 63 63 63 63 64 64 64 64 64 65 65
65 66 66 66 66 67 67 68 69 69 70 71 71
[193] 71 71 72 72 72 73 75 75
> sort(id)
  [1]   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]  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]  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]  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]  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] 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] 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] 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] 193 194 195 196 197 198 199 200
> mat <- cor(hsb2)
> corrplot(mat, method = "circle")
> rev(x)
[1] "x" "q" "c" "h" "f" "a"
```

```
> sort(x)
[1] "a" "c" "f" "h" "q" "x"
> sort(x, decreasing = TRUE)
[1] "x" "q" "h" "f" "c" "a"
> clear
Error: object 'clear' not found
> x <- c(10,20,50,100,40,60,70,25,67,98,12,34,90)
> sort(x, decreasing = TRUE)
 [1] 100  98  90  70  67  60  50  40  34  25  20  12  10
> rev(x)
 [1]  90  34  12  98  67  25  70  60  40 100  50  20  10
> y <- "MY NAME IS NICK"
> rev(y)
[1] "MY NAME IS NICK"
> x <- c(10,50,20,30,56,66,22,45,67,1,0,2,-4,-6,22)
> rev(x)
 [1] 22 -6 -4  2  0  1 67 45 22 66 56 30 20 50 10
> sort(x, decreasing = TRUE)
 [1] 67 66 56 50 45 30 22 22 20 10  2  1  0 -4 -6
> rev(id)
  [1] 137 118 187 145  31 179  30 184 175 124  52 188  25 158
23  32 160  92 193  79  63  64  78  59 135
 [26]  36  26  51  13 119  98 186 111 156  69 112 161 198  74
147  39 109 148 110 139  71  10 138  96  43
 [51]  46   6 182  83 191 122  17 142  90  19  55   2  42 190
61  77  72  66  33 116  45  91  28 105 152
 [76]  44 151  73  87  35  37 163  93 130 106  34 125 131   4
162 180  54  89 101  65 166 120  47  99  88
[101] 194   1 100  57 173 129   8  82 149  24  94 133 117 102
146  58   3 174 165 127  14 164 123 159   5
[126]  56 157  68  97 155  18  81 107 171 140 197 108 134 170
181   9 185  22  67  15 132 183  21 128  27
[151]   7 189 136  49 169  62  40 168 177 176 153  16  80 200
144 199 150 192 103 126  29 196 178 154  53
[176]  12  20  41 143 167  85 114 195  76 115  38 104  95  60
75  48  84  11  50 113 172 141  86 121  70
> p <- sort(id)
> rev(p)
  [1] 200 199 198 197 196 195 194 193 192 191 190 189 188 187
186 185 184 183 182 181 180 179 178 177 176
 [26] 175 174 173 172 171 170 169 168 167 166 165 164 163 162
161 160 159 158 157 156 155 154 153 152 151
 [51] 150 149 148 147 146 145 144 143 142 141 140 139 138 137
136 135 134 133 132 131 130 129 128 127 126
 [76] 125 124 123 122 121 120 119 118 117 116 115 114 113 112
111 110 109 108 107 106 105 104 103 102 101
```

```
[101] 100  99  98  97  96  95  94  93  92  91  90  89  88  87
86  85  84  83  82  81  80  79  78  77  76
[126]  75  74  73  72  71  70  69  68  67  66  65  64  63  62
61  60  59  58  57  56  55  54  53  52  51
[151]  50  49  48  47  46  45  44  43  42  41  40  39  38  37
36  35  34  33  32  31  30  29  28  27  26
[176]  25  24  23  22  21  20  19  18  17  16  15  14  13  12
11  10   9   8   7   6   5   4   3   2   1
> sort(rev(x))
 [1] -6 -4  0  1  2 10 20 22 22 30 45 50 56 66 67
> length(x)
[1] 15
> x <- c(10,50,20,30,56,66,22,45,67,1,0,2,22)
> length(x)
[1] 13
> getINDEX( x, value = 200 )
Error in getINDEX(x, value = 200) : could not find function
"getINDEX"
> getINDEX( x, value = 10 )
Error in getINDEX(x, value = 10) : could not find function
"getINDEX"
> which(x == 67)
[1] 9
> x <- c (22, 55, 21, 51, 87)
> which(x==51)
[1] 4
> corrplot(mat, method = "circle")
> unique(x)
[1] 22 55 21 51 87
> x
[1] 22 55 21 51 87
> x <- c(11,22,43,56,77,11,45,56)
> unique(x)
[1] 11 22 43 56 77 45
> unique(-x)
[1] -11 -22 -43 -56 -77 -45
> x
[1] 11 22 43 56 77 11 45 56
> c[x]
Error in c[x] : object of type 'builtin' is not subsettable
> c
function (...)  .Primitive("c")
> log(50)
[1] 3.912023
> log(10)
[1] 2.302585
> log(e)
```

```
Error: object 'e' not found
> log(exp)
Error in log(exp) : non-numeric argument to mathematical
function
> x <- 50
> x
[1] 50
> x <-60
> log(x, base = 2)
[1] 5.906891
> log(x, base = 10)
[1] 1.778151
> log(x, base = exp(10))
[1] 0.4094345
> y <- 4
> log(y, x = 2)
[1] 0.5
> log(y, x = exp(2))
[1] 1.442695
> log(y, x = exp(.5))
[1] 0.3606738
> log(y, base = exp(2))
[1] 0.6931472
> log(y, base = 2)
[1] 2
> x <- 100
> exp(x)
[1] 2.688117e+43
> log2(y)
[1] 2
> log10(y)
[1] 0.60206
> expm1(x)
[1] 2.688117e+43
> sum(1:100)
[1] 5050
> sum(1,2,3,4,5,6,7,8,9,10)
[1] 55
> sum(1:5, 6:10)
[1] 55
> mean(1:10)
[1] 5.5
> mean(1:5,6:10)
Error in mean.default(1:5, 6:10) : 'trim' must be numeric of
length one
> x <- 1:10
> mean(x, trim = 0, na.rm = FALSE)
```

```
[1] 5.5
> mean(x, trim = 10, na.rm = FALSE)
[1] 5.5
> mean(x, trim = .3, na.rm = FALSE)
[1] 5.5
> mean(x, trim = .3, xna.rm = FALSE)
[1] 5.5
> x < 1.5:10.5
 [1] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
> x
 [1]  1  2  3  4  5  6  7  8  9 10
> x < seq(from = 1.5, to = 100.5, by .5)
Error: unexpected numeric constant in "x < seq(from = 1.5, to = 
100.5, by .5"
> x < rep(from = 1.5, to = 100.5, by .5)
Error: unexpected numeric constant in "x < rep(from = 1.5, to = 
100.5, by .5"
> max(1:10)
[1] 10
> max(3,6,1,66,70)
[1] 70
> max(9,34,0,122,56)
[1] 122
> min(1,6,33,98)
[1] 1
> min(1:10)
[1] 1
> min(1:5, 5:10)
[1] 1
> max(1:5,6:10)
[1] 10
> rank(x, na.last = TRUE,
+      ties.method = c("average", "first", "last", "random", 
"max", "min"))
 [1]  1  2  3  4  5  6  7  8  9 10
> x = c(1:10)
> x
 [1]  1  2  3  4  5  6  7  8  9 10
> rank(x, na.last = TRUE,
+      ties.method = c("average", "first", "last", "random", 
"max", "min"))
 [1]  1  2  3  4  5  6  7  8  9 10
> x <- 67.456
> ceiling(x)
[1] 68
> floor(x)
[1] 67
```

```
> trunc(x)
[1] 67
> round(x)
[1] 67
> signif
function (x, digits = 6)  .Primitive("signif")
> signif(x)
[1] 67.456
> cor(mtcars, use="complete.obs", method="kendall")
            mpg        cyl       disp         hp        drat
wt
mpg   1.0000000 -0.7953134 -0.7681311 -0.7428125  0.46454879 -
0.7278321
cyl  -0.7953134  1.0000000  0.8144263  0.7851865 -0.55131785
0.7282611
disp -0.7681311  0.8144263  1.0000000  0.6659987 -0.49898277
0.7433824
hp   -0.7428125  0.7851865  0.6659987  1.0000000 -0.38262689
0.6113081
drat  0.4645488 -0.5513178 -0.4989828 -0.3826269  1.00000000 -
0.5471495
wt   -0.7278321  0.7282611  0.7433824  0.6113081 -0.54714953
1.0000000
qsec  0.3153652 -0.4489698 -0.3008155 -0.4729061  0.03272155 -
0.1419881
vs    0.5896790 -0.7710007 -0.6033059 -0.6305926  0.37510111 -
0.4884787
am    0.4690128 -0.4946212 -0.5202739 -0.3039956  0.57554849 -
0.6138790
gear  0.4331509 -0.5125435 -0.4759795 -0.2794458  0.58392476 -
0.5435956
carb -0.5043945  0.4654299  0.4137360  0.5959842 -0.09535193
0.3713741
            qsec         vs          am        gear        carb
mpg   0.31536522  0.5896790  0.46901280  0.43315089 -0.50439455
cyl  -0.44896982 -0.7710007 -0.49462115 -0.51254349  0.46542994
disp -0.30081549 -0.6033059 -0.52027392 -0.47597955  0.41373600
hp   -0.47290613 -0.6305926 -0.30399557 -0.27944584  0.59598416
drat  0.03272155  0.3751011  0.57554849  0.58392476 -0.09535193
wt   -0.14198812 -0.4884787 -0.61387896 -0.54359562  0.37137413
qsec  1.00000000  0.6575431 -0.16890405 -0.09126069 -0.50643945
vs    0.65754312  1.0000000  0.16834512  0.26974788 -0.57692729
am   -0.16890405  0.1683451  1.00000000  0.77078758 -0.05859929
gear -0.09126069  0.2697479  0.77078758  1.00000000  0.09801487
carb -0.50643945 -0.5769273 -0.05859929  0.09801487  1.00000000
> cov(mtcars, use="complete.obs")
```

```
             mpg         cyl        disp          hp
drat          wt
mpg    36.324103  -9.1723790  -633.09721 -320.732056
2.19506351  -5.1166847
cyl    -9.172379   3.1895161   199.66028  101.931452  -
0.66836694   1.3673710
disp -633.097208 199.6602823 15360.79983 6721.158669 -
47.06401915 107.6842040
hp   -320.732056 101.9314516  6721.15867 4700.866935 -
16.45110887  44.1926613
drat    2.195064  -0.6683669   -47.06402  -16.451109
0.28588135  -0.3727207
wt     -5.116685   1.3673710   107.68420   44.192661  -
0.37272073   0.9573790
qsec    4.509149  -1.8868548   -96.05168  -86.770081
0.08714073  -0.3054816
vs      2.017137  -0.7298387   -44.37762  -24.987903
0.11864919  -0.2736613
am      1.803931  -0.4657258   -36.56401   -8.320565
0.19015121  -0.3381048
gear    2.135685  -0.6491935   -50.80262   -6.358871
0.27598790  -0.4210806
carb   -5.363105   1.5201613    79.06875   83.036290  -
0.07840726   0.6757903
             qsec           vs           am        gear
carb
mpg    4.50914919   2.01713710   1.80393145   2.1356855 -
5.36310484
cyl   -1.88685484  -0.72983871  -0.46572581  -0.6491935
1.52016129
disp -96.05168145 -44.37762097 -36.56401210 -50.8026210
79.06875000
hp   -86.77008065 -24.98790323  -8.32056452  -6.3588710
83.03629032
drat   0.08714073   0.11864919   0.19015121   0.2759879 -
0.07840726
wt    -0.30548161  -0.27366129  -0.33810484  -0.4210806
0.67579032
qsec   3.19316613   0.67056452  -0.20495968  -0.2804032 -
1.89411290
vs     0.67056452   0.25403226   0.04233871   0.0766129 -
0.46370968
am    -0.20495968   0.04233871   0.24899194   0.2923387
0.04637097
gear  -0.28040323   0.07661290   0.29233871   0.5443548
0.32661290
```

```
carb  -1.89411290  -0.46370968   0.04637097   0.3266129
2.60887097
> cor(mtcars, use="complete.obs", method="kendall")
            mpg        cyl       disp          hp        drat
wt
mpg   1.0000000 -0.7953134 -0.7681311 -0.7428125  0.46454879 -
0.7278321
cyl  -0.7953134  1.0000000  0.8144263  0.7851865 -0.55131785
0.7282611
disp -0.7681311  0.8144263  1.0000000  0.6659987 -0.49898277
0.7433824
hp   -0.7428125  0.7851865  0.6659987  1.0000000 -0.38262689
0.6113081
drat  0.4645488 -0.5513178 -0.4989828 -0.3826269  1.00000000 -
0.5471495
wt   -0.7278321  0.7282611  0.7433824  0.6113081 -0.54714953
1.0000000
qsec  0.3153652 -0.4489698 -0.3008155 -0.4729061  0.03272155 -
0.1419881
vs    0.5896790 -0.7710007 -0.6033059 -0.6305926  0.37510111 -
0.4884787
am    0.4690128 -0.4946212 -0.5202739 -0.3039956  0.57554849 -
0.6138790
gear  0.4331509 -0.5125435 -0.4759795 -0.2794458  0.58392476 -
0.5435956
carb -0.5043945  0.4654299  0.4137360  0.5959842 -0.09535193
0.3713741
            qsec         vs          am        gear        carb
mpg   0.31536522  0.5896790  0.46901280  0.43315089 -0.50439455
cyl  -0.44896982 -0.7710007 -0.49462115 -0.51254349  0.46542994
disp -0.30081549 -0.6033059 -0.52027392 -0.47597955  0.41373600
hp   -0.47290613 -0.6305926 -0.30399557 -0.27944584  0.59598416
drat  0.03272155  0.3751011  0.57554849  0.58392476 -0.09535193
wt   -0.14198812 -0.4884787 -0.61387896 -0.54359562  0.37137413
qsec  1.00000000  0.6575431 -0.16890405 -0.09126069 -0.50643945
vs    0.65754312  1.0000000  0.16834512  0.26974788 -0.57692729
am   -0.16890405  0.1683451  1.00000000  0.77078758 -0.05859929
gear -0.09126069  0.2697479  0.77078758  1.00000000  0.09801487
carb -0.50643945 -0.5769273 -0.05859929  0.09801487  1.00000000
```

NB: Here `mtcars` is Motor Trend Car Road Test data from 1974 in an US Magazine. It is a predefined dataset in R. We can use our own data frame. We will discuss about data framing in the next class.
The correlation coefficient of two variables in a data set equals to their covariance divided by the product of their individual standard deviations. It is a normalized measurement of how the two are linearly related.

Formally, the sample correlation coefficient is defined by the following formula, where $s_x$and s_y are the sample standard deviations, and s_{xy} is the sample covariance.

$$r_{xy} = \frac{s_{xy}}{s_x s_y}$$

Similarly, the **population correlation coefficient** is defined as follows, where σ_x and σ_y are the population standard deviations, and σ_{xy} is the population covariance.

$$\rho_{xy} = \frac{\sigma_{xy}}{\sigma_x \sigma_y}$$

If the correlation coefficient is close to 1, it would indicate that the variables are positively linearly related and the scatter plot falls almost along a straight line with positive slope. For -1, it indicates that the variables are negatively linearly related and the scatter plot almost falls along a straight line with negative slope. And for zero, it would indicate a weak linear relationship between the variables.

Problem

Find the correlation coefficient of eruption duration and waiting time in the data set faithful. Observe if there is any linear relationship between the variables.

Solution

We apply the `cor()` function to compute the correlation coefficient of eruptions and waiting.

```
> duration = faithful$eruptions   # eruption durations
>  waiting = faithful$waiting      # the waiting period
> cor(duration, waiting)          # apply the cor function
[1] 0.9008112
```

Answer

The correlation coefficient of eruption duration and waiting time is 0.90081. Since it is rather close to 1, we can conclude that the variables are positively linearly related.

Covariance

The covariance of two variables *x* and *y* in a data set measures how the two are linearly related. A positive covariance would indicate a positive linear relationship between the variables, and a negative covariance would indicate the opposite.

The **sample covariance** is defined in terms of the sample means as:

$$s_{xy} = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})$$

Similarly, the **population covariance** is defined in terms of the population mean μ_x, μ_y as:

$$\sigma_{xy} = \frac{1}{N}\sum_{i=1}^{N}(x_i - \mu_x)(y_i - \mu_y)$$

In R, we use `cov()` function to get the covariance.

Example:

```
> duration = faithful$eruptions   # eruption durations
> waiting = faithful$waiting      # the waiting period
> cov(duration, waiting)          # apply the cov function
[1] 13.97781
```
NB: faithful defines the Old Faithful Geyser Data. We can use our data too.

Assignment:

1. Find a real life problem and get the covariance and correlation and check your result with the result in R console.

Lists

List is used to store multiple data type in a single variable. It is a generic vector. Say, we have 3 vectors as follow.
```
> n = c(3,4,5,6,7)
> s = c("aa", "bb", "cc", "dd", "ee")
> b = c(TRUE, FALSE, TRUE, FALSE, FALSE)
```
Let's put them in a list. We use `list()` function for that and call it after storing the list into a variable `x`.
```
> x = list(n, s, b)
> x
[[1]]
[1] 3 4 5 6 7

[[2]]
[1] "aa" "bb" "cc" "dd" "ee"

[[3]]
[1]  TRUE FALSE  TRUE FALSE FALSE
```
We can explore the list by index numbers.
```
> x[1]
[[1]]
[1] 3 4 5 6 7
> x[2]
[[1]]
[1] "aa" "bb" "cc" "dd" "ee"
> x[3]
[[1]]
[1]  TRUE FALSE  TRUE FALSE FALSE
```
Assignment:

1. What will be the values of `x[1][2]`, `x[c(2, 4)]`, `x[[2]][1]` and why?

END OF CLASS

Next Class

1. Plotting Data
2. Working with datasets
3. Correlations and Covariance with plotting
4. Data frames accessing
5. Reading files

6. Gist about packages and installing packages.