R 기본 활용법 리뷰

서울시 빅데이터 캠퍼스

서울시 빅데이터 캠퍼스에서 가장 중심이 되는 분석툴이 SQL 과 R 입니다. 데이터베이스에 테이블로 저장된 데이터가 아닌 경우는 R 을 이용하시어 분석을 하셔야 할 수 밖에 없기 때문에 통계 분석환경 R 의 사용법이 익숙치 않으신 분들을 위하여 기초적인 벡터 데이터 활용 및 매트릭스와 어레이 데이터 구조의 이해 및 R 의 jpeg 패키지를 활용한 데이터 인덱싱 연습을 하실 수 있도록 구성하였습니다.

수식관련 주요 내장함수를 활용한 공식 코딩

R 은 기본적으로 통계학 공부를 위한 프로그램으로 다양한 수학 공식을 활용할 수 있도록 수학 관련 내장함수들이 아래와 같이 다수 존재 합니다. 이들을 정확히 코딩할 줄 아는 것은 매우 중요합니다.

수식의 감을 잡을 수 있도록 하는데 그래프를 활용하는 것이 좋다고 생각합니다. 먼저 plot() 함수에 대해 간략히 설명 드리겠습니다.

구분	기능	R 내장 함수 (Vectorized 함수)
기본 수식 계산 함수	가 , 감 , 승 , 제	+ , - , *, /
	합계	sum(variable)
	평균	mean(variable)
	절대값	abs(variable)
	제곱근	sqrt(variable)
	N제곱 , n제곱근	variable^n , variable^(1/n)
	올림 , 내림	ceiling(variable), floor(variable)
	지수함수	exp(variable)
	소수점 n자리 반올림	round(variable, digits=n)
	자연로그, 상용로그	log(variable), log10(variable)
	숫자 ↔ 문자열 전환	as.numeric("variable"), as.character(variable)

plot 함수 소개

plot 함수는 주어진 특정한 데이터를 그래프로 화면에 출력해 주는 함수 입니다. 그래프(graph)는 X 축과 Y 축을 잡고, 임의의 x 값에 대응되는 y 값을 구하여 각각의

값이 만나는 그 위치에 점을 찍으면 그려집니다. R 의 plot 함수에서는 x 축에 사용될 벡터와 x 값 벡터에 일대일로 대응하는 y 값 벡터를 지정하면, R 이 그래프를 만들어 화면에 출력을 해줍니다.

plot 함수는 많은 인자(Argument)를 가지고 있습니다만, 그래프를 그리는데 있어서 가장 중요한 Argument 는 x 와 y 입니다. 먼저 데이터를 준비합니다.

```
xVar <- seq(-10,10, by=0.1)
length(xVar)

## [1] 201

head(xVar)

## [1] -10.0  -9.9  -9.8  -9.7  -9.6  -9.5

tail(xVar)

## [1]  9.5  9.6  9.7  9.8  9.9 10.0
```

이러한 x 축값에 대응하는 y 축값을 바꾸어 가면서 그래프를 그려 보겠습니다. 다만, y 값의 변화에 따른 그래프의 변화가 보이도록 투입된 y 값에 맞추어 y 축의 최소-최대 범위가 자동조정되지 않고, y 축에 제가 지정한 범위가 항상 보이도록 하는 plot 함수의 인자(Argument) 키워드인 ylim 을 최대값 및 최소값으로 구성되는 벡터로 c(-30,30)으로 지정하여 xVar 값의 변화에 따라 연동되어 변하는 yVar 값의 변화값 즉, 기울기의 변화가 느껴지도록 하겠습니다.

아래와 같이 한 번 실습해 보시고, plot 함수에서 ylim 을 제거하신 후, 즉 plot(x=xVar, y=yVar) 등과 같이 실습하시면서 무슨 차이가 있는지 보시는 것도 좋으실 것 같습니다.

```
yVar <- xVar
y2Var <- xVar * 2
y3Var <- xVar * 3
yhalfVar <- xVar / 2

length(xVar)
```

```
## [1] 201
```

```
length(yVar)
```

```
## [1] 201
```

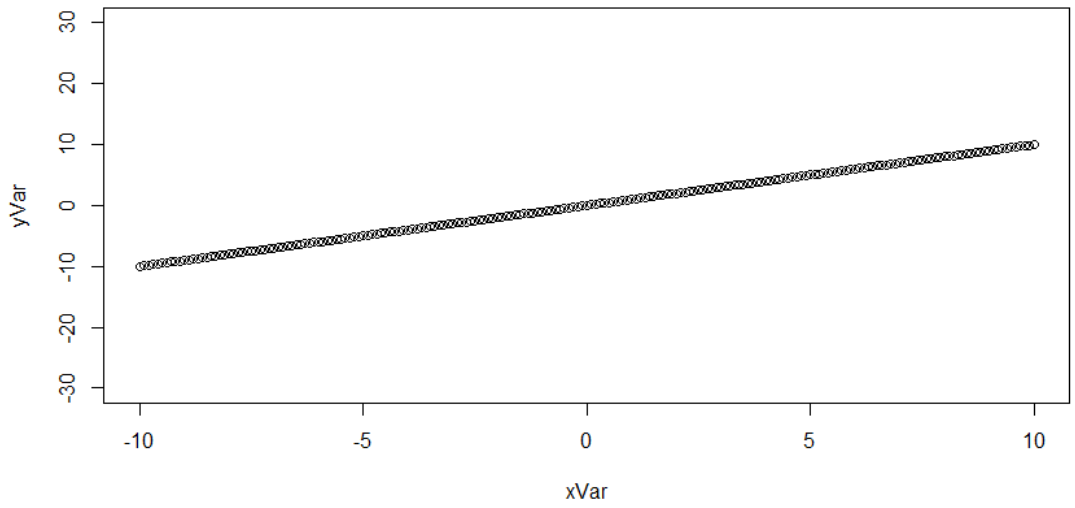
```
fivenum(xVar)
```

```
## [1] -10  -5   0   5  10
```

```
fivenum(yVar)
```

```
## [1] -10  -5   0   5  10
```

```
plot(x=xVar, y=yVar, ylim=c(-30,30))
```




```
length(xVar)
```

```
## [1] 201
```

```
length(y2Var)
```

```
## [1] 201
```

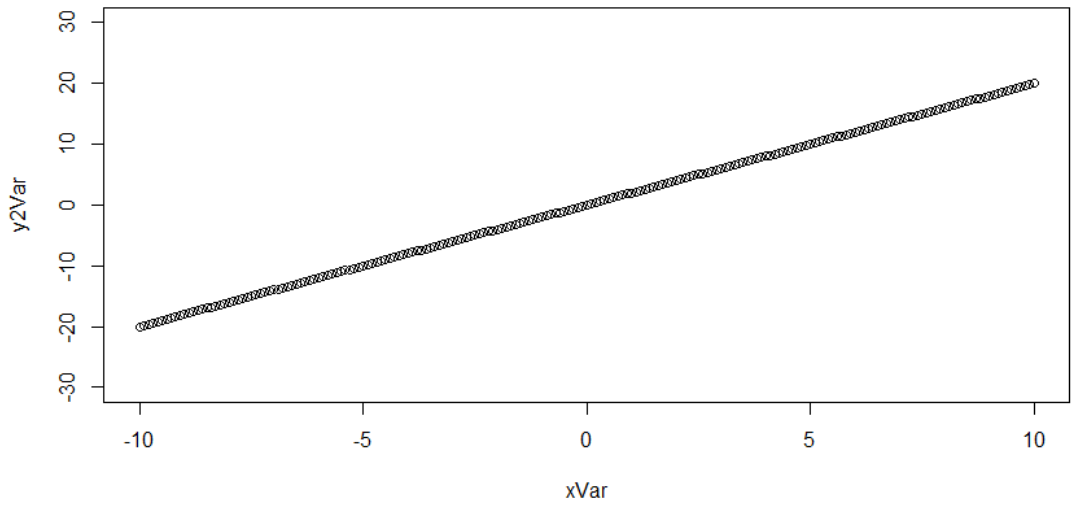
```
fivenum(xVar)
```

```
## [1] -10  -5   0   5  10
```

```
fivenum(y2Var)
```

```
## [1] -20 -10   0  10  20
```

```
plot(x=xVar, y=y2Var, ylim=c(-30,30))
```




```
length(xVar)
```

```
## [1] 201
```

```
length(y3Var)
```

```
## [1] 201
```

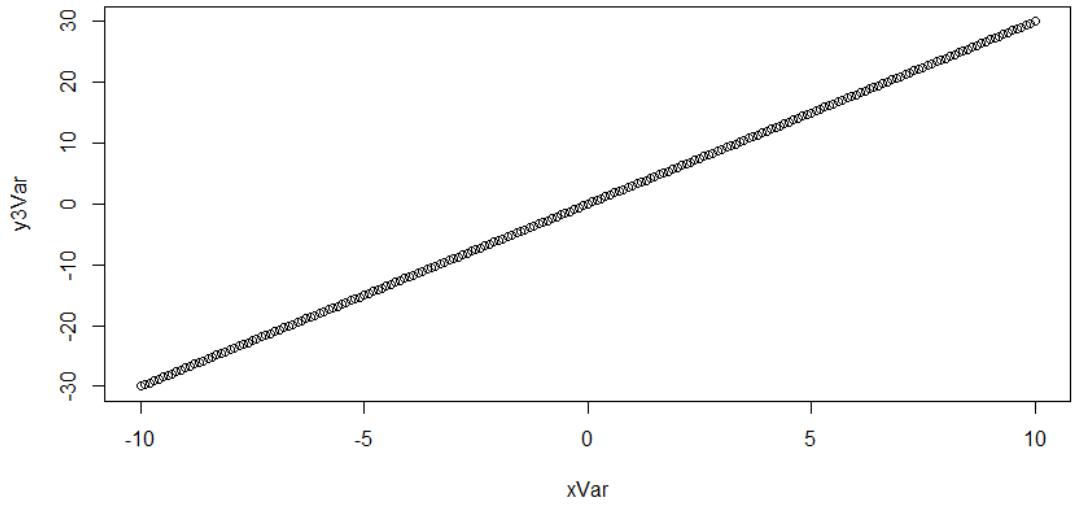
```
fivenum(xVar)
```

```
## [1] -10  -5   0   5  10
```

```
fivenum(y3Var)
```

```
## [1] -30 -15   0  15  30
```

```
plot(x=xVar, y=y3Var, ylim=c(-30,30))
```




```
length(xVar)
```

```
## [1] 201
```

```
length(yhalfVar)
```

```
## [1] 201
```

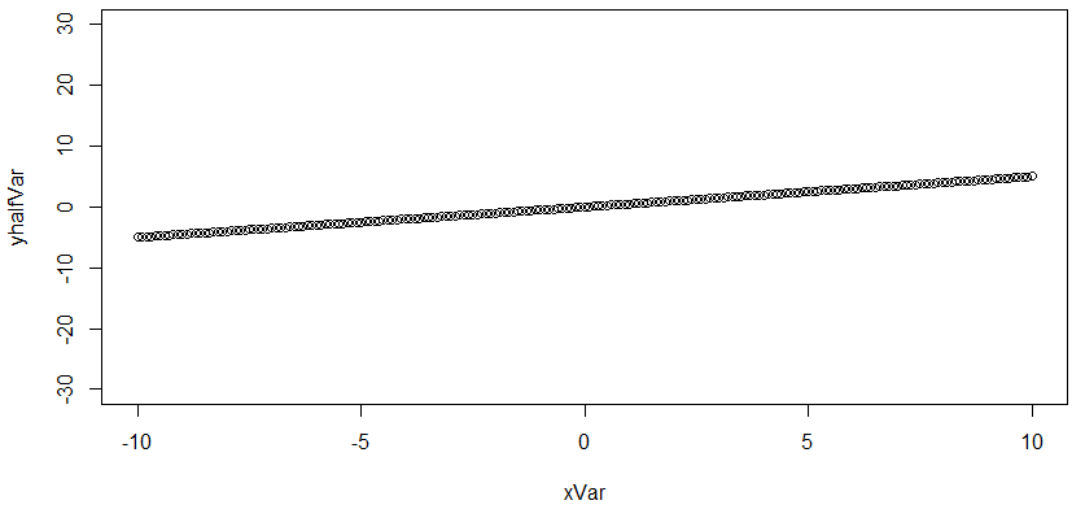
```
fivenum(xVar)
```

```
## [1] -10  -5   0   5  10
```

```
fivenum(yhalfVar)
```

```
## [1] -5.0 -2.5  0.0  2.5  5.0
```

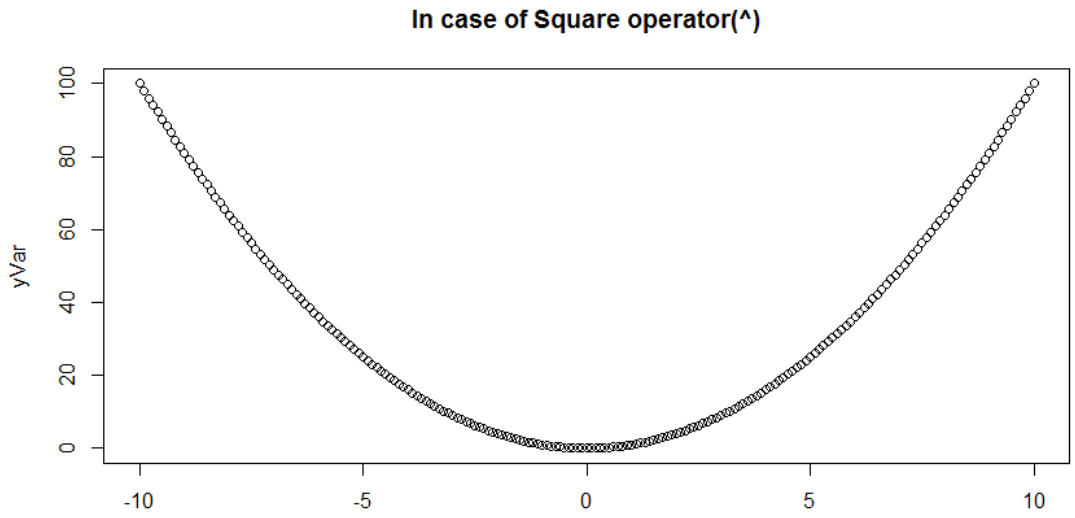
```
plot(x=xVar, y=yhalfVar, ylim=c(-30,30))
```



수식관련 연산자/내장함수 코딩 연습

다음부터는 y 변수의 값을 일되게 yVar 로 통일시키고, 불필요한 지면의 낭비를 지우기 위해 X 축명 표시를 제외하고, 표 제목을 넣도록 하겠습니다. 이러한 작업은 모두 plot 함수의 인자값을 지정하여 실행됩니다.

```
yVar <- xVar^2

c(length(xVar), length(yVar))

## [1] 201 201

fivenum(xVar)

## [1] -10  -5   0   5  10

fivenum(yVar)

## [1]   0.00   6.25  25.00  56.25 100.00

plot(x=xVar, y=yVar, xlab="", main="In case of Square operator(^)")
```



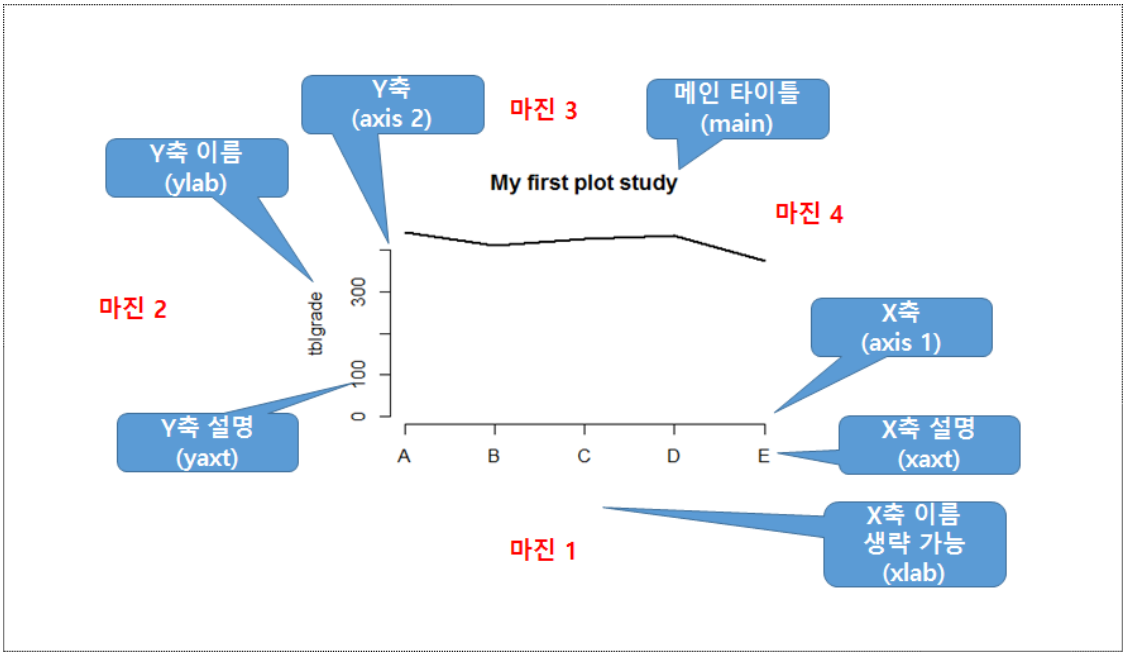
Plot 함수 주요 인자(Argument) 소개

plot 함수는 R 에서 가장 일반적으로 사용되는 함수로 명시적인 인자는 아래와 같은 x 와 y 두개이나, 그래프 영역을 콘트롤하는 인자들이 아주 많이 있습니다.

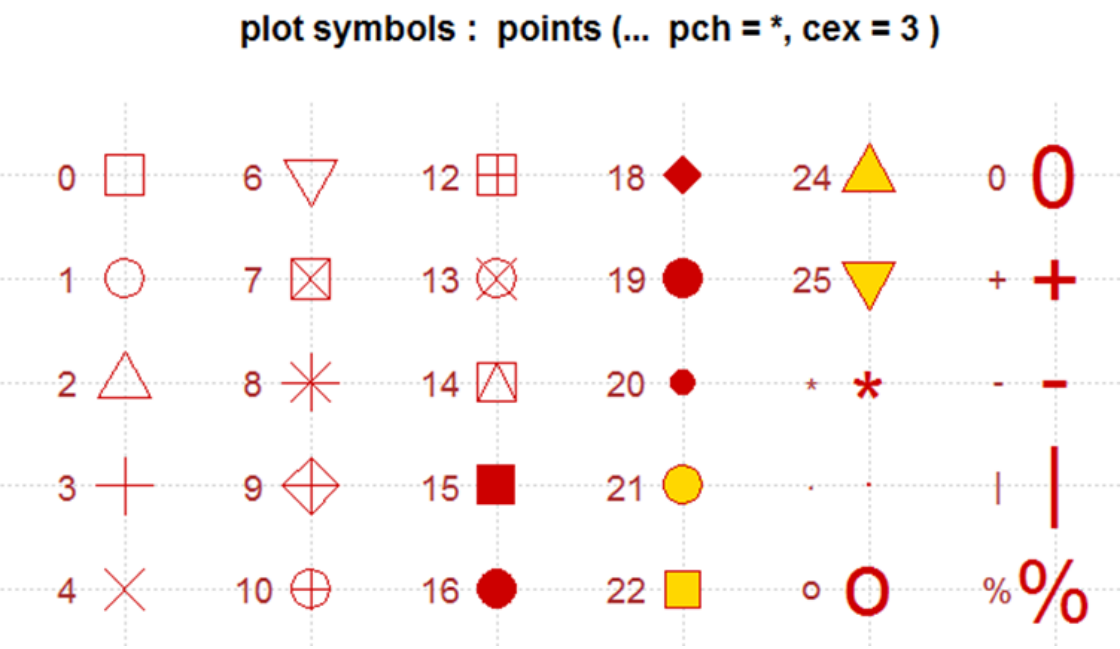
```
args(plot)
```

```
## function (x, y, ...)
## NULL
```

우선 그래프의 영역을 지칭하는 용어들 부터 익히셔야 합니다. 흔히 X 축이라고 하는 가로축은 plot 함수에는 1 번축(axis 1)으로 지칭되며, Y 축은 2 번축(axis 2)으로 지칭됩니다. 기타 그래프의 각 부분을 어떤 용어로 지칭하는지 숙지해 놓으면 나중에 많은 도움이 됩니다.



그리고, 그래프 상에서 흔히 나타나는 점의 형태는 pch 라는 인자값으로 지정이 가능합니다. 예를 들어 pch=24 라고 인자값을 주면, 내부가 황색으로 채워진 삼각형 모양의 점이 그래프상에 찍히게 됩니다. 그 옆의 cex 인자는 그 점의 크기를 조절하는 인자값입니다. Character EXpantion 의 줄임말 이라고 합니다.

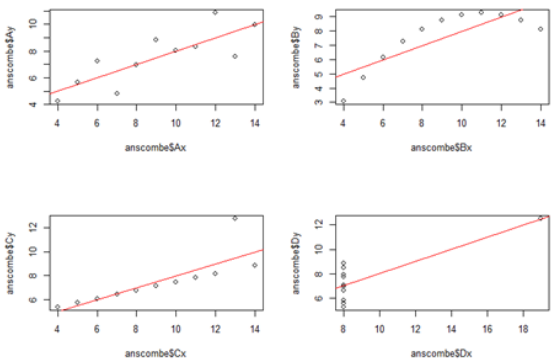


그외에도 그래프의 선의 굵기나 선의 모양을 지정하는 인자값이나 선이나 문자열의 색상을 지정하는 col 인자 등도 알아두시면 유용하게 활용이 가능하실 것입니다.

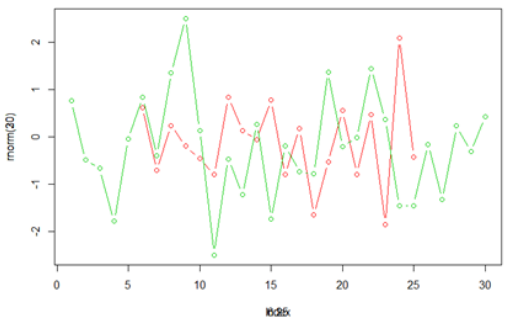
구분	주요 내용	비고
lty, lwd	Line type과 line width (숫자 혹은 문자열로 지정가능)	(0=blank, 1=solid (default), 2=dashed, 3=dotted, 4=dotdash, 5=longdash, 6=twodash) 또는 "blank", "solid", "dashed", "dotted", "dotdash", "longdash", or "twodash",
type	선그래프, 점그래프 등 지정	type=l (선그래프), type=d (선/점그래프 혼합)
cex	글자크기 조정	**정수의 숫자로 지정**
col	선이나 문자열의 색상지정	**숫자 혹은 문자열로 지정**
las	축레이블의 수직/수평여부 지정	0: always parallel to the axis [default], 1: always horizontal, 2: always perpendicular to the axis, 3: always vertical.
pdf, png	pdf 혹은 png 그림파일로 출력	File 이름 및 그림의 크기 별도 지정 필요 ※ 사용후 dev.off() 적용 필수

마지막으로 plot 함수의 인자값은 그래프 출력의 환경을 조절해 주는 par 함수가 유용합니다. 그래프 여러개를 모아서 하나의 그래프를 만들때는 par 함수의 mfrow 나 mfcol 등의 인자값을 활용하고, 한번 그린 그래프위에 추가적으로 그래프 모양을 덧씌워서 출력하고 싶은 경우는 par 함수의 new 인자를 활용합니다. 기타 상세한 plot 함수 이용법은 빅데이터 캠퍼스 분석실에 비치된 R 관련 도서들을 참조하여 주시기 바랍니다.

par(mfrow=c(2,2))



par(new=T)



매트릭스(Matrix) 소개

이제는 한줄 즉 1 차원의 한계를 벗어나는 데이터 구조를 공부하도록 하겠습니다. 가장 일반적인 형태는 여러분들이 엑셀 등을 통해서 익숙해진 표 형식 입니다. 일정한 행과 열을 가지고 있고, 누락된 부분이 없는 데이터중에서 데이터의 모든 원소가 동일한 데이터 타입인 데이터를 매트릭스라고 합니다.

매트릭스는 크게 두가지 방법으로 만듭니다. 그 하나는 직접 매트릭스를 만드는 것이고, 그 두번째는 벡터를 묶어서(bind) 만드는 방법이 있습니다.

먼저 직접 매트릭스를 생성하는 방법입니다.

매트릭스의 원소가 되는 데이터와 함께 행의 갯수(number of rows, nrow)와 열의 갯수(number of columns, ncol)를 지정하여 matrix()함수를 사용해서 만드시면 됩니다.

```
myMatrix <- matrix(1:20, nrow=4, ncol=5)
myMatrix
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    1    5    9   13   17
## [2,]    2    6   10   14   18
## [3,]    3    7   11   15   19
## [4,]    4    8   12   16   20
```

여러분들은 방금 1 차원, 즉 한줄짜리 데이터를 넘어 2 차원의 세계로 들어오셨습니다. Welcome, aboard!!!!

이제 벡터외에 매트릭스 데이터 구조를 활용가능하시게 되었으니, 객체의 내부 구조를 알아보는 명령어(함수)를 하나 배우도록 하겠습니다. 바로 str() 함수 입니다.

```
str(myMatrix)
```

```
##  int [1:4, 1:5] 1 2 3 4 5 6 7 8 9 10 ...
```

별로 감흥이 없으실텐데, 비교를 위해 벡터를 하나 만들어서 비교를 해보겠습니다.

```
myVector <- 1:20
str(myVector)

##  int [1:20] 1 2 3 4 5 6 7 8 9 10 ...
```

str()함수의 결과값은 구조가 벡터나 매트릭스가 거의 유사합니다. int 는 정수(integer) 데이터로 만들어진 벡터 혹은 매트릭스, 아니면 어레이라는 의미입니다.

다른 부분은 그 옆에 대괄호로 되어 있는 부분입니다. 보이시나요?

str(myMatrix) int [1:4, 1:5] 1 2 3 4 5 6 7 8 9 10 ... -> 4 행 5 열의 의미를 행의 갯수가 1 부터 4 까지있고, 열의 갯수가 1 부터 5 까지 있다고 하는 형태로 보여주고 있습니다. (R 은 순서에도 원칙이 있어, 해석할 수 있어야 합니다. 원칙은 행먼저, 다음이 열입니다.)

str(myVector) int [1:20] 1 2 3 4 5 6 7 8 9 10 ... -> 그냥 원소가 1 부터 20 까지 있다고만 이야기 하고 있어, 원자적 벡터임을 보여줍니다.

str()함수는 이외에도 다양한 객체의 특성을 요약해 주는 아주 유용한 함수입니다. 나중에 배우실 데이터프레임(dataframe), 리스트(list), 요인(factor)등 R 에서 활용되는 모든 객체의 구조와 특징을 보여주는 함수이니, 반드시 기억해 두시기 바랍니다.

이번에는 두번째 매트릭스 만드는 방법입니다. 예를 들어 설명하기 위해 다음과 같은 벡터가 있다고 합시다.

```
myVec_c1 <- 1:4
myVec_c2 <- 5:8
myVec_c3 <- 9:12
myVec_c4 <- 13:16
myVec_c5 <- 17:20
myVec_c6 <- 21:25
```

이러한 벡터를 행으로 합치는 함수(row bind, rbind)와 열로 합치는 함수(column bind, cbind)로 합쳐서 매트릭스를 만들 수 있습니다.

다만, 이렇게 합칠때 합쳐지는 벡터의 데이터 타입과 길이가 모두 같아야 합쳐집니다.

R 을 프로그래밍함에 있어서 순서와 길이만 잘 맞추실 줄 아시면 일단 작동하게 하는데 큰 문제는 없으실 것입니다. 순서와 길이가 중요하다는 점을 눈여겨 봐 주시기 바랍니다.

```
myMat_01 <- cbind(1:4, 5:8, 9:12, 13:16, 17:20)
myMat_01
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    1    5    9   13   17
## [2,]    2    6   10   14   18
## [3,]    3    7   11   15   19
## [4,]    4    8   12   16   20
```

이번엔 변수에 저장된 벡터를 묶어서 매트릭스를 만들면 어떻게 될까요?

```
myMat_02 <- cbind(myVec_c1, myVec_c2)   #인수는 원하는 만큼 집어넣어 주시면 됩니다.
myMat_02
```

```
##      myVec_c1 myVec_c2
## [1,]        1        5
## [2,]        2        6
## [3,]        3        7
## [4,]        4        8
```

```
str(myMat_02)
```

```
##  int [1:4, 1:2] 1 2 3 4 5 6 7 8
##  - attr(*, "dimnames")=List of 2
##   ..$ : NULL
##   ..$ : chr [1:2] "myVec_c1" "myVec_c2"
```

결과를 보시면 벡터 변수명이 컬럼의 이름(name)으로 설정되어 매트릭스가 만들어지는 것을 알수 있습니다. 그러면 3 개의 벡터를 묶어보고, 마지막으로 5 개벡터 전체를 묶어보겠습니다.

```r
myMat_03 <- cbind(myVec_c1, myVec_c2, myVec_c3)   #인수는 원하는 만큼 집어넣어 주시면 됩니다.
myMat_03
```

```
##      myVec_c1 myVec_c2 myVec_c3
## [1,]        1        5        9
## [2,]        2        6       10
## [3,]        3        7       11
## [4,]        4        8       12
```

```r
str(myMat_03)
```

```
##  int [1:4, 1:3] 1 2 3 4 5 6 7 8 9 10 ...
##  - attr(*, "dimnames")=List of 2
##   ..$ : NULL
##   ..$ : chr [1:3] "myVec_c1" "myVec_c2" "myVec_c3"
```

```r
myMat_04 <- cbind(myVec_c1, myVec_c2, myVec_c3, myVec_c4, myVec_c5)
myMat_04
```

```
##      myVec_c1 myVec_c2 myVec_c3 myVec_c4 myVec_c5
## [1,]        1        5        9       13       17
## [2,]        2        6       10       14       18
## [3,]        3        7       11       15       19
## [4,]        4        8       12       16       20
```

```r
str(myMat_04)
```

```
##  int [1:4, 1:5] 1 2 3 4 5 6 7 8 9 10 ...
##  - attr(*, "dimnames")=List of 2
##   ..$ : NULL
##   ..$ : chr [1:5] "myVec_c1" "myVec_c2" "myVec_c3" "myVec_c4" ...
```

만약에 길이(length)가 다른 벡터를 묶을려고 하시면 어떻게 될까요?

아래와 같이, 경고 메시지가 나타나게 됩니다. 왜 경고를 할까요?

정답은 벡터의 재사용 때문입니다. 차라리 에러가 나서 정지되면 좋은데, 경고만 뿌려주고 그냥 실행함으로 오히려 안 좋은 사례라 할 것입니다.

myMat_05 <- cbind(myVec_c1, myVec_c6) Warning message: In cbind(myVec_c1, myVec_c6) : number of rows of result is not a multiple of vector length (arg 1)

```
(myMat_05 <- cbind(myVec_c1, myVec_c6))
```

```
##      myVec_c1 myVec_c6
## [1,]        1       21
## [2,]        2       22
## [3,]        3       23
## [4,]        4       24
## [5,]        1       25
```

자, 그럼 컬럼명을 내가 원하는 형태로 만들려면 어떻게 하면될까요? 이럴때는 "=" 연산자를 함수 내부에서 사용하시면 됩니다.

```
myMat_05 <- cbind(V1=myVec_c1, V2=myVec_c2, V3=myVec_c3, V4=myVec_c4, V5=myVec_c5)
myMat_05
```

```
##      V1 V2 V3 V4 V5
## [1,]  1  5  9 13 17
## [2,]  2  6 10 14 18
## [3,]  3  7 11 15 19
## [4,]  4  8 12 16 20
```

```
str(myMat_05)
```

```
##  int [1:4, 1:5] 1 2 3 4 5 6 7 8 9 10 ...
##  - attr(*, "dimnames")=List of 2
##   ..$ : NULL
##   ..$ : chr [1:5] "V1" "V2" "V3" "V4" ...
```

아니면, 일단 먼저 매트릭스를 만드신 다음 컬럼명을 바꾸어 주셔도 됩니다.

```r
myMat_05_1 <- cbind(myVec_c1, myVec_c2, myVec_c3, myVec_c4, myVec_c5)
myMat_05_1 # 이름변경전
```

```
##      myVec_c1 myVec_c2 myVec_c3 myVec_c4 myVec_c5
## [1,]        1        5        9       13       17
## [2,]        2        6       10       14       18
## [3,]        3        7       11       15       19
## [4,]        4        8       12       16       20
```

```r
colnames(myMat_05_1) <- paste0("V",1:5)
myMat_05_1 # 이름변경후
```

```
##      V1 V2 V3 V4 V5
## [1,]  1  5  9 13 17
## [2,]  2  6 10 14 18
## [3,]  3  7 11 15 19
## [4,]  4  8 12 16 20
```

위의 paste0 함수는 2 개의 문자열 벡터를 원소별로 묶어 주는 함수입니다. 나중에 배우신 다음에 사용하는 것이 바람직합니다만, 일일이 c("V1","V2","V3","V4","V5") 입력하여 만들기 귀찮아서 사용했으니, 양해를 바랍니다.

paste0()함수가 첫번째 인자인 길이 1 의 문자열 벡터 "V"를 두번째 나오는 인자인 길이 5 인 정수 벡터에 맞게, 5 번 반복 사용해서 V1 부터 V5 까지 5 개의 문자열 벡터를 만드는 것을 눈여겨 봐 두시기 바랍니다.

rbind 함수는 묶어주는 방향이 행방향이라서 밑으로 벡터가 붙어서 매트릭스가 만들어집니다. cbind 와 거의 유사하나, 열의 이름이 행의 이름으로 바뀌는 부분에 주목해 주시기 바랍니다.

```r
myMat_r_01 <- rbind(1:4, 5:8, 9:12, 13:16, 17:20)
myMat_r_01
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    2    3    4
## [2,]    5    6    7    8
## [3,]    9   10   11   12
## [4,]   13   14   15   16
## [5,]   17   18   19   20
```

```
myMat_r_02 <- rbind(myVec_c1, myVec_c2)   #인수는 원하는 만큼 집어넣어 주시면 됩니다.
myMat_r_02
```

```
##           [,1] [,2] [,3] [,4]
## myVec_c1     1    2    3    4
## myVec_c2     5    6    7    8
```

```
str(myMat_r_02)
```

```
##  int [1:2, 1:4] 1 5 2 6 3 7 4 8
##  - attr(*, "dimnames")=List of 2
##   ..$ : chr [1:2] "myVec_c1" "myVec_c2"
##   ..$ : NULL
```

```
myMat_r_03 <- rbind(myVec_c1, myVec_c2, myVec_c3)   #인수는 원하는 만큼 집어넣어 주시면 됩니다.
myMat_r_03
```

```
##           [,1] [,2] [,3] [,4]
## myVec_c1     1    2    3    4
## myVec_c2     5    6    7    8
## myVec_c3     9   10   11   12
```

```
str(myMat_r_03)
```

```
##  int [1:3, 1:4] 1 5 9 2 6 10 3 7 11 4 ...
##  - attr(*, "dimnames")=List of 2
##   ..$ : chr [1:3] "myVec_c1" "myVec_c2" "myVec_c3"
##   ..$ : NULL
```

```r
myMat_r_04 <- rbind(myVec_c1, myVec_c2, myVec_c3, myVec_c4, myVec_c5)
myMat_r_04
```

```
##           [,1] [,2] [,3] [,4]
## myVec_c1     1    2    3    4
## myVec_c2     5    6    7    8
## myVec_c3     9   10   11   12
## myVec_c4    13   14   15   16
## myVec_c5    17   18   19   20
```

```r
str(myMat_r_04)
```

```
##  int [1:5, 1:4] 1 5 9 13 17 2 6 10 14 18 ...
##  - attr(*, "dimnames")=List of 2
##   ..$ : chr [1:5] "myVec_c1" "myVec_c2" "myVec_c3" "myVec_c4" ...
##   ..$ : NULL
```

cbind 의 경우와 같이 행이름도 함수를 활용하여 변경이 가능합니다. 다시한번 "순서와 길이"에 주의를 기울여 주시기 바랍니다.

```r
rownames(myMat_r_04) <- paste0("R",1:5)
colnames(myMat_r_04) <- paste0("V",1:4)
myMat_r_04  # 이름 변경후
```

```
##    V1 V2 V3 V4
## R1  1  2  3  4
## R2  5  6  7  8
## R3  9 10 11 12
## R4 13 14 15 16
## R5 17 18 19 20
```

루프 함수

만약에 myMat_r_04 객체의 행별 합계를 구한다면 어떻게 하시겠습니까?

벡터의 경우와 다르게 매트릭스의 경우는 대괄호를 콤마(",")로 구분하여 행과 열을 지정, 매트릭스의 부분집합을 지정해서 인덱싱하는 것이 가능합니다.

```
myMat_r_04
```

```
##    V1 V2 V3 V4
## R1  1  2  3  4
## R2  5  6  7  8
## R3  9 10 11 12
## R4 13 14 15 16
## R5 17 18 19 20
```

```
myMat_r_04[2,4] # 매트릭스의 2 행 4 열 지정
```

```
## [1] 8
```

```
myMat_r_04[1:2, 2:3] #매트릭스의 1,2 행과 2,3 열을 모두 지정
```

```
##    V2 V3
## R1  2  3
## R2  6  7
```

벡터와 마찬가지로 콤마(",")의 앞과 뒤를 공란으로 남겨두면, 모든 행원소 혹은 모든 열원소를 지칭하는 것이 됩니다.

```
myMat_r_04
```

```
##    V1 V2 V3 V4
## R1  1  2  3  4
## R2  5  6  7  8
## R3  9 10 11 12
## R4 13 14 15 16
## R5 17 18 19 20
```

```
myMat_r_04[3,] # 매트릭스의 3 행 모두 지정

## V1 V2 V3 V4
##  9 10 11 12

myMat_r_04[, 3:4] #매트릭스의 3,4 열을 모두 지정

##    V3 V4
## R1  3  4
## R2  7  8
## R3 11 12
## R4 15 16
## R5 19 20
```

이제 행이나 열의 합계를 구한다면 for 문을 사용하면 되겠지요? 매트릭스의 행의 갯수나 열의 갯수는 nrow 와 ncol 함수로 구할 수 있습니다.

```
nrow(myMat_r_04)

## [1] 5

ncol(myMat_r_04)

## [1] 4
```

for 문을 돌려서 행의 합계를 만들어 보겠습니다. 아래의 내용이 이해가 가시면 좋겠습니다만, 힘드신 분들이 많이 계시리라 예상합니다. 힘드신 분들은 이부분을 건너뛰어도 좋을 것 같습니다.

```
for(iVar in 1:nrow(myMat_r_04)) {
  cat(iVar,"번째 행의 합은 ",sum(myMat_r_04[iVar,])," 입니다.","₩n")
}

## 1 번째 행의 합은  10  입니다.
## 2 번째 행의 합은  26  입니다.
## 3 번째 행의 합은  42  입니다.
```

```
## 4 번째 행의 합은  58  입니다.
## 5 번째 행의 합은  74  입니다.
```

열의 합계를 구하기 위해서는 for 순서지정 괄호내의 함수가 ncol 로 바뀌고, 명령어 Expression 의 위치가 매트릭스 인덱싱에서 열의 자리로 바뀐 것에 주목하여 주시기 바랍니다.

그리고 for loop 의 매개변수는 관례적으로 i 나 j 를 사용합니다만, 변수명에 적합한 어떠한 것이든지 사용가능합니다. 이경우는 for 순서지정 괄호내의 매개변수명(iVar)과 명령어 Expression 의 매개변수명(myMat_r_04[,iVar]의 iVar 를 말합니다.)을 일치 시키기만 하면 사용이 가능합니다.

```
for(iVar in 1:ncol(myMat_r_04)) {
  cat(iVar,"번째 열의 합은 ",sum(myMat_r_04[,iVar])," 입니다.","₩n")
}

## 1 번째 열의 합은  45  입니다.
## 2 번째 열의 합은  50  입니다.
## 3 번째 열의 합은  55  입니다.
## 4 번째 열의 합은  60  입니다.
```

만약에 위와 같이 형식을 갖춘 출력문이 필요없이 그저 행과 열의 합계가 궁금할 경우, for 문을 작성하는 것이 귀찮을 경우가 많습니다. 그럴 때 쓰라고 만들어둔 함수가 apply() 함수입니다. for 문의 반복적인 루프 적용을 그냥 함수로 실행되게 만든 것이라고 보시면 되겠습니다.

다양한 apply 함수가 있어 "apply 함수군"이라고 일컬어 지기도 합니다만, 원리는 apply 함수와 모두 유사합니다.

일단 문법은 다음과 같습니다.

apply(Matrix, 행과 열 지정 숫자(1:행방향, 2:열방향)), 반복적용할 함수명)

```
apply(myMat_r_04, 1, sum) # 1 은 행을 의미하므로 매트릭스의 행 수치의 합산결과를 보여줍니다.
```

```
## R1 R2 R3 R4 R5
## 10 26 42 58 74

apply(myMat_r_04, 2, sum) # 2 는 열을 의미하므로 매트릭스의 열 수치의 합산결과를 보여줍니다.

## V1 V2 V3 V4
## 45 50 55 60
```

투입되는 데이터 객체의 행이름과 열이름이 없는 경우는 아래와 같이 출력됩니다.

```
apply(myMat_04, 1, sum)  # 1 은 행을 의미하므로 매트릭스의 행 수치의 합산결과를 보여줍니다.

## [1] 45 50 55 60

apply(myMat_04, 2, sum) # 2 는 열을 의미하므로 매트릭스의 열 수치의 합산결과를 보여줍니다.

## myVec_c1 myVec_c2 myVec_c3 myVec_c4 myVec_c5
##       10       26       42       58       74
```

jpeg 패키지 소개

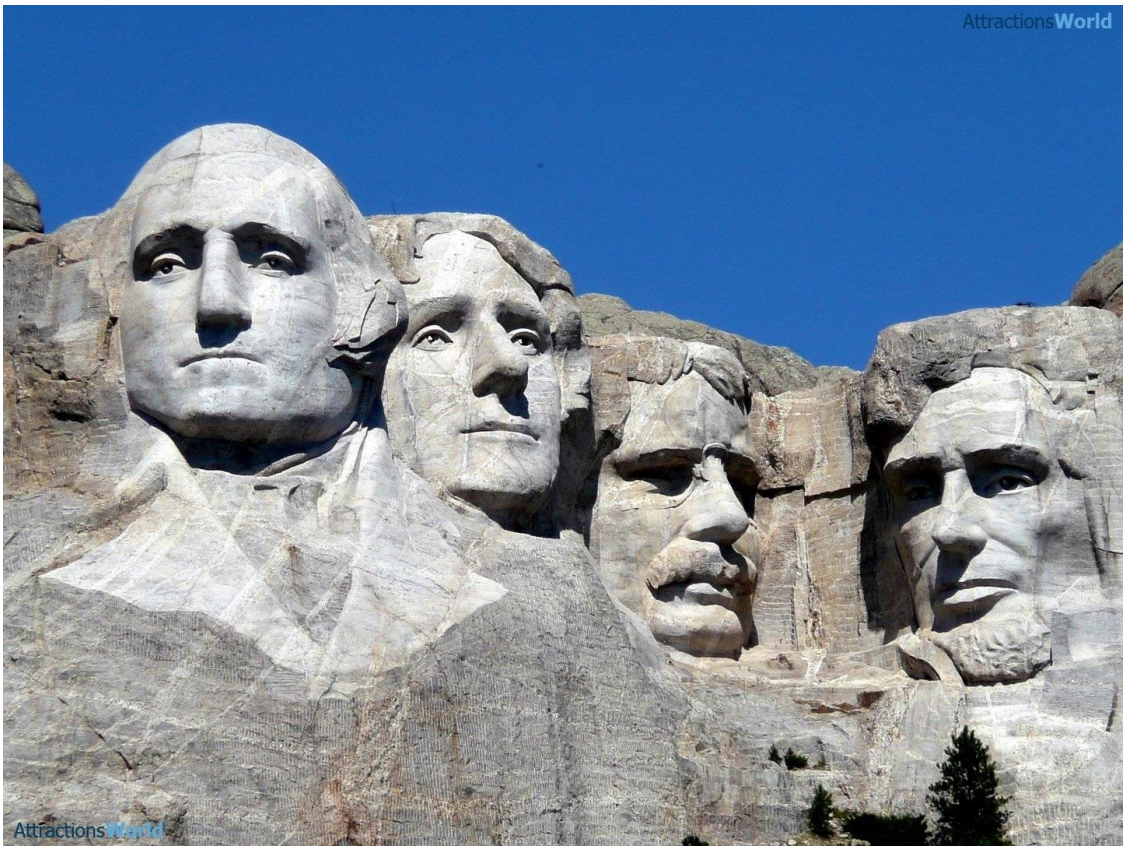
jpeg 패키지는 jpg 포맷의 그림파일(image)을 픽셀별 색상구성으로 변환하여 수정 혹은 변경이 가능하도록 Array 형태로 만들어 주는 패키지 입니다.

R Studio 에서 install.packages("jpeg")로 설치하시면 바로 설치가 진행됩니다. jpeg 패키지는 아래와 같이 2 개의 함수가 있습니다. jpeg 파일을 읽어들이는 함수(readJPEG) 그리고 작업후에 완료된 데이터 객체를 다시 jpg 이미지 파일로 만들어 주는 함수(writeJPEG) 입니다.

```
library(jpeg)
ls("package:jpeg") #패키지 적용으로 사용가능한 함수를 모두 화면에 출력해 줍니다.
```

```
## [1] "readJPEG"   "writeJPEG"
```

실습용 데이터 파일은 현재 작업디렉토리 (working 디렉토리)에 저장해 두는 것을 전제로 실습을 진행하고자 합니다.




```
mtrushBMP <- readJPEG("./image/rushmore_mountain.jpg")
str(mtrushBMP)

##  num [1:1080, 1:1440, 1:3] 0.1098 0.1098 0.1059 0.0941 0.0941 ...

class(mtrushBMP)

## [1] "array"

dim(mtrushBMP)

## [1] 1080 1440    3
```

```
range(mtrushBMP[1, ,1])
```

```
## [1] 0.06666667 0.15686275
```

```
max(mtrushBMP)
```

```
## [1] 1
```

```
min(mtrushBMP)
```

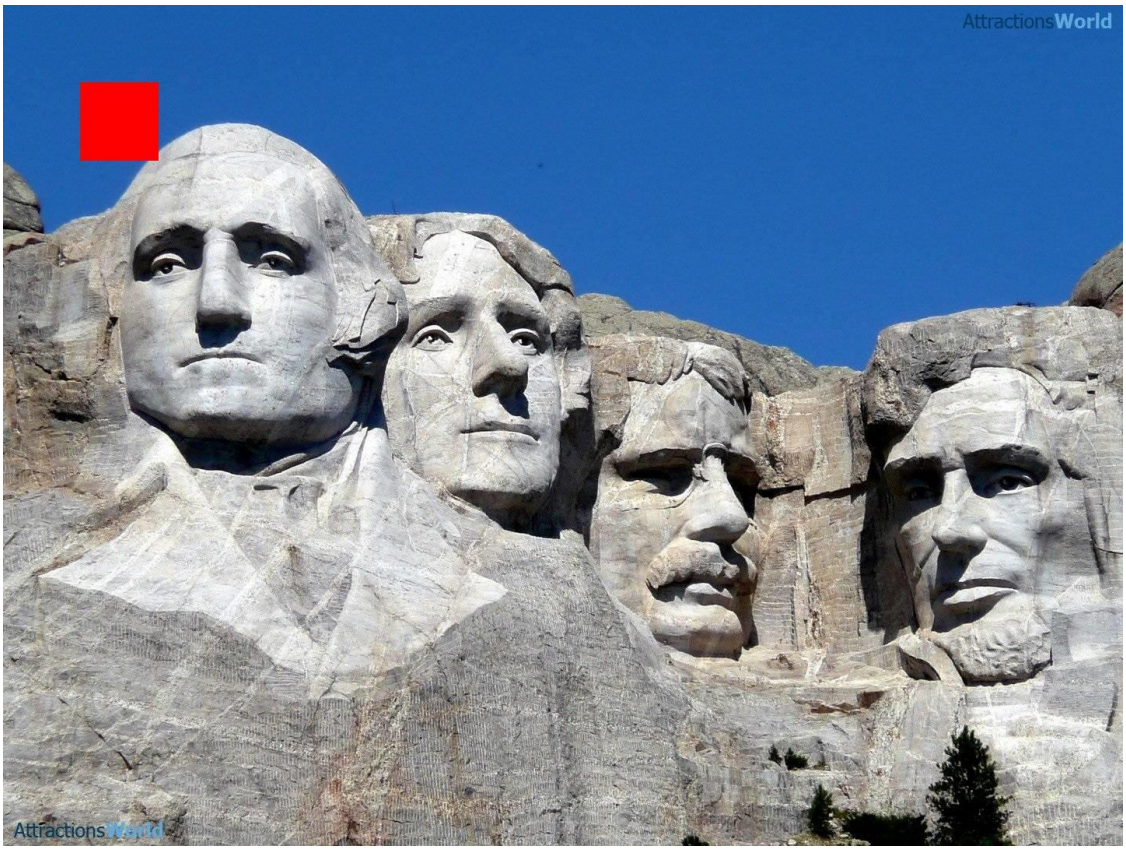
```
## [1] 0
```

아래와 같이 코딩을 하면 위의 그림 중 특정부분, 즉 가로의 100 픽셋부터 200 픽셀까지, 그리고 세로의 100 픽셀부터 200 픽셀까지의 사각형 부분이 적색으로 변하게 됩니다.

직접 실습으로 확인하시기 바랍니다.

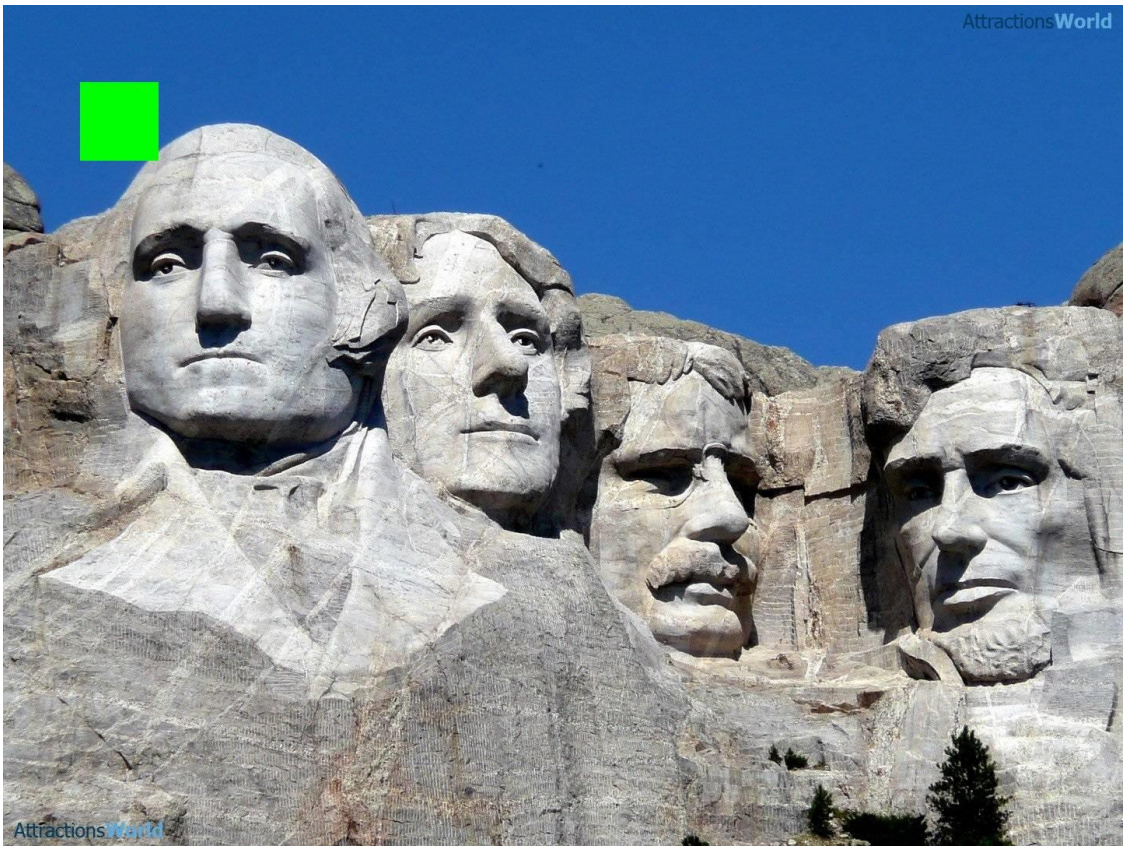
아래 코드의 대괄호안의 콤마는 행과 열, 그리고 차원구분을 나눠주는 구분자 역할을 합니다.

```
mtrushBMP[100:200,100:200,1] <- 1  # Red
mtrushBMP[100:200,100:200,2] <- 0  # Green
mtrushBMP[100:200,100:200,3] <- 0  # Blue

# writeJPEG 함수의 아규먼트중 target 키워드로 이미지 파일명을 지정하여 저장 가능

writeJPEG(mtrushBMP, target="mtrushJPG_red.jpg")
```



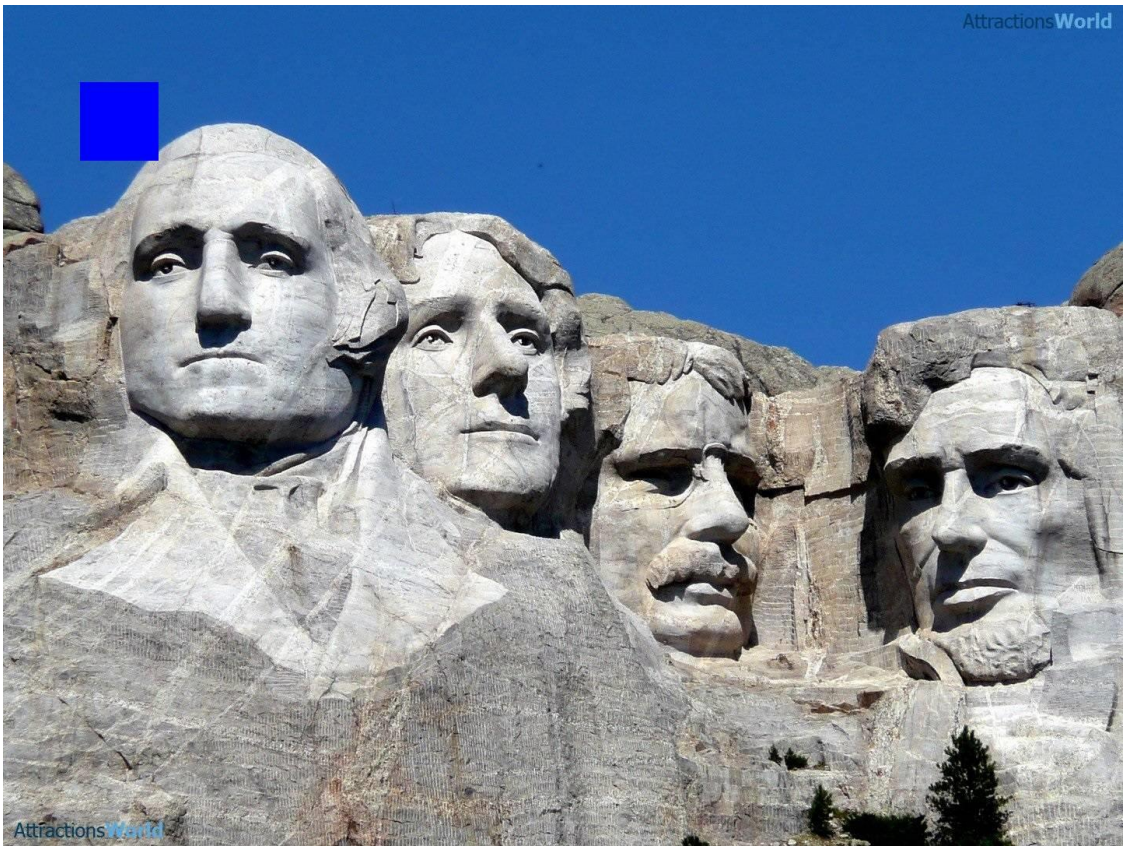

```
mtrushBMP[100:200,100:200,1] <- 0  # Red
mtrushBMP[100:200,100:200,2] <- 1  # Green
mtrushBMP[100:200,100:200,3] <- 0  # Blue

# writeJPEG 함수의 아규먼트중 target 키워드로 이미지 파일명을 지정하여 저장 가능

writeJPEG(mtrushBMP, target="mtrushJPG_green.jpg")
```

아래와 같이 코딩을 하면 위의 특정부분, 즉 가로의 100 픽셋부터 200 픽셀까지, 그리고 세로의 100 픽셀부터 200 픽셀까지의 사각형 부분이 녹색으로 변하게 됩니다.



다시 가로의 100 픽셋부터 200 픽셀까지, 그리고 세로의 100 픽셀부터 200 픽셀까지의 사각형 부분이 파란색으로 변하게 되는 경우입니다.

```
mtrushBMP[100:200,100:200,1] <- 0  # Red
mtrushBMP[100:200,100:200,2] <- 0  # Green
mtrushBMP[100:200,100:200,3] <- 1  # Blue

# writeJPEG 함수의 아규먼트중 target 키워드로 이미지 파일명을 지정하여 저장 가능

writeJPEG(mtrushBMP, target="mtrushJPG_blue.jpg")
```



for 루프를 이용해서 그림에 사선을 그리는 부분도 아래와 같이 가능합니다. 직접 코딩을 하시면서 느껴 보시기 바랍니다.

```
library(jpeg)

start.time <- Sys.time()
count <-0
totalcount <- 0
mtrushBMP <- readJPEG("rushmore_mountain.jpg")
for (i in 1:dim(mtrushBMP)[1]) {
  for (j in 1:dim(mtrushBMP)[2]) {
    if ( (i - j) %% 50 == 0 ) {
      mtrushBMP[i,j,1] <-1
      mtrushBMP[i,j,2] <-1
      mtrushBMP[i,j,3] <-1
```

```
      count = count + 1
    }
    totalcount <- totalcount + 1
  }
}
writeJPEG(mtrushBMP,target="mtrushJPG_cross.jpg")
```

최종 Wrap-Up

이상 간략하게 R Tutorial 을 마치도록 하겠습니다. 추가적인 학습은 저희 빅데이터 캠퍼스안에 비치되어 있는 R 교재들을 참조하여 주시기 바랍니다.

본 R Tutorial 을 위해 사용한 컴퓨터 환경 및 R 의 패키지 내역은 아래와 같습니다.

```
sessionInfo()
```

```
## R version 3.3.2 (2016-10-31)
## Platform: x86_64-w64-mingw32/x64 (64-bit)
## Running under: Windows 8.1 x64 (build 9600)
##
## locale:
## [1] LC_COLLATE=Korean_Korea.949  LC_CTYPE=Korean_Korea.949
## [3] LC_MONETARY=Korean_Korea.949 LC_NUMERIC=C
## [5] LC_TIME=Korean_Korea.949
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## other attached packages:
## [1] jpeg_0.1-8
##
## loaded via a namespace (and not attached):
##  [1] backports_1.0.4 magrittr_1.5    rprojroot_1.1   tools_3.3.2
##  [5] htmltools_0.3.5 yaml_2.1.14     Rcpp_0.12.8     stringi_1.1.2
##  [9] rmarkdown_1.3   knitr_1.15.1    stringr_1.1.0   digest_0.6.10
## [13] evaluate_0.10
```

서울시 빅데이터 캠퍼스와 함께 즐거운 분석 경험을 앞으로도 계속 쌓아가시길 진심으로 바라고 원합니다.