

# 단순 반복 업무를 위한 for문과 apply류 맛보기

## 조건문

if

if는 if (조건) {조건이 true 이면 실행할 부분}으로 구성됩니다.

```
if(TRUE){print(1)}
```

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

```
print(2)
```

```
## [1] 2
```

조건은 결과가 하나의 logical 값으로 나와야 하고 여러 개의 logical 값이면 맨 앞의 값만 사용한다는 warning 을 같이 출력합니다.

```
if(c(T,F,F,F)){print(1)}
```

```
## Warning in if (c(T, F, F, F)) {: length > 1 이라는 조건이 있고, 첫번째 요소  
## 만이 사용될 것입니다
```

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

```
print(2)
```

```
## [1] 2
```

보통은 아래와 같은 형식으로 사용합니다.

```
x<-1  
if ( x > 0 ){  
  print(1)  
}
```

이제까지 조건이라고 말하는 것이 있었는데, 조건이란 TRUE, FALSE로 결과가 나오는 표현 전부를 뜻합니다. 제가 if문에서 나올 만한 예시를 준비했습니다.

```
x<-c()  
if(identical(x,c())){print("x has no data.")}
```

```
## [1] "x has no data."
```

```
print("if part done.")
```

```
## [1] "if part done."
```

```
options(stringsAsFactors = F)  
  
y<-c(1,2,3)  
z<-c(1,2,3)  
if(length(y)==length(z)){  
  tem<-data.frame(y,z)  
  print(tem)  
}
```

```
##   y z  
## 1 1 1  
## 2 2 2  
## 3 3 3
```

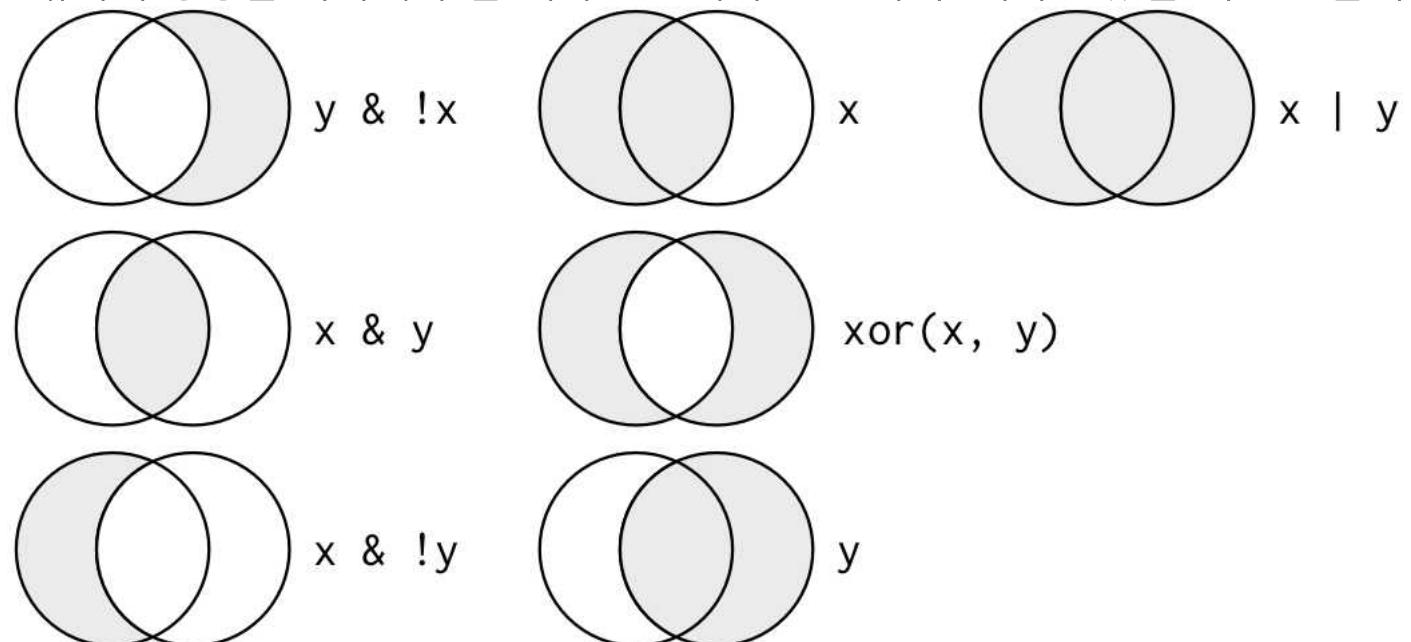
```
print("if part done.")
```

```
## [1] "if part done."
```

`identical`은 두 개의 변수를 비교해주고 같으면 TRUE, 다르면 FALSE를 결과로 주는 함수입니다. 결과를 logical로 주는 덕분에 조건문에 사용하기 딱 좋은 함수입니다. 예를 들어 데이터에 무언가 문제가 생겨서 함수에 들어 가지 못하거나 할때 우회하는 조건을 작성하는데 좋습니다. 저같은 경우는 N2H4 패키지를 작성할 때 `getContent`에서 사용했습니다. 물론 이상적으로는 정규식과 네이버 뉴스의 root url을 바탕으로 비교하는 식으로 해야 더 정교하겠습니다만, `getUrlListByCategory` 함수에서 생성되는 link를 사용하는 형태도 구성되어 있어서 아래와 같이 작성하였습니다.

```
...
if(!identical(url,character(0))){
  if (RCurl::url.exists(url)&
    "error_msg 404"!=read_html(url)%>%html_nodes("div#main_content div div")%>%html_attr("class"))[1]
  ) {
...
}
```

조건을 여러개 두고 and나 or로 묶어서 상황을 파악해야 할 때가 있습니다. 조건이 두 개라고 했을 때 참고할 수



있는 그림이 아래와 같습니다.

**else**

영어 표현을 보면 생각하기 쉬우시겠지만 if(조건){조건이 true 이면 실행할 부분} 이후에 사용해서 조건이 false면 실행할 부분을 작성하는데 사용합니다.

if (조건) {조건이 true 이면 실행할 부분} else {조건이 false 면 실행할 부분}으로 구성됩니다.

```
if(TRUE){  
  print(1)  
} else {  
  print(3)  
}
```

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

```
print(2)
```

```
## [1] 2
```

else는 앞에 if가 조건을 작성했기 때문에 추가적인 조건을 작성하지는 않습니다. 여러 if 조건을 사용하고 그 이후에 else를 사용할 수도 있습니다.

```
x<-1  
  
if(x<0){  
  print(1)  
}  
if(x>10) {  
  print(3)  
} else {  
  print(2)  
}
```

```
## [1] 2
```

```
print(4)
```

```
## [1] 4
```

최근의 코드 작성 스타일은 누가 봐도 읽고 이해하기 쉽게 이다 보니 고려하면 좋을 것 같습니다. 특히 이 스타일은 자기 자신에게도 적용이 되어서, 나중에 봐도 기억하기 쉽게 작성하는 것이 좋습니다.

**ifelse**

`ifelse`는 앞에 함수와는 다른 결과를 제공해서 사용하는 곳이 다릅니다. 우선 형태는 `ifelse(조건, 조건이 true일때 할 것, 조건이 false일때 할 것)`으로 구성됩니다. 그래서 기존의 데이터를 새로운 기준으로 조정해서 사용할 때 많이 사용합니다.

```
library(readr)
sd<-read_csv("./제3회 Big Data Competition-분석용데이터-05.멤버십여부.txt")
```

```
## Parsed with column specification:  
## cols(  
##   고객번호 = col_character(),  
##   멤버십명 = col_character(),  
##   가입년월 = col_integer()  
## )
```

```
names(sd)[1]<- "고객번호"  
sd<-data.frame(sd)  
  
str(sd)
```

```
## 'data.frame': 7456 obs. of 3 variables:  
## $ 고액번호: chr "00011" "00021" "00037" "00043" ...  
## $ 멤버십명: chr "하이마트" "하이마트" "하이마트" "하이마트" ...  
## $ 가입년월: int 201512 201506 201306 201403 201411 201312 201506 201404 201406 201311 ...
```

summary(sd)

```
##
```

```
3rd Qu.:201504  
Max. :201512
```

```
sd$"최근고객 "<-ifelse(sd$"가입년월">mean(sd$"가입년월"), "최근", "최근아님")  
head(sd)
```

```
## 고객번호 멤버십명 가입년월 최근고객  
## 1 00011 하이마트 201512 최근  
## 2 00021 하이마트 201506 최근  
## 3 00037 하이마트 201306 최근아님  
## 4 00043 하이마트 201403 최근아님  
## 5 00044 하이마트 201411 최근아님  
## 6 00061 하이마트 201312 최근아님
```

관련해서 [여기](#)를 가보시면 for문에 대한 간략한 방법을 질문하시고, 댓글로 여러 답변이 달렸는데, ifelse가 가장 좋은 해결책으로 보입니다. 확인해보세요.

## try

try는 error를 우회하거나 활용하기 위해서 사용하는 함수입니다. 직접 사용할 일은 많지 않지만 함수의 실행에서 에러가 났을 때 (ex> data.frame은 데이터의 길이가 다르면 변수를 만들지 못하고 에러를 출력합니다.) 에러가 난 부분만 기록하고 넘기는 형태로 코드를 작성 할 수 있습니다. 더 섬세한 기능의 tryCatch 도 있으니 ?tryCatch를 확인해주세요.

```
no0bj  
print(1)  
## Error in try(no0bj) : object 'no0bj' not found
```

```
try(no0bj)  
print(1)
```

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

콘솔에서 실행하면 try(no0bj)에서 에러가 발생합니다. 하지만 멈추는 것이 아니라 다음 코드를 실행하는 것이 그냥 no0bj를 코드에 작성한 것과 차이점입니다. try를 입력해 보시면 select 옵션이 있는데 TRUE로 해주면 에러 출력도 하지 않습니다.

```
err<-try(noObj)
err
```

```
## [1] "Error in try(noObj) : 객체 'noObj'를 찾을 수 없습니다\n"
## attr(,"class")
## [1] "try-error"
## attr(,"condition")
## <simpleError in doTryCatch(return(expr), name, parentenv, handler): 객체 'noObj'를 찾을 수 없습니다>
```

```
class(err)
```

```
## [1] "try-error"
```

위와 같은 식으로 try(함수)를 변수에 선언하면 class(변수)를 통해 조건문을 활용해서 에러가 발생했을 때를 직접적으로 우회할 수 있습니다.

## 반복문

### repeat

repeat은 가장 단순한 형태의 반복 구분입니다. 그냥 repeat만 사용할 수도 있습니다만, repeat(print(1))을 실행하면 무한히 1을 출력하고 멈추지 않습니다. 강제로 멈추는 활동을 해주어야만 멈추니 주의해 주세요. 그래서 break 문법이 준비되어 있습니다.

### break

break는 말 그대로 멈추라는 명령입니다. break는 독특하게 뒤에 ()를 붙이지 않고 활용하는 함수로 조건문이나 반복문 안에 쓰여서 조건문과 반복문을 멈추는 역할을 합니다.

```
x<-1
repeat(
  if(x>10){
    break
  } else {
```

```
print(x)
x<-x+1
}
)
```

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

## while

사실 repeat문은 사용법이 조금 길어서 잘 사용하지 않습니다. 기능적인 대체는 while문으로 가능한데, while은 while(조건){조건이 true인 동안 할 것}으로 구성됩니다.

```
x<-1
while(x<10){
  print(x)
  x<-x+1
}
```

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

위의 코드가 아까 repeat으로 만든 식과 같은 결과를 보여줍니다. 안의 조건이 달라서 이해가 어려우실 수 있어서 repeat을 다시 작성해보겠습니다.

```
x<-1  
repeat(  
  if(x<10){  
    print(x)  
    x<-x+1  
  } else {  
    break  
  }  
)
```

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

자유도가 높은 `repeat`에 비해서 `while`은 꽤 안의 조건이 TRUE일 때 동안만 동작합니다. 하지만 한 방법으로만 고정되어 있어서 오히려 혼란을 막고, 코드가 읽기 좋게 작성할 수 있는 장점이 있습니다.

## for

`for`는 반복하는 내용을 쉽게 다루기 위해서 준비되어 있습니다. 예를 들어서 위에서 `while`로 작성된 것을 `for`로 다시 작성해 보겠습니다.

```
for(x in 1:9){  
  print(x)  
}
```

```
## [1] 1  
## [1] 2  
## [1] 3  
## [1] 4  
## [1] 5  
## [1] 6  
## [1] 7
```

```
## [1] 8  
## [1] 9
```

while에 비해 훨씬 간결해 졌습니다. 이해하기도 좋구요.

for(반복에 사용할 변수 in 반복에 사용할 변수에 넣을 데이터를 가지는 벡터){반복 실행할 내용 - 반복에 사용의 형태로 사용합니다. 말로 풀어 쓰려니 오히려 어려워 보이는 것 같네요. 몇 가지 예시를 더 들어 보겠습니다.

```
data<-head(sd$"고객번호")  
for(cNum in data){  
  print(sd[sd$"고객번호"==cNum,])  
}
```

```
## 고객번호 멤버십명 가입년월 최근고객  
## 1 00011 하이마트 201512 최근  
## 2 00021 하이마트 201506 최근  
## 3 00037 하이마트 201306 최근아님  
## 4 00043 하이마트 201403 최근아님  
## 5 00044 하이마트 201411 최근아님  
## 6 00061 하이마트 201312 최근아님
```

N2H4의 사용예시도 복잡하게 무려 5중 for문(!)으로 구성되어 있습니다. 중간에 while, try, if도 다 사용되었으니 설명해 드리겠습니다.

## next

에러에 대해 우회하는 것에 대해서 조건문을 주는 방법을 설명드렸었습니다. next는 break와 비슷하지만 조건문이나 반복문을 멈추는 것이 아니라 다음 번으로 넘기는 역할을 합니다. 예를 들면 아래와 같습니다.

```
data<-head(sd$"고객번호")  
for(cNum in data){  
  if(sd[sd$"고객번호"==cNum, "최근고객"]== "최근"){next}  
  print(sd[sd$"고객번호"==cNum,])  
}
```

```

## 고객번호 멤버십명 가입년월 최근고객
## 3 00037 하이마트 201306 최근아님
## 고객번호 멤버십명 가입년월 최근고객
## 4 00043 하이마트 201403 최근아님
## 고객번호 멤버십명 가입년월 최근고객
## 5 00044 하이마트 201411 최근아님
## 고객번호 멤버십명 가입년월 최근고객
## 6 00061 하이마트 201312 최근아님

```

출력된 내용을 보면 `sd[sd$"고객번호"==cNum, "최근고객"]=="최근"`일 때 다음 줄에 있는 `print`를 하지 않고 다음(`next`)으로 넘어간 것을 확인할 수 있습니다. 이걸 통해서 조건에 따라 그 아래 내용을 실행하지 않고 다음 번 반복으로 넘기는 것이 가능합니다.

[N2H4](#)의 사용예시에는 `next`를 사용하지 않고 `while`을 사용했는데, 크롤링 특성상 요청이 일부 실패도 할 수 있기 때문에 추가적인 시도를 하기 위해서 사용했습니다. 데이터를 전부 가져오는 것이 많이 중요하지 않다면 `next`를 사용하는 것이 더 간편하고 빠르게 작성하는 방법이 될 것 같습니다.

지금 예시를 눈으로 보여드리기 위해 `for`문 안을 `print`로 계속 채우고 있는데, `print`의 위치에 수행하고자 하는 함수를 작성하시면 됩니다.

```

X<-as.data.frame(matrix(1:64, ncol=4, dimnames=list(seq(1:16), c("a", "b", "c", "d"))))

X$a[c(1,3,10)]<-0

for (i in 1:nrow(X)){
  if (X$a[i]==0) {
    X$e[i]<-(-999)
  } else {
    X$e[i]<-X$b[i]/X$c[i]
  }
}
X

```

```

##   a   b   c   d         e
## 1  0 17 33 49 -999.0000000
## 2  2 18 34 50     0.5294118
## 3  0 19 35 51 -999.0000000

```

```
## 4 4 20 36 52 0.5555556
## 5 5 21 37 53 0.5675676
## 6 6 22 38 54 0.5789474
## 7 7 23 39 55 0.5897436
## 8 8 24 40 56 0.6000000
## 9 9 25 41 57 0.6097561
## 10 0 26 42 58 -999.0000000
## 11 11 27 43 59 0.6279070
## 12 12 28 44 60 0.6363636
## 13 13 29 45 61 0.6444444
## 14 14 30 46 62 0.6521739
## 15 15 31 47 63 0.6595745
## 16 16 32 48 64 0.6666667
```

`ifelse` 함수에서 소개했던 질문쪽의 `for`로 작성된 코드입니다. `for`로 작성하는 것이 사실 생각하기 쉬운 방법이라고 생각합니다. 저는 심지어 처음에는 `for`로 작성하라고 권장합니다. 문제를 직접 겪고, 그 문제를 해결하는 방법을 찾으려할 때 그 방법이 더 몸에 남는 것 같습니다.

아래 `apply`를 하기 전에 `for`와 `ifelse`가 얼마나 다른지 한 번 비교해 보겠습니다.

```
library(ggplot2)
library(dplyr)
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:lubridate':
##
##     intersect, setdiff, union
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
library(tidyr)

times<-c(100,1000,10000,30000,50000,100000)
tData<-c()
```

```
for(tm in times){

X<-as.data.frame(matrix(1:tm, ncol=4, dimnames=list(seq(1:(tm/4)), c("a", "b", "c", "d"))))

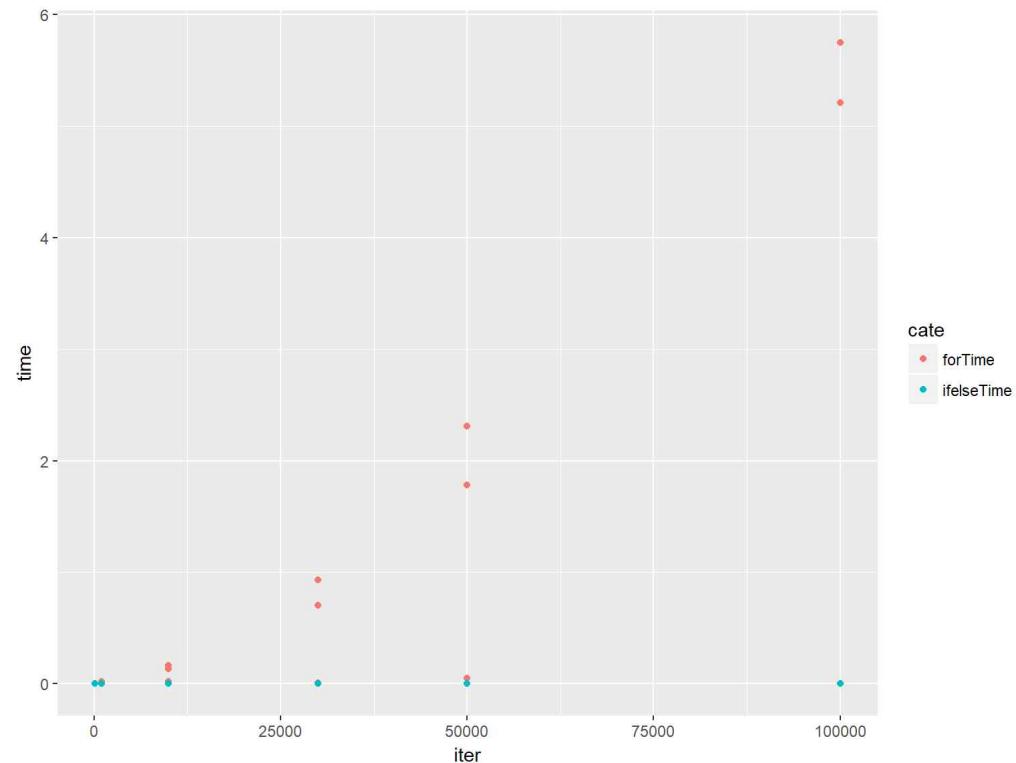
X$a[c(1,3,10)]<-0

forTime<-system.time(
  for (i in 1:nrow(X)){
    if (X$a[i]==0) {
      X$e[i]<-(-999)
    } else {
      X$e[i]<-X$b[i]/X$c[i]
    }
  }
)
ifelseTime<-system.time(X$e <- ifelse(X$a == 0, -999, X$b/X$c))

forTime<-cbind(data.frame(tm,cate="forTime"),t(as.matrix(forTime)))
ifelseTime<-cbind(data.frame(tm,cate="ifelseTime"),t(as.matrix(ifelseTime)))
tData<-rbind(tData,forTime,ifelseTime)

}

tData<-tData %>% select(tm:elapsed) %>% gather(tm,cate)
names(tData)<-c("iter","cate","timeName","time")
ggplot(tData,aes(x=iter,y=time,fill=cate,color=cate)) + geom_point(stat="identity")
```



```
tm<-1000000
X<-as.data.frame(matrix(1:tm, ncol=4, dimnames=list(seq(1:(tm/4)), c("a", "b", "c", "d"))))
ifelseTime<-system.time(X$a <- ifelse(X$a == 0, -999, X$b/X$c))
ifelseTime
```

```
##    user  system elapsed
##    0.03   0.00   0.03
```

이렇게 `for`를 사용하지 않고 다른 방법을 사용하는 것으로 벡터연산이 있습니다. 이름이 중요하진 않으니 R이 감당할 수 있는 수준의 데이터를 `apply`를 통해서 다루는 법을 알아보겠습니다.

## apply류의 함수들

### apply

apply 함수에 대해서 알아보겠습니다. apply는 행이나 열 방향의 데이터를 한 번에 계산하는데 사용합니다.

```
set.seed(1)
( myMat <- matrix(round(rnorm(16,10),2),4,4) )
```

```
##      [,1] [,2] [,3] [,4]
## [1,]  9.37 10.33 10.58  9.38
## [2,] 10.18  9.18  9.69  7.79
## [3,]  9.16 10.49 11.51 11.12
## [4,] 11.60 10.74 10.39  9.96
```

위의 mymat에서 각 열의 평균을 구하고 싶으면 이렇게 하면 됩니다.

```
mean(myMat[,1])
```

```
## [1] 10.0775
```

```
mean(myMat[,2])
```

```
## [1] 10.185
```

```
mean(myMat[,3])
```

```
## [1] 10.5425
```

```
mean(myMat[,4])
```

```
## [1] 9.5625
```

우리는 for를 배웠으니 좀 고쳐 봅시다.

```
for(i in 1:4){
  mean(myMat[,i])}
```

```
}
```

이게 또 데이터가 따로따로라 c도 해줘야 하는군요.

```
myMean <- c(  
  mean(myMat[,1]),  
  mean(myMat[,2]),  
  mean(myMat[,3]),  
  mean(myMat[,4])  
)  
myMean
```

```
## [1] 10.0775 10.1850 10.5425 9.5625
```

for를 사용하면 이렇게 됩니다.

```
myMean <- c()  
for(i in 1:4){  
  myMean<-c(myMean,mean(myMat[,i]))  
}  
myMean
```

```
## [1] 10.0775 10.1850 10.5425 9.5625
```

여기서 함수화도 많이 진행하는 것 같더군요.

```
myLoop <- function(somemat) {  
  myMean <- c()  
  for(i in 1:ncol(somemat)){  
    myMean<-c(myMean,mean(myMat[,i]))  
  }  
  return(myMean)  
}  
  
myLoop(myMat)
```

```
## [1] 10.0775 10.1850 10.5425 9.5625
```

근데 이제 열방향 mean 함수를 만드는게 끝났네요. 행방향을 진행하려면 똑같은걸 더 만들어야 합니다. 한 함수에 합쳐서 옵션으로 줘도 좋을 것 같군요. 한번 만들어 보세요.

하지만 apply는 이 걸 한줄에 할 수 있게 해줍니다.

```
apply(myMat, 2, mean)
```

```
## [1] 10.0775 10.1850 10.5425 9.5625
```

위에 결과와 비교해 보세요. 위에서 사용한 identical 함수로 두 결과를 비교해 보겠습니다.

```
identical(myLoop(myMat),apply(myMat, 2, mean))
```

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

?apply를 통해 중간의 숫자가 어떤 의미를 가지는지 확인해 보세요. 1은 같은 행끼리의 계산을, 2는 같은 열끼리의 계산을 의미합니다. apply는 3가지 옵션을 가지는데, 첫 번째는 데이터, 두 번째는 계산의 방향, 세 번째는 계산에 사용할 함수입니다. 함수부분은 다양한 함수를 사용할 수 있습니다.

```
apply(myMat,2,class)
```

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

```
apply(myMat,2,sum)
```

```
## [1] 40.31 40.74 42.17 38.25
```

```
apply(myMat,2,quantile)
```

```
##      [,1]   [,2]   [,3]   [,4]
## 0% 9.1600 9.1800 9.6900 7.7900
## 25% 9.3175 10.0425 10.2150 8.9825
## 50% 9.7750 10.4100 10.4850 9.6700
## 75% 10.5350 10.5525 10.8125 10.2500
## 100% 11.6000 10.7400 11.5100 11.1200
```

자주 사용하는 평균이나 합 같은 경우는 함수로도 구현되어 있습니다. `rowMeans`, `colMeans`, `rowSums`, `colSums`가 그것입니다. 각각 `apply`로 어떻게 하면 되는지 생각해 보세요.

`apply`에 적용하는 함수 안에 데이터만 들어가는 함수 이외에 다른 옵션을 지정해야 할 수 있습니다. `apply`는 `,`를 이용해서 다음 옵션으로 사용하는 함수 안의 옵션을 작성할 수 있습니다.

```
myMat[1,4]<-NA
apply(myMat,2,sum)
```

```
## [1] 40.31 40.74 42.17 NA
```

```
apply(myMat,2,sum, na.rm = TRUE)
```

```
## [1] 40.31 40.74 42.17 28.87
```

`apply`에서 계산에 사용할 함수는 사용자가 만들어서 진행할 수도 있고, 임시로 만들 수도 있습니다.

```
naSum <- function(x){
  return(sum(x,na.rm = TRUE))
}

apply(myMat,2,naSum)
```

```
## [1] 40.31 40.74 42.17 28.87
```

```
apply(myMat,2,function(x) sum(x,na.rm = TRUE))
```

```
## [1] 40.31 40.74 42.17 28.87
```

만들어야 할 함수가 복잡하지 않으면 저는 임시로 작성하는 방법을 사용하는 편입니다.

## apply-family

apply는 lapply, tapply, sapply, mapply 등의 apply-family를 가지고 있습니다.

우선 lapply부터 살펴보겠습니다. 앞에 l이 붙으면서 list 자료형에 대해 apply의 역할을 수행하는 함수라는 의미가 붙었습니다. 결과도 list로 나옵니다.

```
(listData <- list(a = 1, b = 1:3, c = 10:100) )
```

```
## $a
## [1] 1
##
## $b
## [1] 1 2 3
##
## $c
##  [1] 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26
## [18] 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43
## [35] 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60
## [52] 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77
## [69] 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94
## [86] 95 96 97 98 99 100
```

```
lapply(listData, length)
```

```
## $a
## [1] 1
##
## $b
## [1] 3
##
## $c
## [1] 91
```

```
lapply(listData, sum)
```

```
## $a  
## [1] 1  
##  
## $b  
## [1] 6  
##  
## $c  
## [1] 5005
```

list 자료형은 사용하시면서 느끼시겠지만 다른 곳에 사용하기 불편한 점이 있습니다. 그래서 다시 list를 푸는 방법으로 unlist를 사용하는데요. ?unlist를 입력해서 내용을 확인해보세요.

```
(listData <- list(a = 1, b = 1:3, c = 10:100) )
```

```
## $a  
## [1] 1  
##  
## $b  
## [1] 1 2 3  
##  
## $c  
##  [1] 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26  
## [18] 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43  
## [35] 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60  
## [52] 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77  
## [69] 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94  
## [86] 95 96 97 98 99 100
```

```
unlist(lapply(listData, length))
```

```
##  a  b  c  
## 1 3 91
```

```
unlist(lapply(listData, sum))
```

```
##    a      b      c  
##    1      6 5005
```

그런데 입력을 list로 받는 것은 어쩔수 없다고 쳐도, 결과물은 위처럼 vector로 받는 것이 편한 경우가 많습니다. unlist(lapply(데이터, 함수))는 sapply와 같은 동작을 합니다.

```
(listData <- list(a = 1, b = 1:3, c = 10:100) )
```

```
## $a  
## [1] 1  
##  
## $b  
## [1] 1 2 3  
##  
## $c  
## [1] 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26  
## [18] 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43  
## [35] 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60  
## [52] 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77  
## [69] 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94  
## [86] 95 96 97 98 99 100
```

```
sapply(listData, length)
```

```
## a b c  
## 1 3 91
```

```
sapply(listData, sum)
```

```
## a b c  
## 1 6 5005
```

이외에도 list안에 list까지 계산하는 rapply, 지정한 이름으로 실행할 수 있는 tapply 등이 있습니다만, 거의 apply나 sapply만 사용한 것 같습니다.

더 궁금하신 사항은 [여기](#)를 참고해 주세요.

# tidy data 개념과 dplyr+tidy로 데이터 다루기

데이터 분석을 어렵게 하는 여러 이유들이 있습니다. 이게 개발 커뮤니티에서 말하는 [기술 부채](#)와 같은 개념이지 않나 생각이 들어 데이터 부채라는 표현을 사용해 보았습니다. 여러 데이터 관련 구루들이 강조하는 바, 데이터 분석의 대부분의 시간(약 80%)은 데이터 수집과 전처리, 정제에 사용됩니다. 계륵 같은 일이죠. [garbage in, garbage out; GIGO](#) 이니까요. 데이터가 많지 않았던 시대에는 처리에 시간을 쓰는 것이 너무 당연한 일이었습니다. 하지만 데이터 생성을 설계할 수 있는 입장(서비스 제공자, 마케터 등)에서는 저 시간은 명확하게 비용, 즉 데이터 부채가 되는 것입니다.

이 데이터 부채가 쌓이는 것을 처음부터 막을 수 있도록 데이터가 저장되는 방식에 대해 제안된 개념이 있는데 그것이 [tidy data](#)입니다. tidy data란 개념은 [Hadley Wickham](#)이 제안했습니다.

## 단정한 데이터

이것에 대해 본인이 직접 [장문의 설명](#)을 한 것도 있고 R 한글 형태소 분석 패키지인 konlp를 만드신 고감자님의 [한글 설명](#), MS의 데이터 과학자이시자 헬로우 데이터과학의 저자이신 김진영님의 [도서 블로그](#)에도 너무 잘 설명되어 있습니다. 추가로 더 내용이 필요하시면 참고하시기 바랍니다. 먼저 tidy data의 개념이 필요한 이유는 컴퓨터에게 분석을 시켜야(!) 하기 때문입니다. 그래서 tidy data는 사람이 눈으로 이해하기에는 적절하지 않을 수 있습니다. 이 곳이 진입장벽이 되기도 하는데, 현재 사용하고 있는 엑셀을 바로 R에 넣고 사용하고 싶은데, 잘 안 되는 경우가 많습니다. 그것은 엑셀 파일내 데이터를 사람이 “보기 좋게” 위치했기 때문입니다. 그럼 이제 tidy data의 세 가지 조건을 원문(1)과 고감자님 번역(2), 김진영님 번역(3)순으로 살펴보겠습니다.

- 1.1 Each variable forms a column.  
1.2 각 변수는 개별의 열(column)으로 존재한다.  
1.3 각 열에는 개별 속성이 들어간다.

- 2.1 Each observation forms a row.  
2.2 각 관측치는 행(row)를 구성한다.  
2.3 각 행에는 개별 관찰 항목이 들어간다.

3.1 Each type of observational unit forms a table.

3.2 각 테이블은 단 하나의 관측기준에 의해서 조직된 데이터를 저장한다.

3.3 각 테이블에는 단일 유형의 데이터가 들어간다.

country	year	cases	population
Afghanistan	1999	745	1857071
Afghanistan	2000	2666	20595360
Brazil	1999	37737	172006362
Brazil	2000	80488	174504898
China	1999	212258	1272915272
China	2000	21266	1280428583

variables

country	year	cases	population
Afghanistan	1999	745	1857071
Afghanistan	2000	2666	20595360
Brazil	1999	37737	172006362
Brazil	2000	80488	174504898
China	1999	212258	1272915272
China	2000	21266	1280428583

observations

country	year
Afghanistan	1999
Afghanistan	2000
Brazil	1999
Brazil	2000
China	1999
China	2000

tidydata

1번을 보기 전에 2번을 먼저 보겠습니다.(2>1>3 순으로 쉬워요.) 2번은 단순합니다. 하나의 데이터가 한 줄(행)을 구성해야 한다는 것입니다. 설문지를 예로 들면 한명의 설문 결과가 한 줄로써 저장되는 것이죠. Each observation(개별 관찰)은 하나의 관찰 결과(=설문지 하나)를 뜻합니다. sql이나 data.frame에서 보셨듯 row는 조건으로 데이터를 filter할 수 있는 공간입니다. 그렇기 때문에 각 row는 개별 데이터를 의미합니다.

1번은 2번과 같은 하나가 들어가는 개념이긴 합니다만 하나의 variable, 변수, 개별 속성이라는 점이 조금 어렵습니다. 설문지 예시는 쉽습니다. 하나의 문항이라고 이해하면 되거든요. 그런데 variable이라는게 뭔지를 아는 것이 저는 조금 어려웠습니다. 찾아보니 영어상은 변수, 변할수 있는 수(여기서는 수보다는 값이라고 이해하시면 좋습니다.)인데 그 변수를 대표하는 이름이 컬럼명이라고 생각하면 좋더군요.

그런데 컬럼이 변수에 속하는 값으로 구성되는 경우가 있습니다. 예를들어 날짜가 컬럼명에 들어간 경우죠. 이렇게 생긴 데이터를 **wide form**이라고 합니다. 날짜는 변수에 들어갈 값이기 때문에 컬럼명을 날짜(date, datetime, when 등)로 정하고 컬럼에 속하는 cell에 날짜가 들어가는 형태로 구성하는 것이 tidy data의 조건을 충족하는 셈이 됩니다.

3번은 단일 테이블이 어떻게 구성되어야 하는지를 알려주는 조건입니다. 김진영님의 번역이 좀 이해하기 쉬운 것 같습니다. 테이블 하나에 하나의 데이터가 들어가야 된다는 뜻인데요, 아래 dplyr과 tidyr을 배우면서 예시들도 같이 보겠습니다.

## dplyr + tidyr

dplyr은 plyr 패키지의 data.frame 전용이라는 의미를 가지고 있습니다. plyr은 데이터의 **분해 - 적용 - 재조립 전략**을 실행할 수 있는 **패키지**입니다. 이 전략을 data.frame 형태에서 실행하기 위해서 여러 명령어들을 제공합니다. 잘 정돈된 데이터 프레임은 **분해 - 적용 - 재조립 전략**을 실행하기 쉬우며 데이터를 잘 정돈하기 위해 tidyr 패키지를 함께 사용할 수 있습니다. 최근 ggplot2, dplyr, tidyr 등 tidy data의 개념과 같은 맥락에 있는 패키지들이 하나로 모여 tidyverse 패키지가 되었습니다.

```
library(tidyverse)
```

```
## Loading tidyverse: tibble  
## Loading tidyverse: purrr
```

```
## Conflicts with tidy packages -----
```

```
## as.difftime(): lubridate, base  
## date(): lubridate, base  
## filter(): dplyr, stats  
## intersect(): lubridate, base
```

```
## lag():      dplyr, stats
## setdiff():   lubridate, base
## union():    lubridate, base
```

## pipe 연산자 %>%

%>%는 함수의 사용 방향을 바꿔서 읽고 이해하기 쉽게 만든 연산자입니다.

```
g(f(y)) == y %>% f() %>% g()
```

이렇게 사용하고 tidyverse 패키지 전반적으로 사용하는 방식입니다.

.으로 앞의 변수의 위치를 지정할 수도 있고, 괄호 안에 작성할 것이 없을 때는 괄호를 생략할 수도 있습니다.

```
g(f(x,y,z)) == y %>% f(x, . , z) %>% g
```

## dplyr 명령어 소개

dplyr에는 행에 조건을 줘서 부분을 불러오는 filter(), 필요한 컬럼만 선택하는 select(), 새로운 컬럼을 계산하는 mutate(), 조건에 따라 재정렬 할 수 있는 arrange(), group\_by()와 함께 써서 요약값을 계산할 수 있는 summarise()가 있습니다. group\_by()는 mutate(), filter()와도 사용할 수 있습니다.

```
if(!require(nycflights13)) install.packages("nycflights13")
```

```
## Loading required package: nycflights13
```

```
library(nycflights13)
flights
```

```
## # A tibble: 336,776 x 19
##   year month   day dep_time sched_dep_time dep_delay arr_time
##   <int> <int> <int>    <int>          <int>     <dbl>    <int>
## 1  2013     1     1      517            515        2     830
## 2  2013     1     1      533            529        4     850
## 3  2013     1     1      542            540        2     923
## 4  2013     1     1      544            545       -1    1004
```

```
## 5 2013 1 1 554 600 -6 812
## 6 2013 1 1 554 558 -4 740
## 7 2013 1 1 555 600 -5 913
## 8 2013 1 1 557 600 -3 709
## 9 2013 1 1 557 600 -3 838
## 10 2013 1 1 558 600 -2 753
## # ... with 336,766 more rows, and 12 more variables: sched_arr_time <int>,
## # arr_delay <dbl>, carrier <chr>, flight <int>, tailnum <chr>,
## # origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>, hour <dbl>,
## # minute <dbl>, time_hour <dttm>
```

첫번째 filter()를 사용해 보겠습니다.

```
filter(flights, month == 1, day == 1)
```

```
## Warning: package 'bindrcpp' was built under R version 3.4.1
```

```
## # A tibble: 842 x 19
##   year month   day dep_time sched_dep_time dep_delay arr_time
##   <int> <int> <int>    <int>        <int>     <dbl>    <int>
## 1 2013    1     1      517        515       2     830
## 2 2013    1     1      533        529       4     850
## 3 2013    1     1      542        540       2     923
## 4 2013    1     1      544        545      -1    1004
## 5 2013    1     1      554        600      -6     812
## 6 2013    1     1      554        558      -4     740
## 7 2013    1     1      555        600      -5     913
## 8 2013    1     1      557        600      -3     709
## 9 2013    1     1      557        600      -3     838
## 10 2013   1     1      558        600      -2     753
## # ... with 832 more rows, and 12 more variables: sched_arr_time <int>,
## # arr_delay <dbl>, carrier <chr>, flight <int>, tailnum <chr>,
## # origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>, hour <dbl>,
## # minute <dbl>, time_hour <dttm>
```

```
jan1 <- filter(flights, month == 1, day == 1)
(dec25 <- filter(flights, month == 12, day == 25))
```

```
## # A tibble: 719 x 19
##   year month   day dep_time sched_dep_time dep_delay arr_time
##   <int> <int> <int>    <int>        <int>     <dbl>    <int>
## 1 2013    12    25      456        500      -4     649
## 2 2013    12    25      524        515       9     805
## 3 2013    12    25      542        540       2     832
```

```
## 4 2013 12 25 546 550 -4 1022
## 5 2013 12 25 556 600 -4 730
## 6 2013 12 25 557 600 -3 743
## 7 2013 12 25 557 600 -3 818
## 8 2013 12 25 559 600 -1 855
## 9 2013 12 25 559 600 -1 849
## 10 2013 12 25 600 600 0 850
## # ... with 709 more rows, and 12 more variables: sched_arr_time <int>,
## # arr_delay <dbl>, carrier <chr>, flight <int>, tailnum <chr>,
## # origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>, hour <dbl>,
## # minute <dbl>, time_hour <dttm>
```

```
filter(flights, month == 11 | month == 12)
```

```
## # A tibble: 55,403 x 19
##   year month   day dep_time sched_dep_time dep_delay arr_time
##   <int> <int> <int>    <int>        <int>     <dbl>    <int>
## 1 2013    11     1      5        2359       6.00    352
## 2 2013    11     1     35        2250      105.0    123
## 3 2013    11     1    455        500      -5.00   641
## 4 2013    11     1    539        545      -6.00   856
## 5 2013    11     1    542        545      -3.00   831
## 6 2013    11     1    549        600     -11.00   912
## 7 2013    11     1    550        600     -10.00   705
## 8 2013    11     1    554        600      -6.00   659
## 9 2013    11     1    554        600      -6.00   826
## 10 2013   11     1    554        600      -6.00   749
## # ... with 55,393 more rows, and 12 more variables: sched_arr_time <int>,
## # arr_delay <dbl>, carrier <chr>, flight <int>, tailnum <chr>,
## # origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>, hour <dbl>,
## # minute <dbl>, time_hour <dttm>
```

```
nov_dec <- filter(flights, month %in% c(11, 12))
nov_dec
```

```
## # A tibble: 55,403 x 19
##   year month   day dep_time sched_dep_time dep_delay arr_time
##   <int> <int> <int>    <int>        <int>     <dbl>    <int>
## 1 2013    11     1      5        2359       6.00    352
## 2 2013    11     1     35        2250      105.0    123
## 3 2013    11     1    455        500      -5.00   641
## 4 2013    11     1    539        545      -6.00   856
## 5 2013    11     1    542        545      -3.00   831
## 6 2013    11     1    549        600     -11.00   912
## 7 2013    11     1    550        600     -10.00   705
## 8 2013    11     1    554        600      -6.00   659
```

```
## 9 2013 11 1 554 600 -6 826
## 10 2013 11 1 554 600 -6 749
## # ... with 55,393 more rows, and 12 more variables: sched_arr_time <int>,
## # arr_delay <dbl>, carrier <chr>, flight <int>, tailnum <chr>,
## # origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>, hour <dbl>,
## # minute <dbl>, time_hour <dttm>
```

```
filter(flights, !(arr_delay > 120 | dep_delay > 120))
```

```
## # A tibble: 316,050 x 19
##   year month   day dep_time sched_dep_time dep_delay arr_time
##   <int> <int> <int>     <int>        <int>    <dbl>    <int>
## 1 2013    1     1      517        515       2     830
## 2 2013    1     1      533        529       4     850
## 3 2013    1     1      542        540       2     923
## 4 2013    1     1      544        545      -1    1004
## 5 2013    1     1      554        600      -6     812
## 6 2013    1     1      554        558      -4     740
## 7 2013    1     1      555        600      -5     913
## 8 2013    1     1      557        600      -3     709
## 9 2013    1     1      557        600      -3     838
## 10 2013   1     1      558        600      -2     753
## # ... with 316,040 more rows, and 12 more variables: sched_arr_time <int>,
## # arr_delay <dbl>, carrier <chr>, flight <int>, tailnum <chr>,
## # origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>, hour <dbl>,
## # minute <dbl>, time_hour <dttm>
```

```
filter(flights, arr_delay <= 120, dep_delay <= 120)
```

```
## # A tibble: 316,050 x 19
##   year month   day dep_time sched_dep_time dep_delay arr_time
##   <int> <int> <int>     <int>        <int>    <dbl>    <int>
## 1 2013    1     1      517        515       2     830
## 2 2013    1     1      533        529       4     850
## 3 2013    1     1      542        540       2     923
## 4 2013    1     1      544        545      -1    1004
## 5 2013    1     1      554        600      -6     812
## 6 2013    1     1      554        558      -4     740
## 7 2013    1     1      555        600      -5     913
## 8 2013    1     1      557        600      -3     709
## 9 2013    1     1      557        600      -3     838
## 10 2013   1     1      558        600      -2     753
## # ... with 316,040 more rows, and 12 more variables: sched_arr_time <int>,
## # arr_delay <dbl>, carrier <chr>, flight <int>, tailnum <chr>,
## # origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>, hour <dbl>,
## # minute <dbl>, time_hour <dttm>
```

`arrange()`는 조건을 바탕으로 정렬을 다시 해줍니다.

```
arrange(flights, year, month, day)
```

```
## # A tibble: 336,776 x 19
##   year month   day dep_time sched_dep_time dep_delay arr_time
##   <int> <int> <int>     <int>        <dbl>    <int>
## 1 2013     1     1      517         515      2       830
## 2 2013     1     1      533         529      4       850
## 3 2013     1     1      542         540      2       923
## 4 2013     1     1      544         545     -1      1004
## 5 2013     1     1      554         600     -6       812
## 6 2013     1     1      554         558     -4       740
## 7 2013     1     1      555         600     -5       913
## 8 2013     1     1      557         600     -3       709
## 9 2013     1     1      557         600     -3       838
## 10 2013    1     1      558         600     -2       753
## # ... with 336,766 more rows, and 12 more variables: sched_arr_time <int>,
## #   arr_delay <dbl>, carrier <chr>, flight <int>, tailnum <chr>,
## #   origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>, hour <dbl>,
## #   minute <dbl>, time_hour <dttm>
```

```
arrange(flights, desc(arr_delay))
```

```
## # A tibble: 336,776 x 19
##   year month   day dep_time sched_dep_time dep_delay arr_time
##   <int> <int> <int>     <int>        <dbl>    <int>
## 1 2013     1     9      641         900     1301     1242
## 2 2013     6    15     1432        1935     1137     1607
## 3 2013     1    10     1121        1635     1126     1239
## 4 2013     9    20     1139        1845     1014     1457
## 5 2013     7    22      845        1600     1005     1044
## 6 2013     4    10     1100        1900      960     1342
## 7 2013     3    17     2321        810      911      135
## 8 2013     7    22     2257        759      898      121
## 9 2013    12     5      756        1700      896     1058
## 10 2013    5     3     1133        2055      878     1250
## # ... with 336,766 more rows, and 12 more variables: sched_arr_time <int>,
## #   arr_delay <dbl>, carrier <chr>, flight <int>, tailnum <chr>,
## #   origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>, hour <dbl>,
## #   minute <dbl>, time_hour <dttm>
```

```
df <- tibble(x = c(5, 2, NA))
arrange(df, x)
```

```
## # A tibble: 3 x 1
##   x
##   <dbl>
## 1 2
## 2 5
## 3 NA
```

```
arrange(df, desc(x))
```

```
## # A tibble: 3 x 1
##   x
##   <dbl>
## 1 5
## 2 2
## 3 NA
```

select()는 컬럼을 선택하는 함수라고 했습니다.

```
select(flights, year, month, day)
```

```
## # A tibble: 336,776 x 3
##   year month   day
##   <int> <int> <int>
## 1 2013     1     1
## 2 2013     1     1
## 3 2013     1     1
## 4 2013     1     1
## 5 2013     1     1
## 6 2013     1     1
## 7 2013     1     1
## 8 2013     1     1
## 9 2013     1     1
## 10 2013    1     1
## # ... with 336,766 more rows
```

```
select(flights, year:day)
```

```
## # A tibble: 336,776 x 3
##   year month   day
##   <int> <int> <int>
```

```
## 1 2013 1 1  
## 2 2013 1 1  
## 3 2013 1 1  
## 4 2013 1 1  
## 5 2013 1 1  
## 6 2013 1 1  
## 7 2013 1 1  
## 8 2013 1 1  
## 9 2013 1 1  
## 10 2013 1 1  
## # ... with 336,766 more rows
```

```
select(flights, -(year:day))
```

```
## # A tibble: 336,776 x 16  
##   dep_time sched_dep_time dep_delay arr_time sched_arr_time arr_delay  
##   <int>       <int>    <dbl>    <int>       <int>    <dbl>  
## 1     517         515      2     830        819      11  
## 2     533         529      4     850        830      20  
## 3     542         540      2     923        850      33  
## 4     544         545     -1    1004       1022     -18  
## 5     554         600     -6     812        837     -25  
## 6     554         558     -4     740        728      12  
## 7     555         600     -5     913        854      19  
## 8     557         600     -3     709        723     -14  
## 9     557         600     -3     838        846      -8  
## 10    558         600     -2     753        745      8  
## # ... with 336,766 more rows, and 10 more variables: carrier <chr>,  
## #   flight <int>, tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>,  
## #   distance <dbl>, hour <dbl>, minute <dbl>, time_hour <dttm>
```

select와 함께 사용하는 함수로 starts\_with("abc"), ends\_with("xyz"), contains("ijk"), matches("(.)WW1"), num\_range("x", 1:3) 등을 들 수 있습니다. ?select를 실행해서 자세한 사항을 확인해보세요.

rename()은 컬럼의 이름을 바꾸는 함수고, everything()은 선택한 것 이외에 전부를 뜻합니다.

```
rename(flights, tail_num = tailnum)
```

```
## # A tibble: 336,776 x 19  
##   year month day dep_time sched_dep_time dep_delay arr_time  
##   <int> <int> <int>    <int>       <int>    <dbl>    <int>  
## 1 2013    1     1      517         515      2     830
```

```

## 2 2013 1 1 533 529 4 850
## 3 2013 1 1 542 540 2 923
## 4 2013 1 1 544 545 -1 1004
## 5 2013 1 1 554 600 -6 812
## 6 2013 1 1 554 558 -4 740
## 7 2013 1 1 555 600 -5 913
## 8 2013 1 1 557 600 -3 709
## 9 2013 1 1 557 600 -3 838
## 10 2013 1 1 558 600 -2 753
## # ... with 336,766 more rows, and 12 more variables: sched_arr_time <int>,
## # arr_delay <dbl>, carrier <chr>, flight <int>, tail_num <chr>,
## # origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>, hour <dbl>,
## # minute <dbl>, time_hour <dttm>

```

```
select(flights, time_hour, air_time, everything())
```

```

## # A tibble: 336,776 x 19
##   time_hour    air_time year month day dep_time sched_dep_time
##   <dttm>      <dbl>  <int> <int> <int>    <int>          <int>
## 1 2013-01-01 05:00:00     227 2013     1     1      517        515
## 2 2013-01-01 05:00:00     227 2013     1     1      533        529
## 3 2013-01-01 05:00:00     160 2013     1     1      542        540
## 4 2013-01-01 05:00:00     183 2013     1     1      544        545
## 5 2013-01-01 06:00:00     116 2013     1     1      554        600
## 6 2013-01-01 05:00:00     150 2013     1     1      554        558
## 7 2013-01-01 06:00:00     158 2013     1     1      555        600
## 8 2013-01-01 06:00:00      53 2013     1     1      557        600
## 9 2013-01-01 06:00:00     140 2013     1     1      557        600
## 10 2013-01-01 06:00:00    138 2013     1     1      558        600
## # ... with 336,766 more rows, and 12 more variables: dep_delay <dbl>,
## # arr_time <int>, sched_arr_time <int>, arr_delay <dbl>, carrier <chr>,
## # flight <int>, tailnum <chr>, origin <chr>, dest <chr>, distance <dbl>,
## # hour <dbl>, minute <dbl>

```

`mutate()`는 새로운 변수 계산을 위해서 필요합니다.

```

flights_sml <- select(flights,
  year:day,
  ends_with("delay"),
  distance,
  air_time
)
mutate(flights_sml,
  gain = arr_delay - dep_delay,

```

```
    speed = distance / air_time * 60  
  )
```

```
## # A tibble: 336,776 x 9  
##   year month   day dep_delay arr_delay distance air_time gain   speed  
##   <int> <int> <int>   <dbl>   <dbl>   <dbl>   <dbl> <dbl>   <dbl>  
## 1 2013     1     1       2      11     1400     227     9 370.0441  
## 2 2013     1     1       4      20     1416     227    16 374.2731  
## 3 2013     1     1       2      33     1089     160    31 408.3750  
## 4 2013     1     1      -1     -18     1576     183   -17 516.7213  
## 5 2013     1     1      -6     -25      762     116   -19 394.1379  
## 6 2013     1     1      -4      12      719     150    16 287.6000  
## 7 2013     1     1      -5      19     1065     158    24 404.4304  
## 8 2013     1     1      -3     -14     229      53   -11 259.2453  
## 9 2013     1     1      -3      -8      944     140   -5 404.5714  
## 10 2013    1     1     -2       8      733     138    10 318.6957  
## # ... with 336,766 more rows
```

```
mutate(flights_sm1,  
  gain = arr_delay - dep_delay,  
  hours = air_time / 60,  
  gain_per_hour = gain / hours  
)
```

```
## # A tibble: 336,776 x 10  
##   year month   day dep_delay arr_delay distance air_time gain   hours  
##   <int> <int> <int>   <dbl>   <dbl>   <dbl>   <dbl> <dbl>   <dbl>  
## 1 2013     1     1       2      11     1400     227     9 3.7833333  
## 2 2013     1     1       4      20     1416     227    16 3.7833333  
## 3 2013     1     1       2      33     1089     160    31 2.6666667  
## 4 2013     1     1      -1     -18     1576     183   -17 3.0500000  
## 5 2013     1     1      -6     -25      762     116   -19 1.9333333  
## 6 2013     1     1      -4      12      719     150    16 2.5000000  
## 7 2013     1     1      -5      19     1065     158    24 2.6333333  
## 8 2013     1     1      -3     -14     229      53   -11 0.8833333  
## 9 2013     1     1      -3      -8      944     140   -5 2.3333333  
## 10 2013    1     1     -2       8      733     138    10 2.3000000  
## # ... with 336,766 more rows, and 1 more variables: gain_per_hour <dbl>
```

```
transmute(flights,  
  gain = arr_delay - dep_delay,  
  hours = air_time / 60,  
  gain_per_hour = gain / hours  
)
```

```
## # A tibble: 336,776 x 3
##   gain    hours gain_per_hour
##   <dbl>    <dbl>        <dbl>
## 1 9 3.7833333 2.378855
## 2 16 3.7833333 4.229075
## 3 31 2.6666667 11.625000
## 4 -17 3.0500000 -5.573770
## 5 -19 1.9333333 -9.827586
## 6 16 2.5000000 6.400000
## 7 24 2.6333333 9.113924
## 8 -11 0.8833333 -12.452830
## 9 -5 2.3333333 -2.142857
## 10 10 2.3000000 4.347826
## # ... with 336,766 more rows
```

특별히 `mutate()`와 함께 사용하는 함수중에 `lag()`와 `lead()`를 소개할까 합니다.

```
(x <- 1:10)
```

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

```
lag(x)
```

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

```
lead(x)
```

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

`mutate()`가 각 행에 대한 계산 결과를 하나의 컬럼으로 만들어 주는 것이라면 `summarise()`는 일정 조건(대부분 `group_by()`를 이용한 그룹화)에 해당하는 계산을 수행해줍니다.

```
summarise(flights, delay = mean(dep_delay, na.rm = TRUE))
```

```
## # A tibble: 1 x 1
##       delay
```

```
## #<dbl>
## 1 12.63907
```

```
by_day <- group_by(flights, year, month, day)
summarise(by_day, delay = mean(dep_delay, na.rm = TRUE))
```

```
## # A tibble: 365 x 4
## # Groups: year, month [?]
##   year month day   delay
##   <int> <int> <int>   <dbl>
## 1 2013    1    1 11.548926
## 2 2013    1    2 13.858824
## 3 2013    1    3 10.987832
## 4 2013    1    4  8.951595
## 5 2013    1    5  5.732218
## 6 2013    1    6  7.148014
## 7 2013    1    7  5.417204
## 8 2013    1    8  2.553073
## 9 2013    1    9  2.276477
## 10 2013   1    10  2.844995
## # ... with 355 more rows
```

```
daily <- group_by(flights, year, month, day)
(per_day <- summarise(daily, flights = n()))
```

```
## # A tibble: 365 x 4
## # Groups: year, month [?]
##   year month day flights
##   <int> <int> <int>   <int>
## 1 2013    1    1     842
## 2 2013    1    2     943
## 3 2013    1    3     914
## 4 2013    1    4     915
## 5 2013    1    5     720
## 6 2013    1    6     832
## 7 2013    1    7     933
## 8 2013    1    8     899
## 9 2013    1    9     902
## 10 2013   1    10    932
## # ... with 355 more rows
```

```
(per_month <- summarise(per_day, flights = sum(flights)))
```

```
## # A tibble: 12 x 3
## # Groups: year [?]
##   year month flights
##   <int> <int>   <int>
## 1 2013    1    27004
## 2 2013    2    24951
## 3 2013    3    28834
## 4 2013    4    28330
## 5 2013    5    28796
## 6 2013    6    28243
## 7 2013    7    29425
## 8 2013    8    29327
## 9 2013    9    27574
## 10 2013   10    28889
## 11 2013   11    27268
## 12 2013   12    28135
```

```
(per_year <- summarise(per_month, flights = sum(flights)))
```

```
## # A tibble: 1 x 2
##   year flights
##   <int>   <int>
## 1 2013    336776
```

```
daily %>%
ungroup() %>%
summarise(flights = n())
```

```
## # A tibble: 1 x 1
##   flights
##   <int>
## 1 336776
```

group\_by()를 mutate(), filter()와도 사용할 수 있다고 했습니다.

```
flights_sm1 %>%
group_by(year, month, day) %>%
filter(rank(desc(arr_delay)) < 10)
```

```
## # A tibble: 3,306 x 7
## # Groups: year, month, day [365]
```

```
##      year month   day dep_delay arr_delay distance air_time
##      <int> <int> <int>    <dbl>    <dbl>    <dbl>    <dbl>
## 1 2013     1     1     853     851     184      41
## 2 2013     1     1     290     338    1134     213
## 3 2013     1     1     260     263     266      46
## 4 2013     1     1     157     174     213      60
## 5 2013     1     1     216     222     708     121
## 6 2013     1     1     255     250     589     115
## 7 2013     1     1     285     246    1085     146
## 8 2013     1     1     192     191     199      44
## 9 2013     1     1     379     456    1092     222
## 10 2013    1     2     224    207     550      94
## # ... with 3,296 more rows
```

```
popular_dests <- flights %>%
  group_by(dest) %>%
  filter(n() > 365)
popular_dests
```

```
## # A tibble: 332,577 x 19
## # Groups: dest [77]
##      year month   day dep_time sched_dep_time dep_delay arr_time
##      <int> <int> <int>    <int>    <int>    <dbl>    <int>
## 1 2013     1     1     517      515      2     830
## 2 2013     1     1     533      529      4     850
## 3 2013     1     1     542      540      2     923
## 4 2013     1     1     544      545     -1    1004
## 5 2013     1     1     554      600     -6     812
## 6 2013     1     1     554      558     -4     740
## 7 2013     1     1     555      600     -5     913
## 8 2013     1     1     557      600     -3     709
## 9 2013     1     1     557      600     -3     838
## 10 2013    1     1     558      600     -2     753
## # ... with 332,567 more rows, and 12 more variables: sched_arr_time <int>,
## # arr_delay <dbl>, carrier <chr>, flight <int>, tailnum <chr>,
## # origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>, hour <dbl>,
## # minute <dbl>, time_hour <dttm>
```

```
popular_dests %>%
  filter(arr_delay > 0) %>%
  mutate(prop_delay = arr_delay / sum(arr_delay)) %>%
  select(year:day, dest, arr_delay, prop_delay)
```

```
## # A tibble: 131,106 x 6
## # Groups: dest [77]
##      year month   day dest arr_delay prop_delay
```

```

## #<int> <int> <int> <chr> <dbl> <dbl>
## 1 2013 1 1 IAH 11 1.106740e-04
## 2 2013 1 1 IAH 20 2.012255e-04
## 3 2013 1 1 MIA 33 2.350026e-04
## 4 2013 1 1 ORD 12 4.239594e-05
## 5 2013 1 1 FLL 19 9.377853e-05
## 6 2013 1 1 ORD 8 2.826396e-05
## 7 2013 1 1 LAX 7 3.444441e-05
## 8 2013 1 1 DFW 31 2.817951e-04
## 9 2013 1 1 ATL 12 3.996017e-05
## 10 2013 1 1 DTW 16 1.157257e-04
## # ... with 131,096 more rows

```

## tidyr 명령어 소개

tidyr에는 long form을 wide form으로 바꿔주는 spread(), 반대로 wide form을 long form으로 바꿔주는 gather(), 여러 의미를 지닌 데이터를 특정 글자를 기준으로 분리해 주는 separate(), 그 반대로 합치는 unite(), 데이터를 분리하는 품을 지정해 줄 수 있는 extract()가 있습니다.

우선 내장된 데이터를 소개하겠습니다.

table1

```

## # A tibble: 6 x 4
##       country year  cases population
##       <chr>   <int> <int>      <int>
## 1 Afghanistan 1999    745 19987071
## 2 Afghanistan 2000   2666 20595360
## 3 Brazil     1999  37737 172006362
## 4 Brazil     2000  80488 174504898
## 5 China      1999 212258 1272915272
## 6 China      2000 213766 1280428583

```

table2

```

## # A tibble: 12 x 4
##       country year     type   count
##       <chr>   <int> <chr>   <int>
## 1 Afghanistan 1999   cases     745
## 2 Afghanistan 1999 population 19987071

```

```
## 3 Afghanistan 2000 cases 2666
## 4 Afghanistan 2000 population 20595360
## 5 Brazil 1999 cases 37737
## 6 Brazil 1999 population 172006362
## 7 Brazil 2000 cases 80488
## 8 Brazil 2000 population 174504898
## 9 China 1999 cases 212258
## 10 China 1999 population 1272915272
## 11 China 2000 cases 213766
## 12 China 2000 population 1280428583
```

table3

```
## # A tibble: 6 x 3
##   country year      rate
##   <chr>    <int>    <chr>
## 1 Afghanistan 1999 745/19987071
## 2 Afghanistan 2000 2666/20595360
## 3 Brazil    1999 37737/172006362
## 4 Brazil    2000 80488/174504898
## 5 China     1999 212258/1272915272
## 6 China     2000 213766/1280428583
```

table4a

```
## # A tibble: 3 x 3
##   country `1999` `2000`
##   <chr>    <int>   <int>
## 1 Afghanistan 745  2666
## 2 Brazil    37737 80488
## 3 China     212258 213766
```

table4b

```
## # A tibble: 3 x 3
##   country `1999`   `2000`
##   <chr>    <int>    <int>
## 1 Afghanistan 19987071 20595360
## 2 Brazil    172006362 174504898
## 3 China     1272915272 1280428583
```

이제 많이 사용하게 될 gather() 함수를 보겠습니다.

```
table4a
```

```
## # A tibble: 3 x 3
##       country `1999` `2000`
## *   <chr>     <int>   <int>
## 1 Afghanistan     745    2666
## 2      Brazil  37737   80488
## 3      China 212258  213766
```

```
table4a %>%
  gather(`1999`, `2000`, key = "year", value = "cases")
```

```
## # A tibble: 6 x 3
##       country year  cases
##   <chr>     <chr> <int>
## 1 Afghanistan 1999     745
## 2      Brazil  1999  37737
## 3      China  1999 212258
## 4 Afghanistan 2000    2666
## 5      Brazil  2000   80488
## 6      China  2000  213766
```

table4

country	year	cases	country	1999	2000
Afghanistan	1999	745	Afghanistan	745	2666
Afghanistan	2000	2666	Brazil	37737	80488
Brazil	1999	37737	China	212258	213766
Brazil	2000	80488			
China	1999	212258			
China	2000	213766			

gather

이번엔 반대과정인 spread()를 보겠습니다.

table2

```
## # A tibble: 12 x 4
##       country year     type   count
##       <chr>   <int>   <chr>   <int>
## 1 Afghanistan 1999 cases    745
## 2 Afghanistan 1999 population 19987071
## 3 Afghanistan 2000 cases    2666
## 4 Afghanistan 2000 population 20595360
## 5 Brazil      1999 cases    37737
## 6 Brazil      1999 population 172006362
## 7 Brazil      2000 cases    80488
```

```
## 8      Brazil 2000 population 174504898
## 9      China 1999    cases   212258
## 10     China 1999 population 1272915272
## 11     China 2000    cases   213766
## 12     China 2000 population 1280428583
```

```
spread(table2, key = type, value = count)
```

```
## # A tibble: 6 x 4
##       country year  cases population
## * <chr>     <int> <int>      <int>
## 1 Afghanistan 1999    745 19987071
## 2 Afghanistan 2000   2666 20595360
## 3      Brazil 1999 37737 172006362
## 4      Brazil 2000 80488 174504898
## 5      China 1999 212258 1272915272
## 6      China 2000 213766 1280428583
```

country	year	key	value	country	year	cases	popu
Afghanistan	1999	cases	745	Afghanistan	1999	745	199
Afghanistan	1999	population	19987071	Afghanistan	2000	2666	205
Afghanistan	2000	cases	2666	Brazil	1999	37737	1720
Afghanistan	2000	population	20595360	Brazil	2000	80488	1745
Brazil	1999	cases	37737	China	1999	212258	12729
Brazil	1999	population	172006362	China	2000	213766	12804
Brazil	2000	cases	80488				
Brazil	2000	population	174504898				
China	1999	cases	212258				
China	1999	population	1272915272				
China	2000	cases	213766				
China	2000	population	1280428583				

table2

spread

한 셀에 여러 값이 있어서 나눠야 할 때는 `seperate()`를 사용합니다. `sep`옵션을 주지 않아도, 간단한 것은 알아서 나눠줍니다.

```
table3
```

```
## # A tibble: 6 x 3
##   country year      rate
## * <chr>    <int>    <chr>
## 1 Afghanistan 1999 745/19987071
## 2 Afghanistan 2000 2666/20595360
## 3 Brazil     1999 37737/172006362
## 4 Brazil     2000 80488/174504898
## 5 China      1999 212258/1272915272
## 6 China      2000 213766/1280428583
```

```
table3 %>%
  separate(rate, into = c("cases", "population"))
```

```
## # A tibble: 6 x 4
##   country year cases population
## * <chr>    <int> <chr>      <chr>
## 1 Afghanistan 1999 745 19987071
## 2 Afghanistan 2000 2666 20595360
## 3 Brazil     1999 37737 172006362
## 4 Brazil     2000 80488 174504898
## 5 China      1999 212258 1272915272
## 6 China      2000 213766 1280428583
```

```
table3 %>%
  separate(rate, into = c("cases", "population"), convert = TRUE)
```

```
## # A tibble: 6 x 4
##   country year cases population
## * <chr>    <int> <int>      <int>
## 1 Afghanistan 1999 745 19987071
## 2 Afghanistan 2000 2666 20595360
## 3 Brazil     1999 37737 172006362
## 4 Brazil     2000 80488 174504898
## 5 China      1999 212258 1272915272
## 6 China      2000 213766 1280428583
```

```
table3 %>%
  separate(year, into = c("century", "year"), sep = 2)
```

```
## # A tibble: 6 x 4
##   country century year      rate
## * <chr>    <chr> <chr>    <chr>
## 1 Afghanistan 19  99  745/19987071
## 2 Afghanistan 20  00  2666/20595360
## 3 Brazil     19  99  37737/172006362
## 4 Brazil     20  00  80488/174504898
## 5 China      19  99  212258/1272915272
## 6 China      20  00  213766/1280428583
```

unite()는 합쳐주는 seperate()와는 반대의 기능을 가진 함수입니다.

```
table5 %>%
  unite(new, century, year)
```

```
## # A tibble: 6 x 3
##   country   new      rate
## * <chr>    <chr>    <chr>
## 1 Afghanistan 19_99  745/19987071
## 2 Afghanistan 20_00  2666/20595360
## 3 Brazil     19_99  37737/172006362
## 4 Brazil     20_00  80488/174504898
## 5 China      19_99  212258/1272915272
## 6 China      20_00  213766/1280428583
```

```
table5 %>%
  unite(new, century, year, sep = " ")
```

```
## # A tibble: 6 x 3
##   country   new      rate
## * <chr>    <chr>    <chr>
## 1 Afghanistan 1999  745/19987071
## 2 Afghanistan 2000  2666/20595360
## 3 Brazil     1999  37737/172006362
## 4 Brazil     2000  80488/174504898
## 5 China      1999  212258/1272915272
## 6 China      2000  213766/1280428583
```

## dplyr과 join

dplyr에는 join() 기능도 있습니다. 데이터를 먼저 소개하겠습니다.

## airlines

```
## # A tibble: 16 x 2
##   carrier          name
##   <chr>            <chr>
## 1 9E    Endeavor Air Inc.
## 2 AA   American Airlines Inc.
## 3 AS   Alaska Airlines Inc.
## 4 B6   JetBlue Airways
## 5 DL   Delta Air Lines Inc.
## 6 EV   ExpressJet Airlines Inc.
## 7 F9   Frontier Airlines Inc.
## 8 FL   AirTran Airways Corporation
## 9 HA   Hawaiian Airlines Inc.
## 10 MQ   Envoy Air
## 11 OO   SkyWest Airlines Inc.
## 12 UA   United Air Lines Inc.
## 13 US   US Airways Inc.
## 14 VX   Virgin America
## 15 WN   Southwest Airlines Co.
## 16 YV   Mesa Airlines Inc.
```

## airports

```
## # A tibble: 1,458 x 8
##   faa             name      lat     lon   alt   tz
##   <chr>           <chr>    <dbl>   <dbl> <int> <dbl>
## 1 04G   Lansdowne Airport 41.13047 -80.61958 1044  -5
## 2 06A   Moton Field Municipal Airport 32.46057 -85.68003 264   -6
## 3 06C   Schaumburg Regional 41.98934 -88.10124 801   -6
## 4 06N   Randall Airport 41.43191 -74.39156 523   -5
## 5 09J   Jekyll Island Airport 31.07447 -81.42778 11    -5
## 6 0A9   Elizabethton Municipal Airport 36.37122 -82.17342 1593  -5
## 7 0G6   Williams County Airport 41.46731 -84.50678 730   -5
## 8 0G7   Finger Lakes Regional Airport 42.88356 -76.78123 492   -5
## 9 0P2   Shoestring Aviation Airfield 39.79482 -76.64719 1000  -5
## 10 OS9   Jefferson County Intl 48.05381 -122.81064 108   -8
## # ... with 1,448 more rows, and 2 more variables: dst <chr>, tzone <chr>
```

## planes

```
## # A tibble: 3,322 x 9
##   tailnum year      type manufacturer model
##   <chr>   <int>    <chr>        <chr>    <chr>
## 1 N10156 2004 Fixed wing multi engine EMBRAER EMB-145XR
```

```
## 2 N102UW 1998 Fixed wing multi engine AIRBUS INDUSTRIE A320-214
## 3 N103US 1999 Fixed wing multi engine AIRBUS INDUSTRIE A320-214
## 4 N104UW 1999 Fixed wing multi engine AIRBUS INDUSTRIE A320-214
## 5 N10575 2002 Fixed wing multi engine EMBRAER EMB-145LR
## 6 N105UW 1999 Fixed wing multi engine AIRBUS INDUSTRIE A320-214
## 7 N107US 1999 Fixed wing multi engine AIRBUS INDUSTRIE A320-214
## 8 N108UW 1999 Fixed wing multi engine AIRBUS INDUSTRIE A320-214
## 9 N109UW 1999 Fixed wing multi engine AIRBUS INDUSTRIE A320-214
## 10 N110UW 1999 Fixed wing multi engine AIRBUS INDUSTRIE A320-214
## # ... with 3,312 more rows, and 4 more variables: engines <int>,
## #   seats <int>, speed <int>, engine <chr>
```

weather

```
## # A tibble: 26,130 x 15
##   origin year month day hour temp dewp humid wind_dir wind_speed
##   <chr> <dbl> <dbl> <int> <int> <dbl> <dbl> <dbl> <dbl> <dbl>
## 1 EWR 2013 1 1 0 37.04 21.92 53.97 230 10.35702
## 2 EWR 2013 1 1 1 37.04 21.92 53.97 230 13.80936
## 3 EWR 2013 1 1 2 37.94 21.92 52.09 230 12.65858
## 4 EWR 2013 1 1 3 37.94 23.00 54.51 230 13.80936
## 5 EWR 2013 1 1 4 37.94 24.08 57.04 240 14.96014
## 6 EWR 2013 1 1 6 39.02 26.06 59.37 270 10.35702
## 7 EWR 2013 1 1 7 39.02 26.96 61.63 250 8.05546
## 8 EWR 2013 1 1 8 39.02 28.04 64.43 240 11.50780
## 9 EWR 2013 1 1 9 39.92 28.04 62.21 250 12.65858
## 10 EWR 2013 1 1 10 39.02 28.04 64.43 260 12.65858
## # ... with 26,120 more rows, and 5 more variables: wind_gust <dbl>,
## #   precip <dbl>, pressure <dbl>, visib <dbl>, time_hour <dttm>
```

%>%와 join() 명령어로 쉽게 데이터를 합칠 수 있습니다.

```
flights2 <- flights %>%
  select(year:day, hour, origin, dest, tailnum, carrier)
flights2
```

```
## # A tibble: 336,776 x 8
##   year month day hour origin dest tailnum carrier
##   <int> <int> <int> <dbl> <chr> <chr> <chr> <chr>
## 1 2013 1 1 5 EWR IAH N14228 UA
## 2 2013 1 1 5 LGA IAH N24211 UA
## 3 2013 1 1 5 JFK MIA N619AA AA
## 4 2013 1 1 5 JFK BQN N804JB B6
## 5 2013 1 1 6 LGA ATL N668DN DL
```

```
## 6 2013 1 1 5 EWR ORD N39463 UA
## 7 2013 1 1 6 EWR FLL N516JB B6
## 8 2013 1 1 6 LGA IAD N829AS EV
## 9 2013 1 1 6 JFK MCO N593JB B6
## 10 2013 1 1 6 LGA ORD N3ALAA AA
## # ... with 336,766 more rows
```

```
flights2 %>%
  select(-origin, -dest) %>%
  left_join(airlines, by = "carrier")
```

```
## # A tibble: 336,776 x 7
##   year month day hour tailnum carrier          name
##   <int> <int> <int> <dbl> <chr>  <chr>
## 1 2013    1     1     5 N14228  UA United Air Lines Inc.
## 2 2013    1     1     5 N24211  UA United Air Lines Inc.
## 3 2013    1     1     5 N619AA  AA American Airlines Inc.
## 4 2013    1     1     5 N804JB  B6 JetBlue Airways
## 5 2013    1     1     6 N668DN  DL Delta Air Lines Inc.
## 6 2013    1     1     5 N39463  UA United Air Lines Inc.
## 7 2013    1     1     6 N516JB  B6 JetBlue Airways
## 8 2013    1     1     6 N829AS  EV ExpressJet Airlines Inc.
## 9 2013    1     1     6 N593JB  B6 JetBlue Airways
## 10 2013   1     1     6 N3ALAA  AA American Airlines Inc.
## # ... with 336,766 more rows
```

```
flights2 %>%
  select(-origin, -dest) %>%
  mutate(name = airlines$name[match(carrier, airlines$carrier)])
```

```
## # A tibble: 336,776 x 7
##   year month day hour tailnum carrier          name
##   <int> <int> <int> <dbl> <chr>  <chr>
## 1 2013    1     1     5 N14228  UA United Air Lines Inc.
## 2 2013    1     1     5 N24211  UA United Air Lines Inc.
## 3 2013    1     1     5 N619AA  AA American Airlines Inc.
## 4 2013    1     1     5 N804JB  B6 JetBlue Airways
## 5 2013    1     1     6 N668DN  DL Delta Air Lines Inc.
## 6 2013    1     1     5 N39463  UA United Air Lines Inc.
## 7 2013    1     1     6 N516JB  B6 JetBlue Airways
## 8 2013    1     1     6 N829AS  EV ExpressJet Airlines Inc.
## 9 2013    1     1     6 N593JB  B6 JetBlue Airways
## 10 2013   1     1     6 N3ALAA  AA American Airlines Inc.
## # ... with 336,766 more rows
```

key를 선정해주는 것과 아닌 것이 어떻게 다른지 봄주세요.

```
flights2 %>%  
  left_join(weather)
```

```
## Joining, by = c("year", "month", "day", "hour", "origin")
```

```
## # A tibble: 336,776 x 18  
##   year month day hour origin dest tailnum carrier temp dewp humid  
##   <dbl> <dbl> <int> <dbl> <chr> <chr> <chr> <chr> <dbl> <dbl> <dbl>  
## 1 2013     1     1     5   EWR   IAH  N14228   UA    NA    NA    NA  
## 2 2013     1     1     5   LGA   IAH  N24211   UA    NA    NA    NA  
## 3 2013     1     1     5   JFK   MIA  N619AA   AA    NA    NA    NA  
## 4 2013     1     1     5   JFK   BQN  N804JB   B6    NA    NA    NA  
## 5 2013     1     1     6   LGA   ATL  N668DN   DL    39.92 26.06 57.33  
## 6 2013     1     1     5   EWR   ORD  N39463   UA    NA    NA    NA  
## 7 2013     1     1     6   EWR   FLL  N516JB   B6    39.02 26.06 59.37  
## 8 2013     1     1     6   LGA   IAD  N829AS   EV    39.92 26.06 57.33  
## 9 2013     1     1     6   JFK   MCO  N593JB   B6    39.02 26.06 59.37  
## 10 2013    1     1     6   LGA   ORD  N3ALAA  AA    39.92 26.06 57.33  
## # ... with 336,766 more rows, and 7 more variables: wind_dir <dbl>,  
## #   wind_speed <dbl>, wind_gust <dbl>, precip <dbl>, pressure <dbl>,  
## #   visib <dbl>, time_hour <dttm>
```

```
flights2 %>%  
  left_join(planes, by = "tailnum")
```

```
## # A tibble: 336,776 x 16  
##   year.x month day hour origin dest tailnum carrier year.y  
##   <int> <int> <int> <dbl> <chr> <chr> <chr> <chr> <int>  
## 1 2013     1     1     5   EWR   IAH  N14228   UA    1999  
## 2 2013     1     1     5   LGA   IAH  N24211   UA    1998  
## 3 2013     1     1     5   JFK   MIA  N619AA   AA    1990  
## 4 2013     1     1     5   JFK   BQN  N804JB   B6    2012  
## 5 2013     1     1     6   LGA   ATL  N668DN   DL    1991  
## 6 2013     1     1     5   EWR   ORD  N39463   UA    2012  
## 7 2013     1     1     6   EWR   FLL  N516JB   B6    2000  
## 8 2013     1     1     6   LGA   IAD  N829AS   EV    1998  
## 9 2013     1     1     6   JFK   MCO  N593JB   B6    2004  
## 10 2013    1     1     6   LGA   ORD  N3ALAA  AA    NA  
## # ... with 336,766 more rows, and 7 more variables: type <chr>,  
## #   manufacturer <chr>, model <chr>, engines <int>, seats <int>,  
## #   speed <int>, engine <chr>
```

왼쪽 테이블과 오른쪽 테이블의 어떤 key를 기준으로 join()할 건지 지정할 수 있습니다.

```
flights2 %>%  
  left_join(airports, c("dest" = "faa"))
```

```
## # A tibble: 336,776 x 15  
##   year month   day hour origin dest tailnum carrier  
##   <int> <int> <int> <dbl> <chr> <chr> <chr> <chr>  
## 1 2013     1     1     5   EWR  IAH  N14228    UA  
## 2 2013     1     1     5   LGA  IAH  N24211    UA  
## 3 2013     1     1     5   JFK  MIA  N619AA    AA  
## 4 2013     1     1     5   JFK  BQN  N804JB    B6  
## 5 2013     1     1     6   LGA  ATL  N668DN    DL  
## 6 2013     1     1     5   EWR  ORD  N39463    UA  
## 7 2013     1     1     6   EWR  FLL  N516JB    B6  
## 8 2013     1     1     6   LGA  IAD  N829AS    EV  
## 9 2013     1     1     6   JFK  MCO  N593JB    B6  
## 10 2013    1     1     6   LGA  ORD  N3ALAA   AA  
## # ... with 336,766 more rows, and 7 more variables: name <chr>, lat <dbl>,  
## #   lon <dbl>, alt <int>, tz <dbl>, dst <chr>, tzone <chr>
```

```
flights2 %>%  
  left_join(airports, c("origin" = "faa"))
```

```
## # A tibble: 336,776 x 15  
##   year month   day hour origin dest tailnum carrier  
##   <int> <int> <int> <dbl> <chr> <chr> <chr> <chr>  
## 1 2013     1     1     5   EWR  IAH  N14228    UA  
## 2 2013     1     1     5   LGA  IAH  N24211    UA  
## 3 2013     1     1     5   JFK  MIA  N619AA    AA  
## 4 2013     1     1     5   JFK  BQN  N804JB    B6  
## 5 2013     1     1     6   LGA  ATL  N668DN    DL  
## 6 2013     1     1     5   EWR  ORD  N39463    UA  
## 7 2013     1     1     6   EWR  FLL  N516JB    B6  
## 8 2013     1     1     6   LGA  IAD  N829AS    EV  
## 9 2013     1     1     6   JFK  MCO  N593JB    B6  
## 10 2013    1     1     6   LGA  ORD  N3ALAA   AA  
## # ... with 336,766 more rows, and 7 more variables: name <chr>, lat <dbl>,  
## #   lon <dbl>, alt <int>, tz <dbl>, dst <chr>, tzone <chr>
```

join() 함수는 base::merge(), SQL과 비교할 수 있습니다.

```

inner_join(x, y) == merge(x, y)
left_join(x, y) == merge(x, y, all.x = TRUE)
right_join(x, y) == merge(x, y, all.y = TRUE),
full_join(x, y) == merge(x, y, all.x = TRUE, all.y = TRUE)

inner_join(x, y, by = "z") == SELECT * FROM x INNER JOIN y ON x.z = y.z
left_join(x, y, by = "z") == SELECT * FROM x LEFT OUTER JOIN y ON x.z = y.z
right_join(x, y, by = "z") == SELECT * FROM x RIGHT OUTER JOIN y ON x.z = y.z
full_join(x, y, by = "z") == SELECT * FROM x FULL OUTER JOIN y ON x.z = y.z

```

## data.table

data.table은 지금까지와는 조금 다른 문법을 가지고 있습니다. fread와 fwrite이라는 강력한 I/O 함수를 가지고 있으며 data.table은 패키지 명이면서 data.frame과 호환되는 자료형이기도 합니다. 자세한 내용은 [여기](#)를 참고해 주세요.

```
library(data.table)
```

```

## 
## Attaching package: 'data.table'

```

```

## The following object is masked from 'package:purrr':
## 
##     transpose

```

```

## The following objects are masked from 'package:dplyr':
## 
##     between, first, last

```

```

## The following objects are masked from 'package:lubridate':
## 
##     hour, isoweek, mday, minute, month, quarter, second, wday,
##     week, yday, year

```

```

url<-"https://github.com/arunsrinivasan/flights/wiki/NYCflights14/flights14.csv"
dir.create("./data", showWarnings = F)
download.file(url, destfile = "./data/flights14.csv")
system.time(flights <- read.csv("./data/flights14.csv"))

```

```
##   user  system elapsed
## 2.44    0.11   2.60
```

```
system.time(flights <- fread("./data/flights14.csv"))
```

```
##   user  system elapsed
## 0.27    0.02   0.33
```

```
flights
```

```
##      year month day dep_time dep_delay arr_time arr_delay cancelled
## 1: 2014     1    1      914        14     1238       13        0
## 2: 2014     1    1     1157       -3     1523       13        0
## 3: 2014     1    1     1902        2     2224        9        0
## 4: 2014     1    1      722       -8     1014      -26        0
## 5: 2014     1    1     1347        2     1706        1        0
##   ---
## 253312: 2014    10   31     1459        1     1747      -30        0
## 253313: 2014    10   31      854       -5     1147      -14        0
## 253314: 2014    10   31     1102       -8     1311       16        0
## 253315: 2014    10   31     1106       -4     1325       15        0
## 253316: 2014    10   31      824       -5     1045        1        0
##      carrier tailnum flight origin dest air_time distance hour min
## 1:      AA N338AA     1   JFK  LAX     359     2475    9 14
## 2:      AA N335AA     3   JFK  LAX     363     2475   11 57
## 3:      AA N327AA    21   JFK  LAX     351     2475   19  2
## 4:      AA N3EHAH    29   LGA  PBI     157     1035    7 22
## 5:      AA N319AA   117   JFK  LAX     350     2475   13 47
##   ---
## 253312:      UA N23708   1744   LGA  IAH     201     1416   14 59
## 253313:      UA N33132   1758   EWR  IAH     189     1400   8 54
## 253314:      MQ N827MQ   3591   LGA  RDU      83     431   11  2
## 253315:      MQ N511MQ   3592   LGA  DTW      75     502   11  6
## 253316:      MQ N813MQ   3599   LGA  SDF     110     659   8 24
```

```
dim(flights)
```

```
## [1] 253316    17
```

```
ans <- flights[origin == "JFK" & month == 6]
head(ans)
```

```
##   year month day dep_time dep_delay arr_time arr_delay cancelled carrier
## 1: 2014    6    1     851       -9     1205       -5       0     AA
## 2: 2014    6    1    1220      -10     1522      -13       0     AA
## 3: 2014    6    1     718       18     1014       -1       0     AA
## 4: 2014    6    1    1024      -6     1314      -16       0     AA
## 5: 2014    6    1    1841      -4     2125      -45       0     AA
## 6: 2014    6    1    1454      -6     1757      -23       0     AA
##   tailnum flight origin dest air_time distance hour min
## 1: N787AA     1    JFK   LAX     324     2475     8   51
## 2: N795AA     3    JFK   LAX     329     2475    12   20
## 3: N784AA     9    JFK   LAX     326     2475     7   18
## 4: N791AA    19    JFK   LAX     320     2475    10   24
## 5: N790AA    21    JFK   LAX     326     2475    18   41
## 6: N785AA   117    JFK   LAX     329     2475    14   54
```

```
ans <- flights[1:2]
ans
```

```
##   year month day dep_time dep_delay arr_time arr_delay cancelled carrier
## 1: 2014    1    1     914      14     1238      13       0     AA
## 2: 2014    1    1    1157      -3     1523      13       0     AA
##   tailnum flight origin dest air_time distance hour min
## 1: N338AA     1    JFK   LAX     359     2475     9   14
## 2: N335AA     3    JFK   LAX     363     2475    11   57
```

```
ans <- flights[order(origin, -dest)]
head(ans)
```

```
##   year month day dep_time dep_delay arr_time arr_delay cancelled carrier
## 1: 2014    1    5     836       6     1151      49       0     EV
## 2: 2014    1    6     833       7     1111      13       0     EV
## 3: 2014    1    7     811      -6     1035     -13       0     EV
## 4: 2014    1    8     810      -7     1036     -12       0     EV
## 5: 2014    1    9     833      16     1055       7       0     EV
## 6: 2014    1   13     923      66     1154      66       0     EV
##   tailnum flight origin dest air_time distance hour min
## 1: N12175   4419    EWR   XNA     195     1131     8   36
## 2: N24128   4419    EWR   XNA     190     1131     8   33
## 3: N12142   4419    EWR   XNA     179     1131     8   11
## 4: N11193   4419    EWR   XNA     184     1131     8   10
## 5: N14198   4419    EWR   XNA     181     1131     8   33
## 6: N12157   4419    EWR   XNA     188     1131     9   23
```

```
ans <- flights[, arr_delay]
head(ans)
```

```
## [1] 13 13 9 -26 1 0
```

```
ans <- flights[, .(arr_delay, dep_delay)]
head(ans)
```

```
##   arr_delay dep_delay
## 1:      13      14
## 2:      13     -3
## 3:       9      2
## 4:     -26     -8
## 5:       1      2
## 6:       0      4
```

```
ans <- flights[, .(delay_arr = arr_delay, delay_dep = dep_delay)]
head(ans)
```

```
##   delay_arr delay_dep
## 1:      13      14
## 2:      13     -3
## 3:       9      2
## 4:     -26     -8
## 5:       1      2
## 6:       0      4
```

```
flights[, sum((arr_delay + dep_delay) < 0)]
```

```
## [1] 141814
```

```
flights[origin == "JFK" & month == 6L,
.(m_arr = mean(arr_delay), m_dep = mean(dep_delay))]
```

```
##      m_arr      m_dep
## 1: 5.839349 9.807884
```

```
flights[origin == "JFK" & month == 6L, length(dest)]
```

```
## [1] 8422
```

```
flights[, .(N), by = .(origin)]
```

```
## origin N
## 1: JFK 81483
## 2: LGA 84433
## 3: EWR 87400
```

```
flights[carrier == "AA", .N, by = origin]
```

```
## origin N
## 1: JFK 11923
## 2: LGA 11730
## 3: EWR 2649
```

```
flights[carrier == "AA", .N, by = .(origin,dest)]
```

```
## origin dest N
## 1: JFK LAX 3387
## 2: LGA PBI 245
## 3: EWR LAX 62
## 4: JFK MIA 1876
## 5: JFK SEA 298
## 6: EWR MIA 848
## 7: JFK SFO 1312
## 8: JFK BOS 1173
## 9: JFK ORD 432
## 10: JFK IAH 7
## 11: JFK AUS 297
## 12: EWR DFW 1618
## 13: LGA ORD 4366
## 14: JFK STT 229
## 15: JFK SJU 690
## 16: LGA MIA 3334
## 17: LGA DFW 3785
## 18: JFK LAS 595
## 19: JFK MCO 597
## 20: JFK EGE 85
## 21: JFK DFW 474
## 22: JFK SAN 299
## 23: JFK DCA 172
```

```
## 24: EWR PHX 121  
##      origin dest   N
```

```
flights[carrier == "AA", .N, by = .(origin, dest)][order(origin, -dest)][1:10,]
```

```
##      origin dest   N  
## 1: EWR PHX 121  
## 2: EWR MIA 848  
## 3: EWR LAX  62  
## 4: EWR DFW 1618  
## 5: JFK STT 229  
## 6: JFK SJU 690  
## 7: JFK SFO 1312  
## 8: JFK SEA 298  
## 9: JFK SAN 299  
## 10: JFK ORD 432
```

# 주식 데이터를 tidy 개념으로 tidyquant

tidyquant는 quantmod 등 주식 분석을 주 목적으로 하는 중요 함수를 제공하는 중요한 패키지입니다. tidy data 개념을 활용한 데이터 핸들링, ggplot과 연계된 강한 차트 그리기, 야후를 기본으로 구글 및 각자 독자적인 데이터 소스로 부터 필요한 데이터를 손쉽게 가져오는 기능, 성능 분석 함수들을 제공하고 있습니다.

## 주가 지수 가져오기

tidyquant는 [야후 파이낸스](#)에서 정보를 가져옵니다. 가져오는 데이터 소스를 바꾸고 싶으면 어떤 곳에서 가져올지 결정할 수 있는데, tq\_get\_options()는 가능한 후보를 보여줍니다.

```
if (!require(tidyquant)) install.packages("tidyquant", verbose = F)

## Loading required package: tidyquant

## Loading required package: PerformanceAnalytics

## Warning: package 'PerformanceAnalytics' was built under R version 3.4.1

## Loading required package: xts

## Warning: package 'xts' was built under R version 3.4.1

## Loading required package: zoo

## Warning: package 'zoo' was built under R version 3.4.1

## 
## Attaching package: 'zoo'

## The following objects are masked from 'package:base':
## 
##     as.Date, as.Date.numeric

## 
## Attaching package: 'xts'
```

```
## The following objects are masked from 'package:data.table':
##
##     first, last

## The following objects are masked from 'package:dplyr':
##
##     first, last

## 
## Attaching package: 'PerformanceAnalytics'

## The following object is masked from 'package:graphics':
##
##     legend

## Loading required package: quantmod

## Warning: package 'quantmod' was built under R version 3.4.1

## Loading required package: TTR

## Warning: package 'TTR' was built under R version 3.4.1

## Version 0.4-0 included new data defaults. See ?getSymbols.

## 
## Attaching package: 'tidyquant'

## The following object is masked from 'package:tibble':
##
##     as_tibble

## The following object is masked from 'package:dplyr':
##
##     as_tibble

library(tidyquant)
tq_get_options()

## [1] "stock.prices"      "stock.prices.japan" "financials"
## [4] "key.stats"         "key.ratios"        "dividends"
## [7] "splits"            "economic.data"    "exchange.rates"
## [10] "metal.prices"     "quandl"           "quandl.datatable"
```

이때 코스피와 코스닥을 이르는 이름이 각각 ^KS11와 ^KOSDAQ입니다. 각각 한번 가져와 보겠습니다.

```
tq_get("^KS11")
```

```
## # A tibble: 2,627 x 7
##       date   open   high   low close volume adjusted
##       <date>   <dbl>   <dbl>   <dbl>   <dbl>   <dbl>
## 1 2007-01-02 1438.89 1439.71 1430.06 1435.26 147800 1435.26
## 2 2007-01-03 1436.42 1437.79 1409.31 1409.35 203200 1409.35
## 3 2007-01-04 1410.55 1411.12 1388.50 1397.29 241200 1397.29
## 4 2007-01-05 1398.60 1400.59 1372.36 1385.76 277200 1385.76
## 5 2007-01-08 1376.76 1384.65 1366.48 1370.81 177600 1370.81
## 6 2007-01-09 1376.71 1381.99 1367.74 1374.34 216800 1374.34
## 7 2007-01-10 1372.52 1372.52 1345.08 1355.79 225400 1355.79
## 8 2007-01-11 1357.57 1375.31 1355.63 1365.31 211800 1365.31
## 9 2007-01-12 1379.00 1389.00 1372.87 1388.37 213800 1388.37
## 10 2007-01-15 1396.87 1397.64 1385.81 1390.96 163800 1390.96
## # ... with 2,617 more rows
```

```
tq_get("^KOSDAQ")
```

```
## # A tibble: 1 x 7
##       date   open   high   low close volume adjusted
##       <date>   <dbl>   <dbl>   <dbl>   <dbl>   <dbl>
## 1 2017-07-21 677.84 678.15 674.6 676.6 523728 676.6
```

각 기업의 주가를 가져오려면 종목 번호를 알고 있어야 합니다. 양식은 종목 번호.KS입니다. 종목번호는 세종기업 데이터에서 가져온 정보를 활용하겠습니다.

```
url<-"https://github.com/mrchypark/sejongFinData/raw/master/codeData.csv"
download.file(url,destfile = "./codeData.csv")
codeData<-read.csv("./codeData.csv",stringsAsFactors = F)
head(codeData)
```

```
##   종목번호   종목명
## 1  005930   삼성전자
## 2  000660   SK하이닉스
## 3  005935   삼성전자우
## 4  005380   현대차
```

```
## 5 035420 NAVER
## 6 015760 한국전력
```

삼성전자를 가져와 볼까요.

```
tar<-paste0(codeData[grep("^(삼성전자$",codeData$`종목명`),1], ".KS")
tq_get(tar)
```

```
## # A tibble: 2,627 x 7
##       date   open   high    low close volume adjusted
##   <date>   <dbl>   <dbl>   <dbl>   <dbl>   <dbl>
## 1 2007-01-02 698337.0 706221.5 693831.6 626000 352146 555777.5
## 2 2007-01-03 706221.5 707347.8 688199.9 614000 393530 545123.6
## 3 2007-01-04 688200.0 689326.3 679189.2 606000 365178 538020.9
## 4 2007-01-05 684820.8 685947.1 670178.2 595000 568008 528255.0
## 5 2007-01-08 666799.1 669051.8 652156.6 584000 661631 518489.0
## 6 2007-01-09 661167.6 669052.0 655535.8 586000 402197 520264.5
## 7 2007-01-10 655535.7 657788.4 647651.3 578000 515126 513162.0
## 8 2007-01-11 652156.7 662293.8 651030.3 583000 596787 517601.1
## 9 2007-01-12 664546.6 682568.2 658914.9 606000 806921 538020.9
## 10 2007-01-15 689326.2 692705.2 683694.4 612000 673506 543348.0
## # ... with 2,617 more rows
```

날짜를 지정할 수도 있습니다.

```
tq_get(tar, from="2016-01-01", to="2016-05-05")
```

```
## # A tibble: 84 x 7
##       date   open   high    low close volume adjusted
##   <date>   <dbl>   <dbl>   <dbl>   <dbl>   <dbl>
## 1 2016-01-04 1284780 1284780 1228698 1205000 306939 1181759
## 2 2016-01-05 1225639 1241954 1209325 1208000 216002 1184701
## 3 2016-01-06 1231757 1231757 1190971 1175000 366752 1152337
## 4 2016-01-07 1188931 1206266 1173636 1163000 282388 1140569
## 5 2016-01-08 1185872 1209325 1185872 1171000 257763 1148415
## 6 2016-01-11 1178735 1188931 1168538 1152000 241277 1129781
## 7 2016-01-12 1170577 1188931 1166499 1146000 206283 1123897
## 8 2016-01-13 1175676 1181794 1170578 1148000 143316 1125858
## 9 2016-01-14 1153243 1164459 1153243 1138000 209022 1116051
## 10 2016-01-15 1162420 1174656 1146105 1132000 209464 1110167
## # ... with 74 more rows
```

배당금 정보는 dividends에서 확인하시면 됩니다.

```
tq_get(tar, get = "dividends")
```

```
## # A tibble: 21 x 2
##       date dividends
##   <date>     <dbl>
## 1 2007-06-28     500
## 2 2007-12-27    7500
## 3 2008-06-27     500
## 4 2008-12-29   5000
## 5 2009-06-29     500
## 6 2009-12-29   7500
## 7 2010-06-29   4230
## 8 2010-12-29   5000
## 9 2011-06-29     500
## 10 2011-12-28   5000
## # ... with 11 more rows
```

야후 파이낸스가 데이터 소스이다 보니 모든 정보가 있다고 보기 어렵니다. 네이버 파이낸스에서 크롤링 하는 것이 방법이라고 생각합니다.

## Quandl

Quandl은 방대한 양의 경제, 주식에 대한 정보를 가지고 서비스하는 데이터 판매 기업입니다. Quandl이라는 자체 패키지만을 사용해도 되고, tidyquant가 내장하고 있어서 같이 사용해도 됩니다.

## tidyverse와 함께 사용하는 시계열 데이터

그동안의 주식관련 패키지들은 파이프 연산자 %>%와 함께 사용하지 못했는데, tidyquant는 그런 문제를 해결하였습니다. 아래 2가지 중요한 함수를 추가함으로써 dplyr과 tidyverse의 함수와 함께 사용할 수 있게 되었습니다.

- tq\_transmute(): 계산된 내용의 컬럼만으로 데이터를 구성합니다.
- tq\_mutate(): 데이터에 계산된 내용의 컬럼을 추가합니다.

# tq\_에서 계산 가능한 함수들

tq\_transmute\_fun\_options() 함수는 각 참고 패키지에서 활용할 수 있는 함수의 리스트를 보여줍니다. 모두 zoo, xts, quantmod, TTR, PerformanceAnalytics의 5개 패키지내의 함수를 지원합니다.

```
tq_transmute_fun_options() %>% str
```

```
## List of 5
## $ zoo          : chr [1:14] "rollapply" "rollapplyr" "rollmax" "rollmax.default" ...
## $ xts          : chr [1:27] "apply.daily" "apply.monthly" "apply.quarterly" "apply.weekly" ...
## $ quantmod     : chr [1:25] "allReturns" "annualReturn" "CICI" "dailyReturn" ...
## $ TTR          : chr [1:62] "adjRatios" "ADX" "ALMA" "aroon" ...
## $ PerformanceAnalytics: chr [1:7] "Return.annualized" "Return.annualized.excess" "Return.clean" "Return.cumulative" ...
```

## zoo 함수

```
tq_transmute_fun_options()$zoo
```

```
## [1] "rollapply"      "rollapplyr"      "rollmax"
## [4] "rollmax.default" "rollmaxr"       "rollmean"
## [7] "rollmean.default" "rollmeanr"      "rollmedian"
## [10] "rollmedian.default" "rollmedianr"    "rollsum"
## [13] "rollsum.default"  "rollsumr"
```

### ■ 롤링관련 함수 :

- 롤링 마진에 기능을 적용하는 일반적인 기능.
- *form*  
`:rollapply(data, width, FUN, ..., by = 1, by.column = TRUE, fill = if (na.pad) NA, na.pad = FALSE, partial = FALSE, a)`
- 옵션에는 *rollmax, rollmean, rollmedian, rollsum* 등이 있습니다.

## xts 함수

```
tq_transmute_fun_options()$xts
```

```
## [1] "apply.daily"      "apply.monthly"    "apply.quarterly"
## [4] "apply.weekly"      "apply.yearly"     "diff.xts"
## [7] "lag.xts"           "period.apply"   "period.max"
## [10] "period.min"        "period.prod"    "period.sum"
## [13] "periodicity"       "to.daily"       "to.hourly"
## [16] "to.minutes"        "to.minutes10"  "to.minutes15"
## [19] "to.minutes3"       "to.minutes30"  "to.minutes5"
## [22] "to.monthly"        "to.period"      "to.quarterly"
## [25] "to.weekly"         "to.yearly"      "to_period"
```

## ■ 기간 적용 기능 :

- 기능을 시간 세그먼트 (예 : `max`, `min`, `mean` 등)에 적용합니다.
- 양식 `:apply.daily` (`x`, `FUN`, ...).
- 옵션은 `apply.daily, weekly, monthly, quarterly, yearly`를 포함합니다.

## ■ 기간 기능 :

- 시계열을 낮은 주기성의 시계열로 변환합니다 (예 : 매일 매일의 주기성으로 변환).
- 형식 `:to.period` (`x`, `period = 'months'`, `k = 1`, `indexAt`, `name = NULL`, `OHLC = TRUE`, ...).
- 옵션에는 `to.minutes, hourly, daily, weekly, monthly, quarterly, yearly`가 포함됩니다.
- 참고 `:to.period`와 `:to.monthly` (`:to.weekly, :to.quarterly` 등) 양식의 리턴 구조는 다릅니다. `:to.period`는 날짜를 반환하고, `:to.months`는 `MON YYYY` 문자를 반환합니다. `lubridate`를 통해 시계열로 작업하고 싶다면 `:to.period`을 사용하는 것이 가장 좋습니다.

## quantmod 함수

```
tq_transmute_fun_options()$quantmod
```

```
## [1] "allReturns"      "annualReturn"   "CICI"
## [4] "dailyReturn"     "Delt"          "HiCI"
## [7] "Lag"             "LoCI"          "LoHi"
```

```

## [10] "monthlyReturn"    "Next"          "OpCl"
## [13] "OpHi"           "OpLo"          "OpOp"
## [16] "periodReturn"    "quarterlyReturn" "seriesAccel"
## [19] "seriesDecel"     "seriesDecr"    "seriesHi"
## [22] "seriesIncr"      "seriesLo"       "weeklyReturn"
## [25] "yearlyReturn"

```

## ■ 비율 변경 (Delt) 및 Lag 기능

- *Delt* :*Delt* (*x1*, *x2* = *NULL*, *k* = 0, *type* = *c* ("arithmetic", "log"))
  - Delt의 변형 : CICl, HiCl, LoCl, LoHi, OpCl, OpHi, OpLo, OpOp
  - 양식 :*OpCl* (OHLC)
- *Lag* :*Lag*(*x*, *k* = 1) / *Next* :*Next*(*x*, *k* = 1) (*dplyr* :: *lag*과 *dplyr* :: *lead*도 사용할 수 있습니다)

## ■ 기간 반환 함수 :

- 매일, 매주, 매월, 분기 별 및 연간을 포함하는 다양한 주기에 대한 산술 또는 로그 반환을 가져옵니다.
- 형식 :*periodReturn* (*x*, *period* = 'monthly', 부분 집합 = *NULL*, *type* = 'arithmetic', *leading* = *TRUE*, ...)

## ■ 시리즈 기능 :

- 계열을 설명하는 반환 값. 옵션에는 증감, 가감 및 고저 설명이 포함됩니다.
- 양식 :*seriesHi* (*x*), *seriesIncr* (*x*, *thresh* = 0, *diff.* = 1L), *seriesAccel* (*x*)

## TTR 함수

```
tq_transmute_fun_options()$TTR
```

```

## [ 1] "adjRatios"        "ADX"          "ALMA"
## [ 4] "aroon"           "ATR"          "BBands"
## [ 7] "CCI"             "chaikinAD"    "chaikinVolatility"
## [10] "CLV"              "CMF"          "CMO"
## [13] "DEMA"             "DonchianChannel" "DPO"
## [16] "DVI"              "EMA"          "EMV"
## [19] "EVWMA"            "GMMA"         "growth"
## [22] "HMA"              "KST"          "lags"
## [25] "MACD"             "MFI"          "momentum"

```

```

## [28] "OBV"
## [31] "rollSFM"
## [34] "runCov"
## [37] "runMean"
## [40] "runPercentRank"
## [43] "runVar"
## [46] "SMI"
## [49] "TDI"
## [52] "VHF"
## [55] "VWAP"
## [58] "williamsAD"
## [61] "ZigZag"
"PBands"
"RSI"
"runMAD"
"runMedian"
"runSD"
"SAR"
"SNR"
"TRIX"
"VMA"
"VWMA"
"WMA"
"ZLEMA"
"ROC"
"runCor"
"runMax"
"runMin"
"runSum"
"SMA"
"stoch"
"ultimateOscillator"
"volatility"
"wilderSum"
"WPR"

```

- 웨즈 와일더의 방향 운동 지수 : \*ADX (HLC, n = 14, maType, ...)
- 볼린저 밴드 :
  - BBands (HLC, n = 20, maType, sd = 2, ...): 볼린저 밴드
- 변화율 / 운동량 : ROC (x, n = 1, type = c ("연속", "이산"), na.pad = TRUE): 변화율  
운동량 (x, n = 1, na.pad = TRUE): 운동량
- 이동 평균 (maType) : SMA (x, n = 10, ...): 단순 이동 평균  
EMA (x, n = 10, wilder = FALSE, ratio = NULL, ...): 지수 이동 평균
  - DEMA (x, n = 10, v = 1, wilder = FALSE, ratio = NULL): 이중 지수 이동 평균
  - WMA (x, n = 10, wts = 1:n, ...): 가중 이동 평균
  - EVWMA (가격, 수량, n = 10, ...): 탄성 체중 이동 평균 ZLEMA (x, n = 10, 비율 = NULL, ...): Zero Lag Exponential Moving Average VWAP (가격, 물량, n = 10, ...): 물량 가중 평균 가격
  - VMA (x, w, 비율 = 1, ...): 가변 길이 이동 평균 HMA (x, n = 20, ...): 선체 이동 평균  
ALMA (x, n = 9, offset = 0.85, sigma = 6, ...): Arnaud Legoux 이사 평균
- MACD Oscillator : MACD (x, nFast = 12, nSlow = 26, nSig = 9, maType, percent = TRUE, ...)
- 상대 강도 지수 : \*RSI (가격, n = 14, maType, ...)

- runFun : `runSum (x, n = 10, cumulative = FALSE)`: n-기간 이동 윈도우에 대한 합계를 반환합니다.  
`runMin (x, n = 10, cumulative = FALSE)`: n-기간 이동 윈도우에 대한 최소값을 반환합니다.  
`runMax (x, n = 10, cumulative = FALSE)`: n-기간 이동 윈도우에 대해 최대 값을 반환합니다.  
`runMean (x, n = 10, cumulative = FALSE)`: n-period 이동 윈도우를 의미합니다.  
\*`runMedian (x, n = 10, non.unique = "mean", cumulative = FALSE)`: n-period 이동 윈도우에 대한 중앙값을 반환합니다.
  - `runCov (x, y, n = 10, use = "all.obs", sample = TRUE, 누적 = FALSE)`: n-period 이동 윈도우에 대한 공분산을 반환합니다. `runCor (x, y, n = 10, use = "all.obs", sample = TRUE, 누적 = FALSE)`: n-period 이동 윈도우에 대한 상관관계를 반환합니다. `runVar (x, y = NULL, n = 10, 샘플 = TRUE, 누적 = FALSE)`: n-기간 이동 윈도우에 대한 분산을 반환합니다. `runSD (x, n = 10, 샘플 = TRUE, 누적 = FALSE)`: n-기간 이동 윈도우에 대한 표준 편차를 반환합니다. `runMAD (x, n = 10, center = NULL, stat = "중간 값", 상수 = 1.4826, non.unique = "평균", cumulative = FALSE)`: n-기간 이동에 대한 중간 / 평균 절대 편차를 반환합니다. 창문. `wilderSum (x, n = 10)`: n-기간 이동 윈도우에 대해 Welles Wilder 스타일 가중치 합계를 되돌린다.
- Stochastic Oscillator / Stochastic Momentum Index : Stochastic Oscillator (HLC, nFastK = 14, nFastD = 3, nSlowD = 3, maType, bounded = TRUE, smooth = 1, ...)  
  - `SMI (HLC, n = 13, nFast = 2, nSlow = 25, nSig = 9, maType, bounded = TRUE, ...)` : 확률 모멘텀 지수

## PerformanceAnalytics 함수

```
tq_transmute_fun_options()$PerformanceAnalytics
```

```
## [1] "Return.annualized"           "Return.annualized.excess"
## [3] "Return.clean"               "Return.cumulative"
## [5] "Return.excess"              "Return.Geltner"
## [7] "zerofill"
```

`Return.annualized` 및 `Return.annualized.excess` : 기간 반환을 취하여 연간 수익으로 통합합니다. `Return.clean` : 반환 값에서 특이 값 제거합니다. `Return.excess` : 무위험 이자율을 초과하는 수익률로 수익률에서 무위험 이

자율을 제거합니다. `zero_fill` : 'NA'값을 0으로 대체하는 데 사용됩니다.

## ggplot2와 연계된 차트 그리기

ggplot2 차트를 그리는데 R에서 가장 유명한 패키지입니다. gg는 Grammar of Graphics의 줄임말로 그림을 생성하는 것에 대한 규칙을 제안하고 있습니다. tidyquant은 ggplot2에 더해 아래와 같은 기능을 추가로 제공합니다.

- **차트 종류** : 두 개의 차트 타입 시각화는 `geom_bar`, `geom_barh`과 `geom_candlestick`을 사용하여 가능합니다.
- **이동 평균** : '`geom_ma`'를 사용하여 7 개의 이동 평균 시각화를 사용할 수 있습니다.
- **Bollinger Bands** : Bollinger 밴드는 '`geom_bbands`'를 사용하여 시각화 할 수 있습니다. BBand 이동 평균은 이동 평균에서 사용할 수 있는 7 가지 중 하나 일 수 있습니다.
- **날짜 범위 확대** : 차트의 특정 영역을 확대 할 때 데이터 손실을 방지하는 두 가지 `coord` 함수 (`coord_x_date` 및 `coord_x_datetime`)를 사용할 수 있습니다. 이것은 이동 평균 및 Bollinger 밴드 기하학을 사용할 때 중요합니다.

## 살펴보기

`ta_get`을 이용해서 사용할 데이터를 가져옵니다. 내장 데이터인 FANG과 애플, 아마존을 예시로 사용하겠습니다.

```
data("FANG")
AAPL <- ta_get("AAPL", get = "stock.prices", from = "2015-09-01", to = "2016-12-31")
AMZN <- ta_get("AMZN", get = "stock.prices", from = "2000-01-01", to = "2016-12-31")
```

'end` 매개 변수는 예제 전체에서 날짜 제한을 설정할 때 사용됩니다.

```
end <- as_date("2016-12-31")
```

## 차트 종류

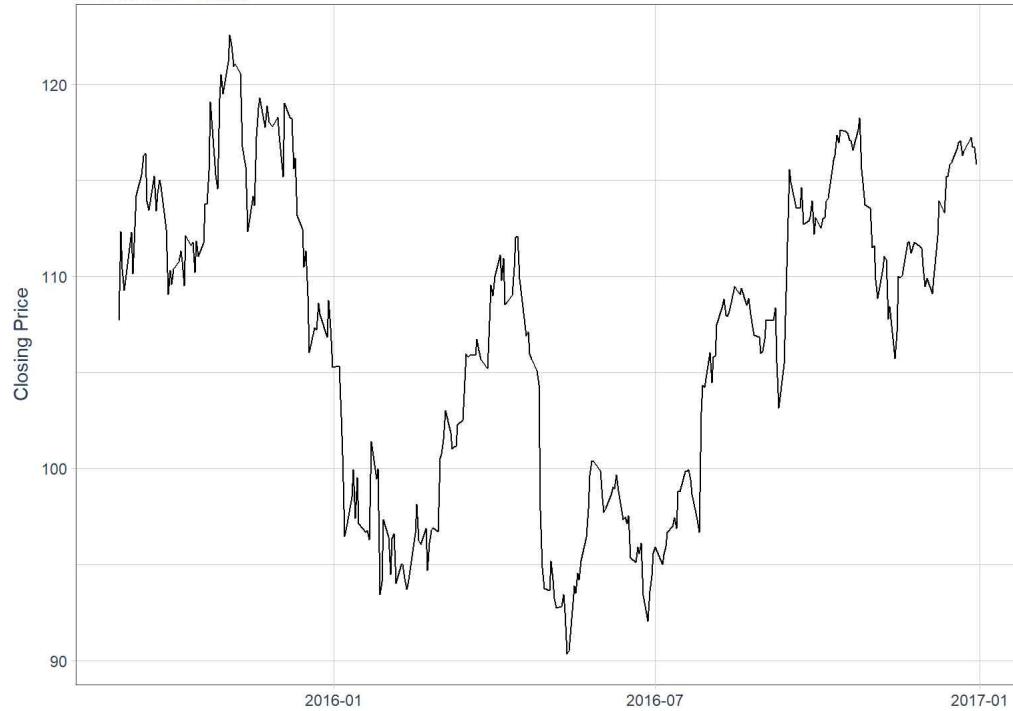
- Bar Chart: geom\_bar chart을 사용합니다.
- Candlestick Chart: geom\_candlestick을 사용합니다.

## 라인 차트

tidyquant의 geom\_함수를 사용하여 가로 막대형 차트와 촛대형 차트를 시각화하기 전에 단순한 선 차트로 주가를 시각화하여 그래픽 문법을 확인해보겠습니다. 이것은 ggplot2 패키지의 geom\_line을 사용하여 이루어집니다. 주식 데이터로 시작하고 파이프 연산자 (%>%)를 사용하여 ggplot ()함수로 보냅니다.

```
AAPL %>%  
  ggplot(aes(x = date, y = close)) +  
  geom_line() +  
  labs(title = "AAPL Line Chart", y = "Closing Price", x = "") +  
  theme_tq()
```

AAPL Line Chart



## 바 차트

바 차트는 `geom_line`을 `geom_barchart`로 바꾸는 걸로 해결됩니다. `aes()`내의 내용을 의미에 맞게 조정하는 것으로 바 차트를 그리는 것이 끝납니다.

```
AAPL %>%
  ggplot(aes(x = date, y = close)) +
  geom_barchart(aes(open = open, high = high, low = low, close = close)) +
  labs(title = "AAPL Bar Chart", y = "Closing Price", x = "") +
  theme_tq()
```



우리는 `coord_x_date`를 사용하여 특정 섹션을 확대 / 축소합니다. 이 섹션에는 `xlim` 및 `ylim` 인수가 `c (start, end)`로 지정되어 차트의 특정 영역에 초점을 맞춥니다. `xlim`의 경우 우리는 `lubridate`를 사용하여 문자 날짜를 날짜 클래스로 변환한 다음 `weeks ()` 함수를 사용하여 6 주를 뽑습니다. `ylim`의 경우 가격을 100에서 120까지 확대합니다.

```
AAPL %>%
  ggplot(aes(x = date, y = close)) +
  geom_barchart(aes(open = open, high = high, low = low, close = close)) +
  labs(title = "AAPL Bar Chart",
       subtitle = "Zoomed in using coord_x_date",
       y = "Closing Price", x = "") +
  coord_x_date(xlim = c(end - weeks(6), end),
               ylim = c(100, 120)) +
  theme_tq()
```



색상은color\_up 및color\_down 인수를 사용하여 수정할 수 있으며size와 같은 매개 변수를 사용하여 모양을 제어 할 수 있습니다.

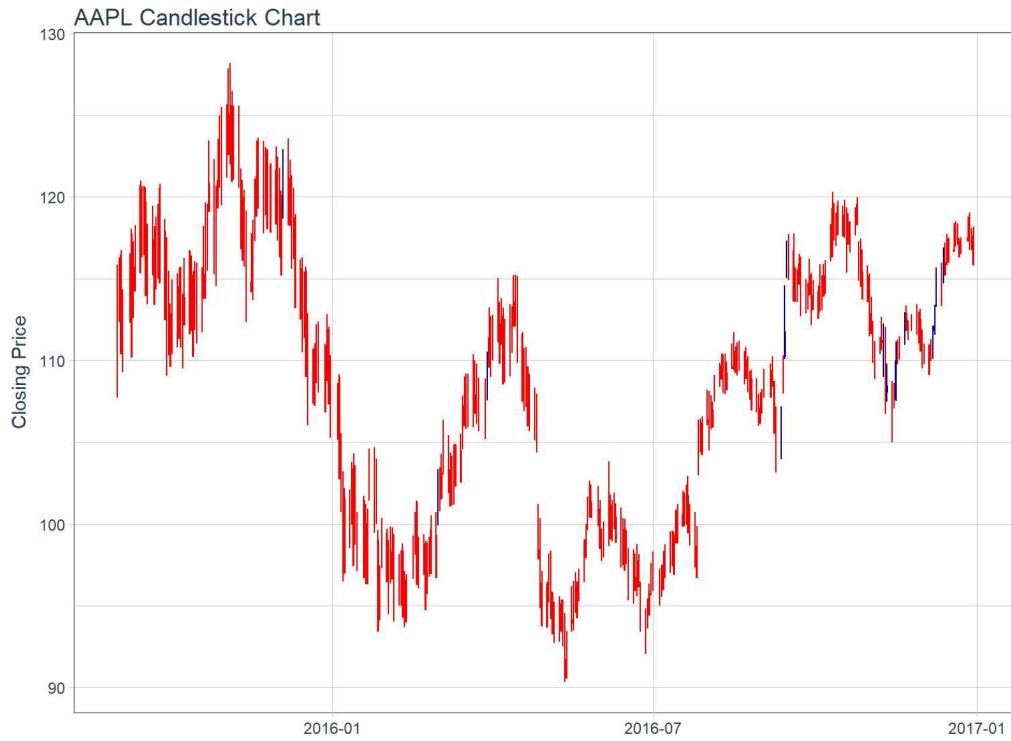
```
AAPL %>%
  ggplot(aes(x = date, y = close)) +
  geom_barchart(aes(open = open, high = high, low = low, close = close),
                color_up = "darkgreen", color_down = "darkred", size = 1) +
  labs(title = "AAPL Bar Chart",
       subtitle = "Zoomed in, Experimenting with Formatting",
       y = "Closing Price", x = "") +
  coord_x_date(xlim = c(end - weeks(6), end),
               ylim = c(100, 120)) +
  theme_tq()
```



## 캔들 차트

캔들 차트 또한 바 차트를 그리는 것과 거의 같습니다.

```
AAPL %>%  
  ggplot(aes(x = date, y = close)) +  
  geom_candlestick(aes(open = open, high = high, low = low, close = close)) +  
  labs(title = "AAPL Candlestick Chart", y = "Closing Price", x = "") +  
  theme_tq()
```



색상은color\_up과color\_down을 사용하여 선 색상을 조절할 수 있고, fill\_up과fill\_down은 사각형을 채웁니다.

```

AAPL %>%
  ggplot(aes(x = date, y = close)) +
  geom_candlestick(aes(open = open, high = high, low = low, close = close),
                   color_up = "darkgreen", color_down = "darkred",
                   fill_up = "darkgreen", fill_down = "darkred") +
  labs(title = "AAPL Candlestick Chart",
       subtitle = "Zoomed in, Experimenting with Formatting",
       y = "Closing Price", x = "") +
  coord_x_date(xlim = c(end - weeks(6), end),
               ylim = c(100, 120)) +
  theme_tq()

```



## 여러개의 차트를 그리기

facet\_wrap을 사용하여 동시에 여러 주식을 시각화 할 수 있습니다. ggplot ()의 aes()에 group을 추가하고 ggplot 워크 플로우의 끝에서 facet\_wrap() 함수와 결합함으로써 네 개의 "FANG"주식을 동시에 모두 볼 수 있습니다.

```
start <- end - weeks(6)
FANG %>%
  filter(date >= start - days(2 * 15)) %>%
  ggplot(aes(x = date, y = close, group = symbol)) +
  geom_candlestick(aes(open = open, high = high, low = low, close = close)) +
  labs(title = "FANG Candlestick Chart",
       subtitle = "Experimenting with Multiple Stocks",
       y = "Closing Price", x = "") +
  coord_x_date(xlim = c(start, end)) +
```

```
facet_wrap(~ symbol, ncol = 2, scale = "free_y") +  
theme_tq()
```



## 트랜드 시각화

Moving averages are critical to evaluating time-series trends. tidyquant includes geoms to enable “rapid prototyping” to quickly visualize signals using moving averages and Bollinger bands.

### 이동 평균

tidyquant에서는 다양한 이동평균 함수를 제공합니다.

- Simple moving averages (SMA)
- Exponential moving averages (EMA)

- Weighted moving averages (WMA)
- Double exponential moving averages (DEMA)
- Zero-lag exponential moving averages (ZLEMA)
- Volume-weighted moving averages (VWMA) (also known as VWAP)
- Elastic, volume-weighted moving averages (EVWMA) (also known as MVWAP)

이동 평균은 `geom_ma` 함수로 차트에 추가 된 레이어로 적용됩니다. 기하 구조는 TTR 패키지에서 SMA, EMA, WMA, DEMA, ZLEMA, VWMA, EVWMA와 같은 기본 이동 평균 함수의 래퍼입니다.

## Example 1: 50일/200일 단순 이동 평균 차트 작성

```
AAPL %>%  
  ggplot(aes(x = date, y = close)) +  
  geom_candlestick(aes(open = open, high = high, low = low, close = close)) +  
  geom_ma(ma_fun = SMA, n = 50, linetype = 5, size = 1.25) +  
  geom_ma(ma_fun = SMA, n = 200, color = "red", size = 1.25) +  
  labs(title = "AAPL Candlestick Chart",  
       subtitle = "50 and 200-Day SMA",  
       y = "Closing Price", x = "") +  
  coord_x_date(xlim = c(end - weeks(24), end),  
               ylim = c(100, 120)) +  
  theme_tq()
```

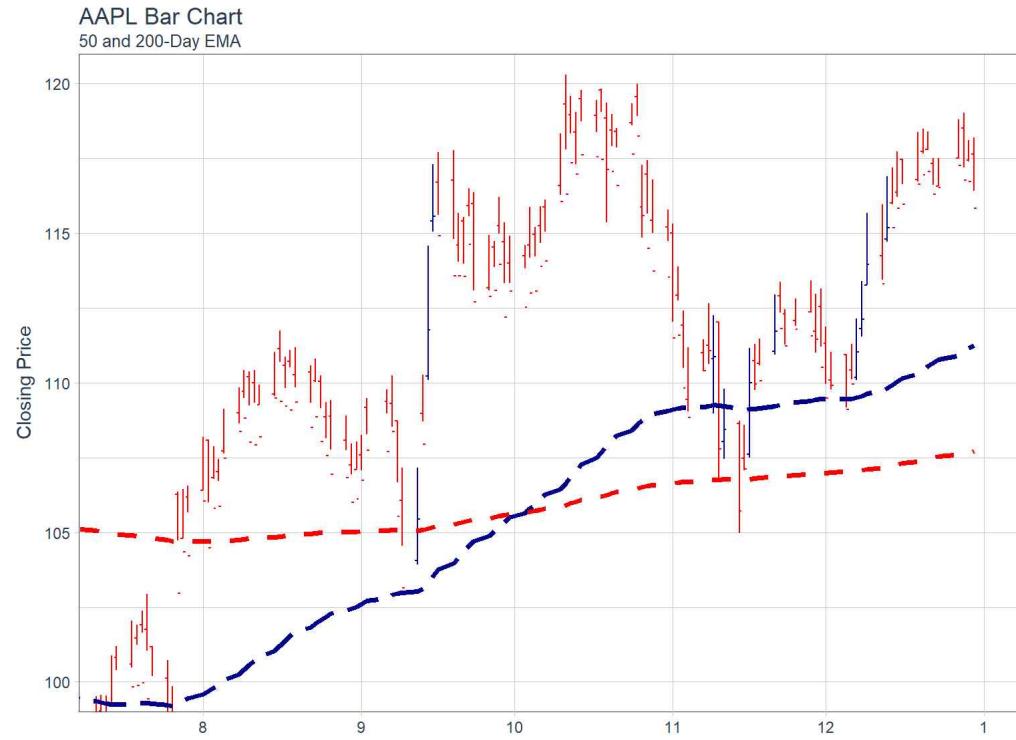


## Example 2: 지수 이동 평균 차트

```

AAPL %>%
  ggplot(aes(x = date, y = close)) +
  geom_barchart(aes(open = open, high = high, low = low, close = close)) +
  geom_ma(ma_fun = EMA, n = 50, wilder = TRUE, linetype = 5, size = 1.25) +
  geom_ma(ma_fun = EMA, n = 200, wilder = TRUE, color = "red", size = 1.25) +
  labs(title = "AAPL Bar Chart",
       subtitle = "50 and 200-Day EMA",
       y = "Closing Price", x = "") +
  coord_x_date(xlim = c(end - weeks(24), end),
               ylim = c(100, 120)) +
  theme_tq()

```



## 볼린저 밴드

[Bollinger Bands] [https://en.wikipedia.org/wiki/Bollinger\\_Bands](https://en.wikipedia.org/wiki/Bollinger_Bands))는 이동 평균(일반적으로 상하 2SD) 주위의 범위를 플로팅하여 변동성을 시각화하는 데 사용됩니다. 그것들은 이동 평균을 사용하기 때문에, geom\_bbands 함수는 geom\_ma와 거의 동일하게 작동합니다. 동일한 7 개의 이동 평균이 호환됩니다. 가장 큰 차이점은 기본적으로 2 인 표준 편차 인sd 인수와 밴드를 계산하는 데 필요한 'high', 'low' 및 'close'를 aes()에 추가하는 것입니다.

### Example 1: SMA를 사용하여 BBands 적용

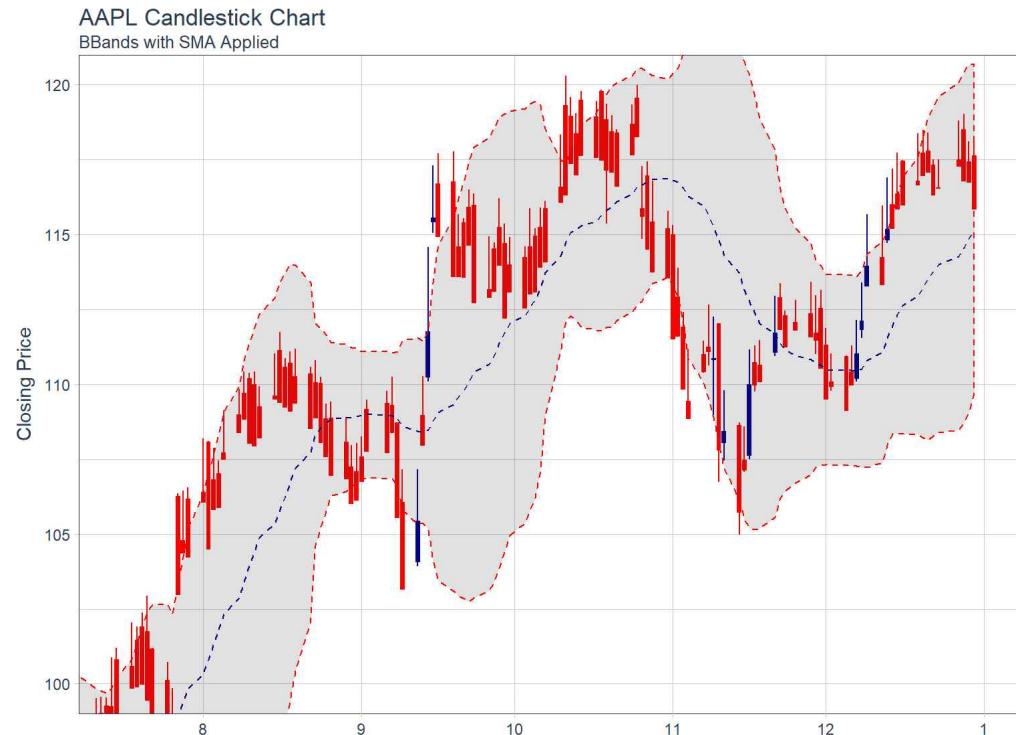
간단한 이동 평균을 사용하여 Bollinger Bands를 추가하는 기본 예제를 살펴 보겠습니다.

```
AAPL %>%
  ggplot(aes(x = date, y = close, open = open,
```

```

high = high, low = low, close = close)) +
geom_candlestick() +
geom_bbands(ma_fun = SMA, sd = 2, n = 20) +
labs(title = "AAPL Candlestick Chart",
subtitle = "BBands with SMA Applied",
y = "Closing Price", x = "") +
coord_x_date(xlim = c(end - weeks(24), end),
ylim = c(100, 120)) +
theme_tq()

```



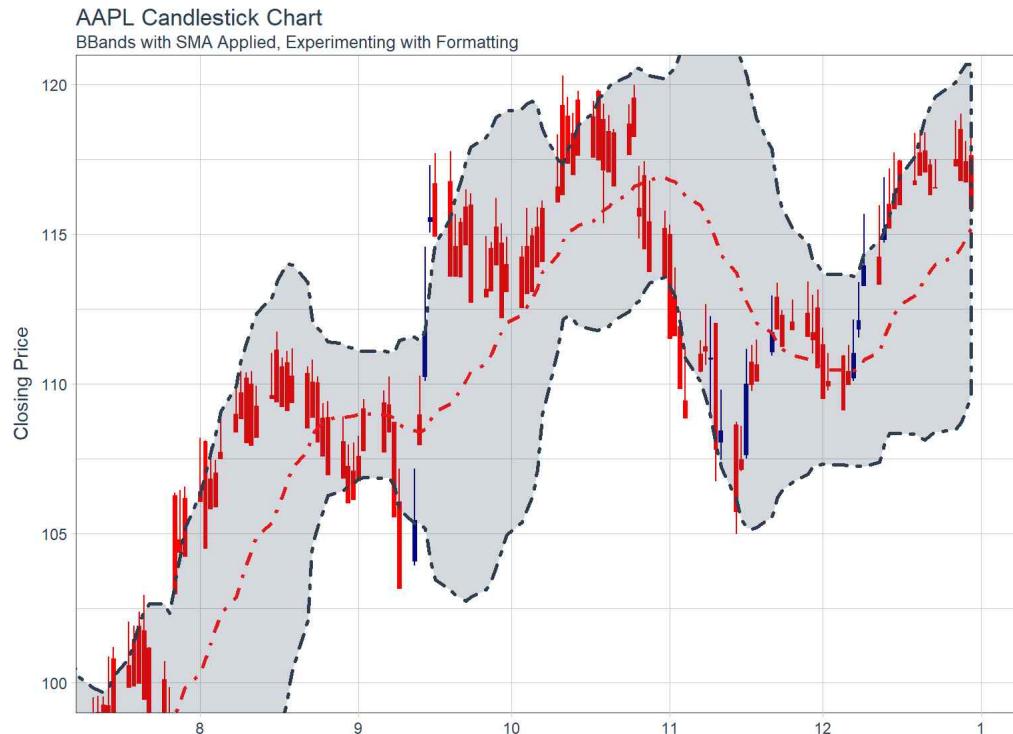
## Example 2: Bollinger Bands의 모양 바꾸기

모양은 color\_ma, color\_bands, alpha, fill 인자를 사용하여 수정할 수 있습니다. BBands에 새로운 서식을 적용한 Example 1과 같은 그림이 있습니다.

```

AAPL %>%
  ggplot(aes(x = date, y = close, open = open,
             high = high, low = low, close = close)) +
  geom_candlestick() +
  geom_bbands(ma_fun = SMA, sd = 2, n = 20,
              linetype = 4, size = 1, alpha = 0.2,
              fill     = palette_light()[[1]],
              color_bands = palette_light()[[1]],
              color_ma    = palette_light()[[2]]) +
  labs(title = "AAPL Candlestick Chart",
       subtitle = "BBands with SMA Applied, Experimenting with Formatting",
       y = "Closing Price", x = "") +
  coord_x_date(xlim = c(end - weeks(24), end),
               ylim = c(100, 120)) +
  theme_tq()

```



### Example 3: 여러 주식에 BBands 추가

```

start <- end - weeks(24)
FANG %>%
  filter(date >= start - days(2 * 20)) %>%
  ggplot(aes(x = date, y = close,
             open = open, high = high, low = low, close = close,
             group = symbol)) +
  geom_barchart() +
  geom_bbands(ma_fun = SMA, sd = 2, n = 20, linetype = 5) +
  labs(title = "FANG Bar Chart",
       subtitle = "BBands with SMA Applied, Experimenting with Multiple Stocks",
       y = "Closing Price", x = "") +
  coord_x_date(xlim = c(start, end)) +
  facet_wrap(~ symbol, ncol = 2, scales = "free_y") +
  theme_tq()

```



## ggplot2 함수

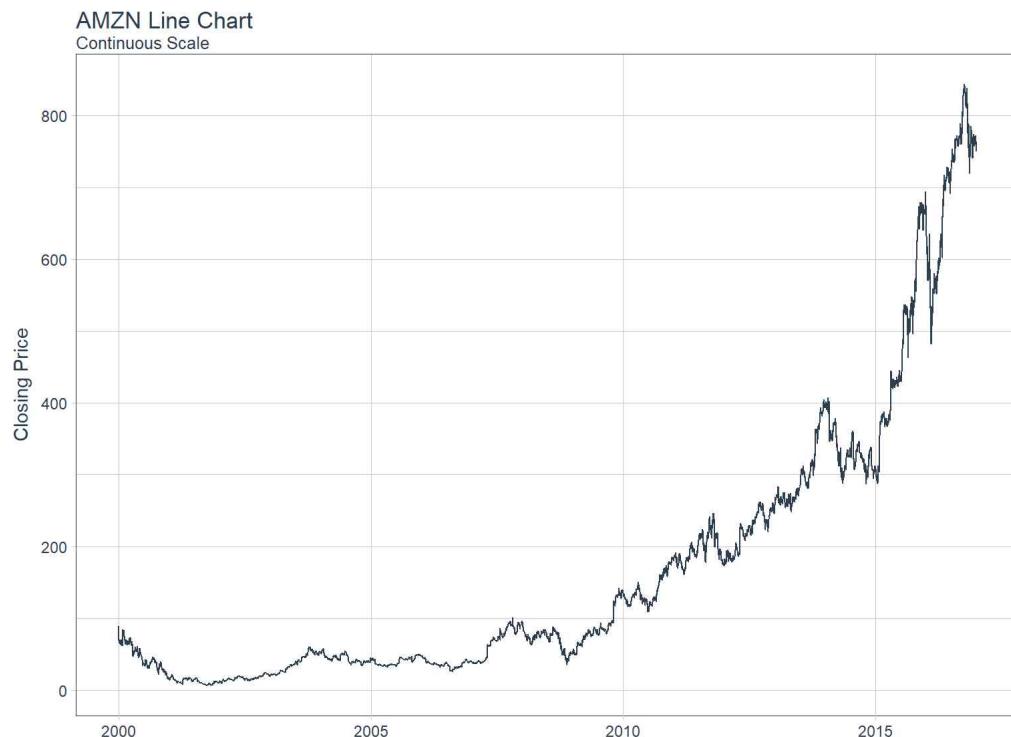
기본 ggplot2는 재무 데이터를 분석하는데 유용한 많은 기능을 가지고 있습니다. 아마존(AMZN)을 사용하여 몇 가지 간단한 예제를 살펴 보겠습니다.

## Example 1 : scale\_y\_log10을 사용한 로그 스케일

ggplot2는 y 축을 로그 스케일로 스케일하기위한scale\_y\_log10 ()함수를 가지고 있습니다. 이는 분석 할 수있는 선형 추세를 조정하는 경향이 있으므로 매우 유용합니다.

### Continuous Scale:

```
AMZN %>%
  ggplot(aes(x = date, y = adjusted)) +
  geom_line(color = palette_light()[[1]]) +
  scale_y_continuous() +
  labs(title = "AMZN Line Chart",
       subtitle = "Continuous Scale",
       y = "Closing Price", x = "") +
  theme_tq()
```



## Log Scale:

```
AMZN %>%
  ggplot(aes(x = date, y = adjusted)) +
  geom_line(color = palette_light()[[1]]) +
  scale_y_log10() +
  labs(title = "AMZN Line Chart",
       subtitle = "Log Scale",
       y = "Closing Price", x = "") +
  theme_tq()
```



## Example 2: geom\_smooth로 회귀 추세선

우리는 워크 플로우에 geom\_smooth () 함수를 빠르게 추가하는 추세선을 적용 할 수 있습니다. 이 함수는 선형(lm)과 loess(loess)를 포함한 몇 가지 예측 방법을 가지고 있습니다.

## Linear:

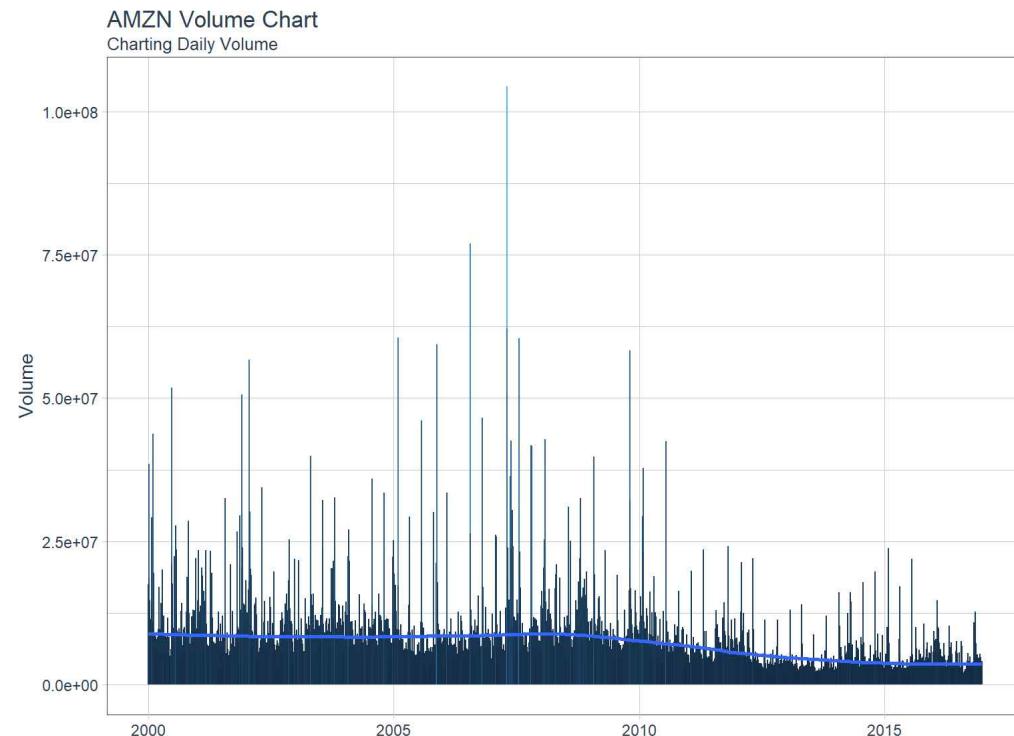
```
AMZN %>%  
  ggplot(aes(x = date, y = adjusted)) +  
  geom_line(color = palette_light()[[1]]) +  
  scale_y_log10() +  
  geom_smooth(method = "lm") +  
  labs(title = "AMZN Line Chart",  
       subtitle = "Log Scale, Applying Linear Trendline",  
       y = "Adjusted Closing Price", x = "") +  
  theme_tq()
```



## Example 3: geom\_segment로 차트 볼륨

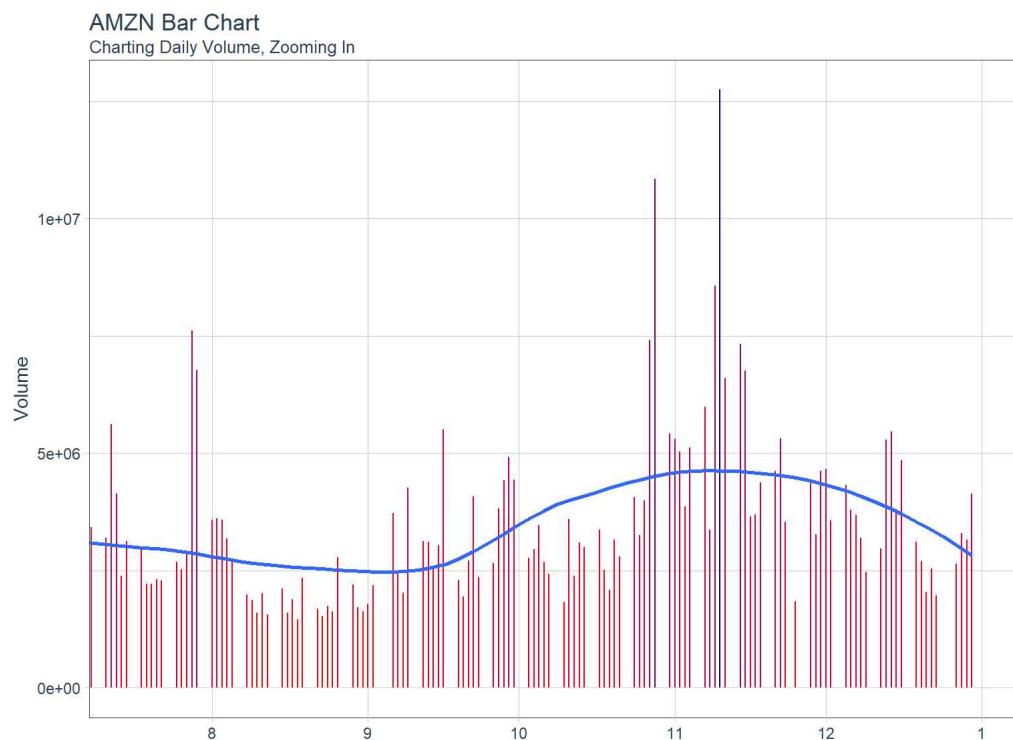
우리는 geom\_segment () 함수를 사용하여 라인의 시작과 끝을 xy 점으로하는 일일 볼륨을 차트로 표시 할 수 있습니다. aes()를 사용하여 볼륨의 값을 기준으로 색상을 지정하여 이러한 데이터를 강조 표시합니다.

```
AMZN %>%
  ggplot(aes(x = date, y = volume)) +
  geom_segment(aes(xend = date, yend = 0, color = volume)) +
  geom_smooth(method = "loess", se = FALSE) +
  labs(title = "AMZN Volume Chart",
       subtitle = "Charting Daily Volume",
       y = "Volume", x = "") +
  theme_tq() +
  theme(legend.position = "none")
```



특정 지역을 확대 할 수 있습니다. `scale_color_gradient`를 사용하여 고점 및 저점을 빠르게 시각화 할 수 있으며 `geom_smooth`를 사용하여 추세를 볼 수 있습니다.

```
start <- end - weeks(24)
AMZN %>%
  filter(date >= start - days(50)) %>%
  ggplot(aes(x = date, y = volume)) +
  geom_segment(aes(xend = date, yend = 0, color = volume)) +
  geom_smooth(method = "loess", se = FALSE) +
  labs(title = "AMZN Bar Chart",
       subtitle = "Charting Daily Volume, Zooming In",
       y = "Volume", x = "") +
  coord_x_date(xlim = c(start, end)) +
  scale_color_gradient(low = "red", high = "darkblue") +
  theme_tq() +
  theme(legend.position = "none")
```



# 테마

tidyquant 패키지는 3 가지 테마로 구성되어있어 신속하게 재무 차트를 조정 할 수 있습니다.

- **Light:** theme\_tq() + scale\_color\_tq() + scale\_fill\_tq()
- **Dark:** theme\_tq\_dark() + scale\_color\_tq(theme = "dark") + scale\_fill\_tq(theme = "dark")
- **Green:** theme\_tq\_green() + scale\_color\_tq(theme = "green") + scale\_fill\_tq(theme = "green")

## Dark

```
n_mavg <- 50 # Number of periods (days) for moving average
FANG %>%
  filter(date >= start - days(2 * n_mavg)) %>%
  ggplot(aes(x = date, y = close, color = symbol)) +
  geom_line(size = 1) +
  geom_ma(n = 15, color = "darkblue", size = 1) +
  geom_ma(n = n_mavg, color = "red", size = 1) +
  labs(title = "Dark Theme",
       x = "", y = "Closing Price") +
  coord_x_date(xlim = c(start, end)) +
  facet_wrap(~ symbol, scales = "free_y") +
  theme_tq_dark() +
  scale_color_tq(theme = "dark") +
  scale_y_continuous(labels = scales::dollar)
```

### Dark Theme

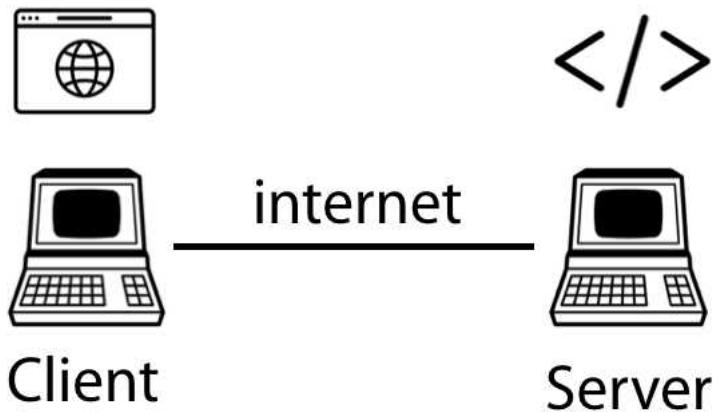


# 웹 크롤링 기초 httr과 rvest

## 서버와 클라이언트

인터넷은 각각의 컴퓨터를 통신으로 연결하여 정보를 주고 받는 것입니다. 전화번호와 같은 IP라는 것이 있고, 번호를 외우는게 어려워서 주소라는 것을 만들어 관리합니다. 8.8.8.8이 IP이고, www.google.com이 주소입니다. 뒤가 더 기억하기 쉽겠죠?

컴퓨터를 연결해서 자기 것처럼 사용하는 것을 원격 연결이라고 합니다. 그와 달리 인터넷의 웹페이지는 모두 제한된 텍스트 문서나 그림 같은 파일들만 확인할 있도록 구성되어 있습니다. 저런 문서와 그림을 가지고 있다가 약속된 요청을 하면 전송해주는 역할을 하것 것을 서버라고 하고, 문서와 그림을 규칙에 맞게 요청하고, 정해진대로 보여주는 역할을 하는 것을 클라이언트(브라우저)라고 합니다.



## http/s 1.1

http/s는 이렇게 서버에 파일을 요청하는 방식을 미리 약속해 둔 것입니다. 요청의 종류는 다양하지만 크롤링에서 사용할 종류는 GET과 POST입니다. GET은 지정된 주소에 저장되어 있는 파일을 달라고 요청하는 것이고, POST

는 내가 가지고 있는 데이터가 있는데 이걸 잘 바꿔서 결과를 돌려주세요 라고 하는 것입니다. 언어에서 함수와 비슷합니다.

- GET: 인터넷 주소를 기준으로 위치하고 있는 데이터나 파일을 달라고 요청하는 것.
- POST: GET과 같이 주소 위치에 요청하는데, 데이터를 같이 추가해서 요청하고, 결과를 받는 것.

위에 요청이라는 표현을 사용했는데, request와 response라는 개념이 있습니다. http 통신은 모두 요청을 해서 응답을 받는 구조입니다. GET은 정상이라고 하는 코드와 몇몇 파일들, 이외에 쿠키, 헤더 등을 응답으로 받습니다. POST는 GET과 같고, 계산 결과를 추가적으로 응답으로 보내기도 합니다.

위와 같은 요청을 하는 방식을 그대로 따라해서 정보를 받아오는 패키지가 curl, httr입니다. 이번에는 다루기 쉬운 httr을 사용해 보겠습니다.

## httr 패키지 소개

### 요청

httr 패키지는 GET요청을 할 있는 GET()함수가 있습니다.

```
library(httr)
```

```
## Warning: package 'httr' was built under R version 3.4.1
```

```
r <- GET("http://httpbin.org/get")
```

위에 요청으로 r이라는 응답 객체가 생깁니다. 응답 객체를 확인하면 사용된 실제 URL (모든 리디렉션 이후), http 상태, 파일 (내용) 유형, 크기 및 텍스트 파일 인 경우 처음 몇 줄의 출력과 같은 유용한 정보가 제공됩니다.

```
r
```

```
## Response [http://httpbin.org/get]
##   Date: 2017-07-22 05:27
##   Status: 200
##   Content-Type: application/json
##   Size: 326 B
## {
##   "args": {},
##   "headers": {
##     "Accept": "application/json, text/xml, application/xml, */*",
##     "Accept-Encoding": "gzip, deflate",
##     "Connection": "close",
##     "Host": "httpbin.org",
##     "User-Agent": "libcurl/7.53.1 r-curl/2.6 httr/1.2.1"
##   },
##   "origin": "221.148.180.35",
## }
```

다양한 헬퍼 메소드를 사용하여 응답의 중요한 부분을 추출하거나 객체로 직접 파고들 수 있습니다.

```
status_code(r)
```

```
## [1] 200
```

```
headers(r)
```

```
## $connection
## [1] "keep-alive"
##
## $server
## [1] "meinheld/0.6.1"
##
## $date
## [1] "Sat, 22 Jul 2017 05:27:49 GMT"
##
## $`content-type`
## [1] "application/json"
##
## $`access-control-allow-origin`
## [1] "*"
##
## $`access-control-allow-credentials`
## [1] "true"
##
```

```
## $`x-powered-by`  
## [1] "Flask"  
##  
## $`x-processed-time`  
## [1] "0.000625848770142"  
##  
## $`content-length`  
## [1] "326"  
##  
## $via  
## [1] "1.1 vegur"  
##  
## attr(,"class")  
## [1] "insensitive" "list"
```

```
str(content(r))
```

```
## List of 4  
## $ args : Named list()  
## $ headers:List of 5  
##   ..$ Accept : chr "application/json, text/xml, application/xml, */*"  
##   ..$ Accept-Encoding: chr "gzip, deflate"  
##   ..$ Connection : chr "close"  
##   ..$ Host : chr "httpbin.org"  
##   ..$ User-Agent : chr "libcurl/7.53.1 r-curl/2.6 httr/1.2.1"  
## $ origin : chr "221.148.180.35"  
## $ url : chr "http://httpbin.org/get"
```

이 소개에서 `httpbin.org` 를 사용할 것이다. 그것은 많은 종류의 http 요청을 받아 들여 그것이 수신 한 데이터를 설명하는 json을 반환합니다. 이렇게하면 httr이하는 일을 쉽게 볼 수 있습니다.

## 응답

서버에서 보낸 데이터는 상태 표시 줄, 헤더 및 본문의 세 부분으로 구성됩니다. 상태 표시 줄의 가장 중요한 부분은 http 상태 코드입니다. 요청이 성공했는지 여부를 알려줍니다. 해당 데이터에 액세스하는 방법, 본문 및 헤더에 액세스하는 방법을 보여 드리겠습니다.

## 상태코드

상태 코드는 요청이 성공했는지 여부를 요약하는 3 자리 숫자입니다 (대화중인 서버에서 정의한대로). `http_status()`를 사용하여 설명적인 메시지와 함께 상태 코드에 접근 할 수 있습니다.

```
r <- GET("http://httpbin.org/get")
http_status(r)
```

```
## $category
## [1] "Success"
##
## $reason
## [1] "OK"
##
## $message
## [1] "Success: (200) OK"
```

```
r$status_code
```

```
## [1] 200
```

요청이 성공하면 항상 200을 반환합니다. 일반적인 오류는 404(파일을 찾을 수 없음) 및 403 (사용 권한을 거부함)입니다. 웹 API와 대화하는 경우 일반 오류 코드 인 500이 표시 될 수 있습니다. `http status cats`로 추가적인 내용을 확인하세요.

You can automatically throw a warning or raise an error if a request did not succeed.

```
warn_for_status(r)
stop_for_status(r)
```

## body

`r`을 만들고 나면 객체로써 내용에 접근하는 방법이 필요합니다. `content()`는 응답 객체에서 body부분만 보여주는 함수로 옵션으로 `text`, `raw`, `parsed`가 있습니다.

`content(r, "text")` 는 내용을 모두 텍스트로 인지하고 보여줍니다.

```
r <- GET("http://httpbin.org/get")
content(r, "text")
```

```
## No encoding supplied: defaulting to UTF-8.
```

```
## [1] "{\n  \"args\": {},\n  \"headers\": {\n    \"Accept\": \"application/json, text/xml, application/xml, */*\",\n    \"Accept-Encoding\": \"gzip, deflate\"\n  }\n}
```

httr은 인코딩을 사용하여 서버의 콘텐츠를 자동으로 해독합니다. content-type라는 정보가 http 헤더에 제공됩니다. 불행히도 항상 그런 것은 아닙니다.

```
content(r, "text", encoding = "euc-kr")
```

올바른 인코딩이 무엇인지 알아내는 데 문제가 있는 경우 stringi :: stri\_enc\_detect (content (r, "raw"))를 사용해볼 수 있습니다.

비 텍스트 요청의 경우(대부분 이미지) 원시 벡터로 요청 본문에 액세스 할 수 있습니다.

```
content(r, "raw")
```

```
## [1] 7b 0a 20 20 22 61 72 67 73 22 3a 20 7b 7d 2c 20 0a 20 20 22 68 65 61
## [24] 64 65 72 73 22 3a 20 7b 0a 20 20 20 22 41 63 63 65 70 74 22 3a 20
## [47] 22 61 70 70 6c 69 63 61 74 69 6f 6e 2f 6a 73 6f 6e 2c 20 74 65 78 74
## [70] 2f 78 6d 6c 2c 20 61 70 70 6c 69 63 61 74 69 6f 6e 2f 78 6d 6c 2c 20
## [93] 2a 2f 2a 22 2c 20 0a 20 20 20 22 41 63 63 65 70 74 2d 45 6e 63 6f
## [116] 64 69 6e 67 22 3a 20 22 67 7a 69 70 2c 20 64 65 66 6c 61 74 65 22 2c
## [139] 20 0a 20 20 20 20 22 43 6f 6e 6e 65 63 74 69 6f 6e 22 3a 20 22 63 6c
## [162] 6f 73 65 22 2c 20 0a 20 20 20 22 48 6f 73 74 22 3a 20 22 68 74 74
## [185] 70 62 69 6e 2e 6f 72 67 22 2c 20 0a 20 20 20 22 55 73 65 72 2d 41
## [208] 67 65 6e 74 22 3a 20 22 6c 69 62 63 75 72 6c 2f 37 2e 35 33 2e 31 20
## [231] 72 2d 63 75 72 6c 2f 32 2e 36 20 68 74 74 72 2f 31 2e 32 2e 31 22 0a
## [254] 20 20 7d 2c 20 0a 20 20 22 6f 72 69 67 69 6e 22 3a 20 22 32 32 31 2e
## [277] 31 34 38 2e 31 38 30 2e 33 35 22 2c 20 0a 20 20 22 75 72 6c 22 3a 20
## [300] 22 68 74 74 70 3a 2f 2f 68 74 74 70 62 69 6e 2e 6f 72 67 2f 67 65 74
## [323] 22 0a 7d 0a
```

이것은 정확히 웹 서버가 보낸 바이트 순서이므로 그대로 저장하면 서버에서 받은 것과 정확히 같습니다.

```
bin <- content(r, "raw")
writeBin(bin, "myfile.txt")
```

httr은 일반적인 파일 형식에 대한 여러 가지 기본 파서를 제공합니다.

```
str(content(r, "parsed"))
```

```
## List of 4
## $ args  : Named list()
## $ headers:List of 5
##   ..$ Accept      : chr "application/json, text/xml, application/xml, */*"
##   ..$ Accept-Encoding: chr "gzip, deflate"
##   ..$ Connection   : chr "close"
##   ..$ Host        : chr "httpbin.org"
##   ..$ User-Agent   : chr "libcurl/7.53.1 r-curl/2.6 httr/1.2.1"
## $ origin : chr "221.148.180.35"
## $ url    : chr "http://httpbin.org/get"
```

## rvest 패키지 소개

rvest는 html 문서에서 필요한 정보만 가져오기 위해 설계되었습니다. html 문서는 각 node의 성격을 정의하는 tag와 id, class를 가지고 있습니다. 이 세 가지로 정보가 필요한 node를 특정할 수 있고, rvest는 특정된 nodes의 정보를 가져올 다양한 방법을 제공합니다.

## chrome 개발자 도구

chrome은 구글이 만든 브라우저로 웹 페이지의 구조를 분석하는데 매우 유용한 기능들을 제공합니다. windows에서는 F12로 개발자 도구를 사용할 수 있습니다. mac은 option(alt) + cmd + i입니다.

여러 기능 중에 필요한 것은 Elements와 Network입니다.

웹 페이지에서 node의 정보가 필요한 요소를 마우스 우클릭을 하고 검사를 확인해보면 Elements 창이 열리면서 요소에 해당하는 html 문서를 하이라이트해서 보여줍니다. 그곳 node의 tag, id, class를 확인하여 rvest로 특정 부분만 가져와 보겠습니다.

# imdb에서 배우 정보 가져오기

imdb는 해외 영화정보 사이트로 html로 구성되어 있어 크롤링 연습에 많이 사용하는 대표적인 곳입니다.

rvest를 사용해서 설국열차에 참석한 사람들의 정보를 가져와 보겠습니다.

```
library(rvest)

## Loading required package: xml2

## 
## Attaching package: 'rvest'

## The following object is masked from 'package:readr':
## 
##     guess_encoding
```

```
html <- read_html("http://www.imdb.com/title/tt4481854/")
cast <- html_nodes(html, "#titleCast .itemprop")
html_text(cast)
```

```
## [1] "Wn Fershid Bharucha"
## [2] "Fershid Bharucha"
## [3] "Wn Joon-ho Bong"
## [4] "Joon-ho Bong"
## [5] "Wn Couetsch Bousset-Lob"
## [6] "Couetsch Bousset-Lob"
## [7] "Wn Benjamin Legrand"
## [8] "Benjamin Legrand"
## [9] "Wn Jacques Lob"
## [10] "Jacques Lob"
## [11] "Wn Clark Middleton"
## [12] "Clark Middleton"
## [13] "Wn Ondrej Nekvasil"
## [14] "Ondrej Nekvasil"
## [15] "Wn Chan-wook Park"
## [16] "Chan-wook Park"
## [17] "Wn Jean-Marc Rochette"
## [18] "Jean-Marc Rochette"
## [19] "Wn Albert Spano"
## [20] "Albert Spano"
```

```
## [21] "Wn Tilda SwintonWn"
## [22] "Tilda Swinton"
```

tag와 id, class를 세세하게 지정하는 것이 자료를 처리하기 쉽게 가져오는 요령입니다.

```
cast <- html_nodes(html, "#titleCast span[itemprop]")
length(cast)
```

```
## [1] 11
```

```
html_text(cast)
```

```
## [1] "Fershid Bharucha"      "Joon-ho Bong"          "Couetsch Bousset-Lob"
## [4] "Benjamin Legrand"        "Jacques Lob"           "Clark Middleton"
## [7] "Ondrej Nekvasil"         "Chan-wook Park"       "Jean-Marc Rochette"
## [10] "Albert Spano"           "Tilda Swinton"
```

tidyverse의 파이프 연산자를 사용해서 가져오는 것이 더 가독성과 재사용성이 좋습니다.

```
url <- "http://www.imdb.com/title/tt1490017/"

read_html(url) %>%
  html_nodes("#titleCast span[itemprop]") %>%
  html_text
```

```
## [1] "Will Arnett"      "Elizabeth Banks" "Craig Berry"
## [4] "Alison Brie"       "David Burrows"   "Anthony Daniels"
## [7] "Charlie Day"       "Amanda Farinos" "Keith Ferguson"
## [10] "Will Ferrell"      "Will Forte"     "Dave Franco"
## [13] "Morgan Freeman"    "Todd Hansen"   "Jonah Hill"
```

# 형태소 분석기 KoNLP

한글 형태소 분석기인 KoNLP는 [한나눔 형태소 분석기](#)를 R에서 사용할 수 있게 작성한 패키지로 R에서 형태소 분석을 할 때 거의 유일한 선택지입니다. 덕분에 명사 추출등을 이용한 워드클라우드 샘플 코드가 공개되면서 많은 사람들이 다양한 워드 클라우드를 만들 수 있게 되었습니다. R에서는 [wordcloud](#)와 다양한 기능이 개선된 [wordcloud2](#)를 사용해서 쉽게 워드클라우드를 만들수 있습니다.

## 함수 설명

```
library(KoNLP)
```

```
## Warning: package 'KoNLP' was built under R version 3.4.1
```

```
## Checking user defined dictionary!
```

```
useSejongDic()
```

```
## Backup was just finished!
## 370957 words dictionary was built.
```

명사를 추출합니다.

```
extractNoun("롯데마트가 판매하고 있는 흑마늘 양념 치킨이 논란이 되고 있다.")
```

```
## [1] "롯데마트" "판매"    "흑마늘"   "양념"    "치킨"    "논란"
```

벡터를 입력으로 받을 수 있습니다. sapply 같은 함수를 함께 사용하지 않아도 됩니다.

```
extractNoun(c("R은 free 소프트웨어이고, [완전하게 무보증]입니다.",
           "일정한 조건에 따르면, 자유롭게 이것을 재배포할수가 있습니다."))
)
```

```
## [[1]]
## [1] "R"      "free"     "소프트웨어" "완전"      "하게"
## [6] "무보"    "증"
##
## [[2]]
## [1] "일정"   "한"       "조건"      "자유"
## [5] "이것"    "재배포할수가"
```

형태소 태깅을 2단계 수준으로 조정해서 할 수 있습니다.

```
SimplePos09("롯데마트가 판매하고 있는 흑마늘 양념 치킨이 논란이 되고 있다.")
```

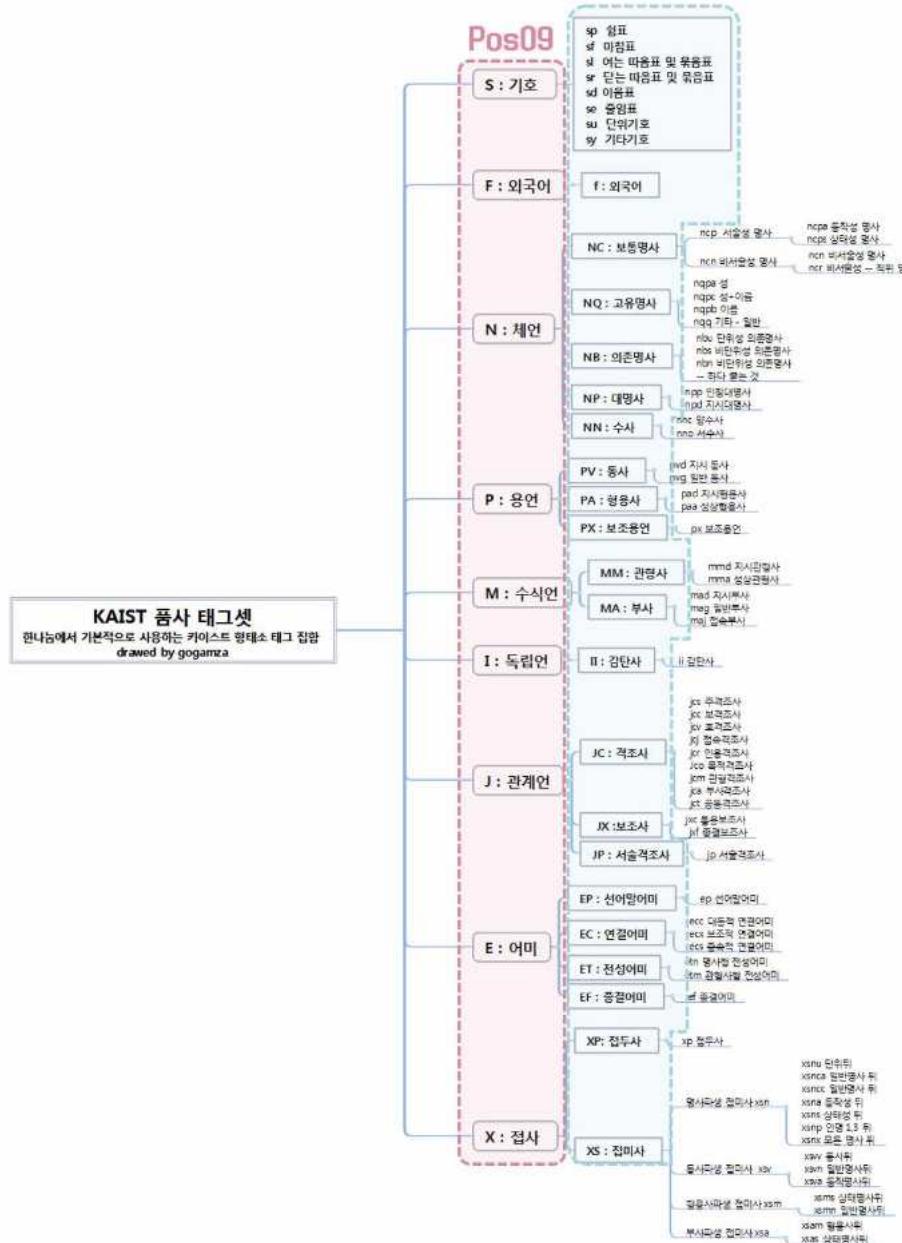
```
## $롯데마트가
## [1] "롯데마트/N+가/J"
##
## $판매하고
## [1] "판매/N+하고/J"
##
## $있는
## [1] "있/P+는/E"
##
## $흑마늘
## [1] "흑마늘/N"
##
## $양념
## [1] "양념/N"
##
## $치킨이
## [1] "치킨/N+이/J"
##
## $논란이
## [1] "논란/N+이/J"
##
## $되고
## [1] "되/P+고/E"
##
## $있다
## [1] "있/P+다/E"
##
## $.
## [1] "./S"
```

```
SimplePos22("롯데마트가 판매하고 있는 흑마늘 양념 치킨이 논란이 되고 있다.")
```

```
## $롯데마트가
## [1] "롯데마트/NC+가/JC"
##
## $판매하고
## [1] "판매/NC+하고/JC"
##
## $있는
## [1] "있/PX+는/ET"
##
## $흑마늘
## [1] "흑마늘/NC"
##
## $양념
## [1] "양념/NC"
##
## $치킨이
## [1] "치킨/NC+OI/JC"
##
## $논란이
## [1] "논란/NC+OI/JC"
##
## $되고
## [1] "되/PV+고/EC"
##
## $있다
## [1] "있/PX+다/EF"
##
## $.
## [1] "./SF"
```

## 품사 태그(KAIST 방식)

## Pos22



몇 가지 방식 있는데, KoNLP는 KAIST 방식을 사용합니다. 엔진을 하나눔을 사용함으로써 종속적이 된 것으로 보입니다.

가능성 있는 모든 결과를 보여줍니다.

```
MorphAnalyzer("롯데마트가 판매하고 있는 흑마늘 양념 치킨이 논란이 되고 있다.")
```

```
## $롯데마트가
## [1] "롯데마트/ncn+가/jcc" "롯데마트/ncn+가/jcs"
##
## $판매하고
## [1] "판매/ncpa+하고/jcj"           "판매/ncpa+하고/jct"
## [3] "판매/ncpa+하/xsvat+고/ecc"    "판매/ncpa+하/xsvat+고/ecs"
## [5] "판매/ncpa+하/xsvat+고/ecx"    "판매/ncpa+하/xsvat+어/ef+고/jcr"
##
## $있는
## [1] "있/paa+는/etm" "있/px+는/etm"
##
## $흑마늘
## [1] "흑마늘/ncn" "흑마늘/nqq"
##
## $양념
## [1] "양념/ncn"
##
## $치킨이
## [1] "치킨/ncn+01/jcc" "치킨/ncn+01/jcs" "치킨/ncn+01/ncn"
##
## $논란이
## [1] "논란/ncpa+01/jcc" "논란/ncpa+01/jcs" "논란/ncpa+01/ncn"
##
## $되고
## [1] "되/nbu+고/jcj"      "되/nbu+01/jp+고/ecc" "되/nbu+01/jp+고/ecs"
## [4] "되/nbu+01/jp+고/ecx" "되/paa+고/ecc"      "되/paa+고/ecs"
## [7] "되/paa+고/ecx"       "되/pvg+고/ecc"      "되/pvg+고/ecs"
## [10] "되/pvg+고/ecx"      "되/px+고/ecc"      "되/px+고/ecs"
## [13] "되/px+고/ecx"
##
## $있다
## [1] "있/paa+다/ef" "있/px+다/ef"
##
## $.
## [1] "./sf" "./sy"
```

자모 단위로 잘라서 보여줍니다.

```
convertHangulStringToJamos("R는 많은 공현자에의한 공동 프로젝트입니다")
```

```
## [1] "R"      "ㄴㅡㄴ"   "ㅁㅏㄴ"   "ㅇㅡㄴ"   "ㄱㅡㅇ"
## [8] "ㅎㅏㄴ"   "ㅈㅏ"    "ㅇㅕ"    "ㅇㅓ"    "ㅎㅏㄴ"   "ㄱㅡㅇ"
## [15] "ㄷㅗㅇ"   "ㅍㅡ"    "ㄹㅗ"    "ㅈㅔㄱ"   "ㅌㅡ"    "ㅇㅣㅂ"
## [22] "ㄴㅣ"    "ㄷㅏ"
```

키보드 매칭 기준으로 영타로 바꿔줍니다.

```
convertHangulStringToKeyStrokes("R는 많은 공현자에의한 공동 프로젝트입니다")
```

```
## [1] "R"      "s m s"   " "      "a k s g"   "d m s"   " "
## [7] "r h d"   "g j s"   "w k"   "d p"     "d m l"   "g k s"
## [13] " "       "r h d"   "e h d"  " "       "v m"    "f h"
## [19] "w p r"   "x m"    "d l q"   "s l"    "e k"
```

```
convertHangulStringToKeyStrokes("R는 많은 공현자에의한 공동 프로젝트입니다", isFullwidth = F)
```

```
## [1] "R"      "sms"    " "      "aksg"    "dms"    " "
## [11] "dml"   "gks"    "rhd"   "ehd"    "vm"    "fh"
## [21] "dlq"   "sl"    "ek"
```

자모 단위로 잘라져 있는 결과를 다시 합칩니다.

```
str <- convertHangulStringToJamos("배포 조건의 상세한것에 대해서는 'license()' 또는 'licence()' 라고 입력해주십시오")
(str2<-paste(str, collapse=""))
```

```
## [1] "ㅂㅐㅍㅗ ㅈㅗㄱㅓㄴㅇㅓ ㅅㅏㅇㅅㅔㅎㅏㄴㄱㅓㅅㅇㅓ ㄷㅐㅎㅓㅅㅓㄴㅡㄴ 'license()' ㄸㅏㄴㅡㄴ 'licence()' ㅋㅏㄱㅓ ㅇㅣㅂㄹㅋㄱㅎㅐㅈㅓㅅ | ㅂㅅ | ㅇㅗ"
```

```
HangulAutomata(str2)
```

```
## [1] "배포 조건의 상세한것에 대해서는 'license()' 또는 'licence()' 라고 입력해주십시오"
```

최근 NIADic을 공개하는 것을 KoNLP 제작자에게 의뢰함으로써 사전의 공개가 R에서 우선적으로 이루어졌습니다. 기존 세종 사전의 9만 단어에서 약 98만 단어로 사전이 확보되어 성능의 개선을 기대할 수 있습니다.

```
useSystemDic()
```

```
## Backup was just finished!
## 283949 words dictionary was built.
```

```
extractNoun("성긴털제비꽃은 너무 예쁘다.")
```

```
## [1] "성긴털제비꽃은"
```

```
SimplePos09("성긴털제비꽃은 너무 예쁘다.")
```

```
## $성긴털제비꽃은
## [1] "성긴털제비꽃은/N"
##
## $너무
## [1] "너무/M"
##
## $예쁘다
## [1] "예쁘/P+다/E"
##
## $.
## [1] "./S"
```

```
SimplePos22("성긴털제비꽃은 너무 예쁘다.")
```

```
## $성긴털제비꽃은
## [1] "성긴털제비꽃은/NC"
##
## $너무
## [1] "너무/MA"
##
## $예쁘다
## [1] "예쁘/PA+다/EF"
##
## $.
## [1] "./SF"
```

```
MorphAnalyzer("성긴털제비꽃은 너무 예쁘다.")
```

```
## $성긴털제비꽃은
## [1] "성긴털제비꽃/ncn+은/j xc" "성긴털제비꽃은/ncn"
## [3] "성긴털제비꽃/nqa+은/j xc" "성긴털제비꽃은/nqa"
##
## $너무
## [1] "너무/mag"
##
## $예쁘다
## [1] "예쁘/paa+다/ef"
##
## $.
## [1] "./sf" "./sy"
```

```
useSejongDic()
```

```
## Backup was just finished!
## 370957 words dictionary was built.
```

```
extractNoun("성긴털제비꽃은 너무 예쁘다.")
```

```
## [1] "성긴털제비꽃"
```

```
SimplePos09("성긴털제비꽃은 너무 예쁘다.")
```

```
## $성긴털제비꽃은
## [1] "성긴털제비꽃/N+은/J"
##
## $너무
## [1] "너무/M"
##
## $예쁘다
## [1] "예쁘/P+다/E"
##
## $.
## [1] "./S"
```

```
SimplePos22("성긴털제비꽃은 너무 예쁘다.")
```

```
## $성긴털제비꽃은
## [1] "성긴털제비꽃/NC+은/JX"
##
```

```
## $너무
## [1] "너무/MA"
##
## $예쁘다
## [1] "예쁘/PA+다/EF"
##
## $.
## [1] "./SF"
```

```
MorphAnalyzer("성긴털제비꽃은 너무 예쁘다.")
```

```
## $성긴털제비꽃은
## [1] "성긴털제비꽃/ncnt+은/jxc" "성긴털제비꽃/ncnt+은/ncn"
##
## $너무
## [1] "너무/mag"
##
## $예쁘다
## [1] "예쁘/paa+다/ef"
##
## $.
## [1] "./sf" "./sy"
```

```
useNIADic()
```

```
## Backup was just finished!
## 983012 words dictionary was built.
```

```
extractNoun("성긴털제비꽃은 너무 예쁘다.")
```

```
## [1] "성긴털" "제비꽃"
```

```
SimplePos09("성긴털제비꽃은 너무 예쁘다.")
```

```
## $성긴털제비꽃은
## [1] "성긴털제비꽃/N+은/J"
##
## $너무
## [1] "너무/M"
##
## $예쁘다
```

```
## [1] "예쁘/P+다/E"
##
## $.
## [1] "./S"
```

```
SimplePos22("성긴털제비꽃은 너무 예쁘다.")
```

```
## $성긴털제비꽃은
## [1] "성긴털제비꽃/NC+은/JX"
##
## $너무
## [1] "너무/MA"
##
## $예쁘다
## [1] "예쁘/PA+다/EF"
##
## $.
## [1] "./SF"
```

```
MorphAnalyzer("성긴털제비꽃은 너무 예쁘다.")
```

```
## $성긴털제비꽃은
## [1] "성긴털/ncn+제비/꽃/ncn+은/jxc"      "성긴털/ncn+제비/꽃/ncn+은/ncn"
## [3] "성긴털/ncn+제비/꽃/ncn+은/jcs"        "성긴털/ncn+제비/ncn+꽃/ncn+은/jxc"
## [5] "성긴털/ncn+제비/ncn+꽃/ncn+은/jcs"
##
## $너무
## [1] "너무/mag"
##
## $예쁘다
## [1] "예쁘/paa+다/ef"
##
## $.
## [1] "./sf" "./sy"
```

## java 문제 해결

아래는 제작자가 작성한 java 설치에 관련된 문제 해결법을 첨부하였습니다.

KoNLP는 한글 처리에 대한 전문적인 지식이 없는 사회학, 경영, 경제학 등의 연구자들 그리고 일반인들이 R 기반으로 통계적 텍스트 분석을 수행하기 위한 편의를 제공하는데 목적을 두고 있다. 최대한 전문적인 경험이 필요

없게 구성을 하고 있으나 몇가지 사용자들이 어려움을 겪는 공통적인 부분이 있다는 것을 이곳에 정리하여 시행착오를 출이고자 한다. 물론 이런 부분을 패키지에 넣으면 되지 않을까 생각하시는 분들이 있을 수 있으나 대부분 이곳에 언급하는 어려움들은 패키지 레벨에서 해결하기 어려운 환경적인 이슈와 엮여 있다는 것을 미리 언급해 둔다. 어떤 사용자분들은 이런 경험을 하나도 하지 못할 경우도 있고, 어떤 분들은 대부분 경험해 봤을 수 있는 이슈들이라 생각되는 것을 정리하고자 한다.

## 윈도우 rJava loading error

R만으로 한글에 대한 처리를 하는건 매우 어렵다. 텍스트 처리 관련 함수도 적고, 여러 인코딩 이슈도 발생한다. 무엇보다 처리 속도 이슈가 있는데, 이를 위해 Java와 Scala 언어를 기본 처리 언어로 사용하고 있다. 따라서 대부분의 R함수들은 Java와 Scala 기반의 함수를 호출하기 위한 Wrapper 역할을 수행한다. 따라서 KoNLP가 수행되기 위한 자바 환경이 구축되는 것은 정상적으로 KoNLP를 사용하기 위한 필수 조건이다. 그리고 KoNLP가 의존성을 가지고 있는 rJava 패키지는 R에서 존재하는 수많은 자바 기반의 패키지들을 구동하는데 매우 필수적으로 사용되는 패키지이다. 따라서 적절히 환경을 설정해 놓는다면 앞으로 R을 사용하는데 많은 도움이 될 것이다.

rJava 로딩 에러는 대부분 윈도우 기반의 환경에서 발생한다. 가장 중요한 부분은 사용자가 설치한 R의 버전과 Java의 버전이 맞는지 확인하는 것이다.

```
Sys.getenv("R_ARCH")
```

```
## [1] "/x64"
```

위 명령어로 확인해보면 R이 32비트 기반인지 64비트 기반인지 출력된다. 만일 /x64로 출력되면 64비트의 R이 설치된 것이다. 따라서 자바의 경우도 설치된 R의 아키텍처에 맞게 설치가 되어야 되는데, rJava로 문제가 생기는 대부분 이 부분이 맞지 않아서 생기는 문제이다.

자바 환경 설치는 [이곳](#)에서 자동으로 다운로드 가능하다. 여기서 다른 문제가 있는데, 바로 구동되는 시스템이 아닌 구동된 브라우저의 아키텍처에 따라 64비트 자바 혹은 32비트 자바가 자동으로 선택되어져 다운로드 된다는 것이다. 이 부분은 [여기](#)를 확인해보면 정확한 정보를 얻을 수 있을 것이다. 64비트 자바를 설치하기 위해서는

64비트 브라우저를 열어 설치 링크를 통해 설치해야 된다. 현재 자신의 시스템에 설치된 자바를 보고 싶다면 윈도우 제어판에서 Java 정보를 확인해 보는것도 도움이 될 것이다.

## 맥에서 rJava 문제

Mac + R + rJava의 조합은 맥에서 가장 어려운 조합으로 보여진다. 이는 맥에서 기본적으로 제공하고 있는 Java 1.6이 시스템의 자바 설정에 강제하고 있는 부분이 있는 것으로 판단되고 이를 R에서 해결하는게 매우 까다로운 것으로 알려져 있기 때문이다. 맥에서의 문제만 아니면 필자는 벌써 Java 1.7이나 1.8로 KoNLP를 바꾸었을 것이다. 지금껏 KoNLP에서 Java 1.6을 지원하는 가장 큰 원인은 맥에서의 문제 때문이다.

# word2vec을 R에서 해보자

word2vec은 딥러닝의 일종으로 한글은 [한글 word2vec 테스트](#)에서 확인해 볼 있습니다.

word2vec의 필요성과 역사 등은 김홍배님의 훌륭한 [자료](#)가 있어서 확인해보시면 좋을 것 같습니다. 정상근 박사님의 [자료](#)도 좋습니다.

## 알고리즘의 종류

빠른 성능으로 개선된 모델인 google의 [word2vec](#)을 시작으로 Stanford에 빌 발표한 [Glove](#), facebook이 발표한 [fastText](#) 등이 있습니다. 우리는 실습이 용이한 [Glove](#)를 사용해 보겠습니다.

```
library(text2vec)
```

```
##  
## Attaching package: 'text2vec'
```

```
## The following object is masked from 'package:dplyr':  
##  
##     collect
```

```
text8_file = "./text8"  
if (!file.exists(text8_file)) {  
  download.file("http://mattmahoney.net/dc/text8.zip", "./text8.zip")  
  unzip("./text8.zip", files = "text8", exdir = ".")  
}  
wiki = readLines(text8_file, n = 1, warn = FALSE)  
  
tokens <- space_tokenizer(wiki)  
  
it = itoken(tokens, progressbar = FALSE)  
vocab <- create_vocabulary(it)  
vocab <- prune_vocabulary(vocab, term_count_min = 5L)  
  
vectorizer <- vocab_vectorizer(vocab)  
  
tcm <- create_tcm(it, vectorizer)
```

```
glove = GlobalVectors$new(word_vectors_size = 50, vocabulary = vocab, x_max = 10)
word_vectors <- glove$fit_transform(tcm, n_iter = 20)
```

```
## INFO [2017-07-25 03:47:54] 2017-07-25 03:47:54 - epoch 1, expected cost 0.0877
## INFO [2017-07-25 03:48:21] 2017-07-25 03:48:21 - epoch 2, expected cost 0.0612
## INFO [2017-07-25 03:48:46] 2017-07-25 03:48:46 - epoch 3, expected cost 0.0541
## INFO [2017-07-25 03:49:11] 2017-07-25 03:49:11 - epoch 4, expected cost 0.0502
## INFO [2017-07-25 03:49:34] 2017-07-25 03:49:34 - epoch 5, expected cost 0.0476
## INFO [2017-07-25 03:50:04] 2017-07-25 03:50:04 - epoch 6, expected cost 0.0458
## INFO [2017-07-25 03:50:29] 2017-07-25 03:50:29 - epoch 7, expected cost 0.0444
## INFO [2017-07-25 03:50:54] 2017-07-25 03:50:54 - epoch 8, expected cost 0.0433
## INFO [2017-07-25 03:51:21] 2017-07-25 03:51:21 - epoch 9, expected cost 0.0425
## INFO [2017-07-25 03:51:46] 2017-07-25 03:51:46 - epoch 10, expected cost 0.0418
## INFO [2017-07-25 03:52:13] 2017-07-25 03:52:13 - epoch 11, expected cost 0.0411
## INFO [2017-07-25 03:52:37] 2017-07-25 03:52:37 - epoch 12, expected cost 0.0406
## INFO [2017-07-25 03:52:58] 2017-07-25 03:52:58 - epoch 13, expected cost 0.0402
## INFO [2017-07-25 03:53:25] 2017-07-25 03:53:25 - epoch 14, expected cost 0.0398
## INFO [2017-07-25 03:53:47] 2017-07-25 03:53:47 - epoch 15, expected cost 0.0394
## INFO [2017-07-25 03:54:14] 2017-07-25 03:54:14 - epoch 16, expected cost 0.0391
## INFO [2017-07-25 03:54:39] 2017-07-25 03:54:39 - epoch 17, expected cost 0.0388
## INFO [2017-07-25 03:55:06] 2017-07-25 03:55:06 - epoch 18, expected cost 0.0385
## INFO [2017-07-25 03:55:28] 2017-07-25 03:55:28 - epoch 19, expected cost 0.0383
## INFO [2017-07-25 03:55:51] 2017-07-25 03:55:51 - epoch 20, expected cost 0.0381
```

```
berlin <- word_vectors["paris", , drop = FALSE] -
  word_vectors["france", , drop = FALSE] +
  word_vectors["germany", , drop = FALSE]
cos_sim = sim2(x = word_vectors, y = berlin, method = "cosine", norm = "l2")
head(sort(cos_sim[,1], decreasing = TRUE), 5)
```

```
##    paris    vienna   germany   munich      rome
## 0.7529268 0.7190628 0.7048425 0.6914796 0.6611500
```