1. 시계열 데이터 처리

우리는 시간 단위로 기록된 데이터를 항상 다루어 왔다. 데이터의 월 평균, 데이터의 전월 대비 증감 등을 산출하기 위해서는 데이터를 월 단위로 구분하여 평균을 구하거나 전월 데이터를 산출하여 당월 데이터와의 차이를 구하는 등의 작업을 해야한다.

R에서 주로 사용하는 데이터프레임에서는 행렬 인덱스나 조건을 []를 이용하여 전달함으로써 데이터를 검색하고 일부 데이터를 부분 집합화(subsetting)하거나 dplyr 패키지에서 제공하는 select, filter, group\_by, arrange, mutate 등의 함수를 많이 사용하여 작업을 수행할 수 있다. 물론 데이터프레임에 저장된 시계열 데이터는 dplyr에서 제공하는 함수를 사용할 수 있지만 ts나 xts의 시계열 class의 객체에 대해서는 적용이 되지 않는다. 또한 시계열 데이터는 타 데이터에는 없는 시간적 특성(예를 들어 1분은 60초, 국가마다 년, 월을 표현하는 방식의 차이, Time Zone)이 있기 때문에 이를 잘 활용하기 위해서는 시계열 데이터의 핸들링 방법을 익혀둘 필요가 있다.

이 장에서는 시계열 데이터를 시간 인덱스 단위로 다루는 방법을 설명할 것이다. 데이터가 저장된 형태에 따라 데이터프레임 객체에 저장된 경우와 xts 객체에 저장된 경우의 처리 방법을 구분하여 설명한다.

시간을 주로 다루기 위해서 R에서 가장 많이 활용되는 패키지는 lubridate 패키지이다. 이 패키지는 시간을 다루기 위한 다양한 함수를 제공하는데 시계열 데이터를 잘 활용하기 위해 lubridate 패키지를 위주로 시간 인덱스를 다루는 방법에 대해 설명하고자 한다.

# 오늘 몇일이야? : 시간 정보 추출

시계열 데이터는 년, 월, 일, 시, 분, 초, 타임존 등의 여러 데이터가 결합되어 생성된 시간 인덱스를 사용하는 데이터이다. 그래서 시계열 데이터를 잘 활용하기 위해서는 시간 인덱스에 포함된 요소들을 목적에 맞게 적절히 추출해내야 활용이 가능해진다.

아래의 함수들은 lubridate 패키지에서 제공하는 시간 정보를 추출하는 함수들이다. 이 함수들을 적절히 사용하면 시간을 다루기가 매우 쉬워진다.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 함수명 | 설명 | 함수명 | 설명 | 함수명 | 설명 |
| year() | 연도 추출 | hour() | 시간 추출 | week() | 1월1일으로부터의 주수 |
| month() | 월 추출 | minute() | 분 추출 | quater(), semester() | 1월1일으로부터의 분기수, 반기수 |
| day() | 일 추출 | second() | 초 추출 | am(), pm() | 오전, 오후의 논리값 |
| wday() | 요일 추출 | yday() | 1월1일으로부터의 날수 | leap\_year() | 윤년 여부의 논리값 |

# lubridate 패키지 로딩  
 library(lubridate)  
 # 현재 시간을 now.date에 저장(date 클래스)  
 (now.date <- Sys.time())   
[1] "2021-03-01 13:58:15 KST"

# 현재 시간을 now.char에 저장(character 클래스)  
 (now.char <- as.character(Sys.time()))   
[1] "2021-03-01 13:58:15"

# year(), month(), day()를 시용하여 날짜를 표기하는 문자열 생성  
 paste0('오늘은 ', year(now.date), '년 ', month(now.char), '월 ', day(now.date), '일 입니다')  
[1] "오늘은 2021년 3월 1일 입니다"

# ydya()를 사용하여 1월 1일부터 오늘까지의 날짜수 계산  
 paste0('1월 1일부터 오늘까지 ', yday(now.date), '일 지났습니다')  
[1] "1월 1일부터 오늘까지 60일 지났습니다"

# qday()를 사용하여 분기시작일로부터 오늘까지의 날짜수 계산  
 paste0('이번 분기 시작일부터 오늘까지 ', qday(now.date), '일 지났습니다')  
[1] "이번 분기 시작일부터 오늘까지 60일 지났습니다"

# wday()를 사용하여 오늘의 요일 표기  
 paste0('오늘은 ', wday(now.date, label = T, abbr = T), '요일입니다')  
[1] "오늘은 월요일입니다"

# hour(), minute(), second()를 사용하여 시간을 표시  
 paste0('지금은 ', hour(now.date), '시 ', minute(now.char), '분 ', second(now.date), '초 입니다')  
[1] "지금은 13시 58분 15.7485609054565초 입니다"

# week()를 사용하여 올해 몇번째 주인지 계산  
 paste0('이번주는 올해의 ', week(now.date), '번째 주입니다')  
[1] "이번주는 올해의 9번째 주입니다"

# 며칠 지났어? : 시간 기간 연산

시간의 연산은 일반적으로 오늘로부터 며칠 후, 언제로부터 몇 개월 전 등과 같은 연산을 말한다. lubridate 패키지는 시간 연산에서 사용되는 유용한 함수들을 많이 제공한다.

* 시간 더하기/빼기

lubridate 패키지에서는 시간의 더하기나 빼기 연산을 지원한다. 일반적인 정수의 연산과 같이 더하거나 빼면 기준일로부터 과거나 미래의 시간을 구할 수 있고 앞에서 설명한 lubridate의 years(), months(), days() 등의 함수를 이용하면 년 단위, 월 단위, 일 단위의 연산도 가능하다.

# 1980년 1월 1일부터 2021년 1월 1일까지의 날짜 수  
 as.Date('2021-01-01') - as.Date('1980-01-01')   
Time difference of 14976 days

# 오늘 날짜를 today에 저장  
 today <- today()

# 오늘부터 100일 후  
 today + 100   
[1] "2021-06-09"

# 오늘부터 2개월 전   
 today - months(2)   
[1] "2021-01-01"

# 오늘부터 1년전  
 today - years(1)   
[1] "2020-03-01"

* 경과 기간 구하기

앞선 예에서 1980년 1월 1일부터 2021년 1월 1일까지는 14,976일로 계산되었다. 그러나 우리는 보통 몇 년 몇 월 몇 일로 표현하여 사용한다. lubridate패키지는 이렇게 몇 년 몇 월 몇 일로 변환하는 것을 위해 interval()를 제공한다. 하지만 바로 계산되는 것은 아니다. interval()에 의해 반환되는 값은interval 클래스의 데이터값으로 시작점으로부터의 초를 기록한다. 이렇게 변환된 interval 객체를 우리가 보기 편하게 년, 월, 일 등으로 표시하기 위해서는 interval 객체를 as.period() 나 as.duration()으로 period나 duration 클래스 객체로 변환하여야 한다.

interval()는 %–% 연산자로 사용할 수도 있다.

# 1980.1.1부터 2021.12.31까지의 interval 클래스를 int에 저장  
 # 결과값을 보면 우리가 생각하는 형태가 아님  
 (int <- lubridate::interval(as.Date('1980-01-01'), as.Date('2021-12-31')))  
[1] 1980-01-01 UTC--2021-12-31 UTC

# 년월일 형태로 interval 출력  
 lubridate::as.period(int)  
[1] "41y 11m 30d 0H 0M 0S"

# 경과 초 형태로 interval 출력  
 lubridate::as.duration(int)   
[1] "1325376000s (~42 years)"

# 1980.1.1부터 2021.12.31까지의 interval 클래스를 int1에 저장  
 int1 <- '1980-01-01' %--% '2021-12-31'  
 # 년월일 형태로 interval 출력  
 lubridate::as.period(int1)   
[1] "41y 11m 30d 0H 0M 0S"

위의 예제와 같이 internal은 시간 범위(time span)를 표현하는데 lubridate패키지에서는 period와 duration의 두가지 시간 범위(time span) 클래스를 제공한다.

lubridate 패키지에서 제공하는 period함수는 복수 형태(years(), months() 등)로 표현된 함수를 사용하고 duration 함수는 함수의 접두어를 ’d’를 사용하여 표현한다.

두 가지 형태의 internal을 제공하는 이유는 시간 경과의 표현과 기간 경과의 표현이 다르기 때문이다. duration은 항상 수학적으로 동일한 시간 경과의 결과를 제공한다. 하지만 period는 우리가 흔히 연월일로 생각하는 기간 경과의 결과를 제공한다. 예를 들어 duration의 1년 후는 시작일로부터 365일 후가 되는 날짜를 반환하지만 period의 1년 후는 연도가 1이 더해진 날짜를 반환한다. 두 가지의 결과가 동일할 것이라고 생각되지만 윤년이 낀 경우는 duration의 결과는 period의 결과와 달라진다. 또 duration 함수에 정수 시퀀스를 넣어주면 해당 연산에 의한 vector값이 얻어진다. [[1]](#footnote-1)

# 2020년은 윤년  
 leap\_year(2020)   
[1] TRUE

# 2020-01-01부터 기간상 1년 후(period)는 우리의 상식대로 2021-01-01  
 as.Date('2020-01-01') + years(1)   
[1] "2021-01-01"

# 2020-01-01부터 시간상 1년 후(duration)는 2020년은 윤년이므로 2020년은 366일임. 그래서 365일 후인 2020-12-31이 표기됨  
 as.Date('2020-01-01') + dyears(1)   
[1] "2020-12-31 06:00:00 UTC"

# 2020-02-01부터 한달후(period)는 21년 3월 1일  
 as.Date('2020-02-01') + months(1)   
[1] "2020-03-01"

# 2020-02-01부터 한달후(duration)는 30일 후 인 20년 3월 2일  
 as.Date('2020-02-01') + dmonths(1)   
[1] "2020-03-02 10:30:00 UTC"

# 2021-02-01부터 한달후(period)는 21년 3월 1일  
 as.Date('2021-02-01') + months(1)   
[1] "2021-03-01"

# 2020-01-01부터 한달 후(duration)는 30일 후 인 3월 2일  
 as.Date('2021-02-01') + dmonths(1)   
[1] "2021-03-03 10:30:00 UTC"

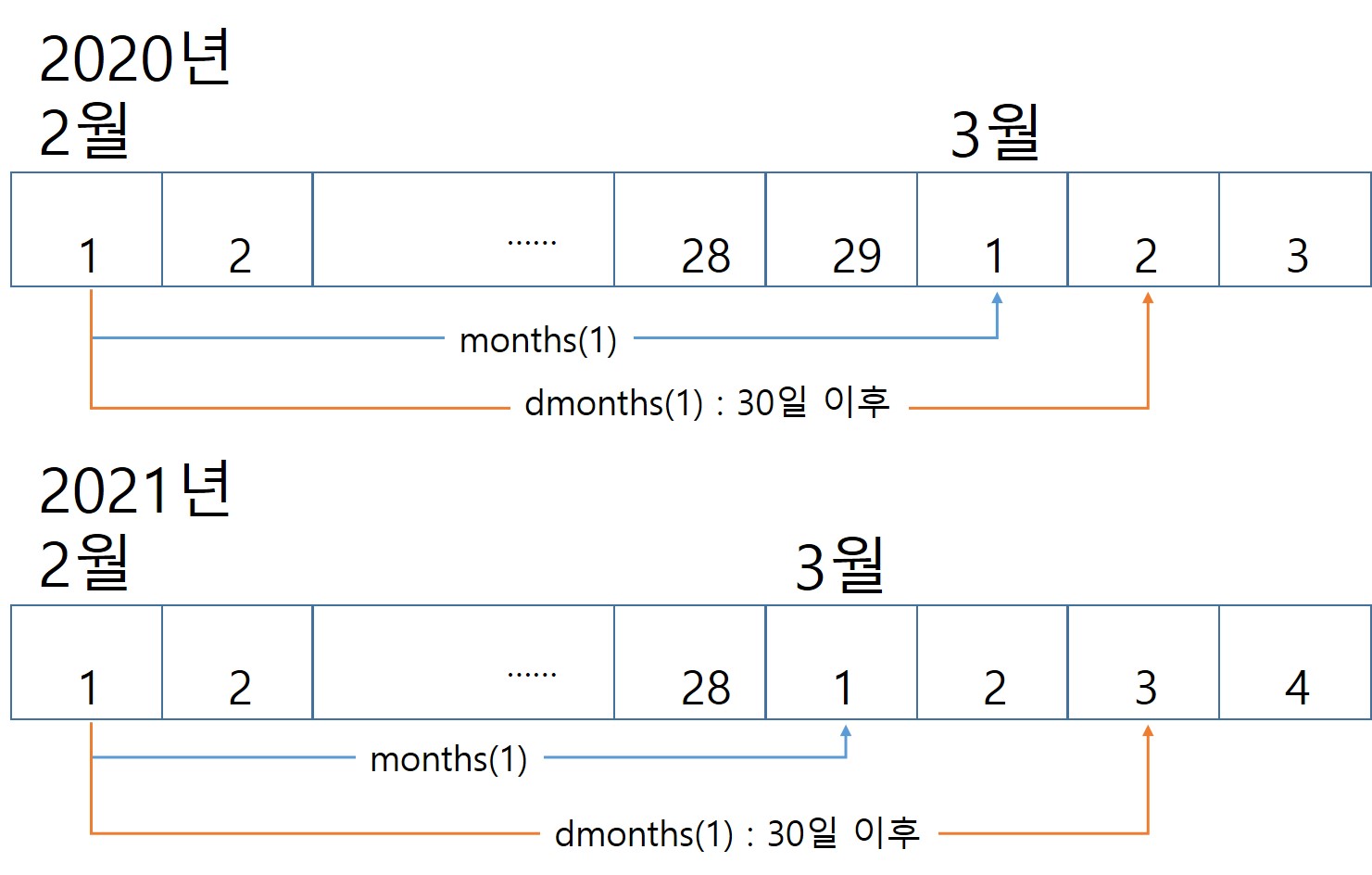


그림 4-1

# 이번 주 마지막 날은 며칠이야? : 시간 반올림

시간의 계산에서도 숫자의 계산과 같이 올림, 내림, 반올림 등이 가능하다. 숫자의 계산에서는 소수점 셋째 자리, 천 단위 반올림, 올림 등과 같이 반올림될 단위를 지정해야 하는데 시간의 올림, 반올림 연산에도 월에서 반올림을 할 것인지 일에서 반올림을 할 것인지와 같은 단위를 지정해야 한다. [[2]](#footnote-2)

lubridate 패키지에서는 시간 데이터의 반올림을 위해 round\_date(), 내림을 위해 floor\_date(), 올림을 위해 ceiling\_date() 함수를 제공한다. 매개변수로 ‘day’, ‘week’, ‘month’, ‘halfyear’, ‘year’ 등을 설정하여 반올림, 올림, 내림의 기본 단위를 지정할 수 있다.

(x <- as.Date("2020-11-12 13:45:40"))  
[1] "2020-11-12"

# 주단위로 반올림  
 round\_date(x, "week")   
[1] "2020-11-15"

# 주단위로 내림  
 floor\_date(x, "week")   
[1] "2020-11-08"

# 주단위로 올림  
 ceiling\_date(x, "week")   
[1] "2020-11-15"

# 월단위로 반올림  
 round\_date(x, "month")   
[1] "2020-11-01"

# 월단위로 내림  
 floor\_date(x, "month")   
[1] "2020-11-01"

# 월단위로 올림  
 ceiling\_date(x, "month")   
[1] "2020-12-01"

# 년단위로 반올림  
 round\_date(x, "year")   
[1] "2021-01-01"

# 년단위로 내림  
 floor\_date(x, "year")   
[1] "2020-01-01"

# 년단위로 올림  
 ceiling\_date(x, "year")   
[1] "2021-01-01"

# 말일을 구하는 코드  
 days\_in\_month(as.Date('2012-2', format = '%Y-%d'))   
Mar   
 31

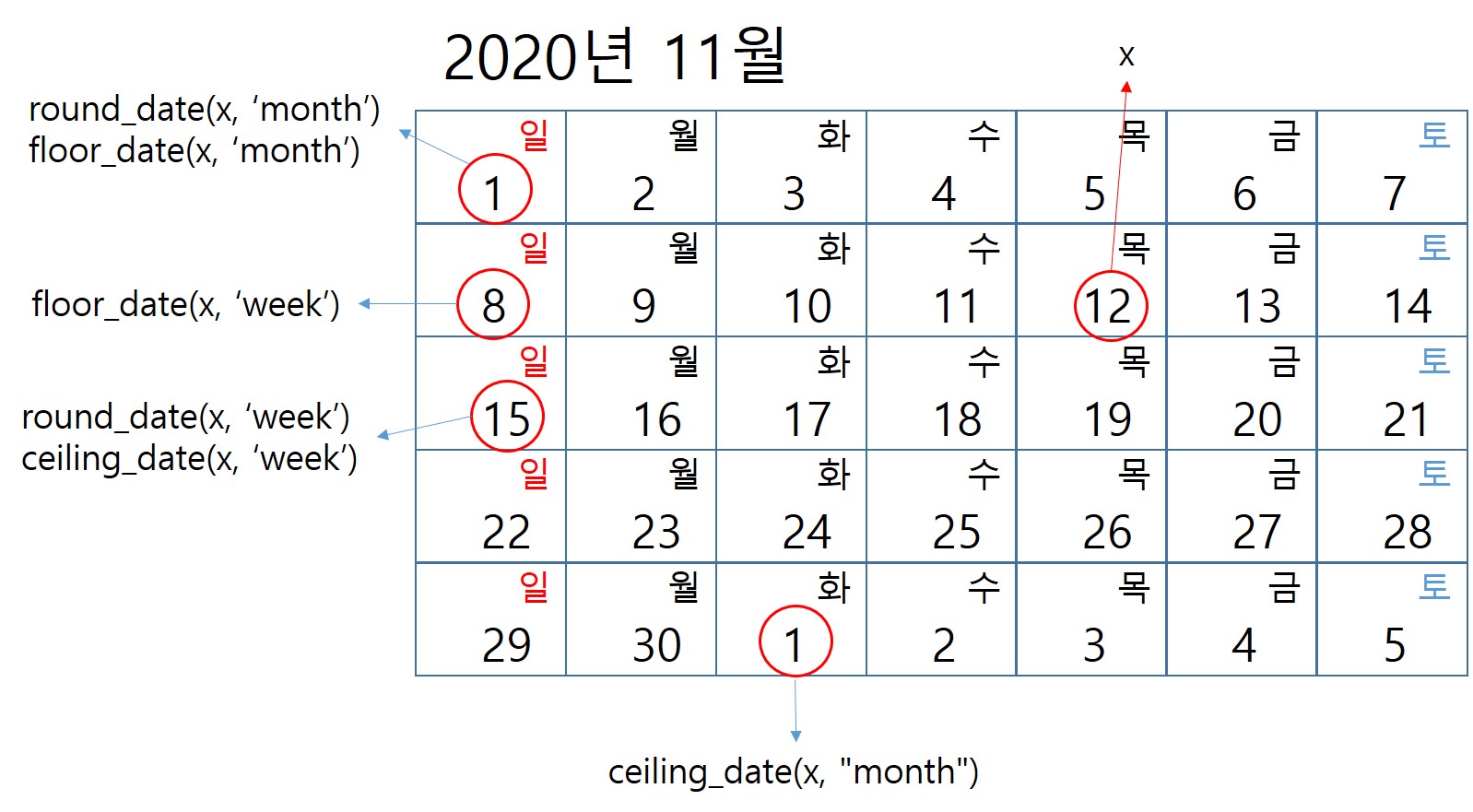


그림 4-2

# 주간, 월간 데이터 합계, 평균은? : 시간 Grouping

시계열 데이터를 사용할 때 많이 사용하는 계산은 월별 평균, 주별 합계 등 일정 단위별로 그룹핑한 후 그룹 함수를 적용하여 산출하는 작업이다.

아쉽게도 앞서 사용한 lubridate 패키지에서는 시계열 데이터를 주 단위, 월 단위, 년 단위 등으로 그룹핑하는 함수를 제공하지 않는다. 따라서 lubridate 함수를 이용하여 그룹화하기 위해서는 dplyr에서 제공하는 파이프 연산자인 %>%와 앞에 설명한 시간 정보 추출 함수를 같이 사용하여 산출할 수 있다.

하지만 앞장에서 소개한 timetk 패키지에서는 그룹핑하여 평균을 내거나 합계를 내는 함수를 제공하기 때문에 이를 사용할 수도 있다.

이번 절에서는 lubridate와 dplyr를 사용하는 방법과 timetk를 사용하는 방법으로 분리해서 설명하겠다.

* 데이터프레임 : lubridate, dplyr 패키지

데이터프레임으로 저장된 시계열 데이터의 그룹핑을 위해서는 먼저 그룹핑 주기를 설정해야 한다. lubridate 패키지의 year(), month(), week() 등의 함수를 사용하여 그룹핑 주기를 설정할 수 있는데 이 결과를 mutate()를 사용하여 열로 만들어 준 다음, group\_by() 를 사용하여 데이터를 그룹핑하고 summarise()를 통해 sum(), mean() 등의 그룹 함수를 사용하여 원하는 값을 산출한다.

library(dplyr)  
 library(ggplot2)  
 # 월별 취업자수를 연별 취업자수로 그룹핑  
 (employees.by.year <- employees %>%  
 mutate(year = year(time)) %>%   
 group\_by(year) %>%  
 summarise(total.year = sum(total),  
 employees.edu = sum(employees.edu)))

# A tibble: 8 x 3  
 year total.year employees.edu  
 <dbl> <int> <int>  
1 2013 303592 21192  
2 2014 310766 21957  
3 2015 314133 22028  
4 2016 316910 22350  
5 2017 320698 22886  
6 2018 321866 22165  
7 2019 325474 22600  
8 2020 322852 21570

코드 설명

* employees 데이터프레임에서 mutate()를 사용하여 time에서 뽑은 연도(year())를 year 열에 저장
* group\_by()로 year 열로 그룹핑 한 후 summarise()를 사용하여 전체 취업자수의 합계를 total.year, 교육서비스업의 합계 employees.edu 열을 생성
* 최종 결과를 employees.by.year에 저장

employees.by.year %>%   
 ggplot(aes(as.factor(year), total.year)) +   
 geom\_line(aes(group = 1)) +  
 geom\_text(aes(label = scales::number(total.year, big.mark = ',')), size = 3, vjust = 1.5) +   
 labs(title = '연별 취업자수', x = '연도', y = '취업자수') +   
 scale\_y\_continuous(labels = scales::number\_format(big.mark = ','))

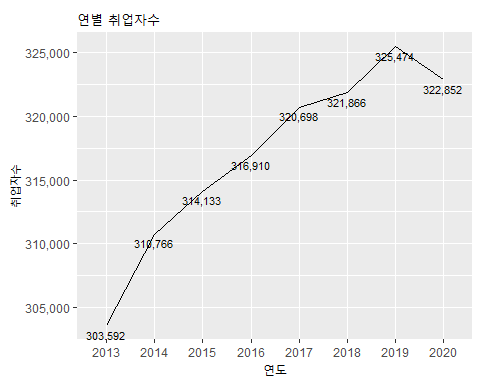


그림 4‑3

# 일별 평균 확진자수를 산출  
 (mean.covid19.by.age <- covid19 %>%  
 mutate(yearmon = yearmonth(date)) %>%  
 group\_by(yearmon) %>%  
 summarise(`01대` = mean(`0-9세`),   
 `10대` = mean(`10-19세`),   
 `20대` = mean(`20-29세`),   
 `30대` = mean(`30-39세`),   
 `40대` = mean(`40-49세`),   
 `50대` = mean(`50-59세`),   
 `60대` = mean(`60-69세`),   
 `70대` = mean(`70-79세`),   
 `80대` = mean(`80세 이상`)))

# A tibble: 11 x 10  
 yearmon `01대` `10대` `20대` `30대` `40대` `50대` `60대` `70대` `80대`  
 <mth> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>  
 1 2020 4 0.636 1.91 5.45 2.55 1.82 1.86 1.64 0.773 0.682  
 2 2020 5 0.533 2.07 7.13 4.13 3.03 2.4 1.8 0.5 0.4   
 3 2020 6 1.11 1.71 6.11 6.71 4.89 7.61 8.39 3.93 2   
 4 2020 7 1.71 2.39 8.32 10.1 8.03 7.55 5.97 3.13 1.35   
 5 2020 8 6.32 11.7 22.6 22 24.6 36.1 36.0 16.7 6.06   
 6 2020 9 4.53 5.37 14.7 13.9 16.2 26.1 27.1 14.7 6.17   
 7 2020 10 3.73 4.57 12.9 12.1 12.4 14.5 13.2 6.63 5.77   
 8 2020 11 10.0 19.8 43.3 36.5 41.1 44.9 34.4 16.9 9.47   
 9 2020 12 33.9 53.0 100. 103. 118. 158. 134. 65.9 48.4   
10 2021 1 28.8 46.2 72.3 72.5 83.5 106. 85.2 40.0 29.1   
11 2021 2 20.9 NA 45.8 47.7 56.7 66.7 59.1 26 14.3

코드 설명

* covid19 데이터프레임을 %>%로 mutate()에 전달
* mutate()를 사용하여 date에서 뽑은 연도, 월(yearmonth())를 yearmon 열에 저장
* group\_by()로 yearmon 열로 그룹핑한후 summarise()를 사용하여 연령대별 평균을 각각의 열로 저장

mean.covid19.by.age %>%  
 tidyr::gather(category, value, 2:10) %>%  
 ggplot(aes(x = yearmon, y = value)) +   
 geom\_line(aes(group = category, color = category)) +   
 labs(title = '월간 평균 확진자수', x = '시간', y = '평균확진자', color = '세대')

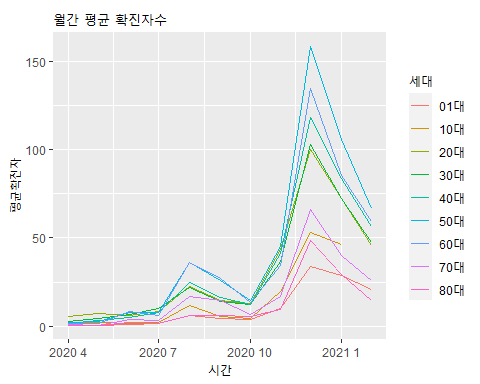


그림 4-4

데이터 프레임 시계열 데이터 그룹핑의 또 하나의 방법은 tibbletime 패키지에서 제공하는 collapse\_by() 함수를 사용하는 방법이다. 이 방법을 사용하기 위해서는 tibbletime 패키지를 설치, 로딩해야 하며 데이터프레임을 tbl\_time 클래스 객체로 변환해야 한다. 변환된 tbl\_time 객체에 적용된 collapse\_by()는 시간 인덱스 열을 직접 컨트롤하여 매개변수로 전달된 기간마다 데이터를 접은(collapse) 후에 group\_by() 함수를 이용해 그룹핑하여 기간별 데이터를 산출할 수 있다.[[3]](#footnote-3) 접힌 데이터의 시간 인덱스는 접힌 마지막 날로 날짜가 바뀌어서 저장된다.

library(tibbletime)  
 as\_tbl\_time(covid19, index = date) %>%  
 collapse\_by('weekly') %>%  
 group\_by(date) %>%   
 summarise(`01대` = mean(`0-9세`),   
 `10대` = mean(`10-19세`),   
 `20대` = mean(`20-29세`),   
 `30대` = mean(`30-39세`),   
 `40대` = mean(`40-49세`),   
 `50대` = mean(`50-59세`),   
 `60대` = mean(`60-69세`),   
 `70대` = mean(`70-79세`),   
 `80대` = mean(`80세 이상`)) %>%  
 tidyr::gather(category, value, 2:10) %>%  
 ggplot(aes(x = date, y = value)) +   
 geom\_line(aes(group = category, color = category)) +   
 labs(title = '주간 평균 확진자수', x = '월', y = '평균확진자', color = '세대')

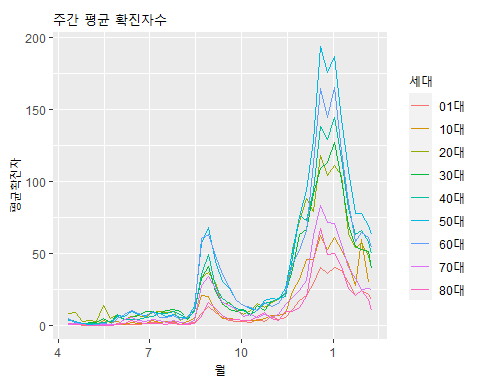


그림 4‑5

코드 설명

* as\_tble\_time()을 사용하여 covid19데이터 프레임을 tibbletime 클래스로 변환
* collapse\_by()를 이용하여 주간단위로 데이터를 접어줌.
* group\_by()로 주간 단위로 바뀌어진 날짜로 grouping
* summarise()로 각 세대별 합계값을 구해 열을 생성
* plotting을 위해 gather()를 사용해 넓은 형태 데이터를 긴 형태 데이터로 변환. 열 이름들을 저장할 열은 category, 열에 저장된 값을 저장할 열은 value, category에 저장될 열들은 2번째부터 10번째 까지임.
* ggplot()로 다변량 라인 plot 생성
* 데이터프레임 : timetk 패키지

데이터프레임으로 저장된 시계열 데이터에 대한 그룹핑의 또 하나의 방법은 timetk 패키지의 summarise\_by\_time() 를 사용하는 것이다. 매개변수인 .date\_var(시간컬럼), .by(그룹핑 단위)를 사용하여 시간 열과 그룹핑 단위를 설정하면 설정된 단위대로 데이터를 그룹핑해준다. 이 방법을 통해 시간을 그룹핑하고 그룹핑된 범위에 해당하는 데이터에 원하는 작업을 수행하여 값들을 얻을 수 있다.

library(timetk)  
 covid19 %>%   
 summarise\_by\_time(.date\_var = date, .by = 'week',   
 `01대` = mean(`0-9세`),   
 `10대` = mean(`10-19세`),  
 `20대` = mean(`20-29세`),  
 `30대` = mean(`30-39세`),  
 `40대` = mean(`40-49세`),  
 `50대` = mean(`50-59세`),  
 `60대` = mean(`60-69세`),  
 `70대` = mean(`70-79세`),  
 `80대` = mean(`80세 이상`)) %>%  
 tidyr::gather(category, value, 2:10) %>%  
 ggplot(aes(x = date, y = value)) +   
 geom\_line(aes(group = category, color = category)) +   
 labs(title = '주간 평균 확진자수', x = '월', y = '평균확진자', color = '세대')

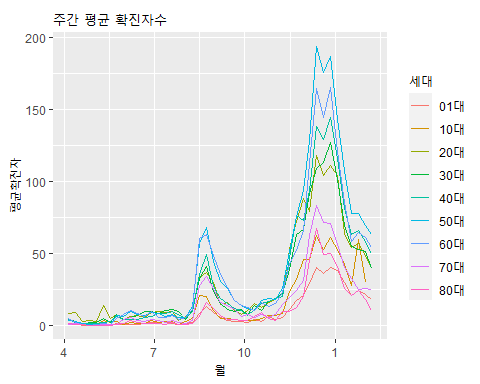


그림 4‑6

코드 설명

* %>%을 사용하여 summarise\_by\_time()에 covid19를 전달하고 그룹핑에 사용할 열(.date\_var =)과 그룹핑 주기(.by = 'week')를 설정
* 이후 각 열에 함수(mean())를 적용.
* plotting을 위해 gather()를 사용해 넓은 형태 데이터를 긴 형태 데이터로 변환. 열 이름들을 저장할 열은 category, 열에 저장된 값을 저장할 열은 value, category에 저장될 열들은 2번째부터 10번째 까지임.
* ggplot()로 다변량 라인 plot 생성

employees %>%   
 summarise\_by\_time(.date\_var = time, .by = 'month',   
 total.year = sum(total),   
 employees.edu = sum(employees.edu)) %>%  
 head(10)

# A tibble: 10 x 3  
 time total.year employees.edu  
 <date> <int> <int>  
 1 2013-01-01 24287 1710  
 2 2013-02-01 24215 1681  
 3 2013-03-01 24736 1716  
 4 2013-04-01 25322 1745  
 5 2013-05-01 25610 1774  
 6 2013-06-01 25686 1786  
 7 2013-07-01 25681 1813  
 8 2013-08-01 25513 1811  
 9 2013-09-01 25701 1794  
10 2013-10-01 25798 1790

코드 설명

* %>%을 사용하여 summarise\_by\_time()에 employees를 전달하고 그룹핑에 사용할 열(.date\_var =)과 그룹핑 주기(.by = 'month')를 설정
* 이 후 total에 함수(sum())를 적용.
* tsibble : index\_by()

tsibble 객체는 데이터프레임 객체와 동일하게 처리 할 수 있다. 하지만 tsibble 객체에서만 동작하는 함수인 index\_by()를 사용하면 쉽게 구할 수 있다. index\_by()는 tsibble 객체 생성시 지정한 index 컬럼을 사용하여 그룹핑한 새로운 컬럼을 생성한다. 여기서 주의해야할 사항은 grouping 함수 앞에 ~를 붙여야 한다는 것과 grouping 함수의 매개변수로 ‘.’을 사용해야 한다는 점이다. ’.’의 의미는 index 컬럼을 사용한다는 의미이다.

그룹핑 주기는 다음과 같다.

|  |  |
| --- | --- |
| 함수명 | 주기 |
| lubridate::year | 연도 주기 |
| yearquarter | 분기 주기 |
| yearmonth | 월 주기 |
| yearweek | 주 주기 |
| as.Date | 일 주기 |
| celling\_date, floor\_date, round\_date | 상세 주기 |

employees.tsibble%>%  
 index\_by(yearqtr = ~ yearquarter(.)) %>%  
 summarise(sum.qtrly = sum(total)) %>%   
 head(10)

# A tsibble: 10 x 2 [1Q]  
 yearqtr sum.qtrly  
 <qtr> <int>  
 1 2013 Q1 73238  
 2 2013 Q2 76618  
 3 2013 Q3 76895  
 4 2013 Q4 76841  
 5 2014 Q1 75629  
 6 2014 Q2 78275  
 7 2014 Q3 78676  
 8 2014 Q4 78186  
 9 2015 Q1 76629  
10 2015 Q2 79024

covid19.tsibble[, c(1,3)]%>%  
 index\_by(yearweek = ~ yearweek(.)) %>%  
 summarise(sum.weekly = sum(`0-9세`)) %>%   
 head(10)

# A tsibble: 10 x 2 [1W]  
 yearweek sum.weekly  
 <week> <dbl>  
 1 2020 W15 4  
 2 2020 W16 8  
 3 2020 W17 3  
 4 2020 W18 -1  
 5 2020 W19 1  
 6 2020 W20 6  
 7 2020 W21 1  
 8 2020 W22 8  
 9 2020 W23 7  
10 2020 W24 3

index\_by()를 사용할 때 장점은 그룹핑 기간을 자유자재로 설정할 수 있다는 점이다. 월, 분기, 년과 같이 달력상의 주기 외에 ‘매 4일마다’, ’매 2달마다’와 같이 주기를 설정할 수 있다.

covid19.tsibble[, c(1,3)]%>%  
 index\_by(twoweek = ~ lubridate::floor\_date(., "2 month")) %>%  
 summarise(sum.2week = sum(`0-9세`)) %>%   
 head(10)

# A tsibble: 6 x 2 [1D]  
 twoweek sum.2week  
 <date> <dbl>  
1 2020-03-01 14  
2 2020-05-01 47  
3 2020-07-01 249  
4 2020-09-01 248  
5 2020-11-01 1317  
6 2021-01-01 1081

covid19.tsibble[, c(1,3)]%>%  
 index\_by(fourday = ~ lubridate::floor\_date(., "4 day")) %>%  
 summarise(sum.4days = sum(`0-9세`)) %>%   
 head(10)

# A tsibble: 10 x 2 [1D]  
 fourday sum.4days  
 <date> <dbl>  
 1 2020-04-09 4  
 2 2020-04-13 5  
 3 2020-04-17 3  
 4 2020-04-21 0  
 5 2020-04-25 2  
 6 2020-04-29 0  
 7 2020-05-01 0  
 8 2020-05-05 1  
 9 2020-05-09 1  
10 2020-05-13 5

* xts

xts 객체의 월별, 분기별, 연별 그룹핑을 위해서 제공하는 함수는 ’apply.’으로 시작하는 함수이다. 이 함수에 xts 객체와 적용하고자 하는 함수(sum, mean 등)을 전달하면 값을 산출할 수 있다.

apply. 계열 함수는 다음의 5가지가 있다. 산출된 결과는 plot.xts()로 간단히 plotting 할 수 있다.

|  |  |
| --- | --- |
| 함수명 | 설명 |
| apply.daily(xts객체, 함수명) | 일별로 함수를 적용한 결과값 반환 |
| apply.weekly(xts객체, 함수명) | 주별로 함수를 적용한 결과값 반환 |
| apply.monthly(xts객체, 함수명) | 월별로 함수를 적용한 결과값 반환 |
| apply.quarterly(xts객체, 함수명) | 분기별로 함수를 적용한 결과값 반환 |
| apply.yearly(xts객체, 함수명) | 년별로 함수를 적용한 결과값 반환 |

library(xts)  
 apply.quarterly(employees.xts, sum) %>% head(10)  
 [,1]  
2013-03-01 78345  
2013-06-01 81923  
2013-09-01 82313  
2013-12-01 82203  
2014-03-01 80977  
2014-06-01 83779  
2014-09-01 84226  
2014-12-01 83741  
2015-03-01 82095  
2015-06-01 84488

apply.yearly(employees.xts, sum) %>% plot.xts()

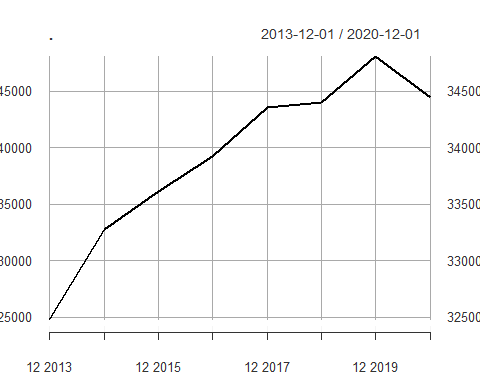


그림 4‑7

apply.monthly(covid19.xts[,1], sum) %>% plot.xts(main = '월별 0-9세 코로나 확진자수')

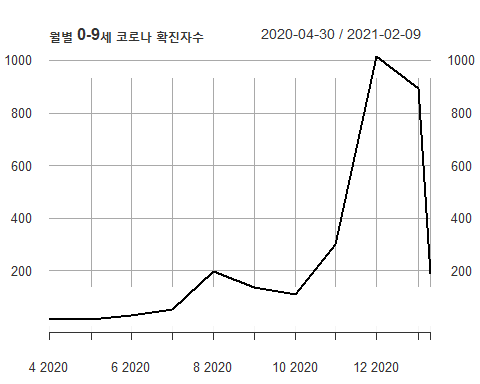


그림 4‑8

apply.quarterly(covid19.xts[,1], sum) %>% plot.xts(main = '분기별 0-9세 코로나 확진자수')

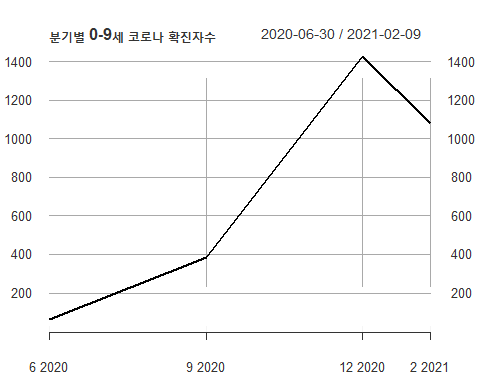


그림 4‑9

* ts

ts 객체는 다음 장부터 다뤄질 시계열 분석 패키지인 forecast 패키지에서 주로 사용되는 객체이다. 하지만 앞선 데이터프레임이나 xts 처럼 처리가 용이하지 않다. 따라서 데이터 처리를 위해서는 데이터프레임이나 xts 객체로 변환하여 사용하는 것이 정신 건강에 좋다.

# 주식 시가, 고가, 저가, 종가는 어떻게 구해? : OHLC

주식 프로그램을 보면 시계열 plot과 함께 제공되는 몇 가지 정보가 있는데 바로 시가, 고가, 저가, 종가 정보이다. 이 네 가지 정보를 시계열 데이터에서는 영문 이니셜을 따서 OHLC라고 한다. OHLC는 주어진 기간 동안의 첫번째 값(Open), 최대값(High), 최소값(Low), 마지막 값(Close) 값을 말한다.

* 데이터프레임 : lubridate, tibbletime, dplyr 패키지

앞서 소개한 collapse\_by() 함수를 사용하면 OHLC 정보를 간단히 산출할 수 있다. collapse\_by()를 이용해 그룹핑할 시간 주기를 설정하고 group\_by()함수를 사용하여 데이터를 그룹핑한 후 lubridate 패키지의 first(), max(), min(), last() 함수를 사용하면 산출이 가능하다.

as\_tbl\_time(covid19, index = date) %>%  
 collapse\_by('weekly') %>%  
 group\_by(date) %>%   
 summarise(Open = first(`0-9세`),   
 High = max(`0-9세`),   
 Low = min(`0-9세`),   
 Close = last(`0-9세`)) %>%  
 head(10)

# A time tibble: 10 x 5  
# Index: date  
 date Open High Low Close  
 <date> <dbl> <dbl> <dbl> <dbl>  
 1 2020-04-11 2 2 1 1  
 2 2020-04-18 0 3 0 2  
 3 2020-04-25 0 1 0 1  
 4 2020-05-02 2 2 -1 0  
 5 2020-05-09 0 1 0 0  
 6 2020-05-16 0 2 0 2  
 7 2020-05-23 0 1 0 0  
 8 2020-05-30 0 3 0 1  
 9 2020-06-06 0 3 0 3  
10 2020-06-13 1 1 0 0

* xts

xts 패키지에서는 OHLC 정보를 확인할 수 있는 함수를 직접 제공한다. to.period()인데 OHLC를 제공하는데 to.period() 함수의 매개변수인 method에 ‘months’, ‘quarters’ 등 적절한 시간 간격을 설정해주면 해당 기간 동안의 OHCL를 자동으로 출력해준다.

to.period(covid19.xts, method = 'months', OLHC = TRUE)

covid19.xts.Open covid19.xts.High covid19.xts.Low covid19.xts.Close  
2020-04-30 2 5 1 2  
2020-05-31 0 11 0 7  
2020-06-30 0 6 0 6  
2020-07-31 0 10 -4 13  
2020-08-31 1 35 0 15  
2020-09-30 14 17 0 20  
2020-10-31 6 11 0 22  
2020-11-30 5 57 9 32  
2020-12-31 15 86 0 132  
2021-01-31 41 138 44 49  
2021-02-08 20 41 30 31

# 3일 평균, 5일 합계는? : 시간 Rolling

주식 거래 프로그램에서는 많은 시계열 그래프들을 보여준다. 아래의 plot에서 보면 초록색 선은 5일 동안의 평균, 빨간색 선은 20일 동안의 평균, 주황색 선은 60일 동안의 평균을 나타낸다. 이를 이동평균이라고 한다.

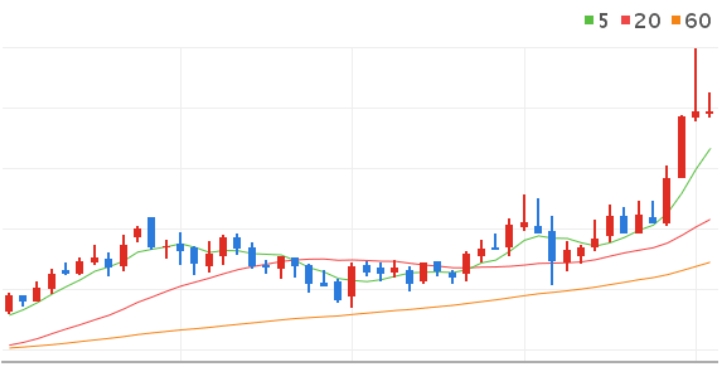


그림 4-10

이동평균은 평균의 산출 기간을 현재일로부터 특정일까지로 정해서 구하는 평균으로 평균을 구하는 기간이 매일 달라진다. 이와 같이 기준일에서 특정 기간 동안에 집합 함수를 적용하는 과정을 rolling이라 하고 rolling에 적용되는 기간을 rolling window라고 한다. 위의 그림에서 5일 이동평균의 rolling window는 5, 20일 이동평균의 rolling window는 20이 된다.

지금까지 시계열 데이터를 다루는데 계속 사용했던 lubridate 패키지에서는 rolling에 관련된 함수를 제공하지 않는다. 대신 zoo 패키지에서 제공하는 함수를 dplyr 함수와 파이프라인을 사용하는 방법, timetk 패키지를 사용하는 방법을 사용하여 산출 할 수 있고 xts 시계열 데이터는 xts 패키지에서 제공하는 함수를 사용할 수 있다.

* 데이터프레임 : zoo, dplyr 패키지

데이터프레임으로 저장된 시계열 객체에 대한 rolling은 zoo 패키지에서 제공하는 roll\* 함수를 사용하면 비교적 쉽게 rolling을 수행할 수있다.

roll\* 함수는 rollapply()와 rollmean()의 두가지를 제공하는데 rollapply()는 apply()를 rolling용으로 수정한 버전이다. rollmean()은 rollapply()에 평균(mean())을 적용하는 함수로 rolling에는 평균을 사용하는 경우가 많기 때문에 이를 위해서 특별히 rollmean()을 제공한다.

library(zoo)  
 employees %>%  
 mutate(ma3 = rollmean(total, k = 3, fill = NA),  
 sum3 = rollapply(total, 3, sum, fill = NA)) %>%  
 select(time, total, ma3, sum3) %>%  
 ggplot(aes(x = time)) +   
 geom\_line(aes(y = total, group = 1, color = 'total')) +  
 geom\_line(aes(y = ma3, group = 1, color = 'ma3')) +   
 scale\_color\_manual(values = c('total' = 'red', 'ma3' = 'blue'))

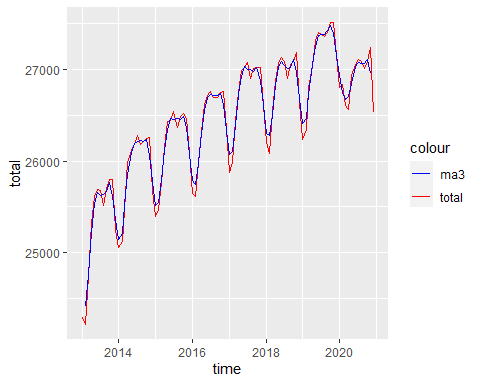


그림 4-11

코드 설명

* %>%를 사용하여 mutate()에 employees를 전달.
* mutate()로 total 열의 데이터를 rolling window가 3(k = 3)인 평균(rollmean)을 구하는 ma3 열을 생성하는데 빈 행은 NA로 채움(fill = NA).
* mutate()로 total 열의 데이터를 rolling window가 3인 합계(sum)를 구하는 ma3 열을 생성(rollapply())하는데 빈 행은 NA로 채움(fill = NA).
* 결과중에 time, total, ma3, sum3만 선택
* ggplot()를 사용하여 다변량 라인 plot을 그림
* 데이터프레임 : timetk 패키지

timetk 패키지에서는 rolling 기능을 위해 slidify() 함수를 제공한다. slidify() 함수가 타 함수와 다른 점은 그 반환 값이 함수의 실행 결과값이 아니고 함수 자체를 반환한다는 것이다.

즉 rolling window 기간에 특정 함수를 적용하는 함수를 제공하기 때문에 이후 함수의 형태로 사용해야 한다.

아래의 예에 ma3와 sum3는 data.frame이나 숫자값이 아니고 함수 객체이다. 이 함수 객체는 mutate() 안에서 함수로 실행하는데 그 매개변수로 rolling을 해야할 컬럼을 전달하면 그 결과 벡터가 전달되는 형태로 실행된다.

ma3 <- slidify(mean, .period = 3, .align = "center")  
 sum3 <- slidify(sum, .period = 3, .align = "center")  
 class(ma3)  
[1] "function"

class(sum3)  
[1] "function"

employees %>%  
 mutate(ma3 = ma3(total), sum3 = sum3(total)) %>%  
 select(time, total, ma3, sum3) %>%  
 ggplot(aes(x = time)) +   
 geom\_line(aes(y = total, group = 1, color = 'total')) +  
 geom\_line(aes(y = ma3, group = 1, color = 'ma3')) +   
 scale\_color\_manual(values = c('total' = 'red', 'ma3' = 'blue'))

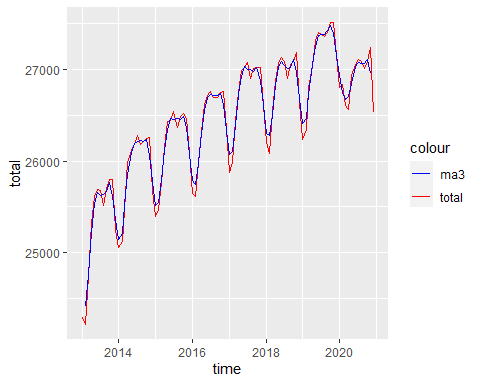


그림 4-12

코드 설명

* slidify()를 사용하여 rolling window가 3인 평균함수를 ma3로 생성
* slidify()를 사용하여 rolling window가 3인 합계함수를 sum3로 생성
* ma3와 sum3의 클래스를 확인해보면 function임을 확인
* mutate()로 slidify()로 생성한 함수로 total 열에 대한 rolling window가 3인 평균과 합계를 구해 ma3, sum3 열을 생성
* 결과중에 time, total, ma3, sum3만 선택
* ggplot()를 사용하여 다변량 라인 plot을 그림
* xts

xts 패키지에서는 rolling을 위한 간단한 함수를 제공한다. rollapply()로 xts 객체와 rolling window 기간, 적용할 함수를 전달하면 rolling을 간단히 구할 수 있다.

rollapply(employees.xts, width = 3, FUN = mean) %>%  
 head(10)

total employees.edu  
2013-01-01 NA NA  
2013-02-01 NA NA  
2013-03-01 24412.67 1702.333  
2013-04-01 24757.67 1714.000  
2013-05-01 25222.67 1745.000  
2013-06-01 25539.33 1768.333  
2013-07-01 25659.00 1791.000  
2013-08-01 25626.67 1803.333  
2013-09-01 25631.67 1806.000  
2013-10-01 25670.67 1798.333

# 지난 달 데이터는? : Filtering (Subsetting)

시계열 데이터가 아닌 데이터들은 데이터 값에 대한 조건을 적용하여 결과를 추출한다. 예를 들어 판매량이 100보다 크거나 취업자수가 10,000명보다 큰 경우 등이다. 물론 시계열 데이터도 데이터 값에 조건을 적용하여 조건에 맞는 행의 추출이 가능하지만 시간 인덱스에 조건을 적용하여 추출도 가능하다. 이 방법은 filtering이라고도 하고 subsetting이라고도 한다. 여기서는 시간 인덱스에 조건을 적용하는 방법에 대해 설명한다.

* 데이터프레임 : lubridate, dplyr 패키지

데이터프레임에 저장된 시계열 객체의 시간 인덱스 컬럼은 조건 연산자를 동일하게 사용할 수 있다. 따라서 <, <=, =, >, >= 등의 연산자를 dplyr 패키지에서 제공하는 filter()와 파이프라인(%>%)을 사용하여 시간 인덱스에 대해 조건 별로 filtering 할 수 있다.

covid19 %>%  
 filter(date = as.Date('2020-10-01') & date <= as.Date('2020-10-10'))

date status 0-9세 10-19세 20-29세 30-39세 40-49세 50-59세 60-69세  
1 2020-10-01 신규 6 4 7 9 14 13 13  
2 2020-10-02 신규 3 6 6 10 6 11 13  
3 2020-10-03 신규 1 1 6 13 11 11 16  
4 2020-10-04 신규 0 4 4 9 13 8 12  
5 2020-10-05 신규 0 3 17 9 8 12 14  
6 2020-10-06 신규 5 8 26 8 2 11 9  
7 2020-10-07 신규 1 2 8 13 10 28 31  
8 2020-10-08 신규 6 5 6 7 9 12 11  
9 2020-10-10 신규 0 0 0 0 0 0 0  
 70-79세 80세 이상  
1 7 4  
2 4 4  
3 13 3  
4 8 6  
5 6 4  
6 4 2  
7 16 5  
8 10 3  
9 0 0

covid19 %>%  
 filter(between(date, as.Date('2021-01-01'), as.Date('2021-01-15')))

date status 0-9세 10-19세 20-29세 30-39세 40-49세 50-59세 60-69세  
1 2021-01-01 신규 41 74 131 145 154 204 159  
2 2021-01-02 신규 39 66 111 109 106 160 125  
3 2021-01-03 신규 32 46 80 75 88 124 117  
4 2021-01-04 신규 40 71 120 127 172 187 153  
5 2021-01-05 신규 45 44 107 105 112 124 108  
6 2021-01-06 신규 45 60 106 106 110 140 148  
7 2021-01-07 신규 41 54 123 120 155 176 113  
8 2021-01-08 신규 34 43 91 86 83 124 125  
9 2021-01-09 신규 32 50 100 91 89 124 80  
10 2021-01-10 신규 36 56 87 79 94 143 87  
11 2021-01-11 신규 19 36 47 54 61 79 79  
12 2021-01-12 신규 32 34 71 59 91 102 74  
13 2021-01-13 신규 24 28 66 81 83 129 87  
14 2021-01-14 신규 25 45 60 67 82 98 71  
15 2021-01-15 신규 42 37 62 70 81 97 91  
 70-79세 80세 이상  
1 80 41  
2 59 49  
3 51 38  
4 69 81  
5 44 26  
6 73 51  
7 59 27  
8 49 37  
9 50 25  
10 49 34  
11 44 31  
12 39 35  
13 47 16  
14 39 29  
15 22 11

employees %>%  
 filter(year(time) == 2019 & month(time) == 5)

time total employees.edu  
1 2019-05-01 27322 1884

# 매월 3일부터 7일까지 filtering  
 covid19 %>%  
 filter(between(day(date), 3, 7)) %>%  
 head(15)

date status 0-9세 10-19세 20-29세 30-39세 40-49세 50-59세 60-69세  
1 2020-05-03 신규 0 1 2 3 2 0 2  
2 2020-05-04 신규 0 0 2 1 3 0 2  
3 2020-05-05 신규 0 0 0 1 1 1 0  
4 2020-05-06 신규 0 0 0 1 0 0 1  
5 2020-05-07 신규 0 1 2 1 0 0 0  
6 2020-06-03 신규 1 2 5 6 5 9 15  
7 2020-06-04 신규 1 0 5 3 3 9 9  
8 2020-06-05 신규 1 2 5 2 3 7 10  
9 2020-06-06 신규 3 1 2 6 11 8 9  
10 2020-06-07 신규 1 6 3 6 8 11 12  
11 2020-07-03 신규 2 10 10 10 6 7 11  
12 2020-07-04 신규 3 2 12 16 8 8 9  
13 2020-07-05 신규 3 3 8 9 11 7 14  
14 2020-07-06 신규 4 1 7 10 6 9 4  
15 2020-07-07 신규 1 2 12 6 7 7 7  
 70-79세 80세 이상  
1 1 2  
2 0 0  
3 0 0  
4 0 0  
5 0 0  
6 6 0  
7 6 3  
8 7 2  
9 8 3  
10 8 2  
11 4 3  
12 2 3  
13 6 0  
14 2 3  
15 2 0

* 데이터프레임 : timetk 패키지

timetk 패키지에서는 filter\_by\_time()을 제공한다. 매개변수 .start로 시작일과 .end 종료일을 전달하면 해당 기간 동안의 데이터를 반환한다.

또 timetk 패키지에서는 filter\_by\_time()에 데이터 값에 대한 조건을 추가할 수 있는 filter\_period()를 제공한다. filter\_by\_time()와 같이 매개변수 .start로 시작일과 .end 종료일을 설정하고 데이터에 적용할 조건을 설정하면 적합한 결과들을 산출해서 반환한다. 예를 들어 매월 최대 매출일을 기록한 날, 매년 월평균 매출액보다 큰 매출을 기록한 월 등을 산출할 때 유용하다.

covid19 %>%  
 filter\_by\_time(.date\_var = date, .start = '2020-10-01', .end = '2020-10-05')

date status 0-9세 10-19세 20-29세 30-39세 40-49세 50-59세 60-69세  
1 2020-10-01 신규 6 4 7 9 14 13 13  
2 2020-10-02 신규 3 6 6 10 6 11 13  
3 2020-10-03 신규 1 1 6 13 11 11 16  
4 2020-10-04 신규 0 4 4 9 13 8 12  
5 2020-10-05 신규 0 3 17 9 8 12 14  
 70-79세 80세 이상  
1 7 4  
2 4 4  
3 13 3  
4 8 6  
5 6 4

covid19 %>%  
 filter(`0-9세` != 0) %>%  
 filter\_period(.date\_var = date, .period = '1 month', `0-9세` == max(`0-9세`)) %>%  
 head(10)

# A tibble: 10 x 11  
 date status `0-9세` `10-19세` `20-29세` `30-39세` `40-49세` `50-59세`  
 <date> <fct> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>  
 1 2020-04-16 신규 3 3 5 1 2 3  
 2 2020-05-27 신규 3 2 8 6 3 6  
 3 2020-06-06 신규 3 1 2 6 11 8  
 4 2020-06-23 신규 3 0 4 9 9 9  
 5 2020-06-29 신규 3 1 9 5 6 5  
 6 2020-06-30 신규 3 4 10 6 8 6  
 7 2020-07-17 신규 10 4 26 37 32 23  
 8 2020-08-23 신규 16 35 34 40 62 82  
 9 2020-08-28 신규 16 17 32 48 53 73  
10 2020-08-29 신규 16 18 31 36 47 54  
# ... with 3 more variables: `60-69세` <dbl>, `70-79세` <dbl>, `80세  
# 이상` <dbl>

코드 설명

* 2020-10-01부터 2020-10-05까지의 데이터 산출
* filter\_period()를 사용하여 코로나19 데이터에서 0-9세가 0인 행을 제거(filter(`0-9세` != 0))하고 월단위로 쪼개서(.period = ‘1 month’) 0-9세 확진자수가 가장 많은 데이터(`0-9세` == max(`0-9세`))를 산출
* xts

xts 객체의 시간 인덱스 filtering은 앞선 경우 와는 조금 다른 방법을 사용한다. xts 객체는 기간 조건을 설정할 때에는 함수를 사용하지 않고 연산자인 /로 기간 구간을 설정할 수 있다. /을 사용하여 조건을 적용할 때는 []을 사용하여 조건을 적용할 수 있다.

/연산자는 우리가 흔히 쓰는 ~로 생각하면 편리한데 날짜 사이에 /가 들어가면 두 날짜 사이 기간을 의미하고 날짜 앞에 /가 오면 처음부터 날짜까지의 기간, 날짜 뒤에 /가 오면 날짜부터 끝까지의 기간을 의미한다.

# 2020-10-02에 해당하는 데이터 filtering  
 covid19.xts['2020-10-02']  
 0-9세 10-19세 20-29세 30-39세 40-49세 50-59세 60-69세 70-79세  
2020-10-02 3 6 6 10 6 11 13 4

# 2020-10-01에서부터 2020-10-10 까지 데이터 filtering  
 covid19.xts['2020-10-01/2020-10-10']   
 0-9세 10-19세 20-29세 30-39세 40-49세 50-59세 60-69세 70-79세  
2020-10-01 6 4 7 9 14 13 13 7  
2020-10-02 3 6 6 10 6 11 13 4  
2020-10-03 1 1 6 13 11 11 16 13  
2020-10-04 0 4 4 9 13 8 12 8  
2020-10-05 0 3 17 9 8 12 14 6  
2020-10-06 5 8 26 8 2 11 9 4  
2020-10-07 1 2 8 13 10 28 31 16  
2020-10-08 6 5 6 7 9 12 11 10  
2020-10-10 0 0 0 0 0 0 0 0

# 2021-02-05일 부터 끝까지 데이터 filtering  
 covid19.xts['2021-02-05/']   
 0-9세 10-19세 20-29세 30-39세 40-49세 50-59세 60-69세 70-79세  
2021-02-05 30 28 30 60 42 65 75 25  
2021-02-06 25 31 53 41 60 77 63 30  
2021-02-07 21 27 51 47 44 79 66 26  
2021-02-08 19 9 35 31 53 52 59 22  
2021-02-09 15 NA 33 41 52 59 38 28

# 처음부터 2020-04-11까지의 filtering  
 covid19.xts['/2020-04-11']   
 0-9세 10-19세 20-29세 30-39세 40-49세 50-59세 60-69세 70-79세  
2020-04-09 2 4 12 7 7 2 2 0  
2020-04-10 1 1 7 4 2 3 6 2  
2020-04-11 1 5 5 2 3 6 7 0

# 월별, 분기별, 연별 증감량

월별, 분기별, 연별 증감량을 산출하기 위해서는 바로 직전 데이터와의 차감 데이터를 계산해야 한다. 이와 같이 시계열 분석에서는 이전 데이터와의 차감 데이터가 매우 중요하게 활용된다. 차감데이터가 증감량이며 이 차감 데이터를 원본데이터로 나누면 증감률이 된다.

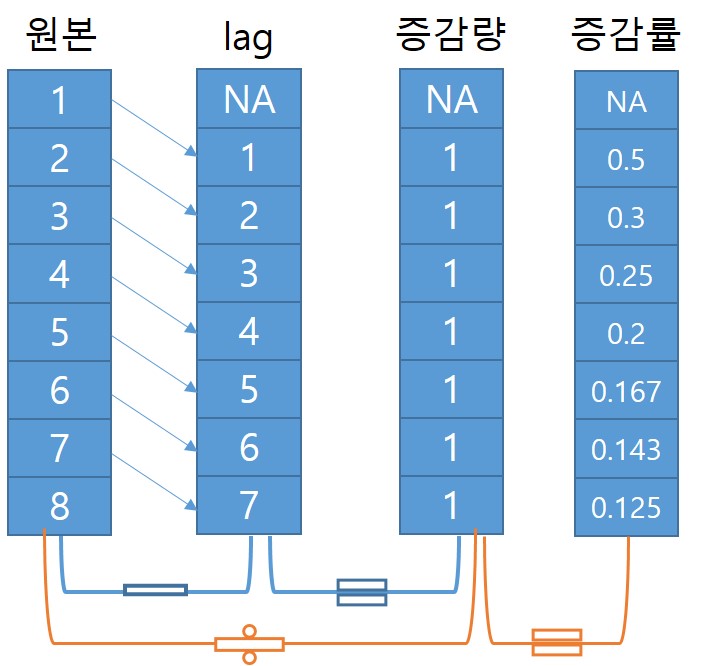


그림 4-13

* 데이터프레임 : lag() 함수

시계열 데이터가 저장된 데이터프레임의 데이터 열에 lag()를 적용하면 데이터를 하나씩 Shift 시킨 데이터 열을 돌려준다. 이 열을 데이터프레임에 추가해주고 mutate()를 사용해 원본데이터에서 lag된 데이터를 빼주면 증감량을 얻을 수 있고 증감량을 원본 데이터로 나눠주면 증감률을 구할 수 있다.

students\_lag <- cbind(연도 = students$연도,   
 학생수계 = students$학생수계,   
 전년 = students %>%   
 lag(1)%>%   
 select(학생수계) %>%   
 rename(전년 = 학생수계)  
 ) %>%  
 mutate(증감 = 학생수계 - 전년, 증감율 = round((학생수계/전년)-1, 3) \* 100)  
 students\_lag %>% head()

연도 학생수계 전년 증감 증감율  
1 1999-01-01 8658358 NA NA NA  
2 2000-01-01 8535867 8658358 -122491 -1.4  
3 2001-01-01 8414423 8535867 -121444 -1.4  
4 2002-01-01 8361933 8414423 -52490 -0.6  
5 2003-01-01 8379775 8361933 17842 0.2  
6 2004-01-01 8371630 8379775 -8145 -0.1

* 코드 설명
  + cbind()로 student의 연도, 학생수계 열과 lag()의 결과인 lag된 열을 묶어서 student\_lag 데이터프레임을 생성
  + 이중 lag 열의 이름을 전년으로 바꿔줌
  + mutate()로 증감(학생수계 - 전년)과 증감률(round((학생수계/전년)-1, 3) \* 100) 열을 생성

students\_lag %>%  
 ggplot(aes(as.factor(year(연도)), 증감)) +   
 geom\_line(aes(group = 1)) +   
 geom\_point() +  
 ggrepel::geom\_text\_repel(aes(label = scales::comma(증감)), vjust = 1, size = 3) + # ggrepel::geom\_text\_repel() 함수로 숫자들이 겹치지 않게 plotting  
 labs(title = '전년대비 전체 학생수 증감 추이', x = '연도', y = '학생수 증감량') +  
 scale\_y\_continuous(labels = scales::number\_format(big.mark = ',')) +  
 theme(axis.text.x=element\_text(angle=90,hjust=1))

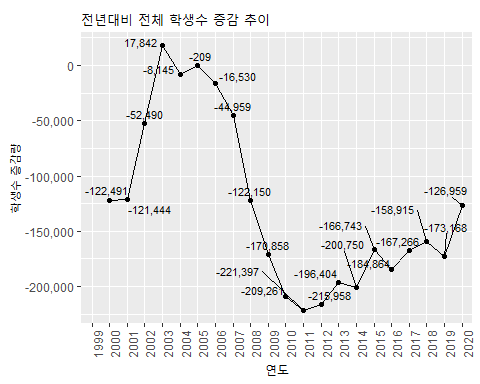


그림 4-14

* tsibble : diffrence()

앞선 예제에서는 lag() 함수를 사용하여 원 데이터를 하나씩 shift한 데이터 열을 생성하고 두 열의 차이를 구한 열을 mutate()로 만들었다. tsibble 객체는 이 과정을 바로 지원하는 difference()를 제공한다. difference()는 시차 데이터(lag)를 생성하고 이 시차 데이터와 원본 데이터의 차이를 가지는 데이터 열을 생성해주는 함수이다. 따라서 lag()를 사용하는 것 보다 단계를 줄일 수 있다.

students.tsibble%>% select(1, 2) %>%  
+ mutate(증감 = difference(.$학생수계, lag = 1)) %>%  
+ mutate(증감율 = round((증감/학생수계), 3) \* 100) %>% head(10)

# A tsibble: 10 x 4 [1D]  
 연도 학생수계 증감 증감율  
 <date> <dbl> <dbl> <dbl>  
 1 1999-01-01 8658358 NA NA   
 2 2000-01-01 8535867 -122491 -1.4  
 3 2001-01-01 8414423 -121444 -1.4  
 4 2002-01-01 8361933 -52490 -0.6  
 5 2003-01-01 8379775 17842 0.2  
 6 2004-01-01 8371630 -8145 -0.1  
 7 2005-01-01 8371421 -209 0   
 8 2006-01-01 8354891 -16530 -0.2  
 9 2007-01-01 8309932 -44959 -0.5  
10 2008-01-01 8187782 -122150 -1.5

employees%>%  
 mutate(증감 = difference(employees.tsibble$total, lag = 1)) %>%  
 mutate(증감율 = round((증감/total), 3) \* 100) %>% select(1, 2, 4, 5) %>% head(10)

time total 증감 증감율  
1 2013-01-01 24287 NA NA  
2 2013-02-01 24215 -72 -0.3  
3 2013-03-01 24736 521 2.1  
4 2013-04-01 25322 586 2.3  
5 2013-05-01 25610 288 1.1  
6 2013-06-01 25686 76 0.3  
7 2013-07-01 25681 -5 0.0  
8 2013-08-01 25513 -168 -0.7  
9 2013-09-01 25701 188 0.7  
10 2013-10-01 25798 97 0.4

* xts : diff()

xts 객체도 tsibble 객체와 같이 차분 함수인 diff() 함수를 제공한다. 이후 과정은 tsibble과 동일한 과정을 거친다.

students.xts$증감 <- diff(students.xts[,2])   
 students.xts$증감율 <- round((students.xts$증감/students.xts$학생수계), 3) \* 100  
 students.xts[, c('학생수계', '증감', '증감율')] %>% head(10)

학생수계 증감 증감율  
1999-01-01 8658358 NA NA  
2000-01-01 8535867 11097 0.1  
2001-01-01 8414423 -121 0.0  
2002-01-01 8361933 5114 0.1  
2003-01-01 8379775 -3725 0.0  
2004-01-01 8371630 -4818 -0.1  
2005-01-01 8371421 -110 0.0  
2006-01-01 8354891 4209 0.1  
2007-01-01 8309932 -4262 -0.1  
2008-01-01 8187782 -3728 0.0

plot.xts(students.xts[, '증감율'], main = '전년대비 학생수 증감률')

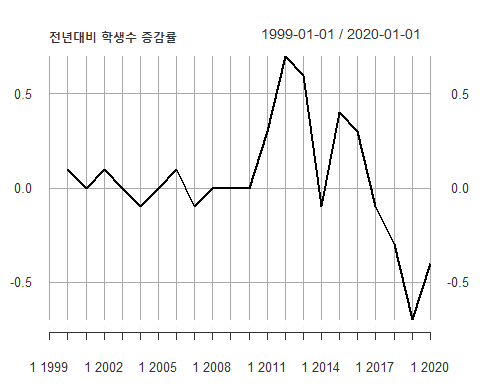


그림 4-15

employees.xts$증감 <- diff(employees.xts$total)  
 employees.xts$증감율 <- round((employees.xts$증감/employees.xts$total), 3) \* 100  
 employees.xts[, c('total', '증감', '증감율')] %>% head(10)

total 증감 증감율  
2013-01-01 24287 NA NA  
2013-02-01 24215 -72 -0.3  
2013-03-01 24736 521 2.1  
2013-04-01 25322 586 2.3  
2013-05-01 25610 288 1.1  
2013-06-01 25686 76 0.3  
2013-07-01 25681 -5 0.0  
2013-08-01 25513 -168 -0.7  
2013-09-01 25701 188 0.7  
2013-10-01 25798 97 0.4

plot.xts(employees.xts[, c('증감율')], main = '전월대비 전체 취업자 증감률')

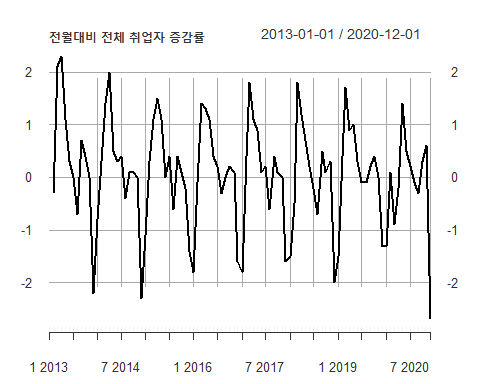


그림 4-16

# 월 비중 백분율, 연 비중 백분율

연간 전체 매출의 월별 비중, 월간 전체 합계 중 일별, 주간별 비중 또는 백분율 등을 산출해야 할 때가 있다. 이런 경우 연간 전체 매출의 월별 비중이나 백분율을 모두 합하면 1 또는 100%가 되어야 한다.

예를 들어 매월 매출액이 기록된 시계열 데이터에서 1년중 비율을 구하기 위해서는 매월 매출액의 옆 컬럼에 해당 년의 전체 매출액 합계가 있어야 비율을 구할 수 있다.

따라서 월 비중, 연 비중 백분율 등을 구하기 위해서는 총 합계가 먼저 계산되어야 한다.

* 데이터프레임 : group\_by(), mutate()

데이터프레임에서 총합계를 구하기 위해서는 먼저 구하고자 하는 총 합계의 주기로 그룹핑 하여 전체 합계를 구해야 한다. 이때 주의해야할 점이 summarise()를 통해 합계를 구하지 않고 mutate()를 사용해 합계를 구해야 한다는 것이다.

group\_by()후 summarise()를 통해 합계를 구하면 전체 데이터 구조가 그룹핑 된 구조로 바뀌게 되지만 mutate() 함수를 통해 합계를 구하면 그룹핑 된 주기별로 합계를 구한 컬럼이 추가되게 되어 비중을 구하기 쉽다. 이 경우 그룹핑 영향을 받는 부분을 정확히 파악하여 group\_by()의 영향을 벗어나야 하는 곳에서 ungroup()을 사용해서 더이상 group\_by()의 영향을 받지 않도록 해줘야 한다.

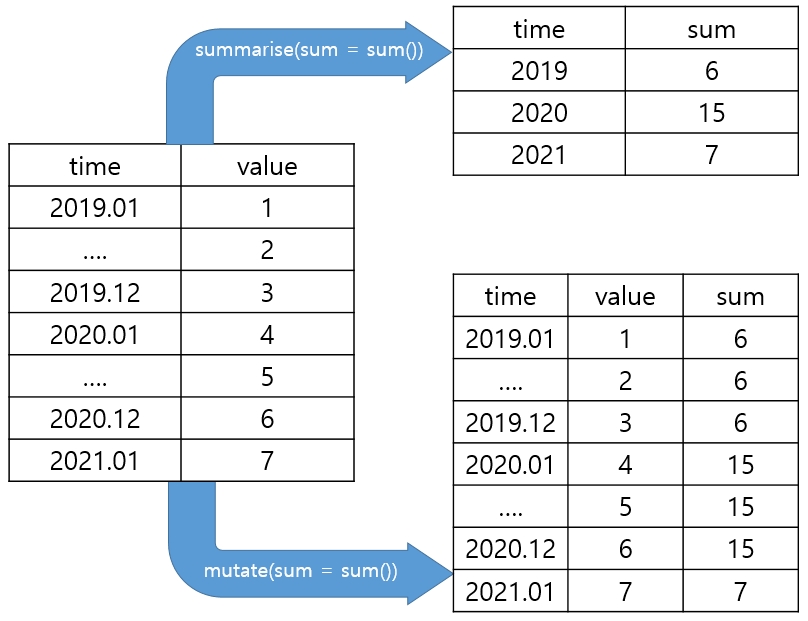


그림 4-17

employees %>%   
 group\_by(year(time)) %>%  
 mutate(sum.by.year = sum(total)) %>%  
 ungroup() %>%  
 mutate(rate.by.year = round(total/sum.by.year, 3) \* 100) %>%  
 head(15)

# A tibble: 15 x 6  
 time total employees.edu `year(time)` sum.by.year rate.by.year  
 <date> <int> <int> <dbl> <int> <dbl>  
 1 2013-01-01 24287 1710 2013 303592 8   
 2 2013-02-01 24215 1681 2013 303592 8   
 3 2013-03-01 24736 1716 2013 303592 8.1  
 4 2013-04-01 25322 1745 2013 303592 8.3  
 5 2013-05-01 25610 1774 2013 303592 8.4  
 6 2013-06-01 25686 1786 2013 303592 8.5  
 7 2013-07-01 25681 1813 2013 303592 8.5  
 8 2013-08-01 25513 1811 2013 303592 8.4  
 9 2013-09-01 25701 1794 2013 303592 8.5  
10 2013-10-01 25798 1790 2013 303592 8.5  
11 2013-11-01 25795 1793 2013 303592 8.5  
12 2013-12-01 25248 1779 2013 303592 8.3  
13 2014-01-01 25050 1748 2014 310766 8.1  
14 2014-02-01 25116 1786 2014 310766 8.1  
15 2014-03-01 25463 1814 2014 310766 8.2

covid19 %>%   
 group\_by(yearmonth(date)) %>%  
 mutate(sum.by.month = sum(`0-9세`)) %>%  
 ungroup() %>%  
 mutate(rate.by.month = round(`0-9세`/sum.by.month, 3) \* 100) %>%  
 select(date, `0-9세`, sum.by.month, rate.by.month)

# A tibble: 302 x 4  
 date `0-9세` sum.by.month rate.by.month  
 <date> <dbl> <dbl> <dbl>  
 1 2020-04-09 2 14 14.3  
 2 2020-04-10 1 14 7.1  
 3 2020-04-11 1 14 7.1  
 4 2020-04-12 0 14 0   
 5 2020-04-13 2 14 14.3  
 6 2020-04-14 0 14 0   
 7 2020-04-15 0 14 0   
 8 2020-04-16 3 14 21.4  
 9 2020-04-17 1 14 7.1  
10 2020-04-18 2 14 14.3  
# ... with 292 more rows

covid19 %>%   
 group\_by(year(date), month(date), week(date)) %>%  
 mutate(sum.by.week = sum(`0-9세`)) %>%  
 ungroup() %>%  
 mutate(rate.by.week = round(`0-9세`/sum.by.week, 3) \* 100) %>%  
 select(date, `0-9세`, sum.by.week, rate.by.week)

# A tibble: 302 x 4  
 date `0-9세` sum.by.week rate.by.week  
 <date> <dbl> <dbl> <dbl>  
 1 2020-04-09 2 6 33.3  
 2 2020-04-10 1 6 16.7  
 3 2020-04-11 1 6 16.7  
 4 2020-04-12 0 6 0   
 5 2020-04-13 2 6 33.3  
 6 2020-04-14 0 6 0   
 7 2020-04-15 0 6 0   
 8 2020-04-16 3 6 50   
 9 2020-04-17 1 6 16.7  
10 2020-04-18 2 6 33.3  
# ... with 292 more rows

* tsibble : index\_by(), mutate() 함수

앞선 월별, 분기별 합산에서 사용했던 index\_by() 함수를 이용하면 데이터를 주기별로 그룹핑 할 수 있다. 이 후 데이터프레임에서 했던 것 처럼 summarise() 함수가 아닌 mutate() 함수를 사용하여 전체 행에 그룹핑된 합계열을 생성한 후 원본 데이터를 합계열로 나누어 비율을 구할 수 있다.

# 취업자수의 분기별 비율  
employees.tsibble%>%  
 index\_by(yearqtr = ~ yearquarter(.)) %>%  
 mutate(sum.qtrly = sum(total)) %>%   
 mutate(rate.qtrly = total/sum.qtrly) %>%  
 head(15)

# A tsibble: 15 x 6 [1D]  
# Groups: @ yearqtr [5]  
 time total employees.edu yearqtr sum.qtrly rate.qtrly  
 <date <int <int <qtr <int <dbl  
 1 2013-01-01 24287 1710 2013 Q1 73238 0.332  
 2 2013-02-01 24215 1681 2013 Q1 73238 0.331  
 3 2013-03-01 24736 1716 2013 Q1 73238 0.338  
 4 2013-04-01 25322 1745 2013 Q2 76618 0.330  
 5 2013-05-01 25610 1774 2013 Q2 76618 0.334  
 6 2013-06-01 25686 1786 2013 Q2 76618 0.335  
 7 2013-07-01 25681 1813 2013 Q3 76895 0.334  
 8 2013-08-01 25513 1811 2013 Q3 76895 0.332  
 9 2013-09-01 25701 1794 2013 Q3 76895 0.334  
10 2013-10-01 25798 1790 2013 Q4 76841 0.336  
11 2013-11-01 25795 1793 2013 Q4 76841 0.336  
12 2013-12-01 25248 1779 2013 Q4 76841 0.329  
13 2014-01-01 25050 1748 2014 Q1 75629 0.331  
14 2014-02-01 25116 1786 2014 Q1 75629 0.332  
15 2014-03-01 25463 1814 2014 Q1 75629 0.337

# 취업자수의 월별 비율  
employees.tsibble%>%  
 index\_by(yearqtr = ~ year(.)) %>%  
 mutate(sum.qtrly = sum(total)) %>%   
 mutate(rate.qtrly = (total/sum.qtrly)\*100) %>%  
 head(15)

# A tsibble: 15 x 6 [1D]  
# Groups: @ yearqtr [2]  
 time total employees.edu yearqtr sum.qtrly rate.qtrly  
 <date <int <int <dbl <int <dbl  
 1 2013-01-01 24287 1710 2013 303592 8.00  
 2 2013-02-01 24215 1681 2013 303592 7.98  
 3 2013-03-01 24736 1716 2013 303592 8.15  
 4 2013-04-01 25322 1745 2013 303592 8.34  
 5 2013-05-01 25610 1774 2013 303592 8.44  
 6 2013-06-01 25686 1786 2013 303592 8.46  
 7 2013-07-01 25681 1813 2013 303592 8.46  
 8 2013-08-01 25513 1811 2013 303592 8.40  
 9 2013-09-01 25701 1794 2013 303592 8.47  
10 2013-10-01 25798 1790 2013 303592 8.50  
11 2013-11-01 25795 1793 2013 303592 8.50  
12 2013-12-01 25248 1779 2013 303592 8.32  
13 2014-01-01 25050 1748 2014 310766 8.06  
14 2014-02-01 25116 1786 2014 310766 8.08  
15 2014-03-01 25463 1814 2014 310766 8.19

# 월별, 분기별, 연별 누적합계

시계열 데이터는 시간 동안의 데이터 흐름을 나타내지만 경우에 따라서 계속된 누적 합계를 나타내야 할 경우가 있다. 예를 들어 주간별 매출액의 연도별 누적 매출액, 웹사이트의 누적 사용자와 같은 경우이다. 누적합은 앞서 설명한 그룹핑과 함께 쓰면 더 좋은 결과를 낼 수 있다.

* 데이터프레임 : cumsum(), group\_by(), collapse\_by() 함수

데이터프레임에 저장된 시계열 데이터에 대한 누적합을 구하기 위해서는 dplyr 패키지의 누적합을 산출하는 함수인 cumsum() 를 사용한다. 데이터를 그룹핑하지 않고 전체 데이터에 대한 누적합은 mutate() 함수를 사용하여 cumsum() 결과 열을 생성함으로써 구할 수 있다.

# 누적 취업자수 산출  
employees %>%   
 mutate(cumsum = cumsum(total)) %>%  
 select(time, total, cumsum) %>%  
 head(15)

time total cumsum  
1 2013-01-01 24287 24287  
2 2013-02-01 24215 48502  
3 2013-03-01 24736 73238  
4 2013-04-01 25322 98560  
5 2013-05-01 25610 124170  
6 2013-06-01 25686 149856  
7 2013-07-01 25681 175537  
8 2013-08-01 25513 201050  
9 2013-09-01 25701 226751  
10 2013-10-01 25798 252549  
11 2013-11-01 25795 278344  
12 2013-12-01 25248 303592  
13 2014-01-01 25050 328642  
14 2014-02-01 25116 353758  
15 2014-03-01 25463 379221

# 0-9세 코로나 확진자의 누적 plot  
covid19 %>%  
 mutate(cumsum = cumsum(`0-9세`)) %>%  
 select(date, `0-9세`, cumsum) %>%  
 ggplot(aes(date, cumsum)) +  
 geom\_line(aes(group = 1)) +  
 scale\_x\_date(date\_breaks = "1 month", date\_labels = "%y.%m") +  
 theme(axis.text.x=element\_text(angle=90,hjust=1))

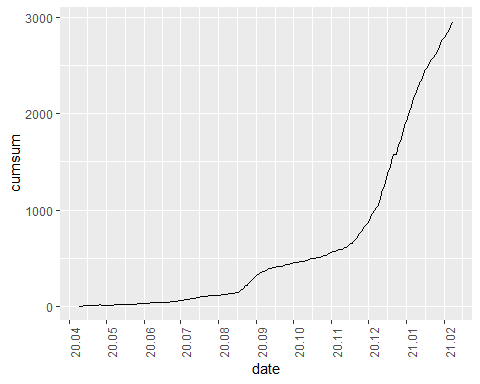


그림 4-18

전체 데이터에 대한 누적 합계가 아닌 연별, 월별, 주별 등의 주기별 누적합계를 구하려면 앞서 설명한group\_by()와 함께 사용하면 원하는 기간 동안의 누적 합계를 구할 수 있다.

employees %>%   
 group\_by(year(time)) %>%  
 mutate(cumsum.total = cumsum(total),   
 cumsum.edu = cumsum(employees.edu)) %>%  
 select(time, total, cumsum.total, employees.edu, cumsum.edu) %>%  
 head(15)

# A tibble: 15 x 6  
# Groups: year(time) [2]  
 `year(time)` time total cumsum.total employees.edu cumsum.edu  
 <dbl><date> <int> <int> <int> <int>  
 1 2013 2013-01-01 24287 24287 1710 1710  
 2 2013 2013-02-01 24215 48502 1681 3391  
 3 2013 2013-03-01 24736 73238 1716 5107  
 4 2013 2013-04-01 25322 98560 1745 6852  
 5 2013 2013-05-01 25610 124170 1774 8626  
 6 2013 2013-06-01 25686 149856 1786 10412  
 7 2013 2013-07-01 25681 175537 1813 12225  
 8 2013 2013-08-01 25513 201050 1811 14036  
 9 2013 2013-09-01 25701 226751 1794 15830  
10 2013 2013-10-01 25798 252549 1790 17620  
11 2013 2013-11-01 25795 278344 1793 19413  
12 2013 2013-12-01 25248 303592 1779 21192  
13 2014 2014-01-01 25050 25050 1748 1748  
14 2014 2014-02-01 25116 50166 1786 3534  
15 2014 2014-03-01 25463 75629 1814 5348

* tsibble : index\_by() 함수

tsibble 객체에 대한 누적 합계는 앞서 설명한 그룹핑 방법과 유사한데, index\_by()를 사용하여 grouping 하고 합계를 산출하는 방법의 sum()를 cumsum()으로, summarise()를 mutate() 로 바꾸어 주면 간단히 끝난다. mutate()를 쓰는 이유는 누적 합계는 모든 관측치에 필요한 열이기 때문이다.

employees.tsibble%>%  
 index\_by(yearqtr = ~ yearquarter(.)) %>%  
 mutate(cumsum.qtrly = cumsum(total)) %>%   
 select(yearqtr, cumsum.qtrly) %>%  
 head(10)

# A tsibble: 10 x 3 [1D]  
# Groups: @ yearqtr [4]  
 yearqtr cumsum.qtrly time   
 <qtr <int <date   
 1 2013 Q1 24287 2013-01-01  
 2 2013 Q1 48502 2013-02-01  
 3 2013 Q1 73238 2013-03-01  
 4 2013 Q2 25322 2013-04-01  
 5 2013 Q2 50932 2013-05-01  
 6 2013 Q2 76618 2013-06-01  
 7 2013 Q3 25681 2013-07-01  
 8 2013 Q3 51194 2013-08-01  
 9 2013 Q3 76895 2013-09-01  
10 2013 Q4 25798 2013-10-01

covid19.tsibble[, c(1,3)]%>%  
 index\_by(yearweek = ~ yearweek(.)) %>%  
 mutate(cumsum.weekly = cumsum(`0-9세`)) %>%   
 head(10)

# A tsibble: 10 x 4 [1D]  
# Groups: @ yearweek [2]  
 date `0-9세` yearweek cumsum.weekly  
 <date <dbl <week <dbl  
 1 2020-04-09 2 2020 W15 2  
 2 2020-04-10 1 2020 W15 3  
 3 2020-04-11 1 2020 W15 4  
 4 2020-04-12 0 2020 W15 4  
 5 2020-04-13 2 2020 W16 2  
 6 2020-04-14 0 2020 W16 2  
 7 2020-04-15 0 2020 W16 2  
 8 2020-04-16 3 2020 W16 5  
 9 2020-04-17 1 2020 W16 6  
10 2020-04-18 2 2020 W16 8

* xts

xts 패키지에서는 누적합을 산출하는 함수를 제공하지 않는다.

그렇기 때문에 누적합을 구하기 위해서는 몇 단계를 거쳐야 한다. 먼저 split() 함수를 사용하여 누적합을 적용할 시간 단위로 데이터를 분리해준다. 분리한 결과는 list의 형태로 반환되는데 이 결과를 lapply()함수를 이용하여 cumsum() 함수를 각각의 list에 적용시켜 준다. 이 결과는 list의 형태이기 때문에 do.call() 함수를 이용하여 하나의 xts 형태로 다시 묶어 주면 원하는 값을 얻을 수 있다.

이 과정에서 list를 다루는 함수인 lapply()와 do.call() 함수가 사용되었다. 이 두 함수 모두 매개변수로 받은 함수를 각각의 list 요소에 적용하는데 차이점은 lapply()의 경우 각각의 list의 모든 행에 대해 매개변수 함수를 적용하는데 반해 do.call()은 list의 요소에 대해 함수를 적용한다는 점에서 차이가 있다.

아래의 예에서 lapply()로 전달된 cumsum() 는 split()로 분리된 12개의 요소의 각 행(총 1,424개)에 적용되고 do.call()로 전달된 rbind() 함수는 split()로 분리된 12개의 요소에 적용되기 때문에 단순히 12번 호출된다.

do.call(rbind, lapply(split(employees.xts, f = 'year'), cumsum)) %>%  
 head(15)

total employees.edu 증감 증감율  
2013-01-01 24287 1710 NA NA  
2013-02-01 48502 3391 NA NA  
2013-03-01 73238 5107 NA NA  
2013-04-01 98560 6852 NA NA  
2013-05-01 124170 8626 NA NA  
2013-06-01 149856 10412 NA NA  
2013-07-01 175537 12225 NA NA  
2013-08-01 201050 14036 NA NA  
2013-09-01 226751 15830 NA NA  
2013-10-01 252549 17620 NA NA  
2013-11-01 278344 19413 NA NA  
2013-12-01 303592 21192 NA NA  
2014-01-01 25050 1748 -198 -0.8  
2014-02-01 50166 3534 -132 -0.5  
2014-03-01 75629 5348 215 0.9

# 동월별, 동분기별, 동년별 Plot

앞선 plot에서처럼 시계열 데이터는 선 plot으로 그려지는 경우가 많다. 하지만 동년, 동월, 동분기 데이터를 전체적으로 확인해야 할 경우가 있다. 이 plot은 모든 시계열 객체에서 그릴 수는 없다. ggplot()으로 데이터프레임이 가장 구현하기가 쉬운데 일부 시계열 패키지에서 제공하는 계절성 plot 함수를 사용할 수도 있다.

이 절에서는 주로 상자 plot을 활용하여 정보를 표현하도록 하겠다.

* 데이터프레임 : group\_by()

데이터프레임을 사용한 동월, 동분기별, 동년별 plot은 우선 데이터를 적절히 그룹핑 해야한다.

앞의 예제에서처럼 월별 데이터로 그룹핑하기를 원하면 더 큰 시간인 년도도 같이 그룹핑해서 전체적인 시간적 흐름을 놓치지 않도록 유지하였다. 하지만 매 1월 판매량 등과 같이 시간적 흐름보다는 시간적 특성만을 원할 경우는 그룹핑 할 때 원하는 해당 정보에 대해서만 그룹핑 함으로써 원하는 정보를 얻을 수 있다.

employees %>%  
 mutate(year = lubridate::year(employees$time)) %>%  
 ggplot(aes(as.factor(year), total)) +   
 geom\_boxplot() +   
 geom\_jitter(alpha = 0.2) +   
 labs(title = '동년별 취업자 분포', x = '연도', y = '취업자수')

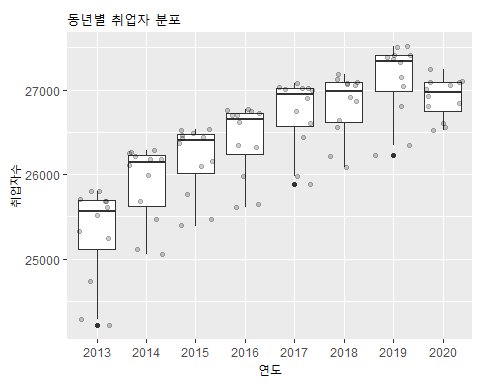


그림 4-19

employees %>%  
 mutate(month = lubridate::month(employees$time)) %>%  
 ggplot(aes(as.factor(month), total)) +   
 geom\_boxplot() +   
 geom\_jitter(alpha = 0.2) +   
 labs(title = '동월별 취업자 분포', x = '월', y = '취업자수')

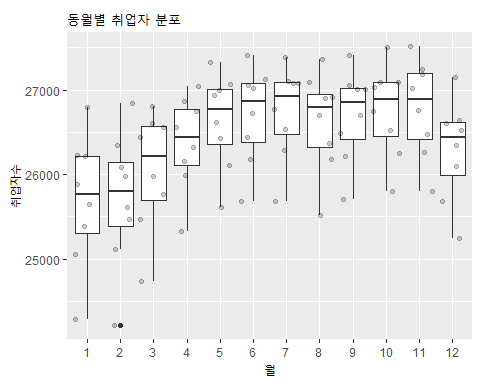


그림 4-20

employees %>%  
 mutate(quarter = lubridate::quarter(employees$time)) %>%  
 ggplot(aes(as.factor(quarter), total)) +   
 geom\_boxplot() +   
 geom\_jitter(alpha = 0.2) +   
 labs(title = '동분기별 취업자 분포', x = '분기', y = '취업자수')

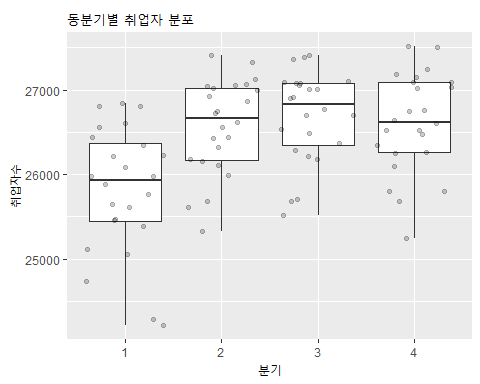


그림 4-21

covid19 %>%  
 mutate(month = lubridate::month(covid19$date)) %>%  
 ggplot(aes(as.factor(month), `0-9세`)) +   
 geom\_boxplot() +   
 geom\_jitter(alpha = 0.2) +   
 labs(title = '동월별 확진자 분포', x = '연도', y = '확진자수')

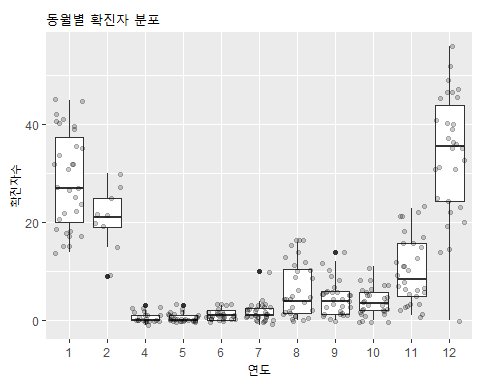


그림 4-22

covid19 %>%  
 mutate(wday = lubridate::wday(covid19$date, label = TRUE)) %>%  
 ggplot(aes(as.factor(wday), `50-59세`)) +   
 geom\_boxplot() +   
 geom\_jitter(alpha = 0.2) +   
 labs(title = '동요일별 확진자 분포', x = '연도', y = '확진자수')

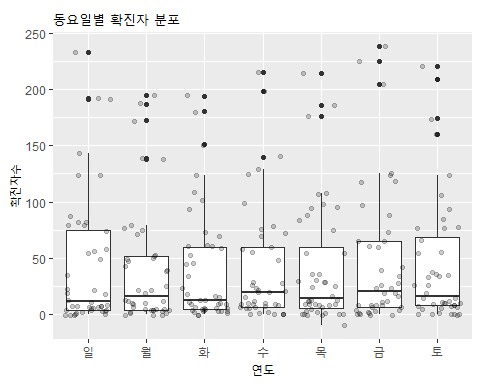


그림 4-23

* 데이터프레임 : plot\_seasonal\_diagnostics()

데이터프레임에서 계절성을 plotting 하기 위해 앞의 예제에서는 계절 주기로 그룹핑 한 컬럼을 활용하였다. 하지만 계절성을 알고 있는 경우는 쉽게 구할 수 있지만 계절성을 모르는 경우는 여러 plot을 확인해야 한다.

이런 경우 사용할 수 있는 함수가 timetk 패키지의 plot\_seasonal\_diagnostics() 이다. 이 함수는 주어진 시계열 데이터에 가능한 모든 동월, 동분기, 동요일 등의 plot을 동시에 생성해준다. plot\_ly와 ggplot2를 기본으로 작성된 plot이기 때문에 가장 큰 장점이 반응형(interactive) plot이 생성된다.

employees %>%  
 timetk::plot\_seasonal\_diagnostics(.date\_var = time, .value = total, .title = '전체 취업자의 주기별 Plot')

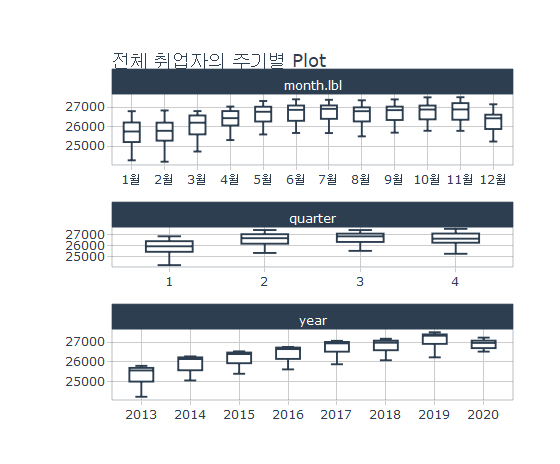


그림 4-24

covid19 %>%   
+ timetk::plot\_seasonal\_diagnostics(.date\_var = date, .value = `0-9세`, .title = '코로나 확진자(0-9세)의 주기별 Plot')

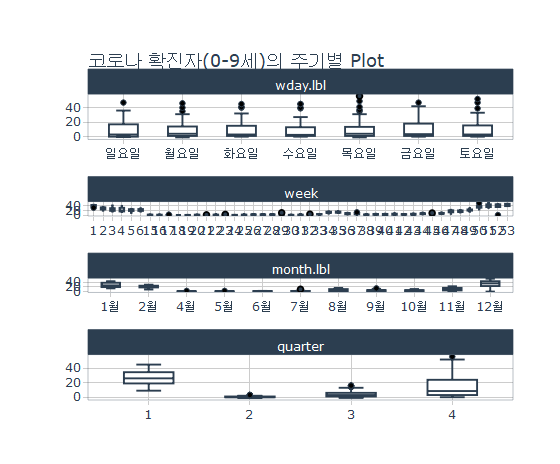


그림 4-25

1. <https://cran.r-project.org/web/packages/lubridate/vignettes/lubridate.html> [↑](#footnote-ref-1)
2. <https://rdrr.io/cran/lubridate/man/round_date.html> [↑](#footnote-ref-2)
3. <https://cran.r-project.org/web/packages/tibbletime/vignettes/TT-04-use-with-dplyr.html> [↑](#footnote-ref-3)